Tencent Real-Time Communication

Integration (No UI)

Product Documentation

Contents

Integration (No UI)
SDK Download
SDK Download

Last updated：2024-04-03 20:00:23

Tencent Real-Time Communication (TRTC) is a low-latency, high-quality audio and video communication service provided by Tencent Cloud, dedicated to providing stable, reliable, and low-cost audio and video transmission capabilities for Tencent Cloud customers. This service consists of a global network of audio and video transmission and a set of terminal SDKs. You can download the TRTC SDK covering currently mainstream client platforms and popular frameworks on this page.

TRTC SDK

All-In-One SDK

In addition to the Tencent Real-Time Communication (TRTC) SDK, Tencent Cloud also offers quick integration solutions for various audio and video scenarios, including live streaming, short video, and player SDKs.
The All-In-One SDK is designed to address dependency conflicts that may arise when using multiple SDKs simultaneously. It includes all the features of the TRTC SDK, live streaming (MLVB) SDK, short video (UGSV) SDK, and player SDK in one package. If you require multiple SDKs, you can download this version.

Release Notes (App)

Last updated：2024-06-13 17:08:37

This page includes the version history of TRTC SDK and All-In-One SDK. Please choose the ideal SDK according to your function needs.

TRTC SDK 11.9 Released on June 12, 2024

New features

All platforms: Optimized AI denoise effect, significantly improving performance in internet bars, meetings, and other scenarios.
All platforms: Supports the APE BGM file.
All platforms: Improved the accuracy of existing operational events, enhancing troubleshooting efficiency for silent or low-volume abnormal scenarios.
iOS & Android: Support for high-definition screenshot.
iOS: TXDeviceManager adds a new API setCameraCapturerParam (API url) for setting capture parameters.

Bug fixing

Android: Resolved the occasional electronic noise issue during microphone capture.
All platforms: Fixed stability issues caused by specific BGM formats and illegal data.
Android: Resolved the occasional low volume issue in speaker mode.
Windows: Fixed the issue of occasionally retrieving the incorrect current camera.

TRTC SDK 11.8 Released on May 10, 2024

New features

iOS: Supports picture-in-picture feature.

Improvements

Android: Optimized audio compatibility issues and added support for more comprehensive device selection routes.
Windows: Optimized AI noise reduction performance.
Windows: Improved upstream video quality in single anchor scenarios.
All platforms: Optimized BGM error message prompts.
Windows: Optimizing the display scaling effect.

Bug fixing

Android: Optimized the issue of low speaker volume.
Mac: Fixed the issue of no sound when killing the process with Loopback open.
Mac: Fixed the issue of no sound when connecting to HDMI or DisplayPort.

Windows: Fixed the issue where switching between different audio quality types caused AGC to fail.
Android: Screen sharing adapted for high TargetVersion.

Interface behavior adjustment

All platforms: The C++ interface supports the onMixedAllAudioFrame callback.

TRTC SDK 11.7 Released on Mar 4, 2024

New features

Full platform: Add new Warning codes for camera capture.
Full platform: Adjust the gravity sensing API, see setGravitySensorAdaptiveMode .
Full platform: Support callback for local voice volume information, see enableAudioVolumeEvaluation.

Improvements

Full platform: Optimize room entry process to significantly reduce the time taken for secondary room entry.
Full platform: Dashboard monitors the maximum number of end-to-end call quality for a single user, increasing from 16 to 50 channels.
Android: Optimize the parameters and calling order of the system capture audio API to reduce the probability of capturing only noise and improve the sound capture effect.
iOS: Optimize the capture restart logic after system interruption to reduce the probability of capturing silence.
iOS: Optimize Unity 3D engine compatibility issues.
iOS: Improve the focusing effect of rear triple and dual camera capture, increasing the focusing speed.

Bug fixing

Android: Fix some echo leakage cases.
Android: Fix some Bluetooth interruptions causing external playback issues.

Interface behavior adjustment

Full platform: When the API switchRole is frequently called, only the last call result is returned in onSwitchRole.
Full platform: Add camera capture warning codes.

Warning Code	Description
WARNING_CAMERA_IS_OCCUPIED = 1114	The camera is occupied by other process
WARNING_CAMERA_DEVICE_ERROR = 1115	The camera device is error
WARNING_CAMERA_DISCONNECTED = 1116	The camera is disconnected
WARNING_CAMERA_START_FAILED = 1117	The camera is started failed
WARNING_CAMERA_SERVER_DIED = 1118	The camera sever is died

Full platform: Adjust the default frame rate for the setVideoMuteImage interface from 0fps to 5fps, and limit the maximum frame rate to 10fps.
iOS: Change the behavior of the API setSubStreamEncoderParam from setting screen sharing encoding parameters to setting auxiliary stream encoding parameters.
Windows: Change the default highlight color (highLightColor) in the selectScreenCaptureTarget from green to yellow.
Full platform: Adjust the gravity sensing API:

New API	(void)setGravitySensorAdaptiveMode:(TRTCGravitySensorAdaptiveMode)mode;	See Doc
Deprecated API	setVideoEncoderRotation	Deprecated
	setVideoEncoderMirror	Deprecated
	setGSensorMode	Deprecated

TRTC SDK 11.6 Released on Jan 15, 2024

New features

iOS：Added Picture-in-Picture support for `TXLivePlayer`.
Windows: Added support for capturing audio from specific application. For more details, see startSystemAudioLoopback.

Improvements

Android & iOS: Optimized the success rate of playing BGM with URL.
Windows: Optimized and adapted to Intel HEVC software decoder (Quick Sync Video).
All platforms: Optimized poor network performance of audio in low-bandwidth conditions.
All platforms: Optimized poor network performance of video in low-bandwidth conditions.
All platforms: Optimized poor network audio quality under high packet loss and high latency conditions.
All platforms: Optimized SDK overall stability.

Bug fixing

Android: Fixed occasional slow decoding of the first frame when `switchRoom` is called frequently..
Android: Fixed occasional incorrect screen scaling when trying to `startScreenCapture` again in a single session.Windows: Fixed the issue that some virtual cameras fail to capture when setting vertical resolution.

TRTC SDK 11.5 Released on Nov 27, 2023

Improvements

All platforms: Optimized the performance and stability of the video engine.

All platforms: Optimized the stability of the audio engine.
All platforms: Optimized the behavior strategy of some APIs, see the adjustment of interface behavior for details.
All platforms: Optimized the strategy and performance of the background music module (BGM module), reducing the occurrence of BGM playback exceptions.
Windows: Optimized HEVC hardware decoding compatibility with AMD and Nvidia graphics cards.
Windows: Optimized the overall performance of screen sharing, improved screen capture frame rate and stability.
Android: Optimized the playback effect of TRTC with VODPlayer.
iOS & Mac: Optimized the performance of pre-processing and rendering using Metal.

Adjustment of Interface Behavior

All platforms: When the video resolution is set to vertical 540P (expecting 540x960), the specific resolution for processing is adjusted from 544x960 to 536x960.
All platforms: The callback interval of BGM progress callback `onPlayProgress` is adjusted from 200ms to 300ms.
All platforms: The internal implementation of the BGM module is adjusted to a singleton, and the musicID needs to be globally unique in multiple instances. When developers use sub-instances to play BGM, please make sure that different instances use different musicIDs.
All platforms: Local recording event status codes are adjusted to be returned asynchronously. The default return is 0 after the related interface is called, and the specific status code is obtained through the corresponding event callback.
All platforms: Adjust the following status codes for the `onLocalRecordBegin` callback for starting recording events.

Event	Status code before v11.5	Status code in v11.5
Recording has started, stop previous recording first	-1	-6
Directory has no write permission, please check directory permissions	-2	-8
Incorrect file extension (e.g. unsupported recording format)	-3	-2

iOS & Android: Optimized the continuity of mobile screen sharing, retaining the last frame sent during sharing pause, with a frame rate of 1-2 fps.
iOS & Android: Adjusted the response behavior of gravity sensor, only responding to gravity sensor on or off.

TRTC SDK 11.4 Released on Sep 14, 2023

New features

All platforms: `TRTCLocalRecordingParams` adds a new parameter `max_duration_per_file` to control the duration of segmented recording. The path of the segmented file can be obtained through the `onLocalRecordFragment` callback.
Android & iOS: V2TXLivePusher adds a rendering mode setting API `setRenderFillMode` for local preview streaming.

Mac: Added `enableCrashMonitoring` to support capturing crash information and storing it locally. You need to add `TXCCrashMonitor.framework` to your project when using this feature.

Improvements

Cross-platform: Optimized performance in IPv6 network environments.
Cross-platform: Optimized lyric alignment in the online singing scenario.
Cross-platform: Optimized AI noise suppression algorithm for better experience.
Cross-platform: Optimized the smoothness of pure audio playback.
Cross-platform: Optimized the speed of `switchRoom`.
Android & iOS: Optimized and improve the live streaming playback startup speed.
Android & iOS: Optimized audio capture processing strategies to reduce the probability of no audio issue caused by abnormal capture devices.
Android: Optimized callback notifications when the microphone is muted by the system.
Android: Optimized the adaptation of gravity sensor for specific customized Android devices that were providing incorrect screen rotation angles.
Android: Optimized the rendering process to support closely following pinch-to-zoom, which means the view undergoes simultaneous scales and moves, closely following the movements of the fingers during pinch-to-zoom. Improving the user experience during floating window playback.
iOS: Optimized the audio capture strategy when app in the background state to reduce the probability of no audio issue caused by system interruption.
iOS: Optimized the speed of audio device restarts.

TRTC SDK 11.3 Released on July 7, 2023

New features

All Platforms: Added trapezoid correction for video (only supported by the Professional version) for manual correction of perspective distortion. See `setPerspectiveCorrectionPoints` for details.
All Platforms: Added audio spectrum callback, which can be used for sound wave animation or volume spectrum display. See `enableAudioVolumeEvaluation` and `TRTCVolumeInfo` for details.
All Platforms: Added a new reverb effect "Studio 2". See `TXVoiceReverbType` for details.
All Platforms: Added SEI parameter settings for mixed stream, used for transport SEI infomation when publishing stream to CDN. See `TRTCTranscodingConfig` for details.
Windows: Added real-time music scoring for Yinsuda Authorized Music, which can be used for real-time scoring of online singing. See `createSongScore` for details.
iOS & Android: Added support for .ogg format music files in `startPlayMusic`.
Flutter: Added `setSystemAudioLoopbackVolume`(iOS).

Improvements

All platforms: Optimized adaptive digital gain algorithm to improve listening experience.

All platforms: Optimized the loading speed of the first video frame after entering the room.
All platforms: Optimized weak network resistance for single user streaming to improve smoothness under network delay and jitter.
Android: Optimized audio capture and playback feature to avoid abnormal sound issues on some Android devices.
Android: Optimized video sub-stream hardware encoding performance, improving quality of screen sharing.
iOS: Optimized audio device restart strategy to reduce the occurrence of sound interruptions.
iOS & Android: Removed on-demand related interfaces from `TXLivePlayer`. For on-demand video playback, please use `TXVodPlayer`.

Bug fixing

Android: Fixed the issue where some locally recorded videos on Android 12 and above system versions cannot be played on Apple's Safari.

TRTC SDK 11.2 Released on June 5, 2023

TRTC

New features

Cross-platform: Supports seamless switching between instrumentals and original vocals of BGM in duet scenes. See `setMusicTrack` for details.
Android: To be compatible with the foreground service launch restrictions on Android 12 and above, a foreground service is initiated during screen capture. See `enableForegroundService` for details.
iOS: Supports use in the Xcode simulator running on Apple chip hardware.
Mac: `TRTCScreenSourceInfo` adds property value width and height.

Improvements

Cross-platform: Optimized sound quality in duet scenes, and reduced end-to-end latency.
Cross-platform: Optimized performance when turning on/off microphone, providing a smoother experience.
Cross-platform: Optimized audio experience under extremely bad networks.
Cross-platform: Optimized weak network experience when broadcast live stream only.
Cross-platform: Optimized the smoothness of switching high-quality and low-quality remote video streams.
Android & iOS: Optimized audio quality in music scenes.
Android & iOS: Optimized the experience with Bluetooth headphones.
Android: Optimized hardware decoder latency, improving the speed of rendering the first video image.
Android: Optimized the in-ear monitoring feature, enhancing the experience when switching on/off in-ear monitoring.
Android: Optimized the audio devices capture compatibility.
iOS: Optimized quality of video, enhancing image clarity.

Bug fixing

Windows: Fixed the occasional screen flickering issue during window sharing.
Mac: Fixed the occasional periodic blurring when using camera with Intel chip hardware encoder.

TRTC SDK 10.8 Released on October 27, 2022

TRTC

New features

All platforms: Added the DJ scratch effect and improved the karaoke experience. For details, see `TXAudioEffectManager.setMusicScratchSpeedRate`.

Improvements

Android: Sped up video decoding, which reduces the time to first frame to as short as 50 ms.
All platforms: Improved the accuracy of NTP time. For details, see `TXLiveBase.updateNetworkTime`.

Bug fixing

All platforms: Fixed the occasional issue where, when the streams of a room are mixed and pushed to another TRTC room that does not have upstream audio or video, playback fails and callbacks stop working.
All platforms: Fixed the occasional issue where, after an audience member changes their role upon room entry, they fail to publish audio and video due to network type change.
All platforms: Fixed the issue where, after a disconnection, audio quality cannot be changed during reconnection.
All platforms: Fixed the issue where, after a disconnection, there is sometimes no audio in the published stream during reconnection.
Android & iOS: Fixed the issue where `muteRemoteVideoStream` removes the last video frame.

TRTC SDK 10.7 Released on September 20, 2022

TRTC

New features

All platforms: You can now independently adjust the audio volume of each stream in On-Cloud MixTranscoding. For details, see `TRTCMixUser.soundLevel`.
All platforms: Added the `onRemoteAudioStatusUpdated` API, which is used to monitor the audio status of remote streams.

Improvements

All platforms: Upgraded the encoding engine, improving the video quality of screen sharing streams.

All platforms: Improved rate control for encoding under poor network conditions.

Bug fixing

iOS: Fixed the issue of low capturing volume on some iPad devices.
Android: Fixed the occasional issue where Bluetooth earphones are connected, but audio is played from the device’s speaker.
All platforms: Fixed the issue where the SDK occasionally crashes if a user enters and leaves the room repeatedly.

TRTC SDK 10.6 Released on September 9, 2022

TRTC

Improvements

All platforms: Sped up room entry in IPv6 networks.
All platforms: Improved the audio recovery performance and audio-to-video synchronization under bad network conditions, enhancing user experience.
All platforms: Improved the ability to maintain connection under poor network conditions, reducing disconnections.
All platforms: Fixed the issue where the volume is low in the music mode (which is specified when `startLocalAudio` is called).
macOS: Improved call experiences when Bluetooth earphones are used, reducing noise and delivering clearer audio.
Android: Supported stereo audio capturing for more devices.
Android: Fixed occasional echoes, improving call experience.

Bug fixing

Android & iOS: Fixed the issue of audio loss in the speech mode (which is specified when `startLocalAudio` is called).
macOS: Fixed the issue where echo cancellation occasionally fails to work after the mic is changed.

All-in-One SDK 10.5 Released on August 24, 2022

MLVB

Improvements

Android: Optimized memory management for video decoding, preventing the accumulation of memory leaks.
Windows: Optimized noise suppression for the built-in mic, especially in the music mode.
macOS: Fixed the frequent noise issue when the mic is turned on.

Bug fixing

All platforms: Fixed the issue where, in the LEB scenario with V2TXLivePlayer, SEI messages are sometimes not received.
All platforms: Fixed the issue where, in the LEB scenario with V2TXLivePlayer, audio is missing because the timestamp moves backward.

UGSV

Bug fixing

Android: Fixed the green screen issue in videos made from pictures on HarmonyOS.
Android: Fixed the issue of incorrect length for edited videos.
Android: Fixed failure to play or re-encode videos with multiple audio tracks.
Android: Fixed the issue where the “rock light” effect is applied only once during the selected time period.
Android & iOS: Fixed the issue where, after a video segment is deleted during shooting, the playback progress of the background music does not match.

TRTC

Improvements

All platforms: Optimized the QoS control policy, enhancing user experience under poor network conditions.
iOS & Android: Reduced end-to-end latency and improved in-ear monitoring experience.
Android: Optimized memory management for video decoding, preventing the accumulation of memory leaks.
Windows: Optimized noise suppression for the built-in mic, especially in the music mode.
macOS: Fixed the frequent noise issue when the mic is turned on.

Bug fixing

All platforms: Fixed occasional errors for the OnUserVideoAvailable and OnUserAudioAvailable callbacks when a user enters and leaves different rooms consecutively.

Player

Bug fixing

Android & iOS: Fixed failure to play URLs that do not include video formats at the end.

All-in-One SDK 10.4 Released on July 25, 2022

MLVB

New features

iOS & Android: V2TXLivePlayer can now freeze the last frame after playback ends.

Improvements

All platforms: Fixed the issue of high memory usage when TXLivePlayer\\V2TXLivePlayer plays FLV streams.

Android: Fixed occasional playback stutter with TXLivePlayer\\V2TXLivePlayer.

Android: Improved the compatibility of low-latency in-ear monitoring and dual-channel capturing.

Android: Optimized the policy for switching from hardware to software decoding.

iOS: Fixed the issue of low capturing volume on iPad.

Bug fixing

Android: Fixed the issue where TXLivePlayer\\V2TXLivePlayer occasionally switches to software decoding when playing streams.

UGSV

Improvements

Android: Added the `setBGMLoop` API for video editing.

Bug fixing

Android: Fixed the issue of `setWaterMark` not working.

Android: Fixed the issue where, when videos are previewed, TXVideoEditor fails to use the specified rendering mode.

TRTC

New features

iOS & Android: Added support for the RGBA32 format for custom capturing. For details, see `sendCustomVideoData` .

Windows & macOS: Added support for watermark preview after configuration. For details, see `setWaterMark` .

Improvements

Android: Improved the compatibility of low-latency in-ear monitoring and dual-channel capturing.

Android: Optimized the policy for switching from hardware to software decoding.

iOS: Fixed the issue of low capturing volume on iPad.

Bug fixing

All platforms: Fixed occasional room entry/exit callback errors.

Windows: Fixed the issue where, after the window shared changes, the new window is not displayed in full.

Player

Improvements

Android & iOS: Added support for adaptive bitrate HLS playback.

Bug fixing

Android: Fixed abnormal intervals for the `onNetStatus` callback and the progress callback.

Android: Fixed the null pointer exception caused by failure to call `setConfig` .

iOS: Fixed the stuttering issue when videos are replayed in some scenarios.

All-in-One SDK 10.3 Released on July 8, 2022

MLVB

New features:

All platforms: TXLivePlayer\\V2TXLivePlayer added support for HLS playback.

Improvement:

All platforms: Improved audio quality in the music mode.

All platforms: Optimized the SEI parsing logic. TXLivePlayer\\V2TXLivePlayer can parse some non-standard SEI messages now.

All platforms: Fixed the issue of audio and video being out of sync as a result of the timestamp moving backward when TXLivePlayer\\V2TXLivePlayer plays FLV or RTMP streams.

Bug fixing:

All platforms: Fixed the abnormal audio that occurs when TXLivePlayer\\V2TXLivePlayer plays some AAC-HEv2 streams in the LEB scenario.

All platforms: Fixed incorrect cache calculation with TXLivePlayer.

UGSV

Bug fixing:

Android: Fixed the issue of `setZoom` not working during video shooting.

Android: Fixed failure to shoot videos with Samsung Galaxy S22.

iOS: Fixed failure to trigger the callback for custom video pre-processing.

TRTC

New features:

Windows: Added support for recording live streaming sessions and audio/video calls to local storage. For details, see the description of `ITXLiteAVLocalRecord` .

Windows & macOS: Added a parameter to `startMicDeviceTest` , which allows you to specify whether to play the audio captured during mic testing. For details, see the description of `startMicDeviceTest` .

Improvement:

All platforms: Improved audio quality in the music mode.

Bug fixing:

All platforms: Fixed occasional errors for the user list callback.

Windows: Fixed the issue where videos sometimes freeze during playback.

Windows: Fixed occasional video playback failure.

Windows: Fixed the echo issue for custom audio capturing.

Player

New features:

iOS: Added support for picture-in-picture playback.

Bug fixing:

Android: Fixed the issue where, when hardware decoding is used and a video playlist is played in the background, the player fails to automatically play the next video when one video is finished.

Android & iOS: Fixed failure to trigger the callback when seeking is completed.

All-in-One SDK 10.2 Released on June 26, 2022

MLVB

New features:

All platforms: Added support for license authentication for playback with TXLivePlayer\\V2TXLivePlayer.

All platforms: Added support for HTTP header configuration for FLV playback with V2TXLivePlayer.

All platforms: Allowed changing audio encoding parameters dynamically when pushing RTMP streams with TXLivePusher\\V2TXLivePusher.

Improvement:

All platforms: Optimized the adaptive bitrate API of V2TXLivePlayer for LEB.

All platforms: Fixed the issue where V2TXLivePlayer takes a long time to reconnect in the LEB scenario.

All platforms: Fixed the issue of small local cache size when TXLivePlayer\\V2TXLivePlayer plays FLV or RTMP streams.

Android: Sped up the loading of the first frame for playback with TXLivePlayer\\V2TXLivePlayer.

iOS: Reduced the size of the iOS SDK package.

iOS: Packaged `TXLiveBase.h` into the MLVB SDK.

Bug fixing:

All platforms: Fixed the issue where the stutter rate limit configured for TXLivePlayer does not take effect.

All platforms: Fixed abnormal timing of the callback for the first audio/video frame when V2TXLivePusher pushes RTC streams.

Android: Fixed the black screen issue that occurs when TXLivePlayer\\V2TXLivePlayer stops and starts playback within a short period of time.

UGSV

New features:

Android: Added support for editing videos without audio tracks.

Improvement:

Android: Sped up the loading of the first frame for short video playback.

Bug fixing:

Android: Fixed the issue where the wrong section of video is cropped during video shooting.

Android: Fixed incorrect aspect ratio for H.265 videos decoded with hardware.

iOS: Fixed the issue of incorrect video clipping time.

iOS: Fixed occasional noise that occurs in videos shot with devices with OS later than iOS 14.

iOS: Fixed the issue where the SDK occasionally crashes when the user returns to the shooting view after finishing video shooting.

TRTC

New features:

All platforms: Launched a new API for stream mixing and relaying, which offers more powerful features and greater flexibility. For details, see the description of `startPublishMediaStream` .

All platforms: Added support for 3D spatial audio. For details, see the description of `enable3DSpatialAudioEffect` .

All platforms: Added support for voice activity detection. This feature works even when local audio is muted (`muteLoalAudio`) or the capturing volume is set to zero (`setAudioCaptureVolume`). It allows you to remind users when they are talking but have not turned their mics on. For details, see the description of `enableAudioVolumeEvaluation` .

All platforms: Added support for checking a user's permission when they switch roles. For details, see the description of `switchRole(TRTCRoleType role, const char* privateMapKey)` .

iOS & macOS: The C++ API for custom pre-processing supported using textures for video processing.

Improvement:

Android: Optimized in-ear monitoring, reducing latency.

Android: Optimized audio capturing, fixing the issue of noise on some devices.

iOS: Optimized the processing of upstream video data, reducing CPU and GPU usage.

Windows & macOS: Improved encoding for screen sharing. The height and width of the output video are no longer limited by the window size.
Windows: Reduced memory fragmentation and performance overhead.

Bug fixing:

All platforms: Fixed the issue where push occasionally fails after changing to a different type of network.
iOS: Fixed the issue of noise in recording files saved locally on some devices with iOS 14.

Player

Improvement:

Android & iOS: Optimized the callback of information including cached bytes and IP address during playback.

Bug fixing:

Android & iOS: Fixed failure to play H.265 videos when hardware decoding is used.
Android & iOS: Fixed HLS playback errors.
iOS: Fixed failure to get `supportedBitrates` in some scenarios.

Release Notes (Electron)

Last updated：2023-09-19 14:20:14

Version 11.0.501 Released on June 30, 2023

improvements:

Added two events onStartPublishing and onStopPublishing to the API documentation.
Improved the documentation of the onScreenCapturePaused interface fields in the API documentation.
Electron SDK now supports Linux (beta version).

Version 10.9.405 Released on April 17,2023

improvements:

New interface "setCameraCaptureParams" added, supports setting camera capture parameters, currently only available on Windows.
New interface "setVideoMuteImage" added, supports setting a placeholder image when camera is muted.
New interface "enableFollowingDefaultAudioDevice" added, supports speaker and microphone following current system device.
Interface "setMixTranscodingConfig" modified, supports setting input type, rendering mode, and placeholder image for each video stream.
Interface "getScreenCaptureSources" modified, adds a new field "isMainWindow" to the return value, currently only available on Windows.

Bug fixing:

Fix the issue of occasional application crashes caused by refreshing the page at the application layer.

Version 10.7.405 Released on February 27,2023

improvements:

New interfaces, updateLocalView and updateRemoteView, have been added to support modifying the position of video viewing and preview on the page.

Under Windows, the getScreenCaptureSources interface returns additional isMainScreen field information for screens and windows.

Bug fixing:

Fixed the issue where multiple people's screen sharing couldn't be rendered for viewing at the same time in a room.
Changed the video rendering DOM element scaling from transform scale to zoom to maintain backward compatibility.
Resolved the issue of occasional green screen frames appearing during local preview when the camera is frequently turned on and off.
The dynamic library for audio and video software decoding under Mac has been changed from a physical file to an external link file to resolve the signature exception that occurs when building the application package under Mac.
Resolved the issue where the fillMode parameter set by the setRemoteRenderParams interface does not take effect when called before startRemoteView

Version 10.3.406 Released on February 4,2023

improvements:

On Windows, the getScreenCaptureSources API now includes the isMainScreen field for screen and window information.

Bug fixing:

On Windows, if the microphone is in a mute state when starting up, the mute will be canceled automatically.
Fixed an issue where the highlight setting was not effective in the selectScreenCaptureTarget API.
Optimized the video rendering process on Windows.

Version 10.3.405 Released on December 12,2022

Bug fixing:

Fixed some usability issues that were discovered.

Version 10.7.404 Released on October 31,2022

improvements:

The setWaterMark interface now supports Windows systems, and on Mac, it also supports setting watermarks through image paths.

The startPublishCDNStream interface for pushing streams to non-
Tencent Cloud CDNs now has a new input parameter called "streamId," which supports setting the stream ID.

Version 10.6.404 Released on October 31,2022

improvements:

The setWaterMark interface now supports Windows systems, and on Mac, it also supports setting watermarks throu gh image paths.
The startPublishCDNStream interface for pushing streams to non-
Tencent Cloud CDNs now has a new input parameter called "streamId," which supports setting the stream ID.

Bug fixing:

Fixed the issue where calling setRemoteVideoStreamType() would cause the rendering of small video streams to bec ome stuck.
Fixed the occasional green screen frame issue during video rendering.
Fixed the problem of mirror image not being supported during camera detection on Mac, and solved the issue where the first shared window did not appear with a highlighted green frame on Mac.
Fixed the issue where the upstream frame rate was zero when sharing a window or screen on Mac, which caused th e remote user to not receive the onUserSubStreamAvailable event.

Version 10.3.404 Released on October 31,2022

Fixed the issue of video rendering getting stuck when calling setRemoteVideoStreamType() to switch to low stream.

Version 10.6.403 Released on September 09,2022

improvements:

Windows & Mac: Added local media recording interface, which supports saving local audio and video data to local fil es during live broadcast. Specific interfaces include:startLocalRecording、stopLocalRecording、onLocalRecordBegin、onLocalRecording、onLocalRecordComplete。

Function modification:

Windows&Mac:Abandoned the setRenderModeinterface and no longer support calling this interface to modify the def ault rendering method (WebGL or Canvas 2D) of the video. The SDK will automatically select the appropriate renderi ng method internally to improve video rendering performance.

Function improvement:

Optimization of video rendering performance.Upgrade the underlying library.
On Mac, support building applications with the ARM64 instruction set to leverage the advantages of the M1 chip and improve performance.

Version 10.3.402 Released on August 12,2022

Bug fixing:

Window & Mac: After calling the mixing interface, the mixing event returns an error of -3324 "user id invalid".

Version 10.3.401 Released on July 20, 2022

Improvements

Improved performance.
Updated the underlying library.

Version 9.3.201 Released on January 5, 2022

New features

Windows & macOS: Added the onSpeedTestResult callback, which returns the result of network speed testing.

Improvements

Windows & macOS: Improved the performance of the speed testing API startSpeedTest.
Windows & macOS: Added the `streamType` parameter to the muteLocalVideo API.
Windows & macOS: Added the `streamType` parameter to the muteRemoteVideoStream API.
Windows & macOS: Added the `params` parameter to the startScreenCapture API.

Bug fixing

macOS: Fixed the camera video capturing issue on macOS 12.
Windows & macOS: Optimized the QoS control policy under poor network conditions for smoother communication.
Windows: Improved the AGC algorithm, reducing cases of excessively low or high volume.
Windows: Fixed the issue of abnormal capturing frame rate for screen sharing.

Version 8.9.102 Released on August 11, 2021

New features

Windows & macOS: Added the new parameter `gatewayRtt` to the onStatistics callback.

Bug fixing

macOS: Fixed crash caused by logging on special devices.
macOS: Fixed the issue where, after `setAudioCaptureVolume(0)` is used to mute audio, the mic testing volume is 0.
Windows: Improved performance and fixed the issue of black screen after the camera is turned on.
Windows :Fixed the issue where the resolution is not restored after being automatically reduced during screen sharing.
Windows & macOS: Fixed other bugs.

Version 8.6.101 Released on May 28, 2021

New features

Windows & macOS: Added APIs for excluding windows from screen sharing: addExcludedShareWindow, removeExcludedShareWindow, removeAllExcludedShareWindow.
Windows & macOS: Added the `isMinimizeWindow` field to the data returned by the getScreenCaptureSources API.
Windows & macOS: Added support for passing constructor functions as parameters to APIs.

Bug fixing

Windows: Fixed the issue where paths containing Chinese characters are not supported for plugin loading.
Windows & macOS: Fixed the `webgl context lost` issue.
Windows & macOS: Fixed the issue where the videos of remote users freeze after the local user switches to the low-quality stream (dual-stream mode enabled).
Windows & macOS: Fixed the issue where, when a user enters the room and starts pulling streams, the images of remote users are blurred before they gradually become clear.

Version 8.4.1 Released on March 26, 2021

New features

macOS: Added support for capturing system audio via startSystemAudioLoopback, i.e., the system loopback feature that is enabled on Windows. The feature allows the SDK to capture system audio so that anchors can stream local audio or video files to other users.
macOS: Added the onSystemAudioLoopbackError callback, which updates you on the status of the system audio driver.

macOS: Added support for local preview for screen sharing. You can now display screen sharing preview in a small window.
All platforms: Added support for beauty filter plugins.

Improvements

All platforms: Improved audio quality in the music mode, which makes it more suitable for Clubhouse-like audio streaming scenarios.
All platforms: Improved the adaptability to poor network conditions. Smooth audio and video can be delivered even when the packet loss rate reaches 70%.
Windows: Improved audio quality in some streaming scenarios by significantly reducing audio damage.
Windows: Improved performance by 20%-30% in some scenarios.

Bug fixing

macOS: Fixed the issue where, after the screen sharing user switches to desktop sharing and then back to the sharing of a specific window on Mac mini (M1), remote users still see the user's desktop.
macOS: Fixed the issue where the shared content is not highlighted (on macOS 11.1 and 10.14.5, there isn't a green border around the shared content; on macOS 10.3.2, the green border is displayed, but the shared window flickers when maximized).
macOS: Fixed the issue where, when users on Mac mini (M1) get the screen sharing list, the SDK crashes because `sourceName` is null and "" is returned.
macOS: Fixed the issue on Mac mini (M1) where, when `getCurrentMicDevice` is called, the SDK crashes because `sourceName` is empty.
Windows: Fixed the issue where the SDK crashes when the desktop is shared on Windows Server 2019 Datacenter x64.
Windows: Fixed the issue where screen sharing sometimes ends unexpectedly when the target window is resized during screen sharing.
Windows: Fixed image capturing failure with some cameras.

Version 8.2.7 Released on January 6, 2021

New features

Windows & macOS: Added the switchRoom API.
Windows & macOS: Added the setLocalRenderParams API, which can be used to set rendering parameters for the local image (primary stream).
Windows & macOS: Added the setRemoteRenderParams API, which can be used to set rendering parameters for a remote image.
Windows & macOS: Added the startPlayMusic API, which is used to play background music.

Windows & macOS: Added the stopPlayMusic API, which is used to stop background music.
Windows & macOS: Added the pausePlayMusic API, which is used to pause background music.
Windows & macOS: Added the resumePlayMusic API, which is used to resume background music.
Windows & macOS: Added the getMusicDurationInMS API, which is used to get the total length of the background music file, in milliseconds.
Windows & macOS: Added the seekMusicToPosInTime API, which is used to set the playback progress of background music.
Windows & macOS: Added the setAllMusicVolume API, which is used to set the audio mixing volume of background music.
Windows & macOS: Added the setMusicPlayoutVolume API, which is used to set the local playback volume of background music.
Windows & macOS: Added the setMusicPublishVolume API, which is used to set the remote playback volume of background music.
Windows & macOS: Added the onSwitchRoom callback for room switching.
Windows & macOS: Added the setRemoteAudioVolume API, which is used to set the playback volume of a remote user.
Windows & macOS: Added the snapshotVideo API, which is used to take a video screenshot.
Windows & macOS: Added the onSnapshotComplete callback for the completion of a screenshot.

Improvements

Windows & macOS: Added support for string-type `strRoomId` for the `enterRoom` and `switchRoom` APIs.
Windows & macOS: Fixed other bugs.

Version 7.9.348 Released on November 12, 2020

Improvements

Windows: Added support for the use of paths containing Chinese characters to save recording files.
Windows: Added support for the anti-covering feature in the window capturing area.

Version 7.8.342 Released on October 10, 2020

New features

Windows & macOS: Added the onAudioDeviceCaptureVolumeChanged callback for volume change of the current audio capturing device.
Windows & macOS: Added the onAudioDevicePlayoutVolumeChanged callback for volume change of the current audio playback device.

Version 7.7.330 Released on September 11, 2020

New features

Windows & macOS: Added the setAudioQuality API, which is used to adjust audio quality.

Improvements

Windows: Fixed the issue of high CPU utilization when some low-end cameras are used.

Windows: Improved compatibility with multiple USB cameras and mics to make it easier to turn on such devices.

Windows: Optimized the selection policy of cameras and mics to avoid audio/video capturing errors caused by device change.

Windows & macOS: Fixed other bugs.

Version 7.6.300 Released on August 26, 2020

New features

Windows & macOS: Added APIs setCurrentMicDeviceMute, getCurrentMicDeviceMute, setCurrentSpeakerDeviceMute, and getCurrentSpeakerDeviceMute, which are used to control mics and speakers on PC.

Version 7.5.210 Released on August 11, 2020

Improvements

Windows & macOS: Fixed the issue where SDK callbacks are not in sequence.

Windows & macOS: Fixed the issue where switching rendering modes causes crashes.

Windows & macOS: Fixed the issue where rendering fails for certain resolutions.

Windows & macOS: Fixed other bugs.

Version 7.4.204 Released on July 01, 2020

Improvements

Windows: Optimized the acoustic echo cancellation (AEC) effect on Windows.

Windows: Improved compatibility with cameras on Windows.

Windows: Improved compatibility with audio devices (mics and speakers) on Windows.

Windows: Fixed the issue where the `UserID` returned by `onPlayAudioFrame` is incorrect on Windows.

Windows: Added support for system audio mixing on 64-bit Windows.

Version 7.2.174 Released on April 20, 2020

Improvements

macOS: Fixed occasional resolution inconsistency for local custom rendering on macOS.
Windows: Optimized the `getCurrentCameraDevice` logic on Windows to return the first device when the camera is not used.
Windows: Fixed the issue where the highlighted window is displayed as a gray screen during screen sharing.
Windows: Fixed the issue where the system occasionally freezes when users get screen share thumbnails on Windows 10.
Windows & macOS: Fixed the issue where the custom stream ID occasionally fails to take effect immediately after role switching.
Windows & macOS: Fixed the issue where encoding parameters for screen sharing do not take effect.
Windows: Fixed the issue where it takes a long time for a screen shared by a Windows user to be displayed to a WebRTC user.

Version 7.1.157 Released on April 02, 2020

New features

Supported screen sharing via the primary stream.

Improvements

Improved the usability of preset stream mixing templates.
Increased the success rate of stream mixing.
Optimized screen sharing on Windows.

Version 7.0.149 Released on March 19, 2020

New features

Added the trtc.d.ts file for TypeScript developers.

Release Notes (Web)

Last updated：2024-07-22 16:45:43

A version number is in the format of `major.minor.patch` .

`major` : Major version number. If there is major version refactoring, this field will be incremented. Generally, the APIs of different major versions are not compatible with each other.

`minor` : Minor version number. The APIs of different minor versions are compatible with each other. If there is a new or optimized API, this field will be incremented.

`patch` : Patch number. If there is a feature improvement or bug fix, this field will be incremented.

Note:

Please update to the latest version in a timely manner for service stability and better online support.

For version upgrade precautions, please refer to: Upgrade Guide.

Version 5.7.0 @2024.07.19

Feature

Support sending & receiving sei message in sub stream.

Improvement

The preview box will only display the current page when screen sharing using the captureElement parameter.

Avoid the chrome issue that unplug the headset will cause no audio issue in Android Webview.

Improve the detection logic of h264 capability.

Bug Fixed

Fix the issue that cannot see remote video in the browser that support h264 decoding, but not h264 encoding.

Version 5.6.3 @2024.06.28

Feature

Support listening to audio playback progress events in AudioMixer plugin.

Support set blur level in VirtualBackground plugin

Improvement

Improve the success rate of resuming normal calls after interrupting iOS calls.

Improve the success rate of audio autoplay in iOS.

Bug Fixed

Fixed the occasional reconnection issue in Chrome M91-.

Fixed the audio lag issue after remote user mute & unmute mircrophone.

Fixed the issue that remote publishing screen share event was mistakenly thrown in certain scenarios.

Version 5.6.2 @2024.06.07

Improvement

Reduce the video or audio lag for the audience.

Improve the success rate of reconnection.

Bug Fixed

Fixed the issue that capturing camera 1920 * 1280 failed in Mac Safari.

Fixed a occasional issue that cannot resume audio after autoplay failed in mobile chrome.

Fixed a occasional issue that cannot receive remote audio.

Fixed a occasional issue that muteRemoteAudio throws an abort error.

Version 5.6.1 @2024.05.23

Bug Fixed

Fixed a no audio issue when autoReceiveAudio disabled.

Version 5.6.0 @2024.05.17

Breaking Changed

Set the default value of `autoReceiveVideo` to false, refer to: Upgrade Guide.

Feature

Support trtc.sendCustomMessage.

Bug Fixed

Fixed the issue that startRemoteVideo got error in Chrome 123.

Fixed the issue that enter room failed in iOS 12.0.

Fixed the issue that the encoding mirror occasionally fails.

Fixed the issue that the AudioMixer plugin loop would not work.

Version 5.5.2 @2024.04.29

Improvement

Speed up loading by deploying model files yourself, refer to: Enable Visual Background.

Avoid the video flicker issue in iOS 17, refer to: webkit bug.

Bug Fixed

Fixed the issue that getting error occasionally when subscribing remote small video in Chrome M123+.

Fixed the issue that playing remote video failed occasionally.

Version 5.5.1 @2024.04.12

Improvement

Improve the success rate of reconnection.

Improve the success rate of recapturing.

Bug Fixed

Fix the issue that video was not playing in iOS 15.1.

Fix the issue that screen sharing was publishing on the main stream but not sub stream after calling `trtc.updateScreenShare({ publish: true })`

Fix the issue that the bitrate was not 128kbps when setting the audio profile to `TRTC.TYPE.AUDIO_PROFILE_HIGH` .

Fix the issue that the base64 imageUrl was unable to set to the WaterMark plugin.

Version 5.5.0 @2024.03.29

Feature

Add Beauty plugin.

Improvement

Optimize the capability of the AI Denoiser plugin in mobile phone.

Improve the success rate of media recapturing.

Bug Fixed

Fixed the issue that the preview of local video is black in iOS 16.

Fixed the issue that the user cannot hear the sound of remote audio occasionally in iOS 14.

Fixed the issue that startLocalAudio throws TypeError occasionally.

Version 5.4.3 @2024.03.15

Feature

Added support for passing MediaStreamTrack to the AudioMixer plugin.

Added support for calling trtc.getAudioTrack to obtain the screen sharing audio MediaStreamTrack.

Bug Fixed

Fixed the occasional issue where setting setRemoteAudioVolume to 0 did not works.

Fixed the issue that screen sharing audio was not published after calling updateScreenShare({ publish: true }).

Fixed the issue where virtual backgrounds could not be enabled in Safari.

Version 5.4.2 @2024.03.01

Bug Fixed

Fixed the issue that startRemoteVideo failed occasionally.

Fixed the issue that unpublish failed occasionally.

Fixed the issue that audio & video are not synchronized.

Fixed the issue that enterRoom failed when using nginx proxy.

Version 5.4.1 @2024.02.05

Improvement

Optimize the reconnection logic to improve the success rate of reconnection.

Bug Fixed

Fixed the issue that mirror is reset by calling updateLocalVideo.

Fixed the issue that CONNECTING state is not emitted by CONNECTION_STATE_CHANGED event.

Version 5.4.0 @2024.01.16

Feature

Support obtaining video snapshot. Refer to: getVideoSnapshot().

Support setting an image after mute video. Refer to: the mute parameter of startLocalVideo().

Support publishing the video streams only when the view is visible. Refer to: Multi-Person Video Calls.

Support screen sharing to capture a certain DOM element on the page. Refer to: startScreenShare().

Improvement

Optimize the logic of enterRoom and reduce the time cost of enterRoom.

Optimize the reconnection logic of laptop closed and reopened.

Bug Fixed

Fix the issue that publish failed on the version before Chrome 69.

Fix the issue that publish 1080p video failed on iOS 13 & 14.

Version 5.3.2 @2023.12.26

Feature

Support watermark plugin, refer to: Enable Watermark.

Support encoding mirror, refer to: startLocalVideo() 's mirror parameter.

Improvement

Improve audio and video encoding stability and quality.

Bug Fixed

Fix known issues with the CDNStreaming plugin.

Fix the issue where the volume value returned by the volume event is 0 after setting remoteAudioVolume to 0.

Fix the issue of occasional vocals dropping words when AI denoiser is enabled on some external microphones.

Version 5.3.1 @2023.12.08

Bug Fixed

Fix the abnormal issue with the mixing plugin.

Fix the inability to enter the room in earlier versions of Chrome 74.

Fix the issue that some audio interfaces do not perform as expected with AI noise reduction open.

Fix the issue that in multiple TRTC instance scenarios, the destruction of one instance prevents the others from receiving the DEVICE_CHANGED event.

Version 5.3.0 @2023.12.01

Feature

Support SEI messaging, which can be used for functions such as lyric synchronization and live quiz. For more details, please refer to sendSEIMessage.

Enable dynamic switching of large and small streams. For more details, please refer to the **option.small** parameter in updateLocalVideo.

Support mute pushing. For more details, please refer to the **mute** parameter in startLocalAudio().

Enable role switching with updated **privateMapKey**. For more details, please refer to the **privateMapKey** parameter in switchRole.

Add TRTC.EVENT.TRACK event.

Improvement

Improve the room entry process to reduce the time consuming.

Improve the encoding quality for high-resolution call scenarios and earlier devices of Android Chrome.

Improve device acquisition logic. With media access unauthorized, the SDK may temporarily request media permissions to ensure normal access to media devices, which will then be released.

Improve the parsing logic of the **url** parameter in the mixing plugin.

Improve the noise reduction effect of the AI noise reduction plugin.

Bug Fixed

Fix the issue of Android Chrome being unable to encode 120p.

Fix the issue where stopping screen sharing in non-pushing scenarios would cause the camera pushing to stop.

Fix the issue of the invail **CDN mixing plugin** parameter.

Version 5.2.1 @2023.11.08

Feature

Add 'captureVolume' parameter to API startLocalAudio & updateLocalAudio.

Support TRTC.EVENT.DEVICE_CHANGED event on mobile phone. You can implement the feature that auto switch to new microphone when a new headset is connected based on this event. Refer to Handle Device Change.

Bug Fixed

Fix the issue that local microphone is muted after switching microphone.

Fix the issue that the mediaType of TRTC.EVENT.PUBLISH_STATE_CHANGED is wrong after stopScreenShare.

Version 5.2.0 @2023.10.31

Feature

Add TRTC.EVENT.STATISTICS event.

Improvement

Improve the success rate of device capture.

Optimize the mirror processing logic of "Picture-in-Picture mode".

When the user's system rejects the browser permission, RtcError.handler() can be called to jump to the system authorization settings and guide the user to turn on the permission quickly. Refer to 5302.

Bug Fixed

Fixed a occasional issue that remote audio is not playing in low version of Chrome.

Version 5.1.3 @2023.09.11

Feature

trtc.setRemoteAudioVolume supports setting the volume higher than 100 to gain the remote playback volume.

Improvement

Avoided iOS 15.1 bug that caused page crash when switching camera.

Bug Fixed

Fix the issue that Firefox stopLocalVideo and then restart startLocalVideo failed.

Fix the issue that Firefox fails to capture camera with certain resolution, e.g. 640 * 360.

Fix the issue of remote video not playing occasionally.

Version 5.1.2 @2023.08.25

Improvement

Reduce time cost to enter a room.

Bug Fixed

Fix the issue that webpack package build of trtc.esm.js occasionally reporting errors.

Fix the issue that startLocalAudio passing in custom capture audioTrack does not work.

Version 5.1.1 @2023.08.18

Improvement

Default video profile changed to 480p_2 to reduce uplink bandwidth consumption.

Avoid the Chrome Bug that Android Chrome 115 occasionally fails to encode at resolutions lower than 360p.

Bug Fixed

Fix the issue that cannot enter room or startLocalVideo on Windows Chrome 57 and iOS Safari 12.

Fix the issue that the video bitrate is abnormal on Dashboard.

Version 5.1.0 @2023.08.11

Breaking Change

Restrict the roomId parameter of the trtc.enterRoom interface to be of number type and no longer support passing in string type. If you want to use a string roomId, please use the strRoomId parameter. When upgrading, please pay attention, see Upgrade Guide for details.

Feature.

Support background music plugin, refer to the tutorial: Implement Background Music.

Support AI noise reduction plugin, refer to tutorial: Implement AI Denoise.

Bug Fixed.

Fix the issue that setting screen sharing capture resolution does not work.

Fix the issue of occasional playback failure of remote screen sharing.

Version 5.0.3 @2023.07.31

Improvement

Optimize the reconnection mechanism to improve the stability of network connection.

Bug Fixed

Fix the issue that when calling trtc.stopRemoteVideo to stop the main video, the sub video is also stopped.

Version 5.0.2 @2023.07.21

Improvement

Optimize the performance and weak network resistance in multi-person call.

Optimize device capture logic to avoid the issue that some Lenovo devices cannot turn on the camera.

Optimize the capture parameters of screen sharing to avoid the issue of occasional frame dropping in long-time screen sharing.

Bug Fixed

Fix the issue that the small stream bit rate setting does not take effect.

Fix the issue that systemAudio parameter does not work.

Fix the issue that video tag is not destroyed after remote user screen sharing stopped.

Version 5.0.1 @2023.06.25

Feature

Support playing video with multiple view at the same time.

Bug Fixed

Fix the issue that screen sharing cannot be restarted after clicking the browser hover window to close screen sharing.

Version 5.0.0 @2023.05.26

The new architecture version of the TRTC Web SDK provides a flat interface that dramatically simplifies the API and reduces access costs. Features of the new API:

Flatter APIs that are easier to access.

Better stability.

Better performance.

Release Notes (Flutter)

Last updated：2024-06-24 15:22:32

Version 2.8.2 @ 2024.6.20

New features

Android&iOS: Newly added `onAudioRouteChanged` callback.

Added new audio route such as `TRTC_AUDIO_ROUTE_WIREDHEADSET` , see TRTCCloudDef for details.

Defect repair

Windows: Fixed onDeviceChange callback not triggered.

Version 2.8.1 @ 2024.6.14

New features

Android&iOS: Newly added setVoicePitch API

Added new reverb effects such as `Studio 2` , see TXVoiceReverbType for details.

Version 2.8.0 @ 2024.5.21

New features

Windows: Newly added getScreenCaptureSources and selectScreenCaptureTarget API to support Windows screen sharing.

Version 2.7.9 @ 2024.5.20

Dependency Update

Android SDK update to 11.8.0.14188

iOS SDK update to 11.8.15687

Version 2.7.8 @ Apr 18, 2024

Dependency Update

Windows SDK updated to 11.7.0.14863.

MacOS SDK updated to 11.7.15304.

Android SDK updated to 11.7.0.13910.

iOS SDK updated to 11.7.15343.

New features

Android&iOS: Added createSubCloud and destroySubCloud API.

Defect repair

Windows: Fixed a data parsing error in the onRecvCustomCmdMsg callback.

Version 2.7.7 @ Apr 03, 2024

Defect repair

Web: Fixed an issue where invoking switchRole was ineffective.

Version 2.7.6 @ Feb 29, 2024

Defect repair

Windows: Fixed a data upload issue in the Window library.

Version 2.7.5 @ Feb 29, 2024

New features

Windows: Updated startSystemAudioLoopback interface to support the collection of audio from specific applications.

Version 2.7.4 @ Feb 29, 2024

New features

Windows: Added startSystemAudioLoopback interface to support system audio capture, such as from third-party music players.

Version 2.7.3 @ Jan 15, 2024

Defect repair

Web: Fixed a compilation failure issue on the Web platform due to the introduction of dart:ffi.

Version 2.7.2 @ Jan 11, 2024

Dependency Update

Window: Upgraded Client SDK version to 11.4.0.

Version 2.7.1 @ Dec 28, 2023

Defect repair

macOS: Fixed an occasional crash issue that occurred during the TexTure rendering process.

New features

Android&iOS: Added setAudioFrameListener API.

Version 2.7.0 @ Dec 13, 2023

New features

Web: Upgraded Web TRTC SDK to the latest version (v5) to achieve a more stable feature.

Defect repair

Web: Fixed an issue where screen sharing from Android and iOS devices could not be viewed in the Web version.

Version 2.6.0 @ Nov 21, 2023

Defect repair

Web: Fixed an issue that occurred when calling the getCurrentDevice and getDevicesList APIs.

Version 2.5.9 @ Sep 28, 2023

New features

Android&iOS: Added startPublishMediaStream API.

Android&iOS: Added updatePublishMediaStream API.

Android&iOS: Added stopPublishMediaStream API.

Version 2.5.8 @ Sep 13, 2023

New features

Replaced document jump links with International Site.

Version 2.5.7 @ Sep 11, 2023

Dependency Update

Android SDK updated to 11.4.0.13189.

iOS SDK updated to 11.4.14445.

macOS SDK updated to 11.4.14445.

Version 2.5.6 @ Aug 09, 2023

Defect repair

Windows: Optimized Dart code style.

Version 2.5.5 @ Aug 02, 2023

Dependency Update

Android SDK updated to 11.3.0.13200.

iOS SDK updated to 11.3.14354.

macOS SDK updated to 11.3.14333.

Version 2.5.4 @ Jul 10, 2023

Dependency Update

Android SDK updated to 11.3.0.13176.
iOS SDK updated to 11.3.14333.

Version 2.5.3 @ Jun 27, 2023

Dependency Update

Android SDK updated to 11.2.13145.
iOS SDK updated to 11.2.14217.

Version 2.5.2 @ Jun 16, 2023

Defect repair

Windows: Fixed the issue where the startSpeedTest function returned excessively long Data Events and had No Response.
Web: Marked setAudioPlayoutVolume and getAudioPlayoutVolume as unavailable.

Version 2.5.1 @ Jun 02, 2023

New features

Windows&Mac&Web: Restored support for the Windows&Mac&Web platforms.
iOS: Added the setSystemAudioLoopbackVolume API, supporting System Volume adjustment during Screen Sharing.

Defect repair

iOS: Fixed occasional memory leak issues with TRTCCloudVideoView in specific scenarios.

Dependency Update

Windows&Mac: Upgraded Client SDK version to 11.1.0.

Version 2.5.0 @ May 04, 2023

Feature optimization

Temporarily removed support for Web, MacOS, and Windows platforms.

Version 2.4.6 @ May 04, 2023

Dependency Update

Android SDK updated to 11.1.0.13111.
iOS SDK updated to 11.1.14143.

Version 2.4.5 @ Mar 14, 2023

New features

Android: Added startSystemAudioLoopback feature.

Version 2.4.4 @ Mar 06, 2023

Feature optimization

Optimized part of the code.

Version 2.4.2 @ Jan 09, 2023

Dependency Update

Android SDK updated to 10.9.0.24004.

Version 2.4.2 @ Jan 09, 2023

Defect repair

Fixed the issue where snapshotVideo was empty on the iOS platform.

Version 2.4.1 @ Dec 01, 2022

Dependency Update

Android SDK updated to 10.8.0.13065.
iOS SDK updated to 10.8.12025.

Version 2.4.0 @ Oct 31, 2022

Feature optimization

Optimized part of the code.

Version 2.3.9 @ Oct 18, 2022

Dependency Update

Android SDK updated to 10.7.0.13053.

iOS SDK updated to 10.7.11936.

Version 2.3.8 @ Sep 20, 2022

Feature optimization

Optimization for Windows Platform, automatically add related DLL files.

Version 2.3.7 @ Sep 16, 2022

Defect repair

Repair the issue with "Can't use 'Function' as a name" on Web Platform.

Version 2.3.6 @ Sep 05, 2022

Feature optimization

Optimized part of the code.

Version 2.3.5 @ Aug 23, 2022

Dependency Update

Android SDK updated to 10.3.0.11196.

iOS SDK updated to 10.3.12231.

Version 2.3.4 @ Jul 21, 2022

New features

Updated Windows, MacOS, and Web platforms to support Pure Video Mode.
setMixTranscodingConfig: Only supports Mixed Video.

Version 2.3.2 @ Jul 14, 2022

Dependency Update

Android/iOS SDK updated to 10.3.

Version 2.3.1 @ Jun 23, 2022

Feature optimization

Optimized part of the code.

Version 2.3.0 @ Jun 20, 2022

Dependency Update

Android/iOS SDK updated to version 10.1 of LiteAVSDK_Professional.

New features

Support for Third-Party Beauty Filters.

Version 2.2.4 @ May 11, 2022

Feature optimization

Optimization of the setMixTranscodingConfig interface.

Version 2.2.3 @ May 07, 2022

Dependency Update

Android SDK set to 9.9.0.11820.
iOS SDK set to 9.5.11346.

Version 2.2.2 @ May 05, 2022

Feature optimization

Update Log Module.

Version 2.2.1 @ Apr 21, 2022

Feature optimization

PlatformView supports the 'onTap' event.

Version 2.2.0 @ Mar 30, 2022

Dependency Update

Update Android/iOS SDK to TXLiteAVSDK_Live.

Version 2.1.7 @ Mar 22, 2022

Defect repair

Repair the issue with iOS video rendering in version 2.1.6.

Version 2.1.6 @ Mar 17, 2022

Feature optimization

Optimize iOS texture.

Version 2.1.5 @ Mar 11, 2022

Feature optimization

Update remoteView to adjust the parameter order.

Version 2.1.4 @ Mar 10, 2022

Feature optimization

Update documents.

Version 2.1.3 @ Mar 10, 2022

Feature optimization

Update documents.

Version 2.1.2 @ Mar 07, 2022

Dependency Update

Android SDK updated to 9.5.11207.

iOS SDK updated to 9.5.11207.

Version 2.1.1 @ Feb 15, 2022

Defect repair

Repair the issue with incorrect callback data in onSpeedTest.

Version 2.1.0 @ Jan 25, 2022

Feature optimization

Optimize initialization timing.

Version 2.0.9 @ Jan 13, 2022

Defect repair

Repair the issue where the web folder was not found.

Dependency Update

Android & iOS SDK updated to 9.5.

Version 2.0.7 @ Jan 10, 2022

Feature optimization

Resolve warning.

Version 2.0.6 @ Jan 10, 2022

Feature optimization

Delete web folder.

Version 2.0.5 @ Jan 10, 2022

Feature optimization

Optimize document display.

Version 2.0.1 @ Jan 07, 2022

Feature optimization

Encapsulate video texture rendering into PlatformView.

Version 2.0.0 @ Jan 04, 2022

Feature optimization

Support smooth web.

Version 1.3.1 @ Jan 04, 2022

Feature optimization

Optimize part of the documents.

Version 1.3.0 @ Nov 22, 2021

Feature optimization

Android video rendering changed from SurfaceView to GLSurfaceView.

Version 1.2.9 @ Nov 15, 2021

Dependency Update

The underlying Android SDK version has been updated to 9.3.10765.

Version 1.2.8 @ Nov 15, 2021

Feature optimization

The underlying GLSurfaceView in Android has been replaced with TextureView, updateView only supports TextureView.

Version 1.2.7 @ Nov 05, 2021

Feature optimization

Screen sharing supports streams of specified sizes.

Version 1.2.6 @ Nov 01, 2021

Defect repair

Fixed the iOS video rendering issue caused by the previous version.

Version 1.2.5 @ Oct 27, 2021

New features

Android video rendering supports hybrid integration mode. The default mode is the virtual display mode. The view mode of TRTCCloudVideoView is passed to TRTCCloudDef.TRTC_VideoView_Mode_Hybrid.

Version 1.2.4 @ Sep 29, 2021

Feature optimization

Fluent's Windows platform supports texture rendering.
Fluent English documentation is available online.

Version 1.2.3 @ Sep 28, 2021

New features

Android supports updateLocalView and updateRemoteView interfaces.

Version 1.2.2 @ Sep 10, 2021

Defect repair

Fix Android texture rendering memory growth issue with multiple video switches.

Version 1.2.1 @ Sep 10, 2021

Feature optimization

Optimize certain features.

Version 1.2.0 @ Sep 10, 2021

New features

Specify the return value of the Beauty Filter, Equipment, and Sound management module.

Version 1.1.9 @ Aug 19, 2021

Defect repair

Fix issues such as Screenshot failure on iOS and MacOS.

Version 1.1.8 @ Aug 12, 2021

Feature optimization

Android Texture Rendering compatibility with meglhelper.

Version 1.1.7 @ Aug 10, 2021

Defect repair

Fix the issue of missing businessInfo field on Android.

Version 1.1.6 @ Aug 03, 2021

Feature optimization

Optimize certain features.

Version 1.1.5 @ Aug 03, 2021

Defect repair

Fix the crash issue caused by special string parameters when playing music in publication mode on iOS and MacOS.

Version 1.1.4 @ Aug 03, 2021

Defect repair

Fix the crash issue caused by special string parameters on iOS and MacOS.

Version 1.1.3 @ Jul 27, 2021

Defect repair

Fix the issue with no network quality data in the onspeedtest callback on iOS and MacOS.

Fix the issue with meglcore being null in Android texture rendering.

Version 1.1.2 @ Jul 23, 2021

Defect repair

Fixed an issue where iOS and macOS did not support auxiliary stream rendering.

Version 1.1.1 @ Jul 21, 2021

New features

New form of texture rendering.

Version 1.1.0 @ Jul 14, 2021

Defect repair

Fixed an issue where video could not be rendered after restarting remote view on Android.

Version 1.0.9 @ Jun 30, 2021

New features

Android and iOS support local recording startLocalRecording.

Version 1.0.8 @ Jun 28, 2021

New features

Support for Windows and macOS. Currently, only audio-related interfaces are supported, video rendering is not supported.

Version 1.0.5 @ Jun 09, 2021

Defect repair

Fixed platform exception error appearing after Android video view termination.

Version 1.0.4 @ Jun 02, 2021

New features

iOS adds updateLocalView and updateRemoteView interfaces.

Version 1.0.3 @ Jun 01, 2021

Defect repair

Fixed the issue where setting the audio routing was ineffective after closing the microphone on Android.

Version 1.0.2 @ May 27, 2021

Feature optimization

Modify document comments.

Version 1.0.1 @ Apr 28, 2021

Defect repair

Fixed the problem that Android could not enter the room when room ID exceeded 2147483647. Supported value range: 1 - 4294967294.

Version 1.0.0 @ Apr 23, 2021

Feature optimization

Upgrade to Flutter 2.0, support Zero Security.

API Examples
iOS

Last updated：2024-07-05 19:30:25

This document describes how to quickly run the demo for the TRTC iOS SDK.

Prerequisites

Xcode 11.0 or later
A valid developer signature for your project
Qt Creator 4.13.3 (macOS) or later

Steps To Run Demo

Step 1. Download the Demo

Download the iOS sample demo code on github, or run the following command on the terminal:

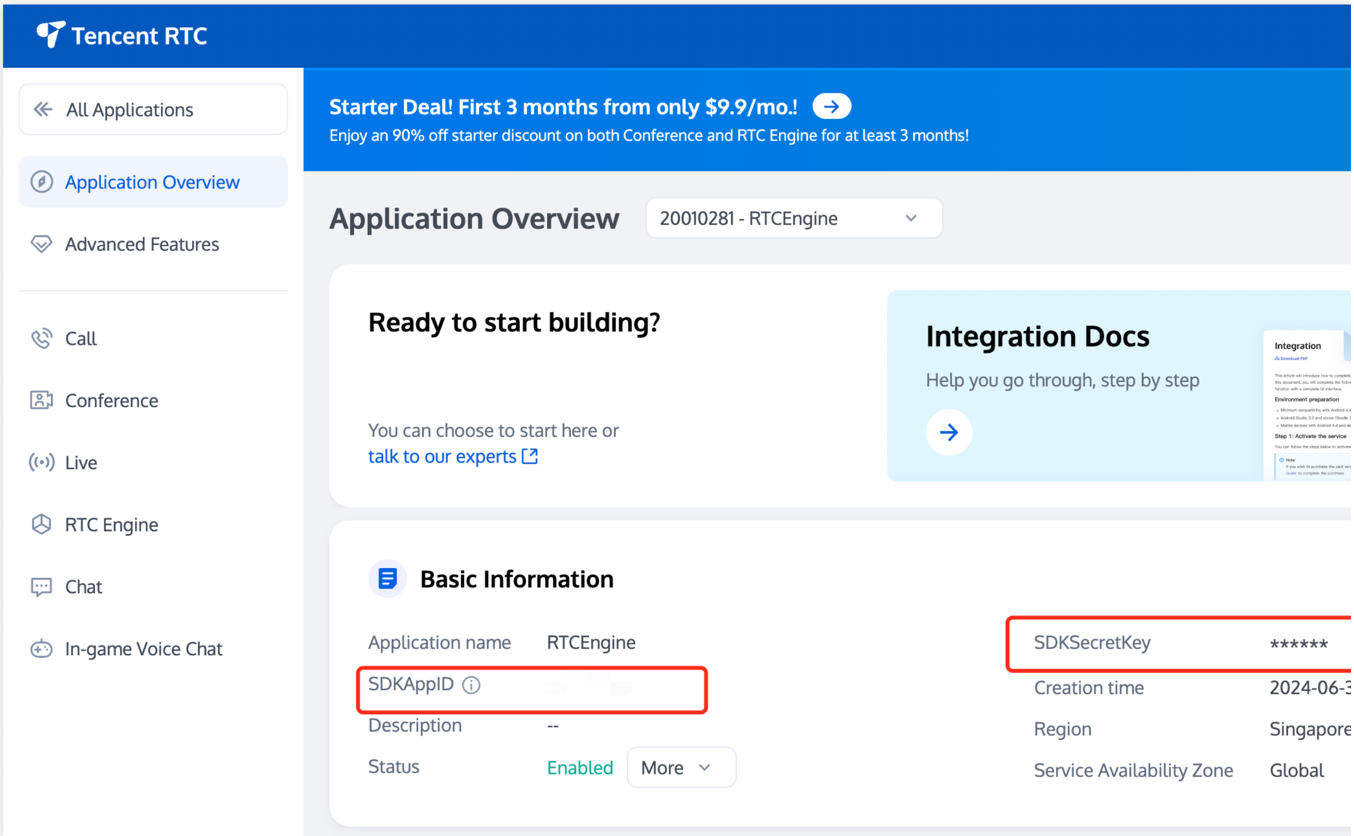
```
git clone https://github.com/Tencent-RTC/TRTC_iOS.git
```

Run `pod init` in a terminal window after entering the directory of your project, and take no notice of other steps in iOS SDK Importing.

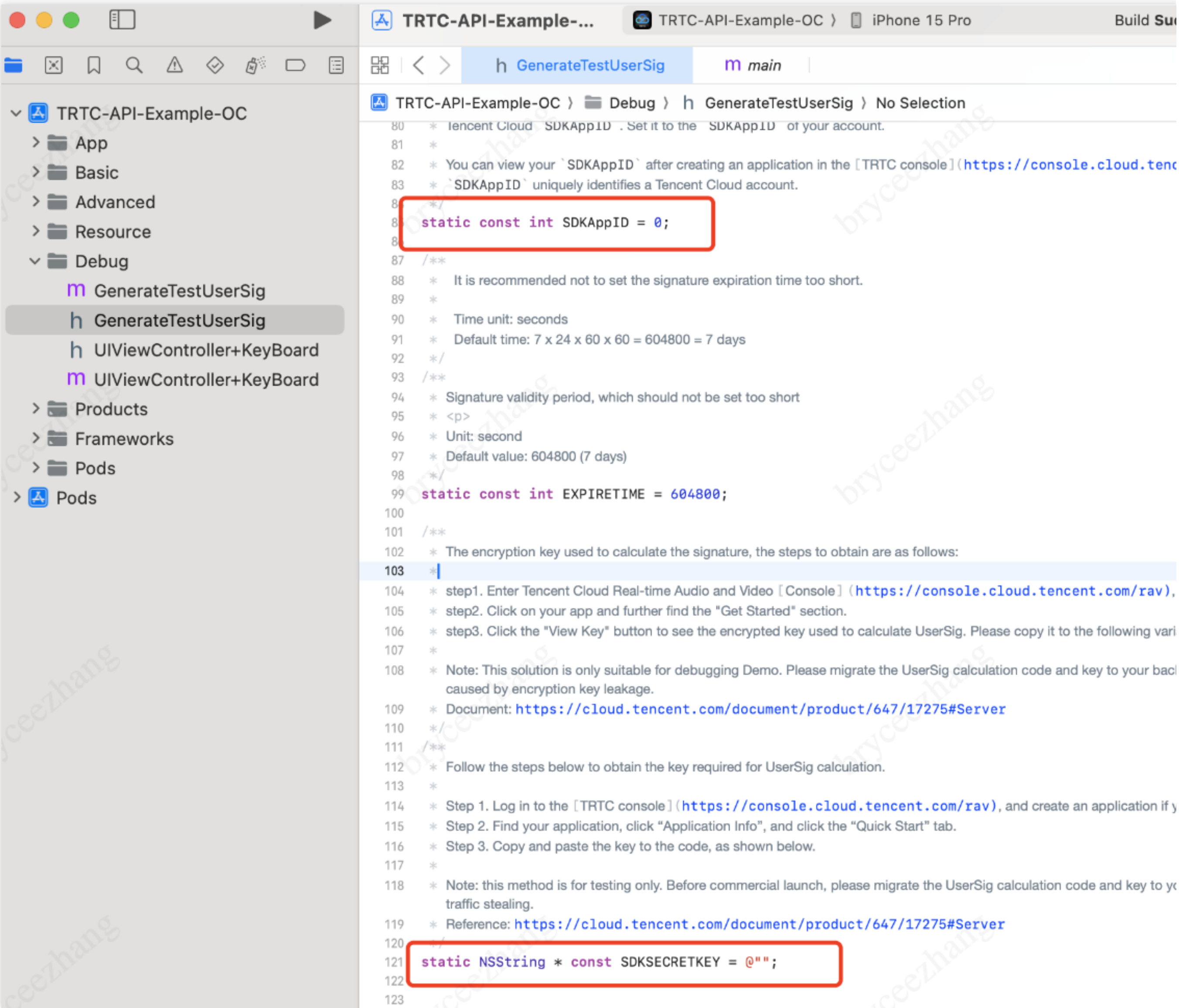
Step 2. Configure the Demo

1. Log in to the TRTC Console and click **Create Application**. If you have already done so, you may skip this step.

2. And then, your own `SDKAppID` and `SDKSecretKey` of your created application can be obtained in the **Basic Information** section.



3. Replace the values of `SDKAPPID` and `SDKSECRETKEY` with the information you obtained in **Step 2** in `GenerateTestUserSig.h` file or `GenerateTestUserSig.swift` file under `TRTC-API-Example-XX/Debug` directory.



Note：

In the demo above, we used **SDKSecretKey** to generate **UserSig** locally in order to help you go through the demo easier. However, in the production environment, you are not supposed to generate userSig in this way, which may lead to **SDKSecretKey** leakage, thereby creating a chance for attackers to steal your TRTC traffic. **The correct way to generate UserSig is to integrate Server-Side Generation of UserSig on your server.** When an user enters the room:

Send a http request to your server.

Generate a UserSig on your server.

Return it to the user to enter the room.

When you deploy your page to a production environment, you need to have your page accessed through the HTTPS(e.g. `https://domain/xxx`). For the reason, please refer to the document Page Access Protocol Restriction Description.

Step 3. Run the Demo

Open the `TRTC-API-Example-OC.xcworkspace` project in the source code directory with Xcode (11.0 or later) and compile and run the TRTC-API-Example project.

Step 4. Experience the Demo

You can choose the functions you are interested in to experience.

FAQs

If you encounter any problems with access and use, please refer to FAQs.

If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

Mac

Last updated：2024-07-09 15:09:58

This document describes how to quickly run the demo for the TRTC macOS SDK.

Prerequisites

Xcode 11.0 or later.

A valid developer signature for your project.

Qt Creator 4.13.3 (macOS) or later.

Steps To Run Demo

Step 1. Download the Demo

Download the iOS sample demo code on github, or run the following command on the terminal:

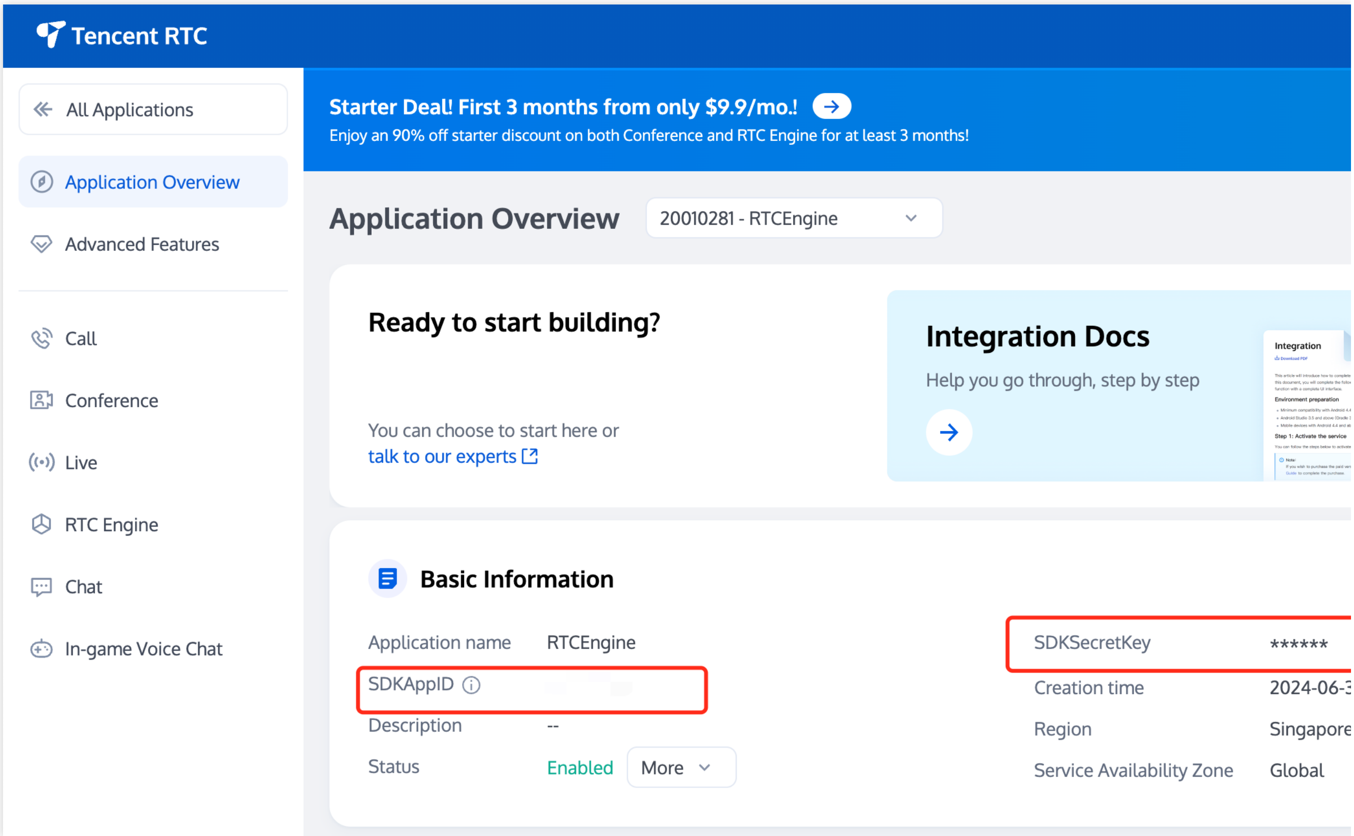

```
git clone https://github.com/Tencent-RTC/TRTC_Mac.git
```

Run `pod init` in a terminal window after entering the directory of your project, and take no notice of other steps in iOS SDK Importing.

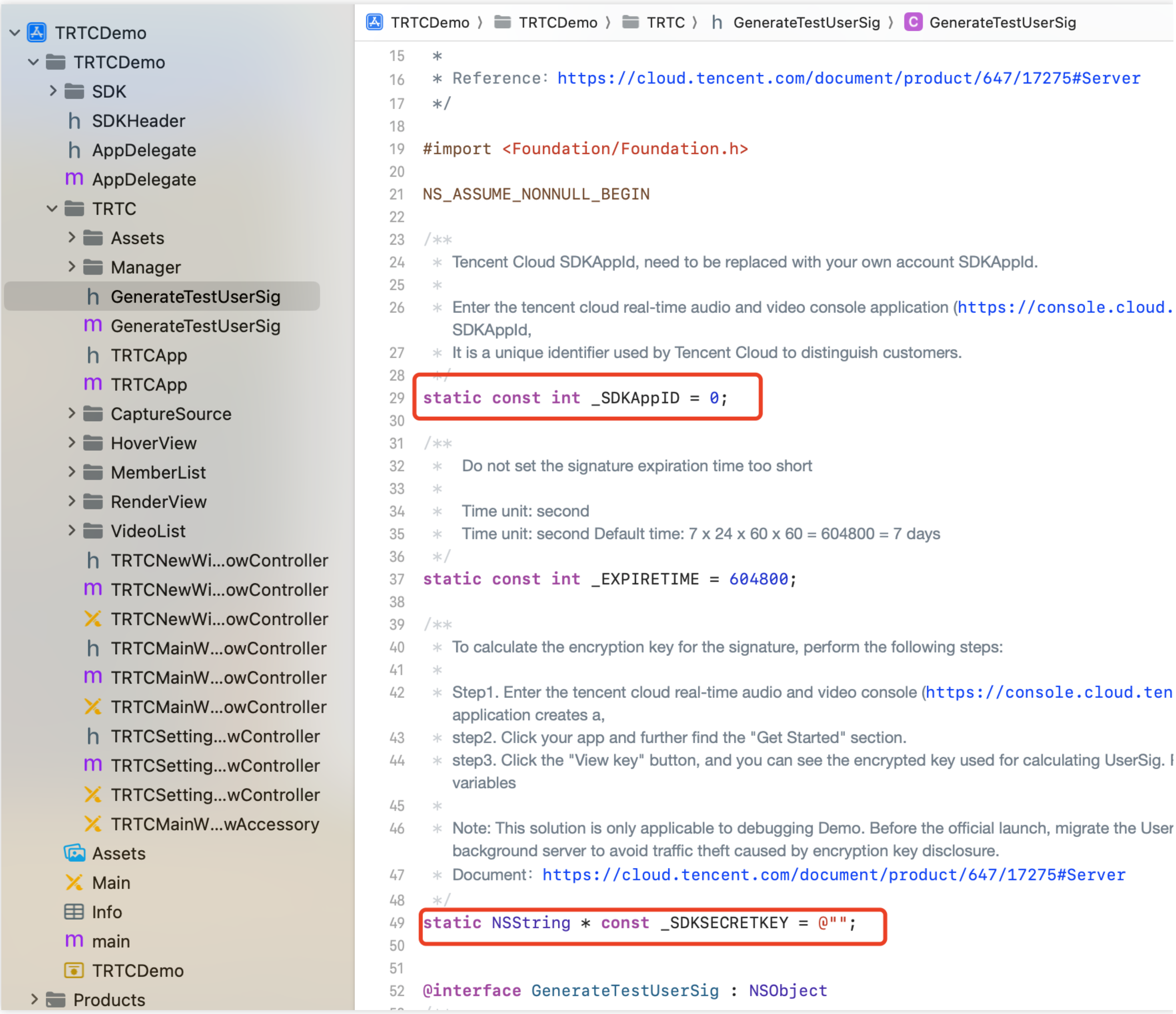
Step 2. Configure the Demo

1. Log in to the TRTC Console and click **Create Application**. If you have already done so, you may skip this step.

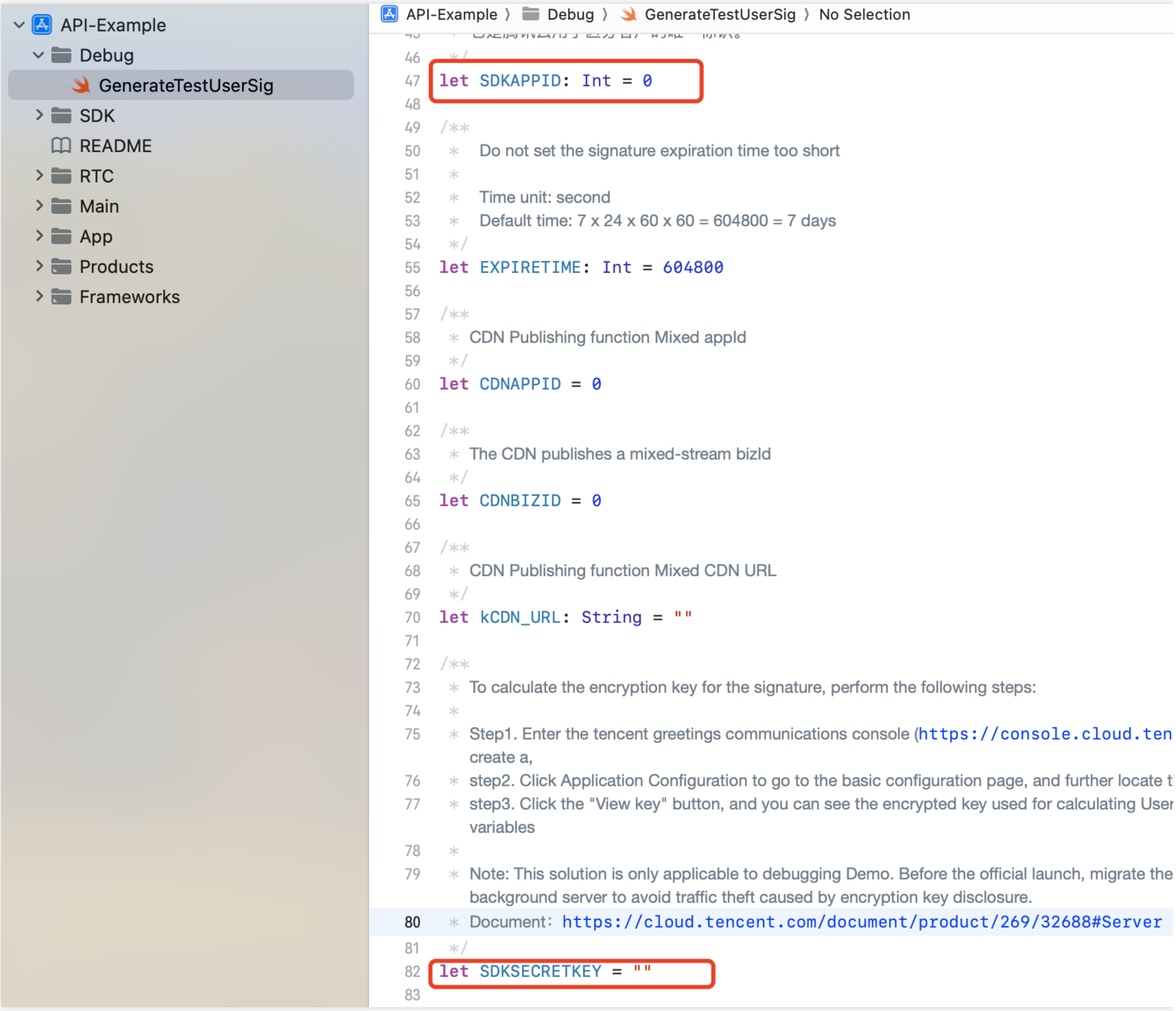
2. And then, your own `SDKAppID` and `SDKSecretKey` of your created application can be obtained in the **Basic Information** section.



3. If you choose **OCDemo,** replace the values of `SDKAPPID` and `SDKSECRETKEY` with the information you obtained in **Step 2** in `GenerateTestUserSig.h` file under `TRTCDemo/TRTC` directory.



Or if you choose **SwiftDemo,** replace the values of `SDKAPPID` and `SDKSECRETKEY` with the information you obtained in **Step 2** in `GenerateTestUserSig.swift` file under `API-Example/Debug` directory.



Note：

In the demo above, we used **SDKSecretKey** to generate **UserSig** locally in order to help you go through the demo easier. However, in the production environment, you are not supposed to generate userSig in this way, which may lead to **SDKSecretKey** leakage, thereby creating a chance for attackers to steal your TRTC traffic. **The correct way to generate UserSig is to integrate Server-Side Generation of UserSig on your server.** When an user enters the room:

Send a http request to your server.

Generate a UserSig on your server.

Return it to the user to enter the room.

When you deploy your page to a production environment, you need to have your page accessed through the HTTPS(e.g. `https://domain/xxx`). For the reason, please refer to the document Page Access Protocol Restriction Description.

Step 3. Run the Demo

Open the `TRTCDemo.xcworkspace/API-Example.xcworkspace` project in the source code directory with Xcode (11.0 or later) and compile and run the TRTC-API-Example project.

FAQs

If you encounter any problems with access and use, please refer to FAQs.

If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

Android

Last updated：2024-07-05 19:30:25

This document describes how to quickly run the demo for the TRTC Android SDK.

Prerequisites

Android 4.1 (SDK API level 16) or later; Android 5.0 (SDK API level 21) or later is recommended.
Android Studio 3.5 or later.
Devices with Android 4.1 or later.

Steps To Run Demo

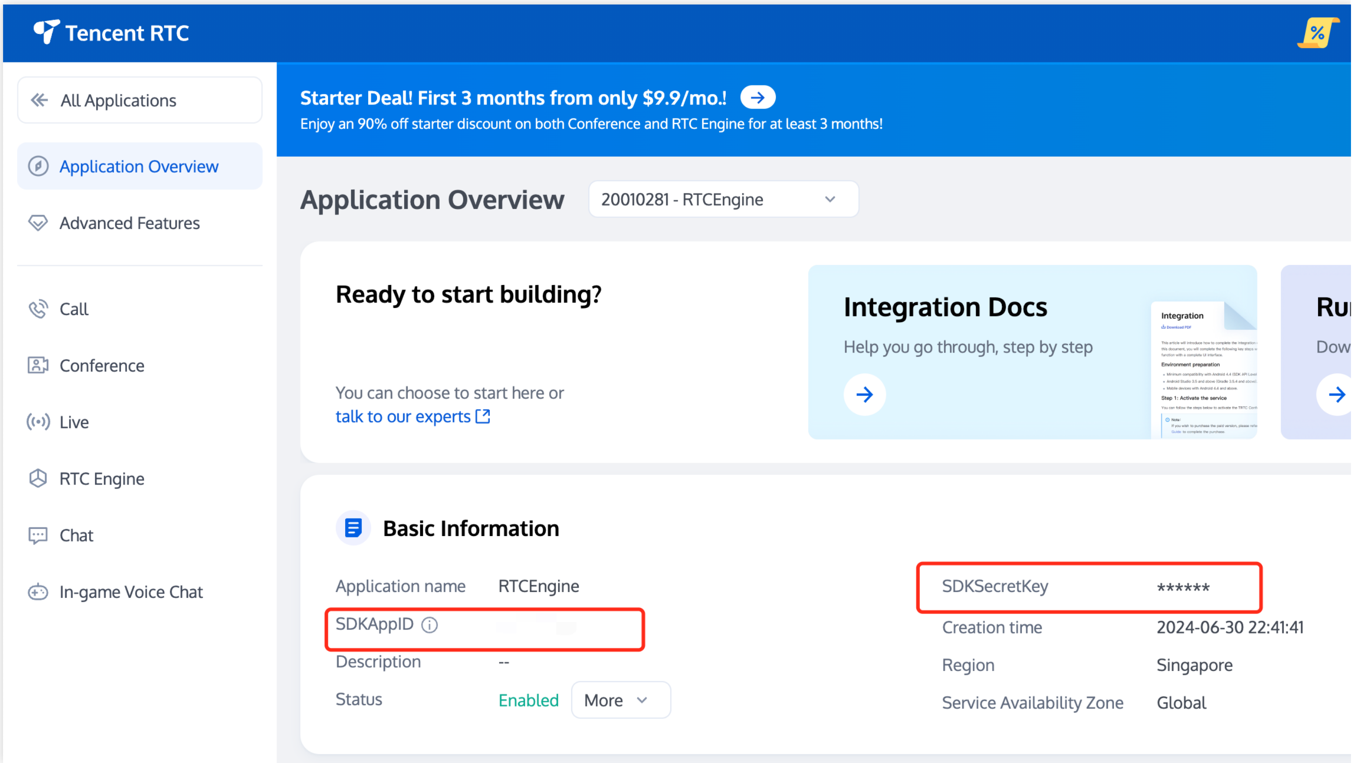
Step 1. Download the Demo

Download the Andorid sample demo code on github which SDK has been imported into sample project, or run the following command on the terminal:

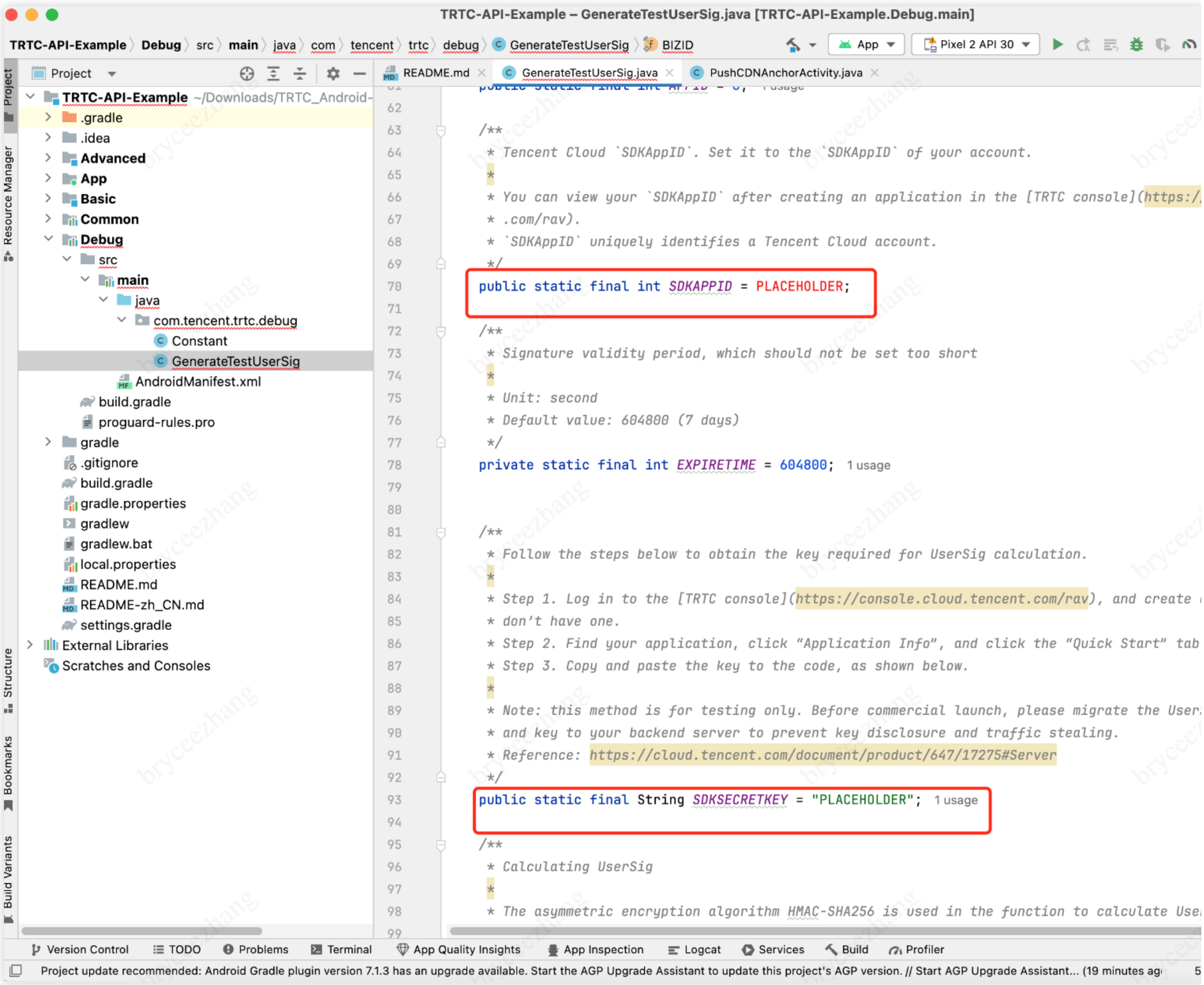
```
git clone https://github.com/Tencent-RTC/TRTC_Android.git
```

Step 2. Configure the Demo

1. Log in to the TRTC Console and click **Create Application**. If you have already done so, you may skip this step.
2. And then, your own **SDKAppID** and **SDKSecretKey** of your created application can be obtained in the Basic Information section.



3. Replace the values of **SDKAPPID** and **SDKSECRETKEY** with the information you obtained in **Step 2** in `GenerateTestUserSig` under `TRTC-API-Example/Debug/src/main/java/com.tencent.trtc.debug` directory.



Note：

In the demo above, we used **SDKSecretKey** to generate **UserSig** locally in order to help you go through the demo easier. However, in the production environment, you are not supposed to generate userSig in this way, which may lead to **SDKSecretKey** leakage, thereby creating a chance for attackers to steal your TRTC traffic. **The correct way to generate UserSig is to integrate Server-Side Generation of UserSig on your server.** When an user enters the room:

Send a http request to your server.

Generate a UserSig on your server.

Return it to the user to enter the room.

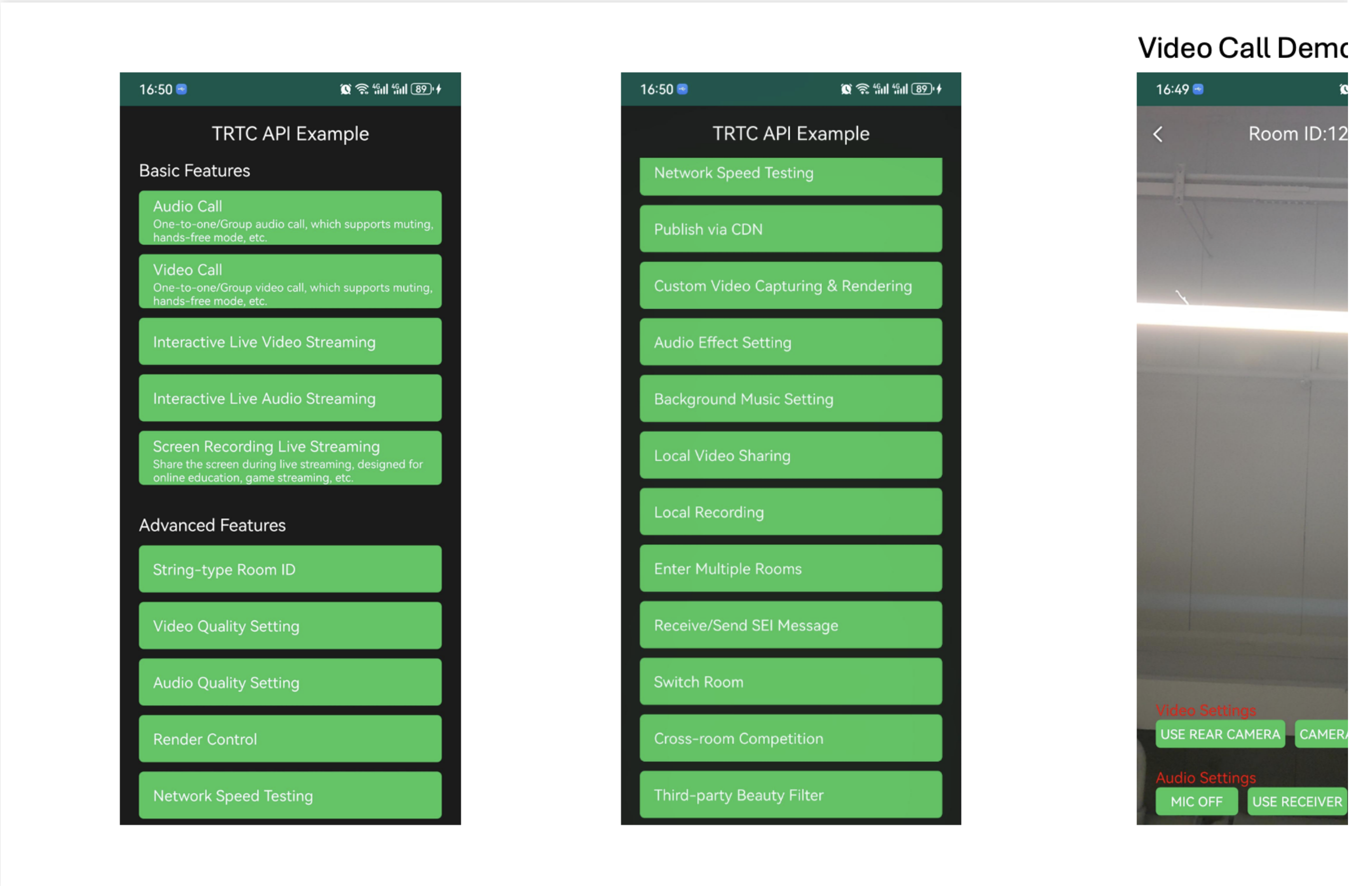
When you deploy your page to a production environment, you need to have your page accessed through the HTTPS(e.g. `https://domain/xxx`). For the reason, please refer to the document Page Access Protocol Restriction Description.

Step 3. Run the Demo

Open `TRTC-API Example` with Android Studio(v3.5 or later) and get it run.

Step 4. Experience the Demo

You can choose the functions you are interested in to experience.



FAQs

If you encounter any problems with access and use, please refer to FAQs.

If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

Windows C++

Last updated：2024-07-05 19:30:25

This document describes how to quickly run the demo for the TRTC Windows SDK.

Prerequisites

Install Visual Studio 2017 or later (v2019 is recommended).

Install Qt 5.14.x.

Find the right version of Qt Add-in for your Visual Studio on the Qt website. Download and install it.

Open Visual Studio, in the menu bar, select **Extension** > **QT VS Tools** > **Qt Options** > **Qt Versions**, and add a **MSVC compiler**.

Copy all the DLL files in `SDK/CPlusPlus/Win64/lib` (for 64-bit Windows) to the `debug/release` folder of the project directory.

Note：

`debug/release` is auto-generated after environment configuration in Visual Studio. For 32-bit Windows, copy all the DLL files in `SDK/CPlusPlus/Win64/lib to the debug/release` folder of the project directory.

Steps To Run Demo

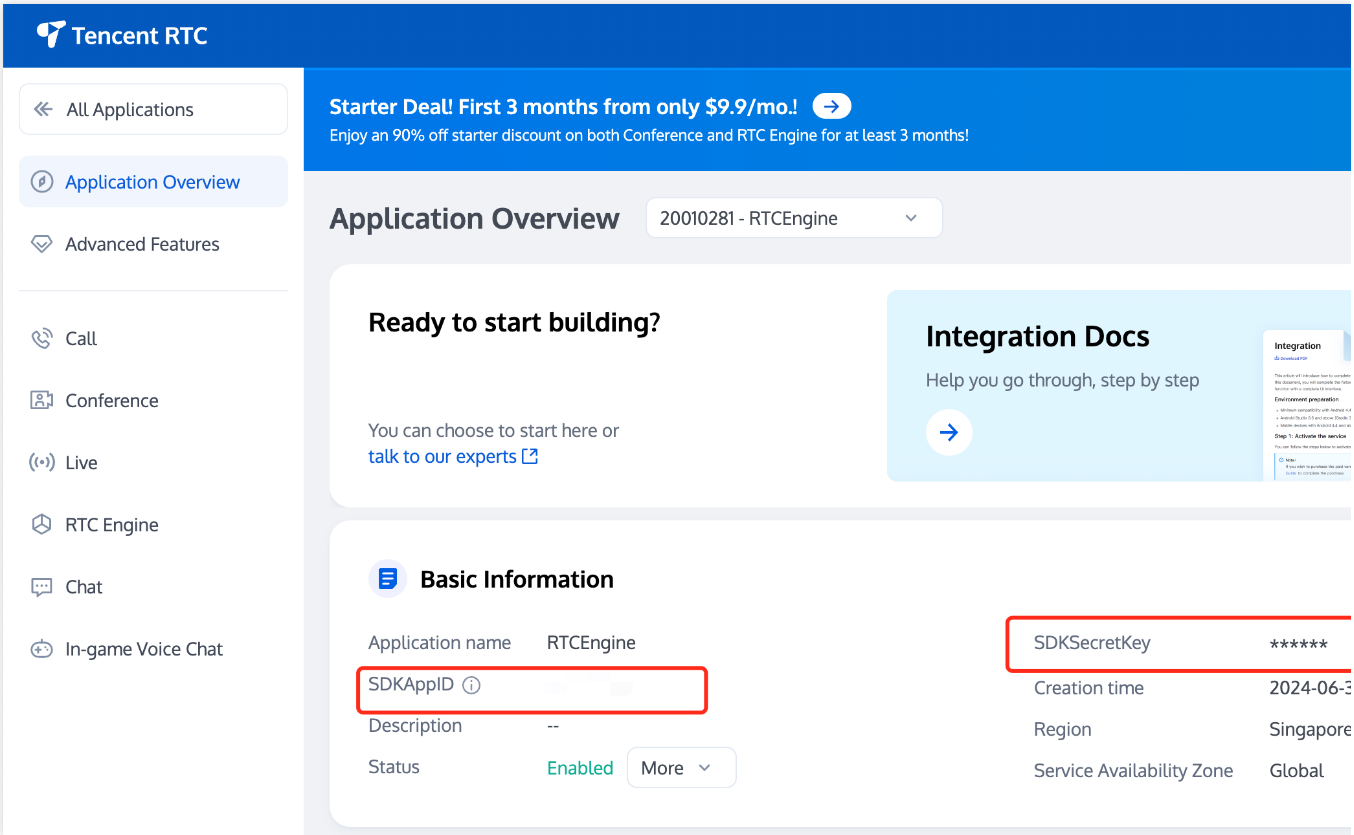
Step 1. Download the Demo

Download the TRTC_Windows-C++ sample demo code on github which SDK has been imported into, or run the following command on the terminal:

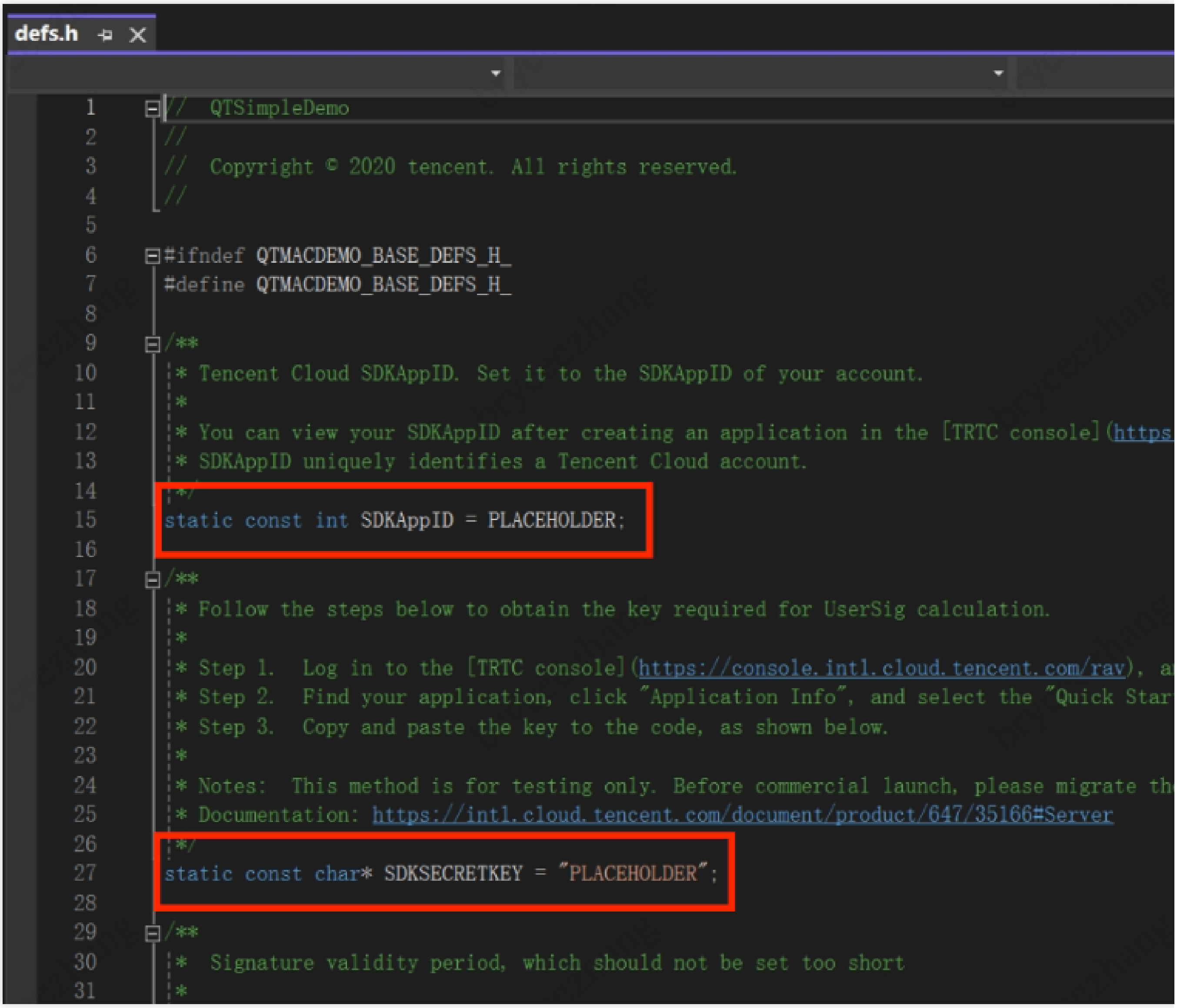
```
git clone https://github.com/Tencent-RTC/TRTC_Windows.git
```

Step 2. Configure the Demo

1. Log in to the TRTC Console and click **Create Application**. If you have already done so, you may skip this step.
2. And then, your own `SDKAppID` and `SDKSecretKey` of your created application can be obtained in the **Basic Information** section.



3. Replace the values of `SDKAPPID` and `SDKSECRETKEY` with the information you obtained in **Step 2** in `defs.h` file under `TRTC-API-Example-C++/TRTC-API-Example-Qt/src/Util/` directory.



Note：

In the demo above, we used **SDKSecretKey** to generate **UserSig** locally in order to help you go through the demo easier. However, in the production environment, you are not supposed to generate userSig in this way, which may lead to **SDKSecretKey** leakage, thereby creating a chance for attackers to steal your TRTC traffic. **The correct way to generate UserSig is to integrate Server-Side Generation of UserSig on your server.** When an user enters the room:

Send a http request to your server.

Generate a UserSig on your server.

Return it to the user to enter the room.

When you deploy your page to a production environment, you need to have your page accessed through the HTTPS(e.g. `https://domain/xxx`). For the reason, please refer to the document Page Access Protocol Restriction Description.

Step 3. Run the Demo

Open `QTDemo.sln` in the TRTC-API-Example-Qt directory with Microsoft Visual Studio (v2019 is recommended), set up the Qt environment (Qt 5.14 is recommended), and run the project.

FAQs

If you encounter any problems with access and use, please refer to FAQs.

If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

Web

Last updated：2024-06-24 11:14:54

This document describes how to quickly run the demo for the TRTC Web SDK.

Prerequisites

You need to create a Tencent RTC account.

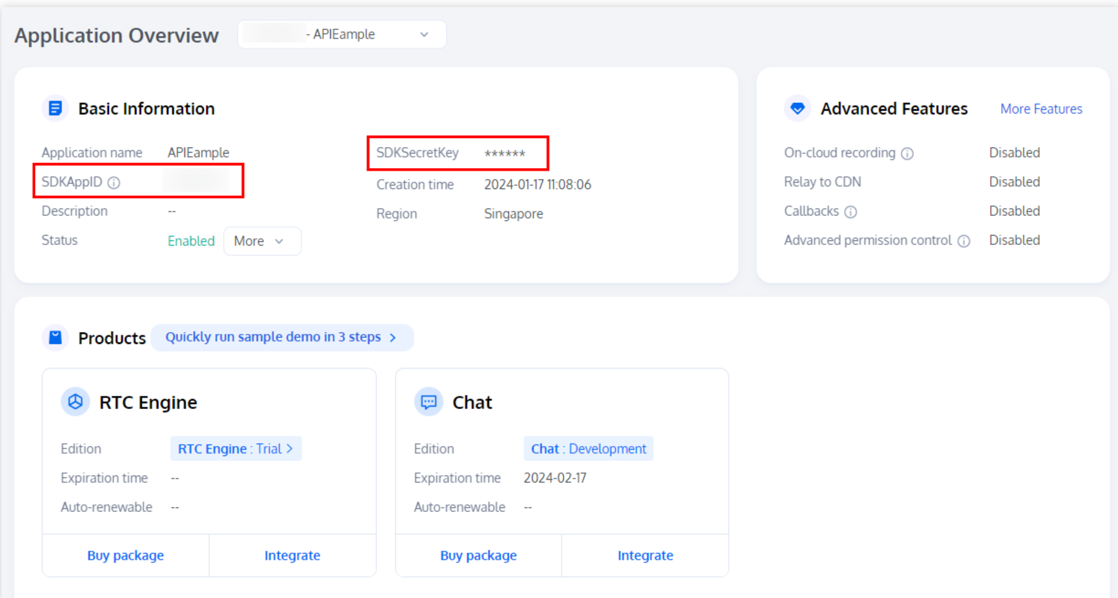
Steps To Run Demo

1. Create an application

Log in to the TRTC Console and create an **RTC Engine** application. If you have already done so, you may skip this step.

2. Get SDKAppID and SDKSecretKey

Get `SDKAppID` and `SDKSecretKey` of your created application in the Basic Information.



3. Run demo

1. Run demo online.

We provide the following basic demos. You may choose a project framework that you are familiar with to experience the demo:

1.1 quick-demo-js is a demo based on Javascript. Souce code: GitHub.

1.2 quick-demo-vue2-js is a demo based on Vue2 + Javascript. Source code: GitHub.

1.3 quick-demo-vue3-ts is a demo based on Vue3 + Typescript. Source code: GitHub.

2. Run demo locally.

2.1 Download source code by GitHub or Zip.

2.2 Run demo locally with the following steps.

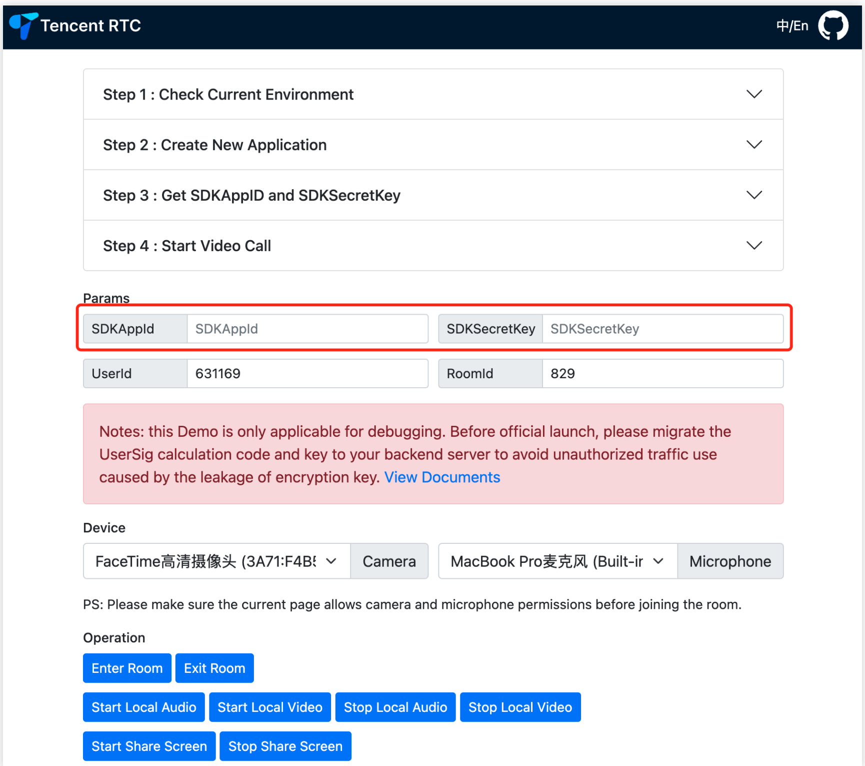
quick-demo-js

quick-demo-vue2-js

quick-demo-vue3-ts

1. Open `TRTC_Web/quick-demo-js/index.html` .

2. Fill in the **SDKAppId** and **SDKSecretKey** obtained in **Step2**



3. Function Experience:

Click **Enter Room** to enter the room.

Click **Start Local Audio/Video** to capture microphone or camera.

Click **Stop Local Audio/Video** to stop capturing microphone or camera.

Click **Start Share Screen** to start screen sharing.

Click **Stop Share Screen** to cancel screen sharing.

4. After entering the room, you can invite others to experience the TRTC Web voice and video communication feature together by sharing the invitation link.

1. Change to the directory `TRTC_Web/quick-demo-vue2-js/` .

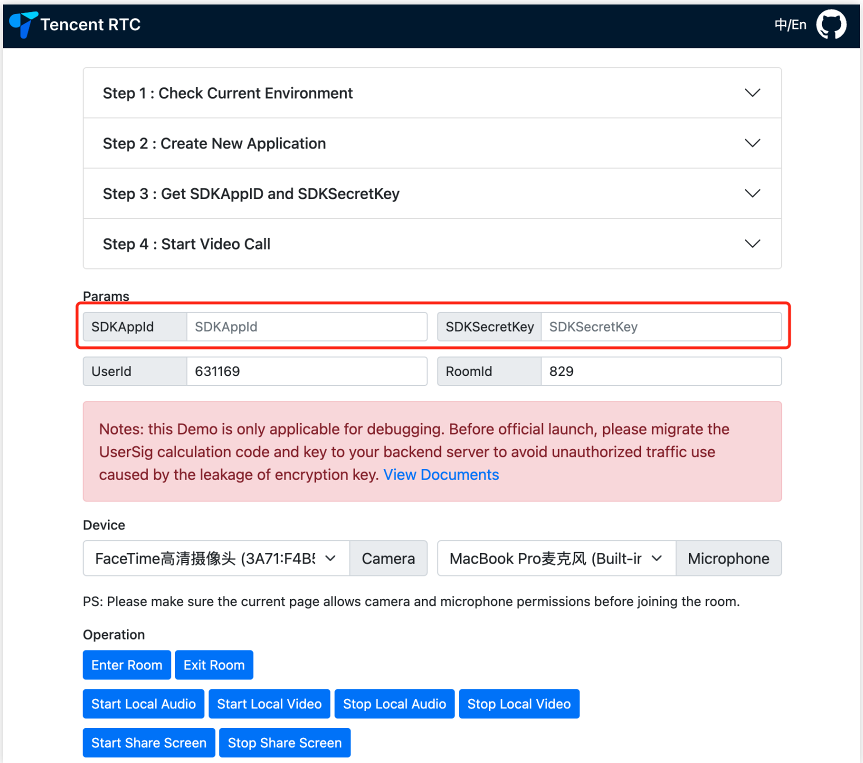
```
cd quick-demo-vue2-js
```

2. Run the demo locally by executing the following command in the terminal, which will automatically install dependencies and run the demo.

You may need to install Node.js in advance.

```
npm start
```

3. The default browser will automatically open the address `http://localhost:8080/` .
4. Fill in the **SDKAppId** and **SDKSecretKey** obtained in **Step2**.



5. Experience

Input userId and roomId

Click **Enter Room** to enter the room

Click **Start Local Audio/Video** to capture microphone or camera

Click **Stop Local Audio/Video** to stop capturing microphone or camera

Click **Start Share Screen** to start screen sharing

Click **Stop Share Screen** to stop screen sharing

6. After entering the room, you can invite others to experience TRTC's web-based audio and video communication feature by sharing the invitation link.

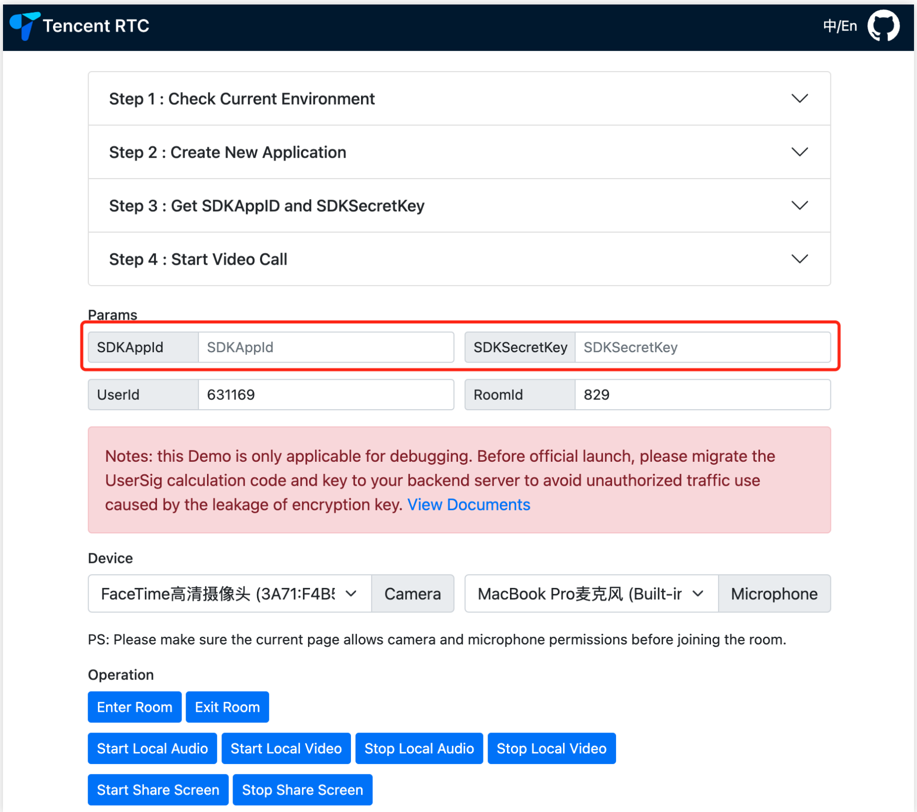
1. Change to the directory `TRTC_Web/quick-demo-vue3-ts/` .

2. Run the demo locally by executing the following command in the terminal, which will automatically install dependencies and run the demo.

You may need to install Nodejs first.


```
npm start
```

3. The default browser will automatically open the address `http://localhost:3000/` .
4. Fill in the **SDKAppId** and **SDKSecretKey** obtained in **Step2.**



5. Experience

Input userId and roomId

Click **Enter Room** to enter the room

Click **Start Local Audio/Video** to capture microphone or camera

Click **Stop Local Audio/Video** to stop capturing microphone or camera

Click **Start Share Screen** to start screen sharing

Click **Stop Share Screen** to stop screen sharing

6. After entering the room, you can invite others to experience TRTC's web-based audio and video communication feature by sharing the invitation link.

Note：

In the demo above, we used **SDKSecretKey** to generate **UserSig** locally in order to help you go through the demo easier. However, in the production environment, you are not supposed to generate userSig in this way, which may lead to **SDKSecretKey** leakage, thereby creating a chance for attackers to steal your TRTC traffic.

The correct way to generate UserSig is to integrate Server-Side Generation of UserSig on your server.

When an user enters the room:

Send a http request to your server.

Generate a UserSig on your server.

Return it to the user to enter the room.

When you deploy your page to a production environment, you need to have your page accessed through the HTTPS(e.g. `https://domain/xxx`). For the reason, please refer to the document Page Access Protocol

Restriction Description.

Start Integration

Please refer to SDK Quick Start.

FAQs

Supported platforms

TRTC Web SDK supports desktop and major mobile browsers(**Chrome, Edge, Safari, Firefox, Opera**). For details, please refer to Supported Platforms.

Others

More useful information at Web FAQs.

Electron

Last updated：2024-05-13 10:43:56

This document shows you how to quickly run TRTC-API-Example (Electron).

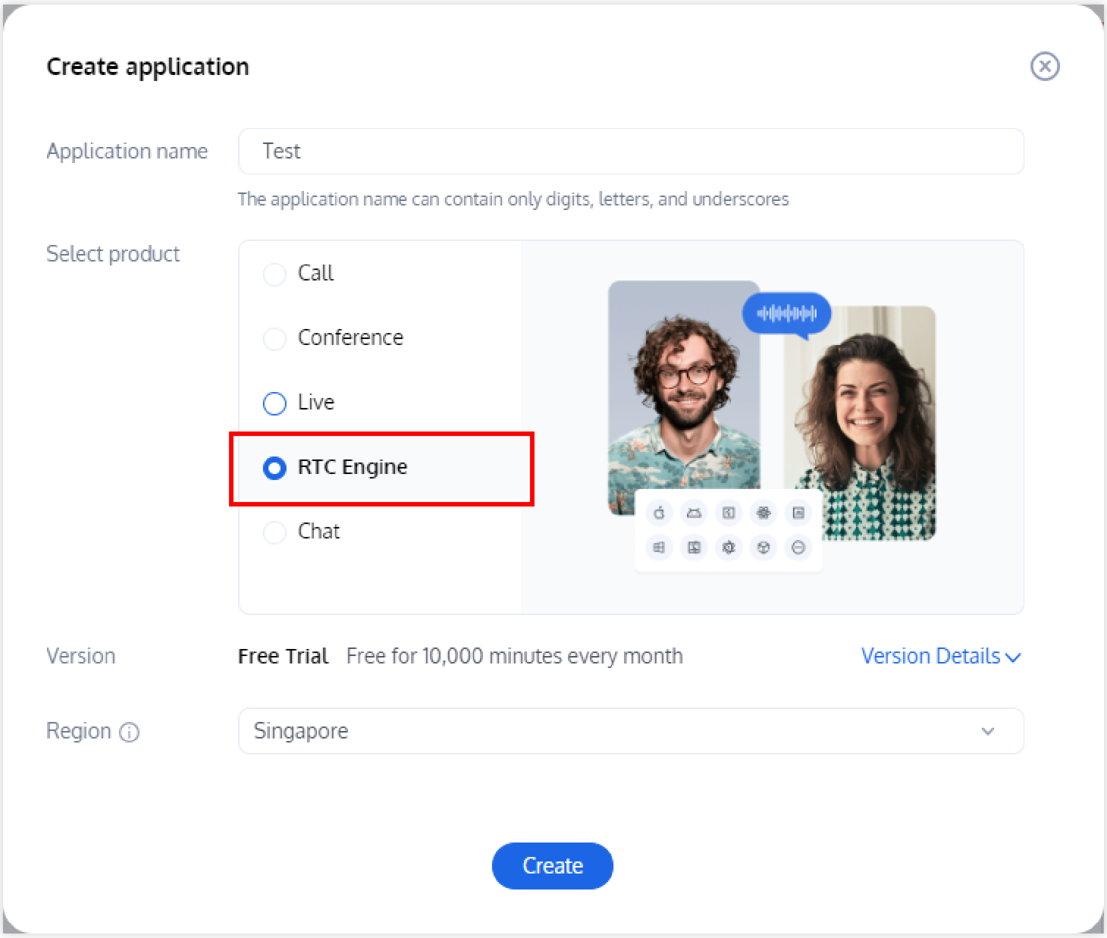
Prerequisites

You have signed up for a Tencent Cloud account.

Directions

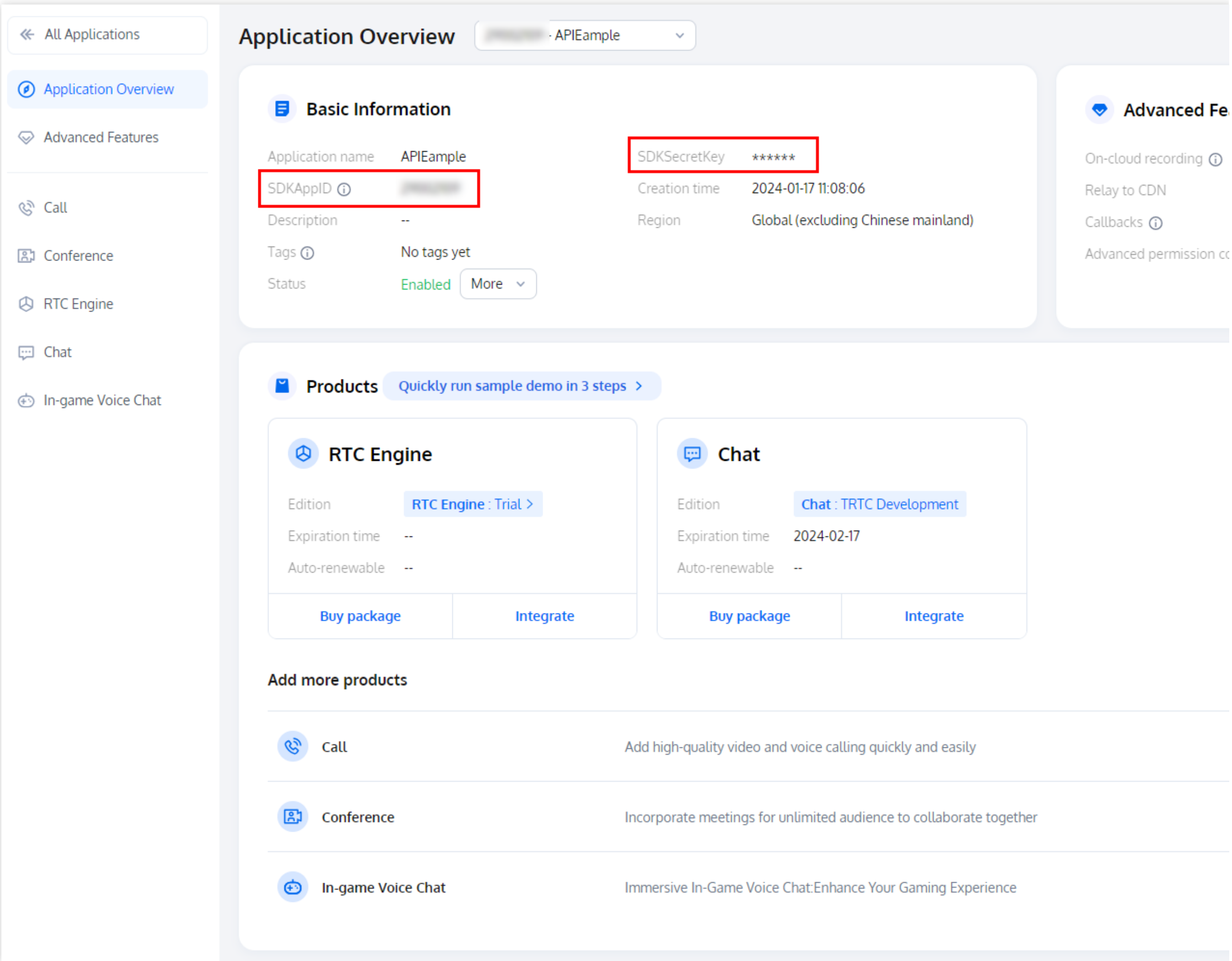
Step 1. Create an application

1. Log in to the TRTC console overview page, click **Create Application**.
2. In the popup page, select RTC Engine, enter the application name, and then click **Create**.



Step 2. Get your SDKAppId and SecretKey

After your application created, you can get your `SDKAppID` and `SDKSecretKey` on Basic informaction. `SDKAppID` and `SDKSecretKey` is needed for running demo.



Step 3. Download the sample code

1.Go to GitHub to download the sample code for your platform.


```
git clone https://github.com/LiteAVSDK/TRTC_Electron.git
```

2.The steps to import SDK can refer to Electron SDK import.

Step 4. Configure the project

Find and open `Electron/TRTC-API-Example/assets/debug/gen-test-user-sig.js` and set the following parameters:

`SDKAPPID` : A placeholder by default. Set it to the actual SDKAppID.

`SDKSECRETKEY` : A placeholder by default. Set it to the actual key.

Note

The method for generating `UserSig` described in this document involves configuring `SDKSECRETKEY` in the client code. In this method, `SDKSECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for the local execution and debugging of TRTC-API-Example**.

The best practice is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to your server for a dynamic `UserSig`. For more information, see How do I calculate UserSig during production?.

FAQs

1. What firewall restrictions does the SDK face?

The SDK uses the UDP protocol for audio/video transmission and therefore cannot be used in office networks that block UDP. If you encounter such a problem, refer to Firewall Restrictions to troubleshoot the issue.

Flutter

Last updated：2024-05-13 11:31:34

This document describes how to quickly run the demo for the TRTC Flutter SDK.

Note

Currently, screen sharing and device selection are not supported on Windows or macOS.

Environment Requirements

Flutter 2.0 or later.

Developing for Android:

Android Studio 3.5 or later.

Devices with Android 4.1 or later.

Developing for iOS and macOS:

Xcode 11.0 or later.

OS X 10.11 or later.

A valid developer signature for your project.

Developing for Windows:

OS: Windows 7 SP1 or later (64-bit based on x86-64).

Disk space: At least 1.64 GB of space after the IDE and relevant tools are installed.

Visual Studio 2019.

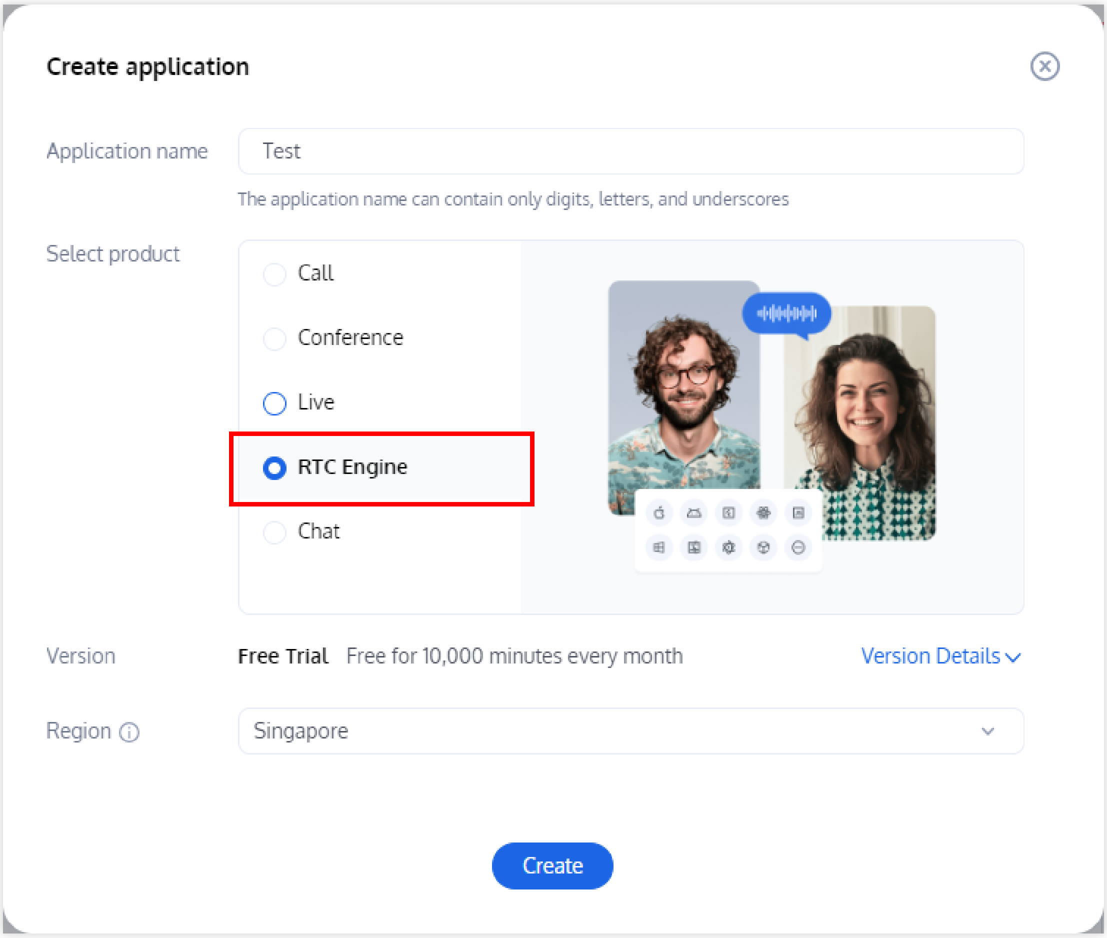
Prerequisites

You have signed up for a Tencent Cloud account.

Directions

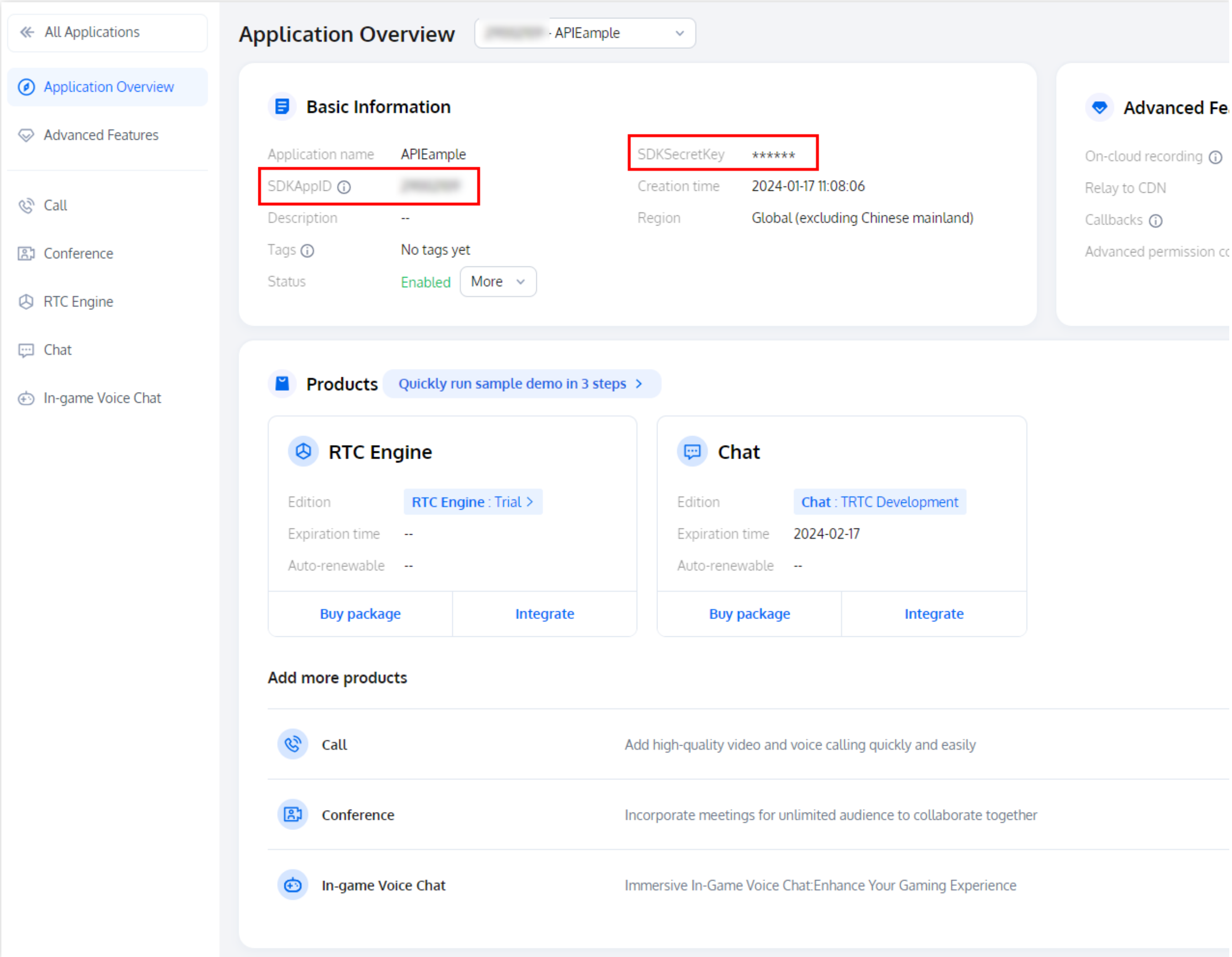
Step 1. Create an application

1. Log in to the TRTC console overview page, click **Create Application**.
2. In the popup page, select RTC Engine, enter the application name, and then click **Create**.



Step 2. Get your SDKAppId and SecretKey

After your application created, you can get your `SDKAppID` and `SDKSecretKey` on Basic informaction. `SDKAppID` and `SDKSecretKey` is needed for running demo.



Step 3. Download the sample code

1.Go to GitHub to download the SDK and demo source code.

```
git clone https://github.com/LiteAVSDK/TRTC_Flutter.git
```

2.The steps to import SDK can refer to Flutter SDK import.

Step 4. Configure the project

Open the file downloaded previously, find and open `/lib/debug/GenerateTestUserSig.dart` , and set the following parameters:

`SDKAPPID` : A placeholder by default. Set it to the actual `SDKAppID` .

`SDKSECRETKEY` : A placeholder by default. Set it to the actual key.

Note

The method for generating UserSig described in this document involves configuring `SDKSECRETKEY` in the client code. In this method, `SDKSECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, this method is only suitable for the local execution and debugging of TRTC-Simple-Demo.

The best practice is to integrate the calculation code of UserSig into your server and provide an application-oriented API. When UserSig is needed, your application can send a request to your server for a dynamic UserSig. For more information, see How do I calculate UserSig during production?.

Step 5. Compile and run the demo

1. Execute `flutter pub get` .
2. Build and run the project.

Android

1. Execute `flutter run` .
2. Open the demo project with Android Studio (3.5 or later), and run the project.

iOS

1. Execute `cd ios` .
2. Execute `pod install` .
3. Open `/ios` in the source code directory with Xcode (11.0 or later). Compile and run the demo project.

Windows

1. Execute `flutter config --enable-windows-desktop` .
2. Execute `flutter run -d windows` .

macOS

1. Execute `flutter config --enable-macos-desktop` .
2. Execute `cd macos` .
3. Execute `pod install` .
4. Execute `flutter run -d macos` .

FAQs

How do I view TRTC logs?

TRTC logs are compressed and encrypted by default (XLOG format). You can find them at the following paths:
iOS: `Documents/log` of the application sandbox

Android:

v6.7 or earlier: `/sdcard/log/tencent/liteav`

v6.8 or later: `/sdcard/Android/data/package name/files/log/tencent/liteav/`

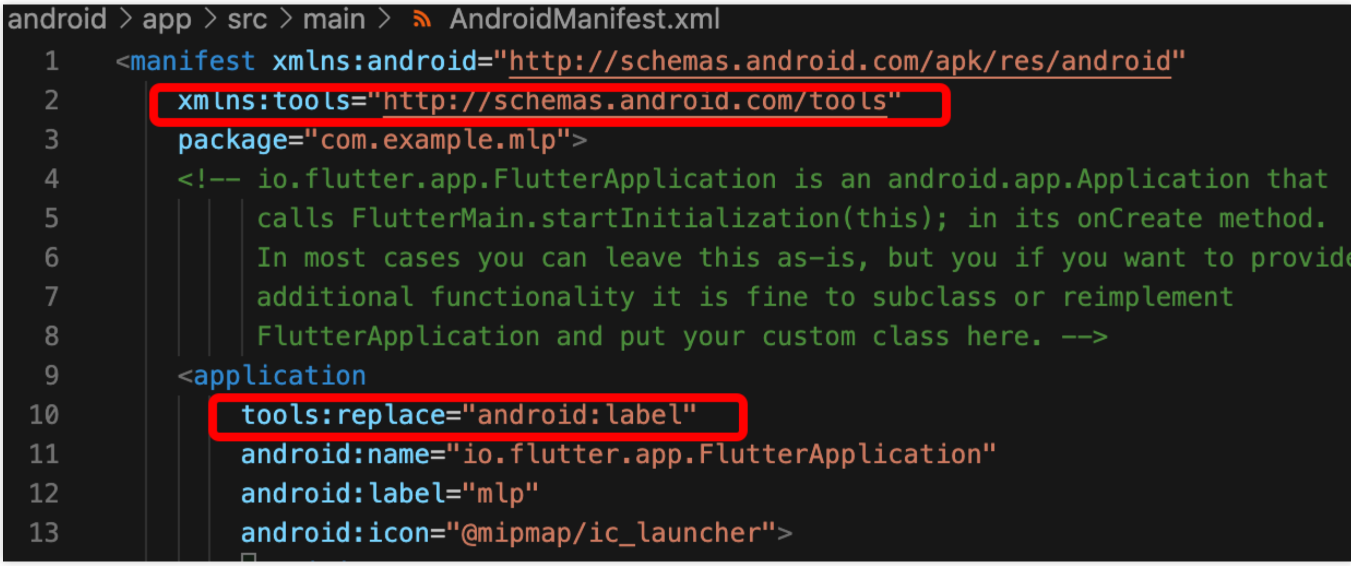
What should I do if videos show on Android but not on iOS?

Make sure that in `info.plist` of your project, the value of `io.flutter.embedded_views_preview` is `YES` .

What should I do if the "Manifest merge failed" error occurs in Android Studio?

Open `/example/android/app/src/main/AndroidManifest.xml` .

1. Add `xmlns:tools="http://schemas.android.com/tools"` to `manifest` .
2. Add `tools:replace="android:label"` to `application` .



Note

For more FAQs, see Flutter.

Unity

Last updated：2024-05-13 10:43:55

This document shows you how to integrate the TRTC SDK in Unity to enable audio/video calls in games.

The demo includes the following features:

Room entry/exit

Custom video rendering

Device management and music/audio effects

Note

For details about the APIs and their parameters, see Overview.

Unity 2020.2.1f1c1 is recommended.

Supported platforms: Android, iOS, Windows, macOS (alpha testing)

Modules required: `Android Build Support` , `iOS Build Support` , `Windows Build Support` , `MacOS Build Support` .

If you are developing for iOS, you also need:

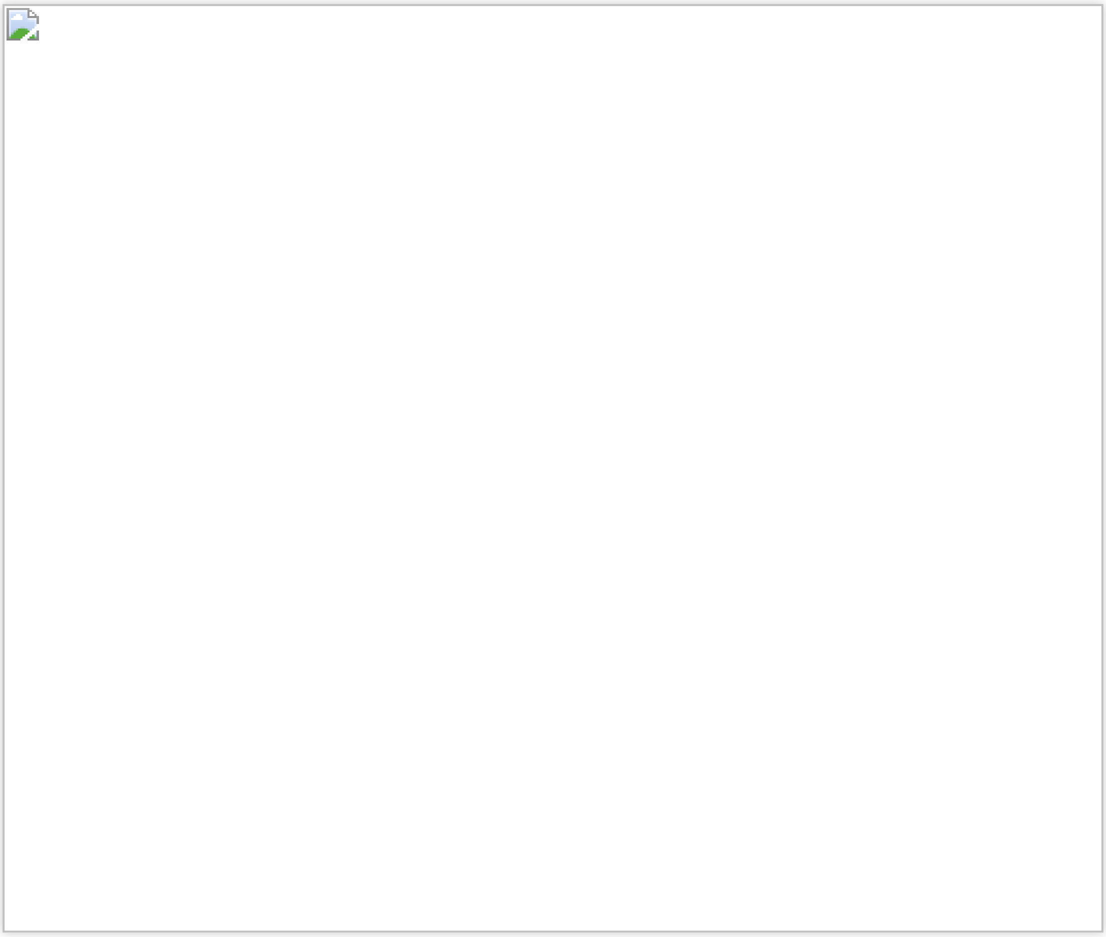
Xcode 11.0 or later

A valid developer signature for your project

Directions

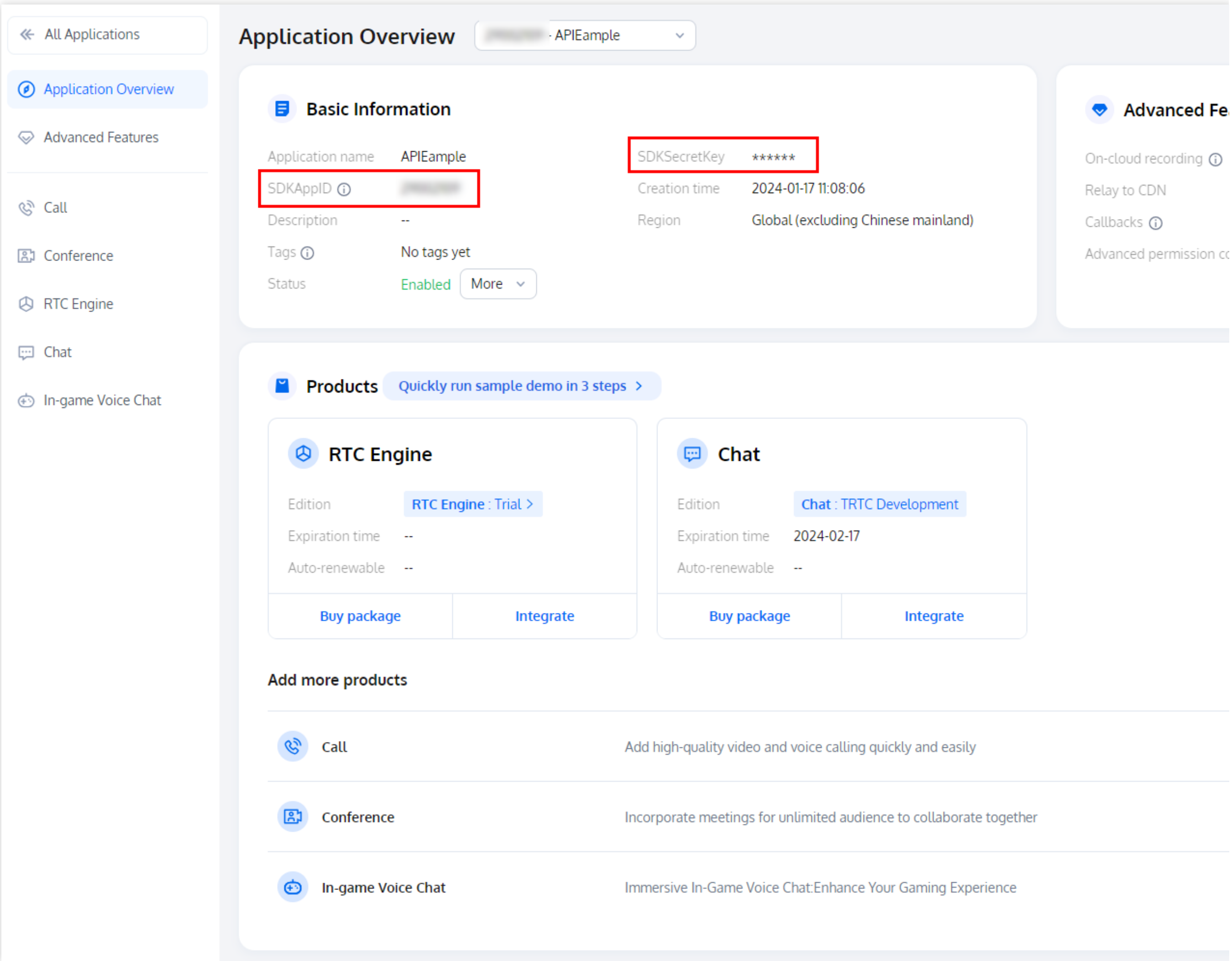
Step 1. Create an application

1. Log in to the TRTC console overview page, click **Create Application**.
2. In the popup page, select RTC Engine, enter the application name, and then click **Create**.



Step 2. Get your SDKAppId and SecretKey

After your application created, you can get your `SDKAppID` and `SDKSecretKey` on Basic informaction. `SDKAppID` and `SDKSecretKey` is needed for running demo.



Step 3. Download the sample code

1. Go to GitHub to download the SDK and demo source code.


```
git clone https://github.com/LiteAVSDK/TRTC_Unity.git
```

2. The steps to import SDK can refer to Unity SDK import.

Step 4. Configure the project

1. Open the file downloaded previously, find and open `TRTC-Simple-Demo/Assets/TRTCSDK/Demo/Tools/GenerateTestUserSig.cs`, and set the following parameters:

`SDKAPPID`: A placeholder by default. Set it to the actual `SDKAppID`.

`SDKSECRETKEY`: A placeholder by default. Set it to the actual key.

Note

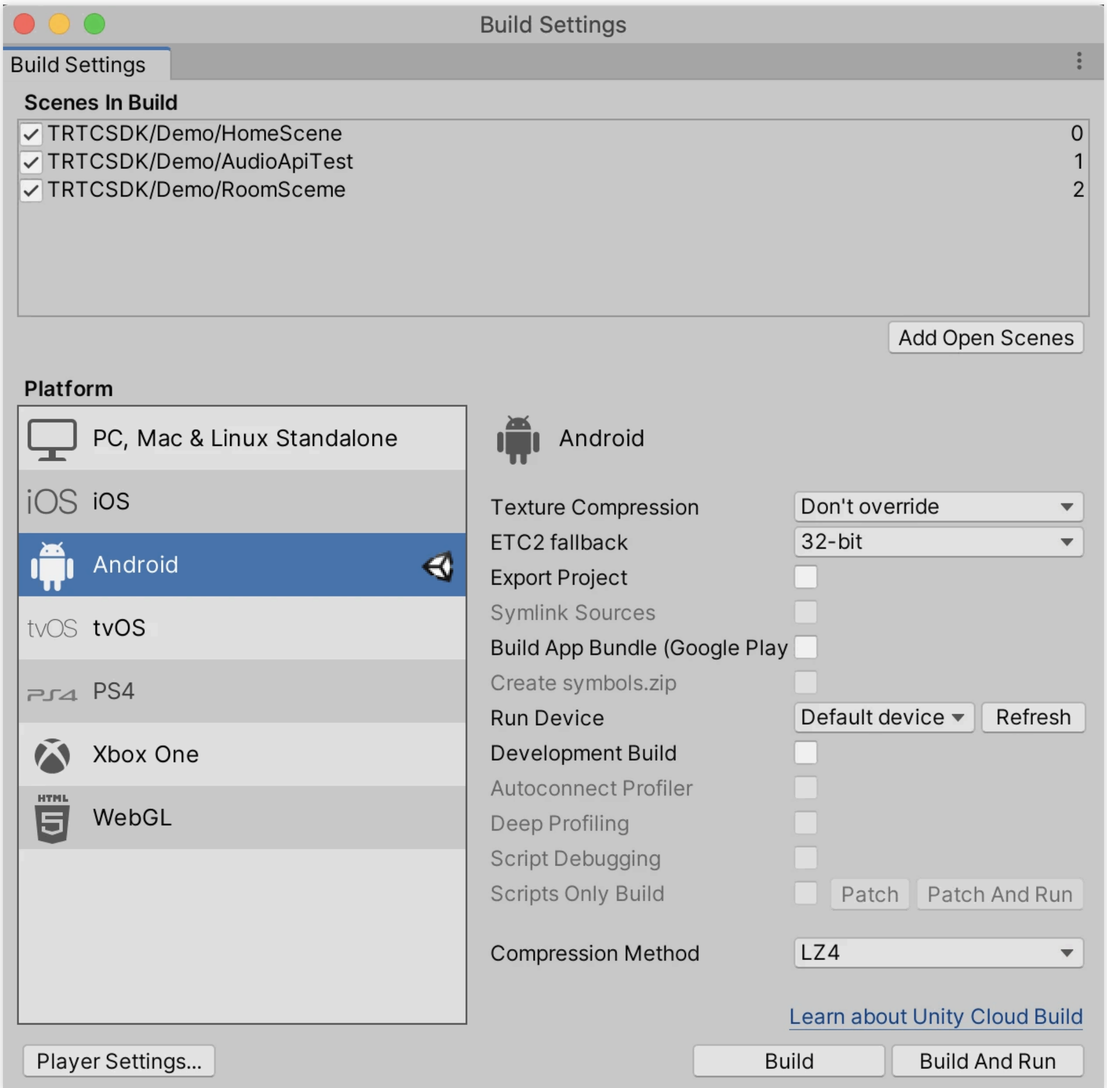
The method for generating `UserSig` described in this document involves configuring `SDKSECRETKEY` in the client code. In this method, `SDKSECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for the local execution and debugging of TRTC-Simple-Demo**.

The best practice is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to your server for a dynamic `UserSig`. For more information, see How do I calculate UserSig during production?.

Step 5. Compile and run the demo

Android

1. Open Unity Editor, go to **File** > **Build Settings**, and select **Android** for **Platform**.

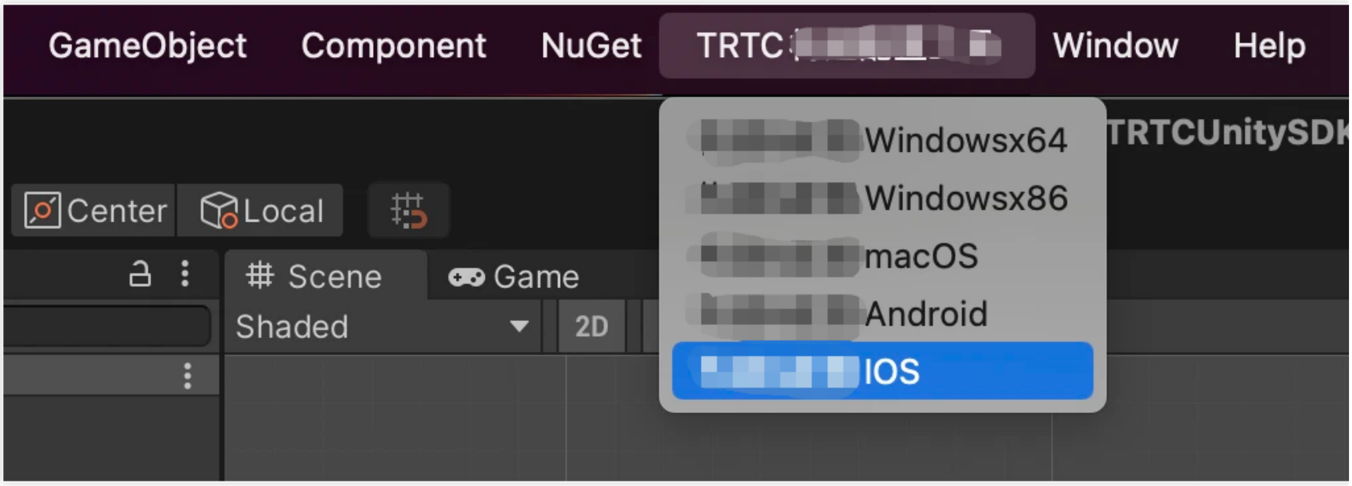


2. Connect to a real Android device and click **Build And Run** to run the demo.

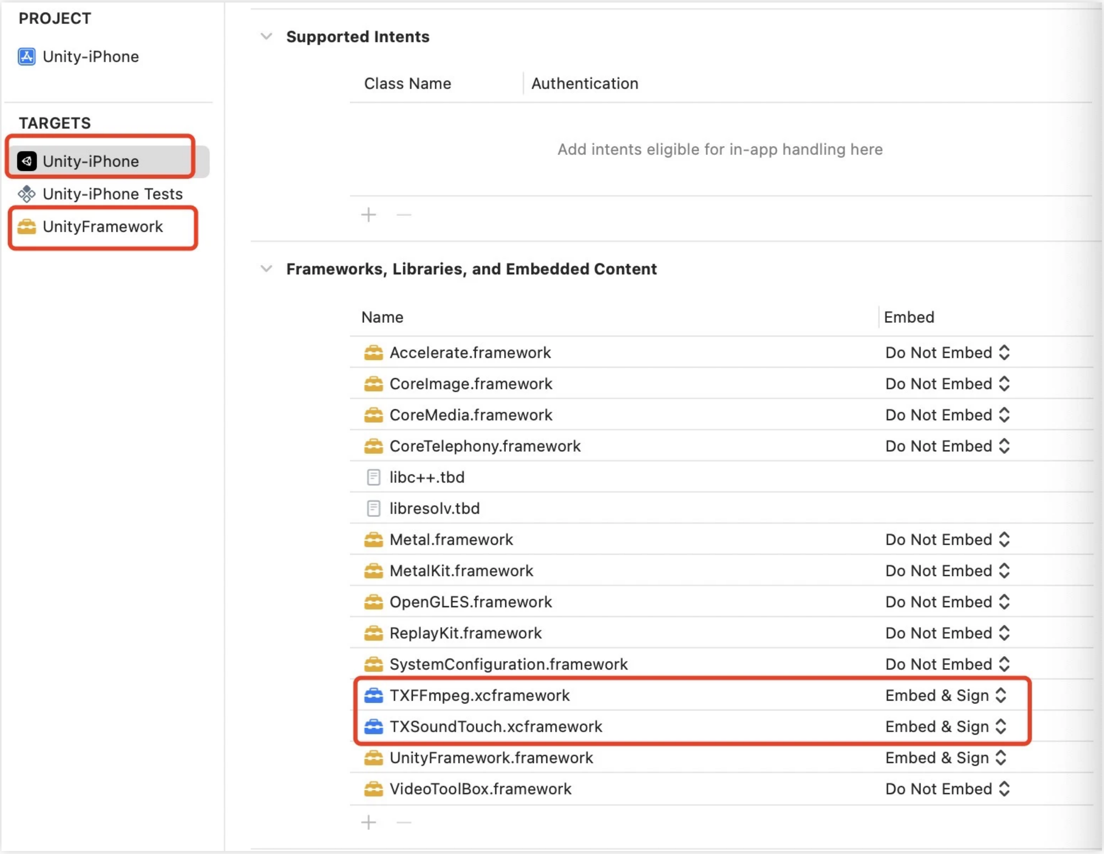
3. Call `enterRoom` first and go on to test other APIs. The data display window shows whether the call is successful, and the other window displays the callback information.

iOS

1. Open the TRTC build and configuration tool (from the menu at the top).
2. Click **Build & Configure iOS** to generate a project.



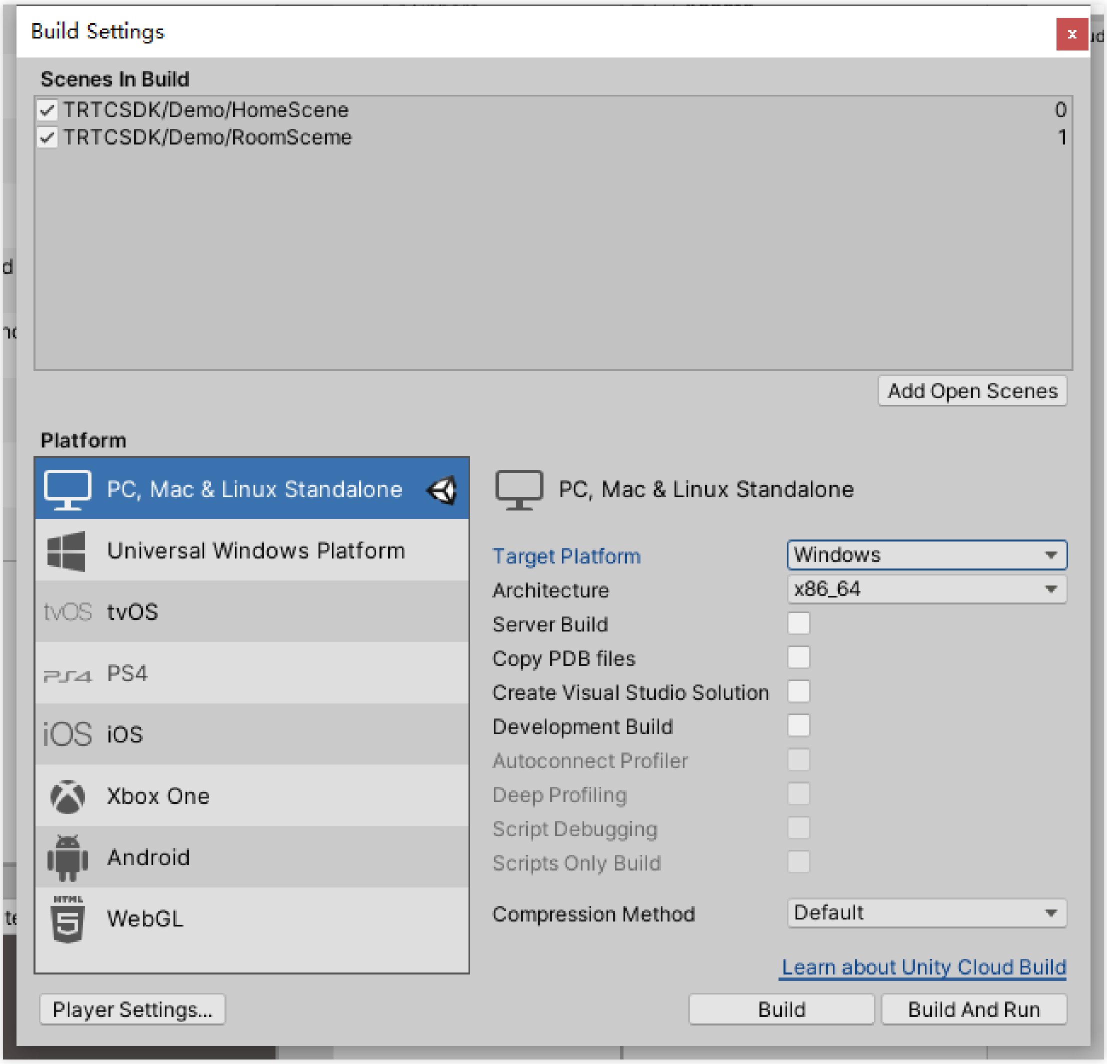
3. Open the project generated (`Unity-iPhone.xcodeproj`) with Xcode.
4. Download the underlying TRTC SDK. Click **General**, select **Frameworks, Libraries, and Embedded Content**, click **+** at the bottom to add the dynamic libraries required – `FFmpeg.xcframework` and `SoundTouch.xcframework` , and click **Embed & Sign**.



5. Connect to a real iOS device to debug the project.

Windows

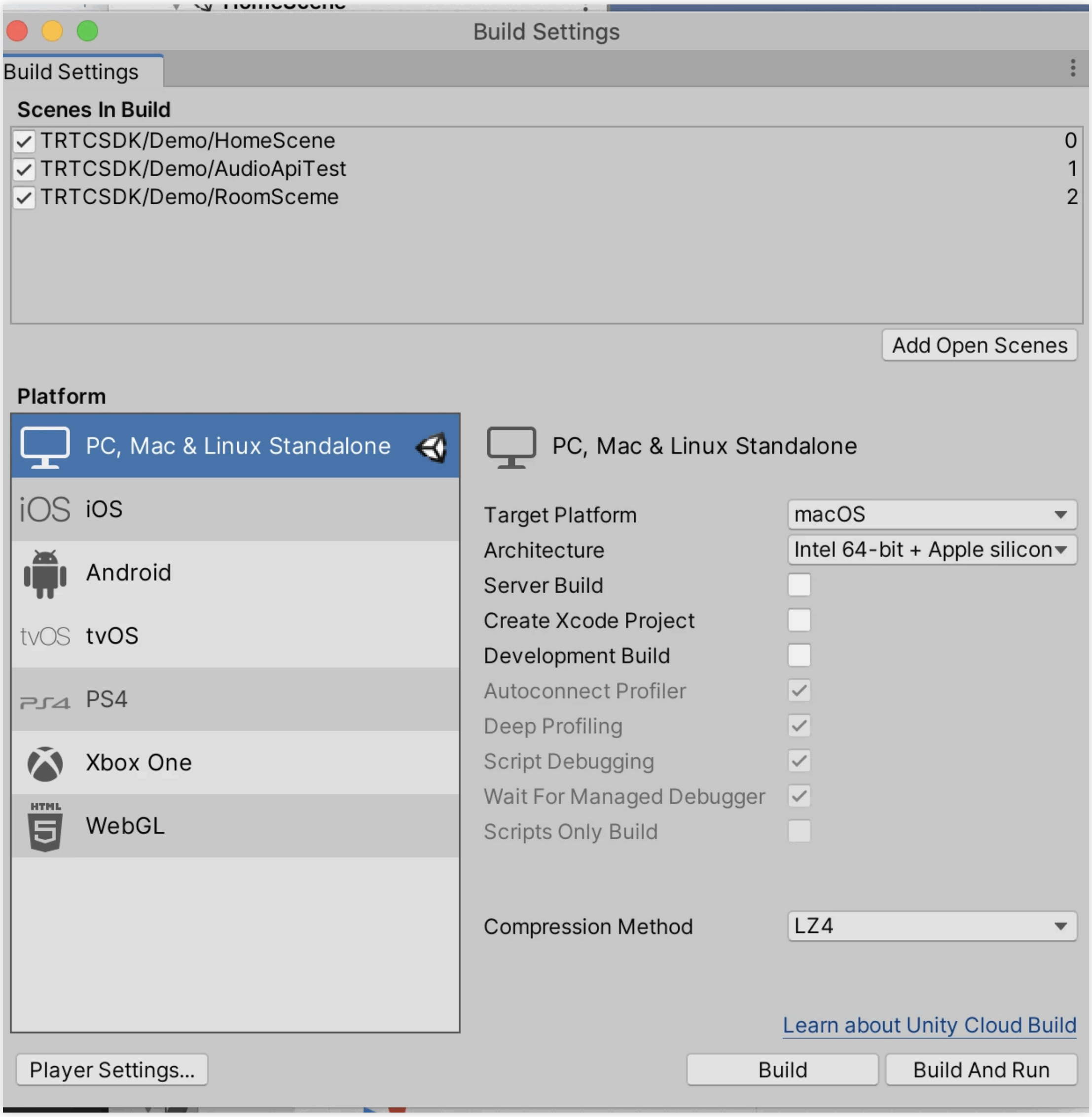
1. Open Unity Editor, go to **File** > **Build Settings**, and select **PC, Mac & Linux Standalone** for **Platform**, and **Windows** for **Target Platform**.



2. Click **Build And Run** to run the demo.

macOS

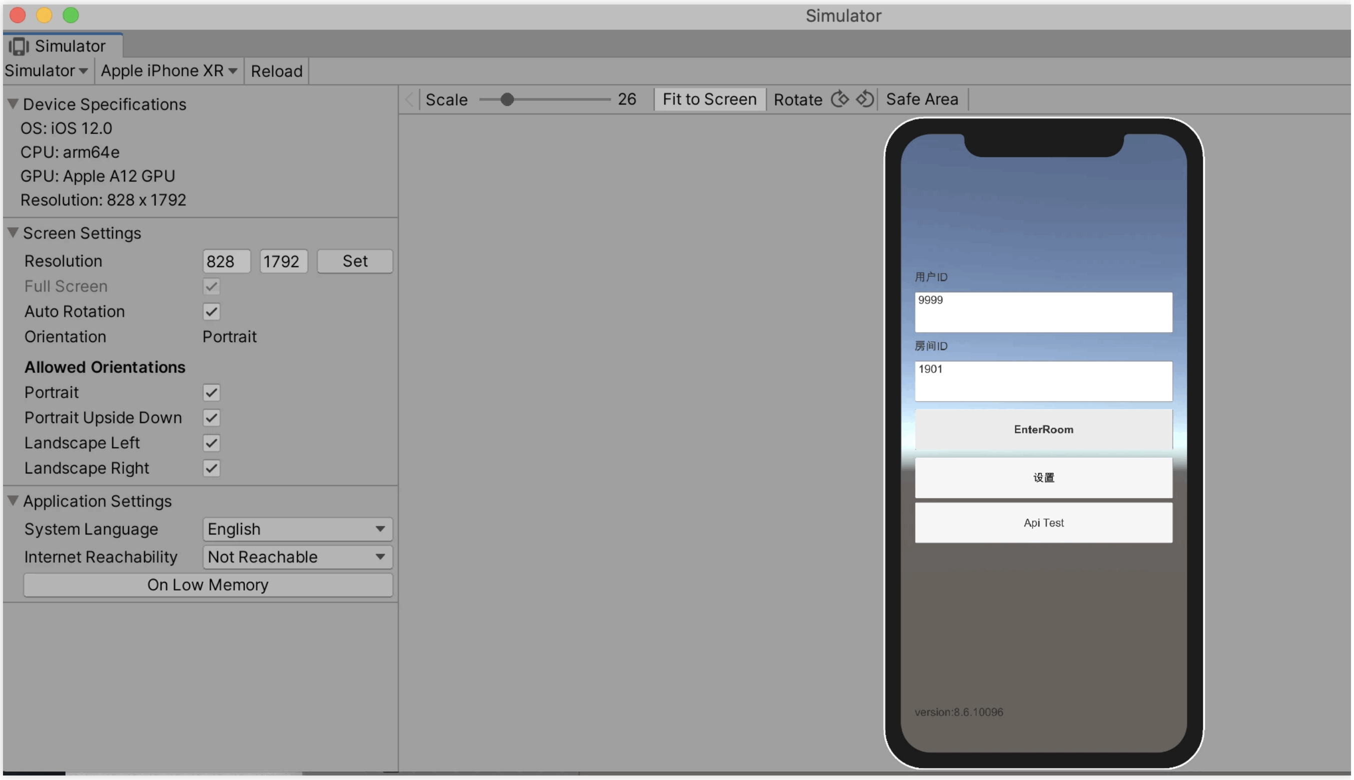
1. Open Unity Editor, go to **File** > **Build Settings**, and select **PC, Mac & Linux Standalone** for **Platform**, and **macOS** for **Target Platform**.



2. Click **Build And Run** to run the demo.

3. To use the simulator feature of Unity Editor, you must install `Device Simulator Package` .

4. Click **Windows** > **General** > **Device Simulator**.



Demo

The demo integrates most of the APIs launched so far, which can be used for testing and as reference for API calls. For more information about APIs, see Client APIs > Unity > Overview.

Note

The UI of the latest version of the demo may look different.

Directory Structure

```
├─Assets
├── Editor                            // Unity Editor script
│    ├── BuildScript.cs               // Unity Editor build menu
│    ├── IosPostProcess.cs            // Script for building iOS application in Unity E
├── Plugins
│    ├── Android
│    │    ├── AndroidManifest.xml    //Android configuration file
├── StreamingAssets                   // Audio/Video stream files for the Unity demo
├── TRTCSDK
     ├── Demo                         // Unity demo
     ├── SDK                          // TRTC SDK for Unity
```



```
          ├── Implement              // Implementation of TRTC SDK for Unity
          ├── Include                // Header files of TRTC SDK for Unity
          └── Plugins                // Underlying implementation of TRTC SDK for Unit
```

React Native

Last updated：2024-05-13 10:43:55

This document describes how to quickly run the TRTC demo for React Native.

Environment Requirements

React Native 0.63 or later
Node (later than v12) & Watchman
Developing for Android:
Android Studio 3.5 or later
Devices with Android 4.1 or later
Developing for iOS and macOS:
Xcode 11.0 or later
OS X 10.11 or later
A valid developer signature for your project
For how to set up the environment, see the React Native official document.

Prerequisites

You have signed up for a Tencent Cloud account.

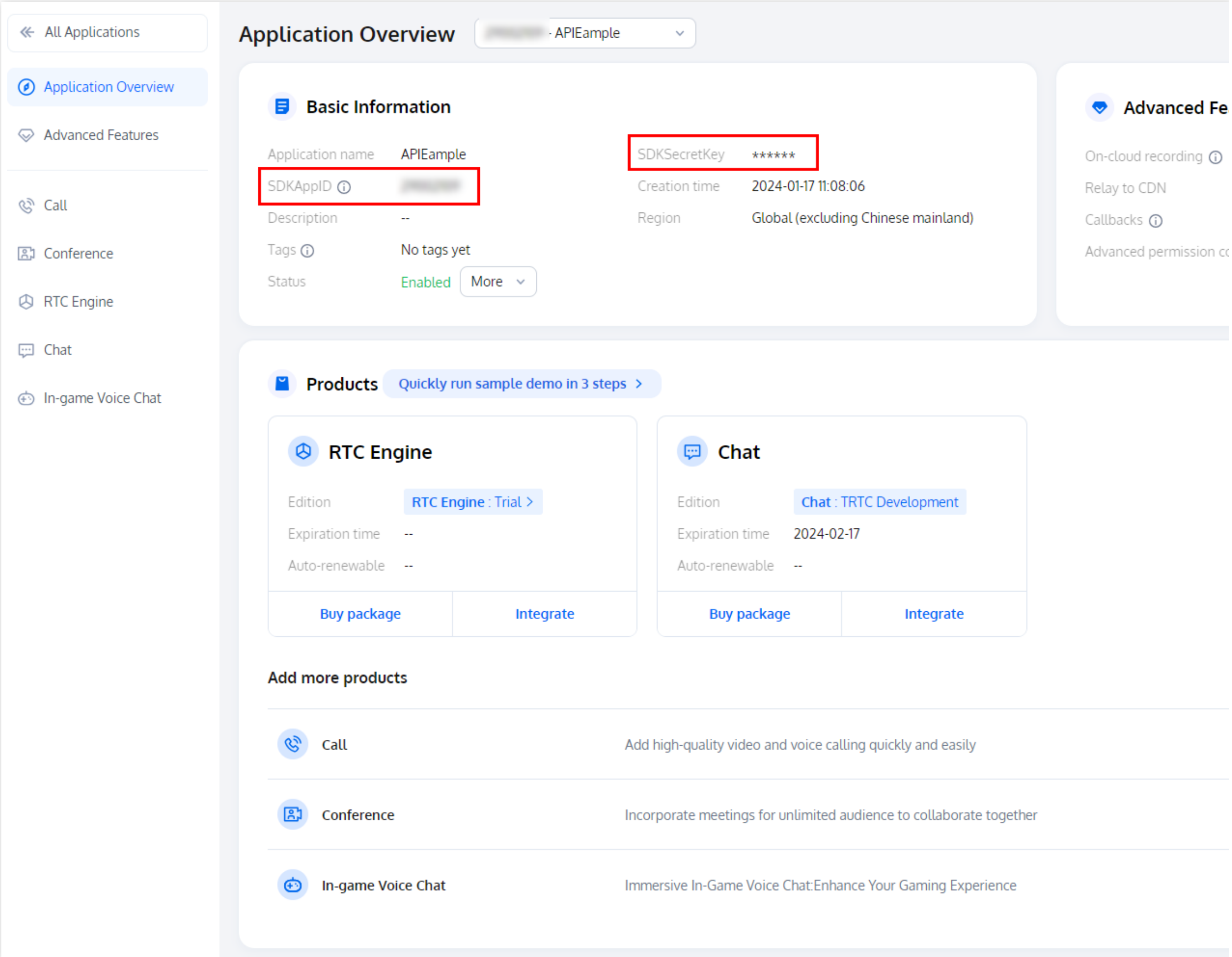
Directions

Step 1. Create an application

1. Log in to the TRTC console overview page, click **Create Application**.
2. In the popup page, select RTC Engine, enter the application name, and then click **Create**.

Step 2. Get your SDKAppId and SecretKey

After your application created, you can get your `SDKAppID` and `SDKSecretKey` on Basic informaction.
`SDKAppID` and `SDKSecretKey` is needed for running demo.



Step 3. Download the sample code

Go to GitHub to download the SDK and demo source code.

```
git clone https://github.com/LiteAVSDK/TRTC_ReactNative.git
```

Step 4. Configure the project

1. Open the file downloaded previously, find and open `/TRTC-Simple-Demo/debug/config.js` , and set the following parameters:

`SDKAPPID` : A placeholder by default. Set it to the actual `SDKAppID` .

`SDKSECRETKEY` : A placeholder by default. Set it to the actual key.

Note

The method for generating `UserSig` described in this document involves configuring `SDKSECRETKEY` in the client code. In this method, `SDKSECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for the local execution and debugging of TRTC-Simple-Demo**.
The best practice is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to your server for a dynamic `UserSig`. For more information, see How do I calculate UserSig during production?.

Step 5. Configure permission requests

You need to configure permission requests in order to run the demo.

Android

1. Configure application permissions in `AndroidManifest.xml`. The TRTC SDK requires the following permissions:

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

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

Do not use `android:hardwareAccelerated="false"` . Disabling hardware acceleration will result in failure to render remote videos.

You need to request audio and video permissions manually for Android.

```
if (Platform.OS === 'android') {
  await PermissionsAndroid.requestMultiple([
    PermissionsAndroid.PERMISSIONS.RECORD_AUDIO, //For audio calls
    PermissionsAndroid.PERMISSIONS.CAMERA, // For video calls
  ]);
```

```
}
```

iOS

1. Configure application permissions in `Info.plist` . The TRTC SDK requires the following permissions:

```
<key>NSCameraUsageDescription</key>
<string>You can make video calls only if you grant the app camera permission.</stri
<key>NSMicrophoneUsageDescription</key>
<string>You can make audio calls only if you grant the app mic permission.</string>
```

Step 6. Build and run the project

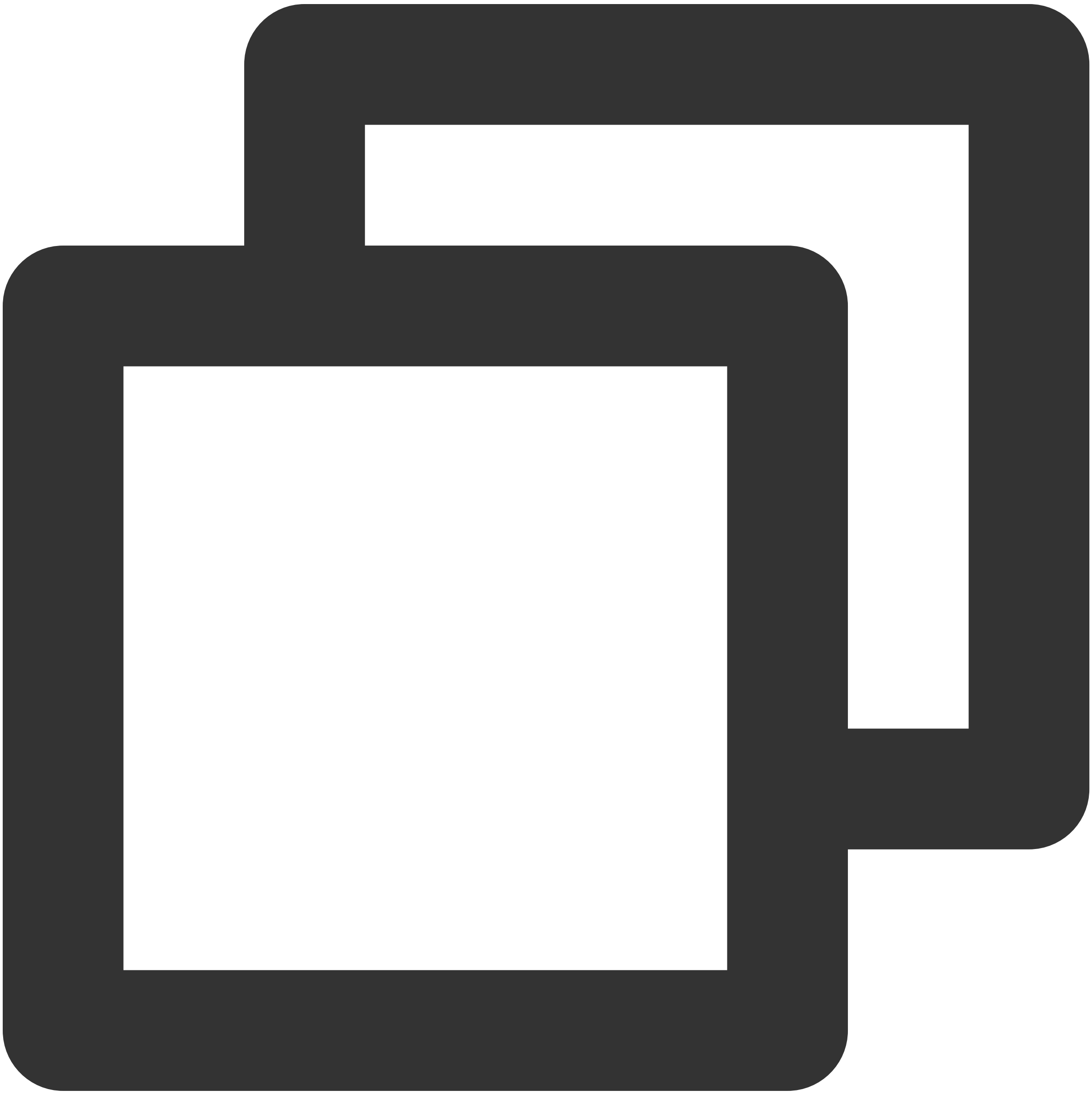
Run `npm install` .

Android

1. Start Metro in the demo directory.

```
npx react-native start
```

2. Open a new window in the demo directory and start debugging.



```
npx react-native run-android
```

iOS

1. Run `pod install` in the demo iOS directory to install dependencies.
2. Start Metro in the demo directory.

```
npx react-native start
```

3. Open a new window in the demo directory and start debugging (if an error occurs, please use Xcode to debug your project).

```
npx react-native run-ios
```

API Usage Guidelines
SDK Quick Start
Android

Last updated：2024-07-18 15:19:28

This tutorial mainly introduces how to implement a basic audio and video call.

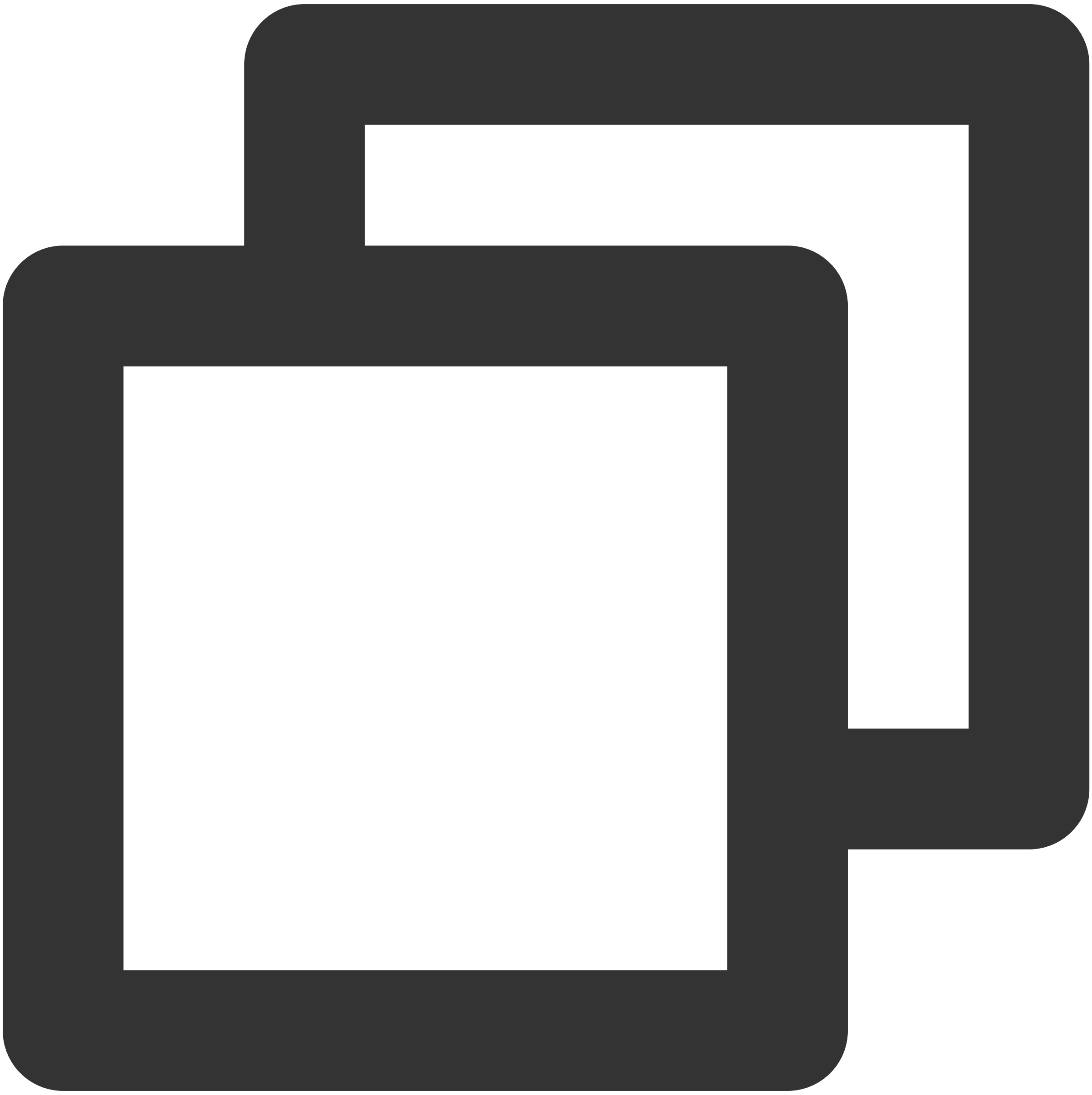
Prerequisites

Android Studio 3.5 or later.

Android 4.1 (SDK API level 16) or later

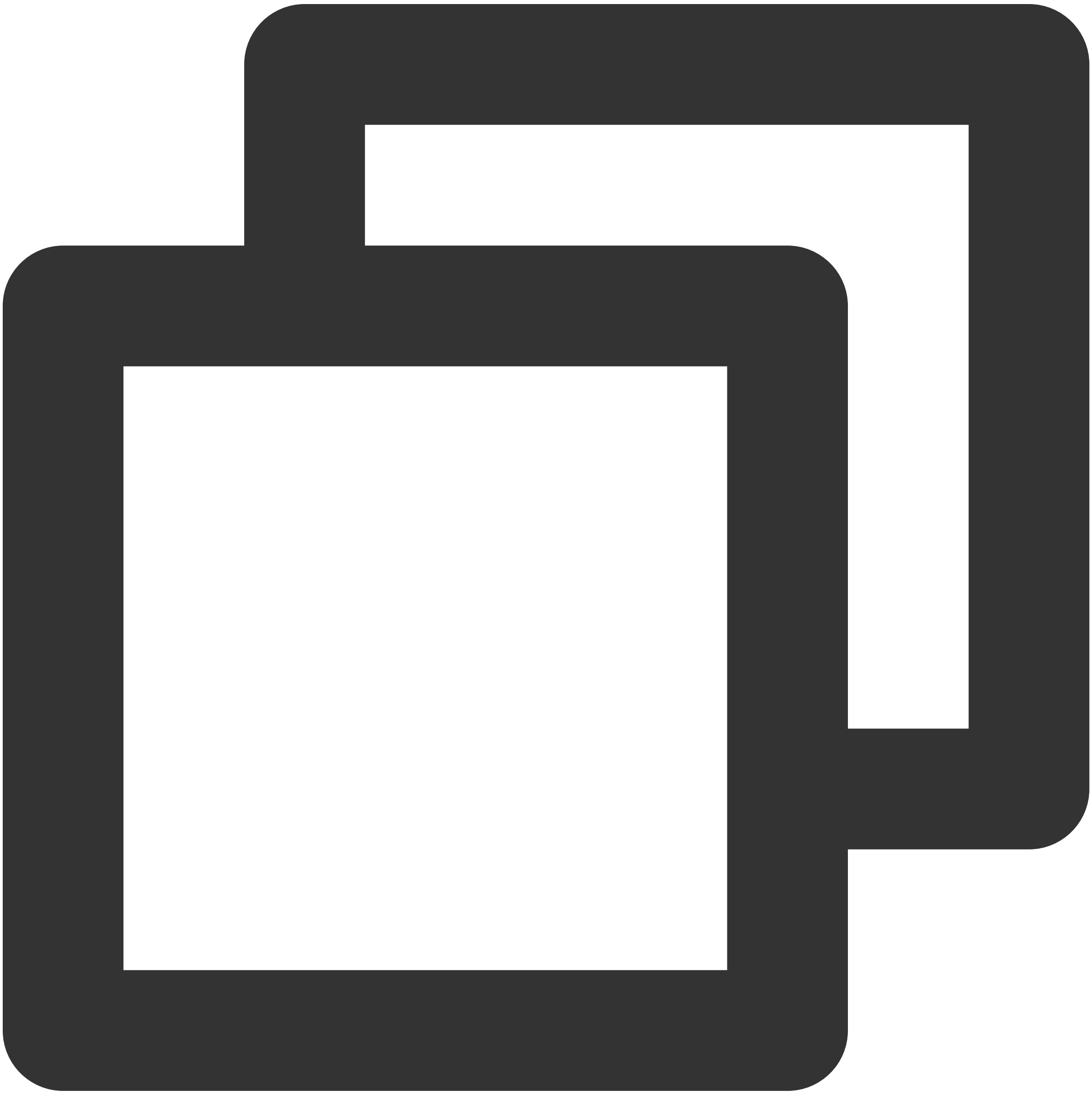
Step 1. Import TRTC SDK

1. Add **TRTC SDK** dependencies to the `app/build.gradle` file.



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

2. Specify the CPU architecture used by the App in the `defaultConfig` file.



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

Once the above configuration completed, clicking `Sync Now` will automatically integrate the SDK into the target project.

Step 2. Configure project

1. Configure the permissions required by the TRTC SDK in the `AndroidManifest.xml` file.

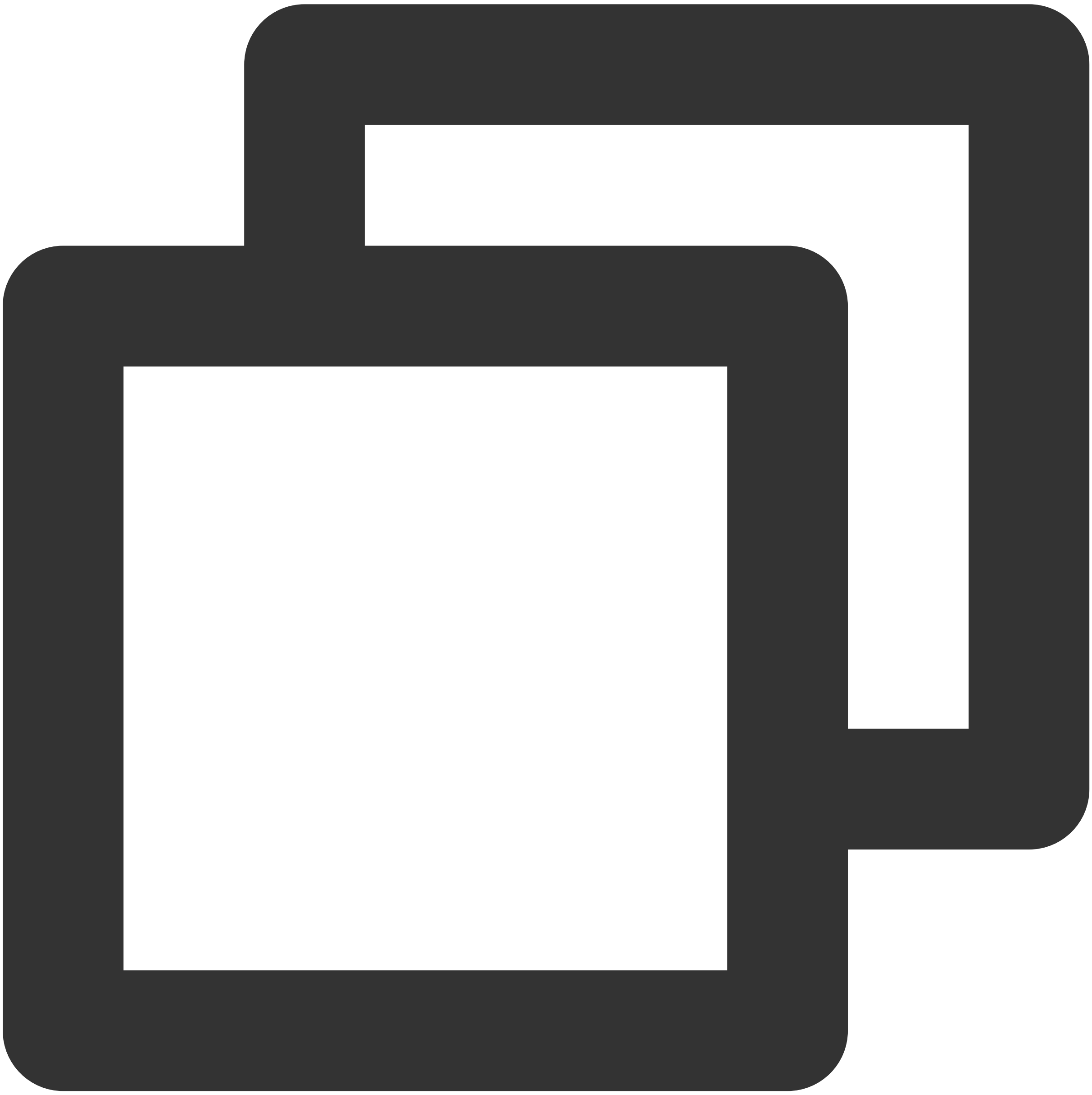
```xml
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission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

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

Note：

Do not set the `android: hardwareAccelerated = "false"` . When hardware acceleration is turned off, the other party's video stream cannot be rendered.

2. In the `proguard-rules.pro` file, add the TRTC SDK-related classes to the "non-obfuscation" list.



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

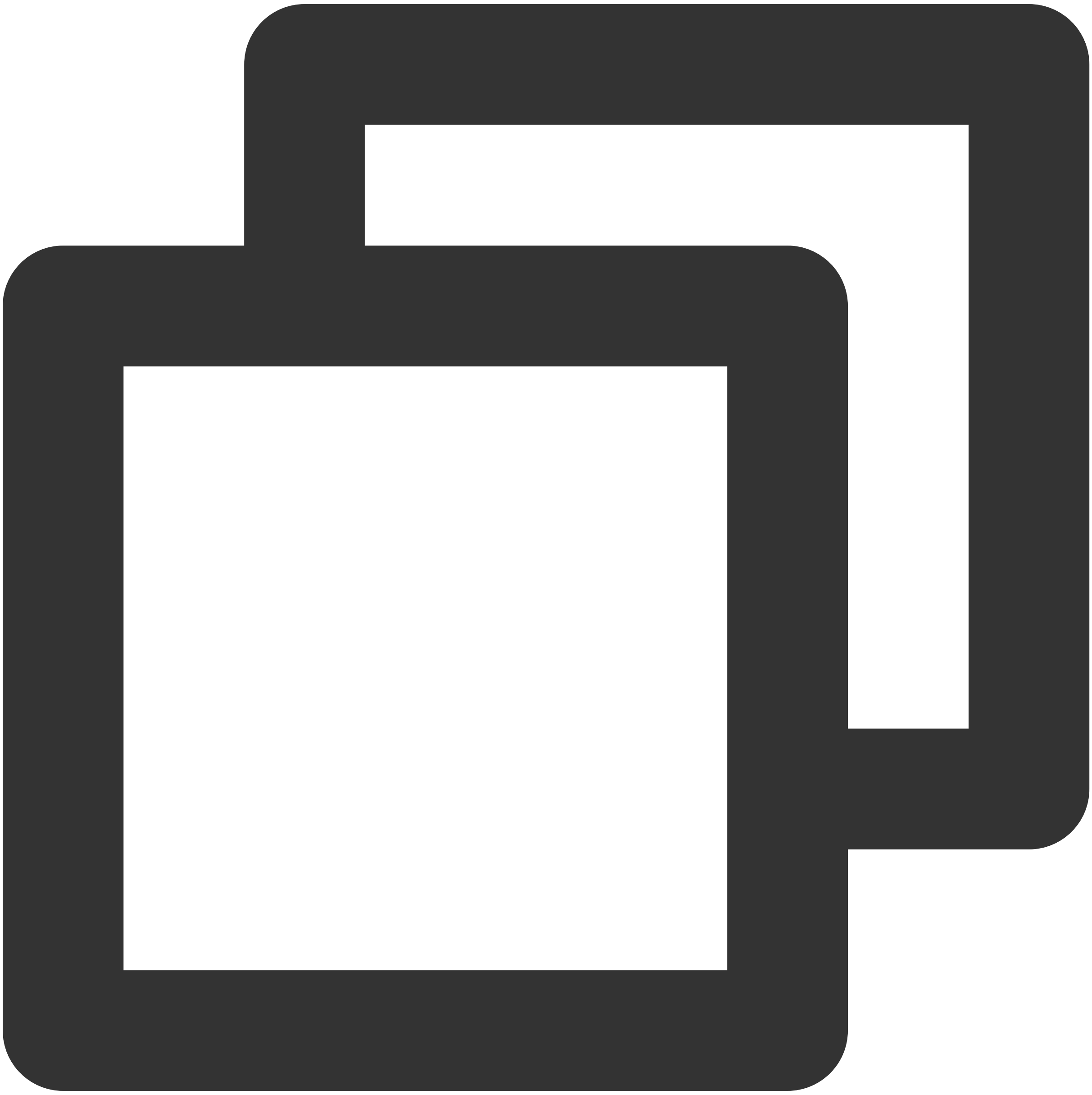
Step 3. Create TRTC instance

1. Declare member variables

```
private final static String TAG = MainActivity.class.getSimpleName();
private static final int REQUEST_CAMERA_AND_MICROPHONE = 1;

private TRTCCloud mCloud; // Declare the mCloud member variable
```

2. Call the initialization interface to create the TRTC instance and set the event callbacks.



```
// Create trtc instance(singleton)  and set up event listeners
mCloud = TRTCCloud.sharedInstance(getApplicationContext());
mCloud.setListener(new TRTCCloudListener() {

    // Listen to the "onError" event, and print logs for errors such as "Camera is
    @Override
    public void onError(int errCode, String errMsg, Bundle extraInfo) {
        super.onError(errCode, errMsg, extraInfo);
        if (errCode == TXLiteAVCode.ERR_CAMERA_NOT_AUTHORIZED) {
            Log.d(TAG, "Current application is not authorized to use the camera");
        }
```

```
    }

    // Listen for the `onEnterRoom` event of the SDK and learn whether the room is
    @Override
    public void onEnterRoom(long result) {
        super.onEnterRoom(result);
        if (result > 0) {
            Log.d(TAG, "Enter room succeed");
        } else {
            Log.d(TAG, "Enter room failed");
        }

    }
});
```

Step 4. Enter the room

1. Request **CAMERA** and **MICROPHONE** permissions.

}

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // ... Other codes

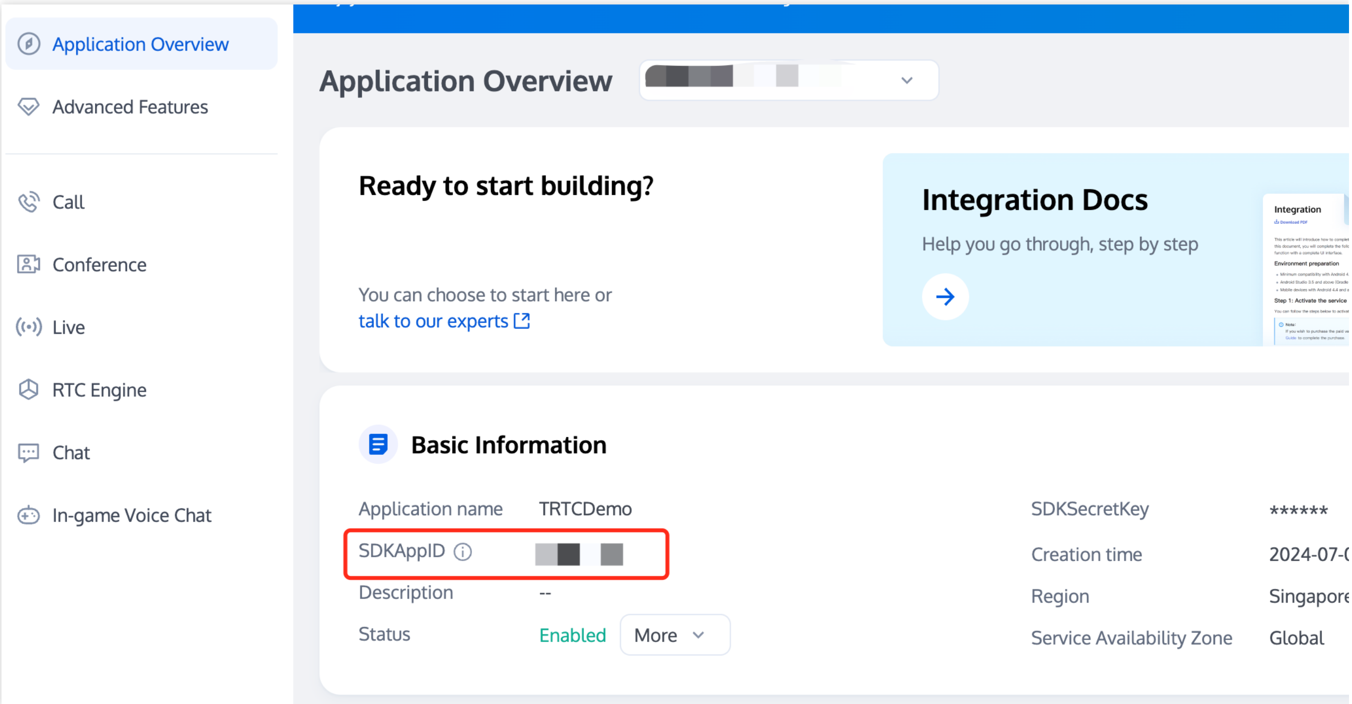
    // Request camera and microphone permissions
    requestCameraAndMicrophonePermission();
}

private void requestCameraAndMicrophonePermission() {
```

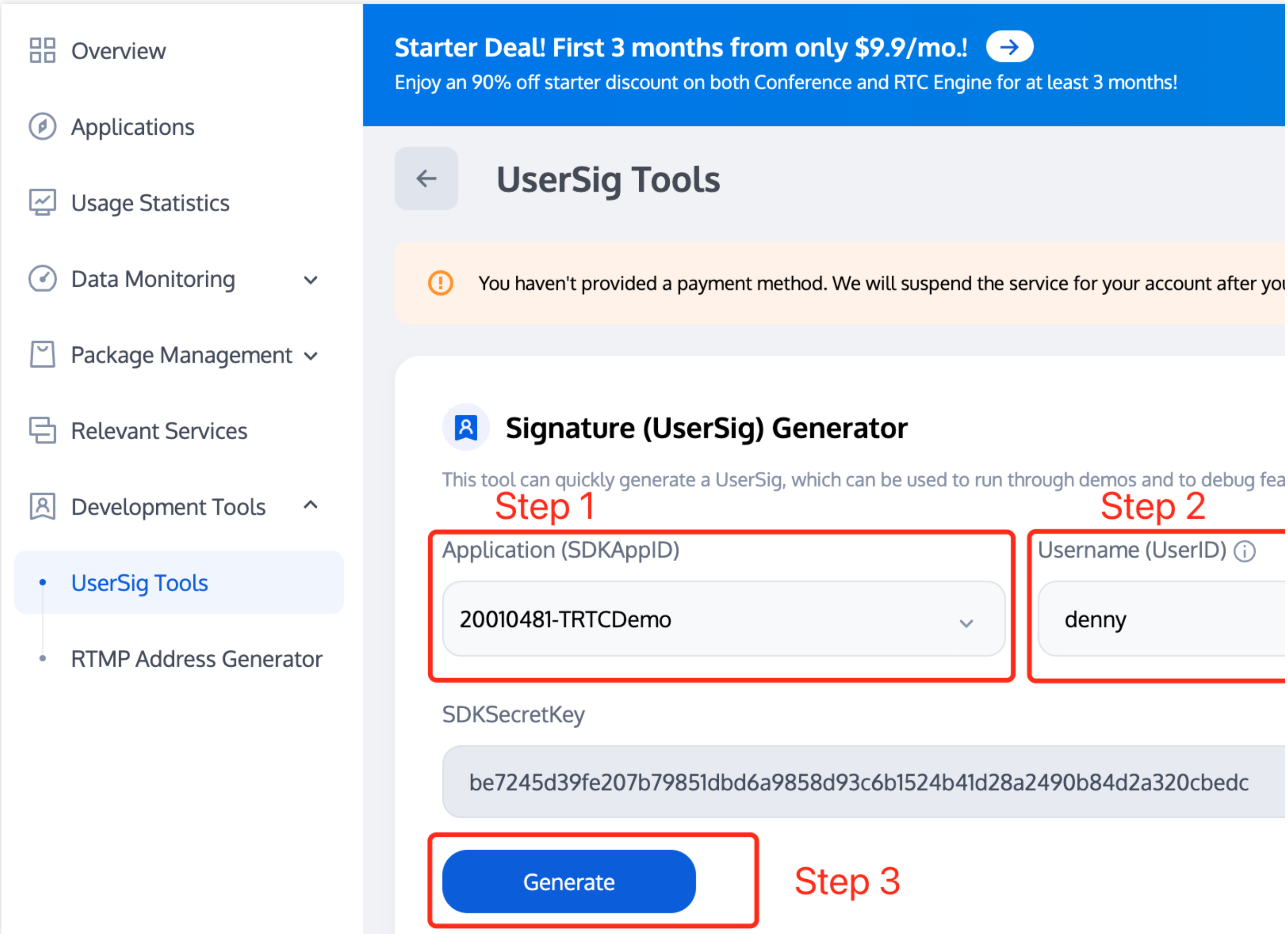
```
    String[] permissions = {android.Manifest.permission.CAMERA, android.Manifest.pe
    ActivityCompat.requestPermissions(this, permissions, REQUEST_CAMERA_AND_MICROPH
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissio
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == REQUEST_CAMERA_AND_MICROPHONE) {
        boolean allPermissionsGranted = true;
        for (int grantResult : grantResults) {
            if (grantResult != PackageManager.PERMISSION_GRANTED) {
                allPermissionsGranted = false;
                break;
            }
        }
        if (allPermissionsGranted) {
            // All permissions are granted, you can start using the camera and micr
            // Here to create TRTC instances, enter the room, publish audio and vid
        } else {
            // Show a message to the user that the permissions are required to use
            Toast.makeText(this, "Camera and Microphone permissions are required",
        }
    }
}
```

2. Click `Create Application` in the Tencent RTC console to get the **SDKAppID** under **Application Overview** tab.

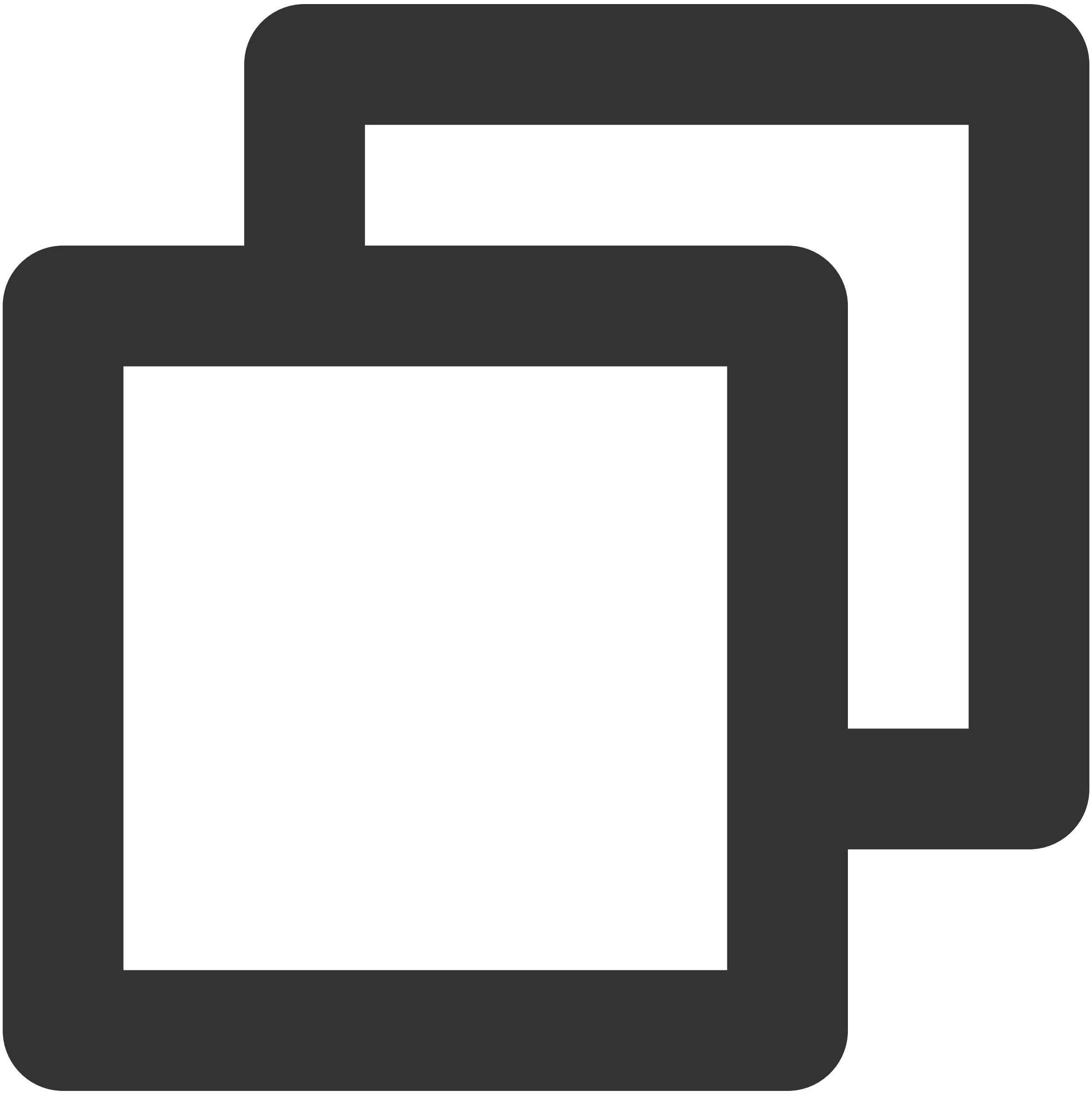


3. Select **SDKAppID** down in the **UserSig Tools**, enter your **UserID**, and click `Generate` to get your own **UserSig**.



4. After setting the **TRTCParams** for room entry, call the `enterRoom` to enter the room.

As an Anchor：



```
// Please replace each field in TRTCParams with your own parameters
TRTCCloudDef.TRTCParams trtcParams = new TRTCCloudDef.TRTCParams();
trtcParams.sdkAppId = 1400000123;  // Please replace with your own SDKAppID
trtcParams.userId = "denny";       // Please replace with your own userid
trtcParams.roomId = 123321;        // Please replace with your own room number
trtcParams.userSig = "xxx";        // Please replace with your own userSig
trtcParams.role = TRTCCloudDef.TRTCRoleAnchor;

// If your application scenario is a video call between several people, please use
mCloud.enterRoom(trtcParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
```

As an audience：

```
// Please replace each field in TRTCParams with your own parameters
TRTCCloudDef.TRTCParams trtcParams = new TRTCCloudDef.TRTCParams();
trtcParams.sdkAppId = 1400000123;  // Please replace with your own SDKAppID
trtcParams.userId = "denny";       // Please replace with your own userid
trtcParams.roomId = 123321;        // Please replace with your own room number
trtcParams.userSig = "xxx";        // Please replace with your own userSig
trtcParams.role = TRTCCloudDef.TRTCRoleAudience;

// If your application scenario is a video call between several people, please use
```

```
mCloud.enterRoom(trtcParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
```

Note：

If you enter the room as an **audience**, **sdkAppId** and **roomId** need to be the same as on the anchor side, while **userId** and **userSig** need to be replaced with your own values.

Step 5. Turn on Camera

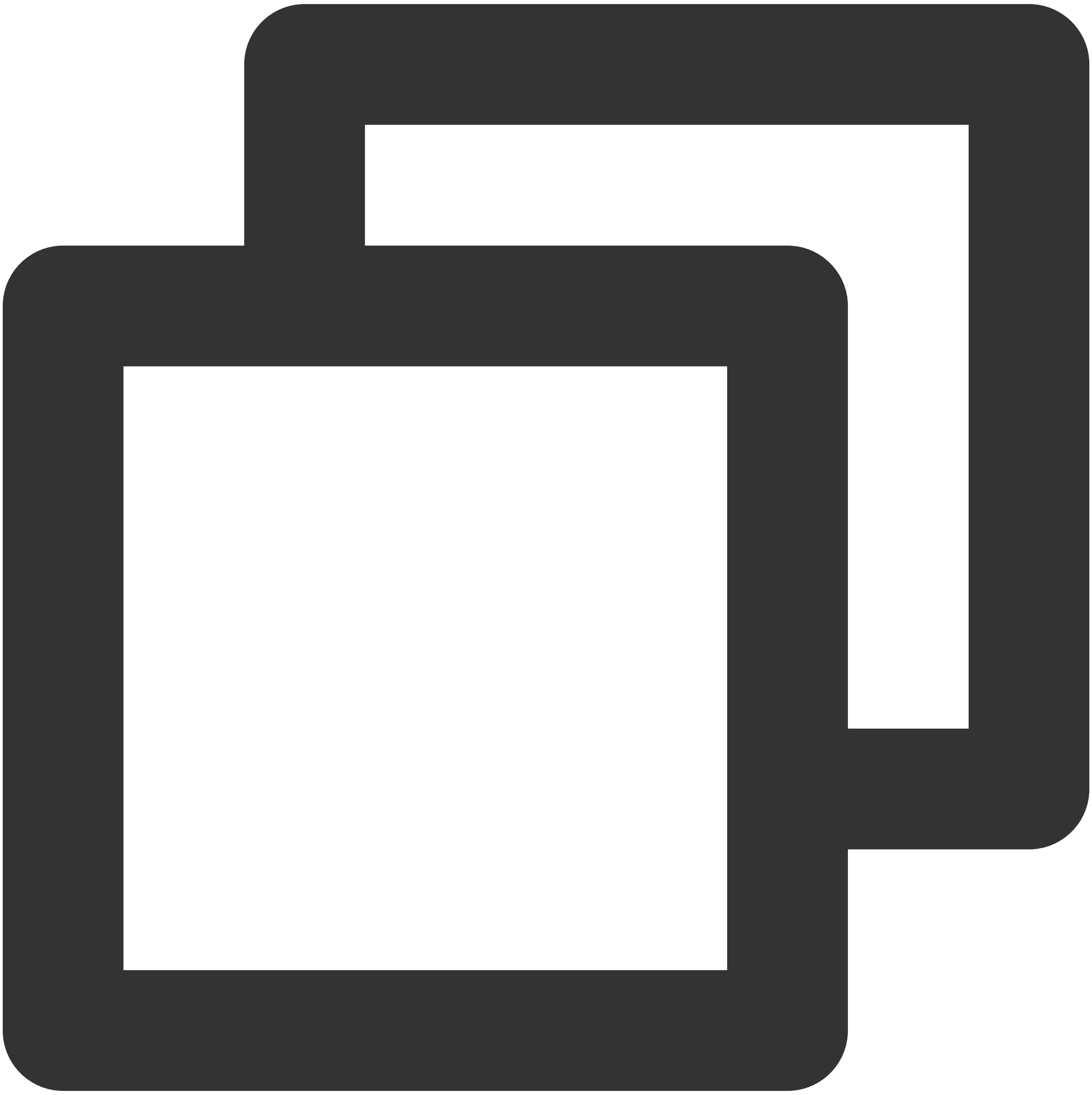
1. Add in the corresponding **.xml** file, as shown in the following code.


```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.an
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.tencent.rtmp.ui.TXCloudVideoView
        android:id="@+id/txcvv_main_local"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

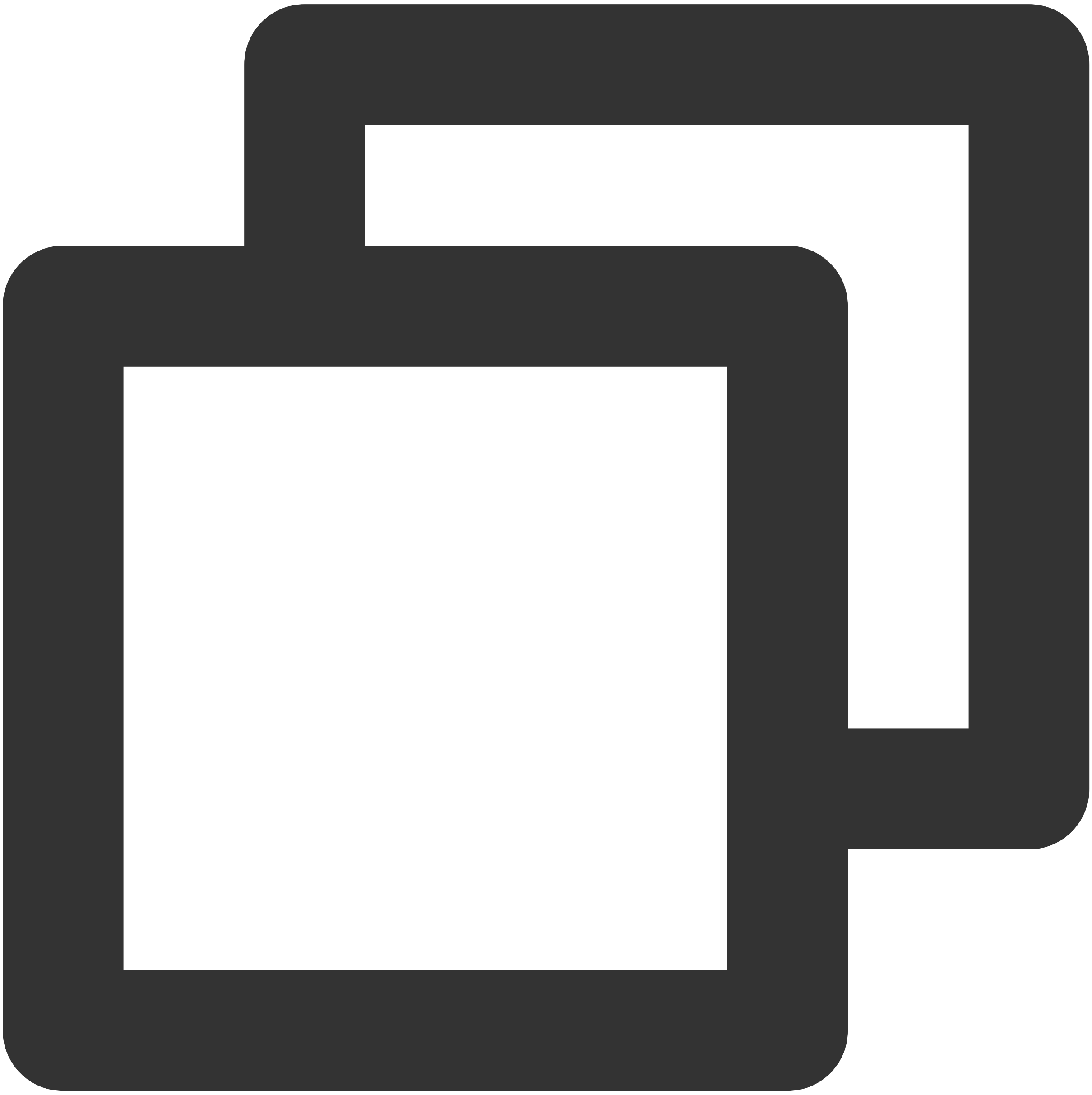
2. Before calling the `startLocalPreview` to open the camera preview, set the **TRTCRenderParams** of the local preview by calling the `setLocalRenderParams` .



```
// Set the preview mode of the local screen
TRTCCloudDef.TRTCRenderParams trtcRenderParams = new TRTCCloudDef.TRTCRenderParams(
trtcRenderParams.fillMode    = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL;
trtcRenderParams.mirrorType = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_AUTO;
mCloud.setLocalRenderParams(trtcRenderParams);

// Start a preview of the local camera
TXCloudVideoView cameraVideo = findViewById(R.id.txcvv_main_local);
mCloud.startLocalPreview(true, cameraVideo);
```

3. Call the `TXDeviceManager` to perform operations such as **Switching between front and rear cameras**, **Setting Focus Mode**, and **Enabling**.



```
// Enable auto focus and turn on the flash using the TXDeviceManager
TXDeviceManager manager = mCloud.getDeviceManager();
if (manager.isAutoFocusEnabled()) {
    manager.enableCameraAutoFocus(true);
}
manager.enableCameraTorch(true);
```

Note：

The front camera is turned on by default. If you need to use the rear camera, call `manager.switchCamera(false)` to turn on the rear camera.

Step 6. Turn on microphone

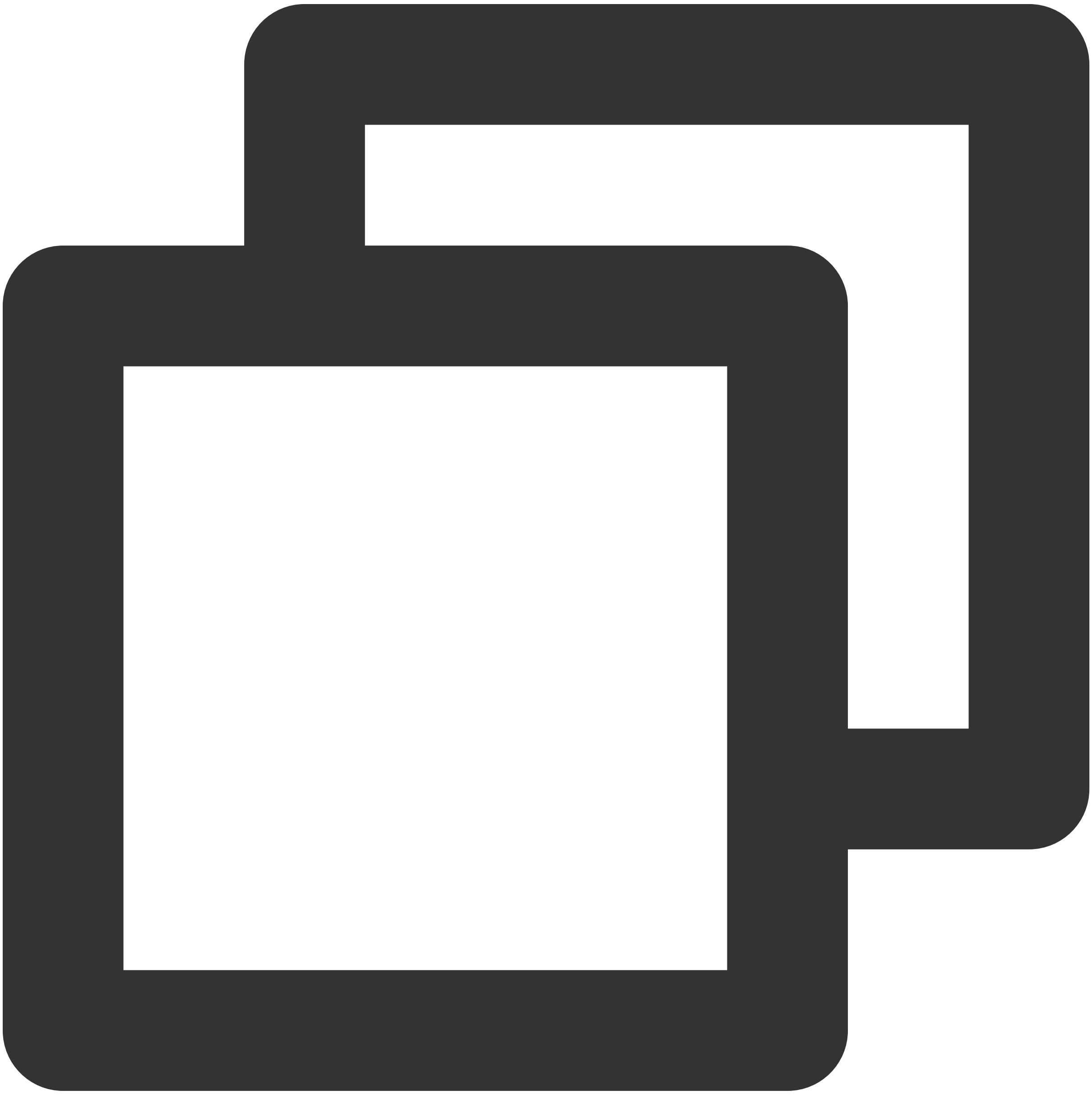
Call `startLocalAudio` to enable microphone capture. This interface requires you to determine the capture mode by the `quality` parameter. It is recommended to **select one of the following modes that is suitable for your project according to your needs**.


```
// Enable microphone acquisition and set the current scene to: Voice mode
// For high noise suppression capability, strong and weak network resistance
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_SPEECH );

// Enable microphone acquisition, and set the current scene to: Music mode (
// For high fidelity acquisition, low sound quality loss, recommended to use with p
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
```

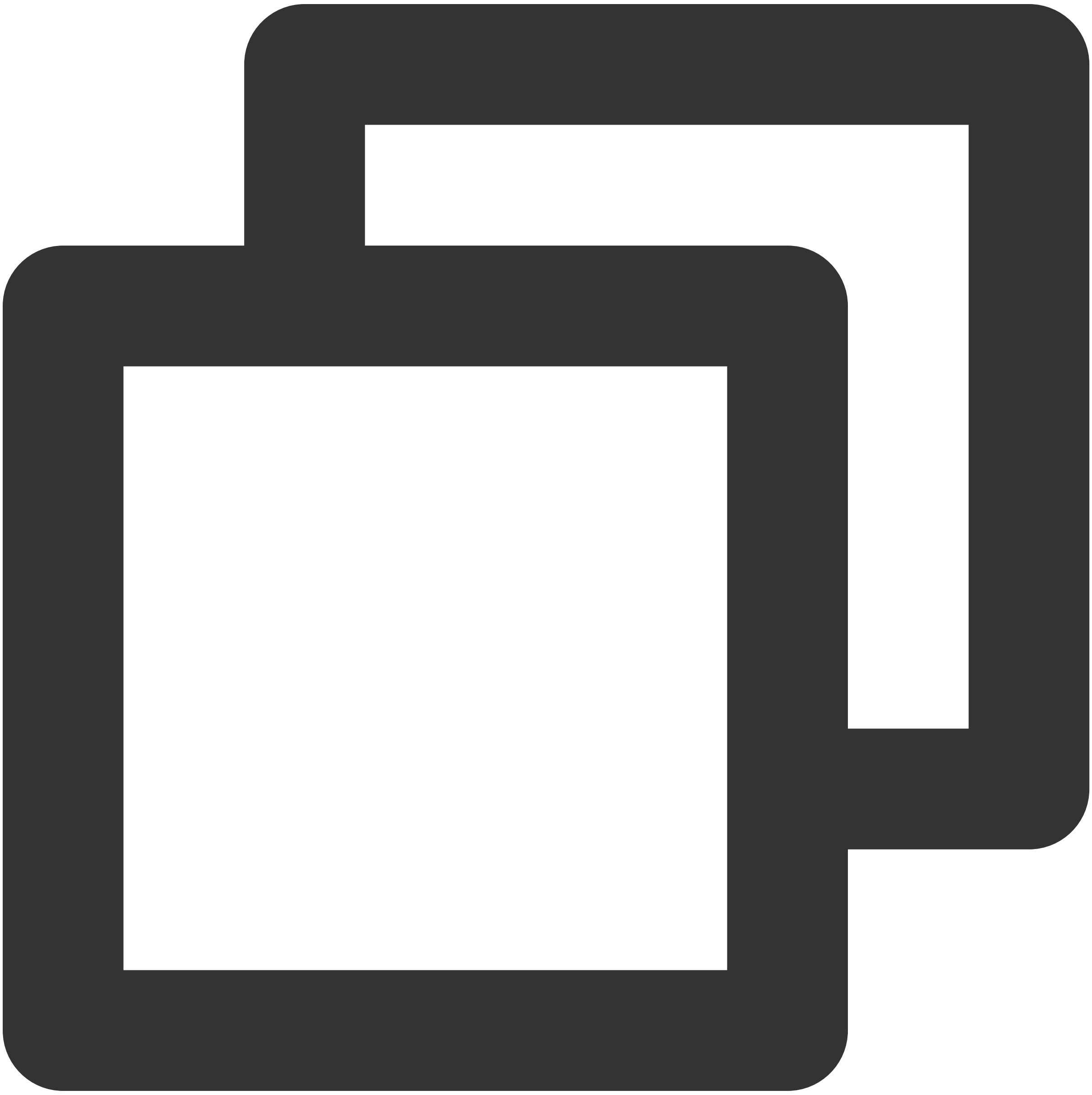
Step 7. Play/stop video streaming

After you enter denny's room as an audience by following steps 1-4 to create a new project, you can play a video of the remote user by calling the `startRemoteView` .



```
// Play denny's camera footage
mCloud.startRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, cameraView
```

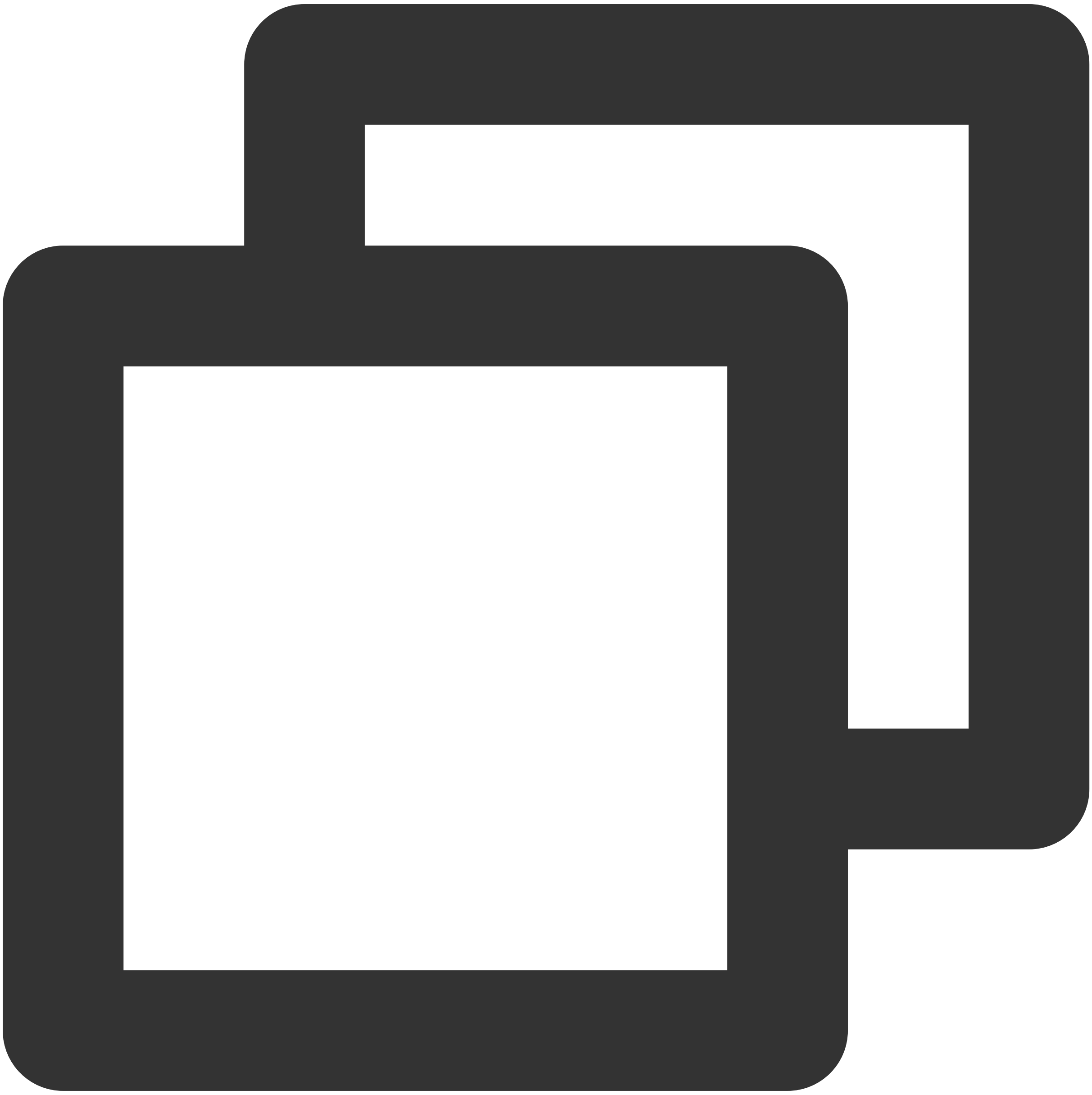
Then, you can call the `stopRemoteView` to stop the videos of a remote user. Alternatively, you can also stop the videos of all remote users via the `stopAllRemoteView` .



```
// Stop denny's camera footage
mCloud.stopRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, cameraView);
// Stop all camera footages
mCloud.stopAllRemoteView();
```

Step 8. Play/stop the audio stream

Mute the voice of remote user denny by calling the `muteRemoteAudio("denny", true)` .



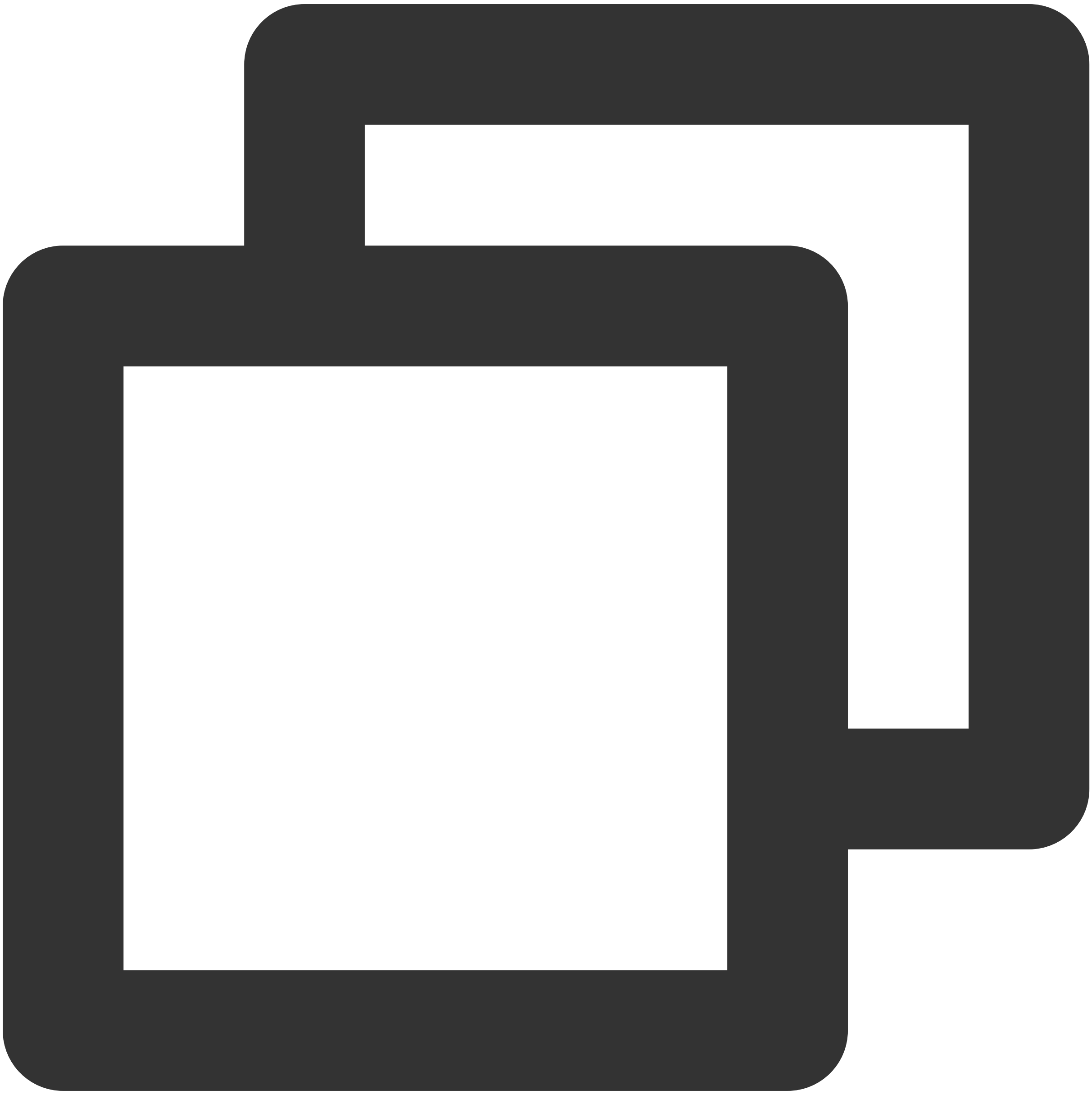
```
// Mute user with id denny
mCloud.muteRemoteAudio("denny", true);
```

You can also unmute him later by calling the `muteRemoteAudio("denny", false)` .

```
// Unmute user with id denny
mCloud.muteRemoteAudio("denny", false);
```

Step 9. Exit the room

Call the `exitRoom` to exit the current room, the SDK will notify you after the check-out through the `onExitRoom(int reason)` callback event.



```
// Exit current room
mCloud.exitRoom();

// Listen for the `onExitRoom` callback to get the reason for room exit
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        Log.d(TAG, "Kicked out of the current room by server through the restful ap
    } else if (reason == 2) {
```



```
            Log.d(TAG, "Current room is dissolved by server through the restful api..."
        }
    }
```

FAQs

API Reference at API Reference.

If you encounter any issues during integration and use, please refer to Frequently Asked Questions.

Contact us

If you have any suggestions or feedback, please contact `info_rtc@tencent.com` .

iOS

Last updated：2024-07-18 15:20:21

This tutorial mainly introduces how to implement a basic audio and video call.

Prerequisites

Xcode 9.0 or later

iPhone or iPad with iOS 9.0 or later

A valid developer signature for your project

Step 1. Import TRTC SDK

1. Run the following command in the terminal window to install CocoaPods. If you have installed CocoaPods, skip this step.

```
sudo gem install cocoapods
```

2. After going to the **TRTCDemo** root directory, enter the following command to create the **Podfile** file for your project.

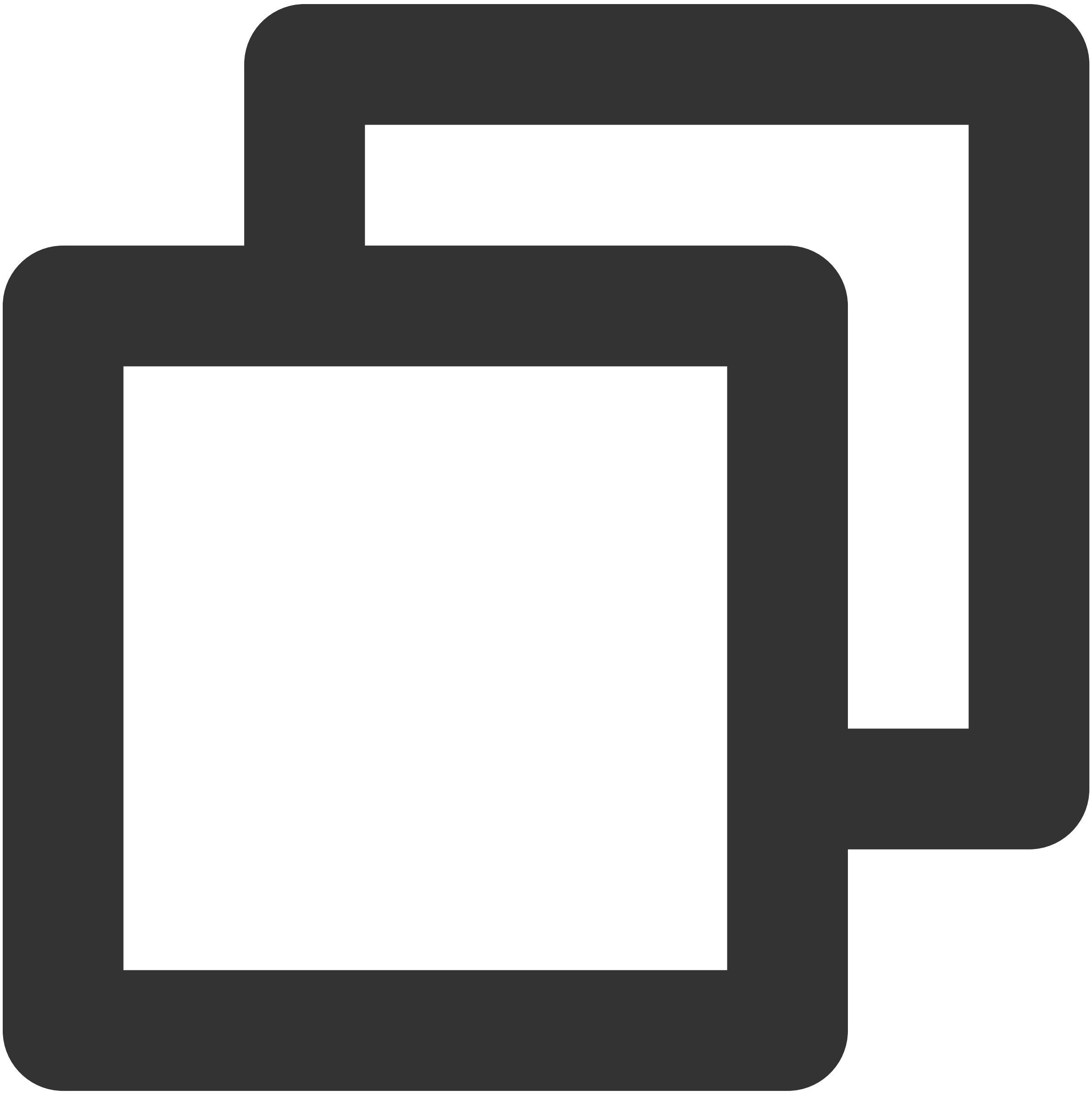
```
pod init
```

3. Edit the **Podfile** file as follows and change the **App** to the name of your own project.

```
platform :ios, '8.0'

target 'App' do
pod 'TXLiteAVSDK_TRTC', :podspec => 'https://liteav.sdk.qcloud.com/pod/liteavsdkspe
end
```

4. Enter the following command to update the local library file and install the SDK.



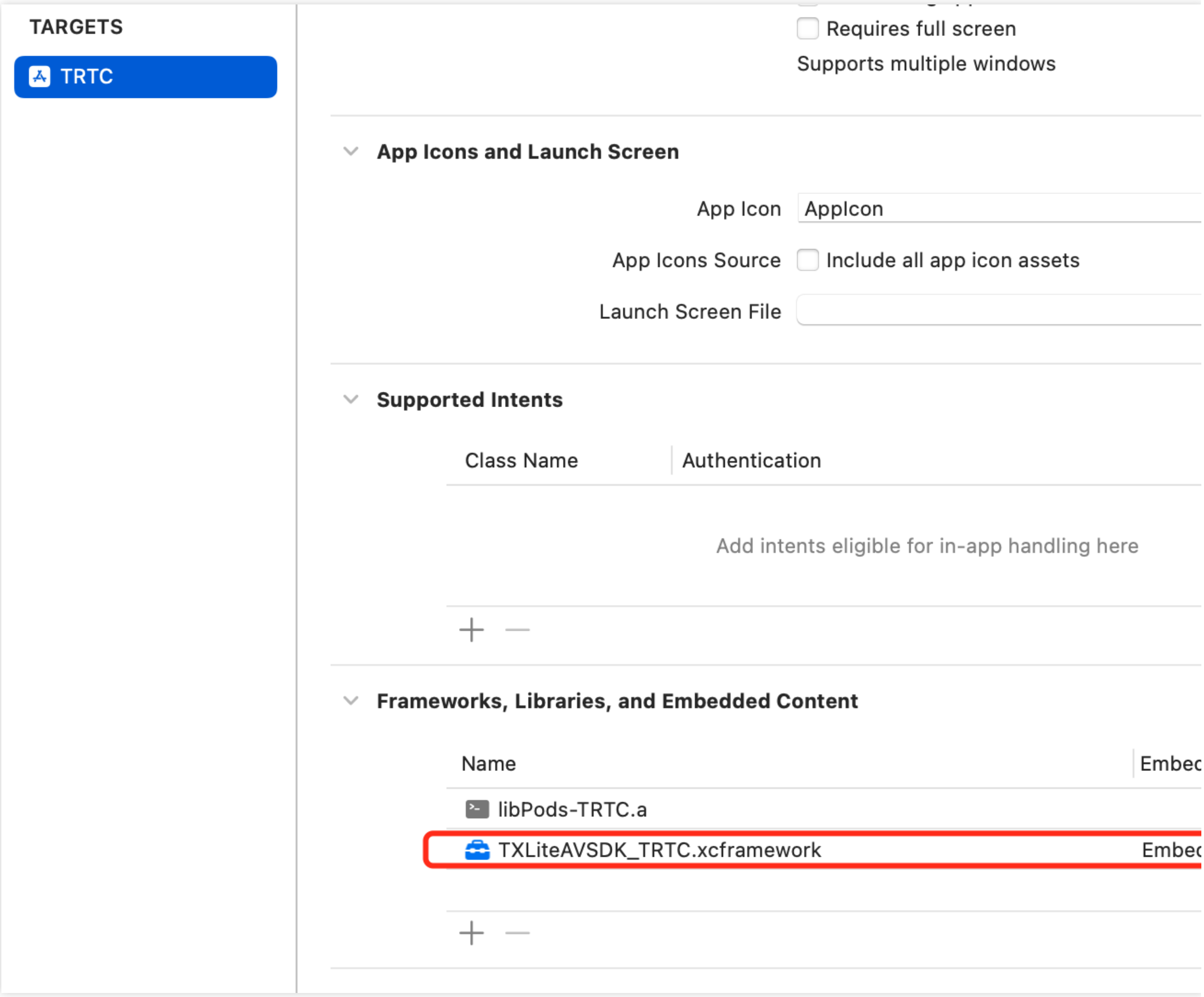
```
pod install
```

Note：

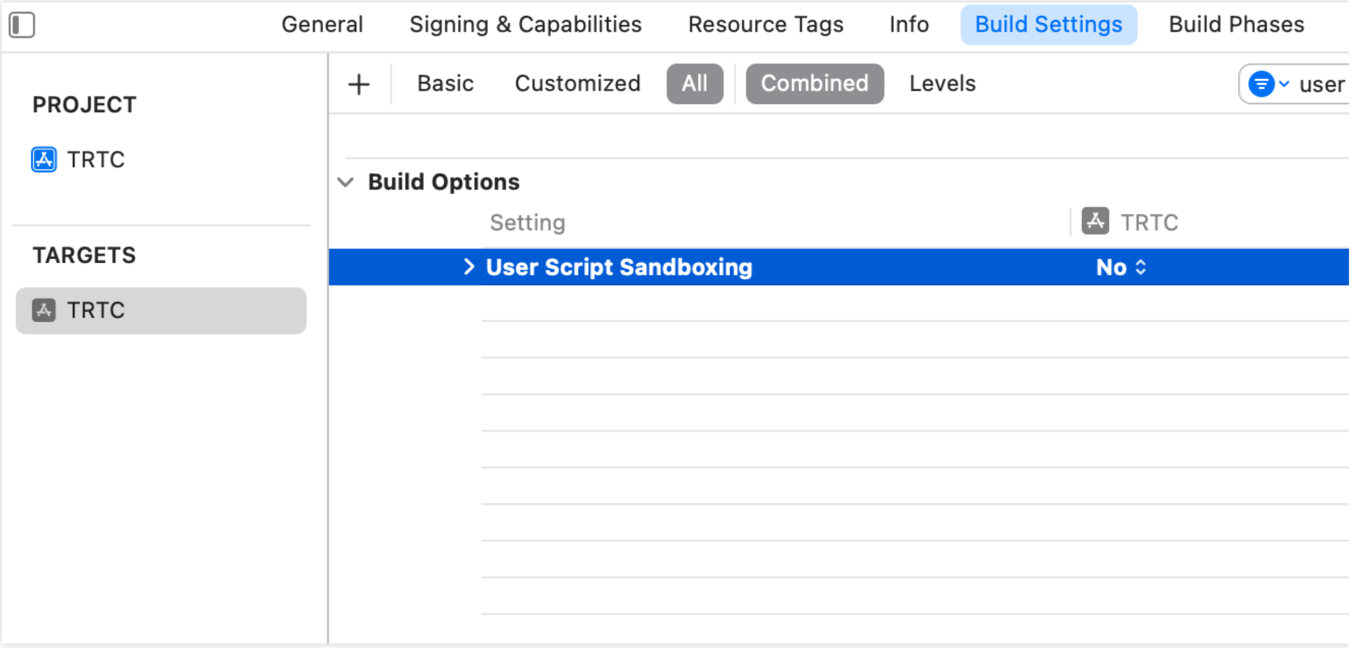
After the pod command is executed, a project file with the **.xcworkspace** suffix integrated with the SDK is generated. Double-click the **.xcworkspace** file to open it.

Step 2. Configure project

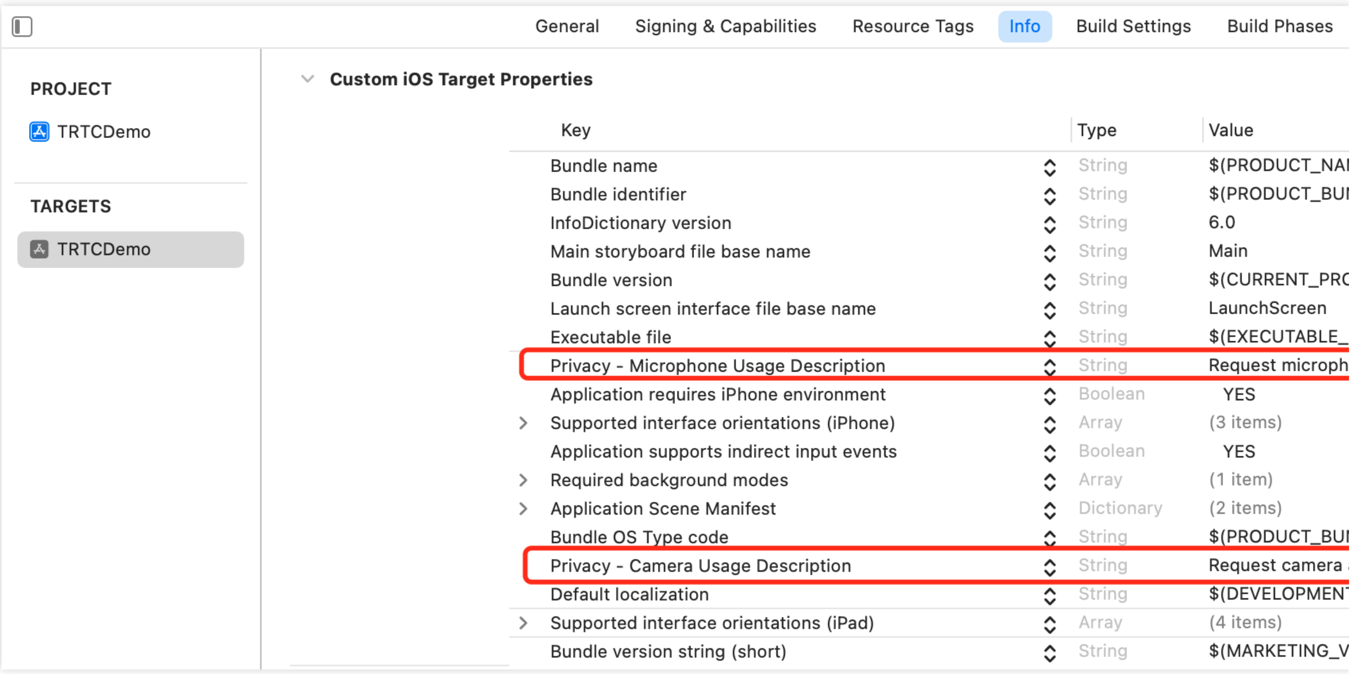
1. After opening the **.xcworkspace** file, add **TXLiteAVSDK_TRTC.xcframework** to the **Frameworks, Libraries, and Embedded Content** section in **General** tab.



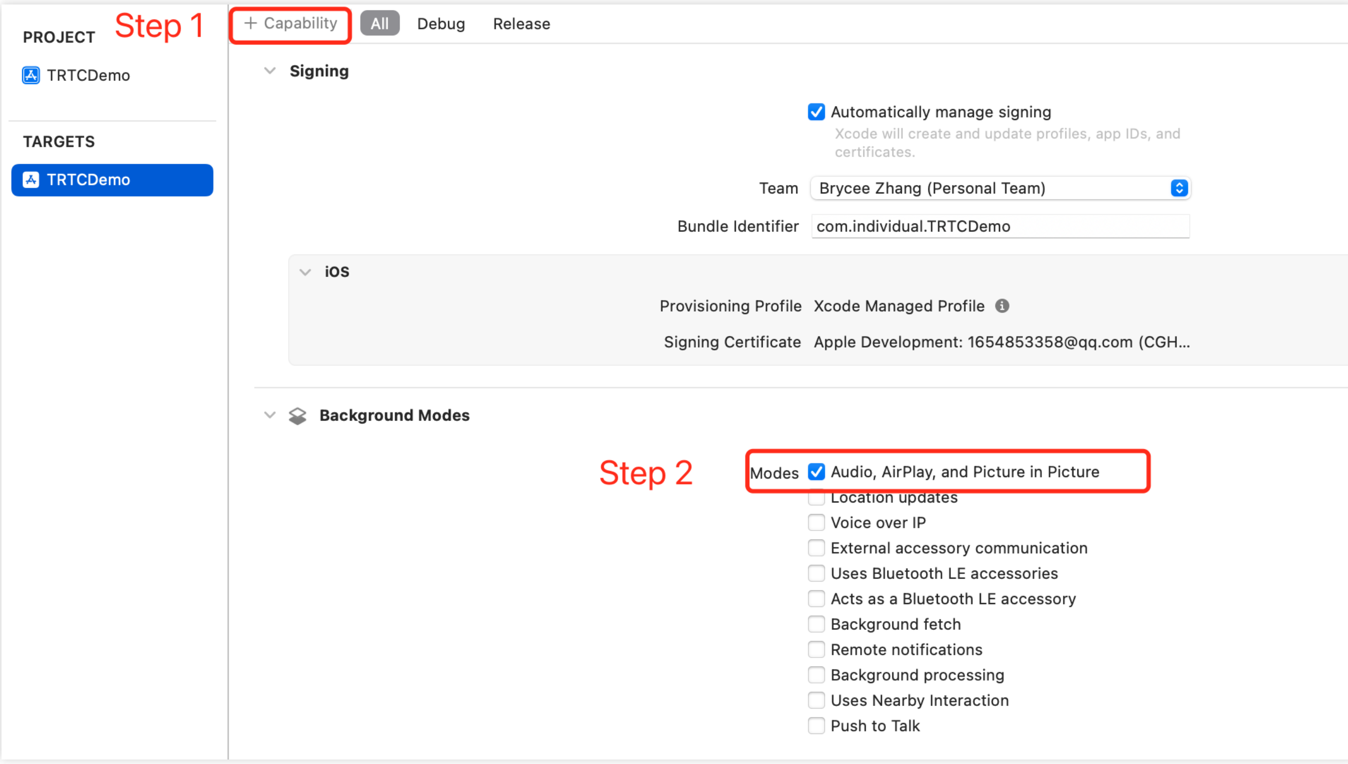
2. Search for **User Script Sandboxing** in **Build Settings** tab and set its value to **No**.



3. Add **Privacy-Microphone Usage Description** and **Privacy-Microphone Usage Description** to **Info.plist** tab, and fill in the target prompt words used by the Microphone/Camera to obtain the permissions to use the microphone and camera.

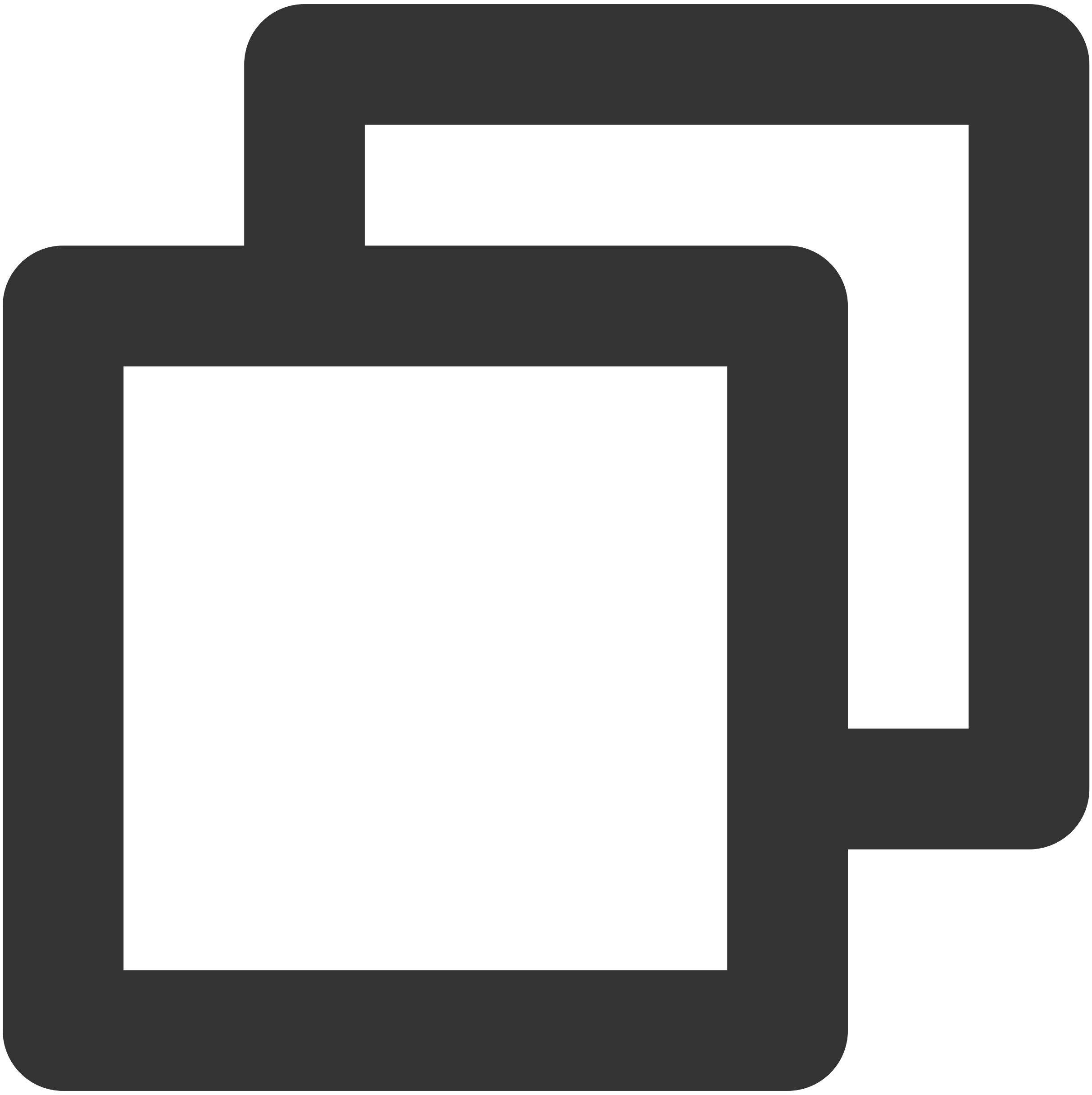


4. Add **Background Modes** to the **Signing & Capabilities** tab and check **Audio, AirPlay and Picture in Picture**.



Step 3. Import Module

Add a module reference to the SDK in the **AppDelegate.h** file:



```
@import TXLiteAVSDK_TRTC;
```

Step 4. Create TRTC instance

1. Add the following properties to the **AppDelegate.h** file and declare the `toastTip:` method.

```
#import <UIKit/UIKit.h>
@import TXLiteAVSDK_TRTC;

@interface AppDelegate : UIResponder <UIApplicationDelegate, TRTCCloudDelegate>

@property (strong, nonatomic) UIWindow *window; // Add the window property
@property (nonatomic, strong) TRTCCloud *trtcCloud; // Add the trtcCloud property
@property (nonatomic, strong) UIView *localCameraVideoView; // Add the localCameraV

- (void)toastTip:(NSString *)tip; // Declare the toastTip: method
@end
```

2. Implement the `toastTip:` method in the **AppDelegate.m** file.

```
// Implement the toastTip: method
- (void)toastTip:(NSString *)tip {
    UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"Tip: "
    UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"Confirm: " style:UIA
    [alert addAction:okAction];
    [self.window.rootViewController presentViewController:alert animated:YES comple
}
```


3. Call the interface to create a TRTC instance in the `didFinishLaunchingWithOptions()` , and set up the event callbacks.

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    // Override point for customization after application launch.

    // Create trtc instance(singleton)  and set up event listeners
    _trtcCloud = [TRTCCloud sharedInstance];
    _trtcCloud.delegate = self;

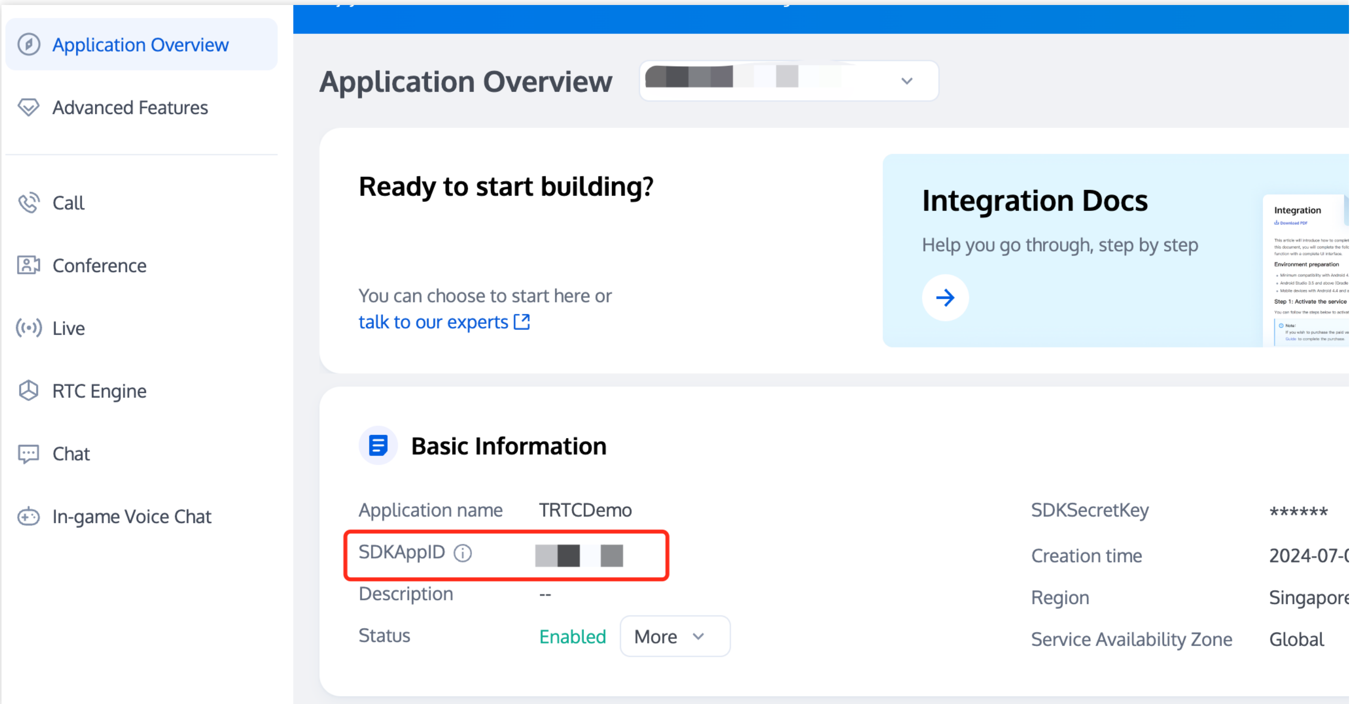
    return YES;
```

```
}

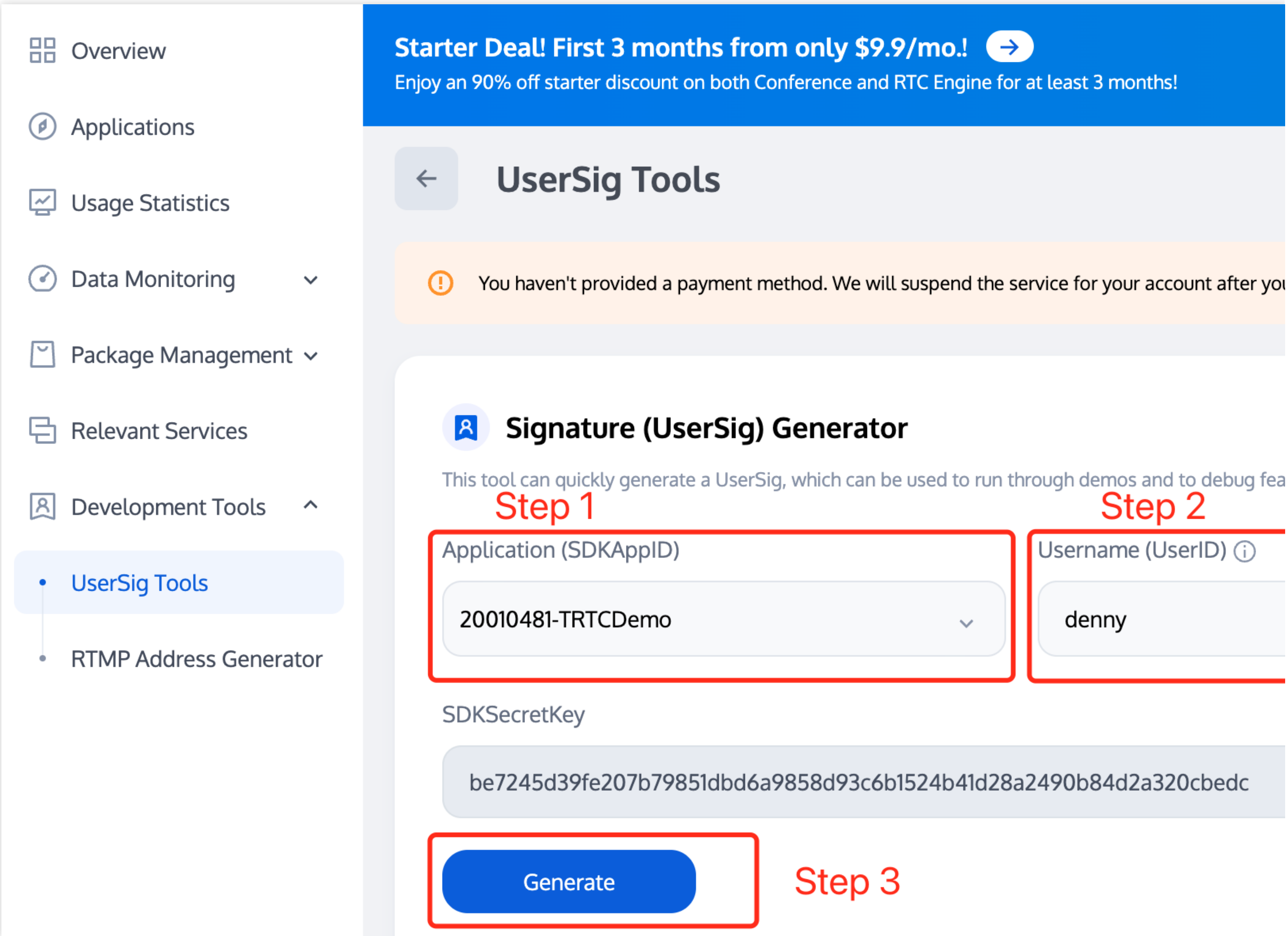
// Listen to the "onError" event, and print logs for errors such as "Camera is not
- (void)onError:(TXLiteAVError)errCode
         errMsg:(nullable NSString *)errMsg
        extInfo:(nullable NSDictionary *)extInfo{
    if (ERR_CAMERA_NOT_AUTHORIZED == errCode) {
        NSString *errorInfo = @"Current application is not authorized to use the ca
        errorInfo = [errorInfo stringByAppendingString : errMsg];
        [self toastTip:errorInfo];
    }
}
```

Step 5. Enter the room

1. Click `Create Application` in the Tencent RTC console to get the **SDKAppID** under **Application Overview**.



2. Select **SDKAppID** down in the **UserSig Tools**, enter your **UserID**, and click `Generate` to get your own **UserSig**.



3. After setting the **TRTCParams** for room entry, call the `enterRoom` to enter the room.

As an Anchor：


```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    // Override point for customization after application launch.

    // ...Other codes

    // Please replace each field in TRTCParams with your own parameters
    TRTCParams *trtcParams = [[TRTCParams alloc] init];
    trtcParams.sdkAppId = 1400000123;  // Please replace with your own SDKAppID
    trtcParams.roomId = 123321; // Please replace with your own room number
    trtcParams.userId = @"denny";   // Please replace with your own userid
    trtcParams.userSig = @""; // Please replace with your own userSig
```



```
    trtcParams.role = TRTCRoleAnchor;

    // If your application scenario is a video call between several people, please
    [self.trtcCloud enterRoom:trtcParams appScene:TRTCAppSceneLIVE];

    return YES;
}

// Listen for the `onEnterRoom` event of the SDK and learn whether the room is succ
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        [self toastTip:@"Enter room succeed!"];
    } else {
        [self toastTip:@"Enter room failed!"];
    }
}
```

As an audience:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    // Override point for customization after application launch.

    // ...Other codes

    // Please replace each field in TRTCParams with your own parameters
    TRTCParams *trtcParams = [[TRTCParams alloc] init];
    trtcParams.sdkAppId = 1400000123;  // Please replace with your own SDKAppID
    trtcParams.roomId = 123321; // Please replace with your own room number
    trtcParams.userId = @"denny";   // Please replace with your own userid
    trtcParams.userSig = @""; // Please replace with your own userSig
```



```
    trtcParams.role = TRTCRoleAudience;

    // If your application scenario is a video call between several people, please
    [self.trtcCloud enterRoom:trtcParams appScene:TRTCAppSceneLIVE];

    return YES;
}

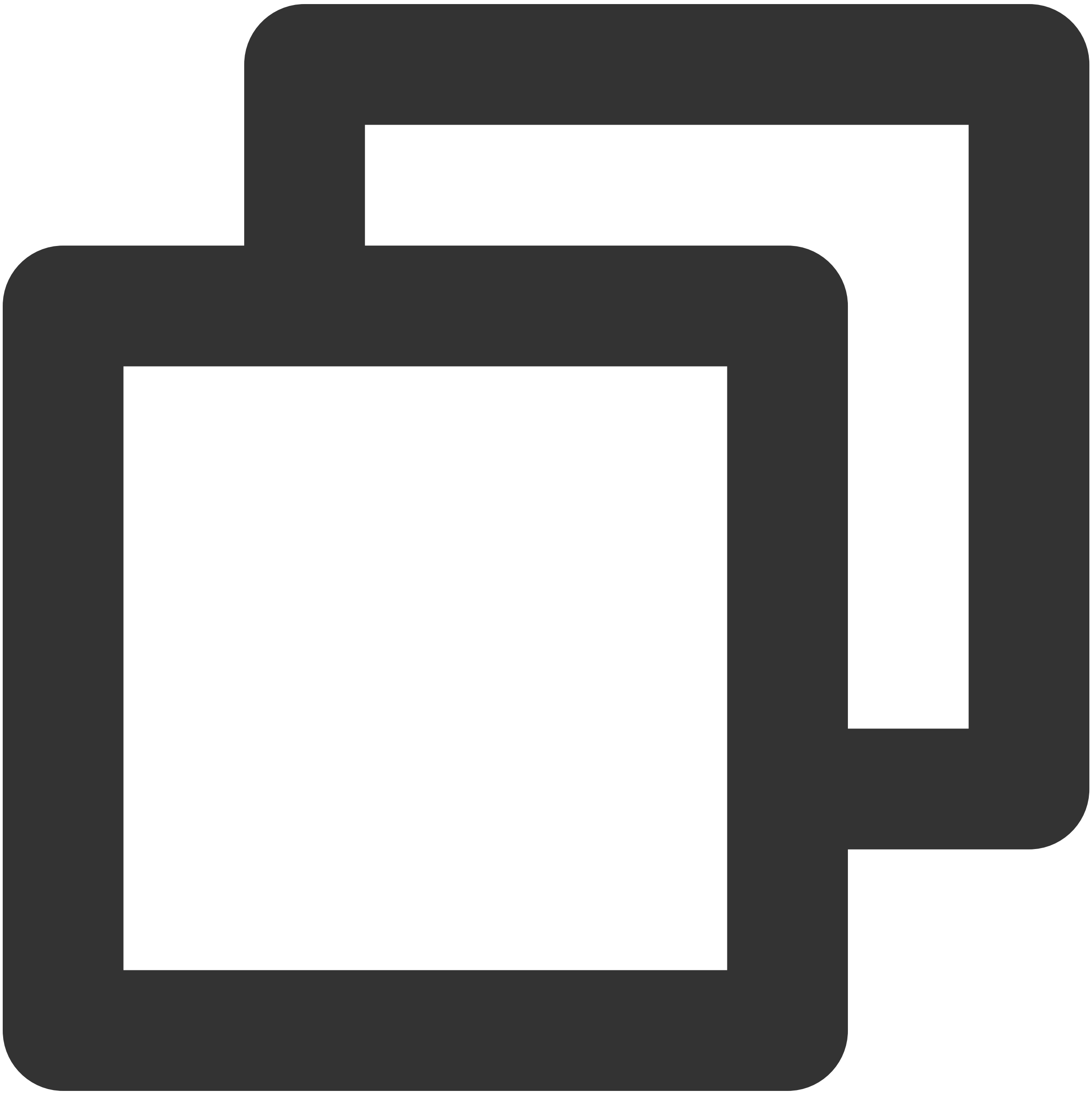
// Listen for the `onEnterRoom` event of the SDK and learn whether the room is succ
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        [self toastTip:@"Enter room succeed!"];
    } else {
        [self toastTip:@"Enter room failed!"];
    }
}
```

Note：

If you enter the room as an **audience**, **sdkAppId** and **roomId** need to be the same as on the anchor side, while **userId** and **userSig** need to be replaced with your own values.

Step 6. Turn on Camera

1. Initialize **localCameraVideoView** in `didFinishLaunchingWithOptions()` method, and call the `setLocalRenderParams` to set the local preview render parameters.



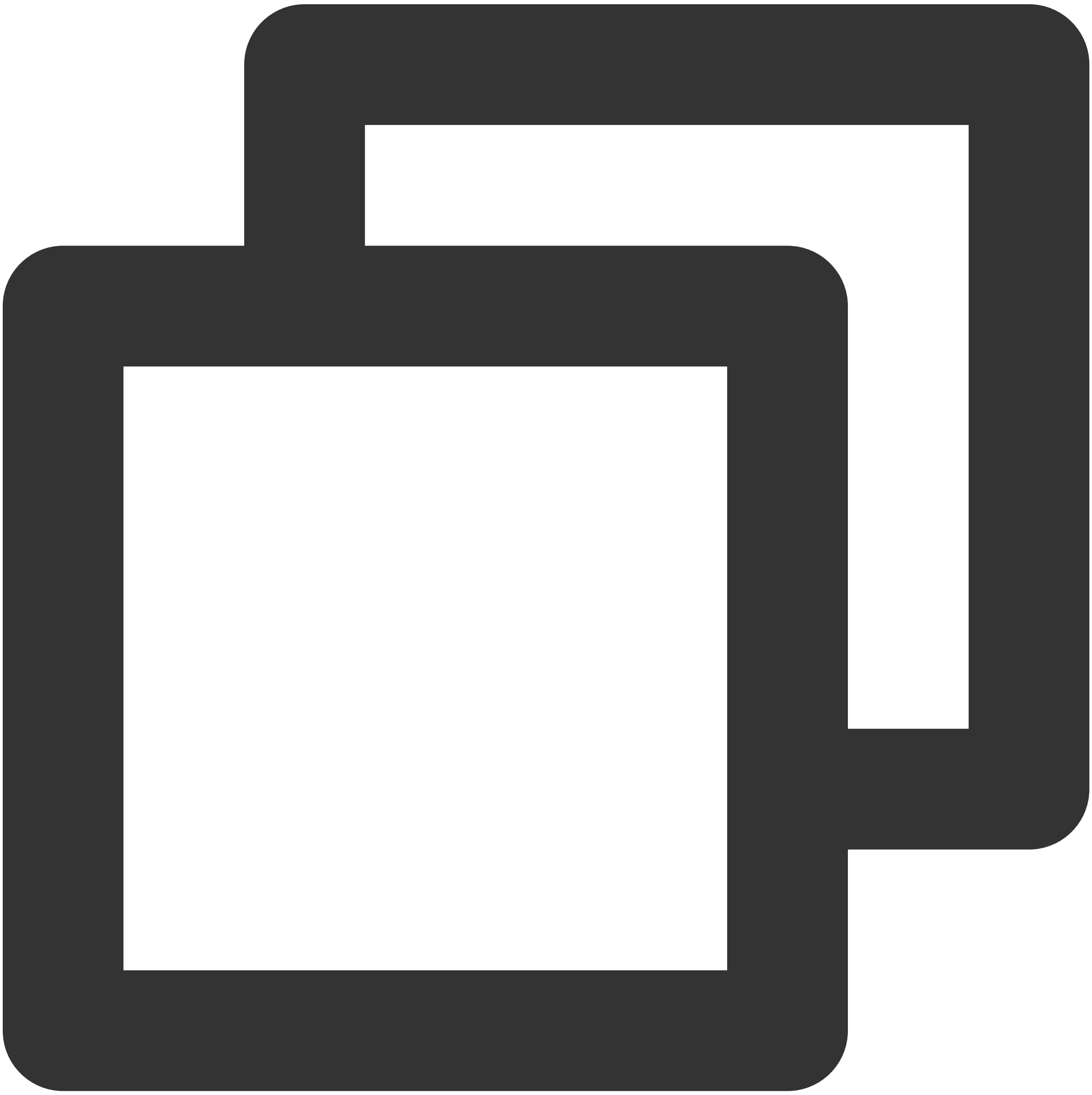
```
// Initialize localCameraVideoView
self.localCameraVideoView = [[UIView alloc] initWithFrame:self.window.bounds];
self.localCameraVideoView.backgroundColor = [UIColor blackColor];
[self.window addSubview:self.localCameraVideoView];

// Set the preview mode of the local screen
TRTCRenderParams *trtcRenderParams = [[TRTCRenderParams alloc] init];
trtcRenderParams.fillMode   = TRTCVideoFillMode_Fill;
trtcRenderParams.mirrorType = TRTCVideoMirrorTypeAuto;
[self.trtcCloud setLocalRenderParams:trtcRenderParams];
```



```
// Start a preview of the local camera
[self.trtcCloud startLocalPreview:YES view:self.localCameraVideoView];
```

2. Call the `TXDeviceManager` to perform operations such as **Switching between front and rear cameras**, **Setting Focus Mode**, and **Enabling**.

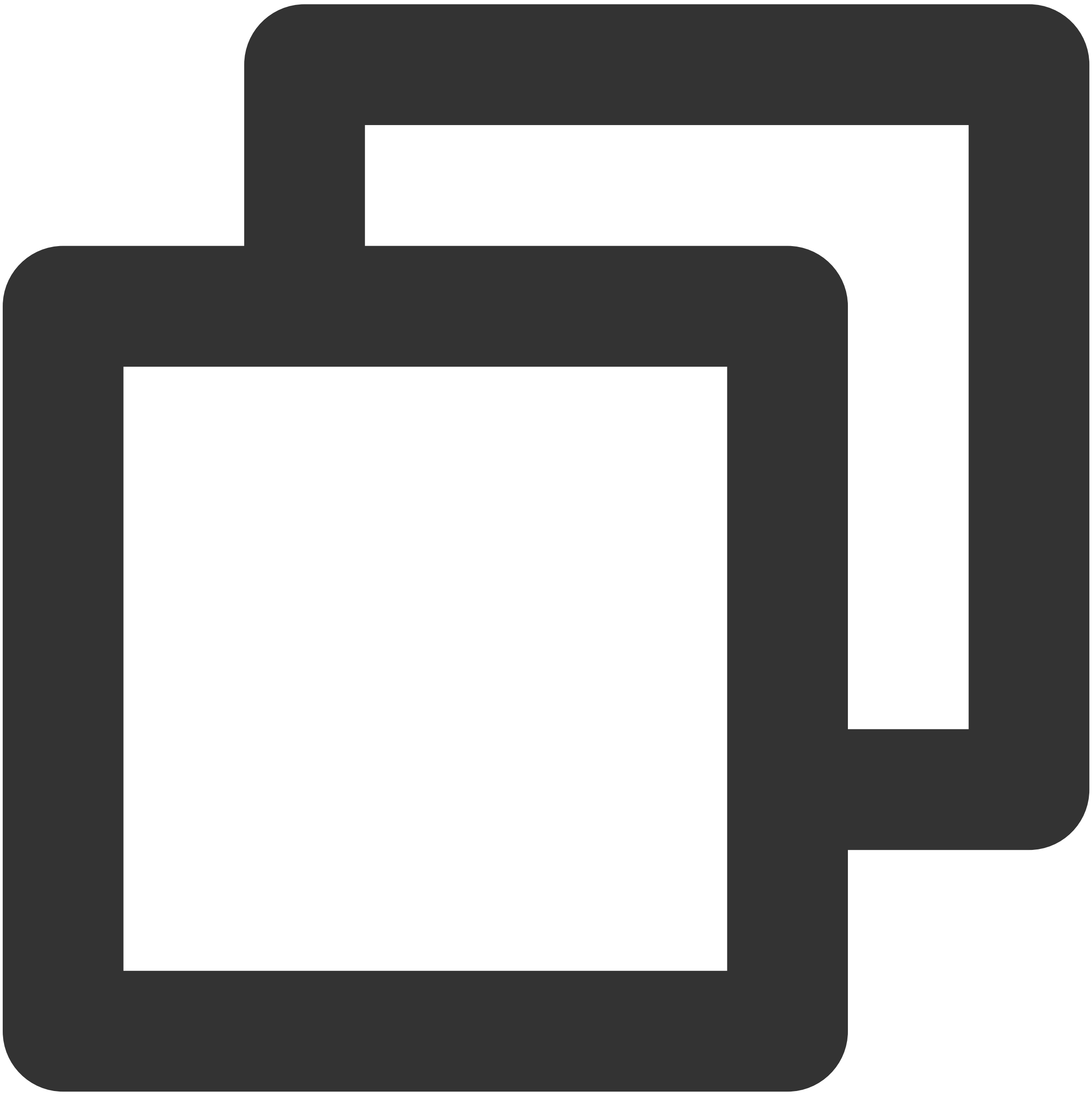


```
// Enable auto focus using TXDeviceManager
TXDeviceManager *manager = [self.trtcCloud getDeviceManager];
if ([manager isAutoFocusEnabled]) {
    [manager enableCameraAutoFocus:YES];
}
```

Note：

The front camera is turned on by default. If you need to use the rear camera, call `manager.switchCamera(false)` to turn on the rear camera.

3. Add the **localCameraVideoView** property to **ViewController.h** file.



```
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController
```



```
@property (nonatomic, strong) UIView *localCameraVideoView; // Add the localCameraV

@end
```

4. Initialize the **localCameraVideoView** in **ViewController.m** file.

```
#import "ViewController.h"
#import "AppDelegate.h"

@interface ViewController ()

@end
```

```
@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    // Initialize the localCameraVideoView
    self.localCameraVideoView = [[UIView alloc] initWithFrame:self.view.bounds];
    self.localCameraVideoView.backgroundColor = [UIColor blackColor];
}

- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];
    // Get the AppDelegate instance
    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] de

    // Add the localCameraVideoView to the trtcCloud of the AppDelegate
    [appDelegate.trtcCloud startLocalPreview:YES view:self.localCameraVideoView];

    // Add the localCameraVideoView to the view of the ViewController
    [self.view addSubview:self.localCameraVideoView];
}

@end
```

Step 7. Turn on microphone

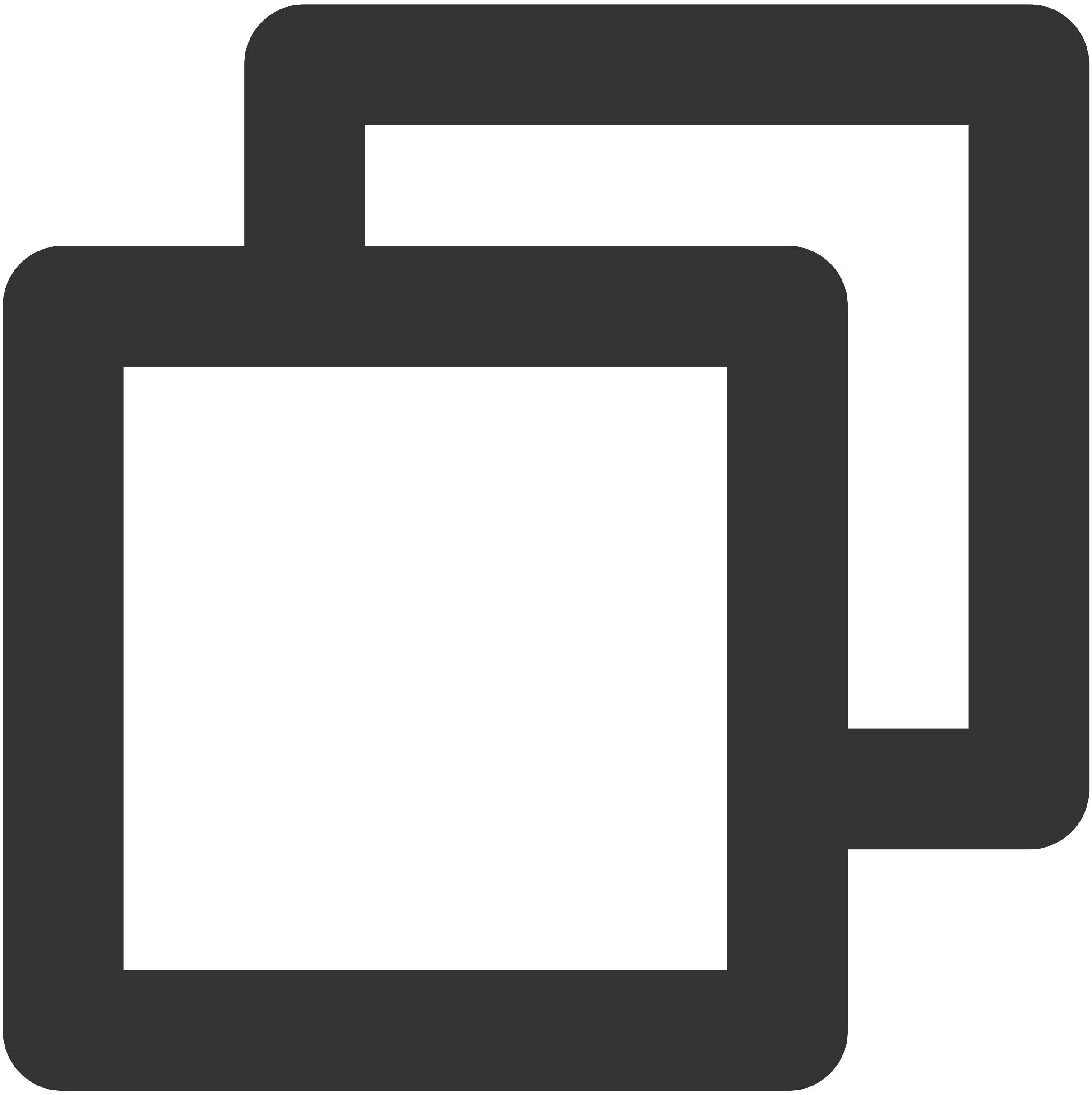
Call `startLocalAudio` to enable microphone capture. This interface requires you to determine the capture mode by the `quality` parameter. It is recommended to **select one of the following modes that is suitable for your project according to your needs**.

```
// Enable microphone acquisition and set the current scene to: Voice mode
// For high noise suppression capability, strong and weak network resistance
[self.trtcCloud startLocalAudio:TRTCAudioQualitySpeech];

// Enable microphone acquisition, and set the current scene to: Music mode
// For high fidelity acquisition, low sound quality loss, recommended to use with p
[self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
```

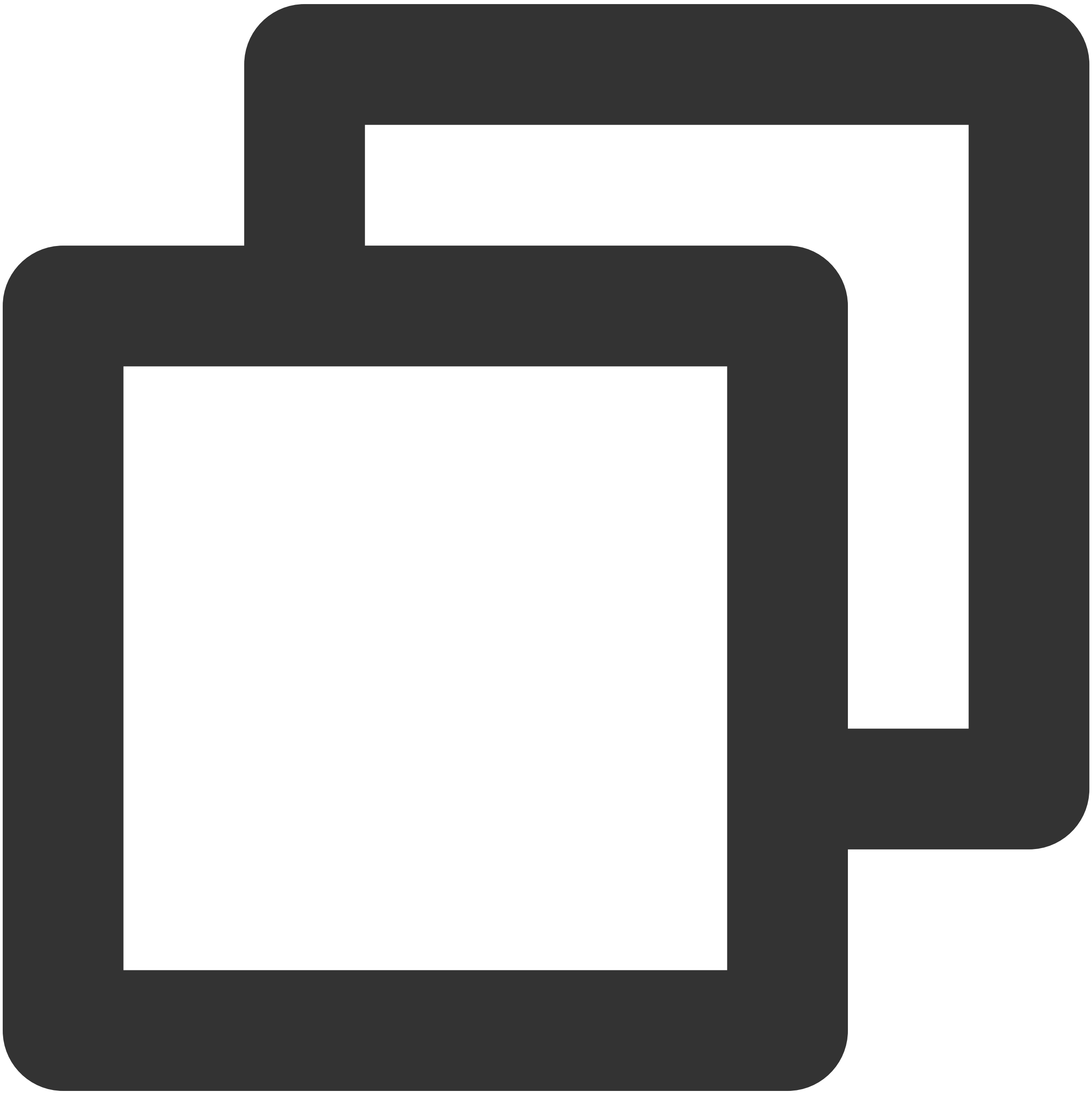
Step 8. Play/stop video streaming

After you enter denny's room as an audience by following steps 1-4 to create a new project, you can play a video of the remote user by calling the `startRemoteView` .



```
// Play denny's camera footage
[self.trtcCloud startRemoteView:@"denny" streamType:TRTCVideoStreamTypeBig view:cam
```

Then, you can call the `stopRemoteView` to stop the videos of a remote user. Alternatively, you can also stop the videos of all remote users via the `stopAllRemoteView` .



```
// Stop denny's camera footage
[self.trtcCloud stopRemoteView:@"denny" streamType:TRTCVideoStreamTypeBig view:came
// Stop all camera footages
[self.trtcCloud stopAllRemoteView];
```

Step 9. Play/stop the audio stream

Mute the voice of remote user denny by calling the `muteRemoteAudio("denny", true)` .


```
// Mute user with id denny
[self.trtcCloud muteRemoteAudio:@"denny" mute:YES];
```

You can also unmute him later by calling the `muteRemoteAudio("denny", false)` .


```
// Unmute user with id denny
[self.trtcCloud muteRemoteAudio:@"denny" mute:YES];
```

Step 10. Exit the room

Call the `exitRoom` to exit the current room, the SDK will notify you after the check-out through the `onExitRoom(int reason)` callback event.

```
// Exit current room
[self.trtcCloud exitRoom];

// Listen for the onExitRoom callback to find out why you checked out
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        NSLog(@"Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        NSLog(@"Kicked out of the current room by server through the restful api...
    } else if (reason == 2) {
        NSLog(@"Current room is dissolved by server through the restful api...");
```

```
    }
}
```

FAQs

API Reference at API Reference.

If you encounter any issues during integration and use, please refer to Frequently Asked Questions.

Contact us

If you have any suggestions or feedback, please contact `info_rtc@tencent.com` .

Mac

Last updated：2024-07-18 15:21:10

This tutorial mainly introduces how to implement a basic audio and video call.

Prerequisites

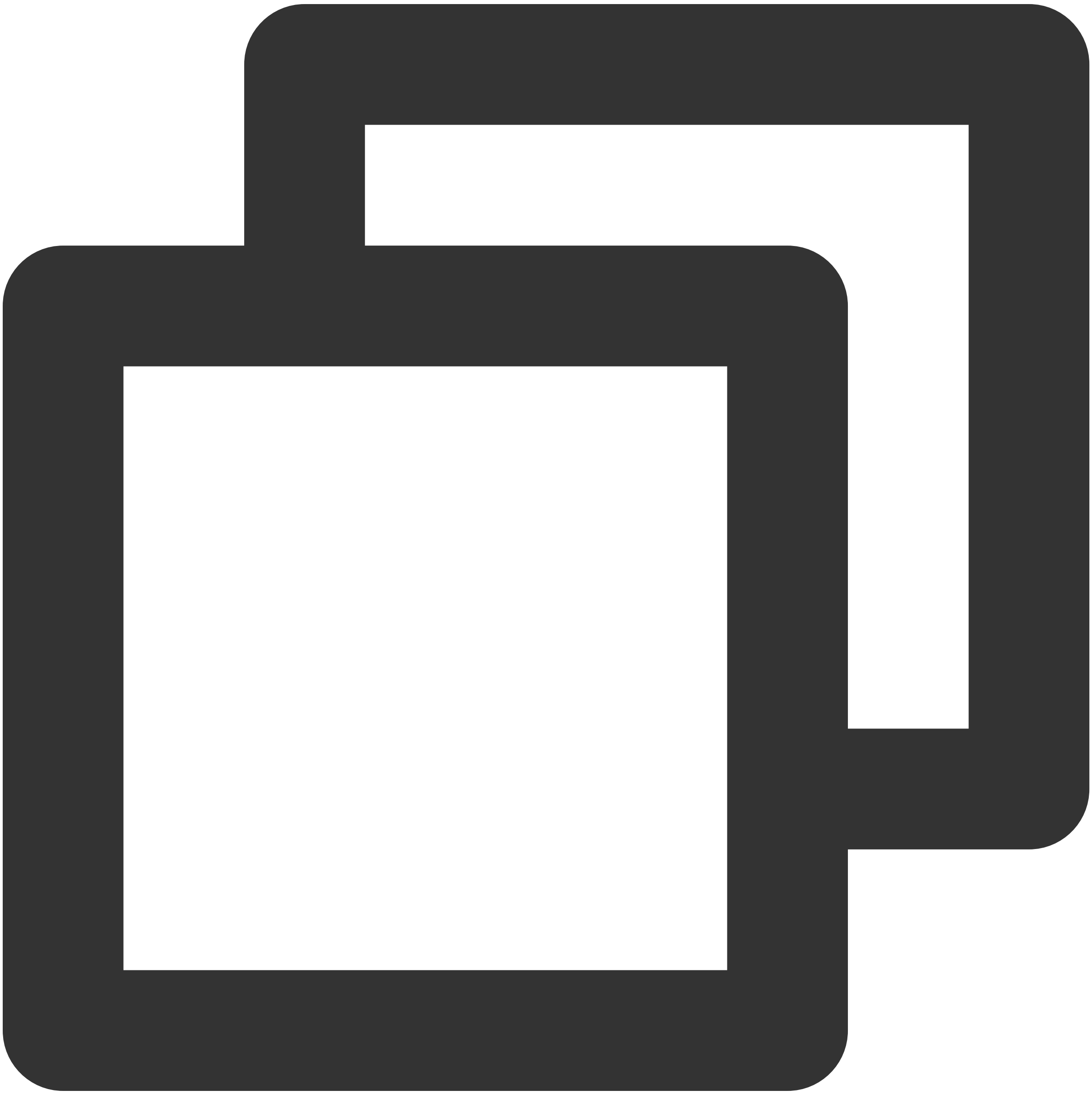
Xcode 9.0 or later.

A Mac computer with OS X 10.10 or later.

A valid developer signature for your project.

Step 1. Import TRTC SDK

1. Run the following command in the terminal window to install CocoaPods. If you have installed CocoaPods, skip this step.

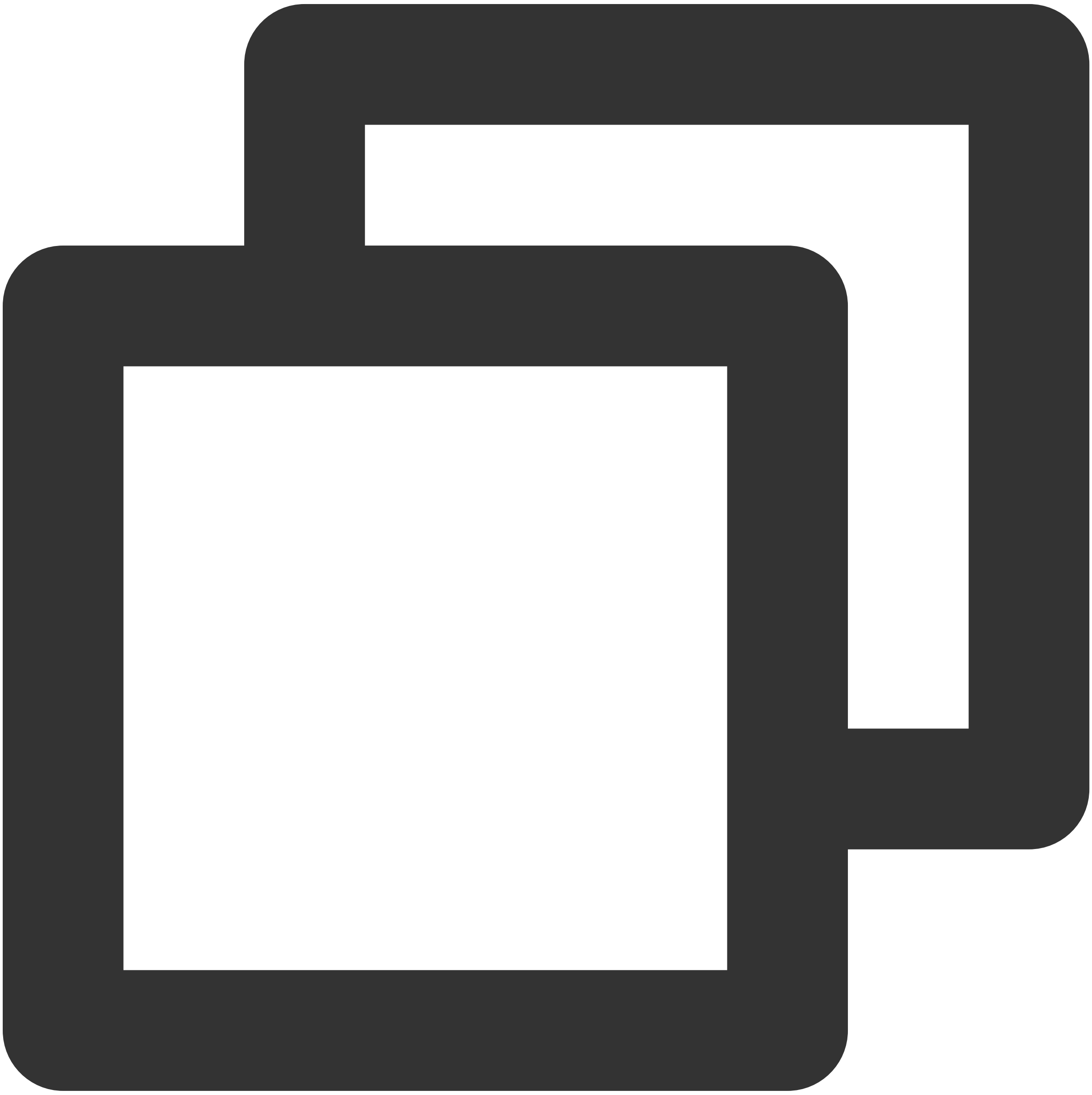


```
sudo gem install cocoapods
```

2. After going to the **TRTCDemo** root directory, enter the following command to create the **Podfile** file for your project.

```
pod init
```

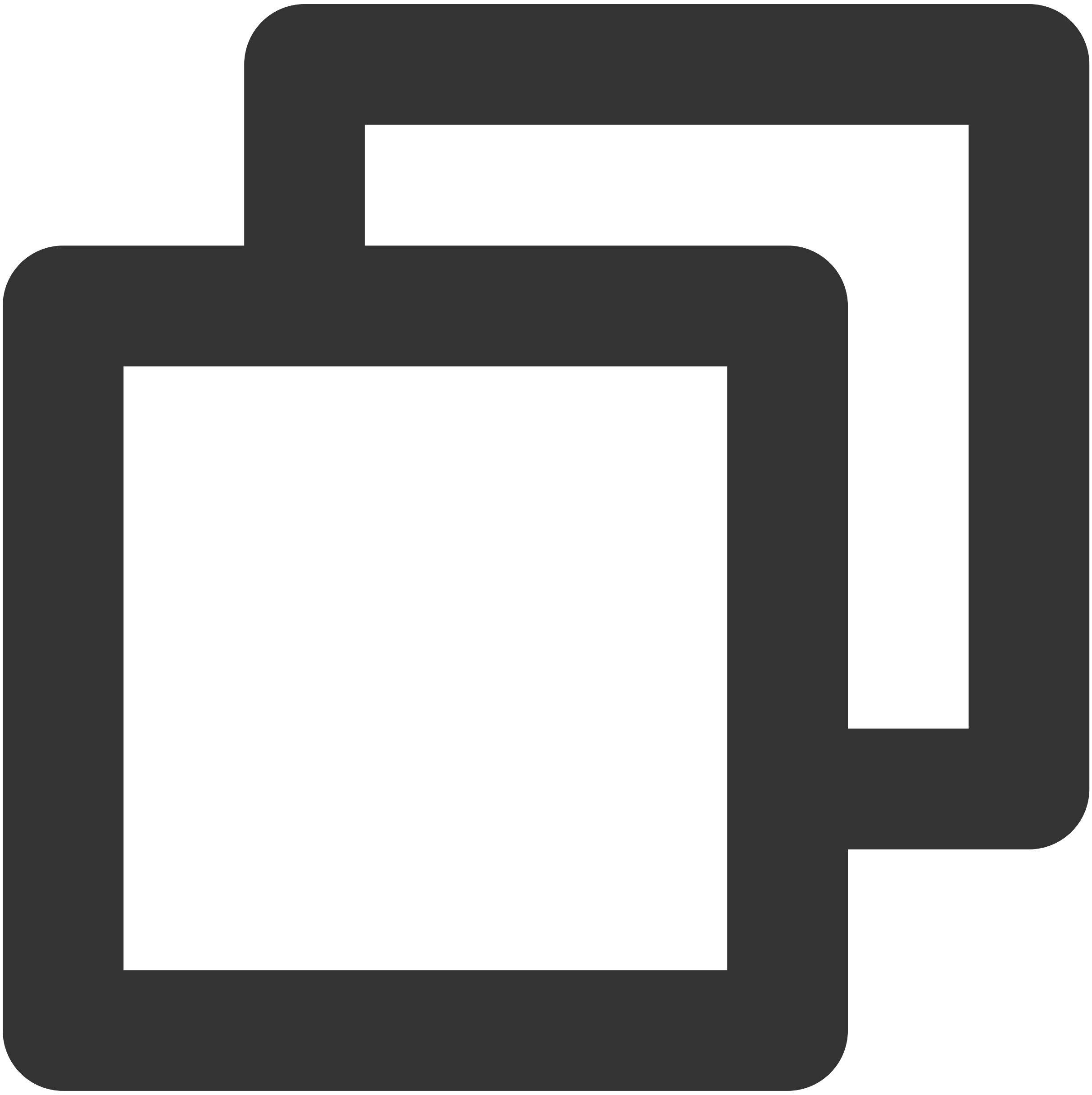
3. Edit the Podfile file as follows and change **Your Target** to your own project name.



```
platform :osx, '10.10'

target 'Your Target' do
pod 'TXLiteAVSDK_TRTC_Mac', :podspec => 'https://liteav.sdk.qcloud.com/pod/liteavsd
end
```

4. Enter the following command to update the local library file and install the SDK.



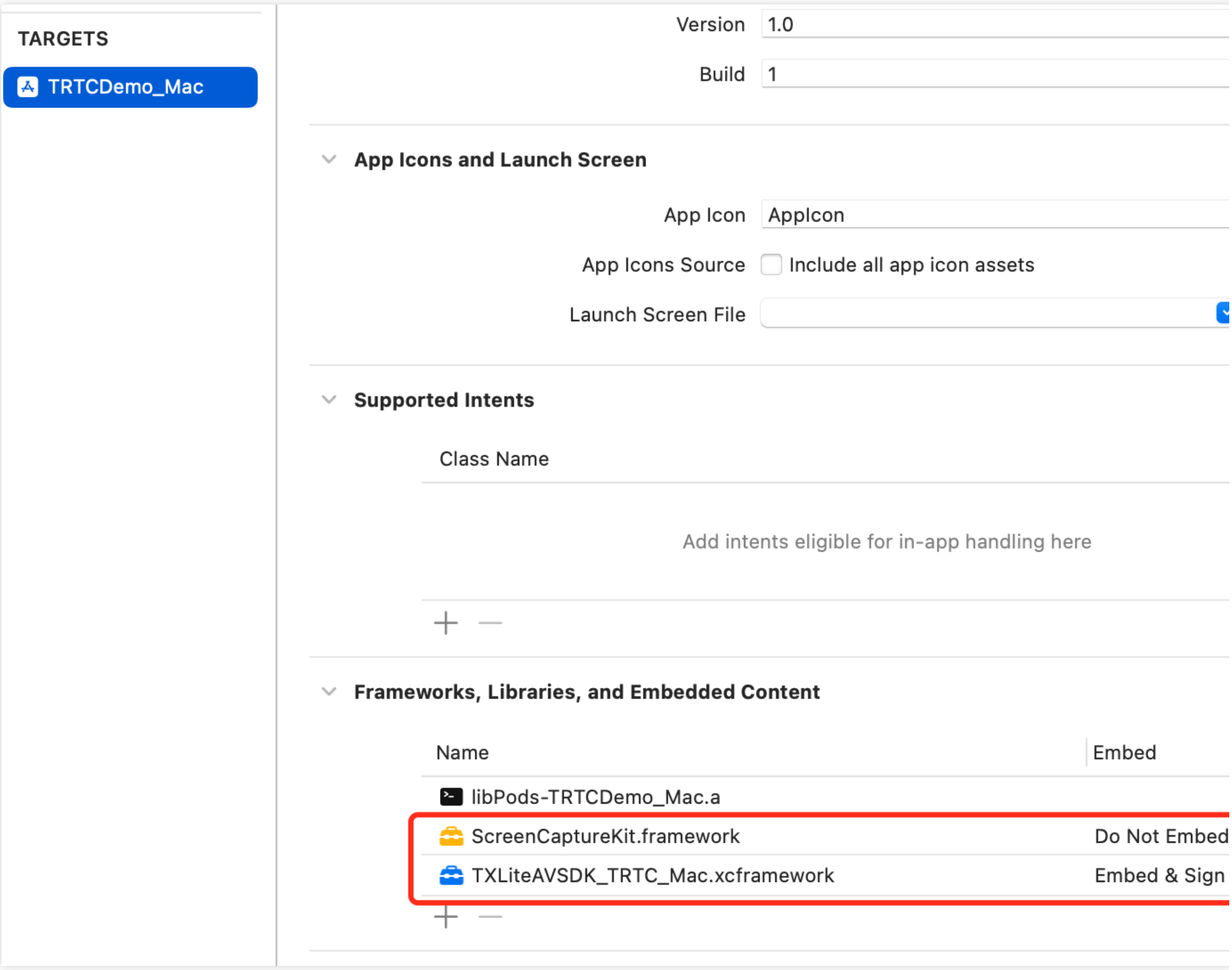
```
pod install
```

Note：

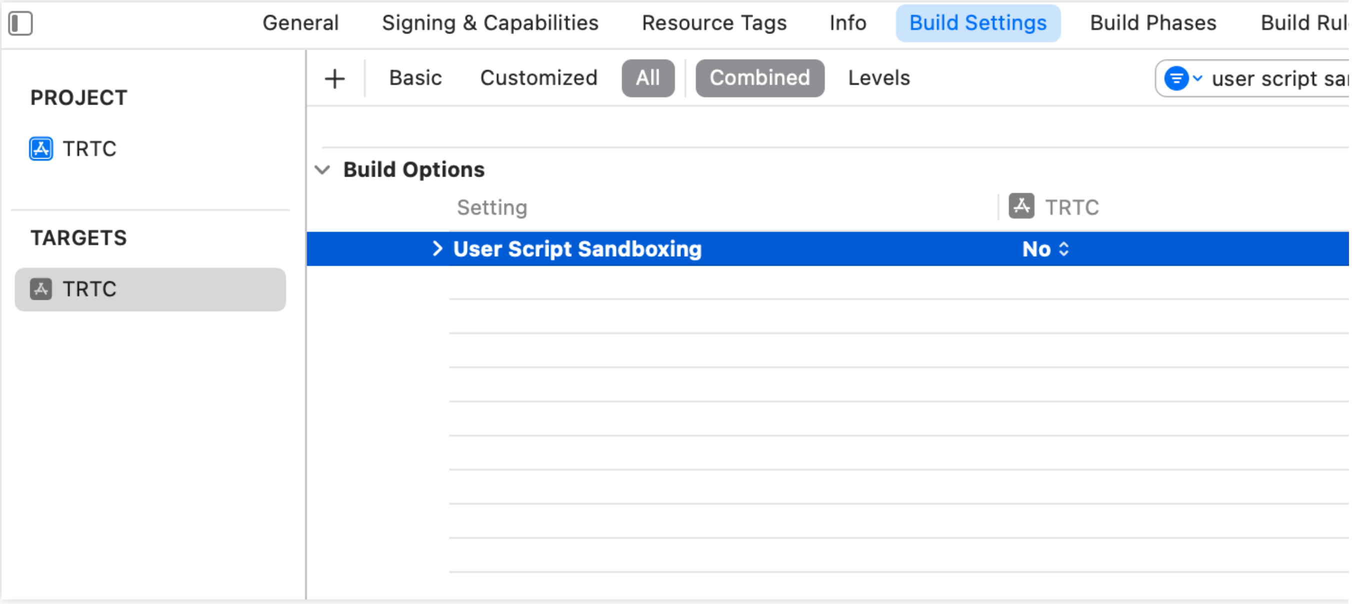
After the pod command is executed, a project file with the **.xcworkspace** suffix integrated with the SDK is generated. Double-click the **.xcworkspace** file to open it.

Step 2. Configure project

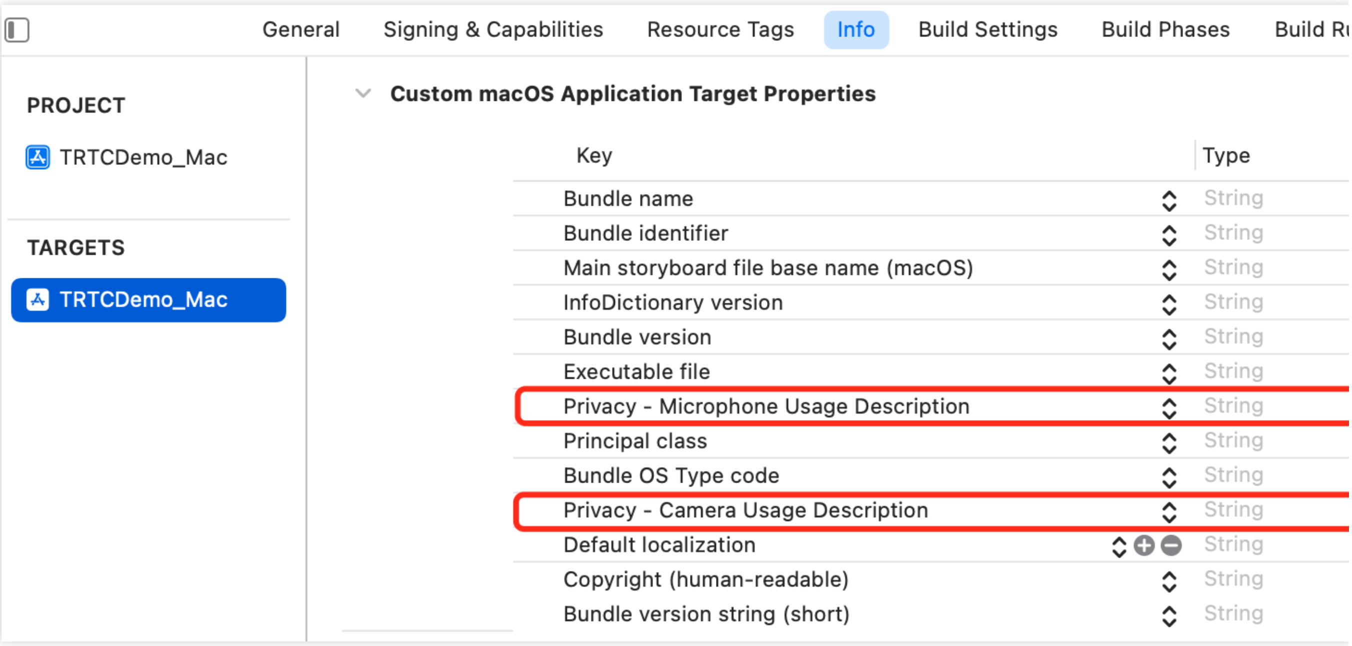
1. After opening the **.xcworkspace** file, add **TXLiteAVSDK_TRTC.xcframework** and **ScreenCaptureKit.framework** to the **Frameworks, Libraries, and Embedded Content** section in **General** tab.



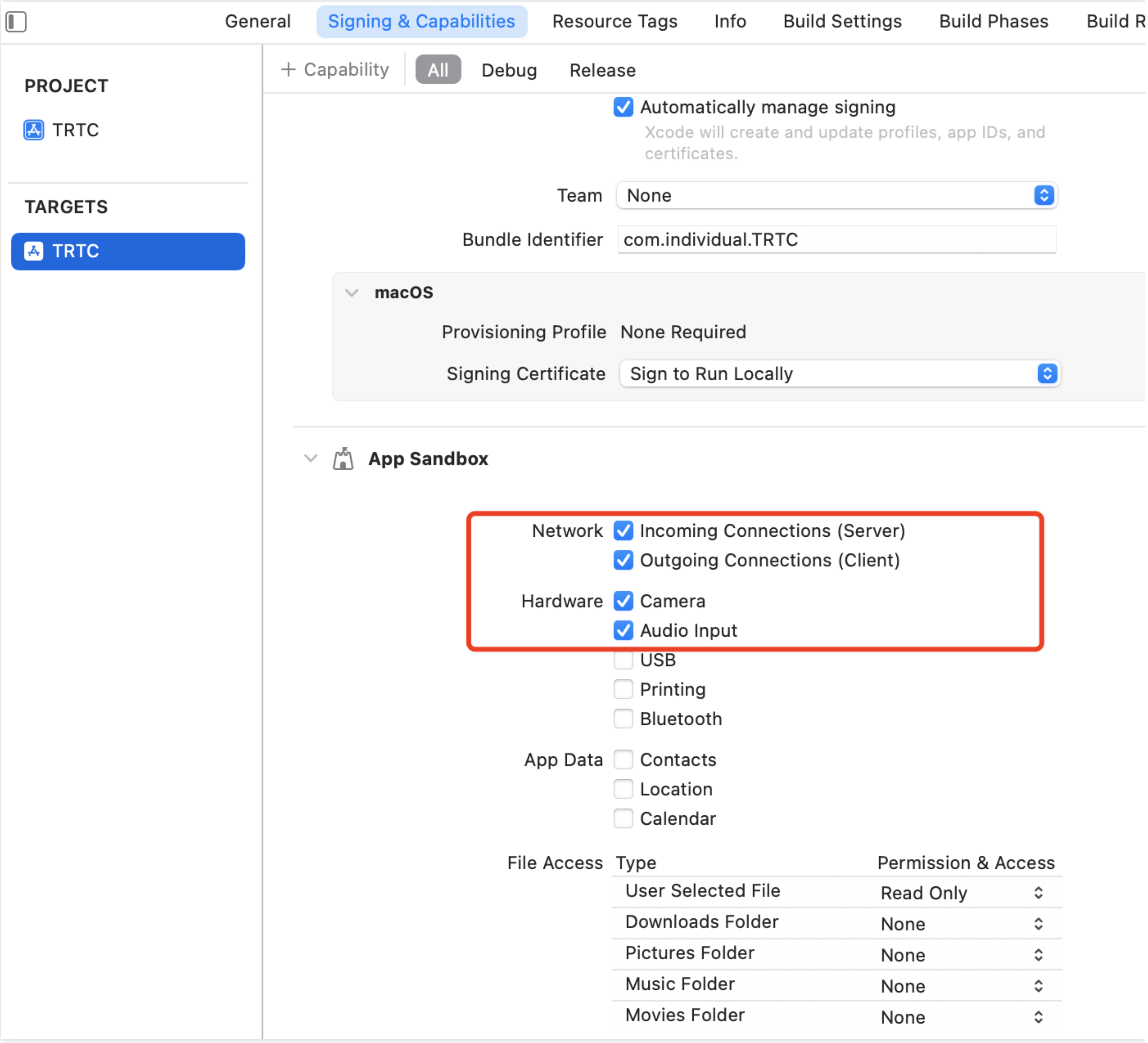
2. Search for **User Script Sandboxing** in **Build Settings** tab and set its value to **No**.



3. Add **Privacy-Microphone Usage Description** and **Privacy-Microphone Usage Description** to **Info.plist** tab, and fill in the target prompt words used by the Microphone/Camera to obtain the permissions to use the microphone and camera.



4. Check the following in the **App Sandbox** section of **Signing & Capabilities**.



Step 3. Create TRTC instance

1. Add a module reference to the SDK in the **AppDelegate.h** file:

```
@import TXLiteAVSDK_TRTC_Mac;
```

2. Add the following properties to the AppDelegate.h file and declare the `toastTip:` method.

```
#import <Cocoa/Cocoa.h>
@import TXLiteAVSDK_TRTC_Mac;

@interface AppDelegate : NSObject <NSApplicationDelegate>

@property (nonatomic, strong) NSWindow *window; // Add window property
@property (nonatomic, strong) TRTCCloud *trtcCloud; // Add trtcCloud property
@property (nonatomic, strong) NSView *localCameraVideoView; // Add localCameraVideo

- (void)toastTip:(NSString *)tip; // Declare the toastTip: method
```

```
@end
```

3. Implement the `toastTip:` method in the **AppDelegate.m** file.

```
// Implement toastTip: method
- (void)toastTip:(NSString *)tip {
    NSAlert *alert = [[NSAlert alloc] init];
    [alert setMessageText:tip];
    [alert runModal];
}
```

4. Call the create TRTC instance initialization interface in the `didFinishLaunchingWithOptions()` method , and set up the event callbacks .

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    // Override point for customization after application launch.

    // Create trtc instance(singleton)  and set up event listeners
    _trtcCloud = [TRTCCloud sharedInstance];
    _trtcCloud.delegate = self;

    return YES;
```

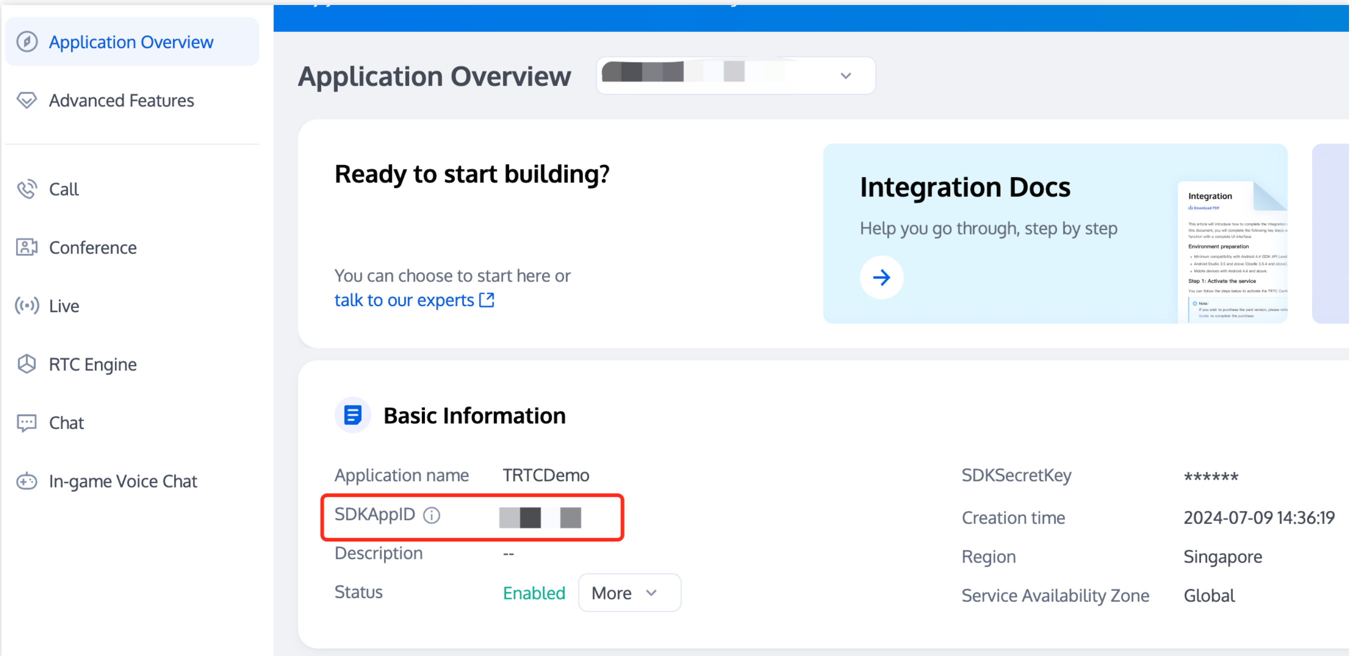


```
}

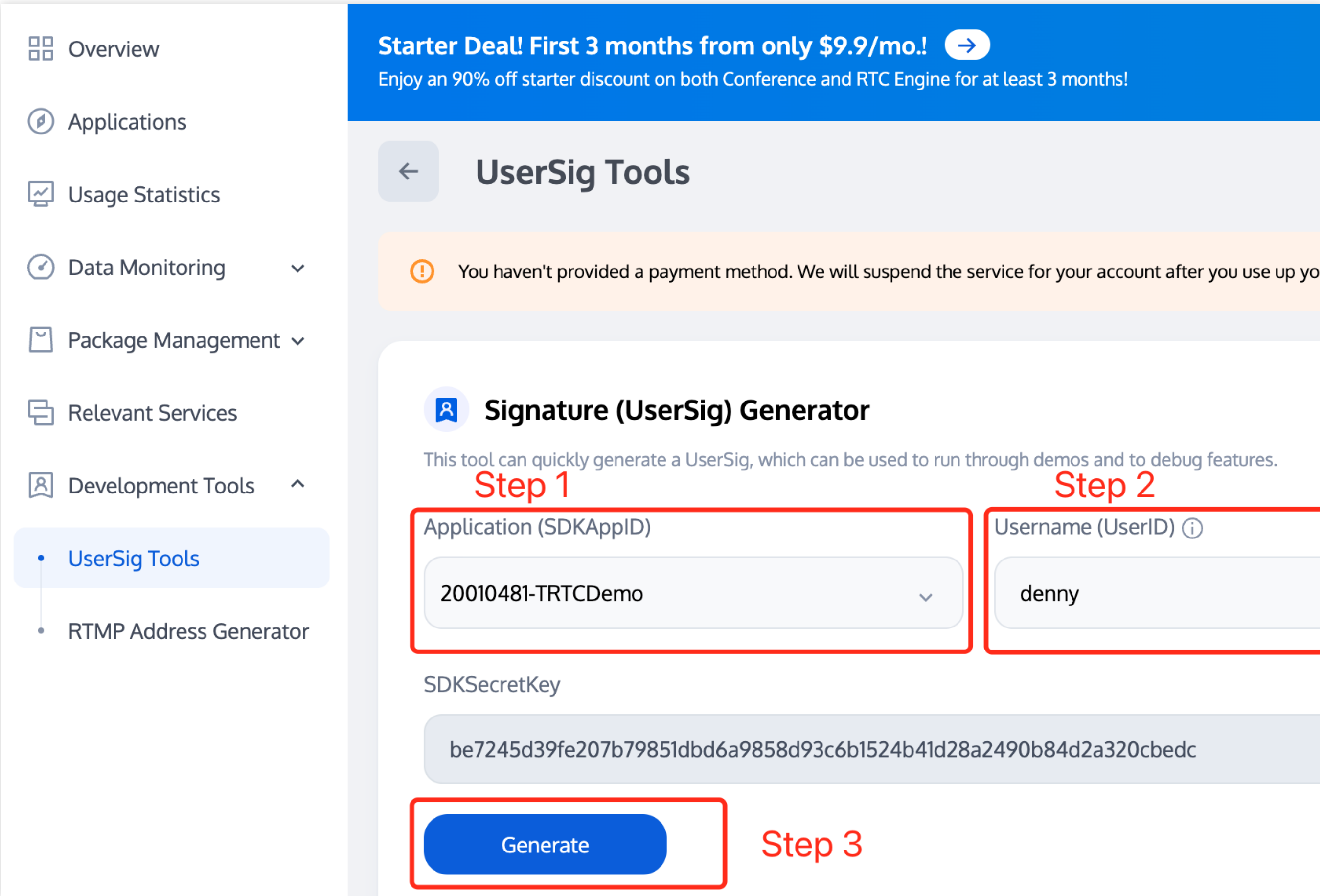
// Listen to the "onError" event, and print logs for errors such as "Camera is not
- (void)onError:(TXLiteAVError)errCode
         errMsg:(nullable NSString *)errMsg
        extInfo:(nullable NSDictionary *)extInfo{
    if (ERR_CAMERA_NOT_AUTHORIZED == errCode) {
        NSString *errorInfo = @"Current application is not authorized to use the ca
        errorInfo = [errorInfo stringByAppendingString : errMsg];
        [self toastTip:errorInfo];
    }
}
```

Step 4. Enter the room

1. Click `Create Application` in the Tencent RTC console to get the **SDKAppID** under **Application Overview**.



2. Select **SDKAppID** down in the **UserSig Tools**, enter your **UserID**, and click `Generate` to get your own **UserSig**.



3. After setting the **TRTCParams** for room entry, call the `enterRoom` to enter the room.

As an Anchor：


```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    // Override point for customization after application launch.

    // ...Other codes

    // Please replace each field in TRTCParams with your own parameters
    TRTCParams *trtcParams = [[TRTCParams alloc] init];
    trtcParams.sdkAppId = 1400000123;  // Please replace with your own SDKAppID
    trtcParams.roomId = 123321; // Please replace with your own room number
    trtcParams.userId = @"denny";   // Please replace with your own userid
    trtcParams.userSig = @""; // Please replace with your own userSig
```

```
    trtcParams.role = TRTCRoleAnchor;

    // If your application scenario is a video call between several people, please
    [self.trtcCloud enterRoom:trtcParams appScene:TRTCAppSceneLIVE];

    return YES;
}

// Listen for the `onEnterRoom` event of the SDK and learn whether the room is succ
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        [self toastTip:@"Enter room succeed!"];
    } else {
        [self toastTip:@"Enter room failed!"];
    }
}
```

As an audience:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    // Override point for customization after application launch.

    // ...Other codes

    // Please replace each field in TRTCParams with your own parameters
    TRTCParams *trtcParams = [[TRTCParams alloc] init];
    trtcParams.sdkAppId = 1400000123;  // Please replace with your own SDKAppID
    trtcParams.roomId = 123321; // Please replace with your own room number
    trtcParams.userId = @"denny";   // Please replace with your own userid
    trtcParams.userSig = @""; // Please replace with your own userSig
```



```
    trtcParams.role = TRTCRoleAudience;

    // If your application scenario is a video call between several people, please
    [self.trtcCloud enterRoom:trtcParams appScene:TRTCAppSceneLIVE];

    return YES;
}

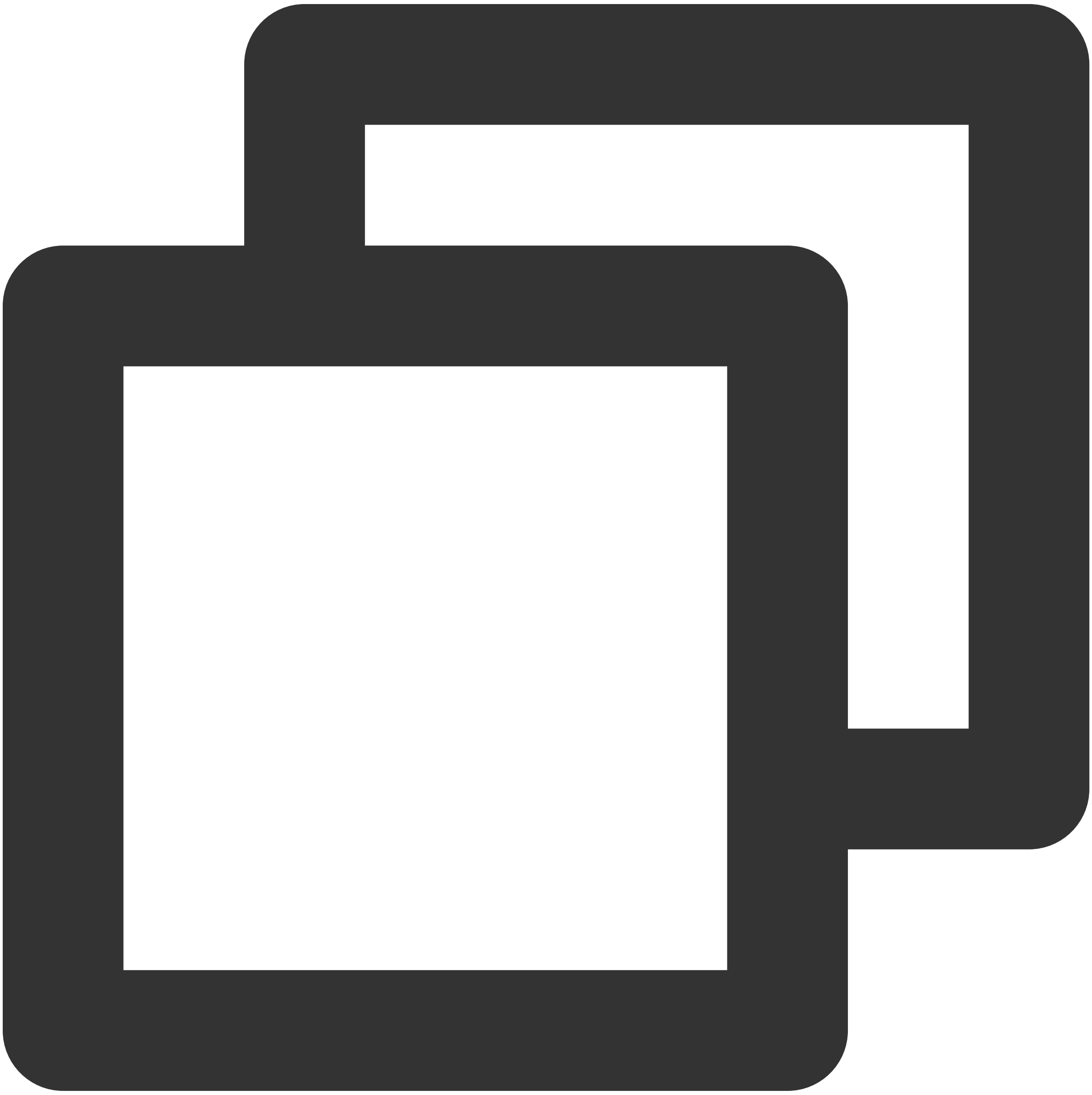
// Listen for the `onEnterRoom` event of the SDK and learn whether the room is succ
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        [self toastTip:@"Enter room succeed!"];
    } else {
        [self toastTip:@"Enter room failed!"];
    }
}
```

Note：

If you enter the room as an **audience**, **sdkAppId** and **roomId** need to be the same as on the anchor side, while **userId** and **userSig** need to be replaced with your own values.

Step 5. Turn on Camera

Initialize **localCameraVideoView** in `didFinishLaunchingWithOptions()` method, and call the `setLocalRenderParams` to set the local preview render parameters.



```
// Create a window
self.window = [[NSWindow alloc] initWithContentRect:NSMakeRect(0, 0, 800, 600) styl
[self.window center];
[self.window setTitle:@"TRTCDemo_Mac"];
[self.window makeKeyAndOrderFront:nil];
self.window.releasedWhenClosed = NO;

// Initialize localCameraVideoView
self.localCameraVideoView = [[NSView alloc] initWithFrame:NSMakeRect(0, 0, 300, 300
[self.window.contentView addSubview:self.localCameraVideoView];
```

```
// Adjust the localCameraVideoView frame to match the size of the window
self.localCameraVideoView.frame = self.window.contentView.bounds;

// Set the preview mode of the local screen
TRTCRenderParams *trtcRenderParams = [[TRTCRenderParams alloc] init];
trtcRenderParams.fillMode   = TRTCVideoFillMode_Fill;
trtcRenderParams.mirrorType = TRTCVideoMirrorTypeAuto;
[self.trtcCloud setLocalRenderParams:trtcRenderParams];

// Start a preview of the local camera
[_trtcCloud startLocalPreview:_localCameraVideoView];
```

Step 6. Turn on microphone

Call `startLocalAudio` to enable microphone capture. This interface requires you to determine the capture mode by the `quality` parameter. It is recommended to **select one of the following modes that is suitable for your project according to your needs**.

```
// Enable microphone acquisition and set the current scene to: Voice mode
// For high noise suppression capability, strong and weak network resistance
[self.trtcCloud startLocalAudio:TRTCAudioQualitySpeech];

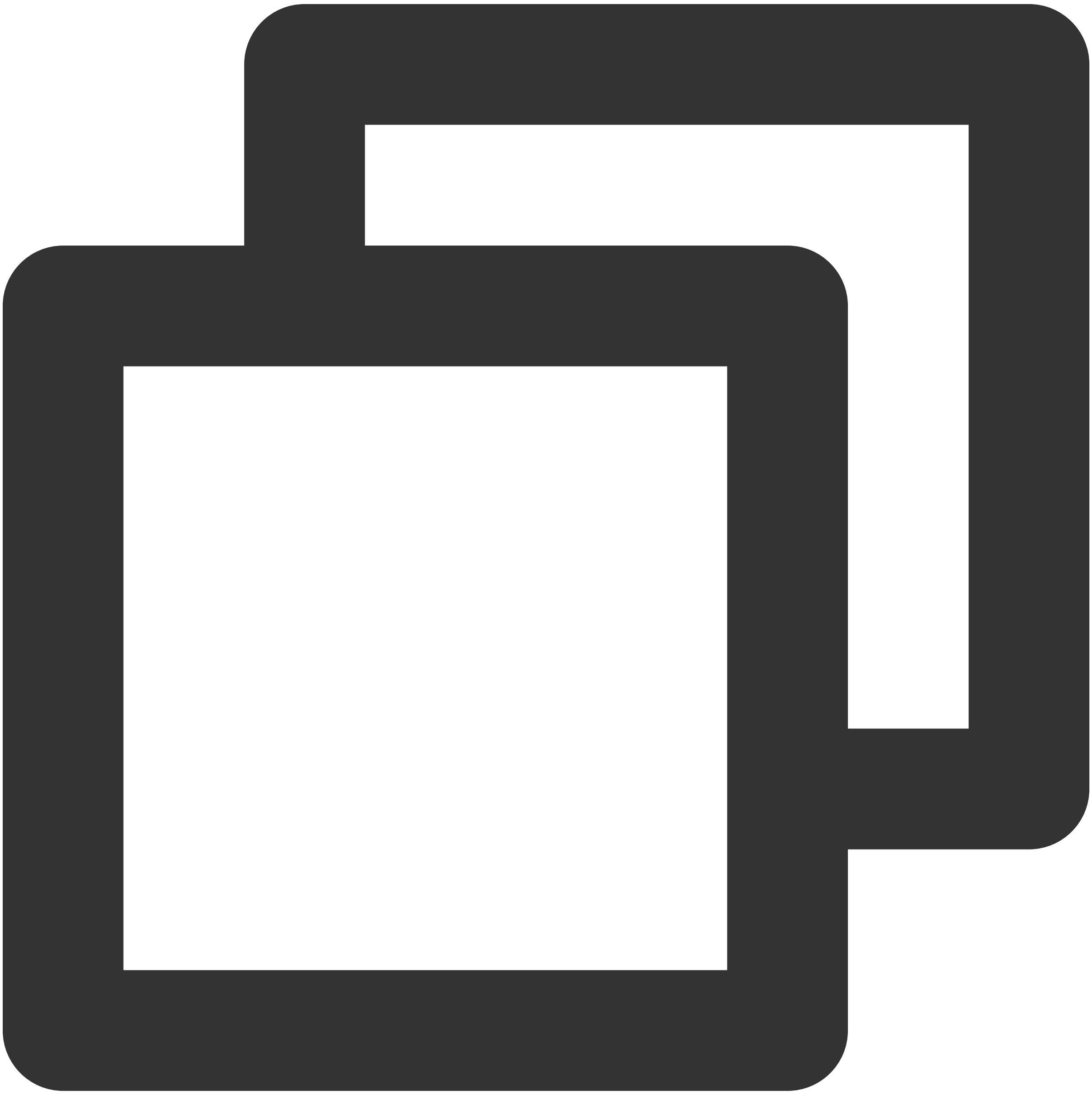
// Enable microphone acquisition, and set the current scene to: Music mode
// For high fidelity acquisition, low sound quality loss, recommended to use with p
[self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
```

Step 7. Play/stop video streaming

After you enter denny's room as an audience by following **steps 1-4** to create a new project, you can play a video of the remote user by calling the `startRemoteView` .

```
// Play denny's camera footage
[self.trtcCloud startRemoteView:@"denny" streamType:TRTCVideoStreamTypeBig view:cam
```

Then, you can call the `stopRemoteView` to stop the videos of a remote user. Alternatively, you can also stop the videos of all remote users via the `stopAllRemoteView` .



```
// Stop denny's camera footage
[self.trtcCloud stopRemoteView:@"denny" streamType:TRTCVideoStreamTypeBig view:came
// Stop all camera footages
[self.trtcCloud stopAllRemoteView];
```

Step 8. Play/stop the audio stream

Mute the voice of remote user denny by calling the `muteRemoteAudio("denny", true)` .

```
// Mute user with id denny
[self.trtcCloud muteRemoteAudio:@"denny" mute:YES];
```

You can also unmute him later by calling the `muteRemoteAudio("denny", false)` .

```
// Unmute user with id denny
[self.trtcCloud muteRemoteAudio:@"denny" mute:YES];
```

Step 9. Exit the room

Call the `exitRoom` to exit the current room, the SDK will notify you after the check-out through the `onExitRoom(int reason)` callback event.

```
// Exit current room
[self.trtcCloud exitRoom];

// Listen for the onExitRoom callback to find out why you checked out
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        NSLog(@"Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        NSLog(@"Kicked out of the current room by server through the restful api...
    } else if (reason == 2) {
        NSLog(@"Current room is dissolved by server through the restful api...");
```

```
    }
}
```

FAQs

API Reference at API Reference.

If you encounter any issues during integration and use, please refer to Frequently Asked Questions.

Contact us

If you have any suggestions or feedback, please contact `info_rtc@tencent.com` .

Windows C++

Last updated：2024-07-18 15:26:12

This tutorial mainly introduces how to implement a basic audio and video call.

Prerequisites

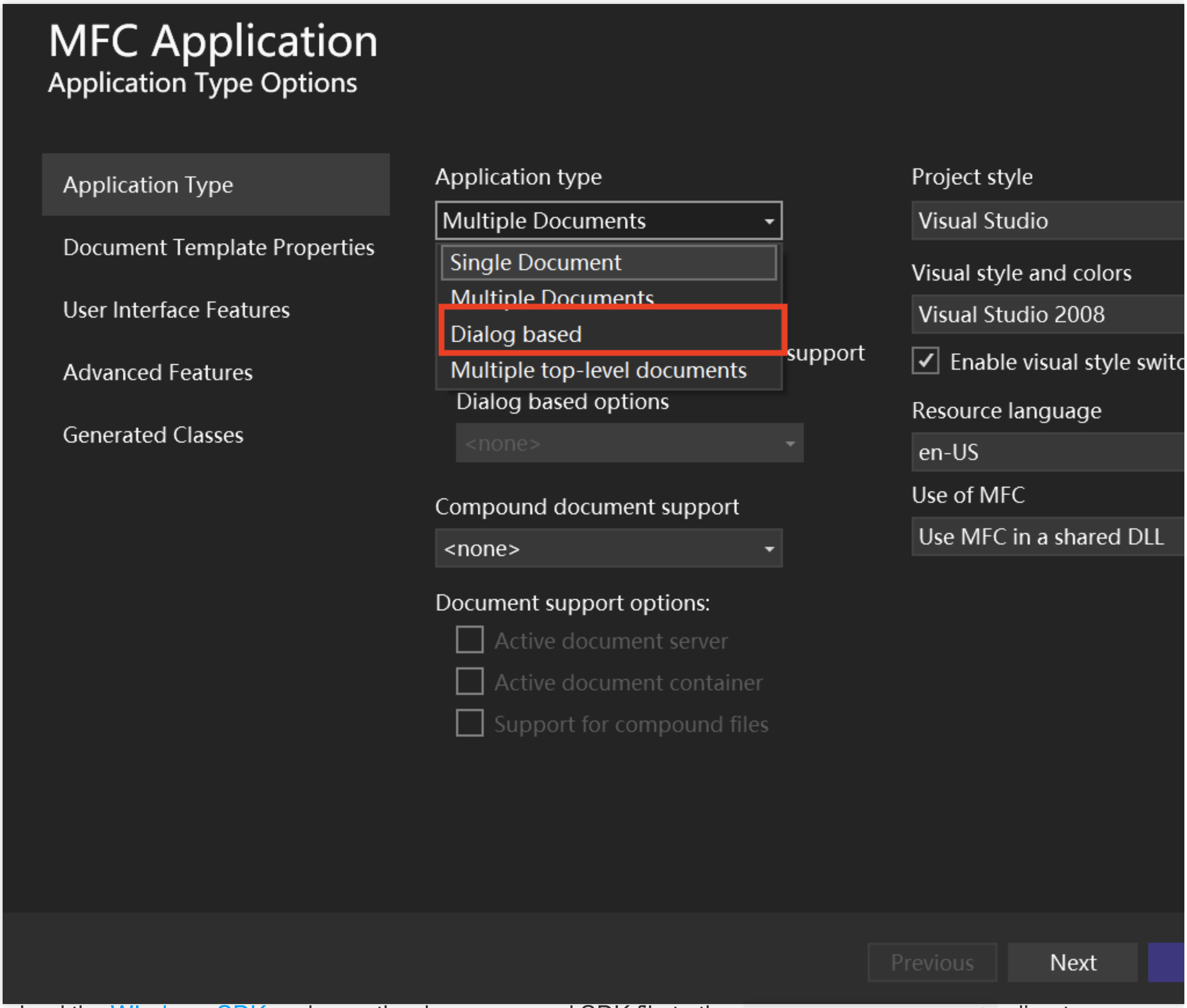
OS: Windows 7 or later.

Development environment: Visual Studio 2010 or later (v2015 is recommended).

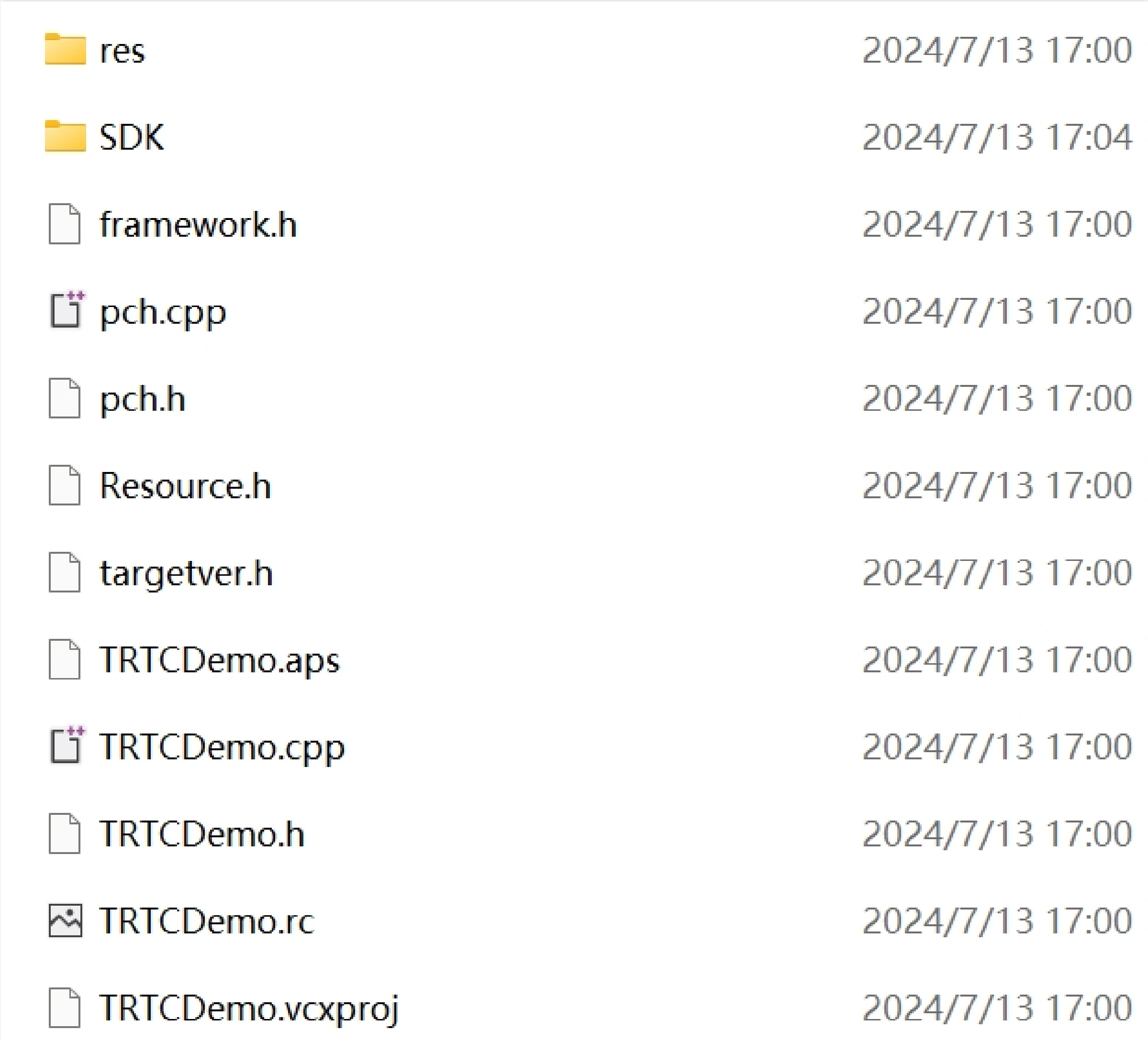
Step 1. Import TRTC SDK

1. Open Visual Studio and create a new **MFC** application called **TRTCDemo**.

On the **MFC Application** page of the wizard, select **Dialog based** for **Application typ**e and default for other wizard configurations. Click **Finish**.



2. Download the [WIndows SDK] and copy the decompressed SDK file to the `TRTCDemo.vcxproj` directory.

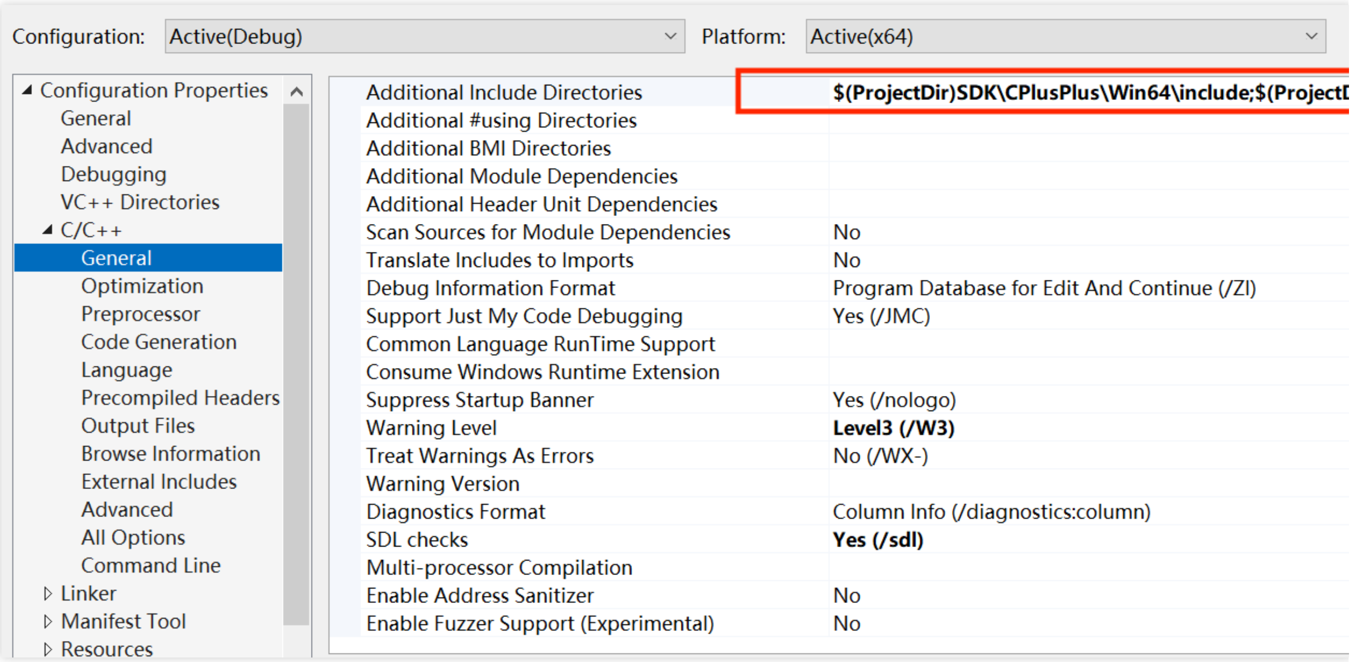


Step 2. Configure project

Open the T**RTCDemo.sln** property page, following **Solution Explorer** > **Right-click menu for TRTCDemo project** > **Properties**, and perform the following configuration:

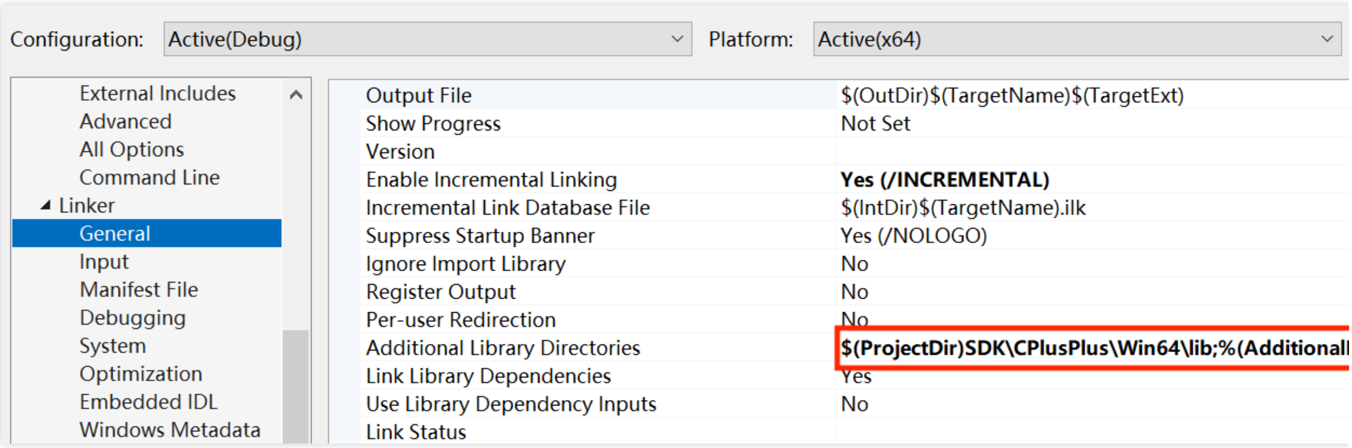
1. **Add Additional Include Directories:**

Following **C/C++** > **General** > **Additional Include Directories**, add the SDK header directory： `$(ProjectDir)SDK\\CPlusPlus\\Win64\\include` and `$(ProjectDir)SDK\\CPlusPlus\\Win64\\include\\TRTC` .



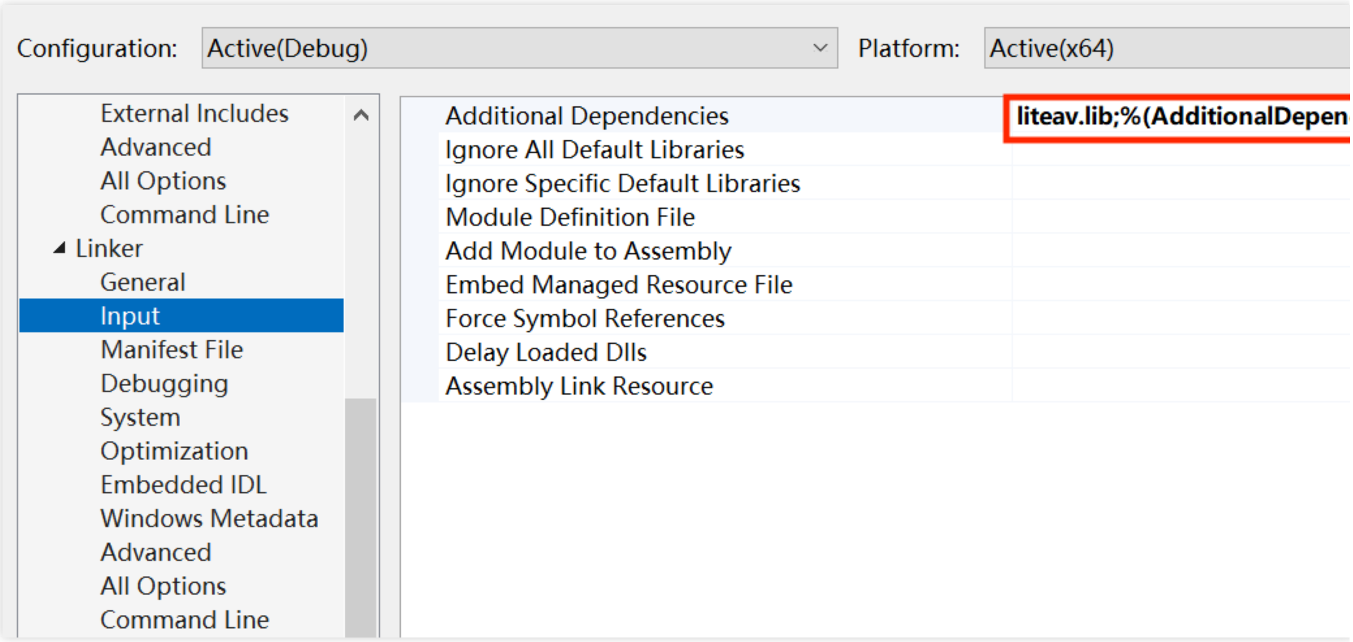
2. **Add Additional Library Directories：**

Following **Linker** > **General** > **Additional Library Directories**， add the SDK library directory: `$(ProjectDir)SDK\\CPlusPlus\\Win64\\lib` .



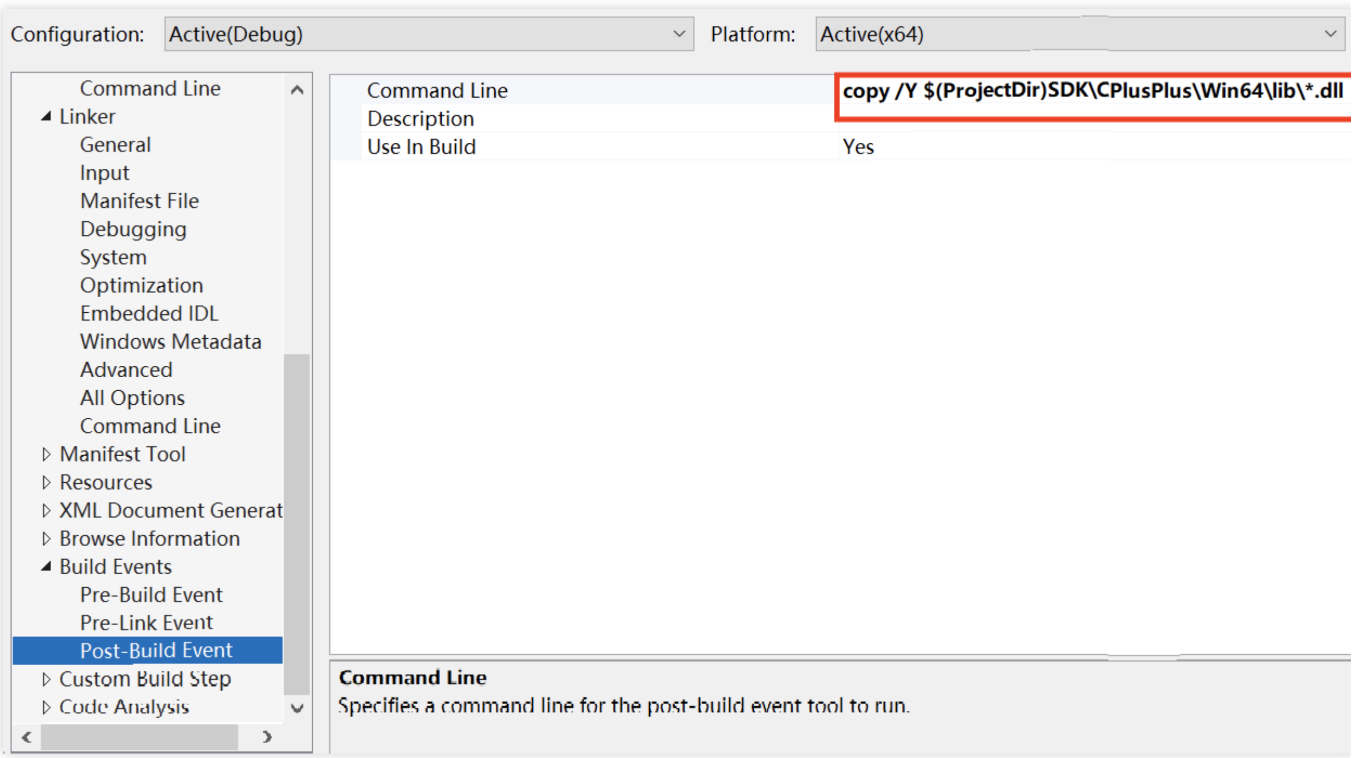
3. **Add Additional Dependencies：**

Following **Linker** > **Input** > **Additional Dependencie**， add SDK library files: `liteav.lib` .



4. **Add Command line**：

Following **Build Events** > **Post-Build Event** > **Command line**， add `copy /Y $(ProjectDir)SDK\\CPlusPlus\\Win64\\lib\\*.dll $(OutDir)` .



5. **Print SDK version:**

Introduce a header file at the top of the **TRTCDemoDlg.cpp** file:


```
#include "ITRTCCloud.h"
```

Note：

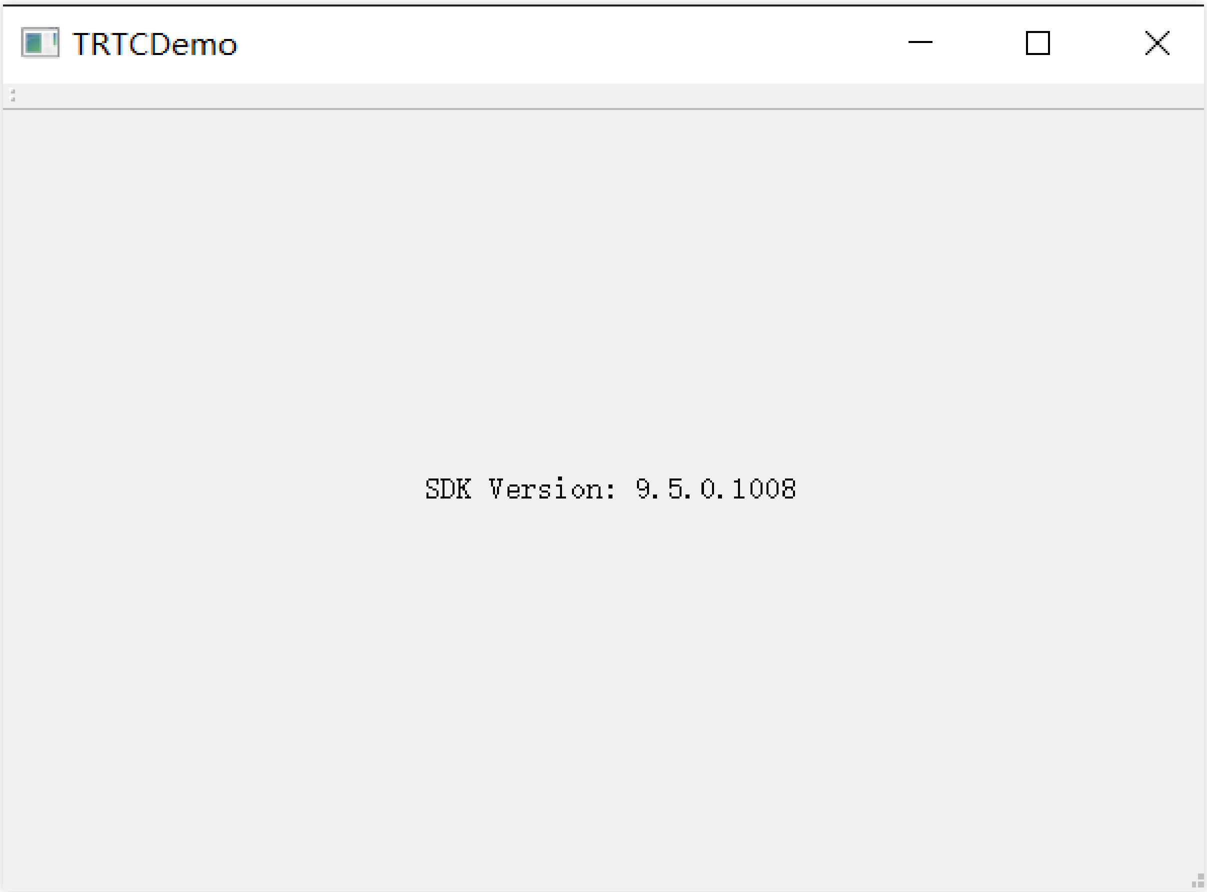
Refer to **"ITRTCCloud.h"** after the existing header file, otherwise you will get an error `ITRTCCloud: undefined identifier`.

Add the following codes in `CTRTCDemoDlg::OnInitDialog` function:

```
ITRTCCloud * pTRTCCloud = getTRTCShareInstance();
CString szText;
szText.Format(L"SDK version: %hs", pTRTCCloud->getSDKVersion());

CWnd *pStatic = GetDlgItem(IDC_STATIC);
pStatic->SetWindowTextW(szText);
```

After completing the preceding steps, click **Run** to print the SDK version number.



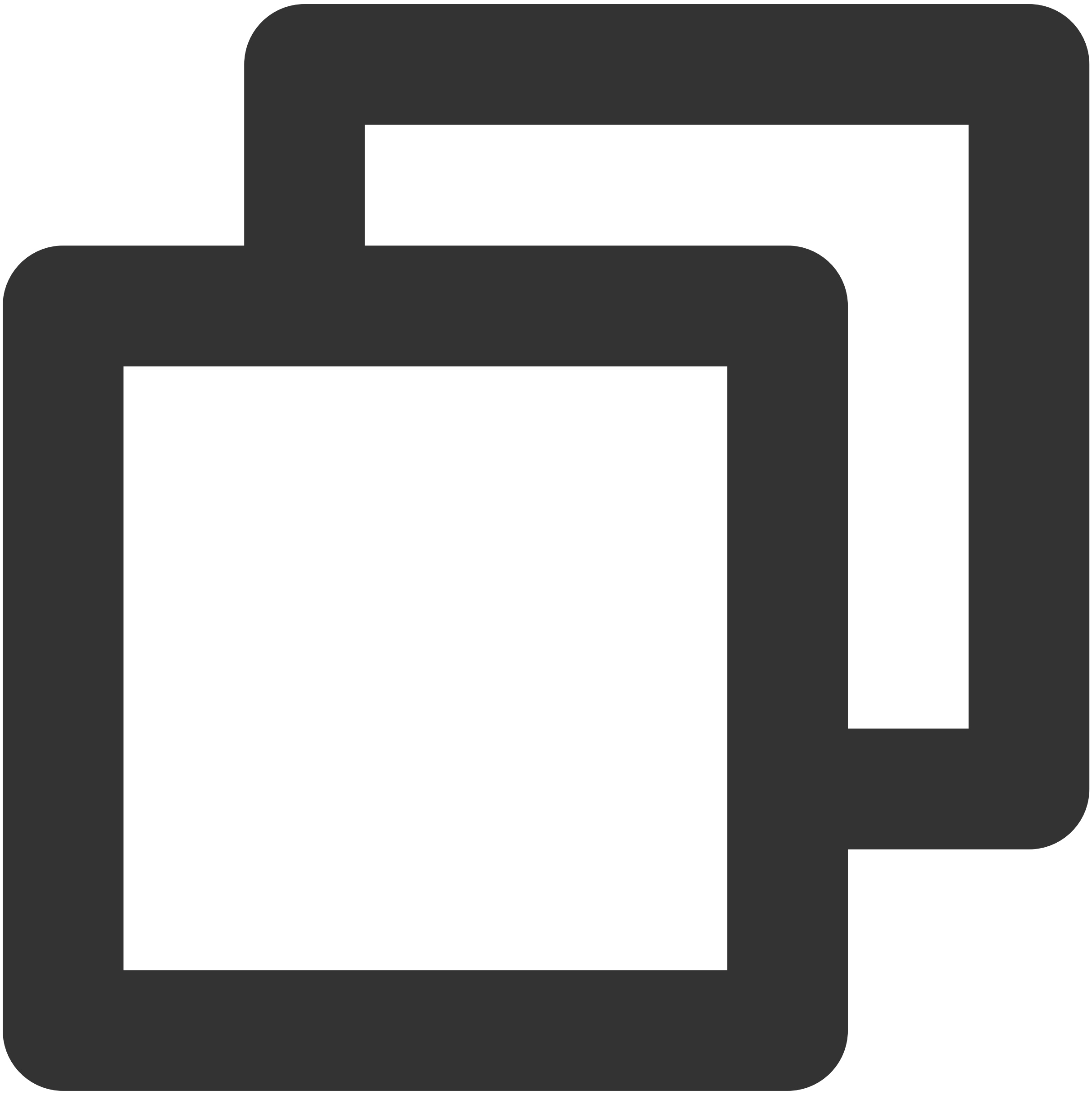
Note：

If you get the error message "module machine type 'x86' conflicts with target machine type 'x64', select 'x64' in the solution platform.

Step 3. Create TRTC instance

1. Reference header `"ITRTCCloud.h"` in the **TRTCDemo.h** file.

The **CTRTCDemo** class is publicly inherited from **CWinApp** and **ITRTCCloudCallback** and declares callback functions and member variables.



```
#include "ITRTCCloud.h"

class CTRTCDemoApp : public CWinApp, public ITRTCCloudCallback
{
public:
        CTRTCDemoApp();

// Overwrite
public:
        virtual BOOL InitInstance();
        virtual void onError(TXLiteAVError errCode, const char* errMsg, void* extra
```



```
        virtual void onWarning(TXLiteAVWarning warningCode, const char* warningMsg,
        virtual void onEnterRoom(int result) override;
        virtual void onExitRoom(int reason) override;
// Emplement

        DECLARE_MESSAGE_MAP()

private:
        ITRTCCloud* trtc_cloud_; // Declare the member variable trtc cloud
};
```

2. Call the initialization interface to create an object instance of TRTC in **CTRTCDemo::InitInstance()** method within the **TRTCDemo.cpp** file.

```
// Create trtc instance(singleton)  and set up event listeners
trtc_cloud_ = getTRTCShareInstance();
trtc_cloud_->addCallback(this);
```

Note：

Add the initialization interface code after the `SetRegistryKey(_T("Local AppWizard-Generated Applications"))` method.

3. Implement the declared callback method.

```
// Listen for SDK events, and print logs for error messages such as "Current applic
// Listen to the "onError" event, and print logs for errors such as "Camera is not 
void CTRTCDemoApp::onError(TXLiteAVError errCode, const char* errMsg, void* extraIn
        if (errCode == ERR_CAMERA_NOT_AUTHORIZED) {
                printf("Current application is not authorized to use the camera");
        }
}

// Listen for SDK events, and print logs for warning messages
// Listen to the "onWarning" event, and print logs for errors such as "WARNING_VIDE
void CTRTCDemoApp::onWarning(TXLiteAVWarning warningCode, const char* warningMsg, v
```

```
        if (warningCode == WARNING_VIDEO_RENDER_FAIL) {
                printf("WARNING_VIDEO_RENDER_FAIL");
        }

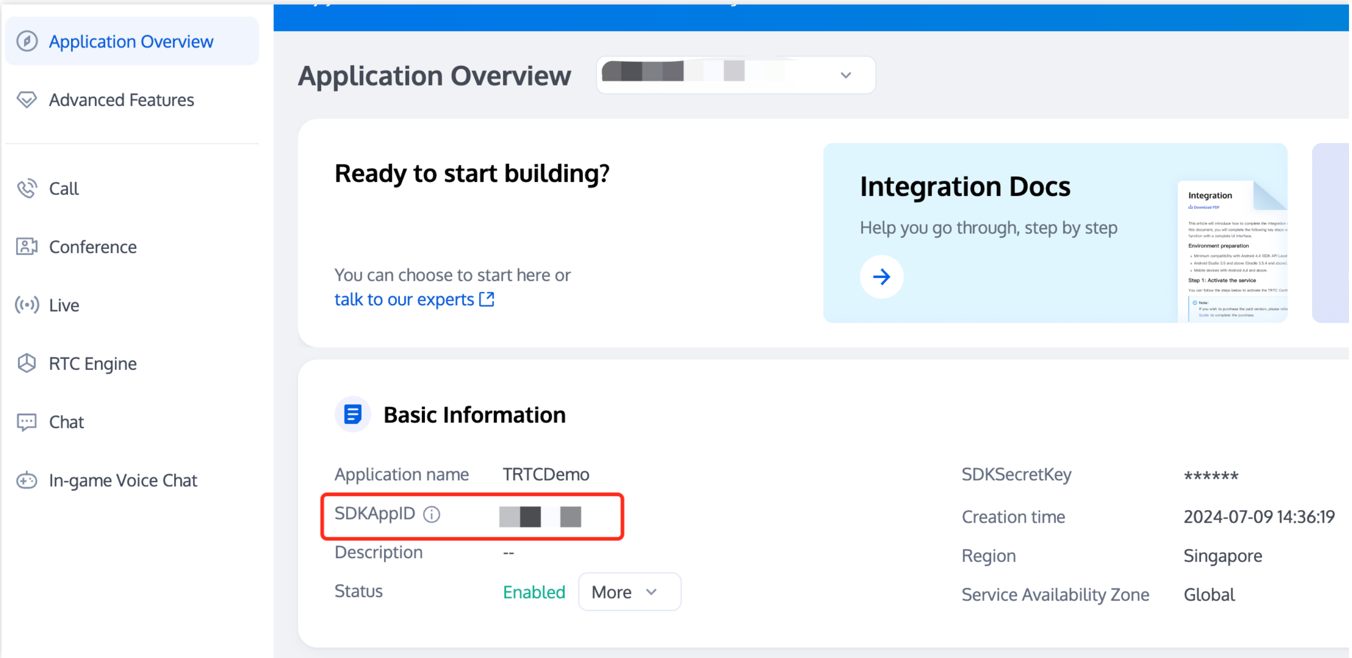
}

// Listen for the `onEnterRoom` event of the SDK to get the room entry result
// override to the onEnterRoom event of the SDK and learn whether the room is succe
void CTRTCDemoApp::onEnterRoom(int result) {
        if (result > 0) {
                printf("Enter room succeed");
        }
        else {
                printf("Enter room failed");
        }
}

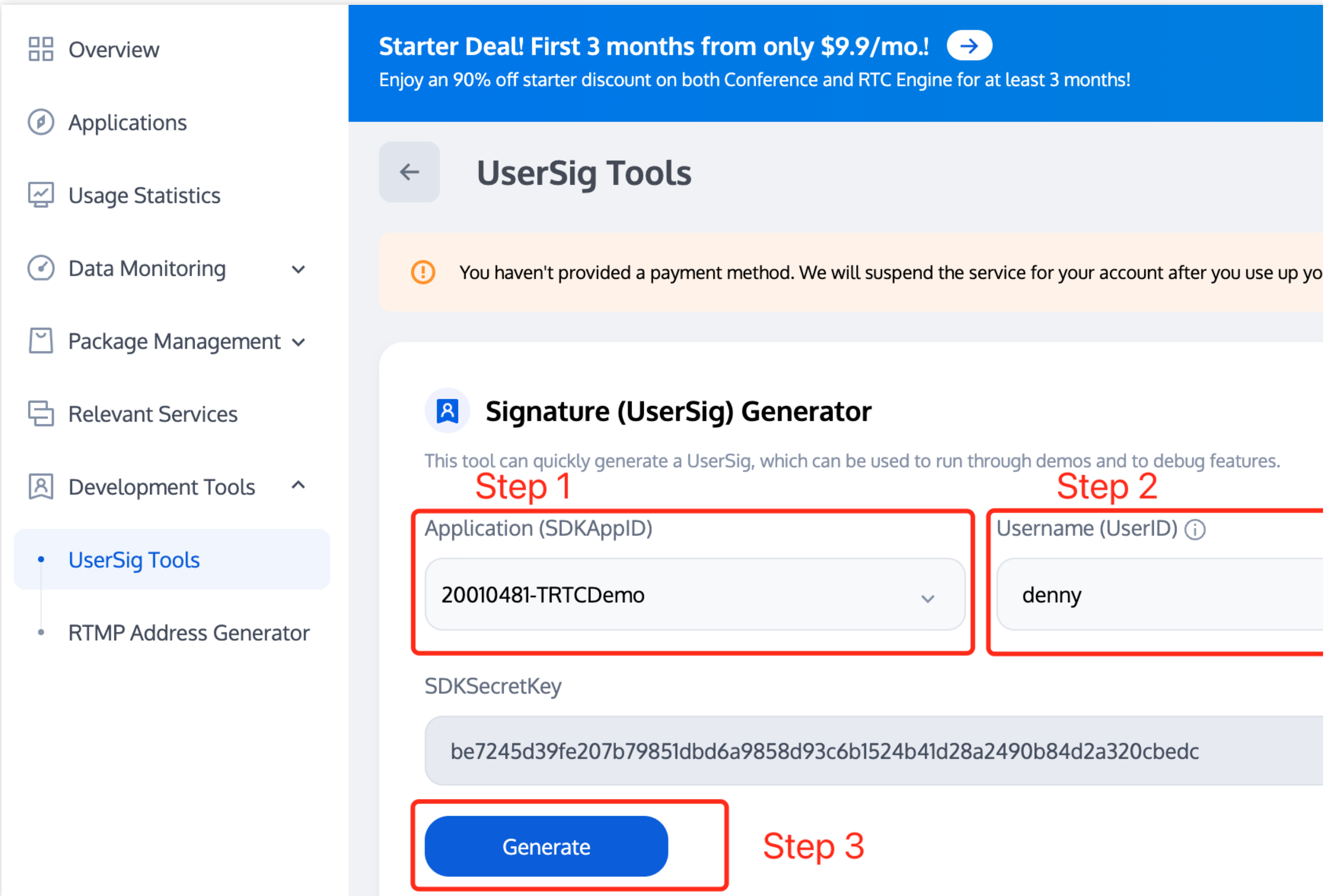
// Listen for the `onExitRoom` event of the SDK to get the room exit result
// override to the onExitRoom event of the SDK and learn whether the room is succes
void CTRTCDemoApp::onExitRoom(int reason) {
    if (reason == 0) {
        printf("Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        printf("Kicked out of the current room by server through the restful api...
    } else if (reason == 2) {
        printf("Current room is dissolved by server through the restful api...");
    }
}
```

Step 4. Enter the room

1. Click `Create Application` in the Tencent RTC console to get the **SDKAppID** under **Application Overview**.



2. Select **SDKAppID** down in the **UserSig Tools**, enter your **UserID**, and click `Generate` to get your own **UserSig**.



3. After setting the incoming parameter **TRTCParams** in the `CTRTCDemo::InitInstance()` method, call the `enterRoom` interface to enter the room.

As an Anchor：

```
BOOL CTRTCDemo::InitInstance()
{
    // ...other codes

    // Assemble TRTC room entry parameters. Replace the field values in `TRTCParams
    // Replace each field in TRTCParams with your own parameters
    liteav::TRTCParams trtcParams;
```

```
    trtcParams.sdkAppId = 1400000123;  // Replace with your own SDKAppID
    trtcParams.userId = "denny";       // Replace with your own user ID
    trtcParams.roomId = 123321;        // Replace with your own room number
    trtcParams.userSig = "xxx";        // Replace with your own userSig
    trtcParams.role = liteav::TRTCRoleAnchor;

    // If your scenario is live streaming, set the application scenario to `TRTC_AP
    // If your application scenario is a group video call, use "TRTC_APP_SCENE_LIVE
    trtc_cloud_->enterRoom(trtcParams, liteav::TRTCAppSceneLIVE);

    // ...other codes
}
```

As an audience:


```
BOOL CTRTCDemo::InitInstance()
{
    // ...other codes

    // Assemble TRTC room entry parameters. Replace the field values in `TRTCParams
    // Replace each field in TRTCParams with your own parameters
    liteav::TRTCParams trtcParams;
    trtcParams.sdkAppId = 1400000123;   // Replace with your own SDKAppID
    trtcParams.userId = "denny";        // Replace with your own user ID
    trtcParams.roomId = 123321;         // Replace with your own room number
    trtcParams.userSig = "xxx";         // Replace with your own userSig
```



```
    trtcParams.role = liteav::TRTCRoleAudience;

    // If your scenario is live streaming, set the application scenario to `TRTC_AP
    // If your application scenario is a group video call, use "TRTC_APP_SCENE_LIVE
    trtc_cloud_->enterRoom(trtcParams, liteav::TRTCAppSceneLIVE)

    // ...other codes
}
```

Note：

If you enter the room as an **audience**, **sdkAppId** and **roomId** need to be the same as on the anchor side, while **userId** and **userSig** need to be replaced with your own values.

Step 5. Turn on Camera

1. In the resource file **IDD_TRTCDEMO_DIALOG**, click **Toolbox** in the left border and add **Picture Control** to the dialog box.
2. Select **Properties** from the right-click menu and select **AFX_IDC_PICTURE** for ID.
3. Call the `setLocalRenderParams` in `CTRTCDemo::onEnterRoom()` to set the rendering parameters of the local preview, then call the `startLocalPreview` to open the camera preview, as shown in the following code:

```
void CTRTCDemo::onEnterRoom(int result) {
        if (result > 0) {
                // Set the preview mode of the local video image: Enable horizontal
                liteav::TRTCRenderParams render_params;
                render_params.mirrorType = liteav::TRTCVideoMirrorType_Enable;
                render_params.fillMode = TRTCVideoFillMode_Fill;
                trtc_cloud_->setLocalRenderParams(render_params);

        // Start a preview of the local camera
                CWnd* pLocalVideoView = m_pMainWnd->GetDlgItem(AFX_IDC_PICTURE);
                auto local_view = (liteav::TXView)(pLocalVideoView->GetSafeHwnd());
```

```
                trtc_cloud_->startLocalPreview(local_view);

                printf("Enter room succeed");
        }
        else {
                printf("Enter room failed");
        }
}
```

Step 6. Turn on microphone

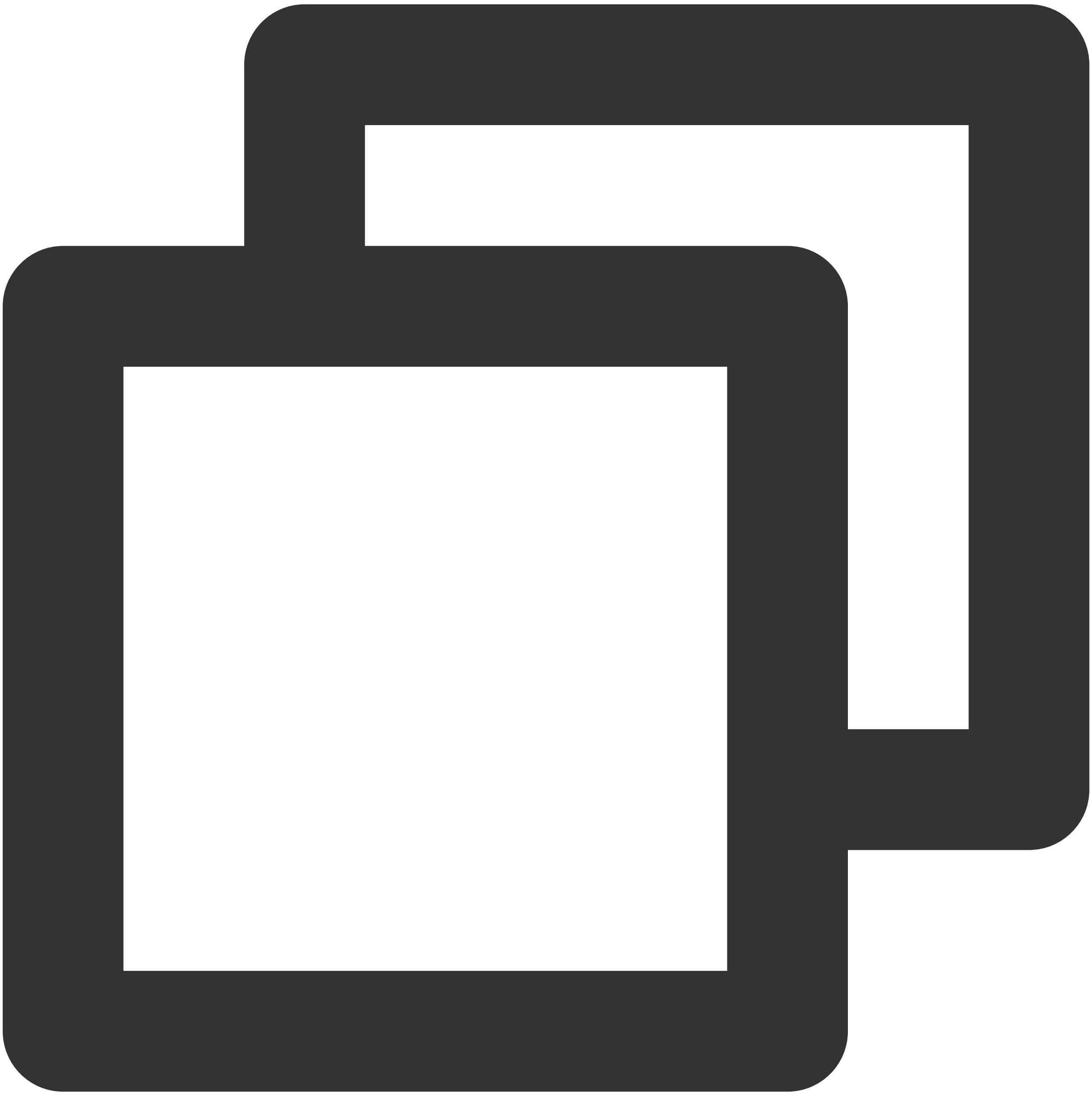
Call `startLocalAudio` to enable microphone capture. This interface requires you to determine the capture mode by the `quality` parameter. It is recommended to **select one of the following modes that is suitable for your project according to your needs**.

```
// Enable microphone acquisition and set the current scene to: Voice mode
// For high noise suppression capability, strong and weak network resistance
trtc_cloud_->startLocalAudio(TRTCAudioQualitySpeech);

// Enable microphone acquisition, and set the current scene to: Music mode
// For high fidelity acquisition, low sound quality loss, recommended to use with p
trtc_cloud_->startLocalAudio(TRTCAudioQualityMusic);
```

Step 7. Play/stop video streaming

1. Follow **steps 1-4** to create a new project and enter denny's room as an **audience**.
2. Add a **Picture Control** to the resource file **IDD_TRTCDEMO_DIALOG** and select the **ID** as **AFX_IDC_PICTURE**.
3. Call the `startRemoteView` in the `CTRTCDemo::onEnterRoom()` method to play the video of the remote user.



```
if (result > 0) {
    // Start a preview of the local camera
    CWnd* pVideoView = m_pMainWnd->GetDlgItem(AFX_IDC_PICTURE);
```

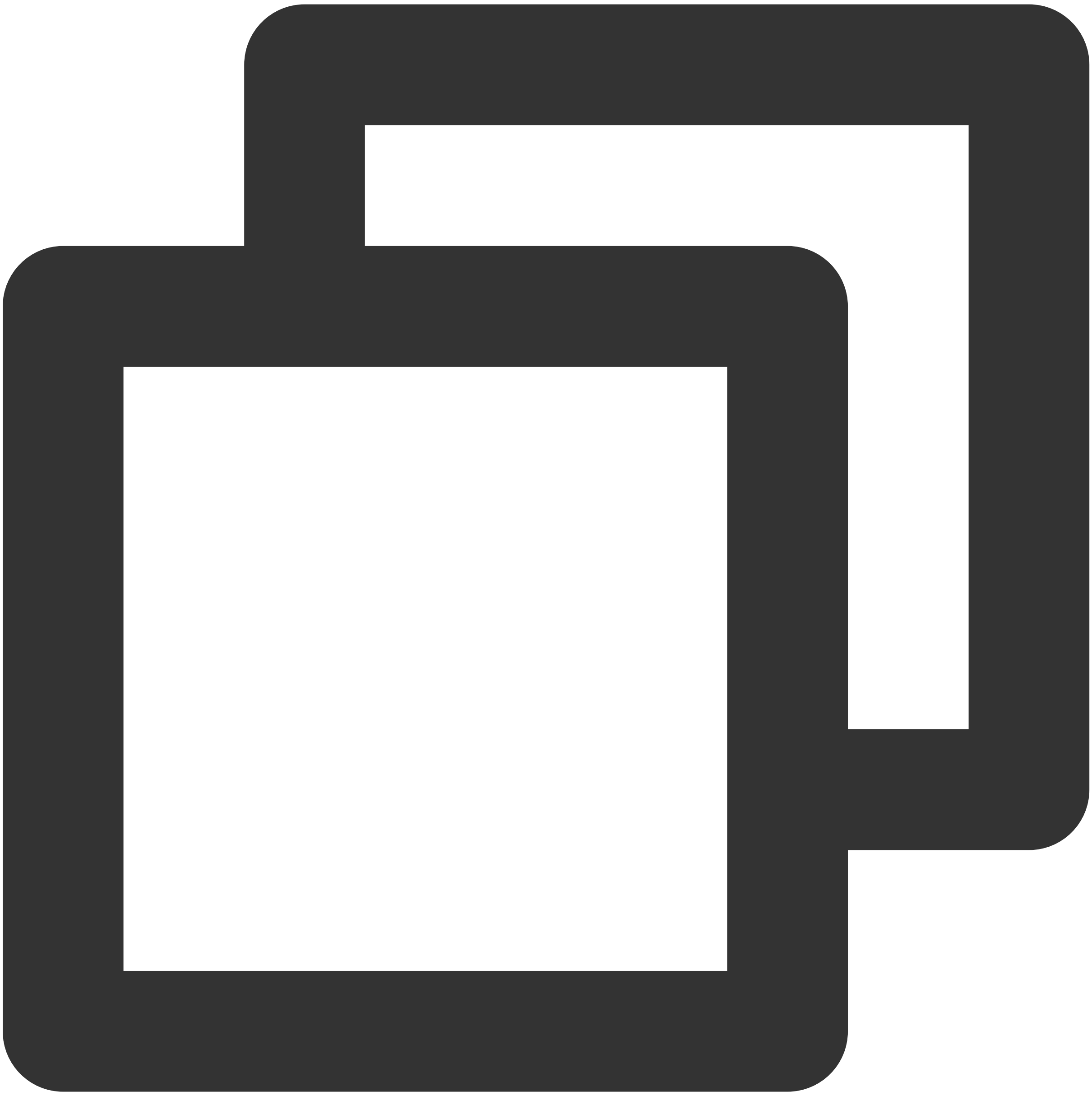


```
    auto video_view = (liteav::TXView)(pVideoView->GetSafeHwnd());

    // Play denny's camera footage
    trtc_cloud_->startRemoteView("denny", liteav::TRTCVideoStreamTypeBig, video_vie

    printf("Enter room succeed");
}
```

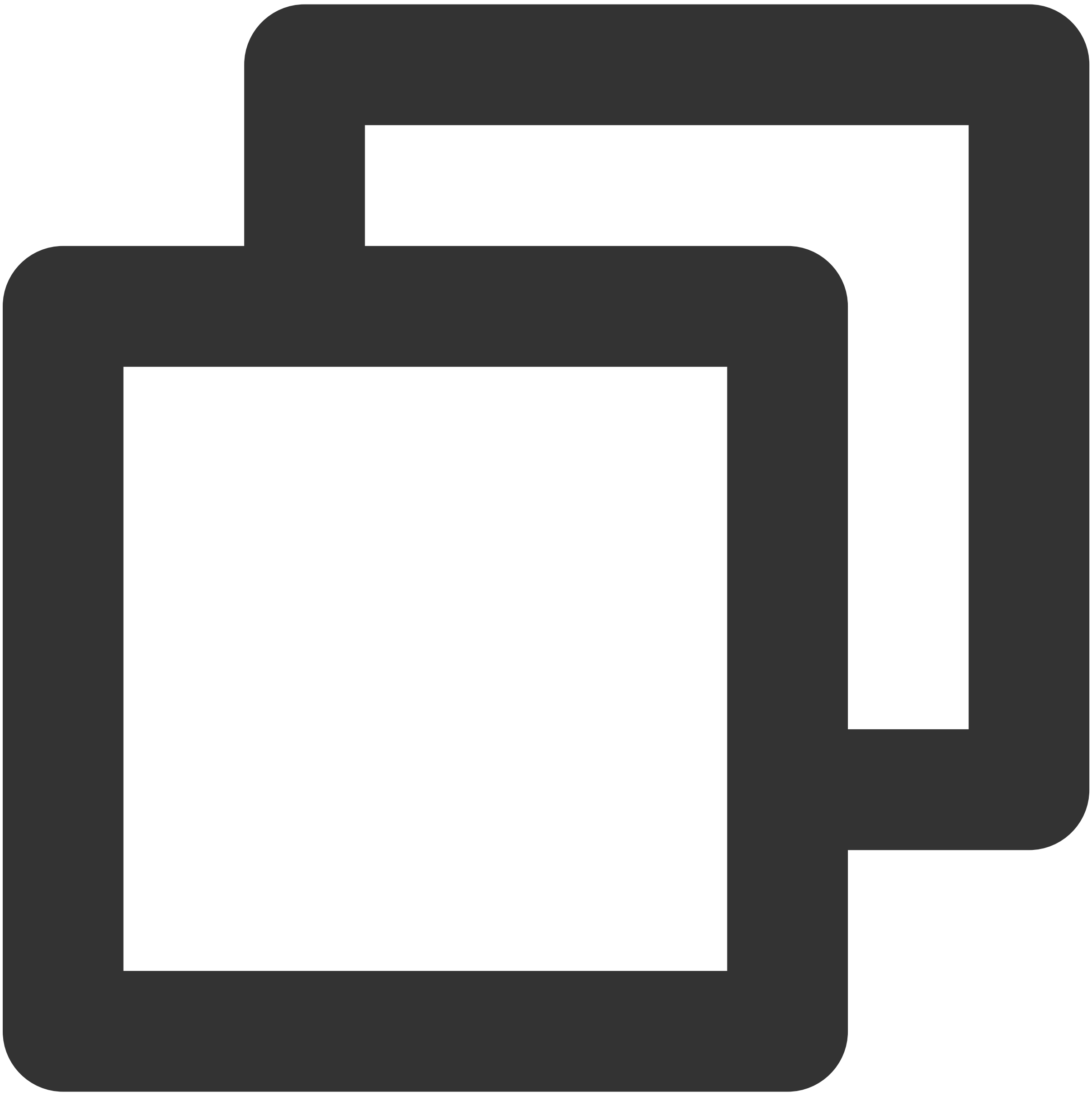
Then, you can call the `stopRemoteView` to stop the videos of a remote user. Alternatively, you can also stop the videos of all remote users via the `stopAllRemoteView` .




```
// Stop denny's camera footage
trtc_cloud_->stopRemoteView("denny", liteav::TRTCVideoStreamTypeBig);
// Stop all camera footages
trtc_cloud_->stopAllRemoteView();
```

Step 8. Play/stop the audio stream

Mute the voice of remote user denny by calling the `muteRemoteAudio("denny", true)` .




```
// Mute user with id denny
trtc_cloud_->muteRemoteAudio("denny", true);
```

You can also unmute him later by calling the `muteRemoteAudio("denny", false)` .

```
// Unmute user with id denny
trtc_cloud_->muteRemoteAudio("denny", false);
```

Step 9. Exit the room

Call the exitRoom to exit the current room, the SDK will notify you after the check-out through the `onExitRoom(int reason)` callback event.

```
// Exit current room
trtc_cloud_->exitRoom();
```

FAQs

API Reference at API Reference.

If you encounter any issues during integration and use, please refer to Frequently Asked Questions.

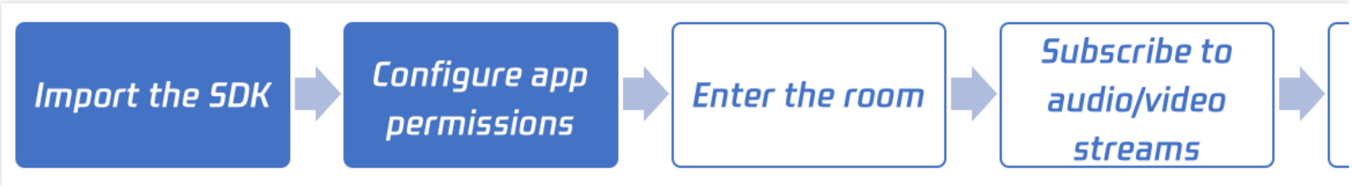
Contact us

If you have any suggestions or feedback, please contact `info_rtc@tencent.com` .

01. Importing the SDK
iOS

Last updated：2024-05-21 15:05:29

This document describes how to import the SDK into your project.



Environment Requirements

Xcode 9.0 or later

iPhone or iPad with iOS 9.0 or later

A valid developer signature for your project

Step 1. Import the SDK

You can use CocoaPods or download and import the SDK manually into your project.

Method 1. Use CocoaPods

1. **Install CocoaPods.**

Enter the following command in a terminal window (you need to install Ruby on your Mac first):

```
sudo gem install cocoapods
```

2. **Create a Podfile.**

Go to the directory of your project and enter the following command to create a Podfile in the directory.

```
pod init
```

3. **Edit the Podfile.**

Choose an appropriate edition according to your project needs and edit the Podfile:

Option 1: Lite

The installation package is the smallest but only supports two features: real-time communication (TRTC) and live player (TXLivePlayer). To choose this version, edit the Podfile as follows:

```
platform :ios, '8.0'

 target 'App' do
 pod 'TXLiteAVSDK_TRTC', :podspec => 'https://liteav.sdk.qcloud.com/pod/liteavsdks
 end
```

Option 2: Professional

The installation package includes real-time communication (TRTC), live player (TXLivePlayer), RTMP streaming (TXLivePusher), VOD player (TXVodPlayer), short video recording and editing (UGSV), and many other features. To choose this edition, edit the Podfile as follows:

```
platform :ios, '8.0'

 target 'App' do
 pod 'TXLiteAVSDK_Professional', :podspec => 'https://liteav.sdk.qcloud.com/pod/li
 end
```

4. **Update the local repository and install the SDK**

Enter the following command in a terminal window to update the local repository and install the SDK:

```
pod install
```

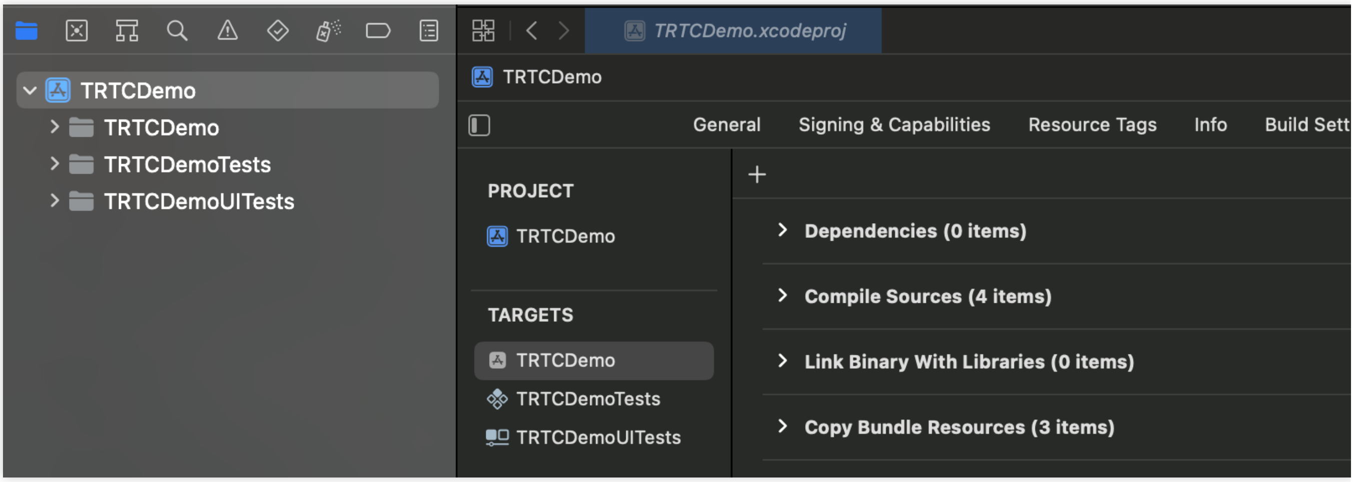
Or, run this command to update the local repository:

```
pod update
```

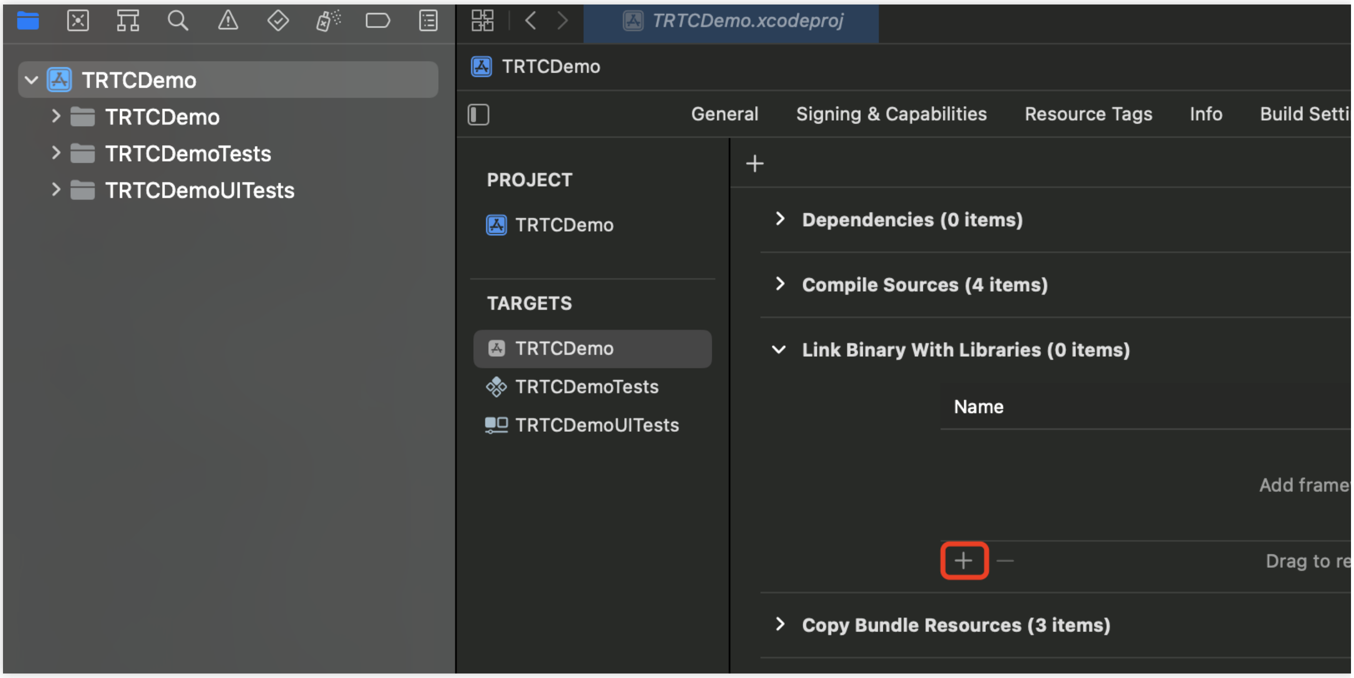
An XCWORKSPACE project file integrated with the TRTC SDK will be generated. Double-click to open it.

Method 2. Download the SDK and import it manually

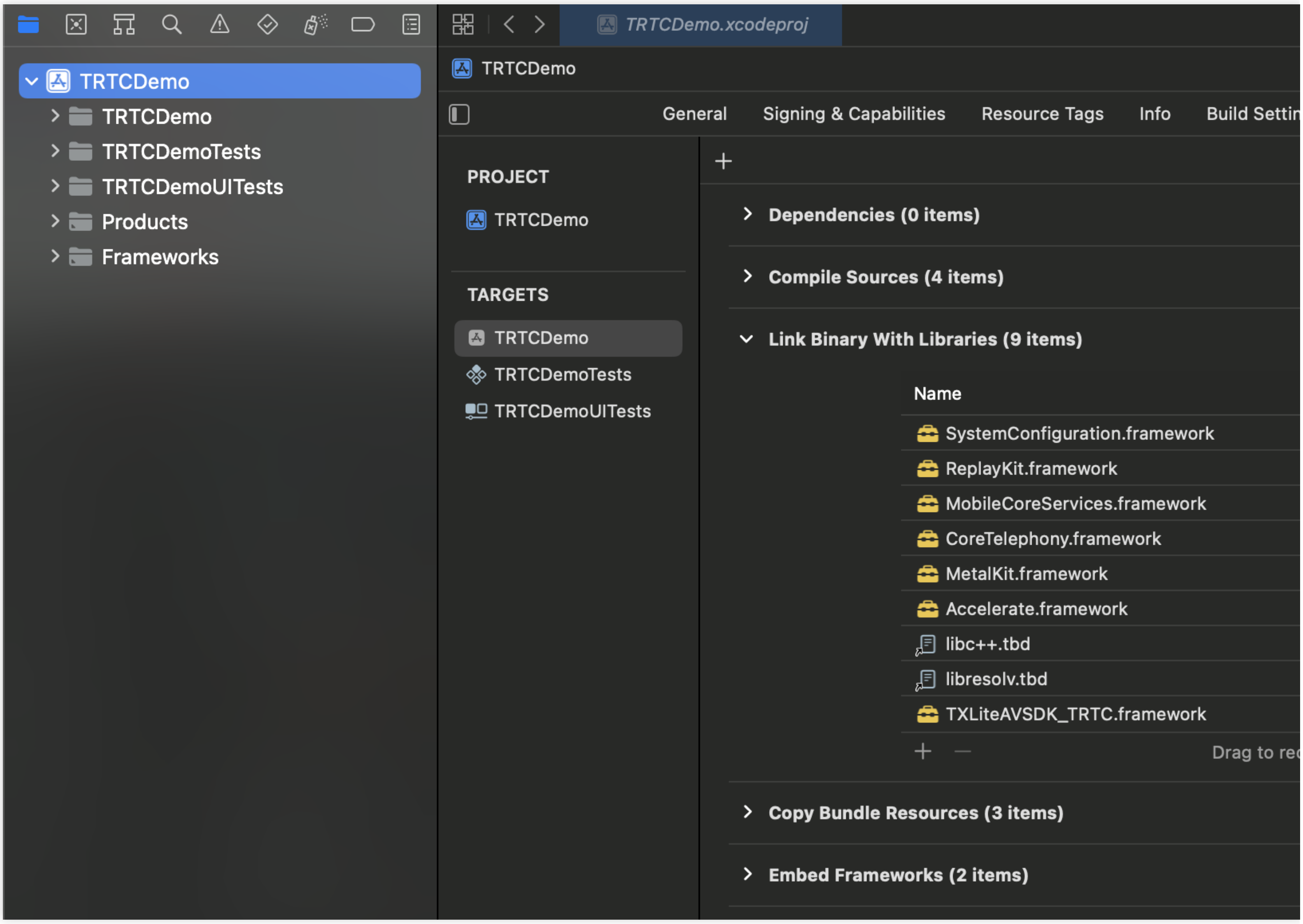
1. Download and decompress the SDK package.
2. Open your Xcode project, select the target you want to run, and click **Build Phases**.



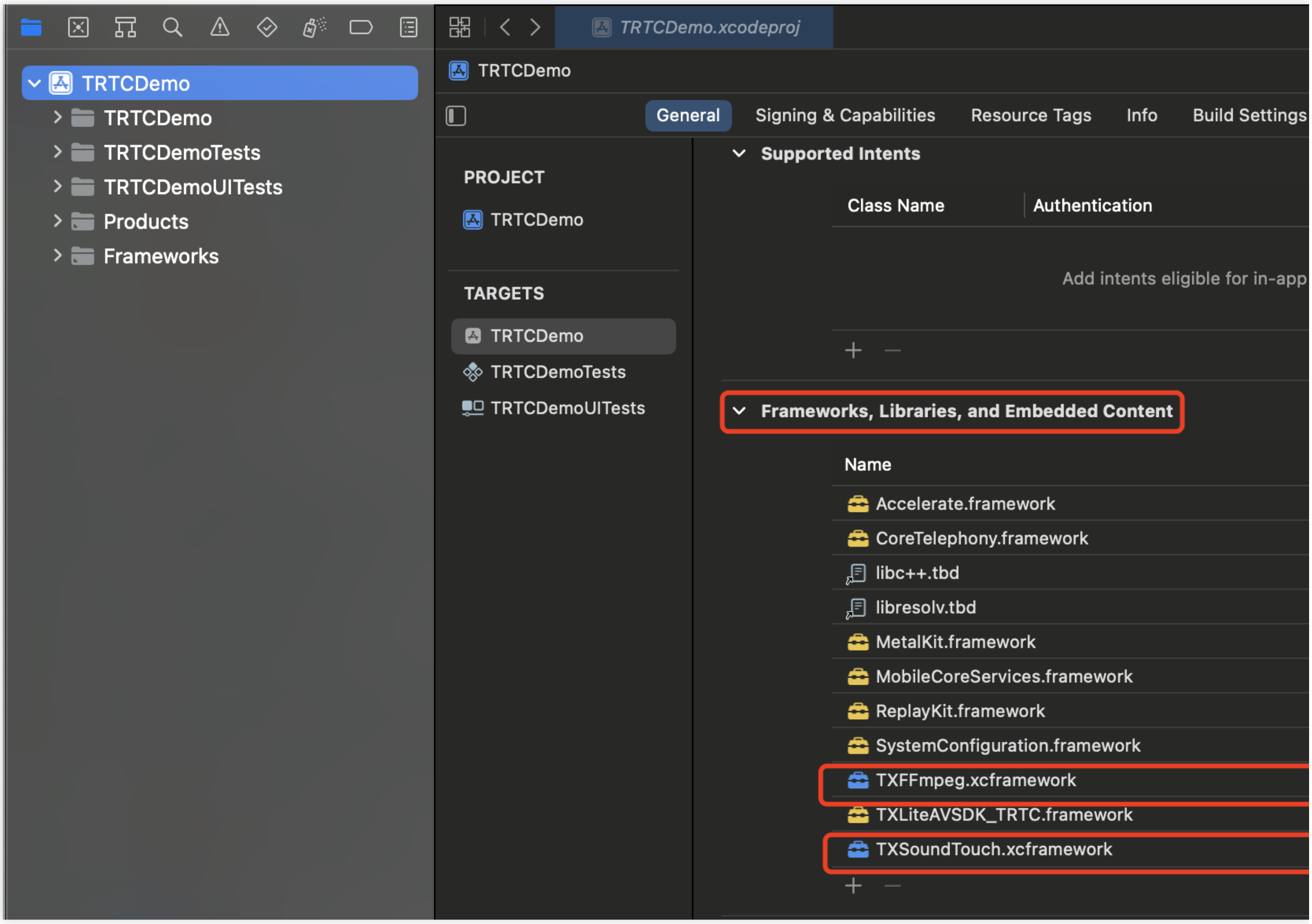
3. Expand **Link Binary With Libraries** and click the **+** icon at the bottom to add dependent libraries.



4. Add the downloaded `TXLiteAVSDK_TRTC.Framework` (or `TXLiteAVSDK_Professional.Framework`), `TXFFmpeg.xcframework` , `TXSoundTouch.xcframework` , and the frameworks they depend on: `GLKit.framework` , `AssetsLibrary.framework` , `SystemConfiguration.framework` , `libsqlite3.0.tbd` , `CoreTelephony.framework` , `AVFoundation.framework` , `OpenGLES.framework` , `Accelerate.framework` , `MetalKit.framework` , `libresolv.tbd` , `MobileCoreServices.framework` , `libc++.tbd` , `CoreMedia.framework` .



5. Click **General**, expand **Frameworks, Libraries, and Embedded Content**, and check if the dynamic libraries required by `TXLiteAVSDK_TRTC.framework` (**TXFFmpeg.xcframework** and **TXSoundTouch.xcframework**) have been added and set to **Embed & Sign**. If not, click **+** at the bottom to add them.

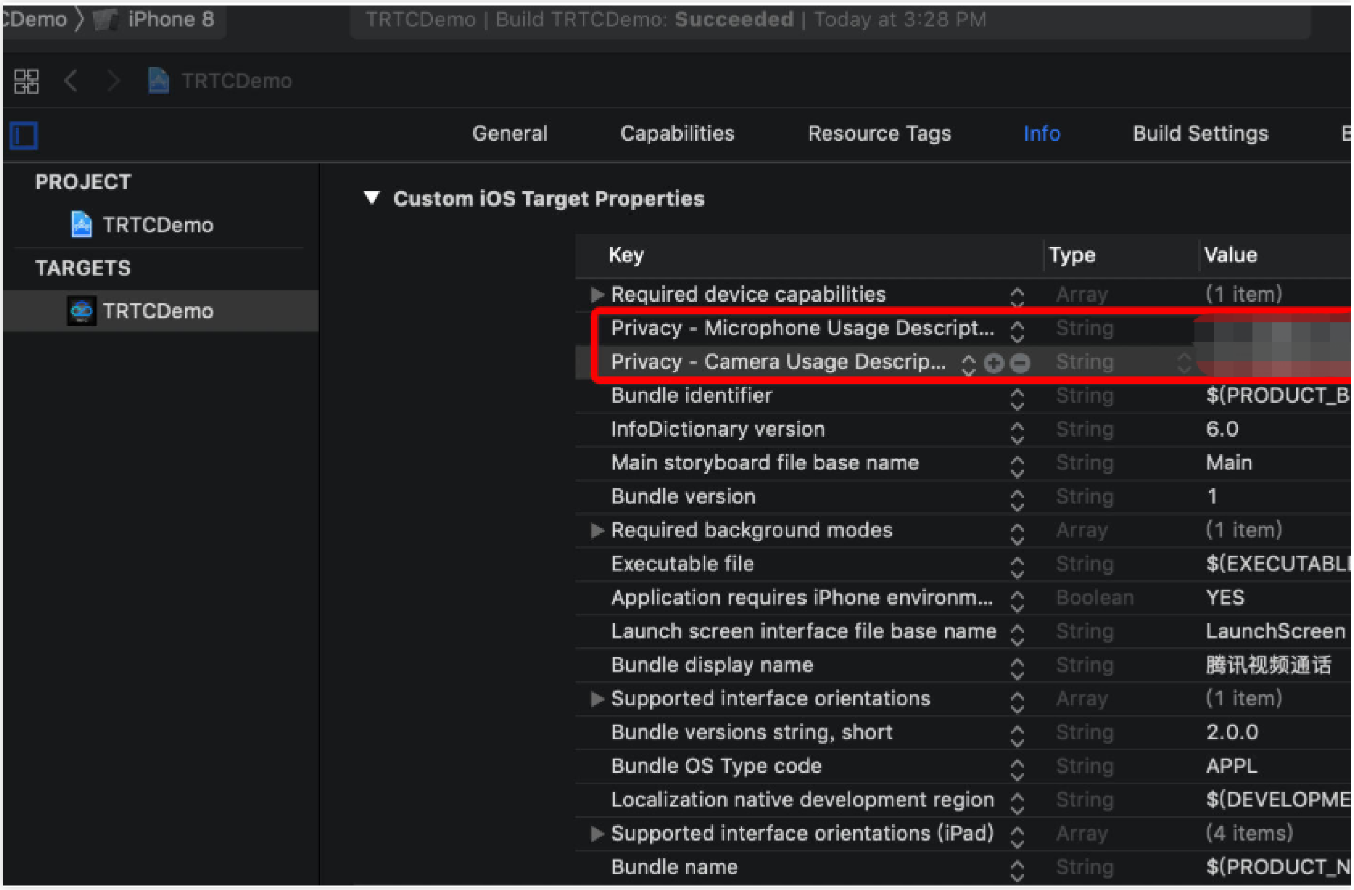


Step 2. Configure app permissions

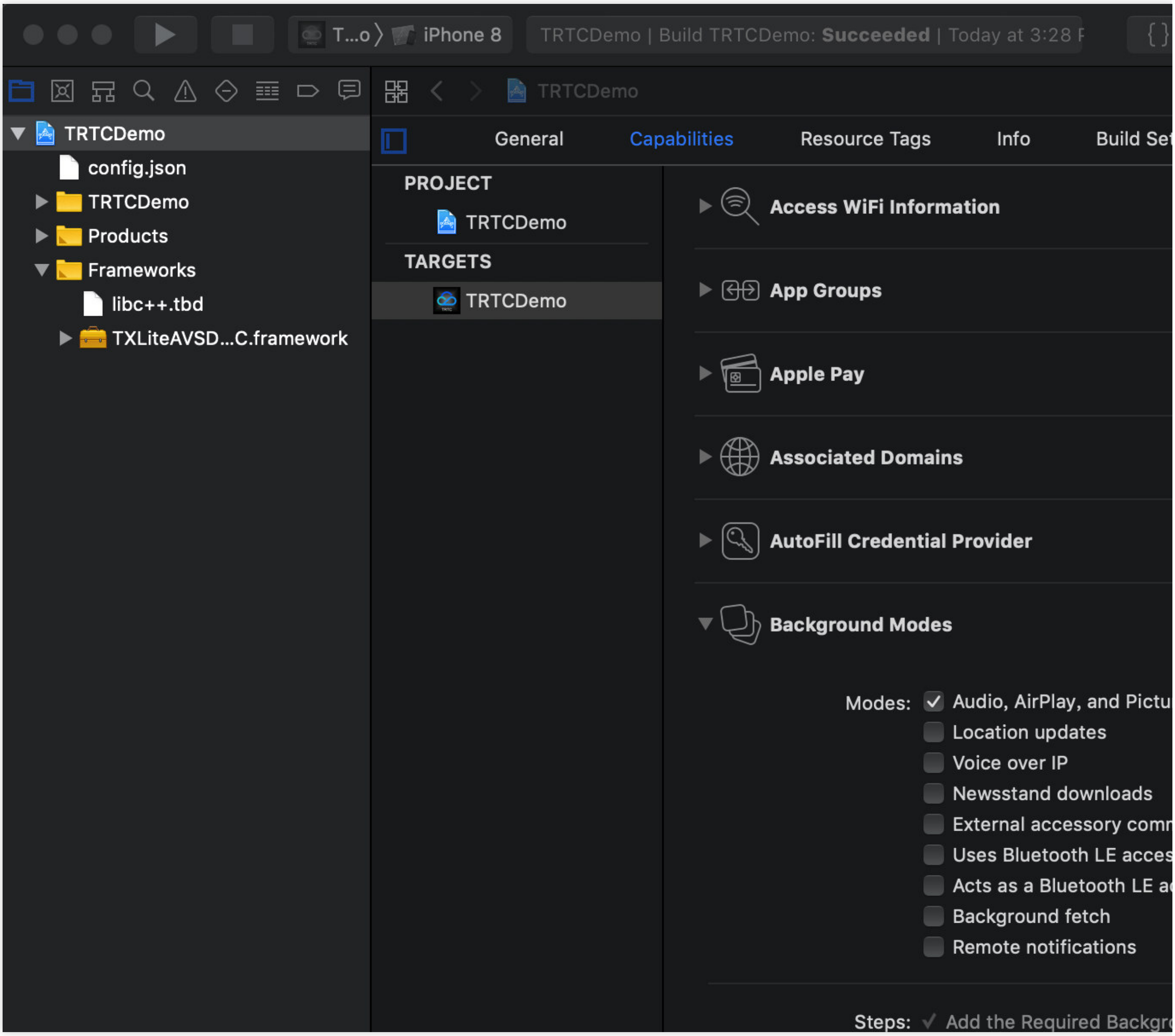
1. To use the audio/video features of the SDK, you need to grant the application mic and camera permissions. Add the two items below to `Info.plist` of your application. Their content is what users see in the mic and camera access pop-up windows.

Privacy - Microphone Usage Description. Include a statement specifying why mic access is needed.

Privacy - Camera Usage Description. Include a statement specifying why camera access is needed.



2. If you want the SDK to run in the background, select your project, under the **Capabilities** tab, set **Background Modes** to **ON**, and select **Audio, AirPlay, and Picture in Picture**.



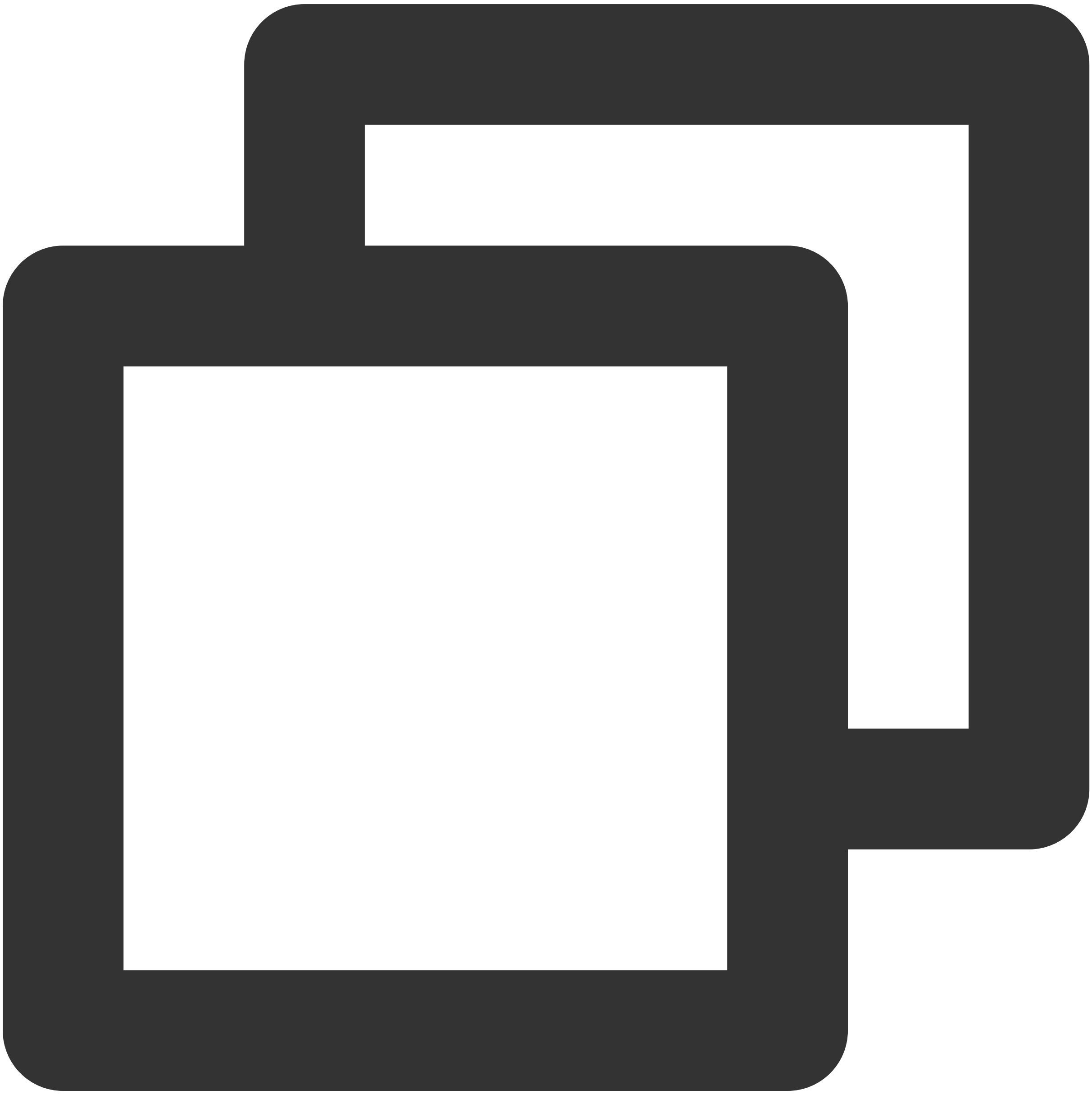
Step 3. Import the SDK into the project

After completing the first step of importing and the second step of granting device permissions, you can import the APIs provided by the SDK into your project.

Using Objective-C or Swift APIs

There are two ways to use the SDK in Objective-C or Swift:

Import the module: Import the SDK module in the files that will use the SDK APIs.



```
@import TXLiteAVSDK_TRTC;
```

Import the header file: Import the header file in the files that will use the SDK APIs.

```
#import "TXLiteAVSDK_TRTC/TRTCCloud.h"
```

Note:

For more information on how to use Objective-C APIs, see Overview.

Using C++ APIs (optional)

If your project imports the SDK through a cross-platform framework such as Qt or Electron, import the header files in the `TXLiteAVSDK_TRTC.framework/Headers/cpp_interface` directory:

```
#include "TXLiteAVSDK_TRTC/cpp_interface/ITRTCCloud.h"
```

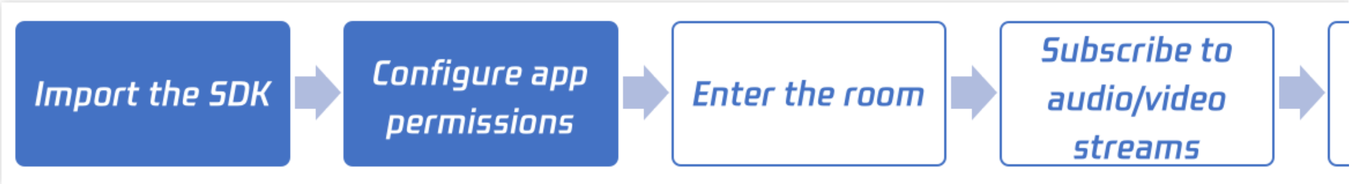
Note:

For more information on how to use C++ APIs, see Overview.

Android

Last updated：2024-05-21 15:05:29

This document describes how to import the SDK into your project.



Environment Requirements

Android Studio 3.5 or later.

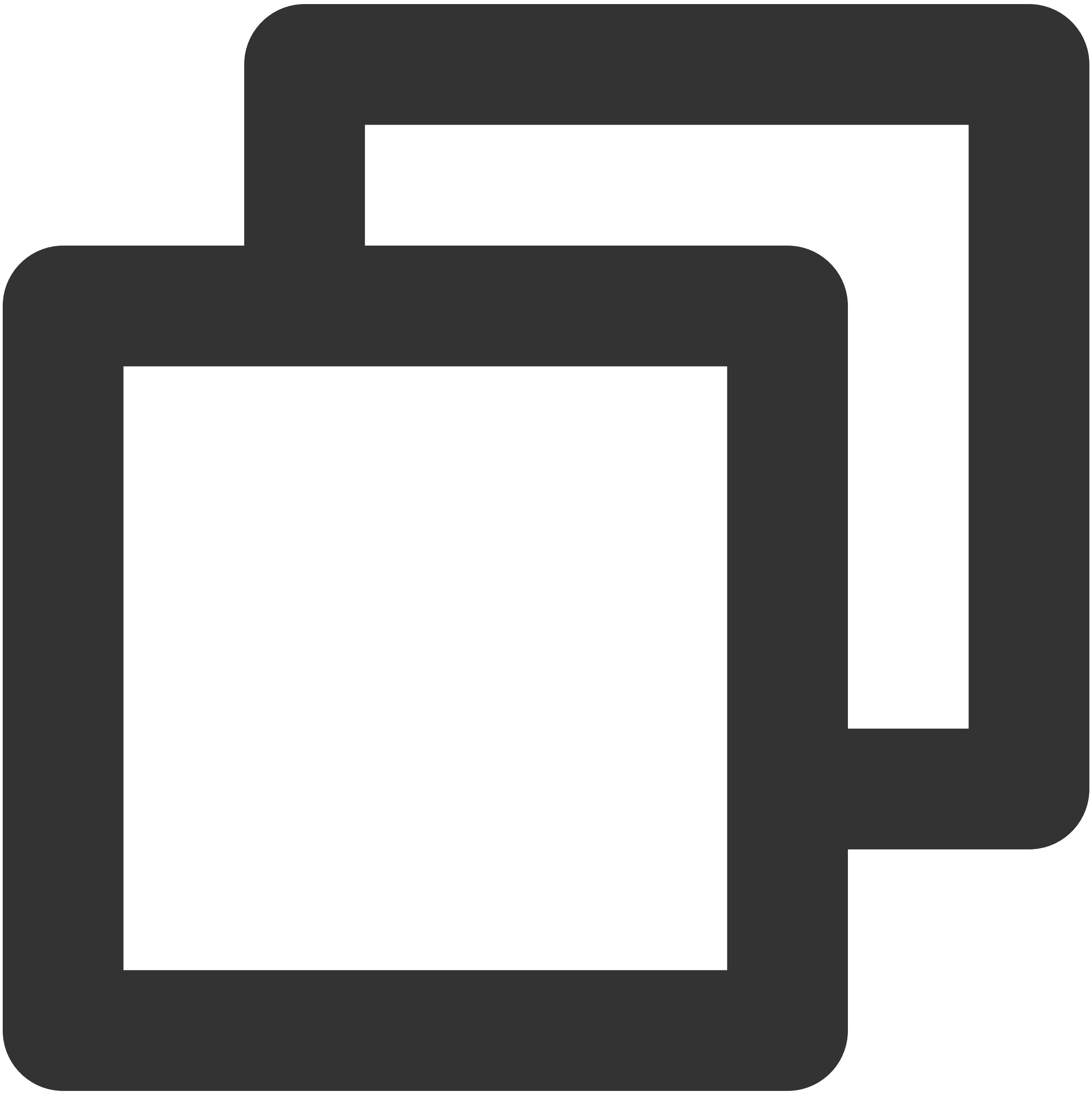
Android 4.1 (SDK API level 16) or later

Step 1. Import the SDK

Method 1. Automatic loading (aar)

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

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



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

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

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

Note:

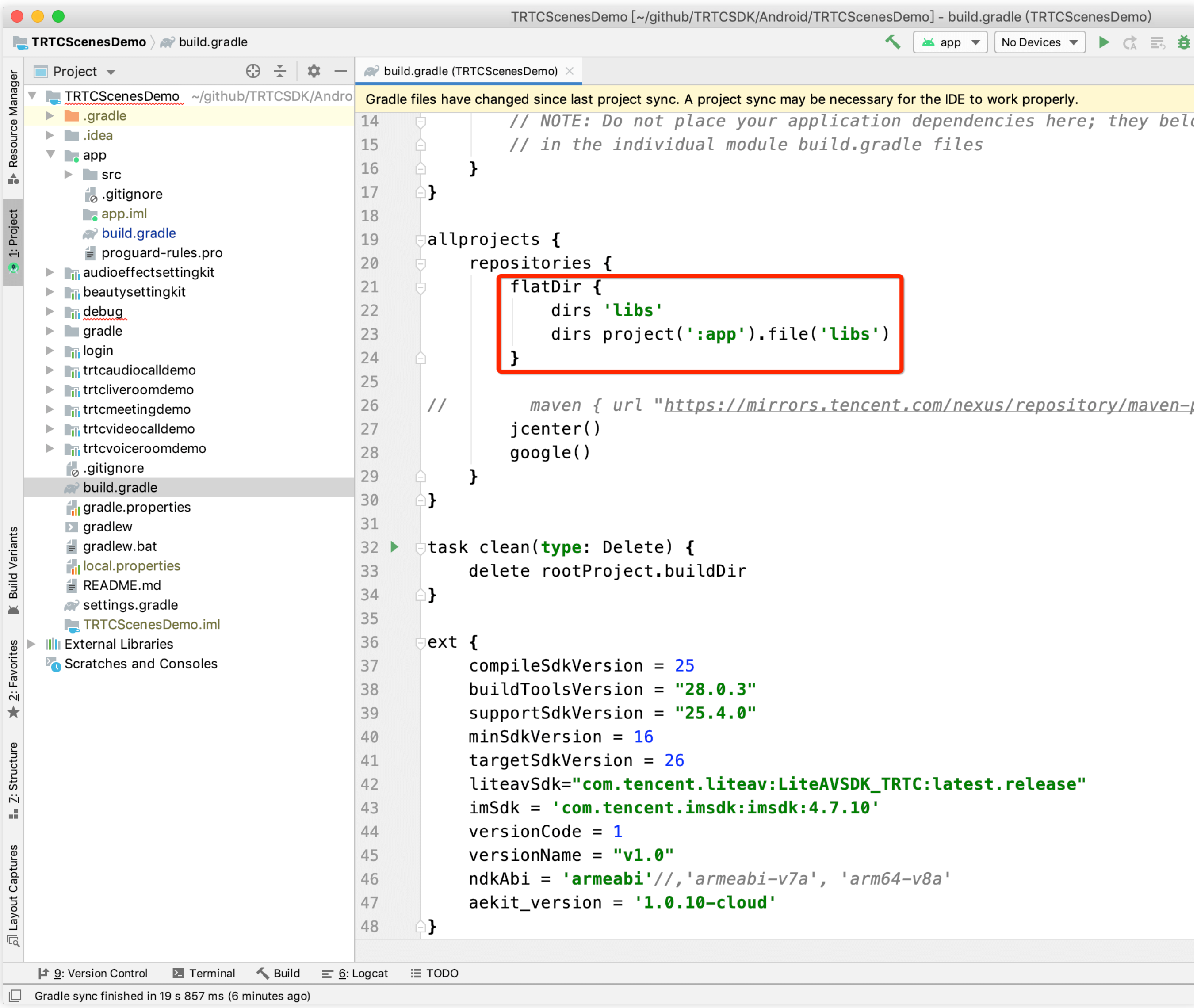
Currently, the TRTC SDK supports armeabi-v7a, and arm64-v8a.

3. Click

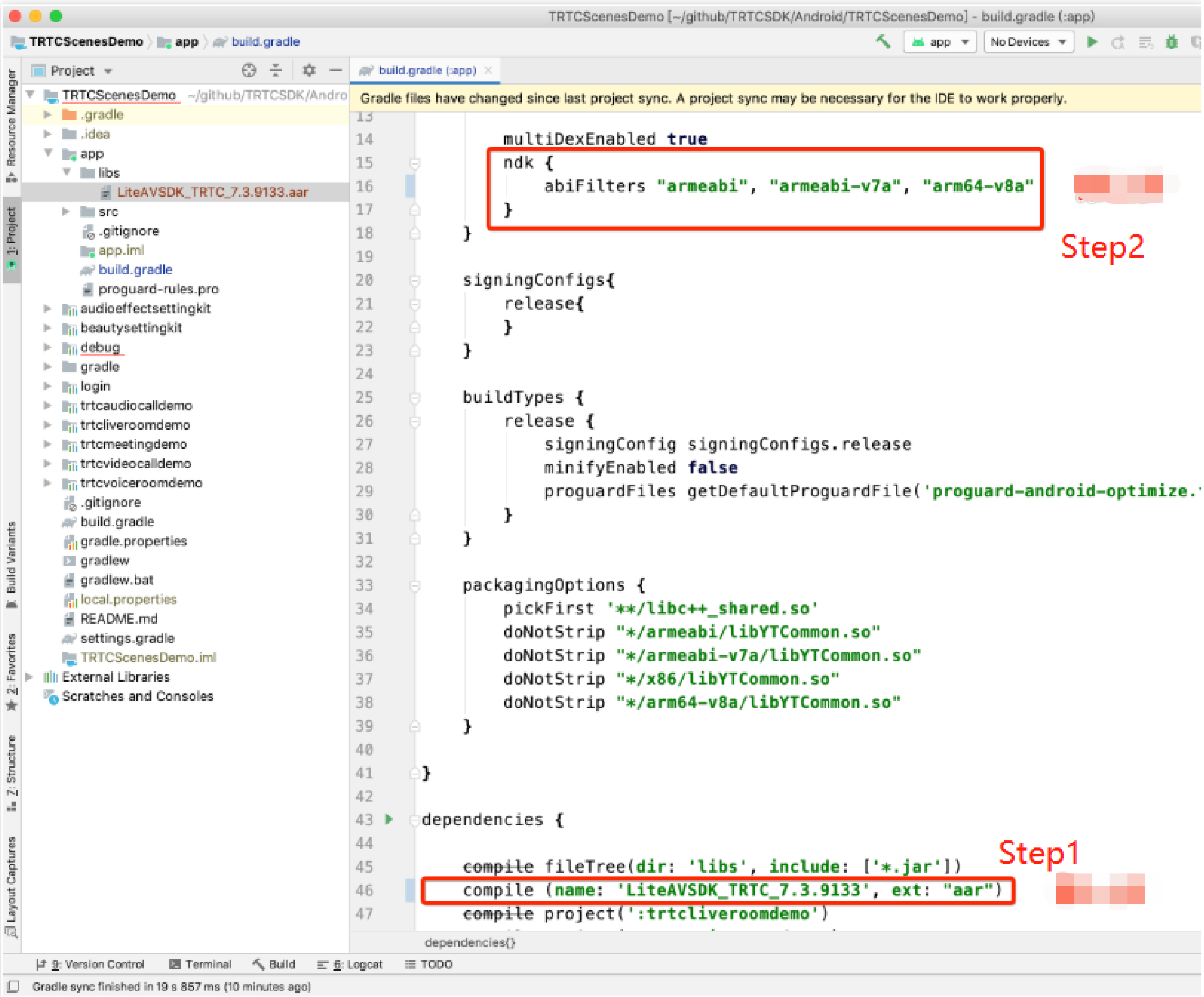
Sync Now to automatically download the SDK and integrate them into your project.

Method 2. Download the SDK and import it manually

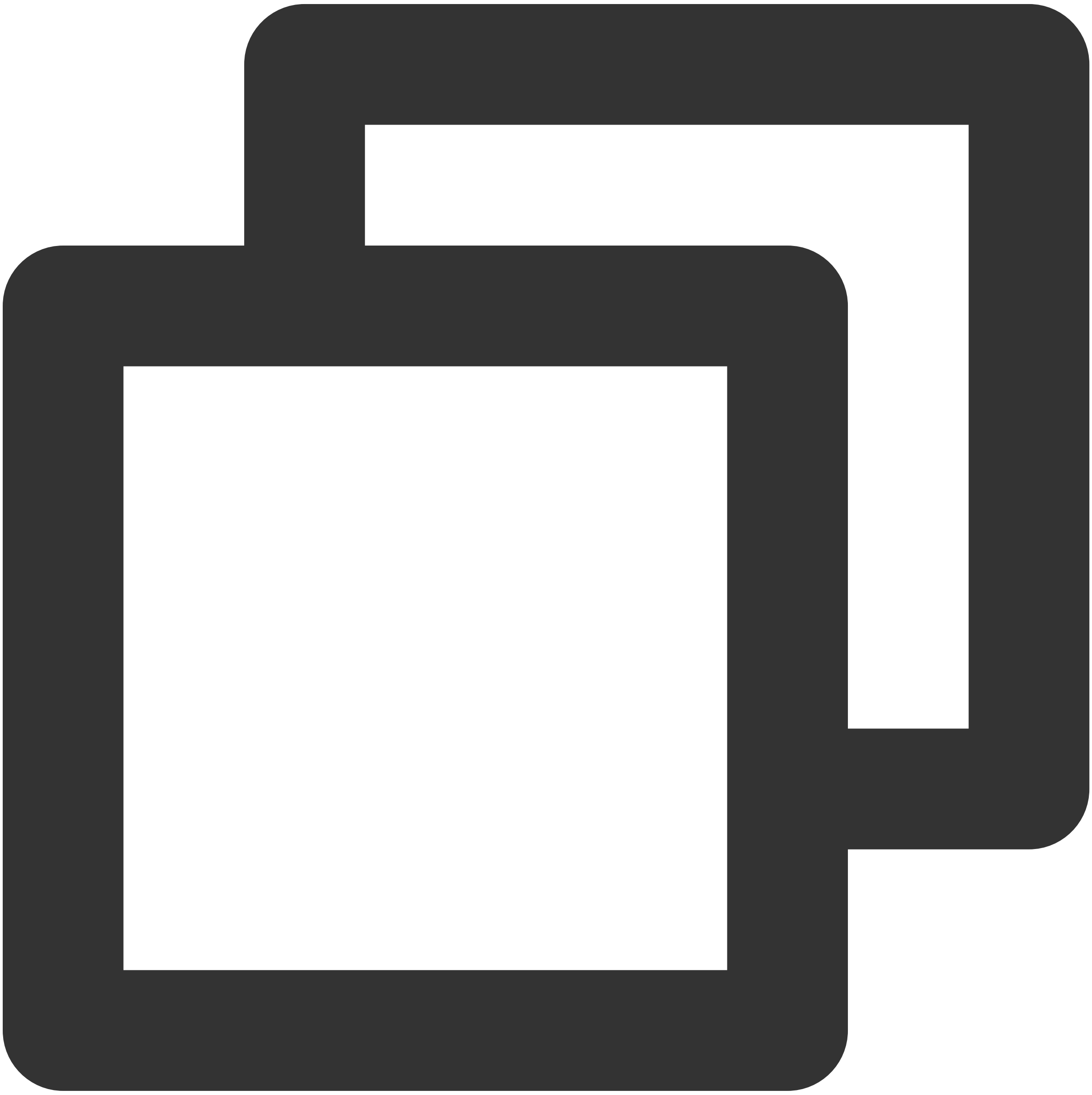
1. Download the SDK and decompress it locally.
2. Copy the decompressed AAR file to the **app/libs** directory of your project.
3. Add **flatDir** to `build.gradle` under your project's root directory to specify the local path for the repository.



4. Add code in `app/build.gradle` to import the AAR file.



5. In `defaultConfig` of `app/build.gradle` , specify the CPU architecture to be used by your application.



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

Note:

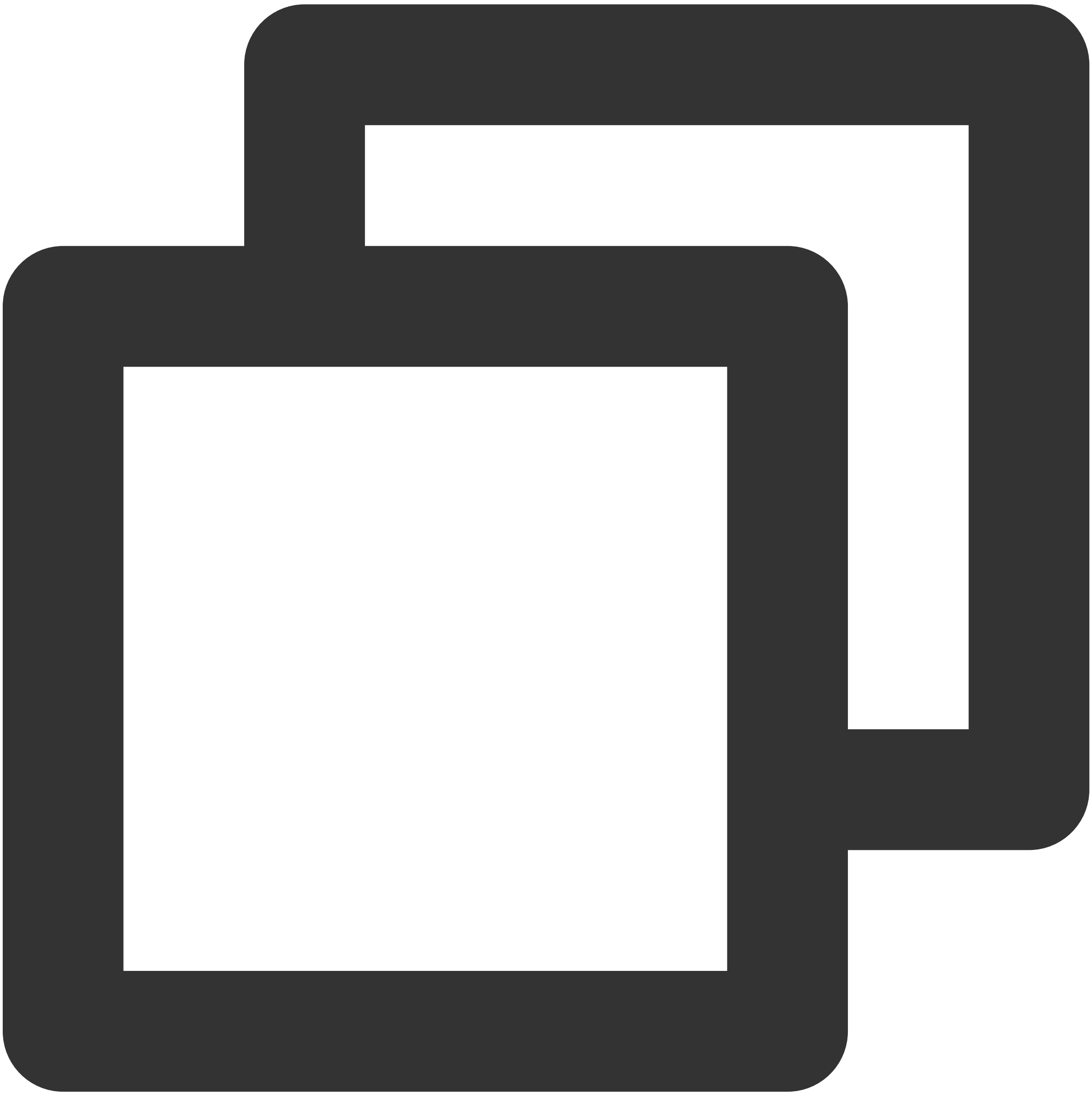
Currently, the TRTC SDK supports armeabi, armeabi-v7a, and arm64-v8a.

6. Click

Sync Now to integrate the TRTC SDK.

Step 2. Configure app permissions

Configure application permissions in `AndroidManifest.xml` . The TRTC SDK requires the following permissions:




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

Note:

Do not set `android:hardwareAccelerated="false"` . Disabling hardware acceleration will result in failure to render remote users' videos.

Step 3. Set obfuscation rules

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

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

Using SDK Through C++ APIs (Optional)

If you prefer to use C++ APIs instead of Java for development, you can perform this step. If you only use Java to call the TRTC SDK, skip this step.

1. First, you need to integrate the TRTC SDK by importing JAR and SO libraries as instructed above.

2. Copy the C++ header file in the SDK to the project (path: `SDK/LiteAVSDK_TRTC_xxx/libs/include`) and configure the `include` folder path and dynamic link to the SO library in `CMakeLists.txt` .

```
cmake_minimum_required(VERSION 3.6)

# Configure the C++ API header file path
include_directories(
      ${CMAKE_CURRENT_SOURCE_DIR}/include  # Copied from `SDK/LiteAVSDK_TRTC_xxx/lib
)

add_library(
```



```
      native-lib
      SHARED
      native-lib.cpp)

# Configure the path of the `libliteavsdk.so` dynamic library
add_library(libliteavsdk SHARED IMPORTED)
set_target_properties(libliteavsdk  PROPERTIES IMPORTED_LOCATION ${CMAKE_CURRENT_SO

find_library(
      log-lib
      log)

# Configure the dynamic link as `libliteavsdk.so`
target_link_libraries(
      native-lib
      libliteavsdk
      ${log-lib})
```

3. Use the namespace: The methods and types of cross-platform C++ APIs are all defined in the `trtc` namespace. To simplify your code, we recommend you use the `trtc` namespace.

```
using namespace trtc;
```

Note:

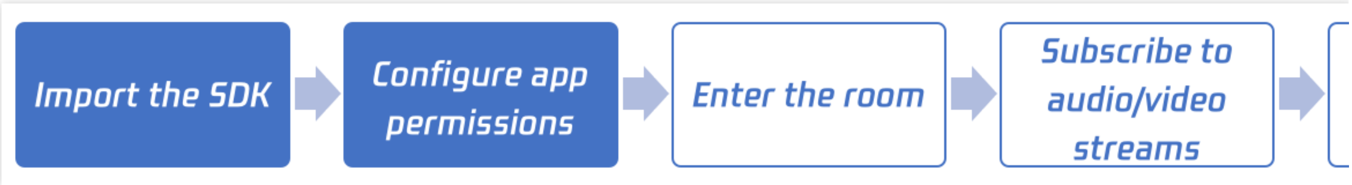
For more information on how to configure the Android Studio C/C++ development environments, see Add C and C++ code to your project.

Currently, only the TRTC edition of the SDK supports C++ APIs. For more information on how to use C++ APIs, see Overview.

macOS

Last updated：2024-05-21 15:05:29

This document describes how to quickly integrate the TRTC macOS SDK into your project.



Environment Requirements

Xcode 9.0 or later

A Mac computer with OS X 10.10 or later

A valid developer signature for your project

Step 1. Import the SDK

You can use CocoaPods to automatically load the SDK or download and import it manually into your project.

Method 1. Use CocoaPods

1. **Install CocoaPods.**

Enter the following command in a terminal window (you need to install Ruby on your Mac first):

```
sudo gem install cocoapods
```

2. **Create a Podfile.**

Go to the directory of your project and enter the following command to create a Podfile in the directory.

```
pod init
```

3. **Edit the Podfile.**

There are two ways to edit the Podfile:

Method 1: Use the pod path of the LiteAV SDK

```
platform :osx, '10.10'

target 'Your Target' do
pod 'TXLiteAVSDK_TRTC_Mac', :podspec => 'https://liteav.sdk.qcloud.com/pod/liteavsd
end
```

Method 2: Use CocoaPod's official source, which allows version selection

```
platform :osx, '10.10'
source 'https://github.com/CocoaPods/Specs.git'

target 'Your Target' do
pod 'TXLiteAVSDK_TRTC_Mac'
end
```

4. **Install and update the SDK.**

Enter the following command in a terminal window to install the SDK.

```
pod install
```

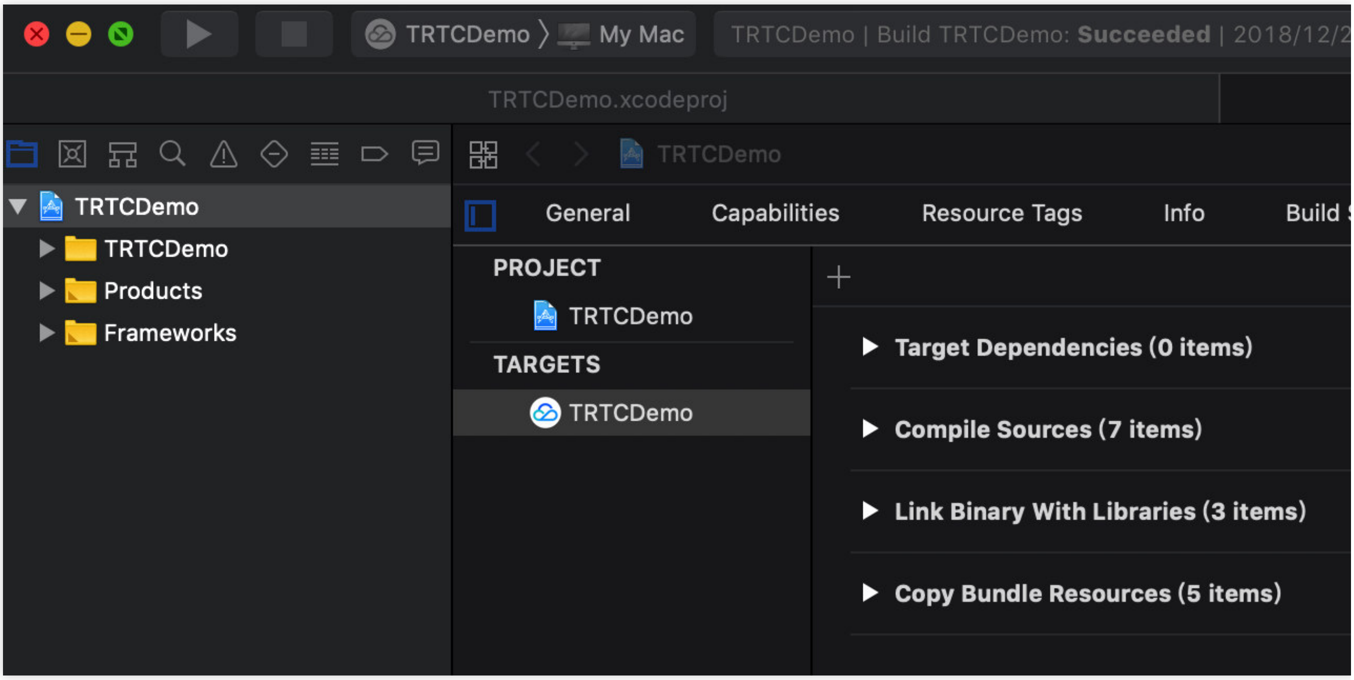
Or, run this command to update the local repository:


```
pod update
```

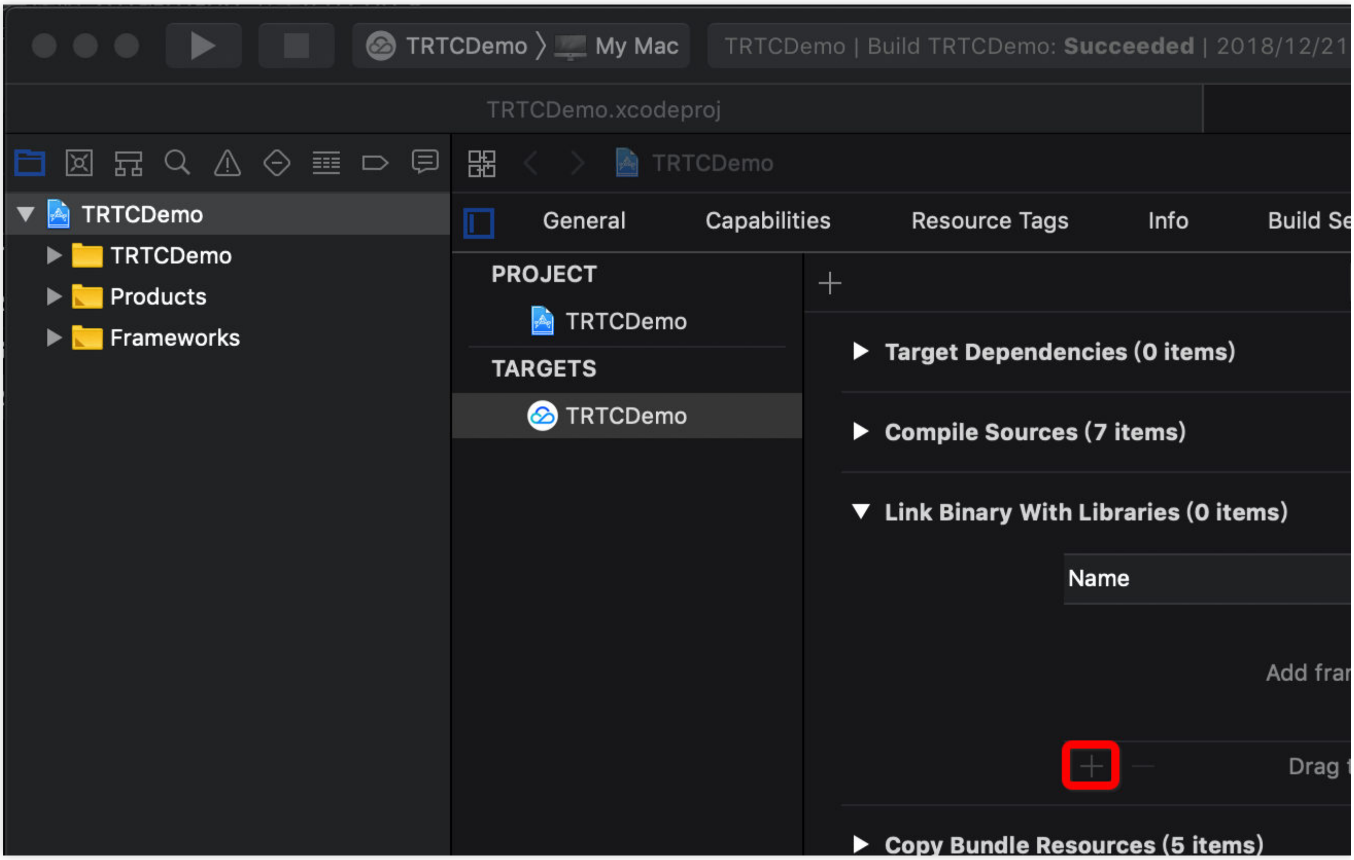
An XCWORKSPACE project file integrated with LiteAVSDK will be generated. Double-click to open the file.

Method 2. Manually integrate

1. Download the TRTC macOS SDK.
2. Open your Xcode project and import into it the framework downloaded in step 1.
3. Select the target you want to run and click **Build Phases**.

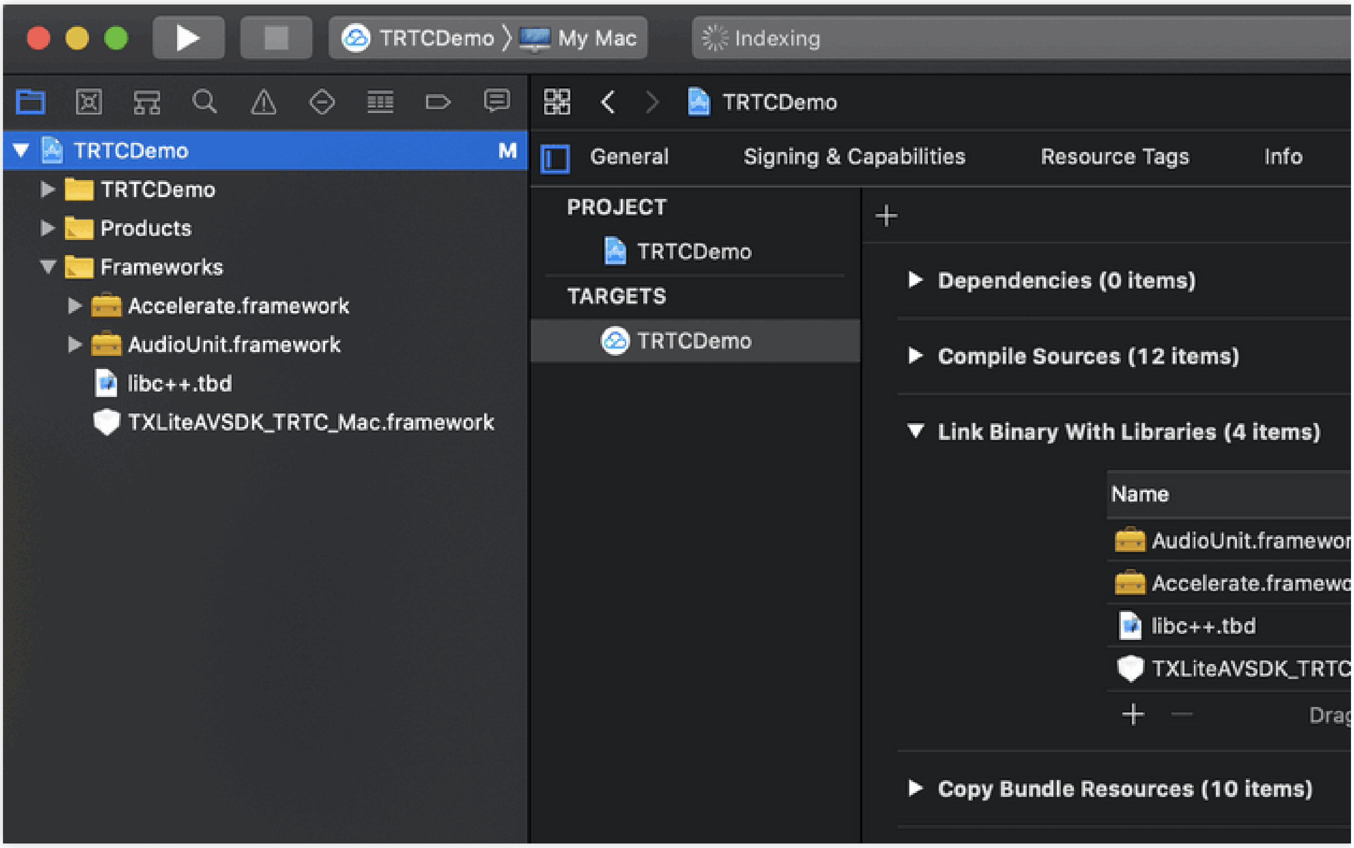


4. Expand **Link Binary With Libraries** and click the **+** icon at the bottom to add dependent libraries.



5. Add the downloaded SDK framework and its required dependencies in sequence: `TXFFmpeg.xcframework` , `TXSoundTouch.xcframework` , `libc++.tbd` , `Accelerate.framework` ,

`SystemConfiguration.framework` , `MetalKit.framework` .If it is successful, you will see the following:



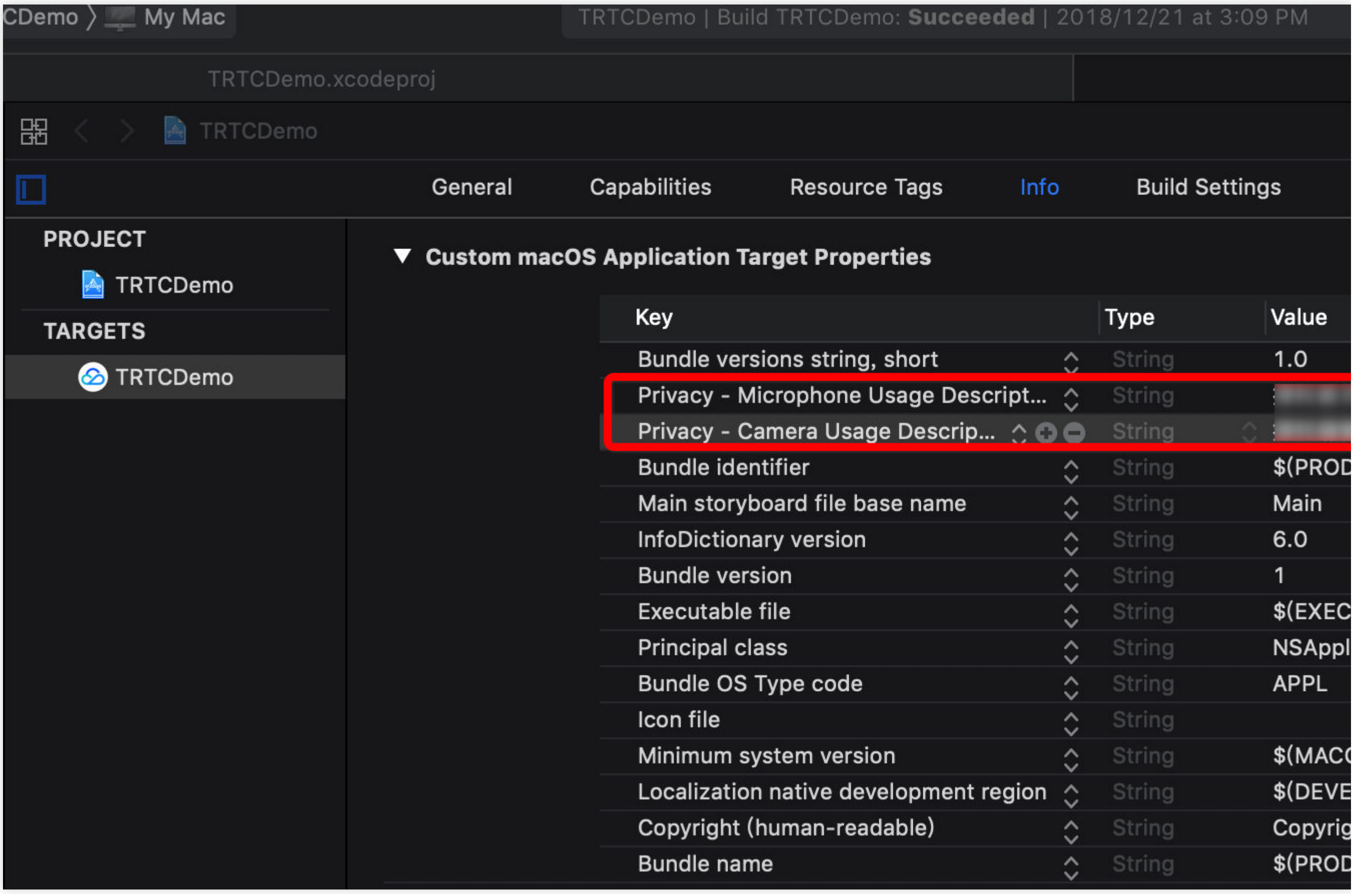
Step 2. Configure app permissions

To use the audio/video features of the SDK, you need to grant it mic and camera permissions. Add the two items below to `Info.plist` of your application. Their content is what users see in the mic and camera access pop-up windows.

Privacy - Microphone Usage Description. Include a statement specifying why mic access is needed

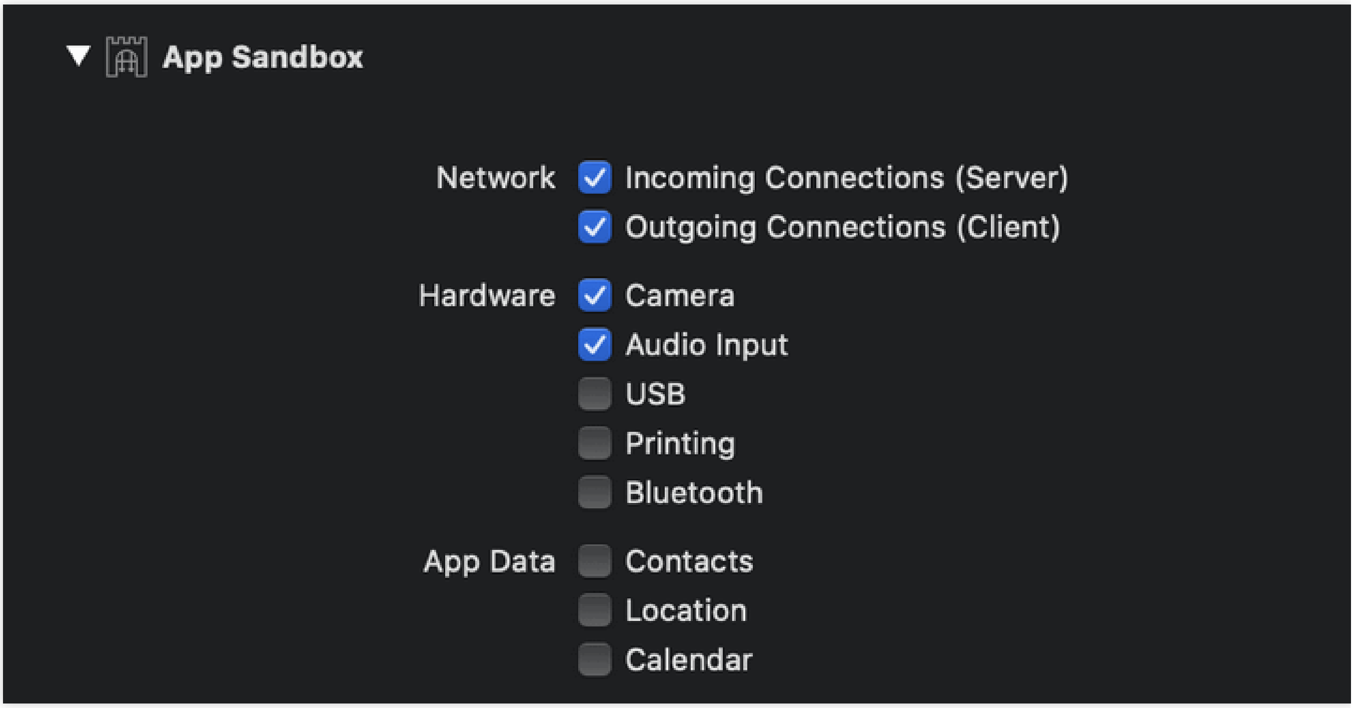
Privacy - Camera Usage Description. Include a statement specifying why camera access is needed

As shown below:

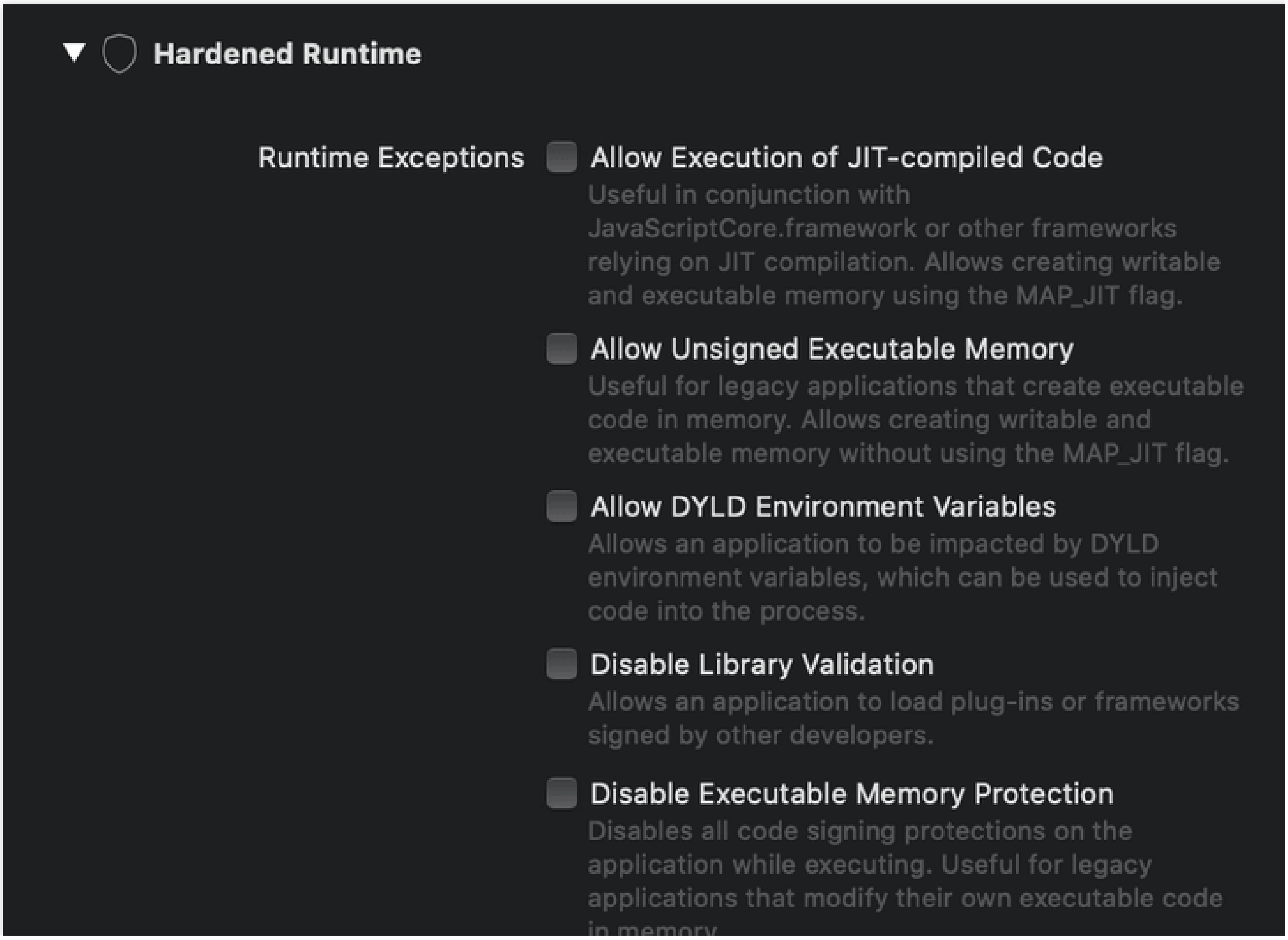


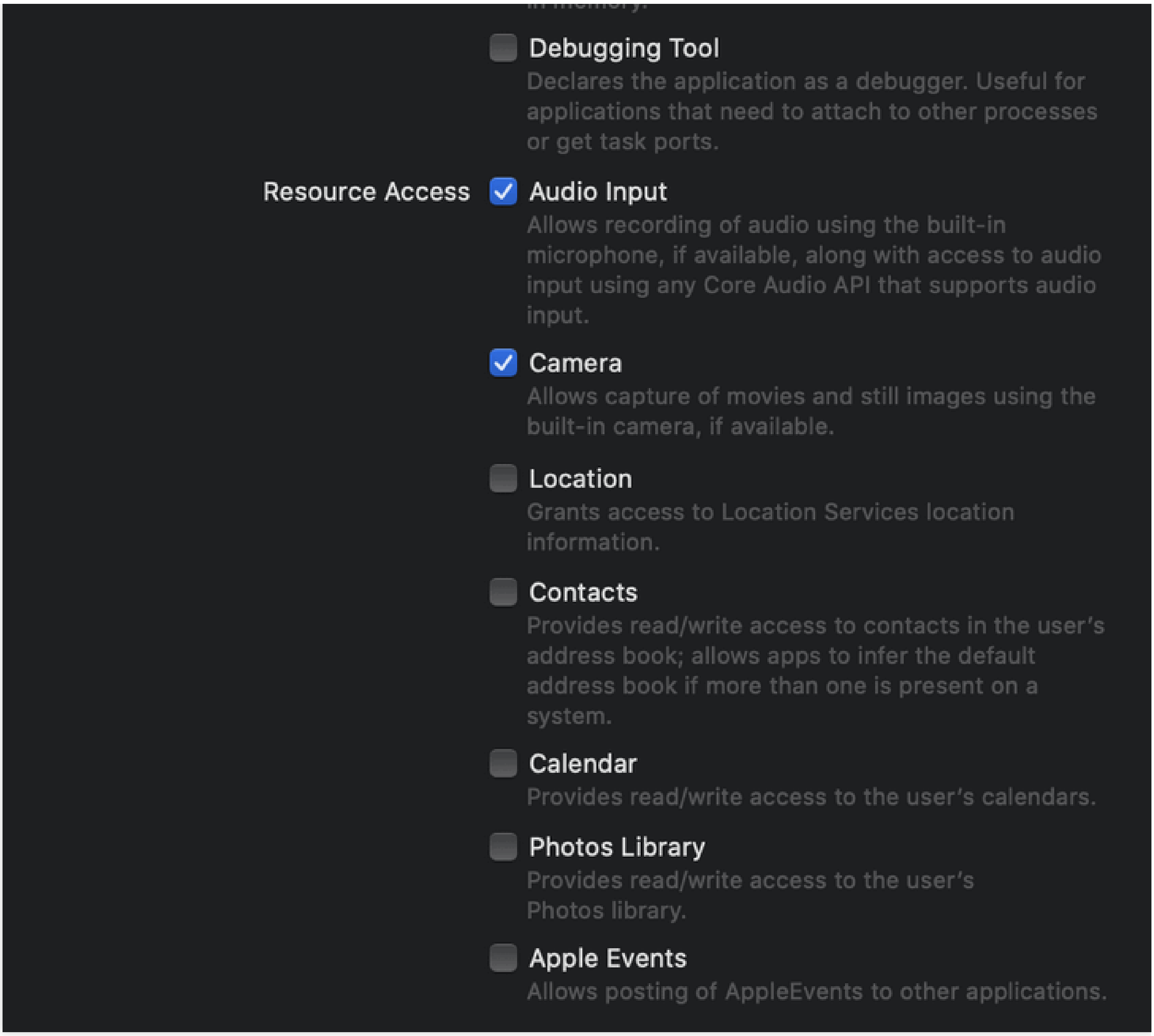
If **App Sandbox** or **Hardened Runtime** is enabled for your application, select `Network` , `Camera` , and `Audio Input` .

For App Sandbox:



For Hardened Runtime:





Step 3. Using the SDK in your project

After completing the first step of importing and the second step of granting device permissions, you can use the APIs provided by the SDK in your project.

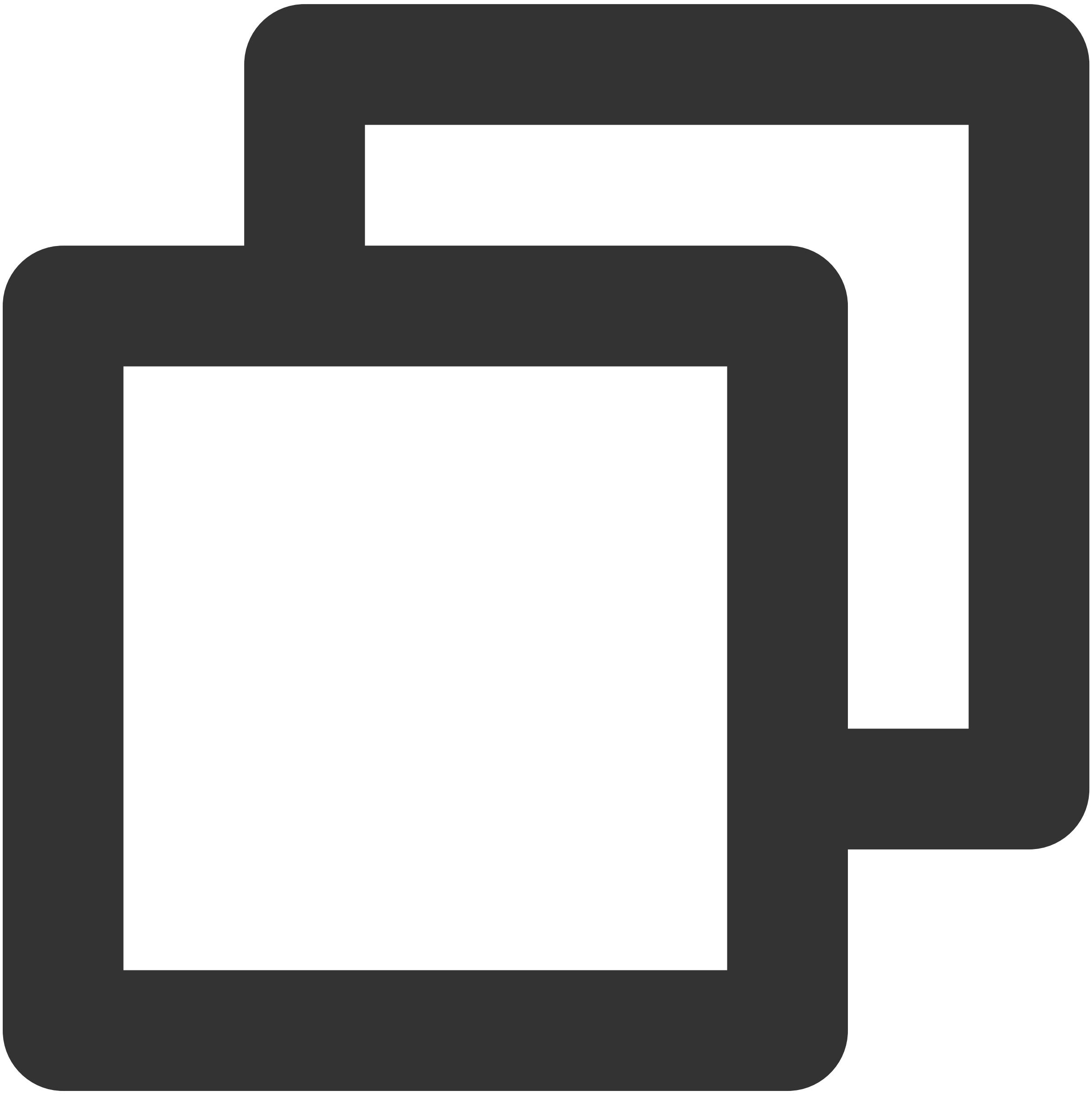
Using Objective-C or Swift APIs

There are two ways to use the SDK in Objective-C or Swift:

Import the module: Import the SDK module in the files that will use the SDK APIs.

```
@import TXLiteAVSDK_TRTC_Mac;
```

Import the header file: Import the header file in the files that will use the SDK APIs.



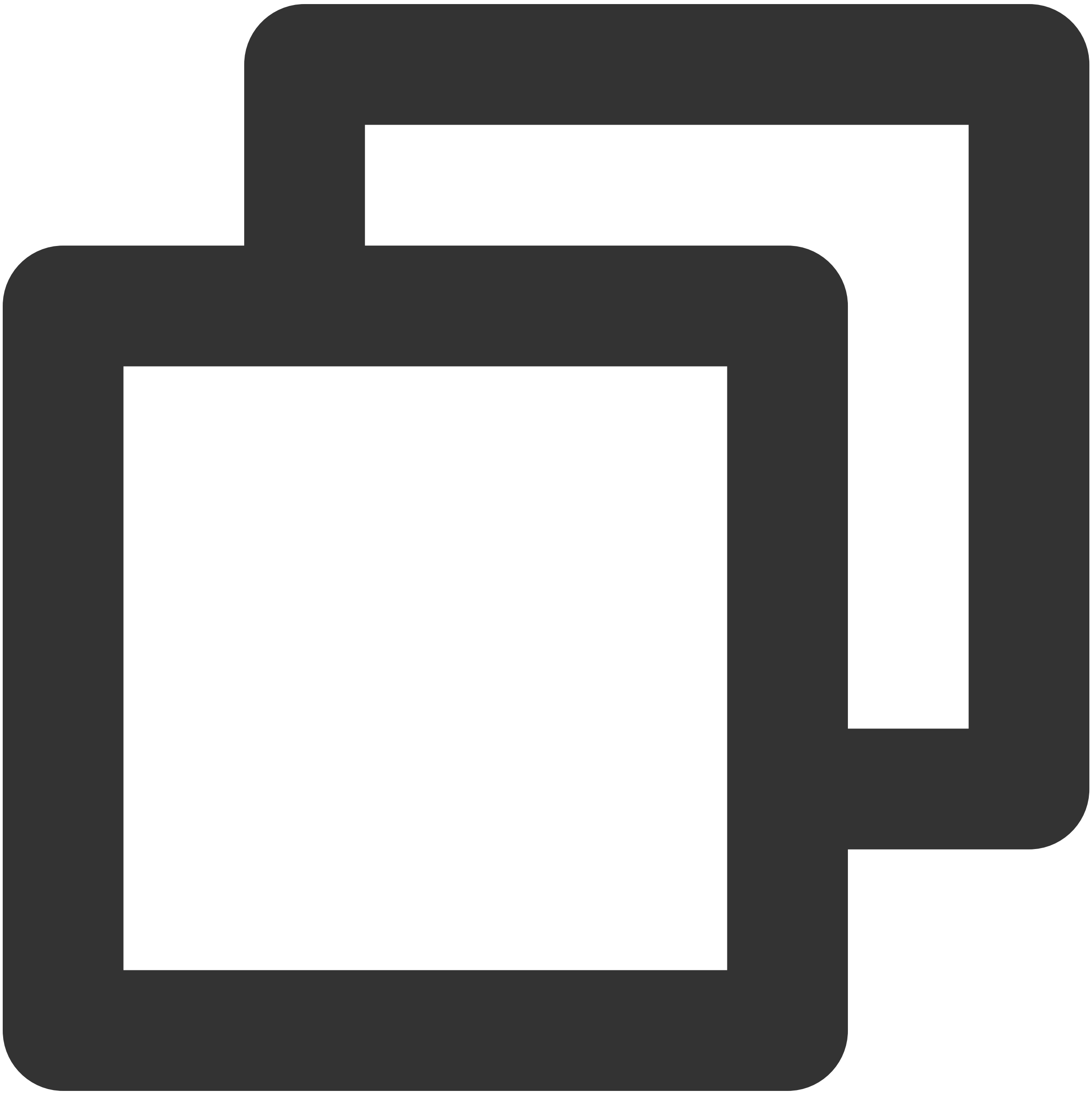
```
#import TXLiteAVSDK_TRTC_Mac/TRTCCloud.h
```

Using C++ APIs (optional)

1. **Import the header file**: If you want to use C++ APIs to develop your macOS application, import the header file in the `TXLiteAVSDK_TRTC_Mac.framework/Headers/cpp_interface` directory.


```
#include TXLiteAVSDK_TRTC_Mac/cpp_interface/ITRTCCloud.h
```

2. **Use the namespace**: The cross-platform C++ APIs and types are all defined in the TRTC namespace, which you can use directly. This method can simplify your code and is recommended.



```
using namespace trtc;
```

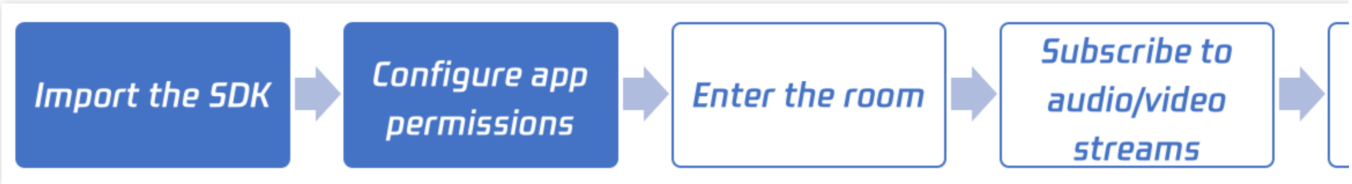
Note:

For more information on how to use C++ APIs, see Overview.

Windows C++

Last updated：2024-05-21 15:05:29

This document describes how to quickly integrate the TRTC Windows C++ SDK using an MFC project.



Environment Requirements

OS: Windows 7 or later
Development environment: Visual Studio 2010 or later (v2015 is recommended)

Integrating C++ SDK via MFC project

The following describes how to integrate the TRTC Windows C++ SDK into an MFC project in Visual Studio.

Step 1. Download the SDK

Download the SDK, decompress, and open it. You only need to import the SDK files for Windows C++ in the `SDK` folder. For example, you can find the SDK files for 64-bit Windows in `./SDK/CPlusPlus/Win64/` . The folder contains the following files:

Directory	Description
include	API header files with comments
lib	The LIB file for compilation and DLL files to load

Step 2. Create a project

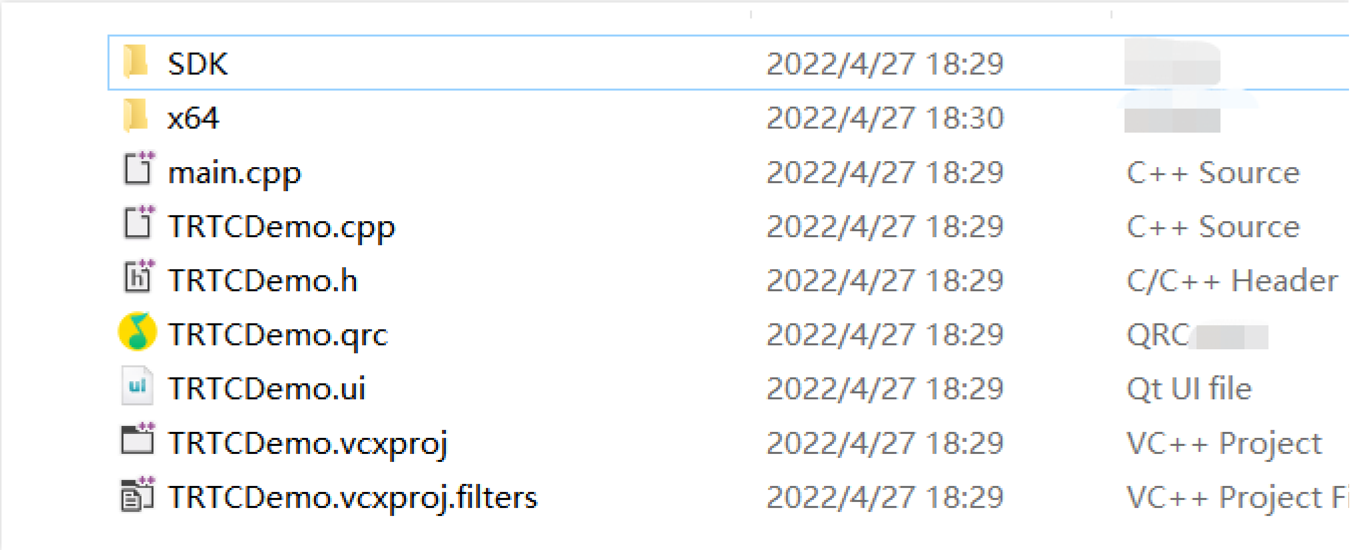
Open Visual Studio and create an MFC application named `TRTCDemo` .
To better describe how to integrate quickly, we choose the relatively simple **Dialog-based** type on the **Application Type** page of the wizard.
For other configuration items, keep the default configurations.

Step 3. Copy and paste files

Copy the `SDK` folder to the directory where `TRTCDemo.vcxproj` is located.

Note:

Because you will only need the C++ SDK, you can delete the `CSharp` folder in `SDK` .



Step 4. Modify the project configuration

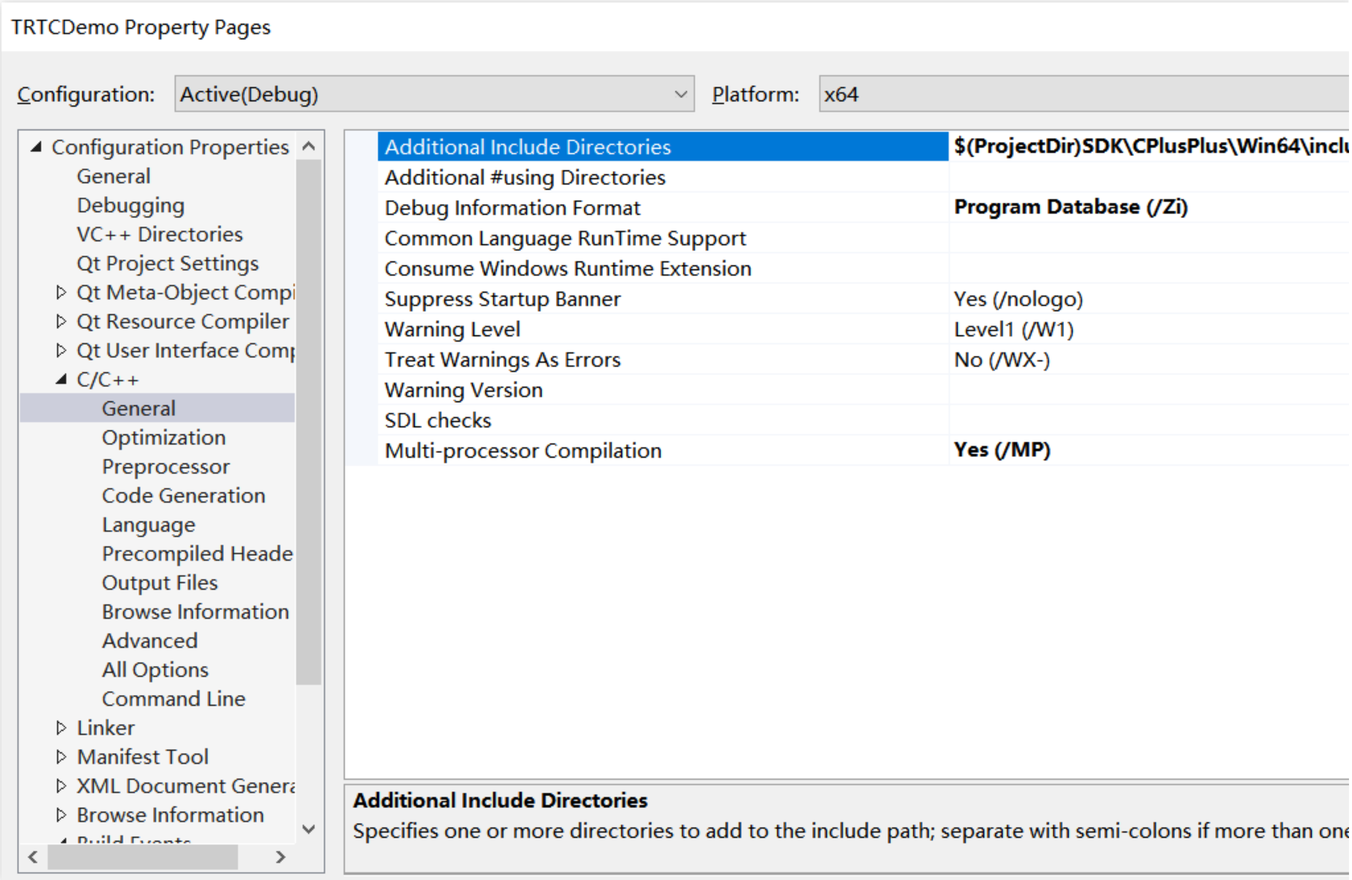
Select **Solution Explorer**, right-click `TRTCDemo` , and select **Properties**. Configure the project as follows:

1. **Add include directories.**

Go to **C/C++** > **General**. Add the `$(ProjectDir)SDK\\CPlusPlus\\Win64\\include` and `$(ProjectDir)SDK\\CPlusPlus\\Win64\\include\\TRTC` header file directories (for 64-bit Windows) to **Additional Include Directories**.

Note:

For 32-bit Windows, add `$(ProjectDir)SDK\\CPlusPlus\\Win32\\include` and `$(ProjectDir)SDK\\CPlusPlus\\Win32\\include\\TRTC` .

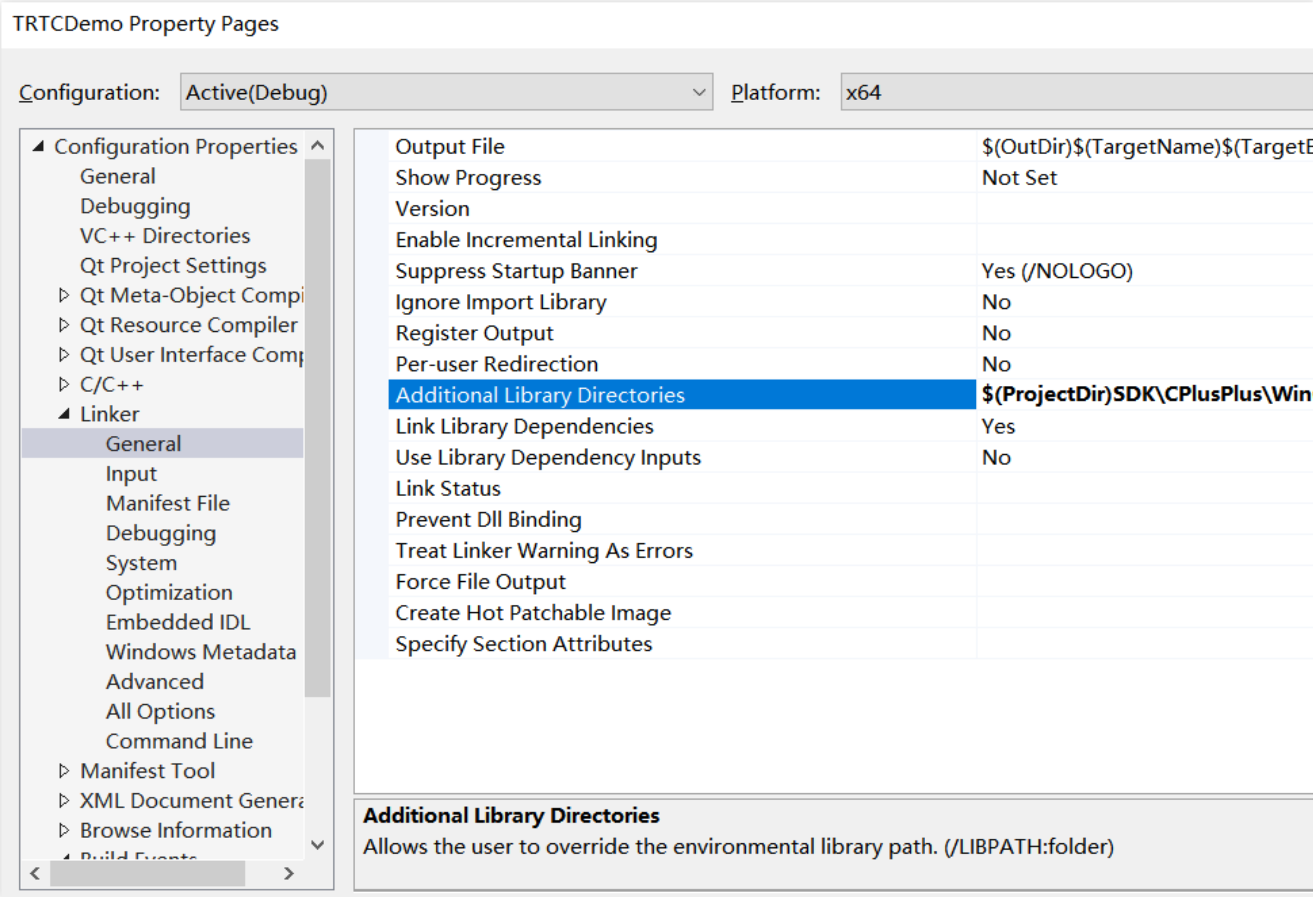


2. **Add additional library directories**

Go to **Linker** > **General**. Add the `$(ProjectDir)SDK\\CPlusPlus\\Win64\\lib` directory to **Additional Library Directories**.

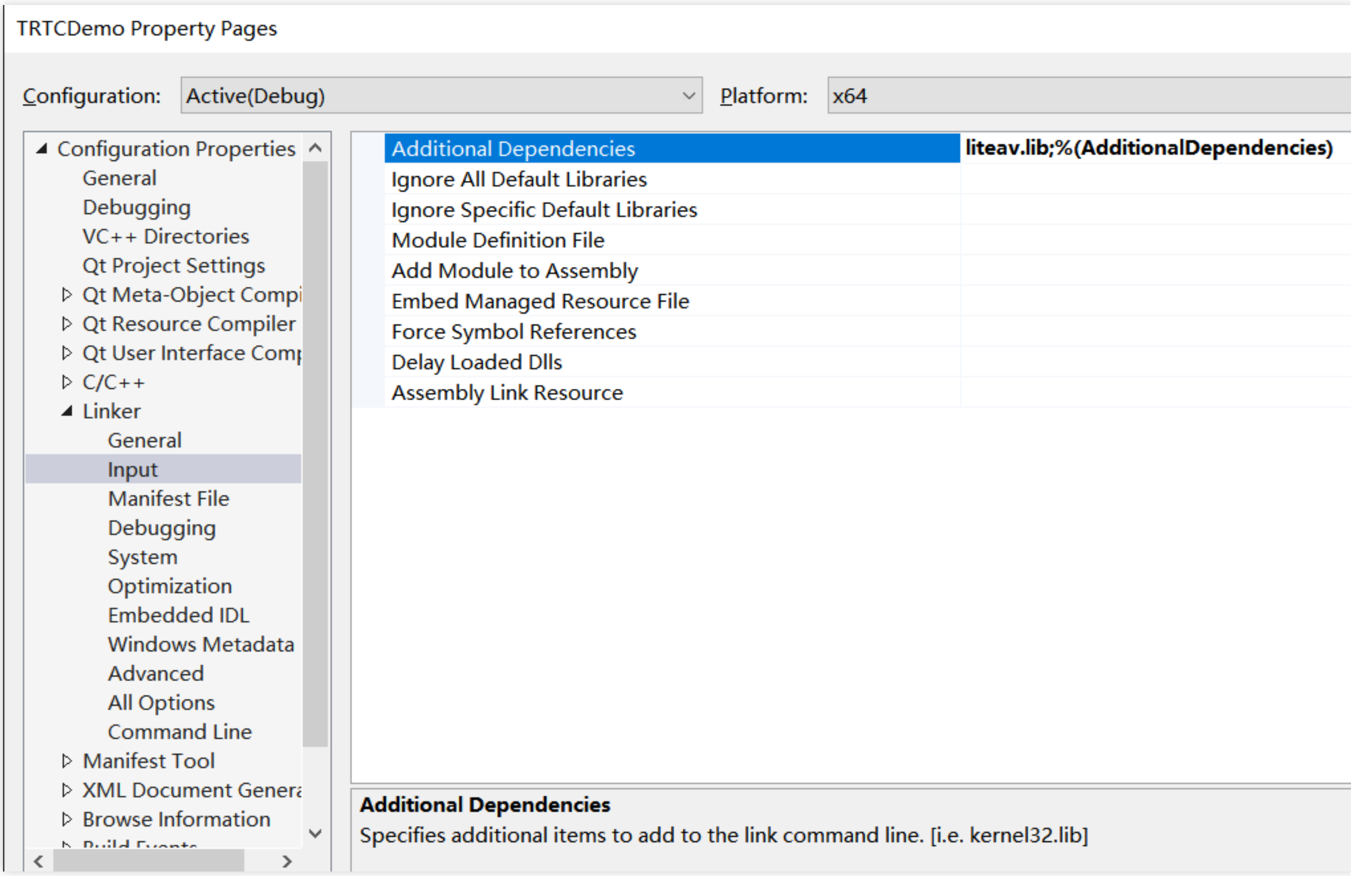
Note:

For 32-bit Windows, add `$(ProjectDir)SDK\\CPlusPlus\\Win32\\lib` .



3. **Add the library file**

Go to **Linker** > **Input**, and add the library file `liteav.lib` to **Additional Dependencies**.

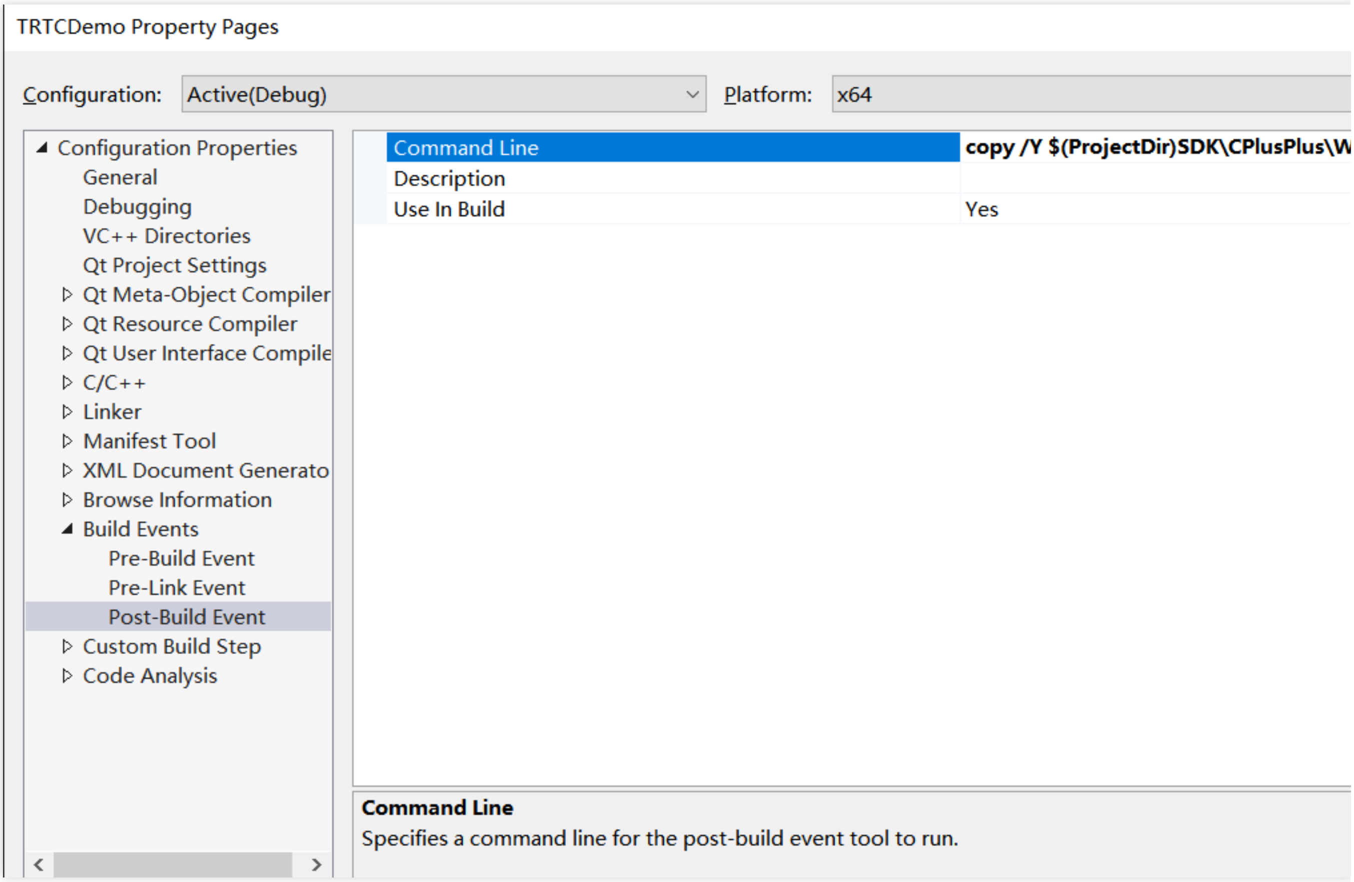


4. **Add the copy command**

Go to **Build Events** > **Post-build Events** and add the copy command `copy /Y $(ProjectDir)SDK\\CPlusPlus\\Win64\\lib\\*.dll $(OutDir)` (for 64-bit Windows) to **Command Line**. This ensures that the DLL files of the SDK are automatically copied to the project's execution directory after compilation.

Note:

For 32-bit Windows, add `copy /Y $(ProjectDir)SDK\\CPlusPlus\\Win32\\lib\\*.dll $(OutDir)`.



Step 5. Print the SDK version number

1. At the top of the `TRTCDemoDlg.cpp` file, add the code below to import the header file:

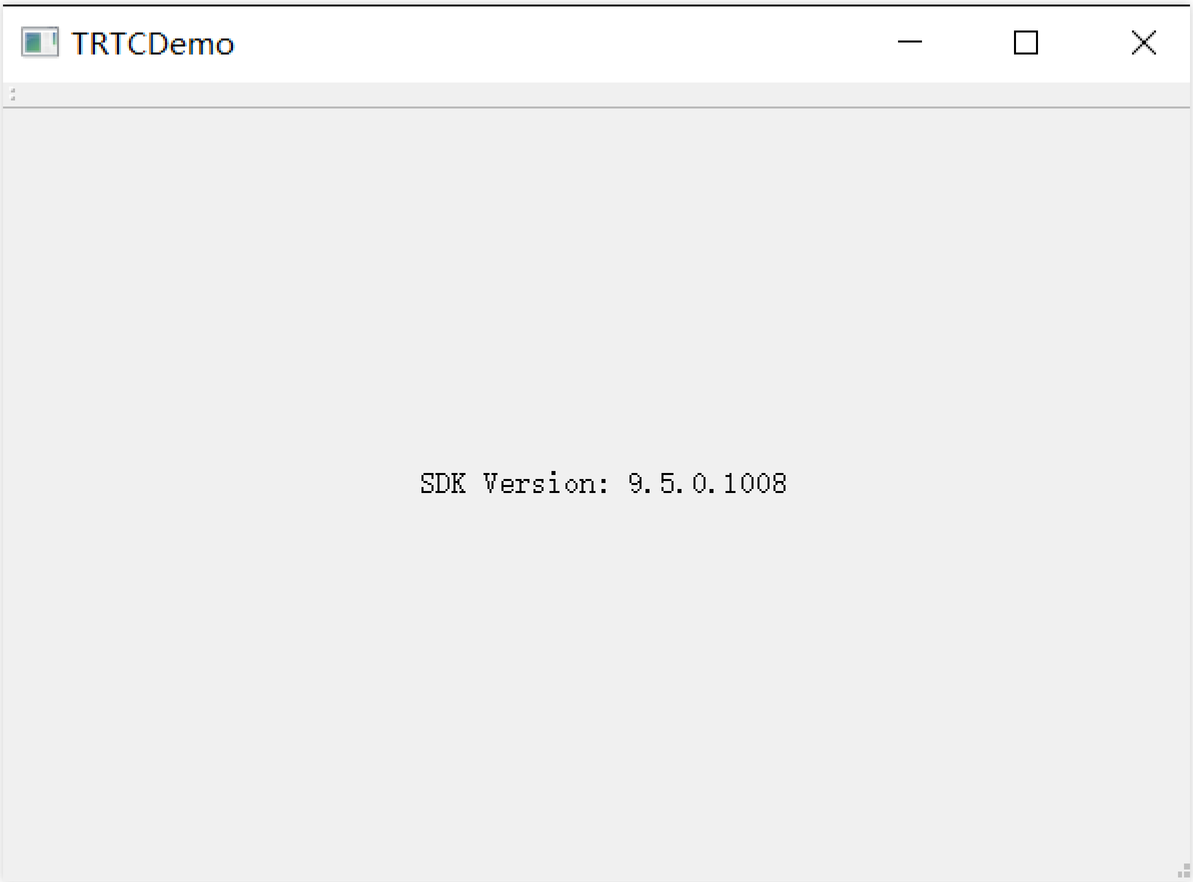
```
#include "ITRTCCloud.h"
```

2. In the `CTRTCDemoDlg::OnInitDialog` function, add the following test code:

```
ITRTCCloud * pTRTCCloud = getTRTCShareInstance();
CString szText;
szText.Format(L"SDK version: %hs", pTRTCCloud->getSDKVersion());

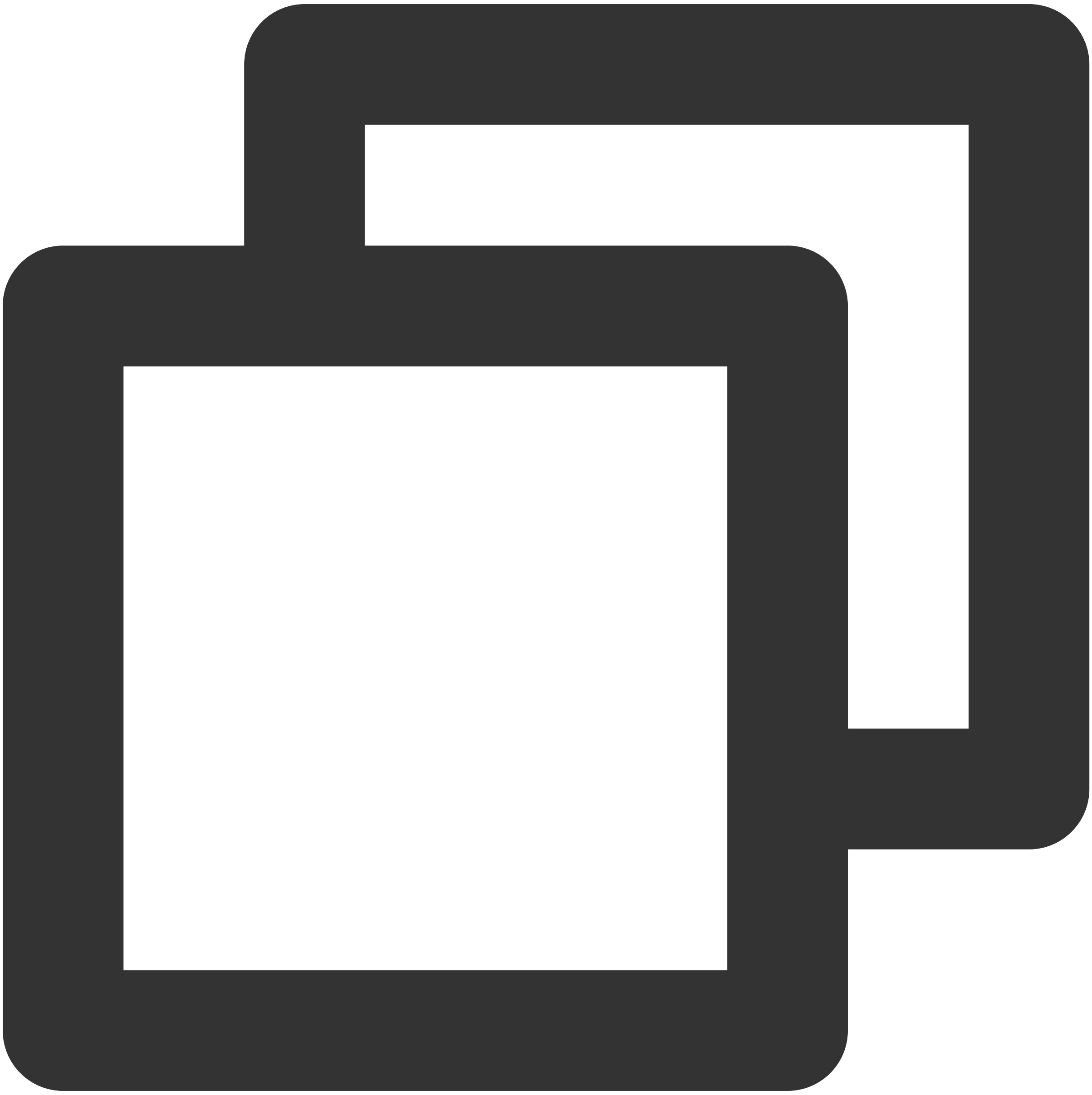
CWnd *pStatic = GetDlgItem(IDC_STATIC);
pStatic->SetWindowTextW(szText);
```

3. Press F5 to run the project and print the version number of the SDK.



FAQs

If the following error occurs, check whether the SDK header file directories are correctly added as described in the project configuration step above.



```
fatal error C1083: Could not open include file: "TRTCCloud.h": No such file or dire
```

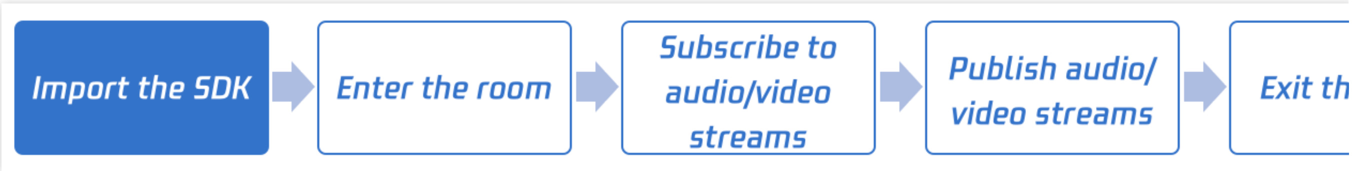
If the following error occurs, check whether the SDK library directory and library file are correctly added as described in the project configuration step above.

```
error LNK2019: unresolved external symbol "__declspec(dllimport) public: static cla
```

Web

Last updated：2023-09-08 10:23:54

This document describes how to import TRTC Web SDK into your project.



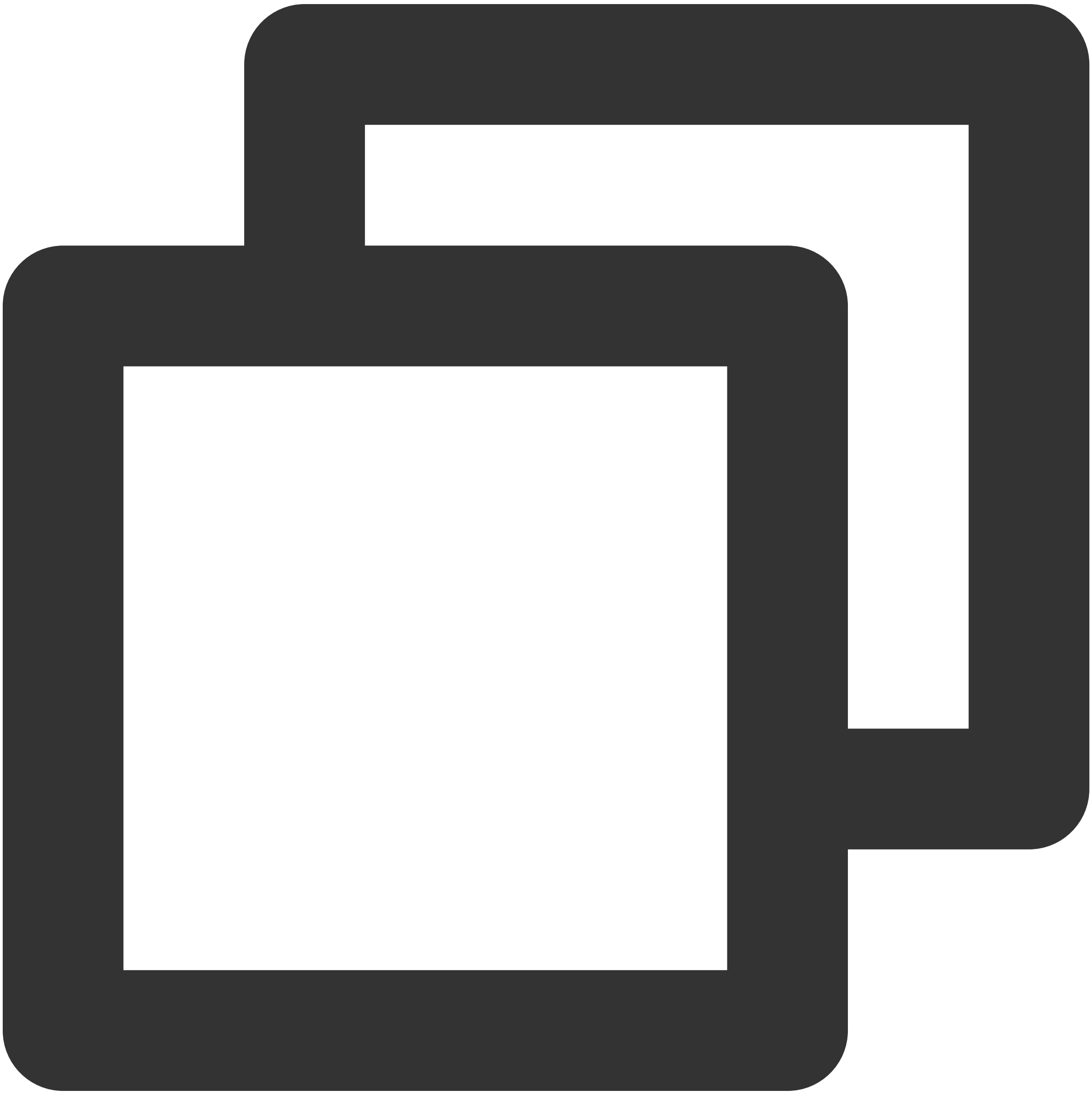
Supported Platforms

The TRTC Web SDK supports all major browsers such as **Chrome, Edge, Firefox, Safari, and Opera**. For a list of browsers supported by TRTC, see Supported Platforms.

Import SDK to your project

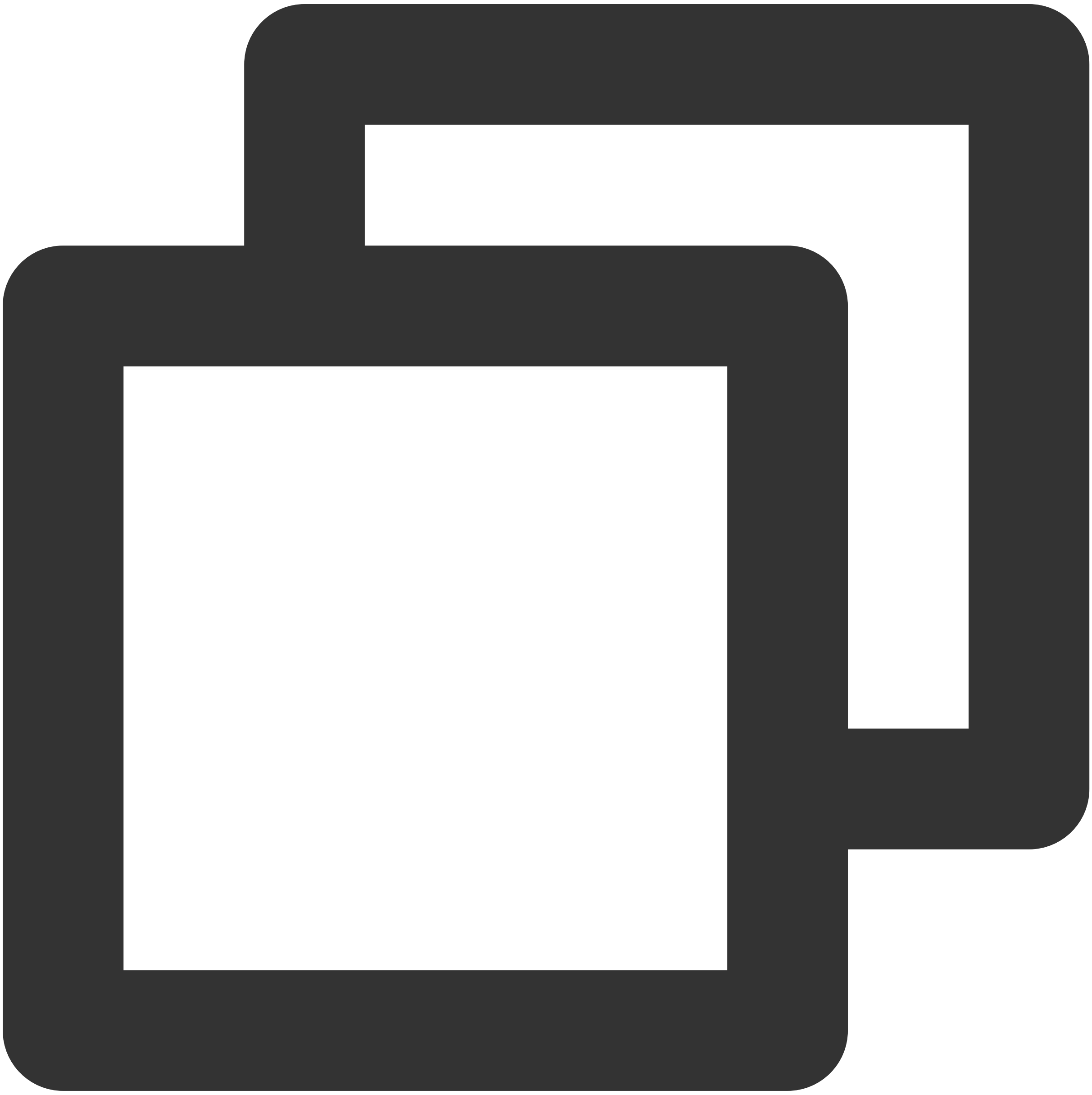
By npm

1. Use npm to install trtc-sdk-v5 package in your project.



```
npm install trtc-sdk-v5 --save
```

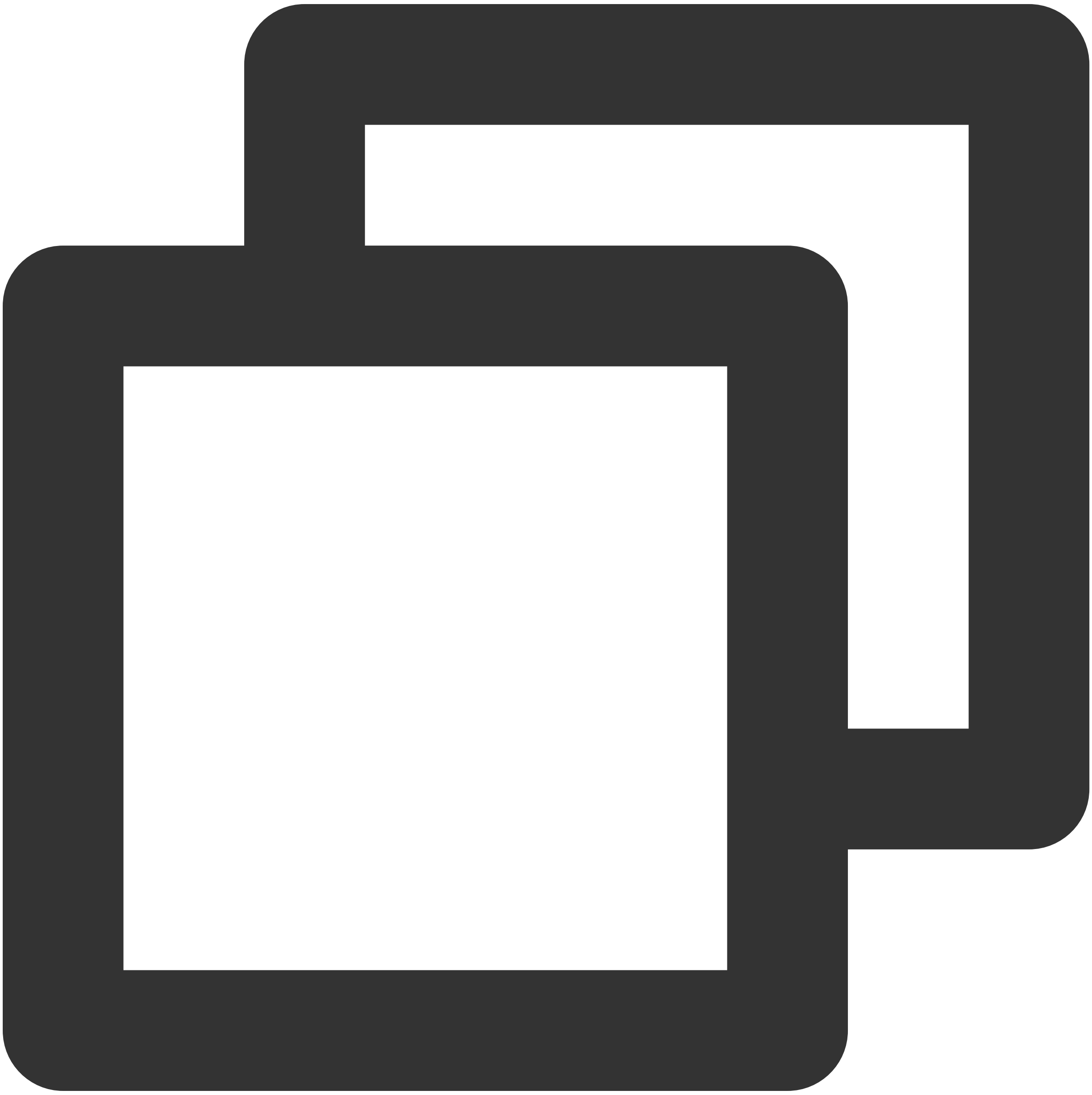
2. Import the module in the project script.



```
import TRTC from 'trtc-sdk-v5';
```

By script

1. Download SDK file trtc.js from Github.
2. Add the following code to your webpage:



```
<script src="trtc.js"></script>
```

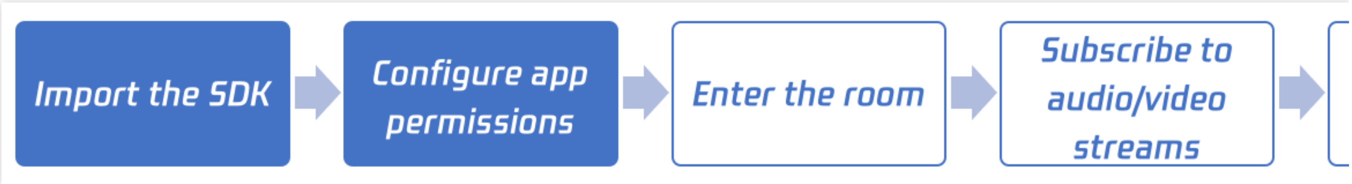
Enter Room

After import SDK to your project, you can use TRTC to enter a room, please refer to: Enter Room.

Electron

Last updated：2024-05-21 15:05:29

This document describes how to quickly integrate the TRTC Electron SDK into your project.



Supported Platforms

Windows

macOS

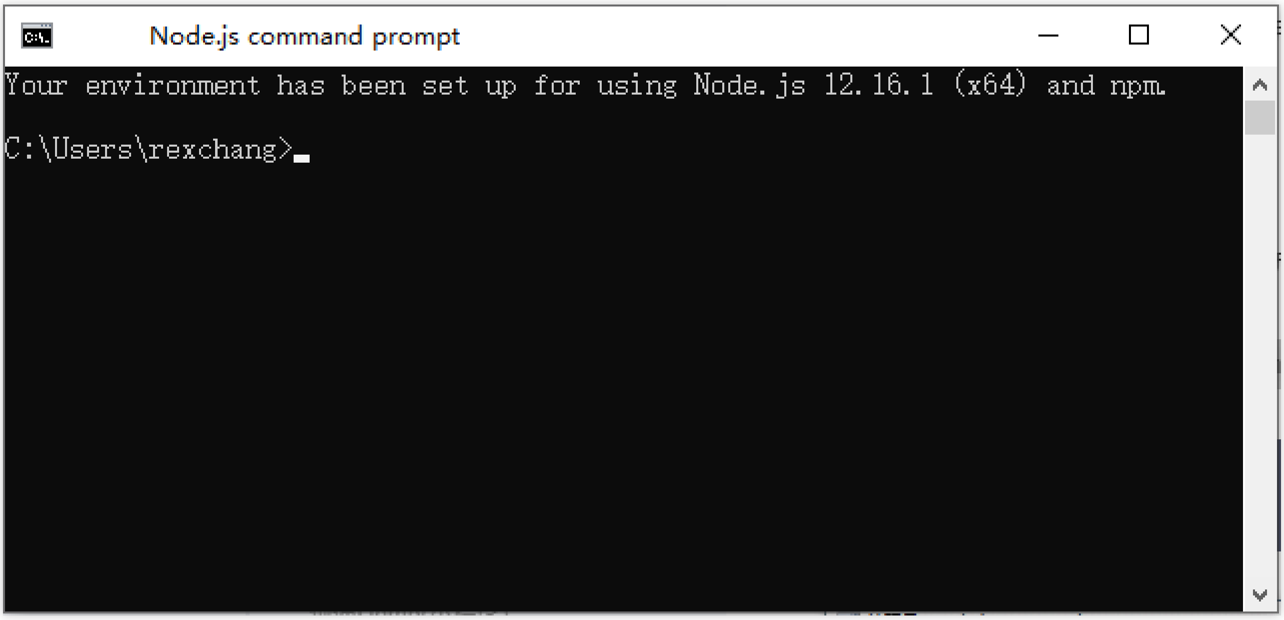
Importing the SDK

Step 1. Install Node.js

Windows

macOS

1. Download the latest version of Node.js installer `Windows Installer (.msi) 64-bit` .
2. Open `Node.js command prompt` in the application list and open a terminal window.



1. Open the terminal window and run the following command to install Homebrew. If you have already installed it, skip this step.

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/i
```

2. Run the following command to install Node.js (v10.0 or later).

```
$ brew install node
```

Step 2. Install Electron

Run the following command in the terminal to install Electron. V4.0.0 or later is recommended.

```
$ npm install electron@latest --save-dev
```

Step 3. Install the TRTC Electron SDK

1. Use the following nmp command in your Electron project to install the SDK.


```
$ npm install trtc-electron-sdk@latest --save
```

Note:

You can view the information of the latest version of the TRTC Electron SDK here.

2. In the project script, import and use the module:

```
const TRTCCloud = require('trtc-electron-sdk').default;
// import TRTCCloud from 'trtc-electron-sdk';
this.rtcCloud = new TRTCCloud();
// Get the SDK version number
this.rtcCloud.getSDKVersion();
```

Since v7.9.348, the TRTC Electron SDK has integrated `trtc.d.ts` for developers using TypeScript.

```
import TRTCCloud from 'trtc-electron-sdk';
const rtcCloud: TRTCCloud = new TRTCCloud();
// Get the SDK version number
rtcCloud.getSDKVersion();
```

Step 4. Create an executable program

We recommend you use the build tool `electron-builder. You can run the following command to install it.

```
$ npm install electron-builder@latest --save-dev
```

In order to package the TRTC Electron SDK (`trtc_electron_sdk.node`) correctly, you must also run the following command to install `native-ext-loader` .


```
$ npm install native-ext-loader@latest --save-dev
```

Step 5. Modify build configuration (`webpack.config.js`)

The `webpack.config.js` file contains the configuration information for project building. You can locate it in the following ways.

Normally, `webpack.config.js` is in the root directory of the project.

If you create your project with `create-react-app` , the configuration file will be `node_modules/react-scripts/config/webpack.config.js` .

If you create your project with `vue-cli` , webpack configuration will be stored in the `configureWebpack` property of `vue.config.js` .

If your project is customized, please locate webpack configuration according to your project.

1. First, add the following code before `module.exports` to make `webpack.config.js` accept the `--target_platform` command line parameter so that your project can be built correctly for its target platform.

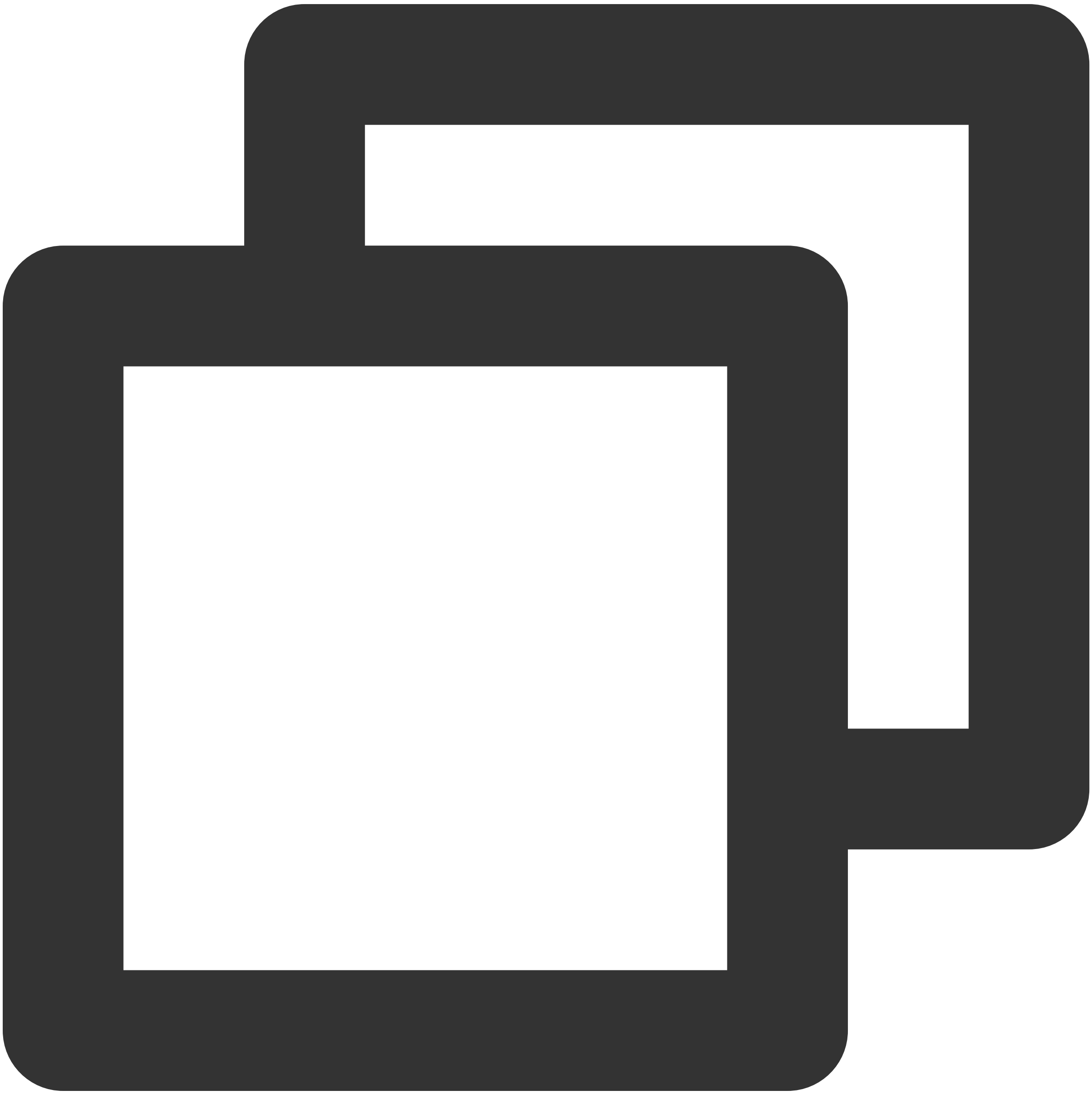
```
const os = require('os');
const targetPlatform = (function(){
    let target = os.platform();
    for (let i=0; i<process.argv.length; i++) {
```

```
            if (process.argv[i].includes('--target_platform=')) {
                target = process.argv[i].replace('--target_platform=', '');
                break;
            }
        }
        if (!['win32', 'darwin'].includes) target = os.platform();
        return target;
  })();
```

Note:

In the result returned by `os.platform()` , `darwin` means macOS, and `win32` means Windows (64-bit or 32-bit).

2. Add the following configuration to the `rules` option. The `targetPlatform` variable ensures that `rewritePath` changes automatically according to the target platform.



```
rules: [
  {
        test: /\\.node$/,
        loader: 'native-ext-loader',
        options: {
            rewritePath: targetPlatform === 'win32' ? './resources' : '../Resource
            // Build for different platforms
            // rewritePath: './node_modules/trtc-electron-sdk/build/Release'
        }
    },
]
```


The above code achieves the following:

If you create an EXE file for Windows, `native-ext-loader` will load the TRTC SDK in `[application root directory]/resources`.

If you create a DMG file for macOS, `native-ext-loader` will load the TRTC SDK in `[application directory]/Contents/Frameworsk/../Resources`.

For local building, `native-ext-loader` will load the TRTC SDK in `./node_modules/trtc-electron-sdk/build/Release`. For details, see TRTCSimpleDemo configuration.

You also need to add the `--target_platform` parameter to the build script of `package.json`, which brings us to the next step.

Step 6. Modify `package.json`

The `package.json` file is in the root directory of the project and contains the necessary build information. Normally, to successfully build your project, you need to modify the path in `package.json` as follows.

1. Modify `main`.

```
// In most cases, the name of the `main` file can be customized. For example, in TR
"main": "main.electron.js",

// However, for projects created with the `create-react-app` scaffolding tool, `mai
"main": "public/electron.js",
```

2. Copy the following `build` configuration to your `package.json` file for `electron-builder` to read.

```
"build": {
  "appId": "[Custom appId]",
  "directories": {
 "output": "./bin"
  },
  "win": {
 "extraFiles": [
   {
     "from": "node_modules/trtc-electron-sdk/build/Release/",
     "to": "./resources",
     "filter": ["**/*"]
```

```
    }
  ]
   },
   "mac": {
  "extraFiles": [
    {
      "from": "node_modules/trtc-electron-sdk/build/Release/trtc_electron_sdk.node",
      "to": "./Resources"
    }
  ]
   }
 },
```

3. Add command scripts for building and packaging under `scripts`.

The following command scripts are for projects created with `create-react-app` and `vue-cli`. They provide samples for projects created with other tools too.

```
}
```

```
// Use this configuration for projects created with `create-react-app`.
"scripts": {
  "build:mac": "react-scripts build --target_platform=darwin",
  "build:win": "react-scripts build --target_platform=win32",
  "compile:mac": "node_modules/.bin/electron-builder --mac",
  "compile:win64": "node_modules/.bin/electron-builder --win --x64",
  "pack:mac": "npm run build:mac && npm run compile:mac",
  "pack:win64": "npm run build:win && npm run compile:win64"
}

// Use this configuration for projects created with `vue-cli`.
```

```
"scripts": {
  "build:mac": "vue-cli-service build --target_platform=darwin",
  "build:win": "vue-cli-service build --target_platform=win32",
  "compile:mac": "node_modules/.bin/electron-builder --mac",
  "compile:win64": "node_modules/.bin/electron-builder --win --x64",
  "pack:mac": "npm run build:mac && npm run compile:mac",
  "pack:win64": "npm run build:win && npm run compile:win64"
}
```

Parameter	Description
main	The entry point file of Electron, which can be customized in most cases. However, if your project is created with create-react-app, the entry point file must be public/electron.js.
build.win.extraFiles	When building for Windows, electron-builder will copy all files in the directory specified by from to bin/win-unpacked/resources (all in lowercase).
build.mac.extraFiles	When packaging for macOS, electron-builder will copy the trtc_electron_sdk.node file specified by from to bin/mac/your-app-name.app/Contents/Resources (capitalize the first letter of each word)
build.directories.output	The output path. In the sample above, the output file is saved to bin. You can modify it as needed.
build.scripts.build:mac	The script for building for macOS.
build.scripts.build:win	The script for building for Windows.
build.scripts.compile:mac	Creates a DMG file for macOS
build.scripts.compile:win64	Creates an EXE file for Windows
build.scripts.pack:mac	Calls build:mac to build the project and then `compile:mac` to generate a DMG file
build.scripts.pack:win64	Calls build:win to build the project and then `compile:win64` to generate an EXE file

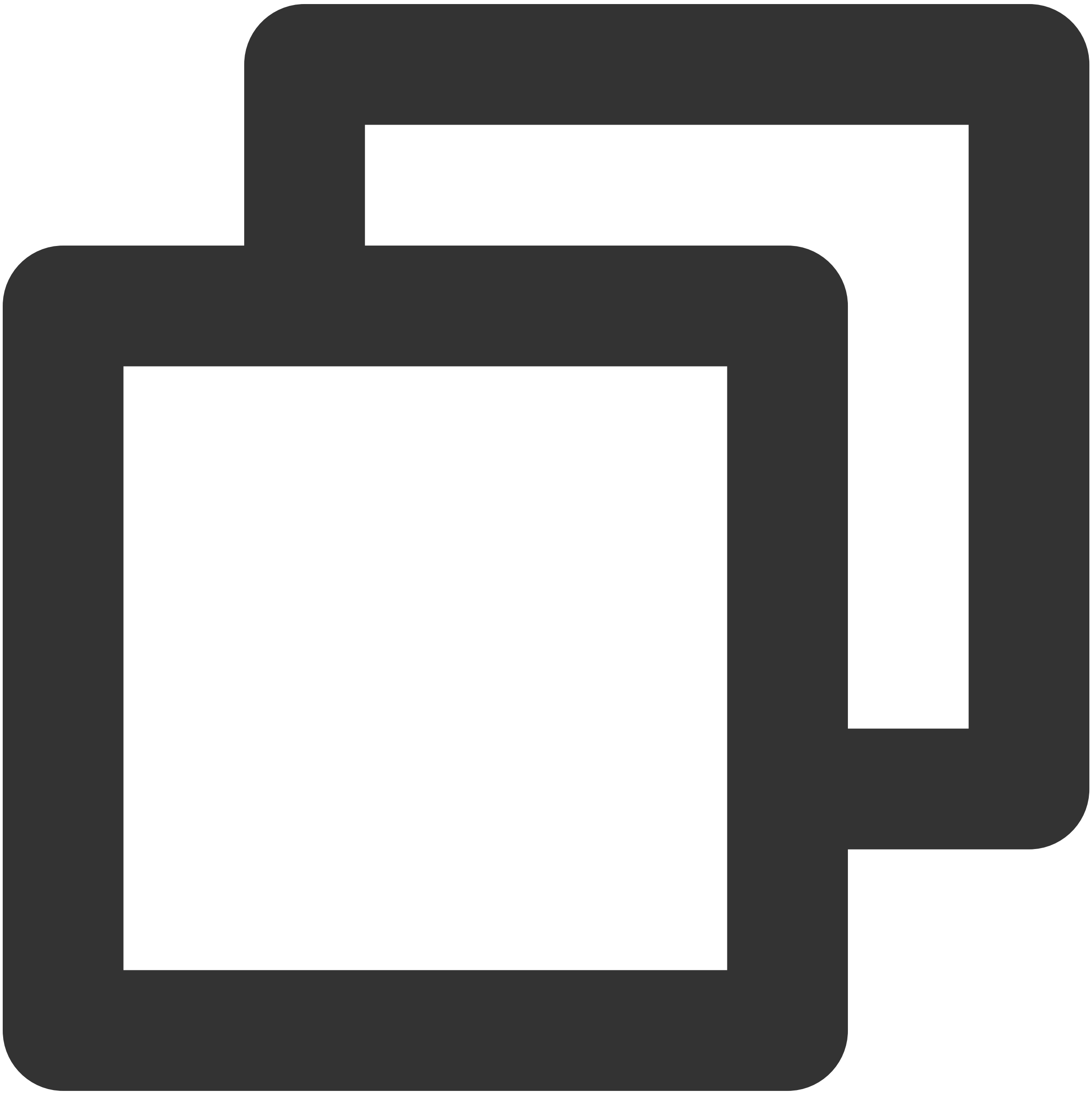
Step 7. Run the build command

Build the project into a DMG file for macOS:

```
$ cd [Project directory]
$ npm run pack:mac
```

The build tool will generate an installation file named `bin/your-app-name-0.1.0.dmg` . Publish this file.

Build the project into an EXE file for Windows:



```
$ cd [Project directory]
$ npm run pack:win64
```

The build tool will generate an installation file named `bin/your-app-name Setup 0.1.0.exe`. Publish this file.

Note:

Currently, the TRTC Electron SDK does not support cross-platform building. This means you cannot build your project into an EXE file on macOS or a DMG file on Windows. We are working on this and may add support for it in the future.

FAQs

What firewall restrictions does TRTC face?

The SDK uses the UDP protocol for audio/video transmission and therefore cannot be used in office networks that block UDP. If you encounter such a problem, see Firewall Restrictions.

What should I do if an error occurs when I install or build the TRTC Electron SDK?

If an error occurs when you integrate the TRTC Electron SDK, for example, if installation times out or the `trtc_electron_sdk.node` file fails to load after building, please contact us.

Learn More

SDK API Guide
Release Notes (Electron)
Simple Demo Source Code
API Example Source Code
FAQs

Flutter

Last updated：2024-05-21 15:05:29

This document describes how to quickly integrate the TRTC Flutter SDK into your project.

Note:

Currently, screen sharing and device selection are not supported on Windows or macOS.

Environment Requirements

Flutter 2.0 or later

Developing for Android:

Android Studio 3.5 or later

Devices with Android 4.1 or later

Developing for iOS and macOS:

Xcode 11.0 or later

OS X 10.11 or later

A valid developer signature for your project

Developing for Windows:

OS: Windows 7 SP1 or later (64-bit based on x86-64)

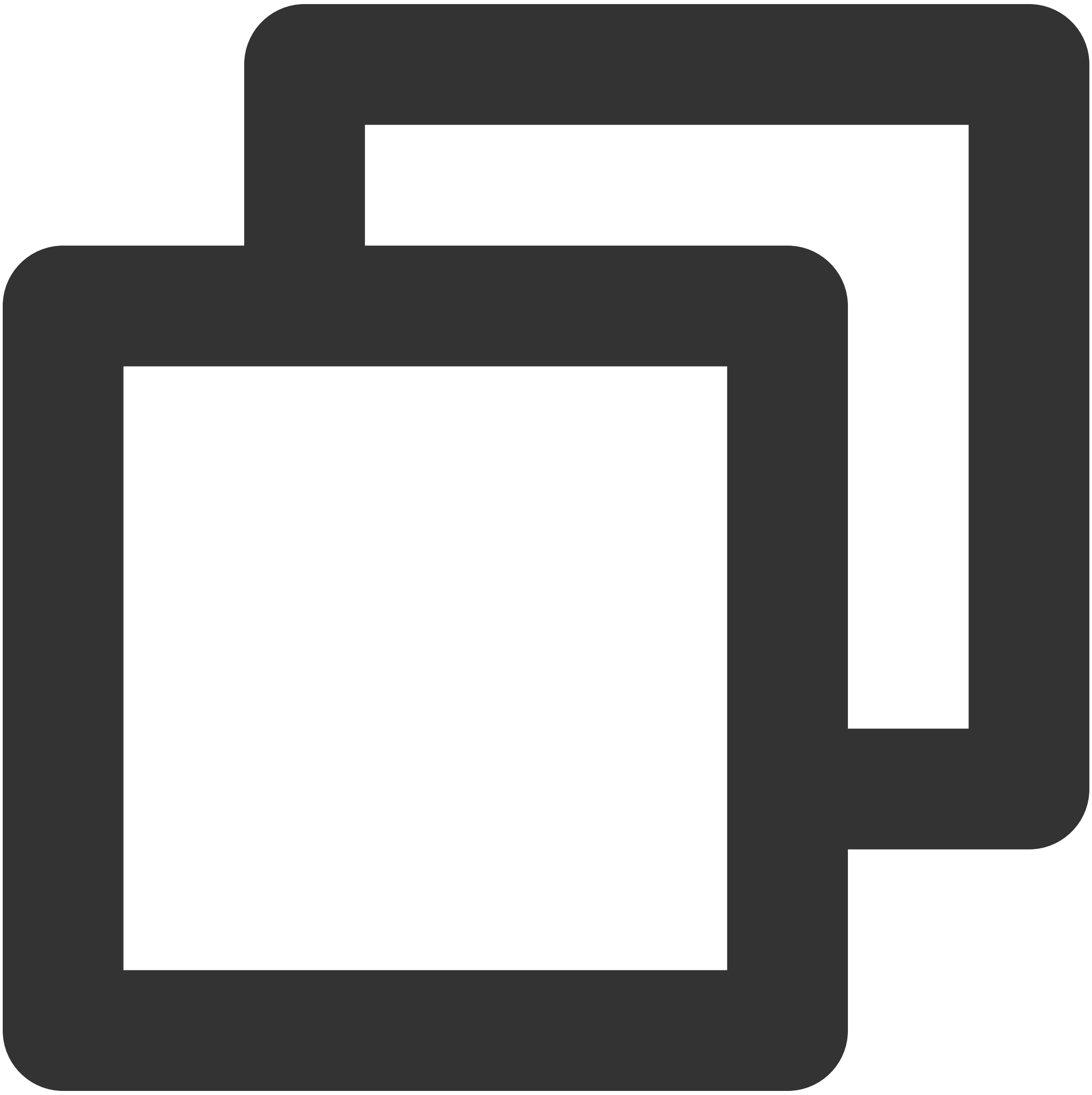
Disk space: At least 1.64 GB of space after the IDE and relevant tools are installed

Visual Studio 2019

Integrating the SDK

The Flutter SDK has been published on Pub. You can have the SDK downloaded and updated automatically by configuring `pubspec.yaml` .

1. Add the following dependency to `pubspec.yaml` of your project.



```
dependencies:
  tencent_trtc_cloud: latest version number
```

2. Obtain **camera** and **mic** permissions to enable the audio and video call features.

iOS

macOS

Android

Windows

1. Add requests for camera and mic permissions in `Info.plist` :


```
<key>NSCameraUsageDescription</key>
<string>Video calls are possible only with camera permission.</string>
<key>NSMicrophoneUsageDescription</key>
<string>Audio calls are possible only with mic access.</string>
```

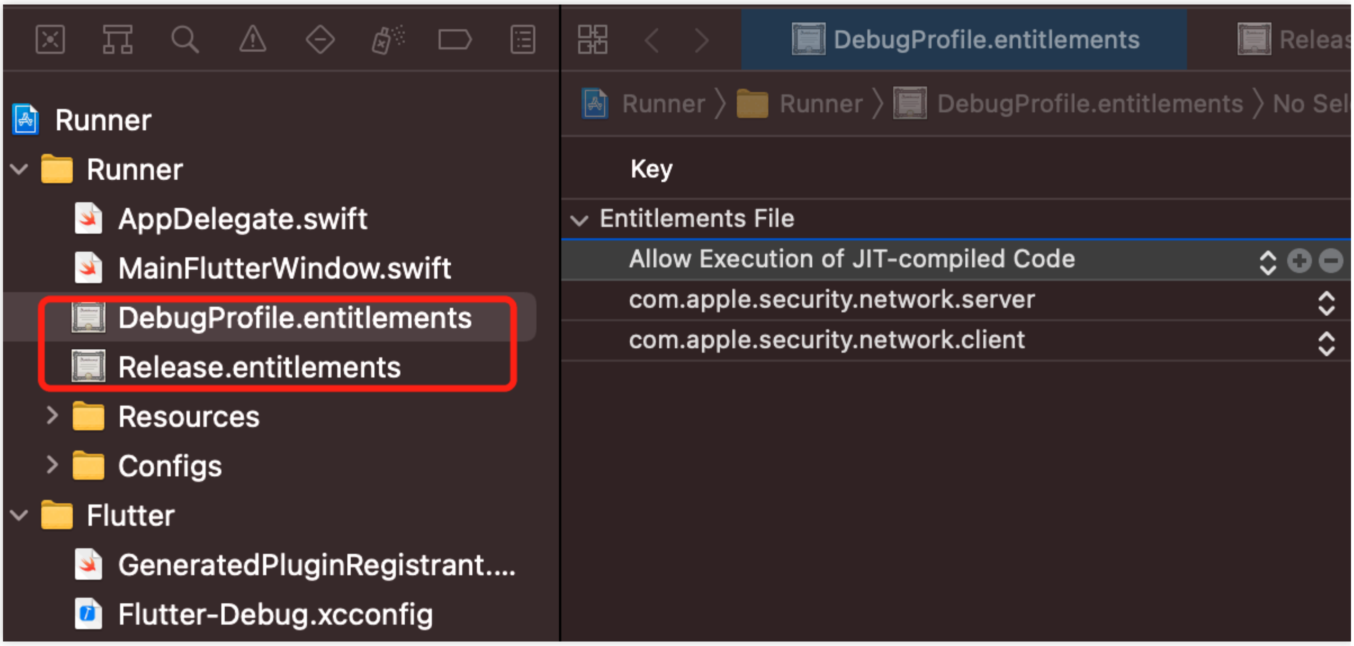
2. Add the field `io.flutter.embedded_views_preview` and set the value to `Yes` .

1. Add requests for camera and mic permissions in `Info.plist` :

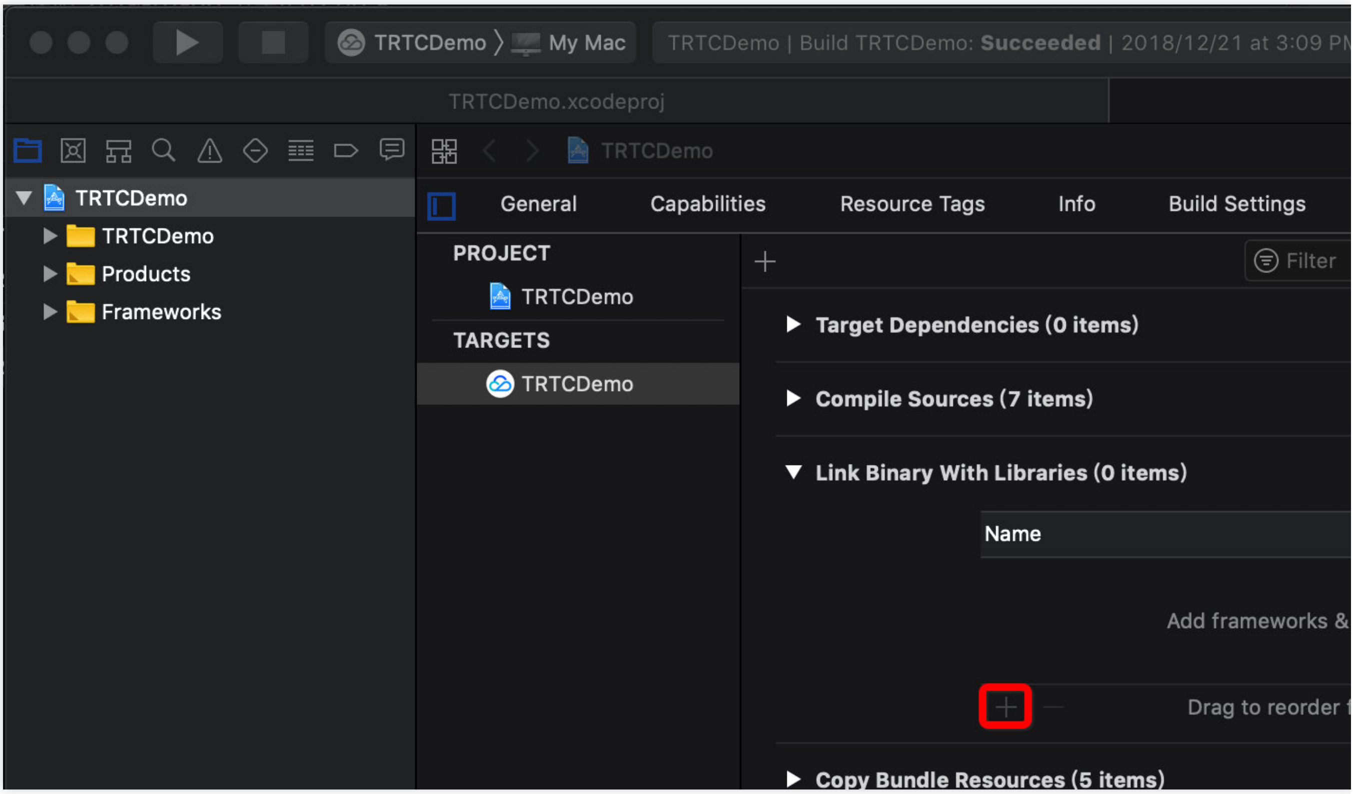
```
<key>NSCameraUsageDescription</key>
<string>Video calls are possible only with camera permission.</string>
<key>NSMicrophoneUsageDescription</key>
<string>Audio calls are possible only with mic access.</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>The app needs your approval to access your gallery.</string>
```

2. Add `com.apple.security.network.client` and `com.apple.security.network.server` to `macos/Runner/*.entitlements`.

If it is successful, you will see the figure below:

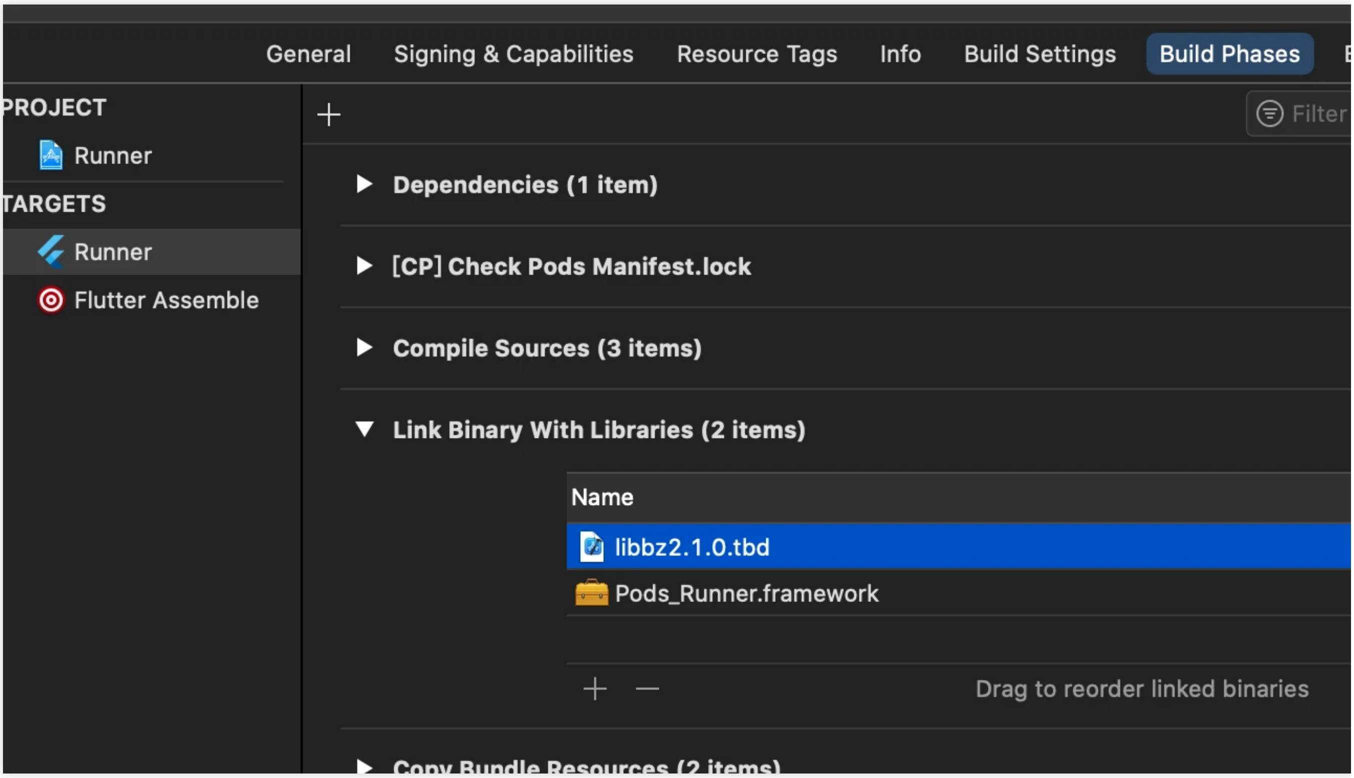


3. Expand **Link Binary With Libraries** and click the **+** icon at the bottom to add dependent libraries.



4. Add the library `libbz2.1.0.tbd` .

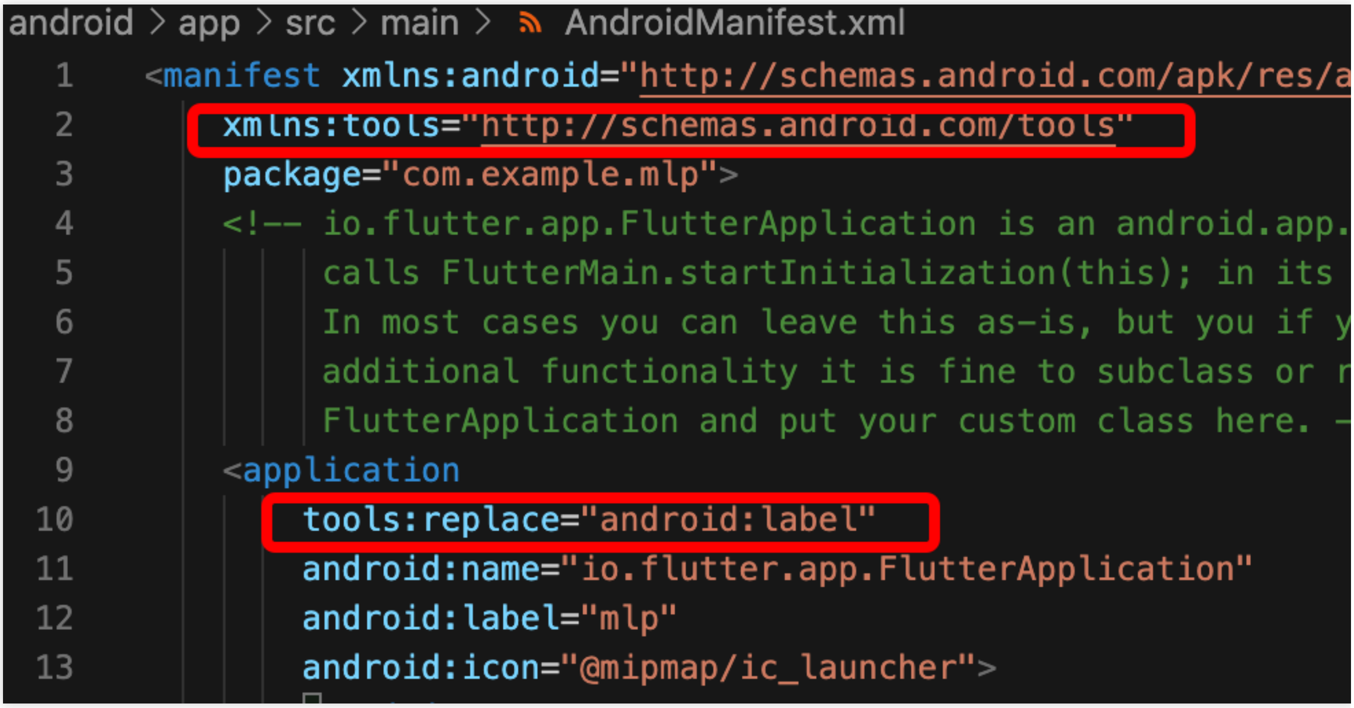
If it is successful, you will see the figure below:



1. Open `/android/app/src/main/AndroidManifest.xml` .
2. Add `xmlns:tools="http://schemas.android.com/tools"` to `manifest` .
3. Add `tools:replace="android:label"` to `application` .

Note:

Without the above steps, the Android Manifest merge failed error will occur and the compilation will fail.



1. Run `flutter config --enable-windows-desktop` .
2. Run `flutter run -d windows` .

FAQs

What should I do if my iOS project crashes when I build and run it?

What should I do if videos show on Android but not on iOS?

What should I do if an error occurs when I run CocoaPods for iOS after updating to the latest version of the SDK?

What should I do if the "Manifest merge failed" error occurs in Android Studio?

What should I do if an error occurs due to the absence of signatures when I debug my project on a real device?

Why can’t I find the corresponding file after deleting/adding content in the swift file of the plugin?

What should I do if the error "Info.plist, error: No value at that key path or invalid key path: NSBonjourServices" occurs when I run my project?

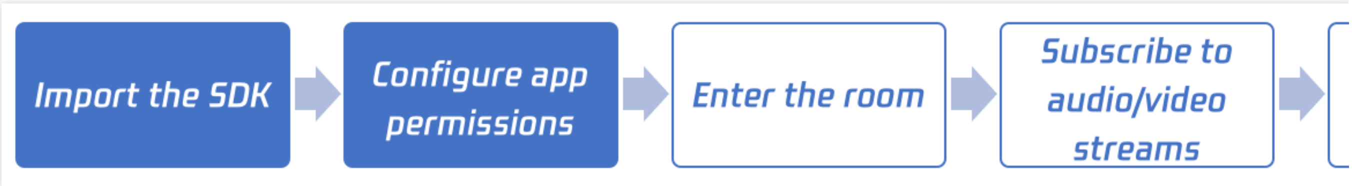
What should I do if an error occurs when I run `pod install` ?

What should I do if a dependency error occurs when I run my iOS project?

Qt

Last updated：2024-05-21 15:05:29

This document describes how to quickly integrate the TRTC Windows or macOS SDK into your project using Qt.



Integration for Windows

Development environment requirements

OS: Windows 7 or later

Development environment: Visual Studio 2015 or later (you need to set up a Qt development environment). Visual Studio 2015 is recommended.

Note:

If you are not sure how to set up a Qt development environment in Visual Studio, see step 4 in README.

Directions

The following describes how to integrate the TRTC Windows C++ SDK into a Qt project in Visual Studio.

Step 1. Download the SDK

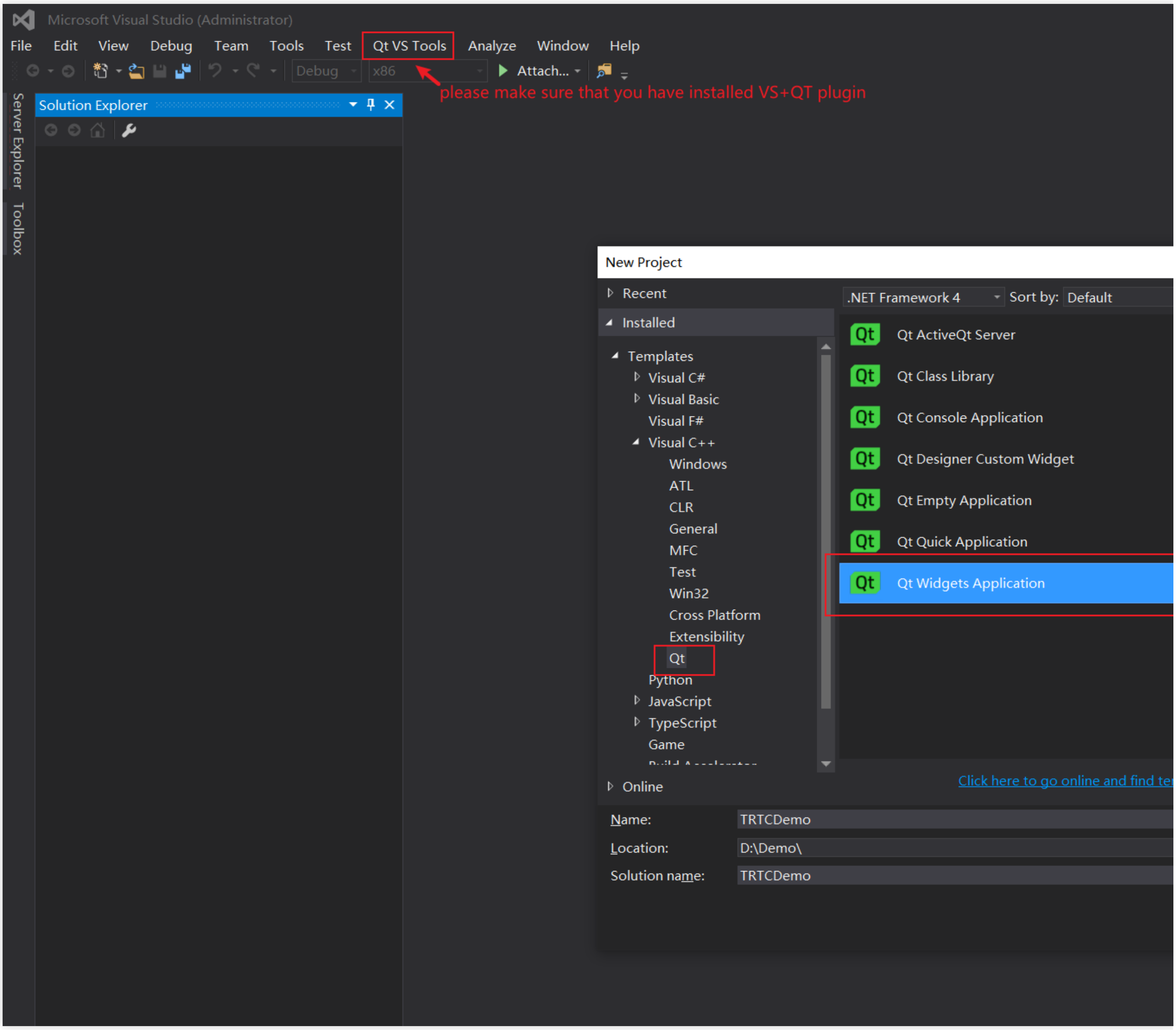
Download the latest version of the TRTC SDK.

You only need to import the SDK files for Windows C++ in the `SDK` folder. For example, you can find the SDK files for 64-bit Windows in `./SDK/CPlusPlus/Win64/` . The folder contains the following files:

Directory	Description
include	API header files with comments
lib	The LIB file for compilation and DLL files to load

Step 2. Create a project

Take Visual Studio 2015 for example. Make sure you have installed Qt and Qt Visual Studio Add-in. Then, open Visual Studio and create a Qt application named `TRTCDemo` .



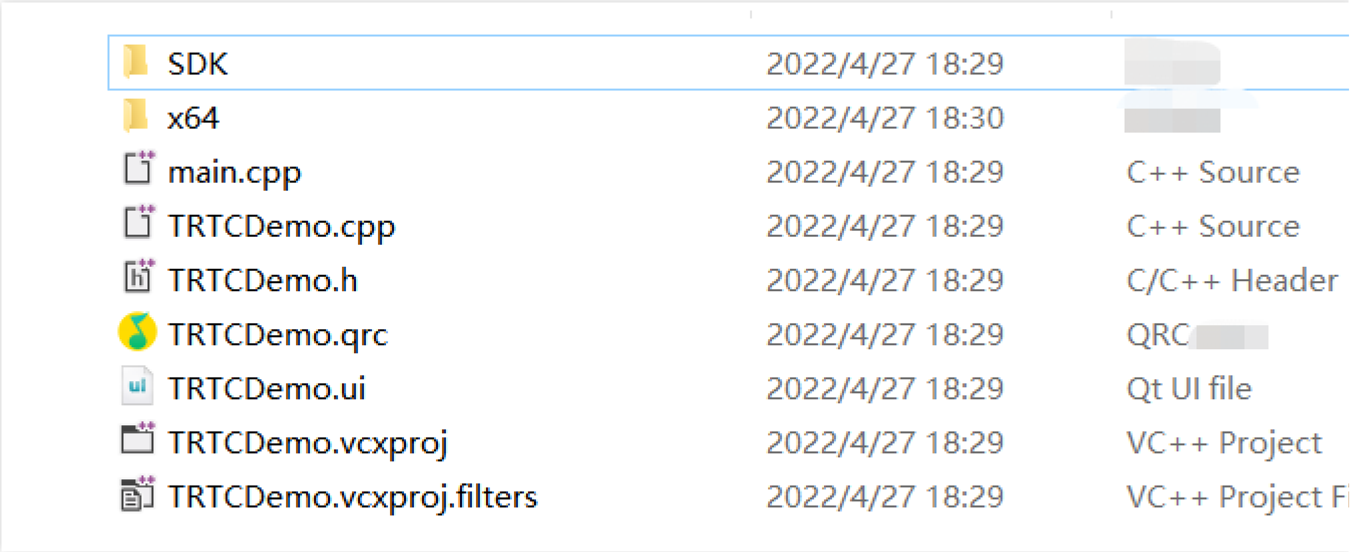
For the example in this guide, **Qt Widgets Application** is chosen. Click **Add**, and then click **Next** in subsequent steps until the project is created.

Step 3. Copy and paste files

Copy the `SDK` folder to the directory where `TRTCDemo.vcxproj` is located.

Note:

Because you will only need the C++ SDK, you can delete the `CSharp` folder in `SDK` .



Step 4. Modify project configuration

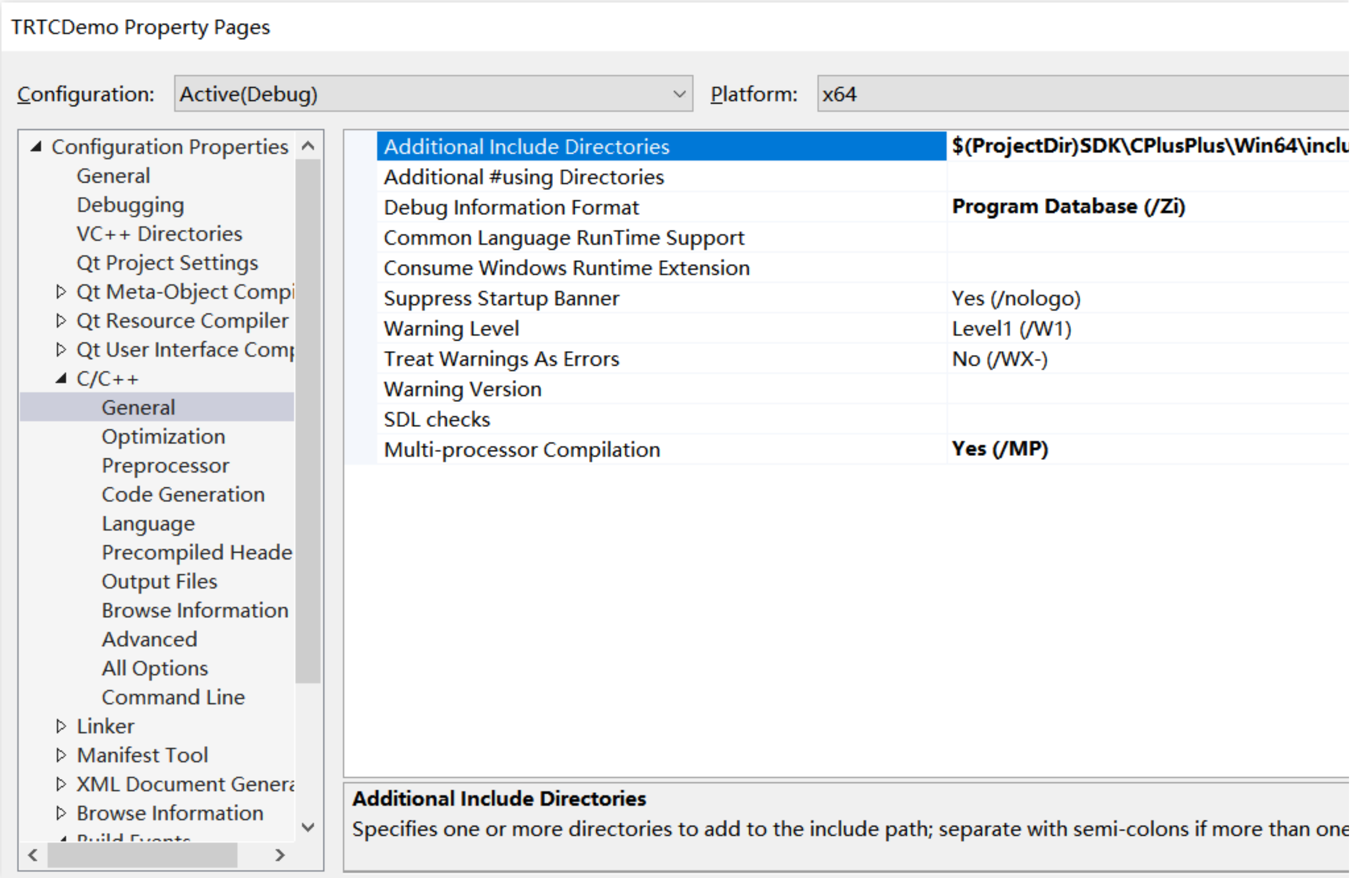
Select **Solution Explorer**, right-click `TRTCDemo` , and select **Properties**. Configure the project as follows:

1. **Add include directories.**

Go **C/C++** > **General**. Add the `$(ProjectDir)SDK\\CPlusPlus\\Win64\\include` and `$(ProjectDir)SDK\\CPlusPlus\\Win64\\include\\TRTC` header file directories (for 64-bit Windows) to **Additional Include Directories**.

Note:

For 32-bit Windows, add `$(ProjectDir)SDK\\CPlusPlus\\Win32\\include` and `$(ProjectDir)SDK\\CPlusPlus\\Win32\\include\\TRTC` .

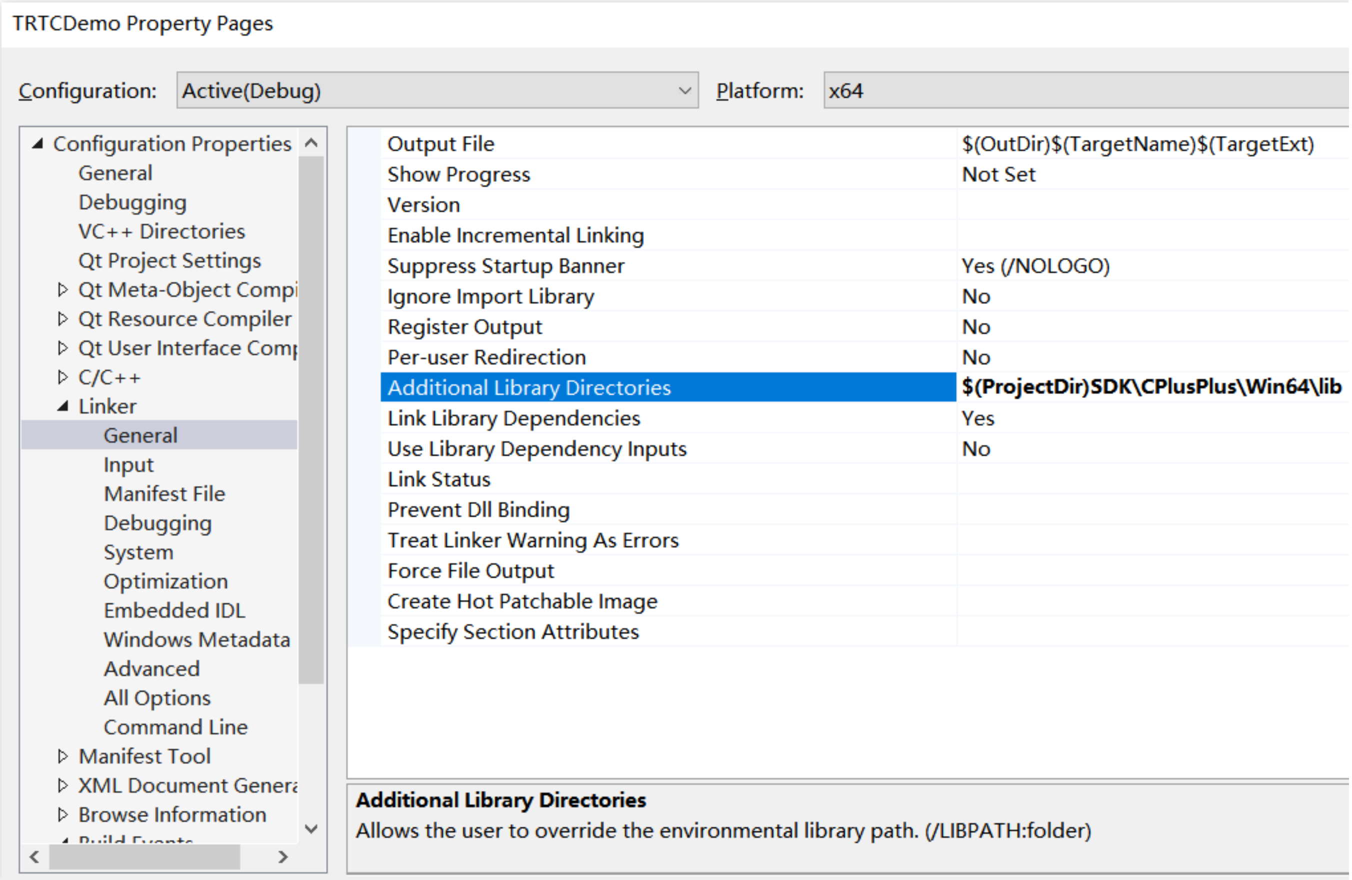


2. **Add additional library directories**

Go to **Linker** > **General**. Add the `$(ProjectDir)SDK\\CPlusPlus\\Win64\\lib` directory (for 64-bit Windows) to Additional **Library Directories**.

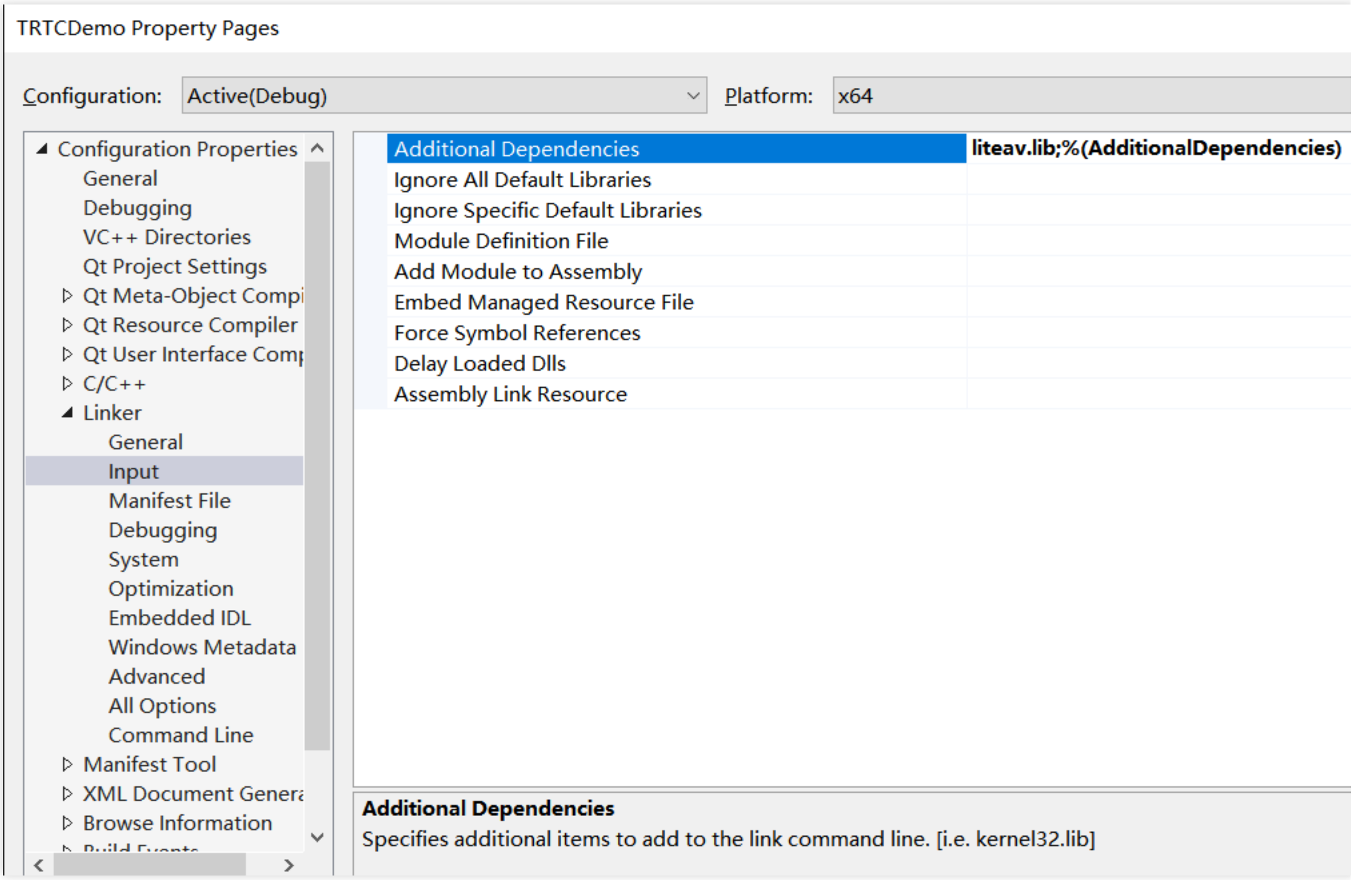
Note:

For 32-bit Windows, add `$(ProjectDir)SDK\\CPlusPlus\\Win32\\lib`.



3. **Add the library file**

Go to **Linker** > **Input**, and add the library file `liteav.lib` to **Additional Dependencies**.

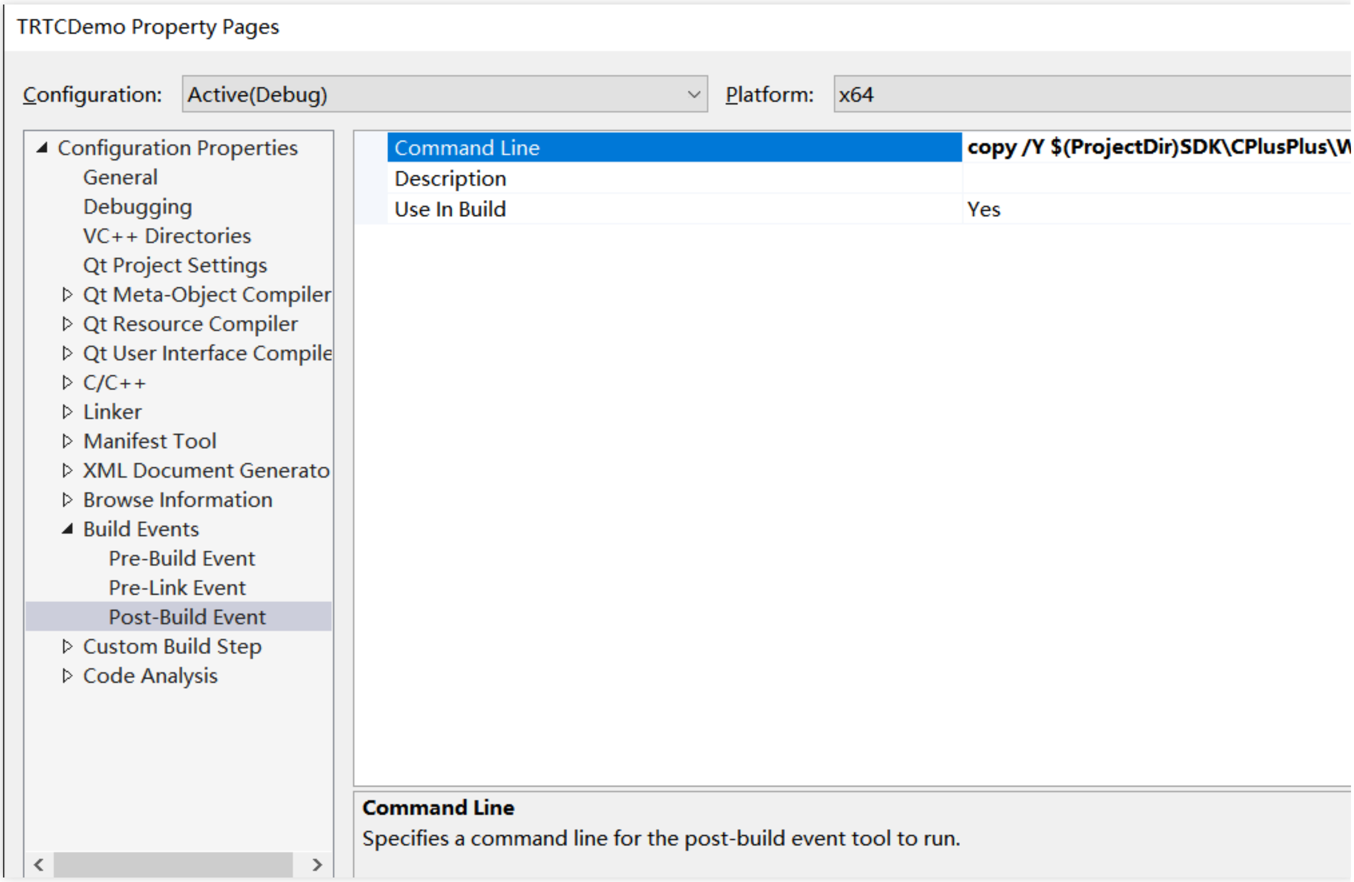


4. **Add the copy command**

Go to **Build Events** > **Post-build Events** and add the copy command `copy /Y $(ProjectDir)SDK\\CPlusPlus\\Win64\\lib\\*.dll $(OutDir)` (for 64-bit Windows) to **Command Line**. This ensures that the DLL files of the SDK are automatically copied to the project's execution directory after compilation.

Note:

For 32-bit Windows, add `copy /Y $(ProjectDir)SDK\\CPlusPlus\\Win32\\lib\\*.dll $(OutDir)` .



Step 5. Print the SDK version number

1. At the top of the `TRTCDemo.cpp` file, add the code below to import the header file:

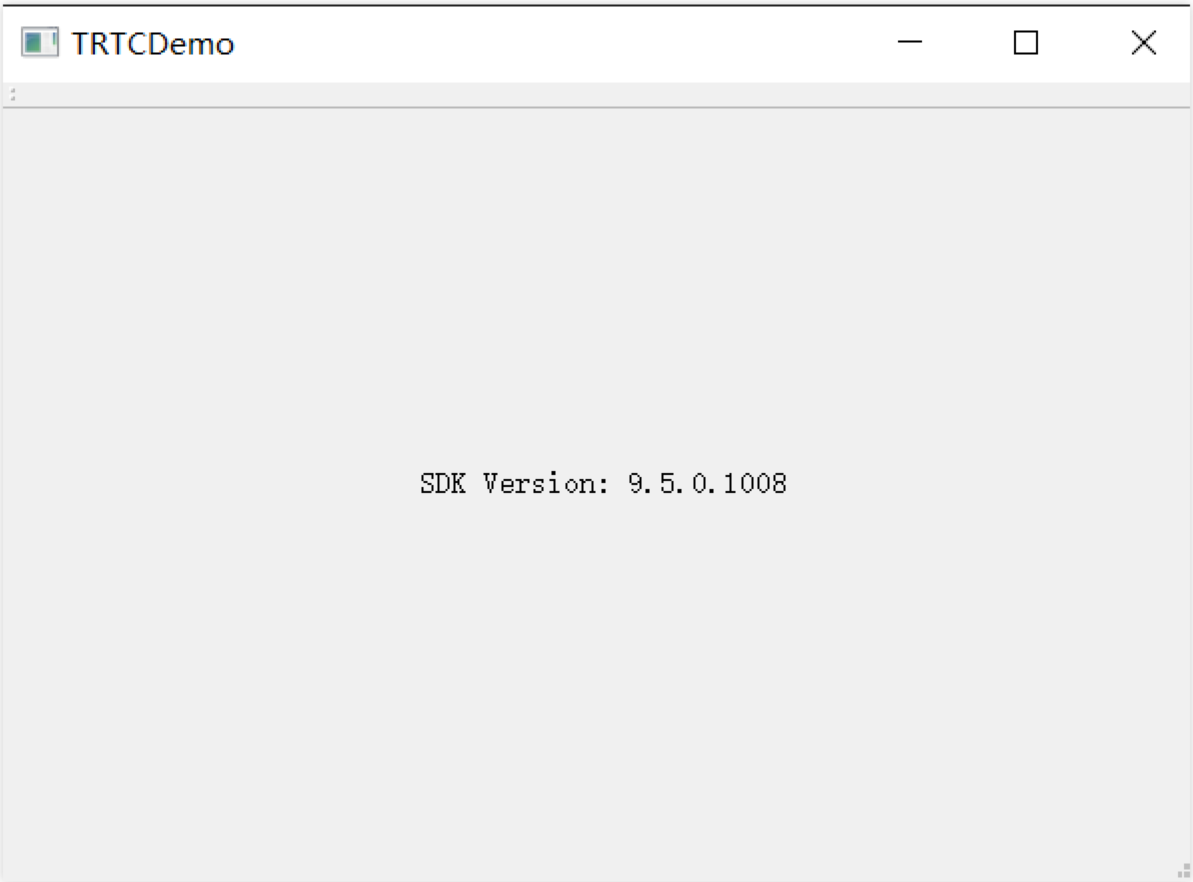
```
#include "ITRTCCloud.h"
#include <QLabel>
```

2. In the `TRTCDemo::TRTCDemo` constructor of `TRTCDemo.cpp` , add the following testing code:

```
ITRTCCloud * pTRTCCloud = getTRTCShareInstance();
std::string version(pTRTCCloud->getSDKVersion());

QString sdk_version = QString("SDK Version: %1").arg(version.c_str());
QLabel* label_text = new QLabel(this);
label_text->setAlignment(Qt::AlignCenter);
label_text->resize(this->width(), this->height());
label_text->setText(sdk_version);
```

3. Press F5 to run the project and print the version number of the SDK.



Integration for macOS

Development environment requirements

OS: OS X 10.10 or later

Development environment: Qt Creator 4.10.3 or later (Qt Creator 4.13.3 or later is recommended.)

Development framework: Qt 5.10 or later

Directions

This section uses a QtTest project as an example to show you how to integrate the TRTC C++ SDK into your project in Qt Creator.

1. **Download the cross-platform C++ SDK**

1.1 Download the SDK, and decompress and open the file.

1.2 Create an empty folder for the SDK in the same directory as your QtTest project, and copy `TXLiteAVSDKTRTCMacx.x.x/SDK/TXLiteAVSDKTRTC_Mac.framework` to the folder.

2. **Configure QTTest.pro**

Go to the directory of your QtTest project, open `QTTest.pro` with a text editor, and add the following SDK references.

```
INCLUDEPATH += $$PWD/.
DEPENDPATH += $$PWD/.

LIBS += "-F$$PWD/base/util/mac/usersig"
LIBS += "-F$$PWD/../SDK"
LIBS += -framework TXLiteAVSDK_TRTC_Mac
LIBS += -framework Accelerate
```

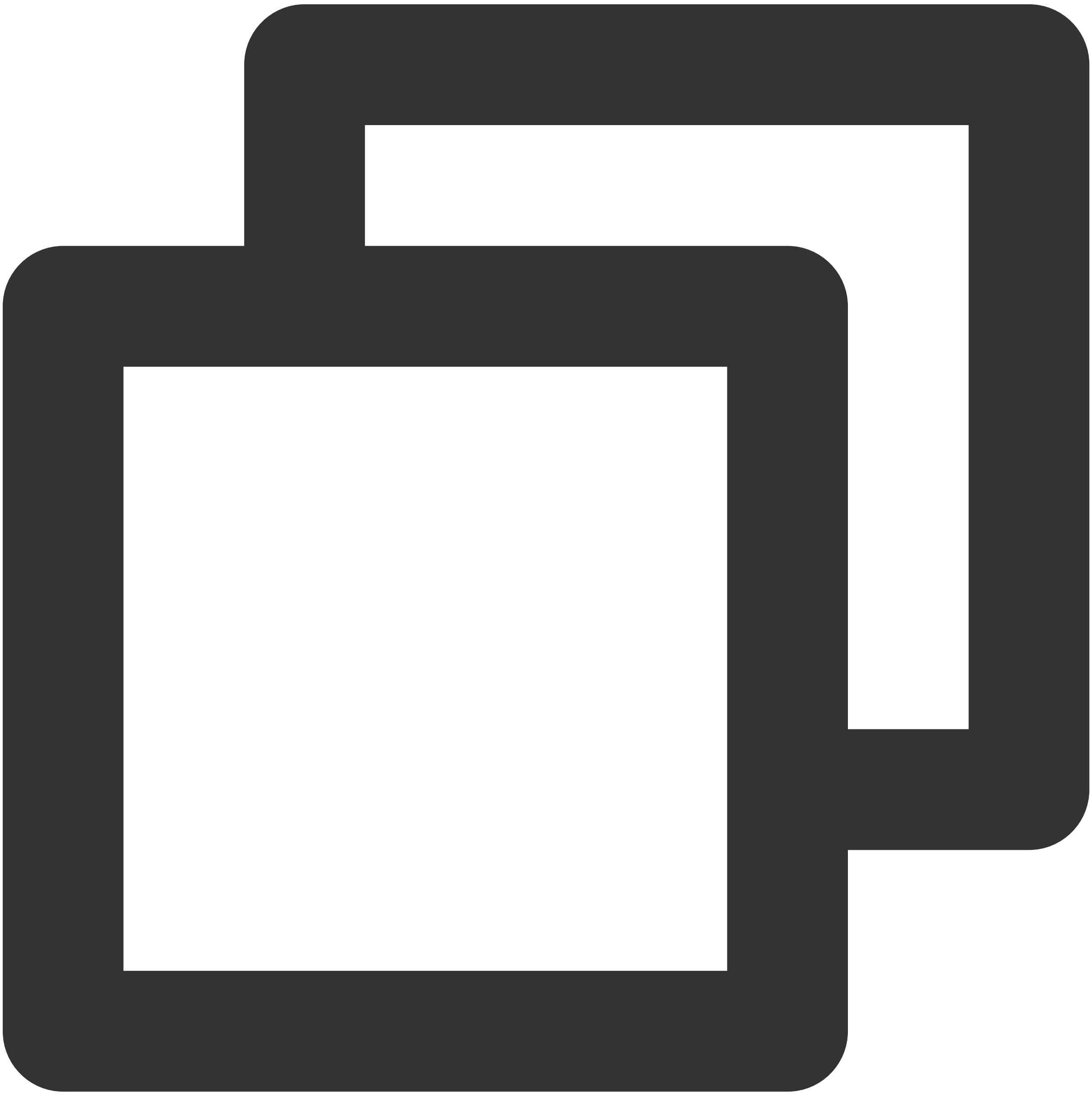
```
LIBS += -framework AudioUnit

INCLUDEPATH += $$PWD/../SDK/TXLiteAVSDK_TRTC_Mac.framework/Headers/cpp_interface

INCLUDEPATH += $$PWD/base/util/mac/usersig/include
DEPENDPATH += $$PWD/base/util/mac/usersig/include
```

3. **Grant camera and mic permissions**

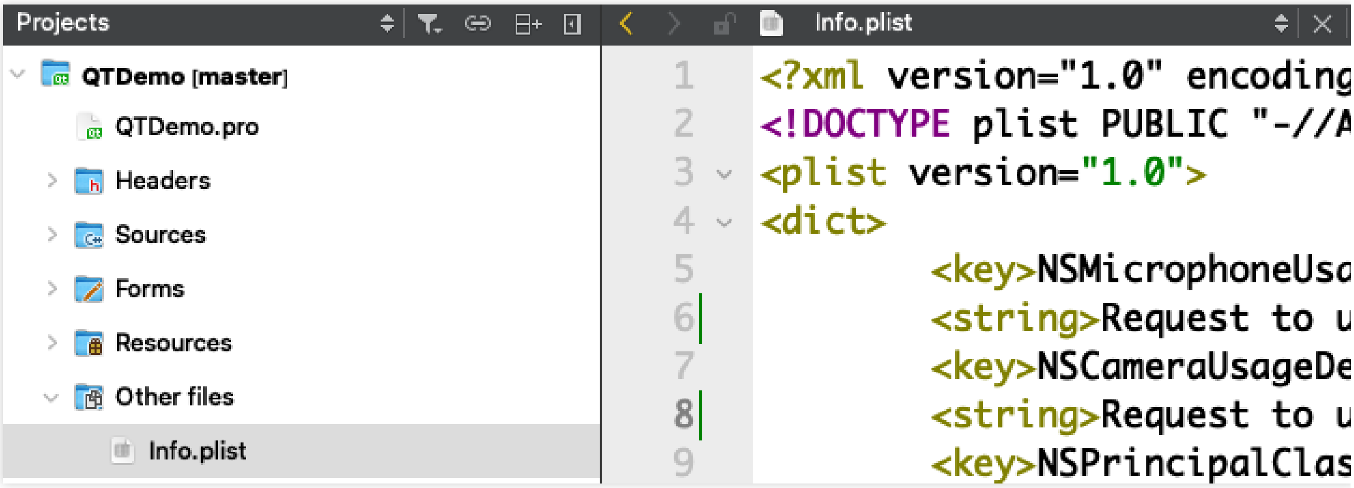
The TRTC SDK will use the device’s camera and mic, so you need to add permission requests to `Info.plist` .



```
NSMicrophoneUsageDescription: requesting mic access
```

```
NSCameraUsageDescription: requesting camera access
```

As shown below:



4. **Reference the TRTC SDK**

Reference the SDK via the header file: `#include "ITRTCCloud.h"` .

Use the namespace: The methods and types of cross-platform C++ APIs are all defined in the `trtc` namespace. To simplify your code, we recommend you use the `trtc` namespace.

Note:

This concludes the integration process, and you can proceed to compile your project. You can download QTDemo to get more information on the use of cross-platform APIs of the SDK.

FAQs

If the following error occurs, check whether the SDK header file directories are correctly added as described in the project configuration step above.


```
fatal error C1083: Could not open include file: “TRTCCloud.h”: No such file or dire
```

If the following error occurs, check whether the SDK library directory and library file are correctly added as described in the project configuration step above.

```
error LNK2019: unresolved external symbol "__declspec(dllimport) public: static cla
```

Unity

Last updated：2023-09-26 16:57:22

This document describes how to quickly integrate TRTC SDK for Unity into your project.

Environment Requirements

Unity 2020.2.1f1c1 is recommended.

Supported platforms: Android, iOS, Windows, macOS (alpha testing)

Modules required: `Android Build Support` , `iOS Build Support` , `Windows Build Support` , `MacOS Build Support`

If you are developing for iOS, you also need:

Xcode 11.0 or above

A valid developer signature for your project

Integrating SDK

1. Download the SDK and demo source code.
2. Decompress the ZIP file and copy `TRTCUnitySDK/Assets/TRTCSDK/SDK` to the `Assets` directory of your project.

FAQs

What should I do if a network access error occurs on Android?

Copy `/Assets/Plugins/AndroidManifest.xml` to the same directory of your project.

What should I do if the SDK does not have mic or camera access on Android?

You need to add mic and camera permission requests manually when building for Android. For details, see the code below:

```
#if PLATFORM_ANDROID
if (!Permission.HasUserAuthorizedPermission(Permission.Microphone))
 {
     Permission.RequestUserPermission(Permission.Microphone);
 }
 if (!Permission.HasUserAuthorizedPermission(Permission.Camera))
 {
     Permission.RequestUserPermission(Permission.Camera);
 }
 #endif
```


React Native

Last updated：2024-05-21 15:05:29

This document describes how to quickly integrate the TRTC SDK for React Native into your project.

Environment Requirements

React Native 0.63 or above

Node (above v12) & Watchman

Developing for Android:

Android Studio 3.5 or above

Devices with Android 4.1 or above

Java Development Kit

Developing for iOS and macOS:

Xcode 11.0 or above

OS X 10.11 or above

A valid developer signature for your project

For how to set up the environment, see the React Native official document.

Integrating SDK

We have released the TRTC SDK for React Native to npm. You can configure `package.json` to install the SDK.

1. Add the following dependencies to `package.json` of your project:

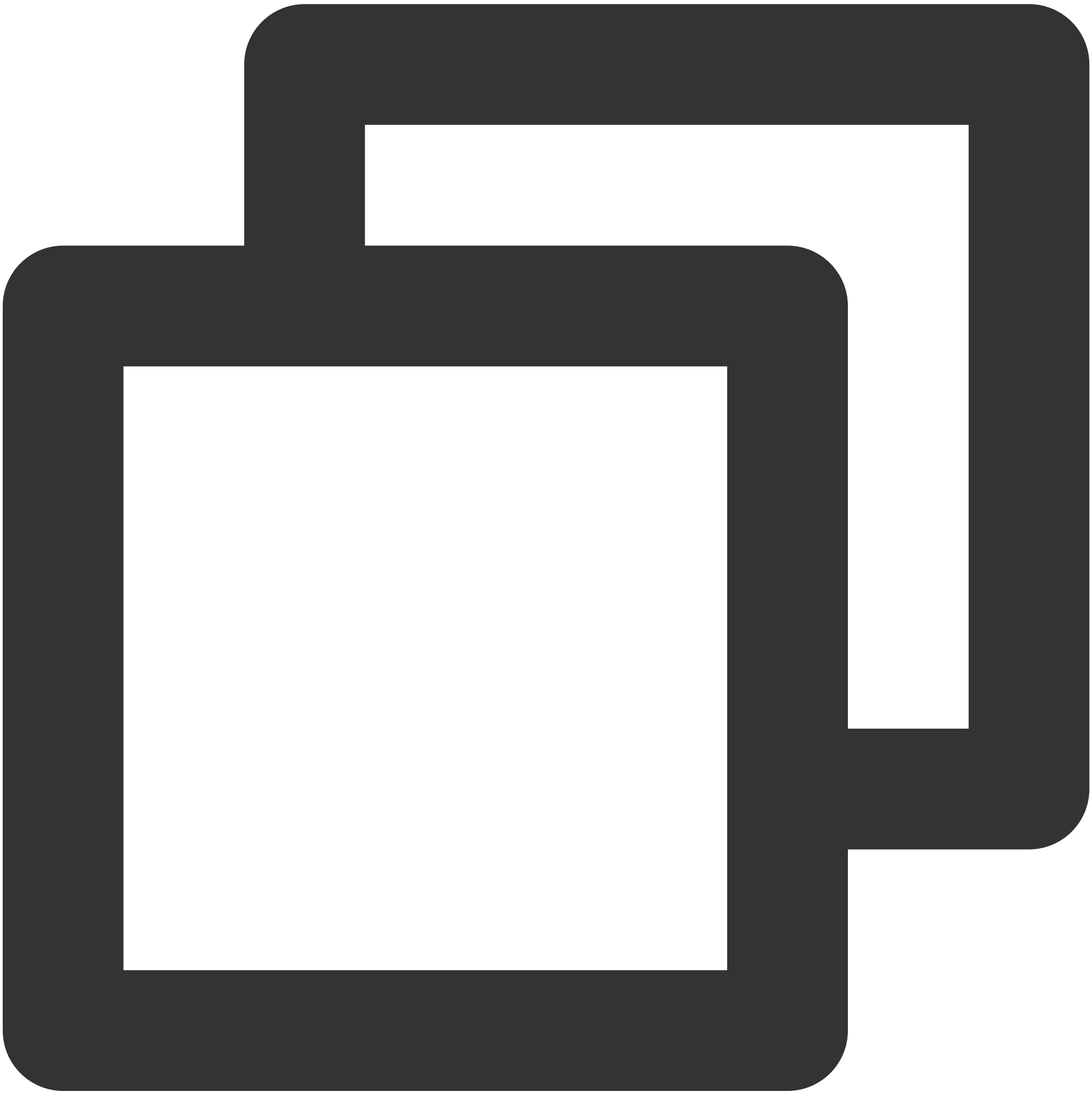
```
"dependencies": {
  "trtc-react-native": "^2.0.0"
},
```

2. Grant **camera** and **mic** permissions to enable the audio and video call features.

Android

iOS

1. Configure application permissions in `AndroidManifest.xml` . The TRTC SDK requires the following permissions:



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

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

Note:

Do not set `android:hardwareAccelerated="false"` . Disabling hardware acceleration will result in failure to render remote users' videos.

2. Manually configure audio and video permission requests.

```
if (Platform.OS === 'android') {
  await PermissionsAndroid.requestMultiple([
 PermissionsAndroid.PERMISSIONS.RECORD_AUDIO, //For audio calls
 PermissionsAndroid.PERMISSIONS.CAMERA, // For video calls
```



```
  ]);
}
```

1. Add requests for camera and mic permissions in `Info.plist` :

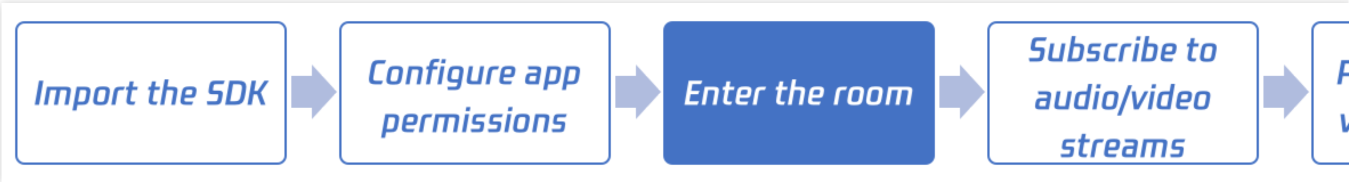
```
<key>NSCameraUsageDescription</key>
<string>Video calls are possible only with camera permission.</string>
<key>NSMicrophoneUsageDescription</key>
<string>Audio calls are possible only with mic access.</string>
```

2. Link native libraries. For detailed directions, see Linking Libraries.

02. Entering a Room
Android, iOS, Windows, and macOS

Last updated：2024-01-24 16:06:40

This document describes how to enter a TRTC room. Only after entering an audio/video room can a user subscribe to the audio/video streams of other users in the room or publish his or her own audio/video streams.



Call Guide

Step 1. Import the SDK and set the application permissions

Import the SDK as instructed in iOS.

Step 2. Create an SDK instance and set an event listener

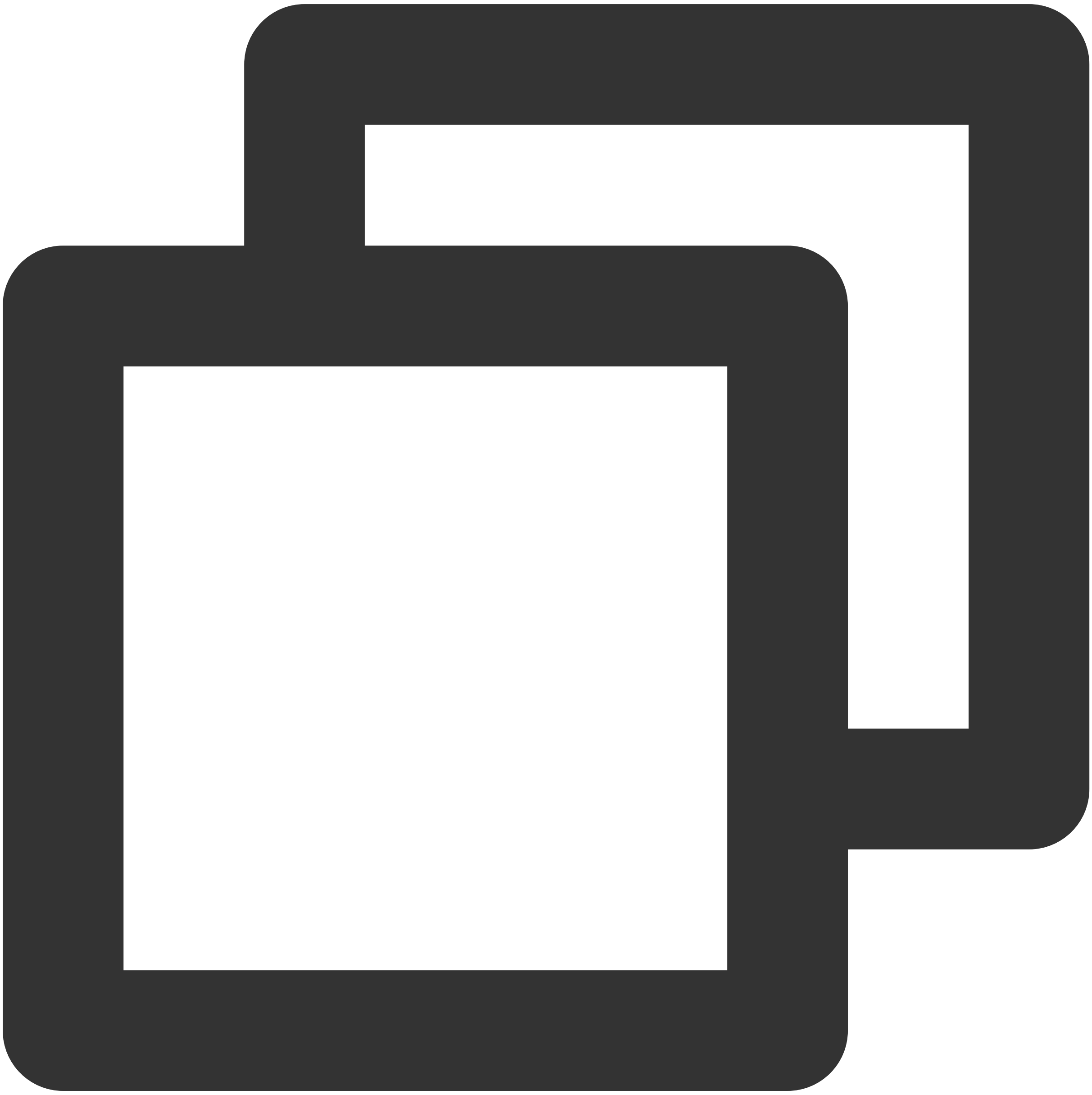
Call the initialization API to create a TRTC object instance.

Android

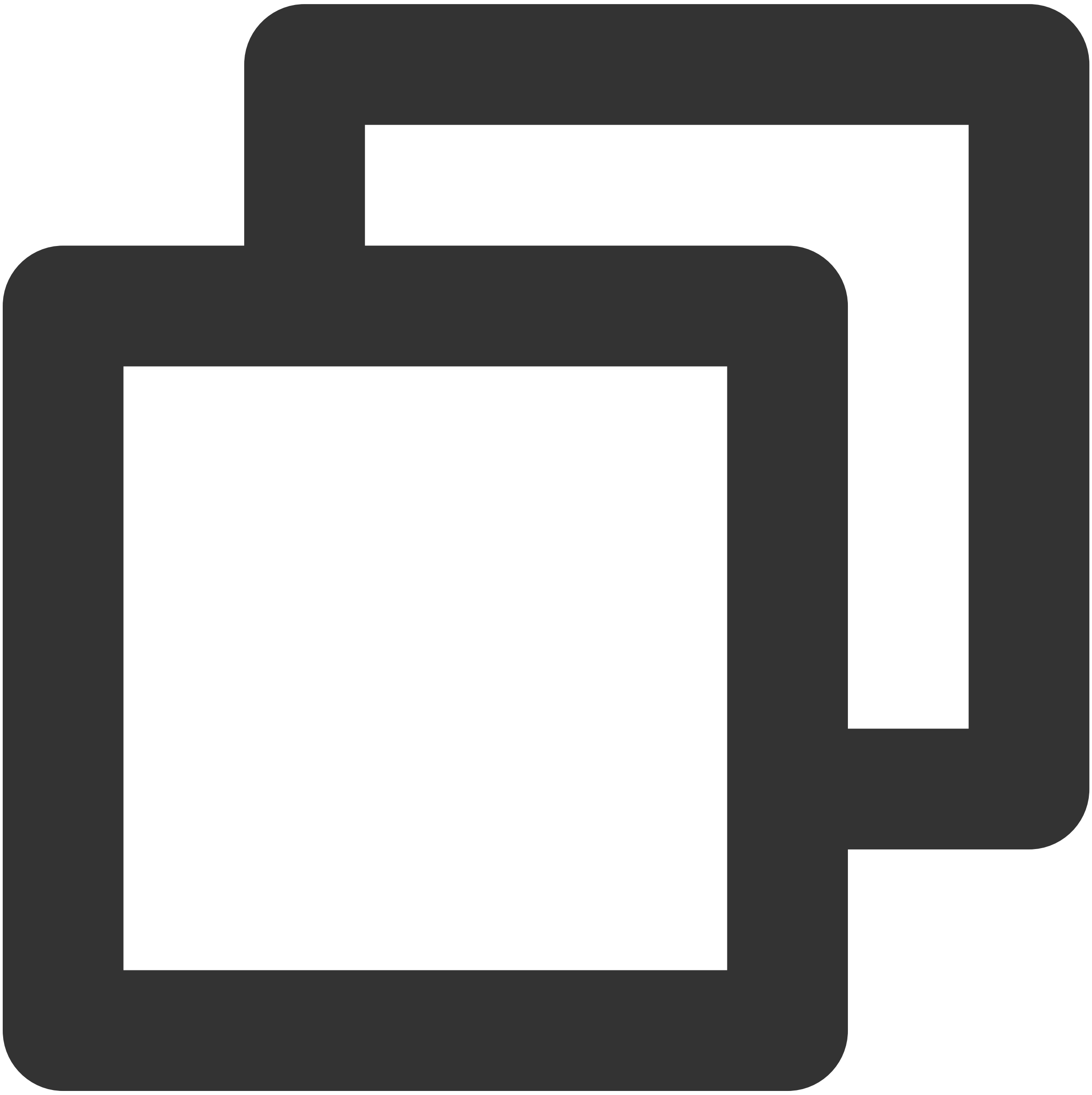
iOS&Mac

Windows

```
// Create an SDK instance in singleton mode and set an event listener
// Create trtc instance(singleton) and set up event listeners
mCloud = TRTCCloud.sharedInstance(getApplicationContext());
mCloud.setListener(this);
```



```
// Create an SDK instance in singleton mode and set an event listener
// Create trtc instance(singleton) and set up event listeners
_trtcCloud = [TRTCCloud sharedInstance];
_trtcCloud.delegate = self;
```



```
// Create an SDK instance in singleton mode and set an event listener
// Create trtc instance (singleton) and set up event listeners
trtc_cloud_ = getTRTCShareInstance();
trtc_cloud_->addCallback(this);
```

Step 3. Listen for SDK events

You can use the `setListener` API to set callbacks and listen for errors, warnings, traffic statistics, and network quality information as well as various audio/video events during SDK operations.

Android
iOS&Mac
Windows

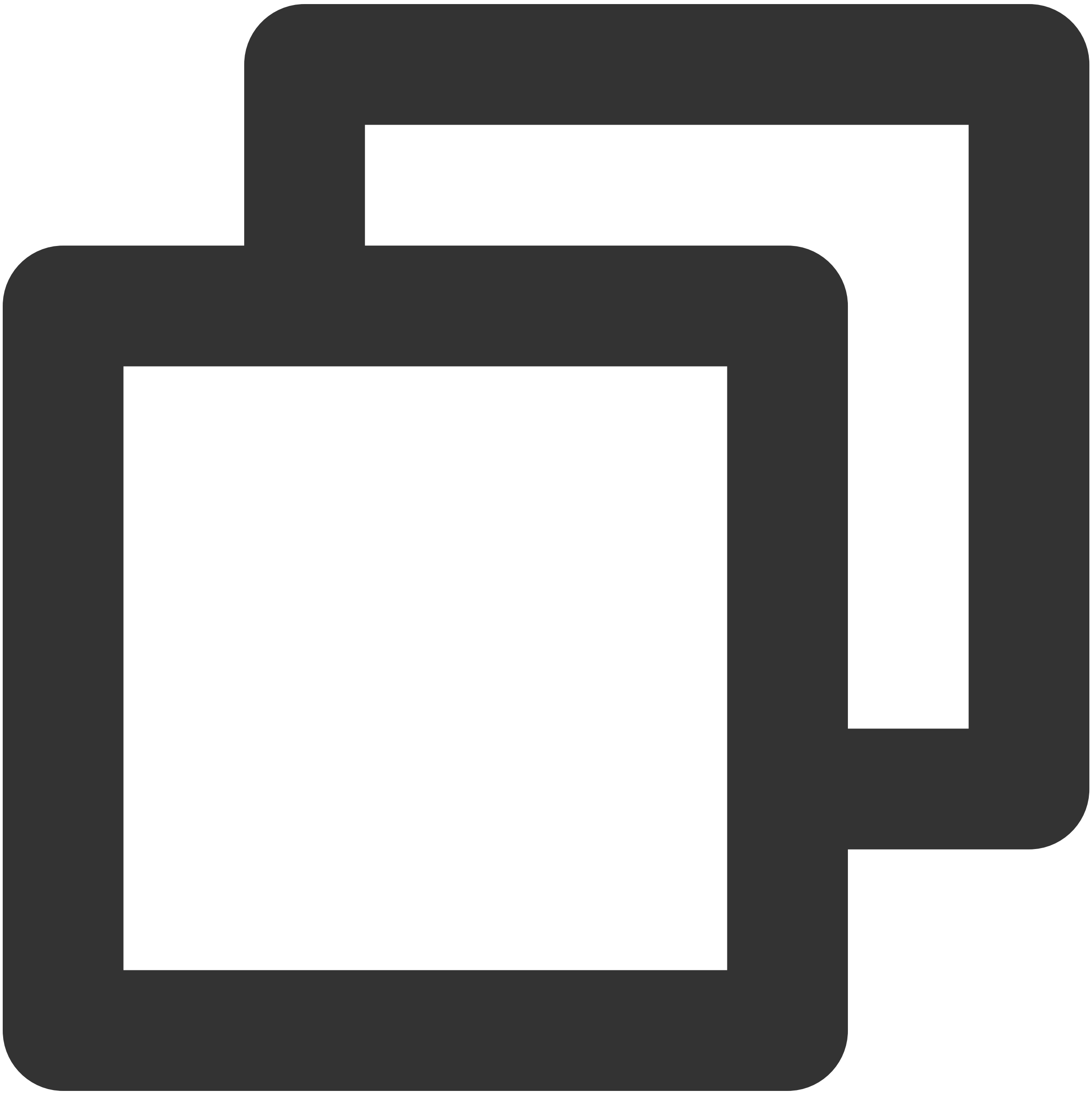
```
You can make your business class inherit **TRTCCloudListener**, reload the `onError
```java
// Listen for SDK events, and print logs for error messages such as "Current applic
// Listen to the "onError" event, and print logs for errors such as "Camera is not
mCloud = TRTCCloud.sharedInstance(getApplicationContext());
mCloud.setListener(this);
```
```

````
@Override
public void onError(int errCode, String errMsg, Bundle extraInfo) {
    if (errCode == TXLiteAVCode.ERR_CAMERA_NOT_AUTHORIZED) {
        Log.d(TAG, "Current application is not authorized to use the camera");
    }
}
```
````

````
You can make your business class inherit **TRTCCloudDelegate**, reload the `onError
```ObjectiveC
````

```
// Listen for SDK events, and print logs for error messages such as "Current applic
// Listen to the "onError" event, and print logs for errors such as "Camera is not 
_trtcCloud = [TRTCCloud sharedInstance];
_trtcCloud.delegate = self;

- (void)onError:(TXLiteAVError)errCode
         errMsg:(nullable NSString *)errMsg
        extInfo:(nullable NSDictionary *)extInfo{
    if (ERR_CAMERA_NOT_AUTHORIZED == errCode) {
        NSString *errorInfo = @"Current application is not authorized to use the ca
        errorInfo = [errorInfo stringByAppendingString errMsg];
        [self toastTip:errorInfo];
    }
}
```
```
```



```
You can make your business class inherit **ITRTCCloudCallback**, reload the `onErro
```C++
// Listen for SDK events, and print logs for error messages such as "Current applic
// Listen to the "onError" event, and print logs for errors such as "Camera is not
trtc_cloud_ = getTRTCShareInstance();
trtc_cloud_->addCallback(this);

// Reload the `onError` function
void onError(TXLiteAVError errCode, const char* errMsg, void* extraInfo) {
    if (errCode == ERR_CAMERA_NOT_AUTHORIZED) {
        printf("Current application is not authorized to use the camera");
```
```

```
    }
}
```
```

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

When calling the `enterRoom` API, you need to enter two key parameters: `TRTCParams` and `TRTCAppScene`.

**Parameter 1: TRTCAppScene**

This parameter is used to specify whether your application scenario is **live streaming** or **real-time call**.

**Real-time call:**

Real-time call includes `TRTCAppSceneVideoCall` (video call) and `TRTCAppSceneAudioCall` (audio call). This mode is suitable for one-to-one audio/video calls or online meetings for up to 300 attendees.

**Live streaming:**

Live streaming includes `TRTCAppSceneLIVE` (video live streaming) and `TRTCAppSceneVoiceChatRoom` (audio live streaming). This mode is suitable for live streaming to up to 100,000 users. However, it requires you to specify the **role** field in the `TRTCParams` parameter for users in the room: **anchor** or **audience**.

**Parameter 2: TRTCParams**

`TRTCParams` consists of many fields; however, you usually only need to pay attention to how to set the following fields:

| Parameter | Description | Remarks | Data Type | Sample Value |
| --- | --- | --- | --- | --- |
| SDKAppID | Application ID | You can view the `SDKAppID` in the TRTC console. If it doesn't exist, click **Create Application** to create an application. | Numeric | 1400000123 |
| userId | User ID | Username. It can contain only letters, digits, underscores, and hyphens. In TRTC, a user cannot use the same `userId` to enter the same room on two different devices at the same time. | String | `denny` or `123321` |
| userSig | Room entry signature | You can calculate `userSig` based on `SDKAppID` and `userId` as instructed in UserSig. | String | eJyrVareCeYrSy1SslI... |
| roomId | Room ID | Room ID of the numeric type. To use | Number | 29834 |
```

		a string-type room ID, use only the **strRoomId** field instead of the `roomId` field, as they cannot be used together.		
strRoomId	Room ID	Room ID of the string type. Do not use `strRoomId` and `roomId` at the same time. `"123"` and `123` are considered different rooms by the TRTC backend.	String	"29834"
role	Role	There are two roles: anchor and audience. This field is required only when `TRTCAppScene` is set to the `TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom` live streaming scenarios.	Enumeration	`TRTCRoleAnchor` or `TRTCRoleAudience`

Note

In TRTC, a user cannot use the same `userId` to enter the same room on two different devices at the same time; otherwise, there will be a conflict.

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

Step 5. Enter the room (`enterRoom`)

After preparing the two parameters `TRTCAppScene` and `TRTCParams` described in step 4, you can call the `enterRoom` API to enter the room.

Android

iOS&Mac

Windows

```
mCloud = TRTCCloud.sharedInstance(getApplicationContext());
mCloud.setListener(mTRTCCloudListener);

// Assemble TRTC room entry parameters. Replace the field values in `TRTCParams` wi
// Replace each field in TRTCParams with your own parameters
TRTCCloudDef.TRTCParams param = new TRTCCloudDef.TRTCParams();
params.sdkAppId = 1400000123;  // Replace with your own SDKAppID
params.userId = "denny";       // Replace with your own user ID
params.roomId = 123321;        // Replace with your own room number
params.userSig = "xxx";        // Replace with your own userSig
params.role = TRTCCloudDef.TRTCRoleAnchor;
```



```
// If your scenario is live streaming, set the application scenario to `TRTC_APP_SC
// If your application scenario is a group video call, use "TRTC_APP_SCENE_LIVE"
mCloud.enterRoom(param, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
self.trtcCloud.delegate = self;

// Assemble TRTC room entry parameters. Replace the field values in `TRTCParams` wi
// Replace each field in TRTCParams with your own parameters
```



```
TRTCParams *params = [[TRTCParams alloc] init];
params.sdkAppId = 1400000123;  // Replace with your own SDKAppID
params.roomId = 123321; // Replace with your own room number
params.userId = @"denny";   // Replace with your own userid
params.userSig = @"xxx"; // Replace with your own userSig
params.role = TRTCRoleAnchor;

// If your scenario is live streaming, set the application scenario to `TRTC_APP_SC
// If your application scenario is a group video call, use "TRTC_APP_SCENE_LIVE"
[self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
```

```
trtc_cloud_ = getTRTCShareInstance();
trtc_cloud_->addCallback(this);

// Assemble TRTC room entry parameters. Replace the field values in `TRTCParams` wi
// Replace each field in TRTCParams with your own parameters
liteav::TRTCParams params;
params.sdkAppId = 1400000123;  // Replace with your own SDKAppID
params.userId = "denny";       // Replace with your own user ID
params.roomId = 123321;        // Replace with your own room number
params.userSig = "xxx";        // Replace with your own userSig
params.role = liteav::TRTCRoleAnchor;

// If your scenario is live streaming, set the application scenario to `TRTC_APP_SC
// If your application scenario is a group video call, use "TRTC_APP_SCENE_LIVE"
trtc_cloud_->enterRoom(params, liteav::TRTCAppSceneLIVE);
```

Event callback

If room entry succeeds, the SDK will call back the `onEnterRoom(result)` event. Here, `result` is a value greater than 0, indicating the time in milliseconds taken to enter the room.

If room entry fails, the SDK will also call back the `onEnterRoom(result)` event, but the value of `result` will be a negative number, indicating the error code for the room entry failure.

Android

iOS&Mac

Windows


```
    }
```

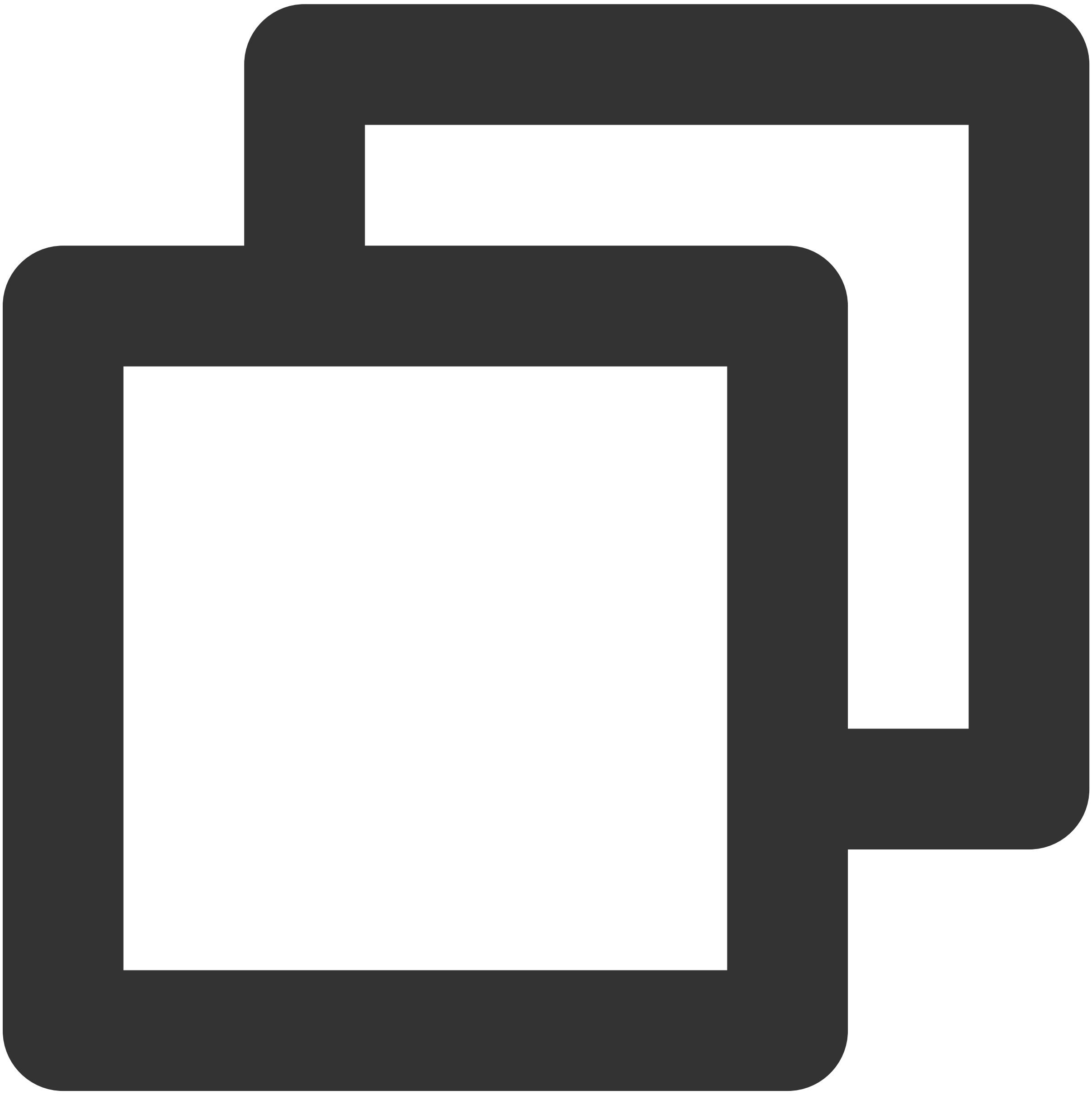
```
// Listen for the `onEnterRoom` event of the SDK to get the room entry result
// Listen for the `onEnterRoom` event of the SDK and learn whether the room is succ
@Override
public void onEnterRoom(long     result) {
    if (result > 0) {
        Log.d(TAG, "Enter room succeed");
    } else {
        Log.d(TAG, "Enter room failed");
    }
}
```

```
}
```

```
// Listen for the `onEnterRoom` event of the SDK to get the room entry result
// Listen for the `onEnterRoom` event of the SDK and learn whether the room is succ
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        [self toastTip:@"Enter room succeed!"];
    } else {
        [self toastTip:@"Enter room failed!"];
    }
}
```



```
}
```

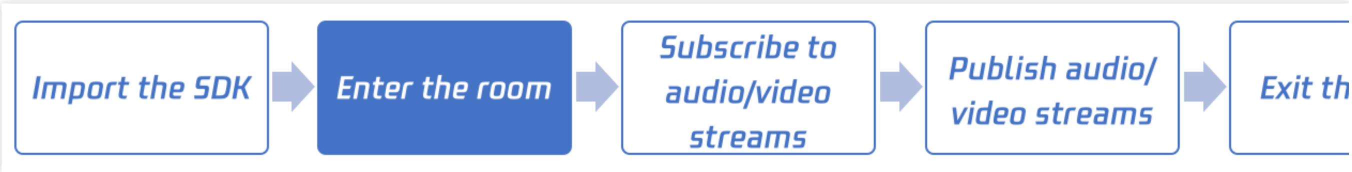


```
// Listen for the `onEnterRoom` event of the SDK to get the room entry result
// override to the onEnterRoom event of the SDK and learn whether the room is succe
void onEnterRoom(int result) {
    if (result > 0) {
        printf("Enter room succeed");
    } else {
        printf("Enter room failed");
    }
}
```


Web

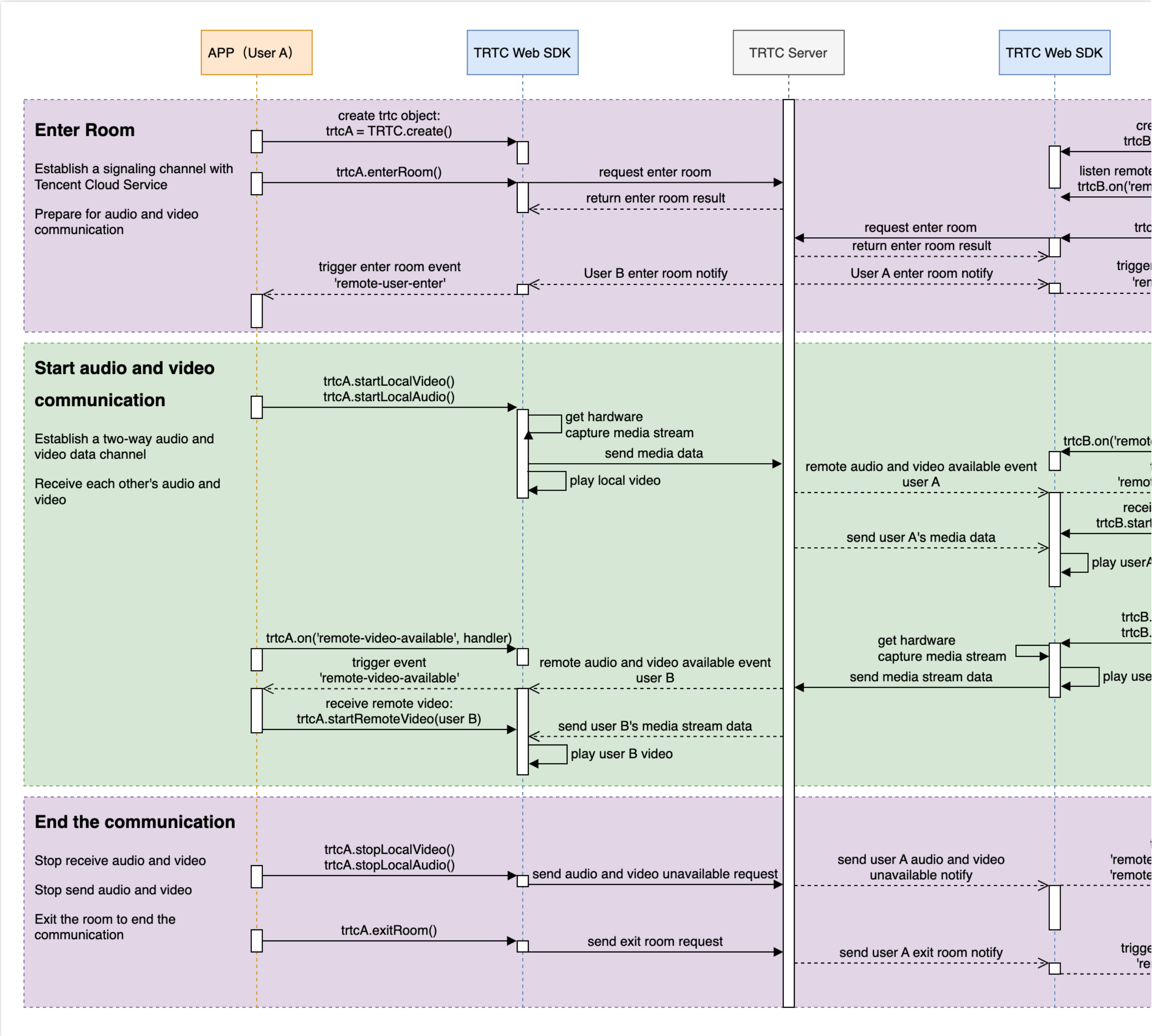
Last updated：2024-06-28 15:03:32

This document describes how to enter a TRTC room. Only after entering an audio/video room can a user subscribe to the audio/video streams of other users in the room or publish his or her own audio/video streams.



SDK Usage Overview

1. Call the `TRTC.create()` method to create the `trtc` object.
2. Call the `trtc.enterRoom()` method to enter the room, then other users will receive the TRTC.EVENT.REMOTE_USER_ENTER event.
3. After entering the room, you can turn on the camera and microphone and publish them to the room.
Call the `TRTC.startLocalVideo()` method to turn on the camera and publish it to the room.
Call the `TRTC.startLocalAudio()` method to turn on the microphone and publish it to the room.
4. When a remote user publishes audio and video, the SDK will automatically play the remote audio by default. You need to play the remote video by:
Listen for the TRTC.EVENT.REMOTE_VIDEO_AVAILABLE event before entering the room to receive all remote user video publishing events.
In the event callback function, call the `trtc.startRemoteVideo()` method to play the remote video.



Step 1. Creating a TRTC object

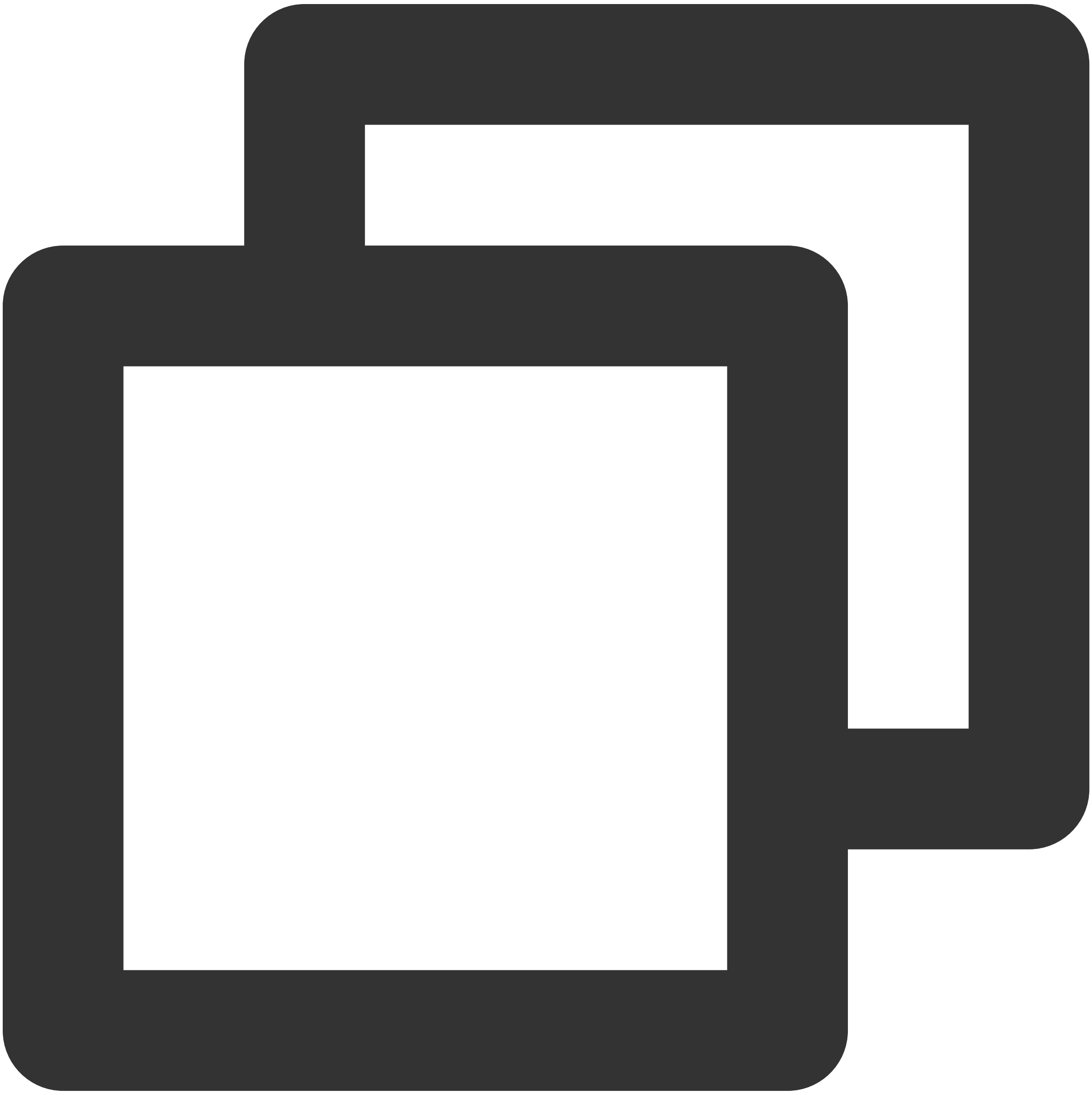
`TRTC` class, whose instance represents a local client. The object methods of TRTC provide functions such as joining a call room, previewing a local camera, publishing a local camera and microphone, and playing remote audio and video.

Create a `TRTC` object through the `TRTC.create()` method

```
const trtc = TRTC.create();
```

Note:

If you use the Vue3 framework, it is necessary to use markRaw to mark the trtc instance in order to avoid the conversion of trtc into the Proxy object by Vue, which may cause some unexpected problems.



```
import { markRaw } from 'vue';
const trtc = markRaw(TRTC.create());
```

Step 2. Entering the room

Call the `trtc.enterRoom()` method to enter the room. Usually called in the click callback of the `Start Call` button.

Key parameters:

Name	Description	Type	Example
sdkAppId	The sdkAppId of the audio and video application you created on Tencent Cloud.	number	1400000123
userId	It is recommended to limit the length to 32 bytes, and only allow uppercase and lowercase English letters (a-zA-Z), numbers (0-9), underscores, and hyphens.	string	"mike"
userSig	User signature, refer to UserSig.	string	eJyrVareCeYrSy1SslI...
roomId	Numeric type roomId. The value must be an integer between 1 and 4294967294. If you need to use a string type roomId, please use the `strRoomId` parameter. One of `roomId` and `strRoomId` must be passed in. If both are passed in, the roomId will be selected first.	number	29834
strRoomId	String type roomId. The length is limited to 64 bytes, and only supports the following characters: Uppercase and lowercase English letters (a-zA-Z) Numbers (0-9) Space ! # $ % & ( ) + - : ; < = . > ? @ [ ] ^ _ { } | Note: It is recommended to use a numeric type roomId. The string type room id "123" is not the same room as the numeric type room id 123.	string	"29834"
scene	`rtc` : Real-time call scene. `live` : Interactive live streaming scene	string	'rtc' or 'live'
role	User role, only works for `live` scene `anchor` `audience` The audience role does not have the permission to publish local audio and video, only the permission to watch remote streams. If the audience wants to interact with the anchor by connecting to the microphone, please switch the role to the anchor through trtc.switchRole() before publishing local audio and video.	string	'anchor' or 'audience'

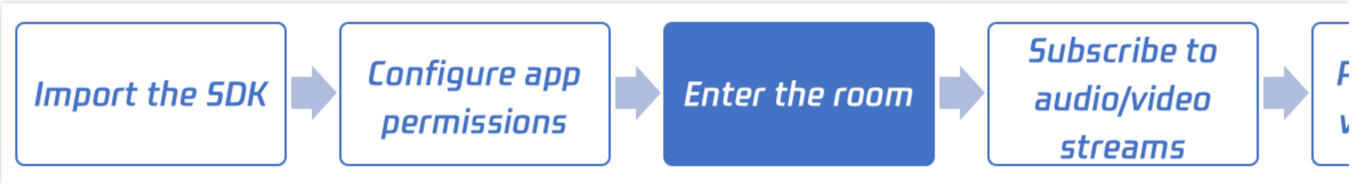
For more detailed parameter descriptions, refer to the interface document `trtc.enterRoom()` .


```
try {
  await trtc.enterRoom({ roomId: 8888, scene:'rtc', sdkAppId, userId, userSig });
  console.log('Entered the room successfully');
} catch (error) {
  console.error('Failed to enter the room ' + error);
}
```

Electron

Last updated：2024-01-24 16:05:27

This document describes how to enter a TRTC room. Only after entering an audio/video room can a user subscribe to the audio/video streams of other users in the room or publish his or her own audio/video streams.

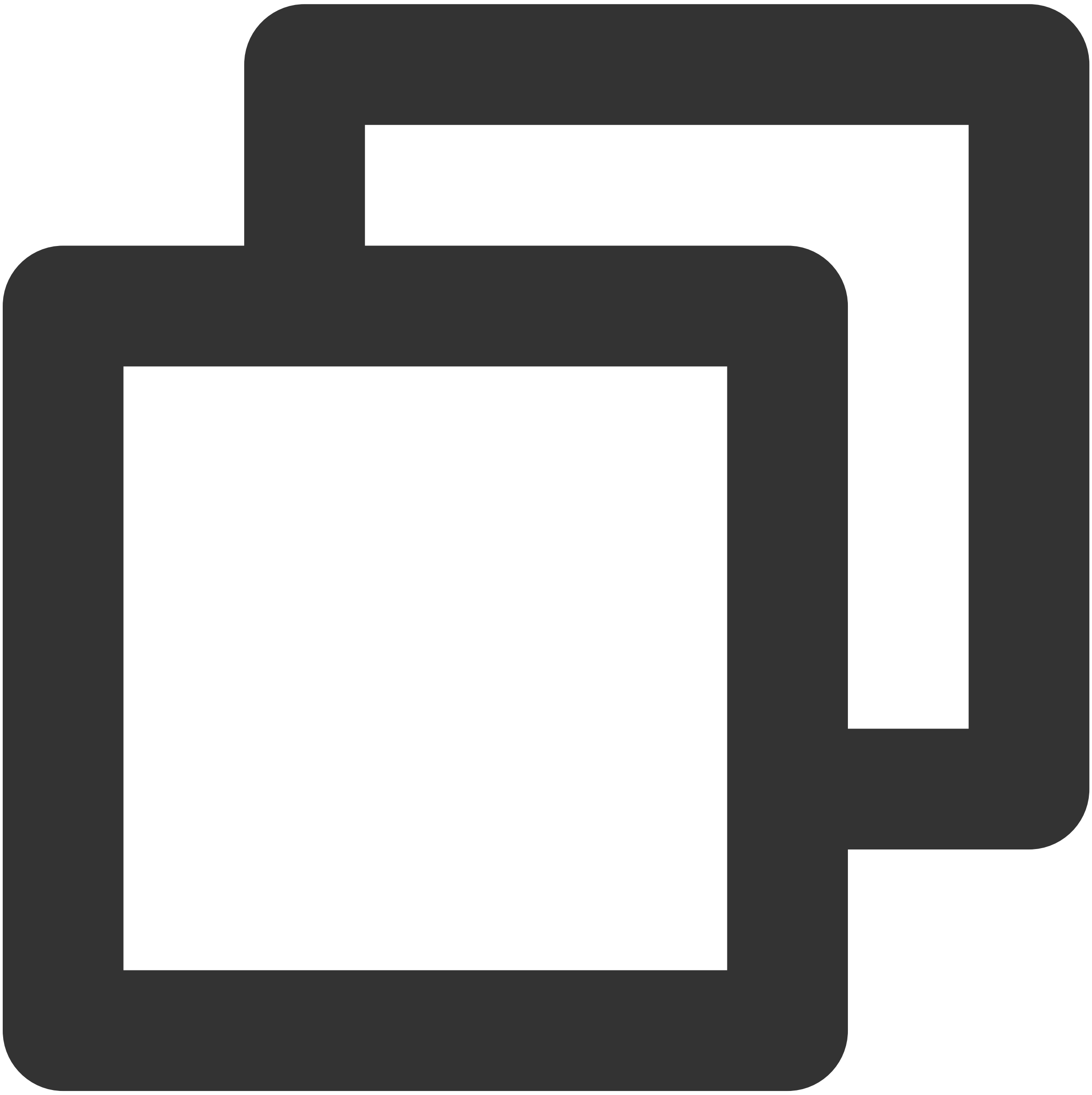


Call Guide

Step 1. Import the SDK

Import the SDK as instructed in Electron.

Step 2. Create an SDK instance



```
import TRTCCloud from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();
```

Step 3. Listen for SDK events

You can use callback APIs to listen for errors, warnings, traffic statistics, network quality, as well as various audio/video events of the SDK.

```
function onError(errCode, errMsg) {
  // For information on the error codes, see https://intl.cloud.tencent.com/documen
  console.log(errCode, errMsg);
}

function onWarning(warningCode, warningMsg) {
  // For information on the warning codes, see https://intl.cloud.tencent.com/docum
  console.log(warningCode, warningMsg);
}

rtcCloud.on('onError', onError);
```



```
rtcCloud.on('onWarning', onWarning);
```

Step 4. Assemble the room entry parameter `TRTCParams`

When calling the `enterRoom` API, you need to enter two key parameters: `TRTCParams` and `TRTCAppScene` .

Parameter 1: TRTCAppScene

This parameter is used to specify whether your application scenario is **live streaming** or **real-time call**.
For **real-time calls**, set the parameter to `TRTCAppSceneVideoCall` (video call) or `TRTCAppSceneAudioCall` (audio call). This mode is suitable for one-to-one audio/video calls or online meetings for up to 300 attendees.
For **live streaming**, set the parameter to `TRTCAppSceneLIVE` (video live streaming) or `TRTCAppSceneVoiceChatRoom` (audio live streaming). This mode is suitable for live streaming to up to 100,000 users. Make sure you specify the **role** field (valid values: **anchor**, **audience**) in `TRTCParams` if you use this mode.

Parameter 2: TRTCParams

`TRTCParams` consists of many fields; however, you usually only need to pay attention to how to set the following fields:

Parameter	Description	Remarks	Data Type	Sample Value
SDKAppID	Application ID	You can view your application ID in the TRTC console. If you don't have an application yet, click **Create application** to create one.	Number	1400000123
userId	User ID	It can contain only letters, digits, underscores, and hyphens. In TRTC, a user cannot use the same user ID to enter the same room on two different devices at the same time.	String	`denny` or `123321`
userSig	The authentication ticket needed to enter a room	`userSig` is calculated based on `userId` and `SDKAppID` . For the calculation method, see UserSig.	String	eJyrVareCeYrSy1SslI
roomId	Room ID	Numeric room ID. If you want to use string-type room IDs, specify **strRoomId**. Do not use `strRoomId` and `roomId` at the same time.	Number	29834

strRoomId	Room ID	String-type room ID. Do not use `strRoomId` and `roomId` at the same time. `"123"` and `123` are considered different rooms by the TRTC backend.	String	"29834"
role	Role	There are two roles: anchor and audience. This field is required only when `TRTCAppScene` is set to the `TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom` live streaming scenarios.	Enumeration	`TRTCRoleAnchor` `TRTCRoleAudienc`

Note

In TRTC, a user cannot use the same `userId` to enter the same room on two different devices at the same time; otherwise, there will be a conflict.

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

Step 5. Enter the room (`enterRoom`)

After preparing `TRTCAppScene` and `TRTCParams` as described in step 4, you can call the `enterRoom` API to enter the room.

```
import { TRTCParams, TRTCRoleType, TRTCAppScene } from 'trtc-electron-sdk';

const param = new TRTCParams();
param.sdkAppId = 1400000123;
param.userId = "denny";
param.roomId = 123321;
param.userSig = "xxx";
param.role = TRTCRoleType.TRTCRoleAnchor;

// If your scenario is live streaming, set the application scenario to `TRTC_APP_SC
rtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneLIVE);
```

Callbacks:

If room entry succeeds, the SDK will call back the `onEnterRoom(result)` event, and the value of `result` will be greater than 0, indicating the time in milliseconds taken to enter the room.

If room entry fails, the SDK will also call back the `onEnterRoom(result)` event, but the value of `result` will be a negative number, indicating the error code for the room entry failure.

```
function onEnterRoom(result) {
  // For details about `onEnterRoom`, see https://web.sdk.qcloud.com/trtc/electron/
  if (result > 0) {
    console.log('Enter room succeed');
```

```
  } else {
    // For room entry error codes, see https://intl.cloud.tencent.com/document/prod
    console.log('Enter room failed');
  }

}

rtcCloud.on('onEnterRoom', onEnterRoom);
```

Flutter

Last updated：2024-02-02 18:42:44

This document describes how to enter a TRTC room. Only after entering an audio/video room can a user subscribe to the audio/video streams of other users in the room or publish his or her own audio/video streams.

Call Guidelines

Step 1: Import SDK and Configure App Permissions

Please refer to the document Import SDK into the project to complete the SDK import.

Step 2: Create an SDK instance and set an event listener

Invoke various platform initialization interfaces to create the object instance of TRTC.

```
// Create a TRTCCloud singleton
trtcCloud = (await TRTCCloud.sharedInstance())!;
// Register TRTC event callback
trtcCloud.registerListener(onRtcListener);
```

Step 3. Listen for SDK events

You can use callback APIs to listen for errors, warnings, traffic statistics, network quality, as well as various audio/video events of the SDK.

```
// We need to define a method to handle event callbacks, processing appropriately b
// Taking onError as an example
trtcCloud.registerListener(onRtcListener);

onRtcListener(type, param) async {
  if (type == TRTCCloudListener.onError) {
    if (param['errCode'] == -1308) {
      MeetingTool.toast('Failed to initiate screen recording', context);
    } else {
      showErrordDialog(param['errMsg']);
    }
```

```
    }
}
```

Step 4: Prepare the TRTCParams for entering the room

When calling the enterRoom interface, two key parameters need to be filled, namely `TRTCParams` and the application scene. A detailed introduction is as follows:

Parameter one: scene

This parameter refers to the specific application scene, whether it be **video calls, interactive video broadcasting, audio calls**, or **interactive audio broadcasting**:

Scenario Type	Scenario Introduction
TRTC_APP_SCENE_VIDEOCALL	Within the context of video calling, 720p and 1080p high-definition image quality is supported. A single room can accommodate up to 300 simultaneous online users, with a maximum of 50 users speaking at the same time.
TRTC_APP_SCENE_LIVE	In the context of interactive video broadcasting, the mic can be smoothly turned on/off without switching latency, with host latency as low as 300 milliseconds. Supports live streaming for hundreds of thousands of concurrent viewers, with playback delay reduced to 1000 milliseconds. **Note**: In this scenario, you need to specify the current user's role using the 'role' field in TRTCParams.
TRTC_APP_SCENE_AUDIOCALL	In the audio call context, it supports 48 kHz duplex audio calls. A single room accommodates up to 300 concurrent online users, with a maximum of 50 people speaking at once.
TRTC_APP_SCENE_VOICE_CHATROOM	In the context of interactive audio live streaming, microphones can be switched on and off smoothly without delay. The host experiences a low latency of fewer than 300 milliseconds. It accommodates hundreds of thousands concurrent viewer users, with the broadcast delay reduced to 1000 milliseconds. **Note**: In this scenario, you need to specify the current user's role using the 'role' field in TRTCParams.

Parameter 2: TRTCParams

TRTCParams is composed of numerous parameters, but typically, your attention could be principally directed towards filling out the following parameters:

Parameter name	Field Description	Supplementary Information	Data Type	Example
sdkAppId	Application ID	You can locate the SDKAppID within the Tencent Real-Time Communication console, if not present, click on the "Create Application" button to institute a new application.	Number	1400000123
userId	User ID	It can contain only letters, digits, underscores, and hyphens. In TRTC, a user cannot use the same user ID to enter the same room on two different devices at the same time.	String	"denny" or "123321"
userSig	The authentication ticket needed to enter a room	You can calculate userSig using the SDKAppID and userId. Please refer to Calculating and Using UserSig for the calculation method.	String	eJyrVareCeYrSy1SslI
roomId	Room ID	Numeric type 'Room ID'. Be aware, if you wish to utilize a character sequence as the Room ID, please resort to the **strRoomId** field, rather than the roomId field, as strRoomId and roomId should not be used interchangeably.	Number	29834
strRoomId	Room ID	Room ID of string type. Note that `strRoomId` and `roomId` shouldn't be used interchangeably, as to the TRTC backend service, "123" and 123 are not the same room.	String	"29834"
role	Roles	Divided into "Anchor" and "Audience" roles, this field only needs to be	Enumeration	TRTCRoleAnchor or TRTCRoleAudience

		specified when the TRTCAppScene is designated as `TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom` , these two live streaming scenarios.		

Note:

TRTC does not support the simultaneous entry of the same userId on two different devices. Doing so could lead to interference.

Each endpoint in the application scenario, appScene, must be unified to prevent unpredictable issues from cropping up.

Step 5: Enter the room (enterRoom)

After preparing the two parameters from Step 4 (application scenario and TRTCParams), you can call the enterRoom function to enter the room.

```
enterRoom() async {
  try {
    userInfo['userSig'] =
        await GenerateTestUserSig.genTestSig(userInfo['userId']);
    meetModel.setUserInfo(userInfo);
  } catch (err) {
    userInfo['userSig'] = '';
    print(err);
  }
  // If your scenario is "interactive video live broadcast", please set the scene t
  await trtcCloud.enterRoom(
```



```
        TRTCParams(
            sdkAppId: GenerateTestUserSig.sdkAppId,
            userId: userInfo['userId'],
            userSig: userInfo['userSig'],
            role: TRTCCloudDef.TRTCRoleAnchor,
            roomId: meetId!),
        TRTCCloudDef.TRTC_APP_SCENE_LIVE);
  }
```

Event Callback

If room entry is successful, SDK will return the onEnterRoom(result) event where result is a positive number, indicating the time consumed to join the room in milliseconds (ms).

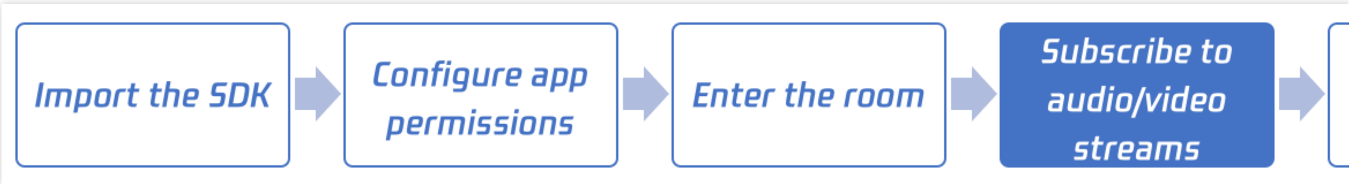
If room entry fails, SDK will also call back the onEnterRoom(result) event, but the parameter `result` will be a negative number, representing the error code of room entry failure.

```
//Listen for SDK's onEnterRoom event to check if room entry is successful or not
onRtcListener(type, param) async {
  if (type == TRTCCloudListener.onEnterRoom) {
    if (param > 0) {
      MeetingTool.toast('Enter room success', context);
    }
  }
}
```

03. Subscribing to Audio/Video Streams Android, iOS, Windows, and macOS

Last updated：2024-05-21 15:05:29

This document describes how to subscribe to the audio/video streams of another user in the room, i.e., how to play back the audio/video of another user. For the sake of convenience, "another user in the room" is called a "remote user" in this document.



Call Guide

Step 1. Perform prerequisite steps

Import the SDK and configure the application permissions as instructed in iOS.

Step 2. Set the subscription mode (optional)

You can call the **setDefaultStreamRecvMode** API in `TRTCCloud` to set the subscription mode. TRTC provides two subscription modes:

Automatic subscription: The SDK automatically plays back remote users' audio without additional manual operations required. This is the default subscription mode.

Manual subscription: The SDK doesn't automatically pull or play back remote users' audio. You need to call **muteRemoteAudio(userId, false)** to play back the audio.

Note:

If you do not call `setDefaultStreamRecvMode` , the automatic subscription mode will be used. However, if you want to use the manual subscription mode, note that `setDefaultStreamRecvMode` can take effect only if it is called before `enterRoom` .

Step 3. Enter a TRTC room

Make the current user enter a TRTC room as instructed in Entering a Room. A user can subscribe to the audio/video streams of a remote user only after a successful room entry.

Step 4. Play back an audio stream

You can call `muteRemoteAudio("denny", true)` to mute the remote user `denny` and then call `muteRemoteAudio("denny", false)` to unmute `denny`.

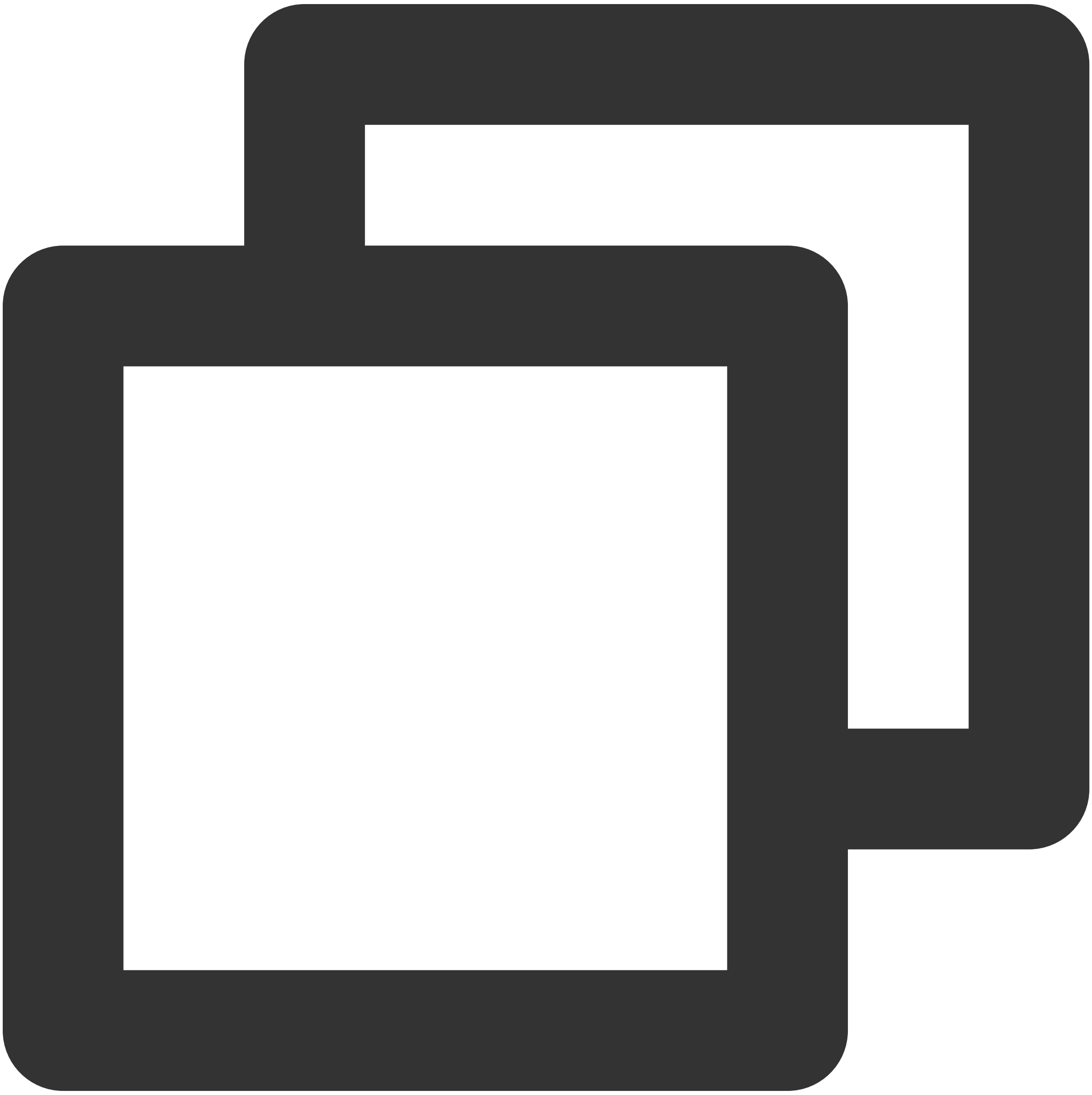
Android

iOS&Mac

Windows

```
// Mute the user with ID denny
mCloud.muteRemoteAudio("denny", true);
// Unmute the user with ID denny
mCloud.muteRemoteAudio("denny", false);
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
// Mute the user with ID denny
[self.trtcCloud muteRemoteAudio:@"denny" mute:YES];
// Unmute the user with ID denny
[self.trtcCloud muteRemoteAudio:@"denny" mute:YES];
```

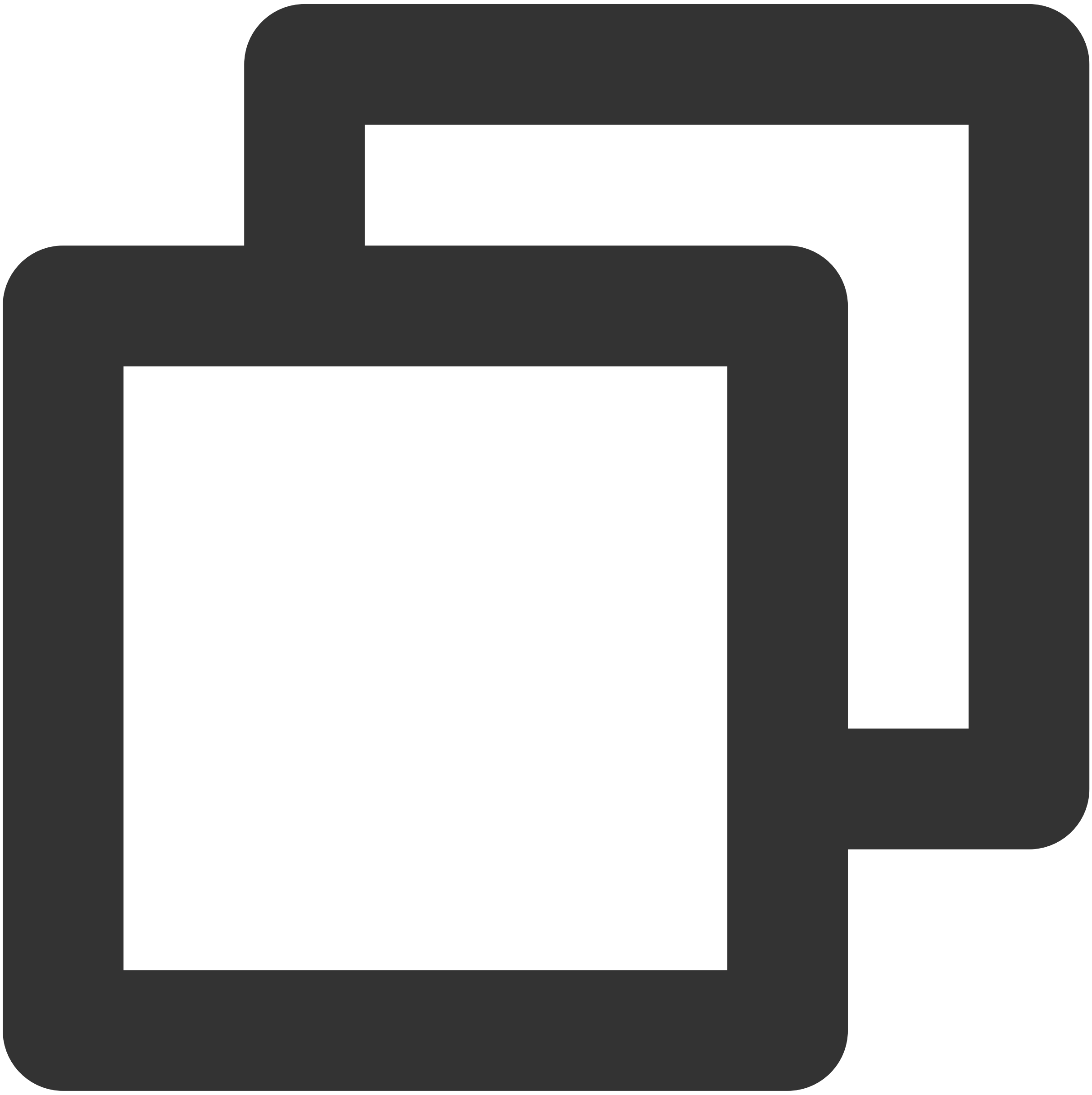


```
// Mute the user with ID denny
trtc_cloud_->muteRemoteAudio("denny", true);
// Unmute the user with ID denny
trtc_cloud_->muteRemoteAudio("denny", false);
```

Step 5. Play back a video stream

1. Start and stop playback (startRemoteView + stopRemoteView)

You can call `startRemoteView` to play back the video of a remote user, but only after you pass in a view object to the SDK as the rendering control that carries the user's video image.
The first parameter of `startRemoteView` is `userId` of the remote user, the second is the stream type of the user, and the third is the view object to be passed in. The second parameter `streamType` has three valid values:
TRTCVideoStreamTypeBig: The primary stream, which is generally used to display the user's camera image.
TRTCVideoStreamTypeSub: The substream, which is generally used to display the user's screen sharing image.
TRTCVideoStreamTypeSmall: A lower quality video of the user's primary stream. You can play back the lower quality video of a remote user only after the user enables dual-stream mode (`enableEncSmallVideoStream`). The high quality stream and low quality stream cannot be played back at the same time.
You can call the `stopRemoteView` API to stop playing back the video of one remote user or call the `stopAllRemoteView` API to stop playing back videos of all remote users.
Android
iOS&Mac
Windows



```
// Play back the camera (primary stream) image of `denny`
mCloud.startRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, cameraView
// Play back the screen sharing (substream) image of `denny`
mCloud.startRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SUB, screenView
// Play back the lower quality video image of `denny` (The high quality stream and
mCloud.startRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SMALL, cameraVi
// Stop playing back the camera image of `denny`
mCloud.stopRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, cameraView)
// Stop playing back the video images of all remote users
mCloud.stopAllRemoteView();
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
// Play back the camera (primary stream) image of `denny`
[self.trtcCloud startRemoteView:@"denny" streamType:TRTCVideoStreamTypeBig view:cam
// Play back the screen sharing (substream) image of `denny`
[self.trtcCloud startRemoteView:@"denny" streamType:TRTCVideoStreamTypeSub view:scr
// Play back the lower quality video image of `denny` (The high quality stream and
[self.trtcCloud startRemoteView:@"denny" streamType:TRTCVideoStreamTypeSmall view:c
// Stop playing back the camera image of `denny`
[self.trtcCloud stopRemoteView:@"denny" streamType:TRTCVideoStreamTypeBig view:came
// Stop playing back the video images of all remote users
```

```
[self.trtcCloud stopAllRemoteView];
```

```
// Play back the camera (primary stream) image of `denny`
trtc_cloud_->startRemoteView("denny", liteav::TRTCVideoStreamTypeBig, (liteav::TXVi
// Play back the screen sharing (substream) image of `denny`
trtc_cloud_->startRemoteView("denny", liteav::TRTCVideoStreamTypeSub, (liteav::TXVi
// Play back the lower quality video image of `denny` (The high quality stream and
trtc_cloud_->startRemoteView("denny", liteav::TRTCVideoStreamTypeSmall, (liteav::TX
// Stop playing back the camera image of `denny`
trtc_cloud_->stopRemoteView("denny", liteav::TRTCVideoStreamTypeBig);
```

```
// Stop playing back the video images of all remote users
trtc_cloud_->stopAllRemoteView();
```

2. Set playback parameters (updateRemoteView and setRemoteRenderParams)

You can call the `updateRemoteView` API to change the view object during playback. This is useful for switching the video rendering control.

You can use `setRemoteRenderParams` to set the video image fill mode, rotation angle, and mirror mode.

Fill mode: You can use the fill mode or fit mode. In both modes, the original image aspect ratio is not changed. The difference is whether black bars are displayed.

Rotation angle: You can set the rotation angle to 0, 90, 180, or 270 degrees.

Mirror mode: Indicates whether to flip the image horizontally.

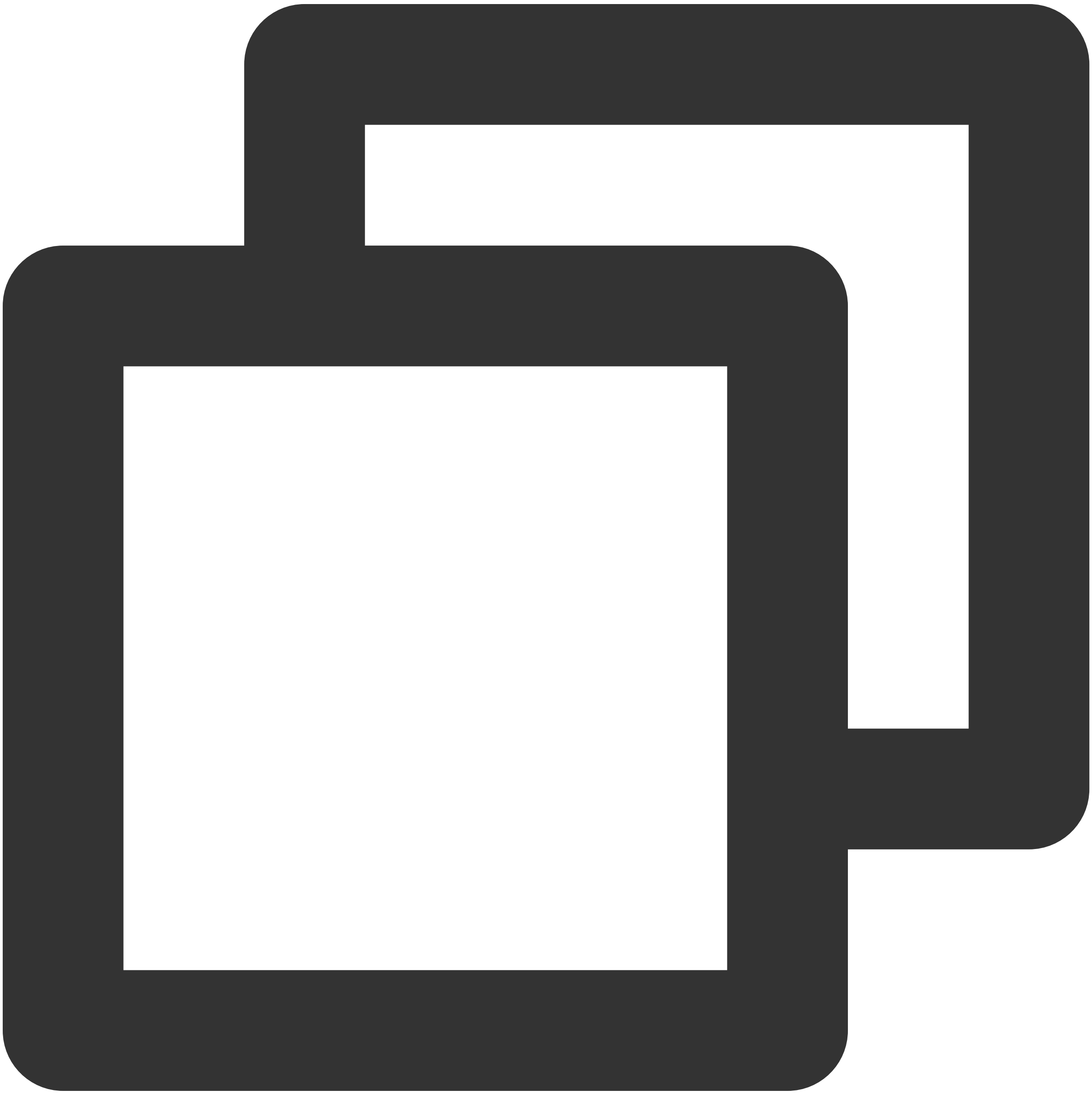
Android

iOS&Mac

Windows

```
// Switch the primary stream image of `denny` to a small floating window (`miniFloa
mCloud.updateRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, miniFloat

// Set the fill mode of the primary stream image of the remote user `denny` to fill
TRTCCloudDef.TRTCRenderParams param = new TRTCCloudDef.TRTCRenderParams();
param.fillMode    = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL;
param.mirrorType    = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_DISABLE;
mCloud.setRemoteRenderParams("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, para
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
// Switch the primary stream image of `denny` to a small floating window (`miniFloa
[self.trtcCloud updateRemoteView:miniFloatingView streamType:TRTCVideoStreamTypeBig
// Set the fill mode of the primary stream image of the remote user `denny` to fill
TRTCRenderParams *param = [[TRTCRenderParams alloc] init];
param.fillMode      = TRTCVideoFillMode_Fill;
param.mirrorType    = TRTCVideoMirrorTypeDisable;
[self.trtcCloud setRemoteRenderParams:@"denny" streamType:TRTCVideoStreamTypeBig pa
```



```
// Switch the primary stream image of `denny` to another window (suppose the handle
trtc_cloud_->updateRemoteView("denny", liteav::TRTCVideoStreamTypeBig, (liteav::TXV

// Set the fill mode for the primary stream image of the remote user `denny` to fil
liteav::TRTCRenderParams param;
param.fillMode = TRTCVideoFillMode_Fill;
param.mirrorType = TRTCVideoMirrorType_Enable;
trtc_cloud_->setRemoteRenderParams("denny", TRTCVideoStreamTypeBig, param);
```

Step 6. Get the audio/video status of a remote user in the room

In steps 4 and 5, you can control the audio/video playback of remote users. However, if there isn't sufficient information, you won't be able to know:

What users are in the current room

The camera and mic status of users in the room

To solve this problem, you need to listen for the following event callbacks from the SDK:

Audio status change notification (onUserAudioAvailable)

Listen for `onUserAudioAvailable(userId,boolean)` to be notified when a remote user turns their mic on/off.

Video status change notification (onUserVideoAvailable)

Listen for `onUserVideoAvailable(userId,boolean) to be notified when a remote user turns their camera on/off.`

`Listen for` onUserSubStreamAvailable(userId,boolean)` to be notified when a remote user enables/disables screen sharing.

User room entry/exit notification (onRemoteUserEnter/LeaveRoom)

When a remote user enters the current room, you can use `onRemoteUserEnterRoom(userId)` to get `userId` of the user. When a remote user exits the current room, you can use `onRemoteUserLeaveRoom(userId, reason)` to get `userId` of the user and the reason for the exit.

Note:

More accurately, `onRemoteUserEnter/LeaveRoom` is triggered only when an anchor enters/leaves the room. This prevents the problem of receiving frequent notifications of room entries/exits when there are a high number of audience members in the room.

With the event callbacks above, you can know the users in the room and whether they have turned on their cameras and mics. In the sample code below, `mCameraUserList` , `mMicrophoneUserList` , and `mUserList` are used to maintain the following information respectively:

What users (anchors) are in the room

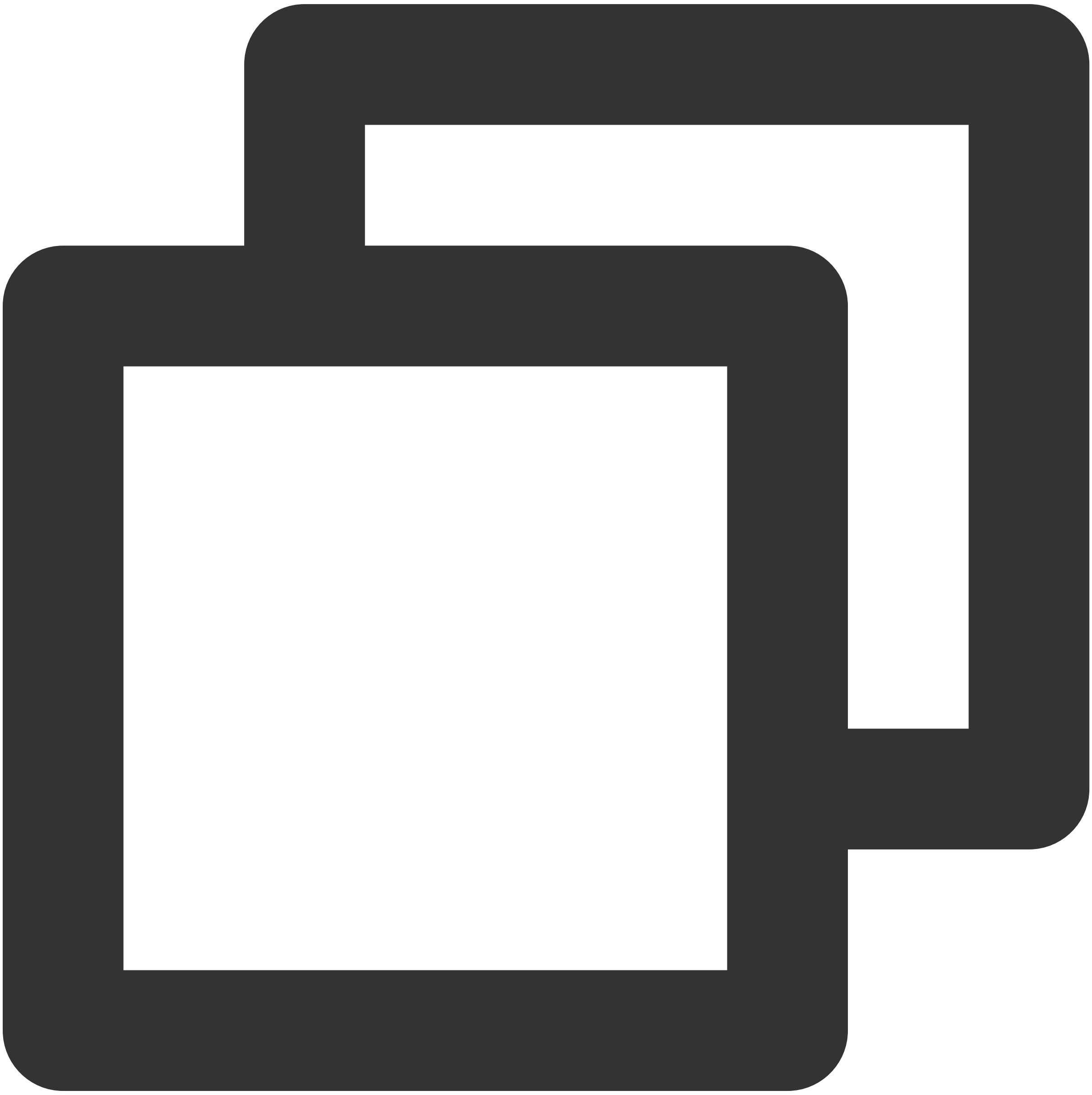
Which users have turned on their cameras

Which users have turned on their mics

Android

iOS&Mac

Windows



```
// Get the change of the video status of a remote user and update the list of users
@Override
public void onUserVideoAvailable(String userId, boolean available) {
    available?mCameraUserList.add(userId) : mCameraUserList.remove(userId);
}

// Get the change of the audio status of a remote user and update the list of users
@Override
public void onUserAudioAvailable(String userId, boolean available) {
    available?mMicrophoneUserList.add(userId) : mMicrophoneUserList.remove(userId);
}
```

```
// Get the room entry notification of a remote user and update the remote user list
@Override
public void onRemoteUserEnterRoom(String userId) {
    mUserList.add(userId);
}

// Get the room exit notification of a remote user and update the remote user list
@Override
public void onRemoteUserLeaveRoom(String userId, int reason) {
    mUserList.remove(userId);
}
```

```
// Get the change of the video status of a remote user and update the list of users
- (void)onUserVideoAvailable:(NSString *)userId available:(BOOL)available{
    if (available) {
        [mCameraUserList addObject:userId];
    } else {
        [mCameraUserList removeObject:userId];
    }
}

// Get the change of the audio status of a remote user and update the list of users
- (void)onUserAudioAvailable:(NSString *)userId available:(BOOL)available{
```

```
    if (available) {
        [mMicrophoneUserList addObject:userId];
    } else {
        [mMicrophoneUserList removeObject:userId];
    }
}

// Get the room entry notification of a remote user and update the remote user list
- (void)onRemoteUserEnterRoom:(NSString *)userId{
    [mUserList addObject:userId];
}

// Get the room exit notification of a remote user and update the remote user list
- (void)onRemoteUserLeaveRoom:(NSString *)userId reason:(NSInteger)reason{
    [mUserList removeObject:userId];
}
```

}

```
// Get the change of the video status of a remote user and update the list of users
void onUserVideoAvailable(const char* user_id, bool available) {
    available ? mCameraUserList.push_back(user_id) : mCameraUserList.remove(user_id
}

// Get the change of the audio status of a remote user and update the list of users
void onUserAudioAvailable(const char* user_id, bool available) {
    available ? mMicrophoneUserList.push_back(user_id) : mMicrophoneUserList.remove
}

// Get the room entry notification of a remote user and update the remote user list
```

```
void onRemoteUserEnterRoom(const char* user_id) {
    mUserList.push_back(user_id);
}

// Get the room exit notification of a remote user and update the remote user list
void onRemoteUserLeaveRoom(const char* user_id, int reason) {
    mUserList.remove(user_id);
}
```

Advanced Guide

1. What are the differences between different muting methods?

There are three muting methods which work in completely different ways:

Method 1. The player stops subscribing to the audio stream

To stop playing back the audio of the remote user `denny` , you can call `muteRemoteAudio("denny", true)` , and the SDK will stop pulling audio data from `denny` . In this mode, less traffic is used. However, because the SDK needs to restart the audio data pull process to resume audio playback, switching from the muted to unmuted status is slow.

Method 2. Adjust the playback volume level to zero

If you need to switch between the muted and unmuted status more quickly, you can use `setRemoteAudioVolume("denny", 0)` , which sets the playback volume level of the remote user `denny` to zero. Because this API doesn't involve network operations, it takes effect very quickly.

Method 3. The remote user turns off the mic

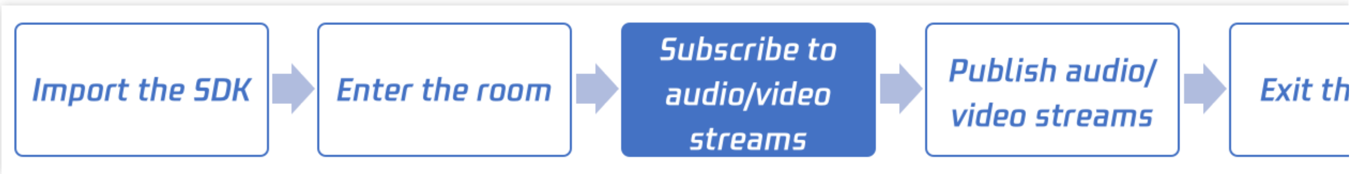
All operations described in this document are performed in the player and take effect only for the local user. For example, if you call `muteRemoteAudio("denny", true)` to mute the remote user `denny` , other users in the room can still hear `denny` .

To completely disable the audio of `denny` , you need to change the way their audio is published. For more information, see Publishing Audio/Video Streams.

Web

Last updated：2023-09-26 16:59:51

This document describes how to subscribe to the audio/video streams of another user (remote user) in the room, i.e., how to play the audio/video of a remote user.



Step 1. Enter Room

For detailed directions, see Enter Room.

Step 2. Play Remote Audio and Video

Play Remote Audio

By default, the SDK will automatically play remote audio, and you do not need to call any API to play remote audio.

If you do not want the SDK to automatically play remote audio, you can

1. Set autoReceiveAudio = false to turn off automatic audio playback by calling `trtc.enterRoom({ autoReceiveAudio: false })`.
2. Listen for the `TRTC.EVENT.REMOTE_AUDIO_AVAILABLE` event before entering the room.
3. Save the userId of remote user when this event fired.
4. Call `trtc.muteRemoteAudio(userId, false)` method when you need to play remote audio.

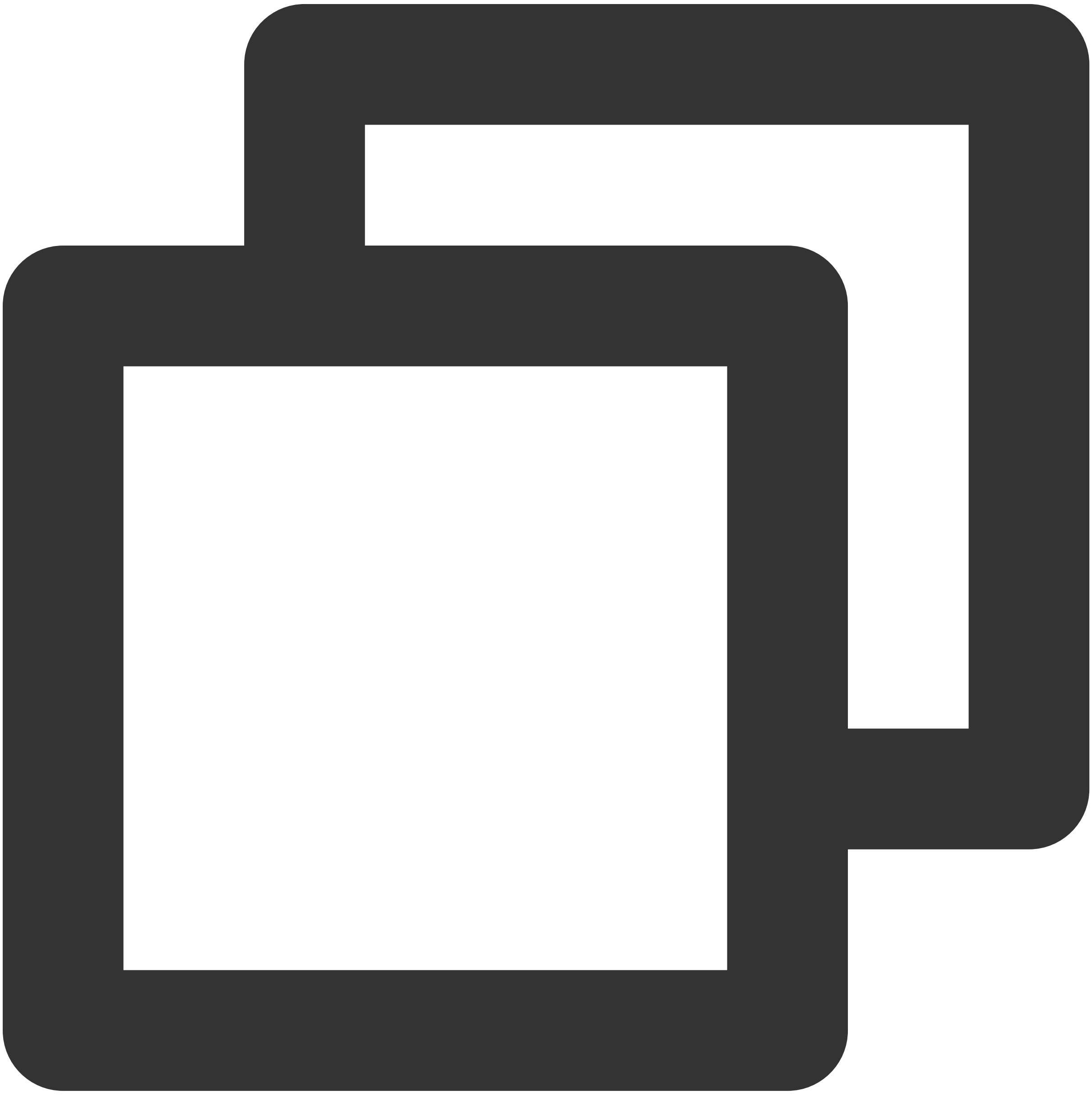
Note：

If the user has not interacted with the page before entering the room, automatic audio playback may fail due to Browser's Autoplay Policy. You need to refer to the Handle Autoplay Restriction for processing.

Play Remote Video

1. Listen for the `TRTC.EVENT.REMOTE_VIDEO_AVAILABLE` event before entering the room to receive all remote user video publishing events.

2. Use the `trtc.startRemoteVideo()` method to play the remote video stream when you receive the event.

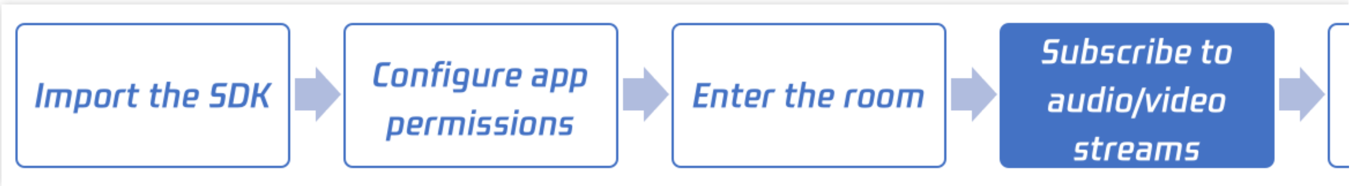


```
trtc.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, ({ userId, streamType }) => {
  // To play the video image, you need to place an HTMLElement in the DOM, which ca
  const view = `${userId}_${streamType}`;
  trtc.startRemoteVideo({ userId, streamType, view });
});
```

Electron

Last updated：2024-05-21 15:05:29

This document describes how to subscribe to the audio/video streams of another user (remote user) in the room, i.e., how to play a remote user's audio/video.



Call Guide

Step 1. Perform prerequisite steps

Import the SDK as instructed in Electron.

Step 2. Set the subscription mode (optional)

You can call the **setDefaultStreamRecvMode** API in `TRTCCloud` to set the subscription mode. TRTC provides two subscription modes:

Automatic subscription: The SDK automatically plays remote users' audios. This is the default subscription mode.

Manual subscription: The SDK doesn't automatically pull or play remote users' audios. You need to call **muteRemoteAudio(userId, false)** manually to play the audio of a remote user.

Note:

If you don't call `setDefaultStreamRecvMode` , the automatic subscription mode will apply. If you want to use the manual subscription mode, **make sure you call** `setDefaultStreamRecvMode` **before** `enterRoom` .

Step 3. Enter a TRTC room

Make the current user enter a TRTC room as instructed in Entering a Room. A user can subscribe to the audio/video streams of a remote user only after a successful room entry.

Step 4. Play the audio of a remote user

You can call `muteRemoteAudio("denny", true)` to mute the remote user `denny` and then call `muteRemoteAudio("denny", false)` to unmute `denny` .

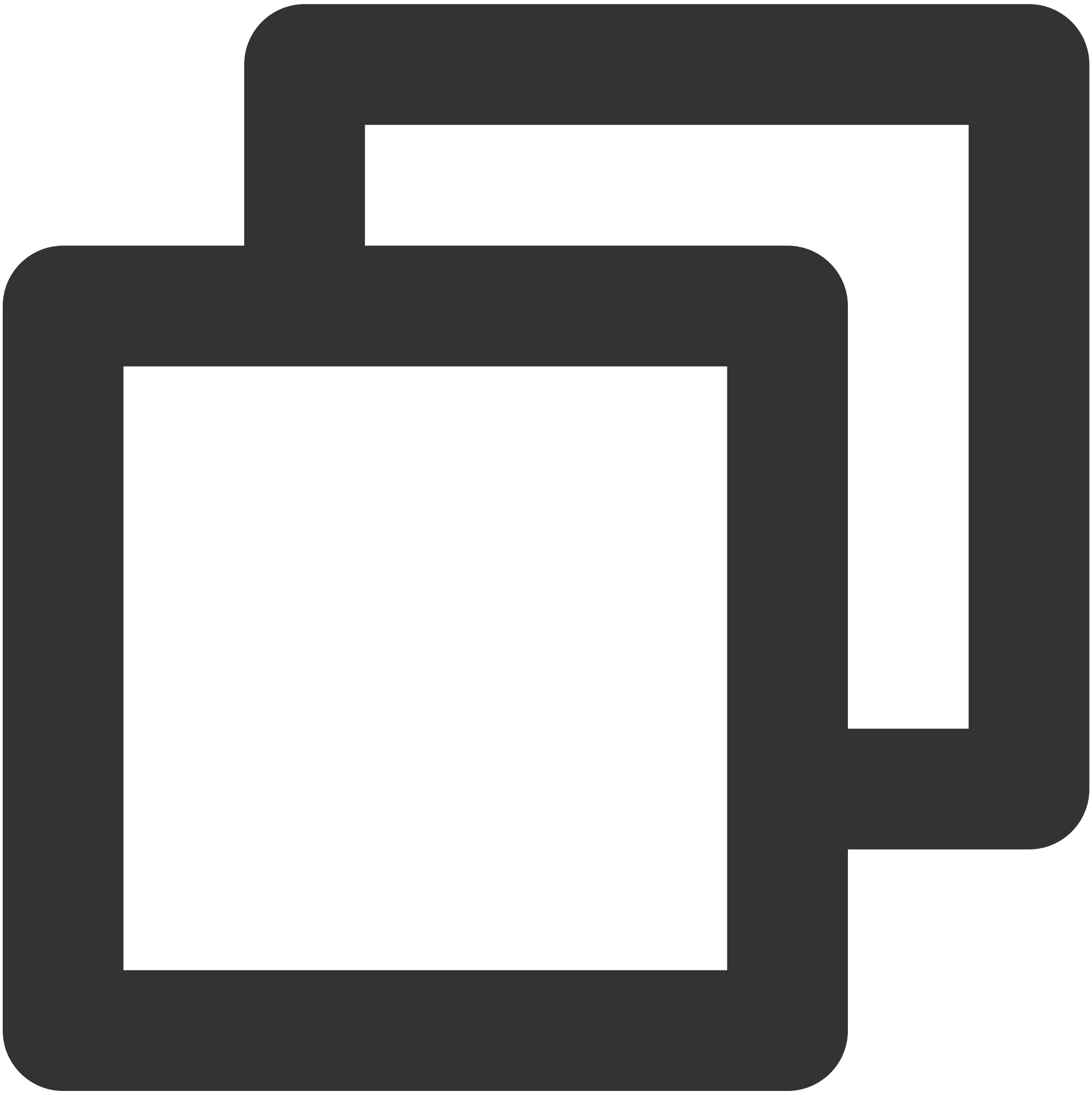

```
import TRTCCloud from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();

// For details, see https://web.sdk.qcloud.com/trtc/electron/doc/zh-cn/trtc_electro
// Mute the user “denny”
rtcCloud.muteRemoteAudio('denny', true);
// Unmute the user with ID denny
rtcCloud.muteRemoteAudio('denny', false);
```

Step 5. Play the video of a remote user

1. Start and stop playback (startRemoteView + stopRemoteView)

You can call `startRemoteView` to play the video of a remote user, but only after you pass in a view object to the SDK as the rendering control for the user's video.
The first parameter of `startRemoteView` is `userId` of the remote user, the second is the stream type of the user, and the third is the view object to be passed in. Here, the second parameter `streamType` has three valid values:
TRTCVideoStreamTypeBig: The primary stream, which is usually a user's camera video.
TRTCVideoStreamTypeSub: The substream, which is usually a user's screen sharing image.
TRTCVideoStreamTypeSmall: A lower quality video of a user's primary stream. You can play the lower quality video of a remote user only after the user enables dual-stream mode (`enableEncSmallVideoStream`). The original and lower quality streams cannot be played at the same time.
You can call the `stopRemoteView` API to stop playing back the video of one remote user or call the `stopAllRemoteView` API to stop playing back videos of all remote users.



```
// For details, see https://web.sdk.qcloud.com/trtc/electron/doc/zh-cn/trtc_electro

import TRTCCloud, { TRTCVideoStreamType } from 'trtc-electron-sdk';

const cameraView = document.querySelector('.user-dom');
const screenView = document.querySelector('.screen-dom');
const rtcCloud = new TRTCCloud();
// Play back the camera (primary stream) image of `denny`
rtcCloud.startRemoteView('denny', cameraView, TRTCVideoStreamType.TRTCVideoStreamTy
// Play back the screen sharing (substream) image of `denny`
rtcCloud.startRemoteView('denny', screenView, TRTCVideoStreamType.TRTCVideoStreamTy
```



```
// Play back the lower quality video of `denny` (The original and lower quality str
rtcCloud.startRemoteView('denny', cameraView, TRTCVideoStreamType.TRTCVideoStreamTy
// Stop playing back the camera image of `denny`
rtcCloud.stopRemoteView('denny', TRTCVideoStreamType.TRTCVideoStreamTypeBig);
// Stop playing back the videos of all remote users
rtcCloud.stopAllRemoteView();
```

2. Set playback parameters (`setRemoteRenderParams`)

You can use `setRemoteRenderParams` to set the video image fill mode, rotation angle, and mirror mode.

Fill mode: You can use the fill mode or fit mode. In both modes, the original image aspect ratio is not changed. The difference is whether black bars are displayed.

Rotation angle: You can set the rotation angle to 0, 90, 180, or 270 degrees.

Mirror mode: Indicates whether to flip the image horizontally.


```
// For details, see https://web.sdk.qcloud.com/trtc/electron/doc/zh-cn/trtc_electro
// Set the fill mode for the primary stream of the remote user `denny` to fill, and
import TRTCCloud, {
  TRTCRenderParams, TRTCVideoStreamType, TRTCVideoRotation,
  TRTCVideoFillMode, TRTCVideoMirrorType
} from 'trtc-electron-sdk';

const param = new TTRTCRenderParams(
  TRTCVideoRotation.TRTCVideoRotation0,
  TRTCVideoFillMode.TRTCVideoFillMode_Fill,
  TRTCVideoMirrorType.TRTCVideoMirrorType_Enable
```



```
);
const rtcCloud = new TRTCCloud();
rtcCloud.setRemoteRenderParams('denny', TRTCVideoStreamType.TRTCVideoStreamTypeBig,
```

Step 6. Get the audio/video status of a remote user in the room

Step 4 and step 5 showed how to use APIs to control the playback of the audio and video of a remote user. However, without the following information, you may not know what user ID to pass in when calling the APIs.

What users are in the current room

The camera and mic status of users in the room

To solve this problem, you need to listen for the following event callbacks from the SDK:

Audio status change notification (onUserAudioAvailable)

You can listen for `onUserAudioAvailable(userId,boolean)` to be notified when a remote user turns their mic on/off.

Video status change notification (onUserVideoAvailable)

You can listen for `onUserVideoAvailable(userId,boolean)` to be notified when a remote user turns their camera on/off.

You can listen for `onUserSubStreamAvailable(userId,boolean)` to be notified when a remote user enables/disables screen sharing.

User room entry/exit notification (onRemoteUserEnter/LeaveRoom)

When a remote user enters the current room, you can get the user's ID from `onRemoteUserEnterRoom(userId)` . When a remote user exits the current room, you can get the user's ID and the reason for their exit from `onRemoteUserLeaveRoom(userId, reason)` .

Note:

More accurately, `onRemoteUserEnter/LeaveRoom` only notifies you about the room entry/exit of anchors. This prevents frequent notifications when there are a large audience in the room.

With the event callbacks above, you can know the users in the room and whether they have turned on their cameras and mics. In the sample code below, `mCameraUserList` , `mMicrophoneUserList` , and `mUserList` are used to maintain the following information:

What users (anchors) are in the room

Which users have turned on their cameras

Which users have turned on their mics

```
import TRTCCloud from 'trtc-electron-sdk';
let openCameraUserList = [];
let openMicUserList = [];
let roomUserList = [];

function onUserVideoAvailable(userId, available) {
  if (available === 1) {
    openCameraUserList.push(userId);
  } else {
    openCameraUserList = openCameraUserList.filter((item) => item !== userId);
  }
```

```
}

function onUserAudioAvailable(userId, available) {
  if (available === 1) {
    openMicUserList.push(userId);
  } else {
    openMicUserList = openMicUserList.filter((item) => item !== userId);
  }
}

function onRemoteUserEnterRoom(userId) {
  roomUserList.push(userId);
}

function onRemoteUserLeaveRoom(userId, reason) {
  roomUserList = roomUserList.filter((item) => item !== userId);
}

const rtcCloud = new TRTCCloud();
rtcCloud.on('onUserVideoAvailable', onUserVideoAvailable);
rtcCloud.on('onUserAudioAvailable', onUserAudioAvailable);
rtcCloud.on('onRemoteUserEnterRoom', onRemoteUserEnterRoom);
rtcCloud.on('onRemoteUserLeaveRoom', onRemoteUserLeaveRoom);
```

Advanced Guide

What are the differences between different muting methods?

There are three muting methods which work in completely different ways:

Method 1. The player stops subscribing to the audio stream

To stop playing back the audio of the remote user `denny` , you can call `muteRemoteAudio("denny", true)` , and the SDK will stop pulling the audio data of `denny` . In this mode, less traffic is used. However, it will be slow to resume audio playback because the SDK needs to restart the audio data pull process.

Method 2. Adjust the playback volume level to zero

If you want to mute a user more quickly, you can call `setRemoteAudioVolume("denny", 0)` , which sets the playback volume of the remote user `denny` to zero. Because this API doesn't involve network operations, it takes effect very quickly.

Method 3. The remote user turns off the mic

All operations described in this document are performed in the player and take effect only for the local user. For example, if you call `muteRemoteAudio("denny", true)` to mute the remote user `denny` , other users in the room can still hear `denny` .

To completely disable the audio of `denny` , you need to change the way their audio is published. For more information, see Publishing Audio/Video Streams.

Flutter

Last updated：2024-02-02 18:51:23

This document primarily elucidates the process of subscribing to the audio and video streams of other users in a room, essentially illustrating how to play others' audio and video. For the sake of convenience, throughout the remainder of this document, we will collectively refer to 'other users in the room' as 'remote users'.

Call Guidelines

Step 1. Perform prerequisite steps

Refer to the document Import SDK into the project to accomplish the import of SDK and for the configuration of App permissions.

Step 2. Set the subscription mode (optional)

You can set the subscription mode by calling **setDefaultStreamRecvMode** interface in TRTCCloud. TRTC provides two subscription modes:
Automatic subscription: The SDK automatically plays remote users' audios. This is the default subscription mode.
Manual subscription: The SDK does not automatically pull and play the audio of remote users. You need to manually invoke **muteRemoteAudio(userId, false)** to trigger the playback of the audio.

Note:

It is important to note that it's perfectly fine if you don't invoke setDefaultStreamRecvMode; by default, the SDK is set to automatic subscription. However, if manual subscription is preferred, ensure that setDefaultStreamRecvMode is invoked before entering a room as it only takes effect when done so.

Step 3. Enter a TRTC room

Follow the instructions in Entering a Room to allow a user to enter a TRTC room. The user can only subscribe to the audio or video streams of other users after successfully entering the room.

Step 4. Play the audio of a remote user

You have the option to mute or unmute a specified user by invoking the muteRemoteAudio interface.

```
// Mute user with id denny
trtcCloud.muteRemoteAudio('denny', true);
// Unmute user with id denny
trtcCloud.muteRemoteAudio('denny', false);
```

Step 5. Play the video of a remote user

1. Initiate and Terminate Playback (startRemoteView + stopRemoteView)

You are able to initiate the projection of the video image of remote users by invoking the startRemoteView API. However, the precondition is that you need to pass a view object to the SDK to serve as the rendering control for the user's video image.

The first parameter of startRemoteView is the userId of the remote user, the second is the stream type of the remote user, the third is the view object that you need to pass. Among them, the second parameter streamType has three optional values, which are:

`TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG` : This signifies the primary stream of the user, typically used to transmit the image from the user's camera.

`TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SUB` : This represents the auxiliary stream of the user, generally used to transmit the screen sharing image of the user.

`TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SMALL` : Pertaining to the user's low-resolution thumbnail image, this nomenclature is relative to the primary route image. It is only after the remote user has tacitly enabled "dual-channel encoding (enableEncSmallVideoStream)", we can then render their low-resolution image. Notwithstanding, a choice must be made between the primary route image and the low-resolution thumbnail - only one can be utilized simultaneously.

By calling the stopRemoteView interface, you can stop playing the video of a specific remote user or halt all video playbacks from remote users through the stopAllRemoteView interface.

```
// Rendering the video output from Denny's camera (aptly referred to by us as the "
trtcCloud.startRemoteView('denny', TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, cameraV
// Displaying the screen that Denny shares (which we term as the "auxiliary route i
trtcCloud.startRemoteView('denny', TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SUB, screenV
// Play back the lower quality video of `denny` (The original and lower quality str
trtcCloud.startRemoteView('denny', TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SMALL, camer
// Stop playing back the camera image of `denny`
trtcCloud.stopRemoteView('denny', TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, cameraVi
// Stop the playback of all video footages
trtcCloud.stopAllRemoteView();
```

2. Set playback parameters (updateRemoteView + setRemoteRenderParams)

By invoking the updateRemoteView API, you can modify the view object while playing, which is very useful when switching video rendering controls.

By using setRemoteRenderParams, you can configure the screen's fill mode, rotation angle and mirror settings.

Fill mode: You can use the fill mode or fit mode. In both modes, the original image aspect ratio is not changed. The difference is whether black bars are displayed.

Rotation angle: You can set the rotation angle to 0, 90, 180, or 270 degrees.

Mirror mode: Indicates whether to flip the image horizontally.

```
// Switch the primary view of `denny` to a floating mini-window (assuming the mini-
trtcCloud.updateRemoteView('denny', TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG,miniFlo

// Rotate the primary view of the remote user `denny` 90 degrees clockwise, set it
TRTCRenderParams params = TRTCRenderParams(
                    rotation: TRTCCloudDef.TRTC_VIDEO_ROTATION_90,
                    fillMode: TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL,
                    mirrorType: TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_DISABLE);
trtcCloud.setRemoteRenderParams('denny', TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, p
```

Step 6. Get the audio/video status of a remote user in the room

In Steps 4 and 5, you can control the playback of audio and video from remote users, but without sufficient information, you would not ascertain:

What users are in the current room

Whether they have turned on their cameras and microphones?

To solve this problem, you need to listen for the following event callbacks from the SDK:

Notification of Audio Status Change(onUserAudioAvailable)

You can monitor the status change when the remote user turns the microphone on or off by listening to onUserAudioAvailable(userId,bool).

Notification of Video Status Change(onUserVideoAvailable)

You can monitor the status change when the remote user activates or deactivates the camera feed by listening to onUserVideoAvailable(userId,bool).

You can likewise monitor the status change when the remote user enables or disables screen sharing by listening onUserSubStreamAvailable(userId,bool).

Notification of User Joining and Leaving the Room(onRemoteUserEnter/onRemoteUserLeaveRoom)

When a remote user enters the current room, you can get his/her userId through onRemoteUserEnterRoom(userId).

When a remote user leaves the room, you can understand the userId of this user and his/her reasons for leaving through onRemoteUserLeaveRoom(userId, reason).

Note:

To put it precisely, onRemoteUserEnter/LeaveRoom can only perceive the notification of the entry and exit of users who are cast as anchors. Such design is purposed to prevent the so-called 'signaling storm' attack caused by the frequent entry and exit of people, particularly when the online audience inside a room is quite substantial.

Note:

In Dart, we receive and process callbacks of the TRTC SDK through a method, which is categorized as ListenerValue, the parameter that needs to be inputted in registerListener. ListenerValue has two parameters, representing the callback type 'type' and 'params', the parameters of the callback returned by the SDK.

With these event webhooks, you can grasp which users are in the room as well as whether they have turned on their cameras and microphones. Refer to the example code below. In this piece of exemplary code, we have employed userList, cameraUserList, and microphoneUserList to separately maintain:

What users (anchors) are in the room

Which users have turned on their cameras

Which users have turned on their mics

```
onRtcListener(type, param) async {
  if (type == TRTCCloudListener.onUserAudioAvailable) {
```

```
      String userId = param['userId'];
      bool available = param['available'];
      available ? microphoneUserList.add(userId) : microphoneUserList.remove(userId);
    }
    if (type == TRTCCloudListener.onUserVideoAvailable) {
      String userId = param['userId'];
      bool available = param['available'];
      available ? cameraUserList.add(userId) : cameraUserList.remove(userId);
    }
    if (type == TRTCCloudListener.onRemoteUserEnterRoom) {
      userList.add({
        'userId': param
      });
    }
    if (type == TRTCCloudListener.onRemoteUserLeaveRoom) {
      String userId = param['userId'];
      userList.removeWhere((user) => user['userId'] == userId);
    }
  }
```

Advanced Guide

1. Both are "mute", what is the difference?

As your business requirements continually deepen, you will discover three distinct types of 'silence'. Although they are all termed 'silence', their counting principles are entirely distinct:

First method: The player stops Subscription to the audio stream

If you call the muteRemoteAudio("denny", true) function, it indicates your desire to cease hearing the audio of remote user 'denny'. At this point, the SDK will stop fetching the data stream of denny's audio. This pattern is more bandwidth-efficient. However, should you wish to hear denny's audio again, the SDK needs to initiate the process of fetching the audio data anew. Therefore, the transition from 'mute' to 'unmute' states can be a rather slow process.

The second approach: adjust the playback volume to zero

If your business scenario requires a faster response for mute switching, you can set the playback volume of the remote user 'Denny' to zero using setRemoteAudioVolume("denny", 0). Since this interface does not involve network operations, the effect is exceptionally swift.

The third case: the remote user turns off the microphone themselves

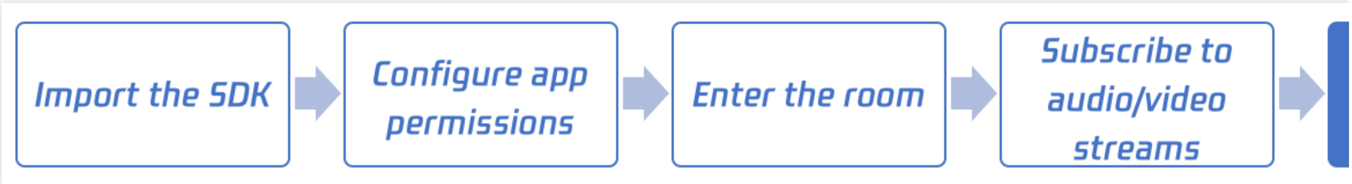
All the operations introduced in this document pertain to the instruction of the playback end and their effects only apply to the current user. For instance, if you mute a remote user 'Denny' through the function muteRemoteAudio("denny", true), the voice of 'Denny' can still be heard by other users in the room.

In order to effectively "silence" Denny, it would be necessary to influence Denny's audio publishing behavior. We will discuss this in detail in our following document titled Relay of audio and video streams.

04. Publishing Audio/Video Streams
Android, iOS, Windows, and macOS

Last updated：2024-05-21 15:05:29

This document describes how an anchor publishes audio/video streams. "Publishing" refers to turning on the mic and camera to make the audio heard and video seen by other users in the room.



Call Guide

Step 1. Perform prerequisite steps

Import the SDK and configure the application permissions as instructed in iOS.

Step 2. Enable camera preview

You can call the **startLocalPreview** API to enable camera preview. The SDK will request camera permission from the system. Camera images can be captured only after the permission is granted.
You can call the **setLocalRenderParams** API to set rendering parameters for the local preview. Playback may be choppy if rendering parameters are set after preview is enabled, so if you want to set rendering parameters, we recommend you call this API before enabling preview.
You can call the **TXDeviceManager** API to switch between the front and rear cameras, set the focus mode, and turn the flash on/off.
You can adjust the beauty filter effect and image quality as instructed in Setting Image Quality.
Android
iOS
Mac
Windows


```
// Set the preview mode of the local video image: Enable horizontal mirroring and s
TRTCCloudDef.TRTCRenderParams param = new TRTCCloudDef.TRTCRenderParams();
param.fillMode    = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL;
param.mirrorType = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_AUTO;
mCloud.setLocalRenderParams(param);

// Enable local camera preview (`localCameraVideo` is the control used to render th
TXCloudVideoView cameraVideo = findViewById(R.id.txcvv_main_local);
mCloud.startLocalPreview(true, cameraVideo);

// Use `TXDeviceManager` to enable autofocus and turn on the flash
```

```
TXDeviceManager manager = mCloud.getDeviceManager();
if (manager.isAutoFocusEnabled()) {
    manager.enableCameraAutoFocus(true);
}
manager.enableCameraTorch(true);
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
// Set the preview mode of the local video image: Enable horizontal mirroring and s
TRTCRenderParams *param = [[TRTCRenderParams alloc] init];
param.fillMode   = TRTCVideoFillMode_Fill;
```



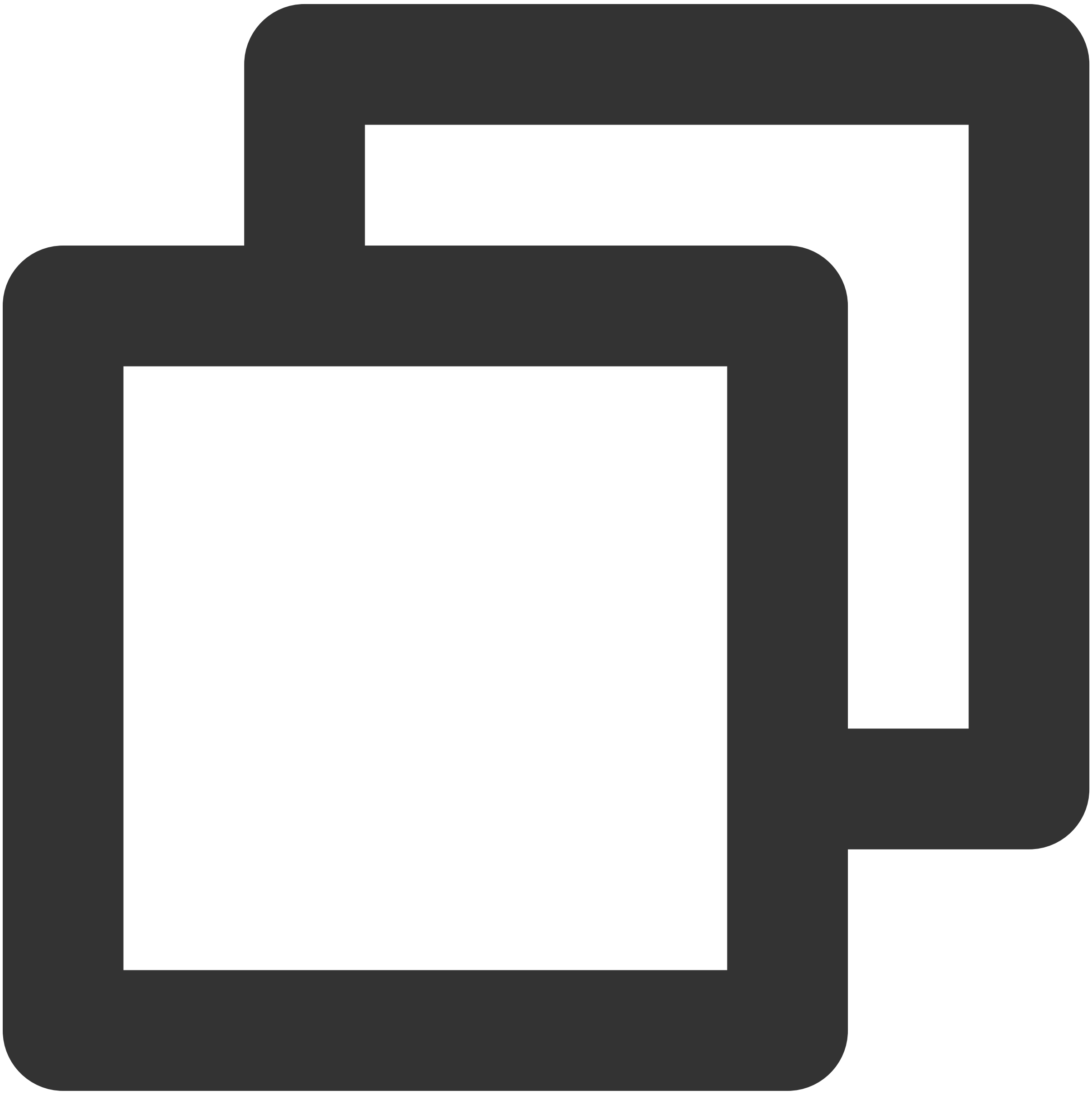
```
param.mirrorType = TRTCVideoMirrorTypeAuto;
[self.trtcCloud setLocalRenderParams:param];

// Enable local camera preview (`localCameraVideoView` is the control used to rende
[self.trtcCloud startLocalPreview:YES view:localCameraVideoView];

// Use `TXDeviceManager` to enable autofocus and turn on the flash
TXDeviceManager *manager = [self.trtcCloud getDeviceManager];
if ([manager isAutoFocusEnabled]) {
    [manager enableCameraAutoFocus:YES];
}
[manager enableCameraTorch:YES];
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
// Set the preview mode of the local video image: Enable horizontal mirroring and s
TRTCRenderParams *param = [[TRTCRenderParams alloc] init];
param.fillMode    = TRTCVideoFillMode_Fill;
param.mirrorType = TRTCVideoMirrorTypeAuto;
[self.trtcCloud setLocalRenderParams:param];

// Enable local camera preview (`localCameraVideoView` is the control used to rende
[self.trtcCloud startLocalPreview:localCameraVideoView];
```

```
// Set the preview mode of the local video image: Enable horizontal mirroring and s
liteav::TRTCRenderParams render_params;
render_params.mirrorType = liteav::TRTCVideoMirrorType_Enable;
render_params.fillMode = TRTCVideoFillMode_Fill;
trtc_cloud_->setLocalRenderParams(render_params);

// Enable local camera preview (`view` is the control handle used to render the loc
liteav::TXView local_view = (liteav::TXView)(view);
trtc_cloud_->startLocalPreview(local_view);
```

Step 3. Enable mic capture

You can call **startLocalAudio** to start mic capture. You need to specify the `quality` parameter when calling the API to set the capturing mode. A higher quality isn't necessarily better. You need to set an appropriate quality based on your business scenario.

SPEECH

In this mode, the SDK audio module is dedicated to capturing audio signals and filtering environmental noise as much as possible. In addition, the audio data in this mode has the highest immunity to a poor network quality. Therefore, it is especially suitable for scenarios highlighting audio communication, such as video calls and online meetings.

MUSIC

In this mode, the SDK uses a high audio processing bandwidth and stereo mode to maximize the capture quality while adjusting the audio DSP module to the lowest level, so as to deliver the best audio quality possible. Therefore, it is suitable for music live streaming scenarios, especially where an anchor uses a professional sound card to live stream music.

DEFAULT

In this mode, the SDK uses the smart recognition algorithm to recognize the current environment and selects the most appropriate processing mode accordingly. However, the recognition algorithm may sometimes be inaccurate. If you are very familiar with the use cases of your product, we recommend you select `SPEECH` or `MUSIC` for better audio communication or music quality.

Android

iOS&Mac

Windows

```
// Enable mic capture and set `quality` to `SPEECH` (it has a high noise suppressio
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_SPEECH );

// Enable mic capture and set `quality` to `MUSIC` (it captures high fidelity audio
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
// Enable mic capture and set `quality` to `SPEECH` (it has a high noise suppressio
[self.trtcCloud startLocalAudio:TRTCAudioQualitySpeech];

// Enable mic capture and set `quality` to `MUSIC` (it captures high fidelity audio
[self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
```



```
// Enable mic capture and set `quality` to `SPEECH` (it has a high noise suppressio
trtc_cloud_->startLocalAudio(TRTCAudioQualitySpeech);

// Enable mic capture and set `quality` to `MUSIC` (it captures high fidelity audio
trtc_cloud_->startLocalAudio(TRTCAudioQualityMusic);
```

Step 4. Enter a TRTC room

Make the current user enter a TRTC room as instructed in Entering a Room. The SDK will start publishing an audio stream to remote users upon a successful room entry.

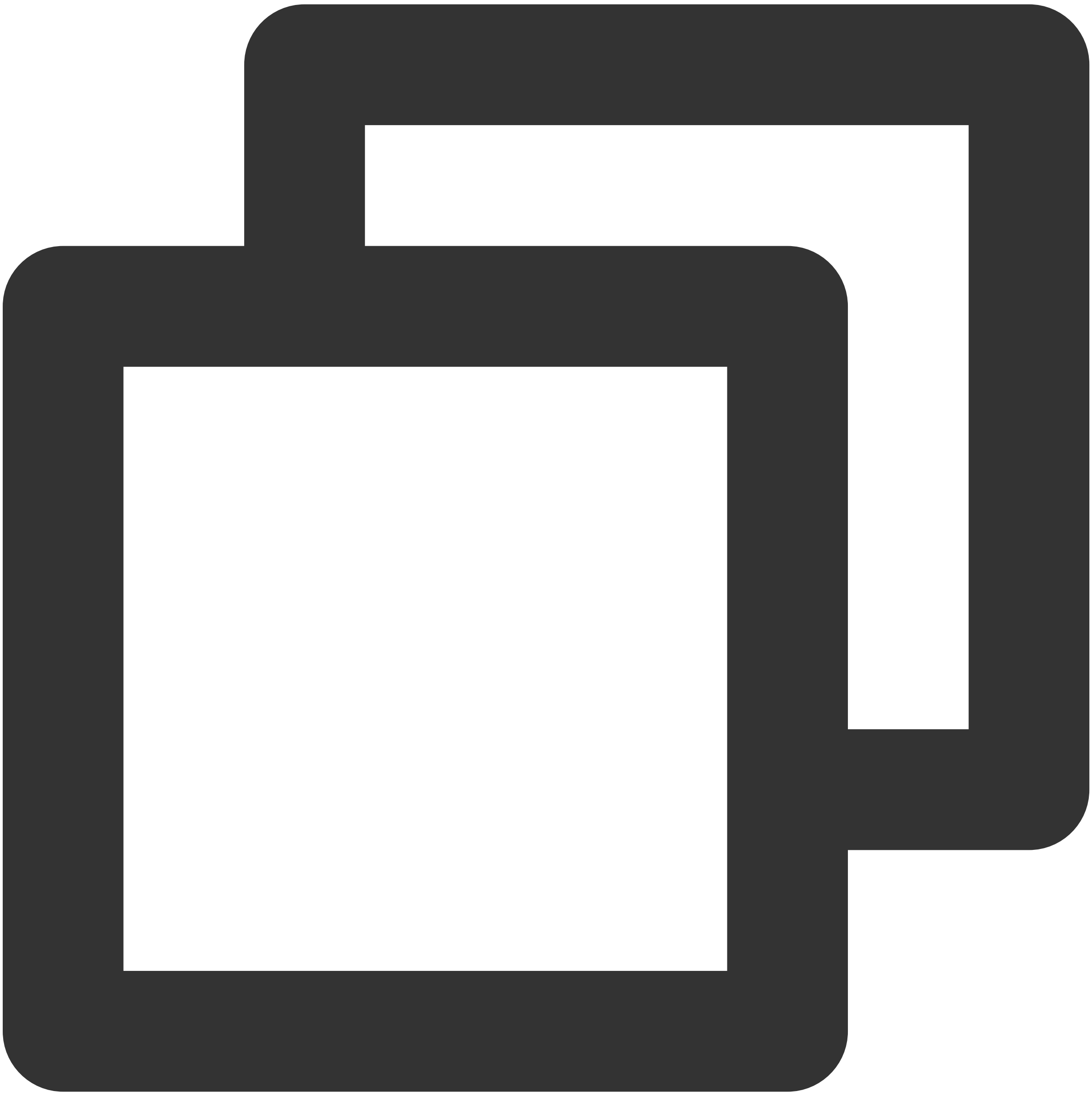
Note:

You can enable camera preview and mic capture after room entry (`enterRoom`), but in live streaming scenarios, you need to leave a certain amount of time for the anchor to test the mic and adjust the beauty filters; therefore, it is more common to turn on the camera and mic first and then enter a room.

Android

iOS&Mac

Windows



```
mCloud = TRTCCloud.sharedInstance(getApplicationContext());
mCloud.setListener(mTRTCCloudListener);

// Assemble TRTC room entry parameters. Replace the field values in `TRTCParams` wi
// Replace each field in TRTCParams with your own parameters
TRTCCloudDef.TRTCParams param = new TRTCCloudDef.TRTCParams();
params.sdkAppId = 1400000123;  // Replace with your own SDKAppID
params.userId = "denny";       // Replace with your own user ID
params.roomId = 123321;        // Replace with your own room number
params.userSig = "xxx";        // Replace with your own userSig
params.role = TRTCCloudDef.TRTCRoleAnchor;

// If your scenario is live streaming, set the application scenario to `TRTC_APP_SC
// If your application scenario is a group video call, use "TRTC_APP_SCENE_LIVE"
mCloud.enterRoom(param, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
self.trtcCloud.delegate = self;

// Assemble TRTC room entry parameters. Replace the field values in `TRTCParams` wi
// Replace each field in TRTCParams with your own parameters
TRTCParams *params = [[TRTCParams alloc] init];
params.sdkAppId = 1400000123;  // Replace with your own SDKAppID
params.roomId = 123321; // Replace with your own room number
params.userId = @"denny";   // Replace with your own userid
params.userSig = @"xxx"; // Replace with your own userSig
params.role = TRTCRoleAnchor;
```



```
// If your scenario is live streaming, set the application scenario to `TRTC_APP_SC
// If your application scenario is a group video call, use "TRTC_APP_SCENE_LIVE"
[self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
```

```
trtc_cloud_ = getTRTCShareInstance();

// Assemble TRTC room entry parameters. Replace the field values in `TRTCParams` wi
// Replace each field in TRTCParams with your own parameters
liteav::TRTCParams params;
```

```
params.sdkAppId = 1400000123;  // Replace with your own SDKAppID
params.userId = "denny";       // Replace with your own user ID
params.roomId = 123321;        // Replace with your own room number
params.userSig = "xxx";        // Replace with your own userSig
params.role = TRTCCloudDef.TRTCRoleAnchor;

// If your scenario is live streaming, set the application scenario to `TRTC_APP_SC
// If your application scenario is a group video call, use "TRTC_APP_SCENE_LIVE"
trtc_cloud_->enterRoom(params, liteav::TRTCAppSceneLIVE);
```

Step 5. Switch the role

Role in TRTC

In video call (`TRTC_APP_SCENE_VIDEOCALL`) and audio call (`TRTC_APP_SCENE_AUDIOCALL`) scenarios, you don't need to set the role when entering a room, as each user is an anchor by default in these two scenarios.

In video live streaming (`TRTC_APP_SCENE_LIVE`) and audio live streaming (`TRTC_APP_SCENE_VOICE_CHATROOM`) scenarios, each user needs to set their own role to anchor or audience when entering a room.

Role switch

In TRTC, only anchors can publish audio/video streams.

Therefore, if you set the role to audience when entering a room, you need to call the **switchRole** API first to switch the role to anchor before publishing audio/video streams. This process is the so-called "mic-on".

Android

iOS

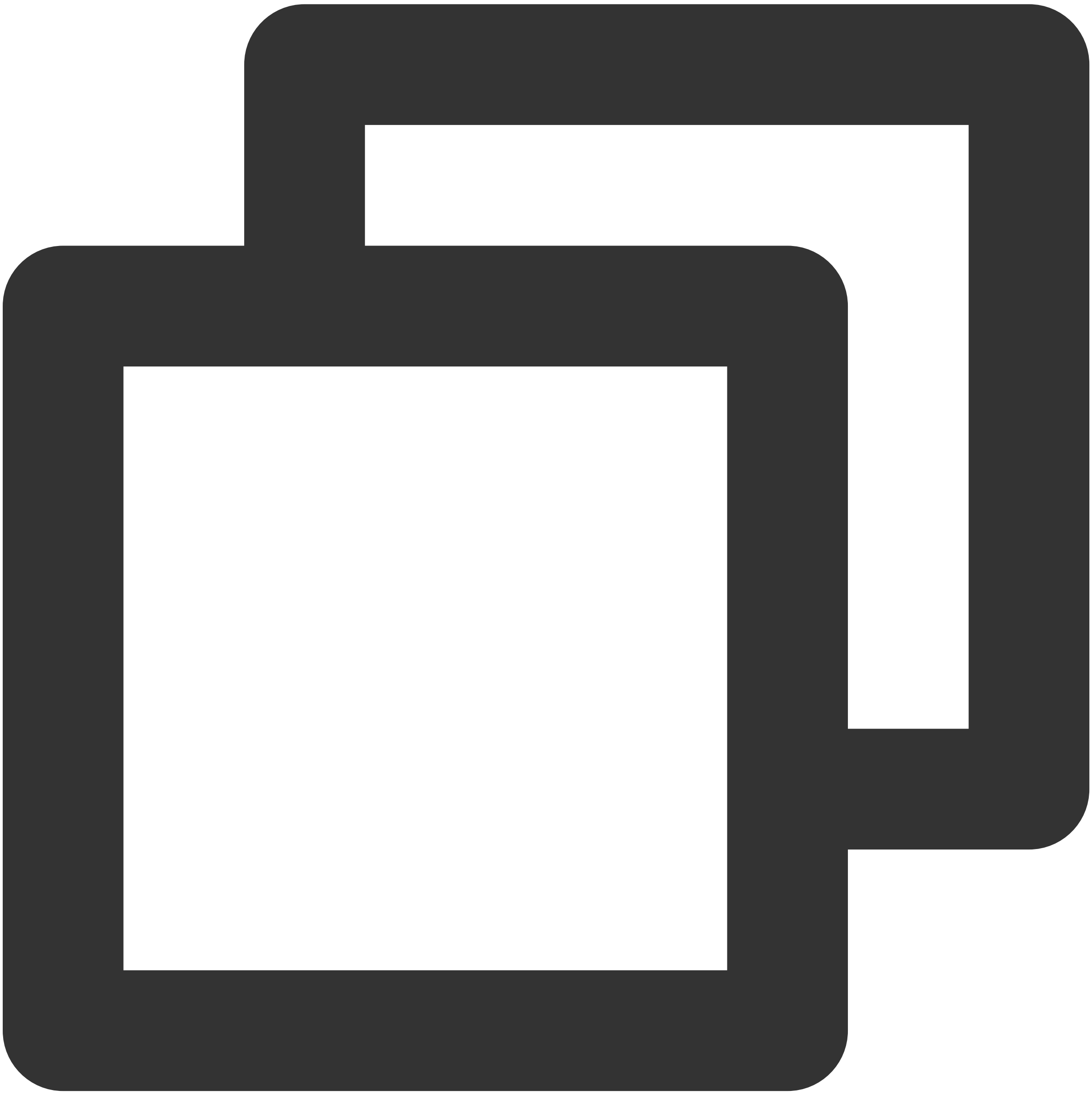
Mac

Windows

```
// If your current role is audience, you need to call `switchRole` first to switch
// If your current role is 'audience', you need to call switchRole to switch to 'an
mCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
mCloud.startLocalPreview(true, cameraVide);

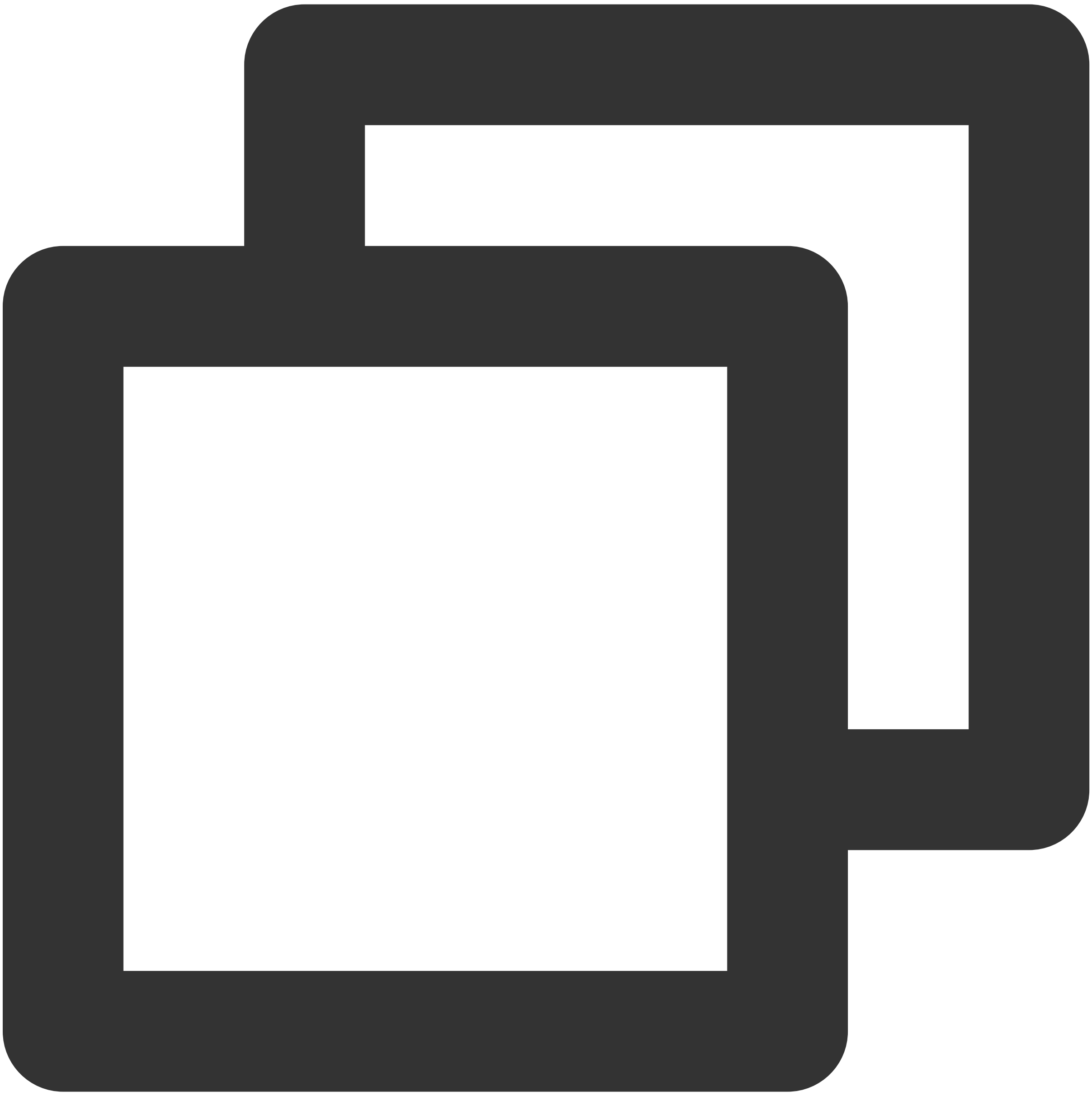
// If role switch failed, the error code of the `onSwitchRole` callback is not `0`
// If switching operation failed, the error code of the 'onSwitchRole' is not zero
@Override
public void onSwitchRole(final int errCode, final String errMsg) {
    if (errCode != 0) {
```

```
        Log.d(TAG, "Switching operation failed...");
    }
}
```



```
self.trtcCloud = [TRTCCloud sharedInstance];
// If your current role is audience, you need to call `switchRole` first to switch
// If your current role is 'audience', you need to call switchRole to switch to 'an
[self.trtcCloud switchRole:TRTCRoleAnchor];
[self.trtcCloud startLocalAudio:TRTCAudioQualityDefault];
[self.trtcCloud startLocalPreview:YES view:localCameraVideoView];
```

```
// If role switch failed, the error code of the `onSwitchRole` callback is not `0`
// If switching operation failed, the error code of the 'onSwitchRole' is not zero
- (void)onSwitchRole:(TXLiteAVError)errCode errMsg:(nullable NSString *)errMsg {
    if (errCode != 0) {
        NSLog(@"Switching operation failed... ");
    }
}
```



```
self.trtcCloud = [TRTCCloud sharedInstance];
```

```
// If your current role is audience, you need to call `switchRole` first to switch
// If your current role is 'audience', you need to call switchRole to switch to 'an
[self.trtcCloud switchRole:TRTCRoleAnchor];
[self.trtcCloud startLocalAudio:TRTCAudioQualityDefault];
[self.trtcCloud startLocalPreview:localCameraVideoView];

// If role switch failed, the error code of the `onSwitchRole` callback is not `0`
// If switching operation failed, the error code of the 'onSwitchRole' is not zero
- (void)onSwitchRole:(TXLiteAVError)errCode errMsg:(nullable NSString *)errMsg {
    if (errCode != 0) {
        NSLog(@"Switching operation failed... ");
    }
}
```

```
// If your current role is audience, you need to call `switchRole` first to switch
// If your current role is 'audience', you need to call switchRole to switch to 'an
trtc_cloud_->switchRole(liteav::TRTCRoleAnchor);
trtc_cloud_->startLocalAudio(TRTCAudioQualityDefault);
trtc_cloud_->startLocalPreview(hWnd);

// If role switch failed, the error code of the `onSwitchRole` callback is not `ERR
// If switching operation failed, the error code of the 'onSwitchRole' is not zero
void onSwitchRole(TXLiteAVError errCode, const char* errMsg) {
    if (errCode != ERR_NULL) {
        printf("Switching operation failed...");
```



```
    }
}
```

Note: If there are too many anchors in a room, `switchRole` will fail, and TRTC will notify you of the error code through `onSwitchRole` . Therefore, if you no longer want to publish audio/video streams, you need to call `switchRole` again to switch to audience. This process is the so-called "mic-off".

Note:

Only an anchor can publish audio/video streams, but you cannot make each user enter the room as an anchor. For the specific reason, see 1. How many concurrent audio/video streams can a room have at most?

Advanced Guide

1. How many concurrent audio/video streams can a room have at most?

A TRTC room can have up to **50** concurrent audio/video streams. When the number of streams reaches 50, additional streams will be automatically dropped.

With proper **role management**, 50 concurrent audio/video streams can meet the needs of most scenarios, from a one-to-one video call to live streams watched by tens of thousands of users.

"Role management" refers to how roles are assigned to users entering a room.

If a user is an anchor in live streaming, a teacher in online education, or a host in an online meeting, they can be assigned the anchor role.

If a user is a live stream viewer, a student in online education, or an attendee in an online meeting, they should be assigned the audience role; otherwise, the room will quickly reach the maximum number of anchors.

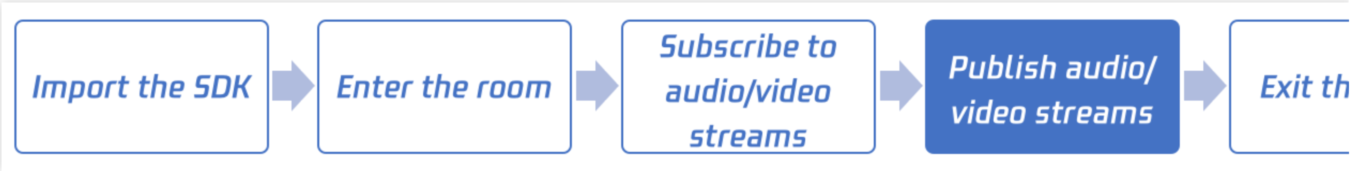
When an audience member wants to publish an audio/video stream (mic on), they need to be switched to the anchor role through `switchRole` . When they no longer want to publish their audio/video streams (mic off), they need to be switched back to the audience role immediately.

With appropriate role management, generally no more than 50 anchors in a room need to publish audio/video streams concurrently. If a room contains more than six anchors, audience members will find it difficult to distinguish between speakers who are speaking at the same time.

Web

Last updated：2023-09-26 17:01:08

This document describes how an anchor publishes audio/video streams. "Publishing" refers to turning on the mic and camera to make the audio heard and video seen by other users in the room.



Step 1. Enter Room

For detailed directions, see Enter Room.

Step 2. Turn on camera

Use the `trtc.startLocalVideo()` API to turn on the camera and publish it to the room.

```
// To preview the camera image, you need to place an HTMLElement in the DOM, which
const view = 'local-video';
await trtc.startLocalVideo({ view });
```

Step 3. Turn on microphone

Use the `trtc.startLocalAudio()` API to turn on the microphone and publish it to the room.


```
await trtc.startLocalAudio();
```

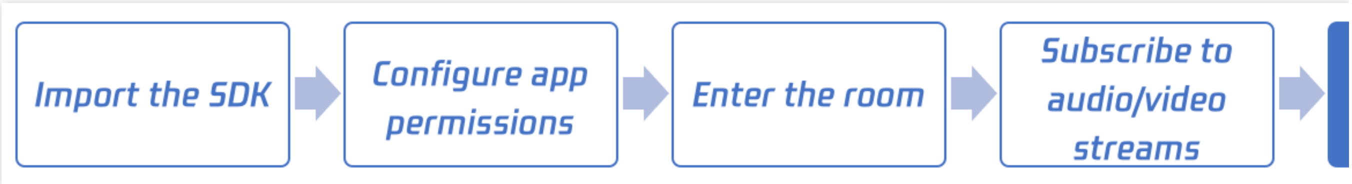
Step 4. Turn off camera/microphone

Use the trtc.stopLocalVideotrtc.stopLocalAudio to turn off camera/microphone.

Electron

Last updated：2024-05-21 15:05:29

This document describes how an anchor publishes audio/video streams. "Publishing" refers to turning on the mic and camera to make the audio heard and video seen by other users in the room.



Call Guide

Step 1. Perform prerequisite steps

Import the SDK as instructed in Electron.

Step 2. Enable camera preview

You can call the **startLocalPreview** API to enable camera preview. The SDK will request camera permission from the system. Camera images can be captured only after the permission is granted.
You can call the **setLocalRenderParams** API to set the rendering parameters of local preview. Image jitters may occur if preview parameters are set after preview is enabled, so we recommend you call this API before enabling preview.

```
// Set the rendering parameters of local preview: Flip the video horizontally and u
import TRTCCloud, {
    TRTCRenderParams, TRTCVideoRotation,
    TRTCVideoFillMode, TRTCVideoMirrorType
} from 'trtc-electron-sdk';

const param = new TRTCRenderParams(
    TRTCVideoRotation.TRTCVideoRotation0,
    TRTCVideoFillMode.TRTCVideoFillMode_Fill,
    TRTCVideoMirrorType.TRTCVideoMirrorType_Auto
);
```

```
const rtcCloud = new TRTCCloud();
rtcCloud.setLocalRenderParams(param);
const cameraVideoDom = document.querySelector('.camera-dom');
rtcCloud.startLocalPreview(cameraVideoDom);
```

Step 3. Enable mic capture

You can call **startLocalAudio** to start mic capture. When calling this API, you need to specify `quality`. A higher quality isn't necessarily better. We recommend you set this parameter based on the application scenario.

SPEECH

In this mode, the SDK audio module is dedicated to capturing audio signals and filtering environmental noise as much as possible. In addition, the audio data in this mode has the highest immunity to a poor network quality. Therefore, it is especially suitable for scenarios highlighting audio communication, such as video calls and online meetings.

MUSIC

In this mode, the SDK uses a high bandwidth for audio processing and uses the stereo mode to maximize the capturing quality while minimizing the role of the DSP module. It guarantees audio quality and is therefore suitable for music streaming scenarios, especially when an anchor uses a high-end sound card.

DEFAULT

In this mode, the SDK uses an algorithm to recognize the current environment and selects the processing mode accordingly. However, the recognition algorithm is not always accurate, so if your product has a clear positioning (for example, an audio chat app or a music streaming app), we recommend you set the parameter to `SPEECH` or `MUSIC`.

```
import { TRTCAudioQuality } from 'trtc-electron-sdk';
// Enable mic capture and set `quality` to `SPEECH` (strong in noise suppression an
rtcCloud.startLocalAudio(TRTCAudioQuality.TRTCAudioQualitySpeech);

// Enable mic capture and set `quality` to `MUSIC` (high fidelity, minimum audio qu
rtcCloud.startLocalAudio(TRTCAudioQuality.TRTCAudioQualityMusic);
```

Step 4. Enter a TRTC room

Make the current user enter a TRTC room as instructed in Entering a Room. The SDK will start publishing the audio of the local user to remote users upon successful room entry.

Note:

You can enable camera preview and mic capture after room entry (`enterRoom`), but in live streaming scenarios, you need to leave a certain amount of time for the anchor to test the mic and adjust the beauty filters; therefore, it is more common to turn on the camera and mic first and then enter a room.

```
import { TRTCParams, TRTCRoleType, TRTCAppScene } from 'trtc-electron-sdk';

// Assemble TRTC room entry parameters. Replace the field values in `TRTCParams` wi
```

```
// Replace each field in TRTCParams with your own parameters
const param = new TRTCParams();
params.sdkAppId = 1400000123;  // Replace with your own SDKAppID
params.userId = "denny";       // Replace with your own user ID
params.roomId = 123321;        // Replace with your own room number
params.userSig = "xxx";        // Replace with your own userSig
params.role = TRTCRoleType.TRTCRoleAnchor;

// If your scenario is live streaming, set the application scenario to `TRTC_APP_SC
// If your application scenario is a group video call, use "TRTC_APP_SCENE_LIVE"
rtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneLIVE);
```

Flutter

Last updated：2024-02-02 18:51:54

This document describes how an anchor publishes audio/video streams. "Publishing" refers to turning on the mic and camera to make the audio heard and video seen by other users in the room.

Call Guidelines

Step 1. Perform prerequisite steps

Refer to the document Import SDK into the project to accomplish the import of SDK and for the configuration of App permissions.

Step 2. Enable camera preview

You can call the **startLocalPreview** API to enable camera preview. At this point, the SDK will request usage permission from the system and the camera's capturing process will begin after user authorization.
If you wish to set the rendering parameters for the local image, you can use the **setLocalRenderParams** API. To prevent image flickering from occuring due to setting preview parameters after the preview starts, it's recommended to call this before initiating the preview.
If you want to control various camera parameters, you can use the **TXDeviceManager** API, which allows operations such as "switching between cameras", "setting focus mode","turning the flash on or off", amongst others.
If you wish to adjust the beauty filter effect and image quality, this will be detailed in Setting Image Quality.


```
// Set the rendering parameters of local preview: Flip the video horizontally and u
trtcCloud.setLocalRenderParams(TRTCRenderParams(
    fillMode: TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL
    mirrorType: TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_ENABLE);

// Initiate preview for the local camera (`viewId` denotes the unique view identifi
trtcCloud.startLocalPreview(isFrontCamera, viewId);

// Use `TXDeviceManager` to enable autofocus and turn on the flash
bool? isAutoFocusEnabled = await manager.isAutoFocusEnabled();
if (isAutoFocusEnabled ?? false) {
```

```
  manager.enableCameraAutoFocus(true);
}
manager.enableCameraTorch(true);
```

Step 3. Enable mic capture

You may invoke **startLocalAudio** to initiate microphone acquisition, this interface requires you to establish a collection pattern via the `quality` parameter. Although named quality, it does not denote that a higher value yields superior results, different business scenarios require specific parameter selection (a more accurate name would be 'scene').

TRTC_AUDIO_QUALITY_SPEECH

Under this pattern, the SDK's audio module centers on refining speech signals, strives to filter ambient noise to the highest degree possible, and the audio data will also attain optimal resistance against poor network quality. Thus, this pattern proves particularly useful for scenarios emphasizing vocal communication, such as "video conferencing" and "online meetings".

TRTC_AUDIO_QUALITY_MUSIC

Under this pattern, the SDK will employ a high level of audio processing bandwidth and stereoscopic pattern, which while maximizing the collection quality will also condition the audio's DSP processing module to the weakest level, ensuring the audio quality to the fullest extent. Therefore, this pattern is suitable for "music live broadcast" scenarios, and is especially beneficial for hosts making use of professional sound cards for music live broadcasts.

TRTC_AUDIO_QUALITY_DEFAULT

Under this pattern, the SDK will activate Intelligent Identification algorithm to recognise the current environment and choose the most appropriate handling pattern accordingly. However, even the best detection algorithms are not always accurate, so if you have a clear understanding of the positioning of your product, it is more recommended for you to choose between the Speech focused 'SPEECH' and the music quality focused 'MUSIC'.

```
// Enable mic capture and set `quality` to `SPEECH` (strong in noise suppression an
trtcCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_SPEECH );

// Enable mic capture and set `quality` to `MUSIC` (high fidelity, minimum audio qu
trtcCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
```

Step 4. Enter a TRTC room

Refer to the document Enter the Room to guide the current user to enter the TRTC room. After successfully entering the room, the SDK will begin to publish its own audio stream to other users in the room.

Note:

Naturally, you can turn on the camera preview and microphone capture after entering the room (enterRoom). However, in live broadcast situations, we need to give the host some time to test the microphone and adjust the beauty filter. Therefore, it is more common to start the camera and the microphone before entering the room.

```
// Create a TRTCCloud singleton
trtcCloud = (await TRTCCloud.sharedInstance())!;
// Register TRTC event callback
trtcCloud.registerListener(onRtcListener);
```

```
enterRoom() async {
  try {
    userInfo['userSig'] =
        await GenerateTestUserSig.genTestSig(userInfo['userId']);
    meetModel.setUserInfo(userInfo);
  } catch (err) {
    userInfo['userSig'] = '';
    print(err);
  }
  // If your scenario is "interactive video live broadcast", please set the scene t
  await trtcCloud.enterRoom(
      TRTCParams(
          sdkAppId: GenerateTestUserSig.sdkAppId,
          userId: userInfo['userId'],
          userSig: userInfo['userSig'],
          role: TRTCCloudDef.TRTCRoleAnchor,
          roomId: meetId!),
      TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}
```

Step 5. Switch the role

"role" in TRTC

In the "Video Call" (TRTC_APP_SCENE_VIDEOCALL) and "Voice Call" (TRTC_APP_SCENE_AUDIOCALL) scenarios, there is no necessity to establish a role upon entering the room, for in these two patterns, each participant is inherently designated as an Anchor.

In the contexts of both "Video Broadcasting" (TRTC_APP_SCENE_LIVE) and "Voice Broadcasting" (TRTC_APP_SCENE_VOICE_CHATROOM), every user needs to designate their specific "role" upon entering a room. They either become an "Anchor" or an "Audience Member".

Role Transition

Within the framework of TRTC, only the "Anchor" possesses the authority to disseminate audio and video streams. The "Audience" lacks this permission.

Consequently, should you opt for the 'Audience' role upon entering the room, it necessitates an initial invocation of the **switchRole** interface to transform your role into an "Anchor", followed by the dissemination of audio and video streams, colloquially known as 'going live'.


```
// If your current role is audience, you need to call `switchRole` first to switch
// If your current role is 'audience', you need to call switchRole to switch to 'an
trtcCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);
trtcCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
trtcCloud.startLocalPreview(true, cameraVideo);

// If role switch failed, the error code of the `onSwitchRole` callback is not `0`
// If switching operation failed, the error code of the 'onSwitchRole' is not zero
onRtcListener(type, param) async {
  if (type == TRTCCloudListener.onSwitchRole) {
    if (param['errCode'] != 0) {
```



```
        // TO DO
      }
    }
  }
```

Note:

If there are already too many hosts in the room, it can lead to a switchRole role change failure, and the error code will be called back to you through TRTC's onSwitchRole as a notification. Therefore, when you no longer need to broadcast audio and video streams (commonly referred to as "stepping down"), you need to call switchRole again and switch to "Audience".

Note:

You may have a query: if only the host can publish the audio-video streams, wouldn't it be possible for every user to step into the room using the host's role? The answer is certainly no. The rationale can be explored in the advanced guide What's the maximum number of simultaneous audio and video streams that can be facilitated in a single room?

Advanced Guide

1. How many concurrent audio/video streams can a room have at most?

In the confines of a TRTC room, it is permissible to maintain a maximum of **50** synchronized audio-visual streams; any excess streams will be discarded based on the principle of "first come, first served".

Under the majority of scenarios, ranging from video calls between two individuals to online live broadcasts watched by tens of thousands simultaneously, the provision of 50 concurrent audio-visual streams would suffice for the needs of application scenarios. However, satisfying this precondition necessitates the proper administration of the **role management**.

"Role management" refers to how roles are assigned to users entering a room.

Should a user hold the role of an "Anchor" in a live broadcasting scenario, a "Teacher" in an online education setting, or a "Host" in an online conference scenario, they can all be assigned the role of "Anchor".

If a user is inherently an "audience" in a live streaming scenario, a "student" in an online education scenario, or an "observer" in an online meeting scenario, they should be delegated to the "Audience" role. Otherwise, their overwhelming number could instantaneously "overload" the limit of the host's count.

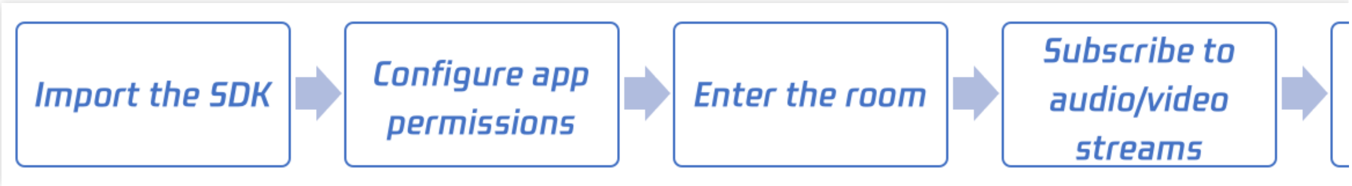
Only when the "audience" needs to broadcast audio and video streams ("going on mic"), do they need to switch to the "anchor" role through switchRole. As soon as they no longer need to broadcast audio and video streams ("off mic"), they should immediately switch back to the audience role.

Through adept role management, you will discover that the number of 'broadcasters' that need to concurrently transmit audio and video streams in a room typically does not exceed 50. Otherwise, the entire room would descend into a state of 'chaos', bear in mind, once the number of simultaneous voices exceeds 6, it becomes rather difficult for the common person to discern who precisely is speaking.

05. Exiting a Room
Android, iOS, Windows, and macOS

Last updated：2023-09-26 17:02:02

This document describes how to actively exit the current TRTC room and in which cases will a user be forced to exit a room.



Call Guide

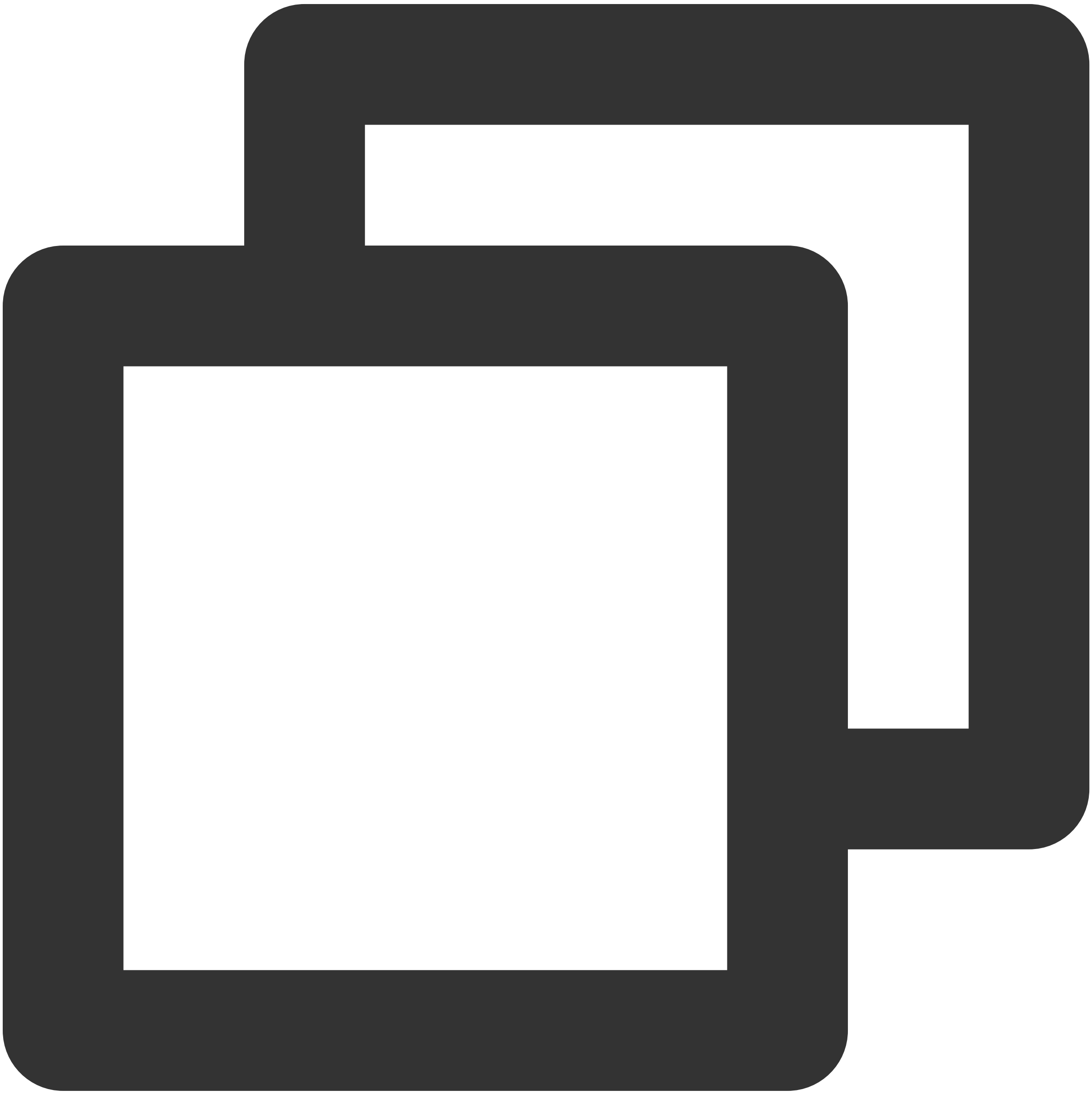
Step 1. Perform prerequisite steps

Import the SDK and configure the application permissions as instructed in iOS.
Enter the room as instructed in Entering a Room.

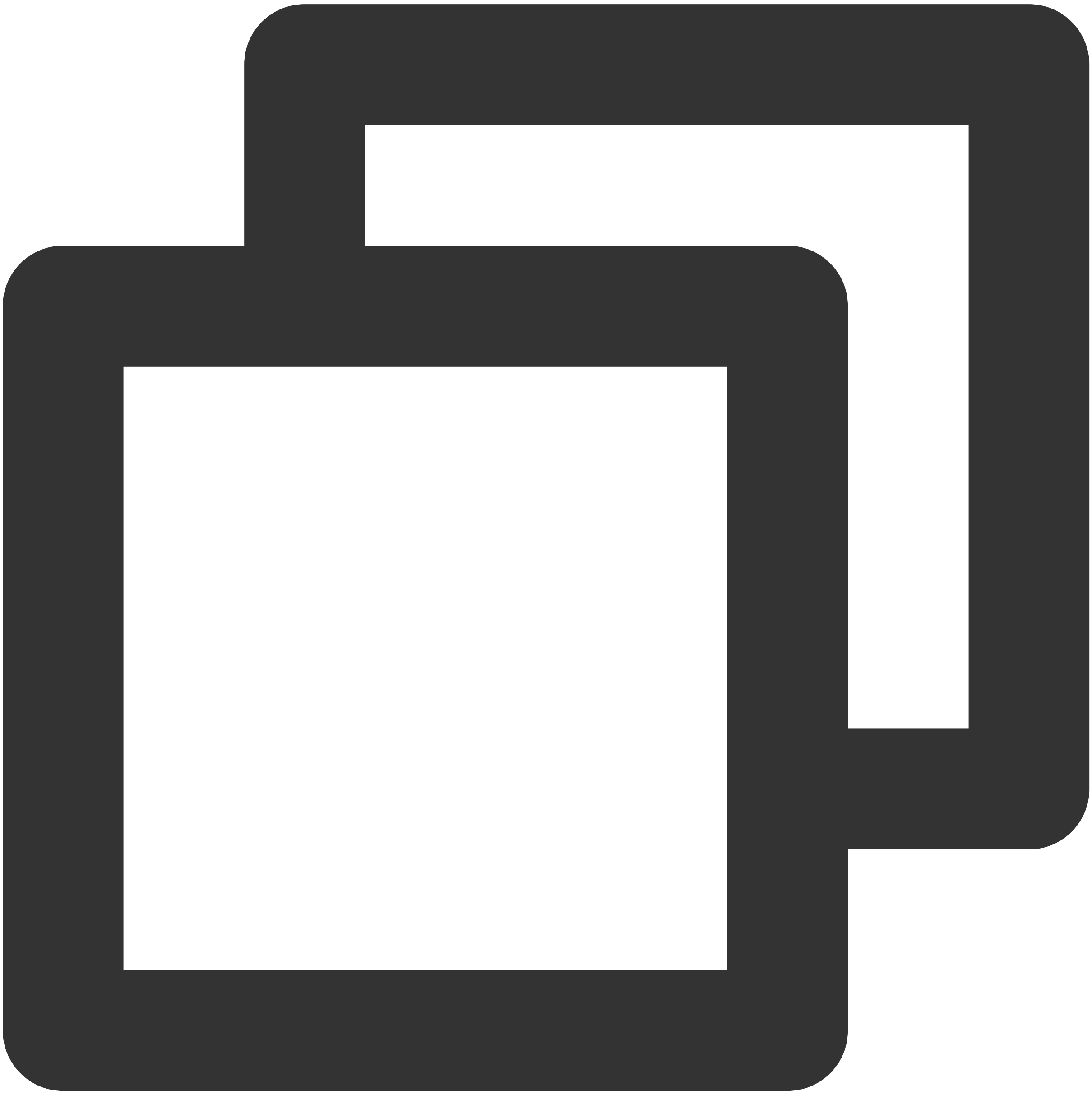
Step 2. Actively exit the current room

Call the `exitRoom` API to exit the current room, and the SDK will use the `onExitRoom(int reason)` callback to notify you of the reason the room was exited.

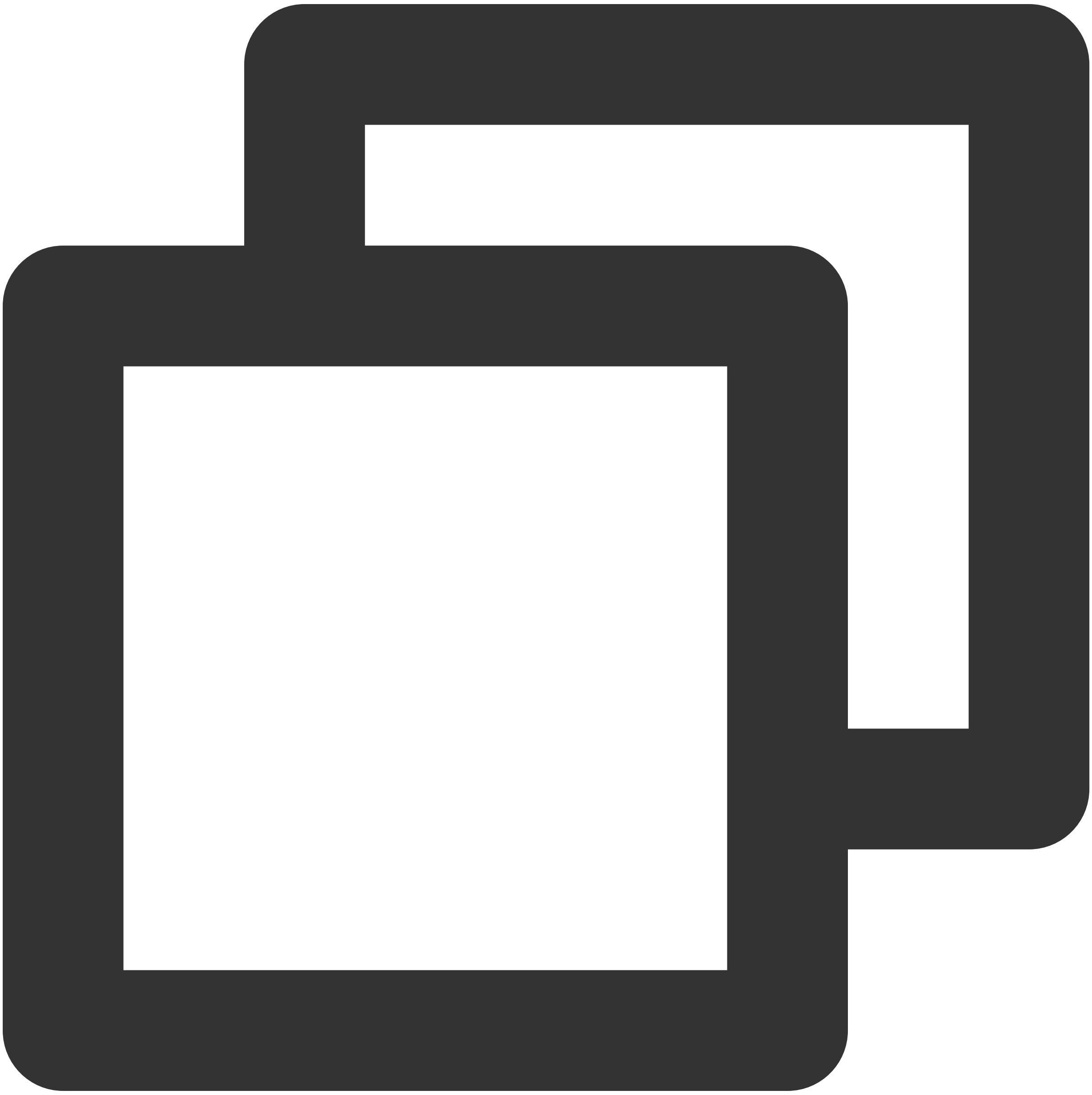
Android
iOS&Mac
Windows



```
// Exit the current room
mCloud.exitRoom();
```



```
self.trtcCloud = [TRTCCloud sharedInstance];
// Exit the current room
[self.trtcCloud exitRoom];
```



```
trtc_cloud_ = getTRTCShareInstance();
// Exit the current room
trtc_cloud_->exitRoom();
```

After the `exitRoom` API is called, the SDK will enter the room exit process, where two key tasks need to be completed:

1. Notify the exit of the current user

Notify other users in the room of the upcoming room exit, and they will receive the **onRemoteUserLeaveRoom** callback from the current user; otherwise, other users may think the current user's video image is simply frozen.

2. Revoke device permissions

If the current user is publishing an audio/video stream before exiting the room, the user needs to turn off the camera and mic and release the device permissions during the room exit process.

Therefore, we recommend you release the `TRTCCloud` instance after receiving the `onExitRoom` callback.

Step 3. Be forced to exit the current room

The `onExitRoom(int reason)` callback will also be received in other two cases in addition to active room exit:

Case 1. A user is kicked out of the room

You can use the RemoveUser or RemoveUserByStrRoomId API to kick a user out of a TRTC room. After being kicked out, the user will receive the `onExitRoom(1)` callback.

Case 2. The current room is closed

You can call the DismissRoom or DismissRoomByStrRoomId API to close a TRTC room. After the room is closed, all users in the room will receive the `onExitRoom(2)` callback.

Android

iOS&Mac

Windows

```
// Listen for the `onExitRoom` callback to get the reason for room exit
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        Log.d(TAG, "Kicked out of the current room by server through the restful ap
    } else if (reason == 2) {
        Log.d(TAG, "Current room is dissolved by server through the restful api..."
    }
}
```

```
// Listen for the `onExitRoom` callback to get the reason for room exit
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        NSLog(@"Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        NSLog(@"Kicked out of the current room by server through the restful api...
    } else if (reason == 2) {
        NSLog(@"Current room is dissolved by server through the restful api...");
    }
```

```
}
```

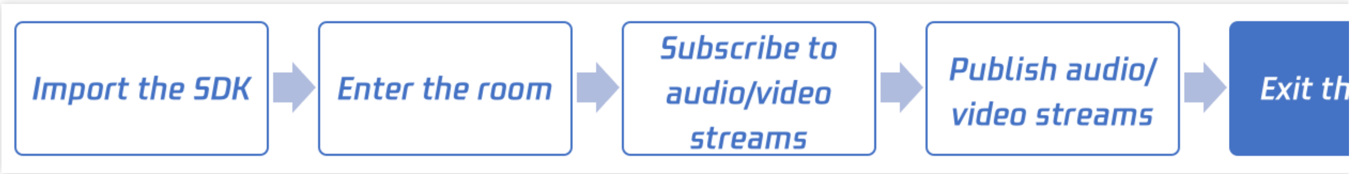
```
// Listen for the `onExitRoom` callback to get the reason for room exit
void onExitRoom(int reason) {
    if (reason == 0) {
        printf("Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        printf("Kicked out of the current room by server through the restful api...
    } else if (reason == 2) {
        printf("Current room is dissolved by server through the restful api...");
```

```
        }
}
```

Web

Last updated：2023-10-27 10:10:04

This document describes how to exit the current TRTC room and in which cases a user is forced to exit the room.



Step 1. Enter a room

Create a trtc instance and enter a room. For detailed directions, see Entering a Room.

Step 2. Exit the room

Call the `trtc.exitRoom()` method to exit the room and end the audio and video call.

```
await trtc.exitRoom();
// After the exit is successful, if you do not need to use the trtc instance later,
trtc.destroy();
```

Handling being kicked out

In addition to actively exiting the room, users may also be kicked out of the room for the following reasons.

1. `kick` : Two users with the same userId enter the same room, and the user who enters the room first will be kicked out. It is not allowed for users with the same name to enter the same room at the same time, which may cause abnormal audio and video calls between the two parties, so this situation should be avoided.

`banned` : A user is kicked out of a TRTC room through the server's RemoveUser | RemoveUserByStrRoomId interface. The user will receive a kicked event, and the reason is `banned` .
2. `room-disband` : A TRTC room is dissolved through the server's DismissRoom | DismissRoomByStrRoomId interface. After the room is dissolved, all users in the room will receive a kicked event, and the reason is `room-disband` .
At this time, the SDK will throw the KICKED_OUT event. There is no need to call `trtc.exitRoom()` to exit the room, and the SDK will automatically enter the exit room state.

```
trtc.on(TRTC.EVENT.KICKED_OUT, error => {
  console.error(`kicked out, reason:${error.reason}, message:${error.message}`);
```

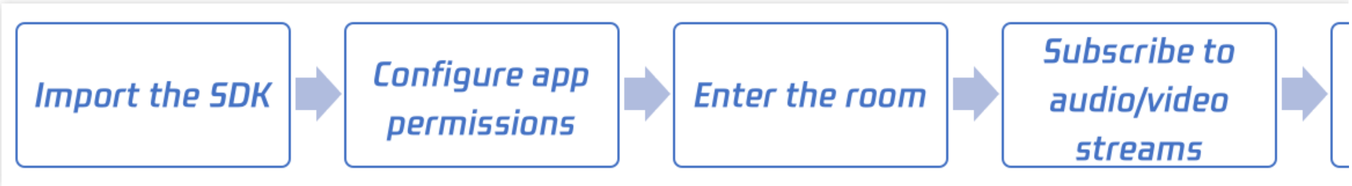


```
  // error.reason has the following situations
  // 'kick' The user with the same userId enters the same room, causing the user wh
  // 'banned' The administrator removed the user from the room
  // 'room-disband' The administrator dissolved the room
});
```


Electron

Last updated：2023-09-26 17:03:05

This document describes how to actively exit the current TRTC room and in which cases will a user be forced to exit a room.



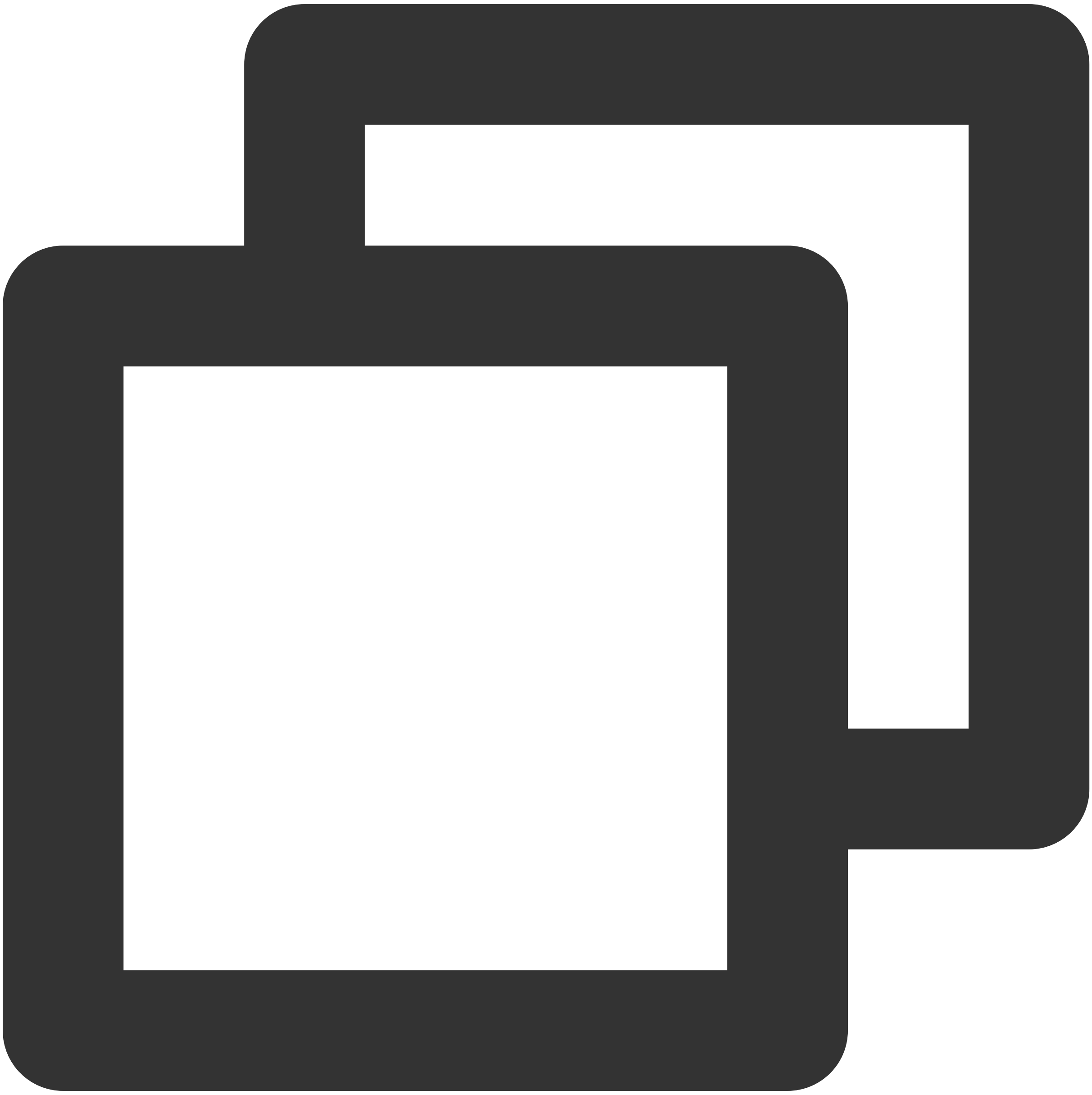
Call Guide

Step 1. Perform prerequisite steps

1. Import the SDK and configure the application permissions as instructed in Electron.
2. Implement the room entry process as instructed in Electron.

Step 2. Actively exit the current room

Call the exitRoom API to exit the current room. The SDK uses the `onExitRoom(int reason)` callback event to notify you of the reason the room was exited.



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

// Exit the current room
trtcCloud.exitRoom();
```

After the `exitRoom` API is called, the SDK will enter the room exit process, where two key tasks need to be completed:

1. Notify the exit of the current user

Notify other users in the room of the upcoming room exit, and they will receive the **onRemoteUserLeaveRoom** callback from the current user; otherwise, other users may think the current user's video image is simply frozen.

2. Revoke device permissions

If the current user is publishing an audio/video stream before exiting the room, the user needs to turn off the camera and mic and release the device permissions during the room exit process.

Therefore, we recommend you release the `TRTCCloud` instance after receiving the `onExitRoom` callback.

Step 3. Be forced to exit the current room

The onExitRoom callback will also be received in other two cases in addition to active room exit:

Case 1. A user is kicked out of the room

You can use the RemoveUser or RemoveUserByStrRoomId API to kick a user out of a TRTC room. After being kicked out, the user will receive the `onExitRoom(1)` callback.

Case 2. The current room is closed

You can call the DismissRoom or DismissRoomByStrRoomId API to close a TRTC room. After the room is closed, all users in the room will receive the `onExitRoom(2)` callback.

```
// Listen for the `onExitRoom` callback to get the reason for room exit
function onExitRoom(reason) {
  console.log(`onExitRoom reason: ${reason}`);
}
trtcCloud.on('onExitRoom', onExitRoom);
```


Flutter

Last updated：2024-02-02 18:46:32

This document describes how to actively exit the current TRTC room and in which cases will a user be forced to exit a room.

Call Guidelines

Step 1. Perform prerequisite steps

Kindly refer to the document Importing the SDK into your project for the completion of SDK import and App permission configuration.
Please refer to the document Entering the Room for the completion of the room entry process.

Step 2. Actively exit the current room

Invoke the exitRoom interface to leave the current room. The SDK will notify you via the onExitRoom(int reason) callback event once the room exit completes.

```
// Exit the current room
await trtcCloud.exitRoom();
```

Upon invoking the exitRoom interface, the SDK initiates the room-exit process, which entails two particularly significant tasks:

Key Task One: Announce Your Departure

Notify the other users in the room that you are about to leave. They will receive a **onRemoteUserLeaveRoom** callback from the departing user, otherwise they might mistakenly think that the user has "frozen".

Key Task Two: Release Device Permissions

If the user is streaming audio or video before leaving, the process of leaving requires the shutdown of the camera and microphone, as well as the release of the device's permissions.

Therefore, if you wish to release the TRTCCloud instance, it is recommended to do so only after receiving the onExitRoom callback.

Step 3. Be forced to exit the current room

Aside from a user's active departure from a room, there are two other scenarios in which you will receive an onExitRoom (int reason) callback:

Scenario One: Being ejected from the current room

Via the server side's RemoveUser | RemoveUserByStrRoomId interface, a specific user can be ejected from a particular TRTC room. Following the ejection of this user, they will receive an onExitRoom(1) callback.

Scenario Two: The current room is dismissed

Through the server side's DismissRoom | DismissRoomByStrRoomId interface, a certain TRTC room can be dismissed, and upon the dissolution of the room, all users within it will receive an onExitRoom(2) callback.

```
onRtcListener(type, param) async {
    if (type == TRTCCloudListener.onExitRoom) {
      if (param == 0) {
          log('Exit current room by calling the 'exitRoom' api of sdk ...');
      } else if (param == 1) {
          log('Kicked out of the current room by server through the restful api...'
      } else if (param == 2) {
          log('Current room is dissolved by server through the restful api...');
      }
    }
}
```


06. Advanced Guide
Sensing Network Quality
Android, iOS, Windows, and macOS

Last updated：2023-09-26 17:03:35

This document describes how to assess the current network conditions.
When you have a video call on WeChat under poor network conditions (for example, when you are in an elevator), WeChat displays a message indicating the current network quality is poor. This document describes how to use TRTC to implement a similar interaction in your own application.

Call Guide

TRTC provides the **onNetworkQuality** callback to report the current network quality once every two seconds. It contains two parameters: `localQuality` and `remoteQuality` .
localQuality indicates your current network quality, which has six levels: `Excellent` , `Good` , `Poor` , `Bad` , `VeryBad` , and `Down` .
remoteQuality is an array indicating the network quality of remote users. In this array, each element represents the network quality of a remote user.

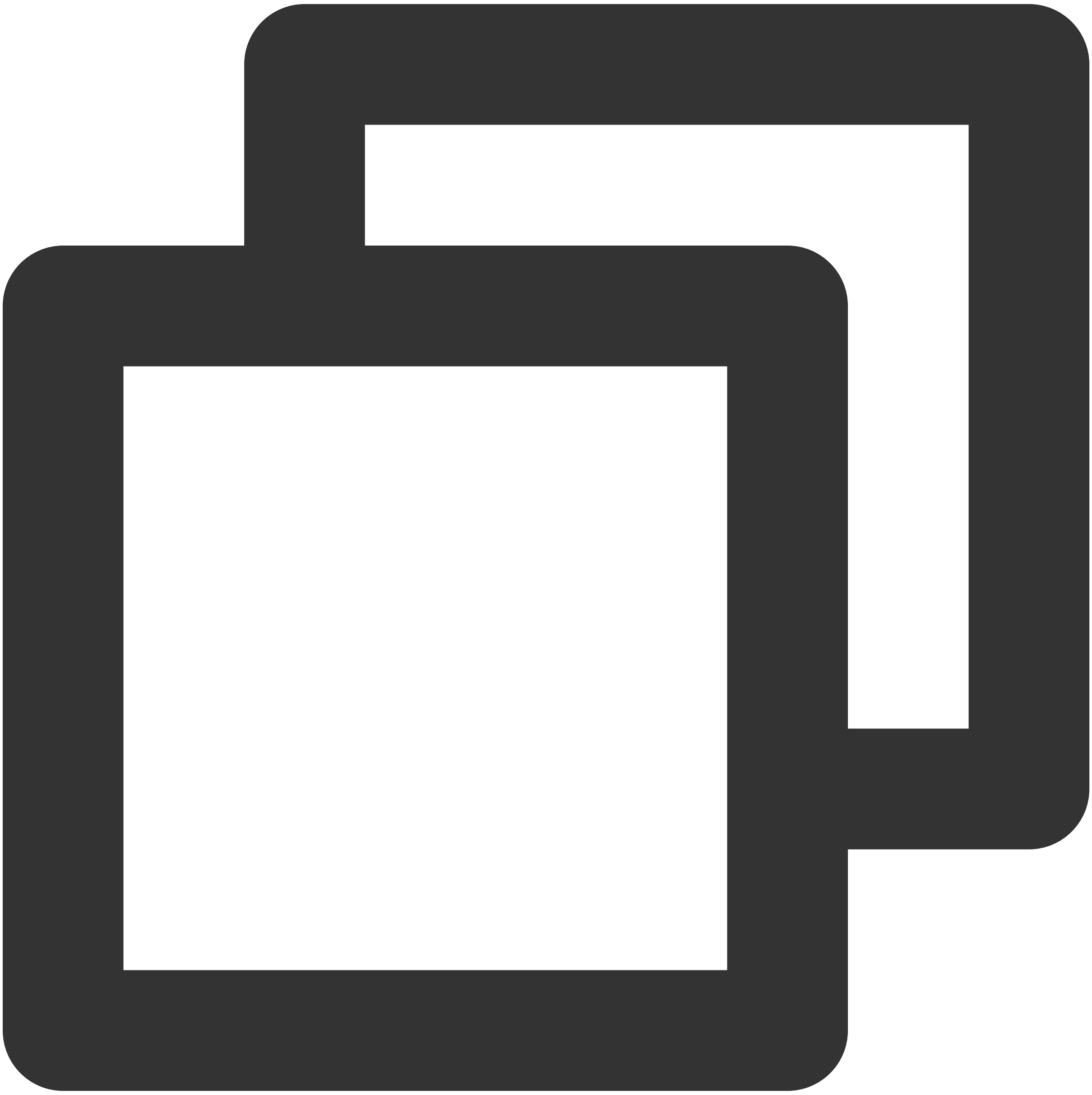
Quality	Name	Description
0	Unknown	Unknown
1	Excellent	The current network is excellent.
2	Good	The current network is good.
3	Poor	The current network is fine.
4	Bad	The current network is poor, and there may be obvious stuttering and delay.
5	VeryBad	The current network is very poor, and TRTC can merely sustain the connection but cannot guarantee the communication quality.
6	Down	The current network cannot meet the minimum requirements of TRTC, and it is impossible to have a normal audio/video call.

You only need to listen for `onNetworkQuality` of TRTC and display the corresponding prompt on the UI.
Android
iOS&Mac
Windows

```
// Listen for the `onNetworkQuality` callback to get the change of the current netw
@Override
public void onNetworkQuality(TRTCCloudDef.TRTCQuality localQuality,
                             ArrayList<TRTCCloudDef.TRTCQuality> remoteQuality)
{
    // Get your local network quality
    switch(localQuality) {
        case TRTCQuality_Unknown:
            Log.d(TAG, "SDK has not yet sensed the current network quality.");
            break;
        case TRTCQuality_Excellent:
```



```
                Log.d(TAG, "The current network is very good.");
                break;
            case TRTCQuality_Good:
                Log.d(TAG, "The current network is good.");
                break;
            case TRTCQuality_Poor:
                Log.d(TAG, "The current network quality barely meets the demand.");
                break;
            case TRTCQuality_Bad:
                Log.d(TAG, "The current network is poor, and there may be significant f
                break;
            case TRTCQuality_VeryBad:
                Log.d(TAG, "The current network is very poor, the communication quality
                break;
            case TRTCQuality_Down:
                Log.d(TAG, "The current network does not meet the minimum requirements.
                break;
            default:
                break;
        }
        // Get the network quality of remote users
        for (TRTCCloudDef.TRTCQuality info : arrayList) {
            Log.d(TAG, "remote user : = " + info.userId + ", quality = " + info.quality
        }
    }
```



```
// Listen for the `onNetworkQuality` callback to get the change of the current netw
- (void)onNetworkQuality:(TRTCQualityInfo *)localQuality remoteQuality:(NSArray<TRT
    // Get your local network quality
    switch(localQuality.quality) {
        case TRTCQuality_Unknown:
            NSLog(@"SDK has not yet sensed the current network quality.");
            break;
        case TRTCQuality_Excellent:
            NSLog(@"The current network is very good.");
            break;
        case TRTCQuality_Good:
```



```
            NSLog(@"The current network is good.");
            break;
        case TRTCQuality_Poor:
            NSLog(@"The current network quality barely meets the demand.");
            break;
        case TRTCQuality_Bad:
            NSLog(@"The current network is poor, and there may be significant freez
            break;
        case TRTCQuality_VeryBad:
           NSLog(@"The current network is very poor, the communication quality cann
            break;
        case TRTCQuality_Down:
            NSLog(@"The current network does not meet the minimum requirements.");
            break;
        default:
            break;
    }
    // Get the network quality of remote users
    for (TRTCQualityInfo *info in arrayList) {
        NSLog(@"remote user : = %@, quality = %@", info.userId, @(info.quality));
    }
}
```

```
// Listen for the `onNetworkQuality` callback to get the change of the current netw
void onNetworkQuality(liteav::TRTCQualityInfo local_quality,
    liteav::TRTCQualityInfo* remote_quality, uint32_t remote_quality_count) {
    // Get your local network quality
    switch (local_quality.quality) {
    case TRTCQuality_Unknown:
        printf("SDK has not yet sensed the current network quality.");
        break;
    case TRTCQuality_Excellent:
        printf("The current network is very good.");
        break;
```

```
    case TRTCQuality_Good:
        printf("The current network is good.");
        break;
    case TRTCQuality_Poor:
        printf("The current network quality barely meets the demand.");
        break;
    case TRTCQuality_Bad:
        printf("The current network is poor, and there may be significant freezes a
        break;
    case TRTCQuality_Vbad:
        printf("The current network is very poor, the communication quality cannot
        break;
    case TRTCQuality_Down:
        printf("The current network does not meet the minimum requirements.");
        break;
    default:
        break;
    }
    // Get the network quality of remote users
    for (int i = 0; i < remote_quality_count; ++i) {
        printf("remote user : = %s, quality = %d", remote_quality[i].userId, remote
    }
}
```


Web

Last updated：2024-05-21 15:05:29

Before entering the room or during the call, you can check the user's network quality to determine the current network quality. If the user's network quality is too poor, it is recommended that the user change the network environment to ensure normal call quality.

This article mainly introduces how to implement network quality detection before the call or during the call based on the `NETWORK_QUALITY` event.

Detect network quality during the call

```
const trtc = TRTC.create();
trtc.on(TRTC.EVENT.NETWORK_QUALITY, event => {
   console.log(`network-quality, uplinkNetworkQuality:${event.uplinkNetworkQuality}
   console.log(`uplink rtt:${event.uplinkRTT} loss:${event.uplinkLoss}`)
   console.log(`downlink rtt:${event.downlinkRTT} loss:${event.downlinkLoss}`)
})
```

Detect network quality before the call

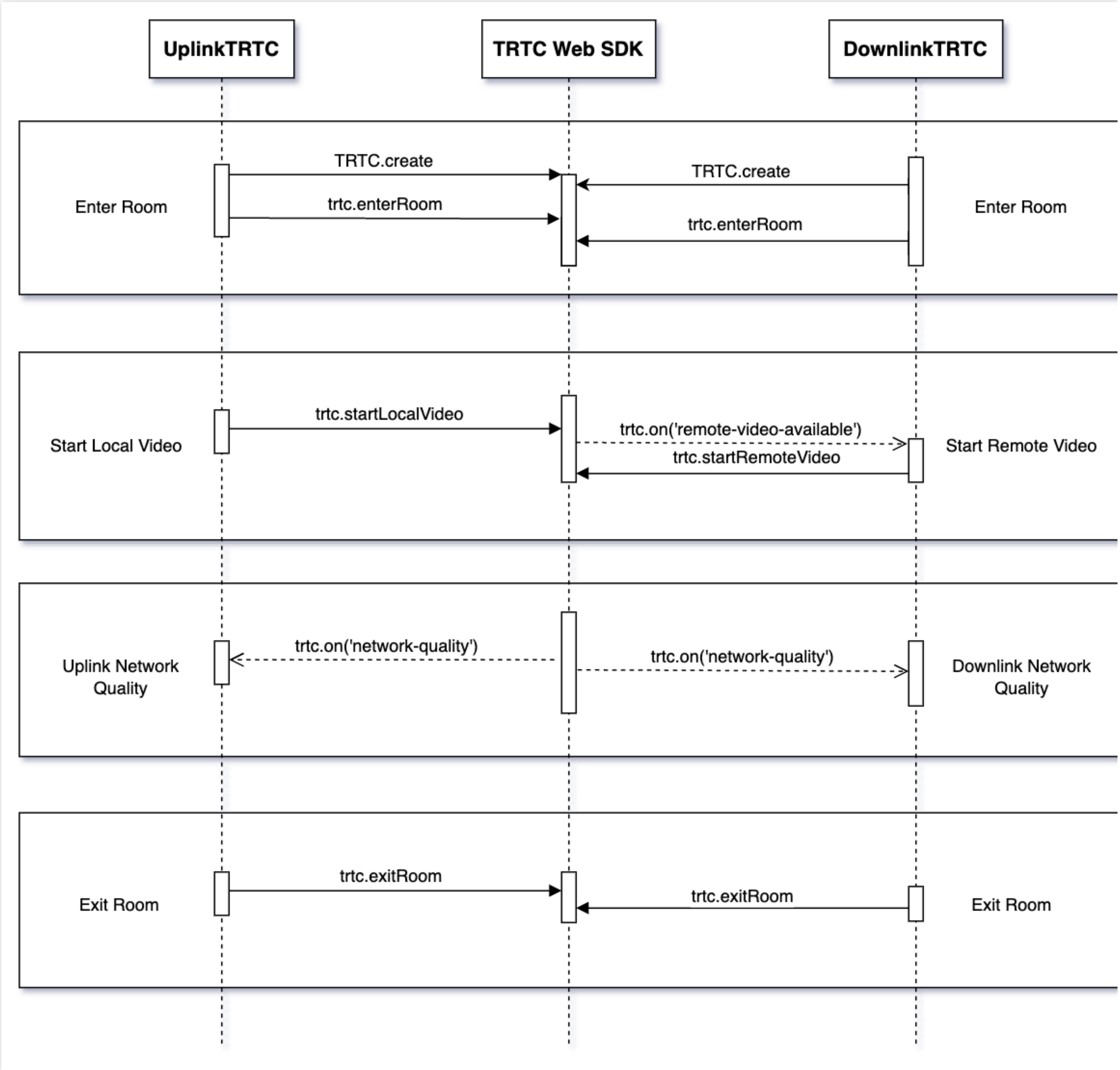
Implementation process

1. Call `TRTC.create()` to create two TRTCs, referred to as uplinkTRTC and downlinkTRTC.
2. Both TRTCs enter the same room.
3. Use uplinkTRTC to push the stream, and listen to the `NETWORK_QUALITY` event to detect the uplink network quality.
4. Use downlinkTRTC to pull the stream, and listen to the `NETWORK_QUALITY` event to detect the downlink network quality.
5. The entire process lasts for about 15 seconds, and finally takes the average network quality to roughly determine the uplink and downlink network conditions.

Note:

The process of checking network quality incurs a small basic service fee. If a resolution is not specified, the stream will be published at a resolution of 640 x 480.

API Call Sequence


UplinkTRTC
TRTC Web SDK
DownlinkTRTC
Enter Room
TRTC.create
trtc.enterRoom
TRTC.create
trtc.enterRoom
Enter Room
Start Local Video
trtc.startLocalVideo
trtc.on('remote-video-available')
trtc.startRemoteVideo
Start Remote Video
Uplink Network Quality
trtc.on('network-quality')
trtc.on('network-quality')
Downlink Network Quality
Exit Room
trtc.exitRoom
trtc.exitRoom
Exit Room

Sample Code

```
let uplinkTRTC = null; // Used to detect uplink network quality
let downlinkTRTC = null; // Used to detect downlink network quality
let localStream = null; // Stream used for testing
let testResult = {
  // Record uplink network quality data
  uplinkNetworkQualities: [],
  // Record downlink network quality data
  downlinkNetworkQualities: [],
  average: {
    uplinkNetworkQuality: 0,
```

```
    downlinkNetworkQuality: 0
  }
}

// 1. Test uplink network quality
async function testUplinkNetworkQuality() {
  uplinkTRTC = TRTC.create();

  uplinkTRTC.enterRoom({
    roomId: 8080,
    sdkAppId: 0, // Fill in sdkAppId
    userId: 'user_uplink_test',
    userSig: '', // userSig of uplink_test
    scene: 'rtc'
  })

  uplinkTRTC.on(TRTC.EVENT.NETWORK_QUALITY, event => {
    const { uplinkNetworkQuality } = event;
    testResult.uplinkNetworkQualities.push(uplinkNetworkQuality);
  });

}

// 2. Detect downlink network quality
async function testDownlinkNetworkQuality() {
  downlinkTRTC = TRTC.create();
  downlinkTRTC.enterRoom({
    roomId: 8080,
    sdkAppId: 0, // Fill in sdkAppId
    userId: 'user_downlink_test',
    userSig: '', // userSig
    scene: 'rtc'
  });

  downlinkTRTC.on(TRTC.EVENT.NETWORK_QUALITY, event => {
      const { downlinkNetworkQuality } = event;
      testResult.downlinkNetworkQualities.push(downlinkNetworkQuality);
  })
}

// 3. Start detection
testUplinkNetworkQuality();
testDownlinkNetworkQuality();

// 4. Stop detection after 15s and calculate the average network quality
setTimeout(() => {
  // Calculate the average uplink network quality
```



```
    if (testResult.uplinkNetworkQualities.length > 0) {
      testResult.average.uplinkNetworkQuality = Math.ceil(
        testResult.uplinkNetworkQualities.reduce((value, current) => value + current,
      );
    }

    if (testResult.downlinkNetworkQualities.length > 0) {
      // Calculate the average downlink network quality
      testResult.average.downlinkNetworkQuality = Math.ceil(
        testResult.downlinkNetworkQualities.reduce((value, current) => value + curren
      );
    }

    // Detection is over, clean up related states.
    uplinkTRTC.exitRoom();
    downlinkTRTC.exitRoom();
  }, 15 * 1000);
```

Result Analysis

After the above steps, you can get the average uplink network quality and the average downlink network quality. The enumeration values of network quality are as follows:

Value	Meaning
0	The network condition is unknown, indicating that the current TRTC instance has not established an uplink/downlink connection
1	The network condition is excellent
2	The network condition is good
3	The network condition is average
4	The network condition is poor
5	The network condition is extremely poor
6	The network connection has been disconnected. Note: If the downlink network quality is this value, it means that all downlink connections have been disconnected.

Suggestion:

When the network quality is greater than 3, it is recommended to guide the user to check the network and try to change the network environment, otherwise it is difficult to ensure normal audio and video call. You can also reduce bandwidth consumption through the following strategies:

If the uplink network quality is greater than 3, you can reduce the bitrate through the `TRTC.updateLocalVideo()` interface or close the video through the `TRTC.stopLocalVideo()` method to reduce uplink bandwidth consumption.

If the downlink network quality is greater than 3, you can reduce the downlink bandwidth consumption by subscribing to a small stream (refer to: Enable Small Stream Transmission) or only subscribing to audio.

Electron

Last updated：2023-09-28 11:41:21

This document describes how to assess the current network conditions.

When you have a video call on WeChat under poor network conditions (for example, when you are in an elevator), WeChat displays a message indicating the current network quality is poor. This document describes how to use TRTC to implement a similar interaction in your own application.

Call Guide

TRTC provides the **onNetworkQuality** callback to report the current network quality once every two seconds. It contains two parameters: `localQuality` and `remoteQuality` .
localQuality indicates your current network quality, which has six levels: `Excellent` , `Good` , `Poor` , `Bad` , `VeryBad` , and `Down` .
remoteQuality is an array indicating the network quality of remote users. In this array, each element represents the network quality of a remote user.

Quality	Name	Description
0	Unknown	Unknown
1	Excellent	The current network is excellent.
2	Good	The current network is good.
3	Poor	The current network is fine.
4	Bad	The current network is poor. There may be obvious stuttering and delay.
5	VeryBad	The current network is very poor. TRTC can sustain the connection but cannot guarantee the communication quality.
6	Down	The current network cannot meet the minimum requirements of TRTC, and it is impossible to have a normal audio/video call.

You only need to listen for onNetworkQuality of TRTC and display the corresponding prompt on the UI.

```
import TRTCCloud, { TRTCQuality } from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();

function onNetworkQuality(localQuality, remoteQuality) {
  switch(localQuality.quality) {
    case TRTCQuality.TRTCQuality_Unknown:
      console.log('SDK has not yet sensed the current network quality.');
      break;
    case TRTCQuality.TRTCQuality_Excellent:
      console.log('The current network is very good.');
      break;
```

```
    case TRTCQuality.TRTCQuality_Good:
      console.log('The current network is good.');
      break;
    case TRTCQuality.TRTCQuality_Poor:
      console.log('The current network quality barely meets the demand.');
      break;
    case TRTCQuality.TRTCQuality_Bad:
      console.log('The current network is poor, and there may be significant freeze
      break;
    case TRTCQuality.TRTCQuality_Vbad:
      console.log('The current network is very poor, the communication quality cann
      break;
    case TRTCQuality.TRTCQuality_Down:
      console.log('The current network does not meet the minimum requirements.');
      break;
    default:
      break;
  }
  for (let i = 0; i < remoteQuality.length; i++) {
    console.log(`remote user: ${remoteQuality[i].userId}, quality: ${remoteQuality[
  }
}

rtcCloud.on('onNetworkQuality', onNetworkQuality);
```


Flutter

Last updated：2024-02-02 18:47:09

This document primarily discusses how to perceive the quality of the current network.
When engaging in a video call using WeChat, if we encounter inferior network conditions (such as after entering an elevator), WeChat will prompt on the video call interface that "Your current network quality is poor". This document primarily explores how to accomplish the same interaction using TRTC.

Call Guidelines

TRTC offers a callback event known as **onNetworkQuality**, which reports the current network quality to you every two seconds. Its parameters include two parts: localQuality and remoteQuality

localQuality: Represents your current network quality, divided into 6 levels, which are Excellent, Good, Poor, Bad, VeryBad, and Down.
remoteQuality: Represents the network quality of remote users. This is an array, each Element (XML) in the array represents the network quality of a remote user.

Quality	Name	Description
1	Unknown	Unperceived
1	Excellent	The present network is exceedingly good
2	Good	The current network is fairly good
3	Poor	Current network is average
4	Bad	Present network quality is poor, it might cause noticeable stutters and communication delays
5	VeryBad	The current network conditions are abysmal, TRTC can barely maintain a connection, yet it can't guarantee the quality of communication
6	Down	The current network does not meet the minimum requirements of TRTC, obstructing the normal audio and video conversation

All you need is to monitor TRTC's onNetworkQuality and make corresponding prompts on the interface:

```
// Monitor the onNetworkQuality callback and perceive the alterations in the curren
if (type == TRTCCloudListener.onNetworkQuality) {
    if (type == TRTCCloudDef.TRTC_QUALITY_UNKNOWN) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Excellent) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Good) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Poor) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Bad) {
```

```
        // TODO
      } else if (type == TRTCCloudDef.TRTC_QUALITY_Vbad) {
        // TODO
      } else if (type == TRTCCloudDef.TRTC_QUALITY_Down) {
        // TODO
      }
      // Get the network quality of remote users
      for (var info in param['remoteQuality']) {
          // TODO
      }
  }
```


Enabling Screen Sharing
iOS

Last updated：2023-09-28 11:41:53

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 later. 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
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

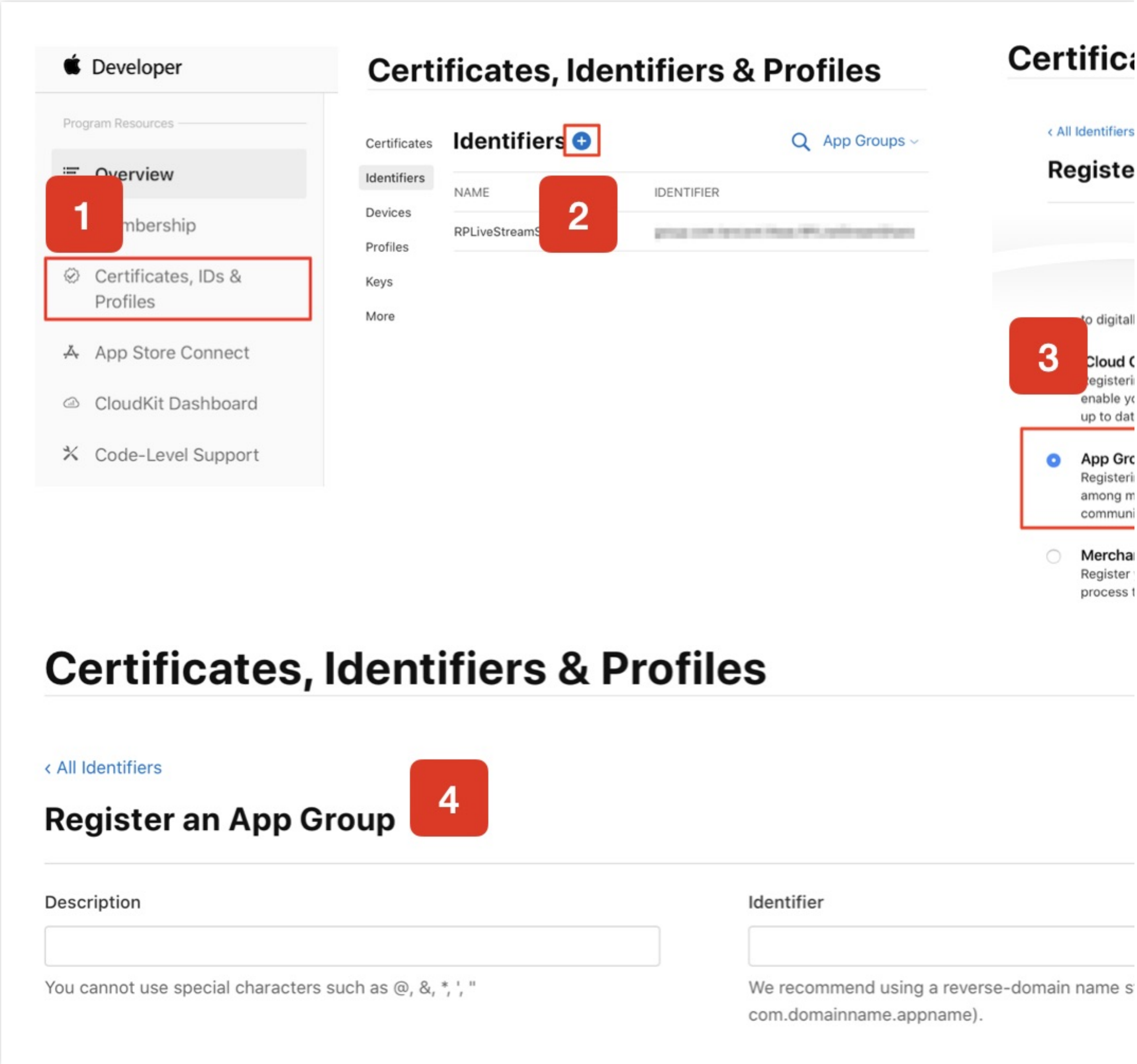
To enable cross-app screen sharing on iOS, you need to add the screen recording process **Broadcast Upload Extension**, which works with the host app to push streams. A Broadcast Upload Extension is created by the system when screen sharing is requested and is responsible for receiving the screen images 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 import into it `TXLiteAVSDK_ReplayKitExt.framework` from the SDK package.
3. Put the host app on standby to receive screen recording data from the **Broadcast Upload Extension**.

notice

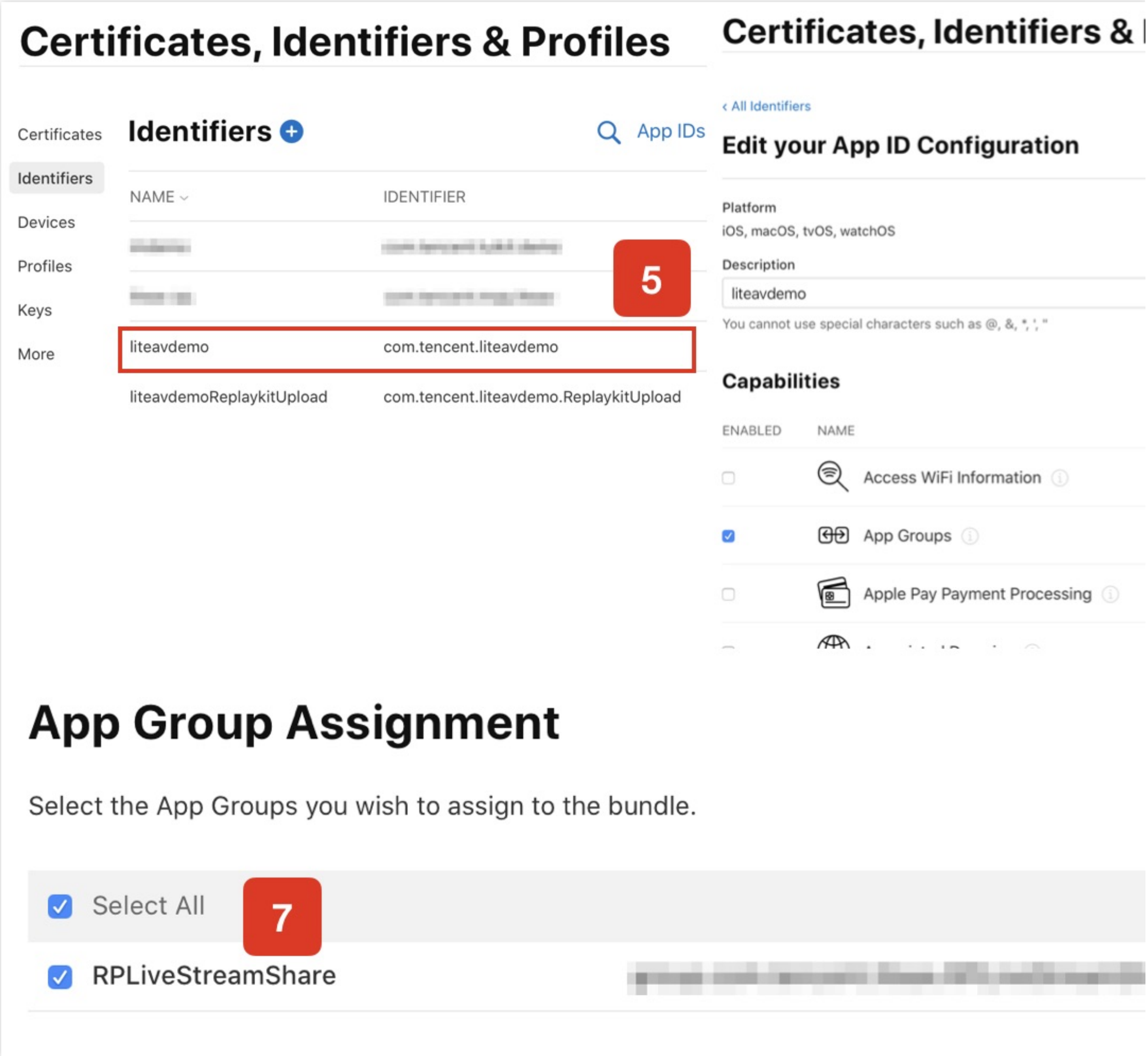
If you skip step 1, that is, if you do not configure an App Group (passing in `nil` to the API), you can still enable screen sharing, but its stability will be compromised. We suggest you configure an App Group as described below.

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. Fill in the **Description** and **Identifier** boxes. For **Identifier**, type the `AppGroup` value passed in to the API. Click **Continue**.



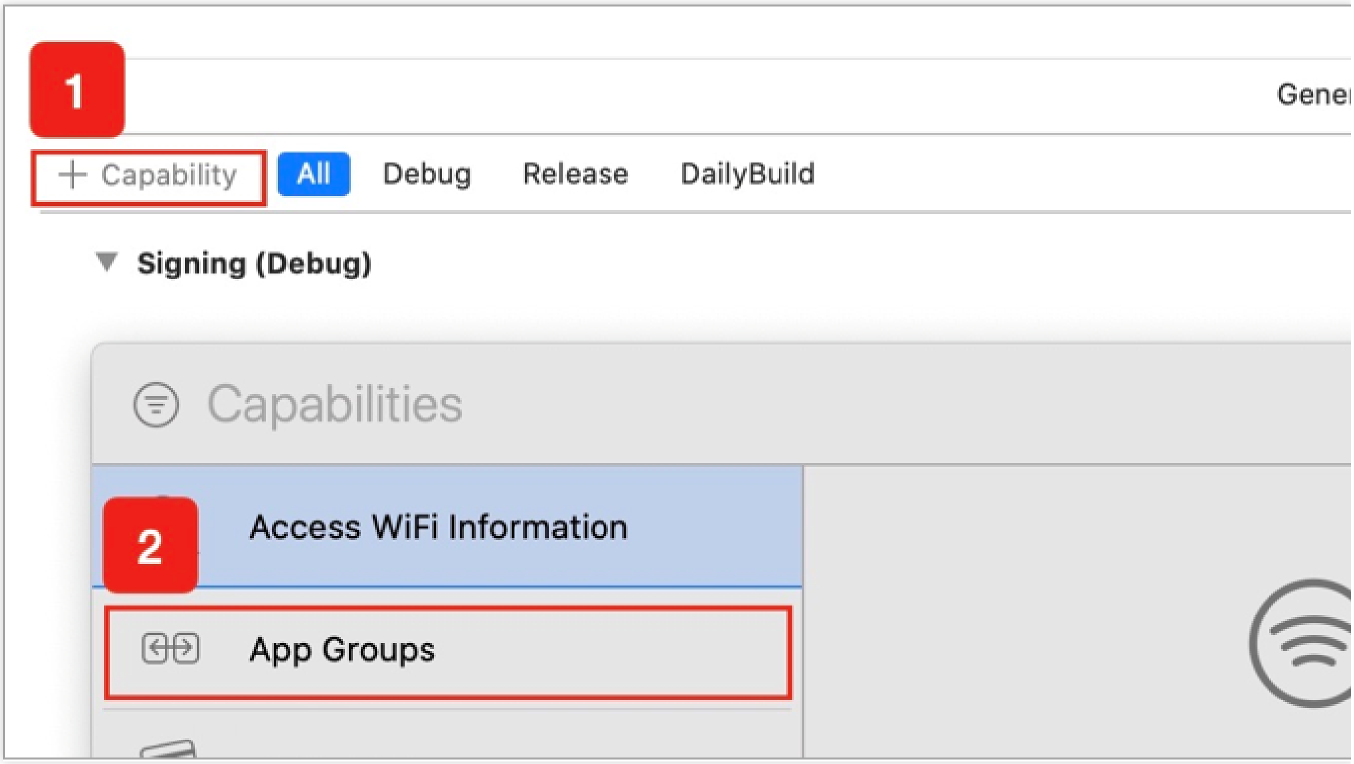
5. Select **Identifiers** on the top left sidebar, and click your App ID (you need to configure the App ID for the host app and extension in the same way).
6. Select **App Groups** and click **Edit**.
7. Select the App Group you created, click **Continue** to return to the edit page, and click **Save**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



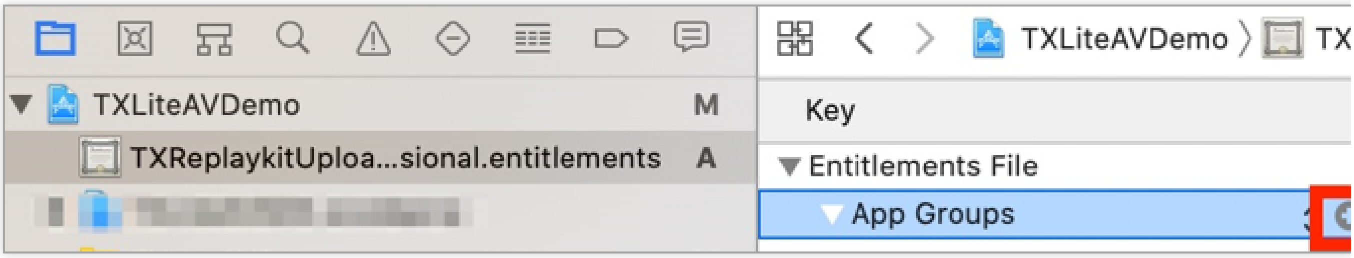
8. Download the provisioning profile again and import it to Xcode.

Step 2. Create a Broadcast Upload Extension

1. Click **File** > **New** > **Target...** in the Xcode menu and select **Broadcast Upload Extension**.
2. In the dialog box that pops up, enter the information required. You **don't** need to select **Include UI Extension**. After entering the required information, click **Finish**.
3. Drag `TXLiteAVSDK_ReplayKitExt.framework` in the SDK package into the project and select the target created.
4. Select the target you created, click **+ Capability**, and double-click **App Groups**.



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 ID of the App Group created earlier.
```

```
- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject *> *)setupI
    [[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)
{
    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:(RPSampleBuffe
    switch (sampleBufferType) {
        case RPSampleBufferTypeVideo:
            [[TXReplayKitExt sharedInstance] sendVideoSampleBuffer:sampleBuffer];
            break;
        case RPSampleBufferTypeAudioApp:
```

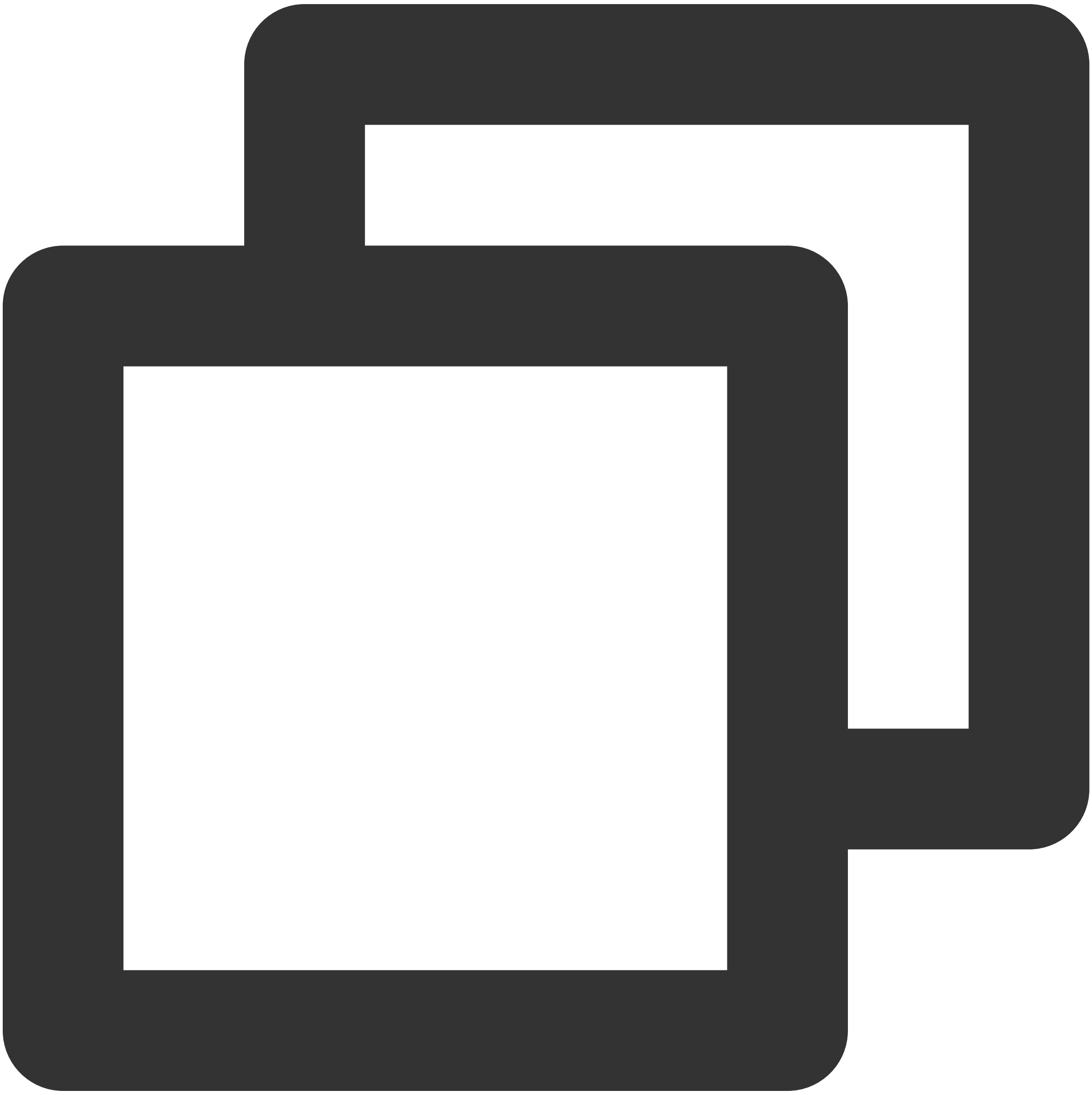
```
            // Handle audio sample buffer for app audio
            break;
        case RPSampleBufferTypeAudioMic:
            // Handle audio sample buffer for mic audio
            break;

        default:
            break;
    }
}
@end
```

Step 3. Make the host app wait to receive data

Before screen sharing starts, the host application must be put on standby to receive screen recording data from the Broadcast Upload Extension. To do 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 ID of the App Grou
- (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 throug
- (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. The TRTCBroadcastExtensionLauncher file in the Demo implements screen sharing. Find 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];
}
```

notice

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.
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 the substream, and other users in the room will be notified via onUserSubStreamAvailable in `TRTCCloudDelegate` .
Users who want to watch the shared screen can start rendering the substream 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 via 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：2023-09-28 11:42:23

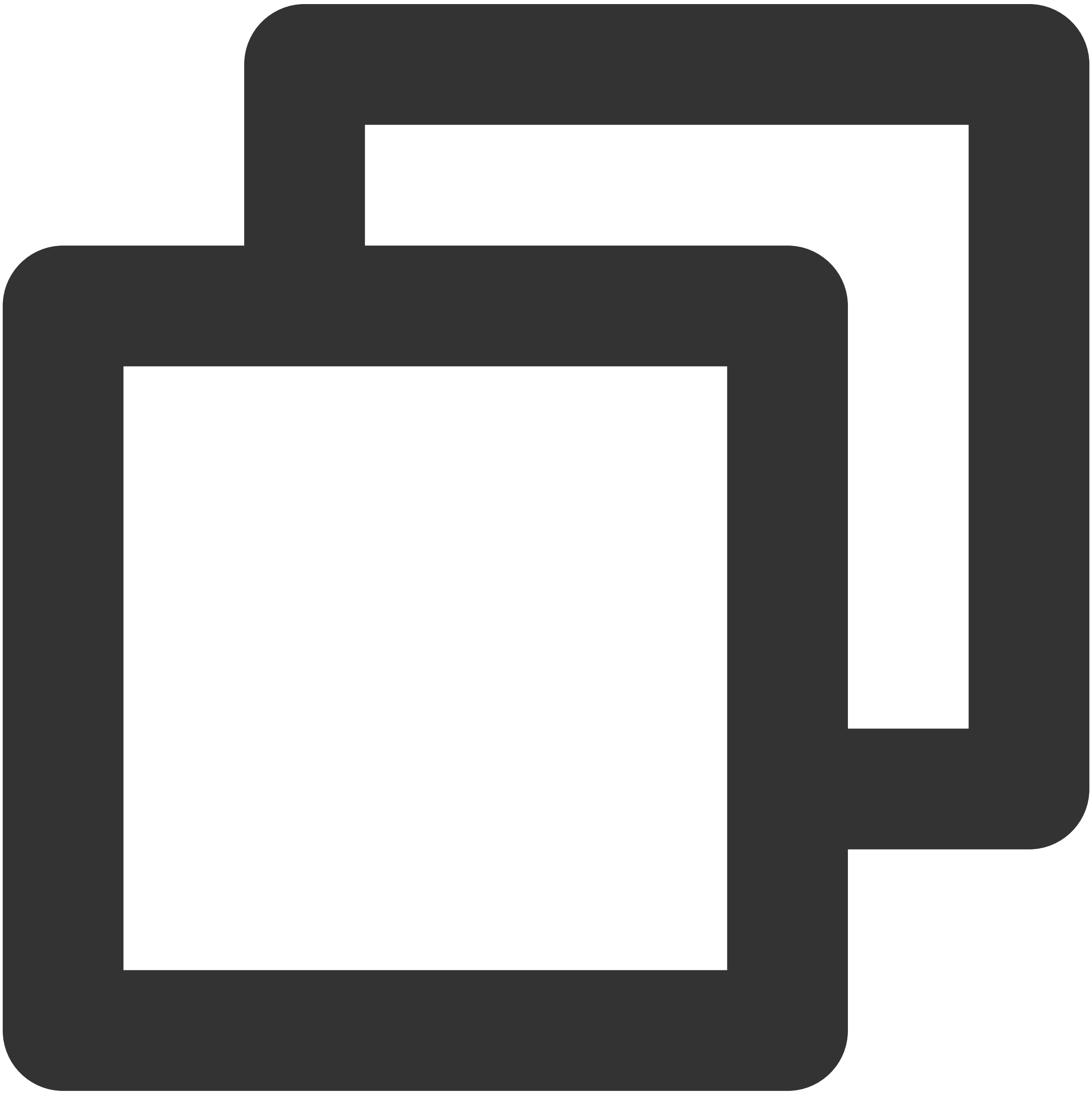
This document describes how to share the screen. Currently, a TRTC room can have only one screen sharing stream at a time.

Call Guide

Enabling screen sharing

Step 1. Add 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$TXScreenCaptureAssistantAc
    android:theme="@android:style/Theme.Translucent"/>
```

Step 2. Start screen sharing

To start screen sharing on Android, simply call the startScreenCapture() API in `TRTCCloud` .

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
Frame rate	videoFps	10 fps	8 fps
Highest bitrate	videoBitrate	1600 Kbps	2000 Kbps
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.

Step 3. Display a floating window to avoid the application being closed (optional)

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

The code in FloatingView.java offers an example of how to create a mini floating window:


```
public void showView(View view, int width, int height) {
        mWindowManager = (WindowManager) mContext.getSystemService(Context.WINDOW_S
        int type = WindowManager.LayoutParams.TYPE_TOAST;
        // `TYPE_TOAST` applies only to Android 4.4 and later. On earlier versions,
        // Android 7.1 and later set restrictions on `TYPE_TOAST`.
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            type = WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY;
        } else 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);
}
```

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 `TRTCCloudListener` .

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 view the shared screen can start rendering the primary stream of the remote user by calling the startRemoteView API.

macOS

Last updated：2023-09-28 11:42:50

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
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 `Screen0` , `Screen1` , and so on.
extInfo	NSDictionary	Extra information
Thumbnail	NSImage	Window thumbnail
Icon	NSImage	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}.
explain
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

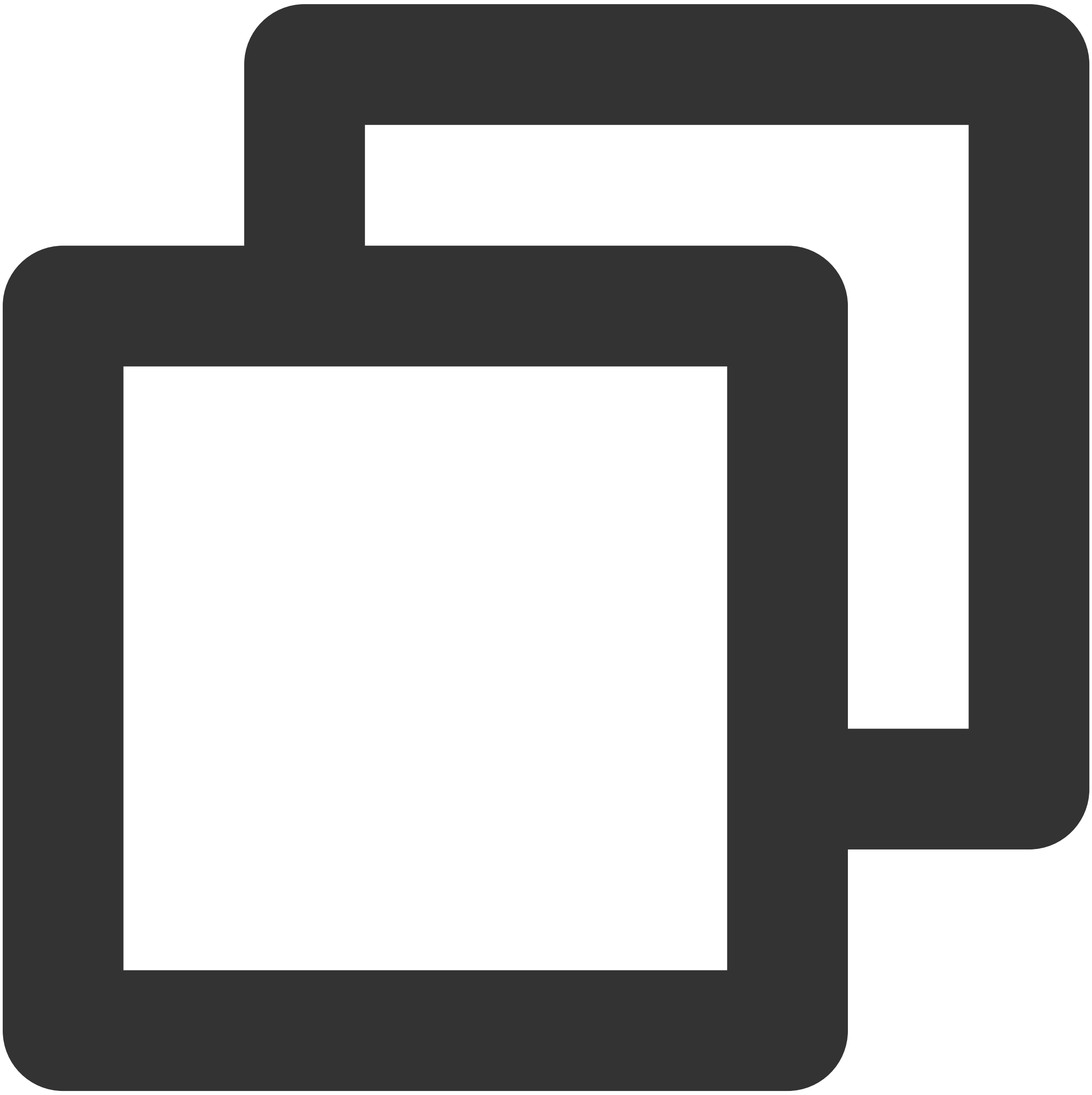
When a macOS/Windows user in a room starts screen sharing, the screen will be shared through a substream, and other users in the room will be notified through onUserSubStreamAvailable in `TRTCCloudDelegate` .

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.capturePreviewWin
    } 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

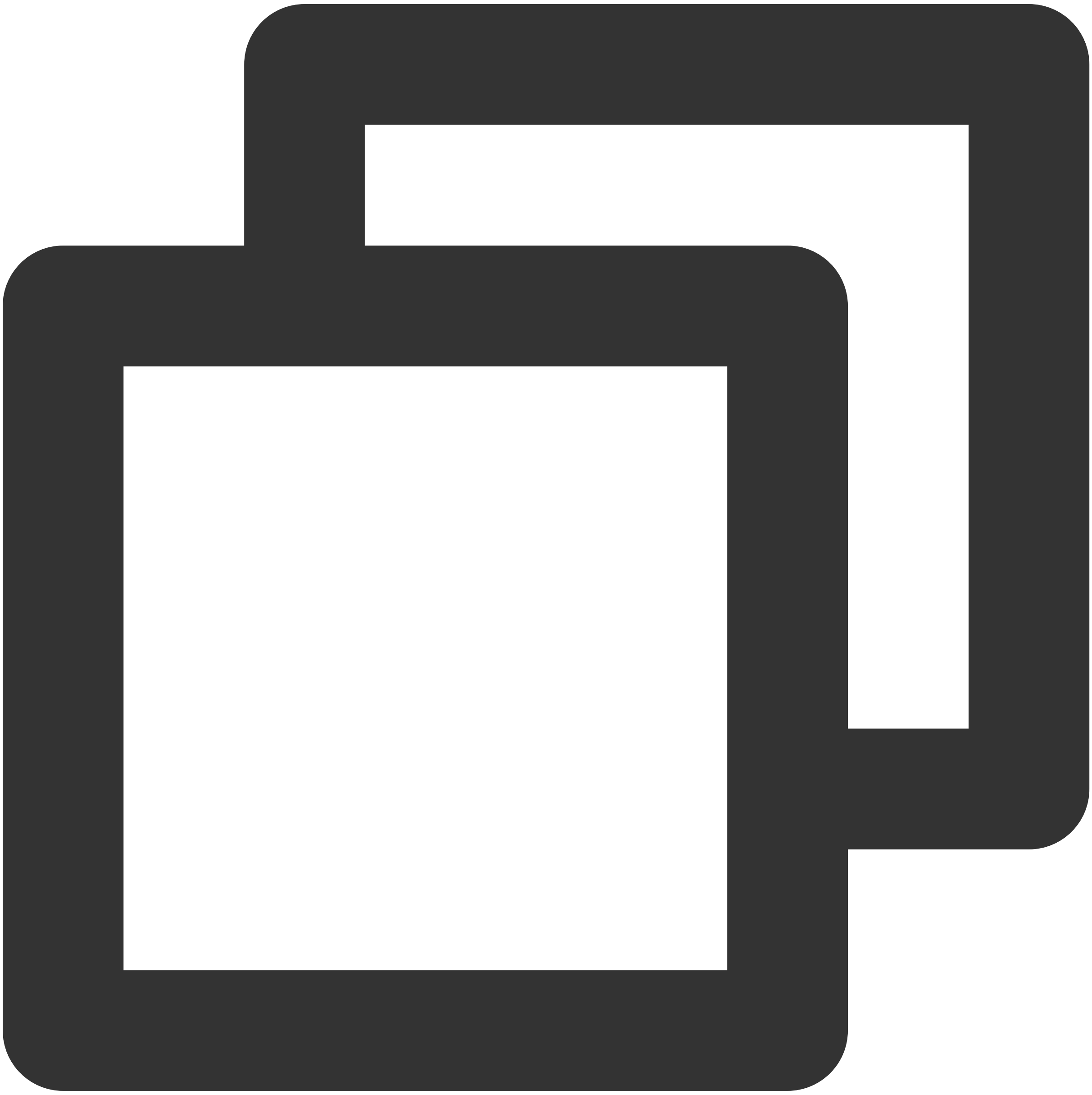
Last updated：2023-10-30 11:11:57

Function Description

This article mainly introduces how to implement screen sharing in TRTC Web SDK.

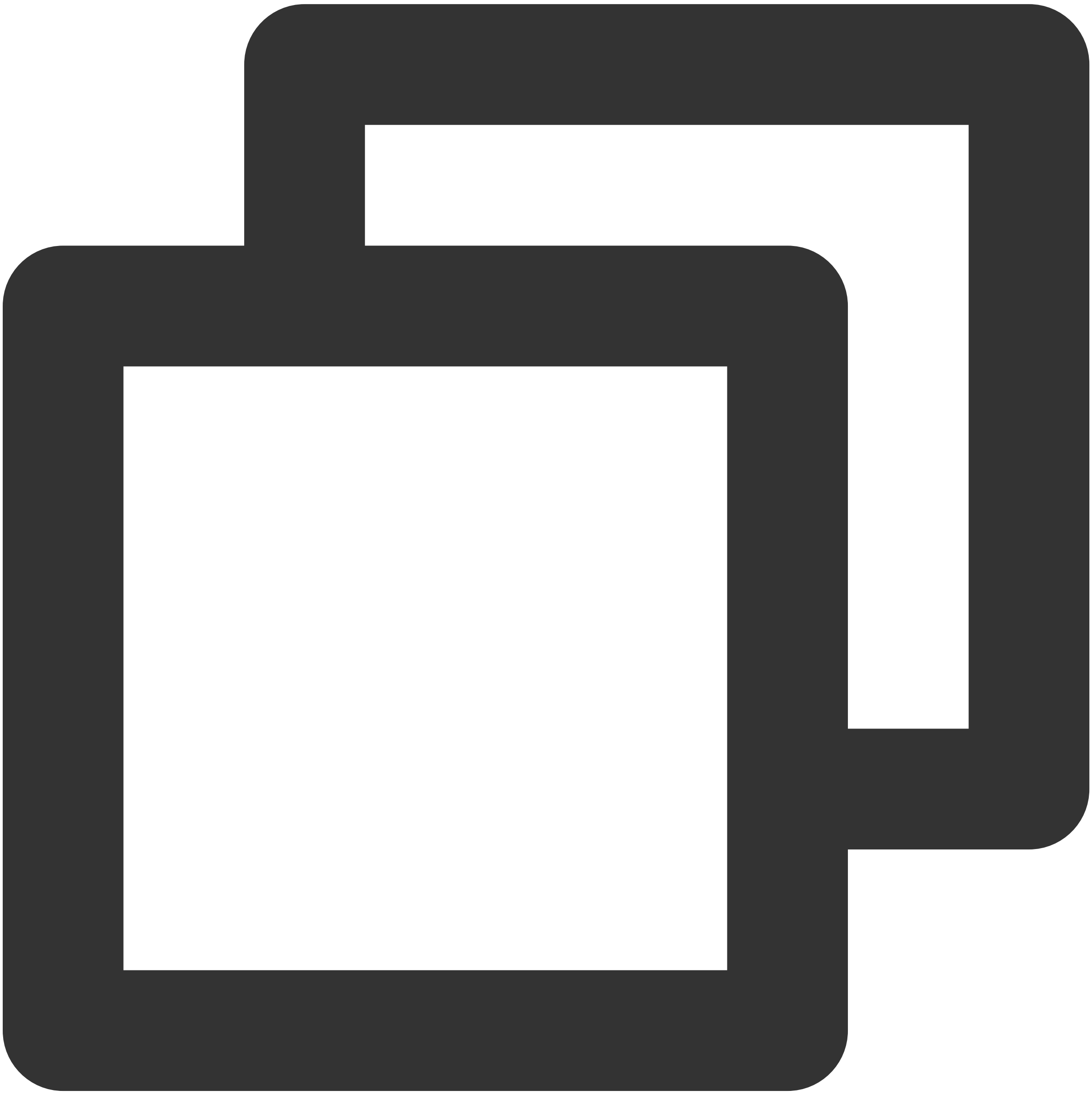
Implementation Process

1. Start Local Screen Sharing



```
const trtcA = TRTC.create();
await trtcA.enterRoom({
  scene: 'rtc',
  sdkAppId: 140000000, // Fill in your sdkAppId
  userId: 'userA', // Fill in your userId
  userSig: 'userA_sig', // Fill in userSig corresponding to userId
  roomId: 6969
})
await trtcA.startScreenShare();
```

2. Play Remote Screen Sharing

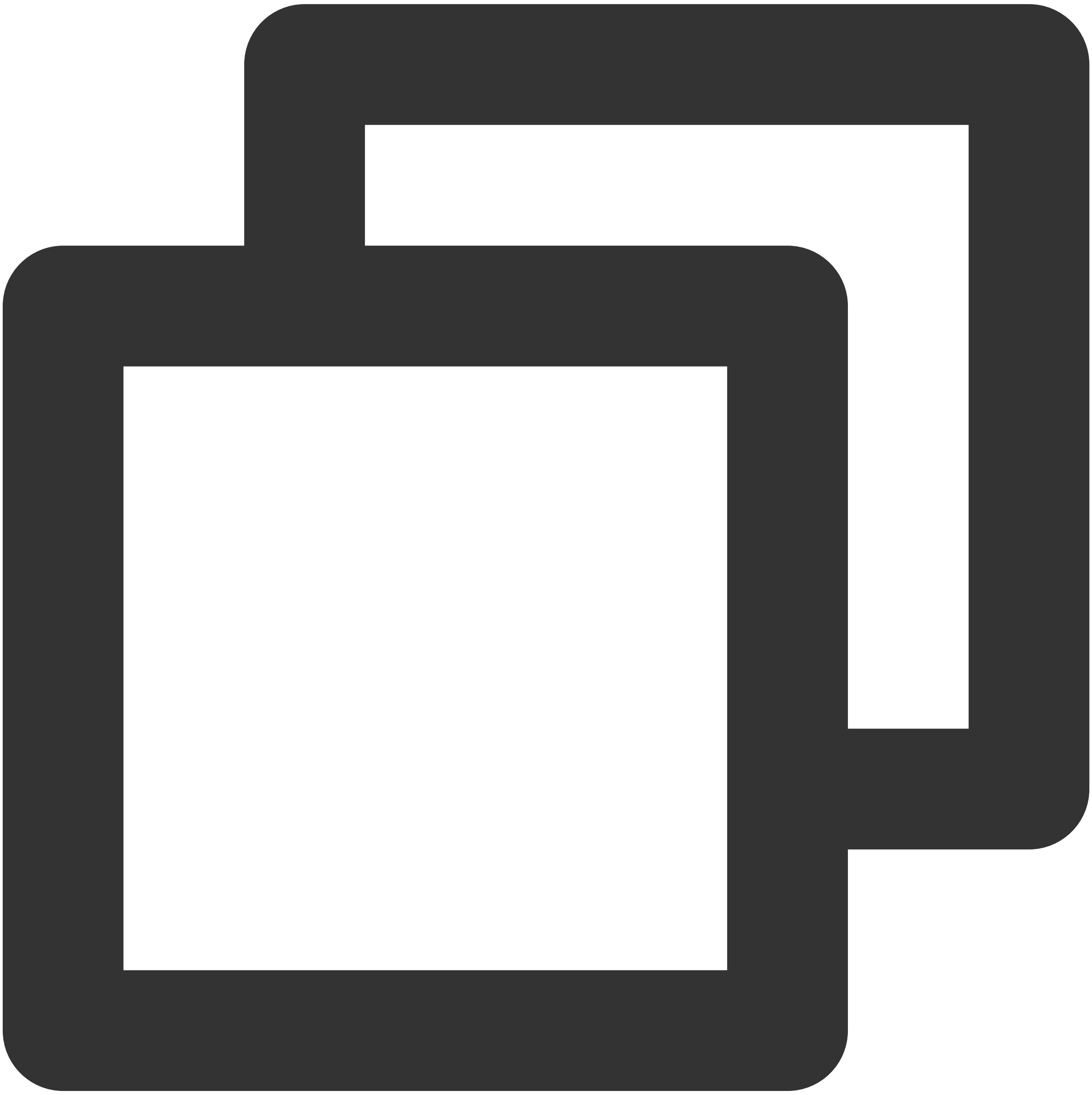


```
const trtcB = TRTC.create();
trtcB.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, ({ userId, streamType }) => {
  // Main video stream, generally the stream pushed by the camera
  if (streamType === TRTC.TYPE.STREAM_TYPE_MAIN) {
    // 1. Place a div tag with an id of `${userId}_main` on the page to play the ma
    // 2. Play the main video stream
    trtcB.startRemoteVideo({ userId, streamType,  view: `${userId}_main` });
  } else {
    // Sub video stream, generally the stream pushed by screen sharing.
    // 1. Place a div tag with an id of `${userId}_screen` on the page to play the
    // 2. Play screen sharing
```

```
    trtcB.startRemoteVideo({ userId, streamType, view: `${userId}_screen` });
  }
});

await trtcB.enterRoom({
  scene: 'rtc',
  sdkAppId: 140000000, // Fill in your sdkAppId
  userId: 'userB', // Fill in your userId
  userSig: 'userB_sig', // Fill in userSig corresponding to userId
  roomId: 6969
})
```

3. Start Camera + Screen Sharing at the Same Time



```
await trtcA.startLocalVideo();
await trtcA.startScreenShare();
```

4. Screen Sharing + System Audio

System audio is supported by Chrome M74+

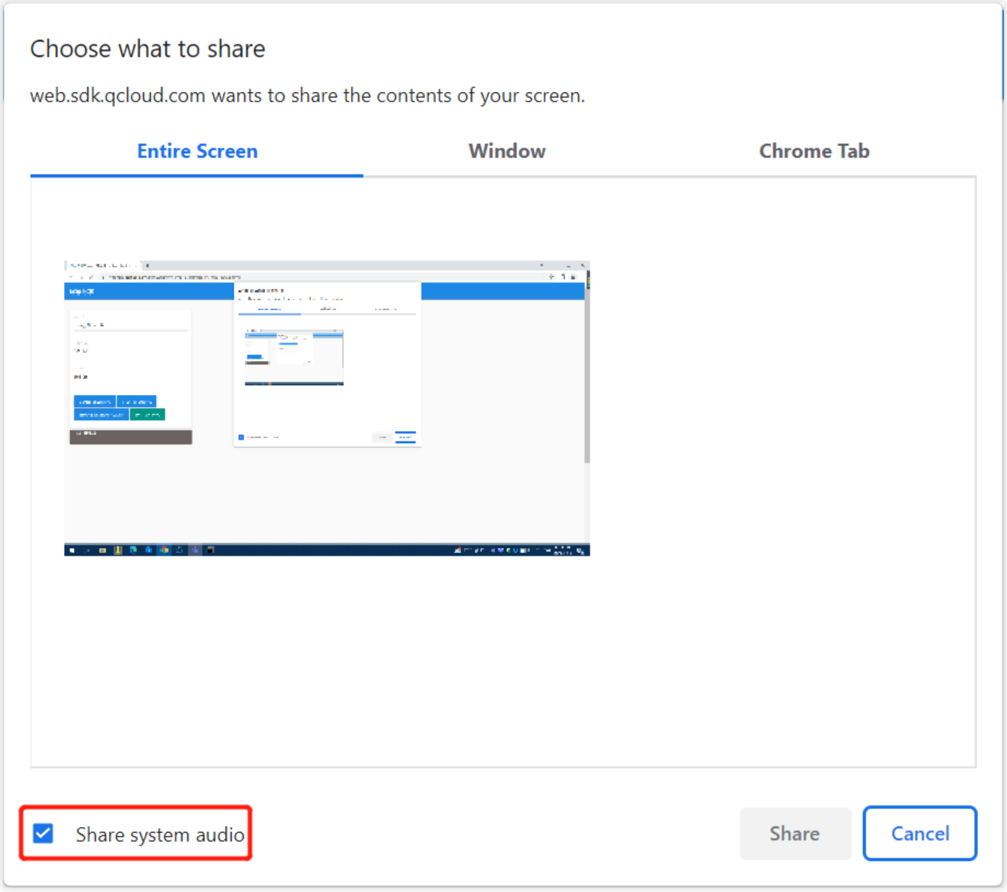
On Windows and Chrome OS, the audio of the entire system can be collected.

On Linux and Mac, only the audio of a certain page can be collected.

Other Chrome versions, other systems, and other browsers are not supported.

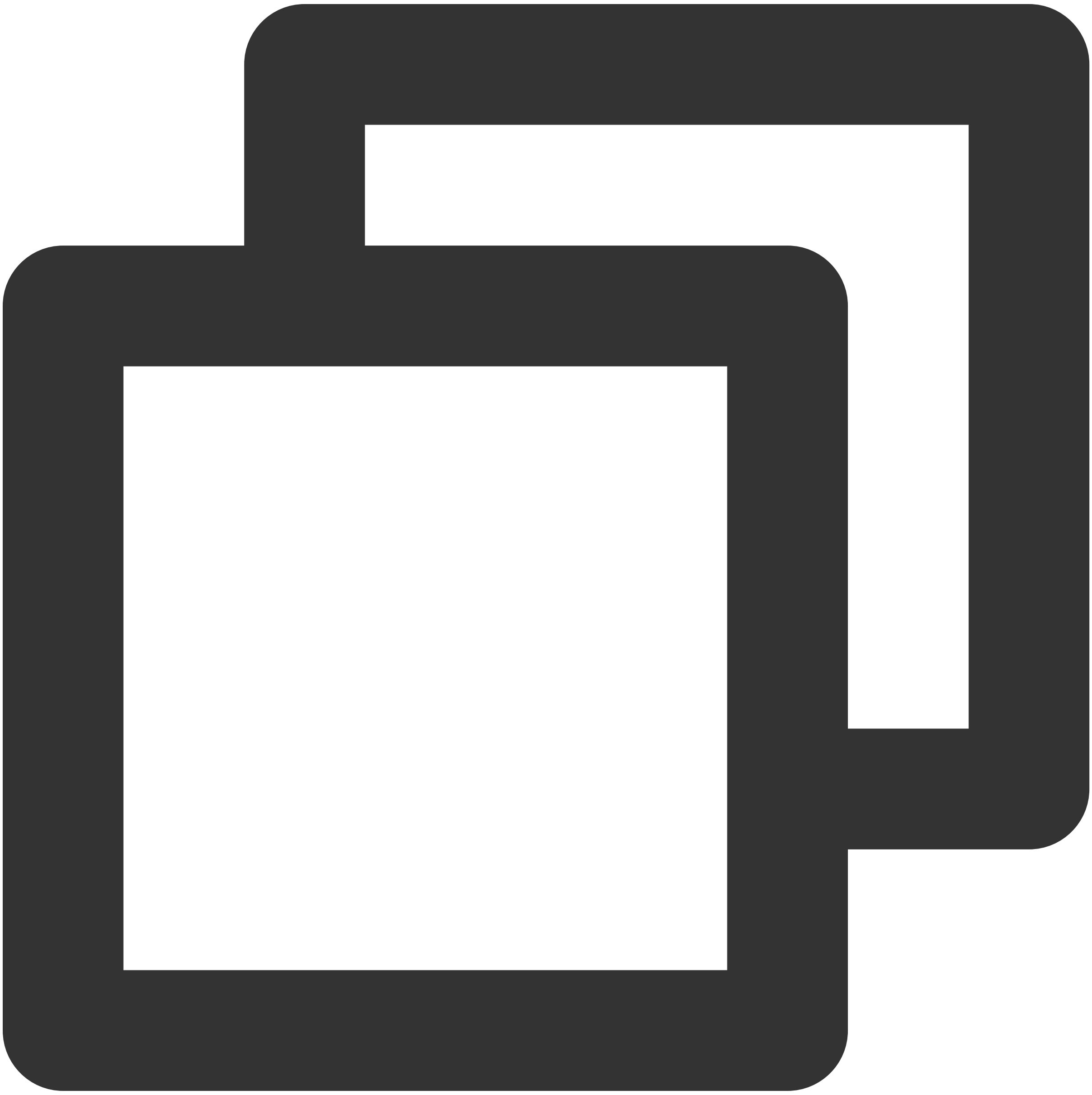
```
await trtcA.startScreenShare({ option: { systemAudio: true }});
```

Check `Share sytem audio` in the pop-up dialog box, and the system audio will be mixed with the local microphone and published. Other users in the room will receive the REMOTE_AUDIO_AVAILABLE event.



5. Stop Screen Sharing

```
})
```

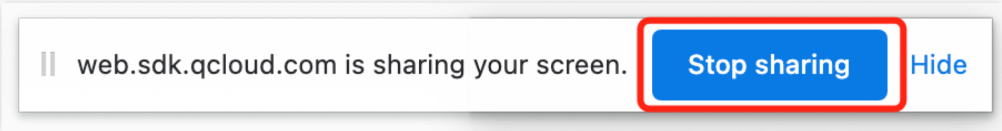


```
// Stop screen sharing collection and publishing
await trtcA.stopScreenShare();

// Other users in the room will receive the TRTC.EVENT.REMOTE_VIDEO_UNAVAILABLE ev
trtcB.on(TRTC.EVENT.REMOTE_VIDEO_UNAVAILABLE, ({ userId, streamType }) => {
   if (streamType === TRTC.TYPE.STREAM_TYPE_SUB) {

   }
})
```

In addition, users may also stop screen sharing through the browser's own button, so the screen sharing stream needs to listen for the screen sharing stop event and respond accordingly.




```
// Listen for screen sharing stop event
```



```
trtcA.on(TRTC.EVENT.SCREEN_SHARE_STOPPED, () => {
  console.log('screen sharing was stopped');
});
```

Precautions

1. What is the main/sub video stream?
2. The SDK uses the `1080p` parameter configuration by default to collect screen sharing. For details, refer to the interface: startScreenShare

Common Issues

1. Safari screen sharing error `getDisplayMedia must be called from a user gesture handler`

This is because Safari restricts the `getDisplayMedia` screen capture interface, which must be called within 1 second of the callback function of the user click event.

Reference: webkit issue.


```
// good
async function onClick() {
  // It is recommended to execute the collection logic first when onClick is execut
  await trtcA.startScreenShare();
  await trtcA.enterRoom({
    roomId: 123123,
    sdkAppId: 140000000, // Fill in your sdkAppId
    userId: 'userA', // Fill in your userId
    userSig: 'userA_sig', // Fill in userSig corresponding to userId
  });
}
```

```
// bad
async function onClick() {
  await trtcA.enterRoom({
    roomId: 123123,
    sdkAppId: 140000000, // Fill in your sdkAppId
    userId: 'userA', // Fill in your userId
    userSig: 'userA_sig', // Fill in userSig corresponding to userId
  });
  // Entering the room may take more than 1s, and the collection may fail
  await trtcA.startScreenShare();
}
```

2. Mac Chrome screen sharing fails with the error message "NotAllowedError: Permission denied by system" or "NotReadableError: Could not start video source" when screen recording is already authorized. Chrome bug. Solution: Open 【Settings】> Click 【Security & Privacy】> Click 【Privacy】> Click 【Screen Recording】> Turn off Chrome screen recording authorization > Reopen Chrome screen recording authorization > Close Chrome browser > Reopen Chrome browser.
3. WebRTC screen sharing known issues and workarounds

Windows

Last updated：2023-09-28 11:43:41

This document describes how to share the screen. Currently, a TRTC room can have only one screen sharing stream at a time.

TRTC supports screen sharing in primary stream and substream modes on Windows.

Substream sharing

In TRTC, you can share the screen via a dedicated stream 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.

Primary stream sharing

In TRTC, the image from the user's camera is published via the primary stream (**bigstream**). In primary stream sharing, an anchor publishes screen sharing images via the primary stream. Because 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.

APIs

Description	C++	C#	Electron
Selects a sharing source	selectScreenCaptureTarget	selectScreenCaptureTarget	selectScreenCaptureTarget
Starts screen sharing	startScreenCapture	startScreenCapture	startScreenCapture
Pauses screen sharing	pauseScreenCapture	pauseScreenCapture	pauseScreenCapture
Resumes screen sharing	resumeScreenCapture	resumeScreenCapture	resumeScreenCapture
Ends screen sharing	stopScreenCapture	stopScreenCapture	stopScreenCapture

Getting Sharing Sources

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

explain

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.

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

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 `pauseScreenCapture` pauses screen capturing and displays the image at the moment it is paused. Remote users see the paused video image until screen capturing is resumed.

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 x 1080	10	800 Kbps
HD	1280 x 720	10	600 Kbps
SD	960 x 720	10	400 Kbps

Watching Shared Screen

When a 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 `ITRTCCloudCallback` .

Users who want to view the shared screen can start rendering the substream image 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", UTF82Wide(userId
    liteav::ITRTCCloud* trtc_cloud_ = getTRTCShareInstance();
    if (available) {
        trtc_cloud_->startRemoteView(userId, liteav::TRTCVideoStreamTypeSub, hWnd);
    } else {
        trtc_cloud_->stopRemoteView(userId, liteav::TRTCVideoStreamTypeSub);
    }
}
```

FAQs

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

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

2. 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.

Electron

Last updated：2023-09-28 11:44:09

This document describes how to share the screen. Currently, a TRTC room can have only one screen sharing stream at a time.

TRTC supports screen sharing in primary stream and substream modes on Electron:

Substream sharing

In TRTC, you can share the screen via a dedicated stream 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.

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.

Step 1. Get sharing sources

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

explain

On Electron, 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.

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

```
import TRTCCloud from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();
// https://web.sdk.qcloud.com/trtc/electron/doc/en-us/trtc_electron_sdk/TRTCCloud.h
const screenList = rtcCloud.getScreenCaptureSources();
```

Step 2. Start screen sharing

You can select the sharing source by calling selectScreenCaptureTarget.

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 `pauseScreenCapture` pauses screen capturing and displays the image at the moment it is paused. Remote users see the paused video image until screen capturing is resumed.

```
import TRTCCloud, {
    Rect, TRTCScreenCaptureProperty, TRTCVideoStreamType, TRTCVideoEncParam,
    TRTCVideoResolution, TRTCVideoResolutionMode
```

```
} from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();
// https://web.sdk.qcloud.com/trtc/electron/doc/zh-cn/trtc_electron_sdk/TRTCCloud.h
const screenList = rtcCloud.getScreenCaptureSources();
// https://web.sdk.qcloud.com/trtc/electron/doc/zh-cn/trtc_electron_sdk/Rect.html
const captureRect = new Rect(0, 0, 0, 0);
// https://web.sdk.qcloud.com/trtc/electron/doc/zh-cn/trtc_electron_sdk/TRTCScreenC
const property = new TRTCScreenCaptureProperty(
    true, true, true, 0, 0, false
);
if (screenList.length > 0) {
    rtcCloud.selectScreenCaptureTarget(screenList[0], captureRect, property)
}

const screenshareDom = document.querySelector('screen-dom');
// https://web.sdk.qcloud.com/trtc/electron/doc/zh-cn/trtc_electron_sdk/TRTCVideoEn
const encParam = new TRTCVideoEncParam(
    TRTCVideoResolution.TRTCVideoResolution_1920_1080,
    TRTCVideoResolutionMode.TRTCVideoResolutionModeLandscape,
    15,
    2000,
    0,
    false
);
rtcCloud.startScreenCapture(screenshareDom, TRTCVideoStreamType.TRTCVideoStreamType
```

Step 3. Set the video quality

You can use the third parameter (`encParam`) of the `startScreenCapture` API to set the video quality of screen sharing (see step 2), including resolution, bitrate, and frame rate. We recommend the following settings:

Clarity	Resolution	Frame Rate	Bitrate
HD+	1920 x 1080	10	2000 kbps
HD	1280 x 720	10	600 Kbps
SD	960 × 720	10	400 Kbps

Step 4. Watch the shared screen

When a 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.
Users who want to view the shared screen can start rendering the substream image of the remote user using the startRemoteView API.

```
import TRTCCloud, {
    TRTCVideoStreamType
} from 'trtc-electron-sdk';
```

```
const rtcCloud = new TRTCCloud();

const remoteDom = document.querySelector('.remote-user');
function onUserSubStreamAvailable(userId, available) {
    if (available === 1) {
        rtcCloud.startRemoteView(userId, remoteDom, TRTCVideoStreamType.TRTCVideoSt
    } else {
        rtcCloud.stopRemoteView(userId, TRTCVideoStreamType.TRTCVideoStreamTypeSub)
    }
}

rtcCloud.on('onUserSubStreamAvailable', onUserSubStreamAvailable);
```

FAQs

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

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

2. 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.

Flutter

Last updated：2024-05-31 17:08:26

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$TXScreenCaptureAssistantAc
    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
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.

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 Uplo
│   ├── Broadcast.Upload.entitlements        // Code for configuring an App Group to
│   ├── Broadcast.UploadDebug.entitlements   // Code for configuring an App Group to
│   ├── Info.plist
│   └── SampleHandler.swift      // Code for receiving screen recording data from th
├── 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
```

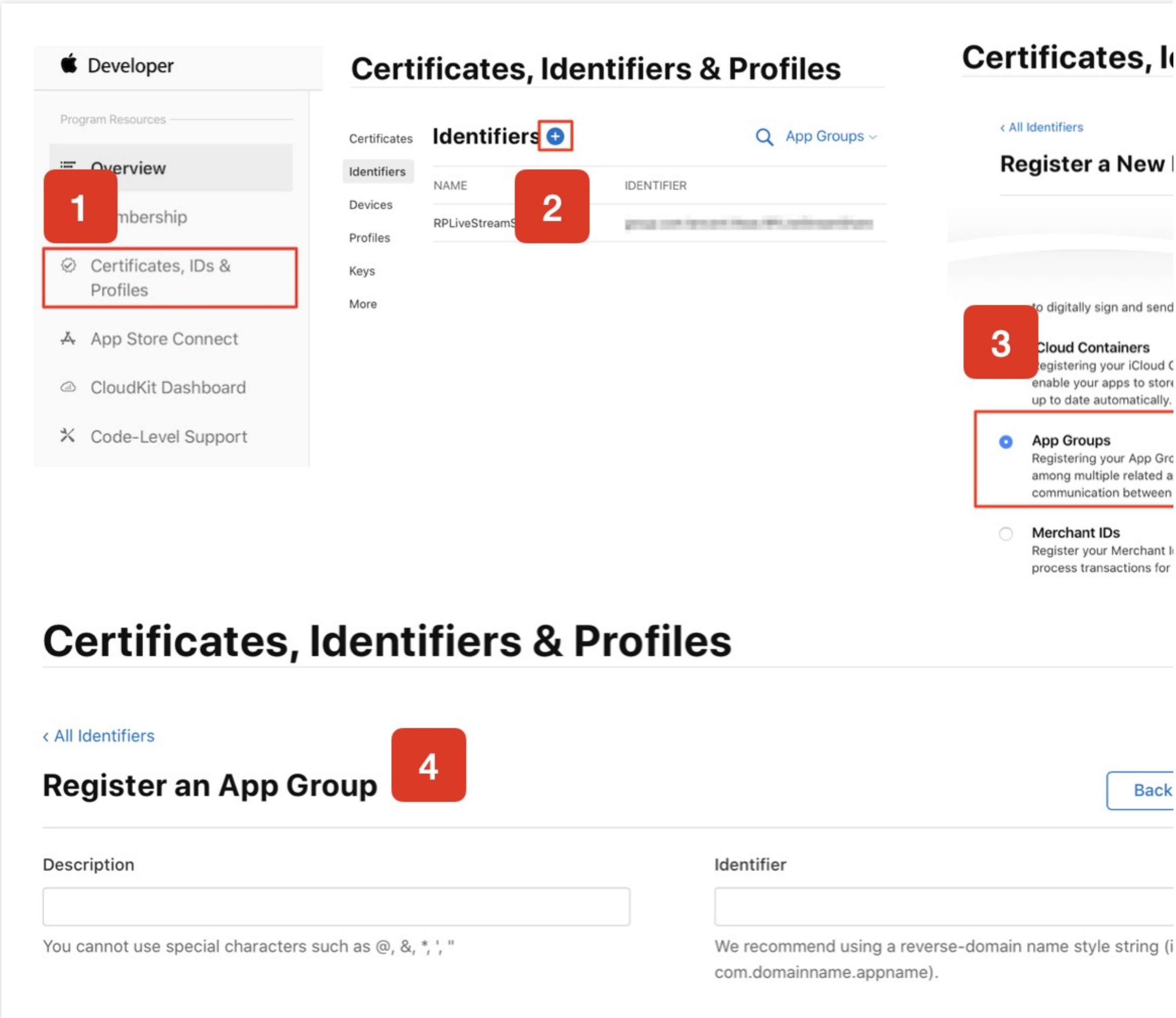
Note:

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

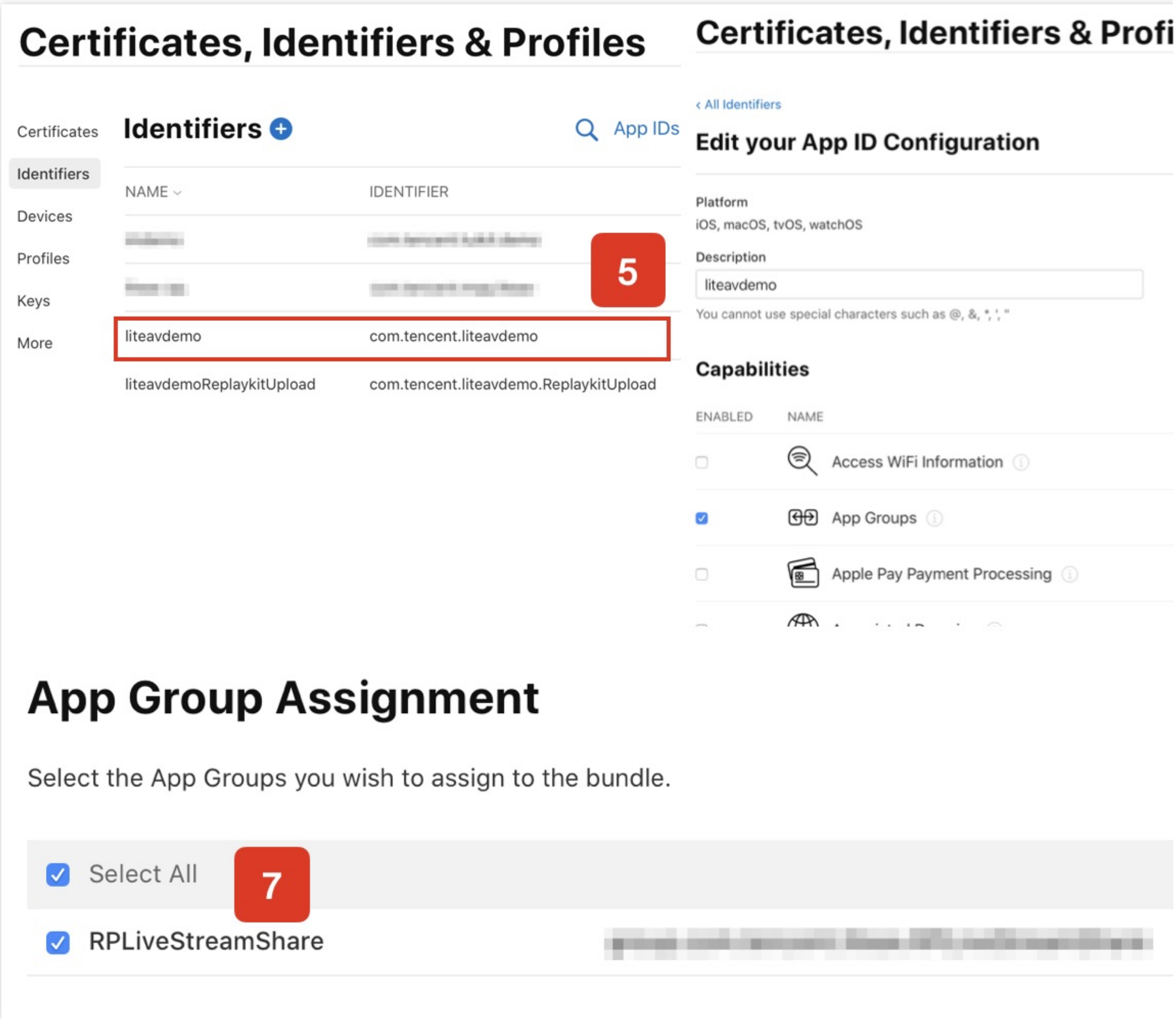
Log in to **https://developer.apple.com/** and do the following. **You need to download the provisioning profile again afterwards**.

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**.



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**.

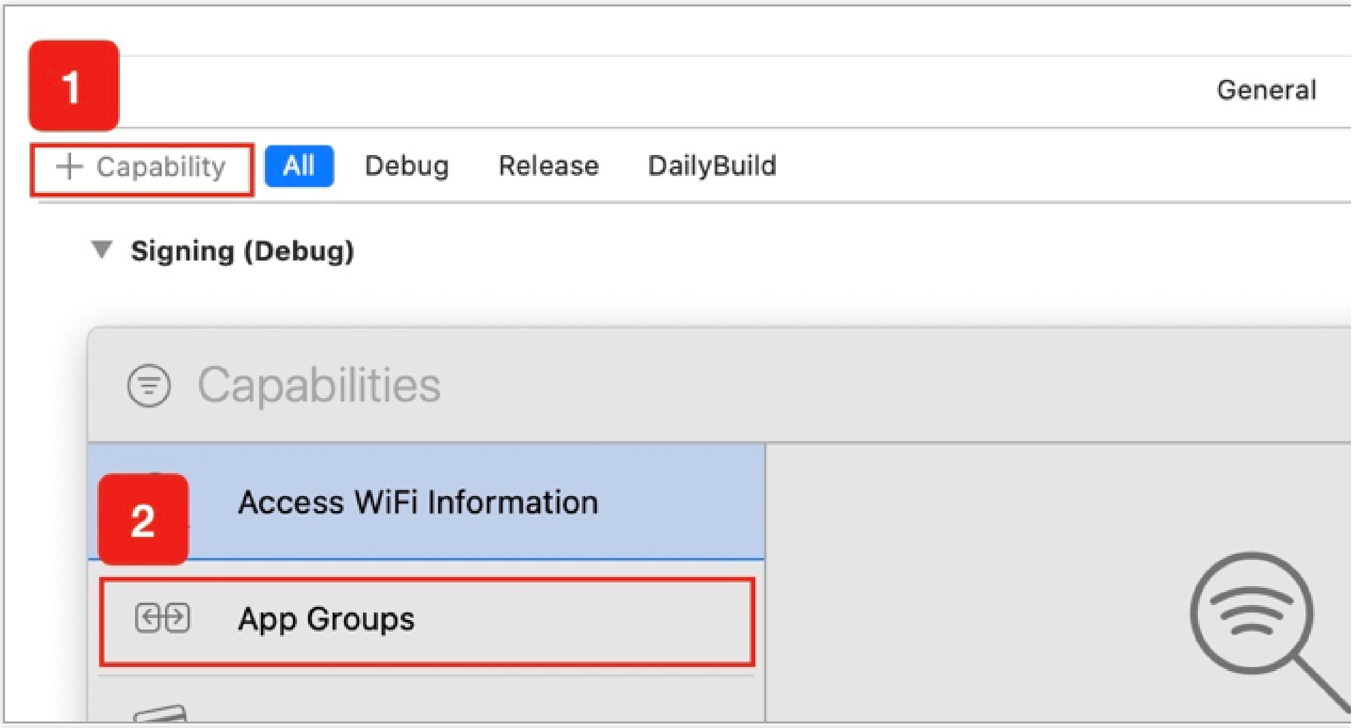
7. In the form that pops up, select the App Group you created, click **Continue** to return to the edit page, and click **Save** to save the settings.



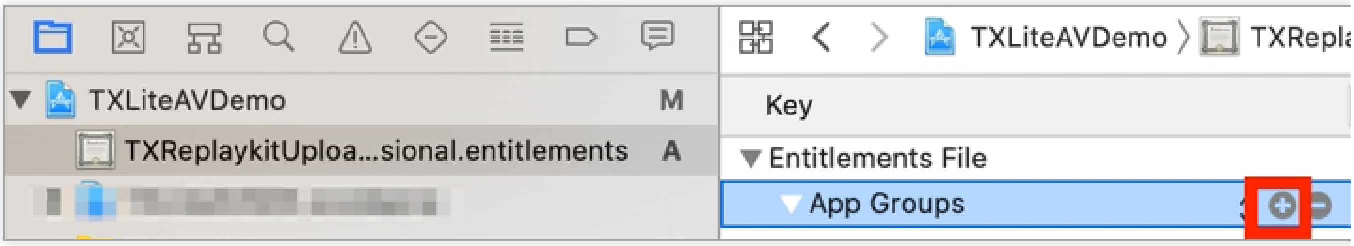
8. Download the provisioning profile again and import it to Xcode.

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 extens
        TXReplayKitExt.sharedInstance().setup(withAppGroup: APPGROUP, delegate: sel
    }

    override func broadcastPaused() {
        // User has requested to pause the broadcast. Samples will stop being deliv
    }

    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: TXReplayKitExtReaso
        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,
        finishBroadcastWithError(error)
    }

    override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer, with sampleBu
        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
trtcCloud.startScreenCapture(
    TRTCVideoEncParam(
        videoFps: 10,
        videoResolution: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_1280_720,
        videoBitrate: 1600,
        videoResolutionMode: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT,
    ),
    iosAppGroup,
);
```



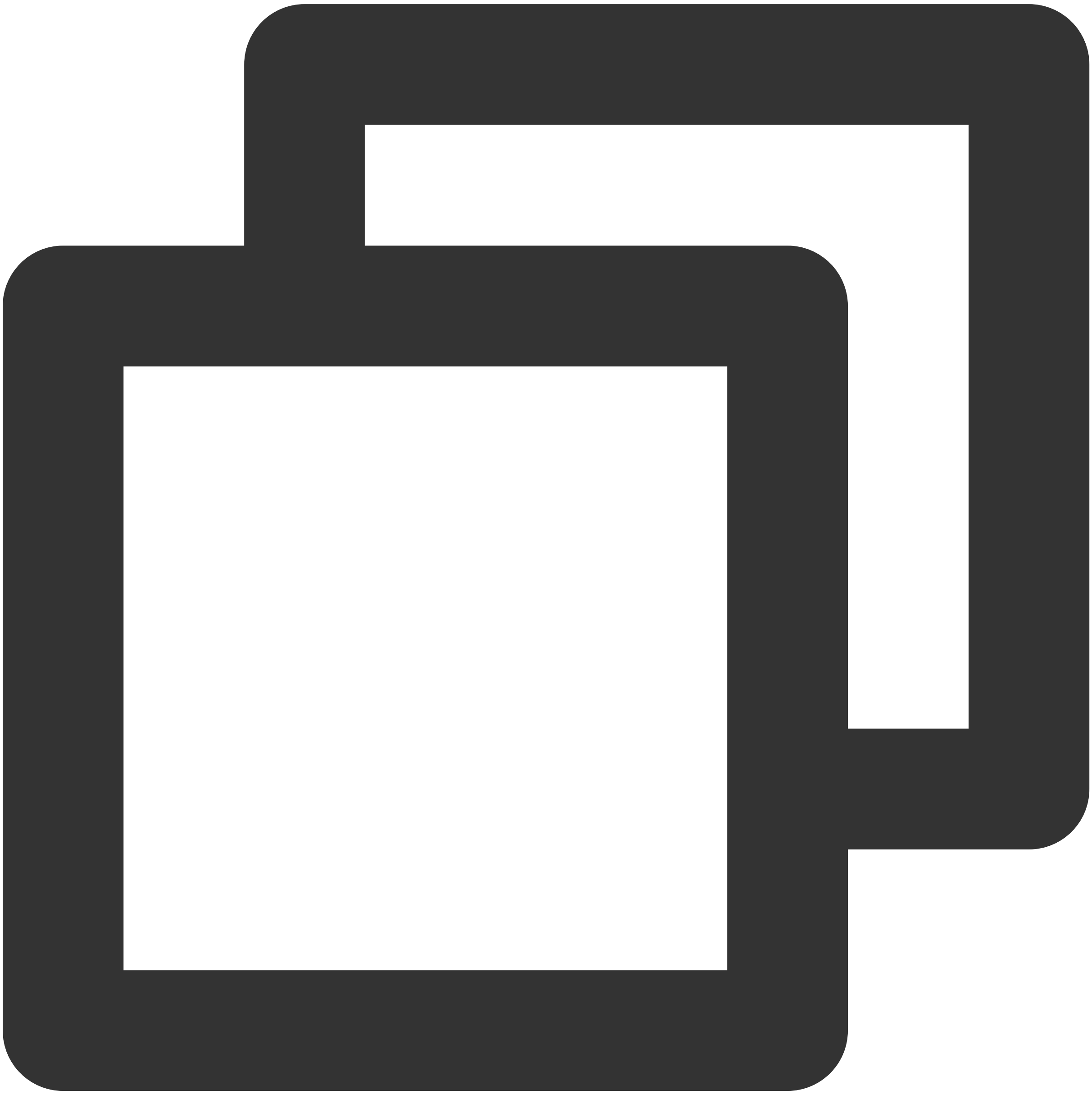
```
// Stop screen sharing
await trtcCloud.stopScreenCapture();


// Event notification for the start of screen sharing, which can be received throug
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);
 }
}
```

Based on the Windows platform

Screen sharing on the Windows platform supports two schemes: primary stream sharing and secondary stream sharing:

Secondary Stream Sharing

In TRTC, we can open a separate upstream video stream for screen sharing, which is called "secondary stream (**substream**)". Secondary stream sharing means the host uploads both the camera image and the screen image simultaneously. This is the scheme used by Tencent Meeting. You can enable this mode by specifying the `TRTCVideoStreamType` parameter as `TRTCVideoStreamTypeSub` when calling the `startScreenCapture` interface.

Primary Stream Sharing

In TRTC, we usually call the camera's channel "primary stream (**bigstream**)", which means sharing the screen through the camera channel. In this mode, the host has only one upstream video stream, either uploading the camera image or the screen image, and the two are mutually exclusive. You can enable this mode by specifying the `TRTCVideoStreamType` parameter as `TRTCVideoStreamTypeBig` when calling the `startScreenCapture` interface.

Step 1: Get sharing sources

You can enumerate a list of sharable windows using getScreenCaptureSources, with the list returned in the parameter sourceInfoList.

Note:

In Windows, the desktop screen is also considered a window, known as a Desktop Window. With two monitors, each monitor has its corresponding Desktop Window. Therefore, the window list returned by getScreenCaptureSources will also include Desktop Windows.

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

Step 2: Select sharing target

After obtaining the screens and windows that can be shared through getScreenCaptureSources, you can call the selectScreenCaptureTarget interface to select the desired target screen or window for sharing.

Step 3: Start screen sharing

After selecting a sharing target, the startScreenCapture API can be used to initiate screen sharing.

During the sharing process, you can change the sharing target by calling the selectScreenCaptureTarget API.

The difference between pauseScreenCapture and stopScreenCapture is that pause stops the capture of screen content and uses the image from the moment of pausing as a placeholder, so the remote side always sees the last frame until it resumes.

Set video encoding parameters

By setting the first parameter `encParams` in startScreenCapture(), you can specify the encoding quality of screen sharing, including resolution, bitrate, and frame rate. We provide the following suggested reference values:

Clarity Level	Resolution	Frame Rate	Bitrate
Ultra HD (HD+)	1920 × 1080	10	800kbps
HD	1280 × 720	10	600kbps
SD	960 × 720	10	400kbps

If you set `encParams` to null, the SDK will automatically use the previously set encoding parameters.

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.

Sharing Computer Audio
Web

Last updated：2023-10-30 11:13:12

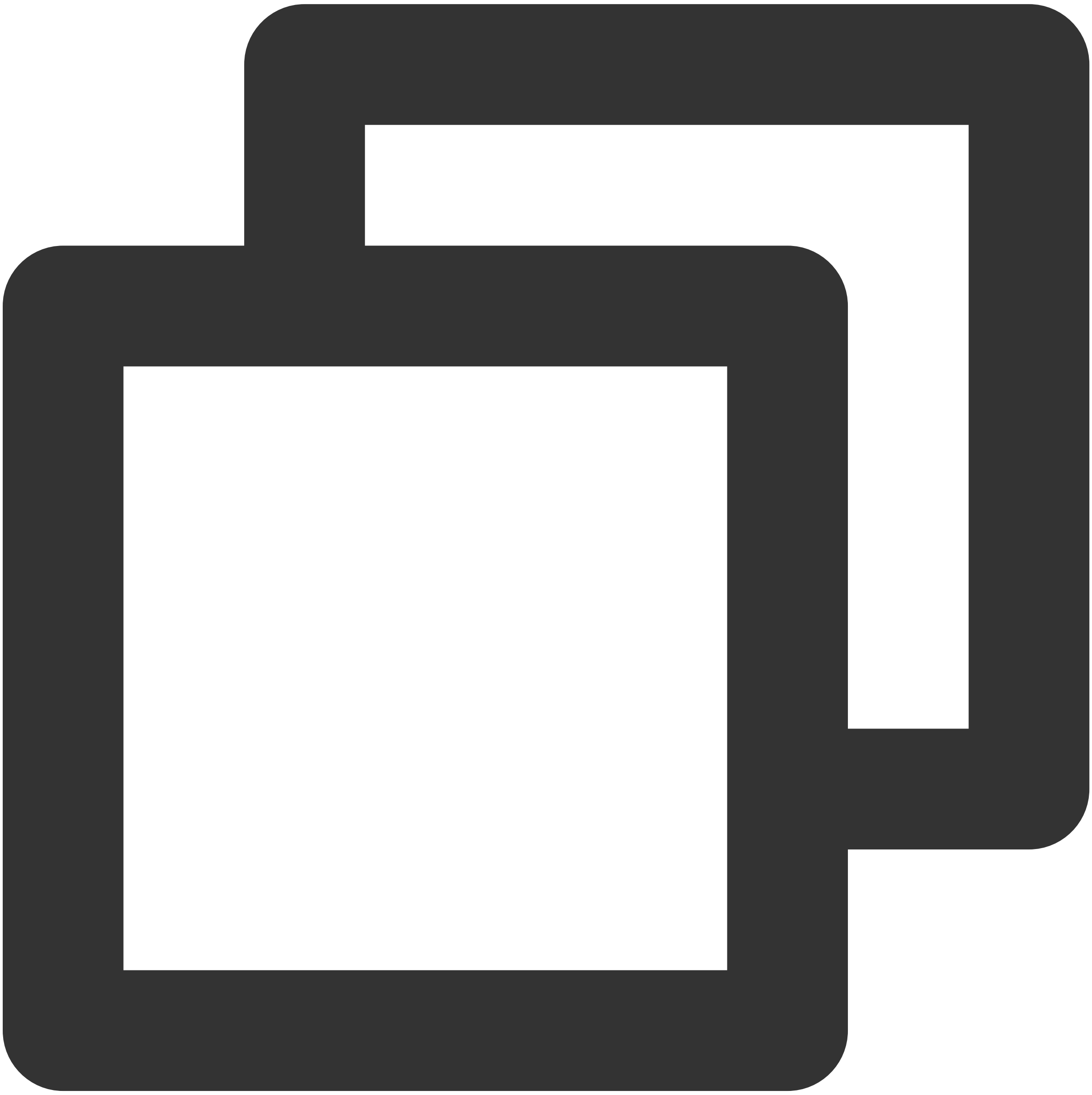
Function Description

This article mainly introduces how to share system audio in TRTC Web SDK.

Implementation Process

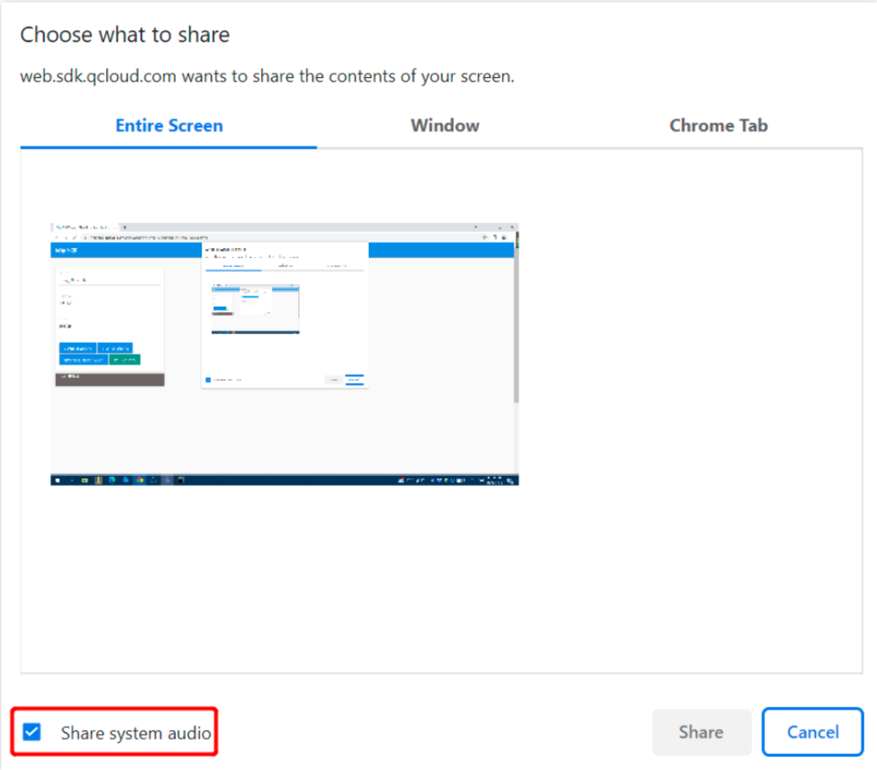
To share system audio on the web, you need to use it together with screen sharing. It is not possible to share system audio without screen sharing.

Example



```
await trtcA.startScreenShare({ option: { systemAudio: true }});
```

Check **Share system audio** in the screen sharing selection box and click Share. After publishing to the room, other users in the room will receive the
TRTC.EVENT.REMOTE_AUDIO_AVAILABLEEvent



Note:

If a microphone is captured while sharing system audio, the system audio will be mixed with the local microphone and published.

Supported Browsers

Share System Audio only supports browsers based on Chromium version 74+, such as Chrome, Edge, Opera, and so on. Other browsers are not supported at this time, e.g. Safari, Firefox.

Chrome for Windows & Chrome OS supports sharing system audio + tab audio.

Chrome for MacOS & Linux only supports sharing the tab audio.

Chrome for Android & iOS does not support this.

macOS

Last updated：2023-09-28 11:45:05

Pain Point and Solution

It is often necessary to share system audio in scenarios such as screen sharing, but the sound cards of Mac computers do not allow the capturing of system audio, making it impossible to share system audio on Mac computers. To solve this problem, TRTC has introduced a feature that records system audio on Mac computers. See below for details on how to enable the feature.

Directions

Step 1. Integrate the TRTCPrivilegedTask library

The TRTC SDK uses the TRTCPrivilegedTask library to get root access and install the virtual sound card plugin `TRTCAudioPlugin.driver` in the system directory `/Library/Audio/Plug-Ins/HAL` .

Integration via CocoaPods

Manual integration

1. Open the `Podfile` file in the root directory of your project and add the following content:

```
platform :osx, '10.10'

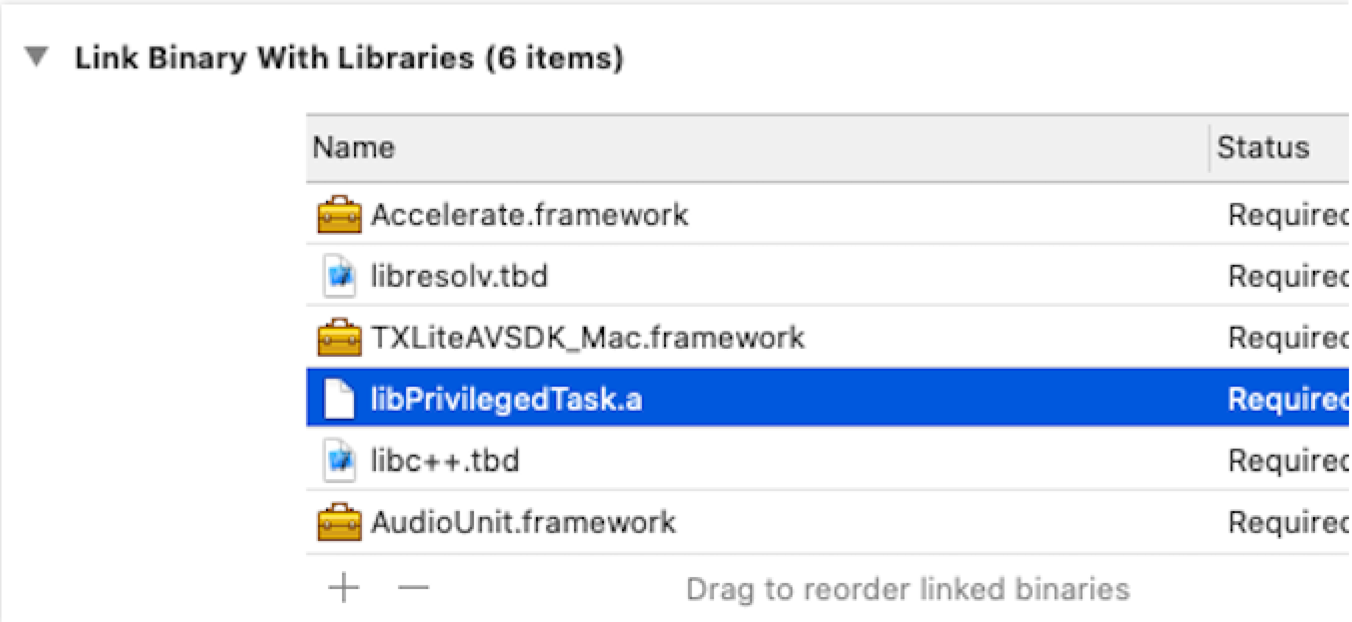
target 'Your Target' do
    pod 'TRTCPrivilegedTask', :podspec => 'https://pod-1252463788.cos.ap-guangzhou.
end
```

2. Run the `pod install` command to install the **TRTCPrivilegedTask** library.

explain:

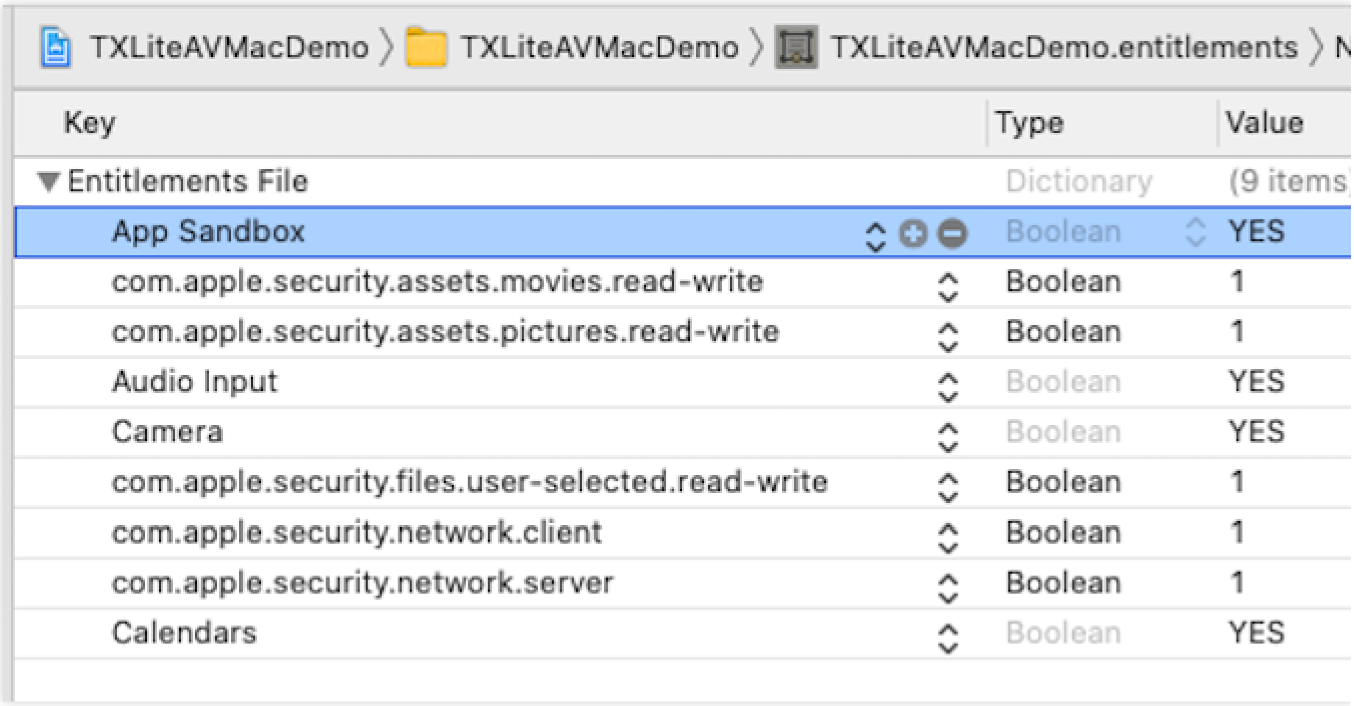
If you cannot find a Podfile file in the directory, run the pod init command to create one and then add the above content.
For how to install CocoaPods, see CocoaPods' official installation document.

1. Download the TRTCPrivilegedTask library.
2. Decompress the downloaded file, open your Xcode project, and import the file to the project.
3. Select the target to run, select **Build Phases**, expand **Link Binary with Libraries**, click **+**, and add the dependent library `libPrivilegedTask.a` .



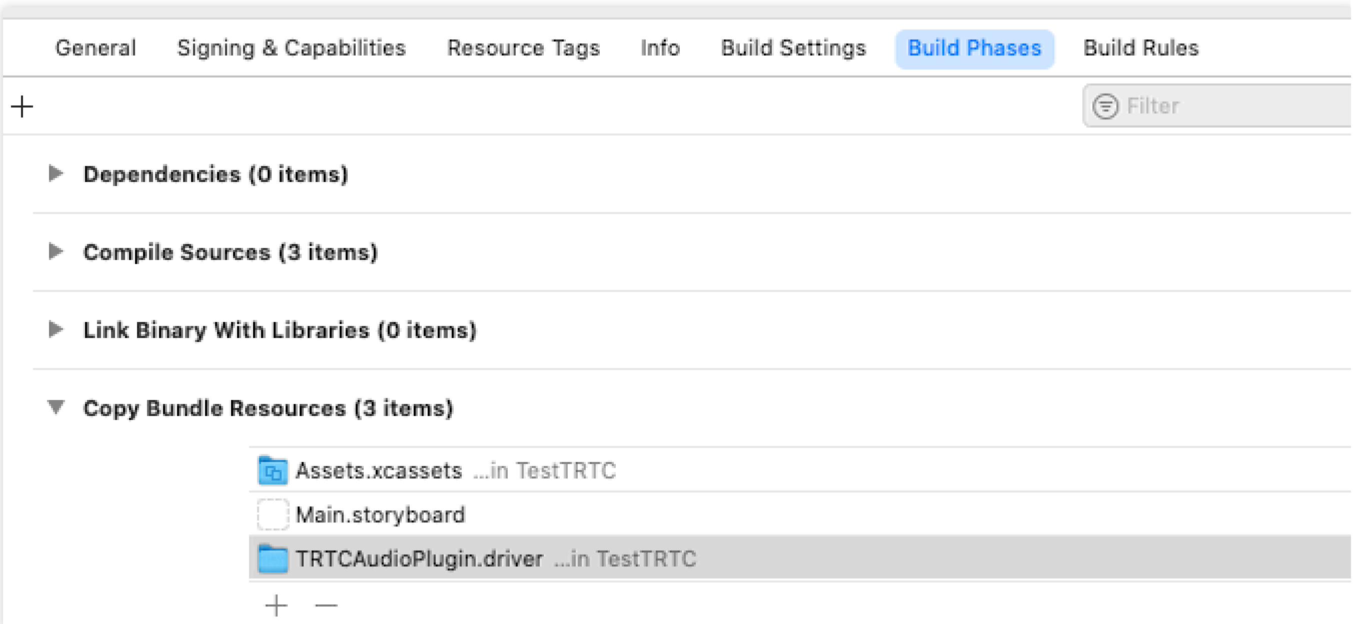
Step 2. Disable App Sandbox

In the entitlements file of the app, delete **App Sandbox**.

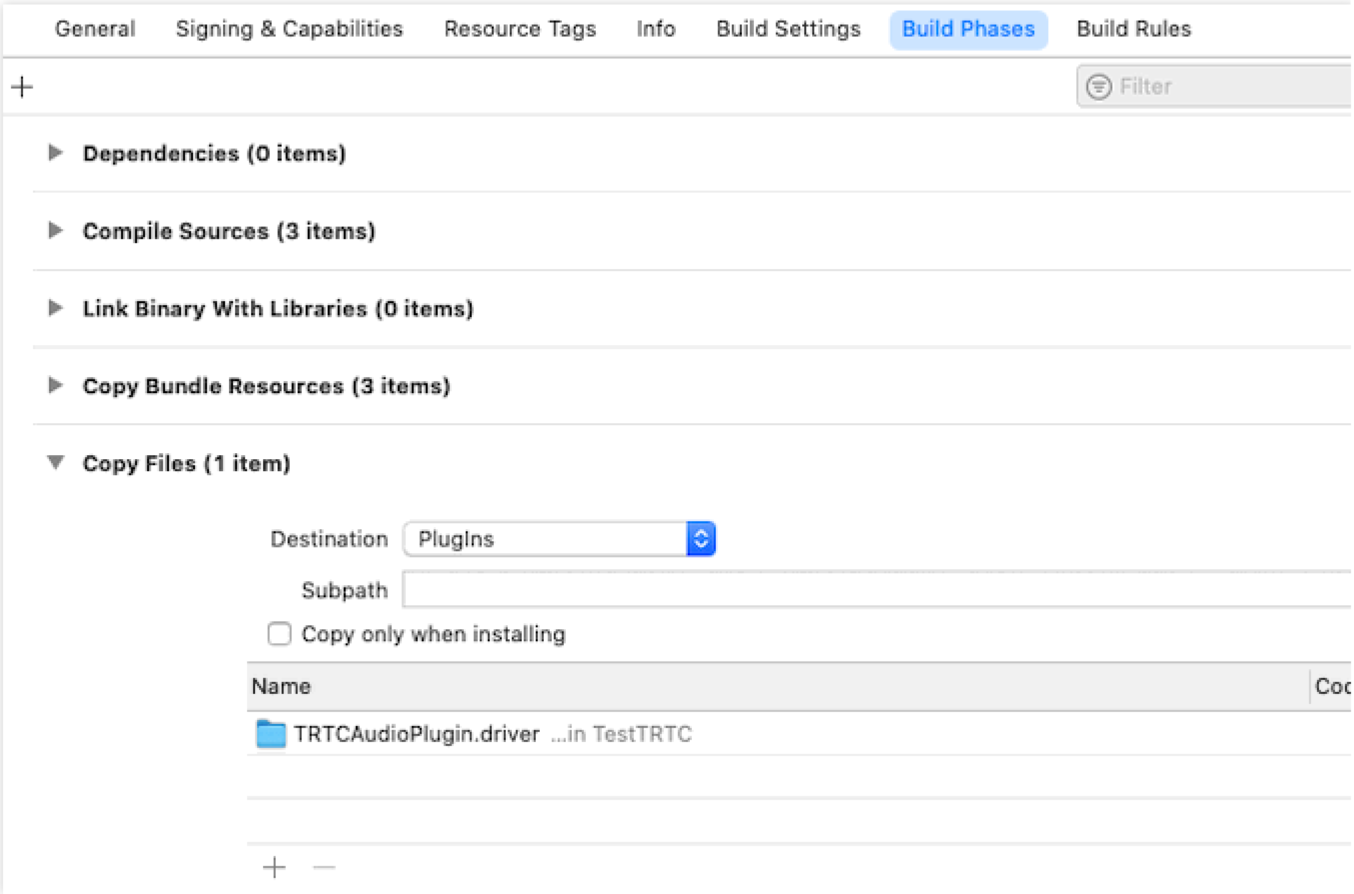


Step 3. Package the virtual sound card plugin

After you integrate the TRTCPrivilegedTask library and disable App Sandbox, when you use the system audio recording feature for the first time, the SDK will download the virtual sound card plugin from the internet and install it. To accelerate this process, you can package the virtual sound card plugin `TRTCAudioPlugin.driver` in the `PlugIns` directory of `TXLiteAVSDK_TRTC_Mac.framework` to the resources directory of the app's bundle, as shown below.



Alternatively, copy the file to the `PlugIns` directory of the app's bundle.



Step 4. Start capturing system audio

Call the startSystemAudioLoopback API to start system audio capturing and mix the audio into the upstream audio stream. The result is called back via onSystemAudioLoopbackError.


```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[_trtc startLocalAudio];
[trtcCloud startSystemAudioLoopback];
```

notice

After the TRTCPrivilegedTask library is integrated and App Sandbox disabled, when you call `startSystemAudioLoopback` for the first time, the SDK will request root access.After being granted root access, the SDK will start installing the virtual sound card plugin to the computer automatically.

Step 5. Stop capturing system audio

Call the stopSystemAudioLoopback API to stop capturing system audio.

```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[trtcCloud stopSystemAudioLoopback];
```

Step 6. Set the volume of system audio capturing

Call the setSystemAudioLoopbackVolume API to set the volume of system audio capturing.


```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[trtcCloud setSystemAudioLoopbackVolume:80];
```

Summary

TRTC records system audio on Mac computers using the virtual sound card plugin `TRTCAudioPlugin.driver` . For the plugin to work, you need to copy it to the system directory `/Library/Audio/Plug-Ins/HAL` and restart the audio service. You can check whether the plugin is installed successfully using the Audio MIDI Setup app, which can be found in the `Other` folder of Launchpad. The presence of a device named "TRTC Audio Device" in the device list of the app indicates that the plugin is installed successfully.

The purpose of integrating the TRTCPrivilegedTask library and disabling App Sandbox is for the SDK to get root access so as to install the virtual sound card plugin; otherwise it cannot automatically install the plugin. However, if a virtual sound card is already installed in the system, you can use the system audio recording feature without integrating the TRTCPrivilegedTask library or disabling App Sandbox.

explain

You can also manually install a virtual sound card to enable the feature.

1. Copy `TRTCAudioPlugin.driver` in the `PlugIns` directory of `TXLiteAVSDK_TRTC_Mac.framework` to the system directory `/Library/Audio/Plug-Ins/HAL` .
2. Restart the system audio service.

```
sudo cp -R TXLiteAVSDK_TRTC_Mac.framework/PlugIns/TRTCAudioPlugin.driver /Library/
sudo kill -9 `ps ax|grep 'coreaudio[a-z]' |awk '{print $1}'`
```

Notes

Disabling App Sandbox will change the user paths obtained in your app.Directories returned via methods such as the calling of `NSSearchPathForDirectoriesInDomains` will change from sandbox directories to user directories. For example, `~/Documents` and `~/Library` will become `/Users/UsernameDocuments` and `/Users/Username/Library`.

You may be unable to release your app to the Mac App Store after integrating the TRTCPrivilegedTask library.App Sandbox must be disabled for the SDK to get root access and automatically install a virtual sound card. This may cause your app to be rejected by the Mac App Store. For details, see App Store Review Guidelines.If you need to release your app to the Mac App Store or want to use the Sandbox feature, consider manually installing a virtual sound card.

Electron

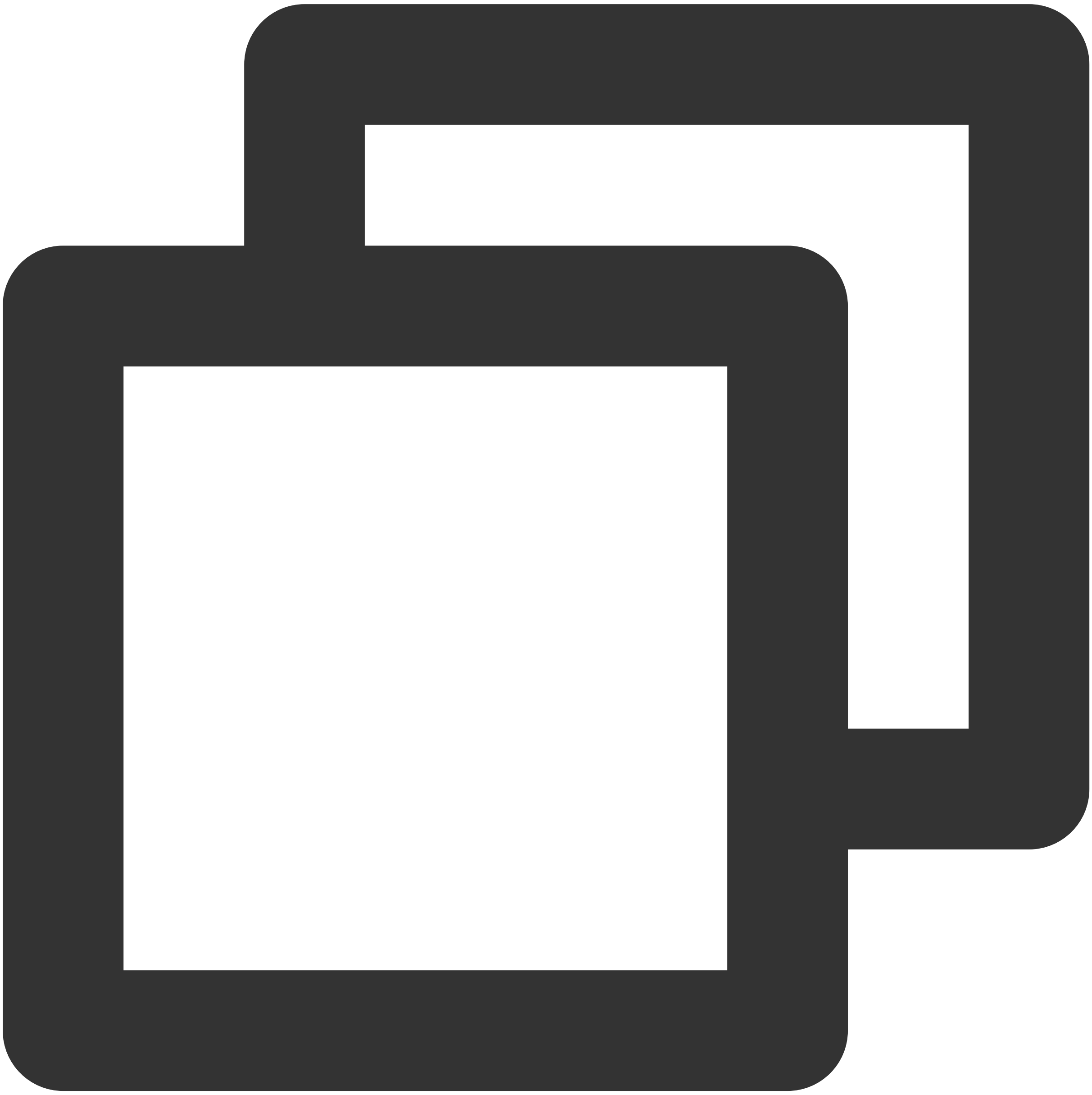
Last updated：2023-09-28 11:45:46

Pain Points and Solutions

It is often necessary to share system audio in scenarios such as screen sharing, but the sound cards of Mac computers do not allow the capturing of system audio when the application is packaged by Electron, making it impossible to share system audio on Mac computers. To solve this problem, TRTC has introduced a feature that records system audio on Mac computers. See below for details on how to enable the feature.

Step 1. Start capturing system audio

Call the startSystemAudioLoopback API to start system audio capturing and mix the audio into the upstream audio stream. The result is called back via onSystemAudioLoopbackError.



```
import TRTCCloud, { TRTCAudioQuality } from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();
function onSystemAudioLoopbackError(errCode) {
  if (errCode === 0) {
    console.log('Started successfully');
  }
  if (errCode === -1330) {
    console.log('Failed to enable system sound recording; for example, the audio dr
  }
  if (errCode === -1331) {
    console.log('No permission to install the audio driver plugin');
```

```
  }
  if (errCode === -1332) {
    console.log('Failed to install the audio driver plugin');
  }
}

trtcCloud.on('onSystemAudioLoopbackError', onSystemAudioLoopbackError);
trtcCloud.startLocalAudio(TRTCAudioQuality.TRTCAudioQualityDefault);
trtcCloud.startSystemAudioLoopback();
```

notice

When you call `startSystemAudioLoopback` for the first time, the SDK will request root access. After being granted root access, the SDK will start installing the virtual sound card plugin to the computer automatically.

Step 2. Stop capturing system audio

Call the stopSystemAudioLoopback API to stop system audio capturing.

```
trtcCloud.stopSystemAudioLoopback();
```

Step 3. Set the volume of system audio capturing

Call the setSystemAudioLoopbackVolume API to set the volume of system audio capturing.

```
trtcCloud.setSystemAudioLoopbackVolume(60);
```

Summary

TRTC records system audio on Mac computers using the virtual sound card plugin `TRTCAudioPlugin.driver` . For the plugin to work, you need to copy it to the system directory `/Library/Audio/Plug-Ins/HAL` and restart the audio service. You can check whether the plug is installed successfully using the Audio MIDI Setup app, which can be found in the `Other` folder of Launchpad. The presence of a device named "TRTC Audio Device" in the device list of the app indicates that the plugin is installed successfully.

Flutter

Last updated：2024-02-02 18:48:51

This document primarily delineates the process of sharing system sounds. At present, TRTC defaults not to collect the audio of the local application.

Call Guidelines

Android

iOS

This document primarily delineates the process of sharing system sounds. At present, TRTC defaults not to collect the audio of the local application.

Note:

Only Android 10.0 or higher supports sharing system audio.

Initiate system sound sharing

Step 1: Initiate Screen Sharing

Follow the steps in Enable screen sharing - Based on Android platform to turn on screen sharing.

Step 2: Initiate Share System Audio

By invoking the `TRTCCloud` 's startSystemAudioLoopback interface, the collected system audio will be automatically mixed into the upstream.

Step 3: Terminate Share System Audio

Invoke the `TRTCCloud` 's stopSystemAudioLoopback interface.

Step 1: Enable the microphone

Within the App, invoke startLocalAudio to kickstart microphone collection, and it's suggested to utilize TRTCAudioQualityDefault for audio quality.

Note:

This action is indispensable; by initiating microphone collection, the App can preserve its functioning even when relegated to the background.

Step 2: Initiate screen sharing

Due to the restrictions of iOS, system sounds can only be collected during screen recording. Therefore, to implement this feature, the iOS screen sharing function must be connected first.

Follow the steps in Enable Screen Sharing - Based on iOS platform to start screen recording, and system sound will be automatically collected.

Note:

When starting screen recording, do not light up the microphone icon. Voice collection has already been started in the app.

Step 3: Midway open and close system sound

The system audio capture and screen recording are done concurrently, initiating automatically with the start of the recording and ceasing alongside the recording termination. It is not feasible to independently switch on or mute the system audio.

TRTCCloud proffers the setSystemAudioLoopbackVolume method for system audio volume conditioning. When there is no desire to output the system audio, the volume can be set to 0.

Setting Video Quality
Android&iOS&Windows&Mac

Last updated：2023-09-28 11:46:23

Introduction

In TRTCCloud, you can adjust the video quality in the following ways:
`TRTCAppScene` in TRTCCloud.enterRoom: used to select your application scenario.
TRTCCloud.setVideoEncoderParam: used to set the encoding parameter.
TRTCCloud.setNetworkQosParam: used to set the network control policy.
This document describes how to configure these parameters to make the video quality of the TRTC SDK meet your project-specific needs.
You can also see the following demos:
iOS：SetVideoQualityViewController.m
Android：SetVideoQualityActivity.java
Windows: TRTCMainViewController.cpp

Supported Platforms

iOS	Android	macOS	Windows	Web	Electron	Flutter
✓	✓	✓	✓	✓	✓	✓

For detailed directions on how to set video quality for the Web, please see Configuration Guide.

TRTCAppScene

VideoCall
This corresponds to the scenario where most of the time there are two or more people on video calls, and the optimization for internal encoders and network protocols focuses on smoothness to reduce call latency and lagging.
LIVE
This corresponds to the scenario where most of the time there is only one person speaking or performing and occasionally multiple people interact with one another through video, and the optimization for internal encoders and network protocols focuses on performance and compatibility to deliver better performance and definition.

TRTCVideoEncParam

Recommended configuration

Application Scenario	videoResolution	videoFps	videoBitrate
Video call (mobile)	640x360	15	550 Kbps
Video conferencing (primary image on macOS or Windows)	1280x720	15	1,200 Kbps
Video conferencing (primary image on mobile device)	640x360	15	900 Kbps
Video conferencing (small image)	320x180	15	250 Kbps
Online education (teacher on macOS or Windows)	960x540	15	850 Kbps
Online education (teacher on iPad)	640x360	15	550 Kbps
Online education (student)	320x180	15	250 Kbps

Detailed description of fields

(TRTCVideoResolution) videoResolution

Encoded resolution; for example, 640x360 refers to the width (pixels) x height (pixels) of the encoded video image. In the `TRTCVideoResolution` enum definition, only landscape resolution (i.e., width >= height) is defined. If you want to use portrait resolution, you need to set `resMode` to `Portrait` .

notice

As many hardware codecs only support pixel widths that are divisible by 16, the actual resolution encoded by the SDK is not necessarily exactly the same as configured by the parameter; instead, it will be automatically corrected based on the divisor of 16; for example, the resolution 640x360 may be adapted to 640x368 inside the SDK.

(TRTCVideoResolutionMode) resMode

This determines the landscape or portrait resolution. Because only landscape resolution is defined in `TRTCVideoResolution` , if you want to use portrait resolution such as 360x640, you need to specify `resMode` as `TRTCVideoResolutionModePortrait` . Generally, landscape resolution is used on PCs and Macs, while portrait resolution is used on mobile devices.

(int) videoFps

Frame rate (FPS), which indicates how many frames are encoded per second. The recommended value is 15 FPS, which can ensure that the video image is smooth enough without reducing the video definition due to too many frames per second.

If you have high requirements for smoothness, you can set the frame rate to 20 or 25 FPS. However, please do not set a value above 25 FPS, because the normal frame rate of movies is only 24 FPS.

(int) videoBitrate

Video bitrate, which indicates how many Kbits of encoded binary data is output by the encoder per second. If you set `videoBitrate` to 800 Kbps, the encoder will generate 800 Kbits of video data per second. If such data is stored as a file, the file size will be 800 Kbits, which is 100 KB or 0.1 MB.

A higher video bitrate is not always better; instead, it must have a proper mapping relationship with resolution as shown in the table below.

Resolution-bitrate reference table

Resolution Definition	Aspect Ratio	Recommended Bitrate(VideoCall)	Recommended Bitrate(LIVE)
TRTCVideoResolution_120_120	1:1	80 Kbps	120 Kbps
TRTCVideoResolution_160_160	1:1	100 Kbps	150 Kbps
TRTCVideoResolution_270_270	1:1	200 Kbps	300 Kbps
TRTCVideoResolution_480_480	1:1	350 Kbps	525 Kbps
TRTCVideoResolution_160_120	4:3	100 Kbps	150 Kbps
TRTCVideoResolution_240_180	4:3	150 Kbps	225 Kbps
TRTCVideoResolution_280_210	4:3	200 Kbps	300 Kbps
TRTCVideoResolution_320_240	4:3	250 Kbps	375 Kbps
TRTCVideoResolution_400_300	4:3	300 Kbps	450 Kbps
TRTCVideoResolution_480_360	4:3	400 Kbps	600 Kbps
TRTCVideoResolution_640_480	4:3	600 Kbps	900 Kbps
TRTCVideoResolution_960_720	4:3	1,000 Kbps	1,500 Kbps
TRTCVideoResolution_160_90	16:9	150 Kbps	250 Kbps
TRTCVideoResolution_256_144	16:9	200 Kbps	300 Kbps
TRTCVideoResolution_320_180	16:9	250 Kbps	400 Kbps
TRTCVideoResolution_480_270	16:9	350 Kbps	550 Kbps
TRTCVideoResolution_640_360	16:9	550 Kbps	900 Kbps

TRTCVideoResolution_960_540	16:9	850 Kbps	1,300 Kbps
TRTCVideoResolution_1280_720	16:9	1,200 Kbps	1,800 Kbps
TRTCVideoResolution_1920_1080	16:9	2,000kbps	3,000kbps

TRTCNetworkQosParam

QosPreference

If the network bandwidth is sufficient, both high definition and smoothness can be ensured; however, if the user's network connection is not ideal, should priority be given to definition or smoothness? You can make a choice by specifying the `preference` parameter in `TRTCNetworkQosParam` .

Smoothness preferred (TRTCVideoQosPreferenceSmooth)

Smoothness is ensured on a weak network, while the video image will have a lot of blurs but can be smooth with no or slight lagging.

Definition preferred (TRTCVideoQosPreferenceClear)

Definition is ensured on a weak network, i.e., the image will be as clear as possible but tend to lag.

ControlMode

For the `controlMode` parameter, select **TRTCQosControlModeServer**. `TRTCQosControlModeClient` is used for internal debugging by the Tencent Cloud R&D team and should be ignored.

Common Misunderstandings

1. **The higher the resolution, the better?**

Higher resolutions require higher bitrates for support. If the resolution is 1280x720, but the bitrate is specified as 200 Kbps, the video image will have a lot of blurs. We recommend that you set it as described in the Resolution-bitrate reference table.

2. **The higher the frame rate, the better?**

Because the image captured by the camera is a complete mapping to all real objects in the exposure phase, it is not that the higher the frame rate, the smoother the video, which is different from the concept of FPS in games. On the contrary, if the frame rate is too high, the quality of each video frame will be lowered, and the exposure time of the camera will be reduced, worsening the image effect.

3. **The higher the bitrate, the better?**

4. Higher bitrates also require higher resolutions for a match. For a resolution of 320x240, a 1,000 Kbps bitrate would be wasteful. We recommend that you set it as described in the Resolution-bitrate reference table.

5. **High resolution and bitrate can be set when connected to a Wi-Fi network?**

It is not that the Wi-Fi network speed is constant. If the device is far from the wireless router or the router channel is occupied, the Wi-Fi network may not be as fast as 4G.
In response to this issue, the TRTC SDK provides a speed test feature, which can perform speed test to determine the network quality based on the score before a video call is established.

Web

Last updated：2023-05-30 17:14:23

This article mainly introduces how to set video properties in video calls or interactive live broadcasts. Developers can adjust the clarity and fluency of the video according to specific business needs to obtain a better user experience. Video properties include resolution, frame rate, and bit rate.

Implementation

Set the video properties through the `trtc.startLocalVideo()` or `trtc.updateLocalVideo()` method of the trtc object:

Specify a predefined Profile, each Profile corresponds to a set of recommended resolution, frame rate, and bit rate.

```
// Specify video properties when starting
await trtc.startLocalVideo({
  option: { profile: '480p' }
});
// Dynamically adjust video properties during the call
await trtc.updateLocalVideo({
  option: { profile: '360p' }
});
```

Specify custom resolution, frame rate, and bit rate


```
// Specify video properties when starting
await trtc.startLocalVideo({
  option: { profile: { width: 640, height: 480, frameRate: 15, bitrate: 900 /* kpbs
});
// Dynamically adjust video properties during the call
await trtc.updateLocalVideo({
  option: { profile: { width: 640, height: 360, frameRate: 15, bitrate: 800 /* kpbs
});
```

Video Property Profile List

Video Profile	Resolution (width x height)	Frame Rate (fps)	Bit Rate (kbps)
120p	160 x 120	15	200
180p	320 x 180	15	350
240p	320 x 240	15	400
360p	640 x 360	15	800
480p	640 x 480	15	900
720p	1280 x 720	15	1500
1080p	1920 x 1080	15	2000
1440p	2560 x 1440	30	4860
4K	3840 x 2160	30	9000

Due to device and browser limitations, the video resolution may not match exactly. In this case, the browser will automatically adjust the resolution to be close to the resolution corresponding to the Profile.

Electron

Last updated：2023-09-28 11:47:39

This document describes how to set the image quality for a video call or live streaming session. You can set the properties according to your requirements for video quality and smoothness to deliver better user experience. Video properties include resolution, frame rate, and bitrate.

Overview

In the Electron SDK, you can adjust the image quality in the following ways:
`TRTCAppScene` parameter in enterRoom: Used to select the scenario.
setVideoEncoderParam: Used to set the encoding parameter.
setNetworkQosParam: Used to specify the QoS control policy.
This document describes how to configure these parameters to make the video quality of the TRTC SDK meet your project-specific needs.
You can also refer to Electron API Example: video-quality.

TRTCAppScene

VideoCall: This scenario is most suitable when there are two or more people on a video call. The internal encoders and network protocols are optimized for video smoothness to reduce call latency and stuttering.
LIVE: This scenario is most suitable when there is only one person speaking or performing for an online audience, and occasionally multiple people interact with one another through video. The internal encoders and network protocols are optimized for performance and compatibility to deliver better performance and video clarity.

TRTCVideoEncParam

Recommended configuration

Application Scenario	videoResolution	videoFps	videoBitrate
Video conferencing (primary image on macOS or Windows)	1280 x 720	15	1,200 Kbps
Online education (teacher on macOS or Windows)	960 x 540	15	850 Kbps

Detailed description of fields

(TRTCVideoResolution) videoResolution

Encoded resolution (for example, 640 x 360) refers to the width (pixels) x height (pixels) of the encoded video image. In the `TRTCVideoResolution` enum definition, only landscape resolution (i.e., width >= height) is defined. If you want to use portrait resolution, you need to set `resMode` to `Portrait` .

notice

Because many hardware codecs only support pixel widths that are divisible by 16, the actual resolution encoded by the SDK may not be exactly the same as configured by the parameter; instead, it will be automatically corrected based on a multiple of 16. For example, the resolution 640 x 360 may be adapted to 640 x 368 inside the SDK.

(TRTCVideoResolutionMode) resMode

Whether to use landscape or portrait resolutions. Because only landscape resolutions are defined in `TRTCVideoResolution` , if you want to a use portrait resolution such as 360 x 640, you need to specify `resMode` as `TRTCVideoResolutionModePortrait` . Generally, landscape resolutions are used on PCs and Macs, while portrait resolutions are used on mobile devices.

(int) videoFps

Frame rate (FPS), which indicates how many frames are encoded per second. The recommended value is 15 fps, which can ensure that the video image is smooth enough without reducing the video quality due to too many frames per second.

If you have high requirements for smoothness, you can set the frame rate to 20 or 25 fps. However, do not set a value above 25 fps, because the normal frame rate of movies is only 24 fps.

(int) videoBitrate

Video bitrate, which indicates how many Kbits of encoded binary data is output by the encoder per second. If you set `videoBitrate` to 800 Kbps, the encoder will generate 800 Kbits of video data per second. If the data is stored as a file, the file size will be 800 Kbits, which is 100 KB or 0.1 MB.

A higher video bitrate is not always better; instead, it should be chosen appropriately based on the resolution as shown in the table below.

Resolution-bitrate reference table

Resolution Definition	Aspect Ratio	Recommended Bitrate(VideoCall)	Recommended Bitrate(LIVE)
TRTCVideoResolution_120_120	1:1	80 Kbps	120 Kbps
TRTCVideoResolution_160_160	1:1	100 Kbps	150 Kbps
TRTCVideoResolution_270_270	1:1	200 Kbps	300 Kbps
TRTCVideoResolution_480_480	1:1	350 Kbps	525 Kbps
TRTCVideoResolution_160_120	4:3	100 Kbps	150 Kbps

TRTCVideoResolution_240_180	4:3	150 Kbps	225 Kbps
TRTCVideoResolution_280_210	4:3	200 Kbps	300 Kbps
TRTCVideoResolution_320_240	4:3	250 Kbps	375 Kbps
TRTCVideoResolution_400_300	4:3	300 Kbps	450 Kbps
TRTCVideoResolution_480_360	4:3	400 Kbps	600 Kbps
TRTCVideoResolution_640_480	4:3	600 Kbps	900 Kbps
TRTCVideoResolution_960_720	4:3	1,000 Kbps	1,500 Kbps
TRTCVideoResolution_160_90	16:9	150 Kbps	250 Kbps
TRTCVideoResolution_256_144	16:9	200 Kbps	300 Kbps
TRTCVideoResolution_320_180	16:9	250 Kbps	400 Kbps
TRTCVideoResolution_480_270	16:9	350 Kbps	550 Kbps
TRTCVideoResolution_640_360	16:9	550 Kbps	900 Kbps
TRTCVideoResolution_960_540	16:9	850 Kbps	1,300 Kbps
TRTCVideoResolution_1280_720	16:9	1,200 Kbps	1,800 Kbps
TRTCVideoResolution_1920_1080	16:9	2,000kbps	3,000kbps

TRTCNetworkQosParam

QosPreference

If the network bandwidth is sufficient, both high definition and smoothness can be ensured. However, if the user's network connection is not ideal, you can choose whether to give priority to video quality or smoothness by specifying the `preference` parameter in `TRTCNetworkQosParam` .

Smoothness preferred (TRTCVideoQosPreferenceSmooth)

Smoothness is ensured on a weak network. The video image may be blurry, but it can be viewed smoothly with little or no stuttering.

Quality preferred (TRTCVideoQosPreferenceClear)

Video quality is ensured on a weak network. The image will be as clear as possible but will tend to stutter.

ControlMode

For the `controlMode` parameter, select **TRTCQosControlModeServer**. `TRTCQosControlModeClient` is used for internal debugging by the Tencent Cloud R&D team and should be ignored.

Common Misconceptions

1. Higher resolution is always better

A higher resolution also requires a higher bitrate. If the resolution is 1280 x 720, but the bitrate is specified as 200 Kbps, the video will be very blurry. We recommend you set parameters by referring to the Resolution and Bitrate Reference Table.

2. Higher frame rate is always better

Because the image captured by the camera is the result of physical light exposure, setting a higher frame rate does not always result in a smoother video. On the contrary, if the frame rate is too high, the quality of each video frame will be lowered because the exposure time is reduced.

3. Higher bitrate is always better

A higher bitrate also requires a higher resolution. However, for a resolution of 320 x 240, a bitrate of 1000 Kbps is a waste. We recommend you set parameters by referring to the Resolution and Bitrate Reference Table.

4. High resolution and bitrate can always be set on a Wi-Fi network

Wi-Fi network speed is generally not constant. If the device is far from the wireless router or the router channel is occupied, the Wi-Fi network may not be as fast as 4G.
The TRTC SDK provides a speed test feature, which can perform speed testing before a video call to determine the network quality based on a score.

Flutter

Last updated：2024-02-02 18:49:21

Overview

In TRTCCloud, you can adjust the video quality in the following ways:
TRTCCloud.enterRoom TRTCAppScene parameter: For selecting your application scenario.
TRTCCloud.setVideoEncoderParam: For setting encoding parameters.
TRTCCloud.setNetworkQosParam: Used for setting up network regulation policies.
This document describes how to configure these parameters to make the video quality of the TRTC SDK meet your project-specific needs.
You can also see the following demos:
Flutter:SetVideoQualityPage.dart

Supported Platforms

iOS	Android	Mac OS	Windows	Web	Electron	Flutter
✓	✓	✓	✓	✓	✓	✓

For detailed operations on how to set the screen quality on the Web, please refer to the Configuration Guide.

Room Scenario

Scenario Type	Scenario Introduction
TRTC_APP_SCENE_VIDEOCALL	Within the context of video calling, 720p and 1080p high-definition image quality is supported. A single room can accommodate up to 300 simultaneous online users, with a maximum of 50 users speaking at the same time.
TRTC_APP_SCENE_LIVE	In the context of interactive video broadcasting, the mic can be smoothly turned on/off without switching latency, with host latency as low as 300 milliseconds. Supports live streaming for hundreds of thousands of concurrent viewers, with playback delay reduced to 1000 milliseconds.

	Note: In this scenario, you need to specify the current user's role using the 'role' field in TRTCParams.
TRTC_APP_SCENE_AUDIOCALL	In the audio call context, it supports 48 kHz duplex audio calls. A single room accommodates up to 300 concurrent online users, with a maximum of 50 people speaking at once.
TRTC_APP_SCENE_VOICE_CHATROOM	In the context of interactive audio live streaming, microphones can be switched on and off smoothly without delay. The host experiences a low latency of fewer than 300 milliseconds. It accommodates hundreds of thousands concurrent viewer users, with the broadcast delay reduced to 1000 milliseconds. **Note**: In this scenario, you need to specify the current user's role using the 'role' field in TRTCParams.

TRTCVideoEncParam

Recommended configuration

Application Scenario	videoResolution	videoFps	videoBitrate
Video call (mobile)	640x360	15	550kbps
Video conferencing (primary image on macOS or Windows)	1280x720	15	1200kbps
Video conferencing (primary image on mobile device)	640x360	15	900kbps
Video conferencing (small image)	320x180	15	250kbps
Online education (teacher on macOS or Windows)	960x540	15	850kbps
Online education (teacher on iPad)	640x360	15	550kbps
Online education (student)	320x180	15	250kbps

Detailed description of fields

(int) videoResolution

The encoding resolution (TRTCCloudDef.TRTC_VIDEO_RESOLUTION_), for instance, 640 x 360, indicates the width (in pixels) x height (in pixels) of the output image.We have only predefined horizontal (landscape) resolutions in the TRTCVideoResolution enumeration where the width >= height. If you want to use a vertical (portrait) resolution, you need to set resMode to Portrait.

Note:

Because many hardware codecs only support pixel widths that are divisible by 16, the actual resolution encoded by the SDK may not be exactly the same as configured by the parameter; instead, it will be automatically corrected based on a multiple of 16. For example, the resolution 640 x 360 may be adapted to 640 x 368 inside the SDK.

(int) videoResolutionMode

This parameter designates the screen orientation resolution (TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_). Since TRTCVideoResolution only defines horizontal screen resolution, if you want to use vertical screen resolutions like 360 x 640, you'd need to set resMode as TRTCVideoResolutionModePortrait. Generally, PCs and Macs employ horizontal (Landscape) resolution, while smartphones use vertical (Portrait) resolution.

(int) videoFps

The Frame Rate (FPS) refers to the number of frames to be encoded per second. A recommended setting is 15 FPS, which assures sufficient video fluidity without compromising clarity due to an excessive number of frames per second. If you require higher smoothness, settings of 20 FPS or 25 FPS can be used. However, resist settings above 25 FPS, since the conventional frame rate for movies is 24 FPS.

(int) videoBitrate

Video Bitrate (Bitrate) refers to how much Kbit binary data the encoder outputs per second after encoding. If you set videoBitrate to 800kbps, the encoder will produce 800kbit video data per second. If stored into a file, the size of this file would amount to 800kbit, which is equivalent to 100KB, or 0.1MB.

A higher video bitrate is not always better; instead, it should be chosen appropriately based on the resolution as shown in the table below.

Resolution-bitrate reference table

Resolution Definition	Aspect Ratio	Recommended Bitrate(VideoCall)	Recommended Bitrate(LIVE)
TRTCVideoResolution_120_120	1:1	80kbps	120kbps
TRTCVideoResolution_160_160	1:1	100kbps	150kbps
TRTCVideoResolution_270_270	1:1	200kbps	300kbps
TRTCVideoResolution_480_480	1:1	350kbps	525kbps
TRTCVideoResolution_160_120	4:3	100kbps	150kbps
TRTCVideoResolution_240_180	4:3	150kbps	225kbps
TRTCVideoResolution_280_210	4:3	200kbps	300kbps

TRTCVideoResolution_320_240	4:3	250kbps	375kbps
TRTCVideoResolution_400_300	4:3	300kbps	450kbps
TRTCVideoResolution_480_360	4:3	400kbps	600kbps
TRTCVideoResolution_640_480	4:3	600kbps	900kbps
TRTCVideoResolution_960_720	4:3	1000kbps	1500kbps
TRTCVideoResolution_160_90	16:9	150kbps	250kbps
TRTCVideoResolution_256_144	16:9	200kbps	300kbps
TRTCVideoResolution_320_180	16:9	250kbps	400kbps
TRTCVideoResolution_480_270	16:9	350kbps	550kbps
TRTCVideoResolution_640_360	16:9	550kbps	900kbps
TRTCVideoResolution_960_540	16:9	850kbps	1300kbps
TRTCVideoResolution_1280_720	16:9	1200kbps	1800kbps
TRTCVideoResolution_1920_1080	16:9	2000kbps	3000kbps

TRTCNetworkQosParam

QosPreference

In an environment where network bandwidth is ample, clarity and fluidity can be balanced. However, when a user's network conditions are not optimal, the decision needs to be made whether to prioritize clarity or fluidity. This designation can be made through the preference parameter within TRTCNetworkQosParam.

Smoothness preferred (TRTCVideoQosPreferenceSmooth)

When users experience a weak network, the display could turn blurry and may contain many mosaics, but smoothness can be maintained with minimal or no stutter.

Quality preferred (TRTCVideoQosPreferenceClear)

When users are constrained by a weak network, the image will strive to remain as clear as possible, but stuttering might be a frequent occurrence.

ControlMode

Select the **TRTCQosControlModeServer** for the controlMode parameter. The TRTCQosControlModeClient is used by the Tencent Cloud Research and Development team for internal debugging purposes, please do not focus on it.

Common Misconceptions

1. **Is a higher resolution better?**

A higher resolution necessitates a higher bit rate for support. If a resolution of 1280 x 720 is chosen, but the bit rate is specified as 200kbps, the picture will contain a substantial blur. It is recommended to refer to the Resolution-bitrate reference table while making adjustments.

Rotating Videos
Android, iOS, Windows, and macOS

Last updated：2023-10-09 17:12:27

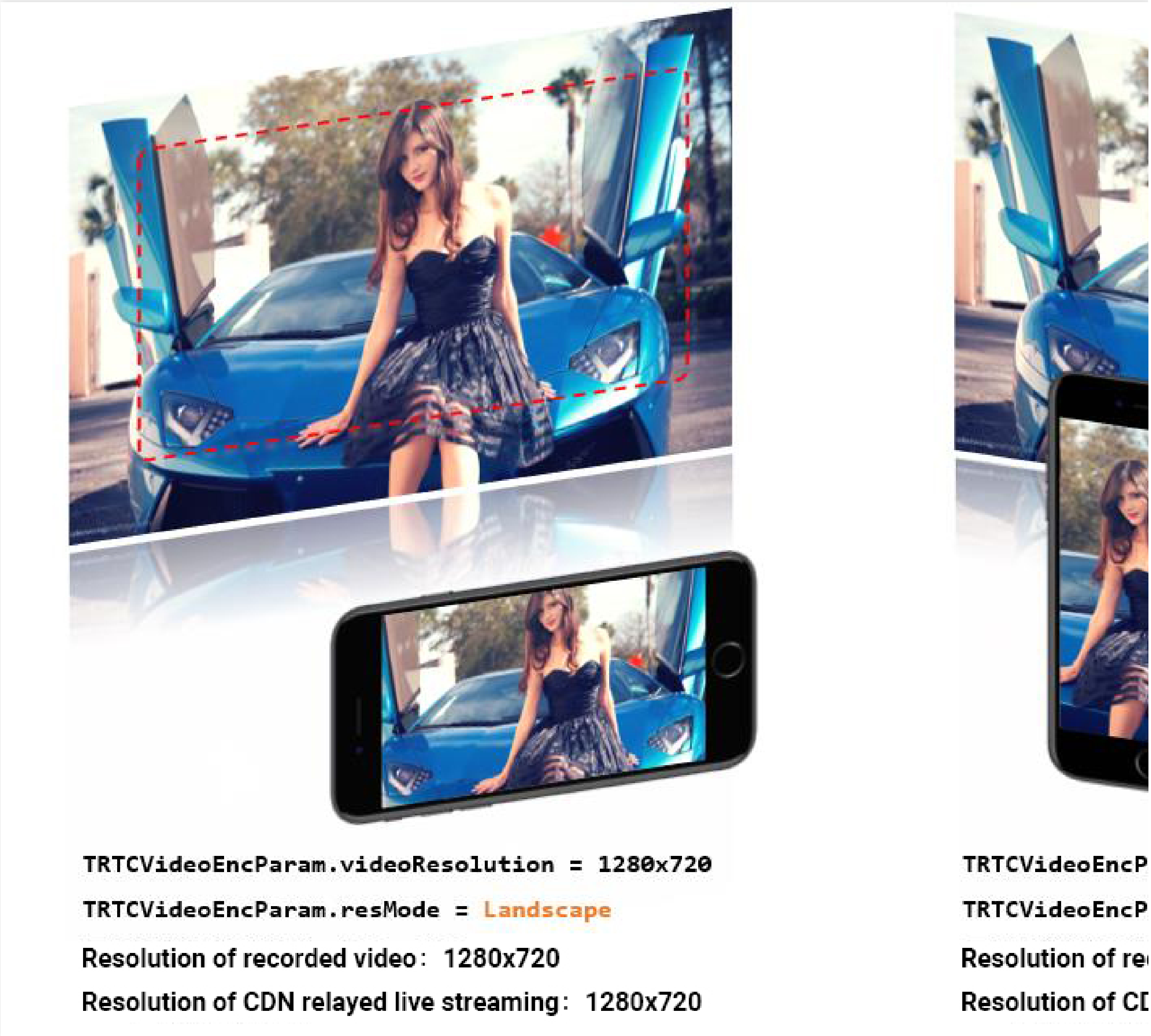
Overview

Mobile live streaming uses mainly the portrait mode, but TRTC supports both the landscape and portrait modes, making it necessary to implement different page orientation logics. This document introduces you to the following:

How to implement the portrait mode, for example, for video calls like those on WeChat

How to implement the landscape mode, for example, for group communication applications such as Zoom

How to customize settings for the rotation and rendering mode of the local image and remote images.



Supported Platforms

iOS	Android	macOS	Windows	Electron	Web
✓	✓	✓	✓	✓	×

Portrait Mode

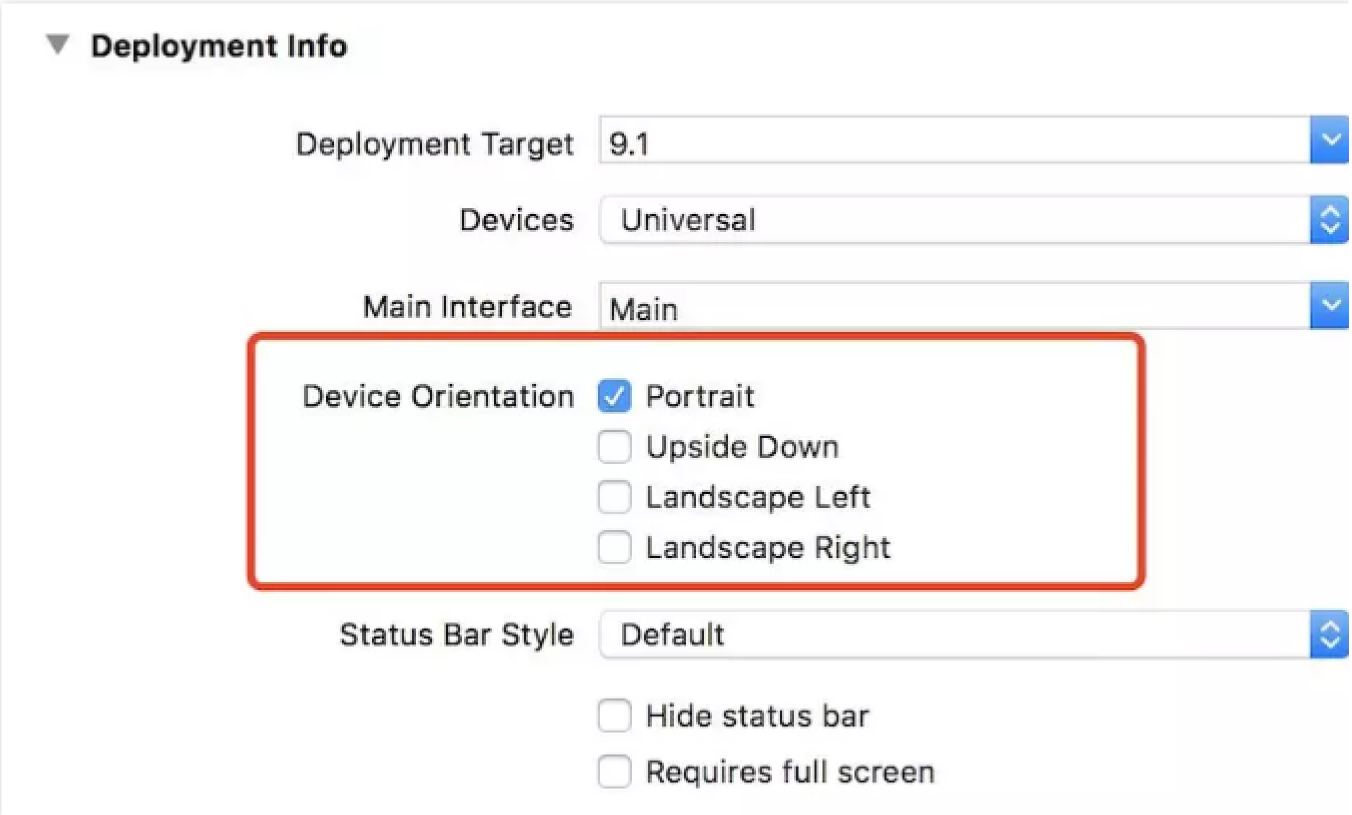
To deliver an experience similar to that of WeChat video calls, you need to do two things.

1. Set the orientation of your app to portrait.

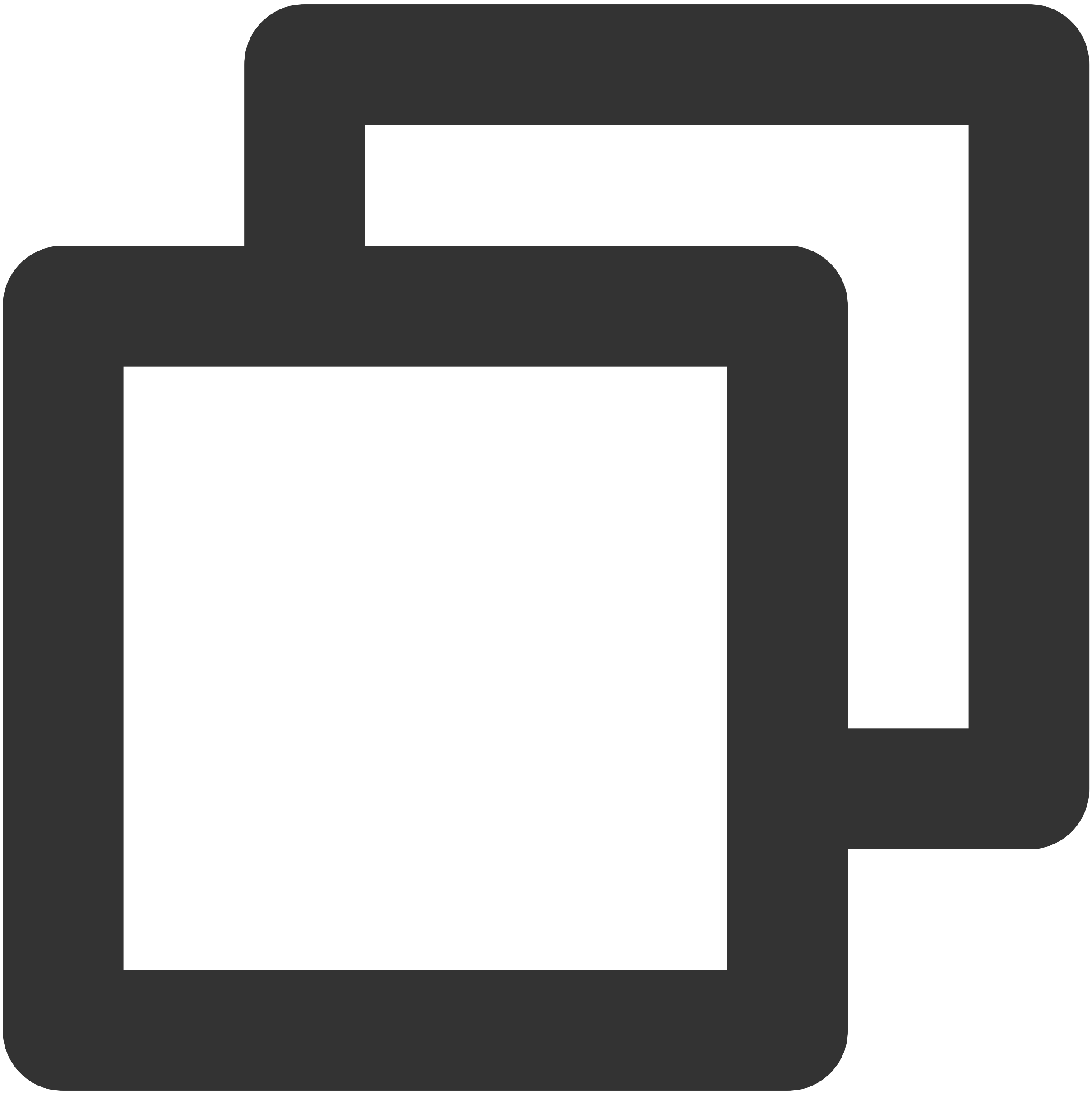
iOS

Android

Set the page orientation in Xcode > **General** > **Deployment Info** > **Device Orientation**.



Alternatively, use the `supportedInterfaceOrientationsForWindow` method in `Appdelegate` .



```
- (UIInterfaceOrientationMask)application:(UIApplication *)application
    supportedInterfaceOrientationsForWindow:(UIWindow *)window
{

    return  UIInterfaceOrientationMaskPortrait ;

}
```

explain

This CSDN article offers a detailed guide on page orientation and adaptation on iOS for developers.

Set the `screenOrientation` attribute of the activity element to `portrait` :

```
<activity android:name=".trtc.TRTCMainActivity"  android:launchMode="singleTask" an
          android:screenOrientation="portrait" />
```

2. Set the orientation of the SDK to portrait.

When you use the `setVideoEncoderParam` API of `TRTCCloud` to set video encoding parameters, set `resMode` to `TRTCVideoResolutionModePortrait`.

Below is the sample code:

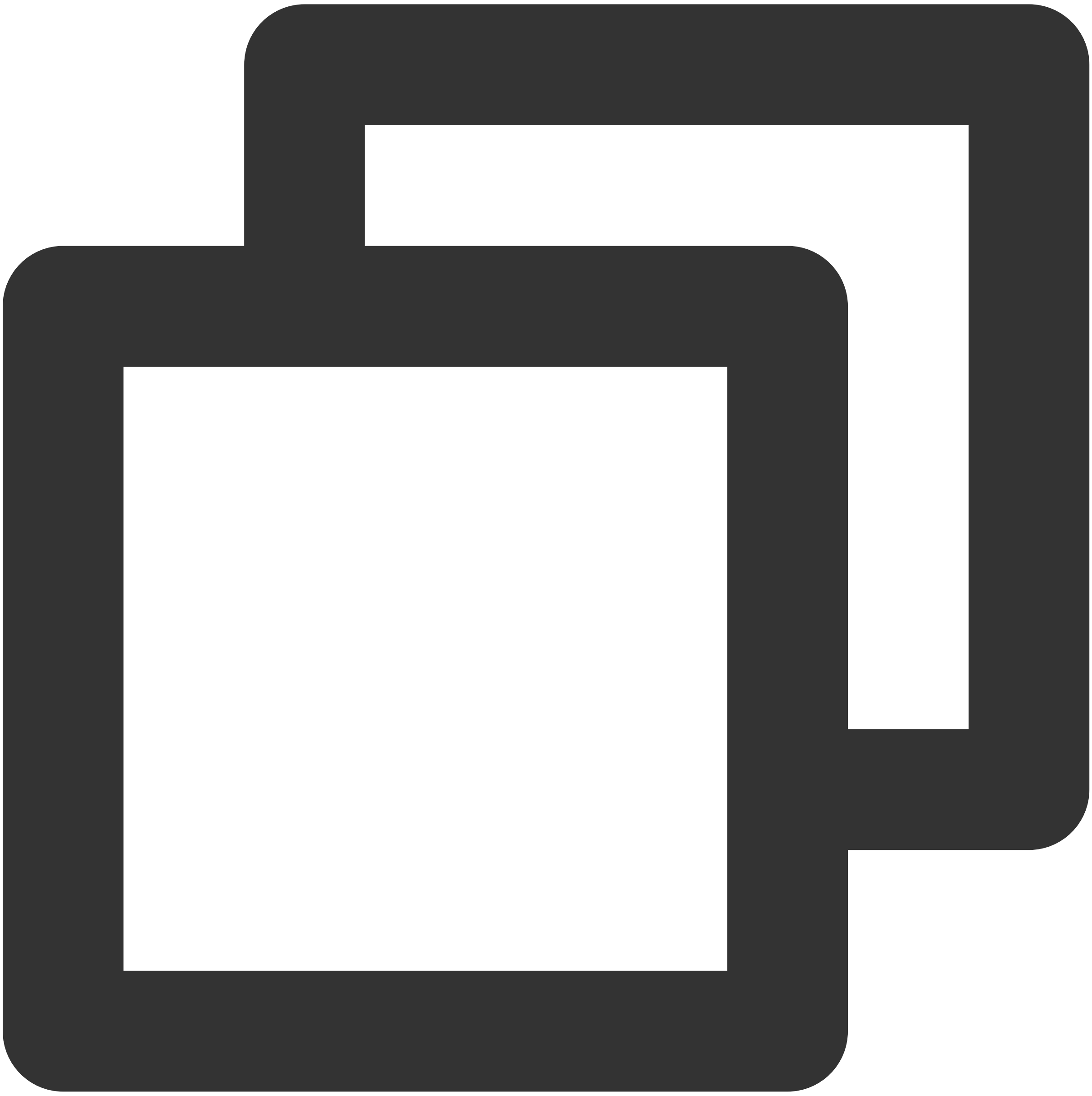
iOS

Android

```
TRTCVideoEncParam* encParam = [TRTCVideoEncParam new];
encParam.videoResolution = TRTCVideoResolution_640_360;
encParam.videoBitrate = 600;
encParam.videoFps = 15;
```

```
encParam.resMode = TRTCVideoResolutionModePortrait; // Set the resolution mode to p

[trtc setVideoEncoderParam: encParam];
```



```
TRTCCloudDef.TRTCVideoEncParam encParam = new TRTCCloudDef.TRTCVideoEncParam();
encParam.videoResolution = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_640_360;
encParam.videoBitrate = 600;
encParam.videoFps = 15;
encParam.videoResolutionMode = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT; //
trtc.setVideoEncoderParam(encParam);
```


Landscape Mode

The steps to implement the landscape mode for your app are similar to the steps of implementing the portrait mode, except that different values are used for the parameters in step 1 and step 2.

In particular, regarding the value of `resMode` in `TRTCVideoEncParam` in step 2,

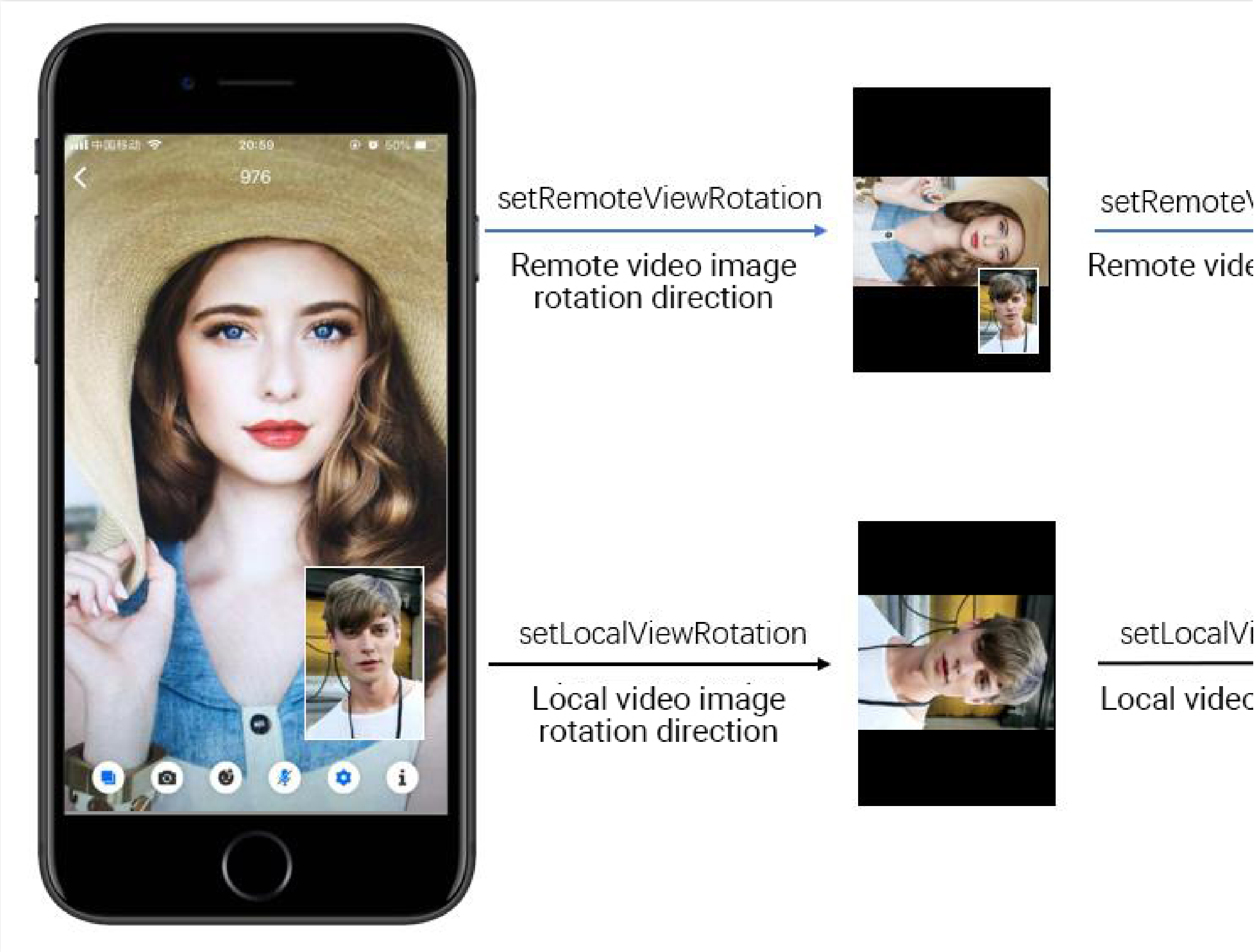
on iOS, set it to `TRTCVideoResolutionModeLandscape` .

on Android, set it to `TRTC_VIDEO_RESOLUTION_MODE_LANDSCAPE` .

Custom Settings

The TRTC SDK provides different APIs for the setting of the rotation and rendering mode of the local image and remote images.

API	Description	Remarks
setLocalViewRotation	Set the clockwise rotation of the local image preview	Rotate 90, 180, or 270 degrees clockwise
setLocalViewFillMode	Set the rendering mode of the local image preview	Crop the image or fill the blank space with black bars
setRemoteViewRotation	Set the clockwise rotation of remote video images	Rotate 90, 180, or 270 degrees clockwise
setRemoteViewFillMode	Set the rendering mode of remote video images	Crop the image or fill the blank space with black bars
setVideoEncoderRotation	Set the clockwise rotation of encoded images	Rotate 90, 180, or 270 degrees clockwise



GSensorMode

For adaptation during video recording and CDN live streaming, the TRTC SDK provides a simple gravity-sensing adaptation feature, which you can enable using the `setGSensorMode` API of `TRTCCloud` .

The feature supports 90-degrees, 180-degrees and 270-degrees adaptive rotation. This means that when a user's phone is turned, the orientation of the user's image seen by remote users remains the same. Since the feature is achieved through encoder-based rotation adjustment, adaptive rotation is also possible for recorded videos and videos played via HTML5 players.

notice

Another way to achieve adaptive rotation is by embedding the gravity direction of a video in the information of each video frame, and adjusting the rotation degree of the video at the viewer end. This scheme requires the introduction of additional transcoding resources to adjust the orientation of recorded videos as expected and is therefore not recommended.

Electron

Last updated：2023-09-28 11:49:15

You can customize settings for the rotation and rendering modes of local and remote video images.

Custom Control of Local Image

You can set local rendering parameters by calling setLocalRenderParams.

```
import TRTCCloud, {
  TRTCRenderParams, TRTCVideoRotation, TRTCVideoFillMode,
  TRTCVideoMirrorType
} from 'trtc-electron-sdk';
const trtcCloud = new TRTCCloud();
const param = new TRTCRenderParams(
  TRTCVideoRotation.TRTCVideoRotation90,
  TRTCVideoFillMode.TRTCVideoFillMode_Fill,
  TRTCVideoMirrorType.TRTCVideoMirrorType_Enable
);
```

```
trtcCloud.setLocalRenderParams(param);
const localUserDom = document.querySelector('local-user');
trtcCloud.startLocalPreview(localUserDom);
```

Custom Control of Remote Image

You can set remote rendering parameters by calling setRemoteRenderParams.

```
import TRTCCloud, {
```

```
  TRTCRenderParams, TRTCVideoRotation, TRTCVideoFillMode,
  TRTCVideoMirrorType, TRTCVideoStreamType
} from 'trtc-electron-sdk';
const trtcCloud = new TRTCCloud();
const param = new TRTCRenderParams(
  TRTCVideoRotation.TRTCVideoRotation180,
  TRTCVideoFillMode.TRTCVideoFillMode_Fill,
  TRTCVideoMirrorType.TRTCVideoMirrorType_Disable
);

const remoteUserId = 'remoteUser';
trtcCloud.setRemoteRenderParams(remoteUserId, TRTCVideoStreamType.TRTCVideoStreamTy
const remoteUserDom = document.querySelector('remote-user');
trtcCloud.startRemoteView(remoteUserId, remoteUserDom, TRTCVideoStreamType.TRTCVide
```

Web

Last updated：2023-09-28 11:50:43

This section mainly introduces how to control the mirror and fill mode of video rendering through TRTC Web SDK

Notes：

This tutorial is based on the 5.x TRTC Web SDK. If you are using the 4.x version SDK, you can refer to the objectFit attribute of stream.play.

Mirror

There are 2 ways to control the rendering mirror effects of local and remote videos, respectively are trtc.startLocalVideo({ option: { mirror: true } }) and trtc.startRemoteVideo({ option: { mirror: true }}).

```
// Local camera rendering image, the default is true
await trtc.startLocalVideo({ option: { mirror: true }});

// Dynamically updating parameters
await trtc.updateLocalVideo({ option: { mirror: false }});

trtc.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, async ({ userId, streamType }) => {
  await trtc.startRemoteVideo({
    userId,
    streamType,
    // You need to pre-place the video container in the DOM.
```

```
      // It is recommended to use '${userId}_${streamType}' as the element id.
      view: `${userId}_${streamType}`,
      // Mirror the remote video. The default value is false
      option: { mirror: true }
      });

    // Dynamically updating parameters
    await trtc.updateRemoteVideo({ userId, streamType, option: { mirror: false }})
  });
```

Notice：

The mirroring effect is only used for rendering, and there is no mirroring effect on the actual encoded or decoded picture. You can use the custom collection method of canvas to flip the canvas to achieve the effect of coding mirror.

Fill

There are 2 ways to control the rendering fill mode of local and remote videos, respectively are trtc.startLocalVideo({ option: { fillMode: 'cover' } }) and trtc.startRemoteVideo({ option: { fillMode: 'cover' }}).

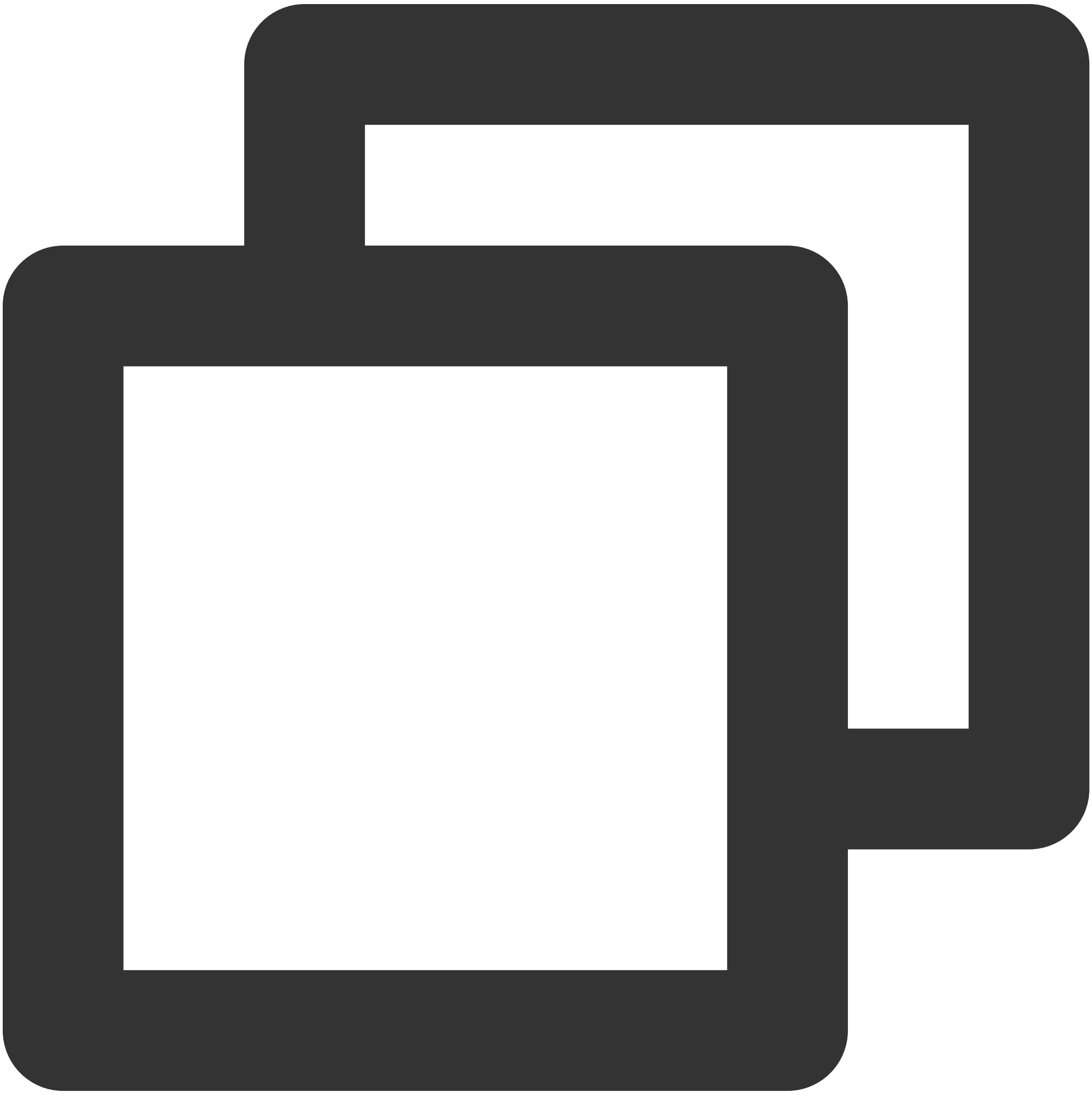
Parameters：

`contain` Keep the aspect ratio and display the full picture in the target container. If the aspect ratio does not match the target container, it will be filled with a black edge. You are advised to use this parameter for screen sharing.

`cover` A default value, retaining the aspect ratio, is displayed in the target container, and if the aspect ratio does not match the target container, the screen is cropped to fill the entire target container.

`fill` The aspect ratio is not retained and is displayed in the target container. If the aspect ratio does not match the target container, the screen is stretched to fill the entire template container.

Refer to: CSS object-fit .



```
// Local camera fill mode, default is cover
await trtc.startLocalVideo({ option: { fillMode: 'cover' }});

// Dynamically updating parameters
await trtc.updateLocalVideo({ option: { fillMode: 'contain' }});

trtc.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, async ({ userId, streamType }) => {
  await trtc.startRemoteVideo({
    userId,
    streamType,
    // You need to pre-place the video container in the DOM.
```

```
    // It is recommended to use '${userId}_${streamType}' as the element id.
    view: `${userId}_${streamType}`,
    option: { fillMode: 'contain' }
    });

  // Dynamically updating parameters
  await trtc.updateRemoteVideo({ userId, streamType, option: { fillMode: 'cover' }}
});
```


Flutter

Last updated：2024-02-02 18:50:53

Overview

Unlike the ubiquitous vertical screen experience of mobile live streaming, Tencent Real-Time Communication (TRTC) must accommodate both landscape and portrait viewing modes, thereby requiring extensive handling of screen orientation. This article primarily discusses:

The implementation of the portrait mode pattern, for instance: WeChat's video calling is a typical example of the portrait experience pattern.

The execution of the landscape mode pattern, for example: Multi-person audio-video room applications (like Little Fish Easy Connection) typically adopt the landscape mode pattern.

How to customize the control of the rotation direction and fill pattern for both local and remote images.

Supported Platforms

iOS	Android	Mac OS	Windows	Electron	Web
✓	✓	✓	✓	✓	×

Portrait pattern

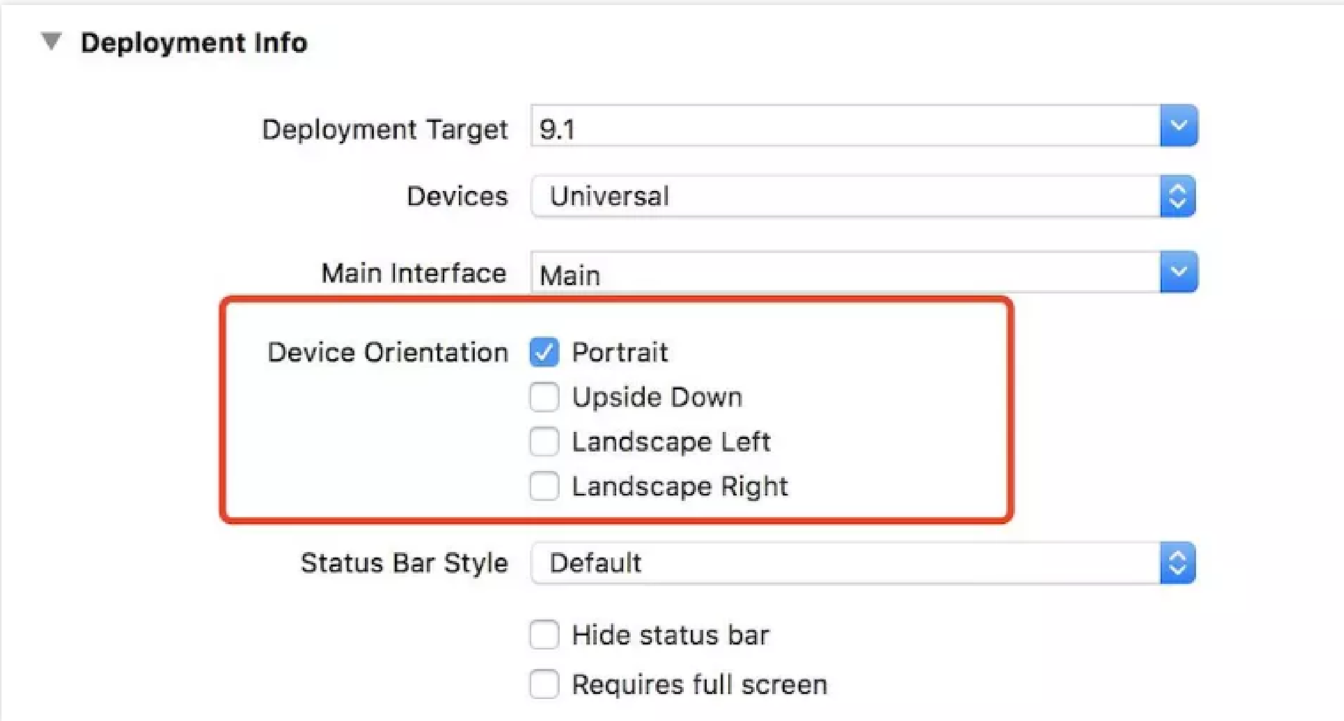
To realize a WeChat-like video call experience pattern, two tasks must be performed:

1. Configure the App's UI to display in portrait mode

iOS platform

Android platform

You can directly set this in XCode's **General** > **Deployment Info** > **Device Orientation**:



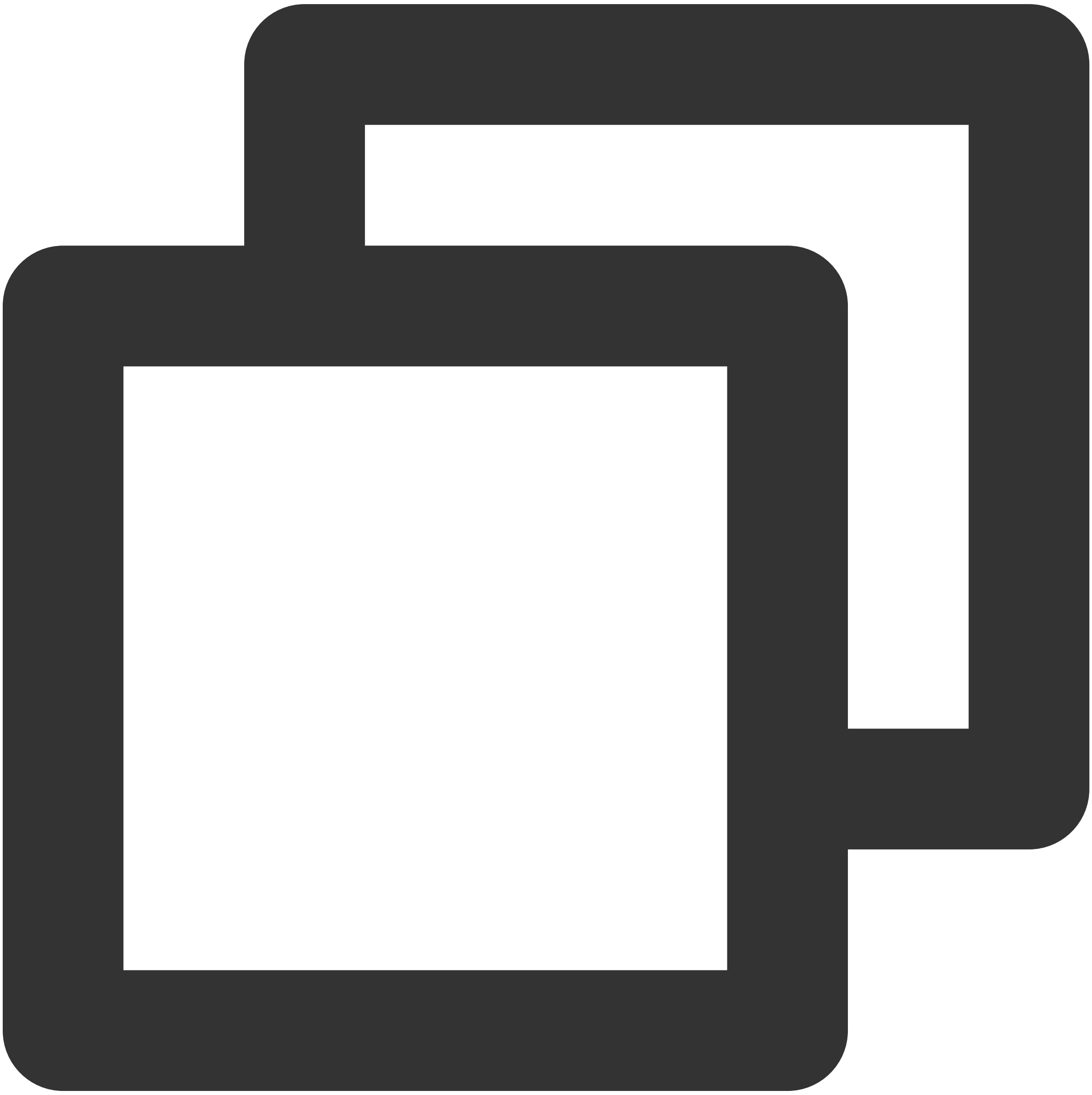
By setting the `screenOrientation` attribute of the activity to portrait, one can specify that this interface is in portrait pattern

```
<activity android:name=".trtc.TRTCMainActivity"  android:launchMode="singleTask" an
          android:screenOrientation="portrait" />
```

2. Configuration SDK utilizing vertical screen resolution

Whilst utilizing the TRTCCloud's setVideoEncoderParam setting for video coding parameters, specifying videoResolutionMode as `TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT` will suffice.

Below is the exemplar code:



```
trtcCloud.setVideoEncoderParam(TRTCVideoEncParam(
    videoFps: 15,
    videoResolution: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_640_360,
    videoBitrate: 600,
    videoResolutionMode:
        TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT));
```

Landscape mode

If you wish for the App to have a landscape orientation experience, the work you need to carry out is similar to that of the portrait pattern. You simply need to adjust the parameters in the first and second steps accordingly.
Specifically in the second step, the value of videoResolutionMode in TRTCVideoEncParam should be: `TRTC_VIDEO_RESOLUTION_MODE_LANDSCAPE` .

Tailored Control

The TRTC SDK provides interface functions to manipulate both the local and remote screen's rotation direction and fill pattern:

Interface Function	Functionality	Annotation Note
setVideoEncoderRotation	Establishing the clockwise rotation angle of the encoder output display	Supports rotation in two directions: 0 and 180 degrees clockwise

GSensorMode

Given the various compatibility issues involved in screen rotation, recording, and CDN live streaming, TRTC SDK only provides a simple gravity-sensing adaptive function. You can enable this via the `setGSensorMode` interface of TRTCCloud.
This function supports 90-degree, 180-degree and 270-degree rotation adaptation. That is, when the user's own phone rotates, the orientation of the image seen by others remains unchanged. Furthermore, this adaptation is based on adjustments to the encoder's direction. Therefore, recorded videos, as well as images viewed on mini-programs and H5 end, can maintain the original orientation.
Note:
Another implementation of gravity-sensing adaptation involves encoding the gravity direction of each video frame, and then adaptively adjusting the rendering direction at the end of the remote user. However, this embodiment requires the introduction of additional transcoding resources to solve the problem of keeping the direction of the recorded video consistent with the expected video direction. Therefore, it is not recommended.

07.FAQs

FAQs for Beginners

Last updated：2023-09-28 11:51:16

What is UserSig?

`UserSig` is a security signature designed by Tencent Cloud to prevent attackers from accessing your Tencent Cloud account.

Currently, Tencent Cloud services including TRTC, Chat, and MLVB all use this security mechanism. Whenever you want to use these services, you must provide three key pieces of information, i.e. `SDKAppID` , `UserID` , and `UserSig` in the initialization or login function of the corresponding SDK.

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

See the figure below for how `UserSig` is calculated. Basically, it involves hashing crucial information such as `SDKAppID` , `UserID` , and `ExpireTime` .

```
// UserSig formula, in which `secretkey` is the key used to calculate UserSig

usersig = hmacsha256(secretkey, (userid + sdkappid + currtime + expire +
                                 base64(userid + sdkappid + currtime + expire)))
```

Note

`currtime` is the current system time and `expire` the expiration time of the signature.

For detailed directions on how to calculate and get `UserSig`, please see UserSig.

How many rooms can there be in TRTC at the same time?

There can be up to 4,294,967,294 concurrent rooms in TRTC. No limits are set on the number of non-concurrent rooms.

How long is the average delay in TRTC?

The average end-to-end delay of TRTC around the globe is less than 300 ms.

Does TRTC support screen sharing on PCs?

Yes. For details, see the following documents:
Real-Time Screen Sharing (Windows)
Real-Time Screen Sharing (macOS)
Real-Time Screen Sharing (Web)
For more information on the screen sharing APIs, please see Client APIs > All Platforms (C++) > Overview or Client APIs > Electron > Overview.

What platforms does TRTC support?

TRTC supports platforms including iOS, Android, Windows (C++), Windows (C#), macOS, web, and Electron. For more information, see Supported Platforms.

How many people can there be in a TRTC call?

In call scenarios, each room can accommodate up to 300 concurrent users, and up to 50 of them can turn on their cameras or mics.
In live streaming scenarios, each room can accommodate up to 100,000 concurrent users, and up to 50 of them can be assigned the anchor role and turn on their cameras or mics.

How do I start a live streaming session in TRTC?

TRTC offers a dedicated low-latency interactive live streaming solution that allows up to 100,000 participants with co-anchoring latency kept as low as 200 ms and watch latency below 1s. It adapts excellently to poor network conditions and is optimized for the complicated mobile network environments.
For detailed directions, please see Live Streaming Mode.

What roles are supported during live streaming in TRTC? How do they differ from each other?

The live streaming scenarios (`TRTCAppSceneLIVE` and `TRTCAppSceneVoiceChatRoom`) support two roles: `TRTCRoleAnchor` (anchor) and `TRTCRoleAudience` (audience). An anchor can both send and receive audio/video data, but audience can only receive and play back others' data. You can call `switchRole()` to switch roles.

Can I kick a user out, forbid a user to speak, or mute a user in a TRTC room?

Yes, you can.
To enable the features through simple signaling operations, use `sendCustomCmdMsg` , the custom signaling API of TRTC, to define your own control signaling, and users who receive the message will perform the action expected. For example, to kick out a user, just define a kick-out signaling, and the user receiving it will exit the room.
If you want to implement a more comprehensive operation logic, we recommend that you use Instant Messaging to map the TRTC room to an Chat group and enable the features via the sending/receiving of custom messages in the group.

Can TRTC pull and play back streams through CDN?

Yes. For details, please see CDN Relayed Live Streaming.

Does TRTC support Swift integration on iOS?

Yes. Just integrate the SDK in the same steps as you do a third-party library or by following the steps in Demo Quick Start (iOS & macOS).

What browsers does the SDK for web support?

It is well supported by Chrome (desktop) and Safari (desktop and mobile) but poorly or not supported by other platforms such as browsers on Android. For more information, please see Client APIs > Supported Platforms.
You can open WebRTC Support Level Test in a browser to test whether the environment fully supports WebRTC.

What do the errors `NotFoundError` , `NotAllowedError` , `NotReadableError` , `OverConstrainedError` , and `AbortError` found in the log of TRTC SDK for web mean?

Error	Description	Suggested Solution

NotFoundError	The media (audio, video, or screen sharing) of the request parameters are not found. For example, this error occurs if the PC has no cameras but the browser requests a video stream.	Remind users to check devices such as cameras and mics before making a call.
NotAllowedError	The user has rejected the request of the current browser instance to access the camera/mic or share screens.	Remind the user that audio/video calls are not possible without camera/mic access.
NotReadableError	The user has granted access to the requested device, but it is still inaccessible due to a hardware, browser or webpage error.	Handle the error according to the error message returned, and send this message to the user: “The camera/mic cannot be accessed. Please make sure that no other applications are requesting access and try again.”
OverConstrainedError	The `cameraId/microphoneId` value is invalid.	Make sure that the `cameraId/microphoneId` value passed in is valid.
AbortError	The device cannot be accessed due to an unknown reason.	-

For more information, please see initialize.

How do I check whether TRTC SDK for web can get the device (camera/mic) list?

1. Check whether the browser can access the devices:
Open the console with the browser and enter `navigator.mediaDevices.enumerateDevices()` to see if the device list can be obtained.
Normally, a promise containing an array of `MediaDeviceInfo` objects will be returned, each object corresponding to an available media device.
If the SDK fails to enumerate the devices, a rejected promise will be returned, indicating that the browser fails to detect any devices. You need to check the browser or devices.
2. If the device list can be obtained, enter `navigator.mediaDevices.getUserMedia({ audio: true, video: true })` to see if the `MediaStream` object can be returned. If it is not returned, it indicates that the browser failed to obtain any data. You need to check your browser configuration.

How do live streaming, interactive live streaming, TRTC, and relayed live streaming differ from and relate to each other?

Live streaming (keywords: one-to-many, RTMP/HLS/HTTP-FLV, CDN)
Live streaming consists of the push end, the playback end, and the cloud live streaming service. Streams are pushed over the universal protocol RTMP, delivered through CDNs, and can be watched over protocols including RTMP, HTTP-FLV, or HLS (for HTML5).
Interactive live streaming (keywords: co-anchoring, anchor competition)
In interactive live streaming, audience can co-anchor with anchors and anchors from different rooms can compete with each other.
Real-time communication (keywords: multi-person interaction, UDP-based proprietary protocol, low latency)
The main application scenarios for TRTC (Tencent Real-Time Communication) are audio/video interaction and low-latency live streaming. It uses a UDP-based proprietary protocol and can keep the latency as low as 100 ms. Typical applications include zoom meeting, FaceTime, and online group classes. TRTC is supported by mainstream platforms including iOS, Android, and Windows and can communicate over WebRTC. It supports relaying streams to CDNs through on-cloud stream mixing.
Relayed live streaming (keywords: on-cloud stream mixing, RTC relayed live streaming, CDN)
The relayed live streaming technology replicates multiple streams in a low-latency co-anchoring room and mixes them into one stream in the cloud before pushing it to a live streaming CDN for delivery and playback.

How do I view my call duration and usage?

You can find the information on the Usage Statistics page of the TRTC console.

How do I fix stutter?

You can check call quality by room ID or user ID in Monitoring Dashboard in the TRTC console.
Check the send and receive statistics from the recipient’s perspective.
Check the send and receive packet loss. High packet loss suggest that the stutter may be caused by unstable network connections.
Check the frame rate and CPU usage. Both low frame rates and high CPU usage can cause stutter.

How do I fix low-quality, blurry and pixelated videos?

Resolution is mainly associated with bitrate. Check whether the bitrate is set too low. Pixelation tends to occur when resolution is high but bitrate low.
TRTC dynamically adjusts bitrate and resolution based on network conditions according to its on-cloud QoS control policy. It reduces the bitrate in case of poor network connections, which leads to decreased definition.

Check whether the `VideoCall` or `Live` mode is used during room entry. As the `VideoCall` mode is designed for calls and features low latency and smoothness, it tends to sacrifice video quality for smoothness when network connections are poor. We recommend that you use the `Live` mode for application scenarios with high requirements on video quality.

How do I view the latest version number of the SDK?

In the case of automatic loading, `latest.release` will load the latest version automatically. You don’t need to modify the version number. For detailed instructions on integration, please see SDK Quick Integration.
You can find the latest version number of the SDK on the release notes page.
For iOS & Android, please see Release Notes (App).
For web, please see Release Notes (Web).
For Electron, please see Release Notes (Electron).

API Reference Manual
iOS and macOS
Overview

Last updated：2024-06-06 15:26:14

API OVERVIEW

Create Instance And Event Callback

FuncList	DESC
sharedInstance	Create TRTCCloud instance (singleton mode)
destroySharedInstance	Terminate TRTCCloud instance (singleton mode)
addDelegate:	Add TRTC event callback
removeDelegate:	Remove TRTC event callback
delegateQueue	Set the queue that drives the TRTCCloudDelegate event callback

Room APIs

FuncList	DESC
enterRoom:appScene:	Enter room
exitRoom	Exit room
switchRole:	Switch role
switchRole:privateMapKey:	Switch role(support permission credential)
switchRoom:	Switch room
connectOtherRoom:	Request cross-room call
disconnectOtherRoom	Exit cross-room call

setDefaultStreamRecvMode:video:	Set subscription mode (which must be set before room entry for it to take effect)
createSubCloud	Create room subinstance (for concurrent multi-room listen/watch)
destroySubCloud:	Terminate room subinstance
updateOtherRoomForwardMode:	

CDN APIs

FuncList	DESC
startPublishing:type:	Start publishing audio/video streams to Tencent Cloud CSS CDN
stopPublishing	Stop publishing audio/video streams to Tencent Cloud CSS CDN
startPublishCDNStream:	Start publishing audio/video streams to non-Tencent Cloud CDN
stopPublishCDNStream	Stop publishing audio/video streams to non-Tencent Cloud CDN
setMixTranscodingConfig:	Set the layout and transcoding parameters of On-Cloud MixTranscoding
startPublishMediaStream:encoderParam:mixingConfig:	Publish a stream
updatePublishMediaStream:publishTarget:encoderParam:mixingConfig:	Modify publishing parameters
stopPublishMediaStream:	Stop publishing

Video APIs

FuncList	DESC
startLocalPreview:view:	Enable the preview image of local camera (mobile)
startLocalPreview:	Enable the preview image of local camera

	(desktop)
updateLocalView:	Update the preview image of local camera
stopLocalPreview	Stop camera preview
muteLocalVideo:mute:	Pause/Resume publishing local video stream
setVideoMuteImage:fps:	Set placeholder image during local video pause
startRemoteView:streamType:view:	Subscribe to remote user's video stream and bind video rendering control
updateRemoteView:streamType:forUser:	Update remote user's video rendering control
stopRemoteView:streamType:	Stop subscribing to remote user's video stream and release rendering control
stopAllRemoteView	Stop subscribing to all remote users' video streams and release all rendering resources
muteRemoteVideoStream:streamType:mute:	Pause/Resume subscribing to remote user's video stream
muteAllRemoteVideoStreams:	Pause/Resume subscribing to all remote users' video streams
setVideoEncoderParam:	Set the encoding parameters of video encoder
setNetworkQosParam:	Set network quality control parameters
setLocalRenderParams:	Set the rendering parameters of local video image
setRemoteRenderParams:streamType:params:	Set the rendering mode of remote video image
enableEncSmallVideoStream:withQuality:	Enable dual-channel encoding mode with big and small images
setRemoteVideoStreamType:type:	Switch the big/small image of specified remote user
snapshotVideo:type:sourceType:	Screencapture video
setPerspectiveCorrectionWithUser:srcPoints:dstPoints:	Sets perspective correction coordinate points.
setGravitySensorAdaptiveMode:	Set the adaptation mode of gravity sensing (version 11.7 and above)

Audio APIs

FuncList	DESC
startLocalAudio:	Enable local audio capturing and publishing
stopLocalAudio	Stop local audio capturing and publishing
muteLocalAudio:	Pause/Resume publishing local audio stream
muteRemoteAudio:mute:	Pause/Resume playing back remote audio stream
muteAllRemoteAudio:	Pause/Resume playing back all remote users' audio streams
setAudioRoute:	Set audio route
setRemoteAudioVolume:volume:	Set the audio playback volume of remote user
setAudioCaptureVolume:	Set the capturing volume of local audio
getAudioCaptureVolume	Get the capturing volume of local audio
setAudioPlayoutVolume:	Set the playback volume of remote audio
getAudioPlayoutVolume	Get the playback volume of remote audio
enableAudioVolumeEvaluation:withParams:	Enable volume reminder
startAudioRecording:	Start audio recording
stopAudioRecording	Stop audio recording
startLocalRecording:	Start local media recording
stopLocalRecording	Stop local media recording
setRemoteAudioParallelParams:	Set the parallel strategy of remote audio streams
enable3DSpatialAudioEffect:	Enable 3D spatial effect
updateSelf3DSpatialPosition	Update self position and orientation for 3D spatial effect
updateRemote3DSpatialPosition:	Update the specified remote user's position for 3D spatial effect

set3DSpatialReceivingRange:range:	Set the maximum 3D spatial attenuation range for userId's audio stream

Device management APIs

FuncList	DESC
getDeviceManager	Get device management class (TXDeviceManager)

Beauty filter and watermark APIs

FuncList	DESC
getBeautyManager	Get beauty filter management class (TXBeautyManager)
setWatermark:streamType:rect:	Add watermark

Background music and sound effect APIs

FuncList	DESC
getAudioEffectManager	Get sound effect management class (TXAudioEffectManager)
startSystemAudioLoopback	Enable system audio capturing(iOS not supported)
stopSystemAudioLoopback	Stop system audio capturing(iOS not supported)
setSystemAudioLoopbackVolume:	Set the volume of system audio capturing

Screen sharing APIs

FuncList	DESC
startScreenCaptureInApp:encParam:	Start in-app screen sharing (for iOS 13.0 and above only)
startScreenCaptureByReplaykit:encParam:appGroup:	Start system-level screen sharing (for iOS 11.0 and above only)

startScreenCapture:streamType:encParam:	Start screen sharing
stopScreenCapture	Stop screen sharing
pauseScreenCapture	Pause screen sharing
resumeScreenCapture	Resume screen sharing
getScreenCaptureSourcesWithThumbnailSize:iconSize:	Enumerate shareable screens and windows (for macOS only)
selectScreenCaptureTarget:rect:capturesCursor:highlight:	Select the screen or window to share (for macOS only)
setSubStreamEncoderParam:	Set the video encoding parameters of screen sharing (i.e., substream) (for desktop and mobile systems)
setSubStreamMixVolume:	Set the audio mixing volume of screen sharing (for desktop systems only)
addExcludedShareWindow:	Add specified windows to the exclusion list of screen sharing (for desktop systems only)
removeExcludedShareWindow:	Remove specified windows from the exclusion list of screen sharing (for desktop systems only)
removeAllExcludedShareWindows	Remove all windows from the exclusion list of screen sharing (for desktop systems only)
addIncludedShareWindow:	Add specified windows to the inclusion list of screen sharing (for desktop systems only)
removeIncludedShareWindow:	Remove specified windows from the inclusion list of screen sharing (for desktop systems only)
removeAllIncludedShareWindows	Remove all windows from the inclusion list of screen sharing (for desktop systems only)

Custom capturing and rendering APIs

FuncList	DESC

enableCustomVideoCapture:enable:	Enable/Disable custom video capturing mode
sendCustomVideoData:frame:	Deliver captured video frames to SDK
enableCustomAudioCapture:	Enable custom audio capturing mode
sendCustomAudioData:	Deliver captured audio data to SDK
enableMixExternalAudioFrame:playout:	Enable/Disable custom audio track
mixExternalAudioFrame:	Mix custom audio track into SDK
setMixExternalAudioVolume:playoutVolume:	Set the publish volume and playback volume of mixed custom audio track
generateCustomPTS	Generate custom capturing timestamp
setLocalVideoProcessDelegete:pixelFormat:bufferType:	Set video data callback for third-party beauty filters
setLocalVideoRenderDelegate:pixelFormat:bufferType:	Set the callback of custom rendering for local video
setRemoteVideoRenderDelegate:delegate:pixelFormat:bufferType:	Set the callback of custom rendering for remote video
setAudioFrameDelegate:	Set custom audio data callback
setCapturedAudioFrameDelegateFormat:	Set the callback format of audio frames captured by local mic
setLocalProcessedAudioFrameDelegateFormat:	Set the callback format of preprocessed local audio frames
setMixedPlayAudioFrameDelegateFormat:	Set the callback format of audio frames to be played back by system
enableCustomAudioRendering:	Enabling custom audio playback
getCustomAudioRenderingFrame:	Getting playable audio data

Custom message sending APIs

FuncList	DESC

sendCustomCmdMsg:data:reliable:ordered:	Use UDP channel to send custom message to all users in room
sendSEIMsg:repeatCount:	Use SEI channel to send custom message to all users in room

Network test APIs

FuncList	DESC
startSpeedTest:	Start network speed test (used before room entry)
stopSpeedTest	Stop network speed test

Debugging APIs

FuncList	DESC
getSDKVersion	Get SDK version information
setLogLevel:	Set log output level
setConsoleEnabled:	Enable/Disable console log printing
setLogCompressEnabled:	Enable/Disable local log compression
setLogDirPath:	Set local log storage path
setLogDelegate:	Set log callback
showDebugView:	Display dashboard
setDebugViewMargin:margin:	Set dashboard margin
callExperimentalAPI:	Call experimental APIs

Encrypted interface

FuncList	DESC
enablePayloadPrivateEncryption:params:	Enable or disable private encryption of media streams

Error and warning events

FuncList	DESC
onError:errMsg:extInfo:	Error event callback
onWarning:warningMsg:extInfo:	Warning event callback

Room event callback

FuncList	DESC
onEnterRoom:	Whether room entry is successful
onExitRoom:	Room exit
onSwitchRole:errMsg:	Role switching
onSwitchRoom:errMsg:	Result of room switching
onConnectOtherRoom:errCode:errMsg:	Result of requesting cross-room call
onDisconnectOtherRoom:errMsg:	Result of ending cross-room call
onUpdateOtherRoomForwardMode:errMsg:	Result of changing the upstream capability of the cross-room anchor

User event callback

FuncList	DESC
onRemoteUserEnterRoom:	A user entered the room
onRemoteUserLeaveRoom:reason:	A user exited the room
onUserVideoAvailable:available:	A remote user published/unpublished primary stream video
onUserSubStreamAvailable:available:	A remote user

	published/unpublished substream video
onUserAudioAvailable:available:	A remote user published/unpublished audio
onFirstVideoFrame:streamType:width:height:	The SDK started rendering the first video frame of the local or a remote user
onFirstAudioFrame:	The SDK started playing the first audio frame of a remote user
onSendFirstLocalVideoFrame:	The first local video frame was published
onSendFirstLocalAudioFrame	The first local audio frame was published
onRemoteVideoStatusUpdated:streamType:streamStatus:reason:extrainfo:	Change of remote video status
onRemoteAudioStatusUpdated:streamStatus:reason:extrainfo:	Change of remote audio status
onUserVideoSizeChanged:streamType:newWidth:newHeight:	Change of remote video size

Callback of statistics on network and technical metrics

FuncList	DESC
onNetworkQuality:remoteQuality:	Real-time network quality statistics
onStatistics:	Real-time statistics on technical metrics
onSpeedTestResult:	Callback of network speed test

Callback of connection to the cloud

FuncList	DESC
onConnectionLost	The SDK was disconnected from the cloud
onTryToReconnect	The SDK is reconnecting to the cloud

onConnectionRecovery	The SDK is reconnected to the cloud

Callback of hardware events

FuncList	DESC
onCameraDidReady	The camera is ready
onMicDidReady	The mic is ready
onAudioRouteChanged:fromRoute:	The audio route changed (for mobile devices only)
onUserVoiceVolume:totalVolume:	Volume
onDevice:type:stateChanged:	The status of a local device changed (for desktop OS only)
onAudioDeviceCaptureVolumeChanged:muted:	The capturing volume of the mic changed
onAudioDevicePlayoutVolumeChanged:muted:	The playback volume changed
onSystemAudioLoopbackError:	Whether system audio capturing is enabled successfully (for macOS only)

Callback of the receipt of a custom message

FuncList	DESC
onRecvCustomCmdMsgUserId:cmdID:seq:message:	Receipt of custom message
onMissCustomCmdMsgUserId:cmdID:errCode:missed:	Loss of custom message
onRecvSEIMsg:message:	Receipt of SEI message

CDN event callback

FuncList	DESC
onStartPublishing:errMsg:	Started publishing to Tencent Cloud CSS CDN

onStopPublishing:errMsg:	Stopped publishing to Tencent Cloud CSS CDN
onStartPublishCDNStream:errMsg:	Started publishing to non-Tencent Cloud's live streaming CDN
onStopPublishCDNStream:errMsg:	Stopped publishing to non-Tencent Cloud's live streaming CDN
onSetMixTranscodingConfig:errMsg:	Set the layout and transcoding parameters for On-Cloud MixTranscoding
onStartPublishMediaStream:code:message:extraInfo:	Callback for starting to publish
onUpdatePublishMediaStream:code:message:extraInfo:	Callback for modifying publishing parameters
onStopPublishMediaStream:code:message:extraInfo:	Callback for stopping publishing
onCdnStreamStateChanged:status:code:msg:extraInfo:	Callback for change of RTMP/RTMPS publishing status

Screen sharing event callback

FuncList	DESC
onScreenCaptureStarted	Screen sharing started
onScreenCapturePaused:	Screen sharing was paused
onScreenCaptureResumed:	Screen sharing was resumed
onScreenCaptureStoped:	Screen sharing stopped

Callback of local recording and screenshot events

FuncList	DESC
onLocalRecordBegin:storagePath:	Local recording started
onLocalRecording:storagePath:	Local media is being recorded
onLocalRecordFragment:	Record fragment finished.

onLocalRecordComplete:storagePath:	Local recording stopped

Disused callbacks

FuncList	DESC
onUserEnter:	An anchor entered the room (disused)
onUserExit:reason:	An anchor left the room (disused)
onAudioEffectFinished:code:	Audio effects ended (disused)

Callback of custom video processing

FuncList	DESC
onRenderVideoFrame:userId:streamType:	Custom video rendering
onGLContextCreated	An OpenGL context was created in the SDK.
onProcessVideoFrame:dstFrame:	Video processing by third-party beauty filters
onGLContextDestory	The OpenGL context in the SDK was destroyed

Callback of custom audio processing

FuncList	DESC
onCapturedAudioFrame:	Audio data captured by the local mic and pre-processed by the audio module
onLocalProcessedAudioFrame:	Audio data captured by the local mic, pre-processed by the audio module, effect-processed and BGM-mixed
onRemoteUserAudioFrame:userId:	Audio data of each remote user before audio mixing
onMixedPlayAudioFrame:	Data mixed from each channel before being submitted to the system for playback
onMixedAllAudioFrame:	Data mixed from all the captured and to-be-played audio in the SDK

onVoiceEarMonitorAudioFrame:	In-ear monitoring data

Other event callbacks

FuncList	DESC
onLog:LogLevel:WhichModule:	Printing of local log

Voice effect APIs

FuncList	DESC
enableVoiceEarMonitor:	Enabling in-ear monitoring
setVoiceEarMonitorVolume:	Setting in-ear monitoring volume
setVoiceReverbType:	Setting voice reverb effects
setVoiceChangerType:	Setting voice changing effects
setVoiceVolume:	Setting speech volume
setVoicePitch:	Setting speech pitch

Background music APIs

FuncList	DESC
startPlayMusic:onStart:onProgress:onComplete:	Starting background music
stopPlayMusic:	Stopping background music
pausePlayMusic:	Pausing background music
resumePlayMusic:	Resuming background music
setAllMusicVolume:	Setting the local and remote playback volume of background music
setMusicPublishVolume:volume:	Setting the remote playback volume of a specific music track

setMusicPlayoutVolume:volume:	Setting the local playback volume of a specific music track
setMusicPitch:pitch:	Adjusting the pitch of background music
setMusicSpeedRate:speedRate:	Changing the speed of background music
getMusicCurrentPosInMS:	Getting the playback progress (ms) of background music
getMusicDurationInMS:	Getting the total length (ms) of background music
seekMusicToPosInMS:pts:	Setting the playback progress (ms) of background music
setMusicScratchSpeedRate:speedRate:	Adjust the speed change effect of the scratch disc
preloadMusic:onProgress:onError:	Preload background music
getMusicTrackCount:	Get the number of tracks of background music
setMusicTrack:track:	Specify the playback track of background music

beauty interface

FuncList	DESC
setBeautyStyle:	Sets the beauty (skin smoothing) filter algorithm.
setBeautyLevel:	Sets the strength of the beauty filter.
setWhitenessLevel:	Sets the strength of the brightening filter.
enableSharpnessEnhancement:	Enables clarity enhancement.
setRuddyLevel:	Sets the strength of the rosy skin filter.
setFilter:	Sets color filter.
setFilterStrength:	Sets the strength of color filter.
setGreenScreenFile:	Sets green screen video
setEyeScaleLevel:	Sets the strength of the eye enlarging filter.
setFaceSlimLevel:	Sets the strength of the face slimming filter.

setFaceVLevel:	Sets the strength of the chin slimming filter.
setChinLevel:	Sets the strength of the chin lengthening/shortening filter.
setFaceShortLevel:	Sets the strength of the face shortening filter.
setFaceNarrowLevel:	Sets the strength of the face narrowing filter.
setNoseSlimLevel:	Sets the strength of the nose slimming filter.
setEyeLightenLevel:	Sets the strength of the eye brightening filter.
setToothWhitenLevel:	Sets the strength of the teeth whitening filter.
setWrinkleRemoveLevel:	Sets the strength of the wrinkle removal filter.
setPounchRemoveLevel:	Sets the strength of the eye bag removal filter.
setSmileLinesRemoveLevel:	Sets the strength of the smile line removal filter.
setForeheadLevel:	Sets the strength of the hairline adjustment filter.
setEyeDistanceLevel:	Sets the strength of the eye distance adjustment filter.
setEyeAngleLevel:	Sets the strength of the eye corner adjustment filter.
setMouthShapeLevel:	Sets the strength of the mouth shape adjustment filter.
setNoseWingLevel:	Sets the strength of the nose wing narrowing filter.
setNosePositionLevel:	Sets the strength of the nose position adjustment filter.
setLipsThicknessLevel:	Sets the strength of the lip thickness adjustment filter.
setFaceBeautyLevel:	Sets the strength of the face shape adjustment filter.
setMotionTmpl:inDir:	Selects the AI animated effect pendant.
setMotionMute:	Sets whether to mute during animated effect playback.

Type definitions of audio/video devices

FuncList	DESC
onDeviceChanged:type:state:	The status of a local device changed (for desktop OS only)

Device APIs

FuncList	DESC
isFrontCamera	Querying whether the front camera is being used
switchCamera:	Switching to the front/rear camera (for mobile OS)
isCameraZoomSupported	Querying whether the current camera supports zooming (for mobile OS)
getCameraZoomMaxRatio	Getting the maximum zoom ratio of the camera (for mobile OS)
setCameraZoomRatio:	Setting the camera zoom ratio (for mobile OS)
isAutoFocusEnabled	Querying whether automatic face detection is supported (for mobile OS)
enableCameraAutoFocus:	Enabling auto focus (for mobile OS)
setCameraFocusPosition:	Adjusting the focus (for mobile OS)
isCameraTorchSupported	Querying whether flash is supported (for mobile OS)
enableCameraTorch:	Enabling/Disabling flash, i.e., the torch mode (for mobile OS)
setAudioRoute:	Setting the audio route (for mobile OS)
setExposureCompensation:	Set the exposure parameters of the camera, ranging from - 1 to 1
getDevicesList:	Getting the device list (for desktop OS)
setCurrentDevice:deviceId:	Setting the device to use (for desktop OS)
getCurrentDevice:	Getting the device currently in use (for desktop OS)
setCurrentDeviceVolume:deviceType:	Setting the volume of the current device (for desktop OS)
getCurrentDeviceVolume:	Getting the volume of the current device (for desktop OS)
setCurrentDeviceMute:deviceType:	Muting the current device (for desktop OS)
getCurrentDeviceMute:	Querying whether the current device is muted (for

	desktop OS)
enableFollowingDefaultAudioDevice:enable:	Set the audio device used by SDK to follow the system default device (for desktop OS)
startCameraDeviceTest:	Starting camera testing (for desktop OS)
stopCameraDeviceTest	Ending camera testing (for desktop OS)
startMicDeviceTest:	Starting mic testing (for desktop OS)
startMicDeviceTest:playback:	Starting mic testing (for desktop OS)
stopMicDeviceTest	Ending mic testing (for desktop OS)
startSpeakerDeviceTest:	Starting speaker testing (for desktop OS)
stopSpeakerDeviceTest	Ending speaker testing (for desktop OS)
setObserver:	set onDeviceChanged callback (for Mac)
setCameraCapturerParam:	Set camera acquisition preferences

Disused APIs

FuncList	DESC
setSystemVolumeType:	Setting the system volume type (for mobile OS)

Disused APIs

FuncList	DESC
destroySharedIntance	Terminate TRTCCloud instance (singleton mode)
delegate	Set TRTC event callback
setBeautyStyle:beautyLevel:whitenessLevel:ruddinessLevel:	Set the strength of beauty, brightening, and rosy skin filters.
setEyeScaleLevel:	Set the strength of eye enlarging filter
setFaceScaleLevel:	Set the strength of face slimming filter

setFaceVLevel:	Set the strength of chin slimming filter
setChinLevel:	Set the strength of chin lengthening/shortening filter
setFaceShortLevel:	Set the strength of face shortening filter
setNoseSlimLevel:	Set the strength of nose slimming filter
selectMotionTmpl:	Set animated sticker
setMotionMute:	Mute animated sticker
setFilter:	Set color filter
setFilterConcentration:	Set the strength of color filter
setGreenScreenFile:	Set green screen video
setReverbType:	Set reverb effect
setVoiceChangerType:	Set voice changing type
enableAudioEarMonitoring:	Enable or disable in-ear monitoring
enableAudioVolumeEvaluation:	Enable volume reminder
enableAudioVolumeEvaluation:enable_vad:	Enable volume reminder
switchCamera	Switch camera
isCameraZoomSupported	Query whether the current camera supports zoom
setZoom:	Set camera zoom ratio (focal length)
isCameraTorchSupported	Query whether the device supports flash
enbaleTorch:	Enable/Disable flash
isCameraFocusPositionInPreviewSupported	Query whether the camera supports setting focus
setFocusPosition:	Set the focal position of camera
isCameraAutoFocusFaceModeSupported	Query whether the device supports the automatic recognition of face position
enableAutoFaceFoucs:	Enable/Disable face auto focus

setSystemVolumeType:	Setting the system volume type (for mobile OS)
snapshotVideo:type:	Screencapture video
startScreenCaptureByReplaykit:appGroup:	Start system-level screen sharing (for iOS 11.0 and above only)
startLocalAudio	Set sound quality
startRemoteView:view:	Start displaying remote video image
stopRemoteView:	Stop displaying remote video image and pulling the video data stream of remote user
setLocalViewFillMode:	Set the rendering mode of local image
setLocalViewRotation:	Set the clockwise rotation angle of local image
setLocalViewMirror:	Set the mirror mode of local camera's preview image
setRemoteViewFillMode:mode:	Set the fill mode of substream image
setRemoteViewRotation:rotation:	Set the clockwise rotation angle of remote image
startRemoteSubStreamView:view:	Start displaying the substream image of remote user
stopRemoteSubStreamView:	Stop displaying the substream image of remote user
setRemoteSubStreamViewFillMode:mode:	Set the fill mode of substream image
setRemoteSubStreamViewRotation:rotation:	Set the clockwise rotation angle of substream image
setAudioQuality:	Set sound quality
setPriorRemoteVideoStreamType:	Specify whether to view the big or small image
setMicVolumeOnMixing:	Set mic volume
playBGM:	Start background music
stopBGM	Stop background music

pauseBGM	Stop background music
resumeBGM	Stop background music
getBGMDuration:	Get the total length of background music in ms
setBGMPosition:	Set background music playback progress
setBGMVolume:	Set background music volume
setBGMPlayoutVolume:	Set the local playback volume of background music
setBGMPublishVolume:	Set the remote playback volume of background music
playAudioEffect:	Play sound effect
setAudioEffectVolume:volume:	Set sound effect volume
stopAudioEffect:	Stop sound effect
stopAllAudioEffects	Stop all sound effects
setAllAudioEffectsVolume:	Set the volume of all sound effects
pauseAudioEffect:	Pause sound effect
resumeAudioEffect:	Pause sound effect
enableCustomVideoCapture:	Enable custom video capturing mode
sendCustomVideoData:	Deliver captured video data to SDK
muteLocalVideo:	Pause/Resume publishing local video stream
muteRemoteVideoStream:mute:	Pause/Resume subscribing to remote user's video stream
startSpeedTest:userId:userSig:	Start network speed test (used before room entry)
startScreenCapture:	Start screen sharing
getCameraDevicesList	Get the list of cameras
setCurrentCameraDevice:	Set the camera to be used currently

getCurrentCameraDevice	Get the currently used camera
getMicDevicesList	Get the list of mics
getCurrentMicDevice	Get the current mic device
setCurrentMicDevice:	Select the currently used mic
getCurrentMicDeviceVolume	Get the current mic volume
setCurrentMicDeviceVolume:	Set the current mic volume
setCurrentMicDeviceMute:	Set the mute status of the current system mic
getCurrentMicDeviceMute	Get the mute status of the current system mic
getSpeakerDevicesList	Get the list of speakers
getCurrentSpeakerDevice	Get the currently used speaker
setCurrentSpeakerDevice:	Set the speaker to use
getCurrentSpeakerDeviceVolume	Get the current speaker volume
setCurrentSpeakerDeviceVolume:	Set the current speaker volume
getCurrentSpeakerDeviceMute	Get the mute status of the current system speaker
setCurrentSpeakerDeviceMute:	Set whether to mute the current system speaker
startCameraDeviceTestInView:	Start camera test
stopCameraDeviceTest	Start camera test
startMicDeviceTest:	Start mic test
stopMicDeviceTest	Start mic test
startSpeakerDeviceTest:	Start speaker test
stopSpeakerDeviceTest	Stop speaker test
startScreenCaptureInApp:	start in-app screen sharing (for iOS 13.0 and above only)
setVideoEncoderRotation:	Set the direction of image output by video encoder

setVideoEncoderMirror:	Set the mirror mode of image output by encoder
setGSensorMode:	Set the adaptation mode of G-sensor

TRTCCloud

Last updated：2024-06-06 15:26:14

Module: TRTCCloud @ TXLiteAVSDK

Function: TRTC's main feature API

Version: 11.9

TRTCCloud

TRTCCloud

FuncList	DESC
sharedInstance	Create TRTCCloud instance (singleton mode)
destroySharedInstance	Terminate TRTCCloud instance (singleton mode)
addDelegate:	Add TRTC event callback
removeDelegate:	Remove TRTC event callback
delegateQueue	Set the queue that drives the TRTCCloudDelegate event callback
enterRoom:appScene:	Enter room
exitRoom	Exit room
switchRole:	Switch role
switchRole:privateMapKey:	Switch role(support permission credential)
switchRoom:	Switch room
connectOtherRoom:	Request cross-room call

disconnectOtherRoom	Exit cross-room call
setDefaultStreamRecvMode:video:	Set subscription mode (which must be set before room entry for it to take effect)
createSubCloud	Create room subinstance (for concurrent multi-room listen/watch)
destroySubCloud:	Terminate room subinstance
updateOtherRoomForwardMode:	
startPublishing:type:	Start publishing audio/video streams to Tencent Cloud CSS CDN
stopPublishing	Stop publishing audio/video streams to Tencent Cloud CSS CDN
startPublishCDNStream:	Start publishing audio/video streams to non-Tencent Cloud CDN
stopPublishCDNStream	Stop publishing audio/video streams to non-Tencent Cloud CDN
setMixTranscodingConfig:	Set the layout and transcoding parameters of On-Cloud MixTranscoding
startPublishMediaStream:encoderParam:mixingConfig:	Publish a stream
updatePublishMediaStream:publishTarget:encoderParam:mixingConfig:	Modify publishing parameters
stopPublishMediaStream:	Stop publishing
startLocalPreview:view:	Enable the preview image of local camera (mobile)
startLocalPreview:	Enable the preview image of local camera (desktop)
updateLocalView:	Update the preview image of local camera
stopLocalPreview	Stop camera preview
muteLocalVideo:mute:	Pause/Resume publishing local video stream

setVideoMuteImage:fps:	Set placeholder image during local video pause
startRemoteView:streamType:view:	Subscribe to remote user's video stream and bind video rendering control
updateRemoteView:streamType:forUser:	Update remote user's video rendering control
stopRemoteView:streamType:	Stop subscribing to remote user's video stream and release rendering control
stopAllRemoteView	Stop subscribing to all remote users' video streams and release all rendering resources
muteRemoteVideoStream:streamType:mute:	Pause/Resume subscribing to remote user's video stream
muteAllRemoteVideoStreams:	Pause/Resume subscribing to all remote users' video streams
setVideoEncoderParam:	Set the encoding parameters of video encoder
setNetworkQosParam:	Set network quality control parameters
setLocalRenderParams:	Set the rendering parameters of local video image
setRemoteRenderParams:streamType:params:	Set the rendering mode of remote video image
enableEncSmallVideoStream:withQuality:	Enable dual-channel encoding mode with big and small images
setRemoteVideoStreamType:type:	Switch the big/small image of specified remote user
snapshotVideo:type:sourceType:	Screencapture video
setPerspectiveCorrectionWithUser:srcPoints:dstPoints:	Sets perspective correction coordinate points.
setGravitySensorAdaptiveMode:	Set the adaptation mode of gravity

	sensing (version 11.7 and above)
startLocalAudio:	Enable local audio capturing and publishing
stopLocalAudio	Stop local audio capturing and publishing
muteLocalAudio:	Pause/Resume publishing local audio stream
muteRemoteAudio:mute:	Pause/Resume playing back remote audio stream
muteAllRemoteAudio:	Pause/Resume playing back all remote users' audio streams
setAudioRoute:	Set audio route
setRemoteAudioVolume:volume:	Set the audio playback volume of remote user
setAudioCaptureVolume:	Set the capturing volume of local audio
getAudioCaptureVolume	Get the capturing volume of local audio
setAudioPlayoutVolume:	Set the playback volume of remote audio
getAudioPlayoutVolume	Get the playback volume of remote audio
enableAudioVolumeEvaluation:withParams:	Enable volume reminder
startAudioRecording:	Start audio recording
stopAudioRecording	Stop audio recording
startLocalRecording:	Start local media recording
stopLocalRecording	Stop local media recording
setRemoteAudioParallelParams:	Set the parallel strategy of remote audio streams
enable3DSpatialAudioEffect:	Enable 3D spatial effect

updateSelf3DSpatialPosition	Update self position and orientation for 3D spatial effect
updateRemote3DSpatialPosition:	Update the specified remote user's position for 3D spatial effect
set3DSpatialReceivingRange:range:	Set the maximum 3D spatial attenuation range for userId's audio stream
getDeviceManager	Get device management class (TXDeviceManager)
getBeautyManager	Get beauty filter management class (TXBeautyManager)
setWatermark:streamType:rect:	Add watermark
getAudioEffectManager	Get sound effect management class (TXAudioEffectManager)
startSystemAudioLoopback	Enable system audio capturing(iOS not supported)
stopSystemAudioLoopback	Stop system audio capturing(iOS not supported)
setSystemAudioLoopbackVolume:	Set the volume of system audio capturing
startScreenCaptureInApp:encParam:	Start in-app screen sharing (for iOS 13.0 and above only)
startScreenCaptureByReplaykit:encParam:appGroup:	Start system-level screen sharing (for iOS 11.0 and above only)
startScreenCapture:streamType:encParam:	Start screen sharing
stopScreenCapture	Stop screen sharing
pauseScreenCapture	Pause screen sharing
resumeScreenCapture	Resume screen sharing
getScreenCaptureSourcesWithThumbnailSize:iconSize:	Enumerate shareable screens and windows (for macOS only)
selectScreenCaptureTarget:rect:capturesCursor:highlight:	Select the screen or window to share (for macOS only)

setSubStreamEncoderParam:	Set the video encoding parameters of screen sharing (i.e., substream) (for desktop and mobile systems)
setSubStreamMixVolume:	Set the audio mixing volume of screen sharing (for desktop systems only)
addExcludedShareWindow:	Add specified windows to the exclusion list of screen sharing (for desktop systems only)
removeExcludedShareWindow:	Remove specified windows from the exclusion list of screen sharing (for desktop systems only)
removeAllExcludedShareWindows	Remove all windows from the exclusion list of screen sharing (for desktop systems only)
addIncludedShareWindow:	Add specified windows to the inclusion list of screen sharing (for desktop systems only)
removeIncludedShareWindow:	Remove specified windows from the inclusion list of screen sharing (for desktop systems only)
removeAllIncludedShareWindows	Remove all windows from the inclusion list of screen sharing (for desktop systems only)
enableCustomVideoCapture:enable:	Enable/Disable custom video capturing mode
sendCustomVideoData:frame:	Deliver captured video frames to SDK
enableCustomAudioCapture:	Enable custom audio capturing mode
sendCustomAudioData:	Deliver captured audio data to SDK
enableMixExternalAudioFrame:playout:	Enable/Disable custom audio track
mixExternalAudioFrame:	Mix custom audio track into SDK

setMixExternalAudioVolume:playoutVolume:	Set the publish volume and playback volume of mixed custom audio track
generateCustomPTS	Generate custom capturing timestamp
setLocalVideoProcessDelegete:pixelFormat:bufferType:	Set video data callback for third-party beauty filters
setLocalVideoRenderDelegate:pixelFormat:bufferType:	Set the callback of custom rendering for local video
setRemoteVideoRenderDelegate:delegate:pixelFormat:bufferType:	Set the callback of custom rendering for remote video
setAudioFrameDelegate:	Set custom audio data callback
setCapturedAudioFrameDelegateFormat:	Set the callback format of audio frames captured by local mic
setLocalProcessedAudioFrameDelegateFormat:	Set the callback format of preprocessed local audio frames
setMixedPlayAudioFrameDelegateFormat:	Set the callback format of audio frames to be played back by system
enableCustomAudioRendering:	Enabling custom audio playback
getCustomAudioRenderingFrame:	Getting playable audio data
sendCustomCmdMsg:data:reliable:ordered:	Use UDP channel to send custom message to all users in room
sendSEIMsg:repeatCount:	Use SEI channel to send custom message to all users in room
startSpeedTest:	Start network speed test (used before room entry)
stopSpeedTest	Stop network speed test
getSDKVersion	Get SDK version information
setLogLevel:	Set log output level
setConsoleEnabled:	Enable/Disable console log printing
setLogCompressEnabled:	Enable/Disable local log

	compression
setLogDirPath:	Set local log storage path
setLogDelegate:	Set log callback
showDebugView:	Display dashboard
setDebugViewMargin:margin:	Set dashboard margin
callExperimentalAPI:	Call experimental APIs
enablePayloadPrivateEncryption:params:	Enable or disable private encryption of media streams

sharedInstance

sharedInstance

Create TRTCCloud instance (singleton mode)

Param	DESC
context	It is only applicable to the Android platform. The SDK internally converts it into the `ApplicationContext` of Android to call the Android system API.

Note

1. If you use `delete ITRTCCloud*` , a compilation error will occur. Please use `destroyTRTCCloud` to release the object pointer.
2. On Windows, macOS, or iOS, please call the `getTRTCShareInstance()` API.
3. On Android, please call the `getTRTCShareInstance(void *context)` API.

destroySharedInstance

destroySharedInstance

Terminate TRTCCloud instance (singleton mode)

addDelegate:

addDelegate:

- (void)addDelegate:	(id<TRTCCloudDelegate>)delegate

Add TRTC event callback

You can use TRTCCloudDelegate to get various event notifications from the SDK, such as error codes, warning codes, and audio/video status parameters.

removeDelegate:

removeDelegate:

- (void)removeDelegate:	(id<TRTCCloudDelegate>)delegate

Remove TRTC event callback

delegateQueue

delegateQueue

Set the queue that drives the TRTCCloudDelegate event callback

If you do not specify a `delegateQueue` , the SDK will use `MainQueue` as the queue for driving TRTCCloudDelegate event callbacks by default.
In other words, if you do not set the `delegateQueue` attribute, all callback functions in TRTCCloudDelegate will be driven by `MainQueue` .

Note

If you specify a `delegateQueue` , please do not manipulate the UI in the TRTCCloudDelegate callback function; otherwise, thread safety issues will occur.

enterRoom:appScene:

enterRoom:appScene:

- (void)enterRoom:	(TRTCParams *)param
appScene:	(TRTCAppScene)scene

Enter room

All TRTC users need to enter a room before they can "publish" or "subscribe to" audio/video streams. "Publishing" refers to pushing their own streams to the cloud, and "subscribing to" refers to pulling the streams of other users in the room from the cloud.

When calling this API, you need to specify your application scenario (TRTCAppScene) to get the best audio/video transfer experience. We provide the following four scenarios for your choice:
TRTCAppSceneVideoCall:
Video call scenario. Use cases: [one-to-one video call], [video conferencing with up to 300 participants], [online medical diagnosis], [small class], [video interview], etc.
In this scenario, each room supports up to 300 concurrent online users, and up to 50 of them can speak simultaneously.
TRTCAppSceneAudioCall:
Audio call scenario. Use cases: [one-to-one audio call], [audio conferencing with up to 300 participants], [audio chat], [online Werewolf], etc.
In this scenario, each room supports up to 300 concurrent online users, and up to 50 of them can speak simultaneously.
TRTCAppSceneLIVE:
Live streaming scenario. Use cases: [low-latency video live streaming], [interactive classroom for up to 100,000 participants], [live video competition], [video dating room], [remote training], [large-scale conferencing], etc.
In this scenario, each room supports up to 100,000 concurrent online users, but you should specify the user roles: anchor (TRTCRoleAnchor) or audience (TRTCRoleAudience).
TRTCAppSceneVoiceChatRoom:
Audio chat room scenario. Use cases: [Clubhouse], [online karaoke room], [music live room], [FM radio], etc.
In this scenario, each room supports up to 100,000 concurrent online users, but you should specify the user roles: anchor (TRTCRoleAnchor) or audience (TRTCRoleAudience).

After calling this API, you will receive the `onEnterRoom(result)` callback from TRTCCloudDelegate:
If room entry succeeded, the `result` parameter will be a positive number (`result` > 0), indicating the time in milliseconds (ms) between function call and room entry.
If room entry failed, the `result` parameter will be a negative number (`result` < 0), indicating the TXLiteAVError for room entry failure.

Param	DESC
param	Room entry parameter, which is used to specify the user's identity, role, authentication credentials, and other information. For more information, please see TRTCParams.
scene	Application scenario, which is used to specify the use case. The same TRTCAppScene should be configured for all users in the same room.

Note

1. If `scene` is specified as TRTCAppSceneLIVE or TRTCAppSceneVoiceChatRoom, you must use the `role` field in TRTCParams to specify the role of the current user in the room.
2. The same `scene` should be configured for all users in the same room.
3. Please try to ensure that enterRoom and exitRoom are used in pair; that is, please make sure that "the previous room is exited before the next room is entered"; otherwise, many issues may occur.

exitRoom

exitRoom

Exit room

Calling this API will allow the user to leave the current audio or video room and release the camera, mic, speaker, and other device resources.
After resources are released, the SDK will use the `onExitRoom()` callback in TRTCCloudDelegate to notify you.

If you need to call enterRoom again or switch to the SDK of another provider, we recommend you wait until you receive the `onExitRoom()` callback, so as to avoid the problem of the camera or mic being occupied.

switchRole:

switchRole:

-(void)switchRole:	(TRTCRoleType)role

Switch role

This API is used to switch the user role between `anchor` and `audience` .

As video live rooms and audio chat rooms need to support an audience of up to 100,000 concurrent online users, the rule "only anchors can publish their audio/video streams" has been set. Therefore, when some users want to publish their streams (so that they can interact with anchors), they need to switch their role to "anchor" first.

You can use the `role` field in TRTCParams during room entry to specify the user role in advance or use the `switchRole` API to switch roles after room entry.

Param	DESC

role	Role, which is `anchor` by default: TRTCRoleAnchor: anchor, who can publish their audio/video streams. Up to 50 anchors are allowed to publish streams at the same time in one room. TRTCRoleAudience: audience, who cannot publish their audio/video streams, but can only watch streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room supports an audience of up to 100,000 concurrent online users.

Note

1. This API is only applicable to two scenarios: live streaming (TRTCAppSceneLIVE) and audio chat room (TRTCAppSceneVoiceChatRoom).
2. If the `scene` you specify in enterRoom is TRTCAppSceneVideoCall or TRTCAppSceneAudioCall, please do not call this API.

switchRole:privateMapKey:

switchRole:privateMapKey:

-(void)switchRole:	(TRTCRoleType)role
privateMapKey:	(NSString*)privateMapKey

Switch role(support permission credential)

This API is used to switch the user role between `anchor` and `audience`.

As video live rooms and audio chat rooms need to support an audience of up to 100,000 concurrent online users, the rule "only anchors can publish their audio/video streams" has been set. Therefore, when some users want to publish their streams (so that they can interact with anchors), they need to switch their role to "anchor" first.

You can use the `role` field in TRTCParams during room entry to specify the user role in advance or use the `switchRole` API to switch roles after room entry.

Param	DESC
privateMapKey	Permission credential used for permission control. If you want only users with the specified `userId` values to enter a room or push streams, you need to use `privateMapKey` to restrict the permission. We recommend you use this parameter only if you have high security requirements. For more information, please see Enabling Advanced Permission Control.
role	Role, which is `anchor` by default:

	TRTCRoleAnchor: anchor, who can publish their audio/video streams. Up to 50 anchors are allowed to publish streams at the same time in one room. TRTCRoleAudience: audience, who cannot publish their audio/video streams, but can only watch streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room supports an audience of up to 100,000 concurrent online users.

Note

1. This API is only applicable to two scenarios: live streaming (TRTCAppSceneLIVE) and audio chat room (TRTCAppSceneVoiceChatRoom).
2. If the `scene` you specify in enterRoom is TRTCAppSceneVideoCall or TRTCAppSceneAudioCall, please do not call this API.

switchRoom:

switchRoom:

- (void)switchRoom:	(TRTCSwitchRoomConfig *)config

Switch room

This API is used to quickly switch a user from one room to another.

If the user's role is `audience`, calling this API is equivalent to `exitRoom` (current room) + `enterRoom` (new room).

If the user's role is `anchor`, the API will retain the current audio/video publishing status while switching the room; therefore, during the room switch, camera preview and sound capturing will not be interrupted.

This API is suitable for the online education scenario where the supervising teacher can perform fast room switch across multiple rooms. In this scenario, using `switchRoom` can get better smoothness and use less code than `exitRoom + enterRoom`.

The API call result will be called back through `onSwitchRoom(errCode, errMsg)` in TRTCCloudDelegate.

Param	DESC
config	Room parameter. For more information, please see TRTCSwitchRoomConfig.

Note

Due to the requirement for compatibility with legacy versions of the SDK, the `config` parameter contains both `roomId` and `strRoomId` parameters. You should pay special attention as detailed below when specifying these two parameters:

1. If you decide to use `strRoomId`, then set `roomId` to 0. If both are specified, `roomId` will be used.
2. All rooms need to use either `strRoomId` or `roomId` at the same time. They cannot be mixed; otherwise, there will be many unexpected bugs.

connectOtherRoom:

connectOtherRoom:

- (void)connectOtherRoom:	(NSString *)param

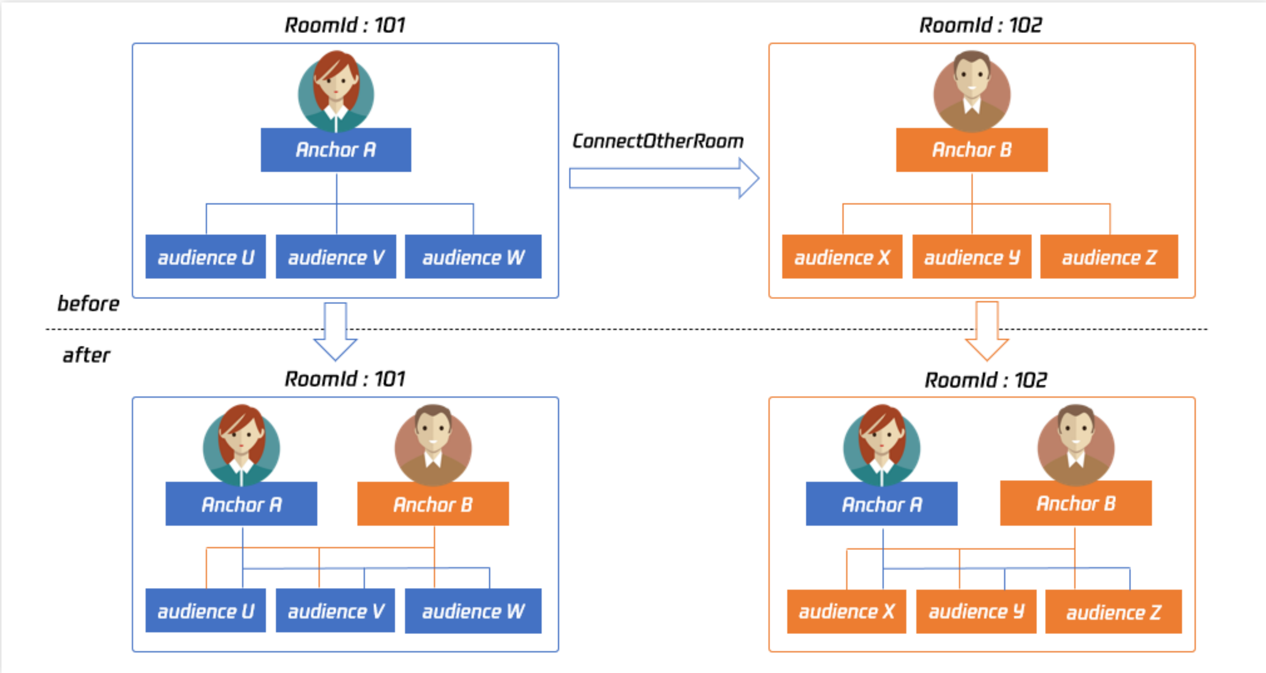
Request cross-room call

By default, only users in the same room can make audio/video calls with each other, and the audio/video streams in different rooms are isolated from each other.
However, you can publish the audio/video streams of an anchor in another room to the current room by calling this API. At the same time, this API will also publish the local audio/video streams to the target anchor's room.

In other words, you can use this API to share the audio/video streams of two anchors in two different rooms, so that the audience in each room can watch the streams of these two anchors. This feature can be used to implement anchor competition.

The result of requesting cross-room call will be returned through the onConnectOtherRoom callback in TRTCCloudDelegate.

For example, after anchor A in room "101" uses `connectOtherRoom()` to successfully call anchor B in room "102":
All users in room "101" will receive the `onRemoteUserEnterRoom(B)` and `onUserVideoAvailable(B,YES)` event callbacks of anchor B; that is, all users in room "101" can subscribe to the audio/video streams of anchor B.
All users in room "102" will receive the `onRemoteUserEnterRoom(A)` and `onUserVideoAvailable(A,YES)` event callbacks of anchor A; that is, all users in room "102" can subscribe to the audio/video streams of anchor A.



For compatibility with subsequent extended fields for cross-room call, parameters in JSON format are used currently.

Case 1: numeric room ID

If anchor A in room "101" wants to co-anchor with anchor B in room "102", then anchor A needs to pass in {"roomId": 102, "userId": "userB"} when calling this API.

Below is the sample code:

```
NSMutableDictionaryjsonDict = [[NSMutableDictionary alloc] init];
[jsonDict setObject:@(102) forKey:@"roomId"];
[jsonDict setObject:@"userB" forKey:@"userId"];
NSData* jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict options:NSJSO
NSString* jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Str
[trtc connectOtherRoom:jsonString];
```

Case 2: string room ID

If you use a string room ID, please be sure to replace the `roomId` in JSON with `strRoomId` , such as {"strRoomId": "102", "userId": "userB"}

Below is the sample code:

```
NSMutableDictionaryjsonDict = [[NSMutableDictionary alloc] init];
[jsonDict setObject:@"102" forKey:@"strRoomId"];
[jsonDict setObject:@"userB" forKey:@"userId"];
NSData* jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict options:NSJSO
NSString* jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Str
[trtc connectOtherRoom:jsonString];
```

Param	DESC
param	You need to pass in a string parameter in JSON format: `roomId` represents the room ID in numeric format, `strRoomId` represents the room ID in string format, and `userId` represents the user ID of the target anchor.

disconnectOtherRoom

disconnectOtherRoom

Exit cross-room call

The result will be returned through the `onDisconnectOtherRoom()` callback in TRTCCloudDelegate.

setDefaultStreamRecvMode:video:

setDefaultStreamRecvMode:video:

- (void)setDefaultStreamRecvMode:	(BOOL)autoRecvAudio
video:	(BOOL)autoRecvVideo

Set subscription mode (which must be set before room entry for it to take effect)

You can switch between the "automatic subscription" and "manual subscription" modes through this API:
Automatic subscription: this is the default mode, where the user will immediately receive the audio/video streams in the room after room entry, so that the audio will be automatically played back, and the video will be automatically decoded (you still need to bind the rendering control through the `startRemoteView` API).
Manual subscription: after room entry, the user needs to manually call the startRemoteView API to start subscribing to and decoding the video stream and call the muteRemoteAudio (NO) API to start playing back the audio stream.

In most scenarios, users will subscribe to the audio/video streams of all anchors in the room after room entry. Therefore, TRTC adopts the automatic subscription mode by default in order to achieve the best "instant streaming experience".
In your application scenario, if there are many audio/video streams being published at the same time in each room, and each user only wants to subscribe to 1–2 streams of them, we recommend you use the "manual subscription" mode to reduce the traffic costs.

Param	DESC
autoRecvAudio	YES: automatic subscription to audio; NO: manual subscription to audio by calling

	`muteRemoteAudio(NO)` . Default value: YES
autoRecvVideo	YES: automatic subscription to video; NO: manual subscription to video by calling `startRemoteView` . Default value: YES

Note

1. The configuration takes effect only if this API is called before room entry (enterRoom).
2. In the automatic subscription mode, if the user does not call startRemoteView to subscribe to the video stream after room entry, the SDK will automatically stop subscribing to the video stream in order to reduce the traffic consumption.

createSubCloud

createSubCloud

Create room subinstance (for concurrent multi-room listen/watch)

`TRTCCloud` was originally designed to work in the singleton mode, which limited the ability to watch concurrently in multiple rooms.
By calling this API, you can create multiple `TRTCCloud` instances, so that you can enter multiple different rooms at the same time to listen/watch audio/video streams.

However, it should be noted that your ability to publish audio and video streams in multiple `TRTCCloud` instances will be limited.

This feature is mainly used in the "super small class" use case in the online education scenario to break the limit that "only up to 50 users can publish their audio/video streams simultaneously in one TRTC room".

Below is the sample code:

```
//In the small room that needs interaction, enter the room as an anchor and pus
TRTCCloud *mainCloud = [TRTCCloud sharedInstance];
TRTCParams *mainParams = [[TRTCParams alloc] init];
//Fill your params
mainParams.role = TRTCRoleAnchor;
[mainCloud enterRoom:mainParams appScene:TRTCAppSceneLIVE)];
//...
[mainCloud startLocalPreview:YES view:videoView];
[mainCloud startLocalAudio:TRTCAudioQualityDefault];

//In the large room that only needs to watch, enter the room as an audience and
```



```
    TRTCCloud *subCloud = [mainCloud createSubCloud];
    TRTCParams *subParams = [[TRTCParams alloc] init];
    //Fill your params
    subParams.role = TRTCRoleAudience;
    [subCloud enterRoom:subParams appScene:TRTCAppSceneLIVE)];
    //...
    [subCloud startRemoteView:userId streamType:TRTCVideoStreamTypeBig view:videoVi
    //...
    //Exit from new room and release it.
    [subCloud exitRoom];
    [mainCloud destroySubCloud:subCloud];
```

Note

The same user can enter multiple rooms with different `roomId` values by using the same `userId`.

Two devices cannot use the same `userId` to enter the same room with a specified `roomId`.

You can set TRTCCloudDelegate separately for different instances to get their own event notifications.

The same user can push streams in multiple `TRTCCloud` instances at the same time, and can also call APIs related to local audio/video in the sub instance. But need to pay attention to:

Audio needs to be collected by the microphone or custom data at the same time in all instances, and the result of API calls related to the audio device will be based on the last time;

The result of camera-related API call will be based on the last time: startLocalPreview.

Return Desc:

`TRTCCloud` subinstance

destroySubCloud:

destroySubCloud:

- (void)destroySubCloud:	(TRTCCloud *)subCloud

Terminate room subinstance

Param	DESC
subCloud	

startPublishing:type:

startPublishing:type:

- (void)startPublishing:	(NSString *)streamId
type:	(TRTCVideoStreamType)streamType

Start publishing audio/video streams to Tencent Cloud CSS CDN

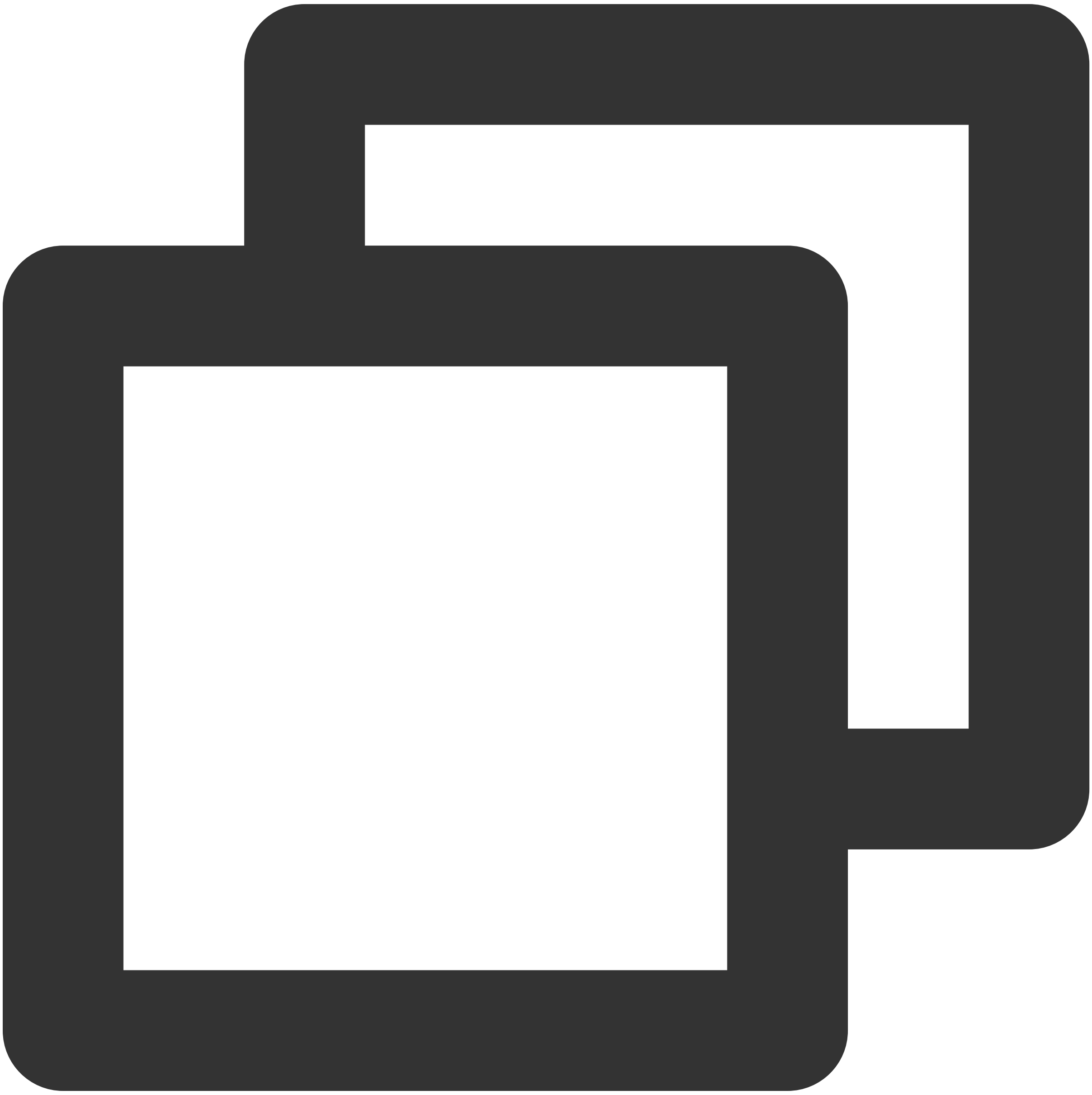
This API sends a command to the TRTC server, requesting it to relay the current user's audio/video streams to CSS CDN.

You can set the `StreamId` of the live stream through the `streamId` parameter, so as to specify the playback address of the user's audio/video streams on CSS CDN.

For example, if you specify the current user's live stream ID as `user_stream_001` through this API, then the corresponding CDN playback address is:

"http://yourdomain/live/user_stream_001.flv", where `yourdomain` is your playback domain name with an ICP filing.

You can configure your playback domain name in the CSS console. Tencent Cloud does not provide a default playback domain name.



```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
[trtcCloud startLocalPreview:frontCamera view:localView];
[trtcCloud startLocalAudio];
[trtcCloud startPublishing: @"user_stream_001" type:TRTCVideoStreamTypeBig];
```

You can also specify the `streamId` when setting the `TRTCParams` parameter of `enterRoom` , which is the recommended approach.

Param	DESC

streamId	Custom stream ID.
streamType	Only `TRTCVideoStreamTypeBig` and `TRTCVideoStreamTypeSub` are supported.

Note

You need to enable the "Enable Relayed Push" option on the "Function Configuration" page in the TRTC console in advance.

If you select "Specified stream for relayed push", you can use this API to push the corresponding audio/video stream to Tencent Cloud CDN and specify the entered stream ID.

If you select "Global auto-relayed push", you can use this API to adjust the default stream ID.

stopPublishing

stopPublishing

Stop publishing audio/video streams to Tencent Cloud CSS CDN

startPublishCDNStream:

startPublishCDNStream:

- (void)startPublishCDNStream:	(TRTCPublishCDNParam*)param

Start publishing audio/video streams to non-Tencent Cloud CDN

This API is similar to the `startPublishing` API. The difference is that `startPublishing` can only publish audio/video streams to Tencent Cloud CDN, while this API can relay streams to live streaming CDN services of other cloud providers.

Param	DESC
param	CDN relaying parameter. For more information, please see TRTCPublishCDNParam

Note

Using the `startPublishing` API to publish audio/video streams to Tencent Cloud CSS CDN does not incur additional fees.

Using the `startPublishCDNStream` API to publish audio/video streams to non-Tencent Cloud CDN incurs additional relaying bandwidth fees.

stopPublishCDNStream

stopPublishCDNStream

Stop publishing audio/video streams to non-Tencent Cloud CDN

setMixTranscodingConfig:

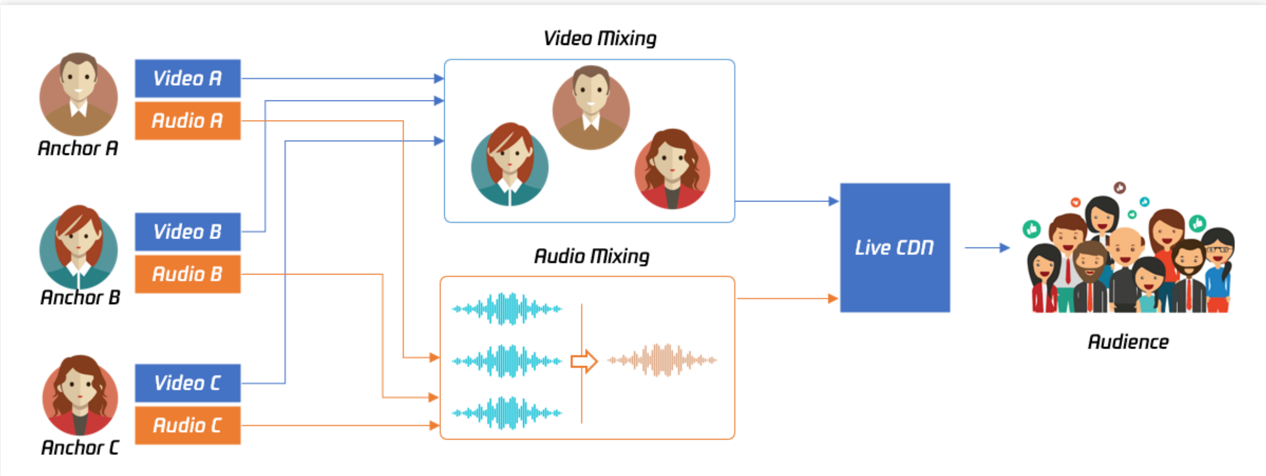
setMixTranscodingConfig:

- (void)setMixTranscodingConfig:	(nullable TRTCTranscodingConfig*)config

Set the layout and transcoding parameters of On-Cloud MixTranscoding

In a live room, there may be multiple anchors publishing their audio/video streams at the same time, but for audience on CSS CDN, they only need to watch one video stream in HTTP-FLV or HLS format.
When you call this API, the SDK will send a command to the TRTC mixtranscoding server to combine multiple audio/video streams in the room into one stream.
You can use the TRTCTranscodingConfig parameter to set the layout of each channel of image. You can also set the encoding parameters of the mixed audio/video streams.

For more information, please see On-Cloud MixTranscoding.



Param	DESC
config	If `config` is not empty, On-Cloud MixTranscoding will be started; otherwise, it will be stopped. For more information, please see TRTCTranscodingConfig.

Note

Notes on On-Cloud MixTranscoding:

Mixed-stream transcoding is a chargeable function, calling the interface will incur cloud-based mixed-stream transcoding fees, see Billing of On-Cloud MixTranscoding.

If the user calling this API does not set `streamId` in the `config` parameter, TRTC will mix the multiple channels of images in the room into the audio/video streams corresponding to the current user, i.e., A + B => A.

If the user calling this API sets `streamId` in the `config` parameter, TRTC will mix the multiple channels of images in the room into the specified `streamId`, i.e., A + B => streamId.

Please note that if you are still in the room but do not need mixtranscoding anymore, be sure to call this API again and leave `config` empty to cancel it; otherwise, additional fees may be incurred.

Please rest assured that TRTC will automatically cancel the mixtranscoding status upon room exit.

startPublishMediaStream:encoderParam:mixingConfig:

startPublishMediaStream:encoderParam:mixingConfig:

- (void)startPublishMediaStream:	(TRTCPublishTarget*)target
encoderParam:	(nullable TRTCStreamEncoderParam*)param
mixingConfig:	(nullable TRTCStreamMixingConfig*)config

Publish a stream

After this API is called, the TRTC server will relay the stream of the local user to a CDN (after transcoding or without transcoding), or transcode and publish the stream to a TRTC room.

You can use the TRTCPublishMode parameter in TRTCPublishTarget to specify the publishing mode.

Param	DESC
config	The On-Cloud MixTranscoding settings. This parameter is invalid in the relay-to-CDN mode. It is required if you transcode and publish the stream to a CDN or to a TRTC room. For details, see TRTCStreamMixingConfig.
params	The encoding settings. This parameter is required if you transcode and publish the stream to a CDN or to a TRTC room. If you relay to a CDN without transcoding, to improve the relaying stability and playback compatibility, we also recommend you set this parameter. For details, see TRTCStreamEncoderParam.
target	The publishing destination. You can relay the stream to a CDN (after transcoding or without transcoding) or transcode and publish the stream to a TRTC room. For details, see

	TRTCPublishTarget.

Note

1. The SDK will send a task ID to you via the onStartPublishMediaStream callback.
2. You can start a publishing task only once and cannot initiate two tasks that use the same publishing mode and publishing cdn url. Note the task ID returned, which you need to pass to updatePublishMediaStream to modify the publishing parameters or stopPublishMediaStream to stop the task.
3. You can specify up to 10 CDN URLs in `target` . You will be charged only once for transcoding even if you relay to multiple CDNs.
4. To avoid causing errors, do not specify the same URLs for different publishing tasks executed at the same time. We recommend you add "sdkappid_roomid_userid_main" to URLs to distinguish them from one another and avoid application conflicts.

updatePublishMediaStream:publishTarget:encoderParam:mixingConfig:

updatePublishMediaStream:publishTarget:encoderParam:mixingConfig:

- (void)updatePublishMediaStream:	(NSString *)taskId
publishTarget:	(TRTCPublishTarget*)target
encoderParam:	(nullable TRTCStreamEncoderParam*)param
mixingConfig:	(nullable TRTCStreamMixingConfig*)config

Modify publishing parameters

You can use this API to change the parameters of a publishing task initiated by startPublishMediaStream.

Param	DESC
config	The On-Cloud MixTranscoding settings. This parameter is invalid in the relay-to-CDN mode. It is required if you transcode and publish the stream to a CDN or to a TRTC room. For details, see TRTCStreamMixingConfig.
params	The encoding settings. This parameter is required if you transcode and publish the stream to a CDN or to a TRTC room. If you relay to a CDN without transcoding, to improve the relaying stability and playback compatibility, we recommend you set this parameter. For details, see TRTCStreamEncoderParam.
target	The publishing destination. You can relay the stream to a CDN (after transcoding or without

	transcoding) or transcode and publish the stream to a TRTC room. For details, see TRTCPublishTarget.
taskId	The task ID returned to you via the onStartPublishMediaStream callback.

Note

1. You can use this API to add or remove CDN URLs to publish to (you can publish to up to 10 CDNs at a time). To avoid causing errors, do not specify the same URLs for different tasks executed at the same time.
2. You can use this API to switch a relaying task to transcoding or vice versa. For example, in cross-room communication, you can first call startPublishMediaStream to relay to a CDN. When the anchor requests cross-room communication, call this API, passing in the task ID to switch the relaying task to a transcoding task. This can ensure that the live stream and CDN playback are not interrupted (you need to keep the encoding parameters consistent).
3. You can not switch output between "only audio" 、 "only video" and "audio and video" for the same task.

stopPublishMediaStream:

stopPublishMediaStream:

- (void)stopPublishMediaStream:	(NSString *)taskId

Stop publishing

You can use this API to stop a task initiated by startPublishMediaStream.

Param	DESC
taskId	The task ID returned to you via the onStartPublishMediaStream callback.

Note

1. If the task ID is not saved to your backend, you can call startPublishMediaStream again when an anchor re-enters the room after abnormal exit. The publishing will fail, but the TRTC backend will return the task ID to you.
2. If `taskId` is left empty, the TRTC backend will end all tasks you started through startPublishMediaStream. You can leave it empty if you have started only one task or want to stop all publishing tasks started by you.

startLocalPreview:view:

startLocalPreview:view:

- (void)startLocalPreview:	(BOOL)frontCamera

view:	(nullable TXView *)view

Enable the preview image of local camera (mobile)

If this API is called before `enterRoom` , the SDK will only enable the camera and wait until `enterRoom` is called before starting push.
If it is called after `enterRoom` , the SDK will enable the camera and automatically start pushing the video stream.
When the first camera video frame starts to be rendered, you will receive the `onCameraDidReady` callback in TRTCCloudDelegate.

Param	DESC
frontCamera	YES: front camera; NO: rear camera
view	Control that carries the video image

Note
If you want to preview the camera image and adjust the beauty filter parameters through `BeautyManager` before going live, you can:
Scheme 1. Call `startLocalPreview` before calling `enterRoom`
Scheme 2. Call `startLocalPreview` and `muteLocalVideo(YES)` after calling `enterRoom`

startLocalPreview:

startLocalPreview:

- (void)startLocalPreview:	(nullable TXView *)view

Enable the preview image of local camera (desktop)

Before this API is called, `setCurrentCameraDevice` can be called first to select whether to use the macOS device's built-in camera or an external camera.
If this API is called before `enterRoom` , the SDK will only enable the camera and wait until `enterRoom` is called before starting push.
If it is called after `enterRoom` , the SDK will enable the camera and automatically start pushing the video stream.
When the first camera video frame starts to be rendered, you will receive the `onCameraDidReady` callback in TRTCCloudDelegate.

Param	DESC
view	Control that carries the video image

Note

If you want to preview the camera image and adjust the beauty filter parameters through `BeautyManager` before going live, you can:

Scheme 1. Call `startLocalPreview` before calling `enterRoom`

Scheme 2. Call `startLocalPreview` and `muteLocalVideo(YES)` after calling `enterRoom`

updateLocalView:

updateLocalView:

- (void)updateLocalView:	(nullable TXView *)view

Update the preview image of local camera

stopLocalPreview

stopLocalPreview

Stop camera preview

muteLocalVideo:mute:

muteLocalVideo:mute:

- (void)muteLocalVideo:	(TRTCVideoStreamType)streamType
mute:	(BOOL)mute

Pause/Resume publishing local video stream

This API can pause (or resume) publishing the local video image. After the pause, other users in the same room will not be able to see the local image.

This API is equivalent to the two APIs of `startLocalPreview/stopLocalPreview` when TRTCVideoStreamTypeBig is specified, but has higher performance and response speed.

The `startLocalPreview/stopLocalPreview` APIs need to enable/disable the camera, which are hardware device-related operations, so they are very time-consuming.

In contrast, `muteLocalVideo` only needs to pause or allow the data stream at the software level, so it is more efficient and more suitable for scenarios where frequent enabling/disabling are needed.

After local video publishing is paused, other members in the same room will receive the `onUserVideoAvailable(userId, NO)` callback notification.
After local video publishing is resumed, other members in the same room will receive the `onUserVideoAvailable(userId, YES)` callback notification.

Param	DESC
mute	YES: pause; NO: resume
streamType	Specify for which video stream to pause (or resume). Only TRTCVideoStreamTypeBig and TRTCVideoStreamTypeSub are supported

setVideoMuteImage:fps:

setVideoMuteImage:fps:

- (void)setVideoMuteImage:	(nullable TXImage *)image
fps:	(NSInteger)fps

Set placeholder image during local video pause

When you call `muteLocalVideo(YES)` to pause the local video image, you can set a placeholder image by calling this API. Then, other users in the room will see this image instead of a black screen.

Param	DESC
fps	Frame rate of the placeholder image. Minimum value: 5. Maximum value: 10. Default value: 5
image	Placeholder image. A null value means that no more video stream data will be sent after `muteLocalVideo`. The default value is null.

startRemoteView:streamType:view:

startRemoteView:streamType:view:

- (void)startRemoteView:	(NSString *)userId
streamType:	(TRTCVideoStreamType)streamType
view:	(nullable TXView *)view

Subscribe to remote user's video stream and bind video rendering control

Calling this API allows the SDK to pull the video stream of the specified `userId` and render it to the rendering control specified by the `view` parameter. You can set the display mode of the video image through setRemoteRenderParams.

If you already know the `userId` of a user who has a video stream in the room, you can directly call `startRemoteView` to subscribe to the user's video image.

If you don't know which users in the room are publishing video streams, you can wait for the notification from onUserVideoAvailable after `enterRoom`.

Calling this API only starts pulling the video stream, and the image needs to be loaded and buffered at this time. After the buffering is completed, you will receive a notification from onFirstVideoFrame.

Param	DESC
streamType	Video stream type of the `userId` specified for watching: HD big image: TRTCVideoStreamTypeBig Smooth small image: TRTCVideoStreamTypeSmall (the remote user should enable dual-channel encoding through enableEncSmallVideoStream for this parameter to take effect) Substream image (usually used for screen sharing): TRTCVideoStreamTypeSub
userId	ID of the specified remote user
view	Rendering control that carries the video image

Note

The following requires your attention:

1. The SDK supports watching the big image and substream image or small image and substream image of a `userId` at the same time, but does not support watching the big image and small image at the same time.
2. Only when the specified `userId` enables dual-channel encoding through enableEncSmallVideoStream can the user's small image be viewed.
3. If the small image of the specified `userId` does not exist, the SDK will switch to the big image of the user by default.

updateRemoteView:streamType:forUser:

updateRemoteView:streamType:forUser:

- (void)updateRemoteView:	(nullable TXView *)view
streamType:	(TRTCVideoStreamType)streamType

forUser:	(NSString *)userId

Update remote user's video rendering control

This API can be used to update the rendering control of the remote video image. It is often used in interactive scenarios where the display area needs to be switched.

Param	DESC
streamType	Type of the stream for which to set the preview window (only TRTCVideoStreamTypeBig and TRTCVideoStreamTypeSub are supported)
userId	ID of the specified remote user
view	Control that carries the video image

stopRemoteView:streamType:

stopRemoteView:streamType:

- (void)stopRemoteView:	(NSString *)userId
streamType:	(TRTCVideoStreamType)streamType

Stop subscribing to remote user's video stream and release rendering control

Calling this API will cause the SDK to stop receiving the user's video stream and release the decoding and rendering resources for the stream.

Param	DESC
streamType	Video stream type of the `userId` specified for watching: HD big image: TRTCVideoStreamTypeBig Smooth small image: TRTCVideoStreamTypeSmall Substream image (usually used for screen sharing): TRTCVideoStreamTypeSub
userId	ID of the specified remote user

stopAllRemoteView

stopAllRemoteView

Stop subscribing to all remote users' video streams and release all rendering resources

Calling this API will cause the SDK to stop receiving all remote video streams and release all decoding and rendering resources.

Note

If a substream image (screen sharing) is being displayed, it will also be stopped.

muteRemoteVideoStream:streamType:mute:

muteRemoteVideoStream:streamType:mute:

- (void)muteRemoteVideoStream:	(NSString*)userId
streamType:	(TRTCVideoStreamType)streamType
mute:	(BOOL)mute

Pause/Resume subscribing to remote user's video stream

This API only pauses/resumes receiving the specified user's video stream but does not release displaying resources; therefore, the video image will freeze at the last frame before it is called.

Param	DESC
mute	Whether to pause receiving
streamType	Specify for which video stream to pause (or resume): HD big image: TRTCVideoStreamTypeBig Smooth small image: TRTCVideoStreamTypeSmall Substream image (usually used for screen sharing): TRTCVideoStreamTypeSub
userId	ID of the specified remote user

Note

This API can be called before room entry (enterRoom), and the pause status will be reset after room exit (exitRoom). After calling this API to pause receiving the video stream from a specific user, simply calling the startRemoteView API will not be able to play the video from that user. You need to call muteRemoteVideoStream(NO) or muteAllRemoteVideoStreams(NO) to resume it.

muteAllRemoteVideoStreams:

muteAllRemoteVideoStreams:

- (void)muteAllRemoteVideoStreams:	(BOOL)mute

Pause/Resume subscribing to all remote users' video streams

This API only pauses/resumes receiving all users' video streams but does not release displaying resources; therefore, the video image will freeze at the last frame before it is called.

Param	DESC
mute	Whether to pause receiving

Note

This API can be called before room entry (enterRoom), and the pause status will be reset after room exit (exitRoom).
After calling this interface to pause receiving video streams from all users, simply calling the startRemoteView interface will not be able to play the video from a specific user. You need to call muteRemoteVideoStream(NO) or muteAllRemoteVideoStreams(NO) to resume it.

setVideoEncoderParam:

setVideoEncoderParam:

- (void)setVideoEncoderParam:	(TRTCVideoEncParam*)param

Set the encoding parameters of video encoder

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.

Param	DESC
param	It is used to set relevant parameters for the video encoder. For more information, please see TRTCVideoEncParam.

Note

Begin from v11.5 version, the encoding output resolution will be aligned according to width 8 and height 2 bytes, and will be adjusted downward, eg: input resolution 540x960, actual encoding output resolution 536x960.

setNetworkQosParam:

setNetworkQosParam:

- (void)setNetworkQosParam:	(TRTCNetworkQosParam*)param

Set network quality control parameters

This setting determines the quality control policy in a poor network environment, such as "image quality preferred" or "smoothness preferred".

Param	DESC
param	It is used to set relevant parameters for network quality control. For details, please refer to TRTCNetworkQosParam.

setLocalRenderParams:

setLocalRenderParams:

- (void)setLocalRenderParams:	(TRTCRenderParams *)params

Set the rendering parameters of local video image

The parameters that can be set include video image rotation angle, fill mode, and mirror mode.

Param	DESC
params	Video image rendering parameters. For more information, please see TRTCRenderParams.

setRemoteRenderParams:streamType:params:

setRemoteRenderParams:streamType:params:

- (void)setRemoteRenderParams:	(NSString *)userId
streamType:	(TRTCVideoStreamType)streamType
params:	(TRTCRenderParams *)params

Set the rendering mode of remote video image

The parameters that can be set include video image rotation angle, fill mode, and mirror mode.

Param	DESC
params	Video image rendering parameters. For more information, please see TRTCRenderParams.
streamType	It can be set to the primary stream image (TRTCVideoStreamTypeBig) or substream image (TRTCVideoStreamTypeSub).

userId	ID of the specified remote user

enableEncSmallVideoStream:withQuality:

enableEncSmallVideoStream:withQuality:

- (int)enableEncSmallVideoStream:	(BOOL)enable
withQuality:	(TRTCVideoEncParam*)smallVideoEncParam

Enable dual-channel encoding mode with big and small images

In this mode, the current user's encoder will output two channels of video streams, i.e., **HD big image** and **Smooth small image**, at the same time (only one channel of audio stream will be output though).
In this way, other users in the room can choose to subscribe to the **HD big image** or **Smooth small image** according to their own network conditions or screen size.

Param	DESC
enable	Whether to enable small image encoding. Default value: NO
smallVideoEncParam	Video parameters of small image stream

Note

Dual-channel encoding will consume more CPU resources and network bandwidth; therefore, this feature can be enabled on macOS, Windows, or high-spec tablets, but is not recommended for phones.

Return Desc:

0: success; -1: the current big image has been set to a lower quality, and it is not necessary to enable dual-channel encoding

setRemoteVideoStreamType:type:

setRemoteVideoStreamType:type:

- (void)setRemoteVideoStreamType:	(NSString*)userId
type:	(TRTCVideoStreamType)streamType

Switch the big/small image of specified remote user

After an anchor in a room enables dual-channel encoding, the video image that other users in the room subscribe to through startRemoteView will be **HD big image** by default.
You can use this API to select whether the image subscribed to is the big image or small image. The API can take effect before or after startRemoteView is called.

Param	DESC
streamType	Video stream type, i.e., big image or small image. Default value: big image
userId	ID of the specified remote user

Note
To implement this feature, the target user must have enabled the dual-channel encoding mode through enableEncSmallVideoStream; otherwise, this API will not work.

snapshotVideo:type:sourceType:

snapshotVideo:type:sourceType:

- (void)snapshotVideo:	(nullable NSString *)userId
type:	(TRTCVideoStreamType)streamType
sourceType:	(TRTCSnapshotSourceType)sourceType

Screencapture video

You can use this API to screencapture the local video image or the primary stream image and substream (screen sharing) image of a remote user.

Param	DESC
sourceType	Video image source, which can be the video stream image (TRTCSnapshotSourceTypeStream, generally in higher definition)、 the video rendering image (TRTCSnapshotSourceTypeView) or the capture picture (TRTCSnapshotSourceTypeCapture).The captured picture screenshot will be clearer.
streamType	Video stream type, which can be the primary stream image (TRTCVideoStreamTypeBig, generally for camera) or substream image (TRTCVideoStreamTypeSub, generally for screen sharing)
userId	User ID. A null value indicates to screencapture the local video.

Note

On Windows, only video image from the TRTCSnapshotSourceTypeStream source can be screencaptured currently.

setPerspectiveCorrectionWithUser:srcPoints:dstPoints:

setPerspectiveCorrectionWithUser:srcPoints:dstPoints:

- (void)setPerspectiveCorrectionWithUser:	(nullable NSString *)userId
srcPoints:	(nullable NSArray *)srcPoints
dstPoints:	(nullable NSArray *)dstPoints

Sets perspective correction coordinate points.

This function allows you to specify coordinate areas for perspective correction.

Param	DESC
dstPoints	The coordinates of the four vertices of the target corrected area should be passed in the order of top-left, bottom-left, top-right, bottom-right. All coordinates need to be normalized to the [0,1] range based on the render view width and height, or null to stop perspective correction of the corresponding stream.
srcPoints	The coordinates of the four vertices of the original stream image area should be passed in the order of top-left, bottom-left, top-right, bottom-right. All coordinates need to be normalized to the [0,1] range based on the render view width and height, or null to stop perspective correction of the corresponding stream.
userId	userId which corresponding to the target stream. If null value is specified, it indicates that the function is applied to the local stream.

setGravitySensorAdaptiveMode:

setGravitySensorAdaptiveMode:

- (void)setGravitySensorAdaptiveMode:	(TRTCGravitySensorAdaptiveMode) mode

Set the adaptation mode of gravity sensing (version 11.7 and above)

After turning on gravity sensing, if the device on the collection end rotates, the images on the collection end and the audience will be rendered accordingly to ensure that the image in the field of view is always facing up.
It only takes effect in the camera capture scene inside the SDK, and only takes effect on the mobile terminal.

1. This interface only works for the collection end. If you only watch the picture in the room, opening this interface is invalid.
2. When the capture device is rotated 90 degrees or 270 degrees, the picture seen by the capture device or the audience may be cropped to maintain proportional coordination.

Param	DESC
mode	Gravity sensing mode, see TRTCGravitySensorAdaptiveMode_Disable、TRTCGravitySensorAdaptiveMode_FillByCenterCrop and TRTCGravitySensorAdaptiveMode_FitWithBlackBorder for details, default value: TRTCGravitySensorAdaptiveMode_Disable.

startLocalAudio:

startLocalAudio:

- (void)startLocalAudio:	(TRTCAudioQuality)quality

Enable local audio capturing and publishing

The SDK does not enable the mic by default. When a user wants to publish the local audio, the user needs to call this API to enable mic capturing and encode and publish the audio to the current room.
After local audio capturing and publishing is enabled, other users in the room will receive the onUserAudioAvailable(userId, YES) notification.

Param	DESC
quality	Sound quality TRTCAudioQualitySpeech - Smooth: sample rate: 16 kHz; mono channel; audio bitrate: 16 Kbps. This is suitable for audio call scenarios, such as online meeting and audio call. TRTCAudioQualityDefault - Default: sample rate: 48 kHz; mono channel; audio bitrate: 50 Kbps. This is the default sound quality of the SDK and recommended if there are no special requirements. TRTCAudioQualityMusic - HD: sample rate: 48 kHz; dual channel + full band; audio bitrate: 128 Kbps. This is suitable for scenarios where Hi-Fi music transfer is required, such as online karaoke and music live streaming.

Note

This API will check the mic permission. If the current application does not have permission to use the mic, the SDK will automatically ask the user to grant the mic permission.

stopLocalAudio

stopLocalAudio

Stop local audio capturing and publishing

After local audio capturing and publishing is stopped, other users in the room will receive the onUserAudioAvailable(userId, NO) notification.

muteLocalAudio:

muteLocalAudio:

- (void)muteLocalAudio:	(BOOL)mute

Pause/Resume publishing local audio stream

After local audio publishing is paused, other users in the room will receive the onUserAudioAvailable(userId, NO) notification.
After local audio publishing is resumed, other users in the room will receive the onUserAudioAvailable(userId, YES) notification.

Different from stopLocalAudio, `muteLocalAudio(YES)` does not release the mic permission; instead, it continues to send mute packets with extremely low bitrate.
This is very suitable for scenarios that require on-cloud recording, as video file formats such as MP4 have a high requirement for audio continuity, while an MP4 recording file cannot be played back smoothly if stopLocalAudio is used.
Therefore, `muteLocalAudio` instead of `stopLocalAudio` is recommended in scenarios where the requirement for recording file quality is high.

Param	DESC
mute	YES: mute; NO: unmute

muteRemoteAudio:mute:

muteRemoteAudio:mute:

- (void)muteRemoteAudio:	(NSString *)userId
mute:	(BOOL)mute

Pause/Resume playing back remote audio stream

When you mute the remote audio of a specified user, the SDK will stop playing back the user's audio and pulling the user's audio data.

Param	DESC
mute	YES: mute; NO: unmute
userId	ID of the specified remote user

Note

This API works when called either before or after room entry (enterRoom), and the mute status will be reset to `NO` after room exit (exitRoom).

muteAllRemoteAudio:

muteAllRemoteAudio:

- (void)muteAllRemoteAudio:	(BOOL)mute

Pause/Resume playing back all remote users' audio streams

When you mute the audio of all remote users, the SDK will stop playing back all their audio streams and pulling all their audio data.

Param	DESC
mute	YES: mute; NO: unmute

Note

This API works when called either before or after room entry (enterRoom), and the mute status will be reset to `NO` after room exit (exitRoom).

setAudioRoute:

setAudioRoute:

- (void)setAudioRoute:	(TRTCAudioRoute)route

Set audio route

Setting "audio route" is to determine whether the sound is played back from the speaker or receiver of a mobile device; therefore, this API is only applicable to mobile devices such as phones.

Generally, a phone has two speakers: one is the receiver at the top, and the other is the stereo speaker at the bottom.
If audio route is set to the receiver, the volume is relatively low, and the sound can be heard clearly only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.
If audio route is set to the speaker, the volume is relatively high, so there is no need to put the phone near the ear. Therefore, this mode can implement the "hands-free" feature.

Param	DESC
route	Audio route, i.e., whether the audio is output by speaker or receiver. Default value: TRTCAudioModeSpeakerphone

setRemoteAudioVolume:volume:

setRemoteAudioVolume:volume:

- (void)setRemoteAudioVolume:	(NSString *)userId
volume:	(int)volume

Set the audio playback volume of remote user

You can mute the audio of a remote user through `setRemoteAudioVolume(userId, 0)` .

Param	DESC
userId	ID of the specified remote user
volume	Volume. 100 is the original volume. Value range: [0,150]. Default value: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setAudioCaptureVolume:

setAudioCaptureVolume:

- (void)setAudioCaptureVolume:	(NSInteger)volume

Set the capturing volume of local audio

Param	DESC
volume	Volume. 100 is the original volume. Value range: [0,150]. Default value: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

getAudioCaptureVolume

getAudioCaptureVolume

Get the capturing volume of local audio

setAudioPlayoutVolume:

setAudioPlayoutVolume:

- (void)setAudioPlayoutVolume:	(NSInteger)volume

Set the playback volume of remote audio

This API controls the volume of the sound ultimately delivered by the SDK to the system for playback. It affects the volume of the recorded local audio file but not the volume of in-ear monitoring.

Param	DESC
volume	Volume. 100 is the original volume. Value range: [0,150]. Default value: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

getAudioPlayoutVolume

getAudioPlayoutVolume

Get the playback volume of remote audio

enableAudioVolumeEvaluation:withParams:

enableAudioVolumeEvaluation:withParams:

- (void)enableAudioVolumeEvaluation:	(BOOL)enable
withParams:	(TRTCAudioVolumeEvaluateParams *)params

Enable volume reminder

After this feature is enabled, the SDK will return the audio volume assessment information of local user who sends stream and remote users in the onUserVoiceVolume callback of TRTCCloudDelegate.

Param	DESC
enable	Whether to enable the volume prompt. It's disabled by default.
params	Volume evaluation and other related parameters, please see TRTCAudioVolumeEvaluateParams

Note

To enable this feature, call this API before calling `startLocalAudio` .

startAudioRecording:

startAudioRecording:

- (int)startAudioRecording:	(TRTCAudioRecordingParams*) param

Start audio recording

After you call this API, the SDK will selectively record local and remote audio streams (such as local audio, remote audio, background music, and sound effects) into a local file.

This API works when called either before or after room entry. If a recording task has not been stopped through `stopAudioRecording` before room exit, it will be automatically stopped after room exit.

The startup and completion status of the recording will be notified through local recording-related callbacks. See TRTCCloud related callbacks for reference.

Param	DESC
param	Recording parameter. For more information, please see TRTCAudioRecordingParams

Note

Since version 11.5, the results of audio recording have been changed to be notified through asynchronous callbacks instead of return values. Please refer to the relevant callbacks of TRTCCloud.

Return Desc:

0: success; -1: audio recording has been started; -2: failed to create file or directory; -3: the audio format of the specified file extension is not supported.

stopAudioRecording

stopAudioRecording

Stop audio recording

If a recording task has not been stopped through this API before room exit, it will be automatically stopped after room exit.

startLocalRecording:

startLocalRecording:

- (void)startLocalRecording:	(TRTCLocalRecordingParams *)params

Start local media recording

This API records the audio/video content during live streaming into a local file.

Param	DESC
params	Recording parameter. For more information, please see TRTCLocalRecordingParams

stopLocalRecording

stopLocalRecording

Stop local media recording

If a recording task has not been stopped through this API before room exit, it will be automatically stopped after room exit.

setRemoteAudioParallelParams:

setRemoteAudioParallelParams:

- (void)setRemoteAudioParallelParams:	(TRTCAudioParallelParams*)params

Set the parallel strategy of remote audio streams

For room with many speakers.

Param	DESC
params	Audio parallel parameter. For more information, please see TRTCAudioParallelParams

enable3DSpatialAudioEffect:

enable3DSpatialAudioEffect:

- (void)enable3DSpatialAudioEffect:	(BOOL)enabled

Enable 3D spatial effect

Enable 3D spatial effect. Note that TRTCAudioQualitySpeech smooth or TRTCAudioQualityDefault default audio quality should be used.

Param	DESC
enabled	Whether to enable 3D spatial effect. It’s disabled by default.

updateSelf3DSpatialPosition

updateSelf3DSpatialPosition

Update self position and orientation for 3D spatial effect

Update self position and orientation in the world coordinate system. The SDK will calculate the relative position between self and the remote users according to the parameters of this method, and then render the spatial sound effect. Note that the length of array should be 3.

Param	DESC
axisForward	The unit vector of the forward axis of user coordinate system. The three values represent the forward, right and up coordinate values in turn.

axisRight	The unit vector of the right axis of user coordinate system. The three values represent the forward, right and up coordinate values in turn.
axisUp	The unit vector of the up axis of user coordinate system. The three values represent the forward, right and up coordinate values in turn.
position	The coordinate of self in the world coordinate system. The three values represent the forward, right and up coordinate values in turn.

Note

Please limit the calling frequency appropriately. It's recommended that the interval between two operations be at least 100ms.

updateRemote3DSpatialPosition:

updateRemote3DSpatialPosition:

- (void)updateRemote3DSpatialPosition:	(NSString *)userId

Update the specified remote user's position for 3D spatial effect

Update the specified remote user's position in the world coordinate system. The SDK will calculate the relative position between self and the remote users according to the parameters of this method, and then render the spatial sound effect. Note that the length of array should be 3.

Param	DESC
position	The coordinate of self in the world coordinate system. The three values represent the forward, right and up coordinate values in turn.
userId	ID of the specified remote user.

Note

Please limit the calling frequency appropriately. It's recommended that the interval between two operations of the same remote user be at least 100ms.

set3DSpatialReceivingRange:range:

set3DSpatialReceivingRange:range:

- (void)set3DSpatialReceivingRange:	(NSString *)userId

range:	(NSInteger)range

Set the maximum 3D spatial attenuation range for userId's audio stream

After set the range, the specified user's audio stream will attenuate to zero within the range.

Param	DESC
range	Maximum attenuation range of the audio stream.
userId	ID of the specified user.

getDeviceManager

getDeviceManager

Get device management class (TXDeviceManager)

getBeautyManager

getBeautyManager

Get beauty filter management class (TXBeautyManager)

You can use the following features with beauty filter management:
 Set beauty effects such as "skin smoothing", "brightening", and "rosy skin".
 Set face adjustment effects such as "eye enlarging", "face slimming", "chin slimming", "chin lengthening/shortening", "face shortening", "nose narrowing", "eye brightening", "teeth whitening", "eye bag removal", "wrinkle removal", and "smile line removal".
 Set face adjustment effects such as "hairline", "eye distance", "eye corners", "mouth shape", "nose wing", "nose position", "lip thickness", and "face shape".
 Set makeup effects such as "eye shadow" and "blush".
 Set animated effects such as animated sticker and facial pendant.

setWatermark:streamType:rect:

setWatermark:streamType:rect:

- (void)setWatermark:	(nullable TXImage*)image
streamType:	(TRTCVideoStreamType)streamType

rect:	(CGRect)rect

Add watermark

The watermark position is determined by the `rect` parameter, which is a quadruple in the format of (x, y, width, height).

x: X coordinate of watermark, which is a floating-point number between 0 and 1.

y: Y coordinate of watermark, which is a floating-point number between 0 and 1.

width: width of watermark, which is a floating-point number between 0 and 1.

height: it does not need to be set. The SDK will automatically calculate it according to the watermark image's aspect ratio.

Sample parameter:

If the encoding resolution of the current video is 540x960, and the `rect` parameter is set to (0.1, 0.1, 0.2, 0.0), then the coordinates of the top-left point of the watermark will be (540 * 0.1, 960 * 0.1), i.e., (54, 96), the watermark width will be 540 * 0.2 = 108 px, and the watermark height will be calculated automatically by the SDK based on the watermark image's aspect ratio.

Param	DESC
image	Watermark image, which must be a PNG image with transparent background
rect	Unified coordinates of the watermark relative to the encoded resolution. Value range of `x`, `y`, `width`, and `height`: 0–1.
streamType	Specify for which image to set the watermark. For more information, please see TRTCVideoStreamType.

Note

If you want to set watermarks for both the primary image (generally for the camera) and the substream image (generally for screen sharing), you need to call this API twice with `streamType` set to different values.

getAudioEffectManager

getAudioEffectManager

Get sound effect management class (TXAudioEffectManager)

`TXAudioEffectManager` is a sound effect management API, through which you can implement the following features:

Background music: both online music and local music can be played back with various features such as speed adjustment, pitch adjustment, original voice, accompaniment, and loop.
In-ear monitoring: the sound captured by the mic is played back in the headphones in real time, which is generally used for music live streaming.
Reverb effect: karaoke room, small room, big hall, deep, resonant, and other effects.
Voice changing effect: young girl, middle-aged man, heavy metal, and other effects.
Short sound effect: short sound effect files such as applause and laughter are supported (for files less than 10 seconds in length, please set the `isShortFile` parameter to `YES`).

startSystemAudioLoopback

startSystemAudioLoopback

Enable system audio capturing(iOS not supported)

This API captures audio data from the sound card of a macOS computer and mixes it into the current audio data stream of the SDK, so that other users in the room can also hear the sound played back on the current macOS system.
In use cases such as video teaching or music live streaming, the teacher can use this feature to let the SDK capture the sound in the video played back by the teacher, so that students in the same room can also hear the sound in the video.
Note
1. This feature needs to install a virtual audio device plugin on the user's macOS system. After the installation is completed, the SDK will capture sound from the installed virtual device.
2. The SDK will automatically download the appropriate plugin from the internet for installation, but the download may be slow. If you want to speed up this process, you can package the virtual audio plugin file into the `Resources` directory of your app bundle.

stopSystemAudioLoopback

stopSystemAudioLoopback

Stop system audio capturing(iOS not supported)

setSystemAudioLoopbackVolume:

setSystemAudioLoopbackVolume:

- (void)setSystemAudioLoopbackVolume:	(uint32_t)volume

Set the volume of system audio capturing

Param	DESC
volume	Set volume. Value range: [0, 150]. Default value: 100

startScreenCaptureInApp:encParam:

startScreenCaptureInApp:encParam:

- (void)startScreenCaptureInApp:	(TRTCVideoStreamType)streamType
encParam:	(TRTCVideoEncParam *)encParams

Start in-app screen sharing (for iOS 13.0 and above only)

This API captures the real-time screen content of the current application and shares it with other users in the same room. It is applicable to iOS 13.0 and above.
If you want to capture the screen content of the entire iOS system (instead of the current application), we recommend you use startScreenCaptureByReplaykit.

Video encoding parameters recommended for screen sharing on iPhone (TRTCVideoEncParam):
Resolution (videoResolution): 1280x720
Frame rate (videoFps): 10 fps
Bitrate (videoBitrate): 1600 Kbps
Resolution adaption (enableAdjustRes): NO

Param	DESC
encParams	Video encoding parameters for screen sharing. We recommend you use the above configuration.If you set `encParams` to `nil`, the SDK will use the video encoding parameters you set before calling the startScreenCapture API.
streamType	Channel used for screen sharing, which can be the primary stream (TRTCVideoStreamTypeBig) or substream (TRTCVideoStreamTypeSub).

startScreenCaptureByReplaykit:encParam:appGroup:

startScreenCaptureByReplaykit:encParam:appGroup:

- (void)startScreenCaptureByReplaykit:	(TRTCVideoStreamType)streamType
encParam:	(TRTCVideoEncParam *)encParams
appGroup:	(NSString *)appGroup

Start system-level screen sharing (for iOS 11.0 and above only)

This API supports capturing the screen of the entire iOS system, which can implement system-wide screen sharing similar to VooV Meeting.
However, the integration steps are slightly more complicated than those of startScreenCaptureInApp. You need to implement a ReplayKit extension module for your application.

For more information, please see iOS

Video encoding parameters recommended for screen sharing on iPhone (TRTCVideoEncParam):
Resolution (videoResolution): 1280x720
Frame rate (videoFps): 10 fps
Bitrate (videoBitrate): 1600 Kbps
Resolution adaption (enableAdjustRes): NO

Param	DESC
appGroup	Specify the `Application Group Identifier` shared by your application and the screen sharing process. You can specify this parameter as `nil` , but we recommend you set it as instructed in the documentation for higher reliability.
encParams	Video encoding parameters for screen sharing. We recommend you use the above configuration. If you set `encParams` to `nil` , the SDK will use the video encoding parameters you set before calling the `startScreenCapture` API.
streamType	Channel used for screen sharing, which can be the primary stream (TRTCVideoStreamTypeBig) or substream (TRTCVideoStreamTypeSub).

startScreenCapture:streamType:encParam:

startScreenCapture:streamType:encParam:

- (void)startScreenCapture:	(nullable NSView *)view
streamType:	(TRTCVideoStreamType)streamType

encParam:	(nullable TRTCVideoEncParam *)encParam

Start screen sharing

This API can capture the content of the entire screen or a specified application and share it with other users in the same room.

Param	DESC
encParam	Image encoding parameters used for screen sharing, which can be set to empty, indicating to let the SDK choose the optimal encoding parameters (such as resolution and bitrate).
streamType	Channel used for screen sharing, which can be the primary stream (TRTCVideoStreamTypeBig) or substream (TRTCVideoStreamTypeSub).
view	Parent control of the rendering control, which can be set to a null value, indicating not to display the preview of the shared screen.

Note

1. A user can publish at most one primary stream (TRTCVideoStreamTypeBig) and one substream (TRTCVideoStreamTypeSub) at the same time.
2. By default, screen sharing uses the substream image. If you want to use the primary stream for screen sharing, you need to stop camera capturing (through stopLocalPreview) in advance to avoid conflicts.
3. Only one user can use the substream for screen sharing in the same room at any time; that is, only one user is allowed to enable the substream in the same room at any time.
4. When there is already a user in the room using the substream for screen sharing, calling this API will return the `onError(ERR_SERVER_CENTER_ANOTHER_USER_PUSH_SUB_VIDEO)` callback from TRTCCloudDelegate.

stopScreenCapture

stopScreenCapture

Stop screen sharing

pauseScreenCapture

pauseScreenCapture

Pause screen sharing

Note

Begin from v11.5 version, paused screen capture will use the last frame to output at a frame rate of 1fps.

resumeScreenCapture

resumeScreenCapture

Resume screen sharing

getScreenCaptureSourcesWithThumbnailSize:iconSize:

getScreenCaptureSourcesWithThumbnailSize:iconSize:

- (NSArray<TRTCScreenCaptureSourceInfo*>*)getScreenCaptureSourcesWithThumbnailSize:	(CGSize)thumbn
iconSize:	(CGSize)iconSiz

Enumerate shareable screens and windows (for macOS only)

When you integrate the screen sharing feature of a desktop system, you generally need to display a UI for selecting the sharing target, so that users can use the UI to choose whether to share the entire screen or a certain window. Through this API, you can query the IDs, names, and thumbnails of sharable windows on the current system. We provide a default UI implementation in the demo for your reference.

Param	DESC
iconSize	Specify the icon size of the window to be obtained.
thumbnailSize	Specify the thumbnail size of the window to be obtained. The thumbnail can be drawn on the window selection UI.

Note

The returned list contains the screen and the application windows. The screen is the first element in the list. If the user has multiple displays, then each display is a sharing target.

Return Desc:

List of windows (including the screen)

selectScreenCaptureTarget:rect:capturesCursor:highlight:

selectScreenCaptureTarget:rect:capturesCursor:highlight:

- (void)selectScreenCaptureTarget:	(TRTCScreenCaptureSourceInfo *)screenSource
rect:	(CGRect)rect
capturesCursor:	(BOOL)capturesCursor
highlight:	(BOOL)highlight

Select the screen or window to share (for macOS only)

After you get the sharable screen and windows through `getScreenCaptureSources` , you can call this API to select the target screen or window you want to share.

During the screen sharing process, you can also call this API at any time to switch the sharing target.

Param	DESC
capturesCursor	Whether to capture mouse cursor
highlight	Whether to highlight the window being shared
rect	Specify the area to be captured (set this parameter to `CGRectZero` : when the sharing target is a window, the entire window will be shared, and when the sharing target is the desktop, the entire desktop will be shared)
screenSource	Specify sharing source

setSubStreamEncoderParam:

setSubStreamEncoderParam:

- (void)setSubStreamEncoderParam:	(TRTCVideoEncParam *)param

Set the video encoding parameters of screen sharing (i.e., substream) (for desktop and mobile systems)

This API can set the image quality of screen sharing (i.e., the substream) viewed by remote users, which is also the image quality of screen sharing in on-cloud recording files.

Please note the differences between the following two APIs:

setVideoEncoderParam is used to set the video encoding parameters of the primary stream image (TRTCVideoStreamTypeBig, generally for camera).

setSubStreamEncoderParam is used to set the video encoding parameters of the substream image (TRTCVideoStreamTypeSub, generally for screen sharing).

Param	DESC

param	Substream encoding parameters. For more information, please see TRTCVideoEncParam.

setSubStreamMixVolume:

setSubStreamMixVolume:

- (void)setSubStreamMixVolume:	(NSInteger)volume

Set the audio mixing volume of screen sharing (for desktop systems only)

The greater the value, the larger the ratio of the screen sharing volume to the mic volume. We recommend you not set a high value for this parameter as a high volume will cover the mic sound.

Param	DESC
volume	Set audio mixing volume. Value range: 0–100

addExcludedShareWindow:

addExcludedShareWindow:

- (void)addExcludedShareWindow:	(NSInteger)windowID

Add specified windows to the exclusion list of screen sharing (for desktop systems only)

The excluded windows will not be shared. This feature is generally used to add a certain application's window to the exclusion list to avoid privacy issues.
You can set the filtered windows before starting screen sharing or dynamically add the filtered windows during screen sharing.

Param	DESC
window	Window not to be shared

Note

1. This API takes effect only if the `type` in TRTCScreenCaptureSourceInfo is specified as TRTCScreenCaptureSourceTypeScreen; that is, the feature of excluding specified windows works only when the entire screen is shared.
2. The windows added to the exclusion list through this API will be automatically cleared by the SDK after room exit.

3. On macOS, please pass in the window ID (CGWindowID), which can be obtained through the `sourceId` member in TRTCScreenCaptureSourceInfo.

removeExcludedShareWindow:

removeExcludedShareWindow:

- (void)removeExcludedShareWindow:	(NSInteger)windowID

Remove specified windows from the exclusion list of screen sharing (for desktop systems only)

Param	DESC
windowID	

removeAllExcludedShareWindows

removeAllExcludedShareWindows

Remove all windows from the exclusion list of screen sharing (for desktop systems only)

addIncludedShareWindow:

addIncludedShareWindow:

- (void)addIncludedShareWindow:	(NSInteger)windowID

Add specified windows to the inclusion list of screen sharing (for desktop systems only)

This API takes effect only if the `type` in TRTCScreenCaptureSourceInfo is specified as TRTCScreenCaptureSourceTypeWindow; that is, the feature of additionally including specified windows works only when a window is shared.
You can call it before or after startScreenCapture.

Param	DESC
windowID	Window to be shared (which is a window handle `HWND` on Windows)

Note

The windows added to the inclusion list by this method will be automatically cleared by the SDK after room exit.

removeIncludedShareWindow:

removeIncludedShareWindow:

- (void)removeIncludedShareWindow:	(NSInteger)windowID

Remove specified windows from the inclusion list of screen sharing (for desktop systems only)

This API takes effect only if the `type` in TRTCScreenCaptureSourceInfo is specified as TRTCScreenCaptureSourceTypeWindow.
That is, the feature of additionally including specified windows works only when a window is shared.

Param	DESC
windowID	Window to be shared (window ID on macOS or HWND on Windows)

removeAllIncludedShareWindows

removeAllIncludedShareWindows

Remove all windows from the inclusion list of screen sharing (for desktop systems only)

This API takes effect only if the `type` in TRTCScreenCaptureSourceInfo is specified as TRTCScreenCaptureSourceTypeWindow.
That is, the feature of additionally including specified windows works only when a window is shared.

enableCustomVideoCapture:enable:

enableCustomVideoCapture:enable:

- (void)enableCustomVideoCapture:	(TRTCVideoStreamType)streamType
enable:	(BOOL)enable

Enable/Disable custom video capturing mode

After this mode is enabled, the SDK will not run the original video capturing process (i.e., stopping camera data capturing and beauty filter operations) and will retain only the video encoding and sending capabilities.
You need to use sendCustomVideoData to continuously insert the captured video image into the SDK.

Param	DESC
enable	Whether to enable. Default value: NO
streamType	Specify video stream type (TRTCVideoStreamTypeBig: HD big image; TRTCVideoStreamTypeSub: substream image).

sendCustomVideoData:frame:

sendCustomVideoData:frame:

- (void)sendCustomVideoData:	(TRTCVideoStreamType)streamType
frame:	(TRTCVideoFrame *)frame

Deliver captured video frames to SDK

You can use this API to deliver video frames you capture to the SDK, and the SDK will encode and transfer them through its own network module.

We recommend you enter the following information for the TRTCVideoFrame parameter (other fields can be left empty):

pixelFormat: TRTCVideoPixelFormat_NV12 is recommended.
bufferType: TRTCVideoBufferType_PixelBuffer is recommended.
pixelBuffer: common video data format on iOS/macOS.
data: raw video data format, which is used if `bufferType` is `NSData` .
timestamp (ms): Set it to the timestamp when video frames are captured, which you can obtain by calling generateCustomPTS after getting a video frame.
width: video image length, which needs to be set if `bufferType` is `NSData` .
height: video image width, which needs to be set if `bufferType` is `NSData` .

For more information, please see Custom Capturing and Rendering.

Param	DESC
frame	Video data, which can be in PixelBuffer NV12, BGRA, or I420 format.
streamType	Specify video stream type (TRTCVideoStreamTypeBig: HD big image; TRTCVideoStreamTypeSub: substream image).

Note

1. We recommend you call the generateCustomPTS API to get the `timestamp` value of a video frame immediately after capturing it, so as to achieve the best audio/video sync effect.
2. The video frame rate eventually encoded by the SDK is not determined by the frequency at which you call this API, but by the FPS you set in setVideoEncoderParam.
3. Please try to keep the calling interval of this API even; otherwise, problems will occur, such as unstable output frame rate of the encoder or out-of-sync audio/video.

enableCustomAudioCapture:

enableCustomAudioCapture:

- (void)enableCustomAudioCapture:	(BOOL)enable

Enable custom audio capturing mode

After this mode is enabled, the SDK will not run the original audio capturing process (i.e., stopping mic data capturing) and will retain only the audio encoding and sending capabilities.
You need to use sendCustomAudioData to continuously insert the captured audio data into the SDK.

Param	DESC
enable	Whether to enable. Default value: NO

Note
As acoustic echo cancellation (AEC) requires strict control over the audio capturing and playback time, after custom audio capturing is enabled, AEC may fail.

sendCustomAudioData:

sendCustomAudioData:

- (void)sendCustomAudioData:	(TRTCAudioFrame *)frame

Deliver captured audio data to SDK

We recommend you enter the following information for the TRTCAudioFrame parameter (other fields can be left empty):
audioFormat: audio data format, which can only be `TRTCAudioFrameFormatPCM` .
data: audio frame buffer. Audio frame data must be in PCM format, and it supports a frame length of 5–100 ms (20 ms is recommended). Length calculation method: **for example, if the sample rate is 48000, then the frame length**

for mono channel will be `48000 * 0.02s * 1 * 16 bit = 15360 bit = 1920 bytes`.
sampleRate: sample rate. Valid values: 16000, 24000, 32000, 44100, 48000.
channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel.
timestamp (ms): Set it to the timestamp when audio frames are captured, which you can obtain by calling generateCustomPTS after getting a audio frame.

For more information, please see Custom Capturing and Rendering.

Param	DESC
frame	Audio data

Note

Please call this API accurately at intervals of the frame length; otherwise, sound lag may occur due to uneven data delivery intervals.

enableMixExternalAudioFrame:playout:

enableMixExternalAudioFrame:playout:

- (void)enableMixExternalAudioFrame:	(BOOL)enablePublish
playout:	(BOOL)enablePlayout

Enable/Disable custom audio track

After this feature is enabled, you can mix a custom audio track into the SDK through this API. With two boolean parameters, you can control whether to play back this track remotely or locally.

Param	DESC
enablePlayout	Whether the mixed audio track should be played back locally. Default value: NO
enablePublish	Whether the mixed audio track should be played back remotely. Default value: NO

Note

If you specify both `enablePublish` and `enablePlayout` as `NO`, the custom audio track will be completely closed.

mixExternalAudioFrame:

mixExternalAudioFrame:

- (int)mixExternalAudioFrame:	(TRTCAudioFrame *)frame

Mix custom audio track into SDK

Before you use this API to mix custom PCM audio into the SDK, you need to first enable custom audio tracks through enableMixExternalAudioFrame.

You are expected to feed audio data into the SDK at an even pace, but we understand that it can be challenging to call an API at absolutely regular intervals.

Given this, we have provided a buffer pool in the SDK, which can cache the audio data you pass in to reduce the fluctuations in intervals between API calls.

The value returned by this API indicates the size (ms) of the buffer pool. For example, if `50` is returned, it indicates that the buffer pool has 50 ms of audio data. As long as you call this API again within 50 ms, the SDK can make sure that continuous audio data is mixed.

If the value returned is `100` or greater, you can wait after an audio frame is played to call the API again. If the value returned is smaller than `100`, then there isn't enough data in the buffer pool, and you should feed more audio data into the SDK until the data in the buffer pool is above the safety level.

Fill the fields in TRTCAudioFrame as follows (other fields are not required).

`data` : audio frame buffer. Audio frames must be in PCM format. Each frame can be 5-100 ms (20 ms is recommended) in duration. Assume that the sample rate is 48000, and sound channels mono-channel. Then the **frame size would be 48000 x 0.02s x 1 x 16 bit = 15360 bit = 1920 bytes**.

`sampleRate` : sample rate. Valid values: 16000, 24000, 32000, 44100, 48000

`channel` : number of sound channels (if dual-channel is used, data is interleaved). Valid values: `1` (mono-channel); `2` (dual channel)

`timestamp` : timestamp (ms). Set it to the timestamp when audio frames are captured, which you can obtain by calling generateCustomPTS after getting an audio frame.

Param	DESC
frame	Audio data

Return Desc:

If the value returned is `0` or greater, the value represents the current size of the buffer pool; if the value returned is smaller than `0`, it means that an error occurred. `-1` indicates that you didn't call enableMixExternalAudioFrame to enable custom audio tracks.

setMixExternalAudioVolume:playoutVolume:

setMixExternalAudioVolume:playoutVolume:

- (void)setMixExternalAudioVolume:	(NSInteger)publishVolume
playoutVolume:	(NSInteger)playoutVolume

Set the publish volume and playback volume of mixed custom audio track

Param	DESC
playoutVolume	set the play volume, from 0 to 100, -1 means no change
publishVolume	set the publish volume, from 0 to 100, -1 means no change

generateCustomPTS

generateCustomPTS

Generate custom capturing timestamp

This API is only suitable for the custom capturing mode and is used to solve the problem of out-of-sync audio/video caused by the inconsistency between the capturing time and delivery time of audio/video frames.

When you call APIs such as sendCustomVideoData or sendCustomAudioData for custom video or audio capturing, please use this API as instructed below:
1. First, when a video or audio frame is captured, call this API to get the corresponding PTS timestamp.
2. Then, send the video or audio frame to the preprocessing module you use (such as a third-party beauty filter or sound effect component).
3. When you actually call sendCustomVideoData or sendCustomAudioData for delivery, assign the PTS timestamp recorded when the frame was captured to the `timestamp` field in TRTCVideoFrame or TRTCAudioFrame.

Return Desc:

Timestamp in ms

setLocalVideoProcessDelegete:pixelFormat:bufferType:

setLocalVideoProcessDelegete:pixelFormat:bufferType:

- (int)setLocalVideoProcessDelegete:	(nullable id<TRTCVideoFrameDelegate>)delegate
pixelFormat:	(TRTCVideoPixelFormat)pixelFormat

bufferType:	(TRTCVideoBufferType)bufferType

Set video data callback for third-party beauty filters

After this callback is set, the SDK will call back the captured video frames through the `delegate` you set and use them for further processing by a third-party beauty filter component. Then, the SDK will encode and send the processed video frames.

Param	DESC
bufferType	Specify the format of the data called back. Currently, only TRTCVideoBufferType_Texture is supported
delegate	Custom preprocessing callback. For more information, please see TRTCVideoFrameDelegate
pixelFormat	Specify the format of the pixel called back. Currently, only TRTCVideoPixelFormat_Texture_2D is supported

Return Desc:

0: success; values smaller than 0: error

setLocalVideoRenderDelegate:pixelFormat:bufferType:

setLocalVideoRenderDelegate:pixelFormat:bufferType:

- (int)setLocalVideoRenderDelegate:	(nullable id<TRTCVideoRenderDelegate>)delegate
pixelFormat:	(TRTCVideoPixelFormat)pixelFormat
bufferType:	(TRTCVideoBufferType)bufferType

Set the callback of custom rendering for local video

After this callback is set, the SDK will skip its own rendering process and call back the captured data. Therefore, you need to complete image rendering on your own.

`pixelFormat` specifies the format of the called back data, such as NV12, I420, and 32BGRA.

`bufferType` specifies the buffer type. `PixelBuffer` has the highest efficiency, while `NSData` makes the SDK perform a memory conversion internally, which will result in extra performance loss.

For more information, please see Custom Capturing and Rendering.

Param	DESC

bufferType	PixelBuffer: this can be directly converted to `UIImage` by using `imageWithCVImageBuffer` ; NSData: this is memory-mapped video data.
delegate	Callback for custom rendering
pixelFormat	Specify the format of the pixel called back

Return Desc:

0: success; values smaller than 0: error

setRemoteVideoRenderDelegate:delegate:pixelFormat:bufferType:

setRemoteVideoRenderDelegate:delegate:pixelFormat:bufferType:

- (int)setRemoteVideoRenderDelegate:	(NSString*)userId
delegate:	(nullable id<TRTCVideoRenderDelegate>)delegate
pixelFormat:	(TRTCVideoPixelFormat)pixelFormat
bufferType:	(TRTCVideoBufferType)bufferType

Set the callback of custom rendering for remote video

After this callback is set, the SDK will skip its own rendering process and call back the captured data. Therefore, you need to complete image rendering on your own.

`pixelFormat` specifies the format of the called back data, such as NV12, I420, and 32BGRA.

`bufferType` specifies the buffer type. `PixelBuffer` has the highest efficiency, while `NSData` makes the SDK perform a memory conversion internally, which will result in extra performance loss.

For more information, please see Custom Capturing and Rendering.

Param	DESC
bufferType	PixelBuffer: this can be directly converted to `UIImage` by using `imageWithCVImageBuffer` ; NSData: this is memory-mapped video data.
delegate	Callback for custom rendering
pixelFormat	Specify the format of the pixel called back
userId	ID of the specified remote user

Note

Before this API is called, `startRemoteView(nil)` needs to be called to get the video stream of the remote user (`view` can be set to `nil` for this end); otherwise, there will be no data called back.

Return Desc:

0: success; values smaller than 0: error

setAudioFrameDelegate:

setAudioFrameDelegate:

- (void)setAudioFrameDelegate:	(nullable id<TRTCAudioFrameDelegate>)delegate

Set custom audio data callback

After this callback is set, the SDK will internally call back the audio data (in PCM format), including:
onCapturedAudioFrame: callback of the audio data captured by the local mic
onLocalProcessedAudioFrame: callback of the audio data captured by the local mic and preprocessed by the audio module
onRemoteUserAudioFrame: audio data from each remote user before audio mixing
onMixedPlayAudioFrame: callback of the audio data that will be played back by the system after audio streams are mixed

Note

Setting the callback to null indicates to stop the custom audio callback, while setting it to a non-null value indicates to start the custom audio callback.

setCapturedAudioFrameDelegateFormat:

setCapturedAudioFrameDelegateFormat:

- (int)setCapturedAudioFrameDelegateFormat:	(TRTCAudioFrameDelegateFormat *)format

Set the callback format of audio frames captured by local mic

This API is used to set the `AudioFrame` format called back by onCapturedAudioFrame:
sampleRate: sample rate. Valid values: 16000, 32000, 44100, 48000
channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel
samplesPerCall: number of sample points, which defines the frame length of the callback data. The frame length must be an integer multiple of 10 ms.

If you want to calculate the callback frame length in milliseconds, the formula for converting the number of milliseconds into the number of sample points is as follows: number of sample points = number of milliseconds * sample rate / 1000
For example, if you want to call back the data of 20 ms frame length with 48000 sample rate, then the number of sample points should be entered as 960 = 20 * 48000 / 1000
Note that the frame length of the final callback is in bytes, and the calculation formula for converting the number of sample points into the number of bytes is as follows: number of bytes = number of sample points * number of channels * 2 (bit width)
For example, if the parameters are 48000 sample rate, dual channel, 20 ms frame length, and 960 sample points, then the number of bytes is 3840 = 960 * 2 * 2

Param	DESC
format	Audio data callback format

Return Desc:

0: success; values smaller than 0: error

setLocalProcessedAudioFrameDelegateFormat:

setLocalProcessedAudioFrameDelegateFormat:

- (int)setLocalProcessedAudioFrameDelegateFormat:	(TRTCAudioFrameDelegateFormat *)format

Set the callback format of preprocessed local audio frames

This API is used to set the `AudioFrame` format called back by onLocalProcessedAudioFrame:
sampleRate: sample rate. Valid values: 16000, 32000, 44100, 48000
channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel
samplesPerCall: number of sample points, which defines the frame length of the callback data. The frame length must be an integer multiple of 10 ms.

If you want to calculate the callback frame length in milliseconds, the formula for converting the number of milliseconds into the number of sample points is as follows: number of sample points = number of milliseconds * sample rate / 1000
For example, if you want to call back the data of 20 ms frame length with 48000 sample rate, then the number of sample points should be entered as 960 = 20 * 48000 / 1000

Note that the frame length of the final callback is in bytes, and the calculation formula for converting the number of sample points into the number of bytes is as follows: number of bytes = number of sample points * number of channels * 2 (bit width)
For example, if the parameters are 48000 sample rate, dual channel, 20 ms frame length, and 960 sample points, then the number of bytes is 3840 = 960 * 2 * 2

Param	DESC
format	Audio data callback format

Return Desc:

0: success; values smaller than 0: error

setMixedPlayAudioFrameDelegateFormat:

setMixedPlayAudioFrameDelegateFormat:

- (int)setMixedPlayAudioFrameDelegateFormat:	(TRTCAudioFrameDelegateFormat *)format

Set the callback format of audio frames to be played back by system

This API is used to set the `AudioFrame` format called back by onMixedPlayAudioFrame:
sampleRate: sample rate. Valid values: 16000, 32000, 44100, 48000
channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel
samplesPerCall: number of sample points, which defines the frame length of the callback data. The frame length must be an integer multiple of 10 ms.

If you want to calculate the callback frame length in milliseconds, the formula for converting the number of milliseconds into the number of sample points is as follows: number of sample points = number of milliseconds * sample rate / 1000
For example, if you want to call back the data of 20 ms frame length with 48000 sample rate, then the number of sample points should be entered as 960 = 20 * 48000 / 1000
Note that the frame length of the final callback is in bytes, and the calculation formula for converting the number of sample points into the number of bytes is as follows: number of bytes = number of sample points * number of channels * 2 (bit width)
For example, if the parameters are 48000 sample rate, dual channel, 20 ms frame length, and 960 sample points, then the number of bytes is 3840 = 960 * 2 * 2

Param	DESC

format	Audio data callback format

Return Desc:

0: success; values smaller than 0: error

enableCustomAudioRendering:

enableCustomAudioRendering:

- (void)enableCustomAudioRendering:	(BOOL)enable

Enabling custom audio playback

You can use this API to enable custom audio playback if you want to connect to an external audio device or control the audio playback logic by yourself.
After you enable custom audio playback, the SDK will stop using its audio API to play back audio. You need to call getCustomAudioRenderingFrame to get audio frames and play them by yourself.

Param	DESC
enable	Whether to enable custom audio playback. It's disabled by default.

Note

The parameter must be set before room entry to take effect.

getCustomAudioRenderingFrame:

getCustomAudioRenderingFrame:

- (void)getCustomAudioRenderingFrame:	(TRTCAudioFrame *)audioFrame

Getting playable audio data

Before calling this API, you need to first enable custom audio playback using enableCustomAudioRendering.

Fill the fields in TRTCAudioFrame as follows (other fields are not required):
`sampleRate` : sample rate (required). Valid values: 16000, 24000, 32000, 44100, 48000
`channel` : number of sound channels (required). `1` : mono-channel; `2` : dual-channel; if dual-channel is used, data is interleaved.

`data` : the buffer used to get audio data. You need to allocate memory for the buffer based on the duration of an audio frame.
The PCM data obtained can have a frame duration of 10 ms or 20 ms. 20 ms is recommended.
Assume that the sample rate is 48000, and sound channels mono-channel. The buffer size for a 20 ms audio frame would be 48000 x 0.02s x 1 x 16 bit = 15360 bit = 1920 bytes.

Param	DESC
audioFrame	Audio frames

Note
1. You must set `sampleRate` and `channel` in `audioFrame` , and allocate memory for one frame of audio in advance.
2. The SDK will fill the data automatically based on `sampleRate` and `channel` .
3. We recommend that you use the system's audio playback thread to drive the calling of this API, so that it is called each time the playback of an audio frame is complete.

sendCustomCmdMsg:data:reliable:ordered:

sendCustomCmdMsg:data:reliable:ordered:

- (BOOL)sendCustomCmdMsg:	(NSInteger)cmdID
data:	(NSData *)data
reliable:	(BOOL)reliable
ordered:	(BOOL)ordered

Use UDP channel to send custom message to all users in room

This API allows you to use TRTC's UDP channel to broadcast custom data to other users in the current room for signaling transfer.
Other users in the room can receive the message through the `onRecvCustomCmdMsg` callback in TRTCCloudDelegate.

Param	DESC
cmdID	Message ID. Value range: 1–10
data	Message to be sent. The maximum length of one single message is 1 KB.
ordered	Whether orderly sending is enabled, i.e., whether the data packets should be received in the

	same order in which they are sent; if so, a certain delay will be caused.
reliable	Whether reliable sending is enabled. Reliable sending can achieve a higher success rate but with a longer reception delay than unreliable sending.

Note

1. Up to 30 messages can be sent per second to all users in the room (this is not supported for web and mini program currently).
2. A packet can contain up to 1 KB of data; if the threshold is exceeded, the packet is very likely to be discarded by the intermediate router or server.
3. A client can send up to 8 KB of data in total per second.
4. `reliable` and `ordered` must be set to the same value (`YES` or `NO`) and cannot be set to different values currently.
5. We strongly recommend you set different `cmdID` values for messages of different types. This can reduce message delay when orderly sending is required.
6. Currently only the anchor role is supported.

Return Desc:

YES: sent the message successfully; NO: failed to send the message.

sendSEIMsg:repeatCount:

sendSEIMsg:repeatCount:

- (BOOL)sendSEIMsg:	(NSData *)data
repeatCount:	(int)repeatCount

Use SEI channel to send custom message to all users in room

This API allows you to use TRTC's SEI channel to broadcast custom data to other users in the current room for signaling transfer.

The header of a video frame has a header data block called SEI. This API works by embedding the custom signaling data you want to send in the SEI block and sending it together with the video frame.
Therefore, the SEI channel has a better compatibility than sendCustomCmdMsg as the signaling data can be transferred to the CSS CDN along with the video frame.
However, because the data block of the video frame header cannot be too large, we recommend you limit the size of the signaling data to only a few bytes when using this API.

The most common use is to embed the custom timestamp into video frames through this API so as to implement a perfect alignment between the message and video image (such as between the teaching material and video signal in the education scenario).
Other users in the room can receive the message through the `onRecvSEIMsg` callback in TRTCCloudDelegate.

Param	DESC
data	Data to be sent, which can be up to 1 KB (1,000 bytes)
repeatCount	Data sending count

Note
This API has the following restrictions:
1. The data will not be instantly sent after this API is called; instead, it will be inserted into the next video frame after the API call.
2. Up to 30 messages can be sent per second to all users in the room (this limit is shared with `sendCustomCmdMsg`).
3. Each packet can be up to 1 KB (this limit is shared with `sendCustomCmdMsg`). If a large amount of data is sent, the video bitrate will increase, which may reduce the video quality or even cause lagging.
4. Each client can send up to 8 KB of data in total per second (this limit is shared with `sendCustomCmdMsg`).
5. If multiple times of sending is required (i.e., `repeatCount` > 1), the data will be inserted into subsequent `repeatCount` video frames in a row for sending, which will increase the video bitrate.
6. If `repeatCount` is greater than 1, the data will be sent for multiple times, and the same message may be received multiple times in the `onRecvSEIMsg` callback; therefore, deduplication is required.

Return Desc:

YES: the message is allowed and will be sent with subsequent video frames; NO: the message is not allowed to be sent

startSpeedTest:

startSpeedTest:

- (int)startSpeedTest:	(TRTCSpeedTestParams *)params

Start network speed test (used before room entry)

Param	DESC

params	speed test options

Note

1. The speed measurement process will incur a small amount of basic service fees, See Purchase Guide > Base Services.
2. Please perform the Network speed test before room entry, because if performed after room entry, the test will affect the normal audio/video transfer, and its result will be inaccurate due to interference in the room.
3. Only one network speed test task is allowed to run at the same time.

Return Desc:

interface call result, <0: failure

stopSpeedTest

stopSpeedTest

Stop network speed test

getSDKVersion

getSDKVersion

Get SDK version information

setLogLevel:

setLogLevel:

+ (void)setLogLevel:	(TRTCLogLevel)level

Set log output level

Param	DESC
level	For more information, please see TRTCLogLevel. Default value: TRTCLogLevelNone

setConsoleEnabled:

setConsoleEnabled:

+ (void)setConsoleEnabled:	(BOOL)enabled

Enable/Disable console log printing

Param	DESC
enabled	Specify whether to enable it, which is disabled by default

setLogCompressEnabled:

setLogCompressEnabled:

+ (void)setLogCompressEnabled:	(BOOL)enabled

Enable/Disable local log compression

If compression is enabled, the log size will significantly reduce, but logs can be read only after being decompressed by the Python script provided by Tencent Cloud.
If compression is disabled, logs will be stored in plaintext and can be read directly in Notepad, but will take up more storage capacity.

Param	DESC
enabled	Specify whether to enable it, which is enabled by default

setLogDirPath:

setLogDirPath:

+ (void)setLogDirPath:	(NSString *)path

Set local log storage path

You can use this API to change the default storage path of the SDK's local logs, which is as follows:
Windows: C:/Users/[username]/AppData/Roaming/liteav/log, i.e., under `%appdata%/liteav/log` .
iOS or macOS: under `sandbox Documents/log` .
Android: under `/app directory/files/log/liteav/` .

Param	DESC

path	Log storage path

Note

Please be sure to call this API before all other APIs and make sure that the directory you specify exists and your application has read/write permissions of the directory.

setLogDelegate:

setLogDelegate:

+ (void)setLogDelegate:	(nullable id<TRTCLogDelegate>)logDelegate

Set log callback

showDebugView:

showDebugView:

- (void)showDebugView:	(NSInteger)showType

Display dashboard

"Dashboard" is a semi-transparent floating layer for debugging information on top of the video rendering control. It is used to display audio/video information and event information to facilitate integration and debugging.

Param	DESC
showType	0: does not display; 1: displays lite edition (only with audio/video information); 2: displays full edition (with audio/video information and event information).

setDebugViewMargin:margin:

setDebugViewMargin:margin:

- (void)setDebugViewMargin:	(NSString *)userId
margin:	(TXEdgeInsets)margin

Set dashboard margin

This API is used to adjust the position of the dashboard in the video rendering control. It must be called before `showDebugView` for it to take effect.

Param	DESC
margin	Inner margin of the dashboard. It should be noted that this is based on the percentage of `parentView`. Value range: 0–1
userId	User ID

callExperimentalAPI:

callExperimentalAPI:

- (NSString*)callExperimentalAPI:	(NSString*)jsonStr

Call experimental APIs

enablePayloadPrivateEncryption:params:

enablePayloadPrivateEncryption:params:

- (int)enablePayloadPrivateEncryption:	(BOOL)enabled
params:	(TRTCPayloadPrivateEncryptionConfig *)config

Enable or disable private encryption of media streams

In scenarios with high security requirements, TRTC recommends that you call the enablePayloadPrivateEncryption method to enable private encryption of media streams before joining a room.
After the user exits the room, the SDK will automatically close the private encryption. To re-enable private encryption, you need to call this method before the user joins the room again.

Param	DESC
config	Configure the algorithm and key for private encryption of media streams, please see TRTCPayloadPrivateEncryptionConfig.
enabled	Whether to enable media stream private encryption.

Note

TRTC has built-in encryption for media streams before transmission. After private encryption of media streams is enabled, it will be re-encrypted with the key and initial vector you pass in.

Return Desc:

Interface call result, 0: Method call succeeded, -1: The incoming parameter is invalid, -2: Your subscription has expired. If you want to renew it, Please update to RTC Engine Pro Plans and fill out application form. Approval is required before use.

TRTCCloudDelegate

Last updated：2024-06-06 15:26:14

Module: TRTCCloudDelegate @ TXLiteAVSDK

Function: event callback APIs for TRTC's video call feature

TRTCCloudDelegate

TRTCCloudDelegate

FuncList	DESC
onError:errMsg:extInfo:	Error event callback
onWarning:warningMsg:extInfo:	Warning event callback
onEnterRoom:	Whether room entry is successful
onExitRoom:	Room exit
onSwitchRole:errMsg:	Role switching
onSwitchRoom:errMsg:	Result of room switching
onConnectOtherRoom:errCode:errMsg:	Result of requesting cross-room call
onDisconnectOtherRoom:errMsg:	Result of ending cross-room call
onUpdateOtherRoomForwardMode:errMsg:	Result of changing the upstream capability of the cross-room anchor
onRemoteUserEnterRoom:	A user entered the room
onRemoteUserLeaveRoom:reason:	A user exited the room
onUserVideoAvailable:available:	A remote user published/unpublished primary stream video

onUserSubStreamAvailable:available:	A remote user published/unpublished substream video
onUserAudioAvailable:available:	A remote user published/unpublished audio
onFirstVideoFrame:streamType:width:height:	The SDK started rendering the first video frame of the local or a remote user
onFirstAudioFrame:	The SDK started playing the first audio frame of a remote user
onSendFirstLocalVideoFrame:	The first local video frame was published
onSendFirstLocalAudioFrame	The first local audio frame was published
onRemoteVideoStatusUpdated:streamType:streamStatus:reason:extrainfo:	Change of remote video status
onRemoteAudioStatusUpdated:streamStatus:reason:extrainfo:	Change of remote audio status
onUserVideoSizeChanged:streamType:newWidth:newHeight:	Change of remote video size
onNetworkQuality:remoteQuality:	Real-time network quality statistics
onStatistics:	Real-time statistics on technical metrics
onSpeedTestResult:	Callback of network speed test
onConnectionLost	The SDK was disconnected from the cloud
onTryToReconnect	The SDK is reconnecting to the cloud
onConnectionRecovery	The SDK is reconnected to the cloud
onCameraDidReady	The camera is ready
onMicDidReady	The mic is ready
onAudioRouteChanged:fromRoute:	The audio route changed (for

	mobile devices only)
onUserVoiceVolume:totalVolume:	Volume
onDevice:type:stateChanged:	The status of a local device changed (for desktop OS only)
onAudioDeviceCaptureVolumeChanged:muted:	The capturing volume of the mic changed
onAudioDevicePlayoutVolumeChanged:muted:	The playback volume changed
onSystemAudioLoopbackError:	Whether system audio capturing is enabled successfully (for macOS only)
onRecvCustomCmdMsgUserId:cmdID:seq:message:	Receipt of custom message
onMissCustomCmdMsgUserId:cmdID:errCode:missed:	Loss of custom message
onRecvSEIMsg:message:	Receipt of SEI message
onStartPublishing:errMsg:	Started publishing to Tencent Cloud CSS CDN
onStopPublishing:errMsg:	Stopped publishing to Tencent Cloud CSS CDN
onStartPublishCDNStream:errMsg:	Started publishing to non-Tencent Cloud's live streaming CDN
onStopPublishCDNStream:errMsg:	Stopped publishing to non-Tencent Cloud's live streaming CDN
onSetMixTranscodingConfig:errMsg:	Set the layout and transcoding parameters for On-Cloud MixTranscoding
onStartPublishMediaStream:code:message:extraInfo:	Callback for starting to publish
onUpdatePublishMediaStream:code:message:extraInfo:	Callback for modifying publishing parameters
onStopPublishMediaStream:code:message:extraInfo:	Callback for stopping publishing
onCdnStreamStateChanged:status:code:msg:extraInfo:	Callback for change of RTMP/RTMPS publishing status

onScreenCaptureStarted	Screen sharing started
onScreenCapturePaused:	Screen sharing was paused
onScreenCaptureResumed:	Screen sharing was resumed
onScreenCaptureStoped:	Screen sharing stopped
onLocalRecordBegin:storagePath:	Local recording started
onLocalRecording:storagePath:	Local media is being recorded
onLocalRecordFragment:	Record fragment finished.
onLocalRecordComplete:storagePath:	Local recording stopped
onUserEnter:	An anchor entered the room (disused)
onUserExit:reason:	An anchor left the room (disused)
onAudioEffectFinished:code:	Audio effects ended (disused)

TRTCVideoRenderDelegate

FuncList	DESC
onRenderVideoFrame:userId:streamType:	Custom video rendering

TRTCVideoFrameDelegate

FuncList	DESC
onGLContextCreated	An OpenGL context was created in the SDK.
onProcessVideoFrame:dstFrame:	Video processing by third-party beauty filters
onGLContextDestory	The OpenGL context in the SDK was destroyed

TRTCAudioFrameDelegate

FuncList	DESC
onCapturedAudioFrame:	Audio data captured by the local mic and pre-processed by the audio module
onLocalProcessedAudioFrame:	Audio data captured by the local mic, pre-processed by the audio module, effect-processed and BGM-mixed
onRemoteUserAudioFrame:userId:	Audio data of each remote user before audio mixing
onMixedPlayAudioFrame:	Data mixed from each channel before being submitted to the system for playback
onMixedAllAudioFrame:	Data mixed from all the captured and to-be-played audio in the SDK
onVoiceEarMonitorAudioFrame:	In-ear monitoring data

TRTCLogDelegate

FuncList	DESC
onLog:LogLevel:WhichModule:	Printing of local log

onError:errMsg:extInfo:

onError:errMsg:extInfo:

- (void)onError:	(TXLiteAVError)errCode
errMsg:	(nullable NSString *)errMsg
extInfo:	(nullable NSDictionary*)extInfo

Error event callback

Error event, which indicates that the SDK threw an irrecoverable error such as room entry failure or failure to start device

For more information, see Error Codes.

Param	DESC
errCode	Error code

errMsg	Error message
extInfo	Extended field. Certain error codes may carry extra information for troubleshooting.

onWarning:warningMsg:extInfo:

onWarning:warningMsg:extInfo:

- (void)onWarning:	(TXLiteAVWarning)warningCode
warningMsg:	(nullable NSString *)warningMsg
extInfo:	(nullable NSDictionary*)extInfo

Warning event callback

Warning event, which indicates that the SDK threw an error requiring attention, such as video lag or high CPU usage
For more information, see Error Codes.

Param	DESC
extInfo	Extended field. Certain warning codes may carry extra information for troubleshooting.
warningCode	Warning code
warningMsg	Warning message

onEnterRoom:

onEnterRoom:

- (void)onEnterRoom:	(NSInteger)result

Whether room entry is successful

After calling the `enterRoom()` API in `TRTCCloud` to enter a room, you will receive the `onEnterRoom(result)` callback from `TRTCCloudDelegate`.
If room entry succeeded, `result` will be a positive number (`result` > 0), indicating the time in milliseconds (ms) the room entry takes.
If room entry failed, `result` will be a negative number (result < 0), indicating the error code for the failure.
For more information on the error codes for room entry failure, see Error Codes.

Param	DESC

result	If `result` is greater than 0, it indicates the time (in ms) the room entry takes; if `result` is less than 0, it represents the error code for room entry.

Note

1. In TRTC versions below 6.6, the `onEnterRoom(result)` callback is returned only if room entry succeeds, and the `onError()` callback is returned if room entry fails.
2. In TRTC 6.6 and above, the `onEnterRoom(result)` callback is returned regardless of whether room entry succeeds or fails, and the `onError()` callback is also returned if room entry fails.

onExitRoom:

onExitRoom:

- (void)onExitRoom:	(NSInteger)reason

Room exit

Calling the `exitRoom()` API in `TRTCCloud` will trigger the execution of room exit-related logic, such as releasing resources of audio/video devices and codecs.
After all resources occupied by the SDK are released, the SDK will return the `onExitRoom()` callback.

If you need to call `enterRoom()` again or switch to another audio/video SDK, please wait until you receive the `onExitRoom()` callback.
Otherwise, you may encounter problems such as the camera or mic being occupied.

Param	DESC
reason	Reason for room exit. `0` : the user called `exitRoom` to exit the room; `1` : the user was removed from the room by the server; `2` : the room was dismissed.

onSwitchRole:errMsg:

onSwitchRole:errMsg:

- (void)onSwitchRole:	(TXLiteAVError)errCode
errMsg:	(nullable NSString *)errMsg

Role switching

You can call the `switchRole()` API in `TRTCCloud` to switch between the anchor and audience roles. This is accompanied by a line switching process.
After the switching, the SDK will return the `onSwitchRole()` event callback.

Param	DESC
errCode	Error code. `ERR_NULL` indicates a successful switch. For more information, please see Error Codes.
errMsg	Error message

onSwitchRoom:errMsg:

onSwitchRoom:errMsg:

- (void)onSwitchRoom:	(TXLiteAVError)errCode
errMsg:	(nullable NSString *)errMsg

Result of room switching

You can call the `switchRoom()` API in `TRTCCloud` to switch from one room to another.
After the switching, the SDK will return the `onSwitchRoom()` event callback.

Param	DESC
errCode	Error code. `ERR_NULL` indicates a successful switch. For more information, please see Error Codes.
errMsg	Error message

onConnectOtherRoom:errCode:errMsg:

onConnectOtherRoom:errCode:errMsg:

- (void)onConnectOtherRoom:	(NSString*)userId
errCode:	(TXLiteAVError)errCode
errMsg:	(nullable NSString *)errMsg

Result of requesting cross-room call

You can call the `connectOtherRoom()` API in `TRTCCloud` to establish a video call with the anchor of another room. This is the “anchor competition” feature.
The caller will receive the `onConnectOtherRoom()` callback, which can be used to determine whether the cross-room call is successful.
If it is successful, all users in either room will receive the `onUserVideoAvailable()` callback from the anchor of the other room.

Param	DESC
errCode	Error code. `ERR_NULL` indicates that cross-room connection is established successfully. For more information, please see Error Codes.
errMsg	Error message
userId	The user ID of the anchor (in another room) to be called

onDisconnectOtherRoom:errMsg:

onDisconnectOtherRoom:errMsg:

- (void)onDisconnectOtherRoom:	(TXLiteAVError)errCode
errMsg:	(nullable NSString *)errMsg

Result of ending cross-room call

onUpdateOtherRoomForwardMode:errMsg:

onUpdateOtherRoomForwardMode:errMsg:

- (void)onUpdateOtherRoomForwardMode:	(TXLiteAVError)errCode
errMsg:	(nullable NSString *)errMsg

Result of changing the upstream capability of the cross-room anchor

onRemoteUserEnterRoom:

onRemoteUserEnterRoom:

- (void)onRemoteUserEnterRoom:	(NSString *)userId

A user entered the room

Due to performance concerns, this callback works differently in different scenarios (i.e., `AppScene` , which you can specify by setting the second parameter when calling `enterRoom`).

Live streaming scenarios (`TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom`): in live streaming scenarios, a user is either in the role of an anchor or audience. The callback is returned only when an anchor enters the room.

Call scenarios (`TRTCAppSceneVideoCall` or `TRTCAppSceneAudioCall`): in call scenarios, the concept of roles does not apply (all users can be considered as anchors), and the callback is returned when any user enters the room.

Param	DESC
userId	User ID of the remote user

Note

1. The `onRemoteUserEnterRoom` callback indicates that a user entered the room, but it does not necessarily mean that the user enabled audio or video.
2. If you want to know whether a user enabled video, we recommend you use the `onUserVideoAvailable()` callback.

onRemoteUserLeaveRoom:reason:

onRemoteUserLeaveRoom:reason:

- (void)onRemoteUserLeaveRoom:	(NSString *)userId
reason:	(NSInteger)reason

A user exited the room

As with `onRemoteUserEnterRoom` , this callback works differently in different scenarios (i.e., `AppScene` , which you can specify by setting the second parameter when calling `enterRoom`).

Live streaming scenarios (`TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom`): the callback is triggered only when an anchor exits the room.

Call scenarios (`TRTCAppSceneVideoCall` or `TRTCAppSceneAudioCall`): in call scenarios, the concept of roles does not apply, and the callback is returned when any user exits the room.

Param	DESC

reason	Reason for room exit. `0` : the user exited the room voluntarily; `1` : the user exited the room due to timeout; `2` : the user was removed from the room; `3` : the anchor user exited the room due to switch to audience.
userId	User ID of the remote user

onUserVideoAvailable:available:

onUserVideoAvailable:available:

- (void)onUserVideoAvailable:	(NSString *)userId
available:	(BOOL)available

A remote user published/unpublished primary stream video

The primary stream is usually used for camera images. If you receive the `onUserVideoAvailable(userId, YES)` callback, it indicates that the user has available primary stream video.
You can then call startRemoteView to subscribe to the remote user's video. If the subscription is successful, you will receive the `onFirstVideoFrame(userid)` callback, which indicates that the first video frame of the user is rendered.

If you receive the `onUserVideoAvailable(userId, NO)` callback, it indicates that the video of the remote user is disabled, which may be because the user called muteLocalVideo or stopLocalPreview.

Param	DESC
available	Whether the user published (or unpublished) primary stream video. `YES` : published; `NO` : unpublished
userId	User ID of the remote user

onUserSubStreamAvailable:available:

onUserSubStreamAvailable:available:

- (void)onUserSubStreamAvailable:	(NSString *)userId
available:	(BOOL)available

A remote user published/unpublished substream video

The substream is usually used for screen sharing images. If you receive the `onUserSubStreamAvailable(userId, YES)` callback, it indicates that the user has available substream video.

You can then call startRemoteView to subscribe to the remote user's video. If the subscription is successful, you will receive the `onFirstVideoFrame(userid)` callback, which indicates that the first frame of the user is rendered.

Param	DESC
available	Whether the user published (or unpublished) substream video. `YES` : published; `NO` : unpublished
userId	User ID of the remote user

Note

The API used to display substream images is startRemoteView, not startRemoteSubStreamView, startRemoteSubStreamView is deprecated.

onUserAudioAvailable:available:

onUserAudioAvailable:available:

- (void)onUserAudioAvailable:	(NSString *)userId
available:	(BOOL)available

A remote user published/unpublished audio

If you receive the `onUserAudioAvailable(userId, YES)` callback, it indicates that the user published audio.

In auto-subscription mode, the SDK will play the user's audio automatically.

In manual subscription mode, you can call muteRemoteAudio(userid, NO) to play the user's audio.

Param	DESC
available	Whether the user published (or unpublished) audio. `YES` : published; `NO` : unpublished
userId	User ID of the remote user

Note

The auto-subscription mode is used by default. You can switch to the manual subscription mode by calling setDefaultStreamRecvMode, but it must be called before room entry for the switch to take effect.

onFirstVideoFrame:streamType:width:height:

onFirstVideoFrame:streamType:width:height:

- (void)onFirstVideoFrame:	(NSString*)userId
streamType:	(TRTCVideoStreamType)streamType
width:	(int)width
height:	(int)height

The SDK started rendering the first video frame of the local or a remote user

The SDK returns this event callback when it starts rendering your first video frame or that of a remote user. The `userId` in the callback can help you determine whether the frame is yours or a remote user's.

If `userId` is empty, it indicates that the SDK has started rendering your first video frame. The precondition is that you have called startLocalPreview or startScreenCapture.

If `userId` is not empty, it indicates that the SDK has started rendering the first video frame of a remote user. The precondition is that you have called startRemoteView to subscribe to the user's video.

Param	DESC
height	Video height
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	The user ID of the local or a remote user. If it is empty, it indicates that the first local video frame is available; if it is not empty, it indicates that the first video frame of a remote user is available.
width	Video width

Note

1. The callback of the first local video frame being rendered is triggered only after you call startLocalPreview or startScreenCapture.
2. The callback of the first video frame of a remote user being rendered is triggered only after you call startRemoteView or startRemoteSubStreamView.

onFirstAudioFrame:

onFirstAudioFrame:

- (void)onFirstAudioFrame:	(NSString*)userId

The SDK started playing the first audio frame of a remote user

The SDK returns this callback when it plays the first audio frame of a remote user. The callback is not returned for the playing of the first audio frame of the local user.

Param	DESC
userId	User ID of the remote user

onSendFirstLocalVideoFrame:

onSendFirstLocalVideoFrame:

- (void)onSendFirstLocalVideoFrame:	(TRTCVideoStreamType)streamType

The first local video frame was published

After you enter a room and call startLocalPreview or startScreenCapture to enable local video capturing (whichever happens first),
the SDK will start video encoding and publish the local video data via its network module to the cloud.
It returns the `onSendFirstLocalVideoFrame` callback after publishing the first local video frame.

Param	DESC
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.

onSendFirstLocalAudioFrame

onSendFirstLocalAudioFrame

The first local audio frame was published

After you enter a room and call startLocalAudio to enable audio capturing (whichever happens first),
the SDK will start audio encoding and publish the local audio data via its network module to the cloud.
The SDK returns the `onSendFirstLocalAudioFrame` callback after sending the first local audio frame.

onRemoteVideoStatusUpdated:streamType:streamStatus:reason:extrainfo:

onRemoteVideoStatusUpdated:streamType:streamStatus:reason:extrainfo:

- (void)onRemoteVideoStatusUpdated:	(NSString *)userId
streamType:	(TRTCVideoStreamType)streamType
streamStatus:	(TRTCAVStatusType)status
reason:	(TRTCAVStatusChangeReason)reason
extrainfo:	(nullable NSDictionary *)extrainfo

Change of remote video status

You can use this callback to get the status (`Playing` , `Loading` , or `Stopped`) of the video of each remote user and display it on the UI.

Param	DESC
extraInfo	Extra information
reason	Reason for the change of status
status	Video status, which may be `Playing` , `Loading` , or `Stopped`
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	User ID

onRemoteAudioStatusUpdated:streamStatus:reason:extrainfo:

onRemoteAudioStatusUpdated:streamStatus:reason:extrainfo:

- (void)onRemoteAudioStatusUpdated:	(NSString *)userId
streamStatus:	(TRTCAVStatusType)status
reason:	(TRTCAVStatusChangeReason)reason
extrainfo:	(nullable NSDictionary *)extrainfo

Change of remote audio status

You can use this callback to get the status (`Playing` , `Loading` , or `Stopped`) of the audio of each remote user and display it on the UI.

Param	DESC
extraInfo	Extra information
reason	Reason for the change of status
status	Audio status, which may be `Playing` , `Loading` , or `Stopped`
userId	User ID

onUserVideoSizeChanged:streamType:newWidth:newHeight:

onUserVideoSizeChanged:streamType:newWidth:newHeight:

- (void)onUserVideoSizeChanged:	(NSString *)userId
streamType:	(TRTCVideoStreamType)streamType
newWidth:	(int)newWidth
newHeight:	(int)newHeight

Change of remote video size

If you receive the `onUserVideoSizeChanged(userId, streamtype, newWidth, newHeight)` callback, it indicates that the user changed the video size. It may be triggered by `setVideoEncoderParam` or `setSubStreamEncoderParam` .

Param	DESC
newHeight	Video height
newWidth	Video width
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	User ID

onNetworkQuality:remoteQuality:

onNetworkQuality:remoteQuality:

- (void)onNetworkQuality:	(TRTCQualityInfo*)localQuality
remoteQuality:	(NSArray<TRTCQualityInfo*>*)remoteQuality

Real-time network quality statistics

This callback is returned every 2 seconds and notifies you of the upstream and downstream network quality detected by the SDK.

The SDK uses a built-in proprietary algorithm to assess the current latency, bandwidth, and stability of the network and returns a result.

If the result is `1` (excellent), it means that the current network conditions are excellent; if it is `6` (down), it means that the current network conditions are too bad to support TRTC calls.

Param	DESC
localQuality	Upstream network quality
remoteQuality	Downstream network quality, it refers to the data quality finally measured on the local side after the data flow passes through a complete transmission link of "remote ->cloud ->local". Therefore, the downlink network quality here represents the joint impact of the remote uplink and the local downlink.

Note

The uplink quality of remote users cannot be determined independently through this interface.

onStatistics:

onStatistics:

- (void)onStatistics:	(TRTCStatistics *)statistics

Real-time statistics on technical metrics

This callback is returned every 2 seconds and notifies you of the statistics on technical metrics related to video, audio, and network. The metrics are listed in TRTCStatistics:

Video statistics: video resolution (`resolution`), frame rate (`FPS`), bitrate (`bitrate`), etc.

Audio statistics: audio sample rate (`samplerate`), number of audio channels (`channel`), bitrate (`bitrate`), etc.

Network statistics: the round trip time (`rtt`) between the SDK and the cloud (SDK -> Cloud -> SDK), package loss rate (`loss`), upstream traffic (`sentBytes`), downstream traffic (`receivedBytes`), etc.

Param	DESC
statistics	Statistics, including local statistics and the statistics of remote users. For details, please see TRTCStatistics.

Note

If you want to learn about only the current network quality and do not want to spend much time analyzing the statistics returned by this callback, we recommend you use onNetworkQuality.

onSpeedTestResult:

onSpeedTestResult:

- (void)onSpeedTestResult:	(TRTCSpeedTestResult *)result

Callback of network speed test

The callback is triggered by startSpeedTest:.

Param	DESC
result	Speed test data, including loss rates, rtt and bandwidth rates, please refer to TRTCSpeedTestResult for details.

onConnectionLost

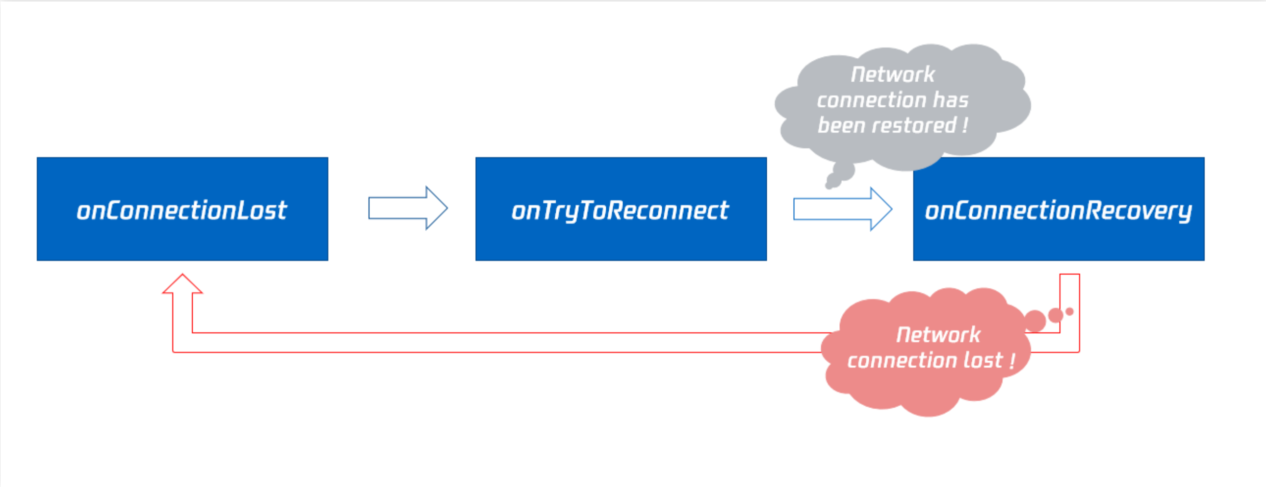
onConnectionLost

The SDK was disconnected from the cloud

The SDK returns this callback when it is disconnected from the cloud, which may be caused by network unavailability or change of network, for example, when the user walks into an elevator.

After returning this callback, the SDK will attempt to reconnect to the cloud, and will return the onTryToReconnect callback. When it is reconnected, it will return the onConnectionRecovery callback.

In other words, the SDK proceeds from one event to the next in the following order:



onTryToReconnect

onTryToReconnect

The SDK is reconnecting to the cloud

When the SDK is disconnected from the cloud, it returns the onConnectionLost callback. It then attempts to reconnect and returns this callback (onTryToReconnect). After it is reconnected, it returns the onConnectionRecovery callback.

onConnectionRecovery

onConnectionRecovery

The SDK is reconnected to the cloud

When the SDK is disconnected from the cloud, it returns the onConnectionLost callback. It then attempts to reconnect and returns the onTryToReconnect callback. After it is reconnected, it returns this callback (onConnectionRecovery).

onCameraDidReady

onCameraDidReady

The camera is ready

After you call startLocalPreivew, the SDK will try to start the camera and return this callback if the camera is started.

If it fails to start the camera, it’s probably because the application does not have access to the camera or the camera is being used.
You can capture the onError callback to learn about the exception and let users know via UI messages.

onMicDidReady

onMicDidReady

The mic is ready

After you call startLocalAudio, the SDK will try to start the mic and return this callback if the mic is started.
If it fails to start the mic, it’s probably because the application does not have access to the mic or the mic is being used.
You can capture the onError callback to learn about the exception and let users know via UI messages.

onAudioRouteChanged:fromRoute:

onAudioRouteChanged:fromRoute:

- (void)onAudioRouteChanged:	(TRTCAudioRoute)route
fromRoute:	(TRTCAudioRoute)fromRoute

The audio route changed (for mobile devices only)

Audio route is the route (speaker or receiver) through which audio is played.
When audio is played through the receiver, the volume is relatively low, and the sound can be heard only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.
When audio is played through the speaker, the volume is relatively high, and there is no need to put the phone near the ear. This mode enables the "hands-free" feature.
When audio is played through the wired earphone.
When audio is played through the bluetooth earphone.
When audio is played through the USB sound card.

Param	DESC
fromRoute	The audio route used before the change
route	Audio route, i.e., the route (speaker or receiver) through which audio is played

onUserVoiceVolume:totalVolume:

onUserVoiceVolume:totalVolume:

- (void)onUserVoiceVolume:	(NSArray<TRTCVolumeInfo *> *)userVolumes
totalVolume:	(NSInteger)totalVolume

Volume

The SDK can assess the volume of each channel and return this callback on a regular basis. You can display, for example, a waveform or volume bar on the UI based on the statistics returned.

You need to first call enableAudioVolumeEvaluation to enable the feature and set the interval for the callback.

Note that the SDK returns this callback at the specified interval regardless of whether someone is speaking in the room.

Param	DESC
totalVolume	The total volume of all remote users. Value range: 0-100
userVolumes	An array that represents the volume of all users who are speaking in the room. Value range: 0-100

Note

`userVolumes` is an array. If `userId` is empty, the elements in the array represent the volume of the local user’s audio. Otherwise, they represent the volume of a remote user’s audio.

onDevice:type:stateChanged:

onDevice:type:stateChanged:

- (void)onDevice:	(NSString *)deviceId
type:	(TRTCMediaDeviceType)deviceType
stateChanged:	(NSInteger)state

The status of a local device changed (for desktop OS only)

The SDK returns this callback when a local device (camera, mic, or speaker) is connected or disconnected.

Param	DESC
deviceId	Device ID

deviceType	Device type
state	Device status. `0` : disconnected; `1` : connected

onAudioDeviceCaptureVolumeChanged:muted:

onAudioDeviceCaptureVolumeChanged:muted:

- (void)onAudioDeviceCaptureVolumeChanged:	(NSInteger)volume
muted:	(BOOL)muted

The capturing volume of the mic changed

On desktop OS such as macOS and Windows, users can set the capturing volume of the mic in the audio control panel.
The higher volume a user sets, the higher the volume of raw audio captured by the mic.
On some keyboards and laptops, users can also mute the mic by pressing a key (whose icon is a crossed out mic).
When users set the mic capturing volume via the UI or a keyboard shortcut, the SDK will return this callback.

Param	DESC
muted	Whether the mic is muted. `YES` : muted; `NO` : unmuted
volume	System audio capturing volume, which users can set in the audio control panel. Value range: 0-100

Note

You need to call enableAudioVolumeEvaluation and set the callback interval (`interval` > 0) to enable the callback. To disable the callback, set `interval` to `0` .

onAudioDevicePlayoutVolumeChanged:muted:

onAudioDevicePlayoutVolumeChanged:muted:

- (void)onAudioDevicePlayoutVolumeChanged:	(NSInteger)volume
muted:	(BOOL)muted

The playback volume changed

On desktop OS such as macOS and Windows, users can set the system's playback volume in the audio control panel. On some keyboards and laptops, users can also mute the speaker by pressing a key (whose icon is a crossed out speaker).

When users set the system's playback volume via the UI or a keyboard shortcut, the SDK will return this callback.

Param	DESC
muted	Whether the speaker is muted. `YES` : muted; `NO` : unmuted
volume	The system playback volume, which users can set in the audio control panel. Value range: 0-100

Note

You need to call enableAudioVolumeEvaluation and set the callback interval (`interval` > 0) to enable the callback. To disable the callback, set `interval` to `0` .

onSystemAudioLoopbackError:

onSystemAudioLoopbackError:

- (void)onSystemAudioLoopbackError:	(TXLiteAVError)err

Whether system audio capturing is enabled successfully (for macOS only)

On macOS, you can call startSystemAudioLoopback to install an audio driver and have the SDK capture the audio played back by the system.

In use cases such as video teaching and music live streaming, the teacher can use this feature to let the SDK capture the sound of the video played by his or her computer, so that students in the room can hear the sound too.

The SDK returns this callback after trying to enable system audio capturing. To determine whether it is actually enabled, pay attention to the error parameter in the callback.

Param	DESC
err	If it is `ERR_NULL` , system audio capturing is enabled successfully. Otherwise, it is not.

onRecvCustomCmdMsgUserId:cmdID:seq:message:

onRecvCustomCmdMsgUserId:cmdID:seq:message:

- (void)onRecvCustomCmdMsgUserId:	(NSString *)userId

cmdID:	(NSInteger)cmdID
seq:	(UInt32)seq
message:	(NSData *)message

Receipt of custom message

When a user in a room uses sendCustomCmdMsg to send a custom message, other users in the room can receive the message through the `onRecvCustomCmdMsg` callback.

Param	DESC
cmdID	Command ID
message	Message data
seq	Message serial number
userId	User ID

onMissCustomCmdMsgUserId:cmdID:errCode:missed:

onMissCustomCmdMsgUserId:cmdID:errCode:missed:

- (void)onMissCustomCmdMsgUserId:	(NSString *)userId
cmdID:	(NSInteger)cmdID
errCode:	(NSInteger)errCode
missed:	(NSInteger)missed

Loss of custom message

When you use sendCustomCmdMsg to send a custom UDP message, even if you enable reliable transfer (by setting `reliable` to `YES`), there is still a chance of message loss. Reliable transfer only helps maintain a low probability of message loss, which meets the reliability requirements in most cases.
If the sender sets `reliable` to `YES` , the SDK will use this callback to notify the recipient of the number of custom messages lost during a specified time period (usually 5s) in the past.

Param	DESC
cmdID	Command ID

errCode	Error code
missed	Number of lost messages
userId	User ID

Note

The recipient receives this callback only if the sender sets `reliable` to `YES` .

onRecvSEIMsg:message:

onRecvSEIMsg:message:

- (void)onRecvSEIMsg:	(NSString *)userId
message:	(NSData*)message

Receipt of SEI message

If a user in the room uses sendSEIMsg to send an SEI message via video frames, other users in the room can receive the message through the `onRecvSEIMsg` callback.

Param	DESC
message	Data
userId	User ID

onStartPublishing:errMsg:

onStartPublishing:errMsg:

- (void)onStartPublishing:	(int)err
errMsg:	(NSString*)errMsg

Started publishing to Tencent Cloud CSS CDN

When you call startPublishing to publish streams to Tencent Cloud CSS CDN, the SDK will sync the command to the CVM immediately.
The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC

err	`0` : successful; other values: failed
errMsg	Error message

onStopPublishing:errMsg:

onStopPublishing:errMsg:

- (void)onStopPublishing:	(int)err
errMsg:	(NSString*)errMsg

Stopped publishing to Tencent Cloud CSS CDN

When you call stopPublishing to stop publishing streams to Tencent Cloud CSS CDN, the SDK will sync the command to the CVM immediately.
The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onStartPublishCDNStream:errMsg:

onStartPublishCDNStream:errMsg:

- (void)onStartPublishCDNStream:	(int)err
errMsg:	(NSString *)errMsg

Started publishing to non-Tencent Cloud's live streaming CDN

When you call startPublishCDNStream to start publishing streams to a non-Tencent Cloud's live streaming CDN, the SDK will sync the command to the CVM immediately.
The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

Note

If you receive a callback that the command is executed successfully, it only means that your command was sent to Tencent Cloud's backend server. If the CDN vendor does not accept your streams, the publishing will still fail.

onStopPublishCDNStream:errMsg:

onStopPublishCDNStream:errMsg:

- (void)onStopPublishCDNStream:	(int)err
errMsg:	(NSString *)errMsg

Stopped publishing to non-Tencent Cloud's live streaming CDN

When you call stopPublishCDNStream to stop publishing to a non-Tencent Cloud's live streaming CDN, the SDK will sync the command to the CVM immediately.

The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onSetMixTranscodingConfig:errMsg:

onSetMixTranscodingConfig:errMsg:

- (void)onSetMixTranscodingConfig:	(int)err
errMsg:	(NSString*)errMsg

Set the layout and transcoding parameters for On-Cloud MixTranscoding

When you call setMixTranscodingConfig to modify the layout and transcoding parameters for On-Cloud MixTranscoding, the SDK will sync the command to the CVM immediately.

The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onStartPublishMediaStream:code:message:extraInfo:

onStartPublishMediaStream:code:message:extraInfo:

- (void)onStartPublishMediaStream:	(NSString*)taskId
code:	(int)code
message:	(NSString*)message
extraInfo:	(nullable NSDictionary *)extraInfo

Callback for starting to publish

When you call startPublishMediaStream to publish a stream to the TRTC backend, the SDK will immediately update the command to the cloud server.

The SDK will then receive the publishing result from the cloud server and will send the result to you via this callback.

Param	DESC
code	: `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The callback information.
taskId	: If a request is successful, a task ID will be returned via the callback. You need to provide this task ID when you call updatePublishMediaStream to modify publishing parameters or stopPublishMediaStream to stop publishing.

onUpdatePublishMediaStream:code:message:extraInfo:

onUpdatePublishMediaStream:code:message:extraInfo:

- (void)onUpdatePublishMediaStream:	(NSString*)taskId
code:	(int)code
message:	(NSString*)message
extraInfo:	(nullable NSDictionary *)extraInfo

Callback for modifying publishing parameters

When you call updatePublishMediaStream to modify publishing parameters, the SDK will immediately update the command to the cloud server.
The SDK will then receive the modification result from the cloud server and will send the result to you via this callback.

Param	DESC
code	: `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The callback information.
taskId	: The task ID you pass in when calling updatePublishMediaStream, which is used to identify a request.

onStopPublishMediaStream:code:message:extraInfo:

onStopPublishMediaStream:code:message:extraInfo:

- (void)onStopPublishMediaStream:	(NSString*)taskId
code:	(int)code
message:	(NSString*)message
extraInfo:	(nullable NSDictionary *)extraInfo

Callback for stopping publishing

When you call stopPublishMediaStream to stop publishing, the SDK will immediately update the command to the cloud server.
The SDK will then receive the modification result from the cloud server and will send the result to you via this callback.

Param	DESC
code	: `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The callback information.
taskId	: The task ID you pass in when calling stopPublishMediaStream, which is used to identify a request.

onCdnStreamStateChanged:status:code:msg:extraInfo:

onCdnStreamStateChanged:status:code:msg:extraInfo:

- (void)onCdnStreamStateChanged:	(NSString*)cdnUrl
status:	(int)status
code:	(int)code
msg:	(NSString*)msg
extraInfo:	(nullable NSDictionary *)info

Callback for change of RTMP/RTMPS publishing status

When you call startPublishMediaStream to publish a stream to the TRTC backend, the SDK will immediately update the command to the cloud server.

If you set the publishing destination (TRTCPublishTarget) to the URL of Tencent Cloud or a third-party CDN, you will be notified of the RTMP/RTMPS publishing status via this callback.

Param	DESC
cdnUrl	: The URL you specify in TRTCPublishTarget when you call startPublishMediaStream.
code	: The publishing result. `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The publishing information.
status	: The publishing status. 0: The publishing has not started yet or has ended. This value will be returned after you call stopPublishMediaStream. 1: The TRTC server is connecting to the CDN server. If the first attempt fails, the TRTC backend will retry multiple times and will return this value via the callback (every five seconds). After publishing succeeds, the value `2` will be returned. If a server error occurs or publishing is still unsuccessful after 60 seconds, the value `4` will be returned. 2: The TRTC server is publishing to the CDN. This value will be returned if the publishing succeeds. 3: The TRTC server is disconnected from the CDN server and is reconnecting. If a CDN error occurs or publishing is interrupted, the TRTC backend will try to reconnect and resume publishing and will return this value via the callback (every five seconds). After publishing resumes, the value `2` will be returned. If a server error occurs or the attempt to resume publishing is still unsuccessful after 60 seconds, the value `4` will be returned.

	4: The TRTC server is disconnected from the CDN server and failed to reconnect within the timeout period. In this case, the publishing is deemed to have failed. You can call updatePublishMediaStream to try again. 5: The TRTC server is disconnecting from the CDN server. After you call stopPublishMediaStream, the SDK will return this value first and then the value `0` .

onScreenCaptureStarted

onScreenCaptureStarted

Screen sharing started

The SDK returns this callback when you call startScreenCapture and other APIs to start screen sharing.

onScreenCapturePaused:

onScreenCapturePaused:

- (void)onScreenCapturePaused:	(int)reason

Screen sharing was paused

The SDK returns this callback when you call pauseScreenCapture to pause screen sharing.

Param	DESC
reason	Reason. `0` : the user paused screen sharing. `1` : screen sharing was paused because the shared window became invisible(Mac). screen sharing was paused because setting parameters(Windows). `2` : screen sharing was paused because the shared window became minimum(only for Windows). `3` : screen sharing was paused because the shared window became invisible(only for Windows).

onScreenCaptureResumed:

onScreenCaptureResumed:

- (void)onScreenCaptureResumed:	(int)reason

Screen sharing was resumed

The SDK returns this callback when you call resumeScreenCapture to resume screen sharing.

Param	DESC
reason	Reason. `0` : the user resumed screen sharing. `1` : screen sharing was resumed automatically after the shared window became visible again(Mac). screen sharing was resumed automatically after setting parameters(Windows). `2` : screen sharing was resumed automatically after the shared window became minimize recovery(only for Windows). `3` : screen sharing was resumed automatically after the shared window became visible again(only for Windows).

onScreenCaptureStoped:

onScreenCaptureStoped:

- (void)onScreenCaptureStoped:	(int)reason

Screen sharing stopped

The SDK returns this callback when you call stopScreenCapture to stop screen sharing.

Param	DESC
reason	Reason. `0` : the user stopped screen sharing; `1` : screen sharing stopped because the shared window was closed.

onLocalRecordBegin:storagePath:

onLocalRecordBegin:storagePath:

- (void)onLocalRecordBegin:	(NSInteger)errCode
storagePath:	(NSString *)storagePath

Local recording started

When you call startLocalRecording to start local recording, the SDK returns this callback to notify you whether recording is started successfully.

Param	DESC

errCode	status. 0: successful. -1: failed. -2: unsupported format. -6: recording has been started. Stop recording first. -7: recording file already exists and needs to be deleted. -8: recording directory does not have the write permission. Please check the directory permission.
storagePath	Storage path of recording file

onLocalRecording:storagePath:

onLocalRecording:storagePath:

- (void)onLocalRecording:	(NSInteger)duration
storagePath:	(NSString *)storagePath

Local media is being recorded

The SDK returns this callback regularly after local recording is started successfully via the calling of startLocalRecording.

You can capture this callback to stay up to date with the status of the recording task.

You can set the callback interval when calling startLocalRecording.

Param	DESC
duration	Cumulative duration of recording, in milliseconds
storagePath	Storage path of recording file

onLocalRecordFragment:

onLocalRecordFragment:

- (void)onLocalRecordFragment:	(NSString *)storagePath

Record fragment finished.

When fragment recording is enabled, this callback will be invoked when each fragment file is finished.

Param	DESC

storagePath	Storage path of the fragment.

onLocalRecordComplete:storagePath:

onLocalRecordComplete:storagePath:

- (void)onLocalRecordComplete:	(NSInteger)errCode
storagePath:	(NSString *)storagePath

Local recording stopped

When you call stopLocalRecording to stop local recording, the SDK returns this callback to notify you of the recording result.

Param	DESC
errCode	status 0: successful. -1: failed. -2: Switching resolution or horizontal and vertical screen causes the recording to stop. -3: recording duration is too short or no video or audio data is received. Check the recording duration or whether audio or video capture is enabled.
storagePath	Storage path of recording file

onUserEnter:

onUserEnter:

- (void)onUserEnter:	(NSString *)userId

An anchor entered the room (disused)

@deprecated This callback is not recommended in the new version. Please use onRemoteUserEnterRoom instead.

onUserExit:reason:

onUserExit:reason:

- (void)onUserExit:	(NSString *)userId

reason:	(NSInteger)reason

An anchor left the room (disused)

@deprecated This callback is not recommended in the new version. Please use onRemoteUserLeaveRoom instead.

onAudioEffectFinished:code:

onAudioEffectFinished:code:

- (void)onAudioEffectFinished:	(int) effectId
code:	(int) code

Audio effects ended (disused)

@deprecated This callback is not recommended in the new version. Please use ITXAudioEffectManager instead.
Audio effects and background music can be started using the same API (startPlayMusic) now instead of separate ones.

onRenderVideoFrame:userId:streamType:

onRenderVideoFrame:userId:streamType:

- (void) onRenderVideoFrame:	(TRTCVideoFrame * _Nonnull)frame
userId:	(NSString* __nullable)userId
streamType:	(TRTCVideoStreamType)streamType

Custom video rendering

If you have configured the callback of custom rendering for local or remote video, the SDK will return to you via this callback video frames that are otherwise sent to the rendering control, so that you can customize rendering.

Param	DESC
frame	Video frames to be rendered
streamType	Stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	`userId` of the video source. This parameter can be ignored if the callback is for local video (`setLocalVideoRenderDelegate`).

onGLContextCreated

onGLContextCreated

An OpenGL context was created in the SDK.

onProcessVideoFrame:dstFrame:

onProcessVideoFrame:dstFrame:

- (uint32_t)onProcessVideoFrame:	(TRTCVideoFrame * _Nonnull)srcFrame
dstFrame:	(TRTCVideoFrame * _Nonnull)dstFrame

Video processing by third-party beauty filters

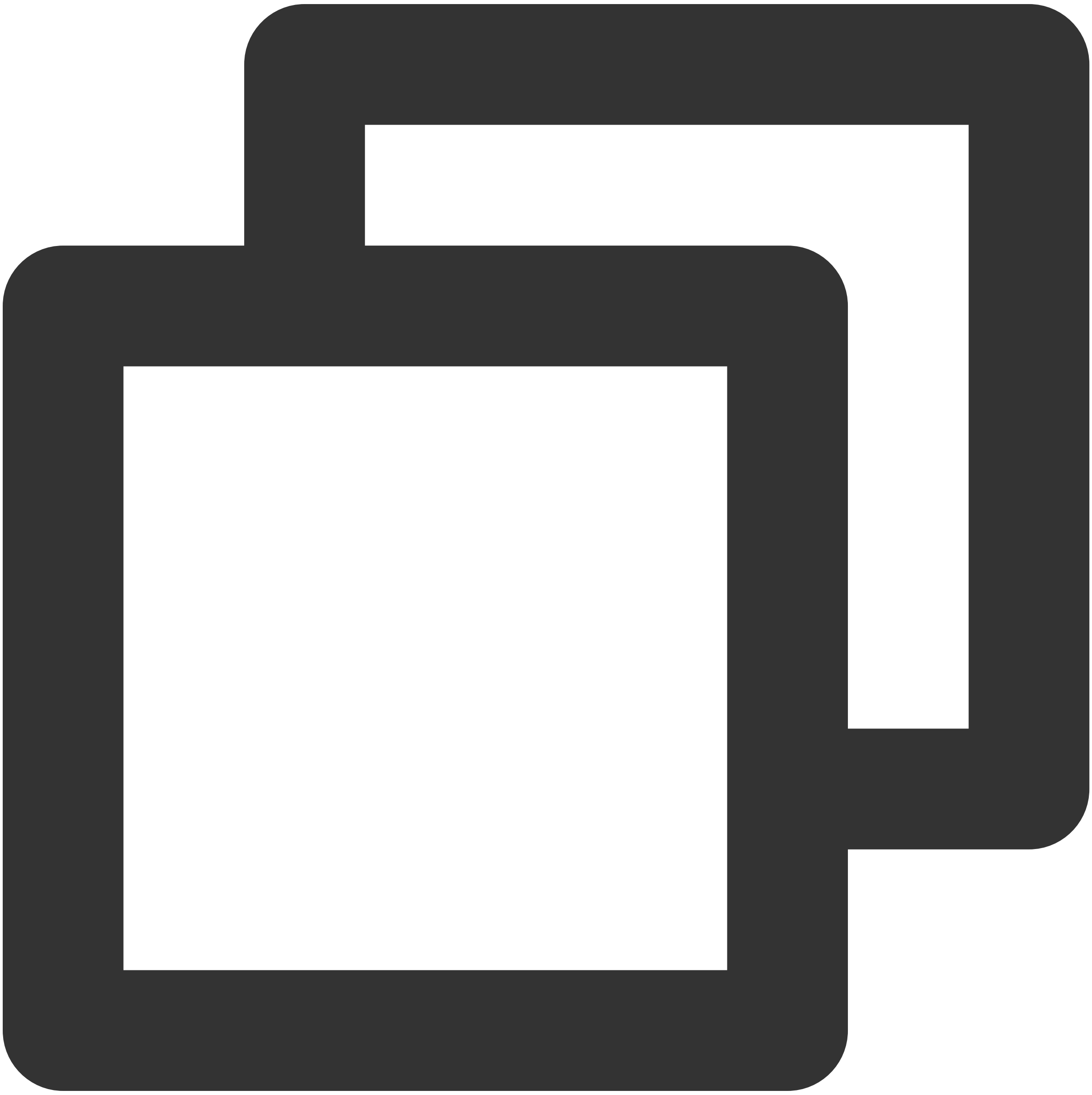
If you use a third-party beauty filter component, you need to configure this callback in `TRTCCloud` to have the SDK return to you video frames that are otherwise pre-processed by TRTC.
You can then send the video frames to the third-party beauty filter component for processing. As the data returned can be read and modified, the result of processing can be synced to TRTC for subsequent encoding and publishing.

Case 1: the beauty filter component generates new textures
If the beauty filter component you use generates a frame of new texture (for the processed image) during image processing, please set `dstFrame.textureId` to the ID of the new texture in the callback function.

```
uint32_t onProcessVideoFrame(TRTCVideoFrame * _Nonnull)srcFrame dstFrame:(TRTCVideo
    self.frameID += 1;
    dstFrame.pixelBuffer = [[FURenderer shareRenderer] renderPixelBuffer:srcFrame.p
                                                             withFrameId:self.frame
                                                                   items:self.rende
                                                               itemCount:self.rende
    return 0;
}
```

Case 2: you need to provide target textures to the beauty filter component

If the third-party beauty filter component you use does not generate new textures and you need to manually set an input texture and an output texture for the component, you can consider the following scheme:



```
uint32_t onProcessVideoFrame(TRTCVideoFrame * _Nonnull)srcFrame dstFrame:(TRTCVideo
    thirdparty_process(srcFrame.textureId, srcFrame.width, srcFrame.height, dstFram
    return 0;
}
```

Param	DESC
dstFrame	Used to receive video images processed by third-party beauty filters

srcFrame	Used to carry images captured by TRTC via the camera

Note

Currently, only the OpenGL texture scheme is supported(PC supports TRTCVideoBufferType_Buffer format Only)

onGLContextDestory

onGLContextDestory

The OpenGL context in the SDK was destroyed

onCapturedAudioFrame:

onCapturedAudioFrame:

- (void) onCapturedAudioFrame:	(TRTCAudioFrame *)frame

Audio data captured by the local mic and pre-processed by the audio module

After you configure the callback of custom audio processing, the SDK will return via this callback the data captured and pre-processed (ANS, AEC, and AGC) in PCM format.

The audio returned is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function. The SDK processes an audio frame every 20 ms, so if your operation takes more than 20 ms, it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.
3. The audio data is returned via this callback after ANS, AEC and AGC, but it **does not include** pre-processing effects like background music, audio effects, or reverb, and therefore has a short delay.

onLocalProcessedAudioFrame:

onLocalProcessedAudioFrame:

- (void) onLocalProcessedAudioFrame:	(TRTCAudioFrame *)frame

Audio data captured by the local mic, pre-processed by the audio module, effect-processed and BGM-mixed

After you configure the callback of custom audio processing, the SDK will return via this callback the data captured, pre-processed (ANS, AEC, and AGC), effect-processed and BGM-mixed in PCM format, before it is submitted to the network module for encoding.
The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.
The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.
Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Instructions:
You could write data to the `TRTCAudioFrame.extraData` filed, in order to achieve the purpose of transmitting signaling.
Because the data block of the audio frame header cannot be too large, we recommend you limit the size of the signaling data to only a few bytes when using this API. If extra data more than 100 bytes, it won't be sent.
Other users in the room can receive the message through the `TRTCAudioFrame.extraData` in `onRemoteUserAudioFrame` callback in TRTCAudioFrameDelegate.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function. The SDK processes an audio frame every 20 ms, so if your operation takes more than 20 ms, it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.
3. Audio data is returned via this callback after ANS, AEC, AGC, effect-processing and BGM-mixing, and therefore the delay is longer than that with onCapturedAudioFrame.

onRemoteUserAudioFrame:userId:

onRemoteUserAudioFrame:userId:

- (void) onRemoteUserAudioFrame:	(TRTCAudioFrame *)frame
userId:	(NSString *)userId

Audio data of each remote user before audio mixing

After you configure the callback of custom audio processing, the SDK will return via this callback the raw audio data (PCM format) of each remote user before mixing.

The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format
userId	User ID

Note

The audio data returned via this callback can be read but not modified.

onMixedPlayAudioFrame:

onMixedPlayAudioFrame:

- (void) onMixedPlayAudioFrame:	(TRTCAudioFrame *)frame

Data mixed from each channel before being submitted to the system for playback

After you configure the callback of custom audio processing, the SDK will return to you via this callback the data (PCM format) mixed from each channel before it is submitted to the system for playback.

The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function. The SDK processes an audio frame every 20 ms, so if your operation takes more than 20 ms, it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.
3. The audio data returned via this callback is the audio data mixed from each channel before it is played. It does not include the in-ear monitoring data.

onMixedAllAudioFrame:

onMixedAllAudioFrame:

- (void) onMixedAllAudioFrame:	(TRTCAudioFrame *)frame

Data mixed from all the captured and to-be-played audio in the SDK

After you configure the callback of custom audio processing, the SDK will return via this callback the data (PCM format) mixed from all captured and to-be-played audio in the SDK, so that you can customize recording.

The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. This data returned via this callback is mixed from all audio in the SDK, including local audio after pre-processing (ANS, AEC, and AGC), special effects application, and music mixing, as well as all remote audio, but it does not

include the in-ear monitoring data.
2. The audio data returned via this callback cannot be modified.

onVoiceEarMonitorAudioFrame:

onVoiceEarMonitorAudioFrame:

- (void) onVoiceEarMonitorAudioFrame:	(TRTCAudioFrame *)frame

In-ear monitoring data

After you configure the callback of custom audio processing, the SDK will return to you via this callback the in-ear monitoring data (PCM format) before it is submitted to the system for playback.
The audio returned is in PCM format and has a not-fixed frame length (time).
The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.
Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The length of 0.02s frame in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function, or it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.

onLog:LogLevel:WhichModule:

onLog:LogLevel:WhichModule:

-(void) onLog:	(nullable NSString*)log
LogLevel:	(TRTCLogLevel)level
WhichModule:	(nullable NSString*)module

Printing of local log

If you want to capture the local log printing event, you can configure the log callback to have the SDK return to you via this callback all logs that are to be printed.

Param	DESC
level	Log level. For more information, please see `TRTC_LOG_LEVEL` .
log	Log content
module	Reserved field, which is not defined at the moment and has a fixed value of `TXLiteAVSDK` .

TRTCStatistics

Last updated：2024-06-06 15:26:14

Module: TRTC audio/video metrics (read-only)

Function: the TRTC SDK reports to you the current real-time audio/video metrics (frame rate, bitrate, lag, etc.) once every two seconds

TRTCStatistics

StructType

FuncList	DESC
TRTCLocalStatistics	Local audio/video metrics
TRTCRemoteStatistics	Remote audio/video metrics
TRTCStatistics	Network and performance metrics

TRTCLocalStatistics

TRTCLocalStatistics

Local audio/video metrics

EnumType	DESC
audioBitrate	Field description: local audio bitrate in Kbps, i.e., how much audio data is generated per second
audioCaptureState	Field description:Audio equipment collection status(0：Normal；1：Long silence detected；2：Broken sound detected；3：Abnormal intermittent sound detected;)
audioSampleRate	Field description: local audio sample rate (Hz)
frameRate	Field description: local video frame rate in fps, i.e., how many video frames there

	are per second
height	Field description: local video height in px
streamType	Field description: video stream type (HD big image | smooth small image | substream image)
videoBitrate	Field description: local video bitrate in Kbps, i.e., how much video data is generated per second
width	Field description: local video width in px

TRTCRemoteStatistics

TRTCRemoteStatistics

Remote audio/video metrics

EnumType	DESC
audioBitrate	Field description: local audio bitrate (Kbps)
audioBlockRate	Field description: audio playback lag rate (%) Audio playback lag rate (audioBlockRate) = cumulative audio playback lag duration (audioTotalBlockTime)/total audio playback duration
audioPacketLoss	Field description: total packet loss rate (%) of the audio stream `audioPacketLoss` represents the packet loss rate eventually calculated on the audience side after the audio/video stream goes through the complete transfer linkage of "anchor -> cloud -> audience". The smaller the `audioPacketLoss` , the better. The packet loss rate of 0 indicates that all data of the audio stream has entirely reached the audience. If `downLoss` is `0` but `audioPacketLoss` isn't, there is no packet loss on the linkage of "cloud -> audience" for the audiostream, but there are unrecoverable packet losses on the linkage of "anchor -> cloud".
audioSampleRate	Field description: local audio sample rate (Hz)
audioTotalBlockTime	Field description: cumulative audio playback lag duration (ms)
finalLoss	Field description: total packet loss rate (%) of the audio/video stream Deprecated, please use audioPacketLoss and videoPacketLoss instead.
frameRate	Field description: remote video frame rate (fps)

height	Field description: remote video height in px
jitterBufferDelay	Field description: playback delay (ms) In order to avoid audio/video lags caused by network jitters and network packet disorders, TRTC maintains a playback buffer on the playback side to organize the received network data packets. The size of the buffer is adaptively adjusted according to the current network quality and converted to the length of time in milliseconds, i.e., `jitterBufferDelay` .
point2PointDelay	Field description: end-to-end delay (ms) `point2PointDelay` represents the delay of "anchor -> cloud -> audience". To be more precise, it represents the delay of the entire linkage of "collection -> encoding -> network transfer -> receiving -> buffering -> decoding -> playback". `point2PointDelay` works only if both the local and remote SDKs are on version 8.5 or above. If the remote SDK is on a version below 8.5, this value will always be 0 and thus meaningless.
remoteNetworkRTT	Field description: round-trip delay (ms) from the SDK to cloud This value represents the total time it takes to send a network packet from the SDK to the cloud and then send a network packet back from the cloud to the SDK, i.e., the total time it takes for a network packet to go through the linkage of "SDK -> cloud -> SDK". The smaller the value, the better. If `remoteNetworkRTT` is below 50 ms, it means a short audio/video call delay; if `remoteNetworkRTT` is above 200 ms, it means a long audio/video call delay. It should be explained that `remoteNetworkRTT` represents the total time spent on the linkage of "SDK -> cloud -> SDK"; therefore, there is no need to distinguish between `remoteNetworkUpRTT` and `remoteNetworkDownRTT` .
remoteNetworkUplinkLoss	Field description: upstream packet loss rate (%) from the SDK to cloud The smaller the value, the better. If `remoteNetworkUplinkLoss` is `0%` , the upstream network quality is very good, and the data packets uploaded to the cloud are basically not lost. If `remoteNetworkUplinkLoss` is `30%` , 30% of the audio/video data packets sent to the cloud by the SDK are lost on the transfer linkage.
streamType	Field description: video stream type (HD big image | smooth small image | substream image)
userId	Field description: user ID

videoBitrate	Field description: remote video bitrate (Kbps)
videoBlockRate	Field description: video playback lag rate (%) Video playback lag rate (videoBlockRate) = cumulative video playback lag duration (videoTotalBlockTime)/total video playback duration
videoPacketLoss	Field description: total packet loss rate (%) of the video stream `videoPacketLoss` represents the packet loss rate eventually calculated on the audience side after the audio/video stream goes through the complete transfer linkage of "anchor -> cloud -> audience". The smaller the `videoPacketLoss` , the better. The packet loss rate of 0 indicates that all data of the video stream has entirely reached the audience. If `downLoss` is `0` but `videoPacketLoss` isn't, there is no packet loss on the linkage of "cloud -> audience" for the video stream, but there are unrecoverable packet losses on the linkage of "anchor -> cloud".
videoTotalBlockTime	Field description: cumulative video playback lag duration (ms)
width	Field description: remote video width in px

TRTCStatistics

TRTCStatistics

Network and performance metrics

EnumType	DESC
appCpu	Field description: CPU utilization (%) of the current application, Android 8.0 and above systems are not supported
downLoss	Field description: downstream packet loss rate (%) from cloud to the SDK The smaller the value, the better. If `downLoss` is `0%` , the downstream network quality is very good, and the data packets received from the cloud are basically not lost. If `downLoss` is `30%` , 30% of the audio/video data packets sent to the SDK by the cloud are lost on the transfer linkage.
gatewayRtt	Field description: round-trip delay (ms) from the SDK to gateway This value represents the total time it takes to send a network packet from the SDK to the gateway and then send a network packet back from the gateway to the SDK, i.e., the total time it takes for a network packet to go through the linkage of "SDK -> gateway -> SDK".

	The smaller the value, the better. If `gatewayRtt` is below 50 ms, it means a short audio/video call delay; if `gatewayRtt` is above 200 ms, it means a long audio/video call delay. It should be explained that `gatewayRtt` is invalid for cellular network.
localStatistics	Field description: local audio/video statistics As there may be three local audio/video streams (i.e., HD big image, smooth small image, and substream image), the local audio/video statistics are an array.
receivedBytes	Field description: total number of received bytes (including signaling data and audio/video data)
remoteStatistics	Field description: remote audio/video statistics As there may be multiple concurrent remote users, and each of them may have multiple concurrent audio/video streams (i.e., HD big image, smooth small image, and substream image), the remote audio/video statistics are an array.
rtt	Field description: round-trip delay (ms) from the SDK to cloud This value represents the total time it takes to send a network packet from the SDK to the cloud and then send a network packet back from the cloud to the SDK, i.e., the total time it takes for a network packet to go through the linkage of "SDK -> cloud -> SDK". The smaller the value, the better. If `rtt` is below 50 ms, it means a short audio/video call delay; if `rtt` is above 200 ms, it means a long audio/video call delay. It should be explained that `rtt` represents the total time spent on the linkage of "SDK -> cloud -> SDK"; therefore, there is no need to distinguish between `upRtt` and `downRtt`.
sentBytes	Field description: total number of sent bytes (including signaling data and audio/video data)
systemCpu	Field description: CPU utilization (%) of the current system, Android 8.0 and above systems are not supported
upLoss	Field description: upstream packet loss rate (%) from the SDK to cloud The smaller the value, the better. If `upLoss` is `0%`, the upstream network quality is very good, and the data packets uploaded to the cloud are basically not lost. If `upLoss` is `30%`, 30% of the audio/video data packets sent to the cloud by the SDK are lost on the transfer linkage.

TXAudioEffectManager

Last updated：2024-06-06 15:26:14

Module: management class for background music, short audio effects, and voice effects

Description: sets background music, short audio effects, and voice effects

TXAudioEffectManager

TXAudioEffectManager

FuncList	DESC
enableVoiceEarMonitor:	Enabling in-ear monitoring
setVoiceEarMonitorVolume:	Setting in-ear monitoring volume
setVoiceReverbType:	Setting voice reverb effects
setVoiceChangerType:	Setting voice changing effects
setVoiceVolume:	Setting speech volume
setVoicePitch:	Setting speech pitch
startPlayMusic:onStart:onProgress:onComplete:	Starting background music
stopPlayMusic:	Stopping background music
pausePlayMusic:	Pausing background music
resumePlayMusic:	Resuming background music
setAllMusicVolume:	Setting the local and remote playback volume of background music
setMusicPublishVolume:volume:	Setting the remote playback volume of a specific music track
setMusicPlayoutVolume:volume:	Setting the local playback volume of a specific music track

setMusicPitch:pitch:	Adjusting the pitch of background music
setMusicSpeedRate:speedRate:	Changing the speed of background music
getMusicCurrentPosInMS:	Getting the playback progress (ms) of background music
getMusicDurationInMS:	Getting the total length (ms) of background music
seekMusicToPosInMS:pts:	Setting the playback progress (ms) of background music
setMusicScratchSpeedRate:speedRate:	Adjust the speed change effect of the scratch disc
preloadMusic:onProgress:onError:	Preload background music
getMusicTrackCount:	Get the number of tracks of background music
setMusicTrack:track:	Specify the playback track of background music

StructType

FuncList	DESC
TXAudioMusicParam	Background music playback information

EnumType

EnumType	DESC
TXVoiceReverbType	Reverb effects
TXVoiceChangeType	Voice changing effects

enableVoiceEarMonitor:

enableVoiceEarMonitor:

- (void)enableVoiceEarMonitor:	(BOOL)enable

Enabling in-ear monitoring

After enabling in-ear monitoring, anchors can hear in earphones their own voice captured by the mic. This is designed for singing scenarios.

In-ear monitoring cannot be enabled for Bluetooth earphones. This is because Bluetooth earphones have high latency. Please ask anchors to use wired earphones via a UI reminder.
Given that not all phones deliver excellent in-ear monitoring effects, we have blocked this feature on some phones.

Param	DESC
enable	`YES:` enable; `NO` : disable

Note
In-ear monitoring can be enabled only when earphones are used. Please remind anchors to use wired earphones.

setVoiceEarMonitorVolume:

setVoiceEarMonitorVolume:

- (void)setVoiceEarMonitorVolume:	(NSInteger)volume

Setting in-ear monitoring volume

This API is used to set the volume of in-ear monitoring.

Param	DESC
volume	Volume. Value range: 0-100; default: 100

Note
If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setVoiceReverbType:

setVoiceReverbType:

- (void)setVoiceReverbType:	(TXVoiceReverbType)reverbType

Setting voice reverb effects

This API is used to set reverb effects for human voice. For the effects supported, please see TXVoiceReverbType.

Note

Effects become invalid after room exit. If you want to use the same effect after you enter the room again, you need to set the effect again using this API.

setVoiceChangerType:

setVoiceChangerType:

- (void)setVoiceChangerType:	(TXVoiceChangeType)changerType

Setting voice changing effects

This API is used to set voice changing effects. For the effects supported, please see TXVoiceChangeType.

Note

Effects become invalid after room exit. If you want to use the same effect after you enter the room again, you need to set the effect again using this API.

setVoiceVolume:

setVoiceVolume:

- (void)setVoiceVolume:	(NSInteger)volume

Setting speech volume

This API is used to set the volume of speech. It is often used together with the music volume setting API setAllMusicVolume to balance between the volume of music and speech.

Param	DESC
volume	Volume. Value range: 0-100; default: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setVoicePitch:

setVoicePitch:

-(void)setVoicePitch:	(double)pitch

Setting speech pitch

This API is used to set the pitch of speech.

Param	DESC
pitch	Ptich, Value range: -1.0f~1.0f; default: 0.0f。

startPlayMusic:onStart:onProgress:onComplete:

startPlayMusic:onStart:onProgress:onComplete:

- (void)startPlayMusic:	(TXAudioMusicParam *)musicParam
onStart:	(TXAudioMusicStartBlock _Nullable)startBlock
onProgress:	(TXAudioMusicProgressBlock _Nullable)progressBlock
onComplete:	(TXAudioMusicCompleteBlock _Nullable)completeBlock

Starting background music

You must assign an ID to each music track so that you can start, stop, or set the volume of music tracks by ID.

Param	DESC
completeBlock	Callback of ending music
musicParam	Music parameter
progressBlock	Callback of playback progress
startBlock	Callback of starting music

Note

1. If you play the same music track multiple times, please use the same ID instead of a separate ID for each playback.
2. If you want to play different music tracks at the same time, use different IDs for them.
3. If you use the same ID to play a music track different from the current one, the SDK will stop the current one before playing the new one.

stopPlayMusic:

stopPlayMusic:

- (void)stopPlayMusic:	(int32_t)id

Stopping background music

Param	DESC
id	Music ID

pausePlayMusic:

pausePlayMusic:

- (void)pausePlayMusic:	(int32_t)id

Pausing background music

Param	DESC
id	Music ID

resumePlayMusic:

resumePlayMusic:

- (void)resumePlayMusic:	(int32_t)id

Resuming background music

Param	DESC
id	Music ID

setAllMusicVolume:

setAllMusicVolume:

- (void)setAllMusicVolume:	(NSInteger)volume

Setting the local and remote playback volume of background music

This API is used to set the local and remote playback volume of background music.
Local volume: the volume of music heard by anchors
Remote volume: the volume of music heard by audience

Param	DESC
volume	Volume. Value range: 0-100; default: 60

Note
If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setMusicPublishVolume:volume:

setMusicPublishVolume:volume:

- (void)setMusicPublishVolume:	(int32_t)id
volume:	(NSInteger)volume

Setting the remote playback volume of a specific music track

This API is used to control the remote playback volume (the volume heard by audience) of a specific music track.

Param	DESC
id	Music ID
volume	Volume. Value range: 0-100; default: 60

Note
If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setMusicPlayoutVolume:volume:

setMusicPlayoutVolume:volume:

- (void)setMusicPlayoutVolume:	(int32_t)id
volume:	(NSInteger)volume

Setting the local playback volume of a specific music track

This API is used to control the local playback volume (the volume heard by anchors) of a specific music track.

Param	DESC
id	Music ID
volume	Volume. Value range: 0-100. default: 60

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setMusicPitch:pitch:

setMusicPitch:pitch:

- (void)setMusicPitch:	(int32_t)id
pitch:	(double)pitch

Adjusting the pitch of background music

Param	DESC
id	Music ID
pitch	Pitch. Value range: floating point numbers in the range of [-1, 1]; default: 0.0f

setMusicSpeedRate:speedRate:

setMusicSpeedRate:speedRate:

- (void)setMusicSpeedRate:	(int32_t)id
speedRate:	(double)speedRate

Changing the speed of background music

Param	DESC
id	Music ID
speedRate	Music speed. Value range: floating point numbers in the range of [0.5, 2]; default: 1.0f

getMusicCurrentPosInMS:

getMusicCurrentPosInMS:

- (NSInteger)getMusicCurrentPosInMS:	(int32_t)id

Getting the playback progress (ms) of background music

Param	DESC
id	Music ID

Return Desc:

The milliseconds that have passed since playback started. -1 indicates failure to get the the playback progress.

getMusicDurationInMS:

getMusicDurationInMS:

- (NSInteger)getMusicDurationInMS:	(NSString *)path

Getting the total length (ms) of background music

Param	DESC
path	Path of the music file.

Return Desc:

The length of the specified music file is returned. -1 indicates failure to get the length.

seekMusicToPosInMS:pts:

seekMusicToPosInMS:pts:

- (void)seekMusicToPosInMS:	(int32_t)id
pts:	(NSInteger)pts

Setting the playback progress (ms) of background music

Param	DESC
id	Music ID
pts	Unit: millisecond

Note

Do not call this API frequently as the music file may be read and written to each time the API is called, which can be time-consuming.

Wait till users finish dragging the progress bar before you call this API.

The progress bar controller on the UI tends to update the progress at a high frequency as users drag the progress bar. This will result in poor user experience unless you limit the frequency.

setMusicScratchSpeedRate:speedRate:

setMusicScratchSpeedRate:speedRate:

- (void)setMusicScratchSpeedRate:	(int32_t)id
speedRate:	(double)scratchSpeedRate

Adjust the speed change effect of the scratch disc

Param	DESC
id	Music ID
scratchSpeedRate	Scratch disc speed, the default value is 1.0f, the range is: a floating point number between [-12.0 ~ 12.0], the positive/negative speed value indicates the direction is positive/negative, and the absolute value indicates the speed.

Note

Precondition preloadMusic succeeds.

preloadMusic:onProgress:onError:

preloadMusic:onProgress:onError:

- (void)preloadMusic:	(TXAudioMusicParam *)preloadParam
onProgress:	(TXMusicPreloadProgressBlock _Nullable)progressBlock

onError:	(TXMusicPreloadErrorBlock _Nullable)errorBlock

Preload background music

You must assign an ID to each music track so that you can start, stop, or set the volume of music tracks by ID.

Param	DESC
musicParam	Music parameter

Note

1. Preload supports up to 2 preloads with different IDs at the same time, and the preload time does not exceed 10 minutes,you need to stopPlayMusic after use, otherwise the memory will not be released.
2. If the music corresponding to the ID is being played, the preloading fails, and stopPlayMusic must be called first.
3. When the musicParam passed to startPlayMusic is exactly the same, preloading works.

getMusicTrackCount:

getMusicTrackCount:

- (NSInteger)getMusicTrackCount:	(int32_t)id

Get the number of tracks of background music

Param	DESC
id	Music ID

setMusicTrack:track:

setMusicTrack:track:

- (void)setMusicTrack:	(int32_t)id
track:	(NSInteger)track

Specify the playback track of background music

Param	DESC
id	Music ID

index	Specify which track to play (the first track is played by default). Value range [0, total number of tracks).

Note

The total number of tracks can be obtained through the getMusicTrackCount interface.

TXVoiceReverbType

TXVoiceReverbType

Reverb effects

Reverb effects can be applied to human voice. Based on acoustic algorithms, they can mimic voice in different environments. The following effects are supported currently:
0: original; 1: karaoke; 2: room; 3: hall; 4: low and deep; 5: resonant; 6: metal; 7: husky; 8: ethereal; 9: studio; 10: melodious; 11: studio2;

Enum	Value	DESC
TXVoiceReverbType_0	0	disable
TXVoiceReverbType_1	1	KTV
TXVoiceReverbType_2	2	small room
TXVoiceReverbType_3	3	great hall
TXVoiceReverbType_4	4	deep voice
TXVoiceReverbType_5	5	loud voice
TXVoiceReverbType_6	6	metallic sound
TXVoiceReverbType_7	7	magnetic sound
TXVoiceReverbType_8	8	ethereal
TXVoiceReverbType_9	9	studio
TXVoiceReverbType_10	10	melodious
TXVoiceReverbType_11	11	studio2

TXVoiceChangeType

TXVoiceChangeType

Voice changing effects

Voice changing effects can be applied to human voice. Based on acoustic algorithms, they change the tone of voice. The following effects are supported currently:
0: original; 1: child; 2: little girl; 3: middle-aged man; 4: metal; 5: nasal; 6: foreign accent; 7: trapped beast; 8: otaku; 9: electric; 10: robot; 11: ethereal

Enum	Value	DESC
TXVoiceChangeType_0	0	disable
TXVoiceChangeType_1	1	naughty kid
TXVoiceChangeType_2	2	Lolita
TXVoiceChangeType_3	3	uncle
TXVoiceChangeType_4	4	heavy metal
TXVoiceChangeType_5	5	catch cold
TXVoiceChangeType_6	6	foreign accent
TXVoiceChangeType_7	7	caged animal trapped beast
TXVoiceChangeType_8	8	indoorsman
TXVoiceChangeType_9	9	strong current
TXVoiceChangeType_10	10	heavy machinery
TXVoiceChangeType_11	11	intangible

TXAudioMusicParam

TXAudioMusicParam

Background music playback information

The information, including playback ID, file path, and loop times, is passed in the startPlayMusic API.
1. If you play the same music track multiple times, please use the same ID instead of a separate ID for each playback.
2. If you want to play different music tracks at the same time, use different IDs for them.
3. If you use the same ID to play a music track different from the current one, the SDK will stop the current one before playing the new one.

EnumType	DESC
ID	`Field description:` music ID **Note** the SDK supports playing multiple music tracks. IDs are used to distinguish different music tracks and control their start, end, volume, etc.
endTimeMS	`Field description:` the point in time in milliseconds for ending music playback. 0 indicates that playback continues till the end of the music track.
isShortFile	`Field description:` whether the music played is a short music track `Valid values:` `YES` : short music track that needs to be looped; `NO` (default): normal-length music track
loopCount	`Field description:` number of times the music track is looped `Valid values:` 0 or any positive integer. 0 (default) indicates that the music is played once, 1 twice, and so on.
path	`Field description:` absolute path of the music file or url.the mp3,aac,m4a,wav supported.
publish	`Field description:` whether to send the music to remote users `Valid values:` `YES` : remote users can hear the music played locally; `NO` (default): only the local user can hear the music.
startTimeMS	`Field description:` the point in time in milliseconds for starting music playback

TXBeautyManager

Last updated：2024-06-06 15:26:14

Module: beauty filter and image processing parameter configurations

Function: you can modify parameters such as beautification, filter, and green screen

TXBeautyManager

TXBeautyManager

FuncList	DESC
setBeautyStyle:	Sets the beauty (skin smoothing) filter algorithm.
setBeautyLevel:	Sets the strength of the beauty filter.
setWhitenessLevel:	Sets the strength of the brightening filter.
enableSharpnessEnhancement:	Enables clarity enhancement.
setRuddyLevel:	Sets the strength of the rosy skin filter.
setFilter:	Sets color filter.
setFilterStrength:	Sets the strength of color filter.
setGreenScreenFile:	Sets green screen video
setEyeScaleLevel:	Sets the strength of the eye enlarging filter.
setFaceSlimLevel:	Sets the strength of the face slimming filter.
setFaceVLevel:	Sets the strength of the chin slimming filter.
setChinLevel:	Sets the strength of the chin lengthening/shortening filter.
setFaceShortLevel:	Sets the strength of the face shortening filter.
setFaceNarrowLevel:	Sets the strength of the face narrowing filter.

setNoseSlimLevel:	Sets the strength of the nose slimming filter.
setEyeLightenLevel:	Sets the strength of the eye brightening filter.
setToothWhitenLevel:	Sets the strength of the teeth whitening filter.
setWrinkleRemoveLevel:	Sets the strength of the wrinkle removal filter.
setPounchRemoveLevel:	Sets the strength of the eye bag removal filter.
setSmileLinesRemoveLevel:	Sets the strength of the smile line removal filter.
setForeheadLevel:	Sets the strength of the hairline adjustment filter.
setEyeDistanceLevel:	Sets the strength of the eye distance adjustment filter.
setEyeAngleLevel:	Sets the strength of the eye corner adjustment filter.
setMouthShapeLevel:	Sets the strength of the mouth shape adjustment filter.
setNoseWingLevel:	Sets the strength of the nose wing narrowing filter.
setNosePositionLevel:	Sets the strength of the nose position adjustment filter.
setLipsThicknessLevel:	Sets the strength of the lip thickness adjustment filter.
setFaceBeautyLevel:	Sets the strength of the face shape adjustment filter.
setMotionTmpl:inDir:	Selects the AI animated effect pendant.
setMotionMute:	Sets whether to mute during animated effect playback.

EnumType

EnumType	DESC
TXBeautyStyle	Beauty (skin smoothing) filter algorithm

setBeautyStyle:

setBeautyStyle:

- (void)setBeautyStyle:	(TXBeautyStyle)beautyStyle

Sets the beauty (skin smoothing) filter algorithm.

TRTC has multiple built-in skin smoothing algorithms. You can select the one most suitable for your product needs:

Param	DESC
beautyStyle	Beauty filter style. `TXBeautyStyleSmooth` : smooth; `TXBeautyStyleNature` : natural; `TXBeautyStylePitu` : Pitu

setBeautyLevel:

setBeautyLevel:

- (void)setBeautyLevel:	(float)beautyLevel

Sets the strength of the beauty filter.

Param	DESC
beautyLevel	Strength of the beauty filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

setWhitenessLevel:

setWhitenessLevel:

- (void)setWhitenessLevel:	(float)whitenessLevel

Sets the strength of the brightening filter.

Param	DESC
whitenessLevel	Strength of the brightening filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

enableSharpnessEnhancement:

enableSharpnessEnhancement:

- (void)enableSharpnessEnhancement:	(BOOL)enable

Enables clarity enhancement.

setRuddyLevel:

setRuddyLevel:

- (void)setRuddyLevel:	(float)ruddyLevel

Sets the strength of the rosy skin filter.

Param	DESC
ruddyLevel	Strength of the rosy skin filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

setFilter:

setFilter:

- (void)setFilter:	(nullable TXImage *)image

Sets color filter.

The color filter is a color lookup table image containing color mapping relationships. You can find several predefined filter images in the official demo we provide.

The SDK performs secondary processing on the original video image captured by the camera according to the mapping relationships in the lookup table to achieve the expected filter effect.

Param	DESC
image	Color lookup table containing color mapping relationships. The image must be in PNG format.

setFilterStrength:

setFilterStrength:

- (void)setFilterStrength:	(float)strength

Sets the strength of color filter.

The larger this value, the more obvious the effect of the color filter, and the greater the color difference between the video image processed by the filter and the original video image.
The default strength is 0.5, and if it is not sufficient, it can be adjusted to a value above 0.5. The maximum value is 1.

Param	DESC
strength	Value range: 0–1. The greater the value, the more obvious the effect. Default value: 0.5

setGreenScreenFile:

setGreenScreenFile:

- (int)setGreenScreenFile:	(nullable NSString *)path

Sets green screen video

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.
The green screen feature enabled by this API is not capable of intelligent keying. It requires that there be a green screen behind the videoed person or object for further chroma keying.

Param	DESC
path	Path of the video file in MP4 format. An empty value indicates to disable the effect.

Return Desc:

0: Success; -5: feature of license not supported.

setEyeScaleLevel:

setEyeScaleLevel:

- (int)setEyeScaleLevel:	(float)eyeScaleLevel

Sets the strength of the eye enlarging filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
eyeScaleLevel	Strength of the eye enlarging filter. Value range: 0–9. `0` indicates to disable the

	filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceSlimLevel:

setFaceSlimLevel:

- (int)setFaceSlimLevel:	(float)faceSlimLevel

Sets the strength of the face slimming filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
faceSlimLevel	Strength of the face slimming filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceVLevel:

setFaceVLevel:

- (int)setFaceVLevel:	(float)faceVLevel

Sets the strength of the chin slimming filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
faceVLevel	Strength of the chin slimming filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setChinLevel:

setChinLevel:

- (int)setChinLevel:	(float)chinLevel

Sets the strength of the chin lengthening/shortening filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
chinLevel	Strength of the chin lengthening/shortening filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates that the chin is shortened, and a value greater than 0 indicates that the chin is lengthened.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceShortLevel:

setFaceShortLevel:

- (int)setFaceShortLevel:	(float)faceShortLevel

Sets the strength of the face shortening filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
faceShortLevel	Strength of the face shortening filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceNarrowLevel:

setFaceNarrowLevel:

- (int)setFaceNarrowLevel:	(float)faceNarrowLevel

Sets the strength of the face narrowing filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
level	Strength of the face narrowing filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setNoseSlimLevel:

setNoseSlimLevel:

- (int)setNoseSlimLevel:	(float)noseSlimLevel

Sets the strength of the nose slimming filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
noseSlimLevel	Strength of the nose slimming filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setEyeLightenLevel:

setEyeLightenLevel:

- (int)setEyeLightenLevel:	(float)eyeLightenLevel

Sets the strength of the eye brightening filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
eyeLightenLevel	Strength of the eye brightening filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setToothWhitenLevel:

setToothWhitenLevel:

- (int)setToothWhitenLevel:	(float)toothWhitenLevel

Sets the strength of the teeth whitening filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
toothWhitenLevel	Strength of the teeth whitening filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setWrinkleRemoveLevel:

setWrinkleRemoveLevel:

- (int)setWrinkleRemoveLevel:	(float)wrinkleRemoveLevel

Sets the strength of the wrinkle removal filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
wrinkleRemoveLevel	Strength of the wrinkle removal filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setPounchRemoveLevel:

setPounchRemoveLevel:

- (int)setPounchRemoveLevel:	(float)pounchRemoveLevel

Sets the strength of the eye bag removal filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
pounchRemoveLevel	Strength of the eye bag removal filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setSmileLinesRemoveLevel:

setSmileLinesRemoveLevel:

- (int)setSmileLinesRemoveLevel:	(float)smileLinesRemoveLevel

Sets the strength of the smile line removal filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
smileLinesRemoveLevel	Strength of the smile line removal filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setForeheadLevel:

setForeheadLevel:

- (int)setForeheadLevel:	(float)foreheadLevel

Sets the strength of the hairline adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
foreheadLevel	Strength of the hairline adjustment filter. Value range: -9–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setEyeDistanceLevel:

setEyeDistanceLevel:

- (int)setEyeDistanceLevel:	(float)eyeDistanceLevel

Sets the strength of the eye distance adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC

eyeDistanceLevel	Strength of the eye distance adjustment filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates to widen, and a value greater than 0 indicates to narrow.

Return Desc:

0: Success; -5: feature of license not supported.

setEyeAngleLevel:

setEyeAngleLevel:

- (int)setEyeAngleLevel:	(float)eyeAngleLevel

Sets the strength of the eye corner adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
eyeAngleLevel	Strength of the eye corner adjustment filter. Value range: -9–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setMouthShapeLevel:

setMouthShapeLevel:

- (int)setMouthShapeLevel:	(float)mouthShapeLevel

Sets the strength of the mouth shape adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
mouthShapeLevel	Strength of the mouth shape adjustment filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates to widen, and a value greater

	than 0 indicates to narrow.

Return Desc:

0: Success; -5: feature of license not supported.

setNoseWingLevel:

setNoseWingLevel:

- (int)setNoseWingLevel:	(float)noseWingLevel

Sets the strength of the nose wing narrowing filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
noseWingLevel	Strength of the nose wing adjustment filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates to widen, and a value greater than 0 indicates to narrow.

Return Desc:

0: Success; -5: feature of license not supported.

setNosePositionLevel:

setNosePositionLevel:

- (int)setNosePositionLevel:	(float)nosePositionLevel

Sets the strength of the nose position adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
nosePositionLevel	Strength of the nose position adjustment filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates to lift, and a value greater than 0 indicates to lower.

Return Desc:

0: Success; -5: feature of license not supported.

setLipsThicknessLevel:

setLipsThicknessLevel:

- (int)setLipsThicknessLevel:	(float)lipsThicknessLevel

Sets the strength of the lip thickness adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
lipsThicknessLevel	Strength of the lip thickness adjustment filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates to thicken, and a value greater than 0 indicates to thin.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceBeautyLevel:

setFaceBeautyLevel:

- (int)setFaceBeautyLevel:	(float)faceBeautyLevel

Sets the strength of the face shape adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
faceBeautyLevel	Strength of the face shape adjustment filter. Value range: 0–9. `0` indicates to disable the filter, and the greater the value, the more obvious the effect.

Return Desc:

0: Success; -5: feature of license not supported.

setMotionTmpl:inDir:

setMotionTmpl:inDir:

- (void)setMotionTmpl:	(nullable NSString *)tmplName
inDir:	(nullable NSString *)tmplDir

Selects the AI animated effect pendant.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
tmplDir	Directory of the animated effect material file
tmplName	Animated effect pendant name

setMotionMute:

setMotionMute:

- (void)setMotionMute:	(BOOL)motionMute

Sets whether to mute during animated effect playback.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect. Some animated effects have audio effects, which can be disabled through this API when they are played back.

Param	DESC
motionMute	`YES` : mute; `NO` : unmute

TXBeautyStyle

TXBeautyStyle

Beauty (skin smoothing) filter algorithm

TRTC has multiple built-in skin smoothing algorithms. You can select the one most suitable for your product needs.

Enum	Value	DESC

TXBeautyStyleSmooth	0	Smooth style, which uses a more radical algorithm for more obvious effect and is suitable for show live streaming.
TXBeautyStyleNature	1	Natural style, which retains more facial details for more natural effect and is suitable for most live streaming use cases.
TXBeautyStylePitu	2	Pitu style, which is provided by YouTu Lab. Its skin smoothing effect is between the smooth style and the natural style, that is, it retains more skin details than the smooth style and has a higher skin smoothing degree than the natural style.

TXDeviceManager

Last updated：2024-06-06 15:26:14

Module: audio/video device management module

Description: manages audio/video devices such as camera, mic, and speaker.

TXDeviceManager

TXDeviceObserver

FuncList	DESC
onDeviceChanged:type:state:	The status of a local device changed (for desktop OS only)

TXDeviceManager

FuncList	DESC
isFrontCamera	Querying whether the front camera is being used
switchCamera:	Switching to the front/rear camera (for mobile OS)
isCameraZoomSupported	Querying whether the current camera supports zooming (for mobile OS)
getCameraZoomMaxRatio	Getting the maximum zoom ratio of the camera (for mobile OS)
setCameraZoomRatio:	Setting the camera zoom ratio (for mobile OS)
isAutoFocusEnabled	Querying whether automatic face detection is supported (for mobile OS)
enableCameraAutoFocus:	Enabling auto focus (for mobile OS)
setCameraFocusPosition:	Adjusting the focus (for mobile OS)

isCameraTorchSupported	Querying whether flash is supported (for mobile OS)
enableCameraTorch:	Enabling/Disabling flash, i.e., the torch mode (for mobile OS)
setAudioRoute:	Setting the audio route (for mobile OS)
setExposureCompensation:	Set the exposure parameters of the camera, ranging from - 1 to 1
getDevicesList:	Getting the device list (for desktop OS)
setCurrentDevice:deviceId:	Setting the device to use (for desktop OS)
getCurrentDevice:	Getting the device currently in use (for desktop OS)
setCurrentDeviceVolume:deviceType:	Setting the volume of the current device (for desktop OS)
getCurrentDeviceVolume:	Getting the volume of the current device (for desktop OS)
setCurrentDeviceMute:deviceType:	Muting the current device (for desktop OS)
getCurrentDeviceMute:	Querying whether the current device is muted (for desktop OS)
enableFollowingDefaultAudioDevice:enable:	Set the audio device used by SDK to follow the system default device (for desktop OS)
startCameraDeviceTest:	Starting camera testing (for desktop OS)
stopCameraDeviceTest	Ending camera testing (for desktop OS)
startMicDeviceTest:	Starting mic testing (for desktop OS)
startMicDeviceTest:playback:	Starting mic testing (for desktop OS)
stopMicDeviceTest	Ending mic testing (for desktop OS)
startSpeakerDeviceTest:	Starting speaker testing (for desktop OS)
stopSpeakerDeviceTest	Ending speaker testing (for desktop OS)
setObserver:	set onDeviceChanged callback (for Mac)
setCameraCapturerParam:	Set camera acquisition preferences
setSystemVolumeType:	Setting the system volume type (for mobile OS)

StructType

FuncList	DESC
TXCameraCaptureParam	Camera acquisition parameters
TXMediaDeviceInfo	Audio/Video device information (for desktop OS)

EnumType

EnumType	DESC
TXSystemVolumeType	System volume type
TXAudioRoute	Audio route (the route via which audio is played)
TXMediaDeviceType	Device type (for desktop OS)
TXMediaDeviceState	Device operation
TXCameraCaptureMode	Camera acquisition preferences

onDeviceChanged:type:state:

onDeviceChanged:type:state:

- (void)onDeviceChanged:	(NSString*)deviceId
type:	(TXMediaDeviceType)mediaType
state:	(TXMediaDeviceState)mediaState

The status of a local device changed (for desktop OS only)

The SDK returns this callback when a local device (camera, mic, or speaker) is connected or disconnected.

Param	DESC
deviceId	Device ID
state	Device status. `0` : connected; `1` : disconnected; `2` : started
type	Device type

isFrontCamera

isFrontCamera

Querying whether the front camera is being used

switchCamera:

switchCamera:

- (NSInteger)switchCamera:	(BOOL)frontCamera

Switching to the front/rear camera (for mobile OS)

isCameraZoomSupported

isCameraZoomSupported

Querying whether the current camera supports zooming (for mobile OS)

getCameraZoomMaxRatio

getCameraZoomMaxRatio

Getting the maximum zoom ratio of the camera (for mobile OS)

setCameraZoomRatio:

setCameraZoomRatio:

- (NSInteger)setCameraZoomRatio:	(CGFloat)zoomRatio

Setting the camera zoom ratio (for mobile OS)

Param	DESC
zoomRatio	Value range: 1-5. 1 indicates the widest angle of view (original), and 5 the narrowest angle of view (zoomed in).The maximum value is recommended to be 5. If the value exceeds 5, the video will become blurred.

isAutoFocusEnabled

isAutoFocusEnabled

Querying whether automatic face detection is supported (for mobile OS)

enableCameraAutoFocus:

enableCameraAutoFocus:

- (NSInteger)enableCameraAutoFocus:	(BOOL)enabled

Enabling auto focus (for mobile OS)

After auto focus is enabled, the camera will automatically detect and always focus on faces.

setCameraFocusPosition:

setCameraFocusPosition:

- (NSInteger)setCameraFocusPosition:	(CGPoint)position

Adjusting the focus (for mobile OS)

This API can be used to achieve the following:

1. A user can tap on the camera preview.
2. A rectangle will appear where the user taps, indicating the spot the camera will focus on.
3. The user passes the coordinates of the spot to the SDK using this API, and the SDK will instruct the camera to focus as required.

Param	DESC
position	The spot to focus on. Pass in the coordinates of the spot you want to focus on.

Note

Before using this API, you must first disable auto focus using enableCameraAutoFocus.

Return Desc:

0: operation successful; negative number: operation failed.

isCameraTorchSupported

isCameraTorchSupported

Querying whether flash is supported (for mobile OS)

enableCameraTorch:

enableCameraTorch:

- (NSInteger)enableCameraTorch:	(BOOL)enabled

Enabling/Disabling flash, i.e., the torch mode (for mobile OS)

setAudioRoute:

setAudioRoute:

- (NSInteger)setAudioRoute:	(TXAudioRoute)route

Setting the audio route (for mobile OS)

A mobile phone has two audio playback devices: the receiver at the top and the speaker at the bottom.
If the audio route is set to the receiver, the volume is relatively low, and audio can be heard only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.
If the audio route is set to the speaker, the volume is relatively high, and there is no need to put the phone near the ear. This mode enables the "hands-free" feature.

setExposureCompensation:

setExposureCompensation:

- (NSInteger)setExposureCompensation:	(CGFloat)value

Set the exposure parameters of the camera, ranging from - 1 to 1

getDevicesList:

getDevicesList:

- (NSArray<TXMediaDeviceInfo *> * _Nullable)getDevicesList:	(TXMediaDeviceType)type

Getting the device list (for desktop OS)

Param	DESC
type	Device type. Set it to the type of device you want to get. For details, please see the definition of `TXMediaDeviceType` .

Note

To ensure that the SDK can manage the lifecycle of the `ITXDeviceCollection` object, after using this API, please call the `release` method to release the resources.

Do not use `delete` to release the Collection object returned as deleting the ITXDeviceCollection* pointer will cause crash.

The valid values of `type` are `TXMediaDeviceTypeMic` , `TXMediaDeviceTypeSpeaker` , and `TXMediaDeviceTypeCamera` .

This API can be used only on macOS and Windows.

setCurrentDevice:deviceId:

setCurrentDevice:deviceId:

- (NSInteger)setCurrentDevice:	(TXMediaDeviceType)type
deviceId:	(NSString *)deviceId

Setting the device to use (for desktop OS)

Param	DESC
deviceId	Device ID. You can get the ID of a device using the getDevicesList API.
type	Device type. For details, please see the definition of `TXMediaDeviceType` .

Return Desc:

0: operation successful; negative number: operation failed.

getCurrentDevice:

getCurrentDevice:

- (TXMediaDeviceInfo * _Nullable)getCurrentDevice:	(TXMediaDeviceType)type

Getting the device currently in use (for desktop OS)

setCurrentDeviceVolume:deviceType:

setCurrentDeviceVolume:deviceType:

- (NSInteger)setCurrentDeviceVolume:	(NSInteger)volume
deviceType:	(TXMediaDeviceType)type

Setting the volume of the current device (for desktop OS)

This API is used to set the capturing volume of the mic or playback volume of the speaker, but not the volume of the camera.

Param	DESC
volume	Volume. Value range: 0-100; default: 100

getCurrentDeviceVolume:

getCurrentDeviceVolume:

- (NSInteger)getCurrentDeviceVolume:	(TXMediaDeviceType)type

Getting the volume of the current device (for desktop OS)

This API is used to get the capturing volume of the mic or playback volume of the speaker, but not the volume of the camera.

setCurrentDeviceMute:deviceType:

setCurrentDeviceMute:deviceType:

- (NSInteger)setCurrentDeviceMute:	(BOOL)mute

deviceType:	(TXMediaDeviceType)type

Muting the current device (for desktop OS)

This API is used to mute the mic or speaker, but not the camera.

getCurrentDeviceMute:

getCurrentDeviceMute:

- (BOOL)getCurrentDeviceMute:	(TXMediaDeviceType)type

Querying whether the current device is muted (for desktop OS)

This API is used to query whether the mic or speaker is muted. Camera muting is not supported.

enableFollowingDefaultAudioDevice:enable:

enableFollowingDefaultAudioDevice:enable:

- (NSInteger)enableFollowingDefaultAudioDevice:	(TXMediaDeviceType)type
enable:	(BOOL)enable

Set the audio device used by SDK to follow the system default device (for desktop OS)

This API is used to set the microphone and speaker types. Camera following the system default device is not supported.

Param	DESC
enable	Whether to follow the system default audio device. true: following. When the default audio device of the system is changed or new audio device is plugged in, the SDK immediately switches the audio device. false：not following. When the default audio device of the system is changed or new audio device is plugged in, the SDK doesn't switch the audio device.
type	Device type. For details, please see the definition of `TXMediaDeviceType` .

startCameraDeviceTest:

startCameraDeviceTest:

- (NSInteger)startCameraDeviceTest:	(NSView *)view

Starting camera testing (for desktop OS)

Note

You can use the setCurrentDevice API to switch between cameras during testing.

stopCameraDeviceTest

stopCameraDeviceTest

Ending camera testing (for desktop OS)

startMicDeviceTest:

startMicDeviceTest:

- (NSInteger)startMicDeviceTest:	(NSInteger)interval

Starting mic testing (for desktop OS)

This API is used to test whether the mic functions properly. The mic volume detected (value range: 0-100) is returned via a callback.

Param	DESC
interval	Interval of volume callbacks

Note

When this interface is called, the sound recorded by the microphone will be played back to the speakers by default.

startMicDeviceTest:playback:

startMicDeviceTest:playback:

- (NSInteger)startMicDeviceTest:	(NSInteger)interval
playback:	(BOOL)playback

Starting mic testing (for desktop OS)

This API is used to test whether the mic functions properly. The mic volume detected (value range: 0-100) is returned via a callback.

Param	DESC
interval	Interval of volume callbacks
playback	Whether to play back the microphone sound. The user will hear his own sound when testing the microphone if `playback` is true.

stopMicDeviceTest

stopMicDeviceTest

Ending mic testing (for desktop OS)

startSpeakerDeviceTest:

startSpeakerDeviceTest:

- (NSInteger)startSpeakerDeviceTest:	(NSString *)audioFilePath

Starting speaker testing (for desktop OS)

This API is used to test whether the audio playback device functions properly by playing a specified audio file. If users can hear audio during testing, the device functions properly.

Param	DESC
filePath	Path of the audio file

stopSpeakerDeviceTest

stopSpeakerDeviceTest

Ending speaker testing (for desktop OS)

setObserver:

setObserver:

- (void)setObserver:	(nullable id<TXDeviceObserver>) observer

set onDeviceChanged callback (for Mac)

setCameraCapturerParam:

setCameraCapturerParam:

- (void)setCameraCapturerParam:	(TXCameraCaptureParam *)params

Set camera acquisition preferences

setSystemVolumeType:

setSystemVolumeType:

- (NSInteger)setSystemVolumeType:	(TXSystemVolumeType)type

Setting the system volume type (for mobile OS)

@deprecated This API is not recommended after v9.5. Please use the `startLocalAudio(quality)` API in `TRTCCloud` instead, which param `quality` is used to decide audio quality.

TXSystemVolumeType(Deprecated)

TXSystemVolumeType(Deprecated)

System volume type

Enum	Value	DESC
TXSystemVolumeTypeAuto	0	Auto
TXSystemVolumeTypeMedia	1	Media volume
TXSystemVolumeTypeVOIP	2	Call volume

TXAudioRoute

TXAudioRoute

Audio route (the route via which audio is played)

Audio route is the route (speaker or receiver) via which audio is played. It applies only to mobile devices such as mobile phones.

A mobile phone has two speakers: one at the top (receiver) and the other the bottom.

If the audio route is set to the receiver, the volume is relatively low, and audio can be heard only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.

If the audio route is set to the speaker, the volume is relatively high, and there is no need to put the phone near the ear. This mode enables the "hands-free" feature.

Enum	Value	DESC
TXAudioRouteSpeakerphone	0	Speakerphone: the speaker at the bottom is used for playback (hands-free). With relatively high volume, it is used to play music out loud.
TXAudioRouteEarpiece	1	Earpiece: the receiver at the top is used for playback. With relatively low volume, it is suitable for call scenarios that require privacy.

TXMediaDeviceType

TXMediaDeviceType

Device type (for desktop OS)

This enumerated type defines three types of audio/video devices, namely camera, mic and speaker, so that you can use the same device management API to manage three types of devices.

Enum	Value	DESC
TXMediaDeviceTypeUnknown	-1	undefined device type
TXMediaDeviceTypeAudioInput	0	microphone
TXMediaDeviceTypeAudioOutput	1	speaker or earpiece
TXMediaDeviceTypeVideoCamera	2	camera

TXMediaDeviceState

TXMediaDeviceState

Device operation

This enumerated value is used to notify the status change of the local device onDeviceChanged.

Enum	Value	DESC
TXMediaDeviceStateAdd	0	The device has been plugged in
TXMediaDeviceStateRemove	1	The device has been removed
TXMediaDeviceStateActive	2	The device has been enabled
TXMediaDefaultDeviceChanged	3	system default device changed

TXCameraCaptureMode

TXCameraCaptureMode

Camera acquisition preferences

This enum is used to set camera acquisition parameters.

Enum	Value	DESC
TXCameraResolutionStrategyAuto	0	Auto adjustment of camera capture parameters. SDK selects the appropriate camera output parameters according to the actual acquisition device performance and network situation, and maintains a balance between device performance and video preview quality.
TXCameraResolutionStrategyPerformance	Not Defined	Give priority to equipment performance. SDK selects the closest camera output parameters according to the user's encoder resolution and frame rate, so as to ensure the performance of the device.
TXCameraResolutionStrategyHighQuality	Not Defined	Give priority to the quality of video preview. SDK selects higher camera output parameters to improve the quality of

		preview video. In this case, it will consume more CPU and memory to do video preprocessing.
TXCameraCaptureManual	Not Defined	Allows the user to set the width and height of the video captured by the local camera.

TXCameraCaptureParam

TXCameraCaptureParam

Camera acquisition parameters

This setting determines the quality of the local preview image.

EnumType	DESC
height	Field description: height of acquired image
mode	Field description: camera acquisition preferences,please see TXCameraCaptureMode
width	Field description: width of acquired image

TXMediaDeviceInfo

TXMediaDeviceInfo

Audio/Video device information (for desktop OS)

This structure describes key information (such as device ID and device name) of an audio/video device, so that users can choose on the UI the device to use.

EnumType	DESC
deviceId	device id (UTF-8)
deviceName	device name (UTF-8)
deviceProperties	device properties
type	device type

Type Definition

Last updated：2024-06-06 15:50:05

Module: TRTC key class definition

Description: definitions of enumerated and constant values such as resolution and quality level

Type Define

StructType

FuncList	DESC
TRTCParams	Room entry parameters
TRTCVideoEncParam	Video encoding parameters
TRTCNetworkQosParam	Network QoS control parameter set
TRTCRenderParams	Rendering parameters of video image
TRTCQualityInfo	Network quality
TRTCVolumeInfo	Volume
TRTCSpeedTestParams	Network speed testing parameters
TRTCSpeedTestResult	Network speed test result
TRTCVideoFrame	Video frame information
TRTCAudioFrame	Audio frame data
TRTCMixUser	Description information of each video image in On-Cloud MixTranscoding
TRTCTranscodingConfig	Layout and transcoding parameters of On-Cloud MixTranscoding
TRTCPublishCDNParam	Push parameters required to be set when publishing audio/video streams to non-Tencent Cloud CDN

TRTCAudioRecordingParams	Local audio file recording parameters
TRTCLocalRecordingParams	Local media file recording parameters
TRTCAudioEffectParam	Sound effect parameter (disused)
TRTCSwitchRoomConfig	Room switch parameter
TRTCAudioFrameDelegateFormat	Format parameter of custom audio callback
TRTCUser	The users whose streams to publish
TRTCPublishCdnUrl	The destination URL when you publish to Tencent Cloud or a third-party CDN
TRTCPublishTarget	The publishing destination
TRTCVideoLayout	The video layout of the transcoded stream
TRTCWatermark	The watermark layout
TRTCStreamEncoderParam	The encoding parameters
TRTCStreamMixingConfig	The transcoding parameters
TRTCPayloadPrivateEncryptionConfig	Media Stream Private Encryption Configuration
TRTCAudioVolumeEvaluateParams	Volume evaluation and other related parameter settings.

EnumType

EnumType	DESC
TRTCVideoResolution	Video resolution
TRTCVideoResolutionMode	Video aspect ratio mode
TRTCVideoStreamType	Video stream type
TRTCVideoFillMode	Video image fill mode
TRTCVideoRotation	Video image rotation direction
TRTCBeautyStyle	Beauty (skin smoothing) filter algorithm
TRTCVideoPixelFormat	Video pixel format

TRTCVideoBufferType	Video data transfer method
TRTCVideoMirrorType	Video mirror type
TRTCSnapshotSourceType	Data source of local video screenshot
TRTCAppScene	Use cases
TRTCRoleType	Role
TRTCQosControlMode	QoS control mode (disused)
TRTCVideoQosPreference	Image quality preference
TRTCQuality	Network quality
TRTCAVStatusType	Audio/Video playback status
TRTCAVStatusChangeReason	Reasons for playback status changes
TRTCAudioSampleRate	Audio sample rate
TRTCAudioQuality	Sound quality
TRTCAudioRoute	Audio route (i.e., audio playback mode)
TRTCReverbType	Audio reverb mode
TRTCVoiceChangerType	Voice changing type
TRTCSystemVolumeType	System volume type (only for mobile devices)
TRTCAudioFrameOperationMode	Audio callback data operation mode
TRTCLogLevel	Log level
TRTCGSensorMode	G-sensor switch (for mobile devices only)
TRTCScreenCaptureSourceType	Screen sharing target type (for desktops only)
TRTCTranscodingConfigMode	Layout mode of On-Cloud MixTranscoding
TRTCRecordType	Media recording type
TRTCMixInputType	Stream mix input type
TRTCAudioRecordingContent	Audio recording content type
TRTCPublishMode	The publishing mode

TRTCEncryptionAlgorithm	Encryption Algorithm
TRTCSpeedTestScene	Speed Test Scene
TRTCGravitySensorAdaptiveMode	Set the adaptation mode of gravity sensing (only applicable to mobile terminals)

TRTCVideoResolution

TRTCVideoResolution

Video resolution

Here, only the landscape resolution (e.g., 640x360) is defined. If the portrait resolution (e.g., 360x640) needs to be used, `Portrait` must be selected for `TRTCVideoResolutionMode` .

Enum	Value	DESC
TRTCVideoResolution_120_120	1	Aspect ratio: 1:1; resolution: 120x120; recommended bitrate (VideoCall): 80 Kbps; recommended bitrate (LIVE): 120 Kbps.
TRTCVideoResolution_160_160	3	Aspect ratio: 1:1; resolution: 160x160; recommended bitrate (VideoCall): 100 Kbps; recommended bitrate (LIVE): 150 Kbps.
TRTCVideoResolution_270_270	5	Aspect ratio: 1:1; resolution: 270x270; recommended bitrate (VideoCall): 200 Kbps; recommended bitrate (LIVE): 300 Kbps.
TRTCVideoResolution_480_480	7	Aspect ratio: 1:1; resolution: 480x480; recommended bitrate (VideoCall): 350 Kbps; recommended bitrate (LIVE): 500 Kbps.
TRTCVideoResolution_160_120	50	Aspect ratio: 4:3; resolution: 160x120; recommended bitrate (VideoCall): 100 Kbps; recommended bitrate (LIVE): 150 Kbps.
TRTCVideoResolution_240_180	52	Aspect ratio: 4:3; resolution: 240x180; recommended bitrate (VideoCall): 150 Kbps; recommended bitrate (LIVE): 250 Kbps.
TRTCVideoResolution_280_210	54	Aspect ratio: 4:3; resolution: 280x210; recommended bitrate (VideoCall): 200 Kbps; recommended bitrate (LIVE): 300 Kbps.

TRTCVideoResolution_320_240	56	Aspect ratio: 4:3; resolution: 320x240; recommended bitrate (VideoCall): 250 Kbps; recommended bitrate (LIVE): 375 Kbps.
TRTCVideoResolution_400_300	58	Aspect ratio: 4:3; resolution: 400x300; recommended bitrate (VideoCall): 300 Kbps; recommended bitrate (LIVE): 450 Kbps.
TRTCVideoResolution_480_360	60	Aspect ratio: 4:3; resolution: 480x360; recommended bitrate (VideoCall): 400 Kbps; recommended bitrate (LIVE): 600 Kbps.
TRTCVideoResolution_640_480	62	Aspect ratio: 4:3; resolution: 640x480; recommended bitrate (VideoCall): 600 Kbps; recommended bitrate (LIVE): 900 Kbps.
TRTCVideoResolution_960_720	64	Aspect ratio: 4:3; resolution: 960x720; recommended bitrate (VideoCall): 1000 Kbps; recommended bitrate (LIVE): 1500 Kbps.
TRTCVideoResolution_160_90	100	Aspect ratio: 16:9; resolution: 160x90; recommended bitrate (VideoCall): 150 Kbps; recommended bitrate (LIVE): 250 Kbps.
TRTCVideoResolution_256_144	102	Aspect ratio: 16:9; resolution: 256x144; recommended bitrate (VideoCall): 200 Kbps; recommended bitrate (LIVE): 300 Kbps.
TRTCVideoResolution_320_180	104	Aspect ratio: 16:9; resolution: 320x180; recommended bitrate (VideoCall): 250 Kbps; recommended bitrate (LIVE): 400 Kbps.
TRTCVideoResolution_480_270	106	Aspect ratio: 16:9; resolution: 480x270; recommended bitrate (VideoCall): 350 Kbps; recommended bitrate (LIVE): 550 Kbps.
TRTCVideoResolution_640_360	108	Aspect ratio: 16:9; resolution: 640x360; recommended bitrate (VideoCall): 500 Kbps; recommended bitrate (LIVE): 900 Kbps.
TRTCVideoResolution_960_540	110	Aspect ratio: 16:9; resolution: 960x540; recommended bitrate (VideoCall): 850 Kbps; recommended bitrate (LIVE): 1300 Kbps.
TRTCVideoResolution_1280_720	112	Aspect ratio: 16:9; resolution: 1280x720; recommended bitrate (VideoCall): 1200 Kbps; recommended bitrate (LIVE): 1800 Kbps.

TRTCVideoResolution_1920_1080	114	Aspect ratio: 16:9; resolution: 1920x1080; recommended bitrate (VideoCall): 2000 Kbps; recommended bitrate (LIVE): 3000 Kbps.

TRTCVideoResolutionMode

TRTCVideoResolutionMode

Video aspect ratio mode

Only the landscape resolution (e.g., 640x360) is defined in `TRTCVideoResolution` . If the portrait resolution (e.g., 360x640) needs to be used, `Portrait` must be selected for `TRTCVideoResolutionMode` .

Enum	Value	DESC
TRTCVideoResolutionModeLandscape	0	Landscape resolution, such as TRTCVideoResolution_640_360 + TRTCVideoResolutionModeLandscape = 640x360.
TRTCVideoResolutionModePortrait	1	Portrait resolution, such as TRTCVideoResolution_640_360 + TRTCVideoResolutionModePortrait = 360x640.

TRTCVideoStreamType

TRTCVideoStreamType

Video stream type

TRTC provides three different video streams, including:

HD big image: it is generally used to transfer video data from the camera.

Smooth small image: it has the same content as the big image, but with lower resolution and bitrate and thus lower definition.

Substream image: it is generally used for screen sharing. Only one user in the room is allowed to publish the substream video image at any time, while other users must wait for this user to close the substream before they can publish their own substream.

Note

The SDK does not support enabling the smooth small image alone, which must be enabled together with the big image. It will automatically set the resolution and bitrate of the small image.

Enum	Value	DESC

TRTCVideoStreamTypeBig	0	HD big image: it is generally used to transfer video data from the camera.
TRTCVideoStreamTypeSmall	1	Smooth small image: it has the same content as the big image, but with lower resolution and bitrate and thus lower definition.
TRTCVideoStreamTypeSub	2	Substream image: it is generally used for screen sharing. Only one user in the room is allowed to publish the substream video image at any time, while other users must wait for this user to close the substream before they can publish their own substream.

TRTCVideoFillMode

TRTCVideoFillMode

Video image fill mode

If the aspect ratio of the video display area is not equal to that of the video image, you need to specify the fill mode:

Enum	Value	DESC
TRTCVideoFillMode_Fill	0	Fill mode: the video image will be centered and scaled to fill the entire display area, where parts that exceed the area will be cropped. The displayed image may be incomplete in this mode.
TRTCVideoFillMode_Fit	1	Fit mode: the video image will be scaled based on its long side to fit the display area, where the short side will be filled with black bars. The displayed image is complete in this mode, but there may be black bars.

TRTCVideoRotation

TRTCVideoRotation

Video image rotation direction

TRTC provides rotation angle setting APIs for local and remote images. The following rotation angles are all clockwise.

Enum	Value	DESC

TRTCVideoRotation_0	0	No rotation
TRTCVideoRotation_90	1	Clockwise rotation by 90 degrees
TRTCVideoRotation_180	2	Clockwise rotation by 180 degrees
TRTCVideoRotation_270	3	Clockwise rotation by 270 degrees

TRTCBeautyStyle

TRTCBeautyStyle

Beauty (skin smoothing) filter algorithm

TRTC has multiple built-in skin smoothing algorithms. You can select the one most suitable for your product.

Enum	Value	DESC
TRTCBeautyStyleSmooth	0	Smooth style, which uses a more radical algorithm for more obvious effect and is suitable for show live streaming.
TRTCBeautyStyleNature	1	Natural style, which retains more facial details for more natural effect and is suitable for most live streaming use cases.
TRTCBeautyStylePitu	2	Pitu style, which is provided by YouTu Lab. Its skin smoothing effect is between the smooth style and the natural style, that is, it retains more skin details than the smooth style and has a higher skin smoothing degree than the natural style.

TRTCVideoPixelFormat

TRTCVideoPixelFormat

Video pixel format

TRTC provides custom video capturing and rendering features.
For the custom capturing feature, you can use the following enumerated values to describe the pixel format of the video you capture.
For the custom rendering feature, you can specify the pixel format of the video you expect the SDK to call back.

Enum	Value	DESC
TRTCVideoPixelFormat_Unknown	0	Undefined format

TRTCVideoPixelFormat_I420	1	YUV420P (I420) format
TRTCVideoPixelFormat_Texture_2D	7	OpenGL 2D texture format
TRTCVideoPixelFormat_32BGRA	6	BGRA32 format
TRTCVideoPixelFormat_NV12	5	YUV420SP (NV12) format

TRTCVideoBufferType

TRTCVideoBufferType

Video data transfer method

For custom capturing and rendering features, you need to use the following enumerated values to specify the method of transferring video data:

Method 1. This method uses memory buffer to transfer video data. It is efficient on iOS but inefficient on Android. It is the only method supported on Windows currently.

Method 2. This method uses texture to transfer video data. It is efficient on both iOS and Android but is not supported on Windows. To use this method, you should have a general familiarity with OpenGL programming.

Enum	Value	DESC
TRTCVideoBufferType_Unknown	0	Undefined transfer method
TRTCVideoBufferType_PixelBuffer	1	Use memory buffer to transfer video data. iOS: `PixelBuffer` ; Android: `Direct Buffer` for JNI layer; Windows: memory data block.
TRTCVideoBufferType_NSData	2	Use memory buffer to transfer video data. iOS: more compact memory block in `NSData` type after additional processing; Android: `byte[]` for Java layer. This transfer method has a lower efficiency than other methods.
TRTCVideoBufferType_Texture	3	Use OpenGL texture to transfer video data

TRTCVideoMirrorType

TRTCVideoMirrorType

Video mirror type

Video mirroring refers to the left-to-right flipping of the video image, especially for the local camera preview image. After mirroring is enabled, it can bring anchors a familiar "look into the mirror" experience.

Enum	Value	DESC
TRTCVideoMirrorTypeAuto	0	Auto mode: mirror the front camera's image but not the rear camera's image (for mobile devices only).
TRTCVideoMirrorTypeEnable	1	Mirror the images of both the front and rear cameras.
TRTCVideoMirrorTypeDisable	2	Disable mirroring for both the front and rear cameras.

TRTCSnapshotSourceType

TRTCSnapshotSourceType

Data source of local video screenshot

The SDK can take screenshots from the following two data sources and save them as local files:
Video stream: the SDK screencaptures the native video content from the video stream. The screenshots are not controlled by the display of the rendering control.
Rendering layer: the SDK screencaptures the displayed video content from the rendering control, which can achieve the effect of WYSIWYG, but if the display area is too small, the screenshots will also be very small.

Enum	Value	DESC
TRTCSnapshotSourceTypeStream	0	The SDK screencaptures the native video content from the video stream. The screenshots are not controlled by the display of the rendering control.
TRTCSnapshotSourceTypeView	1	The SDK screencaptures the displayed video content from the rendering control, which can achieve the effect of WYSIWYG, but if the display area is too small, the screenshots will also be very small.
TRTCSnapshotSourceTypeCapture	2	The SDK screencaptures the capture video content from the capture control, which can capture the captured high-definition screenshots.

TRTCAppScene

TRTCAppScene

Use cases

TRTC features targeted optimizations for common audio/video application scenarios to meet the differentiated requirements in various verticals. The main scenarios can be divided into the following two categories:

Live streaming scenario (LIVE): including `LIVE` (audio + video) and `VoiceChatRoom` (pure audio).

In the live streaming scenario, users are divided into two roles: "anchor" and "audience". A single room can sustain up to 100,000 concurrent online users. This is suitable for live streaming to a large audience.

Real-Time scenario (RTC): including `VideoCall` (audio + video) and `AudioCall` (pure audio).

In the real-time scenario, there is no role difference between users, but a single room can sustain only up to 300 concurrent online users. This is suitable for small-scale real-time communication.

Enum	Value	DESC
TRTCAppSceneVideoCall	0	In the video call scenario, 720p and 1080p HD image quality is supported. A single room can sustain up to 300 concurrent online users, and up to 50 of them can speak simultaneously. Use cases: [one-to-one video call], [video conferencing with up to 300 participants], [online medical diagnosis], [small class], [video interview], etc.
TRTCAppSceneLIVE	1	In the interactive video live streaming scenario, mic can be turned on/off smoothly without waiting for switchover, and the anchor latency is as low as less than 300 ms. Live streaming to hundreds of thousands of concurrent users in the audience role is supported with the playback latency down to 1,000 ms. Use cases: [low-latency interactive live streaming], [big class], [anchor competition], [video dating room], [online interactive classroom], [remote training], [large-scale conferencing], etc. **Note** In this scenario, you must use the `role` field in `TRTCParams` to specify the role of the current user.
TRTCAppSceneAudioCall	2	Audio call scenario, where the `SPEECH` sound quality is used by default. A single room can sustain up to 300 concurrent online users, and up to 50 of them can speak simultaneously. Use cases: [one-to-one audio call], [audio conferencing with up to 300 participants], [audio chat], [online Werewolf], etc.
TRTCAppSceneVoiceChatRoom	3	In the interactive audio live streaming scenario, mic can be turned on/off smoothly without waiting for switchover,

		and the anchor latency is as low as less than 300 ms. Live streaming to hundreds of thousands of concurrent users in the audience role is supported with the playback latency down to 1,000 ms. Use cases: [audio club], [online karaoke room], [music live room], [FM radio], etc. **Note** In this scenario, you must use the `role` field in `TRTCParams` to specify the role of the current user.

TRTCRoleType

TRTCRoleType

Role

Role is applicable only to live streaming scenarios (`TRTCAppSceneLIVE` and `TRTCAppSceneVoiceChatRoom`). Users are divided into two roles:
Anchor, who can publish their audio/video streams. There is a limit on the number of anchors. Up to 50 anchors are allowed to publish streams at the same time in one room.
Audience, who can only listen to or watch audio/video streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room can sustain up to 100,000 concurrent online users in the audience role.

Enum	Value	DESC
TRTCRoleAnchor	20	An anchor can publish their audio/video streams. There is a limit on the number of anchors. Up to 50 anchors are allowed to publish streams at the same time in one room.
TRTCRoleAudience	21	Audience can only listen to or watch audio/video streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room can sustain up to 100,000 concurrent online users in the audience role.

TRTCQosControlMode(Deprecated)

TRTCQosControlMode(Deprecated)

QoS control mode (disused)

Enum	Value	DESC

TRTCQosControlModeClient	0	Client-based control, which is for internal debugging of SDK and shall not be used by users.
TRTCQosControlModeServer	1	On-cloud control, which is the default and recommended mode.

TRTCVideoQosPreference

TRTCVideoQosPreference

Image quality preference

TRTC has two control modes in weak network environments: "ensuring clarity" and "ensuring smoothness". Both modes will give priority to the transfer of audio data.

Enum	Value	DESC
TRTCVideoQosPreferenceSmooth	1	Ensuring smoothness: in this mode, when the current network is unable to transfer a clear and smooth video image, the smoothness of the image will be given priority, but there will be blurs.
TRTCVideoQosPreferenceClear	2	Ensuring clarity (default value): in this mode, when the current network is unable to transfer a clear and smooth video image, the clarity of the image will be given priority, but there will be lags.

TRTCQuality

TRTCQuality

Network quality

TRTC evaluates the current network quality once every two seconds. The evaluation results are divided into six levels: `Excellent` indicates the best, and `Down` indicates the worst.

Enum	Value	DESC
TRTCQuality_Unknown	0	Undefined
TRTCQuality_Excellent	1	The current network is excellent
TRTCQuality_Good	2	The current network is good

TRTCQuality_Poor	3	The current network is fair
TRTCQuality_Bad	4	The current network is bad
TRTCQuality_Vbad	5	The current network is very bad
TRTCQuality_Down	6	The current network cannot meet the minimum requirements of TRTC

TRTCAVStatusType

TRTCAVStatusType

Audio/Video playback status

This enumerated type is used in the audio status changed API onRemoteAudioStatusUpdated and the video status changed API onRemoteVideoStatusUpdated to specify the current audio/video status.

Enum	Value	DESC
TRTCAVStatusStopped	0	Stopped
TRTCAVStatusPlaying	1	Playing
TRTCAVStatusLoading	2	Loading

TRTCAVStatusChangeReason

TRTCAVStatusChangeReason

Reasons for playback status changes

This enumerated type is used in the audio status changed API onRemoteAudioStatusUpdated and the video status changed API onRemoteVideoStatusUpdated to specify the reason for the current audio/video status change.

Enum	Value	DESC
TRTCAVStatusChangeReasonInternal	0	Default value
TRTCAVStatusChangeReasonBufferingBegin	1	The stream enters the `Loading` state due to network congestion
TRTCAVStatusChangeReasonBufferingEnd	2	The stream enters the `Playing` state after network recovery

TRTCAVStatusChangeReasonLocalStarted	3	As a start-related API was directly called locally, the stream enters the `Playing` state
TRTCAVStatusChangeReasonLocalStopped	4	As a stop-related API was directly called locally, the stream enters the `Stopped` state
TRTCAVStatusChangeReasonRemoteStarted	5	As the remote user started (or resumed) publishing the audio or video stream, the stream enters the `Loading` or `Playing` state
TRTCAVStatusChangeReasonRemoteStopped	6	As the remote user stopped (or paused) publishing the audio or video stream, the stream enters the "Stopped" state

TRTCAudioSampleRate

TRTCAudioSampleRate

Audio sample rate

The audio sample rate is used to measure the audio fidelity. A higher sample rate indicates higher fidelity. If there is music in the use case, `TRTCAudioSampleRate48000` is recommended.

Enum	Value	DESC
TRTCAudioSampleRate16000	16000	16 kHz sample rate
TRTCAudioSampleRate32000	32000	32 kHz sample rate
TRTCAudioSampleRate44100	44100	44.1 kHz sample rate
TRTCAudioSampleRate48000	48000	48 kHz sample rate

TRTCAudioQuality

TRTCAudioQuality

Sound quality

TRTC provides three well-tuned modes to meet the differentiated requirements for sound quality in various verticals:

Speech mode (Speech): it is suitable for application scenarios that focus on human communication. In this mode, the audio transfer is more resistant, and TRTC uses various voice processing technologies to ensure the optimal smoothness even in weak network environments.

Music mode (Music): it is suitable for scenarios with demanding requirements for music. In this mode, the amount of transferred audio data is very large, and TRTC uses various technologies to ensure that the high-fidelity details of music signals can be restored in each frequency band.

Default mode (Default): it is between `Speech` and `Music`. In this mode, the reproduction of music is better than that in `Speech` mode, and the amount of transferred data is much lower than that in `Music` mode; therefore, this mode has good adaptability to various scenarios.

Enum	Value	DESC
TRTCAudioQualitySpeech	1	Speech mode: sample rate: 16 kHz; mono channel; bitrate: 16 Kbps. This mode has the best resistance among all modes and is suitable for audio call scenarios, such as online meeting and audio call.
TRTCAudioQualityDefault	2	Default mode: sample rate: 48 kHz; mono channel; bitrate: 50 Kbps. This mode is between the speech mode and the music mode as the default mode in the SDK and is recommended.
TRTCAudioQualityMusic	3	Music mode: sample rate: 48 kHz; full-band stereo; bitrate: 128 Kbps. This mode is suitable for scenarios where Hi-Fi music transfer is required, such as online karaoke and music live streaming.

TRTCAudioRoute

TRTCAudioRoute

Audio route (i.e., audio playback mode)

"Audio route" determines whether the sound is played back from the speaker or receiver of a mobile device; therefore, this API is applicable only to mobile devices such as phones.

Generally, a phone has two speakers: one is the receiver at the top, and the other is the stereo speaker at the bottom.

If the audio route is set to the receiver, the volume is relatively low, and the sound can be heard clearly only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.

If the audio route is set to the speaker, the volume is relatively high, so there is no need to put the phone near the ear. Therefore, this mode can implement the "hands-free" feature.

Enum	Value	DESC
TRTCAudioModeSpeakerphone	0	Speakerphone: the speaker at the bottom is used for

		playback (hands-free). With relatively high volume, it is used to play music out loud.
TRTCAudioModeEarpiece	1	Earpiece: the receiver at the top is used for playback. With relatively low volume, it is suitable for call scenarios that require privacy.
TRTCAudioModeWiredHeadset	2	WiredHeadset：play using wired headphones.
TRTCAudioModeBluetoothHeadset	3	BluetoothHeadset：play with bluetooth headphones.
TRTCAudioModeSoundCard	4	SoundCard：play using a USB sound card.

TRTCReverbType

TRTCReverbType

Audio reverb mode

This enumerated value is used to set the audio reverb mode in the live streaming scenario and is often used in show live streaming.

Enum	Value	DESC
TRTCReverbType_0	0	Disable reverb
TRTCReverbType_1	1	KTV
TRTCReverbType_2	2	Small room
TRTCReverbType_3	3	Hall
TRTCReverbType_4	4	Deep
TRTCReverbType_5	5	Resonant
TRTCReverbType_6	6	Metallic
TRTCReverbType_7	7	Husky

TRTCVoiceChangerType

TRTCVoiceChangerType

Voice changing type

This enumerated value is used to set the voice changing mode in the live streaming scenario and is often used in show live streaming.

Enum	Value	DESC
TRTCVoiceChangerType_0	0	Disable voice changing
TRTCVoiceChangerType_1	1	Child
TRTCVoiceChangerType_2	2	Girl
TRTCVoiceChangerType_3	3	Middle-Aged man
TRTCVoiceChangerType_4	4	Heavy metal
TRTCVoiceChangerType_5	5	Nasal
TRTCVoiceChangerType_6	6	Punk
TRTCVoiceChangerType_7	7	Trapped beast
TRTCVoiceChangerType_8	8	Otaku
TRTCVoiceChangerType_9	9	Electronic
TRTCVoiceChangerType_10	10	Robot
TRTCVoiceChangerType_11	11	Ethereal

TRTCSystemVolumeType

TRTCSystemVolumeType

System volume type (only for mobile devices)

Smartphones usually have two types of system volume: call volume and media volume.
Call volume is designed for call scenarios. It comes with acoustic echo cancellation (AEC) and supports audio capturing by Bluetooth earphones, but its sound quality is average.
If you cannot turn the volume down to 0 (i.e., mute the phone) using the volume buttons, then your phone is using call volume.
Media volume is designed for media scenarios such as music playback. AEC does not work when media volume is used, and Bluetooth earphones cannot be used for audio capturing. However, media volume delivers better music listening experience.
If you are able to mute your phone using the volume buttons, then your phone is using media volume.

The SDK offers three system volume control modes: auto, call volume, and media volume.

Enum	Value	DESC
TRTCSystemVolumeTypeAuto	0	Auto: In the auto mode, call volume is used for anchors, and media volume for audience. This mode is suitable for live streaming scenarios. If the scenario you select during `enterRoom` is `TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom` , the SDK will automatically use this mode.
TRTCSystemVolumeTypeMedia	1	Media volume: In this mode, media volume is used in all scenarios. It is rarely used, mainly suitable for music scenarios with demanding requirements on audio quality. Use this mode if most of your users use peripheral devices such as audio cards. Otherwise, it is not recommended.
TRTCSystemVolumeTypeVOIP	2	Call volume: In this mode, the audio module does not change its work mode when users switch between anchors and audience, enabling seamless mic on/off. This mode is suitable for scenarios where users need to switch frequently between anchors and audience. If the scenario you select during `enterRoom` is `TRTCAppSceneVideoCall` or `TRTCAppSceneAudioCall` , the SDK will automatically use this mode.

TRTCAudioFrameOperationMode

TRTCAudioFrameOperationMode

Audio callback data operation mode

TRTC provides two modes of operation for audio callback data.
Read-only mode (ReadOnly): Get audio data only from the callback.
ReadWrite mode (ReadWrite): You can get and modify the audio data of the callback.

Enum	Value	DESC
TRTCAudioFrameOperationModeReadWrite	0	Read-write mode: You can get and modify

		the audio data of the callback, the default mode.
TRTCAudioFrameOperationModeReadOnly	1	Read-only mode: Get audio data from callback only.

TRTCLogLevel

TRTCLogLevel

Log level

Different log levels indicate different levels of details and number of logs. We recommend you set the log level to `TRTCLogLevelInfo` generally.

Enum	Value	DESC
TRTCLogLevelVerbose	0	Output logs at all levels
TRTCLogLevelDebug	1	Output logs at the DEBUG, INFO, WARNING, ERROR, and FATAL levels
TRTCLogLevelInfo	2	Output logs at the INFO, WARNING, ERROR, and FATAL levels
TRTCLogLevelWarn	3	Output logs at the WARNING, ERROR, and FATAL levels
TRTCLogLevelError	4	Output logs at the ERROR and FATAL levels
TRTCLogLevelFatal	5	Output logs at the FATAL level
TRTCLogLevelNone	6	Do not output any SDK logs

TRTCGSensorMode

TRTCGSensorMode

G-sensor switch (for mobile devices only)

Enum	Value	DESC
TRTCGSensorMode_Disable	0	Do not adapt to G-sensor orientation This mode is the default value for desktop platforms. In this mode, the video image published by the current

		user is not affected by the change of the G-sensor orientation.
TRTCGSensorMode_UIAutoLayout	1	Adapt to G-sensor orientation This mode is the default value on mobile platforms. In this mode, the video image published by the current user is adjusted according to the G-sensor orientation, while the orientation of the local preview image remains unchanged. One of the adaptation modes currently supported by the SDK is as follows: when the phone or tablet is upside down, in order to ensure that the screen orientation seen by the remote user is normal, the SDK will automatically rotate the published video image by 180 degrees. If the UI layer of your application has enabled G-sensor adaption, we recommend you use the `UIFixLayout` mode.
TRTCGSensorMode_UIFixLayout	2	Adapt to G-sensor orientation In this mode, the video image published by the current user is adjusted according to the G-sensor orientation, and the local preview image will also be rotated accordingly. One of the features currently supported is as follows: when the phone or tablet is upside down, in order to ensure that the screen orientation seen by the remote user is normal, the SDK will automatically rotate the published video image by 180 degrees. If the UI layer of your application doesn't support G-sensor adaption, but you want the video image in the SDK to adapt to the G-sensor orientation, we recommend you use the `UIFixLayout` mode. @deprecated Begin from v11.5 version, it no longer supports TRTCGSensorMode_UIFixLayout and only supports the above two modes.

TRTCScreenCaptureSourceType

TRTCScreenCaptureSourceType

Screen sharing target type (for desktops only)

Enum	Value	DESC

TRTCScreenCaptureSourceTypeUnknown	-1	Undefined
TRTCScreenCaptureSourceTypeWindow	0	The screen sharing target is the window of an application
TRTCScreenCaptureSourceTypeScreen	1	The screen sharing target is the entire screen

TRTCTranscodingConfigMode

TRTCTranscodingConfigMode

Layout mode of On-Cloud MixTranscoding

TRTC's On-Cloud MixTranscoding service can mix multiple audio/video streams in the room into one stream. Therefore, you need to specify the layout scheme of the video images. The following layout modes are provided:

Enum	Value	DESC
TRTCTranscodingConfigMode_Unknown	0	Undefined
TRTCTranscodingConfigMode_Manual	1	Manual layout mode In this mode, you need to specify the precise position of each video image. This mode has the highest degree of freedom, but its ease of use is the worst: You need to enter all the parameters in `TRTCTranscodingConfig` , including the position coordinates of each video image (TRTCMixUser). You need to listen on the `onUserVideoAvailable()` and `onUserAudioAvailable()` event callbacks in `TRTCCloudDelegate` and constantly adjust the `mixUsers` parameter according to the audio/video status of each user with mic on in the current room.
TRTCTranscodingConfigMode_Template_PureAudio	2	Pure audio mode This mode is suitable for pure audio scenarios such as audio call (AudioCall) and audio chat room (VoiceChatRoom).

		You only need to set it once through the `setMixTranscodingConfig()` API after room entry, and then the SDK will automatically mix the audio of all mic-on users in the room into the current user's live stream. You don't need to set the `mixUsers` parameter in `TRTCTranscodingConfig`; instead, you only need to set the `audioSampleRate`, `audioBitrate` and `audioChannels` parameters.
TRTCTranscodingConfigMode_Template_PresetLayout	3	Preset layout mode This is the most popular layout mode, because it allows you to set the position of each video image in advance through placeholders, and then the SDK automatically adjusts it dynamically according to the number of video images in the room. In this mode, you still need to set the `mixUsers` parameter, but you can set `userId` as a "placeholder". Placeholder values include: "$PLACE_HOLDER_REMOTE$": image of remote user. Multiple images can be set. "$PLACE_HOLDER_LOCAL_MAIN$": local camera image. Only one image can be set. "$PLACE_HOLDER_LOCAL_SUB$": local screen sharing image. Only one image can be set. In this mode, you don't need to listen on the `onUserVideoAvailable()` and `onUserAudioAvailable()` callbacks in `TRTCCloudDelegate` to make real-time adjustments. Instead, you only need to call `setMixTranscodingConfig()` once after successful room entry. Then,

		the SDK will automatically populate the placeholders you set with real `userId` values.
TRTCTranscodingConfigMode_Template_ScreenSharing	4	Screen sharing mode This mode is suitable for screen sharing-based use cases such as online education and supported only by the SDKs for Windows and macOS. In this mode, the SDK will first build a canvas according to the target resolution you set (through the `videoWidth` and `videoHeight` parameters). Before the teacher enables screen sharing, the SDK will scale up the teacher's camera image and draw it onto the canvas. After the teacher enables screen sharing, the SDK will draw the video image shared on the screen onto the same canvas. The purpose of this layout mode is to ensure consistency in the output resolution of the mixtranscoding module and avoid problems with blurred screen during course replay and webpage playback (web players don't support adjustable resolution). Meanwhile, the audio of mic-on students will be mixed into the teacher's audio/video stream by default. Video content is primarily the shared screen in teaching mode, and it is a waste of bandwidth to transfer camera image and screen image at the same time. Therefore, the recommended practice is to directly draw the camera image onto the current screen through the `setLocalVideoRenderCallback` API. In this mode, you don't need to set the `mixUsers` parameter in

		`TRTCTranscodingConfig` , and the SDK will not mix students' images so as not to interfere with the screen sharing effect. You can set width x height in `TRTCTranscodingConfig` to 0 px x 0 px, and the SDK will automatically calculate a suitable resolution based on the aspect ratio of the user's current screen. If the teacher's current screen width is less than or equal to 1920 px, the SDK will use the actual resolution of the teacher's current screen. If the teacher's current screen width is greater than 1920 px, the SDK will select one of the three resolutions of 1920x1080 (16:9), 1920x1200 (16:10), and 1920x1440 (4:3) according to the current screen aspect ratio.

TRTCRecordType

TRTCRecordType

Media recording type

This enumerated type is used in the local media recording API startLocalRecording to specify whether to record audio/video files or pure audio files.

Enum	Value	DESC
TRTCRecordTypeAudio	0	Record audio only
TRTCRecordTypeVideo	1	Record video only
TRTCRecordTypeBoth	2	Record both audio and video

TRTCMixInputType

TRTCMixInputType

Stream mix input type

Enum	Value	DESC
TRTCMixInputTypeUndefined	0	Default. Considering the compatibility with older versions, if you specify the inputType as Undefined, the SDK will determine the stream mix input type according to the value of the `pureAudio` parameter
TRTCMixInputTypeAudioVideo	1	Mix both audio and video
TRTCMixInputTypePureVideo	2	Mix video only
TRTCMixInputTypePureAudio	3	Mix audio only
TRTCMixInputTypeWatermark	4	Mix watermark In this case, you don't need to specify the `userId` parameter, but you need to specify the `image` parameter. It is recommended to use png format.

TRTCAudioRecordingContent

TRTCAudioRecordingContent

Audio recording content type

This enumerated type is used in the audio recording API startAudioRecording to specify the content of the recorded audio.

Enum	Value	DESC
TRTCAudioRecordingContentAll	0	Record both local and remote audio
TRTCAudioRecordingContentLocal	1	Record local audio only
TRTCAudioRecordingContentRemote	2	Record remote audio only

TRTCPublishMode

TRTCPublishMode

The publishing mode

This enum type is used by the publishing API startPublishMediaStream.

TRTC can mix multiple streams in a room and publish the mixed stream to a CDN or to a TRTC room. It can also publish the stream of the local user to Tencent Cloud or a third-party CDN.

You can specify one of the following publishing modes to use:

Enum	Value	DESC
TRTCPublishModeUnknown	0	Undefined
TRTCPublishBigStreamToCdn	1	Use this parameter to publish the primary stream (TRTCVideoStreamTypeBig) in the room to Tencent Cloud or a third-party CDN (only RTMP is supported).
TRTCPublishSubStreamToCdn	2	Use this parameter to publish the substream (TRTCVideoStreamTypeSub) in the room to Tencent Cloud or a third-party CDN (only RTMP is supported).
TRTCPublishMixStreamToCdn	3	Use this parameter together with the encoding parameter TRTCStreamEncoderParam and On-Cloud MixTranscoding parameter TRTCStreamMixingConfig to transcode the streams you specify and publish the mixed stream to Tencent Cloud or a third-party CDN (only RTMP is supported).
TRTCPublishMixStreamToRoom	4	Use this parameter together with the encoding parameter TRTCStreamEncoderParam and On-Cloud MixTranscoding parameter TRTCStreamMixingConfig to transcode the streams you specify and publish the mixed stream to the room you specify. Use `TRTCUser` in TRTCPublishTarget to specify the robot that publishes the transcoded stream to a TRTC room.

TRTCEncryptionAlgorithm

TRTCEncryptionAlgorithm

Encryption Algorithm

This enumeration type is used for media stream private encryption algorithm selection.

Enum	Value	DESC
TRTCEncryptionAlgorithmAes128Gcm	0	AES GCM 128。
TRTCEncryptionAlgorithmAes256Gcm	1	AES GCM 256。

TRTCSpeedTestScene

TRTCSpeedTestScene

Speed Test Scene

This enumeration type is used for speed test scene selection.

Enum	Value	DESC
TRTCSpeedTestScene_DelayTesting	1	Delay testing.
TRTCSpeedTestScene_DelayAndBandwidthTesting	2	Delay and bandwidth testing.
TRTCSpeedTestScene_OnlineChorusTesting	3	Online chorus testing.

TRTCGravitySensorAdaptiveMode

TRTCGravitySensorAdaptiveMode

Set the adaptation mode of gravity sensing (only applicable to mobile terminals)

Enum	Value	DESC
TRTCGravitySensorAdaptiveMode_Disable	0	Turn off the gravity sensor and make a decision based on the current acquisition resolution and the set encoding resolution. If the two are inconsistent, rotate 90 degrees to ensure the maximum frame.
TRTCGravitySensorAdaptiveMode_FillByCenterCrop	1	Turn on the gravity sensor to always ensure that the remote screen image is positive. When the intermediate process needs to deal with inconsistent resolutions, use the center cropping mode.
TRTCGravitySensorAdaptiveMode_FitWithBlackBorder	2	Turn on the gravity sensor to always ensure that the remote screen image is positive. When the resolution needs to be processed inconsistently in the

		intermediate process, use the superimposed black border mode.

TRTCParams

TRTCParams

Room entry parameters

As the room entry parameters in the TRTC SDK, these parameters must be correctly set so that the user can successfully enter the audio/video room specified by `roomId` or `strRoomId` .
For historical reasons, TRTC supports two types of room IDs: `roomId` and `strRoomId` .
Note: do not mix `roomId` and `strRoomId` , because they are not interchangeable. For example, the number `123` and the string `123` are two completely different rooms in TRTC.

EnumType	DESC
bussInfo	Field description: business data, which is optional. This field is needed only by some advanced features. Recommended value: do not set this field on your own.
privateMapKey	Field description: permission credential used for permission control, which is optional. If you want only users with the specified `userId` values to enter a room, you need to use `privateMapKey` to restrict the permission. Recommended value: we recommend you use this parameter only if you have high security requirements. For more information, please see Enabling Advanced Permission Control.
role	Field description: role in the live streaming scenario, which is applicable only to the live streaming scenario (TRTCAppSceneLIVE or TRTCAppSceneVoiceChatRoom) but doesn't take effect in the call scenario. Recommended value: default value: anchor (TRTCRoleAnchor).
roomId	Field description: numeric room ID. Users (userId) in the same room can see one another and make audio/video calls. Recommended value: value range: 1-4294967294. @note `roomId` and `strRoomId` are mutually exclusive. If you decide to use `strRoomId` , then `roomId` should be entered as 0. If both are entered, `roomId` will be used. **Note** do not mix `roomId` and `strRoomId` , because they are not interchangeable. For example, the number `123` and the string `123` are two completely different rooms in TRTC.

sdkAppId	Field description: application ID, which is required. Tencent Cloud generates bills based on `sdkAppId` . Recommended value: the ID can be obtained on the account information page in the TRTC console after the corresponding application is created.
strRoomId	Field description: string-type room ID. Users (userId) in the same room can see one another and make audio/video calls. @note `roomId` and `strRoomId` are mutually exclusive. If you decide to use `strRoomId` , then `roomId` should be entered as 0. If both are entered, `roomId` will be used. **Note** do not mix `roomId` and `strRoomId` , because they are not interchangeable. For example, the number `123` and the string `123` are two completely different rooms in TRTC. Recommended value: the length limit is 64 bytes. The following 89 characters are supported: Uppercase and lowercase letters (a–z and A–Z) Digits (0–9) Space, "!", "#", "$", "%", "&", "(", ")", "+", "-", ":", ";", "<", "=", ".", ">", "?", "@", "[", "]", "^", "_", "{", "}", "|", "~", and ",".
streamId	Field description: specified `streamId` in Tencent Cloud CSS, which is optional. After setting this field, you can play back the user's audio/video stream on Tencent Cloud CSS CDN through a standard pull scheme (FLV or HLS). Recommended value: this parameter can contain up to 64 bytes and can be left empty. We recommend you use `sdkappid_roomid_userid_main` as the `streamid` , which is easier to identify and will not cause conflicts in your multiple applications. **Note** to use Tencent Cloud CSS CDN, you need to enable the auto-relayed live streaming feature on the "Function Configuration" page in the console first. For more information, please see CDN Relayed Live Streaming.
userDefineRecordId	Field description: on-cloud recording field, which is optional and used to specify whether to record the user's audio/video stream in the cloud. For more information, please see On-Cloud Recording and Playback. Recommended value: it can contain up to 64 bytes. Letters (a–z and A–Z), digits (0–9), underscores, and hyphens are allowed. Scheme 1. Manual recording 1. Enable on-cloud recording in "Application Management" > "On-cloud Recording Configuration" in the console. 2. Set "Recording Mode" to "Manual Recording". 3. After manual recording is set, in a TRTC room, only users with the `userDefineRecordId` parameter set will have video recording files in the cloud, while users without this parameter set will not.

	4. The recording file will be named in the format of "userDefineRecordId_start time_end time" in the cloud. Scheme 2. Auto-recording 1. You need to enable on-cloud recording in "Application Management" > "On-cloud Recording Configuration" in the console. 2. Set "Recording Mode" to "Auto-recording". 3. After auto-recording is set, any user who upstreams audio/video in a TRTC room will have a video recording file in the cloud. 4. The file will be named in the format of "userDefineRecordId_start time_end time". If `userDefineRecordId` is not specified, the file will be named in the format of "streamId_start time_end time".
userId	Field description: user ID, which is required. It is the `userId` of the local user in UTF-8 encoding and acts as the username. Recommended value: if the ID of a user in your account system is "mike", `userId` can be set to "mike".
userSig	Field description: user signature, which is required. It is the authentication signature corresponding to the current `userId` and acts as the login password for Tencent Cloud services. Recommended value: for the calculation method, please see UserSig.

TRTCVideoEncParam

TRTCVideoEncParam

Video encoding parameters

These settings determine the quality of image viewed by remote users as well as the image quality of recorded video files in the cloud.

EnumType	DESC
enableAdjustRes	Field description: whether to allow dynamic resolution adjustment. Once enabled, this field will affect on-cloud recording. Recommended value: this feature is suitable for scenarios that don't require on-cloud recording. After it is enabled, the SDK will intelligently select a suitable resolution according to the current network conditions to avoid the inefficient encoding mode of "large resolution + small bitrate". **Note** default value: NO. If you need on-cloud recording, please do not enable this feature, because if the video resolution changes, the MP4 file recorded in the cloud cannot be played back normally by common players.
minVideoBitrate	Field description: minimum video bitrate. The SDK will reduce the bitrate to as low as

	the value specified by `minVideoBitrate` to ensure the smoothness only if the network conditions are poor. Note: default value: 0, indicating that a reasonable value of the lowest bitrate will be automatically calculated by the SDK according to the resolution you specify. Recommended value: you can set the `videoBitrate` and `minVideoBitrate` parameters at the same time to restrict the SDK's adjustment range of the video bitrate: If you want to "ensure clarity while allowing lag in weak network environments", you can set `minVideoBitrate` to 60% of `videoBitrate` . If you want to "ensure smoothness while allowing blur in weak network environments", you can set `minVideoBitrate` to a low value, for example, 100 Kbps. If you set `videoBitrate` and `minVideoBitrate` to the same value, it is equivalent to disabling the adaptive adjustment capability of the SDK for the video bitrate.
resMode	Field description: resolution mode (landscape/portrait) Recommended value: for mobile platforms (iOS and Android), `Portrait` is recommended; for desktop platforms (Windows and macOS), `Landscape` is recommended. **Note** to use a portrait resolution, please specify `resMode` as `Portrait` ; for example, when used together with `Portrait` , 640x360 represents 360x640.
videoBitrate	Field description: target video bitrate. The SDK encodes streams at the target video bitrate and will actively reduce the bitrate only in weak network environments. Recommended value: please see the optimal bitrate for each specification in `TRTCVideoResolution` . You can also slightly increase the optimal bitrate. For example, `TRTCVideoResolution_1280_720` corresponds to the target bitrate of 1,200 Kbps. You can also set the bitrate to 1,500 Kbps for higher definition. **Note** you can set the `videoBitrate` and `minVideoBitrate` parameters at the same time to restrict the SDK's adjustment range of the video bitrate: If you want to "ensure clarity while allowing lag in weak network environments", you can set `minVideoBitrate` to 60% of `videoBitrate` . If you want to "ensure smoothness while allowing blur in weak network environments", you can set `minVideoBitrate` to a low value, for example, 100 Kbps. If you set `videoBitrate` and `minVideoBitrate` to the same value, it is equivalent to disabling the adaptive adjustment capability of the SDK for the video bitrate.
videoFps	Field description: video capturing frame rate Recommended value: 15 or 20 fps. If the frame rate is lower than 5 fps, there will be obvious lagging; if lower than 10 fps but higher than 5 fps, there will be slight lagging;

	if higher than 20 fps, the bandwidth will be wasted (the frame rate of movies is generally 24 fps). **Note** the front cameras on certain Android phones do not support a capturing frame rate higher than 15 fps. For some Android phones that focus on beautification features, the capturing frame rate of the front cameras may be lower than 10 fps.
videoResolution	Field description: video resolution Recommended value For mobile video call, we recommend you select a resolution of 360x640 or below and select `Portrait` (portrait resolution) for `resMode` . For mobile live streaming, we recommend you select a resolution of 540x960 and select `Portrait` (portrait resolution) for `resMode` . For desktop platforms (Windows and macOS), we recommend you select a resolution of 640x360 or above and select `Landscape` (landscape resolution) for `resMode` . **Note** to use a portrait resolution, please specify `resMode` as `Portrait` ; for example, when used together with `Portrait` , 640x360 represents 360x640.

TRTCNetworkQosParam

TRTCNetworkQosParam

Network QoS control parameter set

Network QoS control parameter. The settings determine the QoS control policy of the SDK in weak network conditions (e.g., whether to "ensure clarity" or "ensure smoothness").

EnumType	DESC
controlMode	Field description: QoS control mode (disused) Recommended value: on-cloud control **Note** please set the on-cloud control mode (TRTCQosControlModeServer).
preference	Field description: whether to ensure smoothness or clarity Recommended value: ensuring clarity **Note** this parameter mainly affects the audio/video performance of TRTC in weak network environments: Ensuring smoothness: in this mode, when the current network is unable to transfer a clear and smooth video image, the smoothness of the image will be given priority, but there will be blurs. See TRTCVideoQosPreferenceSmooth

	Ensuring clarity (default value): in this mode, when the current network is unable to transfer a clear and smooth video image, the clarity of the image will be given priority, but there will be lags. See TRTCVideoQosPreferenceClear

TRTCRenderParams

TRTCRenderParams

Rendering parameters of video image

You can use these parameters to control the video image rotation angle, fill mode, and mirror mode.

EnumType	DESC
fillMode	Field description: image fill mode Recommended value: fill (the image may be stretched or cropped) or fit (there may be black bars in unmatched areas). Default value: TRTCVideoFillMode_Fill
mirrorType	Field description: image mirror mode Recommended value: default value: TRTCVideoMirrorType_Auto
rotation	Field description: clockwise image rotation angle Recommended value: rotation angles of 90, 180, and 270 degrees are supported. Default value: TRTCVideoRotation_0

TRTCQuality

TRTCQuality

Network quality

This indicates the quality of the network. You can use it to display the network quality of each user on the UI.

EnumType	DESC
quality	Network quality
userId	User ID

TRTCVolumeInfo

TRTCVolumeInfo

Volume

This indicates the audio volume value. You can use it to display the volume of each user in the UI.

EnumType	DESC
pitch	The local user's vocal frequency (unit: Hz), the value range is [0 - 4000]. For remote users, this value is always 0.
spectrumData	Audio spectrum data, which divides the sound frequency into 256 frequency domains, spectrumData records the energy value of each frequency domain, The value range of each energy value is [-300, 0] in dBFS. **Note** The local spectrum is calculated using the audio data before encoding, which will be affected by the capture volume, BGM, etc.; the remote spectrum is calculated using the received audio data, and operations such as adjusting the remote playback volume locally will not affect it.
userId	`userId` of the speaker. An empty value indicates the local user.
vad	Vad result of the local user. 0: not speech 1: speech.
volume	Volume of the speaker. Value range: 0–100.

TRTCSpeedTestParams

TRTCSpeedTestParams

Network speed testing parameters

You can test the network speed through the startSpeedTest: interface before the user enters the room (this API cannot be called during a call).

EnumType	DESC
expectedDownBandwidth	Expected downstream bandwidth (kbps, value range: 10 to 5000, no downlink bandwidth test when it is 0). **Note** When the parameter `scene` is set to `TRTCSpeedTestScene_OnlineChorusTesting` , in order to obtain more accurate information such as rtt / jitter, the value range is limited to 10 ~ 1000.
expectedUpBandwidth	Expected upstream bandwidth (kbps, value range: 10 to 5000, no uplink bandwidth test when it is 0). **Note**

	When the parameter `scene` is set to `TRTCSpeedTestScene_OnlineChorusTesting`, in order to obtain more accurate information such as rtt / jitter, the value range is limited to 10 ~ 1000.
scene	Speed test scene.
sdkAppId	Application identification, please refer to the relevant instructions in TRTCParams.
userId	User identification, please refer to the relevant instructions in TRTCParams.
userSig	User signature, please refer to the relevant instructions in TRTCParams.

TRTCSpeedTestResult

TRTCSpeedTestResult

Network speed test result

The startSpeedTest: API can be used to test the network speed before a user enters a room (this API cannot be called during a call).

EnumType	DESC
availableDownBandwidth	Downstream bandwidth (in kbps, -1: invalid value).
availableUpBandwidth	Upstream bandwidth (in kbps, -1: invalid value).
downJitter	Downlink data packet jitter (ms) refers to the stability of data communication in the user's current network environment. The smaller the value, the better. The normal value range is 0ms - 100ms. -1 means that the speed test failed to obtain an effective value. Generally, the Jitter of the WiFi network will be slightly larger than that of the 4G/5G environment.
downLostRate	Downstream packet loss rate between 0 and 1.0. For example, 0.2 indicates that 2 data packets may be lost in every 10 packets received from the server.
errMsg	Error message for network speed test.
ip	Server IP address.
quality	Network quality, which is tested and calculated based on the internal evaluation algorithm. For more information, please see TRTCQuality

rtt	Delay in milliseconds, which is the round-trip time between the current device and TRTC server. The smaller the value, the better. The normal value range is 10–100 ms.
success	Whether the network speed test is successful.
upJitter	Uplink data packet jitter (ms) refers to the stability of data communication in the user's current network environment. The smaller the value, the better. The normal value range is 0ms - 100ms. -1 means that the speed test failed to obtain an effective value. Generally, the Jitter of the WiFi network will be slightly larger than that of the 4G/5G environment.
upLostRate	Upstream packet loss rate between 0 and 1.0. For example, 0.3 indicates that 3 data packets may be lost in every 10 packets sent to the server.

TRTCVideoFrame

TRTCVideoFrame

Video frame information

`TRTCVideoFrame` is used to describe the raw data of a frame of the video image, which is the image data before frame encoding or after frame decoding.

EnumType	DESC
bufferType	Field description: video data structure type
data	Field description: video data when `bufferType` is TRTCVideoBufferType_NSData, which carries the memory data blocks in `NSData` type.
height	Field description: video height Recommended value: please enter the height of the video data passed in.
pixelBuffer	Field description: video data when `bufferType` is TRTCVideoBufferType_PixelBuffer, which carries the `PixelBuffer` unique to iOS.
pixelFormat	Field description: video pixel format
rotation	Field description: clockwise rotation angle of video pixels
textureId	Field description: video texture ID, i.e., video data when `bufferType` is TRTCVideoBufferType_Texture, which carries the texture data used for OpenGL rendering.

timestamp	Field description: video frame timestamp in milliseconds Recommended value: this parameter can be set to 0 for custom video capturing. In this case, the SDK will automatically set the `timestamp` field. However, please "evenly" set the calling interval of `sendCustomVideoData` .
width	Field description: video width Recommended value: please enter the width of the video data passed in.

TRTCAudioFrame

TRTCAudioFrame

Audio frame data

EnumType	DESC
channels	Field description: number of sound channels
data	Field description: audio data
extraData	Field description: extra data in audio frame, message sent by remote users through `onLocalProcessedAudioFrame` that add to audio frame will be callback through this field.
sampleRate	Field description: sample rate
timestamp	Field description: timestamp in ms

TRTCMixUser

TRTCMixUser

Description information of each video image in On-Cloud MixTranscoding

`TRTCMixUser` is used to specify the location, size, layer, and stream type of each video image in On-Cloud MixTranscoding.

EnumType	DESC
image	Field description: specify the placeholder or watermark image. The placeholder image will be displayed when there is no upstream video.A watermark image is a semi-transparent image posted in the mixed image, and this image will always be overlaid on the mixed image.

	When the `inputType` field is set to TRTCMixInputTypePureAudio, the image is a placeholder image, and you need to specify `userId` . When the `inputType` field is set to TRTCMixInputTypeWatermark, the image is a watermark image, and you don't need to specify `userId` . Recommended value: default value: null, indicating not to set the placeholder or watermark image. **Note** TRTC's backend service will mix the image specified by the URL address into the final stream.URL link length is limited to 512 bytes. The image size is limited to 10MB.Support png, jpg, jpeg, bmp format. Take effects iff the `inputType` field is set to TRTCMixInputTypePureAudio or TRTCMixInputTypeWatermark.
inputType	Field description: specify the mixed content of this stream (audio only, video only, audio and video, or watermark). Recommended value: default value: TRTCMixInputTypeUndefined. **Note** When specifying `inputType` as TRTCMixInputTypeUndefined and specifying `pureAudio` to YES, it is equivalent to setting `inputType` to `TRTCMixInputTypePureAudio` . When specifying `inputType` as TRTCMixInputTypeUndefined and specifying `pureAudio` to NO, it is equivalent to setting `inputType` to `TRTCMixInputTypeAudioVideo` . When specifying `inputType` as TRTCMixInputTypeWatermark, you don't need to specify the `userId` field, but you need to specify the `image` field.
pureAudio	Field description: specify whether this stream mixes audio only Recommended value: default value: NO **Note** this field has been disused. We recommend you use the new field `inputType` introduced in v8.5.
rect	Field description: specify the coordinate area of this video image in px
renderMode	Field description: specify the display mode of this stream. Recommended value: default value: 0. 0 is cropping, 1 is zooming, 2 is zooming and displaying black background. **Note** image doesn't support setting `renderMode` temporarily, the default display mode is forced stretch.
roomID	Field description: ID of the room where this audio/video stream is located (an empty value indicates the local room ID)
soundLevel	Field description: specify the target volumn level of On-Cloud MixTranscoding. (value range: 0-100) Recommended value: default value: 100.

streamType	Field description: specify whether this video image is the primary stream image (TRTCVideoStreamTypeBig) or substream image (TRTCVideoStreamTypeSub).
userId	Field description: user ID
zOrder	Field description: specify the level of this video image (value range: 1–15; the value must be unique)

TRTCTranscodingConfig

TRTCTranscodingConfig

Layout and transcoding parameters of On-Cloud MixTranscoding

These parameters are used to specify the layout position information of each video image and the encoding parameters of mixtranscoding during On-Cloud MixTranscoding.

EnumType	DESC
appId	Field description: `appId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `appId` in `Relayed Live Streaming Info`.
audioBitrate	Field description: specify the target audio bitrate of On-Cloud MixTranscoding Recommended value: default value: 64 Kbps. Value range: [32,192].
audioChannels	Field description: specify the number of sound channels of On-Cloud MixTranscoding Recommended value: default value: 1, which means mono channel. Valid values: 1: mono channel; 2: dual channel.
audioCodec	Field description: specify the audio encoding type of On-Cloud MixTranscoding Recommended value: default value: 0, which means LC-AAC. Valid values: 0: LC-AAC; 1: HE-AAC; 2: HE-AACv2. **Note** HE-AAC and HE-AACv2 only support [48000, 44100, 32000, 24000, 16000] sample rate. HE-AACv2 only support dual channel. HE-AAC and HE-AACv2 take effects iff the output streamId is specified.
audioSampleRate	Field description: specify the target audio sample rate of On-Cloud MixTranscoding Recommended value: default value: 48000 Hz. Valid values: 12000 Hz, 16000 Hz, 22050 Hz, 24000 Hz, 32000 Hz, 44100 Hz, 48000 Hz.
backgroundColor	Field description: specify the background color of the mixed video image.

	Recommended value: default value: 0x000000, which means black and is in the format of hex number; for example: "0x61B9F1" represents the RGB color (97,158,241).
backgroundImage	Field description: specify the background image of the mixed video image. **Recommended value: default value: null, indicating not to set the background image. **Note** TRTC's backend service will mix the image specified by the URL address into the final stream.URL link length is limited to 512 bytes. The image size is limited to 10MB.Support png, jpg, jpeg, bmp format.
bizId	Field description: `bizId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `bizId` in `Relayed Live Streaming Info` .
mixUsers	Field description: specify the position, size, layer, and stream type of each video image in On-Cloud MixTranscoding Recommended value: this field is an array in `TRTCMixUser` type, where each element represents the information of a video image.
mode	Field description: layout mode Recommended value: please choose a value according to your business needs. The preset mode has better applicability.
streamId	Field description: ID of the live stream output to CDN Recommended value: default value: null, that is, the audio/video streams in the room will be mixed into the audio/video stream of the caller of this API. If you don't set this parameter, the SDK will execute the default logic, that is, it will mix the multiple audio/video streams in the room into the audio/video stream of the caller of this API, i.e., A + B => A. If you set this parameter, the SDK will mix the audio/video streams in the room into the live stream you specify, i.e., A + B => C (C is the `streamId` you specify).
videoBitrate	Field description: specify the target video bitrate (Kbps) of On-Cloud MixTranscoding Recommended value: if you enter 0, TRTC will estimate a reasonable bitrate value based on `videoWidth` and `videoHeight` . You can also refer to the recommended bitrate value in the video resolution enumeration definition (in the comment section).
videoFramerate	Field description: specify the target video frame rate (fps) of On-Cloud MixTranscoding Recommended value: default value: 15 fps. Value range: (0,30].
videoGOP	Field description: specify the target video keyframe interval (GOP) of On-Cloud

	MixTranscoding Recommended value: default value: 2 (in seconds). Value range: [1,8].
videoHeight	Field description: specify the target resolution (height) of On-Cloud MixTranscoding Recommended value: 640 px. If you only mix audio streams, please set both `width` and `height` to 0; otherwise, there will be a black background in the live stream after mixtranscoding.
videoSeiParams	Field description: SEI parameters. default value: null **Note** the parameter is passed in the form of a JSON string. Here is an example to use it: `json { "payLoadContent":"xxx", "payloadType":5, "payloadUuid":"1234567890abcdef1234567890abcdef", "interval":1000, "followIdr":false } ` The currently supported fields and their meanings are as follows: payloadContent: Required. The payload content of the passthrough SEI, which cannot be empty. payloadType: Required. The type of the SEI message, with a value range of 5 or an integer within the range of [100, 254] (excluding 244, which is an internally defined timestamp SEI). payloadUuid: Required when payloadType is 5, and ignored in other cases. The value must be a 32-digit hexadecimal number. interval: Optional, default is 1000. The sending interval of the SEI, in milliseconds. followIdr: Optional, default is false. When this value is true, the SEI will be ensured to be carried when sending a key frame, otherwise it is not guaranteed.
videoWidth	Field description: specify the target resolution (width) of On-Cloud MixTranscoding Recommended value: 360 px. If you only mix audio streams, please set both `width` and `height` to 0; otherwise, there will be a black background in the live stream after mixtranscoding.

TRTCPublishCDNParam

TRTCPublishCDNParam

Push parameters required to be set when publishing audio/video streams to non-Tencent Cloud CDN

TRTC's backend service supports publishing audio/video streams to third-party live CDN service providers through the standard RTMP protocol.
If you use the Tencent Cloud CSS CDN service, you don't need to care about this parameter; instead, just use the startPublish API.

EnumType	DESC
appId	Field description: `appId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `appId` in `Relayed Live Streaming Info` .
bizId	Field description: `bizId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `bizId` in `Relayed Live Streaming Info` .
streamId	Field description: specify the push address (in RTMP format) of this audio/video stream at the third-party live streaming service provider Recommended value: default value: null,that is, the audio/video streams in the room will be pushed to the target service provider of the caller of this API.
url	Field description: specify the push address (in RTMP format) of this audio/video stream at the third-party live streaming service provider Recommended value: the push URL rules vary greatly by service provider. Please enter a valid push URL according to the requirements of the target service provider. TRTC's backend server will push audio/video streams in the standard format to the third-party service provider according to the URL you enter. **Note** the push URL must be in RTMP format and meet the specifications of your target live streaming service provider; otherwise, the target service provider will reject the push requests from TRTC's backend service.

TRTCAudioRecordingParams

TRTCAudioRecordingParams

Local audio file recording parameters

This parameter is used to specify the recording parameters in the audio recording API startAudioRecording.

EnumType	DESC
filePath	Field description: storage path of the audio recording file, which is required. **Note**

	this path must be accurate to the file name and extension. The extension determines the format of the audio recording file. Currently, supported formats include PCM, WAV, and AAC. For example, if you specify the path as `mypath/record/audio.aac` , it means that you want the SDK to generate an audio recording file in AAC format.Please specify a valid path with read/write permissions; otherwise, the audio recording file cannot be generated.
maxDurationPerFile	Field description: `maxDurationPerFile` is the max duration of each recorded file segments, in milliseconds, with a minimum value of 10000. The default value is 0, indicating no segmentation.
recordingContent	Field description: Audio recording content type. Note: Record all local and remote audio by default.

TRTCLocalRecordingParams

TRTCLocalRecordingParams

Local media file recording parameters

This parameter is used to specify the recording parameters in the local media file recording API startLocalRecording. The `startLocalRecording` API is an enhanced version of the `startAudioRecording` API. The former can record video files, while the latter can only record audio files.

EnumType	DESC
filePath	Field description: address of the recording file, which is required. Please ensure that the path is valid with read/write permissions; otherwise, the recording file cannot be generated. **Note** this path must be accurate to the file name and extension. The extension determines the format of the recording file. Currently, only the MP4 format is supported. For example, if you specify the path as `mypath/record/test.mp4` , it means that you want the SDK to generate a local video file in MP4 format. Please specify a valid path with read/write permissions; otherwise, the recording file cannot be generated.
interval	Field description: `interval` is the update frequency of the recording information in milliseconds. Value range: 1000–10000. Default value: -1, indicating not to call back
maxDurationPerFile	Field description: `maxDurationPerFile` is the max duration of each recorded file segments, in milliseconds, with a minimum value of 10000. The

	default value is 0, indicating no segmentation.
recordType	Field description: media recording type, which is `TRTCRecordTypeBoth` by default, indicating to record both audio and video.

TRTCSwitchRoomConfig

TRTCSwitchRoomConfig

Room switch parameter

This parameter is used for the room switch API switchRoom, which can quickly switch a user from one room to another.

EnumType	DESC
privateMapKey	Field description: permission credential used for permission control, which is optional. If you want only users with the specified `userId` values to enter a room, you need to use `privateMapKey` to restrict the permission. Recommended value: we recommend you use this parameter only if you have high security requirements. For more information, please see Enabling Advanced Permission Control.
roomId	Field description: numeric room ID, which is optional. Users in the same room can see one another and make audio/video calls. Recommended value: value range: 1-4294967294. **Note** either `roomId` or `strRoomId` must be entered. If both are entered, `roomId` will be used.
strRoomId	Field description: string-type room ID, which is optional. Users in the same room can see one another and make audio/video calls. **Note** either `roomId` or `strRoomId` must be entered. If both are entered, `roomId` will be used.
userSig	Field description: user signature, which is optional. It is the authentication signature corresponding to the current `userId` and acts as the login password. If you don't specify the newly calculated `userSig` during room switch, the SDK will continue to use the `userSig` you specified during room entry (enterRoom). This requires you to ensure that the old `userSig` is still within the validity period allowed by the signature at the moment of room switch; otherwise, room switch will fail. Recommended value: for the calculation method, please see UserSig.

TRTCAudioFrameDelegateFormat

TRTCAudioFrameDelegateFormat

Format parameter of custom audio callback

This parameter is used to set the relevant format (including sample rate and number of channels) of the audio data called back by the SDK in the APIs related to custom audio callback.

EnumType	DESC
channels	Field description: number of sound channels Recommended value: default value: 1, which means mono channel. Valid values: 1: mono channel; 2: dual channel.
mode	Field description: audio callback data operation mode Recommended value: TRTCAudioFrameOperationModeReadOnly, get audio data from callback only. The modes that can be set are TRTCAudioFrameOperationModeReadOnly, TRTCAudioFrameOperationModeReadWrite.
sampleRate	Field description: sample rate Recommended value: default value: 48000 Hz. Valid values: 16000, 32000, 44100, 48000.
samplesPerCall	Field description: number of sample points Recommended value: the value must be an integer multiple of sampleRate/100.

TRTCUser

TRTCUser

The users whose streams to publish

You can use this parameter together with the publishing destination parameter TRTCPublishTarget and On-Cloud MixTranscoding parameter TRTCStreamMixingConfig to transcode the streams you specify and publish the mixed stream to the destination you specify.

EnumType	DESC
intRoomId	**Description:** Numeric room ID. The room ID must be of the same type as that in TRTCParams. **Value:** Value range: 1-4294967294 **Note:** You cannot use both `intRoomId` and `strRoomId`. If you specify `strRoomId`, you need to set `intRoomId` to `0`. If you set both, only

	`intRoomId` will be used.
strRoomId	**Description:** String-type room ID. The room ID must be of the same type as that in TRTCParams. **Note:** You cannot use both `intRoomId` and `strRoomId` . If you specify `roomId` , you need to leave `strRoomId` empty. If you set both, only `intRoomId` will be used. **Value:** 64 bytes or shorter; supports the following character set (89 characters): Uppercase and lowercase letters (a-z and A-Z) Numbers (0-9) Space, "!", "#", "$", "%", "&", "(", ")", "+", "-", ":", ";", "<", "=", ".", ">", "?", "@", "[", "]", "^", "_", " {", "}", "|", "~", ","
userId	/**Description**: UTF-8-encoded user ID (required) **Value:** For example, if the ID of a user in your account system is "mike", set it to `mike` .

TRTCPublishCdnUrl

TRTCPublishCdnUrl

The destination URL when you publish to Tencent Cloud or a third-party CDN

This enum type is used by the publishing destination parameter TRTCPublishTarget of the publishing API startPublishMediaStream.

EnumType	DESC
isInternalLine	**Description:** Whether to publish to Tencent Cloud **Value:** The default value is `true` . **Note:** If the destination URL you set is provided by Tencent Cloud, set this parameter to `true` , and you will not be charged relaying fees.
rtmpUrl	**Description:** The destination URL (RTMP) when you publish to Tencent Cloud or a third-party CDN. **Value:** The URLs of different CDN providers may vary greatly in format. Please enter a valid URL as required by your service provider. TRTC's backend server will push audio/video streams in the standard format to the URL you provide. **Note:** The URL must be in RTMP format. It must also meet the requirements of your service provider, or your service provider may reject push requests from the TRTC backend.

TRTCPublishTarget

TRTCPublishTarget

The publishing destination

This enum type is used by the publishing API startPublishMediaStream.

EnumType	DESC
cdnUrlList	`Description:` The destination URLs (RTMP) when you publish to Tencent Cloud or third-party CDNs. `Note:` You don't need to set this parameter if you set the publishing mode to `TRTCPublishMixStreamToRoom` .
mixStreamIdentity	`Description:` The information of the robot that publishes the transcoded stream to a TRTC room. `Note:` You need to set this parameter only if you set the publishing mode to `TRTCPublishMixStreamToRoom` . `Note:` After you set this parameter, the stream will be pushed to the room you specify. We recommend you set it to a special user ID to distinguish the robot from the anchor who enters the room via the TRTC SDK. `Note:` Users whose streams are transcoded cannot subscribe to the transcoded stream. `Note:` If you set the subscription mode (@link setDefaultStreamRecvMode}) to manual before room entry, you need to manage the streams to receive by yourself (normally, if you receive the transcoded stream, you need to unsubscribe from the streams that are transcoded). `Note:` If you set the subscription mode (setDefaultStreamRecvMode) to auto before room entry, users whose streams are not transcoded will receive the transcoded stream automatically and will unsubscribe from the users whose streams are transcoded. You call muteRemoteVideoStream and muteRemoteAudio to unsubscribe from the transcoded stream.
mode	`Description:` The publishing mode. `Value:` You can relay streams to a CDN, transcode streams, or publish streams to an RTC room. Select the mode that fits your needs. **Note** If you need to use more than one publishing mode, you can call startPublishMediaStream multiple times and set `TRTCPublishTarget` to a different value each time.You can use one mode each time you call the startPublishMediaStream) API. To modify the configuration, call updatePublishCDNStream.

TRTCVideoLayout

TRTCVideoLayout

The video layout of the transcoded stream

This enum type is used by the On-Cloud MixTranscoding parameter TRTCStreamMixingConfig of the publishing API startPublishMediaStream.

You can use this parameter to specify the position, size, layer, and stream type of each video in the transcoded stream.

EnumType	DESC
backgroundColor	`Description:` The background color of the mixed stream. `Value:` The value must be a hex number. For example, "0x61B9F1" represents the RGB color value (97,158,241). Default value: 0x000000 (black).
fillMode	`Description:` The rendering mode. `Value:` The rendering mode may be fill (the image may be stretched or cropped) or fit (there may be black bars). Default value: TRTCVideoFillMode_Fill.
fixedVideoStreamType	`Description:` Whether the video is the primary stream (TRTCVideoStreamTypeBig) or substream (e TRTCVideoStreamTypeSub).
fixedVideoUser	`Description:` The users whose streams are transcoded. **Note** If you do not specify TRTCUser (`userId` , `intRoomId` , `strRoomId`), the TRTC backend will automatically mix the streams of anchors who are sending audio/video in the room according to the video layout you specify.
placeHolderImage	`Description:` The URL of the placeholder image. If a user sends only audio, the image specified by the URL will be mixed during On-Cloud MixTranscoding. `Value:` This parameter is left empty by default, which means no placeholder image will be used. **Note** You need to specify the `userId` parameter in `fixedVideoUser` . The URL can be 512 bytes long at most, and the image must not exceed 2 MB. The image can be in PNG, JPG, JPEG, or BMP format. We recommend you use a semitransparent image in PNG format.
rect	`Description:` The coordinates (in pixels) of the video.
zOrder	`Description:` The layer of the video, which must be unique. Value

	range: 0-15.

TRTCWatermark

TRTCWatermark

The watermark layout

This enum type is used by the On-Cloud MixTranscoding parameter TRTCStreamMixingConfig of the publishing API startPublishMediaStream.

EnumType	DESC
rect	`Description:` The coordinates (in pixels) of the watermark.
watermarkUrl	`Description:` The URL of the watermark image. The image specified by the URL will be mixed during On-Cloud MixTranscoding. **Note** The URL can be 512 bytes long at most, and the image must not exceed 2 MB. The image can be in PNG, JPG, JPEG, or BMP format. We recommend you use a semitransparent image in PNG format.
zOrder	`Description:` The layer of the watermark, which must be unique. Value range: 0-15.

TRTCStreamEncoderParam

TRTCStreamEncoderParam

The encoding parameters

`Description:` This enum type is used by the publishing API startPublishMediaStream.

`Note:` This parameter is required if you set the publishing mode to `TRTCPublish_MixStream_ToCdn` or `TRTCPublish_MixStream_ToRoom` in TRTCPublishTarget.

`Note:` If you use the relay to CDN feature (the publishing mode set to `RTCPublish_BigStream_ToCdn` or `TRTCPublish_SubStream_ToCdn`), to improve the relaying stability and playback compatibility, we also recommend you set this parameter.

EnumType	DESC
audioEncodedChannelNum	`Description:` The sound channels of the stream to publish. `Value:` Valid values: 1 (mono channel); 2 (dual-channel). Default: 1.

audioEncodedCodecType	`Description:` The audio codec of the stream to publish. `Value:` Valid values: 0 (LC-AAC); 1 (HE-AAC); 2 (HE-AACv2). Default: 0. **Note** The audio sample rates supported by HE-AAC and HE-AACv2 are 48000, 44100, 32000, 24000, and 16000. When HE-AACv2 is used, the output stream can only be dual-channel.
audioEncodedKbps	`Description:` The audio bitrate (Kbps) of the stream to publish. `Value:` Value range: [32,192]. Default: 50.
audioEncodedSampleRate	`Description:` The audio sample rate of the stream to publish. `Value:` Valid values: [48000, 44100, 32000, 24000, 16000, 8000]. Default: 48000 (Hz).
videoEncodedCodecType	`Description:` The video codec of the stream to publish. `Value:` Valid values: 0 (H264); 1 (H265). Default: 0.
videoEncodedFPS	`Description:` The frame rate (fps) of the stream to publish. `Value:` Value range: (0,30]. Default: 20.
videoEncodedGOP	`Description:` The keyframe interval (GOP) of the stream to publish. `Value:` Value range: [1,5]. Default: 3 (seconds).
videoEncodedHeight	`Description:` The resolution (height) of the stream to publish. `Value:` Recommended value: 640. If you mix only audio streams, to avoid displaying a black video in the transcoded stream, set both `width` and `height` to `0` .
videoEncodedKbps	`Description:` The video bitrate (Kbps) of the stream to publish. `Value:` If you set this parameter to `0` , TRTC will work out a bitrate based on `videoWidth` and `videoHeight` . For details, refer to the recommended bitrates for the constants of the resolution enum type (see comment).
videoEncodedWidth	`Description:` The resolution (width) of the stream to publish. `Value:` Recommended value: 368. If you mix only audio streams, to avoid displaying a black video in the transcoded stream, set both `width` and `height` to `0` .
videoSeiParams	`Description:` SEI parameters. Default: null `Note:` the parameter is passed in the form of a JSON string. Here is an example to use it:


```
{
  "payLoadContent":"xxx",
  "payloadType":5,
  "payloadUuid":"1234567890abcdef1234567890abcdef",
  "interval":1000,
  "followIdr":false
}
```

The currently supported fields and their meanings are as follows:
payloadContent: Required. The payload content of the passthrough SEI, which cannot be empty.
payloadType: Required. The type of the SEI message, with a value range of 5 or an integer within the range of [100, 254] (excluding 244, which is an internally defined timestamp SEI).
payloadUuid: Required when payloadType is 5, and ignored in other cases. The value must be a 32-digit hexadecimal number.
interval: Optional, default is 1000. The sending interval of the SEI, in milliseconds.
followIdr: Optional, default is false. When this value is true, the SEI will be ensured to be carried when sending a key frame, otherwise it is not guaranteed.

TRTCStreamMixingConfig

TRTCStreamMixingConfig

The transcoding parameters

This enum type is used by the publishing API startPublishMediaStream.

You can use this parameter to specify the video layout and input audio information for On-Cloud MixTranscoding.

EnumType	DESC
audioMixUserList	`Description:` The information of each audio stream to mix. `Value:` This parameter is an array. Each `TRTCUser` element in the array indicates the information of an audio stream. **Note** If you do not specify this array, the TRTC backend will automatically mix all streams of the anchors who are sending audio in the room according to the audio encode param TRTCStreamEncoderParam you specify (currently only supports up to 16 audio and video inputs).
backgroundColor	`Description:` The background color of the mixed stream. `Value:` The value must be a hex number. For example, "0x61B9F1" represents the RGB color value (97,158,241). Default value: 0x000000 (black).
backgroundImage	`Description:` The URL of the background image of the mixed stream. The image specified by the URL will be mixed during On-Cloud MixTranscoding. `Value:` This parameter is left empty by default, which means no background image will be used. **Note** The URL can be 512 bytes long at most, and the image must not exceed 2 MB. The image can be in PNG, JPG, JPEG, or BMP format. We recommend you use a semitransparent image in PNG format.
videoLayoutList	`Description:` The position, size, layer, and stream type of each video in On-Cloud MixTranscoding. `Value:` This parameter is an array. Each `TRTCVideoLayout` element in the array indicates the information of a video in On-Cloud MixTranscoding.
watermarkList	`Description:` The position, size, and layer of each watermark image in On-Cloud MixTranscoding. `Value:` This parameter is an array. Each `TRTCWatermark` element in the array indicates the information of a watermark.

TRTCPayloadPrivateEncryptionConfig

TRTCPayloadPrivateEncryptionConfig

Media Stream Private Encryption Configuration

This configuration is used to set the algorithm and key for media stream private encryption.

EnumType	DESC
encryptionAlgorithm	`Description:` Encryption algorithm, the default is TRTCEncryptionAlgorithmAes128Gcm.
encryptionKey	`Description:` encryption key, string type. `Value:` If the encryption algorithm is TRTCEncryptionAlgorithmAes128Gcm, the key length must be 16 bytes; if the encryption algorithm is TRTCEncryptionAlgorithmAes256Gcm, the key length must be 32 bytes.
encryptionSalt	`Description:` Salt, initialization vector for encryption. `Value:` It is necessary to ensure that the array filled in this parameter is not empty, not all 0 and the data length is 32 bytes.

TRTCAudioVolumeEvaluateParams

TRTCAudioVolumeEvaluateParams

Volume evaluation and other related parameter settings.

This setting is used to enable vocal detection and sound spectrum calculation.

EnumType	DESC
enablePitchCalculation	`Description:` Whether to enable local vocal frequency calculation.
enableSpectrumCalculation	`Description:` Whether to enable sound spectrum calculation.
enableVadDetection	`Description:` Whether to enable local voice detection. **Note** Call before startLocalAudio.
interval	`Description:` Set the trigger interval of the onUserVoiceVolume callback, the unit is milliseconds, the minimum interval is 100ms, if it is less than or equal to 0, the callback will be closed. `Value:` Recommended value: 300, in milliseconds. **Note**

	When the interval is greater than 0, the volume prompt will be enabled by default, no additional setting is required.

Deprecated Interface

Last updated：2024-06-06 15:50:05

Deprecate

TRTCCloud

FuncList	DESC
destroySharedIntance	Terminate TRTCCloud instance (singleton mode)
delegate	Set TRTC event callback
setBeautyStyle:beautyLevel:whitenessLevel:ruddinessLevel:	Set the strength of beauty, brightening, and rosy skin filters.
setEyeScaleLevel:	Set the strength of eye enlarging filter
setFaceScaleLevel:	Set the strength of face slimming filter
setFaceVLevel:	Set the strength of chin slimming filter
setChinLevel:	Set the strength of chin lengthening/shortening filter
setFaceShortLevel:	Set the strength of face shortening filter
setNoseSlimLevel:	Set the strength of nose slimming filter
selectMotionTmpl:	Set animated sticker
setMotionMute:	Mute animated sticker
setFilter:	Set color filter
setFilterConcentration:	Set the strength of color filter
setGreenScreenFile:	Set green screen video
setReverbType:	Set reverb effect
setVoiceChangerType:	Set voice changing type

enableAudioEarMonitoring:	Enable or disable in-ear monitoring
enableAudioVolumeEvaluation:	Enable volume reminder
enableAudioVolumeEvaluation:enable_vad:	Enable volume reminder
switchCamera	Switch camera
isCameraZoomSupported	Query whether the current camera supports zoom
setZoom:	Set camera zoom ratio (focal length)
isCameraTorchSupported	Query whether the device supports flash
enbaleTorch:	Enable/Disable flash
isCameraFocusPositionInPreviewSupported	Query whether the camera supports setting focus
setFocusPosition:	Set the focal position of camera
isCameraAutoFocusFaceModeSupported	Query whether the device supports the automatic recognition of face position
enableAutoFaceFoucs:	Enable/Disable face auto focus
setSystemVolumeType:	Setting the system volume type (for mobile OS)
snapshotVideo:type:	Screencapture video
startScreenCaptureByReplaykit:appGroup:	Start system-level screen sharing (for iOS 11.0 and above only)
startLocalAudio	Set sound quality
startRemoteView:view:	Start displaying remote video image
stopRemoteView:	Stop displaying remote video image and pulling the video data stream of remote user
setLocalViewFillMode:	Set the rendering mode of local image
setLocalViewRotation:	Set the clockwise rotation angle of local image
setLocalViewMirror:	Set the mirror mode of local camera's preview image

setRemoteViewFillMode:mode:	Set the fill mode of substream image
setRemoteViewRotation:rotation:	Set the clockwise rotation angle of remote image
startRemoteSubStreamView:view:	Start displaying the substream image of remote user
stopRemoteSubStreamView:	Stop displaying the substream image of remote user
setRemoteSubStreamViewFillMode:mode:	Set the fill mode of substream image
setRemoteSubStreamViewRotation:rotation:	Set the clockwise rotation angle of substream image
setAudioQuality:	Set sound quality
setPriorRemoteVideoStreamType:	Specify whether to view the big or small image
setMicVolumeOnMixing:	Set mic volume
playBGM:	Start background music
stopBGM	Stop background music
pauseBGM	Stop background music
resumeBGM	Stop background music
getBGMDuration:	Get the total length of background music in ms
setBGMPosition:	Set background music playback progress
setBGMVolume:	Set background music volume
setBGMPlayoutVolume:	Set the local playback volume of background music
setBGMPublishVolume:	Set the remote playback volume of background music
playAudioEffect:	Play sound effect
setAudioEffectVolume:volume:	Set sound effect volume
stopAudioEffect:	Stop sound effect

stopAllAudioEffects	Stop all sound effects
setAllAudioEffectsVolume:	Set the volume of all sound effects
pauseAudioEffect:	Pause sound effect
resumeAudioEffect:	Pause sound effect
enableCustomVideoCapture:	Enable custom video capturing mode
sendCustomVideoData:	Deliver captured video data to SDK
muteLocalVideo:	Pause/Resume publishing local video stream
muteRemoteVideoStream:mute:	Pause/Resume subscribing to remote user's video stream
startSpeedTest:userId:userSig:	Start network speed test (used before room entry)
startScreenCapture:	Start screen sharing
getCameraDevicesList	Get the list of cameras
setCurrentCameraDevice:	Set the camera to be used currently
getCurrentCameraDevice	Get the currently used camera
getMicDevicesList	Get the list of mics
getCurrentMicDevice	Get the current mic device
setCurrentMicDevice:	Select the currently used mic
getCurrentMicDeviceVolume	Get the current mic volume
setCurrentMicDeviceVolume:	Set the current mic volume
setCurrentMicDeviceMute:	Set the mute status of the current system mic
getCurrentMicDeviceMute	Get the mute status of the current system mic
getSpeakerDevicesList	Get the list of speakers
getCurrentSpeakerDevice	Get the currently used speaker
setCurrentSpeakerDevice:	Set the speaker to use

getCurrentSpeakerDeviceVolume	Get the current speaker volume
setCurrentSpeakerDeviceVolume:	Set the current speaker volume
getCurrentSpeakerDeviceMute	Get the mute status of the current system speaker
setCurrentSpeakerDeviceMute:	Set whether to mute the current system speaker
startCameraDeviceTestInView:	Start camera test
stopCameraDeviceTest	Start camera test
startMicDeviceTest:	Start mic test
stopMicDeviceTest	Start mic test
startSpeakerDeviceTest:	Start speaker test
stopSpeakerDeviceTest	Stop speaker test
startScreenCaptureInApp:	start in-app screen sharing (for iOS 13.0 and above only)
setVideoEncoderRotation:	Set the direction of image output by video encoder
setVideoEncoderMirror:	Set the mirror mode of image output by encoder
setGSensorMode:	Set the adaptation mode of G-sensor

destroySharedIntance

destroySharedIntance

Terminate TRTCCloud instance (singleton mode)

@deprecated This API is not recommended after 11.5 Please use destroySharedInstance instead.

delegate

delegate

Set TRTC event callback

@deprecated This API is not recommended after v11.4 Please use addDelegate instead.

setBeautyStyle:beautyLevel:whitenessLevel:ruddinessLevel:

setBeautyStyle:beautyLevel:whitenessLevel:ruddinessLevel:

- (void)setBeautyStyle:	(TRTCBeautyStyle)beautyStyle
beautyLevel:	(NSInteger)beautyLevel
whitenessLevel:	(NSInteger)whitenessLevel
ruddinessLevel:	(NSInteger)ruddinessLevel

Set the strength of beauty, brightening, and rosy skin filters.

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setEyeScaleLevel:

setEyeScaleLevel:

- (void)setEyeScaleLevel:	(float)eyeScaleLevel

Set the strength of eye enlarging filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setFaceScaleLevel:

setFaceScaleLevel:

- (void)setFaceScaleLevel:	(float)faceScaleLevel

Set the strength of face slimming filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setFaceVLevel:

setFaceVLevel:

- (void)setFaceVLevel:	(float)faceVLevel

Set the strength of chin slimming filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setChinLevel:

setChinLevel:

- (void)setChinLevel:	(float)chinLevel

Set the strength of chin lengthening/shortening filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setFaceShortLevel:

setFaceShortLevel:

- (void)setFaceShortLevel:	(float)faceShortlevel

Set the strength of face shortening filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setNoseSlimLevel:

setNoseSlimLevel:

- (void)setNoseSlimLevel:	(float)noseSlimLevel

Set the strength of nose slimming filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

selectMotionTmpl:

selectMotionTmpl:

- (void)selectMotionTmpl:	(NSString *)tmplPath

Set animated sticker

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setMotionMute:

setMotionMute:

- (void)setMotionMute:	(BOOL)motionMute

Mute animated sticker

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setFilter:

setFilter:

- (void)setFilter:	(TXImage *)image

Set color filter

@deprecated This API is not recommended after v7.2. Please use getBeautyManager instead.

setFilterConcentration:

setFilterConcentration:

- (void)setFilterConcentration:	(float)concentration

Set the strength of color filter

@deprecated This API is not recommended after v7.2. Please use getBeautyManager instead.

setGreenScreenFile:

setGreenScreenFile:

- (void)setGreenScreenFile:	(NSURL *)file

Set green screen video

@deprecated This API is not recommended after v7.2. Please use getBeautyManager instead.

setReverbType:

setReverbType:

- (void)setReverbType:	(TRTCReverbType)reverbType

Set reverb effect

@deprecated This API is not recommended after v7.3. Please use setVoiceReverbType API in TXAudioEffectManager instead.

setVoiceChangerType:

setVoiceChangerType:

- (void)setVoiceChangerType:	(TRTCVoiceChangerType)voiceChangerType

Set voice changing type

@deprecated This API is not recommended after v7.3. Please use setVoiceChangerType API in TXAudioEffectManager instead.

enableAudioEarMonitoring:

enableAudioEarMonitoring:

- (void)enableAudioEarMonitoring:	(BOOL)enable

Enable or disable in-ear monitoring

@deprecated This API is not recommended after v7.3. Please use setVoiceEarMonitor API in TXAudioEffectManager instead.

enableAudioVolumeEvaluation:

enableAudioVolumeEvaluation:

- (void)enableAudioVolumeEvaluation:	(NSUInteger)interval

Enable volume reminder

@deprecated This API is not recommended after v10.1. Please use enableAudioVolumeEvaluation(enable, params) instead.

enableAudioVolumeEvaluation:enable_vad:

enableAudioVolumeEvaluation:enable_vad:

- (void)enableAudioVolumeEvaluation:	(NSUInteger)interval
enable_vad:	(BOOL)enable_vad

Enable volume reminder

@deprecated This API is not recommended after v11.2. Please use enableAudioVolumeEvaluation(enable, params) instead.

switchCamera

switchCamera

Switch camera

@deprecated This API is not recommended after v8.0. Please use the switchCamera API in TXDeviceManager instead.

isCameraZoomSupported

isCameraZoomSupported

Query whether the current camera supports zoom

@deprecated This API is not recommended after v8.0. Please use the isCameraZoomSupported API in TXDeviceManager instead.

setZoom:

setZoom:

- (void)setZoom:	(CGFloat)distance

Set camera zoom ratio (focal length)

@deprecated This API is not recommended after v8.0. Please use the setCameraZoomRatio API in TXDeviceManager instead.

isCameraTorchSupported

isCameraTorchSupported

Query whether the device supports flash

@deprecated This API is not recommended after v8.0. Please use the isCameraTorchSupported API in TXDeviceManager instead.

enbaleTorch:

enbaleTorch:

- (BOOL)enbaleTorch:	(BOOL)enable

Enable/Disable flash

@deprecated This API is not recommended after v8.0. Please use the enableCameraTorch API in TXDeviceManager instead.

isCameraFocusPositionInPreviewSupported

isCameraFocusPositionInPreviewSupported

Query whether the camera supports setting focus

@deprecated This API is not recommended after v8.0.

setFocusPosition:

setFocusPosition:

- (void)setFocusPosition:	(CGPoint)touchPoint

Set the focal position of camera

@deprecated This API is not recommended after v8.0. Please use the setCameraFocusPosition API in TXDeviceManager instead.

isCameraAutoFocusFaceModeSupported

isCameraAutoFocusFaceModeSupported

Query whether the device supports the automatic recognition of face position

@deprecated This API is not recommended after v8.0. Please use the isAutoFocusEnabled API in TXDeviceManager instead.

enableAutoFaceFoucs:

enableAutoFaceFoucs:

- (void)enableAutoFaceFoucs:	(BOOL)enable

Enable/Disable face auto focus

@deprecated This API is not recommended after v8.0. Please use the enableCameraAutoFocus API in TXDeviceManager instead.

setSystemVolumeType:

setSystemVolumeType:

- (void)setSystemVolumeType:	(TRTCSystemVolumeType)type

Setting the system volume type (for mobile OS)

@deprecated This API is not recommended after v8.0. Please use the startLocalAudio instead, which param `quality` is used to decide audio quality.

snapshotVideo:type:

snapshotVideo:type:

- (void)snapshotVideo:	(NSString *)userId
type:	(TRTCVideoStreamType)streamType

Screencapture video

@deprecated This API is not recommended after v8.2. Please use snapshotVideo instead.

startScreenCaptureByReplaykit:appGroup:

startScreenCaptureByReplaykit:appGroup:

- (void)startScreenCaptureByReplaykit:	(TRTCVideoEncParam *)encParams
appGroup:	(NSString *)appGroup

Start system-level screen sharing (for iOS 11.0 and above only)

@deprecated This API is not recommended after v8.6. Please use startScreenCaptureByReplaykit instead.

startLocalAudio

startLocalAudio

Set sound quality

@deprecated This API is not recommended after v8.0. Please use startLocalAudio:quality instead.

startRemoteView:view:

startRemoteView:view:

- (void)startRemoteView:	(NSString *)userId

view:	(TXView *)view

Start displaying remote video image

@deprecated This API is not recommended after v8.0. Please use startRemoteView:streamType:view: instead.

stopRemoteView:

stopRemoteView:

- (void)stopRemoteView:	(NSString *)userId

Stop displaying remote video image and pulling the video data stream of remote user

@deprecated This API is not recommended after v8.0. Please use stopRemoteView:streamType: instead.

setLocalViewFillMode:

setLocalViewFillMode:

- (void)setLocalViewFillMode:	(TRTCVideoFillMode)mode

Set the rendering mode of local image

@deprecated This API is not recommended after v8.0. Please use setLocalRenderParams instead.

setLocalViewRotation:

setLocalViewRotation:

- (void)setLocalViewRotation:	(TRTCVideoRotation)rotation

Set the clockwise rotation angle of local image

@deprecated This API is not recommended after v8.0. Please use setLocalRenderParams instead.

setLocalViewMirror:

setLocalViewMirror:

- (void)setLocalViewMirror:	(TRTCLocalVideoMirrorType)mirror

Set the mirror mode of local camera's preview image

@deprecated This API is not recommended after v8.0. Please use setLocalRenderParams instead.

setRemoteViewFillMode:mode:

setRemoteViewFillMode:mode:

- (void)setRemoteViewFillMode:	(NSString*)userId
mode:	(TRTCVideoFillMode)mode

Set the fill mode of substream image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

setRemoteViewRotation:rotation:

setRemoteViewRotation:rotation:

- (void)setRemoteViewRotation:	(NSString*)userId
rotation:	(TRTCVideoRotation)rotation

Set the clockwise rotation angle of remote image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

startRemoteSubStreamView:view:

startRemoteSubStreamView:view:

- (void)startRemoteSubStreamView:	(NSString *)userId
view:	(TXView *)view

Start displaying the substream image of remote user

@deprecated This API is not recommended after v8.0. Please use startRemoteView:streamType:view: instead.

stopRemoteSubStreamView:

stopRemoteSubStreamView:

- (void)stopRemoteSubStreamView:	(NSString *)userId

Stop displaying the substream image of remote user

@deprecated This API is not recommended after v8.0. Please use stopRemoteView:streamType: instead.

setRemoteSubStreamViewFillMode:mode:

setRemoteSubStreamViewFillMode:mode:

- (void)setRemoteSubStreamViewFillMode:	(NSString *)userId
mode:	(TRTCVideoFillMode)mode

Set the fill mode of substream image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

setRemoteSubStreamViewRotation:rotation:

setRemoteSubStreamViewRotation:rotation:

- (void)setRemoteSubStreamViewRotation:	(NSString*)userId
rotation:	(TRTCVideoRotation)rotation

Set the clockwise rotation angle of substream image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

setAudioQuality:

setAudioQuality:

- (void)setAudioQuality:	(TRTCAudioQuality)quality

Set sound quality

@deprecated This API is not recommended after v8.0. Please use startLocalAudio:quality instead.

setPriorRemoteVideoStreamType:

setPriorRemoteVideoStreamType:

- (void)setPriorRemoteVideoStreamType:	(TRTCVideoStreamType)streamType

Specify whether to view the big or small image

@deprecated This API is not recommended after v8.0. Please use startRemoteView:streamType:view: instead.

setMicVolumeOnMixing:

setMicVolumeOnMixing:

- (void)setMicVolumeOnMixing:	(NSInteger)volume

Set mic volume

@deprecated This API is not recommended after v6.9. Please use setAudioCaptureVolume instead.

playBGM:

playBGM:

- (void) playBGM:	(NSString *)path

Start background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

stopBGM

stopBGM

Stop background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

pauseBGM

pauseBGM

Stop background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

resumeBGM

resumeBGM

Stop background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

getBGMDuration:

getBGMDuration:

- (NSInteger)getBGMDuration:	(NSString *)path

Get the total length of background music in ms

@deprecated This API is not recommended after v7.3. Please use getMusicDurationInMS API in TXAudioEffectManager instead.

setBGMPosition:

setBGMPosition:

- (int)setBGMPosition:	(NSInteger)pos

Set background music playback progress

@deprecated This API is not recommended after v7.3. Please use seekMusicToPosInMS API in TXAudioEffectManager instead.

setBGMVolume:

setBGMVolume:

- (void)setBGMVolume:	(NSInteger)volume

Set background music volume

@deprecated This API is not recommended after v7.3. Please use setMusicVolume API in TXAudioEffectManager instead.

setBGMPlayoutVolume:

setBGMPlayoutVolume:

- (void)setBGMPlayoutVolume:	(NSInteger)volume

Set the local playback volume of background music

@deprecated This API is not recommended after v7.3. Please use setMusicPlayoutVolume API in TXAudioEffectManager instead.

setBGMPublishVolume:

setBGMPublishVolume:

- (void)setBGMPublishVolume:	(NSInteger)volume

Set the remote playback volume of background music

@deprecated This API is not recommended after v7.3. Please use setBGMPublishVolume API in TXAudioEffectManager instead.

playAudioEffect:

playAudioEffect:

- (void)playAudioEffect:	(TRTCAudioEffectParam*)effect

Play sound effect

@deprecated This API is not recommended after v7.3. Please use startPlayMusic API in TXAudioEffectManager instead.

setAudioEffectVolume:volume:

setAudioEffectVolume:volume:

- (void)setAudioEffectVolume:	(int)effectId
volume:	(int) volume

Set sound effect volume

@deprecated This API is not recommended after v7.3. Please use setMusicPublishVolume and setMusicPlayoutVolume API in TXAudioEffectManager instead.

stopAudioEffect:

stopAudioEffect:

- (void)stopAudioEffect:	(int)effectId

Stop sound effect

@deprecated This API is not recommended after v7.3. Please use stopPlayMusic API in TXAudioEffectManager instead.

stopAllAudioEffects

stopAllAudioEffects

Stop all sound effects

@deprecated This API is not recommended after v7.3. Please use stopPlayMusic API in TXAudioEffectManager instead.

setAllAudioEffectsVolume:

setAllAudioEffectsVolume:

- (void)setAllAudioEffectsVolume:	(int)volume

Set the volume of all sound effects

@deprecated This API is not recommended after v7.3. Please use setMusicPublishVolume and setMusicPlayoutVolume API in TXAudioEffectManager instead.

pauseAudioEffect:

pauseAudioEffect:

- (void)pauseAudioEffect:	(int)effectId

Pause sound effect

@deprecated This API is not recommended after v7.3. Please use pauseAudioEffect API in TXAudioEffectManager instead.

resumeAudioEffect:

resumeAudioEffect:

- (void)resumeAudioEffect:	(int)effectId

Pause sound effect

@deprecated This API is not recommended after v7.3. Please use resumePlayMusic API in TXAudioEffectManager instead.

enableCustomVideoCapture:

enableCustomVideoCapture:

- (void)enableCustomVideoCapture:	(BOOL)enable

Enable custom video capturing mode

@deprecated This API is not recommended after v8.5. Please use enableCustomVideoCapture instead.

sendCustomVideoData:

sendCustomVideoData:

- (void)sendCustomVideoData:	(TRTCVideoFrame *)frame

Deliver captured video data to SDK

@deprecated This API is not recommended after v8.5. Please use sendCustomVideoData instead.

muteLocalVideo:

muteLocalVideo:

- (void)muteLocalVideo:	(BOOL)mute

Pause/Resume publishing local video stream

@deprecated This API is not recommended after v8.9. Please use muteLocalVideo (streamType, mute) instead.

muteRemoteVideoStream:mute:

muteRemoteVideoStream:mute:

- (void)muteRemoteVideoStream:	(NSString*)userId
mute:	(BOOL)mute

Pause/Resume subscribing to remote user's video stream

@deprecated This API is not recommended after v8.9. Please use muteRemoteVideoStream (userId, streamType, mute) instead.

startSpeedTest:userId:userSig:

startSpeedTest:userId:userSig:

- (void)startSpeedTest:	(uint32_t)sdkAppId

userId:	(NSString *)userId
userSig:	(NSString *)userSig

Start network speed test (used before room entry)

@deprecated This API is not recommended after v9.2. Please use startSpeedTest (params) instead.

startScreenCapture:

startScreenCapture:

- (void)startScreenCapture:	(nullable NSView *)view

Start screen sharing

@deprecated This API is not recommended after v7.2. Please use `startScreenCapture:streamType:encParam:` instead.

getCameraDevicesList

getCameraDevicesList

Get the list of cameras

@deprecated This API is not recommended after v8.0. Please use the getDevicesList API in TXDeviceManager instead.

setCurrentCameraDevice:

setCurrentCameraDevice:

- (int)setCurrentCameraDevice:	(NSString *)deviceId

Set the camera to be used currently

@deprecated This API is not recommended after v8.0. Please use the setCurrentDevice API in TXDeviceManager instead.

getCurrentCameraDevice

getCurrentCameraDevice

Get the currently used camera

@deprecated This API is not recommended after v8.0. Please use the getCurrentDevice API in TXDeviceManager instead.

getMicDevicesList

getMicDevicesList

Get the list of mics

@deprecated This API is not recommended after v8.0. Please use the getDevicesList API in TXDeviceManager instead.

getCurrentMicDevice

getCurrentMicDevice

Get the current mic device

@deprecated This API is not recommended after v8.0. Please use the getCurrentDevice API in TXDeviceManager instead.

setCurrentMicDevice:

setCurrentMicDevice:

- (int)setCurrentMicDevice:	(NSString*)deviceId

Select the currently used mic

@deprecated This API is not recommended after v8.0. Please use the setCurrentDevice API in TXDeviceManager instead.

getCurrentMicDeviceVolume

getCurrentMicDeviceVolume

Get the current mic volume

@deprecated This API is not recommended after v8.0. Please use the getCurrentDeviceVolume API in TXDeviceManager instead.

setCurrentMicDeviceVolume:

setCurrentMicDeviceVolume:

- (void)setCurrentMicDeviceVolume:	(NSInteger)volume

Set the current mic volume

@deprecated This API is not recommended after v8.0. Please use the setCurrentDeviceVolume API in TXDeviceManager instead.

setCurrentMicDeviceMute:

setCurrentMicDeviceMute:

- (void)setCurrentMicDeviceMute:	(BOOL)mute

Set the mute status of the current system mic

@deprecated This API is not recommended after v8.0. Please use the setCurrentDeviceMute API in TXDeviceManager instead.

getCurrentMicDeviceMute

getCurrentMicDeviceMute

Get the mute status of the current system mic

@deprecated This API is not recommended after v8.0. Please use the getCurrentDeviceMute API in TXDeviceManager instead.

getSpeakerDevicesList

getSpeakerDevicesList

Get the list of speakers

@deprecated This API is not recommended after v8.0. Please use the getDevicesList API in TXDeviceManager instead.

getCurrentSpeakerDevice

getCurrentSpeakerDevice

Get the currently used speaker

@deprecated This API is not recommended after v8.0. Please use the getCurrentDevice API in TXDeviceManager instead.

setCurrentSpeakerDevice:

setCurrentSpeakerDevice:

- (int)setCurrentSpeakerDevice:	(NSString*)deviceId

Set the speaker to use

@deprecated This API is not recommended after v8.0. Please use the setCurrentDevice API in TXDeviceManager instead.

getCurrentSpeakerDeviceVolume

getCurrentSpeakerDeviceVolume

Get the current speaker volume

@deprecated This API is not recommended after v8.0. Please use the getCurrentDeviceVolume API in TXDeviceManager instead.

setCurrentSpeakerDeviceVolume:

setCurrentSpeakerDeviceVolume:

- (int)setCurrentSpeakerDeviceVolume:	(NSInteger)volume

Set the current speaker volume

@deprecated This API is not recommended after v8.0. Please use the setCurrentDeviceVolume API in TXDeviceManager instead.

getCurrentSpeakerDeviceMute

getCurrentSpeakerDeviceMute

Get the mute status of the current system speaker

@deprecated This API is not recommended after v8.0. Please use the getCurrentDeviceMute API in TXDeviceManager instead.

setCurrentSpeakerDeviceMute:

setCurrentSpeakerDeviceMute:

- (void)setCurrentSpeakerDeviceMute:	(BOOL)mute

Set whether to mute the current system speaker

@deprecated This API is not recommended after v8.0. Please use the setCurrentDeviceMute API in TXDeviceManager instead.

startCameraDeviceTestInView:

startCameraDeviceTestInView:

- (void)startCameraDeviceTestInView:	(NSView *)view

Start camera test

@deprecated This API is not recommended after v8.0. Please use the startCameraDeviceTest API in TXDeviceManager instead.

stopCameraDeviceTest

stopCameraDeviceTest

Start camera test

@deprecated This API is not recommended after v8.0. Please use the stopCameraDeviceTest API in TXDeviceManager instead.

startMicDeviceTest:

startMicDeviceTest:

- (void)startMicDeviceTest:	(NSInteger)interval

Start mic test

@deprecated This API is not recommended after v8.0. Please use the startMicDeviceTest API in TXDeviceManager instead.

stopMicDeviceTest

stopMicDeviceTest

Start mic test

@deprecated This API is not recommended after v8.0. Please use the stopMicDeviceTest API in TXDeviceManager instead.

startSpeakerDeviceTest:

startSpeakerDeviceTest:

- (void)startSpeakerDeviceTest:	(NSString*)audioFilePath

Start speaker test

@deprecated This API is not recommended after v8.0. Please use the startSpeakerDeviceTest API in TXDeviceManager instead.

stopSpeakerDeviceTest

stopSpeakerDeviceTest

Stop speaker test

@deprecated This API is not recommended after v8.0. Please use the stopSpeakerDeviceTest API in TXDeviceManager instead.

startScreenCaptureInApp:

startScreenCaptureInApp:

- (void)startScreenCaptureInApp:	(TRTCVideoEncParam *)encParams

start in-app screen sharing (for iOS 13.0 and above only)

@deprecated This API is not recommended after v8.6. Please use startScreenCaptureInApp instead.

setVideoEncoderRotation:

setVideoEncoderRotation:

- (void)setVideoEncoderRotation:	(TRTCVideoRotation)rotation

Set the direction of image output by video encoder

@deprecated It is deprecated starting from v11.7.

setVideoEncoderMirror:

setVideoEncoderMirror:

- (void)setVideoEncoderMirror:	(BOOL)mirror

Set the mirror mode of image output by encoder

@deprecated It is deprecated starting from v11.7.

setGSensorMode:

setGSensorMode:

- (void)setGSensorMode:	(TRTCGSensorMode) mode

Set the adaptation mode of G-sensor

@deprecated It is deprecated starting from v11.7. It is recommended to use the setGravitySensorAdaptiveMode interface instead.

ErrorCode

Last updated：2024-03-07 15:33:58

Module: TRTC ErrorCode

Function: Used to notify customers of warnings and errors that occur during the use of TRTC

See All Platform C++ ErrorCode

Android

Overview

Last updated：2024-06-06 15:26:15

API OVERVIEW

Create Instance And Event Callback

FuncList	DESC
sharedInstance	Create TRTCCloud instance (singleton mode)
destroySharedInstance	Terminate TRTCCloud instance (singleton mode)
addListener	Add TRTC event callback
removeListener	Remove TRTC event callback
setListenerHandler	Set the queue that drives the TRTCCloudListener event callback

Room APIs

FuncList	DESC
enterRoom	Enter room
exitRoom	Exit room
switchRole	Switch role
switchRoom	Switch room
ConnectOtherRoom	Request cross-room call
DisconnectOtherRoom	Exit cross-room call
setDefaultStreamRecvMode	Set subscription mode (which must be set before room entry for it to take effect)
createSubCloud	Create room subinstance (for concurrent multi-room listen/watch)

destroySubCloud	Terminate room subinstance
updateOtherRoomForwardMode	

CDN APIs

FuncList	DESC
startPublishing	Start publishing audio/video streams to Tencent Cloud CSS CDN
stopPublishing	Stop publishing audio/video streams to Tencent Cloud CSS CDN
startPublishCDNStream	Start publishing audio/video streams to non-Tencent Cloud CDN
stopPublishCDNStream	Stop publishing audio/video streams to non-Tencent Cloud CDN
setMixTranscodingConfig	Set the layout and transcoding parameters of On-Cloud MixTranscoding
startPublishMediaStream	Publish a stream
updatePublishMediaStream	Modify publishing parameters
stopPublishMediaStream	Stop publishing

Video APIs

FuncList	DESC
startLocalPreview	Enable the preview image of local camera (mobile)
updateLocalView	Update the preview image of local camera
stopLocalPreview	Stop camera preview
muteLocalVideo	Pause/Resume publishing local video stream
setVideoMuteImage	Set placeholder image during local video pause
startRemoteView	Subscribe to remote user's video stream and bind video rendering control
updateRemoteView	Update remote user's video rendering control
stopRemoteView	Stop subscribing to remote user's video stream and release

	rendering control
stopAllRemoteView	Stop subscribing to all remote users' video streams and release all rendering resources
muteRemoteVideoStream	Pause/Resume subscribing to remote user's video stream
muteAllRemoteVideoStreams	Pause/Resume subscribing to all remote users' video streams
setVideoEncoderParam	Set the encoding parameters of video encoder
setNetworkQosParam	Set network quality control parameters
setLocalRenderParams	Set the rendering parameters of local video image
setRemoteRenderParams	Set the rendering mode of remote video image
enableEncSmallVideoStream	Enable dual-channel encoding mode with big and small images
setRemoteVideoStreamType	Switch the big/small image of specified remote user
snapshotVideo	Screencapture video
setPerspectiveCorrectionPoints	Sets perspective correction coordinate points.
setGravitySensorAdaptiveMode	Set the adaptation mode of gravity sensing (version 11.7 and above)

Audio APIs

FuncList	DESC
startLocalAudio	Enable local audio capturing and publishing
stopLocalAudio	Stop local audio capturing and publishing
muteLocalAudio	Pause/Resume publishing local audio stream
muteRemoteAudio	Pause/Resume playing back remote audio stream
muteAllRemoteAudio	Pause/Resume playing back all remote users' audio streams
setAudioRoute	Set audio route
setRemoteAudioVolume	Set the audio playback volume of remote user
setAudioCaptureVolume	Set the capturing volume of local audio

getAudioCaptureVolume	Get the capturing volume of local audio
setAudioPlayoutVolume	Set the playback volume of remote audio
getAudioPlayoutVolume	Get the playback volume of remote audio
enableAudioVolumeEvaluation	Enable volume reminder
startAudioRecording	Start audio recording
stopAudioRecording	Stop audio recording
startLocalRecording	Start local media recording
stopLocalRecording	Stop local media recording
setRemoteAudioParallelParams	Set the parallel strategy of remote audio streams
enable3DSpatialAudioEffect	Enable 3D spatial effect
updateSelf3DSpatialPosition	Update self position and orientation for 3D spatial effect
updateRemote3DSpatialPosition	Update the specified remote user's position for 3D spatial effect
set3DSpatialReceivingRange	Set the maximum 3D spatial attenuation range for userId's audio stream

Device management APIs

FuncList	DESC
getDeviceManager	Get device management class (TXDeviceManager)

Beauty filter and watermark APIs

FuncList	DESC
getBeautyManager	Get beauty filter management class (TXBeautyManager)
setWatermark	Add watermark

Background music and sound effect APIs

FuncList	DESC
getAudioEffectManager	Get sound effect management class (TXAudioEffectManager)
startSystemAudioLoopback	Enable system audio capturing
stopSystemAudioLoopback	Stop system audio capturing(iOS not supported)

Screen sharing APIs

FuncList	DESC
startScreenCapture	Start screen sharing
stopScreenCapture	Stop screen sharing
pauseScreenCapture	Pause screen sharing
resumeScreenCapture	Resume screen sharing
setSubStreamEncoderParam	Set the video encoding parameters of screen sharing (i.e., substream) (for desktop and mobile systems)

Custom capturing and rendering APIs

FuncList	DESC
enableCustomVideoCapture	Enable/Disable custom video capturing mode
sendCustomVideoData	Deliver captured video frames to SDK
enableCustomAudioCapture	Enable custom audio capturing mode
sendCustomAudioData	Deliver captured audio data to SDK
enableMixExternalAudioFrame	Enable/Disable custom audio track
mixExternalAudioFrame	Mix custom audio track into SDK
setMixExternalAudioVolume	Set the publish volume and playback volume of mixed custom audio track
generateCustomPTS	Generate custom capturing timestamp

setLocalVideoProcessListener	Set video data callback for third-party beauty filters
setLocalVideoRenderListener	Set the callback of custom rendering for local video
setRemoteVideoRenderListener	Set the callback of custom rendering for remote video
setAudioFrameListener	Set custom audio data callback
setCapturedAudioFrameCallbackFormat	Set the callback format of audio frames captured by local mic
setLocalProcessedAudioFrameCallbackFormat	Set the callback format of preprocessed local audio frames
setMixedPlayAudioFrameCallbackFormat	Set the callback format of audio frames to be played back by system
enableCustomAudioRendering	Enabling custom audio playback
getCustomAudioRenderingFrame	Getting playable audio data

Custom message sending APIs

FuncList	DESC
sendCustomCmdMsg	Use UDP channel to send custom message to all users in room
sendSEIMsg	Use SEI channel to send custom message to all users in room

Network test APIs

FuncList	DESC
startSpeedTest	Start network speed test (used before room entry)
stopSpeedTest	Stop network speed test

Debugging APIs

FuncList	DESC

getSDKVersion	Get SDK version information
setLogLevel	Set log output level
setConsoleEnabled	Enable/Disable console log printing
setLogCompressEnabled	Enable/Disable local log compression
setLogDirPath	Set local log storage path
setLogListener	Set log callback
showDebugView	Display dashboard
TRTCViewMargin	Set dashboard margin
callExperimentalAPI	Call experimental APIs

Encrypted interface

FuncList	DESC
enablePayloadPrivateEncryption	Enable or disable private encryption of media streams

Error and warning events

FuncList	DESC
onError	Error event callback
onWarning	Warning event callback

Room event callback

FuncList	DESC
onEnterRoom	Whether room entry is successful
onExitRoom	Room exit
onSwitchRole	Role switching

onSwitchRoom	Result of room switching
onConnectOtherRoom	Result of requesting cross-room call
onDisConnectOtherRoom	Result of ending cross-room call
onUpdateOtherRoomForwardMode	Result of changing the upstream capability of the cross-room anchor

User event callback

FuncList	DESC
onRemoteUserEnterRoom	A user entered the room
onRemoteUserLeaveRoom	A user exited the room
onUserVideoAvailable	A remote user published/unpublished primary stream video
onUserSubStreamAvailable	A remote user published/unpublished substream video
onUserAudioAvailable	A remote user published/unpublished audio
onFirstVideoFrame	The SDK started rendering the first video frame of the local or a remote user
onFirstAudioFrame	The SDK started playing the first audio frame of a remote user
onSendFirstLocalVideoFrame	The first local video frame was published
onSendFirstLocalAudioFrame	The first local audio frame was published
onRemoteVideoStatusUpdated	Change of remote video status
onRemoteAudioStatusUpdated	Change of remote audio status
onUserVideoSizeChanged	Change of remote video size

Callback of statistics on network and technical metrics

FuncList	DESC
onNetworkQuality	Real-time network quality statistics
onStatistics	Real-time statistics on technical metrics

onSpeedTestResult	Callback of network speed test

Callback of connection to the cloud

FuncList	DESC
onConnectionLost	The SDK was disconnected from the cloud
onTryToReconnect	The SDK is reconnecting to the cloud
onConnectionRecovery	The SDK is reconnected to the cloud

Callback of hardware events

FuncList	DESC
onCameraDidReady	The camera is ready
onMicDidReady	The mic is ready
onAudioRouteChanged	The audio route changed (for mobile devices only)
onUserVoiceVolume	Volume

Callback of the receipt of a custom message

FuncList	DESC
onRecvCustomCmdMsg	Receipt of custom message
onMissCustomCmdMsg	Loss of custom message
onRecvSEIMsg	Receipt of SEI message

CDN event callback

FuncList	DESC

onStartPublishing	Started publishing to Tencent Cloud CSS CDN
onStopPublishing	Stopped publishing to Tencent Cloud CSS CDN
onStartPublishCDNStream	Started publishing to non-Tencent Cloud’s live streaming CDN
onStopPublishCDNStream	Stopped publishing to non-Tencent Cloud’s live streaming CDN
onSetMixTranscodingConfig	Set the layout and transcoding parameters for On-Cloud MixTranscoding
onStartPublishMediaStream	Callback for starting to publish
onUpdatePublishMediaStream	Callback for modifying publishing parameters
onStopPublishMediaStream	Callback for stopping publishing
onCdnStreamStateChanged	Callback for change of RTMP/RTMPS publishing status

Screen sharing event callback

FuncList	DESC
onScreenCaptureStarted	Screen sharing started
onScreenCapturePaused	Screen sharing was paused
onScreenCaptureResumed	Screen sharing was resumed
onScreenCaptureStopped	Screen sharing stopped

Callback of local recording and screenshot events

FuncList	DESC
onLocalRecordBegin	Local recording started
onLocalRecording	Local media is being recorded
onLocalRecordFragment	Record fragment finished.
onLocalRecordComplete	Local recording stopped
onSnapshotComplete	Finished taking a local screenshot

Disused callbacks

FuncList	DESC
onUserEnter	An anchor entered the room (disused)
onUserExit	An anchor left the room (disused)
onAudioEffectFinished	Audio effects ended (disused)
onSpeedTest	Result of server speed testing (disused)

Callback of custom video processing

FuncList	DESC
onRenderVideoFrame	Custom video rendering
onGLContextCreated	An OpenGL context was created in the SDK.
onProcessVideoFrame	Video processing by third-party beauty filters
onGLContextDestory	The OpenGL context in the SDK was destroyed

Callback of custom audio processing

FuncList	DESC
onCapturedAudioFrame	Audio data captured by the local mic and pre-processed by the audio module
onLocalProcessedAudioFrame	Audio data captured by the local mic, pre-processed by the audio module, effect-processed and BGM-mixed
onRemoteUserAudioFrame	Audio data of each remote user before audio mixing
onMixedPlayAudioFrame	Data mixed from each channel before being submitted to the system for playback
onMixedAllAudioFrame	Data mixed from all the captured and to-be-played audio in the SDK

onVoiceEarMonitorAudioFrame	In-ear monitoring data

Other event callbacks

FuncList	DESC
onLog	Printing of local log

Background music preload event callback

FuncList	DESC
onLoadProgress	Background music preload progress
onLoadError	Background music preload error

Callback of playing background music

FuncList	DESC
onStart	Background music started.
onPlayProgress	Playback progress of background music
onComplete	Background music ended

Voice effect APIs

FuncList	DESC
enableVoiceEarMonitor	Enabling in-ear monitoring
setVoiceEarMonitorVolume	Setting in-ear monitoring volume
setVoiceReverbType	Setting voice reverb effects
setVoiceChangerType	Setting voice changing effects

setVoiceCaptureVolume	Setting speech volume
setVoicePitch	Setting speech pitch

Background music APIs

FuncList	DESC
setMusicObserver	Setting the background music callback
startPlayMusic	Starting background music
stopPlayMusic	Stopping background music
pausePlayMusic	Pausing background music
resumePlayMusic	Resuming background music
setAllMusicVolume	Setting the local and remote playback volume of background music
setMusicPublishVolume	Setting the remote playback volume of a specific music track
setMusicPlayoutVolume	Setting the local playback volume of a specific music track
setMusicPitch	Adjusting the pitch of background music
setMusicSpeedRate	Changing the speed of background music
getMusicCurrentPosInMS	Getting the playback progress (ms) of background music
getMusicDurationInMS	Getting the total length (ms) of background music
seekMusicToPosInMS	Setting the playback progress (ms) of background music
setMusicScratchSpeedRate	Adjust the speed change effect of the scratch disc
setPreloadObserver	Setting music preload callback
preloadMusic	Preload background music
getMusicTrackCount	Get the number of tracks of background music
setMusicTrack	Specify the playback track of background music

beauty interface

FuncList	DESC
setBeautyStyle	Sets the beauty (skin smoothing) filter algorithm.
setBeautyLevel	Sets the strength of the beauty filter.
setWhitenessLevel	Sets the strength of the brightening filter.
enableSharpnessEnhancement	Enables clarity enhancement.
setRuddyLevel	Sets the strength of the rosy skin filter.
setFilter	Sets color filter.
setFilterStrength	Sets the strength of color filter.
setGreenScreenFile	Sets green screen video
setEyeScaleLevel	Sets the strength of the eye enlarging filter.
setFaceSlimLevel	Sets the strength of the face slimming filter.
setFaceVLevel	Sets the strength of the chin slimming filter.
setChinLevel	Sets the strength of the chin lengthening/shortening filter.
setFaceShortLevel	Sets the strength of the face shortening filter.
setFaceNarrowLevel	Sets the strength of the face narrowing filter.
setNoseSlimLevel	Sets the strength of the nose slimming filter.
setEyeLightenLevel	Sets the strength of the eye brightening filter.
setToothWhitenLevel	Sets the strength of the teeth whitening filter.
setWrinkleRemoveLevel	Sets the strength of the wrinkle removal filter.
setPounchRemoveLevel	Sets the strength of the eye bag removal filter.
setSmileLinesRemoveLevel	Sets the strength of the smile line removal filter.
setForeheadLevel	Sets the strength of the hairline adjustment filter.
setEyeDistanceLevel	Sets the strength of the eye distance adjustment filter.
setEyeAngleLevel	Sets the strength of the eye corner adjustment filter.
setMouthShapeLevel	Sets the strength of the mouth shape adjustment filter.

setNoseWingLevel	Sets the strength of the nose wing narrowing filter.
setNosePositionLevel	Sets the strength of the nose position adjustment filter.
setLipsThicknessLevel	Sets the strength of the lip thickness adjustment filter.
setFaceBeautyLevel	Sets the strength of the face shape adjustment filter.
setMotionTmpl	Selects the AI animated effect pendant.
setMotionMute	Sets whether to mute during animated effect playback.

Device APIs

FuncList	DESC
isFrontCamera	Querying whether the front camera is being used
switchCamera	Switching to the front/rear camera (for mobile OS)
getCameraZoomMaxRatio	Getting the maximum zoom ratio of the camera (for mobile OS)
setCameraZoomRatio	Setting the camera zoom ratio (for mobile OS)
isAutoFocusEnabled	Querying whether automatic face detection is supported (for mobile OS)
enableCameraAutoFocus	Enabling auto focus (for mobile OS)
setCameraFocusPosition	Adjusting the focus (for mobile OS)
enableCameraTorch	Enabling/Disabling flash, i.e., the torch mode (for mobile OS)
setAudioRoute	Setting the audio route (for mobile OS)
setExposureCompensation	Set the exposure parameters of the camera, ranging from - 1 to 1
setCameraCapturerParam	Set camera acquisition preferences

Disused APIs

FuncList	DESC
setSystemVolumeType	Setting the system volume type (for mobile OS)

Disused APIs

FuncList	DESC
setListener	Set TRTC event callback
setBeautyStyle	Set the strength of beauty, brightening, and rosy skin filters.
setEyeScaleLevel	Set the strength of eye enlarging filter
setFaceSlimLevel	Set the strength of face slimming filter
setFaceVLevel	Set the strength of chin slimming filter
setChinLevel	Set the strength of chin lengthening/shortening filter
setFaceShortLevel	Set the strength of face shortening filter
setNoseSlimLevel	Set the strength of nose slimming filter
selectMotionTmpl	Set animated sticker
setMotionMute	Mute animated sticker
setFilter	Set color filter
setFilterConcentration	Set the strength of color filter
setGreenScreenFile	Set green screen video
setReverbType	Set reverb effect
setVoiceChangerType	Set voice changing type
enableAudioEarMonitoring	Enable or disable in-ear monitoring
enableAudioVolumeEvaluation	Enable volume reminder
switchCamera	Switch camera
isCameraZoomSupported	Query whether the current camera supports zoom
setZoom	Set camera zoom ratio (focal length)
isCameraTorchSupported	Query whether the device supports flash
enableTorch	Enable/Disable flash

isCameraFocusPositionInPreviewSupported	Query whether the camera supports setting focus
setFocusPosition	Set the focal position of camera
isCameraAutoFocusFaceModeSupported	Query whether the device supports the automatic recognition of face position
setSystemVolumeType	Setting the system volume type (for mobile OS)
checkAudioCapabilitySupport	Query whether a certain audio capability is supported (only for Android)
startLocalAudio	Set sound quality
startRemoteView	Start displaying remote video image
stopRemoteView	Stop displaying remote video image and pulling the video data stream of remote user
setLocalViewFillMode	Set the rendering mode of local image
setLocalViewRotation	Set the clockwise rotation angle of local image
setLocalViewMirror	Set the mirror mode of local camera's preview image
setRemoteViewFillMode	Set the fill mode of substream image
setRemoteViewRotation	Set the clockwise rotation angle of remote image
startRemoteSubStreamView	Start displaying the substream image of remote user
stopRemoteSubStreamView	Stop displaying the substream image of remote user
setRemoteSubStreamViewFillMode	Set the fill mode of substream image
setRemoteSubStreamViewRotation	Set the clockwise rotation angle of substream image
setAudioQuality	Set sound quality
setPriorRemoteVideoStreamType	Specify whether to view the big or small image
setMicVolumeOnMixing	Set mic volume
playBGM	Start background music
stopBGM	Stop background music
pauseBGM	Stop background music
resumeBGM	Stop background music

getBGMDuration	Get the total length of background music in ms
setBGMPosition	Set background music playback progress
setBGMVolume	Set background music volume
setBGMPlayoutVolume	Set the local playback volume of background music
setBGMPublishVolume	Set the remote playback volume of background music
playAudioEffect	Play sound effect
setAudioEffectVolume	Set sound effect volume
stopAudioEffect	Stop sound effect
stopAllAudioEffects	Stop all sound effects
setAllAudioEffectsVolume	Set the volume of all sound effects
pauseAudioEffect	Pause sound effect
resumeAudioEffect	Pause sound effect
enableCustomVideoCapture	Enable custom video capturing mode
sendCustomVideoData	Deliver captured video data to SDK
muteLocalVideo	Pause/Resume publishing local video stream
muteRemoteVideoStream	Pause/Resume subscribing to remote user's video stream
snapshotVideo	Screencapture video
startSpeedTest	Start network speed test (used before room entry)
startScreenCapture	Start screen sharing
setVideoEncoderRotation	Set the direction of image output by video encoder
setVideoEncoderMirror	Set the mirror mode of image output by encoder
setGSensorMode	Set the adaptation mode of G-sensor

TRTCCloud

Last updated：2024-06-06 15:26:15

Module: TRTCCloud @ TXLiteAVSDK

Function: TRTC's main feature API

Version: 11.9

TRTCCloud

TRTCCloud

FuncList	DESC
sharedInstance	Create TRTCCloud instance (singleton mode)
destroySharedInstance	Terminate TRTCCloud instance (singleton mode)
addListener	Add TRTC event callback
removeListener	Remove TRTC event callback
setListenerHandler	Set the queue that drives the TRTCCloudListener event callback
enterRoom	Enter room
exitRoom	Exit room
switchRole	Switch role
switchRole	Switch role(support permission credential)
switchRoom	Switch room
ConnectOtherRoom	Request cross-room call
DisconnectOtherRoom	Exit cross-room call

setDefaultStreamRecvMode	Set subscription mode (which must be set before room entry for it to take effect)
createSubCloud	Create room subinstance (for concurrent multi-room listen/watch)
destroySubCloud	Terminate room subinstance
updateOtherRoomForwardMode	
startPublishing	Start publishing audio/video streams to Tencent Cloud CSS CDN
stopPublishing	Stop publishing audio/video streams to Tencent Cloud CSS CDN
startPublishCDNStream	Start publishing audio/video streams to non-Tencent Cloud CDN
stopPublishCDNStream	Stop publishing audio/video streams to non-Tencent Cloud CDN
setMixTranscodingConfig	Set the layout and transcoding parameters of On-Cloud MixTranscoding
startPublishMediaStream	Publish a stream
updatePublishMediaStream	Modify publishing parameters
stopPublishMediaStream	Stop publishing
startLocalPreview	Enable the preview image of local camera (mobile)
updateLocalView	Update the preview image of local camera
stopLocalPreview	Stop camera preview
muteLocalVideo	Pause/Resume publishing local video stream
setVideoMuteImage	Set placeholder image during local video pause
startRemoteView	Subscribe to remote user's video stream and bind video rendering control
updateRemoteView	Update remote user's video rendering control
stopRemoteView	Stop subscribing to remote user's video stream and release rendering control

stopAllRemoteView	Stop subscribing to all remote users' video streams and release all rendering resources
muteRemoteVideoStream	Pause/Resume subscribing to remote user's video stream
muteAllRemoteVideoStreams	Pause/Resume subscribing to all remote users' video streams
setVideoEncoderParam	Set the encoding parameters of video encoder
setNetworkQosParam	Set network quality control parameters
setLocalRenderParams	Set the rendering parameters of local video image
setRemoteRenderParams	Set the rendering mode of remote video image
enableEncSmallVideoStream	Enable dual-channel encoding mode with big and small images
setRemoteVideoStreamType	Switch the big/small image of specified remote user
snapshotVideo	Screencapture video
setPerspectiveCorrectionPoints	Sets perspective correction coordinate points.
setGravitySensorAdaptiveMode	Set the adaptation mode of gravity sensing (version 11.7 and above)
startLocalAudio	Enable local audio capturing and publishing
stopLocalAudio	Stop local audio capturing and publishing
muteLocalAudio	Pause/Resume publishing local audio stream
muteRemoteAudio	Pause/Resume playing back remote audio stream
muteAllRemoteAudio	Pause/Resume playing back all remote users' audio streams
setAudioRoute	Set audio route
setRemoteAudioVolume	Set the audio playback volume of remote user
setAudioCaptureVolume	Set the capturing volume of local audio
getAudioCaptureVolume	Get the capturing volume of local audio
setAudioPlayoutVolume	Set the playback volume of remote audio

getAudioPlayoutVolume	Get the playback volume of remote audio
enableAudioVolumeEvaluation	Enable volume reminder
startAudioRecording	Start audio recording
stopAudioRecording	Stop audio recording
startLocalRecording	Start local media recording
stopLocalRecording	Stop local media recording
setRemoteAudioParallelParams	Set the parallel strategy of remote audio streams
enable3DSpatialAudioEffect	Enable 3D spatial effect
updateSelf3DSpatialPosition	Update self position and orientation for 3D spatial effect
updateRemote3DSpatialPosition	Update the specified remote user's position for 3D spatial effect
set3DSpatialReceivingRange	Set the maximum 3D spatial attenuation range for userId's audio stream
getDeviceManager	Get device management class (TXDeviceManager)
getBeautyManager	Get beauty filter management class (TXBeautyManager)
setWatermark	Add watermark
getAudioEffectManager	Get sound effect management class (TXAudioEffectManager)
startSystemAudioLoopback	Enable system audio capturing
stopSystemAudioLoopback	Stop system audio capturing(iOS not supported)
startScreenCapture	Start screen sharing
stopScreenCapture	Stop screen sharing
pauseScreenCapture	Pause screen sharing
resumeScreenCapture	Resume screen sharing
setSubStreamEncoderParam	Set the video encoding parameters of screen sharing (i.e., substream) (for desktop and mobile systems)

enableCustomVideoCapture	Enable/Disable custom video capturing mode
sendCustomVideoData	Deliver captured video frames to SDK
enableCustomAudioCapture	Enable custom audio capturing mode
sendCustomAudioData	Deliver captured audio data to SDK
enableMixExternalAudioFrame	Enable/Disable custom audio track
mixExternalAudioFrame	Mix custom audio track into SDK
setMixExternalAudioVolume	Set the publish volume and playback volume of mixed custom audio track
generateCustomPTS	Generate custom capturing timestamp
setLocalVideoProcessListener	Set video data callback for third-party beauty filters
setLocalVideoRenderListener	Set the callback of custom rendering for local video
setRemoteVideoRenderListener	Set the callback of custom rendering for remote video
setAudioFrameListener	Set custom audio data callback
setCapturedAudioFrameCallbackFormat	Set the callback format of audio frames captured by local mic
setLocalProcessedAudioFrameCallbackFormat	Set the callback format of preprocessed local audio frames
setMixedPlayAudioFrameCallbackFormat	Set the callback format of audio frames to be played back by system
enableCustomAudioRendering	Enabling custom audio playback
getCustomAudioRenderingFrame	Getting playable audio data
sendCustomCmdMsg	Use UDP channel to send custom message to all users in room
sendSEIMsg	Use SEI channel to send custom message to all users in room
startSpeedTest	Start network speed test (used before room entry)
stopSpeedTest	Stop network speed test
getSDKVersion	Get SDK version information

setLogLevel	Set log output level
setConsoleEnabled	Enable/Disable console log printing
setLogCompressEnabled	Enable/Disable local log compression
setLogDirPath	Set local log storage path
setLogListener	Set log callback
showDebugView	Display dashboard
TRTCViewMargin	Set dashboard margin
callExperimentalAPI	Call experimental APIs
enablePayloadPrivateEncryption	Enable or disable private encryption of media streams

sharedInstance

sharedInstance

TRTCCloud sharedInstance	(Context context)

Create TRTCCloud instance (singleton mode)

Param	DESC
context	It is only applicable to the Android platform. The SDK internally converts it into the `ApplicationContext` of Android to call the Android system API.

Note

1. If you use `delete ITRTCCloud*` , a compilation error will occur. Please use `destroyTRTCCloud` to release the object pointer.
2. On Windows, macOS, or iOS, please call the `getTRTCShareInstance()` API.
3. On Android, please call the `getTRTCShareInstance(void *context)` API.

destroySharedInstance

destroySharedInstance

Terminate TRTCCloud instance (singleton mode)

addListener

addListener

void addListener	(TRTCCloudListener listener)

Add TRTC event callback

You can use TRTCCloudListener to get various event notifications from the SDK, such as error codes, warning codes, and audio/video status parameters.

removeListener

removeListener

void removeListener	(TRTCCloudListener listener)

Remove TRTC event callback

setListenerHandler

setListenerHandler

void setListenerHandler	(Handler listenerHandler)

Set the queue that drives the TRTCCloudListener event callback

If you do not specify a `listenerHandler` , the SDK will use `MainQueue` as the queue for driving TRTCCloudListener event callbacks by default.
In other words, if you do not set the `listenerHandler` attribute, all callback functions in TRTCCloudListener will be driven by `MainQueue` .

Param	DESC
listenerHandler	

Note

If you specify a `listenerHandler` , please do not manipulate the UI in the TRTCCloudListener callback function; otherwise, thread safety issues will occur.

enterRoom

enterRoom

void enterRoom	(TRTCCloudDef.TRTCParams param
	int scene)

Enter room

All TRTC users need to enter a room before they can "publish" or "subscribe to" audio/video streams. "Publishing" refers to pushing their own streams to the cloud, and "subscribing to" refers to pulling the streams of other users in the room from the cloud.

When calling this API, you need to specify your application scenario (TRTCAppScene) to get the best audio/video transfer experience. We provide the following four scenarios for your choice:
TRTC_APP_SCENE_VIDEOCALL:
Video call scenario. Use cases: [one-to-one video call], [video conferencing with up to 300 participants], [online medical diagnosis], [small class], [video interview], etc.
In this scenario, each room supports up to 300 concurrent online users, and up to 50 of them can speak simultaneously.
TRTC_APP_SCENE_AUDIOCALL:
Audio call scenario. Use cases: [one-to-one audio call], [audio conferencing with up to 300 participants], [audio chat], [online Werewolf], etc.
In this scenario, each room supports up to 300 concurrent online users, and up to 50 of them can speak simultaneously.
TRTC_APP_SCENE_LIVE:
Live streaming scenario. Use cases: [low-latency video live streaming], [interactive classroom for up to 100,000 participants], [live video competition], [video dating room], [remote training], [large-scale conferencing], etc.
In this scenario, each room supports up to 100,000 concurrent online users, but you should specify the user roles: anchor (TRTCRoleAnchor) or audience (TRTCRoleAudience).
TRTC_APP_SCENE_VOICE_CHATROOM:
Audio chat room scenario. Use cases: [Clubhouse], [online karaoke room], [music live room], [FM radio], etc.
In this scenario, each room supports up to 100,000 concurrent online users, but you should specify the user roles: anchor (TRTCRoleAnchor) or audience (TRTCRoleAudience).

After calling this API, you will receive the `onEnterRoom(result)` callback from TRTCCloudListener:
If room entry succeeded, the `result` parameter will be a positive number (`result` > 0), indicating the time in milliseconds (ms) between function call and room entry.

If room entry failed, the `result` parameter will be a negative number (`result` < 0), indicating the TXLiteAVError for room entry failure.

Param	DESC
param	Room entry parameter, which is used to specify the user's identity, role, authentication credentials, and other information. For more information, please see TRTCParams.
scene	Application scenario, which is used to specify the use case. The same TRTCAppScene should be configured for all users in the same room.

Note

1. If `scene` is specified as TRTCAppSceneLIVE or TRTCAppSceneVoiceChatRoom, you must use the `role` field in TRTCParams to specify the role of the current user in the room.
2. The same `scene` should be configured for all users in the same room.
3. Please try to ensure that enterRoom and exitRoom are used in pair; that is, please make sure that "the previous room is exited before the next room is entered"; otherwise, many issues may occur.

exitRoom

exitRoom

Exit room

Calling this API will allow the user to leave the current audio or video room and release the camera, mic, speaker, and other device resources.
After resources are released, the SDK will use the `onExitRoom()` callback in TRTCCloudListener to notify you.

If you need to call enterRoom again or switch to the SDK of another provider, we recommend you wait until you receive the `onExitRoom()` callback, so as to avoid the problem of the camera or mic being occupied.

switchRole

switchRole

void switchRole	(int role)

Switch role

This API is used to switch the user role between `anchor` and `audience` .

As video live rooms and audio chat rooms need to support an audience of up to 100,000 concurrent online users, the rule "only anchors can publish their audio/video streams" has been set. Therefore, when some users want to publish their streams (so that they can interact with anchors), they need to switch their role to "anchor" first.

You can use the `role` field in TRTCParams during room entry to specify the user role in advance or use the `switchRole` API to switch roles after room entry.

Param	DESC
role	Role, which is `anchor` by default: TRTCRoleAnchor: anchor, who can publish their audio/video streams. Up to 50 anchors are allowed to publish streams at the same time in one room. TRTCRoleAudience: audience, who cannot publish their audio/video streams, but can only watch streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room supports an audience of up to 100,000 concurrent online users.

Note

1. This API is only applicable to two scenarios: live streaming (TRTC_APP_SCENE_LIVE) and audio chat room (TRTC_APP_SCENE_VOICE_CHATROOM).
2. If the `scene` you specify in enterRoom is TRTC_APP_SCENE_VIDEOCALL or TRTC_APP_SCENE_AUDIOCALL, please do not call this API.

switchRole

switchRole

void switchRole	(int role
	final String privateMapKey)

Switch role(support permission credential)

This API is used to switch the user role between `anchor` and `audience`.

As video live rooms and audio chat rooms need to support an audience of up to 100,000 concurrent online users, the rule "only anchors can publish their audio/video streams" has been set. Therefore, when some users want to publish their streams (so that they can interact with anchors), they need to switch their role to "anchor" first.

You can use the `role` field in TRTCParams during room entry to specify the user role in advance or use the `switchRole` API to switch roles after room entry.

Param	DESC
privateMapKey	Permission credential used for permission control. If you want only users with the specified `userId` values to enter a room or push streams, you need to use `privateMapKey` to restrict the permission. We recommend you use this parameter only if you have high security requirements. For more information, please see Enabling Advanced Permission Control.
role	Role, which is `anchor` by default: TRTCRoleAnchor: anchor, who can publish their audio/video streams. Up to 50 anchors are allowed to publish streams at the same time in one room. TRTCRoleAudience: audience, who cannot publish their audio/video streams, but can only watch streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room supports an audience of up to 100,000 concurrent online users.

Note

1. This API is only applicable to two scenarios: live streaming (TRTCAppSceneLIVE) and audio chat room (TRTCAppSceneVoiceChatRoom).
2. If the `scene` you specify in enterRoom is TRTCAppSceneVideoCall or TRTCAppSceneAudioCall, please do not call this API.

switchRoom

switchRoom

void switchRoom	(final TRTCCloudDef.TRTCSwitchRoomConfig config)

Switch room

This API is used to quickly switch a user from one room to another.
If the user's role is `audience`, calling this API is equivalent to `exitRoom` (current room) + `enterRoom` (new room).
If the user's role is `anchor`, the API will retain the current audio/video publishing status while switching the room; therefore, during the room switch, camera preview and sound capturing will not be interrupted.

This API is suitable for the online education scenario where the supervising teacher can perform fast room switch across multiple rooms. In this scenario, using `switchRoom` can get better smoothness and use less code than

`exitRoom + enterRoom` .
The API call result will be called back through `onSwitchRoom(errCode, errMsg)` in TRTCCloudListener.

Param	DESC
config	Room parameter. For more information, please see TRTCSwitchRoomConfig.

Note

Due to the requirement for compatibility with legacy versions of the SDK, the `config` parameter contains both `roomId` and `strRoomId` parameters. You should pay special attention as detailed below when specifying these two parameters:
1. If you decide to use `strRoomId` , then set `roomId` to 0. If both are specified, `roomId` will be used.
2. All rooms need to use either `strRoomId` or `roomId` at the same time. They cannot be mixed; otherwise, there will be many unexpected bugs.

ConnectOtherRoom

ConnectOtherRoom

void ConnectOtherRoom	(String param)

Request cross-room call

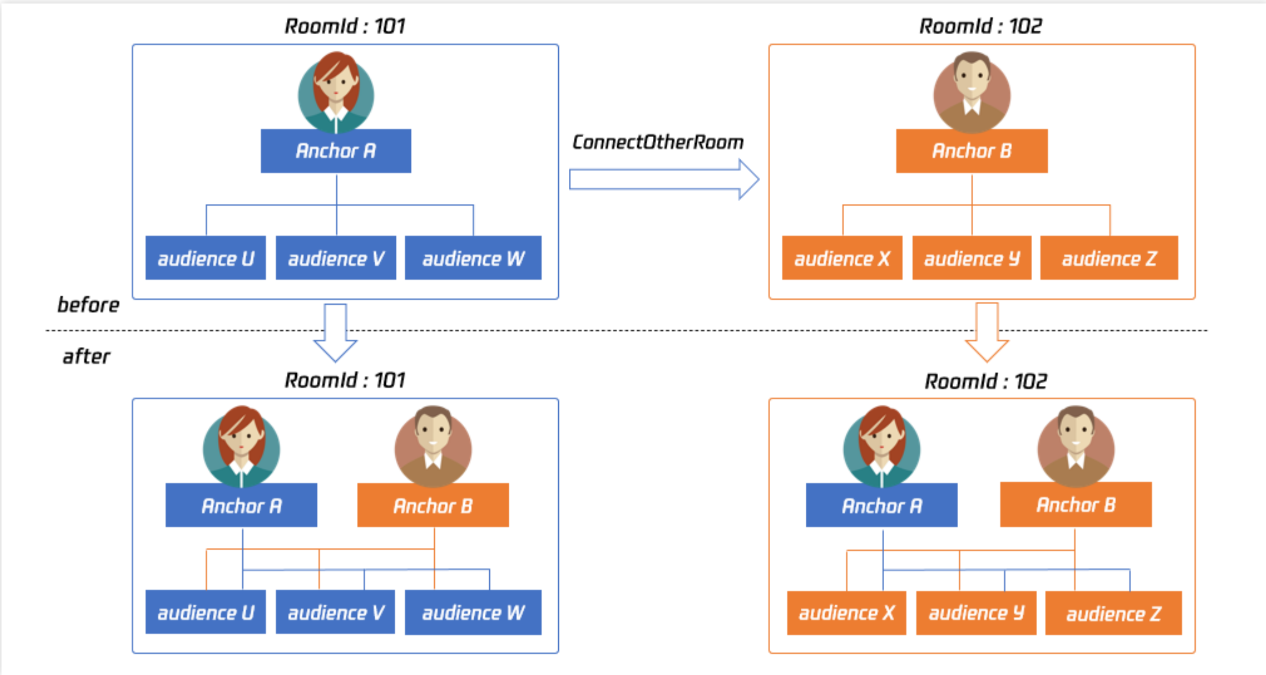
By default, only users in the same room can make audio/video calls with each other, and the audio/video streams in different rooms are isolated from each other.
However, you can publish the audio/video streams of an anchor in another room to the current room by calling this API. At the same time, this API will also publish the local audio/video streams to the target anchor's room.

In other words, you can use this API to share the audio/video streams of two anchors in two different rooms, so that the audience in each room can watch the streams of these two anchors. This feature can be used to implement anchor competition.

The result of requesting cross-room call will be returned through the onConnectOtherRoom callback in TRTCCloudDelegate.

For example, after anchor A in room "101" uses `connectOtherRoom()` to successfully call anchor B in room "102":
All users in room "101" will receive the `onRemoteUserEnterRoom(B)` and `onUserVideoAvailable(B,true)` event callbacks of anchor B; that is, all users in room "101" can subscribe to

the audio/video streams of anchor B.
All users in room "102" will receive the `onRemoteUserEnterRoom(A)` and `onUserVideoAvailable(A,true)` event callbacks of anchor A; that is, all users in room "102" can subscribe to the audio/video streams of anchor A.



For compatibility with subsequent extended fields for cross-room call, parameters in JSON format are used currently.

Case 1: numeric room ID
If anchor A in room "101" wants to co-anchor with anchor B in room "102", then anchor A needs to pass in {"roomId": 102, "userId": "userB"} when calling this API.
Below is the sample code:


```
JSONObject jsonObj = new JSONObject();
jsonObj.put("roomId", 102);
jsonObj.put("userId", "userB");
trtc.ConnectOtherRoom(jsonObj.toString());
```

Case 2: string room ID

If you use a string room ID, please be sure to replace the `roomId` in JSON with `strRoomId` , such as {"strRoomId": "102", "userId": "userB"}

Below is the sample code:

```
JSONObject jsonObj = new JSONObject();
jsonObj.put("strRoomId", "102");
jsonObj.put("userId", "userB");
trtc.ConnectOtherRoom(jsonObj.toString());
```

Param	DESC
param	You need to pass in a string parameter in JSON format: `roomId` represents the room ID in numeric format, `strRoomId` represents the room ID in string format, and `userId` represents the user ID of the target anchor.

DisconnectOtherRoom

DisconnectOtherRoom

Exit cross-room call

The result will be returned through the `onDisconnectOtherRoom()` callback in TRTCCloudDelegate.

setDefaultStreamRecvMode

setDefaultStreamRecvMode

void setDefaultStreamRecvMode	(boolean autoRecvAudio
	boolean autoRecvVideo)

Set subscription mode (which must be set before room entry for it to take effect)

You can switch between the "automatic subscription" and "manual subscription" modes through this API:
Automatic subscription: this is the default mode, where the user will immediately receive the audio/video streams in the room after room entry, so that the audio will be automatically played back, and the video will be automatically decoded (you still need to bind the rendering control through the `startRemoteView` API).
Manual subscription: after room entry, the user needs to manually call the startRemoteView API to start subscribing to and decoding the video stream and call the muteRemoteAudio (false) API to start playing back the audio stream.

In most scenarios, users will subscribe to the audio/video streams of all anchors in the room after room entry. Therefore, TRTC adopts the automatic subscription mode by default in order to achieve the best "instant streaming experience".
In your application scenario, if there are many audio/video streams being published at the same time in each room, and each user only wants to subscribe to 1–2 streams of them, we recommend you use the "manual subscription" mode to reduce the traffic costs.

Param	DESC
autoRecvAudio	true: automatic subscription to audio; false: manual subscription to audio by calling `muteRemoteAudio(false)` . Default value: true
autoRecvVideo	true: automatic subscription to video; false: manual subscription to video by calling `startRemoteView` . Default value: true

Note

1. The configuration takes effect only if this API is called before room entry (enterRoom).
2. In the automatic subscription mode, if the user does not call startRemoteView to subscribe to the video stream after room entry, the SDK will automatically stop subscribing to the video stream in order to reduce the traffic consumption.

createSubCloud

createSubCloud

Create room subinstance (for concurrent multi-room listen/watch)

`TRTCCloud` was originally designed to work in the singleton mode, which limited the ability to watch concurrently in multiple rooms.
By calling this API, you can create multiple `TRTCCloud` instances, so that you can enter multiple different rooms at the same time to listen/watch audio/video streams.

However, it should be noted that your ability to publish audio and video streams in multiple `TRTCCloud` instances will be limited.

This feature is mainly used in the "super small class" use case in the online education scenario to break the limit that "only up to 50 users can publish their audio/video streams simultaneously in one TRTC room".

Below is the sample code:

```
//In the small room that needs interaction, enter the room as an anchor and pus
TRTCCloud mainCloud = TRTCCloud.sharedInstance(mContext);
TRTCCloudDef.TRTCParams mainParams = new TRTCCloudDef.TRTCParams();
//Fill your params
mainParams.role = TRTCCloudDef.TRTCRoleAnchor;
mainCloud.enterRoom(mainParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
//...
mainCloud.startLocalPreview(true, videoView);
mainCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);

//In the large room that only needs to watch, enter the room as an audience and
```

```
    TRTCCloud subCloud = mainCloud.createSubCloud();
    TRTCCloudDef.TRTCParams subParams = new TRTCCloudDef.TRTCParams();
    //Fill your params
    subParams.role = TRTCCloudDef.TRTCRoleAudience;
    subCloud.enterRoom(subParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
    //...
    subCloud.startRemoteView(userId, TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, view)
    //...
    //Exit from new room and release it.
    subCloud.exitRoom();
    mainCloud.destroySubCloud(subCloud);
```

Note

The same user can enter multiple rooms with different `roomId` values by using the same `userId`.

Two devices cannot use the same `userId` to enter the same room with a specified `roomId`.

You can set TRTCCloudListener separately for different instances to get their own event notifications.

The same user can push streams in multiple `TRTCCloud` instances at the same time, and can also call APIs related to local audio/video in the sub instance. But need to pay attention to:

Audio needs to be collected by the microphone or custom data at the same time in all instances, and the result of API calls related to the audio device will be based on the last time;

The result of camera-related API call will be based on the last time: startLocalPreview.

Return Desc:

`TRTCCloud` subinstance

destroySubCloud

destroySubCloud

void destroySubCloud	(final TRTCCloud subCloud)

Terminate room subinstance

Param	DESC
subCloud	

startPublishing

startPublishing

void startPublishing	(final String streamId
	final int streamType)

Start publishing audio/video streams to Tencent Cloud CSS CDN

This API sends a command to the TRTC server, requesting it to relay the current user's audio/video streams to CSS CDN.

You can set the `StreamId` of the live stream through the `streamId` parameter, so as to specify the playback address of the user's audio/video streams on CSS CDN.

For example, if you specify the current user's live stream ID as `user_stream_001` through this API, then the corresponding CDN playback address is:

"http://yourdomain/live/user_stream_001.flv", where `yourdomain` is your playback domain name with an ICP filing.

You can configure your playback domain name in the CSS console. Tencent Cloud does not provide a default playback domain name.

You can also specify the `streamId` when setting the `TRTCParams` parameter of `enterRoom` , which is the recommended approach.

Param	DESC
streamId	Custom stream ID.
streamType	Only `TRTCVideoStreamTypeBig` and `TRTCVideoStreamTypeSub` are supported.

Note

You need to enable the "Enable Relayed Push" option on the "Function Configuration" page in the TRTC console in advance.

If you select "Specified stream for relayed push", you can use this API to push the corresponding audio/video stream to Tencent Cloud CDN and specify the entered stream ID.

If you select "Global auto-relayed push", you can use this API to adjust the default stream ID.

stopPublishing

stopPublishing

Stop publishing audio/video streams to Tencent Cloud CSS CDN

startPublishCDNStream

startPublishCDNStream

void startPublishCDNStream	(TRTCCloudDef.TRTCPublishCDNParam param)

Start publishing audio/video streams to non-Tencent Cloud CDN

This API is similar to the `startPublishing` API. The difference is that `startPublishing` can only publish audio/video streams to Tencent Cloud CDN, while this API can relay streams to live streaming CDN services of other cloud providers.

Param	DESC
param	CDN relaying parameter. For more information, please see TRTCPublishCDNParam

Note

Using the `startPublishing` API to publish audio/video streams to Tencent Cloud CSS CDN does not incur additional fees.

Using the `startPublishCDNStream` API to publish audio/video streams to non-Tencent Cloud CDN incurs additional relaying bandwidth fees.

stopPublishCDNStream

stopPublishCDNStream

Stop publishing audio/video streams to non-Tencent Cloud CDN

setMixTranscodingConfig

setMixTranscodingConfig

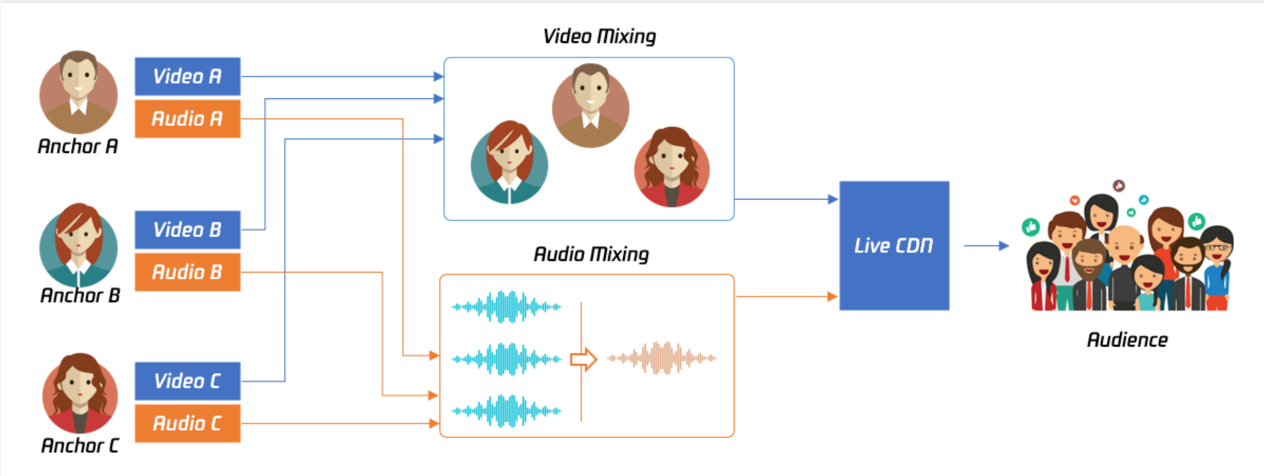
void setMixTranscodingConfig	(TRTCCloudDef.TRTCTranscodingConfig config)

Set the layout and transcoding parameters of On-Cloud MixTranscoding

In a live room, there may be multiple anchors publishing their audio/video streams at the same time, but for audience on CSS CDN, they only need to watch one video stream in HTTP-FLV or HLS format.

When you call this API, the SDK will send a command to the TRTC mixtranscoding server to combine multiple audio/video streams in the room into one stream.
You can use the TRTCTranscodingConfig parameter to set the layout of each channel of image. You can also set the encoding parameters of the mixed audio/video streams.

For more information, please see On-Cloud MixTranscoding.



Param	DESC
config	If `config` is not empty, On-Cloud MixTranscoding will be started; otherwise, it will be stopped. For more information, please see TRTCTranscodingConfig.

Note

Notes on On-Cloud MixTranscoding:

Mixed-stream transcoding is a chargeable function, calling the interface will incur cloud-based mixed-stream transcoding fees, see Billing of On-Cloud MixTranscoding.

If the user calling this API does not set `streamId` in the `config` parameter, TRTC will mix the multiple channels of images in the room into the audio/video streams corresponding to the current user, i.e., A + B => A.

If the user calling this API sets `streamId` in the `config` parameter, TRTC will mix the multiple channels of images in the room into the specified `streamId`, i.e., A + B => streamId.

Please note that if you are still in the room but do not need mixtranscoding anymore, be sure to call this API again and leave `config` empty to cancel it; otherwise, additional fees may be incurred.

Please rest assured that TRTC will automatically cancel the mixtranscoding status upon room exit.

startPublishMediaStream

startPublishMediaStream

void startPublishMediaStream	(TRTCCloudDef.TRTCPublishTarget target
	TRTCCloudDef.TRTCStreamEncoderParam params
	TRTCCloudDef.TRTCStreamMixingConfig config)

Publish a stream

After this API is called, the TRTC server will relay the stream of the local user to a CDN (after transcoding or without transcoding), or transcode and publish the stream to a TRTC room.

You can use the TRTCPublishMode parameter in TRTCPublishTarget to specify the publishing mode.

Param	DESC
config	The On-Cloud MixTranscoding settings. This parameter is invalid in the relay-to-CDN mode. It is required if you transcode and publish the stream to a CDN or to a TRTC room. For details, see TRTCStreamMixingConfig.
params	The encoding settings. This parameter is required if you transcode and publish the stream to a CDN or to a TRTC room. If you relay to a CDN without transcoding, to improve the relaying stability and playback compatibility, we also recommend you set this parameter. For details, see TRTCStreamEncoderParam.
target	The publishing destination. You can relay the stream to a CDN (after transcoding or without transcoding) or transcode and publish the stream to a TRTC room. For details, see TRTCPublishTarget.

Note

1. The SDK will send a task ID to you via the onStartPublishMediaStream callback.
2. You can start a publishing task only once and cannot initiate two tasks that use the same publishing mode and publishing cdn url. Note the task ID returned, which you need to pass to updatePublishMediaStream to modify the publishing parameters or stopPublishMediaStream to stop the task.
3. You can specify up to 10 CDN URLs in `target`. You will be charged only once for transcoding even if you relay to multiple CDNs.
4. To avoid causing errors, do not specify the same URLs for different publishing tasks executed at the same time. We recommend you add "sdkappid_roomid_userid_main" to URLs to distinguish them from one another and avoid application conflicts.

updatePublishMediaStream

updatePublishMediaStream

void updatePublishMediaStream	(final String taskId
	TRTCCloudDef.TRTCPublishTarget target
	TRTCCloudDef.TRTCStreamEncoderParam params
	TRTCCloudDef.TRTCStreamMixingConfig config)

Modify publishing parameters

You can use this API to change the parameters of a publishing task initiated by startPublishMediaStream.

Param	DESC
config	The On-Cloud MixTranscoding settings. This parameter is invalid in the relay-to-CDN mode. It is required if you transcode and publish the stream to a CDN or to a TRTC room. For details, see TRTCStreamMixingConfig.
params	The encoding settings. This parameter is required if you transcode and publish the stream to a CDN or to a TRTC room. If you relay to a CDN without transcoding, to improve the relaying stability and playback compatibility, we recommend you set this parameter. For details, see TRTCStreamEncoderParam.
target	The publishing destination. You can relay the stream to a CDN (after transcoding or without transcoding) or transcode and publish the stream to a TRTC room. For details, see TRTCPublishTarget.
taskId	The task ID returned to you via the onStartPublishMediaStream callback.

Note

1. You can use this API to add or remove CDN URLs to publish to (you can publish to up to 10 CDNs at a time). To avoid causing errors, do not specify the same URLs for different tasks executed at the same time.
2. You can use this API to switch a relaying task to transcoding or vice versa. For example, in cross-room communication, you can first call startPublishMediaStream to relay to a CDN. When the anchor requests cross-room communication, call this API, passing in the task ID to switch the relaying task to a transcoding task. This can ensure that the live stream and CDN playback are not interrupted (you need to keep the encoding parameters consistent).
3. You can not switch output between "only audio" 、 "only video" and "audio and video" for the same task.

stopPublishMediaStream

stopPublishMediaStream

void stopPublishMediaStream	(final String taskId)

Stop publishing

You can use this API to stop a task initiated by startPublishMediaStream.

Param	DESC
taskId	The task ID returned to you via the onStartPublishMediaStream callback.

Note

1. If the task ID is not saved to your backend, you can call startPublishMediaStream again when an anchor re-enters the room after abnormal exit. The publishing will fail, but the TRTC backend will return the task ID to you.
2. If `taskId` is left empty, the TRTC backend will end all tasks you started through startPublishMediaStream. You can leave it empty if you have started only one task or want to stop all publishing tasks started by you.

startLocalPreview

startLocalPreview

void startLocalPreview	(boolean frontCamera
	TXCloudVideoView view)

Enable the preview image of local camera (mobile)

If this API is called before `enterRoom`, the SDK will only enable the camera and wait until `enterRoom` is called before starting push.
If it is called after `enterRoom`, the SDK will enable the camera and automatically start pushing the video stream.
When the first camera video frame starts to be rendered, you will receive the `onCameraDidReady` callback in TRTCCloudListener.

Param	DESC
frontCamera	true: front camera; false: rear camera
view	Control that carries the video image

Note

If you want to preview the camera image and adjust the beauty filter parameters through `BeautyManager` before going live, you can:

Scheme 1. Call `startLocalPreview` before calling `enterRoom`

Scheme 2. Call `startLocalPreview` and `muteLocalVideo(true)` after calling `enterRoom`

updateLocalView

updateLocalView

void updateLocalView	(TXCloudVideoView view)

Update the preview image of local camera

stopLocalPreview

stopLocalPreview

Stop camera preview

muteLocalVideo

muteLocalVideo

void muteLocalVideo	(int streamType
	boolean mute)

Pause/Resume publishing local video stream

This API can pause (or resume) publishing the local video image. After the pause, other users in the same room will not be able to see the local image.

This API is equivalent to the two APIs of `startLocalPreview/stopLocalPreview` when TRTCVideoStreamTypeBig is specified, but has higher performance and response speed.

The `startLocalPreview/stopLocalPreview` APIs need to enable/disable the camera, which are hardware device-related operations, so they are very time-consuming.

In contrast, `muteLocalVideo` only needs to pause or allow the data stream at the software level, so it is more efficient and more suitable for scenarios where frequent enabling/disabling are needed.

After local video publishing is paused, other members in the same room will receive the `onUserVideoAvailable(userId, false)` callback notification.
After local video publishing is resumed, other members in the same room will receive the `onUserVideoAvailable(userId, true)` callback notification.

Param	DESC
mute	true: pause; false: resume
streamType	Specify for which video stream to pause (or resume). Only TRTCVideoStreamTypeBig and TRTCVideoStreamTypeSub are supported

setVideoMuteImage

setVideoMuteImage

void setVideoMuteImage	(Bitmap image
	int fps)

Set placeholder image during local video pause

When you call `muteLocalVideo(true)` to pause the local video image, you can set a placeholder image by calling this API. Then, other users in the room will see this image instead of a black screen.

Param	DESC
fps	Frame rate of the placeholder image. Minimum value: 5. Maximum value: 10. Default value: 5
image	Placeholder image. A null value means that no more video stream data will be sent after `muteLocalVideo` . The default value is null.

startRemoteView

startRemoteView

void startRemoteView	(String userId
	int streamType
	TXCloudVideoView view)

Subscribe to remote user's video stream and bind video rendering control

Calling this API allows the SDK to pull the video stream of the specified `userId` and render it to the rendering control specified by the `view` parameter. You can set the display mode of the video image through setRemoteRenderParams.
If you already know the `userId` of a user who has a video stream in the room, you can directly call `startRemoteView` to subscribe to the user's video image.
If you don't know which users in the room are publishing video streams, you can wait for the notification from onUserVideoAvailable after `enterRoom` .

Calling this API only starts pulling the video stream, and the image needs to be loaded and buffered at this time. After the buffering is completed, you will receive a notification from onFirstVideoFrame.

Param	DESC
streamType	Video stream type of the `userId` specified for watching: HD big image: TRTCVideoStreamTypeBig Smooth small image: TRTCVideoStreamTypeSmall (the remote user should enable dual-channel encoding through enableEncSmallVideoStream for this parameter to take effect) Substream image (usually used for screen sharing): TRTCVideoStreamTypeSub
userId	ID of the specified remote user
view	Rendering control that carries the video image

Note
The following requires your attention:
1. The SDK supports watching the big image and substream image or small image and substream image of a `userId` at the same time, but does not support watching the big image and small image at the same time.
2. Only when the specified `userId` enables dual-channel encoding through enableEncSmallVideoStream can the user's small image be viewed.
3. If the small image of the specified `userId` does not exist, the SDK will switch to the big image of the user by default.

updateRemoteView

updateRemoteView

void updateRemoteView	(String userId
	int streamType
	TXCloudVideoView view)

Update remote user's video rendering control

This API can be used to update the rendering control of the remote video image. It is often used in interactive scenarios where the display area needs to be switched.

Param	DESC
streamType	Type of the stream for which to set the preview window (only TRTCVideoStreamTypeBig and TRTCVideoStreamTypeSub are supported)
userId	ID of the specified remote user
view	Control that carries the video image

stopRemoteView

stopRemoteView

void stopRemoteView	(String userId
	int streamType)

Stop subscribing to remote user's video stream and release rendering control

Calling this API will cause the SDK to stop receiving the user's video stream and release the decoding and rendering resources for the stream.

Param	DESC
streamType	Video stream type of the `userId` specified for watching: HD big image: TRTCVideoStreamTypeBig Smooth small image: TRTCVideoStreamTypeSmall Substream image (usually used for screen sharing): TRTCVideoStreamTypeSub
userId	ID of the specified remote user

stopAllRemoteView

stopAllRemoteView

Stop subscribing to all remote users' video streams and release all rendering resources

Calling this API will cause the SDK to stop receiving all remote video streams and release all decoding and rendering resources.

Note

If a substream image (screen sharing) is being displayed, it will also be stopped.

muteRemoteVideoStream

muteRemoteVideoStream

void muteRemoteVideoStream	(String userId
	int streamType
	boolean mute)

Pause/Resume subscribing to remote user's video stream

This API only pauses/resumes receiving the specified user's video stream but does not release displaying resources; therefore, the video image will freeze at the last frame before it is called.

Param	DESC
mute	Whether to pause receiving
streamType	Specify for which video stream to pause (or resume): HD big image: TRTCVideoStreamTypeBig Smooth small image: TRTCVideoStreamTypeSmall Substream image (usually used for screen sharing): TRTCVideoStreamTypeSub
userId	ID of the specified remote user

Note

This API can be called before room entry (enterRoom), and the pause status will be reset after room exit (exitRoom). After calling this API to pause receiving the video stream from a specific user, simply calling the startRemoteView API will not be able to play the video from that user. You need to call muteRemoteVideoStream(false) or muteAllRemoteVideoStreams(false) to resume it.

muteAllRemoteVideoStreams

muteAllRemoteVideoStreams

void muteAllRemoteVideoStreams	(boolean mute)

Pause/Resume subscribing to all remote users' video streams

This API only pauses/resumes receiving all users' video streams but does not release displaying resources; therefore, the video image will freeze at the last frame before it is called.

Param	DESC
mute	Whether to pause receiving

Note

This API can be called before room entry (enterRoom), and the pause status will be reset after room exit (exitRoom). After calling this interface to pause receiving video streams from all users, simply calling the startRemoteView interface will not be able to play the video from a specific user. You need to call muteRemoteVideoStream(false) or muteAllRemoteVideoStreams(false) to resume it.

setVideoEncoderParam

setVideoEncoderParam

void setVideoEncoderParam	(TRTCCloudDef.TRTCVideoEncParam param)

Set the encoding parameters of video encoder

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.

Param	DESC
param	It is used to set relevant parameters for the video encoder. For more information, please see TRTCVideoEncParam.

Note

Begin from v11.5 version, the encoding output resolution will be aligned according to width 8 and height 2 bytes, and will be adjusted downward, eg: input resolution 540x960, actual encoding output resolution 536x960.

setNetworkQosParam

setNetworkQosParam

void setNetworkQosParam	(TRTCCloudDef.TRTCNetworkQosParam param)

Set network quality control parameters

This setting determines the quality control policy in a poor network environment, such as "image quality preferred" or "smoothness preferred".

Param	DESC
param	It is used to set relevant parameters for network quality control. For details, please refer to TRTCNetworkQosParam.

setLocalRenderParams

setLocalRenderParams

void setLocalRenderParams	(TRTCCloudDef.TRTCRenderParams renderParams)

Set the rendering parameters of local video image

The parameters that can be set include video image rotation angle, fill mode, and mirror mode.

Param	DESC
params	Video image rendering parameters. For more information, please see TRTCRenderParams.

setRemoteRenderParams

setRemoteRenderParams

void setRemoteRenderParams	(String userId
	int streamType
	TRTCCloudDef.TRTCRenderParams renderParams)

Set the rendering mode of remote video image

The parameters that can be set include video image rotation angle, fill mode, and mirror mode.

Param	DESC
params	Video image rendering parameters. For more information, please see TRTCRenderParams.
streamType	It can be set to the primary stream image (TRTCVideoStreamTypeBig) or substream image (TRTCVideoStreamTypeSub).
userId	ID of the specified remote user

enableEncSmallVideoStream

enableEncSmallVideoStream

int enableEncSmallVideoStream	(boolean enable
	TRTCCloudDef.TRTCVideoEncParam smallVideoEncParam)

Enable dual-channel encoding mode with big and small images

In this mode, the current user's encoder will output two channels of video streams, i.e., **HD big image** and **Smooth small image**, at the same time (only one channel of audio stream will be output though).
In this way, other users in the room can choose to subscribe to the **HD big image** or **Smooth small image** according to their own network conditions or screen size.

Param	DESC
enable	Whether to enable small image encoding. Default value: false
smallVideoEncParam	Video parameters of small image stream

Note

Dual-channel encoding will consume more CPU resources and network bandwidth; therefore, this feature can be enabled on macOS, Windows, or high-spec tablets, but is not recommended for phones.

Return Desc:

0: success; -1: the current big image has been set to a lower quality, and it is not necessary to enable dual-channel encoding

setRemoteVideoStreamType

setRemoteVideoStreamType

int setRemoteVideoStreamType	(String userId
	int streamType)

Switch the big/small image of specified remote user

After an anchor in a room enables dual-channel encoding, the video image that other users in the room subscribe to through startRemoteView will be **HD big image** by default.

You can use this API to select whether the image subscribed to is the big image or small image. The API can take effect before or after startRemoteView is called.

Param	DESC
streamType	Video stream type, i.e., big image or small image. Default value: big image
userId	ID of the specified remote user

Note

To implement this feature, the target user must have enabled the dual-channel encoding mode through enableEncSmallVideoStream; otherwise, this API will not work.

snapshotVideo

snapshotVideo

void snapshotVideo	(String userId
	int streamType
	int sourceType
	TRTCCloudListener.TRTCSnapshotListener listener)

Screencapture video

You can use this API to screencapture the local video image or the primary stream image and substream (screen sharing) image of a remote user.

Param	DESC
sourceType	Video image source, which can be the video stream image (TRTCSnapshotSourceTypeStream, generally in higher definition) 、 the video rendering image (TRTCSnapshotSourceTypeView) or the capture picture (TRTCSnapshotSourceTypeCapture).The captured picture screenshot will be clearer.
streamType	Video stream type, which can be the primary stream image (TRTCVideoStreamTypeBig, generally for camera) or substream image (TRTCVideoStreamTypeSub, generally for screen sharing)
userId	User ID. A null value indicates to screencapture the local video.

Note

On Windows, only video image from the TRTCSnapshotSourceTypeStream source can be screencaptured currently.

setPerspectiveCorrectionPoints

setPerspectiveCorrectionPoints

void setPerspectiveCorrectionPoints	(String userId
	PointF[] srcPoints
	PointF[] dstPoints)

Sets perspective correction coordinate points.

This function allows you to specify coordinate areas for perspective correction.

Param	DESC
dstPoints	The coordinates of the four vertices of the target corrected area should be passed in the order of top-left, bottom-left, top-right, bottom-right. All coordinates need to be normalized to the [0,1] range based on the render view width and height, or null to stop perspective correction of the corresponding stream.
srcPoints	The coordinates of the four vertices of the original stream image area should be passed in the order of top-left, bottom-left, top-right, bottom-right. All coordinates need to be normalized to the [0,1] range based on the render view width and height, or null to stop perspective correction of the corresponding stream.
userId	userId which corresponding to the target stream. If null value is specified, it indicates that the function is applied to the local stream.

setGravitySensorAdaptiveMode

setGravitySensorAdaptiveMode

void setGravitySensorAdaptiveMode	(int mode)

Set the adaptation mode of gravity sensing (version 11.7 and above)

After turning on gravity sensing, if the device on the collection end rotates, the images on the collection end and the audience will be rendered accordingly to ensure that the image in the field of view is always facing up.

It only takes effect in the camera capture scene inside the SDK, and only takes effect on the mobile terminal.

1. This interface only works for the collection end. If you only watch the picture in the room, opening this interface is invalid.
2. When the capture device is rotated 90 degrees or 270 degrees, the picture seen by the capture device or the audience may be cropped to maintain proportional coordination.

Param	DESC
mode	Gravity sensing mode, see TRTC_GRAVITY_SENSOR_ADAPTIVE_MODE_DISABLE、TRTC_GRAVITY_SENSOR_ADAPTIVE_ and TRTC_GRAVITY_SENSOR_ADAPTIVE_MODE_FIT_WITH_BLACK_BORDER for details, default TRTC_GRAVITY_SENSOR_ADAPTIVE_MODE_DISABLE.

startLocalAudio

startLocalAudio

void startLocalAudio	(int quality)

Enable local audio capturing and publishing

The SDK does not enable the mic by default. When a user wants to publish the local audio, the user needs to call this API to enable mic capturing and encode and publish the audio to the current room.
After local audio capturing and publishing is enabled, other users in the room will receive the onUserAudioAvailable(userId, true) notification.

Param	DESC
quality	Sound quality TRTC_AUDIO_QUALITY_SPEECH - Smooth: sample rate: 16 kHz; mono channel; audio bitrate: 16 Kbps. This is suitable for audio call scenarios, such as online meeting and audio call. TRTC_AUDIO_QUALITY_DEFAULT - Default: sample rate: 48 kHz; mono channel; audio bitrate: 50 Kbps. This is the default sound quality of the SDK and recommended if there are no special requirements. TRTC_AUDIO_QUALITY_MUSIC - HD: sample rate: 48 kHz; dual channel + full band; audio bitrate: 128 Kbps. This is suitable for scenarios where Hi-Fi music transfer is required, such as online karaoke and music live streaming.

Note

This API will check the mic permission. If the current application does not have permission to use the mic, the SDK will automatically ask the user to grant the mic permission.

stopLocalAudio

stopLocalAudio

Stop local audio capturing and publishing

After local audio capturing and publishing is stopped, other users in the room will receive the onUserAudioAvailable(userId, false) notification.

muteLocalAudio

muteLocalAudio

void muteLocalAudio	(boolean mute)

Pause/Resume publishing local audio stream

After local audio publishing is paused, other users in the room will receive the onUserAudioAvailable(userId, false) notification.
After local audio publishing is resumed, other users in the room will receive the onUserAudioAvailable(userId, true) notification.

Different from stopLocalAudio, `muteLocalAudio(true)` does not release the mic permission; instead, it continues to send mute packets with extremely low bitrate.
This is very suitable for scenarios that require on-cloud recording, as video file formats such as MP4 have a high requirement for audio continuity, while an MP4 recording file cannot be played back smoothly if stopLocalAudio is used.
Therefore, `muteLocalAudio` instead of `stopLocalAudio` is recommended in scenarios where the requirement for recording file quality is high.

Param	DESC
mute	true: mute; false: unmute

muteRemoteAudio

muteRemoteAudio

void muteRemoteAudio	(String userId
	boolean mute)

Pause/Resume playing back remote audio stream

When you mute the remote audio of a specified user, the SDK will stop playing back the user's audio and pulling the user's audio data.

Param	DESC
mute	true: mute; false: unmute
userId	ID of the specified remote user

Note

This API works when called either before or after room entry (enterRoom), and the mute status will be reset to `false` after room exit (exitRoom).

muteAllRemoteAudio

muteAllRemoteAudio

void muteAllRemoteAudio	(boolean mute)

Pause/Resume playing back all remote users' audio streams

When you mute the audio of all remote users, the SDK will stop playing back all their audio streams and pulling all their audio data.

Param	DESC
mute	true: mute; false: unmute

Note

This API works when called either before or after room entry (enterRoom), and the mute status will be reset to `false` after room exit (exitRoom).

setAudioRoute

setAudioRoute

void setAudioRoute	(int route)

Set audio route

Setting "audio route" is to determine whether the sound is played back from the speaker or receiver of a mobile device; therefore, this API is only applicable to mobile devices such as phones.

Generally, a phone has two speakers: one is the receiver at the top, and the other is the stereo speaker at the bottom.
If audio route is set to the receiver, the volume is relatively low, and the sound can be heard clearly only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.
If audio route is set to the speaker, the volume is relatively high, so there is no need to put the phone near the ear. Therefore, this mode can implement the "hands-free" feature.

Param	DESC
route	Audio route, i.e., whether the audio is output by speaker or receiver. Default value: TRTC_AUDIO_ROUTE_SPEAKER

setRemoteAudioVolume

setRemoteAudioVolume

void setRemoteAudioVolume	(String userId
	int volume)

Set the audio playback volume of remote user

You can mute the audio of a remote user through `setRemoteAudioVolume(userId, 0)` .

Param	DESC
userId	ID of the specified remote user
volume	Volume. 100 is the original volume. Value range: [0,150]. Default value: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setAudioCaptureVolume

setAudioCaptureVolume

void setAudioCaptureVolume	(int volume)

Set the capturing volume of local audio

Param	DESC
volume	Volume. 100 is the original volume. Value range: [0,150]. Default value: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

getAudioCaptureVolume

getAudioCaptureVolume

Get the capturing volume of local audio

setAudioPlayoutVolume

setAudioPlayoutVolume

void setAudioPlayoutVolume	(int volume)

Set the playback volume of remote audio

This API controls the volume of the sound ultimately delivered by the SDK to the system for playback. It affects the volume of the recorded local audio file but not the volume of in-ear monitoring.

Param	DESC
volume	Volume. 100 is the original volume. Value range: [0,150]. Default value: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

getAudioPlayoutVolume

getAudioPlayoutVolume

Get the playback volume of remote audio

enableAudioVolumeEvaluation

enableAudioVolumeEvaluation

void enableAudioVolumeEvaluation	(boolean enable
	TRTCCloudDef.TRTCAudioVolumeEvaluateParams params)

Enable volume reminder

After this feature is enabled, the SDK will return the audio volume assessment information of local user who sends stream and remote users in the onUserVoiceVolume callback of TRTCCloudListener.

Param	DESC
enable	Whether to enable the volume prompt. It's disabled by default.
params	Volume evaluation and other related parameters, please see TRTCAudioVolumeEvaluateParams

Note

To enable this feature, call this API before calling `startLocalAudio` .

startAudioRecording

startAudioRecording

int startAudioRecording	(TRTCCloudDef.TRTCAudioRecordingParams param)

Start audio recording

After you call this API, the SDK will selectively record local and remote audio streams (such as local audio, remote audio, background music, and sound effects) into a local file.

This API works when called either before or after room entry. If a recording task has not been stopped through `stopAudioRecording` before room exit, it will be automatically stopped after room exit.

The startup and completion status of the recording will be notified through local recording-related callbacks. See TRTCCloud related callbacks for reference.

Param	DESC
param	Recording parameter. For more information, please see TRTCAudioRecordingParams

Note

Since version 11.5, the results of audio recording have been changed to be notified through asynchronous callbacks instead of return values. Please refer to the relevant callbacks of TRTCCloud.

Return Desc:

0: success; -1: audio recording has been started; -2: failed to create file or directory; -3: the audio format of the specified file extension is not supported.

stopAudioRecording

stopAudioRecording

Stop audio recording

If a recording task has not been stopped through this API before room exit, it will be automatically stopped after room exit.

startLocalRecording

startLocalRecording

void startLocalRecording	(TRTCCloudDef.TRTCLocalRecordingParams params)

Start local media recording

This API records the audio/video content during live streaming into a local file.

Param	DESC
params	Recording parameter. For more information, please see TRTCLocalRecordingParams

stopLocalRecording

stopLocalRecording

Stop local media recording

If a recording task has not been stopped through this API before room exit, it will be automatically stopped after room exit.

setRemoteAudioParallelParams

setRemoteAudioParallelParams

void setRemoteAudioParallelParams	(TRTCCloudDef.TRTCAudioParallelParams params)

Set the parallel strategy of remote audio streams

For room with many speakers.

Param	DESC
params	Audio parallel parameter. For more information, please see TRTCAudioParallelParams

enable3DSpatialAudioEffect

enable3DSpatialAudioEffect

void enable3DSpatialAudioEffect	(boolean enabled)

Enable 3D spatial effect

Enable 3D spatial effect. Note that TRTC_AUDIO_QUALITY_SPEECH smooth or TRTC_AUDIO_QUALITY_DEFAULT default audio quality should be used.

Param	DESC
enabled	Whether to enable 3D spatial effect. It's disabled by default.

updateSelf3DSpatialPosition

updateSelf3DSpatialPosition

void updateSelf3DSpatialPosition	(int[] position
	float[] axisForward
	float[] axisRight
	float[] axisUp)

Update self position and orientation for 3D spatial effect

Update self position and orientation in the world coordinate system. The SDK will calculate the relative position between self and the remote users according to the parameters of this method, and then render the spatial sound effect. Note that the length of array should be 3.

Param	DESC
axisForward	The unit vector of the forward axis of user coordinate system. The three values represent the forward, right and up coordinate values in turn.
axisRight	The unit vector of the right axis of user coordinate system. The three values represent the forward, right and up coordinate values in turn.
axisUp	The unit vector of the up axis of user coordinate system. The three values represent the forward, right and up coordinate values in turn.
position	The coordinate of self in the world coordinate system. The three values represent the forward, right and up coordinate values in turn.

Note

Please limit the calling frequency appropriately. It's recommended that the interval between two operations be at least 100ms.

updateRemote3DSpatialPosition

updateRemote3DSpatialPosition

void updateRemote3DSpatialPosition	(String userId
	int[] position)

Update the specified remote user's position for 3D spatial effect

Update the specified remote user's position in the world coordinate system. The SDK will calculate the relative position between self and the remote users according to the parameters of this method, and then render the spatial sound effect. Note that the length of array should be 3.

Param	DESC
position	The coordinate of self in the world coordinate system. The three values represent the forward, right and up coordinate values in turn.
userId	ID of the specified remote user.

Note

Please limit the calling frequency appropriately. It's recommended that the interval between two operations of the same remote user be at least 100ms.

set3DSpatialReceivingRange

set3DSpatialReceivingRange

void set3DSpatialReceivingRange	(String userId
	int range)

Set the maximum 3D spatial attenuation range for userId's audio stream

After set the range, the specified user's audio stream will attenuate to zero within the range.

Param	DESC
range	Maximum attenuation range of the audio stream.
userId	ID of the specified user.

getDeviceManager

getDeviceManager

Get device management class (TXDeviceManager)

getBeautyManager

getBeautyManager

Get beauty filter management class (TXBeautyManager)

You can use the following features with beauty filter management:
Set beauty effects such as "skin smoothing", "brightening", and "rosy skin".
Set face adjustment effects such as "eye enlarging", "face slimming", "chin slimming", "chin lengthening/shortening", "face shortening", "nose narrowing", "eye brightening", "teeth whitening", "eye bag removal", "wrinkle removal", and "smile line removal".
Set face adjustment effects such as "hairline", "eye distance", "eye corners", "mouth shape", "nose wing", "nose position", "lip thickness", and "face shape".

Set makeup effects such as "eye shadow" and "blush".
Set animated effects such as animated sticker and facial pendant.

setWatermark

setWatermark

void setWatermark	(Bitmap image
	int streamType
	float x
	float y
	float width)

Add watermark

The watermark position is determined by the `rect` parameter, which is a quadruple in the format of (x, y, width, height).

x: X coordinate of watermark, which is a floating-point number between 0 and 1.
y: Y coordinate of watermark, which is a floating-point number between 0 and 1.
width: width of watermark, which is a floating-point number between 0 and 1.
height: it does not need to be set. The SDK will automatically calculate it according to the watermark image's aspect ratio.

Sample parameter:
If the encoding resolution of the current video is 540x960, and the `rect` parameter is set to (0.1, 0.1, 0.2, 0.0), then the coordinates of the top-left point of the watermark will be (540 * 0.1, 960 * 0.1), i.e., (54, 96), the watermark width will be 540 * 0.2 = 108 px, and the watermark height will be calculated automatically by the SDK based on the watermark image's aspect ratio.

Param	DESC
image	Watermark image, which must be a PNG image with transparent background
rect	Unified coordinates of the watermark relative to the encoded resolution. Value range of `x` , `y` , `width` , and `height` : 0–1.
streamType	Specify for which image to set the watermark. For more information, please see TRTCVideoStreamType.

Note

If you want to set watermarks for both the primary image (generally for the camera) and the substream image (generally for screen sharing), you need to call this API twice with `streamType` set to different values.

getAudioEffectManager

getAudioEffectManager

Get sound effect management class (TXAudioEffectManager)

`TXAudioEffectManager` is a sound effect management API, through which you can implement the following features:

Background music: both online music and local music can be played back with various features such as speed adjustment, pitch adjustment, original voice, accompaniment, and loop.

In-ear monitoring: the sound captured by the mic is played back in the headphones in real time, which is generally used for music live streaming.

Reverb effect: karaoke room, small room, big hall, deep, resonant, and other effects.

Voice changing effect: young girl, middle-aged man, heavy metal, and other effects.

Short sound effect: short sound effect files such as applause and laughter are supported (for files less than 10 seconds in length, please set the `isShortFile` parameter to `true`).

startSystemAudioLoopback

startSystemAudioLoopback

Enable system audio capturing

This API captures audio data from another app and mixes it into the current audio stream of the SDK. This ensures that other users in the room hear the audio played back by the another app.

In online education scenarios, a teacher can use this API to have the SDK capture the audio of instructional videos and broadcast it to students in the room.

In live music scenarios, an anchor can use this API to have the SDK capture the music played back by his or her player so as to add background music to the room.

Note

1. This interface only works on Android API 29 and above.
2. You need to use this interface to enable system sound capture first, and it will take effect only when you call startScreenCapture to enable screen sharing.

3. You need to add a foreground service to ensure that the system sound capture is not silenced, and set android:foregroundServiceType="mediaProjection".
4. The SDK only capture audio of applications that satisfies the capture strategy and audio usage. Currently, the audio usage captured by the SDK includes USAGE_MEDIA, USAGE_GAME。

stopSystemAudioLoopback

stopSystemAudioLoopback

Stop system audio capturing(iOS not supported)

startScreenCapture

startScreenCapture

void startScreenCapture	(int streamType
	TRTCCloudDef.TRTCVideoEncParam encParams
	TRTCCloudDef.TRTCScreenShareParams shareParams)

Start screen sharing

This API supports capturing the screen of the entire Android system, which can implement system-wide screen sharing similar to VooV Meeting.

For more information, please see Android

Video encoding parameters recommended for screen sharing on Android (TRTCVideoEncParam):
Resolution (videoResolution): 1280x720
Frame rate (videoFps): 10 fps
Bitrate (videoBitrate): 1200 Kbps
Resolution adaption (enableAdjustRes): false

Param	DESC
encParams	Encoding parameters. For more information, please see TRTCCloudDef#TRTCVideoEncParam. If `encParams` is set to `null` , the SDK will automatically use the previously set encoding parameter.
shareParams	For more information, please see TRTCCloudDef#TRTCScreenShareParams. You can

	use the `floatingView` parameter to pop up a floating window (you can also use Android's `WindowManager` parameter to configure automatic pop-up).

stopScreenCapture

stopScreenCapture

Stop screen sharing

pauseScreenCapture

pauseScreenCapture

Pause screen sharing

Note

Begin from v11.5 version, paused screen capture will use the last frame to output at a frame rate of 1fps.

resumeScreenCapture

resumeScreenCapture

Resume screen sharing

setSubStreamEncoderParam

setSubStreamEncoderParam

void setSubStreamEncoderParam	(TRTCCloudDef.TRTCVideoEncParam param)

Set the video encoding parameters of screen sharing (i.e., substream) (for desktop and mobile systems)

This API can set the image quality of screen sharing (i.e., the substream) viewed by remote users, which is also the image quality of screen sharing in on-cloud recording files.

Please note the differences between the following two APIs:

setVideoEncoderParam is used to set the video encoding parameters of the primary stream image (TRTCVideoStreamTypeBig, generally for camera).

setSubStreamEncoderParam is used to set the video encoding parameters of the substream image (TRTCVideoStreamTypeSub, generally for screen sharing).

Param	DESC
param	Substream encoding parameters. For more information, please see TRTCVideoEncParam.

enableCustomVideoCapture

enableCustomVideoCapture

void enableCustomVideoCapture	(int streamType
	boolean enable)

Enable/Disable custom video capturing mode

After this mode is enabled, the SDK will not run the original video capturing process (i.e., stopping camera data capturing and beauty filter operations) and will retain only the video encoding and sending capabilities.
You need to use sendCustomVideoData to continuously insert the captured video image into the SDK.

Param	DESC
enable	Whether to enable. Default value: false
streamType	Specify video stream type (TRTCVideoStreamTypeBig: HD big image; TRTCVideoStreamTypeSub: substream image).

sendCustomVideoData

sendCustomVideoData

void sendCustomVideoData	(int streamType
	TRTCCloudDef.TRTCVideoFrame frame)

Deliver captured video frames to SDK

You can use this API to deliver video frames you capture to the SDK, and the SDK will encode and transfer them through its own network module.
There are two delivery schemes for Android:

Memory-based delivery scheme: its connection is easy but its performance is poor, so it is not suitable for scenarios with high resolution.
Video memory-based delivery scheme: its connection requires certain knowledge in OpenGL, but its performance is good. For resolution higher than 640x360, please use this scheme.

For more information, please see Custom Capturing and Rendering.

Param	DESC
frame	Video data. If the memory-based delivery scheme is used, please set the `data` field; if the video memory-based delivery scheme is used, please set the `TRTCTexture` field. For more information, please see com::tencent::trtc::TRTCCloudDef::TRTCVideoFrame TRTCVideoFrame.
streamType	Specify video stream type (TRTCVideoStreamTypeBig: HD big image; TRTCVideoStreamTypeSub: substream image).

Note

1. We recommend you call the generateCustomPTS API to get the `timestamp` value of a video frame immediately after capturing it, so as to achieve the best audio/video sync effect.
2. The video frame rate eventually encoded by the SDK is not determined by the frequency at which you call this API, but by the FPS you set in setVideoEncoderParam.
3. Please try to keep the calling interval of this API even; otherwise, problems will be caused, such as unstable output frame rate of the encoder or out-of-sync audio/video.

enableCustomAudioCapture

enableCustomAudioCapture

void enableCustomAudioCapture	(boolean enable)

Enable custom audio capturing mode

After this mode is enabled, the SDK will not run the original audio capturing process (i.e., stopping mic data capturing) and will retain only the audio encoding and sending capabilities.
You need to use sendCustomAudioData to continuously insert the captured audio data into the SDK.

Param	DESC
enable	Whether to enable. Default value: false

Note

As acoustic echo cancellation (AEC) requires strict control over the audio capturing and playback time, after custom audio capturing is enabled, AEC may fail.

sendCustomAudioData

sendCustomAudioData

void sendCustomAudioData	(TRTCCloudDef.TRTCAudioFrame frame)

Deliver captured audio data to SDK

We recommend you enter the following information for the TRTCAudioFrame parameter (other fields can be left empty):

audioFormat: audio data format, which can only be `TRTCAudioFrameFormatPCM` .

data: audio frame buffer. Audio frame data must be in PCM format, and it supports a frame length of 5–100 ms (20 ms is recommended). Length calculation method: **for example, if the sample rate is 48000, then the frame length for mono channel will be `48000 * 0.02s * 1 * 16 bit = 15360 bit = 1920 bytes`.**

sampleRate: sample rate. Valid values: 16000, 24000, 32000, 44100, 48000.

channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel.

timestamp (ms): Set it to the timestamp when audio frames are captured, which you can obtain by calling generateCustomPTS after getting a audio frame.

For more information, please see Custom Capturing and Rendering.

Param	DESC
frame	Audio data

Note

Please call this API accurately at intervals of the frame length; otherwise, sound lag may occur due to uneven data delivery intervals.

enableMixExternalAudioFrame

enableMixExternalAudioFrame

void enableMixExternalAudioFrame	(boolean enablePublish
	boolean enablePlayout)

Enable/Disable custom audio track

After this feature is enabled, you can mix a custom audio track into the SDK through this API. With two boolean parameters, you can control whether to play back this track remotely or locally.

Param	DESC
enablePlayout	Whether the mixed audio track should be played back locally. Default value: false
enablePublish	Whether the mixed audio track should be played back remotely. Default value: false

Note

If you specify both `enablePublish` and `enablePlayout` as `false`, the custom audio track will be completely closed.

mixExternalAudioFrame

mixExternalAudioFrame

int mixExternalAudioFrame	(TRTCCloudDef.TRTCAudioFrame frame)

Mix custom audio track into SDK

Before you use this API to mix custom PCM audio into the SDK, you need to first enable custom audio tracks through enableMixExternalAudioFrame.

You are expected to feed audio data into the SDK at an even pace, but we understand that it can be challenging to call an API at absolutely regular intervals.

Given this, we have provided a buffer pool in the SDK, which can cache the audio data you pass in to reduce the fluctuations in intervals between API calls.

The value returned by this API indicates the size (ms) of the buffer pool. For example, if `50` is returned, it indicates that the buffer pool has 50 ms of audio data. As long as you call this API again within 50 ms, the SDK can make sure that continuous audio data is mixed.

If the value returned is `100` or greater, you can wait after an audio frame is played to call the API again. If the value returned is smaller than `100`, then there isn’t enough data in the buffer pool, and you should feed more audio data into the SDK until the data in the buffer pool is above the safety level.

Fill the fields in TRTCAudioFrame as follows (other fields are not required).

`data` : audio frame buffer. Audio frames must be in PCM format. Each frame can be 5-100 ms (20 ms is recommended) in duration. Assume that the sample rate is 48000, and sound channels mono-channel. Then the

frame size would be 48000 x 0.02s x 1 x 16 bit = 15360 bit = 1920 bytes.

`sampleRate` : sample rate. Valid values: 16000, 24000, 32000, 44100, 48000

`channel` : number of sound channels (if dual-channel is used, data is interleaved). Valid values: `1` (mono-channel); `2` (dual channel)

`timestamp` : timestamp (ms). Set it to the timestamp when audio frames are captured, which you can obtain by calling generateCustomPTS after getting an audio frame.

Param	DESC
frame	Audio data

Return Desc:

If the value returned is `0` or greater, the value represents the current size of the buffer pool; if the value returned is smaller than `0` , it means that an error occurred. `-1` indicates that you didn't call enableMixExternalAudioFrame to enable custom audio tracks.

setMixExternalAudioVolume

setMixExternalAudioVolume

void setMixExternalAudioVolume	(int publishVolume
	int playoutVolume)

Set the publish volume and playback volume of mixed custom audio track

Param	DESC
playoutVolume	set the play volume， from 0 to 100, -1 means no change
publishVolume	set the publish volume， from 0 to 100, -1 means no change

generateCustomPTS

generateCustomPTS

Generate custom capturing timestamp

This API is only suitable for the custom capturing mode and is used to solve the problem of out-of-sync audio/video caused by the inconsistency between the capturing time and delivery time of audio/video frames.

When you call APIs such as sendCustomVideoData or sendCustomAudioData for custom video or audio capturing, please use this API as instructed below:
1. First, when a video or audio frame is captured, call this API to get the corresponding PTS timestamp.
2. Then, send the video or audio frame to the preprocessing module you use (such as a third-party beauty filter or sound effect component).
3. When you actually call sendCustomVideoData or sendCustomAudioData for delivery, assign the PTS timestamp recorded when the frame was captured to the `timestamp` field in TRTCVideoFrame or TRTCAudioFrame.

Return Desc:

Timestamp in ms

setLocalVideoProcessListener

setLocalVideoProcessListener

int setLocalVideoProcessListener	(int pixelFormat
	int bufferType
	TRTCCloudListener.TRTCVideoFrameListener listener)

Set video data callback for third-party beauty filters

After this callback is set, the SDK will call back the captured video frames through the `listener` you set and use them for further processing by a third-party beauty filter component. Then, the SDK will encode and send the processed video frames.

Param	DESC
bufferType	Specify the format of the data called back. Currently, it supports: TRTC_VIDEO_BUFFER_TYPE_TEXTURE: suitable when `pixelFormat` is set to TRTC_VIDEO_PIXEL_FORMAT_Texture_2D. TRTC_VIDEO_BUFFER_TYPE_BYTE_BUFFER: suitable when `pixelFormat` is set to TRTC_VIDEO_PIXEL_FORMAT_I420. TRTC_VIDEO_BUFFER_TYPE_BYTE_ARRAY: suitable when `pixelFormat` is set to TRTC_VIDEO_PIXEL_FORMAT_I420.
listener	Custom preprocessing callback. For more information, please see TRTCVideoFrameListener
pixelFormat	Specify the format of the pixel called back. Currently, it supports: TRTC_VIDEO_PIXEL_FORMAT_Texture_2D: video memory-based texture scheme.

	TRTC_VIDEO_PIXEL_FORMAT_I420: memory-based data scheme.

Return Desc:

0: success; values smaller than 0: error

setLocalVideoRenderListener

setLocalVideoRenderListener

int setLocalVideoRenderListener	(int pixelFormat
	int bufferType
	TRTCCloudListener.TRTCVideoRenderListener listener)

Set the callback of custom rendering for local video

After this callback is set, the SDK will skip its own rendering process and call back the captured data. Therefore, you need to complete image rendering on your own.

`pixelFormat` specifies the format of the data called back. Currently, Texture2D, I420, and RGBA formats are supported.

`bufferType` specifies the buffer type. `BYTE_BUFFER` is suitable for the JNI layer, while `BYTE_ARRAY` can be used in direct operations at the Java layer.

For more information, please see Custom Capturing and Rendering.

Param	DESC
bufferType	Specify the data structure of the video frame: TRTC_VIDEO_BUFFER_TYPE_TEXTURE: suitable when `pixelFormat` is set to TRTC_VIDEO_PIXEL_FORMAT_Texture_2D. TRTC_VIDEO_BUFFER_TYPE_BYTE_BUFFER: suitable when `pixelFormat` is set to TRTC_VIDEO_PIXEL_FORMAT_I420 or TRTC_VIDEO_PIXEL_FORMAT_RGBA. TRTC_VIDEO_BUFFER_TYPE_BYTE_ARRAY: suitable when `pixelFormat` is set to TRTC_VIDEO_PIXEL_FORMAT_I420 or TRTC_VIDEO_PIXEL_FORMAT_RGBA.
listener	Callback of custom video rendering. The callback is returned once for each video frame
pixelFormat	Specify the format of the video frame, such as: TRTC_VIDEO_PIXEL_FORMAT_Texture_2D: OpenGL texture format, which is suitable for GPU processing and has a high processing efficiency. TRTC_VIDEO_PIXEL_FORMAT_I420: standard I420 format, which is suitable for CPU processing and has a poor processing efficiency.

	TRTC_VIDEO_PIXEL_FORMAT_RGBA: RGBA format, which is suitable for CPU processing and has a poor processing efficiency.

Return Desc:

0: success; values smaller than 0: error

setRemoteVideoRenderListener

setRemoteVideoRenderListener

int setRemoteVideoRenderListener	(String userId
	int pixelFormat
	int bufferType
	TRTCCloudListener.TRTCVideoRenderListener listener)

Set the callback of custom rendering for remote video

After this callback is set, the SDK will skip its own rendering process and call back the captured data. Therefore, you need to complete image rendering on your own.

`pixelFormat` specifies the format of the called back data, such as NV12, I420, and 32BGRA.

`bufferType` specifies the buffer type. `PixelBuffer` has the highest efficiency, while `NSData` makes the SDK perform a memory conversion internally, which will result in extra performance loss.

For more information, please see Custom Capturing and Rendering.

Param	DESC
bufferType	Specify video data structure type.
listener	listen for custom rendering
pixelFormat	Specify the format of the pixel called back
userId	ID of the specified remote user

Note

Before this API is called, `startRemoteView(nil)` needs to be called to get the video stream of the remote user (`view` can be set to `nil` for this end); otherwise, there will be no data called back.

Return Desc:

0: success; values smaller than 0: error

setAudioFrameListener

setAudioFrameListener

void setAudioFrameListener	(TRTCCloudListener.TRTCAudioFrameListener listener)

Set custom audio data callback

After this callback is set, the SDK will internally call back the audio data (in PCM format), including:

onCapturedAudioFrame: callback of the audio data captured by the local mic

onLocalProcessedAudioFrame: callback of the audio data captured by the local mic and preprocessed by the audio module

onRemoteUserAudioFrame: audio data from each remote user before audio mixing

onMixedPlayAudioFrame: callback of the audio data that will be played back by the system after audio streams are mixed

Note

Setting the callback to null indicates to stop the custom audio callback, while setting it to a non-null value indicates to start the custom audio callback.

setCapturedAudioFrameCallbackFormat

setCapturedAudioFrameCallbackFormat

int setCapturedAudioFrameCallbackFormat	(TRTCCloudDef.TRTCAudioFrameCallbackFormat format)

Set the callback format of audio frames captured by local mic

This API is used to set the `AudioFrame` format called back by onCapturedAudioFrame:

sampleRate: sample rate. Valid values: 16000, 32000, 44100, 48000

channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel

samplesPerCall: number of sample points, which defines the frame length of the callback data. The frame length must be an integer multiple of 10 ms.

If you want to calculate the callback frame length in milliseconds, the formula for converting the number of milliseconds into the number of sample points is as follows: number of sample points = number of milliseconds * sample rate / 1000

For example, if you want to call back the data of 20 ms frame length with 48000 sample rate, then the number of sample points should be entered as 960 = 20 * 48000 / 1000
Note that the frame length of the final callback is in bytes, and the calculation formula for converting the number of sample points into the number of bytes is as follows: number of bytes = number of sample points * number of channels * 2 (bit width)
For example, if the parameters are 48000 sample rate, dual channel, 20 ms frame length, and 960 sample points, then the number of bytes is 3840 = 960 * 2 * 2

Param	DESC
format	Audio data callback format

Return Desc:

0: success; values smaller than 0: error

setLocalProcessedAudioFrameCallbackFormat

setLocalProcessedAudioFrameCallbackFormat

int setLocalProcessedAudioFrameCallbackFormat	(TRTCCloudDef.TRTCAudioFrameCallbackFormat format)

Set the callback format of preprocessed local audio frames

This API is used to set the `AudioFrame` format called back by onLocalProcessedAudioFrame:
sampleRate: sample rate. Valid values: 16000, 32000, 44100, 48000
channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel
samplesPerCall: number of sample points, which defines the frame length of the callback data. The frame length must be an integer multiple of 10 ms.

If you want to calculate the callback frame length in milliseconds, the formula for converting the number of milliseconds into the number of sample points is as follows: number of sample points = number of milliseconds * sample rate / 1000
For example, if you want to call back the data of 20 ms frame length with 48000 sample rate, then the number of sample points should be entered as 960 = 20 * 48000 / 1000
Note that the frame length of the final callback is in bytes, and the calculation formula for converting the number of sample points into the number of bytes is as follows: number of bytes = number of sample points * number of channels * 2 (bit width)

For example, if the parameters are 48000 sample rate, dual channel, 20 ms frame length, and 960 sample points, then the number of bytes is 3840 = 960 * 2 * 2

Param	DESC
format	Audio data callback format

Return Desc:

0: success; values smaller than 0: error

setMixedPlayAudioFrameCallbackFormat

setMixedPlayAudioFrameCallbackFormat

int setMixedPlayAudioFrameCallbackFormat	(TRTCCloudDef.TRTCAudioFrameCallbackFormat format)

Set the callback format of audio frames to be played back by system

This API is used to set the `AudioFrame` format called back by onMixedPlayAudioFrame:
sampleRate: sample rate. Valid values: 16000, 32000, 44100, 48000
channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel
samplesPerCall: number of sample points, which defines the frame length of the callback data. The frame length must be an integer multiple of 10 ms.

If you want to calculate the callback frame length in milliseconds, the formula for converting the number of milliseconds into the number of sample points is as follows: number of sample points = number of milliseconds * sample rate / 1000
For example, if you want to call back the data of 20 ms frame length with 48000 sample rate, then the number of sample points should be entered as 960 = 20 * 48000 / 1000
Note that the frame length of the final callback is in bytes, and the calculation formula for converting the number of sample points into the number of bytes is as follows: number of bytes = number of sample points * number of channels * 2 (bit width)
For example, if the parameters are 48000 sample rate, dual channel, 20 ms frame length, and 960 sample points, then the number of bytes is 3840 = 960 * 2 * 2

Param	DESC
format	Audio data callback format

Return Desc:

0: success; values smaller than 0: error

enableCustomAudioRendering

enableCustomAudioRendering

void enableCustomAudioRendering	(boolean enable)

Enabling custom audio playback

You can use this API to enable custom audio playback if you want to connect to an external audio device or control the audio playback logic by yourself.
After you enable custom audio playback, the SDK will stop using its audio API to play back audio. You need to call getCustomAudioRenderingFrame to get audio frames and play them by yourself.

Param	DESC
enable	Whether to enable custom audio playback. It's disabled by default.

Note
The parameter must be set before room entry to take effect.

getCustomAudioRenderingFrame

getCustomAudioRenderingFrame

void getCustomAudioRenderingFrame	(final TRTCCloudDef.TRTCAudioFrame audioFrame)

Getting playable audio data

Before calling this API, you need to first enable custom audio playback using enableCustomAudioRendering.

Fill the fields in TRTCAudioFrame as follows (other fields are not required):
`sampleRate` : sample rate (required). Valid values: 16000, 24000, 32000, 44100, 48000
`channel` : number of sound channels (required). `1` : mono-channel; `2` : dual-channel; if dual-channel is used, data is interleaved.
`data` : the buffer used to get audio data. You need to allocate memory for the buffer based on the duration of an audio frame.
The PCM data obtained can have a frame duration of 10 ms or 20 ms. 20 ms is recommended.

Assume that the sample rate is 48000, and sound channels mono-channel. The buffer size for a 20 ms audio frame would be 48000 x 0.02s x 1 x 16 bit = 15360 bit = 1920 bytes.

Param	DESC
audioFrame	Audio frames

Note

1. You must set `sampleRate` and `channel` in `audioFrame`, and allocate memory for one frame of audio in advance.
2. The SDK will fill the data automatically based on `sampleRate` and `channel`.
3. We recommend that you use the system's audio playback thread to drive the calling of this API, so that it is called each time the playback of an audio frame is complete.

sendCustomCmdMsg

sendCustomCmdMsg

boolean sendCustomCmdMsg	(int cmdID
	byte[] data
	boolean reliable
	boolean ordered)

Use UDP channel to send custom message to all users in room

This API allows you to use TRTC's UDP channel to broadcast custom data to other users in the current room for signaling transfer.

Other users in the room can receive the message through the `onRecvCustomCmdMsg` callback in TRTCCloudListener.

Param	DESC
cmdID	Message ID. Value range: 1–10
data	Message to be sent. The maximum length of one single message is 1 KB.
ordered	Whether orderly sending is enabled, i.e., whether the data packets should be received in the same order in which they are sent; if so, a certain delay will be caused.
reliable	Whether reliable sending is enabled. Reliable sending can achieve a higher success rate but with a longer reception delay than unreliable sending.

Note

1. Up to 30 messages can be sent per second to all users in the room (this is not supported for web and mini program currently).
2. A packet can contain up to 1 KB of data; if the threshold is exceeded, the packet is very likely to be discarded by the intermediate router or server.
3. A client can send up to 8 KB of data in total per second.
4. `reliable` and `ordered` must be set to the same value (`true` or `false`) and cannot be set to different values currently.
5. We strongly recommend you set different `cmdID` values for messages of different types. This can reduce message delay when orderly sending is required.
6. Currently only the anchor role is supported.

Return Desc:

true: sent the message successfully; false: failed to send the message.

sendSEIMsg

sendSEIMsg

boolean sendSEIMsg	(byte[] data
	int repeatCount)

Use SEI channel to send custom message to all users in room

This API allows you to use TRTC's SEI channel to broadcast custom data to other users in the current room for signaling transfer.

The header of a video frame has a header data block called SEI. This API works by embedding the custom signaling data you want to send in the SEI block and sending it together with the video frame.
Therefore, the SEI channel has a better compatibility than sendCustomCmdMsg as the signaling data can be transferred to the CSS CDN along with the video frame.
However, because the data block of the video frame header cannot be too large, we recommend you limit the size of the signaling data to only a few bytes when using this API.

The most common use is to embed the custom timestamp into video frames through this API so as to implement a perfect alignment between the message and video image (such as between the teaching material and video signal in the education scenario).

Other users in the room can receive the message through the `onRecvSEIMsg` callback in TRTCCloudListener.

Param	DESC
data	Data to be sent, which can be up to 1 KB (1,000 bytes)
repeatCount	Data sending count

Note

This API has the following restrictions:

1. The data will not be instantly sent after this API is called; instead, it will be inserted into the next video frame after the API call.
2. Up to 30 messages can be sent per second to all users in the room (this limit is shared with `sendCustomCmdMsg`).
3. Each packet can be up to 1 KB (this limit is shared with `sendCustomCmdMsg`). If a large amount of data is sent, the video bitrate will increase, which may reduce the video quality or even cause lagging.
4. Each client can send up to 8 KB of data in total per second (this limit is shared with `sendCustomCmdMsg`).
5. If multiple times of sending is required (i.e., `repeatCount` > 1), the data will be inserted into subsequent `repeatCount` video frames in a row for sending, which will increase the video bitrate.
6. If `repeatCount` is greater than 1, the data will be sent for multiple times, and the same message may be received multiple times in the `onRecvSEIMsg` callback; therefore, deduplication is required.

Return Desc:

true: the message is allowed and will be sent with subsequent video frames; false: the message is not allowed to be sent

startSpeedTest

startSpeedTest

int startSpeedTest	(TRTCCloudDef.TRTCSpeedTestParams params)

Start network speed test (used before room entry)

Param	DESC
params	speed test options

Note

1. The speed measurement process will incur a small amount of basic service fees, See Purchase Guide > Base Services.
2. Please perform the Network speed test before room entry, because if performed after room entry, the test will affect the normal audio/video transfer, and its result will be inaccurate due to interference in the room.
3. Only one network speed test task is allowed to run at the same time.

Return Desc:

interface call result, <0: failure

stopSpeedTest

stopSpeedTest

Stop network speed test

getSDKVersion

getSDKVersion

Get SDK version information

setLogLevel

setLogLevel

void setLogLevel	(int level)

Set log output level

Param	DESC
level	For more information, please see TRTCLogLevel. Default value: TRTCLogLevelNone

setConsoleEnabled

setConsoleEnabled

void setConsoleEnabled	(boolean enabled)

Enable/Disable console log printing

Param	DESC
enabled	Specify whether to enable it, which is disabled by default

setLogCompressEnabled

setLogCompressEnabled

void setLogCompressEnabled	(boolean enabled)

Enable/Disable local log compression

If compression is enabled, the log size will significantly reduce, but logs can be read only after being decompressed by the Python script provided by Tencent Cloud.
If compression is disabled, logs will be stored in plaintext and can be read directly in Notepad, but will take up more storage capacity.

Param	DESC
enabled	Specify whether to enable it, which is enabled by default

setLogDirPath

setLogDirPath

void setLogDirPath	(String path)

Set local log storage path

You can use this API to change the default storage path of the SDK's local logs, which is as follows:
Windows: C:/Users/[username]/AppData/Roaming/liteav/log, i.e., under `%appdata%/liteav/log` .
iOS or macOS: under `sandbox Documents/log` .
Android: under `/app directory/files/log/liteav/` .

Param	DESC
path	Log storage path

Note

Please be sure to call this API before all other APIs and make sure that the directory you specify exists and your application has read/write permissions of the directory.

setLogListener

setLogListener

void setLogListener	(final TRTCCloudListener.TRTCLogListener logListener)

Set log callback

showDebugView

showDebugView

void showDebugView	(int showType)

Display dashboard

"Dashboard" is a semi-transparent floating layer for debugging information on top of the video rendering control. It is used to display audio/video information and event information to facilitate integration and debugging.

Param	DESC
showType	0: does not display; 1: displays lite edition (only with audio/video information); 2: displays full edition (with audio/video information and event information).

TRTCViewMargin

TRTCViewMargin

public TRTCViewMargin	(float leftMargin
	float rightMargin
	float topMargin
	float bottomMargin)

Set dashboard margin

This API is used to adjust the position of the dashboard in the video rendering control. It must be called before `showDebugView` for it to take effect.

Param	DESC
margin	Inner margin of the dashboard. It should be noted that this is based on the percentage of `parentView` . Value range: 0–1
userId	User ID

callExperimentalAPI

callExperimentalAPI

String callExperimentalAPI	(String jsonStr)

Call experimental APIs

enablePayloadPrivateEncryption

enablePayloadPrivateEncryption

int enablePayloadPrivateEncryption	(boolean enabled
	TRTCCloudDef.TRTCPayloadPrivateEncryptionConfig config)

Enable or disable private encryption of media streams

In scenarios with high security requirements, TRTC recommends that you call the enablePayloadPrivateEncryption method to enable private encryption of media streams before joining a room.
After the user exits the room, the SDK will automatically close the private encryption. To re-enable private encryption, you need to call this method before the user joins the room again.

Param	DESC
config	Configure the algorithm and key for private encryption of media streams, please see TRTCPayloadPrivateEncryptionConfig.
enabled	Whether to enable media stream private encryption.

Note

TRTC has built-in encryption for media streams before transmission. After private encryption of media streams is enabled, it will be re-encrypted with the key and initial vector you pass in.

Return Desc:

Interface call result, 0: Method call succeeded, -1: The incoming parameter is invalid, -2: Your subscription has expired. If you want to renew it, Please update to RTC Engine Pro Plans and fill out application form. Approval is required before use.

TRTCCloudListener

Last updated：2024-06-06 15:26:15

Module: TRTCCloudListener @ TXLiteAVSDK

Function: event callback APIs for TRTC's video call feature

TRTCCloudListener

TRTCVideoRenderListener

FuncList	DESC
onRenderVideoFrame	Custom video rendering

TRTCVideoFrameListener

FuncList	DESC
onGLContextCreated	An OpenGL context was created in the SDK.
onProcessVideoFrame	Video processing by third-party beauty filters
onGLContextDestory	The OpenGL context in the SDK was destroyed

TRTCAudioFrameListener

FuncList	DESC
onCapturedAudioFrame	Audio data captured by the local mic and pre-processed by the audio module
onLocalProcessedAudioFrame	Audio data captured by the local mic, pre-processed by the audio module, effect-processed and BGM-mixed

onRemoteUserAudioFrame	Audio data of each remote user before audio mixing
onMixedPlayAudioFrame	Data mixed from each channel before being submitted to the system for playback
onMixedAllAudioFrame	Data mixed from all the captured and to-be-played audio in the SDK
onVoiceEarMonitorAudioFrame	In-ear monitoring data

TRTCLogListener

FuncList	DESC
onLog	Printing of local log

TRTCCloudListener

FuncList	DESC
onError	Error event callback
onWarning	Warning event callback
onEnterRoom	Whether room entry is successful
onExitRoom	Room exit
onSwitchRole	Role switching
onSwitchRoom	Result of room switching
onConnectOtherRoom	Result of requesting cross-room call
onDisConnectOtherRoom	Result of ending cross-room call
onUpdateOtherRoomForwardMode	Result of changing the upstream capability of the cross-room anchor
onRemoteUserEnterRoom	A user entered the room
onRemoteUserLeaveRoom	A user exited the room
onUserVideoAvailable	A remote user published/unpublished primary stream video
onUserSubStreamAvailable	A remote user published/unpublished substream video

onUserAudioAvailable	A remote user published/unpublished audio
onFirstVideoFrame	The SDK started rendering the first video frame of the local or a remote user
onFirstAudioFrame	The SDK started playing the first audio frame of a remote user
onSendFirstLocalVideoFrame	The first local video frame was published
onSendFirstLocalAudioFrame	The first local audio frame was published
onRemoteVideoStatusUpdated	Change of remote video status
onRemoteAudioStatusUpdated	Change of remote audio status
onUserVideoSizeChanged	Change of remote video size
onNetworkQuality	Real-time network quality statistics
onStatistics	Real-time statistics on technical metrics
onSpeedTestResult	Callback of network speed test
onConnectionLost	The SDK was disconnected from the cloud
onTryToReconnect	The SDK is reconnecting to the cloud
onConnectionRecovery	The SDK is reconnected to the cloud
onCameraDidReady	The camera is ready
onMicDidReady	The mic is ready
onAudioRouteChanged	The audio route changed (for mobile devices only)
onUserVoiceVolume	Volume
onRecvCustomCmdMsg	Receipt of custom message
onMissCustomCmdMsg	Loss of custom message
onRecvSEIMsg	Receipt of SEI message
onStartPublishing	Started publishing to Tencent Cloud CSS CDN
onStopPublishing	Stopped publishing to Tencent Cloud CSS CDN
onStartPublishCDNStream	Started publishing to non-Tencent Cloud’s live streaming CDN

onStopPublishCDNStream	Stopped publishing to non-Tencent Cloud’s live streaming CDN
onSetMixTranscodingConfig	Set the layout and transcoding parameters for On-Cloud MixTranscoding
onStartPublishMediaStream	Callback for starting to publish
onUpdatePublishMediaStream	Callback for modifying publishing parameters
onStopPublishMediaStream	Callback for stopping publishing
onCdnStreamStateChanged	Callback for change of RTMP/RTMPS publishing status
onScreenCaptureStarted	Screen sharing started
onScreenCapturePaused	Screen sharing was paused
onScreenCaptureResumed	Screen sharing was resumed
onScreenCaptureStopped	Screen sharing stopped
onLocalRecordBegin	Local recording started
onLocalRecording	Local media is being recorded
onLocalRecordFragment	Record fragment finished.
onLocalRecordComplete	Local recording stopped
onSnapshotComplete	Finished taking a local screenshot
onUserEnter	An anchor entered the room (disused)
onUserExit	An anchor left the room (disused)
onAudioEffectFinished	Audio effects ended (disused)
onSpeedTest	Result of server speed testing (disused)

onRenderVideoFrame

onRenderVideoFrame

void onRenderVideoFrame	(String userId
	int streamType
	TRTCCloudDef.TRTCVideoFrame frame)

Custom video rendering

If you have configured the callback of custom rendering for local or remote video, the SDK will return to you via this callback video frames that are otherwise sent to the rendering control, so that you can customize rendering.

Param	DESC
frame	Video frames to be rendered
streamType	Stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	`userId` of the video source. This parameter can be ignored if the callback is for local video (`setLocalVideoRenderDelegate`).

onGLContextCreated

onGLContextCreated

An OpenGL context was created in the SDK.

onProcessVideoFrame

onProcessVideoFrame

int onProcessVideoFrame	(TRTCCloudDef.TRTCVideoFrame srcFrame
	TRTCCloudDef.TRTCVideoFrame dstFrame)

Video processing by third-party beauty filters

If you use a third-party beauty filter component, you need to configure this callback in `TRTCCloud` to have the SDK return to you video frames that are otherwise pre-processed by TRTC.
You can then send the video frames to the third-party beauty filter component for processing. As the data returned can be read and modified, the result of processing can be synced to TRTC for subsequent encoding and publishing.

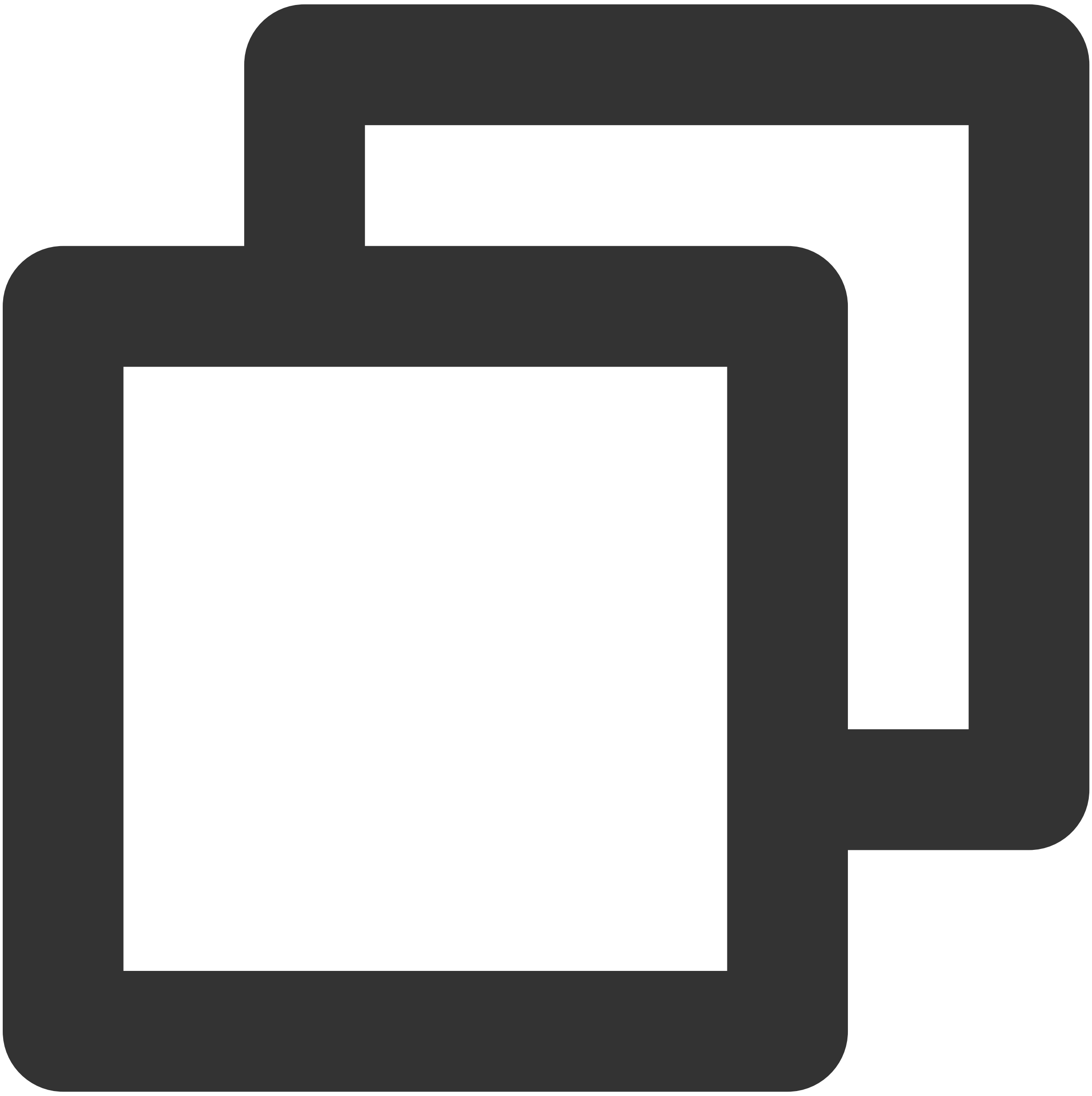
Case 1: the beauty filter component generates new textures
If the beauty filter component you use generates a frame of new texture (for the processed image) during image processing, please set `dstFrame.textureId` to the ID of the new texture in the callback function.

```
private final TRTCVideoFrameListener mVideoFrameListener = new TRTCVideoFrameListen
    @Override
    public void onGLContextCreated() {
        mFURenderer.onSurfaceCreated();
        mFURenderer.setUseTexAsync(true);
    }
    @Override
    public int onProcessVideoFrame(TRTCVideoFrame srcFrame, TRTCVideoFrame dstFrame
        dstFrame.texture.textureId = mFURenderer.onDrawFrameSingleInput(srcFrame.te
        return 0;
    }
```

```
    @Override
    public void onGLContextDestory() {
        mFURenderer.onSurfaceDestroyed();
    }
};
```

Case 2: you need to provide target textures to the beauty filter component

If the third-party beauty filter component you use does not generate new textures and you need to manually set an input texture and an output texture for the component, you can consider the following scheme:



```
int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame srcFrame, TRTCCloudDef.TRTCVide
```

```
    thirdparty_process(srcFrame.texture.textureId, srcFrame.width, srcFrame.height,
    return 0;
}
```

Param	DESC
dstFrame	Used to receive video images processed by third-party beauty filters
srcFrame	Used to carry images captured by TRTC via the camera

Note

Currently, only the OpenGL texture scheme is supported(PC supports TRTCVideoBufferType_Buffer format Only)

onGLContextDestory

onGLContextDestory

The OpenGL context in the SDK was destroyed

onCapturedAudioFrame

onCapturedAudioFrame

void onCapturedAudioFrame	(TRTCCloudDef.TRTCAudioFrame frame)

Audio data captured by the local mic and pre-processed by the audio module

After you configure the callback of custom audio processing, the SDK will return via this callback the data captured and pre-processed (ANS, AEC, and AGC) in PCM format.

The audio returned is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function. The SDK processes an audio frame every 20 ms, so if your operation takes more than 20 ms, it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.
3. The audio data is returned via this callback after ANS, AEC and AGC, but it **does not include** pre-processing effects like background music, audio effects, or reverb, and therefore has a short delay.

onLocalProcessedAudioFrame

onLocalProcessedAudioFrame

void onLocalProcessedAudioFrame	(TRTCCloudDef.TRTCAudioFrame frame)

Audio data captured by the local mic, pre-processed by the audio module, effect-processed and BGM-mixed

After you configure the callback of custom audio processing, the SDK will return via this callback the data captured, pre-processed (ANS, AEC, and AGC), effect-processed and BGM-mixed in PCM format, before it is submitted to the network module for encoding.

The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Instructions:

You could write data to the `TRTCAudioFrame.extraData` filed, in order to achieve the purpose of transmitting signaling.

Because the data block of the audio frame header cannot be too large, we recommend you limit the size of the signaling data to only a few bytes when using this API. If extra data more than 100 bytes, it won't be sent.

Other users in the room can receive the message through the `TRTCAudioFrame.extraData` in `onRemoteUserAudioFrame` callback in TRTCAudioFrameDelegate.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function. The SDK processes an audio frame every 20 ms, so if your operation takes more than 20 ms, it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.
3. Audio data is returned via this callback after ANS, AEC, AGC, effect-processing and BGM-mixing, and therefore the delay is longer than that with onCapturedAudioFrame.

onRemoteUserAudioFrame

onRemoteUserAudioFrame

void onRemoteUserAudioFrame	(TRTCCloudDef.TRTCAudioFrame frame
	String userId)

Audio data of each remote user before audio mixing

After you configure the callback of custom audio processing, the SDK will return via this callback the raw audio data (PCM format) of each remote user before mixing.

The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format
userId	User ID

Note

The audio data returned via this callback can be read but not modified.

onMixedPlayAudioFrame

onMixedPlayAudioFrame

void onMixedPlayAudioFrame	(TRTCCloudDef.TRTCAudioFrame frame)

Data mixed from each channel before being submitted to the system for playback

After you configure the callback of custom audio processing, the SDK will return to you via this callback the data (PCM format) mixed from each channel before it is submitted to the system for playback.

The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function. The SDK processes an audio frame every 20 ms, so if your operation takes more than 20 ms, it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.
3. The audio data returned via this callback is the audio data mixed from each channel before it is played. It does not include the in-ear monitoring data.

onMixedAllAudioFrame

onMixedAllAudioFrame

void onMixedAllAudioFrame	(TRTCCloudDef.TRTCAudioFrame frame)

Data mixed from all the captured and to-be-played audio in the SDK

After you configure the callback of custom audio processing, the SDK will return via this callback the data (PCM format) mixed from all captured and to-be-played audio in the SDK, so that you can customize recording.

The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits =**

1920 bytes.

Param	DESC
frame	Audio frames in PCM format

Note

1. This data returned via this callback is mixed from all audio in the SDK, including local audio after pre-processing (ANS, AEC, and AGC), special effects application, and music mixing, as well as all remote audio, but it does not include the in-ear monitoring data.
2. The audio data returned via this callback cannot be modified.

onVoiceEarMonitorAudioFrame

onVoiceEarMonitorAudioFrame

void onVoiceEarMonitorAudioFrame	(TRTCCloudDef.TRTCAudioFrame frame)

In-ear monitoring data

After you configure the callback of custom audio processing, the SDK will return to you via this callback the in-ear monitoring data (PCM format) before it is submitted to the system for playback.
The audio returned is in PCM format and has a not-fixed frame length (time).
The formula to convert a frame length in seconds to one in bytes is **sample rate** * **frame length in seconds** * **number of sound channels** * **audio bit depth**.
Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The length of 0.02s frame in bytes will be **48000** * **0.02s** * **1** * **16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function, or it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.

onLog

onLog

void onLog	(String log
	int level
	String module)

Printing of local log

If you want to capture the local log printing event, you can configure the log callback to have the SDK return to you via this callback all logs that are to be printed.

Param	DESC
level	Log level. For more information, please see `TRTC_LOG_LEVEL` .
log	Log content
module	Reserved field, which is not defined at the moment and has a fixed value of `TXLiteAVSDK` .

onError

onError

void onError	(int errCode
	String errMsg
	Bundle extraInfo)

Error event callback

Error event, which indicates that the SDK threw an irrecoverable error such as room entry failure or failure to start device

For more information, see Error Codes.

Param	DESC
errCode	Error code
errMsg	Error message
extInfo	Extended field. Certain error codes may carry extra information for troubleshooting.

onWarning

onWarning

void onWarning	(int warningCode
	String warningMsg
	Bundle extraInfo)

Warning event callback

Warning event, which indicates that the SDK threw an error requiring attention, such as video lag or high CPU usage
For more information, see Error Codes.

Param	DESC
extInfo	Extended field. Certain warning codes may carry extra information for troubleshooting.
warningCode	Warning code
warningMsg	Warning message

onEnterRoom

onEnterRoom

void onEnterRoom	(long result)

Whether room entry is successful

After calling the `enterRoom()` API in `TRTCCloud` to enter a room, you will receive the `onEnterRoom(result)` callback from `TRTCCloudDelegate` .
If room entry succeeded, `result` will be a positive number (`result` > 0), indicating the time in milliseconds (ms) the room entry takes.
If room entry failed, `result` will be a negative number (result < 0), indicating the error code for the failure.
For more information on the error codes for room entry failure, see Error Codes.

Param	DESC
result	If `result` is greater than 0, it indicates the time (in ms) the room entry takes; if `result` is less than 0, it represents the error code for room entry.

Note

1. In TRTC versions below 6.6, the `onEnterRoom(result)` callback is returned only if room entry succeeds, and the `onError()` callback is returned if room entry fails.
2. In TRTC 6.6 and above, the `onEnterRoom(result)` callback is returned regardless of whether room entry succeeds or fails, and the `onError()` callback is also returned if room entry fails.

onExitRoom

onExitRoom

void onExitRoom	(int reason)

Room exit

Calling the `exitRoom()` API in `TRTCCloud` will trigger the execution of room exit-related logic, such as releasing resources of audio/video devices and codecs.
After all resources occupied by the SDK are released, the SDK will return the `onExitRoom()` callback.

If you need to call `enterRoom()` again or switch to another audio/video SDK, please wait until you receive the `onExitRoom()` callback.
Otherwise, you may encounter problems such as the camera or mic being occupied.

Param	DESC
reason	Reason for room exit. `0` : the user called `exitRoom` to exit the room; `1` : the user was removed from the room by the server; `2` : the room was dismissed.

onSwitchRole

onSwitchRole

void onSwitchRole	(final int errCode
	final String errMsg)

Role switching

You can call the `switchRole()` API in `TRTCCloud` to switch between the anchor and audience roles.
This is accompanied by a line switching process.
After the switching, the SDK will return the `onSwitchRole()` event callback.

Param	DESC
errCode	Error code. `ERR_NULL` indicates a successful switch. For more information, please see Error Codes.
errMsg	Error message

onSwitchRoom

onSwitchRoom

void onSwitchRoom	(final int errCode
	final String errMsg)

Result of room switching

You can call the `switchRoom()` API in `TRTCCloud` to switch from one room to another.
After the switching, the SDK will return the `onSwitchRoom()` event callback.

Param	DESC
errCode	Error code. `ERR_NULL` indicates a successful switch. For more information, please see Error Codes.
errMsg	Error message

onConnectOtherRoom

onConnectOtherRoom

void onConnectOtherRoom	(final String userId
	final int errCode
	final String errMsg)

Result of requesting cross-room call

You can call the `connectOtherRoom()` API in `TRTCCloud` to establish a video call with the anchor of another room. This is the “anchor competition” feature.
The caller will receive the `onConnectOtherRoom()` callback, which can be used to determine whether the cross-room call is successful.

If it is successful, all users in either room will receive the `onUserVideoAvailable()` callback from the anchor of the other room.

Param	DESC
errCode	Error code. `ERR_NULL` indicates that cross-room connection is established successfully. For more information, please see Error Codes.
errMsg	Error message
userId	The user ID of the anchor (in another room) to be called

onDisConnectOtherRoom

onDisConnectOtherRoom

void onDisConnectOtherRoom	(final int errCode
	final String errMsg)

Result of ending cross-room call

onUpdateOtherRoomForwardMode

onUpdateOtherRoomForwardMode

void onUpdateOtherRoomForwardMode	(final int errCode
	final String errMsg)

Result of changing the upstream capability of the cross-room anchor

onRemoteUserEnterRoom

onRemoteUserEnterRoom

void onRemoteUserEnterRoom	(String userId)

A user entered the room

Due to performance concerns, this callback works differently in different scenarios (i.e., `AppScene`, which you can specify by setting the second parameter when calling `enterRoom`).

Live streaming scenarios (`TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom`): in live streaming scenarios, a user is either in the role of an anchor or audience. The callback is returned only when an anchor enters the room.

Call scenarios (`TRTCAppSceneVideoCall` or `TRTCAppSceneAudioCall`): in call scenarios, the concept of roles does not apply (all users can be considered as anchors), and the callback is returned when any user enters the room.

Param	DESC
userId	User ID of the remote user

Note

1. The `onRemoteUserEnterRoom` callback indicates that a user entered the room, but it does not necessarily mean that the user enabled audio or video.
2. If you want to know whether a user enabled video, we recommend you use the `onUserVideoAvailable()` callback.

onRemoteUserLeaveRoom

onRemoteUserLeaveRoom

void onRemoteUserLeaveRoom	(String userId
	int reason)

A user exited the room

As with `onRemoteUserEnterRoom`, this callback works differently in different scenarios (i.e., `AppScene`, which you can specify by setting the second parameter when calling `enterRoom`).

Live streaming scenarios (`TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom`): the callback is triggered only when an anchor exits the room.

Call scenarios (`TRTCAppSceneVideoCall` or `TRTCAppSceneAudioCall`): in call scenarios, the concept of roles does not apply, and the callback is returned when any user exits the room.

Param	DESC
reason	Reason for room exit. `0`: the user exited the room voluntarily; `1`: the user exited the room due to timeout; `2`: the user was removed from the room; `3`: the anchor user exited the room due to switch to audience.

userId	User ID of the remote user

onUserVideoAvailable

onUserVideoAvailable

void onUserVideoAvailable	(String userId
	boolean available)

A remote user published/unpublished primary stream video

The primary stream is usually used for camera images. If you receive the `onUserVideoAvailable(userId, true)` callback, it indicates that the user has available primary stream video.
You can then call startRemoteView to subscribe to the remote user's video. If the subscription is successful, you will receive the `onFirstVideoFrame(userid)` callback, which indicates that the first video frame of the user is rendered.

If you receive the `onUserVideoAvailable(userId, false)` callback, it indicates that the video of the remote user is disabled, which may be because the user called muteLocalVideo or stopLocalPreview.

Param	DESC
available	Whether the user published (or unpublished) primary stream video. `true` : published; `false` : unpublished
userId	User ID of the remote user

onUserSubStreamAvailable

onUserSubStreamAvailable

void onUserSubStreamAvailable	(String userId
	boolean available)

A remote user published/unpublished substream video

The substream is usually used for screen sharing images. If you receive the `onUserSubStreamAvailable(userId, true)` callback, it indicates that the user has available substream video.

You can then call startRemoteView to subscribe to the remote user's video. If the subscription is successful, you will receive the `onFirstVideoFrame(userid)` callback, which indicates that the first frame of the user is rendered.

Param	DESC
available	Whether the user published (or unpublished) substream video. `true` : published; `false` : unpublished
userId	User ID of the remote user

Note

The API used to display substream images is startRemoteView, not startRemoteSubStreamView, startRemoteSubStreamView is deprecated.

onUserAudioAvailable

onUserAudioAvailable

void onUserAudioAvailable	(String userId
	boolean available)

A remote user published/unpublished audio

If you receive the `onUserAudioAvailable(userId, true)` callback, it indicates that the user published audio.

In auto-subscription mode, the SDK will play the user's audio automatically.

In manual subscription mode, you can call muteRemoteAudio(userid, false) to play the user's audio.

Param	DESC
available	Whether the user published (or unpublished) audio. `true` : published; `false` : unpublished
userId	User ID of the remote user

Note

The auto-subscription mode is used by default. You can switch to the manual subscription mode by calling setDefaultStreamRecvMode, but it must be called before room entry for the switch to take effect.

onFirstVideoFrame

onFirstVideoFrame

void onFirstVideoFrame	(String userId
	int streamType
	int width
	int height)

The SDK started rendering the first video frame of the local or a remote user

The SDK returns this event callback when it starts rendering your first video frame or that of a remote user. The `userId` in the callback can help you determine whether the frame is yours or a remote user's.
If `userId` is empty, it indicates that the SDK has started rendering your first video frame. The precondition is that you have called startLocalPreview or startScreenCapture.
If `userId` is not empty, it indicates that the SDK has started rendering the first video frame of a remote user. The precondition is that you have called startRemoteView to subscribe to the user's video.

Param	DESC
height	Video height
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	The user ID of the local or a remote user. If it is empty, it indicates that the first local video frame is available; if it is not empty, it indicates that the first video frame of a remote user is available.
width	Video width

Note

1. The callback of the first local video frame being rendered is triggered only after you call startLocalPreview or startScreenCapture.
2. The callback of the first video frame of a remote user being rendered is triggered only after you call startRemoteView or startRemoteSubStreamView.

onFirstAudioFrame

onFirstAudioFrame

void onFirstAudioFrame	(String userId)

The SDK started playing the first audio frame of a remote user

The SDK returns this callback when it plays the first audio frame of a remote user. The callback is not returned for the playing of the first audio frame of the local user.

Param	DESC
userId	User ID of the remote user

onSendFirstLocalVideoFrame

onSendFirstLocalVideoFrame

void onSendFirstLocalVideoFrame	(int streamType)

The first local video frame was published

After you enter a room and call startLocalPreview or startScreenCapture to enable local video capturing (whichever happens first),
the SDK will start video encoding and publish the local video data via its network module to the cloud.
It returns the `onSendFirstLocalVideoFrame` callback after publishing the first local video frame.

Param	DESC
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.

onSendFirstLocalAudioFrame

onSendFirstLocalAudioFrame

The first local audio frame was published

After you enter a room and call startLocalAudio to enable audio capturing (whichever happens first),
the SDK will start audio encoding and publish the local audio data via its network module to the cloud.
The SDK returns the `onSendFirstLocalAudioFrame` callback after sending the first local audio frame.

onRemoteVideoStatusUpdated

onRemoteVideoStatusUpdated

void onRemoteVideoStatusUpdated	(String userId
	int streamType
	int status
	int reason
	Bundle extraInfo)

Change of remote video status

You can use this callback to get the status (`Playing` , `Loading` , or `Stopped`) of the video of each remote user and display it on the UI.

Param	DESC
extraInfo	Extra information
reason	Reason for the change of status
status	Video status, which may be `Playing` , `Loading` , or `Stopped`
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	User ID

onRemoteAudioStatusUpdated

onRemoteAudioStatusUpdated

void onRemoteAudioStatusUpdated	(String userId
	int status
	int reason
	Bundle extraInfo)

Change of remote audio status

You can use this callback to get the status (`Playing` , `Loading` , or `Stopped`) of the audio of each remote user and display it on the UI.

Param	DESC
extraInfo	Extra information
reason	Reason for the change of status
status	Audio status, which may be `Playing` , `Loading` , or `Stopped`
userId	User ID

onUserVideoSizeChanged

onUserVideoSizeChanged

void onUserVideoSizeChanged	(String userId
	int streamType
	int newWidth
	int newHeight)

Change of remote video size

If you receive the `onUserVideoSizeChanged(userId, streamtype, newWidth, newHeight)` callback, it indicates that the user changed the video size. It may be triggered by `setVideoEncoderParam` or `setSubStreamEncoderParam` .

Param	DESC
newHeight	Video height
newWidth	Video width
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	User ID

onNetworkQuality

onNetworkQuality

void onNetworkQuality	(TRTCCloudDef.TRTCQuality localQuality
	ArrayList<TRTCCloudDef.TRTCQuality> remoteQuality)

Real-time network quality statistics

This callback is returned every 2 seconds and notifies you of the upstream and downstream network quality detected by the SDK.

The SDK uses a built-in proprietary algorithm to assess the current latency, bandwidth, and stability of the network and returns a result.

If the result is `1` (excellent), it means that the current network conditions are excellent; if it is `6` (down), it means that the current network conditions are too bad to support TRTC calls.

Param	DESC
localQuality	Upstream network quality
remoteQuality	Downstream network quality, it refers to the data quality finally measured on the local side after the data flow passes through a complete transmission link of "remote ->cloud ->local". Therefore, the downlink network quality here represents the joint impact of the remote uplink and the local downlink.

Note

The uplink quality of remote users cannot be determined independently through this interface.

onStatistics

onStatistics

void onStatistics	(TRTCStatistics statistics)

Real-time statistics on technical metrics

This callback is returned every 2 seconds and notifies you of the statistics on technical metrics related to video, audio, and network. The metrics are listed in TRTCStatistics:

Video statistics: video resolution (`resolution`), frame rate (`FPS`), bitrate (`bitrate`), etc.

Audio statistics: audio sample rate (`samplerate`), number of audio channels (`channel`), bitrate (`bitrate`), etc.

Network statistics: the round trip time (`rtt`) between the SDK and the cloud (SDK -> Cloud -> SDK), package loss rate (`loss`), upstream traffic (`sentBytes`), downstream traffic (`receivedBytes`), etc.

Param	DESC
statistics	Statistics, including local statistics and the statistics of remote users. For details, please see TRTCStatistics.

Note

If you want to learn about only the current network quality and do not want to spend much time analyzing the statistics returned by this callback, we recommend you use onNetworkQuality.

onSpeedTestResult

onSpeedTestResult

void onSpeedTestResult	(TRTCCloudDef.TRTCSpeedTestResult result)

Callback of network speed test

The callback is triggered by startSpeedTest:.

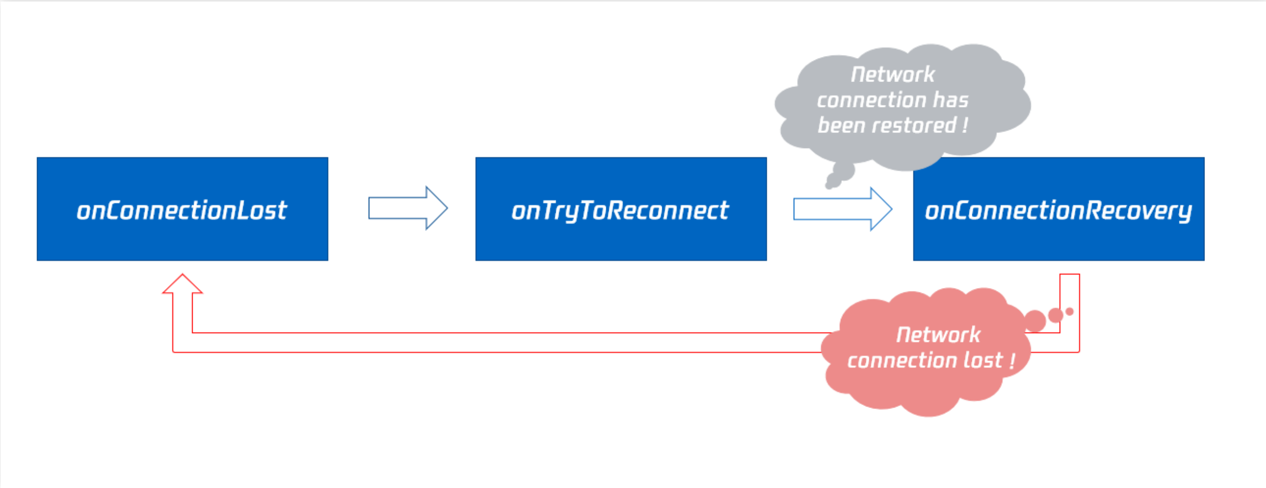
Param	DESC
result	Speed test data, including loss rates, rtt and bandwidth rates, please refer to TRTCSpeedTestResult for details.

onConnectionLost

onConnectionLost

The SDK was disconnected from the cloud

The SDK returns this callback when it is disconnected from the cloud, which may be caused by network unavailability or change of network, for example, when the user walks into an elevator.
After returning this callback, the SDK will attempt to reconnect to the cloud, and will return the onTryToReconnect callback. When it is reconnected, it will return the onConnectionRecovery callback.
In other words, the SDK proceeds from one event to the next in the following order:



onTryToReconnect

onTryToReconnect

The SDK is reconnecting to the cloud

When the SDK is disconnected from the cloud, it returns the onConnectionLost callback. It then attempts to reconnect and returns this callback (onTryToReconnect). After it is reconnected, it returns the onConnectionRecovery callback.

onConnectionRecovery

onConnectionRecovery

The SDK is reconnected to the cloud

When the SDK is disconnected from the cloud, it returns the onConnectionLost callback. It then attempts to reconnect and returns the onTryToReconnect callback. After it is reconnected, it returns this callback (onConnectionRecovery).

onCameraDidReady

onCameraDidReady

The camera is ready

After you call startLocalPreivew, the SDK will try to start the camera and return this callback if the camera is started.

If it fails to start the camera, it's probably because the application does not have access to the camera or the camera is being used.
You can capture the onError callback to learn about the exception and let users know via UI messages.

onMicDidReady

onMicDidReady

The mic is ready

After you call startLocalAudio, the SDK will try to start the mic and return this callback if the mic is started.
If it fails to start the mic, it's probably because the application does not have access to the mic or the mic is being used.
You can capture the onError callback to learn about the exception and let users know via UI messages.

onAudioRouteChanged

onAudioRouteChanged

void onAudioRouteChanged	(int newRoute
	int oldRoute)

The audio route changed (for mobile devices only)

Audio route is the route (speaker or receiver) through which audio is played.
When audio is played through the receiver, the volume is relatively low, and the sound can be heard only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.
When audio is played through the speaker, the volume is relatively high, and there is no need to put the phone near the ear. This mode enables the "hands-free" feature.
When audio is played through the wired earphone.
When audio is played through the bluetooth earphone.
When audio is played through the USB sound card.

Param	DESC
fromRoute	The audio route used before the change
route	Audio route, i.e., the route (speaker or receiver) through which audio is played

onUserVoiceVolume

onUserVoiceVolume

void onUserVoiceVolume	(ArrayList<TRTCCloudDef.TRTCVolumeInfo> userVolumes
	int totalVolume)

Volume

The SDK can assess the volume of each channel and return this callback on a regular basis. You can display, for example, a waveform or volume bar on the UI based on the statistics returned.

You need to first call enableAudioVolumeEvaluation to enable the feature and set the interval for the callback.

Note that the SDK returns this callback at the specified interval regardless of whether someone is speaking in the room.

Param	DESC
totalVolume	The total volume of all remote users. Value range: 0-100
userVolumes	An array that represents the volume of all users who are speaking in the room. Value range: 0-100

Note

`userVolumes` is an array. If `userId` is empty, the elements in the array represent the volume of the local user's audio. Otherwise, they represent the volume of a remote user's audio.

onRecvCustomCmdMsg

onRecvCustomCmdMsg

void onRecvCustomCmdMsg	(String userId
	int cmdID
	int seq
	byte[] message)

Receipt of custom message

When a user in a room uses sendCustomCmdMsg to send a custom message, other users in the room can receive the message through the `onRecvCustomCmdMsg` callback.

Param	DESC
cmdID	Command ID
message	Message data
seq	Message serial number
userId	User ID

onMissCustomCmdMsg

onMissCustomCmdMsg

void onMissCustomCmdMsg	(String userId
	int cmdID
	int errCode
	int missed)

Loss of custom message

When you use sendCustomCmdMsg to send a custom UDP message, even if you enable reliable transfer (by setting `reliable` to `true`), there is still a chance of message loss. Reliable transfer only helps maintain a low probability of message loss, which meets the reliability requirements in most cases.
If the sender sets `reliable` to `true` , the SDK will use this callback to notify the recipient of the number of custom messages lost during a specified time period (usually 5s) in the past.

Param	DESC
cmdID	Command ID
errCode	Error code
missed	Number of lost messages
userId	User ID

Note

The recipient receives this callback only if the sender sets `reliable` to `true` .

onRecvSEIMsg

onRecvSEIMsg

void onRecvSEIMsg	(String userId
	byte[] data)

Receipt of SEI message

If a user in the room uses sendSEIMsg to send an SEI message via video frames, other users in the room can receive the message through the `onRecvSEIMsg` callback.

Param	DESC
message	Data
userId	User ID

onStartPublishing

onStartPublishing

void onStartPublishing	(int err
	String errMsg)

Started publishing to Tencent Cloud CSS CDN

When you call startPublishing to publish streams to Tencent Cloud CSS CDN, the SDK will sync the command to the CVM immediately.
The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onStopPublishing

onStopPublishing

void onStopPublishing	(int err
	String errMsg)

Stopped publishing to Tencent Cloud CSS CDN

When you call stopPublishing to stop publishing streams to Tencent Cloud CSS CDN, the SDK will sync the command to the CVM immediately.

The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onStartPublishCDNStream

onStartPublishCDNStream

void onStartPublishCDNStream	(int err
	String errMsg)

Started publishing to non-Tencent Cloud's live streaming CDN

When you call startPublishCDNStream to start publishing streams to a non-Tencent Cloud's live streaming CDN, the SDK will sync the command to the CVM immediately.

The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

Note

If you receive a callback that the command is executed successfully, it only means that your command was sent to Tencent Cloud's backend server. If the CDN vendor does not accept your streams, the publishing will still fail.

onStopPublishCDNStream

onStopPublishCDNStream

void onStopPublishCDNStream	(int err
	String errMsg)

Stopped publishing to non-Tencent Cloud's live streaming CDN

When you call stopPublishCDNStream to stop publishing to a non-Tencent Cloud's live streaming CDN, the SDK will sync the command to the CVM immediately.
The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onSetMixTranscodingConfig

onSetMixTranscodingConfig

void onSetMixTranscodingConfig	(int err
	String errMsg)

Set the layout and transcoding parameters for On-Cloud MixTranscoding

When you call setMixTranscodingConfig to modify the layout and transcoding parameters for On-Cloud MixTranscoding, the SDK will sync the command to the CVM immediately.
The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onStartPublishMediaStream

onStartPublishMediaStream

void onStartPublishMediaStream	(String taskId

	int code
	String message
	Bundle extraInfo)

Callback for starting to publish

When you call startPublishMediaStream to publish a stream to the TRTC backend, the SDK will immediately update the command to the cloud server.

The SDK will then receive the publishing result from the cloud server and will send the result to you via this callback.

Param	DESC
code	: `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The callback information.
taskId	: If a request is successful, a task ID will be returned via the callback. You need to provide this task ID when you call updatePublishMediaStream to modify publishing parameters or stopPublishMediaStream to stop publishing.

onUpdatePublishMediaStream

onUpdatePublishMediaStream

void onUpdatePublishMediaStream	(String taskId
	int code
	String message
	Bundle extraInfo)

Callback for modifying publishing parameters

When you call updatePublishMediaStream to modify publishing parameters, the SDK will immediately update the command to the cloud server.

The SDK will then receive the modification result from the cloud server and will send the result to you via this callback.

Param	DESC
code	: `0` : Successful; other values: Failed.

extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The callback information.
taskId	: The task ID you pass in when calling updatePublishMediaStream, which is used to identify a request.

onStopPublishMediaStream

onStopPublishMediaStream

void onStopPublishMediaStream	(String taskId
	int code
	String message
	Bundle extraInfo)

Callback for stopping publishing

When you call stopPublishMediaStream to stop publishing, the SDK will immediately update the command to the cloud server.

The SDK will then receive the modification result from the cloud server and will send the result to you via this callback.

Param	DESC
code	: `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The callback information.
taskId	: The task ID you pass in when calling stopPublishMediaStream, which is used to identify a request.

onCdnStreamStateChanged

onCdnStreamStateChanged

void onCdnStreamStateChanged	(String cdnUrl

	int status
	int code
	String msg
	Bundle extraInfo)

Callback for change of RTMP/RTMPS publishing status

When you call startPublishMediaStream to publish a stream to the TRTC backend, the SDK will immediately update the command to the cloud server.

If you set the publishing destination (TRTCPublishTarget) to the URL of Tencent Cloud or a third-party CDN, you will be notified of the RTMP/RTMPS publishing status via this callback.

Param	DESC
cdnUrl	: The URL you specify in TRTCPublishTarget when you call startPublishMediaStream.
code	: The publishing result. `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The publishing information.
status	: The publishing status. 0: The publishing has not started yet or has ended. This value will be returned after you call stopPublishMediaStream. 1: The TRTC server is connecting to the CDN server. If the first attempt fails, the TRTC backend will retry multiple times and will return this value via the callback (every five seconds). After publishing succeeds, the value `2` will be returned. If a server error occurs or publishing is still unsuccessful after 60 seconds, the value `4` will be returned. 2: The TRTC server is publishing to the CDN. This value will be returned if the publishing succeeds. 3: The TRTC server is disconnected from the CDN server and is reconnecting. If a CDN error occurs or publishing is interrupted, the TRTC backend will try to reconnect and resume publishing and will return this value via the callback (every five seconds). After publishing resumes, the value `2` will be returned. If a server error occurs or the attempt to resume publishing is still unsuccessful after 60 seconds, the value `4` will be returned. 4: The TRTC server is disconnected from the CDN server and failed to reconnect within the timeout period. In this case, the publishing is deemed to have failed. You can call updatePublishMediaStream to try again. 5: The TRTC server is disconnecting from the CDN server. After you call stopPublishMediaStream, the SDK will return this value first and then the value `0` .

onScreenCaptureStarted

onScreenCaptureStarted

Screen sharing started

The SDK returns this callback when you call startScreenCapture and other APIs to start screen sharing.

onScreenCapturePaused

onScreenCapturePaused

Screen sharing was paused

The SDK returns this callback when you call pauseScreenCapture to pause screen sharing.

onScreenCaptureResumed

onScreenCaptureResumed

Screen sharing was resumed

The SDK returns this callback when you call resumeScreenCapture to resume screen sharing.

onScreenCaptureStopped

onScreenCaptureStopped

void onScreenCaptureStopped	(int reason)

Screen sharing stopped

The SDK returns this callback when you call stopScreenCapture to stop screen sharing.

Param	DESC
reason	Reason. `0` : the user stopped screen sharing; `1` : screen sharing stopped because the shared window was closed.

onLocalRecordBegin

onLocalRecordBegin

void onLocalRecordBegin	(int errCode
	String storagePath)

Local recording started

When you call startLocalRecording to start local recording, the SDK returns this callback to notify you whether recording is started successfully.

Param	DESC
errCode	status. 0: successful. -1: failed. -2: unsupported format. -6: recording has been started. Stop recording first. -7: recording file already exists and needs to be deleted. -8: recording directory does not have the write permission. Please check the directory permission.
storagePath	Storage path of recording file

onLocalRecording

onLocalRecording

void onLocalRecording	(long duration
	String storagePath)

Local media is being recorded

The SDK returns this callback regularly after local recording is started successfully via the calling of startLocalRecording.

You can capture this callback to stay up to date with the status of the recording task.

You can set the callback interval when calling startLocalRecording.

Param	DESC
duration	Cumulative duration of recording, in milliseconds
storagePath	Storage path of recording file

onLocalRecordFragment

onLocalRecordFragment

void onLocalRecordFragment	(String storagePath)

Record fragment finished.

When fragment recording is enabled, this callback will be invoked when each fragment file is finished.

Param	DESC
storagePath	Storage path of the fragment.

onLocalRecordComplete

onLocalRecordComplete

void onLocalRecordComplete	(int errCode
	String storagePath)

Local recording stopped

When you call stopLocalRecording to stop local recording, the SDK returns this callback to notify you of the recording result.

Param	DESC
errCode	status 0: successful. -1: failed. -2: Switching resolution or horizontal and vertical screen causes the recording to stop. -3: recording duration is too short or no video or audio data is received. Check the recording duration or whether audio or video capture is enabled.
storagePath	Storage path of recording file

onSnapshotComplete

onSnapshotComplete

void onSnapshotComplete	(Bitmap bmp)

Finished taking a local screenshot

Param	DESC
bmp	Screenshot result. If it is `null` , the screenshot failed to be taken.
data	Screenshot data. If it is `nullptr` , it indicates that the SDK failed to take the screenshot.
format	Screenshot data format. Only `TRTCVideoPixelFormat_BGRA32` is supported now.
height	Screenshot height
length	Screenshot data length. In BGRA32 format, length = width * height * 4.
type	Video stream type
userId	User ID. If it is empty, the screenshot is a local image.
width	Screenshot width

Note

The parameters of the full-platform C++ interface and the Java interface are different. The C++ interface uses 7 parameters to describe a screenshot, while the Java interface uses only one Bitmap to describe a screenshot.

onUserEnter

onUserEnter

void onUserEnter	(String userId)

An anchor entered the room (disused)

@deprecated This callback is not recommended in the new version. Please use onRemoteUserEnterRoom instead.

onUserExit

onUserExit

void onUserExit	(String userId
	int reason)

An anchor left the room (disused)

@deprecated This callback is not recommended in the new version. Please use onRemoteUserLeaveRoom instead.

onAudioEffectFinished

onAudioEffectFinished

void onAudioEffectFinished	(int effectId
	int code)

Audio effects ended (disused)

@deprecated This callback is not recommended in the new version. Please use ITXAudioEffectManager instead.
Audio effects and background music can be started using the same API (startPlayMusic) now instead of separate ones.

onSpeedTest

onSpeedTest

void onSpeedTest	(TRTCCloudDef.TRTCSpeedTestResult currentResult
	int finishedCount
	int totalCount)

Result of server speed testing (disused)

@deprecated This callback is not recommended in the new version. Please use onSpeedTestResult: instead.

TRTCStatistics

Last updated：2024-06-06 15:26:15

Module: TRTC audio/video metrics (read-only)

Function: the TRTC SDK reports to you the current real-time audio/video metrics (frame rate, bitrate, lag, etc.) once every two seconds

TRTCStatistics

StructType

FuncList	DESC
TRTCLocalStatistics	Local audio/video metrics
TRTCRemoteStatistics	Remote audio/video metrics
TRTCStatistics	Network and performance metrics

TRTCLocalStatistics

TRTCLocalStatistics

Local audio/video metrics

EnumType	DESC
audioBitrate	Field description: local audio bitrate in Kbps, i.e., how much audio data is generated per second
audioCaptureState	Field description:Audio equipment collection status(0：Normal；1：Long silence detected；2：Broken sound detected；3：Abnormal intermittent sound detected;)
audioSampleRate	Field description: local audio sample rate (Hz)
frameRate	Field description: local video frame rate in fps, i.e., how many video frames there

	are per second
height	Field description: local video height in px
streamType	Field description: video stream type (HD big image | smooth small image | substream image)
videoBitrate	Field description: local video bitrate in Kbps, i.e., how much video data is generated per second
width	Field description: local video width in px

TRTCRemoteStatistics

TRTCRemoteStatistics

Remote audio/video metrics

EnumType	DESC
audioBitrate	Field description: local audio bitrate (Kbps)
audioBlockRate	Field description: audio playback lag rate (%) Audio playback lag rate (audioBlockRate) = cumulative audio playback lag duration (audioTotalBlockTime)/total audio playback duration
audioPacketLoss	Field description: total packet loss rate (%) of the audio stream `audioPacketLoss` represents the packet loss rate eventually calculated on the audience side after the audio/video stream goes through the complete transfer linkage of "anchor -> cloud -> audience". The smaller the `audioPacketLoss` , the better. The packet loss rate of 0 indicates that all data of the audio stream has entirely reached the audience. If `downLoss` is `0` but `audioPacketLoss` isn't, there is no packet loss on the linkage of "cloud -> audience" for the audiostream, but there are unrecoverable packet losses on the linkage of "anchor -> cloud".
audioSampleRate	Field description: local audio sample rate (Hz)
audioTotalBlockTime	Field description: cumulative audio playback lag duration (ms)
finalLoss	Field description: total packet loss rate (%) of the audio/video stream Deprecated, please use audioPacketLoss and videoPacketLoss instead.
frameRate	Field description: remote video frame rate (fps)

height	Field description: remote video height in px
jitterBufferDelay	Field description: playback delay (ms) In order to avoid audio/video lags caused by network jitters and network packet disorders, TRTC maintains a playback buffer on the playback side to organize the received network data packets. The size of the buffer is adaptively adjusted according to the current network quality and converted to the length of time in milliseconds, i.e., `jitterBufferDelay` .
point2PointDelay	Field description: end-to-end delay (ms) `point2PointDelay` represents the delay of "anchor -> cloud -> audience". To be more precise, it represents the delay of the entire linkage of "collection -> encoding -> network transfer -> receiving -> buffering -> decoding -> playback". `point2PointDelay` works only if both the local and remote SDKs are on version 8.5 or above. If the remote SDK is on a version below 8.5, this value will always be 0 and thus meaningless.
remoteNetworkRTT	Field description: round-trip delay (ms) from the SDK to cloud This value represents the total time it takes to send a network packet from the SDK to the cloud and then send a network packet back from the cloud to the SDK, i.e., the total time it takes for a network packet to go through the linkage of "SDK -> cloud -> SDK". The smaller the value, the better. If `remoteNetworkRTT` is below 50 ms, it means a short audio/video call delay; if `remoteNetworkRTT` is above 200 ms, it means a long audio/video call delay. It should be explained that `remoteNetworkRTT` represents the total time spent on the linkage of "SDK -> cloud -> SDK"; therefore, there is no need to distinguish between `remoteNetworkUpRTT` and `remoteNetworkDownRTT` .
remoteNetworkUplinkLoss	Field description: upstream packet loss rate (%) from the SDK to cloud The smaller the value, the better. If `remoteNetworkUplinkLoss` is `0%` , the upstream network quality is very good, and the data packets uploaded to the cloud are basically not lost. If `remoteNetworkUplinkLoss` is `30%` , 30% of the audio/video data packets sent to the cloud by the SDK are lost on the transfer linkage.
streamType	Field description: video stream type (HD big image | smooth small image | substream image)
userId	Field description: user ID

videoBitrate	Field description: remote video bitrate (Kbps)
videoBlockRate	Field description: video playback lag rate (%) Video playback lag rate (videoBlockRate) = cumulative video playback lag duration (videoTotalBlockTime)/total video playback duration
videoPacketLoss	Field description: total packet loss rate (%) of the video stream `videoPacketLoss` represents the packet loss rate eventually calculated on the audience side after the audio/video stream goes through the complete transfer linkage of "anchor -> cloud -> audience". The smaller the `videoPacketLoss` , the better. The packet loss rate of 0 indicates that all data of the video stream has entirely reached the audience. If `downLoss` is `0` but `videoPacketLoss` isn't, there is no packet loss on the linkage of "cloud -> audience" for the video stream, but there are unrecoverable packet losses on the linkage of "anchor -> cloud".
videoTotalBlockTime	Field description: cumulative video playback lag duration (ms)
width	Field description: remote video width in px

TRTCStatistics

TRTCStatistics

Network and performance metrics

EnumType	DESC
appCpu	Field description: CPU utilization (%) of the current application, Android 8.0 and above systems are not supported
downLoss	Field description: downstream packet loss rate (%) from cloud to the SDK The smaller the value, the better. If `downLoss` is `0%` , the downstream network quality is very good, and the data packets received from the cloud are basically not lost. If `downLoss` is `30%` , 30% of the audio/video data packets sent to the SDK by the cloud are lost on the transfer linkage.
gatewayRtt	Field description: round-trip delay (ms) from the SDK to gateway This value represents the total time it takes to send a network packet from the SDK to the gateway and then send a network packet back from the gateway to the SDK, i.e., the total time it takes for a network packet to go through the linkage of "SDK -> gateway -> SDK".

	The smaller the value, the better. If `gatewayRtt` is below 50 ms, it means a short audio/video call delay; if `gatewayRtt` is above 200 ms, it means a long audio/video call delay. It should be explained that `gatewayRtt` is invalid for cellular network.
localArray	Field description: local audio/video statistics As there may be three local audio/video streams (i.e., HD big image, smooth small image, and substream image), the local audio/video statistics are an array.
receiveBytes	Field description: total number of received bytes (including signaling data and audio/video data)
remoteArray	Field description: remote audio/video statistics As there may be multiple concurrent remote users, and each of them may have multiple concurrent audio/video streams (i.e., HD big image, smooth small image, and substream image), the remote audio/video statistics are an array.
rtt	Field description: round-trip delay (ms) from the SDK to cloud This value represents the total time it takes to send a network packet from the SDK to the cloud and then send a network packet back from the cloud to the SDK, i.e., the total time it takes for a network packet to go through the linkage of "SDK -> cloud -> SDK". The smaller the value, the better. If `rtt` is below 50 ms, it means a short audio/video call delay; if `rtt` is above 200 ms, it means a long audio/video call delay. It should be explained that `rtt` represents the total time spent on the linkage of "SDK -> cloud -> SDK"; therefore, there is no need to distinguish between `upRtt` and `downRtt` .
sendBytes	Field description: total number of sent bytes (including signaling data and audio/video data)
systemCpu	Field description: CPU utilization (%) of the current system, Android 8.0 and above systems are not supported
upLoss	Field description: upstream packet loss rate (%) from the SDK to cloud The smaller the value, the better. If `upLoss` is `0%` , the upstream network quality is very good, and the data packets uploaded to the cloud are basically not lost. If `upLoss` is `30%` , 30% of the audio/video data packets sent to the cloud by the SDK are lost on the transfer linkage.

TXAudioEffectManager

Last updated：2024-06-06 15:26:15

Module: management class for background music, short audio effects, and voice effects

Description: sets background music, short audio effects, and voice effects

TXAudioEffectManager

TXMusicPreloadObserver

FuncList	DESC
onLoadProgress	Background music preload progress
onLoadError	Background music preload error

TXMusicPlayObserver

FuncList	DESC
onStart	Background music started.
onPlayProgress	Playback progress of background music
onComplete	Background music ended

TXAudioEffectManager

FuncList	DESC
enableVoiceEarMonitor	Enabling in-ear monitoring
setVoiceEarMonitorVolume	Setting in-ear monitoring volume

setVoiceReverbType	Setting voice reverb effects
setVoiceChangerType	Setting voice changing effects
setVoiceCaptureVolume	Setting speech volume
setVoicePitch	Setting speech pitch
setMusicObserver	Setting the background music callback
startPlayMusic	Starting background music
stopPlayMusic	Stopping background music
pausePlayMusic	Pausing background music
resumePlayMusic	Resuming background music
setAllMusicVolume	Setting the local and remote playback volume of background music
setMusicPublishVolume	Setting the remote playback volume of a specific music track
setMusicPlayoutVolume	Setting the local playback volume of a specific music track
setMusicPitch	Adjusting the pitch of background music
setMusicSpeedRate	Changing the speed of background music
getMusicCurrentPosInMS	Getting the playback progress (ms) of background music
getMusicDurationInMS	Getting the total length (ms) of background music
seekMusicToPosInMS	Setting the playback progress (ms) of background music
setMusicScratchSpeedRate	Adjust the speed change effect of the scratch disc
setPreloadObserver	Setting music preload callback
preloadMusic	Preload background music
getMusicTrackCount	Get the number of tracks of background music
setMusicTrack	Specify the playback track of background music

StructType

FuncList	DESC

AudioMusicParam	Background music playback information

EnumType

EnumType	DESC
TXVoiceReverbType	Reverb effects
TXVoiceChangerType	Voice changing effects

onLoadProgress

onLoadProgress

void onLoadProgress	(int id
	int progress)

Background music preload progress

onLoadError

onLoadError

void onLoadError	(int id
	int errorCode)

Background music preload error

Param	DESC
errorCode	-4001: Failed to open the file, such as invalid data found when processing input, ffmpeg protocol not found, etc; -4002: Decoding failure, such as audio file corruption, inaccessible network audio file server, etc; -4003: The number of preloads exceeded the limit, Please call stopPlayMusic first to release the useless preload；-4005: Invalid path, Please check whether the path you passed points to a legal music file；-4006: Invalid URL, Please use a browser to check whether the URL address you passed in can download the desired music file；-4007: No audio stream, Please confirm whether the file you passed is a legal audio file and whether the file is damaged；-4008: Unsupported format, Please confirm whether the

	file format you passed is a supported file format. The mobile version supports [mp3, aac, m4a, wav, ogg, mp4, mkv], and the desktop version supports [mp3, aac, m4a, wav, mp4, mkv].

onStart

onStart

void onStart	(int id
	int errCode)

Background music started.

Called after the background music starts.

Param	DESC
errCode	0: Start playing successfully; -4001: Failed to open the file, such as invalid data found when processing input, ffmpeg protocol not found, etc; -4005: Invalid path, Please check whether the path you passed points to a legal music file；-4006: Invalid URL, Please use a browser to check whether the URL address you passed in can download the desired music file；-4007: No audio stream, Please confirm whether the file you passed is a legal audio file and whether the file is damaged；-4008: Unsupported format, Please confirm whether the file format you passed is a supported file format. The mobile version supports [mp3, aac, m4a, wav, ogg, mp4, mkv], and the desktop version supports [mp3, aac, m4a, wav, mp4, mkv].
id	music ID.

onPlayProgress

onPlayProgress

void onPlayProgress	(int id
	long curPtsMS
	long durationMS)

Playback progress of background music

onComplete

onComplete

void onComplete	(int id
	int errCode)

Background music ended

Called when the background music playback ends or an error occurs.

Param	DESC
errCode	0: End of play; -4002: Decoding failure, such as audio file corruption, inaccessible network audio file server, etc.
id	music ID.

enableVoiceEarMonitor

enableVoiceEarMonitor

void enableVoiceEarMonitor	(boolean enable)

Enabling in-ear monitoring

After enabling in-ear monitoring, anchors can hear in earphones their own voice captured by the mic. This is designed for singing scenarios.

In-ear monitoring cannot be enabled for Bluetooth earphones. This is because Bluetooth earphones have high latency. Please ask anchors to use wired earphones via a UI reminder.
Given that not all phones deliver excellent in-ear monitoring effects, we have blocked this feature on some phones.

Param	DESC
enable	`true:` enable; `false` : disable

Note

In-ear monitoring can be enabled only when earphones are used. Please remind anchors to use wired earphones.

setVoiceEarMonitorVolume

setVoiceEarMonitorVolume

void setVoiceEarMonitorVolume	(int volume)

Setting in-ear monitoring volume

This API is used to set the volume of in-ear monitoring.

Param	DESC
volume	Volume. Value range: 0-100; default: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setVoiceReverbType

setVoiceReverbType

void setVoiceReverbType	(TXVoiceReverbType type)

Setting voice reverb effects

This API is used to set reverb effects for human voice. For the effects supported, please see TXVoiceReverbType.

Note

Effects become invalid after room exit. If you want to use the same effect after you enter the room again, you need to set the effect again using this API.

setVoiceChangerType

setVoiceChangerType

void setVoiceChangerType	(TXVoiceChangerType type)

Setting voice changing effects

This API is used to set voice changing effects. For the effects supported, please see TXVoiceChangeType.

Note

Effects become invalid after room exit. If you want to use the same effect after you enter the room again, you need to set the effect again using this API.

setVoiceCaptureVolume

setVoiceCaptureVolume

void setVoiceCaptureVolume	(int volume)

Setting speech volume

This API is used to set the volume of speech. It is often used together with the music volume setting API setAllMusicVolume to balance between the volume of music and speech.

Param	DESC
volume	Volume. Value range: 0-100; default: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setVoicePitch

setVoicePitch

void setVoicePitch	(double pitch)

Setting speech pitch

This API is used to set the pitch of speech.

Param	DESC
pitch	Ptich， Value range: -1.0f~1.0f; default: 0.0f。

setMusicObserver

setMusicObserver

void setMusicObserver	(int id
	TXMusicPlayObserver observer)

Setting the background music callback

Before playing background music, please use this API to set the music callback, which can inform you of the playback progress.

Param	DESC

musicId	Music ID
observer	For more information, please see the APIs defined in `ITXMusicPlayObserver` .

Note

1. If the ID does not need to be used, the observer can be set to NULL to release it completely.

startPlayMusic

startPlayMusic

boolean startPlayMusic	(final AudioMusicParam musicParam)

Starting background music

You must assign an ID to each music track so that you can start, stop, or set the volume of music tracks by ID.

Param	DESC
musicParam	Music parameter

Note

1. If you play the same music track multiple times, please use the same ID instead of a separate ID for each playback.
2. If you want to play different music tracks at the same time, use different IDs for them.
3. If you use the same ID to play a music track different from the current one, the SDK will stop the current one before playing the new one.

stopPlayMusic

stopPlayMusic

void stopPlayMusic	(int id)

Stopping background music

Param	DESC
id	Music ID

pausePlayMusic

pausePlayMusic

void pausePlayMusic	(int id)

Pausing background music

Param	DESC
id	Music ID

resumePlayMusic

resumePlayMusic

void resumePlayMusic	(int id)

Resuming background music

Param	DESC
id	Music ID

setAllMusicVolume

setAllMusicVolume

void setAllMusicVolume	(int volume)

Setting the local and remote playback volume of background music

This API is used to set the local and remote playback volume of background music.
Local volume: the volume of music heard by anchors
Remote volume: the volume of music heard by audience

Param	DESC
volume	Volume. Value range: 0-100; default: 60

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setMusicPublishVolume

setMusicPublishVolume

void setMusicPublishVolume	(int id
	int volume)

Setting the remote playback volume of a specific music track

This API is used to control the remote playback volume (the volume heard by audience) of a specific music track.

Param	DESC
id	Music ID
volume	Volume. Value range: 0-100; default: 60

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setMusicPlayoutVolume

setMusicPlayoutVolume

void setMusicPlayoutVolume	(int id
	int volume)

Setting the local playback volume of a specific music track

This API is used to control the local playback volume (the volume heard by anchors) of a specific music track.

Param	DESC
id	Music ID
volume	Volume. Value range: 0-100. default: 60

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setMusicPitch

setMusicPitch

void setMusicPitch	(int id
	float pitch)

Adjusting the pitch of background music

Param	DESC
id	Music ID
pitch	Pitch. Value range: floating point numbers in the range of [-1, 1]; default: 0.0f

setMusicSpeedRate

setMusicSpeedRate

void setMusicSpeedRate	(int id
	float speedRate)

Changing the speed of background music

Param	DESC
id	Music ID
speedRate	Music speed. Value range: floating point numbers in the range of [0.5, 2]; default: 1.0f

getMusicCurrentPosInMS

getMusicCurrentPosInMS

long getMusicCurrentPosInMS	(int id)

Getting the playback progress (ms) of background music

Param	DESC

id	Music ID

Return Desc:

The milliseconds that have passed since playback started. -1 indicates failure to get the the playback progress.

getMusicDurationInMS

getMusicDurationInMS

long getMusicDurationInMS	(String path)

Getting the total length (ms) of background music

Param	DESC
path	Path of the music file.

Return Desc:

The length of the specified music file is returned. -1 indicates failure to get the length.

seekMusicToPosInMS

seekMusicToPosInMS

void seekMusicToPosInMS	(int id
	int pts)

Setting the playback progress (ms) of background music

Param	DESC
id	Music ID
pts	Unit: millisecond

Note

Do not call this API frequently as the music file may be read and written to each time the API is called, which can be time-consuming.

Wait till users finish dragging the progress bar before you call this API.
The progress bar controller on the UI tends to update the progress at a high frequency as users drag the progress bar. This will result in poor user experience unless you limit the frequency.

setMusicScratchSpeedRate

setMusicScratchSpeedRate

void setMusicScratchSpeedRate	(int id
	float scratchSpeedRate)

Adjust the speed change effect of the scratch disc

Param	DESC
id	Music ID
scratchSpeedRate	Scratch disc speed, the default value is 1.0f, the range is: a floating point number between [-12.0 ~ 12.0], the positive/negative speed value indicates the direction is positive/negative, and the absolute value indicates the speed.

Note

Precondition preloadMusic succeeds.

setPreloadObserver

setPreloadObserver

void setPreloadObserver	(TXMusicPreloadObserver observer)

Setting music preload callback

Before preload music, please use this API to set the preload callback, which can inform you of the preload status.

Param	DESC
observer	For more information, please see the APIs defined in `ITXMusicPreloadObserver` .

preloadMusic

preloadMusic

boolean preloadMusic	(final AudioMusicParam preloadParam)

Preload background music

You must assign an ID to each music track so that you can start, stop, or set the volume of music tracks by ID.

Param	DESC
musicParam	Music parameter

Note

1. Preload supports up to 2 preloads with different IDs at the same time, and the preload time does not exceed 10 minutes,you need to stopPlayMusic after use, otherwise the memory will not be released.
2. If the music corresponding to the ID is being played, the preloading fails, and stopPlayMusic must be called first.
3. When the musicParam passed to startPlayMusic is exactly the same, preloading works.

getMusicTrackCount

getMusicTrackCount

int getMusicTrackCount	(int id)

Get the number of tracks of background music

Param	DESC
id	Music ID

setMusicTrack

setMusicTrack

void setMusicTrack	(int id
	int trackIndex)

Specify the playback track of background music

Param	DESC

id	Music ID
index	Specify which track to play (the first track is played by default). Value range [0, total number of tracks).

Note

The total number of tracks can be obtained through the getMusicTrackCount interface.

TXVoiceReverbType

TXVoiceReverbType

Reverb effects

Reverb effects can be applied to human voice. Based on acoustic algorithms, they can mimic voice in different environments. The following effects are supported currently:

0: original; 1: karaoke; 2: room; 3: hall; 4: low and deep; 5: resonant; 6: metal; 7: husky; 8: ethereal; 9: studio; 10: melodious; 11: studio2;

Enum	Value	DESC
TXLiveVoiceReverbType_0	0	disable
TXLiveVoiceReverbType_1	1	KTV
TXLiveVoiceReverbType_2	2	small room
TXLiveVoiceReverbType_3	3	great hall
TXLiveVoiceReverbType_4	4	deep voice
TXLiveVoiceReverbType_5	5	loud voice
TXLiveVoiceReverbType_6	6	metallic sound
TXLiveVoiceReverbType_7	7	magnetic sound
TXLiveVoiceReverbType_8	8	ethereal
TXLiveVoiceReverbType_9	9	studio
TXLiveVoiceReverbType_10	10	melodious
TXLiveVoiceReverbType_11	11	studio2

TXVoiceChangeType

TXVoiceChangeType

Voice changing effects

Voice changing effects can be applied to human voice. Based on acoustic algorithms, they change the tone of voice.
The following effects are supported currently:
0: original; 1: child; 2: little girl; 3: middle-aged man; 4: metal; 5: nasal; 6: foreign accent; 7: trapped beast; 8: otaku; 9: electric; 10: robot; 11: ethereal

Enum	Value	DESC
TXLiveVoiceChangerType_0	0	disable
TXLiveVoiceChangerType_1	1	naughty kid
TXLiveVoiceChangerType_2	2	Lolita
TXLiveVoiceChangerType_3	3	uncle
TXLiveVoiceChangerType_4	4	heavy metal
TXLiveVoiceChangerType_5	5	catch cold
TXLiveVoiceChangerType_6	6	foreign accent
TXLiveVoiceChangerType_7	7	caged animal trapped beast
TXLiveVoiceChangerType_8	8	indoorsman
TXLiveVoiceChangerType_9	9	strong current
TXLiveVoiceChangerType_10	10	heavy machinery
TXLiveVoiceChangerType_11	11	intangible

TXAudioMusicParam

TXAudioMusicParam

Background music playback information

The information, including playback ID, file path, and loop times, is passed in the startPlayMusic API.
1. If you play the same music track multiple times, please use the same ID instead of a separate ID for each playback.

2. If you want to play different music tracks at the same time, use different IDs for them.
3. If you use the same ID to play a music track different from the current one, the SDK will stop the current one before playing the new one.

EnumType	DESC
endTimeMS	`Field description:` the point in time in milliseconds for ending music playback. 0 indicates that playback continues till the end of the music track.
id	`Field description:` music ID **Note** the SDK supports playing multiple music tracks. IDs are used to distinguish different music tracks and control their start, end, volume, etc.
isShortFile	`Field description:` whether the music played is a short music track `Valid values:` `true` : short music track that needs to be looped; `false` (default): normal-length music track
loopCount	`Field description:` number of times the music track is looped `Valid values:` 0 or any positive integer. 0 (default) indicates that the music is played once, 1 twice, and so on.
path	`Field description:` absolute path of the music file or url.the mp3,aac,m4a,wav supported.
publish	`Field description:` whether to send the music to remote users `Valid values:` `true` : remote users can hear the music played locally; `false` (default): only the local user can hear the music.
startTimeMS	`Field description:` the point in time in milliseconds for starting music playback

TXBeautyManager

Last updated：2024-06-06 15:26:14

Module: beauty filter and image processing parameter configurations

Function: you can modify parameters such as beautification, filter, and green screen

TXBeautyManager

TXBeautyManager

FuncList	DESC
setBeautyStyle	Sets the beauty (skin smoothing) filter algorithm.
setBeautyLevel	Sets the strength of the beauty filter.
setWhitenessLevel	Sets the strength of the brightening filter.
enableSharpnessEnhancement	Enables clarity enhancement.
setRuddyLevel	Sets the strength of the rosy skin filter.
setFilter	Sets color filter.
setFilterStrength	Sets the strength of color filter.
setGreenScreenFile	Sets green screen video
setEyeScaleLevel	Sets the strength of the eye enlarging filter.
setFaceSlimLevel	Sets the strength of the face slimming filter.
setFaceVLevel	Sets the strength of the chin slimming filter.
setChinLevel	Sets the strength of the chin lengthening/shortening filter.
setFaceShortLevel	Sets the strength of the face shortening filter.
setFaceNarrowLevel	Sets the strength of the face narrowing filter.

setNoseSlimLevel	Sets the strength of the nose slimming filter.
setEyeLightenLevel	Sets the strength of the eye brightening filter.
setToothWhitenLevel	Sets the strength of the teeth whitening filter.
setWrinkleRemoveLevel	Sets the strength of the wrinkle removal filter.
setPounchRemoveLevel	Sets the strength of the eye bag removal filter.
setSmileLinesRemoveLevel	Sets the strength of the smile line removal filter.
setForeheadLevel	Sets the strength of the hairline adjustment filter.
setEyeDistanceLevel	Sets the strength of the eye distance adjustment filter.
setEyeAngleLevel	Sets the strength of the eye corner adjustment filter.
setMouthShapeLevel	Sets the strength of the mouth shape adjustment filter.
setNoseWingLevel	Sets the strength of the nose wing narrowing filter.
setNosePositionLevel	Sets the strength of the nose position adjustment filter.
setLipsThicknessLevel	Sets the strength of the lip thickness adjustment filter.
setFaceBeautyLevel	Sets the strength of the face shape adjustment filter.
setMotionTmpl	Selects the AI animated effect pendant.
setMotionMute	Sets whether to mute during animated effect playback.

EnumType

EnumType	DESC
TXBeautyStyle	Beauty (skin smoothing) filter algorithm

setBeautyStyle

setBeautyStyle

void setBeautyStyle	(int beautyStyle)

Sets the beauty (skin smoothing) filter algorithm.

TRTC has multiple built-in skin smoothing algorithms. You can select the one most suitable for your product needs:

Param	DESC
beautyStyle	Beauty filter style. `TXBeautyStyleSmooth` : smooth; `TXBeautyStyleNature` : natural; `TXBeautyStylePitu` : Pitu

setBeautyLevel

setBeautyLevel

void setBeautyLevel	(float beautyLevel)

Sets the strength of the beauty filter.

Param	DESC
beautyLevel	Strength of the beauty filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

setWhitenessLevel

setWhitenessLevel

void setWhitenessLevel	(float whitenessLevel)

Sets the strength of the brightening filter.

Param	DESC
whitenessLevel	Strength of the brightening filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

enableSharpnessEnhancement

enableSharpnessEnhancement

void enableSharpnessEnhancement	(boolean enable)

Enables clarity enhancement.

setRuddyLevel

setRuddyLevel

void setRuddyLevel	(float ruddyLevel)

Sets the strength of the rosy skin filter.

Param	DESC
ruddyLevel	Strength of the rosy skin filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

setFilter

setFilter

void setFilter	(Bitmap image)

Sets color filter.

The color filter is a color lookup table image containing color mapping relationships. You can find several predefined filter images in the official demo we provide.
The SDK performs secondary processing on the original video image captured by the camera according to the mapping relationships in the lookup table to achieve the expected filter effect.

Param	DESC
image	Color lookup table containing color mapping relationships. The image must be in PNG format.

setFilterStrength

setFilterStrength

void setFilterStrength	(float strength)

Sets the strength of color filter.

The larger this value, the more obvious the effect of the color filter, and the greater the color difference between the video image processed by the filter and the original video image.
The default strength is 0.5, and if it is not sufficient, it can be adjusted to a value above 0.5. The maximum value is 1.

Param	DESC
strength	Value range: 0–1. The greater the value, the more obvious the effect. Default value: 0.5

setGreenScreenFile

setGreenScreenFile

int setGreenScreenFile	(String path)

Sets green screen video

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.
The green screen feature enabled by this API is not capable of intelligent keying. It requires that there be a green screen behind the videoed person or object for further chroma keying.

Param	DESC
path	Path of the video file in MP4 format. An empty value indicates to disable the effect.

Return Desc:

0: Success; -5: feature of license not supported.

setEyeScaleLevel

setEyeScaleLevel

int setEyeScaleLevel	(float eyeScaleLevel)

Sets the strength of the eye enlarging filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
eyeScaleLevel	Strength of the eye enlarging filter. Value range: 0–9. `0` indicates to disable the

	filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceSlimLevel

setFaceSlimLevel

int setFaceSlimLevel	(float faceSlimLevel)

Sets the strength of the face slimming filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
faceSlimLevel	Strength of the face slimming filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceVLevel

setFaceVLevel

int setFaceVLevel	(float faceVLevel)

Sets the strength of the chin slimming filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
faceVLevel	Strength of the chin slimming filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setChinLevel

setChinLevel

int setChinLevel	(float chinLevel)

Sets the strength of the chin lengthening/shortening filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
chinLevel	Strength of the chin lengthening/shortening filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates that the chin is shortened, and a value greater than 0 indicates that the chin is lengthened.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceShortLevel

setFaceShortLevel

int setFaceShortLevel	(float faceShortLevel)

Sets the strength of the face shortening filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
faceShortLevel	Strength of the face shortening filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceNarrowLevel

setFaceNarrowLevel

int setFaceNarrowLevel	(float faceNarrowLevel)

Sets the strength of the face narrowing filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
level	Strength of the face narrowing filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setNoseSlimLevel

setNoseSlimLevel

int setNoseSlimLevel	(float noseSlimLevel)

Sets the strength of the nose slimming filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
noseSlimLevel	Strength of the nose slimming filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setEyeLightenLevel

setEyeLightenLevel

int setEyeLightenLevel	(float eyeLightenLevel)

Sets the strength of the eye brightening filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
eyeLightenLevel	Strength of the eye brightening filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setToothWhitenLevel

setToothWhitenLevel

int setToothWhitenLevel	(float toothWhitenLevel)

Sets the strength of the teeth whitening filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
toothWhitenLevel	Strength of the teeth whitening filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setWrinkleRemoveLevel

setWrinkleRemoveLevel

int setWrinkleRemoveLevel	(float wrinkleRemoveLevel)

Sets the strength of the wrinkle removal filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
wrinkleRemoveLevel	Strength of the wrinkle removal filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setPounchRemoveLevel

setPounchRemoveLevel

int setPounchRemoveLevel	(float pounchRemoveLevel)

Sets the strength of the eye bag removal filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
pounchRemoveLevel	Strength of the eye bag removal filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setSmileLinesRemoveLevel

setSmileLinesRemoveLevel

int setSmileLinesRemoveLevel	(float smileLinesRemoveLevel)

Sets the strength of the smile line removal filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
smileLinesRemoveLevel	Strength of the smile line removal filter. Value range: 0–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setForeheadLevel

setForeheadLevel

int setForeheadLevel	(float foreheadLevel)

Sets the strength of the hairline adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
foreheadLevel	Strength of the hairline adjustment filter. Value range: -9–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setEyeDistanceLevel

setEyeDistanceLevel

int setEyeDistanceLevel	(float eyeDistanceLevel)

Sets the strength of the eye distance adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC

eyeDistanceLevel	Strength of the eye distance adjustment filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates to widen, and a value greater than 0 indicates to narrow.

Return Desc:

0: Success; -5: feature of license not supported.

setEyeAngleLevel

setEyeAngleLevel

int setEyeAngleLevel	(float eyeAngleLevel)

Sets the strength of the eye corner adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
eyeAngleLevel	Strength of the eye corner adjustment filter. Value range: -9–9. `0` indicates to disable the filter, and `9` indicates the most obvious effect.

Return Desc:

0: Success; -5: feature of license not supported.

setMouthShapeLevel

setMouthShapeLevel

int setMouthShapeLevel	(float mouthShapeLevel)

Sets the strength of the mouth shape adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
mouthShapeLevel	Strength of the mouth shape adjustment filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates to widen, and a value greater

	than 0 indicates to narrow.

Return Desc:

0: Success; -5: feature of license not supported.

setNoseWingLevel

setNoseWingLevel

int setNoseWingLevel	(float noseWingLevel)

Sets the strength of the nose wing narrowing filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
noseWingLevel	Strength of the nose wing adjustment filter. Value range: -9~9. `0` indicates to disable the filter, a value smaller than 0 indicates to widen, and a value greater than 0 indicates to narrow.

Return Desc:

0: Success; -5: feature of license not supported.

setNosePositionLevel

setNosePositionLevel

int setNosePositionLevel	(float nosePositionLevel)

Sets the strength of the nose position adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
nosePositionLevel	Strength of the nose position adjustment filter. Value range: -9~9. `0` indicates to disable the filter, a value smaller than 0 indicates to lift, and a value greater than 0 indicates to lower.

Return Desc:

0: Success; -5: feature of license not supported.

setLipsThicknessLevel

setLipsThicknessLevel

int setLipsThicknessLevel	(float lipsThicknessLevel)

Sets the strength of the lip thickness adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
lipsThicknessLevel	Strength of the lip thickness adjustment filter. Value range: -9–9. `0` indicates to disable the filter, a value smaller than 0 indicates to thicken, and a value greater than 0 indicates to thin.

Return Desc:

0: Success; -5: feature of license not supported.

setFaceBeautyLevel

setFaceBeautyLevel

int setFaceBeautyLevel	(float faceBeautyLevel)

Sets the strength of the face shape adjustment filter.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
faceBeautyLevel	Strength of the face shape adjustment filter. Value range: 0–9. `0` indicates to disable the filter, and the greater the value, the more obvious the effect.

Return Desc:

0: Success; -5: feature of license not supported.

setMotionTmpl

setMotionTmpl

void setMotionTmpl	(String tmplPath)

Selects the AI animated effect pendant.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect.

Param	DESC
tmplPath	Directory of the animated effect material file

setMotionMute

setMotionMute

void setMotionMute	(boolean motionMute)

Sets whether to mute during animated effect playback.

This interface is only available in the enterprise version SDK (the old version has been offline, if you need to use the advanced beauty function in the new version SDK, please refer to Tencent Beauty Effect SDK) in effect. Some animated effects have audio effects, which can be disabled through this API when they are played back.

Param	DESC
motionMute	`true` : mute; `false` : unmute

TXBeautyStyle

TXBeautyStyle

Beauty (skin smoothing) filter algorithm

TRTC has multiple built-in skin smoothing algorithms. You can select the one most suitable for your product needs.

Enum	Value	DESC
TXBeautyStyleSmooth	0	Smooth style, which uses a more radical algorithm for more obvious effect and is suitable for show live streaming.

TXBeautyStyleNature	1	Natural style, which retains more facial details for more natural effect and is suitable for most live streaming use cases.
TXBeautyStylePitu	2	Pitu style, which is provided by YouTu Lab. Its skin smoothing effect is between the smooth style and the natural style, that is, it retains more skin details than the smooth style and has a higher skin smoothing degree than the natural style.

TXDeviceManager

Last updated：2024-06-06 15:26:14

Module: audio/video device management module

Description: manages audio/video devices such as camera, mic, and speaker.

TXDeviceManager

TXDeviceManager

FuncList	DESC
isFrontCamera	Querying whether the front camera is being used
switchCamera	Switching to the front/rear camera (for mobile OS)
getCameraZoomMaxRatio	Getting the maximum zoom ratio of the camera (for mobile OS)
setCameraZoomRatio	Setting the camera zoom ratio (for mobile OS)
isAutoFocusEnabled	Querying whether automatic face detection is supported (for mobile OS)
enableCameraAutoFocus	Enabling auto focus (for mobile OS)
setCameraFocusPosition	Adjusting the focus (for mobile OS)
enableCameraTorch	Enabling/Disabling flash, i.e., the torch mode (for mobile OS)
setAudioRoute	Setting the audio route (for mobile OS)
setExposureCompensation	Set the exposure parameters of the camera, ranging from - 1 to 1
setCameraCapturerParam	Set camera acquisition preferences
setSystemVolumeType	Setting the system volume type (for mobile OS)

StructType

FuncList	DESC
TXCameraCaptureParam	Camera acquisition parameters

EnumType

EnumType	DESC
TXSystemVolumeType	System volume type
TXAudioRoute	Audio route (the route via which audio is played)
TXCameraCaptureMode	Camera acquisition preferences

isFrontCamera

isFrontCamera

Querying whether the front camera is being used

switchCamera

switchCamera

int switchCamera	(boolean frontCamera)

Switching to the front/rear camera (for mobile OS)

getCameraZoomMaxRatio

getCameraZoomMaxRatio

Getting the maximum zoom ratio of the camera (for mobile OS)

setCameraZoomRatio

setCameraZoomRatio

int setCameraZoomRatio	(float zoomRatio)

Setting the camera zoom ratio (for mobile OS)

Param	DESC
zoomRatio	Value range: 1-5. 1 indicates the widest angle of view (original), and 5 the narrowest angle of view (zoomed in).The maximum value is recommended to be 5. If the value exceeds 5, the video will become blurred.

isAutoFocusEnabled

isAutoFocusEnabled

Querying whether automatic face detection is supported (for mobile OS)

enableCameraAutoFocus

enableCameraAutoFocus

int enableCameraAutoFocus	(boolean enabled)

Enabling auto focus (for mobile OS)

After auto focus is enabled, the camera will automatically detect and always focus on faces.

setCameraFocusPosition

setCameraFocusPosition

int setCameraFocusPosition	(int x
	int y)

Adjusting the focus (for mobile OS)

This API can be used to achieve the following:

1. A user can tap on the camera preview.
2. A rectangle will appear where the user taps, indicating the spot the camera will focus on.

3. The user passes the coordinates of the spot to the SDK using this API, and the SDK will instruct the camera to focus as required.

Param	DESC
position	The spot to focus on. Pass in the coordinates of the spot you want to focus on.

Note

Before using this API, you must first disable auto focus using enableCameraAutoFocus.

Return Desc:

0: operation successful; negative number: operation failed.

enableCameraTorch

enableCameraTorch

boolean enableCameraTorch	(boolean enable)

Enabling/Disabling flash, i.e., the torch mode (for mobile OS)

setAudioRoute

setAudioRoute

int setAudioRoute	(TXAudioRoute route)

Setting the audio route (for mobile OS)

A mobile phone has two audio playback devices: the receiver at the top and the speaker at the bottom.
If the audio route is set to the receiver, the volume is relatively low, and audio can be heard only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.
If the audio route is set to the speaker, the volume is relatively high, and there is no need to put the phone near the ear. This mode enables the "hands-free" feature.

setExposureCompensation

setExposureCompensation

int setExposureCompensation	(float value)

Set the exposure parameters of the camera, ranging from - 1 to 1

setCameraCapturerParam

setCameraCapturerParam

void setCameraCapturerParam	(TXCameraCaptureParam params)

Set camera acquisition preferences

setSystemVolumeType

setSystemVolumeType

int setSystemVolumeType	(TXSystemVolumeType type)

Setting the system volume type (for mobile OS)

@deprecated This API is not recommended after v9.5. Please use the `startLocalAudio(quality)` API in `TRTCCloud` instead, which param `quality` is used to decide audio quality.

TXSystemVolumeType(Deprecated)

TXSystemVolumeType(Deprecated)

System volume type

Enum	Value	DESC
TXSystemVolumeTypeAuto	Not Defined	Auto
TXSystemVolumeTypeMedia	Not Defined	Media volume
TXSystemVolumeTypeVOIP	Not Defined	Call volume

TXAudioRoute

TXAudioRoute

Audio route (the route via which audio is played)

Audio route is the route (speaker or receiver) via which audio is played. It applies only to mobile devices such as mobile phones.
A mobile phone has two speakers: one at the top (receiver) and the other the bottom.
If the audio route is set to the receiver, the volume is relatively low, and audio can be heard only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.
If the audio route is set to the speaker, the volume is relatively high, and there is no need to put the phone near the ear. This mode enables the "hands-free" feature.

Enum	Value	DESC
TXAudioRouteSpeakerphone	Not Defined	Speakerphone: the speaker at the bottom is used for playback (hands-free). With relatively high volume, it is used to play music out loud.
TXAudioRouteEarpiece	Not Defined	Earpiece: the receiver at the top is used for playback. With relatively low volume, it is suitable for call scenarios that require privacy.

TXCameraCaptureMode

TXCameraCaptureMode

Camera acquisition preferences

This enum is used to set camera acquisition parameters.

Enum	Value	DESC
TXCameraResolutionStrategyAuto	Not Defined	Auto adjustment of camera capture parameters. SDK selects the appropriate camera output parameters according to the actual acquisition device performance and network situation, and maintains a balance between device performance and video preview quality.
TXCameraResolutionStrategyPerformance	Not	Give priority to equipment performance.

	Defined	SDK selects the closest camera output parameters according to the user's encoder resolution and frame rate, so as to ensure the performance of the device.
TXCameraResolutionStrategyHighQuality	Not Defined	Give priority to the quality of video preview. SDK selects higher camera output parameters to improve the quality of preview video. In this case, it will consume more CPU and memory to do video preprocessing.
TXCameraCaptureManual	Not Defined	Allows the user to set the width and height of the video captured by the local camera.

TXCameraCaptureParam

TXCameraCaptureParam

Camera acquisition parameters

This setting determines the quality of the local preview image.

EnumType	DESC
height	Field description: height of acquired image
mode	Field description: camera acquisition preferences,please see TXCameraCaptureMode
width	Field description: width of acquired image

Type Definition

Last updated：2024-06-06 15:50:05

Module: TRTC key class definition

Description: definitions of enumerated and constant values such as resolution and quality level

Type define

StructType

FuncList	DESC
TRTCParams	Room entry parameters
TRTCVideoEncParam	Video encoding parameters
TRTCNetworkQosParam	Network QoS control parameter set
TRTCRenderParams	Rendering parameters of video image
TRTCQuality	Network quality
TRTCVolumeInfo	Volume
TRTCSpeedTestParams	Network speed testing parameters
TRTCSpeedTestResult	Network speed test result
TRTCTexture	Video texture data
TRTCVideoFrame	Video frame information
TRTCAudioFrame	Audio frame data
TRTCMixUser	Description information of each video image in On-Cloud MixTranscoding
TRTCTranscodingConfig	Layout and transcoding parameters of On-Cloud MixTranscoding
TRTCPublishCDNParam	Push parameters required to be set when publishing

	audio/video streams to non-Tencent Cloud CDN
TRTCAudioRecordingParams	Local audio file recording parameters
TRTCLocalRecordingParams	Local media file recording parameters
TRTCAudioEffectParam	Sound effect parameter (disused)
TRTCSwitchRoomConfig	Room switch parameter
TRTCAudioFrameCallbackFormat	Format parameter of custom audio callback
TRTCScreenShareParams	Screen sharing parameter (for Android only)
TRTCUser	The users whose streams to publish
TRTCPublishCdnUrl	The destination URL when you publish to Tencent Cloud or a third-party CDN
TRTCPublishTarget	The publishing destination
TRTCVideoLayout	The video layout of the transcoded stream
TRTCWatermark	The watermark layout
TRTCStreamEncoderParam	The encoding parameters
TRTCStreamMixingConfig	The transcoding parameters
TRTCPayloadPrivateEncryptionConfig	Media Stream Private Encryption Configuration
TRTCAudioVolumeEvaluateParams	Volume evaluation and other related parameter settings.

EnumType

EnumType	DESC
TRTCVideoResolution	Video resolution
TRTCVideoResolutionMode	Video aspect ratio mode
TRTCVideoStreamType	Video stream type
TRTCVideoFillMode	Video image fill mode
TRTCVideoRotation	Video image rotation direction

TRTCBeautyStyle	Beauty (skin smoothing) filter algorithm
TRTCVideoPixelFormat	Video pixel format
TRTCVideoBufferType	Video data transfer method
TRTCVideoMirrorType	Video mirror type
TRTCSnapshotSourceType	Data source of local video screenshot
TRTCAppScene	Use cases
TRTCRoleType	Role
TRTCQosControlMode(Deprecated)	QoS control mode (disused)
TRTCVideoQosPreference	Image quality preference
TRTCQuality	Network quality
TRTCAVStatusType	Audio/Video playback status
TRTCAVStatusChangeReason	Reasons for playback status changes
TRTCAudioSampleRate	Audio sample rate
TRTCAudioQuality	Sound quality
TRTCAudioRoute	Audio route (i.e., audio playback mode)
TRTCReverbType	Audio reverb mode
TRTCVoiceChangerType	Voice changing type
TRTCSystemVolumeType	System volume type (only for mobile devices)
TRTCAudioFrameFormat	Audio frame content format
TRTCAudioCapabilityType	Audio capability type supported by the system (only for Android devices)
TRTCAudioFrameOperationMode	Audio callback data operation mode
TRTCLogLevel	Log level
TRTCGSensorMode	G-sensor switch (for mobile devices only)
TRTCTranscodingConfigMode	Layout mode of On-Cloud MixTranscoding
TRTCRecordType	Media recording type

TRTCMixInputType	Stream mix input type
TRTCDebugViewLevel	Debugging information displayed in the rendering control
TRTCAudioRecordingContent	Audio recording content type
TRTCPublishMode	The publishing mode
TRTCEncryptionAlgorithm	Encryption Algorithm
TRTCSpeedTestScene	Speed Test Scene
TRTCGravitySensorAdaptiveMode	Set the adaptation mode of gravity sensing (only applicable to mobile terminals)

TRTCVideoResolution

TRTCVideoResolution

Video resolution

Here, only the landscape resolution (e.g., 640x360) is defined. If the portrait resolution (e.g., 360x640) needs to be used, `Portrait` must be selected for `TRTCVideoResolutionMode` .

Enum	Value	DESC
TRTC_VIDEO_RESOLUTION_120_120	1	Aspect ratio: 1:1; resolution: 120x120; recommended bitrate (VideoCall): 80 Kbps; recommended bitrate (LIVE): 120 Kbps.
TRTC_VIDEO_RESOLUTION_160_160	3	Aspect ratio: 1:1; resolution: 160x160; recommended bitrate (VideoCall): 100 Kbps; recommended bitrate (LIVE): 150 Kbps.
TRTC_VIDEO_RESOLUTION_270_270	5	Aspect ratio: 1:1; resolution: 270x270; recommended bitrate (VideoCall): 200 Kbps; recommended bitrate (LIVE): 300 Kbps.
TRTC_VIDEO_RESOLUTION_480_480	7	Aspect ratio: 1:1; resolution: 480x480; recommended bitrate (VideoCall): 350 Kbps; recommended bitrate (LIVE): 500 Kbps.
TRTC_VIDEO_RESOLUTION_160_120	50	Aspect ratio: 4:3; resolution: 160x120; recommended bitrate (VideoCall): 100 Kbps; recommended bitrate (LIVE): 150 Kbps.

TRTC_VIDEO_RESOLUTION_240_180	52	Aspect ratio: 4:3; resolution: 240x180; recommended bitrate (VideoCall): 150 Kbps; recommended bitrate (LIVE): 250 Kbps.
TRTC_VIDEO_RESOLUTION_280_210	54	Aspect ratio: 4:3; resolution: 280x210; recommended bitrate (VideoCall): 200 Kbps; recommended bitrate (LIVE): 300 Kbps.
TRTC_VIDEO_RESOLUTION_320_240	56	Aspect ratio: 4:3; resolution: 320x240; recommended bitrate (VideoCall): 250 Kbps; recommended bitrate (LIVE): 375 Kbps.
TRTC_VIDEO_RESOLUTION_400_300	58	Aspect ratio: 4:3; resolution: 400x300; recommended bitrate (VideoCall): 300 Kbps; recommended bitrate (LIVE): 450 Kbps.
TRTC_VIDEO_RESOLUTION_480_360	60	Aspect ratio: 4:3; resolution: 480x360; recommended bitrate (VideoCall): 400 Kbps; recommended bitrate (LIVE): 600 Kbps.
TRTC_VIDEO_RESOLUTION_640_480	62	Aspect ratio: 4:3; resolution: 640x480; recommended bitrate (VideoCall): 600 Kbps; recommended bitrate (LIVE): 900 Kbps.
TRTC_VIDEO_RESOLUTION_960_720	64	Aspect ratio: 4:3; resolution: 960x720; recommended bitrate (VideoCall): 1000 Kbps; recommended bitrate (LIVE): 1500 Kbps.
TRTC_VIDEO_RESOLUTION_160_90	100	Aspect ratio: 16:9; resolution: 160x90; recommended bitrate (VideoCall): 150 Kbps; recommended bitrate (LIVE): 250 Kbps.
TRTC_VIDEO_RESOLUTION_256_144	102	Aspect ratio: 16:9; resolution: 256x144; recommended bitrate (VideoCall): 200 Kbps; recommended bitrate (LIVE): 300 Kbps.
TRTC_VIDEO_RESOLUTION_320_180	104	Aspect ratio: 16:9; resolution: 320x180; recommended bitrate (VideoCall): 250 Kbps; recommended bitrate (LIVE): 400 Kbps.
TRTC_VIDEO_RESOLUTION_480_270	106	Aspect ratio: 16:9; resolution: 480x270; recommended bitrate (VideoCall): 350 Kbps; recommended bitrate (LIVE): 550 Kbps.
TRTC_VIDEO_RESOLUTION_640_360	108	Aspect ratio: 16:9; resolution: 640x360; recommended bitrate (VideoCall): 500 Kbps; recommended bitrate (LIVE): 900 Kbps.

TRTC_VIDEO_RESOLUTION_960_540	110	Aspect ratio: 16:9; resolution: 960x540; recommended bitrate (VideoCall): 850 Kbps; recommended bitrate (LIVE): 1300 Kbps.
TRTC_VIDEO_RESOLUTION_1280_720	112	Aspect ratio: 16:9; resolution: 1280x720; recommended bitrate (VideoCall): 1200 Kbps; recommended bitrate (LIVE): 1800 Kbps.
TRTC_VIDEO_RESOLUTION_1920_1080	114	Aspect ratio: 16:9; resolution: 1920x1080; recommended bitrate (VideoCall): 2000 Kbps; recommended bitrate (LIVE): 3000 Kbps.

TRTCVideoResolutionMode

TRTCVideoResolutionMode

Video aspect ratio mode

Only the landscape resolution (e.g., 640x360) is defined in `TRTCVideoResolution` . If the portrait resolution (e.g., 360x640) needs to be used, `Portrait` must be selected for `TRTCVideoResolutionMode` .

Enum	Value	DESC
TRTC_VIDEO_RESOLUTION_MODE_LANDSCAPE	0	Landscape resolution, such as TRTCVideoResolution_640_360 + TRTCVideoResolutionModeLandscape = 640x360.
TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT	1	Portrait resolution, such as TRTCVideoResolution_640_360 + TRTCVideoResolutionModePortrait = 360x640.

TRTCVideoStreamType

TRTCVideoStreamType

Video stream type

TRTC provides three different video streams, including:

HD big image: it is generally used to transfer video data from the camera.

Smooth small image: it has the same content as the big image, but with lower resolution and bitrate and thus lower definition.

Substream image: it is generally used for screen sharing. Only one user in the room is allowed to publish the substream video image at any time, while other users must wait for this user to close the substream before they can publish their own substream.

Note

The SDK does not support enabling the smooth small image alone, which must be enabled together with the big image. It will automatically set the resolution and bitrate of the small image.

Enum	Value	DESC
TRTC_VIDEO_STREAM_TYPE_BIG	0	HD big image: it is generally used to transfer video data from the camera.
TRTC_VIDEO_STREAM_TYPE_SMALL	1	Smooth small image: it has the same content as the big image, but with lower resolution and bitrate and thus lower definition.
TRTC_VIDEO_STREAM_TYPE_SUB	2	Substream image: it is generally used for screen sharing. Only one user in the room is allowed to publish the substream video image at any time, while other users must wait for this user to close the substream before they can publish their own substream.

TRTCVideoFillMode

TRTCVideoFillMode

Video image fill mode

If the aspect ratio of the video display area is not equal to that of the video image, you need to specify the fill mode:

Enum	Value	DESC
TRTC_VIDEO_RENDER_MODE_FILL	0	Fill mode: the video image will be centered and scaled to fill the entire display area, where parts that exceed the area will be cropped. The displayed image may be incomplete in this mode.
TRTC_VIDEO_RENDER_MODE_FIT	1	Fit mode: the video image will be scaled based on its long side to fit the display area, where the short side will be filled with black bars. The displayed image is complete in this mode, but there may be black bars.

TRTCVideoRotation

TRTCVideoRotation

Video image rotation direction

TRTC provides rotation angle setting APIs for local and remote images. The following rotation angles are all clockwise.

Enum	Value	DESC
TRTC_VIDEO_ROTATION_0	0	No rotation
TRTC_VIDEO_ROTATION_90	1	Clockwise rotation by 90 degrees
TRTC_VIDEO_ROTATION_180	2	Clockwise rotation by 180 degrees
TRTC_VIDEO_ROTATION_270	3	Clockwise rotation by 270 degrees

TRTCBeautyStyle

TRTCBeautyStyle

Beauty (skin smoothing) filter algorithm

TRTC has multiple built-in skin smoothing algorithms. You can select the one most suitable for your product.

Enum	Value	DESC
TRTC_BEAUTY_STYLE_SMOOTH	0	Smooth style, which uses a more radical algorithm for more obvious effect and is suitable for show live streaming.
TRTC_BEAUTY_STYLE_NATURE	1	Natural style, which retains more facial details for more natural effect and is suitable for most live streaming use cases.
TRTC_BEAUTY_STYLE_PITU	2	Pitu style, which is provided by YouTu Lab. Its skin smoothing effect is between the smooth style and the natural style, that is, it retains more skin details than the smooth style and has a higher skin smoothing degree than the natural style.

TRTCVideoPixelFormat

TRTCVideoPixelFormat

Video pixel format

TRTC provides custom video capturing and rendering features.
For the custom capturing feature, you can use the following enumerated values to describe the pixel format of the video you capture.
For the custom rendering feature, you can specify the pixel format of the video you expect the SDK to call back.

Enum	Value	DESC
TRTC_VIDEO_PIXEL_FORMAT_UNKNOWN	0	Undefined format
TRTC_VIDEO_PIXEL_FORMAT_I420	1	YUV420P (I420) format
TRTC_VIDEO_PIXEL_FORMAT_Texture_2D	2	OpenGL 2D texture format
TRTC_VIDEO_PIXEL_FORMAT_TEXTURE_EXTERNAL_OES	3	OES external texture format (for Android)
TRTC_VIDEO_PIXEL_FORMAT_NV21	4	NV21 format
TRTC_VIDEO_PIXEL_FORMAT_RGBA	5	RGBA format

TRTCVideoBufferType

TRTCVideoBufferType

Video data transfer method

For custom capturing and rendering features, you need to use the following enumerated values to specify the method of transferring video data:
Method 1. This method uses memory buffer to transfer video data. It is efficient on iOS but inefficient on Android. It is the only method supported on Windows currently.
Method 2. This method uses texture to transfer video data. It is efficient on both iOS and Android but is not supported on Windows. To use this method, you should have a general familiarity with OpenGL programming.

Enum	Value	DESC
TRTC_VIDEO_BUFFER_TYPE_UNKNOWN	0	Undefined transfer method
TRTC_VIDEO_BUFFER_TYPE_BYTE_BUFFER	1	Use memory buffer to transfer video data. iOS: `PixelBuffer` ; Android: `Direct Buffer` for JNI layer; Windows: memory data block.

TRTC_VIDEO_BUFFER_TYPE_BYTE_ARRAY	2	Use memory buffer to transfer video data. iOS: more compact memory block in `NSData` type after additional processing; Android: `byte[]` for Java layer. This transfer method has a lower efficiency than other methods.
TRTC_VIDEO_BUFFER_TYPE_TEXTURE	3	Use OpenGL texture to transfer video data

TRTCVideoMirrorType

TRTCVideoMirrorType

Video mirror type

Video mirroring refers to the left-to-right flipping of the video image, especially for the local camera preview image. After mirroring is enabled, it can bring anchors a familiar "look into the mirror" experience.

Enum	Value	DESC
TRTC_VIDEO_MIRROR_TYPE_AUTO	0	Auto mode: mirror the front camera's image but not the rear camera's image (for mobile devices only).
TRTC_VIDEO_MIRROR_TYPE_ENABLE	1	Mirror the images of both the front and rear cameras.
TRTC_VIDEO_MIRROR_TYPE_DISABLE	2	Disable mirroring for both the front and rear cameras.

TRTCSnapshotSourceType

TRTCSnapshotSourceType

Data source of local video screenshot

The SDK can take screenshots from the following two data sources and save them as local files:
Video stream: the SDK screencaptures the native video content from the video stream. The screenshots are not controlled by the display of the rendering control.
Rendering layer: the SDK screencaptures the displayed video content from the rendering control, which can achieve the effect of WYSIWYG, but if the display area is too small, the screenshots will also be very small.

Enum	Value	DESC

TRTC_SNAPSHOT_SOURCE_TYPE_STREAM	0	The SDK screencaptures the native video content from the video stream. The screenshots are not controlled by the display of the rendering control.
TRTC_SNAPSHOT_SOURCE_TYPE_VIEW	1	The SDK screencaptures the displayed video content from the rendering control, which can achieve the effect of WYSIWYG, but if the display area is too small, the screenshots will also be very small.
TRTC_SNAPSHOT_SOURCE_TYPE_CAPTURE	2	The SDK screencaptures the capture video content from the capture control, which can capture the captured high-definition screenshots.

TRTCAppScene

TRTCAppScene

Use cases

TRTC features targeted optimizations for common audio/video application scenarios to meet the differentiated requirements in various verticals. The main scenarios can be divided into the following two categories:

Live streaming scenario (LIVE): including `LIVE` (audio + video) and `VoiceChatRoom` (pure audio).

In the live streaming scenario, users are divided into two roles: "anchor" and "audience". A single room can sustain up to 100,000 concurrent online users. This is suitable for live streaming to a large audience.

Real-Time scenario (RTC): including `VideoCall` (audio + video) and `AudioCall` (pure audio).

In the real-time scenario, there is no role difference between users, but a single room can sustain only up to 300 concurrent online users. This is suitable for small-scale real-time communication.

Enum	Value	DESC
TRTC_APP_SCENE_VIDEOCALL	0	In the video call scenario, 720p and 1080p HD image quality is supported. A single room can sustain up to 300 concurrent online users, and up to 50 of them can speak simultaneously. Use cases: [one-to-one video call], [video conferencing with up to 300 participants], [online medical diagnosis], [small class], [video interview], etc.
TRTC_APP_SCENE_LIVE	1	In the interactive video live streaming scenario, mic

		can be turned on/off smoothly without waiting for switchover, and the anchor latency is as low as less than 300 ms. Live streaming to hundreds of thousands of concurrent users in the audience role is supported with the playback latency down to 1,000 ms. Use cases: [low-latency interactive live streaming], [big class], [anchor competition], [video dating room], [online interactive classroom], [remote training], [large-scale conferencing], etc. **Note** In this scenario, you must use the `role` field in `TRTCParams` to specify the role of the current user.
TRTC_APP_SCENE_AUDIOCALL	2	Audio call scenario, where the `SPEECH` sound quality is used by default. A single room can sustain up to 300 concurrent online users, and up to 50 of them can speak simultaneously. Use cases: [one-to-one audio call], [audio conferencing with up to 300 participants], [audio chat], [online Werewolf], etc.
TRTC_APP_SCENE_VOICE_CHATROOM	3	In the interactive audio live streaming scenario, mic can be turned on/off smoothly without waiting for switchover, and the anchor latency is as low as less than 300 ms. Live streaming to hundreds of thousands of concurrent users in the audience role is supported with the playback latency down to 1,000 ms. Use cases: [audio club], [online karaoke room], [music live room], [FM radio], etc. **Note** In this scenario, you must use the `role` field in `TRTCParams` to specify the role of the current user.

TRTCRoleType

TRTCRoleType

Role

Role is applicable only to live streaming scenarios (`TRTCAppSceneLIVE` and `TRTCAppSceneVoiceChatRoom`). Users are divided into two roles:
Anchor, who can publish their audio/video streams. There is a limit on the number of anchors. Up to 50 anchors are allowed to publish streams at the same time in one room.
Audience, who can only listen to or watch audio/video streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room can sustain up to 100,000 concurrent online users in the audience role.

Enum	Value	DESC
TRTCRoleAnchor	20	An anchor can publish their audio/video streams. There is a limit on the number of anchors. Up to 50 anchors are allowed to publish streams at the same time in one room.
TRTCRoleAudience	21	Audience can only listen to or watch audio/video streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room can sustain up to 100,000 concurrent online users in the audience role.

TRTCQosControlMode(Deprecated)

TRTCQosControlMode(Deprecated)

QoS control mode (disused)

Enum	Value	DESC
VIDEO_QOS_CONTROL_CLIENT	0	Client-based control, which is for internal debugging of SDK and shall not be used by users.
VIDEO_QOS_CONTROL_SERVER	1	On-cloud control, which is the default and recommended mode.

TRTCVideoQosPreference

TRTCVideoQosPreference

Image quality preference

TRTC has two control modes in weak network environments: "ensuring clarity" and "ensuring smoothness". Both modes will give priority to the transfer of audio data.

Enum	Value	DESC
TRTC_VIDEO_QOS_PREFERENCE_SMOOTH	1	Ensuring smoothness: in this mode, when the current network is unable to transfer a clear and smooth video image, the smoothness of the image will be given priority, but there will be blurs.
TRTC_VIDEO_QOS_PREFERENCE_CLEAR	2	Ensuring clarity (default value): in this mode, when the current network is unable to transfer a clear and smooth video image, the clarity of the image will be given priority, but there will be lags.

TRTCQuality

TRTCQuality

Network quality

TRTC evaluates the current network quality once every two seconds. The evaluation results are divided into six levels: `Excellent` indicates the best, and `Down` indicates the worst.

Enum	Value	DESC
TRTC_QUALITY_UNKNOWN	0	Undefined
TRTC_QUALITY_Excellent	1	The current network is excellent
TRTC_QUALITY_Good	2	The current network is good
TRTC_QUALITY_Poor	3	The current network is fair
TRTC_QUALITY_Bad	4	The current network is bad
TRTC_QUALITY_Vbad	5	The current network is very bad
TRTC_QUALITY_Down	6	The current network cannot meet the minimum requirements of TRTC

TRTCAVStatusType

TRTCAVStatusType

Audio/Video playback status

This enumerated type is used in the audio status changed API onRemoteAudioStatusUpdated and the video status changed API onRemoteVideoStatusUpdated to specify the current audio/video status.

Enum	Value	DESC
TRTCAVStatusStopped	0	Stopped
TRTCAVStatusPlaying	1	Playing
TRTCAVStatusLoading	2	Loading

TRTCAVStatusChangeReason

TRTCAVStatusChangeReason

Reasons for playback status changes

This enumerated type is used in the audio status changed API onRemoteAudioStatusUpdated and the video status changed API onRemoteVideoStatusUpdated to specify the reason for the current audio/video status change.

Enum	Value	DESC
TRTCAVStatusChangeReasonInternal	0	Default value
TRTCAVStatusChangeReasonBufferingBegin	1	The stream enters the `Loading` state due to network congestion
TRTCAVStatusChangeReasonBufferingEnd	2	The stream enters the `Playing` state after network recovery
TRTCAVStatusChangeReasonLocalStarted	3	As a start-related API was directly called locally, the stream enters the `Playing` state
TRTCAVStatusChangeReasonLocalStopped	4	As a stop-related API was directly called locally, the stream enters the `Stopped` state
TRTCAVStatusChangeReasonRemoteStarted	5	As the remote user started (or resumed) publishing the audio or video stream, the stream enters the `Loading` or `Playing` state
TRTCAVStatusChangeReasonRemoteStopped	6	As the remote user stopped (or paused) publishing the audio or video stream, the

		stream enters the "Stopped" state

TRTCAudioSampleRate

TRTCAudioSampleRate

Audio sample rate

The audio sample rate is used to measure the audio fidelity. A higher sample rate indicates higher fidelity. If there is music in the use case, `TRTCAudioSampleRate48000` is recommended.

Enum	Value	DESC
TRTCAudioSampleRate16000	16000	16 kHz sample rate
TRTCAudioSampleRate32000	32000	32 kHz sample rate
TRTCAudioSampleRate44100	44100	44.1 kHz sample rate
TRTCAudioSampleRate48000	48000	48 kHz sample rate

TRTCAudioQuality

TRTCAudioQuality

Sound quality

TRTC provides three well-tuned modes to meet the differentiated requirements for sound quality in various verticals:
Speech mode (Speech): it is suitable for application scenarios that focus on human communication. In this mode, the audio transfer is more resistant, and TRTC uses various voice processing technologies to ensure the optimal smoothness even in weak network environments.
Music mode (Music): it is suitable for scenarios with demanding requirements for music. In this mode, the amount of transferred audio data is very large, and TRTC uses various technologies to ensure that the high-fidelity details of music signals can be restored in each frequency band.
Default mode (Default): it is between `Speech` and `Music`. In this mode, the reproduction of music is better than that in `Speech` mode, and the amount of transferred data is much lower than that in `Music` mode; therefore, this mode has good adaptability to various scenarios.

Enum	Value	DESC
TRTC_AUDIO_QUALITY_SPEECH	1	Speech mode: sample rate: 16 kHz; mono channel; bitrate: 16 Kbps. This mode has the best resistance

		among all modes and is suitable for audio call scenarios, such as online meeting and audio call.
TRTC_AUDIO_QUALITY_DEFAULT	2	Default mode: sample rate: 48 kHz; mono channel; bitrate: 50 Kbps. This mode is between the speech mode and the music mode as the default mode in the SDK and is recommended.
TRTC_AUDIO_QUALITY_MUSIC	3	Music mode: sample rate: 48 kHz; full-band stereo; bitrate: 128 Kbps. This mode is suitable for scenarios where Hi-Fi music transfer is required, such as online karaoke and music live streaming.

TRTCAudioRoute

TRTCAudioRoute

Audio route (i.e., audio playback mode)

"Audio route" determines whether the sound is played back from the speaker or receiver of a mobile device; therefore, this API is applicable only to mobile devices such as phones.

Generally, a phone has two speakers: one is the receiver at the top, and the other is the stereo speaker at the bottom.

If the audio route is set to the receiver, the volume is relatively low, and the sound can be heard clearly only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.

If the audio route is set to the speaker, the volume is relatively high, so there is no need to put the phone near the ear. Therefore, this mode can implement the "hands-free" feature.

Enum	Value	DESC
TRTC_AUDIO_ROUTE_SPEAKER	0	Speakerphone: the speaker at the bottom is used for playback (hands-free). With relatively high volume, it is used to play music out loud.
TRTC_AUDIO_ROUTE_EARPIECE	1	Earpiece: the receiver at the top is used for playback. With relatively low volume, it is suitable for call scenarios that require privacy.
TRTC_AUDIO_ROUTE_WIRED_HEADSET	2	WiredHeadset：play using wired headphones.
TRTC_AUDIO_ROUTE_BLUETOOTH_HEADSET	3	BluetoothHeadset：play with bluetooth headphones.

TRTC_AUDIO_ROUTE_SOUND_CARD	4	SoundCard：play using a USB sound card.

TRTCReverbType

TRTCReverbType

Audio reverb mode

This enumerated value is used to set the audio reverb mode in the live streaming scenario and is often used in show live streaming.

Enum	Value	DESC
TRTC_REVERB_TYPE_0	0	Disable reverb
TRTC_REVERB_TYPE_1	1	KTV
TRTC_REVERB_TYPE_2	2	Small room
TRTC_REVERB_TYPE_3	3	Hall
TRTC_REVERB_TYPE_4	4	Deep
TRTC_REVERB_TYPE_5	5	Resonant
TRTC_REVERB_TYPE_6	6	Metallic
TRTC_REVERB_TYPE_7	7	Husky

TRTCVoiceChangerType

TRTCVoiceChangerType

Voice changing type

This enumerated value is used to set the voice changing mode in the live streaming scenario and is often used in show live streaming.

Enum	Value	DESC
TRTC_VOICE_CHANGER_TYPE_0	0	Disable voice changing
TRTC_VOICE_CHANGER_TYPE_1	1	Child
TRTC_VOICE_CHANGER_TYPE_2	2	Girl

TRTC_VOICE_CHANGER_TYPE_3	3	Middle-Aged man
TRTC_VOICE_CHANGER_TYPE_4	4	Heavy metal
TRTC_VOICE_CHANGER_TYPE_5	5	Nasal
TRTC_VOICE_CHANGER_TYPE_6	6	Punk
TRTC_VOICE_CHANGER_TYPE_7	7	Trapped beast
TRTC_VOICE_CHANGER_TYPE_8	8	Otaku
TRTC_VOICE_CHANGER_TYPE_9	9	Electronic
TRTC_VOICE_CHANGER_TYPE_10	10	Robot
TRTC_VOICE_CHANGER_TYPE_11	11	Ethereal

TRTCSystemVolumeType

TRTCSystemVolumeType

System volume type (only for mobile devices)

Smartphones usually have two types of system volume: call volume and media volume.
Call volume is designed for call scenarios. It comes with acoustic echo cancellation (AEC) and supports audio capturing by Bluetooth earphones, but its sound quality is average.
If you cannot turn the volume down to 0 (i.e., mute the phone) using the volume buttons, then your phone is using call volume.
Media volume is designed for media scenarios such as music playback. AEC does not work when media volume is used, and Bluetooth earphones cannot be used for audio capturing. However, media volume delivers better music listening experience.
If you are able to mute your phone using the volume buttons, then your phone is using media volume.

The SDK offers three system volume control modes: auto, call volume, and media volume.

Enum	Value	DESC
TRTCSystemVolumeTypeAuto	0	Auto: In the auto mode, call volume is used for anchors, and media volume for audience. This mode is suitable for live streaming scenarios. If the scenario you select during `enterRoom` is `TRTCAppSceneLIVE` or

		`TRTCAppSceneVoiceChatRoom` , the SDK will automatically use this mode.
TRTCSystemVolumeTypeMedia	1	Media volume: In this mode, media volume is used in all scenarios. It is rarely used, mainly suitable for music scenarios with demanding requirements on audio quality. Use this mode if most of your users use peripheral devices such as audio cards. Otherwise, it is not recommended.
TRTCSystemVolumeTypeVOIP	2	Call volume: In this mode, the audio module does not change its work mode when users switch between anchors and audience, enabling seamless mic on/off. This mode is suitable for scenarios where users need to switch frequently between anchors and audience. If the scenario you select during `enterRoom` is `TRTCAppSceneVideoCall` or `TRTCAppSceneAudioCall` , the SDK will automatically use this mode.

TRTCAudioFrameFormat

TRTCAudioFrameFormat

Audio frame content format

Enum	Value	DESC
TRTC_AUDIO_FRAME_FORMAT_PCM	1	Audio data in PCM format

TRTCAudioCapabilityType

TRTCAudioCapabilityType

Audio capability type supported by the system (only for Android devices)

The SDK currently provides two types of system audio capabilities to query whether they are supported: low-latency chorus capability and low-latency earmonitor capability.

Enum	Value	DESC
TRTCAudioCapabilityLowLatencyChorus	1	low-latency chorus capability

TRTCAudioCapabilityLowLatencyEarMonitor	2	low-latency earmonitor capability

TRTCAudioFrameOperationMode

TRTCAudioFrameOperationMode

Audio callback data operation mode

TRTC provides two modes of operation for audio callback data.
Read-only mode (ReadOnly): Get audio data only from the callback.
ReadWrite mode (ReadWrite): You can get and modify the audio data of the callback.

Enum	Value	DESC
TRTC_AUDIO_FRAME_OPERATION_MODE_READWRITE	0	Read-write mode: You can get and modify the audio data of the callback, the default mode.
TRTC_AUDIO_FRAME_OPERATION_MODE_READONLY	1	Read-only mode: Get audio data from callback only.

TRTCLogLevel

TRTCLogLevel

Log level

Different log levels indicate different levels of details and number of logs. We recommend you set the log level to `TRTCLogLevelInfo` generally.

Enum	Value	DESC
TRTC_LOG_LEVEL_VERBOSE	0	Output logs at all levels
TRTC_LOG_LEVEL_DEBUG	1	Output logs at the DEBUG, INFO, WARNING, ERROR, and FATAL levels
TRTC_LOG_LEVEL_INFO	2	Output logs at the INFO, WARNING, ERROR, and FATAL levels
TRTC_LOG_LEVEL_WARN	3	Output logs at the WARNING, ERROR, and FATAL levels
TRTC_LOG_LEVEL_ERROR	4	Output logs at the ERROR and FATAL levels

TRTC_LOG_LEVEL_FATAL	5	Output logs at the FATAL level
TRTC_LOG_LEVEL_NULL	6	Do not output any SDK logs

TRTCGSensorMode

TRTCGSensorMode

G-sensor switch (for mobile devices only)

Enum	Value	DESC
TRTC_GSENSOR_MODE_DISABLE	0	Do not adapt to G-sensor orientation This mode is the default value for desktop platforms. In this mode, the video image published by the current user is not affected by the change of the G-sensor orientation.
TRTC_GSENSOR_MODE_UIAUTOLAYOUT	1	Adapt to G-sensor orientation This mode is the default value on mobile platforms. In this mode, the video image published by the current user is adjusted according to the G-sensor orientation, while the orientation of the local preview image remains unchanged. One of the adaptation modes currently supported by the SDK is as follows: when the phone or tablet is upside down, in order to ensure that the screen orientation seen by the remote user is normal, the SDK will automatically rotate the published video image by 180 degrees. If the UI layer of your application has enabled G-sensor adaption, we recommend you use the `UIFixLayout` mode.
TRTC_GSENSOR_MODE_UIFIXLAYOUT	2	Adapt to G-sensor orientation In this mode, the video image published by the current user is adjusted according to the G-sensor orientation, and the local preview image will also be rotated accordingly. One of the features currently supported is as follows: when the phone or tablet is upside down, in order to ensure that the screen orientation seen by the remote user is normal, the SDK will

		automatically rotate the published video image by 180 degrees. If the UI layer of your application doesn't support G-sensor adaption, but you want the video image in the SDK to adapt to the G-sensor orientation, we recommend you use the `UIFixLayout` mode. @deprecated Begin from v11.5 version, it no longer supports TRTCGSensorMode_UIFixLayout and only supports the above two modes.

TRTCTranscodingConfigMode

TRTCTranscodingConfigMode

Layout mode of On-Cloud MixTranscoding

TRTC's On-Cloud MixTranscoding service can mix multiple audio/video streams in the room into one stream. Therefore, you need to specify the layout scheme of the video images. The following layout modes are provided:

Enum	Value	DESC
TRTC_TranscodingConfigMode_Unknown	0	Undefined
TRTC_TranscodingConfigMode_Manual	1	Manual layout mode In this mode, you need to specify the precise position of each video image. This mode has the highest degree of freedom, but its ease of use is the worst: You need to enter all the parameters in `TRTCTranscodingConfig` , including the position coordinates of each video image (TRTCMixUser). You need to listen on the `onUserVideoAvailable()` and `onUserAudioAvailable()` event callbacks in `TRTCCloudDelegate` and constantly adjust the `mixUsers` parameter according to the audio/video status of each user with mic on in the current room.

TRTC_TranscodingConfigMode_Template_PureAudio	2	Pure audio mode This mode is suitable for pure audio scenarios such as audio call (AudioCall) and audio chat room (VoiceChatRoom). You only need to set it once through the `setMixTranscodingConfig()` API after room entry, and then the SDK will automatically mix the audio of all mic-on users in the room into the current user's live stream. You don't need to set the `mixUsers` parameter in `TRTCTranscodingConfig` ; instead, you only need to set the `audioSampleRate` , `audioBitrate` and `audioChannels` parameters.
TRTC_TranscodingConfigMode_Template_PresetLayout	3	Preset layout mode This is the most popular layout mode, because it allows you to set the position of each video image in advance through placeholders, and then the SDK automatically adjusts it dynamically according to the number of video images in the room. In this mode, you still need to set the `mixUsers` parameter, but you can set `userId` as a "placeholder". Placeholder values include: "$PLACE_HOLDER_REMOTE$": image of remote user. Multiple images can be set. "$PLACE_HOLDER_LOCAL_MAIN$": local camera image. Only one image can be set. "$PLACE_HOLDER_LOCAL_SUB$": local screen sharing image. Only one image can be set. In this mode, you don't need to listen on the `onUserVideoAvailable()` and

		`onUserAudioAvailable()` callbacks in `TRTCCloudDelegate` to make real-time adjustments. Instead, you only need to call `setMixTranscodingConfig()` once after successful room entry. Then, the SDK will automatically populate the placeholders you set with real `userId` values.
TRTC_TranscodingConfigMode_Template_ScreenSharing	4	Screen sharing mode This mode is suitable for screen sharing-based use cases such as online education and supported only by the SDKs for Windows and macOS. In this mode, the SDK will first build a canvas according to the target resolution you set (through the `videoWidth` and `videoHeight` parameters). Before the teacher enables screen sharing, the SDK will scale up the teacher's camera image and draw it onto the canvas. After the teacher enables screen sharing, the SDK will draw the video image shared on the screen onto the same canvas. The purpose of this layout mode is to ensure consistency in the output resolution of the mixtranscoding module and avoid problems with blurred screen during course replay and webpage playback (web players don't support adjustable resolution). Meanwhile, the audio of mic-on students will be mixed into the teacher's audio/video stream by default. Video content is primarily the shared screen in teaching mode, and it is a waste of bandwidth to transfer camera image and screen image at the same time.

		Therefore, the recommended practice is to directly draw the camera image onto the current screen through the `setLocalVideoRenderCallback` API. In this mode, you don't need to set the `mixUsers` parameter in `TRTCTranscodingConfig`, and the SDK will not mix students' images so as not to interfere with the screen sharing effect. You can set width x height in `TRTCTranscodingConfig` to 0 px x 0 px, and the SDK will automatically calculate a suitable resolution based on the aspect ratio of the user's current screen. If the teacher's current screen width is less than or equal to 1920 px, the SDK will use the actual resolution of the teacher's current screen. If the teacher's current screen width is greater than 1920 px, the SDK will select one of the three resolutions of 1920x1080 (16:9), 1920x1200 (16:10), and 1920x1440 (4:3) according to the current screen aspect ratio.

TRTCRecordType

TRTCRecordType

Media recording type

This enumerated type is used in the local media recording API startLocalRecording to specify whether to record audio/video files or pure audio files.

Enum	Value	DESC
TRTC_RECORD_TYPE_AUDIO	0	Record audio only
TRTC_RECORD_TYPE_VIDEO	1	Record video only

TRTC_RECORD_TYPE_BOTH	2	Record both audio and video

TRTCMixInputType

TRTCMixInputType

Stream mix input type

Enum	Value	DESC
TRTC_MixInputType_Undefined	0	Default. Considering the compatibility with older versions, if you specify the inputType as Undefined, the SDK will determine the stream mix input type according to the value of the `pureAudio` parameter
TRTC_MixInputType_AudioVideo	1	Mix both audio and video
TRTC_MixInputType_PureVideo	2	Mix video only
TRTC_MixInputType_PureAudio	3	Mix audio only
TRTC_MixInputType_Watermark	4	Mix watermark In this case, you don't need to specify the `userId` parameter, but you need to specify the `image` parameter. It is recommended to use png format.

TRTCDebugViewLevel

TRTCDebugViewLevel

Debugging information displayed in the rendering control

Enum	Value	DESC
TRTC_DEBUG_VIEW_LEVEL_GONE	0	Do not display debugging information in the rendering control
TRTC_DEBUG_VIEW_LEVEL_STATUS	1	Display audio/video statistics in the rendering control
TRTC_DEBUG_VIEW_LEVEL_ALL	2	Display audio/video statistics and key historical events in the rendering control

TRTCAudioRecordingContent

TRTCAudioRecordingContent

Audio recording content type

This enumerated type is used in the audio recording API startAudioRecording to specify the content of the recorded audio.

Enum	Value	DESC
TRTC_AudioRecordingContent_All	0	Record both local and remote audio
TRTC_AudioRecordingContent_Local	1	Record local audio only
TRTC_AudioRecordingContent_Remote	2	Record remote audio only

TRTCPublishMode

TRTCPublishMode

The publishing mode

This enum type is used by the publishing API startPublishMediaStream.
TRTC can mix multiple streams in a room and publish the mixed stream to a CDN or to a TRTC room. It can also publish the stream of the local user to Tencent Cloud or a third-party CDN.
You can specify one of the following publishing modes to use:

Enum	Value	DESC
TRTC_PublishMode_Unknown	0	Undefined
TRTC_PublishBigStream_ToCdn	1	Use this parameter to publish the primary stream (TRTCVideoStreamTypeBig) in the room to Tencent Cloud or a third-party CDN (only RTMP is supported).
TRTC_PublishSubStream_ToCdn	2	Use this parameter to publish the substream (TRTCVideoStreamTypeSub) in the room to Tencent Cloud or a third-party CDN (only RTMP is supported).
TRTC_PublishMixStream_ToCdn	3	Use this parameter together with the encoding parameter TRTCStreamEncoderParam and On-Cloud MixTranscoding parameter TRTCStreamMixingConfig to transcode the streams you specify and publish the

		mixed stream to Tencent Cloud or a third-party CDN (only RTMP is supported).
TRTC_PublishMixStream_ToRoom	4	Use this parameter together with the encoding parameter TRTCStreamEncoderParam and On-Cloud MixTranscoding parameter TRTCStreamMixingConfig to transcode the streams you specify and publish the mixed stream to the room you specify. Use `TRTCUser` in TRTCPublishTarget to specify the robot that publishes the transcoded stream to a TRTC room.

TRTCEncryptionAlgorithm

TRTCEncryptionAlgorithm

Encryption Algorithm

This enumeration type is used for media stream private encryption algorithm selection.

Enum	Value	DESC
TRTC_EncryptionAlgorithm_Aes_128_Gcm	0	AES GCM 128。
TRTC_EncryptionAlgorithm_Aes_256_Gcm	1	AES GCM 256。

TRTCSpeedTestScene

TRTCSpeedTestScene

Speed Test Scene

This enumeration type is used for speed test scene selection.

Enum	Value	DESC
TRTC_SpeedTestScene_Delay_Testing	1	Delay testing.
TRTC_SpeedTestScene_Delay_Bandwidth_Testing	2	Delay and bandwidth testing.
TRTC_SpeedTestScene_Online_Chorus_Testing	3	Online chorus testing.

TRTCGravitySensorAdaptiveMode

TRTCGravitySensorAdaptiveMode

Set the adaptation mode of gravity sensing (only applicable to mobile terminals)

Enum	Value	DESC
TRTC_GRAVITY_SENSOR_ADAPTIVE_MODE_DISABLE	0	Turn off the gravity sensor and make a decision based on the current acquisition resolution and the set encoding resolution. If the two are inconsistent, rotate 90 degrees to ensure the maximum frame.
TRTC_GRAVITY_SENSOR_ADAPTIVE_MODE_FILL_BY_CENTER_CROP	1	Turn on the gravity sensor to always ensure that the remote screen image is positive. When the intermediate process needs to deal with inconsistent resolutions, use the center cropping mode.
TRTC_GRAVITY_SENSOR_ADAPTIVE_MODE_FIT_WITH_BLACK_BORDER	2	Turn on the gravity sensor to always ensure that the remote screen image is positive. When the resolution needs to be processed inconsistently in

		the intermediate process, use the superimposed black border mode.

TRTCParams

TRTCParams

Room entry parameters

As the room entry parameters in the TRTC SDK, these parameters must be correctly set so that the user can successfully enter the audio/video room specified by `roomId` or `strRoomId` .
For historical reasons, TRTC supports two types of room IDs: `roomId` and `strRoomId` .
Note: do not mix `roomId` and `strRoomId` , because they are not interchangeable. For example, the number `123` and the string `123` are two completely different rooms in TRTC.

EnumType	DESC
businessInfo	Field description: business data, which is optional. This field is needed only by some advanced features. Recommended value: do not set this field on your own.
privateMapKey	Field description: permission credential used for permission control, which is optional. If you want only users with the specified `userId` values to enter a room, you need to use `privateMapKey` to restrict the permission. Recommended value: we recommend you use this parameter only if you have high security requirements. For more information, please see Enabling Advanced Permission Control.
role	Field description: role in the live streaming scenario, which is applicable only to the live streaming scenario (TRTCAppSceneLIVE or TRTCAppSceneVoiceChatRoom) but doesn't take effect in the call scenario. Recommended value: default value: anchor (TRTCRoleAnchor).
roomId	Field description: numeric room ID. Users (userId) in the same room can see one another and make audio/video calls. Recommended value: value range: 1–4294967294. @note `roomId` and `strRoomId` are mutually exclusive. If you decide to use `strRoomId` , then `roomId` should be entered as 0. If both are entered, `roomId` will be used. **Note**

	do not mix `roomId` and `strRoomId` , because they are not interchangeable. For example, the number `123` and the string `123` are two completely different rooms in TRTC.
sdkAppId	Field description: application ID, which is required. Tencent Cloud generates bills based on `sdkAppId` . Recommended value: the ID can be obtained on the account information page in the TRTC console after the corresponding application is created.
strRoomId	Field description: string-type room ID. Users (userId) in the same room can see one another and make audio/video calls. @note `roomId` and `strRoomId` are mutually exclusive. If you decide to use `strRoomId` , then `roomId` should be entered as 0. If both are entered, `roomId` will be used. **Note** do not mix `roomId` and `strRoomId` , because they are not interchangeable. For example, the number `123` and the string `123` are two completely different rooms in TRTC. Recommended value: the length limit is 64 bytes. The following 89 characters are supported: Uppercase and lowercase letters (a–z and A–Z) Digits (0–9) Space, "!", "#", "$", "%", "&", "(", ")", "+", "-", ":", ";", "<", "=", ".", ">", "?", "@", "[", "]", "^", "_", "{", "}", "|", "~", and ",".
streamId	Field description: specified `streamId` in Tencent Cloud CSS, which is optional. After setting this field, you can play back the user's audio/video stream on Tencent Cloud CSS CDN through a standard pull scheme (FLV or HLS). Recommended value: this parameter can contain up to 64 bytes and can be left empty. We recommend you use `sdkappid_roomid_userid_main` as the `streamid` , which is easier to identify and will not cause conflicts in your multiple applications. **Note** to use Tencent Cloud CSS CDN, you need to enable the auto-relayed live streaming feature on the "Function Configuration" page in the console first. For more information, please see CDN Relayed Live Streaming.
userDefineRecordId	Field description: on-cloud recording field, which is optional and used to specify whether to record the user's audio/video stream in the cloud. For more information, please see On-Cloud Recording and Playback. Recommended value: it can contain up to 64 bytes. Letters (a–z and A–Z), digits (0–9), underscores, and hyphens are allowed. Scheme 1. Manual recording 1. Enable on-cloud recording in "Application Management" > "On-cloud Recording Configuration" in the console. 2. Set "Recording Mode" to "Manual Recording".

	3. After manual recording is set, in a TRTC room, only users with the `userDefineRecordId` parameter set will have video recording files in the cloud, while users without this parameter set will not. 4. The recording file will be named in the format of "userDefineRecordId_start time_end time" in the cloud. Scheme 2. Auto-recording 1. You need to enable on-cloud recording in "Application Management" > "On-cloud Recording Configuration" in the console. 2. Set "Recording Mode" to "Auto-recording". 3. After auto-recording is set, any user who upstreams audio/video in a TRTC room will have a video recording file in the cloud. 4. The file will be named in the format of "userDefineRecordId_start time_end time". If `userDefineRecordId` is not specified, the file will be named in the format of "streamId_start time_end time".
userId	Field description: user ID, which is required. It is the `userId` of the local user in UTF-8 encoding and acts as the username. Recommended value: if the ID of a user in your account system is "mike", `userId` can be set to "mike".
userSig	Field description: user signature, which is required. It is the authentication signature corresponding to the current `userId` and acts as the login password for Tencent Cloud services. Recommended value: for the calculation method, please see UserSig.

TRTCVideoEncParam

TRTCVideoEncParam

Video encoding parameters

These settings determine the quality of image viewed by remote users as well as the image quality of recorded video files in the cloud.

EnumType	DESC
enableAdjustRes	Field description: whether to allow dynamic resolution adjustment. Once enabled, this field will affect on-cloud recording. Recommended value: this feature is suitable for scenarios that don't require on-cloud recording. After it is enabled, the SDK will intelligently select a suitable resolution according to the current network conditions to avoid the inefficient encoding mode of "large resolution + small bitrate". **Note**

	default value: false. If you need on-cloud recording, please do not enable this feature, because if the video resolution changes, the MP4 file recorded in the cloud cannot be played back normally by common players.
minVideoBitrate	Field description: minimum video bitrate. The SDK will reduce the bitrate to as low as the value specified by `minVideoBitrate` to ensure the smoothness only if the network conditions are poor. Note: default value: 0, indicating that a reasonable value of the lowest bitrate will be automatically calculated by the SDK according to the resolution you specify. Recommended value: you can set the `videoBitrate` and `minVideoBitrate` parameters at the same time to restrict the SDK's adjustment range of the video bitrate: If you want to "ensure clarity while allowing lag in weak network environments", you can set `minVideoBitrate` to 60% of `videoBitrate` . If you want to "ensure smoothness while allowing blur in weak network environments", you can set `minVideoBitrate` to a low value, for example, 100 Kbps. If you set `videoBitrate` and `minVideoBitrate` to the same value, it is equivalent to disabling the adaptive adjustment capability of the SDK for the video bitrate.
videoBitrate	Field description: target video bitrate. The SDK encodes streams at the target video bitrate and will actively reduce the bitrate only in weak network environments. Recommended value: please see the optimal bitrate for each specification in `TRTCVideoResolution` . You can also slightly increase the optimal bitrate. For example, `TRTCVideoResolution_1280_720` corresponds to the target bitrate of 1,200 Kbps. You can also set the bitrate to 1,500 Kbps for higher definition. **Note** you can set the `videoBitrate` and `minVideoBitrate` parameters at the same time to restrict the SDK's adjustment range of the video bitrate: If you want to "ensure clarity while allowing lag in weak network environments", you can set `minVideoBitrate` to 60% of `videoBitrate` . If you want to "ensure smoothness while allowing blur in weak network environments", you can set `minVideoBitrate` to a low value, for example, 100 Kbps. If you set `videoBitrate` and `minVideoBitrate` to the same value, it is equivalent to disabling the adaptive adjustment capability of the SDK for the video bitrate.
videoFps	Field description: video capturing frame rate Recommended value: 15 or 20 fps. If the frame rate is lower than 5 fps, there will be obvious lagging; if lower than 10 fps but higher than 5 fps, there will be

	slight lagging; if higher than 20 fps, the bandwidth will be wasted (the frame rate of movies is generally 24 fps). **Note** the front cameras on certain Android phones do not support a capturing frame rate higher than 15 fps. For some Android phones that focus on beautification features, the capturing frame rate of the front cameras may be lower than 10 fps.
videoResolution	Field description: video resolution Recommended value For mobile video call, we recommend you select a resolution of 360x640 or below and select `Portrait` (portrait resolution) for `resMode` . For mobile live streaming, we recommend you select a resolution of 540x960 and select `Portrait` (portrait resolution) for `resMode` . For desktop platforms (Windows and macOS), we recommend you select a resolution of 640x360 or above and select `Landscape` (landscape resolution) for `resMode` . **Note** to use a portrait resolution, please specify `resMode` as `Portrait` ; for example, when used together with `Portrait` , 640x360 represents 360x640.
videoResolutionMode	Field description: resolution mode (landscape/portrait) Recommended value: for mobile platforms (iOS and Android), `Portrait` is recommended; for desktop platforms (Windows and macOS), `Landscape` is recommended. **Note** to use a portrait resolution, please specify `resMode` as `Portrait` ; for example, when used together with `Portrait` , 640x360 represents 360x640.

TRTCNetworkQosParam

TRTCNetworkQosParam

Network QoS control parameter set

Network QoS control parameter. The settings determine the QoS control policy of the SDK in weak network conditions (e.g., whether to "ensure clarity" or "ensure smoothness").

EnumType	DESC
controlMode	Field description: QoS control mode (disused) Recommended value: on-cloud control

	Note please set the on-cloud control mode (TRTCQosControlModeServer).
preference	Field description: whether to ensure smoothness or clarity Recommended value: ensuring clarity **Note** this parameter mainly affects the audio/video performance of TRTC in weak network environments: Ensuring smoothness: in this mode, when the current network is unable to transfer a clear and smooth video image, the smoothness of the image will be given priority, but there will be blurs. See TRTC_VIDEO_QOS_PREFERENCE_SMOOTH Ensuring clarity (default value): in this mode, when the current network is unable to transfer a clear and smooth video image, the clarity of the image will be given priority, but there will be lags. See TRTC_VIDEO_QOS_PREFERENCE_CLEAR

TRTCRenderParams

TRTCRenderParams

Rendering parameters of video image

You can use these parameters to control the video image rotation angle, fill mode, and mirror mode.

EnumType	DESC
fillMode	Field description: image fill mode Recommended value: fill (the image may be stretched or cropped) or fit (there may be black bars in unmatched areas). Default value: TRTCVideoFillMode_Fill
mirrorType	Field description: image mirror mode Recommended value: default value: TRTCVideoMirrorType_Auto
rotation	Field description: clockwise image rotation angle Recommended value: rotation angles of 90, 180, and 270 degrees are supported. Default value: TRTCVideoRotation_0

TRTCQuality

TRTCQuality

Network quality

This indicates the quality of the network. You can use it to display the network quality of each user on the UI.

EnumType	DESC
quality	Network quality
userId	User ID

TRTCVolumeInfo

TRTCVolumeInfo

Volume

This indicates the audio volume value. You can use it to display the volume of each user in the UI.

EnumType	DESC
pitch	The local user's vocal frequency (unit: Hz), the value range is [0 - 4000]. For remote users, this value is always 0.
spectrumData	Audio spectrum data, which divides the sound frequency into 256 frequency domains, spectrumData records the energy value of each frequency domain, The value range of each energy value is [-300, 0] in dBFS. **Note** The local spectrum is calculated using the audio data before encoding, which will be affected by the capture volume, BGM, etc.; the remote spectrum is calculated using the received audio data, and operations such as adjusting the remote playback volume locally will not affect it.
userId	`userId` of the speaker. An empty value indicates the local user.
vad	Vad result of the local user. 0: not speech 1: speech.
volume	Volume of the speaker. Value range: 0–100.

TRTCSpeedTestParams

TRTCSpeedTestParams

Network speed testing parameters

You can test the network speed through the startSpeedTest: interface before the user enters the room (this API cannot be called during a call).

EnumType	DESC

expectedDownBandwidth	Expected downstream bandwidth (kbps, value range: 10 to 5000, no downlink bandwidth test when it is 0). **Note** When the parameter `scene` is set to `TRTCSpeedTestScene_OnlineChorusTesting` , in order to obtain more accurate information such as rtt / jitter, the value range is limited to 10 ~ 1000.
expectedUpBandwidth	Expected upstream bandwidth (kbps, value range: 10 to 5000, no uplink bandwidth test when it is 0). **Note** When the parameter `scene` is set to `TRTCSpeedTestScene_OnlineChorusTesting` , in order to obtain more accurate information such as rtt / jitter, the value range is limited to 10 ~ 1000.
scene	Speed test scene.
sdkAppId	Application identification, please refer to the relevant instructions in TRTCParams.
userId	User identification, please refer to the relevant instructions in TRTCParams.
userSig	User signature, please refer to the relevant instructions in TRTCParams.

TRTCSpeedTestResult

TRTCSpeedTestResult

Network speed test result

The startSpeedTest: API can be used to test the network speed before a user enters a room (this API cannot be called during a call).

EnumType	DESC
availableDownBandwidth	Downstream bandwidth (in kbps, -1: invalid value).
availableUpBandwidth	Upstream bandwidth (in kbps, -1: invalid value).
downJitter	Downlink data packet jitter (ms) refers to the stability of data communication in the user's current network environment. The smaller the value, the better. The normal value range is 0ms - 100ms. -1 means that the speed test failed to obtain an effective value. Generally, the Jitter of the WiFi network will be slightly larger than that of the 4G/5G environment.

downLostRate	Downstream packet loss rate between 0 and 1.0. For example, 0.2 indicates that 2 data packets may be lost in every 10 packets received from the server.
errMsg	Error message for network speed test.
ip	Server IP address.
quality	Network quality, which is tested and calculated based on the internal evaluation algorithm. For more information, please see TRTCQuality
rtt	Delay in milliseconds, which is the round-trip time between the current device and TRTC server. The smaller the value, the better. The normal value range is 10–100 ms.
success	Whether the network speed test is successful.
upJitter	Uplink data packet jitter (ms) refers to the stability of data communication in the user's current network environment. The smaller the value, the better. The normal value range is 0ms - 100ms. -1 means that the speed test failed to obtain an effective value. Generally, the Jitter of the WiFi network will be slightly larger than that of the 4G/5G environment.
upLostRate	Upstream packet loss rate between 0 and 1.0. For example, 0.3 indicates that 3 data packets may be lost in every 10 packets sent to the server.

TRTCTexture

TRTCTexture

Video texture data

EnumType	DESC
eglContext10	Field description: OpenGL context defined by `(javax.microedition.khronos.egl.*)`
eglContext14	Field description: OpenGL context defined by `(android.opengl.*)`
textureId	Field description: video texture ID

TRTCVideoFrame

TRTCVideoFrame

Video frame information

`TRTCVideoFrame` is used to describe the raw data of a frame of the video image, which is the image data before frame encoding or after frame decoding.

EnumType	DESC
buffer	Field description: video data when `bufferType` is TRTCCloudDef#TRTC_VIDEO_BUFFER_TYPE_BYTE_BUFFER, which carries the `Direct Buffer` used for the JNI layer.
bufferType	Field description: video data structure type
data	Field description: video data when `bufferType` is TRTCCloudDef#TRTC_VIDEO_BUFFER_TYPE_BYTE_ARRAY, which carries the byte array used for the Java layer.
height	Field description: video height Recommended value: please enter the height of the video data passed in.
pixelFormat	Field description: video pixel format
rotation	Field description: clockwise rotation angle of video pixels
texture	Field description: video data when `bufferType` is TRTCCloudDef#TRTC_VIDEO_PIXEL_FORMAT_Texture_2D, which carries the texture data used for OpenGL rendering.
timestamp	Field description: video frame timestamp in milliseconds Recommended value: this parameter can be set to 0 for custom video capturing. In this case, the SDK will automatically set the `timestamp` field. However, please "evenly" set the calling interval of `sendCustomVideoData` .
width	Field description: video width Recommended value: please enter the width of the video data passed in.

TRTCAudioFrame

TRTCAudioFrame

Audio frame data

EnumType	DESC

channel	Field description: number of sound channels
data	Field description: audio data
extraData	Field description: extra data in audio frame, message sent by remote users through `onLocalProcessedAudioFrame` that add to audio frame will be callback through this field.
sampleRate	Field description: sample rate
timestamp	Field description: timestamp in ms

TRTCMixUser

TRTCMixUser

Description information of each video image in On-Cloud MixTranscoding

`TRTCMixUser` is used to specify the location, size, layer, and stream type of each video image in On-Cloud MixTranscoding.

EnumType	DESC
height	Field description: specify the height of this video image in px
image	Field description: specify the placeholder or watermark image. The placeholder image will be displayed when there is no upstream video.A watermark image is a semi-transparent image posted in the mixed image, and this image will always be overlaid on the mixed image. When the `inputType` field is set to TRTCMixInputTypePureAudio, the image is a placeholder image, and you need to specify `userId` . When the `inputType` field is set to TRTCMixInputTypeWatermark, the image is a watermark image, and you don't need to specify `userId` . Recommended value: default value: null, indicating not to set the placeholder or watermark image. **Note** TRTC's backend service will mix the image specified by the URL address into the final stream.URL link length is limited to 512 bytes. The image size is limited to 10MB.Support png, jpg, jpeg, bmp format. Take effects iff the `inputType` field is set to TRTCMixInputTypePureAudio or TRTCMixInputTypeWatermark.
inputType	Field description: specify the mixed content of this stream (audio only, video only, audio and video, or watermark). Recommended value: default value: TRTCMixInputTypeUndefined. **Note**

	When specifying `inputType` as TRTCMixInputTypeUndefined and specifying `pureAudio` to YES, it is equivalent to setting `inputType` to `TRTCMixInputTypePureAudio` . When specifying `inputType` as TRTCMixInputTypeUndefined and specifying `pureAudio` to NO, it is equivalent to setting `inputType` to `TRTCMixInputTypeAudioVideo` . When specifying `inputType` as TRTCMixInputTypeWatermark, you don't need to specify the `userId` field, but you need to specify the `image` field.
pureAudio	Field description: specify whether this stream mixes audio only Recommended value: default value: false **Note** this field has been disused. We recommend you use the new field `inputType` introduced in v8.5.
renderMode	Field description: specify the display mode of this stream. Recommended value: default value: 0. 0 is cropping, 1 is zooming, 2 is zooming and displaying black background. **Note** image doesn't support setting `renderMode` temporarily, the default display mode is forced stretch.
roomId	Field description: ID of the room where this audio/video stream is located (an empty value indicates the local room ID)
soundLevel	Field description: specify the target volumn level of On-Cloud MixTranscoding. (value range: 0-100) Recommended value: default value: 100.
streamType	Field description: specify whether this video image is the primary stream image (TRTCVideoStreamTypeBig) or substream image (TRTCVideoStreamTypeSub).
userId	Field description: user ID
width	Field description: specify the width of this video image in px
x	Field description: specify the X coordinate of this video image in px
y	Field description: specify the Y coordinate of this video image in px
zOrder	Field description: specify the level of this video image (value range: 1–15; the value must be unique)

TRTCTranscodingConfig

TRTCTranscodingConfig

Layout and transcoding parameters of On-Cloud MixTranscoding

These parameters are used to specify the layout position information of each video image and the encoding parameters of mixtranscoding during On-Cloud MixTranscoding.

EnumType	DESC
appId	Field description: `appId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `appId` in `Relayed Live Streaming Info`.
audioBitrate	Field description: specify the target audio bitrate of On-Cloud MixTranscoding Recommended value: default value: 64 Kbps. Value range: [32,192].
audioChannels	Field description: specify the number of sound channels of On-Cloud MixTranscoding Recommended value: default value: 1, which means mono channel. Valid values: 1: mono channel; 2: dual channel.
audioCodec	Field description: specify the audio encoding type of On-Cloud MixTranscoding Recommended value: default value: 0, which means LC-AAC. Valid values: 0: LC-AAC; 1: HE-AAC; 2: HE-AACv2. **Note** HE-AAC and HE-AACv2 only support [48000, 44100, 32000, 24000, 16000] sample rate. HE-AACv2 only support dual channel. HE-AAC and HE-AACv2 take effects iff the output streamId is specified.
audioSampleRate	Field description: specify the target audio sample rate of On-Cloud MixTranscoding Recommended value: default value: 48000 Hz. Valid values: 12000 Hz, 16000 Hz, 22050 Hz, 24000 Hz, 32000 Hz, 44100 Hz, 48000 Hz.
backgroundColor	Field description: specify the background color of the mixed video image. Recommended value: default value: 0x000000, which means black and is in the format of hex number; for example: "0x61B9F1" represents the RGB color (97,158,241).
backgroundImage	Field description: specify the background image of the mixed video image. **Recommended value: default value: null, indicating not to set the background image. **Note** TRTC's backend service will mix the image specified by the URL address into the final stream.URL link length is limited to 512 bytes. The image size is limited to 10MB.Support png, jpg, jpeg, bmp format.

bizId	Field description: `bizId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `bizId` in `Relayed Live Streaming Info` .
mixUsers	Field description: specify the position, size, layer, and stream type of each video image in On-Cloud MixTranscoding Recommended value: this field is an array in `TRTCMixUser` type, where each element represents the information of a video image.
mode	Field description: layout mode Recommended value: please choose a value according to your business needs. The preset mode has better applicability.
streamId	Field description: ID of the live stream output to CDN Recommended value: default value: null, that is, the audio/video streams in the room will be mixed into the audio/video stream of the caller of this API. If you don't set this parameter, the SDK will execute the default logic, that is, it will mix the multiple audio/video streams in the room into the audio/video stream of the caller of this API, i.e., A + B => A. If you set this parameter, the SDK will mix the audio/video streams in the room into the live stream you specify, i.e., A + B => C (C is the `streamId` you specify).
videoBitrate	Field description: specify the target video bitrate (Kbps) of On-Cloud MixTranscoding Recommended value: if you enter 0, TRTC will estimate a reasonable bitrate value based on `videoWidth` and `videoHeight` . You can also refer to the recommended bitrate value in the video resolution enumeration definition (in the comment section).
videoFramerate	Field description: specify the target video frame rate (fps) of On-Cloud MixTranscoding Recommended value: default value: 15 fps. Value range: (0,30].
videoGOP	Field description: specify the target video keyframe interval (GOP) of On-Cloud MixTranscoding Recommended value: default value: 2 (in seconds). Value range: [1,8].
videoHeight	Field description: specify the target resolution (height) of On-Cloud MixTranscoding Recommended value: 640 px. If you only mix audio streams, please set both `width` and `height` to 0; otherwise, there will be a black background in the live stream after mixtranscoding.
videoSeiParams	Field description: SEI parameters. default value: null **Note** the parameter is passed in the form of a JSON string. Here is an example to use it: `` `json

	{
	"payLoadContent":"xxx",
	"payloadType":5,
	"payloadUuid":"1234567890abcdef1234567890abcdef",
	"interval":1000,
	"followIdr":false
	}
	`
	The currently supported fields and their meanings are as follows:
	payloadContent: Required. The payload content of the passthrough SEI, which cannot be empty.
	payloadType: Required. The type of the SEI message, with a value range of 5 or an integer within the range of [100, 254] (excluding 244, which is an internally defined timestamp SEI).
	payloadUuid: Required when payloadType is 5, and ignored in other cases. The value must be a 32-digit hexadecimal number.
	interval: Optional, default is 1000. The sending interval of the SEI, in milliseconds.
	followIdr: Optional, default is false. When this value is true, the SEI will be ensured to be carried when sending a key frame, otherwise it is not guaranteed.

videoWidth	Field description: specify the target resolution (width) of On-Cloud MixTranscoding Recommended value: 360 px. If you only mix audio streams, please set both `width` and `height` to 0; otherwise, there will be a black background in the live stream after mixtranscoding.

TRTCPublishCDNParam

TRTCPublishCDNParam

Push parameters required to be set when publishing audio/video streams to non-Tencent Cloud CDN

TRTC's backend service supports publishing audio/video streams to third-party live CDN service providers through the standard RTMP protocol.

If you use the Tencent Cloud CSS CDN service, you don't need to care about this parameter; instead, just use the startPublish API.

EnumType	DESC
appId	Field description: `appId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `appId` in `Relayed Live Streaming Info` .
bizId	Field description: `bizId` of Tencent Cloud CSS

	Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `bizId` in `Relayed Live Streaming Info` .
streamId	Field description: specify the push address (in RTMP format) of this audio/video stream at the third-party live streaming service provider Recommended value: default value: null,that is, the audio/video streams in the room will be pushed to the target service provider of the caller of this API.
url	Field description: specify the push address (in RTMP format) of this audio/video stream at the third-party live streaming service provider Recommended value: the push URL rules vary greatly by service provider. Please enter a valid push URL according to the requirements of the target service provider. TRTC's backend server will push audio/video streams in the standard format to the third-party service provider according to the URL you enter. **Note** the push URL must be in RTMP format and meet the specifications of your target live streaming service provider; otherwise, the target service provider will reject the push requests from TRTC's backend service.

TRTCAudioRecordingParams

TRTCAudioRecordingParams

Local audio file recording parameters

This parameter is used to specify the recording parameters in the audio recording API startAudioRecording.

EnumType	DESC
filePath	Field description: storage path of the audio recording file, which is required. **Note** this path must be accurate to the file name and extension. The extension determines the format of the audio recording file. Currently, supported formats include PCM, WAV, and AAC. For example, if you specify the path as `mypath/record/audio.aac` , it means that you want the SDK to generate an audio recording file in AAC format.Please specify a valid path with read/write permissions; otherwise, the audio recording file cannot be generated.
maxDurationPerFile	Field description: `maxDurationPerFile` is the max duration of each recorded file segments, in milliseconds, with a minimum value of 10000. The default value is 0, indicating no segmentation.
recordingContent	Field description: Audio recording content type.

	Note: Record all local and remote audio by default.

TRTCLocalRecordingParams

TRTCLocalRecordingParams

Local media file recording parameters

This parameter is used to specify the recording parameters in the local media file recording API startLocalRecording. The `startLocalRecording` API is an enhanced version of the `startAudioRecording` API. The former can record video files, while the latter can only record audio files.

EnumType	DESC
filePath	Field description: address of the recording file, which is required. Please ensure that the path is valid with read/write permissions; otherwise, the recording file cannot be generated. **Note** this path must be accurate to the file name and extension. The extension determines the format of the recording file. Currently, only the MP4 format is supported. For example, if you specify the path as `mypath/record/test.mp4` , it means that you want the SDK to generate a local video file in MP4 format. Please specify a valid path with read/write permissions; otherwise, the recording file cannot be generated.
interval	Field description: `interval` is the update frequency of the recording information in milliseconds. Value range: 1000–10000. Default value: -1, indicating not to call back
maxDurationPerFile	Field description: `maxDurationPerFile` is the max duration of each recorded file segments, in milliseconds, with a minimum value of 10000. The default value is 0, indicating no segmentation.
recordType	Field description: media recording type, which is `TRTCRecordTypeBoth` by default, indicating to record both audio and video.

TRTCSwitchRoomConfig

TRTCSwitchRoomConfig

Room switch parameter

This parameter is used for the room switch API switchRoom, which can quickly switch a user from one room to another.

EnumType	DESC
privateMapKey	Field description: permission credential used for permission control, which is optional. If you want only users with the specified `userId` values to enter a room, you need to use `privateMapKey` to restrict the permission. Recommended value: we recommend you use this parameter only if you have high security requirements. For more information, please see Enabling Advanced Permission Control.
roomId	Field description: numeric room ID, which is optional. Users in the same room can see one another and make audio/video calls. Recommended value: value range: 1-4294967294. **Note** either `roomId` or `strRoomId` must be entered. If both are entered, `roomId` will be used.
strRoomId	Field description: string-type room ID, which is optional. Users in the same room can see one another and make audio/video calls. **Note** either `roomId` or `strRoomId` must be entered. If both are entered, `roomId` will be used.
userSig	Field description: user signature, which is optional. It is the authentication signature corresponding to the current `userId` and acts as the login password. If you don't specify the newly calculated `userSig` during room switch, the SDK will continue to use the `userSig` you specified during room entry (enterRoom). This requires you to ensure that the old `userSig` is still within the validity period allowed by the signature at the moment of room switch; otherwise, room switch will fail. Recommended value: for the calculation method, please see UserSig.

TRTCAudioFrameDelegateFormat

TRTCAudioFrameDelegateFormat

Format parameter of custom audio callback

This parameter is used to set the relevant format (including sample rate and number of channels) of the audio data called back by the SDK in the APIs related to custom audio callback.

EnumType	DESC
channel	Field description: number of sound channels

	Recommended value: default value: 1, which means mono channel. Valid values: 1: mono channel; 2: dual channel.
mode	Field description: audio callback data operation mode Recommended value: TRTCAudioFrameOperationModeReadOnly, get audio data from callback only. The modes that can be set are TRTCAudioFrameOperationModeReadOnly, TRTCAudioFrameOperationModeReadWrite.
sampleRate	Field description: sample rate Recommended value: default value: 48000 Hz. Valid values: 16000, 32000, 44100, 48000.
samplesPerCall	Field description: number of sample points Recommended value: the value must be an integer multiple of sampleRate/100.

TRTCScreenShareParams

TRTCScreenShareParams

Screen sharing parameter (for Android only)

This parameter is used to specify the floating window and other related information during screen sharing in the screen sharing API startScreenCapture.

EnumType	DESC
enableForegroundService	@deprecated Begin from v11.8 version, in order to adapt to targetSdkVersion 34 and above, screen sharing will default to launching a built-in foreground service. This value setting will be invalid.
floatingView	Field description: you can set a floating view through this parameter. Recommended value: starting from Android 7.0, applications running in the background with no session keep-alive configured will be force stopped by the Android system very soon. However, when an application is sharing the screen, it will inevitably be switched to the system background. In this case, if a floating window can pop up, it can prevent the application from being force stopped by the system. In addition, the pop-up floating window also informs the user of the ongoing screen sharing, helping remind the user to avoid the leakage of confidential information. **Note** you can also use the `WindowsManager` API of Android to achieve the same effect.

mediaProjection	Field description: you can set a MediaProjection to SDK through this parameter. Recommended value: this parameter can be set as null normally.

TRTCUser

TRTCUser

The users whose streams to publish

You can use this parameter together with the publishing destination parameter TRTCPublishTarget and On-Cloud MixTranscoding parameter TRTCStreamMixingConfig to transcode the streams you specify and publish the mixed stream to the destination you specify.

EnumType	DESC
intRoomId	**Description:** Numeric room ID. The room ID must be of the same type as that in TRTCParams. **Value:** Value range: 1-4294967294 **Note:** You cannot use both `intRoomId` and `strRoomId`. If you specify `strRoomId`, you need to set `intRoomId` to `0`. If you set both, only `intRoomId` will be used.
strRoomId	**Description:** String-type room ID. The room ID must be of the same type as that in TRTCParams. **Note:** You cannot use both `intRoomId` and `strRoomId`. If you specify `roomId`, you need to leave `strRoomId` empty. If you set both, only `intRoomId` will be used. **Value:** 64 bytes or shorter; supports the following character set (89 characters): Uppercase and lowercase letters (a-z and A-Z) Numbers (0-9) Space, "!", "#", "$", "%", "&", "(", ")", "+", "-", ":", ";", "<", "=", ".", ">", "?", "@", "[", "]", "^", "_", "{", "}", "|", "~", ","
userId	/**Description**: UTF-8-encoded user ID (required) **Value:** For example, if the ID of a user in your account system is "mike", set it to `mike`.

TRTCPublishCdnUrl

TRTCPublishCdnUrl

The destination URL when you publish to Tencent Cloud or a third-party CDN

This enum type is used by the publishing destination parameter TRTCPublishTarget of the publishing API startPublishMediaStream.

EnumType	DESC
isInternalLine	**Description:** Whether to publish to Tencent Cloud **Value:** The default value is `true`. **Note:** If the destination URL you set is provided by Tencent Cloud, set this parameter to `true`, and you will not be charged relaying fees.
rtmpUrl	**Description:** The destination URL (RTMP) when you publish to Tencent Cloud or a third-party CDN. **Value:** The URLs of different CDN providers may vary greatly in format. Please enter a valid URL as required by your service provider. TRTC's backend server will push audio/video streams in the standard format to the URL you provide. **Note:** The URL must be in RTMP format. It must also meet the requirements of your service provider, or your service provider may reject push requests from the TRTC backend.

TRTCPublishTarget

TRTCPublishTarget

The publishing destination

This enum type is used by the publishing API startPublishMediaStream.

EnumType	DESC
cdnUrlList	`Description:` The destination URLs (RTMP) when you publish to Tencent Cloud or third-party CDNs. `Note:` You don’t need to set this parameter if you set the publishing mode to `TRTCPublishMixStreamToRoom`.
mixStreamIdentity	`Description:` The information of the robot that publishes the transcoded stream to a TRTC room. `Note:` You need to set this parameter only if you set the publishing mode to `TRTCPublishMixStreamToRoom`. `Note:` After you set this parameter, the stream will be pushed to the room you specify. We recommend you set it to a special user ID to distinguish the robot from the anchor who enters the room via the TRTC SDK. `Note:` Users whose streams are transcoded cannot subscribe to the transcoded stream.

	`Note:` If you set the subscription mode (@link setDefaultStreamRecvMode}) to manual before room entry, you need to manage the streams to receive by yourself (normally, if you receive the transcoded stream, you need to unsubscribe from the streams that are transcoded). `Note:` If you set the subscription mode (setDefaultStreamRecvMode) to auto before room entry, users whose streams are not transcoded will receive the transcoded stream automatically and will unsubscribe from the users whose streams are transcoded. You call muteRemoteVideoStream and muteRemoteAudio to unsubscribe from the transcoded stream.
mode	`Description:` The publishing mode. `Value:` You can relay streams to a CDN, transcode streams, or publish streams to an RTC room. Select the mode that fits your needs. **Note** If you need to use more than one publishing mode, you can call startPublishMediaStream multiple times and set `TRTCPublishTarget` to a different value each time.You can use one mode each time you call the startPublishMediaStream) API. To modify the configuration, call updatePublishCDNStream.

TRTCVideoLayout

TRTCVideoLayout

The video layout of the transcoded stream

This enum type is used by the On-Cloud MixTranscoding parameter TRTCStreamMixingConfig of the publishing API startPublishMediaStream.

You can use this parameter to specify the position, size, layer, and stream type of each video in the transcoded stream.

EnumType	DESC
backgroundColor	`Description:` The background color of the mixed stream. `Value:` The value must be a hex number. For example, "0x61B9F1" represents the RGB color value (97,158,241). Default value: 0x000000 (black).
fillMode	`Description:` The rendering mode. `Value:` The rendering mode may be fill (the image may be stretched or cropped) or fit (there may be black bars). Default value: TRTCVideoFillMode_Fill.
fixedVideoStreamType	`Description:` Whether the video is the primary stream

	(TRTCVideoStreamTypeBig) or substream (e TRTCVideoStreamTypeSub).
fixedVideoUser	`Description:` The users whose streams are transcoded. **Note** If you do not specify TRTCUser (`userId` , `intRoomId` , `strRoomId`), the TRTC backend will automatically mix the streams of anchors who are sending audio/video in the room according to the video layout you specify.
height	`Description:` The height (in pixels) of the video.
placeHolderImage	`Description:` The URL of the placeholder image. If a user sends only audio, the image specified by the URL will be mixed during On-Cloud MixTranscoding. `Value:` This parameter is left empty by default, which means no placeholder image will be used. **Note** You need to specify the `userId` parameter in `fixedVideoUser` . The URL can be 512 bytes long at most, and the image must not exceed 2 MB. The image can be in PNG, JPG, JPEG, or BMP format. We recommend you use a semitransparent image in PNG format.
width	`Description:` The width (in pixels) of the video.
x	`Description:` The X coordinate (in pixels) of the video.
y	`Description:` The Y coordinate (in pixels) of the video.
zOrder	`Description:` The layer of the video, which must be unique. Value range: 0-15.

TRTCWatermark

TRTCWatermark

The watermark layout

This enum type is used by the On-Cloud MixTranscoding parameter TRTCStreamMixingConfig of the publishing API startPublishMediaStream.

EnumType	DESC
height	`Description:` The height (in pixels) of the watermark.

watermarkUrl	`Description:` The URL of the watermark image. The image specified by the URL will be mixed during On-Cloud MixTranscoding. **Note** The URL can be 512 bytes long at most, and the image must not exceed 2 MB. The image can be in PNG, JPG, JPEG, or BMP format. We recommend you use a semitransparent image in PNG format.
width	`Description:` The width (in pixels) of the watermark.
x	`Description:` The X coordinate (in pixels) of the watermark.
y	`Description:` The Y coordinate (in pixels) of the watermark.
zOrder	`Description:` The layer of the watermark, which must be unique. Value range: 0-15.

TRTCStreamEncoderParam

TRTCStreamEncoderParam

The encoding parameters

`Description:` This enum type is used by the publishing API startPublishMediaStream.

`Note:` This parameter is required if you set the publishing mode to `TRTCPublish_MixStream_ToCdn` or `TRTCPublish_MixStream_ToRoom` in TRTCPublishTarget.

`Note:` If you use the relay to CDN feature (the publishing mode set to `RTCPublish_BigStream_ToCdn` or `TRTCPublish_SubStream_ToCdn`), to improve the relaying stability and playback compatibility, we also recommend you set this parameter.

EnumType	DESC
audioEncodedChannelNum	`Description:` The sound channels of the stream to publish. `Value:` Valid values: 1 (mono channel); 2 (dual-channel). Default: 1.
audioEncodedCodecType	`Description:` The audio codec of the stream to publish. `Value:` Valid values: 0 (LC-AAC); 1 (HE-AAC); 2 (HE-AACv2). Default: 0. **Note** The audio sample rates supported by HE-AAC and HE-AACv2 are 48000, 44100, 32000, 24000, and 16000. When HE-AACv2 is used, the output stream can only be dual-channel.
audioEncodedKbps	`Description:` The audio bitrate (Kbps) of the stream to publish. `Value:` Value range: [32,192]. Default: 50.

audioEncodedSampleRate	`Description:` The audio sample rate of the stream to publish. `Value:` Valid values: [48000, 44100, 32000, 24000, 16000, 8000]. Default: 48000 (Hz).
videoEncodedCodecType	`Description:` The video codec of the stream to publish. `Value:` Valid values: 0 (H264); 1 (H265). Default: 0.
videoEncodedFPS	`Description:` The frame rate (fps) of the stream to publish. `Value:` Value range: (0,30]. Default: 20.
videoEncodedGOP	`Description:` The keyframe interval (GOP) of the stream to publish. `Value:` Value range: [1,5]. Default: 3 (seconds).
videoEncodedHeight	`Description:` The resolution (height) of the stream to publish. `Value:` Recommended value: 640. If you mix only audio streams, to avoid displaying a black video in the transcoded stream, set both `width` and `height` to `0` .
videoEncodedKbps	`Description:` The video bitrate (Kbps) of the stream to publish. `Value:` If you set this parameter to `0` , TRTC will work out a bitrate based on `videoWidth` and `videoHeight` . For details, refer to the recommended bitrates for the constants of the resolution enum type (see comment).
videoEncodedWidth	`Description:` The resolution (width) of the stream to publish. `Value:` Recommended value: 368. If you mix only audio streams, to avoid displaying a black video in the transcoded stream, set both `width` and `height` to `0` .
videoSeiParams	`Description:` SEI parameters. Default: null `Note:` the parameter is passed in the form of a JSON string. Here is an example to use it:

```
{
  "payLoadContent":"xxx",
  "payloadType":5,
  "payloadUuid":"1234567890abcdef1234567890abcdef",
  "interval":1000,
  "followIdr":false
}
```

The currently supported fields and their meanings are as follows:

payloadContent: Required. The payload content of the passthrough SEI, which cannot be empty.

payloadType: Required. The type of the SEI message, with a value range of 5 or an integer within the range of [100, 254] (excluding 244, which is an internally defined timestamp SEI).

payloadUuid: Required when payloadType is 5, and ignored in other cases. The value must be a 32-digit hexadecimal number.

interval: Optional, default is 1000. The sending interval of the SEI, in milliseconds.

followIdr: Optional, default is false. When this value is true, the SEI will be ensured to be carried when sending a key frame, otherwise it is not guaranteed.

TRTCStreamMixingConfig

TRTCStreamMixingConfig

The transcoding parameters

This enum type is used by the publishing API startPublishMediaStream.

You can use this parameter to specify the video layout and input audio information for On-Cloud MixTranscoding.

EnumType	DESC
audioMixUserList	`Description:` The information of each audio stream to mix. `Value:` This parameter is an array. Each `TRTCUser` element in the array indicates the information of an audio stream. **Note** If you do not specify this array, the TRTC backend will automatically mix all streams of the anchors who are sending audio in the room according to the audio encode param TRTCStreamEncoderParam you specify (currently only supports up to 16 audio and video inputs).
backgroundColor	`Description:` The background color of the mixed stream. `Value:` The value must be a hex number. For example, "0x61B9F1" represents the RGB color value (97,158,241). Default value: 0x000000 (black).
backgroundImage	`Description:` The URL of the background image of the mixed stream. The image specified by the URL will be mixed during On-Cloud MixTranscoding. `Value:` This parameter is left empty by default, which means no background image will be used. **Note** The URL can be 512 bytes long at most, and the image must not exceed 2 MB. The image can be in PNG, JPG, JPEG, or BMP format. We recommend you use a semitransparent image in PNG format.
videoLayoutList	`Description:` The position, size, layer, and stream type of each video in On-Cloud MixTranscoding. `Value:` This parameter is an array. Each `TRTCVideoLayout` element in the array indicates the information of a video in On-Cloud MixTranscoding.
watermarkList	`Description:` The position, size, and layer of each watermark image in On-Cloud MixTranscoding. `Value:` This parameter is an array. Each `TRTCWatermark` element in the array indicates the information of a watermark.

TRTCPayloadPrivateEncryptionConfig

TRTCPayloadPrivateEncryptionConfig

Media Stream Private Encryption Configuration

This configuration is used to set the algorithm and key for media stream private encryption.

EnumType	DESC
encryptionAlgorithm	`Description:` Encryption algorithm, the default is TRTCEncryptionAlgorithmAes128Gcm.
encryptionKey	`Description:` encryption key, string type. `Value:` If the encryption algorithm is TRTCEncryptionAlgorithmAes128Gcm, the key length must be 16 bytes; if the encryption algorithm is TRTCEncryptionAlgorithmAes256Gcm, the key length must be 32 bytes.
encryptionSalt	`Description:` Salt, initialization vector for encryption. `Value:` It is necessary to ensure that the array filled in this parameter is not empty, not all 0 and the data length is 32 bytes.

TRTCAudioVolumeEvaluateParams

TRTCAudioVolumeEvaluateParams

Volume evaluation and other related parameter settings.

This setting is used to enable vocal detection and sound spectrum calculation.

EnumType	DESC
enablePitchCalculation	`Description:` Whether to enable local vocal frequency calculation.
enableSpectrumCalculation	`Description:` Whether to enable sound spectrum calculation.
enableVadDetection	`Description:` Whether to enable local voice detection. **Note** Call before startLocalAudio.
interval	`Description:` Set the trigger interval of the onUserVoiceVolume callback, the unit is milliseconds, the minimum interval is 100ms, if it is less than or equal to 0, the callback will be closed. `Value:` Recommended value: 300, in milliseconds. **Note**

	When the interval is greater than 0, the volume prompt will be enabled by default, no additional setting is required.

Deprecated Interface

Last updated：2024-06-06 15:50:05

Deprecate

DeprecatedTRTCCloud

FuncList	DESC
setListener	Set TRTC event callback
setBeautyStyle	Set the strength of beauty, brightening, and rosy skin filters.
setEyeScaleLevel	Set the strength of eye enlarging filter
setFaceSlimLevel	Set the strength of face slimming filter
setFaceVLevel	Set the strength of chin slimming filter
setChinLevel	Set the strength of chin lengthening/shortening filter
setFaceShortLevel	Set the strength of face shortening filter
setNoseSlimLevel	Set the strength of nose slimming filter
selectMotionTmpl	Set animated sticker
setMotionMute	Mute animated sticker
setFilter	Set color filter
setFilterConcentration	Set the strength of color filter
setGreenScreenFile	Set green screen video
setReverbType	Set reverb effect
setVoiceChangerType	Set voice changing type
enableAudioEarMonitoring	Enable or disable in-ear monitoring
enableAudioVolumeEvaluation	Enable volume reminder

enableAudioVolumeEvaluation	Enable volume reminder
switchCamera	Switch camera
isCameraZoomSupported	Query whether the current camera supports zoom
setZoom	Set camera zoom ratio (focal length)
isCameraTorchSupported	Query whether the device supports flash
enableTorch	Enable/Disable flash
isCameraFocusPositionInPreviewSupported	Query whether the camera supports setting focus
setFocusPosition	Set the focal position of camera
isCameraAutoFocusFaceModeSupported	Query whether the device supports the automatic recognition of face position
setSystemVolumeType	Setting the system volume type (for mobile OS)
checkAudioCapabilitySupport	Query whether a certain audio capability is supported (only for Android)
startLocalAudio	Set sound quality
startRemoteView	Start displaying remote video image
stopRemoteView	Stop displaying remote video image and pulling the video data stream of remote user
setLocalViewFillMode	Set the rendering mode of local image
setLocalViewRotation	Set the clockwise rotation angle of local image
setLocalViewMirror	Set the mirror mode of local camera's preview image
setRemoteViewFillMode	Set the fill mode of substream image
setRemoteViewRotation	Set the clockwise rotation angle of remote image
startRemoteSubStreamView	Start displaying the substream image of remote user
stopRemoteSubStreamView	Stop displaying the substream image of remote user
setRemoteSubStreamViewFillMode	Set the fill mode of substream image
setRemoteSubStreamViewRotation	Set the clockwise rotation angle of substream image
setAudioQuality	Set sound quality

setPriorRemoteVideoStreamType	Specify whether to view the big or small image
setMicVolumeOnMixing	Set mic volume
playBGM	Start background music
stopBGM	Stop background music
pauseBGM	Stop background music
resumeBGM	Stop background music
getBGMDuration	Get the total length of background music in ms
setBGMPosition	Set background music playback progress
setBGMVolume	Set background music volume
setBGMPlayoutVolume	Set the local playback volume of background music
setBGMPublishVolume	Set the remote playback volume of background music
playAudioEffect	Play sound effect
setAudioEffectVolume	Set sound effect volume
stopAudioEffect	Stop sound effect
stopAllAudioEffects	Stop all sound effects
setAllAudioEffectsVolume	Set the volume of all sound effects
pauseAudioEffect	Pause sound effect
resumeAudioEffect	Pause sound effect
enableCustomVideoCapture	Enable custom video capturing mode
sendCustomVideoData	Deliver captured video data to SDK
muteLocalVideo	Pause/Resume publishing local video stream
muteRemoteVideoStream	Pause/Resume subscribing to remote user's video stream
snapshotVideo	Screencapture video
startSpeedTest	Start network speed test (used before room entry)

startScreenCapture	Start screen sharing
setVideoEncoderRotation	Set the direction of image output by video encoder
setVideoEncoderMirror	Set the mirror mode of image output by encoder
setGSensorMode	Set the adaptation mode of G-sensor

setListener

setListener

void setListener	(TRTCCloudListener listener)

Set TRTC event callback

@deprecated This API is not recommended after v11.4 Please use addListener instead.

setBeautyStyle

setBeautyStyle

void setBeautyStyle	(int beautyStyle
	int beautyLevel
	int whitenessLevel
	int ruddinessLevel)

Set the strength of beauty, brightening, and rosy skin filters.

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setEyeScaleLevel

setEyeScaleLevel

void setEyeScaleLevel	(int eyeScaleLevel)

Set the strength of eye enlarging filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setFaceSlimLevel

setFaceSlimLevel

void setFaceSlimLevel	(int faceScaleLevel)

Set the strength of face slimming filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setFaceVLevel

setFaceVLevel

void setFaceVLevel	(int faceVLevel)

Set the strength of chin slimming filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setChinLevel

setChinLevel

void setChinLevel	(int chinLevel)

Set the strength of chin lengthening/shortening filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setFaceShortLevel

setFaceShortLevel

void setFaceShortLevel	(int faceShortlevel)

Set the strength of face shortening filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setNoseSlimLevel

setNoseSlimLevel

void setNoseSlimLevel	(int noseSlimLevel)

Set the strength of nose slimming filter

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

selectMotionTmpl

selectMotionTmpl

void selectMotionTmpl	(String motionPath)

Set animated sticker

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setMotionMute

setMotionMute

void setMotionMute	(boolean motionMute)

Mute animated sticker

@deprecated This API is not recommended after v6.9. Please use getBeautyManager instead.

setFilter

setFilter

void setFilter	(Bitmap image)

Set color filter

@deprecated This API is not recommended after v7.2. Please use getBeautyManager instead.

setFilterConcentration

setFilterConcentration

void setFilterConcentration	(float concentration)

Set the strength of color filter

@deprecated This API is not recommended after v7.2. Please use getBeautyManager instead.

setGreenScreenFile

setGreenScreenFile

boolean setGreenScreenFile	(String file)

Set green screen video

@deprecated This API is not recommended after v7.2. Please use getBeautyManager instead.

setReverbType

setReverbType

void setReverbType	(int reverbType)

Set reverb effect

@deprecated This API is not recommended after v7.3. Please use setVoiceReverbType API in TXAudioEffectManager instead.

setVoiceChangerType

setVoiceChangerType

boolean setVoiceChangerType	(int voiceChangerType)

Set voice changing type

@deprecated This API is not recommended after v7.3. Please use setVoiceChangerType API in TXAudioEffectManager instead.

enableAudioEarMonitoring

enableAudioEarMonitoring

void enableAudioEarMonitoring	(boolean enable)

Enable or disable in-ear monitoring

@deprecated This API is not recommended after v7.3. Please use setVoiceEarMonitor API in TXAudioEffectManager instead.

enableAudioVolumeEvaluation

enableAudioVolumeEvaluation

void enableAudioVolumeEvaluation	(int interval)

Enable volume reminder

@deprecated This API is not recommended after v10.1. Please use enableAudioVolumeEvaluation(enable, params) instead.

enableAudioVolumeEvaluation

enableAudioVolumeEvaluation

void enableAudioVolumeEvaluation	(int interval
	boolean enable_vad)

Enable volume reminder

@deprecated This API is not recommended after v11.2. Please use enableAudioVolumeEvaluation(enable, params) instead.

switchCamera

switchCamera

Switch camera

@deprecated This API is not recommended after v8.0. Please use the switchCamera API in TXDeviceManager instead.

isCameraZoomSupported

isCameraZoomSupported

Query whether the current camera supports zoom

@deprecated This API is not recommended after v8.0. Please use the isCameraZoomSupported API in TXDeviceManager instead.

setZoom

setZoom

void setZoom	(int distance)

Set camera zoom ratio (focal length)

@deprecated This API is not recommended after v8.0. Please use the setCameraZoomRatio API in TXDeviceManager instead.

isCameraTorchSupported

isCameraTorchSupported

Query whether the device supports flash

@deprecated This API is not recommended after v8.0. Please use the isCameraTorchSupported API in TXDeviceManager instead.

enableTorch

enableTorch

boolean enableTorch	(boolean enable)

Enable/Disable flash

@deprecated This API is not recommended after v8.0. Please use the enableCameraTorch API in TXDeviceManager instead.

isCameraFocusPositionInPreviewSupported

isCameraFocusPositionInPreviewSupported

Query whether the camera supports setting focus

@deprecated This API is not recommended after v8.0.

setFocusPosition

setFocusPosition

void setFocusPosition	(int x
	int y)

Set the focal position of camera

@deprecated This API is not recommended after v8.0. Please use the setCameraFocusPosition API in TXDeviceManager instead.

isCameraAutoFocusFaceModeSupported

isCameraAutoFocusFaceModeSupported

Query whether the device supports the automatic recognition of face position

@deprecated This API is not recommended after v8.0. Please use the isAutoFocusEnabled API in TXDeviceManager instead.

setSystemVolumeType

setSystemVolumeType

void setSystemVolumeType	(int type)

Setting the system volume type (for mobile OS)

@deprecated This API is not recommended after v8.0. Please use the startLocalAudio instead, which param `quality` is used to decide audio quality.

checkAudioCapabilitySupport

checkAudioCapabilitySupport

int checkAudioCapabilitySupport	(int capabilityType)

Query whether a certain audio capability is supported (only for Android)

@deprecated This API is not recommended after v10.1

Param	DESC
capabilityType	Audio capability type. TRTCAudioCapabilityLowLatencyChorus, Low-latency chorus capability. TRTCAudioCapabilityLowLatencyEarMonitor, Low-latency earmonitor capability.

Return Desc:

0：supported；1：supported。

startLocalAudio

startLocalAudio

Set sound quality

@deprecated This API is not recommended after v8.0. Please use startLocalAudio:quality instead.

startRemoteView

startRemoteView

void startRemoteView	(String userId

	TXCloudVideoView view)

Start displaying remote video image

@deprecated This API is not recommended after v8.0. Please use startRemoteView:streamType:view: instead.

stopRemoteView

stopRemoteView

void stopRemoteView	(String userId)

Stop displaying remote video image and pulling the video data stream of remote user

@deprecated This API is not recommended after v8.0. Please use stopRemoteView:streamType: instead.

setLocalViewFillMode

setLocalViewFillMode

void setLocalViewFillMode	(int mode)

Set the rendering mode of local image

@deprecated This API is not recommended after v8.0. Please use setLocalRenderParams instead.

setLocalViewRotation

setLocalViewRotation

void setLocalViewRotation	(int rotation)

Set the clockwise rotation angle of local image

@deprecated This API is not recommended after v8.0. Please use setLocalRenderParams instead.

setLocalViewMirror

setLocalViewMirror

void setLocalViewMirror	(int mirrorType)

Set the mirror mode of local camera's preview image

@deprecated This API is not recommended after v8.0. Please use setLocalRenderParams instead.

setRemoteViewFillMode

setRemoteViewFillMode

void setRemoteViewFillMode	(String userId
	int mode)

Set the fill mode of substream image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

setRemoteViewRotation

setRemoteViewRotation

void setRemoteViewRotation	(String userId
	int rotation)

Set the clockwise rotation angle of remote image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

startRemoteSubStreamView

startRemoteSubStreamView

void startRemoteSubStreamView	(String userId
	TXCloudVideoView view)

Start displaying the substream image of remote user

@deprecated This API is not recommended after v8.0. Please use startRemoteView:streamType:view: instead.

stopRemoteSubStreamView

stopRemoteSubStreamView

void stopRemoteSubStreamView	(String userId)

Stop displaying the substream image of remote user

@deprecated This API is not recommended after v8.0. Please use stopRemoteView:streamType: instead.

setRemoteSubStreamViewFillMode

setRemoteSubStreamViewFillMode

void setRemoteSubStreamViewFillMode	(String userId
	int mode)

Set the fill mode of substream image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

setRemoteSubStreamViewRotation

setRemoteSubStreamViewRotation

void setRemoteSubStreamViewRotation	(final String userId
	final int rotation)

Set the clockwise rotation angle of substream image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

setAudioQuality

setAudioQuality

void setAudioQuality	(int quality)

Set sound quality

@deprecated This API is not recommended after v8.0. Please use startLocalAudio:quality instead.

setPriorRemoteVideoStreamType

setPriorRemoteVideoStreamType

int setPriorRemoteVideoStreamType	(int streamType)

Specify whether to view the big or small image

@deprecated This API is not recommended after v8.0. Please use startRemoteView:streamType:view: instead.

setMicVolumeOnMixing

setMicVolumeOnMixing

void setMicVolumeOnMixing	(int volume)

Set mic volume

@deprecated This API is not recommended after v6.9. Please use setAudioCaptureVolume instead.

playBGM

playBGM

void playBGM	(String path
	TRTCCloud.BGMNotify notify)

Start background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

stopBGM

stopBGM

Stop background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

pauseBGM

pauseBGM

Stop background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

resumeBGM

resumeBGM

Stop background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

getBGMDuration

getBGMDuration

int getBGMDuration	(String path)

Get the total length of background music in ms

@deprecated This API is not recommended after v7.3. Please use getMusicDurationInMS API in TXAudioEffectManager instead.

setBGMPosition

setBGMPosition

int setBGMPosition	(int pos)

Set background music playback progress

@deprecated This API is not recommended after v7.3. Please use seekMusicToPosInMS API in TXAudioEffectManager instead.

setBGMVolume

setBGMVolume

void setBGMVolume	(int volume)

Set background music volume

@deprecated This API is not recommended after v7.3. Please use setMusicVolume API in TXAudioEffectManager instead.

setBGMPlayoutVolume

setBGMPlayoutVolume

void setBGMPlayoutVolume	(int volume)

Set the local playback volume of background music

@deprecated This API is not recommended after v7.3. Please use setMusicPlayoutVolume API in TXAudioEffectManager instead.

setBGMPublishVolume

setBGMPublishVolume

void setBGMPublishVolume	(int volume)

Set the remote playback volume of background music

@deprecated This API is not recommended after v7.3. Please use setBGMPublishVolume API in TXAudioEffectManager instead.

playAudioEffect

playAudioEffect

void playAudioEffect	(TRTCCloudDef.TRTCAudioEffectParam effect)

Play sound effect

@deprecated This API is not recommended after v7.3. Please use startPlayMusic API in TXAudioEffectManager instead.

setAudioEffectVolume

setAudioEffectVolume

void setAudioEffectVolume	(int effectId
	int volume)

Set sound effect volume

@deprecated This API is not recommended after v7.3. Please use setMusicPublishVolume and setMusicPlayoutVolume API in TXAudioEffectManager instead.

stopAudioEffect

stopAudioEffect

void stopAudioEffect	(int effectId)

Stop sound effect

@deprecated This API is not recommended after v7.3. Please use stopPlayMusic API in TXAudioEffectManager instead.

stopAllAudioEffects

stopAllAudioEffects

Stop all sound effects

@deprecated This API is not recommended after v7.3. Please use stopPlayMusic API in TXAudioEffectManager instead.

setAllAudioEffectsVolume

setAllAudioEffectsVolume

void setAllAudioEffectsVolume	(int volume)

Set the volume of all sound effects

@deprecated This API is not recommended after v7.3. Please use setMusicPublishVolume and setMusicPlayoutVolume API in TXAudioEffectManager instead.

pauseAudioEffect

pauseAudioEffect

void pauseAudioEffect	(int effectId)

Pause sound effect

@deprecated This API is not recommended after v7.3. Please use pauseAudioEffect API in TXAudioEffectManager instead.

resumeAudioEffect

resumeAudioEffect

void resumeAudioEffect	(int effectId)

Pause sound effect

@deprecated This API is not recommended after v7.3. Please use resumePlayMusic API in TXAudioEffectManager instead.

enableCustomVideoCapture

enableCustomVideoCapture

void enableCustomVideoCapture	(boolean enable)

Enable custom video capturing mode

@deprecated This API is not recommended after v8.5. Please use enableCustomVideoCapture instead.

sendCustomVideoData

sendCustomVideoData

void sendCustomVideoData	(TRTCCloudDef.TRTCVideoFrame frame)

Deliver captured video data to SDK

@deprecated This API is not recommended after v8.5. Please use sendCustomVideoData instead.

muteLocalVideo

muteLocalVideo

void muteLocalVideo	(boolean mute)

Pause/Resume publishing local video stream

@deprecated This API is not recommended after v8.9. Please use muteLocalVideo (streamType, mute) instead.

muteRemoteVideoStream

muteRemoteVideoStream

void muteRemoteVideoStream	(String userId
	boolean mute)

Pause/Resume subscribing to remote user's video stream

@deprecated This API is not recommended after v8.9. Please use muteRemoteVideoStream (userId, streamType, mute) instead.

snapshotVideo

snapshotVideo

void snapshotVideo	(String userId

	int streamType
	TRTCCloudListener.TRTCSnapshotListener listener)

Screencapture video

@deprecated This API is not recommended after v11.0. Please use snapshotVideo(userId, streamType, sourceType, listener) instead.

startSpeedTest

startSpeedTest

void startSpeedTest	(int sdkAppId
	String userId
	String userSig)

Start network speed test (used before room entry)

@deprecated This API is not recommended after v9.2. Please use startSpeedTest (params) instead.

startScreenCapture

startScreenCapture

void startScreenCapture	(TRTCCloudDef.TRTCVideoEncParam encParams
	TRTCCloudDef.TRTCScreenShareParams shareParams)

Start screen sharing

@deprecated This API is not recommended after v7.2. Please use `startScreenCapture:streamType:encParam:` instead.

setVideoEncoderRotation

setVideoEncoderRotation

void setVideoEncoderRotation	(int rotation)

Set the direction of image output by video encoder

@deprecated It is deprecated starting from v11.7.

setVideoEncoderMirror

setVideoEncoderMirror

void setVideoEncoderMirror	(boolean mirror)

Set the mirror mode of image output by encoder

@deprecated It is deprecated starting from v11.7.

setGSensorMode

setGSensorMode

void setGSensorMode	(int mode)

Set the adaptation mode of G-sensor

@deprecated It is deprecated starting from v11.7. It is recommended to use the setGravitySensorAdaptiveMode interface instead.

Error Codes

Last updated：2024-06-06 15:50:05

Module: TRTC ErrorCode

Function: Used to notify customers of warnings and errors that occur during the use of TRTC

ErrorCode

EnumType

EnumType	DESC
TXLiteAVError	Error Codes
TXLiteAVWarning	Warning codes

TXLiteAVError

TXLiteAVError

Error Codes

Enum	Value	DESC
ERR_NULL	0	No error.
ERR_FAILED	-1	Unclassified error.
ERR_INVALID_PARAMETER	-2	An invalid parameter was pas in when the API was called.
ERR_REFUSED	-3	The API call was rejected.
ERR_NOT_SUPPORTED	-4	The current API cannot be called.
ERR_INVALID_LICENSE	-5	Failed to call the API because

		the license is invalid.
ERR_REQUEST_SERVER_TIMEOUT	-6	The request timed out.
ERR_SERVER_PROCESS_FAILED	-7	The server cannot process yo request.
ERR_DISCONNECTED	-8	Disconnected from the server
ERR_CAMERA_START_FAIL	-1301	Failed to turn the camera on. This may occur when there is problem with the camera configuration program (driver) Windows or macOS. Disable reenable the camera, restart t camera, or update the configuration program.
ERR_CAMERA_NOT_AUTHORIZED	-1314	No permission to access to th camera. This usually occurs c mobile devices and may be because the user denied acce
ERR_CAMERA_SET_PARAM_FAIL	-1315	Incorrect camera parameter settings (unsupported values others).
ERR_CAMERA_OCCUPY	-1316	The camera is being used. Tr another camera.
ERR_SCREEN_CAPTURE_START_FAIL	-1308	Failed to start screen recordir If this occurs on a mobile devi it may be because the user denied screen sharing permission; if it occurs on Windows or macOS, check whether the parameters of the screen recording API are set required.
ERR_SCREEN_CAPTURE_UNSURPORT	-1309	Screen recording failed. Scre recording is only supported or Android versions later than 5. and iOS versions later than 1
ERR_SCREEN_CAPTURE_STOPPED	-7001	Screen recording was stoppe by the system.

ERR_SCREEN_SHARE_NOT_AUTHORIZED	-102015	No permission to publish the substream.
ERR_SCREEN_SHRAE_OCCUPIED_BY_OTHER	-102016	Another user is publishing the substream.
ERR_VIDEO_ENCODE_FAIL	-1303	Failed to encode video frames This may occur when a user c iOS switches to another app, which may cause the system release the hardware encoder When the user switches back this error may be thrown befor the hardware encoder is restarted.
ERR_UNSUPPORTED_RESOLUTION	-1305	Unsupported video resolution
ERR_PIXEL_FORMAT_UNSUPPORTED	-1327	Custom video capturing: Unsupported pixel format.
ERR_BUFFER_TYPE_UNSUPPORTED	-1328	Custom video capturing: Unsupported buffer type.
ERR_NO_AVAILABLE_HEVC_DECODERS	-2304	No available HEVC decoder found.
ERR_MIC_START_FAIL	-1302	Failed to turn the mic on. This may occur when there is a problem with the mic configuration program (driver) Windows or macOS. Disable reenable the mic, restart the n or update the configuration program.
ERR_MIC_NOT_AUTHORIZED	-1317	No permission to access to th mic. This usually occurs on mobile devices and may be because the user denied acce
ERR_MIC_SET_PARAM_FAIL	-1318	Failed to set mic parameters.
ERR_MIC_OCCUPY	-1319	The mic is being used. The m cannot be turned on when, for example, the user is having a on the mobile device.

ERR_MIC_STOP_FAIL	-1320	Failed to turn the mic off.
ERR_SPEAKER_START_FAIL	-1321	Failed to turn the speaker on. This may occur when there is problem with the speaker configuration program (driver) Windows or macOS. Disable reenable the speaker, restart speaker, or update the configuration program.
ERR_SPEAKER_SET_PARAM_FAIL	-1322	Failed to set speaker parameters.
ERR_SPEAKER_STOP_FAIL	-1323	Failed to turn the speaker off.
ERR_AUDIO_PLUGIN_START_FAIL	-1330	Failed to record computer auc which may be because the au driver is unavailable.
ERR_AUDIO_PLUGIN_INSTALL_NOT_AUTHORIZED	-1331	No permission to install the au driver.
ERR_AUDIO_PLUGIN_INSTALL_FAILED	-1332	Failed to install the audio drive
ERR_AUDIO_PLUGIN_INSTALLED_BUT_NEED_RESTART	-1333	The virtual sound card is installed successfully, but due the restrictions of macOS, you cannot use it right after installation. Ask users to resta the app upon receiving this er code.
ERR_AUDIO_ENCODE_FAIL	-1304	Failed to encode audio frames This may occur if the SDK cou not process the custom audio data passed in.
ERR_UNSUPPORTED_SAMPLERATE	-1306	Unsupported audio sample ra
ERR_TRTC_ENTER_ROOM_FAILED	-3301	Failed to enter the room. For t reason, refer to the error message for -3301 in `onError` .
ERR_TRTC_REQUEST_IP_TIMEOUT	-3307	IP and signature request time out. Check your network

		connection and whether your firewall allows UDP. Try visiting the IP address 162.14.22.165:8000 or 162.14.6.105:8000 and the domain default-query.trtc.tencent-cloud.com:8000.
ERR_TRTC_CONNECT_SERVER_TIMEOUT	-3308	Room entry request timed out Check your network connecti and whether VPN is used. Yo can also switch to 4G to run a test.
ERR_TRTC_ROOM_PARAM_NULL	-3316	Empty room entry parameters Please check whether valid parameters were passed in to the `enterRoom:appScen` API.
ERR_TRTC_INVALID_SDK_APPID	-3317	Incorrect room entry paramete Check whether `TRTCParams.sdkAppId` empty.
ERR_TRTC_INVALID_ROOM_ID	-3318	Incorrect room entry paramete Check whether `TRTCParams.roomId` or `TRTCParams.strRoomId` empty. Note that you cannot s both parameters.
ERR_TRTC_INVALID_USER_ID	-3319	Incorrect room entry paramete Check whether `TRTCParams.userId` is empty.
ERR_TRTC_INVALID_USER_SIG	-3320	Incorrect room entry paramete Check whether `TRTCParams.userSig` is empty.
ERR_TRTC_ENTER_ROOM_REFUSED	-3340	Request to enter room denied Check whether you called

		`enterRoom` twice to enter same room.
ERR_TRTC_INVALID_PRIVATE_MAPKEY	-100006	Advanced permission control enabled but failed to verify `TRTCParams.privateMapI` . For details, see Enabling Advanced Permission Contro
ERR_TRTC_SERVICE_SUSPENDED	-100013	The service is unavailable. Check if you have used up yo package or whether your Tencent Cloud account has overdue payments.
ERR_TRTC_USER_SIG_CHECK_FAILED	-100018	Failed to verify `UserSig` Check whether `TRTCParams.userSig` is correct or valid. For details, see UserSig Generation and Verification.
ERR_TRTC_PUSH_THIRD_PARTY_CLOUD_TIMEOUT	-3321	The relay to CDN request time out
ERR_TRTC_MIX_TRANSCODING_TIMEOUT	-3322	The On-Cloud MixTranscodin request timed out.
ERR_TRTC_PUSH_THIRD_PARTY_CLOUD_FAILED	-3323	Abnormal response packets f relay.
ERR_TRTC_MIX_TRANSCODING_FAILED	-3324	Abnormal response packet fo On-Cloud MixTranscoding.
ERR_TRTC_START_PUBLISHING_TIMEOUT	-3333	Signaling for publishing to the Tencent Cloud CDN timed ou
ERR_TRTC_START_PUBLISHING_FAILED	-3334	Signaling for publishing to the Tencent Cloud CDN was abnormal.
ERR_TRTC_STOP_PUBLISHING_TIMEOUT	-3335	Signaling for stopping publish to the Tencent Cloud CDN tim out.
ERR_TRTC_STOP_PUBLISHING_FAILED	-3336	Signaling for stopping publish

		to the Tencent Cloud CDN wa abnormal.
ERR_TRTC_CONNECT_OTHER_ROOM_TIMEOUT	-3326	The co-anchoring request tim out.
ERR_TRTC_DISCONNECT_OTHER_ROOM_TIMEOUT	-3327	The request to stop co-ancho timed out.
ERR_TRTC_CONNECT_OTHER_ROOM_INVALID_PARAMETER	-3328	Invalid parameter.
ERR_TRTC_CONNECT_OTHER_ROOM_AS_AUDIENCE	-3330	The current user is an audien member and cannot request o stop cross-room communicati Please call `switchRole` to switch to an anchor first.
ERR_BGM_OPEN_FAILED	-4001	Failed to open the file, such as invalid data found when processing input, ffmpeg proto not found, etc.
ERR_BGM_DECODE_FAILED	-4002	Audio file decoding failed.
ERR_BGM_OVER_LIMIT	-4003	The number exceeds the limit such as preloading two background music at the sam time.
ERR_BGM_INVALID_OPERATION	-4004	Invalid operation, such as call a preload function after startin playback.
ERR_BGM_INVALID_PATH	-4005	Invalid path, Please check whether the path you passed points to a legal music file.
ERR_BGM_INVALID_URL	-4006	Invalid URL, Please use a browser to check whether the URL address you passed in c download the desired music fi
ERR_BGM_NO_AUDIO_STREAM	-4007	No audio stream, Please conf whether the file you passed is legal audio file and whether th file is damaged.
ERR_BGM_FORMAT_NOT_SUPPORTED	-4008	Unsupported format, Please

		confirm whether the file forma you passed is a supported file format. The mobile version supports [mp3, aac, m4a, wa ogg, mp4, mkv], and the desk version supports [mp3, aac, m4a, wav, mp4, mkv].

TXLiteAVWarning

TXLiteAVWarning

Warning codes

Enum	Value	DESC
WARNING_HW_ENCODER_START_FAIL	1103	Failed to start the hardware encoder. Switched to software encoding.
WARNING_CURRENT_ENCODE_TYPE_CHANGED	1104	The codec changed. The additional field `type` in `onWarning` indicates the codec currently in use. `0` indicates H.264, and `1` indicates H.265. The additional field `hardware` in `onWarning` indicates the encoder type currently in use. `0` indicates software encoder, and `1` indicates hardware encoder. The additional field `stream` in `onWarning` indicates the stream

		type currently in use. `0` indicates big stream, and `1` indicates small stream, and `2` indicates sub stream.
WARNING_VIDEO_ENCODER_SW_TO_HW	1107	Insufficient CPU for software encoding. Switched to hardware encoding.
WARNING_INSUFFICIENT_CAPTURE_FPS	1108	The capturing frame rate of the camera is insufficient. This error occurs on some Android phones with built-in beauty filters.
WARNING_SW_ENCODER_START_FAIL	1109	Failed to start the software encoder.
WARNING_REDUCE_CAPTURE_RESOLUTION	1110	The capturing frame rate of the camera was reduced for balance between frame rate and performance.
WARNING_CAMERA_DEVICE_EMPTY	1111	No available camera found.
WARNING_CAMERA_NOT_AUTHORIZED	1112	The user didn’t grant the application camera permission.
WARNING_OUT_OF_MEMORY	1113	Some functions may not work properly due to out of memory.
WARNING_CAMERA_IS_OCCUPIED	1114	The camera is occupied.
WARNING_CAMERA_DEVICE_ERROR	1115	The camera device is error.

WARNING_CAMERA_DISCONNECTED	1116	The camera is disconnected.
WARNING_CAMERA_START_FAILED	1117	The camera is started failed.
WARNING_CAMERA_SERVER_DIED	1118	The camera sever is died.
WARNING_SCREEN_CAPTURE_NOT_AUTHORIZED	1206	The user didn’t grant the application screen recording permission.
WARNING_CURRENT_DECODE_TYPE_CHANGED	2008	The codec changed. The additional field `type` in `onWarning` indicates the codec currently in use. `1` indicates H.265, and `0` indicates H.264. This field is not supported on Windows.
WARNING_VIDEO_FRAME_DECODE_FAIL	2101	Failed to decode the current video frame.
WARNING_HW_DECODER_START_FAIL	2106	Failed to start the hardware decoder. The software decoder is used instead.
WARNING_VIDEO_DECODER_HW_TO_SW	2108	The hardware decoder failed to decode the first I-frame of the current stream. The SDK automatically switched to the software decoder.
WARNING_SW_DECODER_START_FAIL	2109	Failed to start the software decoder.

WARNING_VIDEO_RENDER_FAIL	2110	Failed to render the video.
WARNING_VIRTUAL_BACKGROUND_DEVICE_UNSURPORTED	8001	The device does not support virtual background
WARNING_VIRTUAL_BACKGROUND_NOT_AUTHORIZED	8002	Virtual background not authorized
WARNING_VIRTUAL_BACKGROUND_INVALID_PARAMETER	8003	Enable virtual background with invalid parameter
WARNING_VIRTUAL_BACKGROUND_PERFORMANCE_INSUFFICIENT	8004	Virtual background performance insufficient
WARNING_MICROPHONE_DEVICE_EMPTY	1201	No available mic found.
WARNING_SPEAKER_DEVICE_EMPTY	1202	No available speaker found.
WARNING_MICROPHONE_NOT_AUTHORIZED	1203	The user didn’t grant the application mic permission.
WARNING_MICROPHONE_DEVICE_ABNORMAL	1204	The audio capturing device is unavailable (which may be because the device is used by another application or is considered invalid by the system).
WARNING_SPEAKER_DEVICE_ABNORMAL	1205	The audio playback device is unavailable (which may be because the device is used by another application or is considered invalid by the system).
WARNING_BLUETOOTH_DEVICE_CONNECT_FAIL	1207	The bluetooth device

		failed to connect (which may be because another app is occupying the audio channel by setting communication mode).
WARNING_MICROPHONE_IS_OCCUPIED	1208	The audio capturing device is occupied.
WARNING_AUDIO_FRAME_DECODE_FAIL	2102	Failed to decode the current audio frame.
WARNING_AUDIO_RECORDING_WRITE_FAIL	7001	Failed to write recorded audio into the file.
WARNING_MICROPHONE_HOWLING_DETECTED	7002	Detect capture audio howling
WARNING_IGNORE_UPSTREAM_FOR_AUDIENCE	6001	The current user is an audience member and cannot publish audio or video. Please switch to an anchor first.
WARNING_UPSTREAM_AUDIO_AND_VIDEO_OUT_OF_SYNC	6006	The audio or video sending timestamps are abnormal, which may cause audio and video synchronization issues.

All Platforms (C++)

Overview

Last updated：2024-06-06 15:26:15

API OVERVIEW

Create Instance And Event Callback

FuncList	DESC
getTRTCShareInstance	Create TRTCCloud instance (singleton mode)
destroyTRTCShareInstance	Terminate TRTCCloud instance (singleton mode)
addCallback	Add TRTC event callback
removeCallback	Remove TRTC event callback

Room APIs

FuncList	DESC
enterRoom	Enter room
exitRoom	Exit room
switchRole	Switch role
switchRoom	Switch room
connectOtherRoom	Request cross-room call
disconnectOtherRoom	Exit cross-room call
setDefaultStreamRecvMode	Set subscription mode (which must be set before room entry for it to take effect)
createSubCloud	Create room subinstance (for concurrent multi-room listen/watch)
destroySubCloud	Terminate room subinstance

updateOtherRoomForwardMode	

CDN APIs

FuncList	DESC
startPublishing	Start publishing audio/video streams to Tencent Cloud CSS CDN
stopPublishing	Stop publishing audio/video streams to Tencent Cloud CSS CDN
startPublishCDNStream	Start publishing audio/video streams to non-Tencent Cloud CDN
stopPublishCDNStream	Stop publishing audio/video streams to non-Tencent Cloud CDN
setMixTranscodingConfig	Set the layout and transcoding parameters of On-Cloud MixTranscoding
startPublishMediaStream	Publish a stream
updatePublishMediaStream	Modify publishing parameters
stopPublishMediaStream	Stop publishing

Video APIs

FuncList	DESC
startLocalPreview	Enable the preview image of local camera (mobile)
updateLocalView	Update the preview image of local camera
stopLocalPreview	Stop camera preview
muteLocalVideo	Pause/Resume publishing local video stream
setVideoMuteImage	Set placeholder image during local video pause
startRemoteView	Subscribe to remote user's video stream and bind video rendering control
updateRemoteView	Update remote user's video rendering control
stopRemoteView	Stop subscribing to remote user's video stream and release rendering control

stopAllRemoteView	Stop subscribing to all remote users' video streams and release all rendering resources
muteRemoteVideoStream	Pause/Resume subscribing to remote user's video stream
muteAllRemoteVideoStreams	Pause/Resume subscribing to all remote users' video streams
setVideoEncoderParam	Set the encoding parameters of video encoder
setNetworkQosParam	Set network quality control parameters
setLocalRenderParams	Set the rendering parameters of local video image
setRemoteRenderParams	Set the rendering mode of remote video image
enableSmallVideoStream	Enable dual-channel encoding mode with big and small images
setRemoteVideoStreamType	Switch the big/small image of specified remote user
snapshotVideo	Screencapture video
setGravitySensorAdaptiveMode	Set the adaptation mode of gravity sensing (version 11.7 and above)

Audio APIs

FuncList	DESC
startLocalAudio	Enable local audio capturing and publishing
stopLocalAudio	Stop local audio capturing and publishing
muteLocalAudio	Pause/Resume publishing local audio stream
muteRemoteAudio	Pause/Resume playing back remote audio stream
muteAllRemoteAudio	Pause/Resume playing back all remote users' audio streams
setRemoteAudioVolume	Set the audio playback volume of remote user
setAudioCaptureVolume	Set the capturing volume of local audio
getAudioCaptureVolume	Get the capturing volume of local audio
setAudioPlayoutVolume	Set the playback volume of remote audio
getAudioPlayoutVolume	Get the playback volume of remote audio

enableAudioVolumeEvaluation	Enable volume reminder
startAudioRecording	Start audio recording
stopAudioRecording	Stop audio recording
startLocalRecording	Start local media recording
stopLocalRecording	Stop local media recording
setRemoteAudioParallelParams	Set the parallel strategy of remote audio streams
enable3DSpatialAudioEffect	Enable 3D spatial effect
updateSelf3DSpatialPosition	Update self position and orientation for 3D spatial effect
updateRemote3DSpatialPosition	Update the specified remote user's position for 3D spatial effect
set3DSpatialReceivingRange	Set the maximum 3D spatial attenuation range for userId's audio stream

Device management APIs

FuncList	DESC
*getDeviceManager	Get device management class (TXDeviceManager)

Beauty filter and watermark APIs

FuncList	DESC
setBeautyStyle	Set special effects such as beauty, brightening, and rosy skin filters
setWaterMark	Add watermark

Background music and sound effect APIs

FuncList	DESC
getAudioEffectManager	Get sound effect management class (TXAudioEffectManager)

startSystemAudioLoopback	Enable system audio capturing(iOS not supported)
stopSystemAudioLoopback	Stop system audio capturing(iOS not supported)
setSystemAudioLoopbackVolume	Set the volume of system audio capturing

Screen sharing APIs

FuncList	DESC
startScreenCapture	Start screen sharing
stopScreenCapture	Stop screen sharing
pauseScreenCapture	Pause screen sharing
resumeScreenCapture	Resume screen sharing
getScreenCaptureSources	Enumerate shareable screens and windows (for desktop systems only)
selectScreenCaptureTarget	Select the screen or window to share (for desktop systems only)
setSubStreamEncoderParam	Set the video encoding parameters of screen sharing (i.e., substream) (for desktop and mobile systems)
setSubStreamMixVolume	Set the audio mixing volume of screen sharing (for desktop systems only)
addExcludedShareWindow	Add specified windows to the exclusion list of screen sharing (for desktop systems only)
removeExcludedShareWindow	Remove specified windows from the exclusion list of screen sharing (for desktop systems only)
removeAllExcludedShareWindow	Remove all windows from the exclusion list of screen sharing (for desktop systems only)
addIncludedShareWindow	Add specified windows to the inclusion list of screen sharing (for desktop systems only)
removeIncludedShareWindow	Remove specified windows from the inclusion list of screen sharing (for desktop systems only)
removeAllIncludedShareWindow	Remove all windows from the inclusion list of screen sharing (for desktop systems only)

Custom capturing and rendering APIs

FuncList	DESC
enableCustomVideoCapture	Enable/Disable custom video capturing mode
sendCustomVideoData	Deliver captured video frames to SDK
enableCustomAudioCapture	Enable custom audio capturing mode
sendCustomAudioData	Deliver captured audio data to SDK
enableMixExternalAudioFrame	Enable/Disable custom audio track
mixExternalAudioFrame	Mix custom audio track into SDK
setMixExternalAudioVolume	Set the publish volume and playback volume of mixed custom audio track
generateCustomPTS	Generate custom capturing timestamp
enableLocalVideoCustomProcess	.1 Enable third-party beauty filters in video
setLocalVideoCustomProcessCallback	.2 Set video data callback for third-party beauty filters
setLocalVideoRenderCallback	Set the callback of custom rendering for local video
setRemoteVideoRenderCallback	Set the callback of custom rendering for remote video
setAudioFrameCallback	Set custom audio data callback
setCapturedAudioFrameCallbackFormat	Set the callback format of audio frames captured by local mic
setLocalProcessedAudioFrameCallbackFormat	Set the callback format of preprocessed local audio frames
setMixedPlayAudioFrameCallbackFormat	Set the callback format of audio frames to be played back by system
enableCustomAudioRendering	Enabling custom audio playback
getCustomAudioRenderingFrame	Getting playable audio data

Custom message sending APIs

FuncList	DESC
sendCustomCmdMsg	Use UDP channel to send custom message to all users in room
sendSEIMsg	Use SEI channel to send custom message to all users in room

Network test APIs

FuncList	DESC
startSpeedTest	Start network speed test (used before room entry)
stopSpeedTest	Stop network speed test

Debugging APIs

FuncList	DESC
getSDKVersion	Get SDK version information
setLogLevel	Set log output level
setConsoleEnabled	Enable/Disable console log printing
setLogCompressEnabled	Enable/Disable local log compression
setLogDirPath	Set local log storage path
setLogCallback	Set log callback
showDebugView	Display dashboard
callExperimentalAPI	Call experimental APIs

Encrypted interface

FuncList	DESC
enablePayloadPrivateEncryption	Enable or disable private encryption of media streams

Error and warning events

FuncList	DESC
onError	Error event callback
onWarning	Warning event callback

Room event callback

FuncList	DESC
onEnterRoom	Whether room entry is successful
onExitRoom	Room exit
onSwitchRole	Role switching
onSwitchRoom	Result of room switching
onConnectOtherRoom	Result of requesting cross-room call
onDisconnectOtherRoom	Result of ending cross-room call
onUpdateOtherRoomForwardMode	Result of changing the upstream capability of the cross-room anchor

User event callback

FuncList	DESC
onRemoteUserEnterRoom	A user entered the room
onRemoteUserLeaveRoom	A user exited the room
onUserVideoAvailable	A remote user published/unpublished primary stream video
onUserSubStreamAvailable	A remote user published/unpublished substream video
onUserAudioAvailable	A remote user published/unpublished audio
onFirstVideoFrame	The SDK started rendering the first video frame of the local or a remote user

onFirstAudioFrame	The SDK started playing the first audio frame of a remote user
onSendFirstLocalVideoFrame	The first local video frame was published
onSendFirstLocalAudioFrame	The first local audio frame was published
onRemoteVideoStatusUpdated	Change of remote video status
onRemoteAudioStatusUpdated	Change of remote audio status
onUserVideoSizeChanged	Change of remote video size

Callback of statistics on network and technical metrics

FuncList	DESC
onNetworkQuality	Real-time network quality statistics
onStatistics	Real-time statistics on technical metrics
onSpeedTestResult	Callback of network speed test

Callback of connection to the cloud

FuncList	DESC
onConnectionLost	The SDK was disconnected from the cloud
onTryToReconnect	The SDK is reconnecting to the cloud
onConnectionRecovery	The SDK is reconnected to the cloud

Callback of hardware events

FuncList	DESC
onCameraDidReady	The camera is ready
onMicDidReady	The mic is ready
onUserVoiceVolume	Volume

onDeviceChange	The status of a local device changed (for desktop OS only)
onAudioDeviceCaptureVolumeChanged	The capturing volume of the mic changed
onAudioDevicePlayoutVolumeChanged	The playback volume changed
onSystemAudioLoopbackError	Whether system audio capturing is enabled successfully (for macOS only)
onTestMicVolume	Volume during mic test
onTestSpeakerVolume	Volume during speaker test

Callback of the receipt of a custom message

FuncList	DESC
onRecvCustomCmdMsg	Receipt of custom message
onMissCustomCmdMsg	Loss of custom message
onRecvSEIMsg	Receipt of SEI message

CDN event callback

FuncList	DESC
onStartPublishing	Started publishing to Tencent Cloud CSS CDN
onStopPublishing	Stopped publishing to Tencent Cloud CSS CDN
onStartPublishCDNStream	Started publishing to non-Tencent Cloud’s live streaming CDN
onStopPublishCDNStream	Stopped publishing to non-Tencent Cloud’s live streaming CDN
onSetMixTranscodingConfig	Set the layout and transcoding parameters for On-Cloud MixTranscoding
onStartPublishMediaStream	Callback for starting to publish
onUpdatePublishMediaStream	Callback for modifying publishing parameters
onStopPublishMediaStream	Callback for stopping publishing

onCdnStreamStateChanged	Callback for change of RTMP/RTMPS publishing status

Screen sharing event callback

FuncList	DESC
onScreenCaptureStarted	Screen sharing started
onScreenCapturePaused	Screen sharing was paused
onScreenCaptureResumed	Screen sharing was resumed
onScreenCaptureStoped	Screen sharing stopped
onScreenCaptureCovered	The shared window was covered (for Windows only)

Callback of local recording and screenshot events

FuncList	DESC
onLocalRecordBegin	Local recording started
onLocalRecording	Local media is being recorded
onLocalRecordFragment	Record fragment finished.
onLocalRecordComplete	Local recording stopped
onSnapshotComplete	Finished taking a local screenshot

Disused callbacks

FuncList	DESC
onUserEnter	An anchor entered the room (disused)
onUserExit	An anchor left the room (disused)
onAudioEffectFinished	Audio effects ended (disused)
onPlayBGMBegin	Started playing background music (disused)

onPlayBGMProgress	Playback progress of background music (disused)
onPlayBGMComplete	Background music stopped (disused)
onSpeedTest	Result of server speed testing (disused)

Callback of custom video processing

FuncList	DESC
onRenderVideoFrame	Custom video rendering
onGLContextCreated	An OpenGL context was created in the SDK.
onProcessVideoFrame	Video processing by third-party beauty filters
onGLContextDestroy	The OpenGL context in the SDK was destroyed

Callback of custom audio processing

FuncList	DESC
onCapturedAudioFrame	Audio data captured by the local mic and pre-processed by the audio module
onLocalProcessedAudioFrame	Audio data captured by the local mic, pre-processed by the audio module, effect-processed and BGM-mixed
onPlayAudioFrame	Audio data of each remote user before audio mixing
onMixedPlayAudioFrame	Data mixed from each channel before being submitted to the system for playback
onMixedAllAudioFrame	Data mixed from all the captured and to-be-played audio in the SDK

Other event callbacks

FuncList	DESC
onLog	Printing of local log

Background music preload event callback

FuncList	DESC
onLoadProgress	Background music preload progress
onLoadError	Background music preload error

Callback of playing background music

FuncList	DESC
onStart	Background music started.
onPlayProgress	Playback progress of background music
onComplete	Background music ended

Voice effect APIs

FuncList	DESC
enableVoiceEarMonitor	Enabling in-ear monitoring
setVoiceEarMonitorVolume	Setting in-ear monitoring volume
setVoiceReverbType	Setting voice reverb effects
setVoiceChangerType	Setting voice changing effects
setVoiceCaptureVolume	Setting speech volume
setVoicePitch	Setting speech pitch

Background music APIs

FuncList	DESC
setMusicObserver	Setting the background music callback

startPlayMusic	Starting background music
stopPlayMusic	Stopping background music
pausePlayMusic	Pausing background music
resumePlayMusic	Resuming background music
setAllMusicVolume	Setting the local and remote playback volume of background music
setMusicPublishVolume	Setting the remote playback volume of a specific music track
setMusicPlayoutVolume	Setting the local playback volume of a specific music track
setMusicPitch	Adjusting the pitch of background music
setMusicSpeedRate	Changing the speed of background music
getMusicCurrentPosInMS	Getting the playback progress (ms) of background music
getMusicDurationInMS	Getting the total length (ms) of background music
seekMusicToPosInTime	Setting the playback progress (ms) of background music
setMusicScratchSpeedRate	Adjust the speed change effect of the scratch disc
setPreloadObserver	Setting music preload callback
preloadMusic	Preload background music
getMusicTrackCount	Get the number of tracks of background music
setMusicTrack	Specify the playback track of background music

Device APIs

FuncList	DESC
isFrontCamera	Querying whether the front camera is being used
switchCamera	Switching to the front/rear camera (for mobile OS)
getCameraZoomMaxRatio	Getting the maximum zoom ratio of the camera (for mobile OS)
setCameraZoomRatio	Setting the camera zoom ratio (for mobile OS)
isAutoFocusEnabled	Querying whether automatic face detection is supported (for

	mobile OS)
enableCameraAutoFocus	Enabling auto focus (for mobile OS)
setCameraFocusPosition	Adjusting the focus (for mobile OS)
enableCameraTorch	Enabling/Disabling flash, i.e., the torch mode (for mobile OS)
setAudioRoute	Setting the audio route (for mobile OS)
getDevicesList	Getting the device list (for desktop OS)
setCurrentDevice	Setting the device to use (for desktop OS)
getCurrentDevice	Getting the device currently in use (for desktop OS)
setCurrentDeviceVolume	Setting the volume of the current device (for desktop OS)
getCurrentDeviceVolume	Getting the volume of the current device (for desktop OS)
setCurrentDeviceMute	Muting the current device (for desktop OS)
getCurrentDeviceMute	Querying whether the current device is muted (for desktop OS)
enableFollowingDefaultAudioDevice	Set the audio device used by SDK to follow the system default device (for desktop OS)
startCameraDeviceTest	Starting camera testing (for desktop OS)
stopCameraDeviceTest	Ending camera testing (for desktop OS)
startMicDeviceTest	Starting mic testing (for desktop OS)
stopMicDeviceTest	Ending mic testing (for desktop OS)
startSpeakerDeviceTest	Starting speaker testing (for desktop OS)
stopSpeakerDeviceTest	Ending speaker testing (for desktop OS)
setApplicationPlayVolume	Setting the volume of the current process in the volume mixer (for Windows)
getApplicationPlayVolume	Getting the volume of the current process in the volume mixer (for Windows)
setApplicationMuteState	Muting the current process in the volume mixer (for Windows)
getApplicationMuteState	Querying whether the current process is muted in the volume mixer (for Windows)

setCameraCapturerParam	Set camera acquisition preferences
setDeviceObserver	set onDeviceChanged callback

Disused APIs

FuncList	DESC
setSystemVolumeType	Setting the system volume type (for mobile OS)

Disused APIs

FuncList	DESC
enableAudioVolumeEvaluation	Enable volume reminder
startLocalAudio	Set sound quality
startRemoteView	Start displaying remote video image
stopRemoteView	Stop displaying remote video image and pulling the video data stream of remote user
setLocalViewFillMode	Set the rendering mode of local image
setLocalViewRotation	Set the clockwise rotation angle of local image
setLocalViewMirror	Set the mirror mode of local camera's preview image
setRemoteViewFillMode	Set the fill mode of substream image
setRemoteViewRotation	Set the clockwise rotation angle of remote image
startRemoteSubStreamView	Start displaying the substream image of remote user
stopRemoteSubStreamView	Stop displaying the substream image of remote user
setRemoteSubStreamViewFillMode	Set the fill mode of substream image
setRemoteSubStreamViewRotation	Set the clockwise rotation angle of substream image
setAudioQuality	Set sound quality
setPriorRemoteVideoStreamType	Specify whether to view the big or small image

setMicVolumeOnMixing	Set mic volume
playBGM	Start background music
stopBGM	Stop background music
pauseBGM	Stop background music
resumeBGM	Stop background music
getBGMDuration	Get the total length of background music in ms
setBGMPosition	Set background music playback progress
setBGMVolume	Set background music volume
setBGMPlayoutVolume	Set the local playback volume of background music
setBGMPublishVolume	Set the remote playback volume of background music
playAudioEffect	Play sound effect
setAudioEffectVolume	Set sound effect volume
stopAudioEffect	Stop sound effect
stopAllAudioEffects	Stop all sound effects
setAllAudioEffectsVolume	Set the volume of all sound effects
pauseAudioEffect	Pause sound effect
resumeAudioEffect	Pause sound effect
enableCustomVideoCapture	Enable custom video capturing mode
sendCustomVideoData	Deliver captured video data to SDK
muteLocalVideo	Pause/Resume publishing local video stream
muteRemoteVideoStream	Pause/Resume subscribing to remote user's video stream
startSpeedTest	Start network speed test (used before room entry)
startScreenCapture	Start screen sharing
setLocalVideoProcessCallback	Set video data callback for third-party beauty filters
getCameraDevicesList	Get the list of cameras

setCurrentCameraDevice	Set the camera to be used currently
getCurrentCameraDevice	Get the currently used camera
getMicDevicesList	Get the list of mics
getCurrentMicDevice	Get the current mic device
setCurrentMicDevice	Select the currently used mic
getCurrentMicDeviceVolume	Get the current mic volume
setCurrentMicDeviceVolume	Set the current mic volume
setCurrentMicDeviceMute	Set the mute status of the current system mic
getCurrentMicDeviceMute	Get the mute status of the current system mic
getSpeakerDevicesList	Get the list of speakers
getCurrentSpeakerDevice	Get the currently used speaker
setCurrentSpeakerDevice	Set the speaker to use
getCurrentSpeakerVolume	Get the current speaker volume
setCurrentSpeakerVolume	Set the current speaker volume
getCurrentSpeakerDeviceMute	Get the mute status of the current system speaker
setCurrentSpeakerDeviceMute	Set whether to mute the current system speaker
startCameraDeviceTest	Start camera test
stopCameraDeviceTest	Start camera test
startMicDeviceTest	Start mic test
stopMicDeviceTest	Start mic test
startSpeakerDeviceTest	Start speaker test
stopSpeakerDeviceTest	Stop speaker test
selectScreenCaptureTarget	start in-app screen sharing (for iOS 13.0 and above only)
setVideoEncoderRotation	Set the direction of image output by video encoder
setVideoEncoderMirror	Set the mirror mode of image output by encoder

ITRTCCloud

Last updated：2024-06-06 15:26:15

Module: TRTCCloud @ TXLiteAVSDK

Function: TRTC's main feature API

Version: 11.9

ITRTCCloud

ITRTCCloud

FuncList	DESC
getTRTCShareInstance	Create TRTCCloud instance (singleton mode)
destroyTRTCShareInstance	Terminate TRTCCloud instance (singleton mode)
addCallback	Add TRTC event callback
removeCallback	Remove TRTC event callback
enterRoom	Enter room
exitRoom	Exit room
switchRole	Switch role
switchRole	Switch role(support permission credential)
switchRoom	Switch room
connectOtherRoom	Request cross-room call
disconnectOtherRoom	Exit cross-room call
setDefaultStreamRecvMode	Set subscription mode (which must be set before room entry for it to take effect)

createSubCloud	Create room subinstance (for concurrent multi-room listen/watch)
destroySubCloud	Terminate room subinstance
updateOtherRoomForwardMode	
startPublishing	Start publishing audio/video streams to Tencent Cloud CSS CDN
stopPublishing	Stop publishing audio/video streams to Tencent Cloud CSS CDN
startPublishCDNStream	Start publishing audio/video streams to non-Tencent Cloud CDN
stopPublishCDNStream	Stop publishing audio/video streams to non-Tencent Cloud CDN
setMixTranscodingConfig	Set the layout and transcoding parameters of On-Cloud MixTranscoding
startPublishMediaStream	Publish a stream
updatePublishMediaStream	Modify publishing parameters
stopPublishMediaStream	Stop publishing
startLocalPreview	Enable the preview image of local camera (mobile)
startLocalPreview	Enable the preview image of local camera (desktop)
updateLocalView	Update the preview image of local camera
stopLocalPreview	Stop camera preview
muteLocalVideo	Pause/Resume publishing local video stream
setVideoMuteImage	Set placeholder image during local video pause
startRemoteView	Subscribe to remote user's video stream and bind video rendering control
updateRemoteView	Update remote user's video rendering control
stopRemoteView	Stop subscribing to remote user's video stream and release rendering control
stopAllRemoteView	Stop subscribing to all remote users' video streams

	and release all rendering resources
muteRemoteVideoStream	Pause/Resume subscribing to remote user's video stream
muteAllRemoteVideoStreams	Pause/Resume subscribing to all remote users' video streams
setVideoEncoderParam	Set the encoding parameters of video encoder
setNetworkQosParam	Set network quality control parameters
setLocalRenderParams	Set the rendering parameters of local video image
setRemoteRenderParams	Set the rendering mode of remote video image
enableSmallVideoStream	Enable dual-channel encoding mode with big and small images
setRemoteVideoStreamType	Switch the big/small image of specified remote user
snapshotVideo	Screencapture video
setGravitySensorAdaptiveMode	Set the adaptation mode of gravity sensing (version 11.7 and above)
startLocalAudio	Enable local audio capturing and publishing
stopLocalAudio	Stop local audio capturing and publishing
muteLocalAudio	Pause/Resume publishing local audio stream
muteRemoteAudio	Pause/Resume playing back remote audio stream
muteAllRemoteAudio	Pause/Resume playing back all remote users' audio streams
setRemoteAudioVolume	Set the audio playback volume of remote user
setAudioCaptureVolume	Set the capturing volume of local audio
getAudioCaptureVolume	Get the capturing volume of local audio
setAudioPlayoutVolume	Set the playback volume of remote audio
getAudioPlayoutVolume	Get the playback volume of remote audio
enableAudioVolumeEvaluation	Enable volume reminder
startAudioRecording	Start audio recording

stopAudioRecording	Stop audio recording
startLocalRecording	Start local media recording
stopLocalRecording	Stop local media recording
setRemoteAudioParallelParams	Set the parallel strategy of remote audio streams
enable3DSpatialAudioEffect	Enable 3D spatial effect
updateSelf3DSpatialPosition	Update self position and orientation for 3D spatial effect
updateRemote3DSpatialPosition	Update the specified remote user's position for 3D spatial effect
set3DSpatialReceivingRange	Set the maximum 3D spatial attenuation range for userId's audio stream
*getDeviceManager	Get device management class (TXDeviceManager)
setBeautyStyle	Set special effects such as beauty, brightening, and rosy skin filters
setWaterMark	Add watermark
getAudioEffectManager	Get sound effect management class (TXAudioEffectManager)
startSystemAudioLoopback	Enable system audio capturing(iOS not supported)
stopSystemAudioLoopback	Stop system audio capturing(iOS not supported)
setSystemAudioLoopbackVolume	Set the volume of system audio capturing
startScreenCapture	Start screen sharing
stopScreenCapture	Stop screen sharing
pauseScreenCapture	Pause screen sharing
resumeScreenCapture	Resume screen sharing
getScreenCaptureSources	Enumerate shareable screens and windows (for desktop systems only)
selectScreenCaptureTarget	Select the screen or window to share (for desktop systems only)

setSubStreamEncoderParam	Set the video encoding parameters of screen sharing (i.e., substream) (for desktop and mobile systems)
setSubStreamMixVolume	Set the audio mixing volume of screen sharing (for desktop systems only)
addExcludedShareWindow	Add specified windows to the exclusion list of screen sharing (for desktop systems only)
removeExcludedShareWindow	Remove specified windows from the exclusion list of screen sharing (for desktop systems only)
removeAllExcludedShareWindow	Remove all windows from the exclusion list of screen sharing (for desktop systems only)
addIncludedShareWindow	Add specified windows to the inclusion list of screen sharing (for desktop systems only)
removeIncludedShareWindow	Remove specified windows from the inclusion list of screen sharing (for desktop systems only)
removeAllIncludedShareWindow	Remove all windows from the inclusion list of screen sharing (for desktop systems only)
enableCustomVideoCapture	Enable/Disable custom video capturing mode
sendCustomVideoData	Deliver captured video frames to SDK
enableCustomAudioCapture	Enable custom audio capturing mode
sendCustomAudioData	Deliver captured audio data to SDK
enableMixExternalAudioFrame	Enable/Disable custom audio track
mixExternalAudioFrame	Mix custom audio track into SDK
setMixExternalAudioVolume	Set the publish volume and playback volume of mixed custom audio track
generateCustomPTS	Generate custom capturing timestamp
enableLocalVideoCustomProcess	.1 Enable third-party beauty filters in video
setLocalVideoCustomProcessCallback	.2 Set video data callback for third-party beauty filters
setLocalVideoRenderCallback	Set the callback of custom rendering for local video
setRemoteVideoRenderCallback	Set the callback of custom rendering for remote video

setAudioFrameCallback	Set custom audio data callback
setCapturedAudioFrameCallbackFormat	Set the callback format of audio frames captured by local mic
setLocalProcessedAudioFrameCallbackFormat	Set the callback format of preprocessed local audio frames
setMixedPlayAudioFrameCallbackFormat	Set the callback format of audio frames to be played back by system
enableCustomAudioRendering	Enabling custom audio playback
getCustomAudioRenderingFrame	Getting playable audio data
sendCustomCmdMsg	Use UDP channel to send custom message to all users in room
sendSEIMsg	Use SEI channel to send custom message to all users in room
startSpeedTest	Start network speed test (used before room entry)
stopSpeedTest	Stop network speed test
getSDKVersion	Get SDK version information
setLogLevel	Set log output level
setConsoleEnabled	Enable/Disable console log printing
setLogCompressEnabled	Enable/Disable local log compression
setLogDirPath	Set local log storage path
setLogCallback	Set log callback
showDebugView	Display dashboard
callExperimentalAPI	Call experimental APIs
enablePayloadPrivateEncryption	Enable or disable private encryption of media streams

getTRTCShareInstance

getTRTCShareInstance

ITRTCCloud* getTRTCShareInstance	(void *context)

Create TRTCCloud instance (singleton mode)

Param	DESC
context	It is only applicable to the Android platform. The SDK internally converts it into the `ApplicationContext` of Android to call the Android system API.

Note

1. If you use `delete ITRTCCloud*`, a compilation error will occur. Please use `destroyTRTCCloud` to release the object pointer.
2. On Windows, macOS, or iOS, please call the `getTRTCShareInstance()` API.
3. On Android, please call the `getTRTCShareInstance(void *context)` API.

destroyTRTCShareInstance

destroyTRTCShareInstance

Terminate TRTCCloud instance (singleton mode)

addCallback

addCallback

void addCallback	(ITRTCCloudCallback* callback)

Add TRTC event callback

You can use ITRTCCloudCallback to get various event notifications from the SDK, such as error codes, warning codes, and audio/video status parameters.

removeCallback

removeCallback

void removeCallback	(ITRTCCloudCallback* callback)

Remove TRTC event callback

enterRoom

enterRoom

void enterRoom	(const TRTCParams& param
	TRTCAppScene scene)

Enter room

All TRTC users need to enter a room before they can "publish" or "subscribe to" audio/video streams. "Publishing" refers to pushing their own streams to the cloud, and "subscribing to" refers to pulling the streams of other users in the room from the cloud.

When calling this API, you need to specify your application scenario (TRTCAppScene) to get the best audio/video transfer experience. We provide the following four scenarios for your choice:
TRTCAppSceneVideoCall:
Video call scenario. Use cases: [one-to-one video call], [video conferencing with up to 300 participants], [online medical diagnosis], [small class], [video interview], etc.
In this scenario, each room supports up to 300 concurrent online users, and up to 50 of them can speak simultaneously.
TRTCAppSceneAudioCall:
Audio call scenario. Use cases: [one-to-one audio call], [audio conferencing with up to 300 participants], [audio chat], [online Werewolf], etc.
In this scenario, each room supports up to 300 concurrent online users, and up to 50 of them can speak simultaneously.
TRTCAppSceneLIVE:
Live streaming scenario. Use cases: [low-latency video live streaming], [interactive classroom for up to 100,000 participants], [live video competition], [video dating room], [remote training], [large-scale conferencing], etc.
In this scenario, each room supports up to 100,000 concurrent online users, but you should specify the user roles: anchor (TRTCRoleAnchor) or audience (TRTCRoleAudience).
TRTCAppSceneVoiceChatRoom:
Audio chat room scenario. Use cases: [Clubhouse], [online karaoke room], [music live room], [FM radio], etc.
In this scenario, each room supports up to 100,000 concurrent online users, but you should specify the user roles: anchor (TRTCRoleAnchor) or audience (TRTCRoleAudience).

After calling this API, you will receive the `onEnterRoom(result)` callback from ITRTCCloudCallback:
If room entry succeeded, the `result` parameter will be a positive number (`result` > 0), indicating the time in milliseconds (ms) between function call and room entry.

If room entry failed, the `result` parameter will be a negative number (`result` < 0), indicating the TXLiteAVError for room entry failure.

Param	DESC
param	Room entry parameter, which is used to specify the user's identity, role, authentication credentials, and other information. For more information, please see TRTCParams.
scene	Application scenario, which is used to specify the use case. The same TRTCAppScene should be configured for all users in the same room.

Note

1. If `scene` is specified as TRTCAppSceneLIVE or TRTCAppSceneVoiceChatRoom, you must use the `role` field in TRTCParams to specify the role of the current user in the room.
2. The same `scene` should be configured for all users in the same room.
3. Please try to ensure that enterRoom and exitRoom are used in pair; that is, please make sure that "the previous room is exited before the next room is entered"; otherwise, many issues may occur.

exitRoom

exitRoom

Exit room

Calling this API will allow the user to leave the current audio or video room and release the camera, mic, speaker, and other device resources.
After resources are released, the SDK will use the `onExitRoom()` callback in ITRTCCloudCallback to notify you.

If you need to call enterRoom again or switch to the SDK of another provider, we recommend you wait until you receive the `onExitRoom()` callback, so as to avoid the problem of the camera or mic being occupied.

switchRole

switchRole

void switchRole	(TRTCRoleType role)

Switch role

This API is used to switch the user role between `anchor` and `audience` .

As video live rooms and audio chat rooms need to support an audience of up to 100,000 concurrent online users, the rule "only anchors can publish their audio/video streams" has been set. Therefore, when some users want to publish their streams (so that they can interact with anchors), they need to switch their role to "anchor" first.

You can use the `role` field in TRTCParams during room entry to specify the user role in advance or use the `switchRole` API to switch roles after room entry.

Param	DESC
role	Role, which is `anchor` by default: TRTCRoleAnchor: anchor, who can publish their audio/video streams. Up to 50 anchors are allowed to publish streams at the same time in one room. TRTCRoleAudience: audience, who cannot publish their audio/video streams, but can only watch streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room supports an audience of up to 100,000 concurrent online users.

Note
1. This API is only applicable to two scenarios: live streaming (TRTCAppSceneLIVE) and audio chat room (TRTCAppSceneVoiceChatRoom).
2. If the `scene` you specify in enterRoom is TRTCAppSceneVideoCall or TRTCAppSceneAudioCall, please do not call this API.

switchRole

switchRole

void switchRole	(TRTCRoleType role
	const char* privateMapKey)

Switch role(support permission credential)

This API is used to switch the user role between `anchor` and `audience` .

As video live rooms and audio chat rooms need to support an audience of up to 100,000 concurrent online users, the rule "only anchors can publish their audio/video streams" has been set. Therefore, when some users want to publish their streams (so that they can interact with anchors), they need to switch their role to "anchor" first.

You can use the `role` field in TRTCParams during room entry to specify the user role in advance or use the `switchRole` API to switch roles after room entry.

Param	DESC
privateMapKey	Permission credential used for permission control. If you want only users with the specified `userId` values to enter a room or push streams, you need to use `privateMapKey` to restrict the permission. We recommend you use this parameter only if you have high security requirements. For more information, please see Enabling Advanced Permission Control.
role	Role, which is `anchor` by default: TRTCRoleAnchor: anchor, who can publish their audio/video streams. Up to 50 anchors are allowed to publish streams at the same time in one room. TRTCRoleAudience: audience, who cannot publish their audio/video streams, but can only watch streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room supports an audience of up to 100,000 concurrent online users.

Note

1. This API is only applicable to two scenarios: live streaming (TRTCAppSceneLIVE) and audio chat room (TRTCAppSceneVoiceChatRoom).
2. If the `scene` you specify in enterRoom is TRTCAppSceneVideoCall or TRTCAppSceneAudioCall, please do not call this API.

switchRoom

switchRoom

void switchRoom	(const TRTCSwitchRoomConfig& config)

Switch room

This API is used to quickly switch a user from one room to another.
If the user's role is `audience`, calling this API is equivalent to `exitRoom` (current room) + `enterRoom` (new room).
If the user's role is `anchor`, the API will retain the current audio/video publishing status while switching the room; therefore, during the room switch, camera preview and sound capturing will not be interrupted.

This API is suitable for the online education scenario where the supervising teacher can perform fast room switch across multiple rooms. In this scenario, using `switchRoom` can get better smoothness and use less code than `exitRoom + enterRoom` .

The API call result will be called back through `onSwitchRoom(errCode, errMsg)` in ITRTCCloudCallback.

Param	DESC
config	Room parameter. For more information, please see TRTCSwitchRoomConfig.

Note

Due to the requirement for compatibility with legacy versions of the SDK, the `config` parameter contains both `roomId` and `strRoomId` parameters. You should pay special attention as detailed below when specifying these two parameters:

1. If you decide to use `strRoomId` , then set `roomId` to 0. If both are specified, `roomId` will be used.
2. All rooms need to use either `strRoomId` or `roomId` at the same time. They cannot be mixed; otherwise, there will be many unexpected bugs.

connectOtherRoom

connectOtherRoom

void connectOtherRoom	(const char* param)

Request cross-room call

By default, only users in the same room can make audio/video calls with each other, and the audio/video streams in different rooms are isolated from each other.

However, you can publish the audio/video streams of an anchor in another room to the current room by calling this API. At the same time, this API will also publish the local audio/video streams to the target anchor's room.

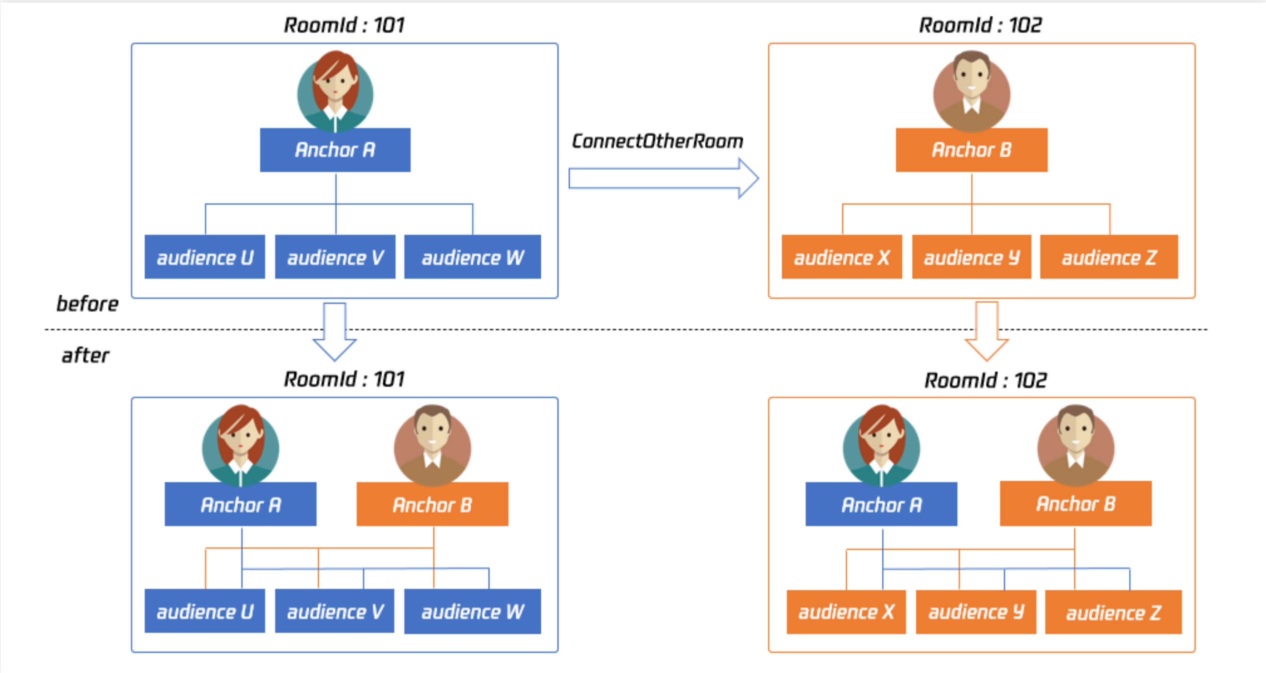
In other words, you can use this API to share the audio/video streams of two anchors in two different rooms, so that the audience in each room can watch the streams of these two anchors. This feature can be used to implement anchor competition.

The result of requesting cross-room call will be returned through the onConnectOtherRoom callback in TRTCCloudDelegate.

For example, after anchor A in room "101" uses `connectOtherRoom()` to successfully call anchor B in room "102":

All users in room "101" will receive the `onRemoteUserEnterRoom(B)` and `onUserVideoAvailable(B,true)` event callbacks of anchor B; that is, all users in room "101" can subscribe to the audio/video streams of anchor B.

All users in room "102" will receive the `onRemoteUserEnterRoom(A)` and `onUserVideoAvailable(A,true)` event callbacks of anchor A; that is, all users in room "102" can subscribe to the audio/video streams of anchor A.



For compatibility with subsequent extended fields for cross-room call, parameters in JSON format are used currently.

Case 1: numeric room ID

If anchor A in room "101" wants to co-anchor with anchor B in room "102", then anchor A needs to pass in {"roomId": 102, "userId": "userB"} when calling this API.

Below is the sample code:


```
Json::Value jsonObj;
jsonObj["roomId"] = 102;
jsonObj["userId"] = "userB";
Json::FastWriter writer;
std::string params = writer.write(jsonObj);
trtc.ConnectOtherRoom(params.c_str());
```

Case 2: string room ID

If you use a string room ID, please be sure to replace the `roomId` in JSON with `strRoomId` , such as {"strRoomId": "102", "userId": "userB"}

Below is the sample code:

```
Json::Value jsonObj;
jsonObj["strRoomId"] = "102";
jsonObj["userId"] = "userB";
Json::FastWriter writer;
std::string params = writer.write(jsonObj);
trtc.ConnectOtherRoom(params.c_str());
```


Param	DESC
param	You need to pass in a string parameter in JSON format: `roomId` represents the room ID in numeric format, `strRoomId` represents the room ID in string format, and `userId` represents the user ID of the target anchor.

disconnectOtherRoom

disconnectOtherRoom

Exit cross-room call

The result will be returned through the `onDisconnectOtherRoom()` callback in TRTCCloudDelegate.

setDefaultStreamRecvMode

setDefaultStreamRecvMode

void setDefaultStreamRecvMode	(bool autoRecvAudio
	bool autoRecvVideo)

Set subscription mode (which must be set before room entry for it to take effect)

You can switch between the "automatic subscription" and "manual subscription" modes through this API:
Automatic subscription: this is the default mode, where the user will immediately receive the audio/video streams in the room after room entry, so that the audio will be automatically played back, and the video will be automatically decoded (you still need to bind the rendering control through the `startRemoteView` API).
Manual subscription: after room entry, the user needs to manually call the startRemoteView API to start subscribing to and decoding the video stream and call the muteRemoteAudio (false) API to start playing back the audio stream.

In most scenarios, users will subscribe to the audio/video streams of all anchors in the room after room entry. Therefore, TRTC adopts the automatic subscription mode by default in order to achieve the best "instant streaming experience".
In your application scenario, if there are many audio/video streams being published at the same time in each room, and each user only wants to subscribe to 1–2 streams of them, we recommend you use the "manual subscription" mode to reduce the traffic costs.

Param	DESC
autoRecvAudio	true: automatic subscription to audio; false: manual subscription to audio by calling

	`muteRemoteAudio(false)` . Default value: true
autoRecvVideo	true: automatic subscription to video; false: manual subscription to video by calling `startRemoteView` . Default value: true

Note

1. The configuration takes effect only if this API is called before room entry (enterRoom).
2. In the automatic subscription mode, if the user does not call startRemoteView to subscribe to the video stream after room entry, the SDK will automatically stop subscribing to the video stream in order to reduce the traffic consumption.

createSubCloud

createSubCloud

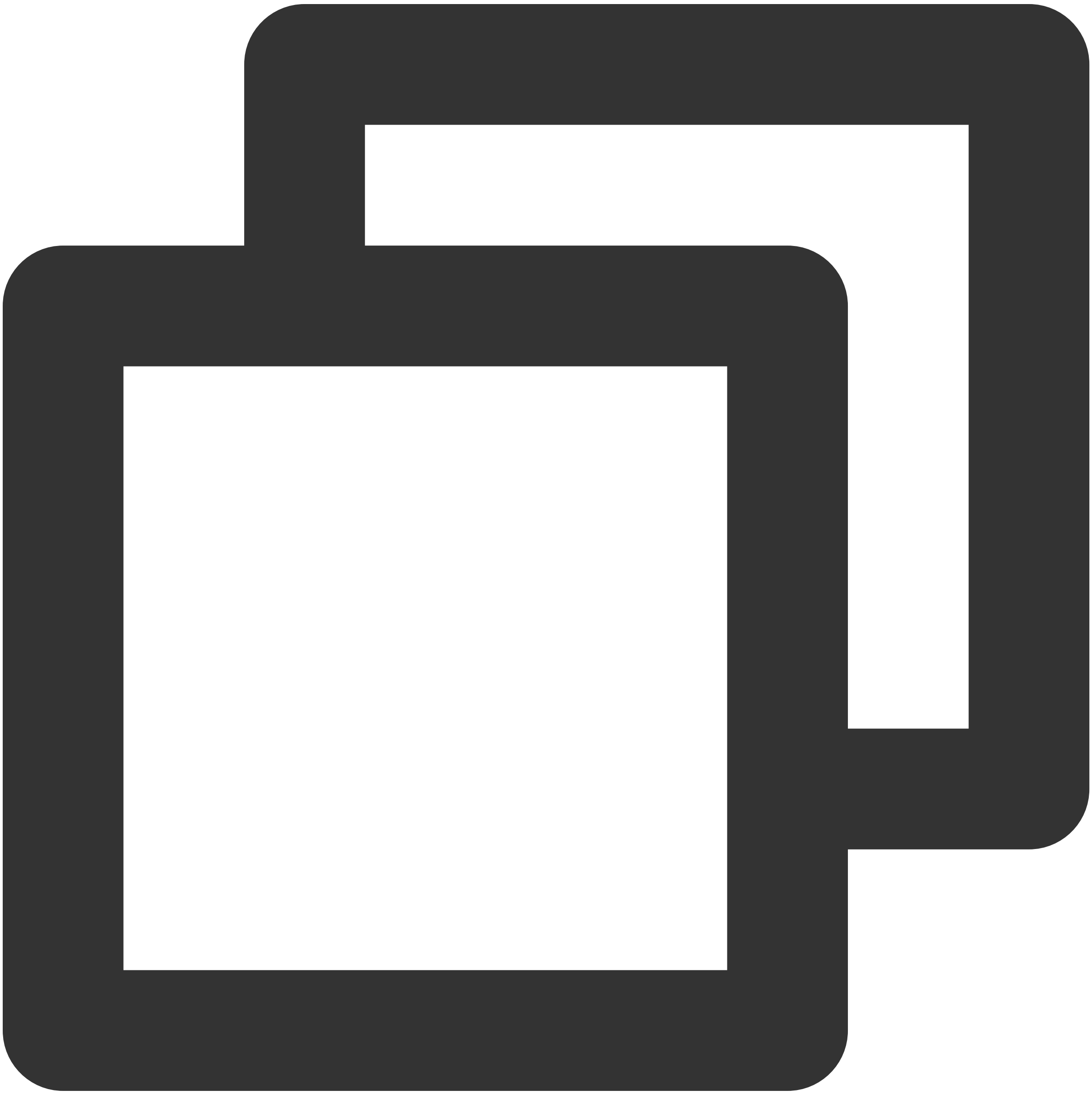
Create room subinstance (for concurrent multi-room listen/watch)

`TRTCCloud` was originally designed to work in the singleton mode, which limited the ability to watch concurrently in multiple rooms.
By calling this API, you can create multiple `TRTCCloud` instances, so that you can enter multiple different rooms at the same time to listen/watch audio/video streams.

However, it should be noted that your ability to publish audio and video streams in multiple `TRTCCloud` instances will be limited.

This feature is mainly used in the "super small class" use case in the online education scenario to break the limit that "only up to 50 users can publish their audio/video streams simultaneously in one TRTC room".

Below is the sample code:



```
//In the small room that needs interaction, enter the room as an anchor and pus
ITRTCCloud *mainCloud = getTRTCShareInstance();
TRTCParams mainParams;
//Fill your params
mainParams.role = TRTCRoleAnchor;
mainCloud->enterRoom(mainParams, TRTCAppSceneLIVE);
//...
mainCloud->startLocalAudio(TRTCAudioQualityDefault);
mainCloud->startLocalPreview(renderView);

//In the large room that only needs to watch, enter the room as an audience and
```

```
    ITRTCCloud *subCloud = mainCloud->createSubCloud();
    TRTCParams subParams;
    //Fill your params
    subParams.role = TRTCRoleAudience;
    subCloud->enterRoom(subParams, TRTCAppSceneLIVE);
    //...
    subCloud->startRemoteView(userId, TRTCVideoStreamTypeBig, renderView);
    //...
    //Exit from new room and release it.
    subCloud->exitRoom();
    mainCloud->destroySubCloud(subCloud);
```

Note

The same user can enter multiple rooms with different `roomId` values by using the same `userId`.

Two devices cannot use the same `userId` to enter the same room with a specified `roomId`.

You can set ITRTCCloudCallback separately for different instances to get their own event notifications.

The same user can push streams in multiple `TRTCCloud` instances at the same time, and can also call APIs related to local audio/video in the sub instance. But need to pay attention to:

Audio needs to be collected by the microphone or custom data at the same time in all instances, and the result of API calls related to the audio device will be based on the last time;

The result of camera-related API call will be based on the last time: startLocalPreview.

Return Desc:

`TRTCCloud` subinstance

destroySubCloud

destroySubCloud

void destroySubCloud	(ITRTCCloud *subCloud)

Terminate room subinstance

Param	DESC
subCloud	

startPublishing

startPublishing

void startPublishing	(const char* streamId
	TRTCVideoStreamType streamType)

Start publishing audio/video streams to Tencent Cloud CSS CDN

This API sends a command to the TRTC server, requesting it to relay the current user's audio/video streams to CSS CDN.
You can set the `StreamId` of the live stream through the `streamId` parameter, so as to specify the playback address of the user's audio/video streams on CSS CDN.

For example, if you specify the current user's live stream ID as `user_stream_001` through this API, then the corresponding CDN playback address is:
"http://yourdomain/live/user_stream_001.flv", where `yourdomain` is your playback domain name with an ICP filing.

You can configure your playback domain name in the CSS console. Tencent Cloud does not provide a default playback domain name.

You can also specify the `streamId` when setting the `TRTCParams` parameter of `enterRoom`, which is the recommended approach.

Param	DESC
streamId	Custom stream ID.
streamType	Only `TRTCVideoStreamTypeBig` and `TRTCVideoStreamTypeSub` are supported.

Note
You need to enable the "Enable Relayed Push" option on the "Function Configuration" page in the TRTC console in advance.
If you select "Specified stream for relayed push", you can use this API to push the corresponding audio/video stream to Tencent Cloud CDN and specify the entered stream ID.
If you select "Global auto-relayed push", you can use this API to adjust the default stream ID.

stopPublishing

stopPublishing

Stop publishing audio/video streams to Tencent Cloud CSS CDN

startPublishCDNStream

startPublishCDNStream

void startPublishCDNStream	(const TRTCPublishCDNParam& param)

Start publishing audio/video streams to non-Tencent Cloud CDN

This API is similar to the `startPublishing` API. The difference is that `startPublishing` can only publish audio/video streams to Tencent Cloud CDN, while this API can relay streams to live streaming CDN services of other cloud providers.

Param	DESC
param	CDN relaying parameter. For more information, please see TRTCPublishCDNParam

Note

Using the `startPublishing` API to publish audio/video streams to Tencent Cloud CSS CDN does not incur additional fees.

Using the `startPublishCDNStream` API to publish audio/video streams to non-Tencent Cloud CDN incurs additional relaying bandwidth fees.

stopPublishCDNStream

stopPublishCDNStream

Stop publishing audio/video streams to non-Tencent Cloud CDN

setMixTranscodingConfig

setMixTranscodingConfig

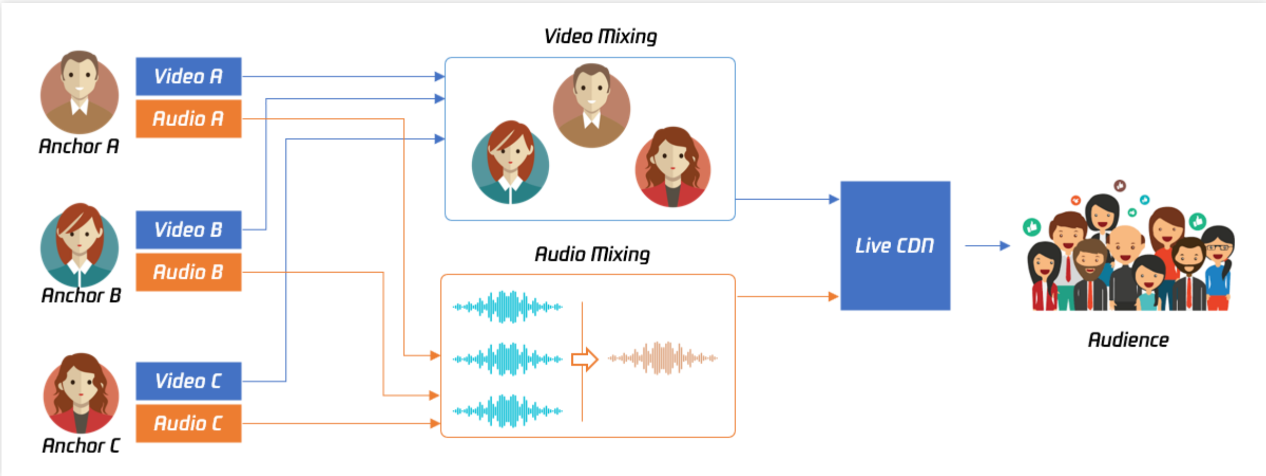
void setMixTranscodingConfig	(TRTCTranscodingConfig* config)

Set the layout and transcoding parameters of On-Cloud MixTranscoding

In a live room, there may be multiple anchors publishing their audio/video streams at the same time, but for audience on CSS CDN, they only need to watch one video stream in HTTP-FLV or HLS format.

When you call this API, the SDK will send a command to the TRTC mixtranscoding server to combine multiple audio/video streams in the room into one stream.
You can use the TRTCTranscodingConfig parameter to set the layout of each channel of image. You can also set the encoding parameters of the mixed audio/video streams.

For more information, please see On-Cloud MixTranscoding.



Param	DESC
config	If `config` is not empty, On-Cloud MixTranscoding will be started; otherwise, it will be stopped. For more information, please see TRTCTranscodingConfig.

Note

Notes on On-Cloud MixTranscoding:

Mixed-stream transcoding is a chargeable function, calling the interface will incur cloud-based mixed-stream transcoding fees, see Billing of On-Cloud MixTranscoding.

If the user calling this API does not set `streamId` in the `config` parameter, TRTC will mix the multiple channels of images in the room into the audio/video streams corresponding to the current user, i.e., A + B => A.

If the user calling this API sets `streamId` in the `config` parameter, TRTC will mix the multiple channels of images in the room into the specified `streamId`, i.e., A + B => streamId.

Please note that if you are still in the room but do not need mixtranscoding anymore, be sure to call this API again and leave `config` empty to cancel it; otherwise, additional fees may be incurred.

Please rest assured that TRTC will automatically cancel the mixtranscoding status upon room exit.

startPublishMediaStream

startPublishMediaStream

void startPublishMediaStream	(TRTCPublishTarget * target
	TRTCStreamEncoderParam * params
	TRTCStreamMixingConfig * config)

Publish a stream

After this API is called, the TRTC server will relay the stream of the local user to a CDN (after transcoding or without transcoding), or transcode and publish the stream to a TRTC room.

You can use the TRTCPublishMode parameter in TRTCPublishTarget to specify the publishing mode.

Param	DESC
config	The On-Cloud MixTranscoding settings. This parameter is invalid in the relay-to-CDN mode. It is required if you transcode and publish the stream to a CDN or to a TRTC room. For details, see TRTCStreamMixingConfig.
params	The encoding settings. This parameter is required if you transcode and publish the stream to a CDN or to a TRTC room. If you relay to a CDN without transcoding, to improve the relaying stability and playback compatibility, we also recommend you set this parameter. For details, see TRTCStreamEncoderParam.
target	The publishing destination. You can relay the stream to a CDN (after transcoding or without transcoding) or transcode and publish the stream to a TRTC room. For details, see TRTCPublishTarget.

Note

1. The SDK will send a task ID to you via the onStartPublishMediaStream callback.
2. You can start a publishing task only once and cannot initiate two tasks that use the same publishing mode and publishing cdn url. Note the task ID returned, which you need to pass to updatePublishMediaStream to modify the publishing parameters or stopPublishMediaStream to stop the task.
3. You can specify up to 10 CDN URLs in `target`. You will be charged only once for transcoding even if you relay to multiple CDNs.
4. To avoid causing errors, do not specify the same URLs for different publishing tasks executed at the same time. We recommend you add "sdkappid_roomid_userid_main" to URLs to distinguish them from one another and avoid application conflicts.

updatePublishMediaStream

updatePublishMediaStream

void updatePublishMediaStream	(const char* taskId
	TRTCPublishTarget * target
	TRTCStreamEncoderParam * params
	TRTCStreamMixingConfig * config)

Modify publishing parameters

You can use this API to change the parameters of a publishing task initiated by startPublishMediaStream.

Param	DESC
config	The On-Cloud MixTranscoding settings. This parameter is invalid in the relay-to-CDN mode. It is required if you transcode and publish the stream to a CDN or to a TRTC room. For details, see TRTCStreamMixingConfig.
params	The encoding settings. This parameter is required if you transcode and publish the stream to a CDN or to a TRTC room. If you relay to a CDN without transcoding, to improve the relaying stability and playback compatibility, we recommend you set this parameter. For details, see TRTCStreamEncoderParam.
target	The publishing destination. You can relay the stream to a CDN (after transcoding or without transcoding) or transcode and publish the stream to a TRTC room. For details, see TRTCPublishTarget.
taskId	The task ID returned to you via the onStartPublishMediaStream callback.

Note

1. You can use this API to add or remove CDN URLs to publish to (you can publish to up to 10 CDNs at a time). To avoid causing errors, do not specify the same URLs for different tasks executed at the same time.
2. You can use this API to switch a relaying task to transcoding or vice versa. For example, in cross-room communication, you can first call startPublishMediaStream to relay to a CDN. When the anchor requests cross-room communication, call this API, passing in the task ID to switch the relaying task to a transcoding task. This can ensure that the live stream and CDN playback are not interrupted (you need to keep the encoding parameters consistent).
3. You can not switch output between "only audio" 、 "only video" and "audio and video" for the same task.

stopPublishMediaStream

stopPublishMediaStream

void stopPublishMediaStream	(const char* taskId)

Stop publishing

You can use this API to stop a task initiated by startPublishMediaStream.

Param	DESC
taskId	The task ID returned to you via the onStartPublishMediaStream callback.

Note

1. If the task ID is not saved to your backend, you can call startPublishMediaStream again when an anchor re-enters the room after abnormal exit. The publishing will fail, but the TRTC backend will return the task ID to you.
2. If `taskId` is left empty, the TRTC backend will end all tasks you started through startPublishMediaStream. You can leave it empty if you have started only one task or want to stop all publishing tasks started by you.

startLocalPreview

startLocalPreview

void startLocalPreview	(bool frontCamera
	TXView view)

Enable the preview image of local camera (mobile)

If this API is called before `enterRoom` , the SDK will only enable the camera and wait until `enterRoom` is called before starting push.
If it is called after `enterRoom` , the SDK will enable the camera and automatically start pushing the video stream. When the first camera video frame starts to be rendered, you will receive the `onCameraDidReady` callback in ITRTCCloudCallback.

Param	DESC
frontCamera	true: front camera; false: rear camera
view	Control that carries the video image

Note

If you want to preview the camera image and adjust the beauty filter parameters through `BeautyManager` before going live, you can:

Scheme 1. Call `startLocalPreview` before calling `enterRoom`

Scheme 2. Call `startLocalPreview` and `muteLocalVideo(true)` after calling `enterRoom`

startLocalPreview

startLocalPreview

void startLocalPreview	(TXView view)

Enable the preview image of local camera (desktop)

Before this API is called, `setCurrentCameraDevice` can be called first to select whether to use the macOS device's built-in camera or an external camera.

If this API is called before `enterRoom`, the SDK will only enable the camera and wait until `enterRoom` is called before starting push.

If it is called after `enterRoom`, the SDK will enable the camera and automatically start pushing the video stream.

When the first camera video frame starts to be rendered, you will receive the `onCameraDidReady` callback in ITRTCCloudCallback.

Param	DESC
view	Control that carries the video image

Note

If you want to preview the camera image and adjust the beauty filter parameters through `BeautyManager` before going live, you can:

Scheme 1. Call `startLocalPreview` before calling `enterRoom`

Scheme 2. Call `startLocalPreview` and `muteLocalVideo(true)` after calling `enterRoom`

updateLocalView

updateLocalView

void updateLocalView	(TXView view)

Update the preview image of local camera

stopLocalPreview

stopLocalPreview

Stop camera preview

muteLocalVideo

muteLocalVideo

void muteLocalVideo	(TRTCVideoStreamType streamType
	bool mute)

Pause/Resume publishing local video stream

This API can pause (or resume) publishing the local video image. After the pause, other users in the same room will not be able to see the local image.
This API is equivalent to the two APIs of `startLocalPreview/stopLocalPreview` when TRTCVideoStreamTypeBig is specified, but has higher performance and response speed.
The `startLocalPreview/stopLocalPreview` APIs need to enable/disable the camera, which are hardware device-related operations, so they are very time-consuming.
In contrast, `muteLocalVideo` only needs to pause or allow the data stream at the software level, so it is more efficient and more suitable for scenarios where frequent enabling/disabling are needed.

After local video publishing is paused, other members in the same room will receive the `onUserVideoAvailable(userId, false)` callback notification.
After local video publishing is resumed, other members in the same room will receive the `onUserVideoAvailable(userId, true)` callback notification.

Param	DESC
mute	true: pause; false: resume
streamType	Specify for which video stream to pause (or resume). Only TRTCVideoStreamTypeBig and TRTCVideoStreamTypeSub are supported

setVideoMuteImage

setVideoMuteImage

void setVideoMuteImage	(TRTCImageBuffer* image
	int fps)

Set placeholder image during local video pause

When you call `muteLocalVideo(true)` to pause the local video image, you can set a placeholder image by calling this API. Then, other users in the room will see this image instead of a black screen.

Param	DESC
fps	Frame rate of the placeholder image. Minimum value: 5. Maximum value: 10. Default value: 5
image	Placeholder image. A null value means that no more video stream data will be sent after `muteLocalVideo`. The default value is null.

startRemoteView

startRemoteView

void startRemoteView	(const char* userId
	TRTCVideoStreamType streamType
	TXView view)

Subscribe to remote user's video stream and bind video rendering control

Calling this API allows the SDK to pull the video stream of the specified `userId` and render it to the rendering control specified by the `view` parameter. You can set the display mode of the video image through setRemoteRenderParams.

If you already know the `userId` of a user who has a video stream in the room, you can directly call `startRemoteView` to subscribe to the user's video image.

If you don't know which users in the room are publishing video streams, you can wait for the notification from onUserVideoAvailable after `enterRoom`.

Calling this API only starts pulling the video stream, and the image needs to be loaded and buffered at this time. After the buffering is completed, you will receive a notification from onFirstVideoFrame.

Param	DESC
streamType	Video stream type of the `userId` specified for watching: HD big image: TRTCVideoStreamTypeBig

	Smooth small image: TRTCVideoStreamTypeSmall (the remote user should enable dual-channel encoding through enableSmallVideoStream for this parameter to take effect) Substream image (usually used for screen sharing): TRTCVideoStreamTypeSub
userId	ID of the specified remote user
view	Rendering control that carries the video image

Note

The following requires your attention:

1. The SDK supports watching the big image and substream image or small image and substream image of a `userId` at the same time, but does not support watching the big image and small image at the same time.
2. Only when the specified `userId` enables dual-channel encoding through enableSmallVideoStream can the user's small image be viewed.
3. If the small image of the specified `userId` does not exist, the SDK will switch to the big image of the user by default.

updateRemoteView

updateRemoteView

void updateRemoteView	(const char* userId
	TRTCVideoStreamType streamType
	TXView view)

Update remote user's video rendering control

This API can be used to update the rendering control of the remote video image. It is often used in interactive scenarios where the display area needs to be switched.

Param	DESC
streamType	Type of the stream for which to set the preview window (only TRTCVideoStreamTypeBig and TRTCVideoStreamTypeSub are supported)
userId	ID of the specified remote user
view	Control that carries the video image

stopRemoteView

stopRemoteView

void stopRemoteView	(const char* userId
	TRTCVideoStreamType streamType)

Stop subscribing to remote user's video stream and release rendering control

Calling this API will cause the SDK to stop receiving the user's video stream and release the decoding and rendering resources for the stream.

Param	DESC
streamType	Video stream type of the `userId` specified for watching: HD big image: TRTCVideoStreamTypeBig Smooth small image: TRTCVideoStreamTypeSmall Substream image (usually used for screen sharing): TRTCVideoStreamTypeSub
userId	ID of the specified remote user

stopAllRemoteView

stopAllRemoteView

Stop subscribing to all remote users' video streams and release all rendering resources

Calling this API will cause the SDK to stop receiving all remote video streams and release all decoding and rendering resources.

Note

If a substream image (screen sharing) is being displayed, it will also be stopped.

muteRemoteVideoStream

muteRemoteVideoStream

void muteRemoteVideoStream	(const char* userId
	TRTCVideoStreamType streamType
	bool mute)

Pause/Resume subscribing to remote user's video stream

This API only pauses/resumes receiving the specified user's video stream but does not release displaying resources; therefore, the video image will freeze at the last frame before it is called.

Param	DESC
mute	Whether to pause receiving
streamType	Specify for which video stream to pause (or resume): HD big image: TRTCVideoStreamTypeBig Smooth small image: TRTCVideoStreamTypeSmall Substream image (usually used for screen sharing): TRTCVideoStreamTypeSub
userId	ID of the specified remote user

Note

This API can be called before room entry (enterRoom), and the pause status will be reset after room exit (exitRoom). After calling this API to pause receiving the video stream from a specific user, simply calling the startRemoteView API will not be able to play the video from that user. You need to call muteRemoteVideoStream(false) or muteAllRemoteVideoStreams(false) to resume it.

muteAllRemoteVideoStreams

muteAllRemoteVideoStreams

void muteAllRemoteVideoStreams	(bool mute)

Pause/Resume subscribing to all remote users' video streams

This API only pauses/resumes receiving all users' video streams but does not release displaying resources; therefore, the video image will freeze at the last frame before it is called.

Param	DESC
mute	Whether to pause receiving

Note

This API can be called before room entry (enterRoom), and the pause status will be reset after room exit (exitRoom). After calling this interface to pause receiving video streams from all users, simply calling the startRemoteView interface will not be able to play the video from a specific user. You need to call muteRemoteVideoStream(false) or muteAllRemoteVideoStreams(false) to resume it.

setVideoEncoderParam

setVideoEncoderParam

void setVideoEncoderParam	(const TRTCVideoEncParam& param)

Set the encoding parameters of video encoder

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.

Param	DESC
param	It is used to set relevant parameters for the video encoder. For more information, please see TRTCVideoEncParam.

Note

Begin from v11.5 version, the encoding output resolution will be aligned according to width 8 and height 2 bytes, and will be adjusted downward, eg: input resolution 540x960, actual encoding output resolution 536x960.

setNetworkQosParam

setNetworkQosParam

void setNetworkQosParam	(const TRTCNetworkQosParam& param)

Set network quality control parameters

This setting determines the quality control policy in a poor network environment, such as "image quality preferred" or "smoothness preferred".

Param	DESC
param	It is used to set relevant parameters for network quality control. For details, please refer to TRTCNetworkQosParam.

setLocalRenderParams

setLocalRenderParams

void setLocalRenderParams	(const TRTCRenderParams &params)

Set the rendering parameters of local video image

The parameters that can be set include video image rotation angle, fill mode, and mirror mode.

Param	DESC
params	Video image rendering parameters. For more information, please see TRTCRenderParams.

setRemoteRenderParams

setRemoteRenderParams

void setRemoteRenderParams	(const char* userId
	TRTCVideoStreamType streamType
	const TRTCRenderParams &params)

Set the rendering mode of remote video image

The parameters that can be set include video image rotation angle, fill mode, and mirror mode.

Param	DESC
params	Video image rendering parameters. For more information, please see TRTCRenderParams.
streamType	It can be set to the primary stream image (TRTCVideoStreamTypeBig) or substream image (TRTCVideoStreamTypeSub).
userId	ID of the specified remote user

enableSmallVideoStream

enableSmallVideoStream

void enableSmallVideoStream	(bool enable
	const TRTCVideoEncParam& smallVideoEncParam)

Enable dual-channel encoding mode with big and small images

In this mode, the current user's encoder will output two channels of video streams, i.e., **HD big image** and **Smooth small image**, at the same time (only one channel of audio stream will be output though).

In this way, other users in the room can choose to subscribe to the **HD big image** or **Smooth small image** according to their own network conditions or screen size.

Param	DESC
enable	Whether to enable small image encoding. Default value: false
smallVideoEncParam	Video parameters of small image stream

Note

Dual-channel encoding will consume more CPU resources and network bandwidth; therefore, this feature can be enabled on macOS, Windows, or high-spec tablets, but is not recommended for phones.

Return Desc:

0: success; -1: the current big image has been set to a lower quality, and it is not necessary to enable dual-channel encoding

setRemoteVideoStreamType

setRemoteVideoStreamType

void setRemoteVideoStreamType	(const char* userId
	TRTCVideoStreamType streamType)

Switch the big/small image of specified remote user

After an anchor in a room enables dual-channel encoding, the video image that other users in the room subscribe to through startRemoteView will be **HD big image** by default.
You can use this API to select whether the image subscribed to is the big image or small image. The API can take effect before or after startRemoteView is called.

Param	DESC
streamType	Video stream type, i.e., big image or small image. Default value: big image
userId	ID of the specified remote user

Note

To implement this feature, the target user must have enabled the dual-channel encoding mode through enableSmallVideoStream; otherwise, this API will not work.

snapshotVideo

snapshotVideo

void snapshotVideo	(const char* userId
	TRTCVideoStreamType streamType
	TRTCSnapshotSourceType sourceType)

Screencapture video

You can use this API to screencapture the local video image or the primary stream image and substream (screen sharing) image of a remote user.

Param	DESC
sourceType	Video image source, which can be the video stream image (TRTCSnapshotSourceTypeStream, generally in higher definition) 、 the video rendering image (TRTCSnapshotSourceTypeView) or the capture picture (TRTCSnapshotSourceTypeCapture).The captured picture screenshot will be clearer.
streamType	Video stream type, which can be the primary stream image (TRTCVideoStreamTypeBig, generally for camera) or substream image (TRTCVideoStreamTypeSub, generally for screen sharing)
userId	User ID. A null value indicates to screencapture the local video.

Note
On Windows, only video image from the TRTCSnapshotSourceTypeStream source can be screencaptured currently.

setGravitySensorAdaptiveMode

setGravitySensorAdaptiveMode

void setGravitySensorAdaptiveMode	(TRTCGravitySensorAdaptiveMode mode)

Set the adaptation mode of gravity sensing (version 11.7 and above)

After turning on gravity sensing, if the device on the collection end rotates, the images on the collection end and the audience will be rendered accordingly to ensure that the image in the field of view is always facing up.
It only takes effect in the camera capture scene inside the SDK, and only takes effect on the mobile terminal.

1. This interface only works for the collection end. If you only watch the picture in the room, opening this interface is invalid.
2. When the capture device is rotated 90 degrees or 270 degrees, the picture seen by the capture device or the audience may be cropped to maintain proportional coordination.

Param	DESC
mode	Gravity sensing mode, see TRTCGravitySensorAdaptiveMode_Disable、TRTCGravitySensorAdaptiveMode_FillByCenterCrop and TRTCGravitySensorAdaptiveMode_FitWithBlackBorder for details, default value: TRTCGravitySensorAdaptiveMode_Disable.

startLocalAudio

startLocalAudio

void startLocalAudio	(TRTCAudioQuality quality)

Enable local audio capturing and publishing

The SDK does not enable the mic by default. When a user wants to publish the local audio, the user needs to call this API to enable mic capturing and encode and publish the audio to the current room.
After local audio capturing and publishing is enabled, other users in the room will receive the onUserAudioAvailable(userId, true) notification.

Param	DESC
quality	Sound quality TRTCAudioQualitySpeech - Smooth: sample rate: 16 kHz; mono channel; audio bitrate: 16 Kbps. This is suitable for audio call scenarios, such as online meeting and audio call. TRTCAudioQualityDefault - Default: sample rate: 48 kHz; mono channel; audio bitrate: 50 Kbps. This is the default sound quality of the SDK and recommended if there are no special requirements. TRTCAudioQualityMusic - HD: sample rate: 48 kHz; dual channel + full band; audio bitrate: 128 Kbps. This is suitable for scenarios where Hi-Fi music transfer is required, such as online karaoke and music live streaming.

Note

This API will check the mic permission. If the current application does not have permission to use the mic, the SDK will automatically ask the user to grant the mic permission.

stopLocalAudio

stopLocalAudio

Stop local audio capturing and publishing

After local audio capturing and publishing is stopped, other users in the room will receive the onUserAudioAvailable(userId, false) notification.

muteLocalAudio

muteLocalAudio

void muteLocalAudio	(bool mute)

Pause/Resume publishing local audio stream

After local audio publishing is paused, other users in the room will receive the onUserAudioAvailable(userId, false) notification.
After local audio publishing is resumed, other users in the room will receive the onUserAudioAvailable(userId, true) notification.

Different from stopLocalAudio, `muteLocalAudio(true)` does not release the mic permission; instead, it continues to send mute packets with extremely low bitrate.
This is very suitable for scenarios that require on-cloud recording, as video file formats such as MP4 have a high requirement for audio continuity, while an MP4 recording file cannot be played back smoothly if stopLocalAudio is used.
Therefore, `muteLocalAudio` instead of `stopLocalAudio` is recommended in scenarios where the requirement for recording file quality is high.

Param	DESC
mute	true: mute; false: unmute

muteRemoteAudio

muteRemoteAudio

void muteRemoteAudio	(const char* userId
	bool mute)

Pause/Resume playing back remote audio stream

When you mute the remote audio of a specified user, the SDK will stop playing back the user's audio and pulling the user's audio data.

Param	DESC
mute	true: mute; false: unmute
userId	ID of the specified remote user

Note

This API works when called either before or after room entry (enterRoom), and the mute status will be reset to `false` after room exit (exitRoom).

muteAllRemoteAudio

muteAllRemoteAudio

void muteAllRemoteAudio	(bool mute)

Pause/Resume playing back all remote users' audio streams

When you mute the audio of all remote users, the SDK will stop playing back all their audio streams and pulling all their audio data.

Param	DESC
mute	true: mute; false: unmute

Note

This API works when called either before or after room entry (enterRoom), and the mute status will be reset to `false` after room exit (exitRoom).

setRemoteAudioVolume

setRemoteAudioVolume

void setRemoteAudioVolume	(const char *userId
	int volume)

Set the audio playback volume of remote user

You can mute the audio of a remote user through `setRemoteAudioVolume(userId, 0)` .

Param	DESC
userId	ID of the specified remote user
volume	Volume. 100 is the original volume. Value range: [0,150]. Default value: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setAudioCaptureVolume

setAudioCaptureVolume

void setAudioCaptureVolume	(int volume)

Set the capturing volume of local audio

Param	DESC
volume	Volume. 100 is the original volume. Value range: [0,150]. Default value: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

getAudioCaptureVolume

getAudioCaptureVolume

Get the capturing volume of local audio

setAudioPlayoutVolume

setAudioPlayoutVolume

void setAudioPlayoutVolume	(int volume)

Set the playback volume of remote audio

This API controls the volume of the sound ultimately delivered by the SDK to the system for playback. It affects the volume of the recorded local audio file but not the volume of in-ear monitoring.

Param	DESC
volume	Volume. 100 is the original volume. Value range: [0,150]. Default value: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

getAudioPlayoutVolume

getAudioPlayoutVolume

Get the playback volume of remote audio

enableAudioVolumeEvaluation

enableAudioVolumeEvaluation

void enableAudioVolumeEvaluation	(bool enable
	const TRTCAudioVolumeEvaluateParams& params)

Enable volume reminder

After this feature is enabled, the SDK will return the audio volume assessment information of local user who sends stream and remote users in the onUserVoiceVolume callback of ITRTCCloudCallback.

Param	DESC
enable	Whether to enable the volume prompt. It's disabled by default.
params	Volume evaluation and other related parameters, please see TRTCAudioVolumeEvaluateParams

Note

To enable this feature, call this API before calling `startLocalAudio` .

startAudioRecording

startAudioRecording

int startAudioRecording	(const TRTCAudioRecordingParams& param)

Start audio recording

After you call this API, the SDK will selectively record local and remote audio streams (such as local audio, remote audio, background music, and sound effects) into a local file.

This API works when called either before or after room entry. If a recording task has not been stopped through `stopAudioRecording` before room exit, it will be automatically stopped after room exit.

The startup and completion status of the recording will be notified through local recording-related callbacks. See TRTCCloud related callbacks for reference.

Param	DESC
param	Recording parameter. For more information, please see TRTCAudioRecordingParams

Note

Since version 11.5, the results of audio recording have been changed to be notified through asynchronous callbacks instead of return values. Please refer to the relevant callbacks of TRTCCloud.

Return Desc:

0: success; -1: audio recording has been started; -2: failed to create file or directory; -3: the audio format of the specified file extension is not supported.

stopAudioRecording

stopAudioRecording

Stop audio recording

If a recording task has not been stopped through this API before room exit, it will be automatically stopped after room exit.

startLocalRecording

startLocalRecording

void startLocalRecording	(const TRTCLocalRecordingParams& params)

Start local media recording

This API records the audio/video content during live streaming into a local file.

Param	DESC
params	Recording parameter. For more information, please see TRTCLocalRecordingParams

stopLocalRecording

stopLocalRecording

Stop local media recording

If a recording task has not been stopped through this API before room exit, it will be automatically stopped after room exit.

setRemoteAudioParallelParams

setRemoteAudioParallelParams

void setRemoteAudioParallelParams	(const TRTCAudioParallelParams& params)

Set the parallel strategy of remote audio streams

For room with many speakers.

Param	DESC
params	Audio parallel parameter. For more information, please see TRTCAudioParallelParams

enable3DSpatialAudioEffect

enable3DSpatialAudioEffect

void enable3DSpatialAudioEffect	(bool enabled)

Enable 3D spatial effect

Enable 3D spatial effect. Note that TRTCAudioQualitySpeech smooth or TRTCAudioQualityDefault default audio quality should be used.

Param	DESC
enabled	Whether to enable 3D spatial effect. It's disabled by default.

updateSelf3DSpatialPosition

updateSelf3DSpatialPosition

void updateSelf3DSpatialPosition	(int position[3]
	float axisForward[3]
	float axisRight[3]
	float axisUp[3])

Update self position and orientation for 3D spatial effect

Update self position and orientation in the world coordinate system. The SDK will calculate the relative position between self and the remote users according to the parameters of this method, and then render the spatial sound effect. Note that the length of array should be 3.

Param	DESC
axisForward	The unit vector of the forward axis of user coordinate system. The three values represent the forward, right and up coordinate values in turn.
axisRight	The unit vector of the right axis of user coordinate system. The three values represent the forward, right and up coordinate values in turn.
axisUp	The unit vector of the up axis of user coordinate system. The three values represent the forward, right and up coordinate values in turn.
position	The coordinate of self in the world coordinate system. The three values represent the forward, right and up coordinate values in turn.

Note

Please limit the calling frequency appropriately. It's recommended that the interval between two operations be at least 100ms.

updateRemote3DSpatialPosition

updateRemote3DSpatialPosition

void updateRemote3DSpatialPosition	(const char* userId
	int position[3])

Update the specified remote user's position for 3D spatial effect

Update the specified remote user's position in the world coordinate system. The SDK will calculate the relative position between self and the remote users according to the parameters of this method, and then render the spatial sound effect. Note that the length of array should be 3.

Param	DESC
position	The coordinate of self in the world coordinate system. The three values represent the forward, right and up coordinate values in turn.
userId	ID of the specified remote user.

Note

Please limit the calling frequency appropriately. It's recommended that the interval between two operations of the same remote user be at least 100ms.

set3DSpatialReceivingRange

set3DSpatialReceivingRange

void set3DSpatialReceivingRange	(const char* userId
	int range)

Set the maximum 3D spatial attenuation range for userId's audio stream

After set the range, the specified user's audio stream will attenuate to zero within the range.

Param	DESC
range	Maximum attenuation range of the audio stream.
userId	ID of the specified user.

*getDeviceManager

***getDeviceManager**

Get device management class (TXDeviceManager)

setBeautyStyle

setBeautyStyle

void setBeautyStyle	(TRTCBeautyStyle style
	uint32_t beautyLevel
	uint32_t whitenessLevel
	uint32_t ruddinessLevel)

Set special effects such as beauty, brightening, and rosy skin filters

The SDK is integrated with two skin smoothing algorithms of different styles:

"Smooth" style, which uses a more radical algorithm for more obvious effect and is suitable for show live streaming.

"Natural" style, which retains more facial details for more natural effect and is suitable for most live streaming use cases.

Param	DESC
beautyLevel	Strength of the beauty filter. Value range: 0–9; 0 indicates that the filter is disabled, and the greater the value, the more obvious the effect.
ruddinessLevel	Strength of the rosy skin filter. Value range: 0–9; 0 indicates that the filter is disabled, and the greater the value, the more obvious the effect.
style	Skin smoothening algorithm ("smooth" or "natural")
whitenessLevel	Strength of the brightening filter. Value range: 0–9; 0 indicates that the filter is disabled, and the greater the value, the more obvious the effect.

setWaterMark

setWaterMark

void setWaterMark	(TRTCVideoStreamType streamType

	const char* srcData
	TRTCWaterMarkSrcType srcType
	uint32_t nWidth
	uint32_t nHeight
	float xOffset
	float yOffset
	float fWidthRatio
	bool isVisibleOnLocalPreview = false)

Add watermark

The watermark position is determined by the `xOffset` , `yOffset` , and `fWidthRatio` parameters.

`xOffset` : X coordinate of watermark, which is a floating-point number between 0 and 1.

`yOffset` : Y coordinate of watermark, which is a floating-point number between 0 and 1.

`fWidthRatio` : watermark dimensions ratio, which is a floating-point number between 0 and 1.

Param	DESC
fWidthRatio	Ratio of watermark width to image width (the watermark will be scaled according to this parameter)
isVisibleOnLocalPreview	true: local preview show wartermark;false: local preview hide wartermark.only effect on win/mac.
nHeight	Pixel height of watermark image (this parameter will be ignored if the source data is a file path)
nWidth	Pixel width of watermark image (this parameter will be ignored if the source data is a file path)
srcData	Source data of watermark image (if `nullptr` is passed in, the watermark will be removed)
srcType	Source data type of watermark image
streamType	Stream type of the watermark to be set (`TRTCVideoStreamTypeBig` or `TRTCVideoStreamTypeSub`)
xOffset	Top-left offset on the X axis of watermark

yOffset	Top-left offset on the Y axis of watermark

Note
This API only supports adding an image watermark to the primary stream

getAudioEffectManager

getAudioEffectManager

Get sound effect management class (TXAudioEffectManager)

`TXAudioEffectManager` is a sound effect management API, through which you can implement the following features:
Background music: both online music and local music can be played back with various features such as speed adjustment, pitch adjustment, original voice, accompaniment, and loop.
In-ear monitoring: the sound captured by the mic is played back in the headphones in real time, which is generally used for music live streaming.
Reverb effect: karaoke room, small room, big hall, deep, resonant, and other effects.
Voice changing effect: young girl, middle-aged man, heavy metal, and other effects.
Short sound effect: short sound effect files such as applause and laughter are supported (for files less than 10 seconds in length, please set the `isShortFile` parameter to `true`).

startSystemAudioLoopback

startSystemAudioLoopback

void startSystemAudioLoopback	(const char* deviceName = nullptr)

Enable system audio capturing(iOS not supported)

This API captures audio data from the sound card of the anchor's computer and mixes it into the current audio stream of the SDK. This ensures that other users in the room hear the audio played back by the anchor's computer.
In online education scenarios, a teacher can use this API to have the SDK capture the audio of instructional videos and broadcast it to students in the room.
In live music scenarios, an anchor can use this API to have the SDK capture the music played back by his or her player so as to add background music to the room.

Param	DESC
deviceName	If this parameter is empty, the audio of the entire system is captured. On Windows, if the

	parameter is a speaker name, you can capture this speaker. About speaker device name you can see TXDeviceManager On Windows, you can also set `deviceName` to the deviceName of an executable file (such as `QQMuisc.exe`) to have the SDK capture only the audio of the application.

Note

You can specify `deviceName` only on Windows and with 32-bit TRTC SDK.

stopSystemAudioLoopback

stopSystemAudioLoopback

Stop system audio capturing(iOS not supported)

setSystemAudioLoopbackVolume

setSystemAudioLoopbackVolume

void setSystemAudioLoopbackVolume	(uint32_t volume)

Set the volume of system audio capturing

Param	DESC
volume	Set volume. Value range: [0, 150]. Default value: 100

startScreenCapture

startScreenCapture

void startScreenCapture	(TXView view
	TRTCVideoStreamType streamType
	TRTCVideoEncParam* encParam)

Start screen sharing

This API can capture the content of the entire screen or a specified application and share it with other users in the same room.

Param	DESC
encParam	Image encoding parameters used for screen sharing, which can be set to empty, indicating to let the SDK choose the optimal encoding parameters (such as resolution and bitrate).
streamType	Channel used for screen sharing, which can be the primary stream (TRTCVideoStreamTypeBig) or substream (TRTCVideoStreamTypeSub).
view	Parent control of the rendering control, which can be set to a null value, indicating not to display the preview of the shared screen.

Note

1. A user can publish at most one primary stream (TRTCVideoStreamTypeBig) and one substream (TRTCVideoStreamTypeSub) at the same time.
2. By default, screen sharing uses the substream image. If you want to use the primary stream for screen sharing, you need to stop camera capturing (through stopLocalPreview) in advance to avoid conflicts.
3. Only one user can use the substream for screen sharing in the same room at any time; that is, only one user is allowed to enable the substream in the same room at any time.
4. When there is already a user in the room using the substream for screen sharing, calling this API will return the `onError(ERR_SERVER_CENTER_ANOTHER_USER_PUSH_SUB_VIDEO)` callback from ITRTCCloudCallback.

stopScreenCapture

stopScreenCapture

Stop screen sharing

pauseScreenCapture

pauseScreenCapture

Pause screen sharing

Note

Begin from v11.5 version, paused screen capture will use the last frame to output at a frame rate of 1fps.

resumeScreenCapture

resumeScreenCapture

Resume screen sharing

getScreenCaptureSources

getScreenCaptureSources

ITRTCScreenCaptureSourceList* getScreenCaptureSources	(const SIZE &thumbnailSize
	const SIZE &iconSize)

Enumerate shareable screens and windows (for desktop systems only)

When you integrate the screen sharing feature of a desktop system, you generally need to display a UI for selecting the sharing target, so that users can use the UI to choose whether to share the entire screen or a certain window. Through this API, you can query the IDs, names, and thumbnails of sharable windows on the current system. We provide a default UI implementation in the demo for your reference.

Param	DESC
iconSize	Specify the icon size of the window to be obtained.
thumbnailSize	Specify the thumbnail size of the window to be obtained. The thumbnail can be drawn on the window selection UI.

Note

1. The returned list contains the screen and the application windows. The screen is the first element in the list. If the user has multiple displays, then each display is a sharing target.
2. Please do not use `delete ITRTCScreenCaptureSourceList*` to delete the `SourceList` ; otherwise, crashes may occur. Instead, please use the `release` method in `ITRTCScreenCaptureSourceList` to release the list.

Return Desc:

List of windows (including the screen)

selectScreenCaptureTarget

selectScreenCaptureTarget

void selectScreenCaptureTarget	(const TRTCScreenCaptureSourceInfo &source
	const RECT& captureRect

	const TRTCScreenCaptureProperty &property)

Select the screen or window to share (for desktop systems only)

After you get the sharable screens and windows through `getScreenCaptureSources`, you can call this API to select the target screen or window you want to share.
During the screen sharing process, you can also call this API at any time to switch the sharing target.

The following four sharing modes are supported:
Sharing the entire screen: for `source` whose `type` is `Screen` in `sourceInfoList`, set `captureRect` to `{ 0, 0, 0, 0 }`.
Sharing a specified area: for `source` whose `type` is `Screen` in `sourceInfoList`, set `captureRect` to a non-nullptr value, e.g., `{ 100, 100, 300, 300 }`.
Sharing an entire window: for `source` whose `type` is `Window` in `sourceInfoList`, set `captureRect` to `{ 0, 0, 0, 0 }`.
Sharing a specified window area: for `source` whose `type` is `Window` in `sourceInfoList`, set `captureRect` to a non-nullptr value, e.g., `{ 100, 100, 300, 300 }`.

Param	DESC
captureRect	Specify the area to be captured
property	Specify the attributes of the screen sharing target, such as capturing the cursor and highlighting the captured window. For more information, please see the definition of `TRTCScreenCaptureProperty`
source	Specify sharing source

Note
Setting the highlight border color and width parameters does not take effect on macOS.

setSubStreamEncoderParam

setSubStreamEncoderParam

void setSubStreamEncoderParam	(const TRTCVideoEncParam& param)

Set the video encoding parameters of screen sharing (i.e., substream) (for desktop and mobile systems)

This API can set the image quality of screen sharing (i.e., the substream) viewed by remote users, which is also the image quality of screen sharing in on-cloud recording files.

Please note the differences between the following two APIs:
setVideoEncoderParam is used to set the video encoding parameters of the primary stream image (TRTCVideoStreamTypeBig, generally for camera).
setSubStreamEncoderParam is used to set the video encoding parameters of the substream image (TRTCVideoStreamTypeSub, generally for screen sharing).

Param	DESC
param	Substream encoding parameters. For more information, please see TRTCVideoEncParam.

setSubStreamMixVolume

setSubStreamMixVolume

void setSubStreamMixVolume	(uint32_t volume)

Set the audio mixing volume of screen sharing (for desktop systems only)

The greater the value, the larger the ratio of the screen sharing volume to the mic volume. We recommend you not set a high value for this parameter as a high volume will cover the mic sound.

Param	DESC
volume	Set audio mixing volume. Value range: 0–100

addExcludedShareWindow

addExcludedShareWindow

void addExcludedShareWindow	(TXView windowID)

Add specified windows to the exclusion list of screen sharing (for desktop systems only)

The excluded windows will not be shared. This feature is generally used to add a certain application's window to the exclusion list to avoid privacy issues.
You can set the filtered windows before starting screen sharing or dynamically add the filtered windows during screen sharing.

Param	DESC
window	Window not to be shared

Note

1. This API takes effect only if the `type` in TRTCScreenCaptureSourceInfo is specified as TRTCScreenCaptureSourceTypeScreen; that is, the feature of excluding specified windows works only when the entire screen is shared.
2. The windows added to the exclusion list through this API will be automatically cleared by the SDK after room exit.
3. On macOS, please pass in the window ID (CGWindowID), which can be obtained through the `sourceId` member in TRTCScreenCaptureSourceInfo.

removeExcludedShareWindow

removeExcludedShareWindow

void removeExcludedShareWindow	(TXView windowID)

Remove specified windows from the exclusion list of screen sharing (for desktop systems only)

Param	DESC
windowID	

removeAllExcludedShareWindow

removeAllExcludedShareWindow

Remove all windows from the exclusion list of screen sharing (for desktop systems only)

addIncludedShareWindow

addIncludedShareWindow

void addIncludedShareWindow	(TXView windowID)

Add specified windows to the inclusion list of screen sharing (for desktop systems only)

This API takes effect only if the `type` in TRTCScreenCaptureSourceInfo is specified as TRTCScreenCaptureSourceTypeWindow; that is, the feature of additionally including specified windows works only when a window is shared.
You can call it before or after startScreenCapture.

Param	DESC
windowID	Window to be shared (which is a window handle `HWND` on Windows)

Note
The windows added to the inclusion list by this method will be automatically cleared by the SDK after room exit.

removeIncludedShareWindow

removeIncludedShareWindow

void removeIncludedShareWindow	(TXView windowID)

Remove specified windows from the inclusion list of screen sharing (for desktop systems only)

This API takes effect only if the `type` in TRTCScreenCaptureSourceInfo is specified as TRTCScreenCaptureSourceTypeWindow.
That is, the feature of additionally including specified windows works only when a window is shared.

Param	DESC
windowID	Window to be shared (window ID on macOS or HWND on Windows)

removeAllIncludedShareWindow

removeAllIncludedShareWindow

Remove all windows from the inclusion list of screen sharing (for desktop systems only)

This API takes effect only if the `type` in TRTCScreenCaptureSourceInfo is specified as TRTCScreenCaptureSourceTypeWindow.
That is, the feature of additionally including specified windows works only when a window is shared.

enableCustomVideoCapture

enableCustomVideoCapture

void enableCustomVideoCapture	(TRTCVideoStreamType streamType
	bool enable)

Enable/Disable custom video capturing mode

After this mode is enabled, the SDK will not run the original video capturing process (i.e., stopping camera data capturing and beauty filter operations) and will retain only the video encoding and sending capabilities.
You need to use sendCustomVideoData to continuously insert the captured video image into the SDK.

Param	DESC
enable	Whether to enable. Default value: false
streamType	Specify video stream type (TRTCVideoStreamTypeBig: HD big image; TRTCVideoStreamTypeSub: substream image).

sendCustomVideoData

sendCustomVideoData

void sendCustomVideoData	(TRTCVideoStreamType streamType
	TRTCVideoFrame* frame)

Deliver captured video frames to SDK

You can use this API to deliver video frames you capture to the SDK, and the SDK will encode and transfer them through its own network module.

We recommend you enter the following information for the TRTCVideoFrame parameter (other fields can be left empty):
pixelFormat: on Windows and Android, only TRTCVideoPixelFormat_I420 is supported; on iOS and macOS, TRTCVideoPixelFormat_I420 and TRTCVideoPixelFormat_BGRA32 are supported.
bufferType: TRTCVideoBufferType_Buffer is recommended.
data: buffer used to carry video frame data.
length: video frame data length. If `pixelFormat` is set to I420, `length` can be calculated according to the following formula: length = width * height * 3 / 2.
width: video image width, such as 640 px.
height: video image height, such as 480 px.
timestamp (ms): Set it to the timestamp when video frames are captured, which you can obtain by calling generateCustomPTS after getting a video frame.

For more information, please see Custom Capturing and Rendering.

Param	DESC

frame	Video data, which can be in I420 format.
streamType	Specify video stream type (TRTCVideoStreamTypeBig: HD big image; TRTCVideoStreamTypeSub: substream image).

Note

1. We recommend you call the generateCustomPTS API to get the `timestamp` value of a video frame immediately after capturing it, so as to achieve the best audio/video sync effect.
2. The video frame rate eventually encoded by the SDK is not determined by the frequency at which you call this API, but by the FPS you set in setVideoEncoderParam.
3. Please try to keep the calling interval of this API even; otherwise, problems will occur, such as unstable output frame rate of the encoder or out-of-sync audio/video.
4. On iOS and macOS, video frames in TRTCVideoPixelFormat_I420 or TRTCVideoPixelFormat_BGRA32 format can be passed in currently.
5. On Windows and Android, only video frames in TRTCVideoPixelFormat_I420 format can be passed in currently.

enableCustomAudioCapture

enableCustomAudioCapture

void enableCustomAudioCapture	(bool enable)

Enable custom audio capturing mode

After this mode is enabled, the SDK will not run the original audio capturing process (i.e., stopping mic data capturing) and will retain only the audio encoding and sending capabilities.
You need to use sendCustomAudioData to continuously insert the captured audio data into the SDK.

Param	DESC
enable	Whether to enable. Default value: false

Note

As acoustic echo cancellation (AEC) requires strict control over the audio capturing and playback time, after custom audio capturing is enabled, AEC may fail.

sendCustomAudioData

sendCustomAudioData

void sendCustomAudioData	(TRTCAudioFrame* frame)

Deliver captured audio data to SDK

We recommend you enter the following information for the TRTCAudioFrame parameter (other fields can be left empty):

audioFormat: audio data format, which can only be `TRTCAudioFrameFormatPCM` .

data: audio frame buffer. Audio frame data must be in PCM format, and it supports a frame length of 5–100 ms (20 ms is recommended). Length calculation method: **for example, if the sample rate is 48000, then the frame length for mono channel will be `48000 * 0.02s * 1 * 16 bit = 15360 bit = 1920 bytes`.**

sampleRate: sample rate. Valid values: 16000, 24000, 32000, 44100, 48000.

channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel.

timestamp (ms): Set it to the timestamp when audio frames are captured, which you can obtain by calling generateCustomPTS after getting a audio frame.

For more information, please see Custom Capturing and Rendering.

Param	DESC
frame	Audio data

Note

Please call this API accurately at intervals of the frame length; otherwise, sound lag may occur due to uneven data delivery intervals.

enableMixExternalAudioFrame

enableMixExternalAudioFrame

void enableMixExternalAudioFrame	(bool enablePublish
	bool enablePlayout)

Enable/Disable custom audio track

After this feature is enabled, you can mix a custom audio track into the SDK through this API. With two boolean parameters, you can control whether to play back this track remotely or locally.

Param	DESC
enablePlayout	Whether the mixed audio track should be played back locally. Default value: false

enablePublish	Whether the mixed audio track should be played back remotely. Default value: false

Note

If you specify both `enablePublish` and `enablePlayout` as `false` , the custom audio track will be completely closed.

mixExternalAudioFrame

mixExternalAudioFrame

int mixExternalAudioFrame	(TRTCAudioFrame* frame)

Mix custom audio track into SDK

Before you use this API to mix custom PCM audio into the SDK, you need to first enable custom audio tracks through enableMixExternalAudioFrame.

You are expected to feed audio data into the SDK at an even pace, but we understand that it can be challenging to call an API at absolutely regular intervals.

Given this, we have provided a buffer pool in the SDK, which can cache the audio data you pass in to reduce the fluctuations in intervals between API calls.

The value returned by this API indicates the size (ms) of the buffer pool. For example, if `50` is returned, it indicates that the buffer pool has 50 ms of audio data. As long as you call this API again within 50 ms, the SDK can make sure that continuous audio data is mixed.

If the value returned is `100` or greater, you can wait after an audio frame is played to call the API again. If the value returned is smaller than `100` , then there isn't enough data in the buffer pool, and you should feed more audio data into the SDK until the data in the buffer pool is above the safety level.

Fill the fields in TRTCAudioFrame as follows (other fields are not required).

`data` : audio frame buffer. Audio frames must be in PCM format. Each frame can be 5-100 ms (20 ms is recommended) in duration. Assume that the sample rate is 48000, and sound channels mono-channel. Then the **frame size would be 48000 x 0.02s x 1 x 16 bit = 15360 bit = 1920 bytes**.

`sampleRate` : sample rate. Valid values: 16000, 24000, 32000, 44100, 48000

`channel` : number of sound channels (if dual-channel is used, data is interleaved). Valid values: `1` (mono-channel); `2` (dual channel)

`timestamp` : timestamp (ms). Set it to the timestamp when audio frames are captured, which you can obtain by calling generateCustomPTS after getting an audio frame.

Param	DESC

frame	Audio data

Return Desc:

If the value returned is `0` or greater, the value represents the current size of the buffer pool; if the value returned is smaller than `0` , it means that an error occurred. `-1` indicates that you didn't call enableMixExternalAudioFrame to enable custom audio tracks.

setMixExternalAudioVolume

setMixExternalAudioVolume

void setMixExternalAudioVolume	(int publishVolume
	int playoutVolume)

Set the publish volume and playback volume of mixed custom audio track

Param	DESC
playoutVolume	set the play volume, from 0 to 100, -1 means no change
publishVolume	set the publish volume, from 0 to 100, -1 means no change

generateCustomPTS

generateCustomPTS

Generate custom capturing timestamp

This API is only suitable for the custom capturing mode and is used to solve the problem of out-of-sync audio/video caused by the inconsistency between the capturing time and delivery time of audio/video frames.

When you call APIs such as sendCustomVideoData or sendCustomAudioData for custom video or audio capturing, please use this API as instructed below:
1. First, when a video or audio frame is captured, call this API to get the corresponding PTS timestamp.
2. Then, send the video or audio frame to the preprocessing module you use (such as a third-party beauty filter or sound effect component).
3. When you actually call sendCustomVideoData or sendCustomAudioData for delivery, assign the PTS timestamp recorded when the frame was captured to the `timestamp` field in TRTCVideoFrame or TRTCAudioFrame.

Return Desc:

Timestamp in ms

enableLocalVideoCustomProcess

enableLocalVideoCustomProcess

int enableLocalVideoCustomProcess	(bool enable
	TRTCVideoPixelFormat pixelFormat
	TRTCVideoBufferType bufferType)

.1 Enable third-party beauty filters in video

After it is enabled, you can get the image frame of the specified pixel format and video data structure type through ITRTCVideoFrameCallback.

Param	DESC
bufferType	Specify the format of the data called back.
enable	Whether to enable local video process. It's disabled by default.
pixelFormat	Specify the format of the pixel called back.

Return Desc:

0: success; values smaller than 0: error

setLocalVideoCustomProcessCallback

setLocalVideoCustomProcessCallback

void setLocalVideoCustomProcessCallback	(ITRTCVideoFrameCallback* callback)

.2 Set video data callback for third-party beauty filters

After this callback is set, the SDK will call back the captured video frames through the `callback` you set and use them for further processing by a third-party beauty filter component. Then, the SDK will encode and send the processed video frames.

Param	DESC

callback	: Custom preprocessing callback. For more information, please see ITRTCVideoFrameCallback

setLocalVideoRenderCallback

setLocalVideoRenderCallback

int setLocalVideoRenderCallback	(TRTCVideoPixelFormat pixelFormat
	TRTCVideoBufferType bufferType
	ITRTCVideoRenderCallback* callback)

Set the callback of custom rendering for local video

After this callback is set, the SDK will skip its own rendering process and call back the captured data. Therefore, you need to complete image rendering on your own.

You can call `setLocalVideoRenderCallback(TRTCVideoPixelFormat_Unknown, TRTCVideoBufferType_Unknown, nullptr)` to stop the callback.

On iOS, macOS, and Windows, only video frames in TRTCVideoPixelFormat_I420 or TRTCVideoPixelFormat_BGRA32 pixel format can be called back currently.

On Android, only video frames in TRTCVideoPixelFormat_I420, TRTCVideoPixelFormat_RGBA32 or TRTCVideoPixelFormat_Texture_2D pixel format can be passed in currently.

Param	DESC
bufferType	Specify video data structure type.
callback	Callback for custom rendering
pixelFormat	Specify the format of the pixel called back

Return Desc:

0: success; values smaller than 0: error

setRemoteVideoRenderCallback

setRemoteVideoRenderCallback

int setRemoteVideoRenderCallback	(const char* userId
	TRTCVideoPixelFormat pixelFormat

	TRTCVideoBufferType bufferType
	ITRTCVideoRenderCallback* callback)

Set the callback of custom rendering for remote video

After this callback is set, the SDK will skip its own rendering process and call back the captured data. Therefore, you need to complete image rendering on your own.

You can call `setRemoteVideoRenderCallback(TRTCVideoPixelFormat_Unknown, TRTCVideoBufferType_Unknown, nullptr)` to stop the callback.

On iOS, macOS, and Windows, only video frames in TRTCVideoPixelFormat_I420 or TRTCVideoPixelFormat_BGRA32 pixel format can be called back currently.

On Android, only video frames in TRTCVideoPixelFormat_I420 , TRTCVideoPixelFormat_RGBA32 or TRTCVideoPixelFormat_Texture_2Dpixel format can be passed in currently.

Param	DESC
bufferType	Specify video data structure type. Only TRTCVideoBufferType_Buffer is supported currently
callback	Callback for custom rendering
pixelFormat	Specify the format of the pixel called back
userId	remote user id

Note

In actual use, you need to call `startRemoteView(userid, nullptr)` to get the video stream of the remote user first (set `view` to `nullptr`); otherwise, there will be no data called back.

Return Desc:

0: success; values smaller than 0: error

setAudioFrameCallback

setAudioFrameCallback

int setAudioFrameCallback	(ITRTCAudioFrameCallback* callback)

Set custom audio data callback

After this callback is set, the SDK will internally call back the audio data (in PCM format), including:

onCapturedAudioFrame: callback of the audio data captured by the local mic
onLocalProcessedAudioFrame: callback of the audio data captured by the local mic and preprocessed by the audio module
onPlayAudioFrame: audio data from each remote user before audio mixing
onMixedPlayAudioFrame: callback of the audio data that will be played back by the system after audio streams are mixed

Note

Setting the callback to null indicates to stop the custom audio callback, while setting it to a non-null value indicates to start the custom audio callback.

setCapturedAudioFrameCallbackFormat

setCapturedAudioFrameCallbackFormat

int setCapturedAudioFrameCallbackFormat	(TRTCAudioFrameCallbackFormat* format)

Set the callback format of audio frames captured by local mic

This API is used to set the `AudioFrame` format called back by onCapturedAudioFrame:
sampleRate: sample rate. Valid values: 16000, 32000, 44100, 48000
channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel
samplesPerCall: number of sample points, which defines the frame length of the callback data. The frame length must be an integer multiple of 10 ms.

If you want to calculate the callback frame length in milliseconds, the formula for converting the number of milliseconds into the number of sample points is as follows: number of sample points = number of milliseconds * sample rate / 1000
For example, if you want to call back the data of 20 ms frame length with 48000 sample rate, then the number of sample points should be entered as 960 = 20 * 48000 / 1000
Note that the frame length of the final callback is in bytes, and the calculation formula for converting the number of sample points into the number of bytes is as follows: number of bytes = number of sample points * number of channels * 2 (bit width)
For example, if the parameters are 48000 sample rate, dual channel, 20 ms frame length, and 960 sample points, then the number of bytes is 3840 = 960 * 2 * 2

Param	DESC
format	Audio data callback format

Return Desc:

0: success; values smaller than 0: error

setLocalProcessedAudioFrameCallbackFormat

setLocalProcessedAudioFrameCallbackFormat

int setLocalProcessedAudioFrameCallbackFormat	(TRTCAudioFrameCallbackFormat* format)

Set the callback format of preprocessed local audio frames

This API is used to set the `AudioFrame` format called back by onLocalProcessedAudioFrame:
sampleRate: sample rate. Valid values: 16000, 32000, 44100, 48000
channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel
samplesPerCall: number of sample points, which defines the frame length of the callback data. The frame length must be an integer multiple of 10 ms.

If you want to calculate the callback frame length in milliseconds, the formula for converting the number of milliseconds into the number of sample points is as follows: number of sample points = number of milliseconds * sample rate / 1000
For example, if you want to call back the data of 20 ms frame length with 48000 sample rate, then the number of sample points should be entered as 960 = 20 * 48000 / 1000
Note that the frame length of the final callback is in bytes, and the calculation formula for converting the number of sample points into the number of bytes is as follows: number of bytes = number of sample points * number of channels * 2 (bit width)
For example, if the parameters are 48000 sample rate, dual channel, 20 ms frame length, and 960 sample points, then the number of bytes is 3840 = 960 * 2 * 2

Param	DESC
format	Audio data callback format

Return Desc:

0: success; values smaller than 0: error

setMixedPlayAudioFrameCallbackFormat

setMixedPlayAudioFrameCallbackFormat

int setMixedPlayAudioFrameCallbackFormat	(TRTCAudioFrameCallbackFormat* format)

Set the callback format of audio frames to be played back by system

This API is used to set the `AudioFrame` format called back by onMixedPlayAudioFrame:
sampleRate: sample rate. Valid values: 16000, 32000, 44100, 48000
channel: number of channels (if stereo is used, data is interwoven). Valid values: 1: mono channel; 2: dual channel
samplesPerCall: number of sample points, which defines the frame length of the callback data. The frame length must be an integer multiple of 10 ms.

If you want to calculate the callback frame length in milliseconds, the formula for converting the number of milliseconds into the number of sample points is as follows: number of sample points = number of milliseconds * sample rate / 1000
For example, if you want to call back the data of 20 ms frame length with 48000 sample rate, then the number of sample points should be entered as 960 = 20 * 48000 / 1000
Note that the frame length of the final callback is in bytes, and the calculation formula for converting the number of sample points into the number of bytes is as follows: number of bytes = number of sample points * number of channels * 2 (bit width)
For example, if the parameters are 48000 sample rate, dual channel, 20 ms frame length, and 960 sample points, then the number of bytes is 3840 = 960 * 2 * 2

Param	DESC
format	Audio data callback format

Return Desc:

0: success; values smaller than 0: error

enableCustomAudioRendering

enableCustomAudioRendering

void enableCustomAudioRendering	(bool enable)

Enabling custom audio playback

You can use this API to enable custom audio playback if you want to connect to an external audio device or control the audio playback logic by yourself.
After you enable custom audio playback, the SDK will stop using its audio API to play back audio. You need to call getCustomAudioRenderingFrame to get audio frames and play them by yourself.

Param	DESC
enable	Whether to enable custom audio playback. It's disabled by default.

Note

The parameter must be set before room entry to take effect.

getCustomAudioRenderingFrame

getCustomAudioRenderingFrame

void getCustomAudioRenderingFrame	(TRTCAudioFrame* audioFrame)

Getting playable audio data

Before calling this API, you need to first enable custom audio playback using enableCustomAudioRendering.

Fill the fields in TRTCAudioFrame as follows (other fields are not required):

`sampleRate` : sample rate (required). Valid values: 16000, 24000, 32000, 44100, 48000

`channel` : number of sound channels (required). `1` : mono-channel; `2` : dual-channel; if dual-channel is used, data is interleaved.

`data` : the buffer used to get audio data. You need to allocate memory for the buffer based on the duration of an audio frame.

The PCM data obtained can have a frame duration of 10 ms or 20 ms. 20 ms is recommended.

Assume that the sample rate is 48000, and sound channels mono-channel. The buffer size for a 20 ms audio frame would be 48000 x 0.02s x 1 x 16 bit = 15360 bit = 1920 bytes.

Param	DESC
audioFrame	Audio frames

Note

1. You must set `sampleRate` and `channel` in `audioFrame` , and allocate memory for one frame of audio in advance.
2. The SDK will fill the data automatically based on `sampleRate` and `channel` .
3. We recommend that you use the system's audio playback thread to drive the calling of this API, so that it is called each time the playback of an audio frame is complete.

sendCustomCmdMsg

sendCustomCmdMsg

bool sendCustomCmdMsg	(uint32_t cmdId
	const uint8_t* data
	uint32_t dataSize
	bool reliable
	bool ordered)

Use UDP channel to send custom message to all users in room

This API allows you to use TRTC's UDP channel to broadcast custom data to other users in the current room for signaling transfer.

Other users in the room can receive the message through the `onRecvCustomCmdMsg` callback in ITRTCCloudCallback.

Param	DESC
cmdID	Message ID. Value range: 1–10
data	Message to be sent. The maximum length of one single message is 1 KB.
ordered	Whether orderly sending is enabled, i.e., whether the data packets should be received in the same order in which they are sent; if so, a certain delay will be caused.
reliable	Whether reliable sending is enabled. Reliable sending can achieve a higher success rate but with a longer reception delay than unreliable sending.

Note

1. Up to 30 messages can be sent per second to all users in the room (this is not supported for web and mini program currently).
2. A packet can contain up to 1 KB of data; if the threshold is exceeded, the packet is very likely to be discarded by the intermediate router or server.
3. A client can send up to 8 KB of data in total per second.
4. `reliable` and `ordered` must be set to the same value (`true` or `false`) and cannot be set to different values currently.
5. We strongly recommend you set different `cmdID` values for messages of different types. This can reduce message delay when orderly sending is required.

6. Currently only the anchor role is supported.

Return Desc:

true: sent the message successfully; false: failed to send the message.

sendSEIMsg

sendSEIMsg

bool sendSEIMsg	(const uint8_t* data
	uint32_t dataSize
	int32_t repeatCount)

Use SEI channel to send custom message to all users in room

This API allows you to use TRTC's SEI channel to broadcast custom data to other users in the current room for signaling transfer.

The header of a video frame has a header data block called SEI. This API works by embedding the custom signaling data you want to send in the SEI block and sending it together with the video frame.
Therefore, the SEI channel has a better compatibility than sendCustomCmdMsg as the signaling data can be transferred to the CSS CDN along with the video frame.
However, because the data block of the video frame header cannot be too large, we recommend you limit the size of the signaling data to only a few bytes when using this API.

The most common use is to embed the custom timestamp into video frames through this API so as to implement a perfect alignment between the message and video image (such as between the teaching material and video signal in the education scenario).
Other users in the room can receive the message through the `onRecvSEIMsg` callback in ITRTCCloudCallback.

Param	DESC
data	Data to be sent, which can be up to 1 KB (1,000 bytes)
repeatCount	Data sending count

Note

This API has the following restrictions:

1. The data will not be instantly sent after this API is called; instead, it will be inserted into the next video frame after the API call.
2. Up to 30 messages can be sent per second to all users in the room (this limit is shared with `sendCustomCmdMsg`).
3. Each packet can be up to 1 KB (this limit is shared with `sendCustomCmdMsg`). If a large amount of data is sent, the video bitrate will increase, which may reduce the video quality or even cause lagging.
4. Each client can send up to 8 KB of data in total per second (this limit is shared with `sendCustomCmdMsg`).
5. If multiple times of sending is required (i.e., `repeatCount` > 1), the data will be inserted into subsequent `repeatCount` video frames in a row for sending, which will increase the video bitrate.
6. If `repeatCount` is greater than 1, the data will be sent for multiple times, and the same message may be received multiple times in the `onRecvSEIMsg` callback; therefore, deduplication is required.

Return Desc:

true: the message is allowed and will be sent with subsequent video frames; false: the message is not allowed to be sent

startSpeedTest

startSpeedTest

int startSpeedTest	(const TRTCSpeedTestParams& params)

Start network speed test (used before room entry)

Param	DESC
params	speed test options

Note

1. The speed measurement process will incur a small amount of basic service fees, See Purchase Guide > Base Services.
2. Please perform the Network speed test before room entry, because if performed after room entry, the test will affect the normal audio/video transfer, and its result will be inaccurate due to interference in the room.
3. Only one network speed test task is allowed to run at the same time.

Return Desc:

interface call result, <0: failure

stopSpeedTest

stopSpeedTest

Stop network speed test

getSDKVersion

getSDKVersion

Get SDK version information

setLogLevel

setLogLevel

void setLogLevel	(TRTCLogLevel level)

Set log output level

Param	DESC
level	For more information, please see TRTCLogLevel. Default value: TRTCLogLevelNone

setConsoleEnabled

setConsoleEnabled

void setConsoleEnabled	(bool enabled)

Enable/Disable console log printing

Param	DESC
enabled	Specify whether to enable it, which is disabled by default

setLogCompressEnabled

setLogCompressEnabled

void setLogCompressEnabled	(bool enabled)

Enable/Disable local log compression

If compression is enabled, the log size will significantly reduce, but logs can be read only after being decompressed by the Python script provided by Tencent Cloud.
If compression is disabled, logs will be stored in plaintext and can be read directly in Notepad, but will take up more storage capacity.

Param	DESC
enabled	Specify whether to enable it, which is enabled by default

setLogDirPath

setLogDirPath

void setLogDirPath	(const char* path)

Set local log storage path

You can use this API to change the default storage path of the SDK's local logs, which is as follows:
Windows: C:/Users/[username]/AppData/Roaming/liteav/log, i.e., under `%appdata%/liteav/log` .
iOS or macOS: under `sandbox Documents/log` .
Android: under `/app directory/files/log/liteav/` .

Param	DESC
path	Log storage path

Note

Please be sure to call this API before all other APIs and make sure that the directory you specify exists and your application has read/write permissions of the directory.

setLogCallback

setLogCallback

void setLogCallback	(ITRTCLogCallback* callback)

Set log callback

showDebugView

showDebugView

void showDebugView	(int showType)

Display dashboard

"Dashboard" is a semi-transparent floating layer for debugging information on top of the video rendering control. It is used to display audio/video information and event information to facilitate integration and debugging.

Param	DESC
showType	0: does not display; 1: displays lite edition (only with audio/video information); 2: displays full edition (with audio/video information and event information).

callExperimentalAPI

callExperimentalAPI

char* callExperimentalAPI	(const char *jsonStr)

Call experimental APIs

enablePayloadPrivateEncryption

enablePayloadPrivateEncryption

int enablePayloadPrivateEncryption	(bool enabled
	const TRTCPayloadPrivateEncryptionConfig& config)

Enable or disable private encryption of media streams

In scenarios with high security requirements, TRTC recommends that you call the enablePayloadPrivateEncryption method to enable private encryption of media streams before joining a room.
After the user exits the room, the SDK will automatically close the private encryption. To re-enable private encryption, you need to call this method before the user joins the room again.

Param	DESC
config	Configure the algorithm and key for private encryption of media streams, please see TRTCPayloadPrivateEncryptionConfig.
enabled	Whether to enable media stream private encryption.

Note

TRTC has built-in encryption for media streams before transmission. After private encryption of media streams is enabled, it will be re-encrypted with the key and initial vector you pass in.

Return Desc:

Interface call result, 0: Method call succeeded, -1: The incoming parameter is invalid, -2: Your subscription has expired. If you want to renew it, Please update to RTC Engine Pro Plans and fill out application form. Approval is required before use.

TRTCCloudCallback

Last updated：2024-06-06 15:26:15

Module: ITRTCCloudCallback @ TXLiteAVSDK

Function: event callback APIs for TRTC's video call feature

TRTCCloudCallback

ITRTCCloudCallback

FuncList	DESC
onError	Error event callback
onWarning	Warning event callback
onEnterRoom	Whether room entry is successful
onExitRoom	Room exit
onSwitchRole	Role switching
onSwitchRoom	Result of room switching
onConnectOtherRoom	Result of requesting cross-room call
onDisconnectOtherRoom	Result of ending cross-room call
onUpdateOtherRoomForwardMode	Result of changing the upstream capability of the cross-room anchor
onRemoteUserEnterRoom	A user entered the room
onRemoteUserLeaveRoom	A user exited the room
onUserVideoAvailable	A remote user published/unpublished primary stream video
onUserSubStreamAvailable	A remote user published/unpublished substream video
onUserAudioAvailable	A remote user published/unpublished audio

onFirstVideoFrame	The SDK started rendering the first video frame of the local or a remote user
onFirstAudioFrame	The SDK started playing the first audio frame of a remote user
onSendFirstLocalVideoFrame	The first local video frame was published
onSendFirstLocalAudioFrame	The first local audio frame was published
onRemoteVideoStatusUpdated	Change of remote video status
onRemoteAudioStatusUpdated	Change of remote audio status
onUserVideoSizeChanged	Change of remote video size
onNetworkQuality	Real-time network quality statistics
onStatistics	Real-time statistics on technical metrics
onSpeedTestResult	Callback of network speed test
onConnectionLost	The SDK was disconnected from the cloud
onTryToReconnect	The SDK is reconnecting to the cloud
onConnectionRecovery	The SDK is reconnected to the cloud
onCameraDidReady	The camera is ready
onMicDidReady	The mic is ready
onUserVoiceVolume	Volume
onDeviceChange	The status of a local device changed (for desktop OS only)
onAudioDeviceCaptureVolumeChanged	The capturing volume of the mic changed
onAudioDevicePlayoutVolumeChanged	The playback volume changed
onSystemAudioLoopbackError	Whether system audio capturing is enabled successfully (for macOS only)
onTestMicVolume	Volume during mic test
onTestSpeakerVolume	Volume during speaker test
onRecvCustomCmdMsg	Receipt of custom message
onMissCustomCmdMsg	Loss of custom message

onRecvSEIMsg	Receipt of SEI message
onStartPublishing	Started publishing to Tencent Cloud CSS CDN
onStopPublishing	Stopped publishing to Tencent Cloud CSS CDN
onStartPublishCDNStream	Started publishing to non-Tencent Cloud’s live streaming CDN
onStopPublishCDNStream	Stopped publishing to non-Tencent Cloud’s live streaming CDN
onSetMixTranscodingConfig	Set the layout and transcoding parameters for On-Cloud MixTranscoding
onStartPublishMediaStream	Callback for starting to publish
onUpdatePublishMediaStream	Callback for modifying publishing parameters
onStopPublishMediaStream	Callback for stopping publishing
onCdnStreamStateChanged	Callback for change of RTMP/RTMPS publishing status
onScreenCaptureStarted	Screen sharing started
onScreenCapturePaused	Screen sharing was paused
onScreenCaptureResumed	Screen sharing was resumed
onScreenCaptureStoped	Screen sharing stopped
onScreenCaptureCovered	The shared window was covered (for Windows only)
onLocalRecordBegin	Local recording started
onLocalRecording	Local media is being recorded
onLocalRecordFragment	Record fragment finished.
onLocalRecordComplete	Local recording stopped
onSnapshotComplete	Finished taking a local screenshot
onUserEnter	An anchor entered the room (disused)
onUserExit	An anchor left the room (disused)
onAudioEffectFinished	Audio effects ended (disused)
onPlayBGMBegin	Started playing background music (disused)
onPlayBGMProgress	Playback progress of background music (disused)

onPlayBGMComplete	Background music stopped (disused)
onSpeedTest	Result of server speed testing (disused)

ITRTCVideoRenderCallback

FuncList	DESC
onRenderVideoFrame	Custom video rendering

ITRTCVideoFrameCallback

FuncList	DESC
onGLContextCreated	An OpenGL context was created in the SDK.
onProcessVideoFrame	Video processing by third-party beauty filters
onGLContextDestroy	The OpenGL context in the SDK was destroyed

ITRTCAudioFrameCallback

FuncList	DESC
onCapturedAudioFrame	Audio data captured by the local mic and pre-processed by the audio module
onLocalProcessedAudioFrame	Audio data captured by the local mic, pre-processed by the audio module, effect-processed and BGM-mixed
onPlayAudioFrame	Audio data of each remote user before audio mixing
onMixedPlayAudioFrame	Data mixed from each channel before being submitted to the system for playback
onMixedAllAudioFrame	Data mixed from all the captured and to-be-played audio in the SDK

ITRTCLogCallback

FuncList	DESC
onLog	Printing of local log

onError

onError

void onError	(TXLiteAVError errCode
	const char* errMsg
	void* extraInfo)

Error event callback

Error event, which indicates that the SDK threw an irrecoverable error such as room entry failure or failure to start device

For more information, see Error Codes.

Param	DESC
errCode	Error code
errMsg	Error message
extInfo	Extended field. Certain error codes may carry extra information for troubleshooting.

onWarning

onWarning

void onWarning	(TXLiteAVWarning warningCode
	const char* warningMsg
	void* extraInfo)

Warning event callback

Warning event, which indicates that the SDK threw an error requiring attention, such as video lag or high CPU usage

For more information, see Error Codes.

Param	DESC

extInfo	Extended field. Certain warning codes may carry extra information for troubleshooting.
warningCode	Warning code
warningMsg	Warning message

onEnterRoom

onEnterRoom

void onEnterRoom	(int result)

Whether room entry is successful

After calling the `enterRoom()` API in `TRTCCloud` to enter a room, you will receive the `onEnterRoom(result)` callback from `TRTCCloudDelegate`.

If room entry succeeded, `result` will be a positive number (`result` > 0), indicating the time in milliseconds (ms) the room entry takes.

If room entry failed, `result` will be a negative number (result < 0), indicating the error code for the failure.

For more information on the error codes for room entry failure, see Error Codes.

Param	DESC
result	If `result` is greater than 0, it indicates the time (in ms) the room entry takes; if `result` is less than 0, it represents the error code for room entry.

Note

1. In TRTC versions below 6.6, the `onEnterRoom(result)` callback is returned only if room entry succeeds, and the `onError()` callback is returned if room entry fails.
2. In TRTC 6.6 and above, the `onEnterRoom(result)` callback is returned regardless of whether room entry succeeds or fails, and the `onError()` callback is also returned if room entry fails.

onExitRoom

onExitRoom

void onExitRoom	(int reason)

Room exit

Calling the `exitRoom()` API in `TRTCCloud` will trigger the execution of room exit-related logic, such as releasing resources of audio/video devices and codecs.
After all resources occupied by the SDK are released, the SDK will return the `onExitRoom()` callback.

If you need to call `enterRoom()` again or switch to another audio/video SDK, please wait until you receive the `onExitRoom()` callback.
Otherwise, you may encounter problems such as the camera or mic being occupied.

Param	DESC
reason	Reason for room exit. `0` : the user called `exitRoom` to exit the room; `1` : the user was removed from the room by the server; `2` : the room was dismissed.

onSwitchRole

onSwitchRole

void onSwitchRole	(TXLiteAVError errCode
	const char* errMsg)

Role switching

You can call the `switchRole()` API in `TRTCCloud` to switch between the anchor and audience roles.
This is accompanied by a line switching process.
After the switching, the SDK will return the `onSwitchRole()` event callback.

Param	DESC
errCode	Error code. `ERR_NULL` indicates a successful switch. For more information, please see Error Codes.
errMsg	Error message

onSwitchRoom

onSwitchRoom

void onSwitchRoom	(TXLiteAVError errCode
	const char* errMsg)

Result of room switching

You can call the `switchRoom()` API in `TRTCCloud` to switch from one room to another.
After the switching, the SDK will return the `onSwitchRoom()` event callback.

Param	DESC
errCode	Error code. `ERR_NULL` indicates a successful switch. For more information, please see Error Codes.
errMsg	Error message

onConnectOtherRoom

onConnectOtherRoom

void onConnectOtherRoom	(const char* userId
	TXLiteAVError errCode
	const char* errMsg)

Result of requesting cross-room call

You can call the `connectOtherRoom()` API in `TRTCCloud` to establish a video call with the anchor of another room. This is the “anchor competition” feature.
The caller will receive the `onConnectOtherRoom()` callback, which can be used to determine whether the cross-room call is successful.
If it is successful, all users in either room will receive the `onUserVideoAvailable()` callback from the anchor of the other room.

Param	DESC
errCode	Error code. `ERR_NULL` indicates that cross-room connection is established successfully. For more information, please see Error Codes.
errMsg	Error message
userId	The user ID of the anchor (in another room) to be called

onDisconnectOtherRoom

onDisconnectOtherRoom

void onDisconnectOtherRoom	(TXLiteAVError errCode
	const char* errMsg)

Result of ending cross-room call

onUpdateOtherRoomForwardMode

onUpdateOtherRoomForwardMode

void onUpdateOtherRoomForwardMode	(TXLiteAVError errCode
	const char* errMsg)

Result of changing the upstream capability of the cross-room anchor

onRemoteUserEnterRoom

onRemoteUserEnterRoom

void onRemoteUserEnterRoom	(const char* userId)

A user entered the room

Due to performance concerns, this callback works differently in different scenarios (i.e., `AppScene` , which you can specify by setting the second parameter when calling `enterRoom`).

Live streaming scenarios (`TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom`): in live streaming scenarios, a user is either in the role of an anchor or audience. The callback is returned only when an anchor enters the room.

Call scenarios (`TRTCAppSceneVideoCall` or `TRTCAppSceneAudioCall`): in call scenarios, the concept of roles does not apply (all users can be considered as anchors), and the callback is returned when any user enters the room.

Param	DESC
userId	User ID of the remote user

Note

1. The `onRemoteUserEnterRoom` callback indicates that a user entered the room, but it does not necessarily mean that the user enabled audio or video.

2. If you want to know whether a user enabled video, we recommend you use the `onUserVideoAvailable()` callback.

onRemoteUserLeaveRoom

onRemoteUserLeaveRoom

void onRemoteUserLeaveRoom	(const char* userId
	int reason)

A user exited the room

As with `onRemoteUserEnterRoom` , this callback works differently in different scenarios (i.e., `AppScene` , which you can specify by setting the second parameter when calling `enterRoom`).
Live streaming scenarios (`TRTCAppSceneLIVE` or `TRTCAppSceneVoiceChatRoom`): the callback is triggered only when an anchor exits the room.
Call scenarios (`TRTCAppSceneVideoCall` or `TRTCAppSceneAudioCall`): in call scenarios, the concept of roles does not apply, and the callback is returned when any user exits the room.

Param	DESC
reason	Reason for room exit. `0` : the user exited the room voluntarily; `1` : the user exited the room due to timeout; `2` : the user was removed from the room; `3` : the anchor user exited the room due to switch to audience.
userId	User ID of the remote user

onUserVideoAvailable

onUserVideoAvailable

void onUserVideoAvailable	(const char* userId
	bool available)

A remote user published/unpublished primary stream video

The primary stream is usually used for camera images. If you receive the `onUserVideoAvailable(userId, true)` callback, it indicates that the user has available primary stream video.

You can then call startRemoteView to subscribe to the remote user's video. If the subscription is successful, you will receive the `onFirstVideoFrame(userid)` callback, which indicates that the first video frame of the user is rendered.

If you receive the `onUserVideoAvailable(userId, false)` callback, it indicates that the video of the remote user is disabled, which may be because the user called muteLocalVideo or stopLocalPreview.

Param	DESC
available	Whether the user published (or unpublished) primary stream video. `true` : published; `false` : unpublished
userId	User ID of the remote user

onUserSubStreamAvailable

onUserSubStreamAvailable

void onUserSubStreamAvailable	(const char* userId
	bool available)

A remote user published/unpublished substream video

The substream is usually used for screen sharing images. If you receive the `onUserSubStreamAvailable(userId, true)` callback, it indicates that the user has available substream video.

You can then call startRemoteView to subscribe to the remote user's video. If the subscription is successful, you will receive the `onFirstVideoFrame(userid)` callback, which indicates that the first frame of the user is rendered.

Param	DESC
available	Whether the user published (or unpublished) substream video. `true` : published; `false` : unpublished
userId	User ID of the remote user

Note

The API used to display substream images is startRemoteView, not startRemoteSubStreamView, startRemoteSubStreamView is deprecated.

onUserAudioAvailable

onUserAudioAvailable

void onUserAudioAvailable	(const char* userId
	bool available)

A remote user published/unpublished audio

If you receive the `onUserAudioAvailable(userId, true)` callback, it indicates that the user published audio.

In auto-subscription mode, the SDK will play the user's audio automatically.

In manual subscription mode, you can call muteRemoteAudio(userid, false) to play the user's audio.

Param	DESC
available	Whether the user published (or unpublished) audio. `true` : published; `false` : unpublished
userId	User ID of the remote user

Note

The auto-subscription mode is used by default. You can switch to the manual subscription mode by calling setDefaultStreamRecvMode, but it must be called before room entry for the switch to take effect.

onFirstVideoFrame

onFirstVideoFrame

void onFirstVideoFrame	(const char* userId
	const TRTCVideoStreamType streamType
	const int width
	const int height)

The SDK started rendering the first video frame of the local or a remote user

The SDK returns this event callback when it starts rendering your first video frame or that of a remote user. The `userId` in the callback can help you determine whether the frame is yours or a remote user's.

If `userId` is empty, it indicates that the SDK has started rendering your first video frame. The precondition is that you have called startLocalPreview or startScreenCapture.

If `userId` is not empty, it indicates that the SDK has started rendering the first video frame of a remote user. The precondition is that you have called startRemoteView to subscribe to the user's video.

Param	DESC
height	Video height
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	The user ID of the local or a remote user. If it is empty, it indicates that the first local video frame is available; if it is not empty, it indicates that the first video frame of a remote user is available.
width	Video width

Note

1. The callback of the first local video frame being rendered is triggered only after you call startLocalPreview or startScreenCapture.
2. The callback of the first video frame of a remote user being rendered is triggered only after you call startRemoteView or startRemoteSubStreamView.

onFirstAudioFrame

onFirstAudioFrame

void onFirstAudioFrame	(const char* userId)

The SDK started playing the first audio frame of a remote user

The SDK returns this callback when it plays the first audio frame of a remote user. The callback is not returned for the playing of the first audio frame of the local user.

Param	DESC
userId	User ID of the remote user

onSendFirstLocalVideoFrame

onSendFirstLocalVideoFrame

void onSendFirstLocalVideoFrame	(const TRTCVideoStreamType streamType)

The first local video frame was published

After you enter a room and call startLocalPreview or startScreenCapture to enable local video capturing (whichever happens first),

the SDK will start video encoding and publish the local video data via its network module to the cloud.

It returns the `onSendFirstLocalVideoFrame` callback after publishing the first local video frame.

Param	DESC
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.

onSendFirstLocalAudioFrame

onSendFirstLocalAudioFrame

The first local audio frame was published

After you enter a room and call startLocalAudio to enable audio capturing (whichever happens first),

the SDK will start audio encoding and publish the local audio data via its network module to the cloud.

The SDK returns the `onSendFirstLocalAudioFrame` callback after sending the first local audio frame.

onRemoteVideoStatusUpdated

onRemoteVideoStatusUpdated

void onRemoteVideoStatusUpdated	(const char* userId
	TRTCVideoStreamType streamType
	TRTCAVStatusType status
	TRTCAVStatusChangeReason reason
	void *extrainfo)

Change of remote video status

You can use this callback to get the status (`Playing` , `Loading` , or `Stopped`) of the video of each remote user and display it on the UI.

Param	DESC
extraInfo	Extra information
reason	Reason for the change of status
status	Video status, which may be `Playing` , `Loading` , or `Stopped`
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	User ID

onRemoteAudioStatusUpdated

onRemoteAudioStatusUpdated

void onRemoteAudioStatusUpdated	(const char* userId
	TRTCAVStatusType status
	TRTCAVStatusChangeReason reason
	void *extrainfo)

Change of remote audio status

You can use this callback to get the status (`Playing` , `Loading` , or `Stopped`) of the audio of each remote user and display it on the UI.

Param	DESC
extraInfo	Extra information
reason	Reason for the change of status
status	Audio status, which may be `Playing` , `Loading` , or `Stopped`
userId	User ID

onUserVideoSizeChanged

onUserVideoSizeChanged

void onUserVideoSizeChanged	(const char* userId
	TRTCVideoStreamType streamType
	int newWidth
	int newHeight)

Change of remote video size

If you receive the `onUserVideoSizeChanged(userId, streamtype, newWidth, newHeight)` callback, it indicates that the user changed the video size. It may be triggered by `setVideoEncoderParam` or `setSubStreamEncoderParam`.

Param	DESC
newHeight	Video height
newWidth	Video width
streamType	Video stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	User ID

onNetworkQuality

onNetworkQuality

void onNetworkQuality	(TRTCQualityInfo localQuality
	TRTCQualityInfo* remoteQuality
	uint32_t remoteQualityCount)

Real-time network quality statistics

This callback is returned every 2 seconds and notifies you of the upstream and downstream network quality detected by the SDK.
The SDK uses a built-in proprietary algorithm to assess the current latency, bandwidth, and stability of the network and returns a result.

If the result is `1` (excellent), it means that the current network conditions are excellent; if it is `6` (down), it means that the current network conditions are too bad to support TRTC calls.

Param	DESC
localQuality	Upstream network quality
remoteQuality	Downstream network quality, it refers to the data quality finally measured on the local side after the data flow passes through a complete transmission link of "remote ->cloud ->local". Therefore, the downlink network quality here represents the joint impact of the remote uplink and the local downlink.

Note

The uplink quality of remote users cannot be determined independently through this interface.

onStatistics

onStatistics

void onStatistics	(const TRTCStatistics& statistics)

Real-time statistics on technical metrics

This callback is returned every 2 seconds and notifies you of the statistics on technical metrics related to video, audio, and network. The metrics are listed in TRTCStatistics:

Video statistics: video resolution (`resolution`), frame rate (`FPS`), bitrate (`bitrate`), etc.

Audio statistics: audio sample rate (`samplerate`), number of audio channels (`channel`), bitrate (`bitrate`), etc.

Network statistics: the round trip time (`rtt`) between the SDK and the cloud (SDK -> Cloud -> SDK), package loss rate (`loss`), upstream traffic (`sentBytes`), downstream traffic (`receivedBytes`), etc.

Param	DESC
statistics	Statistics, including local statistics and the statistics of remote users. For details, please see TRTCStatistics.

Note

If you want to learn about only the current network quality and do not want to spend much time analyzing the statistics returned by this callback, we recommend you use onNetworkQuality.

onSpeedTestResult

onSpeedTestResult

void onSpeedTestResult	(const TRTCSpeedTestResult& result)

Callback of network speed test

The callback is triggered by startSpeedTest:.

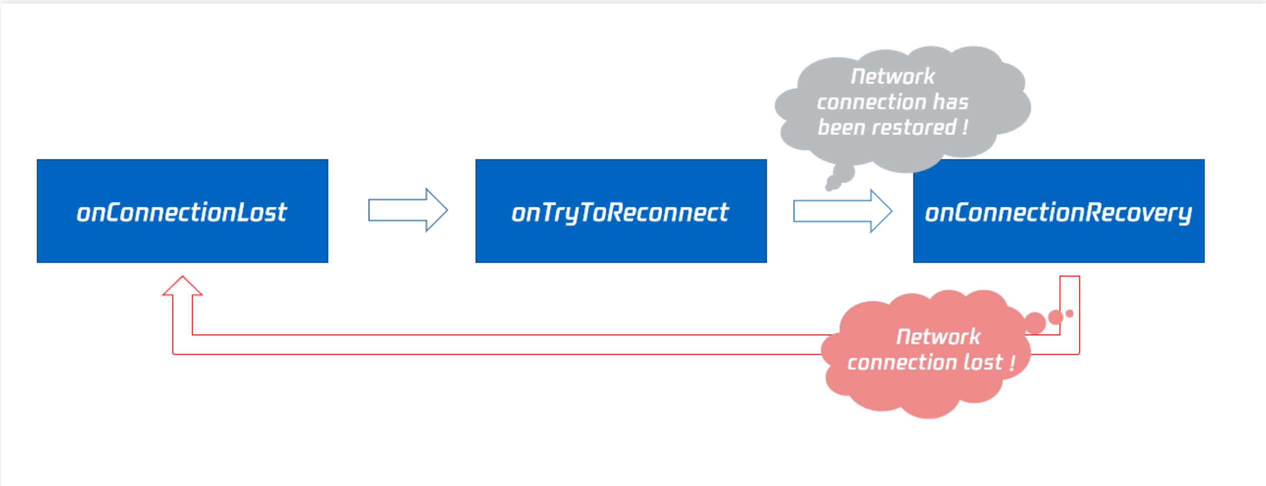
Param	DESC
result	Speed test data, including loss rates, rtt and bandwidth rates, please refer to TRTCSpeedTestResult for details.

onConnectionLost

onConnectionLost

The SDK was disconnected from the cloud

The SDK returns this callback when it is disconnected from the cloud, which may be caused by network unavailability or change of network, for example, when the user walks into an elevator.
After returning this callback, the SDK will attempt to reconnect to the cloud, and will return the onTryToReconnect callback. When it is reconnected, it will return the onConnectionRecovery callback.
In other words, the SDK proceeds from one event to the next in the following order:



onTryToReconnect

onTryToReconnect

The SDK is reconnecting to the cloud

When the SDK is disconnected from the cloud, it returns the onConnectionLost callback. It then attempts to reconnect and returns this callback (onTryToReconnect). After it is reconnected, it returns the onConnectionRecovery callback.

onConnectionRecovery

onConnectionRecovery

The SDK is reconnected to the cloud

When the SDK is disconnected from the cloud, it returns the onConnectionLost callback. It then attempts to reconnect and returns the onTryToReconnect callback. After it is reconnected, it returns this callback (onConnectionRecovery).

onCameraDidReady

onCameraDidReady

The camera is ready

After you call startLocalPreivew, the SDK will try to start the camera and return this callback if the camera is started.
If it fails to start the camera, it’s probably because the application does not have access to the camera or the camera is being used.
You can capture the onError callback to learn about the exception and let users know via UI messages.

onMicDidReady

onMicDidReady

The mic is ready

After you call startLocalAudio, the SDK will try to start the mic and return this callback if the mic is started.
If it fails to start the mic, it’s probably because the application does not have access to the mic or the mic is being used.
You can capture the onError callback to learn about the exception and let users know via UI messages.

onUserVoiceVolume

onUserVoiceVolume

void onUserVoiceVolume	(TRTCVolumeInfo* userVolumes
	uint32_t userVolumesCount
	uint32_t totalVolume)

Volume

The SDK can assess the volume of each channel and return this callback on a regular basis. You can display, for example, a waveform or volume bar on the UI based on the statistics returned.

You need to first call enableAudioVolumeEvaluation to enable the feature and set the interval for the callback.

Note that the SDK returns this callback at the specified interval regardless of whether someone is speaking in the room.

Param	DESC
totalVolume	The total volume of all remote users. Value range: 0-100
userVolumes	An array that represents the volume of all users who are speaking in the room. Value range: 0-100

Note

`userVolumes` is an array. If `userId` is empty, the elements in the array represent the volume of the local user’s audio. Otherwise, they represent the volume of a remote user’s audio.

onDeviceChange

onDeviceChange

void onDeviceChange	(const char* deviceId
	TRTCDeviceType type
	TRTCDeviceState state)

The status of a local device changed (for desktop OS only)

The SDK returns this callback when a local device (camera, mic, or speaker) is connected or disconnected.

Param	DESC

deviceId	Device ID
deviceType	Device type
state	Device status. `0` : connected; `1` : disconnected; `2` : started

onAudioDeviceCaptureVolumeChanged

onAudioDeviceCaptureVolumeChanged

void onAudioDeviceCaptureVolumeChanged	(uint32_t volume
	bool muted)

The capturing volume of the mic changed

On desktop OS such as macOS and Windows, users can set the capturing volume of the mic in the audio control panel.

The higher volume a user sets, the higher the volume of raw audio captured by the mic.

On some keyboards and laptops, users can also mute the mic by pressing a key (whose icon is a crossed out mic).

When users set the mic capturing volume via the UI or a keyboard shortcut, the SDK will return this callback.

Param	DESC
muted	Whether the mic is muted. `true` : muted; `false` : unmuted
volume	System audio capturing volume, which users can set in the audio control panel. Value range: 0-100

Note

You need to call enableAudioVolumeEvaluation and set the callback interval (`interval` > 0) to enable the callback. To disable the callback, set `interval` to `0` .

onAudioDevicePlayoutVolumeChanged

onAudioDevicePlayoutVolumeChanged

void onAudioDevicePlayoutVolumeChanged	(uint32_t volume
	bool muted)

The playback volume changed

On desktop OS such as macOS and Windows, users can set the system's playback volume in the audio control panel. On some keyboards and laptops, users can also mute the speaker by pressing a key (whose icon is a crossed out speaker).

When users set the system's playback volume via the UI or a keyboard shortcut, the SDK will return this callback.

Param	DESC
muted	Whether the speaker is muted. `true` : muted; `false` : unmuted
volume	The system playback volume, which users can set in the audio control panel. Value range: 0-100

Note

You need to call enableAudioVolumeEvaluation and set the callback interval (`interval` > 0) to enable the callback. To disable the callback, set `interval` to `0` .

onSystemAudioLoopbackError

onSystemAudioLoopbackError

void onSystemAudioLoopbackError	(TXLiteAVError errCode)

Whether system audio capturing is enabled successfully (for macOS only)

On macOS, you can call startSystemAudioLoopback to install an audio driver and have the SDK capture the audio played back by the system.

In use cases such as video teaching and music live streaming, the teacher can use this feature to let the SDK capture the sound of the video played by his or her computer, so that students in the room can hear the sound too.

The SDK returns this callback after trying to enable system audio capturing. To determine whether it is actually enabled, pay attention to the error parameter in the callback.

Param	DESC
err	If it is `ERR_NULL` , system audio capturing is enabled successfully. Otherwise, it is not.

onTestMicVolume

onTestMicVolume

void onTestMicVolume	(uint32_t volume)

Volume during mic test

When you call startMicDeviceTest to test the mic, the SDK will keep returning this callback. The `volume` parameter represents the volume of the audio captured by the mic.

If the value of the `volume` parameter fluctuates, the mic works properly. If it is `0` throughout the test, it indicates that there is a problem with the mic, and users should be prompted to switch to a different mic.

Param	DESC
volume	Captured mic volume. Value range: 0-100

onTestSpeakerVolume

onTestSpeakerVolume

void onTestSpeakerVolume	(uint32_t volume)

Volume during speaker test

When you call startSpeakerDeviceTest to test the speaker, the SDK will keep returning this callback.

The `volume` parameter in the callback represents the volume of audio sent by the SDK to the speaker for playback. If its value fluctuates but users cannot hear any sound, the speaker is not working properly.

Param	DESC
volume	The volume of audio sent by the SDK to the speaker for playback. Value range: 0-100

onRecvCustomCmdMsg

onRecvCustomCmdMsg

void onRecvCustomCmdMsg	(const char* userId
	int32_t cmdID
	uint32_t seq
	const uint8_t* message
	uint32_t messageSize)

Receipt of custom message

When a user in a room uses sendCustomCmdMsg to send a custom message, other users in the room can receive the message through the `onRecvCustomCmdMsg` callback.

Param	DESC
cmdID	Command ID
message	Message data
seq	Message serial number
userId	User ID

onMissCustomCmdMsg

onMissCustomCmdMsg

void onMissCustomCmdMsg	(const char* userId
	int32_t cmdID
	int32_t errCode
	int32_t missed)

Loss of custom message

When you use sendCustomCmdMsg to send a custom UDP message, even if you enable reliable transfer (by setting `reliable` to `true`), there is still a chance of message loss. Reliable transfer only helps maintain a low probability of message loss, which meets the reliability requirements in most cases.
If the sender sets `reliable` to `true` , the SDK will use this callback to notify the recipient of the number of custom messages lost during a specified time period (usually 5s) in the past.

Param	DESC
cmdID	Command ID
errCode	Error code
missed	Number of lost messages
userId	User ID

Note

The recipient receives this callback only if the sender sets `reliable` to `true` .

onRecvSEIMsg

onRecvSEIMsg

void onRecvSEIMsg	(const char* userId
	const uint8_t* message
	uint32_t messageSize)

Receipt of SEI message

If a user in the room uses sendSEIMsg to send an SEI message via video frames, other users in the room can receive the message through the `onRecvSEIMsg` callback.

Param	DESC
message	Data
userId	User ID

onStartPublishing

onStartPublishing

void onStartPublishing	(int err
	const char *errMsg)

Started publishing to Tencent Cloud CSS CDN

When you call startPublishing to publish streams to Tencent Cloud CSS CDN, the SDK will sync the command to the CVM immediately.

The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onStopPublishing

onStopPublishing

void onStopPublishing	(int err
	const char *errMsg)

Stopped publishing to Tencent Cloud CSS CDN

When you call stopPublishing to stop publishing streams to Tencent Cloud CSS CDN, the SDK will sync the command to the CVM immediately.

The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onStartPublishCDNStream

onStartPublishCDNStream

void onStartPublishCDNStream	(int errCode
	const char* errMsg)

Started publishing to non-Tencent Cloud's live streaming CDN

When you call startPublishCDNStream to start publishing streams to a non-Tencent Cloud's live streaming CDN, the SDK will sync the command to the CVM immediately.

The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

Note

If you receive a callback that the command is executed successfully, it only means that your command was sent to Tencent Cloud's backend server. If the CDN vendor does not accept your streams, the publishing will still fail.

onStopPublishCDNStream

onStopPublishCDNStream

void onStopPublishCDNStream	(int errCode
	const char* errMsg)

Stopped publishing to non-Tencent Cloud’s live streaming CDN

When you call stopPublishCDNStream to stop publishing to a non-Tencent Cloud’s live streaming CDN, the SDK will sync the command to the CVM immediately.

The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onSetMixTranscodingConfig

onSetMixTranscodingConfig

void onSetMixTranscodingConfig	(int err
	const char* errMsg)

Set the layout and transcoding parameters for On-Cloud MixTranscoding

When you call setMixTranscodingConfig to modify the layout and transcoding parameters for On-Cloud MixTranscoding, the SDK will sync the command to the CVM immediately.

The SDK will then receive the execution result from the CVM and return the result to you via this callback.

Param	DESC
err	`0` : successful; other values: failed
errMsg	Error message

onStartPublishMediaStream

onStartPublishMediaStream

void onStartPublishMediaStream	(const char* taskId
	int code
	const char* message
	void* extraInfo)

Callback for starting to publish

When you call startPublishMediaStream to publish a stream to the TRTC backend, the SDK will immediately update the command to the cloud server.

The SDK will then receive the publishing result from the cloud server and will send the result to you via this callback.

Param	DESC
code	: `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The callback information.
taskId	: If a request is successful, a task ID will be returned via the callback. You need to provide this task ID when you call updatePublishMediaStream to modify publishing parameters or stopPublishMediaStream to stop publishing.

onUpdatePublishMediaStream

onUpdatePublishMediaStream

void onUpdatePublishMediaStream	(const char* taskId
	int code
	const char* message
	void* extraInfo)

Callback for modifying publishing parameters

When you call updatePublishMediaStream to modify publishing parameters, the SDK will immediately update the command to the cloud server.

The SDK will then receive the modification result from the cloud server and will send the result to you via this callback.

Param	DESC
code	: `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The callback information.
taskId	: The task ID you pass in when calling updatePublishMediaStream, which is used to identify a request.

onStopPublishMediaStream

onStopPublishMediaStream

void onStopPublishMediaStream	(const char* taskId
	int code
	const char* message
	void* extraInfo)

Callback for stopping publishing

When you call stopPublishMediaStream to stop publishing, the SDK will immediately update the command to the cloud server.

The SDK will then receive the modification result from the cloud server and will send the result to you via this callback.

Param	DESC
code	: `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The callback information.
taskId	: The task ID you pass in when calling stopPublishMediaStream, which is used to identify a request.

onCdnStreamStateChanged

onCdnStreamStateChanged

void onCdnStreamStateChanged	(const char* cdnUrl
	int status
	int code
	const char* msg
	void* extraInfo)

Callback for change of RTMP/RTMPS publishing status

When you call startPublishMediaStream to publish a stream to the TRTC backend, the SDK will immediately update the command to the cloud server.

If you set the publishing destination (TRTCPublishTarget) to the URL of Tencent Cloud or a third-party CDN, you will be notified of the RTMP/RTMPS publishing status via this callback.

Param	DESC
cdnUrl	: The URL you specify in TRTCPublishTarget when you call startPublishMediaStream.
code	: The publishing result. `0` : Successful; other values: Failed.
extraInfo	: Additional information. For some error codes, there may be additional information to help you troubleshoot the issues.
message	: The publishing information.
status	: The publishing status. 0: The publishing has not started yet or has ended. This value will be returned after you call stopPublishMediaStream. 1: The TRTC server is connecting to the CDN server. If the first attempt fails, the TRTC backend will retry multiple times and will return this value via the callback (every five seconds). After publishing succeeds, the value `2` will be returned. If a server error occurs or publishing is still unsuccessful after 60 seconds, the value `4` will be returned. 2: The TRTC server is publishing to the CDN. This value will be returned if the publishing succeeds. 3: The TRTC server is disconnected from the CDN server and is reconnecting. If a CDN error occurs or publishing is interrupted, the TRTC backend will try to reconnect and resume publishing and will return this value via the callback (every five seconds). After publishing resumes, the value `2` will be returned. If a server error occurs or the attempt to resume publishing is still unsuccessful after 60 seconds, the value `4` will be returned.

	4: The TRTC server is disconnected from the CDN server and failed to reconnect within the timeout period. In this case, the publishing is deemed to have failed. You can call updatePublishMediaStream to try again. 5: The TRTC server is disconnecting from the CDN server. After you call stopPublishMediaStream, the SDK will return this value first and then the value `0`.

onScreenCaptureStarted

onScreenCaptureStarted

Screen sharing started

The SDK returns this callback when you call startScreenCapture and other APIs to start screen sharing.

onScreenCapturePaused

onScreenCapturePaused

void onScreenCapturePaused	(int reason)

Screen sharing was paused

The SDK returns this callback when you call pauseScreenCapture to pause screen sharing.

Param	DESC
reason	Reason. `0` : the user paused screen sharing. `1` : screen sharing was paused because the shared window became invisible(Mac). screen sharing was paused because setting parameters(Windows). `2` : screen sharing was paused because the shared window became minimum(only for Windows). `3` : screen sharing was paused because the shared window became invisible(only for Windows).

onScreenCaptureResumed

onScreenCaptureResumed

void onScreenCaptureResumed	(int reason)

Screen sharing was resumed

The SDK returns this callback when you call resumeScreenCapture to resume screen sharing.

Param	DESC
reason	Reason. `0` : the user resumed screen sharing. `1` : screen sharing was resumed automatically after the shared window became visible again(Mac). screen sharing was resumed automatically after setting parameters(Windows). `2` : screen sharing was resumed automatically after the shared window became minimize recovery(only for Windows). `3` : screen sharing was resumed automatically after the shared window became visible again(only for Windows).

onScreenCaptureStoped

onScreenCaptureStoped

void onScreenCaptureStoped	(int reason)

Screen sharing stopped

The SDK returns this callback when you call stopScreenCapture to stop screen sharing.

Param	DESC
reason	Reason. `0` : the user stopped screen sharing; `1` : screen sharing stopped because the shared window was closed.

onScreenCaptureCovered

onScreenCaptureCovered

The shared window was covered (for Windows only)

The SDK returns this callback when the shared window is covered and cannot be captured. Upon receiving this callback, you can prompt users via the UI to move and expose the window.

onLocalRecordBegin

onLocalRecordBegin

void onLocalRecordBegin	(int errCode
	const char* storagePath)

Local recording started

When you call startLocalRecording to start local recording, the SDK returns this callback to notify you whether recording is started successfully.

Param	DESC
errCode	status. 0: successful. -1: failed. -2: unsupported format. -6: recording has been started. Stop recording first. -7: recording file already exists and needs to be deleted. -8: recording directory does not have the write permission. Please check the directory permission.
storagePath	Storage path of recording file

onLocalRecording

onLocalRecording

void onLocalRecording	(long duration
	const char* storagePath)

Local media is being recorded

The SDK returns this callback regularly after local recording is started successfully via the calling of startLocalRecording.

You can capture this callback to stay up to date with the status of the recording task.

You can set the callback interval when calling startLocalRecording.

Param	DESC
duration	Cumulative duration of recording, in milliseconds
storagePath	Storage path of recording file

onLocalRecordFragment

onLocalRecordFragment

void onLocalRecordFragment	(const char* storagePath)

Record fragment finished.

When fragment recording is enabled, this callback will be invoked when each fragment file is finished.

Param	DESC
storagePath	Storage path of the fragment.

onLocalRecordComplete

onLocalRecordComplete

void onLocalRecordComplete	(int errCode
	const char* storagePath)

Local recording stopped

When you call stopLocalRecording to stop local recording, the SDK returns this callback to notify you of the recording result.

Param	DESC
errCode	status 0: successful. -1: failed. -2: Switching resolution or horizontal and vertical screen causes the recording to stop. -3: recording duration is too short or no video or audio data is received. Check the recording duration or whether audio or video capture is enabled.
storagePath	Storage path of recording file

onSnapshotComplete

onSnapshotComplete

void onSnapshotComplete	(const char* userId

	TRTCVideoStreamType type
	char* data
	uint32_t length
	uint32_t width
	uint32_t height
	TRTCVideoPixelFormat format)

Finished taking a local screenshot

Param	DESC
bmp	Screenshot result. If it is `null` , the screenshot failed to be taken.
data	Screenshot data. If it is `nullptr` , it indicates that the SDK failed to take the screenshot.
format	Screenshot data format. Only `TRTCVideoPixelFormat_BGRA32` is supported now.
height	Screenshot height
length	Screenshot data length. In BGRA32 format, length = width * height * 4.
type	Video stream type
userId	User ID. If it is empty, the screenshot is a local image.
width	Screenshot width

Note

The parameters of the full-platform C++ interface and the Java interface are different. The C++ interface uses 7 parameters to describe a screenshot, while the Java interface uses only one Bitmap to describe a screenshot.

onUserEnter

onUserEnter

void onUserEnter	(const char* userId)

An anchor entered the room (disused)

@deprecated This callback is not recommended in the new version. Please use onRemoteUserEnterRoom instead.

onUserExit

onUserExit

void onUserExit	(const char* userId
	int reason)

An anchor left the room (disused)

@deprecated This callback is not recommended in the new version. Please use onRemoteUserLeaveRoom instead.

onAudioEffectFinished

onAudioEffectFinished

void onAudioEffectFinished	(int effectId
	int code)

Audio effects ended (disused)

@deprecated This callback is not recommended in the new version. Please use ITXAudioEffectManager instead. Audio effects and background music can be started using the same API (startPlayMusic) now instead of separate ones.

onPlayBGMBegin

onPlayBGMBegin

void onPlayBGMBegin	(TXLiteAVError errCode)

Started playing background music (disused)

@deprecated This callback is not recommended in the new version. Please use ITXMusicPlayObserver instead. Audio effects and background music can be started using the same API (startPlayMusic) now instead of separate ones.

onPlayBGMProgress

onPlayBGMProgress

void onPlayBGMProgress	(uint32_t progressMS
	uint32_t durationMS)

Playback progress of background music (disused)

@deprecated This callback is not recommended in the new version. Please use ITXMusicPlayObserver instead. Audio effects and background music can be started using the same API (startPlayMusic) now instead of separate ones.

onPlayBGMComplete

onPlayBGMComplete

void onPlayBGMComplete	(TXLiteAVError errCode)

Background music stopped (disused)

@deprecated This callback is not recommended in the new version. Please use ITXMusicPlayObserver instead. Audio effects and background music can be started using the same API (startPlayMusic) now instead of separate ones.

onSpeedTest

onSpeedTest

void onSpeedTest	(const TRTCSpeedTestResult& currentResult
	uint32_t finishedCount
	uint32_t totalCount)

Result of server speed testing (disused)

@deprecated This callback is not recommended in the new version. Please use onSpeedTestResult: instead.

onRenderVideoFrame

onRenderVideoFrame

void onRenderVideoFrame	(const char* userId
	TRTCVideoStreamType streamType
	TRTCVideoFrame* frame)

Custom video rendering

If you have configured the callback of custom rendering for local or remote video, the SDK will return to you via this callback video frames that are otherwise sent to the rendering control, so that you can customize rendering.

Param	DESC
frame	Video frames to be rendered
streamType	Stream type. The primary stream (`Main`) is usually used for camera images, and the substream (`Sub`) for screen sharing images.
userId	`userId` of the video source. This parameter can be ignored if the callback is for local video (`setLocalVideoRenderDelegate`).

onGLContextCreated

onGLContextCreated

An OpenGL context was created in the SDK.

onProcessVideoFrame

onProcessVideoFrame

int onProcessVideoFrame	(TRTCVideoFrame *srcFrame
	TRTCVideoFrame *dstFrame)

Video processing by third-party beauty filters

If you use a third-party beauty filter component, you need to configure this callback in `TRTCCloud` to have the SDK return to you video frames that are otherwise pre-processed by TRTC.
You can then send the video frames to the third-party beauty filter component for processing. As the data returned can be read and modified, the result of processing can be synced to TRTC for subsequent encoding and publishing.

Case 1: the beauty filter component generates new textures

If the beauty filter component you use generates a frame of new texture (for the processed image) during image processing, please set `dstFrame.textureId` to the ID of the new texture in the callback function.

```
int onProcessVideoFrame(TRTCVideoFrame * srcFrame, TRTCVideoFrame *dstFrame) {
    dstFrame->textureId = mFURenderer.onDrawFrameSingleInput(srcFrame->textureId);
    return 0;
}
```

Case 2: you need to provide target textures to the beauty filter component

If the third-party beauty filter component you use does not generate new textures and you need to manually set an input texture and an output texture for the component, you can consider the following scheme:

```
int onProcessVideoFrame(TRTCVideoFrame *srcFrame, TRTCVideoFrame *dstFrame) {
    thirdparty_process(srcFrame->textureId, srcFrame->width, srcFrame->height, dstF
    return 0;
}
```

Param	DESC
dstFrame	Used to receive video images processed by third-party beauty filters

srcFrame	Used to carry images captured by TRTC via the camera

Note

Currently, only the OpenGL texture scheme is supported(PC supports TRTCVideoBufferType_Buffer format Only)

onGLContextDestroy

onGLContextDestroy

The OpenGL context in the SDK was destroyed

onCapturedAudioFrame

onCapturedAudioFrame

void onCapturedAudioFrame	(TRTCAudioFrame *frame)

Audio data captured by the local mic and pre-processed by the audio module

After you configure the callback of custom audio processing, the SDK will return via this callback the data captured and pre-processed (ANS, AEC, and AGC) in PCM format.

The audio returned is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function. The SDK processes an audio frame every 20 ms, so if your operation takes more than 20 ms, it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.
3. The audio data is returned via this callback after ANS, AEC and AGC, but it **does not include** pre-processing effects like background music, audio effects, or reverb, and therefore has a short delay.

onLocalProcessedAudioFrame

onLocalProcessedAudioFrame

void onLocalProcessedAudioFrame	(TRTCAudioFrame *frame)

Audio data captured by the local mic, pre-processed by the audio module, effect-processed and BGM-mixed

After you configure the callback of custom audio processing, the SDK will return via this callback the data captured, pre-processed (ANS, AEC, and AGC), effect-processed and BGM-mixed in PCM format, before it is submitted to the network module for encoding.
The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.
The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.
Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Instructions:
You could write data to the `TRTCAudioFrame.extraData` filed, in order to achieve the purpose of transmitting signaling.
Because the data block of the audio frame header cannot be too large, we recommend you limit the size of the signaling data to only a few bytes when using this API. If extra data more than 100 bytes, it won't be sent.
Other users in the room can receive the message through the `TRTCAudioFrame.extraData` in `onRemoteUserAudioFrame` callback in TRTCAudioFrameDelegate.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function. The SDK processes an audio frame every 20 ms, so if your operation takes more than 20 ms, it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.
3. Audio data is returned via this callback after ANS, AEC, AGC, effect-processing and BGM-mixing, and therefore the delay is longer than that with onCapturedAudioFrame.

onPlayAudioFrame

onPlayAudioFrame

void onPlayAudioFrame	(TRTCAudioFrame *frame
	const char* userId)

Audio data of each remote user before audio mixing

After you configure the callback of custom audio processing, the SDK will return via this callback the raw audio data (PCM format) of each remote user before mixing.

The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format
userId	User ID

Note

The audio data returned via this callback can be read but not modified.

onMixedPlayAudioFrame

onMixedPlayAudioFrame

void onMixedPlayAudioFrame	(TRTCAudioFrame *frame)

Data mixed from each channel before being submitted to the system for playback

After you configure the callback of custom audio processing, the SDK will return to you via this callback the data (PCM format) mixed from each channel before it is submitted to the system for playback.

The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.

The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.

Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. Please avoid time-consuming operations in this callback function. The SDK processes an audio frame every 20 ms, so if your operation takes more than 20 ms, it will cause audio exceptions.
2. The audio data returned via this callback can be read and modified, but please keep the duration of your operation short.
3. The audio data returned via this callback is the audio data mixed from each channel before it is played. It does not include the in-ear monitoring data.

onMixedAllAudioFrame

onMixedAllAudioFrame

void onMixedAllAudioFrame	(TRTCAudioFrame *frame)

Data mixed from all the captured and to-be-played audio in the SDK

After you configure the callback of custom audio processing, the SDK will return via this callback the data (PCM format) mixed from all captured and to-be-played audio in the SDK, so that you can customize recording.
The audio data returned via this callback is in PCM format and has a fixed frame length (time) of 0.02s.
The formula to convert a frame length in seconds to one in bytes is **sample rate * frame length in seconds * number of sound channels * audio bit depth**.
Assume that the audio is recorded on a single channel with a sample rate of 48,000 Hz and audio bit depth of 16 bits, which are the default settings of TRTC. The frame length in bytes will be **48000 * 0.02s * 1 * 16 bits = 15360 bits = 1920 bytes**.

Param	DESC
frame	Audio frames in PCM format

Note

1. This data returned via this callback is mixed from all audio in the SDK, including local audio after pre-processing (ANS, AEC, and AGC), special effects application, and music mixing, as well as all remote audio, but it does not

include the in-ear monitoring data.
2. The audio data returned via this callback cannot be modified.

onLog

onLog

void onLog	(const char* log
	TRTCLogLevel level
	const char* module)

Printing of local log

If you want to capture the local log printing event, you can configure the log callback to have the SDK return to you via this callback all logs that are to be printed.

Param	DESC
level	Log level. For more information, please see `TRTC_LOG_LEVEL` .
log	Log content
module	Reserved field, which is not defined at the moment and has a fixed value of `TXLiteAVSDK` .

ITRTCStatistics

Last updated：2024-06-06 15:26:14

Module: TRTC audio/video metrics (read-only)

Function: the TRTC SDK reports to you the current real-time audio/video metrics (frame rate, bitrate, lag, etc.) once every two seconds

ITRTCStatistics

StructType

FuncList	DESC
TRTCLocalStatistics	Local audio/video metrics
TRTCRemoteStatistics	Remote audio/video metrics
TRTCStatistics	Network and performance metrics

TRTCLocalStatistics

TRTCLocalStatistics

Local audio/video metrics

EnumType	DESC
audioBitrate	Field description: local audio bitrate in Kbps, i.e., how much audio data is generated per second
audioCaptureState	Field description:Audio equipment collection status(0：Normal；1：Long silence detected；2：Broken sound detected；3：Abnormal intermittent sound detected;)
audioSampleRate	Field description: local audio sample rate (Hz)
frameRate	Field description: local video frame rate in fps, i.e., how many video frames there

	are per second
height	Field description: local video height in px
streamType	Field description: video stream type (HD big image | smooth small image | substream image)
videoBitrate	Field description: local video bitrate in Kbps, i.e., how much video data is generated per second
width	Field description: local video width in px

TRTCRemoteStatistics

TRTCRemoteStatistics

Remote audio/video metrics

EnumType	DESC
audioBitrate	Field description: local audio bitrate (Kbps)
audioBlockRate	Field description: audio playback lag rate (%) Audio playback lag rate (audioBlockRate) = cumulative audio playback lag duration (audioTotalBlockTime)/total audio playback duration
audioPacketLoss	Field description: total packet loss rate (%) of the audio stream `audioPacketLoss` represents the packet loss rate eventually calculated on the audience side after the audio/video stream goes through the complete transfer linkage of "anchor -> cloud -> audience". The smaller the `audioPacketLoss` , the better. The packet loss rate of 0 indicates that all data of the audio stream has entirely reached the audience. If `downLoss` is `0` but `audioPacketLoss` isn't, there is no packet loss on the linkage of "cloud -> audience" for the audiostream, but there are unrecoverable packet losses on the linkage of "anchor -> cloud".
audioSampleRate	Field description: local audio sample rate (Hz)
audioTotalBlockTime	Field description: cumulative audio playback lag duration (ms)
finalLoss	Field description: total packet loss rate (%) of the audio/video stream Deprecated, please use audioPacketLoss and videoPacketLoss instead.
frameRate	Field description: remote video frame rate (fps)

height	Field description: remote video height in px
jitterBufferDelay	Field description: playback delay (ms) In order to avoid audio/video lags caused by network jitters and network packet disorders, TRTC maintains a playback buffer on the playback side to organize the received network data packets. The size of the buffer is adaptively adjusted according to the current network quality and converted to the length of time in milliseconds, i.e., `jitterBufferDelay` .
point2PointDelay	Field description: end-to-end delay (ms) `point2PointDelay` represents the delay of "anchor -> cloud -> audience". To be more precise, it represents the delay of the entire linkage of "collection -> encoding -> network transfer -> receiving -> buffering -> decoding -> playback". `point2PointDelay` works only if both the local and remote SDKs are on version 8.5 or above. If the remote SDK is on a version below 8.5, this value will always be 0 and thus meaningless.
remoteNetworkRTT	Field description: round-trip delay (ms) from the SDK to cloud This value represents the total time it takes to send a network packet from the SDK to the cloud and then send a network packet back from the cloud to the SDK, i.e., the total time it takes for a network packet to go through the linkage of "SDK -> cloud -> SDK". The smaller the value, the better. If `remoteNetworkRTT` is below 50 ms, it means a short audio/video call delay; if `remoteNetworkRTT` is above 200 ms, it means a long audio/video call delay. It should be explained that `remoteNetworkRTT` represents the total time spent on the linkage of "SDK -> cloud -> SDK"; therefore, there is no need to distinguish between `remoteNetworkUpRTT` and `remoteNetworkDownRTT` .
remoteNetworkUplinkLoss	Field description: upstream packet loss rate (%) from the SDK to cloud The smaller the value, the better. If `remoteNetworkUplinkLoss` is `0%` , the upstream network quality is very good, and the data packets uploaded to the cloud are basically not lost. If `remoteNetworkUplinkLoss` is `30%` , 30% of the audio/video data packets sent to the cloud by the SDK are lost on the transfer linkage.
streamType	Field description: video stream type (HD big image | smooth small image | substream image)
userId	Field description: user ID

videoBitrate	Field description: remote video bitrate (Kbps)
videoBlockRate	Field description: video playback lag rate (%) Video playback lag rate (videoBlockRate) = cumulative video playback lag duration (videoTotalBlockTime)/total video playback duration
videoPacketLoss	Field description: total packet loss rate (%) of the video stream `videoPacketLoss` represents the packet loss rate eventually calculated on the audience side after the audio/video stream goes through the complete transfer linkage of "anchor -> cloud -> audience". The smaller the `videoPacketLoss` , the better. The packet loss rate of 0 indicates that all data of the video stream has entirely reached the audience. If `downLoss` is `0` but `videoPacketLoss` isn't, there is no packet loss on the linkage of "cloud -> audience" for the video stream, but there are unrecoverable packet losses on the linkage of "anchor -> cloud".
videoTotalBlockTime	Field description: cumulative video playback lag duration (ms)
width	Field description: remote video width in px

TRTCStatistics

TRTCStatistics

Network and performance metrics

EnumType	DESC
appCpu	Field description: CPU utilization (%) of the current application, Android 8.0 and above systems are not supported
appMemoryUsageInMB	Field description: Memory usage size (MB) of current application
downLoss	Field description: downstream packet loss rate (%) from cloud to the SDK The smaller the value, the better. If `downLoss` is `0%` , the downstream network quality is very good, and the data packets received from the cloud are basically not lost. If `downLoss` is `30%` , 30% of the audio/video data packets sent to the SDK by the cloud are lost on the transfer linkage.
gatewayRtt	Field description: round-trip delay (ms) from the SDK to gateway

	This value represents the total time it takes to send a network packet from the SDK to the gateway and then send a network packet back from the gateway to the SDK, i.e., the total time it takes for a network packet to go through the linkage of "SDK -> gateway -> SDK". The smaller the value, the better. If `gatewayRtt` is below 50 ms, it means a short audio/video call delay; if `gatewayRtt` is above 200 ms, it means a long audio/video call delay. It should be explained that `gatewayRtt` is invalid for cellular network.
localStatisticsArray	Field description: local audio/video statistics As there may be three local audio/video streams (i.e., HD big image, smooth small image, and substream image), the local audio/video statistics are an array.
localStatisticsArraySize	Field description: `localStatisticsArray` array size
receivedBytes	Field description: total number of received bytes (including signaling data and audio/video data)
remoteStatisticsArray	Field description: remote audio/video statistics As there may be multiple concurrent remote users, and each of them may have multiple concurrent audio/video streams (i.e., HD big image, smooth small image, and substream image), the remote audio/video statistics are an array.
remoteStatisticsArraySize	Field description: `remoteStatisticsArray` array size
rtt	Field description: round-trip delay (ms) from the SDK to cloud This value represents the total time it takes to send a network packet from the SDK to the cloud and then send a network packet back from the cloud to the SDK, i.e., the total time it takes for a network packet to go through the linkage of "SDK -> cloud -> SDK". The smaller the value, the better. If `rtt` is below 50 ms, it means a short audio/video call delay; if `rtt` is above 200 ms, it means a long audio/video call delay. It should be explained that `rtt` represents the total time spent on the linkage of "SDK -> cloud -> SDK"; therefore, there is no need to distinguish between `upRtt` and `downRtt`.
sentBytes	Field description: total number of sent bytes (including signaling data and audio/video data)
systemCpu	Field description: CPU utilization (%) of the current system, Android 8.0 and above systems are not supported
systemMemoryInMB	Field description: Memory size (MB) of current system

systemMemoryUsageInMB	Field description: Memory usage size (MB) of current system , iOS and MAC are not supported
upLoss	Field description: upstream packet loss rate (%) from the SDK to cloud The smaller the value, the better. If `upLoss` is `0%` , the upstream network quality is very good, and the data packets uploaded to the cloud are basically not lost. If `upLoss` is `30%` , 30% of the audio/video data packets sent to the cloud by the SDK are lost on the transfer linkage.

ITXAudioEffectManager

Last updated：2024-06-06 15:26:14

Module: management class for background music, short audio effects, and voice effects

Description: sets background music, short audio effects, and voice effects

ITXAudioEffectManager

ITXMusicPreloadObserver

FuncList	DESC
onLoadProgress	Background music preload progress
onLoadError	Background music preload error

ITXMusicPlayObserver

FuncList	DESC
onStart	Background music started.
onPlayProgress	Playback progress of background music
onComplete	Background music ended

ITXAudioEffectManager

FuncList	DESC
enableVoiceEarMonitor	Enabling in-ear monitoring
setVoiceEarMonitorVolume	Setting in-ear monitoring volume

setVoiceReverbType	Setting voice reverb effects
setVoiceChangerType	Setting voice changing effects
setVoiceCaptureVolume	Setting speech volume
setVoicePitch	Setting speech pitch
setMusicObserver	Setting the background music callback
startPlayMusic	Starting background music
stopPlayMusic	Stopping background music
pausePlayMusic	Pausing background music
resumePlayMusic	Resuming background music
setAllMusicVolume	Setting the local and remote playback volume of background music
setMusicPublishVolume	Setting the remote playback volume of a specific music track
setMusicPlayoutVolume	Setting the local playback volume of a specific music track
setMusicPitch	Adjusting the pitch of background music
setMusicSpeedRate	Changing the speed of background music
getMusicCurrentPosInMS	Getting the playback progress (ms) of background music
getMusicDurationInMS	Getting the total length (ms) of background music
seekMusicToPosInTime	Setting the playback progress (ms) of background music
setMusicScratchSpeedRate	Adjust the speed change effect of the scratch disc
setPreloadObserver	Setting music preload callback
preloadMusic	Preload background music
getMusicTrackCount	Get the number of tracks of background music
setMusicTrack	Specify the playback track of background music

StructType

FuncList	DESC

AudioMusicParam	Background music playback information

EnumType

EnumType	DESC
TXVoiceReverbType	Reverb effects
TXVoiceChangerType	Voice changing effects

onLoadProgress

onLoadProgress

void onLoadProgress	(int id
	int progress)

Background music preload progress

onLoadError

onLoadError

void onLoadError	(int id
	int errorCode)

Background music preload error

Param	DESC
errorCode	-4001: Failed to open the file, such as invalid data found when processing input, ffmpeg protocol not found, etc; -4002: Decoding failure, such as audio file corruption, inaccessible network audio file server, etc; -4003: The number of preloads exceeded the limit， Please call stopPlayMusic first to release the useless preload；-4005: Invalid path, Please check whether the path you passed points to a legal music file；-4006: Invalid URL, Please use a browser to check whether the URL address you passed in can download the desired music file；-4007: No audio stream, Please confirm whether the file you passed is a legal audio file and whether the file is damaged；-4008: Unsupported format, Please confirm whether the

	file format you passed is a supported file format. The mobile version supports [mp3, aac, m4a, wav, ogg, mp4, mkv], and the desktop version supports [mp3, aac, m4a, wav, mp4, mkv].

onStart

onStart

void onStart	(int id
	int errCode)

Background music started.

Called after the background music starts.

Param	DESC
errCode	0: Start playing successfully; -4001: Failed to open the file, such as invalid data found when processing input, ffmpeg protocol not found, etc; -4005: Invalid path, Please check whether the path you passed points to a legal music file；-4006: Invalid URL, Please use a browser to check whether the URL address you passed in can download the desired music file；-4007: No audio stream, Please confirm whether the file you passed is a legal audio file and whether the file is damaged；-4008: Unsupported format, Please confirm whether the file format you passed is a supported file format. The mobile version supports [mp3, aac, m4a, wav, ogg, mp4, mkv], and the desktop version supports [mp3, aac, m4a, wav, mp4, mkv].
id	music ID.

onPlayProgress

onPlayProgress

void onPlayProgress	(int id
	long curPtsMS
	long durationMS)

Playback progress of background music

onComplete

onComplete

void onComplete	(int id
	int errCode)

Background music ended

Called when the background music playback ends or an error occurs.

Param	DESC
errCode	0: End of play; -4002: Decoding failure, such as audio file corruption, inaccessible network audio file server, etc.
id	music ID.

enableVoiceEarMonitor

enableVoiceEarMonitor

void enableVoiceEarMonitor	(bool enable)

Enabling in-ear monitoring

After enabling in-ear monitoring, anchors can hear in earphones their own voice captured by the mic. This is designed for singing scenarios.

In-ear monitoring cannot be enabled for Bluetooth earphones. This is because Bluetooth earphones have high latency. Please ask anchors to use wired earphones via a UI reminder.
Given that not all phones deliver excellent in-ear monitoring effects, we have blocked this feature on some phones.

Param	DESC
enable	`true:` enable; `false` : disable

Note

In-ear monitoring can be enabled only when earphones are used. Please remind anchors to use wired earphones.

setVoiceEarMonitorVolume

setVoiceEarMonitorVolume

void setVoiceEarMonitorVolume	(int volume)

Setting in-ear monitoring volume

This API is used to set the volume of in-ear monitoring.

Param	DESC
volume	Volume. Value range: 0-100; default: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setVoiceReverbType

setVoiceReverbType

void setVoiceReverbType	(TXVoiceReverbType type)

Setting voice reverb effects

This API is used to set reverb effects for human voice. For the effects supported, please see TXVoiceReverbType.

Note

Effects become invalid after room exit. If you want to use the same effect after you enter the room again, you need to set the effect again using this API.

setVoiceChangerType

setVoiceChangerType

void setVoiceChangerType	(TXVoiceChangerType type)

Setting voice changing effects

This API is used to set voice changing effects. For the effects supported, please see TXVoiceChangeType.

Note

Effects become invalid after room exit. If you want to use the same effect after you enter the room again, you need to set the effect again using this API.

setVoiceCaptureVolume

setVoiceCaptureVolume

void setVoiceCaptureVolume	(int volume)

Setting speech volume

This API is used to set the volume of speech. It is often used together with the music volume setting API setAllMusicVolume to balance between the volume of music and speech.

Param	DESC
volume	Volume. Value range: 0-100; default: 100

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setVoicePitch

setVoicePitch

void setVoicePitch	(double pitch)

Setting speech pitch

This API is used to set the pitch of speech.

Param	DESC
pitch	Ptich，Value range: -1.0f~1.0f; default: 0.0f。

setMusicObserver

setMusicObserver

void setMusicObserver	(int musicId
	ITXMusicPlayObserver* observer)

Setting the background music callback

Before playing background music, please use this API to set the music callback, which can inform you of the playback progress.

Param	DESC

musicId	Music ID
observer	For more information, please see the APIs defined in `ITXMusicPlayObserver` .

Note

1. If the ID does not need to be used, the observer can be set to NULL to release it completely.

startPlayMusic

startPlayMusic

void startPlayMusic	(AudioMusicParam musicParam)

Starting background music

You must assign an ID to each music track so that you can start, stop, or set the volume of music tracks by ID.

Param	DESC
musicParam	Music parameter

Note

1. If you play the same music track multiple times, please use the same ID instead of a separate ID for each playback.
2. If you want to play different music tracks at the same time, use different IDs for them.
3. If you use the same ID to play a music track different from the current one, the SDK will stop the current one before playing the new one.

stopPlayMusic

stopPlayMusic

void stopPlayMusic	(int id)

Stopping background music

Param	DESC
id	Music ID

pausePlayMusic

pausePlayMusic

void pausePlayMusic	(int id)

Pausing background music

Param	DESC
id	Music ID

resumePlayMusic

resumePlayMusic

void resumePlayMusic	(int id)

Resuming background music

Param	DESC
id	Music ID

setAllMusicVolume

setAllMusicVolume

void setAllMusicVolume	(int volume)

Setting the local and remote playback volume of background music

This API is used to set the local and remote playback volume of background music.

Local volume: the volume of music heard by anchors

Remote volume: the volume of music heard by audience

Param	DESC
volume	Volume. Value range: 0-100; default: 60

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setMusicPublishVolume

setMusicPublishVolume

void setMusicPublishVolume	(int id
	int volume)

Setting the remote playback volume of a specific music track

This API is used to control the remote playback volume (the volume heard by audience) of a specific music track.

Param	DESC
id	Music ID
volume	Volume. Value range: 0-100; default: 60

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setMusicPlayoutVolume

setMusicPlayoutVolume

void setMusicPlayoutVolume	(int id
	int volume)

Setting the local playback volume of a specific music track

This API is used to control the local playback volume (the volume heard by anchors) of a specific music track.

Param	DESC
id	Music ID
volume	Volume. Value range: 0-100. default: 60

Note

If 100 is still not loud enough for you, you can set the volume to up to 150, but there may be side effects.

setMusicPitch

setMusicPitch

void setMusicPitch	(int id
	float pitch)

Adjusting the pitch of background music

Param	DESC
id	Music ID
pitch	Pitch. Value range: floating point numbers in the range of [-1, 1]; default: 0.0f

setMusicSpeedRate

setMusicSpeedRate

void setMusicSpeedRate	(int id
	float speedRate)

Changing the speed of background music

Param	DESC
id	Music ID
speedRate	Music speed. Value range: floating point numbers in the range of [0.5, 2]; default: 1.0f

getMusicCurrentPosInMS

getMusicCurrentPosInMS

long getMusicCurrentPosInMS	(int id)

Getting the playback progress (ms) of background music

Param	DESC

id	Music ID

Return Desc:

The milliseconds that have passed since playback started. -1 indicates failure to get the the playback progress.

getMusicDurationInMS

getMusicDurationInMS

long getMusicDurationInMS	(char* path)

Getting the total length (ms) of background music

Param	DESC
path	Path of the music file.

Return Desc:

The length of the specified music file is returned. -1 indicates failure to get the length.

seekMusicToPosInTime

seekMusicToPosInTime

void seekMusicToPosInTime	(int id
	int pts)

Setting the playback progress (ms) of background music

Param	DESC
id	Music ID
pts	Unit: millisecond

Note

Do not call this API frequently as the music file may be read and written to each time the API is called, which can be time-consuming.

Wait till users finish dragging the progress bar before you call this API.
The progress bar controller on the UI tends to update the progress at a high frequency as users drag the progress bar. This will result in poor user experience unless you limit the frequency.

setMusicScratchSpeedRate

setMusicScratchSpeedRate

void setMusicScratchSpeedRate	(int id
	float scratchSpeedRate)

Adjust the speed change effect of the scratch disc

Param	DESC
id	Music ID
scratchSpeedRate	Scratch disc speed, the default value is 1.0f, the range is: a floating point number between [-12.0 ~ 12.0], the positive/negative speed value indicates the direction is positive/negative, and the absolute value indicates the speed.

Note

Precondition preloadMusic succeeds.

setPreloadObserver

setPreloadObserver

void setPreloadObserver	(ITXMusicPreloadObserver* observer)

Setting music preload callback

Before preload music, please use this API to set the preload callback, which can inform you of the preload status.

Param	DESC
observer	For more information, please see the APIs defined in `ITXMusicPreloadObserver` .

preloadMusic

preloadMusic

void preloadMusic	(AudioMusicParam preloadParam)

Preload background music

You must assign an ID to each music track so that you can start, stop, or set the volume of music tracks by ID.

Param	DESC
musicParam	Music parameter

Note

1. Preload supports up to 2 preloads with different IDs at the same time, and the preload time does not exceed 10 minutes,you need to stopPlayMusic after use, otherwise the memory will not be released.
2. If the music corresponding to the ID is being played, the preloading fails, and stopPlayMusic must be called first.
3. When the musicParam passed to startPlayMusic is exactly the same, preloading works.

getMusicTrackCount

getMusicTrackCount

long getMusicTrackCount	(int id)

Get the number of tracks of background music

Param	DESC
id	Music ID

setMusicTrack

setMusicTrack

void setMusicTrack	(int id
	int trackIndex)

Specify the playback track of background music

Param	DESC

id	Music ID
index	Specify which track to play (the first track is played by default). Value range [0, total number of tracks).

Note

The total number of tracks can be obtained through the getMusicTrackCount interface.

TXVoiceReverbType

TXVoiceReverbType

Reverb effects

Reverb effects can be applied to human voice. Based on acoustic algorithms, they can mimic voice in different environments. The following effects are supported currently:

0: original; 1: karaoke; 2: room; 3: hall; 4: low and deep; 5: resonant; 6: metal; 7: husky; 8: ethereal; 9: studio; 10: melodious; 11: studio2;

Enum	Value	DESC
TXLiveVoiceReverbType_0	0	disable
TXLiveVoiceReverbType_1	1	KTV
TXLiveVoiceReverbType_2	2	small room
TXLiveVoiceReverbType_3	3	great hall
TXLiveVoiceReverbType_4	4	deep voice
TXLiveVoiceReverbType_5	5	loud voice
TXLiveVoiceReverbType_6	6	metallic sound
TXLiveVoiceReverbType_7	7	magnetic sound
TXLiveVoiceReverbType_8	8	ethereal
TXLiveVoiceReverbType_9	9	studio
TXLiveVoiceReverbType_10	10	melodious
TXLiveVoiceReverbType_11	11	studio2

TXVoiceChangeType

TXVoiceChangeType

Voice changing effects

Voice changing effects can be applied to human voice. Based on acoustic algorithms, they change the tone of voice. The following effects are supported currently:

0: original; 1: child; 2: little girl; 3: middle-aged man; 4: metal; 5: nasal; 6: foreign accent; 7: trapped beast; 8: otaku; 9: electric; 10: robot; 11: ethereal

Enum	Value	DESC
TXVoiceChangerType_0	0	disable
TXVoiceChangerType_1	1	naughty kid
TXVoiceChangerType_2	2	Lolita
TXVoiceChangerType_3	3	uncle
TXVoiceChangerType_4	4	heavy metal
TXVoiceChangerType_5	5	catch cold
TXVoiceChangerType_6	6	foreign accent
TXVoiceChangerType_7	7	caged animal trapped beast
TXVoiceChangerType_8	8	indoorsman
TXVoiceChangerType_9	9	strong current
TXVoiceChangerType_10	10	heavy machinery
TXVoiceChangerType_11	11	intangible

TXAudioMusicParam

TXAudioMusicParam

Background music playback information

The information, including playback ID, file path, and loop times, is passed in the startPlayMusic API.

1. If you play the same music track multiple times, please use the same ID instead of a separate ID for each playback.

2. If you want to play different music tracks at the same time, use different IDs for them.
3. If you use the same ID to play a music track different from the current one, the SDK will stop the current one before playing the new one.

EnumType	DESC
endTimeMS	`Field description:` the point in time in milliseconds for ending music playback. 0 indicates that playback continues till the end of the music track.
id	`Field description:` music ID **Note** the SDK supports playing multiple music tracks. IDs are used to distinguish different music tracks and control their start, end, volume, etc.
isShortFile	`Field description:` whether the music played is a short music track `Valid values:` `true` : short music track that needs to be looped; `false` (default): normal-length music track
loopCount	`Field description:` number of times the music track is looped `Valid values:` 0 or any positive integer. 0 (default) indicates that the music is played once, 1 twice, and so on.
path	`Field description:` absolute path of the music file or url.the mp3,aac,m4a,wav supported.
publish	`Field description:` whether to send the music to remote users `Valid values:` `true` : remote users can hear the music played locally; `false` (default): only the local user can hear the music.
startTimeMS	`Field description:` the point in time in milliseconds for starting music playback

ITXDeviceManager

Last updated：2024-06-06 15:26:14

Module: audio/video device management module

Description: manages audio/video devices such as camera, mic, and speaker.

ITXDeviceManager

ITXDeviceManager

FuncList	DESC
isFrontCamera	Querying whether the front camera is being used
switchCamera	Switching to the front/rear camera (for mobile OS)
getCameraZoomMaxRatio	Getting the maximum zoom ratio of the camera (for mobile OS)
setCameraZoomRatio	Setting the camera zoom ratio (for mobile OS)
isAutoFocusEnabled	Querying whether automatic face detection is supported (for mobile OS)
enableCameraAutoFocus	Enabling auto focus (for mobile OS)
setCameraFocusPosition	Adjusting the focus (for mobile OS)
enableCameraTorch	Enabling/Disabling flash, i.e., the torch mode (for mobile OS)
setAudioRoute	Setting the audio route (for mobile OS)
getDevicesList	Getting the device list (for desktop OS)
setCurrentDevice	Setting the device to use (for desktop OS)
getCurrentDevice	Getting the device currently in use (for desktop OS)
setCurrentDeviceVolume	Setting the volume of the current device (for desktop OS)
getCurrentDeviceVolume	Getting the volume of the current device (for desktop OS)

setCurrentDeviceMute	Muting the current device (for desktop OS)
getCurrentDeviceMute	Querying whether the current device is muted (for desktop OS)
enableFollowingDefaultAudioDevice	Set the audio device used by SDK to follow the system default device (for desktop OS)
startCameraDeviceTest	Starting camera testing (for desktop OS)
stopCameraDeviceTest	Ending camera testing (for desktop OS)
startMicDeviceTest	Starting mic testing (for desktop OS)
startMicDeviceTest	Starting mic testing (for desktop OS)
stopMicDeviceTest	Ending mic testing (for desktop OS)
startSpeakerDeviceTest	Starting speaker testing (for desktop OS)
stopSpeakerDeviceTest	Ending speaker testing (for desktop OS)
startCameraDeviceTest	Starting camera testing (for desktop OS)
setApplicationPlayVolume	Setting the volume of the current process in the volume mixer (for Windows)
getApplicationPlayVolume	Getting the volume of the current process in the volume mixer (for Windows)
setApplicationMuteState	Muting the current process in the volume mixer (for Windows)
getApplicationMuteState	Querying whether the current process is muted in the volume mixer (for Windows)
setCameraCapturerParam	Set camera acquisition preferences
setDeviceObserver	set onDeviceChanged callback
setSystemVolumeType	Setting the system volume type (for mobile OS)

StructType

FuncList	DESC
TXCameraCaptureParam	Camera acquisition parameters

ITXDeviceInfo	Audio/Video device information (for desktop OS)
ITXDeviceCollection	Device information list (for desktop OS)

EnumType

EnumType	DESC
TXSystemVolumeType	System volume type
TXAudioRoute	Audio route (the route via which audio is played)
TXMediaDeviceType	Device type (for desktop OS)
TXMediaDeviceState	Device operation
TXCameraCaptureMode	Camera acquisition preferences

isFrontCamera

isFrontCamera

Querying whether the front camera is being used

switchCamera

switchCamera

int switchCamera	(bool frontCamera)

Switching to the front/rear camera (for mobile OS)

getCameraZoomMaxRatio

getCameraZoomMaxRatio

Getting the maximum zoom ratio of the camera (for mobile OS)

setCameraZoomRatio

setCameraZoomRatio

int setCameraZoomRatio	(float zoomRatio)

Setting the camera zoom ratio (for mobile OS)

Param	DESC
zoomRatio	Value range: 1-5. 1 indicates the widest angle of view (original), and 5 the narrowest angle of view (zoomed in).The maximum value is recommended to be 5. If the value exceeds 5, the video will become blurred.

isAutoFocusEnabled

isAutoFocusEnabled

Querying whether automatic face detection is supported (for mobile OS)

enableCameraAutoFocus

enableCameraAutoFocus

int enableCameraAutoFocus	(bool enabled)

Enabling auto focus (for mobile OS)

After auto focus is enabled, the camera will automatically detect and always focus on faces.

setCameraFocusPosition

setCameraFocusPosition

int setCameraFocusPosition	(float x
	float y)

Adjusting the focus (for mobile OS)

This API can be used to achieve the following:

1. A user can tap on the camera preview.
2. A rectangle will appear where the user taps, indicating the spot the camera will focus on.
3. The user passes the coordinates of the spot to the SDK using this API, and the SDK will instruct the camera to focus as required.

Param	DESC
position	The spot to focus on. Pass in the coordinates of the spot you want to focus on.

Note

Before using this API, you must first disable auto focus using enableCameraAutoFocus.

Return Desc:

0: operation successful; negative number: operation failed.

enableCameraTorch

enableCameraTorch

int enableCameraTorch	(bool enabled)

Enabling/Disabling flash, i.e., the torch mode (for mobile OS)

setAudioRoute

setAudioRoute

int setAudioRoute	(TXAudioRoute route)

Setting the audio route (for mobile OS)

A mobile phone has two audio playback devices: the receiver at the top and the speaker at the bottom.
If the audio route is set to the receiver, the volume is relatively low, and audio can be heard only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.
If the audio route is set to the speaker, the volume is relatively high, and there is no need to put the phone near the ear. This mode enables the "hands-free" feature.

getDevicesList

getDevicesList

ITXDeviceCollection* getDevicesList	(TXMediaDeviceType type)

Getting the device list (for desktop OS)

Param	DESC
type	Device type. Set it to the type of device you want to get. For details, please see the definition of `TXMediaDeviceType` .

Note

To ensure that the SDK can manage the lifecycle of the `ITXDeviceCollection` object, after using this API, please call the `release` method to release the resources.

Do not use `delete` to release the Collection object returned as deleting the ITXDeviceCollection* pointer will cause crash.

The valid values of `type` are `TXMediaDeviceTypeMic` , `TXMediaDeviceTypeSpeaker` , and `TXMediaDeviceTypeCamera` .

This API can be used only on macOS and Windows.

setCurrentDevice

setCurrentDevice

int setCurrentDevice	(TXMediaDeviceType type
	const char* deviceId)

Setting the device to use (for desktop OS)

Param	DESC
deviceId	Device ID. You can get the ID of a device using the getDevicesList API.
type	Device type. For details, please see the definition of `TXMediaDeviceType` .

Return Desc:

0: operation successful; negative number: operation failed.

getCurrentDevice

getCurrentDevice

ITXDeviceInfo* getCurrentDevice	(TXMediaDeviceType type)

Getting the device currently in use (for desktop OS)

setCurrentDeviceVolume

setCurrentDeviceVolume

int setCurrentDeviceVolume	(TXMediaDeviceType type
	uint32_t volume)

Setting the volume of the current device (for desktop OS)

This API is used to set the capturing volume of the mic or playback volume of the speaker, but not the volume of the camera.

Param	DESC
volume	Volume. Value range: 0-100; default: 100

getCurrentDeviceVolume

getCurrentDeviceVolume

uint32_t getCurrentDeviceVolume	(TXMediaDeviceType type)

Getting the volume of the current device (for desktop OS)

This API is used to get the capturing volume of the mic or playback volume of the speaker, but not the volume of the camera.

setCurrentDeviceMute

setCurrentDeviceMute

int setCurrentDeviceMute	(TXMediaDeviceType type

	bool mute)

Muting the current device (for desktop OS)

This API is used to mute the mic or speaker, but not the camera.

getCurrentDeviceMute

getCurrentDeviceMute

bool getCurrentDeviceMute	(TXMediaDeviceType type)

Querying whether the current device is muted (for desktop OS)

This API is used to query whether the mic or speaker is muted. Camera muting is not supported.

enableFollowingDefaultAudioDevice

enableFollowingDefaultAudioDevice

int enableFollowingDefaultAudioDevice	(TXMediaDeviceType type
	bool enable)

Set the audio device used by SDK to follow the system default device (for desktop OS)

This API is used to set the microphone and speaker types. Camera following the system default device is not supported.

Param	DESC
enable	Whether to follow the system default audio device. true: following. When the default audio device of the system is changed or new audio device is plugged in, the SDK immediately switches the audio device. false：not following. When the default audio device of the system is changed or new audio device is plugged in, the SDK doesn't switch the audio device.
type	Device type. For details, please see the definition of `TXMediaDeviceType` .

startCameraDeviceTest

startCameraDeviceTest

int startCameraDeviceTest	(void* view)

Starting camera testing (for desktop OS)

Note

You can use the setCurrentDevice API to switch between cameras during testing.

stopCameraDeviceTest

stopCameraDeviceTest

Ending camera testing (for desktop OS)

startMicDeviceTest

startMicDeviceTest

int startMicDeviceTest	(uint32_t interval)

Starting mic testing (for desktop OS)

This API is used to test whether the mic functions properly. The mic volume detected (value range: 0-100) is returned via a callback.

Param	DESC
interval	Interval of volume callbacks

Note

When this interface is called, the sound recorded by the microphone will be played back to the speakers by default.

startMicDeviceTest

startMicDeviceTest

int startMicDeviceTest	(uint32_t interval
	bool playback)

Starting mic testing (for desktop OS)

This API is used to test whether the mic functions properly. The mic volume detected (value range: 0-100) is returned via a callback.

Param	DESC
interval	Interval of volume callbacks
playback	Whether to play back the microphone sound. The user will hear his own sound when testing the microphone if `playback` is true.

stopMicDeviceTest

stopMicDeviceTest

Ending mic testing (for desktop OS)

startSpeakerDeviceTest

startSpeakerDeviceTest

int startSpeakerDeviceTest	(const char* filePath)

Starting speaker testing (for desktop OS)

This API is used to test whether the audio playback device functions properly by playing a specified audio file. If users can hear audio during testing, the device functions properly.

Param	DESC
filePath	Path of the audio file

stopSpeakerDeviceTest

stopSpeakerDeviceTest

Ending speaker testing (for desktop OS)

startCameraDeviceTest

startCameraDeviceTest

int startCameraDeviceTest	(ITRTCVideoRenderCallback* callback)

Starting camera testing (for desktop OS)

This API supports custom rendering, meaning that you can use the callback API `ITRTCVideoRenderCallback` to get the images captured by the camera for custom rendering.

setApplicationPlayVolume

setApplicationPlayVolume

int setApplicationPlayVolume	(int volume)

Setting the volume of the current process in the volume mixer (for Windows)

getApplicationPlayVolume

getApplicationPlayVolume

Getting the volume of the current process in the volume mixer (for Windows)

setApplicationMuteState

setApplicationMuteState

int setApplicationMuteState	(bool bMute)

Muting the current process in the volume mixer (for Windows)

getApplicationMuteState

getApplicationMuteState

Querying whether the current process is muted in the volume mixer (for Windows)

setCameraCapturerParam

setCameraCapturerParam

void setCameraCapturerParam	(const TXCameraCaptureParam& params)

Set camera acquisition preferences

setDeviceObserver

setDeviceObserver

void setDeviceObserver	(ITXDeviceObserver* observer)

set onDeviceChanged callback

setSystemVolumeType

setSystemVolumeType

int setSystemVolumeType	(TXSystemVolumeType type)

Setting the system volume type (for mobile OS)

@deprecated This API is not recommended after v9.5. Please use the `startLocalAudio(quality)` API in `TRTCCloud` instead, which param `quality` is used to decide audio quality.

TXSystemVolumeType(Deprecated)

TXSystemVolumeType(Deprecated)

System volume type

Enum	Value	DESC
TXSystemVolumeTypeAuto	0	Auto
TXSystemVolumeTypeMedia	1	Media volume
TXSystemVolumeTypeVOIP	2	Call volume

TXAudioRoute

TXAudioRoute

Audio route (the route via which audio is played)

Audio route is the route (speaker or receiver) via which audio is played. It applies only to mobile devices such as mobile phones.
A mobile phone has two speakers: one at the top (receiver) and the other the bottom.
If the audio route is set to the receiver, the volume is relatively low, and audio can be heard only when the phone is put near the ear. This mode has a high level of privacy and is suitable for answering calls.
If the audio route is set to the speaker, the volume is relatively high, and there is no need to put the phone near the ear. This mode enables the "hands-free" feature.

Enum	Value	DESC
TXAudioRouteSpeakerphone	0	Speakerphone: the speaker at the bottom is used for playback (hands-free). With relatively high volume, it is used to play music out loud.
TXAudioRouteEarpiece	1	Earpiece: the receiver at the top is used for playback. With relatively low volume, it is suitable for call scenarios that require privacy.

TXMediaDeviceType

TXMediaDeviceType

Device type (for desktop OS)

This enumerated type defines three types of audio/video devices, namely camera, mic and speaker, so that you can use the same device management API to manage three types of devices.

Enum	Value	DESC
TXMediaDeviceTypeUnknown	-1	undefined device type
TXMediaDeviceTypeMic	0	microphone
TXMediaDeviceTypeSpeaker	1	speaker or earpiece
TXMediaDeviceTypeCamera	2	camera

TXMediaDeviceState

TXMediaDeviceState

Device operation

This enumerated value is used to notify the status change of the local device onDeviceChanged.

Enum	Value	DESC
TXMediaDeviceStateAdd	0	The device has been plugged in
TXMediaDeviceStateRemove	1	The device has been removed
TXMediaDeviceStateActive	2	The device has been enabled
TXMediaDefaultDeviceChanged	3	system default device changed

TXCameraCaptureMode

TXCameraCaptureMode

Camera acquisition preferences

This enum is used to set camera acquisition parameters.

Enum	Value	DESC
TXCameraResolutionStrategyAuto	0	Auto adjustment of camera capture parameters. SDK selects the appropriate camera output parameters according to the actual acquisition device performance and network situation, and maintains a balance between device performance and video preview quality.
TXCameraResolutionStrategyPerformance	1	Give priority to equipment performance. SDK selects the closest camera output parameters according to the user's encoder resolution and frame rate, so as to ensure the performance of the device.
TXCameraResolutionStrategyHighQuality	2	Give priority to the quality of video preview. SDK selects higher camera output parameters to improve the quality of preview

		video. In this case, it will consume more CPU and memory to do video preprocessing.
TXCameraCaptureManual	3	Allows the user to set the width and height of the video captured by the local camera.

TXCameraCaptureParam

TXCameraCaptureParam

Camera acquisition parameters

This setting determines the quality of the local preview image.

EnumType	DESC
height	Field description: height of acquired image
mode	Field description: camera acquisition preferences,please see TXCameraCaptureMode
width	Field description: width of acquired image

TXMediaDeviceInfo

TXMediaDeviceInfo

Audio/Video device information (for desktop OS)

This structure describes key information (such as device ID and device name) of an audio/video device, so that users can choose on the UI the device to use.

EnumType	DESC
getDeviceName()	device name (UTF-8)
getDevicePID()	device id (UTF-8)

ITXDeviceCollection

ITXDeviceCollection

Device information list (for desktop OS)

This structure functions as std::vector<ITXDeviceInfo> does. It solves the binary compatibility issue between different versions of STL containers.

EnumType	DESC
getCount()	Size of this list. return Size of this list.
index)	device properties (json format) **Note** examples: {"SupportedResolution":[{"width":640,"height":480},{"width":320,"height":240}]} param index value in [0,getCount),return device properties formatted by json
release()	release function, don't use delete!!!

Type Definition

Last updated：2024-06-06 15:50:06

Module: TRTC key class definition

Description: definitions of enumerated and constant values such as resolution and quality level

Type defiine

StructType

FuncList	DESC
TRTCParams	Room entry parameters
TRTCVideoEncParam	Video encoding parameters
TRTCNetworkQosParam	Network QoS control parameter set
TRTCRenderParams	Rendering parameters of video image
TRTCQualityInfo	Network quality
TRTCVolumeInfo	Volume
TRTCSpeedTestParams	Network speed testing parameters
TRTCSpeedTestResult	Network speed test result
TRTCTexture	Video texture data
TRTCVideoFrame	Video frame information
TRTCAudioFrame	Audio frame data
TRTCMixUser	Description information of each video image in On-Cloud MixTranscoding
TRTCTranscodingConfig	Layout and transcoding parameters of On-Cloud MixTranscoding
TRTCPublishCDNParam	Push parameters required to be set when publishing

	audio/video streams to non-Tencent Cloud CDN
TRTCAudioRecordingParams	Local audio file recording parameters
TRTCLocalRecordingParams	Local media file recording parameters
TRTCAudioEffectParam	Sound effect parameter (disused)
TRTCSwitchRoomConfig	Room switch parameter
TRTCAudioFrameCallbackFormat	Format parameter of custom audio callback
TRTCImageBuffer	Structure for storing window thumbnails and icons.
TRTCUser	The users whose streams to publish
TRTCPublishCdnUrl	The destination URL when you publish to Tencent Cloud or a third-party CDN
TRTCPublishTarget	The publishing destination
TRTCVideoLayout	The video layout of the transcoded stream
TRTCWatermark	The watermark layout
TRTCStreamEncoderParam	The encoding parameters
TRTCStreamMixingConfig	The transcoding parameters
TRTCPayloadPrivateEncryptionConfig	Media Stream Private Encryption Configuration
TRTCAudioVolumeEvaluateParams	Volume evaluation and other related parameter settings.

EnumType

EnumType	DESC
TRTCVideoResolution	Video resolution
TRTCVideoResolutionMode	Video aspect ratio mode
TRTCVideoStreamType	Video stream type
TRTCVideoFillMode	Video image fill mode
TRTCVideoRotation	Video image rotation direction

TRTCBeautyStyle	Beauty (skin smoothing) filter algorithm
TRTCVideoPixelFormat	Video pixel format
TRTCVideoBufferType	Video data transfer method
TRTCVideoMirrorType	Video mirror type
TRTCSnapshotSourceType	Data source of local video screenshot
TRTCAppScene	Use cases
TRTCRoleType	Role
TRTCQosControlMode	QoS control mode (disused)
TRTCVideoQosPreference	Image quality preference
TRTCQuality	Network quality
TRTCAVStatusType	Audio/Video playback status
TRTCAVStatusChangeReason	Reasons for playback status changes
TRTCAudioQuality	Sound quality
TRTCAudioFrameFormat	Audio frame content format
TRTCAudioFrameOperationMode	Audio callback data operation mode
TRTCLogLevel	Log level
TRTCScreenCaptureSourceType	Screen sharing target type (for desktops only)
TRTCTranscodingConfigMode	Layout mode of On-Cloud MixTranscoding
TRTCLocalRecordType	Media recording type
TRTCMixInputType	Stream mix input type
TRTCWaterMarkSrcType	Watermark image source type
TRTCAudioRecordingContent	Audio recording content type
TRTCPublishMode	The publishing mode
TRTCEncryptionAlgorithm	Encryption Algorithm
TRTCSpeedTestScene	Speed Test Scene

TRTCGravitySensorAdaptiveMode	Set the adaptation mode of gravity sensing (only applicable to mobile terminals)

TRTCVideoResolution

TRTCVideoResolution

Video resolution

Here, only the landscape resolution (e.g., 640x360) is defined. If the portrait resolution (e.g., 360x640) needs to be used, `Portrait` must be selected for `TRTCVideoResolutionMode` .

Enum	Value	DESC
TRTCVideoResolution_120_120	1	Aspect ratio: 1:1; resolution: 120x120; recommended bitrate (VideoCall): 80 Kbps; recommended bitrate (LIVE): 120 Kbps.
TRTCVideoResolution_160_160	3	Aspect ratio: 1:1; resolution: 160x160; recommended bitrate (VideoCall): 100 Kbps; recommended bitrate (LIVE): 150 Kbps.
TRTCVideoResolution_270_270	5	Aspect ratio: 1:1; resolution: 270x270; recommended bitrate (VideoCall): 200 Kbps; recommended bitrate (LIVE): 300 Kbps.
TRTCVideoResolution_480_480	7	Aspect ratio: 1:1; resolution: 480x480; recommended bitrate (VideoCall): 350 Kbps; recommended bitrate (LIVE): 500 Kbps.
TRTCVideoResolution_160_120	50	Aspect ratio: 4:3; resolution: 160x120; recommended bitrate (VideoCall): 100 Kbps; recommended bitrate (LIVE): 150 Kbps.
TRTCVideoResolution_240_180	52	Aspect ratio: 4:3; resolution: 240x180; recommended bitrate (VideoCall): 150 Kbps; recommended bitrate (LIVE): 250 Kbps.
TRTCVideoResolution_280_210	54	Aspect ratio: 4:3; resolution: 280x210; recommended bitrate (VideoCall): 200 Kbps; recommended bitrate (LIVE): 300 Kbps.
TRTCVideoResolution_320_240	56	Aspect ratio: 4:3; resolution: 320x240; recommended bitrate (VideoCall): 250 Kbps; recommended bitrate (LIVE): 375 Kbps.

TRTCVideoResolution_400_300	58	Aspect ratio: 4:3; resolution: 400x300; recommended bitrate (VideoCall): 300 Kbps; recommended bitrate (LIVE): 450 Kbps.
TRTCVideoResolution_480_360	60	Aspect ratio: 4:3; resolution: 480x360; recommended bitrate (VideoCall): 400 Kbps; recommended bitrate (LIVE): 600 Kbps.
TRTCVideoResolution_640_480	62	Aspect ratio: 4:3; resolution: 640x480; recommended bitrate (VideoCall): 600 Kbps; recommended bitrate (LIVE): 900 Kbps.
TRTCVideoResolution_960_720	64	Aspect ratio: 4:3; resolution: 960x720; recommended bitrate (VideoCall): 1000 Kbps; recommended bitrate (LIVE): 1500 Kbps.
TRTCVideoResolution_160_90	100	Aspect ratio: 16:9; resolution: 160x90; recommended bitrate (VideoCall): 150 Kbps; recommended bitrate (LIVE): 250 Kbps.
TRTCVideoResolution_256_144	102	Aspect ratio: 16:9; resolution: 256x144; recommended bitrate (VideoCall): 200 Kbps; recommended bitrate (LIVE): 300 Kbps.
TRTCVideoResolution_320_180	104	Aspect ratio: 16:9; resolution: 320x180; recommended bitrate (VideoCall): 250 Kbps; recommended bitrate (LIVE): 400 Kbps.
TRTCVideoResolution_480_270	106	Aspect ratio: 16:9; resolution: 480x270; recommended bitrate (VideoCall): 350 Kbps; recommended bitrate (LIVE): 550 Kbps.
TRTCVideoResolution_640_360	108	Aspect ratio: 16:9; resolution: 640x360; recommended bitrate (VideoCall): 500 Kbps; recommended bitrate (LIVE): 900 Kbps.
TRTCVideoResolution_960_540	110	Aspect ratio: 16:9; resolution: 960x540; recommended bitrate (VideoCall): 850 Kbps; recommended bitrate (LIVE): 1300 Kbps.
TRTCVideoResolution_1280_720	112	Aspect ratio: 16:9; resolution: 1280x720; recommended bitrate (VideoCall): 1200 Kbps; recommended bitrate (LIVE): 1800 Kbps.
TRTCVideoResolution_1920_1080	114	Aspect ratio: 16:9; resolution: 1920x1080; recommended bitrate (VideoCall): 2000 Kbps; recommended bitrate (LIVE): 3000 Kbps.

TRTCVideoResolutionMode

TRTCVideoResolutionMode

Video aspect ratio mode

Only the landscape resolution (e.g., 640x360) is defined in `TRTCVideoResolution` . If the portrait resolution (e.g., 360x640) needs to be used, `Portrait` must be selected for `TRTCVideoResolutionMode` .

Enum	Value	DESC
TRTCVideoResolutionModeLandscape	0	Landscape resolution, such as TRTCVideoResolution_640_360 + TRTCVideoResolutionModeLandscape = 640x360.
TRTCVideoResolutionModePortrait	1	Portrait resolution, such as TRTCVideoResolution_640_360 + TRTCVideoResolutionModePortrait = 360x640.

TRTCVideoStreamType

TRTCVideoStreamType

Video stream type

TRTC provides three different video streams, including:

HD big image: it is generally used to transfer video data from the camera.

Smooth small image: it has the same content as the big image, but with lower resolution and bitrate and thus lower definition.

Substream image: it is generally used for screen sharing. Only one user in the room is allowed to publish the substream video image at any time, while other users must wait for this user to close the substream before they can publish their own substream.

Note

The SDK does not support enabling the smooth small image alone, which must be enabled together with the big image. It will automatically set the resolution and bitrate of the small image.

Enum	Value	DESC
TRTCVideoStreamTypeBig	0	HD big image: it is generally used to transfer video data from the camera.
TRTCVideoStreamTypeSmall	1	Smooth small image: it has the same content as the big image, but with lower resolution and bitrate and thus lower

		definition.
TRTCVideoStreamTypeSub	2	Substream image: it is generally used for screen sharing. Only one user in the room is allowed to publish the substream video image at any time, while other users must wait for this user to close the substream before they can publish their own substream.

TRTCVideoFillMode

TRTCVideoFillMode

Video image fill mode

If the aspect ratio of the video display area is not equal to that of the video image, you need to specify the fill mode:

Enum	Value	DESC
TRTCVideoFillMode_Fill	0	Fill mode: the video image will be centered and scaled to fill the entire display area, where parts that exceed the area will be cropped. The displayed image may be incomplete in this mode.
TRTCVideoFillMode_Fit	1	Fit mode: the video image will be scaled based on its long side to fit the display area, where the short side will be filled with black bars. The displayed image is complete in this mode, but there may be black bars.

TRTCVideoRotation

TRTCVideoRotation

Video image rotation direction

TRTC provides rotation angle setting APIs for local and remote images. The following rotation angles are all clockwise.

Enum	Value	DESC
TRTCVideoRotation0	0	No rotation
TRTCVideoRotation90	1	Clockwise rotation by 90 degrees
TRTCVideoRotation180	2	Clockwise rotation by 180 degrees

TRTCVideoRotation270	3	Clockwise rotation by 270 degrees

TRTCBeautyStyle

TRTCBeautyStyle

Beauty (skin smoothing) filter algorithm

TRTC has multiple built-in skin smoothing algorithms. You can select the one most suitable for your product.

Enum	Value	DESC
TRTCBeautyStyleSmooth	0	Smooth style, which uses a more radical algorithm for more obvious effect and is suitable for show live streaming.
TRTCBeautyStyleNature	1	Natural style, which retains more facial details for more natural effect and is suitable for most live streaming use cases.

TRTCVideoPixelFormat

TRTCVideoPixelFormat

Video pixel format

TRTC provides custom video capturing and rendering features.
For the custom capturing feature, you can use the following enumerated values to describe the pixel format of the video you capture.
For the custom rendering feature, you can specify the pixel format of the video you expect the SDK to call back.

Enum	Value	DESC
TRTCVideoPixelFormat_Unknown	0	Undefined format
TRTCVideoPixelFormat_I420	1	YUV420P (I420) format
TRTCVideoPixelFormat_Texture_2D	2	OpenGL 2D texture format
TRTCVideoPixelFormat_BGRA32	3	BGRA32 format
TRTCVideoPixelFormat_NV21	4	NV21 format
TRTCVideoPixelFormat_RGBA32	5	RGBA format

TRTCVideoBufferType

TRTCVideoBufferType

Video data transfer method

For custom capturing and rendering features, you need to use the following enumerated values to specify the method of transferring video data:

Method 1. This method uses memory buffer to transfer video data. It is efficient on iOS but inefficient on Android. It is the only method supported on Windows currently.

Method 2. This method uses texture to transfer video data. It is efficient on both iOS and Android but is not supported on Windows. To use this method, you should have a general familiarity with OpenGL programming.

Enum	Value	DESC
TRTCVideoBufferType_Unknown	0	Undefined transfer method
TRTCVideoBufferType_Buffer	1	Use memory buffer to transfer video data. iOS: `PixelBuffer` ; Android: `Direct Buffer` for JNI layer; Windows: memory data block.
TRTCVideoBufferType_Texture	3	Use OpenGL texture to transfer video data
TRTCVideoBufferType_TextureD3D11	4	Use D3D11 texture to transfer video data

TRTCVideoMirrorType

TRTCVideoMirrorType

Video mirror type

Video mirroring refers to the left-to-right flipping of the video image, especially for the local camera preview image. After mirroring is enabled, it can bring anchors a familiar "look into the mirror" experience.

Enum	Value	DESC
TRTCVideoMirrorType_Auto	0	Auto mode: mirror the front camera's image but not the rear camera's image (for mobile devices only).
TRTCVideoMirrorType_Enable	1	Mirror the images of both the front and rear cameras.
TRTCVideoMirrorType_Disable	2	Disable mirroring for both the front and rear cameras.

TRTCSnapshotSourceType

TRTCSnapshotSourceType

Data source of local video screenshot

The SDK can take screenshots from the following two data sources and save them as local files:

Video stream: the SDK screencaptures the native video content from the video stream. The screenshots are not controlled by the display of the rendering control.

Rendering layer: the SDK screencaptures the displayed video content from the rendering control, which can achieve the effect of WYSIWYG, but if the display area is too small, the screenshots will also be very small.

Enum	Value	DESC
TRTCSnapshotSourceTypeStream	0	The SDK screencaptures the native video content from the video stream. The screenshots are not controlled by the display of the rendering control.
TRTCSnapshotSourceTypeView	1	The SDK screencaptures the displayed video content from the rendering control, which can achieve the effect of WYSIWYG, but if the display area is too small, the screenshots will also be very small.
TRTCSnapshotSourceTypeCapture	2	The SDK screencaptures the capture video content from the capture control, which can capture the captured high-definition screenshots.

TRTCAppScene

TRTCAppScene

Use cases

TRTC features targeted optimizations for common audio/video application scenarios to meet the differentiated requirements in various verticals. The main scenarios can be divided into the following two categories:

Live streaming scenario (LIVE): including `LIVE` (audio + video) and `VoiceChatRoom` (pure audio).

In the live streaming scenario, users are divided into two roles: "anchor" and "audience". A single room can sustain up to 100,000 concurrent online users. This is suitable for live streaming to a large audience.

Real-Time scenario (RTC): including `VideoCall` (audio + video) and `AudioCall` (pure audio).

In the real-time scenario, there is no role difference between users, but a single room can sustain only up to 300 concurrent online users. This is suitable for small-scale real-time communication.

Enum	Value	DESC

TRTCAppSceneVideoCall	0	In the video call scenario, 720p and 1080p HD image quality is supported. A single room can sustain up to 300 concurrent online users, and up to 50 of them can speak simultaneously. Use cases: [one-to-one video call], [video conferencing with up to 300 participants], [online medical diagnosis], [small class], [video interview], etc.
TRTCAppSceneLIVE	1	In the interactive video live streaming scenario, mic can be turned on/off smoothly without waiting for switchover, and the anchor latency is as low as less than 300 ms. Live streaming to hundreds of thousands of concurrent users in the audience role is supported with the playback latency down to 1,000 ms. Use cases: [low-latency interactive live streaming], [big class], [anchor competition], [video dating room], [online interactive classroom], [remote training], [large-scale conferencing], etc. **Note** In this scenario, you must use the `role` field in `TRTCParams` to specify the role of the current user.
TRTCAppSceneAudioCall	2	Audio call scenario, where the `SPEECH` sound quality is used by default. A single room can sustain up to 300 concurrent online users, and up to 50 of them can speak simultaneously. Use cases: [one-to-one audio call], [audio conferencing with up to 300 participants], [audio chat], [online Werewolf], etc.
TRTCAppSceneVoiceChatRoom	3	In the interactive audio live streaming scenario, mic can be turned on/off smoothly without waiting for switchover, and the anchor latency is as low as less than 300 ms. Live streaming to hundreds of thousands of concurrent users in the audience role is supported with the playback latency down to 1,000 ms. Use cases: [audio club], [online karaoke room], [music live room], [FM radio], etc. **Note** In this scenario, you must use the `role` field in `TRTCParams` to specify the role of the current user.

TRTCRoleType

TRTCRoleType

Role

Role is applicable only to live streaming scenarios (`TRTCAppSceneLIVE` and `TRTCAppSceneVoiceChatRoom`). Users are divided into two roles:
 Anchor, who can publish their audio/video streams. There is a limit on the number of anchors. Up to 50 anchors are allowed to publish streams at the same time in one room.
 Audience, who can only listen to or watch audio/video streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room can sustain up to 100,000 concurrent online users in the audience role.

Enum	Value	DESC
TRTCRoleAnchor	20	An anchor can publish their audio/video streams. There is a limit on the number of anchors. Up to 50 anchors are allowed to publish streams at the same time in one room.
TRTCRoleAudience	21	Audience can only listen to or watch audio/video streams of anchors in the room. If they want to publish their streams, they need to switch to the "anchor" role first through switchRole. One room can sustain up to 100,000 concurrent online users in the audience role.

TRTCQosControlMode(Deprecated)

TRTCQosControlMode(Deprecated)

QoS control mode (disused)

Enum	Value	DESC
TRTCQosControlModeClient	0	Client-based control, which is for internal debugging of SDK and shall not be used by users.
TRTCQosControlModeServer	1	On-cloud control, which is the default and recommended mode.

TRTCVideoQosPreference

TRTCVideoQosPreference

Image quality preference

TRTC has two control modes in weak network environments: "ensuring clarity" and "ensuring smoothness". Both modes will give priority to the transfer of audio data.

Enum	Value	DESC
TRTCVideoQosPreferenceSmooth	1	Ensuring smoothness: in this mode, when the current network is unable to transfer a clear and smooth video image, the smoothness of the image will be given priority, but there will be blurs.
TRTCVideoQosPreferenceClear	2	Ensuring clarity (default value): in this mode, when the current network is unable to transfer a clear and smooth video image, the clarity of the image will be given priority, but there will be lags.

TRTCQuality

TRTCQuality

Network quality

TRTC evaluates the current network quality once every two seconds. The evaluation results are divided into six levels: `Excellent` indicates the best, and `Down` indicates the worst.

Enum	Value	DESC
TRTCQuality_Unknown	0	Undefined
TRTCQuality_Excellent	1	The current network is excellent
TRTCQuality_Good	2	The current network is good
TRTCQuality_Poor	3	The current network is fair
TRTCQuality_Bad	4	The current network is bad
TRTCQuality_Vbad	5	The current network is very bad
TRTCQuality_Down	6	The current network cannot meet the minimum requirements of TRTC

TRTCAVStatusType

TRTCAVStatusType

Audio/Video playback status

This enumerated type is used in the audio status changed API onRemoteAudioStatusUpdated and the video status changed API onRemoteVideoStatusUpdated to specify the current audio/video status.

Enum	Value	DESC
TRTCAVStatusStopped	0	Stopped
TRTCAVStatusPlaying	1	Playing
TRTCAVStatusLoading	2	Loading

TRTCAVStatusChangeReason

TRTCAVStatusChangeReason

Reasons for playback status changes

This enumerated type is used in the audio status changed API onRemoteAudioStatusUpdated and the video status changed API onRemoteVideoStatusUpdated to specify the reason for the current audio/video status change.

Enum	Value	DESC
TRTCAVStatusChangeReasonInternal	0	Default value
TRTCAVStatusChangeReasonBufferingBegin	1	The stream enters the `Loading` state due to network congestion
TRTCAVStatusChangeReasonBufferingEnd	2	The stream enters the `Playing` state after network recovery
TRTCAVStatusChangeReasonLocalStarted	3	As a start-related API was directly called locally, the stream enters the `Playing` state
TRTCAVStatusChangeReasonLocalStopped	4	As a stop-related API was directly called locally, the stream enters the `Stopped` state
TRTCAVStatusChangeReasonRemoteStarted	5	As the remote user started (or resumed) publishing the audio or video stream, the stream enters the `Loading` or `Playing` state
TRTCAVStatusChangeReasonRemoteStopped	6	As the remote user stopped (or paused) publishing the audio or video stream, the

		stream enters the "Stopped" state

TRTCAudioQuality

TRTCAudioQuality

Sound quality

TRTC provides three well-tuned modes to meet the differentiated requirements for sound quality in various verticals:
Speech mode (Speech): it is suitable for application scenarios that focus on human communication. In this mode, the audio transfer is more resistant, and TRTC uses various voice processing technologies to ensure the optimal smoothness even in weak network environments.
Music mode (Music): it is suitable for scenarios with demanding requirements for music. In this mode, the amount of transferred audio data is very large, and TRTC uses various technologies to ensure that the high-fidelity details of music signals can be restored in each frequency band.
Default mode (Default): it is between `Speech` and `Music`. In this mode, the reproduction of music is better than that in `Speech` mode, and the amount of transferred data is much lower than that in `Music` mode; therefore, this mode has good adaptability to various scenarios.

Enum	Value	DESC
TRTCAudioQualitySpeech	1	Speech mode: sample rate: 16 kHz; mono channel; bitrate: 16 Kbps. This mode has the best resistance among all modes and is suitable for audio call scenarios, such as online meeting and audio call.
TRTCAudioQualityDefault	2	Default mode: sample rate: 48 kHz; mono channel; bitrate: 50 Kbps. This mode is between the speech mode and the music mode as the default mode in the SDK and is recommended.
TRTCAudioQualityMusic	3	Music mode: sample rate: 48 kHz; full-band stereo; bitrate: 128 Kbps. This mode is suitable for scenarios where Hi-Fi music transfer is required, such as online karaoke and music live streaming.

TRTCAudioFrameFormat

TRTCAudioFrameFormat

Audio frame content format

Enum	Value	DESC
TRTCAudioFrameFormatNone	0	None
TRTCAudioFrameFormatPCM	Not Defined	Audio data in PCM format

TRTCAudioFrameOperationMode

TRTCAudioFrameOperationMode

Audio callback data operation mode

TRTC provides two modes of operation for audio callback data.
Read-only mode (ReadOnly): Get audio data only from the callback.
ReadWrite mode (ReadWrite): You can get and modify the audio data of the callback.

Enum	Value	DESC
TRTCAudioFrameOperationModeReadWrite	0	Read-write mode: You can get and modify the audio data of the callback, the default mode.
TRTCAudioFrameOperationModeReadOnly	1	Read-only mode: Get audio data from callback only.

TRTCLogLevel

TRTCLogLevel

Log level

Different log levels indicate different levels of details and number of logs. We recommend you set the log level to `TRTCLogLevelInfo` generally.

Enum	Value	DESC
TRTCLogLevelVerbose	0	Output logs at all levels
TRTCLogLevelDebug	1	Output logs at the DEBUG, INFO, WARNING, ERROR, and FATAL levels
TRTCLogLevelInfo	2	Output logs at the INFO, WARNING, ERROR, and FATAL levels

TRTCLogLevelWarn	3	Output logs at the WARNING, ERROR, and FATAL levels
TRTCLogLevelError	4	Output logs at the ERROR and FATAL levels
TRTCLogLevelFatal	5	Output logs at the FATAL level
TRTCLogLevelNone	6	Do not output any SDK logs

TRTCScreenCaptureSourceType

TRTCScreenCaptureSourceType

Screen sharing target type (for desktops only)

Enum	Value	DESC
TRTCScreenCaptureSourceTypeUnknown	-1	Undefined
TRTCScreenCaptureSourceTypeWindow	0	The screen sharing target is the window of an application
TRTCScreenCaptureSourceTypeScreen	1	The screen sharing target is the entire screen
TRTCScreenCaptureSourceTypeCustom	2	The screen sharing target is a user-defined data source

TRTCTranscodingConfigMode

TRTCTranscodingConfigMode

Layout mode of On-Cloud MixTranscoding

TRTC's On-Cloud MixTranscoding service can mix multiple audio/video streams in the room into one stream. Therefore, you need to specify the layout scheme of the video images. The following layout modes are provided:

Enum	Value	DESC
TRTCTranscodingConfigMode_Unknown	0	Undefined
TRTCTranscodingConfigMode_Manual	1	Manual layout mode In this mode, you need to specify the precise position of each video image. This mode has the highest degree of

		freedom, but its ease of use is the worst: You need to enter all the parameters in `TRTCTranscodingConfig` , including the position coordinates of each video image (TRTCMixUser). You need to listen on the `onUserVideoAvailable()` and `onUserAudioAvailable()` event callbacks in `TRTCCloudDelegate` and constantly adjust the `mixUsers` parameter according to the audio/video status of each user with mic on in the current room.
TRTCTranscodingConfigMode_Template_PureAudio	2	Pure audio mode This mode is suitable for pure audio scenarios such as audio call (AudioCall) and audio chat room (VoiceChatRoom). You only need to set it once through the `setMixTranscodingConfig()` API after room entry, and then the SDK will automatically mix the audio of all mic-on users in the room into the current user's live stream. You don't need to set the `mixUsers` parameter in `TRTCTranscodingConfig` ; instead, you only need to set the `audioSampleRate` , `audioBitrate` and `audioChannels` parameters.
TRTCTranscodingConfigMode_Template_PresetLayout	3	Preset layout mode This is the most popular layout mode, because it allows you to set the position of each video image in advance through placeholders, and then the SDK automatically adjusts it dynamically according to the number of video images in the room.

		In this mode, you still need to set the `mixUsers` parameter, but you can set `userId` as a "placeholder". Placeholder values include: "$PLACE_HOLDER_REMOTE$": image of remote user. Multiple images can be set. "$PLACE_HOLDER_LOCAL_MAIN$": local camera image. Only one image can be set. "$PLACE_HOLDER_LOCAL_SUB$": local screen sharing image. Only one image can be set. In this mode, you don't need to listen on the `onUserVideoAvailable()` and `onUserAudioAvailable()` callbacks in `TRTCCloudDelegate` to make real-time adjustments. Instead, you only need to call `setMixTranscodingConfig()` once after successful room entry. Then, the SDK will automatically populate the placeholders you set with real `userId` values.
TRTCTranscodingConfigMode_Template_ScreenSharing	4	Screen sharing mode This mode is suitable for screen sharing-based use cases such as online education and supported only by the SDKs for Windows and macOS. In this mode, the SDK will first build a canvas according to the target resolution you set (through the `videoWidth` and `videoHeight` parameters). Before the teacher enables screen sharing, the SDK will scale up the teacher's camera image and draw it onto the canvas. After the teacher enables screen sharing, the SDK will draw the video image shared on the screen onto the same canvas.

		The purpose of this layout mode is to ensure consistency in the output resolution of the mixtranscoding module and avoid problems with blurred screen during course replay and webpage playback (web players don't support adjustable resolution). Meanwhile, the audio of mic-on students will be mixed into the teacher's audio/video stream by default. Video content is primarily the shared screen in teaching mode, and it is a waste of bandwidth to transfer camera image and screen image at the same time. Therefore, the recommended practice is to directly draw the camera image onto the current screen through the `setLocalVideoRenderCallback` API. In this mode, you don't need to set the `mixUsers` parameter in `TRTCTranscodingConfig`, and the SDK will not mix students' images so as not to interfere with the screen sharing effect. You can set width x height in `TRTCTranscodingConfig` to 0 px x 0 px, and the SDK will automatically calculate a suitable resolution based on the aspect ratio of the user's current screen. If the teacher's current screen width is less than or equal to 1920 px, the SDK will use the actual resolution of the teacher's current screen. If the teacher's current screen width is greater than 1920 px, the SDK will select one of the three resolutions of 1920x1080 (16:9), 1920x1200 (16:10), and 1920x1440 (4:3) according to the current screen aspect ratio.

TRTCRecordType

TRTCRecordType

Media recording type

This enumerated type is used in the local media recording API startLocalRecording to specify whether to record audio/video files or pure audio files.

Enum	Value	DESC
TRTCLocalRecordType_Audio	0	Record audio only
TRTCLocalRecordType_Video	1	Record video only
TRTCLocalRecordType_Both	2	Record both audio and video

TRTCMixInputType

TRTCMixInputType

Stream mix input type

Enum	Value	DESC
TRTCMixInputTypeUndefined	0	Default. Considering the compatibility with older versions, if you specify the inputType as Undefined, the SDK will determine the stream mix input type according to the value of the `pureAudio` parameter
TRTCMixInputTypeAudioVideo	1	Mix both audio and video
TRTCMixInputTypePureVideo	2	Mix video only
TRTCMixInputTypePureAudio	3	Mix audio only
TRTCMixInputTypeWatermark	4	Mix watermark In this case, you don't need to specify the `userId` parameter, but you need to specify the `image` parameter. It is recommended to use png format.

TRTCWaterMarkSrcType

TRTCWaterMarkSrcType

Watermark image source type

Enum	Value	DESC
TRTCWaterMarkSrcTypeFile	0	Path of the image file, which can be in BMP, GIF, JPEG, PNG, TIFF, Exif, WMF, or EMF format
TRTCWaterMarkSrcTypeBGRA32	1	Memory block in BGRA32 format
TRTCWaterMarkSrcTypeRGBA32	2	Memory block in RGBA32 format

TRTCAudioRecordingContent

TRTCAudioRecordingContent

Audio recording content type

This enumerated type is used in the audio recording API startAudioRecording to specify the content of the recorded audio.

Enum	Value	DESC
TRTCAudioRecordingContentAll	0	Record both local and remote audio
TRTCAudioRecordingContentLocal	1	Record local audio only
TRTCAudioRecordingContentRemote	2	Record remote audio only

TRTCPublishMode

TRTCPublishMode

The publishing mode

This enum type is used by the publishing API startPublishMediaStream.
TRTC can mix multiple streams in a room and publish the mixed stream to a CDN or to a TRTC room. It can also publish the stream of the local user to Tencent Cloud or a third-party CDN.
You can specify one of the following publishing modes to use:

Enum	Value	DESC
TRTCPublishModeUnknown	0	Undefined

TRTCPublishBigStreamToCdn	1	Use this parameter to publish the primary stream (TRTCVideoStreamTypeBig) in the room to Tencent Cloud or a third-party CDN (only RTMP is supported).
TRTCPublishSubStreamToCdn	2	Use this parameter to publish the substream (TRTCVideoStreamTypeSub) in the room to Tencent Cloud or a third-party CDN (only RTMP is supported).
TRTCPublishMixStreamToCdn	3	Use this parameter together with the encoding parameter TRTCStreamEncoderParam and On-Cloud MixTranscoding parameter TRTCStreamMixingConfig to transcode the streams you specify and publish the mixed stream to Tencent Cloud or a third-party CDN (only RTMP is supported).
TRTCPublishMixStreamToRoom	4	Use this parameter together with the encoding parameter TRTCStreamEncoderParam and On-Cloud MixTranscoding parameter TRTCStreamMixingConfig to transcode the streams you specify and publish the mixed stream to the room you specify. Use `TRTCUser` in TRTCPublishTarget to specify the robot that publishes the transcoded stream to a TRTC room.

TRTCEncryptionAlgorithm

TRTCEncryptionAlgorithm

Encryption Algorithm

This enumeration type is used for media stream private encryption algorithm selection.

Enum	Value	DESC
TRTCEncryptionAlgorithmAes128Gcm	0	AES GCM 128。
TRTCEncryptionAlgorithmAes256Gcm	1	AES GCM 256。

TRTCSpeedTestScene

TRTCSpeedTestScene

Speed Test Scene

This enumeration type is used for speed test scene selection.

Enum	Value	DESC
TRTCSpeedTestScene_DelayTesting	1	Delay testing.
TRTCSpeedTestScene_DelayAndBandwidthTesting	2	Delay and bandwidth testing.
TRTCSpeedTestScene_OnlineChorusTesting	3	Online chorus testing.

TRTCGravitySensorAdaptiveMode

TRTCGravitySensorAdaptiveMode

Set the adaptation mode of gravity sensing (only applicable to mobile terminals)

Enum	Value	DESC
TRTCGravitySensorAdaptiveMode_Disable	0	Turn off the gravity sensor and make a decision based on the current acquisition resolution and the set encoding resolution. If the two are inconsistent, rotate 90 degrees to ensure the maximum frame.
TRTCGravitySensorAdaptiveMode_FillByCenterCrop	1	Turn on the gravity sensor to always ensure that the remote screen image is positive. When the intermediate process needs to deal with inconsistent resolutions, use the center cropping mode.
TRTCGravitySensorAdaptiveMode_FitWithBlackBorder	2	Turn on the gravity sensor to always ensure that the remote screen image is positive. When the resolution needs to be processed inconsistently in the intermediate process, use the superimposed black border mode.

TRTCParams

TRTCParams

Room entry parameters

As the room entry parameters in the TRTC SDK, these parameters must be correctly set so that the user can successfully enter the audio/video room specified by `roomId` or `strRoomId` .

For historical reasons, TRTC supports two types of room IDs: `roomId` and `strRoomId` .

Note: do not mix `roomId` and `strRoomId` , because they are not interchangeable. For example, the number `123` and the string `123` are two completely different rooms in TRTC.

EnumType	DESC
businessInfo	Field description: business data, which is optional. This field is needed only by some advanced features. Recommended value: do not set this field on your own.
privateMapKey	Field description: permission credential used for permission control, which is optional. If you want only users with the specified `userId` values to enter a room, you need to use `privateMapKey` to restrict the permission. Recommended value: we recommend you use this parameter only if you have high security requirements. For more information, please see Enabling Advanced Permission Control.
role	Field description: role in the live streaming scenario, which is applicable only to the live streaming scenario (TRTCAppSceneLIVE or TRTCAppSceneVoiceChatRoom) but doesn't take effect in the call scenario. Recommended value: default value: anchor (TRTCRoleAnchor).
roomId	Field description: numeric room ID. Users (userId) in the same room can see one another and make audio/video calls. Recommended value: value range: 1–4294967294. @note `roomId` and `strRoomId` are mutually exclusive. If you decide to use `strRoomId` , then `roomId` should be entered as 0. If both are entered, `roomId` will be used. **Note** do not mix `roomId` and `strRoomId` , because they are not interchangeable. For example, the number `123` and the string `123` are two completely different rooms in TRTC.
sdkAppId	Field description: application ID, which is required. Tencent Cloud generates bills based on `sdkAppId` . Recommended value: the ID can be obtained on the account information page in the TRTC console after the corresponding application is created.

strRoomId	Field description: string-type room ID. Users (userId) in the same room can see one another and make audio/video calls. @note `roomId` and `strRoomId` are mutually exclusive. If you decide to use `strRoomId` , then `roomId` should be entered as 0. If both are entered, `roomId` will be used. **Note** do not mix `roomId` and `strRoomId` , because they are not interchangeable. For example, the number `123` and the string `123` are two completely different rooms in TRTC. Recommended value: the length limit is 64 bytes. The following 89 characters are supported: Uppercase and lowercase letters (a–z and A–Z) Digits (0–9) Space, "!", "#", "$", "%", "&", "(", ")", "+", "-", ":", ";", "<", "=", ".", ">", "?", "@", "[", "]", "^", "_", "{", "}", "|", "~", and ",".
streamId	Field description: specified `streamId` in Tencent Cloud CSS, which is optional. After setting this field, you can play back the user's audio/video stream on Tencent Cloud CSS CDN through a standard pull scheme (FLV or HLS). Recommended value: this parameter can contain up to 64 bytes and can be left empty. We recommend you use `sdkappid_roomid_userid_main` as the `streamid` , which is easier to identify and will not cause conflicts in your multiple applications. **Note** to use Tencent Cloud CSS CDN, you need to enable the auto-relayed live streaming feature on the "Function Configuration" page in the console first. For more information, please see CDN Relayed Live Streaming.
userDefineRecordId	Field description: on-cloud recording field, which is optional and used to specify whether to record the user's audio/video stream in the cloud. For more information, please see On-Cloud Recording and Playback. Recommended value: it can contain up to 64 bytes. Letters (a–z and A–Z), digits (0–9), underscores, and hyphens are allowed. Scheme 1. Manual recording 1. Enable on-cloud recording in "Application Management" > "On-cloud Recording Configuration" in the console. 2. Set "Recording Mode" to "Manual Recording". 3. After manual recording is set, in a TRTC room, only users with the `userDefineRecordId` parameter set will have video recording files in the cloud, while users without this parameter set will not. 4. The recording file will be named in the format of "userDefineRecordId_start time_end time" in the cloud. Scheme 2. Auto-recording 1. You need to enable on-cloud recording in "Application Management" > "On-cloud Recording Configuration" in the console.

	2. Set "Recording Mode" to "Auto-recording". 3. After auto-recording is set, any user who upstreams audio/video in a TRTC room will have a video recording file in the cloud. 4. The file will be named in the format of "userDefineRecordId_start time_end time". If `userDefineRecordId` is not specified, the file will be named in the format of "streamId_start time_end time".
userId	Field description: user ID, which is required. It is the `userId` of the local user in UTF-8 encoding and acts as the username. Recommended value: if the ID of a user in your account system is "mike", `userId` can be set to "mike".
userSig	Field description: user signature, which is required. It is the authentication signature corresponding to the current `userId` and acts as the login password for Tencent Cloud services. Recommended value: for the calculation method, please see UserSig.

TRTCVideoEncParam

TRTCVideoEncParam

Video encoding parameters

These settings determine the quality of image viewed by remote users as well as the image quality of recorded video files in the cloud.

EnumType	DESC
enableAdjustRes	Field description: whether to allow dynamic resolution adjustment. Once enabled, this field will affect on-cloud recording. Recommended value: this feature is suitable for scenarios that don't require on-cloud recording. After it is enabled, the SDK will intelligently select a suitable resolution according to the current network conditions to avoid the inefficient encoding mode of "large resolution + small bitrate". **Note** default value: false. If you need on-cloud recording, please do not enable this feature, because if the video resolution changes, the MP4 file recorded in the cloud cannot be played back normally by common players.
minVideoBitrate	Field description: minimum video bitrate. The SDK will reduce the bitrate to as low as the value specified by `minVideoBitrate` to ensure the smoothness only if the network conditions are poor. Note: default value: 0, indicating that a reasonable value of the lowest bitrate will be automatically calculated by the SDK according to the resolution you specify.

	Recommended value: you can set the `videoBitrate` and `minVideoBitrate` parameters at the same time to restrict the SDK's adjustment range of the video bitrate: If you want to "ensure clarity while allowing lag in weak network environments", you can set `minVideoBitrate` to 60% of `videoBitrate` . If you want to "ensure smoothness while allowing blur in weak network environments", you can set `minVideoBitrate` to a low value, for example, 100 Kbps. If you set `videoBitrate` and `minVideoBitrate` to the same value, it is equivalent to disabling the adaptive adjustment capability of the SDK for the video bitrate.
resMode	Field description: resolution mode (landscape/portrait) Recommended value: for mobile platforms (iOS and Android), `Portrait` is recommended; for desktop platforms (Windows and macOS), `Landscape` is recommended. **Note** to use a portrait resolution, please specify `resMode` as `Portrait` ; for example, when used together with `Portrait` , 640x360 represents 360x640.
videoBitrate	Field description: target video bitrate. The SDK encodes streams at the target video bitrate and will actively reduce the bitrate only in weak network environments. Recommended value: please see the optimal bitrate for each specification in `TRTCVideoResolution` . You can also slightly increase the optimal bitrate. For example, `TRTCVideoResolution_1280_720` corresponds to the target bitrate of 1,200 Kbps. You can also set the bitrate to 1,500 Kbps for higher definition. **Note** you can set the `videoBitrate` and `minVideoBitrate` parameters at the same time to restrict the SDK's adjustment range of the video bitrate: If you want to "ensure clarity while allowing lag in weak network environments", you can set `minVideoBitrate` to 60% of `videoBitrate` . If you want to "ensure smoothness while allowing blur in weak network environments", you can set `minVideoBitrate` to a low value, for example, 100 Kbps. If you set `videoBitrate` and `minVideoBitrate` to the same value, it is equivalent to disabling the adaptive adjustment capability of the SDK for the video bitrate.
videoFps	Field description: video capturing frame rate Recommended value: 15 or 20 fps. If the frame rate is lower than 5 fps, there will be obvious lagging; if lower than 10 fps but higher than 5 fps, there will be slight lagging; if higher than 20 fps, the bandwidth will be wasted (the frame rate of movies is generally 24 fps). **Note**

	the front cameras on certain Android phones do not support a capturing frame rate higher than 15 fps. For some Android phones that focus on beautification features, the capturing frame rate of the front cameras may be lower than 10 fps.
videoResolution	Field description: video resolution Recommended value For mobile video call, we recommend you select a resolution of 360x640 or below and select `Portrait` (portrait resolution) for `resMode` . For mobile live streaming, we recommend you select a resolution of 540x960 and select `Portrait` (portrait resolution) for `resMode` . For desktop platforms (Windows and macOS), we recommend you select a resolution of 640x360 or above and select `Landscape` (landscape resolution) for `resMode` . **Note** to use a portrait resolution, please specify `resMode` as `Portrait` ; for example, when used together with `Portrait` , 640x360 represents 360x640.

TRTCNetworkQosParam

TRTCNetworkQosParam

Network QoS control parameter set

Network QoS control parameter. The settings determine the QoS control policy of the SDK in weak network conditions (e.g., whether to "ensure clarity" or "ensure smoothness").

EnumType	DESC
controlMode	Field description: QoS control mode (disused) Recommended value: on-cloud control **Note** please set the on-cloud control mode (TRTCQosControlModeServer).
preference	Field description: whether to ensure smoothness or clarity Recommended value: ensuring clarity **Note** this parameter mainly affects the audio/video performance of TRTC in weak network environments: Ensuring smoothness: in this mode, when the current network is unable to transfer a clear and smooth video image, the smoothness of the image will be given priority, but there will be blurs. See TRTCVideoQosPreferenceSmooth Ensuring clarity (default value): in this mode, when the current network is unable to transfer a clear and smooth video image, the clarity of the image will be given priority, but there will be lags. See TRTCVideoQosPreferenceClear

TRTCRenderParams

TRTCRenderParams

Rendering parameters of video image

You can use these parameters to control the video image rotation angle, fill mode, and mirror mode.

EnumType	DESC
fillMode	Field description: image fill mode Recommended value: fill (the image may be stretched or cropped) or fit (there may be black bars in unmatched areas). Default value: TRTCVideoFillMode_Fill
mirrorType	Field description: image mirror mode Recommended value: default value: TRTCVideoMirrorType_Auto
rotation	Field description: clockwise image rotation angle Recommended value: rotation angles of 90, 180, and 270 degrees are supported. Default value: TRTCVideoRotation_0

TRTCQuality

TRTCQuality

Network quality

This indicates the quality of the network. You can use it to display the network quality of each user on the UI.

EnumType	DESC
quality	Network quality
userId	User ID

TRTCVolumeInfo

TRTCVolumeInfo

Volume

This indicates the audio volume value. You can use it to display the volume of each user in the UI.

EnumType	DESC

pitch	The local user's vocal frequency (unit: Hz), the value range is [0 - 4000]. For remote users, this value is always 0.
spectrumData	Audio spectrum data, which divides the sound frequency into 256 frequency domains, spectrumData records the energy value of each frequency domain, The value range of each energy value is [-300, 0] in dBFS. **Note** The local spectrum is calculated using the audio data before encoding, which will be affected by the capture volume, BGM, etc.; the remote spectrum is calculated using the received audio data, and operations such as adjusting the remote playback volume locally will not affect it.
spectrumDataLength	The length of recorded audio spectrum data, which is 256.
userId	`userId` of the speaker. An empty value indicates the local user.
vad	Vad result of the local user. 0: not speech 1: speech.
volume	Volume of the speaker. Value range: 0–100.

TRTCSpeedTestParams

TRTCSpeedTestParams

Network speed testing parameters

You can test the network speed through the startSpeedTest: interface before the user enters the room (this API cannot be called during a call).

EnumType	DESC
expectedDownBandwidth	Expected downstream bandwidth (kbps, value range: 10 to 5000, no downlink bandwidth test when it is 0). **Note** When the parameter `scene` is set to `TRTCSpeedTestScene_OnlineChorusTesting` , in order to obtain more accurate information such as rtt / jitter, the value range is limited to 10 ~ 1000.
expectedUpBandwidth	Expected upstream bandwidth (kbps, value range: 10 to 5000, no uplink bandwidth test when it is 0). **Note** When the parameter `scene` is set to `TRTCSpeedTestScene_OnlineChorusTesting` , in order to obtain

	more accurate information such as rtt / jitter, the value range is limited to 10 ~ 1000.
scene	Speed test scene.
sdkAppId	Application identification, please refer to the relevant instructions in TRTCParams.
userId	User identification, please refer to the relevant instructions in TRTCParams.
userSig	User signature, please refer to the relevant instructions in TRTCParams.

TRTCSpeedTestResult

TRTCSpeedTestResult

Network speed test result

The startSpeedTest: API can be used to test the network speed before a user enters a room (this API cannot be called during a call).

EnumType	DESC
availableDownBandwidth	Downstream bandwidth (in kbps, -1: invalid value).
availableUpBandwidth	Upstream bandwidth (in kbps, -1: invalid value).
downJitter	Downlink data packet jitter (ms) refers to the stability of data communication in the user's current network environment. The smaller the value, the better. The normal value range is 0ms - 100ms. -1 means that the speed test failed to obtain an effective value. Generally, the Jitter of the WiFi network will be slightly larger than that of the 4G/5G environment.
downLostRate	Downstream packet loss rate between 0 and 1.0. For example, 0.2 indicates that 2 data packets may be lost in every 10 packets received from the server.
errMsg	Error message for network speed test.
ip	Server IP address.
quality	Network quality, which is tested and calculated based on the internal evaluation algorithm. For more information, please see TRTCQuality
rtt	Delay in milliseconds, which is the round-trip time between the current device and TRTC server. The smaller the value, the better. The normal

	value range is 10–100 ms.
success	Whether the network speed test is successful.
upJitter	Uplink data packet jitter (ms) refers to the stability of data communication in the user's current network environment. The smaller the value, the better. The normal value range is 0ms - 100ms. -1 means that the speed test failed to obtain an effective value. Generally, the Jitter of the WiFi network will be slightly larger than that of the 4G/5G environment.
upLostRate	Upstream packet loss rate between 0 and 1.0. For example, 0.3 indicates that 3 data packets may be lost in every 10 packets sent to the server.

TRTCTexture

TRTCTexture

Video texture data

EnumType	DESC
glContext	Field description: The OpenGL context to which the texture corresponds, for Windows and Android.
glTextureId	Field description: video texture ID
}	Field description: The D3D11 texture, which is the pointer of ID3D11Texture2D, only for Windows.

TRTCVideoFrame

TRTCVideoFrame

Video frame information

`TRTCVideoFrame` is used to describe the raw data of a frame of the video image, which is the image data before frame encoding or after frame decoding.

EnumType	DESC
bufferType	Field description: video data structure type
data	Field description: video data when `bufferType` is TRTCVideoBufferType_Buffer, which carries the memory data blocks for the C++ layer.

height	Field description: video height Recommended value: please enter the height of the video data passed in.
length	Field description: video data length in bytes. For I420, length = width * height * 3 / 2; for BGRA32, length = width * height * 4.
rotation	Field description: clockwise rotation angle of video pixels
texture	Field description: video data when `bufferType` is TRTCVideoBufferType_Texture, which carries the texture data used for OpenGL rendering.
timestamp	Field description: video frame timestamp in milliseconds Recommended value: this parameter can be set to 0 for custom video capturing. In this case, the SDK will automatically set the `timestamp` field. However, please "evenly" set the calling interval of `sendCustomVideoData` .
videoFormat	Field description: video pixel format
width	Field description: video width Recommended value: please enter the width of the video data passed in.

TRTCAudioFrame

TRTCAudioFrame

Audio frame data

EnumType	DESC
audioFormat	Field description: audio frame format
channel	Field description: number of sound channels
data	Field description: audio data
extraData	Field description: extra data in audio frame, message sent by remote users through `onLocalProcessedAudioFrame` that add to audio frame will be callback through this field.
extraDataLength	Field description: extra data length
length	Field description: audio data length
sampleRate	Field description: sample rate

timestamp	Field description: timestamp in ms

TRTCMixUser

TRTCMixUser

Description information of each video image in On-Cloud MixTranscoding

`TRTCMixUser` is used to specify the location, size, layer, and stream type of each video image in On-Cloud MixTranscoding.

EnumType	DESC
image	Field description: specify the placeholder or watermark image. The placeholder image will be displayed when there is no upstream video.A watermark image is a semi-transparent image posted in the mixed image, and this image will always be overlaid on the mixed image. When the `inputType` field is set to TRTCMixInputTypePureAudio, the image is a placeholder image, and you need to specify `userId` . When the `inputType` field is set to TRTCMixInputTypeWatermark, the image is a watermark image, and you don't need to specify `userId` . Recommended value: default value: null, indicating not to set the placeholder or watermark image. **Note** TRTC's backend service will mix the image specified by the URL address into the final stream.URL link length is limited to 512 bytes. The image size is limited to 10MB.Support png, jpg, jpeg, bmp format. Take effects iff the `inputType` field is set to TRTCMixInputTypePureAudio or TRTCMixInputTypeWatermark.
inputType	Field description: specify the mixed content of this stream (audio only, video only, audio and video, or watermark). Recommended value: default value: TRTCMixInputTypeUndefined. **Note** When specifying `inputType` as TRTCMixInputTypeUndefined and specifying `pureAudio` to YES, it is equivalent to setting `inputType` to `TRTCMixInputTypePureAudio` . When specifying `inputType` as TRTCMixInputTypeUndefined and specifying `pureAudio` to NO, it is equivalent to setting `inputType` to `TRTCMixInputTypeAudioVideo` . When specifying `inputType` as TRTCMixInputTypeWatermark, you don't need to specify the `userId` field, but you need to specify the `image` field.
pureAudio	Field description: specify whether this stream mixes audio only Recommended value: default value: false

	Note this field has been disused. We recommend you use the new field `inputType` introduced in v8.5.
rect	Field description: specify the coordinate area of this video image in px
renderMode	Field description: specify the display mode of this stream. Recommended value: default value: 0. 0 is cropping, 1 is zooming, 2 is zooming and displaying black background. **Note** image doesn't support setting `renderMode` temporarily, the default display mode is forced stretch.
roomId	Field description: ID of the room where this audio/video stream is located (an empty value indicates the local room ID)
soundLevel	Field description: specify the target volumn level of On-Cloud MixTranscoding. (value range: 0-100) Recommended value: default value: 100.
streamType	Field description: specify whether this video image is the primary stream image (TRTCVideoStreamTypeBig) or substream image (TRTCVideoStreamTypeSub).
userId	Field description: user ID
zOrder	Field description: specify the level of this video image (value range: 1–15; the value must be unique)

TRTCTranscodingConfig

TRTCTranscodingConfig

Layout and transcoding parameters of On-Cloud MixTranscoding

These parameters are used to specify the layout position information of each video image and the encoding parameters of mixtranscoding during On-Cloud MixTranscoding.

EnumType	DESC
appId	Field description: `appId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `appId` in `Relayed Live Streaming Info` .
audioBitrate	Field description: specify the target audio bitrate of On-Cloud MixTranscoding Recommended value: default value: 64 Kbps. Value range: [32,192].

audioChannels	Field description: specify the number of sound channels of On-Cloud MixTranscoding Recommended value: default value: 1, which means mono channel. Valid values: 1: mono channel; 2: dual channel.
audioCodec	Field description: specify the audio encoding type of On-Cloud MixTranscoding Recommended value: default value: 0, which means LC-AAC. Valid values: 0: LC-AAC; 1: HE-AAC; 2: HE-AACv2. **Note** HE-AAC and HE-AACv2 only support [48000, 44100, 32000, 24000, 16000] sample rate. HE-AACv2 only support dual channel. HE-AAC and HE-AACv2 take effects iff the output streamId is specified.
audioSampleRate	Field description: specify the target audio sample rate of On-Cloud MixTranscoding Recommended value: default value: 48000 Hz. Valid values: 12000 Hz, 16000 Hz, 22050 Hz, 24000 Hz, 32000 Hz, 44100 Hz, 48000 Hz.
backgroundColor	Field description: specify the background color of the mixed video image. Recommended value: default value: 0x000000, which means black and is in the format of hex number; for example: "0x61B9F1" represents the RGB color (97,158,241).
backgroundImage	Field description: specify the background image of the mixed video image. **Recommended value: default value: null, indicating not to set the background image. **Note** TRTC's backend service will mix the image specified by the URL address into the final stream.URL link length is limited to 512 bytes. The image size is limited to 10MB.Support png, jpg, jpeg, bmp format.
bizId	Field description: `bizId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `bizId` in `Relayed Live Streaming Info` .
mixUsersArray	Field description: specify the position, size, layer, and stream type of each video image in On-Cloud MixTranscoding Recommended value: this field is an array in `TRTCMixUser` type, where each element represents the information of a video image.
mixUsersArraySize	Field description: number of elements in the `mixUsersArray` array
mode	Field description: layout mode Recommended value: please choose a value according to your business needs. The preset mode has better applicability.

streamId	Field description: ID of the live stream output to CDN Recommended value: default value: null, that is, the audio/video streams in the room will be mixed into the audio/video stream of the caller of this API. If you don't set this parameter, the SDK will execute the default logic, that is, it will mix the multiple audio/video streams in the room into the audio/video stream of the caller of this API, i.e., A + B => A. If you set this parameter, the SDK will mix the audio/video streams in the room into the live stream you specify, i.e., A + B => C (C is the `streamId` you specify).
videoBitrate	Field description: specify the target video bitrate (Kbps) of On-Cloud MixTranscoding Recommended value: if you enter 0, TRTC will estimate a reasonable bitrate value based on `videoWidth` and `videoHeight` . You can also refer to the recommended bitrate value in the video resolution enumeration definition (in the comment section).
videoFramerate	Field description: specify the target video frame rate (fps) of On-Cloud MixTranscoding Recommended value: default value: 15 fps. Value range: (0,30].
videoGOP	Field description: specify the target video keyframe interval (GOP) of On-Cloud MixTranscoding Recommended value: default value: 2 (in seconds). Value range: [1,8].
videoHeight	Field description: specify the target resolution (height) of On-Cloud MixTranscoding Recommended value: 640 px. If you only mix audio streams, please set both `width` and `height` to 0; otherwise, there will be a black background in the live stream after mixtranscoding.
videoSeiParams	Field description: SEI parameters. default value: null **Note** the parameter is passed in the form of a JSON string. Here is an example to use it: ``json { "payLoadContent":"xxx", "payloadType":5, "payloadUuid":"1234567890abcdef1234567890abcdef", "interval":1000, "followIdr":false } `` The currently supported fields and their meanings are as follows: payloadContent: Required. The payload content of the passthrough SEI, which cannot be empty. payloadType: Required. The type of the SEI message, with a value range of 5 or an integer within the range of [100, 254] (excluding 244, which is an internally

	defined timestamp SEI). payloadUuid: Required when payloadType is 5, and ignored in other cases. The value must be a 32-digit hexadecimal number. interval: Optional, default is 1000. The sending interval of the SEI, in milliseconds. followIdr: Optional, default is false. When this value is true, the SEI will be ensured to be carried when sending a key frame, otherwise it is not guaranteed.
videoWidth	Field description: specify the target resolution (width) of On-Cloud MixTranscoding Recommended value: 360 px. If you only mix audio streams, please set both `width` and `height` to 0; otherwise, there will be a black background in the live stream after mixtranscoding.

TRTCPublishCDNParam

TRTCPublishCDNParam

Push parameters required to be set when publishing audio/video streams to non-Tencent Cloud CDN

TRTC's backend service supports publishing audio/video streams to third-party live CDN service providers through the standard RTMP protocol.

If you use the Tencent Cloud CSS CDN service, you don't need to care about this parameter; instead, just use the startPublish API.

EnumType	DESC
appId	Field description: `appId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `appId` in `Relayed Live Streaming Info` .
bizId	Field description: `bizId` of Tencent Cloud CSS Recommended value: please click `Application Management` > `Application Information` in the TRTC console and get the `bizId` in `Relayed Live Streaming Info` .
streamId	Field description: specify the push address (in RTMP format) of this audio/video stream at the third-party live streaming service provider Recommended value: default value: null,that is, the audio/video streams in the room will be pushed to the target service provider of the caller of this API.
url	Field description: specify the push address (in RTMP format) of this audio/video stream at the third-party live streaming service provider Recommended value: the push URL rules vary greatly by service provider. Please enter a valid push URL according to the requirements of the target service provider. TRTC's

	backend server will push audio/video streams in the standard format to the third-party service provider according to the URL you enter. **Note** the push URL must be in RTMP format and meet the specifications of your target live streaming service provider; otherwise, the target service provider will reject the push requests from TRTC's backend service.

TRTCAudioRecordingParams

TRTCAudioRecordingParams

Local audio file recording parameters

This parameter is used to specify the recording parameters in the audio recording API startAudioRecording.

EnumType	DESC
filePath	Field description: storage path of the audio recording file, which is required. **Note** this path must be accurate to the file name and extension. The extension determines the format of the audio recording file. Currently, supported formats include PCM, WAV, and AAC. For example, if you specify the path as `mypath/record/audio.aac` , it means that you want the SDK to generate an audio recording file in AAC format.Please specify a valid path with read/write permissions; otherwise, the audio recording file cannot be generated.
maxDurationPerFile	Field description: `maxDurationPerFile` is the max duration of each recorded file segments, in milliseconds, with a minimum value of 10000. The default value is 0, indicating no segmentation.
recordingContent	Field description: Audio recording content type. Note: Record all local and remote audio by default.

TRTCLocalRecordingParams

TRTCLocalRecordingParams

Local media file recording parameters

This parameter is used to specify the recording parameters in the local media file recording API startLocalRecording.
The `startLocalRecording` API is an enhanced version of the `startAudioRecording` API. The former can record video files, while the latter can only record audio files.

EnumType	DESC
filePath	Field description: address of the recording file, which is required. Please ensure that the path is valid with read/write permissions; otherwise, the recording file cannot be generated. **Note** this path must be accurate to the file name and extension. The extension determines the format of the recording file. Currently, only the MP4 format is supported. For example, if you specify the path as `mypath/record/test.mp4` , it means that you want the SDK to generate a local video file in MP4 format. Please specify a valid path with read/write permissions; otherwise, the recording file cannot be generated.
interval	Field description: `interval` is the update frequency of the recording information in milliseconds. Value range: 1000–10000. Default value: -1, indicating not to call back
maxDurationPerFile	Field description: `maxDurationPerFile` is the max duration of each recorded file segments, in milliseconds, with a minimum value of 10000. The default value is 0, indicating no segmentation.
recordType	Field description: media recording type, which is `TRTCRecordTypeBoth` by default, indicating to record both audio and video.

TRTCSwitchRoomConfig

TRTCSwitchRoomConfig

Room switch parameter

This parameter is used for the room switch API switchRoom, which can quickly switch a user from one room to another.

EnumType	DESC
privateMapKey	Field description: permission credential used for permission control, which is optional. If you want only users with the specified `userId` values to enter a room, you need to use `privateMapKey` to restrict the permission. Recommended value: we recommend you use this parameter only if you have high security requirements. For more information, please see Enabling Advanced Permission Control.
roomId	Field description: numeric room ID, which is optional. Users in the same room can see one another and make audio/video calls.

	Recommended value: value range: 1-4294967294. **Note** either `roomId` or `strRoomId` must be entered. If both are entered, `roomId` will be used.
strRoomId	Field description: string-type room ID, which is optional. Users in the same room can see one another and make audio/video calls. **Note** either `roomId` or `strRoomId` must be entered. If both are entered, `roomId` will be used.
userSig	Field description: user signature, which is optional. It is the authentication signature corresponding to the current `userId` and acts as the login password. If you don't specify the newly calculated `userSig` during room switch, the SDK will continue to use the `userSig` you specified during room entry (enterRoom). This requires you to ensure that the old `userSig` is still within the validity period allowed by the signature at the moment of room switch; otherwise, room switch will fail. Recommended value: for the calculation method, please see UserSig.

TRTCAudioFrameDelegateFormat

TRTCAudioFrameDelegateFormat

Format parameter of custom audio callback

This parameter is used to set the relevant format (including sample rate and number of channels) of the audio data called back by the SDK in the APIs related to custom audio callback.

EnumType	DESC
channel	Field description: number of sound channels Recommended value: default value: 1, which means mono channel. Valid values: 1: mono channel; 2: dual channel.
mode	Field description: audio callback data operation mode Recommended value: TRTCAudioFrameOperationModeReadOnly, get audio data from callback only. The modes that can be set are TRTCAudioFrameOperationModeReadOnly, TRTCAudioFrameOperationModeReadWrite.
sampleRate	Field description: sample rate Recommended value: default value: 48000 Hz. Valid values: 16000, 32000, 44100, 48000.
samplesPerCall	Field description: number of sample points

	Recommended value: the value must be an integer multiple of sampleRate/100.

TRTCImageBuffer

TRTCImageBuffer

Structure for storing window thumbnails and icons.

EnumType	DESC
buffer	image content in BGRA format
height	image height
length	buffer size
width	image width

TRTCUser

TRTCUser

The users whose streams to publish

You can use this parameter together with the publishing destination parameter TRTCPublishTarget and On-Cloud MixTranscoding parameter TRTCStreamMixingConfig to transcode the streams you specify and publish the mixed stream to the destination you specify.

EnumType	DESC
intRoomId	**Description:** Numeric room ID. The room ID must be of the same type as that in TRTCParams. **Value:** Value range: 1-4294967294 **Note:** You cannot use both `intRoomId` and `strRoomId`. If you specify `strRoomId`, you need to set `intRoomId` to `0`. If you set both, only `intRoomId` will be used.
strRoomId	**Description:** String-type room ID. The room ID must be of the same type as that in TRTCParams. **Note:** You cannot use both `intRoomId` and `strRoomId`. If you specify `roomId`, you need to leave `strRoomId` empty. If you set both, only `intRoomId` will be used. **Value:** 64 bytes or shorter; supports the following character set (89 characters):

	Uppercase and lowercase letters (a-z and A-Z) Numbers (0-9) Space, "!", "#", "$", "%", "&", "(", ")", "+", "-", ":", ";", "<", "=", ".", ">", "?", "@", "[", "]", "^", "_", "{", "}", "|", "~", ","
userId	/**Description**: UTF-8-encoded user ID (required) **Value:** For example, if the ID of a user in your account system is "mike", set it to `mike` .

TRTCPublishCdnUrl

TRTCPublishCdnUrl

The destination URL when you publish to Tencent Cloud or a third-party CDN

This enum type is used by the publishing destination parameter TRTCPublishTarget of the publishing API startPublishMediaStream.

EnumType	DESC
isInternalLine	**Description:** Whether to publish to Tencent Cloud **Value:** The default value is `true` . **Note:** If the destination URL you set is provided by Tencent Cloud, set this parameter to `true` , and you will not be charged relaying fees.
rtmpUrl	**Description:** The destination URL (RTMP) when you publish to Tencent Cloud or a third-party CDN. **Value:** The URLs of different CDN providers may vary greatly in format. Please enter a valid URL as required by your service provider. TRTC's backend server will push audio/video streams in the standard format to the URL you provide. **Note:** The URL must be in RTMP format. It must also meet the requirements of your service provider, or your service provider may reject push requests from the TRTC backend.

TRTCPublishTarget

TRTCPublishTarget

The publishing destination

This enum type is used by the publishing API startPublishMediaStream.

EnumType	DESC

cdnUrlList	`Description:` The destination URLs (RTMP) when you publish to Tencent Cloud or third-party CDNs. `Note:` You don't need to set this parameter if you set the publishing mode to `TRTCPublishMixStreamToRoom` .
cdnUrlListSize	`Description:` The length of the `cdnUrlList` array. `Note:` You don't need to set this parameter if you set the publishing mode to `TRTCPublishMixStreamToRoom` .
mixStreamIdentity	`Description:` The information of the robot that publishes the transcoded stream to a TRTC room. `Note:` You need to set this parameter only if you set the publishing mode to `TRTCPublishMixStreamToRoom` . `Note:` After you set this parameter, the stream will be pushed to the room you specify. We recommend you set it to a special user ID to distinguish the robot from the anchor who enters the room via the TRTC SDK. `Note:` Users whose streams are transcoded cannot subscribe to the transcoded stream. `Note:` If you set the subscription mode (@link setDefaultStreamRecvMode}) to manual before room entry, you need to manage the streams to receive by yourself (normally, if you receive the transcoded stream, you need to unsubscribe from the streams that are transcoded). `Note:` If you set the subscription mode (setDefaultStreamRecvMode) to auto before room entry, users whose streams are not transcoded will receive the transcoded stream automatically and will unsubscribe from the users whose streams are transcoded. You call muteRemoteVideoStream and muteRemoteAudio to unsubscribe from the transcoded stream.
mode	`Description:` The publishing mode. `Value:` You can relay streams to a CDN, transcode streams, or publish streams to an RTC room. Select the mode that fits your needs. **Note** If you need to use more than one publishing mode, you can call startPublishMediaStream multiple times and set `TRTCPublishTarget` to a different value each time.You can use one mode each time you call the startPublishMediaStream) API. To modify the configuration, call updatePublishCDNStream.

TRTCVideoLayout

TRTCVideoLayout

The video layout of the transcoded stream

This enum type is used by the On-Cloud MixTranscoding parameter TRTCStreamMixingConfig of the publishing API startPublishMediaStream.
You can use this parameter to specify the position, size, layer, and stream type of each video in the transcoded stream.

EnumType	DESC
backgroundColor	`Description:` The background color of the mixed stream. `Value:` The value must be a hex number. For example, "0x61B9F1" represents the RGB color value (97,158,241). Default value: 0x000000 (black).
fillMode	`Description:` The rendering mode. `Value:` The rendering mode may be fill (the image may be stretched or cropped) or fit (there may be black bars). Default value: TRTCVideoFillMode_Fill.
fixedVideoStreamType	`Description:` Whether the video is the primary stream (TRTCVideoStreamTypeBig) or substream (e TRTCVideoStreamTypeSub).
fixedVideoUser	`Description:` The users whose streams are transcoded. **Note** If you do not specify TRTCUser (`userId` , `intRoomId` , `strRoomId`), the TRTC backend will automatically mix the streams of anchors who are sending audio/video in the room according to the video layout you specify.
placeHolderImage	`Description:` The URL of the placeholder image. If a user sends only audio, the image specified by the URL will be mixed during On-Cloud MixTranscoding. `Value:` This parameter is left empty by default, which means no placeholder image will be used. **Note** You need to specify the `userId` parameter in `fixedVideoUser` . The URL can be 512 bytes long at most, and the image must not exceed 2 MB. The image can be in PNG, JPG, JPEG, or BMP format. We recommend you use a semitransparent image in PNG format.
rect	`Description:` The coordinates (in pixels) of the video.
zOrder	`Description:` The layer of the video, which must be unique. Value range: 0-15.

TRTCWatermark

TRTCWatermark

The watermark layout

This enum type is used by the On-Cloud MixTranscoding parameter TRTCStreamMixingConfig of the publishing API startPublishMediaStream.

EnumType	DESC
rect	`Description:` The coordinates (in pixels) of the watermark.
watermarkUrl	`Description:` The URL of the watermark image. The image specified by the URL will be mixed during On-Cloud MixTranscoding. **Note** The URL can be 512 bytes long at most, and the image must not exceed 2 MB. The image can be in PNG, JPG, JPEG, or BMP format. We recommend you use a semitransparent image in PNG format.
zOrder	`Description:` The layer of the watermark, which must be unique. Value range: 0-15.

TRTCStreamEncoderParam

TRTCStreamEncoderParam

The encoding parameters

`Description:` This enum type is used by the publishing API startPublishMediaStream.
`Note:` This parameter is required if you set the publishing mode to `TRTCPublish_MixStream_ToCdn` or `TRTCPublish_MixStream_ToRoom` in TRTCPublishTarget.
`Note:` If you use the relay to CDN feature (the publishing mode set to `RTCPublish_BigStream_ToCdn` or `TRTCPublish_SubStream_ToCdn`), to improve the relaying stability and playback compatibility, we also recommend you set this parameter.

EnumType	DESC
audioEncodedChannelNum	`Description:` The sound channels of the stream to publish. `Value:` Valid values: 1 (mono channel); 2 (dual-channel). Default: 1.
audioEncodedCodecType	`Description:` The audio codec of the stream to publish. `Value:` Valid values: 0 (LC-AAC); 1 (HE-AAC); 2 (HE-AACv2). Default: 0.

	Note The audio sample rates supported by HE-AAC and HE-AACv2 are 48000, 44100, 32000, 24000, and 16000. When HE-AACv2 is used, the output stream can only be dual-channel.
audioEncodedKbps	`Description:` The audio bitrate (Kbps) of the stream to publish. `Value:` Value range: [32,192]. Default: 50.
audioEncodedSampleRate	`Description:` The audio sample rate of the stream to publish. `Value:` Valid values: [48000, 44100, 32000, 24000, 16000, 8000]. Default: 48000 (Hz).
videoEncodedCodecType	`Description:` The video codec of the stream to publish. `Value:` Valid values: 0 (H264); 1 (H265). Default: 0.
videoEncodedFPS	`Description:` The frame rate (fps) of the stream to publish. `Value:` Value range: (0,30]. Default: 20.
videoEncodedGOP	`Description:` The keyframe interval (GOP) of the stream to publish. `Value:` Value range: [1,5]. Default: 3 (seconds).
videoEncodedHeight	`Description:` The resolution (height) of the stream to publish. `Value:` Recommended value: 640. If you mix only audio streams, to avoid displaying a black video in the transcoded stream, set both `width` and `height` to `0` .
videoEncodedKbps	`Description:` The video bitrate (Kbps) of the stream to publish. `Value:` If you set this parameter to `0` , TRTC will work out a bitrate based on `videoWidth` and `videoHeight` . For details, refer to the recommended bitrates for the constants of the resolution enum type (see comment).
videoEncodedWidth	`Description:` The resolution (width) of the stream to publish. `Value:` Recommended value: 368. If you mix only audio streams, to avoid displaying a black video in the transcoded stream, set both `width` and `height` to `0` .
videoSeiParams	`Description:` SEI parameters. Default: null `Note:` the parameter is passed in the form of a JSON string. Here is an example to use it:

	```{ "payLoadContent":"xxx", "payloadType":5, "payloadUuid":"1234567890abcdef1234567890abcdef", "interval":1000, "followIdr":false }``` The currently supported fields and their meanings are as follows: payloadContent: Required. The payload content of the passthrough SEI, which cannot be empty. payloadType: Required. The type of the SEI message, with a value range of 5 or an integer within the range of [100, 254] (excluding 244, which is an internally defined timestamp SEI). payloadUuid: Required when payloadType is 5, and ignored in other cases. The value must be a 32-digit hexadecimal number. interval: Optional, default is 1000. The sending interval of the SEI, in milliseconds. followIdr: Optional, default is false. When this value is true, the SEI will be ensured to be carried when sending a key frame, otherwise it is not guaranteed.

TRTCStreamMixingConfig

TRTCStreamMixingConfig

The transcoding parameters

This enum type is used by the publishing API startPublishMediaStream.

You can use this parameter to specify the video layout and input audio information for On-Cloud MixTranscoding.

EnumType	DESC
audioMixUserList	`Description:` The information of each audio stream to mix. `Value:` This parameter is an array. Each `TRTCUser` element in the array indicates the information of an audio stream. **Note** If you do not specify this array, the TRTC backend will automatically mix all streams of the anchors who are sending audio in the room according to the audio encode param TRTCStreamEncoderParam you specify (currently only supports up to 16 audio and video inputs).
audioMixUserListSize	**Description:** The length of the `audioMixUserList` array.
backgroundColor	`Description:` The background color of the mixed stream. `Value:` The value must be a hex number. For example, "0x61B9F1" represents the RGB color value (97,158,241). Default value: 0x000000 (black).
backgroundImage	`Description:` The URL of the background image of the mixed stream. The image specified by the URL will be mixed during On-Cloud MixTranscoding. `Value:` This parameter is left empty by default, which means no background image will be used. **Note** The URL can be 512 bytes long at most, and the image must not exceed 2 MB. The image can be in PNG, JPG, JPEG, or BMP format. We recommend you use a semitransparent image in PNG format.
videoLayoutList	`Description:` The position, size, layer, and stream type of each video in On-Cloud MixTranscoding. `Value:` This parameter is an array. Each `TRTCVideoLayout` element in the array indicates the information of a video in On-Cloud MixTranscoding.
videoLayoutListSize	**Description:** The length of the `videoLayoutList` array.

watermarkList	`Description:` The position, size, and layer of each watermark image in On-Cloud MixTranscoding. `Value:` This parameter is an array. Each `TRTCWatermark` element in the array indicates the information of a watermark.
watermarkListSize	`Description:` The length of the `watermarkList` array.

TRTCPayloadPrivateEncryptionConfig

TRTCPayloadPrivateEncryptionConfig

Media Stream Private Encryption Configuration

This configuration is used to set the algorithm and key for media stream private encryption.

EnumType	DESC
encryptionAlgorithm	`Description:` Encryption algorithm, the default is TRTCEncryptionAlgorithmAes128Gcm.
encryptionKey	`Description:` encryption key, string type. `Value:` If the encryption algorithm is TRTCEncryptionAlgorithmAes128Gcm, the key length must be 16 bytes; if the encryption algorithm is TRTCEncryptionAlgorithmAes256Gcm, the key length must be 32 bytes.
encryptionSalt[32]	`Description:` Salt, initialization vector for encryption. `Value:` It is necessary to ensure that the array filled in this parameter is not empty, not all 0 and the data length is 32 bytes.

TRTCAudioVolumeEvaluateParams

TRTCAudioVolumeEvaluateParams

Volume evaluation and other related parameter settings.

This setting is used to enable vocal detection and sound spectrum calculation.

EnumType	DESC
enablePitchCalculation	`Description:` Whether to enable local vocal frequency calculation.
enableSpectrumCalculation	`Description:` Whether to enable sound spectrum calculation.

enableVadDetection	`Description:` Whether to enable local voice detection. **Note** Call before startLocalAudio.
interval	`Description:` Set the trigger interval of the onUserVoiceVolume callback, the unit is milliseconds, the minimum interval is 100ms, if it is less than or equal to 0, the callback will be closed. `Value:` Recommended value: 300, in milliseconds. **Note** When the interval is greater than 0, the volume prompt will be enabled by default, no additional setting is required.

Deprecated Interface

Last updated：2024-06-06 15:50:06

Deprecate

IDeprecatedTRTCCloud

FuncList	DESC
enableAudioVolumeEvaluation	Enable volume reminder
enableAudioVolumeEvaluation	Enable volume reminder
startLocalAudio	Set sound quality
startRemoteView	Start displaying remote video image
stopRemoteView	Stop displaying remote video image and pulling the video data stream of remote user
setLocalViewFillMode	Set the rendering mode of local image
setLocalViewRotation	Set the clockwise rotation angle of local image
setLocalViewMirror	Set the mirror mode of local camera's preview image
setRemoteViewFillMode	Set the fill mode of substream image
setRemoteViewRotation	Set the clockwise rotation angle of remote image
startRemoteSubStreamView	Start displaying the substream image of remote user
stopRemoteSubStreamView	Stop displaying the substream image of remote user
setRemoteSubStreamViewFillMode	Set the fill mode of substream image
setRemoteSubStreamViewRotation	Set the clockwise rotation angle of substream image
setAudioQuality	Set sound quality
setPriorRemoteVideoStreamType	Specify whether to view the big or small image
setMicVolumeOnMixing	Set mic volume

playBGM	Start background music
stopBGM	Stop background music
pauseBGM	Stop background music
resumeBGM	Stop background music
getBGMDuration	Get the total length of background music in ms
setBGMPosition	Set background music playback progress
setBGMVolume	Set background music volume
setBGMPlayoutVolume	Set the local playback volume of background music
setBGMPublishVolume	Set the remote playback volume of background music
playAudioEffect	Play sound effect
setAudioEffectVolume	Set sound effect volume
stopAudioEffect	Stop sound effect
stopAllAudioEffects	Stop all sound effects
setAllAudioEffectsVolume	Set the volume of all sound effects
pauseAudioEffect	Pause sound effect
resumeAudioEffect	Pause sound effect
enableCustomVideoCapture	Enable custom video capturing mode
sendCustomVideoData	Deliver captured video data to SDK
muteLocalVideo	Pause/Resume publishing local video stream
muteRemoteVideoStream	Pause/Resume subscribing to remote user's video stream
startSpeedTest	Start network speed test (used before room entry)
startScreenCapture	Start screen sharing
setLocalVideoProcessCallback	Set video data callback for third-party beauty filters
getCameraDevicesList	Get the list of cameras
setCurrentCameraDevice	Set the camera to be used currently

getCurrentCameraDevice	Get the currently used camera
getMicDevicesList	Get the list of mics
getCurrentMicDevice	Get the current mic device
setCurrentMicDevice	Select the currently used mic
getCurrentMicDeviceVolume	Get the current mic volume
setCurrentMicDeviceVolume	Set the current mic volume
setCurrentMicDeviceMute	Set the mute status of the current system mic
getCurrentMicDeviceMute	Get the mute status of the current system mic
getSpeakerDevicesList	Get the list of speakers
getCurrentSpeakerDevice	Get the currently used speaker
setCurrentSpeakerDevice	Set the speaker to use
getCurrentSpeakerVolume	Get the current speaker volume
setCurrentSpeakerVolume	Set the current speaker volume
getCurrentSpeakerDeviceMute	Get the mute status of the current system speaker
setCurrentSpeakerDeviceMute	Set whether to mute the current system speaker
startCameraDeviceTest	Start camera test
startCameraDeviceTest	
stopCameraDeviceTest	Start camera test
startMicDeviceTest	Start mic test
stopMicDeviceTest	Start mic test
startSpeakerDeviceTest	Start speaker test
stopSpeakerDeviceTest	Stop speaker test
selectScreenCaptureTarget	start in-app screen sharing (for iOS 13.0 and above only)
setVideoEncoderRotation	Set the direction of image output by video encoder
setVideoEncoderMirror	Set the mirror mode of image output by encoder

enableAudioVolumeEvaluation

enableAudioVolumeEvaluation

void enableAudioVolumeEvaluation	(uint32_t interval)

Enable volume reminder

@deprecated This API is not recommended after v10.1. Please use enableAudioVolumeEvaluation(enable, params) instead.

enableAudioVolumeEvaluation

enableAudioVolumeEvaluation

void enableAudioVolumeEvaluation	(uint32_t interval
	bool enable_vad)

Enable volume reminder

@deprecated This API is not recommended after v11.2. Please use enableAudioVolumeEvaluation(enable, params) instead.

startLocalAudio

startLocalAudio

Set sound quality

@deprecated This API is not recommended after v8.0. Please use startLocalAudio:quality instead.

startRemoteView

startRemoteView

void startRemoteView	(const char* userId
	TXView rendView)

Start displaying remote video image

@deprecated This API is not recommended after v8.0. Please use startRemoteView:streamType:view: instead.

stopRemoteView

stopRemoteView

void stopRemoteView	(const char* userId)

Stop displaying remote video image and pulling the video data stream of remote user

@deprecated This API is not recommended after v8.0. Please use stopRemoteView:streamType: instead.

setLocalViewFillMode

setLocalViewFillMode

void setLocalViewFillMode	(TRTCVideoFillMode mode)

Set the rendering mode of local image

@deprecated This API is not recommended after v8.0. Please use setLocalRenderParams instead.

setLocalViewRotation

setLocalViewRotation

void setLocalViewRotation	(TRTCVideoRotation rotation)

Set the clockwise rotation angle of local image

@deprecated This API is not recommended after v8.0. Please use setLocalRenderParams instead.

setLocalViewMirror

setLocalViewMirror

void setLocalViewMirror	(bool mirror)

Set the mirror mode of local camera's preview image

@deprecated This API is not recommended after v8.0. Please use setLocalRenderParams instead.

setRemoteViewFillMode

setRemoteViewFillMode

void setRemoteViewFillMode	(const char* userId
	TRTCVideoFillMode mode)

Set the fill mode of substream image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

setRemoteViewRotation

setRemoteViewRotation

void setRemoteViewRotation	(const char* userId
	TRTCVideoRotation rotation)

Set the clockwise rotation angle of remote image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

startRemoteSubStreamView

startRemoteSubStreamView

void startRemoteSubStreamView	(const char* userId
	TXView rendView)

Start displaying the substream image of remote user

@deprecated This API is not recommended after v8.0. Please use startRemoteView:streamType:view: instead.

stopRemoteSubStreamView

stopRemoteSubStreamView

void stopRemoteSubStreamView	(const char* userId)

Stop displaying the substream image of remote user

@deprecated This API is not recommended after v8.0. Please use stopRemoteView:streamType: instead.

setRemoteSubStreamViewFillMode

setRemoteSubStreamViewFillMode

void setRemoteSubStreamViewFillMode	(const char* userId
	TRTCVideoFillMode mode)

Set the fill mode of substream image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

setRemoteSubStreamViewRotation

setRemoteSubStreamViewRotation

void setRemoteSubStreamViewRotation	(const char* userId
	TRTCVideoRotation rotation)

Set the clockwise rotation angle of substream image

@deprecated This API is not recommended after v8.0. Please use setRemoteRenderParams:streamType:params: instead.

setAudioQuality

setAudioQuality

void setAudioQuality	(TRTCAudioQuality quality)

Set sound quality

@deprecated This API is not recommended after v8.0. Please use startLocalAudio:quality instead.

setPriorRemoteVideoStreamType

setPriorRemoteVideoStreamType

void setPriorRemoteVideoStreamType	(TRTCVideoStreamType type)

Specify whether to view the big or small image

@deprecated This API is not recommended after v8.0. Please use startRemoteView:streamType:view: instead.

setMicVolumeOnMixing

setMicVolumeOnMixing

void setMicVolumeOnMixing	(uint32_t volume)

Set mic volume

@deprecated This API is not recommended after v6.9. Please use setAudioCaptureVolume instead.

playBGM

playBGM

void playBGM	(const char* path)

Start background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

stopBGM

stopBGM

Stop background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

pauseBGM

pauseBGM

Stop background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

resumeBGM

resumeBGM

Stop background music

@deprecated This API is not recommended after v7.3. Please use getAudioEffectManager instead.

getBGMDuration

getBGMDuration

uint32_t getBGMDuration	(const char* path)

Get the total length of background music in ms

@deprecated This API is not recommended after v7.3. Please use getMusicDurationInMS API in TXAudioEffectManager instead.

setBGMPosition

setBGMPosition

void setBGMPosition	(uint32_t pos)

Set background music playback progress

@deprecated This API is not recommended after v7.3. Please use seekMusicToPosInMS API in TXAudioEffectManager instead.

setBGMVolume

setBGMVolume

void setBGMVolume	(uint32_t volume)

Set background music volume

@deprecated This API is not recommended after v7.3. Please use setMusicVolume API in TXAudioEffectManager instead.

setBGMPlayoutVolume

setBGMPlayoutVolume

void setBGMPlayoutVolume	(uint32_t volume)

Set the local playback volume of background music

@deprecated This API is not recommended after v7.3. Please use setMusicPlayoutVolume API in TXAudioEffectManager instead.

setBGMPublishVolume

setBGMPublishVolume

void setBGMPublishVolume	(uint32_t volume)

Set the remote playback volume of background music

@deprecated This API is not recommended after v7.3. Please use setBGMPublishVolume API in TXAudioEffectManager instead.

playAudioEffect

playAudioEffect

void playAudioEffect	(TRTCAudioEffectParam* effect)

Play sound effect

@deprecated This API is not recommended after v7.3. Please use startPlayMusic API in TXAudioEffectManager instead.

setAudioEffectVolume

setAudioEffectVolume

void setAudioEffectVolume	(int effectId
	int volume)

Set sound effect volume

@deprecated This API is not recommended after v7.3. Please use setMusicPublishVolume and setMusicPlayoutVolume API in TXAudioEffectManager instead.

stopAudioEffect

stopAudioEffect

void stopAudioEffect	(int effectId)

Stop sound effect

@deprecated This API is not recommended after v7.3. Please use stopPlayMusic API in TXAudioEffectManager instead.

stopAllAudioEffects

stopAllAudioEffects

Stop all sound effects

@deprecated This API is not recommended after v7.3. Please use stopPlayMusic API in TXAudioEffectManager instead.

setAllAudioEffectsVolume

setAllAudioEffectsVolume

void setAllAudioEffectsVolume	(int volume)

Set the volume of all sound effects

@deprecated This API is not recommended after v7.3. Please use setMusicPublishVolume and setMusicPlayoutVolume API in TXAudioEffectManager instead.

pauseAudioEffect

pauseAudioEffect

void pauseAudioEffect	(int effectId)

Pause sound effect

@deprecated This API is not recommended after v7.3. Please use pauseAudioEffect API in TXAudioEffectManager instead.

resumeAudioEffect

resumeAudioEffect

void resumeAudioEffect	(int effectId)

Pause sound effect

@deprecated This API is not recommended after v7.3. Please use resumePlayMusic API in TXAudioEffectManager instead.

enableCustomVideoCapture

enableCustomVideoCapture

void enableCustomVideoCapture	(bool enable)

Enable custom video capturing mode

@deprecated This API is not recommended after v8.5. Please use enableCustomVideoCapture instead.

sendCustomVideoData

sendCustomVideoData

void sendCustomVideoData	(TRTCVideoFrame* frame)

Deliver captured video data to SDK

@deprecated This API is not recommended after v8.5. Please use sendCustomVideoData instead.

muteLocalVideo

muteLocalVideo

void muteLocalVideo	(bool mute)

Pause/Resume publishing local video stream

@deprecated This API is not recommended after v8.9. Please use muteLocalVideo (streamType, mute) instead.

muteRemoteVideoStream

muteRemoteVideoStream

void muteRemoteVideoStream	(const char* userId
	bool mute)

Pause/Resume subscribing to remote user's video stream

@deprecated This API is not recommended after v8.9. Please use muteRemoteVideoStream (userId, streamType, mute) instead.

startSpeedTest

startSpeedTest

void startSpeedTest	(uint32_t sdkAppId
	const char* userId
	const char* userSig)

Start network speed test (used before room entry)

@deprecated This API is not recommended after v9.2. Please use startSpeedTest (params) instead.

startScreenCapture

startScreenCapture

void startScreenCapture	(TXView rendView)

Start screen sharing

@deprecated This API is not recommended after v7.2. Please use `startScreenCapture:streamType:encParam:` instead.

setLocalVideoProcessCallback

setLocalVideoProcessCallback

int setLocalVideoProcessCallback	(TRTCVideoPixelFormat pixelFormat
	TRTCVideoBufferType bufferType
	ITRTCVideoFrameCallback* callback)

Set video data callback for third-party beauty filters

@deprecated This API is not recommended after v11.4. Please use the enableLocalVideoCustomProcess and setLocalVideoCustomProcessCallback instead.

getCameraDevicesList

getCameraDevicesList

Get the list of cameras

@deprecated This API is not recommended after v8.0. Please use the getDevicesList API in TXDeviceManager instead.

setCurrentCameraDevice

setCurrentCameraDevice

void setCurrentCameraDevice	(const char* deviceId)

Set the camera to be used currently

@deprecated This API is not recommended after v8.0. Please use the setCurrentDevice API in TXDeviceManager instead.

getCurrentCameraDevice

getCurrentCameraDevice

Get the currently used camera

@deprecated This API is not recommended after v8.0. Please use the getCurrentDevice API in TXDeviceManager instead.

getMicDevicesList

getMicDevicesList

Get the list of mics

@deprecated This API is not recommended after v8.0. Please use the getDevicesList API in TXDeviceManager instead.

getCurrentMicDevice

getCurrentMicDevice

Get the current mic device

@deprecated This API is not recommended after v8.0. Please use the getCurrentDevice API in TXDeviceManager instead.

setCurrentMicDevice

setCurrentMicDevice

void setCurrentMicDevice	(const char* micId)

Select the currently used mic

@deprecated This API is not recommended after v8.0. Please use the setCurrentDevice API in TXDeviceManager instead.

getCurrentMicDeviceVolume

getCurrentMicDeviceVolume

Get the current mic volume

@deprecated This API is not recommended after v8.0. Please use the getCurrentDeviceVolume API in TXDeviceManager instead.

setCurrentMicDeviceVolume

setCurrentMicDeviceVolume

void setCurrentMicDeviceVolume	(uint32_t volume)

Set the current mic volume

@deprecated This API is not recommended after v8.0. Please use the setCurrentDeviceVolume API in TXDeviceManager instead.

setCurrentMicDeviceMute

setCurrentMicDeviceMute

void setCurrentMicDeviceMute	(bool mute)

Set the mute status of the current system mic

@deprecated This API is not recommended after v8.0. Please use the setCurrentDeviceMute API in TXDeviceManager instead.

getCurrentMicDeviceMute

getCurrentMicDeviceMute

Get the mute status of the current system mic

@deprecated This API is not recommended after v8.0. Please use the getCurrentDeviceMute API in TXDeviceManager instead.

getSpeakerDevicesList

getSpeakerDevicesList

Get the list of speakers

@deprecated This API is not recommended after v8.0. Please use the getDevicesList API in TXDeviceManager instead.

getCurrentSpeakerDevice

getCurrentSpeakerDevice

Get the currently used speaker

@deprecated This API is not recommended after v8.0. Please use the getCurrentDevice API in TXDeviceManager instead.

setCurrentSpeakerDevice

setCurrentSpeakerDevice

void setCurrentSpeakerDevice	(const char* speakerId)

Set the speaker to use

@deprecated This API is not recommended after v8.0. Please use the setCurrentDevice API in TXDeviceManager instead.

getCurrentSpeakerVolume

getCurrentSpeakerVolume

Get the current speaker volume

@deprecated This API is not recommended after v8.0. Please use the getCurrentDeviceVolume API in TXDeviceManager instead.

setCurrentSpeakerVolume

setCurrentSpeakerVolume

void setCurrentSpeakerVolume	(uint32_t volume)

Set the current speaker volume

@deprecated This API is not recommended after v8.0. Please use the setCurrentDeviceVolume API in TXDeviceManager instead.

getCurrentSpeakerDeviceMute

getCurrentSpeakerDeviceMute

Get the mute status of the current system speaker

@deprecated This API is not recommended after v8.0. Please use the getCurrentDeviceMute API in TXDeviceManager instead.

setCurrentSpeakerDeviceMute

setCurrentSpeakerDeviceMute

void setCurrentSpeakerDeviceMute	(bool mute)

Set whether to mute the current system speaker

@deprecated This API is not recommended after v8.0. Please use the setCurrentDeviceMute API in TXDeviceManager instead.

startCameraDeviceTest

startCameraDeviceTest

void startCameraDeviceTest	(TXView renderView)

Start camera test

@deprecated This API is not recommended after v8.0. Please use the startCameraDeviceTest API in TXDeviceManager instead.

stopCameraDeviceTest

stopCameraDeviceTest

Start camera test

@deprecated This API is not recommended after v8.0. Please use the stopCameraDeviceTest API in TXDeviceManager instead.

startMicDeviceTest

startMicDeviceTest

void startMicDeviceTest	(uint32_t interval)

Start mic test

@deprecated This API is not recommended after v8.0. Please use the startMicDeviceTest API in TXDeviceManager instead.

stopMicDeviceTest

stopMicDeviceTest

Start mic test

@deprecated This API is not recommended after v8.0. Please use the stopMicDeviceTest API in TXDeviceManager instead.

startSpeakerDeviceTest

startSpeakerDeviceTest

void startSpeakerDeviceTest	(const char* testAudioFilePath)

Start speaker test

@deprecated This API is not recommended after v8.0. Please use the startSpeakerDeviceTest API in TXDeviceManager instead.

stopSpeakerDeviceTest

stopSpeakerDeviceTest

Stop speaker test

@deprecated This API is not recommended after v8.0. Please use the stopSpeakerDeviceTest API in TXDeviceManager instead.

selectScreenCaptureTarget

selectScreenCaptureTarget

void selectScreenCaptureTarget	(const TRTCScreenCaptureSourceInfo& source
	const RECT& captureRect
	bool captureMouse = true
	bool highlightWindow = true)

start in-app screen sharing (for iOS 13.0 and above only)

@deprecated This API is not recommended after v8.6. Please use startScreenCaptureInApp instead.

setVideoEncoderRotation

setVideoEncoderRotation

void setVideoEncoderRotation	(TRTCVideoRotation rotation)

Set the direction of image output by video encoder

@deprecated It is deprecated starting from v11.7.

setVideoEncoderMirror

setVideoEncoderMirror

void setVideoEncoderMirror	(bool mirror)

Set the mirror mode of image output by encoder

@deprecated It is deprecated starting from v11.7.

Error Codes

Last updated：2024-06-06 15:50:05

Module: TRTC ErrorCode

Function: Used to notify customers of warnings and errors that occur during the use of TRTC

ErrorCode

EnumType

EnumType	DESC
TXLiteAVError	Error Codes
TXLiteAVWarning	Warning codes

TXLiteAVError

TXLiteAVError

Error Codes

Enum	Value	DESC
ERR_NULL	0	No error.
ERR_FAILED	-1	Unclassified error.
ERR_INVALID_PARAMETER	-2	An invalid parameter was pas in when the API was called.
ERR_REFUSED	-3	The API call was rejected.
ERR_NOT_SUPPORTED	-4	The current API cannot be called.
ERR_INVALID_LICENSE	-5	Failed to call the API because

		the license is invalid.
ERR_REQUEST_SERVER_TIMEOUT	-6	The request timed out.
ERR_SERVER_PROCESS_FAILED	-7	The server cannot process yo request.
ERR_DISCONNECTED	-8	Disconnected from the server
ERR_CAMERA_START_FAIL	-1301	Failed to turn the camera on. This may occur when there is problem with the camera configuration program (driver) Windows or macOS. Disable reenable the camera, restart t camera, or update the configuration program.
ERR_CAMERA_NOT_AUTHORIZED	-1314	No permission to access to th camera. This usually occurs c mobile devices and may be because the user denied acce
ERR_CAMERA_SET_PARAM_FAIL	-1315	Incorrect camera parameter settings (unsupported values others).
ERR_CAMERA_OCCUPY	-1316	The camera is being used. Tr another camera.
ERR_SCREEN_CAPTURE_START_FAIL	-1308	Failed to start screen recordir If this occurs on a mobile devi it may be because the user denied screen sharing permission; if it occurs on Windows or macOS, check whether the parameters of the screen recording API are set a required.
ERR_SCREEN_CAPTURE_UNSURPORT	-1309	Screen recording failed. Scree recording is only supported or Android versions later than 5. and iOS versions later than 1
ERR_SCREEN_CAPTURE_STOPPED	-7001	Screen recording was stoppe by the system.

ERR_SCREEN_SHARE_NOT_AUTHORIZED	-102015	No permission to publish the substream.
ERR_SCREEN_SHRAE_OCCUPIED_BY_OTHER	-102016	Another user is publishing the substream.
ERR_VIDEO_ENCODE_FAIL	-1303	Failed to encode video frames This may occur when a user c iOS switches to another app, which may cause the system release the hardware encode When the user switches back this error may be thrown befo the hardware encoder is restarted.
ERR_UNSUPPORTED_RESOLUTION	-1305	Unsupported video resolution
ERR_PIXEL_FORMAT_UNSUPPORTED	-1327	Custom video capturing: Unsupported pixel format.
ERR_BUFFER_TYPE_UNSUPPORTED	-1328	Custom video capturing: Unsupported buffer type.
ERR_NO_AVAILABLE_HEVC_DECODERS	-2304	No available HEVC decoder found.
ERR_MIC_START_FAIL	-1302	Failed to turn the mic on. This may occur when there is a problem with the mic configuration program (driver) Windows or macOS. Disable reenable the mic, restart the n or update the configuration program.
ERR_MIC_NOT_AUTHORIZED	-1317	No permission to access to th mic. This usually occurs on mobile devices and may be because the user denied acce
ERR_MIC_SET_PARAM_FAIL	-1318	Failed to set mic parameters.
ERR_MIC_OCCUPY	-1319	The mic is being used. The m cannot be turned on when, fo example, the user is having a on the mobile device.

ERR_MIC_STOP_FAIL	-1320	Failed to turn the mic off.
ERR_SPEAKER_START_FAIL	-1321	Failed to turn the speaker on. This may occur when there is problem with the speaker configuration program (driver) Windows or macOS. Disable reenable the speaker, restart speaker, or update the configuration program.
ERR_SPEAKER_SET_PARAM_FAIL	-1322	Failed to set speaker parameters.
ERR_SPEAKER_STOP_FAIL	-1323	Failed to turn the speaker off.
ERR_AUDIO_PLUGIN_START_FAIL	-1330	Failed to record computer auc which may be because the au driver is unavailable.
ERR_AUDIO_PLUGIN_INSTALL_NOT_AUTHORIZED	-1331	No permission to install the au driver.
ERR_AUDIO_PLUGIN_INSTALL_FAILED	-1332	Failed to install the audio drive
ERR_AUDIO_PLUGIN_INSTALLED_BUT_NEED_RESTART	-1333	The virtual sound card is installed successfully, but due the restrictions of macOS, you cannot use it right after installation. Ask users to resta the app upon receiving this er code.
ERR_AUDIO_ENCODE_FAIL	-1304	Failed to encode audio frames This may occur if the SDK cou not process the custom audio data passed in.
ERR_UNSUPPORTED_SAMPLERATE	-1306	Unsupported audio sample ra
ERR_TRTC_ENTER_ROOM_FAILED	-3301	Failed to enter the room. For t reason, refer to the error message for -3301 in `onError` .
ERR_TRTC_REQUEST_IP_TIMEOUT	-3307	IP and signature request time out. Check your network

		connection and whether your firewall allows UDP. Try visiting the IP address 162.14.22.165:8000 or 162.14.6.105:8000 and the domain default-query.trtc.tencent-cloud.com:8000.
ERR_TRTC_CONNECT_SERVER_TIMEOUT	-3308	Room entry request timed out Check your network connecti and whether VPN is used. Yo can also switch to 4G to run a test.
ERR_TRTC_ROOM_PARAM_NULL	-3316	Empty room entry parameters Please check whether valid parameters were passed in to the `enterRoom:appScer` API.
ERR_TRTC_INVALID_SDK_APPID	-3317	Incorrect room entry paramete Check whether `TRTCParams.sdkAppId` empty.
ERR_TRTC_INVALID_ROOM_ID	-3318	Incorrect room entry paramete Check whether `TRTCParams.roomId` or `TRTCParams.strRoomId` empty. Note that you cannot s both parameters.
ERR_TRTC_INVALID_USER_ID	-3319	Incorrect room entry paramete Check whether `TRTCParams.userId` is empty.
ERR_TRTC_INVALID_USER_SIG	-3320	Incorrect room entry paramete Check whether `TRTCParams.userSig` is empty.
ERR_TRTC_ENTER_ROOM_REFUSED	-3340	Request to enter room denied Check whether you called

		`enterRoom` twice to enter same room.
ERR_TRTC_INVALID_PRIVATE_MAPKEY	-100006	Advanced permission control enabled but failed to verify `TRTCParams.privateMapI` . For details, see Enabling Advanced Permission Contro
ERR_TRTC_SERVICE_SUSPENDED	-100013	The service is unavailable. Check if you have used up yo package or whether your Tencent Cloud account has overdue payments.
ERR_TRTC_USER_SIG_CHECK_FAILED	-100018	Failed to verify `UserSig` Check whether `TRTCParams.userSig` is correct or valid. For details, see UserSig Generation and Verification.
ERR_TRTC_PUSH_THIRD_PARTY_CLOUD_TIMEOUT	-3321	The relay to CDN request tim out
ERR_TRTC_MIX_TRANSCODING_TIMEOUT	-3322	The On-Cloud MixTranscodin request timed out.
ERR_TRTC_PUSH_THIRD_PARTY_CLOUD_FAILED	-3323	Abnormal response packets f relay.
ERR_TRTC_MIX_TRANSCODING_FAILED	-3324	Abnormal response packet fo On-Cloud MixTranscoding.
ERR_TRTC_START_PUBLISHING_TIMEOUT	-3333	Signaling for publishing to the Tencent Cloud CDN timed ou
ERR_TRTC_START_PUBLISHING_FAILED	-3334	Signaling for publishing to the Tencent Cloud CDN was abnormal.
ERR_TRTC_STOP_PUBLISHING_TIMEOUT	-3335	Signaling for stopping publish to the Tencent Cloud CDN tim out.
ERR_TRTC_STOP_PUBLISHING_FAILED	-3336	Signaling for stopping publish

		to the Tencent Cloud CDN wa abnormal.
ERR_TRTC_CONNECT_OTHER_ROOM_TIMEOUT	-3326	The co-anchoring request tim out.
ERR_TRTC_DISCONNECT_OTHER_ROOM_TIMEOUT	-3327	The request to stop co-ancho timed out.
ERR_TRTC_CONNECT_OTHER_ROOM_INVALID_PARAMETER	-3328	Invalid parameter.
ERR_TRTC_CONNECT_OTHER_ROOM_AS_AUDIENCE	-3330	The current user is an audien member and cannot request c stop cross-room communicati Please call `switchRole` to switch to an anchor first.
ERR_BGM_OPEN_FAILED	-4001	Failed to open the file, such a invalid data found when processing input, ffmpeg prot not found, etc.
ERR_BGM_DECODE_FAILED	-4002	Audio file decoding failed.
ERR_BGM_OVER_LIMIT	-4003	The number exceeds the limit such as preloading two background music at the sam time.
ERR_BGM_INVALID_OPERATION	-4004	Invalid operation, such as call a preload function after startin playback.
ERR_BGM_INVALID_PATH	-4005	Invalid path, Please check whether the path you passed points to a legal music file.
ERR_BGM_INVALID_URL	-4006	Invalid URL, Please use a browser to check whether the URL address you passed in c download the desired music fi
ERR_BGM_NO_AUDIO_STREAM	-4007	No audio stream, Please conf whether the file you passed is legal audio file and whether th file is damaged.
ERR_BGM_FORMAT_NOT_SUPPORTED	-4008	Unsupported format, Please

		confirm whether the file forma you passed is a supported file format. The mobile version supports [mp3, aac, m4a, wav ogg, mp4, mkv], and the desk version supports [mp3, aac, m4a, wav, mp4, mkv].

TXLiteAVWarning

TXLiteAVWarning

Warning codes

Enum	Value	DESC
WARNING_HW_ENCODER_START_FAIL	1103	Failed to start the hardware encoder. Switched to software encoding.
WARNING_CURRENT_ENCODE_TYPE_CHANGED	1104	The codec changed. The additional field `type` in `onWarning` indicates the codec currently in use. `0` indicates H.264, and `1` indicates H.265. The additional field `hardware` in `onWarning` indicates the encoder type currently in use. `0` indicates software encoder, and `1` indicates hardware encoder. The additional field `stream` in `onWarning` indicates the stream

		type currently in use. `0` indicates big stream, and `1` indicates small stream, and `2` indicates sub stream.
WARNING_VIDEO_ENCODER_SW_TO_HW	1107	Insufficient CPU for software encoding. Switched to hardware encoding.
WARNING_INSUFFICIENT_CAPTURE_FPS	1108	The capturing frame rate of the camera is insufficient. This error occurs on some Android phones with built-in beauty filters.
WARNING_SW_ENCODER_START_FAIL	1109	Failed to start the software encoder.
WARNING_REDUCE_CAPTURE_RESOLUTION	1110	The capturing frame rate of the camera was reduced for balance between frame rate and performance.
WARNING_CAMERA_DEVICE_EMPTY	1111	No available camera found.
WARNING_CAMERA_NOT_AUTHORIZED	1112	The user didn’t grant the application camera permission.
WARNING_OUT_OF_MEMORY	1113	Some functions may not work properly due to out of memory.
WARNING_CAMERA_IS_OCCUPIED	1114	The camera is occupied.
WARNING_CAMERA_DEVICE_ERROR	1115	The camera device is error.

WARNING_CAMERA_DISCONNECTED	1116	The camera is disconnected.
WARNING_CAMERA_START_FAILED	1117	The camera is started failed.
WARNING_CAMERA_SERVER_DIED	1118	The camera sever is died.
WARNING_SCREEN_CAPTURE_NOT_AUTHORIZED	1206	The user didn't grant the application screen recording permission.
WARNING_CURRENT_DECODE_TYPE_CHANGED	2008	The codec changed. The additional field `type` in `onWarning` indicates the codec currently in use. `1` indicates H.265, and `0` indicates H.264. This field is not supported on Windows.
WARNING_VIDEO_FRAME_DECODE_FAIL	2101	Failed to decode the current video frame.
WARNING_HW_DECODER_START_FAIL	2106	Failed to start the hardware decoder. The software decoder is used instead.
WARNING_VIDEO_DECODER_HW_TO_SW	2108	The hardware decoder failed to decode the first I-frame of the current stream. The SDK automatically switched to the software decoder.
WARNING_SW_DECODER_START_FAIL	2109	Failed to start the software decoder.

WARNING_VIDEO_RENDER_FAIL	2110	Failed to render the video.
WARNING_VIRTUAL_BACKGROUND_DEVICE_UNSURPORTED	8001	The device does not support virtual background
WARNING_VIRTUAL_BACKGROUND_NOT_AUTHORIZED	8002	Virtual background not authorized
WARNING_VIRTUAL_BACKGROUND_INVALID_PARAMETER	8003	Enable virtual background with invalid parameter
WARNING_VIRTUAL_BACKGROUND_PERFORMANCE_INSUFFICIENT	8004	Virtual background performance insufficient
WARNING_MICROPHONE_DEVICE_EMPTY	1201	No available mic found.
WARNING_SPEAKER_DEVICE_EMPTY	1202	No available speaker found.
WARNING_MICROPHONE_NOT_AUTHORIZED	1203	The user didn’t grant the application mic permission.
WARNING_MICROPHONE_DEVICE_ABNORMAL	1204	The audio capturing device is unavailable (which may be because the device is used by another application or is considered invalid by the system).
WARNING_SPEAKER_DEVICE_ABNORMAL	1205	The audio playback device is unavailable (which may be because the device is used by another application or is considered invalid by the system).
WARNING_BLUETOOTH_DEVICE_CONNECT_FAIL	1207	The bluetooth device

		failed to connect (which may be because another app is occupying the audio channel by setting communication mode).
WARNING_MICROPHONE_IS_OCCUPIED	1208	The audio capturing device is occupied.
WARNING_AUDIO_FRAME_DECODE_FAIL	2102	Failed to decode the current audio frame.
WARNING_AUDIO_RECORDING_WRITE_FAIL	7001	Failed to write recorded audio into the file.
WARNING_MICROPHONE_HOWLING_DETECTED	7002	Detect capture audio howling
WARNING_IGNORE_UPSTREAM_FOR_AUDIENCE	6001	The current user is an audience member and cannot publish audio or video. Please switch to an anchor first.
WARNING_UPSTREAM_AUDIO_AND_VIDEO_OUT_OF_SYNC	6006	The audio or video sending timestamps are abnormal, which may cause audio and video synchronization issues.

Web
Overview

Last updated：2024-05-29 15:21:54

API Details

TRTC

1. TRTC is the main entry for TRTC SDK, providing APIs such as create trtc instance(TRTC.create), TRTC.getCameraList, TRTC.getMicrophoneList, TRTC.isSupported.
2. trtc instance, provides the core capability for real-time audio and video calls.

Enter room trtc.enterRoom

Exit room trtc.exitRoom

Turn on camera trtc.startLocalVideo

Turn on microphone trtc.startLocalAudio

Turn off camera trtc.stopLocalVideo

Turn off microphone trtc.stopLocalAudio

Play remote video trtc.startRemoteVideo

Stop playing remote video trtc.stopRemoteVideo

Mute/unmute remote audio trtc.muteRemoteAudio

TRTC Static Methods

Name	Description
create	Create a TRTC object for implementing functions such as entering a room, previewing, pushing, and pulling streams.
setLogLevel	Set the log output level It is recommended to set the DEBUG level during development and testing, which includes detailed prompt information. The default output level is INFO, which includes the log information of the main functions of the SDK.
isSupported	Check if the TRTC Web SDK is supported by the current browser
getCameraList	Returns the list of camera devices Note
getMicrophoneList	Returns the list of microphone devices Note
getSpeakerList	Returns the list of speaker devices For security reasons, the label and deviceId fields may be empty before the user authorizes access to the camera or microphone.

	Therefore, it is recommended to call this interface to obtain device details after the user authorizes access.
setCurrentSpeaker	Set the current speaker for audio playback

TRTC Methods

Name	Description
enterRoom	Enter a video call room.
exitRoom	Exit the current audio and video call room.
switchRole	Switches the user role, only effective in TRTC.TYPE.SCENE_LIVE interactive live streaming mode.
destroy	Destroy the TRTC instance
startLocalAudio	Start collecting audio from the local microphone and publish it to the current room.
updateLocalAudio	Update the configuration of the local microphone.
stopLocalAudio	Stop collecting and publishing the local microphone.
startLocalVideo	Start collecting video from the local camera, play the camera's video on the specified HTMLElement tag, and publish the camera's video to the current room.
updateLocalVideo	Update the local camera configuration.
stopLocalVideo	Stop capturing, previewing, and publishing the local camera.
startScreenShare	Start screen sharing.
updateScreenShare	Update screen sharing configuration
stopScreenShare	Stop screen sharing.
startRemoteVideo	Play remote video
updateRemoteVideo	Update remote video playback configuration
stopRemoteVideo	Used to stop remote video playback.
muteRemoteAudio	Mute a remote user and stop pulling audio data from that user. Only effective for the current user, other users in the room can still hear the muted user's voice.

setRemoteAudioVolume	Used to control the playback volume of remote audio.
enableAudioVolumeEvaluation	Enables or disables the volume callback.
on	Listen to the TRTC events
off	Remove event listener
getVideoSnapshot	Get video snapshot
getVideoTrack	Get video track
getAudioTrack	Get audio track
sendSEIMessage	Send SEI message
sendCustomMessage	Send custom message
startPlugin	Start plugin
updatePlugin	Update plugin
stopPlugin	Stop plugin

Note
For FAQs, see Web.

Error Code

TRTC SDK defines 8 types of error codes. TRTC will throws error in the APIs and TRTC.EVENT.ERROR event and you can get the RtcError object for handling error.

Key	Code	Description
INVALID_PARAMETER	5000	The parameters passed in when calling the interface do not meet the API requirements. **Handling suggestion**: Please check whether the passed-in parameters comply with the API specifications, such as whether the parameter type is correct.
INVALID_OPERATION	5100	The prerequisite requirements of the API are not met when calling the interface.

		Handling suggestion: Please check whether the calling logic complies with the API prerequisite requirements according to the corresponding API document. For example: 1. Switching roles before entering the room successfully. 2. The remote user and stream being played do not exist.
ENV_NOT_SUPPORTED	5200	The current environment does not support this function, indicating that the current browser does not support calling the corresponding API. **Handling suggestion**: Usually, TRTC.isSupported can be used to perceive which capabilities the current browser supports. If the browser does not support it, you need to guide the user to use a browser that supports this capability. Reference: Detect Capabilities
DEVICE_ERROR	5300	Capturing media devices failed. The following interfaces will throw this error code when an exception occurs: startLocalVideo, updateLocalVideo, startLocalAudio, updateLocalAudio, startScreenShare, updateScreenShare **Handling suggestion**: Guide the user to check whether the device has a camera and microphone, whether the system has authorized the browser, and whether the browser has authorized the page. It is recommended to increase the device detection process before entering the room to confirm whether the microphone and camera exist and can be captured normally before proceeding to the next call operation. Usually, this exception can be avoided after the device check. Implementation reference: Detect Capabilities If you need to distinguish more detailed exception categories, you can process according to the extraCode
SERVER_ERROR	5400	Got server error. Reasons: expired userSig, Tencent Cloud account arrears, TRTC service not enabled, etc. **Handling suggestion**: Refer to the extraCode.
OPERATION_FAILED	5500	The exception that the SDK cannot solve after multiple retries under the condition of meeting the API call requirements, usually caused by browser or network problems.

		The following interfaces will throw this error code when an exception occurs: enterRoom, startLocalVideo, startLocalAudio, startScreenShare, startRemoteVideo, switchRole **Handling suggestions**: Confirm whether the domain name and port required for communication meet your network environment requirements, refer to Handle Firewall Restriction. Other issues need to be handled by engineers. Submit an issue in github.
OPERATION_ABORT	5998	The error code thrown when the API execution is aborted. When the API is called or repeatedly called without meeting the API lifecycle, the API will abort execution to avoid meaningless operations. For example: Call enterRoom, startLocalVideo continuously, and call exitRoom without entering the room. The following interfaces will throw this error code when an exception occurs: enterRoom, startLocalVideo, startLocalAudio, startScreenShare, startRemoteVideo, switchRole **Handling suggestions**: Capture and identify this error code, then avoid unnecessary calls in business logic, or you can do nothing, because the SDK has done side-effect-free processing, you only need to identify and ignore this error code when catching it.
UNKNOWN_ERROR	5999	Unknown error. Handling suggestions: Submit an issue in github.

Contact Us

Submit an issue in github.

Contact us on telegram.

Error Codes

Last updated：2024-03-26 10:55:04

This document applies to 5.x.x versions of the TRTC Web SDK.
TRTC SDK v5.0 defines 8 types of error codes, which can be obtained through the RtcError object to perform corresponding handling.

Error Code Definitions

Key	Code	Description
INVALID_PARAMETER	5000	The parameters passed in when calling the interface do not meet the API requirements. Handling suggestion: Please check whether the passed-in parameters comply with the API specifications, such as whether the parameter type is correct.
INVALID_OPERATION	5100	The prerequisite requirements of the API are not met when calling the interface. Handling suggestion: Please check whether the calling logic complies with the API prerequisite requirements according to the corresponding API document. For example: 1. Switching roles before entering the room successfully. 2. The remote user and stream being played do not exist.
ENV_NOT_SUPPORTED	5200	The current environment does not support this function, indicating that the current browser does not support calling the corresponding API. Handling suggestion: Usually, TRTC.isSupported can be used to perceive which capabilities the current browser supports. If the browser does not support it, you need to guide the user to use a browser that supports this capability. Reference: Detect Capabilities
DEVICE_ERROR	5300	Description: Exception occurred when obtaining device or collecting audio and video The following interfaces will throw this error code when an exception

		occurs: startLocalVideo, updateLocalVideo, startLocalAudio, updateLocalAudio, startScreenShare, updateScreenShare Suggestion: Guide the user to check whether the device has a camera and microphone, whether the system has authorized the browser, and whether the browser has authorized the page. It is recommended to increase the device detection process before entering the room to confirm whether the microphone and camera exist and can be captured normally before proceeding to the next call operation. Usually, this exception can be avoided after the device check. Implementation reference: Detect Capabilities If you need to distinguish more detailed exception categories, you can process according to the extraCode
SERVER_ERROR	5400	This error code is thrown when abnormal data is returned from the server. The following interfaces will throw this error code when an exception occurs: enterRoom, startLocalVideo, startLocalAudio, startScreenShare, startRemoteVideo, switchRole Handling suggestion: Server exceptions are usually handled during development. Common exceptions include: expired userSig, Tencent Cloud account arrears, TRTC service not enabled, etc. The server returns abnormal data for the following reasons.
OPERATION_FAILED	5500	The exception that the SDK cannot solve after multiple retries under the condition of meeting the API call requirements, usually caused by browser or network problems. The following interfaces will throw this error code when an exception occurs: enterRoom, startLocalVideo, startLocalAudio, startScreenShare, startRemoteVideo, switchRole Handling suggestions: Confirm whether the domain name and port required for communication meet your network environment requirements, refer to the document Dealing with Firewall Restrictions and Setting Proxies Other issues need to be handled by engineers. Contact us on telegram
OPERATION_ABORT	5998	The error code thrown when the API execution is aborted.

		When the API is called or repeatedly called without meeting the API lifecycle, the API will abort execution to avoid meaningless operations. For example: Call enterRoom, startLocalVideo continuously, and call exitRoom without entering the room. The following interfaces will throw this error code when an exception occurs: enterRoom, startLocalVideo, startLocalAudio, startScreenShare, startRemoteVideo, switchRole Handling suggestions: Capture and identify this error code, then avoid unnecessary calls in business logic, or you can do nothing, because the SDK has done side-effect-free processing, you only need to identify and ignore this error code when catching it.
UNKNOWN_ERROR	5999	Description: Unknown error or undefined error Handling suggestions: Contact us on telegram

Electron

Overview

Last updated：2023-10-09 11:53:16

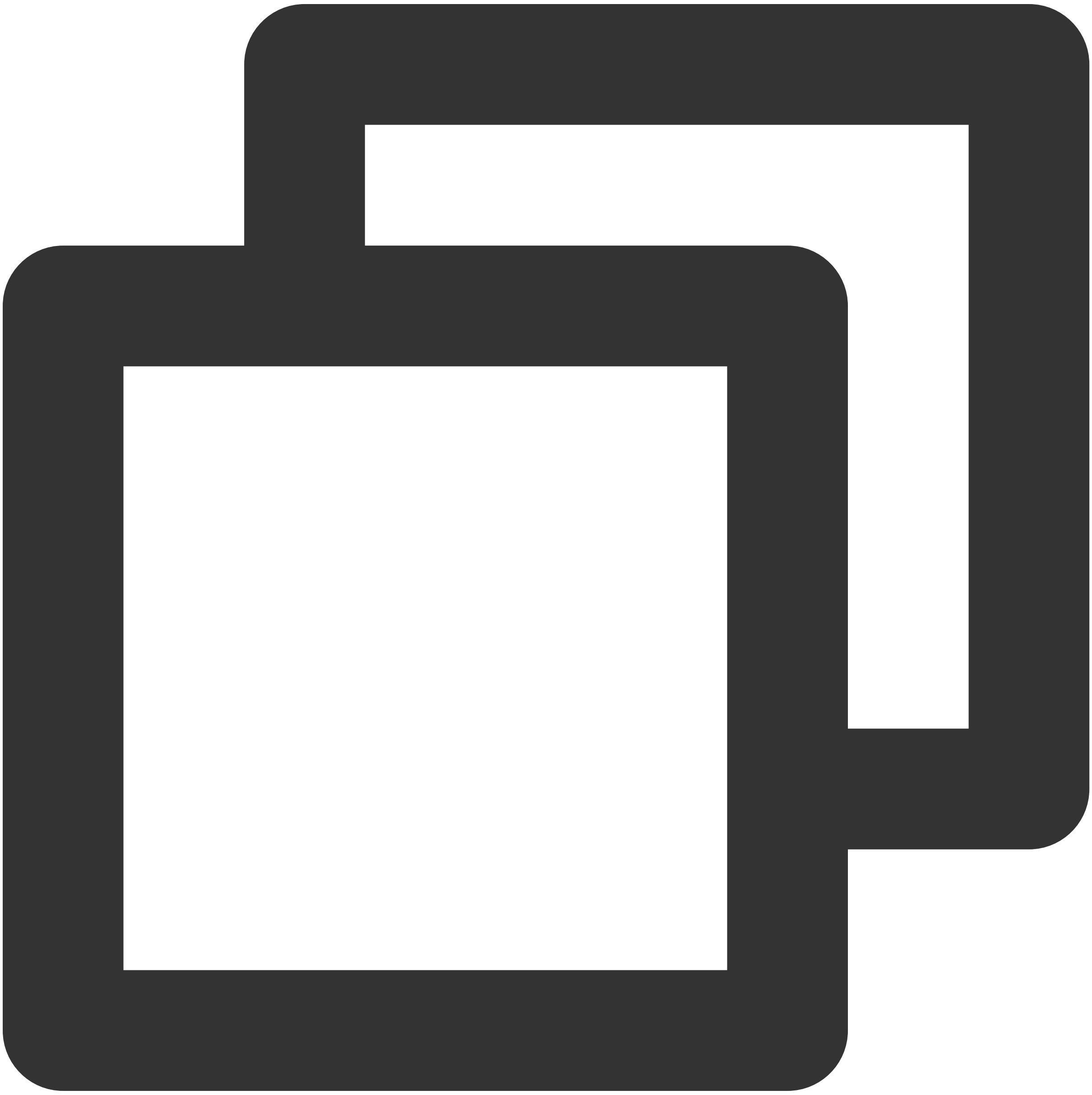
TRTCCloud @ TXLiteAVSDK

TRTC main API classes

Documentation:

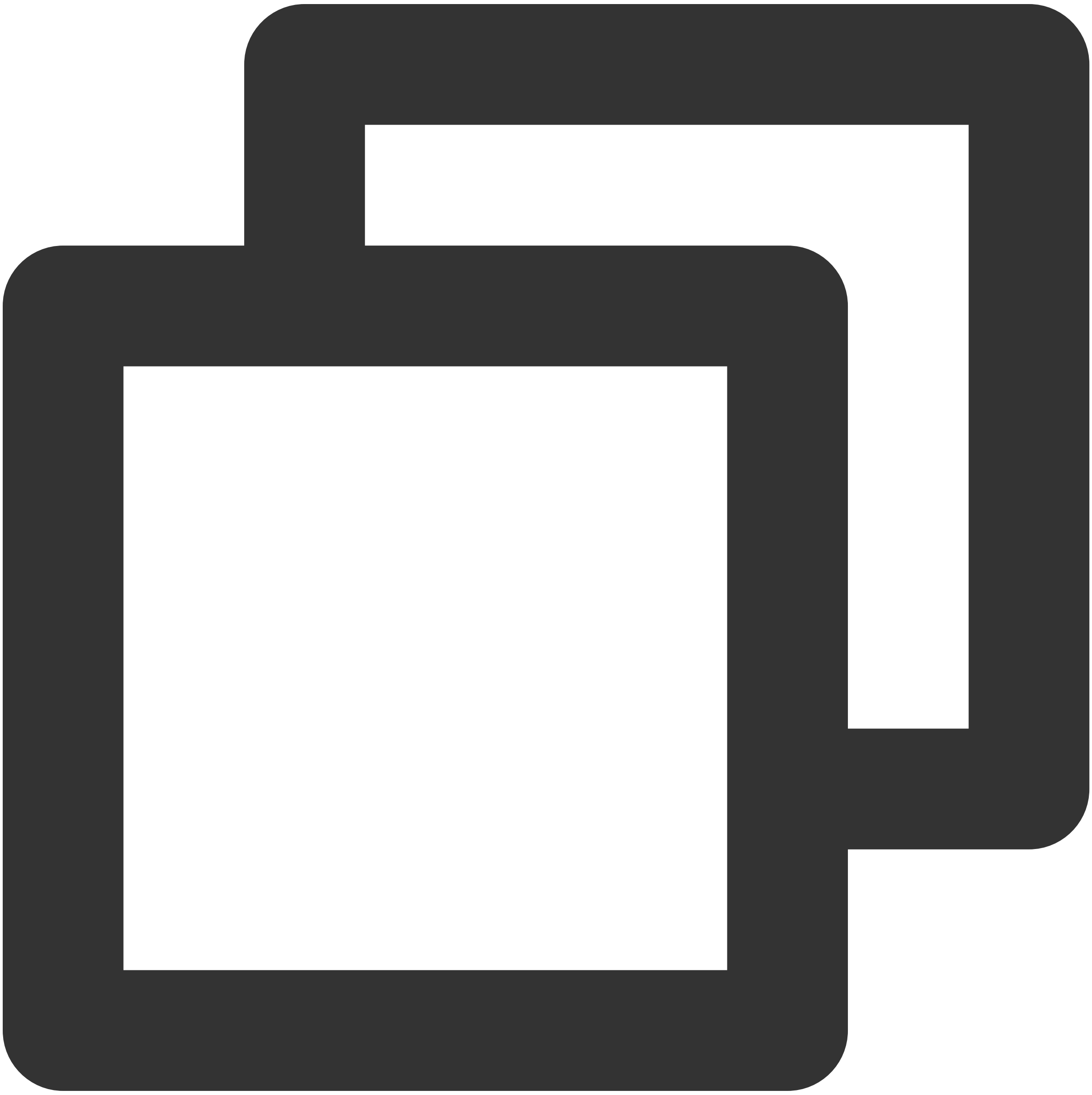
Sample code: TRTC Electron Demo

Creating A TRTC object



```
const TRTCCloud = require('trtc-electron-sdk').default;
// import TRTCCloud from 'trtc-electron-sdk';
this.rtcCloud = new TRTCCloud();
```

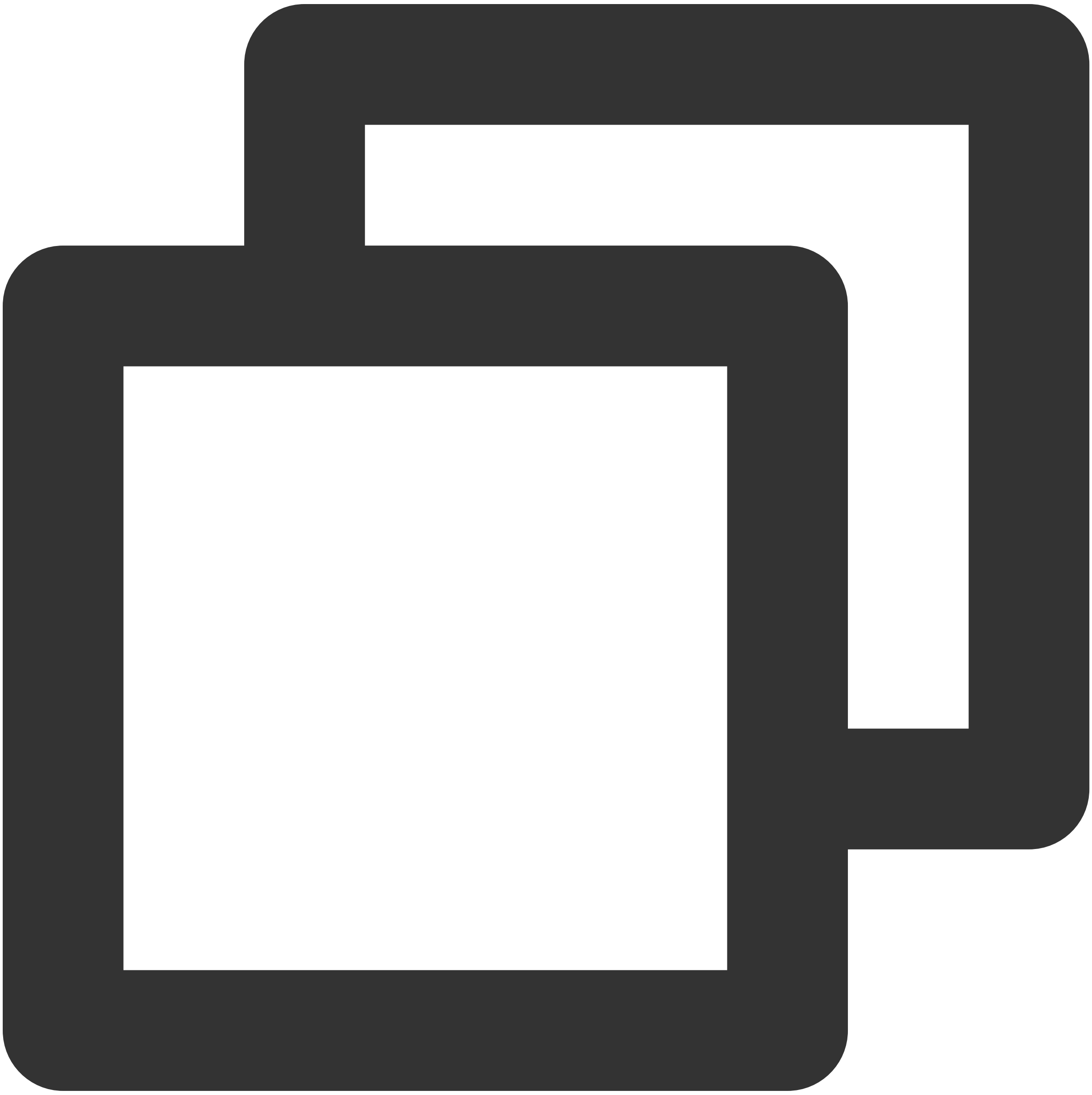
Since v7.9.348, the TRTC Electron SDK has integrated `trtc.d.ts` for developers using TypeScript.



```
import TRTCCloud from 'trtc-electron-sdk';

const rtcCloud: TRTCCloud = new TRTCCloud();
// Get the SDK version number
rtcCloud.getSDKVersion();
```

Setting callbacks



```
subscribeEvents = (rtcCloud) => {
    rtcCloud.on('onError', (errcode, errmsg) => {
    console.info('trtc_demo: onError :' + errcode + " msg" + errmsg);
    });
    rtcCloud.on('onEnterRoom', (elapsed) => {
    console.info('trtc_demo: onEnterRoom elapsed:' + elapsed);
    });
    rtcCloud.on('onExitRoom', (reason) => {
    console.info('onExitRoom: userenter reason:' + reason);
    });
};
```



```
subscribeEvents(this.rtcCloud);
```

Creating and terminating a `TRTCCloud` singleton

API	Description
getTRTCShareInstance	Creates a TRTCCloud singleton object during dynamic DLL loading.
destroyTRTCShareInstance	Releases a TRTCCloud singleton object and frees up resources.

Room APIs

API	Description
enterRoom	Enters a room. If the room does not exist, the system will create one automatically.
exitRoom	Leaves a room.
switchRoom	Switches rooms.
switchRole	Switches roles. This API applies only to the live streaming modes (`TRTCAppSceneLIVE` and `TRTCAppSceneVoiceChatRoom`).
connectOtherRoom	Requests cross-room communication.
disconnectOtherRoom	Ends cross-room communication.
setDefaultStreamRecvMode	Sets the audio/video receiving mode (must be called before room entry to take effect).

CDN APIs

API	Description
startPublishing	Starts publishing to Tencent Cloud’s live streaming CDN.
stopPublishing	Stops publishing to Tencent Cloud’s live streaming CDN.
startPublishCDNStream	Starts relaying to the live streaming CDN of a non-Tencent Cloud vendor.
stopPublishCDNStream	Stops relaying to the live streaming CDN of a non-Tencent Cloud vendor.
setMixTranscodingConfig	Sets On-Cloud MixTranscoding parameters.

Video APIs

API	Description
startLocalPreview	Enables capturing and preview of the local camera.
stopLocalPreview	Disables capturing and preview of the local camera.
muteLocalVideo	Pauses/Resumes publishing the local video.
startRemoteView	Starts playing the video of a remote user.
stopRemoteView	Stops playing and pulling the video of a remote user.
stopAllRemoteView	Stops playing and pulling the videos of all remote users.
muteRemoteVideoStream	Pauses/Resumes receiving the video of a specified remote user.
muteAllRemoteVideoStreams	Pauses/Resumes receiving the videos of all remote users.
setVideoEncoderParam	Sets video encoder parameters.
setNetworkQosParam	Sets video preference.
setLocalRenderParams	Sets rendering parameters for the local video (primary stream).
setLocalViewFillMode	Sets the rendering mode of the local video (deprecated).
setRemoteRenderParams	Sets rendering parameters for a remote video.
setRemoteViewFillMode	Sets the rendering mode of a remote video (deprecated).
setLocalViewRotation	Sets the clockwise rotation of the local video (deprecated).
setRemoteViewRotation	Sets the clockwise rotation of a remote video (deprecated).
setVideoEncoderRotation	Sets the rotation of encoded video images, i.e., images shown to remote users and recorded by the server.
setLocalViewMirror	Sets the mirror mode of the local camera's preview image (deprecated).
setVideoEncoderMirror	Sets the mirror mode of encoded images.
enableSmallVideoStream	Enables/Disables the dual-stream mode (low-quality and high-quality streams).
setRemoteVideoStreamType	Sets whether to view the high-quality or low-quality video of a specified user (`userId`).
setPriorRemoteVideoStreamType	Sets video quality preference for the audience (deprecated).

snapshotVideo	Takes a video screenshot.

Audio APIs

API	Description
startLocalAudio	Enables local audio capturing and publishing.
stopLocalAudio	Disables local audio capturing and publishing.
muteLocalAudio	Mutes/Unmutes the local user.
muteRemoteAudio	Mutes a remote user and stops pulling the user’s audio.
muteAllRemoteAudio	Mutes all remote users and stops pulling their audios.
setAudioCaptureVolume	Sets the SDK capturing volume.
getAudioCaptureVolume	Gets the SDK capturing volume.
setAudioPlayoutVolume	Sets the SDK playback volume.
getAudioPlayoutVolume	Gets the SDK playback volume.
enableAudioVolumeEvaluation	Enables/Disables the volume reminder.
startAudioRecording	Starts audio recording.
stopAudioRecording	Stops audio recording.
setAudioQuality	Sets audio quality (deprecated).
setRemoteAudioVolume	Sets the playback volume of a remote user.

Camera APIs

API	Description
getCameraDevicesList	Gets the camera list.
setCurrentCameraDevice	Sets the camera to use.
getCurrentCameraDevice	Gets the camera currently in use.

Audio device APIs

API	Description
getMicDevicesList	Gets the mic list.
getCurrentMicDevice	Gets the mic currently in use.
setCurrentMicDevice	Sets the mic to use.
getCurrentMicDeviceVolume	Gets the current mic volume.
setCurrentMicDeviceVolume	Sets the current mic volume.
setCurrentMicDeviceMute	Mutes/Unmutes the current mic.
getCurrentMicDeviceMute	Gets whether the current mic is muted.
getSpeakerDevicesList	Gets the speaker list.
getCurrentSpeakerDevice	Gets the speaker currently in use.
setCurrentSpeakerDevice	Sets the speaker to use.
getCurrentSpeakerVolume	Gets the current speaker volume.
setCurrentSpeakerVolume	Sets the current speaker volume.
setCurrentSpeakerDeviceMute	Mutes/Unmutes the current speaker.
getCurrentSpeakerDeviceMute	Gets whether the current speaker is muted.

Beauty filter APIs

API	Description
setBeautyStyle	Sets the strength of the beauty, skin brightening, and rosy skin filters.
setWaterMark	Sets the watermark.

Substream APIs

API	Description
startRemoteSubStreamView	Starts rendering the substream (screen sharing) video of a remote user (deprecated).
stopRemoteSubStreamView	Stops rendering the substream (screen sharing) video of a remote user (deprecated).

setRemoteSubStreamViewFillMode	Sets the rendering mode of the substream (screen sharing) video (deprecated).
setRemoteSubStreamViewRotation	Sets the clockwise rotation of the substream (screen sharing) video (deprecated).
getScreenCaptureSources	Enumerates shareable sources.
selectScreenCaptureTarget	Sets screen sharing parameters. This API can be called during screen sharing.
startScreenCapture	Starts screen sharing.
pauseScreenCapture	Pauses screen sharing.
resumeScreenCapture	Resumes screen sharing.
stopScreenCapture	Stops screen sharing.
setSubStreamEncoderParam	Sets encoder parameters for the substream (screen sharing) video.
setSubStreamMixVolume	Sets the audio mixing volume of the substream (screen sharing) video.
addExcludedShareWindow	Adds a specified window to the exclusion list of screen sharing. Windows in the list will not be shared.
removeExcludedShareWindow	Removes a specified window from the exclusion list of screen sharing.
removeAllExcludedShareWindow	Removes all windows from the exclusion list of screen sharing.

Custom message sending APIs

API	Description
sendCustomCmdMsg	Sends a custom message to all users in a room.
sendSEIMsg	Embeds small-volume custom data into video frames.

Background music mixing APIs

API	Description
playBGM	Starts background music (deprecated).
stopBGM	Stops background music (deprecated).
pauseBGM	Pauses background music (deprecated).

resumeBGM	Resumes background music (deprecated).
getBGMDuration	Gets the total length of the background music file, in milliseconds (deprecated).
setBGMPosition	Sets the playback progress of background music (deprecated).
setBGMVolume	Sets background music volume (deprecated).
setBGMPlayoutVolume	Sets the local playback volume of background music (deprecated).
setBGMPublishVolume	Sets the remote playback volume of background music (deprecated).
startSystemAudioLoopback	Enables system audio capturing.
stopSystemAudioLoopback	Disables system audio capturing.
setSystemAudioLoopbackVolume	Sets system audio capturing volume.
startPlayMusic	Starts background music.
stopPlayMusic	Stops background music.
pausePlayMusic	Pauses background music.
resumePlayMusic	Resumes background music.
getMusicDurationInMS	Gets the total length of the background music file, in milliseconds.
seekMusicToPosInTime	Sets the playback progress of background music.
setAllMusicVolume	Sets background music volume. This API is used to control the audio mixing volume of background music.
setMusicPlayoutVolume	Sets the local playback volume of background music.
setMusicPublishVolume	Sets the remote playback volume of background music.

Audio effect APIs

API	Description
playAudioEffect	Plays an audio effect (deprecated).
setAudioEffectVolume	Sets the volume of an audio effect (deprecated).
stopAudioEffect	Stops an audio effect (deprecated).
stopAllAudioEffects	Stops all audio effects (deprecated).

setAllAudioEffectsVolume	Sets the volume of all audio effects (deprecated).
pauseAudioEffect	Pauses an audio effect (deprecated).
resumeAudioEffect	Resumes an audio effect (deprecated).

Device and network testing APIs

API	Description
startSpeedTest	Starts network speed testing. This may compromise the quality of video calls and should be avoided during a video call.
stopSpeedTest	Stops network speed testing.
startCameraDeviceTest	Starts camera testing.
stopCameraDeviceTest	Stops camera testing.
startMicDeviceTest	Starts mic testing.
stopMicDeviceTest	Stops mic testing.
startSpeakerDeviceTest	Starts speaker testing.
stopSpeakerDeviceTest	Stops speaker testing.

Log APIs

API	Description
getSDKVersion	Gets the SDK version.
setLogLevel	Sets the log output level.
setConsoleEnabled	Enables/Disables console log printing.
setLogCompressEnabled	Enables/Disables local log compression.
setLogDirPath	Sets the path to save logs.
setLogCallback	Sets the log callback.
callExperimentalAPI	Calls the experimental API.

Disused APIs

API	Description
setMicVolumeOnMixing	This API has been deprecated since v6.9.

TRTCCallback @ TXLiteAVSDK

TRTC callback API classes

Error and warning event callback APIs

API	Description
onError	Error callback. This indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI messages should be sent to users if necessary.
onWarning	Warning callback. This alerts you to non-serious problems such as stutter or recoverable decoding failure.

Room event callback APIs

API	Description
onEnterRoom	Callback for room entry
onExitRoom	Callback for room exit
onSwitchRole	Callback for role switching
onConnectOtherRoom	Callback of the result of a cross-room communication request
onDisconnectOtherRoom	Callback of the result of ending cross-room communication
onSwitchRoom	Callback for room switching

Member event callback APIs

API	Description
onRemoteUserEnterRoom	Callback for the entry of a user
onRemoteUserLeaveRoom	Callback for the exit of a user
onUserVideoAvailable	Callback of whether a user has turned their camera on.
onUserSubStreamAvailable	Callback of whether a user has started screen sharing

onUserAudioAvailable	Callback of whether a user is sending audio data
onFirstVideoFrame	Callback for rendering the first video frame of the local user or a remote user
onFirstAudioFrame	Callback for playing the first audio frame of a remote user. No notifications are sent for local audio.
onSendFirstLocalVideoFrame	Callback for sending the first local video frame
onSendFirstLocalAudioFrame	Callback for sending the first local audio frame
onUserEnter	Callback for the entry of an anchor (deprecated)
onUserExit	Callback for the exit of an anchor (deprecated)

Callback APIs for statistics on network quality and technical metrics

API	Description
onNetworkQuality	Callback of network quality. This callback is triggered every 2 seconds to collect statistics on the quality of current upstream and downstream data transfer.
onStatistics	Callback of statistics on technical metrics

Server event callback APIs

API	Description
onConnectionLost	Callback for the disconnection of the SDK from the server
onTryToReconnect	Callback for the SDK trying to reconnect to the server
onConnectionRecovery	Callback for the reconnection of the SDK to the server
onSpeedTest	Callback of server speed test results (deprecated). The SDK tests the speed of multiple server addresses, and the result of each test is returned through this callback.
onSpeedTestResult	Callback of network speed test results.

Hardware event callback APIs

API	Description
onCameraDidReady	Callback for the camera being ready

onMicDidReady	Callback for the mic being ready
onUserVoiceVolume	Callback of volumes, including the volume of each user (`userId`) and the total remote volume. If `userid` is ``, it indicates the local user.
onDeviceChange	Callback for the connection/disconnection of a local device
onTestMicVolume	Volume callback for mic testing
onTestSpeakerVolume	Volume callback for speaker testing
onAudioDeviceCaptureVolumeChanged	Callback for volume change of the current audio capturing device
onAudioDevicePlayoutVolumeChanged	Callback for volume change of the current audio playback device

Custom message receiving callback APIs

API	Description
onRecvCustomCmdMsg	Callback for receiving a custom message
onMissCustomCmdMsg	Callback for losing a custom message
onRecvSEIMsg	Callback for receiving an SEI message

Callback APIs for relay to CDN

API	Description
onStartPublishing	Callback for starting publishing to Tencent Cloud's live streaming CDN. This callback is triggered by the `startPublishing()` API in `TRTCCloud` .
onStopPublishing	Callback for stopping publishing to Tencent Cloud's live streaming CDN. This callback is triggered by the `stopPublishing()` API in `TRTCCloud` .
onStartPublishCDNStream	Callback for relaying to a CDN
onStopPublishCDNStream	Callback for stopping relaying to a CDN
onSetMixTranscodingConfig	Callback for setting On-Cloud MixTranscoding parameters. This callback is triggered by the `setMixTranscodingConfig()` API in `TRTCCloud` .

Callback APIs for system audio capturing

API	Description

onSystemAudioLoopbackError	Callback of the system audio capturing result (only for macOS)

Audio effect callback APIs

API	Description
onAudioEffectFinished	Callback for the end of an audio effect (deprecated)

Screen sharing callback APIs

API	Description
onScreenCaptureCovered	Callback for the screen sharing window being covered. You can prompt users to move the window in this callback.
onScreenCaptureStarted	Callback for starting screen sharing
onScreenCapturePaused	Callback for pausing screen sharing
onScreenCaptureResumed	Callback for resuming screen sharing
onScreenCaptureStopped	Callback for stopping screen sharing

Screenshot callback API

API	Description
onSnapshotComplete	Callback for taking a screenshot

Background music callback APIs

API	Description
onPlayBGMBegin	Callback for starting background music (deprecated)
onPlayBGMProgress	Callback of the playback progress of background music (deprecated)
onPlayBGMComplete	Callback for the end of background music (deprecated)

Definitions of Key Types

Key types

Type	Description
TRTCParams	Room entry parameters
TRTCVideoEncParam	Video encoding parameters
TRTCNetworkQosParam	QoS control parameters
TRTCQualityInfo	Video quality
TRTCVolumeInfo	Volume
TRTCSpeedTestResult	Network speed testing result
TRTCMixUser	Video layout for On-Cloud MixTranscoding
TRTCTranscodingConfig	On-Cloud MixTranscoding configuration
TRTCPublishCDNParam	Relay to CDN parameters
TRTCAudioRecordingParams	Audio recording parameters
TRTCLocalStatistics	Local audio/video statistics
TRTCRemoteStatistics	Remote audio/video statistics
TRTCStatistics	Statistics

Enumerated values

Enumerated Value	Description
TRTCVideoResolution	Video resolution
TRTCVideoResolutionMode	Video resolution mode
TRTCVideoStreamType	Video stream type
TRTCQuality	Video quality
TRTCVideoFillMode	Video image fill mode
TRTCBeautyStyle	Beauty filter (skin smoothing) algorithm
TRTCAppScene	Application scenario
TRTCRoleType	Role, which applies only to live streaming scenarios (`TRTCAppSceneLIVE`)

TRTCQosControlMode	QoS control mode
TRTCVideoQosPreference	Video quality preference
TRTCDeviceState	Device operation
TRTCDeviceType	Device type
TRTCWaterMarkSrcType	Watermark source type
TRTCTranscodingConfigMode	Configuration mode for stream mixing parameters

Error Codes

Last updated：2023-10-09 11:53:42

Error Codes

Basic error codes

Code	Value	Description
ERR_NULL	0	No error.

Error codes for room entry

`TRTCCloud.enterRoom()` will trigger this type of error code if room entry fails. You can use the callback functions `TRTCCloudDelegate.onEnterRoom()` and `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_ROOM_ENTER_FAIL	-3301	Failed to enter room.
ERR_ENTER_ROOM_PARAM_NULL	-3316	Empty room entry parameters. Please check whether valid parameters are passed in the `TRTCCloud.enterRoom():` API when it is called.
ERR_SDK_APPID_INVALID	-3317	Invalid `sdkAppId`.
ERR_ROOM_ID_INVALID	-3318	Invalid `roomId`.
ERR_USER_ID_INVALID	-3319	Invalid `userID`.
ERR_USER_SIG_INVALID	-3320	Invalid `userSig`.
ERR_ROOM_REQUEST_ENTER_ROOM_TIMEOUT	-3308	Room entry request timed out. Please check your network.
ERR_SERVER_INFO_PRIVILEGE_FLAG_ERROR	-100006	Failed to verify the permission ticket. Please check whether `privateMapKey` is correct.
ERR_SERVER_INFO_SERVICE_SUSPENDED	-100013	Service unavailable. Please check whether there are remaining minutes in

		your packages and whether your Tencent Cloud account has overdue payment.
ERR_SERVER_INFO_ECDH_GET_TINYID	-100018	`userSig` verification failed. Please check whether `userSig` is correct.

Error code for room exit

`TRTCCloud.exitRoom()` triggers this error code if room exit fails. You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_ROOM_REQUEST_QUIT_ROOM_TIMEOUT	-3325	Room exit request timed out.

Error codes for devices (camera, mic, and speaker)

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_CAMERA_START_FAIL	-1301	Failed to turn camera on. This error may occur when there is a problem with the camera configuration program (driver) on Windows or macOS. In this case, disable and reenable the camera, restart the camera, or update the configuration program.
ERR_CAMERA_NOT_AUTHORIZED	-1314	Camera not authorized. This error usually occurs on mobile devices and may be because users denied camera permission.
ERR_CAMERA_SET_PARAM_FAIL	-1315	Failed to set camera parameters (unsupported values or others).
ERR_CAMERA_OCCUPY	-1316	Camera occupied. Try using another camera.
ERR_MIC_START_FAIL	-1302	Failed to turn mic on. This error may occur when there is a problem with the mic configuration program (driver) on Windows or macOS. In this case, disable and reenable the mic, restart the mic, or update the configuration program.
ERR_MIC_NOT_AUTHORIZED	-1317	Mic not authorized. This error usually occurs on mobile devices and may be because users denied mic permission.
ERR_MIC_SET_PARAM_FAIL	-1318	Failed to set mic parameters.

ERR_MIC_OCCUPY	-1319	Mic already in use. This error may occur when the user is currently in a call on the mobile device, in which case TRTC will fail to turn the mic on.
ERR_MIC_STOP_FAIL	-1320	Failed to turn mic off.
ERR_SPEAKER_START_FAIL	-1321	Failed to turn speaker on. This error may occur when there is a problem with the speaker configuration program (driver) on Windows or macOS. In this case, disable and reenable the speaker, restart the speaker, or update the configuration program.
ERR_SPEAKER_SET_PARAM_FAIL	-1322	Failed to set speaker parameters.
ERR_SPEAKER_STOP_FAIL	-1323	Failed to turn speaker off.

Error codes for screen sharing

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_SCREEN_CAPTURE_START_FAIL	-1308	Failed to start screen recording. If this error occurs on a mobile device, it may be because users denied screen recording permission; if it occurs on Windows or macOS, check whether the parameters of the screen recording API are set as required.
ERR_SCREEN_CAPTURE_UNSURPORT	-1309	Screen recording failed. If you use Android, make sure its version is 5.0 or later; if you use iOS, make sure its version is 11.0 or later.
ERR_SERVER_CENTER_NO_PRIVILEDGE_PUSH_SUB_VIDEO	-102015	No permission to send substream video images.
ERR_SERVER_CENTER_ANOTHER_USER_PUSH_SUB_VIDEO	-102016	Another user is sending substream video images.
ERR_SCREEN_CAPTURE_STOPPED	-7001	Screen recording stopped

		by the system.

Error codes for encoding and decoding

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_VIDEO_ENCODE_FAIL	-1303	Failed to encode video frames. This error may occur when a user on iOS switches to another app, which may cause the system to release the hardware encoder. When the user switches back, this error may be thrown before the hardware encoder is restarted.
PUSH_ERR_UNSUPPORTED_RESOLUTION	-1305	Unsupported video resolution.
ERR_AUDIO_ENCODE_FAIL	-1304	Failed to encode audio frames. This error may occur when the SDK could not process the custom audio data passed in.
PUSH_ERR_UNSUPPORTED_SAMPLERATE	-1306	Unsupported audio sample rate.

Error codes for custom capturing

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_PIXEL_FORMAT_UNSUPPORTED	-1327	Unsupported pixel format.
ERR_BUFFER_TYPE_UNSUPPORTED	-1328	Unsupported buffer type.

Error codes for CDN binding and stream mixing

You can use the callback functions `TRTCCloudDelegate.onStartPublishing()` and `TRTCCloudDelegate.onSetMixTranscodingConfig()` to capture related notifications.

Code	Value	Description
ERR_PUBLISH_CDN_STREAM_REQUEST_TIME_OUT	-3321	Relay-to-CDN request timed out.
ERR_CLOUD_MIX_TRANSCODING_REQUEST_TIME_OUT	-3322	On-Cloud MixTranscoding request timed out.
ERR_PUBLISH_CDN_STREAM_SERVER_FAILED	-3323	Abnormal response packets for relay.

ERR_CLOUD_MIX_TRANSCODING_SERVER_FAILED	-3324	Abnormal response packets for On-Cloud MixTranscoding.
ERR_ROOM_REQUEST_START_PUBLISHING_TIMEOUT	-3333	Signaling of starting to push to Tencent Cloud’s live streaming CDN timed out.
ERR_ROOM_REQUEST_START_PUBLISHING_ERROR	-3334	Abnormal signaling of starting to push to Tencent Cloud’s live streaming CDN.
ERR_ROOM_REQUEST_STOP_PUBLISHING_TIMEOUT	-3335	Signaling of stopping pushing to Tencent Cloud’s live streaming CDN timed out.
ERR_ROOM_REQUEST_STOP_PUBLISHING_ERROR	-3336	Abnormal signaling of stopping pushing to Tencent Cloud’s live streaming CDN.

Error codes for cross-room communication

`TRTCCloud.ConnectOtherRoom()` will trigger this type of error code if cross-room co-anchoring fails. You can use the callback function `TRTCCloudDelegate.onConnectOtherRoom()` to capture related notifications.

Code	Value	Description
ERR_ROOM_REQUEST_CONN_ROOM_TIMEOUT	-3326	Cross-room communication request timed out.
ERR_ROOM_REQUEST_DISCONN_ROOM_TIMEOUT	-3327	Request to end cross-room communication timed out.
ERR_ROOM_REQUEST_CONN_ROOM_INVALID_PARAM	-3328	Invalid parameter.
ERR_CONNECT_OTHER_ROOM_AS_AUDIENCE	-3330	You are an audience member and cannot initiate or end cross-room communication. You need to switch to the anchor role using `switchRole()` .

ERR_SERVER_CENTER_CONN_ROOM_NOT_SUPPORT	-102031	Cross-room communication not supported.
ERR_SERVER_CENTER_CONN_ROOM_REACH_MAX_NUM	-102032	Reached the maximum number of cross-room calls.
ERR_SERVER_CENTER_CONN_ROOM_REACH_MAX_RETRY_TIMES	-102033	Reached the maximum number of retries for cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_REQ_TIMEOUT	-102034	Cross-room communication request timed out.
ERR_SERVER_CENTER_CONN_ROOM_REQ	-102035	Cross-room communication request format is incorrect.
ERR_SERVER_CENTER_CONN_ROOM_NO_SIG	-102036	No signature for cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_DECRYPT_SIG	-102037	Failed to decrypt signature for cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_NO_KEY	-102038	Decryption key for cross-room communication signature not found.
ERR_SERVER_CENTER_CONN_ROOM_PARSE_SIG	-102039	Signature parsing error for cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_INVALID_SIG_TIME	-102040	Incorrect timestamp of cross-room communication signature.
ERR_SERVER_CENTER_CONN_ROOM_SIG_GROUPID	-102041	Mismatch of room

		ID in cross-room communication signature.
ERR_SERVER_CENTER_CONN_ROOM_NOT_CONNED	-102042	Mismatch of username in cross-room communication signature.
ERR_SERVER_CENTER_CONN_ROOM_USER_NOT_CONNED	-102043	The user did not initiate cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_FAILED	-102044	Failed to start cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_CANCEL_FAILED	-102045	Failed to cancel cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_ROOM_NOT_EXIST	-102046	The room being connected for cross-room communication does not exist.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_REACH_MAX_ROOM	-102047	The room being connected reached the maximum number of cross-room calls.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_USER_NOT_EXIST	-102048	The user being called for cross-room communication does not exist.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_USER_DELETED	-102049	The user being called for cross-room communication was deleted.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_USER_FULL	-102050	All resources of the

		user being called for cross-room communication are occupied.
ERR_SERVER_CENTER_CONN_ROOM_INVALID_SEQ	-102051	Sequence number for cross-room communication not in sequential order.

Warning Codes

Warning codes do not require your special attention. You can choose whether to prompt the user depending on the situation.

Code	Value	Description
WARNING_HW_ENCODER_START_FAIL	1103	Failed to start hardware encoder. The SDK automatically switched to software encoder.
WARNING_VIDEO_ENCODER_SW_TO_HW	1107	Insufficient CPU for software encoder. The SDK automatically switched to hardware encoder.
WARNING_INSUFFICIENT_CAPTURE_FPS	1108	Insufficient frame rate of video captured by camera. This error may occur on Android devices with built-in beauty filter algorithms.
WARNING_SW_ENCODER_START_FAIL	1109	Failed to start software encoder.
WARNING_REDUCE_CAPTURE_RESOLUTION	1110	Camera resolution reduced for balance between frame rate and performance.
WARNING_CAMERA_DEVICE_EMPTY	1111	No available camera found.
WARNING_CAMERA_NOT_AUTHORIZED	1112	User did not grant the application camera access.
WARNING_MICROPHONE_DEVICE_EMPTY	1201	No available mic found.
WARNING_SPEAKER_DEVICE_EMPTY	1202	No available speaker found.
WARNING_MICROPHONE_NOT_AUTHORIZED	1203	User did not grant the application mic

		access.
WARNING_MICROPHONE_DEVICE_ABNORMAL	1204	No audio capturing device available (for example, because the device is occupied).
WARNING_SPEAKER_DEVICE_ABNORMAL	1205	No audio playback device available (for example, because the device is occupied).
WARNING_VIDEO_FRAME_DECODE_FAIL	2101	Failed to decode current video frame.
WARNING_AUDIO_FRAME_DECODE_FAIL	2102	Failed to decode current audio frame.
WARNING_VIDEO_PLAY_LAG	2105	Video playback stuttering.
WARNING_HW_DECODER_START_FAIL	2106	Failed to start hardware decoder. Software decoder is used instead.
WARNING_VIDEO_DECODER_HW_TO_SW	2108	Hardware decoder failed to decode first I-frame of current stream. The SDK automatically switched to software decoder.
WARNING_SW_DECODER_START_FAIL	2109	Failed to start software decoder.
WARNING_VIDEO_RENDER_FAIL	2110	Failed to render video.
WARNING_START_CAPTURE_IGNORED	4000	Video capturing already started. Request ignored.
WARNING_AUDIO_RECORDING_WRITE_FAIL	7001	Failed to write recorded audio to file.
WARNING_ROOM_DISCONNECT	5101	Network disconnected.
WARNING_IGNORE_UPSTREAM_FOR_AUDIENCE	6001	You are in the role of audience. The request to send audio/video data is ignored.
WARNING_NET_BUSY	1101	Bad network connection: Data upload blocked due to limited upstream bandwidth.
WARNING_RTMP_SERVER_RECONNECT	1102	Push error. The network is disconnected. Reconnecting... (max attempts: 3).

WARNING_LIVE_STREAM_SERVER_RECONNECT	2103	Pull error. The network is disconnected. Reconnecting... (max attempts: 3).
WARNING_RECV_DATA_LAG	2104	Unstable incoming packets. This may be caused by insufficient downstream bandwidth or unstable streams from the anchor.
WARNING_RTMP_DNS_FAIL	3001	Live streaming error. DNS resolution failed.
WARNING_RTMP_SEVER_CONN_FAIL	3002	Live streaming error. Failed to connect to server.
WARNING_RTMP_SHAKE_FAIL	3003	Live streaming error. Handshake with RTMP server failed.
WARNING_RTMP_SERVER_BREAK_CONNECT	3004	Live streaming error. Connection dropped by server.
WARNING_RTMP_READ_WRITE_FAIL	3005	Live streaming error. RTMP read/write failed. Disconnecting.
WARNING_RTMP_WRITE_FAIL	3006	Live streaming error. RTMP write failed. This is an internal error code of the SDK and is not thrown.
WARNING_RTMP_READ_FAIL	3007	Live streaming error. RTMP read failed. This is an internal error code of the SDK and is not thrown.
WARNING_RTMP_NO_DATA	3008	Live streaming error. Server disconnected as no data is sent for over 30 seconds.
WARNING_PLAY_LIVE_STREAM_INFO_CONNECT_FAIL	3009	Live streaming error. Failed to call `connect` to connect to server. This is an internal error code of the SDK and is not thrown.
WARNING_NO_STEAM_SOURCE_FAIL	3010	Live streaming error. Connection failed as there was no video in the stream address. This is an internal error code of the SDK and is not thrown.
WARNING_ROOM_RECONNECT	5102	Network disconnected.

		Reconnecting...
WARNING_ROOM_NET_BUSY	5103	Bad network connection: Data upload blocked due to limited upstream bandwidth.

Flutter

Overview

Last updated：2024-06-24 15:22:32

TRTCCloud

Basic APIs

API	Description
sharedInstance	Creates a TRTCCloud singleton.
destroySharedInstance	Destroys a TRTCCloud singleton.
registerListener	Registers an event listener.
unRegisterListener	Unregisters an event listener.

Room APIs

API	Description
enterRoom	Enters a TRTC room. If the room does not exist, the system will create one automatically.
exitRoom	Exits a TRTC room.
switchRole	Switches roles. This API works only in live streaming scenarios （TRTC_APP_SCENE_LIVE and TRTC_APP_SCENE_VOICE_CHATROOM）
setDefaultStreamRecvMode	Sets the audio/video data receiving mode, which must be set before room entry to take effect.
connectOtherRoom	Requests a cross-room call so that two different rooms can share audio and video streams (e.g., "anchor PK" scenarios).
disconnectOtherRoom	Exits a cross-room call.
switchRoom	Switches rooms.
createSubCloud	Create room subinstance (for concurrent multi-room listen/watch)

destroySubCloud	Terminate room subinstance

CDN APIs

API	Description
startPublishing	Starts pushing to Tencent Cloud's live streaming CDN.
stopPublishing	Stops pushing to Tencent Cloud's live streaming CDN.
startPublishCDNStream	Starts relaying to the live streaming CDN of a non-Tencent Cloud vendor.
stopPublishCDNStream	Stops relaying to the live streaming CDN of a non-Tencent Cloud vendor.
setMixTranscodingConfig	Sets On-Cloud MixTranscoding parameters.
startPublishMediaStream	Publish a stream.
updatePublishMediaStream	Modify publishing parameters
stopPublishMediaStream	Stop publishing

Video APIs

API	Description
startLocalPreview	Enable the preview image of local camera (mobile)
updateLocalView	Update the preview image of local camera
updateRemoteView	Update remote user's video rendering control
stopLocalPreview	Stop camera preview
muteLocalVideo	Pause/Resume publishing local video stream
startRemoteView	Subscribe to remote user's video stream and bind video rendering control
stopRemoteView	Stop subscribing to remote user's video stream and release rendering control
stopAllRemoteView	Stop subscribing to all remote users' video streams and release all rendering resources
setVideoMuteImage	Set placeholder image during local video pause
muteRemoteVideoStream	Pause/Resume subscribing to remote user's video stream
muteAllRemoteVideoStreams	Pause/Resume subscribing to all remote users' video streams

setVideoEncoderParam	Set the encoding parameters of video encoder
setNetworkQosParam	Set network quality control parameters
setLocalRenderParams	Set the rendering parameters of local video image
setRemoteRenderParams	Set the rendering mode of remote video image
setVideoEncoderRotation	Set the direction of image output by video encoder
setVideoEncoderMirror	Set the mirror mode of image output by encoder
setGSensorMode	Set the adaptation mode of G-sensor
enableEncSmallVideoStream	Enable dual-channel encoding mode with big and small images
setRemoteVideoStreamType	Switch the big/small image of specified remote user
snapshotVideo	Screencapture video
startLocalRecording	Start local media recording
stopLocalRecording	Stop local media recording

Audio APIs

API	Description
startLocalAudio	Enables local microphone capture and publishes the audio stream to the current room with the ability to set the sound quality.
stopLocalAudio	Disable local audio capturing and upstreaming
muteLocalAudio	Mute/Unmute local audio
setAudioRoute	Set audio route, i.e., earpiece at the top or speaker at the bottom
muteRemoteAudio	Mute/Unmute the specified remote user's audio
muteAllRemoteAudio	Mute/Unmute all users' audio
setRemoteAudioVolume	Set the playback volume of the specified remote user
setAudioCaptureVolume	Set the capturing volume of local audio
getAudioCaptureVolume	Get the capturing volume of local audio
setAudioPlayoutVolume	Set the playback volume of remote audio

getAudioPlayoutVolume	Get the playback volume of remote audio
enableAudioVolumeEvaluation	Enable volume reminder
startAudioRecording	Start audio recording
stopAudioRecording	Stop audio recording
setSystemVolumeType	Setting the system volume type (for mobile OS)
startSystemAudioLoopback	Enable system audio capturing
stopSystemAudioLoopback	Stop system audio capturing(iOS not supported)
setSystemAudioLoopbackVolume	Set the volume of system audio capturing

Device management APIs

API	Description
getDeviceManager	Gets the device management module. For details, see device management APIs

Beauty filter APIs

API	Description
getBeautyManager	Gets the beauty filter management object. For details, see the document on beauty filter management
setWatermark	Adds watermarks.

Custom capturing and rendering APIs

API	Description
setLocalVideoRenderListener	Set the callback of custom rendering for local video
setRemoteVideoRenderListener	Set the callback of custom rendering for remote video
unregisterTexture	Unregister custom rendering callbacks
enableCustomVideoProcess	Enable/DisEnable Custom Video Process
setAudioFrameListener	Set custom audio data callback

Music and voice effect APIs

API	Description
getAudioEffectManager	Gets the audio effect management class TXAudioEffectManager, which is used to manage background music, short audio effects, and voice effects. For details, see the document on audio effect management

Substream APIs

API	Description
startScreenCapture	Starts screen sharing.
stopScreenCapture	Stops screen sharing.
pauseScreenCapture	Pauses screen sharing.
resumeScreenCapture	Resumes screen sharing.
getScreenCaptureSources	Enumerate shareable screens and windows (for Windows only)
selectScreenCaptureTarget	Select the screen or window to share (for Windows only)

Custom message sending APIs

API	Description
sendCustomCmdMsg	Sends a custom message to all users in the room.
sendSEIMsg	Embeds small-volume custom data in video frames.

Network testing APIs

API	Description
startSpeedTest	Starts network speed testing. This may compromise the quality of video calls and should be avoided during a video call.
stopSpeedTest	Stops server speed testing.

Log APIs

API	Description
getSDKVersion	Gets the TRTC SDK version.

setLogLevel	Sets the log output level.
setLogDirPath	Changes the path to save logs.
setLogCompressEnabled	Enables/Disables local log compression.
setConsoleEnabled	Enables/Disables console log printing.
showDebugView	Display debug information floats (can display audio/video information and event information)
callExperimentalAPI	Call experimental APIs

TRTCCloudListener

Callback APIs for the TRTC video call feature

Error and warning event callback APIs

API	Description
onError	Error callback, which indicates that the SDK encountered an irrecoverable error and must be listened on. Corresponding UI reminders should be displayed based on the actual conditions
onWarning	Warning callback. This callback is used to alert you of some non-serious problems such as lag or recoverable decoding failure

Room event callback APIs

API	Description
onEnterRoom	Callback for room entry
onExitRoom	Callback for room exit
onSwitchRole	Callback of role switching
onConnectOtherRoom	Callback of the result of requesting a cross-room call (anchor competition)
onDisConnectOtherRoom	Callback of the result of ending a cross-room call (anchor competition)
onSwitchRoom	Callback of the result of room switching (switchRoom)

User event callback APIs

API	Description
onRemoteUserEnterRoom	Callback of the entry of a user
onRemoteUserLeaveRoom	Callback of the exit of a user
onUserVideoAvailable	Callback of whether a remote user has a playable primary image (usually the image of the camera)
onUserSubStreamAvailable	Callback of whether a remote user has a playable substream image (usually the screen sharing image)
onUserAudioAvailable	Callback of whether a remote user has playable audio
onFirstVideoFrame	Callback of rendering the first video frame of the local user or a remote user
onFirstAudioFrame	Callback of playing the first audio frame of a remote user. No notifications are sent for local audio.
onSendFirstLocalVideoFrame	Callback of sending the first local video frame
onSendFirstLocalAudioFrame	Callback of sending the first local audio frame

Callback APIs for recording task

API	Description
onLocalRecordBegin	Local recording started
onLocalRecording	Local media is being recorded
onLocalRecordFragment	Record fragment finished.
onLocalRecordComplete	Local recording stopped

Callback APIs for background music playback

API	Description
onMusicObserverStart	Callback of starting music playback
onMusicObserverPlayProgress	Callback of the music playback progress
onMusicObserverComplete	Callback of ending music playback

Callback APIs for statistics on network quality and technical metrics

API	Description
onNetworkQuality	Callback of network quality. This callback is triggered every 2 seconds to collect statistics on the quality of current upstream and downstream data transfer.
onStatistics	Callback of statistics on technical metrics

Server event callback APIs

API	Description
onConnectionLost	Callback of the disconnection of the SDK from the server
onTryToReconnect	Callback of the SDK trying to connect to the server again
onConnectionRecovery	Callback of the reconnection of the SDK to the server
onSpeedTest	Callback of server speed test results. The SDK tests the speed of multiple server addresses, and the result of each test is returned through this callback.。

Hardware event callback APIs

API	Description
onCameraDidReady	Callback of the camera being ready
onMicDidReady	Callback of the mic being ready
onUserVoiceVolume	Callback of volume, including the volume of each userId and the total remote volume
onDeviceChange	The status of a local device changed (for desktop OS only)
onTestMicVolume	Volume during mic test
onTestSpeakerVolume	Volume during speaker test
onAudioRouteChanged	The audio route changed (for mobile devices only)

Custom message receiving callback APIs

API	Description
onRecvCustomCmdMsg	Receipt of custom message
onMissCustomCmdMsg	Loss of custom message

onRecvSEIMsg	Receipt of SEI message

Callback APIs for CDN relayed push

API	Description
onStartPublishing	Started publishing to Tencent Cloud CSS CDN, which corresponds to the startPublishing() API in TRTCCloud
onStopPublishing	Stopped publishing to Tencent Cloud CSS CDN, which corresponds to the stopPublishing() API in TRTCCloud
onStartPublishCDNStream	Callback of the completion of starting relayed push to CDNs
onStopPublishCDNStream	Callback of the completion of stopping relayed push to CDNs
onSetMixTranscodingConfig	Callback of setting On-Cloud MixTranscoding parameters, which corresponds to the setMixTranscodingConfig() API in TRTCCloud
onStartPublishMediaStream	Setting up callbacks for mixing and streaming parameters in the cloud, which corresponds to the startPublishMediaStream() API in TRTCCloud
onUpdatePublishMediaStream	Setting up callbacks for mixing and streaming parameters in the cloud, which corresponds to the updatePublishMediaStream() API in TRTCCloud
onStopPublishMediaStream	Setting up callbacks for mixing and streaming parameters in the cloud, which corresponds to the stopPublishMediaStream() API in TRTCCloud

Screen sharing callback APIs

API	Description
onScreenCaptureStarted	Callback of starting screen sharing
onScreenCapturePaused	Callback of pausing screen sharing via the calling of pauseScreenCapture()
onScreenCaptureResumed	Callback of resuming screen sharing via the calling of resumeScreenCapture()
onScreenCaptureStopped	Callback of stopping screen sharing。

Screenshot callback API

API	Description
onSnapshotComplete	Callback of the completion of a screenshot

TXAudioEffectManager

API	Description
enableVoiceEarMonitor	Enable in-ear monitoring
setVoiceEarMonitorVolume	Set the in-ear monitoring volume
setVoiceReverbType	Set the voice reverb effect (karaoke room, small room, big hall, deep, resonant, and other effects)
setVoiceChangerType	Set the voice changing effect (young girl, middle-aged man, heavy metal, punk, and other effects)
setVoiceCaptureVolume	Set the mic voice volume
startPlayMusic	Start background music
stopPlayMusic	Stop background music
pausePlayMusic	Pause background music
resumePlayMusic	Resume background music
setMusicPublishVolume	Set the remote volume of background music. The anchor can use this API to set the volume of background music heard by the remote audience.
setMusicPlayoutVolume	Set the local volume of background music. The anchor can use this API to set the volume of local background music.
setAllMusicVolume	Set the local and remote volumes of global background music
setMusicPitch	Adjust the pitch of background music
setMusicSpeedRate	Adjust the speed of background music
getMusicCurrentPosInMS	Get the current playback progress of background music in milliseconds
seekMusicToPosInMS	Set the playback progress of background music in milliseconds
getMusicDurationInMS	Get the total duration of the background music file in milliseconds
setVoicePitch	Set the voice pitch.

TXBeautyManager

API	Description
setBeautyStyle	Set beauty filter type
setFilter	Specify material filter effect
setFilterStrength	Set the strength of filter
setBeautyLevel	Set the strength of the beauty filter
setWhitenessLevel	Set the strength of the brightening filter
enableSharpnessEnhancement	Enable definition enhancement
setRuddyLevel	Set the strength of the rosy skin filter

TXDeviceManager

API	Description
isFrontCamera	Set whether to use the front camera
switchCamera	Switch camera
getCameraZoomMaxRatio	Get the camera zoom factor
setCameraZoomRatio	Set the zoom factor (focal length) of camera
enableCameraAutoFocus	Set whether to enable the automatic recognition of face position
isAutoFocusEnabled	Query whether the device supports automatic recognition of face position
setCameraFocusPosition	Setting the camera focus position
enableCameraTorch	Enable/Disable flash
setSystemVolumeType	Set the system volume type used in call
setAudioRoute	Set audio route, i.e., earpiece at the top or speaker at the bottom
getDevicesList	Get the list of devices
setCurrentDevice	Specify the current device
getCurrentDevice	Get the currently used device
setCurrentDeviceVolume	Set the volume of the current device

getCurrentDeviceVolume	Get the volume of the current device
setCurrentDeviceMute	Set the mute status of the current device
getCurrentDeviceMute	Query the mute status of the current device
startMicDeviceTest	Start mic test
stopMicDeviceTest	Stop mic test
startSpeakerDeviceTest	Start speaker test
stopSpeakerDeviceTest	Stop speaker test
setApplicationPlayVolume	Set the volume of the current process in the Windows system volume mixer
getApplicationPlayVolume	Get the volume of the current process in the Windows system volume mixer
setApplicationMuteState	Set the mute status of the current process in the Windows system volume mixer
getApplicationMuteState	Get the mute status of the current process in the Windows system volume mixer

Definitions of Key Classes

API	Description
TRTCCloudDef	Key class definition variable
TRTCParams	Room entry parameters
TRTCSwitchRoomConfig	Room switch parameters
TRTCVideoEncParam	Encoding parameters
TRTCNetworkQosParam	Network bandwidth limit parameters
TRTCRenderParams	Remote image parameters
TRTCMixUser	Position information of each channel of subimage in On-Cloud MixTranscoding
TRTCTranscodingConfig	On-Cloud MixTranscoding configuration
TXVoiceChangerType	Voice changing type definition (young girl, middle-aged man, heavy metal, punk...)

TXVoiceReverbType	Reverb effect type definition (karaoke room, small room, big hall, deep, resonant...)
AudioMusicParam	Parameters of music and voice settings APIs
TRTCAudioRecordingParams	Audio recording parameters
TRTCLocalRecordingParams	Recording parameters
TRTCPublishCDNParam	CDN relaying parameters
CustomLocalRender	Parameters of local video rendering with external texture
CustomRemoteRender	Parameters of remote video rendering with external texture
CustomRender	Parameters of video rendering with external texture
TRTCPublishMode	Media stream publishing mode, this enumeration type is used for the Media Stream Publishing interface startPublishMediaStream
TRTCPublishCdnUrl	Configure to publish real-time audio/video (TRTC) streams to Tencent Cloud or a third-party CDN.
TRTCUser	Information about the TRTC user, mainly containing the user ID and the room number of the user.
TRTCPublishTarget	Configure the publication target for the TRTC stream
TRTCStreamEncoderParam	Encoding settings related to the published stream, including resolution, frame rate, keyframe interval, etc.
Rect	Coordinates used to describe some views
TRTCVideoFillMode	Enumeration of TRTC video view display modes, including fill mode and adaptation mode
TRTCVideoStreamType	The different types of video streams offered by the TRTC
TRTCVideoLayout	Configuration of video layout properties for TRTC streaming, including position, size, layers, etc.
TRTCWatermark	Configuration of the properties of the TRTC watermarking function
TRTCStreamMixingConfig	Settings related to TRTC mixing and streaming, including background color, background image, information about all video and audio streams to be mixed, and watermark settings.
TRTCAudioFrame	Audio/video frame data class for processing and transmitting audio data.

TRTCScreenCaptureSourceList	List of screen windows.
TRTCScreenCaptureSourceInfo	Target information for screen sharing (desktop only)
TRTCImageBuffer	TRTC screen sharing icon information and mute image shim
TRTCScreenCaptureProperty	Advanced control parameters for screen sharing
TRTCScreenCaptureSourceType	Screen sharing target type (desktop only)

TRTCCloudVideoView

API	Description
TRTCCloudVideoView	Video view window, which displays the local video, remote video, or substream

Error Codes

Last updated：2023-10-09 11:54:42

Error Codes

Basic error codes

Code	Value	Description
ERR_NULL	0	No error.

Error codes for room entry

`TRTCCloud.enterRoom()` will trigger this type of error code if room entry fails. You can use the callback functions `TRTCCloudDelegate.onEnterRoom()` and `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_ROOM_ENTER_FAIL	-3301	Failed to enter room.
ERR_ENTER_ROOM_PARAM_NULL	-3316	Empty room entry parameters. Please check whether valid parameters are passed in the `TRTCCloud.enterRoom():` API when it is called.
ERR_SDK_APPID_INVALID	-3317	Invalid `sdkAppId` .
ERR_ROOM_ID_INVALID	-3318	Invalid `roomId` .
ERR_USER_ID_INVALID	-3319	Invalid `userID` .
ERR_USER_SIG_INVALID	-3320	Invalid `userSig` .
ERR_ROOM_REQUEST_ENTER_ROOM_TIMEOUT	-3308	Room entry request timed out. Please check your network.
ERR_SERVER_INFO_PRIVILEGE_FLAG_ERROR	-100006	Failed to verify the permission ticket. Please check whether `privateMapKey` is correct.
ERR_SERVER_INFO_SERVICE_SUSPENDED	-100013	Service unavailable. Please check whether there are remaining minutes in

		your packages and whether your Tencent Cloud account has overdue payment.
ERR_SERVER_INFO_ECDH_GET_TINYID	-100018	`userSig` verification failed. Please check whether `userSig` is correct.

Error code for room exit

`TRTCCloud.exitRoom()` triggers this error code if room exit fails. You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_ROOM_REQUEST_QUIT_ROOM_TIMEOUT	-3325	Room exit request timed out.

Error codes for devices (camera, mic, and speaker)

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_CAMERA_START_FAIL	-1301	Failed to turn camera on. This error may occur when there is a problem with the camera configuration program (driver) on Windows or macOS. In this case, disable and reenable the camera, restart the camera, or update the configuration program.
ERR_CAMERA_NOT_AUTHORIZED	-1314	Camera not authorized. This error usually occurs on mobile devices and may be because users denied camera permission.
ERR_CAMERA_SET_PARAM_FAIL	-1315	Failed to set camera parameters (unsupported values or others).
ERR_CAMERA_OCCUPY	-1316	Camera occupied. Try using another camera.
ERR_MIC_START_FAIL	-1302	Failed to turn mic on. This error may occur when there is a problem with the mic configuration program (driver) on Windows or macOS. In this case, disable and reenable the mic, restart the mic, or update the configuration program.
ERR_MIC_NOT_AUTHORIZED	-1317	Mic not authorized. This error usually occurs on mobile devices and may be because users denied mic permission.
ERR_MIC_SET_PARAM_FAIL	-1318	Failed to set mic parameters.

ERR_MIC_OCCUPY	-1319	Mic already in use. This error may occur when the user is currently in a call on the mobile device, in which case TRTC will fail to turn the mic on.
ERR_MIC_STOP_FAIL	-1320	Failed to turn mic off.
ERR_SPEAKER_START_FAIL	-1321	Failed to turn speaker on. This error may occur when there is a problem with the speaker configuration program (driver) on Windows or macOS. In this case, disable and reenable the speaker, restart the speaker, or update the configuration program.
ERR_SPEAKER_SET_PARAM_FAIL	-1322	Failed to set speaker parameters.
ERR_SPEAKER_STOP_FAIL	-1323	Failed to turn speaker off.

Error codes for screen sharing

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_SCREEN_CAPTURE_START_FAIL	-1308	Failed to start screen recording. If this error occurs on a mobile device, it may be because users denied screen recording permission; if it occurs on Windows or macOS, check whether the parameters of the screen recording API are set as required.
ERR_SCREEN_CAPTURE_UNSURPORT	-1309	Screen recording failed. If you use Android, make sure its version is 5.0 or later; if you use iOS, make sure its version is 11.0 or later.
ERR_SERVER_CENTER_NO_PRIVILEDGE_PUSH_SUB_VIDEO	-102015	No permission to send substream video images.
ERR_SERVER_CENTER_ANOTHER_USER_PUSH_SUB_VIDEO	-102016	Another user is sending substream video images.
ERR_SCREEN_CAPTURE_STOPPED	-7001	Screen recording stopped

		by the system.

Error codes for encoding and decoding

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_VIDEO_ENCODE_FAIL	-1303	Failed to encode video frames. This error may occur when a user on iOS switches to another app, which may cause the system to release the hardware encoder. When the user switches back, this error may be thrown before the hardware encoder is restarted.
PUSH_ERR_UNSUPPORTED_RESOLUTION	-1305	Unsupported video resolution.
ERR_AUDIO_ENCODE_FAIL	-1304	Failed to encode audio frames. This error may occur when the SDK could not process the custom audio data passed in.
PUSH_ERR_UNSUPPORTED_SAMPLERATE	-1306	Unsupported audio sample rate.

Error codes for custom capturing

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_PIXEL_FORMAT_UNSUPPORTED	-1327	Unsupported pixel format.
ERR_BUFFER_TYPE_UNSUPPORTED	-1328	Unsupported buffer type.

Error codes for CDN binding and stream mixing

You can use the callback functions `TRTCCloudDelegate.onStartPublishing()` and `TRTCCloudDelegate.onSetMixTranscodingConfig()` to capture related notifications.

Code	Value	Description
ERR_PUBLISH_CDN_STREAM_REQUEST_TIME_OUT	-3321	Relay-to-CDN request timed out.
ERR_CLOUD_MIX_TRANSCODING_REQUEST_TIME_OUT	-3322	On-Cloud MixTranscoding request timed out.
ERR_PUBLISH_CDN_STREAM_SERVER_FAILED	-3323	Abnormal response packets for relay.

ERR_CLOUD_MIX_TRANSCODING_SERVER_FAILED	-3324	Abnormal response packets for On-Cloud MixTranscoding.
ERR_ROOM_REQUEST_START_PUBLISHING_TIMEOUT	-3333	Signaling of starting to push to Tencent Cloud's live streaming CDN timed out.
ERR_ROOM_REQUEST_START_PUBLISHING_ERROR	-3334	Abnormal signaling of starting to push to Tencent Cloud's live streaming CDN.
ERR_ROOM_REQUEST_STOP_PUBLISHING_TIMEOUT	-3335	Signaling of stopping pushing to Tencent Cloud's live streaming CDN timed out.
ERR_ROOM_REQUEST_STOP_PUBLISHING_ERROR	-3336	Abnormal signaling of stopping pushing to Tencent Cloud's live streaming CDN.

Error codes for cross-room communication

`TRTCCloud.ConnectOtherRoom()` will trigger this type of error code if cross-room co-anchoring fails. You can use the callback function `TRTCCloudDelegate.onConnectOtherRoom()` to capture related notifications.

Code	Value	Description
ERR_ROOM_REQUEST_CONN_ROOM_TIMEOUT	-3326	Cross-room communication request timed out.
ERR_ROOM_REQUEST_DISCONN_ROOM_TIMEOUT	-3327	Request to end cross-room communication timed out.
ERR_ROOM_REQUEST_CONN_ROOM_INVALID_PARAM	-3328	Invalid parameter.
ERR_CONNECT_OTHER_ROOM_AS_AUDIENCE	-3330	You are an audience member and cannot initiate or end cross-room communication. You need to switch to the anchor role using `switchRole()`.

ERR_SERVER_CENTER_CONN_ROOM_NOT_SUPPORT	-102031	Cross-room communication not supported.
ERR_SERVER_CENTER_CONN_ROOM_REACH_MAX_NUM	-102032	Reached the maximum number of cross-room calls.
ERR_SERVER_CENTER_CONN_ROOM_REACH_MAX_RETRY_TIMES	-102033	Reached the maximum number of retries for cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_REQ_TIMEOUT	-102034	Cross-room communication request timed out.
ERR_SERVER_CENTER_CONN_ROOM_REQ	-102035	Cross-room communication request format is incorrect.
ERR_SERVER_CENTER_CONN_ROOM_NO_SIG	-102036	No signature for cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_DECRYPT_SIG	-102037	Failed to decrypt signature for cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_NO_KEY	-102038	Decryption key for cross-room communication signature not found.
ERR_SERVER_CENTER_CONN_ROOM_PARSE_SIG	-102039	Signature parsing error for cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_INVALID_SIG_TIME	-102040	Incorrect timestamp of cross-room communication signature.
ERR_SERVER_CENTER_CONN_ROOM_SIG_GROUPID	-102041	Mismatch of room

		ID in cross-room communication signature.
ERR_SERVER_CENTER_CONN_ROOM_NOT_CONNED	-102042	Mismatch of username in cross-room communication signature.
ERR_SERVER_CENTER_CONN_ROOM_USER_NOT_CONNED	-102043	The user did not initiate cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_FAILED	-102044	Failed to start cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_CANCEL_FAILED	-102045	Failed to cancel cross-room communication.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_ROOM_NOT_EXIST	-102046	The room being connected for cross-room communication does not exist.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_REACH_MAX_ROOM	-102047	The room being connected reached the maximum number of cross-room calls.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_USER_NOT_EXIST	-102048	The user being called for cross-room communication does not exist.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_USER_DELETED	-102049	The user being called for cross-room communication was deleted.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_USER_FULL	-102050	All resources of the

		user being called for cross-room communication are occupied.
ERR_SERVER_CENTER_CONN_ROOM_INVALID_SEQ	-102051	Sequence number for cross-room communication not in sequential order.

Warning Codes

Warning codes do not require your special attention. You can choose whether to prompt the user depending on the situation.

Code	Value	Description
WARNING_HW_ENCODER_START_FAIL	1103	Failed to start hardware encoder. The SDK automatically switched to software encoder.
WARNING_VIDEO_ENCODER_SW_TO_HW	1107	Insufficient CPU for software encoder. The SDK automatically switched to hardware encoder.
WARNING_INSUFFICIENT_CAPTURE_FPS	1108	Insufficient frame rate of video captured by camera. This error may occur on Android devices with built-in beauty filter algorithms.
WARNING_SW_ENCODER_START_FAIL	1109	Failed to start software encoder.
WARNING_REDUCE_CAPTURE_RESOLUTION	1110	Camera resolution reduced for balance between frame rate and performance.
WARNING_CAMERA_DEVICE_EMPTY	1111	No available camera found.
WARNING_CAMERA_NOT_AUTHORIZED	1112	User did not grant the application camera access.
WARNING_MICROPHONE_DEVICE_EMPTY	1201	No available mic found.
WARNING_SPEAKER_DEVICE_EMPTY	1202	No available speaker found.
WARNING_MICROPHONE_NOT_AUTHORIZED	1203	User did not grant the application mic

		access.
WARNING_MICROPHONE_DEVICE_ABNORMAL	1204	No audio capturing device available (for example, because the device is occupied).
WARNING_SPEAKER_DEVICE_ABNORMAL	1205	No audio playback device available (for example, because the device is occupied).
WARNING_VIDEO_FRAME_DECODE_FAIL	2101	Failed to decode current video frame.
WARNING_AUDIO_FRAME_DECODE_FAIL	2102	Failed to decode current audio frame.
WARNING_VIDEO_PLAY_LAG	2105	Video playback stuttering.
WARNING_HW_DECODER_START_FAIL	2106	Failed to start hardware decoder. Software decoder is used instead.
WARNING_VIDEO_DECODER_HW_TO_SW	2108	Hardware decoder failed to decode first I-frame of current stream. The SDK automatically switched to software decoder.
WARNING_SW_DECODER_START_FAIL	2109	Failed to start software decoder.
WARNING_VIDEO_RENDER_FAIL	2110	Failed to render video.
WARNING_START_CAPTURE_IGNORED	4000	Video capturing already started. Request ignored.
WARNING_AUDIO_RECORDING_WRITE_FAIL	7001	Failed to write recorded audio to file.
WARNING_ROOM_DISCONNECT	5101	Network disconnected.
WARNING_IGNORE_UPSTREAM_FOR_AUDIENCE	6001	You are in the role of audience. The request to send audio/video data is ignored.
WARNING_NET_BUSY	1101	Bad network connection: Data upload blocked due to limited upstream bandwidth.
WARNING_RTMP_SERVER_RECONNECT	1102	Push error. The network is disconnected. Reconnecting... (max attempts: 3).

WARNING_LIVE_STREAM_SERVER_RECONNECT	2103	Pull error. The network is disconnected. Reconnecting... (max attempts: 3).
WARNING_RECV_DATA_LAG	2104	Unstable incoming packets. This may be caused by insufficient downstream bandwidth or unstable streams from the anchor.
WARNING_RTMP_DNS_FAIL	3001	Live streaming error. DNS resolution failed.
WARNING_RTMP_SEVER_CONN_FAIL	3002	Live streaming error. Failed to connect to server.
WARNING_RTMP_SHAKE_FAIL	3003	Live streaming error. Handshake with RTMP server failed.
WARNING_RTMP_SERVER_BREAK_CONNECT	3004	Live streaming error. Connection dropped by server.
WARNING_RTMP_READ_WRITE_FAIL	3005	Live streaming error. RTMP read/write failed. Disconnecting.
WARNING_RTMP_WRITE_FAIL	3006	Live streaming error. RTMP write failed. This is an internal error code of the SDK and is not thrown.
WARNING_RTMP_READ_FAIL	3007	Live streaming error. RTMP read failed. This is an internal error code of the SDK and is not thrown.
WARNING_RTMP_NO_DATA	3008	Live streaming error. Server disconnected as no data is sent for over 30 seconds.
WARNING_PLAY_LIVE_STREAM_INFO_CONNECT_FAIL	3009	Live streaming error. Failed to call `connect` to connect to server. This is an internal error code of the SDK and is not thrown.
WARNING_NO_STEAM_SOURCE_FAIL	3010	Live streaming error. Connection failed as there was no video in the stream address. This is an internal error code of the SDK and is not thrown.
WARNING_ROOM_RECONNECT	5102	Network disconnected.

		Reconnecting...
WARNING_ROOM_NET_BUSY	5103	Bad network connection: Data upload blocked due to limited upstream bandwidth.

Unity

Overview

Last updated：2023-10-09 11:55:23

Overview

Basic APIs

API	Description
getTRTCShareInstance	Creates a `TRTCCloud` singleton.
destroyTRTCShareInstance	Releases a `TRTCCloud` singleton.
addCallback	Sets the callback API `TRTCCloudCallback` .
removeCallback	Removes event callback.

Room APIs

API	Description
enterRoom	Enters a room. If the room does not exist, the system will create one automatically.
exitRoom	Exits a room.
switchRole	Switches roles. This API works only in live streaming scenarios (`TRTC_APP_SCENE_LIVE` and `TRTC_APP_SCENE_VOICE_CHATROOM`).
setDefaultStreamRecvMode	Sets the audio/video data receiving mode, which must be set before room entry to take effect.
connectOtherRoom	Requests a cross-room call (anchor competition).
disconnectOtherRoom	Exits a cross-room call.
switchRoom	Switches rooms.

CDN APIs

API	Description

startPublishing	Starts pushing to Tencent Cloud's live streaming CDN.
stopPublishing	Stops pushing to Tencent Cloud's live streaming CDN.
startPublishCDNStream	Starts relaying to the live streaming CDN of a non-Tencent Cloud vendor.
stopPublishCDNStream	Stops relaying to non-Tencent Cloud addresses.
setMixTranscodingConfig	Sets On-Cloud MixTranscoding parameters.

Video APIs

API	Description
startLocalPreview	Enables local video preview (only custom rendering is supported currently).
stopLocalPreview	Stops local video capturing and preview.
muteLocalVideo	Pauses/Resumes sending local video data.
startRemoteView	Starts pulling and displaying the image of a specified remote user (only custom rendering is supported currently).
stopRemoteView	Stops displaying and pulling the video image of a remote user.
stopAllRemoteView	Stops displaying and pulling the video images of all remote users.
muteRemoteVideoStream	Pauses/Resumes receiving the video stream of a specified remote user.
muteAllRemoteVideoStreams	Pauses/Resumes receiving all remote video streams.
setVideoEncoderParam	Sets video encoder parameters.
setNetworkQosParam	Sets QoS control parameters.
setVideoEncoderMirror	Sets the mirror mode of encoded images.

Audio APIs

API	Description
startLocalAudio	Enables local audio capturing and upstream data transfer.
stopLocalAudio	Disables local audio capturing and upstream data transfer.
muteLocalAudio	Mutes/Unmutes local audio.
muteRemoteAudio	Mutes/Unmutes a specified remote user.

muteAllRemoteAudio	Mutes/Unmutes all remote users.
setRemoteAudioVolume	Sets the playback volume of a remote user.
setAudioCaptureVolume	Sets the SDK capturing volume.
getAudioCaptureVolume	Gets the SDK capturing volume.
setAudioPlayoutVolume	Sets the SDK playback volume.
getAudioPlayoutVolume	Gets the SDK playback volume.
enableAudioVolumeEvaluation	Enables volume reminders.
startAudioRecording	Starts audio recording.
stopAudioRecording	Stops audio recording.

Device management APIs

API	Description
getDeviceManager	Gets the device management module. For details, please see Specific Device Management APIs.

Music and voice effect APIs

API	Description
getAudioEffectManager	Gets the audio effect management class `TXAudioEffectManager` , which is used to manage background music, short audio effects, and voice effects. For details, please see Specific Music and Voice Effect APIs.

Custom video rendering APIs

API	Description
setLocalVideoRenderCallback	Sets custom rendering for the local video.
setRemoteVideoRenderCallback	Sets custom rendering for the video of a remote user.

Custom message sending APIs

API	Description

sendSEIMsg	Embeds small-volume custom data in video frames.

Network testing APIs

API	Description
startSpeedTest	Starts network speed testing. This may compromise the quality of video calls and should be avoided during a video call.
stopSpeedTest	Stops server speed testing.

Log APIs

API	Description
getSDKVersion	Gets the SDK version.
setLogLevel	Sets the log output level.
setLogDirPath	Changes the path to save logs.
setLogCompressEnabled	Enables/Disables local log compression.
callExperimentalAPI	Calls the experimental API.

TRTCCloudCallback

Callback APIs for the TRTC audio call feature

Error and warning event callback APIs

API	Description
onError	Error callback. This indicates that the SDK encountered an irrecoverable error. Such errors must be listened for, and UI reminders should be displayed to users if necessary.
onWarning	Warning callback. This alerts you to non-serious problems such as lag or recoverable decoding failure.

Room event callback APIs

API	Description
onEnterRoom	Callback of room entry

onExitRoom	Callback of room exit
onSwitchRole	Callback of role switching
onConnectOtherRoom	Callback of the result of requesting a cross-room call (anchor competition)
onDisConnectOtherRoom	Callback of the result of ending a cross-room call (anchor competition)
onSwitchRoom	Callback of the result of room switching (`switchRoom`)

User event callback APIs

API	Description
onRemoteUserEnterRoom	Callback of the entry of a user
onRemoteUserLeaveRoom	Callback of the exit of a user
onUserVideoAvailable	Callback of whether a user has turned the camera on
onUserAudioAvailable	Callback of whether a remote user has playable audio
onFirstVideoFrame	Callback of rendering the first video frame of the local user or a remote user
onFirstAudioFrame	Callback of playing the first audio frame of a remote user. No notifications are sent for local audio.
onSendFirstLocalVideoFrame	Callback of sending the first local video frame
onSendFirstLocalAudioFrame	Callback of sending the first local audio frame

Callback APIs for statistics on network quality and technical metrics

API	Description
onNetworkQuality	Callback of network quality. This callback is triggered every 2 seconds to collect statistics on the current upstream and downstream data transfer.
onStatistics	Callback of statistics on technical metrics

Server event callback APIs

API	Description
onConnectionLost	Callback of the disconnection of the SDK from the server

onTryToReconnect	Callback of the SDK trying to connect to the server again
onConnectionRecovery	Callback of the reconnection of the SDK to the server
onSpeedTest	Callback of server speed test results. The SDK tests the speed of multiple server addresses, and the result of each test is returned through this callback.

Hardware event callback APIs

API	Description
onCameraDidReady	Callback of the camera being ready
onMicDidReady	Callback of the mic being ready
onUserVoiceVolume	Callback of volume, including the volume of each `userId` and the total remote volume
onDeviceChange	Callback of connecting/disconnecting a local device

Custom message receiving callback APIs

API	Description
onRecvSEIMsg	Callback of receiving an SEI message

Callback APIs for CDN relayed push

API	Description
onStartPublishing	Callback of starting to push to Tencent Cloud's live streaming CDN, which corresponds to the `startPublishing()` API in TRTCCloud
onStopPublishing	Callback of stopping pushing to Tencent Cloud's live streaming CDN, which corresponds to the `stopPublishing()` API in TRTCCloud
onStartPublishCDNStream	Callback of the completion of starting relayed push to CDNs
onStopPublishCDNStream	Callback of the completion of stopping relayed push to CDNs
onSetMixTranscodingConfig	Sets On-Cloud MixTranscoding parameters, which corresponds to the `setMixTranscodingConfig()` API in `TRTCCloud`

Definitions of Key Classes

Class	Description
TRTCParams	Room entry parameters
TRTCVideoEncParam	Video encoding parameters
TRTCTranscodingConfig	On-Cloud MixTranscoding configuration
TRTCSwitchRoomConfig	Room switching parameters
TRTCNetworkQosParam	QoS control parameters
TXVoiceReverbType	Reverb effects (karaoke, room, hall, low and deep, resonant, etc.)
AudioMusicParam	Parameters for music and voice effect setting APIs
TRTCAudioRecordingParams	Audio recording parameters

Specific Device Management APIs

API	Description
isFrontCamera	Gets whether the front camera is being used.
switchCamera	Switches cameras.
getCameraZoomMaxRatio	Gets the maximum zoom level of the current camera.
setCameraZoomRatio	Sets the zoom level of the current camera.
isAutoFocusEnabled	Gets whether automatic facial recognition is supported.
enableCameraAutoFocus	Enables/Disables automatic facial recognition.
setCameraFocusPosition	Sets camera focus.

enableCameraTorch	Enables/Disables flash.
setSystemVolumeType	Sets the system volume type to use during calls.
setAudioRoute	Sets the audio route.

Specific Music and Voice Effect APIs

API	Description
setVoiceReverbType	Sets the voice change effects (karaoke, room, hall, low and deep, resonant, etc.)
setMusicObserver	Sets the callback of the playback progress of background music.
startPlayMusic	Starts playing background music.
stopPlayMusic	Stops playing background music.
pausePlayMusic	Pauses background music.
resumePlayMusic	Resumes playing background music.
setMusicPublishVolume	Sets the remote playback volume of background music, i.e., the volume heard by remote users.
setMusicPlayoutVolume	Sets the local playback volume of background music.
setAllMusicVolume	Sets the local and remote playback volume of background music.
setMusicPitch	Changes the pitch of background music.
setMusicSpeedRate	Changes the playback speed of background music.
getMusicCurrentPosInMS	Gets the playback progress (ms) of background music.
seekMusicToPosInMS	Sets the playback progress (ms) of background music.
getMusicDurationInMS	Gets the length (ms) of the background music file.

Error Codes

Last updated：2023-10-09 11:55:48

Error Codes

Basic error codes

Code	Value	Description
ERR_NULL	0	No error.

Error codes for room entry

"TRTCCloud.enterRoom()" will trigger this type of error code if room entry fails. You can use the callback functions "TRTCCloudDelegate.onEnterRoom()" and "TRTCCloudDelegate.OnError()" to capture related notifications.

Code	Value	Description
ERR_ROOM_ENTER_FAIL	-3301	Failed to enter room.
ERR_ENTER_ROOM_PARAM_NULL	-3316	Empty room entry parameters. Please check whether valid parameters are passed in the `TRTCCloud.enterRoom():` API when it is called.
ERR_SDK_APPID_INVALID	-3317	Invalid `sdkAppId` .
ERR_ROOM_ID_INVALID	-3318	Invalid `roomId` .
ERR_USER_ID_INVALID	-3319	Invalid `userID` .
ERR_USER_SIG_INVALID	-3320	Invalid `userSig` .
ERR_ROOM_REQUEST_ENTER_ROOM_TIMEOUT	-3308	Room entry request timed out. Please check your network.
ERR_SERVER_INFO_SERVICE_SUSPENDED	-100013	Service unavailable. Please check whether there are remaining minutes in your packages and whether your Tencent Cloud account has overdue payment.

Error code for room exit

`TRTCCloud.exitRoom()` triggers this error code if room exit fails. You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_ROOM_REQUEST_QUIT_ROOM_TIMEOUT	-3325	Room exit request timed out.

Error codes for devices (camera, mic, and speaker)

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_CAMERA_START_FAIL	-1301	Failed to turn camera on. This error occurs when, for example, there is a problem with the camera configuration program (driver) on Windows or macOS. In this case, turn the camera off and on again, restart the camera, or update the configuration program.
ERR_CAMERA_NOT_AUTHORIZED	-1314	Camera not authorized. This error usually occurs on mobile devices and may be because users denied camera permission.
ERR_CAMERA_SET_PARAM_FAIL	-1315	Failed to set camera parameters (unsupported values or others).
ERR_CAMERA_OCCUPY	-1316	Camera occupied. Try using another camera.
ERR_MIC_START_FAIL	-1302	Failed to turn mic on. This error occurs when, for example, there is a problem with the mic configuration program (driver) on Windows or macOS. In this case, turn the mic off and on again, restart the mic, or update the configuration program.
ERR_MIC_NOT_AUTHORIZED	-1317	Mic not authorized. This error usually occurs on mobile devices and may be because users denied mic permission.
ERR_MIC_SET_PARAM_FAIL	-1318	Failed to set mic parameters.
ERR_MIC_OCCUPY	-1319	Mic occupied. This error occurs when, for example, the user is in a call on the mobile device, in which case TRTC will fail to turn the mic on.
ERR_MIC_STOP_FAIL	-1320	Failed to turn mic off.
ERR_SPEAKER_START_FAIL	-1321	Failed to turn speaker on. This error occurs when, for

		example, there is a problem with the speaker configuration program (driver) on Windows or macOS. In this case, turn the speaker off and on again, restart the speaker, or update the configuration program.
ERR_SPEAKER_SET_PARAM_FAIL	-1322	Failed to set speaker parameters.
ERR_SPEAKER_STOP_FAIL	-1323	Failed to turn speaker off.

Error codes for screen sharing

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_SCREEN_CAPTURE_START_FAIL	-1308	Failed to start screen recording. If this error occurs on a mobile device, it may be because users denied screen recording permission; if it occurs on Windows or macOS, check whether the parameters of the screen recording API are set as required.
ERR_SCREEN_CAPTURE_UNSURPORT	-1309	Screen recording failed. If you use Android, make sure its version is 5.0 or later; if you use iOS, make sure its version is 11.0 or later.
ERR_SERVER_CENTER_NO_PRIVILEDGE_PUSH_SUB_VIDEO	-102015	No permission to send substream video images.
ERR_SERVER_CENTER_ANOTHER_USER_PUSH_SUB_VIDEO	-102016	Another user is sending substream video images.
ERR_SCREEN_CAPTURE_STOPPED	-7001	Screen recording stopped by the system.

Error codes for encoding and decoding

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description

ERR_VIDEO_ENCODE_FAIL	-1303	Failed to encode video frames. This error occurs when, for example, a user on iOS switches to another app, which may cause the system to release the hardware encoder. When the user switches back, this error may be thrown before the hardware encoder is restarted.
PUSH_ERR_UNSUPPORTED_RESOLUTION	-1305	Unsupported video resolution.
ERR_AUDIO_ENCODE_FAIL	-1304	Failed to encode audio frames. This error occurs when, for example, the SDK could not process the custom audio data passed in.
PUSH_ERR_UNSUPPORTED_SAMPLERATE	-1306	Unsupported audio sample rate.

Error codes for custom capturing

You can use the callback function `TRTCCloudDelegate.OnError()` to capture related notifications.

Code	Value	Description
ERR_PIXEL_FORMAT_UNSUPPORTED	-1327	Unsupported pixel format.
ERR_BUFFER_TYPE_UNSUPPORTED	-1328	Unsupported buffer type.

Error codes for CDN binding and stream mixing

You can use the callback functions `TRTCCloudDelegate.onStartPublishing()` and `TRTCCloudDelegate.onSetMixTranscodingConfig()` to capture related notifications.

Code	Value	Description
ERR_PUBLISH_CDN_STREAM_REQUEST_TIME_OUT	-3321	Relay-to-CDN request timed out.
ERR_CLOUD_MIX_TRANSCODING_REQUEST_TIME_OUT	-3322	On-Cloud MixTranscoding request timed out.
ERR_PUBLISH_CDN_STREAM_SERVER_FAILED	-3323	Abnormal response packets for relay.
ERR_CLOUD_MIX_TRANSCODING_SERVER_FAILED	-3324	Abnormal response packets for On-Cloud MixTranscoding.
ERR_ROOM_REQUEST_START_PUBLISHING_TIMEOUT	-3333	Signaling of starting to push to Tencent Cloud's live streaming CDN timed out.

ERR_ROOM_REQUEST_START_PUBLISHING_ERROR	-3334	Abnormal signaling of starting to push to Tencent Cloud's live streaming CDN.
ERR_ROOM_REQUEST_STOP_PUBLISHING_TIMEOUT	-3335	Signaling of stopping pushing to Tencent Cloud's live streaming CDN timed out.
ERR_ROOM_REQUEST_STOP_PUBLISHING_ERROR	-3336	Abnormal signaling of stopping pushing to Tencent Cloud's live streaming CDN.

Error codes for cross-room co-anchoring

"TRTCCloud.ConnectOtherRoom()" will trigger this type of error code if cross-room co-anchoring fails. You can use the callback function "TRTCCloudDelegate.onConnectOtherRoom()" to capture related notifications.

Code	Value	Description
ERR_ROOM_REQUEST_CONN_ROOM_TIMEOUT	-3326	Co-anchoring request timed out.
ERR_ROOM_REQUEST_DISCONN_ROOM_TIMEOUT	-3327	Request to exit co-anchoring timed out.
ERR_ROOM_REQUEST_CONN_ROOM_INVALID_PARAM	-3328	Invalid parameter.
ERR_CONNECT_OTHER_ROOM_AS_AUDIENCE	-3330	You are in the role of audience and cannot initiate or end co-anchoring. You need to switch to the anchor role using `switchRole()`.
ERR_SERVER_CENTER_CONN_ROOM_NOT_SUPPORT	-102031	Cross-room co-anchoring not supported.
ERR_SERVER_CENTER_CONN_ROOM_REACH_MAX_NUM	-102032	Reached the upper limit of co-anchoring calls.
ERR_SERVER_CENTER_CONN_ROOM_REACH_MAX_RETRY_TIMES	-102033	Reached the upper limit of retries for

		cross-room co-anchoring.
ERR_SERVER_CENTER_CONN_ROOM_REQ_TIMEOUT	-102034	Cross-room co-anchoring request timed out.
ERR_SERVER_CENTER_CONN_ROOM_REQ	-102035	Incorrect format of cross-room co-anchoring request.
ERR_SERVER_CENTER_CONN_ROOM_NO_SIG	-102036	No signature for cross-room co-anchoring.
ERR_SERVER_CENTER_CONN_ROOM_DECRYPT_SIG	-102037	Failed to decrypt signature for cross-room co-anchoring.
ERR_SERVER_CENTER_CONN_ROOM_NO_KEY	-102038	Decryption key for cross-room co-anchoring signature not found.
ERR_SERVER_CENTER_CONN_ROOM_PARSE_SIG	-102039	Signature parsing error for cross-room co-anchoring.
ERR_SERVER_CENTER_CONN_ROOM_INVALID_SIG_TIME	-102040	Incorrect timestamp of cross-room co-anchoring signature.
ERR_SERVER_CENTER_CONN_ROOM_SIG_GROUPID	-102041	Mismatch of room ID in cross-room co-anchoring signature.
ERR_SERVER_CENTER_CONN_ROOM_NOT_CONNED	-102042	Mismatch of username in cross-room co-anchoring signature.
ERR_SERVER_CENTER_CONN_ROOM_USER_NOT_CONNED	-102043	The user did not initiate co-anchoring.

ERR_SERVER_CENTER_CONN_ROOM_FAILED	-102044	Failed to start cross-room co-anchoring.
ERR_SERVER_CENTER_CONN_ROOM_CANCEL_FAILED	-102045	Failed to cancel cross-room co-anchoring.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_ROOM_NOT_EXIST	-102046	The room being connected for co-anchoring does not exist.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_REACH_MAX_ROOM	-102047	The room being connected reached the upper limit of co-anchoring calls.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_USER_NOT_EXIST	-102048	The user being called for co-anchoring does not exist.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_USER_DELETED	-102049	The user being called for co-anchoring was deleted.
ERR_SERVER_CENTER_CONN_ROOM_CONNED_USER_FULL	-102050	All resources of the user being called for co-anchoring are occupied.
ERR_SERVER_CENTER_CONN_ROOM_INVALID_SEQ	-102051	Sequence number for co-anchoring not in sequential order.

Warning Codes

Warning codes do not require your special attention. You can choose whether to prompt the user depending on the situation.

Code	Value	Description
WARNING_HW_ENCODER_START_FAIL	1103	Failed to start hardware encoder.

		The SDK automatically switched to software encoder.
WARNING_VIDEO_ENCODER_SW_TO_HW	1107	Insufficient CPU for software encoder. The SDK automatically switched to hardware encoder.
WARNING_INSUFFICIENT_CAPTURE_FPS	1108	Insufficient frame rate of video captured by camera. This error may occur on Android devices with built-in beauty filter algorithms.
WARNING_SW_ENCODER_START_FAIL	1109	Failed to start software encoder.
WARNING_REDUCE_CAPTURE_RESOLUTION	1110	Camera resolution reduced for balance between frame rate and performance.
WARNING_CAMERA_DEVICE_EMPTY	1111	No available camera found.
WARNING_CAMERA_NOT_AUTHORIZED	1112	User did not grant the application camera access.
WARNING_MICROPHONE_DEVICE_EMPTY	1201	No available mic found.
WARNING_SPEAKER_DEVICE_EMPTY	1202	No available speaker found.
WARNING_MICROPHONE_NOT_AUTHORIZED	1203	User did not grant the application mic access.
WARNING_MICROPHONE_DEVICE_ABNORMAL	1204	No audio capturing device available (for example, because the device is occupied).
WARNING_SPEAKER_DEVICE_ABNORMAL	1205	No audio playback device available (for example, because the device is occupied).
WARNING_VIDEO_FRAME_DECODE_FAIL	2101	Failed to decode current video frame.
WARNING_AUDIO_FRAME_DECODE_FAIL	2102	Failed to decode current audio frame.
WARNING_VIDEO_PLAY_LAG	2105	Video playback stuttering.
WARNING_HW_DECODER_START_FAIL	2106	Failed to start hardware decoder. Software decoder is used instead.

WARNING_VIDEO_DECODER_HW_TO_SW	2108	Hardware decoder failed to decode first I-frame of current stream. The SDK automatically switched to software decoder.
WARNING_SW_DECODER_START_FAIL	2109	Failed to start software decoder.
WARNING_VIDEO_RENDER_FAIL	2110	Failed to render video.
WARNING_START_CAPTURE_IGNORED	4000	Video capturing already started. Request ignored.
WARNING_AUDIO_RECORDING_WRITE_FAIL	7001	Failed to write recorded audio to file.
WARNING_ROOM_DISCONNECT	5101	Network disconnected.
WARNING_IGNORE_UPSTREAM_FOR_AUDIENCE	6001	You are in the role of audience. The request to send audio/video data is ignored.
WARNING_NET_BUSY	1101	Bad network connection: data upload blocked due to limited upstream bandwidth.
WARNING_RTMP_SERVER_RECONNECT	1102	Push error. The network is disconnected. Reconnecting... (max attempts: 3).
WARNING_LIVE_STREAM_SERVER_RECONNECT	2103	Pull error. The network is disconnected. Reconnecting... (max attempts: 3).
WARNING_RECV_DATA_LAG	2104	Unstable incoming packets. This may be caused by insufficient downstream bandwidth or unstable streams from the anchor.
WARNING_RTMP_DNS_FAIL	3001	Live streaming error. DNS resolution failed.
WARNING_RTMP_SEVER_CONN_FAIL	3002	Live streaming error. Failed to connect to server.
WARNING_RTMP_SHAKE_FAIL	3003	Live streaming error. Handshake with RTMP server failed.
WARNING_RTMP_SERVER_BREAK_CONNECT	3004	Live streaming error. Connection dropped by server.

WARNING_RTMP_READ_WRITE_FAIL	3005	Live streaming error. RTMP read/write failed. Disconnecting.
WARNING_RTMP_WRITE_FAIL	3006	Live streaming error. RTMP write failed. This is an internal error code of the SDK and is not thrown.
WARNING_RTMP_READ_FAIL	3007	Live streaming error. RTMP read failed. This is an internal error code of the SDK and is not thrown.
WARNING_RTMP_NO_DATA	3008	Live streaming error. Server disconnected as no data is sent for over 30 seconds.
WARNING_PLAY_LIVE_STREAM_INFO_CONNECT_FAIL	3009	Live streaming error. Failed to call `connect` to connect to server. This is an internal error code of the SDK and is not thrown.
WARNING_NO_STEAM_SOURCE_FAIL	3010	Live streaming error. Connection failed as there was no video in the stream address. This is an internal error code of the SDK and is not thrown.
WARNING_ROOM_RECONNECT	5102	Network disconnected. Reconnecting...
WARNING_ROOM_NET_BUSY	5103	Bad network connection: data upload blocked due to limited upstream bandwidth.