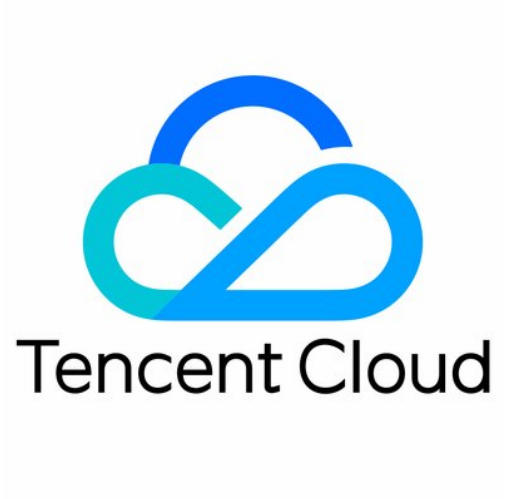
Tencent Real-Time Communication

고급 기능

제품 문서



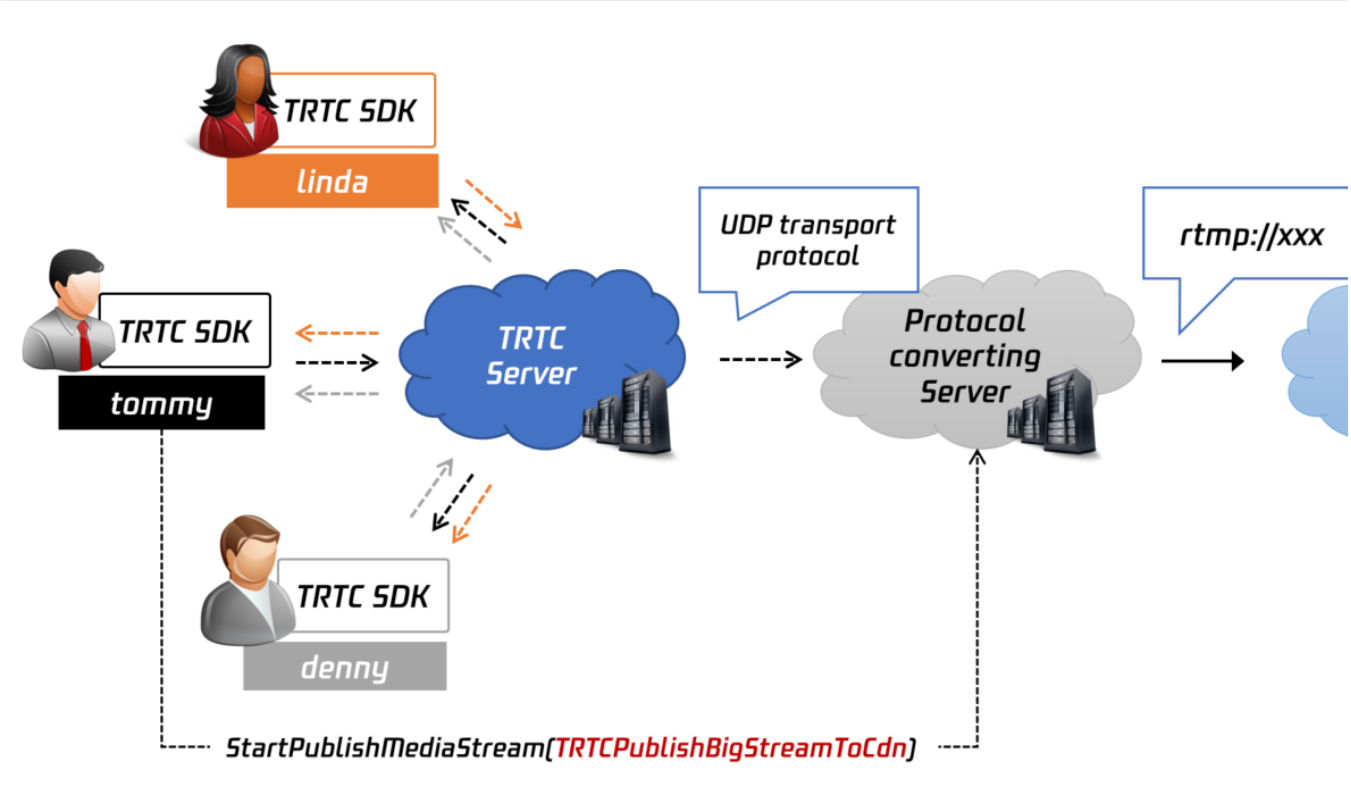
목록:

고급 기능
CDN으로 릴레이

최종 업데이트 날짜: : 2024-07-24 16:43:38

본 문서는 시청자가 표준 라이브 스트리밍 플레이어를 사용하여 스트림을 볼 수 있도록 TRTC의 오디오/비디오 스트림을 CDN에 게시(릴레이)하는 방법을 설명합니다.

로컬 사용자의 스트림을 CDN에 게시



기능 소개

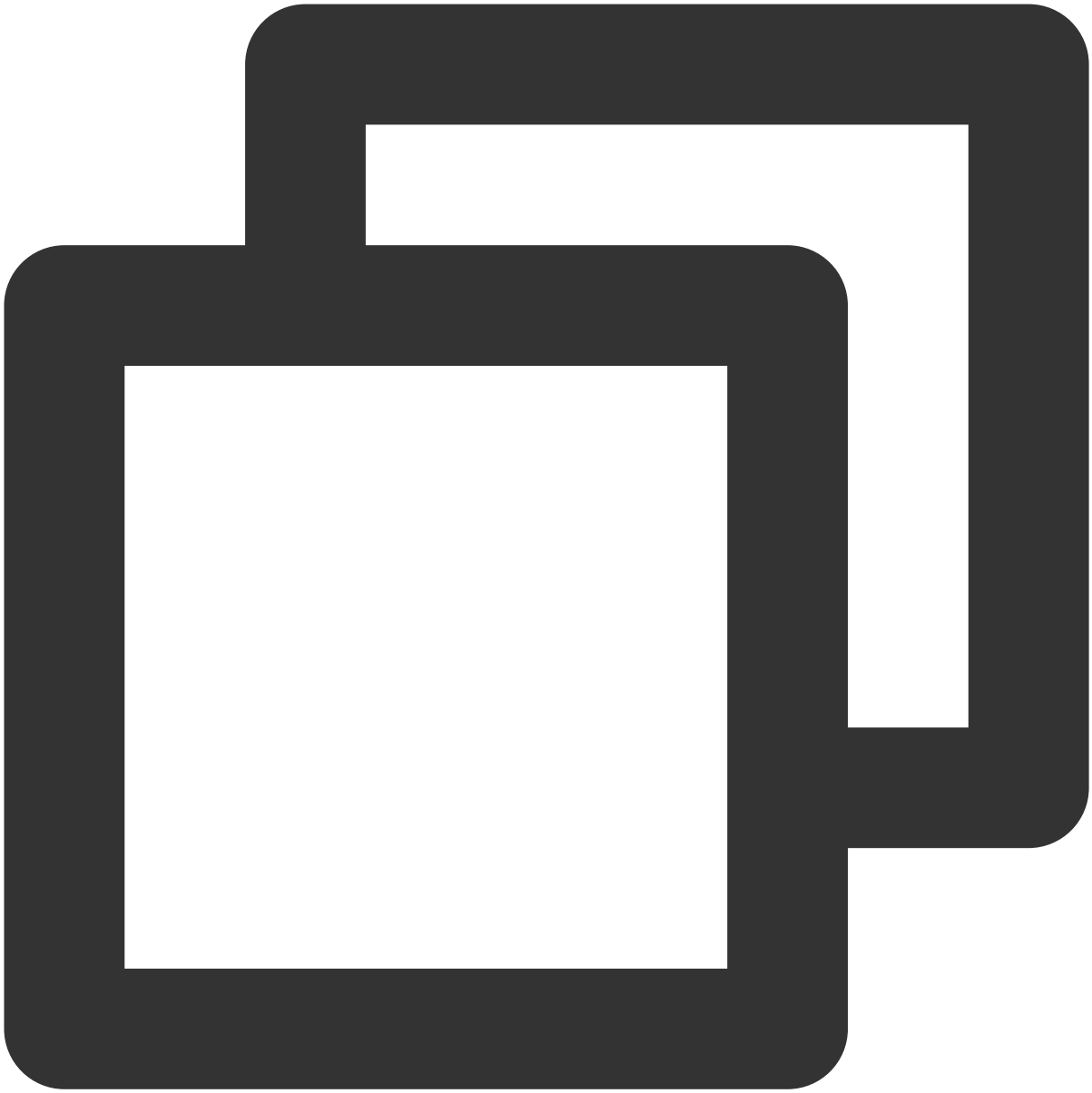
TRTCCloud의 **startPublishMediaStream** API를 사용하여 로컬 사용자의 오디오/비디오 스트림을 라이브 스트리밍 CDN(TRTC에서 'Relay to CDN'이라고 함)에 게시할 수 있습니다.
TRTC 서버는 오디오/비디오 데이터를 CDN 서버로 직접 보냅니다. 데이터가 트랜스 코딩되지 않기 때문에 비용이 비교적 저렴합니다.
그러나 한 방에 오디오/비디오 스트림을 게시하는 여러 사용자가 있는 경우 각 사용자에 대한 CDN 스트림이 있습니다. 스트림을 재생하려면 여러 플레이어가 필요하며 동기화되어 재생되지 않을 수 있습니다.

작업 가이드

로컬 사용자의 스트림을 CDN에 게시하려면 아래 단계를 따르십시오.
1. TRTCPublishTarget 객체를 생성하고 TRTCPublishTarget 객체의 mode를 `TRTCPublishBigStreamToCdn` 또는 `TRTCPublishSubStreamToCdn` 으로 설정합니다. 전자는 사용자의 기본 스트림(일반적으로 카메라)을 게시하는 데 사용되며 후자는 사용자의 서브 스트림(일반적으로 화면)을 게시하는 데 사용됩니다.
2. TRTCPublishTarget 객체의 cdnUrlList를 하나 이상의 CDN 주소(보통 `rtmp://` 로 시작)로 설정합니다. Tencent Cloud CDN에 게시하는 경우 isInternalLine을 true로 설정합니다. 그렇지 않으면 false로 설정하십시오.
3. 데이터가 트랜스 코딩되지 않았으므로 `TRTCStreamEncoderParam` 및 `TRTCStreamMixingConfig` 를 비워 둡니다.
4. startPublishMediaStream을 호출합니다. onStartPublishMediaStream 콜백에서 반환된 taskId 매개변수가 비어 있지 않으면 로컬 API 호출이 성공한 것입니다.
5. 게시를 중지하려면 stopPublishMediaStream을 호출하고 onStartPublishMediaStream에서 반환된 taskId를 전달합니다.

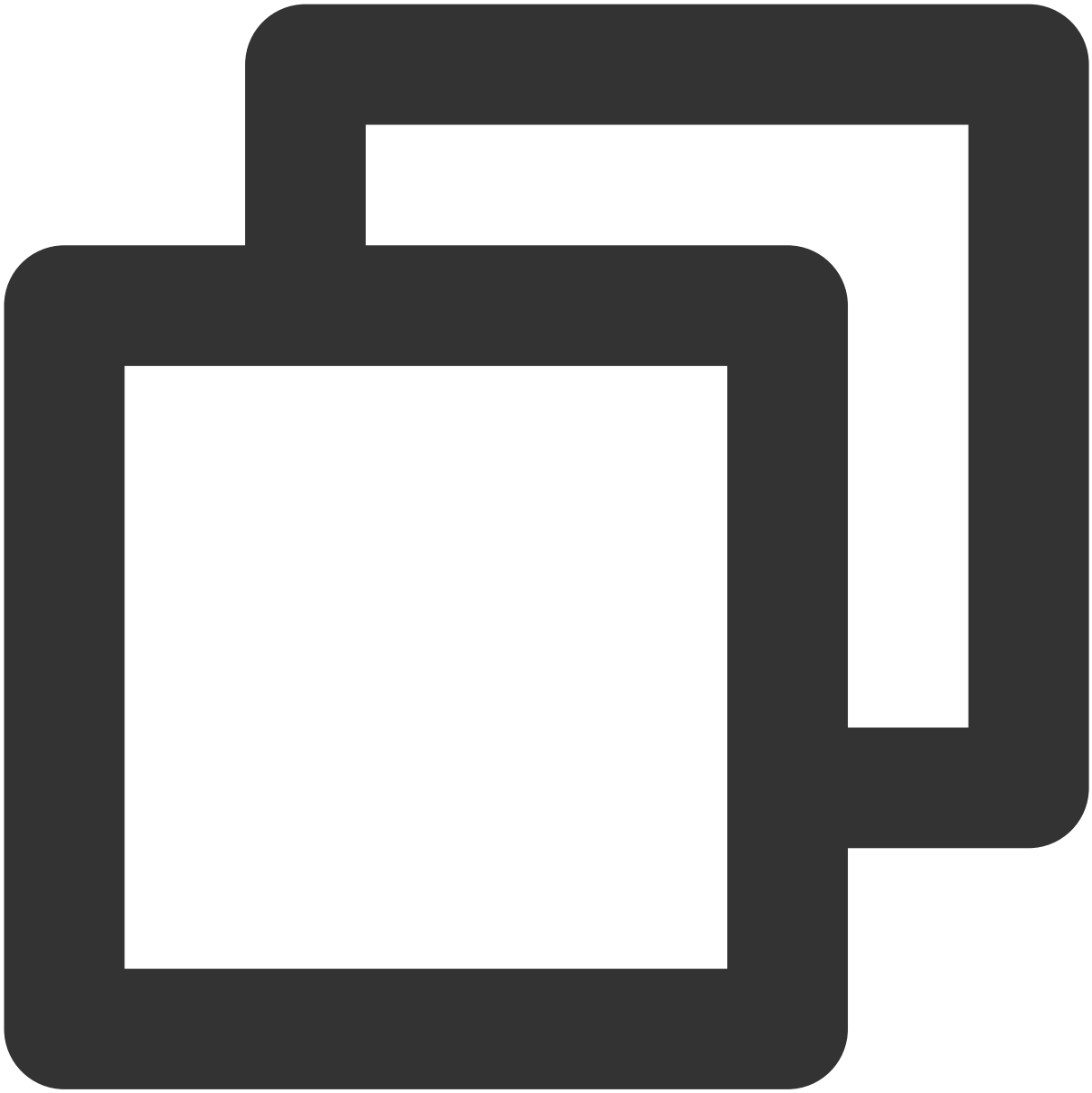
샘플 코드

아래 코드는 로컬 사용자의 스트림을 라이브 스트리밍 CDN에 게시합니다:
Java
ObjC
C++



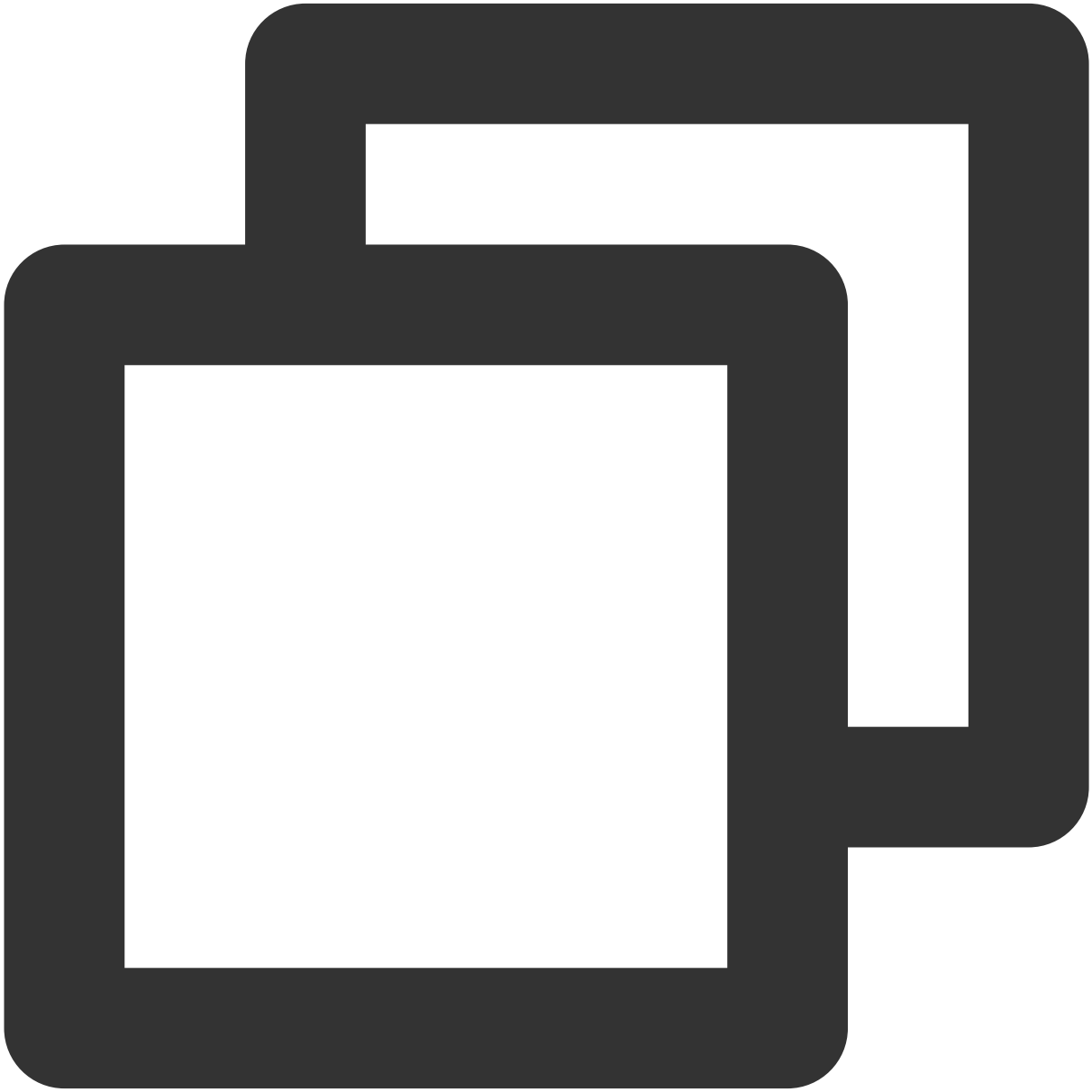
```
// 로컬 사용자의 스트림을 CDN에 게시
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
target.mode = TRTC_PublishBigStream_ToCdn;

TRTCCloudDef.TRTCPublishCdnUrl cdnUrl= new TRTCCloudDef.TRTCPublishCdnUrl();
cdnUrl.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = true;
target.cdnUrlList.add(cdnUrl);

mTRTCCloud.startPublishMediaStream(target, null, null);
```

```
```



```
// 로컬 사용자의 스트림을 CDN에 게시
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishBigStreamToCdn;

TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
cdnUrl.rtmpUrl = @"rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = YES;
```

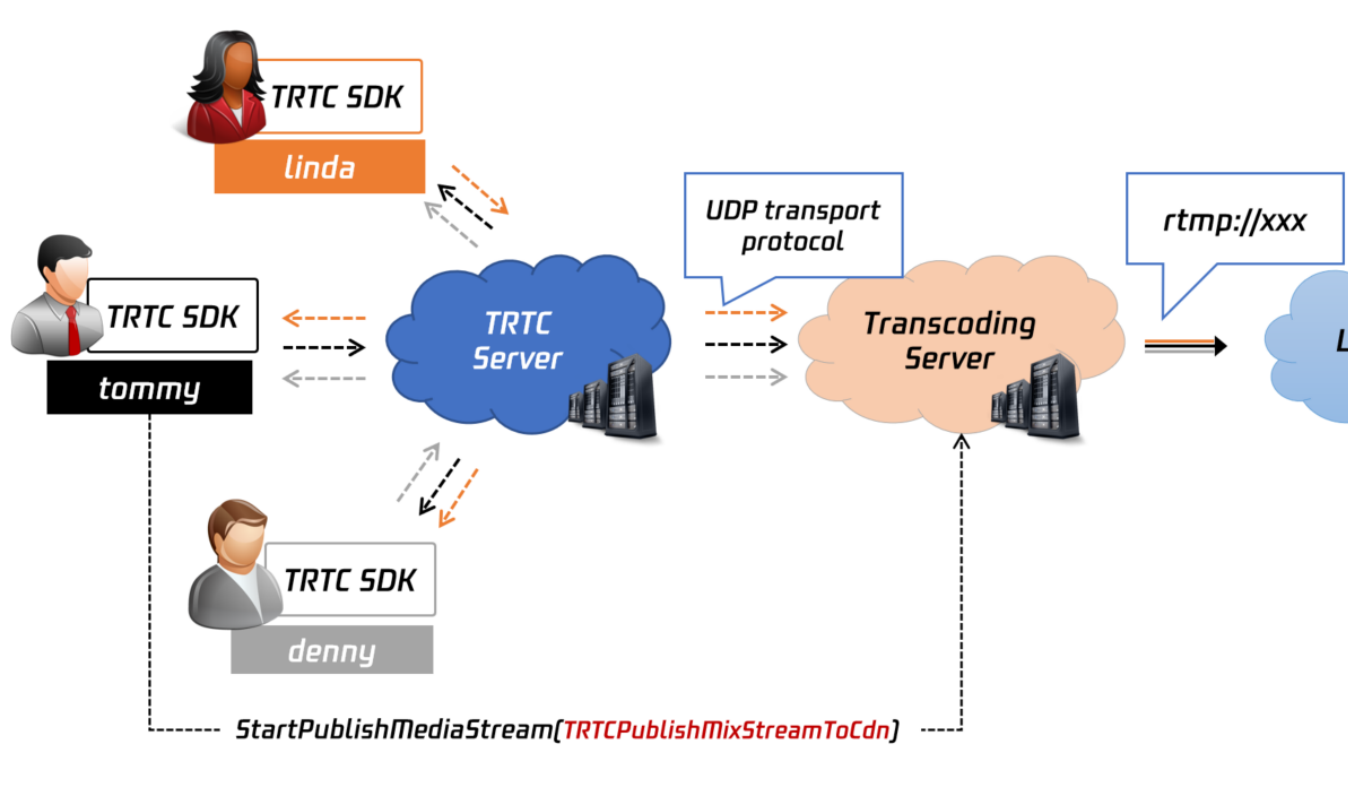
```
NSMutableArray* cdnUrlList = [NSMutableArray new];
[cdnUrlList addObject:cdnUrl];
target.cdnUrlList = cdnUrlList;

[_trtcCloud startPublishMediaStream:target encoderParam:nil mixingConfig:nil];
```



```
// 로컬 사용자의 스트림을 CDN에 게시
```

```
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;
```

혼합 스트림을 CDN에 게시



기능 소개

startPublishMediaStream을 호출하여 TRTC 룸에 있는 여러 사용자의 스트림을 하나의 스트림으로 믹싱하고 스트림을 CDN에 게시할 수 있습니다. API의 `TRTCStreamEncoderParam` 및 `TRTCStreamMixingConfig` 매개변수를 사용하면 스트림 믹싱 및 트랜스 코딩에 대한 세부 정보를 결정할 수 있습니다.
스트림은 클라우드에서 먼저 디코딩 및 믹싱된 다음 지정한 스트림 믹싱 매개변수(`TRTCStreamMixingConfig`) 및 트랜스 코딩 매개변수(`TRTCStreamEncoderParam`)에 따라 다시 인코딩됩니다. 그 후에 CDN에 게시됩니다.
이 모드에서는 추가 트랜스 코딩 요금이 부과됩니다.

작업 가이드

아래 단계에 따라 한 방에서 여러 사용자의 스트림을 믹싱하고 믹싱된 스트림을 CDN에 게시할 수 있습니다.
1 TRTCPublishTarget 객체를 생성하고 TRTCPublishTarget의 mode를 `TRTCPublishMixStreamToCdn` 으로 설정합니다.
2. TRTCPublishTarget 객체의 cdnUrlList를 하나 이상의 CDN 주소(보통 `rtmp://` 로 시작)로 설정합니다. Tencent Cloud CDN에 게시하는 경우 isInternalLine을 true로 설정합니다. 그렇지 않으면 false로 설정하십시오.
3. 인코딩 매개변수 설정(`TRTCStreamEncoderParam`):
비디오 인코딩 매개변수: 해상도, 프레임 레이트(15fps 권장), 비트 레이트 및 GOP(3초 권장)를 지정합니다. 비트 레이트와 해상도는 적절한 매칭 관계가 있습니다. 아래 표에는 몇 가지 권장 해상도 및 비트 레이트 설정이 나와 있습니다.
오디오 인코딩 매개변수: startLocalAudio를 호출할 때 전달한 AudioQuality 값에 따라 코덱, 비트 레이트, 샘플링 레이트 및 사운드 채널을 지정합니다.

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

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

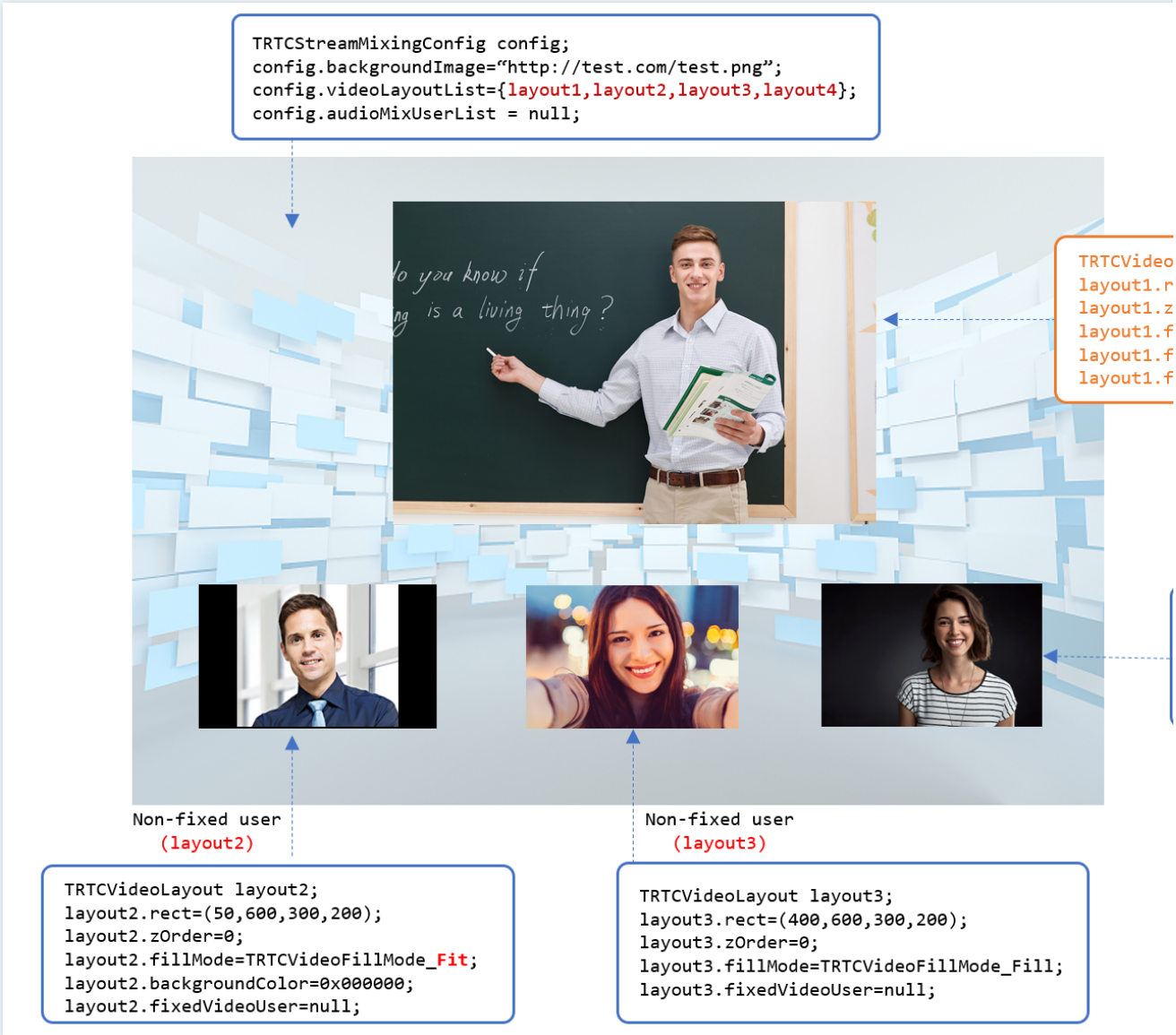
1. 오디오 믹싱 및 비디오 레이아웃에 대한 매개변수 설정(`TRTCStreamMixingConfig`):
오디오 믹싱 매개변수(audioMixUserList): 이 매개변수를 비워 두어 방의 모든 오디오를 믹싱하거나 오디오를 믹싱하려는 사용자의 ID로 설정할 수 있습니다.

비디오 레이아웃 매개변수(videoLayoutList): 비디오 레이아웃은 배열에 의해 결정됩니다. 배열의 각 TRTCVideoLayout 요소는 비디오 창의 위치, 크기 및 배경색을 결정합니다. **fixedVideoUser**를 지정하면 TRTCVideoLayout 요소에 의해 정의된 창에 특정 사용자의 비디오가 표시됩니다. fixedVideoUser를 null로 설정하면 TRTC 서버가 창에 표시할 비디오를 결정합니다.

사례1: 4명의 사용자 스트림을 혼합하고 이미지를 배경으로 사용

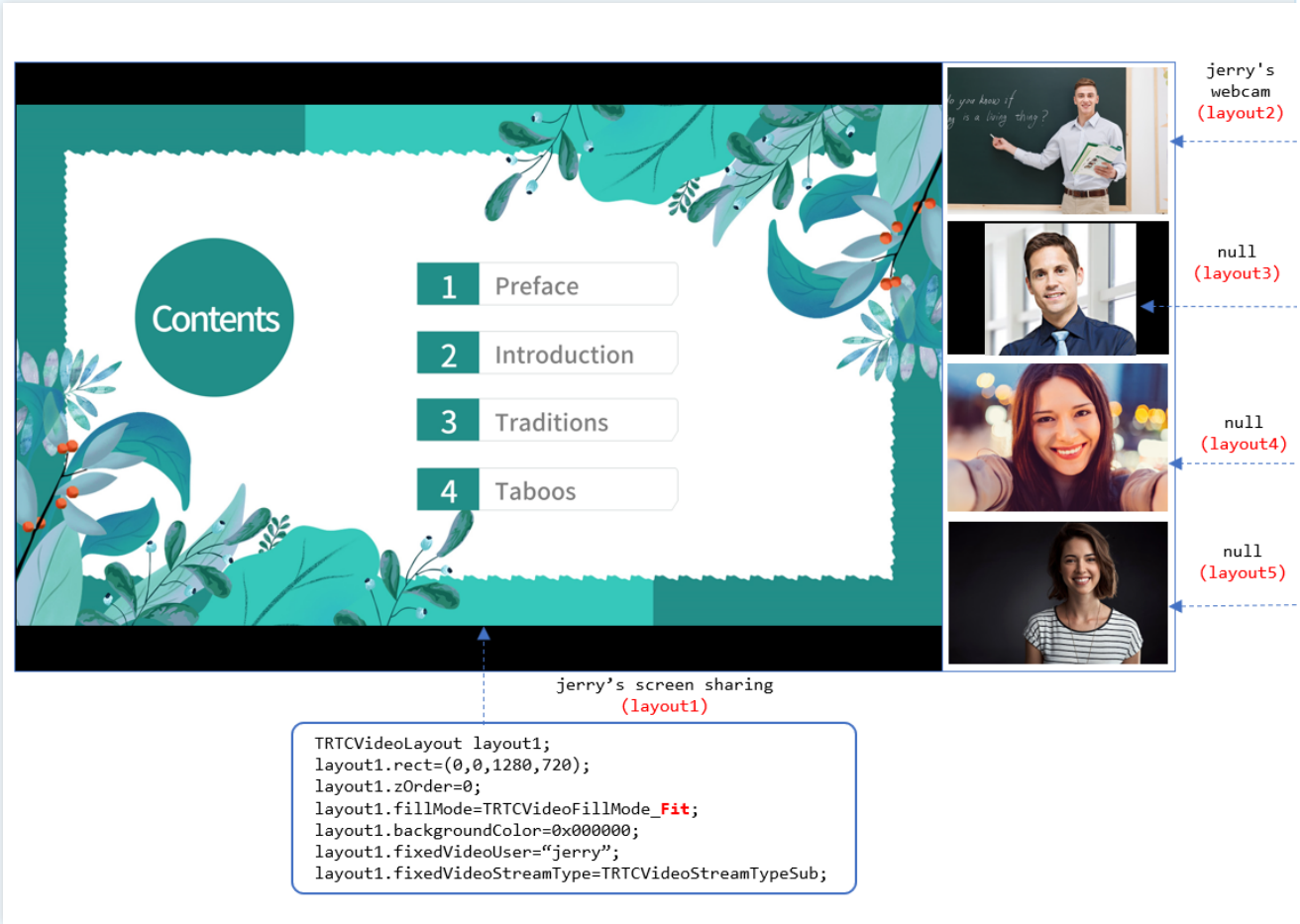
layout1: 사용자 jerry의 카메라 비디오의 위치(캔버스의 위쪽 절반)와 크기(640x480)를 지정합니다.

layout2, layout3, layout4: 사용자 Id가 지정되지 않았기 때문에 TRTC는 자체 규칙에 따라 창에 다른 세 사용자의 비디오를 표시합니다.



사례2: 한 사용자의 카메라 비디오와 화면에 다른 세 사용자의 카메라 비디오를 혼합

layout1: 사용자 jerry의 화면의 위치(왼쪽)와 크기(1280x720)를 지정합니다. 사용된 렌더링 모드는 가로 세로 맞춤(Fit)이며 배경색은 검정색입니다.
layout2: 사용자 jerry의 카메라 비디오의 위치(오른쪽 상단)와 크기(300x200)를 지정합니다. 사용된 렌더링 모드는 화면 채우기(Fill)입니다.
layout3, layout4, layout5: 사용자 Id가 지정되지 않았기 때문에 TRTC는 자체 규칙에 따라 창에 다른 세 사용자의 비디오를 표시합니다.

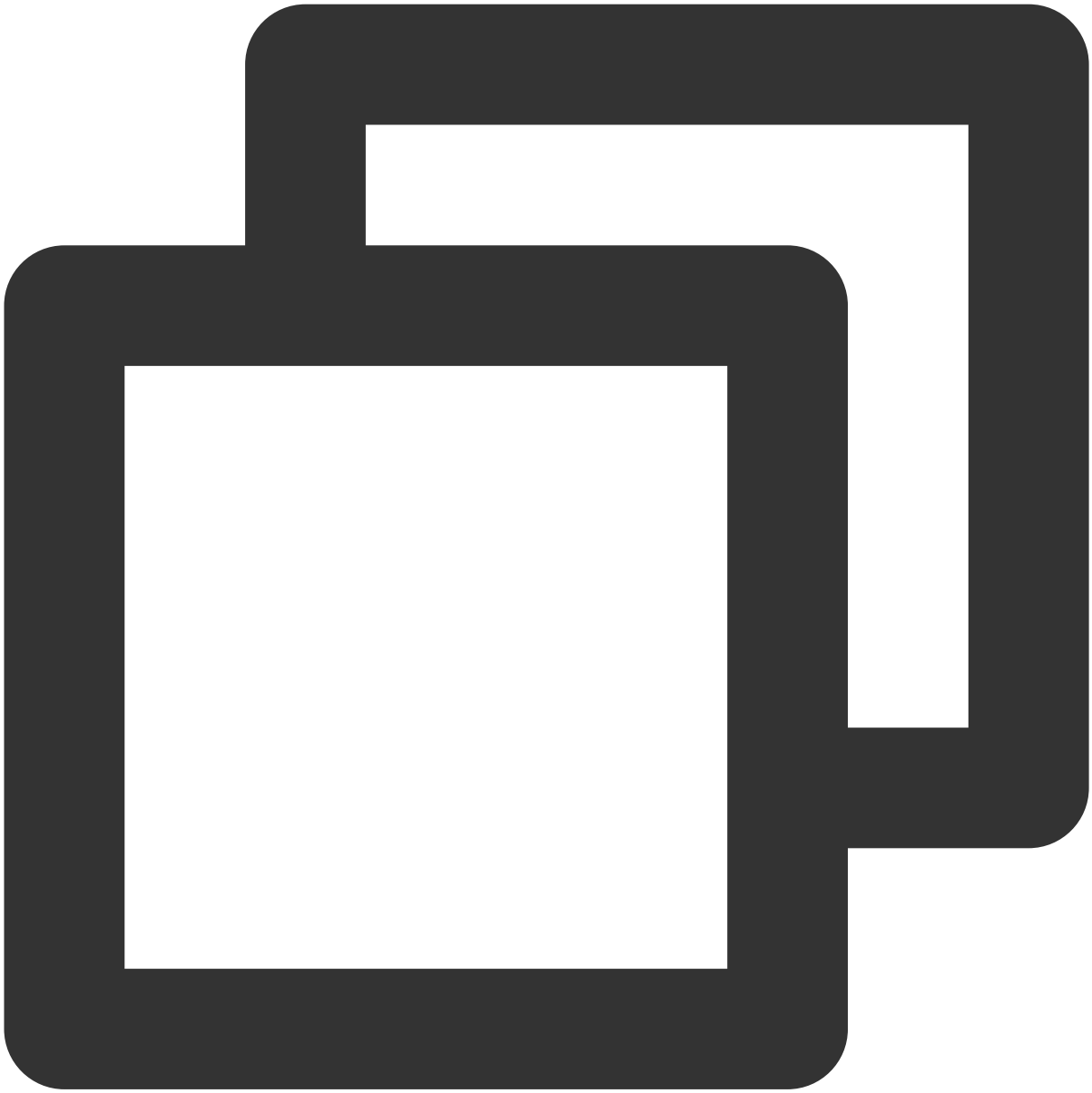


2. startPublishMediaStream을 호출합니다. onStartPublishMediaStream 콜백에서 반환된 taskId 매개변수가 비어 있지 않으면 로컬 API 호출이 성공한 것입니다.
3. 스트림 믹싱 매개변수(예: 비디오 레이아웃)를 변경하려면 updatePublishMediaStream을 호출하고 6단계에서 반환된 taskId와 새로운 TRTCStreamMixingConfig 매개변수를 전달합니다. 릴레이 중에는 TRTCStreamEncoderParam을 변경하지 않는 것이 좋습니다. 그렇게 하면 CDN 재생의 안정성에 영향을 미치기 때문입니다.
4. 게시를 중지하려면 stopPublishMediaStream을 호출하고 onStartPublishMediaStream에서 반환된 taskId를 전달합니다.

샘플 코드

아래 코드는 한 방에 있는 여러 사용자의 스트림을 믹싱하고 그 결과를 CDN에 게시합니다.

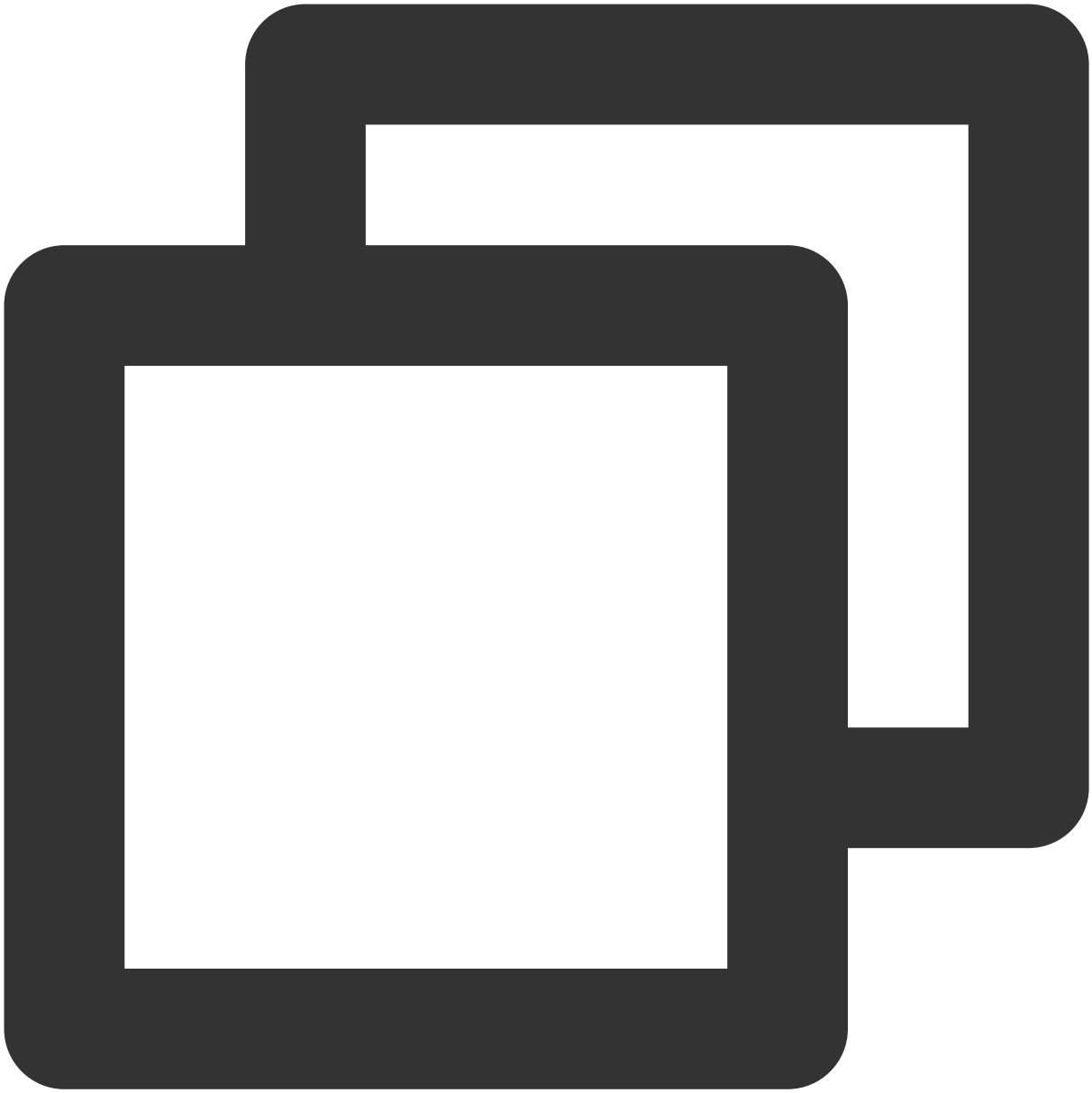
Java
ObjC
C++



```
// 게시 모드를 TRTC_PublishMixedStream_ToCdn으로 지정
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
target.mode = TRTC_PublishMixedStream_ToCdn;

// 게시된 CDN 푸시 스트림 주소 지정
TRTCCloudDef.TRTCPublishCdnUrl cdnUrl= new TRTCCloudDef.TRTCPublishCdnUrl();
```

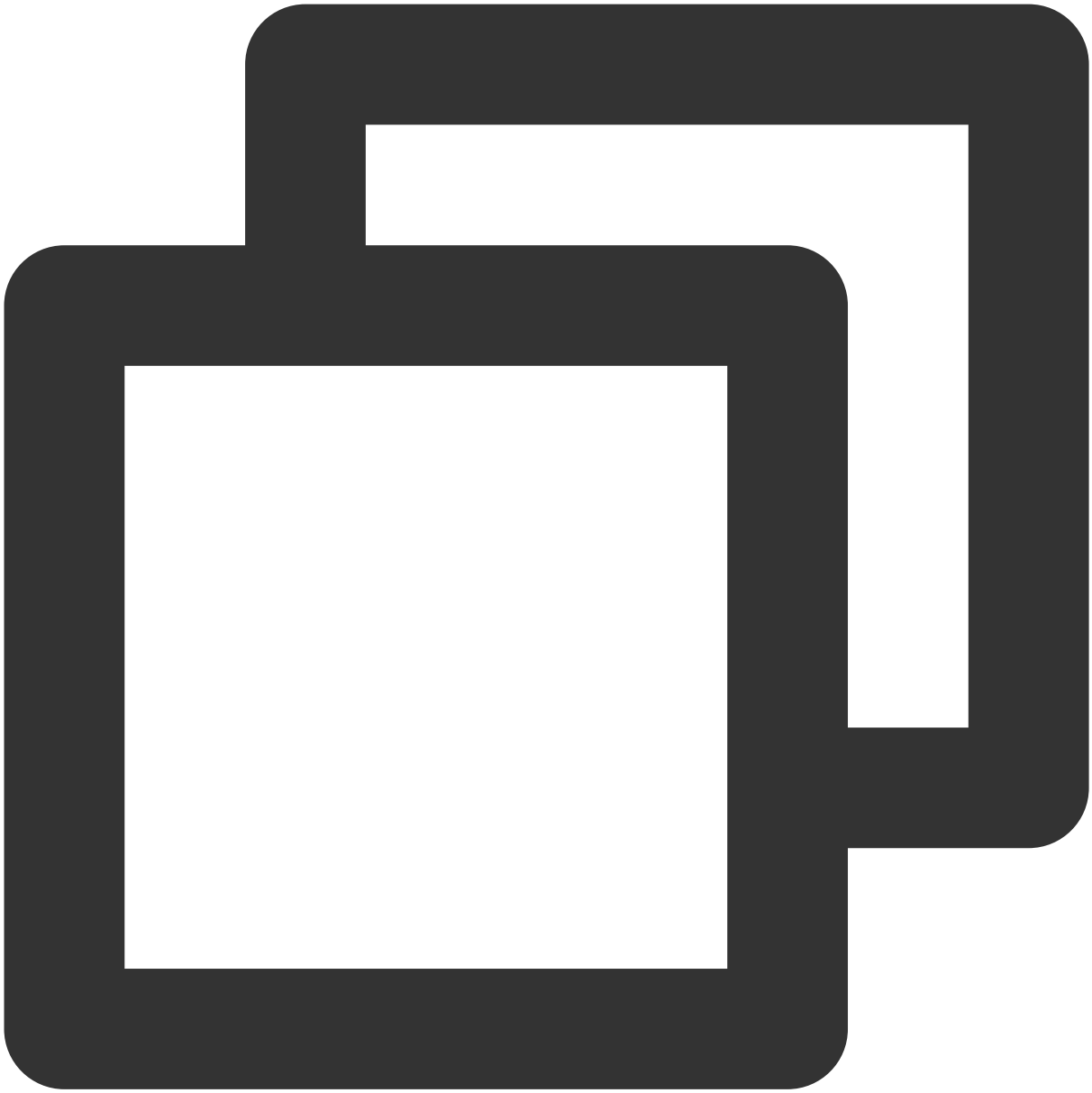
```
cdnUrl.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = true;
target.cdnUrlList.add(cdnUrl);


// 믹싱된 오디오/비디오 스트림의 2차 인코딩 매개변수 설정
TRTCCloudDef.TRTCStreamEncoderParam encoderParam
    = new TRTCCloudDef.TRTCStreamEncoderParam();
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;

// 화면의 레이아웃 매개변수 설정
TRTCCloudDef.TRTCStreamMixingConfig mixingConfig =
      new TRTCCloudDef.TRTCStreamMixingConfig();
TRTCCloudDef.TRTCVideoLayout layout1 = new TRTCCloudDef.TRTCVideoLayout();
layout1.zOrder = 0;
layout1.x = 0;
layout1.y = 0;
layout1.width = 720;
layout1.height = 1280;
layout1.fixedVideoStreamType = TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SUB;
layout1.fixedVideoUser.intRoomId = 1234;
layout1.fixedVideoUser.userId = "mike";

TRTCCloudDef.TRTCVideoLayout layout2 = new TRTCCloudDef.TRTCVideoLayout();
layout2.zOrder = 0;
layout2.x = 1300;
layout2.y = 0;
layout2.width = 300;
layout2.height = 200;
layout2.fixedVideoStreamType = TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG;
layout2.fixedVideoUser.intRoomId = 1234;
layout2.fixedVideoUser.userId = "mike";

TRTCCloudDef.TRTCVideoLayout layout3 = new TRTCCloudDef.TRTCVideoLayout();
layout3.zOrder = 0;
layout3.x = 1300;
layout3.y = 220;
layout3.width = 300;
layout3.height = 200;
```

```
layout3.fixedVideoStreamType = TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SUB;
layout3.fixedVideoUser = null;

mixingConfig.videoLayoutList.add(layout1);
mixingConfig.videoLayoutList.add(layout2);
mixingConfig.videoLayoutList.add(layout3);
mixingConfig.audioMixUserList = null;

// 혼합 스트림 시작
mTRTCCloud.startPublishMediaStream(target, encoderParam, mixingConfig);
```



```
// 게시 모드를 TRTCPublishMixStreamToCdn으로 지정
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishMixStreamToCdn;

// 게시된 CDN 푸시 스트림 주소 지정
TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
cdnUrl.rtmpUrl = @"rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = YES;

NSMutableArray* cdnUrlList = [NSMutableArray new];
```

```
[cdnUrlList addObject:cdnUrl];
target.cdnUrlList = cdnUrlList;

// 믹싱된 오디오/비디오 스트림의 2차 인코딩 매개변수 설정
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;

// 화면의 레이아웃 매개변수 설정
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;

// 혼합 스트림 시작
[_trtcCloud startPublishMediaStream:target encoderParam:encoderParam mixingConfig:c
```



```
// 게시 모드를 TRTCPublishMixStreamToCdn으로 지정
TRTCPublishTarget target;
target.mode = TRTCPublishMode::TRTCPublishMixStreamToCdn;

// 게시된 CDN 푸시 스트림 주소 지정
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;

// 믹싱된 오디오/비디오 스트림의 2차 인코딩 매개변수 설정
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;

// 화면의 레이아웃 매개변수 설정
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;

// 혼합 스트림 시작
trtc->startPublishMediaStream(&target, &encoder_param, &config);
delete fixedVideoUser0;
delete fixedVideoUser1;
delete[] video_layout_list;
```

FAQ

1. CDN 스트림의 상태를 수신할 수 있습니까? 오류가 발생하면 어떻게 해야 합니까?
답변: `onCdnStreamStateChanged` 콜백 이벤트를 수신하여 CDN 작업 중계의 최신 상태를 얻을 수 있습니다.
2. 단일 스트림 게시에서 혼합 스트림 게시로 전환하려면 어떻게 합니까? 먼저 게시를 중지하고 새 릴레이 작업을 만들어야 합니까?
답변: 단일 스트림 게시에서 혼합 스트림 게시로 전환하려면 현재 작업의 `taskid` 를 전달하여 `updatePublishMediaStream` 을 호출하기만 하면 됩니다. 게시 프로세스의 안정성을 보장하기 위해 단일 스트림 게시에서 오디오만 또는 비디오만 믹싱 및 게시로 전환할 수 없습니다. 기본적으로 단일 스트림을 게시하면 오디오 및 비디오 데이터가 모두 게시됩니다. 혼합 스트림 게시로 전환하는 경우 오디오와 비디오도 모두 게시해야 합니다.
3. 오디오 없이 비디오만 믹싱하는 방법은 무엇입니까?
답변: `TRTCStreamEncodeParam` 에서 오디오 매개변수를 설정하지 말고 `TRTCStreamMixingConfig` 의 `audioMixUserList` 를 비워 두십시오.
4. 혼합 스트림에 워터마크를 추가할 수 있습니까?
답변: 예, `TRTCStreamMixingConfig` 의 `watermarkList` 를 사용하여 워터마크를 설정할 수 있습니다.
5. 온라인 학습 시나리오에서 교사가 공유하는 화면을 믹싱할 수 있나요?
답변: 예, 할 수 있습니다. 화면을 서브스트림으로 퍼블리싱하고 교사의 카메라 영상과 화면을 믹싱하는 것이 좋습니다. 스트림 믹싱 매개변수를 지정할 때 `TRTCVideoLayout` 의 `fixedVideoStreamType` 을 `TRTCVideoStreamTypeSub` 로 설정합니다.

6. 사전 설정 레이아웃을 사용할 때 오디오는 어떻게 믹싱됩니까?
답변: 사전 설정 레이아웃을 사용하는 경우 TRTC는 방에 있는 최대 16명의 사용자의 오디오를 믹싱합니다.

관련 요금

요금 계산

스트림을 믹싱하면 MCU 클러스터가 클라우드에서 스트림을 디코딩하고 다시 인코딩하므로 추가 트랜스 코딩 요금이 부과됩니다. 트랜스 코딩 요금은 트랜스 코딩된 스트림의 해상도와 트랜스 코딩 시간에 따라 다릅니다. CDN 릴레이 요금은 매월 사용되는 최대 대역폭을 기준으로 청구됩니다. 자세한 내용은 혼합 트랜스코딩 및 릴레이 과금을 참고하십시오.

요금 절약

클라이언트 측 API를 사용하는 경우 다음 조건 중 하나가 충족되면 백엔드에서 스트림 믹싱이 중지됩니다.
스트림을 믹싱하기 위해 `startPublishMediaStream` 을 호출한 앵커가 방을 나갔습니다.
`stopPublishMediaStream` 을 호출하여 스트림 믹싱을 중지합니다.
다른 모든 경우에 TRTC는 계속해서 클라우드에서 스트림을 혼합합니다. 따라서 비용을 줄이기 위해 더 이상 스트림을 혼합하지 않으려면 위의 방법 중 하나를 사용하여 중지하십시오.

고급 권한 제어 기능 활성화

최종 업데이트 날짜: : 2024-07-24 16:43:38

개요

방 입장을 제한하거나 마이크 켜기를 제한하고 싶거나, 지정한 사용자만 방에 입장할 수 있거나 마이크를 켤 수 있도록 허용하고 싶은 경우, 또는 클라이언트에서 권한을 판단할 경우 쉽게 크래킹 공격을 받을 수 있어 걱정되는 경우 **고급 권한 제어 활성화**를 고려해볼 수 있습니다.
다음과 같은 시나리오에서는 고급 권한 제어 활성화가 필요하지 않습니다.
시나리오 1: 많은 사람이 들어올수록 좋기 때문에 방 입장 제한이 필요 없는 경우.
시나리오 2: 해커의 크래킹에 대한 클라이언트 방어가 급하게 필요하지 않은 경우.
다음과 같은 시나리오에서는 더 최적화된 보안을 위해 고급 권한 제어 활성화를 권장합니다.
시나리오 1: 보안성 요구가 비교적 높은 비디오 통화 또는 음성 통화가 필요한 경우.
시나리오 2: 방별로 각각의 입장 권한을 설정해야 하는 경우.
시나리오 3: 관중의 마이크 켜기 권한을 제어해야 하는 경우.

지원 플랫폼

iOS	Android	Mac OS	Windows	Electron	Web
✓	✓	✓	✓	✓	✓

고급 권한 제어의 원리

고급 권한 제어를 활성화한 후에는 TRTC 백그라운드 서비스 시스템이 UserSig '방 입장 티켓'만을 인증하는 것이 아니라 **PrivateMapKey**라고 하는 '권한 티켓'까지 인증합니다. 권한 티켓에는 암호화된 roomid와 "비트 권한 리스트"가 포함되어 있습니다.
PrivateMapKey에는 roomid가 포함되어 있어 사용자가 UserSig만 제공하고 PrivateMapKey를 제공하지 않는 경우 지정된 방에 입장할 수 없습니다.
PrivateMapKey 상의 '비트 권한 리스트'는 byte의 8개 비트를 사용하며, 각각 해당 티켓을 가지고 있는 사용자를 대표합니다. 해당 티켓이 지정하고 있는 방에서 가지는 8가지 기능 권한은 다음과 같습니다.

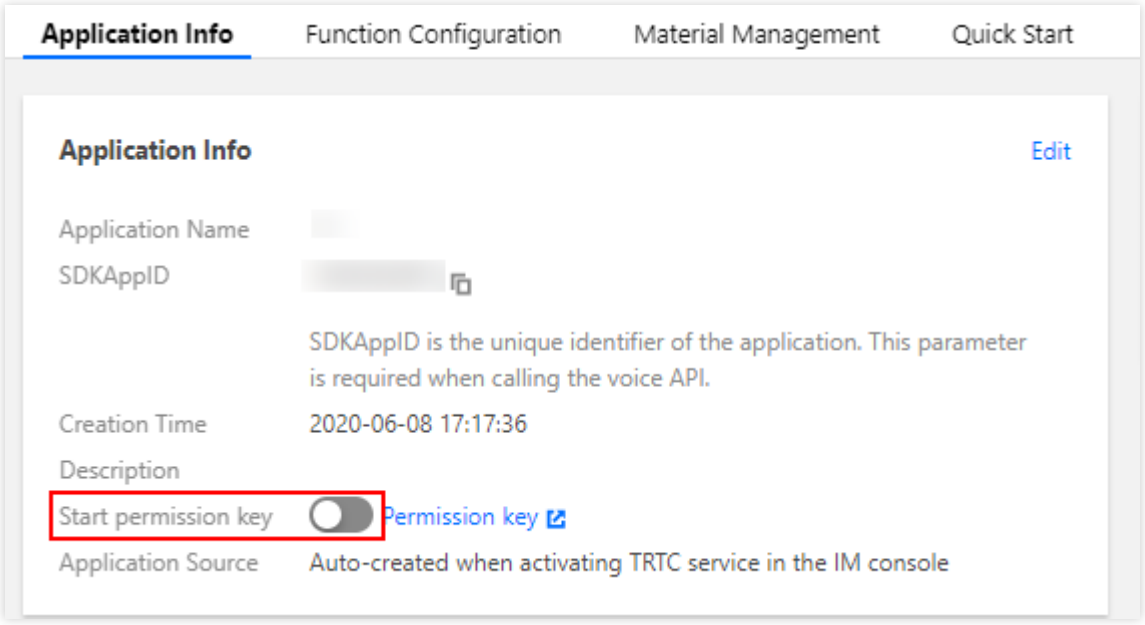
비트 위치	이진법 표시	10진법 숫자	권한 의미
1번째 비트	0000 0001	1	방 생성 권한

2번째 비트	0000 0010	2	방 입장 권한
3번째 비트	0000 0100	4	오디오 발신 권한
4번째 비트	0000 1000	8	오디오 수신 권한
5번째 비트	0001 0000	16	비디오 발신 권한
6번째 비트	0010 0000	32	비디오 수신 권한
7번째 비트	0100 0000	64	비디오 서브 채널(즉, 화면 공유) 발신 권한
8번째 비트	1000 0000	128	비디오 서브 채널(즉, 화면 공유) 수신 권한

고급 권한 제어 활성화

1단계: TRTC 콘솔에서 고급 권한 제어를 활성화합니다.

1. Tencent Cloud TRTC 콘솔에서 왼쪽에 있는 **애플리케이션 관리**를 클릭합니다.
2. 오른쪽 애플리케이션 리스트에서 고급 권한 제어를 활성화할 애플리케이션을 선택하고 **기능 설정**을 클릭합니다.
3. '기능 설정' 페이지에서 **고급 권한 제어 활성화** 버튼을 활성화하고 **확인**을 누르면 고급 권한 제어가 활성화됩니다.



주의사항 :

SDKAppid의 고급 권한 제어를 활성화한 후에는 해당 SDKAppid를 사용하는 모든 사용자는 TRTCParams에 `privateMapKey` 매개변수를 전달해야만 방에 입장(2단계와 같이)할 수 있습니다. 만약 온라인으로 해당 SDKAppid를 사용하는 사용자가 있는 경우 해당 기능을 활성화하지 마십시오.

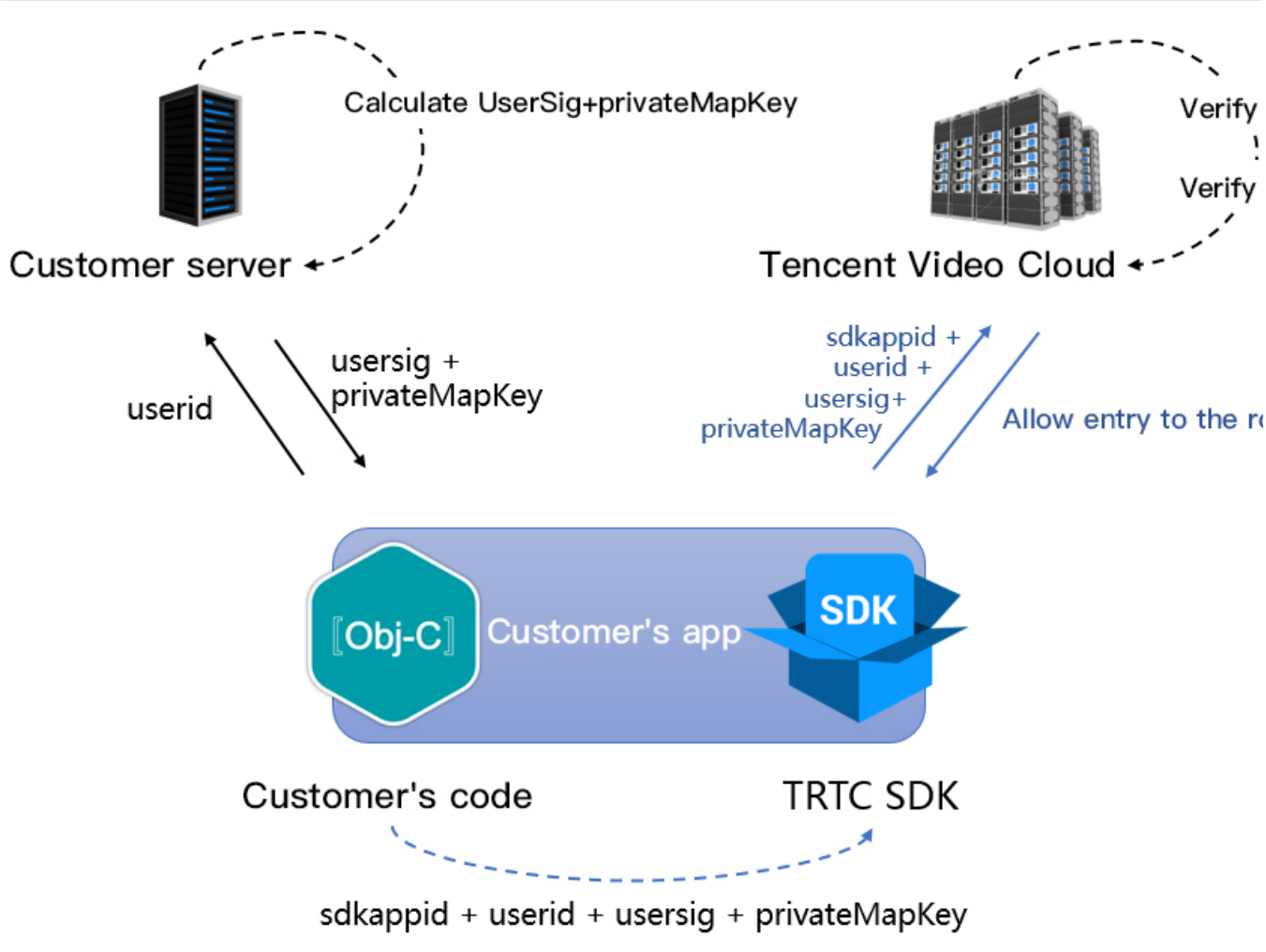
2단계: 귀하의 서버에서 PrivateMapKey를 계산합니다.

PrivateMapKey는 클라이언트에서의 리버스 크래킹을 방지하기 위한 것으로 '비회원도 고급 레벨 방 입장 가능'한 크래킹 버전이 나타날 수 있어, 서버에서 계산 후 다시 App에 반환하는 방식만 가능하며 App에서 직접 계산할 수 없습니다.

Tencent Cloud는 Java, GO, PHP, Node.js, Python, C# 및 C++ 버전의 PrivateMapKey 계산 코드를 제공하며, 직접 다운로드하여 서버에 통합할 수 있습니다.

언어 버전	주요 함수	다운로드 링크
Java	`genPrivateMapKey` 및 `genPrivateMapKeyWithStringRoomID`	Github
GO	`GenPrivateMapKey` 및 `GenPrivateMapKeyWithStringRoomID`	Github
PHP	`genPrivateMapKey` 및 `genPrivateMapKeyWithStringRoomID`	Github
Node.js	`genPrivateMapKey` 및 `genPrivateMapKeyWithStringRoomID`	Github
Python	`genPrivateMapKey` 및 `genPrivateMapKeyWithStringRoomID`	Github
C#	`genPrivateMapKey` 및 `genPrivateMapKeyWithStringRoomID`	Github
C++	`genPrivateMapKey` 및 `genPrivateMapKeyWithStringRoomID`	GitHub

3단계: 귀하의 서버에서 PrivateMapKey를 App으로 전달합니다.



상기 이미지와 같이 귀하의 서버에서 PrivateMapKey를 계산한 후 App으로 전달할 수 있으며, App은 두 가지 방법을 통해 PrivateMapKey를 SDK로 전달할 수 있습니다.

방법1: enterRoom 시 SDK 전달

사용자 방 입장 권한을 제어하려면, `TRTCCloud` 의 `enterRoom` 인터페이스 호출 시 TRTCParams의 **privateMapKey** 매개변수를 설정하여 구현할 수 있습니다.
해당 경우의 방 입장 시 PrivateMapKey 검증 방법은 비교적 간단하며, 방 입장 전 사용자 권한을 정확하게 확인할 수 있는 시나리오에 매우 적합합니다.

방법2: 테스트용 인터페이스를 통해 SDK에 업데이트

라이브 방송 시나리오에서 관중이 마이크를 켜 호스트가 되는 마이크 연결 시나리오가 종종 존재합니다. 관중이 호스트가 되는 경우 TRTC가 다시 방 입장 시 입장 매개변수인 `TRTCParams` 에 존재하는 PrivateMapKey를 인증하며, PrivateMapKey의 유효기간을 '5분' 등으로 비교적 짧게 설정하면 인증에 실패되어 사용자가 방에서 퇴장될 수 있습니다.
해당 문제를 해결하기 위해서는 유효기간을 연장(예: '5분'을 '6시간'으로 변경)하는 방법 이외에도, 관중이 자신의 신분을 호스트로 변경하기 전 `switchRole` 을 통해 다시 귀하의 서버에 privateMapKey를 신청하고 SDK의 테스트용

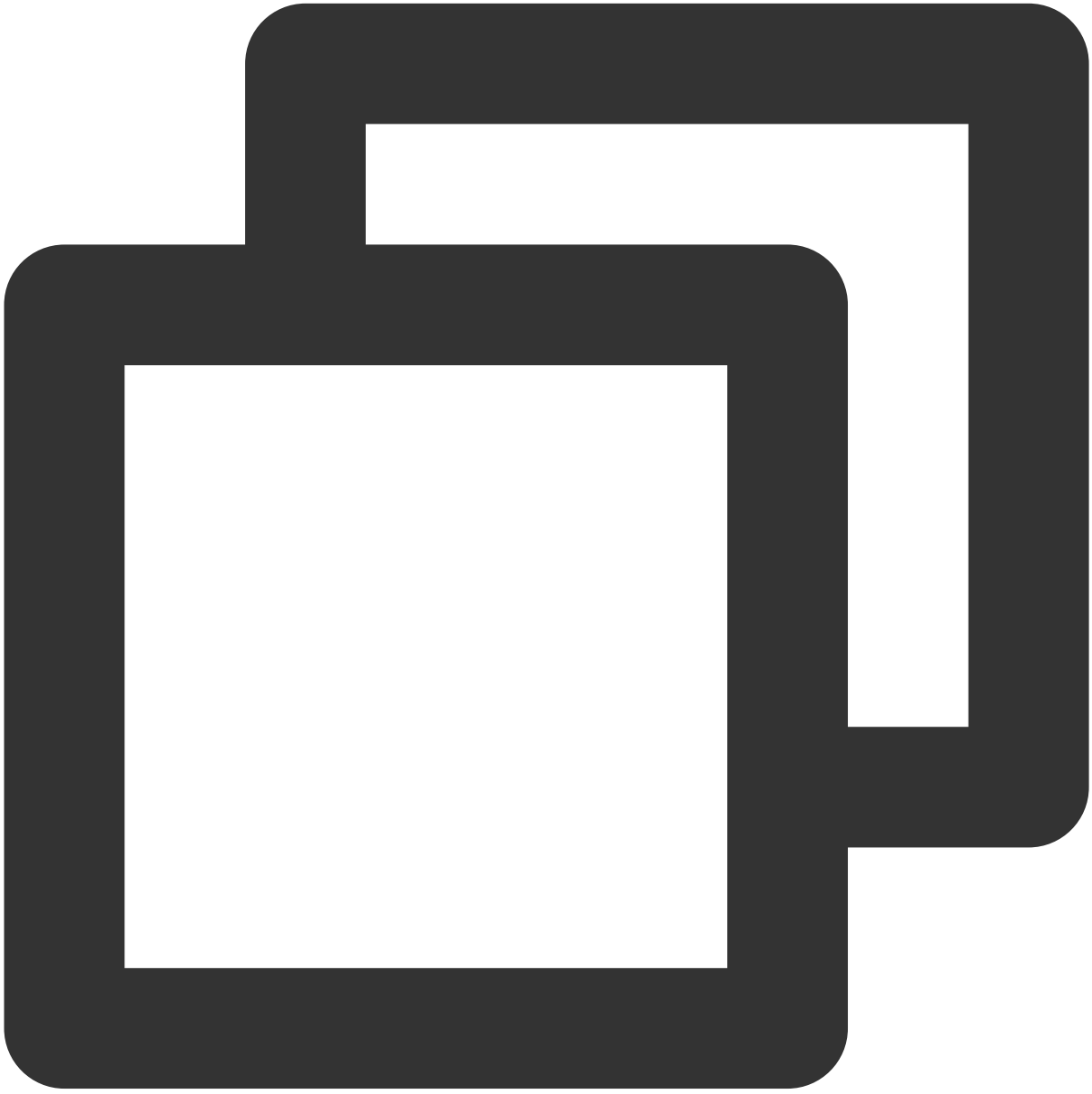
인터페이스인 `updatePrivateMapKey` 를 호출하여 SDK에 업데이트하는 방법이 있습니다. 코드 예시는 다음과 같습니다:

Android

iOS

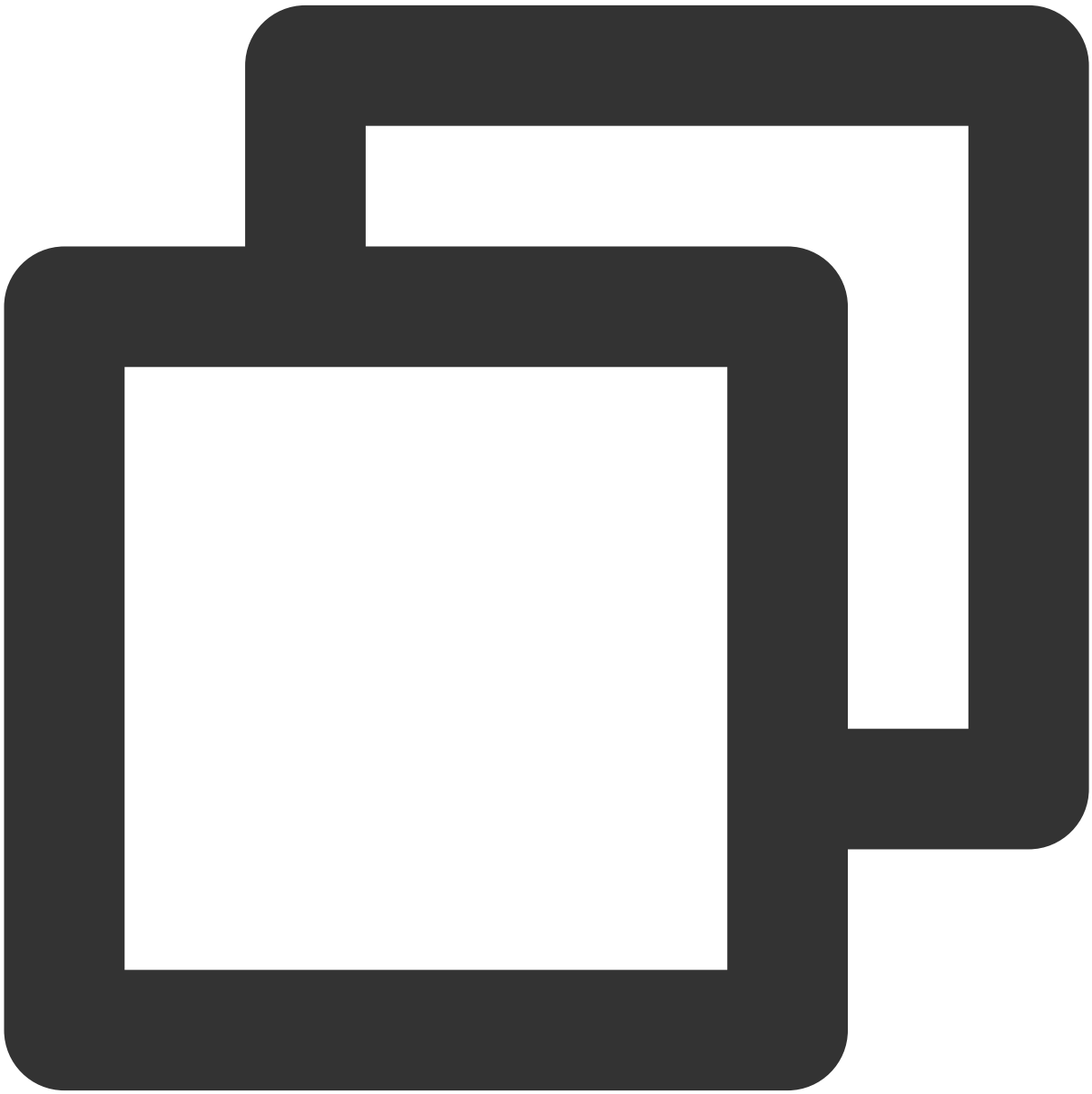
C++

C#



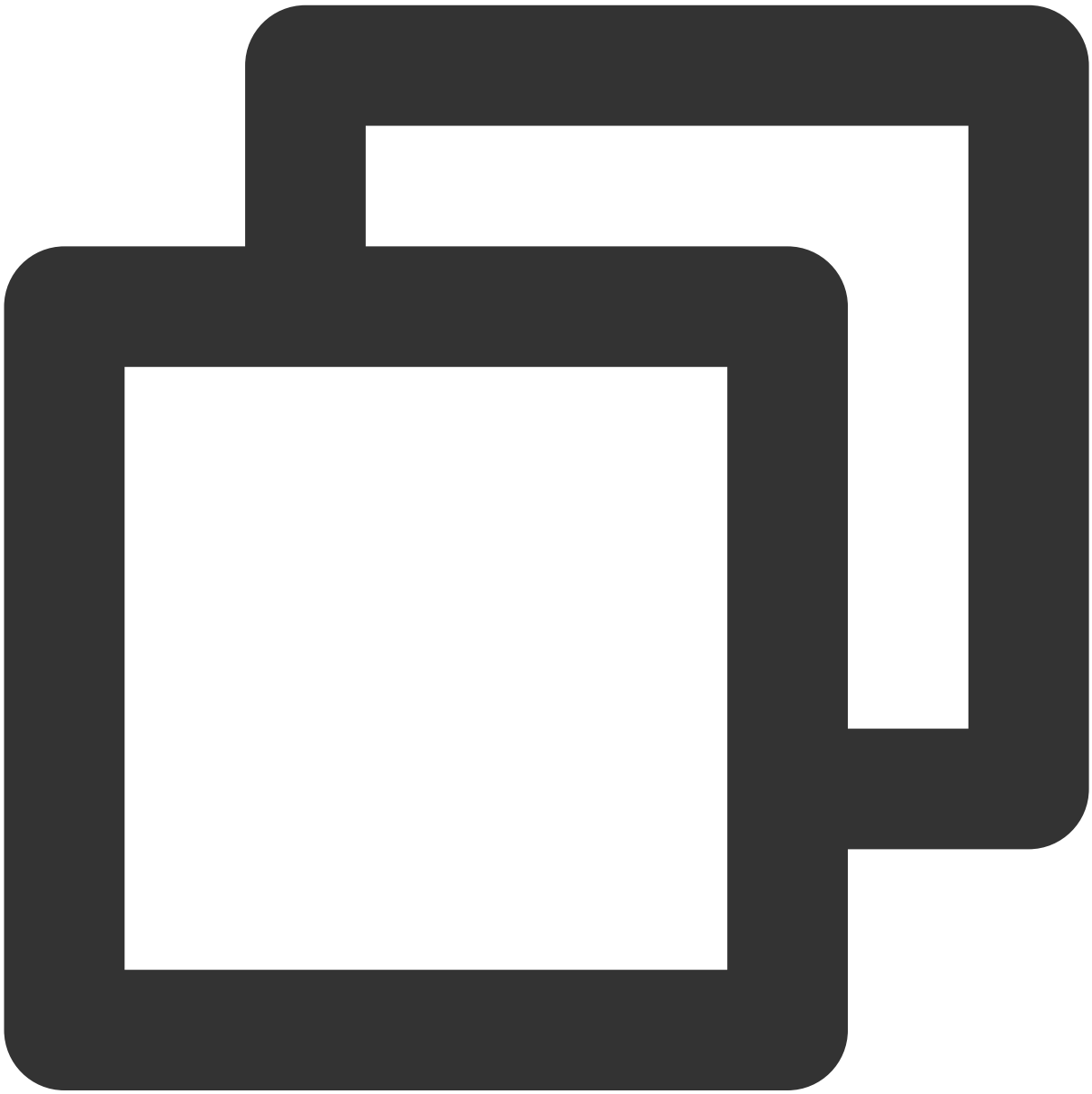
```
JSONObject jsonObject = new JSONObject();
try {
    jsonObject.put("api", "updatePrivateMapKey");
```

```
    JSONObject params = new JSONObject();
    params.put("privateMapKey", "xxxxx"); // 신규 privateMapKey 입력
    jsonObject.put("params", params);
    mTRTCCloud.callExperimentalAPI(jsonObject.toString());
} catch (JSONException e) {
    e.printStackTrace();
}
```

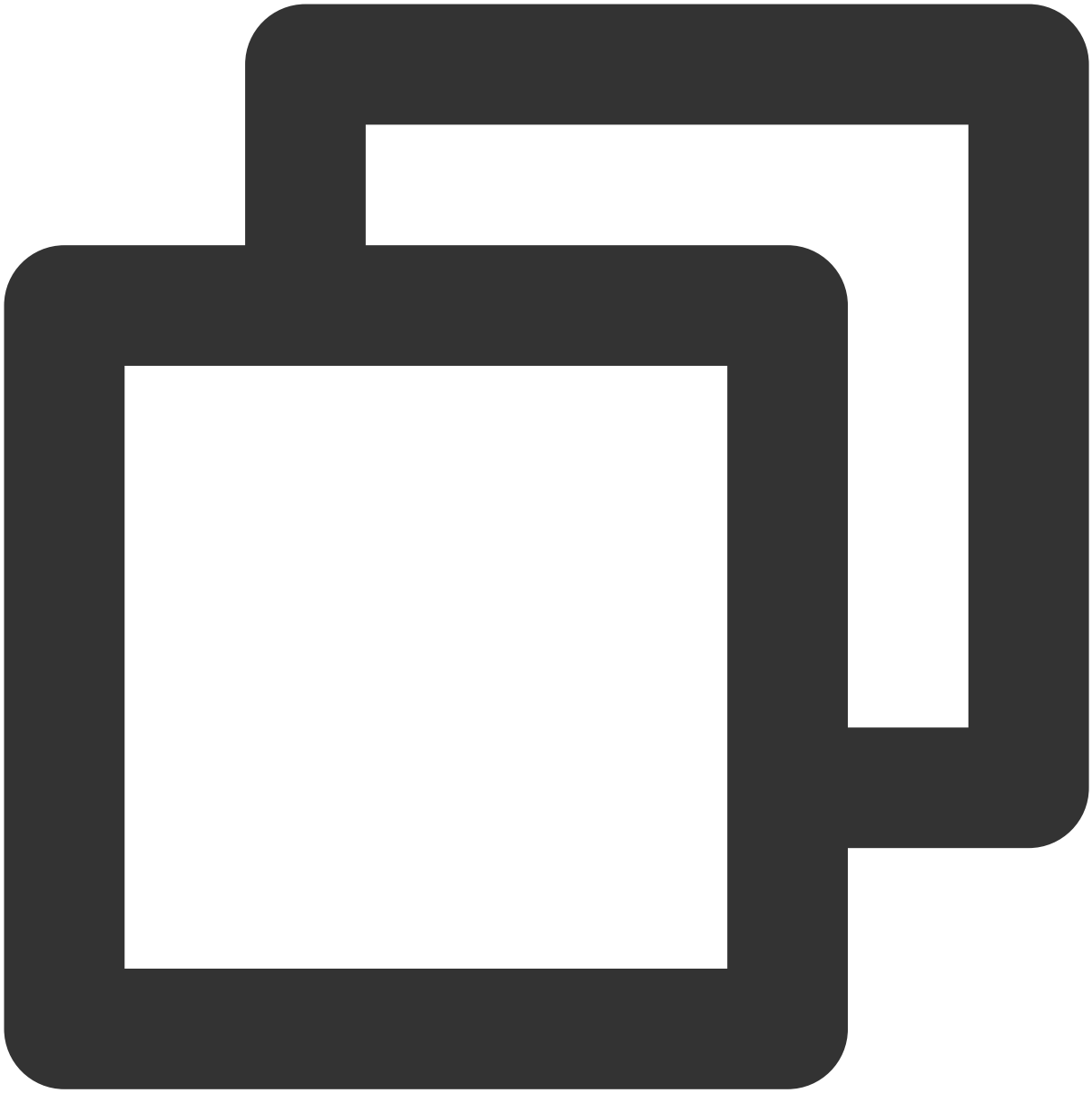


```
NSMutableDictionary *params = [[NSMutableDictionary alloc] init];
[params setObject:@"xxxxx" forKey:@"privateMapKey"]; // 신규 privateMapKey 입력
```

```
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);
```

FAQ

1. 온라인 상의 방에 들어갈 수 없는 이유는 무엇입니까?

방 권한 제어가 활성화되면 현재 SDKAppid의 모든 방은 `TRTCParams` 에 privateMapKey가 설정되어 있어야만 입장할 수 있습니다. 따라서 현재 온라인 서비스를 운영 중인 상태에서 온라인 버전에 privateMapKey 관련 로직이 없다면 해당 기능을 활성화하지 마십시오.

2. PrivateMapKey와 UserSig의 차이점은 무엇입니까?

UserSig은 TRTCParams의 필수 항목으로, 현재 사용자에게 TRTC 클라우드 서비스를 사용 권한이 있는지 여부를 검사하여 해커가 귀하의 SDKAppid 계정 내 트래픽을 도용하지 못하도록 방지합니다.
PrivateMapKey는 TRTCParams의 필수 항목이 아니며, 현재 사용자가 지정된 roomid 방에 입장할 수 있는 권한이 있는지 여부와 해당 방에서 가지는 권한을 검사합니다. 귀하의 비즈니스에 사용자의 신분 구분이 필요한 경우에만 활성화해야 합니다.

하드웨어 장치 테스트 Android&iOS&Windows&Mac

최종 업데이트 날짜: : 2024-07-24 16:43:38

콘텐츠 소개

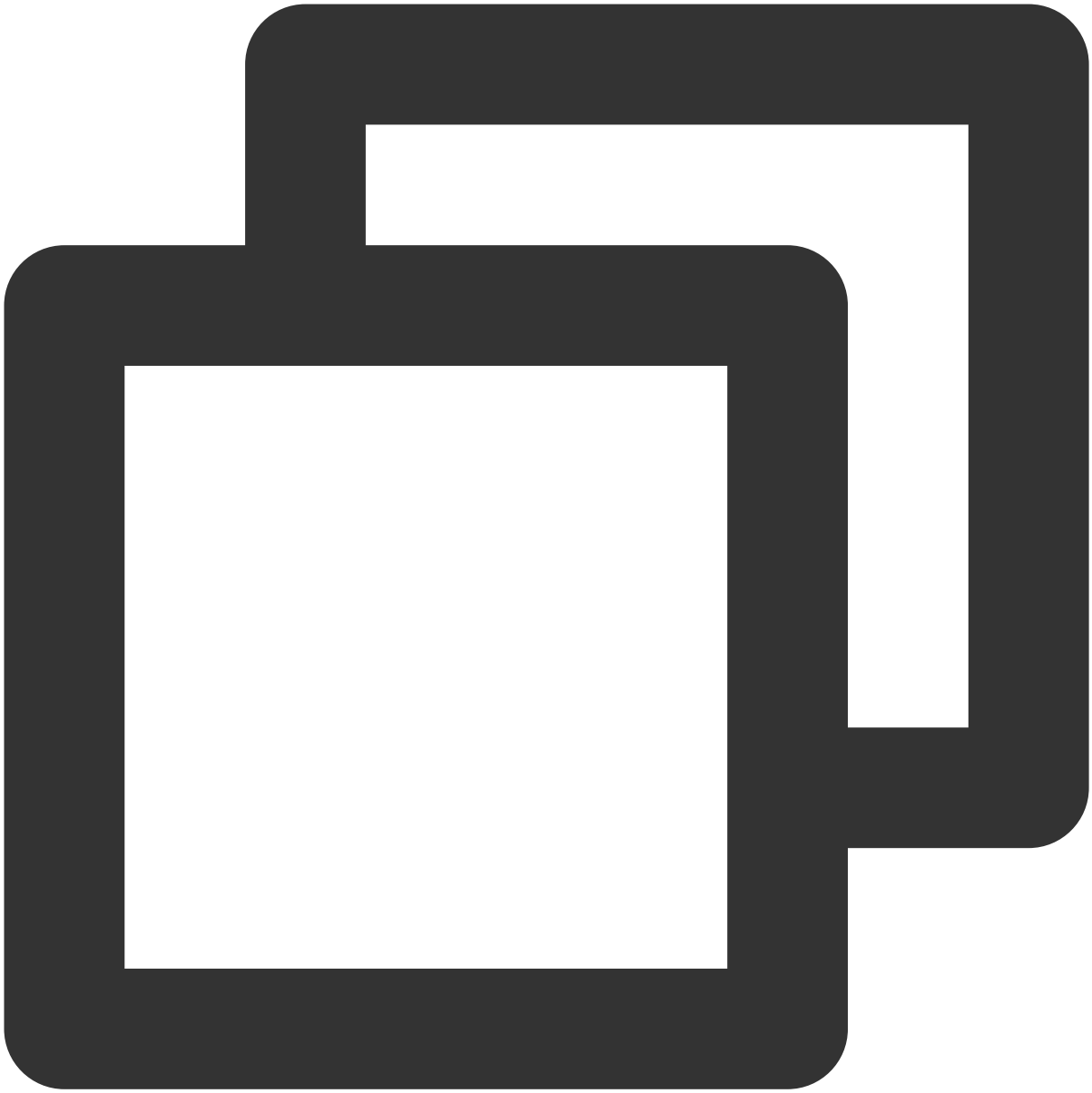
영상 통화 시작 전에 카메라, 마이크 등의 장치를 먼저 테스트 해볼 것을 권장합니다. 그렇지 않으면 사용자가 실제로 통화를 진행하는 동안 장치 문제를 발견하기 어렵습니다.

기능이 지원되는 플랫폼

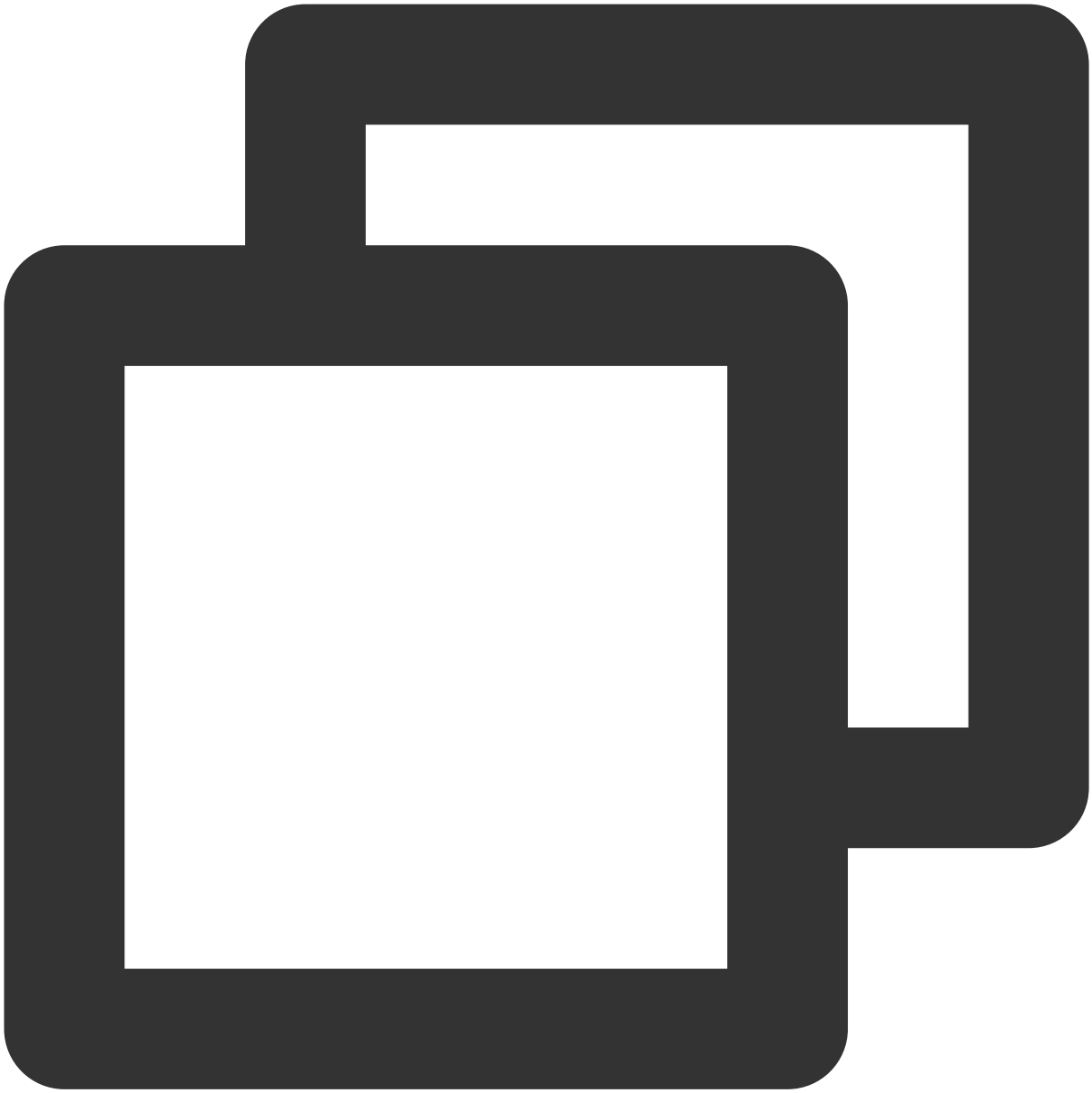
iOS	Android	Mac OS	Windows	Electron	web
×	×	✓	✓	✓	×

카메라 테스트

TRTCCloud의 'startCameraDeviceTestInView' 인터페이스를 사용하여 카메라 테스트를 진행합니다. 테스트 과정 중 'setCurrentCameraDevice' 함수를 호출하여 카메라를 전환할 수 있습니다.
Mac플랫폼
Windows플랫폼(C++)
Windows플랫폼(C#)

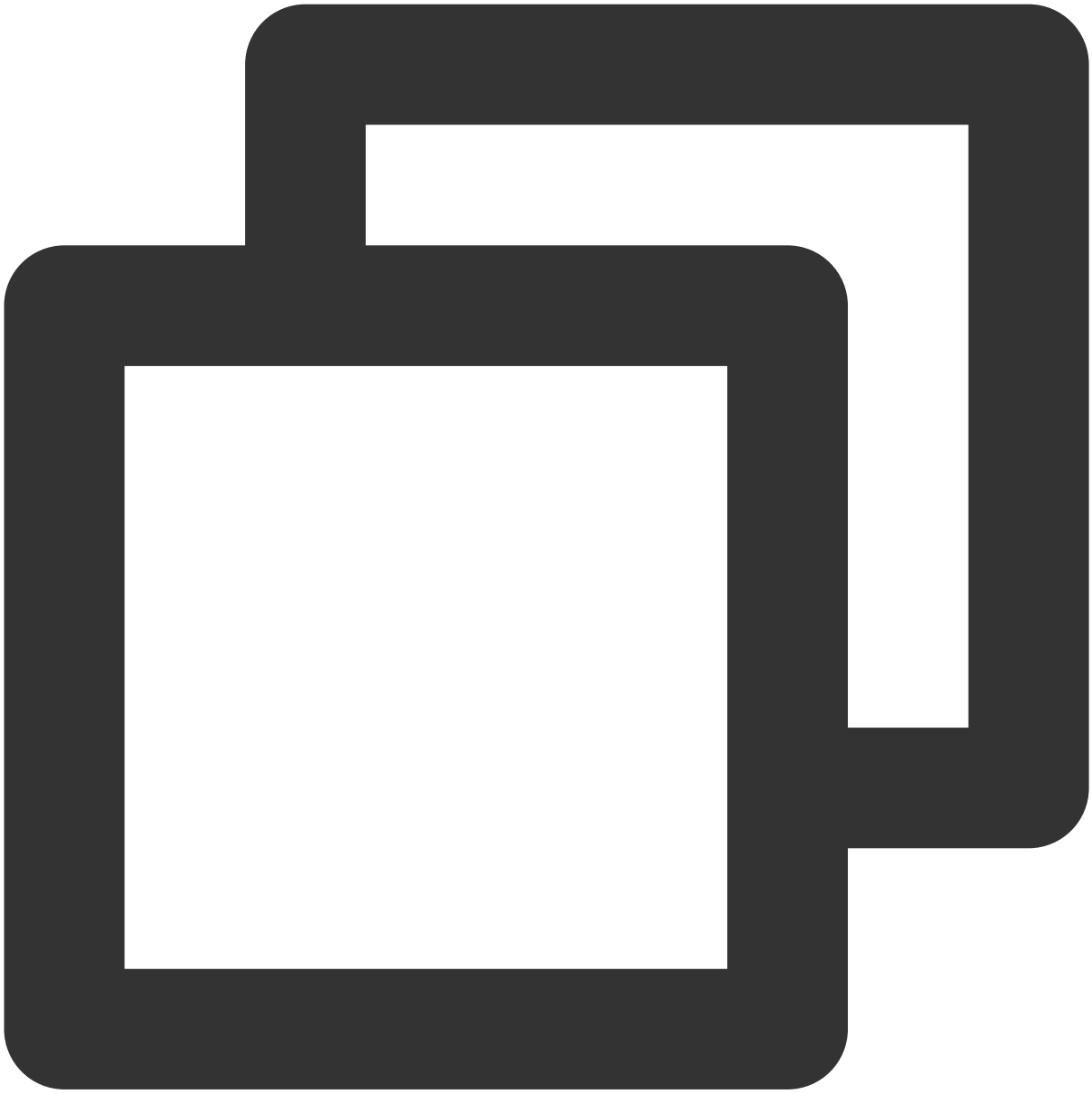


```
// 카메라 테스트 인터페이스 표시(카메라 미리보기, 카메라 전환 지원)
- (IBAction)startCameraTest:(id)sender {
    // 카메라 테스트 시작, cameraPreview는 macOS의 NSView 또는 iOS 플랫폼의 UIView
    [self.trtcCloud startCameraDeviceTestInView:self.cameraPreview];
}

//카메라 테스트 인터페이스 닫기
- (void)windowWillClose:(NSNotification *)notification{
    // 카메라 테스트 종료
    [self.trtcCloud stopCameraDeviceTest];
}
```



```
// 카메라 테스트 실행. 렌더링할 비디오의 컨트롤 핸들을 전달합니다.
void TRTCMainViewController::startTestCameraDevice(HWND hwnd)
{
     trtcCloud->startCameraDeviceTest(hwnd);
}

//  카메라 테스트 종료
void TRTCMainViewController::stopTestCameraDevice()
{
```

```
    trtcCloud->stopCameraDeviceTest();
}
```



```
// 카메라 테스트 실행. 렌더링할 비디오의 컨트롤러 전달
private void startTestCameraDevice(Intptr hwnd)
{
    mTRTCCloud.startCameraDeviceTest(hwnd);
}

// 카메라 테스트 종료
```

```
private void stopTestCameraDevice()
{
     mTRTCCloud.stopCameraDeviceTest();
}
```

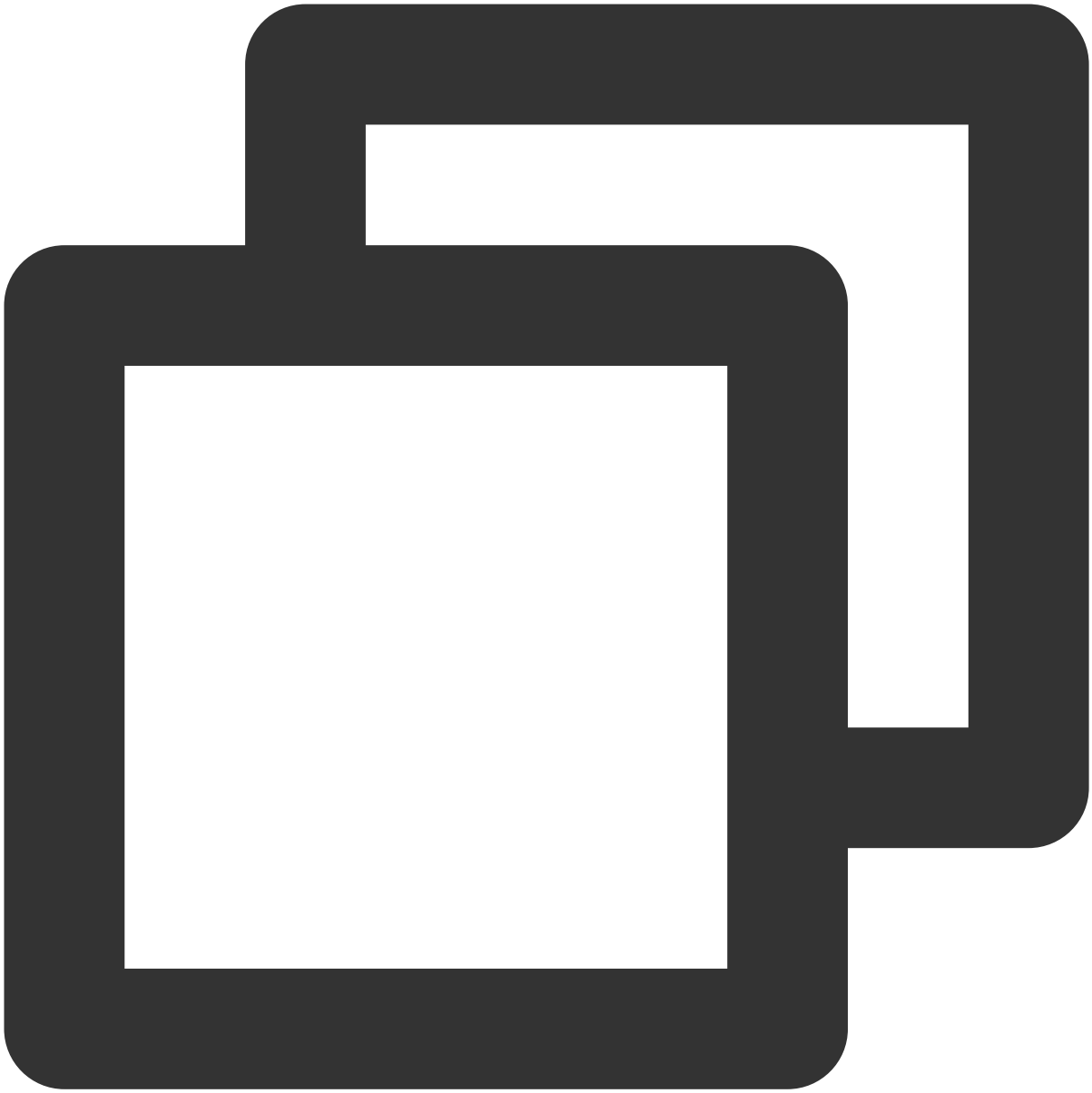
마이크 테스트

TRTCCloud의 `startMicDeviceTest` 함수를 이용하여 마이크 볼륨을 테스트할 수 있으며, 콜백 함수로 실시간 마이크 볼륨 값을 반환할 수 있습니다.

Mac플랫폼

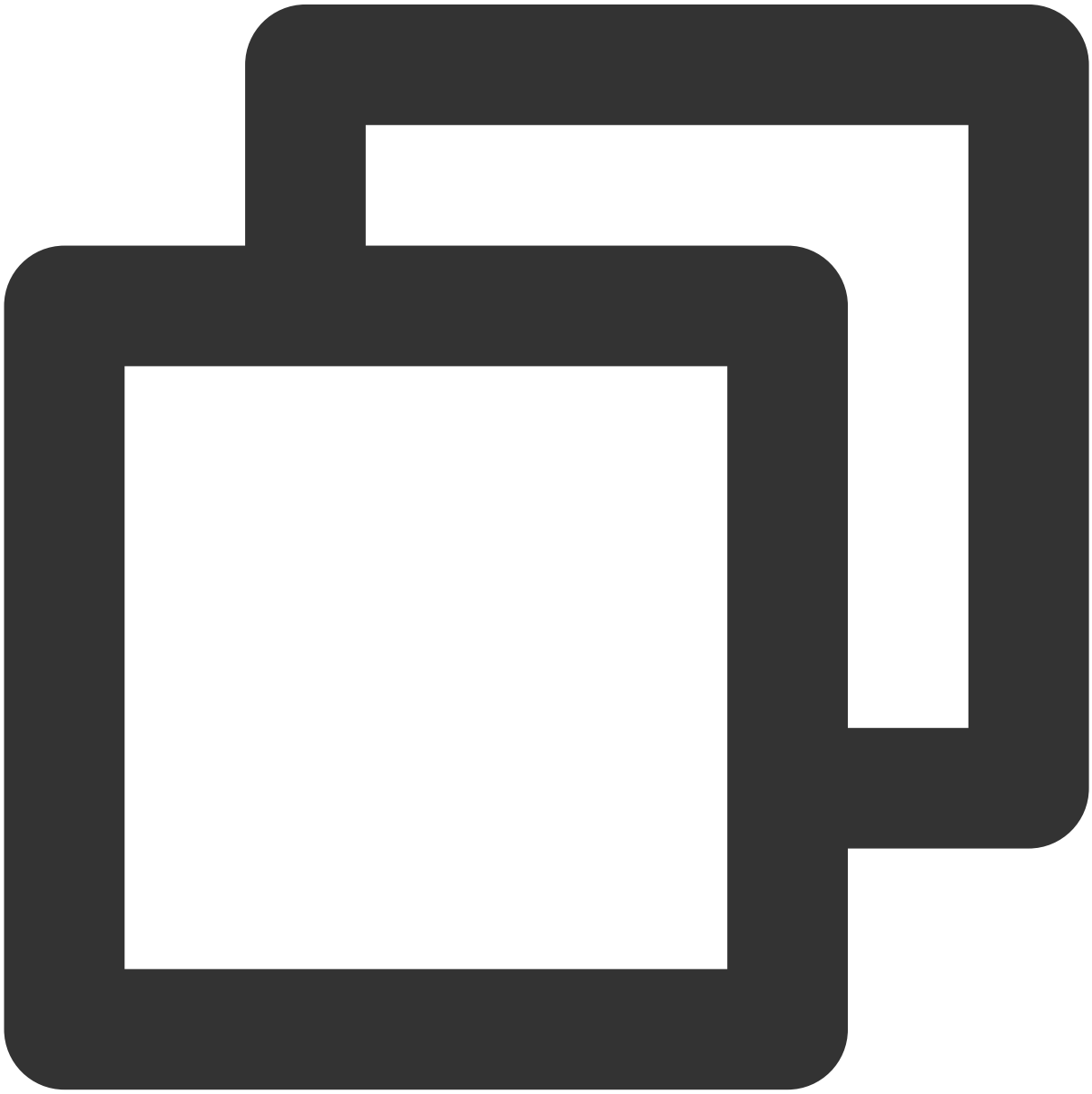
Windows플랫폼(C++)

Windows플랫폼(C#)

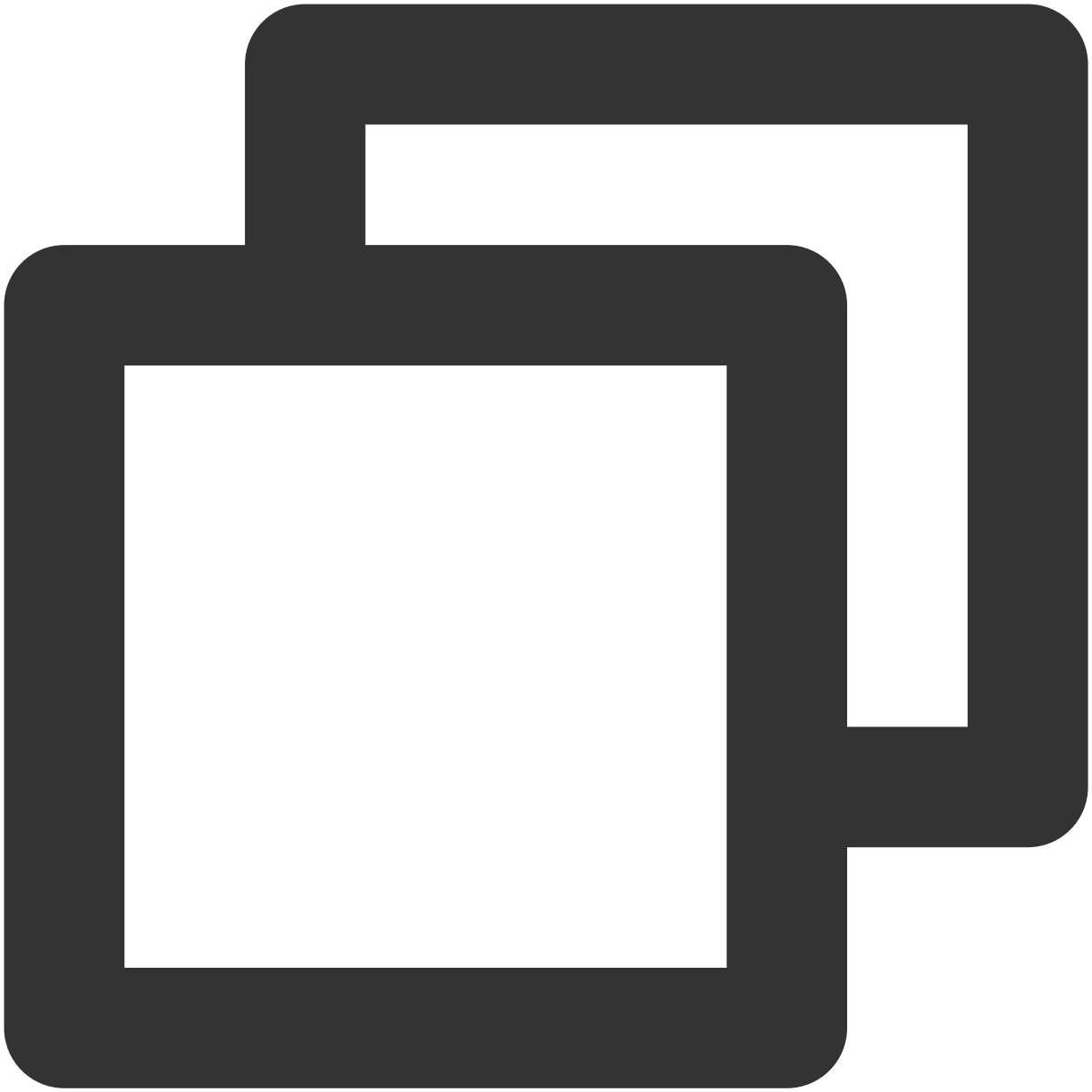


```
// 마이크 테스트 예시 코드
-(IBAction)micTest:(id)sender {
  NSButton *btn = (NSButton *)sender;
  if (btn.state == 1) {
          //마이크 테스트 시작
      __weak __typeof(self) wself = self;
      [self.trtcCloud startMicDeviceTest:500  testEcho:^(NSInteger volume) {
          dispatch_async(dispatch_get_main_queue(), ^{
                          // 마이크 볼륨 프로그레스 바 새로고침
              [wself _updateInputVolume:volume];
          });
```

```
        }];
        btn.title = @'테스트 종료';
    }
    else{
            //마이크 테스트 종료
        [self.trtcCloud stopMicDeviceTest];
        [self _updateInputVolume:0];
        btn.title = @'테스트 시작';
    }
}
```



```
// 마이크 테스트 예시 코드
void TRTCMainViewController::startTestMicDevice()
{
    // 볼륨 콜백 빈도 설정. 본 예시 코드에서는 500ms 마다 한 번씩 콜백하며, onTestMicVolume
    uint32_t interval = 500;
    // 마이크 테스트 시작
    trtcCloud->startMicDeviceTest(interval);
}

// 마이크 테스트 종료
void TRTCMainViewController::stopTestMicDevice()
{
     trtcCloud->stopMicDeviceTest();
}
```



```
// 마이크 테스트 예시 코드
private void startTestMicDevice()
{
    // 볼륨 콜백 빈도 설정. 본 예시 코드에서는 500ms 마다 한 번씩 콜백하며, onTestMicVolume
    uint interval = 500;
    // 마이크 테스트 시작
    mTRTCCloud.startMicDeviceTest(interval);
}

// 마이크 테스트 종료
private void stopTestMicDevice()
```

```
{
      mTRTCCloud.stopMicDeviceTest();
}
```

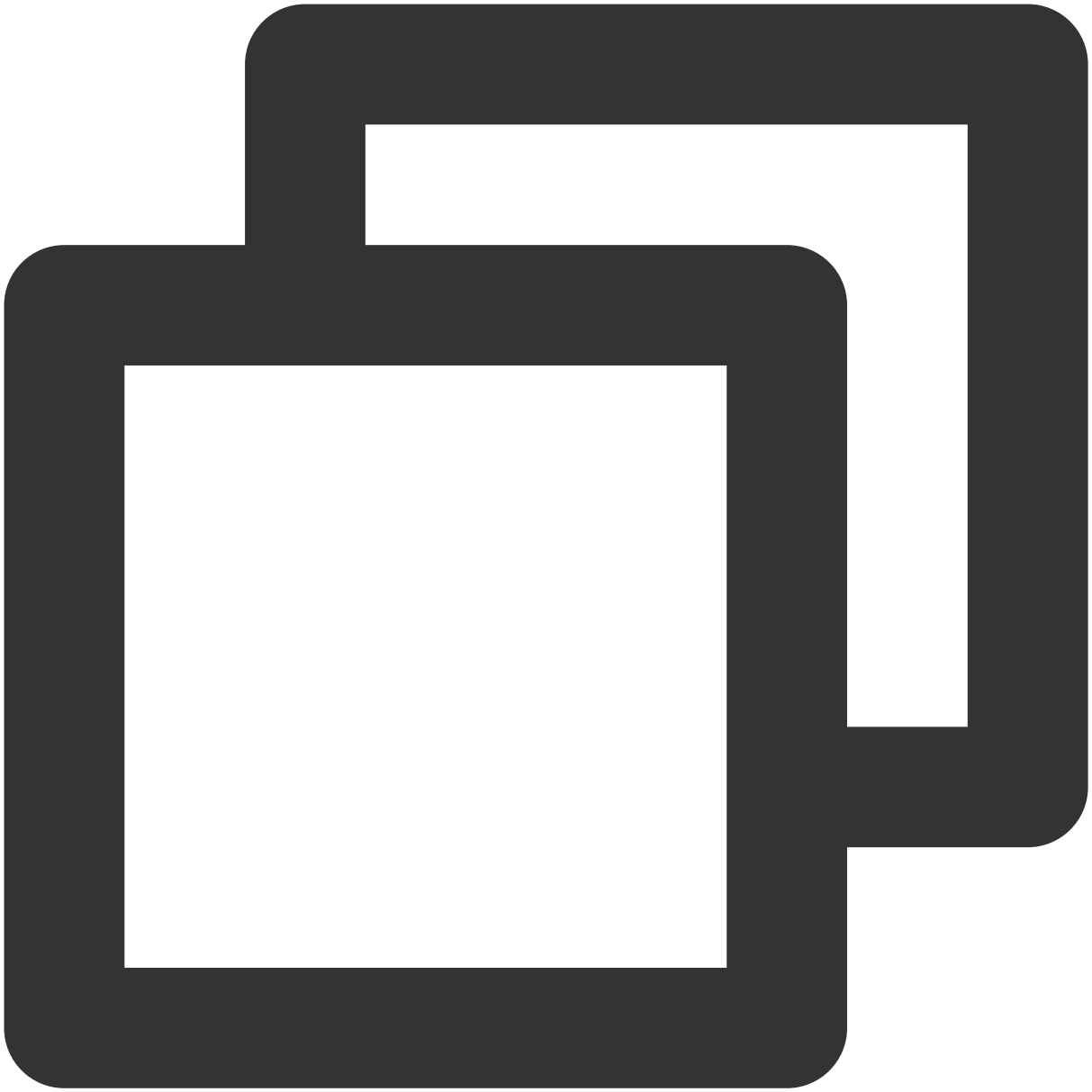
스피커 테스트

TRTCCloud의 `startSpeakerDeviceTest` 함수를 사용해 일부 기본 설정된 mp3 오디오를 재생하여 스피커가 정상 작동되는지 테스트합니다.

Mac플랫폼

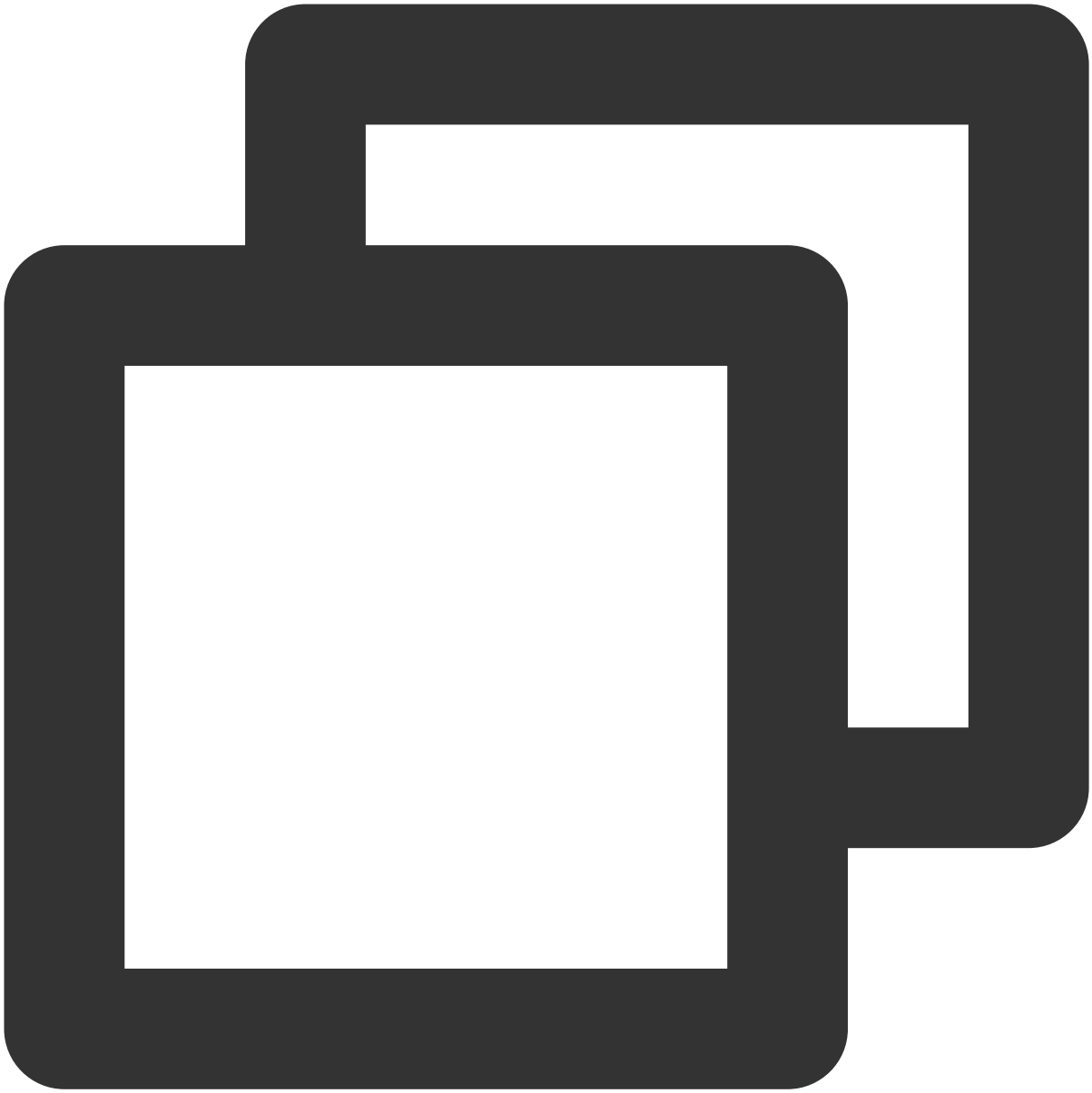
Windows플랫폼(C++)

Windows플랫폼(C#)



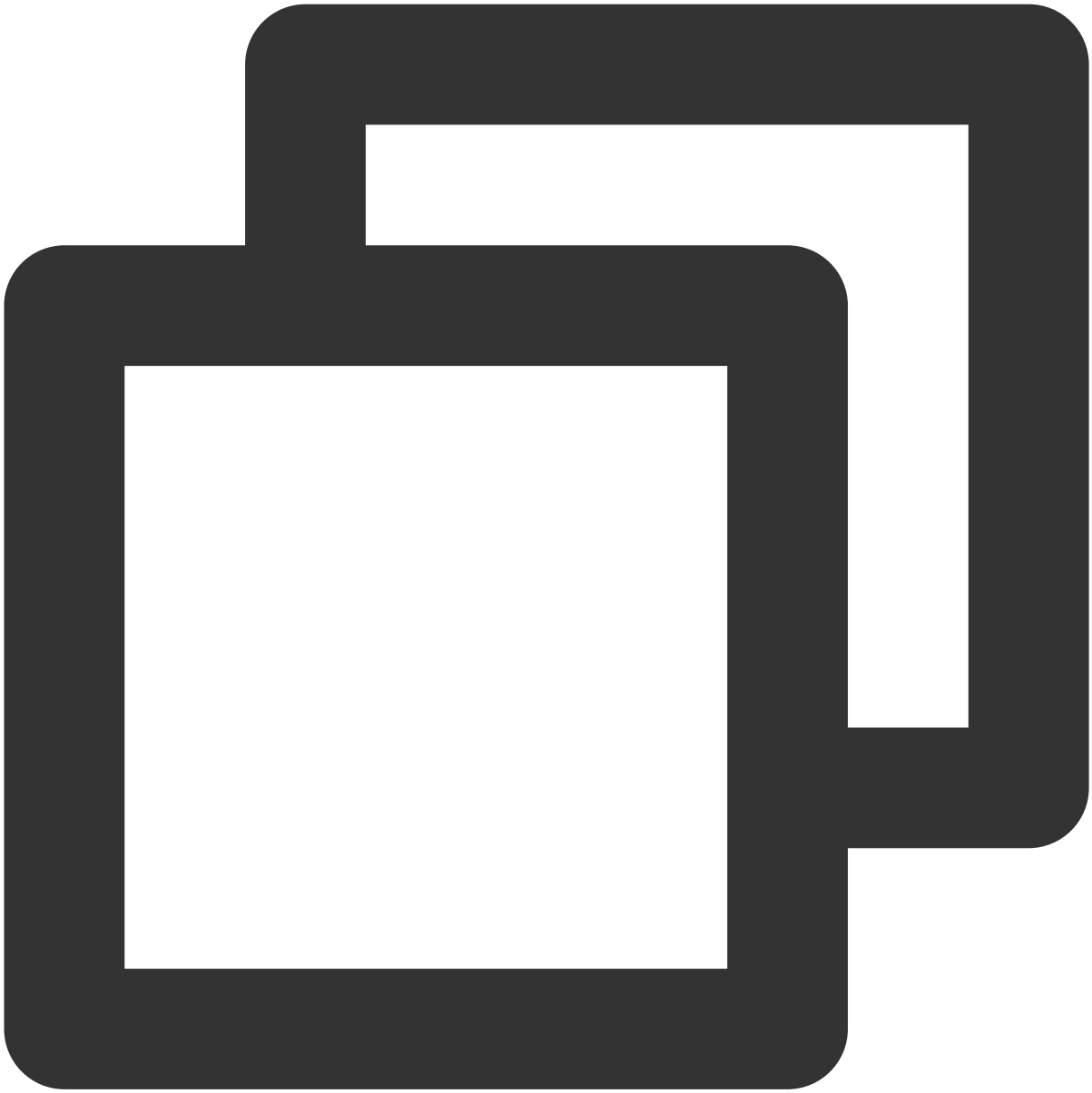
```
// 스피커 테스트 예시 코드
// NSButton의 클릭 이벤트를 예시로, xib에서 Button이 On 및 Off될 때 제목이 '테스트 종료', '
- (IBAction)speakerTest:(NSButton *)btn {
    NSString *path = [[NSBundle mainBundle] pathForResource:@'test-32000-mono' ofTy
    if (btn.state == NSControlStateValueOn) {
        // '테스트 시작' 클릭
        __weak __typeof(self) wself = self;
        [self.trtcEngine startSpeakerDeviceTest:path onVolumeChanged:^(NSInteger vo
            // 다음의 UI 관련 작업은 main queue로 전환해 실행해야 합니다.
            dispatch_async(dispatch_get_main_queue(), ^{
                // 여기에서 _updateOutputVolume은 업데이트 페이지의 스피커 볼륨 지표입니다.
```

```
                [wself _updateOutputVolume:volume];
                if (playFinished) {
                    // 재생 완료 시 버튼 상태 '테스트 시작' 상태로 설정
                    sender.state = NSControlStateValueOff;
                }
            });
        }];
    } else {
        // '테스트 종료' 클릭
        [self.trtcEngine stopSpeakerDeviceTest];
        [self _updateOutputVolume:0];
    }
}

// 스피커 볼륨 측정기 업데이트
- (void)_updateOutputVolume:(NSInteger)volume {
    // speakerVolumeMeter는 NSLevelIndicator임
    self.speakerVolumeMeter.doubleValue = volume / 255.0 * 10;
}
```



```
// 스피커 테스트 예시 코드
void TRTCMainViewController::startTestSpeakerDevice(std::string testAudioFilePath)
{
    // testAudioFilePath: 오디오 파일의 절대 경로. 경로 문자열은 UTF-8 인코딩 포맷을 사용하
    // onTestSpeakerVolume에서 인터페이스를 콜백하여 스피커 테스트 볼륨 값을 모니터링합니다.
    trtcCloud->startSpeakerDeviceTest(testAudioFilePath.c_str());
}

// 스피커 테스트 종료
void TRTCMainViewController::stopTestSpeakerDevice() {
    trtcCloud->stopSpeakerDeviceTest();
```

```
}
```



```
// 스피커 테스트 예시 코드
private void startTestSpeakerDevice(string testAudioFilePath)
{
    // testAudioFilePath: 오디오 파일의 절대 경로. UTF-8 형식의 문자열이어야 합니다. 지원되
    // onTestSpeakerVolume 콜백 인터페이스에서 스피커 테스트 볼륨 값 수신.
    mTRTCCloud.startSpeakerDeviceTest(testAudioFilePath);
}
```

```
// 스피커 테스트 종료
private void stopTestSpeakerDevice() {
    mTRTCCloud.stopSpeakerDeviceTest();
}
```

Web

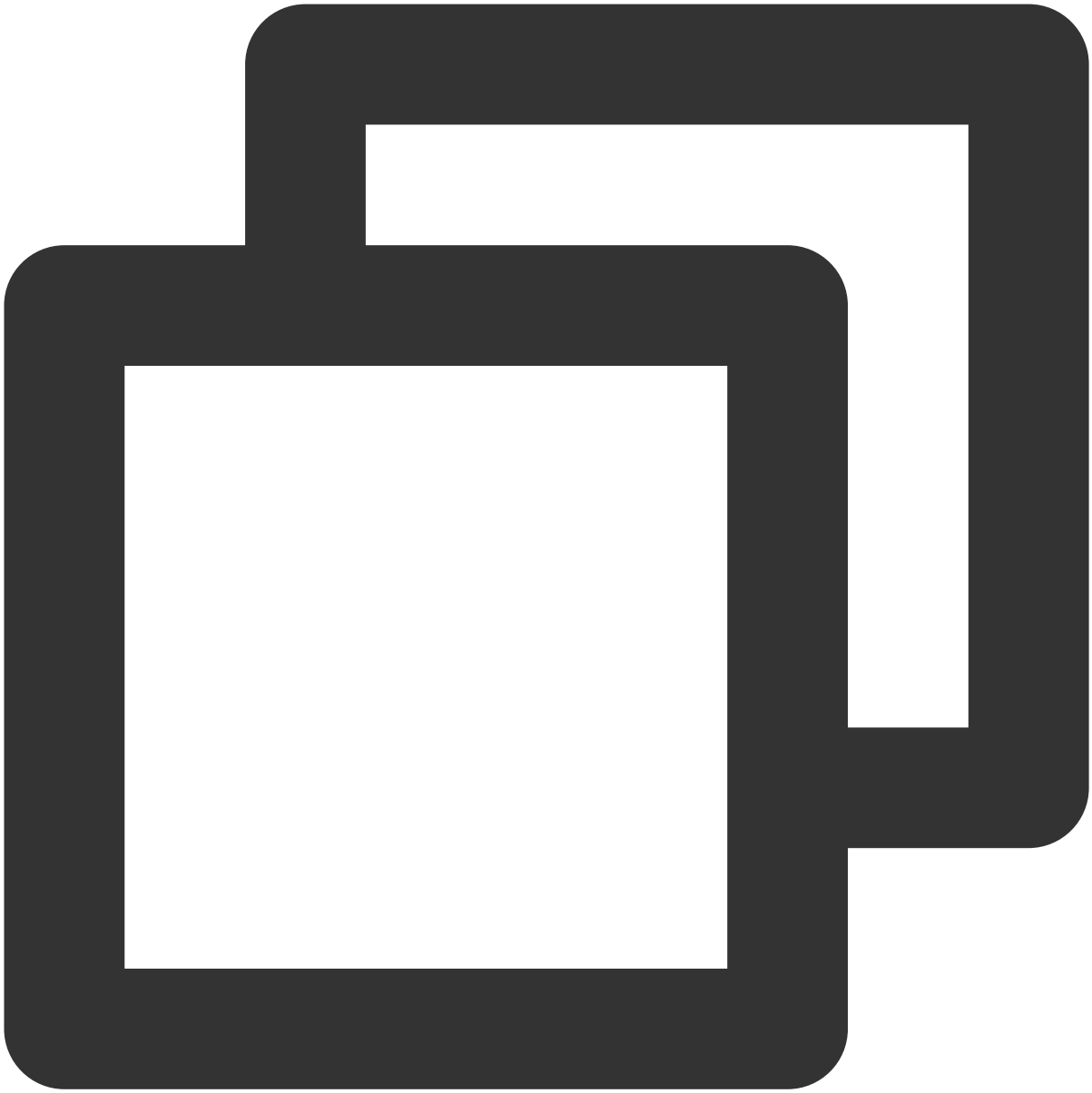
최종 업데이트 날짜: : 2024-07-24 16:43:38

개요

사용자는 통화 중 장치 문제를 감지하기 어렵기 때문에 브라우저를 점검하고 카메라, 마이크 등의 장치를 테스트한 후 영상 통화를 시작하는 것을 권장합니다.

브라우저 점검

SDK의 통신 기능을 호출하기 전에 {@link TRTC.checkSystemRequirements checkSystemRequirements()} API를 사용하여 SDK가 사용자의 브라우저를 지원하는지 점검하는 것이 좋습니다. 브라우저가 지원되지 않는 경우 사용자의 장치 유형에 따라 지원되는 브라우저를 사용하도록 권장하십시오.



```
TRTC.checkSystemRequirements().then(checkResult => {
  if (checkResult.result) {
    // 방 입장 지원 여부 점검
    if (checkResult.isH264DecodeSupported) {
        // 풀 스트림 지원 여부 점검
    }
    if (checkResult.isH264EncodeSupported) {
        // 스트림 푸시가 지원 여부 점검
    }
  }
})
```

⚠ 사용자의 현재 브라우저가 SDK에서 지원되지만 `TRTC.checkSystemRequirements` 에서 반환된 점검 결과가 false인 경우 다음 이유 중 하나 때문일 수 있습니다.
사례1: 링크가 다음 3가지 조건 중 하나를 충족하는 경우
localhost 도메인(Firefox는 localhost 및 로컬 ip에 대한 액세스 지원)
HTTPS가 활성화된 도메인
file:/// 프로토콜을 통해 열린 로컬 파일
사례2: Firefox 브라우저가 설치된 후 H264 코덱을 동적으로 로드해야 합니다. 따라서 검사 결과는 일시적으로 false가 됩니다. 잠시 기다렸다가 다시 시도하거나 다른 권장 브라우저를 사용하여 링크를 여십시오.

다른 브라우저에 대한 알려진 사용 제한

Firefox
Firefox는 30 fps의 프레임 레이트만 지원합니다. 프레임 레이트를 설정해야 하는 경우 SDK에서 지원하는 다른 브라우저를 사용하십시오.
QQ 브라우저
일반 카메라와 마이크가 있는 특정 Windows 장치의 localhost 환경에서 localStream.initialize()를 호출하면 NotFoundError 오류가 발생할 수 있습니다.

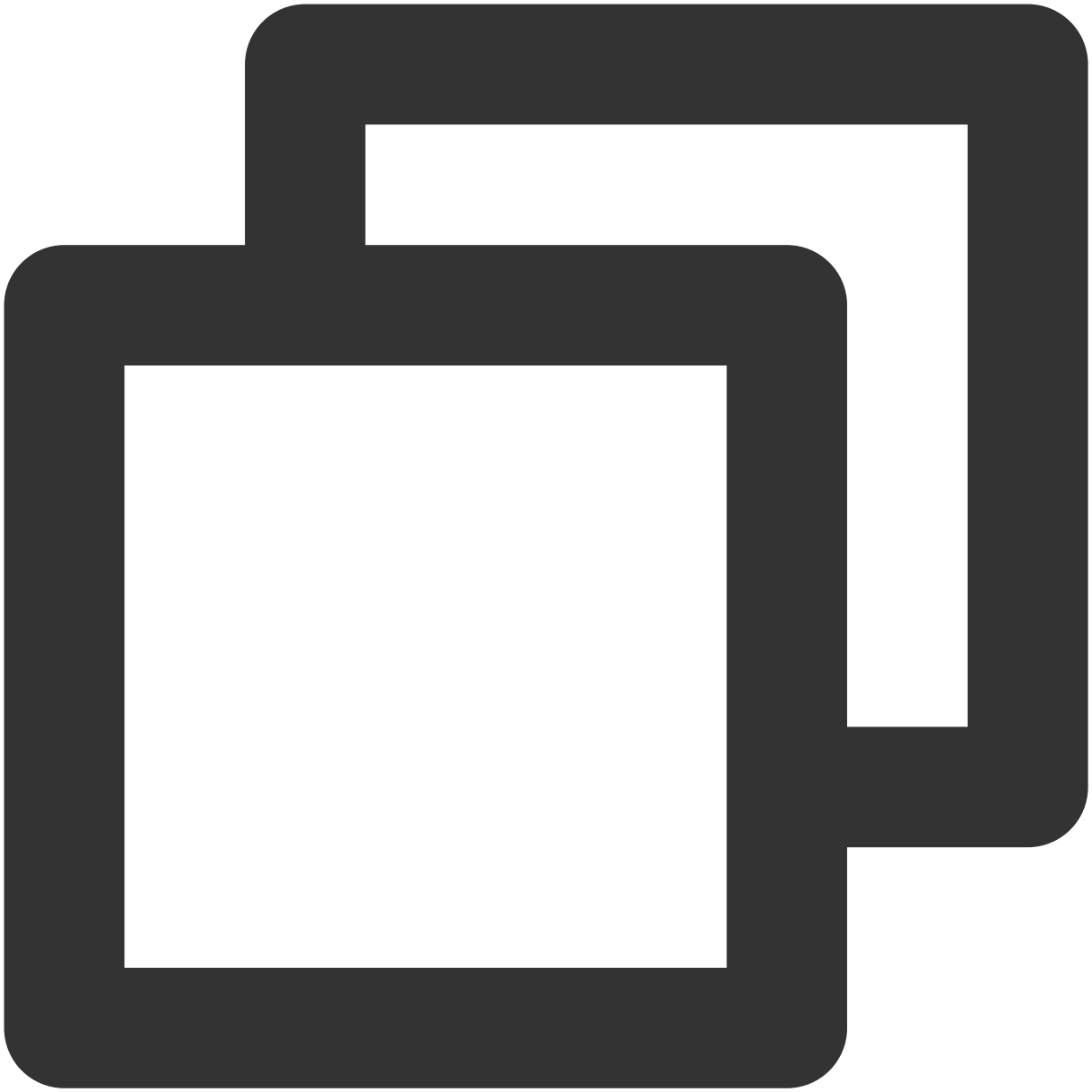
멀티미디어 장치 테스트

사용자가 TRTC SDK로 좋은 사용자 경험을 할 수 있도록 사용자가 TRTC 방에 입장하기 전에 사용자의 장치 및 네트워크 상태를 점검하고 문제 해결 제안을 제공하는 것이 좋습니다.
다음 메소드를 참고하여 장치 및 네트워크 점검 기능을 통합할 수 있습니다.
rtc-detect의 JS 라이브러리
장치 점검용 React 컴포넌트
TRTC 기능 점검 페이지

rtc-detect 라이브러리

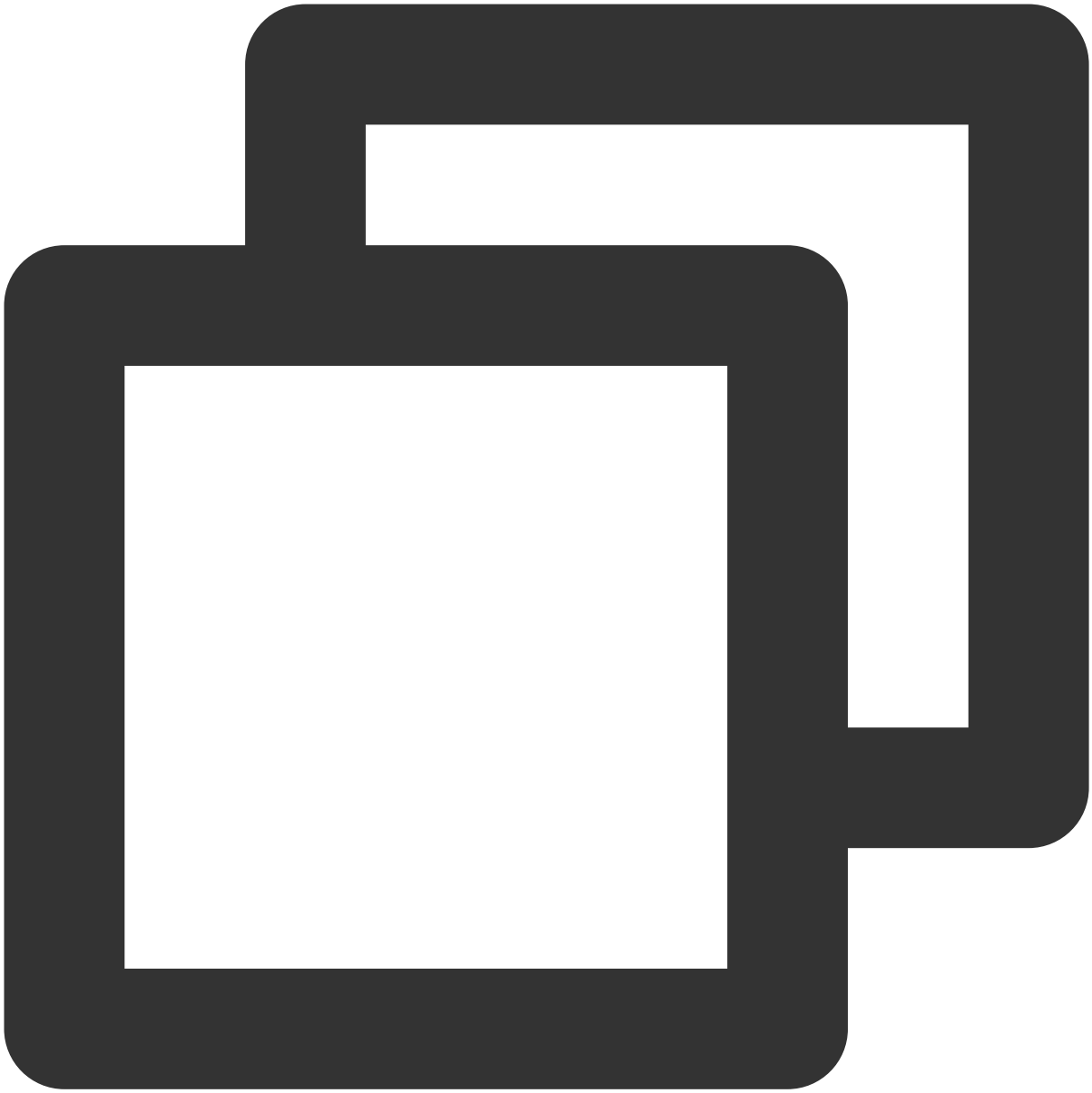
rtc-detect에서 TRTC SDK에 대한 현재 환경의 지원을 점검하고 현재 환경의 세부 정보를 볼 수 있습니다.

설치



```
npm install rtc-detect
```

사용 방법

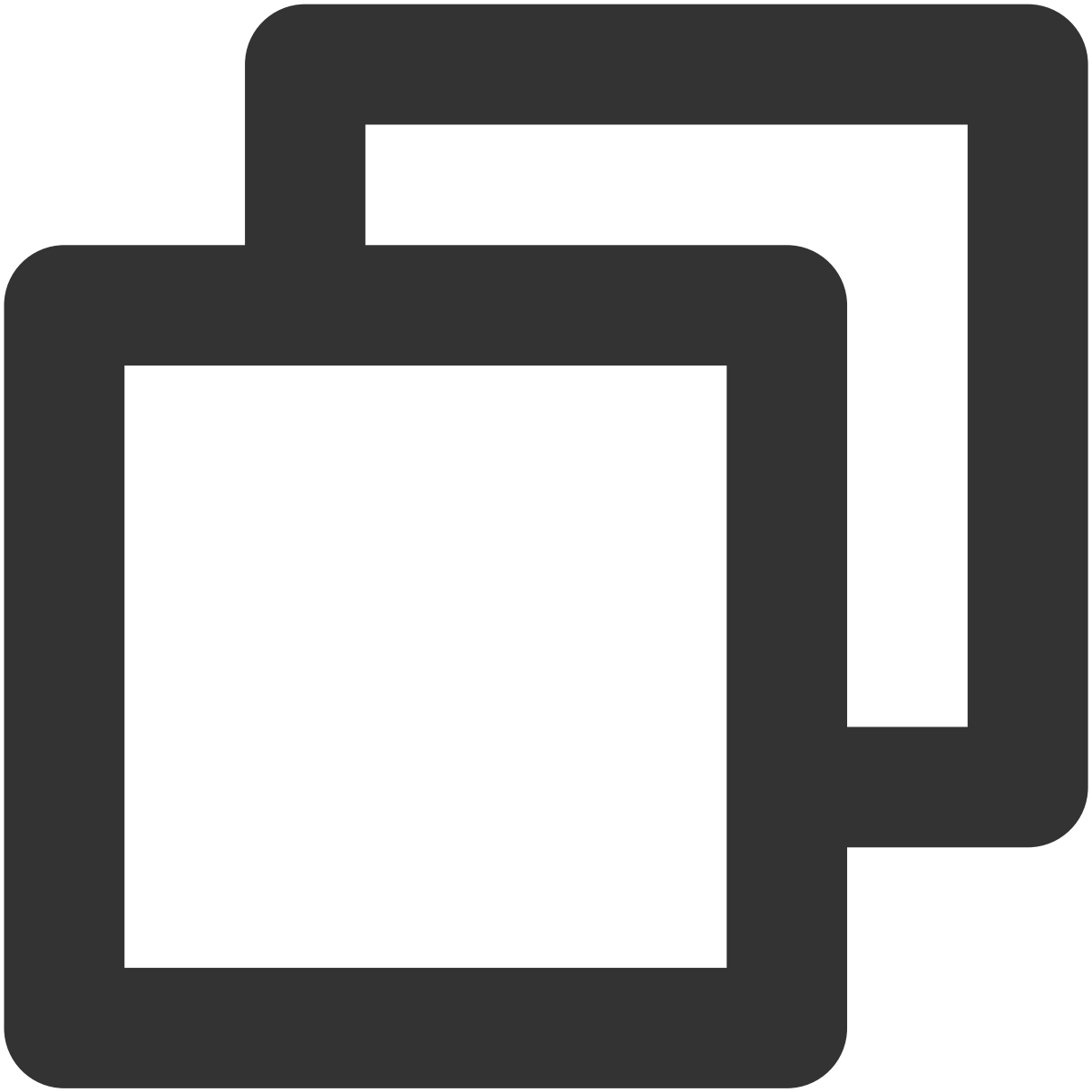


```
import RTCDetect from 'rtc-detect';
// 감지 모듈 초기화
const detect = new RTCDetect();
// 현재 환경의 감지 결과를 가져옴
const result = await detect.getReportAsync();
// result에는 현재 환경 시스템 정보, API 지원, 코덱 지원, 장치 정보가 포함됨
console.log('result is: ' + result);
```

API

(async) isTRTCSupported()

이 API는 현재 환경이 TRTC를 지원하는지 점검하는 데 사용됩니다.

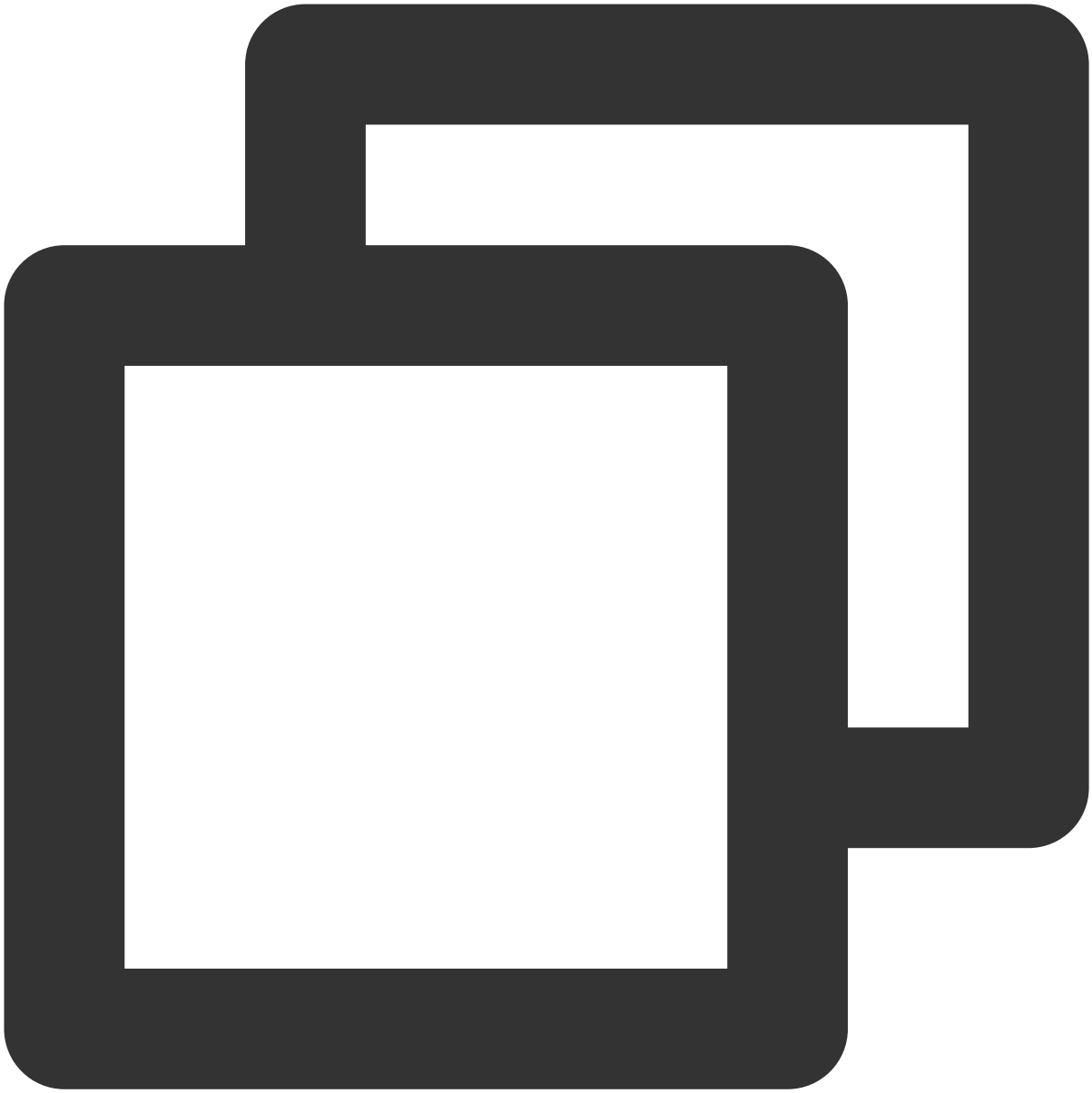


```
const detect = new RTCDetect();
const data = await detect.isTRTCSupported();

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

getSystem()

이 API는 현재 시스템 환경 매개변수를 가져오는 데 사용됩니다.

Item	Type	Description
UA	string	브라우저 ua
OS	string	현재 장치의 운영 체제
browser	object	{ name, version } 형식의 현재 브라우저 정보
displayResolution	object	{ width, height } 형식의 현재 해상도
getHardwareConcurrency	number	현재 장치의 CPU 코어 수



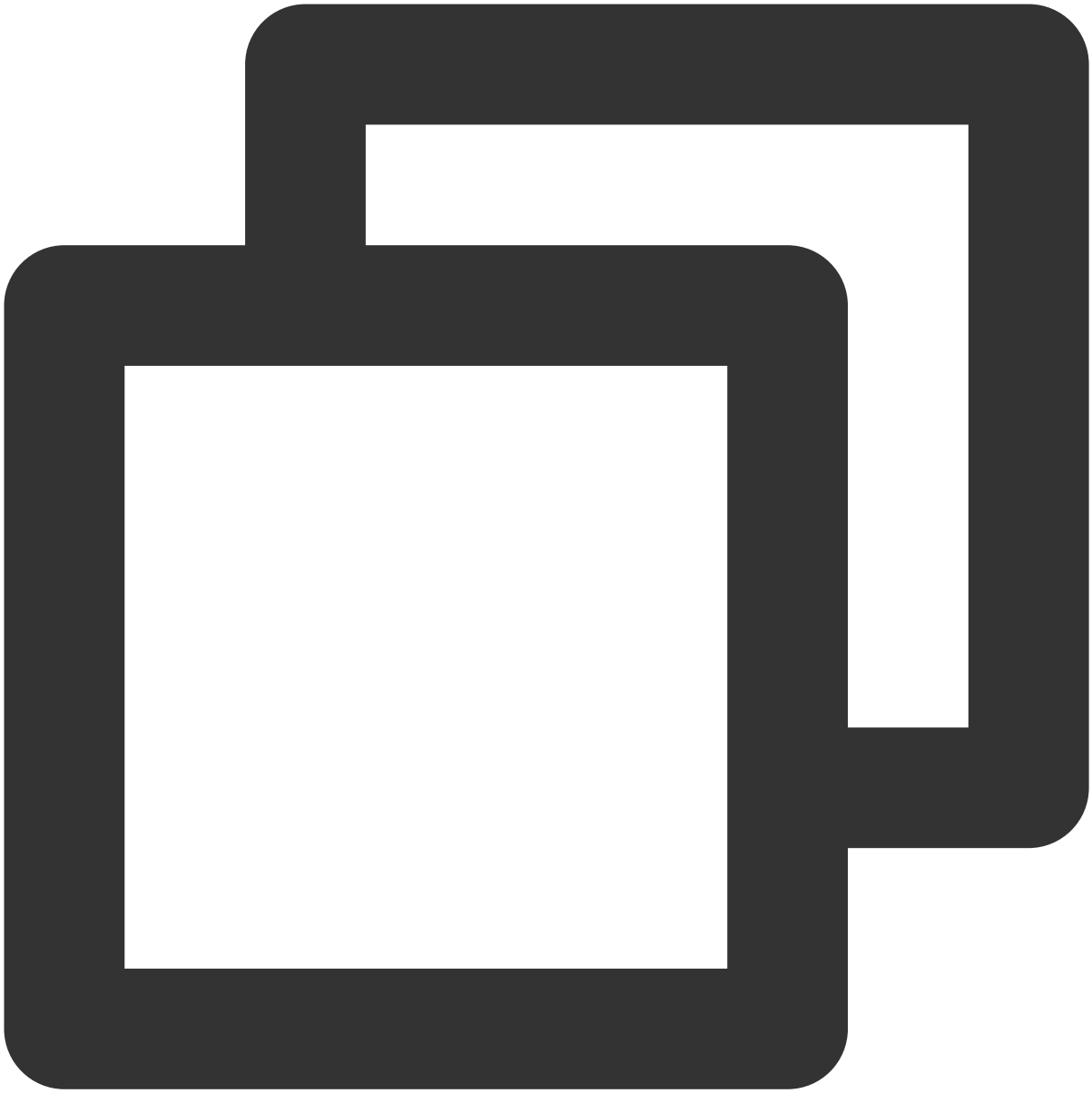
```
const detect = new RTCDetect();
const result = detect.getSystem();
```

getAPISupported()

이 API는 현재 환경의 API 지원을 얻는 데 사용됩니다.

Item	Type	Description
isUserMediaSupported	boolean	사용자 미디어 데이터 스트림 획득 가능 여부

isWebRTCSupported	boolean	WebRTC 지원 여부
isWebSocketSupported	boolean	WebSocket 지원 여부
isWebAudioSupported	boolean	WebAudio 지원 여부
isScreenCaptureAPISupported	boolean	화면 스트림 획득 가능 여부
isCanvasCapturingSupported	boolean	canvas에서 데이터 스트림 획득 가능 여부
isVideoCapturingSupported	boolean	video에서 데이터 스트림 획득 가능 여부
isRTPSenderReplaceTracksSupported	boolean	track이 대체될 때 peerConnection과의 재협상을 생략할 수 있는지 여부
isApplyConstraintsSupported	boolean	다시 getUserMedia를 호출하지 않고 카메라 해상도를 변경할 수 있는지 여부



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

(async) getDevicesAsync()

이 API는 현재 환경에서 사용 가능한 장치를 가져오는 데 사용됩니다.

Item	Type	Description
hasWebCamPermissions	boolean	사용자 카메라 데이터 획득 가능 여부

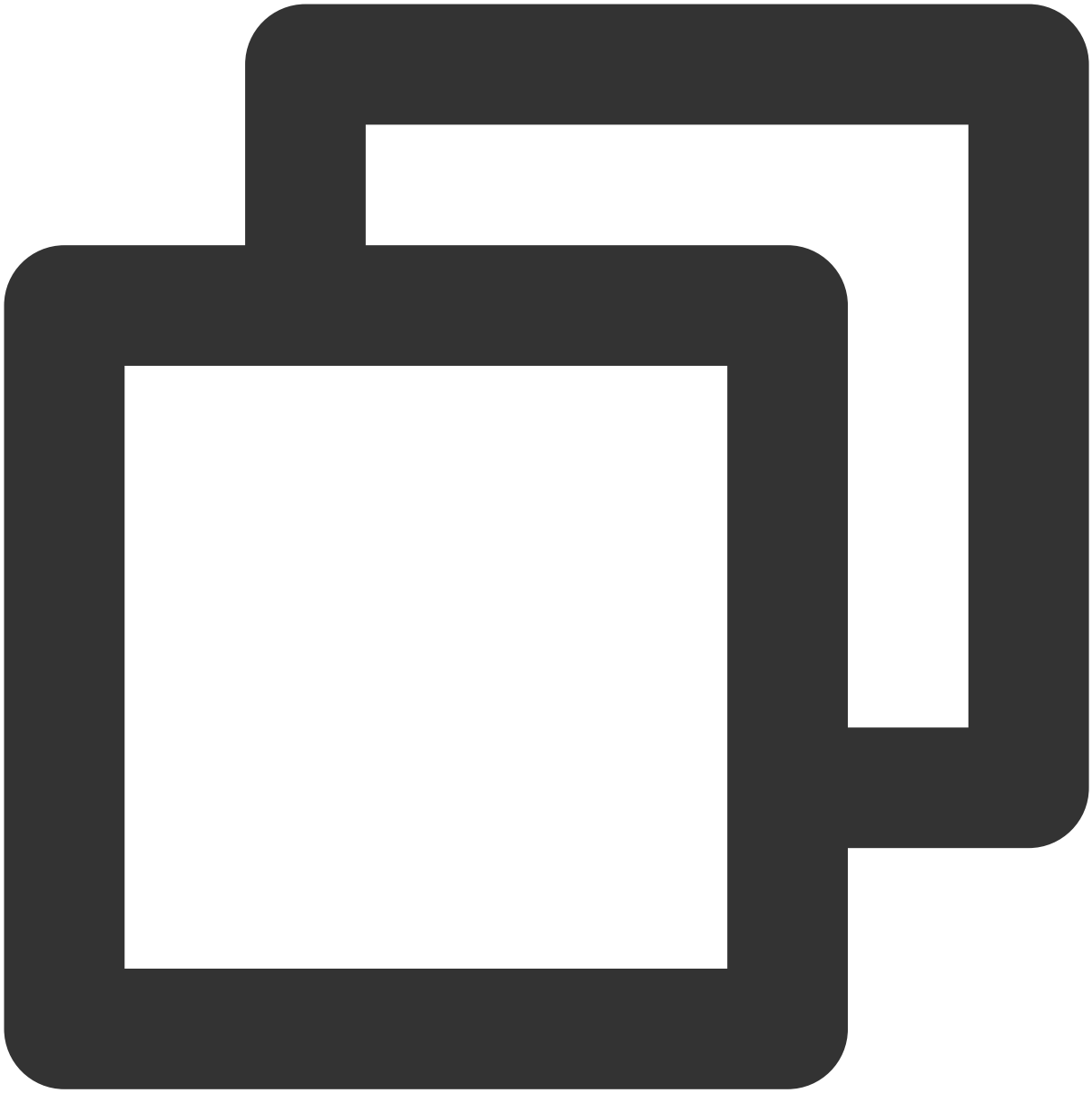
hasMicrophonePermission	boolean	사용자 마이크 데이터 획득 가능 여부
cameras	array	비디오 스트림에 대해 지원되는 해상도, 최대 너비, 최대 높이 및 최대 프레임 레이트(특정 브라우저만 해당)를 포함한 사용자 카메라 목록
microphones	array	사용자 마이크 목록
speakers	array	사용자 스피커 목록

CameraItem

Item	Type	Description
deviceId	string	장치 ID, 일반적으로 고유하며 장치를 식별하는 데 사용할 수 있음
groupId	string	그룹 ID, 두 장치가 동일한 물리적 장치에 속하는 경우 동일한 그룹 ID를 갖음
kind	string	카메라 장치 유형: 'videoinput'
label	string	장치를 설명하는 태그
resolution	object	카메라가 지원하는 최대 해상도 너비, 높이 및 프레임 레이트 {maxWidth: 1280, maxHeight: 720, maxFrameRate: 30}

DeviceItem

Item	Type	Description
deviceId	string	장치 ID, 일반적으로 고유하며 장치를 식별하는 데 사용할 수 있음
groupId	string	그룹 ID, 두 장치가 동일한 물리적 장치에 속하는 경우 동일한 그룹 ID를 갖음
kind	string	장치 유형, 예시: 'audioinput', 'audiooutput'
label	string	장치를 설명하는 태그



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

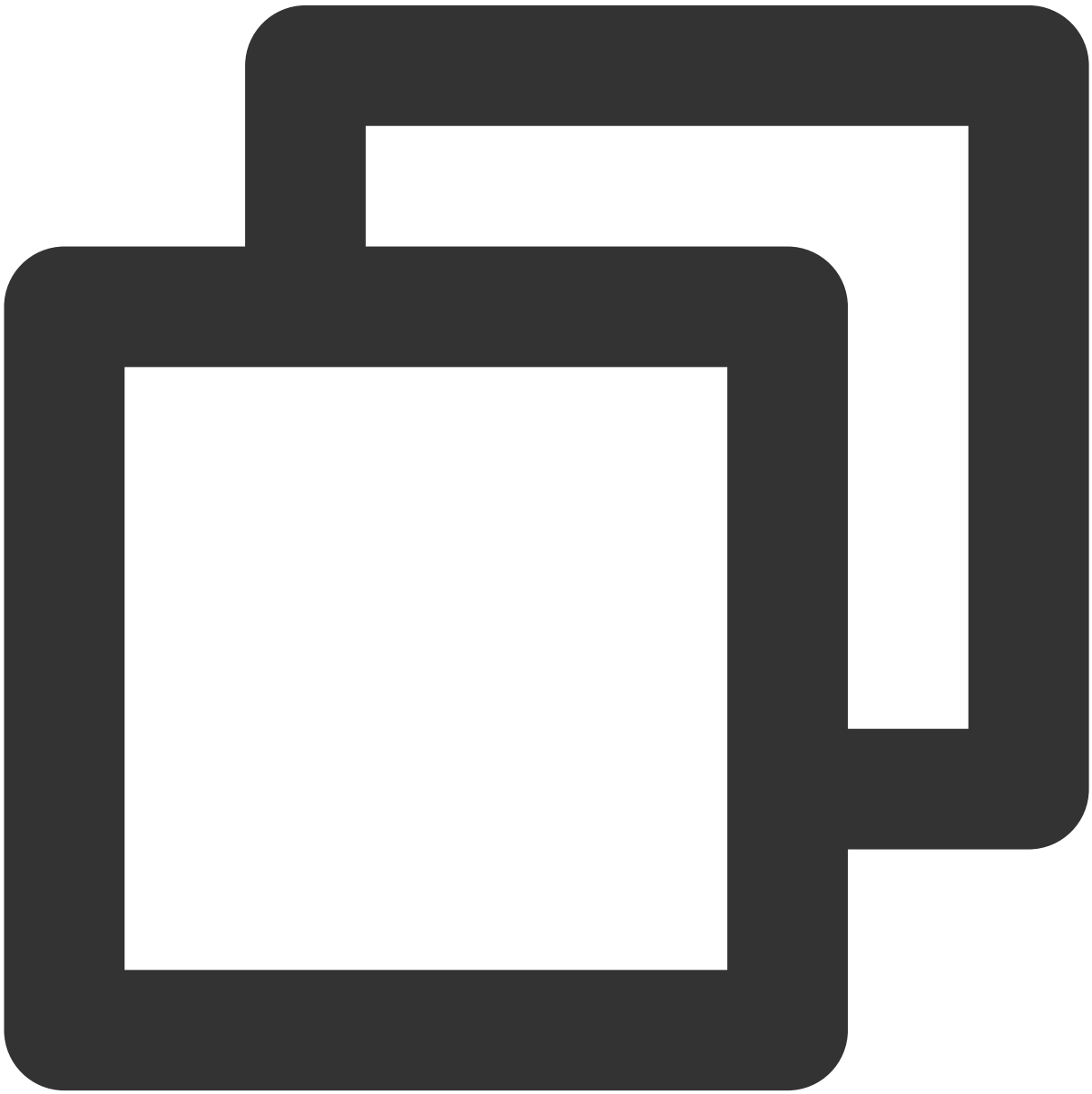
(async) getCodecAsync()

이 API는 현재 환경의 코덱 지원을 얻는 데 사용됩니다.

Item	Type	Description
isH264EncodeSupported	boolean	h264 인코딩 지원 여부

isH264DecodeSupported	boolean	h264 디코딩 지원 여부
isVp8EncodeSupported	boolean	vp8 인코딩 지원 여부
isVp8DecodeSupported	boolean	vp8 디코딩 지원 여부

인코딩이 지원되는 경우 오디오/비디오를 게시할 수 있습니다. 디코딩이 지원되는 경우 재생을 위해 오디오/비디오를 가져올 수 있음

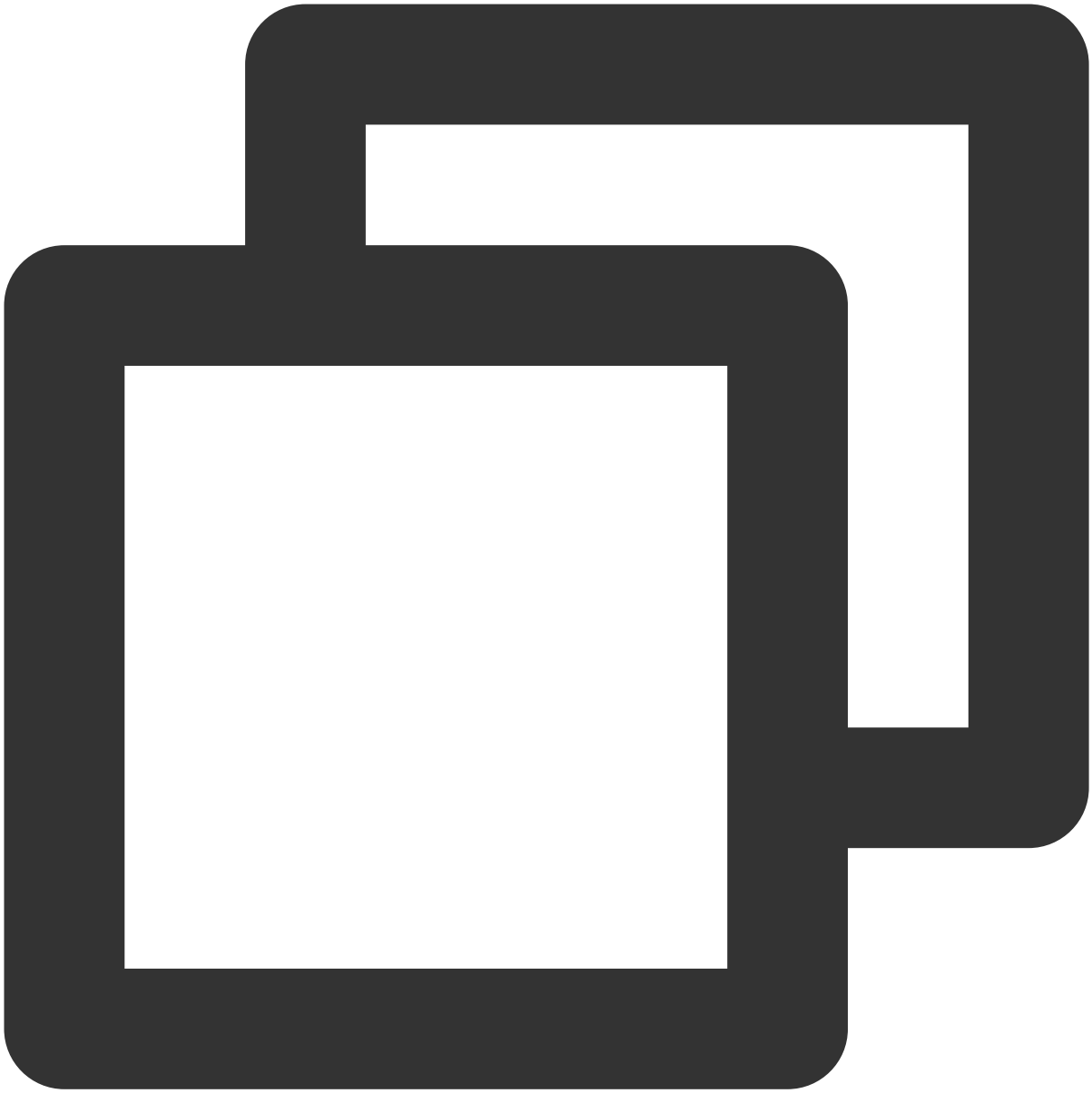


```
const detect = new RTCDetect();
const result = await detect.getCodecAsync();
```

(async) getReportAsync()

이 API는 현재 환경의 탐지 보고서를 가져오는 데 사용됩니다.

Item	Type	Description
system	object	getSystem()의 반환 값과 동일
APISupported	object	getAPISupported()의 반환 값과 동일
codecsSupported	object	getCodecAsync()의 반환 값과 동일
devices	object	getDevicesAsync()의 반환 값과 동일



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

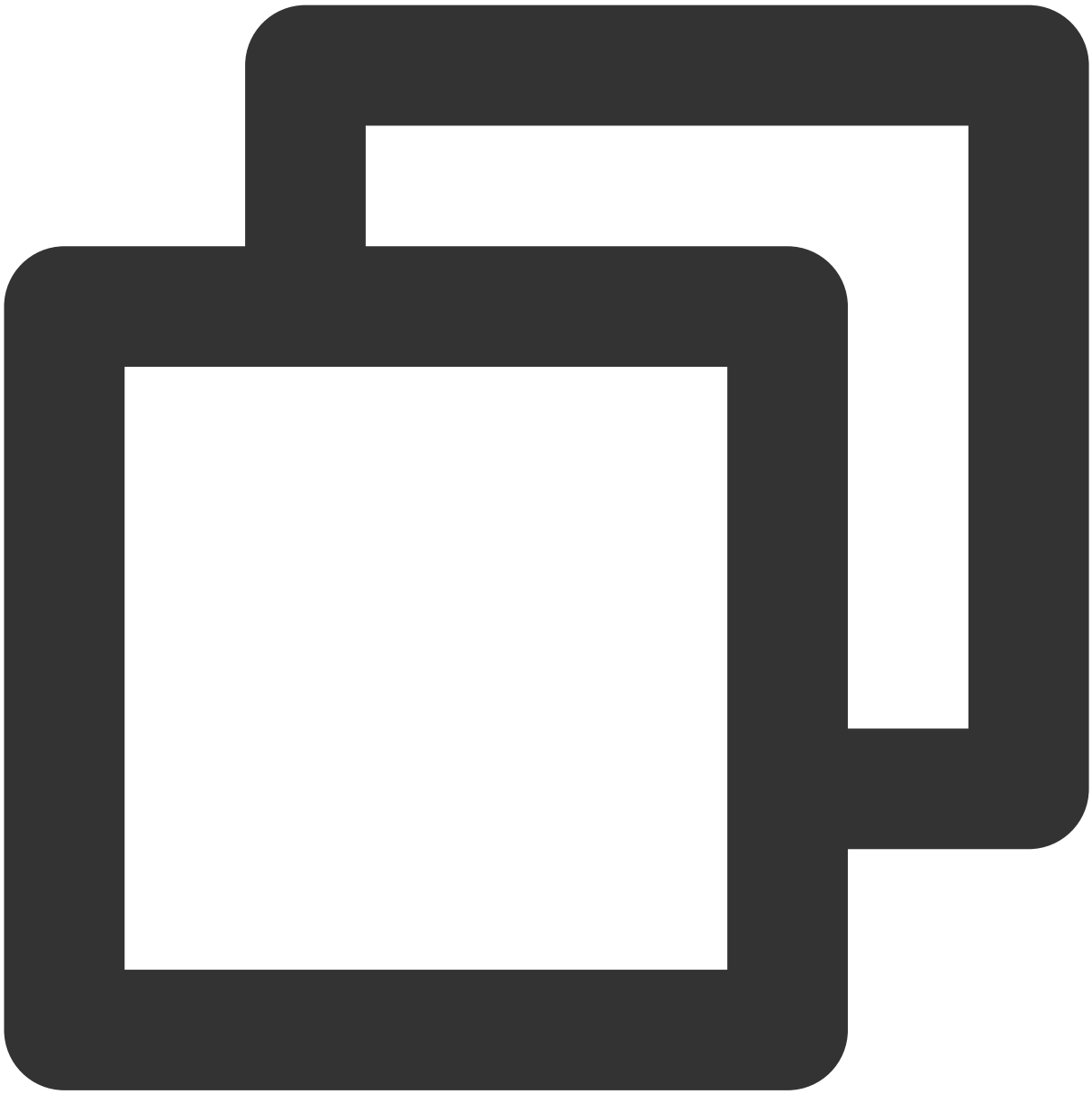
(async) isHardWareAccelerationEnabled()

이 API는 Chrome 브라우저에서 하드웨어 가속이 활성화되어 있는지 점검하는 데 사용됩니다.

주의사항 :

이 API의 구현은 기본 WebRTC API에 따라 다릅니다. isTRTCSupported를 호출한 후 점검을 위해 이 API를 호출하는 것이 좋습니다. 아래 테스트에 따라 점검하는 데 최대 30s가 소요될 수 있습니다.

1. 하드웨어 가속이 활성화된 경우 이 API는 Windows에서 약 2s, Mac에서 약 10s가 걸립니다.
2. 하드웨어 가속이 비활성화된 경우 이 API는 Windows 및 Mac 모두에서 약 30s가 걸립니다.



```
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}.`)
```

```
}
```

장치 점검을 위한 React 컴포넌트

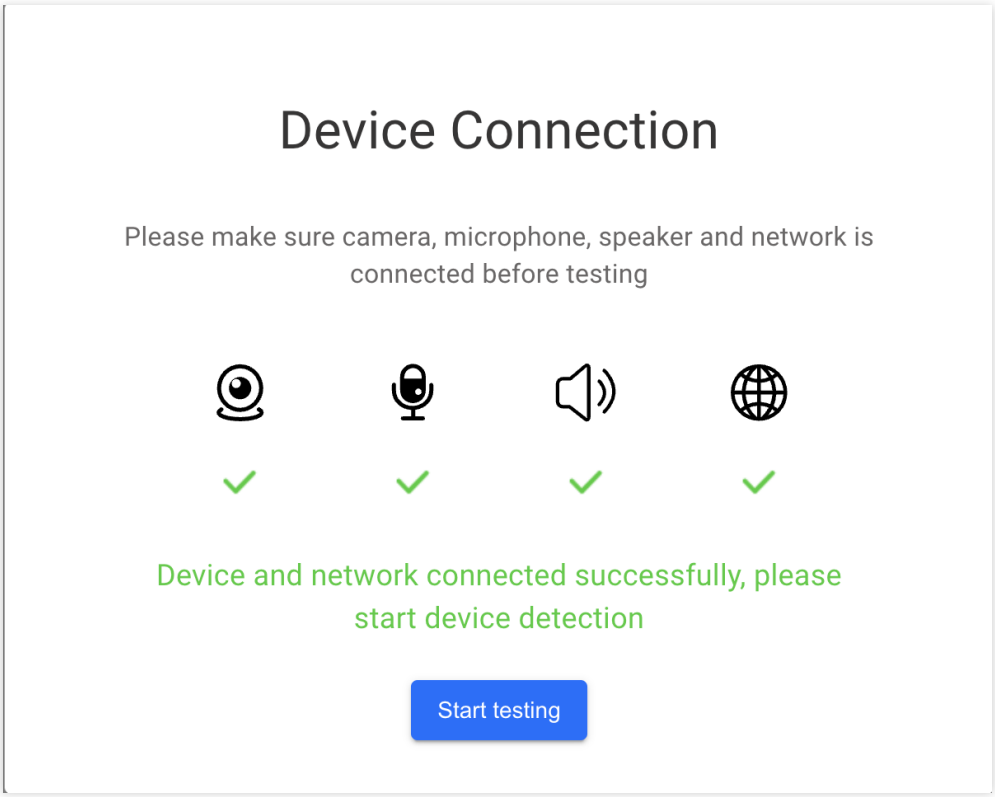
장치 점검 UI 컴포넌트 기능

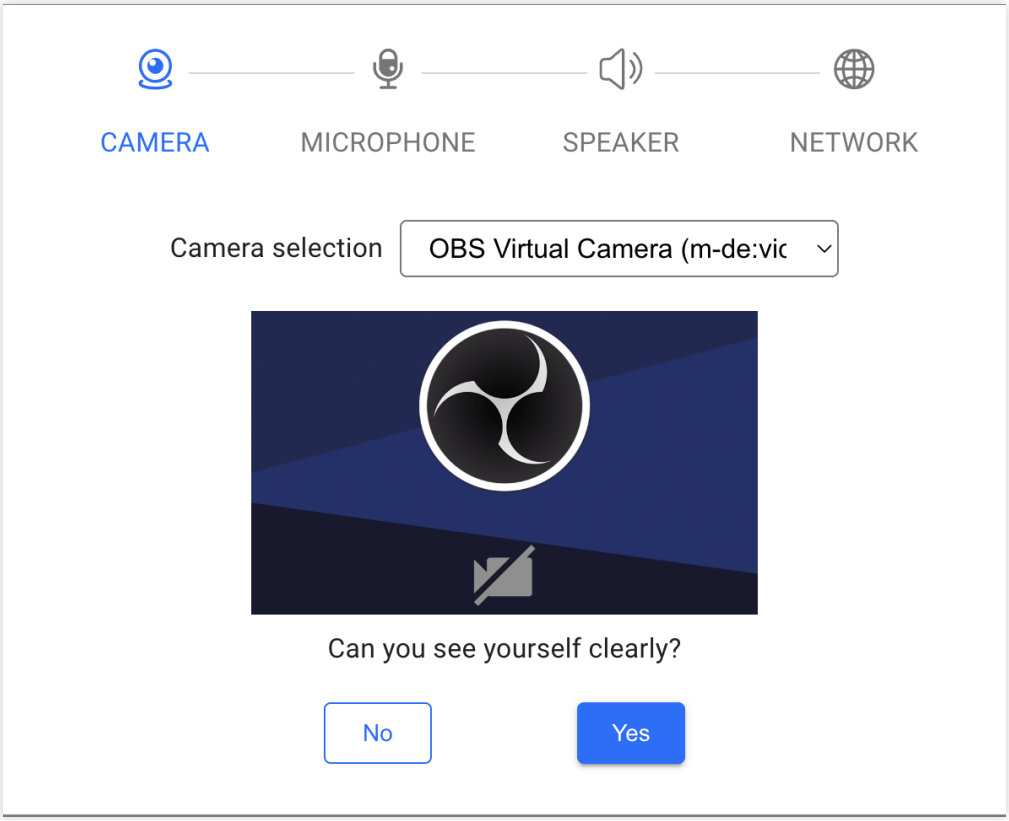
1. 장치 연결 및 로직 처리 점검
2. 네트워크 점검 로직 처리
3. 네트워크 점검 tab(옵션)
4. 중국어 및 영어 지원

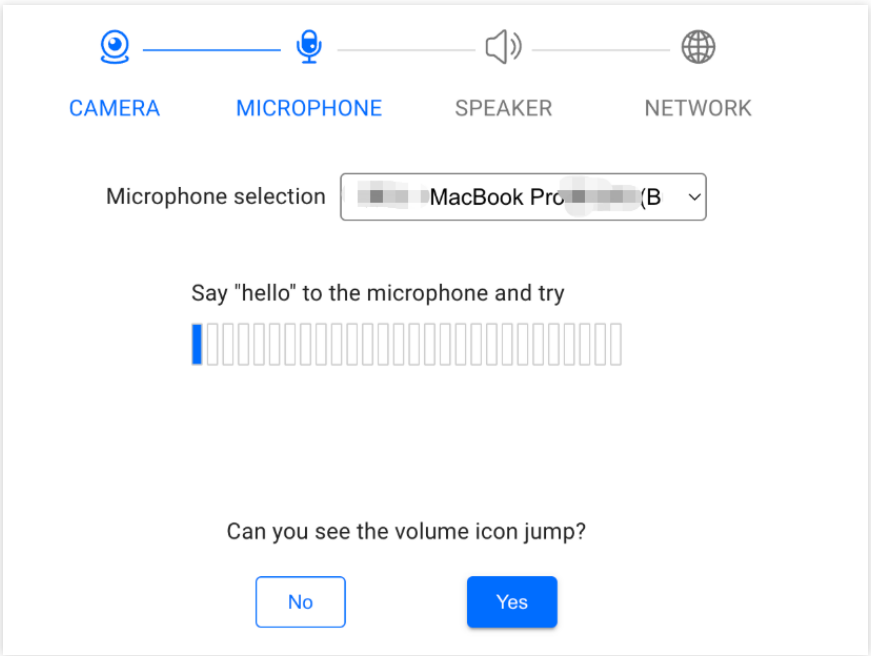
장치 점검 UI 컴포넌트 링크

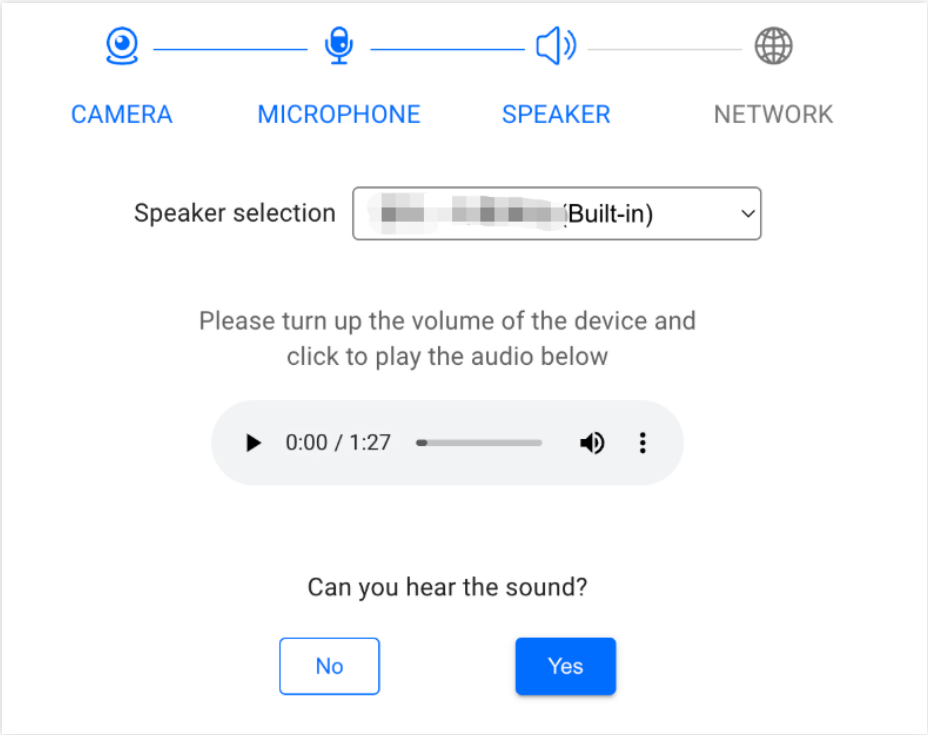
컴포넌트의 npm 패키지를 사용하는 방법에 대한 자세한 내용 참고: rtc-device-detector-react
컴포넌트의 소스 코드를 디버깅 방법에 대한 자세한 내용 참고: github/rtc-device-detector
컴포넌트를 가져오는 방법에 대한 자세한 내용 참고: WebRTC API Example

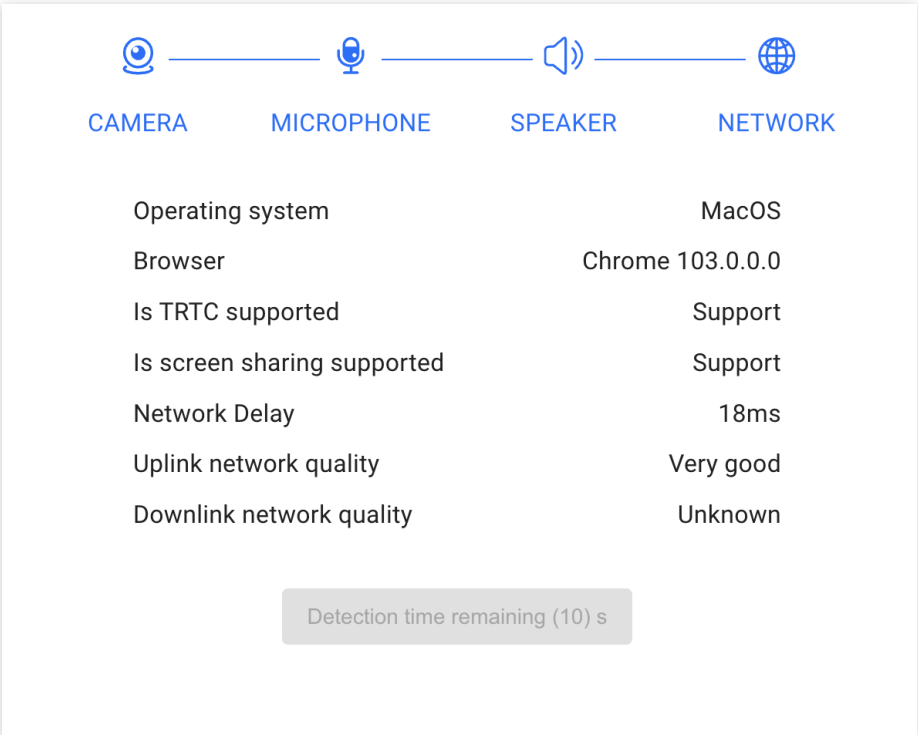
장치 점검 UI 컴포넌트 페이지

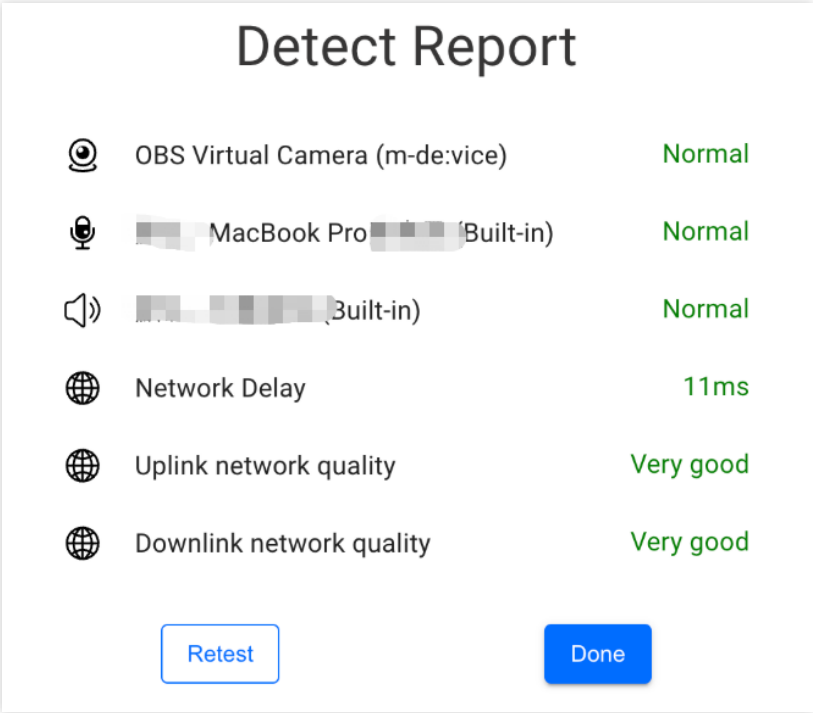








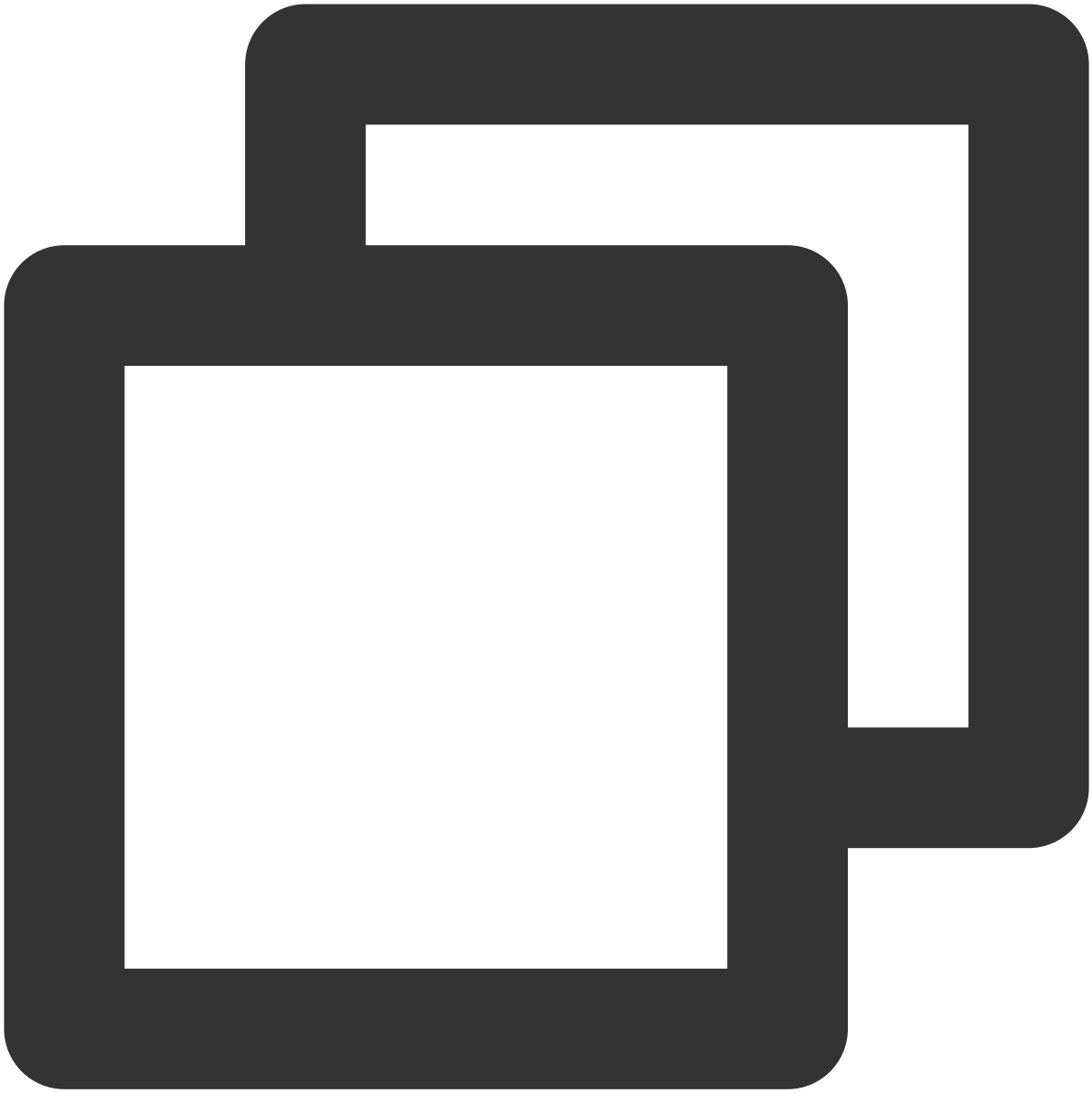




장치 및 네트워크 점검 로직

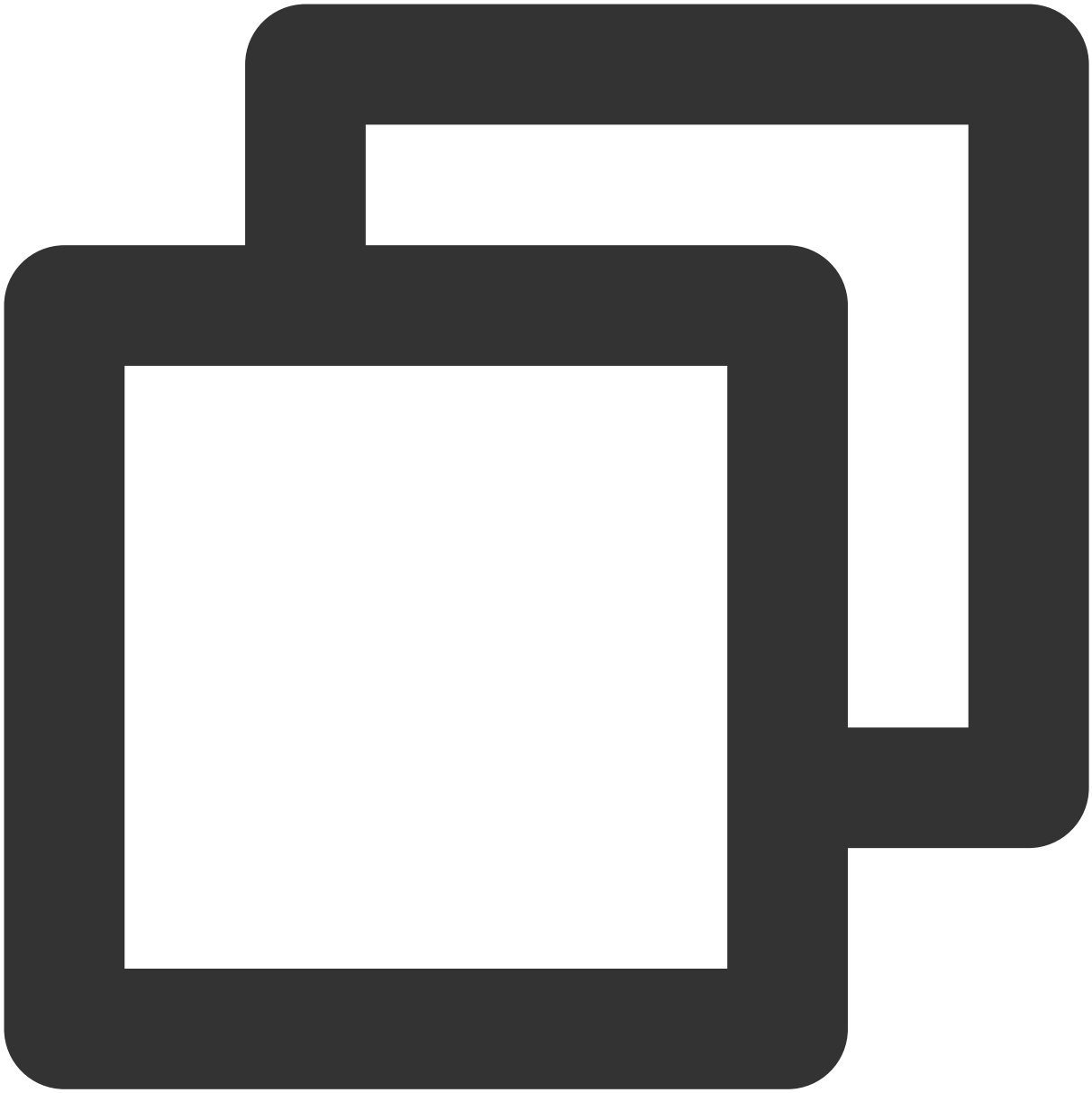
1) 장치 연결

장치 연결의 목적은 사용자의 장치에 카메라, 마이크, 스피커가 있고 네트워크에 연결되어 있는지 점검하는 것입니다. 카메라와 마이크가 있는 경우 시스템은 오디오/비디오 스트림을 가져오려고 시도하고 사용자에게 카메라와 마이크에 대한 액세스 권한을 부여하라는 메시지를 표시합니다.
기기에 카메라, 마이크, 스피커가 있는지 점검



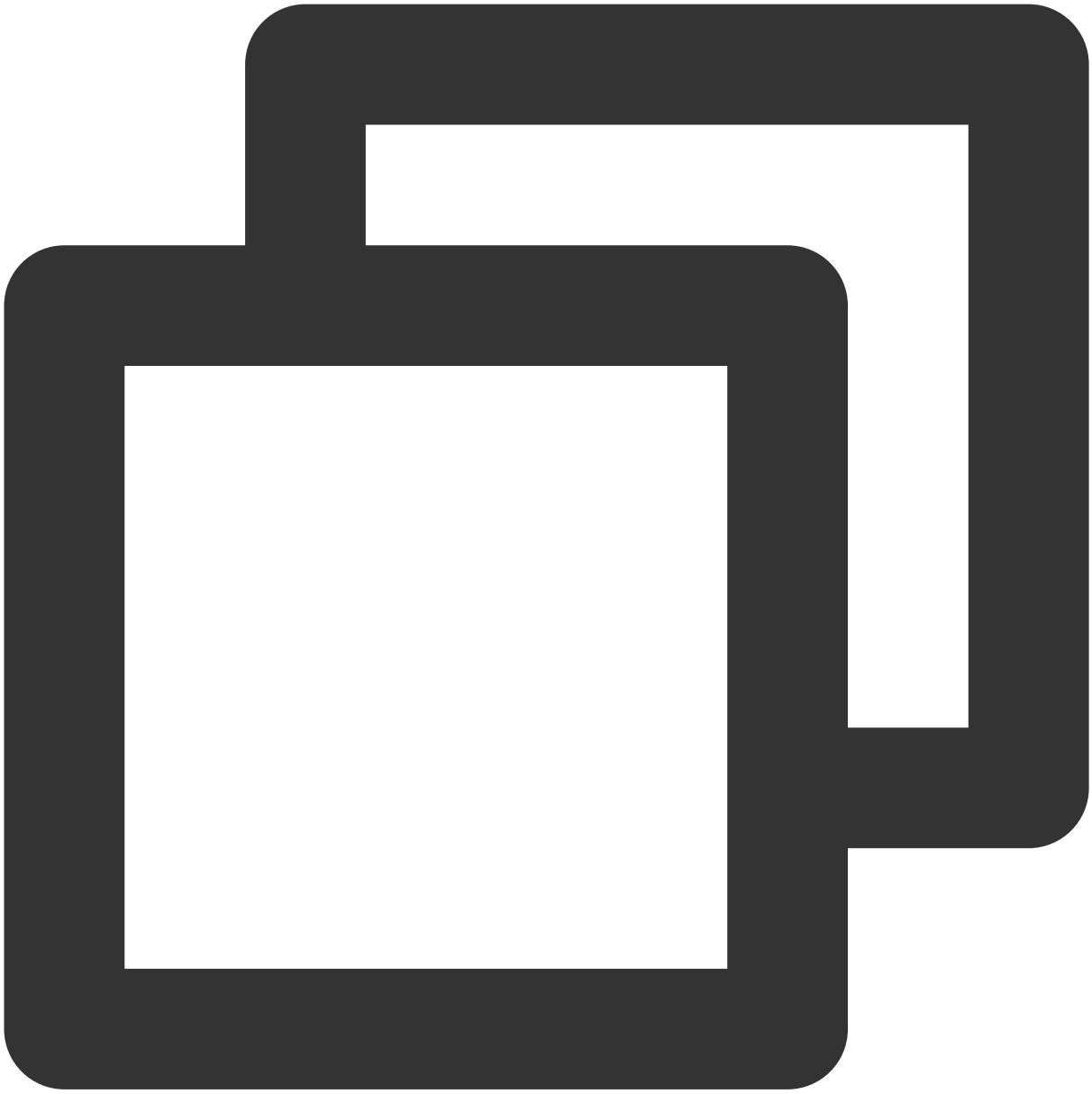
```
import TRTC from 'trtc-js-sdk';

const cameraList = await TRTC.getCameras();
const micList = await TRTC.getMicrophones();
const speakerList = await TRTC.getSpeakers();
const hasCameraDevice = cameraList.length > 0;
const hasMicrophoneDevice = micList.length > 0;
const hasSpeakerDevice = speakerList.length > 0;
```

카메라 및 마이크 액세스 권한 획득



```
navigator.mediaDevices
  .getUserMedia({ video: hasCameraDevice, audio: hasMicrophoneDevice })
  .then((stream) => {
      // 오디오/비디오 스트림 가져오기 성공
    // ...
      // 카메라와 마이크 릴리스
       stream.getTracks().forEach(track => track.stop());
  })
  .catch((error) => {
    // 오디오/비디오 스트림 가져오기 실패
  });
```

장치가 네트워크에 연결되어 있는지 점검

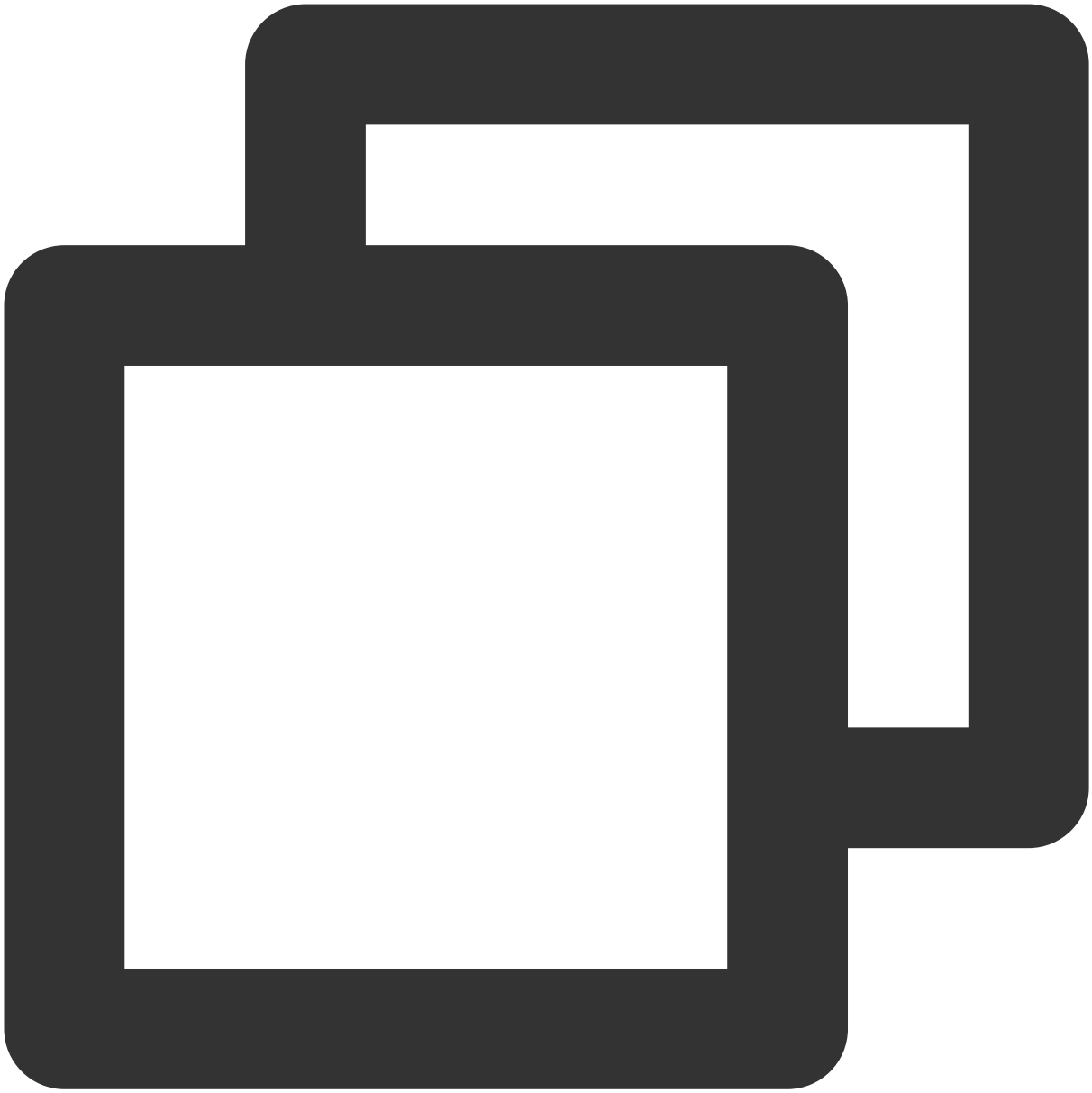


```
export function isOnline() {
  const url = 'https://web.sdk.qcloud.com/trtc/webrtc/assets/trtc-logo.png';
  return new Promise((resolve) => {
    try {
      const xhr = new XMLHttpRequest();
      xhr.onload = function () {
        resolve(true);
      };
      xhr.onerror = function () {
```

```
          resolve(false);
        };
        xhr.open('GET', url, true);
        xhr.send();
      } catch (err) {
        // console.log(err);
      }
    });
}
const isOnline = await isOnline();
```

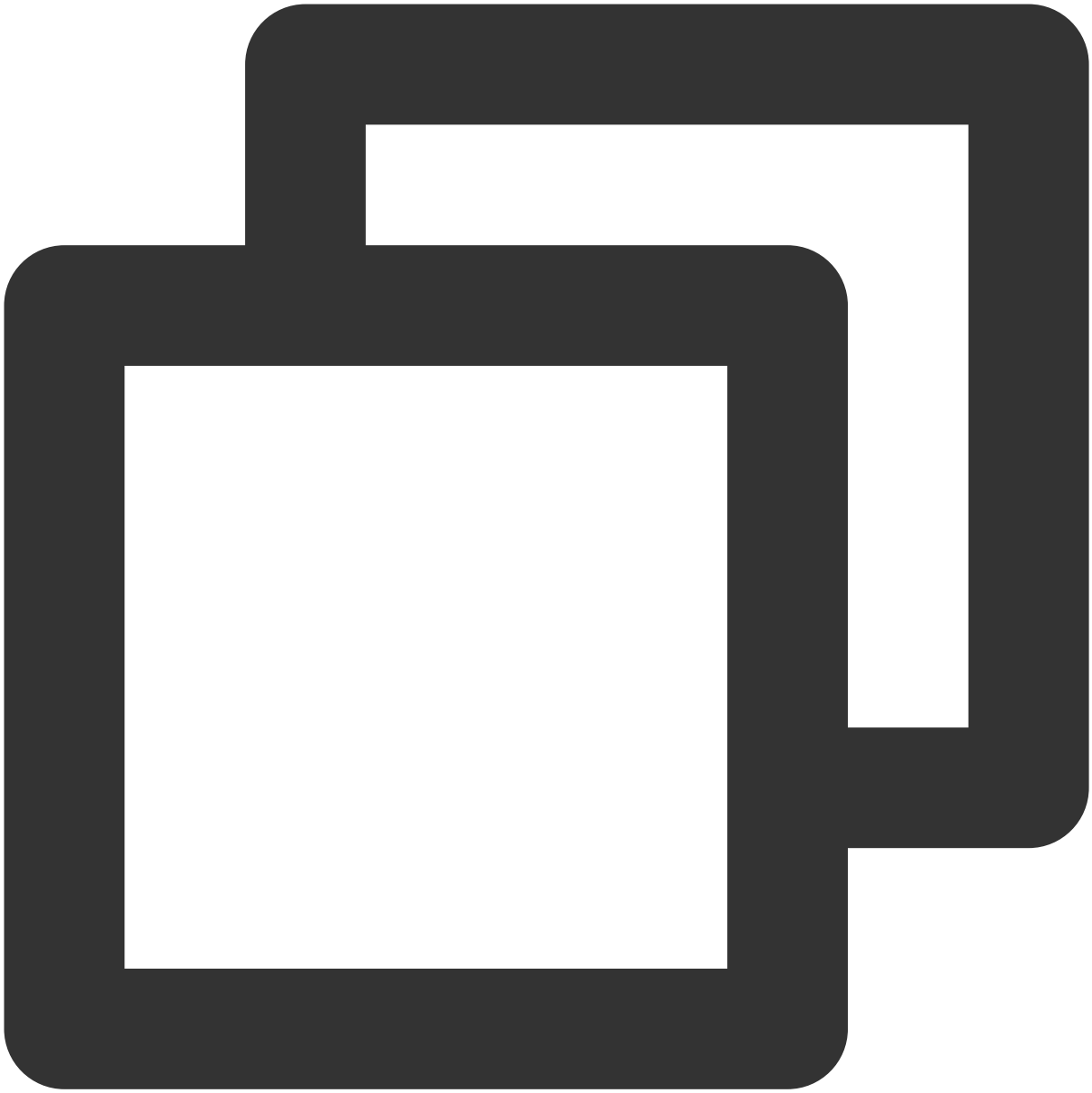
2) 카메라 점검

카메라 점검은 선택된 카메라가 캡쳐한 영상 스트림을 사용자가 카메라를 정상적으로 사용할 수 있는지 판단할 수 있도록 렌더링하는 것입니다.
카메라 목록을 가져옵니다. 기본적으로 목록의 첫 번째 장치가 기본적으로 사용됨



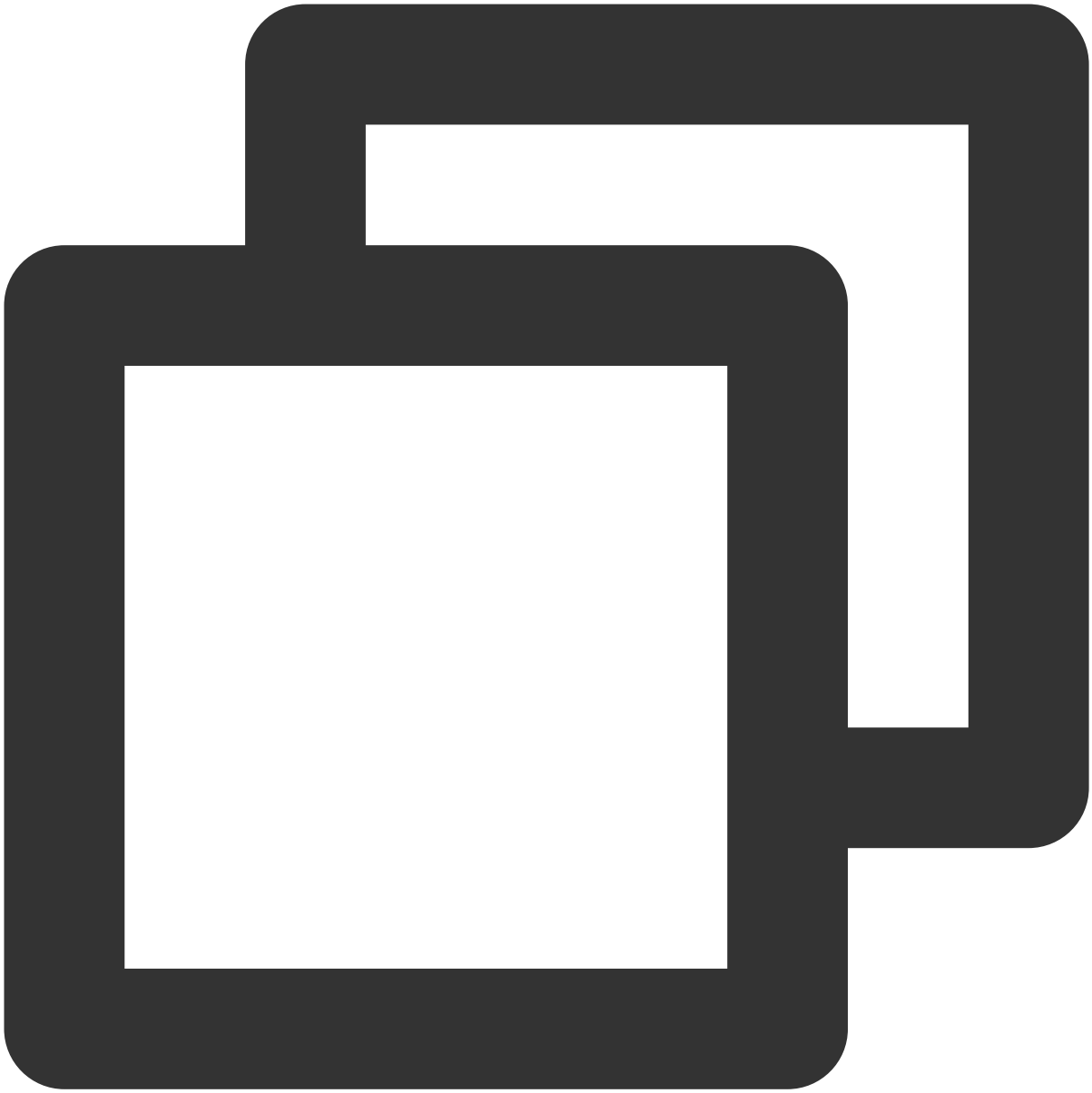
```
import TRTC from 'trtc-js-sdk';

let cameraList = await TRTC.getCameras();
let cameraId = cameraList[0].deviceId;
```

비디오 스트림을 초기화하고 ID가 camera-video인 dom 요소에서 재생



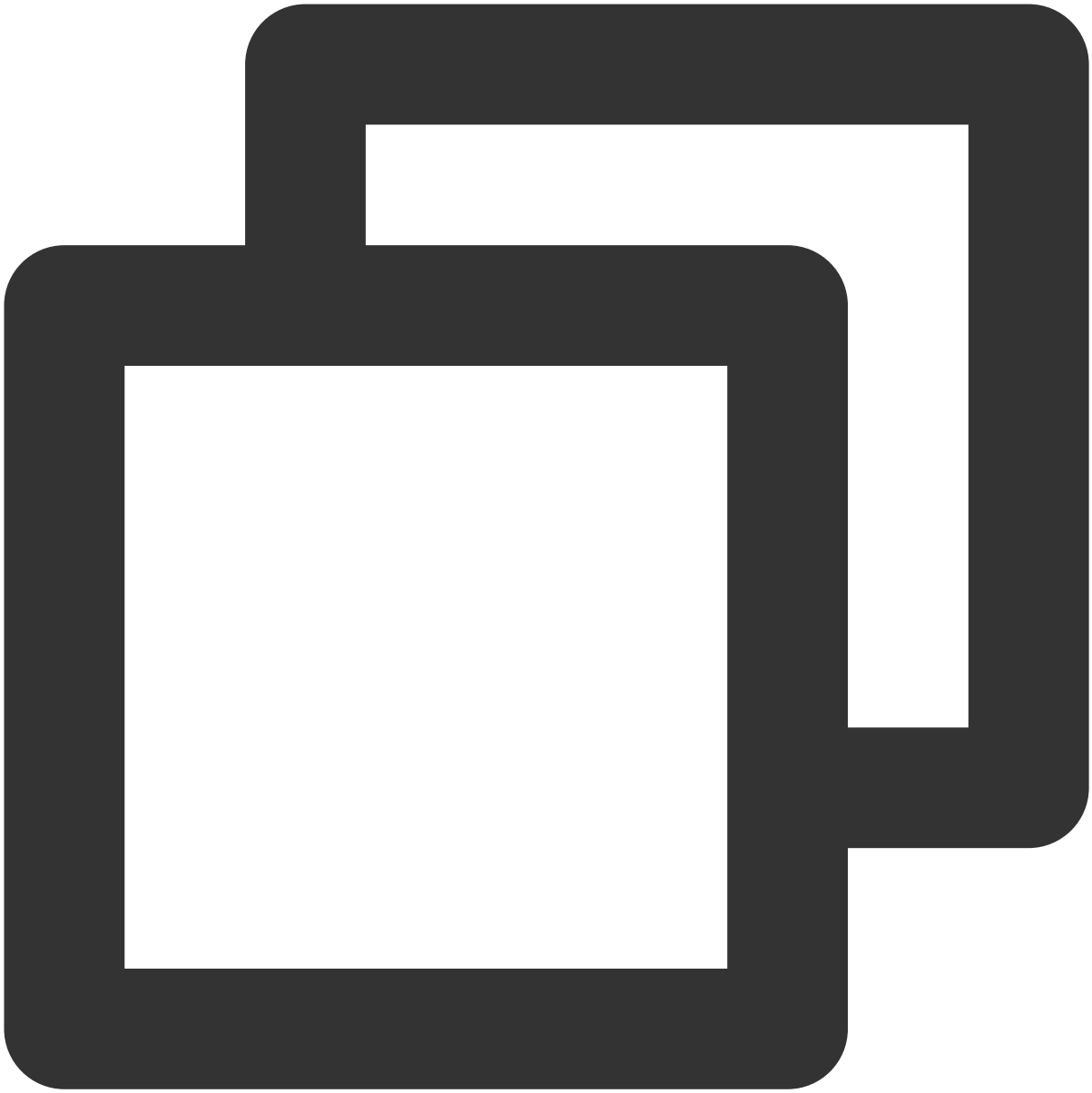
```
const localStream = TRTC.createStream({
  video: true,
  audio: false,
  cameraId,
});
await localStream.initialize();
localStream.play('camera-video');
```

사용자가 카메라를 전환한 후 스트림 업데이트



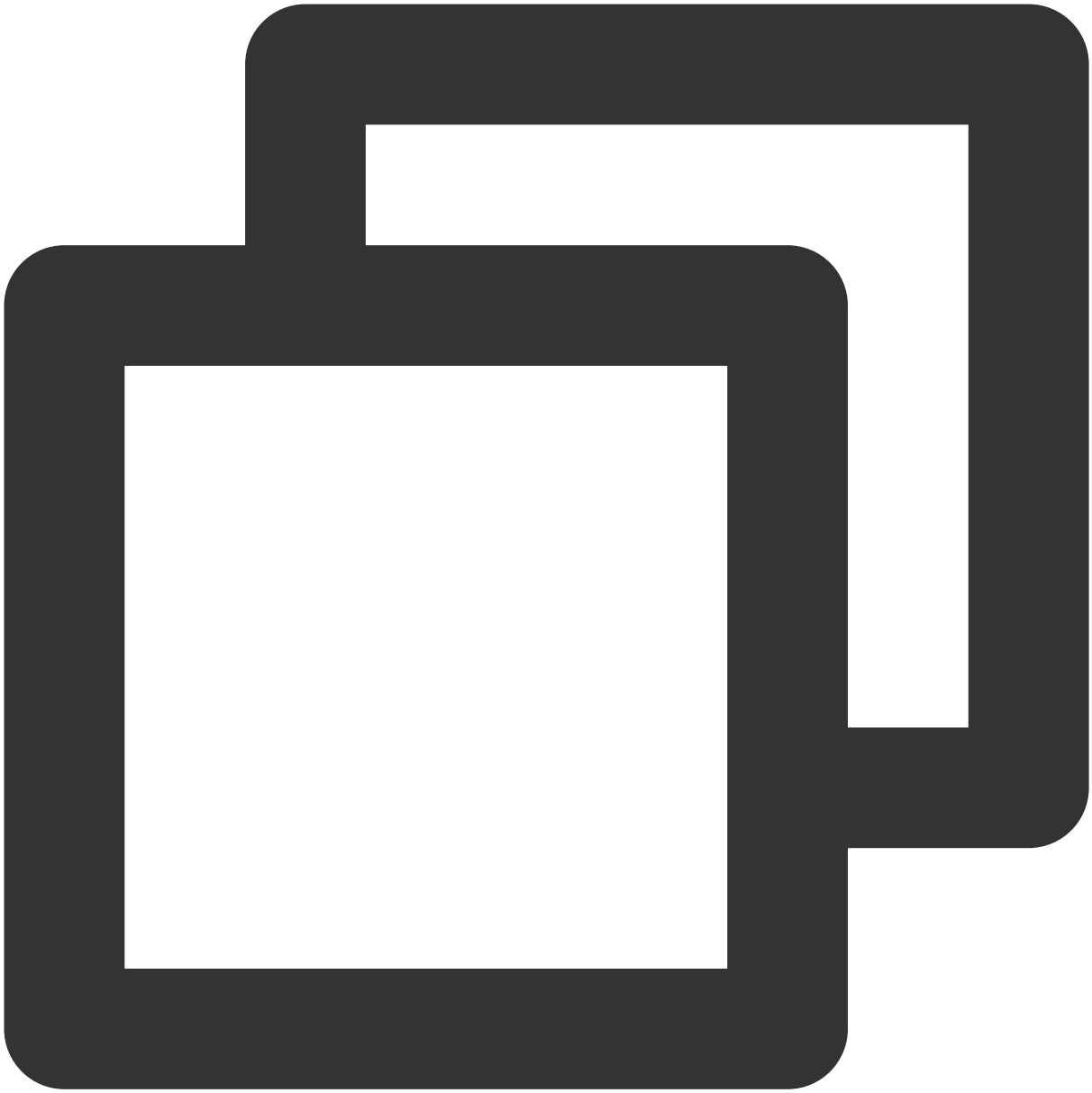
```
localStream.switchDevice('video', cameraId);
```

장치 연결/분리 시 수신



```
navigator.mediaDevices.addEventListener('devicechange', async () => {
  cameraList = await TRTC.getCameras();
    cameraId = cameraList[0].deviceId;
  localStream.switchDevice('video', cameraId);
})
```

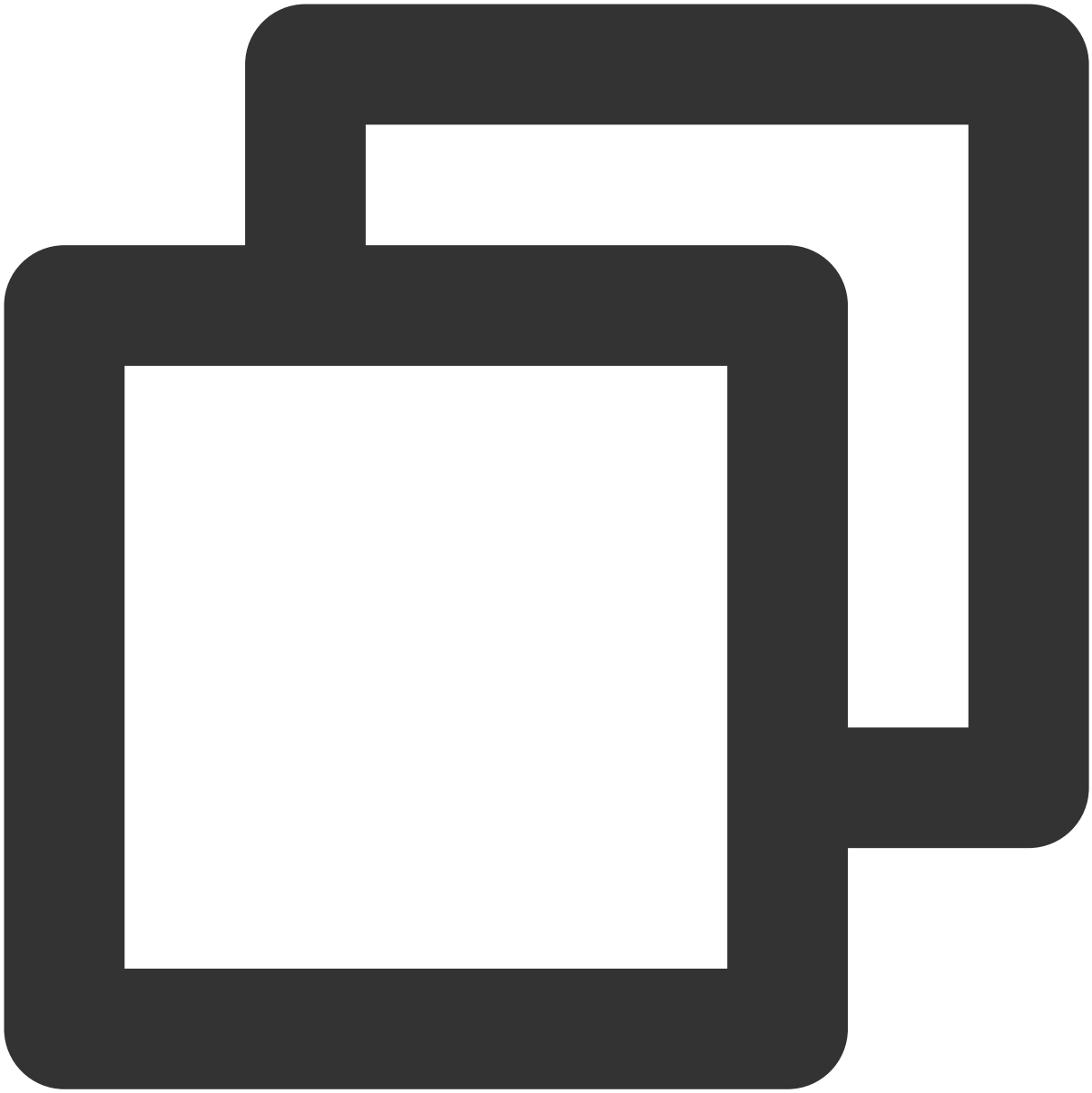
점검 완료 후 카메라 해제



```
localStream.close();
```

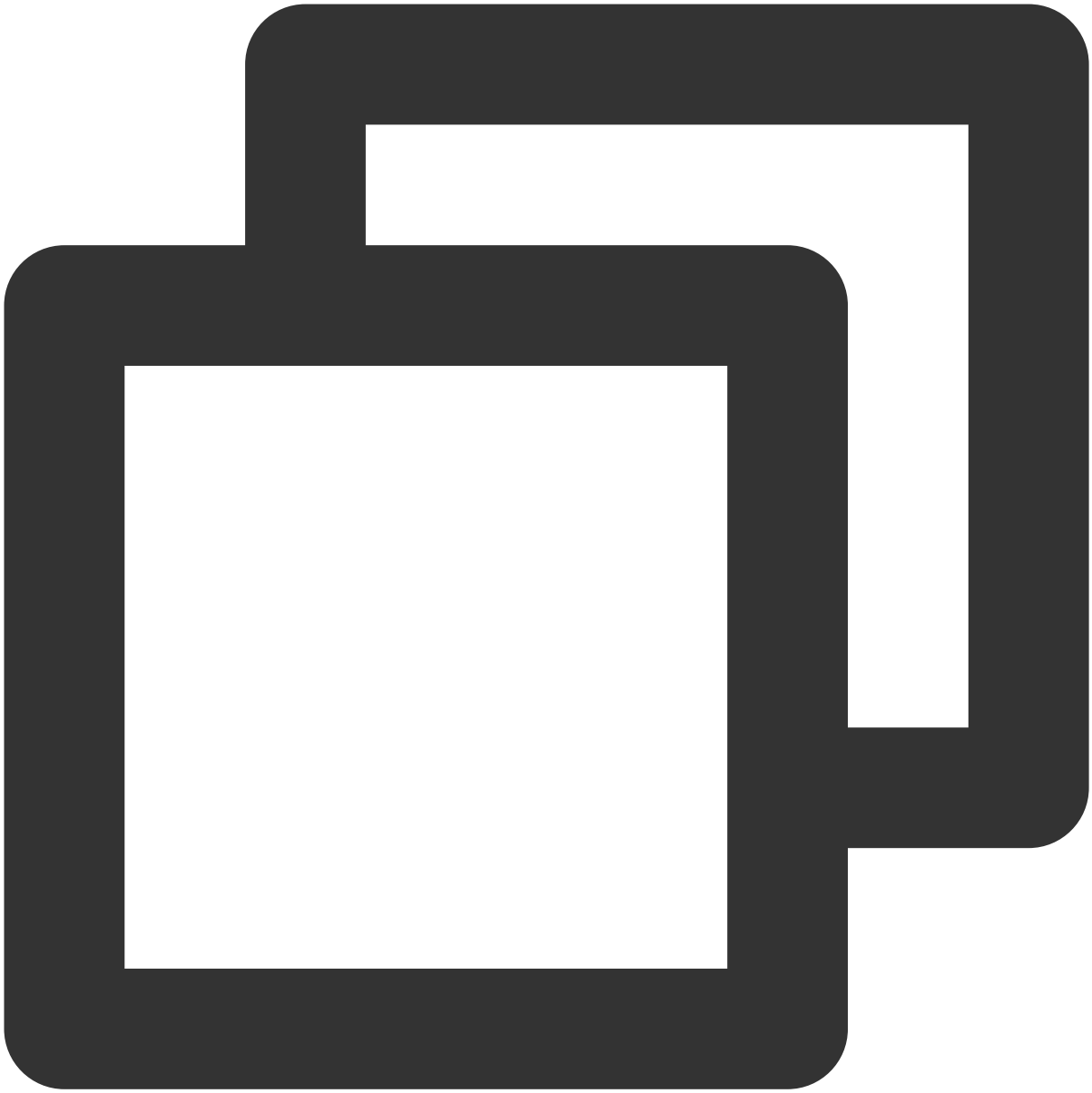
3) 마이크 점검

마이크 점검은 선택한 마이크가 캡처한 오디오 스트림의 볼륨을 렌더링하여 사용자가 마이크를 정상적으로 사용할 수 있는지 판단하는 데 도움이 됩니다.
마이크 목록을 가져옵니다. 기본적으로 목록의 첫 번째 장치가 사용됨



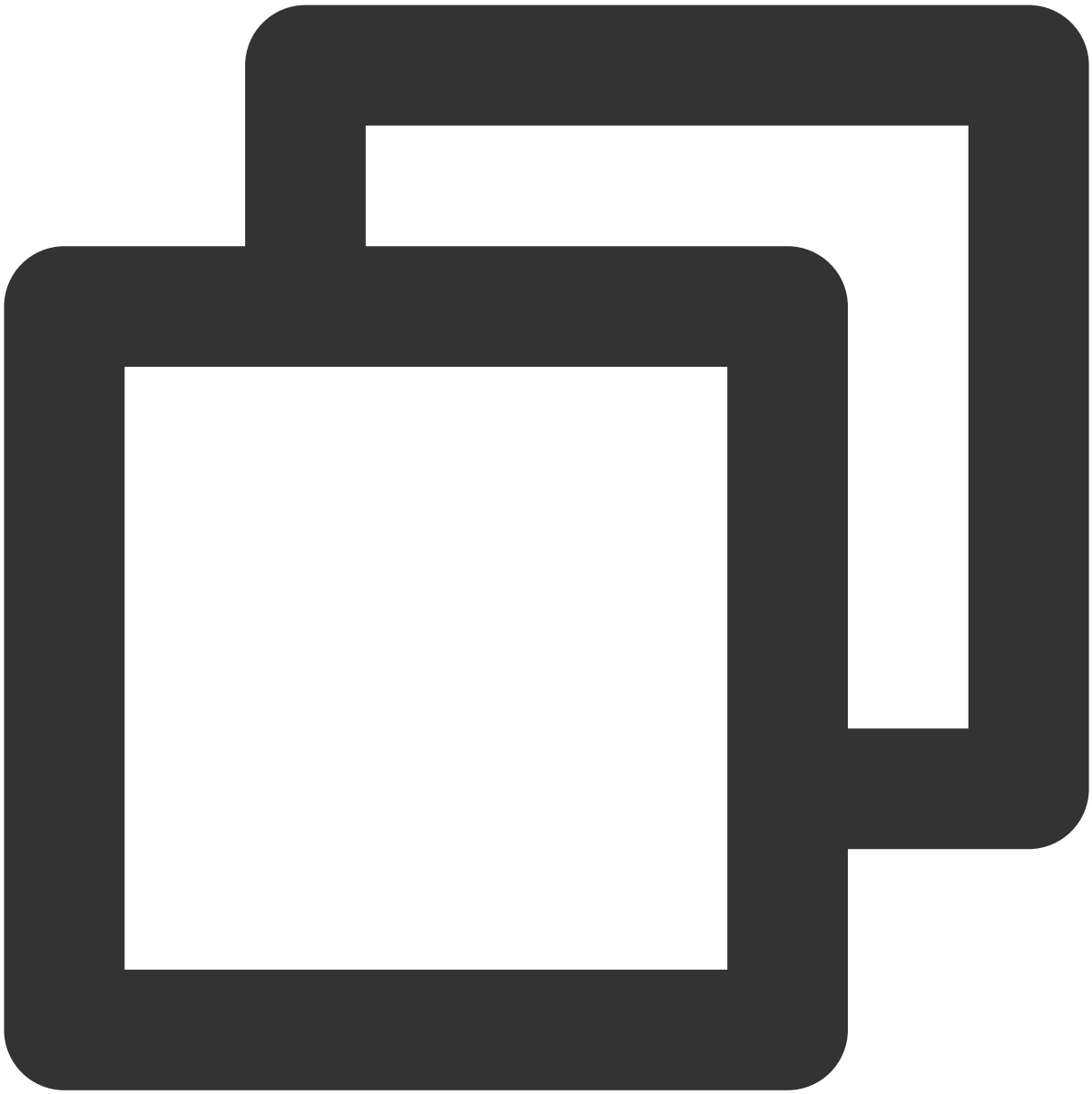
```
import TRTC from 'trtc-js-sdk';

let microphoneList = await TRTC.getMicrophones();
let microphoneId = microphoneList[0].deviceId;
```

오디오 스트림을 초기화하고 ID가 audio-container인 dom 요소에서 재생



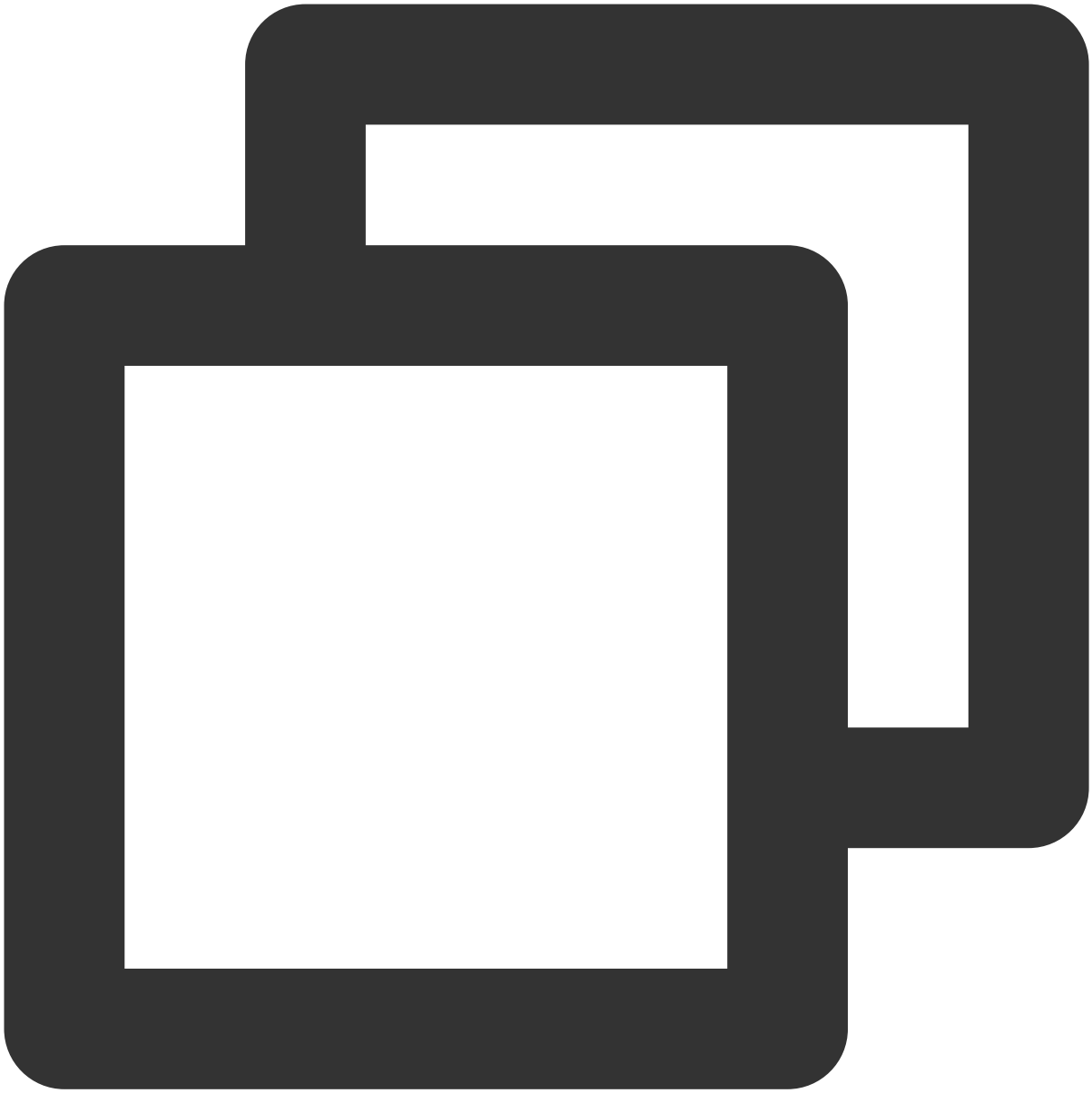
```
const localStream = TRTC.createStream({
  video: false,
  audio: true,
  microphoneId,
});
await localStream.initialize();
localStream.play('audio-container');
timer = setInterval(() => {
  const volume = localStream.getAudioLevel();
}, 100);
```

사용자가 마이크를 전환한 후 스트림 업데이트



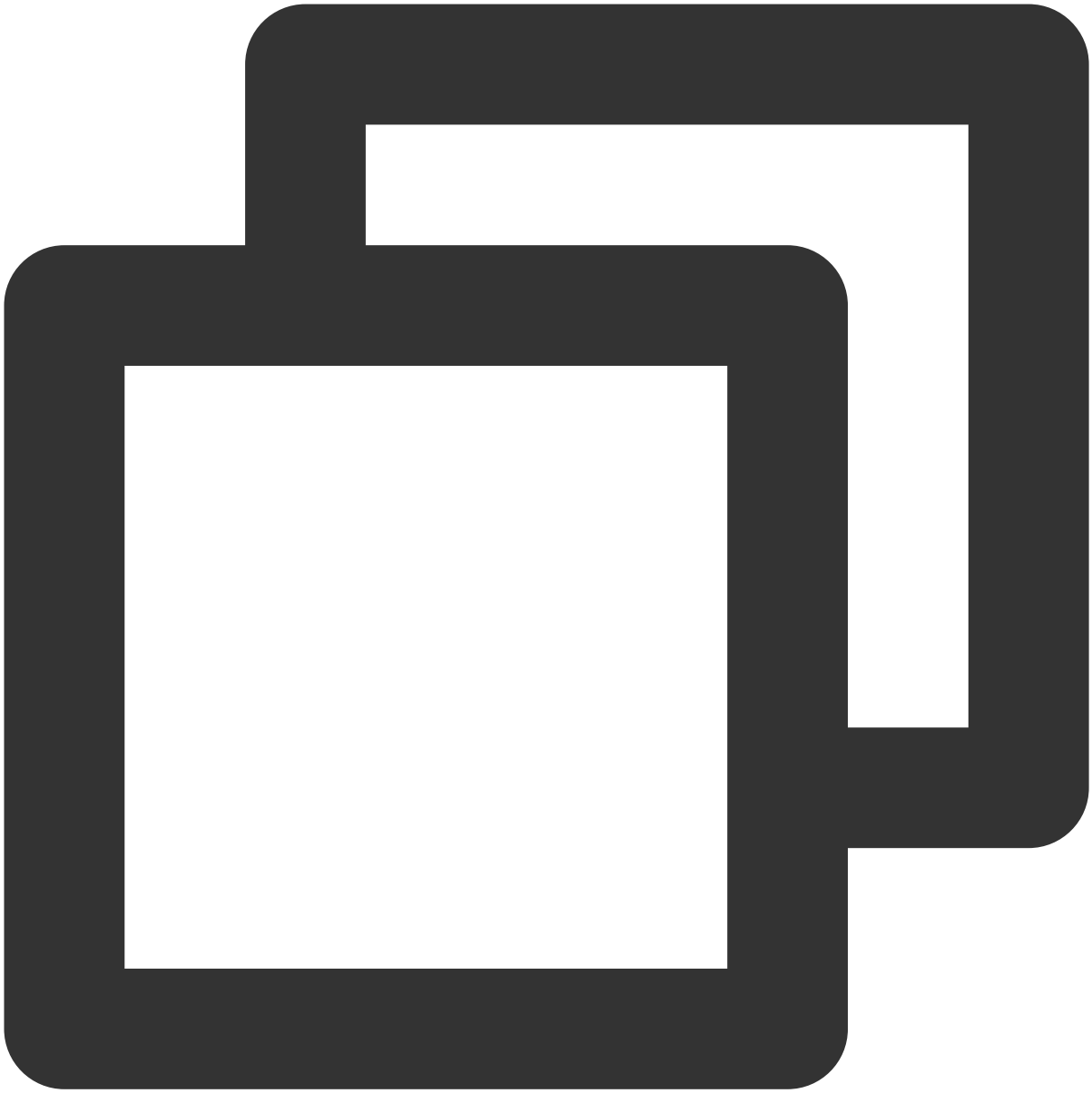
```
// 사용자가 선택한 새로운 microphoneId 가져오기
localStream.switchDevice('audio', microphoneId);
```

장치 연결/분리 시 수신



```
navigator.mediaDevices.addEventListener('devicechange', async () => {
  microphoneList = await TRTC.getMicrophones();
    microphoneId = microphoneList[0].deviceId;
  localStream.switchDevice('audio', microphoneId);
})
```

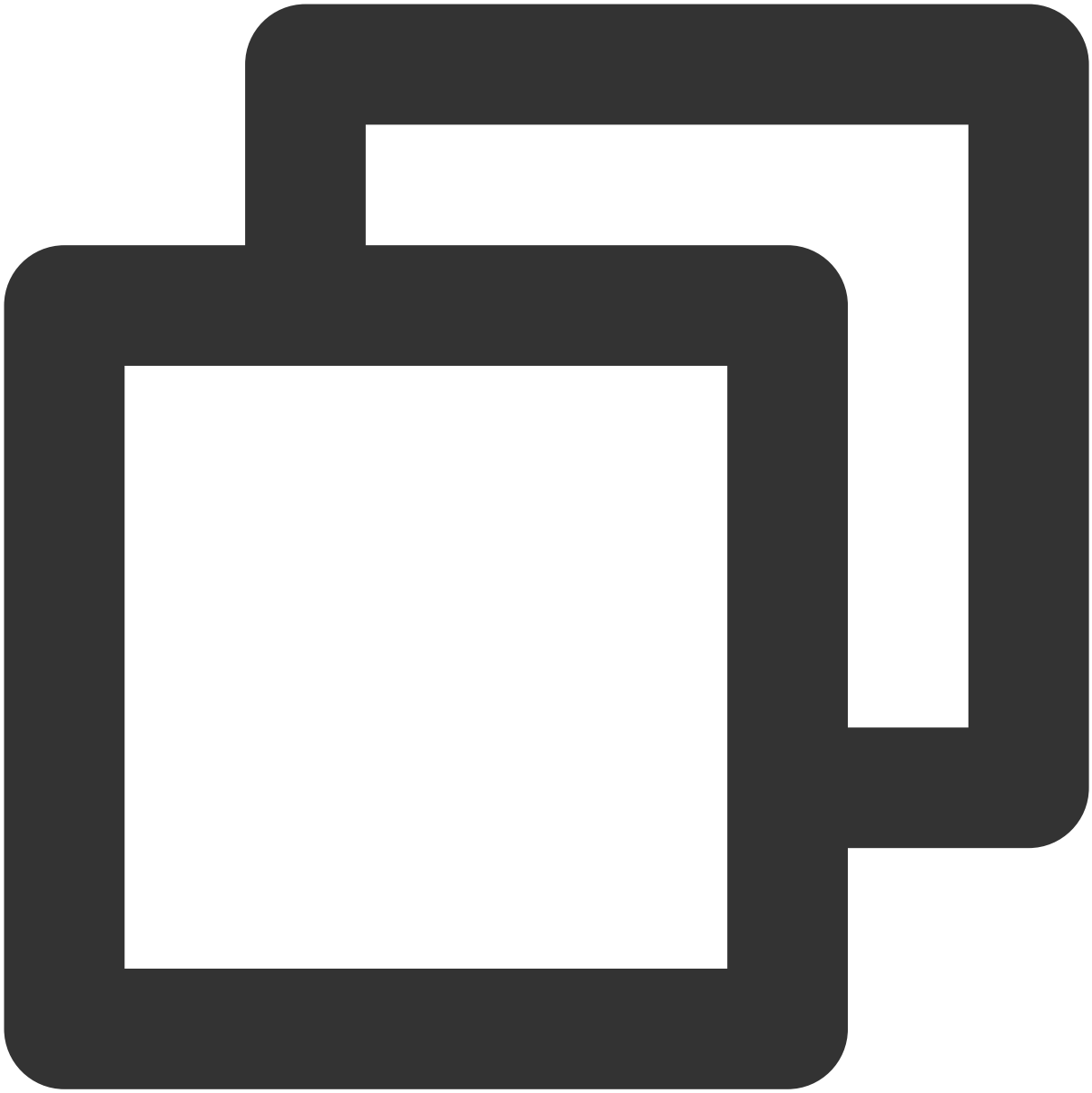
점검 완료 후 마이크에서 손을 떼고 볼륨으로 수신 중지



```
localStream.close();
clearInterval(timer);
```

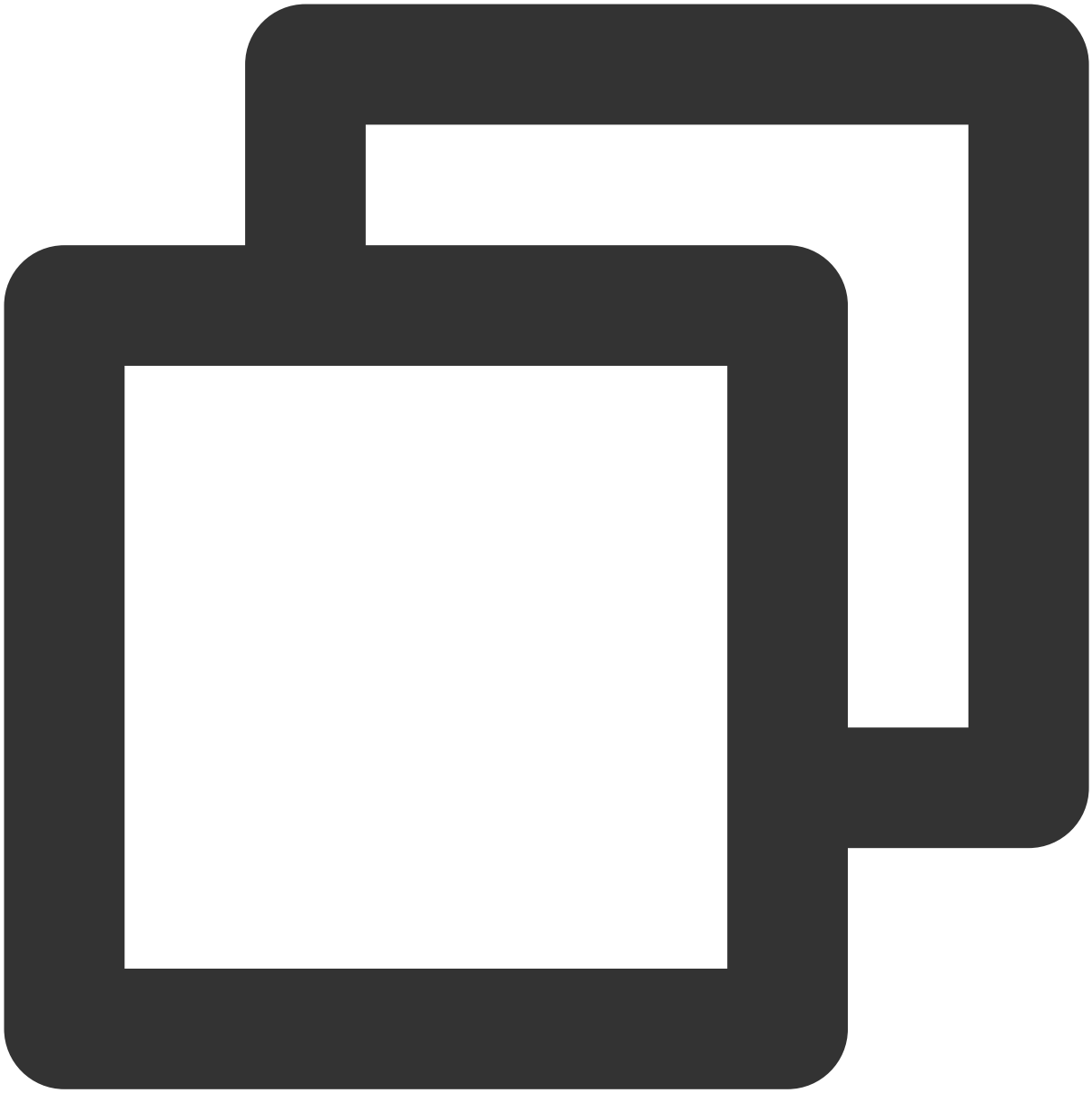
4) 스피커 점검

스피커 점검은 사용자가 오디오를 재생하여 선택한 스피커를 정상적으로 사용할 수 있는지 점검할 수 있는 오디오 플레이어를 제공합니다.
mp3 플레이어를 제공하고 사용자에게 장치 재생 볼륨을 높이고 mp3 파일을 재생하여 스피커가 정상인지 점검하라는 메시지를 표시합니다.



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

점검 완료 후 재생 중지



```
const audioPlayer = document.getElementById('audio-player');
if (!audioPlayer.paused) {
    audioPlayer.pause();
}
audioPlayer.currentTime = 0;
```

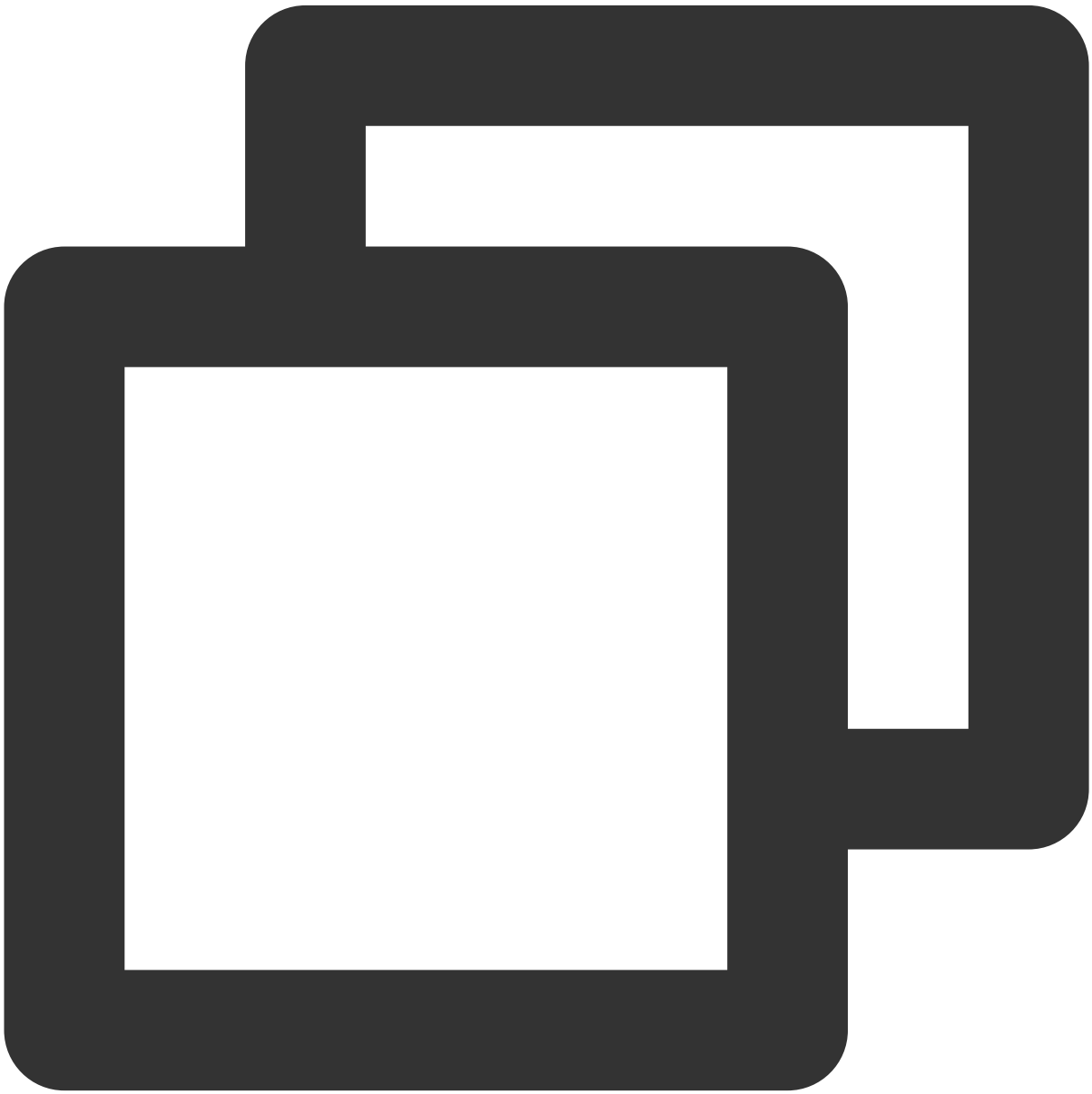
5) 네트워크 점검

TRTC.createClient를 호출하여 uplinkClient 및 downlinkClient의 두 Client를 생성합니다.
두 Client가 같은 방에 들어가도록 합니다.

스트림을 푸시하려면 uplinkClient를 사용하십시오. ClientEvent.NETWORK_QUALITY 이벤트를 수신하고 업스트림 네트워크 품질을 점검하십시오.
downlinkClient를 사용하여 스트림을 가져옵니다. ClientEvent.NETWORK_QUALITY 이벤트를 수신하고 다운스트림 네트워크 품질을 점검하십시오.
전체 프로세스는 약 15s 동안 지속되며 평균 네트워크 품질은 결국 업스트림 및 다운스트림 네트워크 품질을 평가하는 데 사용됩니다.

주의사항 :

점검 절차에 따라 소정의 기본 서비스 요금이 발생할 수 있습니다. 푸시 해상도가 지정되지 않은 경우 스트림은 기본적으로 640*480 해상도로 푸시됩니다.



```
let uplinkClient = null; // 업스트림 네트워크 품질 점검
let downlinkClient = null; // 다운스트림 네트워크 품질 점검
let localStream = null; // 테스트용 스트림
let testResult = {
  // 업스트림 네트워크 품질 데이터 기록
  uplinkNetworkQualities: [],
  // 다운스트림 네트워크 품질 데이터 기록
  downlinkNetworkQualities: [],
  average: {
    uplinkNetworkQuality: 0,
    downlinkNetworkQuality: 0
  }
}

// 1. 업스트림 네트워크 품질 점검
async function testUplinkNetworkQuality() {
  uplinkClient = TRTC.createClient({
    sdkAppId: 0, // sdkAppId 입력
    userId: 'user_uplink_test',
    userSig: '', // uplink_test의 userSig
    mode: 'rtc'
  });

  localStream = TRTC.createStream({ audio: true, video: true });
  // 실제 비즈니스 시나리오를 기반으로 video profile 설정
  localStream.setVideoProfile('480p');
  await localStream.initialize();

  uplinkClient.on('network-quality', event => {
    const { uplinkNetworkQuality } = event;
    testResult.uplinkNetworkQualities.push(uplinkNetworkQuality);
  });

  // 테스트를 위한 방으로 들어갑니다. 충돌을 피하기 위해 방 ID는 무작위여야 합니다.
  await uplinkClient.join({ roomId: 8080 });
  await uplinkClient.publish(localStream);
}

// 2. 다운스트림 네트워크 품질 점검
async function testDownlinkNetworkQuality() {
  downlinkClient = TRTC.createClient({
    sdkAppId: 0, // sdkAppId 입력
    userId: 'user_downlink_test',
    userSig: '', // userSig
    mode: 'rtc'
  });
```

```
  downlinkClient.on('stream-added', async event => {
    await downlinkClient.subscribe(event.stream, { audio: true, video: true });
        // 성공적인 구독 후 네트워크 품질 이벤트 수신 시작
    downlinkClient.on('network-quality', event => {
      const { downlinkNetworkQuality } = event;
      testResult.downlinkNetworkQualities.push(downlinkNetworkQuality);
    });
  })
  // 테스트를 위한 방으로 들어갑니다. 충돌을 피하기 위해 방 ID는 무작위여야 합니다.
  await downlinkClient.join({ roomId: 8080 });
}

// 3. 점검 시작
testUplinkNetworkQuality();
testDownlinkNetworkQuality();

// 4. 15s 후에 점검을 중지하고 평균 네트워크 품질 계산
setTimeout(() => {
  // 평균 업스트림 네트워크 품질 계산
  if (testResult.uplinkNetworkQualities.length > 0) {
    testResult.average.uplinkNetworkQuality = Math.ceil(
      testResult.uplinkNetworkQualities.reduce((value, current) => value + current,
    );
  }

  if (testResult.downlinkNetworkQualities.length > 0) {
    // 평균 다운스트림 네트워크 품질 계산
    testResult.average.downlinkNetworkQuality = Math.ceil(
      testResult.downlinkNetworkQualities.reduce((value, current) => value + curren
    );
  }

  // 점검이 종료됩니다. 관련 상태를 지우십시오.
  uplinkClient.leave();
  downlinkClient.leave();
  localStream.close();
}, 15 * 1000);
```

TRTC 호환성 점검 페이지

TRTC 점검 페이지에서 현재 TRTC SDK를 사용하고자 하는 환경을 점검할 수 있습니다. 보고서 생성을 클릭하여 환경 점검 및 문제 해결을 위한 현재 환경 보고서를 얻을 수도 있습니다.

네트워크 품질 테스트 Android&iOS&Windows&Mac

최종 업데이트 날짜: : 2024-07-24 16:43:38

일반 사용자가 네트워크 품질을 평가하기 어렵기 때문에 영상 통화 전에 네트워크 테스트를 해보는 것을 권장합니다. 속도 테스트를 통해 네트워크 품질을 보다 직관적으로 평가할 수 있습니다.

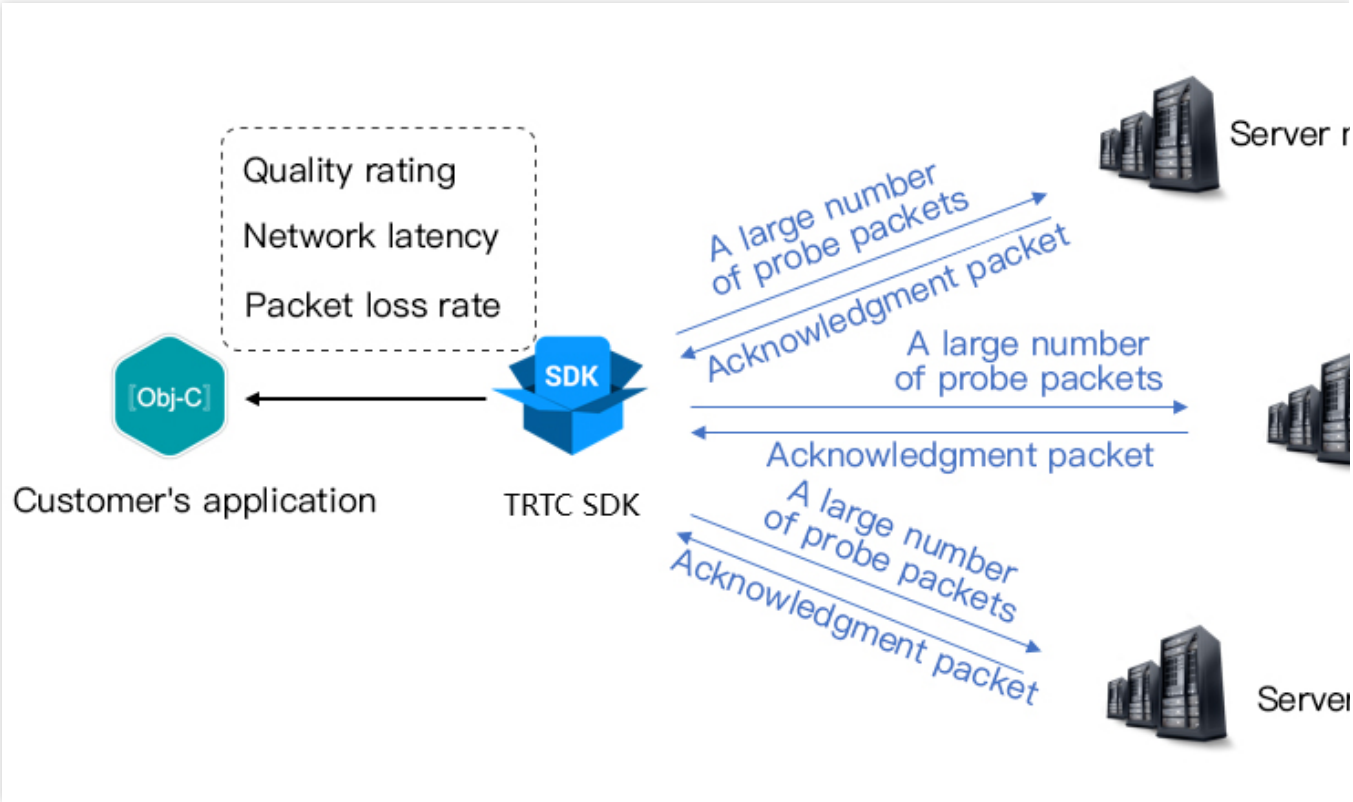
주의 사항

영상 통화 중에는 테스트하지 마십시오. 통화 품질에 영향이 미칠 수 있습니다.
속도 테스트 자체가 일정량의 트래픽을 소모하기 때문에 매우 적은 양의 추가 트래픽 비용이 발생합니다(기본적으로 무시할 수 있음).

지원 플랫폼

iOS	Android	Mac OS	Windows	Electron	Web
✓	✓	✓	✓	✓	✓(참고: Web 튜토리얼)

속도 테스트 원리



속도 테스트의 원리는 SDK가 감지 패킷 배치를 서버 노드로 보낸 다음 반환된 패킷의 품질을 계산하고 속도 테스트 결과를 콜백 인터페이스를 통해 알려주는 것입니다.
속도 테스트 결과는 SDK의 이후 서버 선택 정책을 최적화하는 데 사용됩니다. 따라서 사용자가 최초 통화 전 먼저 속도 테스트를 진행하는 것을 권장하며, 이는 최적의 서버를 선택하는 데 도움이 됩니다. 또한 테스트 결과가 매우 이상적이지 않는 경우 시각화된 UI를 통해 사용자에게 더 나은 네트워크를 선택하도록 안내할 수 있습니다.
속도 테스트 결과 (TRTCSpeedTestResult)는 다음 필드를 포함합니다.

필드	의미	의미 설명
success	성공 여부	테스트 성공 여부
errMsg	오류 정보	대역폭 테스트에 대한 자세한 오류 정보
ip	서버 IP	속도 테스트 서버의 IP
quality	네트워크 품질 평가	평가 알고리즘에 의해 계산된 네트워크 품질은 loss가 낮을수록 rtt가 작아지고 점수가 높아집니다.
upLostRate	업스트림 패킷 손실률	범위는 [0 - 1.0]입니다. 예를 들어 0.3은 서버로 전송되는 10개의 패킷 중 3개의 패킷이 도중 손실될 수 있음을 의미합니다.
downLostRate	다운스트림 패킷 손실률	범위는 [0 - 1.0]입니다. 예를 들어 0.2는 서버에서 수신되는 10개의 패킷 중 2개의 패킷이 도중 손실될 수 있음을 의미합니다.

rtt	네트워크 딜레이	SDK와 서버 사이에 소요되는 시간을 나타냅니다. 값이 작을수록 좋습니다. 정상 값은 10ms - 100ms입니다.
availableUpBandwidth	업스트림 대역폭	예상 업스트림 대역폭. 단위: kbps, -1: 잘못된 값.
availableDownBandwidth	다운스트림 대역폭	예상 다운스트림 대역폭. 단위: kbps, -1: 잘못된 값.

속도 테스트 방법

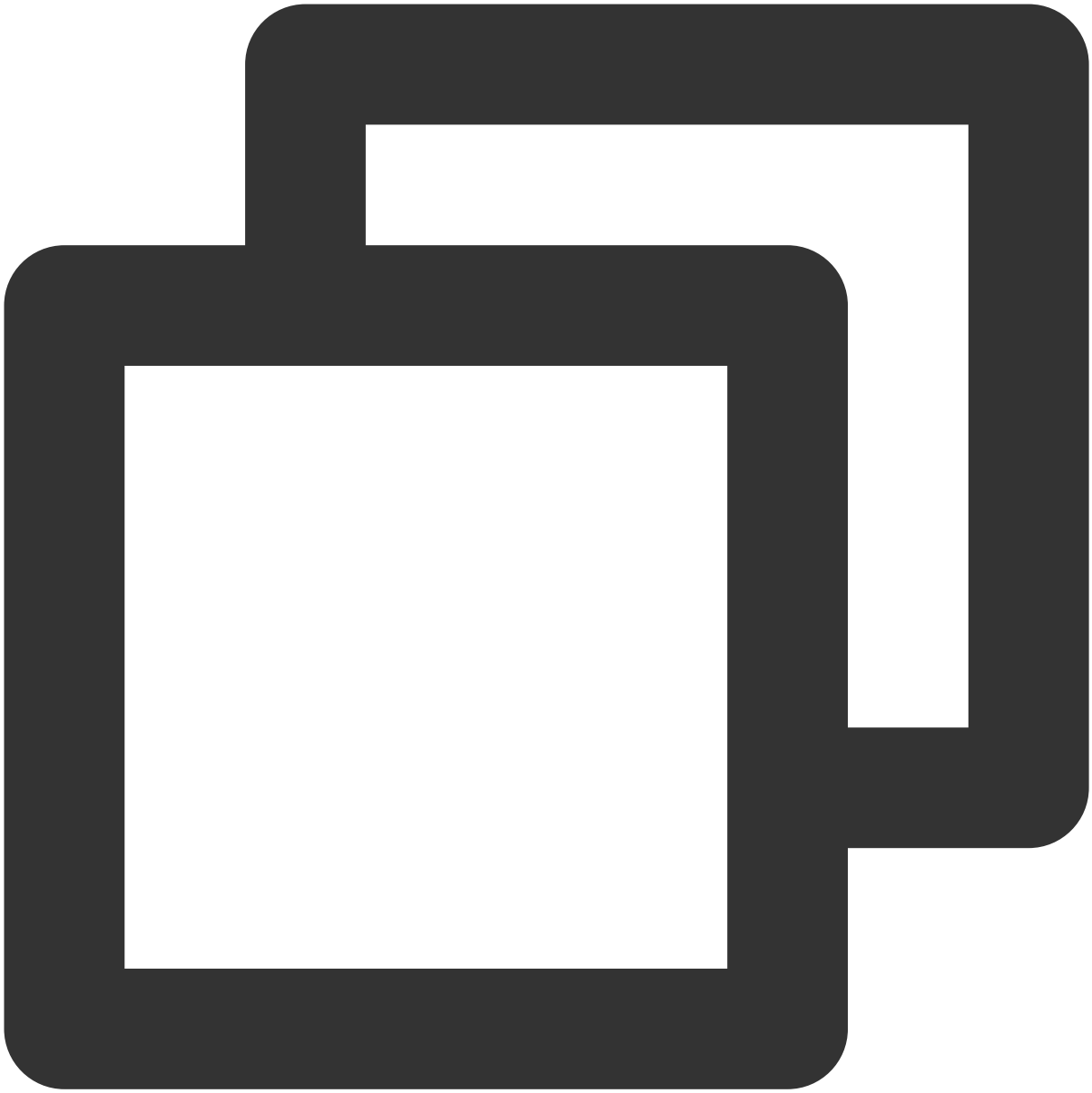
TRTCCloud의 `startSpeedTest` 기능을 통해 속도 측정 기능을 실행할 수 있습니다. 속도 측정 결과는 콜백 함수를 통해 반환됩니다.

Objective-C

Java

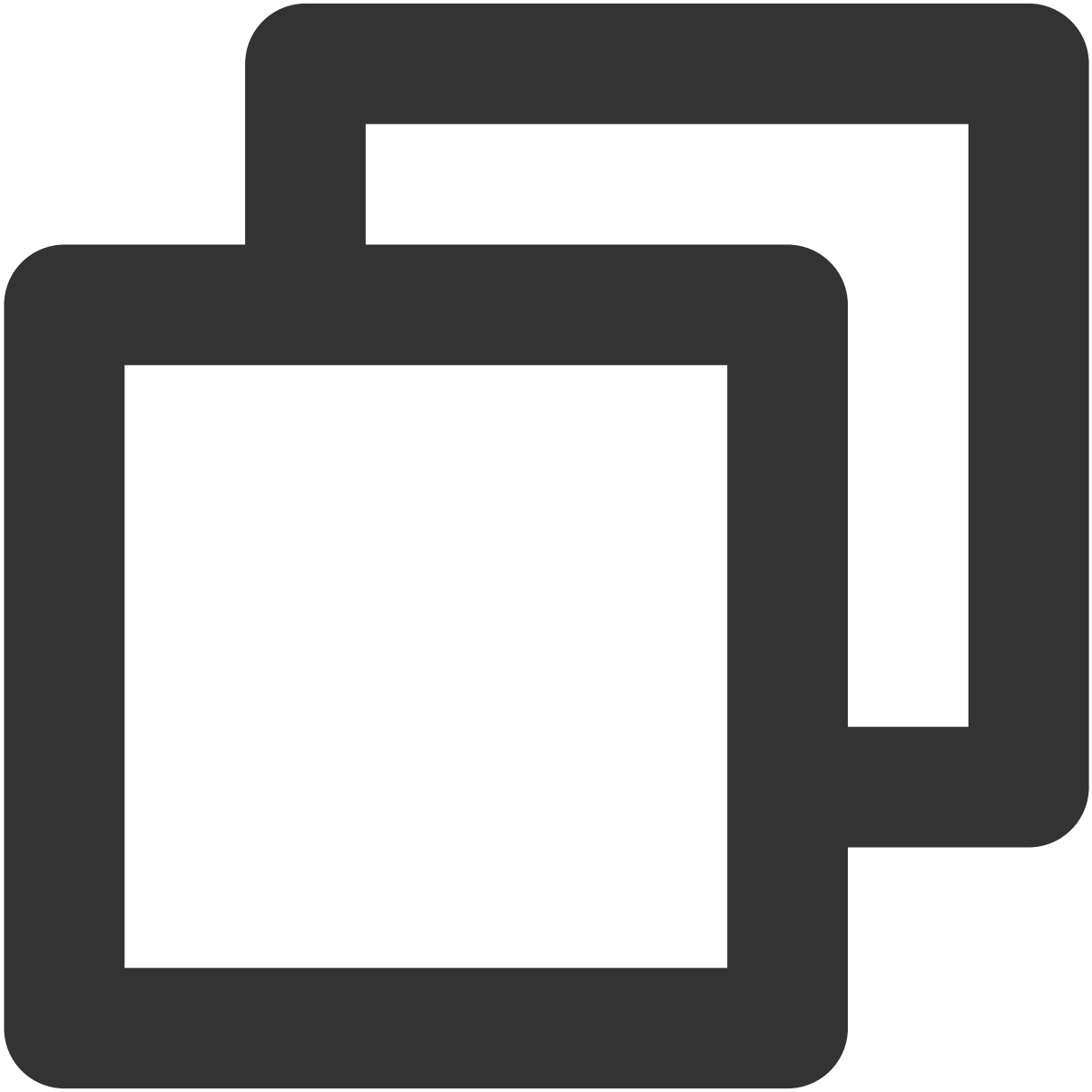
C++

C#



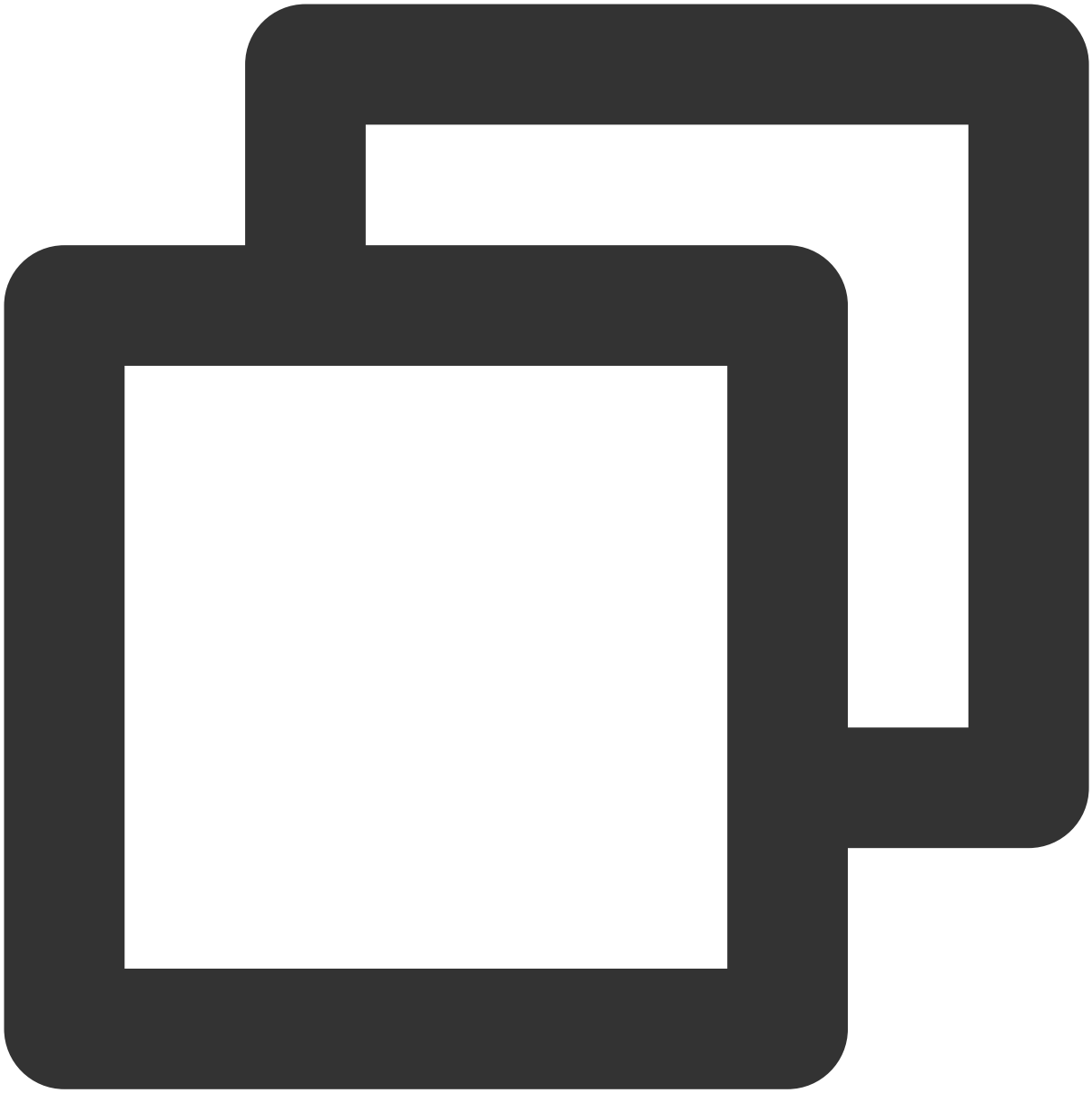
```
// 네트워크 속도 테스트 실행 샘플 코드. sdkAppId, UserSig 필요(획득 방법은 기본 기능 참고)
// 로그인 후 테스트 시작 예시
- (void)onLogin:(NSString *)userId userSig:(NSString *)userSid
{
    TRTCSpeedTestParams *params;
    // sdkAppID는 콘솔에서 획득한 실제 애플리케이션 AppID
    params.sdkAppID = sdkAppId;
    params.userID = userId;
    params.userSig = userSig;
    // 예상 업스트림 대역폭(kbps, 값 범위: 10 ~ 5000, 0일 때 테스트되지 않음)
    params.expectedUpBandwidth = 5000;
```

```
    // 예상 다운스트림 대역폭(kbps, 값 범위: 10 ~ 5000, 0일 때 테스트되지 않음)
    params.expectedDownBandwidth = 5000;
    [trtcCloud startSpeedTest:params];
}
- (void)onSpeedTestResult:(TRTCSpeedTestResult *)result {
    // 속도 측정이 완료된 후 속도 측정 결과 콜백
}
```



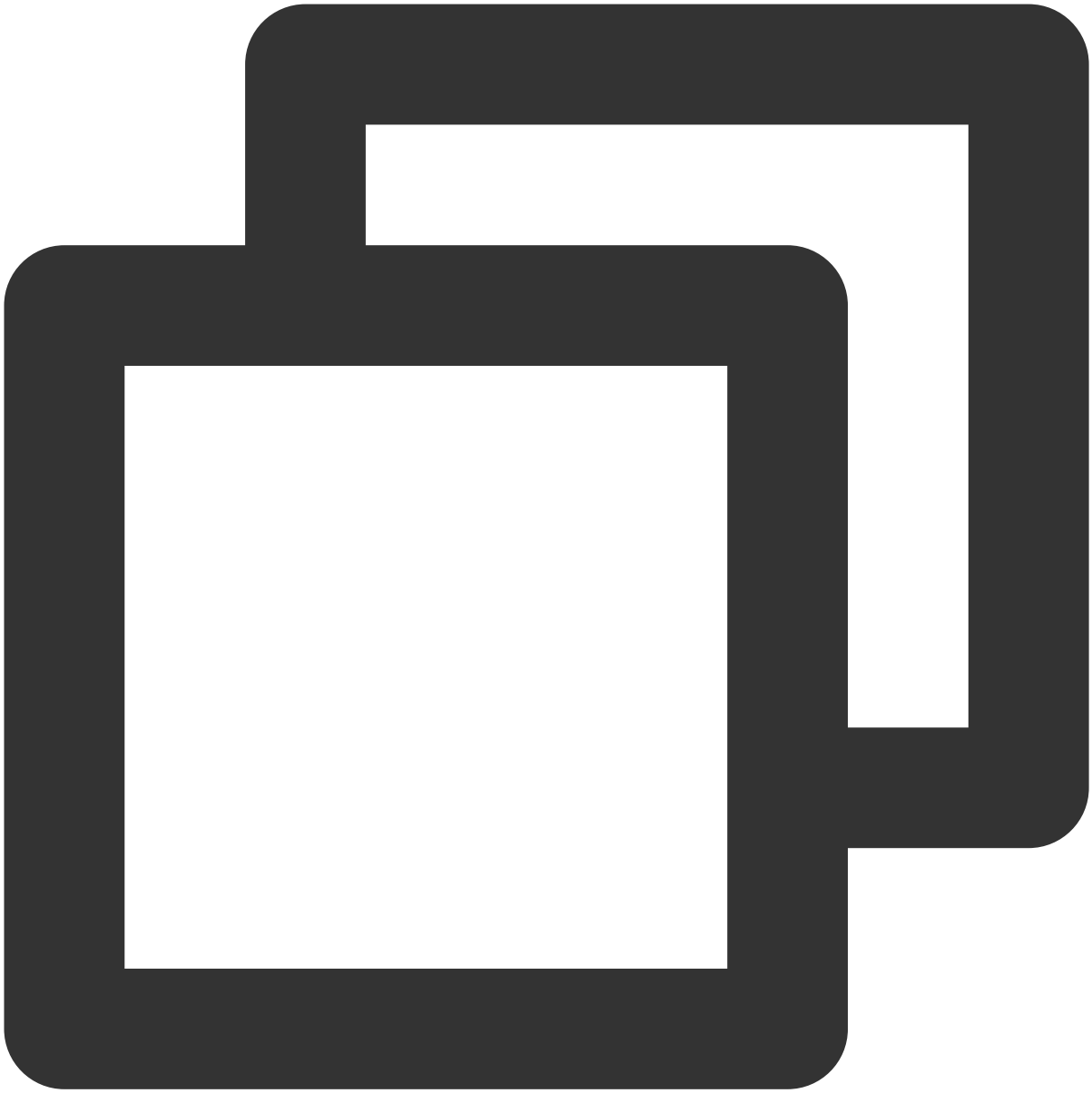
```
//네트워크 속도 테스트 실행 샘플 코드. sdkAppId, UserSig 필요(획득 방법은 기본 기능 참고)
// 로그인 후 테스트 시작 예시
```

```
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);
    // sdkAppID는 콘솔에서 획득한 실제 애플리케이션 AppID
    trtcCloud.startSpeedTest(params);
}

// 속도 테스트 결과 수신. TRTCCloudListener를 상속하여 다음 메소드 구현
void onSpeedTestResult(TRTCCloudDef.TRTCSpeedTestResult result)
{
    // 속도 측정이 완료된 후 속도 측정 결과 콜백
}
```



```
// 네트워크 속도 테스트 실행 샘플 코드. sdkAppId, UserSig 필요(획득 방법은 기본 기능 참고)
// 로그인 후 테스트 시작 예시
void onLogin(const char* userId, const char* userSig)
{
    TRTCSpeedTestParams params;
    // sdkAppID는 콘솔에서 획득한 실제 애플리케이션 AppID
    params.sdkAppID = sdkAppId;
    params.userId   = userid;
    param.userSig = userSig;
    // 예상 업스트림 대역폭(kbps, 값 범위: 10 ~ 5000, 0일 때 테스트되지 않음)
    param.expectedUpBandwidth = 5000;
```

```
    // 예상 다운스트림 대역폭(kbps, 값 범위: 10 ~ 5000, 0일 때 테스트되지 않음)
    param.expectedDownBandwidth = 5000;
    trtcCloud->startSpeedTest(params);
}

// 속도 테스트 결과 수신
void TRTCCloudCallbackImpl::onSpeedTestResult(
            const TRTCSpeedTestResult& result)
{
    // 속도 측정이 완료된 후 속도 측정 결과 콜백
}
```



```
// 네트워크 속도 테스트 실행 샘플 코드. sdkAppId, UserSig 필요(획득 방법은 기본 기능 참고).
// 로그인 후 테스트 시작 예시
private void onLogin(string userId, string userSig)
{
    TRTCSpeedTestParams params;
    // sdkAppID는 콘솔에서 획득한 실제 애플리케이션 AppID
    params.sdkAppID = sdkAppId;
    params.userId   = userid;
    param.userSig = userSig;
    // 예상 업스트림 대역폭(kbps, 값 범위: 10 ~ 5000, 0일 때 테스트되지 않음)
    param.expectedUpBandwidth = 5000;
```

```
    // 예상 다운스트림 대역폭(kbps, 값 범위: 10 ~ 5000, 0일 때 테스트되지 않음)
    param.expectedDownBandwidth = 5000;
    mTRTCCloud.startSpeedTest(params);
}

// 속도 테스트 결과 수신
public void onSpeedTestResult(TRTCSpeedTestResult result)
{
    // 속도 측정이 완료된 후 속도 측정 결과 콜백
}
```

속도 측정 툴

인터페이스를 호출하여 네트워크 속도를 측정하고 싶지 않은 경우, TRTC는 데스크톱에서 네트워크 속도 측정 툴 프로그램을 제공하여 상세한 네트워크 품질 정보를 빠르게 얻을 수 있도록 도와줍니다.

다운로드 링크

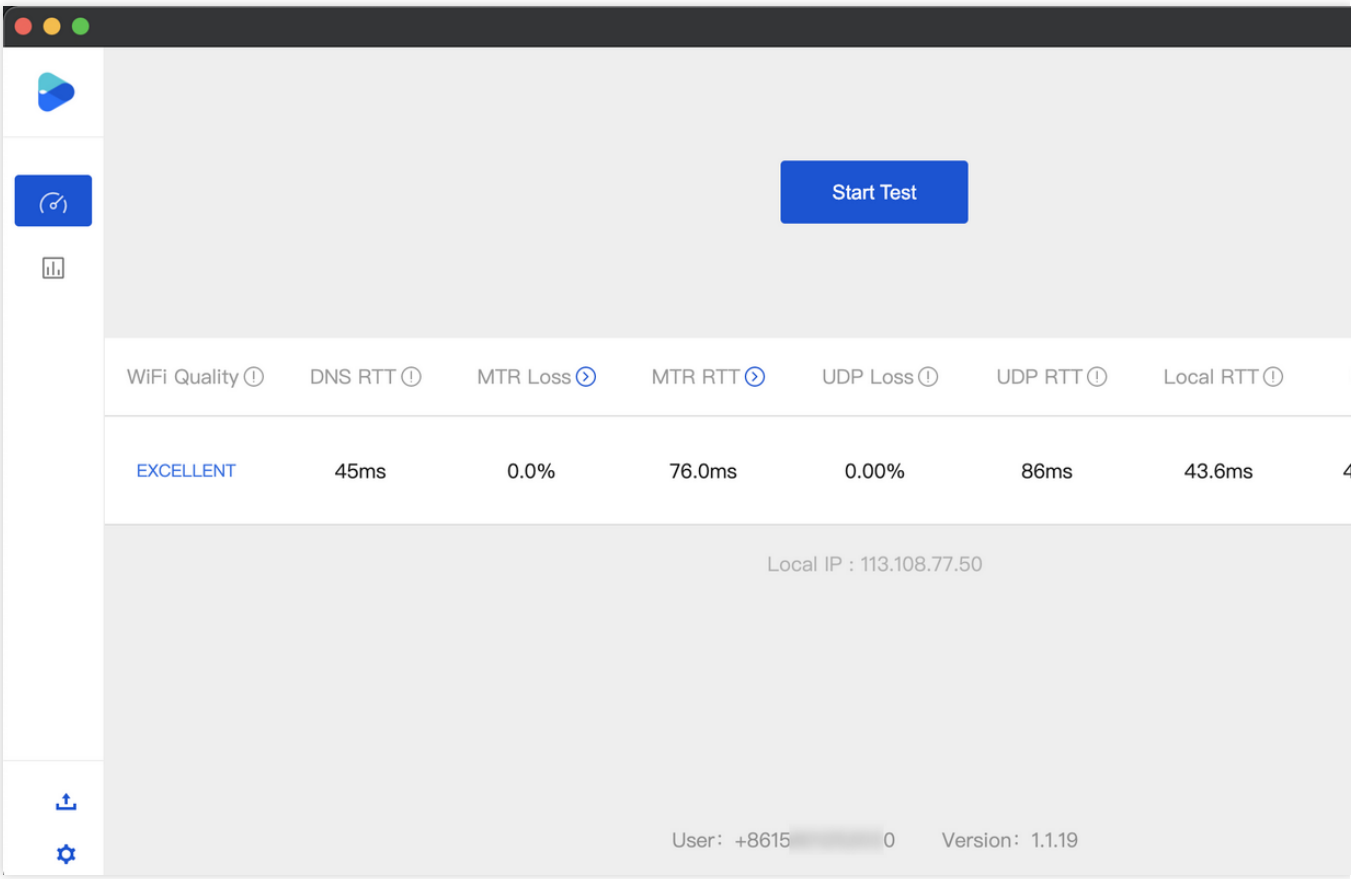
Mac | Windows

테스트 지표

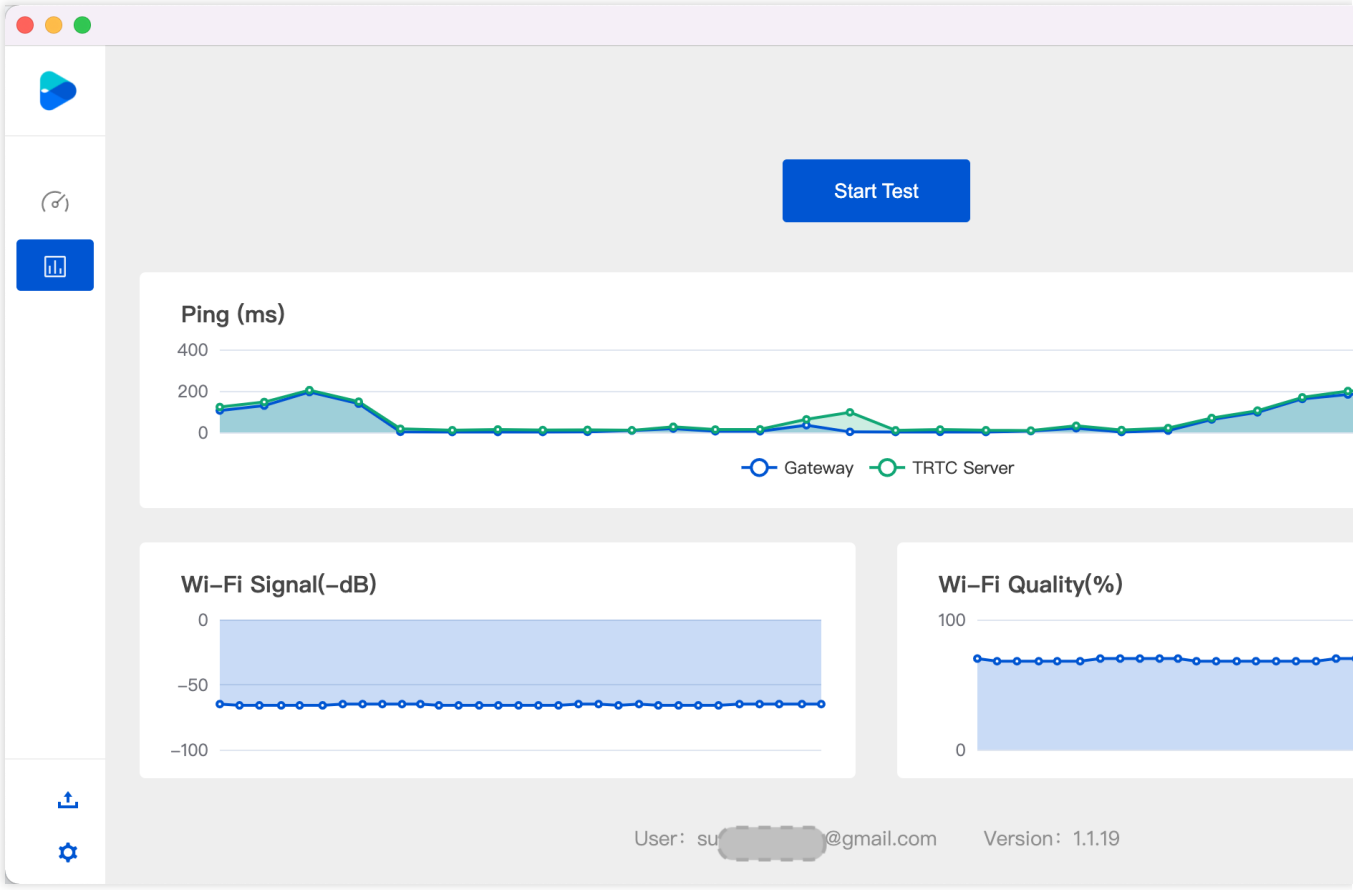
지표	의미
WiFi Quality	Wi-Fi 신호 품질
DNS RTT	Tencent Cloud의 속도 테스트 리졸브 소요 시간
MTR	MTR은 클라이언트에서 TRTC 노드까지의 패킷 손실률 및 지연을 감지할 수 있는 네트워크 테스트 툴이며 라우팅의 각 홉에 대한 특정 정보 확인 가능
UDP Loss	클라이언트에서 TRTC 노드로의 UDP 패킷 손실률
UDP RTT	클라이언트에서 TRTC 노드까지의 UDP 딜레이
Local RTT	클라이언트에서 로컬 게이트웨이로의 딜레이
Upload	업스트림 예상 대역폭
Download	다운스트림 예상 대역폭

툴 화면 캡처

빠른 테스트:



지속적인 테스트:



Web

최종 업데이트 날짜: : 2024-07-24 16:43:38

TRTC는 사용자가 방에 들어가거나 통화에 참여하기 전에 사용자의 네트워크 품질을 확인할 수 있습니다. 네트워크 품질이 만족스럽지 않은 경우 원활한 통화를 위해 더 나은 네트워크 환경으로 이동하도록 안내하는 것이 좋습니다. 본문에서는 NETWORK_QUALITY 이벤트를 기반으로 호출 전 네트워크 품질을 확인하는 방법을 설명합니다. 통화 중 네트워크 품질을 감지하려면NETWORK_QUALITY 이벤트를 수신하면 됩니다.

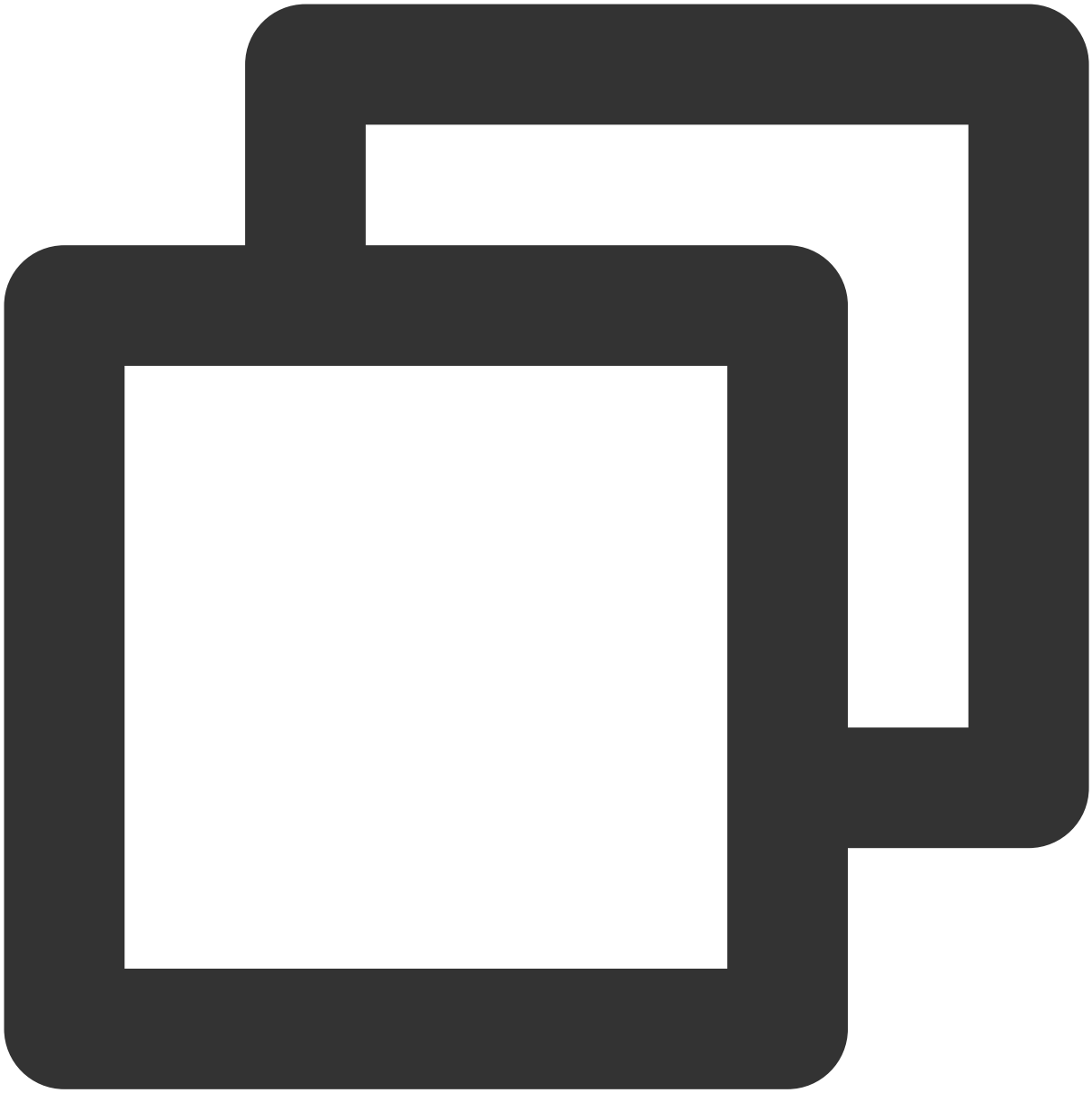
구현 프로세스

1. TRTC.createClient를 호출하여 uplinkClient 및 downlinkClient의 두 Client를 생성합니다.
2. 두 명의 Client가 같은 방에 들어가도록 합니다.
3. uplinkClient를 사용하여 스트림을 푸시합니다. NETWORK_QUALITY 이벤트를 듣고 업스트림 네트워크 품질을 확인하십시오.
4. downlinkClient를 사용하여 스트림을 가져옵니다. NETWORK_QUALITY 이벤트를 듣고 다운스트림 네트워크 품질을 확인하십시오.
5. 전체 프로세스는 약 15s가 소요되며 평균 네트워크 품질은 결국 업스트림 및 다운스트림 네트워크 품질을 평가하는 데 사용됩니다.

주의사항 :

네트워크 품질을 확인하는 과정에는 소정의 기본 서비스 요금이 발생합니다. 푸시 해상도가 지정되지 않은 경우 스트림은 기본적으로 640*480 해상도로 푸시됩니다.

코드 예시



```
let uplinkClient = null; // 업스트림 네트워크 품질 확인
let downlinkClient = null; // 다운스트림 네트워크 품질 확인
let localStream = null; // 테스트할 스트림
let testResult = {
  // 업스트림 네트워크 품질 데이터 기록
  uplinkNetworkQualities: [],
  // 다운스트림 네트워크 품질 데이터 기록
  downlinkNetworkQualities: [],
  average: {
    uplinkNetworkQuality: 0,
    downlinkNetworkQuality: 0
```

```
  }
}

// 1. 업스트림 네트워크 품질 확인
async function testUplinkNetworkQuality() {
  uplinkClient = TRTC.createClient({
    sdkAppId: 0, // sdkAppId 입력
    userId: 'user_uplink_test',
    userSig: '', // uplink_test의 userSig
    mode: 'rtc'
  });

  localStream = TRTC.createStream({ audio: true, video: true });
  // 필요에 따라 video profile 설정
  localStream.setVideoProfile('480p');
  await localStream.initialize();

  uplinkClient.on('network-quality', event => {
    const { uplinkNetworkQuality } = event;
    testResult.uplinkNetworkQualities.push(uplinkNetworkQuality);
  });

  // 테스트를 위한 방으로 들어갑니다. 충돌을 피하기 위해 회의실 ID는 무작위여야 합니다.
  await uplinkClient.join({ roomId: 8080 });
  await uplinkClient.publish(localStream);
}

// 2. 다운스트림 네트워크 품질 확인
async function testDownlinkNetworkQuality() {
  downlinkClient = TRTC.createClient({
    sdkAppId: 0, // sdkAppId 입력
    userId: 'user_downlink_test',
    userSig: '', // userSig
    mode: 'rtc'
  });

  downlinkClient.on('stream-added', async event => {
    await downlinkClient.subscribe(event.stream, { audio: true, video: true });
        // 성공적인 구독 후 네트워크 품질 이벤트 수신 시작
    downlinkClient.on('network-quality', event => {
      const { downlinkNetworkQuality } = event;
      testResult.downlinkNetworkQualities.push(downlinkNetworkQuality);
    });
  })
  // 테스트를 위한 방으로 들어갑니다. 충돌을 피하기 위해 회의실 ID는 무작위여야 합니다.
  await downlinkClient.join({ roomId: 8080 });
}
```

```
// 3. 점검 시작
testUplinkNetworkQuality();
testDownlinkNetworkQuality();

// 4. 15s 후 점검을 중지하고 평균 네트워크 품질 계산
setTimeout(() => {
  // 평균 업스트림 네트워크 품질 계산
  if (testResult.uplinkNetworkQualities.length > 0) {
    testResult.average.uplinkNetworkQuality = Math.ceil(
      testResult.uplinkNetworkQualities.reduce((value, current) => value + current,
    );
  }

  if (testResult.downlinkNetworkQualities.length > 0) {
    // 평균 다운스트림 네트워크 품질 계산
    testResult.average.downlinkNetworkQuality = Math.ceil(
      testResult.downlinkNetworkQualities.reduce((value, current) => value + curren
    );
  }

  // 점검이 종료됩니다. 관련 상태를 지우십시오.
  uplinkClient.leave();
  downlinkClient.leave();
  localStream.close();
}, 15 * 1000);
```

결과 분석

위의 단계를 수행한 후 평균 업스트림 및 다운스트림 네트워크 품질을 얻을 수 있습니다. 네트워크 품질의 열거 값은 다음과 같습니다.

값	설명
0	알 수 없는 네트워크 품질, 현재 client 인스턴스가 업스트림/다운스트림 연결을 설정하지 않았음을 나타냄
1	우수한 네트워크 품질
2	좋은 네트워크 품질
3	평균 네트워크 품질
4	열악한 네트워크 품질
5	매우 열악한 네트워크 품질

6	네트워크 연결 끊김. 참고: 다운스트림 네트워크 품질이 이 값이면 모든 다운스트림 연결이 끊겼음을 나타냄

네트워크 품질이 3보다 나쁠 경우 사용자에게 네트워크를 확인하고 더 나은 네트워크 환경으로 이동하라는 메시지를 표시하는 것이 좋습니다. 그렇지 않으면 정상적인 음성/영상 통화 환경이 영향을 받을 수 있습니다.
다음 방법을 사용하여 대역폭 소비를 줄일 수도 있습니다.
업스트림 네트워크 품질이 3보다 나쁠 경우 LocalStream.setVideoProfile() API를 호출하여 비트레이트를 줄이거나 또는 LocalStream.muteVideo()를 호출하여 업스트림 대역폭 소비를 줄이기 위해 비디오를 비활성화할 수 있습니다.
다운스트림 네트워크 품질이 3보다 낮은 경우 기본 스트림/서브 스트림 전송 활성화의 지침에 따라 낮은 품질의 스트림에 가입하거나 오디오만 가입하여 다운스트림 대역폭 소비를 줄일 수 있습니다.

TRTC 클라우드 녹화 설명

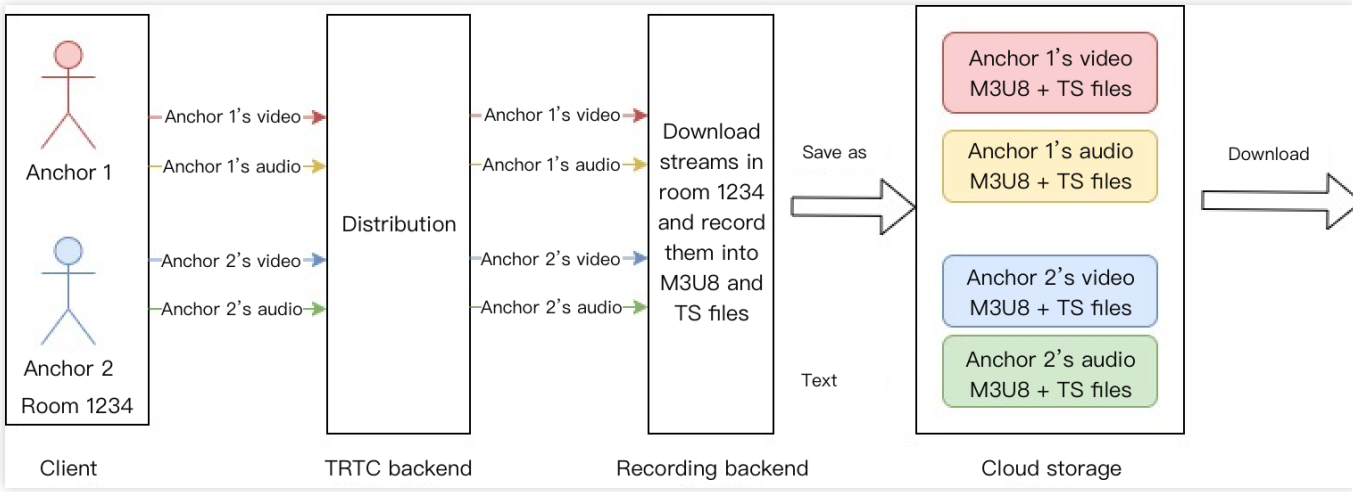
최종 업데이트 날짜: : 2024-07-24 16:43:38

온라인 교육, 라이브 쇼, 화상 회의, 온라인 의료, 원격 은행 등의 응용 시나리오에서 콘텐츠 심사, 비디오 보관 및 비디오 재생 등의 필요성 때문에 전체 영상 통화 또는 인터랙션 라이브 방송 프로세스를 녹화하고 저장해야 하는 경우가 많습니다. 이는 클라우드 녹화 기능을 통해 구현할 수 있습니다.

기능 개요

TRTC의 클라우드 녹화 기능을 통해 REST API 인터페이스를 호출하여 녹화할 멀티미디어 스트림을 구독하는 클라우드 녹화 작업을 실행하고 실시간으로 효율적인 제어를 진행할 수 있습니다. 또한 개발자가 직접 서버 및 녹화 관련 모듈을 배치할 필요가 없으므로 보다 편리합니다.
녹화 모드: 단일 스트림 녹화는 방에 있는 각 사용자의 멀티미디어 스트림을 독립된 파일로 녹화할 수 있습니다. 혼합 스트림 녹화는 같은 방의 멀티미디어 스트림을 하나의 파일로 믹싱하여 녹화합니다.
구독 스트림: 구독 사용자의 블록리스트/얼로우리스트 방식으로 구독할 사용자 미디어 스트림 지정 지원
트랜스 코딩 매개변수: 혼합 스트림 시나리오에서 코덱 매개변수를 설정하여 비디오 녹화 파일의 품질 지정 지원
혼합 스트림 매개변수: 혼합 스트림 시나리오에서 다양한 유연하고 가변적인 자동 다중 화면 레이아웃 템플릿 및 사용자 정의 레이아웃 템플릿 지원
CFS: 지정된 녹화 파일을 클라우드 스토리지/VOD로 저장 지원, 현재 클라우드 스토리지 벤더는 Tencent Cloud의 COS 스토리지 지원, VOD 벤더는 Tencent Cloud VOD 지원
(향후 다른 클라우드 벤더의 스토리지 및 VOD 서비스를 지원할 예정이며, 이 때 클라우드 서비스 계정이 필요하고, 클라우드 스토리지 서비스는 스토리지 매개변수를 제공해야 함)
콜백 공지: 콜백 공지 능력 지원, 콜백 도메인을 설정하여 클라우드 녹화의 이벤트 상태를 귀하의 콜백 서버로 공지

단일 스트림 녹화

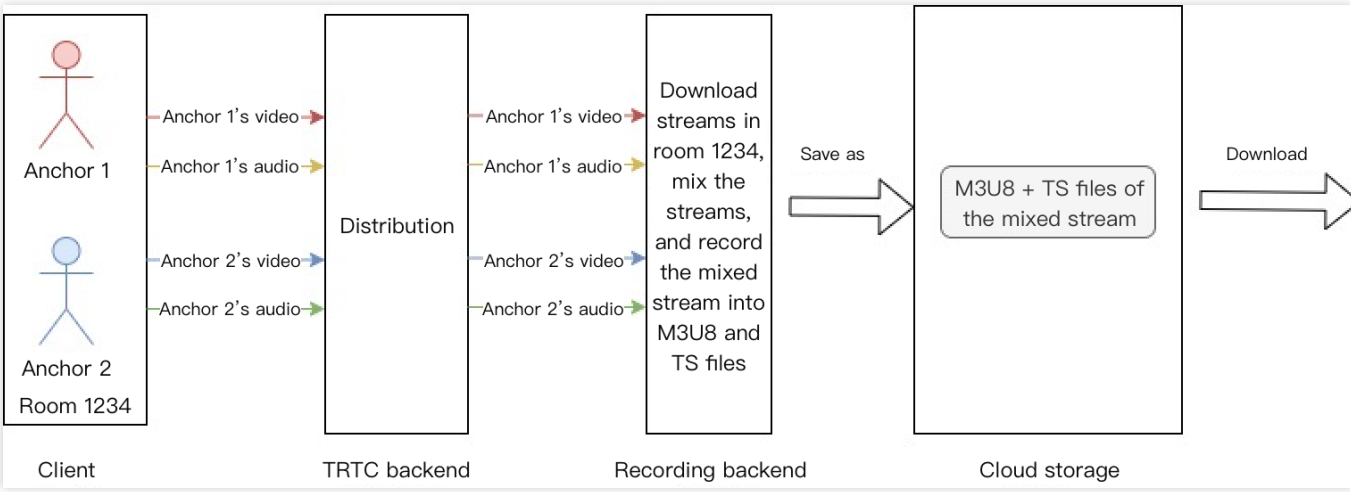


이미지는 단일 스트림 녹화 시나리오를 보여줍니다. 방 1234에서 호스트 1과 호스트 2 모두 멀티미디어를 업스트림했습니다. 호스트 1과 호스트 2의 멀티미디어 스트림을 구독하고 녹화 모드를 단일 스트림 녹화로 설정한다고 가정합니다. 녹화 백그라운드는 각각 호스트 1 및 호스트 2의 멀티미디어 스트림을 풀링해 다음을 포함한 독립 미디어 파일에 녹화합니다.

1. 호스트 1의 비디오 M3U8 인덱스 파일;
2. 호스트 1의 여러 비디오 TS 슬라이스 파일;
3. 호스트 1의 오디오 M3U8 인덱스 파일;
4. 호스트 1의 여러 오디오 TS 슬라이스 파일;
5. 호스트 2의 비디오 M3U8 인덱스 파일;
6. 호스트 2의 여러 비디오 TS 슬라이스 파일;
7. 호스트 2의 오디오 M3U8 인덱스 파일;
8. 호스트 2의 여러 오디오 TS 슬라이스 파일;

녹화 백그라운드는 이 파일을 지정한 클라우드 스토리지 서버에 업로드합니다. 귀하의 업무 백그라운드는 이러한 녹화 파일을 풀링하고 업무 요구에 따라 이러한 파일을 병합 트랜스 코딩해야 합니다. 멀티미디어 병합 트랜스 코딩 스크립트를 제공합니다.

혼합 스트림 녹화



이미지는 혼합 스트림 녹화 시나리오를 보여줍니다. 방 1234에서 호스트 1과 호스트 2 모두 멀티미디어를 업스트림했습니다. 호스트 1과 호스트 2의 멀티미디어 스트림을 구독하고 설정한다고 가정합니다. 녹화 모드를 혼합 스트림 녹화로 설정하면 녹화 백그라운드는 각각 호스트 1 및 호스트 2의 멀티미디어 스트림을 풀링하고 설정한 다중 화면 템플릿에 따라 비디오 스트림을 혼합하고 오디오 스트림을 혼합합니다. 마지막으로 미디어 스트림은 다음을 포함하여 하나의 미디어 파일로 혼합됩니다.

1. 혼합 스트림 후 비디오 M3U8 인덱스 파일;
2. 혼합 스트림 후 여러 비디오 TS 슬라이스 파일;

녹화 백그라운드는 이 파일을 지정한 클라우드 스토리지 서버에 업로드합니다. 귀하의 업무 백그라운드는 이러한 녹화 파일을 풀링하고 업무 요구에 따라 이러한 파일을 병합 트랜스 코딩해야 합니다. 멀티미디어 병합 트랜스 코딩 스크립트를 제공합니다.

주의사항 :

녹화 인터페이스의 호출 주파수: 20qps로 제한
단일 인터페이스 시간 초과: 6초
기본 동시 녹화는 100개 채널을 지원합니다. 더 많은 채널이 필요한 경우 티켓 제출을 통해 문의해주십시오.
단일 녹화 작업은 동시에 구독할 수 있는 단일 방에서 최대 25개의 멀티미디어 스트림을 지원합니다.

호출 프로세스

1. 녹화 실행

백그라운드 서비스를 통해 REST API(CreateCloudRecording)를 호출하여 클라우드 녹화를 실행하며 주의해야 할 매개변수는 다음과 같습니다.

작업 ID(TaskId)

이 매개변수는 녹화 작업의 고유 식별자입니다. 녹화 작업 인터페이스에서 후속 작업을 위해 작업 ID를 입력 매개변수로 저장해야 합니다.

녹화 모드(RecordMode)

단일 스트림 녹화, 구독한 호스트의 오디오 및 비디오 파일을 방에 별도로 녹화하고 녹화된 파일(M3U8/TS)을 클라우드 스토리지에 업로드합니다.
혼합 스트림 녹화, 방에서 구독하는 모든 호스트의 멀티미디어 스트림을 하나의 멀티미디어 파일로 혼합하고 녹화 파일 [M3U8/TS]를 클라우드 스토리지에 업로드합니다.

사용자 블록리스트/얼로우리스트 녹화(SubscribeStreamUserIds)

기본적으로 클라우드 녹화는 방의 모든 미디어 스트림(최대 25개 채널)을 구독합니다. 이 매개변수를 통해 호스트 사용자의 블록리스트/얼로우리스트 구독을 지정할 수도 있으며, 물론 녹화 중 업데이트 작업도 지원합니다.

클라우드 스토리지 매개변수(StorageParams)

클라우드 스토리지 매개변수를 지정하여 귀하가 활성화한 지정된 클라우드 스토리지/VOD 서비스에 녹화된 파일을 업로드합니다. 클라우드 스토리지/VOD 공간 매개변수의 유효성에 주의하고 연체되지 않도록 주의하십시오. 녹화 파일 이름은 고정 규칙이 있습니다.

녹화 파일명 이름 생성 규칙

단일 스트림 녹화 M3U8 파일 이름 규칙:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>.m3u8
단일 스트림 녹화 TS 파일 이름 규칙:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>_<UTC>.ts
단일 스트림 녹화 mp4 파일 이름 규칙:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Index>.mp4
혼합 스트림 녹화 M3U8 파일 이름 규칙:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>.m3u8
혼합 스트림 녹화 TS 파일 이름 규칙:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>_<UTC>.ts
혼합 스트림 녹화 mp4 파일 이름 규칙:
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>_<Index>.mp4
고가용성 풀업 후 파일 이름 규칙
데이터 센터에서 클라우드 녹화 서비스 장애 시 고가용성 솔루션을 통해 녹화 작업을 복구합니다. 이 경우 원본 녹화 파일을 덮어쓰기하지 않기 위해 풀업 후 접두사 ha<1/2/3>를 추가하여 발생한 고가용성의 횟수를 나타냅니다. 녹화 작업에 허용되는 최대 풀업 횟수는 3회입니다.
단일 스트림 녹화 M3U8 파일 이름 규칙:
<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>.m3u8
단일 스트림 녹화 TS 파일 이름 규칙:
<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>_<UTC>.ts

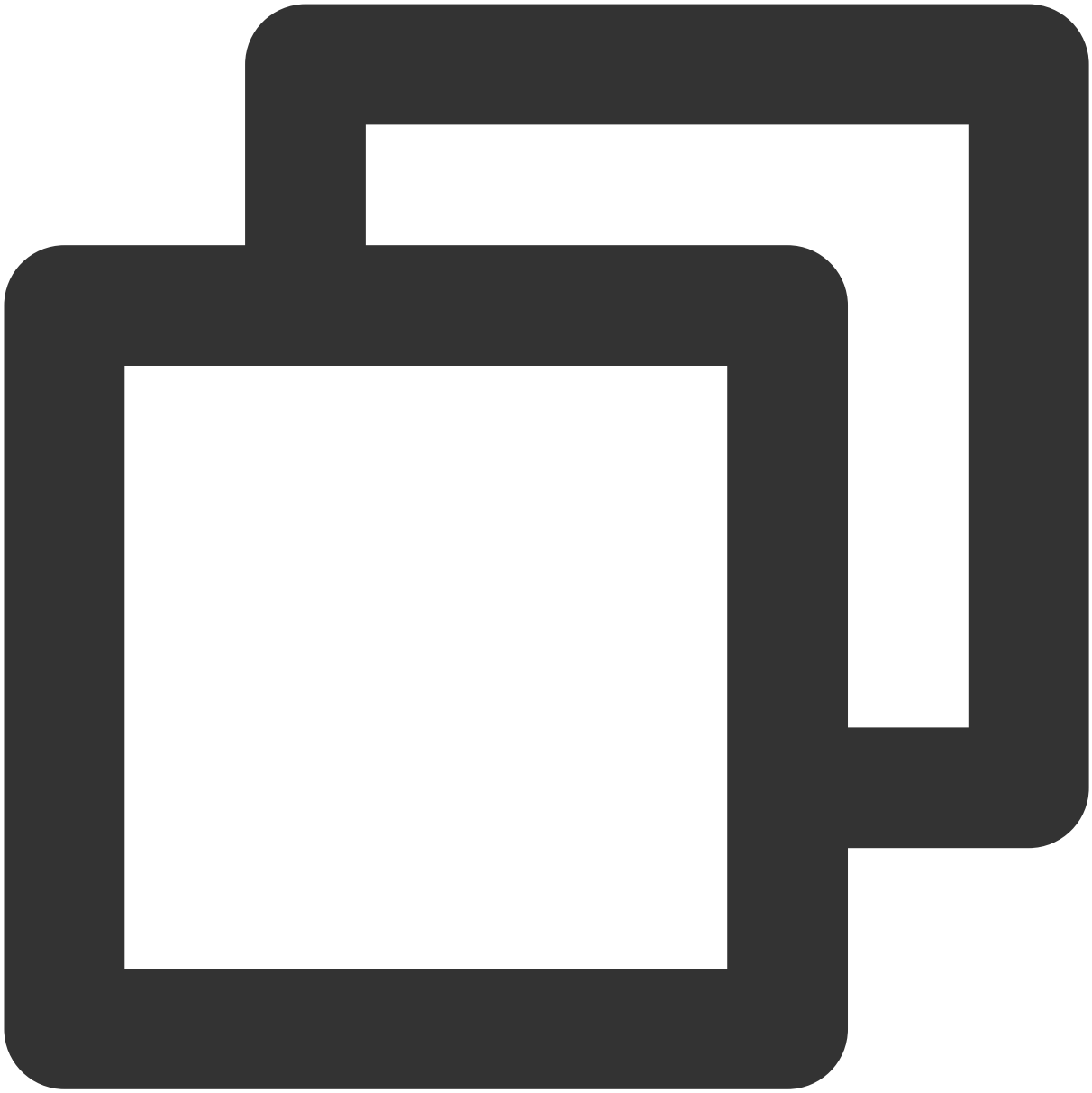
혼합 스트림 녹화 M3U8 파일 이름 규칙:
<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>.m3u8
혼합 스트림 녹화 TS 파일 이름 규칙:
<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>_<UTC>.ts

필드 의미 설명:

<Prefix>: 녹화 매개변수에 설정된 파일 이름 접두사는 설정되지 않은 경우 존재하지 않습니다.
<TaskId>: 녹화된 작업 ID는 전역 고유하며 녹화 실행 후 반환된 매개변수에 있습니다.
<SdkAppId>: 녹화 작업의 SdkAppId;
<RoomId>: 녹화 방 번호;
<UserId>: 특수 base64 처리 후 녹화 스트림의 사용자 ID, 아래 주의사항 참고;
<MediaId>: 메인/서브스트림 식별, main 또는 aux
<Type>: 녹화 스트림 유형, audio 또는 video;
<UTC>: 파일이 시작된 UTC 녹화 시간, 시간대 UTC+0, 년, 월, 일, 시, 분, 초 및 밀리초로 구성;
<Index>: mp4 슬라이스 로직이 트리거되지 않으면(크기가 2GB를 초과하거나 지속 시간이 24시간을 초과하는 경우) 이 필드가 없습니다. 그렇지 않으면 슬라이스의 인덱스 번호가 1부터 증가;
ha<1/2/3>: 고가용성 풀업 접두사, 예를 들어 첫 번째 풀업은<Prefix>/<TaskId>/ha1_<SdkAppId>_<RoomId>.m3u8로 변환
주의사항 :
<RoomId>가 문자열 방 ID인 경우 먼저 방 ID에 대해 base64 작업을 수행하고 base64 다음 문자열에서 '/' 기호를 '-'(하이픈)으로 변경하고, 기호 '='를 '.'로 변경;
녹화 스트림의 <UserId>는 먼저 base64 작업을 수행하고 base64 다음 문자열에서 '/' 기호를 '-'(하이픈)으로 변경하고 '=' 기호를 '.'로 변경합니다.

녹화 시작 시간 가져오기

녹화가 시작되는 시간은 서버가 호스트로부터 처음으로 멀티미디어 데이터를 수신하고 첫 번째 파일 녹화를 시작하는 unix 시간으로 정의됩니다.
녹화 시작 타임스탬프를 가져오는 3가지 방법:
녹화 파일 인터페이스(DescribeCloudRecording)에서 BeginTimeStamp 필드 쿼리
예를 들어 다음 쿼리 인터페이스에서 반환된 정보는 BeginTimeStamp가 1622186279144ms임을 보여줍니다.

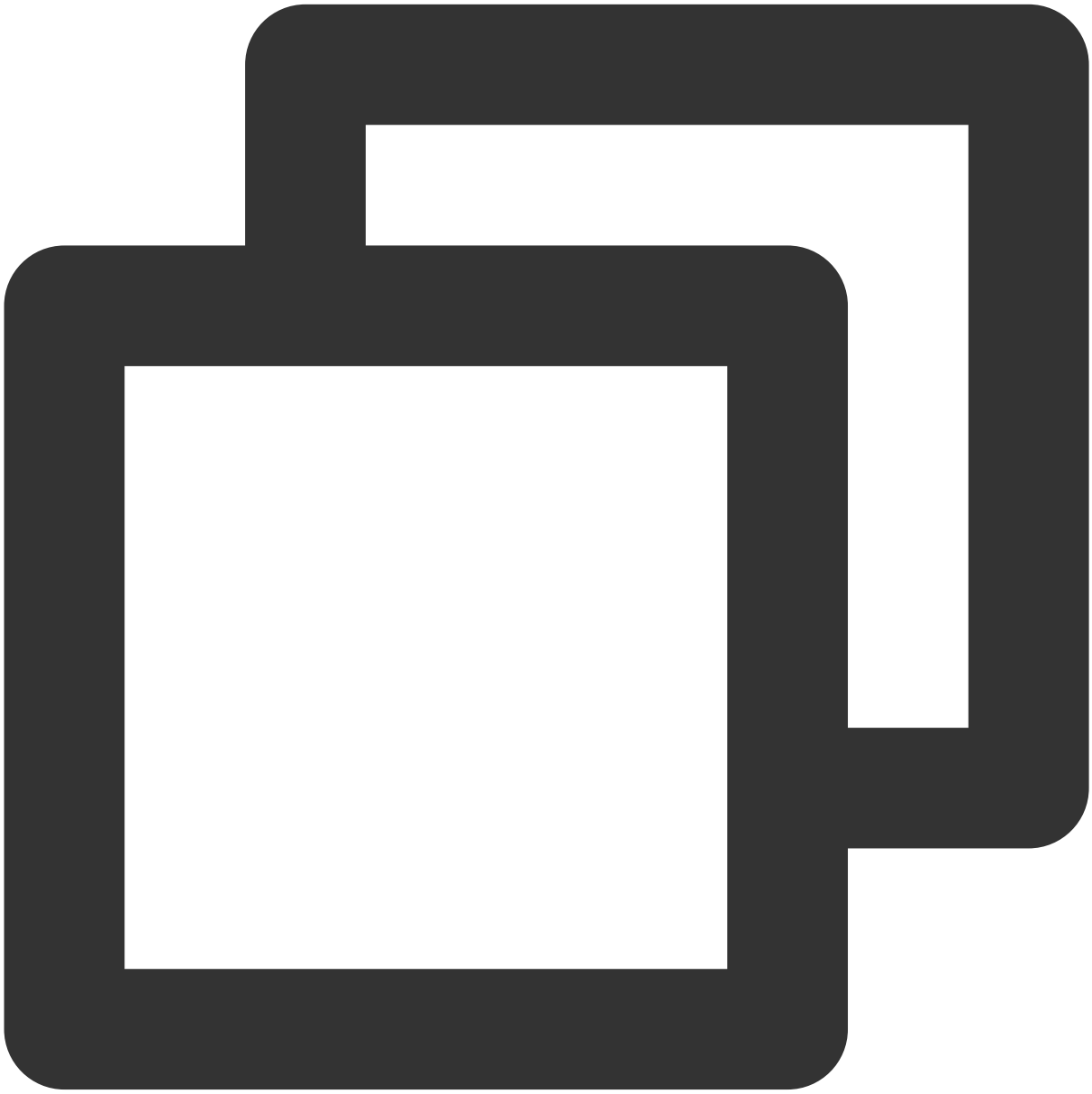


```
{
  "Response":{
    "Status": "xx",
    "StorageFileList": [
      {
        "TrackType": "xx",
        "BeginTimeStamp": 1622186279144,
        "UserId": "xx",
        "FileName": "xx"
      }
    ],
```

```
    "RequestId": "xx",
    "TaskId": "xx"
  }
}
```

M3U8 파일에서 해당 태그 항목 읽기(#EXT-X-TRTC-START-REC-TIME)

예를 들어 다음 M3U8 파일은 녹화 시작의 unix 타임스탬프가 1622425551884ms임을 표시합니다.



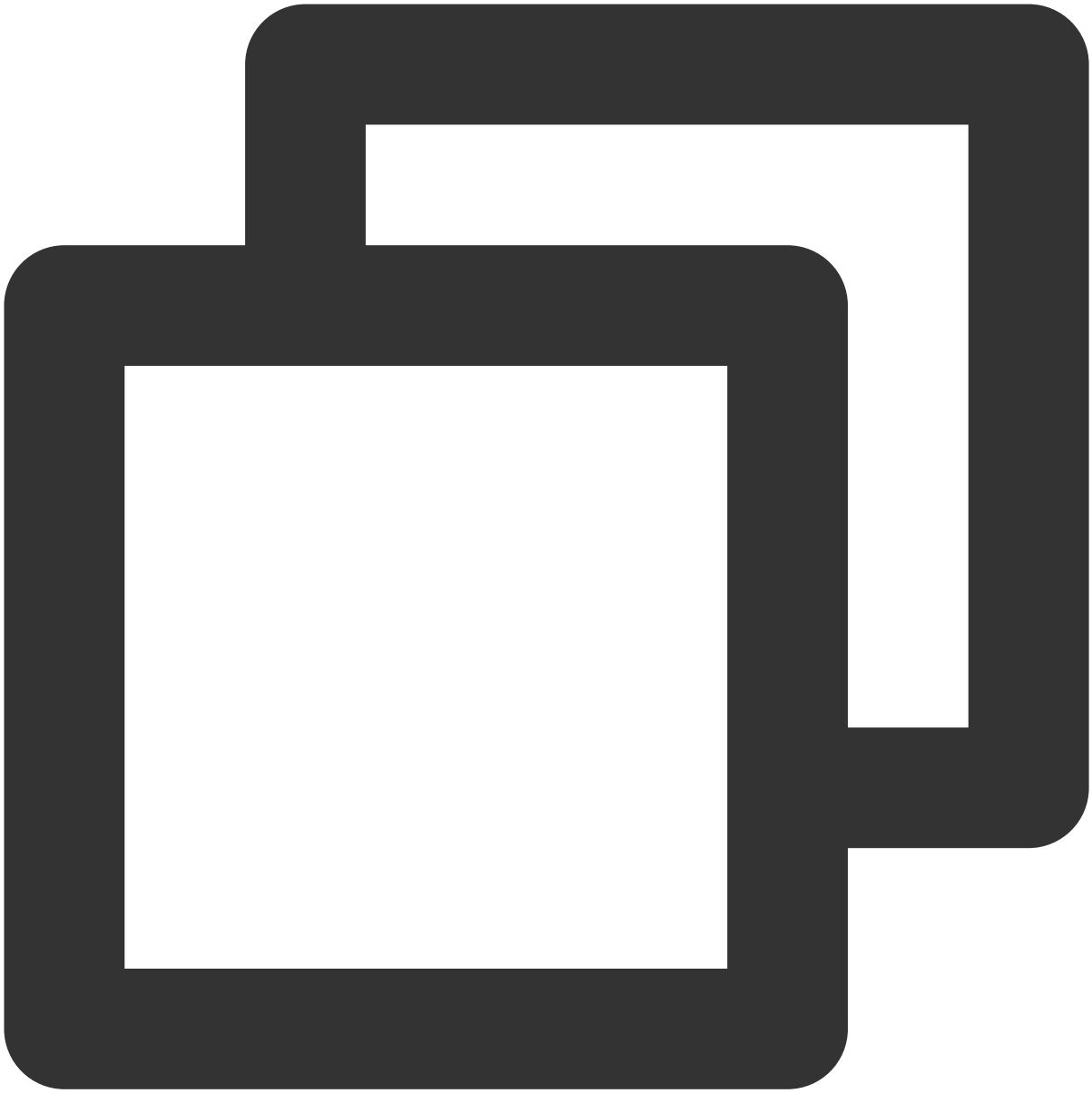
```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:NO
```

```
#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
```

녹화 콜백 이벤트 수신

콜백 구독을 통해 이벤트 유형 307의 BeginTimeStamp 필드에서 녹화 파일에 해당하는 녹화 시작 타임스탬프 가져오기

예를 들어 다음 콜백 이벤트에서 이 파일의 녹화 시작에 대한 unix 타임스탬프가 1622186279144ms임을 읽을 수 있습니다.



```
{
        "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
                }
        }
}
```

혼합 스트림 녹화 워터마크 매개변수(MixWatermark)

혼합 스트림 녹화에서 이미지 추가 워터마크를 지원하며 최대 개수는 25개이며 워터마크는 캔버스의 어느 곳에나 추가할 수 있습니다.

필드 이름	설명
Top	왼쪽 상단 모서리에 대한 워터마크의 수직 변위
Left	왼쪽 상단 모서리에 대한 워터마크의 수평 변위
Width	워터마크의 너비
Height	워터마크의 높이
url	워터마크 파일의 저장 url

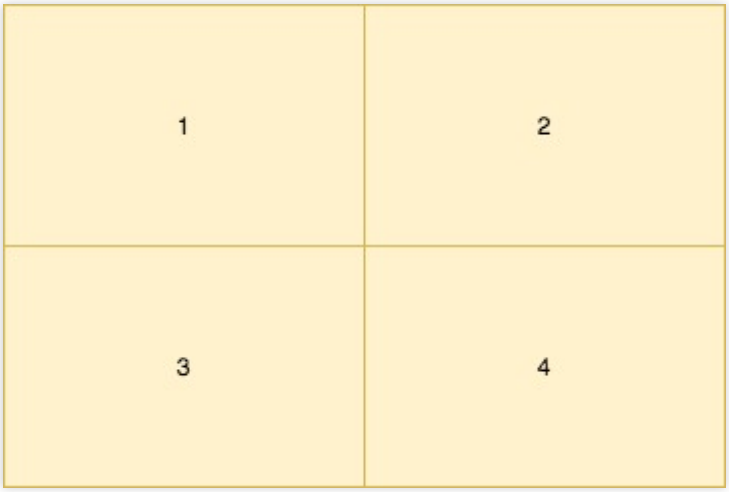
혼합 스트림 녹화 레이아웃 모드 매개변수(MixLayoutMode)

3x3 그리드 레이아웃(기본값), 플로팅 레이아웃, 화면 공유 레이아웃 및 사용자 정의 레이아웃 4가지 레이아웃을 지원합니다.

3x3 그리드 레이아웃:

호스트 수에 따라 각 화면의 크기가 자동으로 조절되며, 각 호스트의 화면 크기는 동일하며 최대 25개의 화면을 지원합니다.
1개의 서브 화면이 있는 경우:
각 작은 화면의 너비와 높이는 각각 전체 캔버스 너비와 높이;
2개의 서브 화면이 있는 경우:
각 작은 화면의 너비는 전체 캔버스 너비의 1/2;
각 작은 화면의 높이는 전체 캔버스 높이;
4개 이하의 서브 화면이 있는 경우:
각 작은 화면의 너비와 높이는 각각 전체 캔버스 너비와 높이의 1/2;
9개 이하의 서브 화면이 있는 경우:
각 작은 화면의 너비와 높이는 각각 전체 캔버스 너비와 높이의 1/3;
16개 이하의 서브 화면이 있는 경우:
각 작은 화면의 너비와 높이는 각각 전체 캔버스 너비와 높이의 1/4;

16개 이상의 서브 화면이 있는 경우:
각 작은 화면의 너비와 높이는 각각 전체 캔버스 너비와 높이의 1/5;
다음 이미지와 같이 구독 서브 화면이 증가함에 따라 3x3 그리드의 레이아웃 변경:



1	2	3
4	5	6
7	8	9

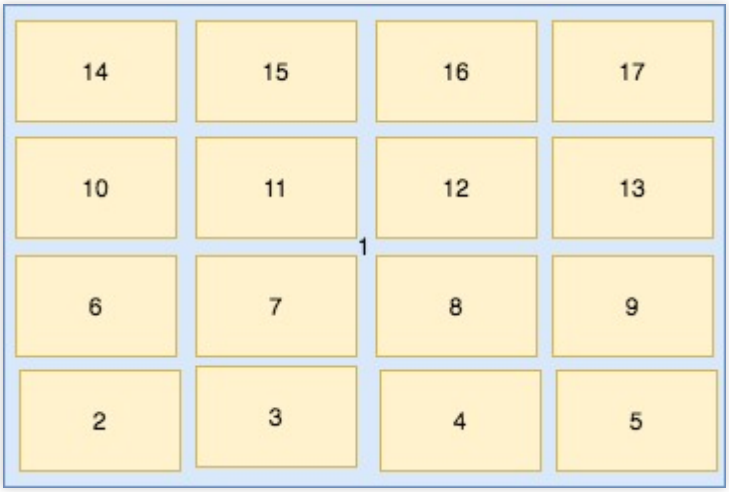
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

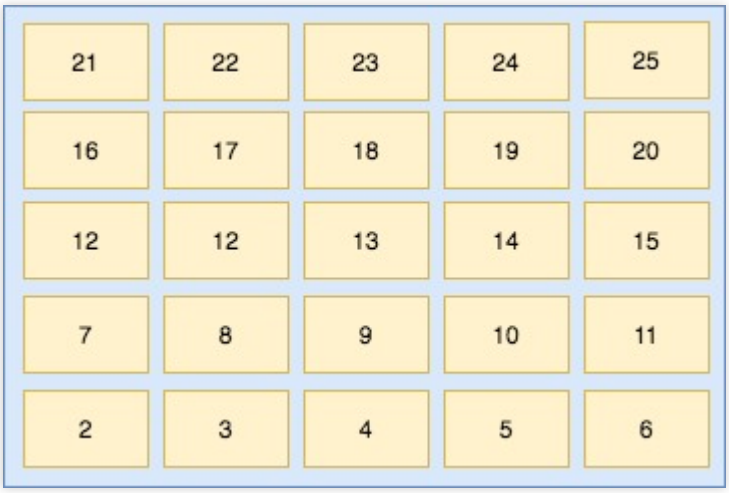
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

플로팅 레이아웃:

기본적으로 방에 입장하는 첫 번째 호스트(호스트를 지정 가능)의 영상 화면이 전체 화면을 채웁니다. 다른 호스트들의 영상 화면은 왼쪽 하단 모서리부터 방에 들어가는 순서대로 가로로 배열되어 작은 화면으로 표시되고 작은 화면은

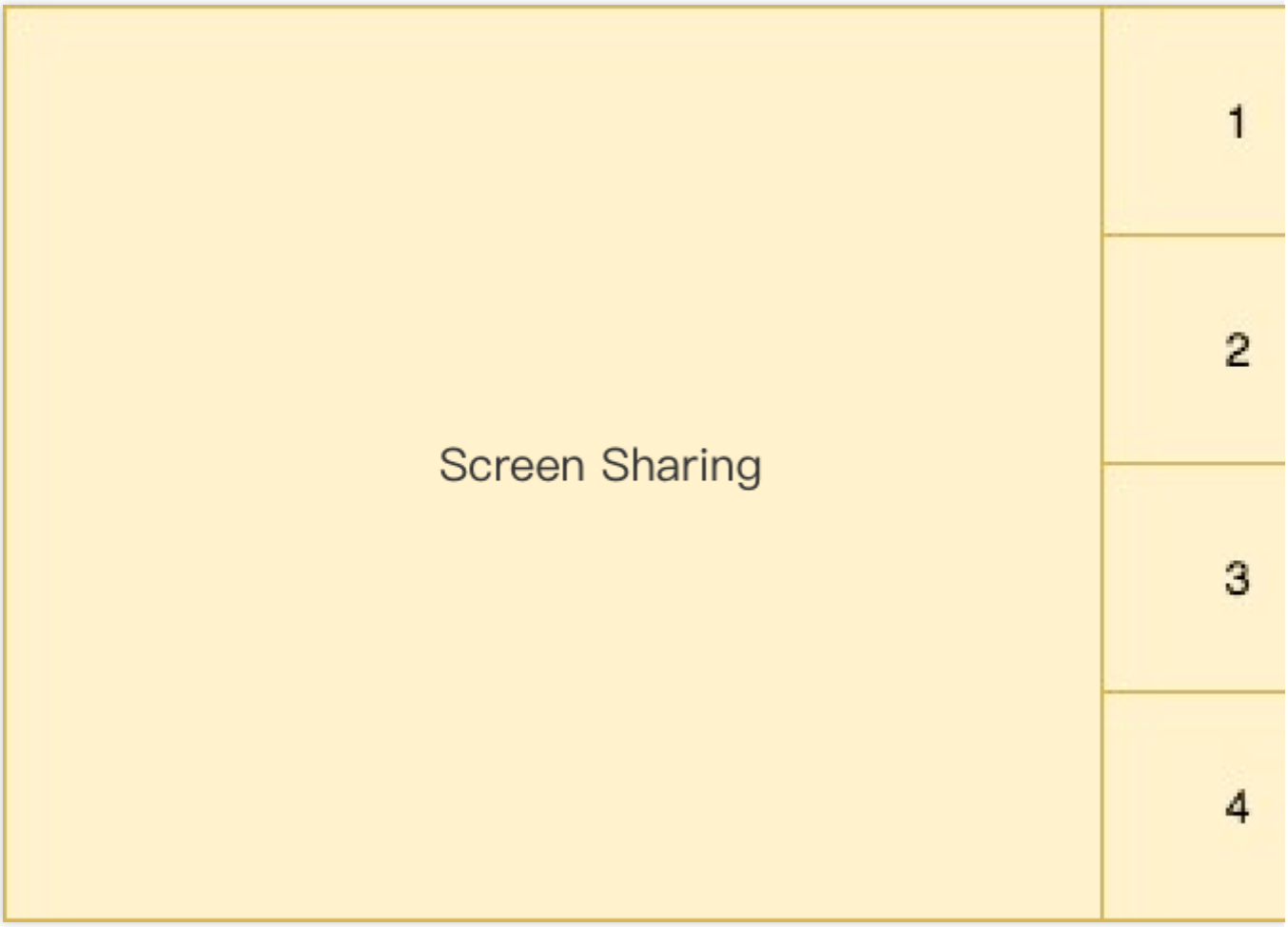
큰 화면 위에 플로팅됩니다. 화면 수가 17개 이하일 경우 1줄에 4개(4 x 4 배열). 화면 개수가 17개 이상일 경우 작은 화면을 한 줄에 5개(5 x 5)로 다시 레이아웃합니다. 최대 25개의 화면을 지원하며 사용자가 오디오만 보내는 경우 여전히 사진 위치를 차지합니다.
17개 이하의 서브 화면이 있는 경우:
각 작은 화면의 너비와 높이는 각각 전체 캔버스 너비와 높이의 0.235;
인접한 작은 화면의 왼쪽과 오른쪽, 위쪽과 아래쪽 거리는 각각 전체 캔버스 너비와 높이의 0.012;
캔버스에서 작은 화면의 가로 및 세로 여백도 전체 캔버스 너비와 높이의 0.012;
17개 이상의 서브 화면이 있는 경우:
각 작은 화면의 너비와 높이는 각각 전체 캔버스 너비와 높이의 0.188;
인접한 작은 화면의 왼쪽과 오른쪽, 위쪽과 아래쪽 거리는 각각 전체 캔버스 너비와 높이의 0.01;
캔버스에서 작은 화면의 가로 및 세로 여백도 전체 캔버스 너비와 높이의 0.01;
플로팅 레이아웃은 다음 이미지와 같이 구독된 서브 화면이 증가함에 따라 변경:

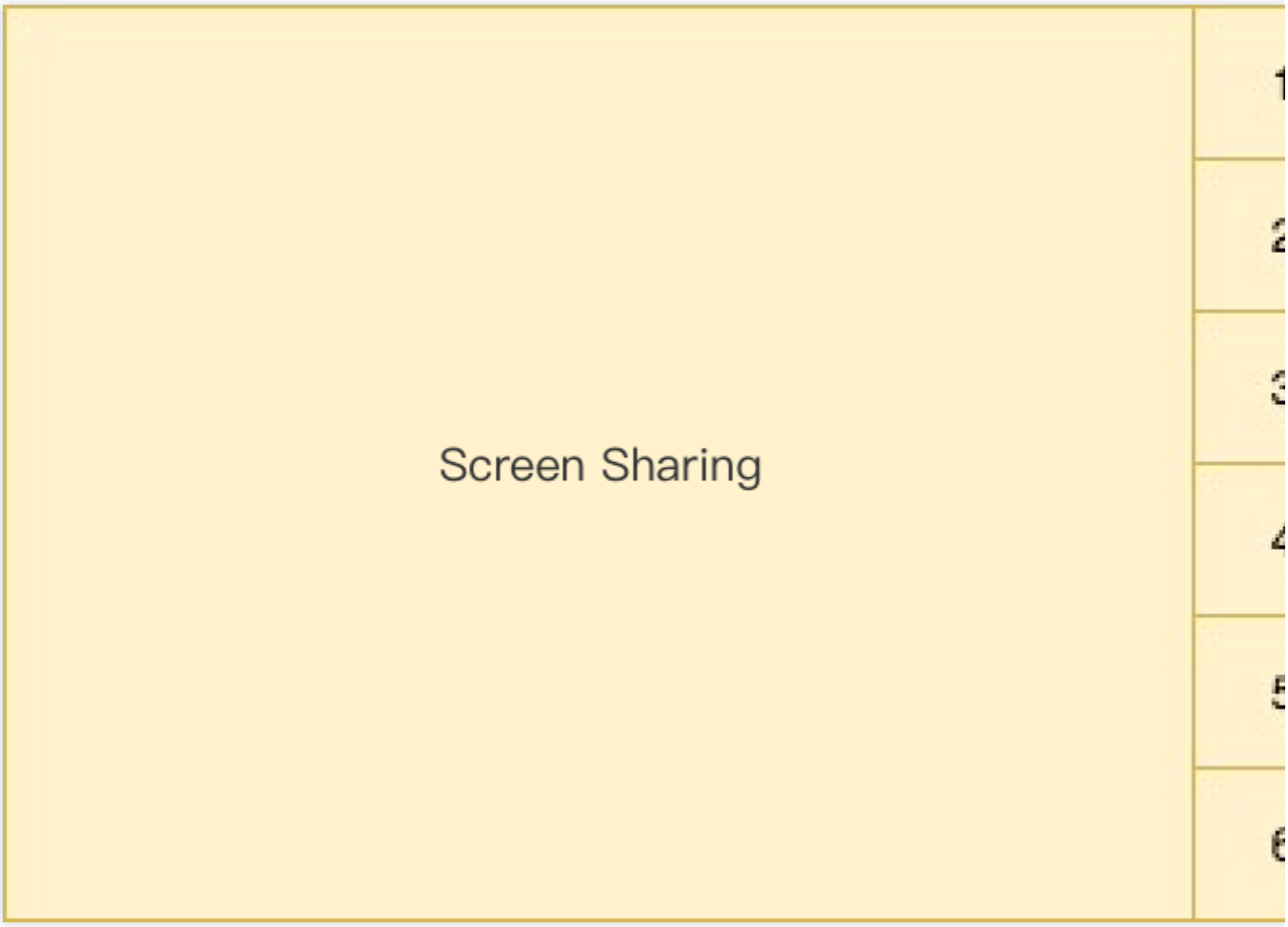


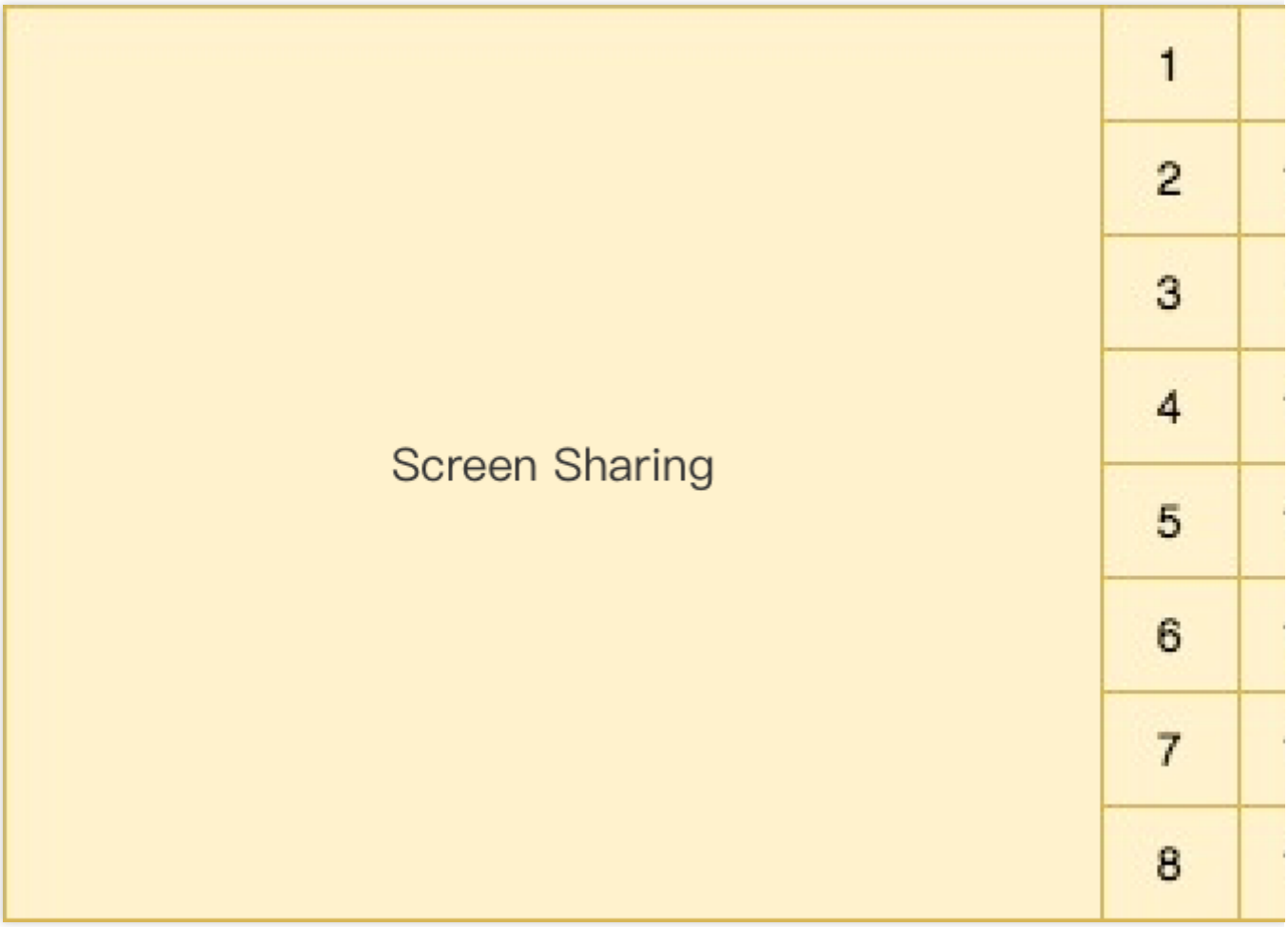


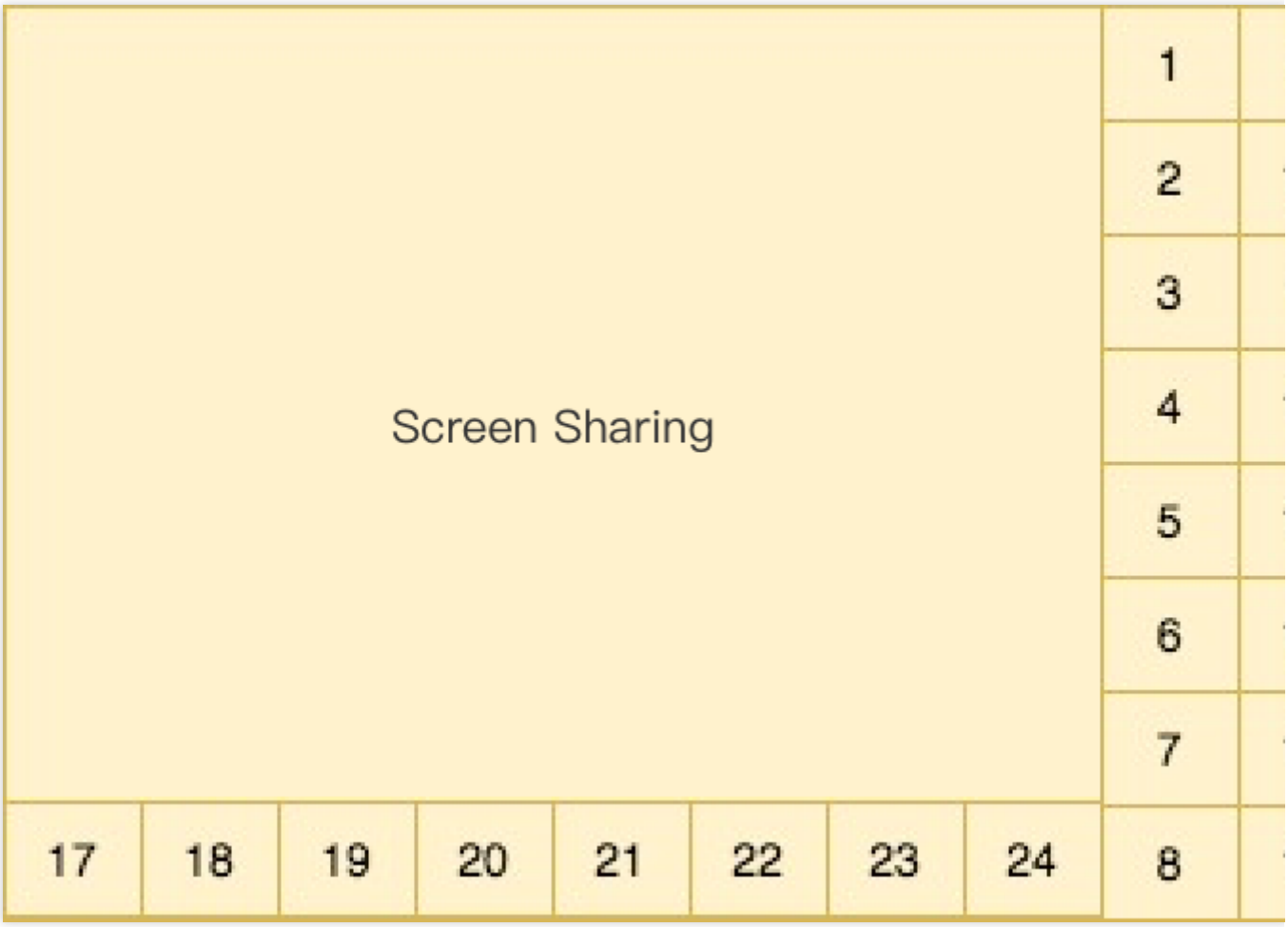
화면 공유 레이아웃:

화면 왼쪽에 있는 호스트의 큰 화면 위치를 지정하고(지정하지 않으면 큰 화면 위치가 배경색이 됨) 다른 호스트는 오른쪽에서 위에서 아래로 세로로 정렬됩니다. 화면 수가 17개 미만인 경우 오른쪽의 각 컬럼은 최대 8명까지 가능하며, 최대 2개의 컬럼을 차지합니다. 화면 개수가 17개 이상일 경우, 17개 이상의 화면이 있는 호스트는 왼쪽 하단 모서리부터 수평으로 정렬됩니다. 최대 24개의 화면을 지원하며 호스트가 오디오만 보내는 경우에도 화면 위치를 차지합니다.
5개 이하의 서브 화면이 있는 경우:
오른쪽 작은 화면의 너비는 전체 캔버스 너비의 1/5이고 오른쪽 작은 화면의 높이는 전체 캔버스 높이의 1/4;
왼쪽의 큰 화면의 너비는 전체 캔버스 너비의 4/5이고 왼쪽의 큰 화면의 높이는 전체 캔버스의 높이;
5보다 크고 7보다 작은 서브 화면이 있는 경우:
오른쪽 작은 화면의 너비는 전체 캔버스 너비의 1/7이고 오른쪽 작은 화면의 높이는 전체 캔버스 높이의 1/6;
왼쪽의 큰 화면의 너비는 전체 캔버스 너비의 6/7이고 왼쪽의 큰 화면의 높이는 전체 캔버스의 높이;
7보다 크고 9보다 작은 서브 화면이 있는 경우:
오른쪽 작은 화면의 너비는 전체 캔버스 너비의 1/9이고 오른쪽 작은 화면의 높이는 전체 캔버스 높이의 1/8;
왼쪽의 큰 화면의 너비는 전체 캔버스 너비의 8/9이고 왼쪽의 큰 화면의 높이는 전체 캔버스의 높이;
9보다 크고 17보다 작은 서브 화면이 있는 경우:
오른쪽 작은 화면의 너비는 전체 캔버스 너비의 1/10이고 오른쪽 작은 화면의 높이는 전체 캔버스 높이의 1/8;
왼쪽의 큰 화면의 너비는 전체 캔버스 너비의 4/5이고 왼쪽의 큰 화면의 높이는 전체 캔버스의 높이;
17개 이상의 서브 화면이 있는 경우:
오른쪽(아래) 쪽의 작은 화면의 너비는 전체 캔버스 너비의 1/10이고 오른쪽(아래) 쪽의 작은 화면의 높이는 전체 캔버스 높이의 1/8;
왼쪽의 큰 화면의 너비는 전체 캔버스 너비의 4/5이고 왼쪽의 큰 화면의 높이는 전체 캔버스의 높이의 7/8;
아래와 같이 구독된 서브 화면이 증가함에 따라 화면 공유 레이아웃 변경:









사용자 정의 레이아웃:

업무 요구에 따라 MixLayoutList에서 각 호스트 화면의 레이아웃 정보를 사용자 정의합니다.

2. 녹화 쿼리(DescribeCloudRecording)

필요한 경우 이 인터페이스를 호출하여 녹화 서비스의 상태를 쿼리할 수 있습니다. 정보는 녹화 작업이 있는 경우에만 쿼리할 수 있으며, 녹화 작업이 종료되면 오류가 반환됩니다.
녹화된 파일 목록(StorageFile)에는 이번에 녹화된 모든 M3U8 인덱스 파일과 녹화의 시작 Unix 타임스탬프 정보가 포함됩니다. VOD 업로드 작업인 경우 이 인터페이스에서 반환된 StorageFile이 비어 있습니다.

3. 녹화 업데이트(ModifyCloudRecording)

필요한 경우 이 인터페이스를 호출하여 SubscribeStreamUserIds(단일 스트림 및 혼합 스트림 녹화에 유효) 및 녹화된 템플릿 매개변수 MixLayoutParams(혼합 스트림 녹화에 유효)와 같은 녹화 서비스의 매개변수를 수정할 수 있습니다. 업데이트 작업은 증분 업데이트 작업이 아닌 전체 적용 작업입니다. 템플릿 매개변수 MixLayoutParams 및 블록리스트/얼로우리스트인 SubscribeStreamUserIds를 포함하여 업데이트할 때마다 전체 정보를 전달해야 합니다. 따라서 녹화를 실행하거나 전체 녹화 관련 매개변수를 다시 계산하려면 이전 매개변수를 저장해야 합니다.

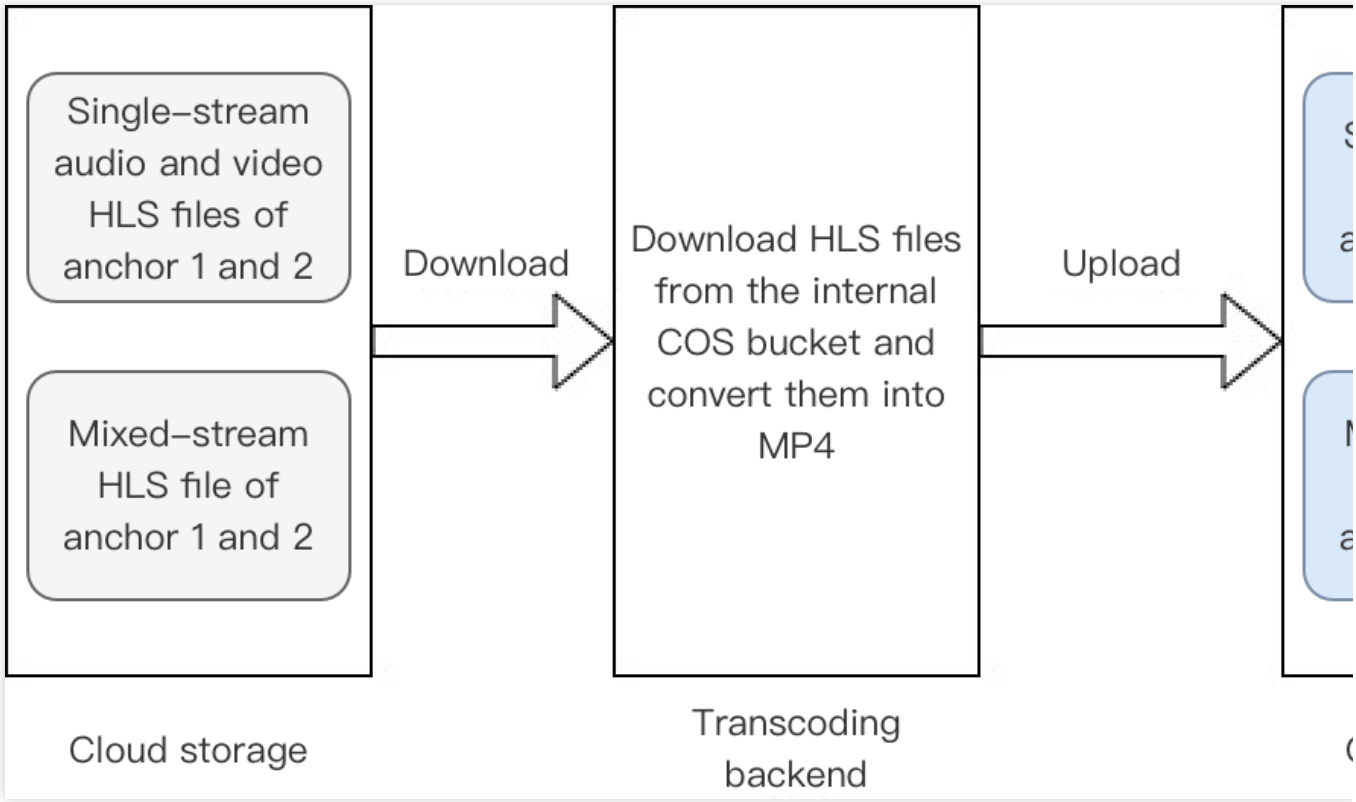
4. 녹화 중지(DeleteCloudRecording)

녹화 종료 후에는 녹화 중지(DeleteCloudRecording) 인터페이스를 호출하여 녹화 작업을 종료해야 하며, 그렇지 않으면 사전 설정된 시간 초과 MaxIdleTime에 도달하면 녹화 작업이 자동으로 종료됩니다. MaxIdleTime의 정의는 방에 호스트가 없는 상태가 MaxIdleTime의 지속시간을 초과하는 것으로, 여기서 방에 호스트가 있지만 호스트에 업스트림 데이터가 없으면 타임아웃 카운트다운 상태에 들어가지 않습니다. 녹화가 종료되면 서비스에서 이 인터페이스를 호출하여 녹화 작업을 종료하는 것이 좋습니다.

고급 기능

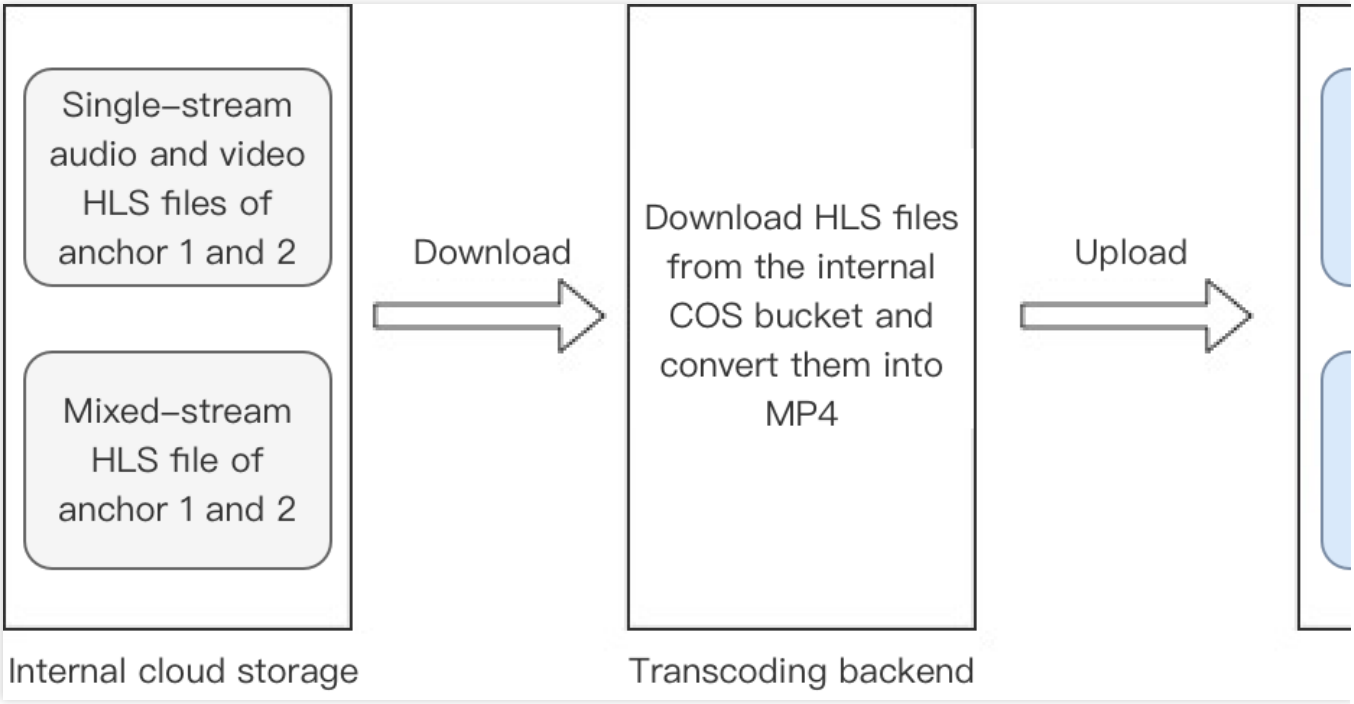
녹화 mp4 파일

출력 파일 형식을 mp4 형식으로 녹화하려면 매개변수(OutputFormat)에 녹화 파일의 출력 형식을 hls+mp4로 지정하여 녹화를 실행할 수 있습니다. hls 파일은 기본적으로 계속 녹화됩니다. hls 녹화가 완료되면 mp4 변환 작업이 즉시 트리거되고 hls가 있는 cos에서 mp4 캡슐화 풀 스트림 후 클라이언트의 cos에 업로드합니다.
여기서 COS의 파일 풀다운 권한을 제공해야 합니다. 그렇지 않으면 hls 파일 풀다운이 실패하여 mp4 작업으로 전송이 종료됩니다.
mp4 파일은 다음과 같은 경우 강제로 샤드됩니다.
1. 녹화 시간이 24시간을 초과하는 경우;
2. 단일 mp4 파일의 크기가 2GB에 도달;
특정 프로세스는 이미지 참고



녹화 VOD 업로드

녹화된 출력 파일을 VOD 플랫폼에 업로드하려면 녹화를 실행하기 위한 스토리지 매개변수(StorageParams)에 CloudVod 참석자 매개변수를 지정할 수 있습니다. 녹화 백그라운드는 녹화가 끝난 후 지정한 방식으로 녹화된 mp4 파일을 VOD 플랫폼에 업로드 하고, 콜백 형태로 재생 주소를 보내드립니다. 녹화 모드가 단일 스트림 녹화 모드인 경우 구독된 각 호스트에는 해당 재생 주소가 있고 녹화 모드가 혼합 스트림 녹화 모드인 경우 혼합 스트림 미디어의 재생 주소는 하나만 있습니다. VOD 작업 업로드에는 다음과 같은 주의 사항이 있습니다.
1. 스토리지 매개변수의 CloudVod 및 CloudStorage 중 하나만 동시에 지정할 수 있습니다. 그렇지 않으면 녹화 실패;
2. VOD 작업을 업로드하는 과정에서 DescribeCloudRecording을 사용하여 쿼리한 작업 상태에는 녹화된 파일의 정보 불포함;
구체적인 과정을 이미지로 나타내었는데, 여기에서 클라우드 스토리지는 녹화를 위한 내부 클라우드 스토리지이며 클라이언트는 관련 매개변수를 입력할 필요가 없습니다.



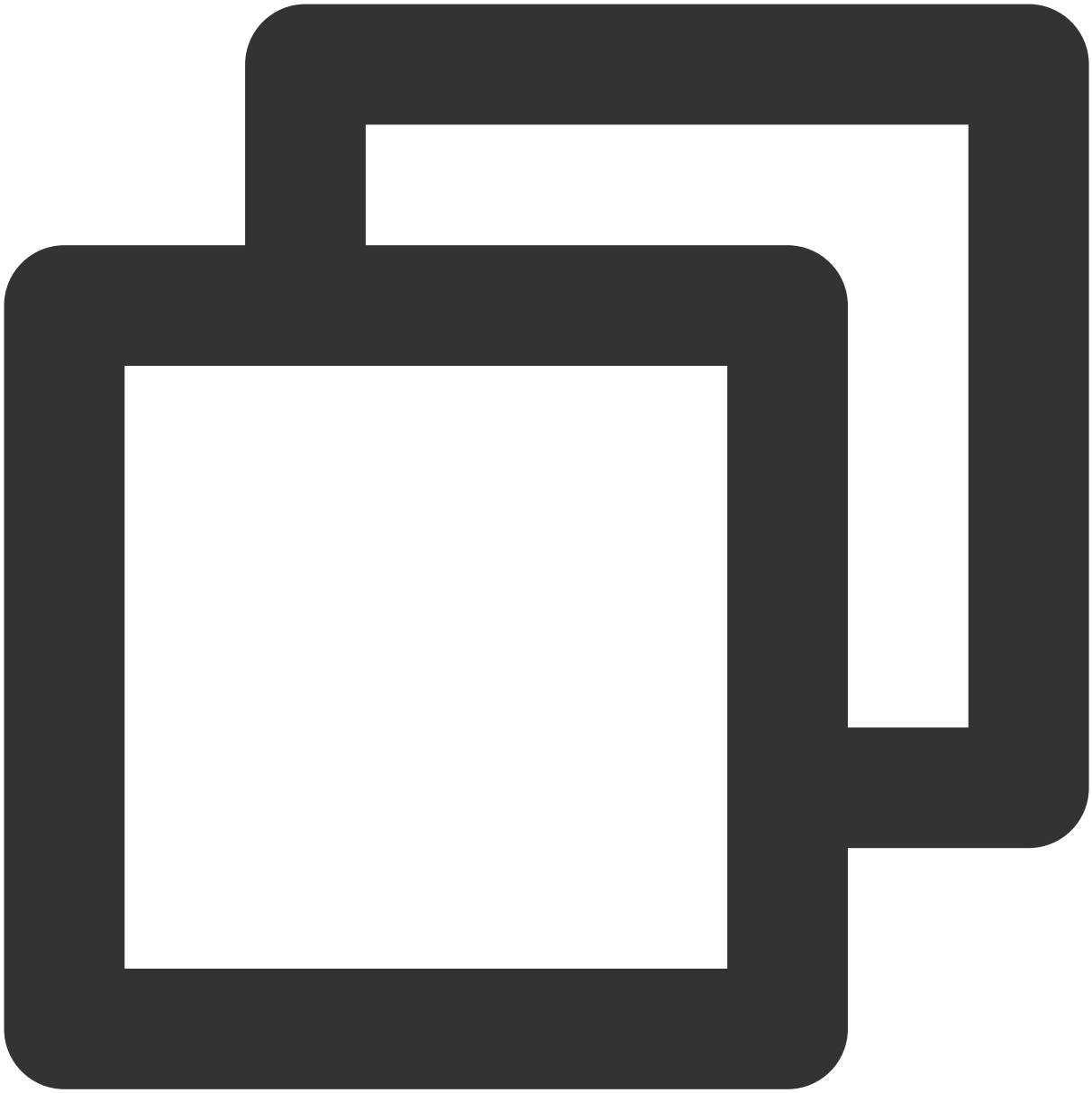
단일 스트림 파일 병합 스크립트

단일 스트림 녹화 멀티미디어 파일을 MP4 파일로 병합하는 데 사용되는 단일 스트림 멀티미디어 파일 병합 스크립트를 제공합니다.
설명：
두 슬라이스 사이의 시간 간격이 15초 이상이고 간격에 오디오/비디오 정보가 없는 경우(서브스트림이 비활성화된 경우 서브스트림 정보는 무시됨) 두 슬라이스를 두 개의 다른 세그먼트로 처리합니다. 녹화 시간이 더 빠른 슬라이스는 이전 세그먼트의 끝 슬라이스로 간주되고, 녹화 시간이 더 늦은 슬라이스는 다음 세그먼트의 시작 슬라이스로 간주됩니다.
세그먼트 모드(-m 0)
이 모드에서 스크립트는 각 UserId 아래의 녹화 파일을 세그먼트별로 병합하고 UserId 아래의 세그먼트는 독립적으

로 파일에 병합됩니다.
병합 모드(-m 1)
동일한 UserId 아래의 모든 세그먼트를 하나의 멀티미디어 파일로 결합합니다. -s 옵션은 세그먼트 사이의 간격을 채울지 여부를 선택하는 데 사용할 수 있습니다.

종속 환경

Python3
Centos: sudo yum install python3
Ubuntu: sudo apt-get install python3
Python3 종속 패키지



```
sortedcontainers：pip3 install sortedcontainers
```

사용 방법

1. 병합 스크립트 실행: python3 TRTC_Merge.py [option]
2. 녹화 파일 디렉터리에 병합된 mp4 파일 생성

예시: python3 TRTC_Merge.py -f /xxx/file -m 0
선택적 매개변수와 해당 기능은 다음 표 참고:

매개	기능

변수	
-f	병합할 파일의 저장 경로를 지정합니다. UserId 녹화 파일이 여러 개인 경우 스크립트는 파일을 별도로 병합합니다.
-m	0: 세그먼트 모드(기본 설정), 이 모드에서 스크립트는 세그먼트별로 각 UserId 아래의 녹화 파일을 병합하고 각 UserId는 여러 파일을 생성할 수 있습니다. 1: 병합 모드에서는 UserId 아래의 모든 멀티미디어 파일이 하나의 파일로 병합됩니다.
-s	저장 모드. 이 매개변수가 설정되면 병합 모드에서 세그먼트 사이의 공백이 삭제되고 파일의 실제 시간은 물리적 일반 시간보다 작습니다.
-a	0: 주요 스트림 병합(기본 설정), 동일한 UserId의 주요 스트림 오디오와 병합되고 서브 스트림은 오디오와 병합되지 않습니다. 1: 자동 병합, 주요 스트림이 있으면 주요 스트림과 오디오가 병합되고, 주요 스트림이 없으면 서브 스트림과 오디오가 병합됩니다. 2: 서브 스트림 병합, 동일한 UserId의 서브 스트림과 오디오가 병합되고 주요 스트림은 오디오와 병합되지 않습니다.
-p	출력 비디오의 fps를 지정합니다. 기본값은 15 fps이고 유효한 범위는 5-120 fps이며 5 fps 미만은 5 fps로, 120 fps 이상은 120 fps로 계산합니다.
-r	출력 비디오의 해상도를 지정합니다. 예를 들어 -r 640 360은 출력 비디오의 너비가 640이고 높이가 360임을 의미합니다.

파일 이름 규칙

멀티미디어 파일 이름 규칙: UserId_timestamp_av.mp4

퓨어 오디오 파일 이름 규칙: UserId_timestamp.m4a

설명：

여기서 UserId는 녹화 스트림의 사용자 ID의 특수한 base64 값입니다. 자세한 내용은 실행 녹화 파일 이름 명명 규칙에 대한 설명을 참고하십시오. timestamp는 각 세그먼트의 첫 번째 ts 슬라이스에 대해 녹화가 실행된 시간입니다.

콜백 인터페이스

콜백을 수신하는 Http/Https 서비스 게이트웨이를 제공하여 콜백 메시지를 구독할 수 있습니다. 관련 이벤트가 발생하면 클라우드 녹화 시스템이 이벤트 공지를 메시지 수신 서버로 다시 콜백합니다.

이벤트 콜백 메시지 포맷

이벤트 콜백 메시지는 HTTP/HTTPS POST 요청 형식으로 사용자의 서버에 전송됩니다.

문자 인코딩 포맷: UTF-8.

요청: body 포맷은 JSON.

응답: HTTP STATUS CODE = 200, 서버가 응답 패키지의 세부 콘텐츠를 무시합니다. 원활한 프로토콜 연결을 위해 클라이언트 응답 콘텐츠에 JSON: {"code":0} 추가를 권장합니다.

매개변수 설명

이벤트 콜백 메시지의 header는 다음과 같은 필드 포함:

필드 이름	값
Content-Type	application/json
Sign	서명 값
SdkAppId	sdk application id

이벤트 콜백 메시지의 body는 다음과 같은 필드 포함:

필드 이름	유형	의미
EventGroupId	Number	이벤트 그룹 ID, 클라우드 녹화를 위해 3으로 고정
EventType	Number	콜백 공지의 이벤트 유형
CallbackTs	Number	이벤트 콜백 서버에서 귀하의 서버로 보낸 콜백 요청의 Unix 타임스탬프, 단위: 밀리초
EventInfo	JSON Object	이벤트 정보

이벤트 유형 설명

필드 이름	유형	의미
EVENT_TYPE_CLOUD_RECORDING_RECORDER_START	301	클라우드 녹화 녹화 모듈 실행
EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP	302	클라우드 녹화 녹화 모듈 종료
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START	303	클라우드 녹화 업로드 모듈 실행
EVENT_TYPE_CLOUD_RECORDING_FILE_INFO	304	클라우드 녹화 m3u8 인덱스 파일 생성, 첫 번째 생성 및 업로드 완료 후 콜백
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP	305	클라우드 녹화 업로드 종료

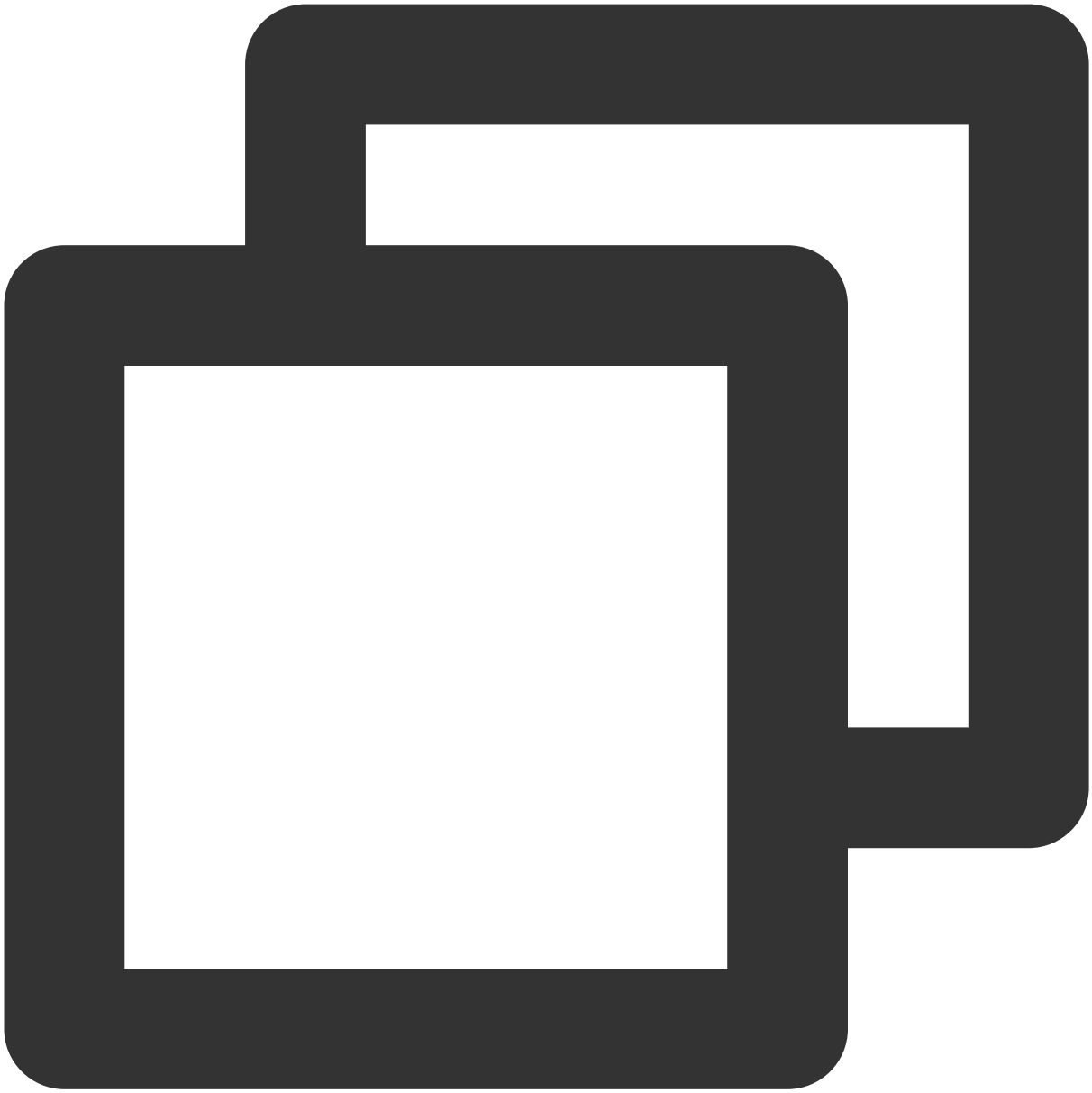
EVENT_TYPE_CLOUD_RECORDING_FAILOVER	306	클라우드 녹화 마이그레이션이 발생, 기존 녹화 작업이 새 부하로 마이그레이션 될 때 트리거
EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE	307	클라우드 녹화 M3U8 파일 생성(첫 번째 ts 샤드 잘라내기) 생성 후 콜백
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR	308	클라우드 녹화 업로드 모듈에서 오류 발생
EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR	309	클라우드 녹화 디코딩된 이미지 파일 다운로드 시 오류 발생
EVENT_TYPE_CLOUD_RECORDING_MP4_STOP	310	클라우드 녹화 mp4 녹화 작업이 종료되고 콜백에 녹화된 mp4 파일 이름 포함
EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT	311	클라우드 녹화 vod 녹화 작업 업로드 미디어 리소스 완료
EVENT_TYPE_CLOUD_RECORDING_VOD_STOP	312	클라우드 녹화 vod 녹화 작업 종료

이벤트 정보 설명

필드 이름	유형	의미
RoomId	String/Number	방 이름(유형이 클라이언트 방 번호 유형과 일치)
EventTs	Number	이벤트 발생 시간의 Unix 타임스탬프, 단위: 초
UserId	String	녹화 로봇의 사용자 ID
TaskId	String	녹화 ID, 클라우드 녹화 작업의 고유 ID
Payload	JsonObject	다양한 이벤트 유형에 대한 다양한 정의

이벤트 유형이 301 EVENT_TYPE_CLOUD_RECORDING_RECORDER_START일 때 Payload 정의

필드 이름	유형	의미
Status	Number	0: 녹화 모듈 실행 성공, 1: 녹화 모듈 실행 실패.

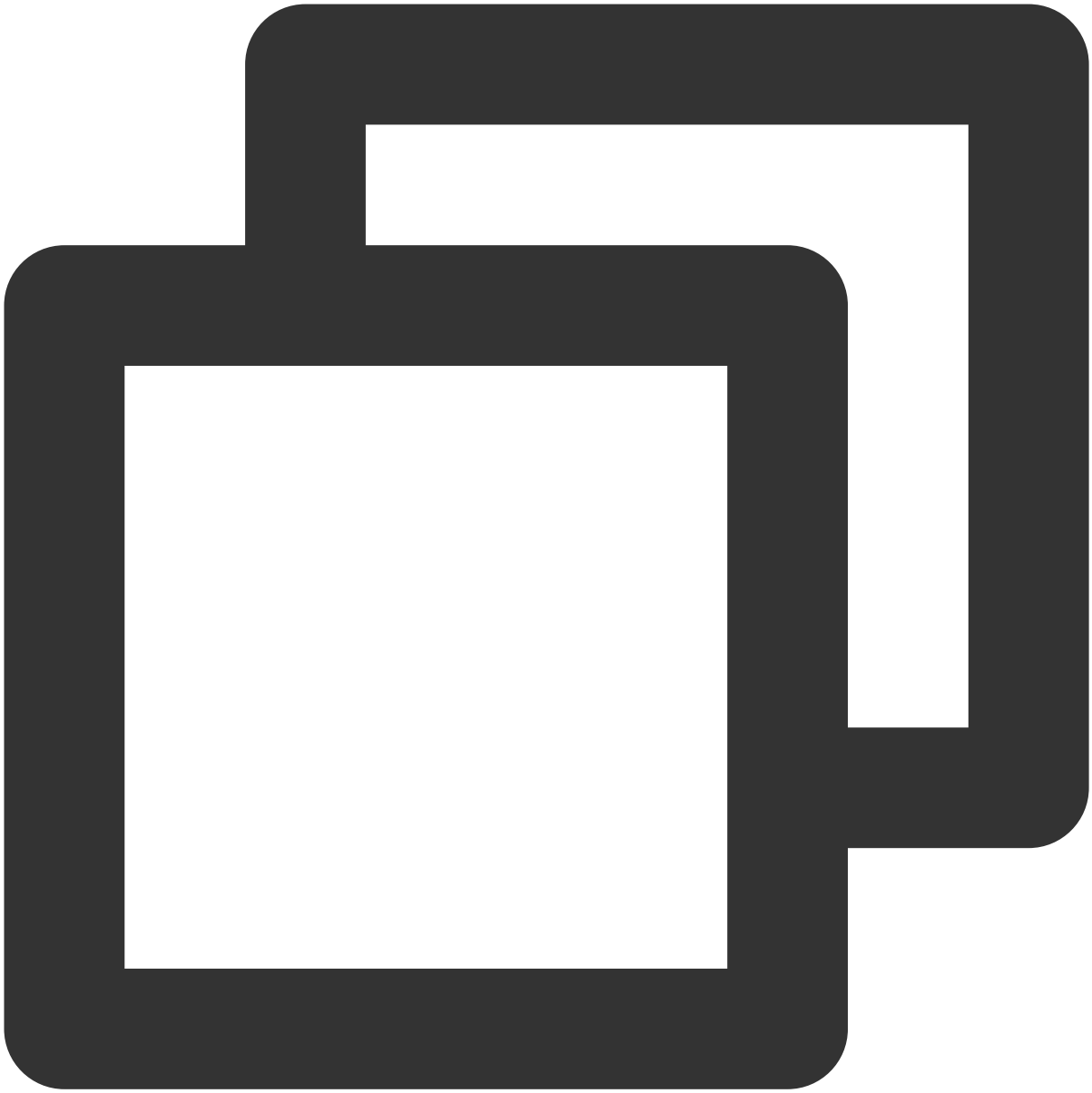


```
{
        "EventGroupId": 3,
        "EventType":    301,
        "CallbackTs":   1622186275913,
        "EventInfo":    {
                "RoomId":       "xx",
                "EventTs":      "1622186275",
                "UserId":       "xx",
                "TaskId":       "xx",
             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

```
                    }
                }
            }
```

이벤트 유형이 302 EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP일 때 Payload 정의

필드 이름	유형	의미
LeaveCode	Number	0: 녹화 모듈은 일반적으로 녹화를 중지하고 종료하도록 호출됨; 1: 녹화 로봇은 클라이언트에 의해 방에서 퇴장; 2: 클라이언트가 방을 해산; 3: 서버가 녹화 로봇을 퇴장시킴; 4: 서버가 방 해산; 99: 방에 녹화 로봇 외에 다른 사용자 스트림이 없으며 지정된 시간 후에 퇴장; 100: 방 시간 초과로 퇴장; 101: 같은 사용자가 같은 방에 반복적으로 입장하면 로봇이 퇴장됨

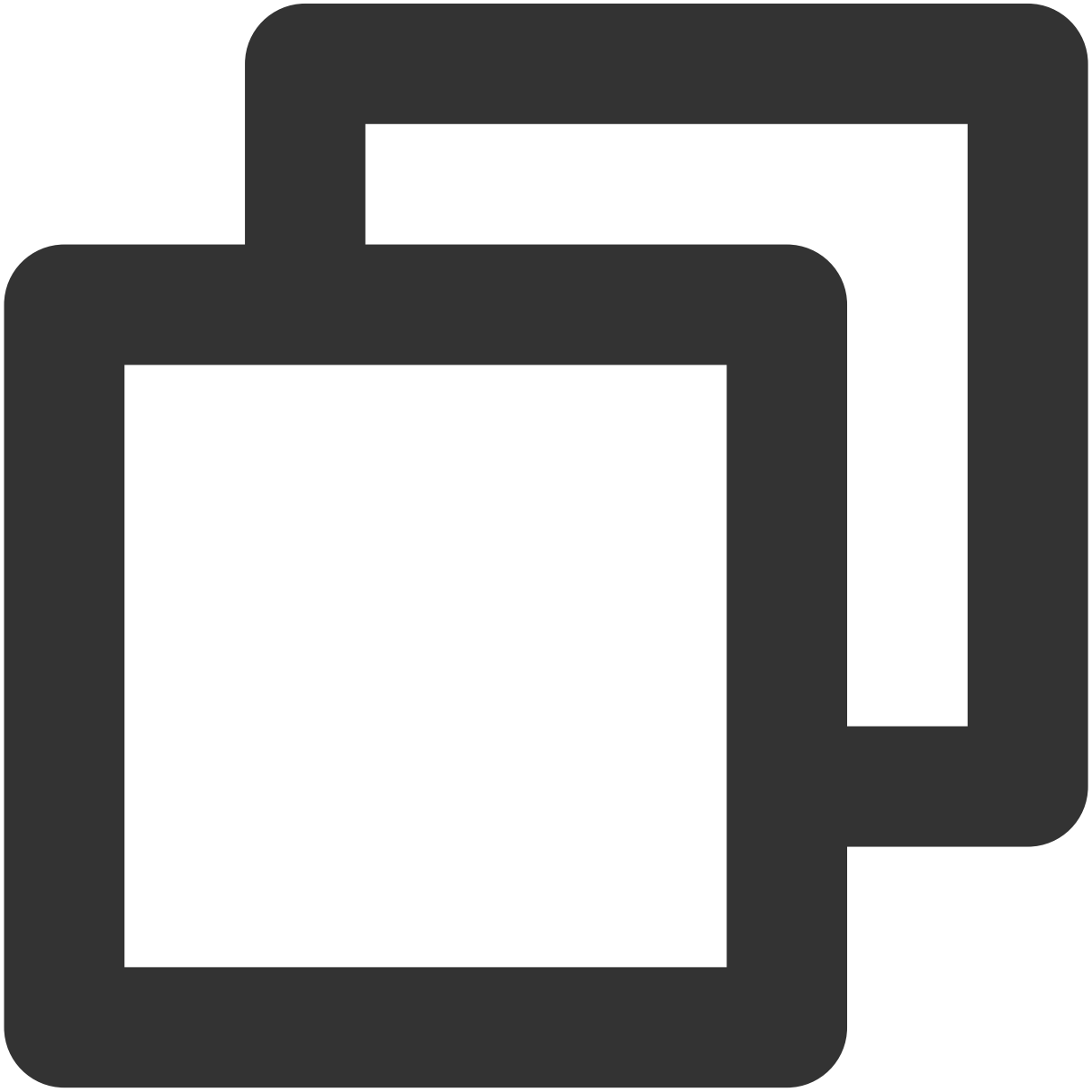


```
{
        "EventGroupId": 3,
        "EventType":    302,
        "CallbackTs":   1622186354806,
        "EventInfo":    {
                "RoomId":       "xx",
                "EventTs":      "1622186354",
                "UserId":       "xx",
                "TaskId":       "xx",
             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

```
                    }
                }
            }
```

이벤트 유형이 303 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START일 때 Payload 정의

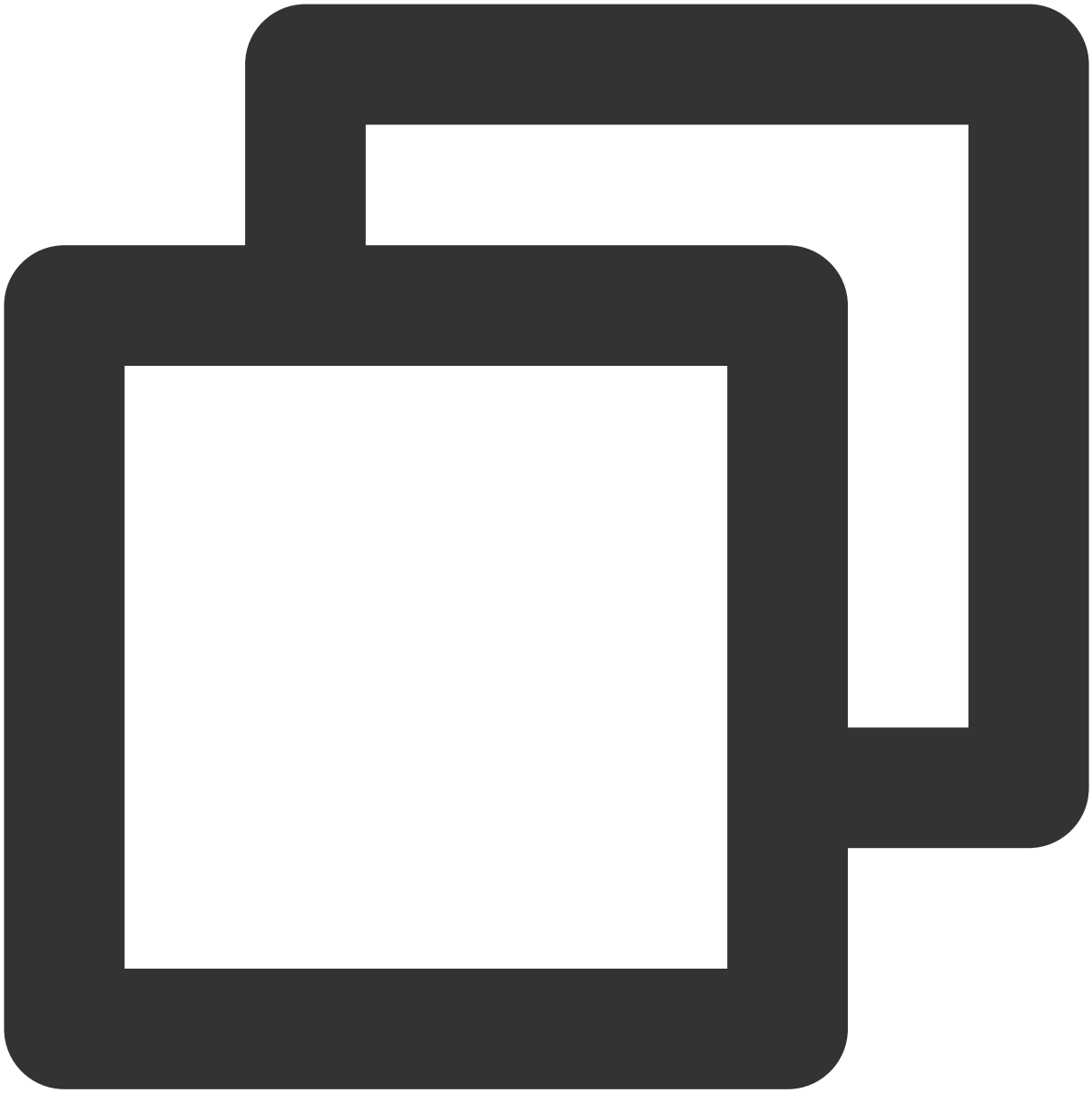
필드 이름	유형	의미
Status	Number	0: 업로드 모듈이 정상적으로 실행 1: 업로드 모듈 초기화 실패.



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

이벤트 유형이 304 EVENT_TYPE_CLOUD_RECORDING_FILE_INFO일 때 Payload 정의

필드	유형	의미
FileList	String	생성된 M3U8 파일 이름

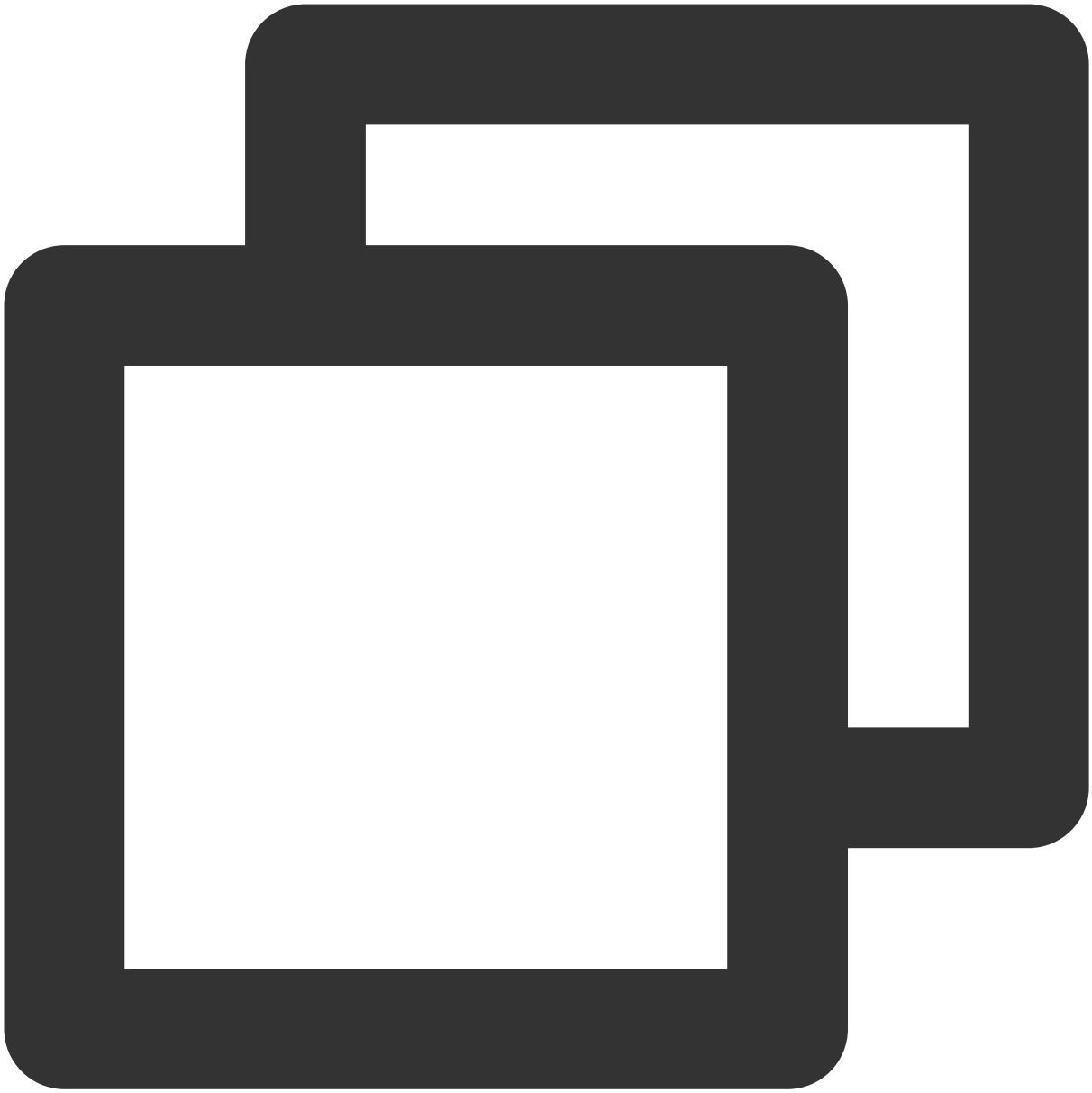


```
{
        "EventGroupId": 3,
        "EventType":    304,
        "CallbackTs":   1622191965350,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
            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

```
                    }
                }
            }
```

이벤트 유형이 305 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP일 때 Payload 정의

필드	유형	의미
Status	Number	0: 녹화 및 업로드 작업이 완료되었으며 모든 파일이 지정된 타사 클라우드 스토리지에 업로드 됨 1: 녹화 및 업로드 작업이 완료되었지만 적어도 하나의 파일이 서버 또는 백업 저장소에 남아 있음 2: 서버 또는 백업 스토리지에 멈춘 파일이 복구되어 지정된 타사 클라우드 스토리지에 업로드

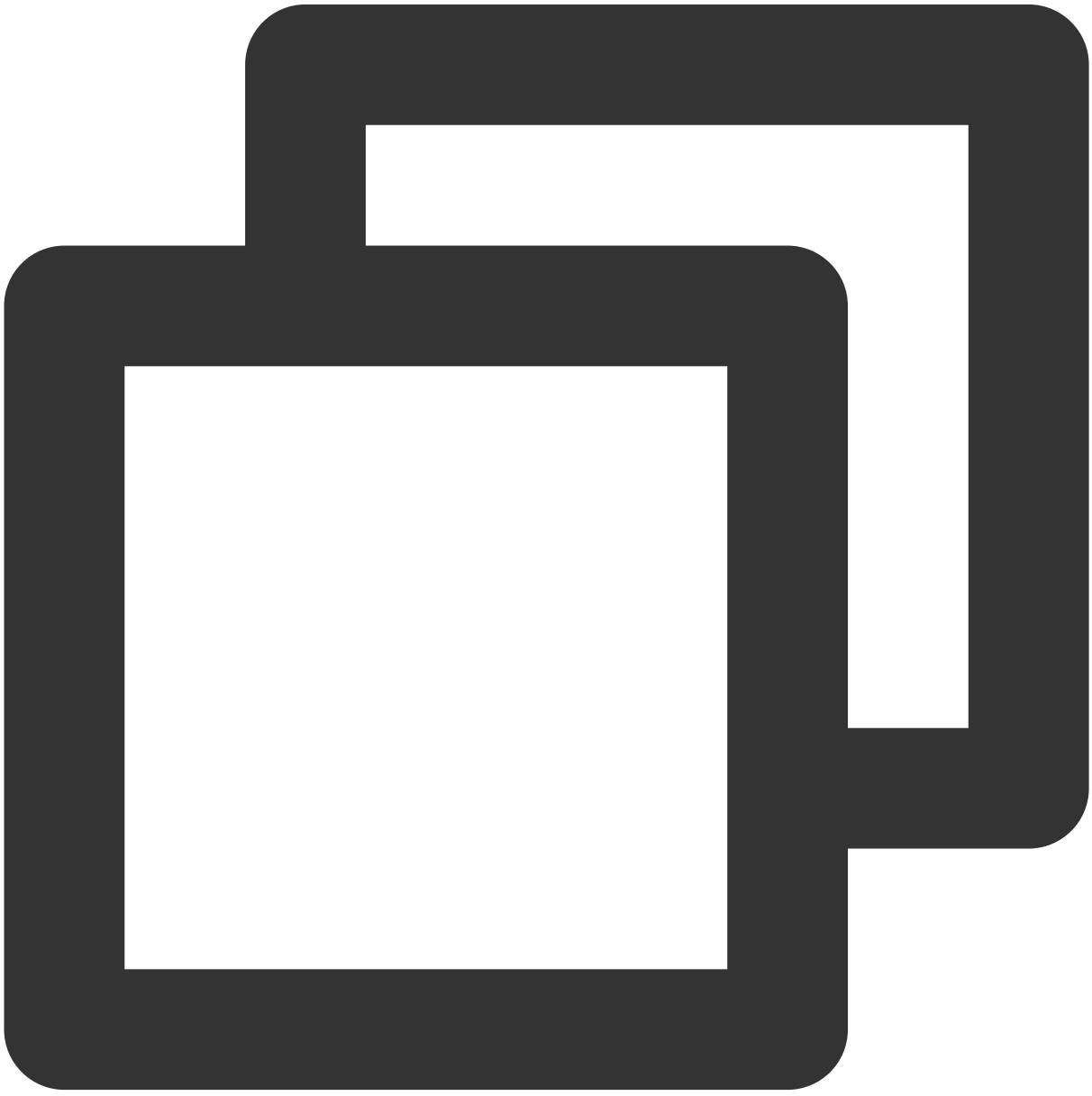


```
{
        "EventGroupId": 3,
        "EventType":    305,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
            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

```
                    }
                }
            }
```

이벤트 유형이 306 EVENT_TYPE_CLOUD_RECORDING_FAILOVER일 때 Payload 정의

필드	유형	의미
Status	Number	0: 마이그레이션 완료됨

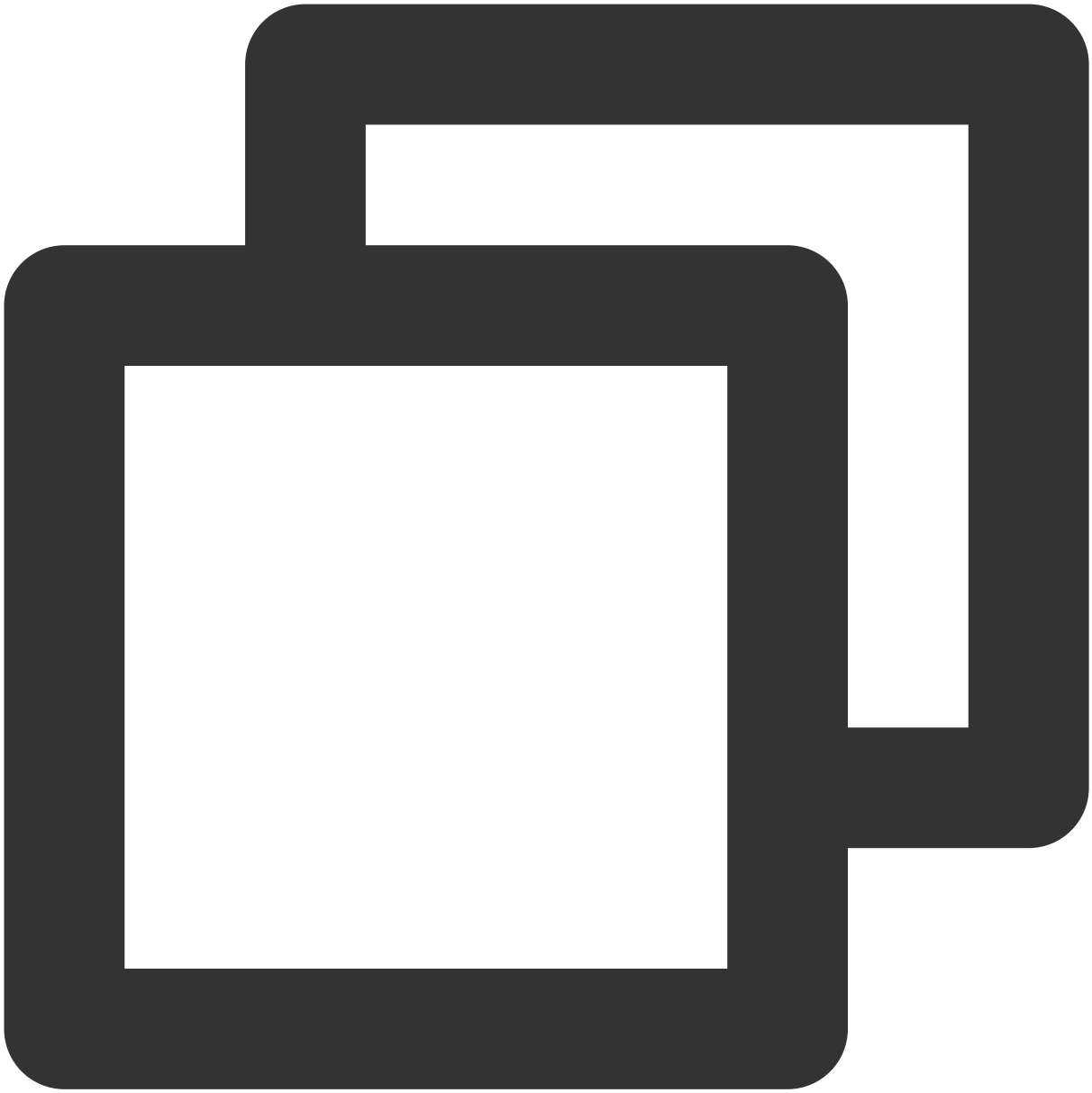


```
{
```

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

이벤트 유형은 307 EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE일 때 Payload 정의

필드	유형	의미
FileName	String	m3u8 파일 이름
UserId	String	이 녹화 파일에 해당하는 사용자 ID
TrackType	String	audio/video/audio_video
BeginTimeStamp	Number	녹화 시작 시 서버 Unix 타임스탬프(밀리초)

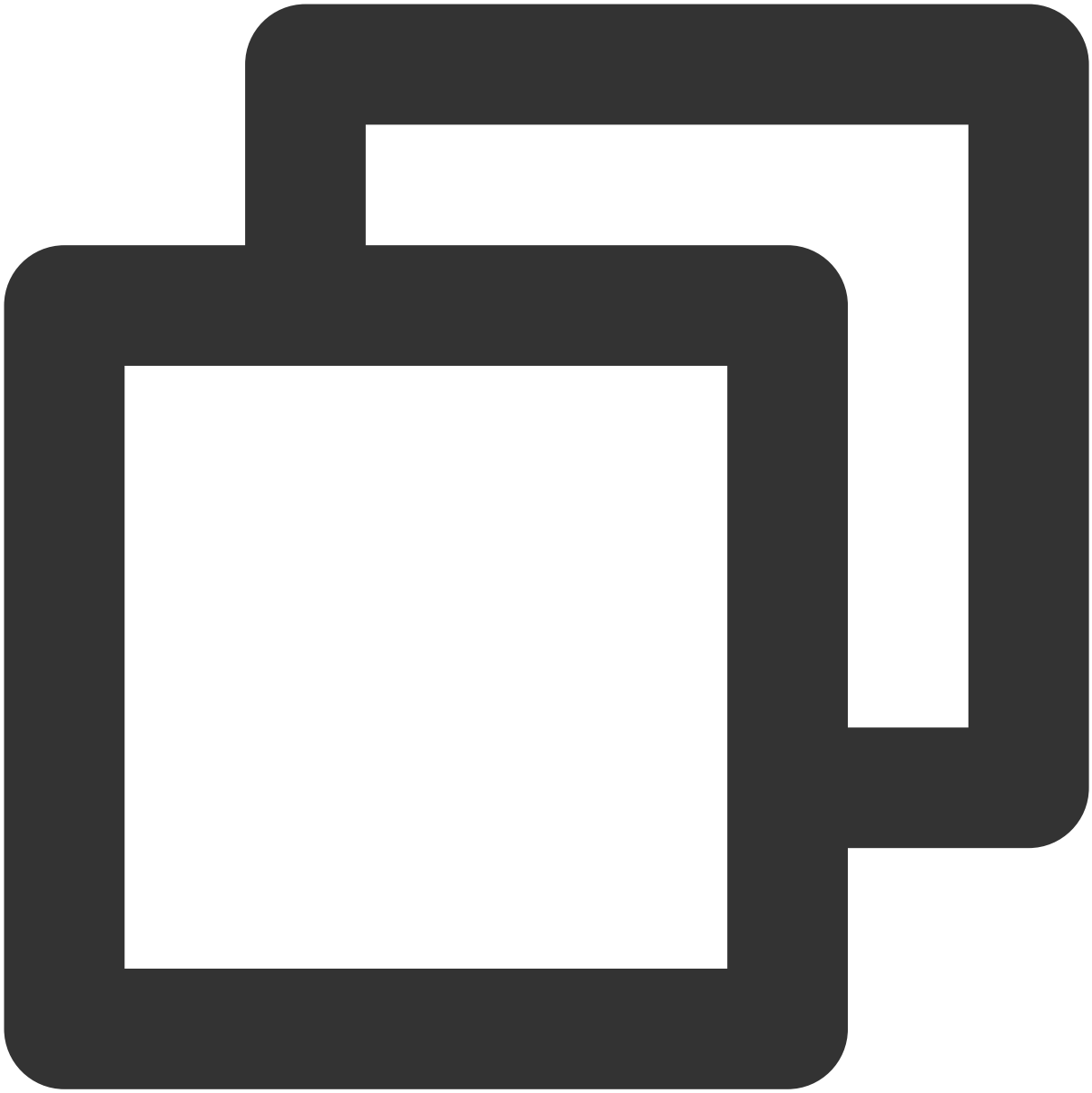


```
{
        "EventGroupId": 3,
        "EventType":    307,
        "CallbackTs":   1622186289148,
        "EventInfo":    {
                "RoomId":       "xx",
                "EventTs":      "1622186289",
                "UserId":       "xx",
                "TaskId":       "xx",
             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

```
                        "UserId":       "xx",
                        "TrackType":    "audio",
                        "BeginTimeStamp":       1622186279144
                }
        }
}
```

이벤트 유형은 308 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR일 때 Payload 정의

필드	유형	의미
Code	String	타사 클라우드 스토리지의 반환 code
Message	String	타사 클라우드 스토리지의 메시지 콘텐츠 반환

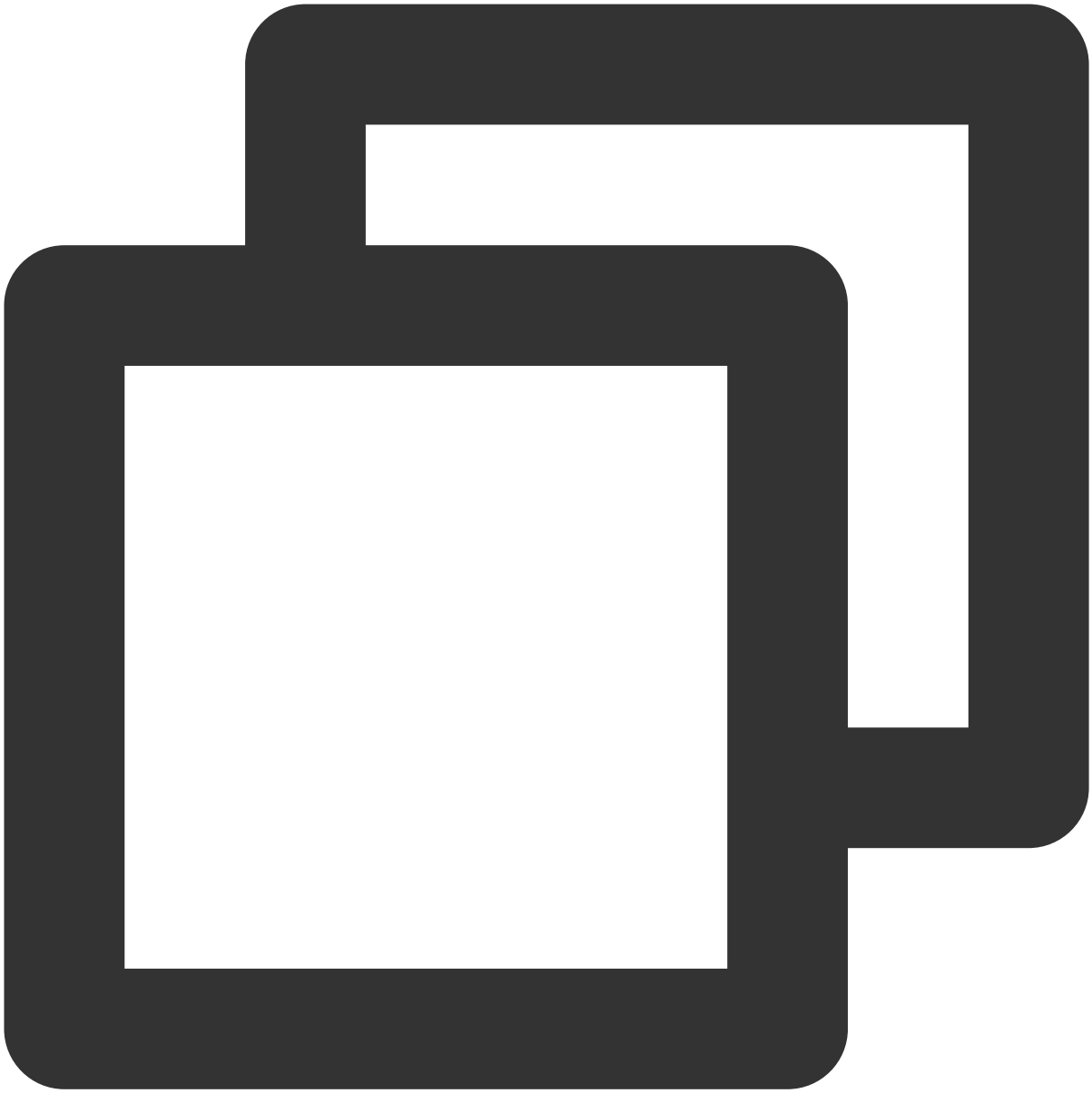


```
{
  "Code": "InvalidParameter",
  "Message": "AccessKey invalid"
}

{
        "EventGroupId": 3,
        "EventType":    308,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
```

```
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Code":         "xx",
                        "Message":      "xx"
                }
        }
}
```

이벤트 유형은 309 EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR일 때 Payload 정의

필드	유형	의미
Url	String	다운로드 실패 url

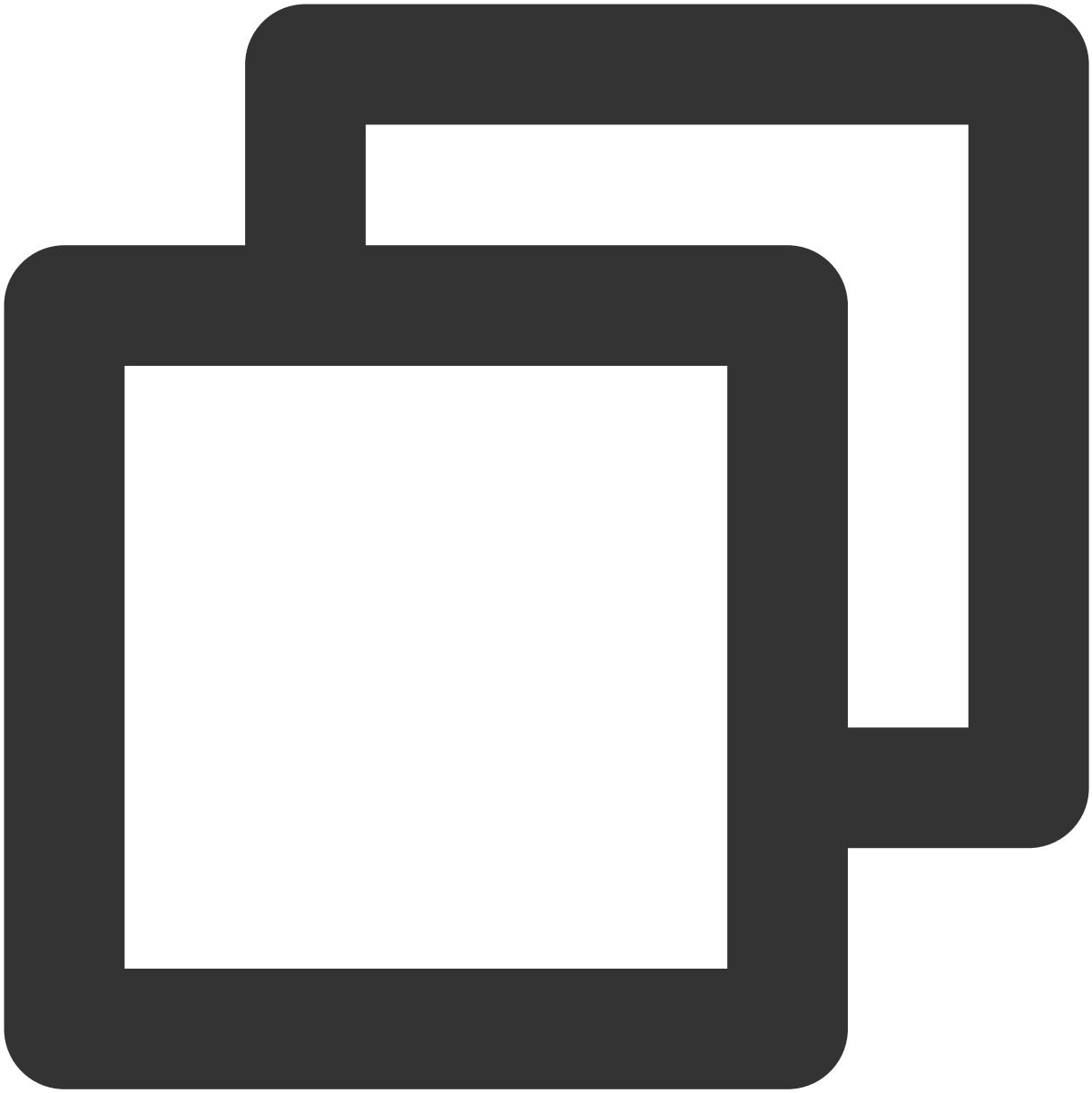


```
{
        "EventGroupId": 3,
        "EventType":    309,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
            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

```
                    }
                }
            }
```

이벤트 유형은 310 EVENT_TYPE_CLOUD_RECORDING_MP4_STOP일 때 Payload 정의

필드 이름	유형	의미
Status	Number	0: mp4 녹화 작업이 정상적으로 종료되었으며 모든 파일이 지정된 타사 클라우드 스토리지에 업로드되었음 1: mp4 녹화 작업이 정상적으로 종료되었지만 적어도 하나의 파일이 서버 또는 백업 스토리지에 남아 있음 2: mp4 녹화 작업의 비정상 종료(예상 원인은 cos의 hls 파일 풀링 실패)
FileList	String	생성된 M3U8 파일 이름



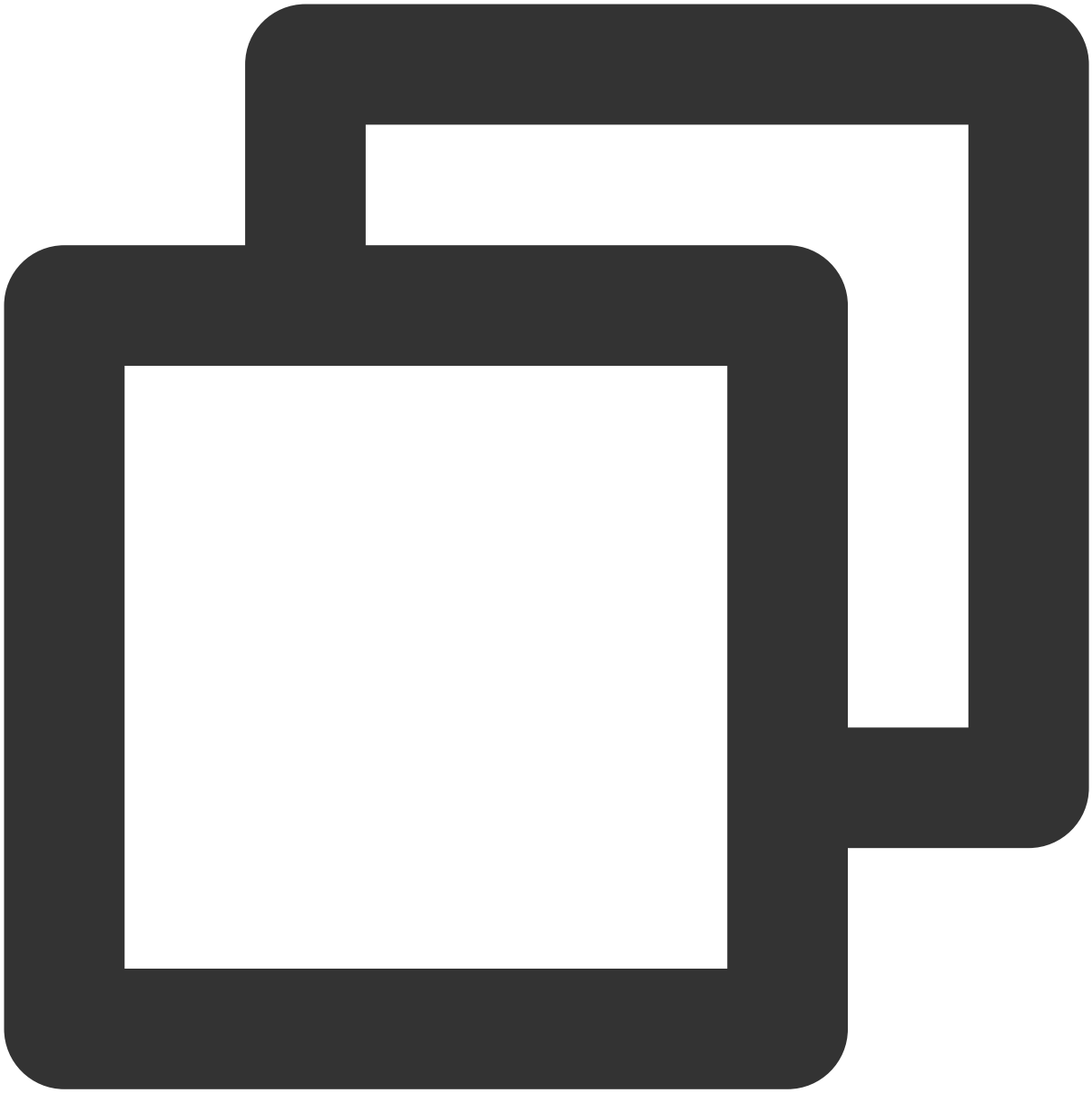
```
{
        "EventGroupId": 3,
        "EventType":    310,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0,
```

```
                        "FileList": ["xxxx1.mp4", "xxxx2.mp4"]
                    }
            }
    }
```

이벤트 유형은 311 EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT일 때 Payload 정의

필드 이름	유형	의미
Status	Number	0: 녹화 파일은 VOD 플랫폼에 정상적으로 업로드 됨 1: 이 녹화 파일은 서버 또는 백업 스토리지에 남아있음 2: 이 녹화 파일을 업로드하는 VOD 작업이 비정상
UserId	String	이 녹화 파일에 해당하는 사용자 ID(녹화 모드가 혼합 스트림 모드인 경우 이 필드는 비어 있음)
TrackType	String	audio/video/audio_video
MediaId	String	main/aux
FileId	String	VOD 플랫폼에서 이 녹화 파일의 고유 id
VideoUrl	String	VOD 플랫폼에서 이 녹화 파일의 재생 주소
CacheFile	String	이 녹화 파일에 해당하는 mp4 파일 이름(VOD 업로드 전)
Errmsg	String	statue가 0이 아닌 경우, 해당 오류 메시지

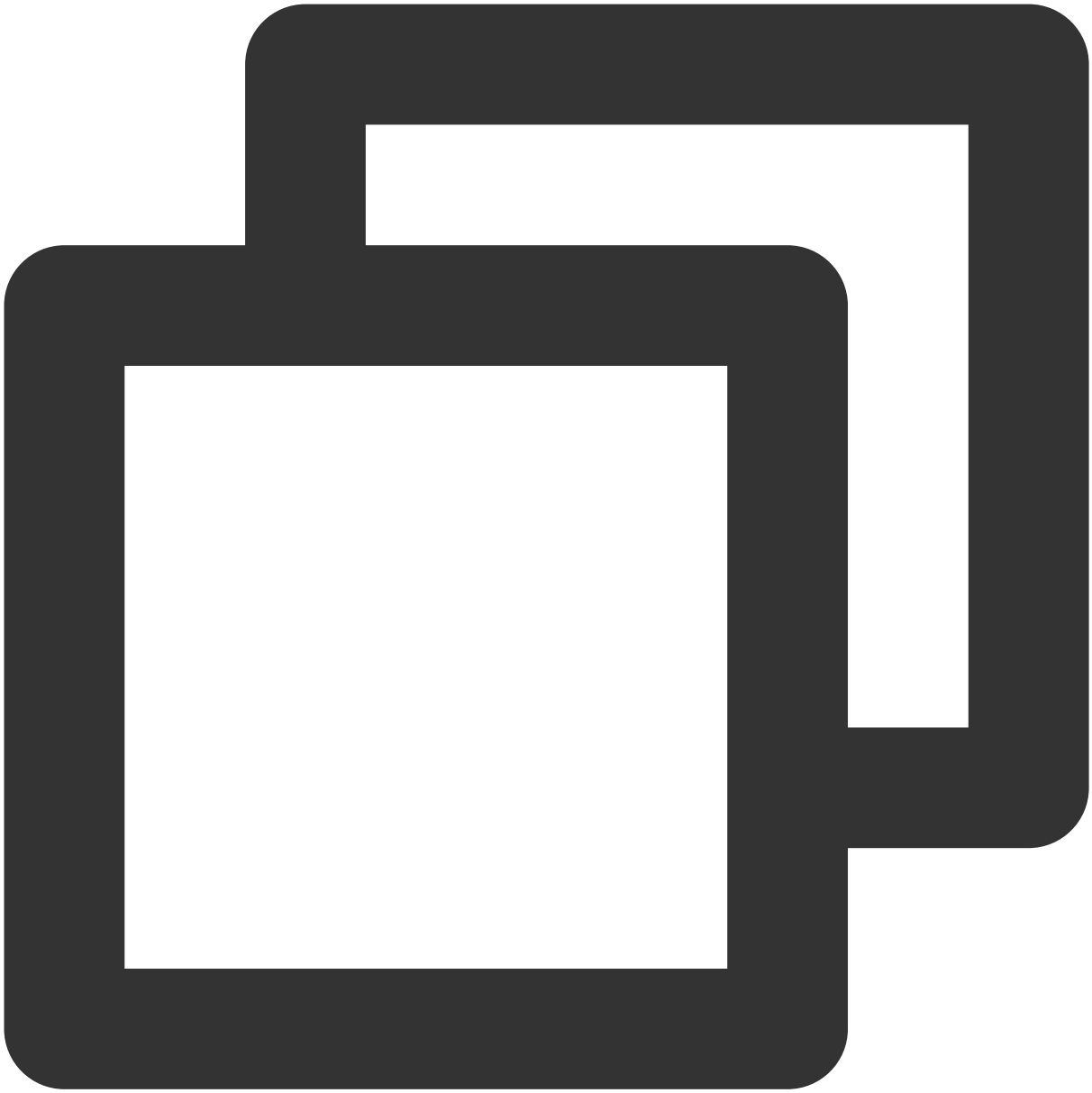
성공적인 콜백 업로드:



```
{
        "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"
                        }
                }
        }
}
```

업로드 실패 콜백:

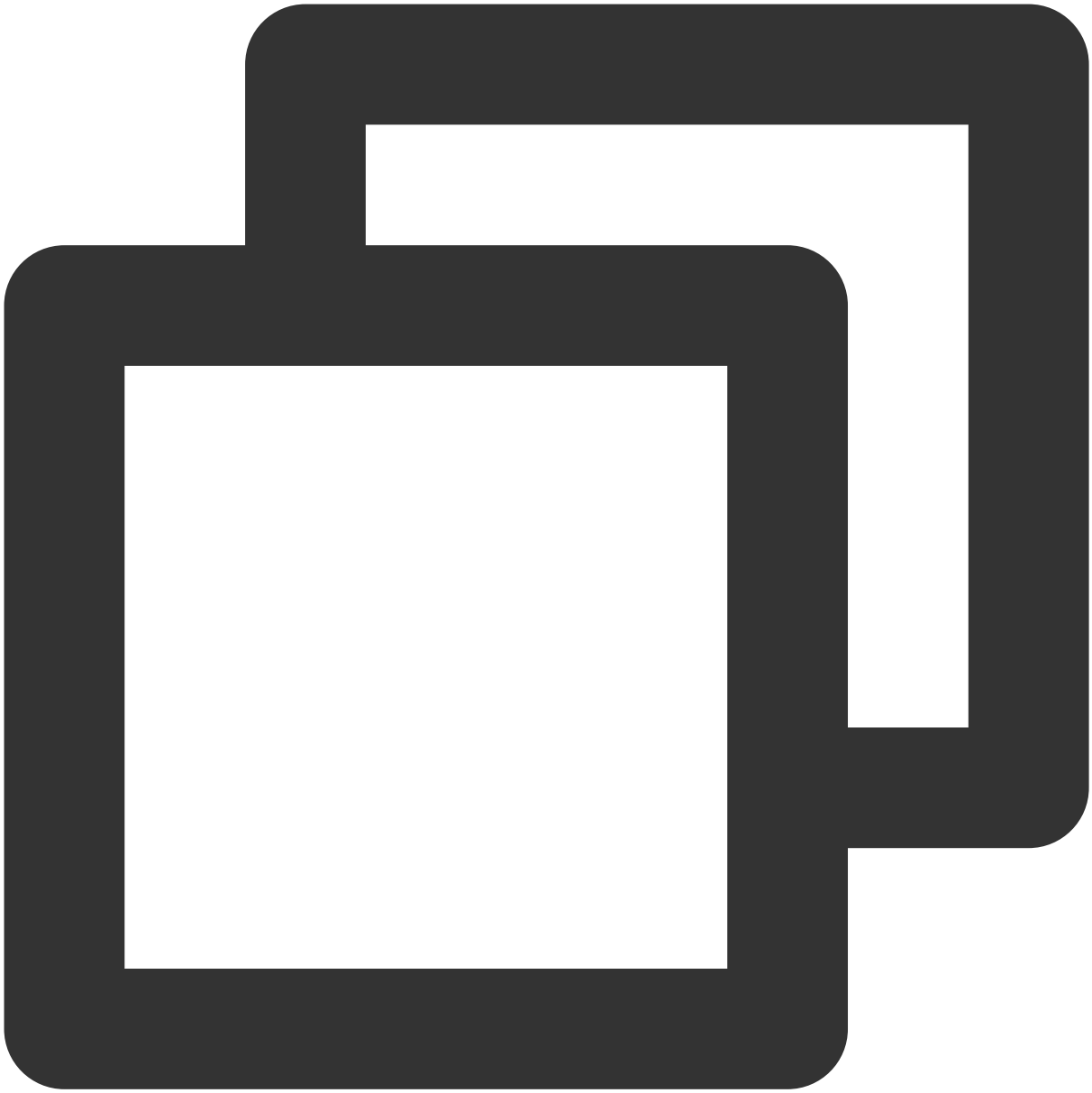


```
{
        "EventGroupId": 3,
        "EventType":    311,
        "CallbackTs":   1622191965320,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       1,
```

```
                        "Errmsg":       "xxx",
                        "TencentVod":   {
                            "UserId":       "123",
                            "TrackType":    "audio_video",
                            "CacheFile":    "xxx.mp4"
                        }
                    }
            }
    }
```

이벤트 유형은 312 EVENT_TYPE_CLOUD_RECORDING_VOD_STOP일 때 Payload 정의

필드	유형	의미
Status	Number	0: 이번 vod 업로드 작업이 정상적으로 종료됨 1: vod 업로드 작업이 비정상적으로 종료



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

```
                    }
                }
            }
```

모범 사례

녹화의 고가용성을 보장하기 위해 Restful API 통합 시 다음 사항을 주의하십시오.

1. CreateCloudRecording 요청을 호출한 후 http response에 주의를 기울이고, 요청이 실패할 경우 특정 상태 코드에 따라 해당 재시도 정책을 채택해야 합니다.

에러 코드는 'InvalidParameter.SdkAppId'와 같이 '1차 에러 코드'와 '2차 에러 코드'로 구성됩니다.

반환된 Code가 InternalError.xxxxx이면 서버 측 오류가 발생했음을 의미합니다. 반환이 정상이고 taskid를 얻을 때까지 동일한 매개변수를 사용하여 여러 번 재시도할 수 있습니다. 첫 번째 3초 재시도, 두 번째 6초 재시도, 세 번째 12초 재시도 등과 같은 백오프 재시도 정책을 사용하는 것이 좋습니다.

반환된 Code가 InValidParameter.xxxxx이면 입력 매개변수가 잘못된 것이므로 프롬프트에 따라 매개변수를 확인하십시오.

반환된 Code가 FailedOperation.RestrictedConcurrency인 경우 클라이언트의 동시 녹화 작업 수가 백그라운드에서 예약된 리소스(기본적으로 100개 채널)를 초과한다는 의미이므로 최대 동시 채널 제한을 조정하려면 Tencent Cloud 기술 지원에 문의하십시오.

2. CreateCloudRecording 인터페이스 호출 시 지정된 UserId/UserSig는 별도의 로봇 사용자로 방에 참여하는 기록된 사용자의 ID이므로 TRTC 방의 다른 사용자와 중복하지 마십시오. 동시에, TRTC 클라이언트에 의해 추가된 방 유형은 녹화 인터페이스에서 지정한 방 유형과 동일해야 합니다. 예를 들어 SDK가 문자열 방 번호를 사용하여 방을 생성하는 경우 클라우드에 녹화된 방 유형은 또한 그에 따라 문자열 방 번호로 설정해야 합니다.

3. 녹화 상태 쿼리를 위해 클라이언트는 다음과 같은 방법으로 녹화의 해당 파일 정보를 얻을 수 있습니다.

CreateCloudRecording 작업이 성공적으로 전달된 후 약 15초 후에 DescribeCloudRecording 인터페이스를 호출하여 녹화 파일에 해당하는 정보를 쿼리합니다. 상태가 idle인 경우 멀티미디어 스트림이 업스트림에 업로드되고 있지 않음을 의미하므로 방에 업스트림 호스트가 있는지 확인하십시오.

CreateCloudRecording을 성공적으로 전달된 후, 방에 업스트림 멀티미디어가 있는지 확인하면서 녹화 파일 이름 생성 규칙에 따라 녹화 파일 이름을 스티칭할 수 있습니다. 특정 파일 이름 규칙에 대해서는 위를 참고하십시오(여기에 파일 이름 규칙에 대한 링크가 있습니다).

녹화 파일의 상태는 콜백을 통해 클라이언트의 서버로 전송되며, 해당 콜백을구독하면 녹화 파일의 상태 정보를 받아볼 수 있습니다(여기에 콜백 페이지 링크가 있습니다).

Cos를 통해 녹화 파일 쿼리, 클라우드 녹화 전달 시 cos의 스토리지 디렉터리를 지정할 수 있으며, 녹화 작업 종료 후 해당 디렉터리를 찾아 녹화 파일을 찾을 수 있습니다.

4. 녹화 사용자(userid)의 usersig 만료 시간은 녹화 라이프사이클보다 긴 시간으로 설정해야 합니다. 녹화 작업 기기 네트워크 연결 끊김을 방지하기 위해서 입니다. 내부 고가용성 적용 시 usersig 만료로 인해 녹화 복구에 실패할 수 있습니다.

사용자 정의 컬렉션 및 렌더링 Android&iOS&Windows&Mac

최종 업데이트 날짜: : 2024-07-24 16:43:38

본문에서는 TRTC SDK를 사용하여 사용자 정의 비디오 캡처 및 렌더링을 구현하는 방법을 설명합니다.

사용자 정의 비디오 캡처

TRTC SDK의 사용자 정의 비디오 캡처 기능은 기능 활성화와 SDK로 비디오 프레임 전송의 두 단계로 사용할 수 있습니다. 특정 API에 대한 자세한 안내는 아래를 참고하십시오. 또한 다양한 플랫폼에 대한 API-Example을 제공합니다.

Android

iOS

Windows

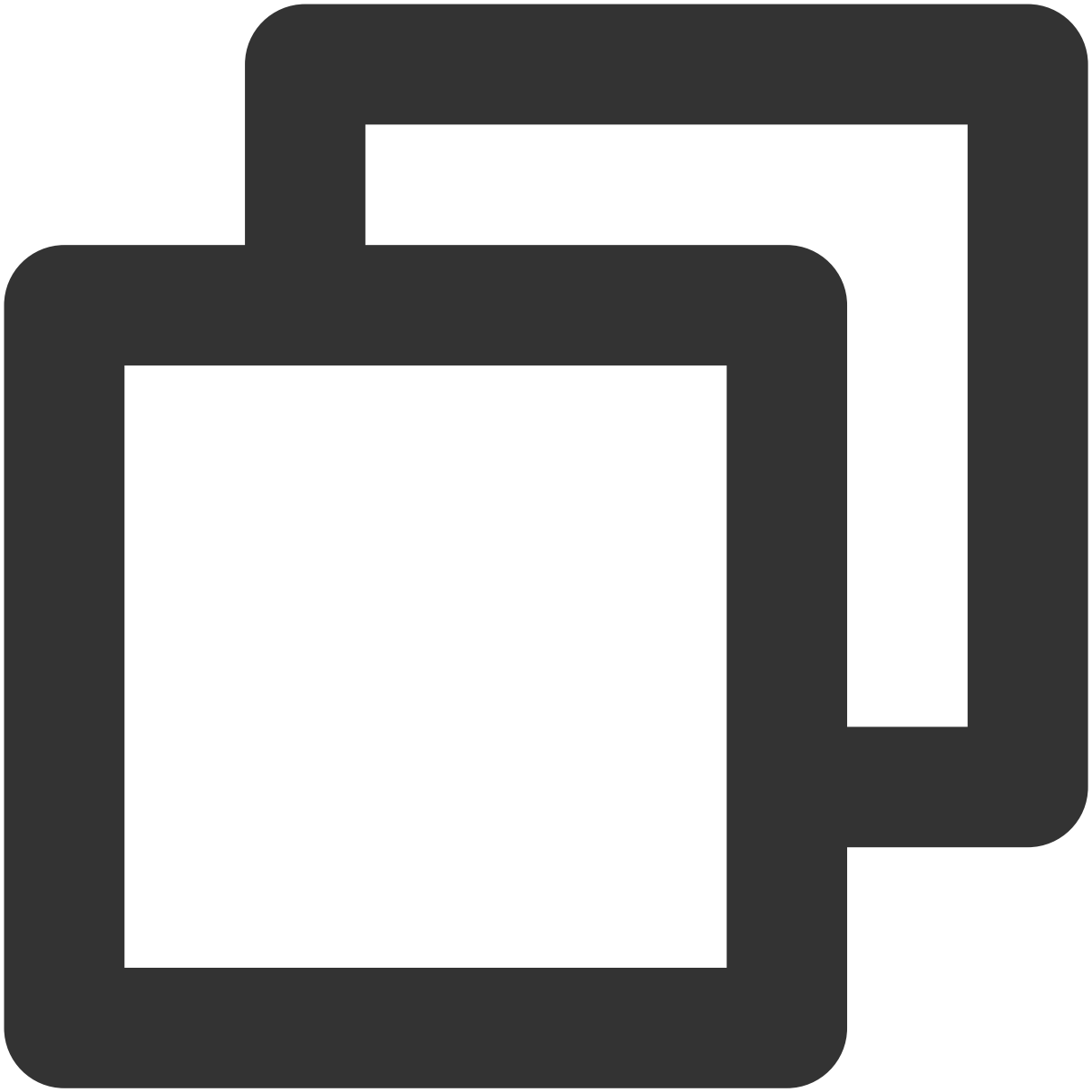
사용자 정의 비디오 캡처 활성화

TRTC SDK의 사용자 정의 비디오 캡처 기능을 활성화하려면 TRTCCloud의 `enableCustomVideoCapture` API를 호출해야 합니다. 그러면 TRTC SDK의 카메라 캡처 및 이미지 처리 로직은 건너뛰고 인코딩 및 전송 기능만 유지됩니다. 아래는 샘플 코드입니다.

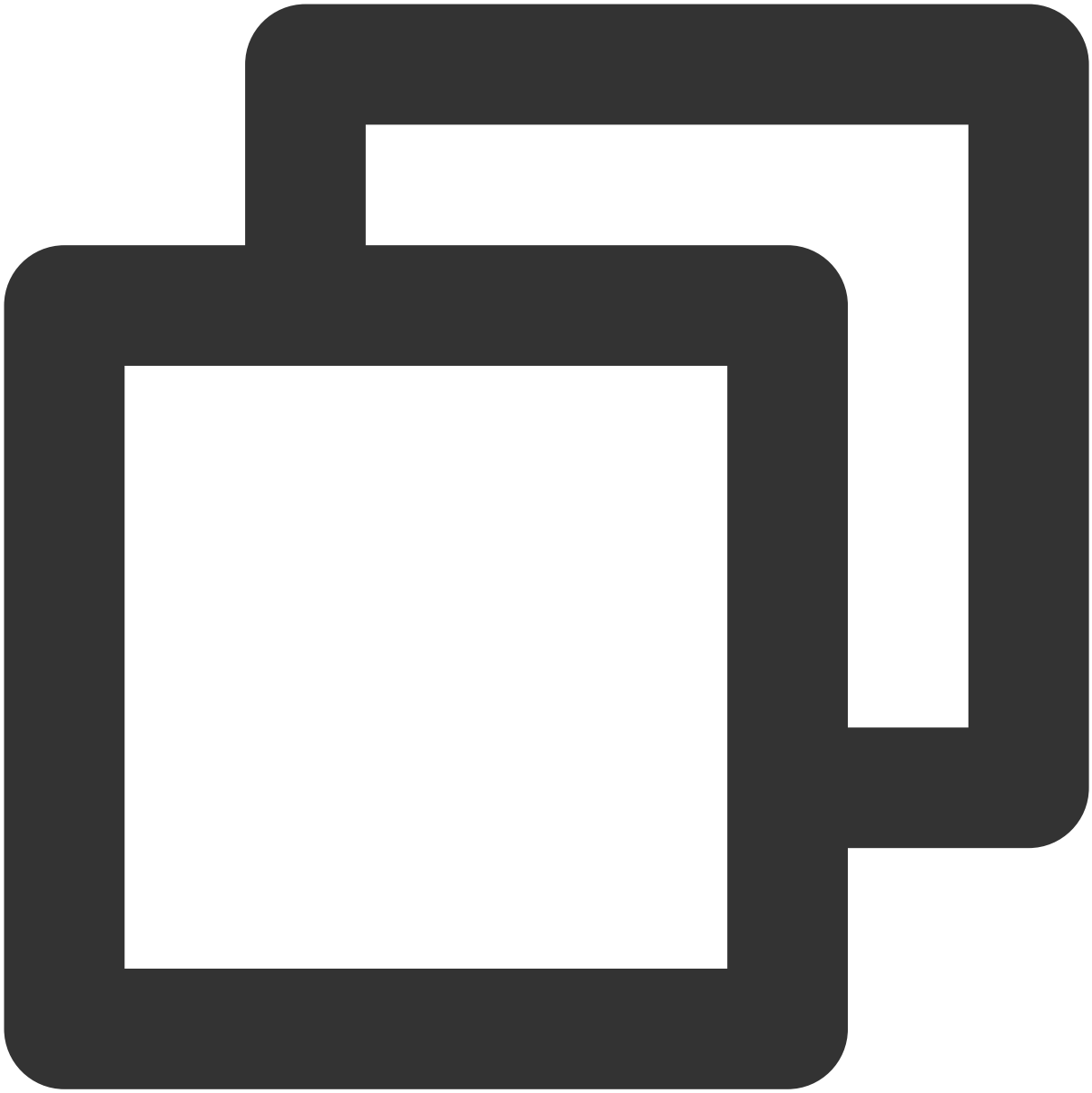
Android

iOS&Mac

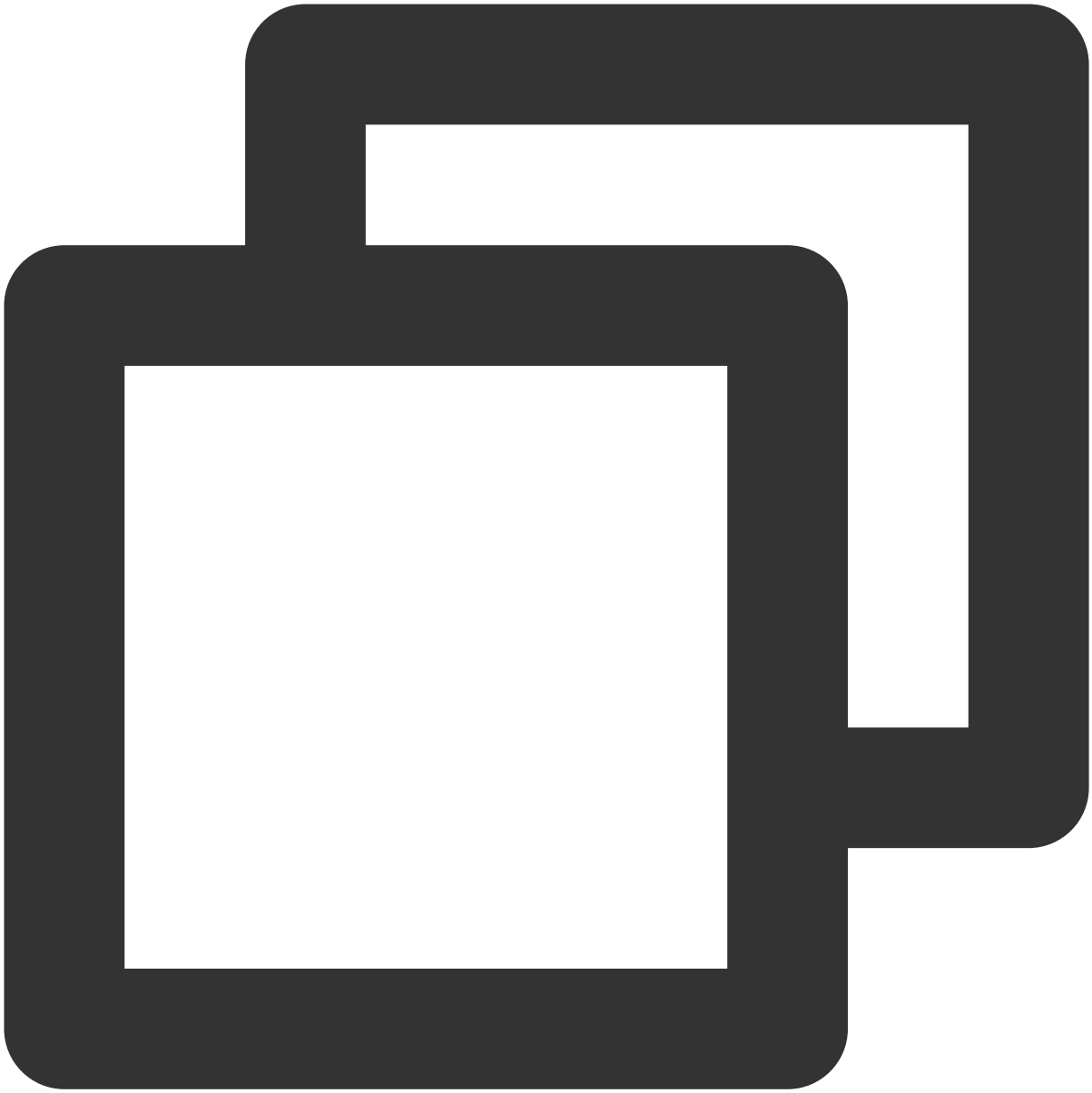
Windows



```
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
```

사용자 정의 비디오 프레임 전송

그런 다음 TRTCCloud의 `sendCustomVideoData` API를 사용하여 TRTC SDK에 자신의 비디오 데이터를 전송할 수 있습니다. 아래는 샘플 코드입니다.

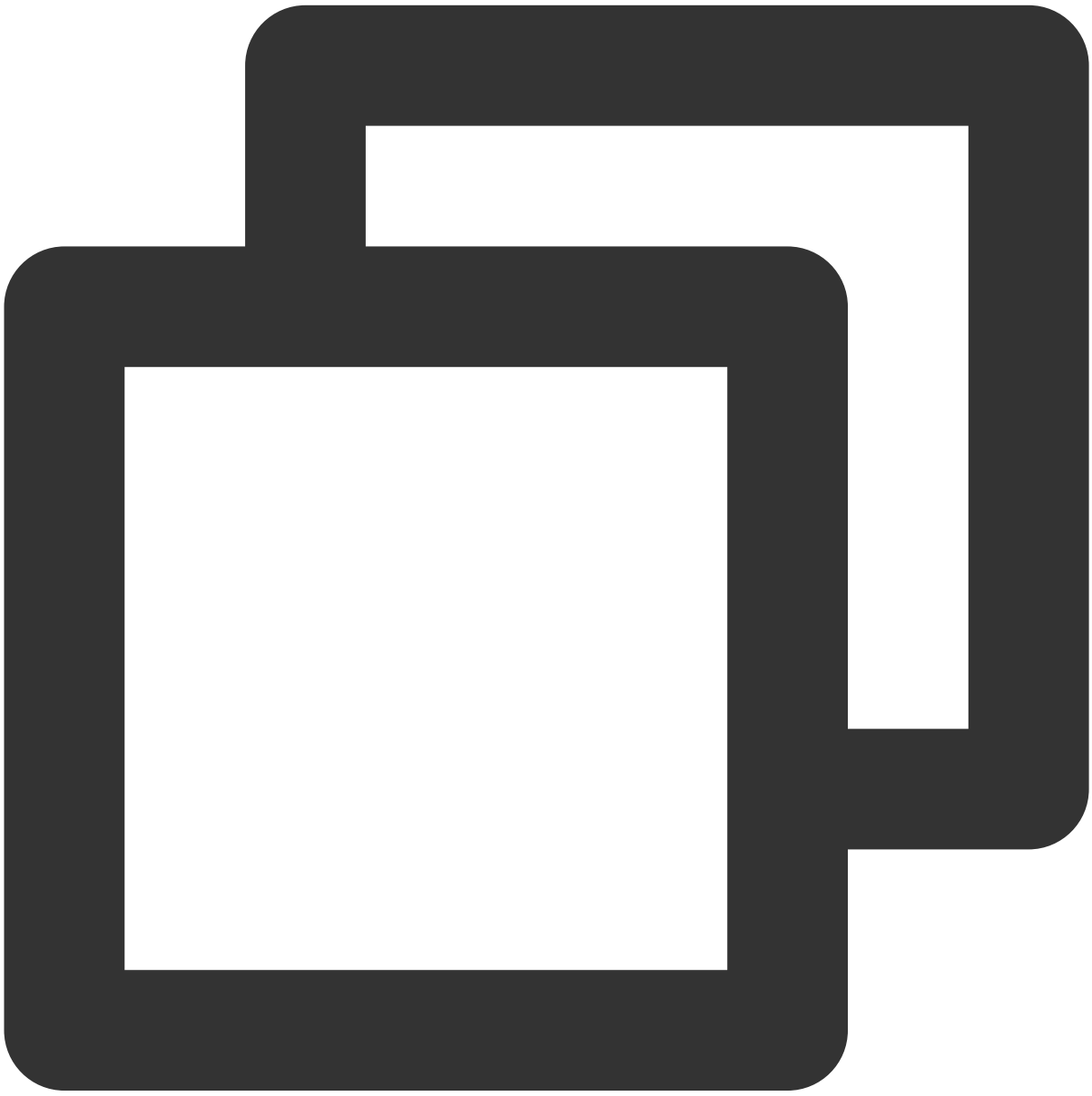
설명：

성능 손실을 방지하기 위해 다양한 플랫폼에서 TRTC SDK에 입력되는 비디오 데이터에 대해 다양한 형식 요구 사항이 있습니다. 자세한 내용은 LiteAVSDK 개요를 참고하십시오.

Android

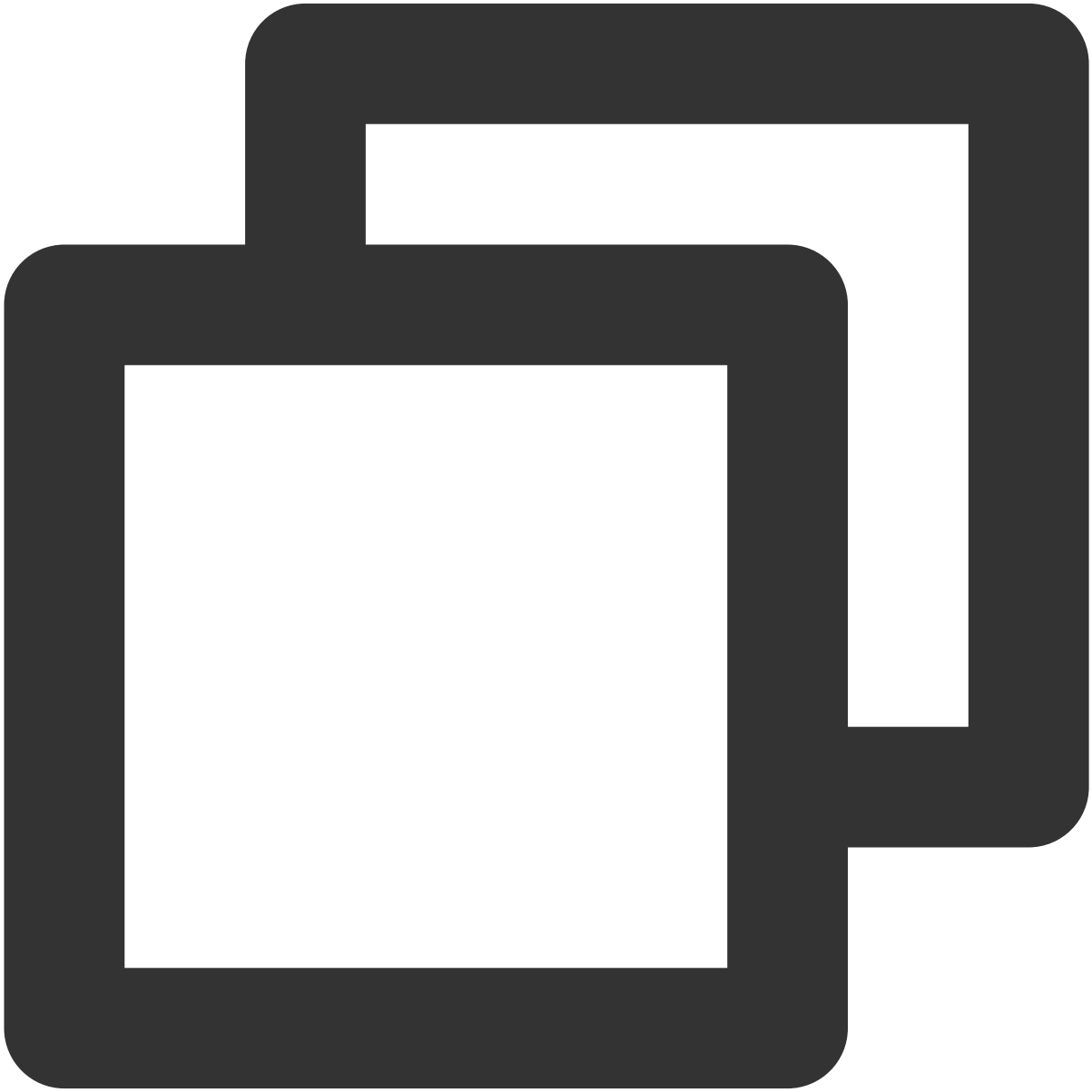
iOS&Mac

Windows



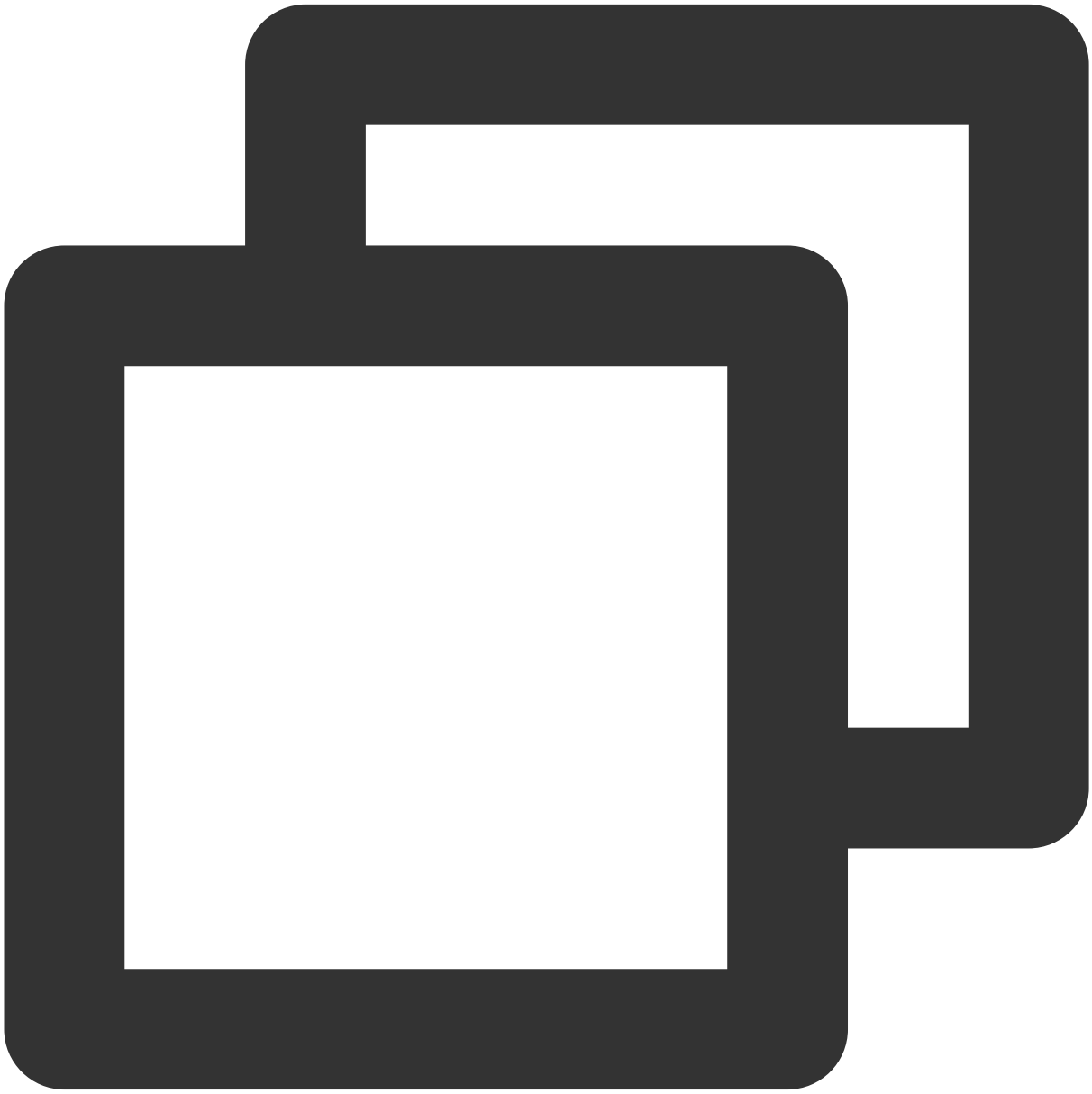
```
// Android에는 Buffer 및 Texture의 두 가지 구성표를 사용할 수 있습니다(권장). 여기에서는 Tex
TRTCCloudDef.TRTCVideoFrame videoFrame = new TRTCCloudDef.TRTCVideoFrame();
videoFrame.texture = new TRTCCloudDef.TRTCTexture();
videoFrame.texture.textureId = textureId;
```

```
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)
```



```
// iOS/Mac에서 카메라로 캡처한 비디오는 NV12 형식이고 기본적으로 지원되는 최고 성능의 비디오 프
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
```



```
// 현재 Windows에서는 Buffer 구성표만 사용할 수 있으며 기능 구현에 권장됩니다.
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);
```

사용자 정의 비디오 렌더링

사용자 정의 랜더링은 크게 로컬 비디오 랜더링과 원격 비디오 랜더링으로 나뉩니다. 로컬/원격 사용자 정의 렌더링에 대한 콜백을 설정할 수 있으며, TRTC SDK는 콜백 함수 `onRenderVideoFrame` 을 통해 해당 비디오 프레임(TRTCVideoFrame)을 전달합니다. 그런 다음 수신된 비디오 프레임의 렌더링을 사용자 정의할 수 있습니다. 이 프로세스에는 OpenGL에 대한 특정 지식이 필요합니다. 또한 다양한 플랫폼에 대한 API-Example을 제공합니다.

Android：

iOS

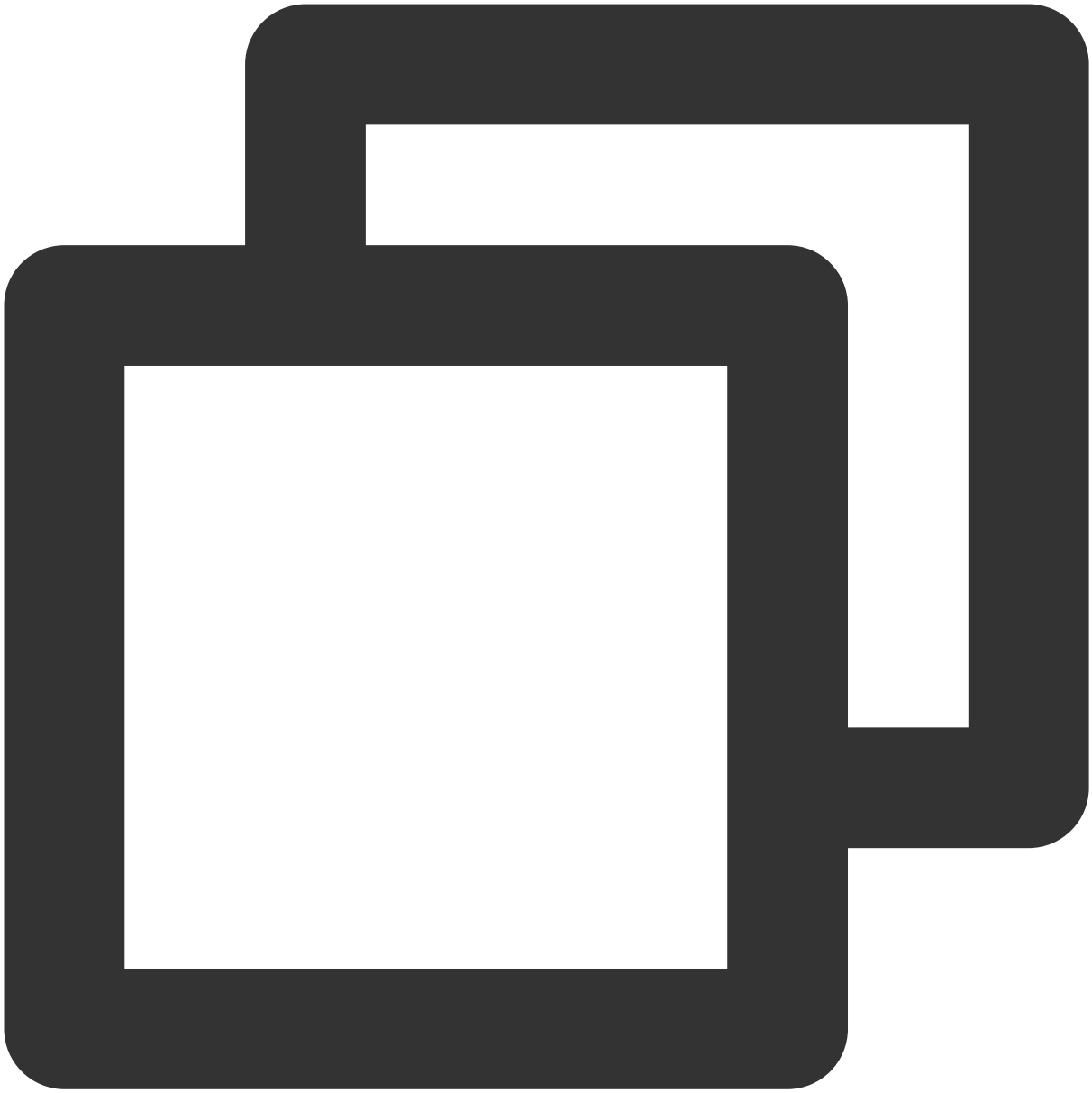
Windows

로컬 비디오 렌더링 콜백 설정

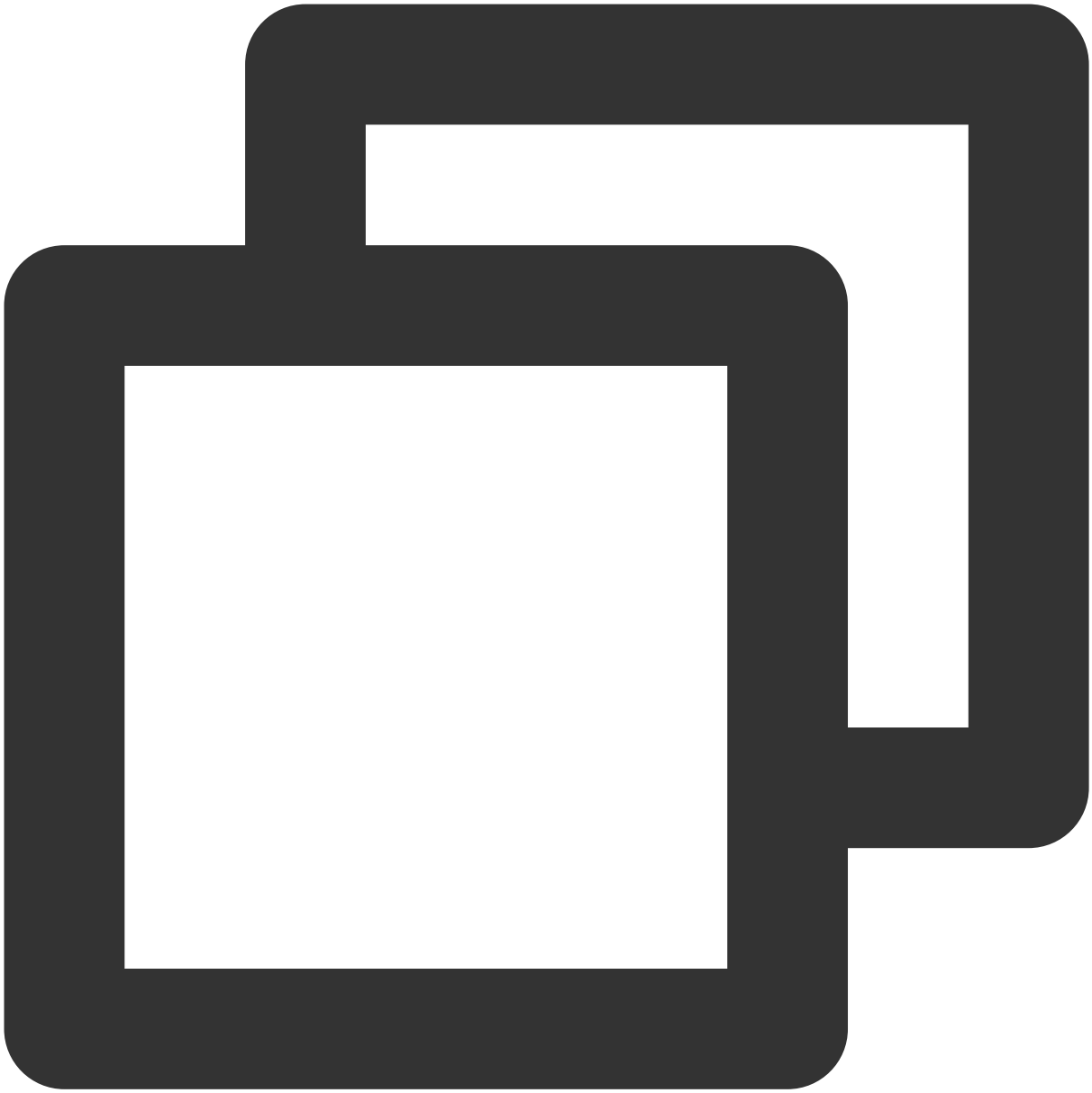
Android

iOS&Mac

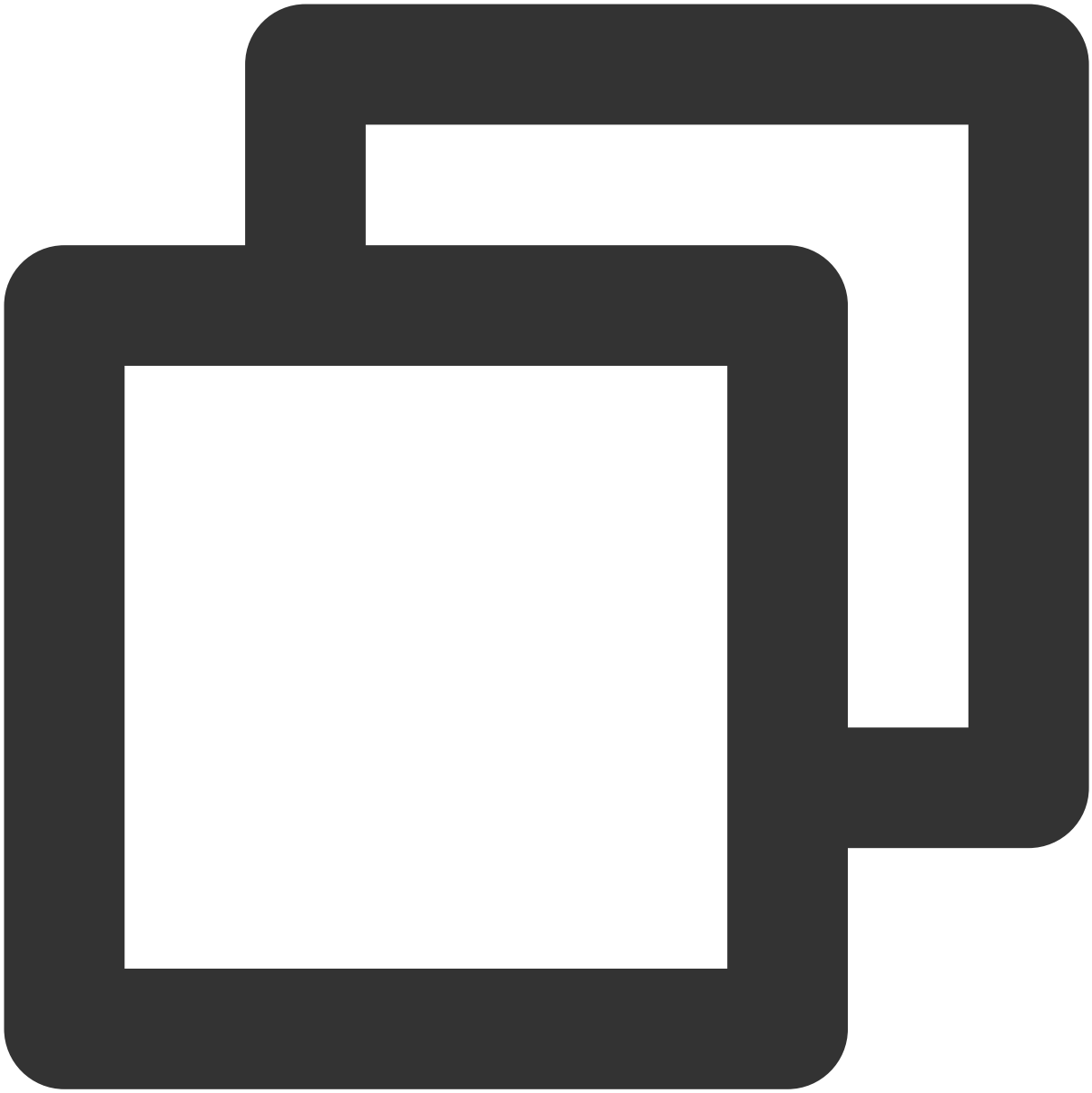
Windows



```
mTRTCCloud.setLocalVideoRenderListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture
    @Override
    public void onRenderVideoFrame(String suserId int streamType, TRTCCloudDef.TRTC
        // 자세한 내용은 TRTC-API-Example의 사용자 정의 렌더링 도구 클래스 com.tencent.trt
    }
});
```



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



```
// 구체적인 구현은 TRTC-API-Example-Qt의 test_custom_render.cpp 참고
void TestCustomRender::onRenderVideoFrame(
    const char* userId,
    liteav::TRTCVideoStreamType streamType,
    liteav::TRTCVideoFrame* frame) {
  if (gl_yuv_widget_ == nullptr) {
    return;
  }

  if (streamType == liteav::TRTCVideoStreamType::TRTCVideoStreamTypeBig) {
```

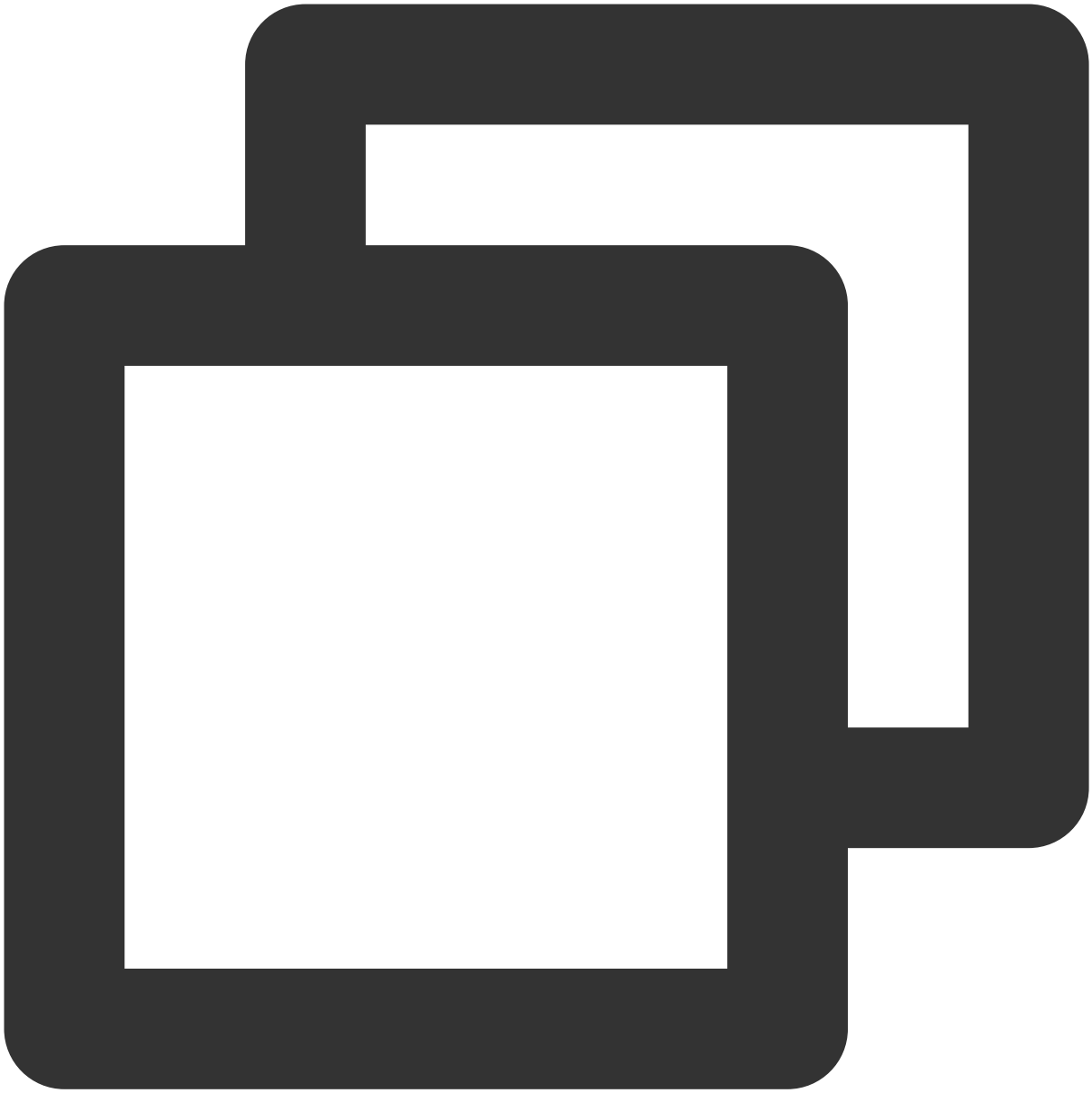
```
    // 렌더링 창 조정
    emit renderViewSize(frame->width, frame->height);
    // 비디오 프레임 그리기
    gl_yuv_widget_->slotShowYuv(reinterpret_cast<uchar*>(frame->data),
                                frame->width, frame->height);
  }
}
```

원격 비디오의 렌더링 콜백 설정

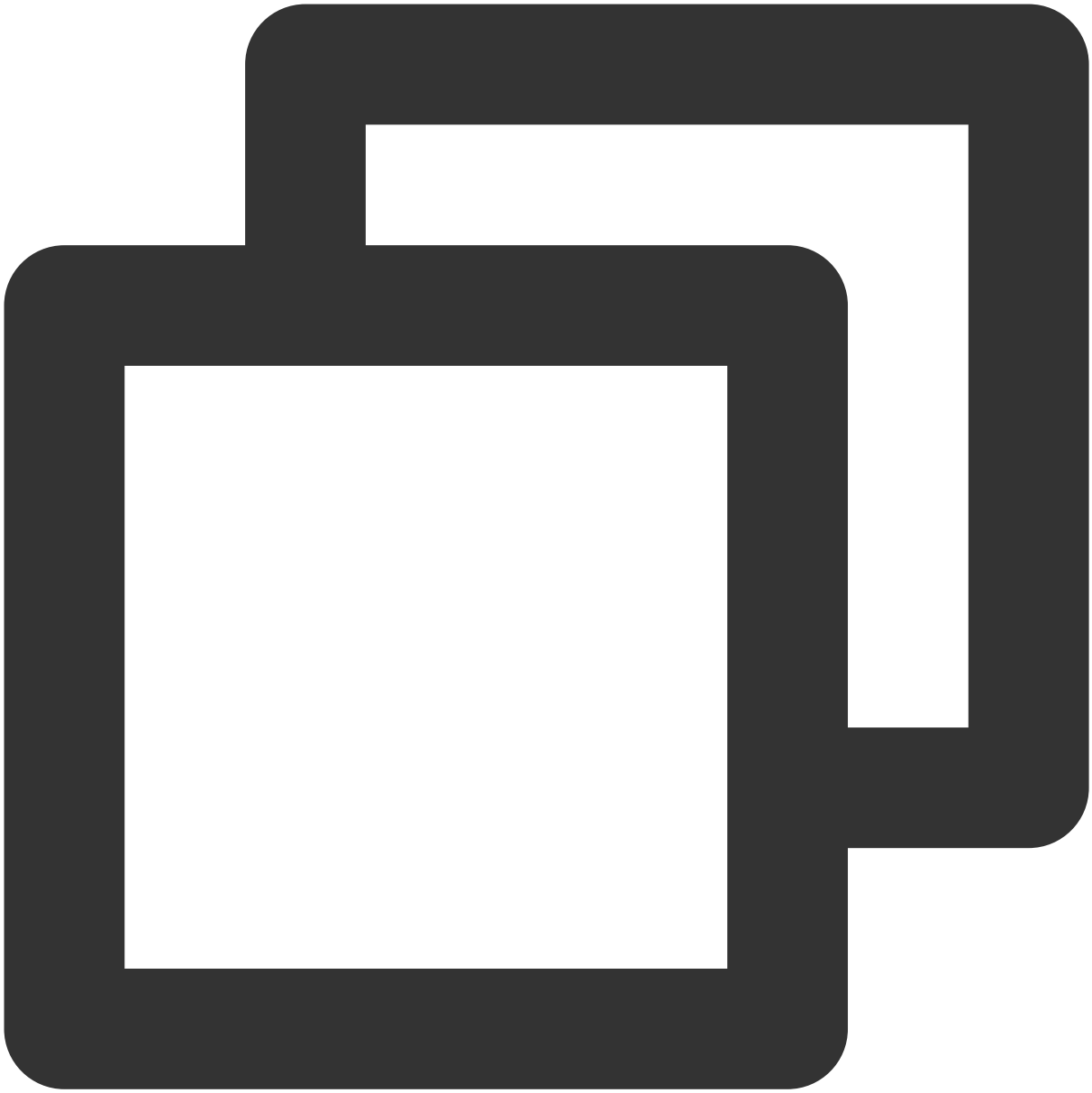
Android

iOS&Mac

Windows

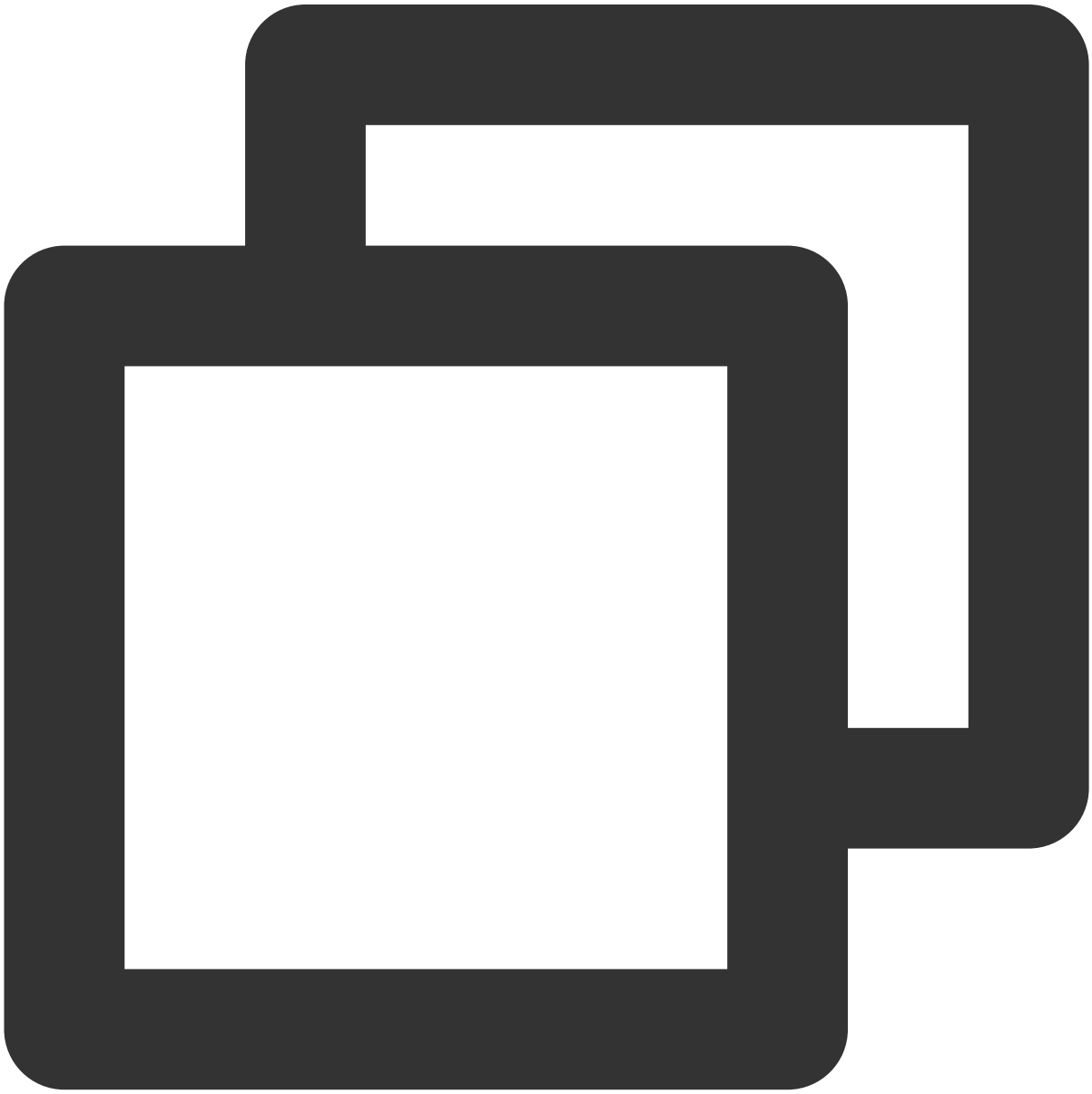


```
mTRTCCloud.setRemoteVideoRenderListener(userId, TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMA
    @Override
    public void onRenderVideoFrame(String userId, int streamType, TRTCCloudDef.TRTC
        // 자세한 내용은 TRTC-API-Example의 사용자 정의 렌더링 도구 클래스 com.tencent.tr
    }
});
```



```
- (void)onRenderVideoFrame:(TRTCVideoFrame *)frame
                    userId:(NSString *)userId
                streamType:(TRTCVideoStreamType)streamType
{
    //userId가 nil이면 로컬 화면, 그렇지 않을 경우 원격 화면
    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);
    });
}
```



```
// 구체적인 구현은 TRTC-API-Example-Qt의 test_custom_render.cpp 참고
void TestCustomRender::onRenderVideoFrame(
    const char* userId,
    liteav::TRTCVideoStreamType streamType,
    liteav::TRTCVideoFrame* frame) {
  if (gl_yuv_widget_ == nullptr) {
    return;
  }

  if (streamType == liteav::TRTCVideoStreamType::TRTCVideoStreamTypeBig) {
    // 렌더링 창 조정
    emit renderViewSize(frame->width, frame->height);
    // 비디오 프레임 그리기
    gl_yuv_widget_->slotShowYuv(reinterpret_cast<uchar*>(frame->data),
                                frame->width, frame->height);
  }
}
```

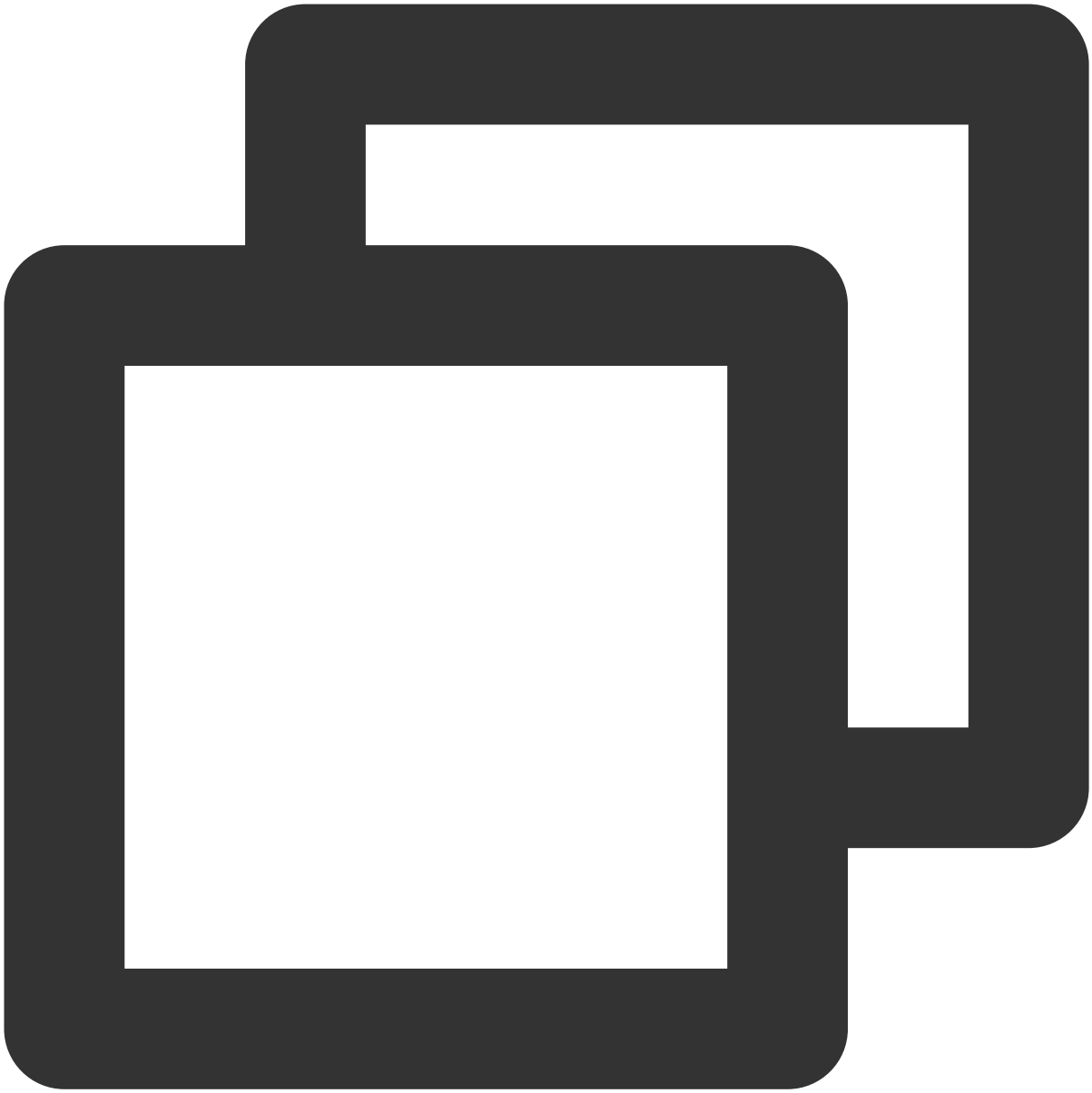
Web

최종 업데이트 날짜: : 2024-07-24 16:43:38

본문에서는 오디오와 비디오를 직접 캡처하고 렌더링하는 방법을 설명합니다.

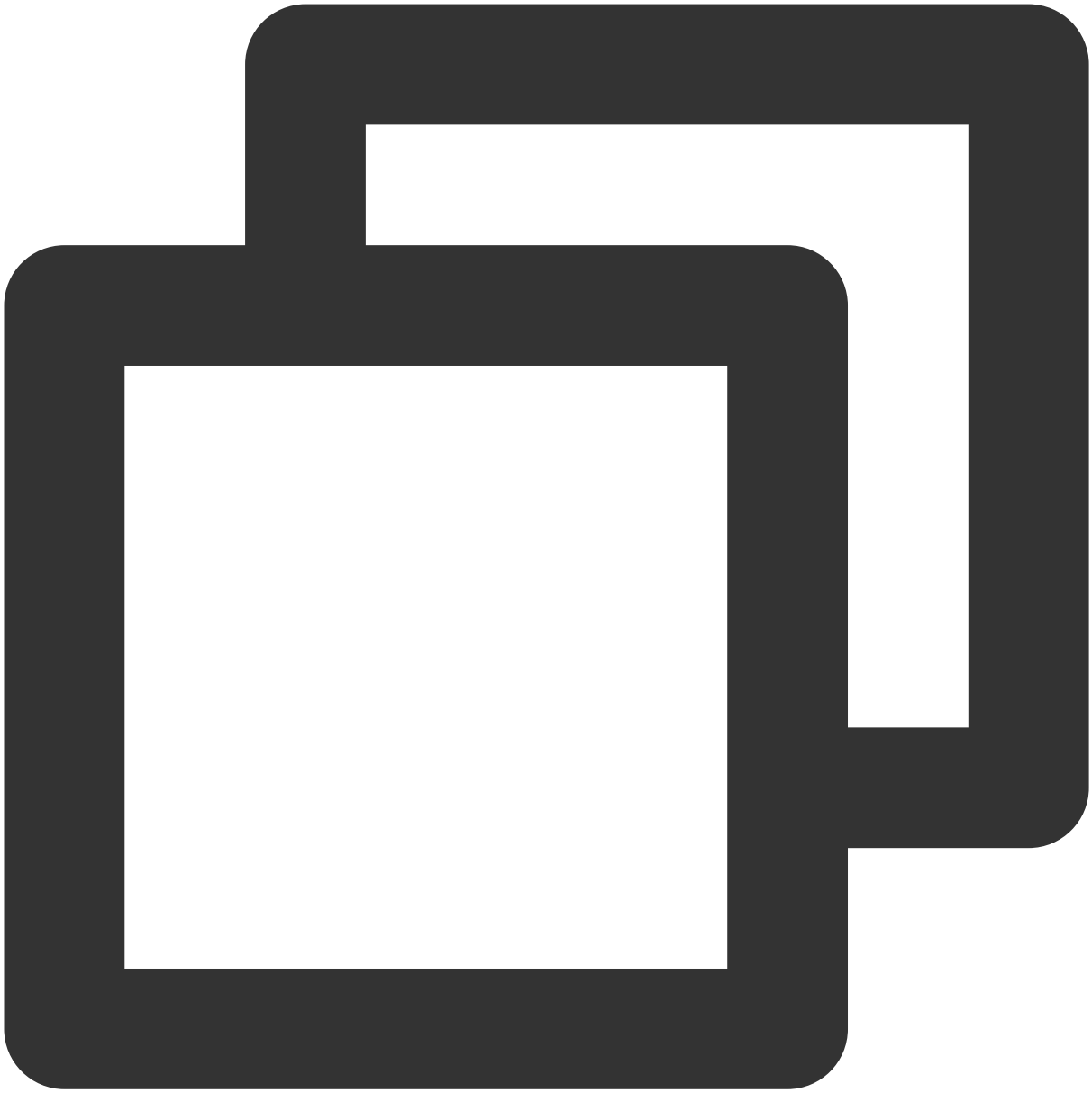
사용자 정의 캡처

로컬 스트림을 생성하기 위해 {@link TRTC.createStream createStream()}을 호출할 때 캡처 방법을 지정할 수 있습니다.
마이크와 카메라에서 오디오 및 비디오 데이터 캡처:



```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
localStream.initialize().then(() => {
  // local stream initialized success
});
```

화면 공유 스트림 캡처:

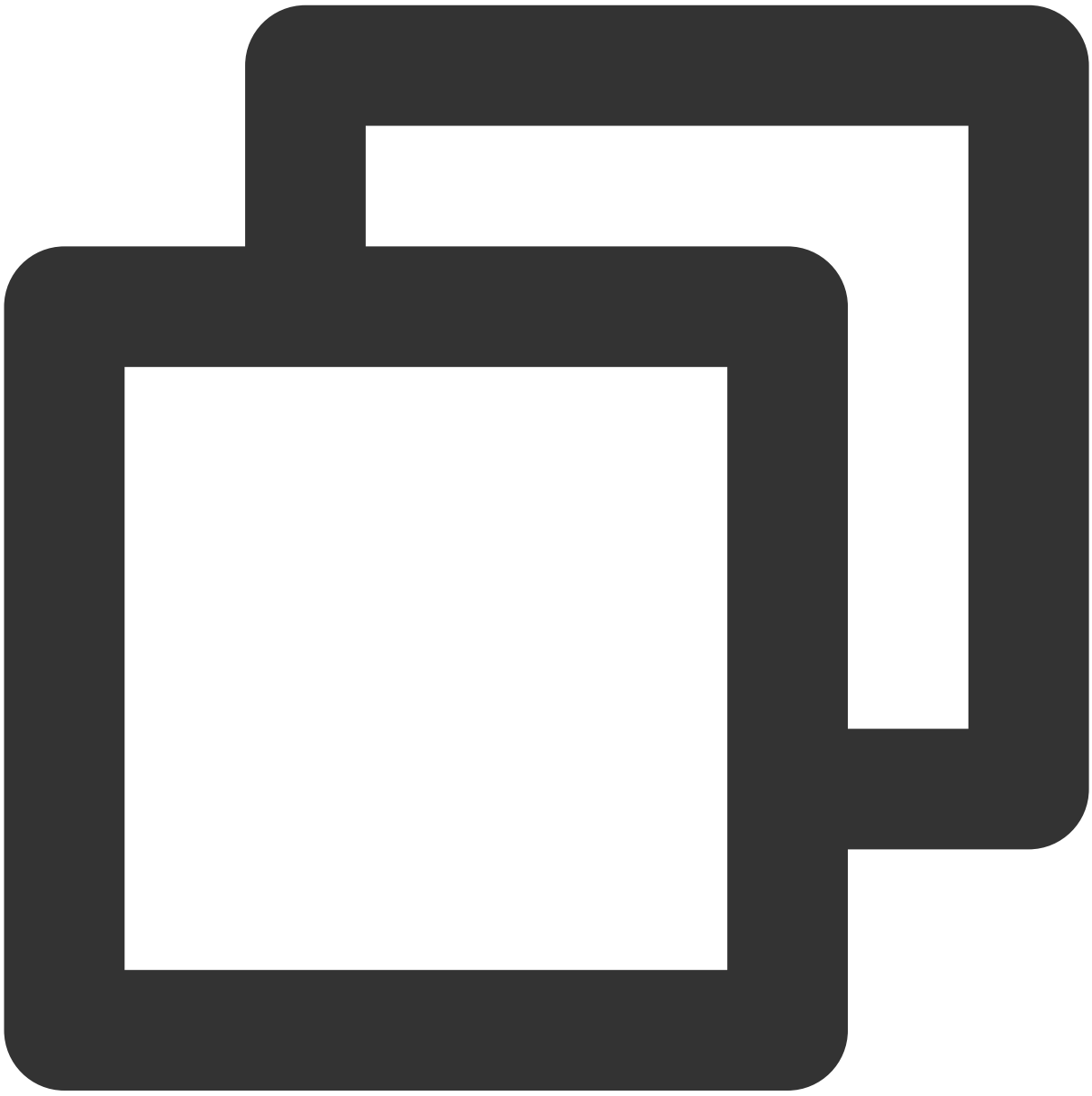


```
const localStream = TRTC.createStream({ userId, audio: false, screen: true });
localStream.initialize().then(() => {
  // local stream initialized success
});
```

상기 두 가지 예시는 SDK에 내장된 캡처 프로세스를 사용합니다. 스트림을 사전 처리하려면 createStream을 사용하여 외부 오디오 및 비디오 소스, 즉 사용자 정의 캡처를 사용하여 로컬 스트림을 생성할 수 있습니다. 예를 들어 다음을 수행할 수 있습니다.
getUserMedia를 사용하여 특정 마이크 및 카메라에서 오디오 및 비디오를 캡처합니다.
getDisplayMedia를 사용하여 화면 콘텐츠를 캡처합니다.

captureStream을 사용하여 웹페이지에서 재생되는 오디오 및 비디오를 캡처합니다.
captureStream을 사용하여 canvas에서 애니메이션을 캡처합니다.

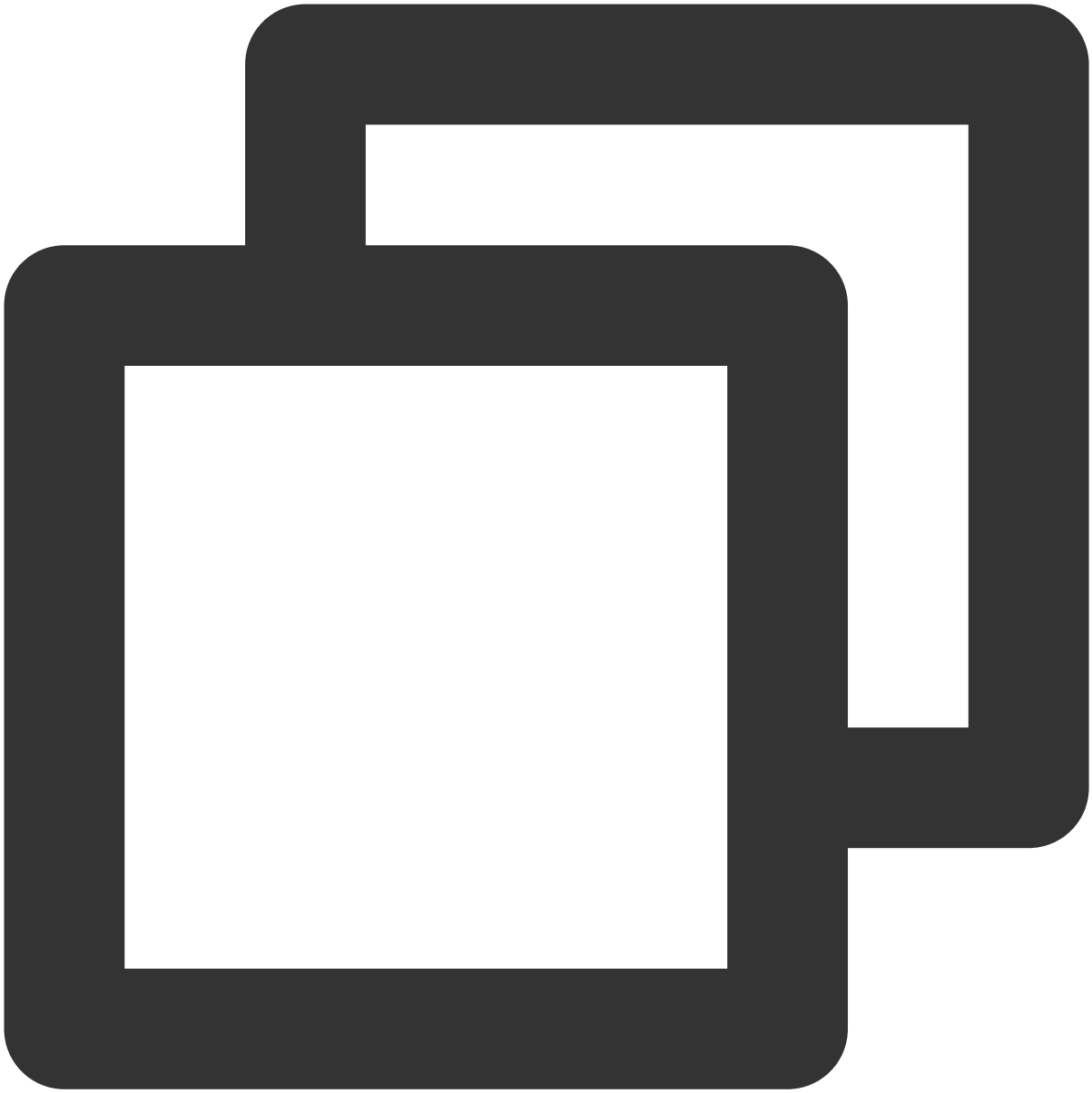
웹 페이지에서 재생되는 비디오 소스 캡처



```
// 현재 브라우저가 video 요소에서 stream 캡처를 지원하는지 확인
const isVideoCapturingSupported = () => {
    ['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) =>
        if (item in document.createElement('video')) {
            return true;
        }
```

```
    });
    return false;
};

// 현재 브라우저가 video 요소에서 stream 캡처를 지원하는지 확인
if (!isVideoCapturingSupported()) {
    console.log('your browser does not support capturing stream from video element'
    return
}
// 페이지에서 재생 중인 video의 태그 가져오기
const video = document.getElementByID('your-video-element-ID');
// 재생중인 비디오에서 비디오 스트림 캡처
const stream = video.captureStream();
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];

const localStream = TRTC.createStream({ userId, audioSource: audioTrack, videoSourc

// 영상 통화 경험을 보장하기 위해 비디오 속성은 외부 비디오 소스의 속성과 동일해야 함
localStream.setVideoProfile('480p');

localStream.initialize().then(() => {
    // local stream initialized success
});
```

canvas에서 애니메이션 캡처하기



```
// 현재 브라우저가 canvas 요소에서 stream 캡처를 지원하는지 확인
const isCanvasCapturingSupported = () => {
    ['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) =>
        if (item in document.createElement('canvas')) {
            return true;
        }
    });
    return false;
};

// 현재 브라우저가 canvas 요소에서 stream 캡처를 지원하는지 확인
```

```
if (!isCanvasCapturingSupported()) {
    console.log('your browser does not support capturing stream from canvas element
    return
}
// canvas 태그 가져오기
const canvas = document.getElementByID('your-canvas-element-ID');

// canvas에서 15 fps 비디오 스트림 캡처
const fps = 15;
const stream = canvas.captureStream(fps);
const videoTrack = stream.getVideoTracks()[0];

const localStream = TRTC.createStream({ userId, videoSource: videoTrack });

// 영상 통화 경험을 보장하기 위해 비디오 속성은 외부 비디오 소스의 속성과 동일해야 함
localStream.setVideoProfile('480p');

localStream.initialize().then(() => {
    // local stream initialized success
});
```

사용자 정의 렌더링

{@link Stream#play Stream.play()}를 사용하여 TRTC.createStream()을 통해 생성 및 초기화된 로컬 스트림 또는 Client.on('stream-added')을 통해 수신된 원격 스트림을 렌더링할 수 있습니다. Stream.play() 메소드는 오디오 플레이어와 비디오 플레이어를 만들고 <audio>/<video> 태그를 App이 전달한 Div 컨테이너에 삽입합니다.

자신의 플레이어를 사용하려면 Stream.play()/stop()을 호출하는 대신 {@link Stream#getAudioTrack Stream.getAudioTrack()}/{@link Stream#getVideoTrack Stream.getVideoTrack()} 오디오 및 비디오 트랙을 가져오고 자신의 플레이어로 렌더링합니다. 이러한 경우 Stream.on('player-state-changed') 콜백이 트리거되지 않습니다. 현재 스트림의 상태에 대한 정보를 얻으려면 App이 'mute/unmute/ended' 및 MediaStreamTrack의 기타 이벤트를 수신해야 합니다.

또한 스트림의 라이프사이클을 모니터링하기 위해 `Client.on('stream-added')`, `Client.on('stream-updated')` 및 `Client.on('stream-removed')`을 수신해야 합니다.

주의사항 :

'stream-added' 또는 'stream-updated' 이벤트 콜백을 수신한 후 스트림에 오디오 또는 비디오 track이 있는지 확인합니다. stream-updated 콜백에 대한 오디오 또는 비디오 track이 있는 경우 플레이어를 업데이트하고 최신 오디오 및 비디오 track을 재생해야 합니다.

온라인 데모

오디오 캡처 및 재생 사용자 정의 Android&iOS&Windows&Mac

최종 업데이트 날짜: : 2024-07-24 16:43:38

본문에서는 사용자 정의 오디오 캡처 및 렌더링을 구현하기 위해 TRTC SDK를 사용하는 방법을 설명합니다.

사용자 정의 오디오 캡처

TRTC SDK의 사용자 정의 오디오 캡처 기능은 기능 활성화와 SDK에 오디오 프레임 전송의 두 단계로 사용할 수 있습니다. 특정 API에 대한 자세한 안내는 아래를 참고하십시오. 또한 다양한 플랫폼에 대한 API-Example을 제공합니다.

Android

iOS

Windows

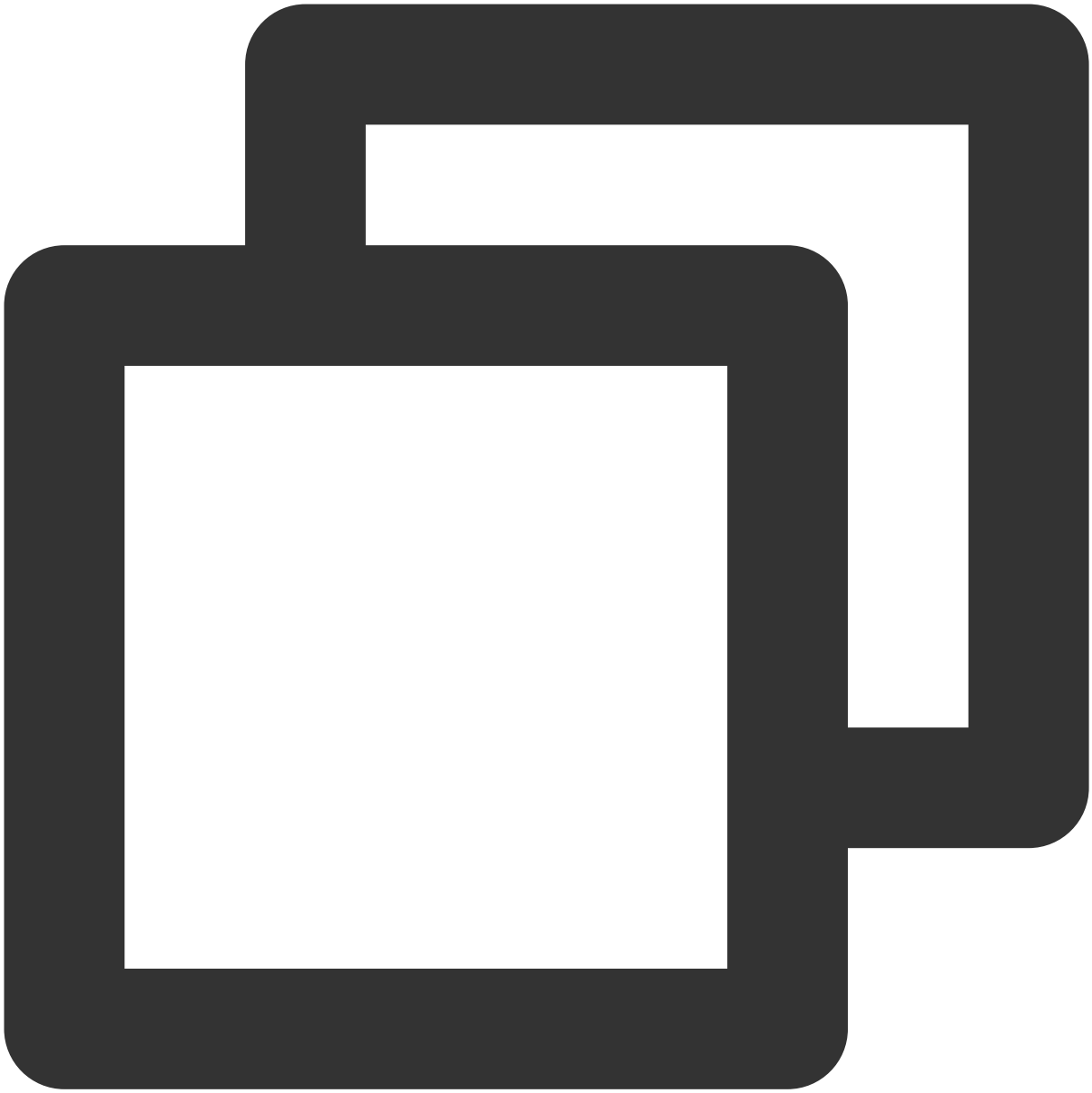
사용자 정의 오디오 캡처 활성화

TRTC SDK의 사용자 정의 오디오 캡처 기능을 활성화하려면 TRTCCloud의 `enableCustomAudioCapture` API를 호출해야 합니다. 아래는 샘플 코드입니다.

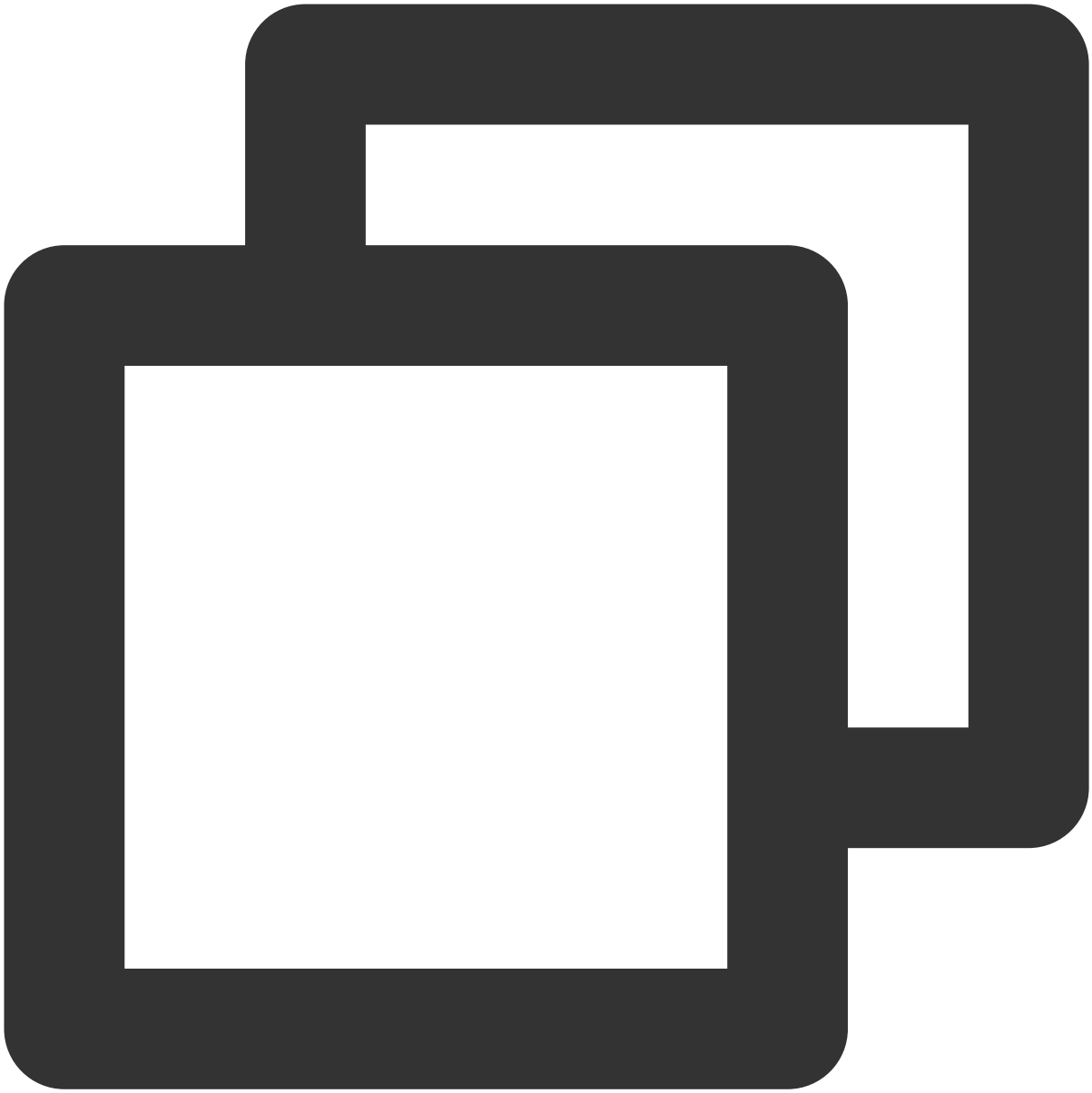
Android

iOS&Mac

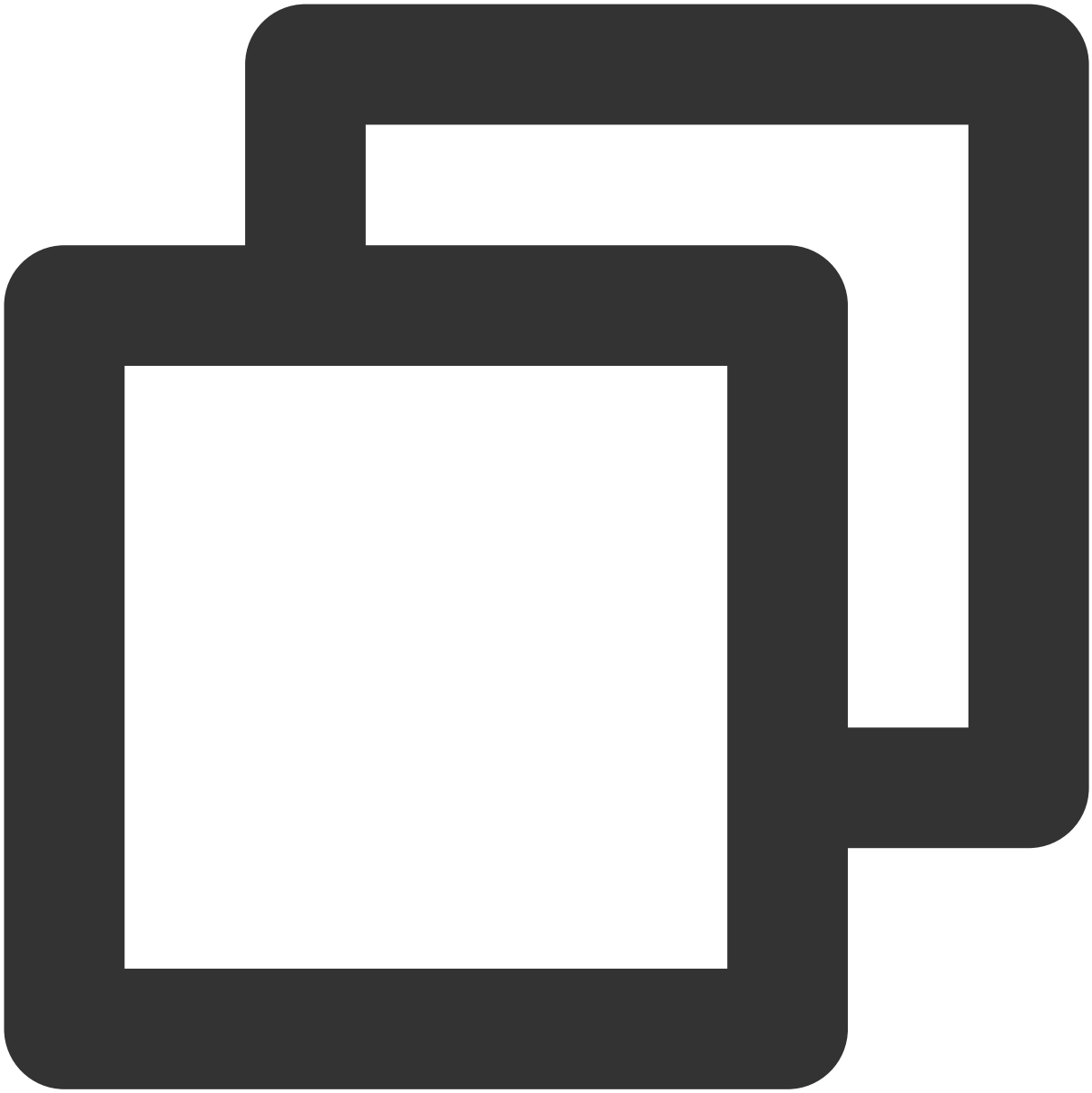
Windows



```
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);
```

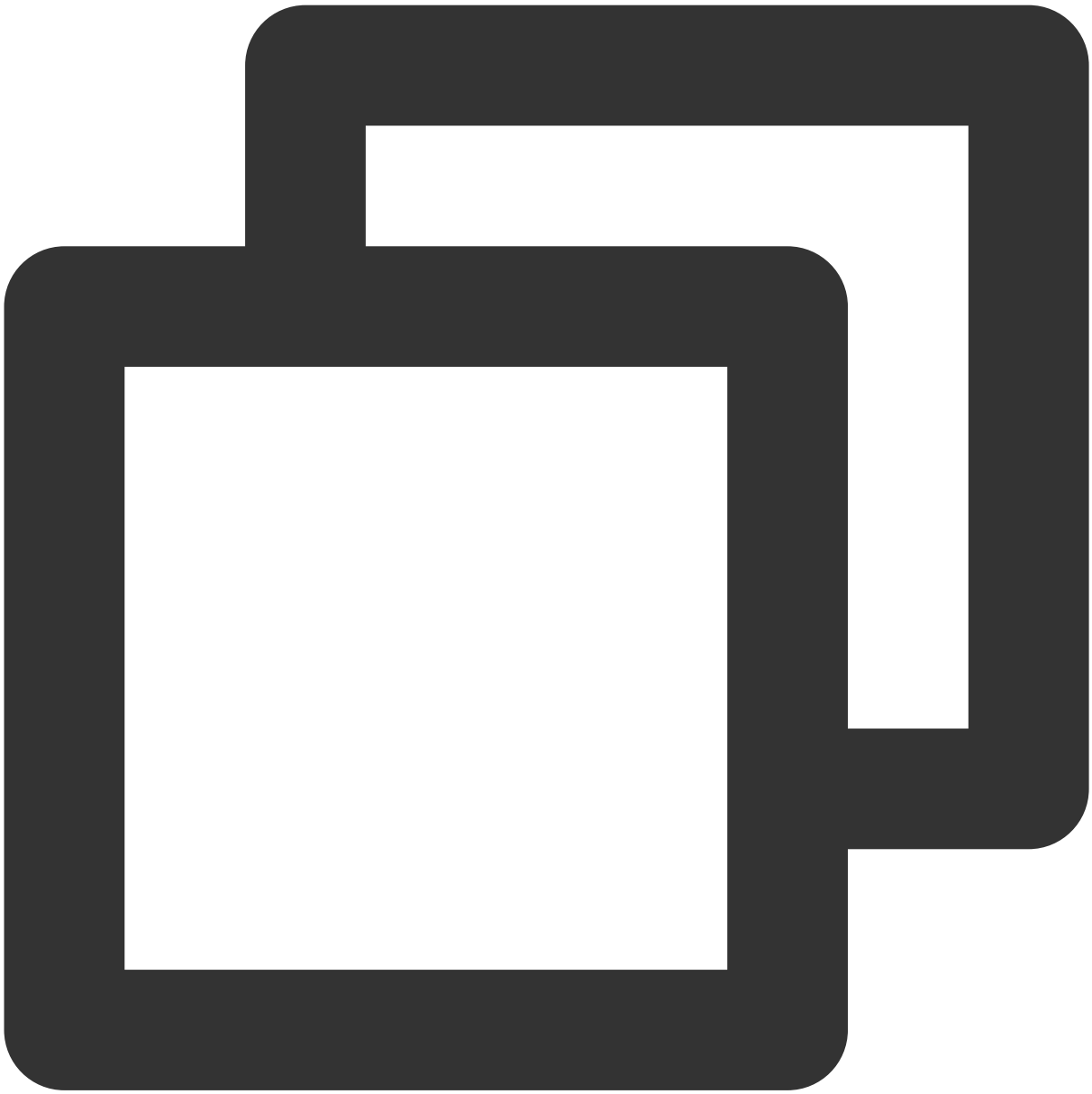
사용자 정의 오디오 프레임 전송

TRTCCloud의 'sendCustomAudioData' API를 사용하여 자신의 오디오 데이터로 TRTC SDK를 채울 수 있습니다. 아래는 샘플 코드입니다.

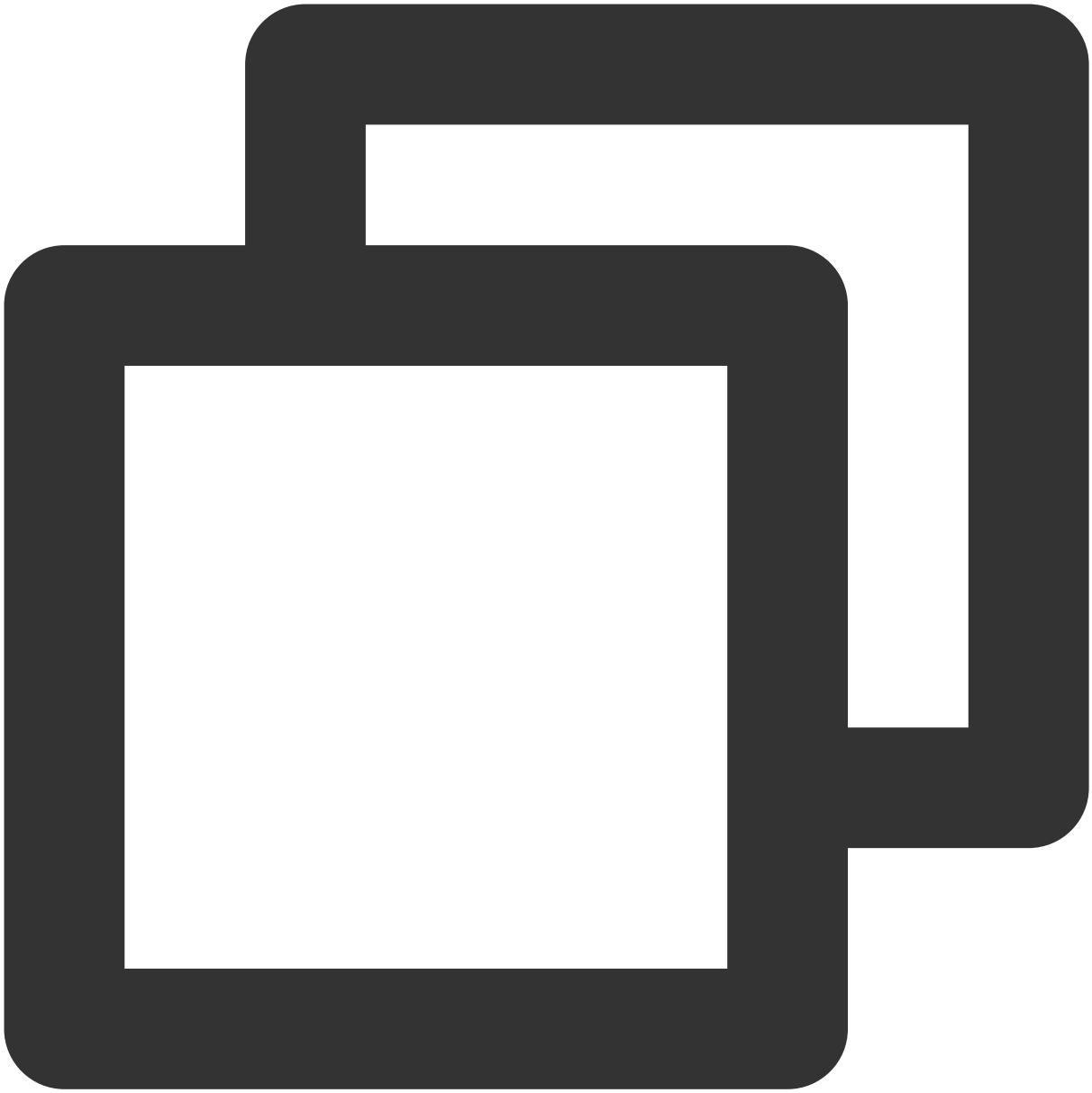
Android

iOS&Mac

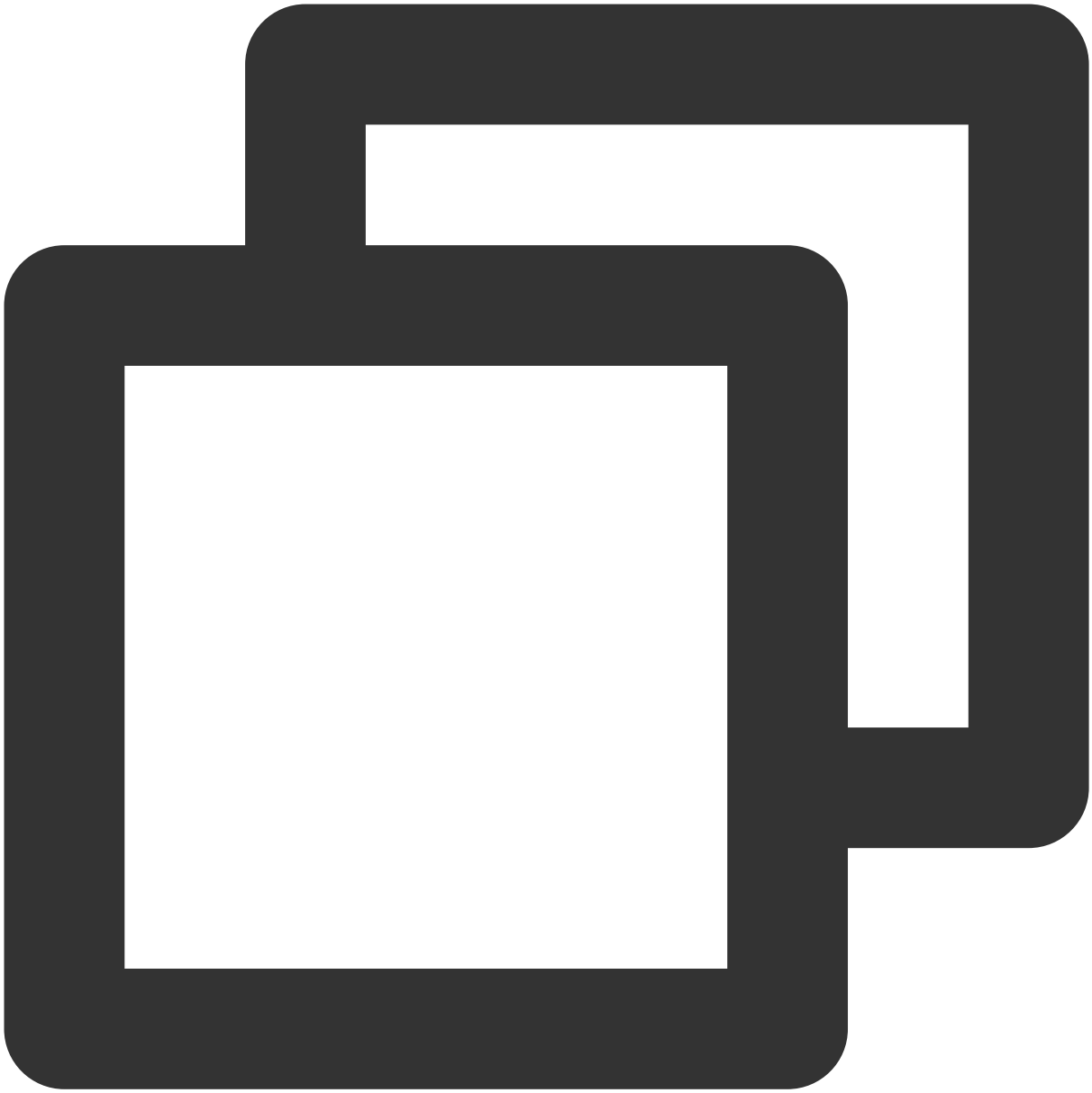
Windows



```
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);
```

주의사항 :

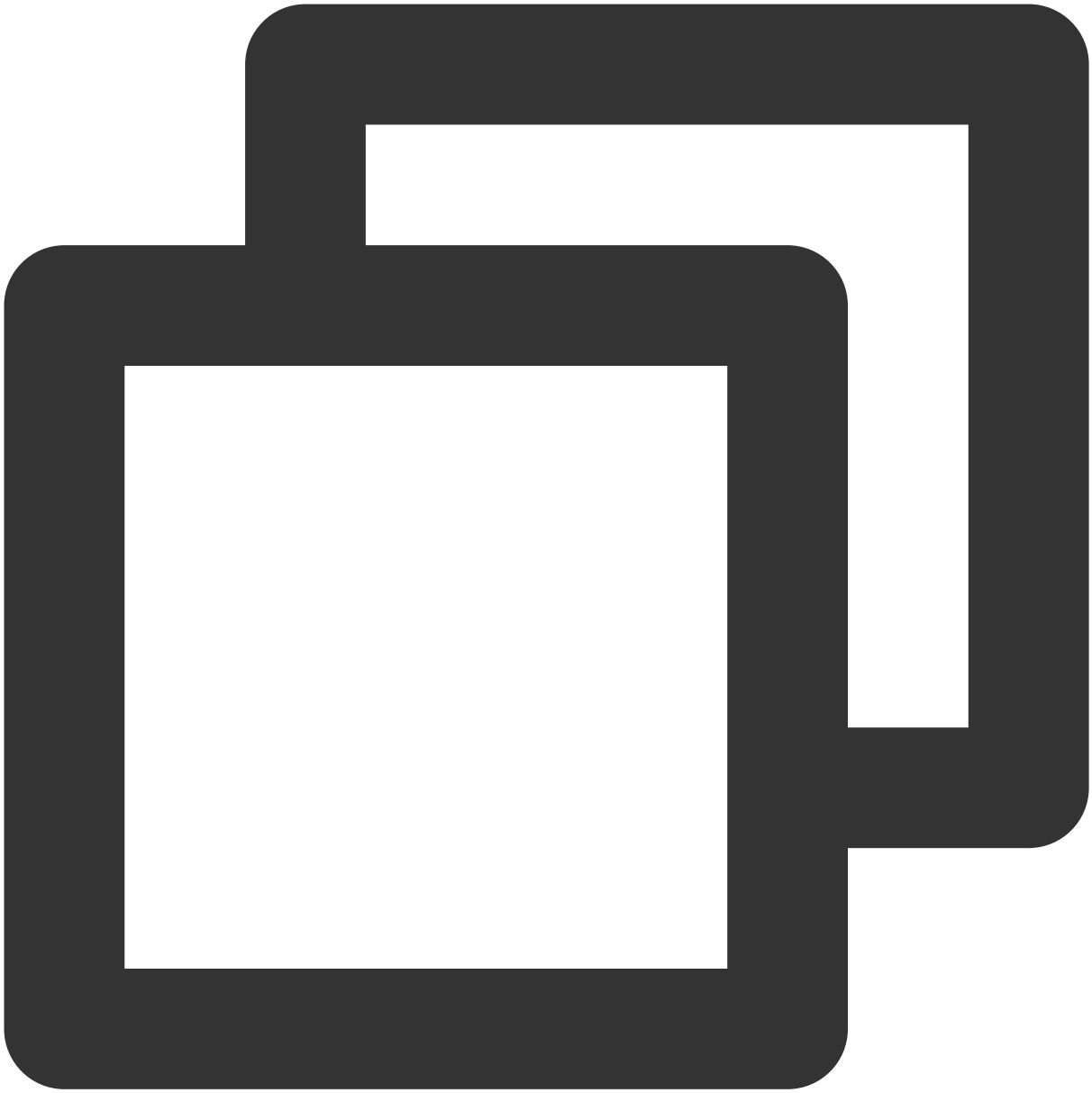
`sendCustomAudioData` 를 사용할 경우 반향 제거(AEC) 기능이 적용되지 않을 수 있습니다.

오디오 원본 데이터 획득

오디오 모듈은 매우 복잡한 모듈이며 TRTC SDK는 오디오 장치의 캡처 및 재생 로직을 엄격하게 제어해야 합니다. 경우에 따라 원격 사용자의 오디오 데이터 또는 로컬 마이크에 의해 캡처된 오디오 데이터를 가져오기 위해 다른 플랫폼용 TRTCCloud의 API를 사용할 수 있습니다. 또한 다음과 같은 플랫폼에 대한 API-Example도 제공합니다.
Android：
iOS
Windows

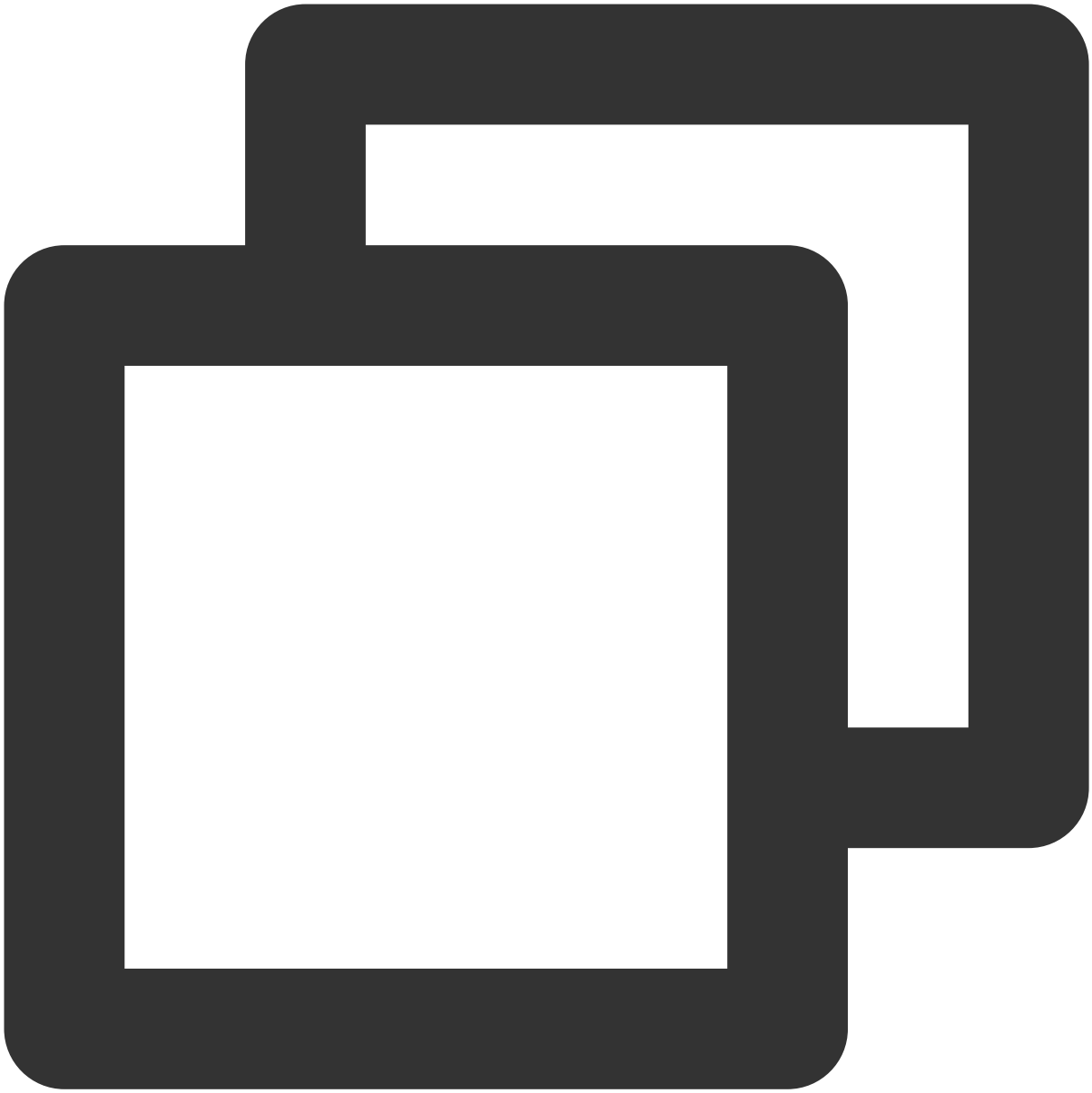
오디오 콜백 설정

Android
iOS&Mac
Windows



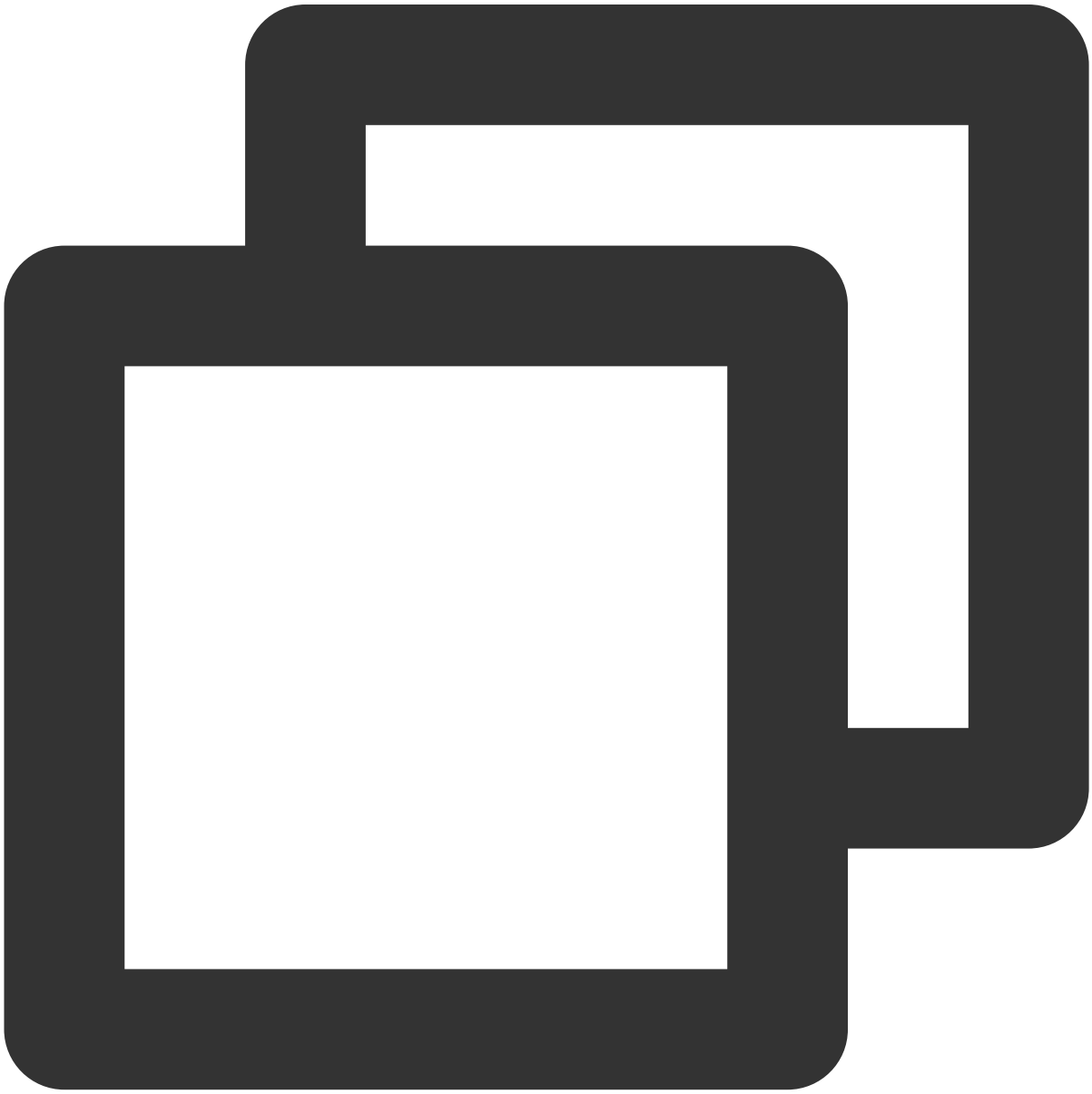
```
mTRTCCloud.setAudioFrameListener(new TRTCCloudListener.TRTCAudioFrameListener() {
        @Override
        public void onCapturedRawAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFr

        }

        @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
            // 자세한 내용은 TRTC-API-Example의 사용자 정의 렌더링 도구 클래스 com.tencent
        }
    });
```



```
 [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");
}
```



```
// 사용자 정의 오디오 데이터 콜백 설정
liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance();
trtc_cloud->setAudioFrameCallback(callback)

// 오디오 데이터 사용자 정의 콜백

virtual void onCapturedRawAudioFrame(TRTCAudioFrame* frame) {
}

virtual void onLocalProcessedAudioFrame(TRTCAudioFrame* frame) {
}

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

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

주의사항

상기 콜백 함수는 시간이 소요되는 어떠한 작업도 해서는 안 되며, 직접 복사하여 다른 스레드를 통해 처리할 것을 권장합니다. 그렇지 않을 경우 오디오가 끊기거나 반향 제거(AEC) 기능이 적용되지 않을 수 있습니다.
상기 콜백 함수에서 콜백하는 데이터는 모두 읽기 및 복사가 가능하며 수정은 불가능합니다. 수정할 경우 알 수 없는 결과가 나타날 수 있습니다.

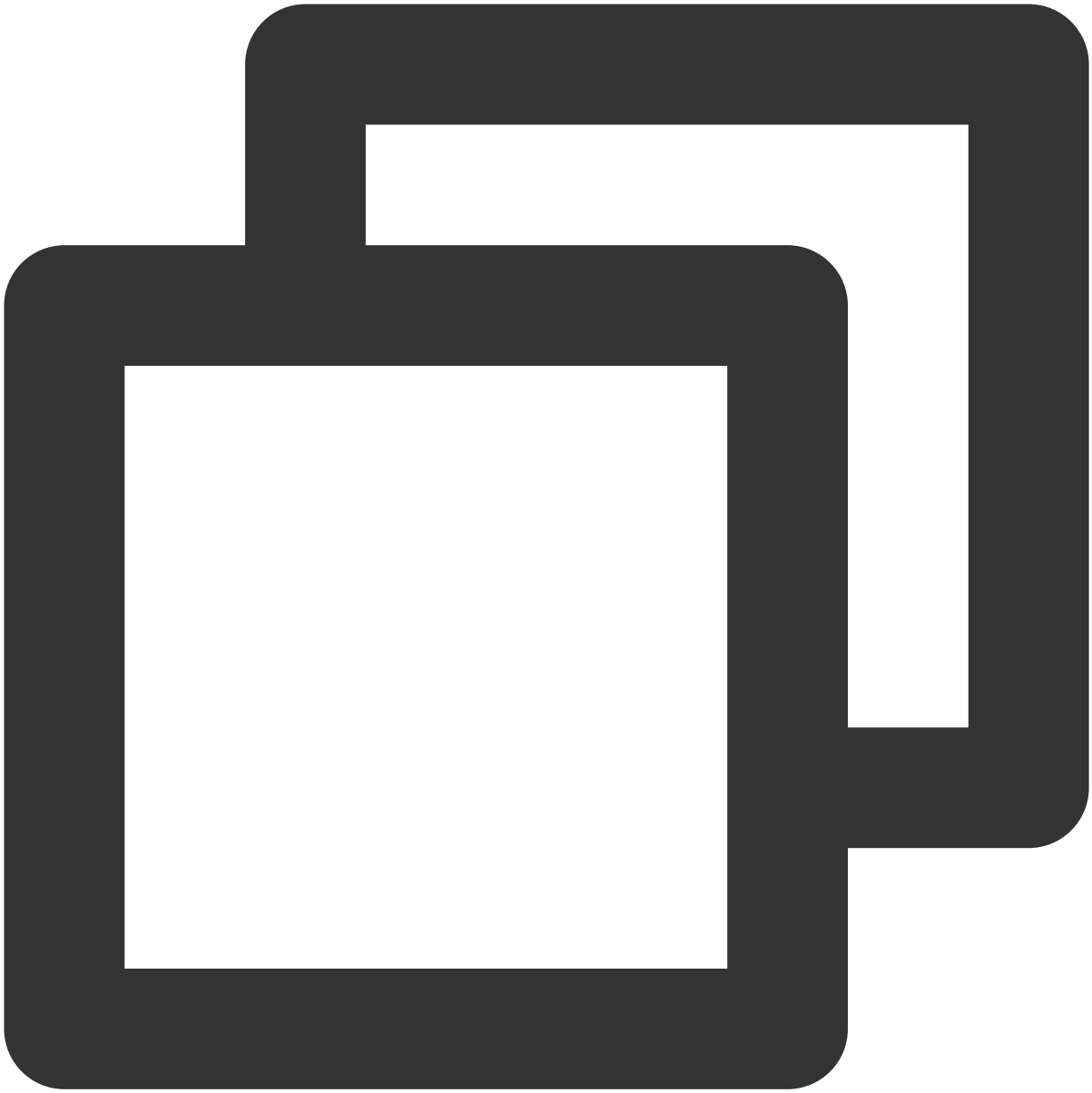
Web

최종 업데이트 날짜: : 2024-07-24 16:43:38

본문에서는 오디오 및 비디오에 대한 사용자 정의 캡처 및 사용자 정의 렌더링을 사용하는 방법을 설명합니다.

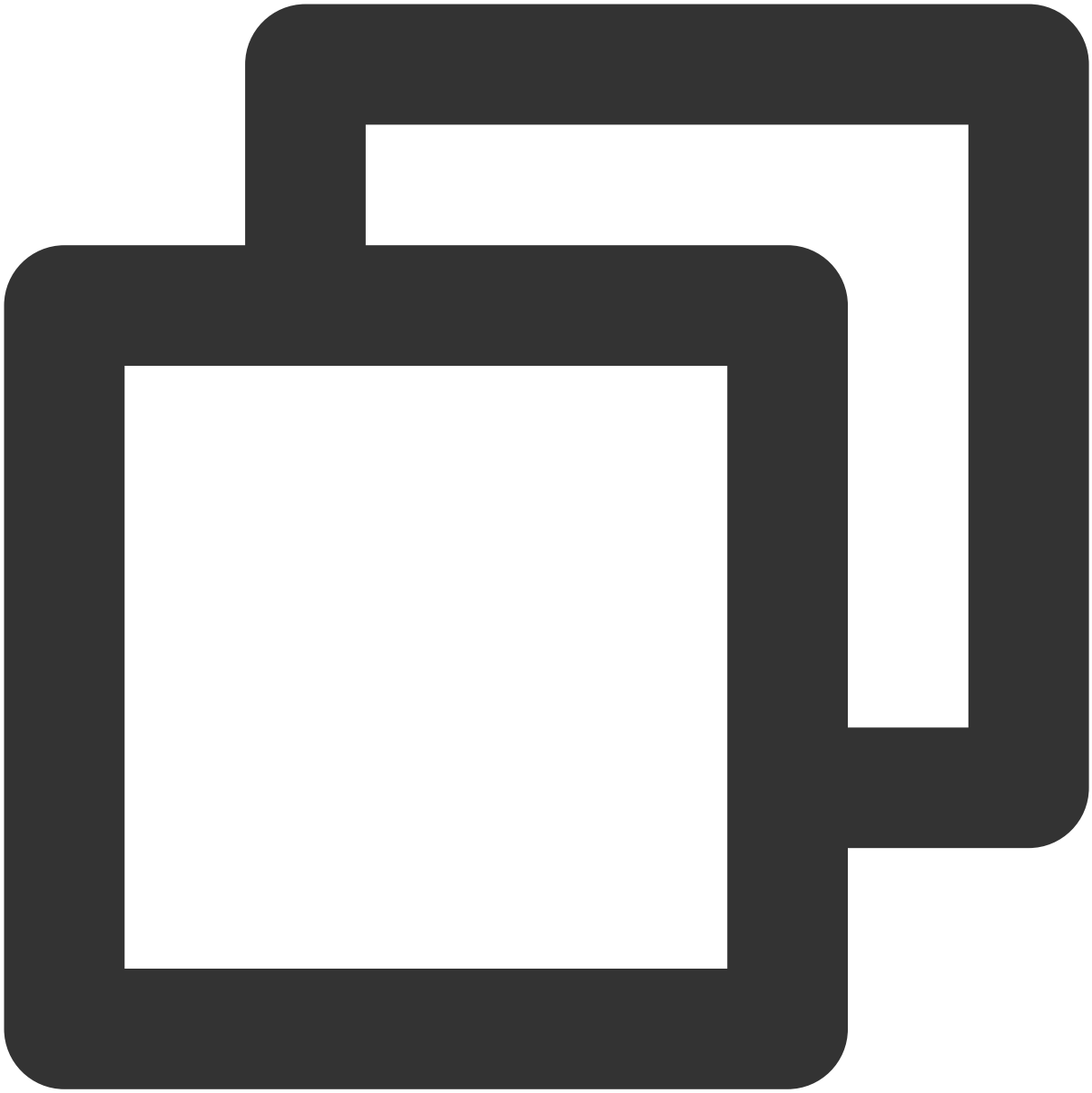
사용자 정의 캡처

로컬 스트림을 생성하기 위해 {@link TRTC.createStream createStream()}을 호출할 때 캡처 방법을 지정할 수 있습니다.
마이크와 카메라에서 오디오 및 비디오 데이터 캡처:



```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
localStream.initialize().then(() => {
  // local stream initialized success
});
```

화면 공유 스트림 캡처:



```
const localStream = TRTC.createStream({ userId, audio: false, screen: true });
localStream.initialize().then(() => {
  // local stream initialized success
});
```

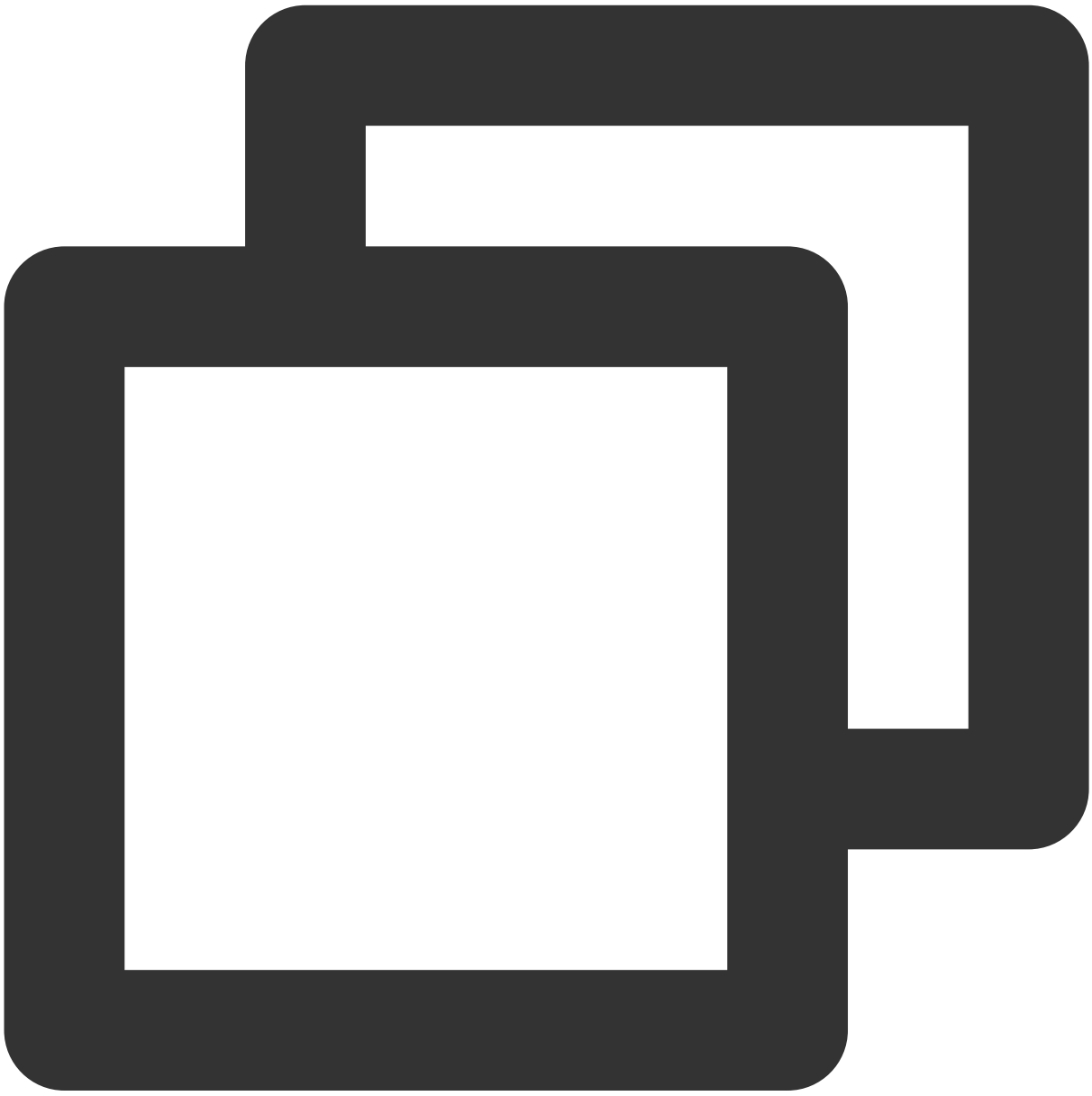
상기 두 가지 예시는 SDK에 내장된 캡처 프로세스를 사용합니다. 스트림을 사전 처리하려면 createStream을 사용하여 외부 오디오/비디오 소스, 즉 사용자 정의 캡처를 사용하여 로컬 스트림을 생성할 수 있습니다. 예를 들어 다음을 수행할 수 있습니다.

getUserMedia를 사용하여 특정 마이크 및 카메라에서 오디오/비디오를 캡처합니다.

getDisplayMedia를 사용하여 디스플레이 콘텐츠를 캡처합니다.

captureStream을 사용하여 웹페이지에서 재생되는 오디오/비디오를 캡처합니다.
captureStream을 사용하여 canvas에서 애니메이션을 캡처합니다.

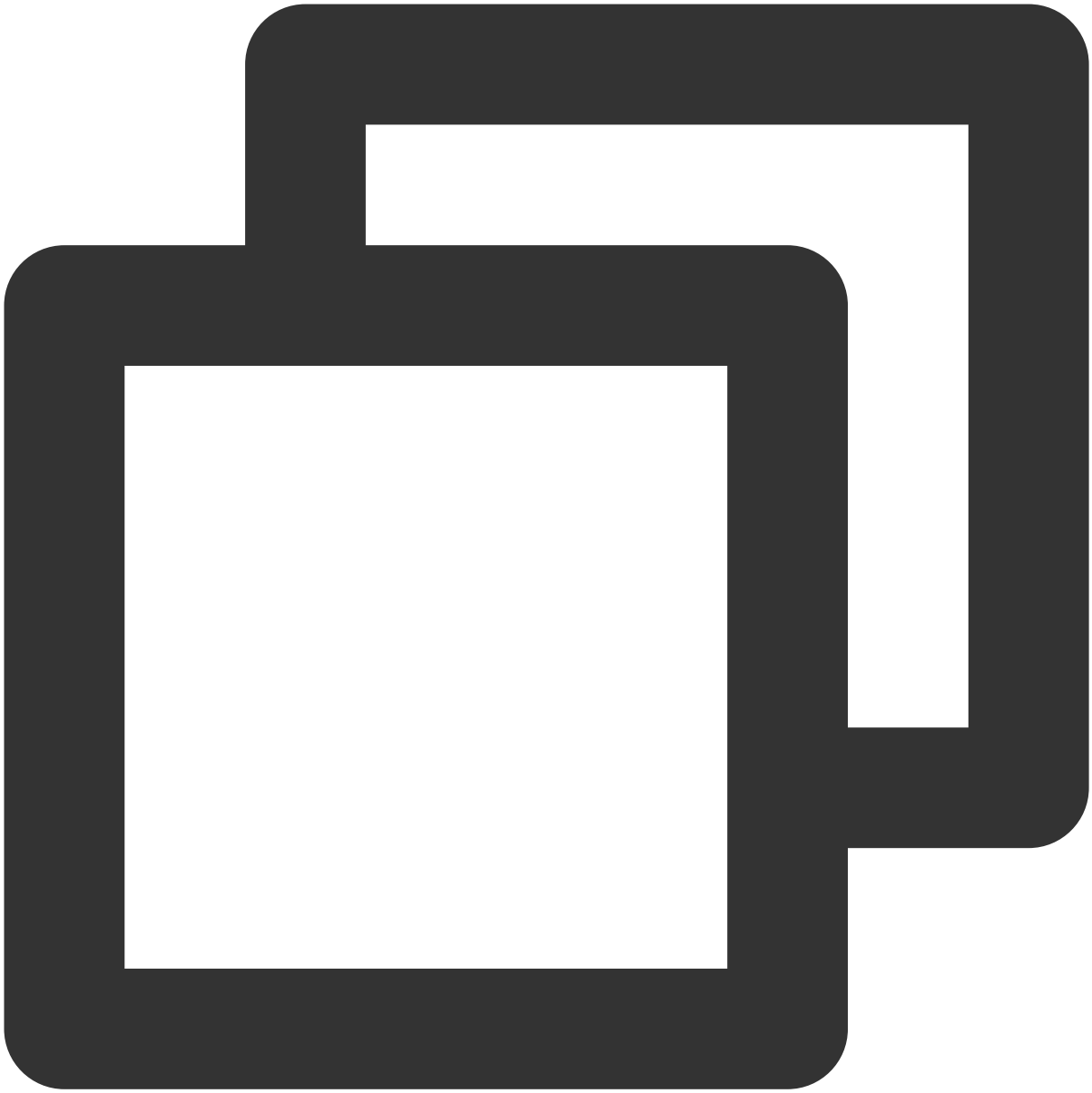
웹 페이지에서 재생되는 비디오 소스 캡처



```
// 현재 브라우저가 video 요소에서 stream 캡처를 지원하는지 확인
const isVideoCapturingSupported = () => {
    ['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) =>
        if (item in document.createElement('video')) {
            return true;
        }
```

```
    });
    return false;
};

// 현재 브라우저가 video 요소에서 stream 캡처를 지원하는지 확인
if (!isVideoCapturingSupported()) {
    console.log('your browser does not support capturing stream from video element'
    return
}
// 페이지에서 재생 중인 video의 태그 가져오기
const video = document.getElementByID('your-video-element-ID');
// 재생중인 비디오에서 비디오 스트림 캡처
const stream = video.captureStream();
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];

const localStream = TRTC.createStream({ userId, audioSource: audioTrack, videoSourc

// 영상 통화 경험을 보장하기 위해 비디오 속성은 외부 비디오 소스의 속성과 동일해야 함
localStream.setVideoProfile('480p');

localStream.initialize().then(() => {
    // local stream initialized success
});
```

canvas에서 애니메이션 캡처하기



```
// 현재 브라우저가 canvas 요소에서 stream 캡처를 지원하는지 확인
const isCanvasCapturingSupported = () => {
    ['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) =>
        if (item in document.createElement('canvas')) {
            return true;
        }
    });
    return false;
};

// 현재 브라우저가 canvas 요소에서 stream 캡처를 지원하는지 확인
```

```
if (!isCanvasCapturingSupported()) {
    console.log('your browser does not support capturing stream from canvas element
    return
}
// canvas 태그 가져오기
const canvas = document.getElementByID('your-canvas-element-ID');

// canvas에서 15 fps 비디오 스트림 캡처
const fps = 15;
const stream = canvas.captureStream(fps);
const videoTrack = stream.getVideoTracks()[0];

const localStream = TRTC.createStream({ userId, videoSource: videoTrack });

// 영상 통화 경험을 보장하기 위해 비디오 속성은 외부 비디오 소스의 속성과 동일해야 함
localStream.setVideoProfile('480p');

localStream.initialize().then(() => {
    // local stream initialized success
});
```

사용자 정의 렌더링

{@link Stream#play Stream.play()}를 사용하여 TRTC.createStream()을 통해 생성 및 초기화된 로컬 스트림 또는 Client.on('stream-added')을 통해 수신된 원격 스트림을 렌더링할 수 있습니다. Stream.play() 메소드는 오디오 플레이어와 비디오 플레이어를 만들고 <audio>/<video> 태그를 App이 전달한 Div 컨테이너에 삽입합니다.
자신의 플레이어를 사용하려면 Stream.play()/stop()을 호출하는 대신 {@link Stream#getAudioTrack Stream.getAudioTrack()}/{@link Stream#getVideoTrack Stream.getVideoTrack()} 오디오 및 비디오 트랙을 가져오고 자신의 플레이어로 렌더링합니다. 이러한 경우 Stream.on('player-state-changed') 콜백이 트리거되지 않습니다. 현재 스트림의 상태에 대한 정보를 얻으려면 App이 'mute/unmute/ended' 및 MediaStreamTrack의 기타 이벤트를 수신해야 합니다.
또한 스트림의 라이프사이클을 모니터링하기 위해 `Client.on('stream-added')` , `Client.on('stream-updated')` 및 `Client.on('stream-removed')` 을 수신해야 합니다.

주의사항 :

'stream-added' 또는 'stream-updated' 이벤트 콜백을 수신한 후 스트림에 오디오 또는 비디오 track이 있는지 확인합니다. stream-updated 콜백에 대한 오디오 또는 비디오 track이 있는 경우 플레이어를 업데이트하고 최신 오디오 및 비디오 track을 재생해야 합니다.

온라인 데모

메시지 보내기 및 받기

최종 업데이트 날짜: : 2024-07-24 16:43:38

콘텐츠 소개

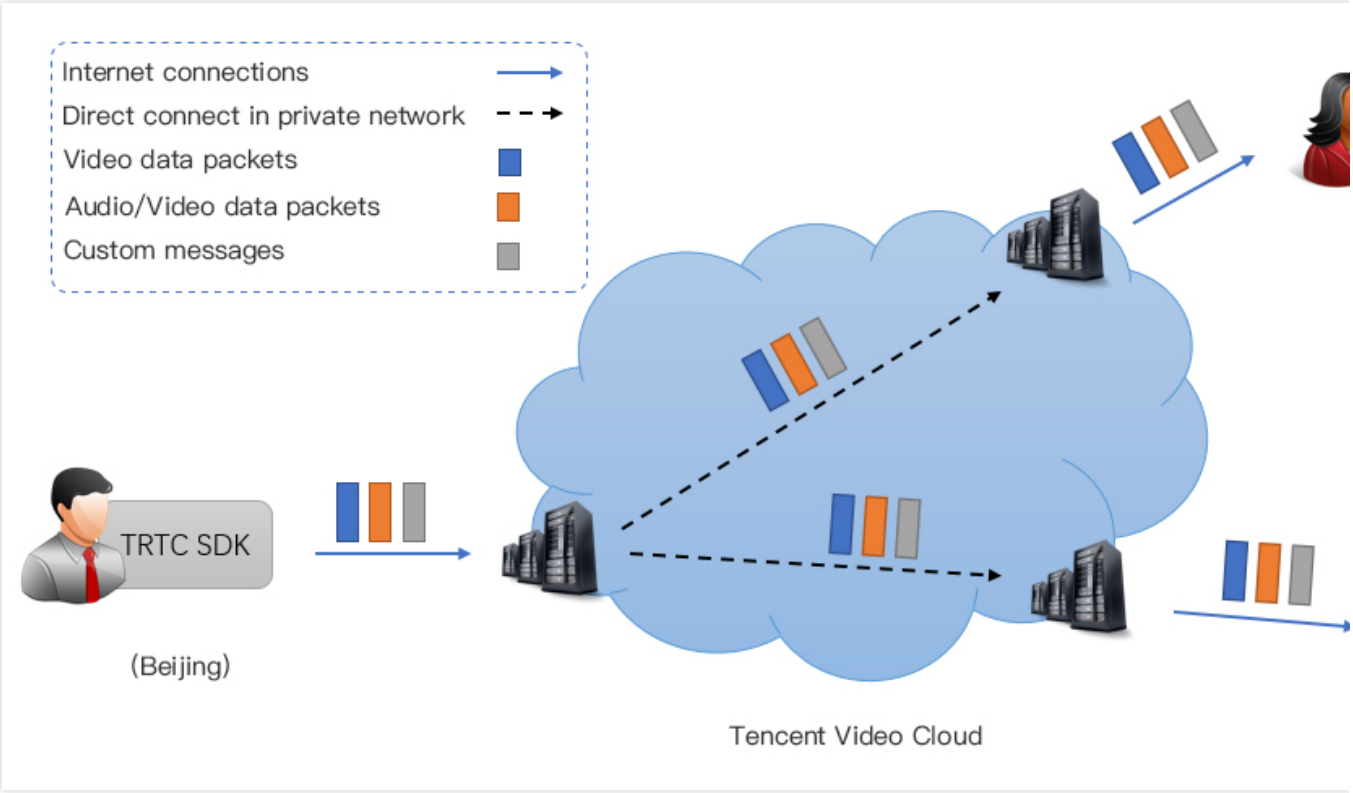
TRTC SDK는 사용자 정보 메시지를 발송하는 기능을 제공하며, 해당 기능을 통해 모든 호스트 역할을 하는 사용자가 동일한 라이브 룸의 다른 사용자에게 자신의 사용자 정의 메시지를 발송할 수 있습니다.

지원 플랫폼

iOS	Android	Mac OS	Windows	Electron	web
✓	✓	✓	✓	✓	×

발신/수신 원리

사용자의 사용자 정의 메시지가 멀티미디어 데이터 스트림에 삽입되어 멀티미디어 데이터와 함께 방 안에 있는 다른 사용자들에게 전송됩니다. 멀티미디어 채널은 본래 100% 신뢰할 수 있지 않으므로 신뢰성을 높이기 위해 TRTC SDK 내부에서 일부 신뢰도가 높은 메커니즘을 구현합니다.



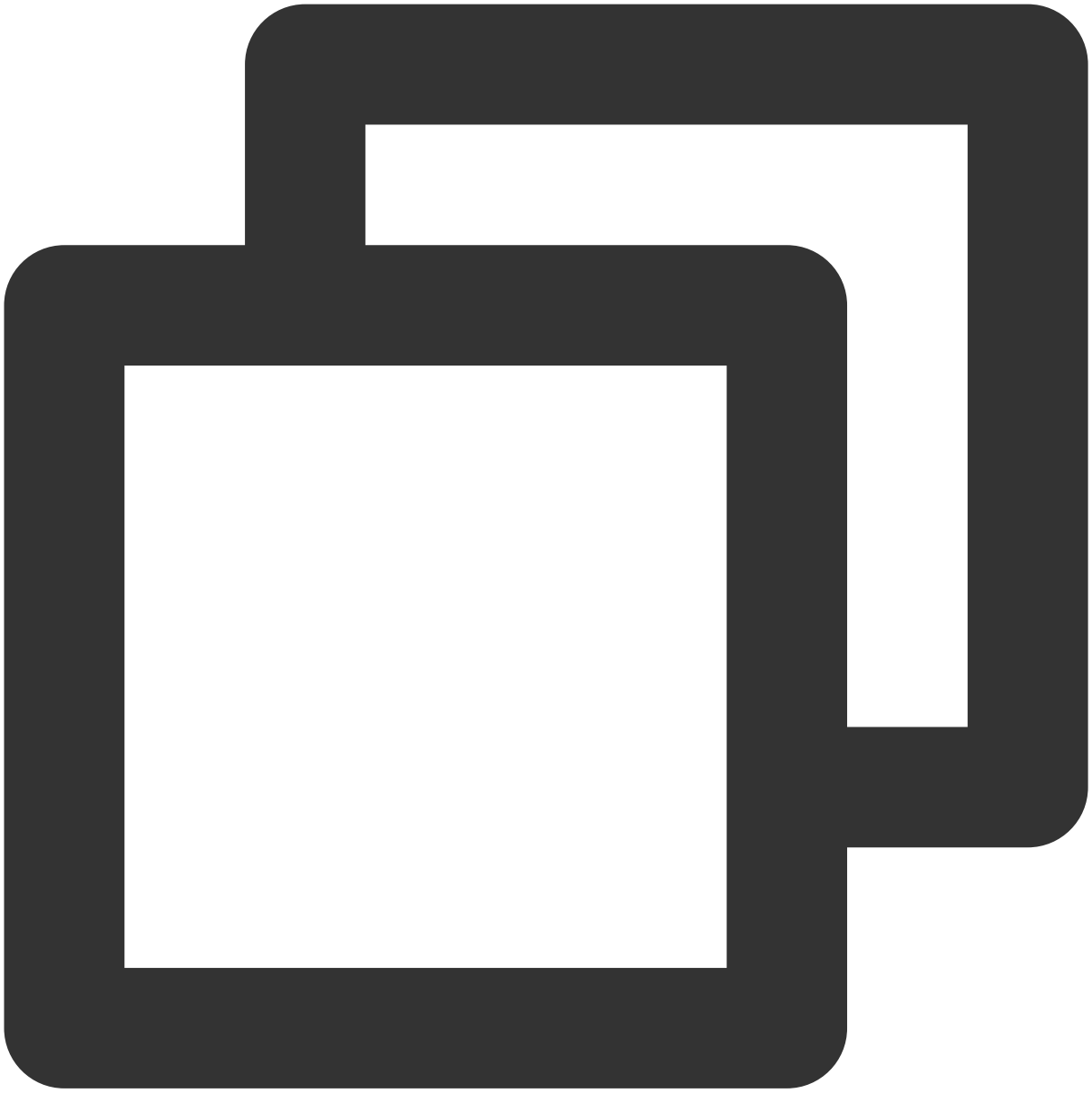
메시지 발송

TRTCCloud의 `sendCustomCmdMsg` 인터페이스 호출을 통해 발송하며, 발송 시 다음 4개의 매개변수를 지정해야 합니다.

매개변수 이름	매개변수 설명
cmdID	메시지 ID. 1 ~ 10으로 설정할 수 있으며, 각 서비스 유형 메시지별로 서로 다른 cmdID를 사용해야 합니다.
data	발송 대기 중인 메시지. 최대 1KB(1000바이트)까지 지원합니다.
reliable	신뢰도 높은 발송 여부. 신뢰도 높은 발송은 일정의 딜레이가 발생하며, 이는 수신측에서 재전송 대기를 위해 일정 시간 동안 데이터를 일시 저장하기 때문입니다.
ordered	순차 요구 여부. 즉, 수신측이 수신하는 데이터 순서와 발신측이 발송하는 순서의 일치 여부를 요구하는지 나타내며, 이는 수신측에서 해당 메시지를 일시 저장하고 정렬하기 때문에 일정한 수신 딜레이가 발생합니다.

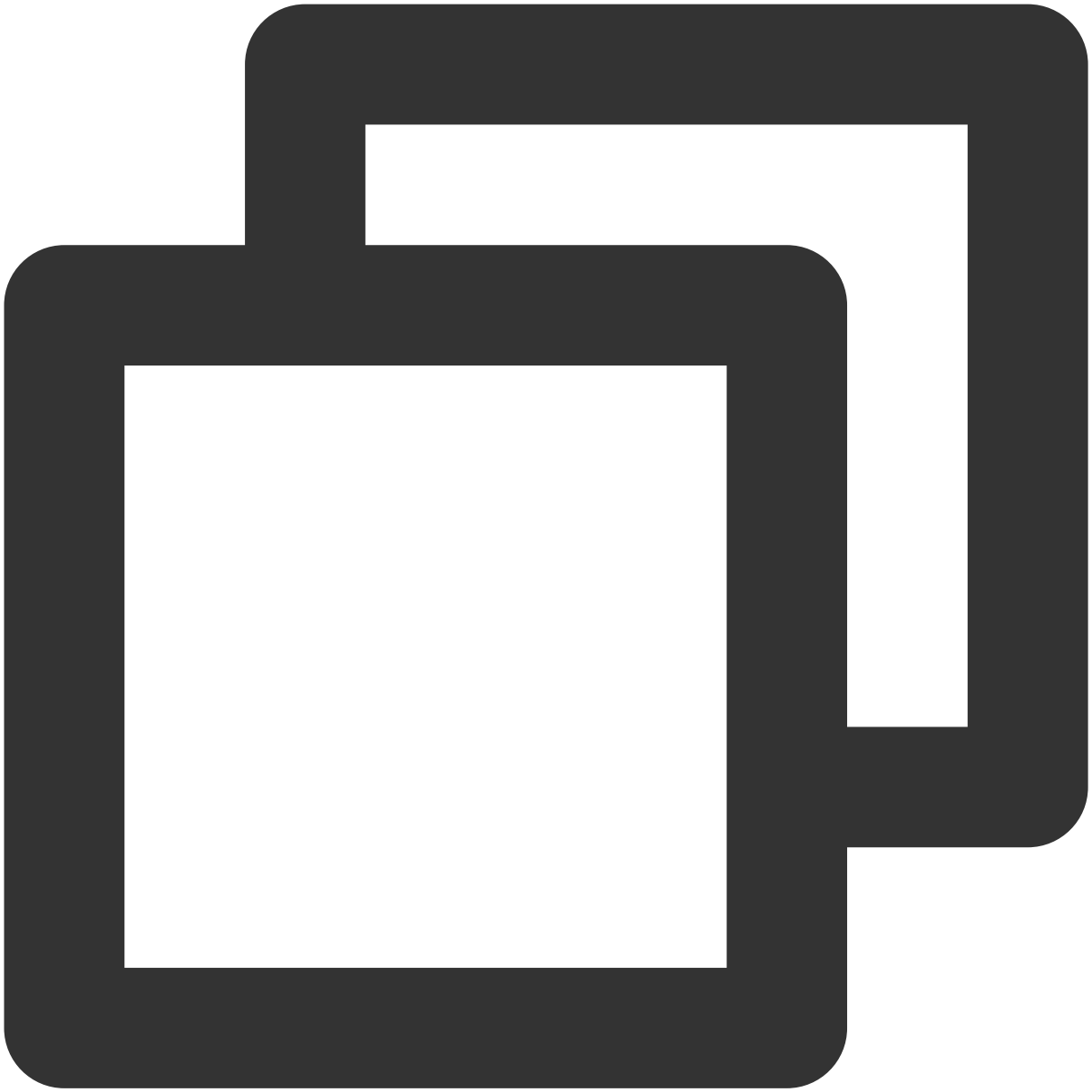
reliable과 ordered를 동시에 YES 또는 NO로 설정하십시오. 현재는 서로 다른 값으로 설정할 수 없습니다.

Objective-C



```
//사용자 정의 메시지 발송 예시 코드
- (void)sendHello {
    // 사용자 정의 메시지 명령어로, 해당 부분은 서비스에서 사용자 정의한 규칙에 따라야 하며, 0X
    NSInteger cmdID = 0x1;
    NSData *data = [@"Hello" dataUsingEncoding:NSUTF8StringEncoding];
    // reliable과 ordered는 동일한 값으로 설정해야 합니다. 본 예시는 신뢰성이 보장되는 메시지
    [trtcCloud sendCustomCmdMsg:cmdID data:data reliable:YES ordered:YES];
}
```

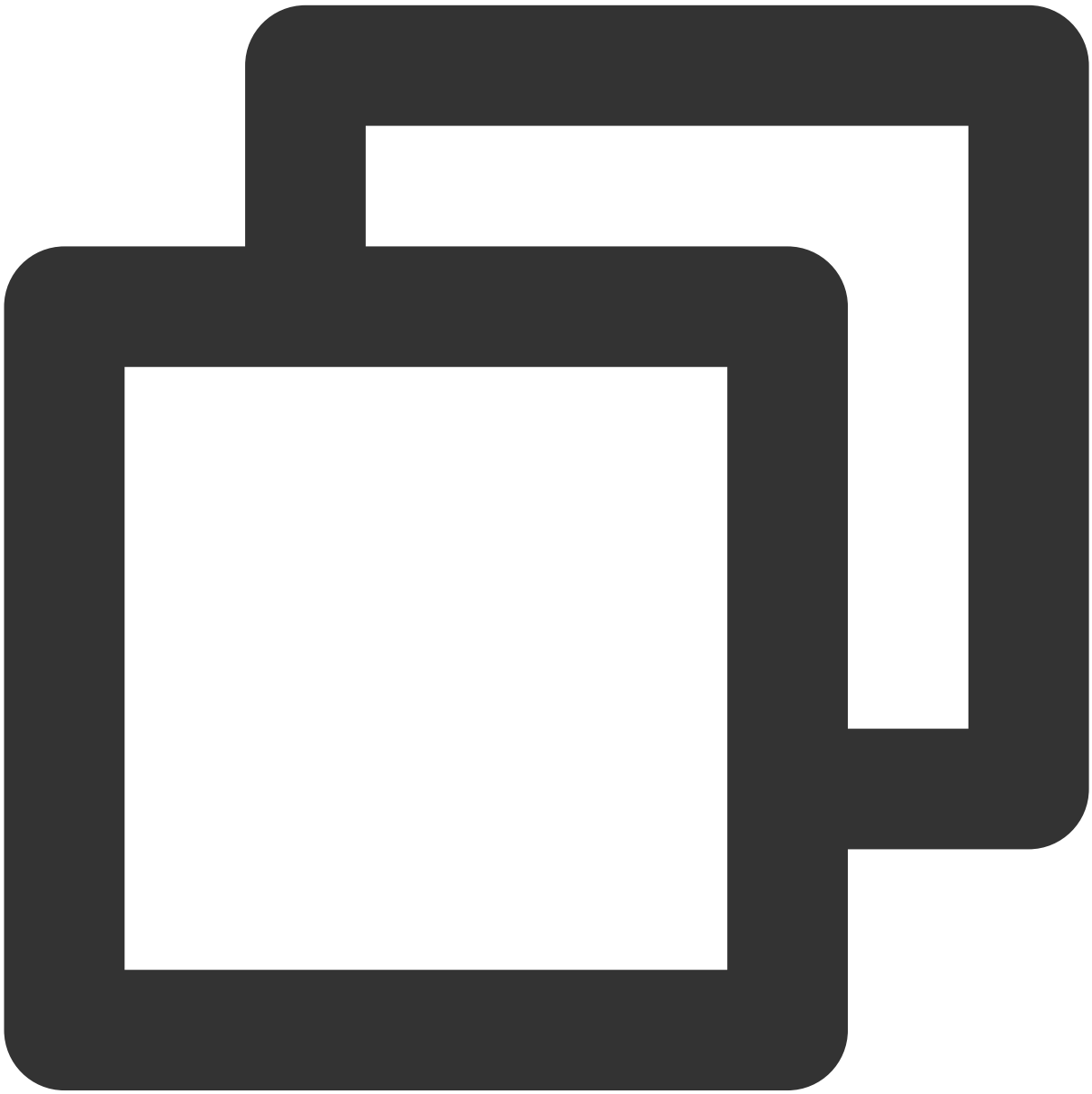
Java



```
//사용자 정의 메시지 발송 예시 코드
public void sendHello() {
    try {
        // 사용자 정의 메시지 명령어로, 해당 부분은 서비스에서 사용자 정의한 규칙에 따라야 하며
        int cmdID = 0x1;
        String hello = "Hello";
        byte[] data = hello.getBytes("UTF-8");
        // reliable과 ordered는 동일한 값으로 설정해야 합니다. 본 예시는 신뢰성이 보장되는 메
        trtcCloud.sendCustomCmdMsg(cmdID, data, true, true);
```

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

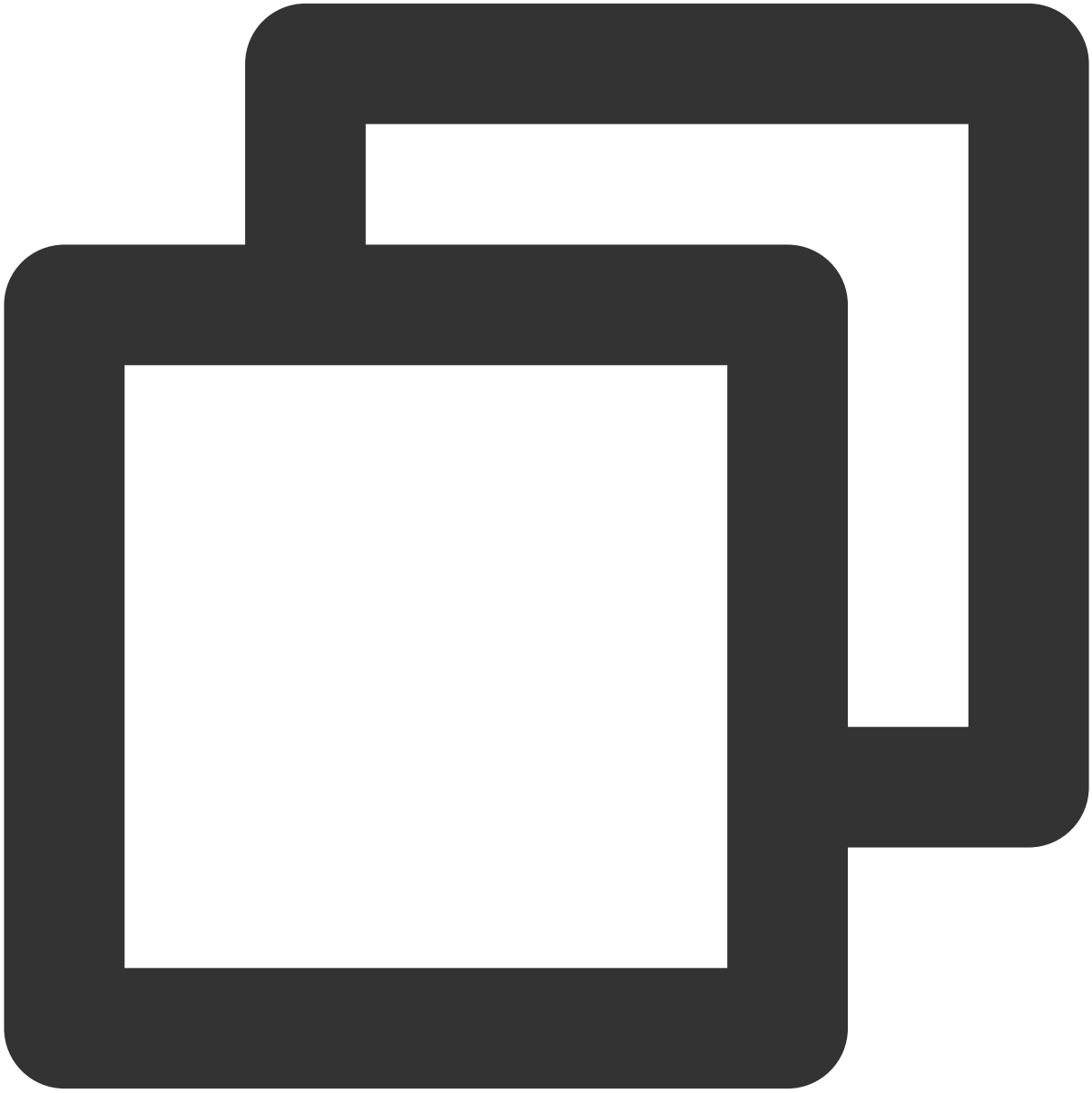
C++



```
// 사용자 정의 메시지 발송 예시 코드
void sendHello()
{
    // 사용자 정의 메시지 명령어로, 해당 부분은 서비스에서 사용자 정의한 규칙에 따라야 하며, 0X
```

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

    // reliable과 ordered는 동일한 값으로 설정해야 합니다. 본 예시는 신뢰성이 보장되는 메시지
    trtcCloud->sendCustomCmdMsg(cmdID, data, dataSize, true, true);
}
```

C#



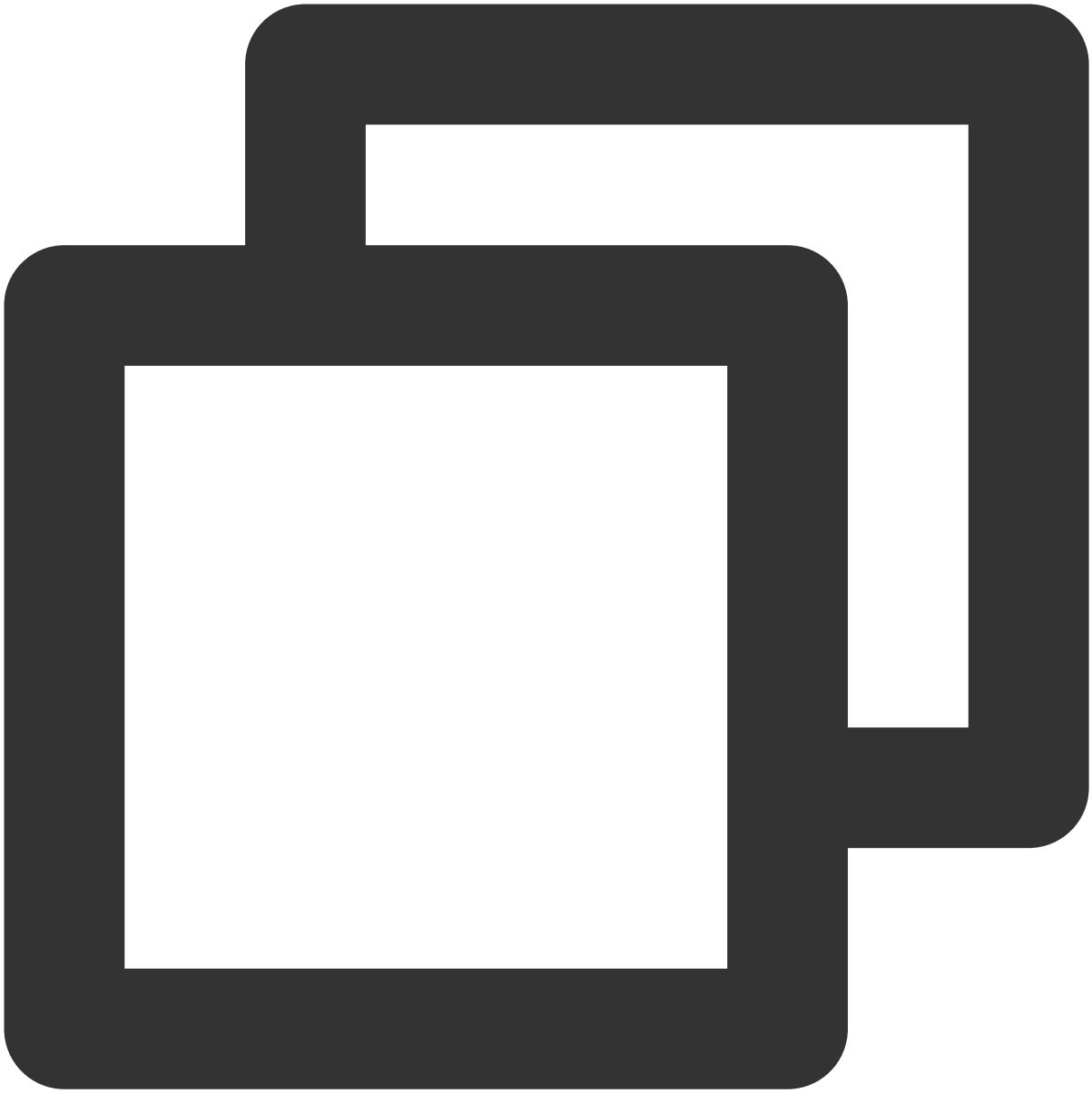
```
// 사용자 정의 메시지 발송 예시 코드
```

```
private void sendHello()
{
    // 사용자 정의 메시지 명령어로, 해당 부분은 서비스에서 사용자 정의한 규칙에 따라야 하며, 0X

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

    // reliable과 ordered는 동일한 값으로 설정해야 합니다. 본 예시는 신뢰성이 보장되는 메시지
    mTRTCCloud.sendCustomCmdMsg(cmdID, data, dataSize, true, true);
}
```

메시지 수신

방 안에 있는 한 사용자가 `sendCustomCmdMsg` 를 통해 사용자 정의 메시지를 발송하면, 방 안에 있는 다른 사용자는 SDK 콜백의 `onRecvCustomCmdMsg` 인터페이스를 통해 해당 메시지를 수신합니다.

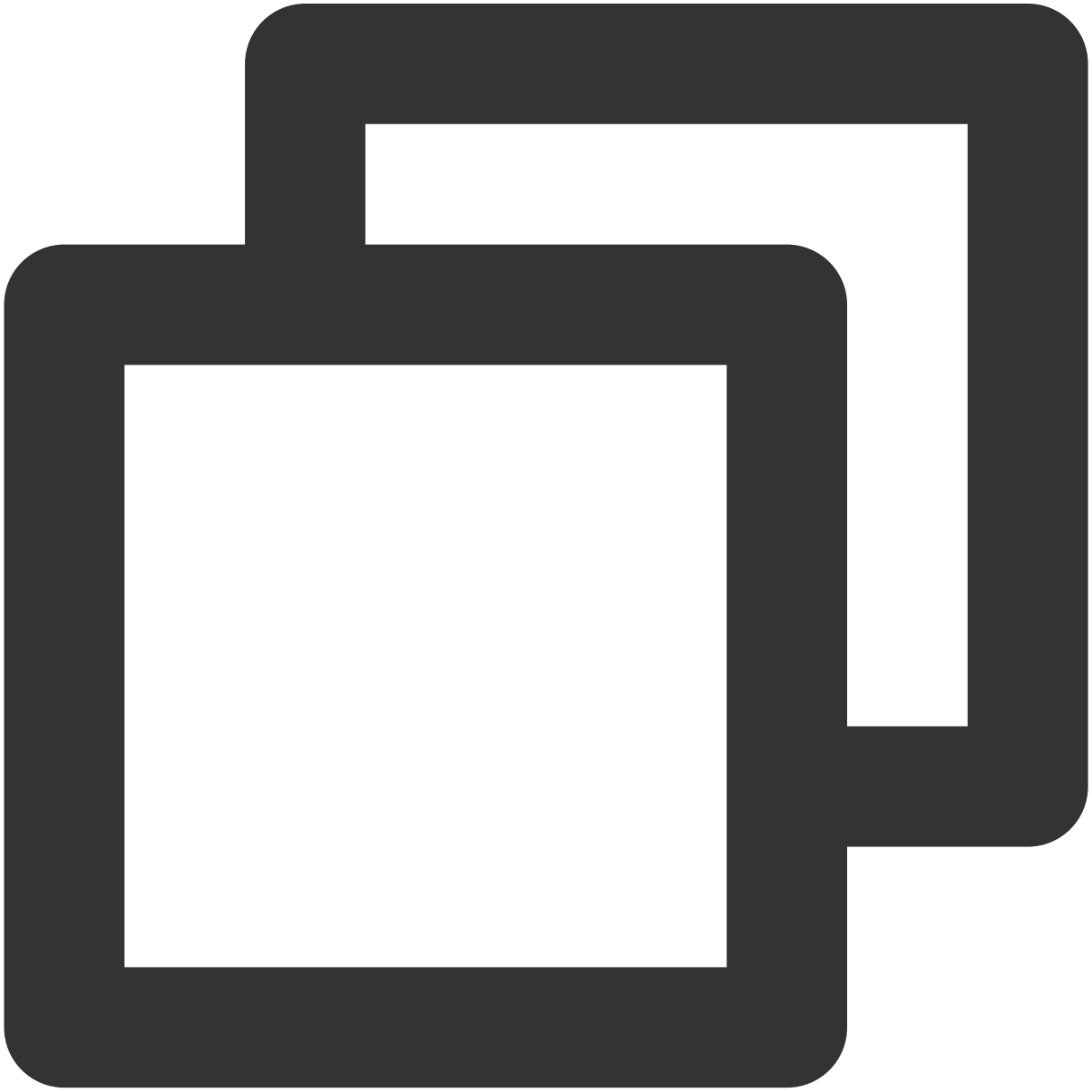
Objective-C



```
//방 안의 다른 사용자가 발송한 메시지 수신 및 처리
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:(NSInteger)cmdId seq:(UIn
{
    // userId가 발송한 메시지 수신
    switch (cmdId)  // 발신측과 수신측이 협의한 cmdId
    {
    case 0:
        // cmdId = 0 메시지 처리
        break;
    case 1:
        // cmdId = 1 메시지 처리
```

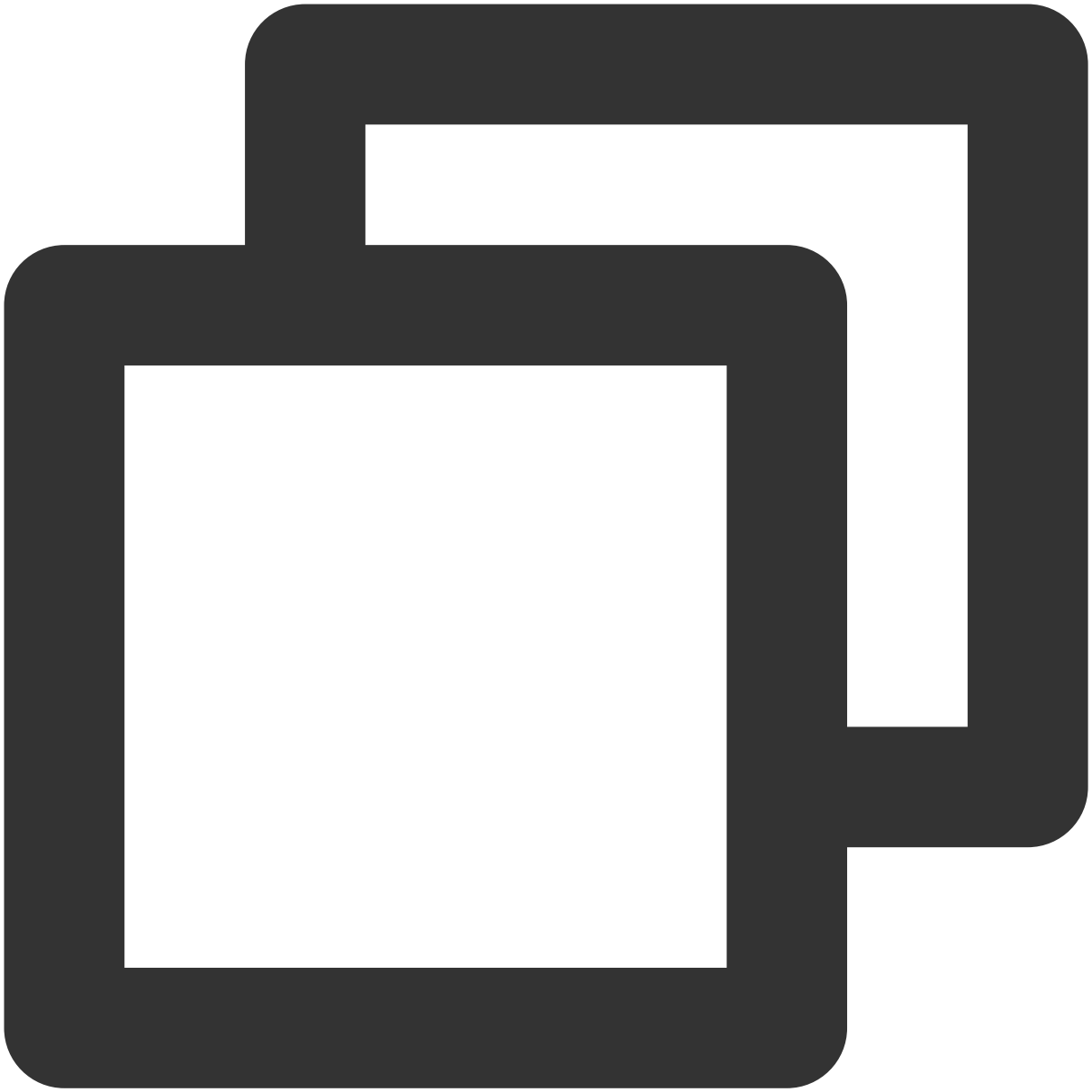
```
            break;
        case 2:
            // cmdId = 2 메시지 처리
            break;
        default:
            break;
        }
    }
```

Java



```
//TRTCCloudListener를 상속하여 onRecvCustomCmdMsg 구현 방법으로 방 안에 있는 다른 사용자의
public void onRecvCustomCmdMsg(String userId, int cmdId, int seq, byte[] message) {
    // userId가 발송한 메시지 수신
    switch (cmdId)  // 발신측과 수신측이 협의한 cmdId
    {
    case 0:
        // cmdId = 0 메시지 처리
        break;
    case 1:
        // cmdId = 1 메시지 처리
        break;
    case 2:
        // cmdId = 2 메시지 처리
        break;
    default:
        break;

}
```

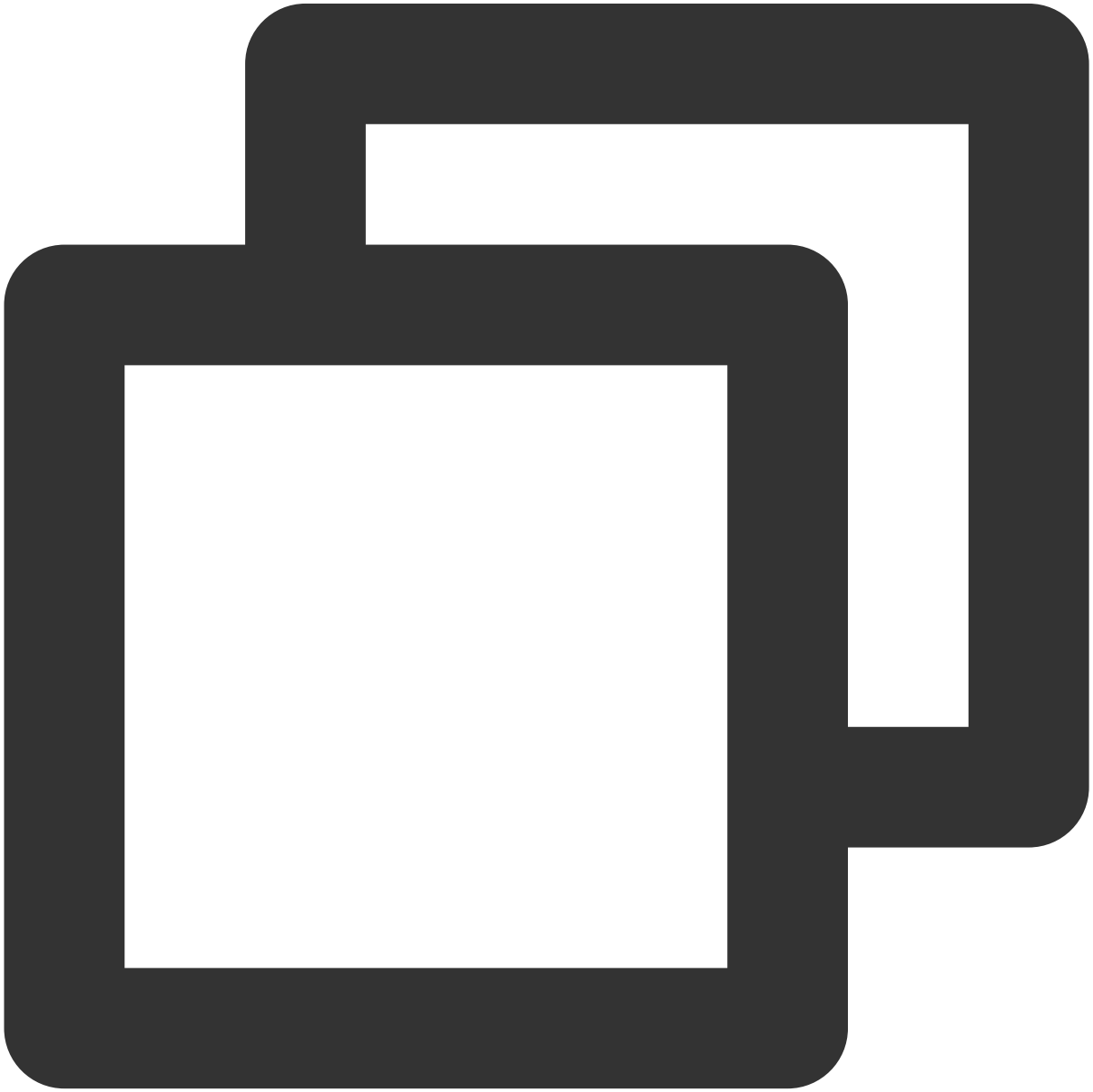
C++



```
// 방 안의 다른 사용자가 발송한 메시지 수신 및 처리
void TRTCCloudCallbackImpl::onRecvCustomCmdMsg(
                            const char* userId, int32_t cmdId, uint32_t seq, const
{
    // userId가 발송한 메시지 수신
    switch (cmdId)  // 발신측과 수신측이 협의한 cmdId
    {
    case 0:
        // cmdId = 0 메시지 처리
        break;
    case 1:
```

```
            // cmdId = 1 메시지 처리
            break;
        case 2:
            // cmdId = 2 메시지 처리
            break;
        default:
            break;
        }
    }
```

C#



```
// 방 안의 다른 사용자가 발송한 메시지 수신 및 처리
public void onRecvCustomCmdMsg(string userId, int cmdId, uint seq, byte[] msg, uint
{
    // userId가 발송한 메시지 수신
    switch (cmdId)  // 발신측과 수신측이 협의한 cmdId
    {
    case 0:
        // cmdId = 0 메시지 처리
        break;
    case 1:
        // cmdId = 1 메시지 처리
        break;
    case 2:
        // cmdId = 2 메시지 처리
        break;
    default:
        break;
    }
}
```

사용 제한

사용자 정의 메시지는 멀티미디어 데이터보다 높은 전송 우선순위를 가집니다. 따라서 사용자 정의 데이터 발송이 너무 많은 경우 멀티미디어 데이터가 간섭을 받을 수 있으며, 이로 인해 화면 랙 또는 흐릿해지는 현상이 발생할 수 있습니다. 이에 따라 사용자 정의 메시지 발송에 대해 다음과 같이 빈도수를 제한합니다.
사용자 정의 메시지는 클라우드 방송에서 방 안에 있는 사용자들에게 발송합니다. 따라서 초당 최대 30개까지 발송할 수 있습니다.
모든 메시지 패킷(즉, data 크기)은 최대 1KB로 제한되며, 이를 초과할 경우 중간 라우터 또는 서버에서 손실될 확률이 높습니다.
모든 클라이언트는 초당 최대 총 8KB의 데이터를 전송할 수 있습니다. 즉, 모든 데이터 패킷이 1KB인 경우 초당 최대 8개의 데이터 패킷만 발송할 수 있습니다.

서버 이벤트 콜백 수신
서버 이벤트 콜백 수신

최종 업데이트 날짜: : 2024-07-24 16:12:10

이벤트 콜백 서비스는 룸 이벤트 그룹(Room Event), 미디어 이벤트 그룹(Media Event) 및 녹화 이벤트 그룹의 일부 이벤트(클라우드 녹화 기능의 콜백 이벤트에 대한 설명은 클라우드 녹화 및 재생을 참고하십시오)를 포함한 HTTP/HTTPS 요청의 형태로 TRTC 이벤트를 서버에 알릴 수 있습니다. 서비스를 활성화하려면 Tencent Cloud에 구성 정보를 제공해야 합니다.

정보 설정

TRTC 콘솔은 콜백 정보 자체 설정을 지원하며, 설정이 완료되면 이벤트 콜백 공지를 수신할 수 있습니다. 자세한 작업 가이드는 콜백 설정을 참고하십시오.

주의사항 :

사용자는 다음과 같은 정보를 사전에 준비해야 합니다.

필수 항목: 콜백 공지를 수신하기 위한 HTTP/HTTPS 서버 주소입니다.

옵션 항목: 서명을 계산하는 키로, 영문 알파벳 대소문자와 숫자로 구성해 최대 32자까지 사용자 정의할 수 있습니다.

요청 시간 초과 후 재시도

이벤트 콜백 서버가 메시지 공지를 발송한 후 5초 이내에 사용자의 서버로부터 응답을 받지 못할 경우 공지 실패로 간주합니다. 첫 번째 공지 실패 후 즉시 재시도하며, 그 이후 실패하면 메시지가 1분 이상 지속될 때까지 **10초** 간격으로 계속 재시도합니다.

이벤트 콜백 메시지 포맷

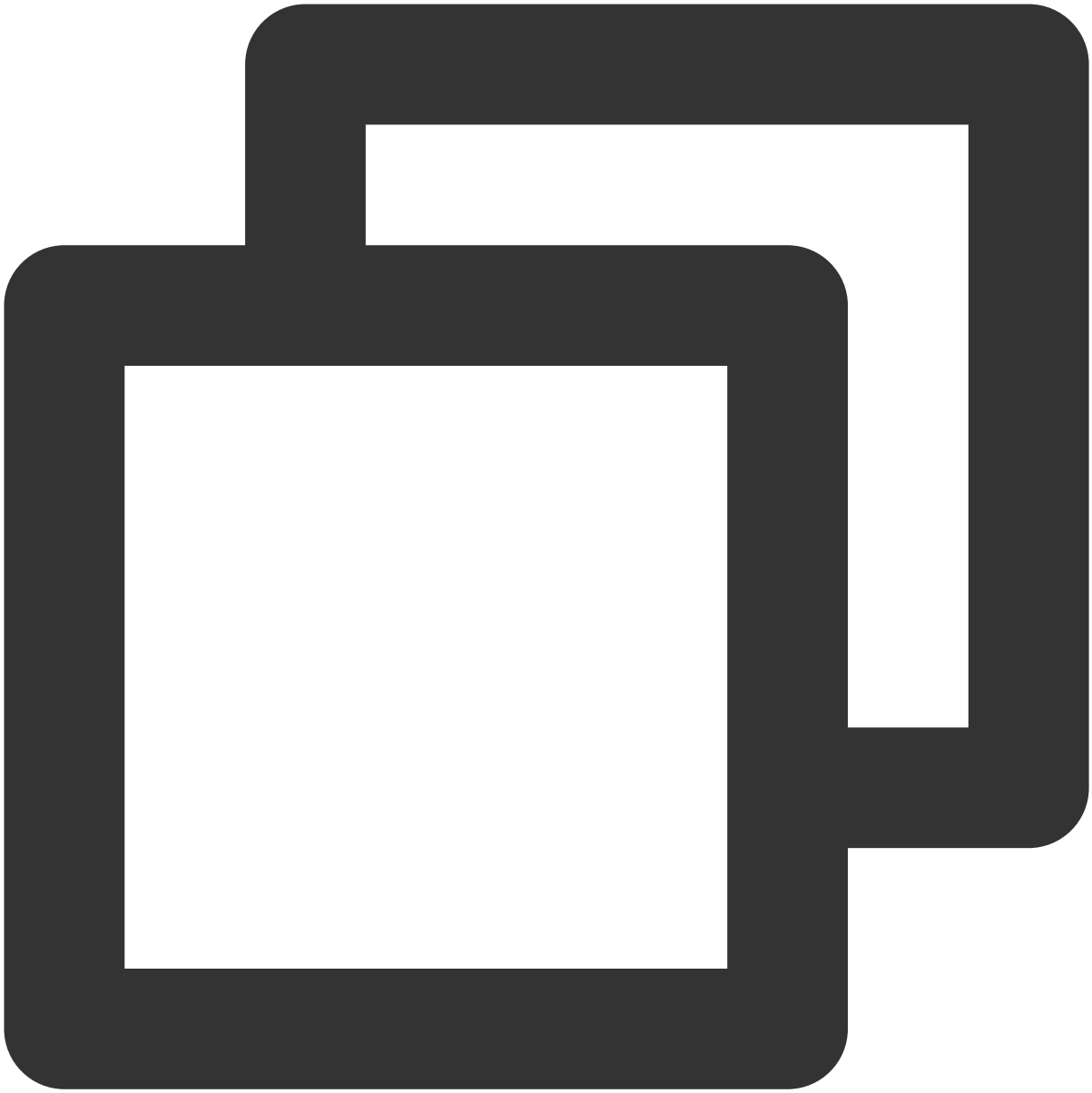
이벤트 콜백 메시지는 HTTP/HTTPS POST 요청 형식으로 사용자의 서버에 전송됩니다.

문자 인코딩 포맷: UTF-8

요청: body 포맷은 JSON

응답: HTTP STATUS CODE = 200, 서버가 응답 패키지의 세부 내용을 무시합니다. 원활한 프로토콜 연결을 위해 클라이언트 응답 콘텐츠에 JSON: {'code':0} 추가를 권장합니다.

패킷 예시: 다음은 '방 이벤트 그룹 - 방 입장' 이벤트의 패킷 예시입니다.



```
{
    'EventGroupId': 1,                  #방 이벤트 그룹
    'EventType': 103,                   #방 입장 이벤트
    'CallbackTs': 1615554923704,        #콜백 시간, 단위: 밀리초
    "EventInfo":    {
        'RoomId': 12345,                  #방 번호 숫자
        'EventTs': 1615554922,            #이벤트 발생 시간, 단위: 초
        'UserId': 'test',                 #사용자ID
        'UniqueId': 1615554922656,        #고유 식별자
        'Role': 20,                       #사용자 역할, 호스트
        'TerminalType': 3,          #단말 유형, IOS
```

```
        'UserType': 3,          #사용자 유형, Native SDK
        'Reason': 1         #입장 원인: 정상 입장
    }
}
```

매개변수 설명

콜백 메시지 매개변수

이벤트 콜백 메시지의 header는 다음과 같은 필드를 포함합니다.

필드 이름	값
Content-Type	application/json
Sign	서명값
SdkAppId	sdk application id

이벤트 콜백 메시지의 body는 다음과 같은 필드를 포함합니다.

필드 이름	유형	의미
EventGroupId	Number	이벤트 그룹 ID
EventType	Number	콜백 공지 이벤트 유형
CallbackTs	Number	이벤트 콜백 서버가 사용자의 서버로 보낸 콜백 요청의 Unix 타임스탬프, 단위: 밀리초
EventInfo	JSON Object	이벤트 정보

이벤트 그룹 ID

필드 이름	값	의미
EVENT_GROUP_ROOM	1	방 이벤트 그룹
EVENT_GROUP_MEDIA	2	미디어 이벤트 그룹

설명：

녹화 이벤트 그룹에 대한 내용은 클라우드 녹화 및 재생을 참고하십시오.

이벤트 유형

필드 이름	값	의미
EVENT_TYPE_CREATE_ROOM	101	방 생성
EVENT_TYPE_DISMISS_ROOM	102	방 삭제
EVENT_TYPE_ENTER_ROOM	103	방 입장
EVENT_TYPE_EXIT_ROOM	104	방 퇴장
EVENT_TYPE_CHANGE_ROLE	105	역할 전환
EVENT_TYPE_START_VIDEO	201	비디오 데이터 푸시 시작
EVENT_TYPE_STOP_VIDEO	202	비디오 데이터 푸시 중지
EVENT_TYPE_START_AUDIO	203	오디오 데이터 푸시 시작
EVENT_TYPE_STOP_AUDIO	204	오디오 데이터 푸시 중지
EVENT_TYPE_START_ASSIT	205	서브 채널 데이터 푸시 시작
EVENT_TYPE_STOP_ASSIT	206	서브 채널 데이터 푸시 중지

주의사항 :

방을 나가면 104 콜백만 트리거되고 202 또는 204 콜백은 트리거되지 않습니다. 202 및 204는 사용자가 비디오 및 오디오를 수동으로 끈 경우에만 트리거됩니다.

이벤트 정보

필드 이름	유형	의미
RoomId	String/Number	방 이름(유형이 클라이언트 방 번호 유형과 일치)
EventTs	Number	이벤트 발생 시간의 Unix 타임스탬프, 단위: 초 (호환 보관)
EventMsTs	Number	이벤트 발생 시간의 Unix 타임스탬프, 단위: 밀리초
UserId	String	사용자 ID
UniqueId	Number	고유 식별자(option: 방 이벤트 그룹) 클라이언트에 네트워크 전환, 진행 프로세스 이상으로 인한 퇴장 및 재입장 등의 특수 상황이 발생하면, 콜백 서버는 동일한 사용자의 방 입장/퇴장 콜백을 여러 번 수신하게 됩니다. UniqueId는 사용자의 동일한 회차의 방 입장/퇴장을 표시하는 데 사용합니다.

Role	Number	역할 유형(option: 입장 및 퇴장 시 사용)
TerminalType	Number	단말 유형(option: 방 입장 시 사용)
UserType	Number	사용자 유형(option: 방 입장 시 사용)
Reason	Number	구체적 원인 (option: 입장 및 퇴장 시 사용)

주의사항 :

'클라이언트의 특수한 행동으로 인한 반복 콜백 필터링' 정책이 배포되었습니다. 2021년 7월 30일 이후에 콜백 서비스에 연결한 경우 기본적으로 새로운 정책이 적용되며, 방 이벤트 그룹은 더 이상 UniqueId(고유 식별자)를 사용하지 않습니다.

역할 유형

필드 이름	값	의미
MEMBER_TRTC_ANCHOR	20	호스트
MEMBER_TRTC_VIEWER	21	시청자

단말 유형

필드 이름	값	의미
TERMINAL_TYPE_WINDOWS	1	Windows
TERMINAL_TYPE_ANDROID	2	Android
TERMINAL_TYPE_IOS	3	iOS
TERMINAL_TYPE_LINUX	4	Linux
TERMINAL_TYPE_OTHER	100	기타

사용자 유형

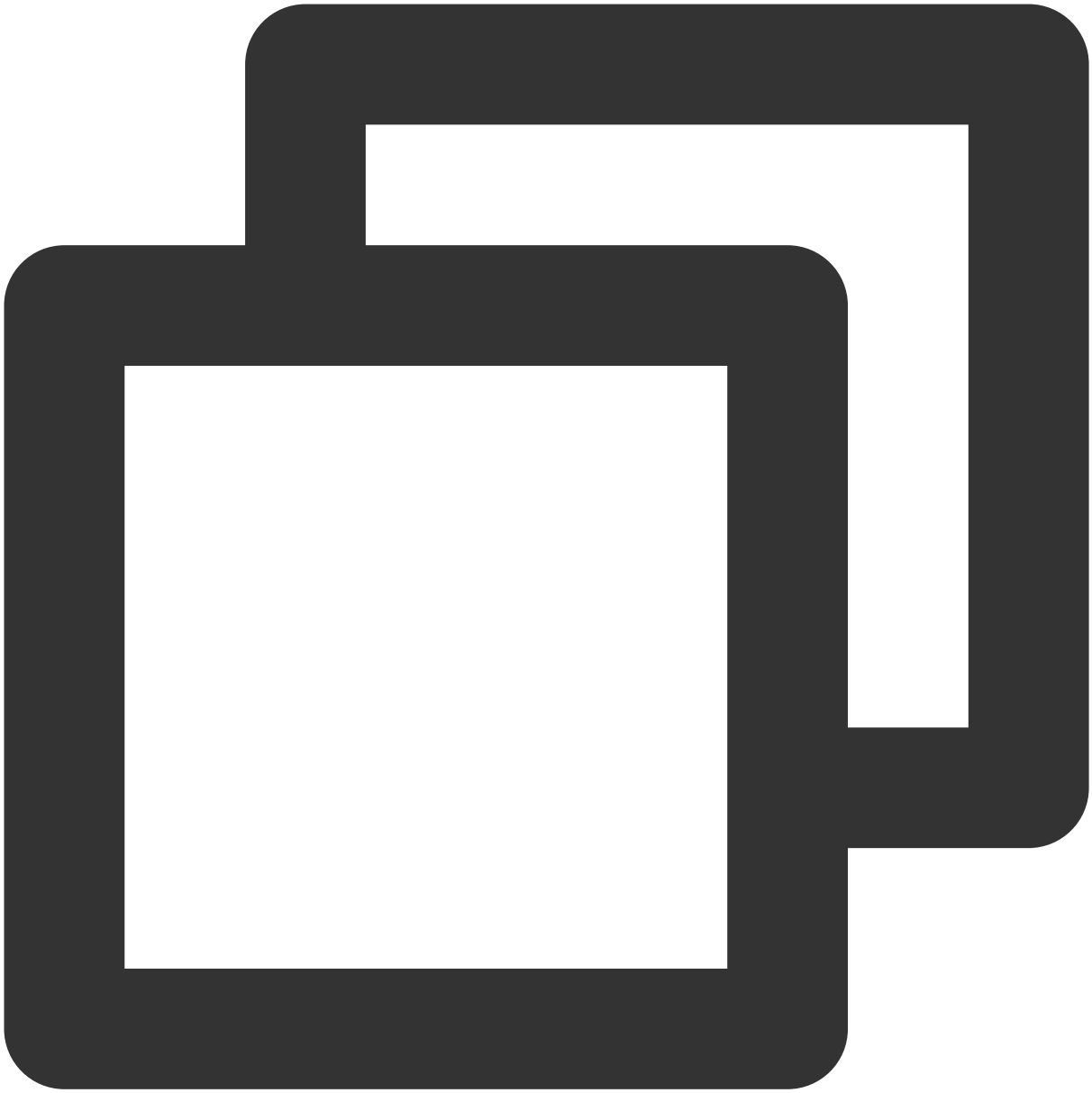
필드 이름	값	의미
USER_TYPE_WEBRTC	1	webrtc
USER_TYPE_APPLET	2	미니프로그램
USER_TYPE_NATIVE_SDK	3	Native SDK

구체적 원인

필드 이름	의미
방 입장	1: 정상 입장 2: 네트워크 전환 3: 요청 시간 초과 후 재시도 4: 크로스 룸 마이크 연결 방 입장
방 퇴장	1: 자발적 퇴장 2: 시간 초과 3: 강제 퇴장됨 4: 공동 호스트 취소 5: 프로세스 강제 종료 **참고: TRTC는 Android에서 강제 종료 이벤트를 캡처할 수 없으며 시간 초과(reason = 2) 후에만 콜백을 보냅니다.**

서명 계산

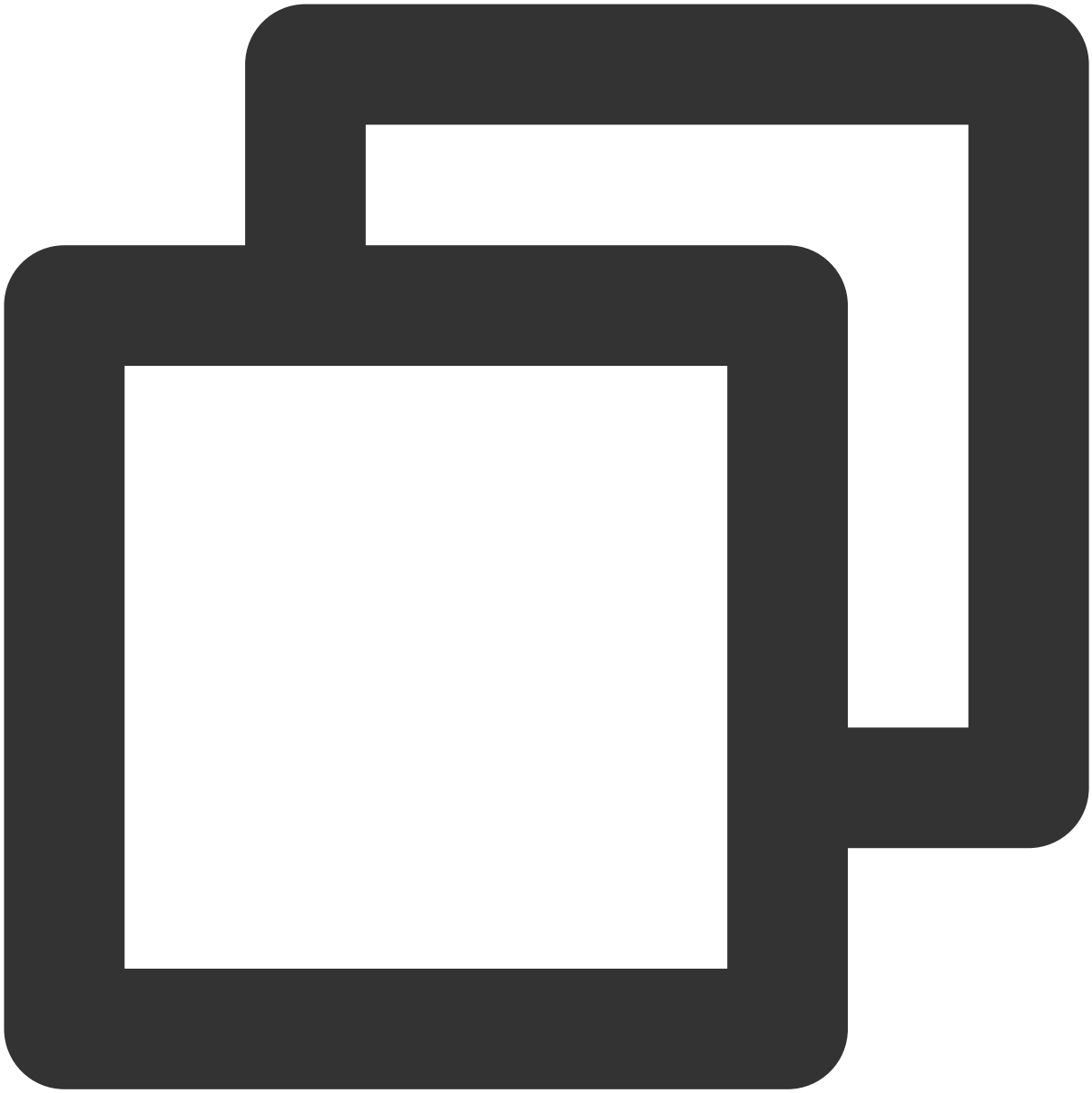
서명은 HMAC SHA256 암호화 알고리즘에 의해 계산됩니다. 사용자의 이벤트 콜백 수신 서버가 콜백 메시지를 수신하면 같은 방식으로 서명을 계산하며, 서명이 동일하면 Tencent Cloud TRTC의 이벤트 콜백이 위조되지 않았음을 의미합니다. 서명 계산 방식은 다음과 같습니다:



```
//서명(Sign) 계산 공식에서 key는 서명 계산용 암호화 키를 의미합니다.
Sign = base64(hmacsha256(key, body))
```

주의사항 :

body는 콜백 요청을 수신하는 원시 패킷이므로 변경해서는 안 됩니다. 예시는 다음과 같습니다.



```
body='{\\n\\t\\'EventGroupId\\':\\t1,\\n\\t\\'EventType\\':\\t103,\\n\\t\\'Callback
```

CAM
CAM 개요

최종 업데이트 날짜: : 2024-07-24 16:43:38

주의사항 :

본 문서는 **TRTC** CAM 기능에 대한 내용을 소개합니다. 기타 제품의 CAM 관련 내용은 CAM 지원 제품을 참조하십시오.

CAM(Cloud Access Management, **CAM**)은 Tencent Cloud에서 제공하는 웹서비스로, 사용자가 Tencent Cloud 계정의 리소스에 대한 액세스 권한을 안전하게 관리하는 데 주로 사용합니다. CAM으로 사용자(그룹)를 생성, 관리 및 폐기할 수 있으며, 신분 관리 및 정책 관리를 통해 누가 어떤 Tencent Cloud 리소스를 사용할지 제어할 수 있습니다.

TRTC가 **CAM**에 이미 연결된 경우, 개발자는 필요에 따라 서브 계정에 적합한 TRTC 액세스 권한을 할당할 수 있습니다.

시작하기

TRTC CAM 사용 시, 사전에 CAM과 TRTC의 기본 개념을 모두 이해해야 합니다. 관련 개념은 다음 문서를 참조하십시오.

CAM 관련: 사용자, 정책

TRTC 관련: 애플리케이션, SDKAppID

적용 시나리오

Tencent Cloud 제품 차원의 권한 격리

기업에서 여러 부서가 Tencent Cloud를 사용 중이며 A 부서는 TRTC 연결만 담당하고 있습니다. A 부서의 직원은 TRTC 액세스 권한은 필요하지만, 다른 Tencent Cloud 제품의 액세스 권한은 가질 수 없습니다. 이 경우, 해당 기업은 루트 계정을 통해 A 부서를 위한 서브 계정을 생성하고 해당 서브 계정에 대해서만 TRTC 관련 권한을 부여한 다음 A 부서에 제공할 수 있습니다.

TRTC 애플리케이션 차원의 권한 격리

기업에서 여러 비즈니스에 TRTC를 사용 중이라면 비즈니스 간 격리가 필요하며, 격리에는 리소스 격리와 권한 격리 두 가지 방법이 있습니다. 리소스 격리는 TRTC 애플리케이션 시스템에서 제공하며, 권한 분리는 TRTC CAM으로 구현됩니다. 해당 기업은 각 비즈니스당 하나의 서브 계정을 생성하고, 관련된 애플리케이션에만 액세스할 수 있도록 TRTC 애플리케이션 권한을 부여할 수 있습니다.

TRTC 작업 차원의 권한 격리

기업의 어느 비즈니스에서 TRTC를 사용 중이라면, 해당 비즈니스의 제품 운영자는 TRTC 콘솔에 액세스하여 사용량 통계 정보를 가져와야 함과 동시에 오작업으로 비즈니스에 영향을 미칠 가능성이 있는 중요 작업(예: 릴레이 푸시 스트림 수정, 클라우드 녹화 설정 수정 등)은 할 수 없습니다. 이 경우, 먼저 TRTC 콘솔 로그인 및 사용량 통계와 관련된 API 액세스 권한을 가진 사용자 정의 정책을 생성합니다. 그 다음 서브 계정을 생성하여 위의 정책을 바인딩하고 해당 서브 계정을 제품 운영자에게 제공할 수 있습니다.

권한 부여 단위

CAM의 핵심 기능은 **특정 리소스에 대한 특정 계정의 특정 작업 수행 허용 또는 금지**입니다. TRTC CAM은 리소스 레벨 권한 부여를 지원하며, 리소스는 TRTC 애플리케이션 단위로, 작업은 서버 API 및 TRTC 콘솔 액세스 시 사용하는 API를 포함한 클라우드 API 단위로 분할됩니다. 자세한 설명은 권한을 부여할 수 있는 리소스 및 작업을 참조하십시오.

기능 제한

TRTC CAM의 리소스는 애플리케이션 단위로 분할되며, 더 작은 단위의 리소스(예: 애플리케이션 정보, 설정 정보 등)에는 권한 부여를 지원하지 않습니다.
TRTC CAM은 프로젝트를 지원하지 않습니다. 태그를 통한 클라우드 서비스 리소스 관리를 권장합니다.

권한을 부여할 수 있는 리소스 및 작업

최종 업데이트 날짜: : 2024-07-24 16:43:38

주의사항 :

본 문서는 **TRTC** CAM 기능에 대한 내용을 소개합니다. 기타 제품의 CAM 관련 내용은 CAM 지원 제품을 참고하십시오.

CAM의 핵심 기능은 **계정이 특정 리소스에 대해 특정 작업을 수행하는 것을 허용하거나 금지하는 것**입니다. TRTC CAM은 리소스 권한 부여를 지원합니다. 리소스의 데이터 분할 정도는 TRTC 애플리케이션이며, 작업 데이터 분할 정도는 Cloud API로 서버 API와 TRTC 콘솔 액세스 시 사용할 수 있는 API를 포함합니다.

TRTC CAM이 필요한 경우 Tencent Cloud의 루트 계정에 로그인하여 사전 설정 정책 또는 사용자 정의 정책을 사용해 세부 권한 부여 작업을 완료하십시오.

권한을 부여할 수 있는 리소스 유형

CAM 권한을 부여할 수 있는 리소스 유형: 애플리케이션.

리소스 수준 권한 부여 지원 API

일부 리소스 권한 부여를 지원하지 않는 API를 제외하고, 본 섹션에 나열된 모든 API 작업은 리소스 권한 부여를 지원합니다. 이러한 API 작업에 대한 권한 부여 정책 구문의 **리소스 구문 설명**은 동일하며 구체적인 내용은 다음과 같습니다.

전체 애플리케이션에 액세스 권한 부여: `qcs::trtc::uin/${uin}:sdkappid/*` .

개별 애플리케이션에 액세스 권한 부여: `qcs::trtc::uin/${uin}:sdkappid/${SdkAppId}` .

서버 API 작업

API 이름	API 분류	기능 설명
DismissRoom	방 관리	방 해산
RemoveUser	방 관리	사용자 삭제
RemoveUserByStrRoomId	방 관리	사용자 삭제(문자열 방 번호)
DismissRoomByStrRoomId	방 관리	방 해산(문자열 방 번호)
StartMCUMixTranscode	혼합 스트림 트랜스 코딩	클라우드 혼합 스트림 실행

StopMCUMixTranscode	혼합 스트림 트랜스 코딩	클라우드 혼합 스트림 종료
StartMCUMixTranscodeByStrRoomId	혼합 스트림 트랜스 코딩	클라우드 혼합 스트림 실행(문자열 방 번호)
StopMCUMixTranscodeByStrRoomId	혼합 스트림 트랜스 코딩	클라우드 혼합 스트림 종료(문자열 방 번호)
CreateTroubleInfo	통화 품질 모니터링	오류 정보 생성
DescribeAbnormalEvent	통화 품질 모니터링	오류 경험 이벤트 조회
DescribeCallDetail	통화 품질 모니터링	사용자 목록 및 통화 지표 조회
DescribeHistoryScale	통화 품질 모니터링	방 및 사용자 수 내역 조회
DescribeRoomInformation	통화 품질 모니터링	방 리스트 조회
DescribeUserInformation	통화 품질 모니터링	기존 사용자 리스트 쿼리

콘솔 API 작업

API 이름	사용 모듈	기능 설명
DescribeAppStatList	TRTC 콘솔 개요 사용량 통계 모니터링 대시보드 개발 보조 > UserSig 생성&검증 애플리케이션 관리	애플리케이션 리스트 가져오기
DescribeSdkAppInfo	TRTC 콘솔 애플리케이션 관리 > 애플리케이션 정보	애플리케이션 정보 가져오기
ModifyAppInfo	TRTC 콘솔 애플리케이션 관리 > 애플리케이션 정보	애플리케이션 정보 편집

ChangeSecretKeyFlag	TRTC 콘솔 애플리케이션 관리 > 애플리케이션 정보	권한 보안키 상태 수정
CreateWatermark	TRTC 콘솔 애플리케이션 관리 > 소재 관리	이미지 업로드
DeleteWatermark	TRTC 콘솔 애플리케이션 관리 > 소재 관리	이미지 삭제
ModifyWatermark	TRTC 콘솔 애플리케이션 관리 > 소재 관리	이미지 편집
DescribeWatermark	TRTC 콘솔 애플리케이션 관리 > 소재 관리	이미지 검색
CreateSecret	TRTC 콘솔 애플리케이션 관리 > 퀵 스타트	대칭 암호화 키 생성
ToggleSecretVersion	TRTC 콘솔애플리케이션 관리 > 퀵 스타트	보안키 버전 전환（퍼블릭키 프라이빗키/대칭 암호화 키）
DescribeSecret	TRTC 콘솔 개발 보조 > 빠른 Demo 활성화 개발 보조 > UserSig 생성&검증 애플리케이션 관리 > 퀵 스타트	대칭 암호화 키 가져오기
DescribeTrtcAppAndAccountInfo	TRTC 콘솔 개발 보조 > UserSig 생성&검증	애플리케이션 및 계정 정보를 불러와 퍼블릭 키와 프라이빗 키 가져오기
CreateSecretUserSig	TRTC 콘솔 개발 보조 > UserSig 생성&검증	대칭 암호화 키로 UserSig 생성

DescribeSig	TRTC 콘솔 개발 보조 > UserSig 생성&검증 애플리케이션 관리 > 퀵 스타트	이전 버전의 퍼블릭키와 프라이빗키로 생성한 UserSig 가져오기
VerifySecretUserSig	TRTC 콘솔 개발 보조 > UserSig 생성&검증	대칭 암호화 키로 생성한 UserSig 검증
VerifySig	TRTC 콘솔 개발 보조 > UserSig 생성&검증	퍼블릭 키와 프라이빗 키로 생성된 UserSig 검증
CreateSpearConf	TRTC 콘솔 애플리케이션 관리 > 화면 설정	화면 설정 사양 추가. 이 기능 설정 카드는 iLiveSDK 1.9.6 이하 버전에만 해당됩니다. TRTC SDK 6.0 이상 버전은 화질 설정 참고
DeleteSpearConf	TRTC 콘솔 애플리케이션 관리 > 화면 설정	화면 설정 사양 삭제. 이 기능 설정 카드는 iLiveSDK 1.9.6 이하 버전에만 해당됩니다. TRTC SDK 6.0 이상 버전은 화질 설정 참고
ModifySpearConf	TRTC 콘솔 애플리케이션 관리 > 화면 설정	화면 설정 사양 수정. 이 기능 설정 카드는 iLiveSDK 1.9.6 이하 버전에만 해당됩니다. TRTC SDK 6.0 이상 버전은 화질 설정 참고
DescribeSpearConf	TRTC 콘솔 애플리케이션 관리 > 화면 설정	화면 설정 사양 가져오기. 이 기능 설정 카드는 iLiveSDK 1.9.6 이하 버전에만 해당됩니다. TRTC SDK 6.0 이상 버전은 화질 설정 참고
ToggleSpearScheme	TRTC 콘솔 애플리케이션 관리 > 화면 설정	화면 설정 시나리오 전환. 이 기능 설정 카드는 iLiveSDK 1.9.6 이하 버전에만 해당됩니다. TRTC SDK 6.0 이상 버전은 화질 설정 참고

리소스 수준 권한 부여 미지원 API

특수한 제한으로 인하여 다음 API는 리소스 권한 부여를 지원하지 않습니다.

서버 API 작업

API 이름	API 분류	기능 설명	특수 제한 설명

DescribeDetailEvent	통화 품질 모니터링	상세 이벤트 가져오기	매개변수를 입력했으나 SDKAppID가 존재하지 않아, 리소스 권한 부여가 불가능합니다.
DescribeRecordStatistic	기타 API	클라우드 녹화 과금 시간 쿼리	리소스 수준 권한 부여 미지원
DescribeTrtcInteractiveTime	기타 API	멀티미디어 인터랙션 과금 시간 쿼리	리소스 수준 권한 부여 미지원
DescribeTrtcMcuTranscodeTime	기타 API	릴레이 트랜스코딩 과금 시간 쿼리	리소스 수준 권한 부여 미지원

콘솔 API 작업

API 이름	사용 모듈	기능 설명	특수 제한 설명
DescribeTrtcStatistic	TRTC 콘솔 개요 사용량 통계	과금 기간 사용량 통계 데이터 가져오기	이 API에는 전량의 SDKAppID를 반환하는 통계 데이터가 포함되어 있어 비전량 SDKAppID를 제한하면 오류가 반환됩니다. DescribeAppStatList API를 통해 조회 가능한 애플리케이션 리스트 제한 가능
DescribeDurationPackages	TRTC 콘솔 개요 패키지 관리	선불 패키지 리스트 가져오기	선불 패키지는 하나의 Tencent Cloud 계정으로 모든 TRTC 애플리케이션에서 공유되며, 패키지 정보에 SDKAppID 매개변수가 없어 리소스 수준의 권한 부여 불가능
GetUserList	TRTC 콘솔 모니터링 대시보드	사용자 목록 가져오기	매개변수를 입력했으나 SDKAppID가 존재하지 않아, 리소스 권한 부여가 불가능합니다. 필요한 경우 DescribeAppStatList API를 통해 조회 가능한 애플리케이션 리스트 제한 가능
GetUserInfo	TRTC 콘솔 모니터링 대시보드	사용자 정보 가져오기	매개변수를 입력했으나 SDKAppID가 존재하지 않아, 리소스 권한 부여가 불가능합니다. 필요한 경우 DescribeAppStatList API를 통해 조회 가능한 애플리케이션 리스트 제한 가능

GetCommState	TRTC 콘솔 모니터링 대시보드	통화 상태 가져오기	매개변수를 입력했으나 SDKAppID가 존재하지 않아, 리소스 권한 부여가 불가능합니다. 필요한 경우 DescribeAppStatList API를 통해 조회 가능한 애플리케이션 리스트 제한 가능
GetElasticSearchData	TRTC 콘솔 모니터링 대시보드	ES 데이터 조회	매개변수를 입력했으나 SDKAppID가 존재하지 않아, 리소스 권한 부여가 불가능합니다. 필요한 경우 DescribeAppStatList API를 통해 조회 가능한 애플리케이션 리스트 제한 가능
CreateTrtcApp	TRTC 콘솔 개발 보조 > 빠른 Demo 활성화 애플리케이션 관리	TRTC 애플리케이션 생성	매개변수를 입력했으나 SDKAppID가 존재하지 않아, 리소스 권한 부여가 불가능합니다. SDKAppID는 TRTC 애플리케이션의 유일한 식별자로, 애플리케이션이 생성되어야 SDKAppID 발급
HardDescribeMixConf	TRTC 콘솔 애플리케이션 관리 > 기능 설정	자동 릴레이 푸시 스트림 상태 조회	매개변수를 입력했으나 SDKAppID가 존재하지 않아, 리소스 권한 부여가 불가능합니다. 필요한 경우 DescribeAppStatList API를 통해 조회 가능한 애플리케이션 리스트 제한 가능
ModifyMixConf	TRTC 콘솔 애플리케이션 관리 > 기능 설정	자동 릴레이 푸시 스트림 시작/종료	매개변수를 입력했으나 SDKAppID가 존재하지 않아, 리소스 권한 부여가 불가능합니다. 필요한 경우 DescribeAppStatList API를 통해 조회 가능한 애플리케이션 리스트 제한 가능
RemindBalance	TRTC 콘솔패키지 관리	선불 패키지 잔액 알람 정보 가져오기	선불 패키지는 하나의 Tencent Cloud 계정으로 모든 TRTC 애플리케이션에서 공유되며, 패키지 정보에 SDKAppID 매개변수가 없어 리소스 수준의 권한 부여 불가능

주의사항 :

리소스 권한 부여를 지원하지 않는 API 작업에 대해서도 사용자 정의 정책을 통해 사용자에게 해당 작업의 사용 권한을 부여할 수 있으나, 반드시 정책 문구의 리소스 요소에 `*` 를 지정해야 합니다.