Tencent Real-Time Communication

Advanced Features

Product Documentation

Contents

Advanced Features

Relay to CDN

Last updated：2025-06-09 16:21:45

This document describes how to publish (relay) audio/video streams in TRTC to CDNs so that viewers can watch the streams using standard live streaming players.

Applicable Scenario

Since TRTC uses User Datagram Protocol (UDP) to transmit audio and video data, while Live Video Broadcasting (LVB) CDN uses Real-Time Messaging Protocol (RTMP), HTTP Live Streaming (HLS), Flash Video (FLV), and other protocols for data transmission, it is necessary to **relay** TRTC's audio and video data to the live streaming CDN for viewers to watch through the CDN.
Integrating TRTC with CDN for viewing is typically used to address the following issues:
Viewing with ultra-high concurrency
TRTC's low-latency viewing capability supports up to 100,000 participants in a single room. Although CDN viewing has higher latency, it supports more than 100,000 concurrent viewers and offers more affordable prices.

Relay to CDN Control Solution

TRTC provides several control solutions for publishing audio and video streams to a live streaming CDN (that is, relay to CDN), which include initiating relay using terminal SDK, initiating relay using RESTful APIs, and automatic relay.
The specific solutions are described as follows:

Solution 1: Initiating Relay Using Terminal SDK

Step 1: Publishing the Local User's Stream to CDNs

Feature Description

You can use the startPublishMediaStream (for example, IOS) API of TRTCCloud to publish the audio/video streams of local users to live streaming CDNs (known in TRTC as "relay to CDN").
The TRTC server will send the audio/video data directly to the CDN server. Because the data is not transcoded, the

cost is relatively low.

However, if there are multiple users publishing audio/video streams in a room, there will be a CDN stream for each user. Multiple players are needed to play the streams, and they may not play in sync.

Directions

Follow the steps below to publish the local user's stream to CDNs.

1. Create a `TRTCPublishTarget` object and set `mode` in the object to `TRTCPublishBigStreamToCdn` or `TRTCPublishSubStreamToCdn` . The former is used to publish the user's primary stream (usually the camera), and the latter is used to publish the user's substream (usually the screen).
2. Set `cdnUrlList` in the `TRTCPublishTarget` object to one or multiple CDN addresses (which usually starts with `rtmp://`). If you publish to the Tencent Cloud CDN (can be generated in CSS console > Address Generator), set `isInternalLine` to `true` ; otherwise, set it to `false` .
3. Because the data is not transcoded, leave `TRTCStreamEncoderParam` and `TRTCStreamMixingConfig` empty.
4. Call `startPublishMediaStream` . If the `taskId` parameter returned by the `onStartPublishMediaStream` callback is not empty, the local API call is successful.
5. To stop publishing, call `stopPublishMediaStream` , passing in the `taskId` returned by `onStartPublishMediaStream` .

Sample Code

The code below publishes the local user's stream to a live streaming CDN.

java

Objective-C

C++

Web

Dart

```
// Publish the local user's stream to a live streaming CDN.
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
target.mode = TRTC_PublishBigStream_ToCdn;
TRTCCloudDef.TRTCPublishCdnUrl cdnUrl= new TRTCCloudDef.TRTCPublishCdnUrl();
cdnUrl.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = true;
target.cdnUrlList.add(cdnUrl);
mTRTCCloud.startPublishMediaStream(target, null, null);
```

```
// Publish the local user's stream to a live streaming CDN.
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishBigStreamToCdn;
TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
cdnUrl.rtmpUrl = @"rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = YES;
NSMutableArray* cdnUrlList = [NSMutableArray new];
[cdnUrlList addObject:cdnUrl];
target.cdnUrlList = cdnUrlList;
[_trtcCloud startPublishMediaStream:target encoderParam:nil mixingConfig:nil];
```

```
// Publish the local user's stream to a live streaming CDN.
TRTCPublishTarget target;
target.mode = TRTCPublishMode::TRTCPublishBigStreamToCdn;
TRTCPublishCdnUrl* cdn_url_list = new TRTCPublishCdnUrl[1];
cdn_url_list[0].rtmpUrl = "rtmp://tencent/live/bestnews";
cdn_url_list[0].isInternalLine = true;
target.cdnUrlList = cdn_url_list;
target.cdnUrlListSize = 1;
trtc->startPublishMediaStream(&target, nullptr, nullptr);
delete[] cdn_url_list;
```

```
const options = {
  target: {
    publishMode: PublishMode.PublishMainStreamToCDN
  }
}
try {
  await trtc.startPlugin('CDNStreaming', options);
} catch (error) {
  console.error('CDNStreaming start failed', error);
}
```

```
TRTCPublishTarget target = TRTCPublishTarget();
target.mode = TRTCPublishMode.TRTCPublishBigStreamToCdn;
TRTCPublishCdnUrl cdnUrlEntity = new TRTCPublishCdnUrl();
cdnUrlEntity.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrlEntity.isInternalLine = true;
target.cdnUrlList.add(cdnUrlEntity);

trtcCloud.startPublishMediaStream(target: target);
```

Note:

Web class names differ slightly, but the usage is consistent. For detailed information, refer to CDNStreaming Plugin.

For relay in the Web 4.x version, refer to Client.startMixTranscode().

Step 2: Publishing Mixed Streams to CDNs

Feature Description

You can call **startPublishMediaStream** to mix the streams of multiple users in a TRTC room into one stream and publish the stream to a CDN. The `TRTCStreamEncoderParam` and `TRTCStreamMixingConfig` parameters of the API allow you to determine the details of stream mixing and transcoding.
The streams will be decoded on the cloud first, mixed, and then re-encoded according to the stream mixing parameters (`TRTCStreamMixingConfig`) and transcoding parameters (`TRTCStreamEncoderParam`) you specify. Afterward, they will be published to CDNs. In this mode, additional transcoding fees are charged.

Directions

Follow the steps below to mix the streams of multiple users in a room and publish the mixed stream to CDNs.
1. Create a `TRTCPublishTarget` object and set `mode` in the object to `TRTCPublishMixStreamToCdn` .
2. Set `cdnUrlList` in the `TRTCPublishTarget` object to one or multiple CDN addresses (which usually start with `rtmp://`). If you publish to the Tencent Cloud CDN, set `isInternalLine` to `true` ; otherwise, set it to `false` .
3. Set the encoding parameters (`TRTCStreamEncoderParam`):
Video encoding parameters: Specify the resolution, frame rate (15 fps is recommended), bitrate, and GOP (3 seconds is recommended). Bitrate and resolution work in correlation with each other. The table below lists some recommended resolution and bitrate settings.

videoEncodedWidth	videoEncodedHeight	videoEncodedFPS	videoEncodedGOP	videoEncodedKbp
640	360	15	3	800 Kbps
960	540	15	3	1200 Kbps
1280	720	15	3	1500 Kbps
1920	1080	15	3	2500 Kbps

Audio encoding parameters: Specify the codec, bitrate, sample rate, and sound channels according to the `AudioQuality` value you pass in when calling `startLocalAudio` .

TRTCAudioQuality	audioEncodedSampleRate	audioEncodedChannelNum	audioEncodedKbps
TRTCAudioQualitySpeech	48000	1	50
TRTCAudioQualityDefault	48000	1	50
TRTCAudioQualityMusic	48000	2	60

4. Set the parameters for audio mixing and video layout (TRTCStreamMixingConfig):
Audio mixing parameters (audioMixUserList): You can leave this parameter empty to mix all audios in a room, or you can set it to the IDs of users whose audios you want to mix.
Video layout parameters (videoLayoutList): Video layout is determined by an array. Each TRTCVideoLayout element in the array determines the position, dimensions, and background color of a video window. If you specify fixedVideoUser, the window defined by the TRTCVideoLayout element will display the video of a specific user. If you set fixedVideoUser to null, the TRTC server will determine whose video to display in the window.
Example:
Example 1: Mix four users' streams and use an image as the background.
`layout1` specifies the position (upper half of the canvas) and dimensions (640 x 480) of the camera video of user `jerry` .
Because no user IDs are specified for `layout2` , `layout3` , and `layout4` , TRTC will display the videos of the other three users in the windows based on its own rule.

Example 2: Mix the camera video and screen of one user plus the camera videos of three other users.
`layout1` specifies the position (left) and dimensions (1280 x 720) of user `jerry` 's screen. The rendering mode used is aspect fit (`Fit`), and the background color is black.
`layout2` specifies the position (top right) and dimensions (300 x 200) of user `jerry` 's camera video. The rendering mode used is aspect fill (`Fill`).
Because no user IDs are specified for `layout3` , `layout4` , and `layout5` , TRTC will display the videos of the other three users in the windows based on its own rule.

5. Call startPublishMediaStream. If the taskId parameter returned by the onStartPublishMediaStream callback is not empty, the local API call is successful.
6. To change the stream mixing parameters (for example, the video layout), call `updatePublishMediaStream` , passing in the `taskId` returned in step 6 as well as the new `TRTCStreamMixingConfig` parameters. We recommend you do not change `TRTCStreamEncoderParam` during relay because doing so will affect the stability of CDN playback.
7. To stop publishing, call `stopPublishMediaStream` , passing in the `taskId` returned by `onStartPublishMediaStream` .

Sample Code

The code below mixes the streams of multiple users in a room and publishes the result to a CDN.
java
Objective-C

C++

Dart

```
// Specify the publishing mode as TRTC_PublishMixedStream_ToCdn.
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
target.mode = TRTC_PublishMixedStream_ToCdn;
// Specify the CDN address for publishing.
TRTCCloudDef.TRTCPublishCdnUrl cdnUrl= new TRTCCloudDef.TRTCPublishCdnUrl();
cdnUrl.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = true;
target.cdnUrlList.add(cdnUrl);
```

```
// Specify the publishing mode as TRTCPublishMixStreamToCdn.
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishMixStreamToCdn;
// Specify the CDN address for publishing.
TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
cdnUrl.rtmpUrl = @"rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = YES;
NSMutableArray* cdnUrlList = [NSMutableArray new];
[cdnUrlList addObject:cdnUrl];
target.cdnUrlList = cdnUrlList;
// Set the secondary encoding parameters for the mixed audio and video streams.
TRTCStreamEncoderParam* encoderParam = [[TRTCStreamEncoderParam alloc] init];
encoderParam.videoEncodedWidth = 1280;
encoderParam.videoEncodedHeight = 720;
encoderParam.videoEncodedFPS = 15;
encoderParam.videoEncodedGOP = 3;
encoderParam.videoEncodedKbps = 1000;
encoderParam.audioEncodedSampleRate = 48000;
encoderParam.audioEncodedChannelNum = 1;
encoderParam.audioEncodedKbps = 50;
encoderParam.audioEncodedCodecType = 0;
// Set the layout parameters for the screen.
TRTCStreamMixingConfig* config = [[TRTCStreamMixingConfig alloc] init];
NSMutableArray* videoLayoutList = [NSMutableArray new];
TRTCVideoLayout* layout1 = [[TRTCVideoLayout alloc] init];
layout1.zOrder = 0;
layout1.rect = CGRectMake(0, 0, 720, 1280);
layout1.fixedVideoStreamType = TRTCVideoStreamTypeSub;
layout1.fixedVideoUser.intRoomId = 1234;
layout1.fixedVideoUser.userId = @"mike";
TRTCVideoLayout* layout2 = [[TRTCVideoLayout alloc] init];
layout2.zOrder = 0;
layout2.rect = CGRectMake(1300, 0, 300, 200);
layout2.fixedVideoStreamType = TRTCVideoStreamTypeBig;
```

```
layout2.fixedVideoUser.intRoomId = 1234;
layout2.fixedVideoUser.userId = @"mike";
TRTCVideoLayout* layout3 = [[TRTCVideoLayout alloc] init];
layout3.zOrder = 0;
layout3.rect = CGRectMake(1300, 220, 300, 200);
layout3.fixedVideoStreamType = TRTCVideoStreamTypeSub;
layout3.fixedVideoUser = nil;
[videoLayoutList addObject:layout1];
[videoLayoutList addObject:layout2];
[videoLayoutList addObject:layout3];
config.videoLayoutList = videoLayoutList;
config.audioMixUserList = nil;
// Initiate stream mixing.
[_trtcCloud startPublishMediaStream:target encoderParam:encoderParam mixingConfig:c
```

```
// Specify the publishing mode as TRTCPublishMixStreamToCdn.
TRTCPublishTarget target;
target.mode = TRTCPublishMode::TRTCPublishMixStreamToCdn;
// Specify the CDN address for publishing.
TRTCPublishCdnUrl* cdn_url = new TRTCPublishCdnUrl[1];
cdn_url[0].rtmpUrl = "rtmp://tencent/live/bestnews";
cdn_url[0].isInternalLine = true;
target.cdnUrlList = cdn_url;
target.cdnUrlListSize = 1;
// Set the secondary encoding parameters for the mixed audio and video streams.
TRTCStreamEncoderParam encoder_param;
encoder_param.videoEncodedWidth = 1280;
encoder_param.videoEncodedHeight = 720;
encoder_param.videoEncodedFPS = 15;
encoder_param.videoEncodedGOP = 3;
encoder_param.videoEncodedKbps = 1000;
encoder_param.audioEncodedSampleRate = 48000;
encoder_param.audioEncodedChannelNum = 1;
encoder_param.audioEncodedKbps = 50;
encoder_param.audioEncodedCodecType = 0;
// Set the layout parameters for the screen.
TRTCStreamMixingConfig config;
TRTCVideoLayout* video_layout_list = new TRTCVideoLayout[3];
TRTCUser* fixedVideoUser0 = new TRTCUser();
fixedVideoUser0->intRoomId = 1234;
fixedVideoUser0->userId = "mike";
video_layout_list[0].zOrder = 0;
video_layout_list[0].rect.left = 0;
video_layout_list[0].rect.top = 0;
video_layout_list[0].rect.right = 720;
video_layout_list[0].rect.bottom = 1280;
```

```
video_layout_list[0].fixedVideoStreamType =
     TRTCVideoStreamType::TRTCVideoStreamTypeSub;
video_layout_list[0].fixedVideoUser = fixedVideoUser0;
TRTCUser* fixedVideoUser1 = new TRTCUser();
fixedVideoUser1->intRoomId = 1234;
fixedVideoUser1->userId = "mike";
video_layout_list[1].zOrder = 0;
video_layout_list[1].rect.left = 1300;
video_layout_list[1].rect.top = 0;
video_layout_list[1].rect.right = 300;
video_layout_list[1].rect.bottom = 200;
video_layout_list[1].fixedVideoStreamType =
     TRTCVideoStreamType::TRTCVideoStreamTypeBig;
video_layout_list[1].fixedVideoUser = fixedVideoUser1;
video_layout_list[2].zOrder = 0;
video_layout_list[2].rect.left = 1300;
video_layout_list[2].rect.top = 220;
video_layout_list[2].rect.right = 300;
video_layout_list[2].rect.bottom = 200;
video_layout_list[2].fixedVideoStreamType =
     TRTCVideoStreamType::TRTCVideoStreamTypeSub;
video_layout_list[2].fixedVideoUser = nullptr;
config.videoLayoutList = video_layout_list;
config.videoLayoutListSize = 3;
config.audioMixUserList = nullptr;
// Initiate stream mixing.
trtc->startPublishMediaStream(&target, &encoder_param, &config);
delete fixedVideoUser0;
delete fixedVideoUser1;
delete[] video_layout_list;
```

```
TRTCPublishTarget target = TRTCPublishTarget();
target.mode = TRTCPublishMode.TRTCPublishMixStreamToCdn;
TRTCPublishCdnUrl cdnUrlEntity = new TRTCPublishCdnUrl();
cdnUrlEntity.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrlEntity.isInternalLine = true;
target.cdnUrlList.add(cdnUrlEntity);

TRTCStreamMixingConfig config = TRTCStreamMixingConfig();
TRTCUser selfUser = TRTCUser();
selfUser.userId = localUserId;
selfUser.intRoomId = localRoomId;

TRTCVideoLayout selfVideoLayout = TRTCVideoLayout();
selfVideoLayout.fixedVideoStreamType = TRTCVideoStreamType.TRTCVideoStreamTypeBig;
selfVideoLayout.rect = Rect(originX: 0, originY: 0, sizeWidth: 1080, sizeHeight: 19
selfVideoLayout.zOrder = 0;
selfVideoLayout.fixedVideoUser = selfUser;
selfVideoLayout.fillMode = TRTCVideoFillMode.TRTCVideoFillMode_Fit;
config.videoLayoutList.add(selfVideoLayout);TRTCUser remoteUser = TRTCUser();

remoteUser.userId = remoteUserId;
remoteUser.intRoomId = remoteRoomId;
TRTCVideoLayout remoteVideoLayout = TRTCVideoLayout();
remoteVideoLayout.fixedVideoStreamType = TRTCVideoStreamType.TRTCVideoStreamTypeBig
remoteVideoLayout.rect = Rect(originX: 100, originY: 50, sizeWidth: 216, sizeHeight
remoteVideoLayout.zOrder = 1;
remoteVideoLayout.fixedVideoUser = remoteUser;
remoteVideoLayout.fillMode = TRTCVideoFillMode.TRTCVideoFillMode_Fit;
config.videoLayoutList.add(remoteVideoLayout);

TRTCStreamEncoderParam param = TRTCStreamEncoderParam();
param.videoEncodedWidth = 1080;
param.videoEncodedHeight = 1920;
param.videoEncodedKbps = 5000;
param.videoEncodedFPS = 30;
param.videoEncodedGOP = 3;
param.audioEncodedSampleRate = 48000;
param.audioEncodedChannelNum = 2;
param.audioEncodedKbps = 128;
param.audioEncodedCodecType = 2;

trtcCloud.startPublishMediaStream(target: target, config: config, params: param);
```

Solution 2: Initiating Relay Using RESTful APIs

The following describes how to use RESTful APIs to publish (relay) audio and video streams from a TRTC room to a live streaming CDN or push them back to the TRTC room, so that viewers can watch the streams using standard live streaming players.

Supported Features

When a relay task is initiated using RESTful APIs, the following features can be achieved:

Relay a single audio and video stream to both the live streaming CDN and TRTC room.

Mix multiple audio and video streams into a new single stream and relay it to both the live streaming CDN and TRTC room.

Support outputting audio only and both audio and video.

Support custom layouts and dynamic templates.

Support setting background images, placeholder images, and watermark images.

Support cropping and scaling input videos and images.

Support adding supplemental enhancement information (SEI) to mixed audio and video streams.

How It Works

Cloud-based stream mixing involves six processes: entering a room, pulling streams, decoding, mixing, encoding, and relay:

Entering a room: Utilizing the specified bot information, the microcontroller unit (MCU) creates a companion bot instance to enter the room.

Pulling streams: Based on the specified mixing layout parameters, the MCU bot pulls the relevant users' audio and video streams.

Decoding: The MCU decodes multiple audio and video streams, including video decoding and audio decoding.

Mixing: The MCU combines multiple screens based on the specified mixing layout parameters. Simultaneously, the MCU also performs audio mixing on the decoded multi-channel audio signals.

Encoding: The MCU re-encodes the mixed videos and audios according to your configured output encoding parameters, packaging them into a single audio and video stream.

Relay: The MCU distributes the encoded and packaged audio and video data to your configured live streaming CDN.

Initiating a Relay Task

Your server can initiate a cloud-based relay task by calling the RESTful API StartPublishCdnStream. The method to initiate it is as follows:

1. **Set basic parameters (required).**

You need to specify the basic information to initiate the relay task, such as your application ID (sdkappid), main room ID (RoomId), main room type (RoomIdType), and whether to transcode (WithTranscoding). You can determine whether to transcode by setting WithTranscoding. If WithTranscoding is set to `true` , it will be a mixed-stream relay. If set to `false` , it will be a relay to CDN.

Field Name	Description	Required
SdkAppId	TRTC's SdkAppId.	Yes
RoomId	Main room ID.	Yes
RoomIdType	Main room type.	Yes
WithTranscoding	Whether to transcode.	Yes

2. **Set the bot parameters (required).**

You need to specify the AgentParams parameters for a bot to enter a room. The MCU will create an instance to enter the room based on the parameters you specify.

Field Name	Description	Required
AgentParams.UserId	The UserId used by the relay service in a TRTC room. Do not use the same UserId as those used by normal users in the room.	Yes
AgentParams.UserSig	The user signature for the relay service to enter a TRTC room.	Yes
AgentParams.MaxIdleTime	Idle wait time.	No

3. **Set audio parameters (required for mixed audio stream output, not required for single-stream relay).**
To output mixed audio streams, you need to specify the AudioParams parameters. The MCU will output the audio streams in the format you configure in AudioEncode. You can configure SubscribeAudioList to specify which users' audio streams to be mixed.

Field Name	Description	Required
AudioParams.AudioEncode	Audio output encoding parameter for stream mixing.	No
AudioParams.SubscribeAudioList	Audio user allowlist for stream mixing.	No

For detailed meanings, refer to the parameter description in McuAudioParams.

4. **Set video parameters (required for mixed video stream output, not required for single-stream relay).**
To output mixed video streams, you need to specify the VideoParams parameters. The MCU will output the video streams in the format you configure in VideoEncode.
Configure LayoutParams to specify the layout you need.
Configure BackGroundColor to specify the canvas background color you need.
Configure BackgroundImageUrl to specify the canvas background image you need.
Configure WaterMarkList to specify the watermark layout you need.

Field Name	Description	Required
VideoParams.VideoEncode	Video output encoding parameter for stream mixing.	No
VideoParams.LayoutParams	Layout parameters for stream mixing.	No
VideoParams.BackGroundColor	Canvas background color for stream mixing.	No
VideoParams.BackgroundImageUrl	Canvas background image URL for stream mixing.	No
VideoParams.WaterMarkList	Watermark parameter for stream mixing.	No

For detailed meanings, refer to the parameter description in McuVideoParams.

Introduction to stream mixing layout types :

VideoParams.LayoutParams.MixLayoutMode has four layout modes. You can choose one according to your needs. The layout modes include dynamic layout (1: floating layout, 2: screen sharing layout, 3: nine-grid layout) and static layout (4: custom layout (default)).

Floating layout

The video of the first user entering the room will fill the entire screen. The videos of other users will be arranged horizontally from the bottom left corner, displayed as small screens. There can be up to 4 rows, with a maximum of 4 videos per row, and the small screens float above the large screen. Up to 1 large screen and 15 small screens are supported. If a user sends only audio, they will still occupy a screen position.

Screen sharing layout

This layout is suitable for video conferences and online education. The shared screen (or the presenter's camera) always occupies the large screen position on the left side of the screen. The screens of other users are arranged vertically on the right side. You need to specify the main screen content on the left side with the VideoParams.LayoutParams.MaxVideoUser parameter. There can be up to 2 columns, with a maximum of 8 small screens per column. Up to 1 large screen and 15 small screens are supported. If a user sends only audio, they will still occupy a screen position.

Nine-grid layout

All users' screens are of equal size, evenly dividing the entire screen. The more users, the smaller each screen becomes. Up to 16 screens are supported. If a user sends only audio, they will still occupy a screen position.

Custom layout (default)

This layout is suitable for scenarios where you need to customize the layout of each screen. You can use the MixLayoutList parameter (an array) in VideoParams.LayoutParams to preset the position of each screen. You may leave the UserId parameter in MixLayoutList unspecified, and the layout engine will assign users to the positions in the MixLayoutList array according to their order of entering the room.

If any position in the MixLayoutList array is configured with a UserId, the layout engine will reserve the position for the specified user. If a user sends only audio, they will still occupy a screen position.

When the preset positions in the MixLayoutList array are used up, the layout engine will no longer mix other users' videos and audios.

For detailed meanings, refer to the parameter description in McuAudioParams.

5. **Set single-stream relay user information (required for single-stream relay).**

To configure single-stream relay, you need to set WithTranscoding to false and specify the SingleSubscribeParams parameters. The MCU will distribute the audio and video streams of the specified user to the specified live streaming CDN.

Field Name	Description	Required
SingleSubscribeParams.UserMediaStream.UserInfo	TRTC user parameters.	No
SingleSubscribeParams.UserMediaStream.StreamType	Primary stream and substream types.	No

For detailed meanings, refer to the parameter description in McuAudioParams.

6. **Set parameters for relay to CDN (required for relay to CDN).**

You need to specify the PublishCdnParams parameters for distributing to CDN. The MCU will relay the encoded audio and video streams to the CDN address you set.

Field Name	Description	Required
PublishCdnParams.N.PublishCdnUrl	CDN relay URL.	Yes
PublishCdnParams.N.IsTencentCdn	Whether it is a Tencent Cloud CDN. 0: non-Tencent Cloud CDN; 1: Tencent Cloud CDN. If this parameter is not specified, the default value is 1. **Note:** **1. To avoid unintended relay fees, it is recommended to explicitly specify this parameter. Relaying to a non-Tencent Cloud CDN will incur relay fees. For details, refer to the API documentation.** **2. By default, the sites in the Chinese mainland only support relaying to a Tencent Cloud CDN. If you need to relay to a third-party CDN, contact Tencent Cloud technical support.**	No

7. **Set the TRTC room push-back parameters (required for pushing back to a TRTC room).**

You need to specify the FeedBackRoomParams parameters for TRTC room push-back. The MCU will relay the encoded audio and video streams to the TRTC room you set.

Field Name	Description	Required
FeedBackRoomParams.N.RoomId	RoomId of the room to which streams are pushed back.	Yes
FeedBackRoomParams.N.RoomIdType	Type of the room to which streams are pushed back. 0: an integer room number; 1: a string room number.	No
FeedBackRoomParams.N.UserId	UserId used by the room to which streams are pushed back. **Note:** **This UserId cannot be the same as other UserIds already used in TRTC or relay services. It is recommended to include the room ID as part of the UserId.**	Yes
FeedBackRoomParams.N.UserSig	The user signature corresponding to the UserId of the	Yes

	room. For the specific calculation method, refer to the TRTC UserSig calculation scheme.	

8. **Set SEI parameters (optional).**

You need to specify either volume layout SEI or pass-through SEI parameters. The MCU will insert the corresponding SEI information into the output video streams.

Field Name	Description	Required
McuSeiParams.LayoutVolume	The SEI for volume layout contains a fixed JSON structure. See the description below for details.	No
McuSeiParams.PassThrough	Pass-through SEI.	No

An example of inserted volume layout SEI is as follows, where app_data indicates the pass-through data, canvas indicates the width and height of the output canvas, regions indicates layout information, volume indicates the mixed stream users' volume (from 0 to 100), with larger values indicating higher volume, ts indicates the local server's second-level timestamp, and ver can be ignored.

```
{
    "app_data": "test",
    "canvas": {
        "w": 1280,
        "h": 720
    },
    "regions": [
        {
            "uid": "test1",
            "zorder": 1,
            "volume": 60,
            "x": 0,
            "y": 0,
            "w": 640,
            "h": 360
        },
        {
            "uid": "test2",
            "zorder": 1,
            "volume": 80,
            "x": 640,
            "y": 0,
            "w": 640,
            "h": 360
        }
    ],
    "ver": "1.0",
    "ts": 1648544726
}
```

Updating a Relay Task

Your server can update a cloud-based relay task by calling the RESTful API UpdatePublishCdnStream. You need to use the TaskId returned by StartPublishCdnStream to initiate the relay update. For details on using this API, refer to Start a relay task.

Stopping a Relay Task

Your server can stop a cloud-based relay task by calling the RESTful API StopPublishCdnStream. You need to use the TaskId returned by StartPublishCdnStream to stop relay.

Field Name	Description
SdkAppId	TRTC's SdkAppId.
TaskId	The unique string ID of the relay task.

Solution 3: Automatic Relay

In addition to manually triggering relay via APIs, TRTC also offers an automatic relay solution. When the automatic relay solution is used, once the host in a TRTC room starts uploading audios and videos, a relay task will automatically be initiated to push the host's single stream to a CDN. Once the host leaves the room, the relay task will automatically end.

Prerequisites

Tencent Cloud CSS service has been enabled. A playback domain name has been configured for CSS. For specific operations, refer to Adding Your Own Domain.

1. Log in to the CSS console.
2. Select **Domain Management** in the left navigation bar, and you will see a new push domain name added to your domain list, formatted as `xxxxx.livepush.myqcloud.com` , where xxxxx represents a number called bizid.
3. Click **Add Domain**, enter the playback domain name you have registered, select the domain name type as **Playback Domain**, select the acceleration region, and click **Confirm**.
4. After the domain name is added successfully, the system will automatically assign you a CNAME domain name (ending with `.liveplay.myqcloud.com`). The CNAME domain name cannot be accessed directly. You need to complete the CNAME configuration with your domain name service provider. After the configuration takes effect, you can enjoy the CSS service. For detailed instructions, please refer to CNAME Configuration.

Note:
You do not need to add a push domain name. After enabling the relayed live streaming feature in Step 1, Tencent Cloud will add a push domain name formatted as `xxxxx.livepush.myqcloud.com` to your CSS console by default. This domain name serves as a default push domain name agreed upon between CSS service and TRTC service.

Global Automatic Relay

Once global automatic relay is enabled, the host in a TRTC room will trigger automatic relay as soon as they start uploading audios and videos.

After enabling global automatic relay, you can also specify a stream ID for pushing to the live streaming CDN using the previously mentioned room entry parameters. If not specified, the system will generate a default stream ID based on the following rules:

Fields used to splice a stream ID

SDKAppID: You can find this in the Console > **Application Management** > **Application Information** section.
bizid: You can find this in the Console > **Application Management** > **Application Information** section.
roomId: Specified by you in `TRTCParams` of the `enterRoom` function.
userId: Specified by you in `TRTCParams` of the `enterRoom` function.
streamType: 'main' is for camera feed, and 'aux' is for screen sharing (WebRTC only supports one upstream at a time, so the stream type for screen sharing on WebRTC is 'main').

Calculation rules for splicing a stream ID

Splicing	Applications created on or after January 9, 2020	Applications created and used before Januar 2020
Splicing Rules	streamId = urlencode(sdkAppId_roomId_userId_streamType)	streamId = bizid_MD5(roomId_userId_streamType)
Calculation Example	If sdkAppId = 12345678, roomId = 12345, userId = userA, and the user is currently using a camera, then streamId = 12345678_12345_userA_main.	If bizid = 1234, roomId = 12345, userId = use and the user is currently using a camera, ther streamId = 1234_MD5(12345_userA_main) 1234_8D0261436C375BB0DEA901D86D7D

Pulling Streams for Playback and Optimization

Configuring a License for the SDK

The TRTC SDK offers comprehensive and powerful live streaming playback capabilities, and can easily integrate with CSS to enable CDN live streaming. If you are using TRTC SDK version 10.1 or later on mobile devices (iOS and Android) to implement CDN live streaming, you need to configure a license; otherwise, you can skip this step.

1. Get the license:

If you have already obtained the license authorization, you need to obtain the license URL and license key in the CSS console.

If you have not obtained the license authorization, you need to apply for it by referring to Adding and Renewing a License.

2. Before your application calls the SDK feature, make the following settings (it is recommended to do so in `Application` / `- [AppDelegate application:didFinishLaunchingWithOptions:]`):

Android

iOS

```
public class MApplication extends Application {

@Override
public void onCreate() {
    super.onCreate();
    String licenceURL = ""; // Obtained license URL
    String licenceKey = ""; // Obtained license key
    V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
    V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
            @Override
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reas
            }
        });
}
```

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    NSString * const licenceURL = @"<Obtained license URL>";
    NSString * const licenceKey = @"<Obtained license key>";

    // V2TXLivePremier is in the V2TXLivePremier.h header file.
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];
    [V2TXLivePremier setObserver:self];
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
    return YES;
}

#pragma mark - V2TXLivePremierObserver
```

```
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

Note:

The packageName/BundleId configured in the license must be consistent with the application itself, otherwise playback will fail.

Obtaining the Playback Address and Connecting for Playback

Once you have completed the stream push operation, you will get the live streaming playback address. The standard format of the playback address is:

```
http://PlaybackDomainName/AppName(default live)/StreamName(StreamID).flv
```

You can check your playback domain name, AppName, and StreamName under Stream Management in the CSS console:

You can get three types of playback addresses:

```
 RTMP protocol playback address:
rtmp://example.myhost.com/AppName_example/StreamName_example
 FLV protocol playback address:
http://example.myhost.com/AppName_example/StreamName_example.flv
 HLS protocol playback address:
http://example.myhost.com/AppName_example/StreamName_example.m3u8
```

Optimizing Playback Latency

After enabling relayed live streaming, the HTTP-FLV address will have higher latency compared to direct conversation in TRTC rooms due to the propagation and distribution by the live streaming CDN.
According to the current Tencent Cloud's live streaming CDN technology, if the V2TXLivePlayer is used, the latency can meet the standards in the table below:

Relayed Stream Type	V2TXLivePlayer Playback Mode	Average Latency	Actual Test Result
Independent screen	Speed mode (recommended)	**2s - 3s**	On the left in the figure below (orange)
Mixed screen	Speed mode (recommended)	**4s - 5s**	On the right in the figure below (blue)

If the latency in your actual test is higher than that in the table above, you can optimize the latency as follows:

Use the TRTC SDK with the built-in V2TXLivePlayer

Ordinary ijkplayer or players based on the ffmpeg kernel lack latency regulation capabilities. If you use such players to play using the live streaming address mentioned above, the latency is generally uncontrollable. V2TXLivePlayer has a Tencent-developed playback engine with latency regulation capabilities.

Set the playback mode of V2TXLivePlayer to speed mode.

You can set the parameters of V2TXLivePlayer to achieve speed mode. For example, on iOS:

```
// Auto mode
[_txLivePlayer setCacheParams:1 maxTime:5];
// Speed mode
[_txLivePlayer setCacheParams:1 maxTime:1];
// Smooth mode
[_txLivePlayer setCacheParams:5 maxTime:5];

// Start playback after setting is completed.
```

Related Fees

Using CDN live streaming requires CSS service resources and terminal SDK live streaming capabilities, which may incur the following fees.

TRTC Fees

Mixed stream transcoding fees: If you use the method of publishing mixed streams to CDNs, mixed stream transcoding fees will be incurred. For details, see MixTranscoding. If you use the method of publishing the local user's stream to CDNs, these fees will not be incurred.

Relay fees: For details, see Relay Fees.

Audio and video duration fees : Audio and video duration fees will be charged based on the actual audio and video usage of users in a room. For details, see Billing of Audio and Video Duration.

Other Cloud Service Fees

CDN live streaming requires the use of **CSS** resources for live distribution. CSS fees mainly include **basic service fees and value-added service fees** . Basic service fees are mainly incurred by consumption of live push/playback services; value-added service fees are mainly incurred by consumption of value-added services during live streaming.

Note:

The prices in this document are examples and for reference only. Final prices and billing policies are subject to the billing explanation of CSS.

Basic service fees:
When relaying TRTC content to a CSS CDN for viewing, **CSS** will charge for the downstream traffic/bandwidth generated by the audience watching. You can choose the billing method that suits your needs. Traffic-based billing is used by default. For details, see CSS > LVB > Traffic/Bandwidth Usage.
Value-added service fees:
If you use CSS features such as transcoding, recording, or Live Video Caster (LVC), additional value-added service fees will be incurred. Value-added services are charged on a pay-as-you-go basis.

Cost Savings

In a stream mixing solution based on the client-side SDK APIs, to stop an backend mixing task, one of the following conditions must be met:
The host who initiated the mixing task (by calling `startPublishMediaStream`) has exited the room.
Actively stop mixing by calling `stopPublishMediaStream` .
In other situations, TRTC cloud will make every effort to maintain the stream mixing state. Therefore, to avoid unexpected mixing fees, stop the cloud-based mixing using the above methods as soon as you do not need it.

FAQs

1. Can I listen for the status of CDN streams? What should I do if an error occurs?

You can listen for the onCdnStreamStateChanged callback event to get the latest status of a relay to CDN task.

2. How do I switch from publishing a single stream to publishing mixed streams? Do I need to stop publishing first and create a new relay task?

To switch from publishing a single stream to publishing mixed streams, just call `updatePublishMediaStream` , passing in the `taskid` of the current task. Note that, in order to ensure the stability of the publishing process, you cannot switch from publishing a single stream to mixing and publishing only audios or only videos. By default, both audio and video data are published when you publish a single stream. If you switch to publishing mixed streams, you must also publish both audios and videos.

3. How can I mix only videos (without audio)?

Do not set the audio parameters in `TRTCStreamEncodeParam` and leave `audioMixUserList` of `TRTCStreamMixingConfig` empty.

4. Can I add watermarks to mixed streams?

Yes, you can use `watermarkList` of `TRTCStreamMixingConfig` to set watermarks.

5. In online learning scenarios, can I mix the screen shared by the teacher?

Yes, you can. We recommend you publish the screen as the substream and mix the teacher's camera video and screen. When specifying the stream mixing parameters, set `fixedVideoStreamType` of `TRTCVideoLayout` to `TRTCVideoStreamTypeSub` .

6. When a preset layout is used, how are audios mixed?

When a preset layout is used, TRTC will mix the audios of up to 16 users in the room.

Enabling Advanced Permission Control

Last updated：2024-08-09 22:25:01

Overview

You may consider **enabling Advanced Permission Control** if you want to allow only specific users to enter a room or use their mics, but are worried that giving permissions on the client side makes the service vulnerable to attacks and cracking.
You do not need to enable advanced permission control in the following scenarios:
Scenario 1: You want an audience as large as possible and do not want to control access to rooms.
Scenario 2: Preventing client-side attacks is not your priority at the moment.
We recommend that you enable advanced permission control for enhanced security in the following scenarios:
Scenario 1: Your video or audio calls have high security requirements.
Scenario 2: You want to implement different access controls for different rooms.
Scenario 3: You want to control the use of mics by audience.

Supported Platforms

iOS	Android	macOS	Windows	Electron	Web	Flutter
✓	✓	✓	✓	✓	✓	✓

Understanding Advanced Permission Control

After you enable advanced permission control, TRTC will verify not only `UserSig` (the room entry ticket), but also **PrivateMapKey** (the permission ticket). The latter contains an encrypted `roomid` and permission bit list.
A user providing only `UserSig` but not `PrivateMapKey` will be unable to enter the specified room.
The permission bit list in `PrivateMapKey` uses the eight bits of a byte to represent different permissions for users holding `PrivateMapKey` .

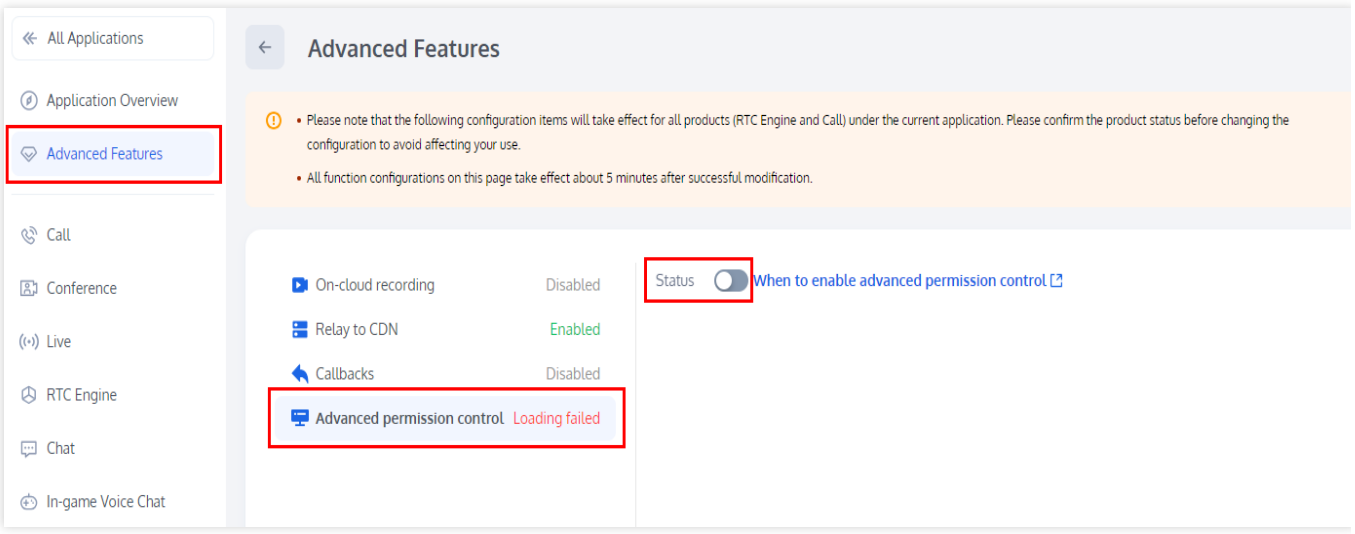
Bit Sequence	Binary	Decimal	Permission
First	0000 0001	1	Room creation
Second	0000 0010	2	Room entry
Third	0000 0100	4	Sending audio

Fourth	0000 1000	8	Receiving audio
Fifth	0001 0000	16	Sending video
Sixth	0010 0000	32	Receiving video
Seventh	0100 0000	64	Sending substream (screen sharing) video
Eighth	1000 0000	128	Receiving substream (screen sharing) video

Enabling Advanced Permission Control

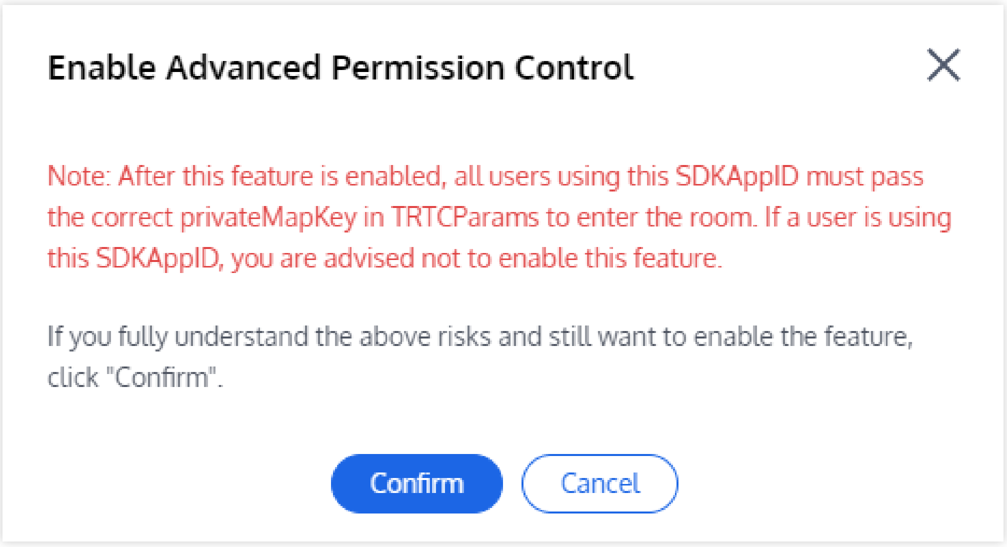
Step 1. Log in to the TRTC console and enable advanced permission control

1. Log to Tencent RTC Console > Applications, click on **Manage** in the row of the target application whose feature configuration needs to be modified, and select **Advanced Features** from the project column on the left.
2. In **Advanced Features**, click the button on the right side of **Enable Advanced Permission Control**, and in the pop-up window, click **Confirm** to complete the activation.



- Please note that the following configuration items will take effect for all products (RTC Engine and Call) under the current application. Please confirm the product status before changing the configuration to avoid affecting your use.
- All function configurations on this page take effect about 5 minutes after successful modification.

On-cloud recording	Disabled
Relay to CDN	Enabled
Callbacks	Disabled
Advanced permission control	Loading failed


Enable Advanced Permission Control

Note: After this feature is enabled, all users using this SDKAppID must pass the correct privateMapKey in TRTCParams to enter the room. If a user is using this SDKAppID, you are advised not to enable this feature.

If you fully understand the above risks and still want to enable the feature, click "Confirm".

Note：

After you enable advanced permission control for an application (`SDKAppid`), all users using the application must pass `privateMapKey` in `TRTCParams` to enter a room (as described in Step 2 below). Therefore, you are not advised to enable the feature if you have active users using the application.

Step 2. Calculate `PrivateMapKey` on your server

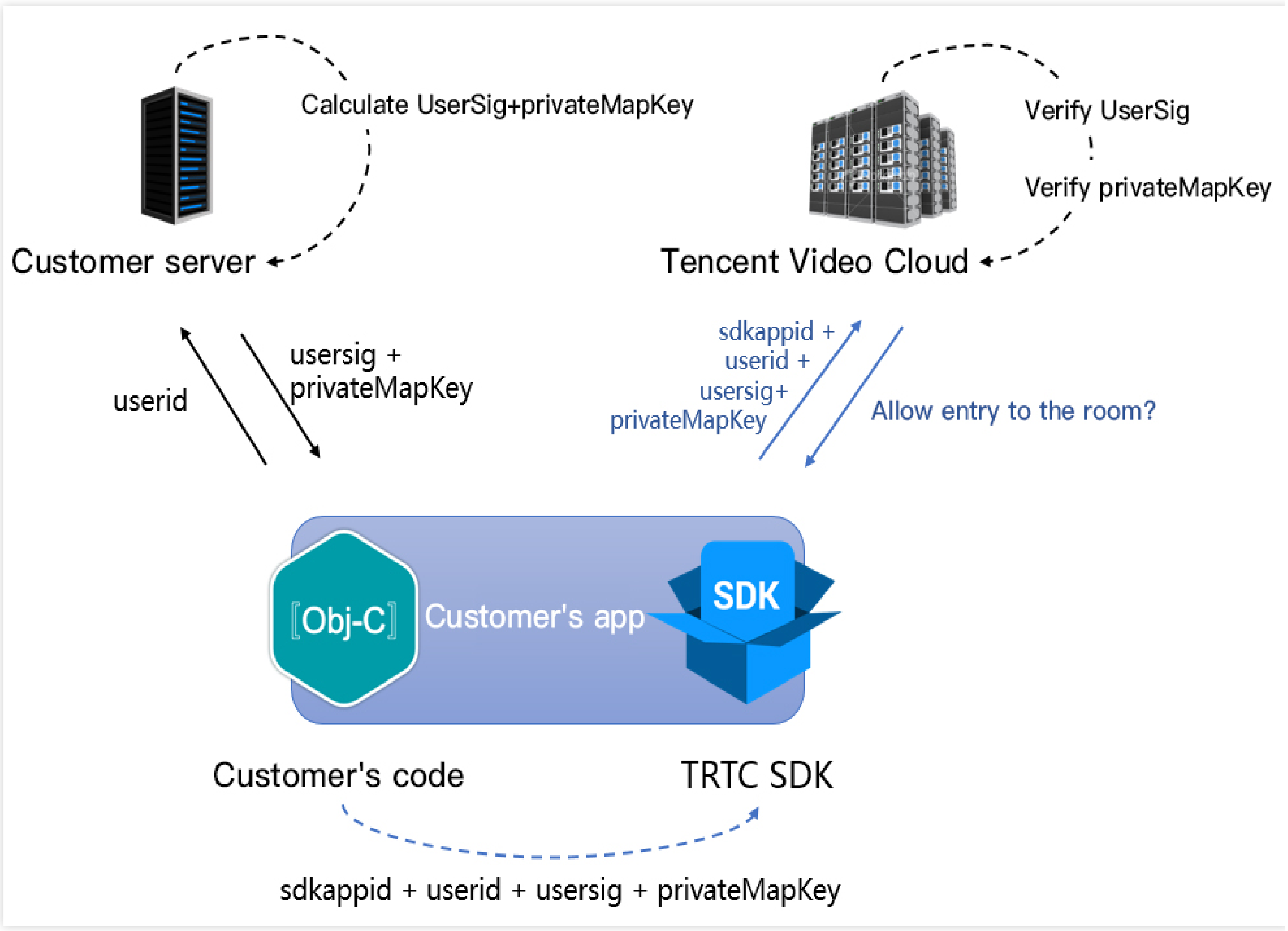
`PrivateMapKey` protects the client from being reverse engineered and cracked and consequently prevents non-members from entering high-level rooms. Therefore, instead of calculating `PrivateMapKey` directly on your application, you should do so on your server and then return the result to your application.

We provide `PrivateMapKey` calculation codes for Java, GO, PHP, Node.js. Python, C#, and C++. You can download and integrate them into your server.

Programming Language	Key Functions	Download Link
Java	`genPrivateMapKey` and `genPrivateMapKeyWithStringRoomID`	GitHub
GO	`GenPrivateMapKey` and `GenPrivateMapKeyWithStringRoomID`	GitHub
PHP	`genPrivateMapKey` and `genPrivateMapKeyWithStringRoomID`	GitHub
Node.js	`genPrivateMapKey` and `genPrivateMapKeyWithStringRoomID`	GitHub
Python	`genPrivateMapKey` and `genPrivateMapKeyWithStringRoomID`	GitHub

C#	`genPrivateMapKey` and `genPrivateMapKeyWithStringRoomID`	GitHub
C++	`genPrivateMapKey` and `genPrivateMapKeyWithStringRoomID`	GitHub

Step 3. Distribute `PrivateMapKey` from your server to your application



As shown in the figure above, `PrivateMapKey` is calculated on your server and distributed to your application, which can then pass the `PrivateMapKey` to the SDK via two methods.

Method 1: passing `PrivateMapKey` to the SDK when calling `enterRoom`

You can set **privateMapKey** in TRTCParams when calling the `enterRoom` API of `TRTCCloud` .
This method verifies `PrivateMapKey` when users enter a room. It is simple and is used to assign permissions to users before room entry.

Method 2: updating `PrivateMapKey` to the SDK through an experimental API

During live streaming, when audience turn their mics on to co-anchor, TRTC will re-verify the `PrivateMapKey` carried in `TRTCParams` at the time of room entry. That means if you set a short validity period for `PrivateMapKey` , such as 5 minutes, the re-verification may fail and cause the audience to be removed from the room when they switch to the role of “anchor”.
To solve this issue, you can extend the validity period, for example, from 5 minutes to 6 hours or, before the audience call `switchRole` to switch to the role of “anchor”, apply for a new `PrivateMapKey` from your server and update it to the SDK by calling the experimental API `updatePrivateMapKey` . Below is the sample code:

Android
iOS
C++
C#

```
JSONObject jsonObject = new JSONObject();
try {
    jsonObject.put("api", "updatePrivateMapKey");
    JSONObject params = new JSONObject();
    params.put("privateMapKey", "xxxxx"); // Enter the new `privateMapKey`.
    jsonObject.put("params", params);
    mTRTCCloud.callExperimentalAPI(jsonObject.toString());
} catch (JSONException e) {
    e.printStackTrace();
}
```

```
NSMutableDictionary *params = [[NSMutableDictionary alloc] init];
[params setObject:@"xxxxx" forKey:@"privateMapKey"]; // Enter the new `privateMapKe
NSDictionary *dic = @{@"api": @"updatePrivateMapKey", @"params": params};
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:dic options:0 error:NULL
NSString *jsonStr = [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEn
[WXTRTCCloud sharedInstance] callExperimentalAPI:jsonStr];
```

```
std::string api = "{\\"api\\":\\"updatePrivateMapKey\\",\\"params\\":{\\"privateMap
TRTCCloudCore::GetInstance()->getTRTCCloud()->callExperimentalAPI(api.c_str());
```

```
std::string api = "{\\"api\\":\\"updatePrivateMapKey\\",\\"params\\":{\\"privateMap
mTRTCCloud.callExperimentalAPI(api);
```

FAQs

1. Why can't I enter any online room?

After you enable room permission control for an application (`SDKAppid`), users must pass `PrivateMapKey` in `TRTCParams` to enter any room under the application. Therefore, if your online business is running, and you haven’t integrated into it the `privateMapKey` logic, please do not enable room permission control.

2. What is the difference between `PrivateMapKey` and `UserSig` ?

`UserSig` is a required parameter of `TRTCParams` , which is used to check whether the current user is authorized to use TRTC services and prevent attackers from stealing the traffic in your application (`SDKAppid`).
`PrivateMapKey` is an optional parameter of `TRTCParams` , which is used to check whether the current user is authorized to enter the specified room (`roomid`) and confirm the user’s permissions in the room. Use `PrivateMapKey` only if you need to distinguish users from one another.

Push Media Stream into TRTC

Last updated：2024-12-02 10:08:55

Overview

Watch together, listen together, play together, learn together... Various experiences that once required face-to-face interaction are now moving online. Even if separated by thousands of miles, friends can still watch movies, listen to music and chat together. This amazing real-time interactive experience is becoming popular among young people and is now a major feature and mainstream direction of audio and video products.

TRTC offers two streaming solutions: push online media stream and RTMP streaming with TRTC, each with their own application scenario, as detailed below:

Push online media stream is used to **pull cloud-based online media streams (online streaming or cloud-based on-demand files)** and push them into a TRTC room.

RTMP streaming with TRTC is used to **stream local media files and audio and video from capture devices** into a TRTC room via the RTMP standard protocol.

Note:

Relevant fees are as follows:

Feature unlocking: **Push online media stream** and **RTMP streaming with TRTC** features need to be unlocked by a subscription to the **basic** or **professional version** of RTC-Engine monthly package.

Usage fees:

Using the streaming feature involves transcoding operations, incurring transcoding costs. For more details, see the Description of Billing of MixTranscoding and Bypass Relay.

The costs of audio duration incurred by the streaming robot in the room are charged (Note: The costs incurred by the robot in the room for push online media stream features will not be charged by August 15, 2024, and they will begin to be charged on August 16, 2024).

The audience in the room subscribing to audio and video streams will incur audio and video call costs. For details, see the Description of Billing of Audio and Video Duration.

Push Online Media Stream

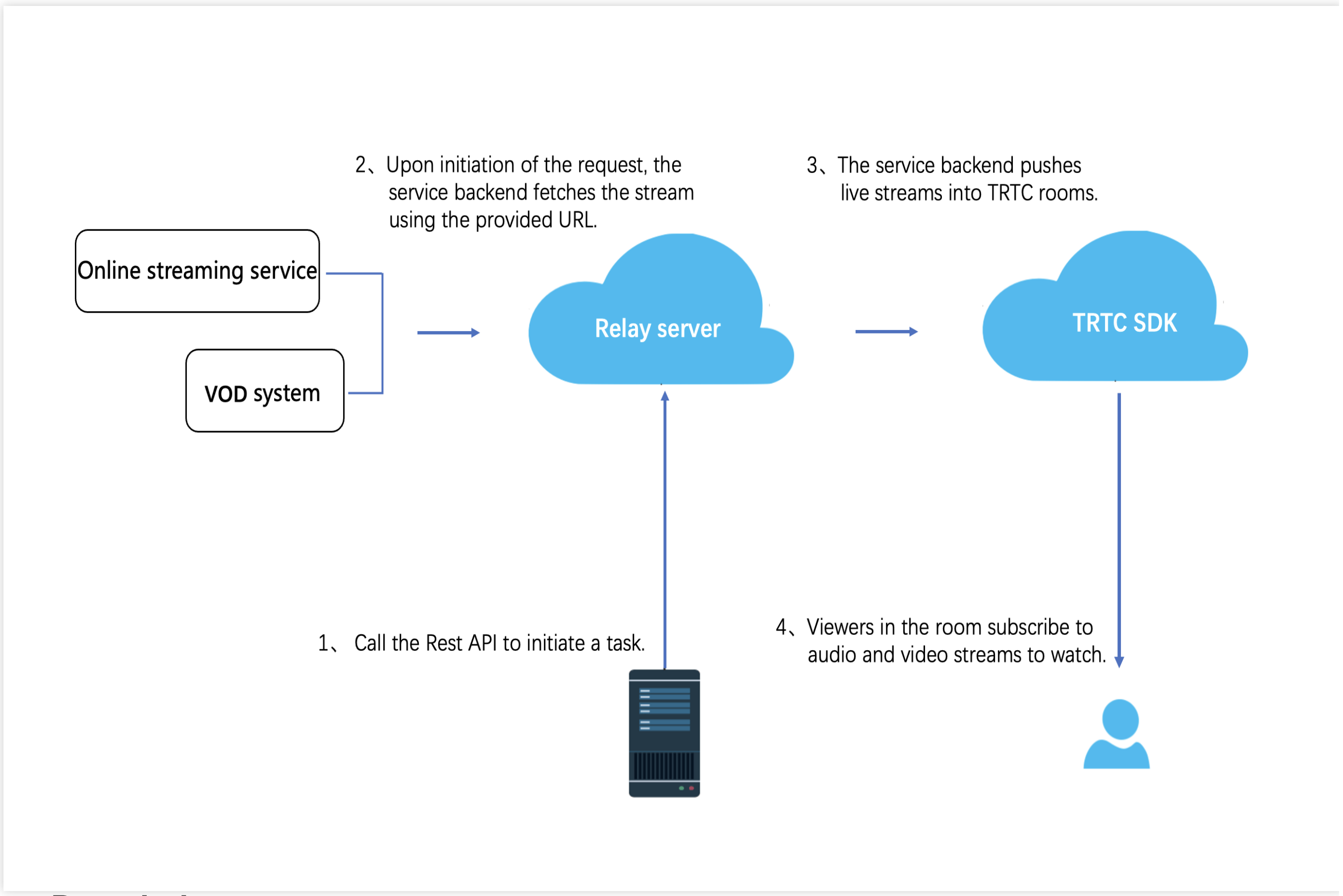
Application Scenario

Scenario type	Description
AI interactive classroom	Relying on TRTC's ability to push online media stream, the platform enables online live interactive teaching by combining recorded real-life teaching videos with AI technology. This significantly reduces operational costs while ensuring teaching effectiveness. Before class,

	the platform records video segments for explanation of knowledge points, interactive questioning, feedback on questions, and answers based on the teacher's course setup, and uploads them to the video library. During the class, these videos are streamed to the TRTC room for live broadcasting through the TRTC push online media capability. Students can engage in interactive learning through voice and touchscreen. The server uses AI technology to evaluate real-time voice and responses of students, seamlessly switches teaching segments, and provides different real-time feedback, thus offering a personalized teaching experience.
"Watch together" room service	Live streaming contents, such as game live streaming, fashion shows, and sports events, can be pushed to a TRTC room through the TRTC's push online media stream capability for ultra-low latency synchronized viewing within the room. With TRTC's real-time interaction capability, the audience can communicate in real-time, cheer together, and enjoy an immersive viewing experience. On-demand programs like movies and music can also be inputted into the TRTC room through the capability, allowing users to share in real-time and chat with friends while watching.

Feature Architecture

1. Users create push online media stream tasks using the REST API. These tasks are executed by the relay server.
2. The relay server pulls online streams or on-demand files.
3. The relay server pushes the fetched audio and video to the TRTC room and automatically generates a virtual anchor user. The username and room number of this user are specified when the task is created.
4. Other TRTC clients can watch these streams and utilize TRTC capabilities such as recording and relay.


2、Upon initiation of the request, the service backend fetches the stream using the provided URL.
3、The service backend pushes live streams into TRTC rooms.
Online streaming service
VOD system
Relay server
TRTC SDK
1、 Call the Rest API to initiate a task.
4、Viewers in the room subscribe to audio and video streams to watch.

Feature Description

The feature description of push online media stream is as follows:

Type	Description
Task initiation method	Users can initiate push online media stream tasks via the REST API. The audience can watch these streams, and features such as recording and relay are supported.
Multiple source protocols and formats	Protocols: HTTP, HTTPS, RTMP, HLS Formats: FLV, MP3, MP4, MPEG-TS, MOV, MKV, M4A Video encoding: H.264, VP8 Audio encoding: AAC, OPUS
Server-side callback	When a push online media stream task is created and completed, it can be called backed to the server on the service side for service logic purposes. For detailed push online media stream events, go to view.

Related Rest API

Start push online media stream: StartStreamIngest

Stop push online media stream: StopStreamIngest

Query push online media stream: DescribeStreamIngest

RTMP Streaming With TRTC

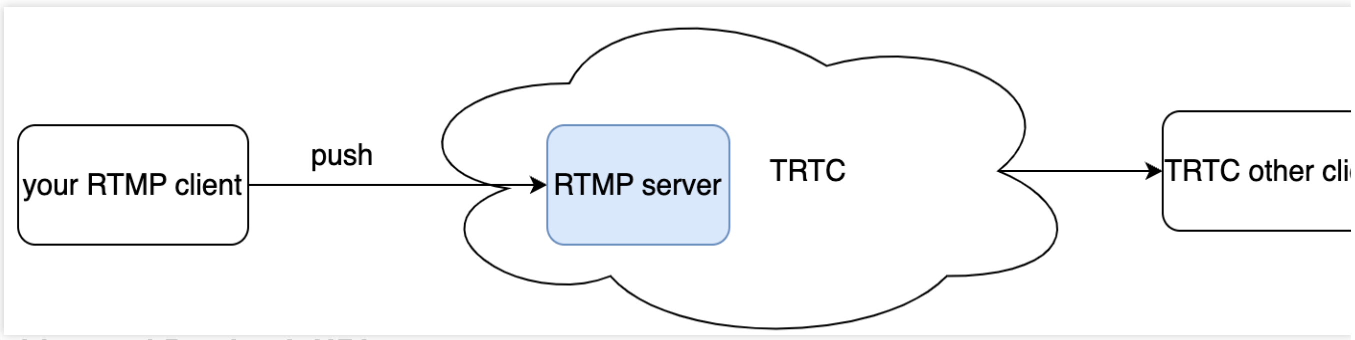
TRTC supports the streaming of **local media files** and **audio and video from capture devices** into a TRTC room via the RTMP standard protocol. To facilitate your integration of TRTC, you can install OBS, FFmpeg, or other RTMP libraries to realize streaming with TRTC. OBS is a third-party open-source tool for live streaming. It is easy to use and free of charge, and it supports OS X, Windows, and Linux. OBS can be used in a wide range of scenarios and is capable of meeting most live streaming needs without requiring additional plugins. You can download its latest version from the OBS website.

Use Cases

Scenario	Description
Online education	Use the desktop edition of OBS or FFmpeg to publish learning materials (most media formats are supported) over RTMP to a TRTC room. Students in the room can play the stream via the TRTC SDK and see the same learning materials as the teacher controls the playback progress/speed or switches between chapters. Excellent synchronization across multiple devices ensures better teaching quality.
Sports watching	Sports event organizers provide content in the form of RTMP streams. You can publish the streams to TRTC rooms so that users in the rooms can watch the event with ultra-low latency. With TRTC’s interaction capability, users can also audio/video chat with each other throughout an event.
Others	You can also use the RTMP publishing feature to implement other real-time interactive applications based on streaming.

Architecture

An RTMP client is a module of TRTC and can communicate with other TRTC clients. The interconnection delay is less than 600ms under normal circumstances. It can also use TRTC capabilities such as recording and relaying. The network architecture is shown in the figure below. **Not support pulling RTMP stream from TRTC; only support pushing stream to TRTC.**



Publishing and Playback URLs

Publishing URLs

```
rtmp://intl-rtmp.rtc.qq.com/push/Room ID?sdkappid=Application ID&userid=User
ID&usersig=Signature
```

Primary domain is intl-rtmp.rtc.qq.com; backup domain is rtmp.rtc-web.com. If there are issues with the primary domain DNS resolution, you can use the backup domain.

For RTMP publishing, `appName` is `push`.

Replace "Room ID", "Application ID", "User ID", and "Signature" with their actual values.

For the sake of simplicity, we support only string-type room IDs. A room ID can contain numbers, letters, and underscores and cannot exceed 64 characters.

Warning:

To play an RTMP stream on other TRTC clients, when entering the room, make sure you use a string-type room ID.

For how to generate `UserSig`, see UserSig. **Make sure your signature is within the validity period**.

Example:

```
rtmp://intl-rtmp.rtc.qq.com/push/hello-string-room?
sdkappid=140*****66&userid=******rtmp2&usersig=eJw1jdE********RBZ8qKGRj8Yp-
wVbv*mGMVZqS7w-mMDQL
```

Usage Example

You can use software or a programming library that supports RTMP to publish RTMP streams. The section below shows you how to do this.

Using OBS to publish streams

Prerequisites

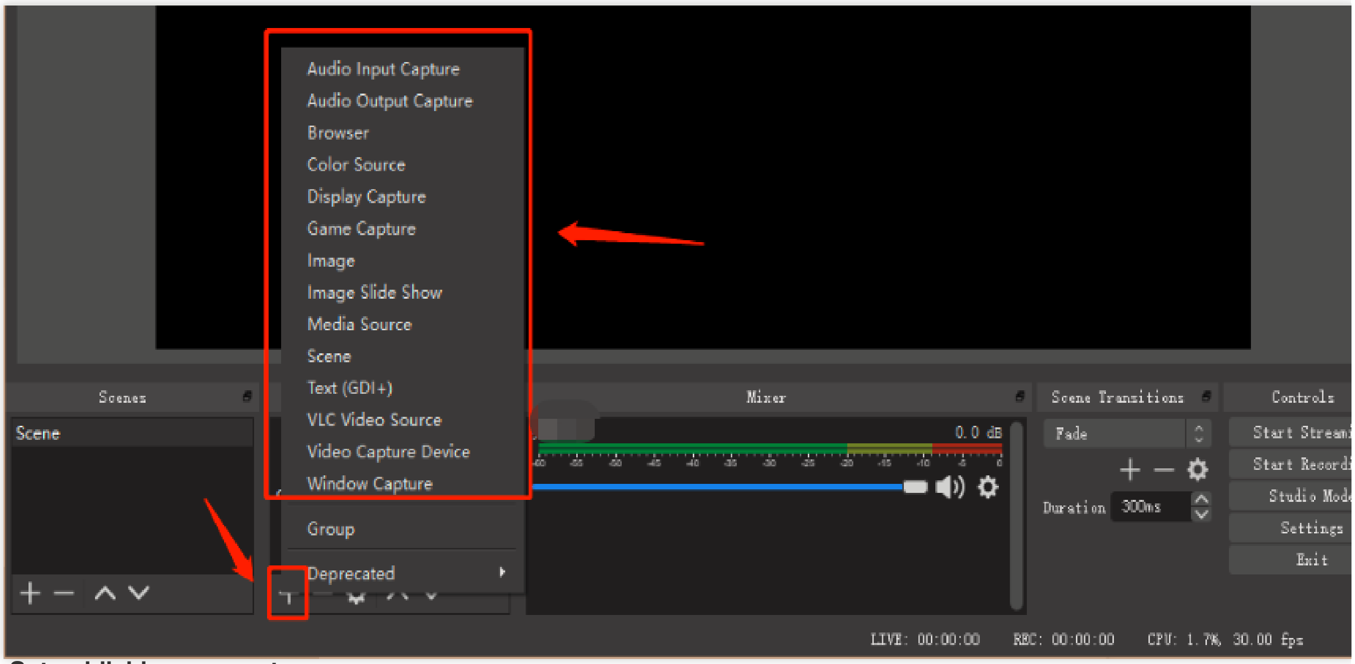
You have installed OBS.

Step 1. Select a source

In the **Sources** panel at the bottom, click **+**, and select a source based on your needs. Common sources include the following:

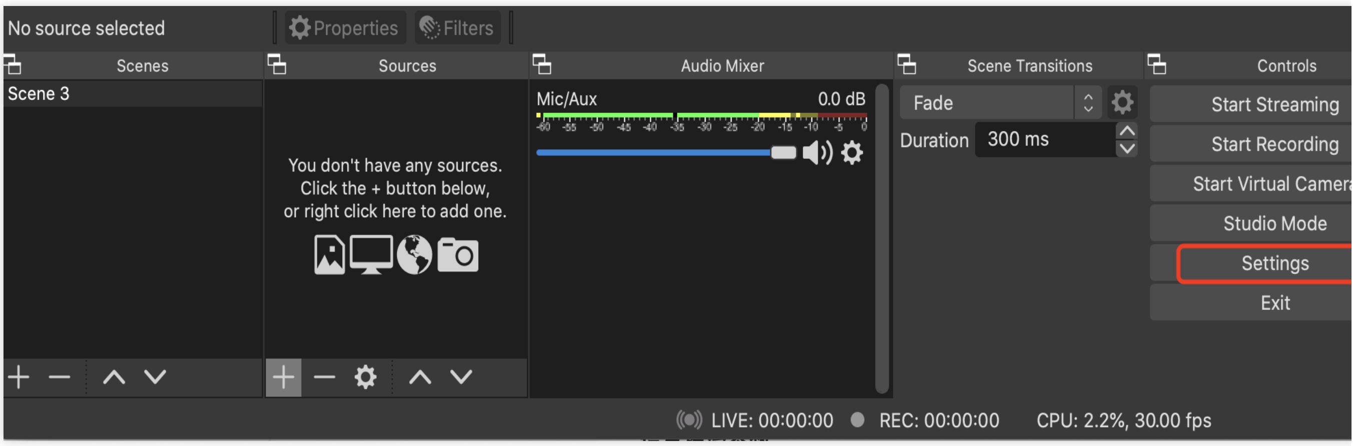
Source	Note
Image	Publishes a single image
Image Slide Show	Publishes multiple images (you can determine the order of playback and whether to loop the playback)
Scene	Inserts an entire scene to enable various streaming effects
Media Source	Uploads a local file and publishes it as a live stream

Text	Adds real-time text to your stream
Window Capture	Captures and publishes the window you select in real time
Video Capture Device	Captures and publishes the images captured by a camera in real time
Audio Input Capture	Audio live streaming (audio input device)
Audio Output Capture	Audio live streaming (audio output device)



Step 2. Set publishing parameters

1. In the **Controls** panel at the bottom, click **Settings**.

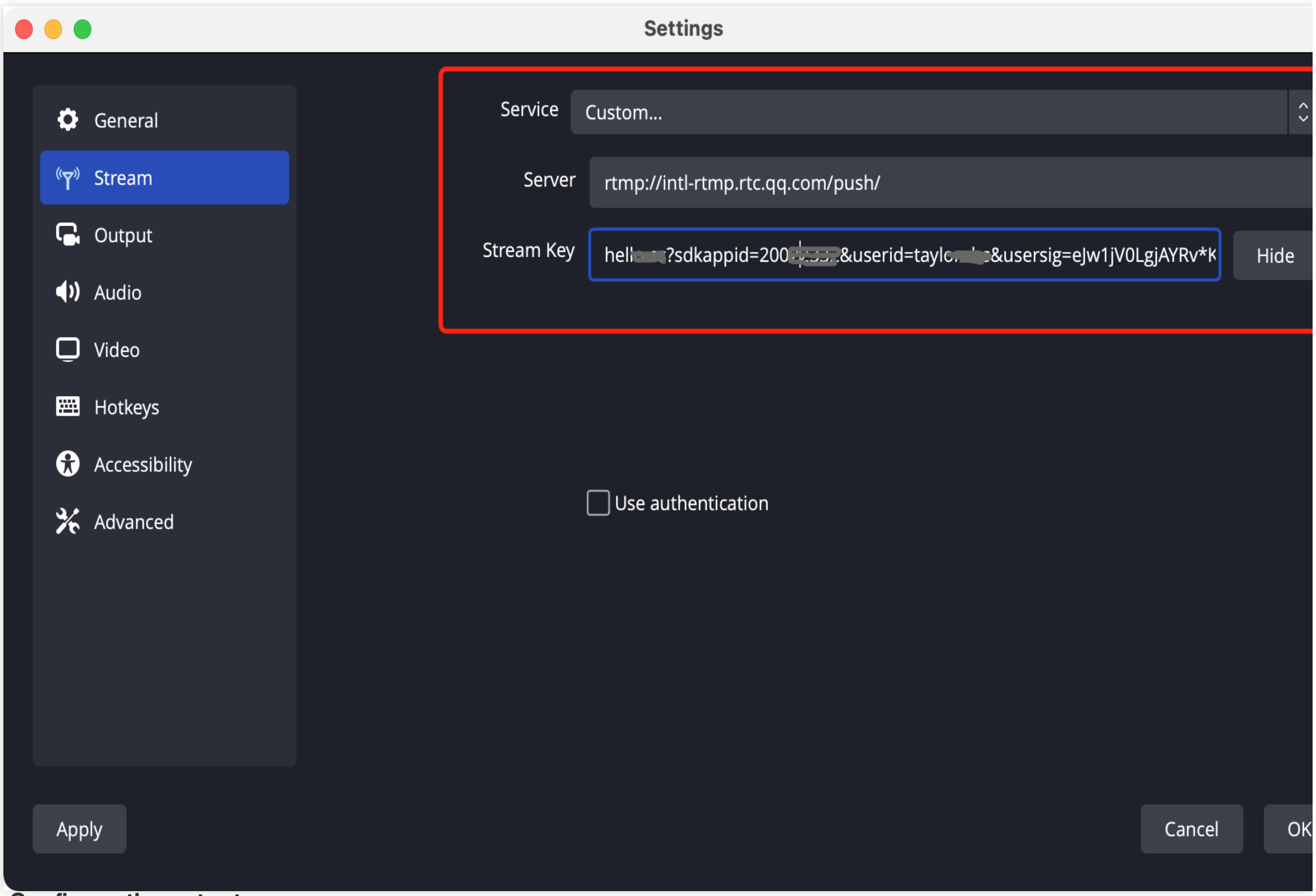


2. Click **Stream** and select **Custom** for **Service**.
3. Enter `rtmp://intl-rtmp.rtc.qq.com/push/` for **Server**.
4. Enter a stream key in the following format:

```
Room ID?sdkappid=Application&userid=User ID&usersig=Signature
```

Replace "Room ID", "Application ID", "User ID", and "Signature" with the actual values, for example:

```
hello-string-room?
sdkappid=140*****66&userid=******rtmp2&usersig=eJw1jdE*************ZLgi5UAgOz
oMhrayt*cjbmiCJ699T09juc833IMT94Ld7I0iHZqVDzvVAqkZsG-
IKlzLiXOnEhswHu1iUyTc9pv*****D8MQwoA496Ke6U1ip4EAH4UMc5H9pSmv6MeTBWLamhwFnWRBZ8
qKGRj8Yp-wVbv*mGMVZqS7w-mMDQL
```



Service: Custom...

Server: rtmp://intl-rtmp.rtc.qq.com/push/

Stream Key: hell[redacted]?sdkappid=200[redacted]&userid=taylo[redacted]&usersig=eJw1jV0LgjAYRv*K

☐ Use authentication

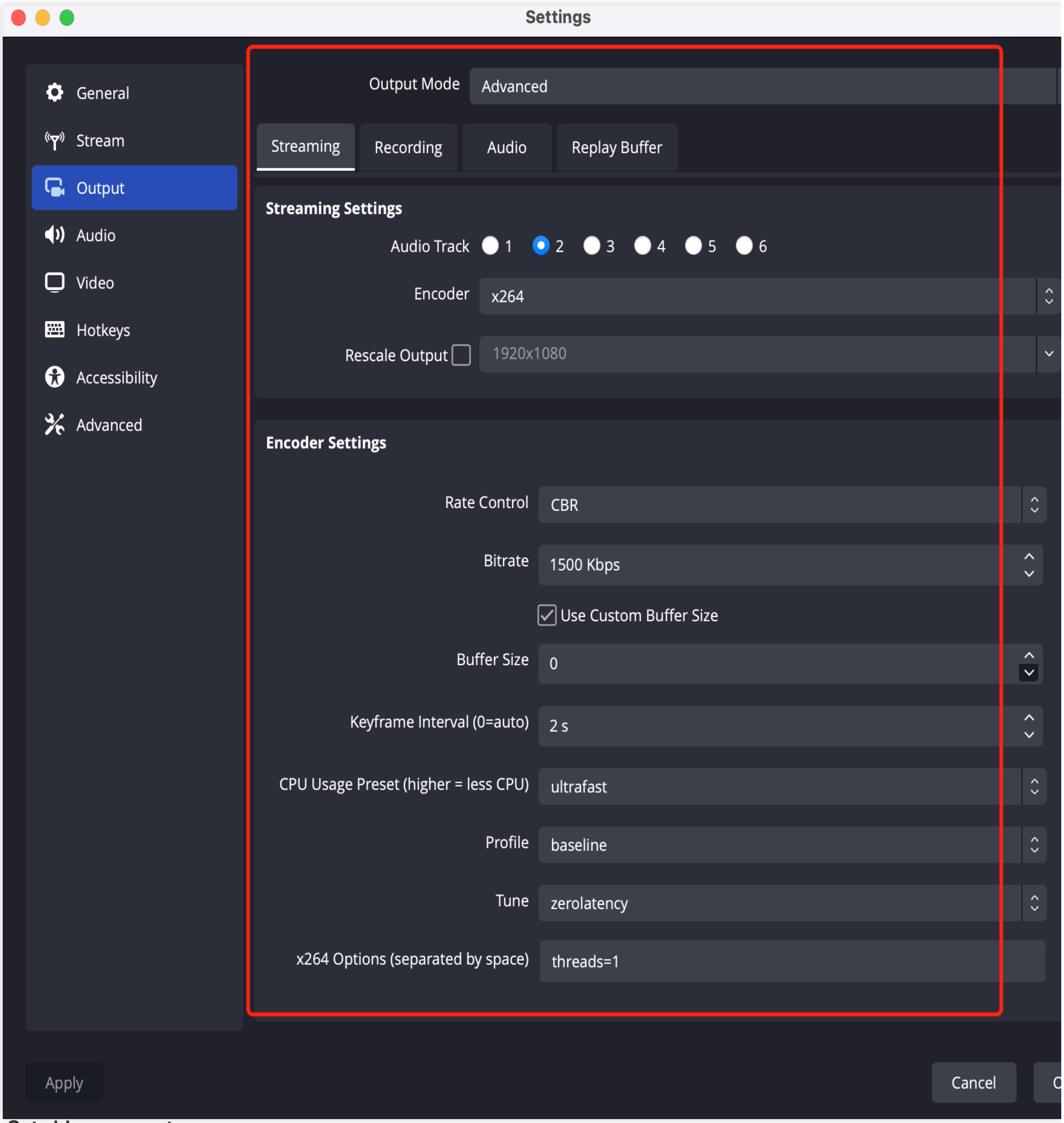
Step 3. Configure the output

Because RTMP does not support B-frames, set the video encoding parameters as follows to remove B-frames.

1. Go to **Controls** > **Settings** > **Output**.
2. Select **Advanced** for **Output Mode**. The recommended **Keyframe Interval** is **1** or **2**. For **CPU Usage Preset**, select **ultrafast**. For **Profile**, select **baseline**. For **Tune**, select **zerolatency**. And for **x264 Options**, enter `threads=1` , and then click **OK**.

Warning:

You need to remove the B-frames in RTMP streams, otherwise the connection will be disconnected after pushing. To achieve this, select baseline for Profile.



Output Mode: Advanced

Streaming Settings

Audio Track: 1 [✓ 2] 3 4 5 6

Encoder: x264

Rescale Output ☐ 1920x1080

Encoder Settings

Rate Control: CBR

Bitrate: 1500 Kbps

☑ Use Custom Buffer Size

Buffer Size: 0

Keyframe Interval (0=auto): 2 s

CPU Usage Preset (higher = less CPU): ultrafast

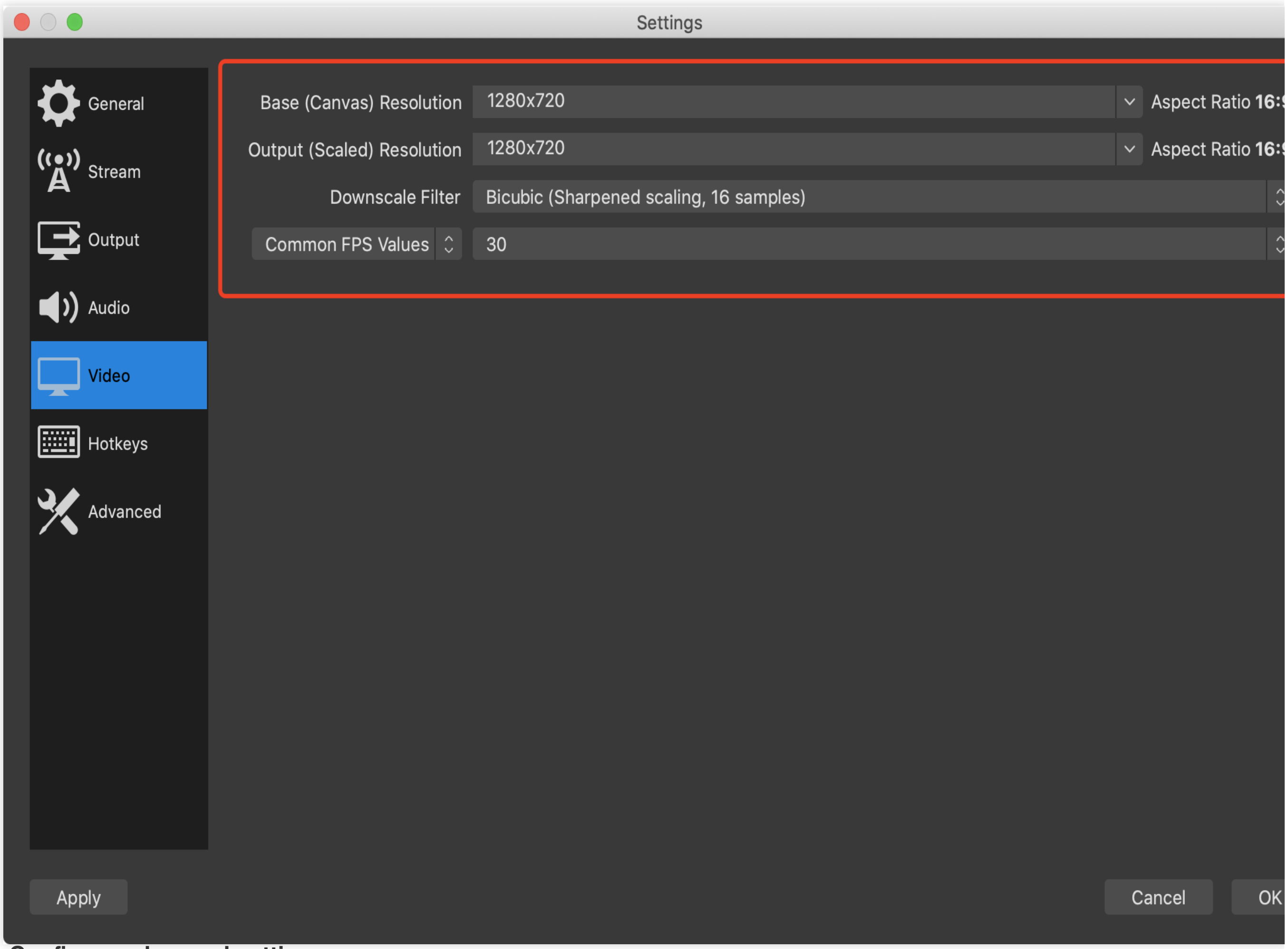
Profile: baseline

Tune: zerolatency

x264 Options (separated by space): threads=1

Step 4. Set video parameters

You can set video resolution and frame rate under the **Video** section of **Settings**. Resolution determines the clarity of video shown to audience members. The higher the resolution, the clearer the video. Frame rate (frames per second) determines playback smoothness. Typical frame rate falls in the range of 24 fps to 30 fps. Playback may stutter if frame rate is lower than 16 fps. Video games require higher frame rate and tend to stutter at a frame rate lower than 30 fps.



Base (Canvas) Resolution 1280x720 Aspect Ratio 16:

Output (Scaled) Resolution 1280x720 Aspect Ratio 16:

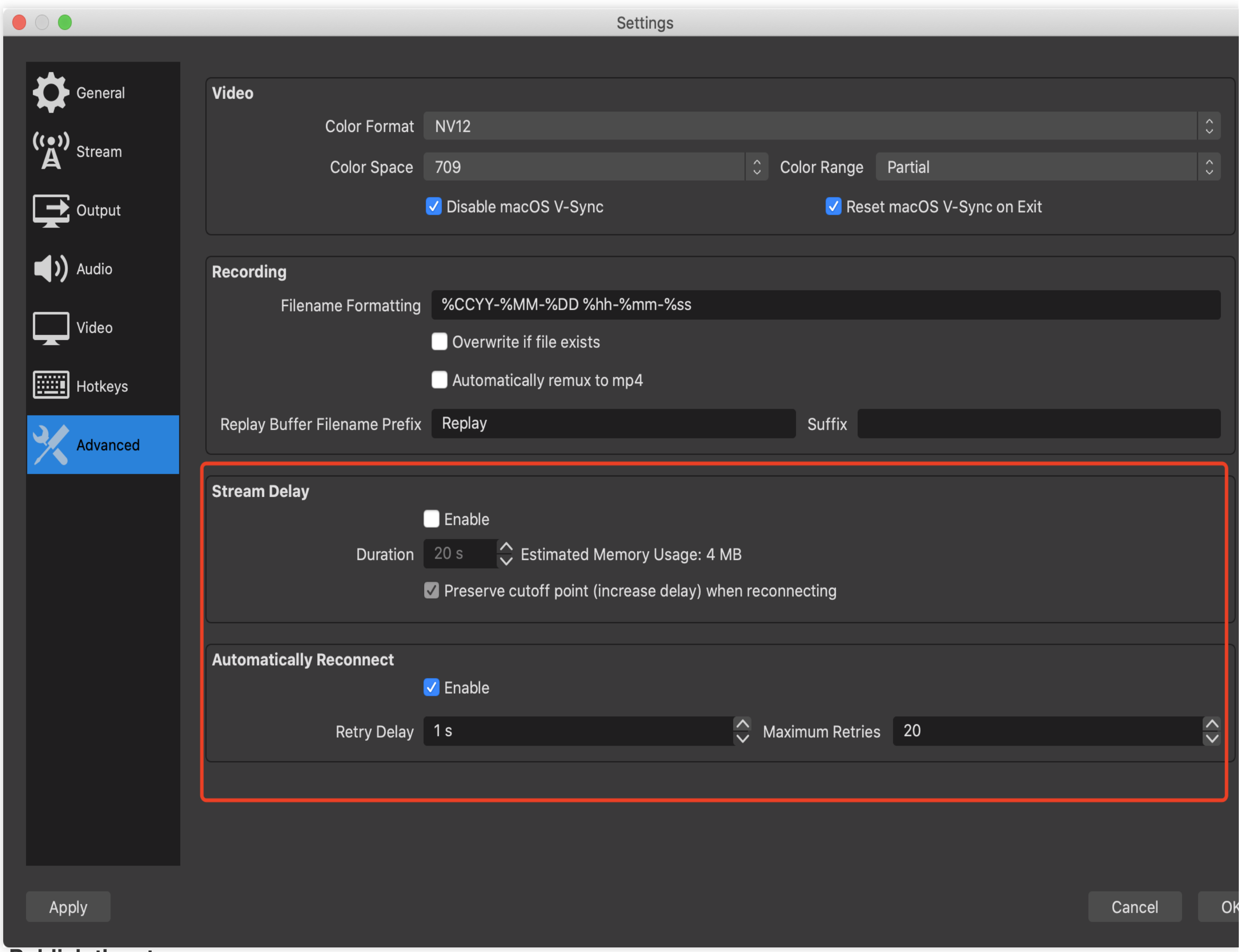
Downscale Filter Bicubic (Sharpened scaling, 16 samples)

Common FPS Values 30

Step 5. Configure advanced settings

To reduce end-to-end delay, we recommend you do not enable **Stream Delay**.

Keep **Automatically Reconnect** enabled and make **Retry Delay** as short as possible so that the publisher can be reconnected quickly after a disconnection occurs due to network jitter.



Video
Color Format: NV12
Color Space: 709 Color Range: Partial
☑ Disable macOS V-Sync ☑ Reset macOS V-Sync on Exit

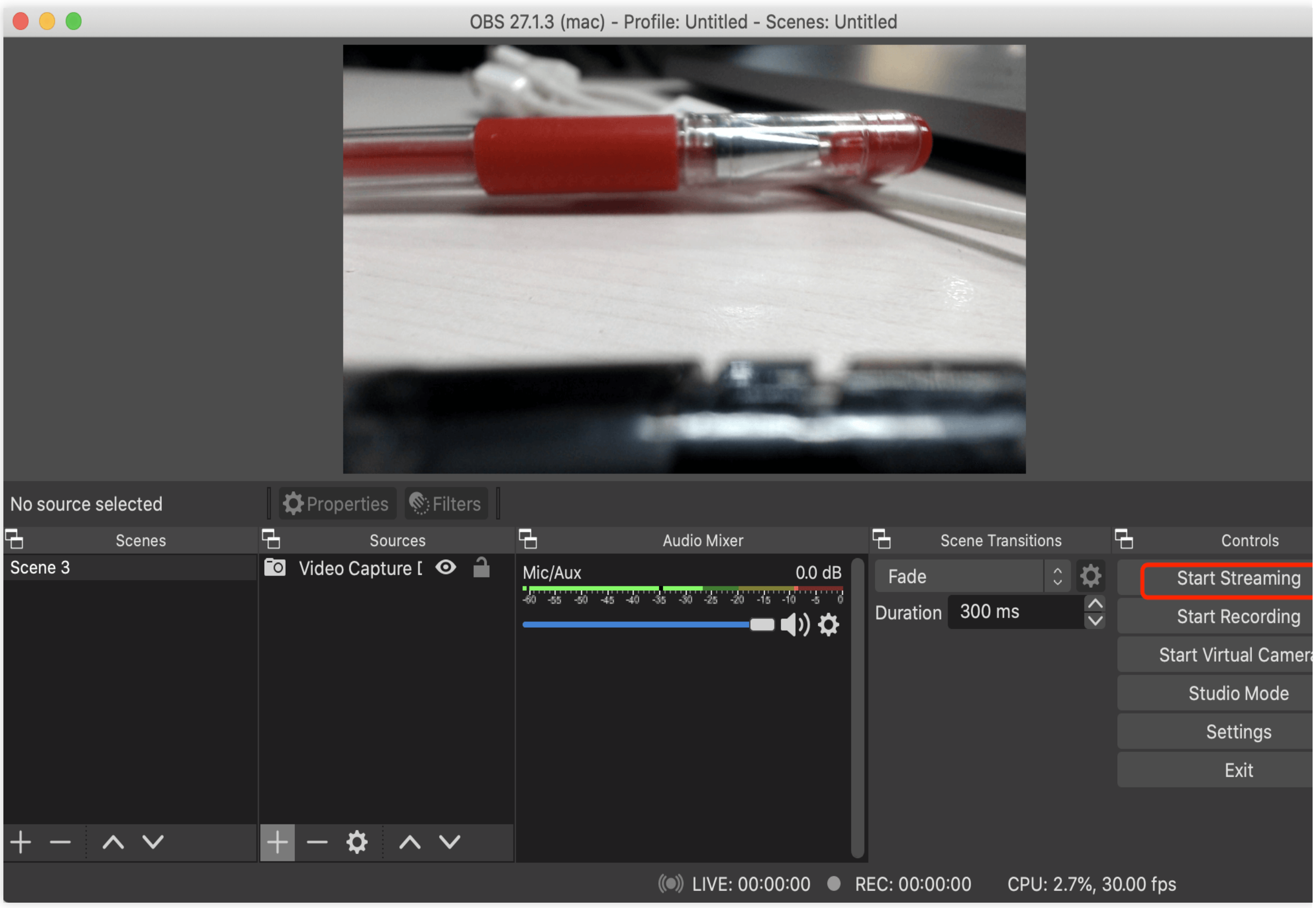
Recording
Filename Formatting: %CCYY-%MM-%DD %hh-%mm-%ss
☐ Overwrite if file exists
☐ Automatically remux to mp4
Replay Buffer Filename Prefix: Replay Suffix:

Stream Delay
☐ Enable
Duration: 20 s Estimated Memory Usage: 4 MB
☑ Preserve cutoff point (increase delay) when reconnecting

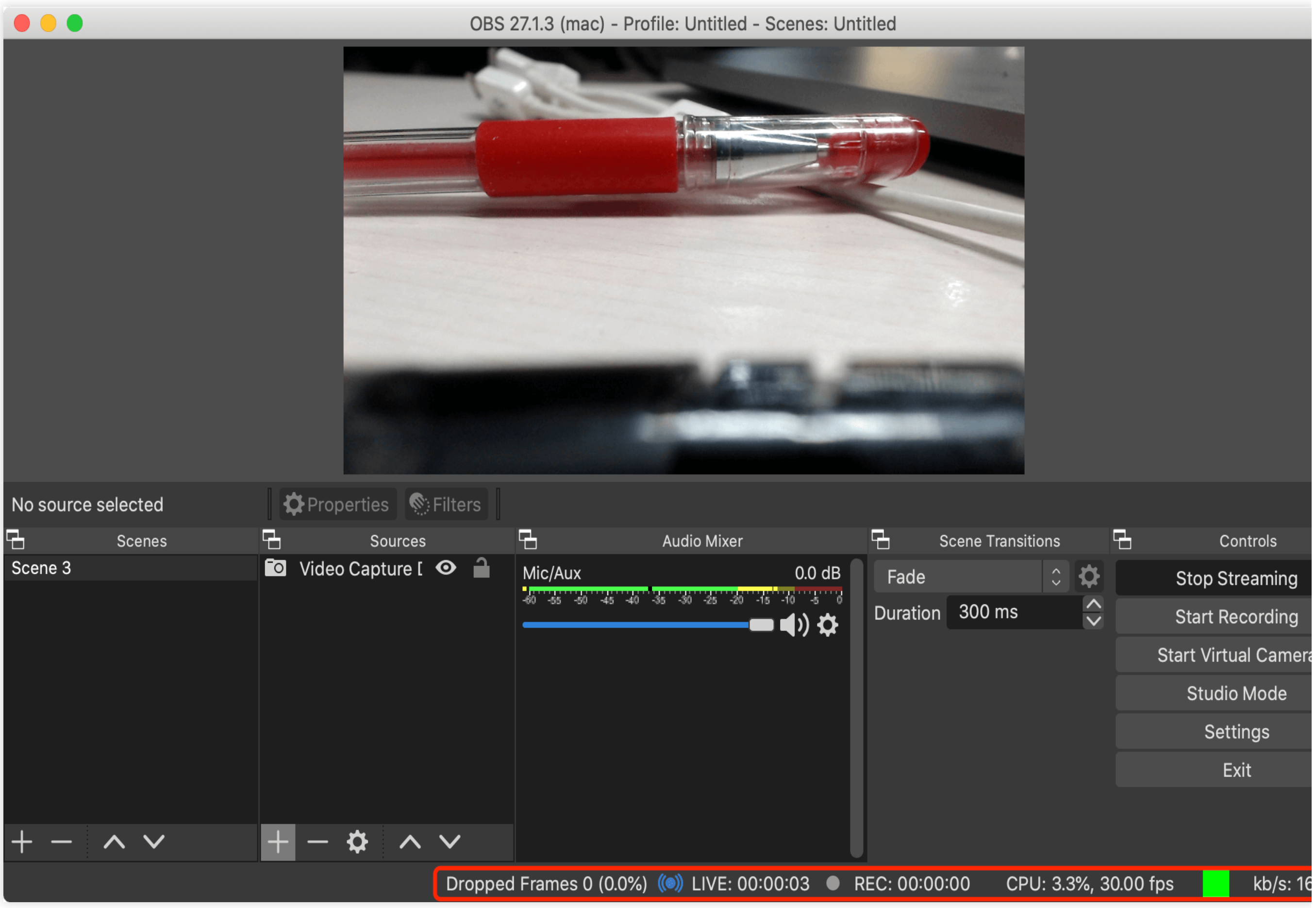
Automatically Reconnect
☑ Enable
Retry Delay: 1 s Maximum Retries: 20

Step 6. Publish the stream

1. In the **Controls** panel at the bottom, click **Start Streaming**.



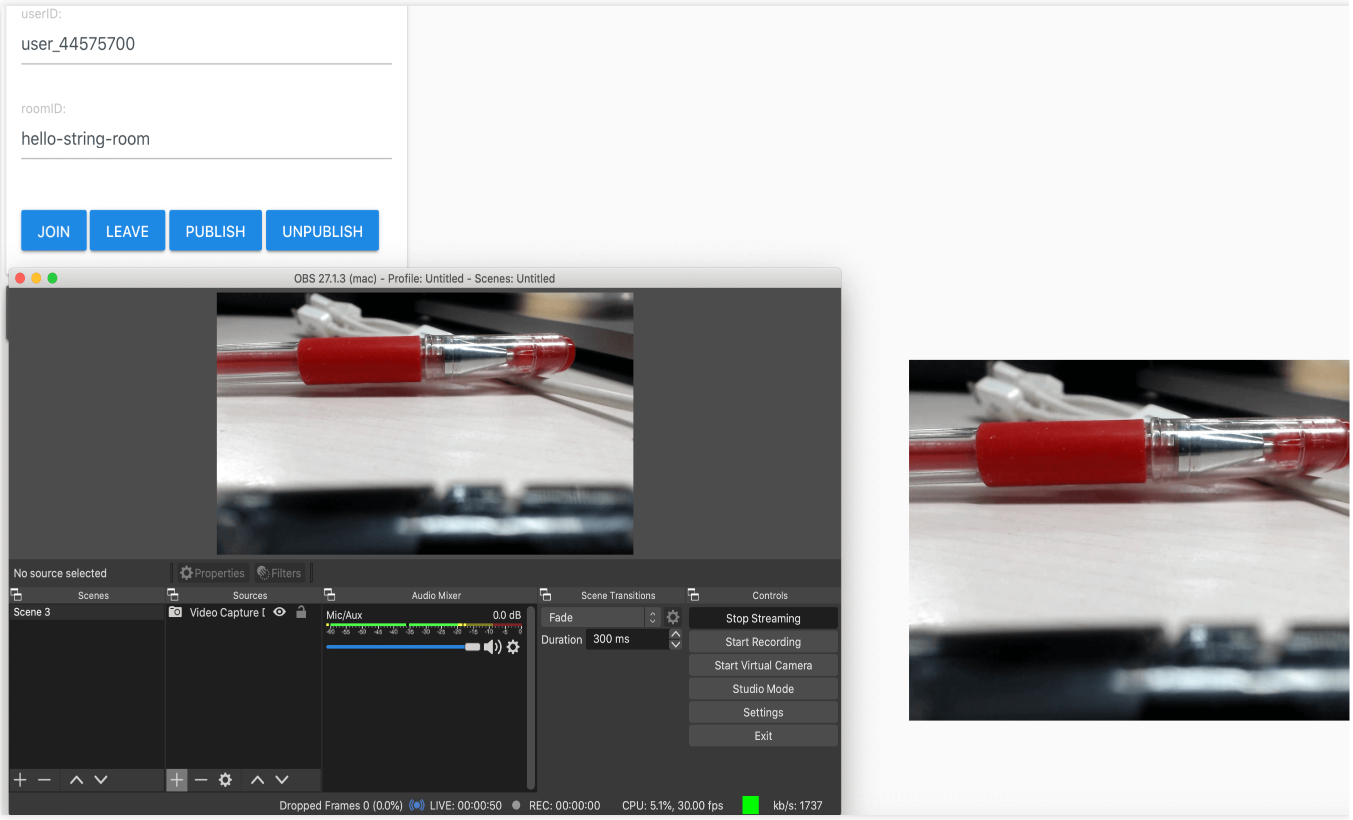
2. If streaming is successful, the bottom bar will show streaming statistics, and the entry of a user will be recorded by the TRTC monitoring dashboard.



Dropped Frames 0 (0.0%) LIVE: 00:00:03 REC: 00:00:00 CPU: 3.3%, 30.00 fps kb/s: 16

Step 7. Play the stream on other clients

As mentioned above in Set publishing parameters, to play the RTMP stream on other TRTC clients, you need to use a string-type room ID when entering the room. The screenshot below is an example of playing the RTMP stream on Web. （ps : You can go to the Demo page and enter the room on any client-side to view the stream .）



userID: user_44575700

roomID: hello-string-room

Using FFmpeg to publish streams

To publish RMTP streams using FFmpeg commands or other RTMP libraries, you need to use the full URL, the H.264 video codec, and the AAC audio codec. For the container format, FLV is recommended. For GOP, one or two seconds is recommended.

The configuration of FFmpeg parameters varies with different scenarios, so you need to have some knowledge of FFmpeg in order to use it to publish streams. The table below lists some common FFmpeg commands. For more options, see FFmpeg documentation.

FFmpeg commands

```
ffmpeg [global_options] {[input_file_options] -i input_url} ...
{[output_file_options] output_url}
```

Common FFmpeg options

Option	Note
-re	Reads input at the native frame rate. This is usually used to read local files.

Options for **output_file_options** include:

Option	Note

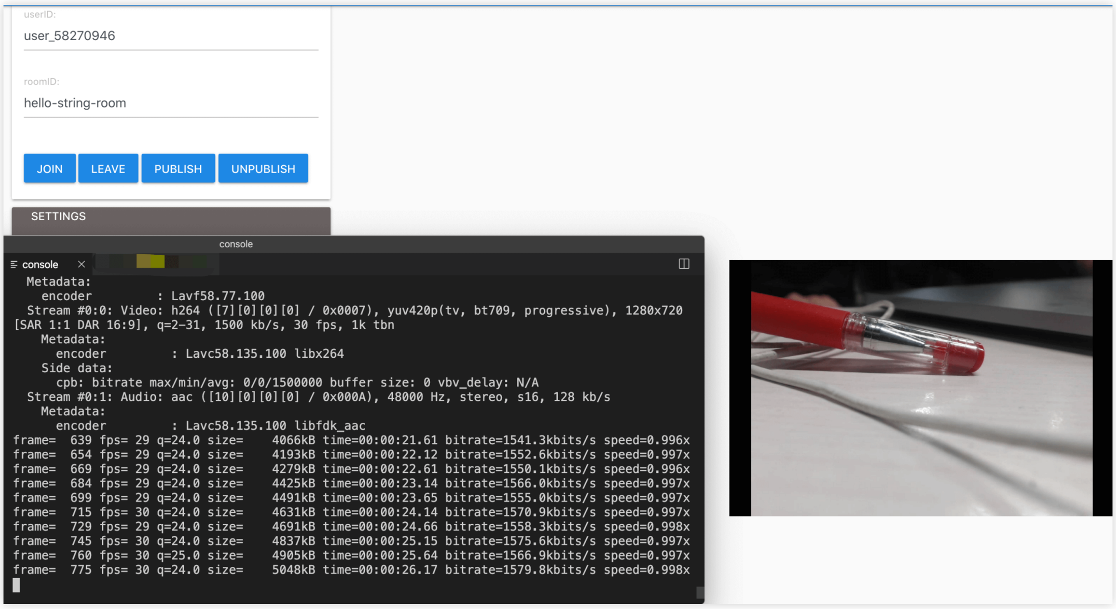
-c:v	The video encoding library. `libx264` is recommended.
-b:v	The video bitrate. For example, `1500k` means 1,500 Kbps.
-r	The video frame rate.
-profile:v	The video profile. If you set it to `baseline` , B-frames will not be encoded. The TRTC backend does not support B-frames.
-g	The GOP (keyframe interval).
-c:a	The audio encoding library. `libfdk_aac` is recommended.
-ac	The number of sound channels. Valid values: `1` , `2` .
-b:a	The audio bitrate.
-f	The container format. Set it to `flv` . The FLV container format is required to publish to TRTC.

Below is an example of reading a local file and publishing it to TRTC (note that quotation marks are required for the URL):

```
ffmpeg -loglevel debug -re -i sample.flv -c:v libx264 -preset ultrafast -
profile:v baseline -g 30 -sc_threshold 0 -b:v 1500k -c:a libfdk_aac -ac 2 -b:a
128k -f flv 'rtmp://intl-rtmp.rtc.qq.com/push/hello-string-room?
userid=rtmpForFfmpeg&sdkappid=140xxxxxx&usersig=xxxxxxxxxx'
```

Playback on other clients

The screenshot below is an example of playback on the Web. You can also play the stream on other clients.



```
  Metadata:
    encoder         : Lavf58.77.100
  Stream #0:0: Video: h264 ([7][0][0][0] / 0x0007), yuv420p(tv, bt709, progressive), 1280x720
[SAR 1:1 DAR 16:9], q=2-31, 1500 kb/s, 30 fps, 1k tbn
    Metadata:
      encoder         : Lavc58.135.100 libx264
    Side data:
      cpb: bitrate max/min/avg: 0/0/1500000 buffer size: 0 vbv_delay: N/A
  Stream #0:1: Audio: aac ([10][0][0][0] / 0x000A), 48000 Hz, stereo, s16, 128 kb/s
    Metadata:
      encoder         : Lavc58.135.100 libfdk_aac
frame=  639 fps= 29 q=24.0 size=    4066kB time=00:00:21.61 bitrate=1541.3kbits/s speed=0.996x
frame=  654 fps= 29 q=24.0 size=    4193kB time=00:00:22.12 bitrate=1552.6kbits/s speed=0.997x
frame=  669 fps= 29 q=24.0 size=    4279kB time=00:00:22.61 bitrate=1550.1kbits/s speed=0.996x
frame=  684 fps= 29 q=24.0 size=    4425kB time=00:00:23.14 bitrate=1566.0kbits/s speed=0.997x
frame=  699 fps= 29 q=24.0 size=    4491kB time=00:00:23.65 bitrate=1555.0kbits/s speed=0.997x
frame=  715 fps= 30 q=24.0 size=    4631kB time=00:00:24.14 bitrate=1570.9kbits/s speed=0.997x
frame=  729 fps= 29 q=24.0 size=    4691kB time=00:00:24.66 bitrate=1558.3kbits/s speed=0.998x
frame=  745 fps= 30 q=24.0 size=    4837kB time=00:00:25.15 bitrate=1575.6kbits/s speed=0.997x
frame=  760 fps= 30 q=25.0 size=    4905kB time=00:00:25.64 bitrate=1566.9kbits/s speed=0.997x
frame=  775 fps= 30 q=24.0 size=    5048kB time=00:00:26.17 bitrate=1579.8kbits/s speed=0.998x
```

FAQ

Streaming failed

Common causes

No package purchased or expired.

Incorrect or expired signature.

Streaming with B-frames (appears as "stream ends after one second" on the dashboard); should set to baseline encoding.

Other causes

If the stream is pushed by an embedded hardware device, the URL may be truncated.

Streaming with H.265; should use H.264.

Setting chunk size too large on the client; recommend setting chunk size to 1360.

Lag and screen artifacts

Check the Tencent Cloud Real-Time Communication dashboard to monitor streaming frame rate stability. If stable, the issue is likely on the player's side - please investigate the player. **If the frame rate is unstable**, consider the following:

Check if the streaming client's local CPU and memory are under high load. If using OBS for streaming, observe the status bar at the bottom for information on dropped frames, network, CPU, and frame rate.

Check if the local **network bandwidth** is sufficient. Ping the streaming domain to observe RTT; use the network diagnostic tool to test the streaming domain and check bandwidth, ideally reaching 10M.

The streaming client may try reducing bitrate and frame rate to lessen client load, refer to the OBS settings in the main text, recommend setting bitrate to 1500 Kbps for 720p.

High Latency

If the client pulling stream uses TRTC anchor role, latency is usually lower than that of TRTC audience role. If not anchor role, change to that and compare to see if there is improvement.
Local encoding and network significantly affect the streaming end. Try different platforms for testing; if using OBS, consider streaming on a Windows system; ping the streaming domain to observe RTT.

Stream not visible on other ends

The streaming end used a string room number, but the pulling end used a numerical room number; modify the pulling end to use a string room number to enter the room.

Frequent disconnections and re-streaming

Username duplication, causing mutual kicking out; ensure that the userid is globally unique under a single sdkappid.
Streaming with B-frames; set to baseline encoding.

Server callback

An RTMP streaming user is also a user in the TRTC room, with no fundamental difference from other end users, see event callback.

Using your own domain

Configure your own domain CNAME to official domain - recommended for future use.

Speech-to-Text

Last updated：2025-04-01 14:33:13

Use Cases

Tencent Real-Time Communication (TRTC) supports the **speech-to-text feature**, which converts the audio streams of specified users or all users in a room into corresponding Chinese text for effects such as real-time captions.

Prerequisites

Log in to the TRTC console, activate the TRTC service, and create an RTC-Engine application.
Go to the purchase page to buy an RTC-Engine package of any version to unlock the speech-to-text feature.
Note:
The speech-to-text feature incurs fees based on usage. See Fee Details for more information.

Feature Overview

After a task is initiated, TRTC AI Service uses an Automatic Speech Recognition (ASR) bot to enter a TRTC room to pull the streams of specified users or all users for speech-to-text recognition, and then relay the recognition results to the client and server in real time.

Integration Guide

Step 1: Receiving Speech-to-Text Results

Method 1: Receiving Text Messages via Client SDK

Use the custom message receiving feature of the TRTC SDK to listen to callbacks on the client and receive real-time speech-to-text result data.
The client callback message format is as follows, taking the web end as an example:

```
trtc.on(TRTC.EVENT.CUSTOM_MESSAGE, event => { // Receive custom messages.
   // event.userId: The userId of the ASR robot.
   // event.cmdId: The message ID, which is fixed at 1 for transcriptions and capti
   // event.seq: The sequence number of a message.
   // event.data: ArrayBuffer type. For content of transcriptions or captions, see
```

```
    const data = new TextDecoder().decode(event.data)
    // Explanation of the data field is as follows.
    console.log(`received custom msg from ${event.userId}, message: ${ data }`)
})
```

Data field explanation

Real-Time Captions

Field Name	Type	Meaning
type	Integer	10000: When there are real-time captions and a complete sentence, the message type will be delivered.
sender	String	Speaker's userid.
receiver	Array	Recipient's userid list. This message is actually broadcast within a room.
payload.text	String	Recognized text, Unicode encoded.
payload.start_time	String	Message start time. It is the absolute time after a task starts.
payload.end_time	String	Message end time. It is the absolute time after a task starts.
payload.end	Boolean	If true, it indicates that this is a complete sentence.

```
{
  "type": 10000,
  "sender": "user_a",
  "payload": {
     "text":"",
     "start_time":"00:00:02",
     "end_time":"00:00:05",
     "end": true
  }
}
```

Note:

Callback example explanation:

Transcription: A complete sentence will be transcribed and pushed.

"How's the weather today?"

Captions: A sentence will be segmented for pushing, with each subsequent segment containing the previous one to ensure real-time performance.

"Today"

"Today's weather"

"How's the weather today?"

Sequence explanation: Caption message > Caption message > .... > Caption message (end = true)

Method 2: Receiving via Server-side Callbacks

The speech-to-text service also provides server-side event callbacks, facilitating your service to receive real-time conversation messages. See Detailed Callback Events.

Step 2: Initiating a Speech-to-Text Task

TRTC provides the following Tencent Cloud APIs for initiating and managing speech-to-text tasks:

Start a speech-to-text task: StartAITranscription

Query a speech-to-text task: DescribeAITranscription

Stop a speech-to-text task: StopAITranscription

Note:

The speech-to-text feature has a concurrency limit of **100 tasks** per SDKAppId. Submit a ticket if you need to increase this limit.

Utilizing Beautification Effects SDK Integration Guide (Flutter)

Last updated：2025-04-10 11:38:07

Step 1: Seamless Integration of Tencent Special Effect Resources

1. Download Demo project.
2. Migrate Tencent special effect resources

Android

iOS

1. Find the `src/main/assets` folder in the `android/app` module of your project. Copy the lut and MotionRes folders from `demo/android/app/src/main/assets` in the demo project to `android/app/src/main/assets` in your project. If your project does not have an assets folder, you can manually create one.

2. In the `android/app/build.gradle` file of your project, add the dependency of the Beauty SDK for the Android side. The specific dependency depends on the package you selected. For example, if you selected the S1-04 package, add the following:

```
dependencies {
   implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

Note:

The maven urls corresponding to each package, please see Documentation. The latest version number of the SDK can be viewed in Version History.

3. If you use an Android version of beauty SDK less than 3.9, you need to find the AndroidManifest.xml file under the app module and add the following tags in the application table:

```
    <uses-native-library
            android:name="libOpenCL.so"
            android:required="false" />
        //true indicates that libOpenCL is necessary for the current app. If there
        //false indicates that libOpenCL is not necessary for the current app. The
        //For the description of uses-native-library, please refer to the Android o
```

As shown below after addition:

4. Obfuscation configuration.
When building a release package with code optimization and obfuscation features enabled (minifyEnabled = true), the compilation tool may remove code that is not explicitly called at the Java/Kotlin layer. If this code is dynamically called by the native layer, a NoSuchMethodError exception (such as no xxx method) will be triggered.
It is recommended to proactively retain the necessary code of the Xmagic module through ProGuard rules in proguard-rules.pro:

```
-keep class com.tencent.xmagic.** { *;}
-keep class org.light.** { *;}
-keep class org.libpag.** { *;}
-keep class org.extra.** { *;}
-keep class com.gyailib.**{ *;}
-keep class com.tencent.cloud.iai.lib.** { *;}
-keep class com.tencent.beacon.** { *;}
-keep class com.tencent.qimei.** { *;}
-keep class androidx.exifinterface.** { *;}
-keep class com.tencent.effect.** { *;}
```

Copy the xmagic folder under the ios/Runner directory in the demo project to the ios/Runner directory in your project. After addition, it is as shown in the figure below:

Note:
The above materials copied from the demo project are test materials. For official materials, you need to contact the staff of Tencent Special Effect Beauty after purchasing a package to obtain and re-add them.

Step Two: Integration of tencent_effect_flutter

You can depend on tencent_effect_flutter in your Flutter project in the following ways.
1. Remote dependency
Add the following reference in your pubspec.yaml file:

```
  tencent_effect_flutter:
   git:
     url: https://github.com/Tencent-RTC/TencentEffect_Flutter
```

2. Local dependency
Download the latest version of tencent_effect_flutter from github, and then add the following reference in the pubspec.yaml file:

```
tencent_effect_flutter:
    path: path to tencent_effect_flutter
```

Step 3: Beauty Effect and TRTC Association

Android

iOS

Add the following code in the onCreate method of the application class (or the onCreate method of the FlutterActivity):

```
TRTCPlugin.setBeautyProcesserFactory(new XmagicProcesserFactory());
```

Add the following code in the didFinishLaunchingWithOptions method in the AppDelegate file under the ios/Runner directory:

Swift

Object-C

```
let instance = XmagicProcesserFactory()
TencentRTCCloud.setBeautyProcesserFactory(factory: instance)
```

```
XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc] init];
[TencentRTCCloud setBeautyProcesserFactoryWithFactory:instance];
```

Step 4: Beauty Effect Resource Initialization and Authorization

1. Resource initialization

V0.3.5.0 and Later

V0.3.1.1 and Earlier Versions

```
TencentEffectApi.getApi()?.setResourcePath(resourceDir);

TencentEffectApi.getApi()?.initXmagic((result) {
    // TODO
});
```

```
TencentEffectApi.getApi()?.initXmagic(dir,(reslut) {
    //TODO
});
```

2. Beauty authorization

```
TencentEffectApi.getApi()?.setLicense(licenseKey, licenseUrl, (errorCode, msg) {
    if (errorCode == 0) {
       // Success
    }
});
```

Step 5: Enable/Disable Beauty Effects

After completing the above operations, you can enable/disable beauty effects through TRTC's hidden interface:

```
_enableCustomBeautyByNative(bool open) {
  trtcCloud.callExperimentalAPI("{\\"api\\": \\"enableVideoProcessByNative\\", \\"p
}
```

Note:

Enable beauty effects on the page. When closing the camera, you need to disable beauty effects first. Enable and disable are used in pairs.

Document Reference

You have completed the association between TRTC and Special Effect Beauty Enhancement. You can learn more about how to use Special Effect Beauty Enhancement through the following document:

Beauty Flutter SDK API Doc

Effects Parameter

Testing Hardware Devices Android&iOS&Windows&Mac

Last updated：2023-09-28 11:53:14

Overview

Given that it is difficult for users to detect device problems during a call, we recommend that you test devices such as cameras and mics before a video call.

Supported Platforms

iOS	Android	macOS	Windows	Electron	Web
×	×	✓	✓	✓	✓ (Web)

Testing Camera

You can use the `startCameraDeviceTestInView` API of `TRTCCloud` to test a camera and can use the `setCurrentCameraDevice` API to switch cameras during testing.

macOS

Windows (C++)

Windows (C#)

```
// Open the camera testing page, on which you can preview camera images and switch
- (IBAction)startCameraTest:(id)sender {
    // Start camera testing. `cameraPreview` is `NSView` on macOS or `UIView` on iO
    [self.trtcCloud startCameraDeviceTestInView:self.cameraPreview];
}

// Close the camera testing page.
- (void)windowWillClose:(NSNotification *)notification{
    // Stop camera testing.
    [self.trtcCloud stopCameraDeviceTest];
}
```

```
// Start camera testing. Pass in the control handle that renders video.
void TRTCMainViewController::startTestCameraDevice(HWND hwnd)
{
     trtcCloud->startCameraDeviceTest(hwnd);
}

// Stop camera testing.
void TRTCMainViewController::stopTestCameraDevice()
{
     trtcCloud->stopCameraDeviceTest();
}
```

```
// Start camera testing. Pass in the control handle that renders video.
private void startTestCameraDevice(Intptr hwnd)
{
     mTRTCCloud.startCameraDeviceTest(hwnd);
}

// Stop camera testing.
private void stopTestCameraDevice()
{
     mTRTCCloud.stopCameraDeviceTest();
}
```

Testing Mic

You can use the `startMicDeviceTest` API of `TRTCCloud` to measure mic volume in real time. The result is returned via a callback.

macOS

Windows (C++)

Windows (C#)

```
   // Sample code for mic testing
   -(IBAction)micTest:(id)sender {
     NSButton *btn = (NSButton *)sender;
     if (btn.state == 1) {
             // Start mic testing.
         __weak __typeof(self) wself = self;
         [self.trtcCloud startMicDeviceTest:500  testEcho:^(NSInteger volume) {
             dispatch_async(dispatch_get_main_queue(), ^{
```

```
                                // Refresh the mic volume bar.
                    [wself _updateInputVolume:volume];
                });
            }];
            btn.title = @"Stop test";
        }
        else{
                // Stop mic testing.
            [self.trtcCloud stopMicDeviceTest];
            [self _updateInputVolume:0];
            btn.title = @"Start test";
        }
    }
```

```
// Sample code for mic testing
void TRTCMainViewController::startTestMicDevice()
{
    // Set the interval for triggering the volume callback and listen for the `onTe
    uint32_t interval = 500;
    // Start mic testing.
    trtcCloud->startMicDeviceTest(interval);
}

// Stop mic testing.
void TRTCMainViewController::stopTestMicDevice()
{
     trtcCloud->stopMicDeviceTest();
}
```

```
// Sample code for mic testing
private void startTestMicDevice()
{
    // Set the interval for triggering the volume callback and listen for the `onTe
    uint interval = 500;
    // Start mic testing.
    mTRTCCloud.startMicDeviceTest(interval);
}

// Stop mic testing.
private void stopTestMicDevice()
{
     mTRTCCloud.stopMicDeviceTest();
}
```

Testing Speaker

You can use the `startSpeakerDeviceTest` API of `TRTCCloud` to test whether a speaker works properly by playing a default MP3 file.

macOS

Windows (C++)

Windows (C#)

```
// Sample code for speaker testing
// Take an NSButton for example. In `xib`, set the title of the button in the on an
- (IBAction)speakerTest:(NSButton *)btn {
    NSString *path = [[NSBundle mainBundle] pathForResource:@"test-32000-mono" ofTy
    if (btn.state == NSControlStateValueOn) {
        // Click "Start Test".
        __weak __typeof(self) wself = self;
        [self.trtcEngine startSpeakerDeviceTest:path onVolumeChanged:^(NSInteger vo
            // The subsequent steps involve the UI and need to be executed in the m
            dispatch_async(dispatch_get_main_queue(), ^{
                // `_updateOutputVolume` means updating the speaker volume indicato
                [wself _updateOutputVolume:volume];
                if (playFinished) {
                    // Set the button status to "Start Test" after playback is comp
                    sender.state = NSControlStateValueOff;
                }
            });
        }];
    } else {
        // Click "Stop Test".
        [self.trtcEngine stopSpeakerDeviceTest];
        [self _updateOutputVolume:0];
    }
}

// Update the speaker volume indicator.
- (void)_updateOutputVolume:(NSInteger)volume {
    // `speakerVolumeMeter` is `NSLevelIndicator`.
    self.speakerVolumeMeter.doubleValue = volume / 255.0 * 10;
}
```

```
// Sample code for speaker testing
void TRTCMainViewController::startTestSpeakerDevice(std::string testAudioFilePath)
```

```
{
    // `testAudioFilePath` is the absolute path of the audio file (in WAV or MP3 fo
    // Listen for the `onTestSpeakerVolume` callback to get the speaker volume.
    trtcCloud->startSpeakerDeviceTest(testAudioFilePath.c_str());
}

// Stop speaker testing.
void TRTCMainViewController::stopTestSpeakerDevice() {
    trtcCloud->stopSpeakerDeviceTest();
}
```

```
// Sample code for speaker testing
private void startTestSpeakerDevice(string testAudioFilePath)
{
    // `testAudioFilePath` is the absolute path of the audio file (in WAV or MP3 fo
    // Listen for the `onTestSpeakerVolume` callback to get the speaker volume.
    mTRTCCloud.startSpeakerDeviceTest(testAudioFilePath);
}

// Stop speaker testing.
private void stopTestSpeakerDevice() {
    mTRTCCloud.stopSpeakerDeviceTest();
}
```

Web

Last updated：2023-11-16 15:09:42

Overview

Because it is difficult for users to detect device problems during a call, we recommend checking the browser and testing devices such as cameras and mics before starting a video call.

Browser Environment Check

Before enterRoom, we recommend you use the TRTC.isSupported API to check whether the SDK supports the current browser first, and if not, please recommend the user to use a supported browser, refer to Supported browsers.

```
TRTC.isSupported().then(checkResult => {
  // Not supported, guide the user to use a supported browser(Chrome 56+, Edge 80+,
  if (!checkResult.result) {}
  // Not support to publish video
  if (!checkResult.detail.isH264EncodeSupported) {}
  // Not support to subscribe video
  if (!checkResult.detail.isH264DecodeSupported) {}
})
```

If the check result returned by TRTC.isSupported is false, this may be because:

1. The web page uses the http protocol. Browsers do not allow http protocol sites to capture cameras and microphones, you need to deploy your page using the https protocol.
2. The current browser does not support WebRTC, you need to guide the user to use the recommended browser, refer to Supported browsers.
3. Firefox browser needs to load H264 codec dynamically after installation, so the detection result will be false for a short period of time, please wait and try again or guide to use other browsers.

Audio/Video Device Test

To ensure that users can have a good user experience with the TRTC SDK, we recommend you check the user's device and network conditions and provide troubleshooting suggestions before the user enters a TRTC room.
You can quickly integrate the device and network check features by referring to the following methods:

`rtc-detect` Library
React Component for Device Check
TRTC Capability Check Page

`rtc-detect` Library

You can use rtc-detect to check whether the current environment is supported by the TRTC SDK and view the details of the current environment.

Installation

```
npm install rtc-detect
```

Usage

```
import RTCDetect from 'rtc-detect';
// Initialize the detection module
const detect = new RTCDetect();
// Get the detection result of the current environment
const result = await detect.getReportAsync();
// `result` contains the current environment system information, API support, codec
console.log('result is: ' + result);
```

API

(async) isTRTCSupported()

This API is used to check whether the current environment supports TRTC.

```
const detect = new RTCDetect();
const data = await detect.isTRTCSupported();

if (data.result) {
  console.log('current browser supports TRTC.')
} else {
  console.log(`current browser does not support TRTC, reason: ${data.reason}.`)
}
```

getSystem()

This API is used to get the current system environment parameters.

Item	Type	Description

UA	string	The browser user-agent.
OS	string	The OS of the current device.
browser	object	The current browser information in the format of `{ name, version }`.
displayResolution	object	The current resolution in the format of `{ width, height }`.
getHardwareConcurrency	number	The number of CPU cores of the current device.

```
const detect = new RTCDetect();
const result = detect.getSystem();
```

getAPISupported()

This API is used to get the API support of the current environment.

Item	Type	Description
isUserMediaSupported	boolean	Whether the user media data stream can be obtained
isWebRTCSupported	boolean	Whether WebRTC is supported
isWebSocketSupported	boolean	Whether WebSocket is supported
isWebAudioSupported	boolean	Whether WebAudio is supported
isScreenCaptureAPISupported	boolean	Whether the screen stream can be obtained
isCanvasCapturingSupported	boolean	Whether the data stream can be obtained from the canvas
isVideoCapturingSupported	boolean	Whether the data stream can be obtained from the video
isRTPSenderReplaceTracksSupported	boolean	Whether renegotiation with `peerConnection` can be skipped when `track` is replaced
isApplyConstraintsSupported	boolean	Whether the camera resolution can be changed without calling `getUserMedia` again

```
const detect = new RTCDetect();
const result = detect.getAPISupported();
```

(async) getDevicesAsync()

This API is used to get the available devices in the current environment.

Item	Type	Description
hasWebCamPermissions	boolean	Whether the user camera data can be obtained
hasMicrophonePermission	boolean	Whether the user mic data can be obtained
cameras	array	List of user cameras, including their resolutions, maximum width, maximum height, and maximum frame rate (for certain browsers only) supported for video streams
microphones	array	List of mics used by users
speakers	array	List of speakers used by users

CameraItem

Item	Type	Description
deviceId	string	The device ID, which is usually unique and can be used to identify devices.
groupId	string	The group ID. If two devices belong to the same physical device, they have the same group ID.
kind	string	The camera device type: 'videoinput'.
label	string	A tag which describes the device.
resolution	object	The maximum resolution width, height, and frame rate supported by the camera {maxWidth: 1280, maxHeight: 720, maxFrameRate: 30}.

DeviceItem

Item	Type	Description
deviceId	string	The device ID, which is usually unique and can be used to identify devices.
groupId	string	The group ID. If two devices belong to the same physical device, they have the same group ID.
kind	string	The device type, such as 'audioinput' and 'audiooutput'.
label	string	A tag which describes the device.

```
const detect = new RTCDetect();
const result = await detect.getDevicesAsync();
```

(async) getCodecAsync()

This API is used to get the codec support of the current environment.

Item	Type	Description
isH264EncodeSupported	boolean	Whether H.264 encoding is supported
isH264DecodeSupported	boolean	Whether H.264 decoding is supported
isVp8EncodeSupported	boolean	Whether VP8 encoding is supported
isVp8DecodeSupported	boolean	Whether VP8 decoding is supported

If encoding is supported, audio/video can be published. If decoding is supported, audio/video can be pulled for playback.

```
const detect = new RTCDetect();
const result = await detect.getCodecAsync();
```

(async) getReportAsync()

This API is used to get the detection report of the current environment.

Item	Type	Description
system	object	Same as the returned value of `getSystem()`
APISupported	object	Same as the returned value of `getAPISupported()`
codecsSupported	object	Same as the returned value of `getCodecAsync()`
devices	object	Same as the returned value of `getDevicesAsync()`

```
const detect = new RTCDetect();
const result = await detect.getReportAsync();
```

(async) isHardWareAccelerationEnabled()

This API is used to check whether hardware acceleration is enabled on the Chrome browser.

notice

The implementation of this API depends on the native WebRTC API. We recommend you call this API for check after calling `isTRTCSupported` . The check can take up to 30 seconds as tested below:
1. If hardware acceleration is enabled, this API will take about 2 seconds on Windows and 10 seconds on macOS.
2. If hardware acceleration is disabled, this API will take about 30 seconds on both Windows and macOS.

```
const detect = new RTCDetect();
const data = await detect.isTRTCSupported();

if (data.result) {
  const result = await detect.isHardWareAccelerationEnabled();
  console.log(`is hardware acceleration enabled: ${result}`);
} else {
  console.log(`current browser does not support TRTC, reason: ${data.reason}.`)
}
```

React Component for Device Check

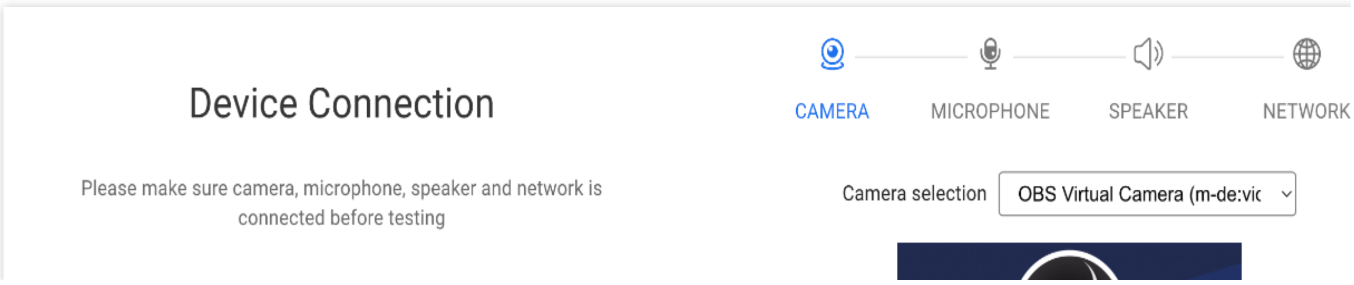
Device check UI component features

1. Device connection and check logic processing
2. Network check logic processing
3. Optional network check tab
4. Support for Chinese and English

Device check UI component links

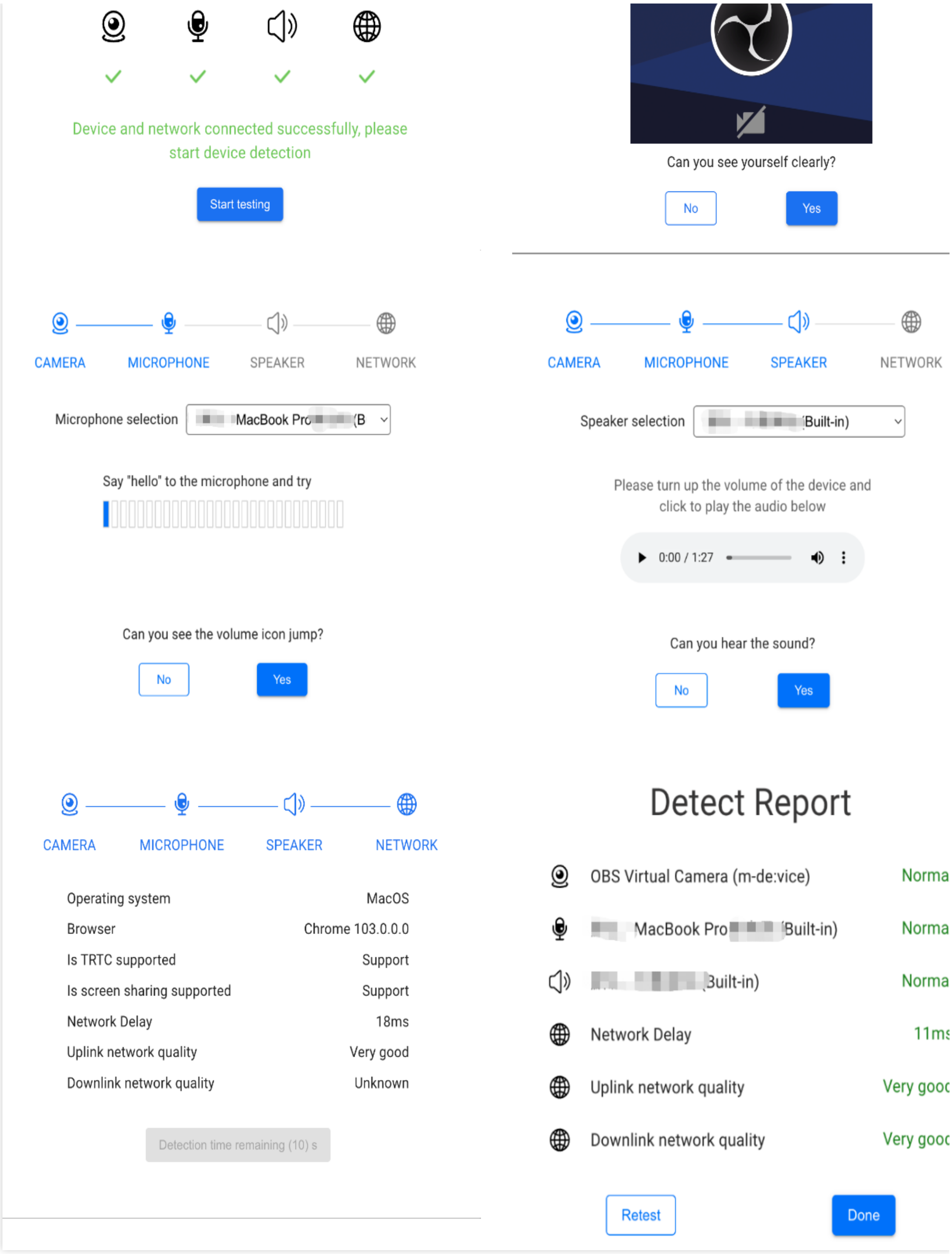
For more information on how to use the component's npm package, see rtc-device-detector-react.
For more information on how to debug the component's source code, see github/rtc-device-detector.
For more information on how to import the component, see WebRTC API Example.

Device check UI component page


Device Connection

Please make sure camera, microphone, speaker and network is connected before testing

Camera selection: OBS Virtual Camera (m-de:vic



Device and network connected successfully, please start device detection

Start testing

Can you see yourself clearly?

No Yes

Microphone selection [redacted] MacBook Pro [redacted] (B

Say "hello" to the microphone and try

Can you see the volume icon jump?

No Yes

Speaker selection [redacted] Built-in)

Please turn up the volume of the device and click to play the audio below

0:00 / 1:27

Can you hear the sound?

No Yes

Operating system	MacOS
Browser	Chrome 103.0.0.0
Is TRTC supported	Support
Is screen sharing supported	Support
Network Delay	18ms
Uplink network quality	Very good
Downlink network quality	Unknown

Detection time remaining (10) s

Detect Report

OBS Virtual Camera (m-de:vice)	Norma
[redacted] MacBook Pro [redacted] Built-in)	Norma
[redacted] Built-in)	Norma
Network Delay	11ms
Uplink network quality	Very gooc
Downlink network quality	Very gooc

Retest Done

Device and Network Detection Logic

1) Device Connection

The purpose of device connection is to detect whether the user's machine has camera, microphone, and speaker devices, and whether it is in a networked state. If there are camera and microphone devices, try to obtain audio and video streams and guide the user to grant access to the camera and microphone.

Determine whether the device has camera, microphone, and speaker devices

```
import TRTC from 'trtc-sdk-v5';

const cameraList = await TRTC.getCameraList();
const micList = await TRTC.getMicrophoneList();
const speakerList = await TRTC.getSpeakerList();
const hasCameraDevice = cameraList.length > 0;
const hasMicrophoneDevice = micList.length > 0;
const hasSpeakerDevice = speakerList.length > 0;
```

Obtain access to the camera and microphone

```
await trtc.startLocalVideo({ publish: false });
await trtc.startLocalAudio({ publish: false });
```

Check whether the device is connected to the network

```
export function isOnline() {
  const url = 'https://web.sdk.qcloud.com/trtc/webrtc/assets/trtc-logo.png';
  return new Promise((resolve) => {
    try {
      const xhr = new XMLHttpRequest();
      xhr.onload = function () {
        resolve(true);
      };
      xhr.onerror = function () {
        resolve(false);
      };
      xhr.open('GET', url, true);
      xhr.send();
    } catch (err) {
      // console.log(err);
    }
  });
}
const isOnline = await isOnline();
```

2） Camera Detection

Detection principle: Open the camera and render the camera image on the page.

Open the camera

```
trtc.startLocalVideo({ view: 'camera-video', publish: false });
```

Switch camera

```
trtc.updateLocalVideo({
  option: { cameraId }
});
```

Device plug and unplug detection

Close the camera after the detection is complete

```
trtc.stopLocalVideo();
```

3) Microphone Detection

Detection principle: Open the microphone and obtain the microphone volume.

Open the microphone

```
trtc.startLocalAudio({ publish: false });
```

Switch microphone

```
trtc.updateLocalAudio({ option: { microphoneId }});
```

Device plug and unplug detection

Release microphone usage after detection is complete

```
trtc.stopLocalAudio();
```

4) Speaker Detection

Detection principle: Play an mp3 media file through the audio tag.

Create an audio tag to remind the user to turn up the volume and play the mp3 to confirm whether the speaker device is working properly.

```
<audio id="audio-player" src="xxxxx" controls></audio>
```

Stop playing after detection is complete

```
const audioPlayer = document.getElementById('audio-player');
if (!audioPlayer.paused) {
    audioPlayer.pause();
}
```

```
audioPlayer.currentTime = 0;
```

5) Network Detection

Reference: Network Quality Detection Before Call

TRTC Capability Detection Page

You can use the TRTC Detection Page where you are currently using the TRTC SDK to detect the current environment. You can also click the "Generate Report" button to get a report of the current environment for environment detection or troubleshooting.

Testing Network Quality Android&iOS&Windows&Mac

Last updated：2024-06-07 22:18:34

It is difficult for ordinary users to measure network quality. Before calls are made, we recommend that you test the network speed to get more accurate results on network quality.

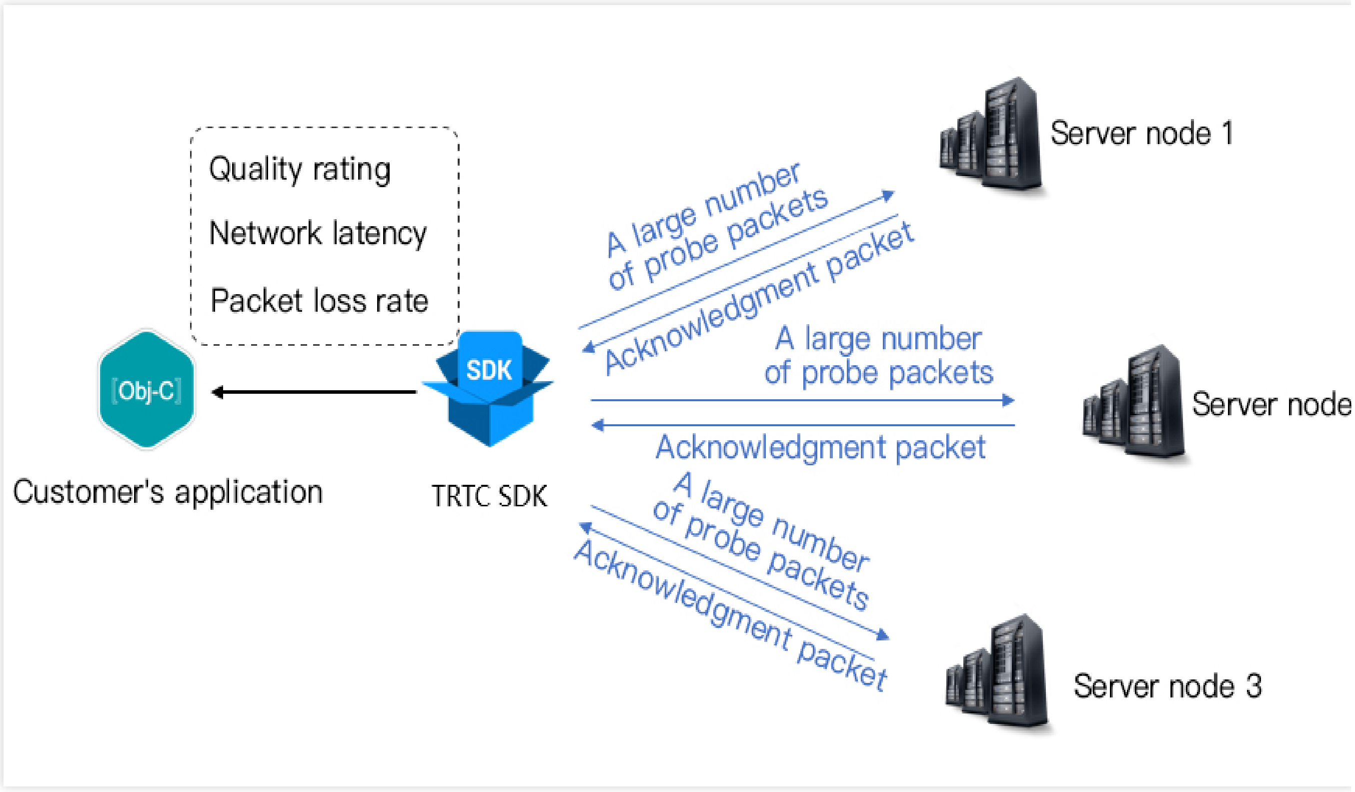
Notes

To ensure call quality, do not run the test during a video call.
Speed testing consumes traffic and consequently generates a small traffic fee (almost negligible).

Supported Platforms

iOS	Android	macOS	Windows	Electron	Web
✓	✓	✓	✓	✓	✓ (Reference: Detect Network Quality)

How Speed Testing Works



During speed testing, the SDK sends a batch of probe packets to the server node, measures the quality of return packets, and returns the testing result via a callback API.

The testing result can be used to optimize the SDK's server selection policy, so you are advised to run the test before the first call, which will help the SDK select the optimal server. If the result is unsatisfactory, you can show a UI message asking users to change to a better network.

The test result (TRTCSpeedTestResult) includes the following parameters:

Parameter	Type	Description
success	Success result	Whether the test is successful.
errMsg	Error message	Error message of bandwidth test.
ip	Server address	Testing server IP
quality	Network quality score	Network quality measured by the SDK. Lower packet loss and shorter RTT result in a higher network quality score.
upLostRate	Upstream packet loss rate	Value range: 0-1.0. `0.3` indicates that for every 10 data packets sent to the server, 3 may be lost.

downLostRate	Downstream packet loss rate	Value range: 0-1.0. `0.2` indicates that for every 10 data packets received from the server, 2 may be lost.
rtt	Latency	The time it takes for data to travel from the SDK to the server and back again. The shorter the RTT, the better. The normal range of RTT is 10-100 ms.
availableUpBandwidth	Upstream bandwidth	Estimated upstream bandwidth in Kbps. -1 indicates an invalid value.
availableDownBandwidth	Downstream bandwidth	Estimated downstream bandwidth in Kbps. -1 indicates an invalid value.

How to Test Speed

The speed test feature can be started through the `startSpeedTest` function of `TRTCCloud`. The speed test result will be called back through the callback function.

Objective-C

Java

C++

C#

```
// Sample code for starting speed testing. `sdkAppId` and `UserSig` are required. F
// The example below starts after login.
- (void)onLogin:(NSString *)userId userSig:(NSString *)userSid
{
    TRTCSpeedTestParams *params;
    // `sdkAppID` is the actual application ID obtained from the console.
    params.sdkAppID = sdkAppId;
    params.userID = userId;
    params.userSig = userSig;
    // Expected upstream bandwidth in Kbps. Value range: 10-5000. 0 indicates not t
    params.expectedUpBandwidth = 5000;
    // Expected downstream bandwidth in Kbps. Value range: 10-5000. 0 indicates not
    params.expectedDownBandwidth = 5000;
    [trtcCloud startSpeedTest:params];
}
- (void)onSpeedTestResult:(TRTCSpeedTestResult *)result {
    // The speed test result will be called back after the test is completed
}
```

```
// Sample code for starting speed testing. `sdkAppId` and `UserSig` are required. F
// The example below starts after login.
public void onLogin(String userId, String userSig)
{
  TRTCCloudDef.TRTCSpeedTestParams params = new TRTCCloudDef.TRTCSpeedTestParams();
  params.sdkAppId = GenerateTestUserSig.SDKAPPID;
  params.userId = mEtUserId.getText().toString();
  params.userSig = GenerateTestUserSig.genTestUserSig(params.userId);
  params.expectedUpBandwidth = Integer.parseInt(expectUpBandwidthStr);
  params.expectedDownBandwidth = Integer.parseInt(expectDownBandwidthStr);
    // `sdkAppID` is the actual application ID obtained from the console.
    trtcCloud.startSpeedTest(params);
}

// Listen for the test result. Inherit `TRTCCloudListener` and implement the follow
void onSpeedTestResult(TRTCCloudDef.TRTCSpeedTestResult result)
{
    // The speed test result will be called back after the test is completed
}
```

```
// Sample code for starting speed testing. `sdkAppId` and `UserSig` are required. F
// The example below starts after login.
void onLogin(const char* userId, const char* userSig)
{
    TRTCSpeedTestParams params;
    // `sdkAppID` is the actual application ID obtained from the console.
    params.sdkAppID = sdkAppId;
    params.userId = userid;
    param.userSig = userSig;
    // Expected upstream bandwidth in Kbps. Value range: 10-5000. 0 indicates not t
    param.expectedUpBandwidth = 5000;
    // Expected downstream bandwidth in Kbps. Value range: 10-5000. 0 indicates not
    param.expectedDownBandwidth = 5000;
    trtcCloud->startSpeedTest(params);
}

// Listen for the testing result
void TRTCCloudCallbackImpl::onSpeedTestResult(
             const TRTCSpeedTestResult& result)
{
    // The speed test result will be called back after the test is completed
}
```

```
// Sample code for starting speed testing. `sdkAppId` and `UserSig` are required. F
// The example below starts after login.
private void onLogin(string userId, string userSig)
{
    TRTCSpeedTestParams params;
    // `sdkAppID` is the actual application ID obtained from the console.
    params.sdkAppID = sdkAppId;
    params.userId = userid;
    param.userSig = userSig;
    // Expected upstream bandwidth in Kbps. Value range: 10-5000. 0 indicates not t
    param.expectedUpBandwidth = 5000;
    // Expected downstream bandwidth in Kbps. Value range: 10-5000. 0 indicates not
    param.expectedDownBandwidth = 5000;
    mTRTCCloud.startSpeedTest(params);
}

// Listen for the testing result
public void onSpeedTestResult(TRTCSpeedTestResult result)
{
    // The speed test result will be called back after the test is completed
}
```

Speed Test Tool

If you don't want to call an API to test the network speed, you can use the network speed test tool for PC provided by TRTC to quickly get the network quality details.

Download link

Mac | Windows

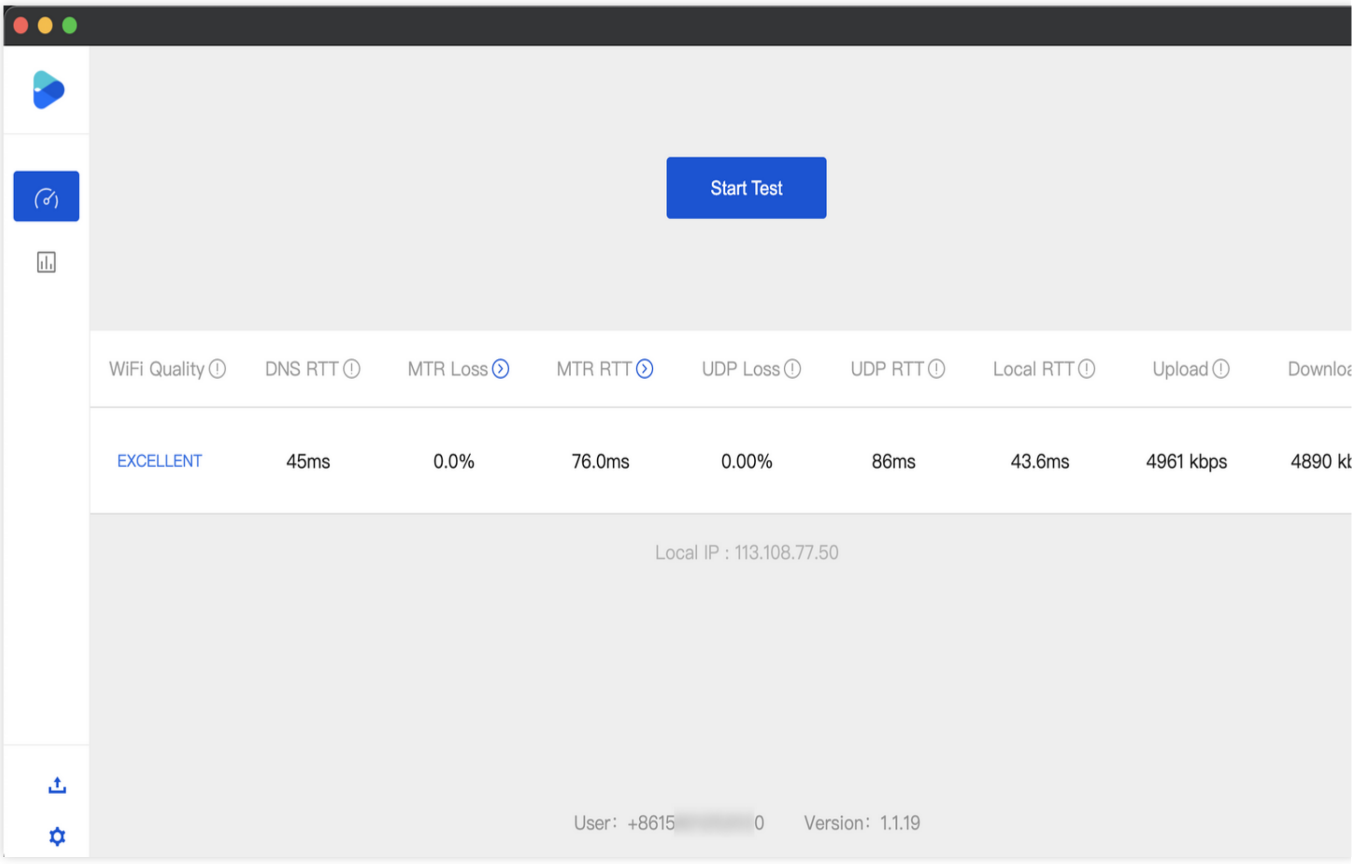
Test metrics

Metric	Description
WiFi Quality	Wi-Fi signal reception quality
DNS RTT	Tencent Cloud testing domain DNS round-trip time (RTT)
MTR	MTR is a network speed test tool, which can detect the packet loss rate and latency between client and TRTC node and display the details of each hop in the route

UDP Loss	UDP packet loss rate between client and TRTC node
UDP RTT	UDP latency between client and TRTC node
Local RTT	Latency between client and local gateway
Upload	Estimated upstream bandwidth
Download	Estimated downstream bandwidth

Tool screenshots

Quick test:

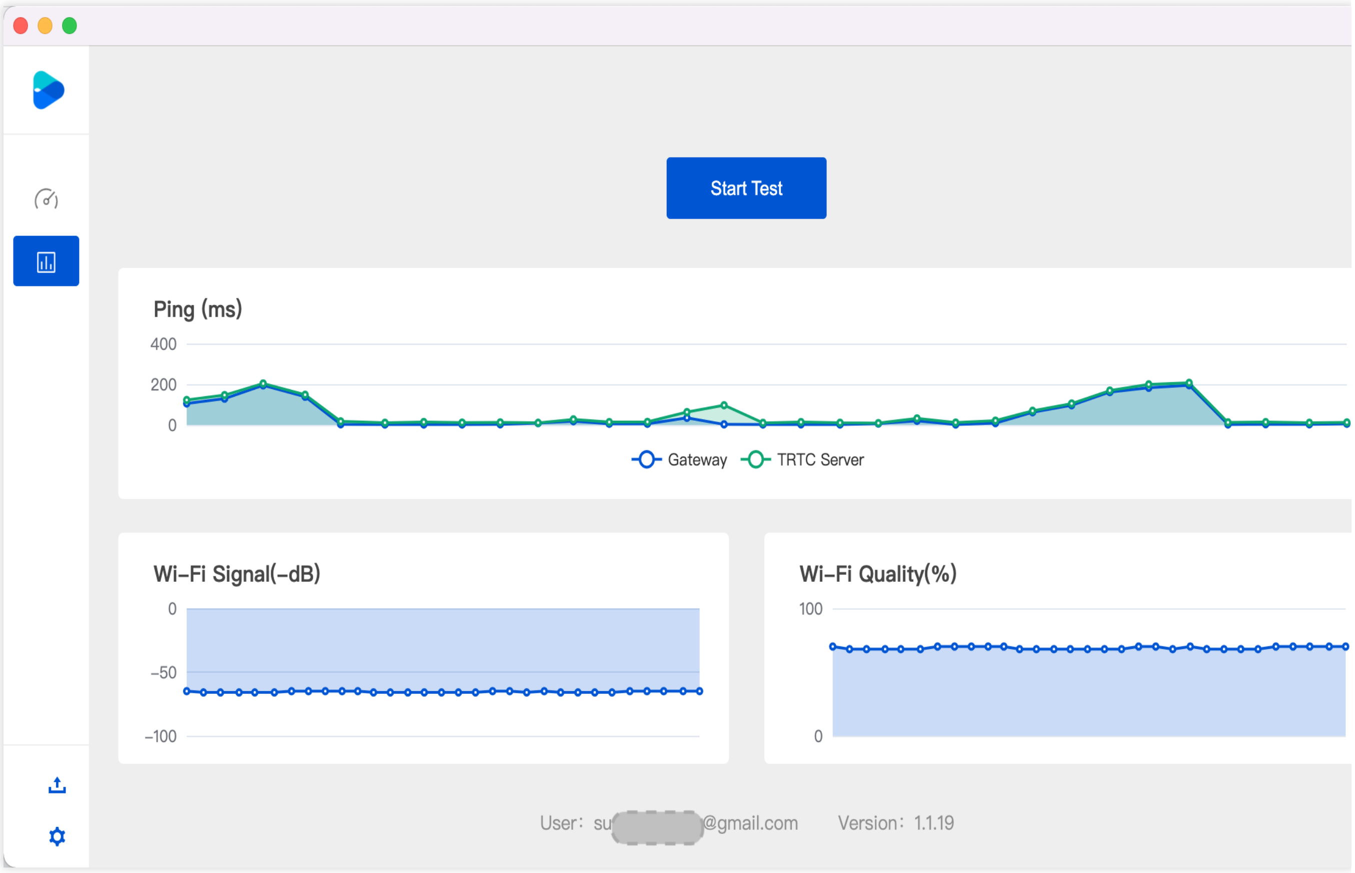


WiFi Quality	DNS RTT	MTR Loss	MTR RTT	UDP Loss	UDP RTT	Local RTT	Upload	Downloa
EXCELLENT	45ms	0.0%	76.0ms	0.00%	86ms	43.6ms	4961 kbps	4890 kt

Local IP : 113.108.77.50

User: +8615[redacted]0 Version: 1.1.19

Continuous test:



User: su[redacted]@gmail.com Version: 1.1.19

Web

Last updated：2023-09-28 11:54:53

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 based on the `NETWORK_QUALITY` event.

Network quality check during the call process

```
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}`)
})
```

Network quality check before making a call

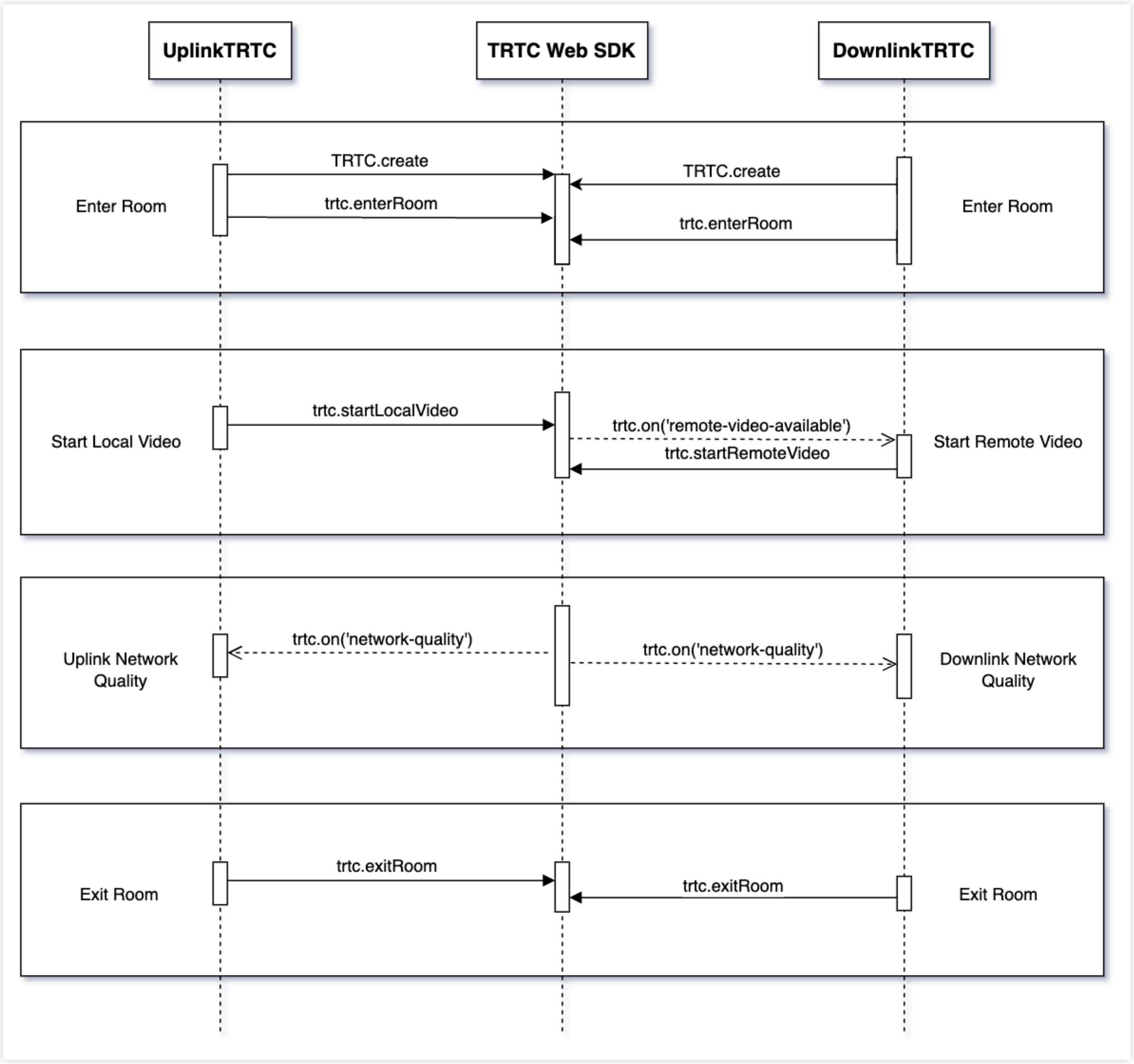
Implementation process

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.

notice

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



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 communication. <br> 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.

On-Cloud Recording

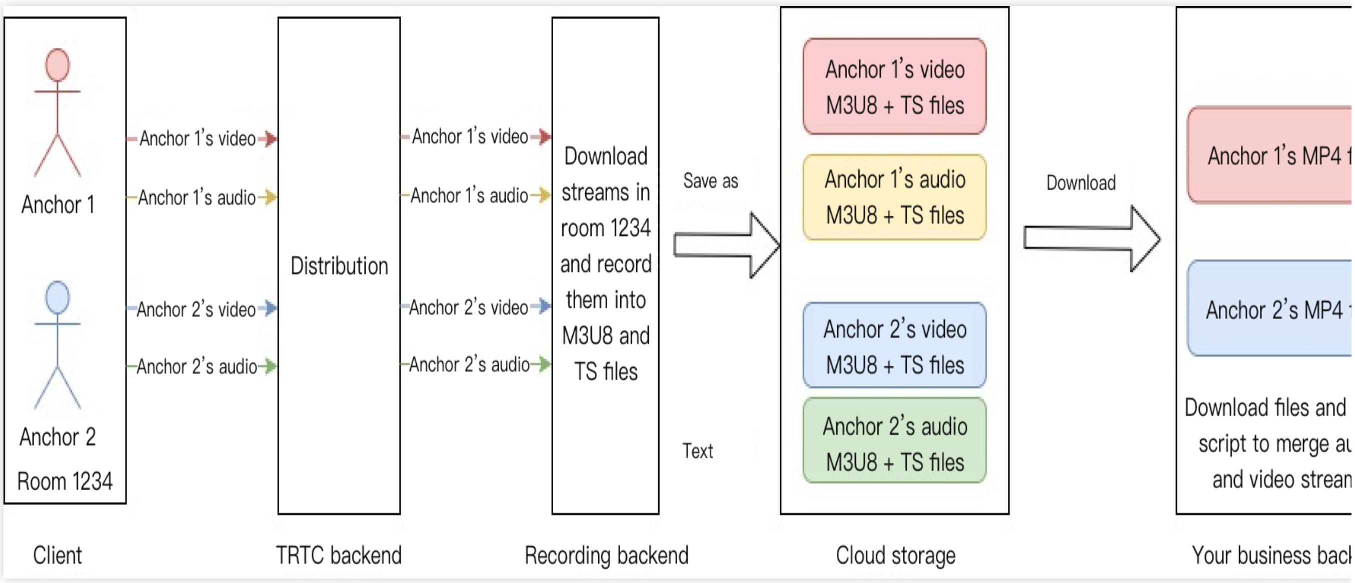
Last updated：2024-04-26 17:43:17

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

Overview

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

Single-stream recording

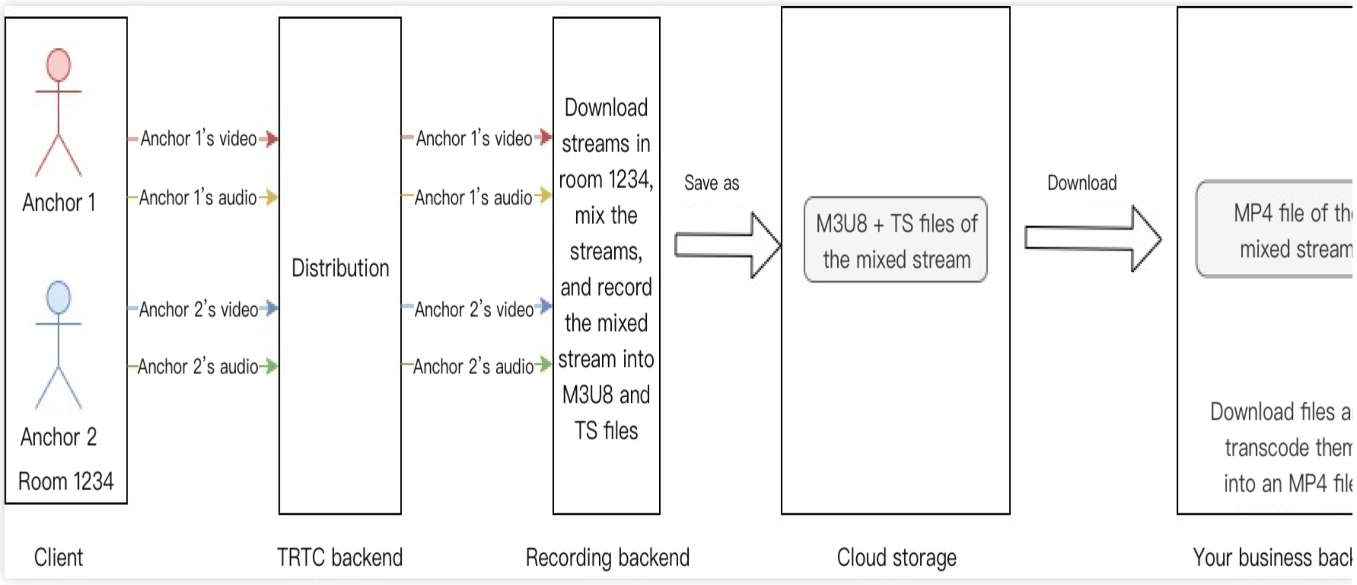


The diagram above shows the workflow of single-stream recording. In room 1234, anchor 1 and anchor 2 are publishing streams. If you subscribe to their streams and enable single-stream recording, the TRTC backend will record the audio and video data of anchor 1 and anchor 2 separately. The recording results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

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

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

Mixed-stream recording


Anchor 1
Anchor 2
Room 1234
Anchor 1's video
Anchor 1's audio
Anchor 2's video
Anchor 2's audio
Distribution
Download streams in room 1234, mix the streams, and record the mixed stream into M3U8 and TS files
Save as
M3U8 + TS files of the mixed stream
Download
MP4 file of th mixed stream
Download files a transcode them into an MP4 file
Client
TRTC backend
Recording backend
Cloud storage
Your business back

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

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

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

Note:

The rate limit for the recording API is 20 calls per second.

The timeout period for a query is six seconds.

We allow up to 100 ongoing recording tasks at the same time. If you need to record more, please submit a ticket.

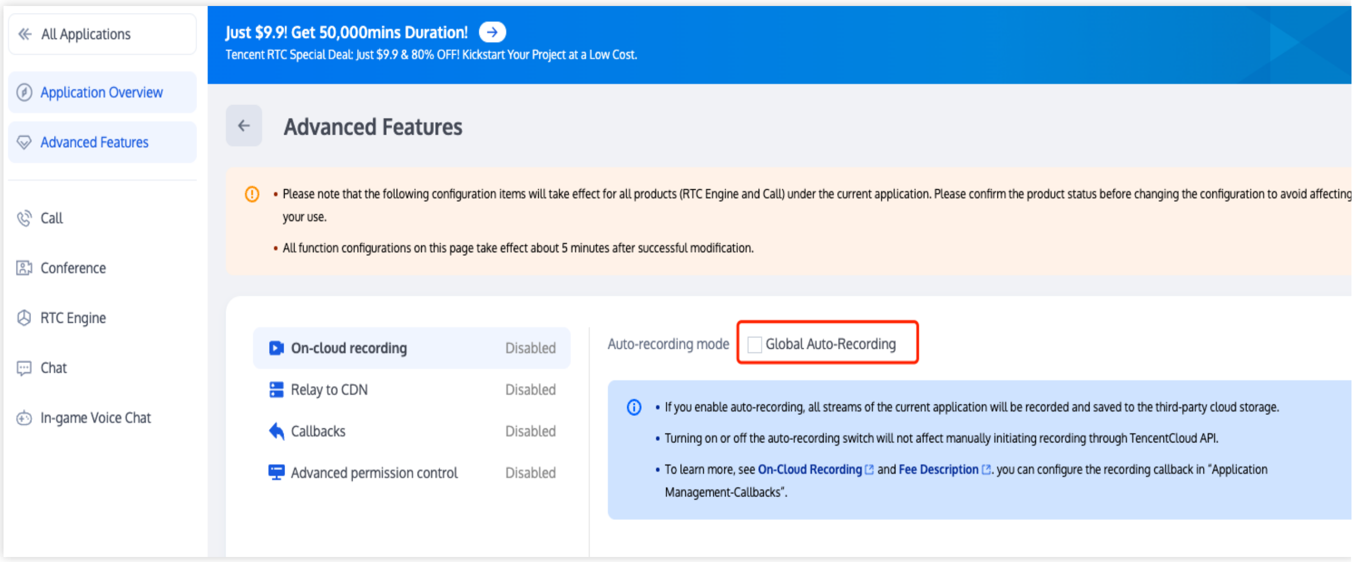
In the single-stream recording mode, you can record up to 25 streams in a room at the same time.

Auto-Recording

TencentRTC offers an auto-recording feature, eliminating the need for manual recording task management. To use this recording solution, go to **Console** > Applications, select the desired application, and click

on the right side to enter the application information page. Then click **Advanced Features** , enable on-cloud auto-recording, complete the global auto-recording template configuration, and submit it.

After it becomes effective (wait for 5-10 minutes for it to take effect), the publishing of audio and video by the anchors in the TRTC room will trigger the start of the recording task. The recording task will be triggered to stop after all the anchors have left the room and the set timeout period for resumption has been exceeded.



Just $9.9! Get 50,000mins Duration!
Tencent RTC Special Deal: Just $9.9 & 80% OFF! Kickstart Your Project at a Low Cost.

- Please note that the following configuration items will take effect for all products (RTC Engine and Call) under the current application. Please confirm the product status before changing the configuration to avoid affecting your use.
- All function configurations on this page take effect about 5 minutes after successful modification.

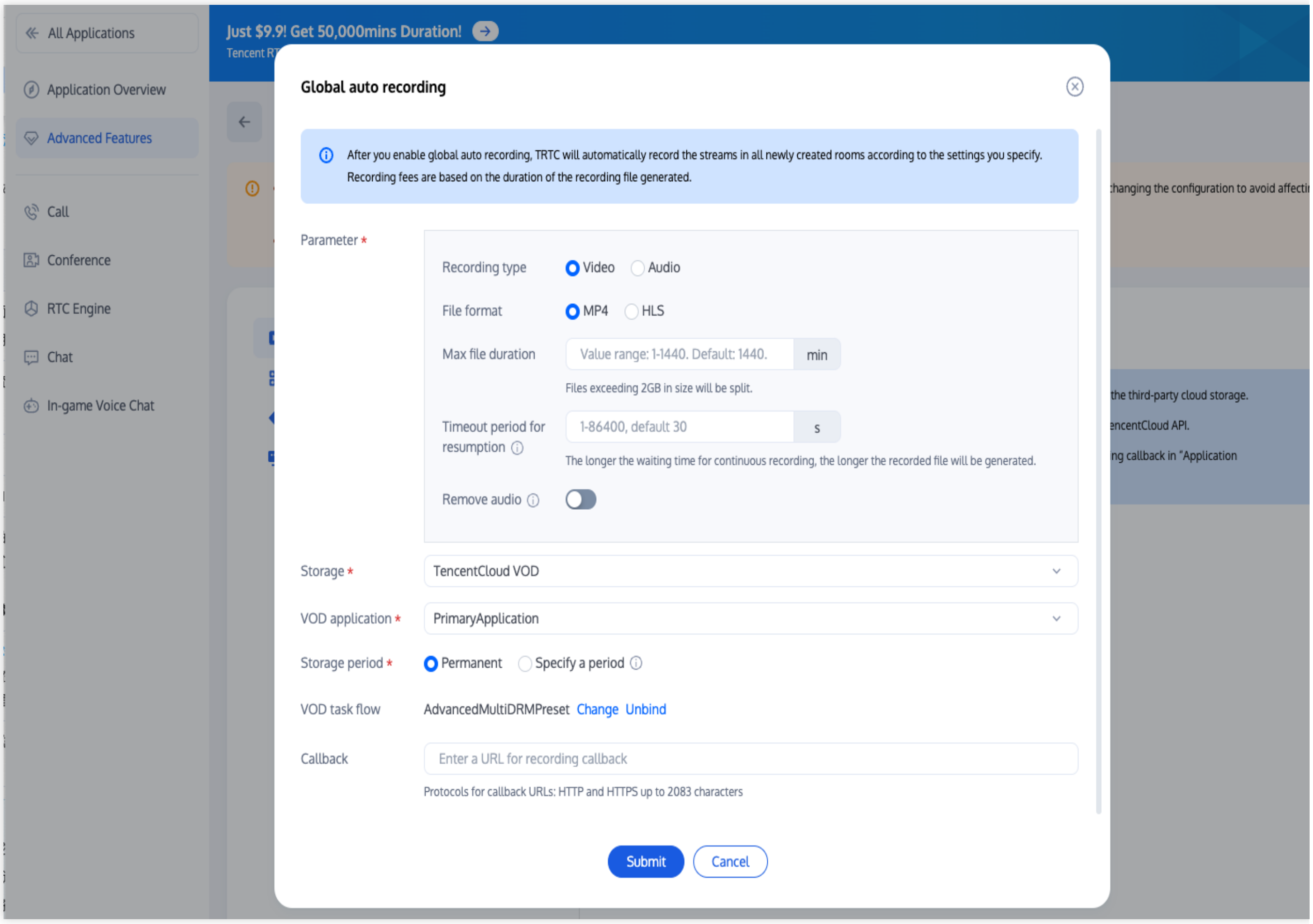
Auto-recording mode: Global Auto-Recording

- If you enable auto-recording, all streams of the current application will be recorded and saved to the third-party cloud storage.
- Turning on or off the auto-recording switch will not affect manually initiating recording through TencentCloud API.
- To learn more, see On-Cloud Recording and Fee Description, you can configure the recording callback in "Application Management-Callbacks".

Before enabling the global auto-recording feature, configure the global auto-recording template. Global auto-recording supports single-stream recording (i.e., record a separate file for each anchor), and once enabled, it is only effective for newly created rooms. It does not apply to rooms created before the auto-recording feature was enabled.
Global single-stream recording supports audio and video recording, audio-only recording, and video-only recording. The recording file formats supported are MP4, HLS, and AAC (in audio-only recording format).

Configuration Item	Description
Recording mode	Single-stream recording: The video data of each anchor in the room will be saved as a separate file. To record the mixed video of multiple anchors, use Manual Merge Recording.
Recording type	Audio and video: Both audio and video streams in the room are recorded, suitable for video calls and interactive live streaming scenarios. Audio-only: Only audio streams in the room are recorded.
File format	Supports MP4, HLS, and AAC (in audio-only format).
Max file duration	Specifies the segment duration of the recording file, with a range of 1-1,440 minutes. The default is 1,440 minutes.
Timeout period for resumption	Sets the timeout period for resuming recording in seconds. When the interruption interval does not exceed the set timeout period, a single call (or live stream) will generate only one file. However, the recording file will be received only after the timeout period for resumption expires. **The value range is 1 - 86,400 (default 30s)** . **Note: During the resumption waiting period, single-stream recording fees will be charged based on the audio duration. Please set it appropriately.**

Storage	Supports storage to Tencent Cloud Video on Demand (VOD), Tencent Cloud Object Storage (COS), and AWS S3 Storage. VOD: Requires support for specifying the VOD application and the storage period of recording files in VOD, and binding VOD task flows. COS & AWS: Requires completion of configuration for the corresponding bucket. Ensure you have the write permission to the bucket.
Callback address and callback key	The latest on-cloud recording service offers detailed recording event features. You can configure a server-side URL to receive recording callback events, and can also configure a callback key to verify the security of callback events. More information available here.



Global auto recording

After you enable global auto recording, TRTC will automatically record the streams in all newly created rooms according to the settings you specify. Recording fees are based on the duration of the recording file generated.

Parameter *

Recording type: [✓ Video] Audio

File format: [✓ MP4] HLS

Max file duration: Value range: 1-1440. Default: 1440. min

Files exceeding 2GB in size will be split.

Timeout period for resumption: 1-86400, default 30 s

The longer the waiting time for continuous recording, the longer the recorded file will be generated.

Remove audio: off

Storage *: TencentCloud VOD

VOD application *: PrimaryApplication

Storage period *: [✓ Permanent] Specify a period

VOD task flow: AdvancedMultiDRMPreset Change Unbind

Callback: Enter a URL for recording callback

Protocols for callback URLs: HTTP and HTTPS up to 2083 characters

Submit Cancel

Note:

In single-stream recording mode, each audio and video stream in the room will be recorded separately according to the push stream parameters, without the need of setting transcoding.

If the timeout period for resumption has not expired, the recording robot will continue to wait in the room for the anchor's publishing to complete the recording. The recording will not end immediately after the anchor leaves the room, so set the timeout period appropriately.

Single-stream recording can record the audio and video of up to 25 anchors in one room. If there are more than 25 anchors in the room, they will be sorted by the time they entered the room, and only the audio and video of the first 25 anchors will be recorded. (For single-stream recording of more than 25 anchors, see API recording).

Manual Recording Process

1. Start recording

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

TaskId

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

RecordMode

Single-stream recording separately records the audios and videos of the anchors you subscribe to and uploads the recording files (including M3U8 and TS segment files) to the cloud.
Mixed-stream recording records all the audios and videos of the anchors you subscribe to into one result and uploads the recording files (including M3U8 and TS segment files) to the cloud.

SubscribeStreamUserIds

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

StorageParams

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

Naming of recording files

M3U8 file in the single-stream recording mode:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>.m3u8
TS segment file in the single-stream recording mode:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>_<UTC>.ts
MP4 file in the single-stream recording mode:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Index>.mp4
M3U8 file in the mixed-stream recording mode:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>.m3u8

TS segment file in the mixed-stream recording mode:

<Prefix>/<TaskId>/<SdkAppId>_<RoomId>_<UTC>.ts

MP4 file in the mixed-stream recording mode:

<Prefix>/<TaskId>/<SdkAppId>_<RoomId>_<Index>.mp4

Naming of recovered files

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

M3U8 file in the single-stream recording mode:

<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>.m3u8

TS segment file in the single-stream recording mode:

<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>_<UTC>.ts

M3U8 file in the mixed-stream recording mode:

<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>.m3u8

TS segment file in the mixed-stream recording mode:

<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>_<UTC>.ts

Field description

<Prefix>: The filename prefix. If this is not specified, the filename will not have a prefix.

<TaskId>: The task ID, which is unique and is returned by the start recording API.

<SdkAppId>: The application ID.

<RoomId>: The room ID.

<UserId>: The Base64-encoded ID of a user whose stream is recorded.

<MediaId>: Whether the primary stream (`main`) or substream (`aux`) is recorded.

<Type>: The type of stream that is recorded (`audio` or `video`).

<UTC>: The recording start time (UTC+0), which consists of the year, month, day, hours, minutes, seconds, and milliseconds.

<Index>: The index of a segment, which starts from 1. This field is used only if an MP4 file exceeds 2 GB or 24 hours and needs to be segmented.

ha<1/2/3>: The prefix for a file recovered by the high availability scheme. For example, if the scheme is used for the first time, the recovered file is named `<Prefix>/<TaskId>/ha1_<SdkAppId>_<RoomId>.m3u8` .

Note:

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

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

Recording start time

Recording starts when you start receiving data from an anchor. Recording start time is the Unix time on the server when recording starts.
You can query the start time of a recording task in three ways:
Via the `DescribeCloudRecording` API. The response parameter `BeginTimeStamp` indicates the recording start time (ms).
Below is a response of the `DescribeCloudRecording` API, in which `BeginTimeStamp` is `1622186279144`.

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

Via the #EXT-X-TRTC-START-REC-TIME directive of the M3U8 file
According to the M3U8 file below, the Unix time (ms) when recording started was 1622425551884.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:NO
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-TARGETDURATION:70
#EXT-X-TRTC-START-REC-TIME:1622425551884
#EXT-X-TRTC-VIDEO-METADATA:WIDTH:1920 HEIGHT:1080
#EXTINF:12.074
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094551825.ts
#EXTINF:11.901
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094603825.ts
#EXTINF:12.076
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094615764.ts
#EXT-X-ENDLIST
```

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

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

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

MixWatermark

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

Field	Description
Top	The vertical offset of the watermark to the top left corner of the video.
Left	The horizontal offset of the watermark to the top left corner of the video.
Width	The watermark width.
Height	The watermark height.
url	The URL of the watermark file.

MixLayoutMode

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

Grid layout

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

When there is only one video:

The video is scaled to fill the canvas.

When there are two videos:

The width of each video is half of the canvas width.

The height of each video is the same as the canvas height.

When there are three or four videos:

The canvas is split evenly into four windows and each video is displayed in one window.

When there are 5-9 videos:

The canvas is split evenly into nine windows and each video is displayed in one window.

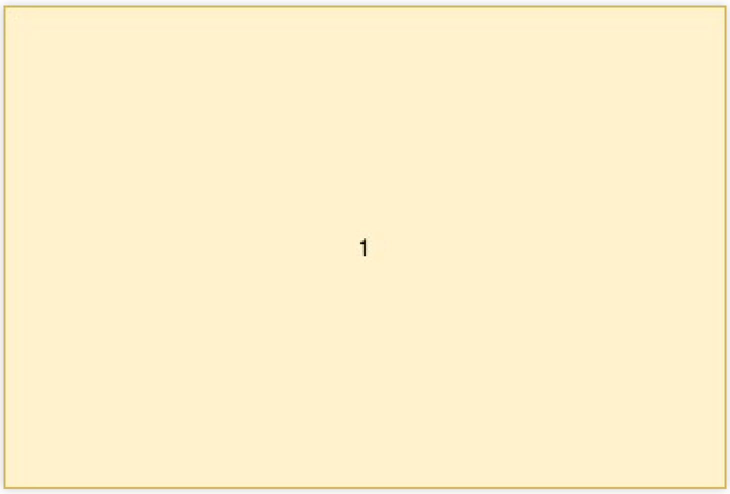
When there are 10-16 videos:

The canvas is split evenly into 16 windows and each video is displayed in one window.

When there are more than 16 videos:

The canvas is split evenly into 25 windows and each video is displayed in one window.

As shown below:



Floating layout

By default, the video of the first anchor in the room (you can also specify an anchor) is scaled to fill the screen. When other anchors enter the room, their videos appear smaller and float over the large video from left to right starting from the bottom of the canvas. If the total number of videos is 17 or less, there will be four windows in each row (4 x 4); if it is greater than 17, there will be five windows in each row (5 x 5). Up to 25 videos can be displayed. A user who publishes only audio will still be displayed in one window.

When there are 17 or fewer videos:

The width and height of each small video are 23.5% of the canvas width and height.

The horizontal space and vertical space between two neighboring videos are 1.2% of the canvas width and height.

The right and left margins are 1.2% of the canvas width and the top and bottom margins are 1.2% of the canvas height.

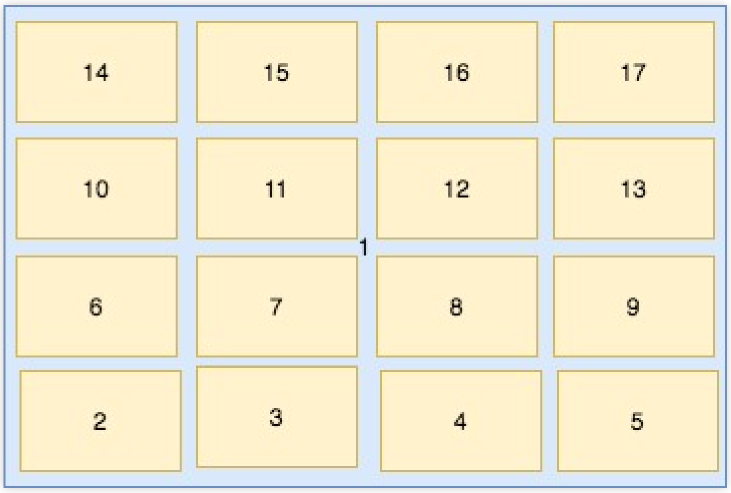
When there are more than 17 videos:

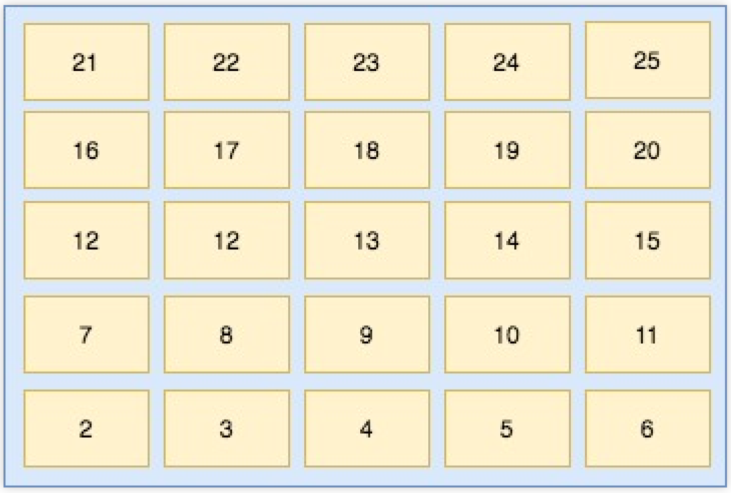
The width and height of each small video are 18.8% of the canvas width and height.

The horizontal space and vertical space between two neighboring videos are 1% of the canvas width and height.

The right and left margins are 1% of the canvas width and the top and bottom margins are 1% of the canvas height.

As shown below:





Screen sharing layout

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

When there are five or fewer videos:
The size of each small video on the right is 1/5 the canvas width and 1/4 the canvas height.
The width of the large video on the left is 4/5 the canvas width, and its height is the same as the canvas height.
When there are six or seven videos:
The size of each small video on the right are 1/7 the canvas width and 1/6 the canvas height.
The width of the large video on the left is 6/7 the canvas width, and its height is the same as the canvas height.
When there are eight or nine videos:
The size of each small video on the right is 1/9 the canvas width and 1/8 the canvas height.
The width of the large video on the left is 8/9 the canvas width, and its height is the same as the canvas height.
When there are 10-17 videos:
The size of each small video on the right side is 1/10 the canvas width and 1/8 the canvas height.
The width of the large video on the left side is 4/5 the canvas width, and its height is the same as the canvas height.
When there are more than 17 videos:
The size of each small video on the right and bottom is 1/10 the canvas width and 1/8 the canvas height.
The width of the large video on the left is 4/5 the canvas width, and its height is 7/8 the canvas height.
As shown below:

Custom layout

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

2. Query the recording task

You can use the `DescribeCloudRecording` API to query the status of an ongoing recording task. If the queried task has already ended, an error will be returned.
The file list (`StorageFile`) that is returned will include all the M3U8 files of the recording as well as the Unix time when recording started. If the task queried is a recording to VOD task, the `StorageFile` returned will be empty.

3. Modify recording parameters

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

4. Stop recording

You can call the `DeleteCloudRecording` API to stop recording. A recording task will also end automatically if there are no anchors (whether they are publishing data or not) in a room for longer than the specified time period (`MaxIdleTime`). We recommend you call the API to stop recording when you no longer need the service.

Advanced Features

Recording in MP4 format

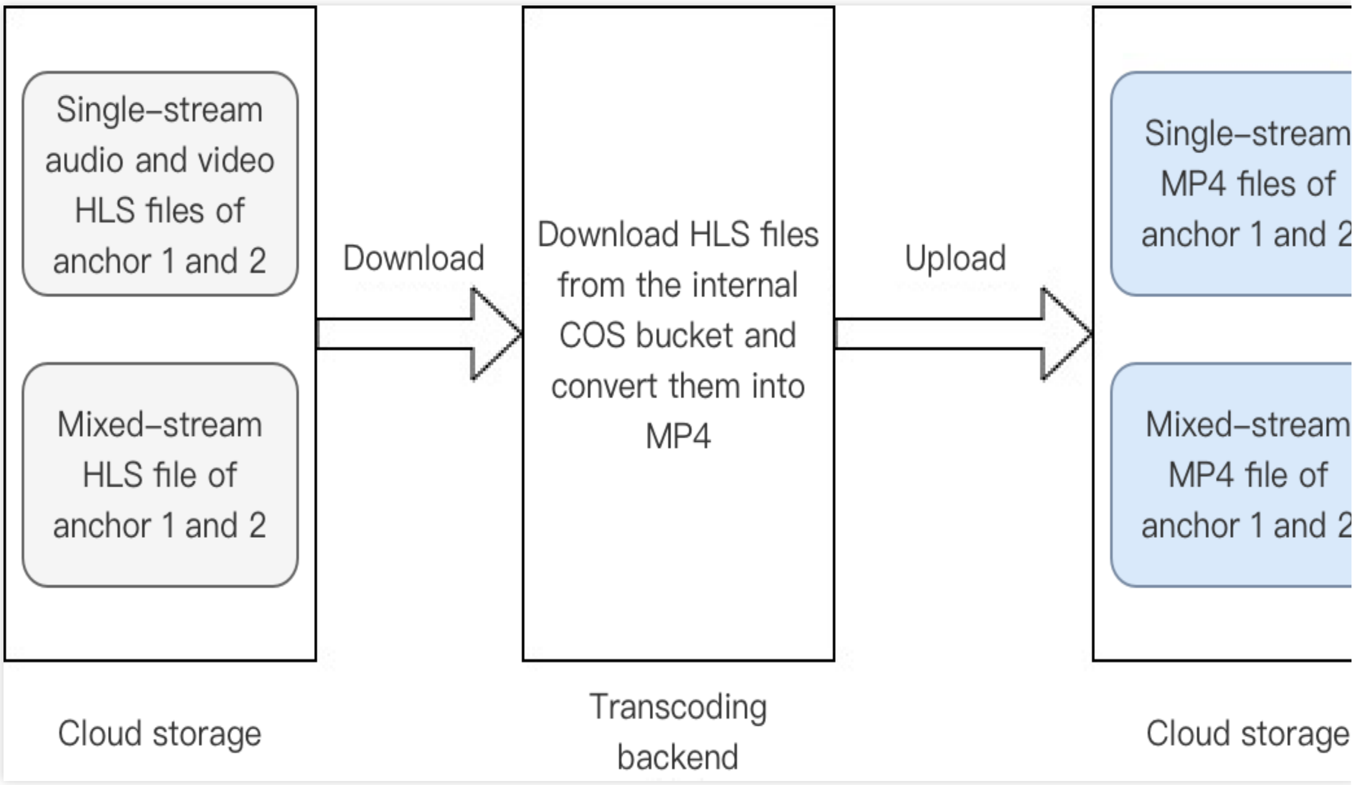
To record in MP4 format, set `OutputFormat` to `hls+mp4` when calling the `CreateCloudRecording` API. TRTC will still record in HLS format, but once recording ends, it will download the HLS file from the COS bucket where it is saved, convert it into MP4 format, and upload the MP4 file to the COS bucket.

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

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

1. The recording duration is longer than 24 hours.
2. The MP4 file exceeds 2 GB.

The workflow is as follows:

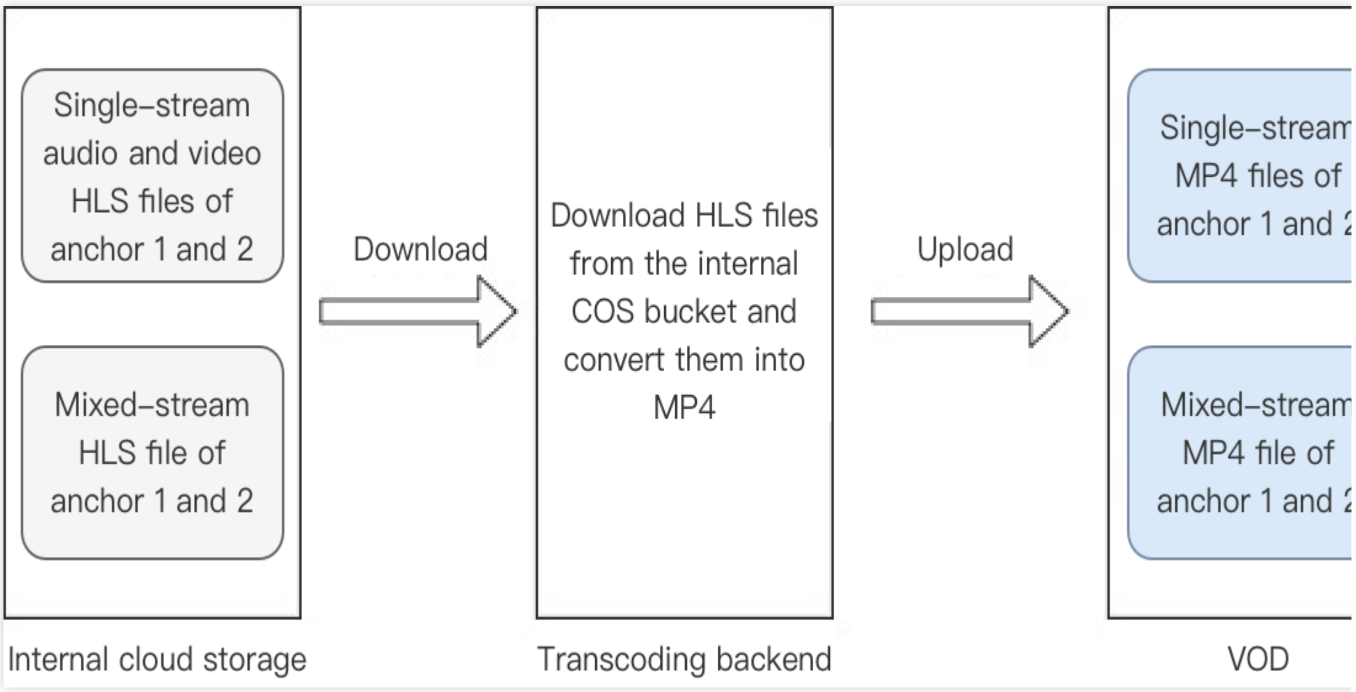

Single–stream audio and video HLS files of anchor 1 and 2
Mixed–stream HLS file of anchor 1 and 2
Cloud storage
Download
Download HLS files from the internal COS bucket and convert them into MP4
Transcoding backend
Upload
Single–stream MP4 files of anchor 1 and 2
Mixed–stream MP4 file of anchor 1 and 2
Cloud storage

Recording to VOD

To record to VOD, specify the `CloudVod` parameter in `StorageParams` when calling the `CreateCloudRecording` API. After recording ends, the backend will save the file in MP4 format to VOD using the method you specify and send you a playback URL via a callback. In the single-stream recording mode, there will be a playback URL for each anchor whose stream is recorded; in the mixed-stream recording mode, there will be only one playback URL. When you record to VOD, pay attention to the following:

1. `CloudVod` and `CloudStorage` are mutually exclusive. If you specify both, the recording task will fail to start.

2. If you use `DescribeCloudRecording` to query a recording to VOD task, the `StorageFile` returned will be empty.
The figure below shows the workflow of recording to VOD. "Internal cloud storage" refers to the internal storage of the recording backend.


Single–stream audio and video HLS files of anchor 1 and 2
Mixed–stream HLS file of anchor 1 and 2
Download
Download HLS files from the internal COS bucket and convert them into MP4
Upload
Single–stream MP4 files of anchor 1 and 2
Mixed–stream MP4 file of anchor 1 and 2
Internal cloud storage
Transcoding backend
VOD

Script for merging single-stream recording files

We offer a script for merging single-stream audio and video files into MP4 files.
Note:
If two segment files are more than 15 seconds apart, during which no audio or video data is recorded (if the substream is disabled, its data will be ignored), the two segments will be considered to belong to different sections, one being the ending segment of the previous section, and the other the starting segment of the next section.
Section-based merge (`-m 0`)
In this mode, the recording files of each user (`UserId`) are merged by section. One MP4 file is generated for each section.
User-based merge (`-m 1`)
In this mode, the recording files of each user (`UserId`) are merged into one MP4 file. You can use the `-s` option to specify whether to fill in the blanks between sections.

Environment Requirements

Python 3
Centos: sudo yum install python3
Ubuntu: sudo apt-get install python3

Python 3 dependency

```
sortedcontainers: pip3 install sortedcontainers
```

Directions

1. Run the merge script `python3 TRTC_Merge.py [option]` .
2. An MP4 file will be generated in the directory of the recording files.

Example: python3 TRTC_Merge.py -f /xxx/file -m 0

Below is a list of the options:

Parameter	Description
-f	The directory of the recording files to be merged. If there are multiple users (`UserId`), their recording files will be merged separately.
-m	`0` : Section-based merge (default). In this mode, the recording files of each user (`UserId`) are merged by section. Multiple files may be generated for each user. `1` : User-based merge. In this mode, the recording files of each user (`UserId`) are merged into one file.
-s	Whether to delete the blanks between sections in the user-based merge mode. If they are deleted, the files generated will be shorter than the recording duration.
-a	`0` : Primary stream merge (default). The audio of a user (`UserId`) is merged with the user's primary stream, not the substream. `1` : Automatic merge. If a user (`UserId`) has a primary stream, the user's audio is merged with the primary stream; if not, it is merged with the substream. `2` : Substream merge. The audio of a user (`UserId`) is merged with the user's substream, not the primary stream.
-p	The frame rate (fps) of the output video, which is `15` by default. Value range: 5-120. If you enter a value smaller than `5` , `5` will be used; if you enter a value greater than `120` , `120` will be used.
-r	The resolution of the output video. For example, `-r 640 360` indicates that the resolution of the output video is 640 x 360.

File Naming

Audio-video file: UserId_timestamp_av.mp4

Audio-only file: UserId_timestamp.m4a

Note:

`UserId` is the Base64-encoded ID of a user whose stream is recorded. For details, see the naming of recording files. `timestamp` is the starting time of the first TS segment of a section.

Callback APIs

You can register callbacks by providing an HTTP/HTTPS gateway to receive callbacks. When an on-cloud recording event occurs, the system will send a callback notification to your callback server.

Callback format

Callbacks are sent to your server in the form of HTTP/HTTPS POST requests.

Character encoding: UTF-8

Request: The request body is in JSON format.

Response: HTTP STATUS CODE = 200. The server ignores the content of the response packet. For protocol-friendliness, we recommend adding `JSON:` {"code":0}`` to the response.

Parameter description

The header of a callback contains the following fields.

Field	Description
Content-Type	application/json
Sign	The signature.
SdkAppId	The application ID.

The body of a callback contains the following fields.

Field	Type	Description
EventGroupId	Number	The event group ID, which is `3` for on-cloud recording.
EventType	Number	The event type.
CallbackTs	Number	The Unix timestamp (ms) when the callback was sent to your server.
EventInfo	JSON Object	The event information.

Event types

Field	Type	Description
EVENT_TYPE_CLOUD_RECORDING_RECORDER_START	301	On-cloud recording - The recording module was started.
EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP	302	On-cloud recording - The

		recording module was stopped.
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START	303	On-cloud recording - The upload module was started.
EVENT_TYPE_CLOUD_RECORDING_FILE_INFO	304	On-cloud recording - The first M3U8 file was generated and uploaded successfully.
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP	305	On-cloud recording - The files are uploaded.
EVENT_TYPE_CLOUD_RECORDING_FAILOVER	306	On-cloud recording - The recording task was migrated.
EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE	307	On-cloud recording - An M3U8 file was generated (the first TS segment was generated).
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR	308	On-cloud recording - The upload module encountered an error.
EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR	309	On-cloud recording - An error occurred when downloading the image decoding file.
EVENT_TYPE_CLOUD_RECORDING_MP4_STOP	310	On-cloud recording - An MP4 recording task is finished. The callback includes the name and other details of the MP4 file generated.
EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT	311	On-cloud recording - The recording files were uploaded to VOD.
EVENT_TYPE_CLOUD_RECORDING_VOD_STOP	312	On-cloud recording - A recording to VOD task is finished.

Event information

Field	Type	Description
RoomId	String/Number	The room ID, which must be the same data type as the room ID on the client.
EventTs	Number	The Unix timestamp (seconds) when the event occurred.
UserId	String	The user ID of the recording robot.
TaskId	String	The ID of a recording task.
Payload	JsonObject	This parameter is defined differently for different event types.

If the event type is `301` (EVENT_TYPE_CLOUD_RECORDING_RECORDER_START):

Field	Type	Description
Status	Number	`0` : The recording module was started successfully. `1` : Failed to start the recording module.

```
{
        "EventGroupId": 3,
        "EventType":    301,
        "CallbackTs":   1622186275913,
        "EventInfo":    {
                "RoomId":       "xx",
                "EventTs":      "1622186275",
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0
                }
        }
}
```

If the event type is `302` (EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP):

Field	Type	Description
LeaveCode	Number	`0` : An API was called to stop the recording module. `1` : The recording robot was removed from the room. `2` : You closed the room. `3` : The server removed the recording robot from the room. `4` : The server closed the room.

		`99` : There were no anchors in the room, and the recording robot left after the specified time period elapsed. `100` : The room timed out. `101` : The recording robot was removed due to repeat entry by the same user.

```
{
        "EventGroupId": 3,
        "EventType":    302,
        "CallbackTs":   1622186354806,
        "EventInfo":    {
                "RoomId":       "xx",
                "EventTs":      "1622186354",
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "LeaveCode":    0
                }
        }
}
```

If the event type is `303` (EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START):

Field	Type	Description
Status	Number	`0` : The upload module was started successfully. `1` : Failed to start the upload module.

```
{
        "EventGroupId": 3,
        "EventType":    303,
        "CallbackTs":   1622191965320,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0
                }
        }
}
```

If the event type is `304` (EVENT_TYPE_CLOUD_RECORDING_FILE_INFO):

Field	Type	Description

FileList	String	The name of the M3U8 file generated.

```
{
        "EventGroupId": 3,
        "EventType":    304,
        "CallbackTs":   1622191965350,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "FileList":     "xx.m3u8"
                }
        }
}
```

If the event type is `305` (EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP):

Field	Type	Description
Status	Number	`0` : The recording task is finished and all the files were uploaded to the specified cloud storage service. `1` : The recording task is finished, but at least one file is still on the server or in backup storage. `2` : The files on the server or in backup storage were uploaded to the specified cloud storage service.

```
{
        "EventGroupId": 3,
        "EventType":    305,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0
                }
        }
}
```

If the event type is `306` (EVENT_TYPE_CLOUD_RECORDING_FAILOVER):

Field	Type	Description
Status	Number	`0` : The migration was completed.

```
{
        "EventGroupId": 3,
        "EventType":    306,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0
                }
        }
}
```

If the event type is `307` (EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE):

Field	Type	Description
FileName	String	The name of the M3U8 file.
UserId	String	The ID of the user whose streams were recorded.
TrackType	String	Valid values: `audio` , `video` , `audio_video` .
BeginTimeStamp	Number	The Unix timestamp (milliseconds) on the server when recording started.

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

If the event type is `308` (EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR):

Field	Type	Description
Code	String	The error code returned by the cloud storage service.
Message	String	The error message returned by the cloud storage service.

```
{
  "Code": "InvalidParameter",
  "Message": "AccessKey invalid"
}

{
        "EventGroupId": 3,
        "EventType":    308,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Code":       "xx",
                        "Message":    "xx"
                }
        }
}
```

If the event type is `309` (EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR):

Field	Type	Description
Url	String	The download URL.

```
{
        "EventGroupId": 3,
        "EventType":    309,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Url":          "http://xx",
                }
        }
}
```

If the event type is `310` (EVENT_TYPE_CLOUD_RECORDING_MP4_STOP):

Field	Type	Description
Status	Number	`0` : The MP4 recording task is finished and all the files were uploaded to the specified cloud storage service. `1` : The MP4 recording task is finished, but at least one file is still on the server or in backup storage. `2` : The MP4 recording task stopped due to an error (probably because the system failed to download the HLS files from COS).
FileList	Array	The names of the MP4 files generated.
FileMessage	Array	The information of the MP4 files generated.
FileName	String	The filename.
UserId	String	The user ID. In the mixed-stream recording mode, this field is empty.
TrackType	String	Valid values: `audio` , `video` , `audio_video` .
MediaId	String	Valid values: `main` , `aux` .
StartTimeStamp	Number	The Unix timestamp (milliseconds) when the MP4 file started.
EndTimeStamp	Number	The Unix timestamp (milliseconds) when the MP4 file ended.

```
{
        "EventGroupId": 3,
        "EventType":    310,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0,
                        "FileList": ["xxxx1.mp4", "xxxx2.mp4"],
                        "FileMessage": [
                                {
                                        "FileName": "xxxx1.mp4",
                                        "UserId": "xxxx",
                                        "TrackType": "audio_video",
                                        "MediaId":      "main",
                                        "StartTimeStamp": 1622186279145,
                                        "EndTimeStamp": 1622186282145
                                },
                                {
                                        "FileName": "xxxx2.mp4",
                                        "UserId": "xxxx",
                                        "TrackType": "audio_video",
                                        "MediaId":      "main",
                                        "StartTimeStamp": 1622186279153,
                                        "EndTimeStamp": 1622186282153
                                }
                        ]
                }
        }
}
```

If the event type is `311` (EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT):

Field	Type	Description
Status	Number	`0` : The recording file was successfully uploaded to VOD. `1` : The recording file is still on the server or in backup storage. `2` : An error occurred when uploading the recording file to VOD.
UserId	String	The user ID. In the mixed-stream recording mode, this field is empty.
TrackType	String	Valid values: `audio` , `video` , `audio_video` .
MediaId	String	Valid values: `main` , `aux` .
FileId	String	The ID of the recording file in VOD.
VideoUrl	String	The playback URL of the recording file in VOD.
CacheFile	String	The name of the MP4 file before it was uploaded to VOD.
Errmsg	String	The error message. This field is not empty if `status` is not `0` .

A callback for successful upload:

```
{
        "EventGroupId": 3,
        "EventType":    311,
        "CallbackTs":   1622191965320,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0,
                        "TencentVod":   {
                            "UserId":       "xx",
                            "TrackType":    "audio_video",
                            "MediaId":      "main",
                            "FileId":       "xxxx",
                            "VideoUrl":     "http://xxxx"
                        }
                }
        }
}
```

A callback for failed upload:

```
{
        "EventGroupId": 3,
        "EventType":    311,
        "CallbackTs":   1622191965320,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       1,
                        "Errmsg":       "xxx",
                        "TencentVod":   {
                            "UserId":       "123",
                            "TrackType":    "audio_video",
                            "CacheFile":    "xxx.mp4"
                        }
                }
        }
}
```

If the event type is `312` (EVENT_TYPE_CLOUD_RECORDING_VOD_STOP):

Field	Type	Description
Status	Number	`0` : The recording to VOD task ended normally. `1` : The recording to VOD task ended due to an error.

```
{
        "EventGroupId": 3,
        "EventType":    312,
        "CallbackTs":   1622191965320,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0
                }
        }
}
```

Best Practices

To ensure the high availability of the recording service, we recommend the following practices when you use the RESTful APIs:

1. Pay attention to the HTTP response after you call `CreateCloudRecording` . If your request fails, fix the problem according to the status code and try again.

The status code consists of two parts, for example `InvalidParameter.SdkAppId` .

`InternalError.xxxxx` indicates that a server error occurred. You can retry until the request succeeds and `TaskId` is returned. We recommend you use the exponential backup algorithm for retry. For example, you can wait for three seconds for the first retry, six seconds for the second, 12 seconds for the third, and so on.

`InvalidParameter.xxxxx` indicates that a parameter value entered was invalid. Please check the parameter.

`FailedOperation.RestrictedConcurrency` indicates that you reached the maximum number (100 by default) of ongoing recording tasks allowed. To raise the limit, please contact technical support.

2. The `UserId` and `UserSig` you pass in when calling `CreateCloudRecording` are for the recording robot. Please make sure that they are different from those of other users in the room. In addition, the room joined by users from the TRTC client must be of the same type as the room you specify when calling the API. For example, if the room created in the TRTC SDK is a string, the room specified for on-cloud recording must also be a string.

3. You can obtain the information of a recording file in the following ways.

Call `DescribeCloudRecording` 15 seconds after a `CreateCloudRecording` request succeeds. If the task status is idle, it indicates that no audio or video data is available for recording. Please check whether there are anchors publishing data in the room.
After a `CreateCloudRecording` request succeeds, if there are anchors publishing data in the room, you can splice the names of the recording files according to the naming rules.
If you have registered on-cloud recording callbacks, the information of recording files will be sent to your server via callbacks.
You can specify a COS bucket to save recording files when calling the `CreateCloudRecording` API. After a recording task ends, you can find the recording files in the COS bucket you specify.
4. Make sure the validity period of the recording user's `UserSig` is longer than the duration of the recording task. This is to avoid cases where the high availability scheme fails to resume a recording task after a disconnection because the `UserSig` has expired.

Custom Capturing and Rendering Android, iOS, Windows, and macOS

Last updated：2023-10-08 15:41:41

This document describes how to use the TRTC SDK to implement custom video capturing and rendering.

Custom Video Capturing

The custom video capturing feature of the TRTC SDK can be used in two steps: enabling the feature and sending video frames to the SDK. For detailed directions of specific APIs, see below. We also provide API examples for different platforms:

Android

iOS

Windows

Enabling custom video capturing

To enable the custom video capturing feature of the TRTC SDK, you need to call the `enableCustomVideoCapture` API of `TRTCCloud`. Then, the TRTC SDK's camera capturing and image processing logic will be skipped, and only its encoding and transfer capabilities will be retained. Below is the sample code:

Android

iOS&Mac

Windows

```
TRTCCloud mTRTCCloud = TRTCCloud.shareInstance();
mTRTCCloud.enableCustomVideoCapture(TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, true);
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
[self.trtcCloud enableCustomVideoCapture:TRTCVideoStreamTypeBig enable:YES];
```

```
liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance();
trtc_cloud->enableCustomVideoCapture(TRTCVideoStreamType::TRTCVideoStreamTypeBig, t
```

Sending custom video frames

Then, you can use the `sendCustomVideoData` API of `TRTCCloud` to populate the TRTC SDK with your own video data. Below is the sample code:

explain

In order to avoid performance loss, there are different format requirements for the video data input to the TRTC SDK on different platforms. For more information, see LiteAVSDK Overview.

Android

iOS&Mac

Windows

```
// Two schemes are available for Android: Texture (recommended) and Buffer. Texture
TRTCCloudDef.TRTCVideoFrame videoFrame = new TRTCCloudDef.TRTCVideoFrame();
videoFrame.texture = new TRTCCloudDef.TRTCTexture();
videoFrame.texture.textureId = textureId;
videoFrame.texture.eglContext14 = eglContext;
videoFrame.width = width;
videoFrame.height = height;
videoFrame.timestamp = timestamp;
videoFrame.pixelFormat = TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture_2D;
videoFrame.bufferType = TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE;
mTRTCCloud.sendCustomVideoData(TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, videoFrame)
```

```
// On iOS and macOS, the video captured by the camera is in NV12 format. The native
TRTCVideoFrame *videoFrame = [[TRTCVideoFrame alloc] init];
videoFrame.pixelFormat = TRTCVideoPixelFormat_NV12;
videoFrame.bufferType = TRTCVideoBufferType_PixelBuffer;
videoFrame.pixelBuffer = imageBuffer;
videoFrame.timestamp = timeStamp;

[[TRTCCloud sharedInstance] sendCustomVideoData:TRTCVideoStreamTypeBig frame:videoF
```

```
// Only the Buffer scheme is available for Windows currently and is recommended for
liteav::TRTCVideoFrame frame;
frame.timestamp = getTRTCShareInstance()->generateCustomPTS();
frame.videoFormat = liteav::TRTCVideoPixelFormat_I420;
frame.bufferType = liteav::TRTCVideoBufferType_Buffer;
frame.length = buffer_size;
frame.data = array.data();
frame.width = YUV_WIDTH;
frame.height = YUV_HEIGHT;
getTRTCShareInstance()->sendCustomVideoData(&frame);
```

Custom Video Rendering

Custom rendering is mainly divided into rendering of the local video and rendering of the remote video. You can set the callback for local/remote custom rendering, and the TRTC SDK will pass the corresponding video frames (`TRTCVideoFrame`) through the callback function `onRenderVideoFrame` . Then, you can customize the rendering of the received video frames. This process requires certain knowledge of OpenGL. We also provide API examples for different platforms:

Android:

iOS

Windows

Setting the local video rendering callback

Android

iOS&Mac

Windows

```
mTRTCCloud.setLocalVideoRenderListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture
    @Override
    public void onRenderVideoFrame(String suserId int streamType, TRTCCloudDef.TRTC
        // For more information, see the custom rendering tool class `com.tencent.t
    }
});
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
[self.trtcCloud setLocalVideoRenderDelegate:self pixelFormat:TRTCVideoPixelFormat_N
```

```
```

// For specific implementation, see `test_custom_render.cpp` in `TRTC-API-Example-Q
void TestCustomRender::onRenderVideoFrame(
    const char* userId,
    liteav::TRTCVideoStreamType streamType,
    liteav::TRTCVideoFrame* frame) {
  if (gl_yuv_widget_ == nullptr) {
    return;
  }

  if (streamType == liteav::TRTCVideoStreamType::TRTCVideoStreamTypeBig) {
    // Adjust the rendering window
```
```

```
    emit renderViewSize(frame->width, frame->height);
    // Draw video frames
    gl_yuv_widget_->slotShowYuv(reinterpret_cast<uchar*>(frame->data),
                                frame->width, frame->height);
  }
}
```
```

## Setting the rendering callback of remote video

Android

iOS&Mac

Windows

```
mTRTCCloud.setRemoteVideoRenderListener(userId, TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMA
    @Override
    public void onRenderVideoFrame(String userId, int streamType, TRTCCloudDef.TRTC
         // For more information, see the custom rendering tool class `com.tencent.
    }
});
```

```
- (void)onRenderVideoFrame:(TRTCVideoFrame *)frame
                    userId:(NSString *)userId
                streamType:(TRTCVideoStreamType)streamType
{
    // If `userId` is `nil`, the image rendered is the local image; otherwise it is
    CFRetain(frame.pixelBuffer);
    __weak __typeof(self) weakSelf = self;
    dispatch_async(dispatch_get_main_queue(), ^{
        TestRenderVideoFrame *strongSelf = weakSelf;
        UIImageView* videoView = nil;
        if (userId) {
            videoView = [strongSelf.userVideoViews objectForKey:userId];
        }
        else {
            videoView = strongSelf.localVideoView;
        }
        videoView.image = [UIImage imageWithCIImage:[CIImage imageWithCVImageBuffer
        videoView.contentMode = UIViewContentModeScaleAspectFit;
        CFRelease(frame.pixelBuffer);
    });
}
```
```

~~~
```
// For specific implementation, see `test_custom_render.cpp` in `TRTC-API-Example-Q
void TestCustomRender::onRenderVideoFrame(
    const char* userId,
    liteav::TRTCVideoStreamType streamType,
    liteav::TRTCVideoFrame* frame) {
  if (gl_yuv_widget_ == nullptr) {
    return;
  }

  if (streamType == liteav::TRTCVideoStreamType::TRTCVideoStreamTypeBig) {
    // Adjust the rendering window
    emit renderViewSize(frame->width, frame->height);
    // Draw video frames
    gl_yuv_widget_->slotShowYuv(reinterpret_cast<uchar*>(frame->data),
                                frame->width, frame->height);
  }
}
```
~~~

Web

Last updated：2023-10-08 15:55:32

Function Description

This article mainly introduces the advanced usage of custom capture and custom rendering.

Custom Capture

By default, `trtc.startLocalVideo()` and `trtc.startLocalAudio()` enable camera and microphone capture.

If you need to customize the capture, you can specify the `option.videoTrack` / `option.audioTrack` parameter of the `trtc.startLocalVideo()` / `trtc.startLocalAudio()` method.

There are usually several ways to obtain `audioTrack` and `videoTrack` :

Use getUserMedia to capture the camera and microphone.

Use getDisplayMedia to capture screen sharing.

Use `videoElement.captureStream` to capture the audio and video being played in the video tag.

Use `canvas.captureStream` to capture the animation in the canvas.

Capture the video being played in the video tag

```
// Check if your current browser supports capturing streams from video elements
if (!HTMLVideoElement.prototype.captureStream) {
  console.log('your browser does not support capturing stream from video element');
  return
}
// Get the video tag that is playing video on your page
const video = document.getElementByID('your-video-element-ID');
// Capture the video stream from the playing video
const stream = video.captureStream();
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];

trtc.startLocalVideo({ option:{ videoTrack } });
trtc.startLocalAudio({ option:{ audioTrack } });
```

Capture the animation in the canvas

```
// Check if your current browser supports capturing streams from canvas elements
if (!HTMLCanvasElement.prototype.captureStream) {
  console.log('your browser does not support capturing stream from canvas element')
  return
}
// Get your canvas tag
const canvas = document.getElementByID('your-canvas-element-ID');

// Capture a 15 fps video stream from the canvas
const fps = 15;
const stream = canvas.captureStream(fps);
const videoTrack = stream.getVideoTracks()[0];

trtc.startLocalVideo({ option:{ videoTrack } });
```

Custom Rendering

By default, when calling `trtc.startLocalVideo(view)` or `trtc.startRemoteVideo(view, streamType, userId)` , you need to pass in the `view` parameter. The SDK will create a `video` tag under the specified element tag to play the video.

If you need to customize the rendering, and do not need the SDK to play the video, you can refer to the following steps:

Do not fill in the `view` parameter or pass in `null` when calling the `startLocalVideo` or `startRemoteVideo` method.

Use the `trtc.getVideoTrack(userId, streamType)` method to obtain the corresponding `videoTrack` .

Use your own player for video rendering.

After using this custom rendering method, the `EVENT.VIDEO_PLAY_STATE_CHANGED` event will not be triggered. You need to listen to the `mute/unmute/ended` events of the video track `MediaStreamTrack` to determine the status of the current video data stream.

For remote video, you also need to listen to the `EVENT.REMOTE_VIDEO_AVAILABLE` and `EVENT.REMOTE_VIDEO_UNAVAILABLE` events to handle the lifecycle of remote video.

Custom rendering of local video

```
await trtc.startLocalVideo();
const videoTrack = trtc.getVideoTrack();

// Use your own player for video rendering
const videoElement = document.getElementById('video-element');
videoElement.srcObject = new MediaStream([videoTrack]);
videoElement.play();
```

Custom rendering of remote video

```
trtc.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, async ({ userId, streamType }) => {
  // Only pull the stream, do not play it
  await trtc.startRemoteVideo({ userId, streamType })
  const videoTrack = trtc.getVideoTrack({ userId, streamType });

  // Use your own player for video rendering
  const videoElement = document.getElementById('remote-video-element');
  videoElement.srcObject = new MediaStream([videoTrack]);
  videoElement.play();
});
```

Flutter

Last updated：2023-12-28 21:30:57

This document mainly introduces how to use TRTC Flutter SDK to implement custom audio raw data acquisition.

Acquiring audio raw data

Flutter TRTC SDK provides two ways to acquire audio raw data:

Native access.

Direct use of Flutter's Dart interface.

Since transferring high-frequency and large audio raw data from Native to Dart layer consumes more performance, we recommend using Native access to acquire audio raw data.

1. Native access

The specific access process and access effect can be experienced using the demo.

1.1 Listen to audio raw data at the Native layer and acquire audio raw data.

java

swift

```
void enableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance(getApplicationContext()).setAudioFrameListener(new Aud
    result.success("");
}

void disableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance(getApplicationContext()).setAudioFrameListener(null);
    result.success("");
}

class AudioFrameListener implements TRTCCloudListener.TRTCAudioFrameListener {
    @Override
    public void onCapturedAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {
        // TODO
    }

    @Override
    public void onLocalProcessedAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFra
        // TODO
    }

    @Override
    public void onRemoteUserAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame,
        // TODO
    }

    @Override
    public void onMixedPlayAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {
        // TODO
    }

    @Override
    public void onMixedAllAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {
        // TODO
    }

    @Override
    public void onVoiceEarMonitorAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFr
        // TODO
    }
}
```

```
let listener = AudioFrameProcessListener()
func enableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance().setAudioFrameDelegate(listener)
    result(nil)
}

func disableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance().setAudioFrameDelegate(nil)
    result(nil)
}

class AudioFrameProcessListener: NSObject, TRTCAudioFrameDelegate {
    func onCapturedAudioFrame(_ frame: TRTCAudioFrame) {
        //MARK: TODO
    }

    func onLocalProcessedAudioFrame(_ frame: TRTCAudioFrame) {
        // MARK: TODO
    }

    func onRemoteUserAudioFrame(_ frame: TRTCAudioFrame, userId: String) {
        // MARK: TODO
    }

    func onMixedAllAudioFrame(_ frame: TRTCAudioFrame) {
        // MARK: TODO
    }

    func onMixedPlay(_ frame: TRTCAudioFrame) {
        // MARK: TODO
    }

    func onVoiceEarMonitorAudioFrame(_ frame: TRTCAudioFrame) {
        // MARK: TODO
    }
}
```

1.2 Use Method Channel to implement start/stop acquisition of audio raw data.

Step 1: Implement the start/stop interface for acquiring audio raw data at the Dart layer.

```
  final channel = MethodChannel('TRCT_FLUTTER_EXAMPLE');

  void enableAudioFrame() async {
     await channel.invokeMethod('enableTRTCAudioFrameDelegate');
  }

  void disableAudioFrame() async {
     await channel.invokeMethod('disableTRTCAudioFrameDelegate');
  }
```

Step 2: Implement the start/stop interface for acquiring audio raw data at the Native layer.

java

swift

```
public class MainActivity extends FlutterActivity {
    private static final String channelName = "TRCT_FLUTTER_EXAMPLE";

    private MethodChannel channel;

    @Override
    public void configureFlutterEngine(@NonNull FlutterEngine flutterEngine) {
        super.configureFlutterEngine(flutterEngine);

        channel = new MethodChannel(flutterEngine.getDartExecutor().getBinaryMessen
        channel.setMethodCallHandler(((call, result) -> {
            switch (call.method) {
                case "enableTRTCAudioFrameDelegate":
                    enableTRTCAudioFrameDelegate();
                    break;
                case "disableTRTCAudioFrameDelegate":
                    disableTRTCAudioFrameDelegate();
                    break;
                default:
                    break;
            }
        }));
    }
}
```

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
    var channel: FlutterMethodChannel?

    override func application(_ application: UIApplication,
                              didFinishLaunchingWithOptions launchOptions: [UIAppli
        GeneratedPluginRegistrant.register(with: self)

        guard let controller = window?.rootViewController as? FlutterViewController
            fatalError("Invalid root view controller")
        }
        channel = FlutterMethodChannel(name: "TRCT_FLUTTER_EXAMPLE", binaryMessenge
        channel?.setMethodCallHandler({ [weak self] call, result in
            guard let self = self else { return }
            switch (call.method) {
            case "enableTRTCAudioFrameDelegate":
                self.enableTRTCAudioFrameDelegate()
                break
            case "disableTRTCAudioFrameDelegate":
                self.disableTRTCAudioFrameDelegate()
                break
            default:
                break
            }
        })
        return super.application(application, didFinishLaunchingWithOptions: launch
    }
}
```

2. Access to Flutter layer interface

Currently, the Flutter Dart interface only supports the use of the onCapturedAudioFrame interface. The specific usage is as follows:

```
TRTCCloud trtcCloud = (await TRTCCloud.sharedInstance())!;

//  Start acquiring audio raw data
final audioFrameListener = TRTCAudioFrameListener(
  onCapturedAudioFrame: (audioFrame) {
    // TODO
  }
);
trtcCloud.setAudioFrameListener(audioFrameListener);

// Stop acquiring audio raw data
trtcCloud.setAudioFrameListener(null);
```

Custom Audio Capturing and Playback Android, iOS, Windows, and macOS

Last updated：2023-10-08 15:56:25

This document describes how to use the TRTC SDK to implement custom audio capturing and rendering.

Custom Audio Capturing

The custom audio capturing feature of the TRTC SDK can be used in two steps: enabling the feature and sending audio frames to the SDK. For detailed directions of specific APIs, see below. We also provide API examples for different platforms:

Android

iOS

Windows

Enabling custom audio capturing

To enable the custom audio capturing feature of the TRTC SDK, you need to call the `enableCustomAudioCapture` API of `TRTCCloud`. Below is the sample code:

Android

iOS&Mac

Windows

```
TRTCCloud mTRTCCloud = TRTCCloud.shareInstance();
mTRTCCloud.enableCustomAudioCapture(true);
```

```
self.trtcCloud = [TRTCCloud sharedInstance];
[self.trtcCloud enableCustomAudioCapture:YES];
```

```
liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance();
trtc_cloud->enableCustomAudioCapture(true);
```

Sending custom audio frames

You can use the `sendCustomAudioData` API of `TRTCCloud` to populate the TRTC SDK with your own audio data. Below is the sample code:

Android

iOS&Mac

Windows

```
TRTCCloudDef.TRTCAudioFrame trtcAudioFrame = new TRTCCloudDef.TRTCAudioFrame();
trtcAudioFrame.data = data;
trtcAudioFrame.sampleRate = sampleRate;
trtcAudioFrame.channel = channel;
trtcAudioFrame.timestamp = timestamp;
mTRTCCloud.sendCustomAudioData(trtcAudioFrame);
```

```
TRTCAudioFrame *audioFrame = [[TRTCAudioFrame alloc] init];
audioFrame.channels = audioChannels;
audioFrame.sampleRate = audioSampleRate;
audioFrame.data = pcmData;

[self.trtcCloud sendCustomAudioData:audioFrame];
```

```
liteav::TRTCAudioFrame frame;
frame.audioFormat = liteav::TRTCAudioFrameFormatPCM;
frame.length = buffer_size;
frame.data = array.data();
frame.sampleRate = 48000;
frame.channel = 1;
getTRTCShareInstance()->sendCustomAudioData(&frame);
```

notice

Using `sendCustomAudioData` may cause AEC to fail.

Getting Raw Audio Data

The audio module is a highly complex module, and the TRTC SDK needs to strictly control the capturing and playback logic of audio devices. In some cases, to get the audio data of a remote user or audio captured by the local mic, you can use the APIs of `TRTCCloud` for different platforms. We also provide API examples for those platforms:

Android:

iOS

Windows

Setting audio callback

Android

iOS&Mac

Windows

```
mTRTCCloud.setAudioFrameListener(new TRTCCloudListener.TRTCAudioFrameListener() {
        @Override
        public void onCapturedRawAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFr

        }

        @Override
        public void onLocalProcessedAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudi

        }

        @Override
        public void onRemoteUserAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFra

        }

        @Override
        public void onMixedPlayAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFram

        }

        @Override
        public void onMixedAllAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame
            // For more information, see the custom rendering tool class `com.tence
        }
    });
```

```
 [self.trtcCloud setAudioFrameDelegate:self];
 // MARK: - TRTCAudioFrameDelegate
 - (void)onCapturedRawAudioFrame:(TRTCAudioFrame *)frame {
        NSLog(@"onCapturedRawAudioFrame");
}

- (void)onLocalProcessedAudioFrame:(TRTCAudioFrame *)frame {
        NSLog(@"onLocalProcessedAudioFrame");
}
```

```
- (void)onRemoteUserAudioFrame:(TRTCAudioFrame *)frame userId:(NSString *)userId {
        NSLog(@"onRemoteUserAudioFrame");
}

- (void)onMixedPlayAudioFrame:(TRTCAudioFrame *)frame {
        NSLog(@"onMixedPlayAudioFrame");
}

- (void)onMixedAllAudioFrame:(TRTCAudioFrame *)frame {
        NSLog(@"onMixedAllAudioFrame");
}
```

```
// Set custom audio data callback
liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance();
trtc_cloud->setAudioFrameCallback(callback)

// Callback APIs for custom audio

virtual void onCapturedRawAudioFrame(TRTCAudioFrame* frame) {
}

virtual void onLocalProcessedAudioFrame(TRTCAudioFrame* frame) {
}

virtual void onPlayAudioFrame(TRTCAudioFrame* frame, const char* userId) {
}

virtual void onMixedPlayAudioFrame(TRTCAudioFrame* frame) {
}
```

notice

Do not perform time-consuming operations with any of the above callback functions. We recommend that you copy the data to another thread for processing to avoid AEC failure and choppy audio.

The data called back by the above callback functions can only be read and copied. Modifying the data may lead to unexpected results.

Web

Last updated：2023-10-08 15:56:46

Function Description

This article mainly introduces the advanced usage of custom capture and custom rendering.

Custom Capture

By default, `trtc.startLocalVideo()` and `trtc.startLocalAudio()` enable camera and microphone capture.
If you need to customize the capture, you can specify the `option.videoTrack` / `option.audioTrack` parameter of the `trtc.startLocalVideo()` / `trtc.startLocalAudio()` method.
There are usually several ways to obtain `audioTrack` and `videoTrack` :
Use getUserMedia to capture the camera and microphone.
Use getDisplayMedia to capture screen sharing.
Use `videoElement.captureStream` to capture the audio and video being played in the video tag.
Use `canvas.captureStream` to capture the animation in the canvas.

Capture the video being played in the video tag

```
// Check if your current browser supports capturing streams from video elements
if (!HTMLVideoElement.prototype.captureStream) {
  console.log('your browser does not support capturing stream from video element');
  return
}
// Get the video tag that is playing video on your page
const video = document.getElementByID('your-video-element-ID');
// Capture the video stream from the playing video
const stream = video.captureStream();
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];

trtc.startLocalVideo({ option:{ videoTrack } });
trtc.startLocalAudio({ option:{ audioTrack } });
```

Capture the animation in the canvas

```
// Check if your current browser supports capturing streams from canvas elements
if (!HTMLCanvasElement.prototype.captureStream) {
  console.log('your browser does not support capturing stream from canvas element')
  return
}
// Get your canvas tag
const canvas = document.getElementByID('your-canvas-element-ID');

// Capture a 15 fps video stream from the canvas
const fps = 15;
const stream = canvas.captureStream(fps);
const videoTrack = stream.getVideoTracks()[0];

trtc.startLocalVideo({ option:{ videoTrack } });
```

Custom Rendering

By default, when calling `trtc.startLocalVideo(view)` or `trtc.startRemoteVideo(view, streamType, userId)` , you need to pass in the `view` parameter. The SDK will create a `video` tag under the specified element tag to play the video.

If you need to customize the rendering, and do not need the SDK to play the video, you can refer to the following steps:

Do not fill in the `view` parameter or pass in `null` when calling the `startLocalVideo` or `startRemoteVideo` method.

Use the `trtc.getVideoTrack(userId, streamType)` method to obtain the corresponding `videoTrack` .

Use your own player for video rendering.

After using this custom rendering method, the `EVENT.VIDEO_PLAY_STATE_CHANGED` event will not be triggered. You need to listen to the `mute/unmute/ended` events of the video track `MediaStreamTrack` to determine the status of the current video data stream.

For remote video, you also need to listen to the `EVENT.REMOTE_VIDEO_AVAILABLE` and `EVENT.REMOTE_VIDEO_UNAVAILABLE` events to handle the lifecycle of remote video.

Custom rendering of local video

```
await trtc.startLocalVideo();
const videoTrack = trtc.getVideoTrack();

// Use your own player for video rendering
const videoElement = document.getElementById('video-element');
videoElement.srcObject = new MediaStream([videoTrack]);
videoElement.play();
```

Custom rendering of remote video

```
trtc.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, async ({ userId, streamType }) => {
  // Only pull the stream, do not play it
  await trtc.startRemoteVideo({ userId, streamType })
  const videoTrack = trtc.getVideoTrack({ userId, streamType });

  // Use your own player for video rendering
  const videoElement = document.getElementById('remote-video-element');
  videoElement.srcObject = new MediaStream([videoTrack]);
  videoElement.play();
});
```

Sending and Receiving Messages

Last updated：2024-09-02 17:22:55

Overview

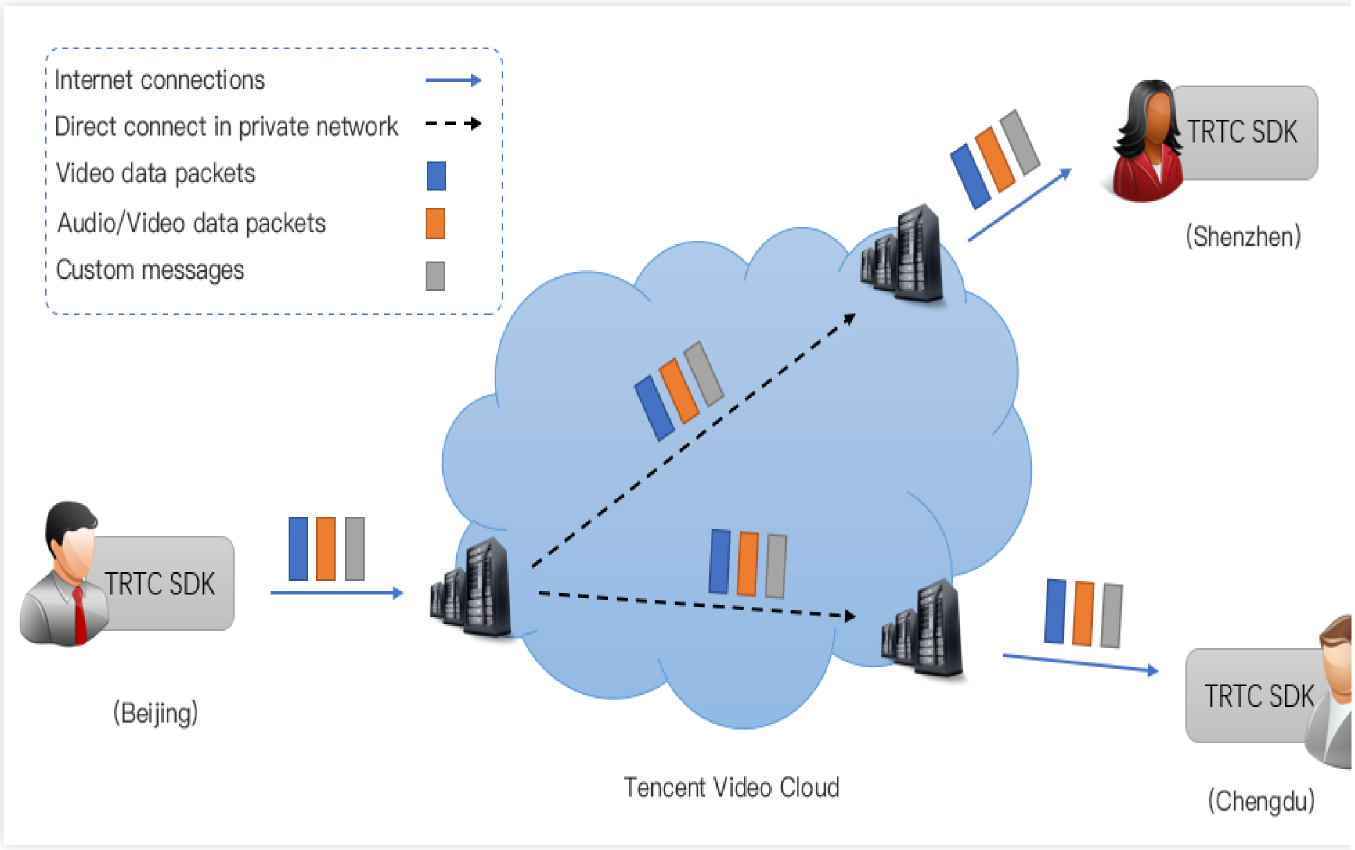
The TRTC SDK provides the ability to send custom messages. With this feature, any user whose role is an anchor can broadcast their own custom messages to other users in the same video room.

Supported Platforms

iOS	Android	macOS	Windows	Electron	Flutter	web
✓	✓	✓	✓	✓	✓	×

How It Works

A user's custom message will be combined into the audio/video data streams and transmitted to other users in the same room alongside. As audio/video channels themselves are not completely reliable, in order to improve the reliability, the TRTC SDK implements a reliability guarantee mechanism internally.



Sending Messages

Messages are sent by calling the `sendCustomCmdMsg` API of TRTCCloud, and the following four parameters need to be specified during sending:

Parameter Name	Description
cmdID	Message ID. Value range: 1–10. Messages in different business types should use different `cmdIDs` .
data	Message to be sent, which can contain up to 1 KB (1,000 bytes) of data.
reliable	Whether reliable sending is enabled; if yes, the receiver needs to temporarily store the data of a certain period to wait for re-sending, which will cause certain delay.
ordered	Whether orderly sending is enabled, i.e., whether the data should be received in the same order in which it is sent; if yes, the receiver needs to temporarily store and sort messages, which will cause certain delay.

Note:

`reliable` and `ordered` must be set to the same value (`YES` or `NO`) and cannot be set to different values currently.

Objective-C

Java

C++

C#

Dart

```
// Sample code for sending a custom message
- (void)sendHello {
    // Command word for the custom message. A set of rules needs to be customized according to the business needs. 0x1 is used as an example to send a text broadcast message
    NSInteger cmdID = 0x1;
    NSData *data = [@"Hello" dataUsingEncoding:NSUTF8StringEncoding];
    // `reliable` and `ordered` need to be consistent for now. Orderly sending is used as an example here
    [trtcCloud sendCustomCmdMsg:cmdID data:data reliable:YES ordered:YES];
}
```

```
// Sample code for sending a custom message
public void sendHello() {
    try {
        // Command word for the custom message. A set of rules needs to be customized according to the business needs. 0x1 is used as an example to send a text broadcast message
        int cmdID = 0x1;
        String hello = "Hello";
        byte[] data = hello.getBytes("UTF-8");
        // `reliable` and `ordered` need to be consistent for now. Orderly sending is used as an example here
        trtcCloud.sendCustomCmdMsg(cmdID, data, true, true);

    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}
```

```
// Sample code for sending a custom message
void sendHello()
```

```
{
    // Command word for the custom message. A set of rules needs to be
customized according to the business needs. 0x1 is used as an example to send a
text broadcast message

    uint32_t cmdID = 0x1;
    uint8_t* data = { '1', '2', '3' };
    uint32_t dataSize = 3;  // Length of data

    // `reliable` and `ordered` need to be consistent for now. Orderly sending
is used as an example here
    trtcCloud->sendCustomCmdMsg(cmdID, data, dataSize, true, true);
}
```

```
// Sample code for sending a custom message
private void sendHello()
{
    // Command word for the custom message. A set of rules needs to be
customized according to the business needs. 0x1 is used as an example to send a
text broadcast message

    uint cmdID = 0x1;
    byte[] data = { '1', '2', '3' };
    uint dataSize = 3;  // Length of data

    // `reliable` and `ordered` need to be consistent for now. Orderly sending
is used as an example here
    mTRTCCloud.sendCustomCmdMsg(cmdID, data, dataSize, true, true);
}
```

```
// Sample code for sending a custom message
sendHello() {
  try {
    // Command word for the custom message. A set of rules needs to be customized a
    int cmdID = 0x1;
    String hello = "Hello";
    // `reliable` and `ordered` need to be consistent for now. Orderly sending is u
    trtcCloud.sendCustomCmdMsg(cmdID, hello, true, true);
  } catch (e) {
    print(e);
  }
}
```

Receiving Messages

After a user in a room uses `sendCustomCmdMsg` to send a custom message, other users in the room can receive the message through the `onRecvCustomCmdMsg` API in the SDK callback.

Objective-C

Java

C++

C#

Dart

```
// Receive and process messages sent by other users in the room
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:(NSInteger)cmdId seq:
(UInt32)seq message:(NSData *)message
{
    // Receive the message sent by `userId`
    switch (cmdId)  // `cmdId` agreed upon between sender and receiver
    {
    case 0:
        // Process the message with `cmdId` = 0
        break;
    case 1:
        // Process the message with `cmdId` = 1
        break;
    case 2:
        // Process the message with `cmdId` = 2
        break;
    default:
        break;
    }
}
```

```
// Inherit `TRTCCloudListener` and implement the `onRecvCustomCmdMsg` method to
receive and process messages sent by others in the room
public void onRecvCustomCmdMsg(String userId, int cmdId, int seq, byte[]
message) {
    // Receive the message sent by `userId`
    switch (cmdId)  // `cmdId` agreed upon between sender and receiver
    {
    case 0:
        // Process the message with `cmdId` = 0
        break;
    case 1:
            // Process the message with `cmdId` = 1
            break;
        case 2:
            // Process the message with `cmdId` = 2
            break;
        default:
            break;

    }
```

```
// Receive and process messages sent by other users in the room
void TRTCCloudCallbackImpl::onRecvCustomCmdMsg(
                            const char* userId, int32_t cmdId, uint32_t seq,
const uint8_t* msg, uint32_t msgSize)
{
    // Receive the message sent by `userId`
    switch (cmdId)  // `cmdId` agreed upon between sender and receiver
    {
    case 0:
        // Process the message with `cmdId` = 0
        break;
    case 1:
        // Process the message with `cmdId` = 1
        break;
    case 2:
        // Process the message with `cmdId` = 2
        break;
    default:
        break;
    }
}
```

```
// Receive and process messages sent by other users in the room
public void onRecvCustomCmdMsg(string userId, int cmdId, uint seq, byte[] msg,
uint msgSize)
{
    // Receive the message sent by `userId`
    switch (cmdId)  // `cmdId` agreed upon between sender and receiver
    {
    case 0:
        // Process the message with `cmdId` = 0
        break;
    case 1:
        // Process the message with `cmdId` = 1
        break;
    case 2:
        // Process the message with `cmdId` = 2
        break;
    default:
        break;
    }
}
```

```
// Register trtc callback
trtcCloud.registerListener(_onRtcListener);

// Implement the onRecvCustomCmdMsg method to receive and process messages sent by
_onRtcListener(type, param) async {
  if (type == TRTCCloudListener.onRecvCustomCmdMsg) {
    // Receive a message sent by userId
    String userId = param['userId'];
    // cmdId agreed upon by the sender and receiver
    switch (param['cmdID']) {
      case 0:
      // Process cmdID = 0 messages
        break;
      case 1:
      // Process cmdID = 1 messages
        break;
      case 2:
      // Process cmdID = 2 messages
        break;
      default:
        break;
    }
  }
}
```

Use Limits

Since custom messages have a higher transmission priority than audio/video data, if too many of them are sent, audio/video data may be interfered with, resulting in video lagging or blurring. Therefore, the following frequency limits apply to custom messages:
As custom messages are broadcast to all users in the same room, up to 30 messages can be sent per second.

A data packet (i.e., data size) can be of up to 1 KB; if the threshold is exceeded, the packet is very likely to be discarded by the intermediate router or server.
A client can send up to 8 KB of data in total per second, that is, if each data packet is of 1 KB, up to 8 packets can be sent per second.

Event Callbacks

Event Callbacks

Last updated：2025-04-02 14:22:51

The event callback service can send notifications about TRTC events in the form of HTTP/HTTPS requests to your server. Currently, you can register callbacks for room events, media events, as well as some recording events (for details about on-cloud recording callbacks, see On-Cloud Recording). To receive such callbacks, you need to configure callback information in the TRTC console.

Callback Information

In order to receive event callback notifications, you need to configure callback information in Tencent RTC Console.

Note:

You need to provide the following information:

Required: An HTTP/HTTPS server address to receive callback notifications.

Optional: A custom key containing up to 32 uppercase and lowercase letters and digits, which is needed for the calculation of signatures.

Timeout and Retry

A notification will be considered failed if the callback server does not receive a response from your server within five seconds of message sending. It will try again immediately after the first failure and retry **10 seconds** after every subsequent failure. The retries will stop one minute after the first try.

Format of Callback Messages

Callbacks are sent to your server in the form of HTTP/HTTPS POST requests.

Character encoding: UTF-8

Request: JSON for the request body

Response: HTTP STATUS CODE = 200. The server ignores the content of the response packet. For protocol-friendliness, we recommend adding `JSON:` {"code":0}` to the response.

Parameters

Callback parameters

The header of a callback message contains the following fields.

Field	Value
Content-Type	application/json
Sign	The signature value.
SdkAppId	The SDK application ID.

The body of a callback message contains the following fields.

Field	Type	Description
EventGroupId	Number	The event group ID.
EventType	Number	The type of the callback event.
CallbackTs	Number	The Unix timestamp (ms) of callback sending.
EventInfo	JSON Object	The event information.

Event group ID

Field	Value	Description
EVENT_GROUP_ROOM	1	Room event group
EVENT_GROUP_MEDIA	2	Media event group

Note:

For on-cloud recording events, see On-Cloud Recording.

Event type

Field	Value	Description
EVENT_TYPE_CREATE_ROOM	101	Creating room
EVENT_TYPE_DISMISS_ROOM	102	Closing room
EVENT_TYPE_ENTER_ROOM	103	Entering room

EVENT_TYPE_EXIT_ROOM	104	Leaving room
EVENT_TYPE_CHANGE_ROLE	105	Switching roles
EVENT_TYPE_START_VIDEO	201	Starting pushing video data
EVENT_TYPE_STOP_VIDEO	202	Stopping pushing video data
EVENT_TYPE_START_AUDIO	203	Starting pushing audio data
EVENT_TYPE_STOP_AUDIO	204	Stopping pushing audio data
EVENT_TYPE_START_ASSIT	205	Starting pushing substream data
EVENT_TYPE_STOP_ASSIT	206	Stopping pushing substream data

Note:

Room exit will trigger only the `104` callback and not the `202` or `204` callback. `202` and `204` are triggered only if a user manually turns their video and audio off.

Event Callback Example

101
102
103
104
105
201
202
203
204
205
206

```
{        "EventGroupId": 1,
         "EventType":    101,
         "CallbackTs":   1687770730166,
         "EventInfo":    {
                 "RoomId":       12345,
         "EventTs":      1687770730,
         "EventMsTs":    1687770730160,
         "UserId":       "test"
         }
}
```

```
{
    "EventGroupId":     1,
    "EventType":        102,
    "CallbackTs":       1687771618531,
    "EventInfo":        {
        "RoomId":       "12345",
        "EventTs":      1687771618,
        "EventMsTs":    1687771618457
     }
}
```

```
{
        "EventGroupId": 1,
        "EventType":    103,
        "CallbackTs":   1687770731932,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687770731,
                "EventMsTs":    1687770731831,
                "UserId":       "test",
                "Role": 21,
                "TerminalType": 2,
                "UserType":     3,
                "Reason":       1
        }
}
```

```
{
        "EventGroupId": 1,
        "EventType":    104,
        "CallbackTs":   1687770731922,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687770731,
                "EventMsTs":    1687770731898,
                "UserId":       "test",
                "Role": 20,
                "Reason":       1
        }
}
```

```
{
        "EventGroupId": 1,
        "EventType":    105,
        "CallbackTs":   1687772245596,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687772245,
                "EventMsTs":    1687772245537,
                "UserId":       "test",
                "Role": 21
        }
}
```

```
{
        "EventGroupId": 2,
        "EventType":    201,
        "CallbackTs":   1687771803198,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687771803,
                "EventMsTs":    1687771803192,
                "UserId":       "test"
        }
}
```

```
{
        "EventGroupId": 2,
        "EventType":    202,
        "CallbackTs":   1687771919458,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687771919,
                "EventMsTs":    1687771919447,
                "UserId":       "test",
                "Reason":       0
        }
}
```

```
{
        "EventGroupId": 2,
        "EventType":    203,
        "CallbackTs":   1687771869377,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687771869,
                "EventMsTs":    1687771869365,
                "UserId":       "test"
        }
}
```

```
{
        "EventGroupId": 2,
        "EventType":    204,
        "CallbackTs":   1687770732498,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687770732,
                "EventMsTs":    1687770732383,
                "UserId":       "test",
                "Reason":       0
        }
}
```

```
{
        "EventGroupId": 2,
        "EventType":    205,
        "CallbackTs":   1687772013823,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687772013,
                "EventMsTs":    1687772013753,
                "UserId":       "test"
        }
}
```

```
{
        "EventGroupId": 2,
        "EventType":    206,
        "CallbackTs":   1687772015054,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687772015,
                "EventMsTs":    1687772015032,
                "UserId":       "test",
                "Reason":       0
        }
}
```

Event information

Field	Type	Description
RoomId	String/Number	The room ID, which is of the same type as the room ID on the client.

EventTs	Number	The Unix timestamp (seconds) of event occurrence. This field is reserved for compatibility purposes.
EventMsTs	Number	The Unix timestamp (ms) of event occurrence.
UserId	String	User ID
UniqueId	Number	The unique identifier of an event (optional), which is valid for the room event group. When a user experiences unusual events such as network change or abnormal exit and reentry, your server may receive multiple callbacks for the entry and exit of the same user. A unique identifier helps identify a room entry or exit.
Role	Number	The role type (optional), which is valid for the room entry/exit callback.
TerminalType	Number	The device type (optional), which is valid for the room entry callback.
UserType	Number	The user type (optional), which is valid for the room entry callback.
Reason	Number	The reason (optional), which is valid for the room entry/exit callback.

Note:
We have developed a policy that prevents repeated callbacks resulting from unusual events on the client. If you start using the callback service after July 30, 2021, the policy will apply by default, and the room event group will no longer carry UniqueId.

Role type

Field	Value	Description
MEMBER_TRTC_ANCHOR	20	Anchor
MEMBER_TRTC_VIEWER	21	Audience

Device type

Field	Value	Description
TERMINAL_TYPE_WINDOWS	1	Windows
TERMINAL_TYPE_ANDROID	2	Android
TERMINAL_TYPE_IOS	3	iOS
TERMINAL_TYPE_LINUX	4	Linux

TERMINAL_TYPE_OTHER	100	Other

User type

Field	Value	Description
USER_TYPE_WEBRTC	1	WebRTC
USER_TYPE_APPLET	2	Mini Program
USER_TYPE_NATIVE_SDK	3	Native SDK

Reason

Field	Description
Room entry	1: Voluntary entry 2: Network change 3: Timeout and retry 4: Cross-room communication
Room exit	1: Voluntary exit 2: Timeout 3: Removed from the room 4: Cross-room communication was canceled 5: The process was force-closed **Note: TRTC cannot capture a force-close event on Android and will send a callback only after timeout (`reason` = `2`).**

Signature calculation

Signatures are calculated using the HMAC SHA256 encryption algorithm. Upon receiving a callback message, your server will calculate a signature using the same method, and if the results match, it indicates that the callback is from TRTC and not forged. See below for the calculation method.

```
// In the formula below, `key` is the key used to calculate a signature.
Sign = base64(hmacsha256(key, body))
```

Note:

`body` is the original packet body of the callback request you receive. Do not make any modifications. Below is an example.

```
body="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t103,\\n\\t\\"Callback
```

Verify signature example

Java

Python

PHP

Golang

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# Function: Third-party callback sign verification
//# Parameters:
//#   key: The key configured in the console
//#   body: The body returned by the Tencent Cloud callback
//#   sign: The sign value returned by the Tencent Cloud callback
//# Return Value:
//#   Status: OK indicates that the verification has passed, FAIL indicates that th
//#   Info: Success/Failure information

public class checkSign {
    public static String getResultSign(String key, String body) throws Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.init(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes(
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': 'validation passed'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': 'validation failed'}");
        }
    }
}
```

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# Function: Third-party callback sign verification
# Parameters:
#   key: The key configured in the console
#   body: The body returned by the Tencent Cloud callback
#   sign: The sign value returned by the Tencent Cloud callback
# Return Value:
#   Status: OK indicates that the verification has passed, FAIL indicates that the
#   Info: Success/Failure information

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = 'validation passed'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = 'validation failed'
        return temp_dict

if __name__ == '__main__':
    key = '123654'
    body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\\":\\t204,\
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
                $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
                return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\\"Callbac

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

```
package main
import "fmt"
import (
        "crypto/hmac"
        "crypto/sha256"
        "encoding/base64"
)

func main () {
    var data = "{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\
    var key = "789"

    //JSRUN engine 2.0, supporting up to 30 types of languages for online running,
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
        h := hmac.New(sha256.New, []byte(key))
        h.Write([]byte(data))
        return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

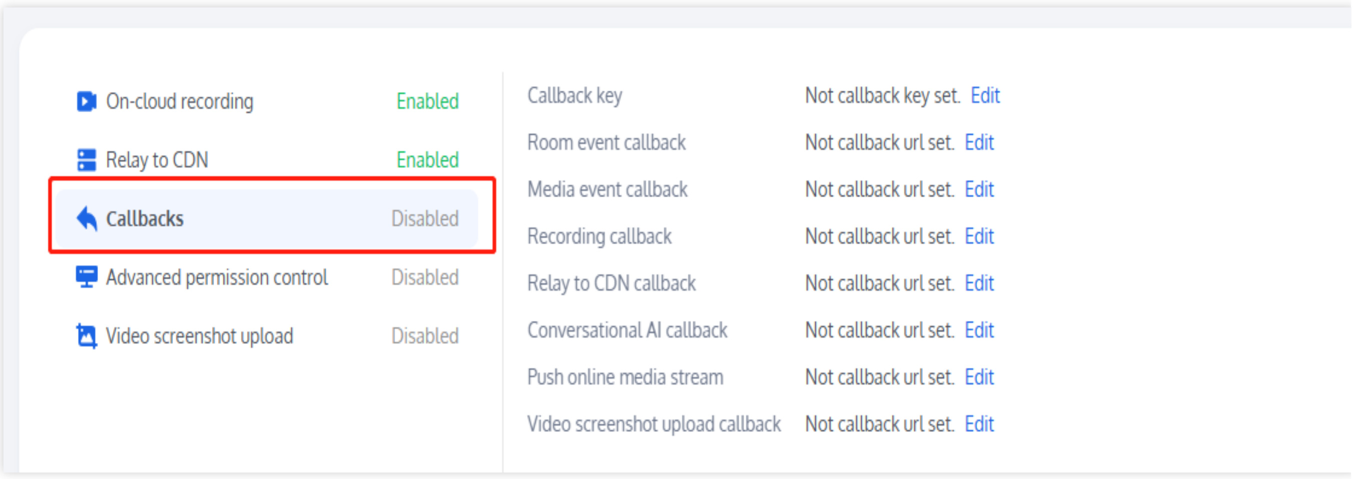
Relay to CDN Callback

Last updated：2024-11-06 18:24:41

The server relay to CDN callback supports notifying your server of the events generated by the relay to CDN REST API in the form of HTTP/HTTPS requests. To receive such callbacks, you need to configure callback information in the TRTC console.

Callback Information

In order to receive event callback notifications, you need to configure callback information in the Tencent RTC Console.



On-cloud recording	Enabled
Relay to CDN	Enabled
Callbacks	Disabled
Advanced permission control	Disabled
Video screenshot upload	Disabled

Callback key	Not callback key set. Edit
Room event callback	Not callback url set. Edit
Media event callback	Not callback url set. Edit
Recording callback	Not callback url set. Edit
Relay to CDN callback	Not callback url set. Edit
Conversational AI callback	Not callback url set. Edit
Push online media stream	Not callback url set. Edit
Video screenshot upload callback	Not callback url set. Edit

Note：

You need to provide the following information:

Required: An HTTP/HTTPS server address to receive callback notifications.

Optional: A custom key containing up to 32 uppercase and lowercase letters and digits, which is needed for the calculation of signatures.

Timeout and Retry

A notification will be considered failed if the callback server does not receive a response from your server within five seconds of message sending. It will try again immediately after the first failure and retry **10 seconds** after every subsequent failure. The retries will stop one minute after the first try.

Format of Callback Messages

Callbacks are sent to your server in the form of HTTP/HTTPS POST requests.

Character encoding: UTF-8

Request: JSON for the request body

Response: HTTP STATUS CODE = 200. The server ignores the content of the response packet. For protocol-friendliness, we recommend adding `JSON:` {"code":0}` to the response.

Packet Example: Below is a packet example for the "relay to CDN Event Group - CDN Streaming in Progress" event.

```
{
    "EventGroupId": 4,
    "EventType": 401,
    "CallbackTs": 1622186275913,
    "EventInfo": {
        "RoomId": "xx",
        "RoomType": 1,
        "EventTsMs": 1622186275913,
        "UserId": "xx",
        "TaskId": "xx",
        "Payload": {
            "Url": "rtmp://tencent-url/xxxx"
            "Status": 2                     //indicates that the CDNs push in progr
        }
    }
}
```

Parameters

Callback parameters

The header of a callback message contains the following fields.

Field	Value
Content-Type	application/json
Sign	The signature value.
SdkAppId	The SDK application ID.

The body of a callback message contains the following fields.

Field	Type	Description

EventGroupId	Number	The event group ID, mix relay event fixed as 4.
EventType	Number	The type of the callback event.
CallbackTs	Number	The Unix timestamp (ms) of callback sending.
EventInfo	JSON Object	The event information.

Event group ID

Field	Value	Description
EVENT_GROUP_CLOUD_PUBLISH	4	Relay event group

Event type

Field	Value	Description
EVENT_TYPE_CLOUD_PUBLISH_CDN_STATUS	401	Cloud relay CDN status callback

Event information

Field	Type	Description
RoomId	String/Number	Room ID (Type consistent with client-side room ID type)
RoomType	Number	0 represents numeric room ID, 1 represents string room ID
EventMsTs	Number	Event's Unix timestamp, unit in milliseconds
UserId	String	User ID of the companion robot specified when initiating the task (AgentParams.UserId)
TaskId	Number	Task ID
Payload	JSON Object	Learn more about the event

Payload (Learn more)

Field	Value	Description
Url	String	Push destination URL

Status	Number	Relay status
ErrorCode	Number	Error code
ErrorMsg	String	Error message

Relay status

Field	Value	Description	Callback Frequency
PUBLISH_CDN_STREAM_STATE_IDLE	0	Push not started or ended	Callback only once
PUBLISH_CDN_STREAM_STATE_CONNECTING	1	Connecting TRTC Server and CDN Server	Callback every 5 seconds, no more callbacks after 60 seconds timeout
PUBLISH_CDN_STREAM_STATE_RUNNING	2	CDNs push in progress	Callback only once
PUBLISH_CDN_STREAM_STATE_RECOVERING	3	TRTC server and CDN server push interrupted, recovering	Callback every 5 seconds, no more callbacks after 60 seconds timeout
PUBLISH_CDN_STREAM_STATE_FAILURE	4	TRTC server and CDN server push interrupted, and recovery or connection timeout	Callback only once
PUBLISH_CDN_STREAM_STATE_DISCONNECTING	5	Disconnecting TRTC Server and CDN Server	Callback only once

Relay status recommendation processing

Status	Processing Method

PUBLISH_CDN_STREAM_STATE_IDLE	Indicates URL removal successful, no need to handle.
PUBLISH_CDN_STREAM_STATE_CONNECTING	Indicates URL is connecting, callback every 5s, until connected successfully with `PUBLISH_CDN_STREAM_STATE_RUNNING` , or after 60s callback `PUBLISH_CDN_STREAM_STATE_FAILURE` . You can replace the problematic URL when receiving `PUBLISH_CDN_STREAM_STATE_FAILURE` , and call `UpdatePublishCdnStream` to update Publish parameters. If your business is time-sensitive, you can replace the problematic URL after receiving 2 or more `PUBLISH_CDN_STREAM_STATE_CONNECTING` callbacks, and call `UpdatePublishCdnStream` to update Publish parameters.
PUBLISH_CDN_STREAM_STATE_RUNNING	Indicates URL push successful, no need to handle.
PUBLISH_CDN_STREAM_STATE_RECOVERING	Indicates an interruption occurred during the push process, reconnecting, callback every 5s, until reconnected successfully with `PUBLISH_CDN_STREAM_STATE_RUNNING` , or after 60s callback `PUBLISH_CDN_STREAM_STATE_FAILURE` . Usually caused by network jitter, no need to handle. If `PUBLISH_CDN_STREAM_STATE_RECOVERING` and `PUBLISH_CDN_STREAM_STATE_RUNNING` appear alternately in a short time, you need to check if there are multiple tasks using the same push URL.
PUBLISH_CDN_STREAM_STATE_FAILURE	Indicates push URL connection failed or failed to recover push within 60s, you can replace the problematic URL and call `UpdatePublishCdnStream` to update Publish parameters.
PUBLISH_CDN_STREAM_STATE_DISCONNECTING	Indicates that the push URL is being removed, and after removal successful, it will callback PUBLISH_CDN_STREAM_STATE_IDLE, no need to handle.

Basic Callback Transfer Example

Initiate relay/add relay address to relay success event transfer

`PUBLISH_CDN_STREAM_STATE_CONNECTING` -> `PUBLISH_CDN_STREAM_STATE_RUNNING`

Stop relay/delete relay address to stop relay success event transfer

`PUBLISH_CDN_STREAM_STATE_RUNNING` -> `PUBLISH_CDN_STREAM_STATE_DISCONNECTING` -> `PUBLISH_CDN_STREAM_STATE_IDLE`

During the relay process, connection failure to retry connection success event transfer

`PUBLISH_CDN_STREAM_STATE_RUNNING` -> `PUBLISH_CDN_STREAM_STATE_RECOVERING` -> `PUBLISH_CDN_STREAM_STATE_RUNNING`

During the relay process, connection failure to retry connection timeout failure event transfer

`PUBLISH_CDN_STREAM_STATE_RUNNING` -> `PUBLISH_CDN_STREAM_STATE_RECOVERING` -> `PUBLISH_CDN_STREAM_STATE_FAILURE` -> `PUBLISH_CDN_STREAM_STATE_IDLE`

Note：

Push callback may arrive at your callback server out of order. In this case, you need to sort the events based on the EventMsTs in EventInfo. If you only care about the latest status of the URL, you can ignore the expired events that arrive later.

Signature calculation

Signatures are calculated using the HMAC SHA256 encryption algorithm. Upon receiving a callback message, your server will calculate a signature using the same method, and if the results match, it indicates that the callback is from TRTC and not forged. See below for the calculation method.

```
// In the formula below, `key` is the key used to calculate a signature.
Sign = base64(hmacsha256(key, body))
```

Note:

`body` is the original packet body of the callback request you receive. Do not make any modifications. Below is an example.

```
body="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t103,\\n\\t\\"Callback
```

Verify signature example

Java

Python

PHP

Golang

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# Function: Third-party callback sign verification
//# Parameters:
//#   key: The key configured in the console
//#   body: The body returned by the Tencent Cloud callback
//#   sign: The sign value returned by the Tencent Cloud callback
//# Return Value:
//#   Status: OK indicates that the verification has passed, FAIL indicates that th
//#   Info: Success/Failure information

public class checkSign {
    public static String getResultSign(String key, String body) throws Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.init(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes(
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': 'validation passed'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': 'validation failed'}");
        }
    }
}
```

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# Function: Third-party callback sign verification
# Parameters:
#   key: The key configured in the console
#   body: The body returned by the Tencent Cloud callback
#   sign: The sign value returned by the Tencent Cloud callback
# Return Value:
```

```
#   Status: OK indicates that the verification has passed, FAIL indicates that the
#   Info: Success/Failure information

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = 'validation passed'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = 'validation failed'
        return temp_dict

if __name__ == '__main__':
    key = '123654'
    body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\\":\\t204,\
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
                $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
                return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
```

```
$data="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\\"Callbac

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

```
package main
import "fmt"
import (
        "crypto/hmac"
        "crypto/sha256"
        "encoding/base64"
)

func main () {
    var data = "{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\
    var key = "789"

    //JSRUN engine 2.0, supporting up to 30 types of languages for online running,
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
        h := hmac.New(sha256.New, []byte(key))
        h.Write([]byte(data))
        return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

Cloud Recording Callback

Last updated：2025-03-27 15:26:00

This document describes the callback events of the updated cloud recording feature.

Configuration Information

On the Tencent RTC Console, you can configure callback information. Upon configuration completion, you can receive event callback notifications.

Note：

You need to prepare the following information in advance and complete the Callback Configuration in the console.

Required: the HTTP/HTTPS server address to receive callback notifications.

Optional: Key for signature computation. You can customize a key of up to 32 characters, composed of uppercase and lowercase letters and numbers.

Timeout Retry

If your server does not respond within 5 seconds after the event callback server sends the message notification, it is deemed as a failed notification. If the initial notification fails, an immediate retry is performed. If the notification fails again, retry will be performed at an interval of **10 seconds** until the message has been kept for 1 minute, after which retries will not be performed.

Callback API

You can assign an HTTP/HTTPS service gateway for subscribing to callback messages. When any related event happens, the cloud recording system calls back the event notification to your message receiving server.

Format of the Event Callback Message

Event callback messages are sent to your server via HTTP/HTTPS POST requests, in which:

Character Encoding Format: UTF-8.

Request: The body is in the JSON format.

Response: HTTP STATUS CODE = 200. The server ignores the specific content of the response package. To ensure protocol friendliness, it is recommended that the customer put the following in the response content: JSON: {"code":0}.

Parameter Description

The Header of the Event Callback Message Contains the Following Fields:

Field name	Value
Content-Type	application/json
Sign	Signature value
SdkAppId	sdk application id

The Body of the Event Callback Message Includes the Following Fields:

Field Name	Type	Description
EventGroupId	Number	Event group ID, fixed at 3 for cloud recording
EventType	Number	Event type for callback notification
CallbackTs	Number	The Unix timestamp (in milliseconds) when the event callback server sends a callback request to your server
EventInfo	JSON Object	The event information

Event Type Description:

Field Name	Type	Description
EVENT_TYPE_CLOUD_RECORDING_RECORDER_START	301	The cloud recording module starts.
EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP	302	The cloud recording module exits.
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START	303	The cloud recording file upload task starts, and it is called back only when COS is chosen.
EVENT_TYPE_CLOUD_RECORDING_FILE_INFO	304	Cloud recording: Generating the M3U8

		index file. After the first successful generation and upload, it is called back only when COS is chosen through API recording.
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP	305	The cloud recording file upload is complete. It is called back only when COS is chosen.
EVENT_TYPE_CLOUD_RECORDING_FAILOVER	306	Cloud recording migration occurs. It is triggered when the existing recording task is migrated to a new load.
EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE	307	Cloud recording: Generating the M3U8 index file (generating the first ts slice). After generation, it is called back only when COS is chosen through API recording.
EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR	309	An error occurs when the cloud recording attempts to download and decode the image file.
EVENT_TYPE_CLOUD_RECORDING_MP4_STOP	310	The MP4 recording task of cloud recording stops, and it is called back only when COS is chosen via API recording (when automatic recording is turned on in the console and the authorized VOD COS is selected as storage, please pay attention to event 311).
EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT	311	The cloud recording VOD recording task has completed the upload of

		media resources, and it is called back when you select **Video on Demand** and **automatically recording to COS** via the console (after file recording ends, the VOD index information is carried. Please subscribe to this type of callback event).
EVENT_TYPE_CLOUD_RECORDING_VOD_STOP	312	The cloud recording VOD task stops, and it is called back only when VOD is chosen.

Note：

The callback statuses from event types 301 to 309 are intermediate states of real-time recording, for you to better understand the recording process and keep track of the status. The successful upload of the actual recording file to video on demand will trigger an event 311 callback, and the overall task is completed and an event 312 is called back.

Event Information Description:

Field Name	Type	Description
RoomId	String/Number	The room name, consistent with the room ID type on the client side
EventTs	Number	The Unix timestamp of when the event occurred, in seconds (this field is not recommended. Instead, EventMsTs is recommended)
EventMsTs	Number	The Unix timestamp of when the event occurred, in milliseconds
UserId	String	The user ID of the recording robot
TaskId	String	The recording ID, which is a unique ID of a single cloud recording task
Payload	JsonObject	Defined based on various event types

When the event type is

301

(EVENT_TYPE_CLOUD_RECORDING_RECORDER_START), the definition of Payload is as follows:

Field Name	Type	Description
Status	Number	0: Indicates that the recording module successfully starts. 1: Indicates that the recording module fails to start.

```
{
    "EventGroupId": 3,
    "EventType": 301,
    "CallbackTs": 1622186275913,
    "EventInfo": {
        "RoomId": "xx",
        "EventTs": "1622186275",
        "EventMsTs": 1622186275757,
        "UserId": "xx",
        "TaskId": "xx",
        "Payload": {
            "Status": 0
        }
    }
}
```

When the event type is

302

(EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP), the definition of Payload is as follows:

Field Name	Type	Description
LeaveCode	Number	0: The invocation of recording module normally stops and exits. 1: The customer kicks out the recording robot from the room. 2: The customer disbands the room. 3: The server kicks out the recording robot from the room. 4: The server disbands the room. 99: There is no other user flow in the room except for the recording robot, which will exit after a specified time. 100: Exit from the room due to timeout. 101: Repeated entry of the same user into the same room causes the robot to exit.

```
{
  "EventGroupId": 3,
  "EventType": 302,
  "CallbackTs": 1622186354806,
  "EventInfo": {
    "RoomId": "xx",
  "EventTs": "1622186354",
  "EventMsTs": 1622186275757,
  "UserId": "xx",
  "TaskId": "xx",
  "Payload": {
    "LeaveCode": 0
  }
}
}
```

When the event type is

303

(EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START), the definition of Payload is as follows:

Field Name	Type	Description
Status	Number	0: Indicates that the upload module normally starts. 1: Indicates that the upload module fails to be initiated.

When the event type is

304

(EVENT_TYPE_CLOUD_RECORDING_FILE_INFO), the definition of Payload is as follows:

Field Name	Type	Description
FileList	String	The generated M3U8 filename

When the event type is

305

(EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP), the definition of Payload is as follows:

Field Name	Type	Description
LeaveCode	Number	0: Indicates that this recording upload task has been completed, and all files have been uploaded to the specified third-party cloud storage. 1: Indicates that this recording upload task has been completed, but at least one file is lingering on the server or backup storage. 2: Indicates that files lingering on the server or backup storage have been restored and uploaded to the designated third-party cloud storage. Note: 305 indicates the event that the HLS file upload is complete.

When the event type is

306
(EVENT_TYPE_CLOUD_RECORDING_FAILOVER), the definition of Payload is as follows:

Field Name	Type	Description
Status	Number	0: Indicates that this migration is completed.

```
{
  "EventGroupId": 3,
  "EventType": 306,
  "CallbackTs": 1622191989674,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191989,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0
    }
  }
}
```

When the event type is
307
(EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE), the definition of Payload is as follows:

Field Name	Type	Description
FileName	String	M3U8 filename
UserId	String	The user ID corresponding to the recorded file
TrackType	String	Audio/Video types: audio/video/audio_video
BeginTimeStamp	String	The Unix timestamp of the server when the recording starts (in milliseconds)

When the event type is
309
(EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR), the definition of Payload is as follows:

Field Name	Type	Description
Url	String	The URL of the failed download

```
{
  "EventGroupId": 3,
  "EventType": 309,
  "CallbackTs": 1622191989674,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191989,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Url": "http://xx"
    }
  }
}
```

When the event type is

310

(EVENT_TYPE_CLOUD_RECORDING_MP4_STOP), the definition of Payload is as follows:

Note：

310 is a callback event after an MP4 file is uploaded to the user-specified third-party COS. A recording task may call back multiple events of 310 (each event corresponds to a recorded file information).

Field Name	Type	Description
Status	Number	0: Indicates that the MP4 recording task has exited normally, and all files have been uploaded to the designated third-party cloud storage. 1: Indicates that this MP4 recording task has exited normally, but at least one file lingers on the server or backup storage. 2: Indicates that this MP4 recording task exits abnormally (the possible reason is the failure of extracting HLS files from COS).
FileList	Array	All generated MP4 file names
FileMessage	Array	All generated MP4 file information
FileName	String	MP4 file name
UserId	String	The user ID corresponding to the MP4 file (this field is empty when the recording mode is set to mixed streaming mode)
TrackType	String	audio for audio / video for pure video / audio_video for audio and video
MediaId	String	The primary and auxiliary stream tag. main indicates the primary stream (camera), aux indicates the auxiliary streams (screen

		sharing), and mix indicates mixed stream recording.
StartTimeStamp	Number	The Unix timestamp at the beginning of the MP4 file (in milliseconds)
EndTimeStamp	Number	The UNIX timestamp at the end of the MP4 file (in milliseconds)

```
{
  "EventGroupId": 3,
  "EventType": 310,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191989,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0,
      "FileList": ["xxxx1.mp4", "xxxx2.mp4"],
      "FileMessage": [
        {
            "FileName": "xxxx1.mp4",
            "UserId": "xxxx",
            "TrackType": "audio_video",
            "MediaId": "main",
            "StartTimeStamp": 1622186279145,
            "EndTimeStamp": 1622186282145
        },
        {
            "FileName": "xxxx2.mp4",
            "UserId": "xxxx",
            "TrackType": "audio_video",
            "MediaId": "main",
            "StartTimeStamp": 1622186279153,
            "EndTimeStamp": 1622186282153
        }
     ]

    }
  }
}
```

When the event type is

311

(EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT), the definition of Payload is as follows:

Field Name	Type	Description
Status	Number	0: Indicates that the recorded file has been successfully uploaded to the VOD platform. 1: Indicates that the recorded file lingers on the server or backup storage. 2: Indicates that the upload and VOD task for this recorded file to is abnormal.
UserId	String	The user ID corresponding to this recorded file (this field is empty when the recording mode is set to mixed stream mode)
TrackType	String	audio for audio / video for pure video / audio_video for audio and video
MediaId	String	The primary and auxiliary stream tag. main indicates the primary stream (camera), aux indicates the auxiliary stream (screen sharing), and mix indicates mixed stream recording.
FileId	String	The unique ID of this recorded file in the VOD platform
VideoUrl	String	The playback address of this recorded file on the VOD platform
CacheFile	String	The filename corresponding to this MP4/HLS recording file
StartTimeStamp	Number	The Unix timestamp of the beginning of this recorded file (in milliseconds)
EndTimeStamp	Number	The Unix timestamp of the end of this recorded file (in milliseconds)
Errmsg	String	Corresponding error message when the status is not 0

Callback for successful upload:

```
{
  "EventGroupId": 3,
  "EventType": 311,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0,
      "TencentVod": {
        "UserId": "xx",
        "TrackType": "audio_video",
        "MediaId": "main",
        "FileId": "xxxx",
        "VideoUrl": "http://xxxx",
        "CacheFile": "xxxx.mp4",
        "StartTimeStamp": 1622186279153,
        "EndTimeStamp": 1622186282153
      }
    }
  }
}
```

Callback for upload failure:

```
  {
    "EventGroupId": 3,
    "EventType": 311,
    "CallbackTs": 1622191965320,
    "EventInfo": {
      "RoomId": "20015",
      "EventTs": 1622191965,
      "EventMsTs": 1622186275757,
      "UserId": "xx",
      "TaskId": "xx",
      "Payload": {
        "Status": 1,
        "Errmsg": "xxx",
        "TencentVod": {
          "UserId": "123",
          "TrackType": "audio_video",
          "CacheFile": "xxx.mp4"
        }
      }
    }
  }
```

Note：

After the callback of 311 is received, the **file upload is completed,** and you need to wait for 30 seconds to 3 minutes for the file to be completely recorded, depending on the file size.

When the event type is

312

(EVENT_TYPE_CLOUD_RECORDING_VOD_STOP), the definition of Payload is as follows:

Field Name	Type	Description

Status	Number	0: Indicates that this VOD upload task has exited normally. 1: Indicates that this VOD upload task exits abnormally.

```
{
  "EventGroupId": 3,
  "EventType": 312,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0
    }
  }
}
```

Calculating a Signature

Signatures are calculated using the HMAC SHA256 encryption algorithm. After your event callback server receives the callback message, it calculates the signature in the same manner. If they match, it is an event callback from Tencent Real-Time Communication (TRTC), not a forged one. The signature calculation is as follows:

```
// In the signature Sign calculation formula, key is the encryption key used for ca
Sign = base64(hmacsha256(key, body))
```

Note：

The body is the original package body of the callback request received by you. Do not convert it. See the following example:

```
body="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t103,\\n\\t\\"Callback
```

Signature Verification Example

Java
Python
PHP
Golang

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# Function: Third-party callback sign verification
//# Parameters:
//#   key: The key configured in the console
//#   body: The body returned in Tencent Cloud callback
//#   sign: Signature value of sign returned in Tencent Cloud callback
//# Returned values:
//#   Status: OK indicates successful verification, and FAIL indicates verification
//#   Info: Success/Failure message

public class checkSign {
    public static String secureFinalSign(String key, String entityBody) throws Exce
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.initialize(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes(
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = obtainResultSignature(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': 'Verification passed'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': 'Verification failed'}")
        }
    }
}
```

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# Function: Third-party callback sign verification
# Parameters:
#   key: The key configured in the console
#   body: The body returned in Tencent Cloud callback
#   Sign: The signature value sign returned in Tencent Cloud's callback
# Returned values:
# Status: OK indicates successful verification, and FAIL indicates verification fai
#   Info: Success/Failure Information

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'
    print(computSign)
    if computSign equals sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = 'Verification passed'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = 'Verification failed'
        return temp_dict

if __name__ == '__main__':
    key = '123654'
    body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\\":\\t204,\
    `sign` = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = verifySignature(key, body, sign)
    print(result)
```

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
                $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
                return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\\"Callbac

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

```
package main
import "fmt"
import (
        "crypto/hmac"
        "crypto/sha256"
        "encoding/base64"
)

func main () {
    var data = "{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\
    var key = "789"

    //JSRUN engine 2.0, supporting up to 30 types of languages for online running,
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
        h := hmac.New(sha256.New, []byte(key))
        h.Write([]byte(data))
        return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

Push Online Media Stream Callback

Last updated：2024-08-01 16:20:02

The server-side push online media stream callback supports notifying your server of push online media stream events generated using the Push Online Media Stream REST API in the form of HTTP/HTTPS requests. You can provide Tencent Cloud with relevant configuration information to enable this service.

Configuration Information

The TRTC console supports self-service configuration of callback information. Once the configuration is completed, you can receive event callback notifications. For detailed operation instructions, see the Callback Configuration.

Note:

You need to prepare the following information in advance:

Required item: An HTTP/HTTPS server address to receive callback notifications.

Optional item: A key for signature calculation, which is a key of up to 32 characters defined by you and consists of uppercase and lowercase letters and numbers.

Timeout Retry

If the event callback server does not receive a response from your server within 5 seconds after sending a message notification, the notification is considered to have failed. After the first notification failure, a retry will be made immediately. Subsequent retries will occur at 10-second intervals until the message retention time exceeds 1 minute, after which no further retries will be made.

Format of Event Callback Messages

Event callback messages are sent to your server via HTTP/HTTPS POST requests, where:

Character Encoding Format: UTF-8.

Request: The body is in the JSON format.

Response: HTTP STATUS CODE = 200. The server ignores the specific content of the response package. For the sake of protocol friendliness, it is recommended that the customer response carry JSON: {"code":0}.

Package Body Example: Below is a package body example for the "push online media stream started successfully" event.

```
{
```

```
          "EventGroupId": 7,
          "EventType":    701,
          "CallbackMsTs":   1701937900012,
          "EventInfo":    {
                  "EventMsTs": 1701937900013,
                  "TaskId":"xx",
                  "Status":0
          }
  }
```

Description of Parameters

Callback Message Parameters

The header of the event callback message contains the following fields:

Field name	Value
Content-Type	application/json
Sign	Signature value
SdkAppId	sdk application id

The body of the event callback message includes the following fields:

Field name	Type	Meaning
EventGroupId	Number	Event group ID, which is 4 for a stream mixing and relaying event
EventType	Number	The event type of the callback notification
CallbackMsTs	Number	The Unix timestamp of the callback request sent by the event callback server to your server, in milliseconds
EventInfo	JSON Object	Event information

Event Group ID

Field name	Value	Meaning
EVENT_GROUP_STREAM_INGEST	7	Push online media stream event group

Event Type

Field name	Value	Meaning
EVENT_TYPE_STREAM_INGEST_START	701	Push online media stream start
EVENT_TYPE_STREAM_INGEST_STOP	702	Push online media stream stop

Definition of Event Information when the Event Type is (EVENT_TYPE_STREAM_INGEST_START 701):

Field name	Type	Meaning
EventMsTs	String	The Unix timestamp of the event occurred, in milliseconds
TaskId	String	Push online media stream task ID
Status	Number	Status of push online media stream start

Status of Push Online Media Stream Start

Field name	Value	Meaning	Callback frequency
STATUS_START_SUCCESS	0	The push online media stream start succeeded.	Callback is made once upon success.
STATUS_START_FAILURE	1	The push online media stream start failed.	Callback is made once upon failure.
STATUS_START_AGAIN	2	The push online media stream starts again.	A retry is made at the 0th, 1st, and 3rd second, with callback during the retry.

Recommended Handling of Push Online Media Stream Status

Status	Handling method
STATUS_START_SUCCESS	It indicates success, with no need for handling.
STATUS_START_FAILURE	If you receive push online media stream failure status three times, check the source URL and restart the push online media stream.
STATUS_START_AGAIN	Received within 1 minute after the push online media stream starts: It indicates the URL connection failed or the RTMP push failed. The system automatically triggers a retry. If it fails in the end, check if the URL is properly connected Received beyond 1 minute after the push online media stream starts: A restart may be triggered due to source stream or background network fluctuation,

	with no need for handling.

Basic Callback Transfer Example

Event transfer of push online media stream failure/push online media stream restart/push online media stream start success

`STATUS_START_FAILURE` -> `STATUS_START_AGAIN` -> `STATUS_START_SUCCESS`

Note:

Push online media stream callback events may arrive at your callback server out of sequence. You need to sort events based on EventMsTs in EventInfo. If you only care about the latest status of the URL, you can ignore the expired events that arrive later.

Definition of Event Information when the Event Type is (EVENT_TYPE_STREAM_INGEST_STOP 702):

Field name	Type	Meaning
EventMsTs	String	The Unix timestamp of the event occurred, in milliseconds
TaskId	String	Push online media stream task ID
Status	Number	Status of push online media stream stop

Status of Push Online Media Stream Stop

Field name	Value	Meaning	Callback Frequency
STATUS_STOP_SUCCESS	0	The push online media stream stop succeeded.	Callback is made once upon success.

Calculating a Signature

A signature is calculated with the HMAC SHA256 encryption algorithm. After your event callback server receives the callback message, it calculates the signature in the same manner. A match means that it is TRTC's event callback, with no falsification. The signature calculation is shown below:

```
// In the signature calculation formula Sign, the key refers to the encryption key
Sign = base64(hmacsha256(key, body))
```

Note:

The body refers to the original package body of the callback request received by you, with no transformation. An example is as follows:

```
body="{\\n\\t\\"Ebody="{\\"EventGroupId\\":7,\\"EventType\\":701,\\"CallbackMsTs\\"
```

Signature Verification Example

Java

Python

PHP

Golang

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# Feature: Third-party Callback Sign Verification
//# Parameters:
//#   key: The Key Configured on the Console
//#   body: The Body Returned by Tencent Cloud Callback
//#   sign: The Signature Value Returned by Tencent Cloud Callback
//# Returned Values:
//#   Status: OK Indicates that Verification Succeeded, and FAIL Indicates that Ver
//#   Info: Success/Failure Information

public class checkSign {
    public static String secureFinalSign(String key, String entityBody) throws Exce
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.initialize(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes(
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = obtainResultSignature(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': 'Verification succeeded'}"
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': 'Verification failed'}")
        }
    }
}
```

```
# -*- coding: utf8 -*-
```

```
import hmac
import base64
from hashlib import sha256

# Feature: Third-party Callback Sign Verification
# Parameters:
#   key: The Key on the Console
#   body: The Body Returned by Tencent Cloud Callback
#   sign: The Signature Value Returned by Tencent Cloud Callback
# Returned Values:
#   Status: OK Indicates that Verification Succeeded, and FAIL Indicates that Verif
#   Info: Success/Failure Information

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'
    print(computSign)
    if computSign equals sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = 'Verification succeeded'
        return temporary_dictionary
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = 'Verification failed'
        return temporary_dictionary

if __name__ == '__main__':
    key = '123654'
    body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\\":\\t204,\
    `sign` = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = verifySignature(key, body, sign)
    print(result)
```

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
                $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
                return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\\"Callbac

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

```
package main
import "fmt"
import (
        "crypto/hmac"
        "crypto/sha256"
        "encoding/base64"
)

func main () {
    var data = "{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\
    var key = "789"

    //JSRUN Engine 2.0, which supports online running in up to 30 languages and ful
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
        h := hmac.New(sha256.New, []byte(key))
        h.Write([]byte(data))
        return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

Conversational AI & Speech-to-Text Callbacks

Last updated：2024-11-19 16:56:42

This document introduces events generated by Tencent Cloud APIs related to AI services (Conversational AI and Speech-to-Text features), which are then notified to your server in the form of HTTP/HTTPS requests. You can provide relevant configuration information to Tencent Cloud to activate this service. You can also use it in combination with Tencent Real-Time Communication (TRTC)'s Event Callbacks to implement more custom logic.

Configuration Information

The TRTC console supports self-service configuration of callback information. Once the configuration is complete, you can receive event callback notifications. For detailed instructions, see Callback Configuration.

Note:

You need to prepare the following information in advance:

Required: the HTTP/HTTPS server address to receive callback notifications.

Optional: the key for calculating a signature, which is a key of up to 32 characters defined by you, consisting of uppercase and lowercase letters and digits.

Timeout Retry

If the event callback server does not receive a response from your server within 5 seconds after sending a message notification, it will consider the notification as failed. After the first failure, an immediate retry is attempted. Subsequent retries will occur at 10-second intervals until the message retention time exceeds 1 minute, after which no further retries will be made.

Event Callback Message Format

Event callback messages are sent to your server via HTTP/HTTPS POST requests, where:

Character encoding format: UTF-8.

Request: The body is in JSON format.

Response: The HTTP status code is 200. The server ignores the specific content of a response packet. For protocol friendliness, it is recommended that the client response content carry JSON: {"code":0}.

Packet example: The following is a packet example for the event of "successfully starting an AI conversation task".

```
{
        "EventGroupId":  9,
        "CallbackTs": 1687770730166,
        "EventInfo": {
            "EventMsTs": 1622186275757,
        "TaskId": "hKPD2Q7kBVzu-6ezFiqmcEBJQCykqbZrS9OOTE46uYlb4NvQDIaEXlpOlLXFtGBi
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Status": 0
        }
        }
}
```

Parameter Description

Callback Message Parameters

The header of an event callback message contains the following fields:

Field Name	Value
Content-Type	application/json.
Sign	Signature value.
SdkAppId	SDK application ID.

The body of an event callback message contains the following fields:

Field Name	Type	Meaning
EventGroupId	Number	Event group ID, which is fixed at 4 for stream mixing and relay events.
EventType	Number	The event type of the callback notification.
CallbackMsTs	Number	The Unix timestamp (in milliseconds) when the event callback server sends a callback request to your server.
EventInfo	JSON Object	Event information.

Event Group ID

Field Name	Value	Meaning

EVENT_GROUP_AI_SERVICE	9	AI service event group.

Event Type

Field Name	Value	Meaning
EVENT_TYPE_AI_SERVICE_START	901	Callback for the AI task start status.
EVENT_TYPE_AI_SERVICE_STOP	902	Callback for the AI task end status.
EVENT_TYPE_AI_SERVICE_MSG	903	Callback for a complete sentence. Conversational AI: Callback after recognizing a complete sentence. Speech-to-text: Callback for a transcribed complete sentence.

Definition of Event Information When the Event Type Is (EVENT_TYPE_AI_SERVICE_START 901):

Field Name	Type	Meaning
EventMsTs	String	The Unix timestamp (in milliseconds) when an event occurs.
TaskId	String	AI task ID.
RoomId	String	TRTC room ID.
RoomIdType	Integer	0: indicates a numeric room number. 1: indicates a string room number.
Payload.Status	Number	0: AI task started successfully. 1: AI task failed to start.

```
{
        "EventGroupId": 9,
        "EventType": 901,
        "CallbackTs": 1687770730166,
        "EventInfo": {
            "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Status": 0
        }
        }
}
```

Definition of Event Information When the Event Type Is (EVENT_TYPE_AI_SERVICE_STOP 902):

Field Name	Type	Meaning
EventMsTs	String	The Unix timestamp (in milliseconds) when an event occurs.
TaskId	String	AI task ID.
RoomId	String	TRTC room ID.
RoomIdType	Integer	0: indicates a numeric room number. 1: indicates a string room number.
Payload.LeaveCode	Integer	0: The task exits after the stop API is normally called. 1: The task exits after the customer's application removes the transcription bot. 2: The task exits after the customer's application dissolves the room. 3: The TRTC server removes the bot. 4: The TRTC server dissolves the room. 98: Internal error. It is recommended to retry. 99: There is no user stream in the room except the transcription bot. The task exits after exceeding the specified time.

```
{
        "EventGroupId": 9,
        "EventType": 902,
        "CallbackTs": 1687770730166,
        "EventInfo": {
            "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "LeaveCode": 0
        }
        }
}
```

Definition of Event Information When the Event Type Is (EVENT_TYPE_AI_SERVICE_MSG 903):

Field Name	Type	Meaning
EventMsTs	String	The Unix timestamp (in milliseconds) when an event occurs.
TaskId	String	AI task ID.
RoomId	String	TRTC room ID.
RoomIdType	Integer	0: indicates a numeric room number. 1: indicates a string room number.

Payload	JSON Object	It is a JSON object, which is consistent with the custom message callback format on the client.
		`{`
		"UserId":"",
		"Text":"",
		"StartTimeMs":1234,
		"EndTimeMs":1269,
		"RoundId":"xxxxxx" // uuid
		`}`

```
{
        "EventGroupId":  9,
        "EventType":     903,
        "CallbackTs":    1687770730166,
        "EventInfo": {
            "EventMsTs": 1622186275757,
       "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "UserId":"",
            "Text":"",
            "StartTimeMs":1234,
            "EndTimeMs":1269,
            "RoundId":"xxxxxx"
        }
        }
}
```

Calculating a Signature

Signatures are calculated using the HMAC SHA256 encryption algorithm. After your event callback server receives a callback message, it calculates the signature in the same manner. If they match, it indicates that it is an event callback from TRTC and has not been forged. The signature calculation is as follows:

```
// In the signature calculation formula, the 'key' is the encryption key used for c
Sign = base64 (hmacsha256 (key, body))
```

Note:

Body refers to the original packet body of the callback request received by you, and do not do any transformation, as shown below:

```
body="{\\n\\t\\"Ebody="{\\"EventGroupId\\":7,\\"EventType\\":701,\\"CallbackMsTs\\"
```

Signature Verification Example

Java

Python

PHP

Golang

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# Feature: third-party callback signature verification
//# Parameters:
//#   key: key configured in the console
//#   body: body returned by Tencent Cloud callbacks
//#   sign: signature value returned by Tencent Cloud callbacks
//# Returned values:
//#   Status: OK indicates verification successful. FAIL indicates verification fai
//#   Info: success/failure information

public class checkSign {
    public static String secureFinalSign(String key, String entityBody) throws Exce
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.initialize(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes(
    ]
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = obtainResultSignature(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': 'Verification successful.'
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': 'Verification failed.'}"
        ]
    ]
]
```

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# Feature: third-party callback signature verification
# Parameters:
#   key: key configured in the console
#   body: body returned by Tencent Cloud callbacks
#   sign: signature value returned by Tencent Cloud callbacks
# Returned values:
#   Status: OK indicates verification successful. FAIL indicates verification faile
#   Info: success/failure information

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'
    print(computSign)
    if computSign equals sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = 'Verification successful.'
        return temporary_dictionary
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = 'Verification failed.'
        return temporary_dictionary

if __name__ == '__main__':
    key = '123654'
    body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\\":\\t204,\
    `sign` = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = verifySignature(key, body, sign)
    print(result)
```

```
<?php
```

```
class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
                $this->body = $body;
    ]

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
                return base64_encode( $hash);
    ]

    public function genEventSig() {
        return $this->__hmacsha256();
    ]
]

$key="789";
$data="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\\"Callbac

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

```
package main
import "fmt"
import (
        "crypto/hmac"
        "crypto/sha256"
        "encoding/base64"
)

func main () {
    var data = "{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\
    var key = "789"

    //JSRUN Engine 2.0 supports running online in up to 30 languages, providing ful
    fmt.Println(hmacsha256(data,key))
]

func hmacsha256(data string, key string) string {
        h := hmac.New(sha256.New, []byte(key))
        h.Write([]byte(data))
        return base64.StdEncoding.EncodeToString(h.Sum(nil))
]
```

Verify Signature Example

Last updated：2023-10-08 16:01:32

Tencent Real-Time Communication(TRTC) console supports self-configuration of callback information. After the configuration is completed, you can receive event callback notifications. Before configuring the callback information, you need to prepare a key for signature calculation. You can define a key with a maximum of 32 characters, composed of uppercase and lowercase letters and numbers.
This document will help you verify signature after calculating it and show you how to perform an example.

Signature calculation

Signatures are calculated using the HMAC SHA256 encryption algorithm. Upon receiving a callback message, your server will calculate a signature using the same method, and if the results match, it indicates that the callback is from TRTC and not forged. See below for the calculation method.

```
// In the formula below, `key` is the key used to calculate a signature.
Sign = base64(hmacsha256(key, body))
```

Note

`body` is the original packet body of the callback request you receive. Do not make any modifications. Below is an example.

```
body="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t103,\\n\\t\\"Callback
```

Verify signature example

Java
Python
PHP
Golang

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# Function: Third-party callback sign verification
//# Parameters:
//#   key: The key configured in the console
//#   body: The body returned by the Tencent Cloud callback
//#   sign: The sign value returned by the Tencent Cloud callback
//# Return Value:
//#   Status: OK indicates that the verification has passed, FAIL indicates that th
//#   Info: Success/Failure information

public class checkSign {
    public static String getResultSign(String key, String body) throws Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.init(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes(
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': 'validation passed'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': 'validation failed'}");
        }
    }
}
```

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# Function: Third-party callback sign verification
# Parameters:
#   key: The key configured in the console
#   body: The body returned by the Tencent Cloud callback
#   sign: The sign value returned by the Tencent Cloud callback
# Return Value:
#   Status: OK indicates that the verification has passed, FAIL indicates that the
#   Info: Success/Failure information

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
```

```
            temp_dict['Info'] = 'validation passed'
            return temp_dict
        else:
            temp_dict['Status'] = 'FAIL'
            temp_dict['Info'] = 'validation failed'
            return temp_dict

if __name__ == '__main__':
    key = '123654'
    body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" + "\\t\\"EventType\\":\\t204,\
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
                $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
                return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\\"Callbac

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

```
package main
import "fmt"
import (
        "crypto/hmac"
        "crypto/sha256"
        "encoding/base64"
)

func main () {
    var data = "{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t101,\\n\\t\
    var key = "789"

    //JSRUN engine 2.0, supporting up to 30 types of languages for online running,
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
        h := hmac.New(sha256.New, []byte(key))
        h.Write([]byte(data))
        return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

Access Management

Overview

Last updated：2023-10-08 16:02:33

notice

This document describes the management of access to **TRTC**. For access management of other Tencent Cloud services, see CAM-Enabled Products.

Cloud Access Management (**CAM**) is a web service provided by Tencent Cloud that helps customers securely manage access to their Tencent Cloud account resources. CAM allows you to create, manage, or terminate users or user groups and control who is allowed to access and use your Tencent Cloud resources through identity and policy management.

TRTC has supported **CAM**. You can grant TRTC permissions to sub-accounts as needed.

Getting Started

Before you start, make sure that you understand the basic concepts of CAM and TRTC, including:

CAM: User Types and Policy

TRTC: Application and SDKAppID

Use Cases

Granting product-level permissions

A company has multiple departments that are using Tencent Cloud's products. Department A is solely responsible for TRTC-related business, and the company needs to grant the department access to TRTC but not to other Tencent Cloud products. To achieve this, the company can create a sub-account for department A under its root account and grant the sub-account only TRTC-related permissions.

Granting application-level permissions

A company has multiple businesses that are using TRTC and needs to isolate them from each other. There are two dimensions to isolation: resource isolation and permission isolation. The former is enabled by TRTC's application system, and the latter by CAM. The company can create a sub-account for each of the businesses and grant them access only to the TRTC applications they are responsible for.

Granting action-level permissions

A company has a business that is using TRTC. It needs to grant the business' operational staff access to the TRTC console so that they can obtain usage statistics, and at the same time deny them access to critical operations such as modifying relayed push and on-cloud recording configurations. To achieve this, the company can create a custom policy that has the permissions to use relevant APIs to log in to the TRTC console and view usage statistics, and associate the policy with the sub-account created for the operational staff.

Authorization Granularity

In essence, CAM enables you to **allow or forbid specified accounts to access certain resources**. TRTC access management supports resource-level authorization. The granularity of manageable resources is TRTC applications, and the granularity of manageable actions is TencentCloud APIs, including server APIs and the APIs used to access the TRTC console. For more information, please see Manageable Resources and Actions.

Limitations

The granularity of manageable resources for TRTC access management is applications. Access control of finer granularity (e.g., application information or configuration information) is not supported.
TRTC does not support project-level access management. We recommend that you use tags to manage your cloud service resources.

Manageable Resources and Actions

Last updated：2023-10-08 16:03:14

notice

This document describes the management of access to **TRTC**. For access management of other Tencent Cloud services, see CAM-Enabled Products.

In essence, CAM enables you to **allow or forbid specified accounts to access certain resources**. TRTC access management supports resource-level authorization. The granularity of manageable resources is TRTC applications, and the granularity of authorizable actions is TencentCloud APIs, including server APIs and APIs that may be needed to access the TRTC console.

If you need to manage access to TRTC, please log in to the console with a Tencent Cloud root account and use a preset policy or a custom policy to grant permissions.

Type of Manageable Resources

TRTC access management allows you to control access to applications.

APIs Supporting Resource-Level Authorization

Barring a few exceptions, all API actions listed in this section support resource-level authorization. Authorization policies related to these API actions use the same **syntax conventions**. See below for details.

Authorizing access to all applications: `qcs::trtc::uin/${uin}:sdkappid/*`

Authorizing access to single applications: `qcs::trtc::uin/${uin}:sdkappid/${SdkAppId}` .

Server API actions

API	Category	Description
DismissRoom	Room management	Closes a room.
RemoveUser	Room management	Removes a user.
RemoveUserByStrRoomId	Room management	Removes a user (string room ID).
DismissRoomByStrRoomId	Room management	Closes a room (string room ID).
StartMCUMixTranscode	Stream mixing and transcoding	Starts On-Cloud MixTranscoding.

StopMCUMixTranscode	Stream mixing and transcoding	Stops On-Cloud MixTranscoding.
StartMCUMixTranscodeByStrRoomId	Stream mixing and transcoding	Starts On-Cloud MixTranscoding (string room ID).
StopMCUMixTranscodeByStrRoomId	Stream mixing and transcoding	Stops On-Cloud MixTranscoding (string room ID).
CreateTroubleInfo	Call quality monitoring	Generates information about exceptional conditions.
DescribeAbnormalEvent	Call quality monitoring	Queries abnormal events.
DescribeCallDetail	Call quality monitoring	Queries user list and call metrics.
DescribeHistoryScale	Call quality monitoring	Queries room and user numbers in the past.
DescribeRoomInformation	Call quality monitoring	Queries room list.
DescribeUserInformation	Call quality monitoring	Queries the list of historical users.

Console API actions

API	Console	Description
DescribeAppStatList	TRTC console: Overview Usage Statistics Monitoring Dashboard Development Assistance > UserSig Generation & Verification Application Management	Gets application list.
DescribeSdkAppInfo	TRTC console: Application Management > Application Info	Gets application information.
ModifyAppInfo	TRTC console: Application Management > Application Info	Modifies application information.
ChangeSecretKeyFlag	TRTC console: Application Management > Application	Enables/Disables encryption keys.

	Info	
CreateWatermark	TRTC console: Application Management > Material Management	Uploads an image.
DeleteWatermark	TRTC console: Application Management > Material Management	Deletes an image.
ModifyWatermark	TRTC console: Application Management > Material Management	Edits an image.
DescribeWatermark	TRTC console: Application Management > Material Management	Searches an image.
CreateSecret	TRTC console: Application Management > Quick Start	Generates a symmetric encryption key.
ToggleSecretVersion	TRTC console:Application Management > Quick Start	Switches between asymmetric keys (private and public keys) and symmetric keys.
DescribeSecret	TRTC console: Development Assistance > Demo Quick Run Development Assistance > UserSig Generation & Verification Application Management > Quick Start	Gets a symmetric encryption key.
DescribeTrtcAppAndAccountInfo	TRTC console: Development Assistance > UserSig Generation & Verification	Gets application and account information to obtain a pair of public and private keys.
CreateSecretUserSig	TRTC console: Development Assistance > UserSig Generation & Verification	Uses a symmetric encryption key to generate a UserSig.
DescribeSig	TRTC console:	Gets a UserSig generated using a pair of public and private keys.

	Development Assistance > UserSig Generation & Verification Application Management > Quick Start	
VerifySecretUserSig	TRTC console: Development Assistance > UserSig Generation & Verification	Verifies a UserSig generated using a symmetric encryption key.
VerifySig	TRTC console: Development Assistance > UserSig Generation & Verification	Verifies a UserSig generated using a pair of public and private keys.
CreateSpearConf	TRTC console: Application Management > Image Settings	Adds an image setting. This module is available only in iLiveSDK 1.9.6 and earlier versions. For TRTC SDK 6.0 and later versions, see Setting Image Quality
DeleteSpearConf	TRTC console: Application Management > Image Settings	Deletes an image setting. This module is available only in iLiveSDK 1.9.6 and earlier versions. For TRTC SDK 6.0 and later versions, see Setting Image Quality
ModifySpearConf	TRTC console: Application Management > Image Settings	Modifies image settings. This module is available only in iLiveSDK 1.9.6 and earlier versions. For TRTC SDK 6.0 and later versions, see Setting Image Quality
DescribeSpearConf	TRTC console: Application Management > Image Settings	Gets image settings. This module is available only in iLiveSDK 1.9.6 and earlier versions. For TRTC SDK 6.0 and later versions, see Setting Image Quality
ToggleSpearScheme	TRTC console: Application Management > Image Settings	Switches image setting scenarios. This module is available only in iLiveSDK 1.9.6 and earlier versions. For TRTC SDK 6.0 and later versions, see Setting Image Quality

APIs Not Supporting Resource-Level Authorization

Due to special restrictions, the following APIs do not support resource-level authorization.

Server API actions

API	Category	Description	Restriction
DescribeDetailEvent	Call quality monitoring	Queries specific events.	The parameters entered do not include `SDKAppID` , making resource-level authorization impossible.
DescribeRecordStatistic	Other APIs	Queries the billing period of on-cloud recording.	For business reasons, resource-level authorization is not supported currently.
DescribeTrtcInteractiveTime	Other APIs	Queries the billing period for audio/video interactive features.	For business reasons, resource-level authorization is not supported currently.
DescribeTrtcMcuTranscodeTime	Other APIs	Queries the billing period of relayed transcoding.	For business reasons, resource-level authorization is not supported currently.

Console API actions

API	Console	Description	Restriction
DescribeTrtcStatistic	TRTC console: Overview Usage Statistics	Gets usage statistics.	This API returns the statistics of all \`SDKAppIDs\`. Limiting a query to specific \`SDKAppIDs\` will lead to an error. You can use \`DescribeAppStatList\` to specify a list of applications to query.
DescribeDurationPackages	TRTC console: Overview Package Management	Gets the list of prepaid packages.	A prepaid package is shared by all TRTC applications under the same Tencent Cloud account. There is no \`SDKAppID\` parameter in the package information, so resource-level authorization cannot be performed.
GetUserList	TRTC console:	Gets user list.	The parameters entered do not include \`SDKAppID\`, making resource-level

	Monitoring Dashboard		authorization impossible. You can use `DescribeAppStatList` to specify a list of applications to query.
GetUserInfo	TRTC console: Monitoring Dashboard	Gets user information.	The parameters entered do not include `SDKAppID`, making resource-level authorization impossible. You can use `DescribeAppStatList` to specify a list of applications to query.
GetCommState	TRTC console: Monitoring Dashboard	Gets call status.	The parameters entered do not include `SDKAppID`, making resource-level authorization impossible. You can use `DescribeAppStatList` to specify a list of applications to query.
GetElasticSearchData	TRTC console: Monitoring Dashboard	Queries Elasticsearch data.	The parameters entered do not include `SDKAppID`, making resource-level authorization impossible. You can use `DescribeAppStatList` to specify a list of applications to query.
CreateTrtcApp	TRTC console: Development Assistance > Demo Quick Run Application Management	Creates a TRTC application.	The parameters entered do not include `SDKAppID`, making resource-level authorization impossible. `SDKAppID` is the unique ID of a TRTC application and is generated after application creation.
HardDescribeMixConf	TRTC console: Application Management > Function Configuration	Queries relayed push status.	The parameters entered do not include `SDKAppID`, making resource-level authorization impossible. You can use `DescribeAppStatList` to specify a list of applications to query.
ModifyMixConf	TRTC console: Application Management > Function Configuration	Enables/Disables relayed push.	The parameters entered do not include `SDKAppID`, making resource-level authorization impossible. You can use `DescribeAppStatList` to specify a list of applications to query.
RemindBalance	TRTC	Gets the balance	A prepaid package is shared by all

	console: Package Management	alarm information of a prepaid package.	TRTC applications under the same Tencent Cloud account. There is no `SDKAppID` parameter in the package information, so resource-level authorization cannot be performed.

notice

You can use a custom policy to control access to an API that does not support resource-level authorization. In the policy statement, set the resource element to `*`.

Preset Policies

Last updated：2023-10-08 16:03:39

notice

This document describes the management of access to **TRTC**. For access management of other Tencent Cloud services, see CAM-Enabled Products.

TRTC access management works by associating permission policies with sub-accounts or granting policies to sub-accounts. The preset policies in the console allow you to perform some simple authorization. For more sophisticated authorization, see Custom Policies.

TRTC offers the following preset policies currently.

Policy	Description
QcloudTRTCFullAccess	Read-and-write permission
QcloudTRTCReadonlyAccess	Read-only permission

Examples of Using Preset Policies

Creating a sub-account with the read-and-write permission

1. Go to the User List page of the CAM console using a Tencent Cloud root account and click **Create User**.
2. On the displayed page, click **Custom Creation** to go to the "Create Sub-user" page.

explain

Finish the steps before **User Permissions** as instructed in Creating a Custom Sub-user.

3. On the **User Permissions** page:

3.1 Search for and check the preset policy `QcloudTRTCFullAccess` .

3.2 Click **Next**.

4. In the **Review** step, click **Complete**. After the sub-user is created successfully, download the login link and security credential file and store them properly. They contain the following information.

Information	Source	Use	Storage Required
Login link	Copied from the console page	Facilitates console login. Root account information is not required for login via the link.	No
User ID	Security credential file in CSV format	Required for console login	Yes
Password	Security credential file	Required for console login	Yes

	in CSV format		
SecretId	Security credential file in CSV format	Required for server API calling. For more information, seeAccess Key	Yes
SecretKey	Security credential file in CSV format	Required for server API calling. For more information, seeAccess Key	Yes

5. Provide the login link and security credentials to the party you want to authorize access, who will be able to use the sub-account to perform all kinds of TRTC operations, including visiting the TRTC console, calling TRTC server APIs, etc.

Granting read-and-write permission to existing sub-account

1. Go to the User List of the CAM console using a Tencent Cloud root account and click the target sub-account.
2. On the **User Details** page, click **Add** under the **Permission** tab. If the sub-account already has permissions, click **Associate Policy**.
3. Click **Select policies from the policy list**, search for and check the preset policy `QcloudTRTCFullAccess` , and complete the authorization as prompted.

Revoke the read-and-write permission of a sub-account

1. Go to the User List of the CAM console using a Tencent Cloud root account and click the target sub-account.
2. On the **User Details** page, find the preset policy `QcloudTRTCFullAccess` under the **Permission** tab, click **Disassociate** on the right, and complete the deauthorization as prompted.

Custom Policies

Last updated：2023-10-08 16:05:09

notice

This document describes the management of access to **TRTC**. For access management of other Tencent Cloud services, see CAM-Enabled Products.

It may be convenient to use a preset policy for access management in TRTC, but with preset policies, the granularity level of permissions is low, and permission granting cannot be specific to TRTC applications or TencentCloud APIs.

To perform fine-grained authorization, you need to create custom policies.

Custom Policy Creation

There are multiple ways to create a custom policy. The table below offers a comparison of different methods. For detailed directions, see the remaining part of the document.

Access	Tool	Effect	Resource	Action	Flexibility	Complexity
CAM console	Policy generator	Manual selection	Syntax conventions	Manual selection	Medium	Medium
CAM console	Policy syntax	Syntax conventions	Syntax conventions	Syntax conventions	High	High
CAM server API	CreatePolicy	Syntax conventions	Syntax conventions	Syntax conventions	High	High

explain

TRTC does **not support** custom policy creation by product feature or project.

Manual selection means that you can select an object from a list of candidates offered in the console.

Syntax conventions means using the permission policy syntax to describe an object.

Permission Policy Syntax

Resource syntax conventions

The granularity level of manageable resources in TRTC access management is applications. Syntax conventions of permission policies for applications are in line with the Resource Description Method. In the example below, the developer (root account ID: `12345678`) has created three applications, whose `SDKAppIDs` are `1400000000` , `1400000001` , and `1400000002` .

Syntax convention of permission policy for all TRTC applications

```
"resource": [
  "qcs::trtc::uin/12345678:sdkappid/*"
]
```

Syntax convention of permission policy for single TRTC applications

```
"resource": [
  "qcs::trtc::uin/12345678:sdkappid/1400000001"
]
```

Syntax convention of permission policy for multiple TRTC applications

```
"resource": [
  "qcs::trtc::uin/12345678:sdkappid/1400000000",
  "qcs::trtc::uin/12345678:sdkappid/1400000001"
]
```

Action syntax conventions

The granularity level of authorizable actions in TRTC access management is TencentCloud APIs. For details, see Manageable Resources and Actions. The examples below use TencentCloud APIs such as `DescribeAppList` (gets application list) and `DescribeAppInfo` (gets application information).

Syntax convention of permission policy for all TencentCloud APIs

```
"action": [
  "name/trtc:*"
]
```

Syntax convention of permission policy for single TencentCloud APIs

```
"action": [
  "name/trtc:DescribeAppStatList"
]
```

Syntax convention of permission policy for multiple TencentCloud APIs

```
"action": [
  "name/trtc:DescribeAppStatList",
  "name/trtc:DescribeTrtcAppAndAccountInfo"
]
```

Examples of Using Custom Policies

Using the policy generator

In the example below, we create a custom policy that allows all actions under TRTC application `1400000001` except calling the server API `RemoveUser` .

1. Go to the **Policy** page of the CAM console using a Tencent Cloud root account and click **Create Custom Policy**.
2. Select **Create by Policy Generator**.
3. Select the service and action.

For **Effect**, select **Allow**.
For **Service**, select **Tencent Real-Time Communication (trtc)** .
For **Action**, check all the items.
For **Resource**, enter `qcs::trtc::uin/12345678:sdkappid/1400000001` , which aligns with the syntax described in Resource syntax conventions.
No configuration is needed for **Condition**.
Click **Add Statement**, and a statement indicating that any action is allowed under TRTC application `1400000001` appears below.

4. Add another statement on the same page.

For **Effect**, select **Deny**.
For **Service**, select **Tencent Real-Time Communication (trtc)**.
For **Action**, select `RemoveUser` . You can use the search feature to quickly locate the action.
For **Resource**, enter `qcs::trtc::uin/12345678:sdkappid/1400000001` , which aligns with the syntax described in Resource syntax conventions.
No configuration is needed for **Condition**.
Click **Add Statement**, and a statement indicating that calling `RemoveUser` is forbidden under TRTC application `1400000001` appears below.

5. Click **Next** and rename the policy if necessary.
6. Click **Done** to complete the creation.

You can then grant the policy to other sub-accounts as described in Granting read-and-write permission to existing sub-account.

Using the policy syntax

In the example below, we create a custom policy that allows all actions under TRTC application `1400000002` and all actions but calling `RemoveUser` under `1400000001` .

1. Go to the Policy page of the CAM console using a Tencent Cloud root account and click **Create Custom Policy**.
2. Select **Create by Policy Syntax**.
3. In the **Select a template type** section, select **Blank Template**.

explain

A policy template allows you to create a policy by modifying a copy of an existing policy (preset or custom). You can choose a policy template that fits your actual conditions to reduce the complexity and workload of writing permission policies.

4. Click **Next** and rename the policy if necessary.
5. Enter the following content in the **Policy Content** box.

```
{
 "version": "2.0",
 "statement":[
     {
         "effect": "allow",
         "action": [
             "name/trtc:*"
         ],
         "resource": [
             "qcs::trtc::uin/12345678:sdkappid/1400000001",
             "qcs::trtc::uin/12345678:sdkappid/1400000002"
         ]
     },
     {
         "effect": "deny",
         "action": [
             "name/trtc:RemoveUser"
         ],
         "resource": [
             "qcs::trtc::uin/12345678:sdkappid/1400000001"
         ]
     }
 ]
}
```

explain

Policy content must align with the Syntax Logic. About the syntax of the resource and action elements, see Resource syntax conventions and Action syntax conventions above.

6. Click **Create Policy** to complete the creation.

You can then grant the policy to other sub-accounts as described in Granting read-and-write permission to existing sub-account.

Using server APIs provided by CAM

Managing access in the console can meet the business needs of most developers, but to automate and systematize your access management, you need to use server APIs.

Permission policy-related server APIs belong to CAM. For details, see CAM documentation. Only a few main APIs are listed below:

CreatePolicy

DeletePolicy

AttachUserPolicy

DetachUserPolicy

Enable Watermark
Flutter

Last updated：2024-09-02 17:21:28

This document mainly explains how to add a watermark to a video stream.

Implementation Steps

Enable the camera

```
trtcCloud.startLocalPreview(isFrontCamera, viewId);
```

Watermarking

Assuming we need to add a local image `bg_main_title.png` as a watermark on the Main Screen (`TRTC_VIDEO_STREAM_TYPE_BIG`):

```
trtcCloud.setWatermark("images/bg_main_title.png", TRTCCloudDef.TRTC_VIDEO_STREAM_T
```

The position of the watermark is specified by the rect parameter, which is a tuple parameter in the format (x,y,width,height)

x: Coordinates of the watermark, value range is a float between 0 and 1.

y: Coordinates of the watermark, value range is a float between 0 and 1.

width: Width of the watermark, value range is a float between 0 and 1.

height: No need to set, the SDK will automatically calculate a suitable height based on the aspect ratio of the watermark image.

Example of parameter setting:

If the current video encoding resolution is 540 × 960, and the rect parameter is set to (0.1, 0.1, 0.2, 0.0), then the top-left coordinate point of the watermark will be (540 × 0.1, 960 × 0.1) which is (54, 96). The watermark's width will be 540 × 0.2 = 108px. The height of the watermark will be automatically calculated by the SDK based on the aspect ratio of the watermark image.

Note:

The watermark image must use a **transparent background PNG format.**

The watermark added through the setWatermark interface is not visible in the local preview. To view the watermark effect, you need to get the user stream with the set watermark from the remote end.

Pulling the video stream with a watermark

Pull the user video stream with a watermark on another device.

```
trtcCloud.startRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, viewId)
```

Cancel the watermark

By passing a null value, you can cancel the watermark in your published video stream.

```
trtcCloud.setWatermark("", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, 0.3, 0.4, 0.4);
```

How to push stream to TRTC room with OBS WHIP

Last updated：2024-08-07 10:53:53

Overview

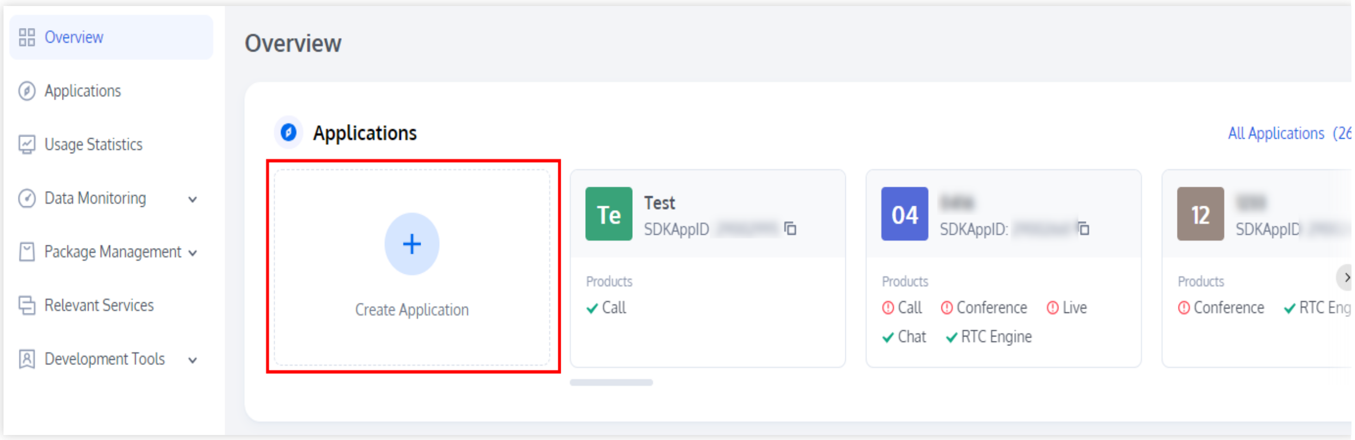
OBS includes WHIP support, which allows you to do many interesting things by combining the powers of both OBS and WHIP.
WHIP is a standard protocol that lets you use HTML5 and different clients to publish and play live streams. Plus, you can use open-source tools to build your own live streaming platform.
You can also use TRTC (Tencent Real-Time Communication) cloud services with OBS WHIP support for a streaming platform. This is a great option if you don't want to build your own platform or need a more reliable and stable platform with dedicated support.
Additionally, TRTC (Tencent Real-Time Communication) provides a free trial that includes a specific amount of streaming time, making it super easy for you to try out.
If you need help or run into any problems, don't hesitate to contact us on Discord.

Prerequisites

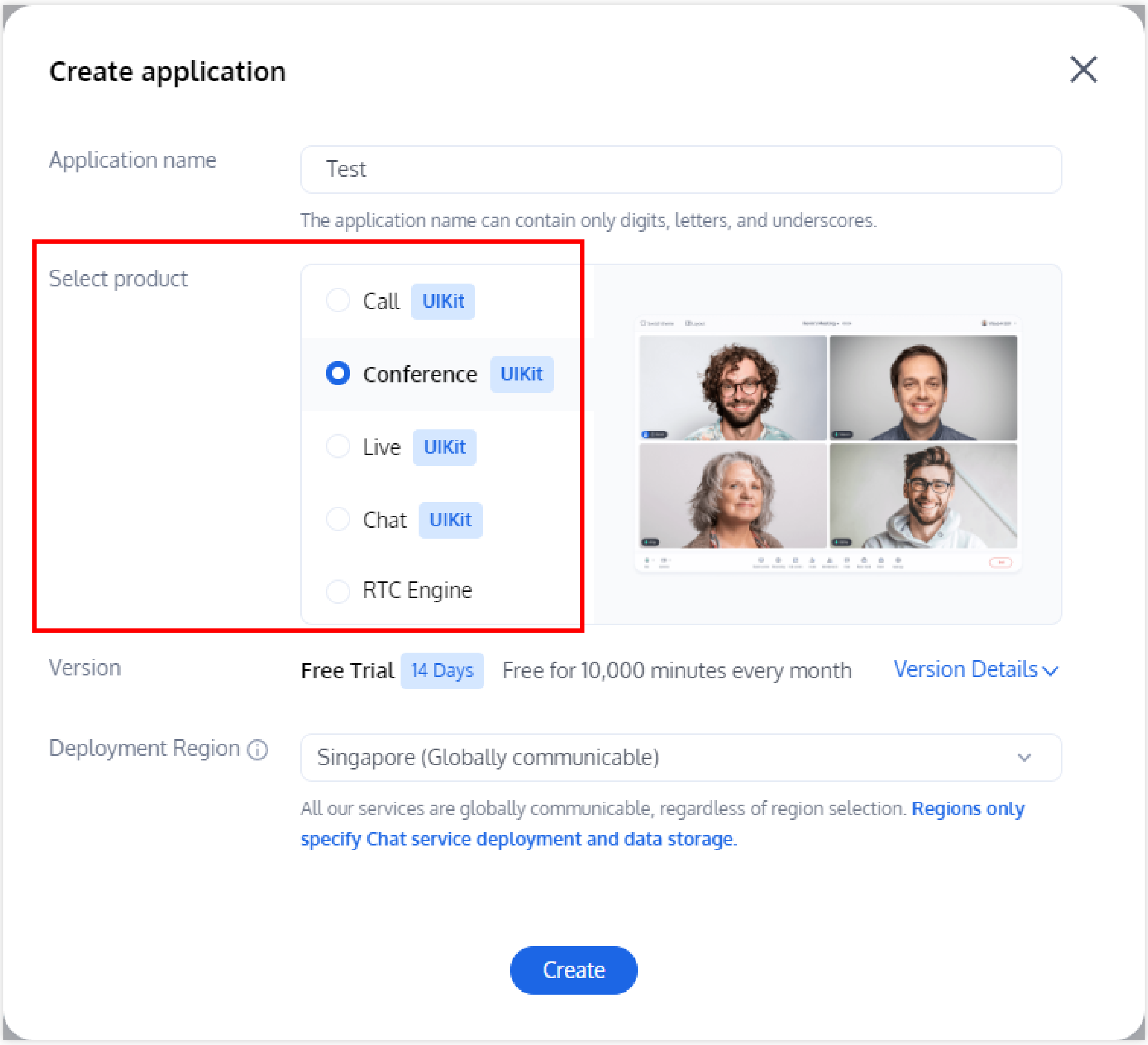
Before you move forward, double-check that you've got these necessary items ready:
OBS with WHIP support, please download from OBS
TRTC (Tencent Real-Time Communication) account, please register here
Next, you need to create a TRTC application and generate a Bearer Token for WHIP.

Step 1: Create a TRTC application

Please follow the steps below to create a TRTC application:
1. Log in to the Tencent RTC Console and click **Create Application**.



2. In the creation pop-up, based on the actual business needs, select a product and enter the application name, select the Data Storage **Region**, and click **Create**.


Create application

Application name: Test

The application name can contain only digits, letters, and underscores.

Select product: Call UIKit / [✓ Conference UIKit] / Live UIKit / Chat UIKit / RTC Engine

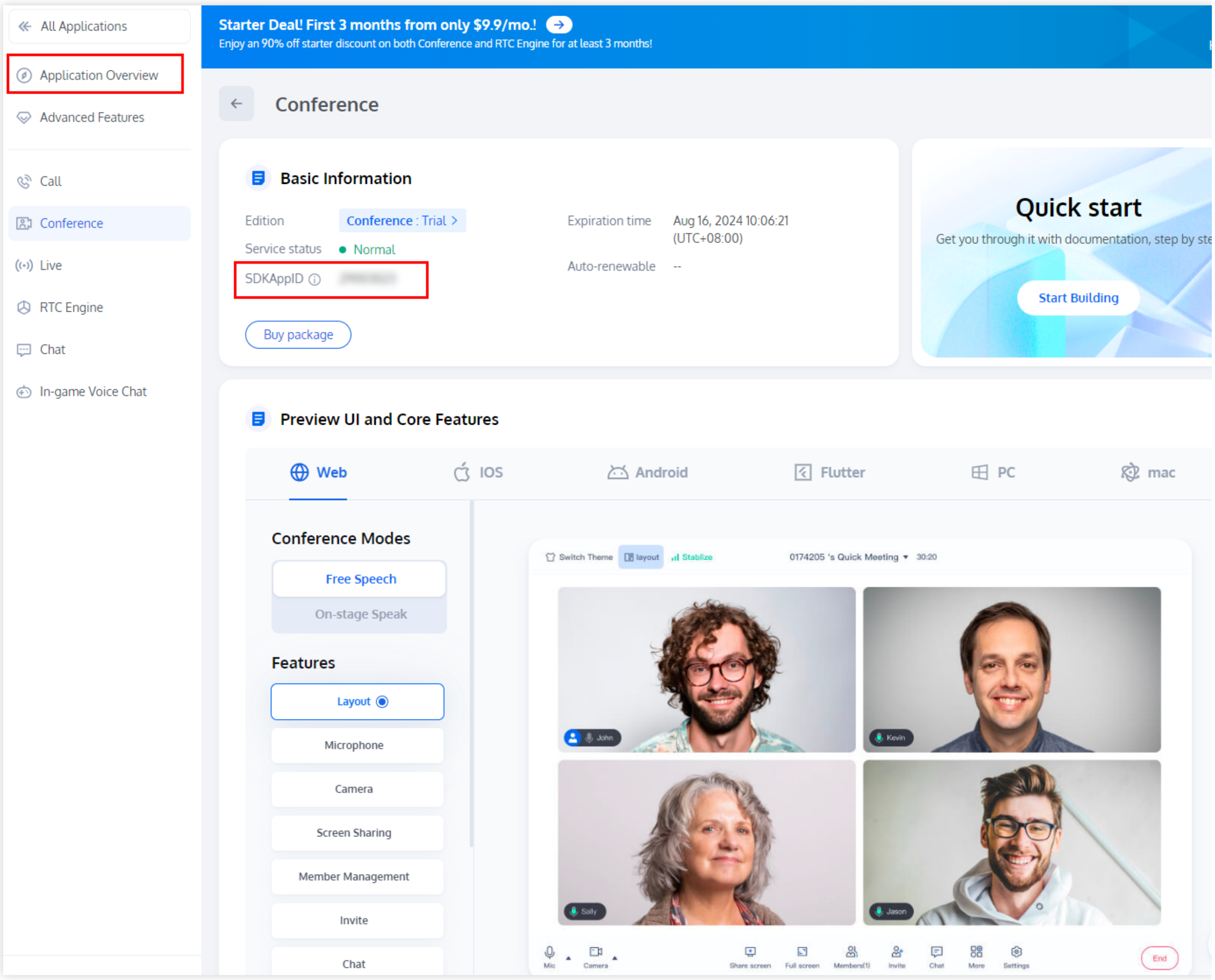
Version: Free Trial 14 Days Free for 10,000 minutes every month Version Details

Deployment Region: Singapore (Globally communicable)

All our services are globally communicable, regardless of region selection. Regions only specify Chat service deployment and data storage.

Create

3. After completing the application creation, you will automatically enter the application details page of the selected product. You can view the `SDKAppID` and `SDKSecretKey` in the **Application Overview**, which will be used in subsequent steps.

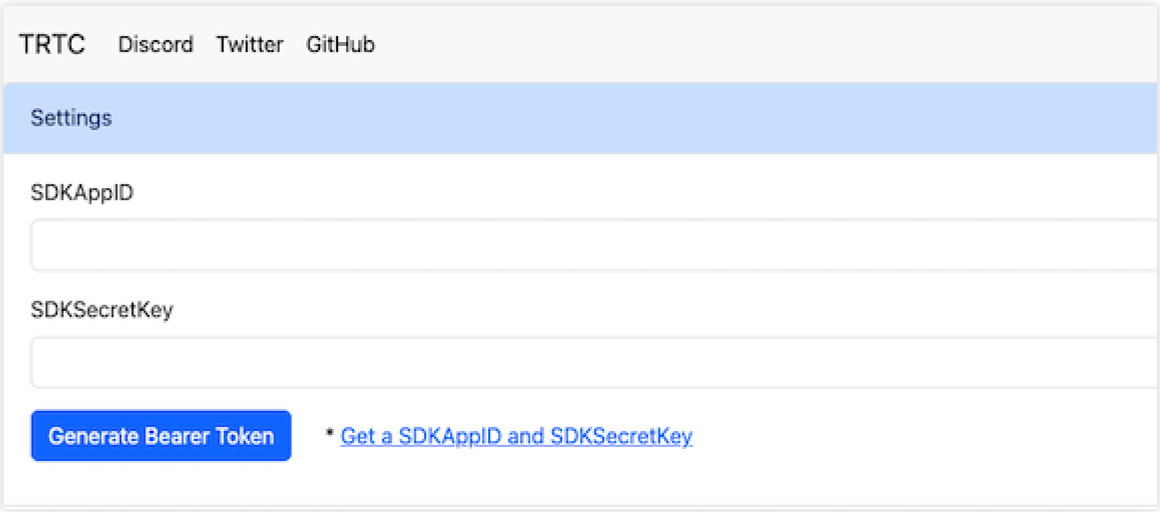


Edition: Conference : Trial
Service status: Normal
SDKAppID: [redacted]
Expiration time: Aug 16, 2024 10:06:21 (UTC+08:00)
Auto-renewable: --

Step 2: Create a Bearer Token for WHIP

Following that, you must generate a Bearer Token for WHIP, which will be utilized in OBS.

You can directly visit https://tencent-rtc.github.io/obs-trtc/bearer.html to create a WHIP Bearer Token. Ensure that use the AppID with your own `SDKAppID` and secret with your own `SDKSecretKey` , then click the `Generate Bearer Token` button.



TRTC Discord Twitter GitHub

Settings

SDKAppID

SDKSecretKey

Generate Bearer Token

* Get a SDKAppID and SDKSecretKey

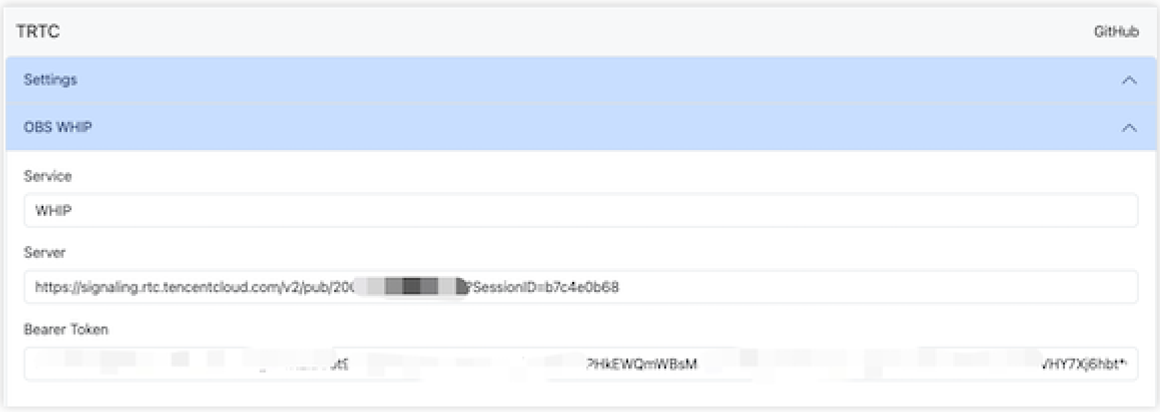
Note：

You can also access the url `https://tencent-rtc.github.io/obs-trtc/bearer.html?appid=2000xxx&secret=yyyyyy` to setup the parameters.

Next, use the generated WHIP Bearer Token to configure OBS.

Step 3: Configure OBS

In the `OBS WHIP` section, you will find the generated WHIP `Server` and `Bearer Token` for configuring OBS.



TRTC   GitHub

Settings

OBS WHIP

Service

WHIP

Server

https://signaling.rtc.tencentcloud.com/v2/pub/20[redacted]?SessionID=b7c4e0b68

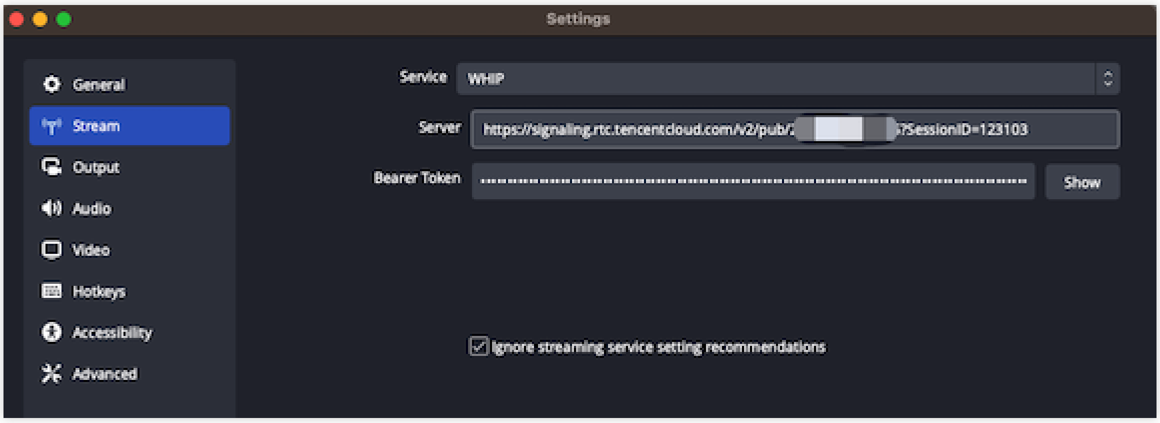
Bearer Token

[redacted]PHkEWQmWBsM[redacted]VHY7Xj6hbt*

Please follow the steps below to configure OBS:

1. Open OBS and click **Settings**.
2. Click **Stream** on the left sidebar.
3. Select `WHIP` for **Service**.
4. Make sure to input the `Server` and `Bearer Token` accurately.

5. Click **OK** to save the settings.
6. Click **Start Streaming** to start.



Service: WHIP

Server: https://signaling.rtc.tencentcloud.com/v2/pub/[redacted]?SessionID=123103

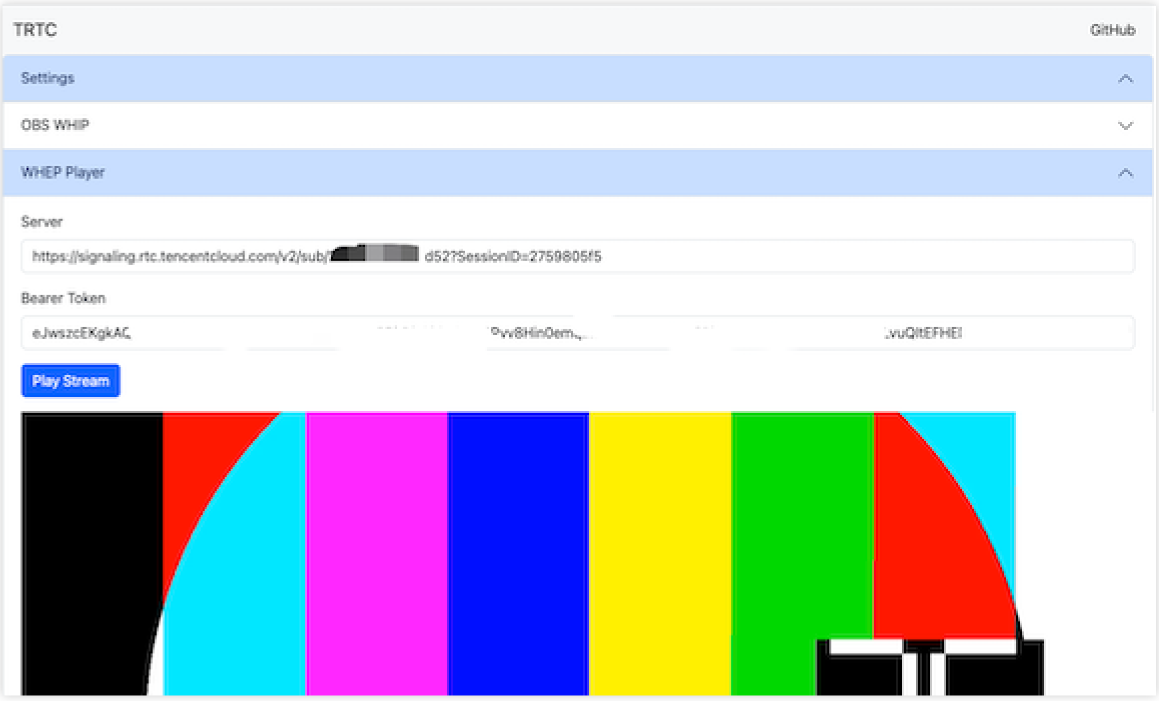
Bearer Token: ••

☑ Ignore streaming service setting recommendations

At this point, the stream is streaming to the TRTC service.

Step 4: Play the stream

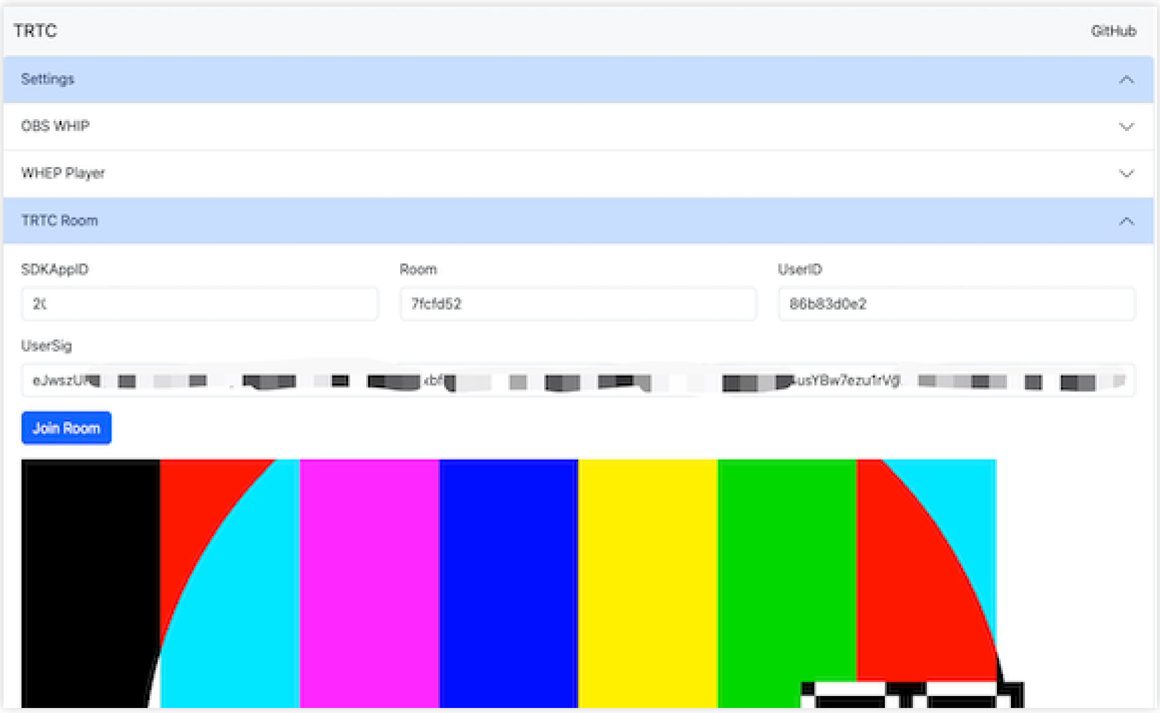
Open the previous webpage, go to the `WHEP Player` section, and click **Play Stream** to play the stream via WHEP.



Server: https://signaling.rtc.tencentcloud.com/v2/sub/[redacted]d52?SessionID=2759805f5

Bearer Token: eJwzcEKgkAC[redacted]Pvv8HinOem[redacted]vuQItEFHEI

Another option is to go to the `TRTC Room` section, and click **Join Room** to access the TRTC room and watch the stream via TRTC, or you can utilize the TRTC mobile SDK to join the room and view the stream.



SDKAppID: 2(
Room: 7fcfd52
UserID: 86b83d0e2
UserSig: eJwszU[redacted]xbf[redacted]usYBw7ezu1rVg[redacted]

Since both WHIP and WHEP are standard protocols, you can utilize any client that supports them to play the stream.

Conclusion

We looked into using TRTC (Tencent Real-Time Communication) cloud services to make a stronger streaming platform and the steps needed to create a TRTC app with OBS WHIP help. These tools make it easier to organize and provide real-time live streaming experiences for different situations, with the power of OBS.

If you require assistance or encounter any difficulties, please feel free to reach out to us via Discord.

Video Screenshot Upload

Last updated：2024-10-21 17:53:26

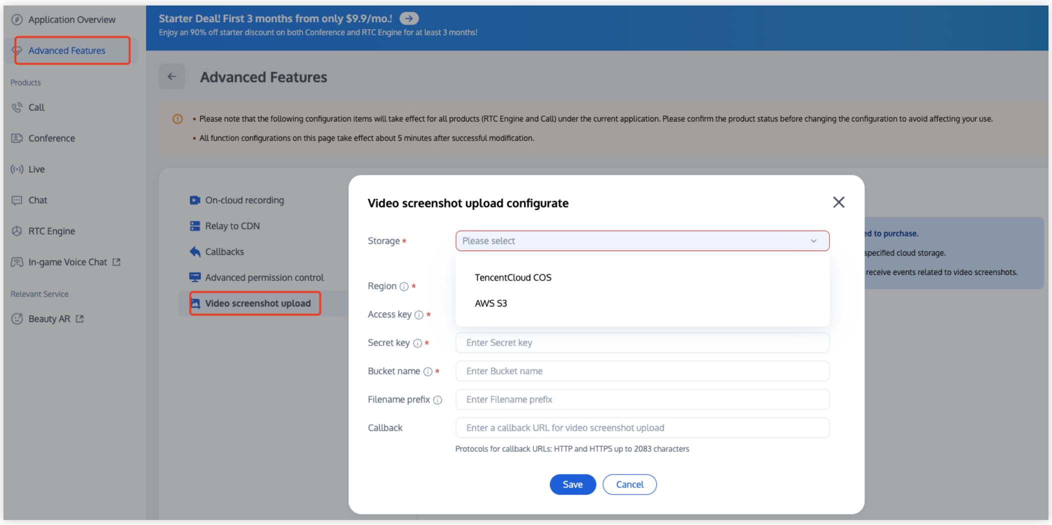
Feature Overview

TRTC currently supports the automatic screenshot upload feature of SDK. Screenshots can be used in scenarios such as third-party review and setting cover images to meet the needs of users.

Prerequisites

Log in to Console to create an RTC Engine application.

Navigate to Application> Advanced Features, enable the Video Screenshot Upload feature, and configure the specified storage location (currently supporting Tencent Cloud Object Storage (COS) and AWS S3).



• Please note that the following configuration items will take effect for all products (RTC Engine and Call) under the current application. Please confirm the product status before changing the configuration to avoid affecting your use.
• All function configurations on this page take effect about 5 minutes after successful modification.

Protocols for callback URLs: HTTP and HTTPS up to 2083 characters

Note:

1. To use the **Video Screenshot Upload** feature, you need to purchase the RTC Engine **Pro Edition** package. For more information on monthly packages, see RTC-Engine Monthly Packages.
2. The **Video Screenshot Upload** feature will incur charges based on the number of screenshots taken. For more details, see Fee Details.
3. To use the **Video Screenshot Upload** feature, please submit a ticket to contact us for the latest version of the SDK (currently supported on Android, iOS/Mac and Windows).

Feature Overview

1. See Prerequisites, enable the feature toggle and set the storage location.
2. Use the SDK experimental interface callExperimentalAPI to utilize this feature. The parameters shall be passed as a JSON string. The parameter description is as follows:

```
{
    "api": "enableAutoSnapshotAndUpload",
    "params" : {
    "'enable": 1, //Start/stop automatic screenshot, int, required. Field values: 0
    "intervalS": 1, //Screenshot interval, int, optional. The interval is expressed
    "streamType": 0, //Stream type, int, optional. Field values: 0: BigStream, and
    "extraInfo": "customized messages" //Screenshot upload additional information,
    }

}
```

Note:

The screenshot upload task will only start after the enterRoom is successful.
It is recommended to call this method after the startLocalPreview is successful to avoid screenshot upload task failure.

Receiving Server-side Event Callbacks

Configuration Information

TRTC Console supports self-configured callback information. Once configured, you can receive event callback notifications. For detailed directions, see Callback Configuration.

Note:

You need to prepare the following information in advance:
Required: The HTTP/HTTPS server address to receive callback notifications.
Optional: The key for calculating the signature. It is a key of up to 32 characters defined by you, consisting of uppercase and lowercase letters and numbers.

Timeout Retry

If the event callback server does not receive a response from your server within 5 seconds after sending a message notification, it is considered a notification failure. After the first failure, an immediate retry is attempted. Subsequent retries will occur at 10-second intervals until the message retention time exceeds 1 minute, after which no further retries will be made.

Format of Event Callback Message

The event callback message is dispatched to your server via HTTP/HTTPS POST requests, in which:
Character Encoding Format: UTF-8.

Request: The body is in JSON format.
Response: HTTP STATUS CODE = 200. The server ignores the specific content of the response package. For the sake of a friendly protocol, it is recommended that the response content from a customer shall carry JSON: {"code":0}.
Package Example: The following is a package example of the "Retweet Time Group - CDN Streaming in Progress" event.

Callback Message Parameters

The header of the event callback message contains the following fields:

Field name	Value
Content-Type	application/json
Sign	Signature value
SdkAppId	sdk application id

The body of the event callback message contains the following fields:

Field name	Type	Meaning
EventGroupId	Number	Event group ID, the value of a screenshot event (EVENT_GROUP_SCREEN_SHOT) is 6
EventType	Number	The event type of the callback notification, the value of a video screenshot (EVENT_TYPE_VIDEO_SCREENSHOT) is 601
CallbackTs	Number	The Unix timestamp when the event callback server sends a callback request to your server, expressed in millisecond
EventInfo	JSON Object	Event information

Description of event information :

Field name	Type	Meaning
eventId	String	The event ID for this callback
callbackData	String	Screenshot upload additional information, reported via the client's extraInfo
pictureURL	String	The URL of the screenshot
code	Number	Task execution status code, default: 0, indicating that the

		task is executed successfully
msg	String	Description information of task execution
roomID	String/Number	Room ID
streamType	String	Stream type of the screenshot, BigStream or SubStream
userID	String	Screenshot username
timestamp	Number	UTC timestamp of the screenshot, accurate to millisecond

Callback Request Example

```
{
   "EventGroupId": 6,
   "EventType": 601,
    "CallbackTs": 1698410059705,
   "EventInfo": {
       "eventID": "ap-guangzhou-1400000000-1698410059243691647-60022-jpg.jpg",
       "callbackData": "test",
       "pictureURL": "https://sotest-1200000000.cos.ap-guangzhou.myqcloud.com/14000
       "code": 0,
       "msg": "",
       "roomID": "464884",
       "streamType": "BigStream",
       "userID": "dd",
       "timestamp": 1698410059693
    }
}
```

Calculating Signature

The signature is calculated by using the HMAC SHA256 encryption algorithm. When your event callback receiver receives a callback message, it calculates the signature in the same way. If the signatures match, it indicates that the callback is from Tencent Cloud Real-Time Audio and Video and has not been tampered with. The calculations of the signature are as follows:

```
//In the calculation formula of Sign, the key is the encryption key for calculating
Sign = base64 (hmacsha256(key, body))
```

Note:

The body is the original package of the callback request you received. **Do not perform any transformations; it is imperative to preserve in its entirety, including \\n\\t escape characters.** An example is provided below:

```
body="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t103,\\n\\t\\"Callback
```

Signature Verification Example (Java)

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# Feature: Verification of the third-party callback sign
//# Parameters:
//# Key: The key configured in the console
//# Body: The body returned by Tencent Cloud callback
//# Sign: The sign returned by Tencent Cloud callback
//# Returned values:
//# Status OK indicates that it has passed the verification, and FAIL indicates tha
//# Info: The information of pass/fail



public class checkSign {
    public static String getResultSign(String key, String body) throws

Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(),
"HmacSHA256");
        hmacSha256.initialize(secret_key);
                return
Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\\n" + "\\t\\"EventGroupId\\":\\t2,\\n" +
"\\t\\"EventType\\":\\t204,\\n" + "\\t\\"CallbackTs\\":\\t1664209748188,\\n" +
"\\t\\"EventInfo\\":\\t{\\n" + "\\t\\t\\"RoomId\\":\\t8489,\\n" +
"\\t\\t\\"EventTs\\":\\t1664209748,\\n" + "\\t\\t\\"EventMsTs\\":\\t1664209748180,\
"\\t\\t\\"UserId\\":\\t\\"user_85034614\\",\\n" + "\\t\\t\\"Reason\\":\\t0\\n" + "\
"}";

        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = obtainResultSignature(key, body);


        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': 'Verification passed'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': 'Verification failed'}")
            }
        }
    }
```

Note:

For more signature examples, see Signature Verification Example.