

# **Tencent Real-Time Communication Integration (UI Included) Product Documentation**



## Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

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

## Trademark Notice



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

## Service Statement

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



# Contents

## Integration (UI Included)

### TUIKit Overview

### Audio/Video Call

#### Overview (TUICallKit)

#### Activate the Service (TUICallKit)

#### Integration (TUICallKit)

##### Android

##### iOS

##### Web&H5 (React)

##### Web&H5 (Vue3)

##### Flutter

##### uniapp (Android&iOS)

#### UI Customization (TUICallKit)

##### Android

##### iOS

##### Web

##### Flutter

#### Offline Call Push (TUICallKit)

##### iOS

##### VoIP

##### APNs

##### Android

##### Flutter

#### On-Cloud Recording (TUICallKit)

#### Additional Features (TUICallKit)

##### Configuring Nicknames and Avatars (All Platform)

##### Configure Resolution and Fill Mode (Web)

##### Group Calls

##### Android&iOS&Flutter

##### Web&H5

##### uni-app (Android&iOS)

##### Floating Window

##### Android&iOS&Flutter

##### Web&H5

##### uni-app (Android&iOS)

Beauty Effects

Flutter

Custom Ringtone

Android

iOS

Web&H5

uni-app (Android&iOS)

Flutter

Monitoring Call Status

Android&iOS&Flutter

Web&H5

uni-app (Android & iOS)

Client APIs (TUICallKit)

Android

API Overview

TUICallKit

TUICallEngine

TUICallObserver

Type Definition

iOS

API Overview

TUICallKit

TUICallEngine

TUICallObserver

Type Definition

Web

API Overview

TUICallKit

TUICallEngine

TUICallEvent

Flutter

API Overview

TUICallKit

TUICallEngine

TUICallObserver

Type Definition

uniapp (Android&iOS)

API Overview

- TUICallKit

- TUICallEngine

- Event

- ErrorCode

- Release Notes (TUICallKit)

- Web

- Android and iOS

- Flutter

- FAQs(TUICallKit)

- Web

- All Platform

- Flutter

- Audio/Video Conference

- Overview (TUIRoomKit)

- Activate the Service (TUIRoomKit)

- Integration (TUIRoomKit)

- iOS

- Android

- Electron

- Windows

- Web

- Flutter

- UI Customization (TUIRoomKit)

- Electron

- Android

- iOS

- Windows

- Web

- Flutter

- Meeting Control (TUIRoomKit)

- API Documentation(TUIRoomKit)

- iOS&Mac

- API Overview

- TUIRoomKit

- TUIRoomEngine

- TUIRoomObserver

- Type Definition

- Android

API Overview

TUIRoomKit

TUIRoomEngine

TUIRoomObserver

Type Definition

## Windows

API Overview

TUIRoomEngine

TUIRoomObserver

Type Definition

## Web

API Overview

TUIRoomKit

TUIRoomEngine

TUIRoomEvents

TUIRoomEngine Defines

## Electron

API Overview

TUIRoomKit

TUIRoomEngine

TUIRoomEvent

TUIRoomEngine Defines

## Flutter

API Overview

TUIRoomObserver

TUIRoomEngine

Type Definition

## FAQs

Web

iOS

Android

Electron

Flutter

## Error Code

## Interactive Live Audio Streaming

Integrating TUIVoiceRoom (Android)

Integrating TUIVoiceRoom (iOS)

TUIVoiceRoom APIs

TRTCVoiceRoom (iOS)

TRTCVoiceRoom (Android)

Interactive Video Streaming

Integrating TUILiveRoom (Android)

Integrating TUILiveRoom (iOS)

Integrating TUIPusher and TUIPlayer (Web)

TUILiveRoom APIs

TRTCLiveRoom APIs (iOS)

TRTCLiveRoom APIs (Android)

Online Karaoke

Quick Integration (TUIKaraoke)

iOS

Android

Solution Overview (TUIKaraoke)

Implementation Steps

Song Synchronization

iOS

Android

Lyric Synchronization

iOS

Android

Vocal Synchronization

iOS

Android

Mixing Stream Solution

iOS

Android

TUIKaraoke APIs

TRTCKaraoke (iOS)

TRTCKaraoke (Android)

FAQs

iOS

Android

# Integration (UI Included)

## TUIKit Overview

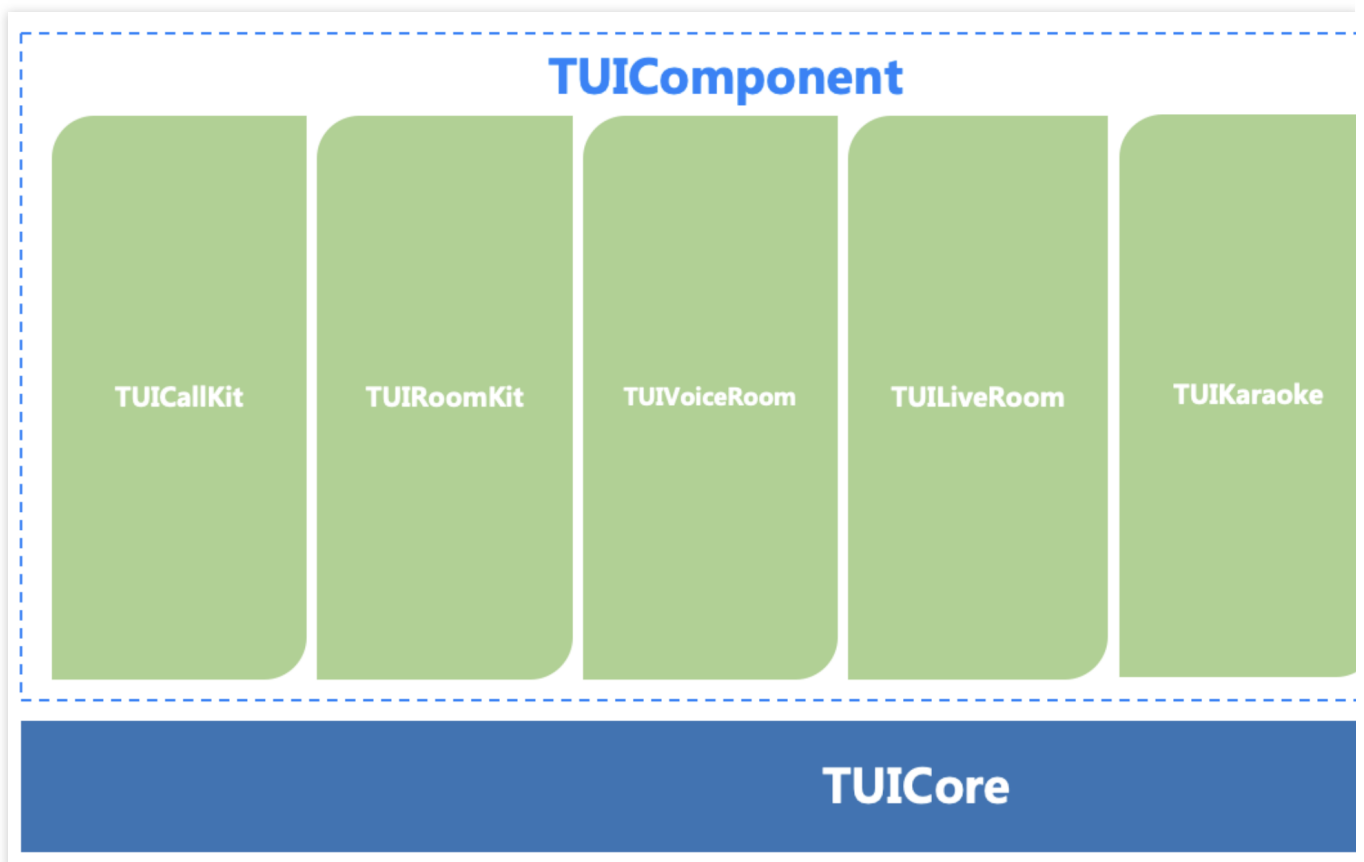
Last updated : 2024-03-06 15:48:15

TUIKit is an open-source solution developed by Tencent Cloud's Audio/Video Team based on its experience serving over 5,000 customers in major audio/video scenarios. It comes with multiple client components for applications such as video call, live streaming, and video room to help you quickly build services for call, customer service, live streaming, audio chat, and education scenarios.

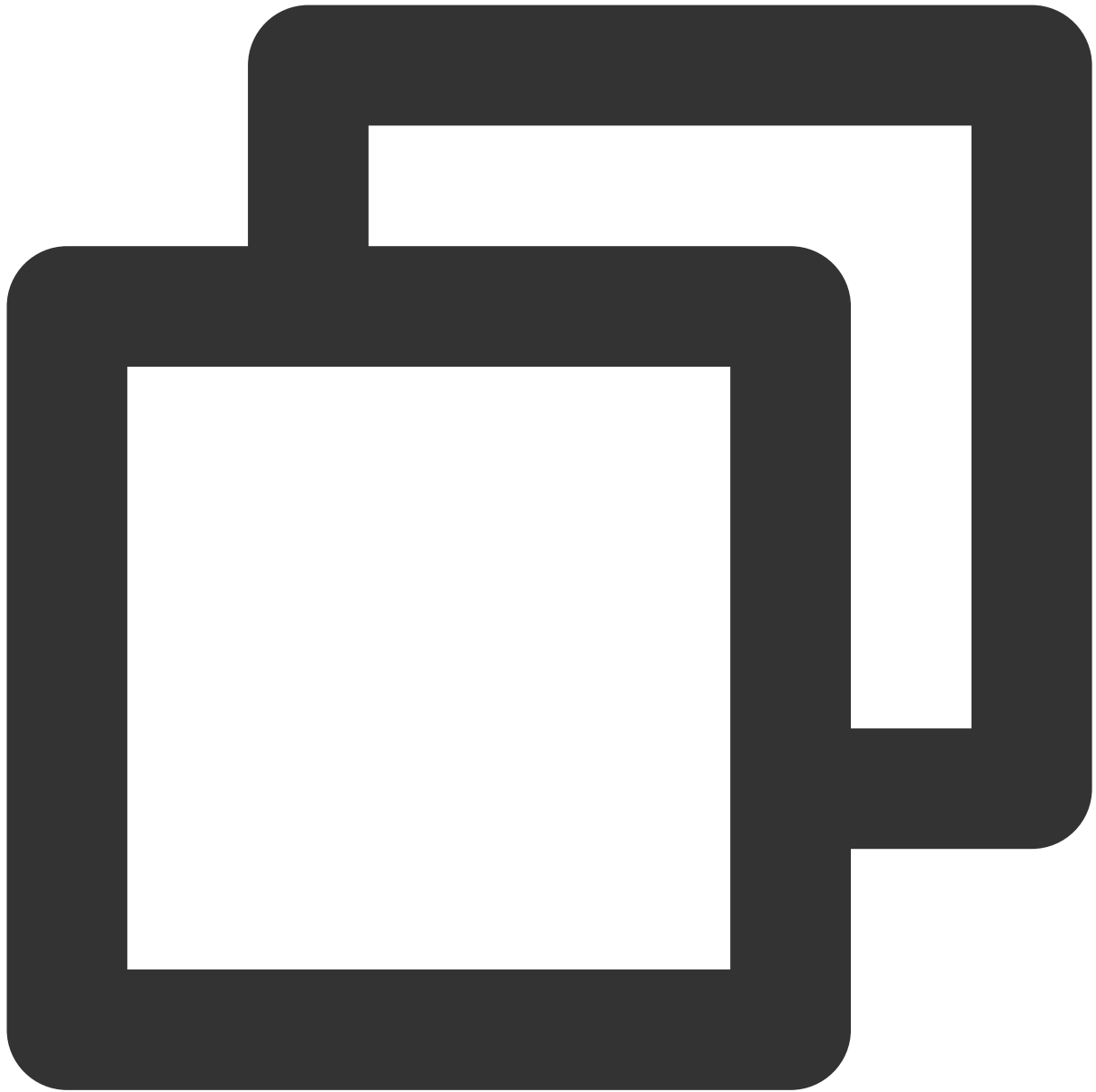
### Note

All components of TUIKit use two basic PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will be activated automatically. For the billing details of Chat services, see [Pricing](#).

## TUIKit Panorama



TUIKit is classified as `TUICompenont` and `TUIWidget` components.



```
├─ TUIComponent
|   ├─ TUICallKit      // Call component, which is similar to the call feature of W
|   ├─ TUIRoomKit      // Group video room component, which can be used in audio/vi
|   ├─ TUIVoiceRoom    // Voice chat component, which can be used in audio scenario
|   ├─ TUILiveRoom     // Video interactive live streaming component, which has fea
|   └─ TUIKaroke       // Karaoke component, an innovative component which you can
├─ TUIWidget
|   ├─ TUIBeauty       // Beauty filter widget
|   └─ TUIAudioEffect  // Sound effect widget
```

```
|   ├── TUIBarrage    // On-screen commenting widget
|   ├── TUIGift       // Gift widget
```

## Questions & Feedback

If you have any questions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).



# Audio/Video Call

## Overview (TUICallKit)

Last updated : 2024-04-03 17:23:11

### Component Overview

TUICallKit is an audio and video call UI component launched by Tencent Cloud. By integrating this component, you only need to write a few lines of code to add audio and video call features to your application.



### Supported Platform

Platform	Android	iOS	Web	Uni-app	Flutter	Uni-	WeChat
----------	---------	-----	-----	---------	---------	------	--------

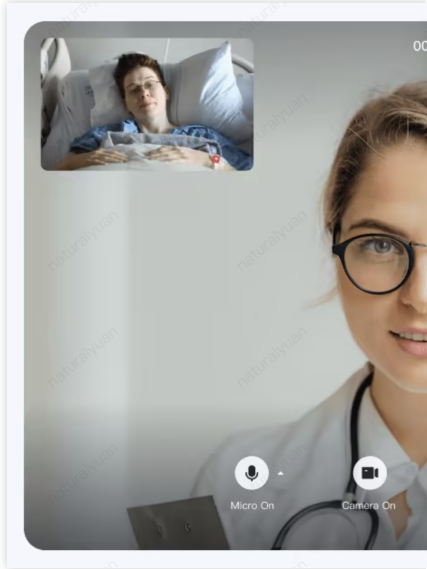
				Mini Program		app Client	Mini Program
Supported							
Supported Languages/Frameworks	Kotlin Java	Swift Objective-C	Vue3 Vue2 React	Vue3 Vue2	Dart	Vue3 Vue2	

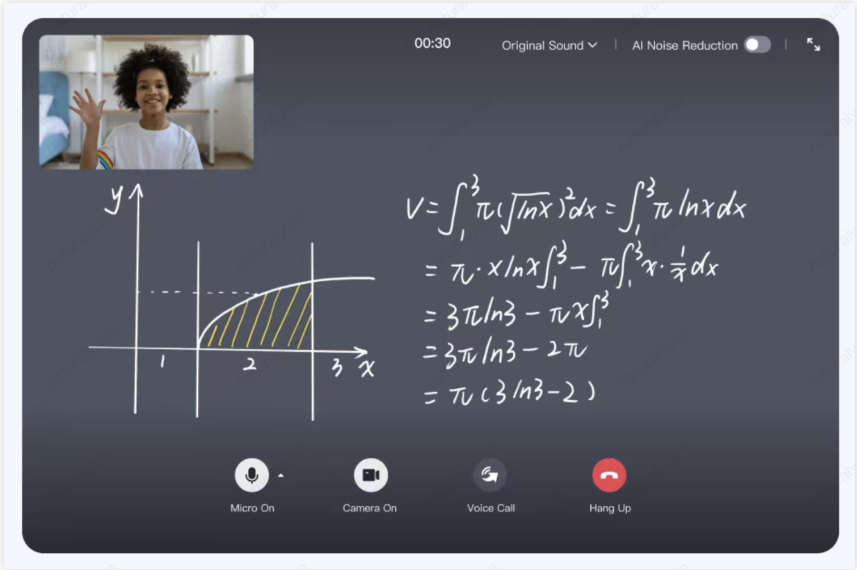
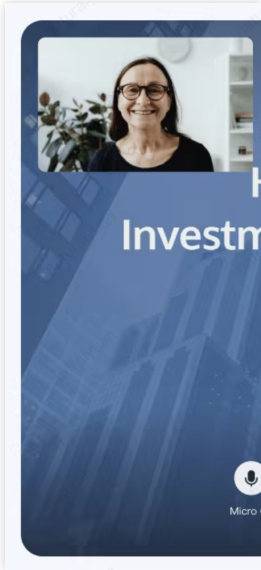
## Description of the Feature

Basic Feature	Advanced Feature	Feature Advantages
1v1 Voice/Video Call <b>Group Call</b> , Invite Others Mid-call, Join Mid-call Customize Incoming Call Ringtone Customize Nickname, Avatar <b>Enable/Disable Floating Window</b> Toggle On/Off the ringtone for incoming calls	<b>Offline Push</b> <b>Virtual Background</b> On-cloud recording AI Noise Reduction Global Interconnectivity Weak Network Jitter Optimization Call Records	<b>Comprehensive UI Interaction</b> <b>Cross-platform Interconnection Support</b> <b>Multi-device Login Support</b>

## Applicable Scenario

Online Social Interaction	Online Consultation



Online Education	Online Sales
	

Trying It Online

Platform	Web	Android	iOS	Others
Online Experience				/

<b>Demo Integration</b>	<a href="#">Github: web</a>	<a href="#">Github: Andorid</a>	<a href="#">Github: iOS</a>	<a href="#">Github: TUICallKit Demo</a>
-----------------------------	-----------------------------	---------------------------------	-----------------------------	---

## Exchange and Feedback

If you have any requirements or feedback, you can contact: [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# Activate the Service (TUICallKit)

Last updated : 2024-04-03 18:02:21

## Activate and purchase guide

### Free trial

In order for you to better experience the Call features, we provide a 7-day Call trial edition for free, including free feature usage rights and **10,000 free minutes**. Each `SDKAppID` can be tried twice for free with a 7-day validity period for each trial, and up to 10 `SDKAppID` can be tried under one account.

You can refer to the following guidelines to activate the trial edition of Call.

1. Visit [TRTC Console > Applications](#), select **Create application**.

The screenshot shows the 'Applications' page in the TRTC Console. A red box highlights the 'Applications' link in the left sidebar. The main content area has a warning banner about payment methods. Below it, the 'My Applications' section contains a table with the following data:

Application name	SDKAppID	Status	Product information	Expiration time	SDKSecretKey	Tags
[Blue Icon]	[Grey Icon]	Enabled	Call : Trial Chat : TRTC Development	2024-01-12 2024-02-05	*****	
TUICallKit	[Grey Icon]	Enabled	Call : Trial Chat : TRTC Development	2024-01-11 2024-02-04	*****	

2. In the Create application pop-up, select **Call** and enter the application name, click **Create**.

## Create application

Select product



**Call**

UI Ex.

Add high-quality video and voice calling quickly and easily



**Conference**

UI Ex.

Incorporate meetings for unlimited audience to collaborate together



**RTC Engine**

Integrate calling and interactive live streaming features

Part of the underlying technology is supported by Chat, so we will enable it when you get started

Application name

TUICallKit

Create

3. After completing the application creation, you will default entry to the application details page, select the **Free Trail** in the floating window, and click to **Get started for free**.

## Select an Call edition

You need to choose a version to use Call properly. We recommend you to get the free trial version with all features, or you can directly purchase other versions.



**Free Trial**

- ✓ 7 days validity, gifted call duration 10,000 minutes/month
- ✓ Experience all features of Audio/Video call

\$0



**1-to-1 Call**

- ✓ Gifted call duration 200,000 minutes/month
- ✓ Complete call UI, supporting 1-to-1 video call, audio call, and other functions

/mo.



**Group Call**

- ✓ Gifted call duration 600,000 minutes/month
- ✓ Support group video/audio calls, multi-end login calls on the same platform, and other functions

/mo.

Get started for free

4. Confirm the pop-up content, click to **Free Trail**, and you can successfully activate the Call Trail edition.



### Free trial of TRTC Call



TRTC Call (TUICallKit) is a UI component for audio/video calling. After integrating it and writing a few simple lines of code, your application will be able to support 1-to-1/Group calls, multi-platform call, and offline call push and other features.



- Call provides a **Free trial with 7 days validity**, allowing you to experience all features of Call for free.
- Some underlying capabilities of Call are based on Chat. If your application has not yet activated Chat, this will activate the Chat development version (free) for you; if your application has already activated Chat, this will not change your existing Chat version and configuration.
- Call Free version **only provides you with the right to use the above functions for free. The Usage time generated during your use will be billed according to [Audio/Video call time](#)**, and the Chat fees will be billed according to [Chat Billing overview](#).
- Tencent Cloud accounts (UIN) that have activated Call will receive a free 10,000-minute [Package of Time](#) every month, which can be used to offset the usage time generated during use. For more information, see [Audio and Video Call](#).

Free Trail

5. After the activation is completed, you can view the edition information on the current page and refer to the [integration guide](#) for integration.

The `SDKAppID` and `SDKSecretKey` here will be used in the integration guide.

The screenshot shows the 'Application Overview' page for an application named 'TUICallKit'. On the left sidebar, 'Application Overview' is highlighted. The main content area has a header 'Application Overview' with a dropdown menu showing '- TUICallKit'. Below the header, a warning message states: 'You haven't provided a payment method. We will suspend the service for your account after you use up your free resources. To avoid service in...'. The 'Basic Information' section contains the following details:

Field	Value
Application name	TUICallKit
SDKAppID	[Redacted]
SDKSecretKey	*****
Description	-- [Edit]
Tags	No tags yet
Status	Enabled [More]
Creation time	2024-01-04 17:43:49
Region	Global (excluding Chinese mair)

## Purchase the official editions of Call

1. Visit the [TRTC Call Purchase page](#), select the Application (SDKAppID) and Call Package you want to purchase. At the same time, we recommend you to enable Auto-renewal to avoid affecting business use. After Confirming purchase information and agreeing to the relevant agreement, check the agreement content and click **Subscribe now**.



Tencent Real-Time Communication

Order

Talk to us

Console

## Call Monthly Packages

Application (SDKAppID) ⓘ  

Create Application

Package editions [Detail](#)  

### 1-to-1 Call

- 20,000 FREE mins included
- 1-to-1 audio/video call
- Complete UI
- Floating window

Most Popular

### Group Call

- 60,000 FREE mins included
- All 1-to-1 Call features
- Group audio/video call
- AI noise suppression


☒ Automatic renewal  
When the package expires, it will be automatically renewed for one month if your account has sufficient balance

☒ I have read and agree to [Tencent Real-Time Communication Service Level Agreement](#) and [Tencent Cloud Instant Messaging Service Level Agreement](#).

Subscribe now

2. Go to the order confirmation page to confirm product information. The underlying capabilities of TRTC Call rely on the Chat service. Therefore, a TRTC Call monthly package is purchased together with a [Chat plan](#) by default, and the order will include 2 types of products (as shown below).

## Please confirm the following product information

 The following orders contain promotional orders for which vouchers cannot be applied. Promotional orders will automatically end after 13 Minute 18 seconds. Please make the payment before the order ends.

### Order

#### TRTC Call

Plan: Group call

Unit Price: USD/month

Quantity: 1

Payment Mode: Prepaid

Term: 1month

#### Chat Premium edition plan

Plan: Premium edition plan

Unit Price: 10USD/month

Quantity: 1

Payment Mode: Prepaid

Term: 1month

### Check the Fees

TRTC Call x1 USD

Chat Premium edition plan x1 USD

Remaining Fees: USD/hours

5USD/hours

Upfront Payment 10 USD

Tax: +0.00 USD

Total USD

Pay Now

3. Go to the payment page to complete the payment. After the purchase is completed, you can go to the [TRTC Console > Call](#) to view application edition information, and refer to the [Integration guide](#) for integration.

### Note :

If you have already created an application in Chat and purchased the paid edition of Chat, Please refer to [Chat documentation](#) to learn **how to purchase and use TRTC Call**.

## Renew the official edition

Refer to the [Purchasing the official edition of Call](#) and purchase the same edition of the Call monthly package again to complete the renewal.

It is recommended that you renew Call by enabling auto-renewal. **When the account balance is sufficient, it will automatically renew monthly after expiration.** Since some underlying services of Call are supported by Chat, auto-renewal for both Call and Chat will be enabled simultaneously. You can enable auto-renewal when purchasing or enable it in the console after the purchase is completed.

Tencent Real-Time Communication

Order

Talk to us

Console

## Call Monthly Packages

Application (SDKAppID) ⓘ  
[SDKAppID] Create Application

Package editions [Detail](#)

**1-to-1 Call**

- 20,000 FREE mins included
- 1-to-1 audio/video call
- Complete UI
- Floating window

**Group Call** Most Popular

- 60,000 FREE mins included
- All 1-to-1 Call features
- Group audio/video call
- AI noise suppression

☒ Automatic renewal  
When the package expires, it will be automatically renewed for one month if your account has sufficient balance

☒ I have read and agree to [Tencent Real-Time Communication Service Level Agreement](#) and [Tencent Cloud Instant Messaging Service Level Agreement](#). [Terms](#) [Subscribe now](#)

The specific steps to enable auto-renewal in the console are as follows:

1. Access the [TRTC Console > Applications](#), find the application and click **manage** to enter the application details page.

Overview

Applications

Usage Statistics

Data Monitoring

Package Management

Relevant Services

UserSig Tools

### Applications

My Applications

Search Application

Application name	SDKAppID	Status	Product information	Expiration time	SDKSecretKey	Tags
[Application Name]	[SDKAppID]	Enabled	Call : 1-to-1 Call Chat : Standard - Prepaid	2024-02-15 2024-02-16	*****	
[Application Name]	[SDKAppID]	Enabled	Call : 1-to-1 Call Chat : Standard - Prepaid	2024-02-15 2024-02-16	*****	
[Application Name]	[SDKAppID]	Enabled	Call : No version Chat : TRTC Development	-- 2023-11-19	*****	

2. In the Call product information, click the **Enable auto-renewal** button, a confirmation pop-up will appear, click **Confirm activation**.

**Application Overview** 29002093 - TEST

**Basic Information**

Application name	TEST	SDKSecretKey	*****
SDKAppID ⓘ	██████████	Creation time	2024-01-16 10:36:23
Description	--	Region	Global (excluding Chinese mainland)
Tags ⓘ	No tags yet	Application source	Auto-created when activating TRTC service in the IM console
Status	Enabled <span>More ▾</span>		

**Advanced Features**

- On-cloud recording ⓘ
- Relay to CDN
- Callbacks ⓘ
- Advanced permission c

**Products** [Quickly run sample demo in 3 steps >](#)

**Call**

Edition: [Call : 1-to-1 Call >](#)

Expiration time: 2024-02-15 ⓘ

Auto-renewable: Not enabled ⓘ **Enable**

[Renewal](#) [Integrate](#)

**Chat**

Edition: [Chat : Standard - Prepaid](#)

Expiration time: 2024-02-16

Auto-renewable: --

[Renewal](#) [Integrate](#)

**Add more products**

- Conference** Incorporate meetings for unlimited audience to collaborate together
- RTC Engine** Integrate calling and interactive live streaming features with RTC SDK
- In-game Voice Chat** Immersive In-Game Voice Chat: Enhance Your Gaming Experience

**Are you sure you want to enable auto-renewal?** ✕

After enabling auto-renewal, if the account balance is sufficient, it will automatically renew monthly after expiration. Since some underlying services of Call are provided by Chat, **this time will also enable auto-renewal for both Call and Chat.**

**Confirm activation** [Cancel](#)

## Features and Pricing of TRTC Call

TUICallKit is supported by Tencent Real-Time Communication TRTC and Instant Messaging at the underlying technical level. **You need to purchase a specific plan to use TUICallKit**, with the method to activate the free trial version available at [Activate TUICallKit Trial Version](#).

The table below shows the recommended packages for TUICallKit in terms of features and prices. You can also freely combine TRTC Monthly Packages and IM versions to get the corresponding version of TUICallKit. The Basic version of TRTC when combined with Professional or Enterprise Editions of IM can utilize the 1v1 Call Version of TUICallKit; The Deluxe and Enterprise Editions of TRTC combined with Professional or Enterprise Editions of IM can utilize the Group Call Version of TUICallKit.

Recommended Package		Trial Edition	1V1 Call Version	Group Call Version 380,000 minutes	Group Call Version 1.4 million minutes
Price		Free for 7 days	<a href="#">Buy Now</a>	<a href="#">Buy Now</a>	<a href="#">Buy Now</a>
Includes Package Version		TRTC Trial Version + IM Any Version	TRTC Basic Version + IM Professional Version	TRTC Deluxe Version + IM Enterprise Edition	TRTC Enterprise Edition + IM Enterprise Edition
Free Resources (Post-paid beyond threshold)	Call duration	-	110,000 minutes/month	380,000 minutes/month	1,400,000 minutes/month
	DAUs	Depends on IM Version	10,000 DAU	10,000 DAU	10,000 DAU
	Peak group count	Depends on IM Version	100,000 per month	100,000 per month	100,000 per month
TUICallKit reserved feature	1v1 Video, Audio Call	✓	✓	✓	✓
	WeChat-like UI Design	✓	✓	✓	✓
	Call Status Display	✓	✓	✓	✓
	Offline Push	✓	✓	✓	✓

(Even when the application is not open, users can receive audio and video ring calls through message push)				
Call Floating Window (After the call interface is collapsed, it can float above the application interface in a floating window style)	✓	✓	✓	✓
Custom Call Ringtone	✓	✓	✓	✓
Call/Answer/Reject/Hang Up	✓	✓	✓	✓
Switch from Video Call to Voice Call	✓	✓	✓	✓
Group call	✓	-	✓	✓
Mid-call/Join a three-way call	✓	-	✓	✓
Call Records (Kanban + API)	✓	-	✓	✓
Mini Program Call Acceleration (Supports Mini Program End Call and Mini Program Room Entry Call Acceleration Optimization, Ensuring Call Quality)	✓	-	✓	✓
Multi-client Sign in Call (Answer on any platform, automatically stop access requests from other clients)	✓	-	✓	✓
Cross-platform multi-client Sign in Call	✓ (For use with IM)	-	✓	✓

	(Supports the same platform, such as multiple iOS device Sign in, after receiving a call, any device answers, automatically stops the access requests from other devices)	flagship version only)			
	AI Noise Reduction (Effectively filter background noise and restore call quality with AI capabilities)	✓	-	✓	✓
	Weak Network Call Lagging Optimization (Optimizes lagging rate in outdoor and other weak network environments for faster instantaneous loading speed)	✓	-	✓	✓
<b>Instant Messaging feature</b>		Depends on IM Version <a href="#">See IM feature</a>	See <a href="#">IM Professional Version feature</a>	See <a href="#">IM Ultimate Edition feature</a>	See <a href="#">IM Ultimate Edition feature</a>
<b>Tencent Real-Time Communication TRTC feature</b>		See <a href="#">TRTC Trial Version feature</a>	See <a href="#">TRTC Basic Version feature</a>	See <a href="#">TRTC Premium Version feature</a>	See <a href="#">TRTC Flagship Version feature</a>
<b>Post-paid for the part exceeding the free resources</b>		For post-paid prices of call durations, see <a href="#">Audio and Video Duration Billing Explanation</a> , For DAU and peak group numbers' post-paid prices, see <a href="#">IM Billing Explanation</a>			
<b>Supported Platforms</b>		iOS, Android, Web, <b>WeChat Mini Program (only supported by the trial and group call versions)</b> , uni-app, Flutter			

**Notes:**

**Includes Package Versions:** TUICallKit is jointly provided by TRTC and IM. To use TUICallKit, you must purchase both TRTC monthly packages and IM basic packages. Different versions offer distinct TUICallKit features.

**DAU:** Each individual user who signs in to Instant Messaging is counted as 1 DAU for that day. DAUs are not incremented for repeated sign-ins by the same user.

**Peak Group Count:** The sum of all groups created and joined by users under a single application (SDKAppID), with billing based on the highest peak value during a calendar month.



# Integration (TUICallKit)

## Android

Last updated : 2024-04-03 17:23:11

This document describes how to quickly integrate the `TUICallKit` component. Performing the following key steps generally takes about ten minutes after which you can implement the video call feature with complete UIs.

## Environment Preparations

Android 5.0 (SDK API level 21) or later.

Gradle 4.2.1 or later

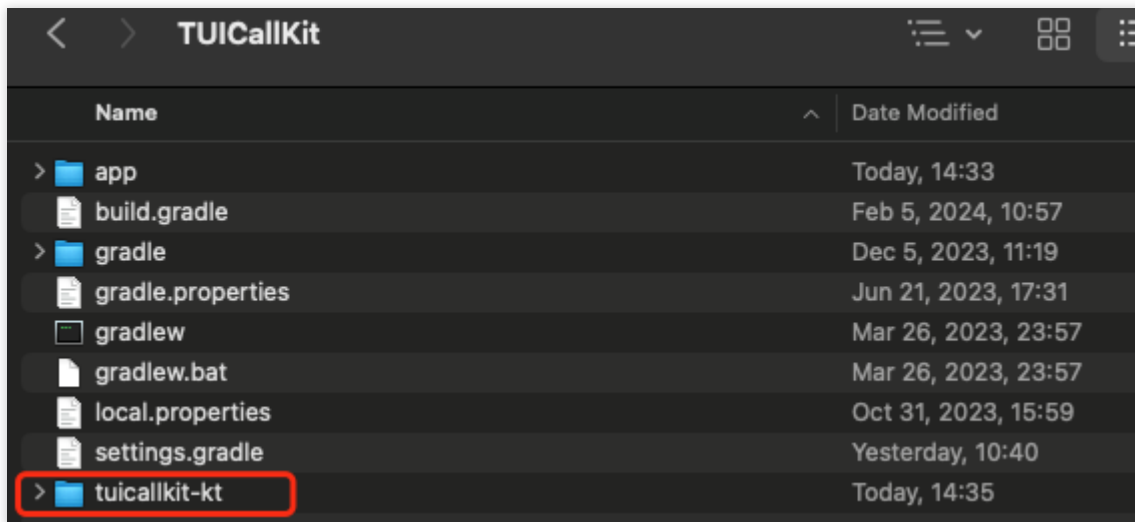
Mobile phone on Android 5.0 or later.

## Step 1. Activate the service

Before using the Audio and Video Services provided by Tencent Cloud, you need to go to the Console and activate the service for your application. For detailed steps, refer to [Activate Service](#).

## Step 2. Download and import the component

Go to [GitHub](#), clone or download the code, and copy the `tuicallkit-kt` subdirectory in the `Android` directory to the directory at the same level as `app` in your current project, as shown below:

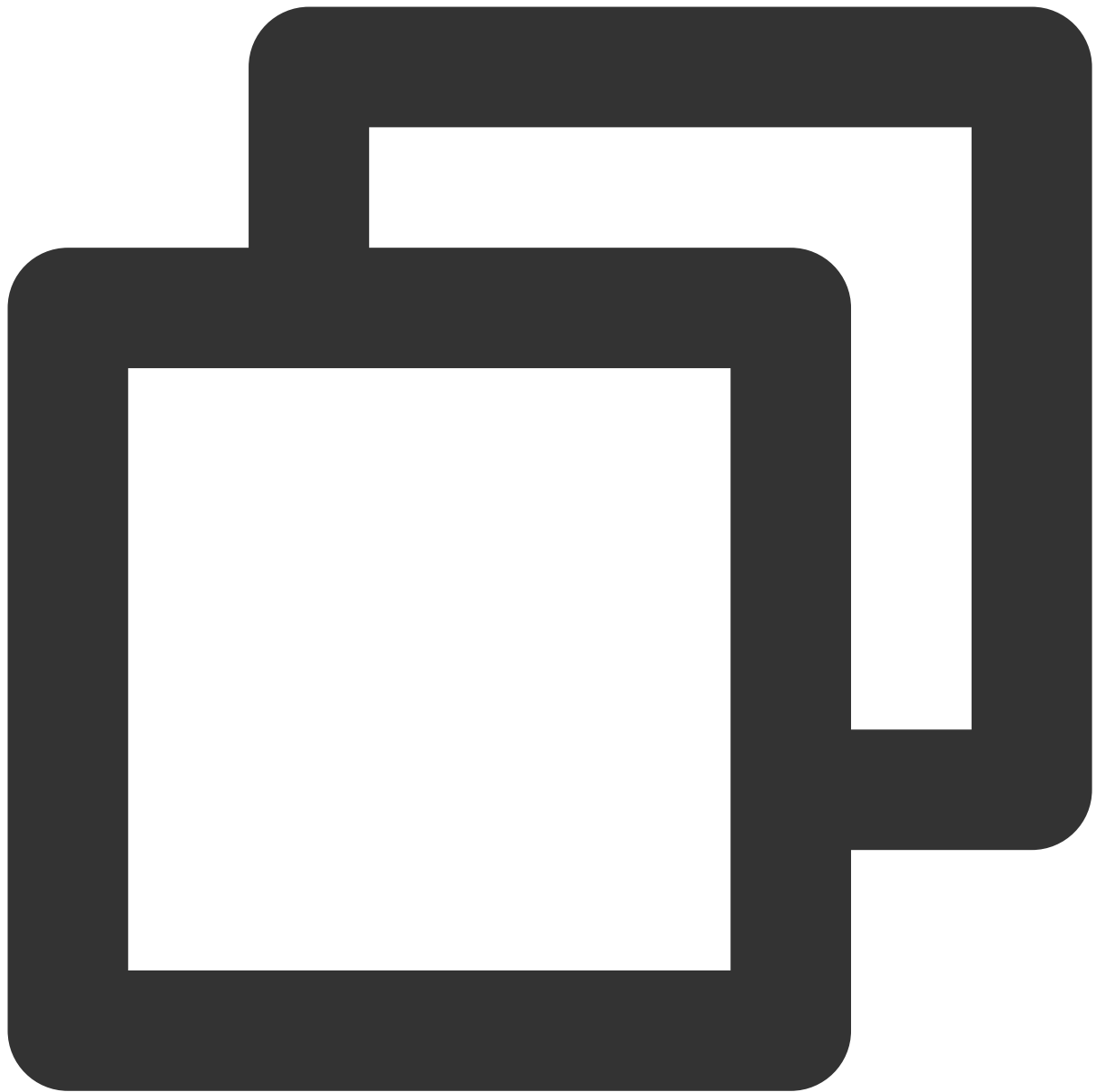


### Step 3. Configure the project

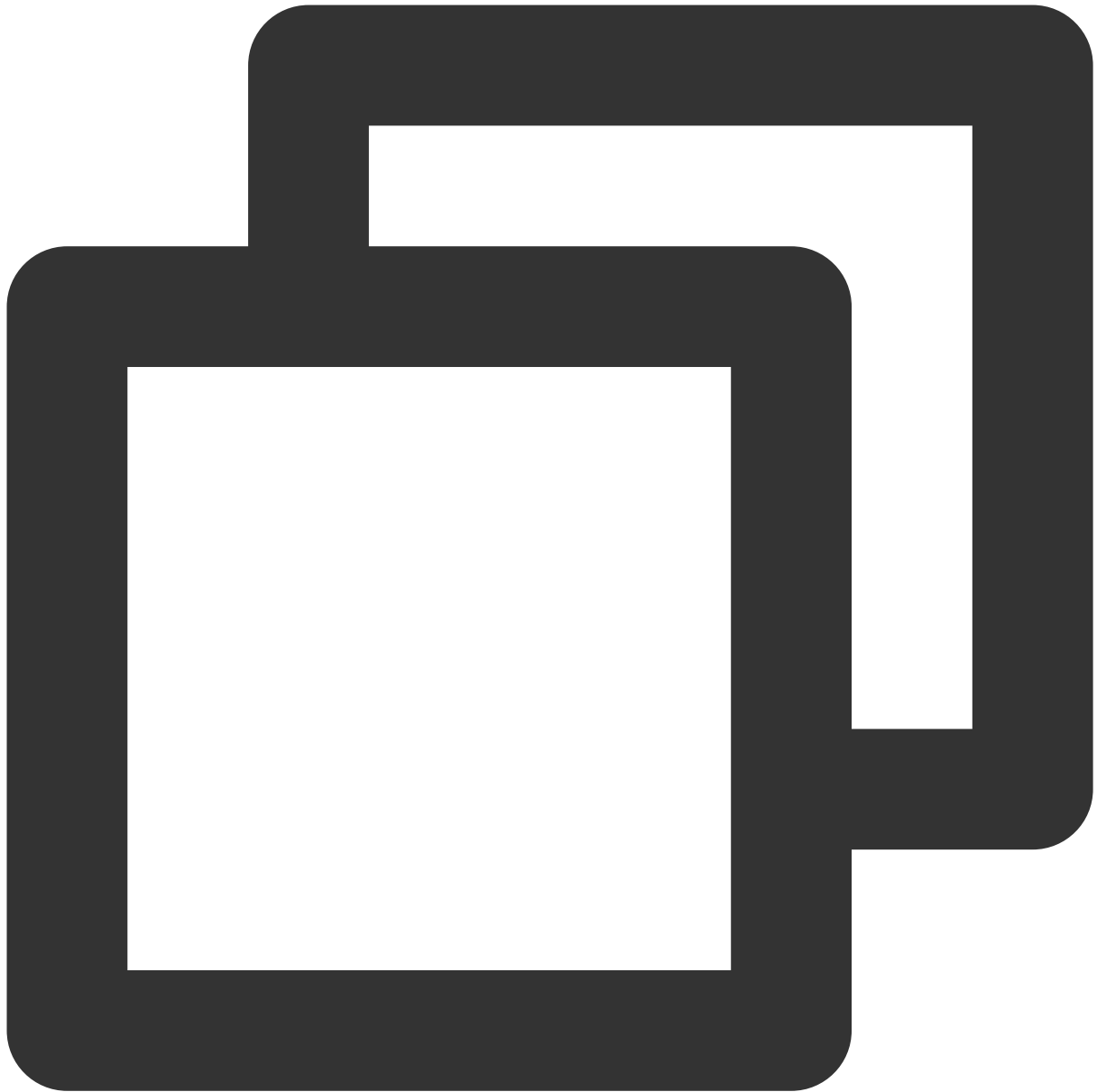
1. Find the `settings.gradle` (or `settings.gradle.kts`) file in the project root directory and add the following code to import the component downloaded in [step 2](#) to your current project:

`settings.gradle`

`settings.gradle.kts`



```
include ':tuicallkit-kt'
```

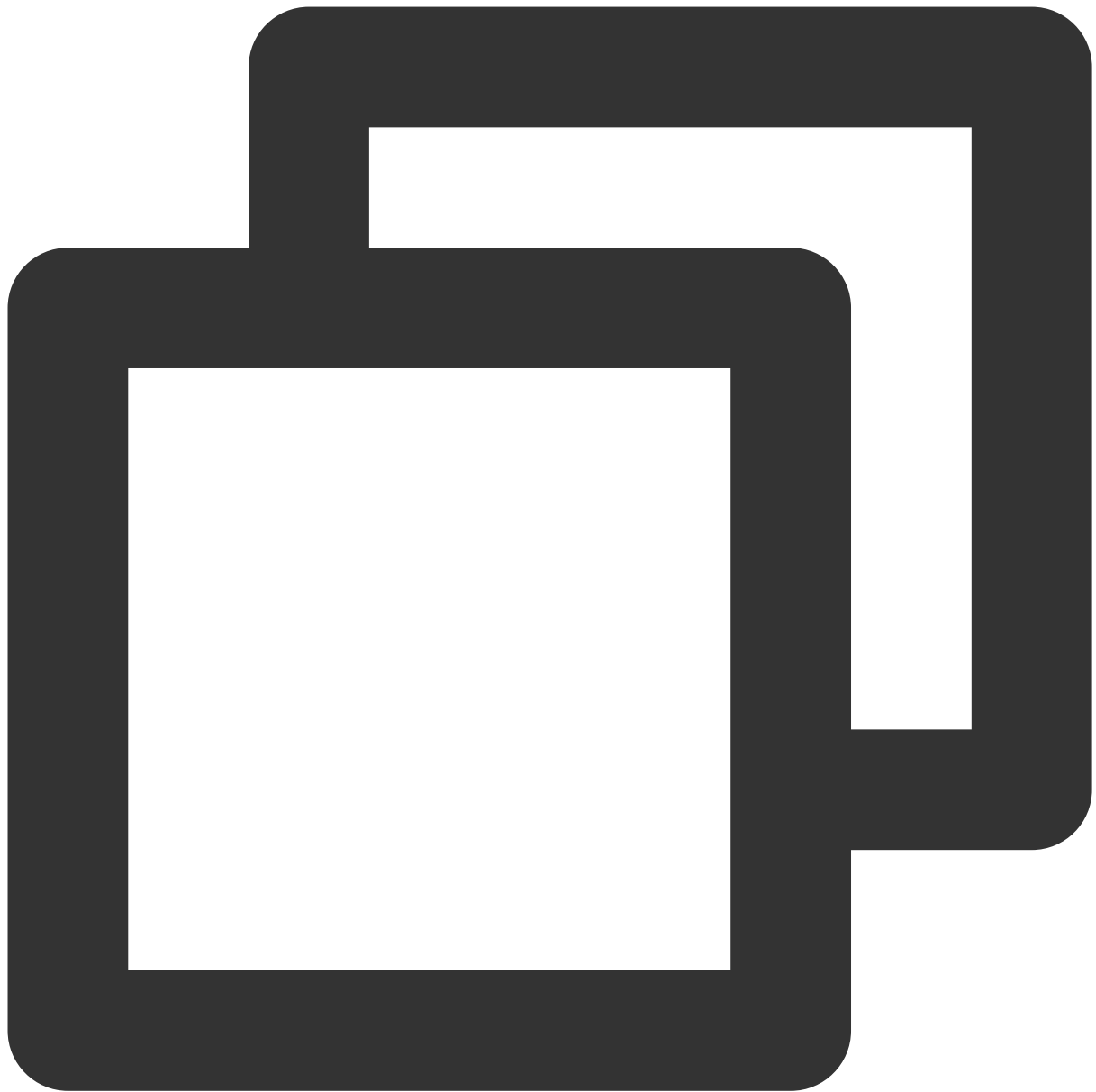


```
include(":tuicallkit-kt")
```

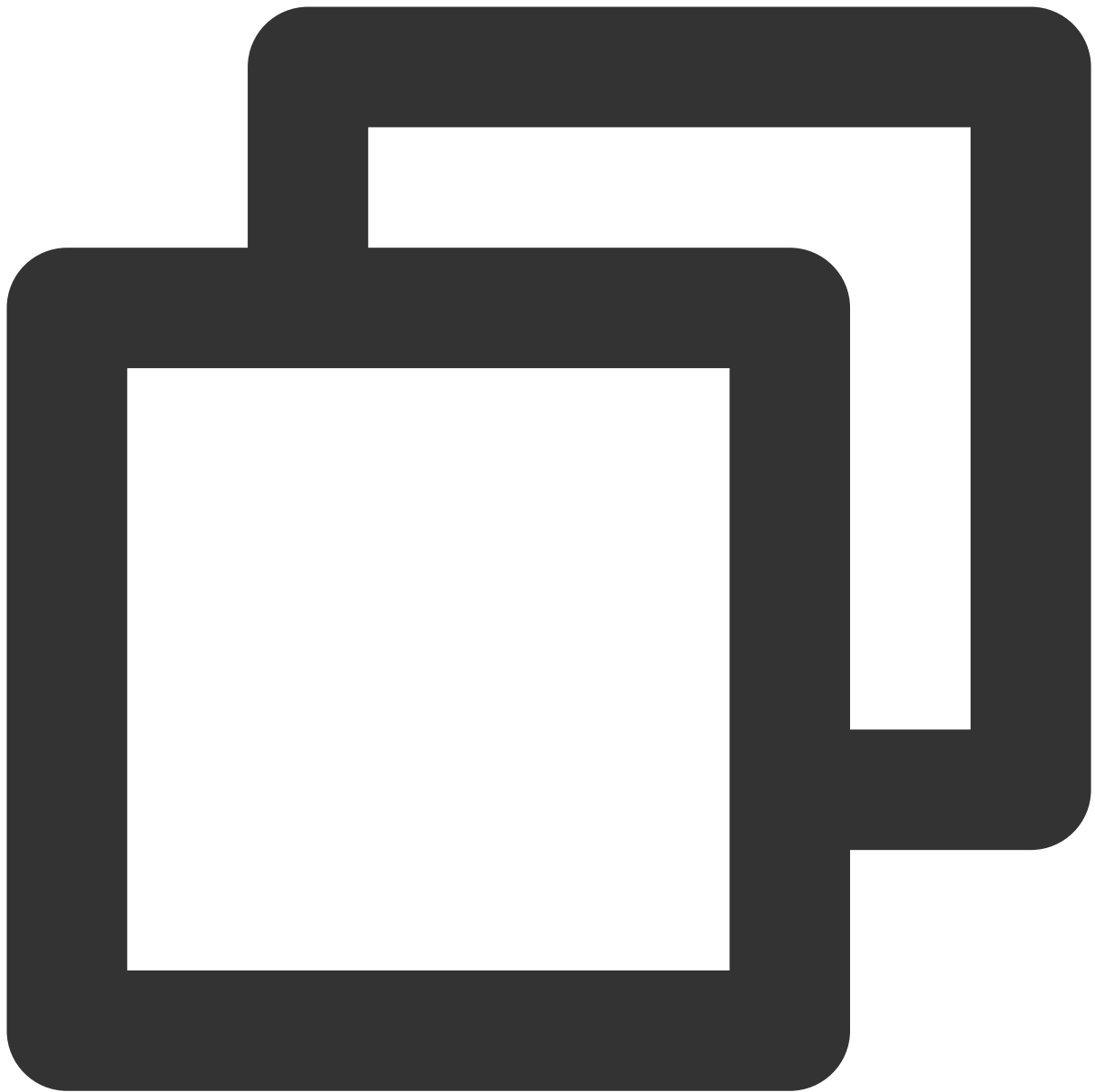
2. Find the `build.gradle` (or `build.gradle.kts`) file in the `app` directory and add the following code to declare the dependencies of the current application on the component just added:

`build.gradle`

`build.gradle.kts`



```
api project(':tuicallkit-kt')
```



```
api(project(":tuicallkit-kt"))
```

**Note**

The `TUICallKit` project depends on `TRTC SDK`, `Chat SDK`, `tuicallengine`, and the `tuicore` public library internally by default with no need of additional configuration. To upgrade the version, modify the version in `tuicallkit-kt/build.gradle` file.

3. As the SDK uses Java's reflection feature internally, you need to add certain classes in the SDK to the obfuscation allowlist by adding the following code to the `proguard-rules.pro` file:



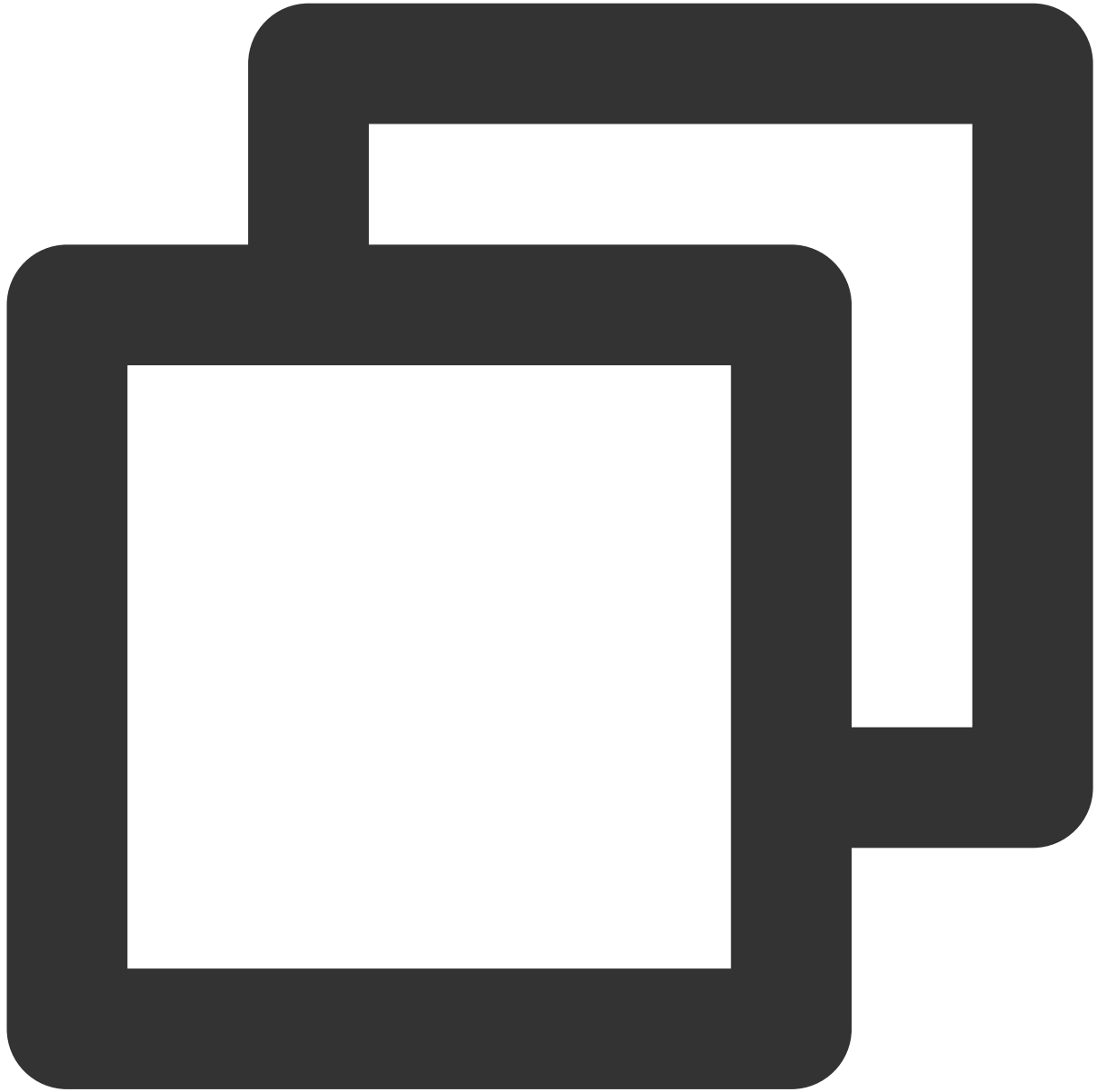
```
-keep class com.tencent.** { *; }
```

#### Note

`TUICallKit` helps you apply for camera, mic, and bluetooth permissions internally. If you need more or fewer permissions based on your actual business conditions, you can modify `tuicallkit-kt/src/main/AndroidManifest.xml`.

## Step 4. Log in to the `TUICallKit` component

Add the following code to your project to call the relevant APIs in `TUICore` to log in to the `TUICallKit` component. This step is very important, as the user can use the component features properly only after a successful login. Carefully check whether the relevant parameters are correctly configured:



```
TUILogin.login(context,
    1400000001,    // Replace it with the `SDKAppID` obtained in step 1.
    "denny",      // Replace it with your `UserID`.
    "xxxxxxxxxxx", // You can calculate a `UserSig` in the console and enter it here
    object : TUICallback() {
        override fun onSuccess() {
        }
    })
```



```
        override fun onError(errorCode: Int, errorMessage: String) {  
        }  
    }  
}
```

### Parameter description:

The key parameters used by the `login` function are as detailed below:

**SDKAppID:** Obtained in the last step in step 1 and not detailed here.

**UserID:** The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), or underscores (\_).

**UserSig:** The authentication credential used by Tencent Cloud to verify whether the current user is allowed to use the TRTC service. You can get it by using the `SDKSecretKey` to encrypt the information such as `SDKAppID` and `UserID`. You can generate a temporary `UserSig` by clicking the [UserSig Generate](#) button in the console.

The screenshot shows the 'UserSig Tools' interface in the Tencent Cloud console. At the top, there is a warning message: 'You haven't provided a payment method. We will suspend the service for your account after you use up your free resources. To avoid service interruption, please [complete your information](#) and [refresh](#).' Below this, there are two main sections: 'Signature (UserSig) Generator' and 'Signature (UserSig) Verifier'. The 'Generator' section includes fields for 'Application (SDKAppID)' (a dropdown menu), 'Username (UserID)' (a text input), and 'Secret key' (a text input with a note 'Auto-generated after you select an application'). There is a 'Generate' button and a 'Generate result' section with a 'Copy' button. The 'Verifier' section includes similar fields for 'Application (SDKAppID)', 'Username (UserID)', and 'Secret key', and a 'UserSig' text input field with a 'Verify' button.

For more information, see [UserSig](#).

### Note

**Many developers have contacted us with many questions regarding this step. Below are some of the frequently encountered problems:**

SDKAppID is invalid.

userSig is set to the value of Secretkey mistakenly. The userSig is generated by using the SecretKey for the purpose of encrypting information such as sdkAppId, userId, and the expiration time. But the value of the userSig that is required cannot be directly substituted with the value of the SecretKey.

userId is set to a simple string such as 1, 123, or 111, and your colleague may be using the same userId while working on a project simultaneously. In this case, login will fail as TRTC doesn't support login on multiple terminals with the same UserID. Therefore, we recommend you use some distinguishable userId values during debugging.

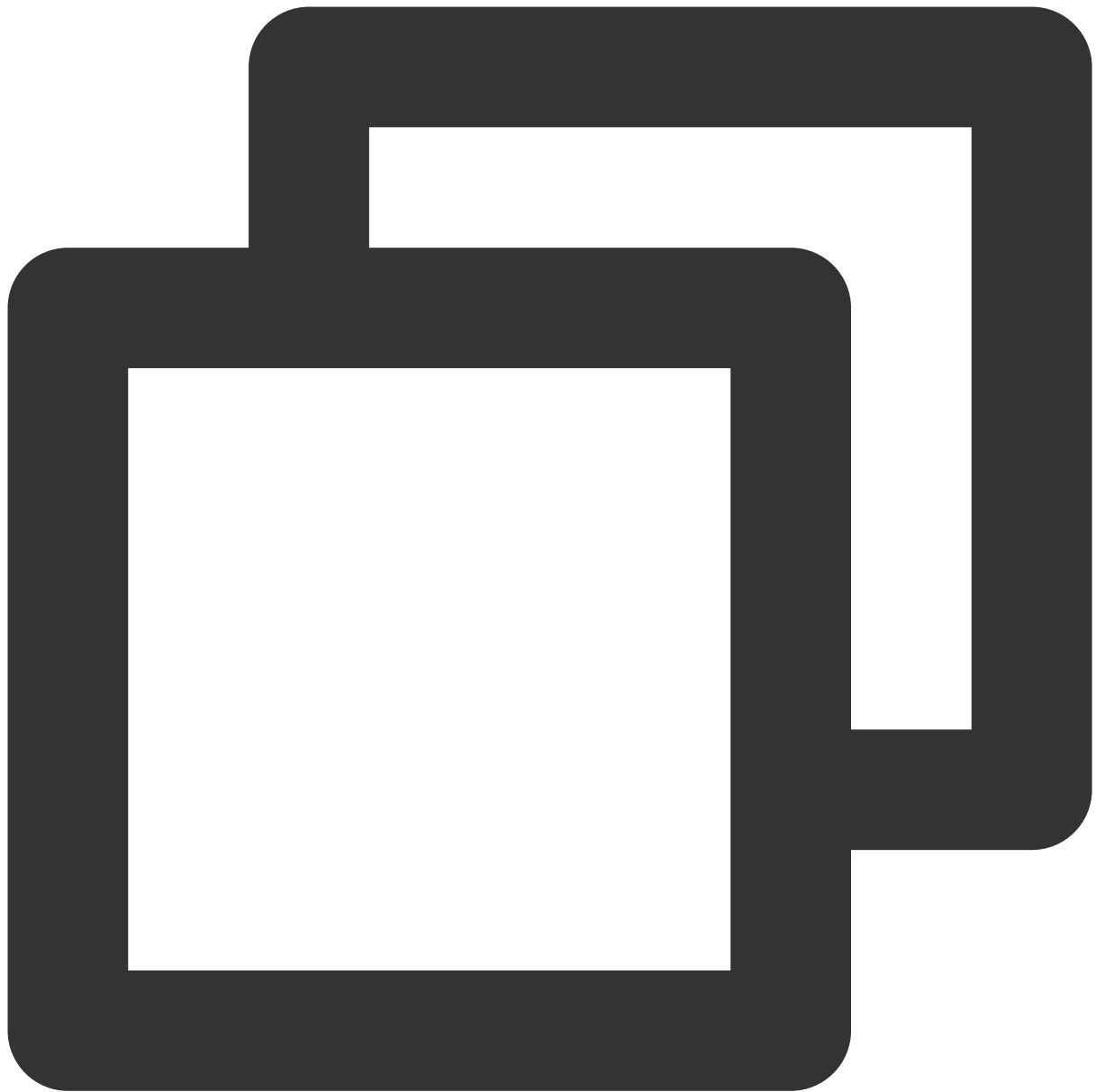
The sample code on GitHub uses the `genTestUserSig` function to calculate `UserSig` locally, so as to help you complete the current integration process more quickly. However, this scheme exposes your `SecretKey` in the application code, which makes it difficult for you to upgrade and protect your `SecretKey` subsequently. Therefore, we strongly recommend you run the `UserSig` calculation logic on the server and make the application request the `UserSig` calculated in real time every time the application uses the `TUICallKit` component from the server.

## Step 5. Make your first phone call

After both the caller and callee have successfully signed in, the caller can initiate an audio or video call by calling the `TUICallKit`'s call method and specifying the call type and the callee's userId. At this point, the callee will receive an incoming call invitation.

Kotlin

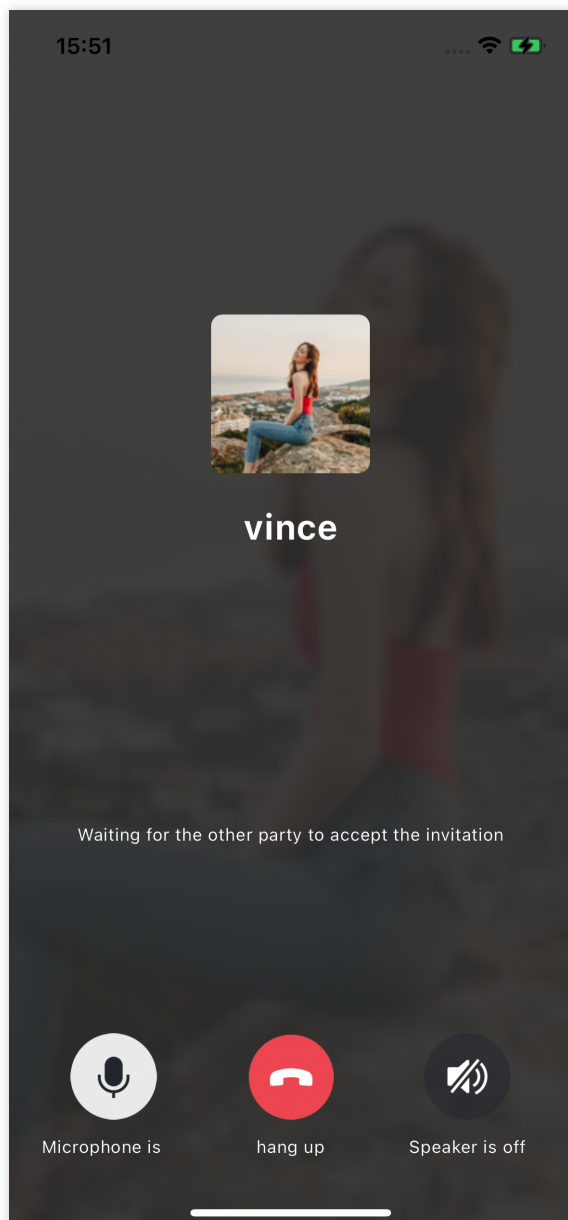
Java



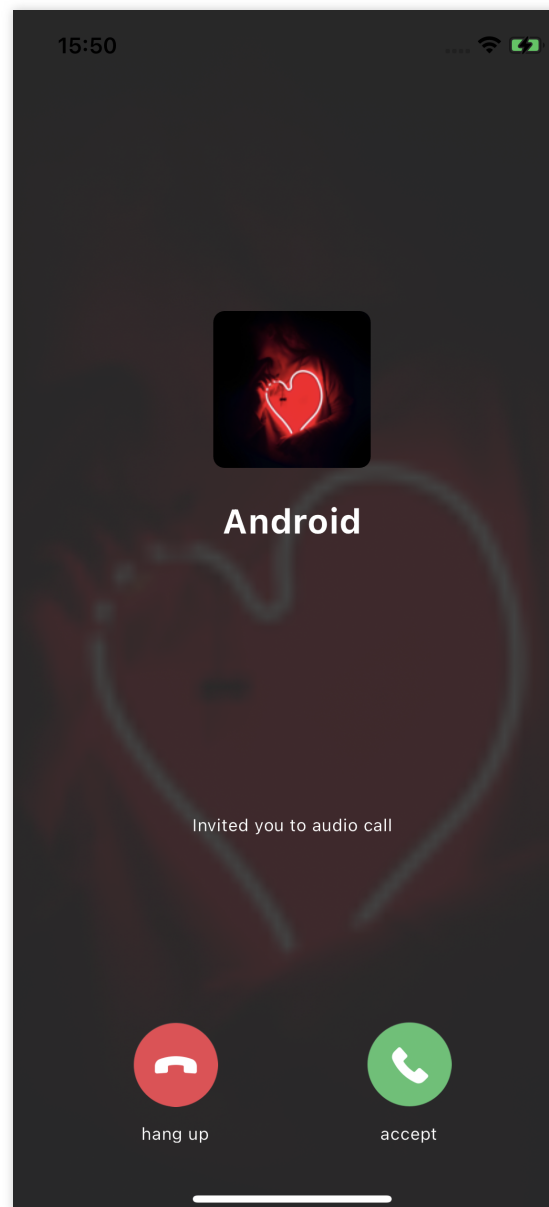
```
// Initiate a one-to-one video call (assuming UserID is mike)
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Video)
```



```
// Initiate a one-to-one video call (assuming UserID is mike)
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Video);
```



Caller



Callee

## Additional Features

[Customize Interface](#)[Offline Push](#)[Group Call](#)[Floating Window](#)[Custom Ringtone](#)[Call Status Monitoring](#)[Cloud Recording](#)

## FAQs

If you encounter any issues during integration and use, please refer to [Frequently Asked Questions](#).

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# iOS

Last updated : 2024-04-15 10:45:42

This document describes how to quickly integrate the `TUICallKit` component. Performing the following key steps generally takes about an hour, after which you can implement the video call feature with complete UIs.

## Environment Preparations

Xcode 13 or later

iOS 12.0 or later

## Step 1. Activate the service

Before using the Audio and Video Services provided by Tencent Cloud, you need to go to the Console and activate the service for your application. For detailed steps, refer to [Activate Service](#).

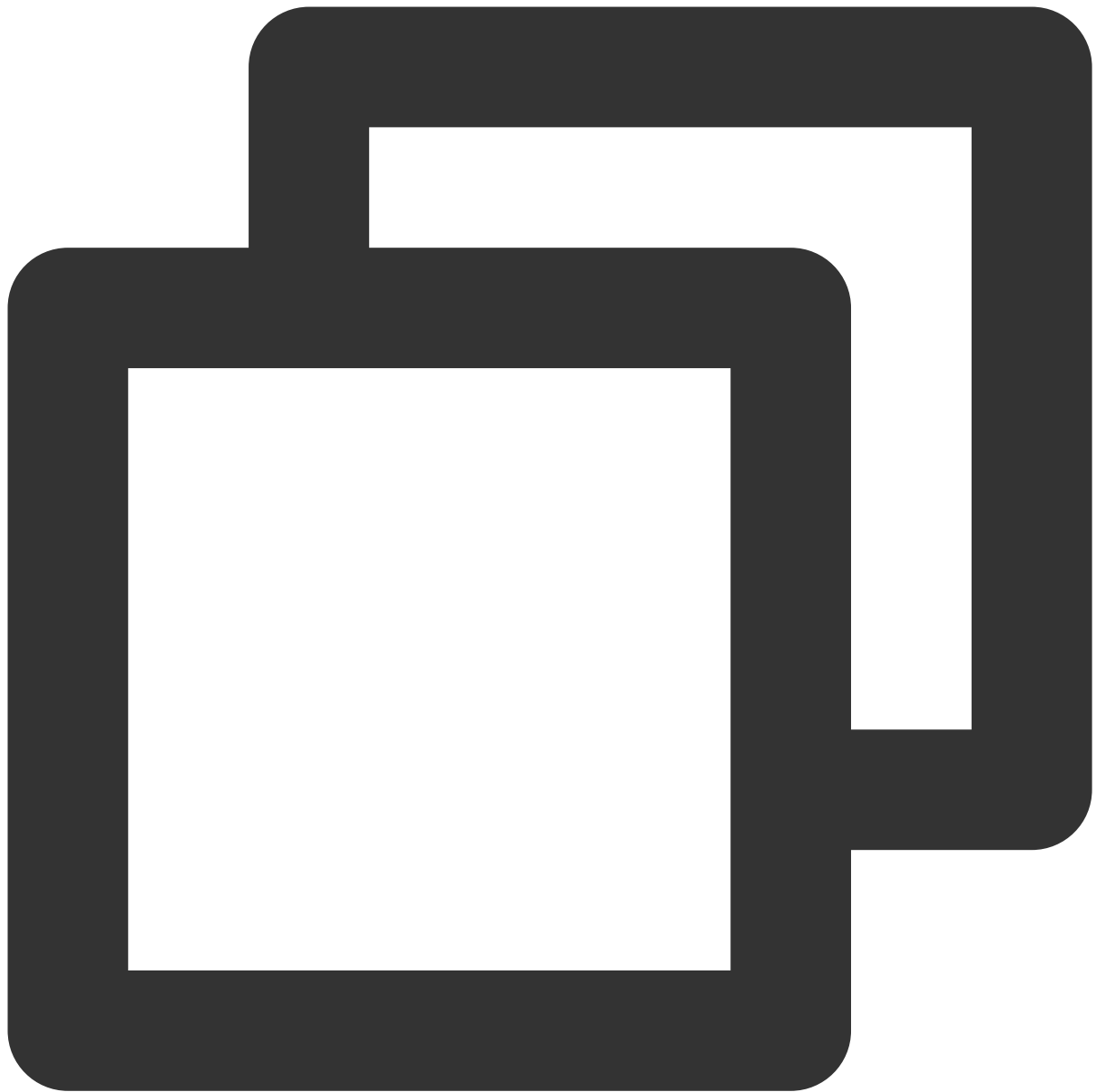
## Step 2. Import the component

Use CocoaPods to import the component as follows:

1. Add the following dependency to your `Podfile` .

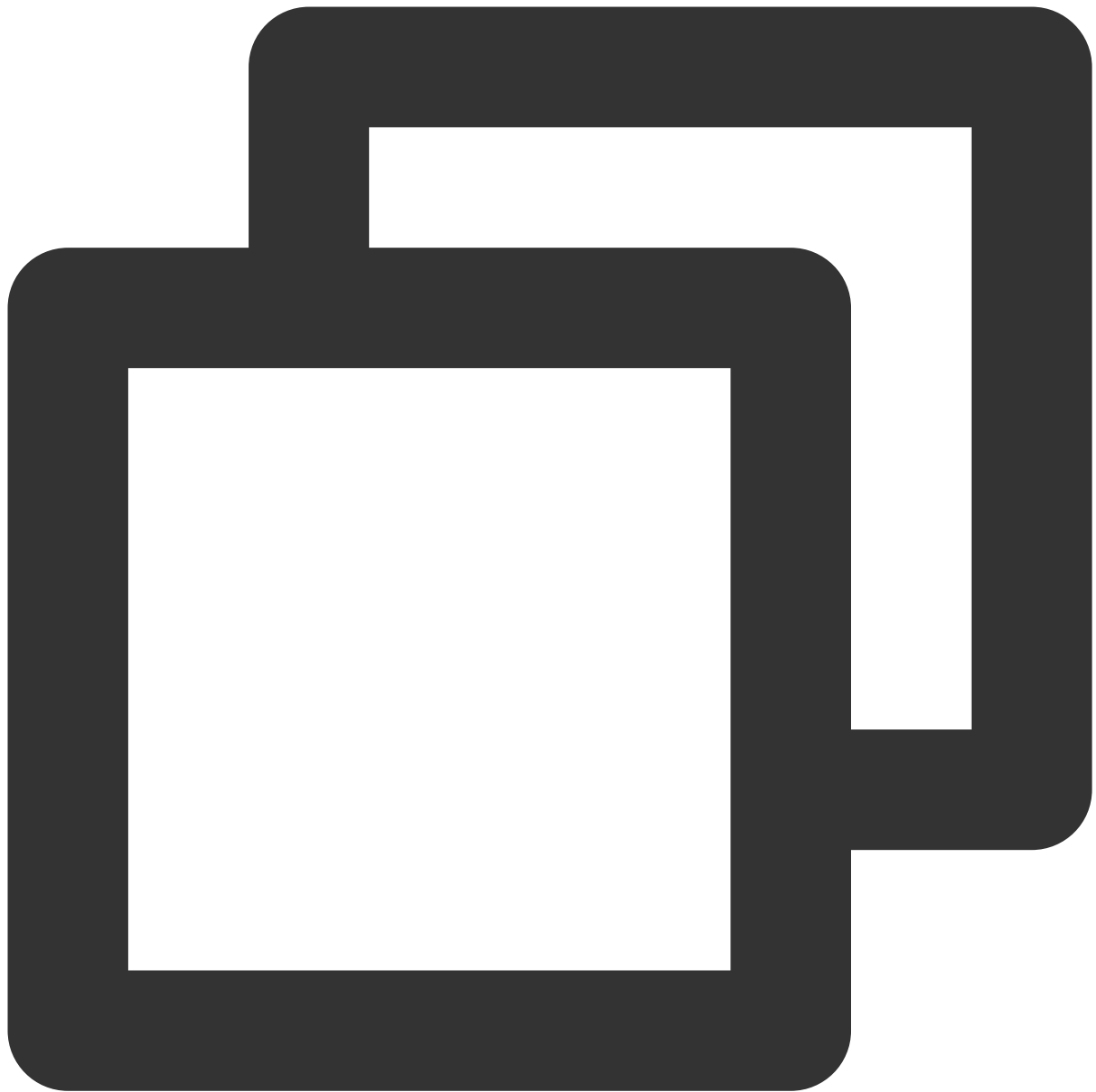
Swift

Objective-C



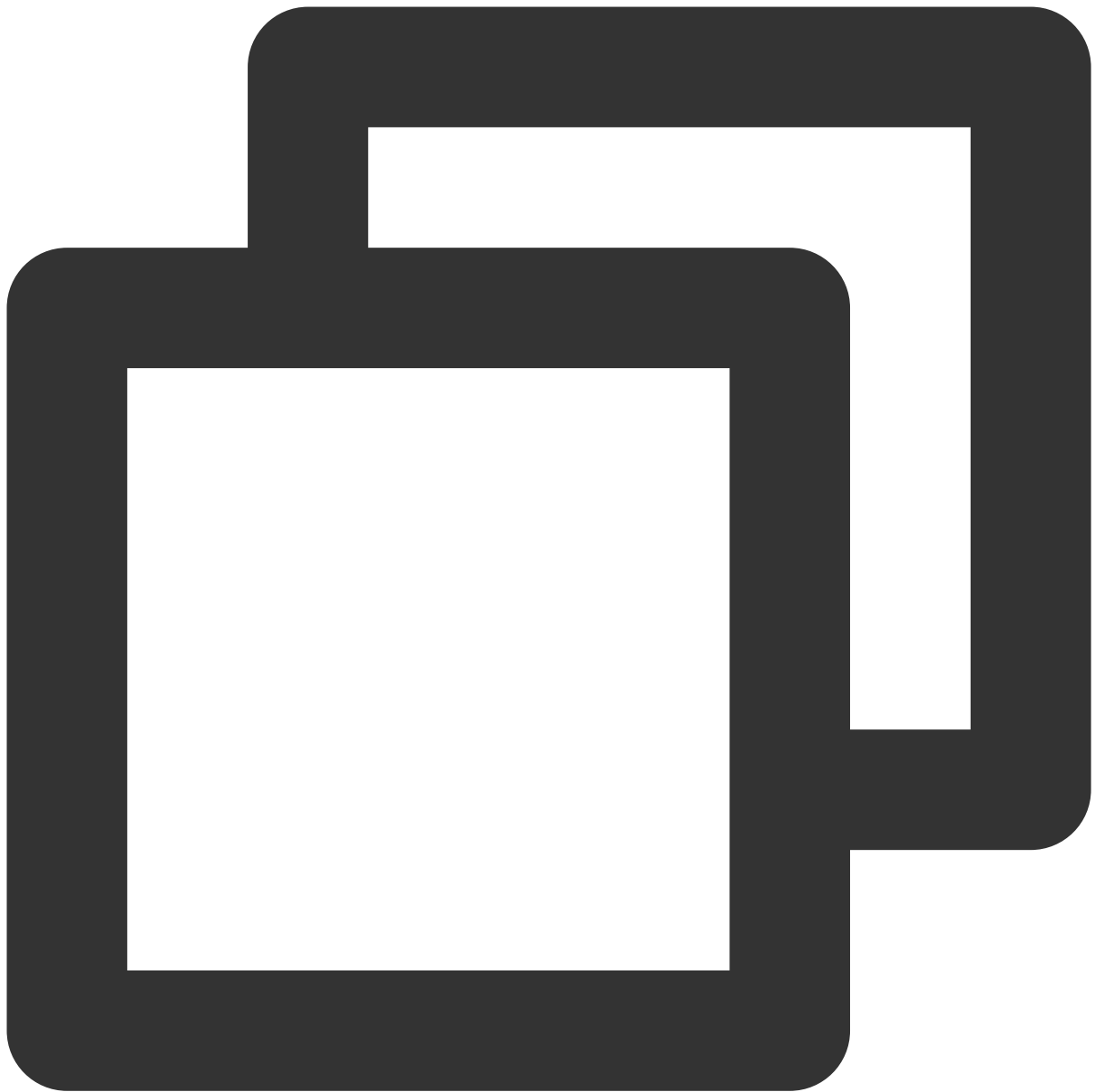
```
pod 'TUICallKit_Swift'
```





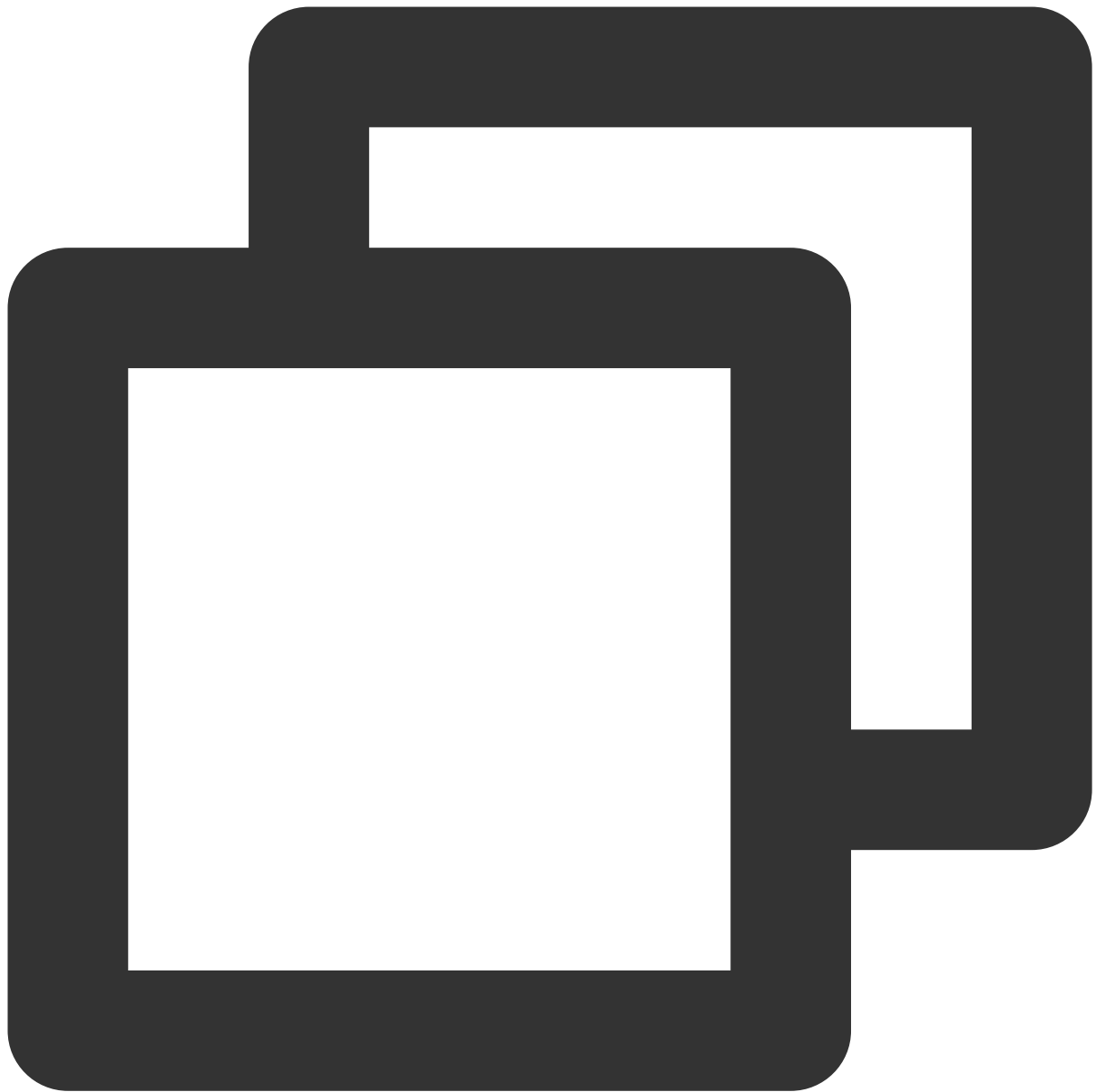
```
pod 'TUICallKit'
```

2. Run the following command to install the component:



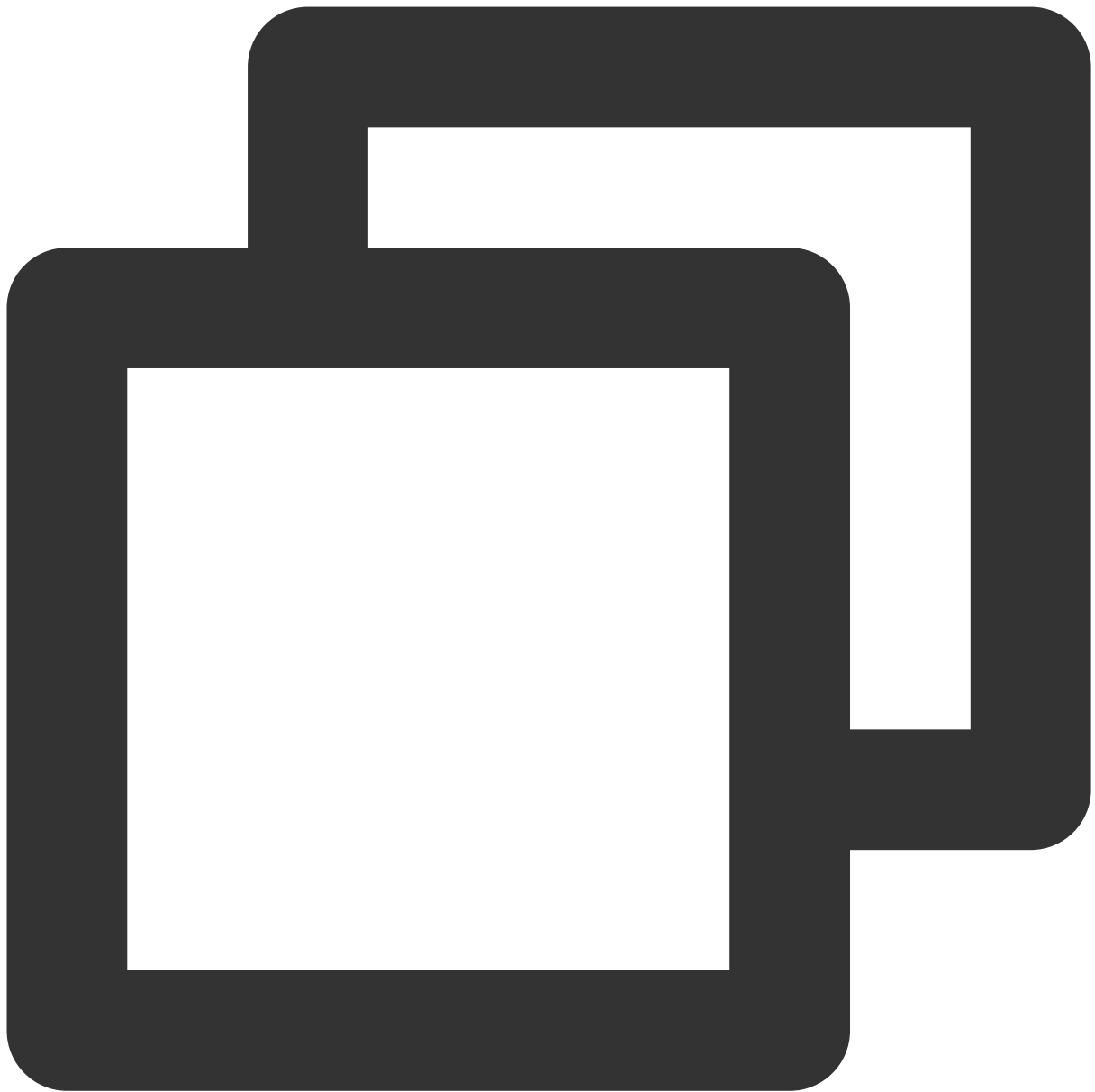
```
pod install
```

If the latest version of `TUICallKit` cannot be installed, run the following command to update the local CocoaPods repository list:



```
pod repo update
```

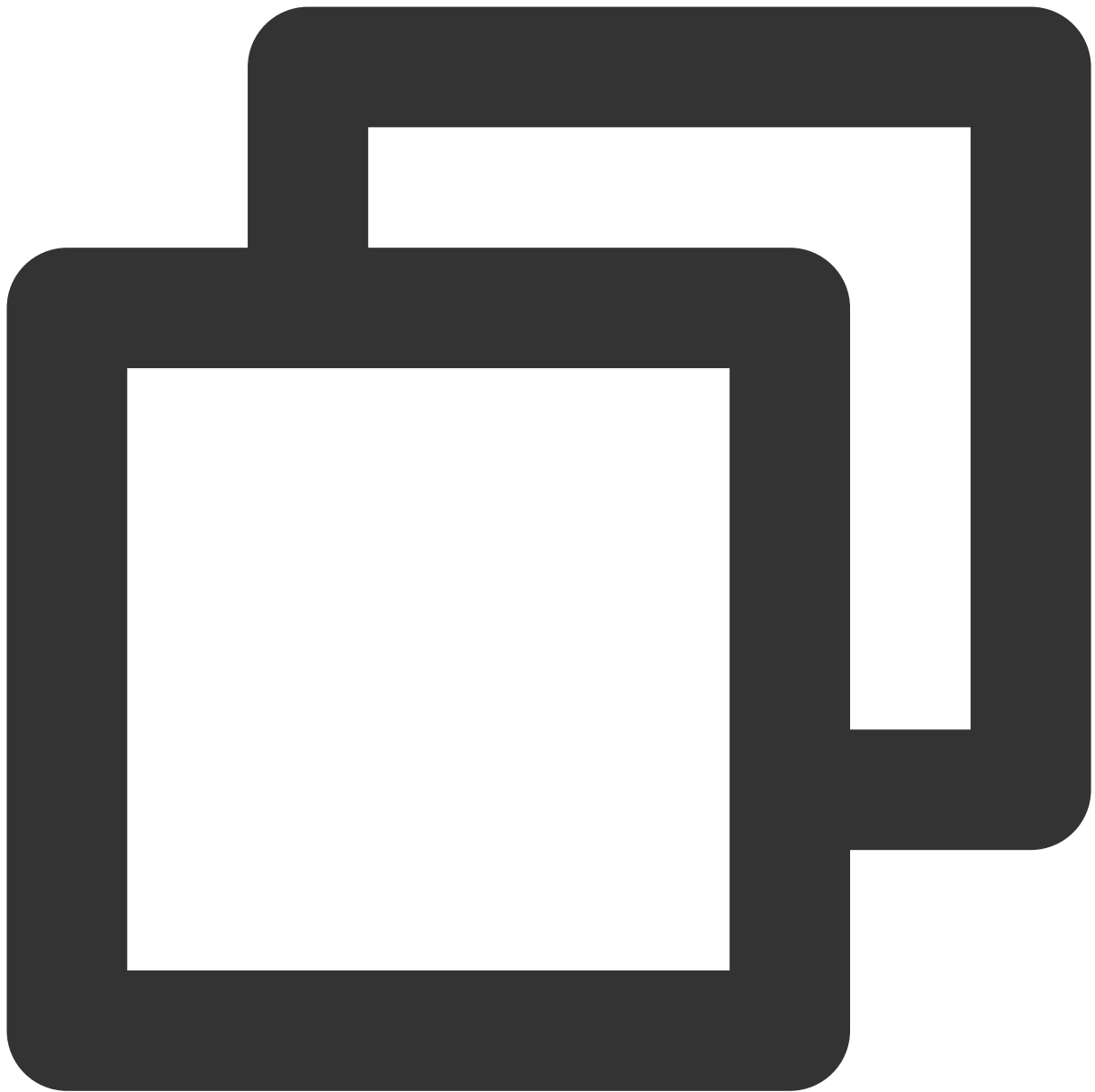
Afterwards, execute the following command to update the Pod version of the component library.



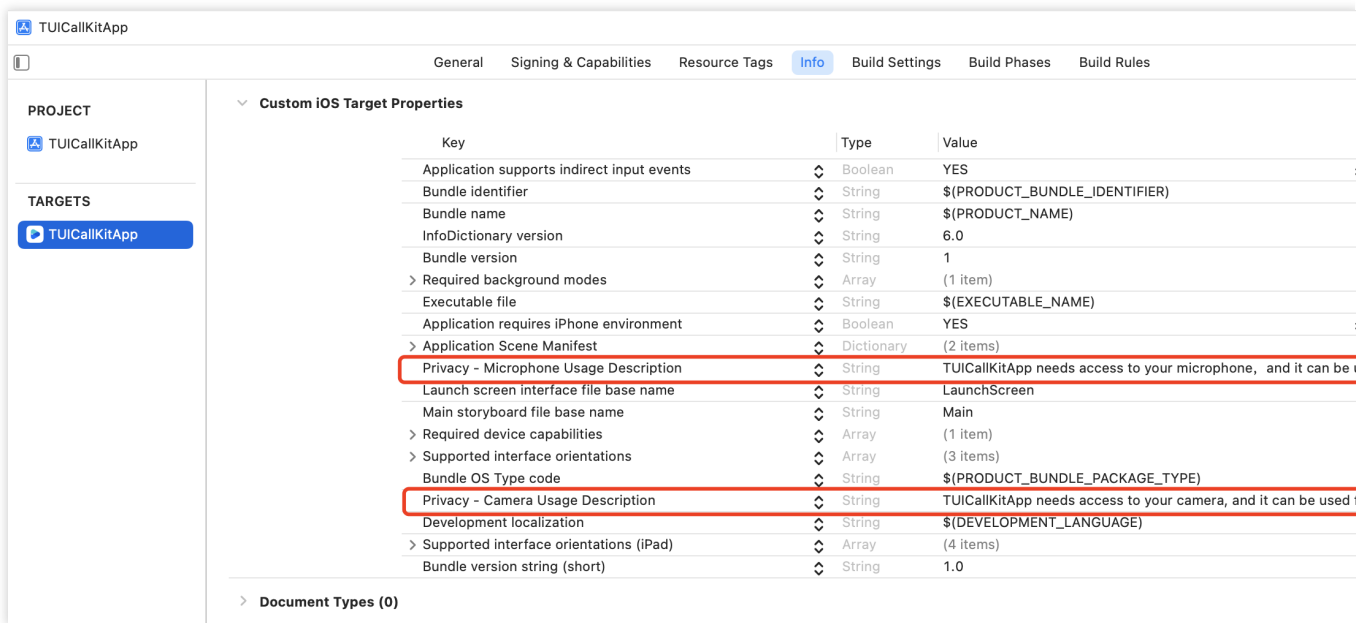
```
pod update
```

## Step 3. Configure the project

Your app needs mic and camera permissions to implement audio/video communication. Add the two items below to `Info.plist` of your app. Their content is what users see in the mic and camera access pop-up windows.



```
<key>NSCameraUsageDescription</key>  
<string>CallingApp needs to access your camera to capture video.</string>  
<key>NSMicrophoneUsageDescription</key>  
<string>CallingApp needs to access your mic to capture audio.</string>
```

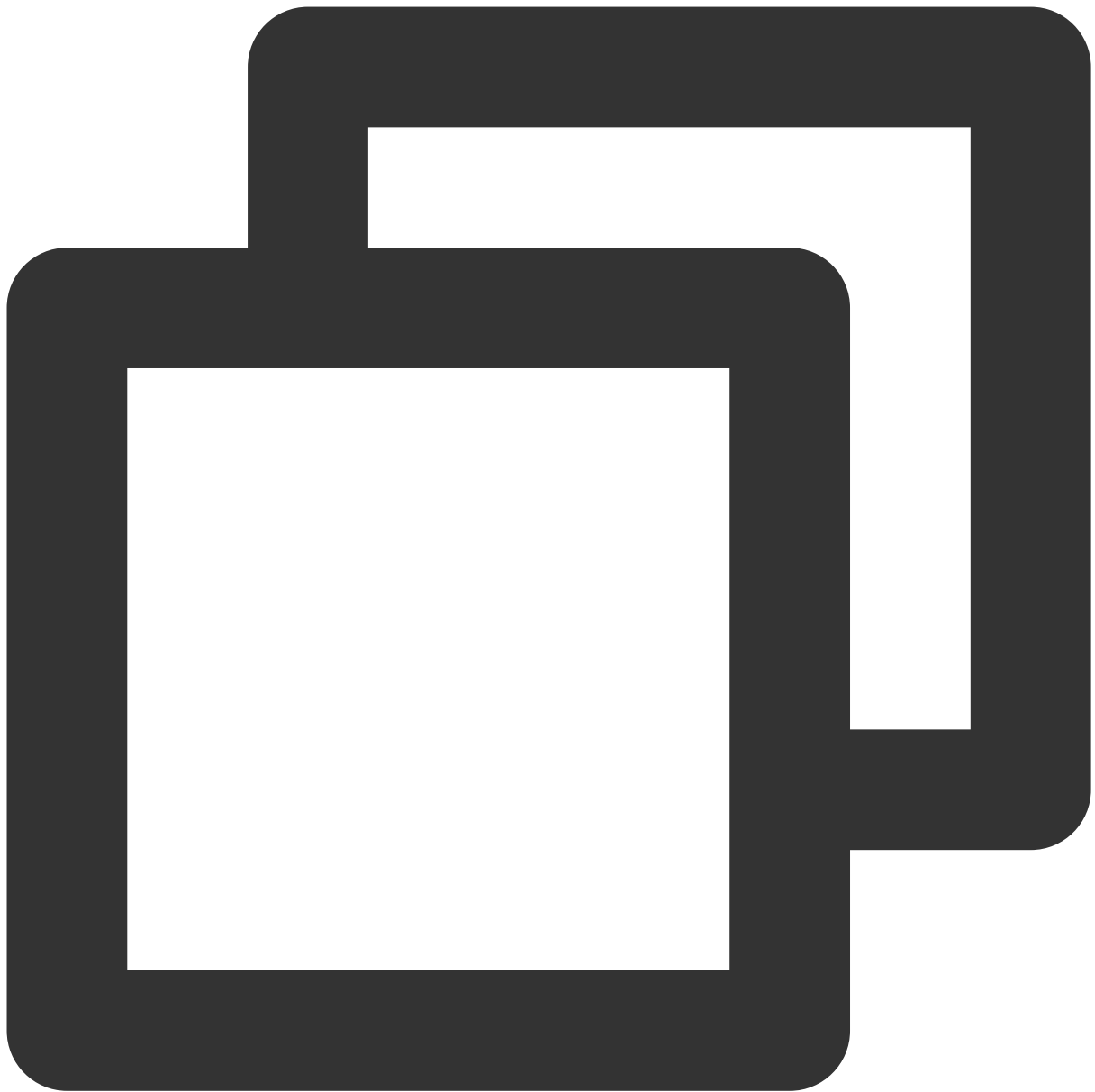


## Step 4. Log in to the TUICallKit component

Add the following code to your project to call the relevant APIs in `TUICore` to log in to the `TUICallKit` component. This step is very important, as the user can use the component features properly only after a successful login. Carefully check whether the relevant parameters are correctly configured:

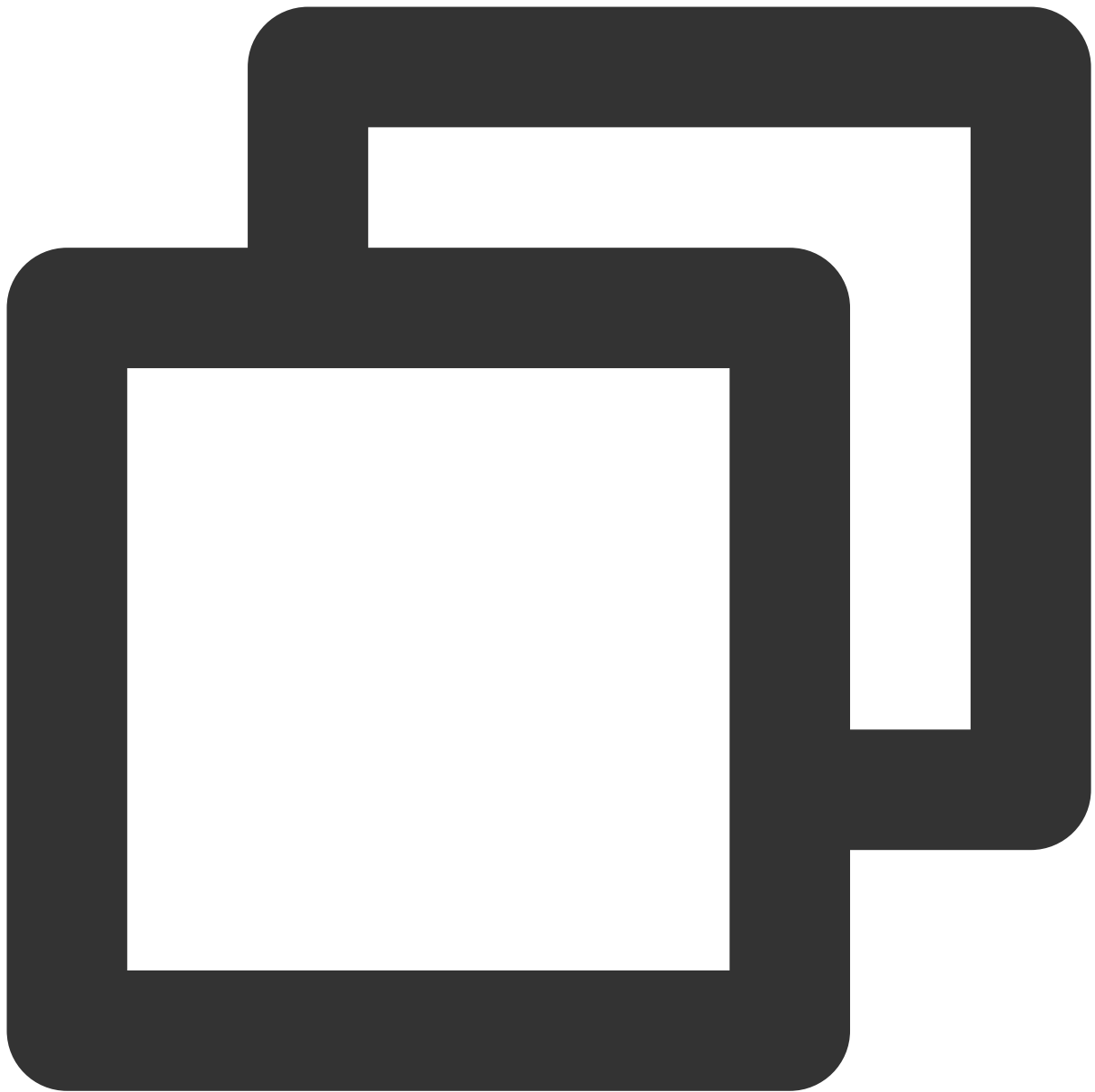
Swift

Objective-C



```
import TUICore

TUILogin.login(1400000001, // Replace it with the `SDKAppID` obtained from the Tencent Cloud console
               userID: "denny", // Replace it with your `UserID`.
               userSig: "xxxxxxxxxxx") { // You can calculate a `UserSig` in the Tencent Cloud console
    print("login success")
} fail: { (code, message) in
    print("login failed, code: \(code), error: \(message ?? "nil")")
}
```



```
#import <TUICore/TUILogin.h>

[TUILogin login:1400000001          // Replace it with the `SDKAppID` obtained in
    userID:@"denny"                  // Replace it with your `UserID`.
    userSig:@"xxxxxxxxxxxx"          // You can calculate a `UserSig` in the consol
    succ:^(
        NSLog(@"login success");
    } fail:^(int code, NSString *msg) {
        NSLog(@"login failed, code: %d, error: %@", code, msg);
    }
}
```



**Parameter description:** The key parameters used by the `login` function are as detailed below:

**SDKAppID:** Obtained in the last step in step 1 and not detailed here.

**UserID:** The ID of the current user, which is a string that can contain only letters (a–z and A–Z), digits (0–9), hyphens (-), or underscores (\_).

**UserSig:** The authentication credential used by Tencent Cloud to verify whether the current user is allowed to use the TRTC service. You can get it by using the `SDKSecretKey` to encrypt the information such as `SDKAppID` and `UserID`. You can generate a temporary `UserSig` by clicking the [UserSig Generate](#) button in the console.

The screenshot shows the 'UserSig Tools' console interface. At the top, there is a warning message: 'You haven't provided a payment method. We will suspend the service for your account after you use up your free resources. To avoid service interruption, please [complete your information](#) and [refresh](#).' Below this, there are two main sections: 'Signature (UserSig) Generator' and 'Signature (UserSig) Verifier'. The 'Generator' section includes fields for 'Application (SDKAppID)' (a dropdown menu), 'Username (UserID)' (a text input), and 'Secret key' (a text area with a note 'Auto-generated after you select an application'). There is a 'Generate' button and a 'Generate result' section with a 'Copy' button. The 'Verifier' section includes similar fields for 'Application (SDKAppID)', 'Username (UserID)', and 'Secret key'. It also has a 'UserSig' text input field and a 'Verify' button.

For more information, see [UserSig](#).

### Note

**Many developers have contacted us with many questions regarding this step. Below are some of the frequently encountered problems:**

SDKAppID is invalid.

userSig is set to the value of Secretkey mistakenly. The userSig is generated by using the SecretKey for the purpose of encrypting information such as sdkAppId, userId, and the expiration time. But the value of the userSig that is required cannot be directly substituted with the value of the SecretKey.

userId is set to a simple string such as 1, 123, or 111, and your colleague may be using the same userId while working on a project simultaneously. In this case, login will fail as TRTC doesn't support login on multiple terminals with the same UserID. Therefore, we recommend you use some distinguishable userId values during debugging.

The sample code on GitHub uses the `genTestUserSig` function to calculate `UserSig` locally, so as to help you complete the current integration process more quickly. However, this scheme exposes your `SecretKey` in the application code, which makes it difficult for you to upgrade and protect your `SecretKey` subsequently. Therefore,

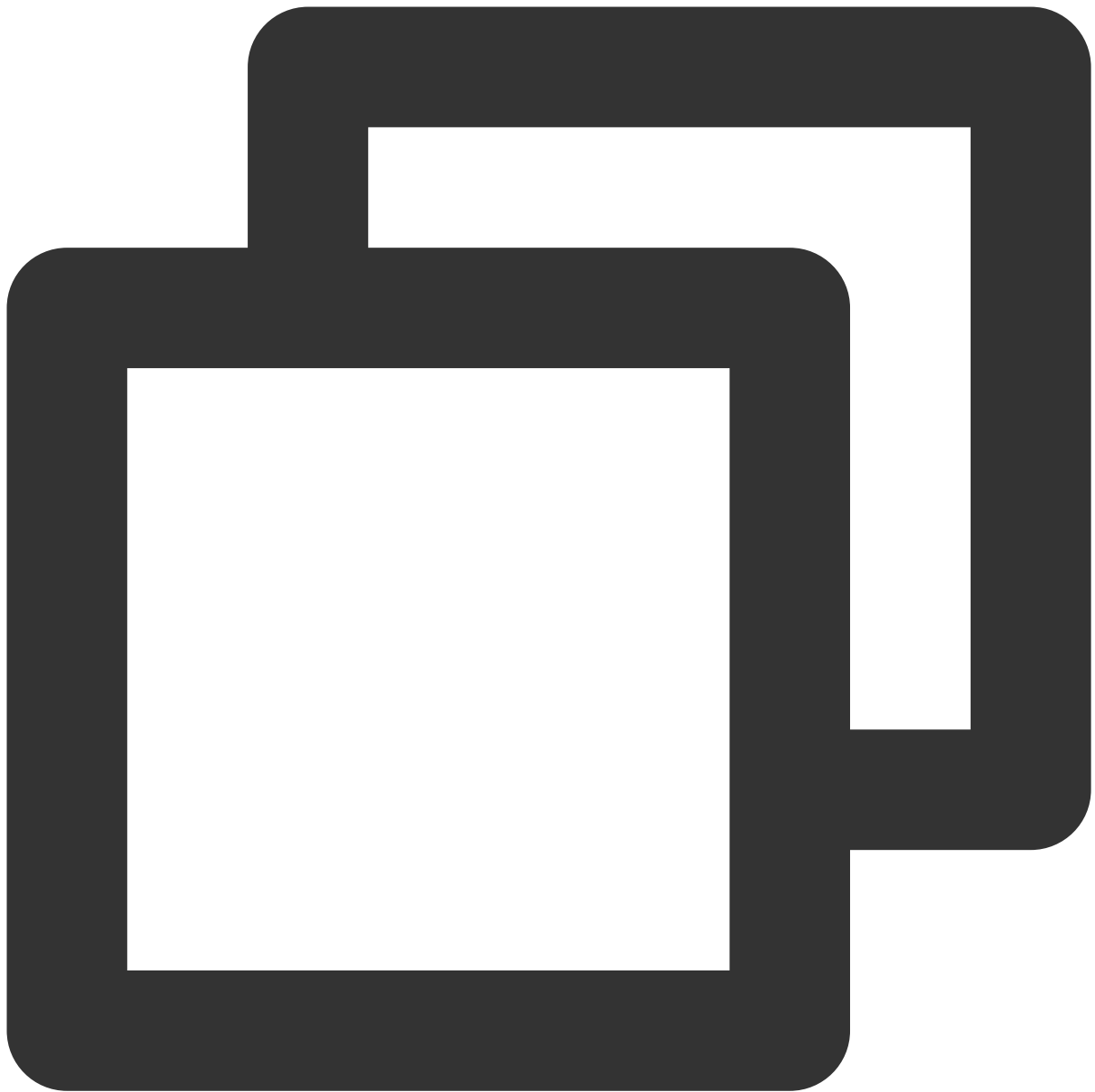
we strongly recommend you run the `UserSig` calculation logic on the server and make the application request the `UserSig` calculated in real time every time the application uses the `TUICallKit` component from the server.

## Step 5. Make your first phone call

After both the caller and callee have successfully signed in, the caller can initiate an audio or video call by calling the `TUICallKit`'s call method and specifying the call type and the callee's `userId`. At this point, the callee will receive an incoming call invitation.

Swift

Objective-C



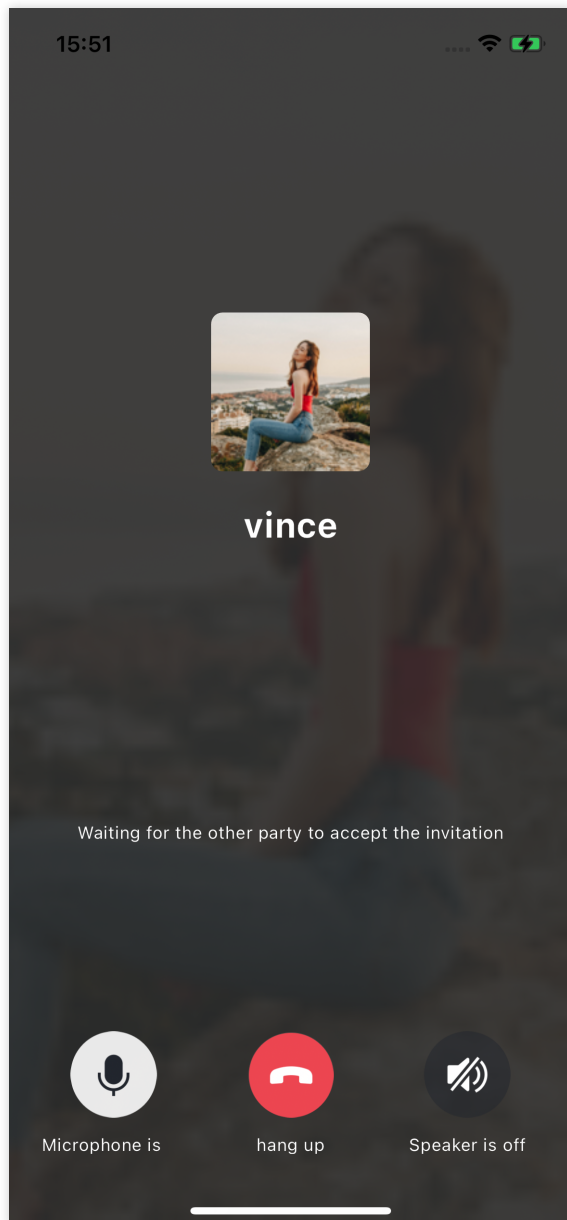
```
import TUICallKit

TUICallKit.createInstance().call(userId: "mike", callMediaType: .video)
```

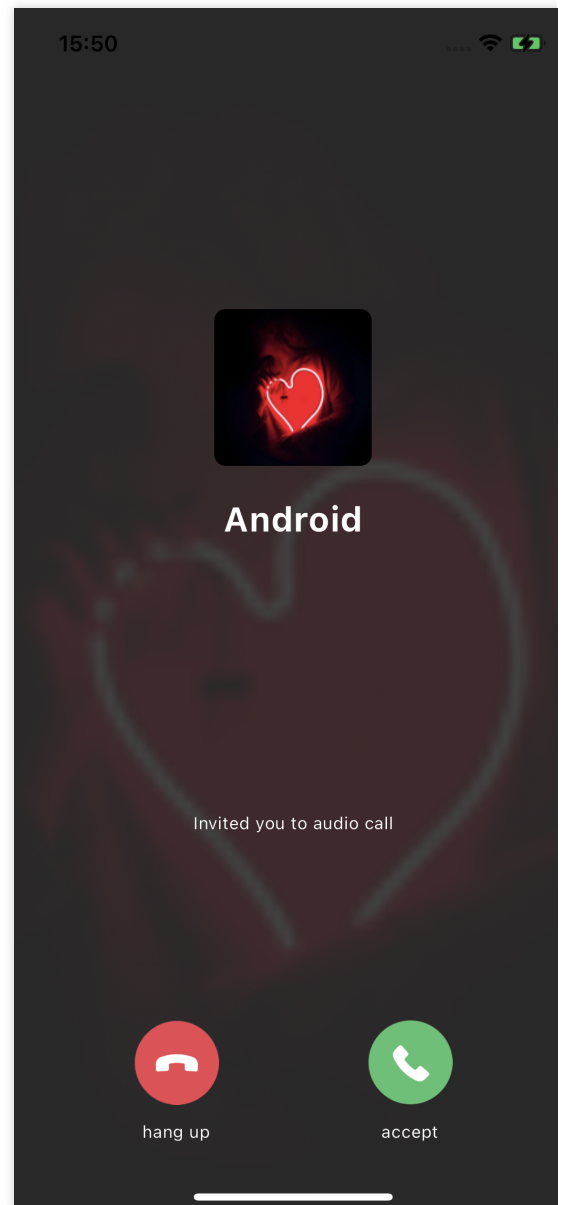


```
#import <TUICallKit/TUICallKit.h>
```

```
[[TUICallKit sharedInstance] call:@"mike" callMediaType:TUICallMediaTypeVideo];
```



Caller



Callee

## Additional Features

[Customize Interface](#)[Offline Push](#)[Group Call](#)[Floating Window](#)[Custom Ringtone](#)[Call Status Monitoring](#)[Cloud Recording](#)

## FAQs

If you encounter any issues during integration and use, please refer to [Frequently Asked Questions](#).

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

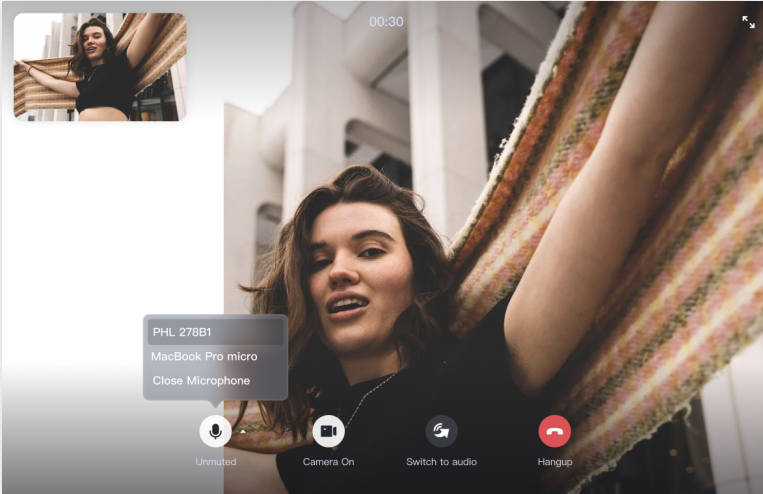
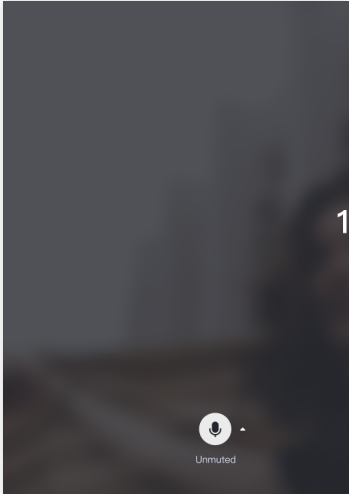
# Web&H5 (React)

Last updated : 2024-04-03 17:23:11

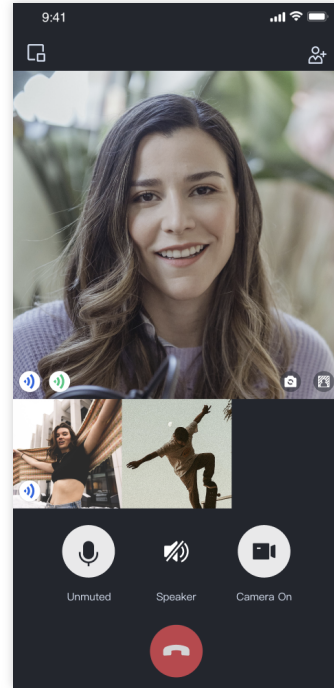
This article will guide you through the quick integration of the TUICallKit component. You will complete several key steps within 10 minutes, ultimately obtaining a video call feature with a complete UI interface.

Desktop Version

Mobile Version

Video Call	Audio Call
	

One-on-one call	Group call



## TUICallKit Demo Experience

If you want to directly try out audio and video calling, [please enter the Demo experience page](#).

If you want to quickly implement a new project, directly read [Web Demo Quick Start](#).

## Environment preparations

React  $\geq$  v18.0.

Node.js Version: Node.js  $\geq$  v12.13.0 (the official LTS version is recommended, please ensure that the npm version matches the node version).

Modern browser, supporting WebRTC APIs.

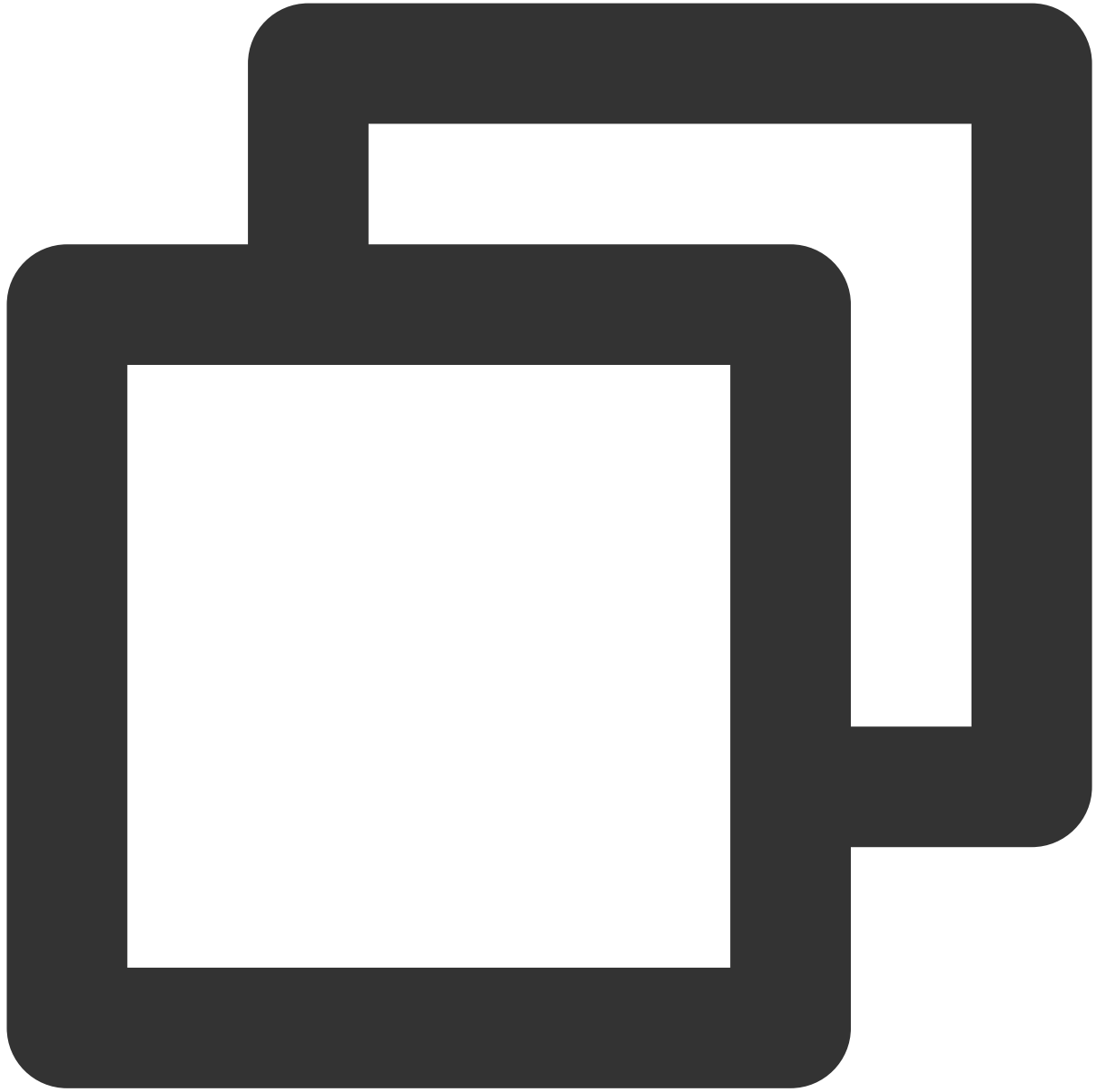
## Step 1. Activate the service

Before using the Audio and Video Services provided by Tencent Cloud, you need to go to the Console and activate the service for your application. For detailed steps, refer to [Activate Service](#).

## Step Two: Establish a React Project

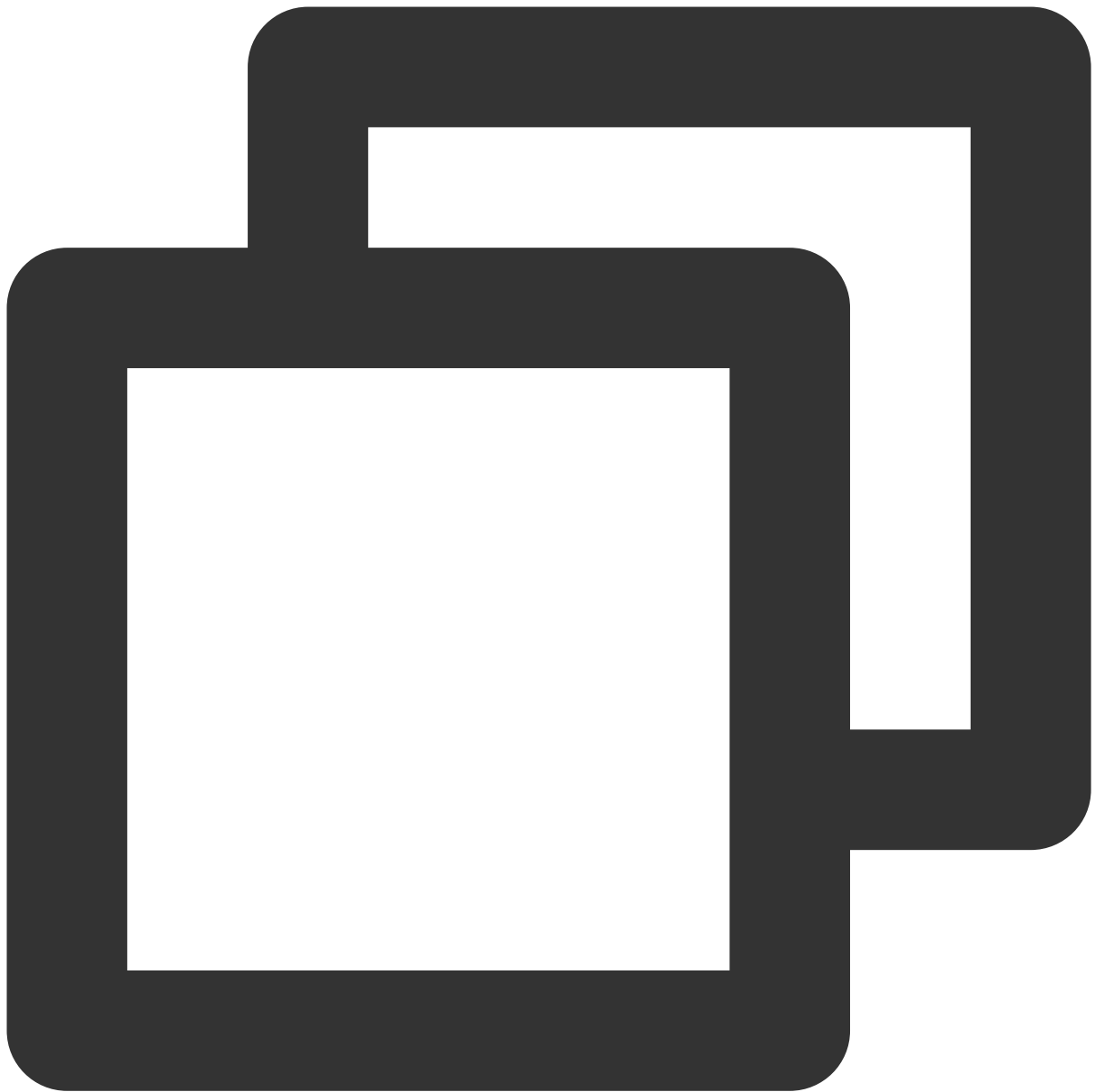


1. If you have not installed create-react-app yet, you can do so in the terminal or cmd using the following method:



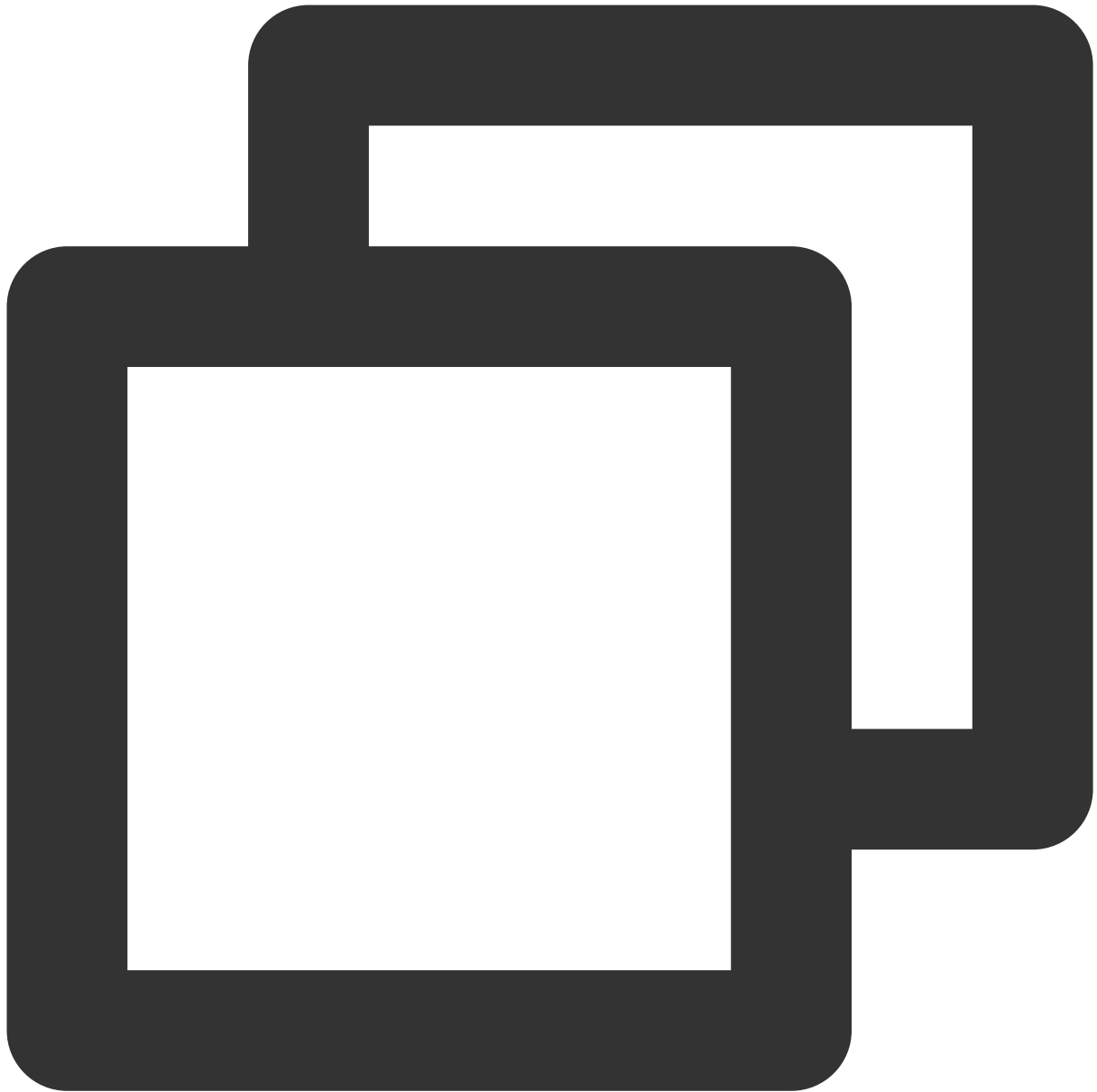
```
npm install -g create-react-app
```

2. Create a fresh React project. It is at your discretion whether to employ a 'ts' template or not.



```
npx create-react-app tuicallkit-demo --template typescript
```

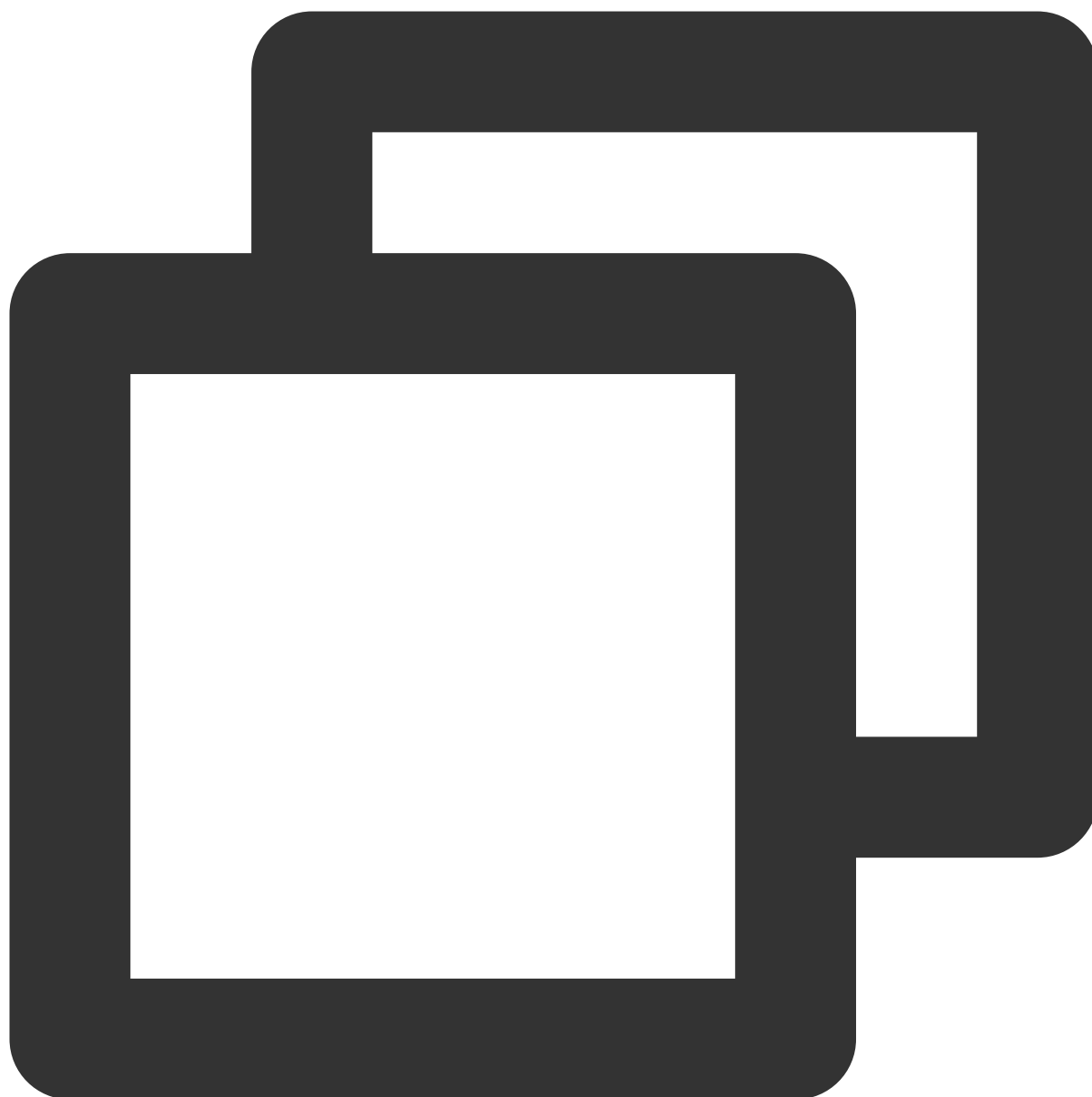
3. After the project is created, go to the project directory.



```
cd tuicallkit-demo
```

## Step Three: Download the TUICallKit Component

1. Download the [@tencentcloud/call-uikit-react](https://github.com/tencentcloud/call-uikit-react) component via npm and use it in your project.

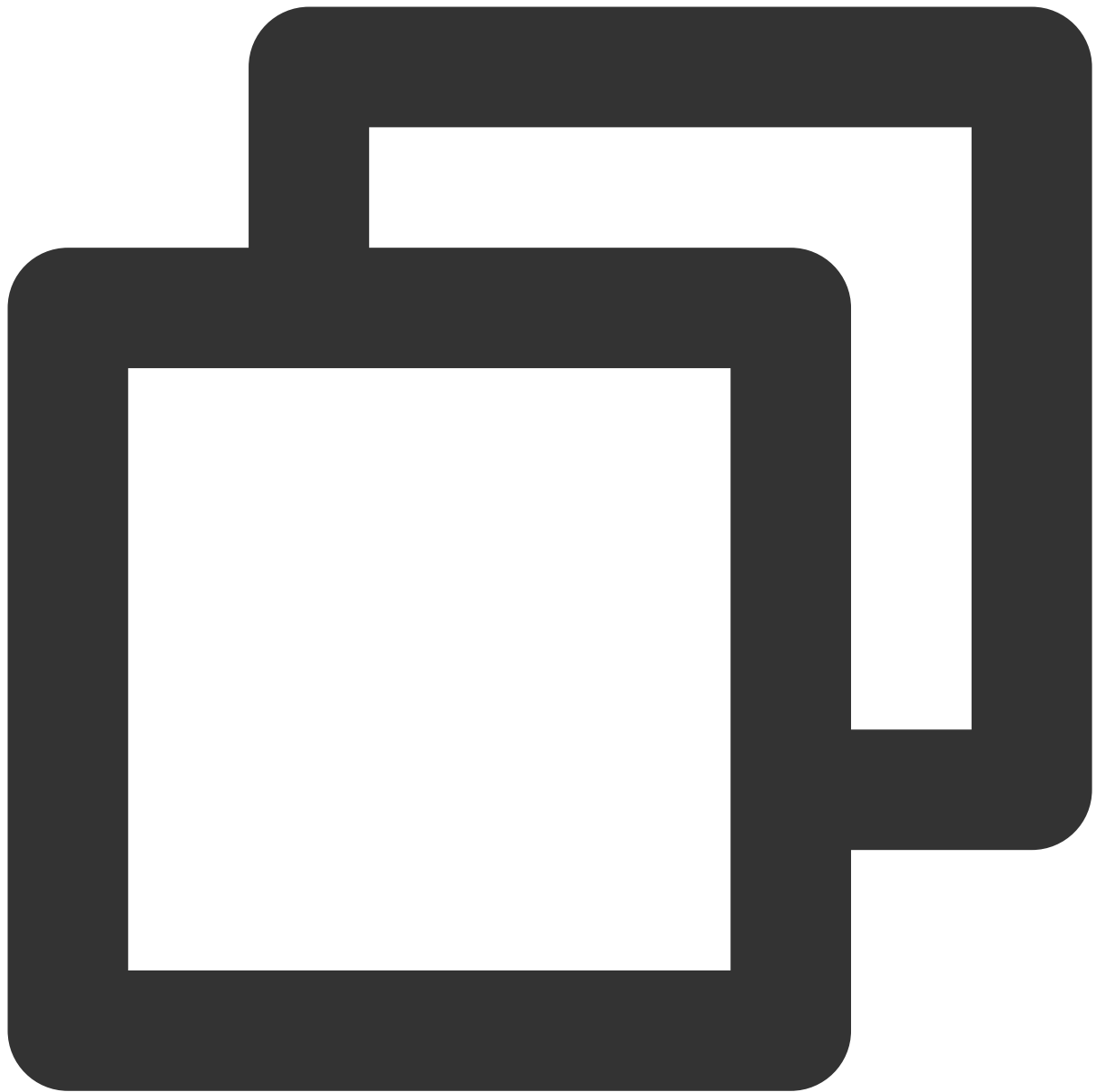


```
npm install @tencentcloud/call-uikit-react
```

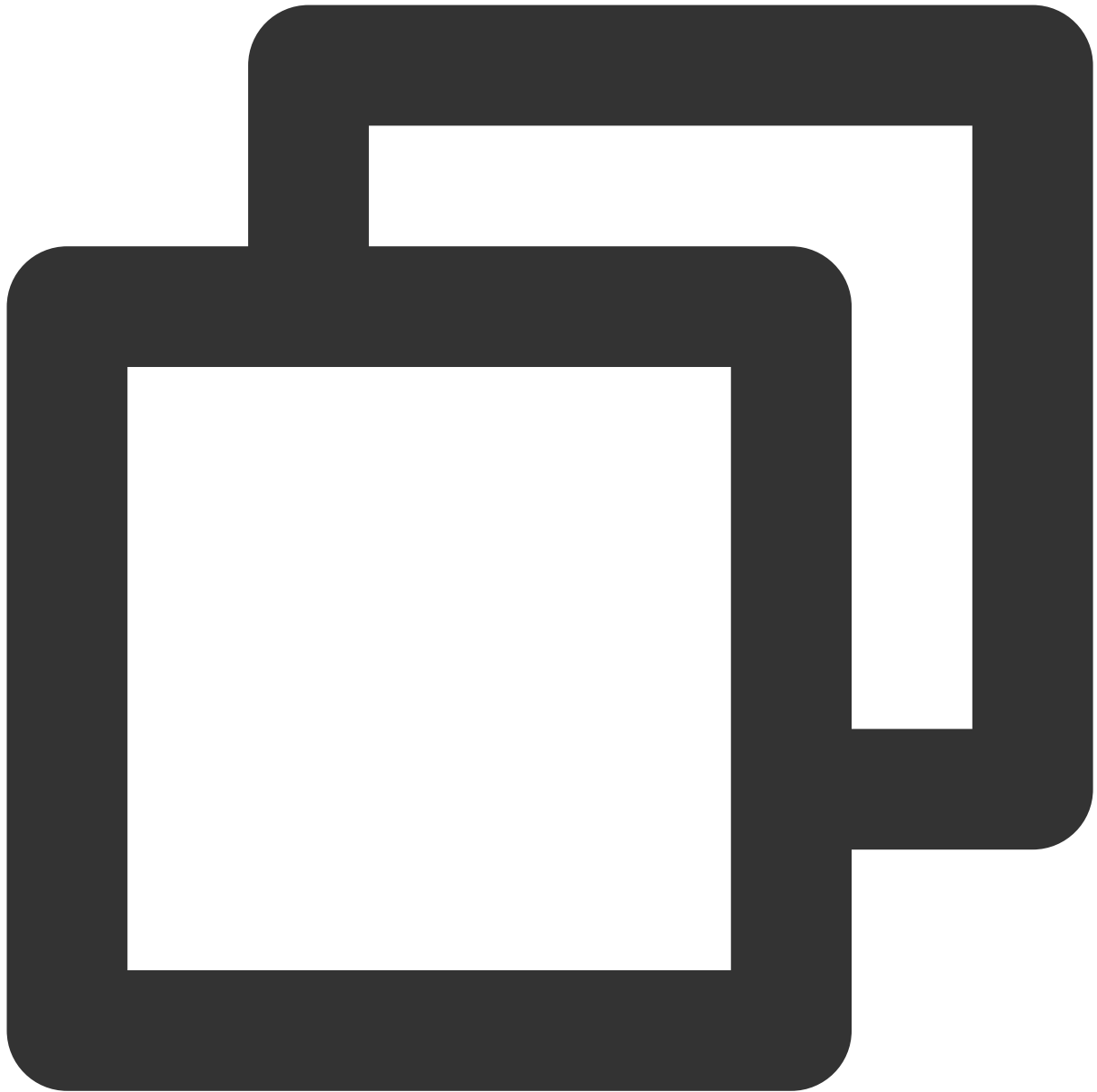
2. Copy the debug directory to your project directory, `tuicallkit-demo/src` .

MacOS

Windows



```
cp -r node_modules/@tencentcloud/call-uikit-react/debug ./src
```



```
xcopy node_modules\\@tencentcloud\\call-uikit-react\\debug .\\src\\debug /i /e
```

## Step Four: Include the TUICallKit Component

1. The following code example includes four parameters: SDKAppID, SDKSecretKey, userID, and callUserID.

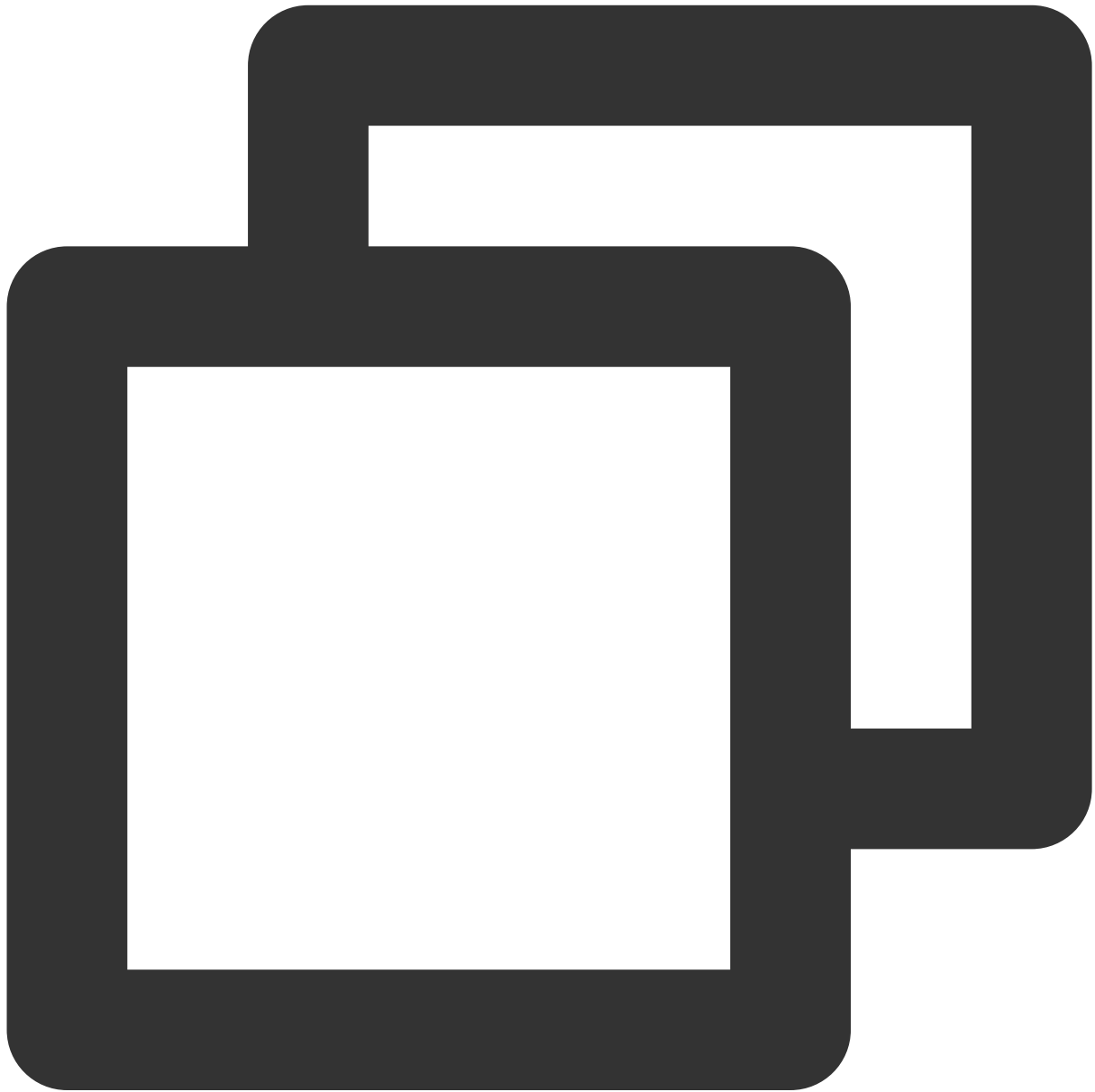
SDKAppID: The Audio and Video Application's SDKAppID created in Tencent Cloud, refer to [Activate the Service](#).

SDKSecretKey: User Signature, refer to [Activate the Service](#).

userID: The caller's userID. It's a string type that can only include English letters (a to z and A to Z), numbers (0 to 9), hyphens (-), and underscores (\_).

callUserID: The called party's userID which must already exist for initialisation. (Within the demo, input the callUserID if there is a callee, otherwise if there isn't a callee it can be left blank).

2. Incorporate the following code in `tuicallkit-demo/src/App.tsx` .



```
import React, { useState, useMemo } from "react";
// @ts-ignore
import { TUICallKit, TUICallKitServer, TUIGlobal } from "@tencentcloud/call-uikit-r
import * as GenerateTestUserSig from "../debug/GenerateTestUserSig-es";
```

```
function App() {
  const [userData, setUserData] = useState({
    userID: "",
    callUserID: "",
    SDKAppID: 0,          // Replace with your SDKAppID
    SDKSecretKey: "",    // Replace with your SDKSecretKey
    isLogin: false,
  });
  const init = async () => {
    const { SDKAppID, SDKSecretKey, userID } = userData;
    if (SDKAppID && SDKSecretKey && userID) {
      try {
        await login(SDKAppID, SDKSecretKey, userID);
        setUserData((prevState) => ({
          ...prevState,
          isLogin: true as boolean,
        }));
        alert("[TUICallKit] login success.");
      } catch (error) {
        console.error(error);
      }
    } else {
      alert("[TUICallKit] Please fill in the SDKAppID, userID and SecretKey.");
    }
  };
  const login = async (SDKAppID: any, SecretKey: any, userId: any) => {
    const { userSig } = GenerateTestUserSig.genTestUserSig({
      userID: userId,
      SDKAppID: Number(SDKAppID),
      SecretKey: SecretKey,
    });
    await TUICallKitServer.init({ //[1]Initialize the TUICallKit component
      userID: userId,
      userSig,
      SDKAppID: Number(SDKAppID),
    });
  };
  const call = async () => {
    await TUICallKitServer.call({ userID: userData.callUserID, type: 2 }); //[2]Make
  };
  const callKitStyle = useMemo(() => {
    if (TUIGlobal.isPC) {
      return { width: '960px', height: '630px', margin: '0 auto' };
    }
    return { width: '100%', height: window.innerHeight };
  }, [TUIGlobal.isPC]);
```



```

return (
  <>
    <TUICallKit style={callKitStyle}></TUICallKit>
    <div style={{ width: 350, margin: '0 auto' }}>
      <h4> {userData.isLogin ? "Call Panel" : "Login Panel"} </h4>
      <input
        value={userData.isLogin ? userData.callUserID : userData.userID}
        onChange={(value) => {
          const key = userData.isLogin ? "callUserID" : "userID";
          setUserData((prevState) => ({
            ...prevState,
            [key]: value.target.value,
          }));
        }}
        style={{ width: 230, margin: '0 auto' }}
        placeholder={
          userData.isLogin ? "Please enter callee's userID" : "Please enter your
        }
      />
      <button onClick={userData.isLogin ? call : init}> {userData.isLogin ? "call
      <p style={{margin: '10'}}> {userData.isLogin ? `Your userID: ${userData.us
    </div>
  </>
);
}

export default App;

```

## Step Five: Make your first phone call

1. Execute `npm run start` in the terminal to run tuicallkit-demo.

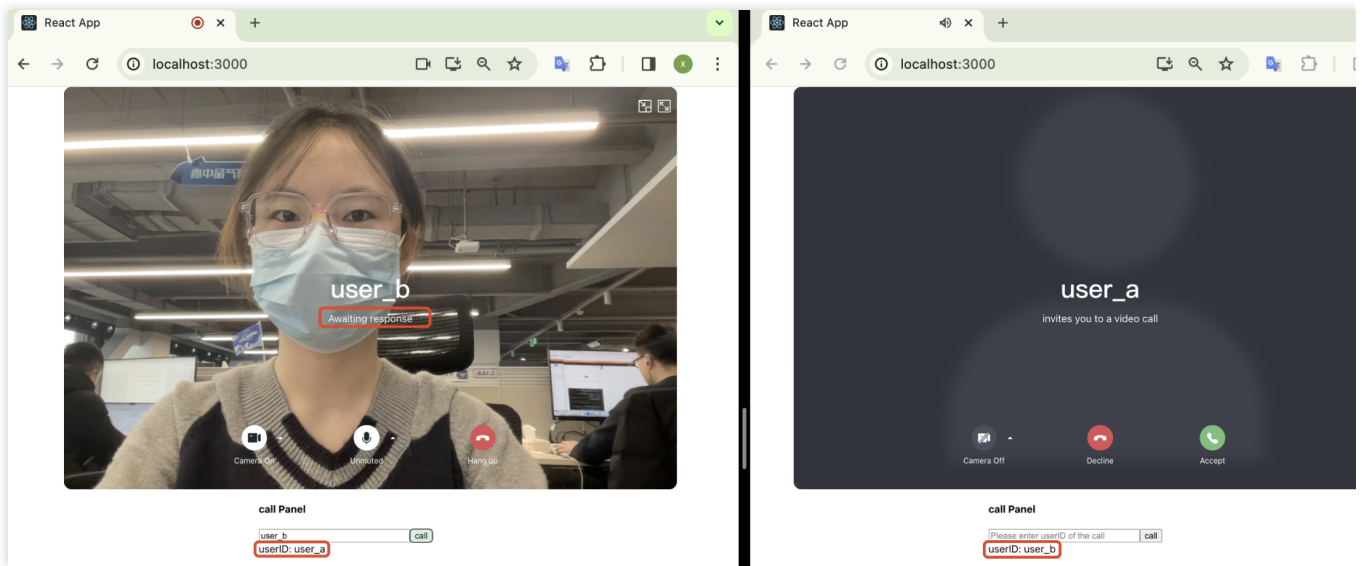
**Alert:** Visit locally through localhost protocol, for details refer to [Network access protocol description](#).

2. Enter the userID for login, click the login button, 'login success' pop-up indicates successful initialization.

The screenshot shows a web application interface with three main sections:

- Login Panel:** Contains a text input field with the placeholder "Please enter your userID" and a "login" button.
- Message Box:** A green-bordered box in the center displays "localhost:3000 显示" and "[TUICallKit] login success." with a green "确定" (Confirm) button.
- Call Panel:** Contains a text input field with the placeholder "Please enter callee's userID" and a "call" button. Below this, it shows "Your userID: user\_a".

3. Enter the recipient's userID, click the call button, initiate a 1v1 video call.



## Additional Features

[Setting Nickname, Avatar](#)

[Group Call](#)

[Floating Window](#)

[Custom Ringtone](#)

[Call Status Monitoring, Component Callback Event](#)

[Setting Resolution, Fill Pattern](#)

[Customize Interface](#)

## FAQs

If you encounter any issues during integration and use, please refer to [Frequently Asked Questions](#).

## Technical Consultation

If you have any requirements or feedback, you can contact: [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

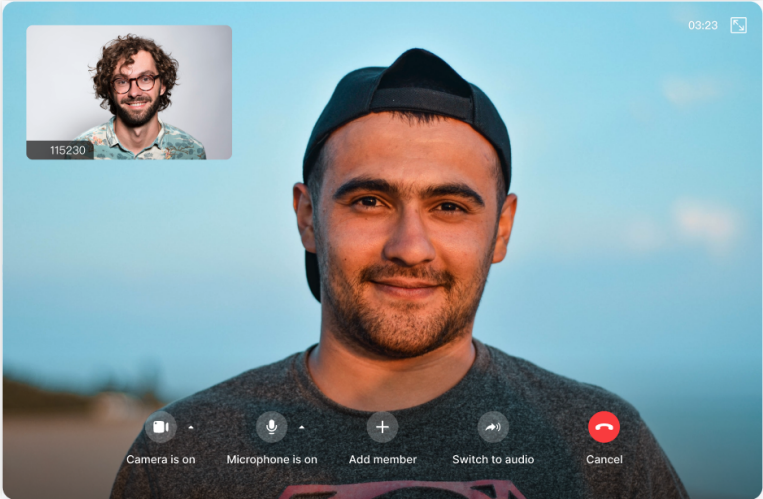
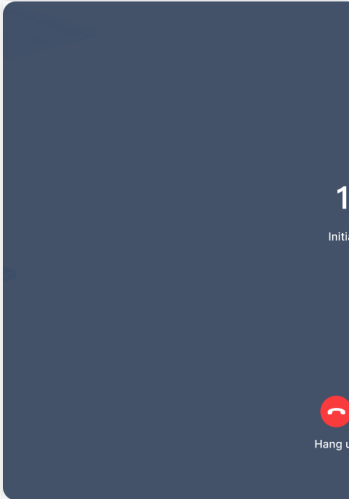
# Web&H5 (Vue3)

Last updated : 2024-04-03 17:23:11

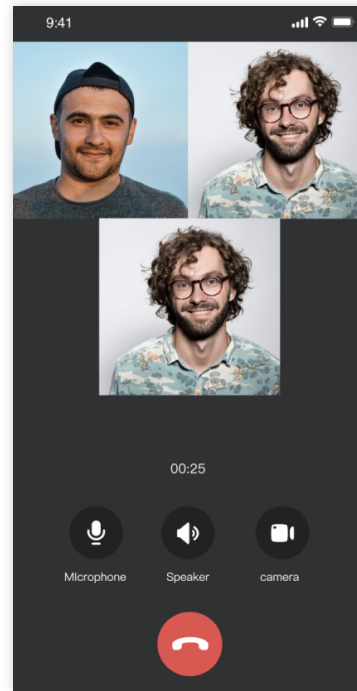
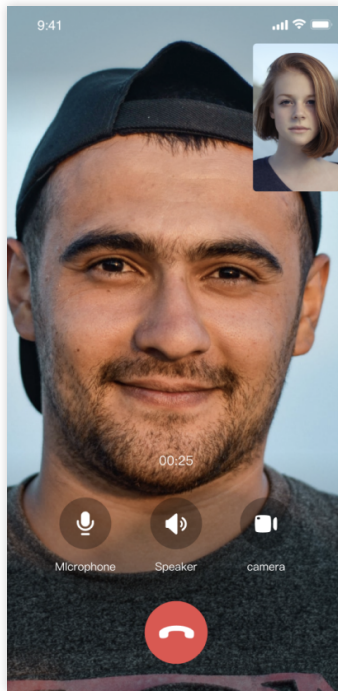
This document elucidates the rapid integration of the TUICallKit component. Within a mere span of ten minutes, you will accomplish several pivotal steps, culminating in a comprehensive User Interface with video call functionality.

Desktop Version

Mobile Version

Video Call	Audio Call
	

One-to-One Call	Group call



## TUICallKit Demo Experience

If you want to directly try out audio and video calling, [please enter the Demo experience page](#).

If you want to quickly implement a new project, directly read [Web Demo Quick Start](#).

## Environment preparations

Node.js Version: Node.js  $\geq$  v12.13.0 (the official LTS version is recommended, please ensure that the npm version matches the node version).

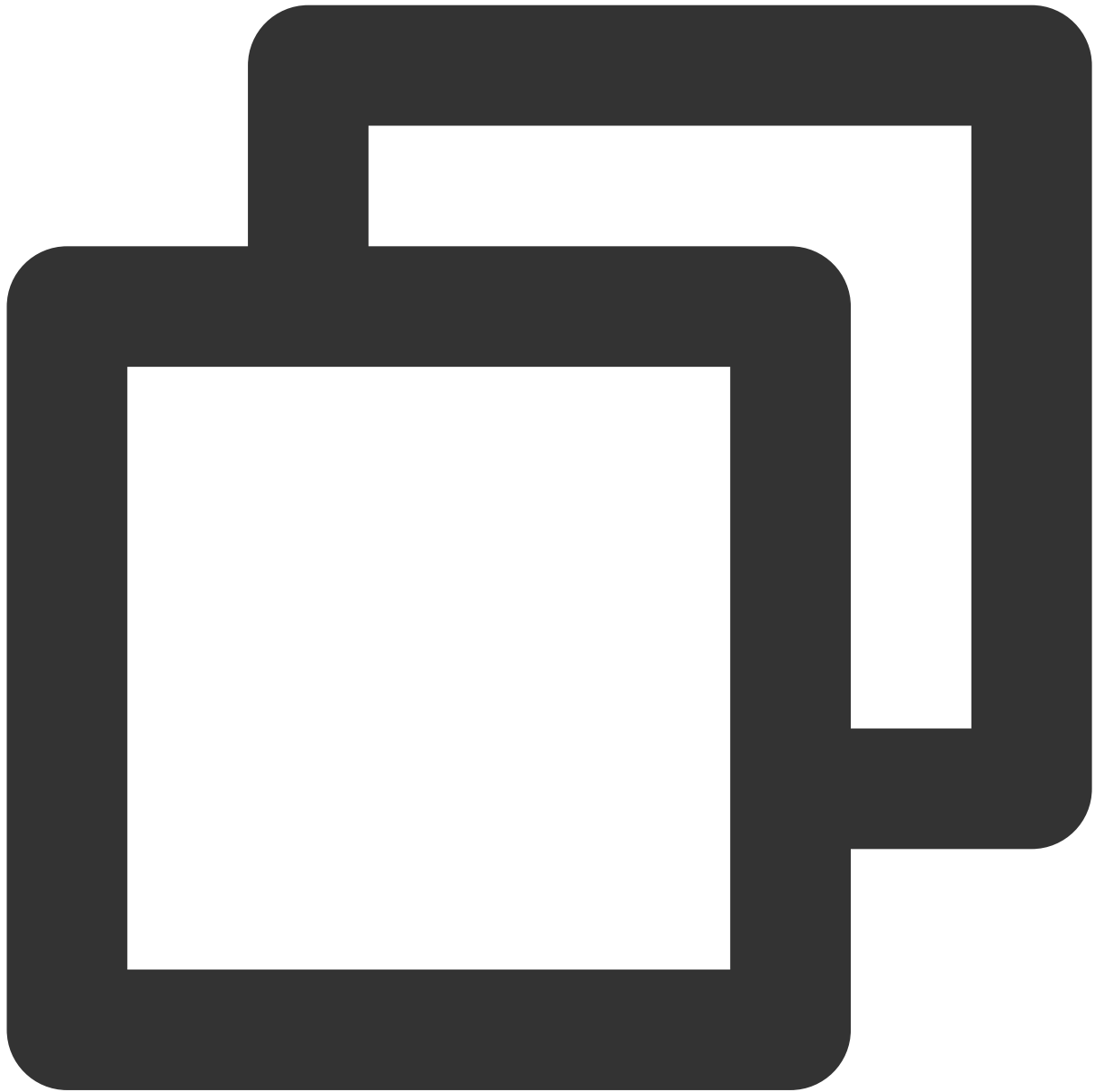
Modern browser, supporting WebRTC APIs.

## Step 1. Activate the service

Before using the Audio and Video Services provided by Tencent Cloud, you need to go to the Console and activate the service for your application. For detailed steps, refer to [Activate Service](#).

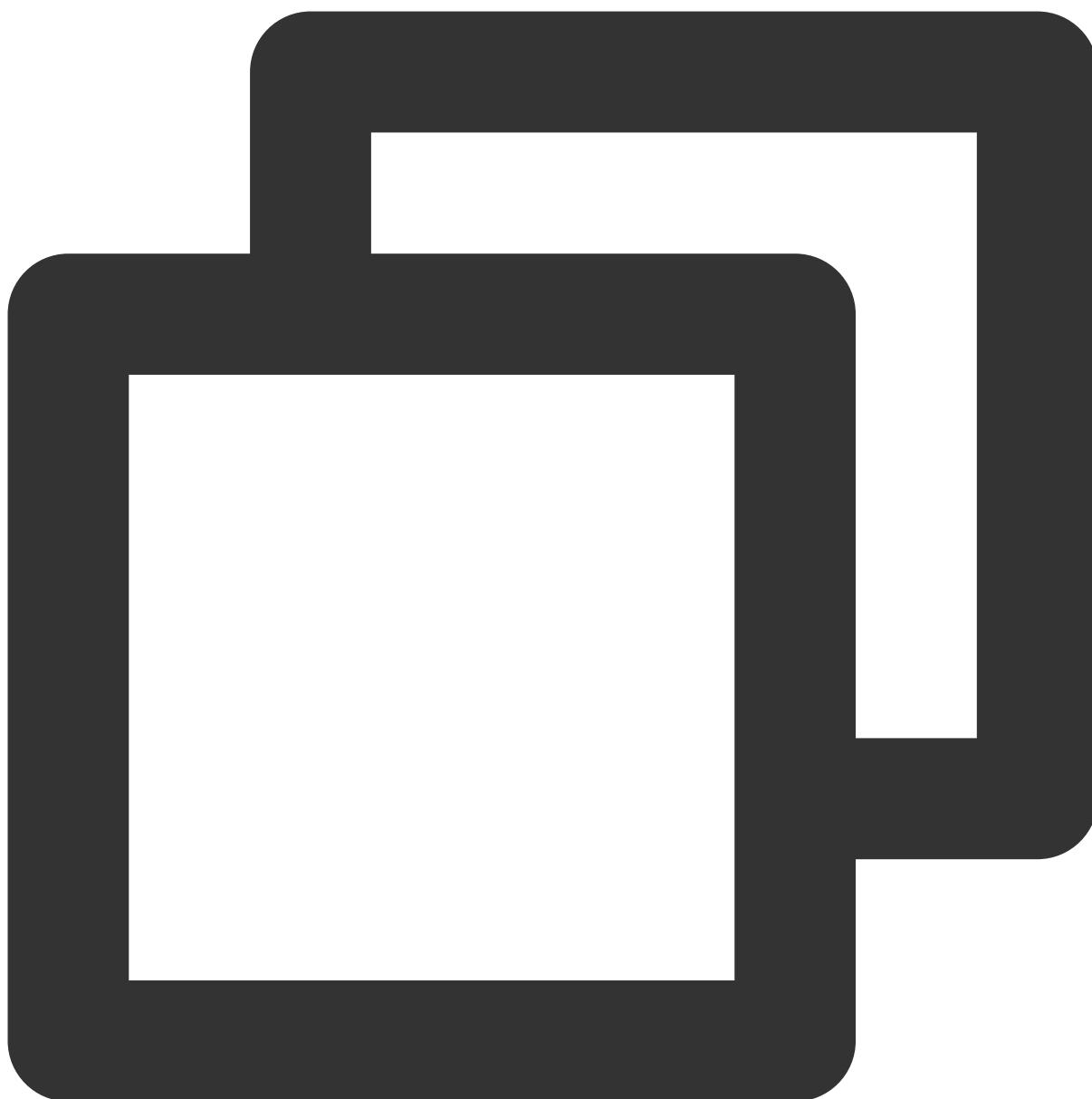
## Step Two: Create a Vue3 Project

1. If you haven't installed Vue CLI yet, you can install it in the terminal or cmd as follows:



```
npm install -g @vue/cli
```

2. Create a project through Vue CLI and select the configuration items depicted below.



```
vue create tuicallkit-demo
```

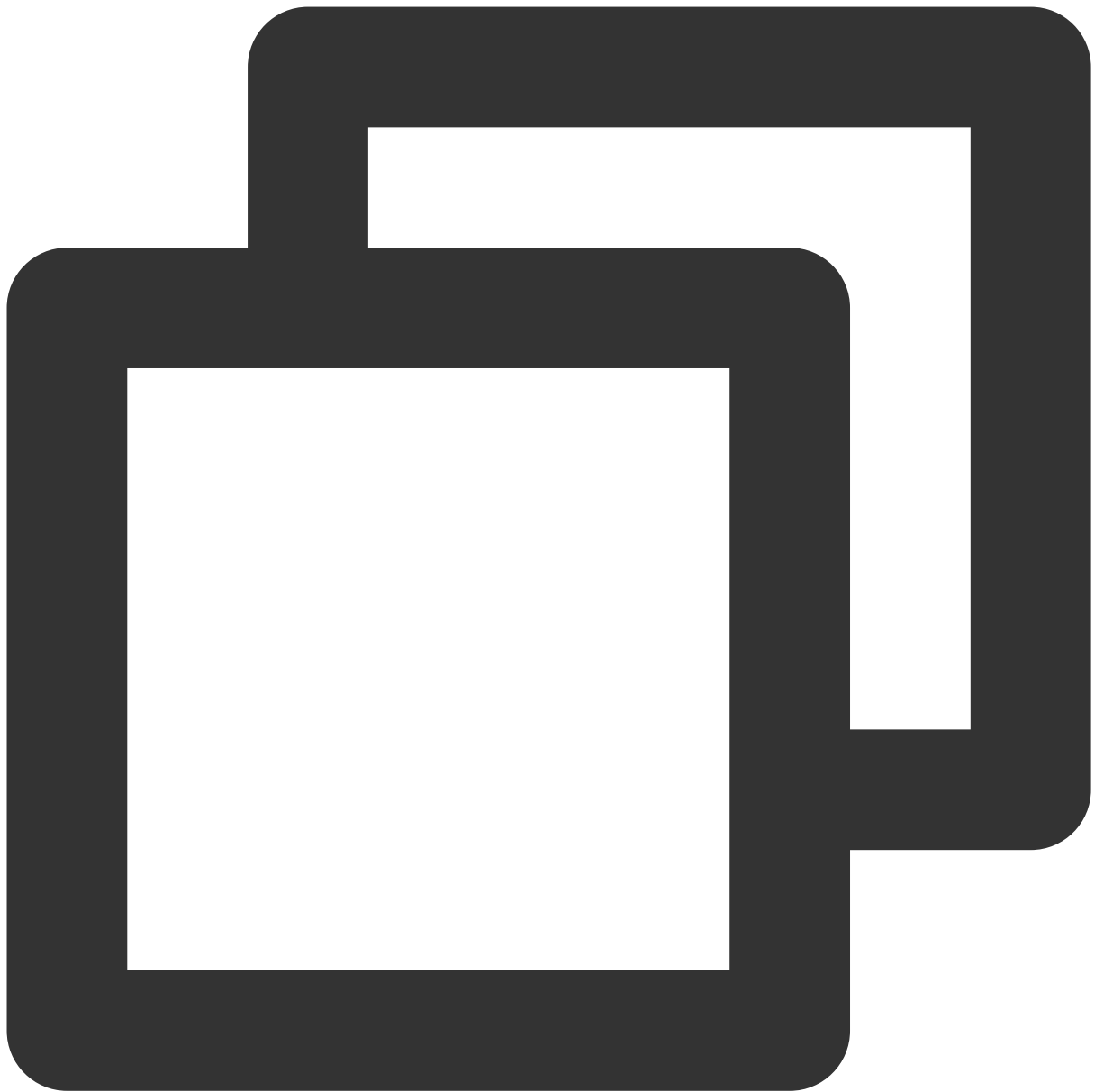
```
Vue CLI v4.5.0

New version available 4.5.0 → 5.0.8
Run npm i -g @vue/cli to update!

? Please pick a preset:
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
  Manually select features
? Check the features needed for your project:
  ● Choose Vue version
  ● Babel
  ● TypeScript
  ○ Progressive Web App (PWA) Support
  ○ Router
  ○ Vuex
  ● CSS Pre-processors
  ● Linter / Formatter
  ○ Unit Testing
  ○ E2E Testing

? Choose a version of Vue.js that you want to start the project with
  2.x
  3.x (Preview)
? Use class-style component syntax? (Y/n) n
? Use Babel alongside TypeScript (required for modern mode,
auto-detected polyfills, transpiling JSX)? (Y/n) Y
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are
supported by default):
  Sass/SCSS (with dart-sass)
  Less
  Stylus
? Pick a linter / formatter config: (Use arrow keys)
  > ESLint with error prevention only
    ESLint + Airbnb config
    ESLint + Standard config
    ESLint + Prettier
? Pick additional lint features: (Press <space> to select, <a> to
toggle all, <i> to invert selection, and <enter> to proceed)
  > ● Lint on save
    ○ Lint and fix on commit
? Where do you prefer placing config for Babel, ESLint, etc.? (Use
arrow keys)
  > In dedicated config files
    In package.json
? Save this as a preset for future projects? (y/N) N
```

3. After completing the project creation, switch to the project directory.

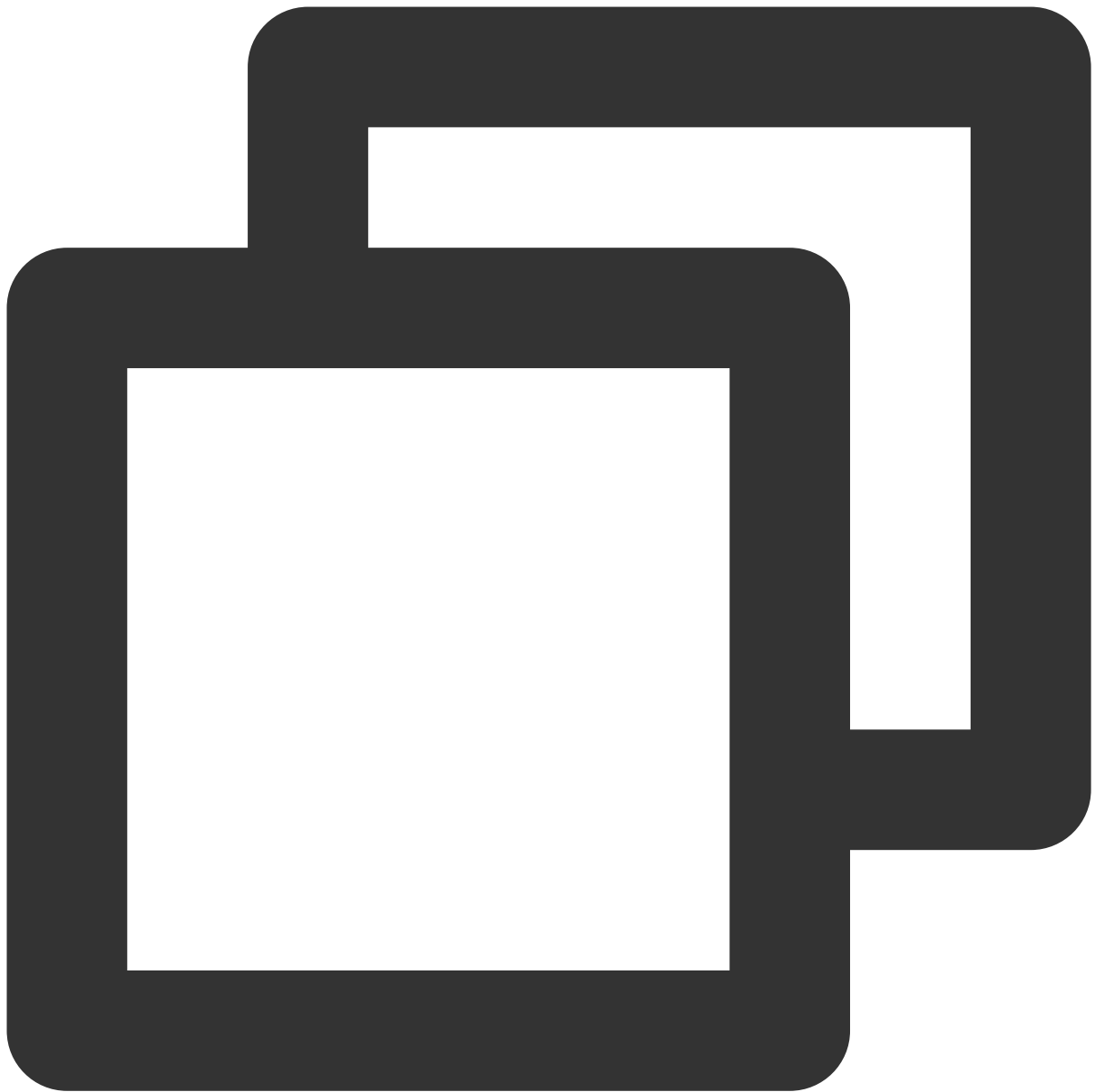


```
cd tuicallkit-demo
```

## Step Three: Download the TUICallKit Component

1. Download the [@tencentcloud/call-uikit-vue](https://github.com/tencentcloud/call-uikit-vue) component via npm and use it in your project.



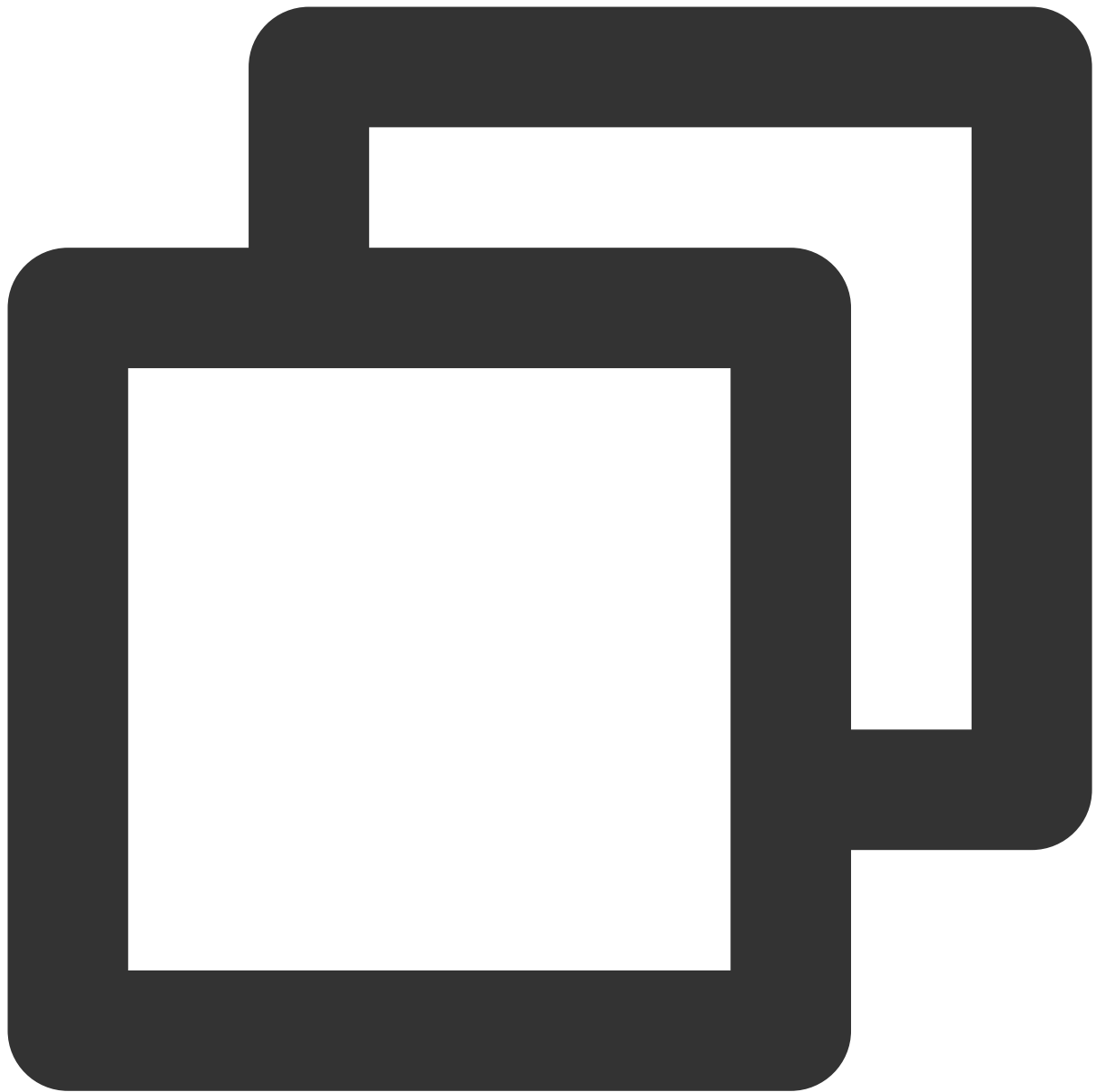


```
npm install @tencentcloud/call-uikit-vue
```

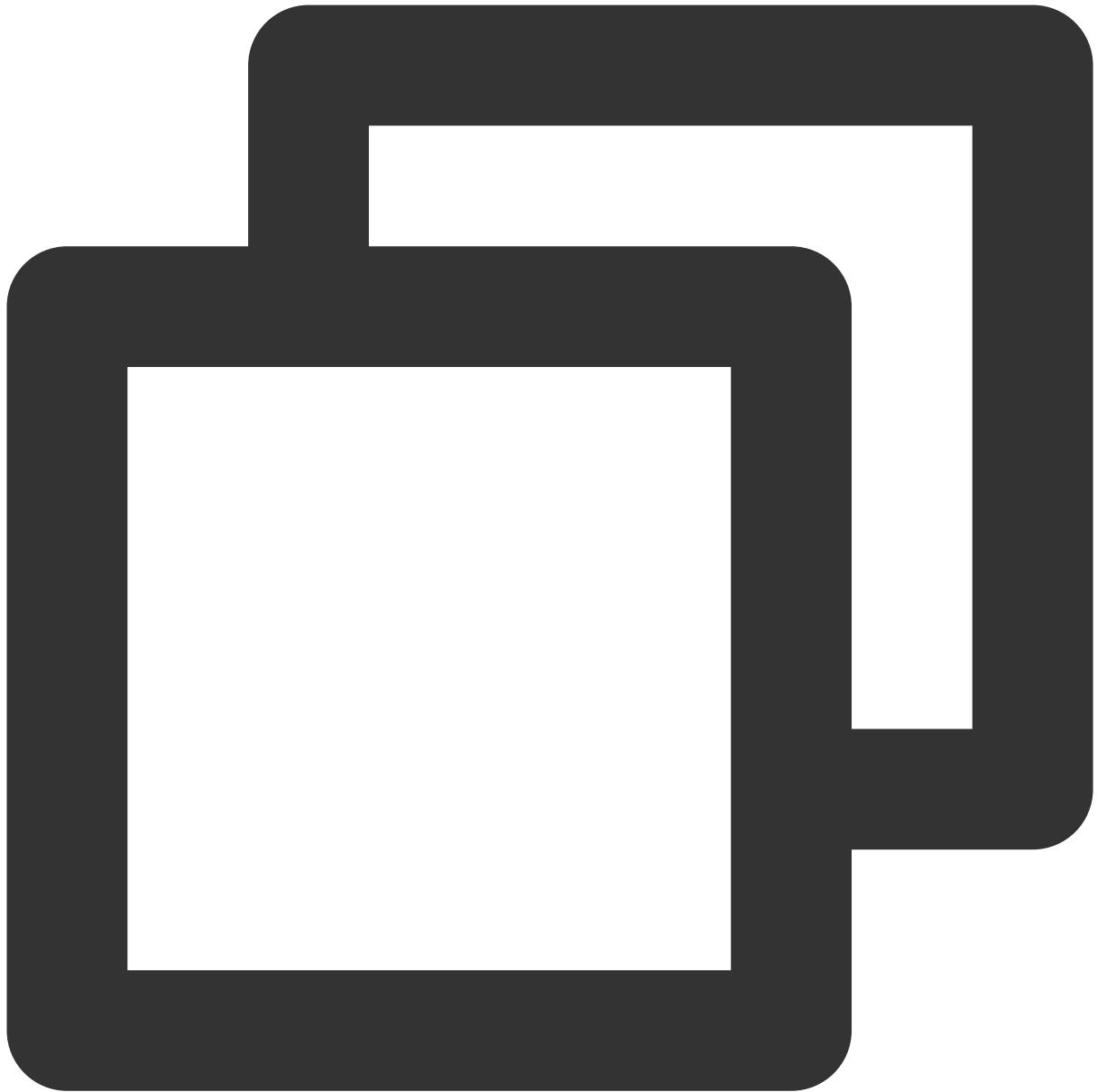
2. Copy the debug directory to your project.

MacOS

Windows



```
cp -r node_modules/@tencentcloud/call-uikit-vue/debug ./src
```



```
xcopy node_modules\\@tencentcloud\\call-uikit-vue\\debug .\\src\\debug /i /e
```

## Step Four: Include the TUICallKit Component

1. The following code example includes four parameters: SDKAppID, SDKSecretKey, userID, and callUserID.

SDKAppID: The Audio and Video Application's SDKAppID created in Tencent Cloud, refer to [Activate the Service](#).

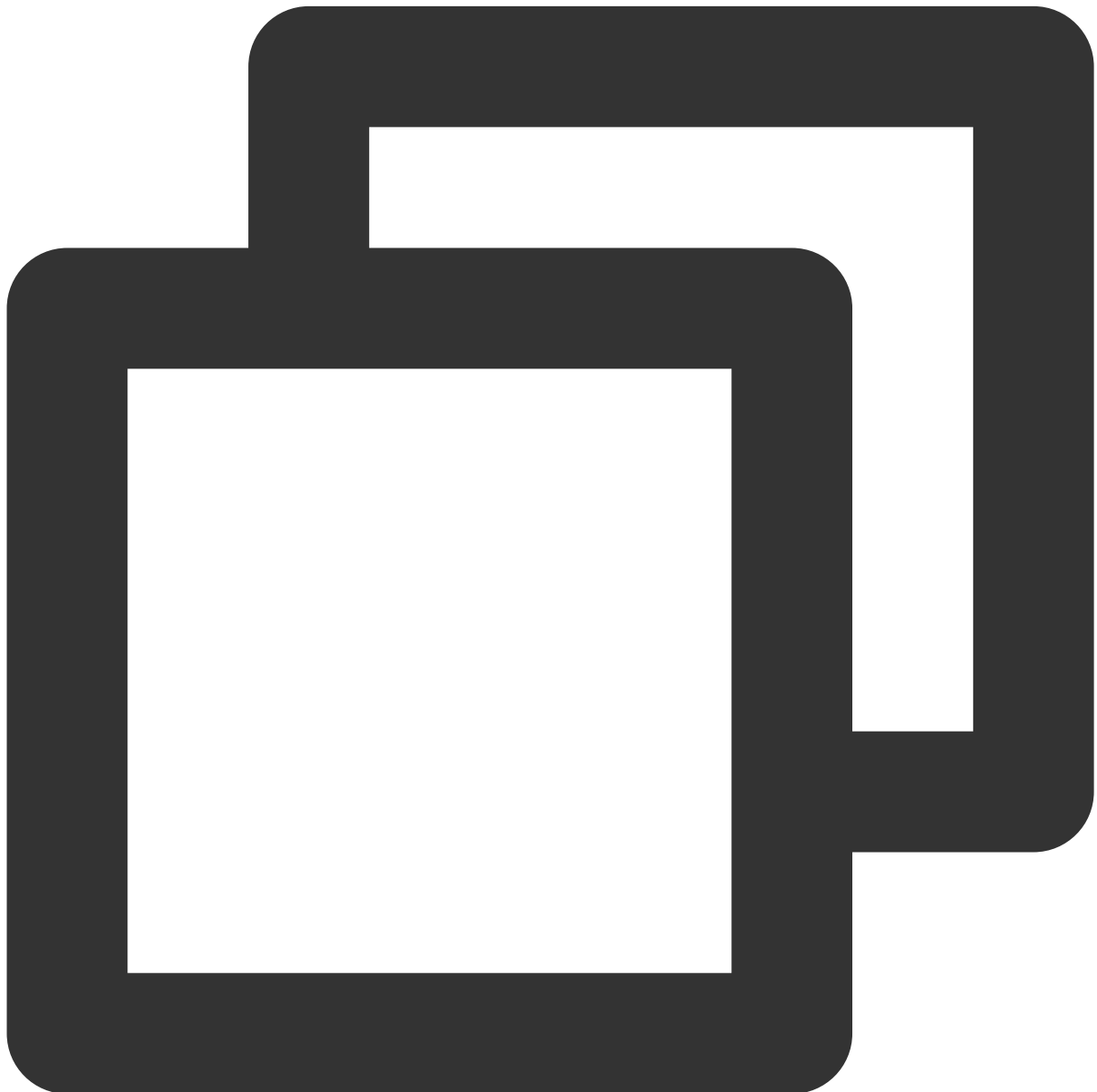
SDKSecretKey: User Signature, refer to [Activate the Service](#).

userID: The caller's userID. It's a string type that can only include English letters (a to z and A to Z), numbers (0 to 9), hyphens (-), and underscores (\_).

callUserID: The called party's userID which must already exist for initialisation. (Within the demo, input the callUserID if there is a callee, otherwise if there isn't a callee it can be left blank).

2. In `tuicallkit-demo/App.vue`, include the following code.

**Note:** The TUICallKit component needs to be placed inside a DOM node to display and control the position, width, height, and other styles of the TUICallKit.



```
<template>
  <div style="padding: 20px 0 0 20px">
```

```
<div><span>userID:</span><input type="text" v-model="userID" /></div>
<div style="margin-top: 6px;"><span>callUserID:</span><input type="text" v-mode
<button @click="init()" style="margin: 10px 10px 10px 0; height: 28px;"> Step 1
<button @click="call()" style="margin: 10px 10px 10px 0; height: 28px;"> Step 2
<div style="width: 50rem; height: 35rem; border: 1px solid salmon;">
  <TUICallKit />
</div>
</div>
</template>
<script lang="ts" setup>
import { ref } from 'vue';
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit
import * as GenerateTestUserSig from "../debug/GenerateTestUserSig-es";

const userID = ref('');
const callUserID = ref('');
const SDKAppID = 0;          // Replace with your SDKAppID
const SDKSecretKey = '';    // Replace with your SDKSecretKey

async function init() {
  try {
    const { userSig } = GenerateTestUserSig.genTestUserSig({
      userID: userID.value,
      SDKAppID: SDKAppID,
      SecretKey: SDKSecretKey,
    });
    await TUICallKitServer.init({ // [1] Initialize the TUICallKit component
      SDKAppID,
      userID: userID.value,
      userSig,
    });
    alert("[TUICallKit] Initialization success.");
  } catch (error: any) {
    alert(`[TUICallKit] Initialization failed. Reason: ${error}`);
  }
}

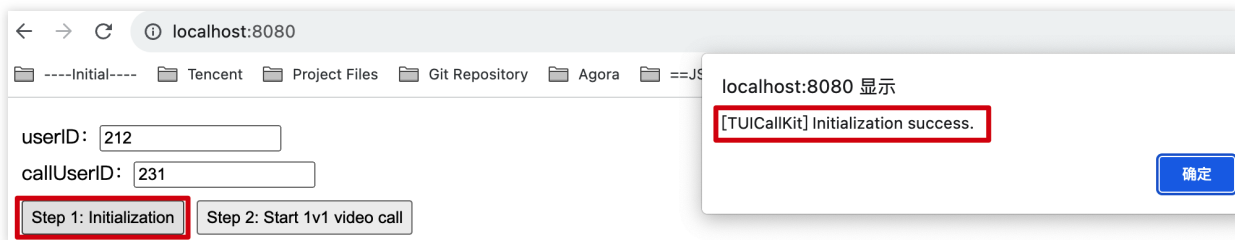
async function call() { // [2] Make a 1v1 video call
  try {
    await TUICallKitServer.call({ userID: callUserID.value, type: TUICallType.VIDEO
  } catch (error: any) {
    alert(`[TUICallKit] Call failed. Reason: ${error}`);
  }
}
</script>
```

## Step Five: Make your first phone call

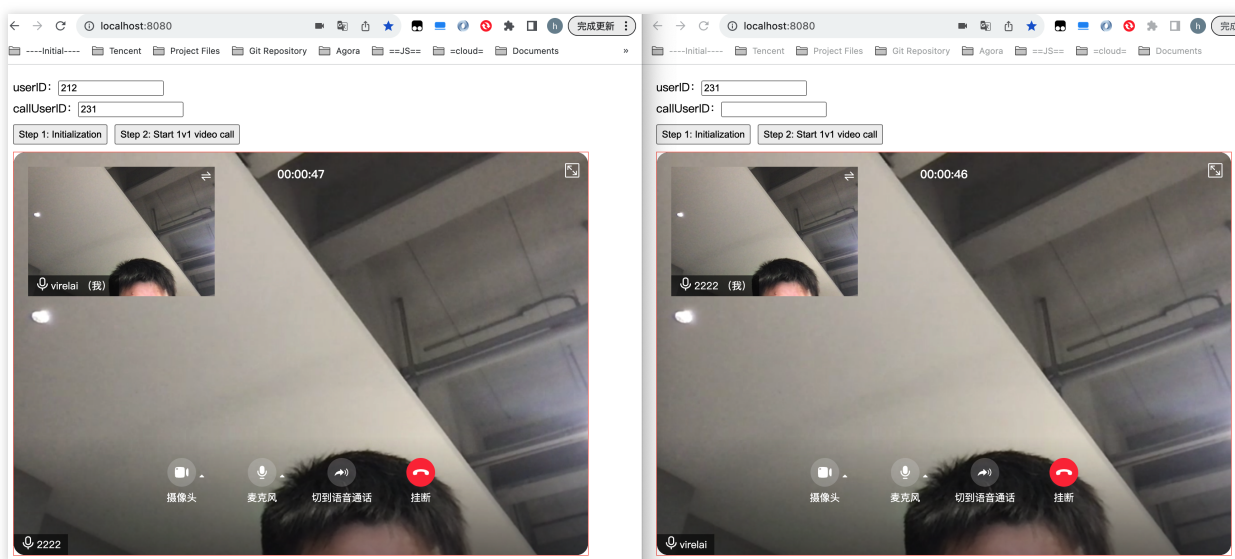
In the terminal, input `npm run serve` to run tuicallkit-demo.

**Alert:** Visit locally through localhost protocol, for details refer to [Network access protocol description](#).

Press the `Step 1: Initialization` button, the pop-up **Initialization successful** indicates the successful initialization.



Click on `Step 2: Start 1v1 video call` to initiate a 1v1 video call, the actual effect is demonstrated in the following image.



## Additional Features

[Setting Nickname, Avatar](#)

[Group Call](#)

[Floating Window](#)

[Custom Ringtone](#)

[Call Status Monitoring, Component Callback Event](#)

[Setting Resolution, Fill Pattern](#)

[Customize Interface](#)

## FAQs

If you encounter any issues during integration and use, please refer to [Frequently Asked Questions](#).

## Technical Consultation

If you have any requirements or feedback, you can contact: [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# Flutter

Last updated : 2024-04-03 17:23:11

This document describes how to quickly integrate the TUICallKit component. Performing the following key steps generally takes about 10 minutes, after which you can implement the audio and video call feature with complete UIs.

## Environment Preparations

Flutter 3.0 or higher version.

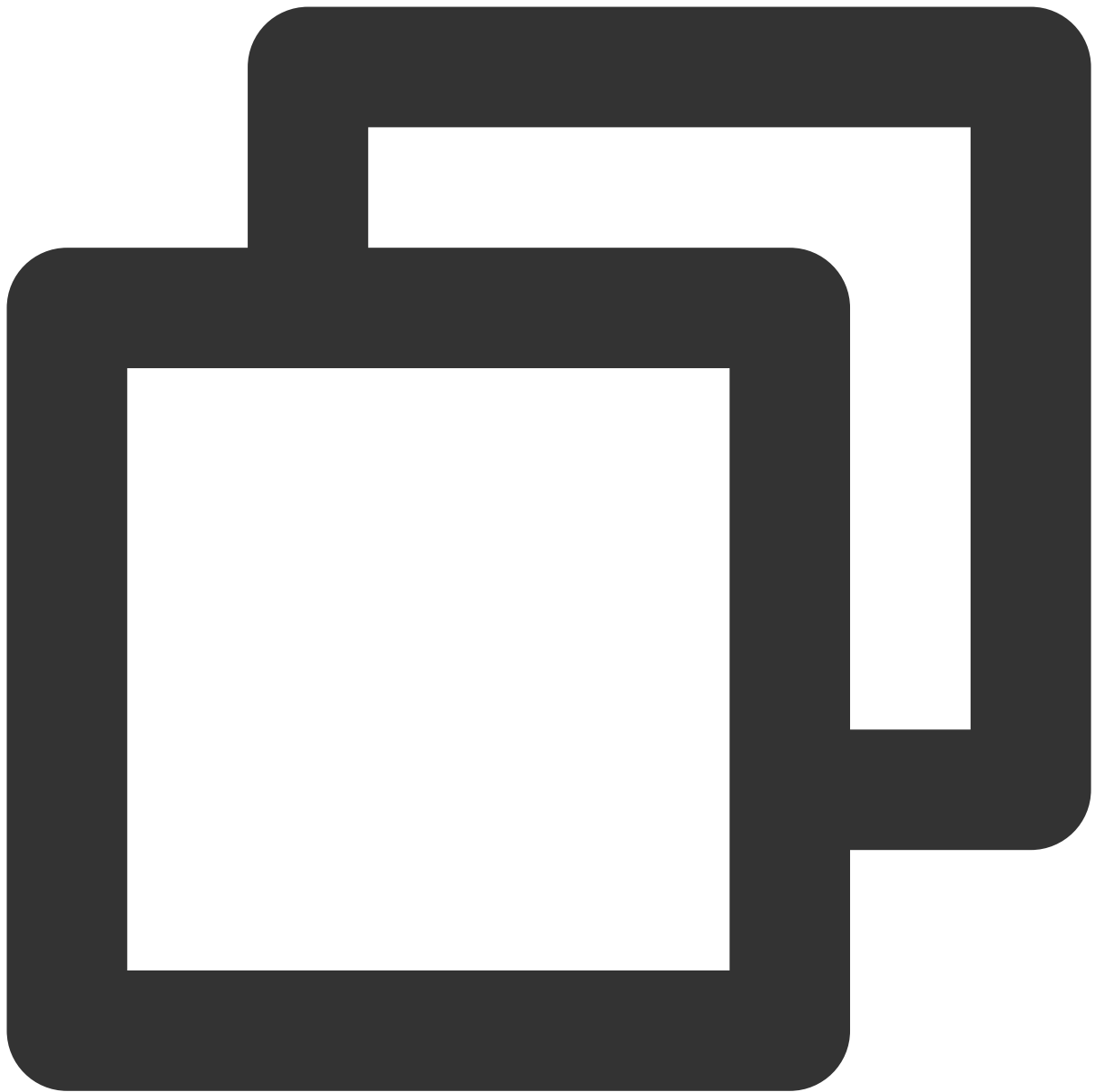
### Step 1. Activate the service

Before using the Audio and Video Services provided by Tencent Cloud, you need to go to the Console and activate the service for your application. For detailed steps, refer to [Activate Service](#).

### Step 2. Import the component

Execute the following command in the command line to install the `tencent_calls_uikit` plugin.

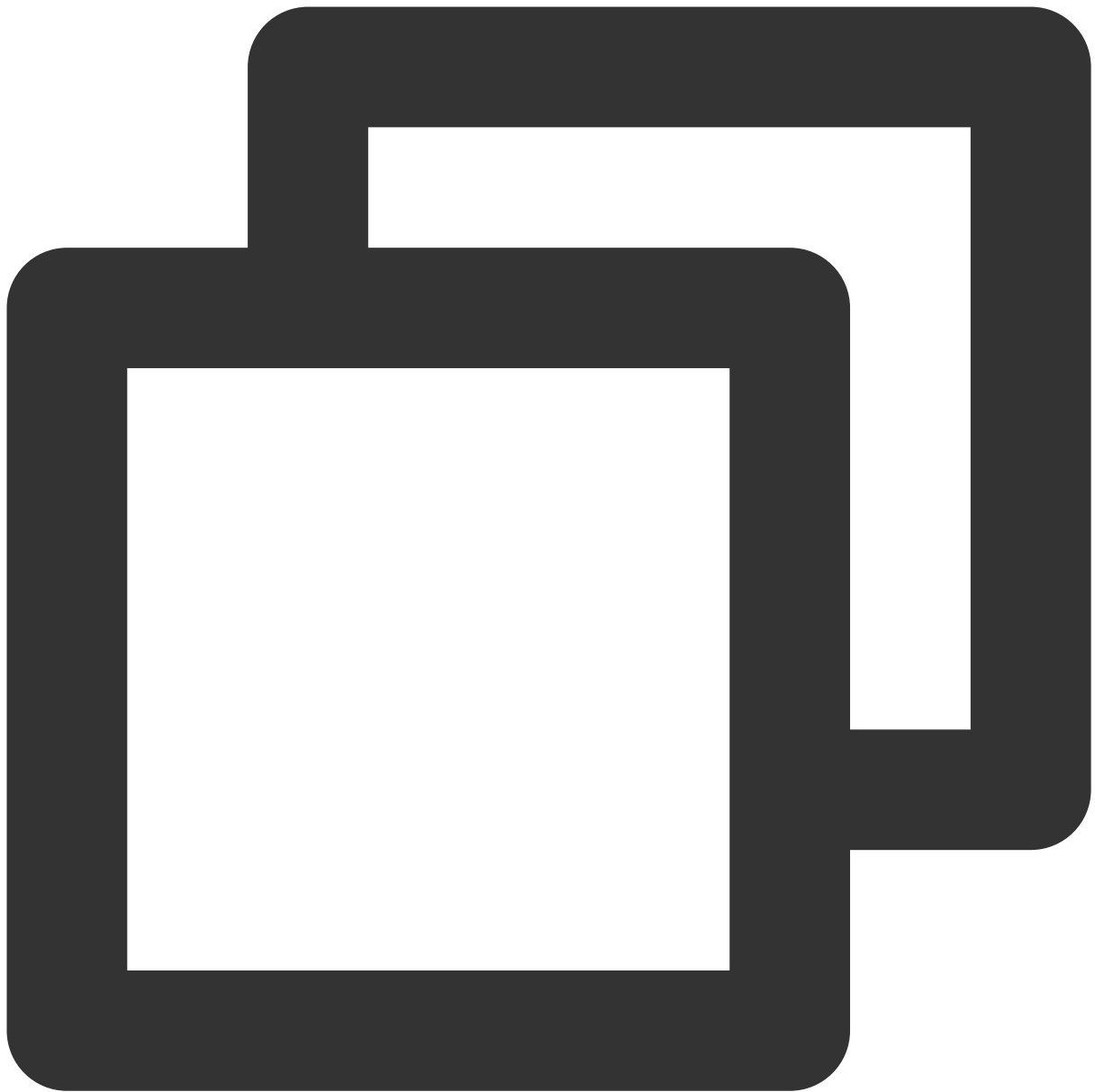




```
flutter pub add tencent_calls_uikit
```

## Step 3. Configure the project

1. Add the navigatorObserver of TUICallKit to the App component, taking MaterialApp as an example, the code is as follows:

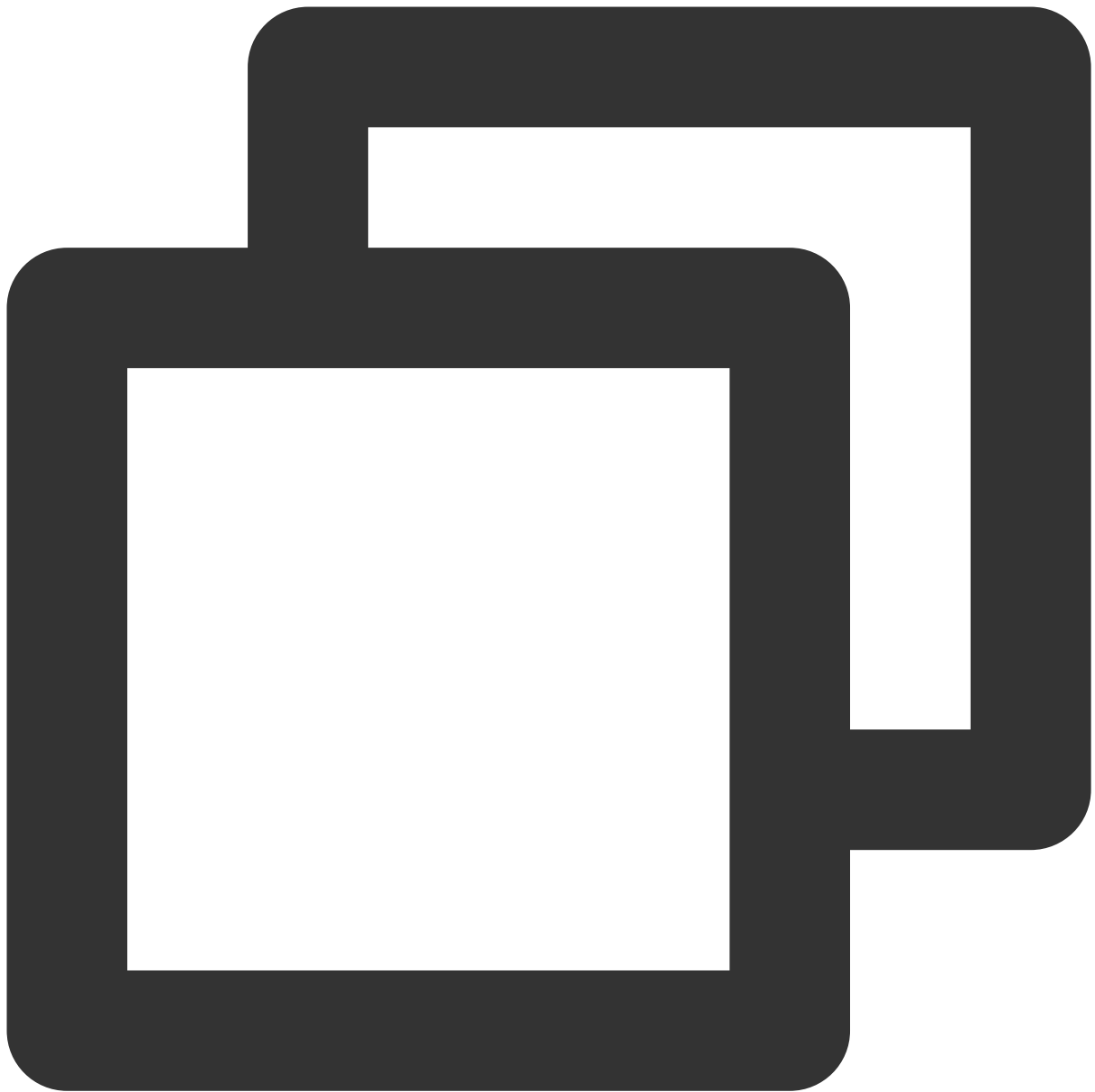


```
import 'package:tencent_calls_uikit/tuicall_kit.dart';

MaterialApp (
  navigatorObservers : [TUICallKit.navigatorObserver],
  ...
)
```

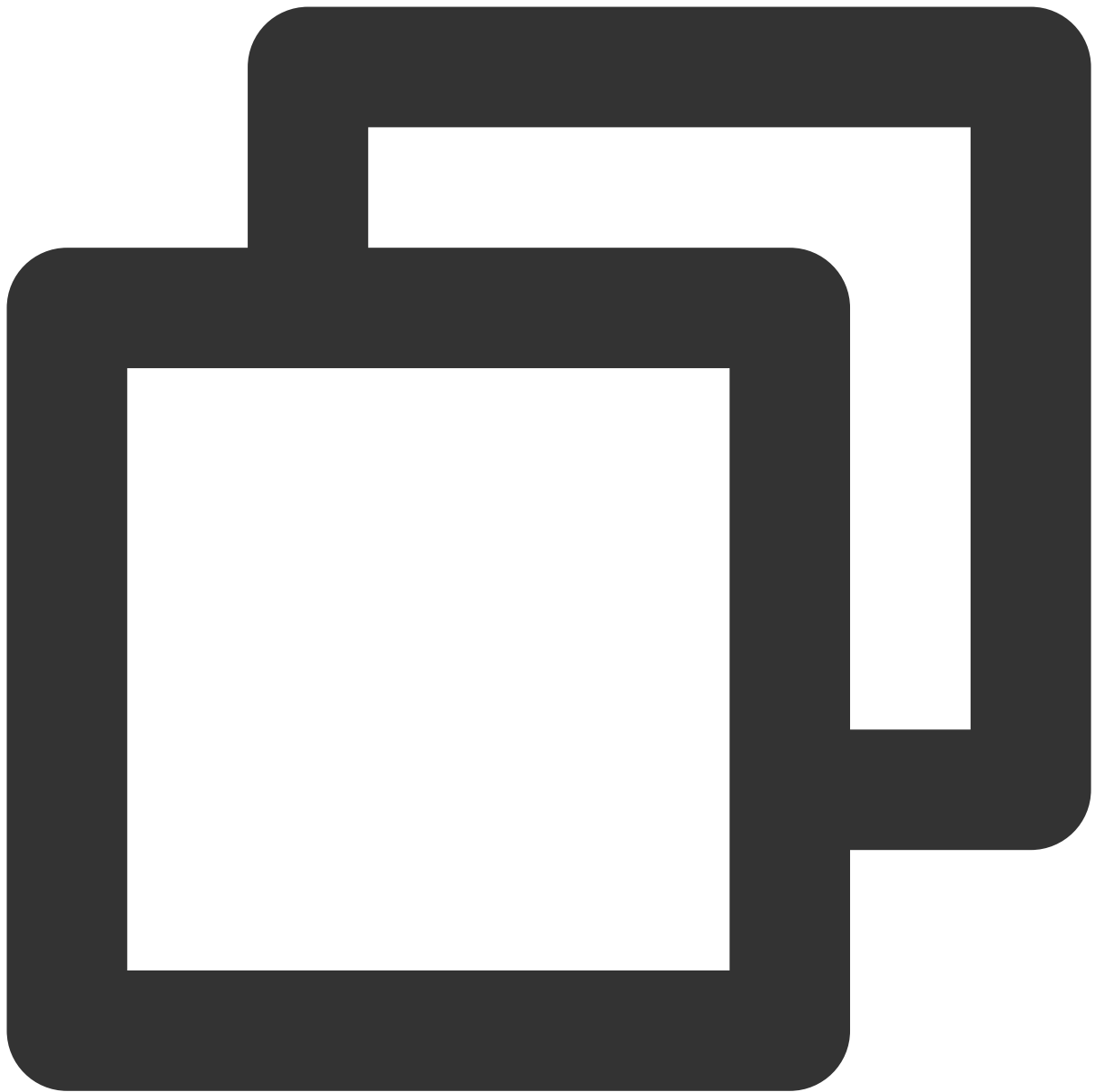
2. If you need to compile and run on the Android platform, since the SDK uses Java's reflection feature internally, certain classes in the SDK must be added to the non-aliasing list.

Firstly, you need to enable aliasing rules by configuring them in the build.gradle file under the app directory:



```
android {  
    ...  
    buildTypes {  
        release {  
            minifyEnabled true  
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard  
        }  
    }  
}
```

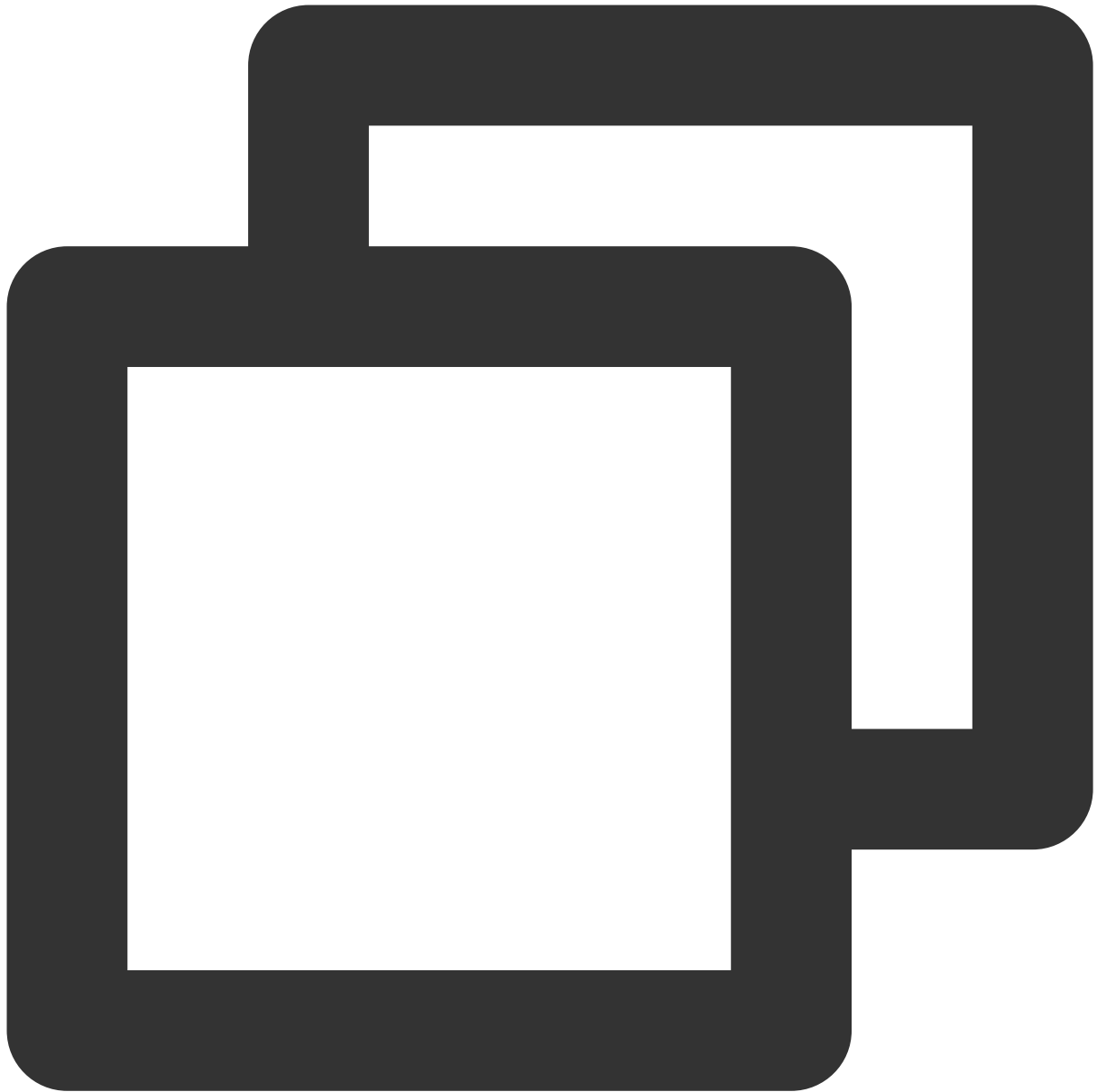
Then add the following code to the proguard-rules.pro file:



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

3. If your project needs to be debugged on the iOS simulator, you need to add the following code to the project's

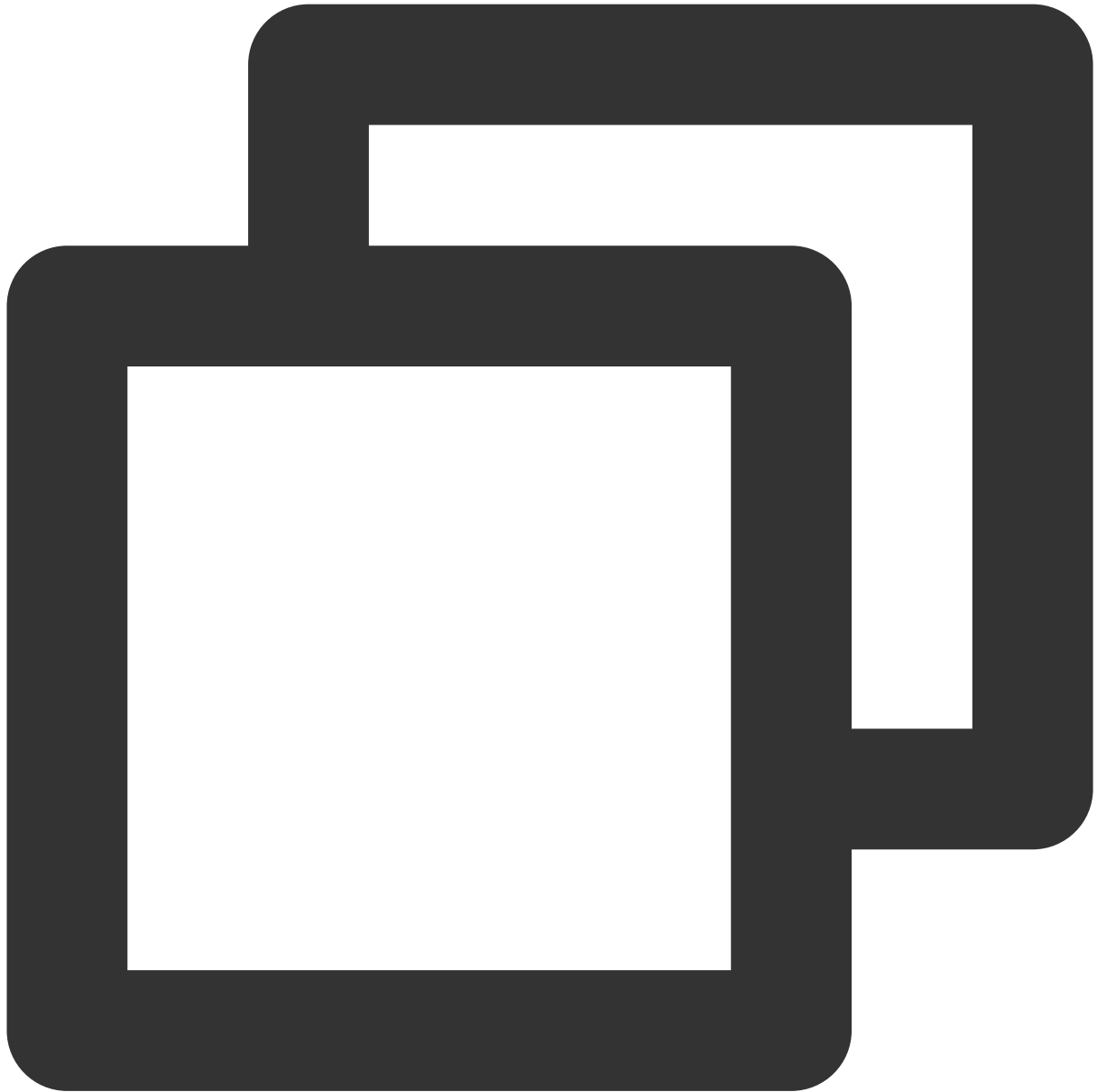
```
/ios/Podfile :
```



```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
    target.build_configurations.each do |config|
      config.build_settings['VALID_ARCHS'] = 'arm64 arm64e x86_64'
      config.build_settings['VALID_ARCHS[sdk=iphonesimulator*]'] = 'x86_64'
    end
  end
end
```

4. If you need to use the audio and video feature on iOS, you need authorization for microphone and camera usage.

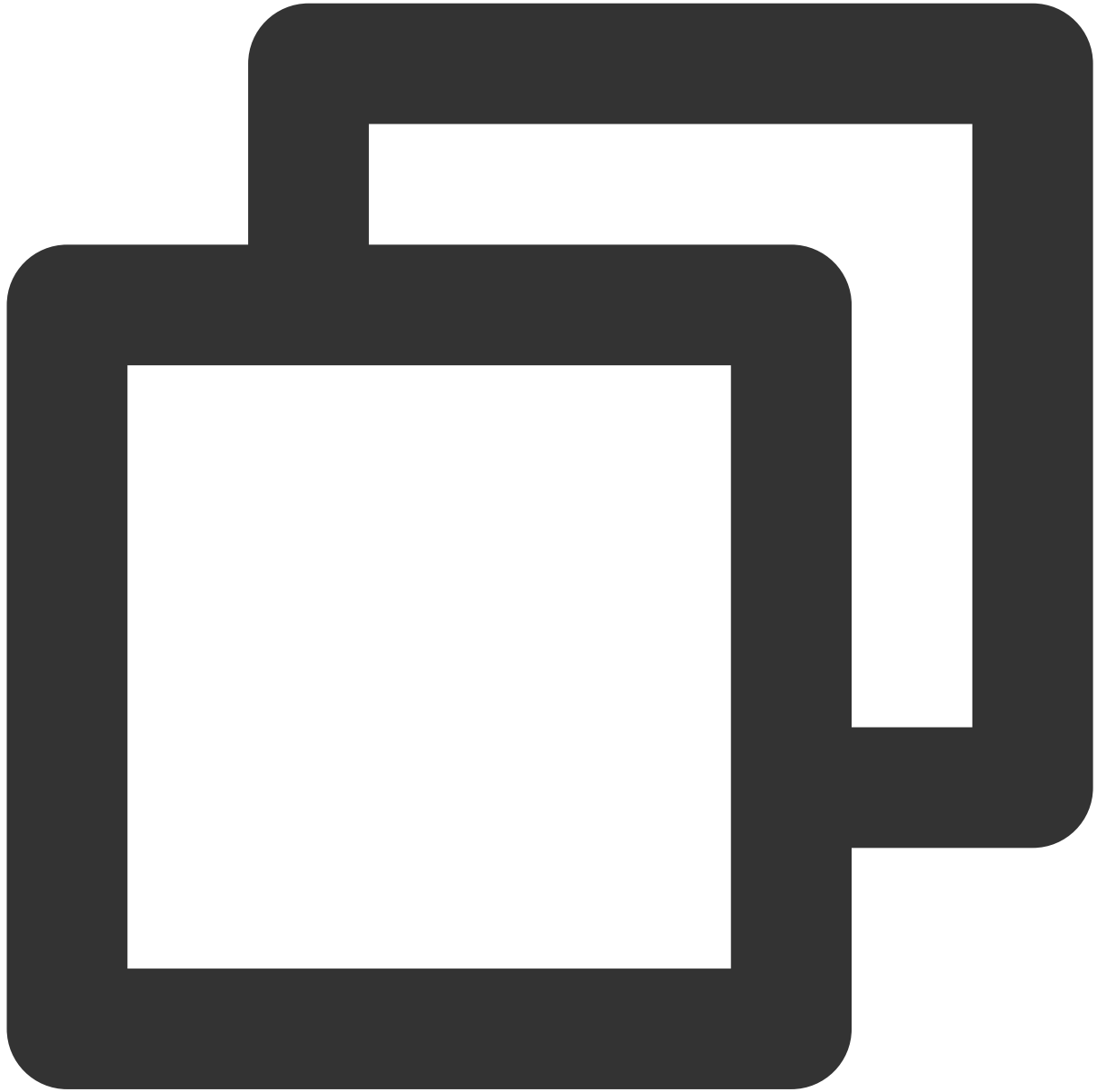
Authorization method: Add the following two items in your iOS project's Info.plist, corresponding to the prompt messages for the microphone and camera when the system pops up the authorization dialogue.



```
<key>NSCameraUsageDescription</key>  
<string>CallingApp needs to access your camera to capture video.</string>  
<key>NSMicrophoneUsageDescription</key>  
<string>CallingApp needs to access your microphone to capture voice.</string>
```

## Step 4. Log in to the `TUICallKit` component

Add the following code to your project to call the relevant APIs in `TUICore` to log in to the `TUICallKit` component. This step is very important, as the user can use the component features properly only after a successful login. Carefully check whether the relevant parameters are correctly configured:



```
TUIResult result = TUICallKit.instance.login(SDKAppID, // Please replace it with  
                                             'userId', // Please replace with  
                                             'userSig'); // You can get a UsersS
```

**Parameter description:** The key parameters used by the `login` function are as detailed below:

**SDKAppID:** Obtained in the last step in step 1 and not detailed here.

**UserID:** The ID of the current user, which is a string that can contain only letters (a–z and A–Z), digits (0–9), hyphens (-), or underscores (\_).

**UserSig:** The authentication credential used by Tencent Cloud to verify whether the current user is allowed to use the TRTC service. You can get it by using the `SDKSecretKey` to encrypt the information such as `SDKAppID` and `UserID`. You can generate a temporary `UserSig` by clicking the [UserSig Generate](#) button in the console.

For more information, see [UserSig](#).

### Note

**Many developers have contacted us with many questions regarding this step. Below are some of the frequently encountered problems:**

SDKAppID is invalid.

userSig is set to the value of Secretkey mistakenly. The userSig is generated by using the SecretKey for the purpose of encrypting information such as sdkAppId, userId, and the expiration time. But the value of the userSig that is required cannot be directly substituted with the value of the SecretKey.

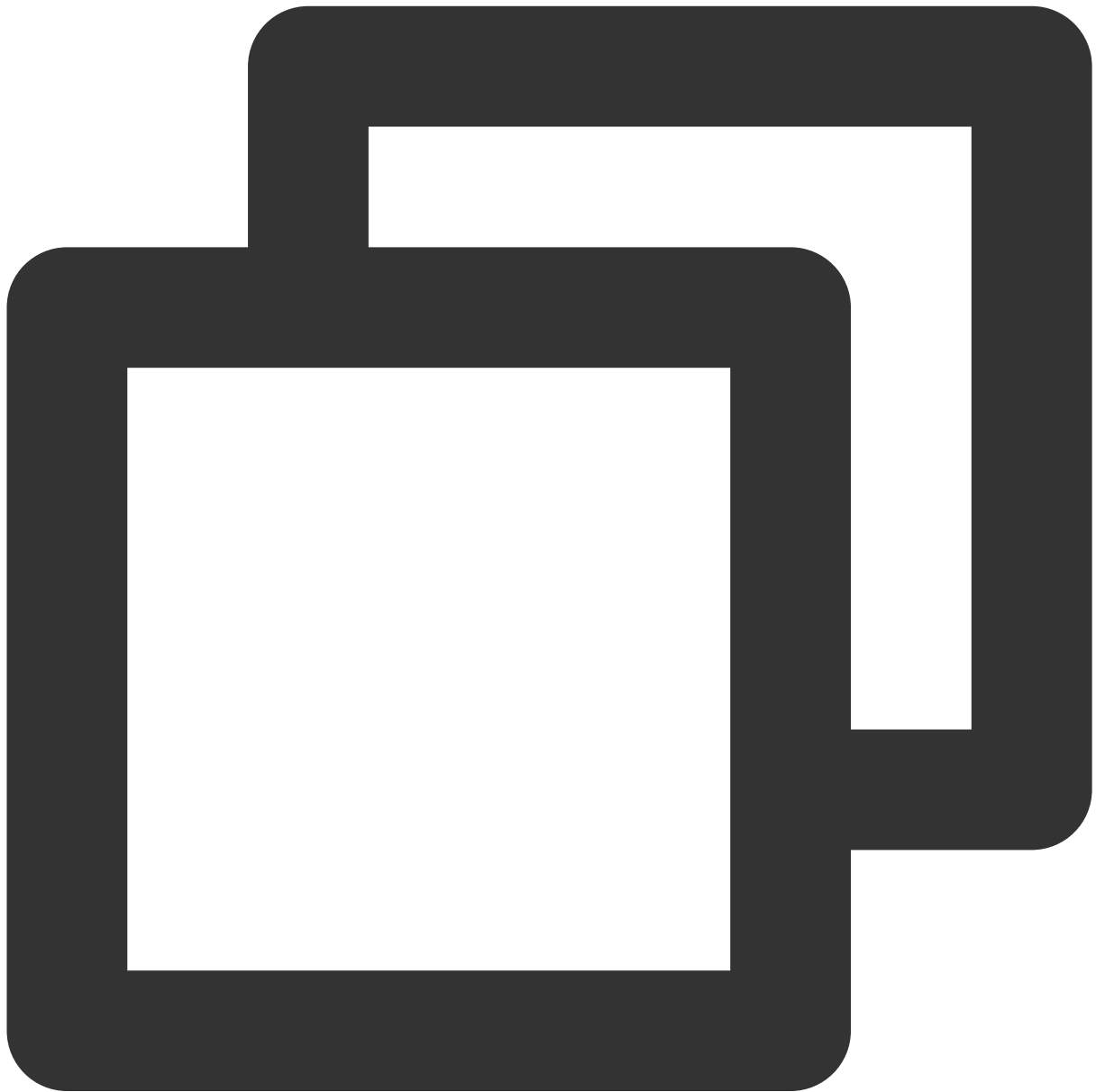
userId is set to a simple string such as 1, 123, or 111, and your colleague may be using the same userId while working on a project simultaneously. In this case, login will fail as TRTC doesn't support login on multiple terminals with the same UserID. Therefore, we recommend you use some distinguishable userId values during debugging.

The sample code on GitHub uses the `genTestUserSig` function to calculate `UserSig` locally, so as to help you complete the current integration process more quickly. However, this scheme exposes your `SecretKey` in the application code, which makes it difficult for you to upgrade and protect your `SecretKey` subsequently. Therefore, we strongly recommend you run the `UserSig` calculation logic on the server and make the application request the `UserSig` calculated in real time every time the application uses the `TUICallKit` component from the server.

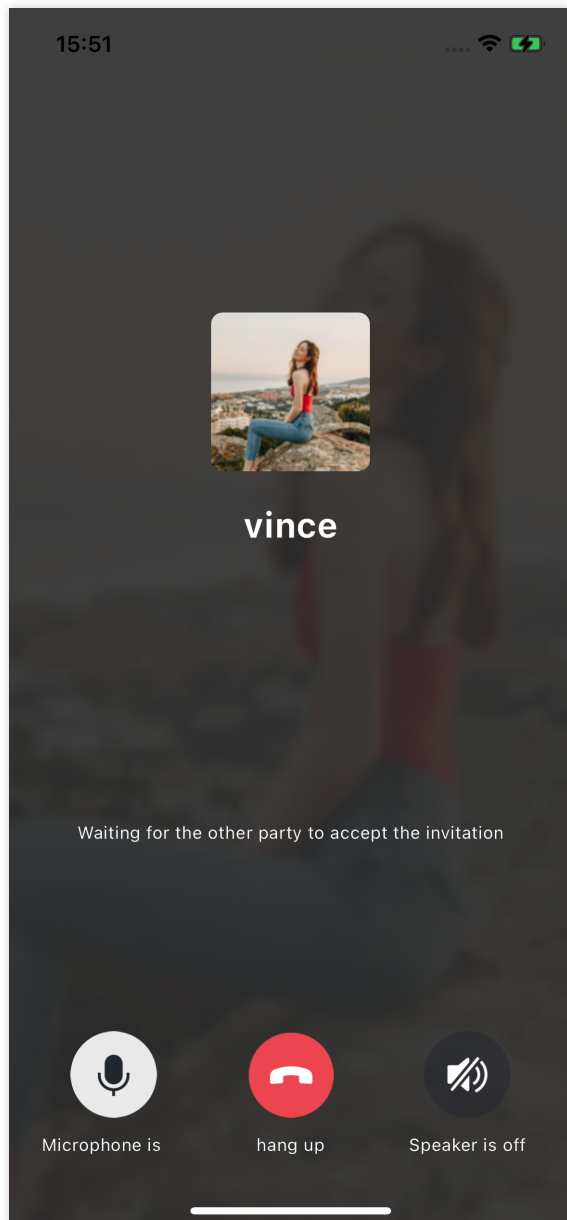


## Step 5. Make your first phone call

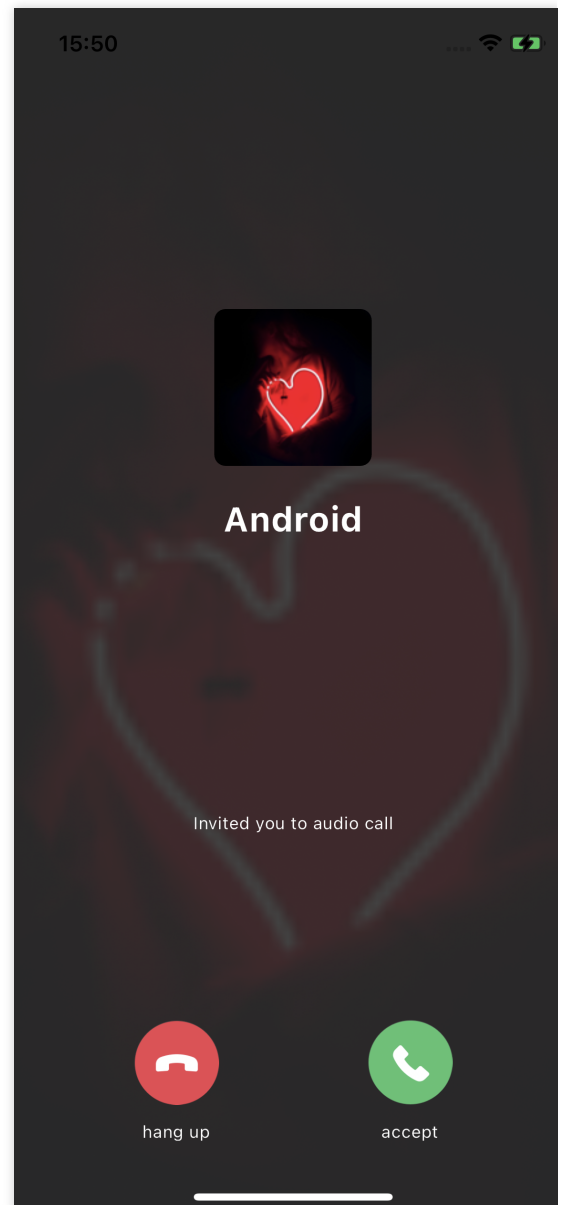
After both the caller and callee have successfully signed in, the caller can initiate an audio or video call by calling the TUICallKit's call method and specifying the call type and the callee's userId. At this point, the callee will receive an incoming call invitation.



```
TUICallKit.instance.call('mike', TUICallMediaType.video);
```



Caller



Callee

## Additional Features

[Customize Interface](#)[Offline Push](#)[Group Call](#)[Floating Window](#)[Custom Ringtone](#)[Call Status Monitoring](#)[Cloud Recording](#)

[Beauty Effects](#)

## FAQs

If you encounter any issues during integration and use, please refer to [Frequently Asked Questions](#).

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# uniapp (Android&iOS)

Last updated : 2024-04-03 17:23:11

This article will guide you through the quick integration of the TUICallKit component. You will complete several key steps within 10 minutes, ultimately obtaining a video call feature with a complete UI interface.



## TUICallKit Demo Experience

TUICallKit Plugin Address: [TUICallKit Plugin Link](#) .

If you want to quickly run a new project, please read [uni-app Demo Quick Start](#) directly.

## Environment Requirements

Development Tool Notes: It is recommended to use the latest version of the HBuilderX editor.

Plugin Debugging Notes: Native plugins do not currently support simulator debugging.

iOS Device Requirements: iOS system  $\geq 9.0$ , supports audio and video calls on actual devices.

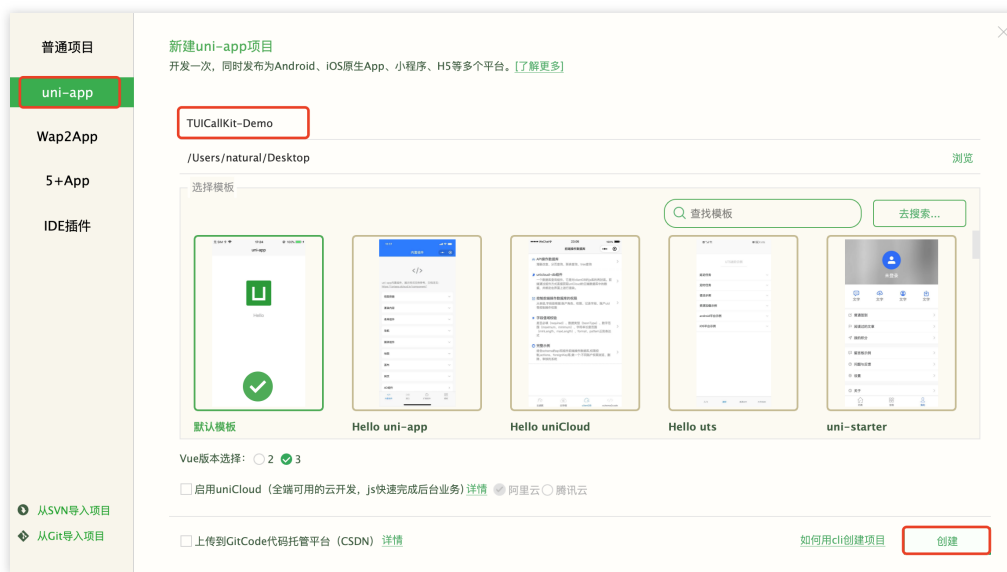
Android Device Requirements: Android system  $\geq 5.0$  (SDK API Level 21), supports audio and video calls on actual devices, [Allow USB Debugging](#).

## Step 1. Activate the service

Before using the Audio and Video Services provided by Tencent Cloud, you need to go to the Console and activate the service for your application. For detailed steps, refer to [Activate Service](#).

## Step 2: Create a uni-app Project

Open HBuilderX development tool, click to create a new uni-app project: Project Name (TUICallKit-Demo).



## Step 3: Download and import the TUICallKit Plugin

1. Visit [TencentCloud-TUICallKit Plugin](#), purchase the plugin on the plugin detail page, and select the corresponding AppID, ensuring the package name is correct.

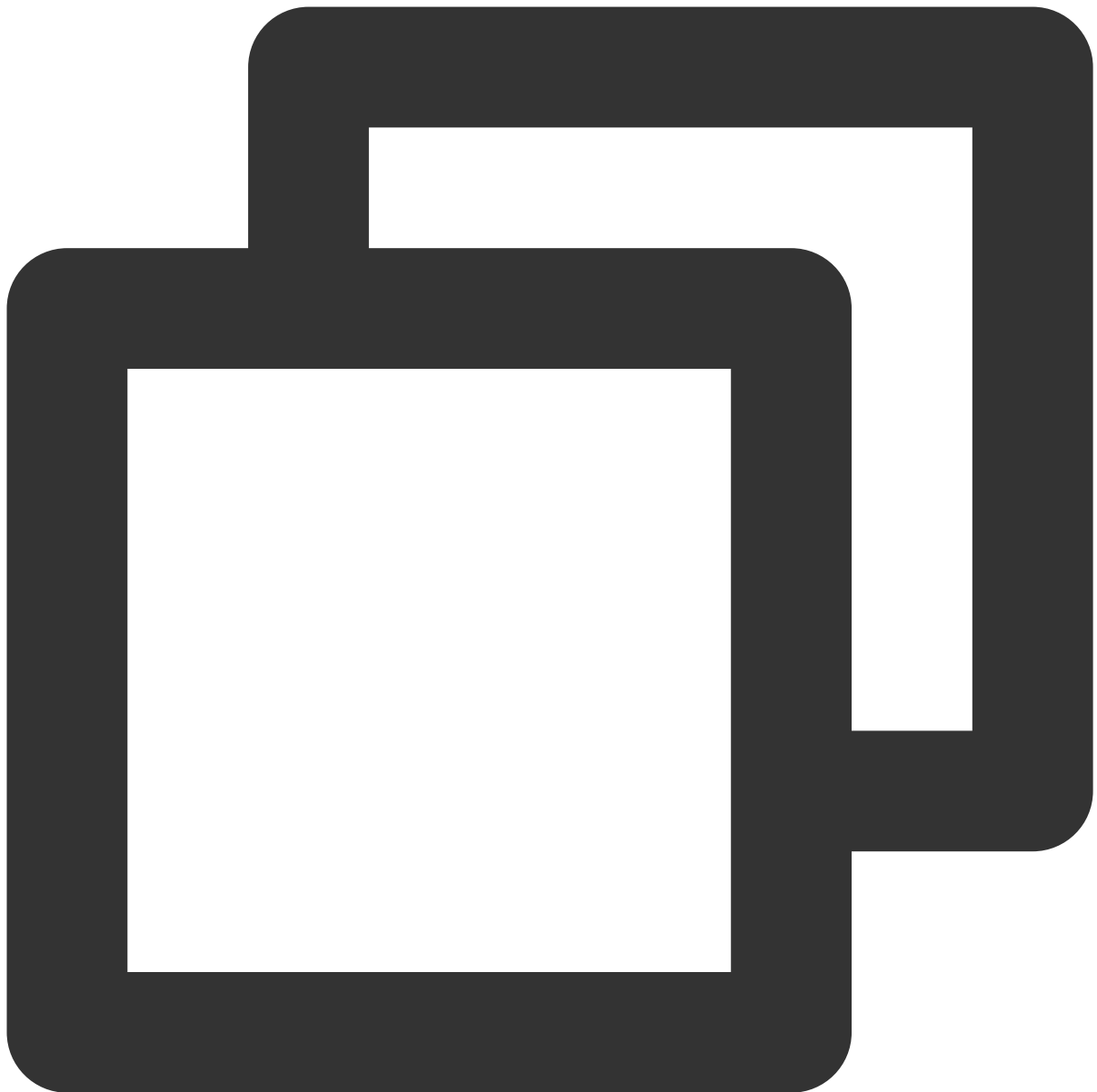


2. Import the plugin in the `TUICallKit-Demo` project .



## Step 4: Use the TUICallKit Plugin

1. Import the following code in `TUICallKit-Demo/pages/index/index.vue` .



```
<template>
  <view class="container">
    <input type="text" v-model="inputID" :placeholder=" isLogin ? 'plea
    <text v-show="isLogin"> your userID: {{ userID }} </text>
    <button v-show="!isLogin" @click="handleLogin"> Login </button>
    <button v-show="isLogin" @click="handleCall"> start call </button>
  </view>
</template>
<script>
  const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit'); //[1
  uni.$TUICallKit = TUICallKit;
```

```

import { genTestUserSig } from '../../debug/GenerateTestUserSig.js'
export default {
  data() {
    return {
      inputID: '',
      isLogin: false,
      userID: '',
    }
  },
  methods: {
    handleLogin() {
      this.userID = this.inputID;
      const { userSig, sdkAppID: SDKAppID } = genTestUserSig
      const loginParams = { SDKAppID, userID: this.userID }
      // [2] Login
      uni.$TUICallKit.login( loginParams, res => {
        if (res.code === 0) {
          this.isLogin = true;
          this.inputID = '';
          console.log('[TUICallKit] login success')
        } else {
          console.error('[TUICallKit] login failed, failed message: ' + res.message)
        }
      })
    },
    handleCall() {
      try {
        const callParams = {
          userID: this.inputID,
          callMediaType: 2, // 1 -- audio call, 2 -- video call
          callParams: { roomID: 234, strRoomName: 'Room 234' }
        };
        // [3] start 1v1 video call
        uni.$TUICallKit.call( callParams, res => {
          console.log('[TUICallKit] call success')
        })
        this.inputID = '';
      } catch (error) {
        console.log('[TUICallKit] call error: ' + error.message)
      }
    }
  }
}

```

</script>

<style>



```
.container {  
    margin: 30px;  
}  
.container input {  
    height: 50px;  
    border: 1px solid;  
}  
.container button {  
    margin-top: 30px;  
}  
</style>
```

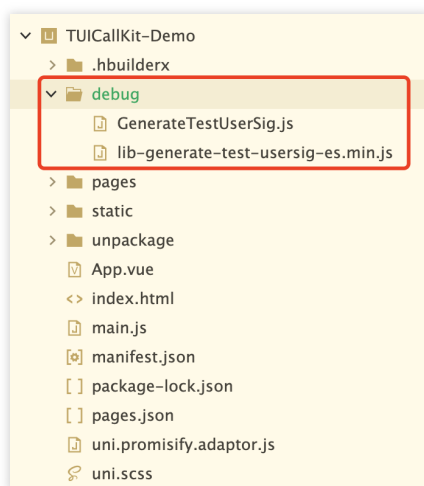
2. Enter the SDKAppID, SDKSecretKey, and userSig parameters.

Client-side userSig generation

Console-generated userSig

Due to the time sensitivity of UserSig, **in a testing environment, it is recommended to use this method.**

1. [Click to download the debug folder](#), and copy the debug directory to your project, as shown below:



2. Fill in the `TUICallKit-Demo/debug/GenerateTestUserSig.js` file with `SDKAppID` and `SDKSecretKey` (refer to [Activate Service](#))

```
GenerateTestUserSig.js
1 import LibGenerateTestUserSig from './lib-generate-test-usersig-es.min.js';
2 /** ...
11
12 const SDKAPPID = 0;
13 /** ...
19
20 const EXPIRETIME = 604800;
21 /** ...
31
32 const SECRETKEY = '';
33 /** ...
50
51 export function genTestUserSig(userID) {
52   const generator = new LibGenerateTestUserSig(SDKAPPID, SECRETKEY, EXPIRETIME);
53   const userSig = generator.genTestUserSig(userID);
54
55   return {
56     sdkAppID: SDKAPPID,
57     userSig,
58   };
59 }
```

If you want to quickly experience TUICallKit, you can generate a temporary UserSig available through the [Auxiliary Tools](#) in the console.



If you are using **Console Generation**, you will need to assign the `SDKAppID`, `userSig` values in the `TUICallKit-Demo/pages/index/index.vue` file.



## Step Five: Make your first phone call

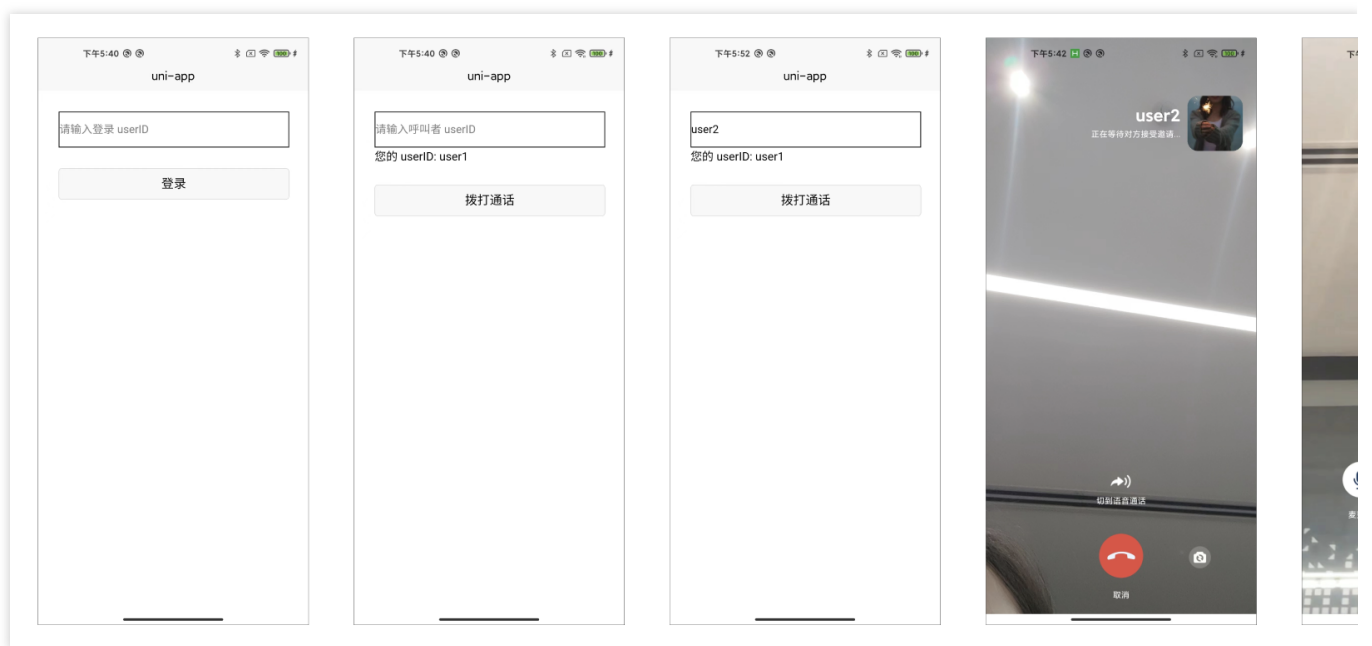
1. To create a custom debugging base, please use the **Traditional Packaging** method.



2. After successfully creating the custom debugging base, **run the project Using Custom Base**.



3. The specific effect of making a 1v1 video call is shown in the figure.



## Additional Features

[Setting Nickname, Avatar](#)

[Group Call](#)

[Floating Window](#)

[Custom Definition Ringtone](#)

[Call Status Monitoring](#)

## FAQs

If you encounter any issues during integration and use, please refer to [Frequently Asked Questions](#).

## Related Case - Online Customer Service Scenario

We provide the source code related to the **Online Customer Service Scenario**. We recommend you [download the Online Customer Service Scenario](#) and integrate it for the experience. This scenario provides a basic template with sample customer groups + sample friends, featuring:

Support for sending text messages, image messages, Voice Message Service, and video messages.

Supports two-person voice and video call features.

Supports creating group chat sessions, group member management, etc.



## Technical Consultation

If you have any requirements or feedback, you can contact: [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# UI Customization (TUICallKit)

## Android

Last updated : 2024-01-18 12:08:17

This document describes how to customize the UI of `TUICallKit` and provides two schemes for customization: **slight UI adjustment** and **custom UI implementation**.

### Scheme 1. Slight UI Adjustment

You can adjust the UI of `TUICallKit` by directly modifying the UI source code in the `Android/tuicallkit` folder in [tencentyun/TUICallKit](#).

#### Replacing icons

You can directly replace the icons in the `res\\drawable-xxhdpi` folder to customize the color tone and style of all the icons in your application. When you replace an icon, make sure the filename is the same as the original icon.



#### Replacing ringtones

You can replace ringtones by replacing the three audio files in the `res\\raw` folder.

| Filename          | Description                |
|-------------------|----------------------------|
| phone_dialing.mp3 | The sound of making a call |

|                   |                                 |
|-------------------|---------------------------------|
| phone_hangup.mp3  | The sound of being hung up      |
| phone_ringing.mp3 | The ringtone for incoming calls |

## Replacing text

You can modify the strings on the video call UI by modifying the `strings.xml` file in `values-zh` and `values-e` .

## Scheme 2. Custom UI Implementation

The entire call feature of `TUICallKit` is implemented based on the UI-less component `TUICallEngine` . You can delete the `tuicallkit` folder and implement your own UIs based entirely on `TUICallEngine` .

### TUICallEngine

`TUICallEngine` is the underlying API of the entire `TUICallKit` component. It provides key APIs such as APIs for making, answering, declining, and hanging up one-to-one audio/video and group calls and device operations.

| API                          | Description  |
|------------------------------|--|
| <code>createInstance</code>  | Creates a <code>TUICallEngine</code> instance (singleton).           |
| <code>destroyInstance</code> | Terminates a <code>TUICallEngine</code> instance (singleton).        |
| <code>init</code>            | Completes the authentication of basic audio/video call capabilities. |
| <code>addObserver</code>     | Registers an event listener.   |
| <code>removeObserver</code>  | Unregisters an event listener.                                       |
| <code>call</code>            | Makes a one-to-one call.   |
| <code>groupCall</code>       | Makes a group call.  |
| <code>accept</code>          | Answers a call.  |
| <code>reject</code>          | Declines a call.   |
| <code>hangup</code>          | Hangs up a call.   |
| <code>ignore</code>          | Ignores a call.  |
| <code>inviteUser</code>      | Invites a user during a group call.                                  |
| <code>joinInGroupCall</code> | Joins the current group call actively.                               |

|  |   |
|--|---|
| <code>switchCallMediaType</code>       | Switches the call media type, such as from video call to audio call.  |
| <code>startRemoteView</code>           | Subscribes to the video stream of a remote user.  |
| <code>stopRemoteView</code>            | Unsubscribes from the video stream of a remote user.  |
| <code>openCamera</code>                | Enables the camera.   |
| <code>closeCamera</code>               | Disables the camera.  |
| <code>switchCamera</code>              | Switches between the front and rear cameras.  |
| <code>openMicrophone</code>            | Enables the mic.  |
| <code>closeMicrophone</code>           | Disables the mic.   |
| <code>selectAudioPlaybackDevice</code> | Selects the audio playback device (receiver/speaker on the device).   |
| <code>setSelfInfo</code>               | Sets the user nickname and profile photo.   |
| <code>enableMultiDeviceAbility</code>  | Enables/Disables the multi-device login mode of <code>TUICallEngine</code> (supported by the premium plan). |

## TUICallObserver

`TUICallObserver` is the callback even class of `TUICallEngine`. You can use it to listen on the desired callback events.

| API                                 | Description                        |
|-------------------------------------|------------------------------------|
| <code>onError</code>                | An error occurred during the call. |
| <code>onCallReceived</code>         | A call was received.               |
| <code>onCallCancelled</code>        | The call was canceled.             |
| <code>onCallBegin</code>            | The call was connected.            |
| <code>onCallEnd</code>              | The call ended.                    |
| <code>onCallMediaTypeChanged</code> | The call media type changed.       |
| <code>onUserReject</code>           | A user declined the call.          |
| <code>onUserNoResponse</code>       | A user didn't respond.             |
| <code>onUserLineBusy</code>         | A user was busy.                   |
| <code>onUserJoin</code>             | A user joined the call.            |



|  |                                     |
|--|-------------------------------------|
| <code>onUserLeave</code>                 | A user left the call.               |
| <code>onUserVideoAvailable</code>        | Whether a user had a video stream.  |
| <code>onUserAudioAvailable</code>        | Whether a user had an audio stream. |
| <code>onUserVoiceVolumeChanged</code>    | The volume levels of all users.     |
| <code>onUserNetworkQualityChanged</code> | The network quality of all users.   |

## Definitions of Key Types

| API  | Description  |
|--|--|
| <code>TUICallDefine.MediaType</code>             | The call media type. Enumeration: Video call and audio call.       |
| <code>TUICallDefine.Role</code>                  | The call role. Enumeration: Caller and callee.                     |
| <code>TUICallDefine.Status</code>                | The call status. Enumeration: Idle, waiting, and answering.        |
| <code>TUICommonDefine.RoomId</code>              | The audio/video room ID, which can be a number or string.          |
| <code>TUICommonDefine.Camera</code>              | The camera type. Enumeration: Front camera and rear camera.        |
| <code>TUICommonDefine.AudioPlaybackDevice</code> | The audio playback device type. Enumeration: Speaker and receiver. |
| <code>TUICommonDefine.NetworkQualityInfo</code>  | The information of the current network quality.                    |

# iOS

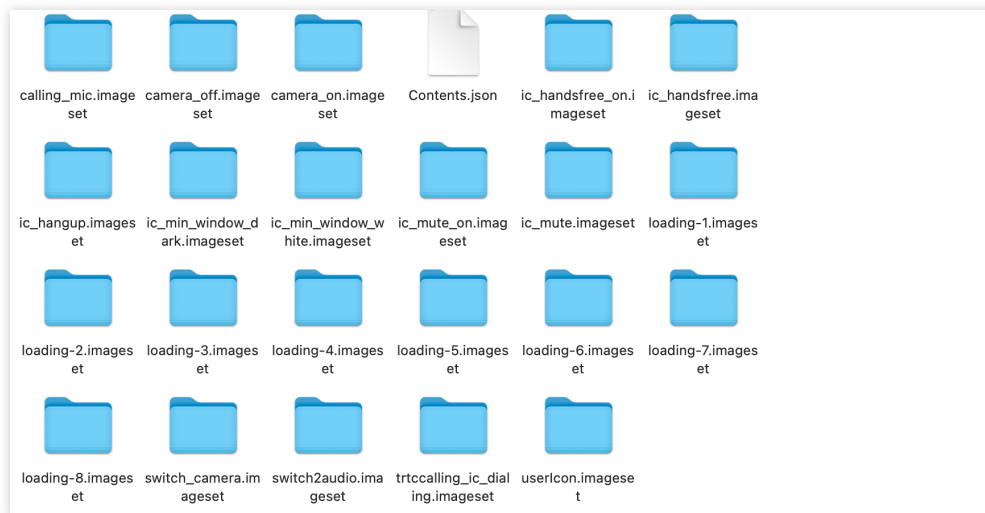
Last updated : 2023-09-19 14:45:50

This document describes how to customize the UI of `TUICallKit` and provides two schemes for customization: **slight UI adjustment** and **custom UI implementation**.

## Scheme 1. Slight UI Adjustment

You can adjust the UI of `TUICallKit` by directly modifying the UI source code in the `iOS/TUICallKit` folder in [tencentyun/tuicallkit](https://github.com/TencentCloud/tencentyun/tuicallkit).

**Replacing icons:** You can directly replace the icons in the `Resources\\Calling.xcassets` folder to customize the color tone and style of all the icons in your application. When you replace an icon, make sure the filename is the same as the original icon.



**Replacing ringtones:** You can replace ringtones by replacing the three audio files in the `Resources\\AudioFile` folder.

| Filename           | Description                      |
|--------------------|----------------------------------|
| phone_dialing.m4a  | The sound of making a call.      |
| phone_hangup.mp3   | The sound of being hung up.      |
| phone_ringing.flac | The ringtone for incoming calls. |

**Replacing text:** You can modify the strings on the video call UI by modifying the `CallingLocalized.strings` file in `zh-Hans.lproj` and `en.lproj`.

## Scheme 2. Custom UI Implementation

The entire call feature of `TUICallKit` is implemented based on the UI-less component `TUICallEngine`. You can delete the `tuicallkit` folder and implement your own UIs based entirely on `TUICallEngine`.

### TUICallEngine

`TUICallEngine` is the underlying API of the entire `TUICallKit` component. It provides key APIs such as APIs for making, answering, declining, and hanging up one-to-one audio/video and group calls and device operations.

| API                              | Description  |
|----------------------------------|--|
| <code>createInstance</code>      | Creates a <code>TUICallEngine</code> instance (singleton).           |
| <code>destroyInstance</code>     | Terminates a <code>TUICallEngine</code> instance (singleton).        |
| <code>init</code>                | Completes the authentication of basic audio/video call capabilities. |
| <code>addObserver</code>         | Registers an event listener.   |
| <code>removeObserver</code>      | Unregisters an event listener.                                       |
| <code>call</code>                | Makes a one-to-one call.   |
| <code>groupCall</code>           | Makes a group call.  |
| <code>accept</code>              | Answers a call.  |
| <code>reject</code>              | Declines a call.   |
| <code>hangup</code>              | Hangs up a call.   |
| <code>ignore</code>              | Ignores a call.  |
| <code>inviteUser</code>          | Invites a user during a group call.                                  |
| <code>joinInGroupCall</code>     | Joins the current group call actively.                               |
| <code>switchCallMediaType</code> | Switches the call media type, such as from video call to audio call. |
| <code>startRemoteView</code>     | Subscribes to the video stream of a remote user.                     |
| <code>stopRemoteView</code>      | Unsubscribes from the video stream of a remote user.                 |

|   |   |
|---|---|
| <a href="#">openCamera</a>                | Enables the camera.   |
| <a href="#">closeCamera</a>               | Disables the camera.  |
| <a href="#">switchCamera</a>              | Switches between the front and rear cameras.  |
| <a href="#">openMicrophone</a>            | Enables the mic.  |
| <a href="#">closeMicrophone</a>           | Disables the mic.   |
| <a href="#">selectAudioPlaybackDevice</a> | Selects the audio playback device (receiver/speaker).   |
| <a href="#">setSelfInfo</a>               | Sets the user nickname and profile photo.   |
| <a href="#">enableMultiDeviceAbility</a>  | Enables/Disables the multi-device login mode of <code>TUICallEngine</code> (supported by the premium plan). |

## TUICallObserver

`TUICallObserver` is the callback even class of `TUICallEngine`. You can use it to listen on the desired callback events.

| API                                    | Description                         |
|--|-------------------------------------|
| <a href="#">onError</a>                | An error occurred during the call.  |
| <a href="#">onCallReceived</a>         | A call was received.                |
| <a href="#">onCallCancelled</a>        | The call was canceled.              |
| <a href="#">onCallBegin</a>            | The call was connected.             |
| <a href="#">onCallEnd</a>              | The call ended.                     |
| <a href="#">onCallMediaTypeChanged</a> | The call media type changed.        |
| <a href="#">onUserReject</a>           | A user declined the call.           |
| <a href="#">onUserNoResponse</a>       | A user didn't respond.              |
| <a href="#">onUserLineBusy</a>         | A user was busy.                    |
| <a href="#">onUserJoin</a>             | A user joined the call.             |
| <a href="#">onUserLeave</a>            | A user left the call.               |
| <a href="#">onUserVideoAvailable</a>   | Whether a user had a video stream.  |
| <a href="#">onUserAudioAvailable</a>   | Whether a user had an audio stream. |

|   |                                   |
|---|-----------------------------------|
| <a href="#">onUserVoiceVolumeChanged</a>    | The volume levels of all users.   |
| <a href="#">onUserNetworkQualityChanged</a> | The network quality of all users. |

## Definitions of key classes

| API                                    | Description  |
|--|--|
| <a href="#">TUICallMediaType</a>       | The call media type. Enumeration: Video call and audio call.       |
| <a href="#">TUICallRole</a>            | The call role. Enumeration: Caller and callee.                     |
| <a href="#">TUICallStatus</a>          | The call status. Enumeration: Idle, waiting, and answering.        |
| <a href="#">TUIRoomId</a>              | The audio/video room ID, which can be a number or string.          |
| <a href="#">TUICamera</a>              | The camera type. Enumeration: Front camera and rear camera.        |
| <a href="#">TUIAudioPlaybackDevice</a> | The audio playback device type. Enumeration: Speaker and receiver. |
| <a href="#">TUINetworkQualityInfo</a>  | The information of the current network quality.                    |

# Web

Last updated : 2023-09-19 14:47:18

## TUICallKit UI Customization Guide

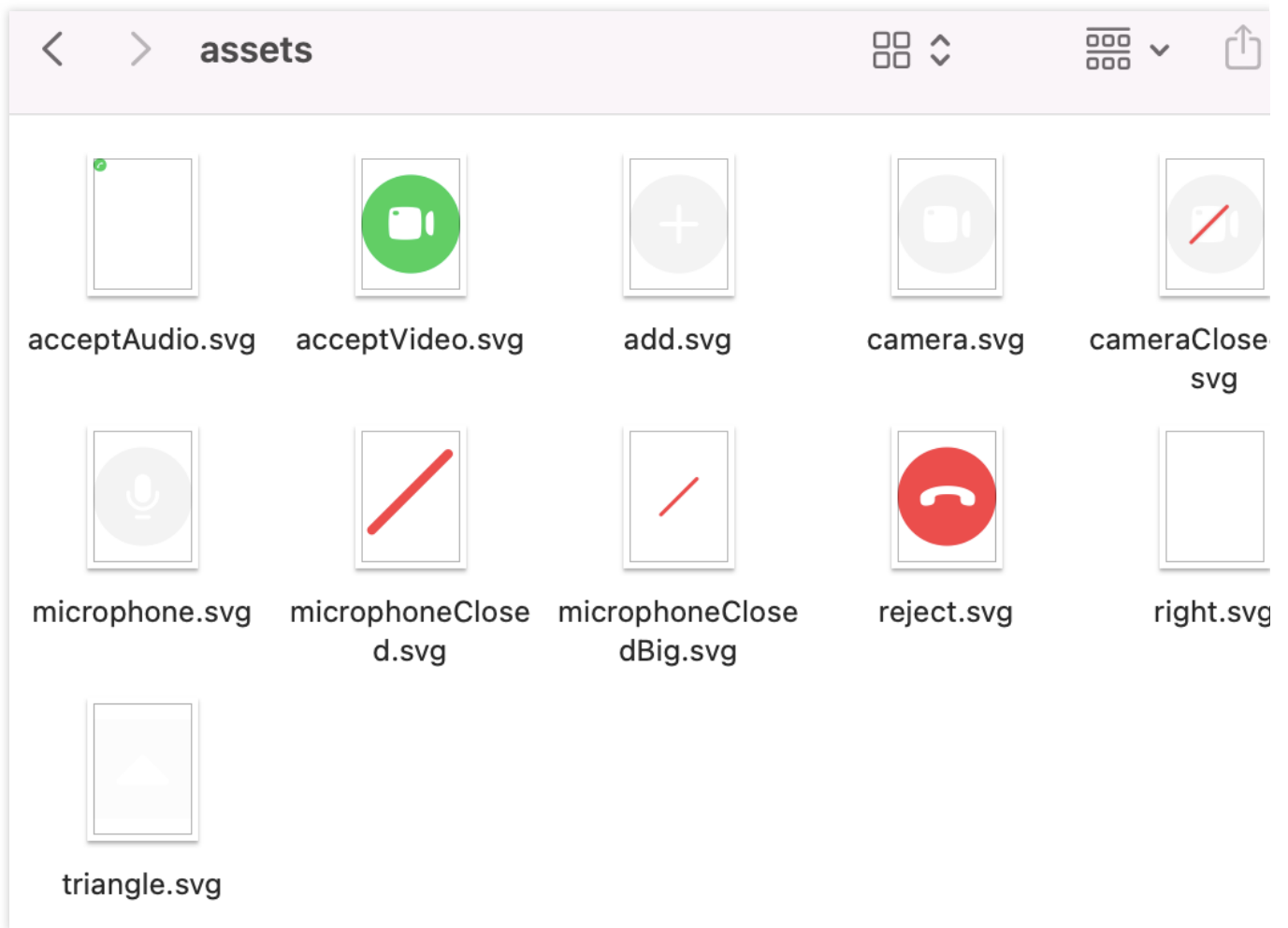
This document describes how to customize the UI of `TUICallKit` . We provide two schemes. You can either make minor modifications to the UI source code we provide or implement your own UI.

### Scheme 1: Using our UI source code

You can modify the UI source code to make minor changes to the ready-made UI we provide. The UI source code is in the `Web/` directory of the [TUICallKit project](#).

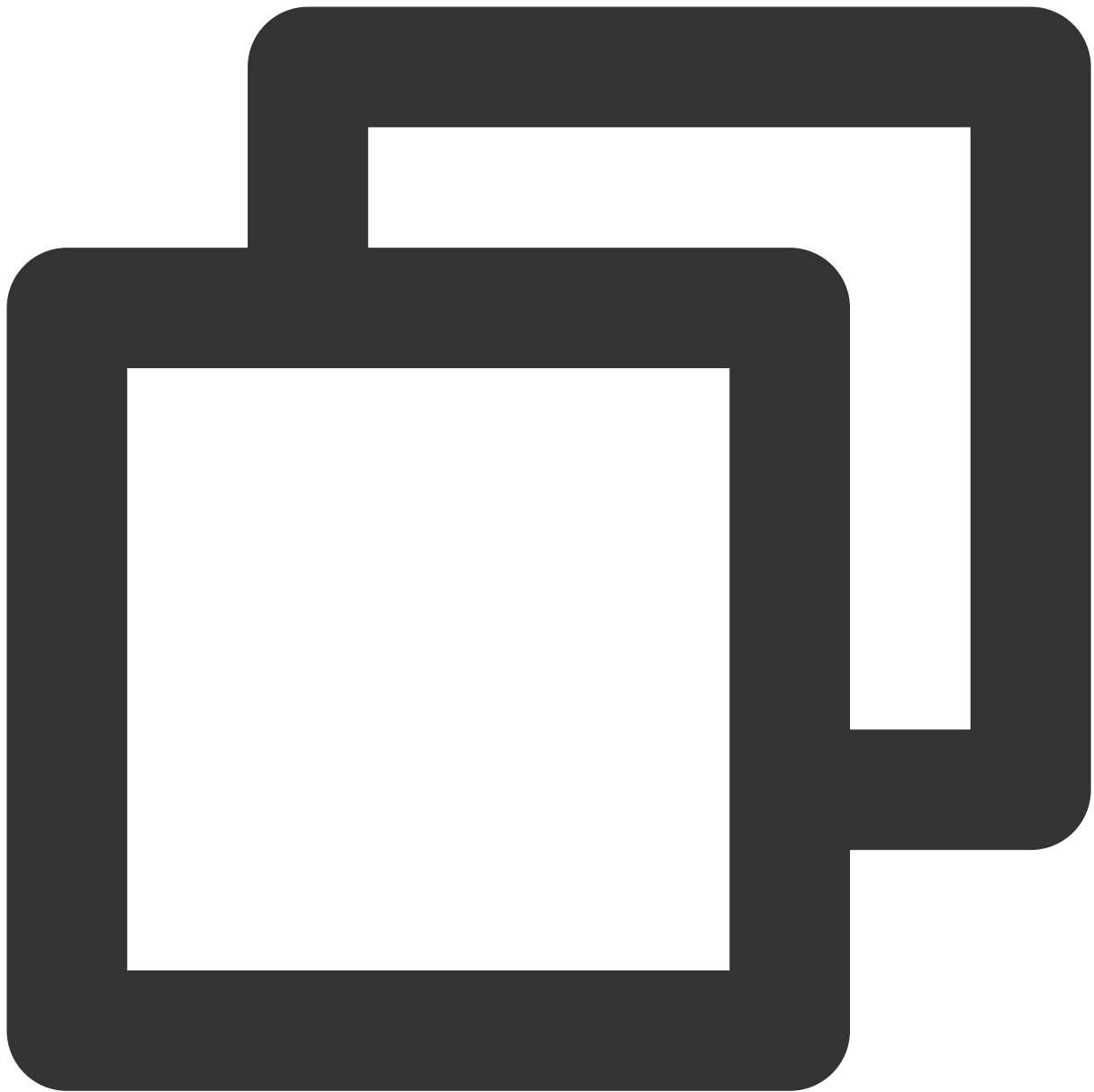
#### Replacing icons

You can replace the icons in the `src/icons` folder to customize the color tone and style of the icons in your application. Make sure you do not change the filenames. To preview icons, go to `src/assets` .



## Changing the layout

You can modify the views in the `src/components/` folder to change the call UI.



- |                        |   |
|------------------------|---|
| - components/          |   |
| - Calling-C2CVideo.vue | One-to-one video call                           |
| - Calling-Group.vue    | Group audio/video call                          |
| - ControlPanel.vue     | The control panel                               |
| - ControlPanelItem.vue | Control panel items                             |
| - Dialing.vue          | The call making UI, incoming call UI, and one-t |
| - MicrophoneIcon.vue   | The mic icon that can display changes in the vo |
| - TUICallKit.vue       | The overall `TUICallKit` component              |

## Changing the style



The style file is `src/style.css` . You can adjust it to implement your desired UI style.

## Scheme 2: Implementing your own UI

The features of `TUICallKit` are implemented based on the `TUICallEngine` SDK, which does not include UI elements. You can use `TUICallEngine` to implement your own UI. For detailed directions, refer to the documents below:

[TUICallEngine integration guide](#)

[TUICallEngine APIs](#)

# Flutter

Last updated : 2024-03-12 14:51:07

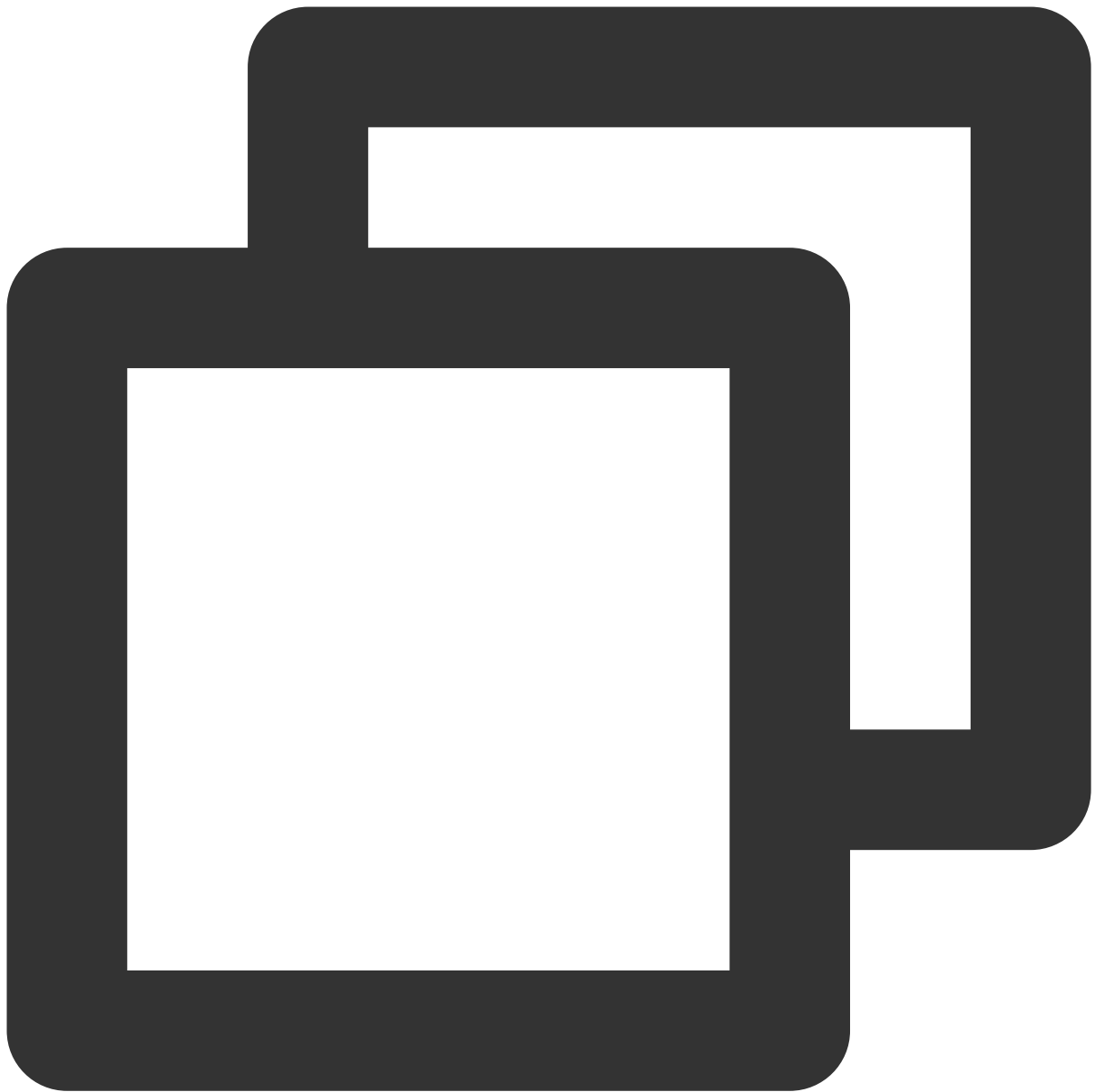
This article will introduce how to customize the user interface of TUICallKit. We provide two solutions for you to choose: **interface fine-tuning solution** and **self-implementation UI** solution.

Note: The page customization solution needs to use the [tencent\\_calls\\_uikit](#) plugin version 1.8.0 or later.

## Scheme 1. Slight UI Adjustment

You can download the latest version of the [tencent\\_calls\\_uikit](#) plugin locally, and then use the local dependency method to access the plugin in your project. The local dependency method is as follows:

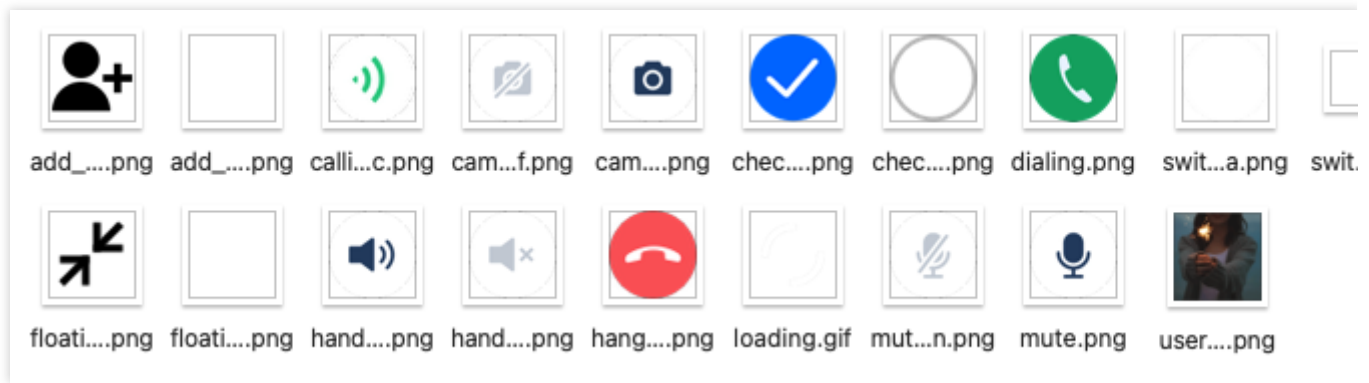
Under the **dependencies** node in the project **pubspec.yaml** file, add the **tencent\_calls\_uikit** plugin dependency, as shown below:



```
dependencies:  
  tencent_calls_uikit:  
    path: your file path
```

## replace icon

You can directly replace the icons under the **assets\\images** folder to ensure that the color tone of the icons in the entire app is consistent. Please keep the name of the icon file unchanged when replacing.



## replace ringtone

You can replace the three audio files in the **assets\audios** folder to achieve the purpose of replacing the ringtone:

| Name              | Purpose                         |
|-------------------|---------------------------------|
| phone_dialing.mp3 | The sound of making a call      |
| phone_hangup.mp3  | The sound of being hung up      |
| phone_ringing.mp3 | The ringtone for incoming calls |

## Replacing text

You can modify the string content in the video call interface by modifying the strings in the **strings.g.dart** file in the **lib\src\i18n** directory.

## Scheme 2. Custom UI Implementation

The entire call feature of `TUICallKit` is implemented based on the UI-less component `TUICallEngine`. You can delete the `tuicallkit` folder and implement your own UIs based entirely on `TUICallEngine`.

### TUICallEngine

`TUICallEngine` is the underlying API of the entire `TUICallKit` component. It provides key APIs such as APIs for making, answering, declining, and hanging up one-to-one audio/video and group calls and device operations.

### TUICallObserver

`TUICallObserver` is the callback even class of `TUICallEngine`. You can use it to listen on the desired callback events.

# Offline Call Push (TUICallKit)

## iOS

## VoIP

Last updated : 2024-01-17 11:58:53

VoIP (Voice over IP) Push is a notification mechanism provided by Apple for responding to VoIP calls. Combining Apple's PushKit.framework and CallKit.framework can achieve system-level call effects. Currently, VoIP Push only supports push notifications when offline.

Note :

Apple requires that VoIP Push be used in conjunction with the CallKit.framework starting with iOS 13.0; otherwise, the app will crash after running.

VoIP Push cannot reuse the general APNs push certificates, so a separate [VoIP Push certificate needs to be applied for](#) on the Apple Developer website.

## Configuring VoIP Push

To receive VoIP Push notifications, follow these steps:


1. Apply for a VoIP Push certificate.
2. Upload the certificate to the IM console.
3. Complete the project configuration.
4. Integrate the TUICallKitVoIPExtension component.

### Step 1: Apply for a VoIP Push certificate

Before applying for a VoIP Push certificate, log in to the [Apple Developer Center](#), [enable your app's remote push capabilities](#).


Once your AppID has Push Notification capabilities, follow these steps to apply for and configure a VoIP Push certificate:


1. Log in to the [Apple Developer Center](#) website, click "Certificates, Identifiers & Profiles" in the sidebar, and go to the "Certificates, IDS & Profiles" page.


 **Developer**


Account


Program Resources


 **Overview**

 Membership


 **Certificates, IDs & Profiles**


 App Store Connect

 CloudKit Dashboard


 Code-Level Support

Additional Resources


 Documentation



Apple Developer Pr



**Certificates, Identifiers & Profiles**  
Manage the certificates, identifiers, and devices you need to develop your app.



**App Store Connect**  
Publish and manage your app on the App Store Connect.

2. Click "+" next to "Certificates".



## Certificates

NAME 

TYPE

## PLATFORM

CREATED BY

\_\_\_\_\_

© 2006 The Authors  
Journal compilation © 2006 Blackwell Publishing Ltd

\_\_\_\_\_

\_\_\_\_\_

100

\_\_\_\_\_

© 2006 The Authors  
Journal compilation © 2006 Blackwell Publishing Ltd



\_\_\_\_\_

\_\_\_\_\_

Page 121 of 2527



# Certificates, Identifiers & Profiles

[< All Certificates](#)

## Create a New Certificate

### Services

- ☐ **Apple Push Notification service SSL (Sandbox)**  
Establish connectivity between your notification server and the Apple Push Notification service sandbox environment to deliver remote notifications to your app. A separate certificate is required for each app you develop.
- ☐ **Apple Push Notification service SSL (Sandbox & Production)**  
Establish connectivity between your notification server, the Apple Push Notification service sandbox, and production environments to deliver remote notifications to your app. When utilizing HTTP/2, the same certificate can be used to deliver app notifications, update ClockKit complication data, and alert background VoIP apps of incoming activity. A separate certificate is required for each app you distribute.
- ☐ **Pass Type ID Certificate**  
Sign and send updates to passes in Wallet.
- ☐ **Order Type ID Certificate**  
Sign and send updates to order information payloads displayed in Apple Wallet.
- ☐ **Website Push ID Certificate**  
Sign and send updates for Websites.
- ☐ **Swift Package Collection Certificate**  
Sign Swift Package Collections for distribution.
- ☐ **WatchKit Services Certificate**  
Establish connectivity between your notification server, the Apple Push Notification service sandbox, and production environment to update ClockKit complication data. When utilizing HTTP/2, the same certificate can be used to deliver app notifications, update ClockKit complication data, and alert background VoIP apps of incoming activity. A separate certificate is required for each app you distribute.
- ☒ **VoIP Services Certificate**  
Establish connectivity between your notification server, the Apple Push Notification service sandbox, and production environment to alert background VoIP apps of incoming activity. A separate certificate is required for each app you distribute.
- ☐ **Apple Pay Payment Processing Certificate**  
Decrypt app transaction data sent by Apple to a merchant/developer.

1

4. In the "Select an App ID for your VoIP Service Certificate" tab, select your app's BundleID, and click "Continue".



Apple Developer

## Certificates, Identifiers & Profiles

[← All Certificates](#)

### Create a new VoIP Services Certificate

#### Select an App ID for your VoIP Service Certificate

Each app you want to use with VoIP Services requires its own individual VoIP Services certificate. The App ID-specific VoIP Services certificate allows your notification server to connect to the VoIP Service. Note that only explicit App IDs with a specific Bundle Identifier can be used to create a VoIP Service Certificate.

App ID:

12 App IDs

x | v

5. The system will prompt you for a Certificate Signing Request (CSR).

Apple Developer

## Certificates, Identifiers & Profiles

[← All Certificates](#)

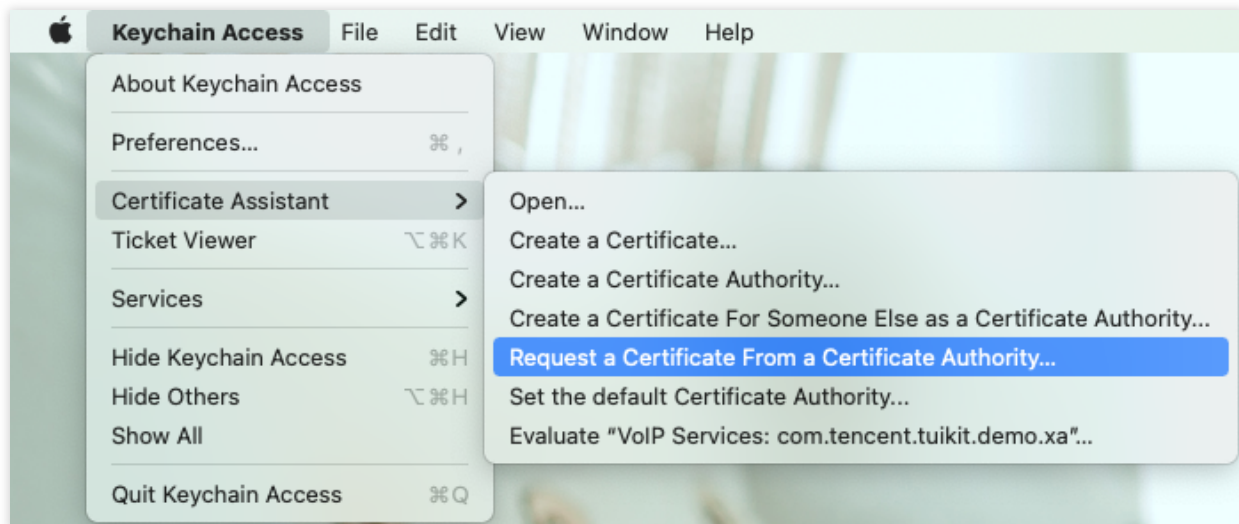
### Create a new VoIP Services Certificate

#### Upload a Certificate Signing Request

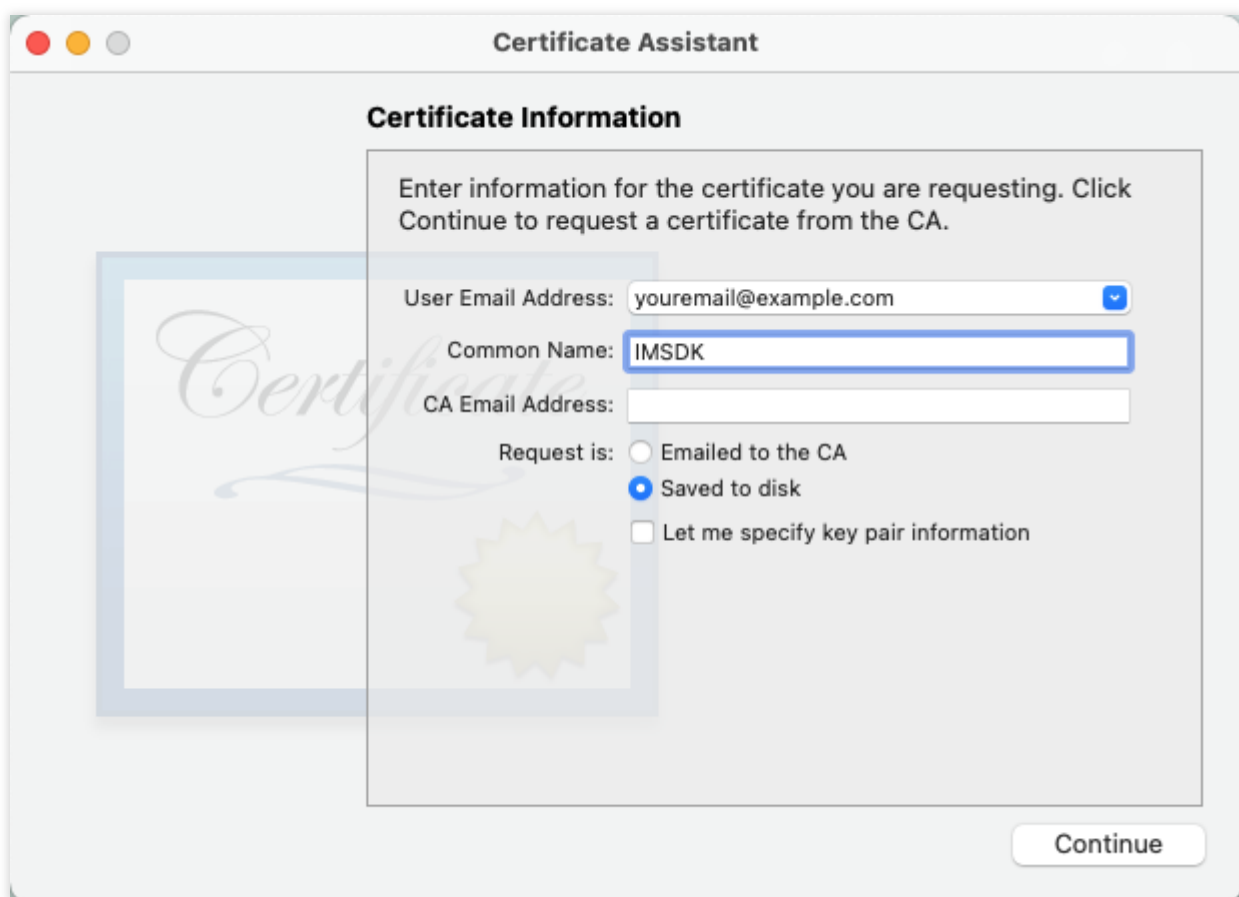
To manually generate a Certificate, you need a **Certificate Signing Request (CSR) file** from your Mac. [Learn more >](#)

[Choose File](#)

6. Next, create a CSR file. Open **Keychain Access** on your Mac, and in the menu, choose **Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority**.



7. Enter your email address, common name (your name or company name), choose **Save to disk**, click "Continue", and the system will generate a \*.certSigningRequest file.



Go back to the Apple Developer website in **Step5** , click "Choose File" and upload the generated \*.certSigningRequest file.

Apple Developer

## Certificates, Identifiers & Profiles

[← All Certificates](#)

### Create a new VoIP Services Certificate

Upload a Certificate Signing Request

To manually generate a Certificate, you need a **Certificate Signing Request (CSR) file** from your Mac. [Learn more >](#)[Choose File](#)

8. Click **Continue** to generate the certificate, and click **Download** to download the corresponding certificate to your local device.

Apple Developer

## Certificates, Identifiers & Profiles

[← All Certificates](#)

### Download Your Certificate

#### Certificate Details

Certificate Name

Certificate Type

VoIP Services

Download your certificate  
Keychain Access. Make s  
keys somewhere secure.

Expiration Date

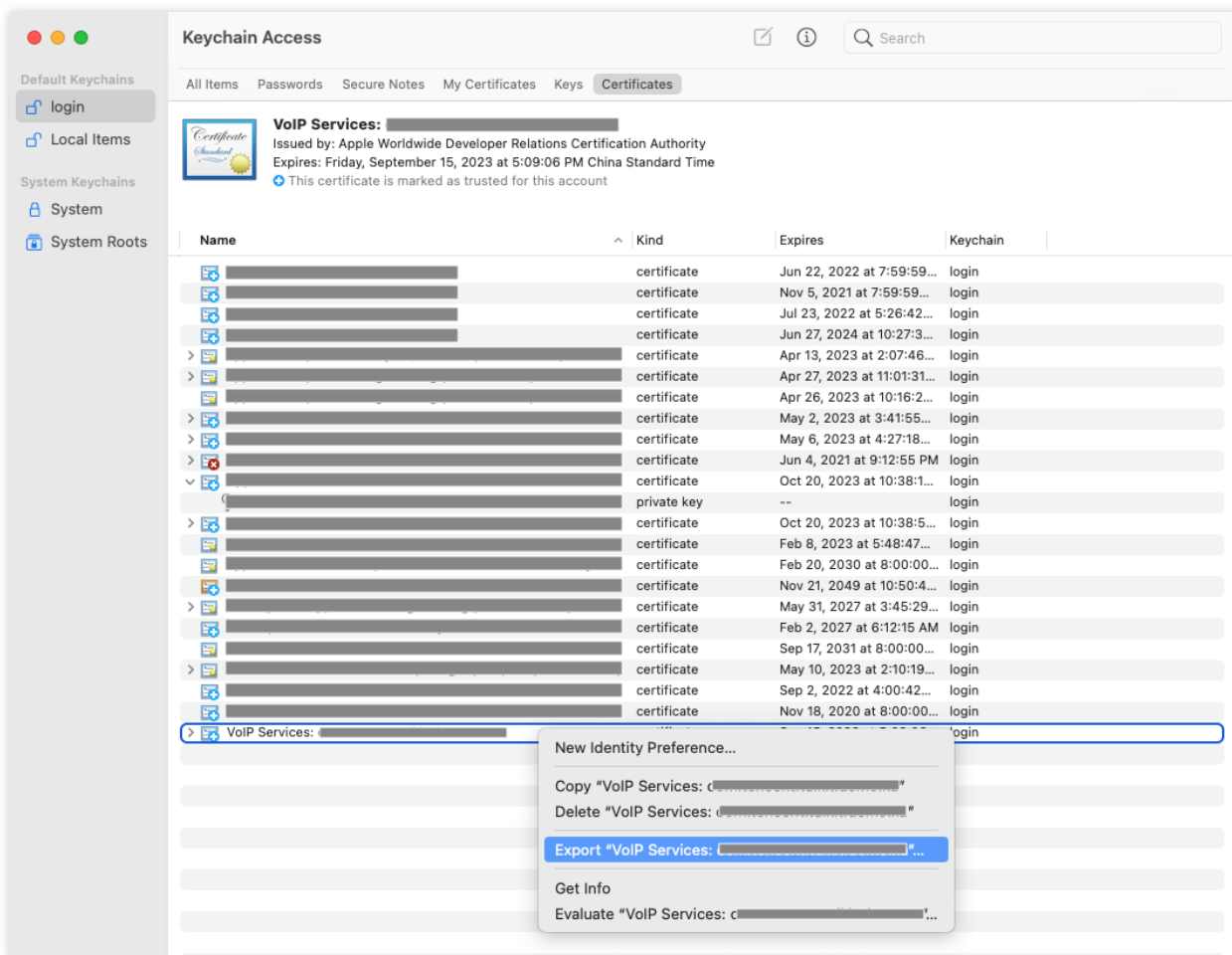
Created By

9. Double-click the downloaded `voip_services.cer` file, and the system will import it into your keychain.

10. Open the Keychain app, under **Login > My Certificates**, right-click on the created VoIP Services certificate.

**Note :**

When saving the `P12` file, be sure to set a password for it.



## Step 2: Upload certificates to the IM Console

Open the [IM Console](#), select your created IM application, and follow the steps below to upload your certificates:

1. Choose your IM application, click **Go new** under the **Offline Push Certificate Configuration** tab.

The screenshot displays the Tencent Real-Time Communication console interface. The left sidebar shows the navigation menu with 'Application management' selected. The main content area is titled 'Basic Configuration' for application '20006172 - TUICallKit'. The 'Standard Billing Plan' section shows the plan as 'TRTC Developer' with an expiration time of '2024-02-04'. The 'App Information' section lists the SDKAppID as '20006172' and the Application Name as 'TUICallKit'. The 'Basic info' section shows the Secret Key as '\*\*\*\*\*' and the Creation time as '2024-01-04'. The 'Tag Configuration' section is at the bottom. On the right sidebar, the 'Offline Push Certificate Configuration' section is highlighted with a red box, containing a 'Go now' button. Below it, the 'RTC Engine' and 'Call' sections are visible.

**Standard Billing Plan**

Status: In use ⓘ

Plan: TRTC Developer

Expiration time: 2024-02-04

[Upgrade Plan](#) [More](#)

**App Information** [Edit](#)

SDKAppID: 20006172 ⓘ

Application Name: TUICallKit

Application Type: -

Application Introduction: -

**Basic info**

Secret Key: \*\*\*\*\* [Display key](#)

Key information is sensitive. Keep it confidential and do not disclose it.

Creation time: 2024-01-04

Last Modified: 2024-01-04

**Tag Configuration** [Edit](#)

TagKey TagValue

**Offline Push Certificate Configuration**

The push configuration module has the left [Go now](#) ⓘ

**RTC Engine**

RTC Engine can help you implement functions in IM applications. RTC Engine distinguishes between data centers.

**Call**

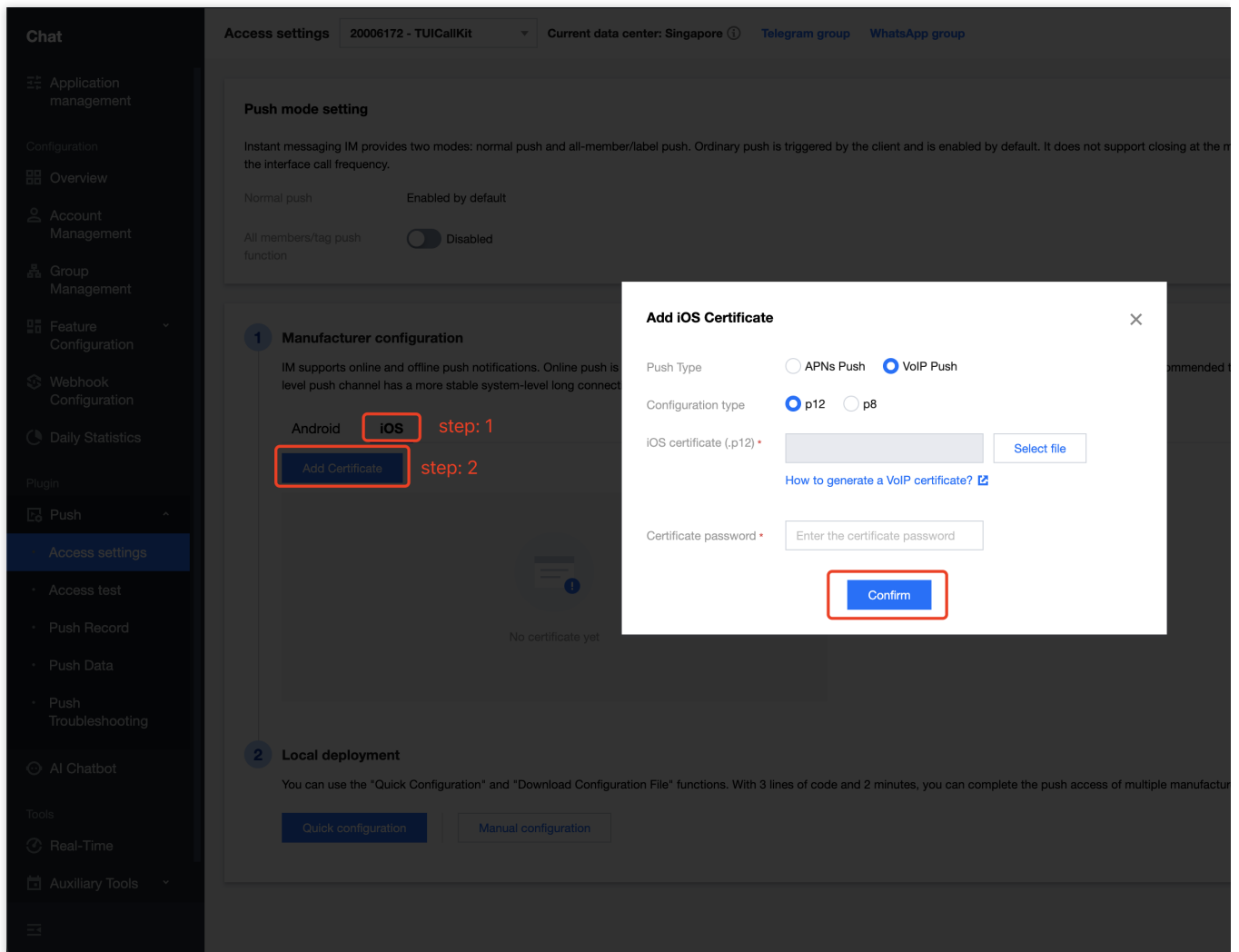
TRTC Call is a call component that and multi-platform call making/ans

Status: Activated

Current version: Trail [Learn more](#)

Expiration time: 2024-01-11

2. In the **Manufacturer configuration**, switch to **iOS**, click the **Add Certificate** button. Then upload the VoIP certificates for both the production and development environments.

**Note :**

The VoIP Push Certificate itself does not distinguish between production and testing environments. Both environments use the same VoIP Push Certificate.

The uploaded certificate name should be in English (especially avoiding brackets and other special characters).

The uploaded certificate must have a password set, or the push notification won't be received.

The certificate for publishing to the App Store needs to be set to the production environment; otherwise, the push notification won't be received.

The p12 certificate you upload must be a valid and genuinely applied certificate.

3. After the upload is completed, record the certificate ID for different environments.

## 1 Manufacturer configuration

IM supports online and offline push notifications. Online push is delivered through the instant messaging IM channel, which is fast and reliable; for offline push, it is recommended to use the system-level push channel. The system-level push channel has a more stable system-level long connection, and the resource consumption is greatly reduced.

Android iOS

Add Certificate

ID: 15853

Delete Edit

Certificate information

Push Type

Certificate Type

mutable-content

Certificate password

Expiration time

VoIP Push

Production environment

Enabled

123321

2024-03-19 15:29:01

.p12

ID: 15853

Certificate information

Push Type

Certificate Type

mutable-content

Certificate password

Expiration time

VoIP Push

Development Environment

Enabled

123321

2024-03-19 15:29:01

## 2 Local deployment

You can use the "Quick Configuration" and "Download Configuration File" functions. With 3 lines of code and 2 minutes, you can complete the push access of multiple projects.

Quick configuration

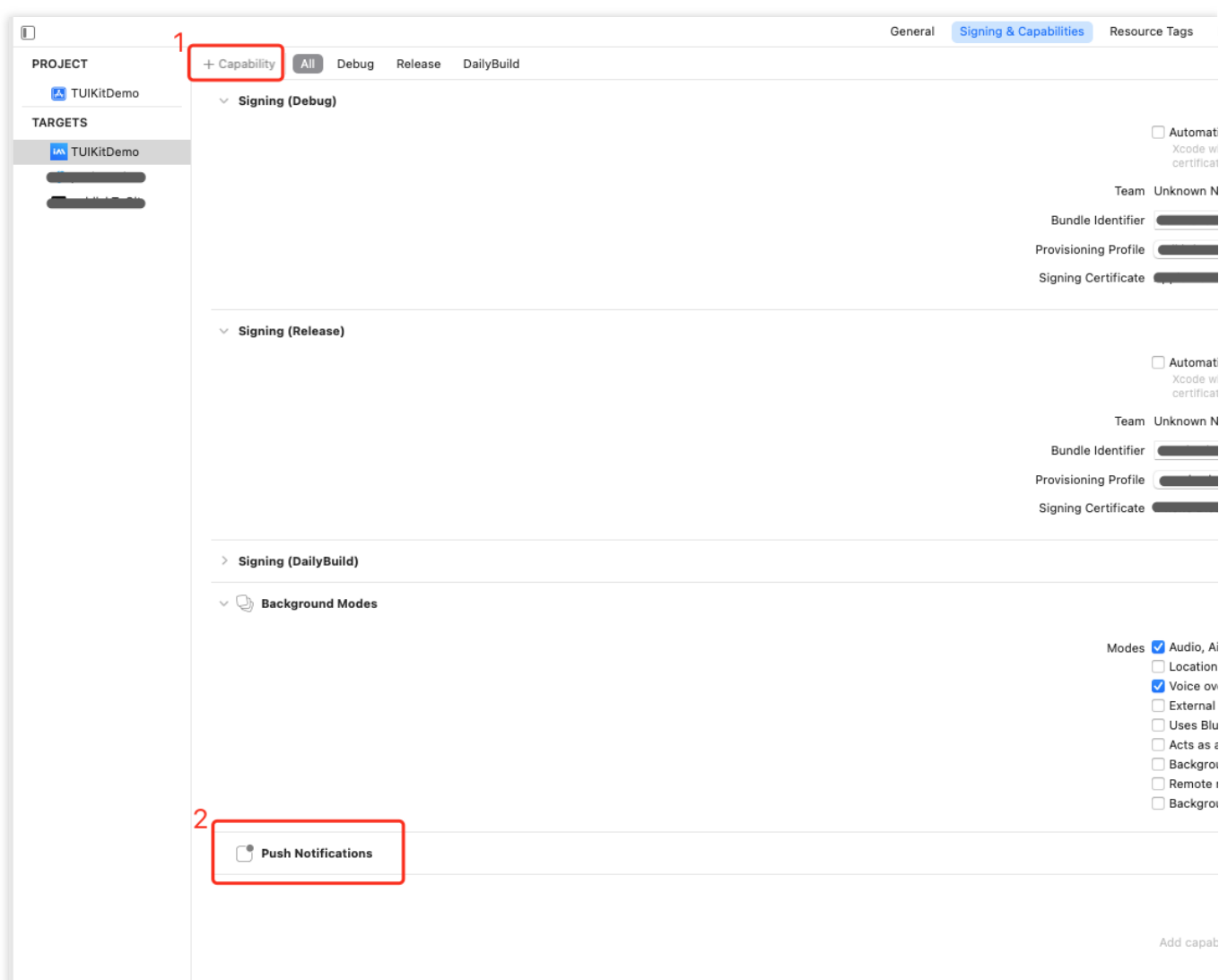
Manual configuration

Note :

The certificate IDs for the development and production environments must be strictly distinguished and filled in according to the actual environment in [Step 4: Integrate the TUICallKitVoIPExtension component.](#)

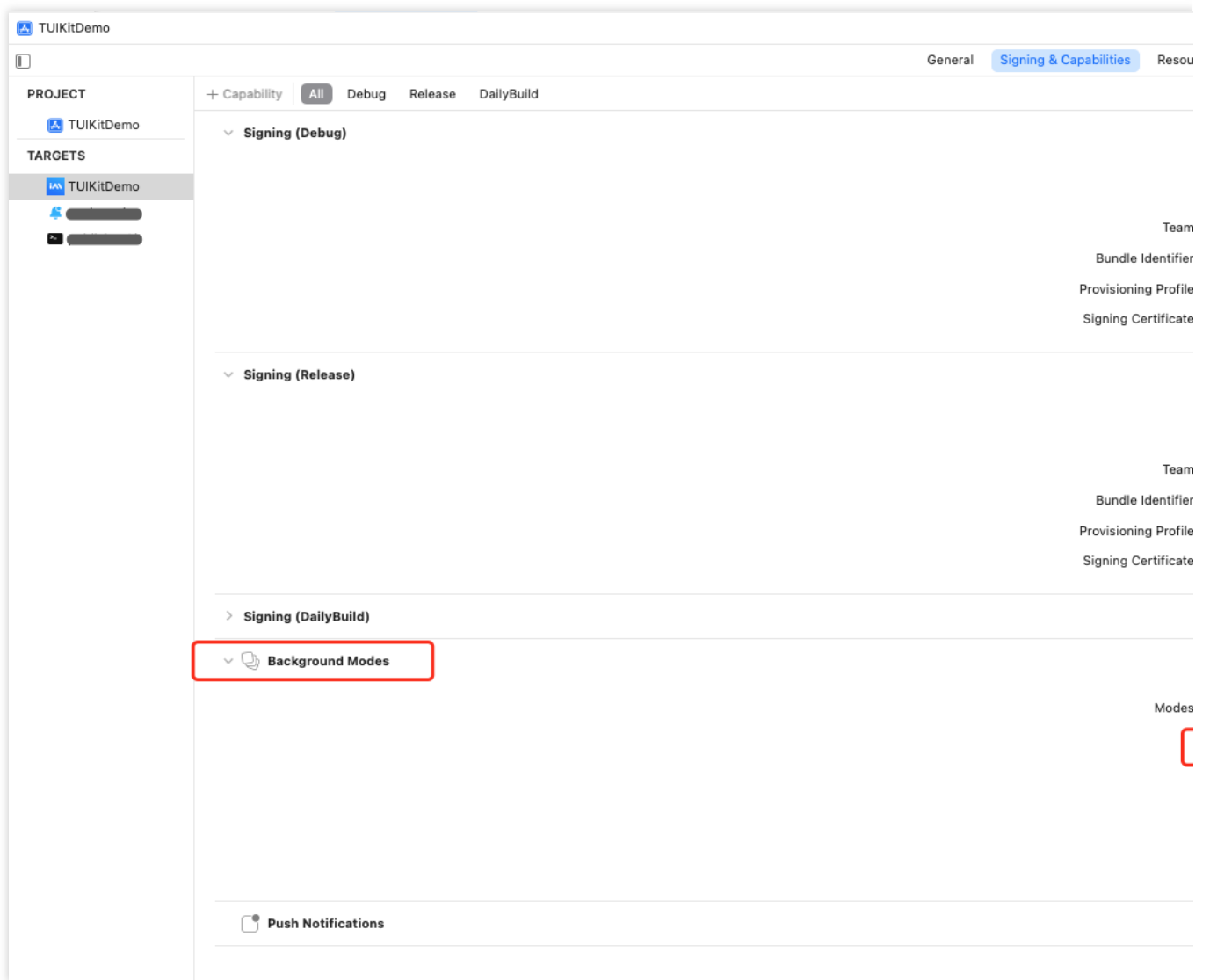
## Step 3: Complete the project configuration

1. As shown below, confirm whether the Push Notification capability has been added to your project's capabilities.



2. As shown below, please check if the Voice over IP option is enabled under Background Modes in your project's capabilities.

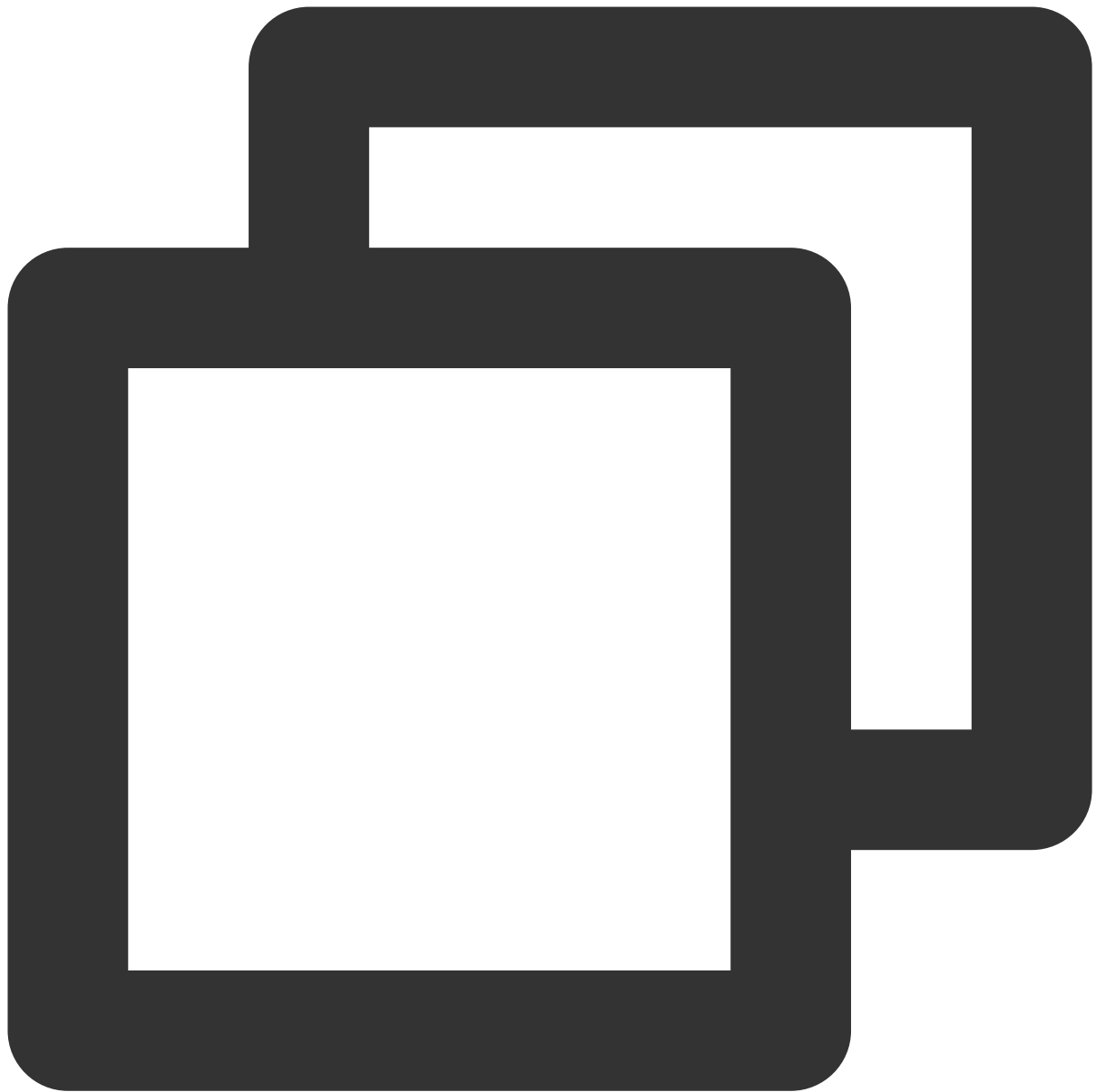




## Step 4: Integrate the TUICallKitVoIPExtension component

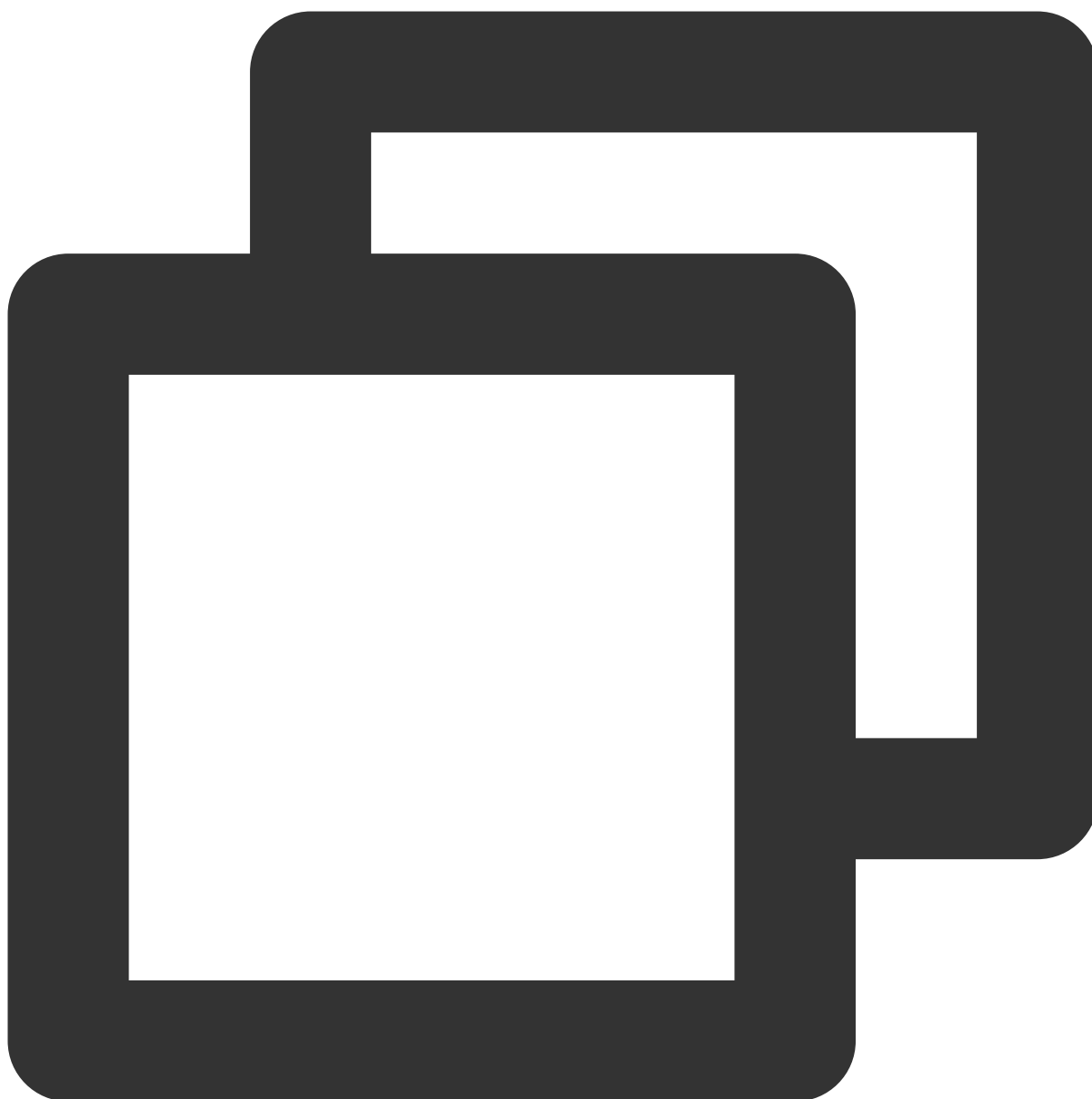
Use CocoaPods to import the component, following these steps:

1. Add the following dependency to your `Podfile` .



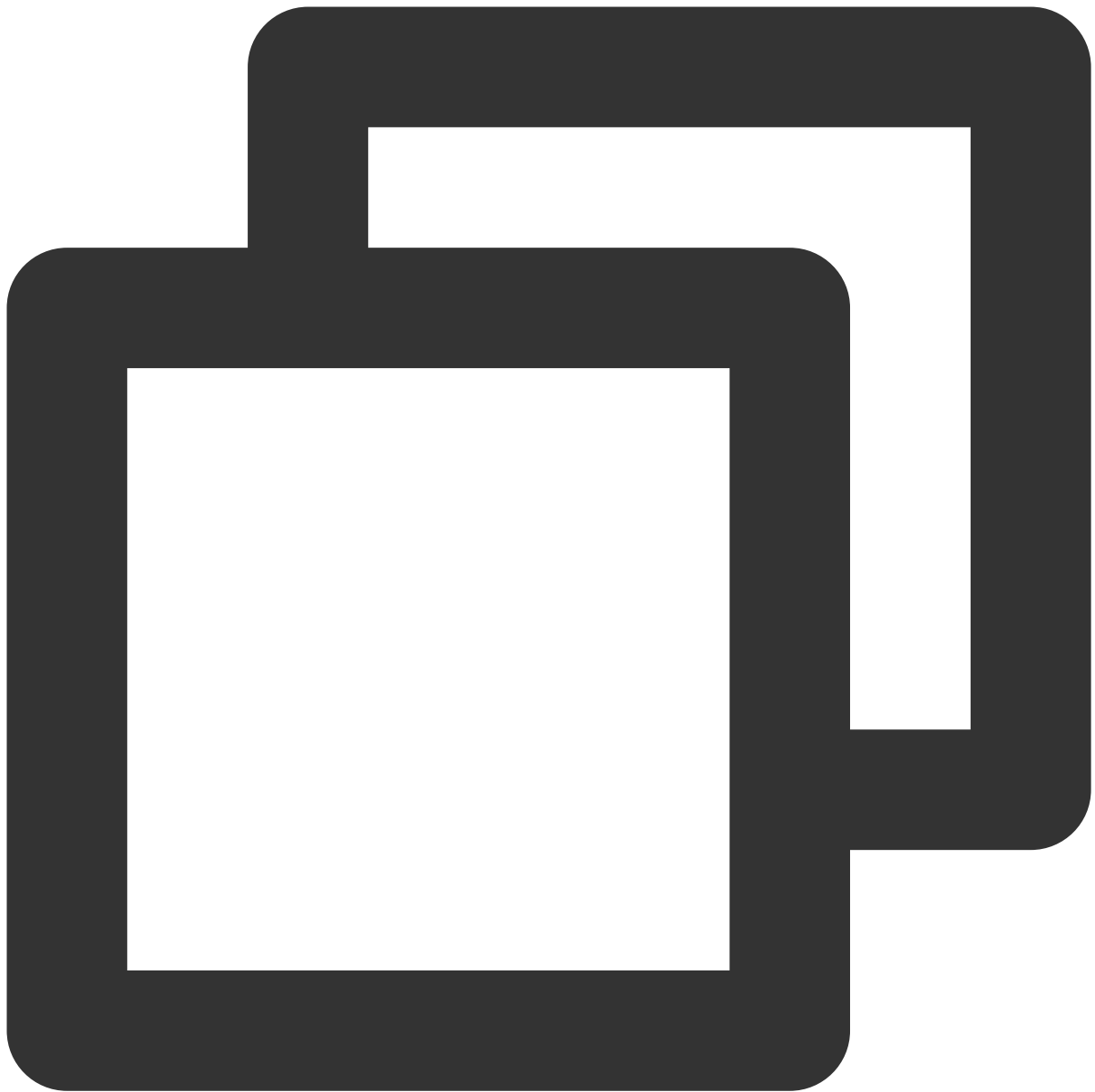
```
pod 'TUICallKitVoIPExtension'
```

2. Execute the command below to install the component.



```
pod install
```

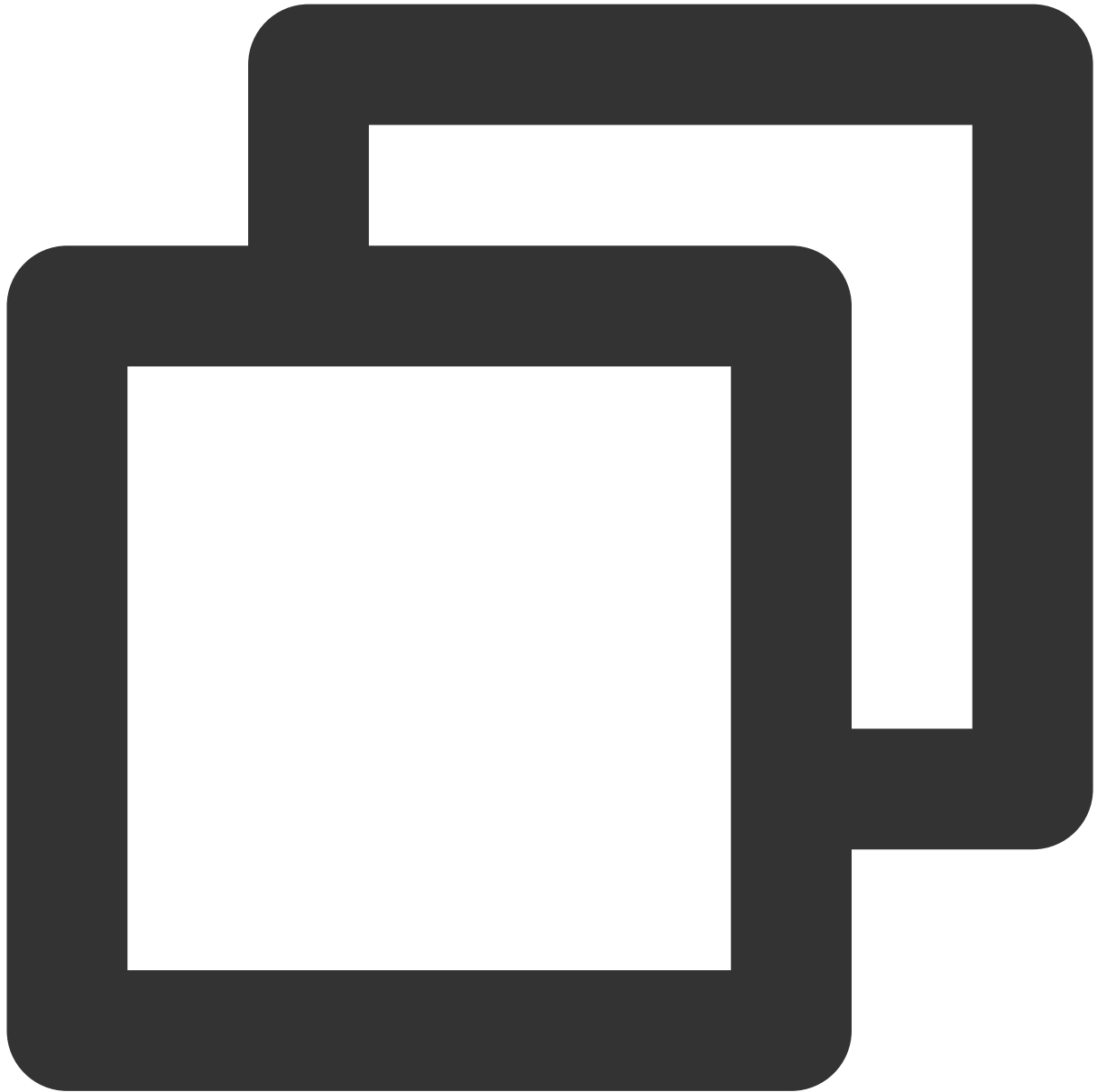
3. Report the [Certificate ID recorded in Step 2.](#)



```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) throws NSError {  
    // Override point for customization after application launch.  
    // Report Certificate ID  
    TUICallKitVoIPExtension.setCertificateID(CertificateID)  
    return true  
}
```

**Note :**

If you cannot install the latest version of TUICallKit, execute the command below to update your local CocoaPods repository list.



```
pod repo update
```

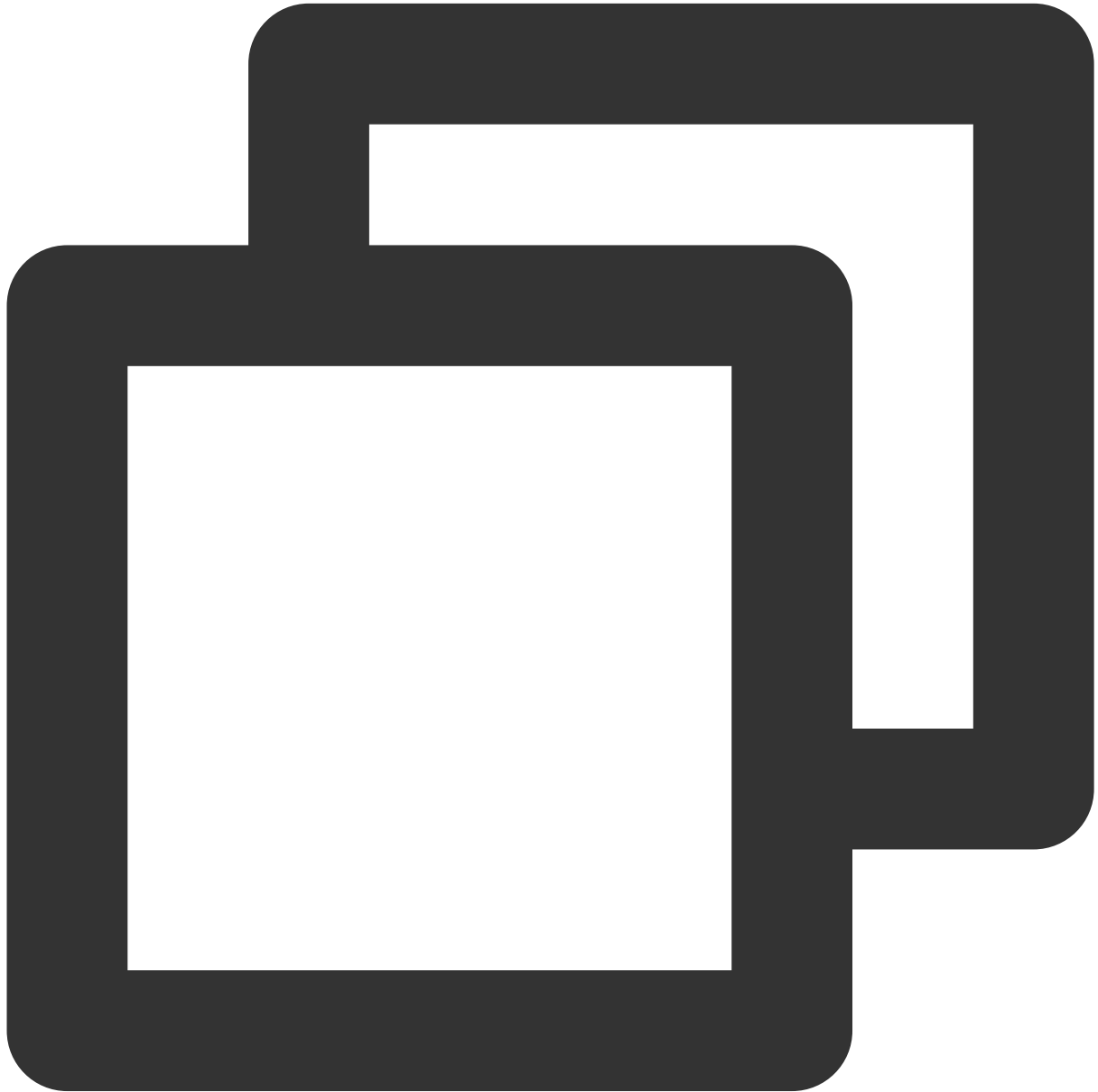
## Make a VoIP call

If you need to make a VoIP call, you need to set the `iOSPushType` field of the `OfflinePushInfo` to `TUICallIOSOfflinePushTypeVoIP` when calling [call](#). The default value is `TUICallIOSOfflinePushTypeAPNs`.

Objective-C

Java

Dart

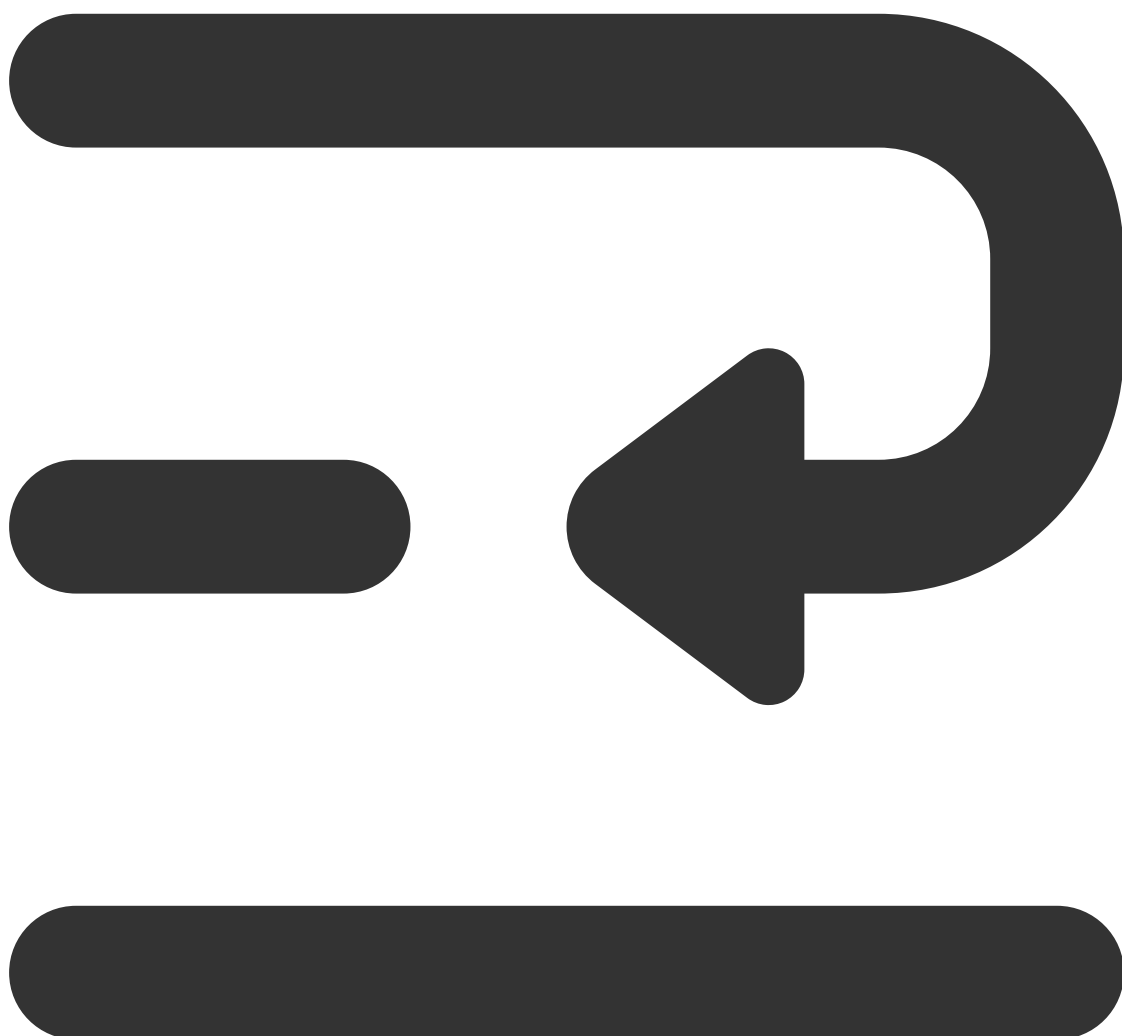


```
- (TUICallParams *)getCallParams {  
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];  
    TUICallParams *callParams = [TUICallParams new];
```

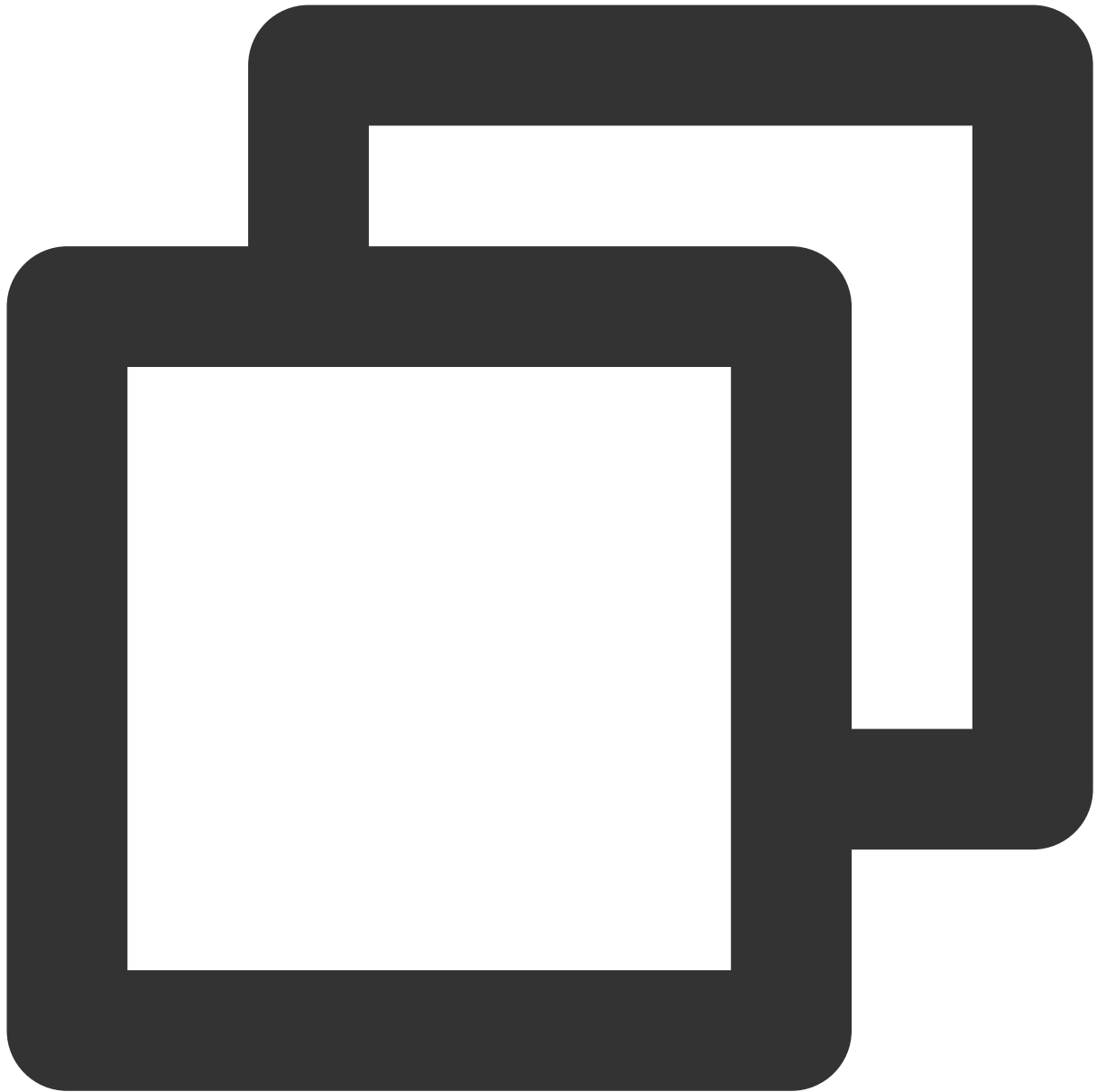
```
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}

- (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = @"You have a new call";
    // iOS push type: if you want user VoIP, please modify type to TUICallIOSOfflinePushTypeVoIP;
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeVoIP;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelID";
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery rate);
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}

[[TUICallKit sharedInstance] call:@"Mike's ID" params:[self getCallParams] callMediaType:0];
```







```
TUICallDefine.OfflinePushInfo offlinePushInfo = new TUICallDefine.OfflinePushInfo()  
offlinePushInfo.setTitle("");  
offlinePushInfo.setDesc("You have receive a new call");  
// OPPO must set a ChannelID to receive push messages. This channelId needs to be t  
offlinePushInfo.setAndroidOPPOChannelID("tuikit");  
offlinePushInfo.setIgnoreIOSBadge(false);  
offlinePushInfo.setIOSSound("phone_ringing.mp3");  
offlinePushInfo.setAndroidSound("phone_ringing"); // Note: don't add suffix  
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)  
offlinePushInfo.setAndroidVIVOClassification(1);  
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
```

```
offlinePushInfo.setAndroidFCMChannelID("fcm_push_channel");  
// Huawei message type  
offlinePushInfo.setAndroidHuaWeiCategory("IM");  
// IOS push type: if you want user VoIP, please modify type to TUICallDefine.IOSOff  
offlinePushInfo.setIOSPushType(TUICallDefine.IOSOfflinePushType.VoIP);  
  
TUICallDefine.CallParams params = new TUICallDefine.CallParams();  
params.offlinePushInfo = offlinePushInfo;  
  
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Video, para
```



```
TUIOfflinePushInfo offlinePushInfo = TUIOfflinePushInfo();
offlinePushInfo.title = "Flutter TUICallKit";
offlinePushInfo.desc = "This is an incoming call from Flutter TUICallkit";
offlinePushInfo.ignoreIOSBadge = false;
offlinePushInfo.iOSSound = "phone_ringing.mp3";
offlinePushInfo.androidSound = "phone_ringing";
offlinePushInfo.androidOPPOChannelID = "Flutter TUICallKit";
offlinePushInfo.androidVIVOClassification = 1;
offlinePushInfo.androidFCMChannelID = "fcm_push_channel";
offlinePushInfo.androidHuaWeiCategory = "Flutter TUICallKit";
offlinePushInfo.iOSPushType = TUICallIOSOfflinePushType.VoIP;
```

```
TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);  
  
TUICallKit.instance.call(callUserId, TUICallMediaType.audio, params);
```

## Call from the system call log

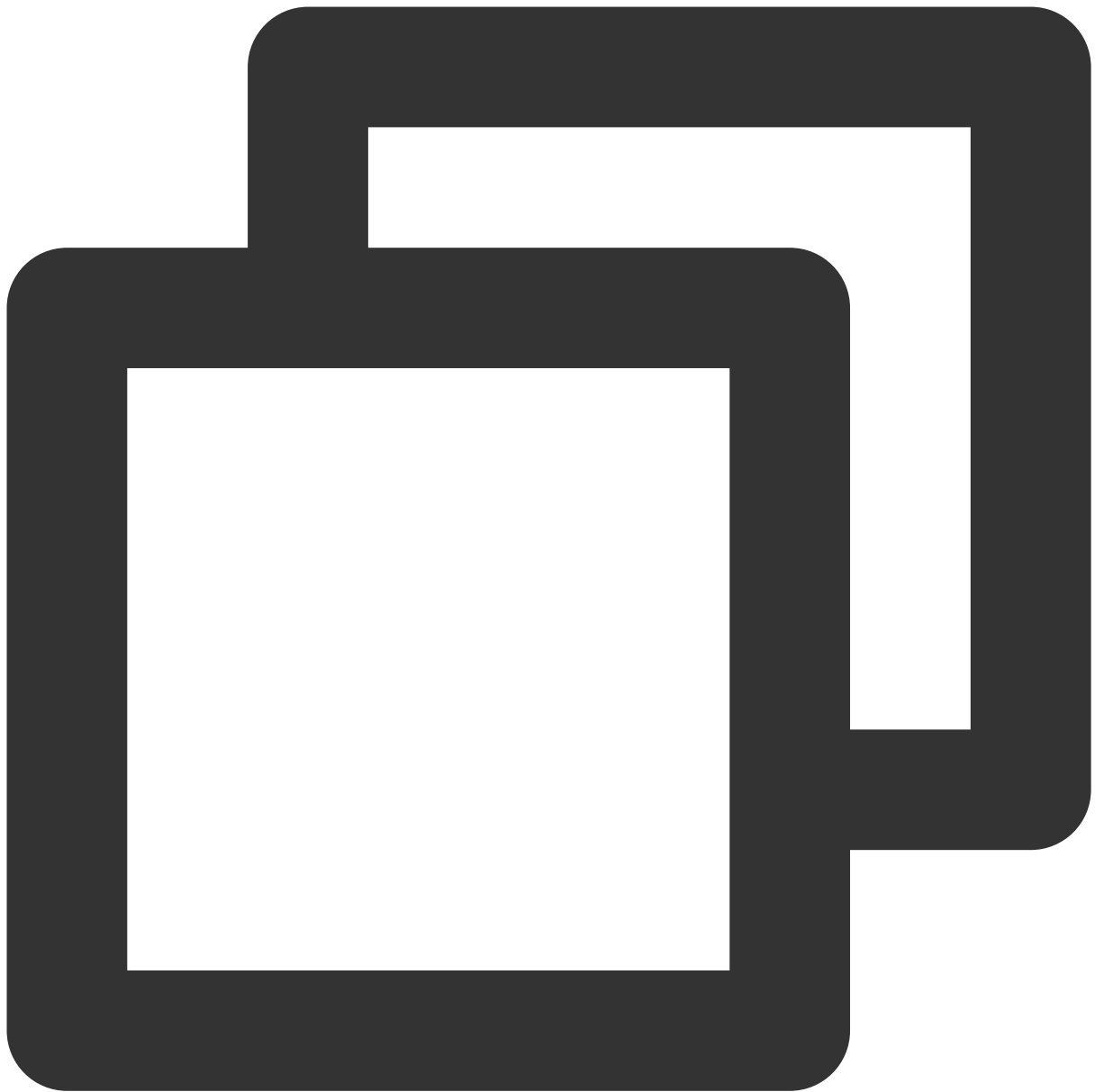
If you want to initiate a one-on-one audio or video call directly by clicking the call record in the **System Phone - Recent Calls** list, you need to use the **callWith** method in the **TUICallKitVoIPExtension** component in the Application lifecycle callback function. Here's an example:

**Note :**

Only supports dialing one-on-one audio and video calls directly.

The logged-in account must be the same account.

1. On iOS 13 (and later versions) with SceneDelegate support, and with a minimum compatibility version prior to iOS 13, you need to implement the following methods in AppDelegate and SceneDelegate respectively.



```
// Implementation in AppDelegate for handling versions before iOS 13
- (BOOL)application:(UIApplication *)application continueUserActivity:(nonnull NSUserActivity *)userActivity
    [TUICallKitVoIPExtension callWith:userActivity];
return YES;
}

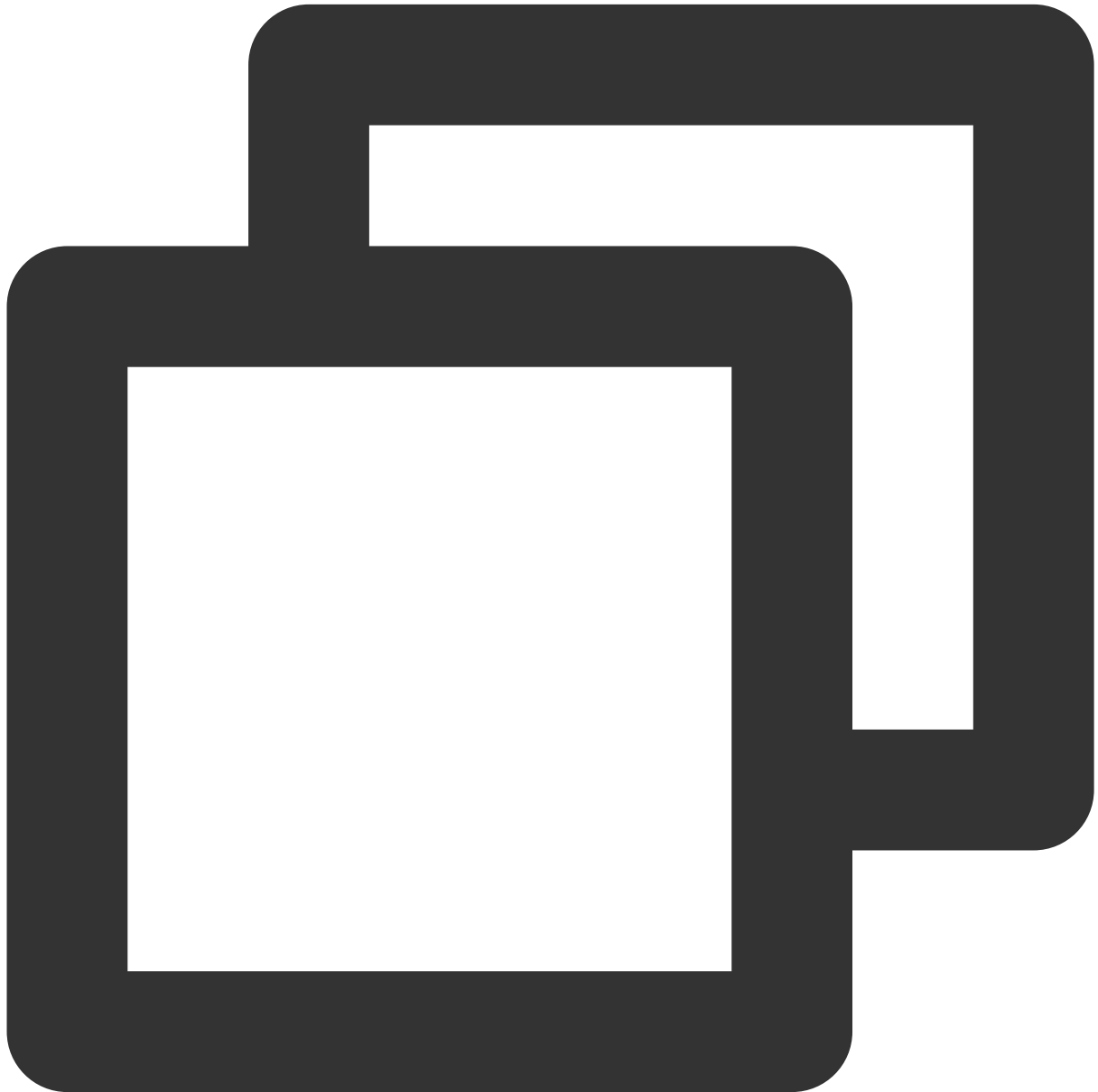
// Implementation in SceneDelegate
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)session options:(NSDictionary<UISceneSessionOption, id> *)options
    [connectionOptions.userActivities enumerateObjectsUsingBlock:^(NSUserActivity *userActivity, NSUInteger idx, BOOL *stop) {
        [TUICallKitVoIPExtension callWith:userActivity];
    }];
```

```
}

- (void)scene:(UIScene *)scene continueUserActivity:(NSUserActivity *)userActivity
    [TUICallKitVoIPExtension callWith:userActivity];

}
```

2. On iOS 13 (and later versions) without **SceneDelegate**, you only need to implement the following method in **AppDelegate**.



```
- (BOOL)application:(UIApplication *)application continueUserActivity:(nonnull NSUs
    [TUICallKitVoIPExtension callWith:userActivity];
```

```
    return YES;  
}
```

## Frequently Asked Questions

### Can't receive VoIP Push

1. First, check if the App's running environment and the certificate environment are consistent, and if the certificate ID matches. If they are inconsistent, you won't be able to receive push notifications.
2. Please make sure your currently logged-in account is offline: press the home key to switch to the background, or log in and then manually kill the process to exit. VoIP Push currently only supports push notifications in offline mode.
3. Check if [Step 3: Complete the Project Configuration](#) is done correctly.
4. Try restarting the test phone to clear the system cache and memory.

# APNs

Last updated : 2024-01-18 11:07:30

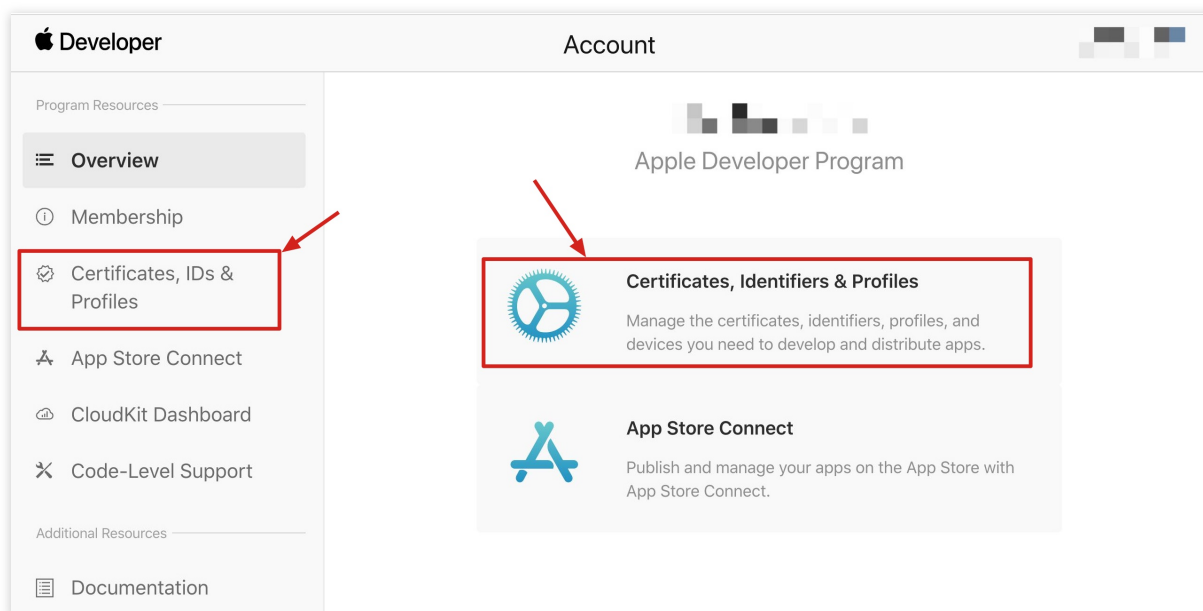
The offline wake-up function allows your app to receive audio and video call alerts even when it is running in the background or in an offline state. TUICallKit uses Apple's system-level push channel (APNs) for message notifications.

## Configure Offline Push

1. [Enable APP remote push.](#)
2. [Generate Push Certificate.](#)
3. [Upload Certificate to IM Console.](#)
4. [Complete Project Configuration.](#)
5. [Obtain deviceToken from Apple on each App login.](#)
6. [Call setAPNS interface to report it to the IM backend.](#)

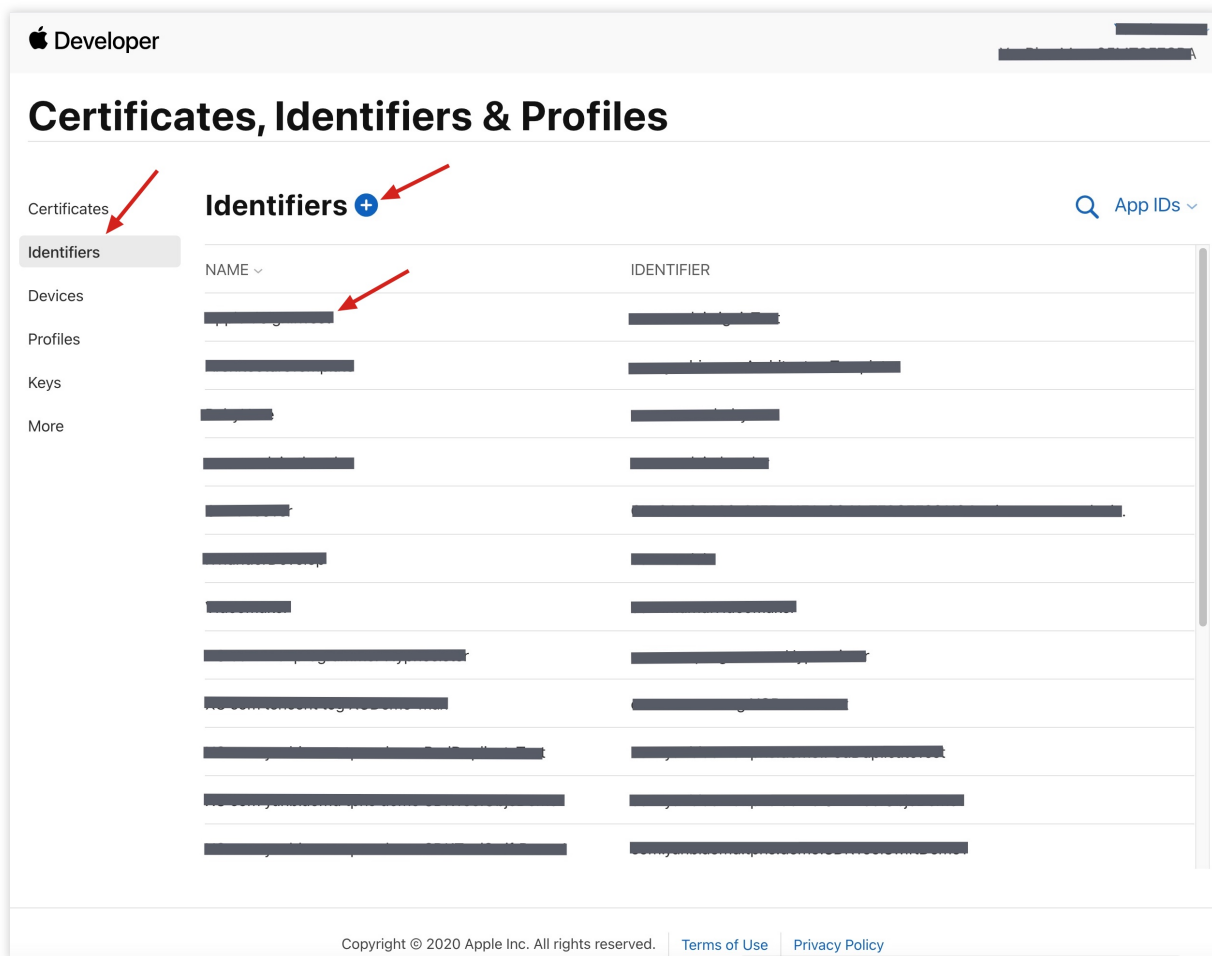
### Step 1: Enable App Remote Push

1. Log in to the [Apple Developer Center](#) website, and click on Certificates, Identifiers & Profiles or Certificates, IDs & Profiles in the sidebar to enter the Certificates, IDs & Profiles page.



2. Click the + next to Identifiers.



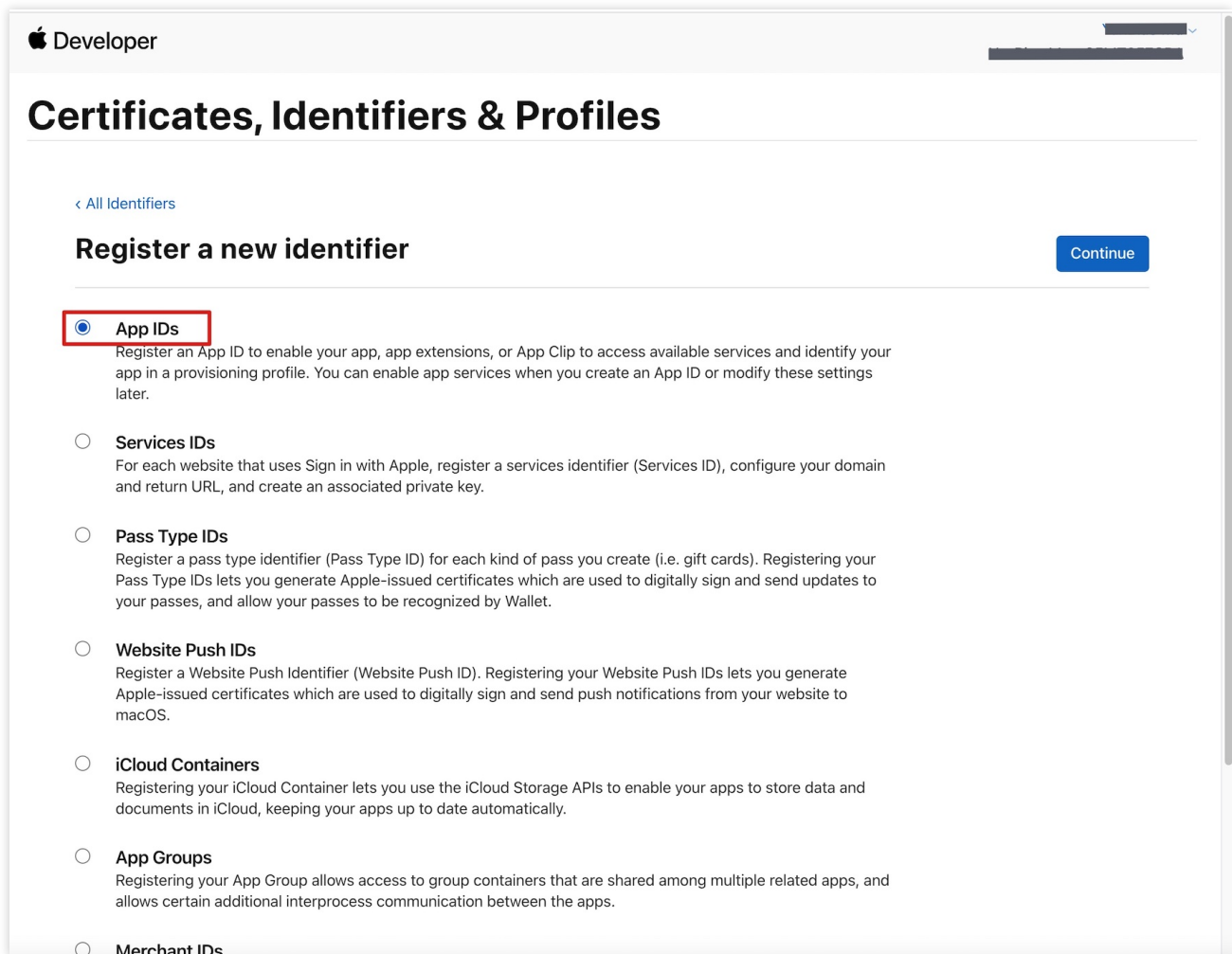


3. You can follow the steps below to create a new AppID, or add a **Push Notification Service** to your existing AppID.

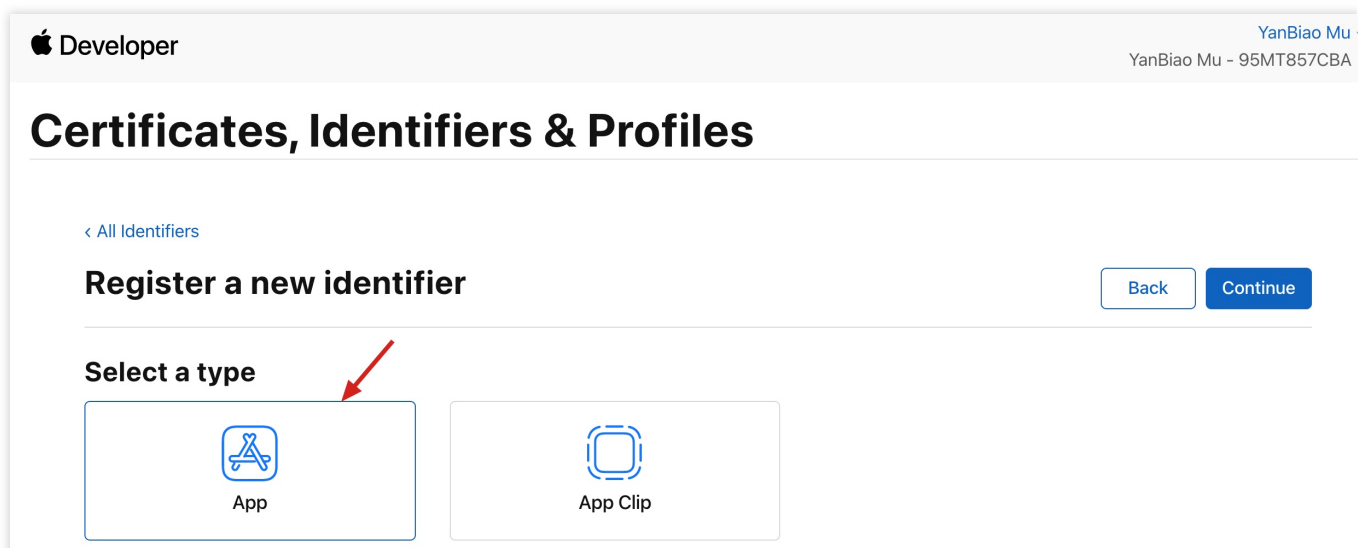
**Note :**

Please note that the Bundle ID of your App cannot use the wildcard \*, otherwise the remote push service cannot be used.

4. Check **App IDs**, click **Continue** to proceed.



5. Select **App** and click **Continue** to proceed.



6. Configure Bundle ID and other information, then click Continue to proceed.

Developer

Certificates, Identifiers & Profiles

[All Identifiers](#)

Register an App ID

BackContinue

Platform

iOS, macOS, tvOS, watchOS

Description

IMSDK Demo

You cannot use special characters such as @, &, \*, ' , " , -, .

App ID Prefix

Bundle ID

Explicit

Wildcard

com.imsdk.pushdemo

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (\*).

Capabilities

App Services

| ENABLED                  | NAME                           |
|--------------------------|--------------------------------|
| <input type="checkbox"/> | Access WiFi Information ⓘ      |
| <input type="checkbox"/> | App Attest ⓘ                   |
| <input type="checkbox"/> | App Groups ⓘ                   |
| <input type="checkbox"/> | Apple Pay Payment Processing ⓘ |
| <input type="checkbox"/> | Associated Domains ⓘ           |
| <input type="checkbox"/> | AutoFill Credential Provider ⓘ |
| <input type="checkbox"/> | ClassKit ⓘ                     |

7. Check **Push Notifications** to enable the remote push service.













©2013-2022 Tencent Cloud. All rights reserved.

Page 149 of 2527

< All Identifiers

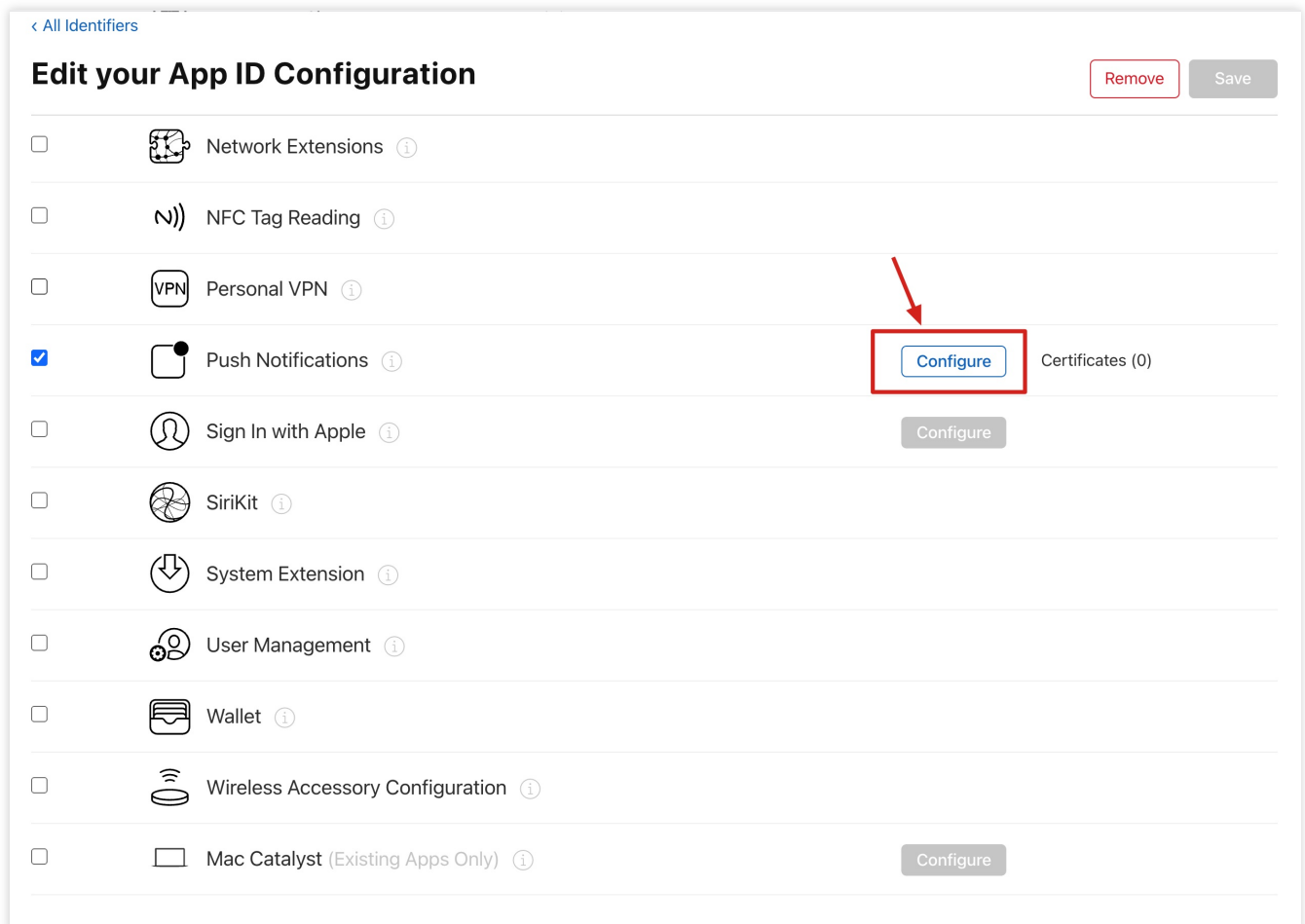
## Register an App ID

Back Continue

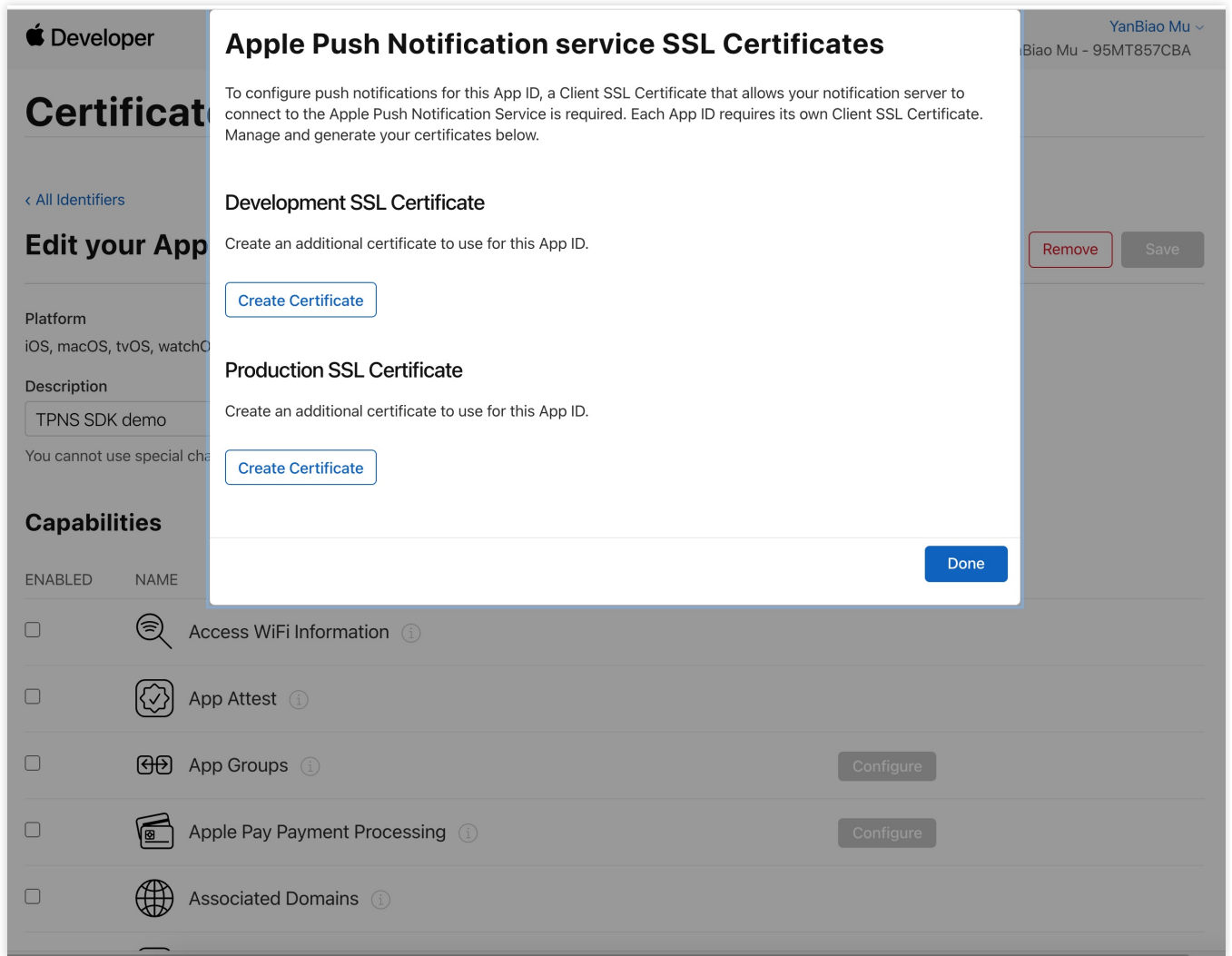
|                                     |   |           |
|-------------------------------------|---|-----------|
| <input type="checkbox"/>            |  Multipath ⓘ                           |           |
| <input type="checkbox"/>            |  Network Extensions ⓘ                  |           |
| <input type="checkbox"/>            |  NFC Tag Reading ⓘ                     |           |
| <input type="checkbox"/>            |  Personal VPN ⓘ                        |           |
| <input checked="" type="checkbox"/> |  Push Notifications ⓘ                  |           |
| <input type="checkbox"/>            |  Sign In with Apple ⓘ                  | Configure |
| <input type="checkbox"/>            |  SiriKit ⓘ                             |           |
| <input type="checkbox"/>            |  System Extension ⓘ                    |           |
| <input type="checkbox"/>            |  User Management ⓘ                     |           |
| <input type="checkbox"/>            |  Wallet ⓘ                              |           |
| <input type="checkbox"/>            |  Wireless Accessory Configuration ⓘ   |           |
| <input type="checkbox"/>            |  Mac Catalyst (Existing Apps Only) ⓘ | Configure |

## Step 2: Generate Push Certificates


1. Select your AppID, and click **Configure**.



2. You can see two **SSL Certificates** in the **Apple Push Notification service SSL Certificates** window, which are used for the remote push certificates of the development environment and production environment, as shown in the picture below:



3. First, select **Create Certificate** for the development environment. The system will prompt us to upload a **Certificate Signing Request (CSR)**.



**Certificates, Identifiers & Profiles**

[< All Certificates](#)

## Create a New Certificate

[Back](#)[Continue](#)

**Certificate Type**  
Apple Push Notification service SSL (Sandbox)

Platform:  

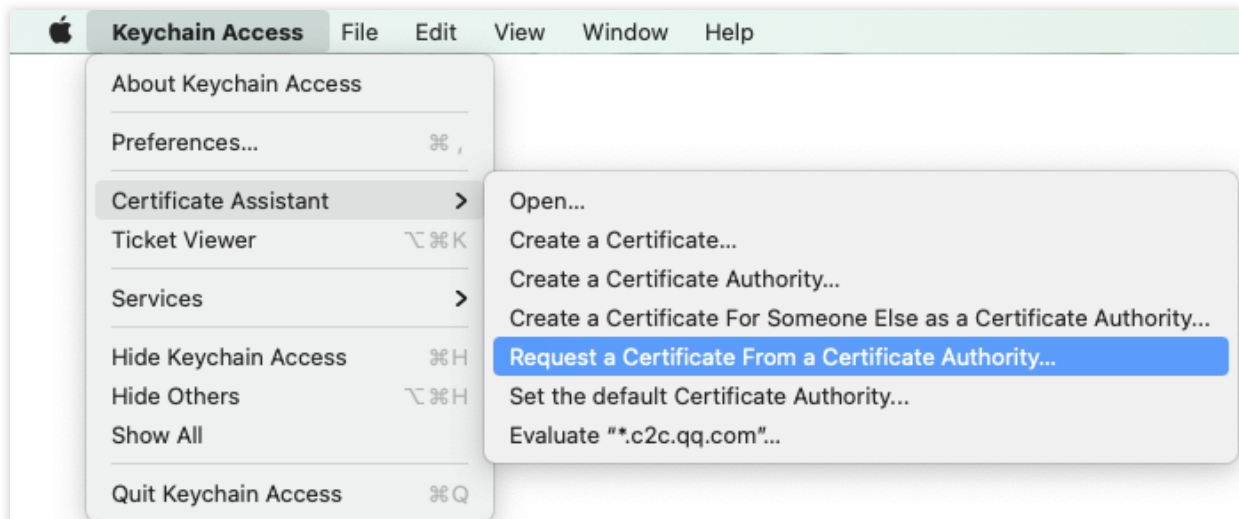
iOS

**Upload a Certificate Signing Request**  
To manually generate a Certificate, you need a **Certificate Signing Request (CSR)** file from your Mac.  
[Learn more >](#)

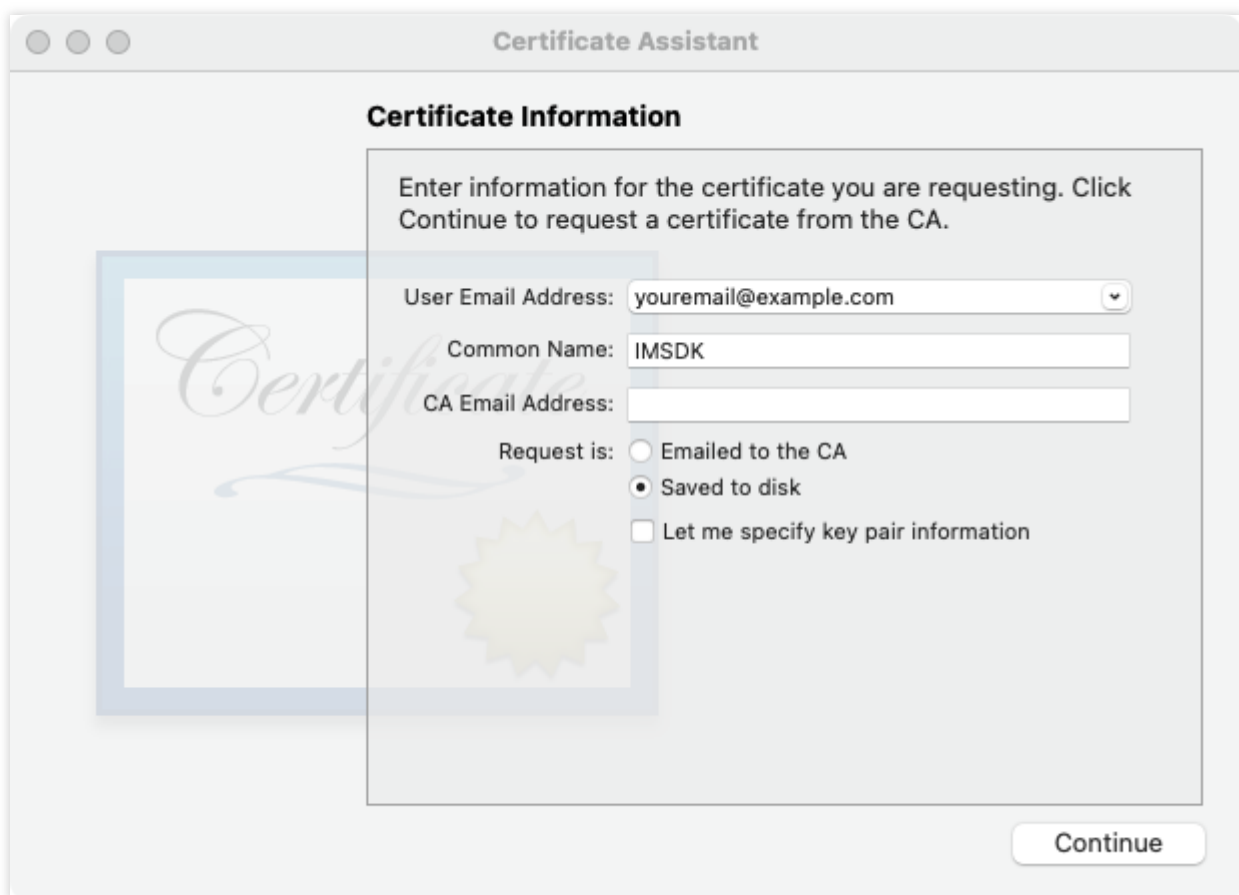
[Choose File](#)

Copyright © 2020 Apple Inc. All rights reserved. | [Terms of Use](#) | [Privacy Policy](#)

4. Next, create a CSR file. Open **Keychain Access** on your Mac, and in the menu, choose **Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority**.




5. Enter your email address, common name (your name or company name), choose **Save to disk**, click "Continue", and the system will generate a `*.certSigningRequest` file.



6. Go back to the Apple Developer website in **Step3**, click "Choose File" and upload the generated `*.certSigningRequest` file.



 Developer

**Certificates, Identifiers & Profiles**

[< All Certificates](#)


**Create a New Certificate** [Back](#) [Continue](#)

**Certificate Type**  
Apple Push Notification service SSL (Sandbox)

Platform:  

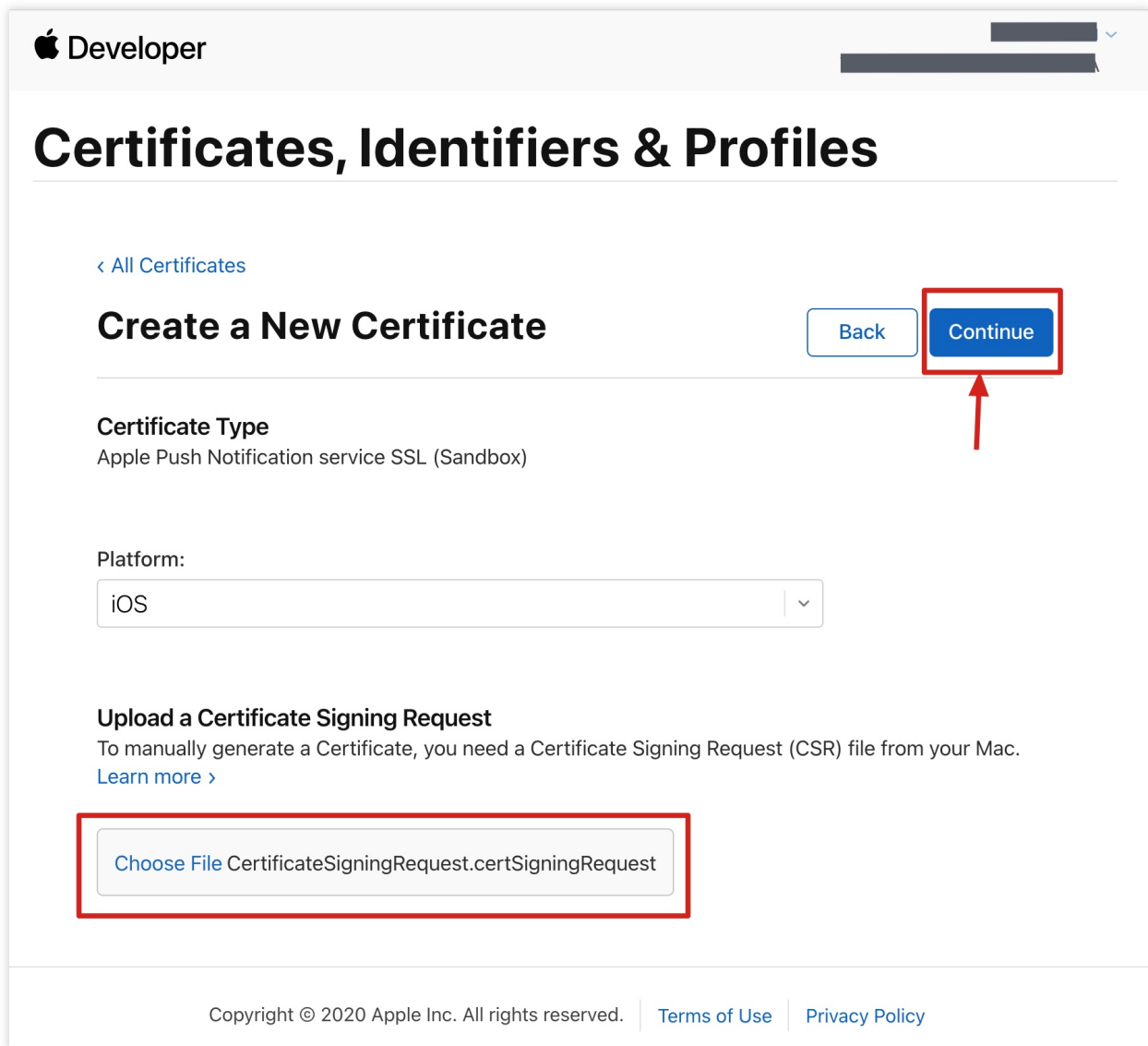
iOS

**Upload a Certificate Signing Request**  
To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac.  
[Learn more >](#)

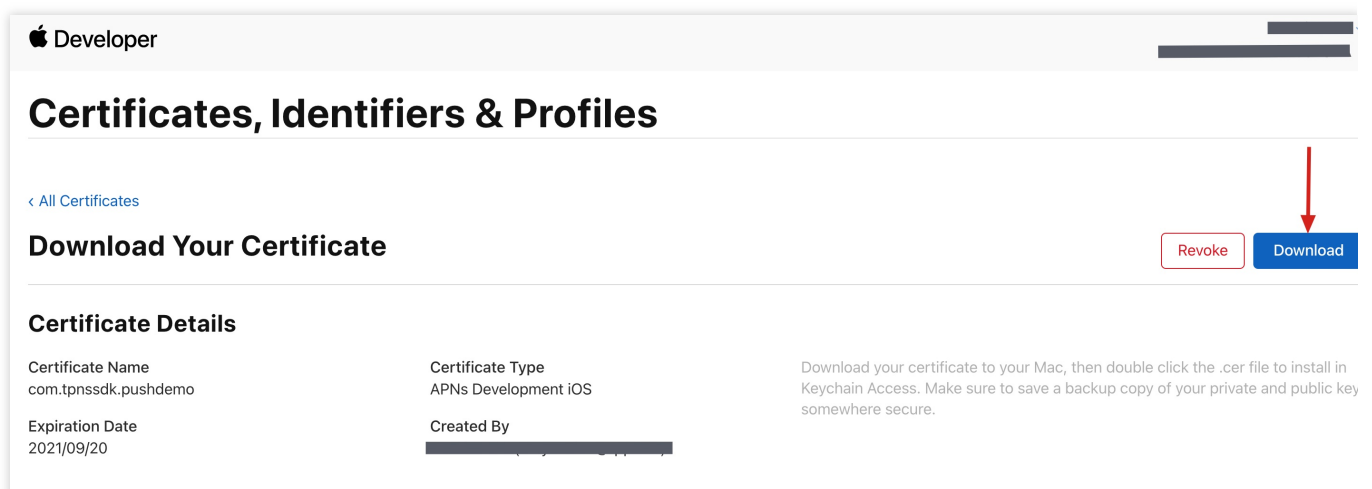
 [Choose File](#)

Copyright © 2020 Apple Inc. All rights reserved. | [Terms of Use](#) | [Privacy Policy](#)

7. Click **Continue** to generate the push certificate.



8. Click **Download** to save the **Development SSL Certificate** locally.



9. Repeat steps 1 to 8 to download the **Production SSL Certificate** for the production environment.

**Note :**

The production environment certificate is actually a combined certificate of development (Sandbox) and production (Production) environments, which can be used as certificates for both development and production environments.

**Apple Developer**

## Certificates, Identifiers & Profiles

[< All Certificates](#)

### Create a New Certificate

[Back](#) [Continue](#)

**Certificate Type**  
Apple Push Notification service SSL (Sandbox & Production)

**Platform:**  
iOS

**Upload a Certificate Signing Request**  
To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac.  
[Learn more >](#)

[Choose File](#) CertificateSigningRequest.certSigningRequest

---

**Apple Developer**

## Certificates, Identifiers & Profiles

[< All Certificates](#)

### Download Your Certificate

[Revoke](#) [Download](#)

**Certificate Details**

|  |  |   |
|--|--|---|
| <b>Certificate Name</b><br>com.tpnsdk.pushdemo | <b>Certificate Type</b><br>Apple Push Services | Download your certificate to your Mac, then double click the .cer file to install Keychain Access. Make sure to save a backup copy of your private and public somewhere secure. |
| <b>Expiration Date</b><br>2021/10/20           | <b>Created By</b><br>[Redacted]                |   |

10. Double-click the downloaded development and production environment **SSL Certificates**, and the system will import them into the keychain.

11. Open the Keychain app, under **Login > My Certificates**, right-click on the created VoIP Services certificate.

#### Note :

When saving the **P12** file, please be sure to set a password for it.

### Step3 : Upload certificates to the IM Console

1. Log in to the [IM Console](#).
2. Choose your IM application, click **Go new** under the **Offline Push Certificate Configuration** tab.

**Basic Configuration** - TUICallKit Current data center: Singapore Telegram group WhatsApp group

**Standard Billing Plan**

Status: In use ①  
Plan: TRTC Developer  
Expiration time: 2024-02-04  
[Upgrade Plan](#) [More](#)

**App Information** [Edit](#)

SDKAppID: 20006172 ①  
Application Name: TUICallKit  
Application Type: -  
Application Introduction: -

**Basic info**

Secret Key: \*\*\*\*\* [Display key](#)  
Key information is sensitive. Keep it confidential and do not disclose it.  
Creation time: 2024-01-04  
Last Modified: 2024-01-10

**Tag Configuration** [Edit](#)

TagKey TagValue

**Offline Push Certificate Configuration**

The push configuration module has been moved to the [Access Settings] page under the [Push M] on the left. [Go now](#)

**RTC Engine**

RTC Engine can help you implement audio and video calls, multi-person voice chats, video conference functions in IM applications. RTC Engine is billed independently, see [Price Overview](#) for details. R distinguish between data centers. Your configuration and log data using RTC Engine will be stored default.

**Call** [Integration Guide](#)

TRTC Call is a call component that comes with a UI kit. You can use it to implement features including calls and multi-platform call making/answering.

Status: Activated  
Current version: Trail [Learn more](#) [Purchase](#)  
Expiration time: 2024-01-11

3. In the **Manufacturer configuration**, switch to **iOS**, click the **Add Certificate** button. Then choose the certificate type, upload the iOS Certificate (p.12), set the certificate password, and click **Confirm**.

**Access settings** 20006172 - TUICallKit Current data center: Singapore Telegram group WhatsApp group

**Push mode setting**

Instant messaging IM provides two modes: normal push and all-member/tag push. Ordinary push is triggered by the client and is enabled by default. It does not support closing at the moment. All-member/tag push is by is sent and supports adjusting the interface call frequency.

Normal push: Enabled by default  
All members/tag push function: Disabled

**1 Manufacturer configuration**

IM supports online and offline push notifications, manufacturer. The system The level push channel

Android **iOS**

**Add Certificate**

**Add iOS Certificate**

Push Type: ☒ APNs Push ☐ VoIP Push  
Certificate Type: ☒ Production environment ☐ Development Environment  
Configuration type: ☒ p12 ☐ p8  
iOS certificate (p12):  [Select file](#)  
[How to generate an APNs certificate?](#)  
Certificate password:   
[Confirm](#)

**2 Local deployment**

You can use the "Quick Configuration" and "Download Configuration File" functions. With 3 lines of code and 2 minutes, you can complete the push access of multiple manufacturers at once.

[Quick configuration](#) [Manual configuration](#)

Note :

The uploaded certificate name should preferably be in English (especially without special characters like parentheses).

A password is required when uploading a certificate; without a password, you won't receive push notifications.

For App Store release certificates, set the environment to production, otherwise, you won't receive push notifications.

The uploaded p12 certificate must be a genuine and valid certificate that you applied for.

4. Once the push certificate information is generated, record the certificate ID.

1

**Manufacturer configuration**

IM supports online and offline push notifications. Online push is delivered through the instant messaging IM channel, which is fast and reliable; for offline push, it is recommended that you use the system-level push channel. The system-level push channel has a more stable system-level long connection, and the resource consumption is greatly reduced.

Android

**iOS**

Add Certificate

ID: 15

Delete Edit

Certificate information

1704887549753.%E8%AF%81%E4%B9%A6.p12

Push Type

APNs Push

Certificate Type

Production environment

mutable-content

Enabled

Certificate password

123321

Expiration time

2024-04-26 11:57:46

2

**Local deployment**

You can use the "Quick Configuration" and "Download Configuration File" functions. With 3 lines of code and 2 minutes, you can complete the push access of multiple manufacturers at once.

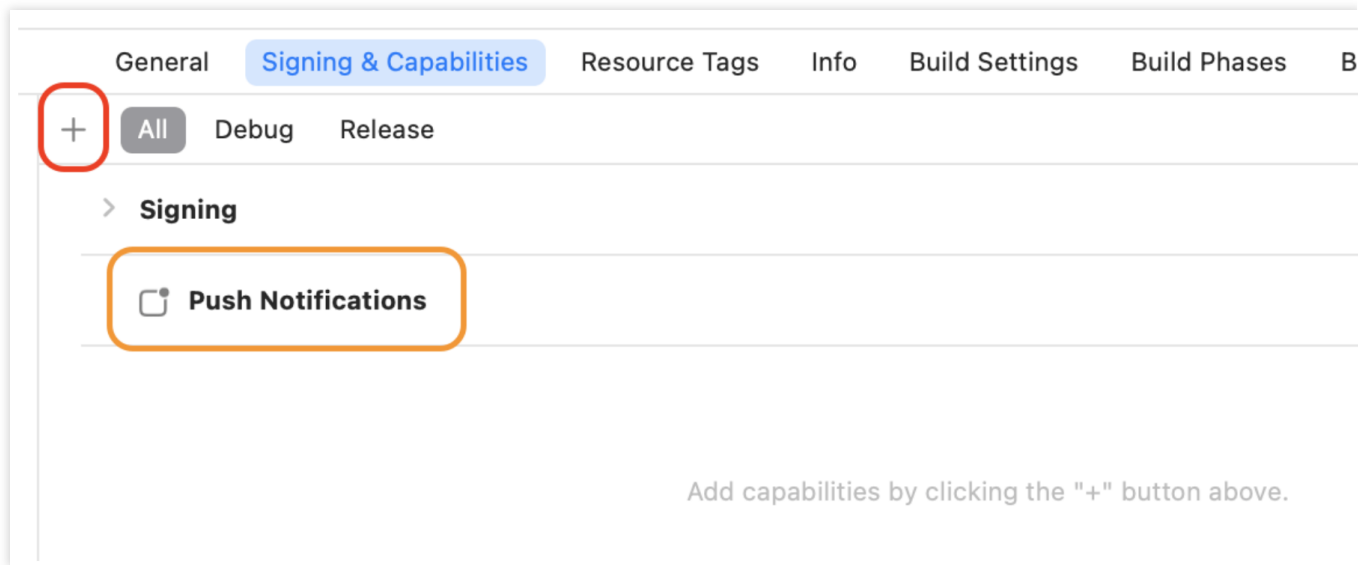
Quick configuration

Manual configuration

## Step4 : Complete Project Configuration

To add the required permissions in your application, enable push notification functionality in your Xcode project.

Open your **Xcode** project, go to **Project > Target > Capabilities**, click the plus button in the red frame, and select and add **Push Notifications**. The added result is shown in the yellow frame in the image below:

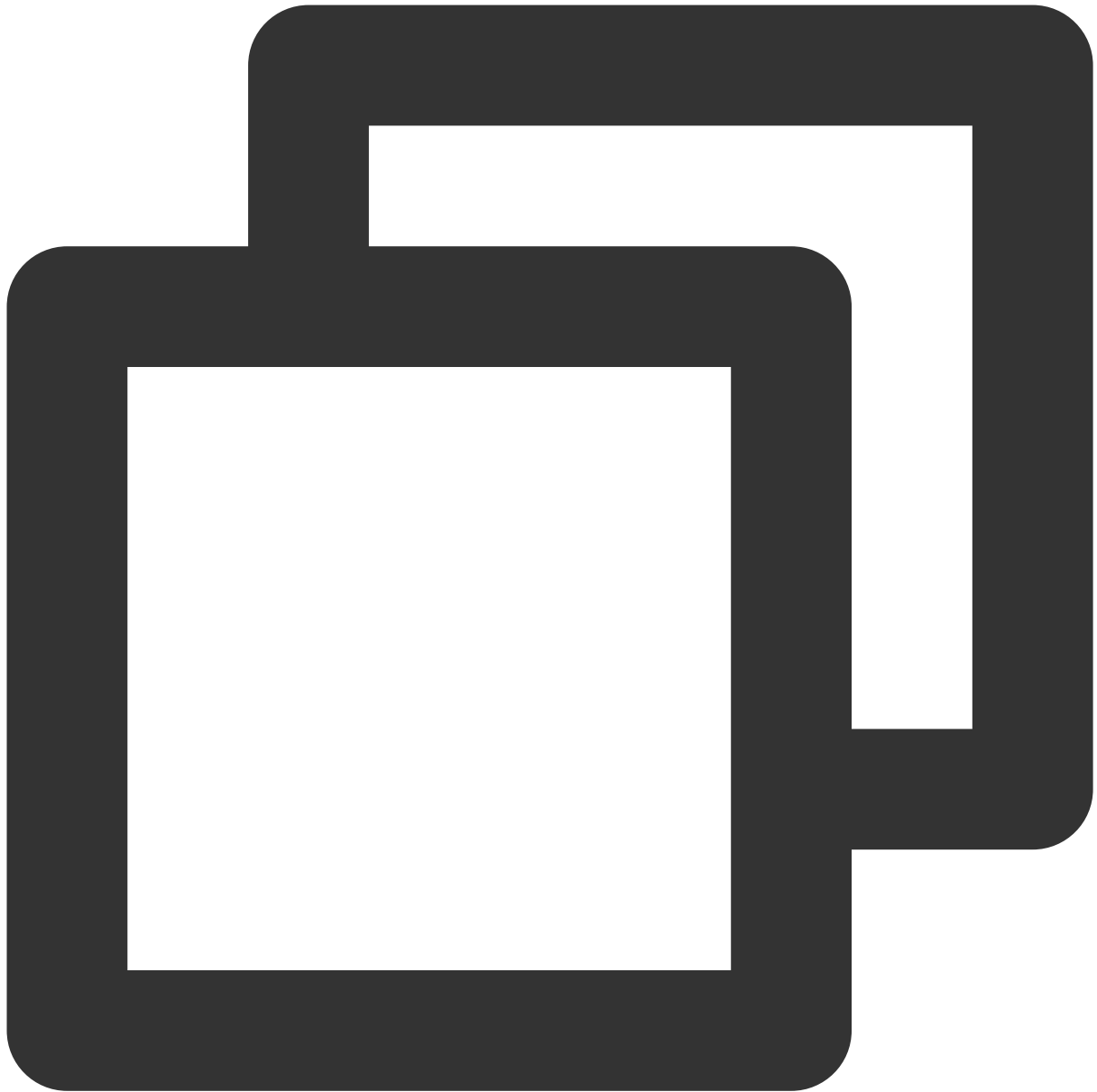


### Step 5: Obtain the deviceToken from Apple every time the App logs in

You can add the following code to your App to request the deviceToken from Apple's backend server:

#### Note :

To ensure compliance, it is recommended to request the deviceToken from Apple only after the user has agreed to the privacy policy.



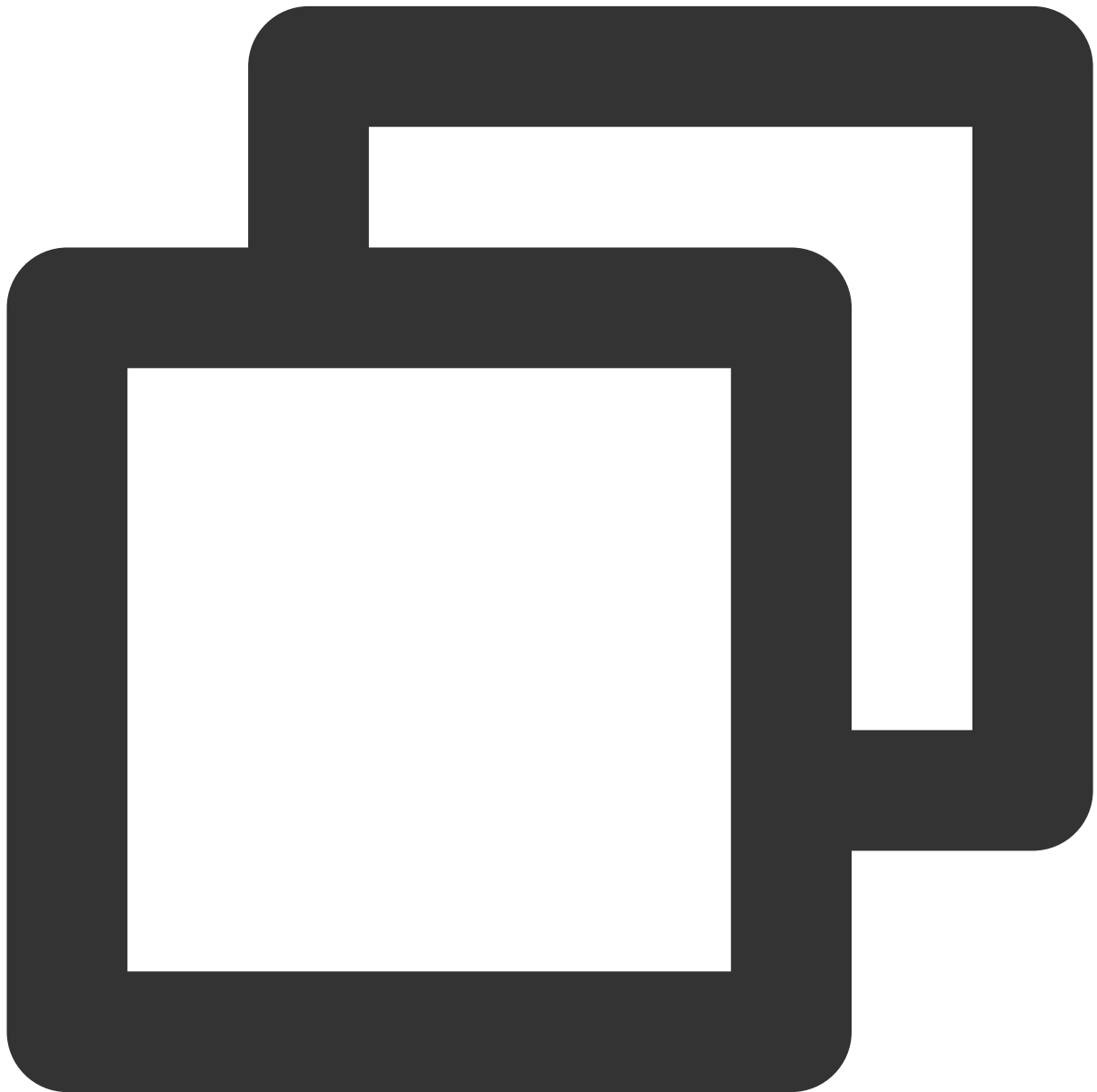
```
// Request DeviceToken from Apple's backend
- (void)registerNotification {
    [[UIApplication sharedApplication] registerUserNotificationSettings:
        [UIUserNotificationSettings settingsForTypes:
            (UIUserNotificationTypeSound | UIUserNotificationTypeAlert | UIUserNo
            categories:nil)]];
    [[UIApplication sharedApplication] registerForRemoteNotifications];
}

// The deviceToken returned in the AppDelegate's callback needs to be reported to T
- (void)application:(UIApplication *)app didRegisterForRemoteNotificationsWithDevice
```

```
// Record the deviceToken returned by Apple
_deviceToken = deviceToken;
}
```

### Step 6: Call the API to report it to the IM backend

Once the IM SDK has successfully logged in, you can call the [setAPNS](#) API to upload the deviceToken obtained in [Step 5](#) to Tencent Cloud backend. The sample code is as follows:



```
- (void)pushRegisterIfLoggedIn {
    if (self.deviceToken) {
```



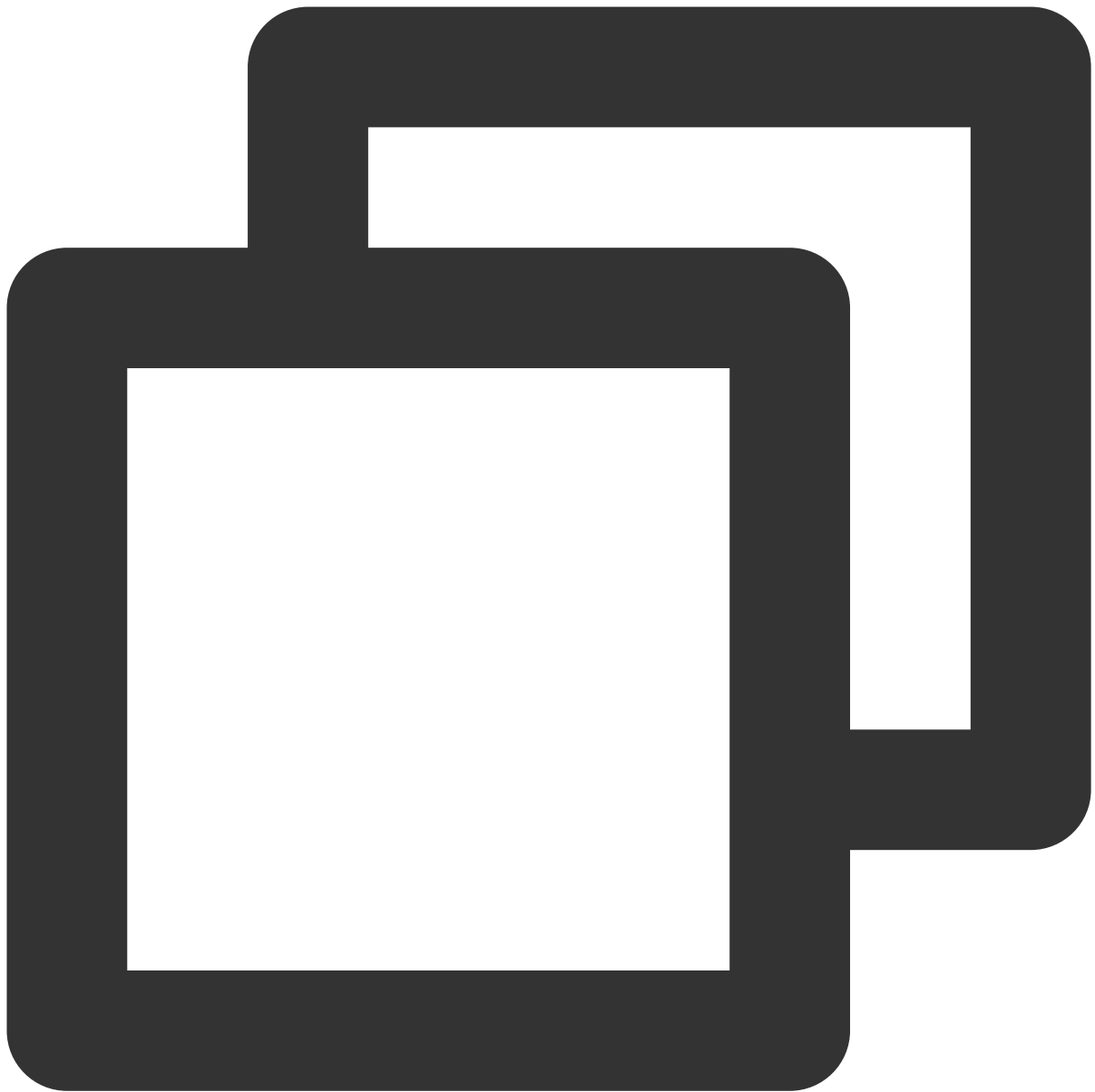
```
V2TIMAPNSConfig *config = [[V2TIMAPNSConfig alloc] init];
// Users register their Apple developer certificate, download, and generate t
// Pass the generated p12 file to the Tencent Certificate Management Console,
// Pass the generated certificate ID to the businessID parameter below.
config.businessID = sdkBusiId;
// Value of deviceToken obtained from Apple's backend server
config.token = self.deviceToken;
[[V2TIMManager sharedInstance] setAPNS:config succ:^(
    NSLog(@"%s, succ", __func__);
} fail:^(int code, NSString *msg) {
    NSLog(@"%s, fail, %d, %@", __func__, code, msg);
}]];
}
// ...
}
```

## Customize Offline Ringtone

To set the `iOSSound` field in the `offlinePushInfo` of the `params`, please call the [call](#) method while dialing and pass the audio file name to `iOSSound`.

### Note :

Offline push sound settings (effective for iOS only). If you want to customize `iOSSound`, you need to link the audio file to the Xcode project first, and then set the audio file name (with extension) to `iOSSound`.



```
[[TUICallKit sharedInstance] call:@"mike's id" params:[self getCallParams] callMedi  
  
- (TUICallParams *)getCallParams {  
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];  
    TUICallParams *callParams = [TUICallParams new];  
    callParams.offlinePushInfo = offlinePushInfo;  
    callParams.timeout = 30;  
    return callParams;  
}
```

```
+ (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = TUICallingLocalize(@"TUICallKit.have.new.invitation");
    // iOS push type: if you want user VoIP, please modify type to TUICallIOSOfflinePushType
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeAPNs;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelID"
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}
```

## Frequently Asked Questions

### 1、Can't receive push notifications and backend reports bad deviceToken error?

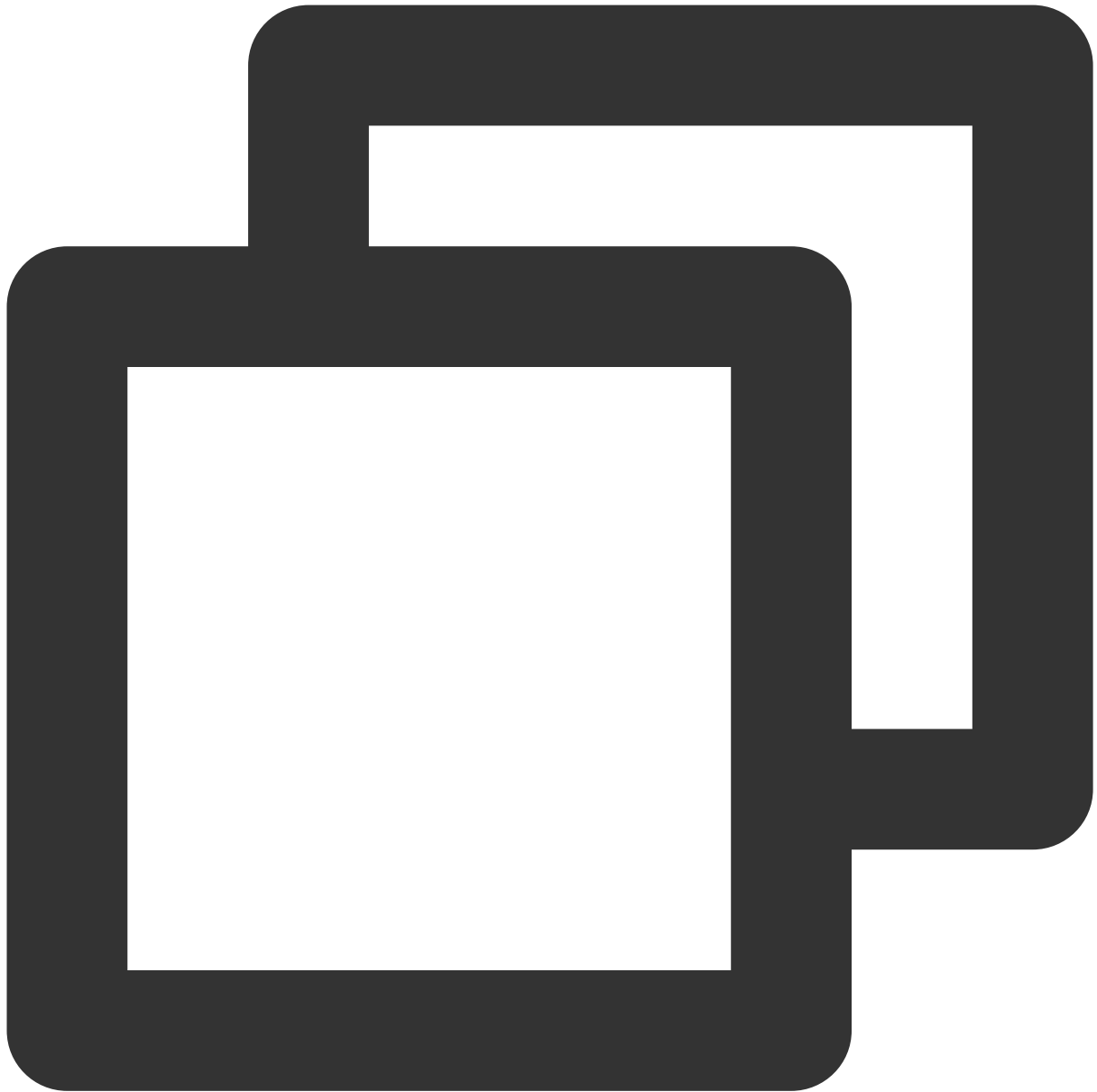
The Apple deviceToken is related to the current compilation environment. If the certificate ID used for [uploading the deviceToken to Tencent Cloud after logging in to IMSDK](#) and the token are inconsistent, an error will be reported.

If you are using a Release environment for compilation, the `-`

`application:didRegisterForRemoteNotificationsWithDeviceToken:` callback returns the production environment token. In this case, the businessID needs to be set to the production environment [Certificate ID](#).

If you are using a Debug environment for compilation, the `-`

`application:didRegisterForRemoteNotificationsWithDeviceToken:` callback returns the development environment token. In this case, the businessID needs to be set to the development environment certificate ID.



```
V2TIMAPNSConfig *config = [[V2TIMAPNSConfig alloc] init];
/* Pass the generated p12 file to the Tencent Certificate Management Console, and t
// Push certificate ID
config.businessID = sdkBusiId;
config.token = self.deviceToken;
[[V2TIMManager sharedInstance] setAPNS:config succ:^(
} fail:^(int code, NSString *msg) {

}];
```

## **2、Occasionally, iOS development environment does not return deviceToken or prompts APNs to request token failure?**

This problem is caused by the instability of APNs service. You can try the following solutions:

1. Insert a SIM card in your phone and test using 4G network.
2. Test after uninstalling and reinstalling the app, restarting the app, or restarting the device.
3. Test with a package under the production environment.
4. Test with other iOS system phones.

# Android

Last updated : 2023-09-21 15:37:01

The offline call push feature enables your application to receive audio/video calls even if it runs in the background or is offline. `TUICallKit` integrates the `TUIOfflinePush` component to implement the offline call push feature. This document describes how to integrate the `TUIOfflinePush` component in your audio/video call project.

## Preparations

1. Register your application on vendors' push platforms. The offline push feature relies on the vendors' own channels. You need to register your application on the vendors' push platforms to get parameters such as `APPID` and `APPKEY`.

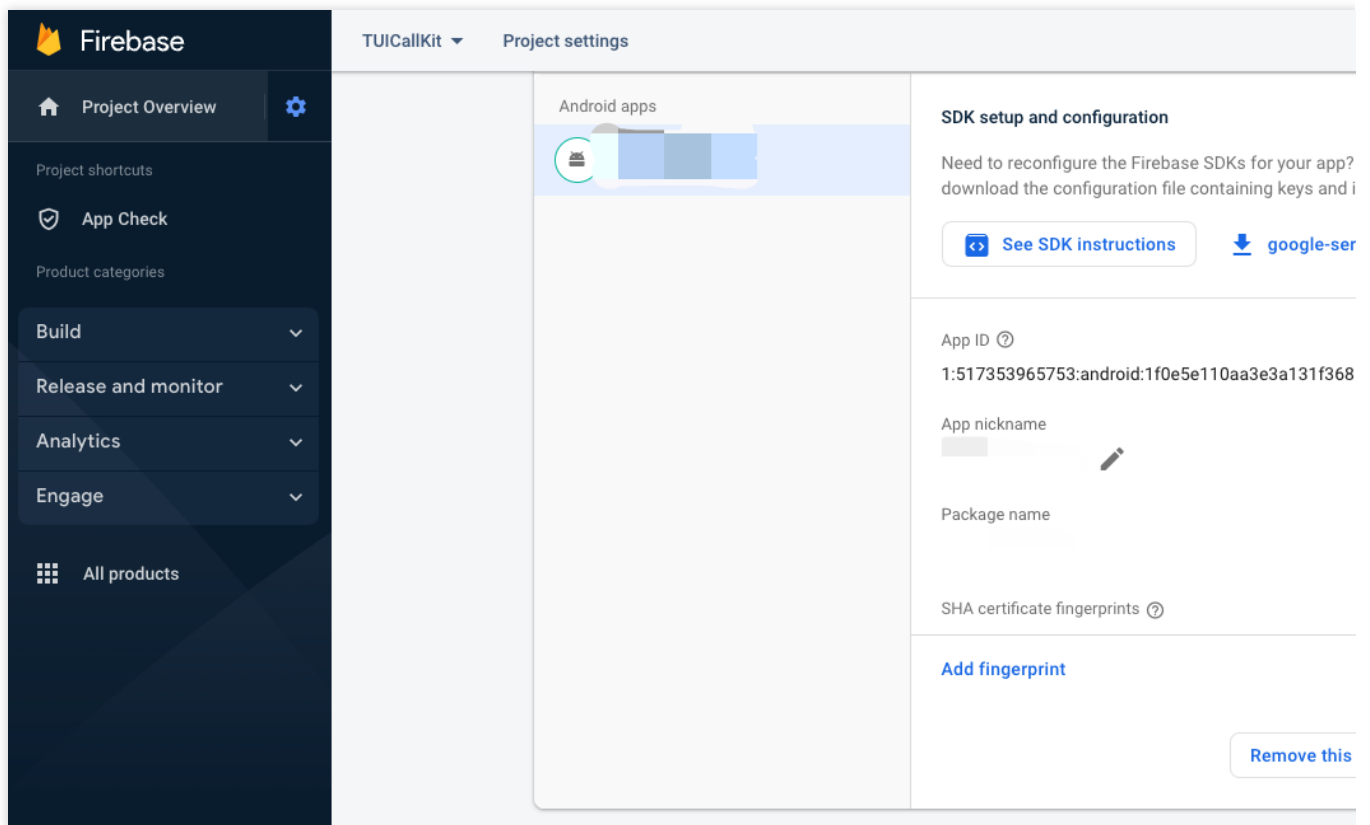
### Note

The following two files will be used in subsequent steps:

When registering on Huawei Push, download and save the `agconnect-services.json` file.

When registering on Google FCM, download and save the `google-services.json` file.

Google FCM :

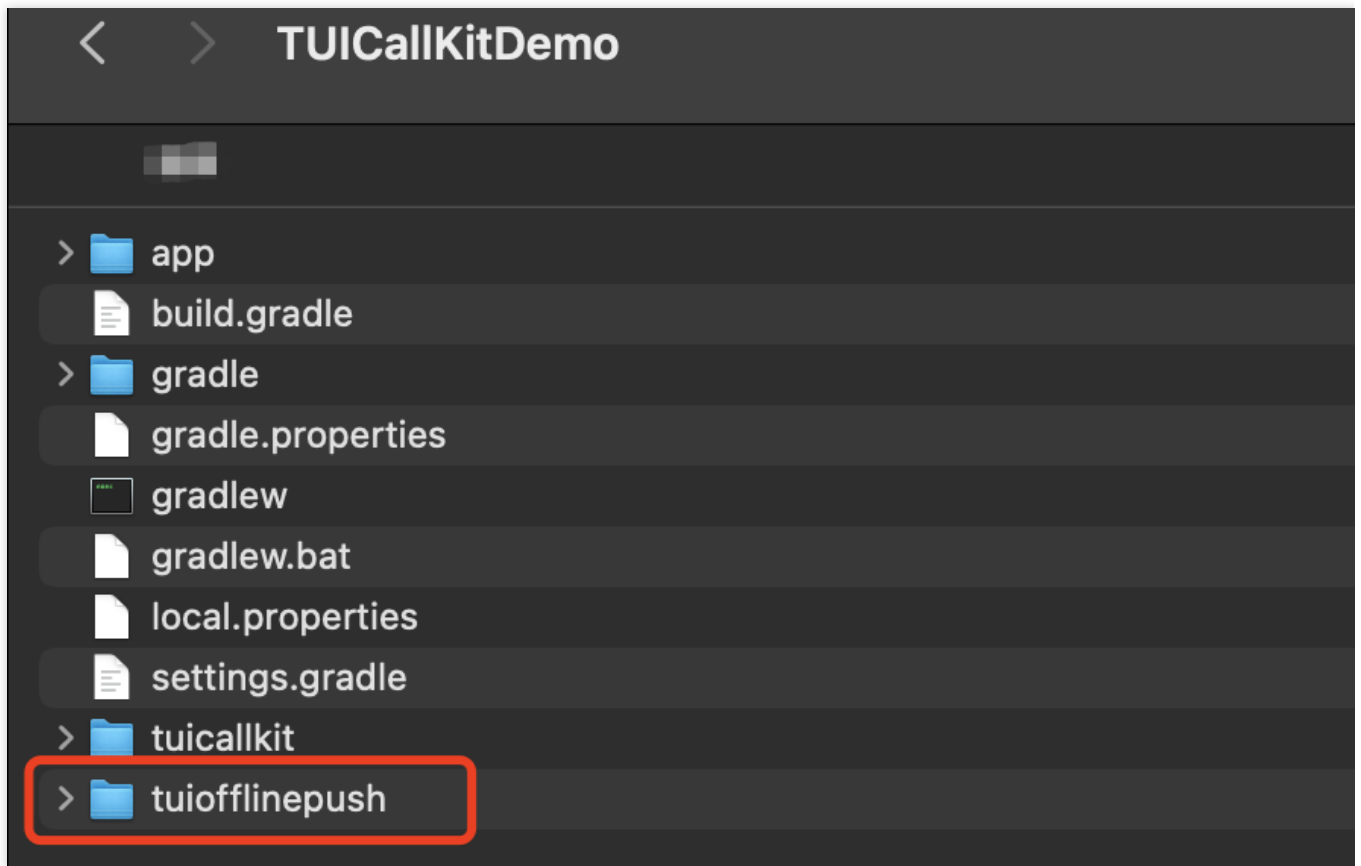


## 2. Configure in the Chat console

You need to enter the same package name when registering on each vendor channel for message interconnection. Record the generated ID, `APPID`, and `APPKEY`, which will be used in subsequent steps.

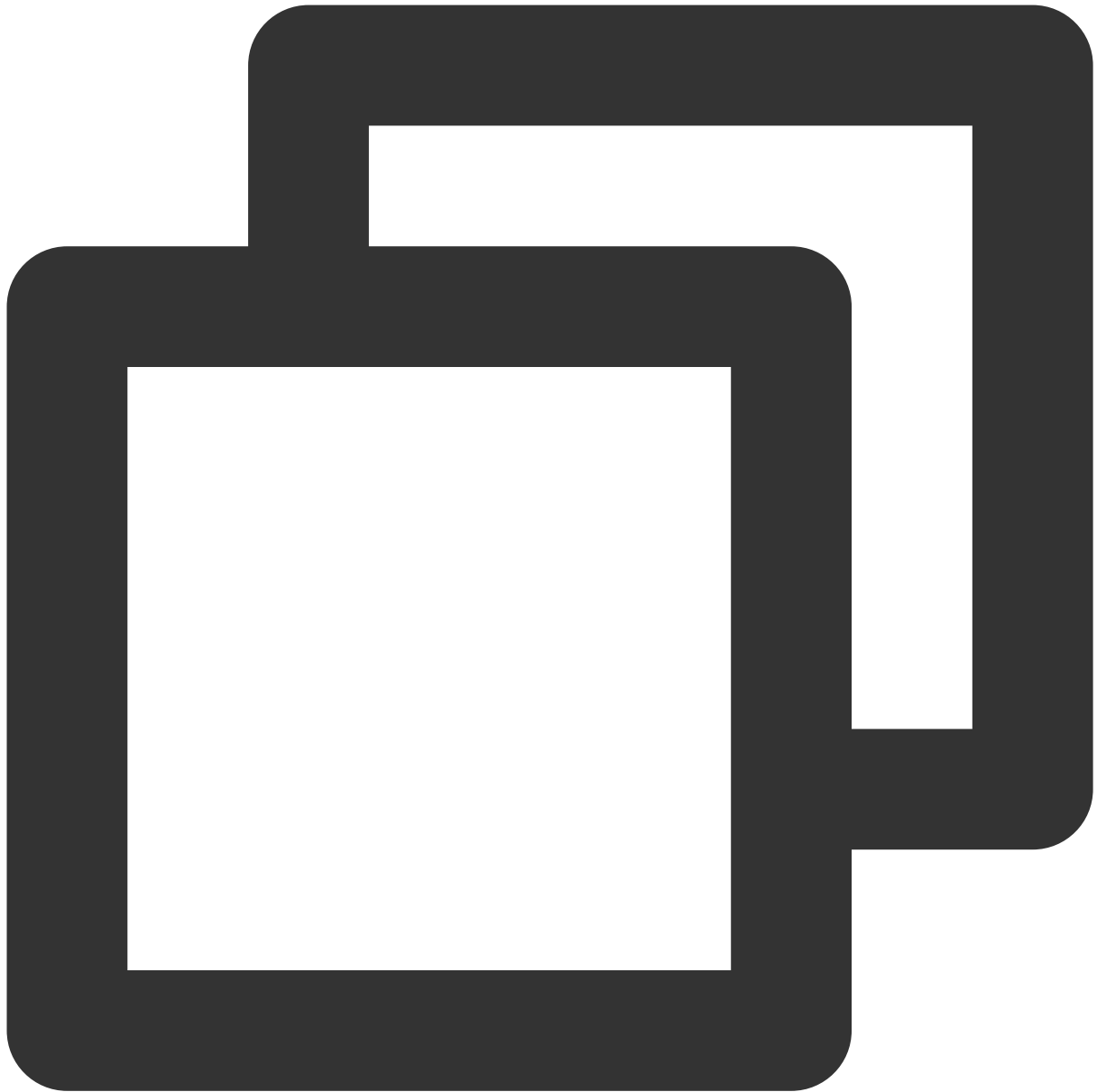
## Step 1. Download and import the component

1. Go to [GitHub](#), clone or download the code, and copy the `tuiofflinepush` subdirectory in the `Android` directory to the same directory as `app` in your project, as shown below.



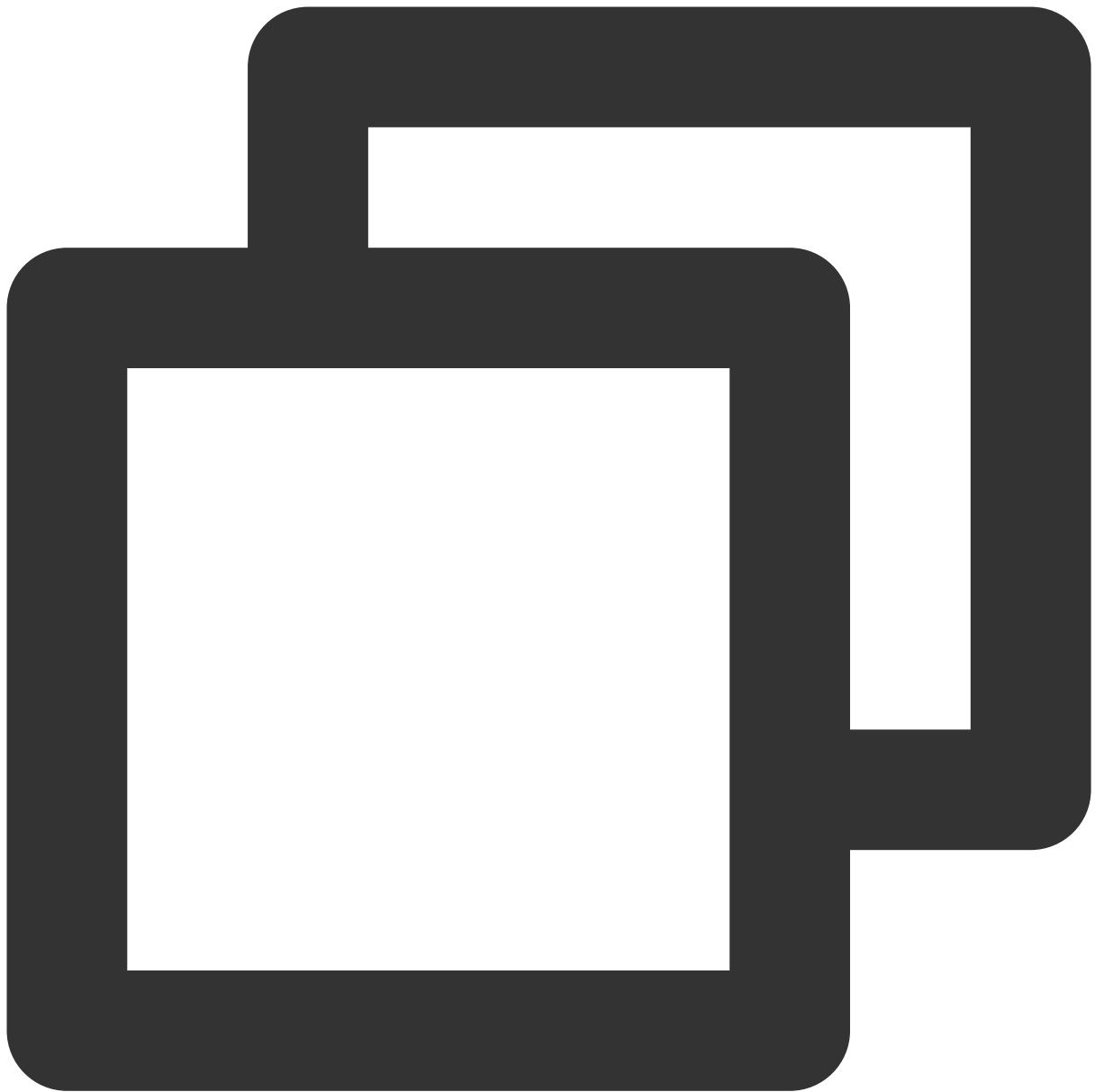
2. Find the `settings.gradle` file in the project root directory and add the following code:





```
include ':tuiofflinepush'
```

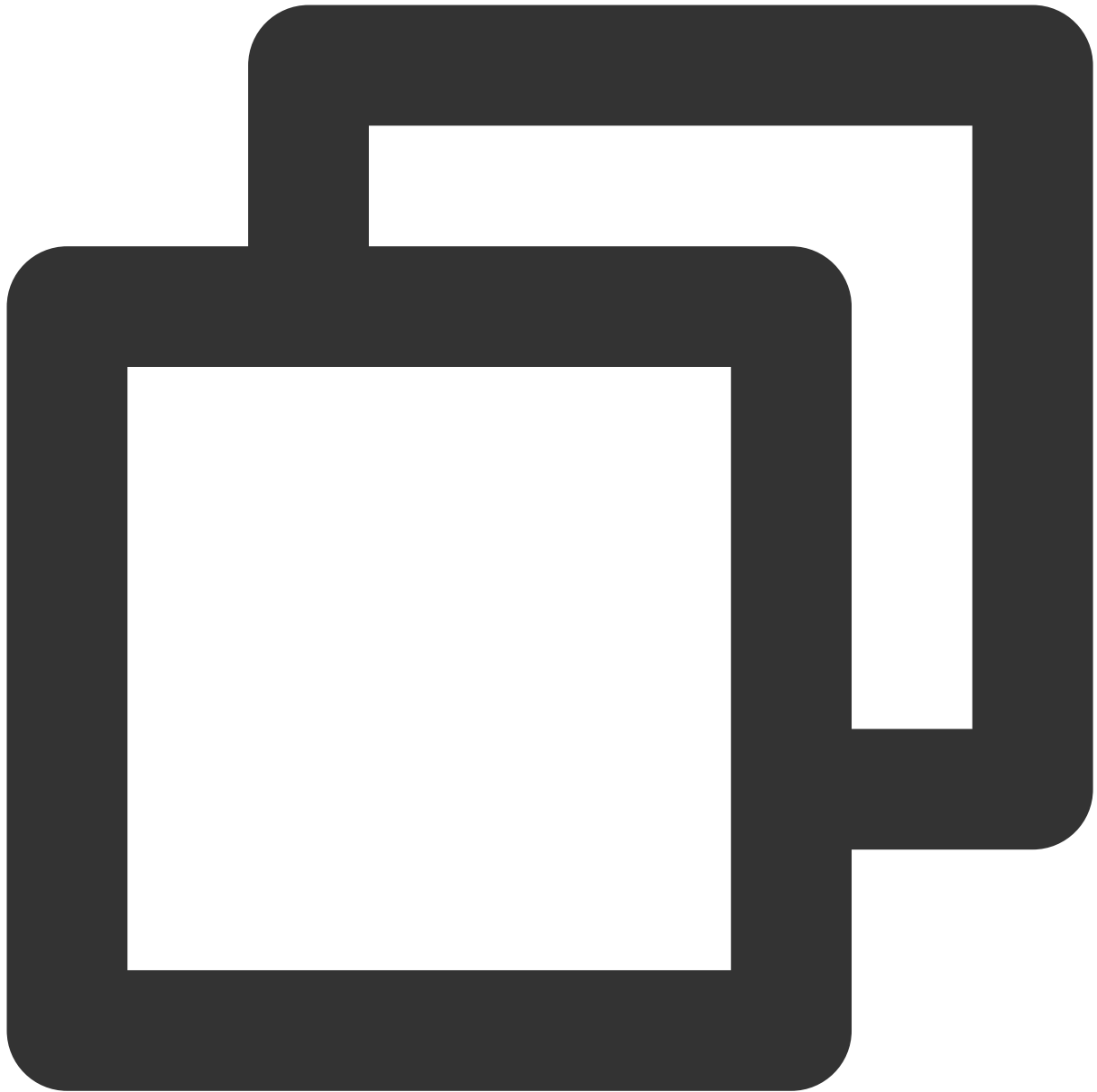
3. Find the `build.gradle` file in the `app` directory and add the following code to declare the dependencies of the current application on the `TUIOfflinePush` component just added:



```
api project(':tuiofflinepush')
```

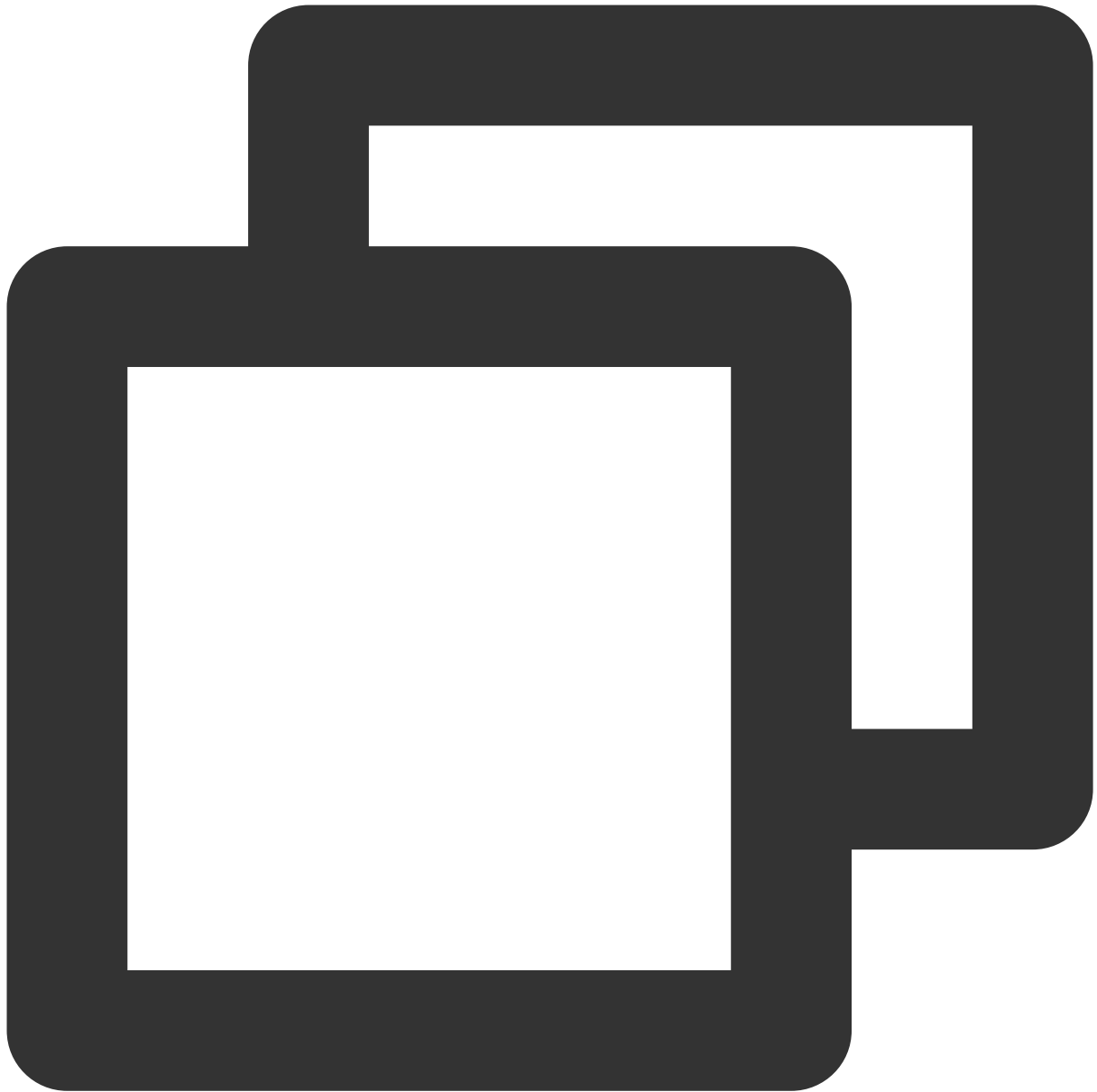
## Step 2. Configure the project

1. In the `app` directory, find the `build.gradle` file and rename the application package as needed.



```
applicationId 'com.****.trtc'
```

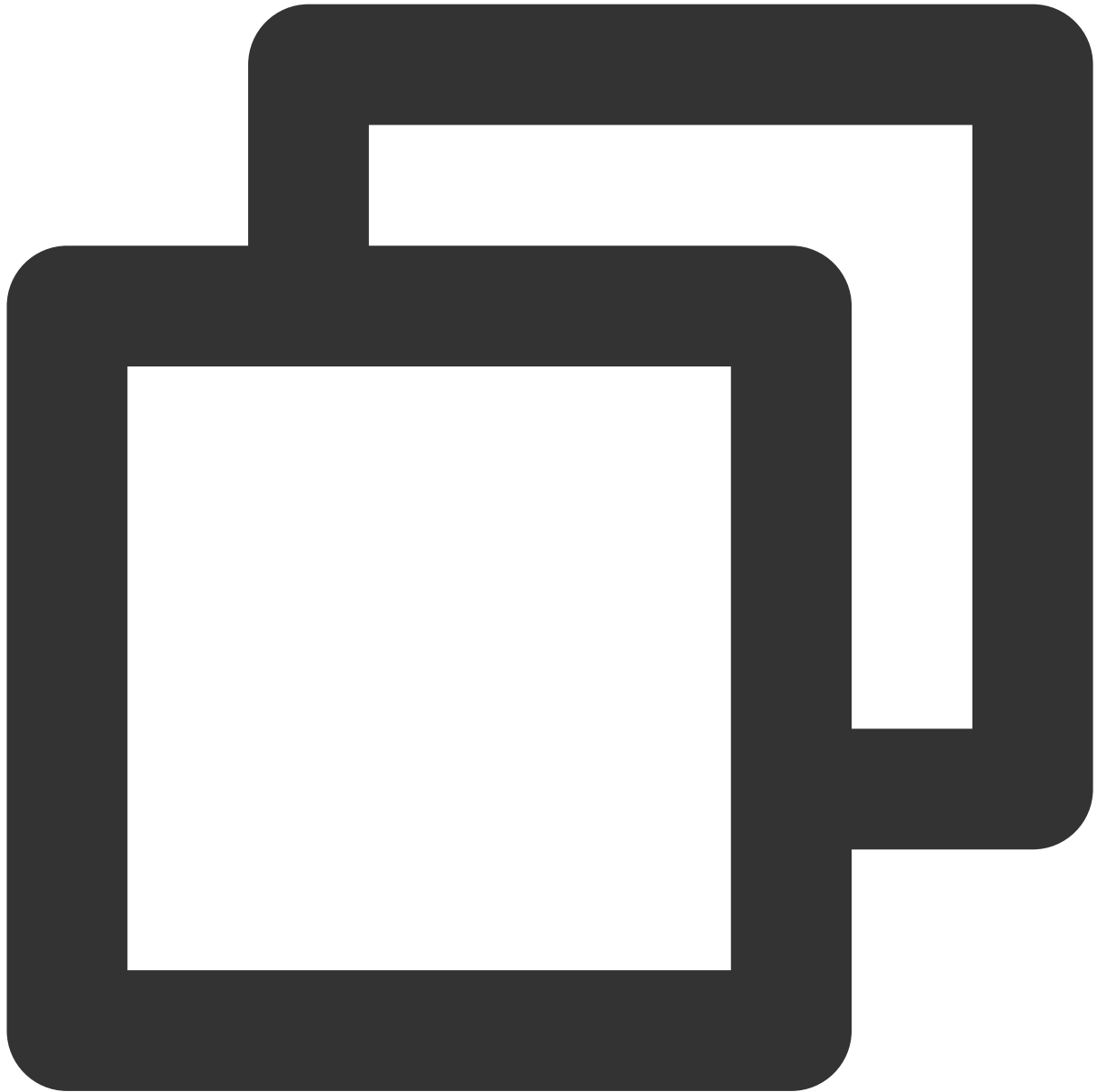
2. In the `app` directory, find the `build.gradle` file and set `ViVo` access parameters `VIVO_APPKEY` , `VIVO_APPID` , and `HONOR_APPID` to avoid compilation or execution errors.



```
manifestPlaceholders = [  
    "VIVO_APPKEY": "PLACEHOLDER",  
    "VIVO_APPID" : "PLACEHOLDER",  
    "HONOR_APPID": "PLACEHOLDER"  
]
```

3. Configure the files for Huawei and Google push platforms: In the `app` directory, replace the `google-services.json` file, which is the one saved when you registered on Google FCM during [Preparations](#). In the `app` directory, add the `agconnect-services.json` file, which is the one saved when you registered on Huawei Push during [Preparations](#).

4. Enter the ID, `APPID` , `APPKEY` recorded during [Preparations](#) in the `PrivateConstants` file and check whether the parameter configuration is correct. Enter the following parameters:



```
public class PrivateConstants {  
    /***** Mi offline push parameters start *****/  
    // Certificate ID generated after uploading a third-party push certificate in t  
    public static final long XM_PUSH_BUZID = ID of the certificate assigned to your  
    // `APPID` and `APPKEY` assigned by the Mi open platform  
    public static final String XM_PUSH_APPID = "`APPID` of the certificate assigned  
    public static final String XM_PUSH_APPKEY = "`APPKEY` of the certificate assign  
    /***** Mi offline push parameters end *****/  
}
```

```
}
```

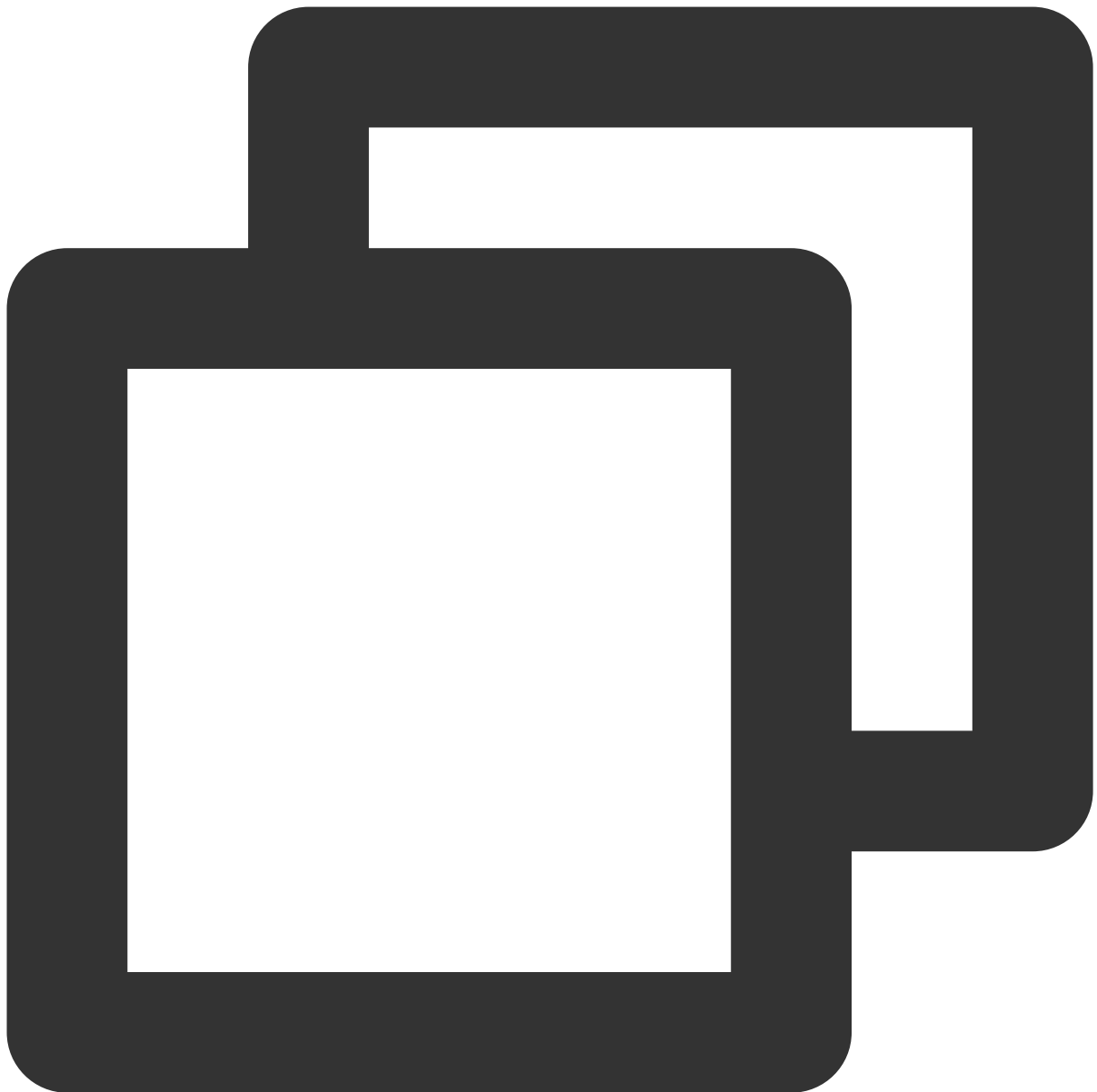
**Caution**

This step is very important. Carefully check whether the parameter configuration is correct.

After you complete the above steps, your project can use the offline call push feature of `TUICallKit`.

## Step 3. Customize the offline notification content

`TUICallKit` provides the default notification format. However, if you want to customize the notification content, modify the [OfflinePushInfoConfig.java](#) file.



```
public static TUIOfflinePushInfo createOfflinePushInfo(Context context) {
    TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
    pushInfo.setTitle("mike");
    pushInfo.setDesc("You have receive a new call");
    //OPPO must set a ChannelID to receive push messages. This channelId needs to be
    pushInfo.setAndroidOPPOChannelID("tuikit");
    pushInfo.setIgnoreIOSBadge(false);
    pushInfo.setIOSSound("phone_ringing.mp3");
    pushInfo.setAndroidSound("phone_ringing"); //Note: don't add suffix
    //VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
    pushInfo.setAndroidVIVOClassification(1);
}
```

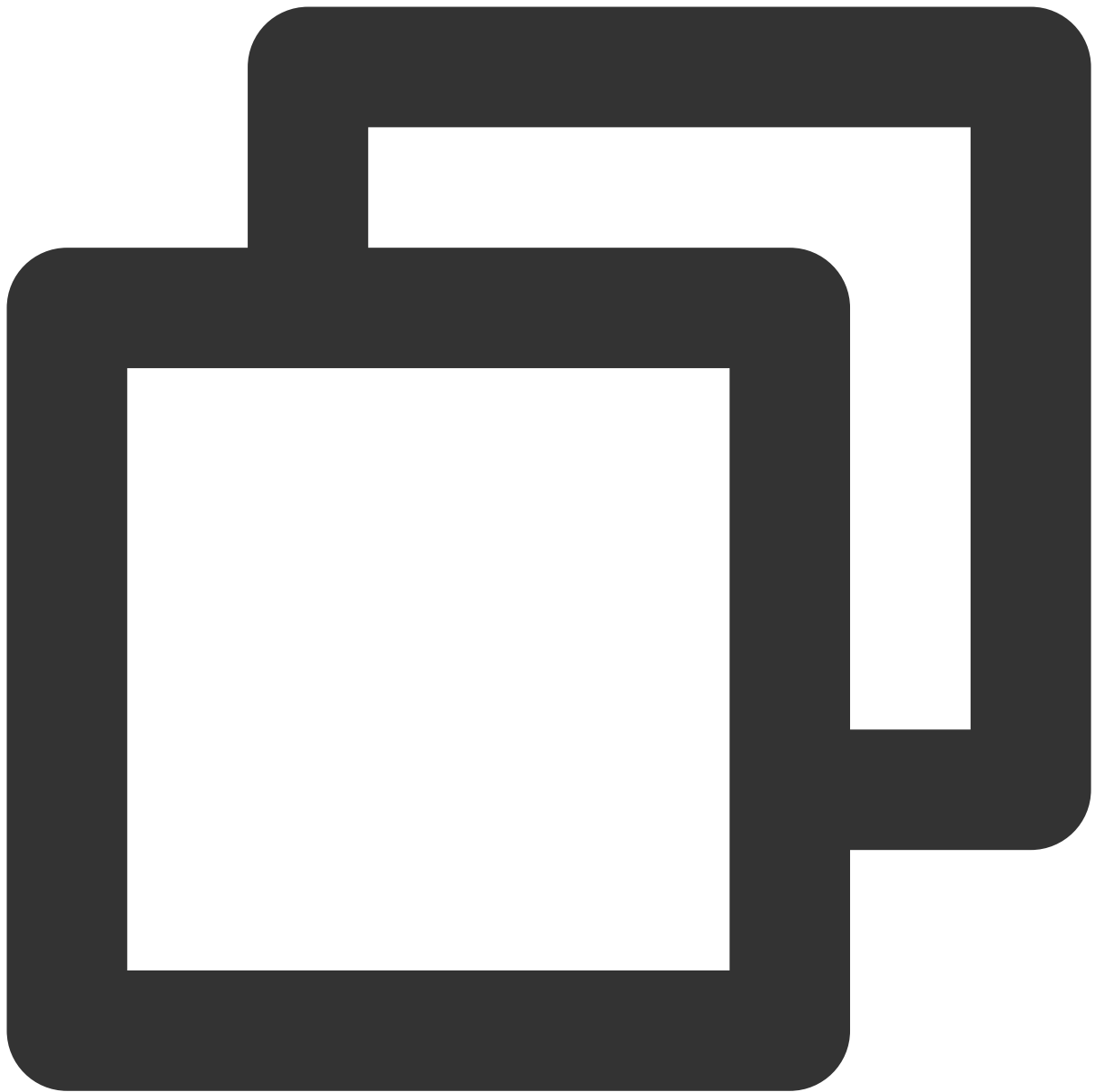
```
//FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannel  
pushInfo.setAndroidFCMChannelID("fcm_push_channel");  
//Huawei message type  
pushInfo.setAndroidHuaWeiCategory("IM");  
//IOS push type: if you want user VoIP, please modify type to TUICallDefine.IOS  
pushInfo.setIOSPushType(TUICallDefine.IOSOfflinePushType.APNs);  
return pushInfo;  
}
```

## Step 4. Customize the offline notification sound

Offline notification sound only supports customization for Huawei, Xiaomi, FCM and APNs. Other manufacturers such as OPPO, VIVO, Honor, etc. are not supported at the moment.

TUICallKit introduces the TUIOfflinePushPush component, which needs to call the following method at the start of the application to enable the ability to customize the ringtone for offline notifications.



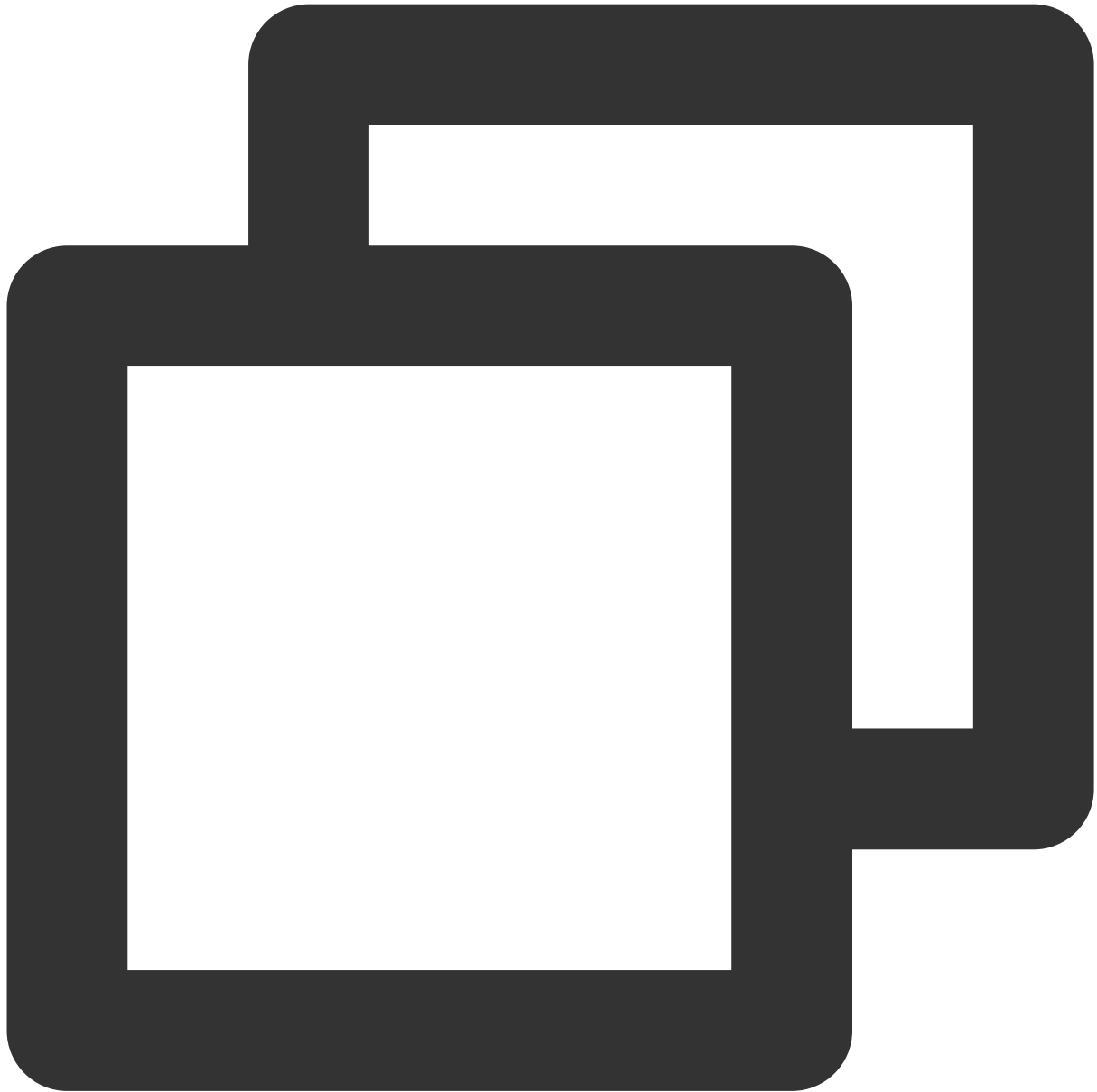


```
public class DemoApplication extends Application {  
    @Override  
    public void onCreate() {  
        TUIOfflinePushConfig.getInstance().setAndroidPrivateRing(true);  
    }  
}
```

## 1. Huawei And APNs

Invoke the interfaces `setAndroidSound()` and `setIOSSound()` .

Customize the ringtone resource file and add it to the res/raw directory of the local Android Studio project. Refer to Step 3 to set ringtones.



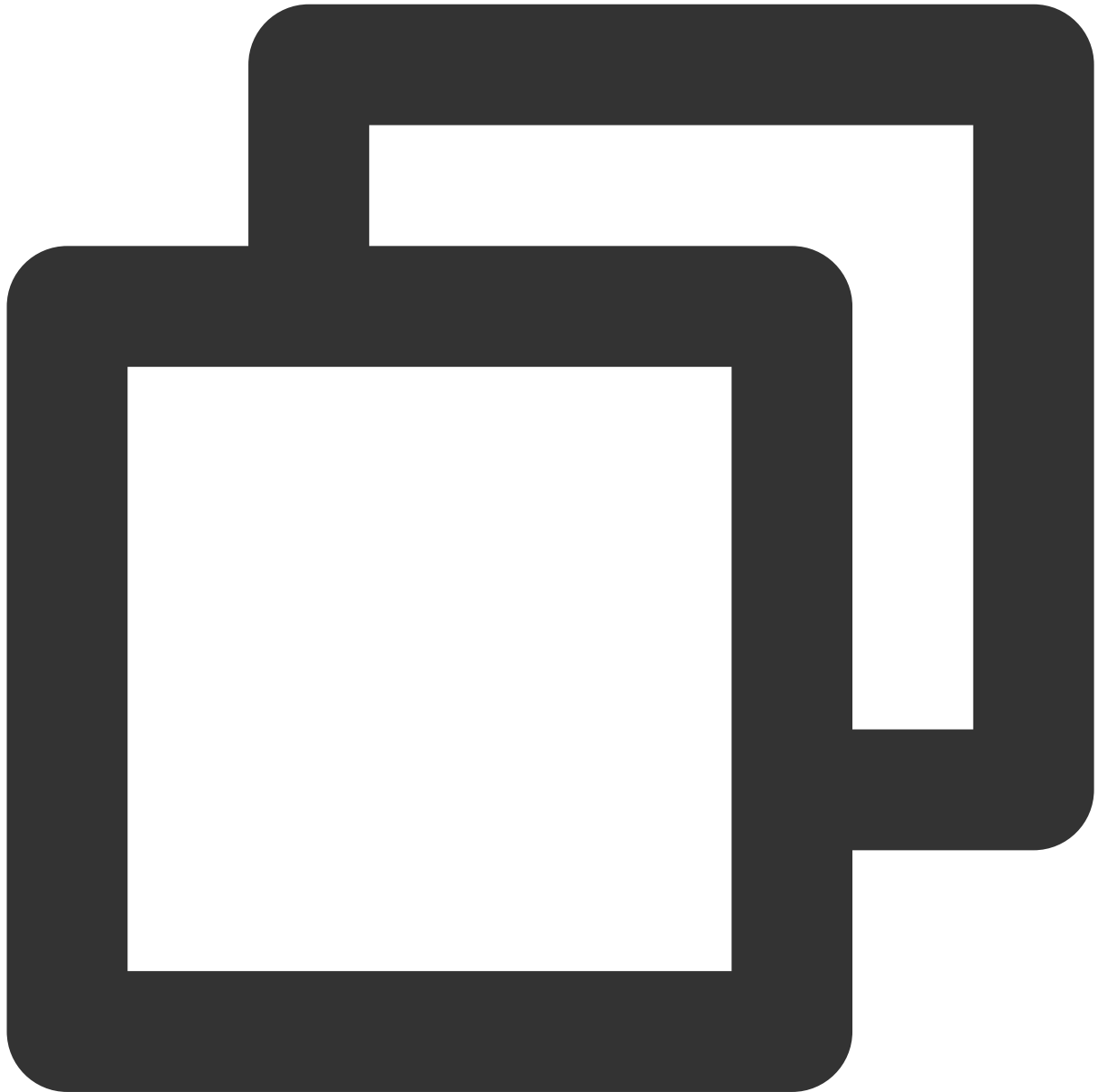
```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();  
pushInfo.setIOSSound("phone_ringing.mp3");  
pushInfo.setAndroidSound("phone_ringing");
```

**Note :**

IMSDK 6.1.2155 and above versions are supported.

## 2. Xiaomi

- (1) Before Android 8.0, invoke the interfaces `setAndroidSound()` and `setIOSSound()`. Customize the ringtone resource file and add it to the `res/raw` directory of the local Android Studio project(Refer to Huawei).
- (2) After Android 8.0, you also need to log in to the Xiaomi vendor console and create a channel with proper configuration, and the ringtone file needs to be added to the local Android Studio project `res/raw` directory.



```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setIOSSound("phone_ringing.mp3");
pushInfo.setAndroidSound("phone_ringing");
pushInfo.setAndroidXiaoMiChannelID("channelID");
```

**Note :**

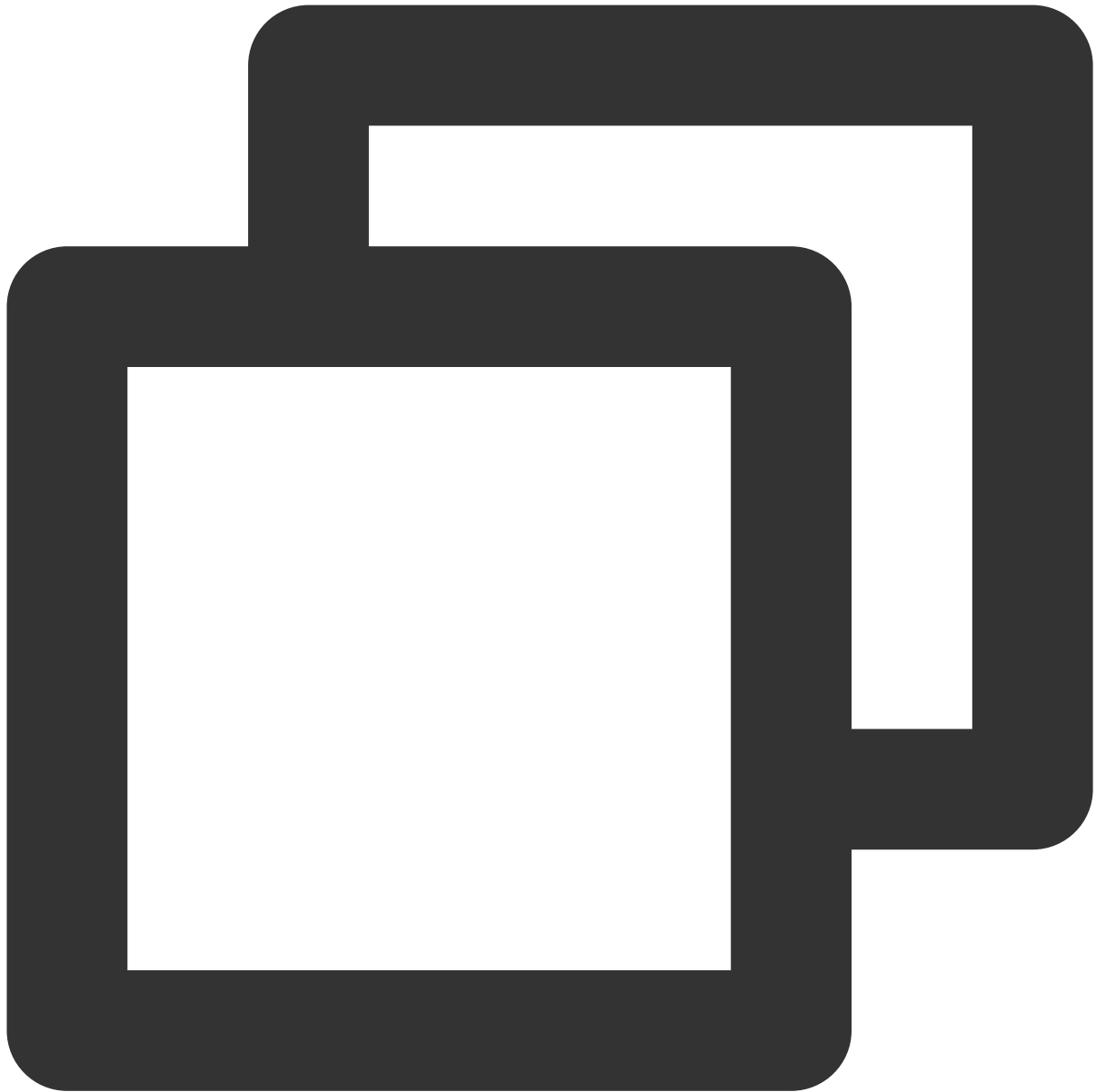
The method Before Android 8.0, IMSDK 6.1.2155 and above versions are supported.

The method After Android 8.0, IMSDK 7.0.3754 and above versions are supported.

### 3. FCM

(1) Before Android 8.0, invoke the interfaces `setAndroidSound()` and `setIOSSound()`. Customize the ringtone resource file and add it to the res/raw directory of the local Android Studio project(Refer to Huawei).

(2) After Android 8.0, FCM needs to configure channelId and ringtone resources. TUIOfflinePush component has handled the playback of custom ringtones. The ringtone file needs to be added to the res/raw directory of your local Android Studio project and specify the name of the ringtone and the name of the channel ID, as shown in [PrivateConstants](#).



```
public class PrivateConstants {  
    // FCM: channel ID  
    public static String fcmPushChannelId = "FCM ChannelID";  
    // FCM: ringtone and don't add suffix  
    public static String fcmPushChannelSoundName = "FCM ringtone";  
}
```

**Note :**

The method Before Android 8.0, IMSDK 6.1.2155 and above versions are supported.

The method After Android 8.0, IMSDK 7.0.3754 and above versions are supported.

FCM only supports custom ringtones or setting channel ID in certificate mode.

## FAQs

If users cannot receive offline push notifications, [contact us](#) for troubleshooting.

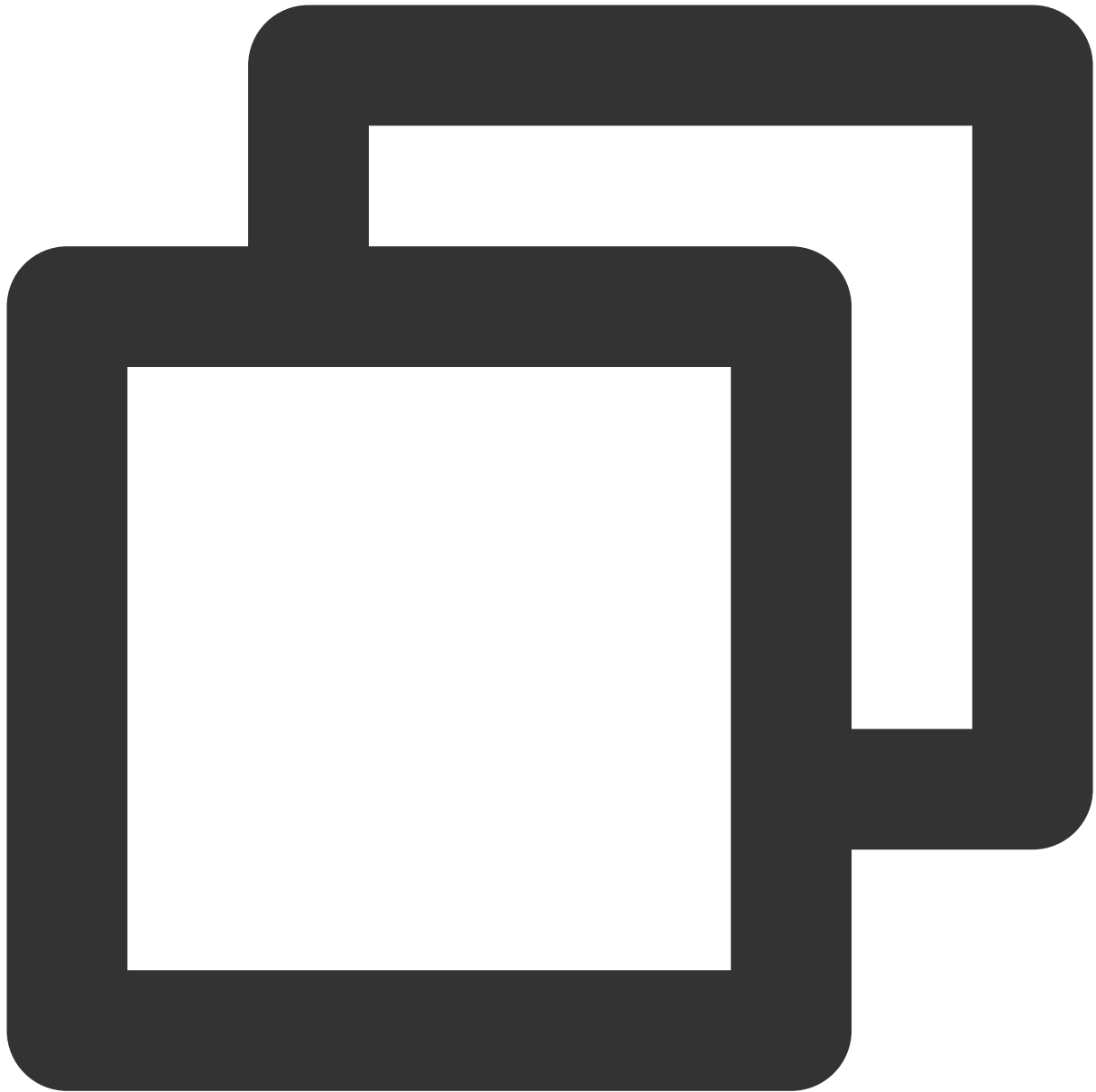
### 1. What should I do if notifications cannot be received?

Test the push service in the vendor console. If notifications can be pushed successfully, the vendor channel is normal.

Then, check whether vendor parameters are configured correctly in the `TUIOfflinePush` console, and if not, enter the parameters as required. (As tested, you must configure the message type in the console for vivo X9).

Some phones will send notifications to the `Unimportant Notifications` folder. In this case, drag down the status bar and check whether the notifications are in the `Unimportant Notifications` folder.

Filter the following log to check whether `TUIOfflinePush` is registered successfully:



TUIOfflinePush

## 2. Why can't the locked screen turn on when a notification is received?

Due to restrictions imposed by the vendor and platform, Android phones require different permissions when their screen is locked. Troubleshoot as follows:

Check whether the system lock screen notification permission is enabled. Some vendors have imposed unified restrictions. For example, if a Mi phone doesn't turn on its screen when receiving offline notifications, select Settings > Lock screen and toggle on Wake Lock screen for notifications.

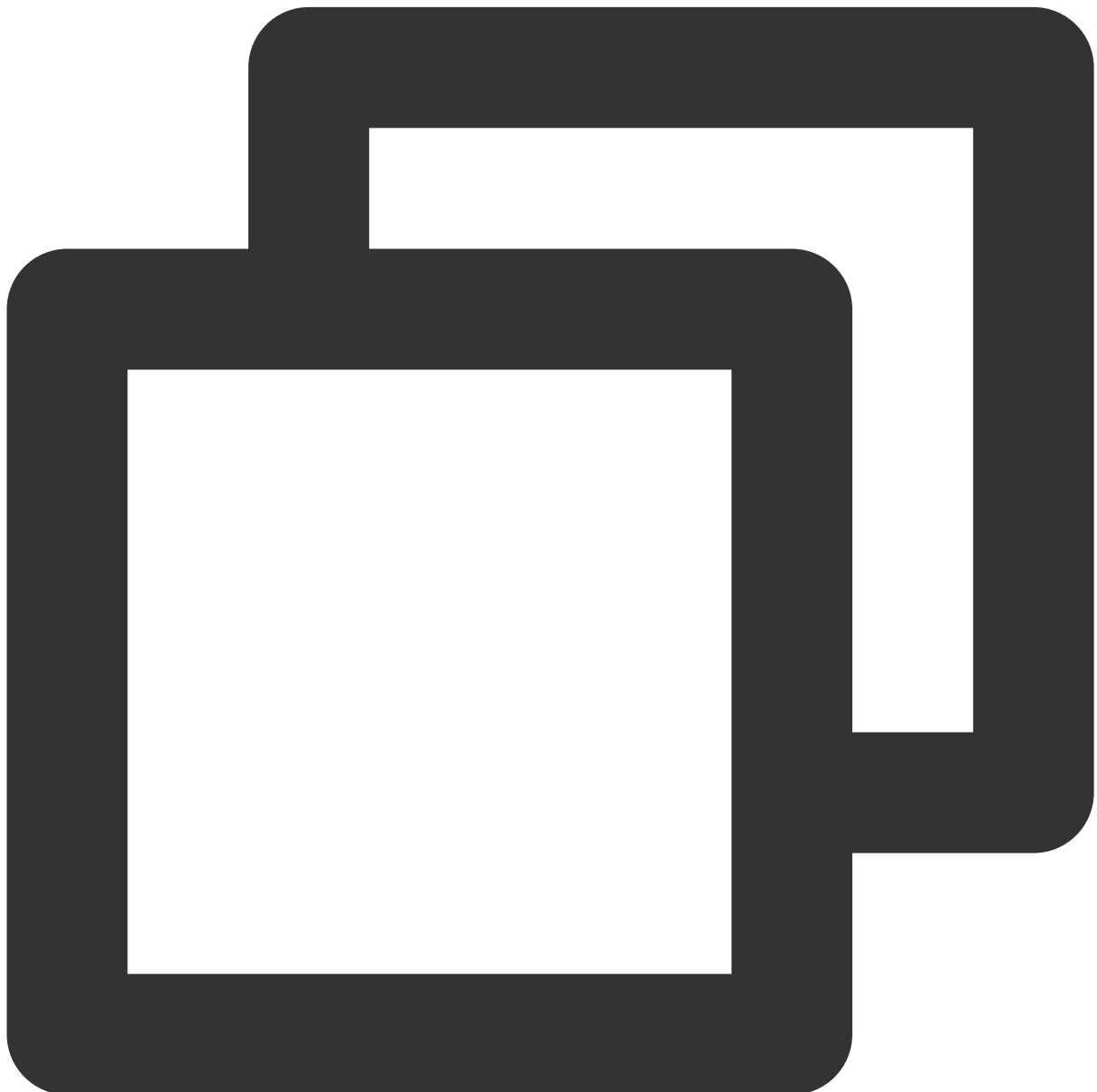
Check whether the lock screen notification permission of the application is enabled. For example, on a Mi phone, you need to toggle on Allow Lock screen notifications.

**Note**

If you need compatibility processing for this problem, contact us for assistance.

**3. Why can't the call UI be pulled when I click an offline push notification?**

Filter the following log to check whether the call request is detected:



```
onReceiveNewInvitation
```



#### 4. Why can't the call UI be pulled to the foreground when the application is running in the background?

To automatically pull the application from the background to the foreground, it is necessary to check whether the "background autostart" or "floating window" permission is enabled on the application.

Note that different vendors or even different Android versions from the same vendor offers different permissions and permission names. For example, you only need to enable the background pop-up window permission for Mi 6, while you need to enable both the background pop-up window and floating window display permissions.

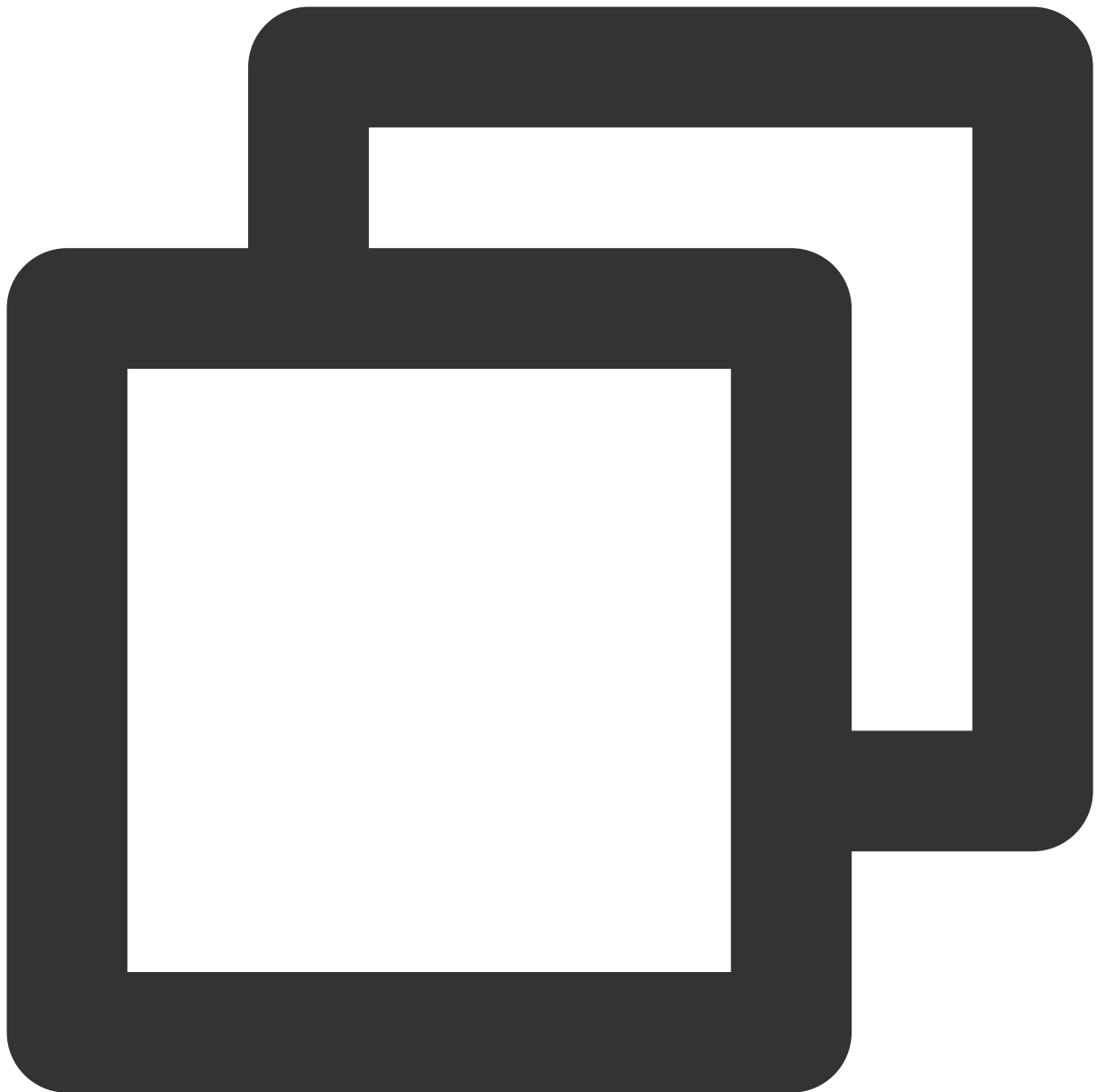
##### Note

If you have manually granted all the necessary permissions and still cannot move your application to the foreground, it may be a vendor compatibility issue.

#### 5. Badge

Currently, only Huawei supports generating badges when receiving offline messages, and other manufacturers do not support it for now. For Huawei, please refer to its official documentation "[Interface Description for Badging on Huawei Desktop](#)".

In the introduced TUIOfflinePush component, this method is also called for setting badges:



```
public void updateBadge(final Context context, final int number) {
    if (BrandUtil.getInstanceType() != TUIOfflinePushConfig.BRAND_HUAWEI) {
        return;
    }
    try {
        Bundle extra = new Bundle();
        extra.putString("package", context.getPackageName());
        extra.putString("class", PrivateConstants.huaweiBadgeClassName); //huaweiBa
        extra.putInt("badgenumber", number);
        context.getContentResolver().call(Uri.parse("content://com.huawei.android.1
    } catch (Exception e) {
```

```
}  
}
```

If you want to clear the badge, you can call this method at the appropriate time to set the badge to 0.

# Flutter

Last updated : 2024-03-12 14:51:07

The offline wake-up function allows your app to receive ringtone calls for audio and video calls while running in the background or in offline mode. TUICallKit Flutter relies on the push capabilities provided by manufacturers: Apple, Google, Vivo, and Xiaomi for message notifications.

## Configure offline push

### iOS

iOS platform's offline wake-up support for **APNs** and **VoIP**, please refer to the specific configuration process:

[APNs offline wake-up configuration process](#)

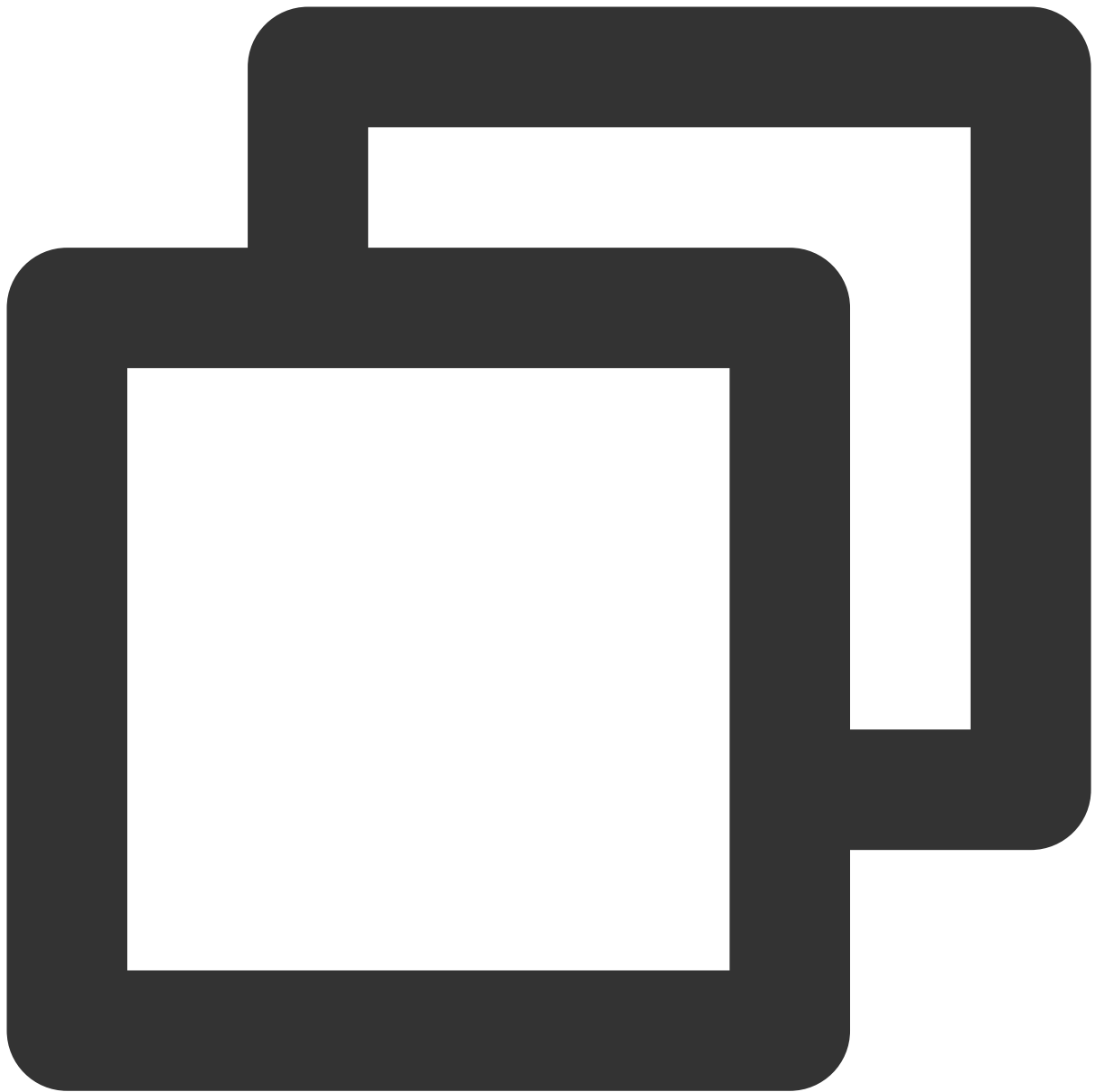
[VoIP offline wake-up configuration process](#)

### Android

For the specific configuration process on the Android platform, please refer to: [Offline Push \(Flutter\)](#).

## Customize offline message content

After the basic offline push configuration is successful, you can receive incoming call push notifications normally. If you need to customize the content of offline messages, you can use the params in the call and groupcall interfaces, and the `TUIOfflinePushInfo` to customize notification content, including title, content, and offline message ringtone. Specific usage as follows:



```
Future<void> call(String userId, TUICallMediaType callMediaType, [TUICallParams? pa
```

```
class TUICallParams {  
    TUIOfflinePushInfo offlinePushInfo;  
}
```

```
class TUIOfflinePushInfo {  
    String? title;  
    String? desc;  
    bool? ignoreIOSBadge;  
    String? iOSSound;
```

```
String? androidSound;  
String? androidOPPOChannelID;  
int? androidVIVOClassification;  
String? androidXiaoMiChannelID;  
String? androidFCMChannelID;  
String? androidHuaWeiCategory;  
bool? isDisablePush;  
TUICallIOSOfflinePushType? iOSPUSHType;  
}}  
  
enum TUICallIOSOfflinePushType {  
    APNs,  
    VoIP,  
}  
}
```

## Customize offline message ringtone

### iOS

Modify the params in the call and groupcall interfaces, and set the `TUIOfflinePushInfo.iOSSound` for the offline message ringtone on the iOS platform.

### Android

#### FCM

After completing step 1: [Configure offline push](#), you can receive message push notifications even when offline, but it is the default ringtone. If you need to support custom ringtones, you need to perform the following operations.

1. In the IM console-basic configuration, select Android native offline push configuration > select Google certificate for editing, add ChannelID, as shown in the following figure:

### Add Android Certificate ✕

Push Platform ☐ Mi ☐ Huawei ☒ Google ☐ Meizu ☐ Vivo ☐ OPPO ☐ Honor

Adding Method ☒ Upload certificate ☐ Enter the server key

Upload certificate 

Upload Again

Delete

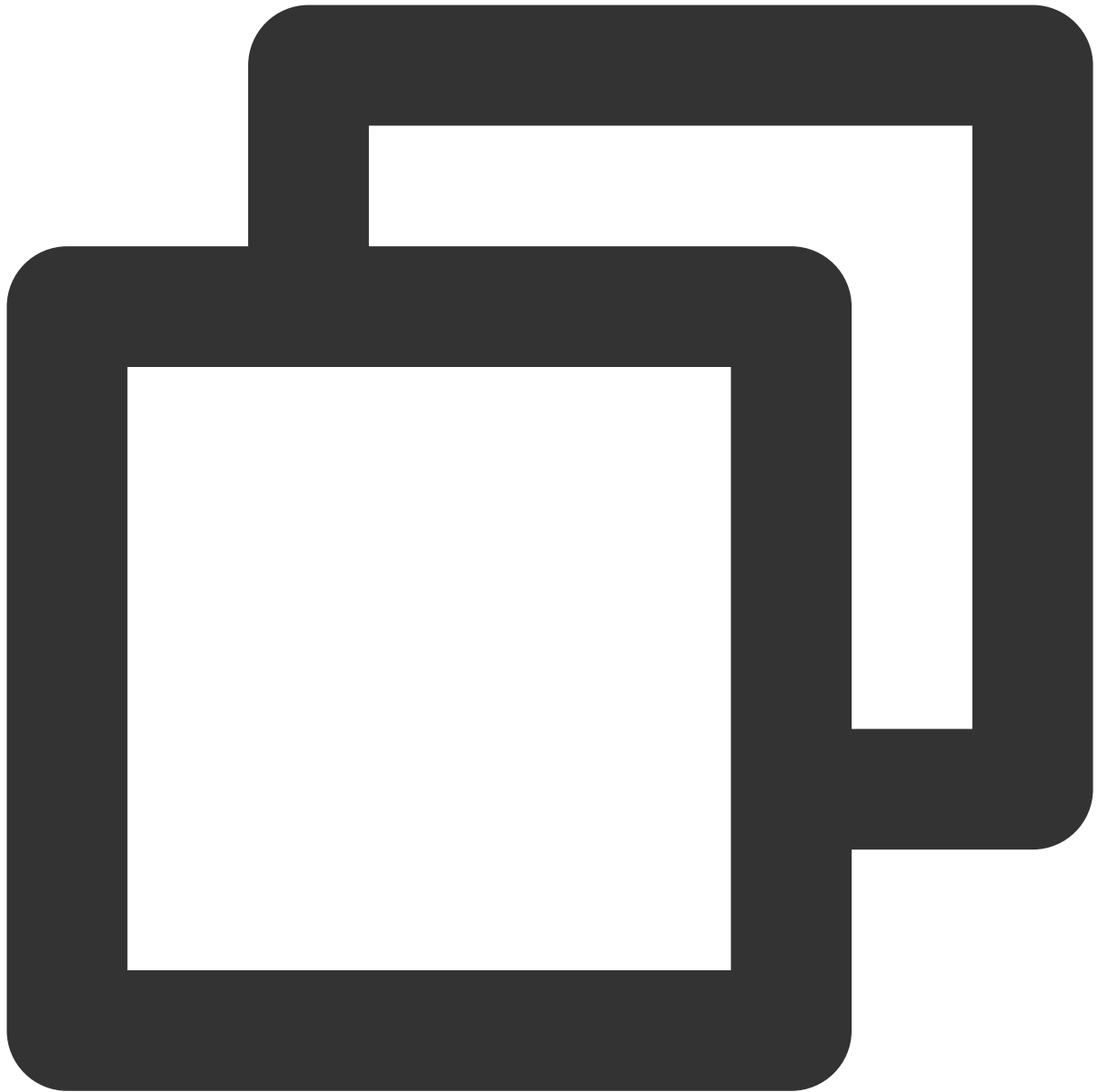
[How to Generate an FCM certificate](#)

ChannelID

Confirm

2. Add custom ringtone files to the `android/app/src/main/res/raw/` directory of your project, such as `"incoming_call.mp3"`.

3. After a successful login, use the `tim_ui_kit_push_plugin` plugin to register a `NotificationChannel` for playing custom ringtones to the Android system. Sample code as follows:



```
void createNotificationChannel() {  
    AndroidNotificationChannelGroup group = new AndroidNotificationChannelGroup("Grou  
    AndroidFlutterLocalNotificationsPlugin plugin = AndroidFlutterLocalNotificationsP  
    plugin.createNotificationChannelGroup(group);  
  
    AndroidNotificationChannel channel = AndroidNotificationChannel(  
        "fcm_push_channel", "ChannelName",  
        groupId: "GroupID",  
        importance: Importance.high,  
        enableLights: true,  
        enableVibration: true,
```



```
        sound: RawResourceAndroidNotificationSound("incoming_call")); // Note: Media
        plugin.createNotificationChannel(channel);
    }
```

# On-Cloud Recording (TUICallKit)

Last updated : 2024-04-03 17:23:11

This document describes how to enable on-cloud recording of `TUICallKit` to help you archive and review important calls. Here, two schemes are provided: [automatic recording](#) and [RESTful API-based recording](#).

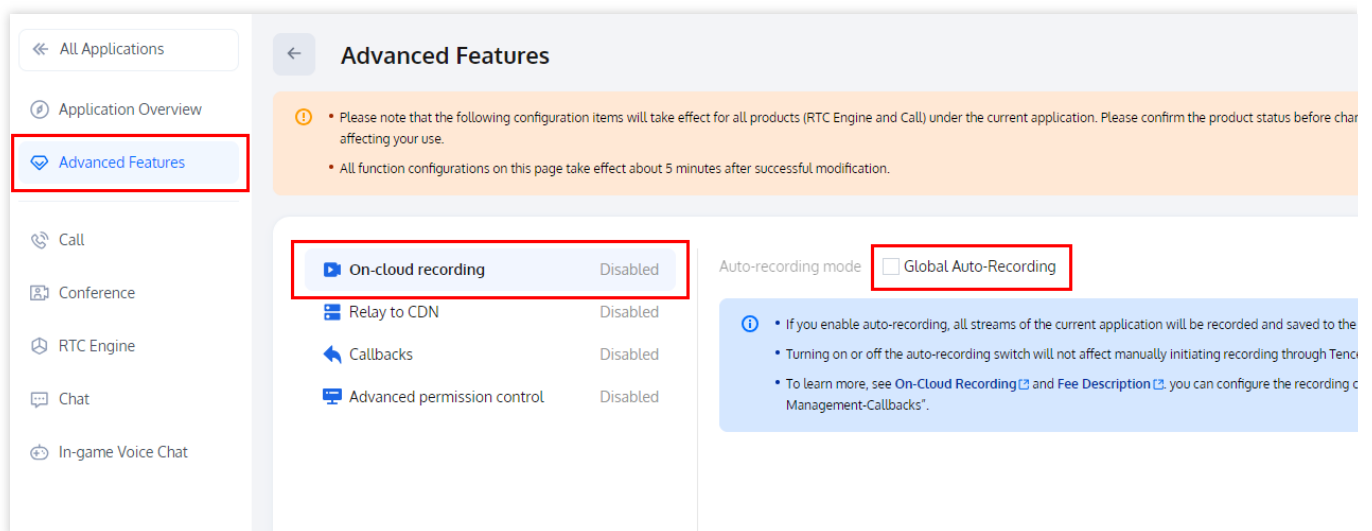
## explain

`TUICallKit` integrates multiple basic Tencent Cloud PaaS services, and its audio/video capabilities rely on [TRTC](#). To use the on-cloud recording feature of `TUICallKit`, you need to configure it in the [TRTC console](#).

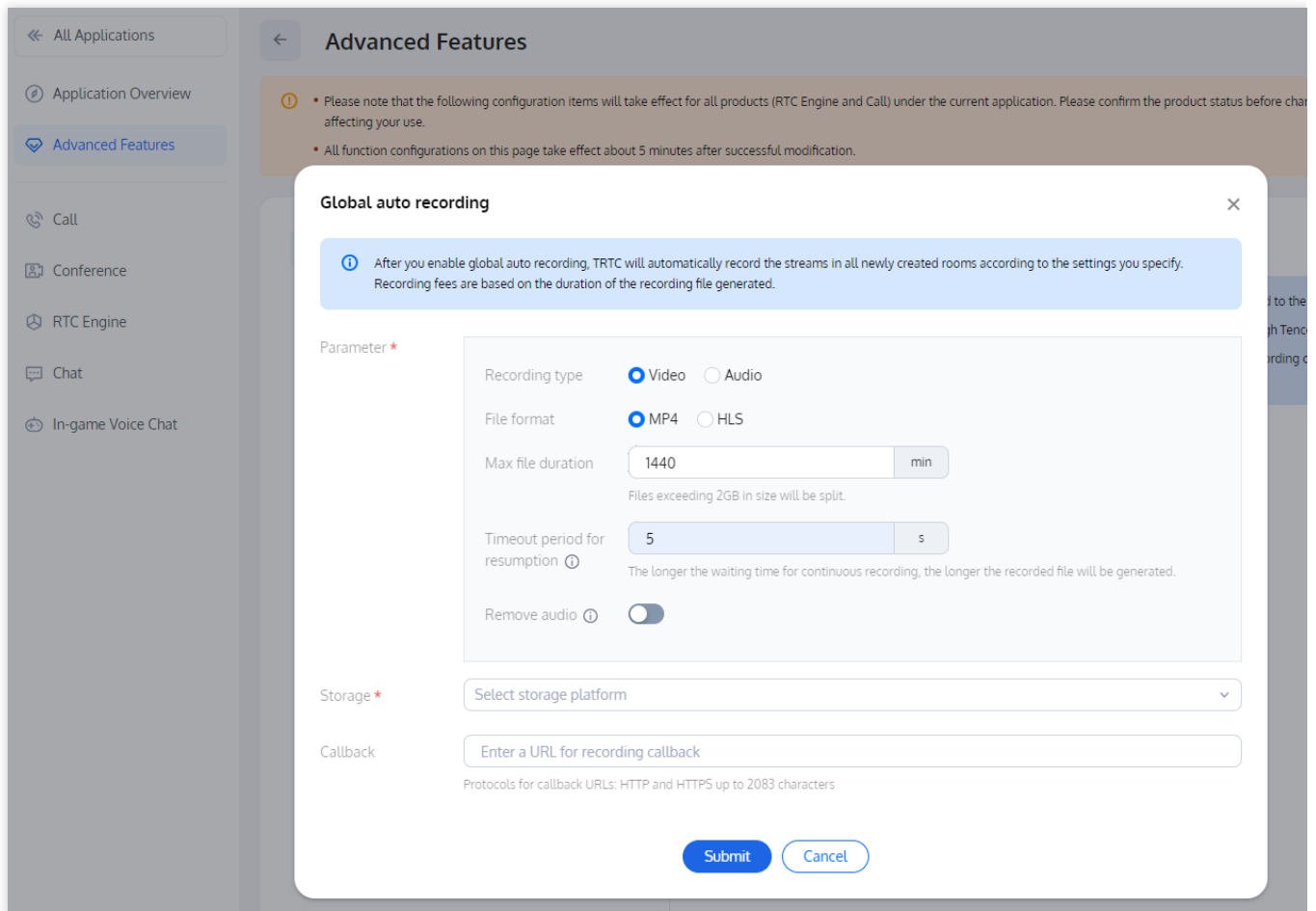
## Scheme 1. Automatic Recording (Recommended)

We recommend you use the automatic recording scheme. In this scheme, recording is not manually started or stopped by you. Instead, recording tasks are managed on the TRTC backend, and a call will be recorded automatically when there is an audio/video stream being upstreamed. You can integrate it quickly and easily as follows:

1. Find the application of the target `SDKAppId` in [Application Management](#) in the TRTC console and enter the **Advanced Feature** page.
2. In the **Advanced Function** configuration page, you can see the option for **On-cloud recording** configuration. Click on **Global Auto-Recording** to enter the configuration popup window.



3. We recommend the following configuration parameter settings for audio/video call business scenarios such as one-to-one and group calls. You can also customize the recording template as needed.



### notice

Global recording can mix the streams of up to eight users. If a call involves more than eight users (including the local user), the stream of the last user cannot be recorded.

After the global automatic recording feature is enabled, when a call is answered and there is audio/video being upstreamed, a recording task will be triggered automatically, and recording will stop automatically when the call ends. If a user leaves the room due to poor network conditions or other exceptions, the recording backend will automatically stop the recording task based on the configured `MaxIdleTime` value (maximum idle time, which is five seconds by default) to avoid incurring unnecessary fees.

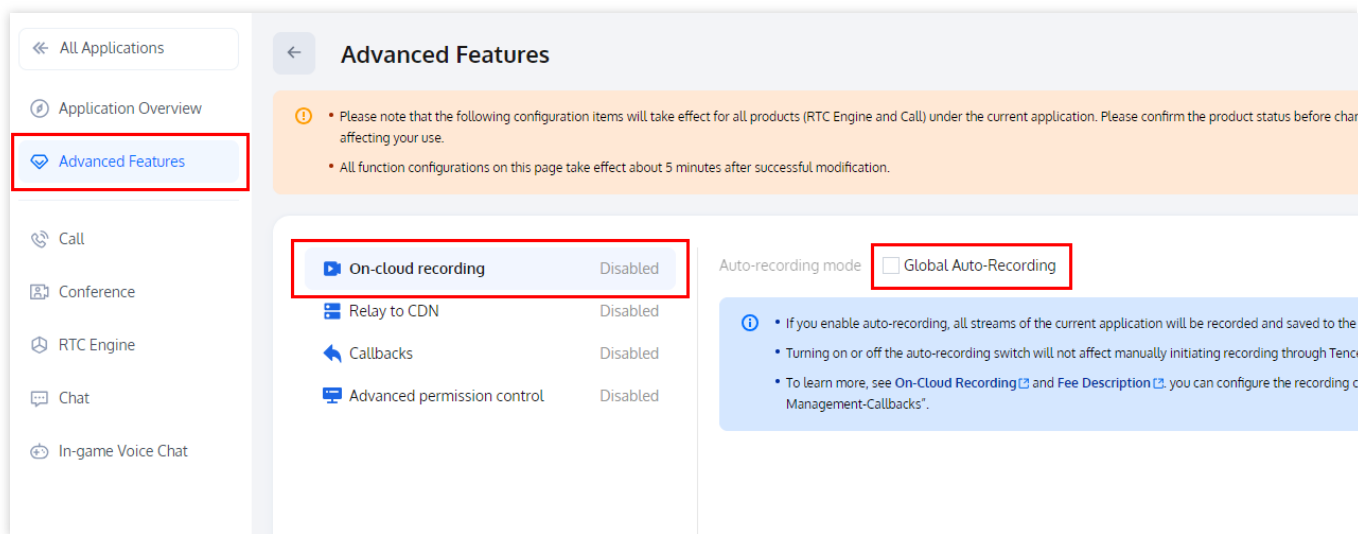
4. After the template is created, select **Global Auto-Recording**.

## Scheme 2. RESTful API-Based Recording

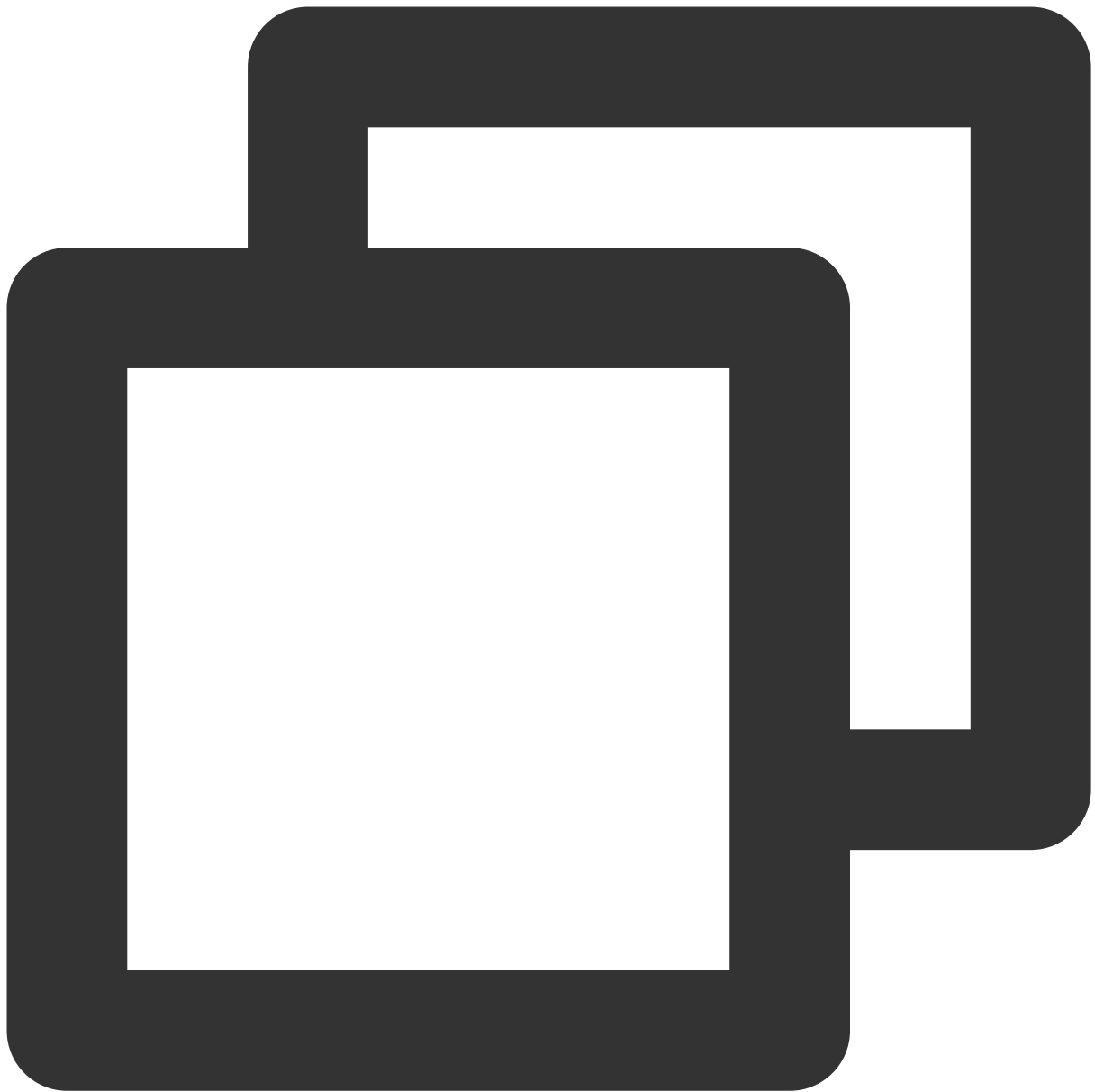
If the automatic recording scheme doesn't meet your needs, you can also use the more flexible RESTful API-based recording scheme. In this scheme, you can record and subscribe to a specified anchor in the room, customize the layout of mixed streams, and update the layout and subscription during recording. However, **using its features**

requires using it together with the business backend service and performing complex integration operations:

1. Find the application of the target `SDKAppId` in [Application Management](#) in the TRTC console and enter the **Advanced Feature** page.
2. In the **Advanced Function** configuration page, you can see the option for **On-cloud recording** configuration. Click on **Global Auto-Recording** to enter the configuration popup window. **Manual custom recording**, i.e., the RESTful API-based recording mode, is selected by default.



3. Then, you can call the RESTful API [CreateCloudRecording](#) to start on-cloud recording. Here, we recommend you listen on the notification event of [TUICallObserver](#) to start recording when an audio/video call starts. Below is the Java code sample:



```
TUICallEngine.createInstance(context).addObserver(new TUICallObserver() {  
    @Override  
    public void onCallBegin(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType  
        // Tell your business backend to start a recording task by using the releva  
    }  
});
```

4. Because a call may be hung up on the client due to an exception such as poor network conditions or process termination, to stop recording, we recommend you subscribe to the callback for the TRTC room status (for more

information, see [Event Callbacks](#)) and call the RESTful API [DeleteCloudRecording](#) to stop the on-cloud recording task when receiving the callback for TRTC room dismissal.



## FAQs

### 1. How do I view the detailed recording durations?

You can view the detailed recording durations on the [On-Cloud Recording](#) page in the TRTC console.

### 2. How do I view recorded files?

Log in to the [VOD console](#), select **Video/Audio Management** on the left sidebar, click **Search by prefix** above the list, select **Search by prefix**, and enter the keyword in the search box. The recording filenames are in the following formats:

The filename format of an MP4 single-stream recording file:

```
<SdkAppId>_<RoomId>_UserId_s_<UserId>_UserId_e_<MediaId>_<Index>.mp4
```

The filename format of an MP4 mixed-stream recording file: `<SdkAppId>_<RoomId>_<Index>.mp4`

# Additional Features (TUICallKit)

## Configuring Nicknames and Avatars (All Platform)

Last updated : 2024-04-03 17:23:11

This article explains how to set up a user's avatar and nickname.

### Setting Avatar, Nickname

To customize the nickname or profile photo, use the following API for update:

Android (Kotlin)

Android (Java)

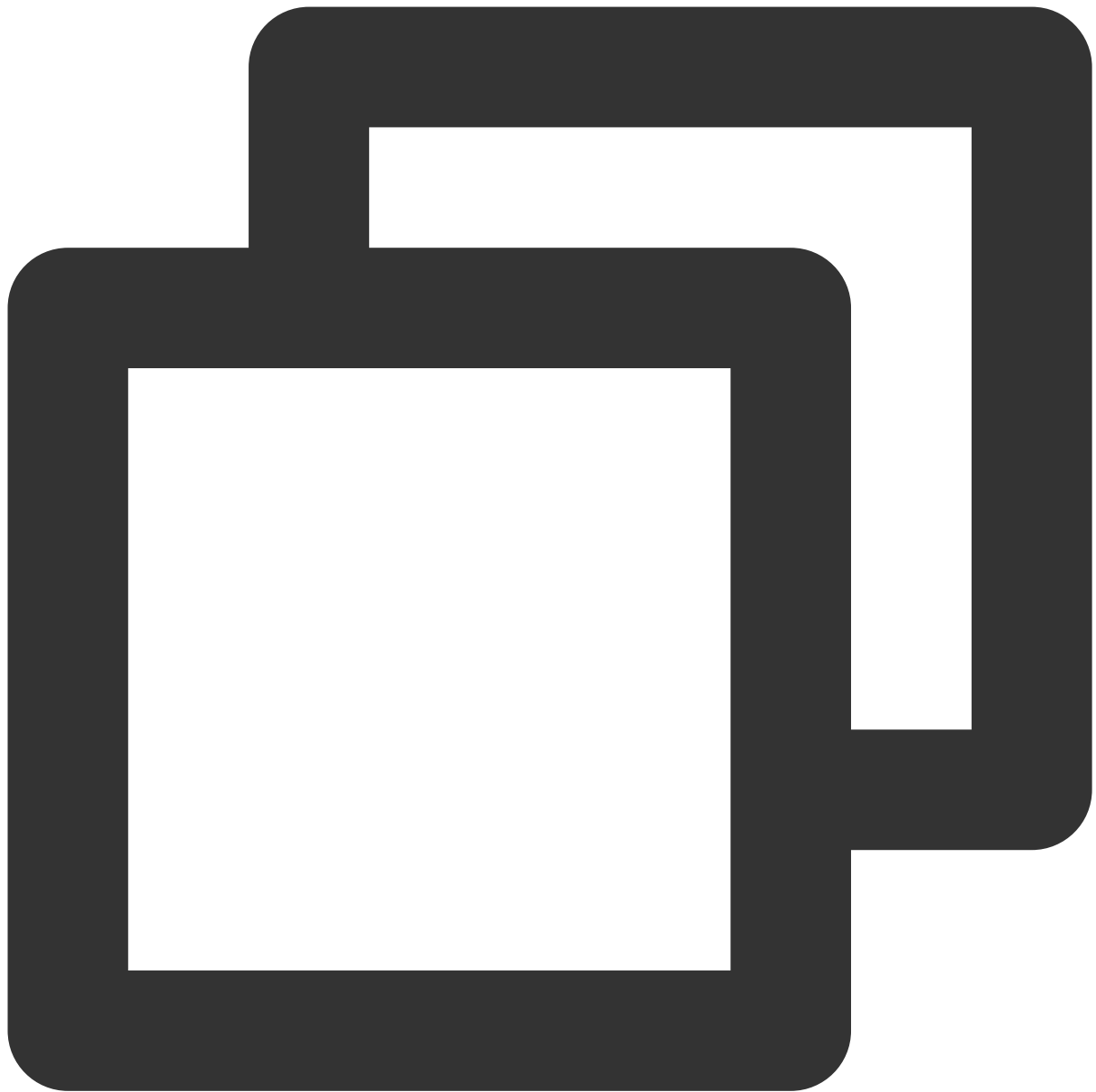
iOS (Objective-C)

iOS (Swift)

Flutter (Dart)

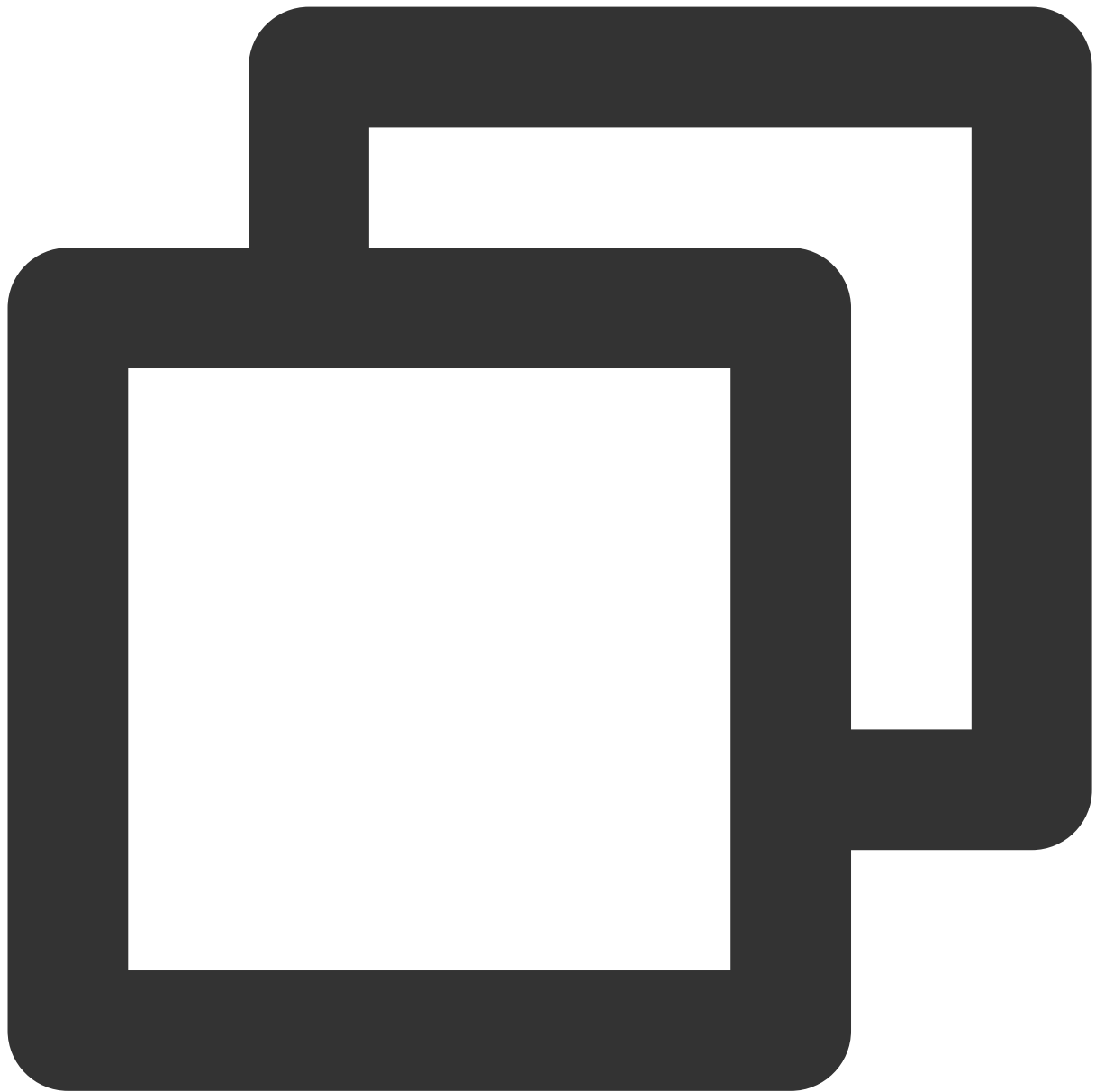
uni-app(Android&iOS)

Web&H5



```
TUICallKit.createInstance(context).setSelfInfo("jack", "https://****/user_avatar.png")
```





```
itUITCallKit.createInstance(context).setSelfInfo("jack", "https://****/user_avatar.pn
```



```
[[TUICallKit sharedInstance] setSelfInfo:@"" avatar:@"" succ:^(  
    } fail:^(int code, NSString *errMsg) {  
    }];
```



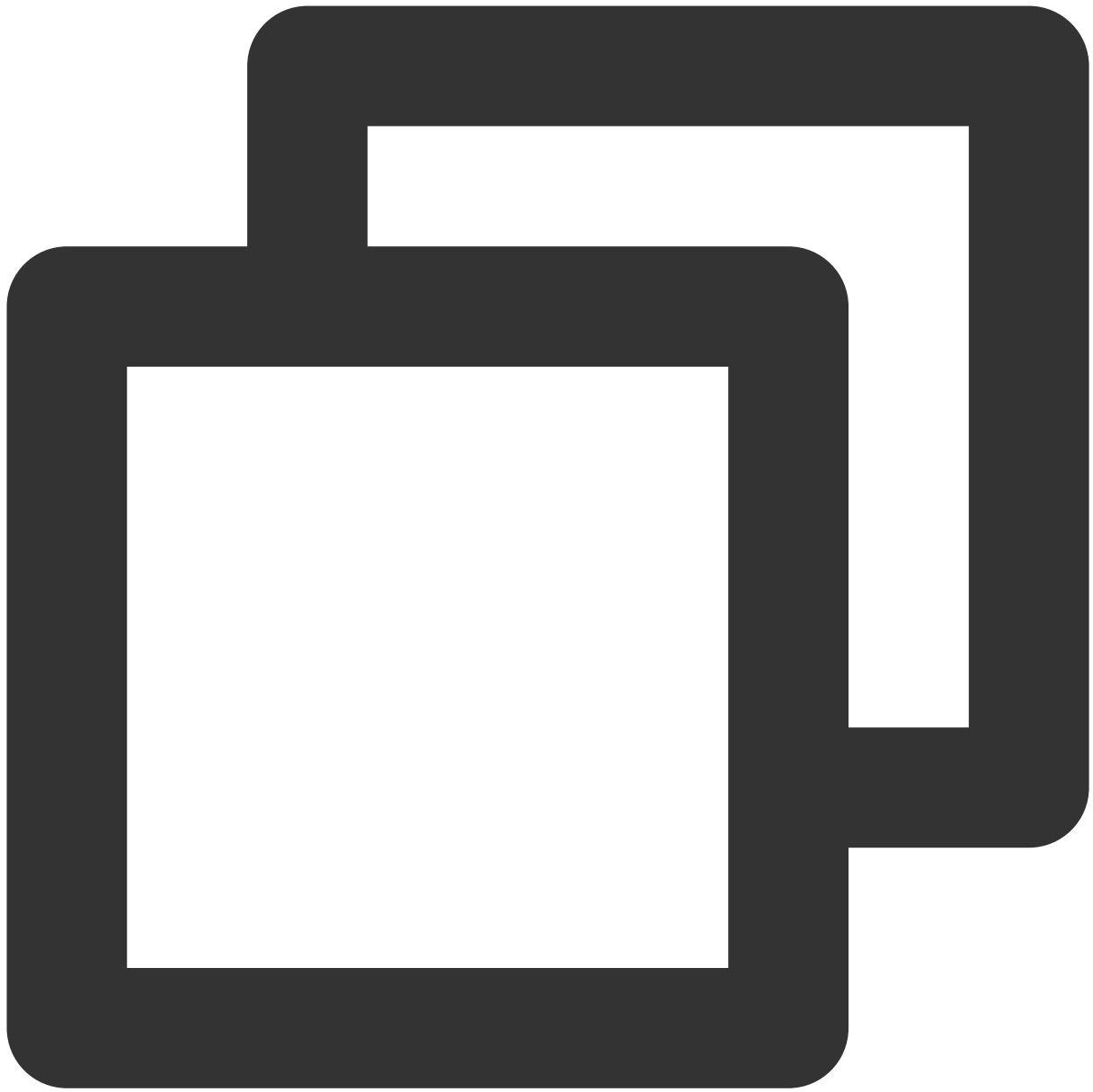
```
TUICallKit.createInstance().setSelfInfo(nickname: "", avatar: "") {  
  } fail: { code, message in  
}
```



```
TUIResult result = TUICallKit.instance.setSelfInfo('userName', 'url:*****');
```



```
const options = {
  nickName: 'jack',
  avatar: 'https://****/user_avatar.png'
}
TUICallKit.setSelfInfo(options, (res) => {
  if (res.code === 0) {
    console.log('setSelfInfo success');
  } else {
    console.log(`setSelfInfo failed, error message = ${res.msg}`);
  }
});
```



```
try {
  await TUICallKitServer.setSelfInfo({ nickName: "jack", avatar: "http://xxx" });
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the setSelfInfo API. Reason: ${error}`);
}
```

**Note**

The update of the callee's nickname and profile photo may be delayed during a call between non-friend users due to the user privacy settings. After a call is made successfully, the information will also be updated properly in subsequent

calls.

# Configure Resolution and Fill Mode (Web)

Last updated : 2024-04-03 17:23:11

Introduction to how to set resolution, fill pattern.

## Setting resolution, fill pattern

The TUICallKit component provides numerous feature switches, allowing for selective enabling or disabling as needed. For specifics, refer to [Introduction to TUICallKit Attributes](#) .

`VideoDisplayMode` : Display mode of the frame.

`VideoResolution` : Call resolution.

React

Vue





```
import { VideoDisplayMode, VideoResolution } from "@tencentcloud/call-uikit-react";

<TUICallKit
  videoDisplayMode={VideoDisplayMode.CONTAIN}
  videoResolution={VideoResolution.RESOLUTION_1080P} />
```



```
import { VideoDisplayMode, VideoResolution } from "@tencentcloud/call-uikit-vue";

<TUICallKit
  :videoDisplayMode="VideoDisplayMode.CONTAIN"
  :videoResolution="VideoResolution.RESOLUTION_1080P" />
```

## videoDisplayMode

There are three values for the `videoDisplayMode` display mode:

`VideoDisplayMode.CONTAIN`

`VideoDisplayMode.COVER``VideoDisplayMode.FILL` , the default value is `VideoDisplayMode.COVER` .

| Attribute        | Value                    | Description   |
|------------------|--------------------------|---|
| videoDisplayMode | VideoDisplayMode.CONTAIN | Ensuring the full display of video content is our top priority.<br>The dimensions of the video are scaled proportionally until one side aligns with the frame of the viewing window.<br>In case of discrepancy in sizes between the video and the display window, the video is scaled - on the premise of maintaining the aspect ratio - to fill the window, resulting in a black border around the scaled video. |
|                  | VideoDisplayMode.COVER   | Priority is given to ensure that the viewing window is filled.<br>The video size is scaled proportionally until the entire window is filled.<br>If the video's dimensions are different from those of the display window, the video stream will be cropped or stretched to match the window's ratio.  |
|                  | VideoDisplayMode.FILL    | Ensuring that the entire video content is displayed while filling the window does not guarantee preservation of the original video's proportion.<br>The dimensions of the video will be stretched to match those of the window.   |

## videoResolution

The resolution `videoResolution` has three possible values:

`VideoResolution.RESOLUTION_480P``VideoResolution.RESOLUTION_720P``VideoResolution.RESOLUTION_1080P` , the default value is `VideoResolution.RESOLUTION_480P` .

### Resolution Explanation:

| Video Profile | Resolution (W x H) | Frame Rate (fps) | Bitrate (Kbps) |
|---------------|--------------------|------------------|----------------|
| 480p          | 640 × 480          | 15               | 900            |
| 720p          | 1280 × 720         | 15               | 1500           |
| 1080p         | 1920 × 1080        | 15               | 2000           |

**Frequently Asked Questions:**

iOS 13&14 does not support encoding videos higher than 720P. It is suggested to limit the highest collection to 720P on these two system versions. Refer to [iOS Safari known issue case 12](#).

Firefox does not permit the customization of video frame rates (default is set to 30fps).

Due to the influence of system performance usage, camera collection capabilities, browser restrictions, and other factors, the actual values of video resolution, frame rate, and bit rate may not necessarily match the set values exactly. In such scenarios, the browser will automatically adjust the Profile to get as close to the set values as feasible.

# Group Calls

## Android&iOS&Flutter

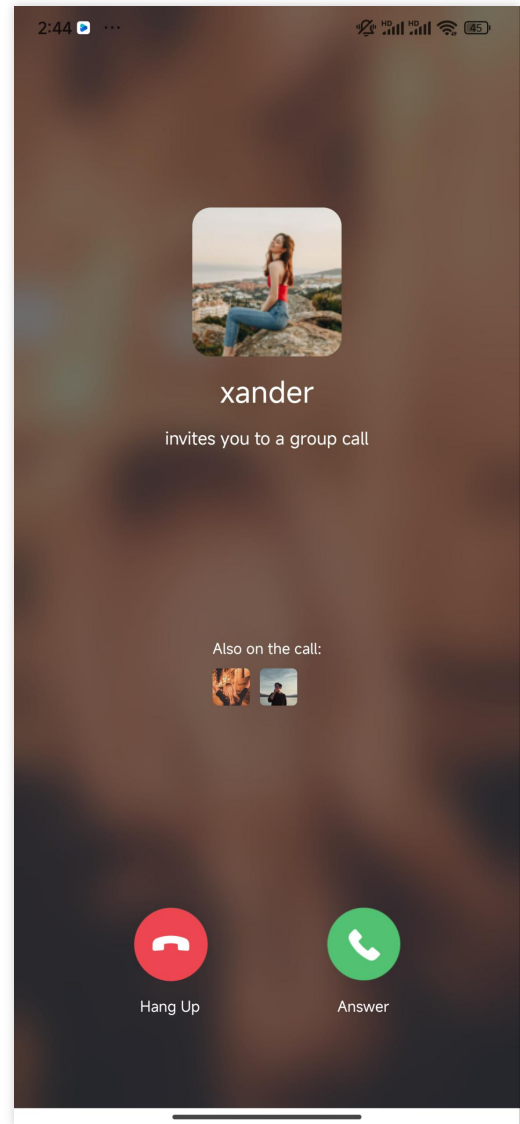
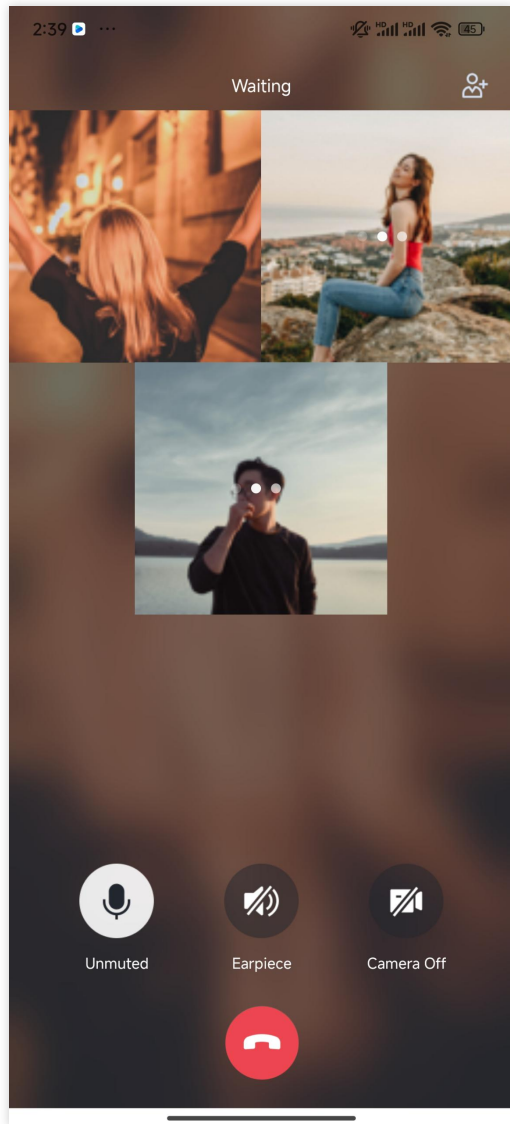
Last updated : 2024-04-03 17:23:11

This article introduces the use of the group call feature, such as initiating a group call and joining a group call.

### Expected outcome

TUICallKit supports group calls. The expected outcome is shown in the figure below.

| Initiate a group call | Receive a group call request |
|-----------------------|------------------------------|
|                       |                              |



## Create groupID

Before using the group call feature, you need to create a group first and initiate a group call in an existing group.

Method one: Create a group by calling the IM API, see [IM Group Management](#).

Method two: Create a group manually through the console, see [Console group management](#).

## Group call

### Initiate a group call

Launch a group call using the groupCall API.

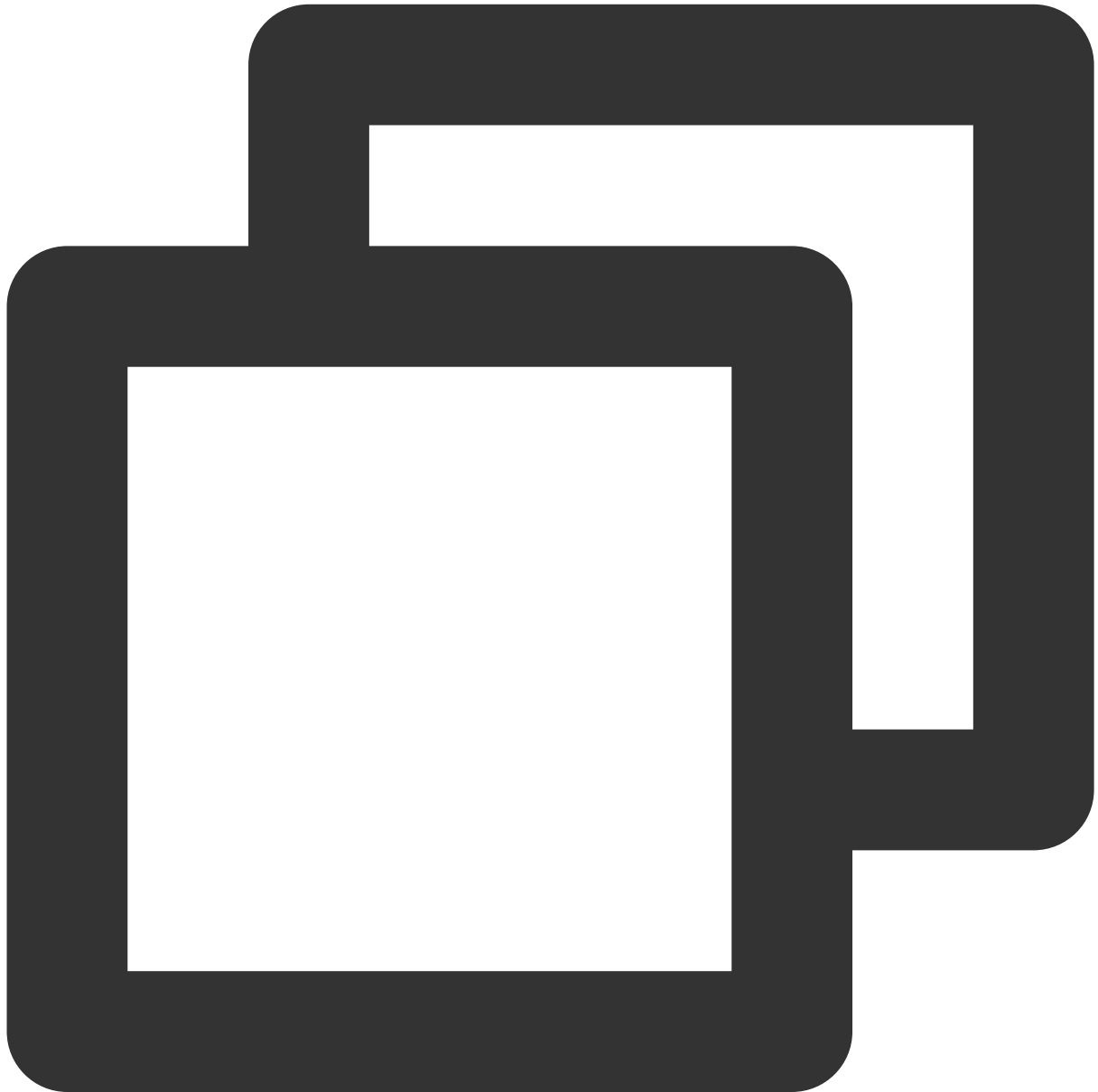
Android(Kotlin)

Android(Java)

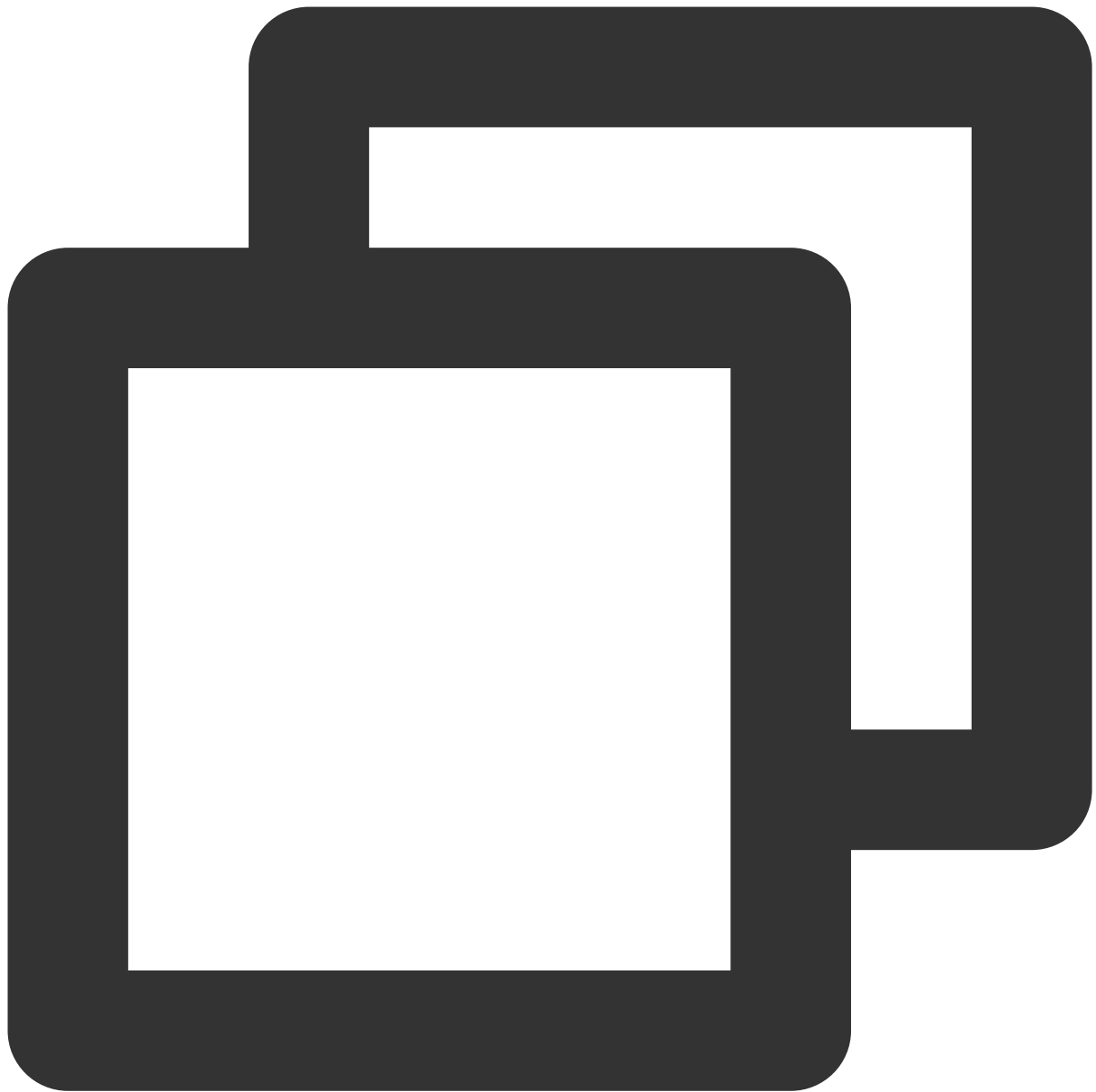
iOS(Swift)

iOS(Objective-C)

Flutter(Dart)

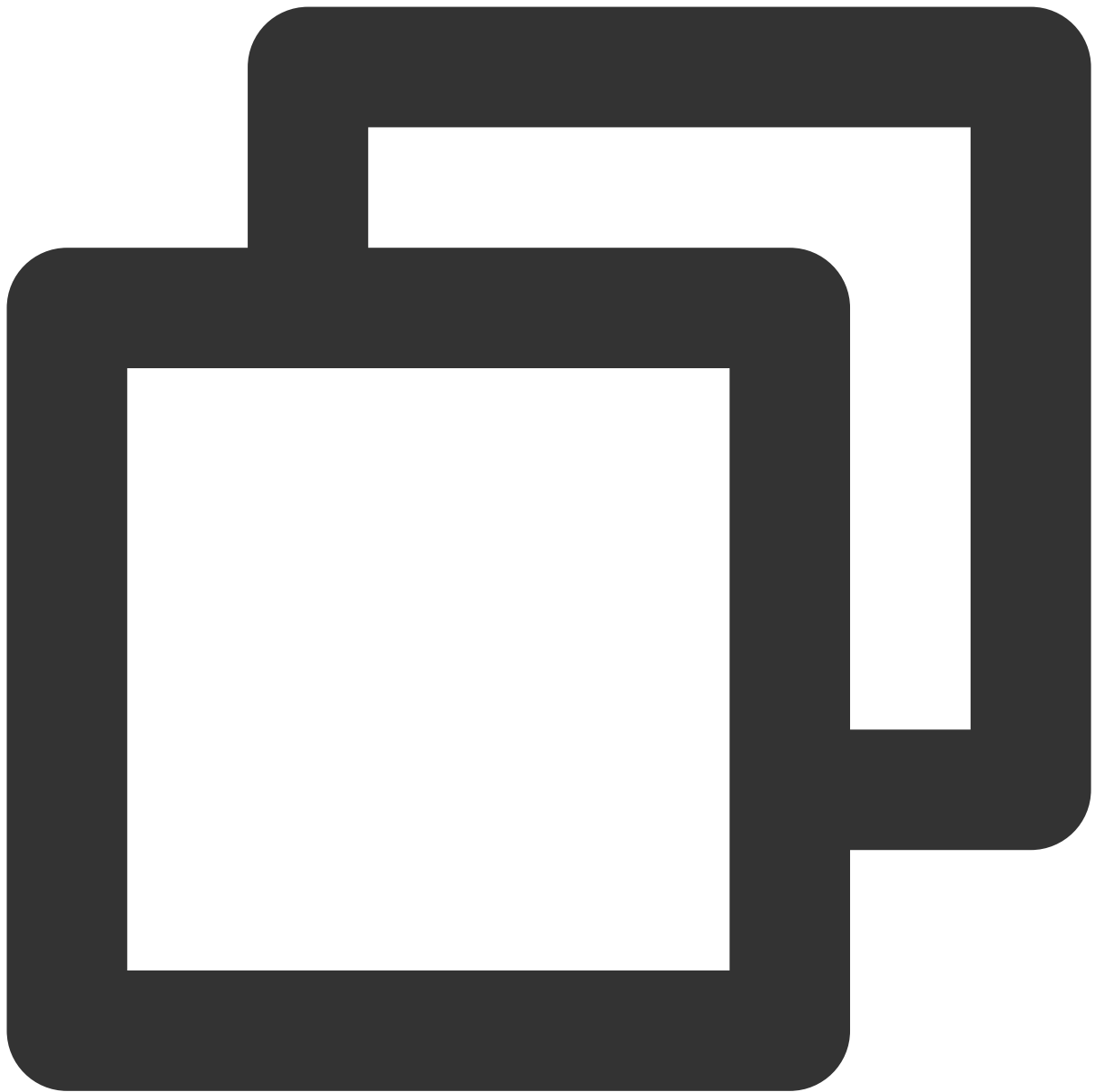


```
TUICallKit.createInstance(context).groupCall("12345678", Arrays.asList("jane", "mik
```



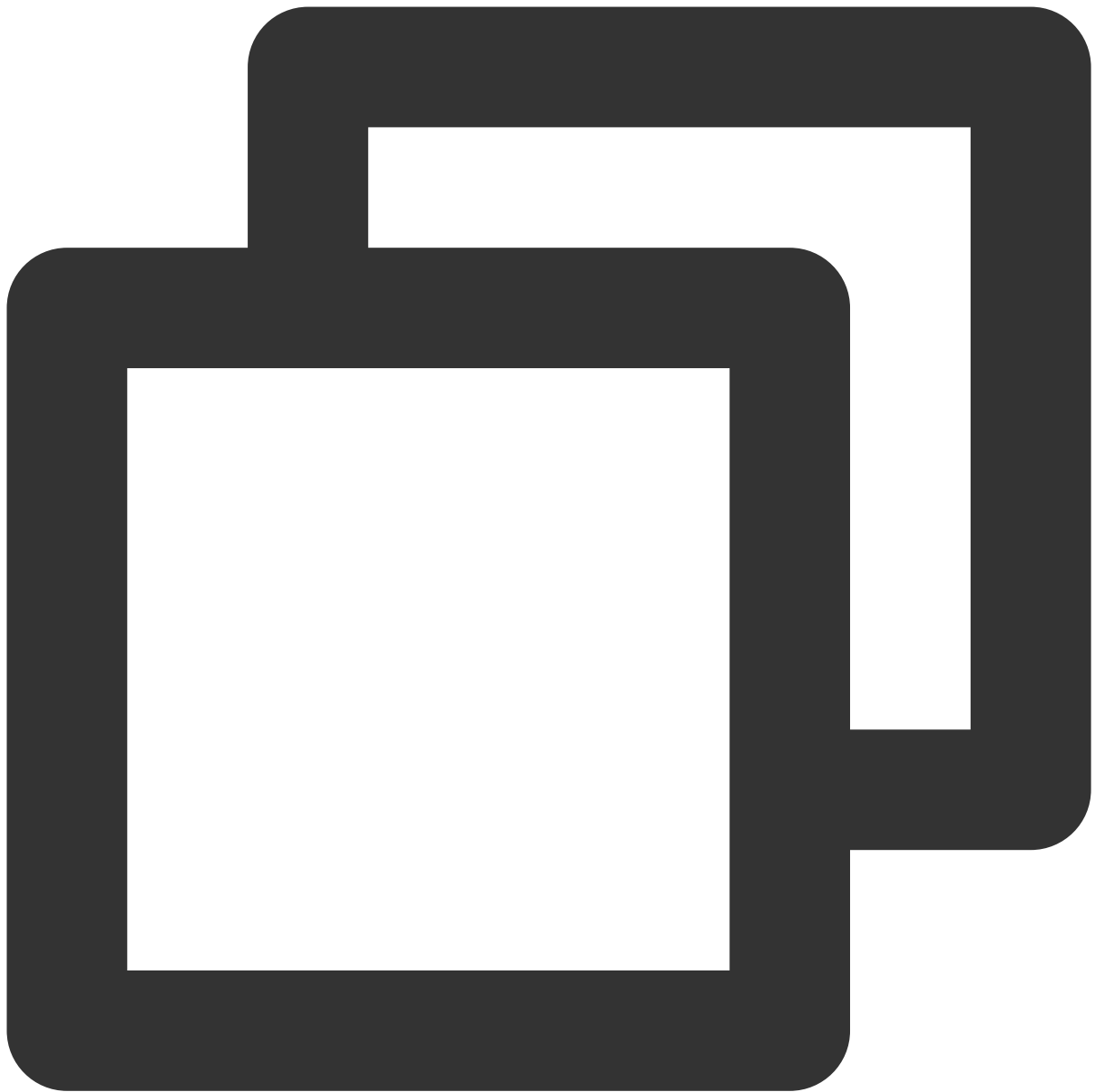
```
TUICallKit.createInstance(context).groupCall("12345678", Arrays.asList("jane", "mik
```





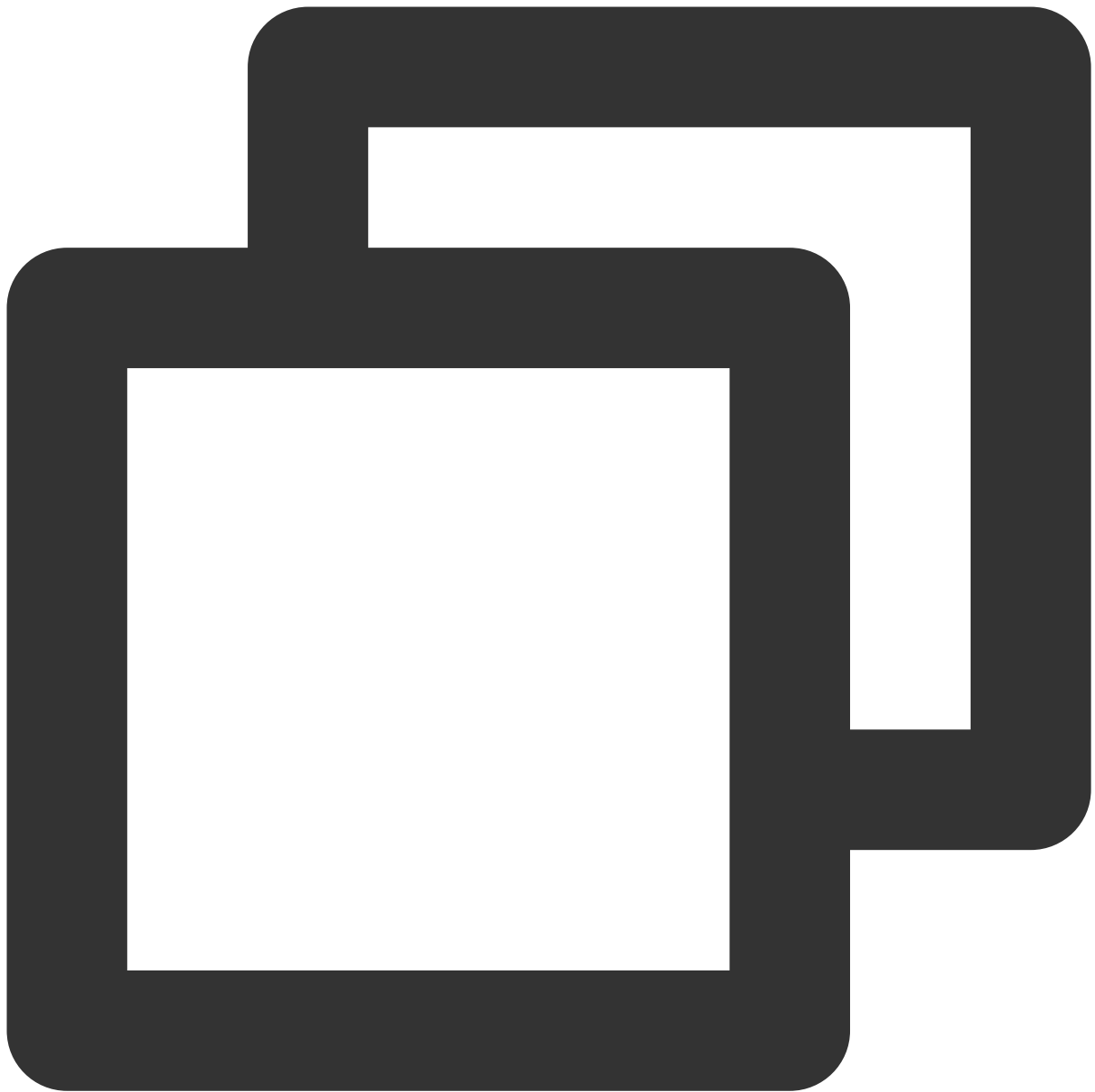
```
import TUICallKit

TUICallKit.createInstance().groupCall(groupId: "12345678",
                                         userIdList: ["denny", "mike", "tommy"],
                                         callMediaType: .video)
```



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

[[TUICallKit sharedInstance] groupCall:@"12345678"
                                userIdList:@[@"denny", @"mike", @"tommy"]
                                callMediaType:TUICallMediaTypeVideo];
```



```
TUICallKit.instance.groupCall('0001', ['denny', 'mike', 'tommy'], TUICallMediaType.
```

## Join a group call

Actively join an existing audio and video call in the group by calling the `joinInGroupCall` API.

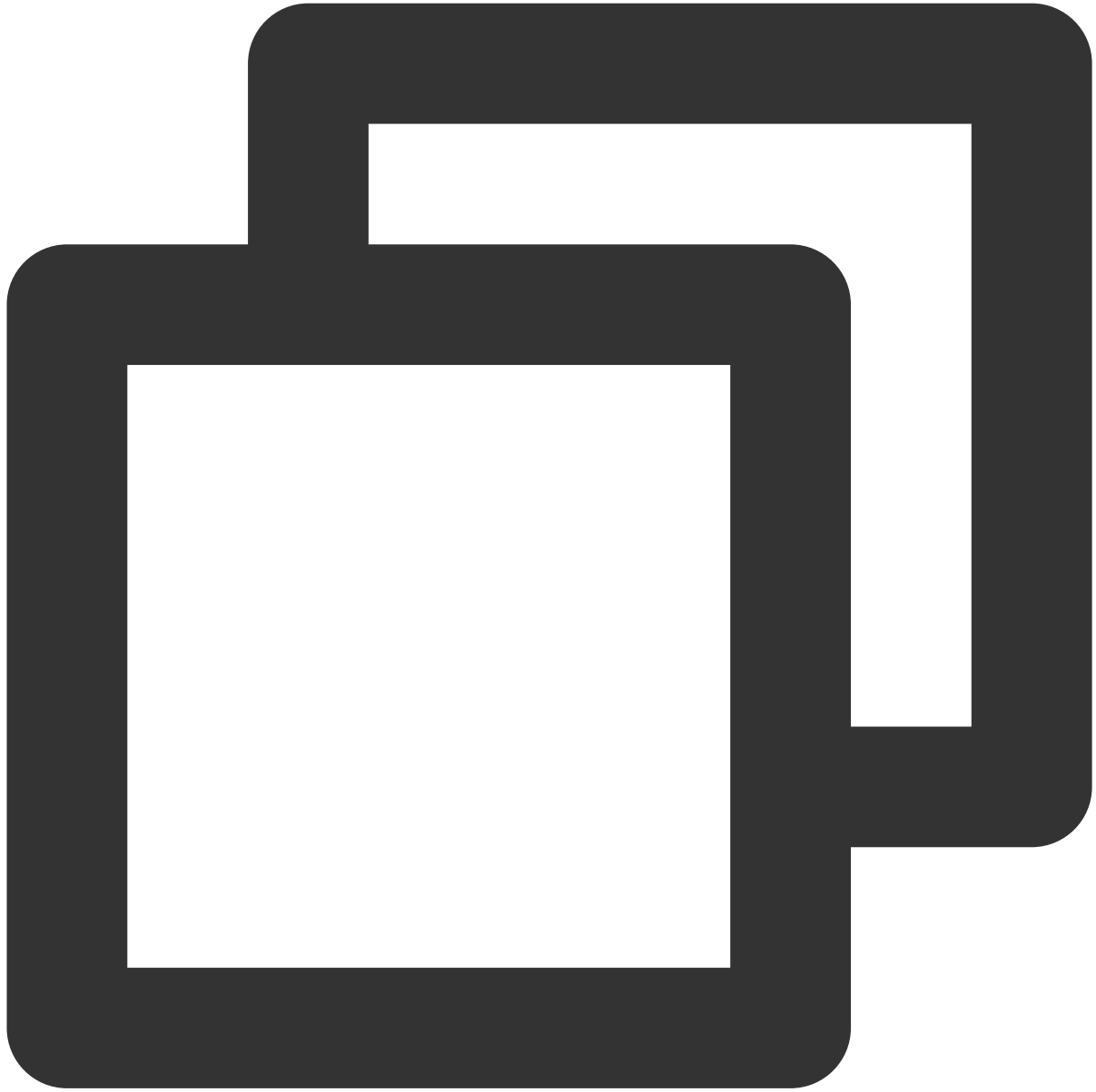
Android(Kotlin)

Android(Java)

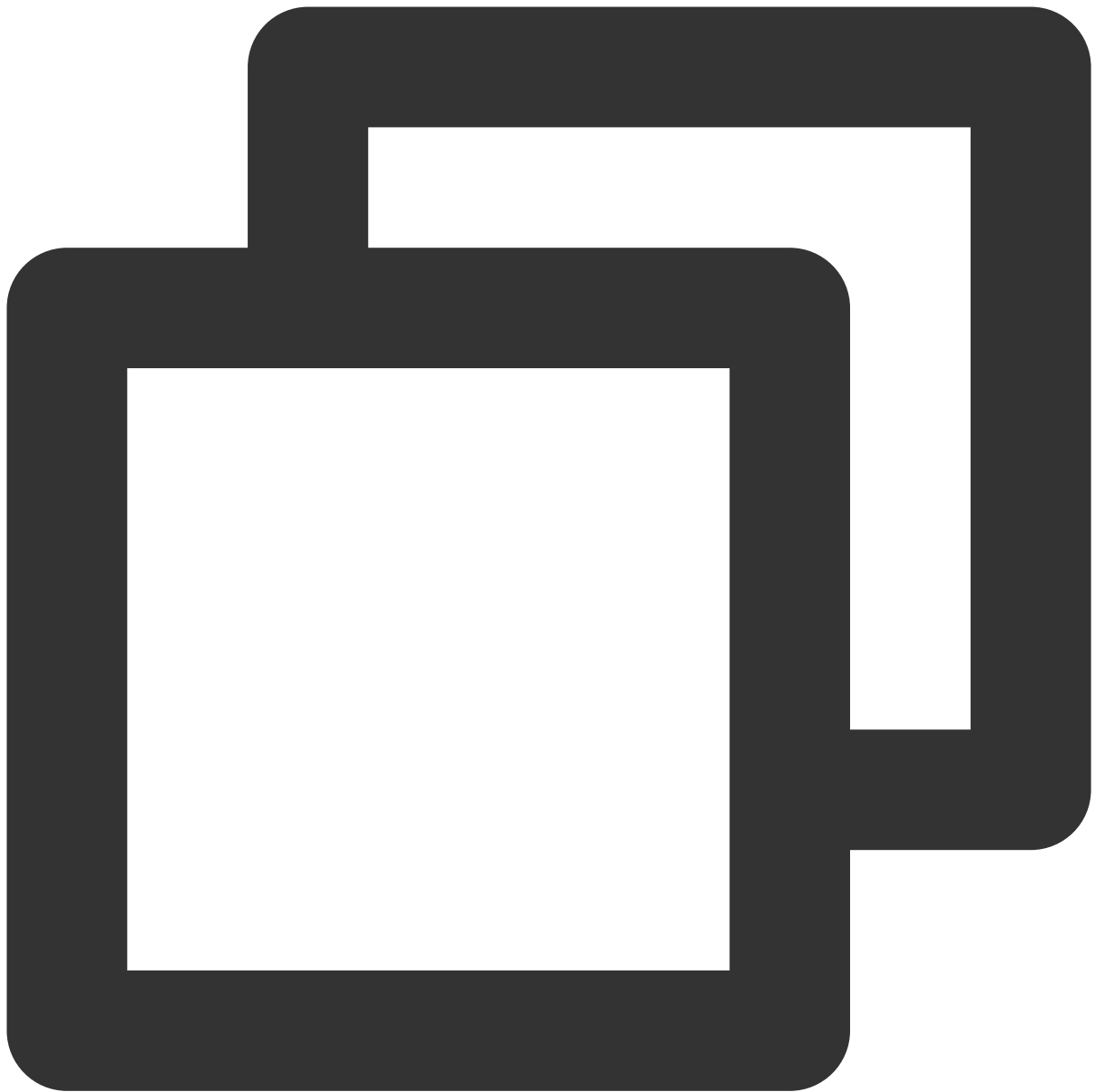
iOS(Swift)

iOS(Objective-C)

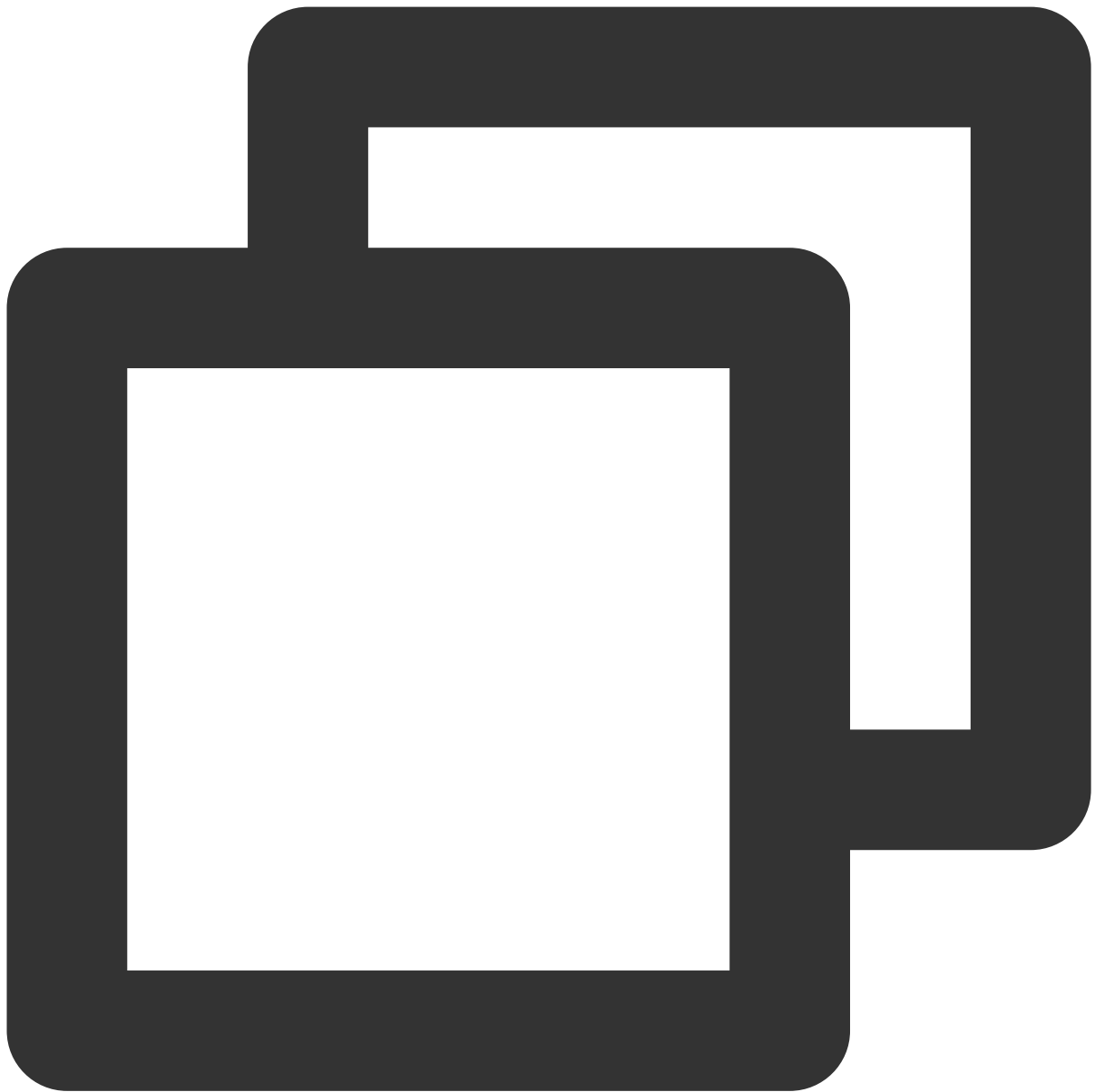
Flutter(Dart)



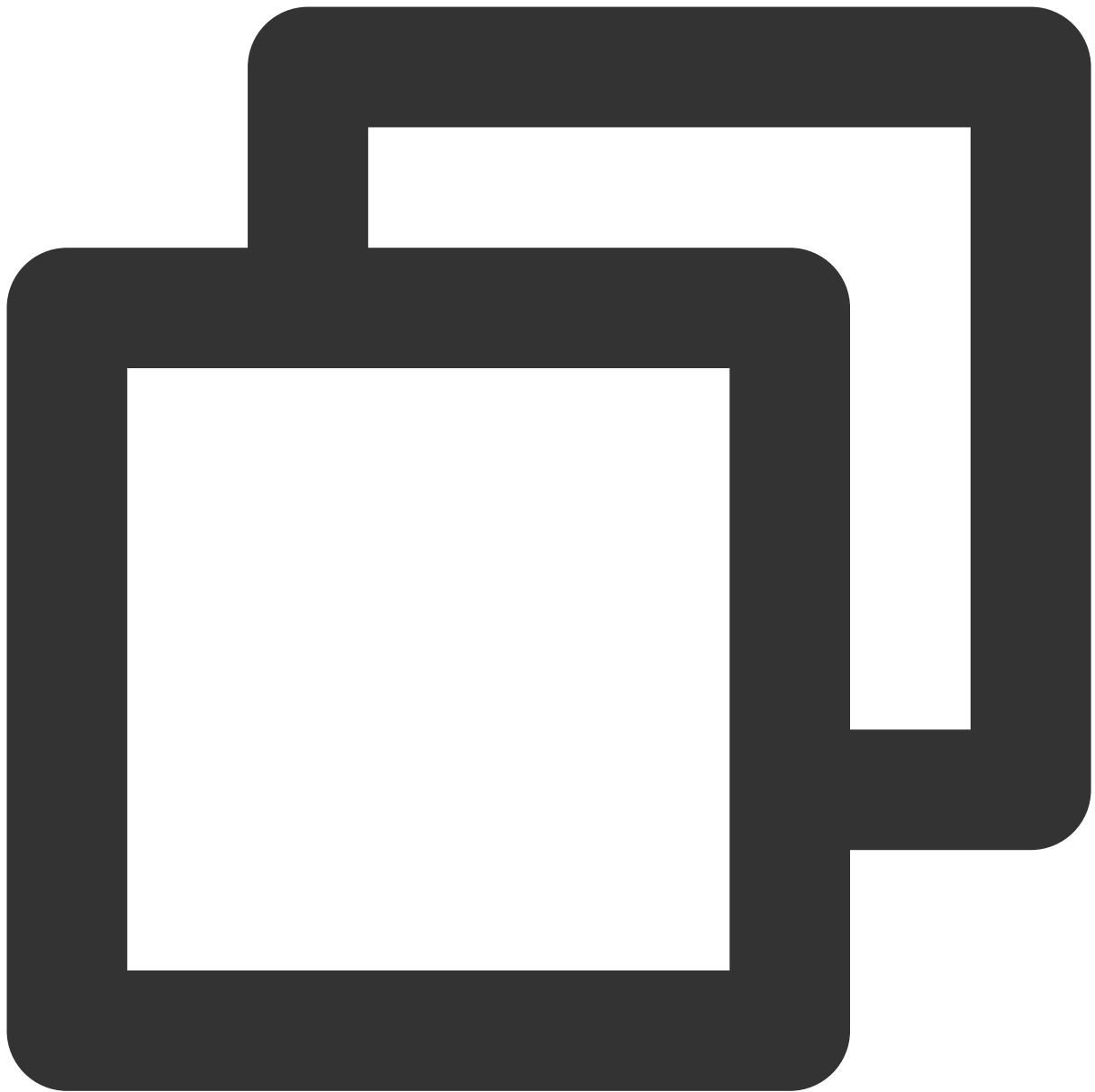
```
RoomId roomId = RoomId();  
roomId.intRoomId = 123321;  
TUICallKit.createInstance(context).joinInGroupCall(roomId, "12345678", TUICallDefin
```



```
RoomId roomId = new RoomId();  
roomId.intRoomId = 123321;  
TUICallKit.createInstance(context).joinInGroupCall(roomId , "12345678", TUICallDefi
```



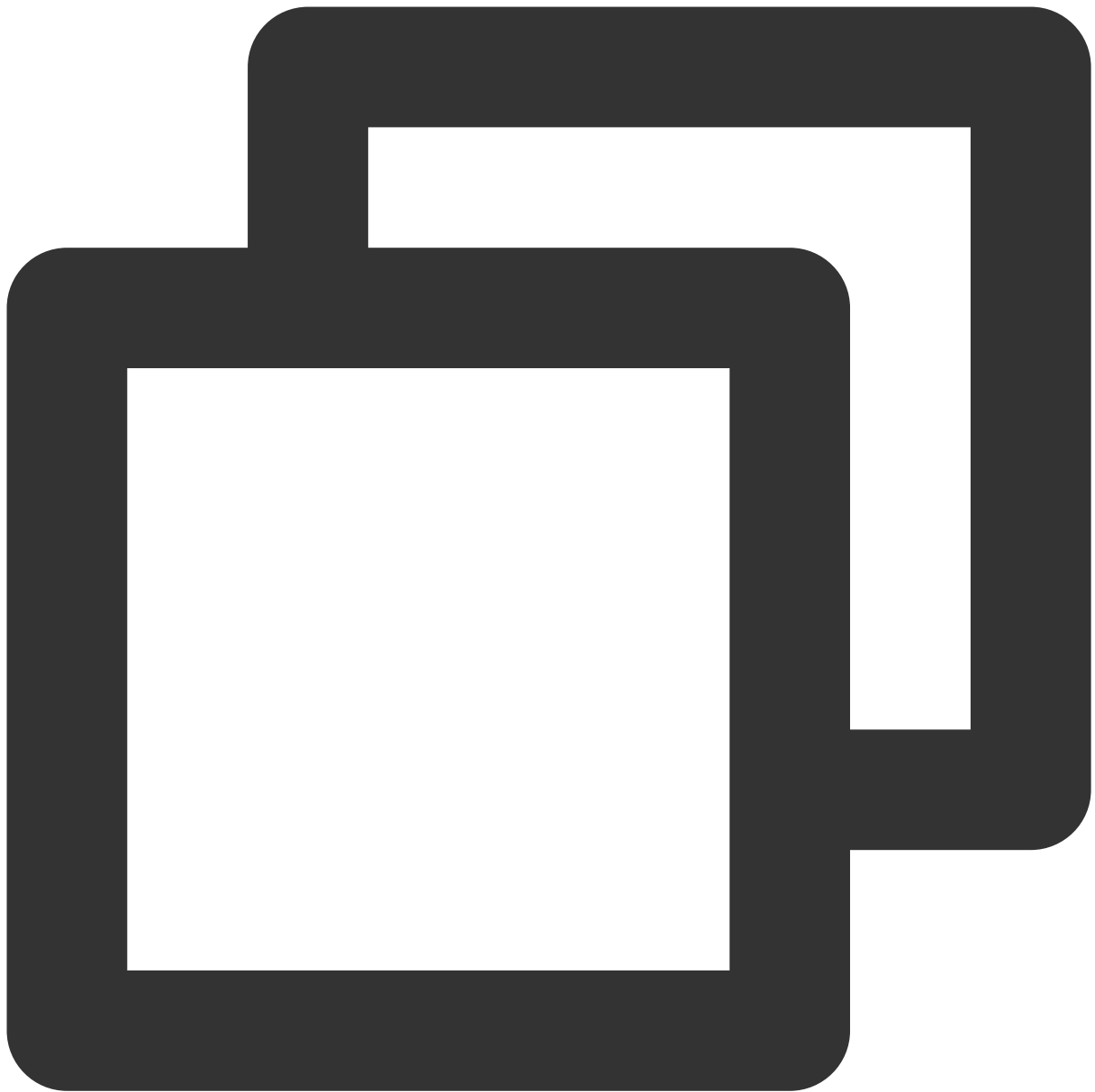
```
import TUICallKit
let roomId = TUIRoomId()
roomId.intRoomId = 123321
TUICallKit.createInstance().joinInGroupCall(roomId: roomId,
                                             groupId: "1234567",
                                             callMediaType: .video)
```



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

TUIRoomId *roomId = [[TUIRoomId alloc] init];
roomId.intRoomId = 123321;

[[TUICallKit sharedInstance] roomId: roomId
                           groupId:@"223344"
                           callMediaType:TUICallMediaTypeVideo];
```



```
TUIRoomId roomId = TUIRoomId()  
roomId.intRoomId = 123321;  
TUICallKit.instance.joinInGroupCall(roomId, '1234567', TUICallMediaType.video);
```



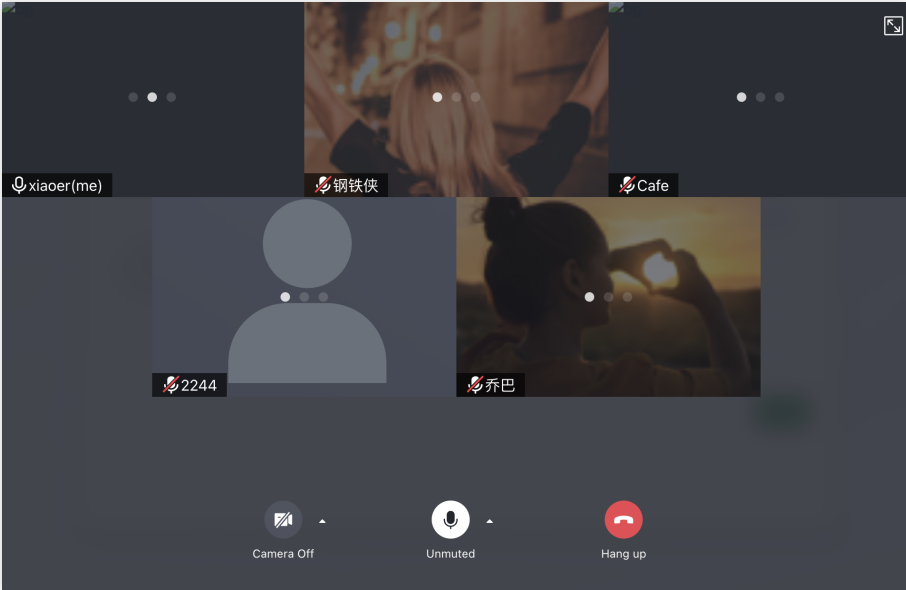
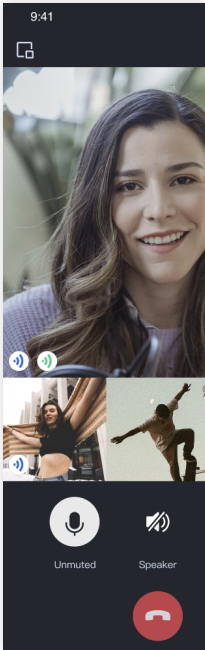
# Web&H5

Last updated : 2024-04-03 17:23:11

This article introduces the use of the group call feature, such as initiating a group call and joining a group call.

## Expected outcome

TUICallKit supports group calls. The expected outcome is shown in the figure below.

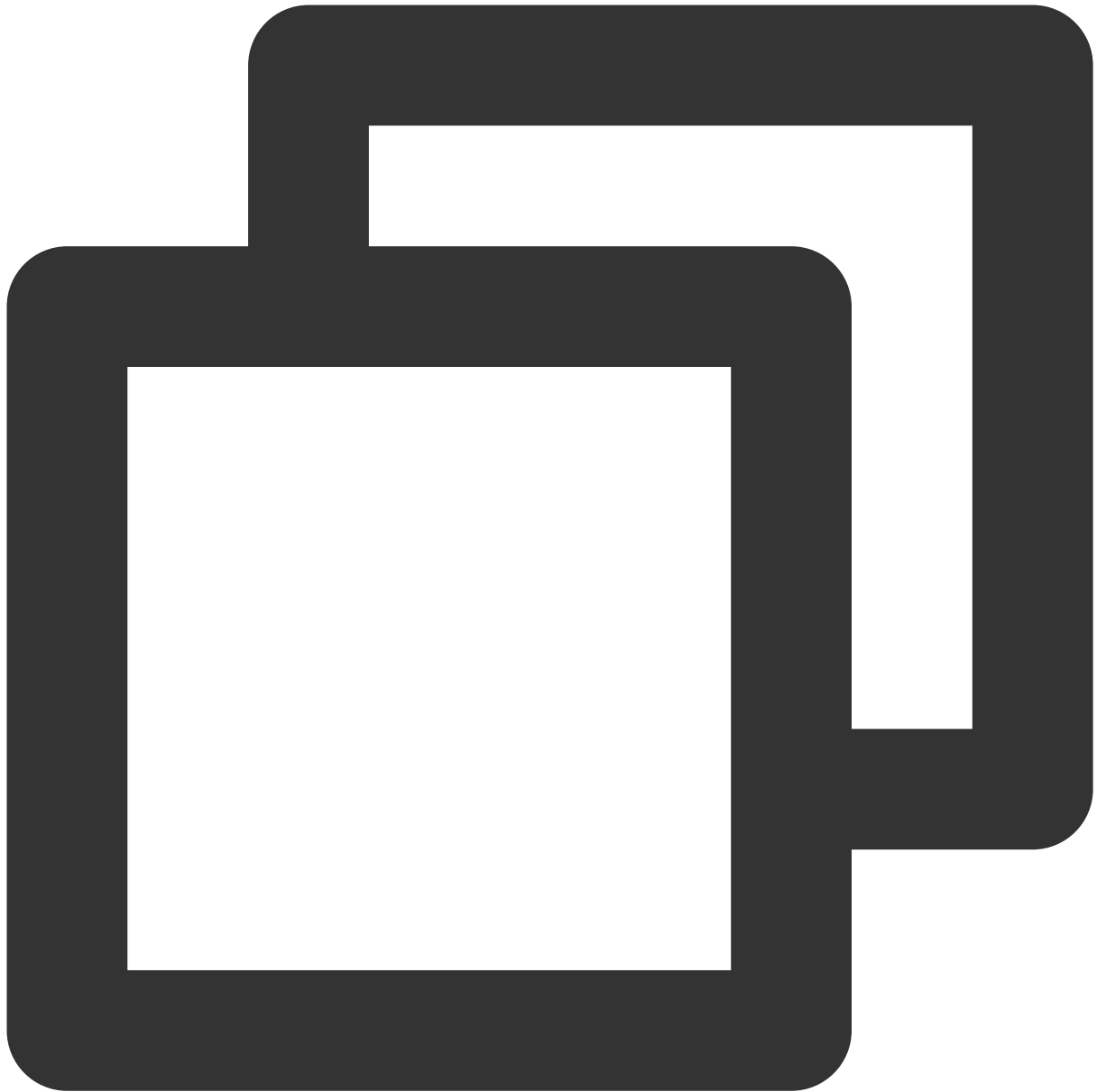
| Web   | Mobile Client  |
|---|--|
|  |  |

## Create groupID

Before using the group call feature, you need to create a group first and initiate a group call in an existing group.

Method one: Create a group by calling the [IM API](#), see [IM Group Management](#) for details.

Method two: Manually create a group through the console, see [Console group management](#) for details.



```
import TIM from "@tencentcloud/chat"; // npm i @tencentcloud/chat

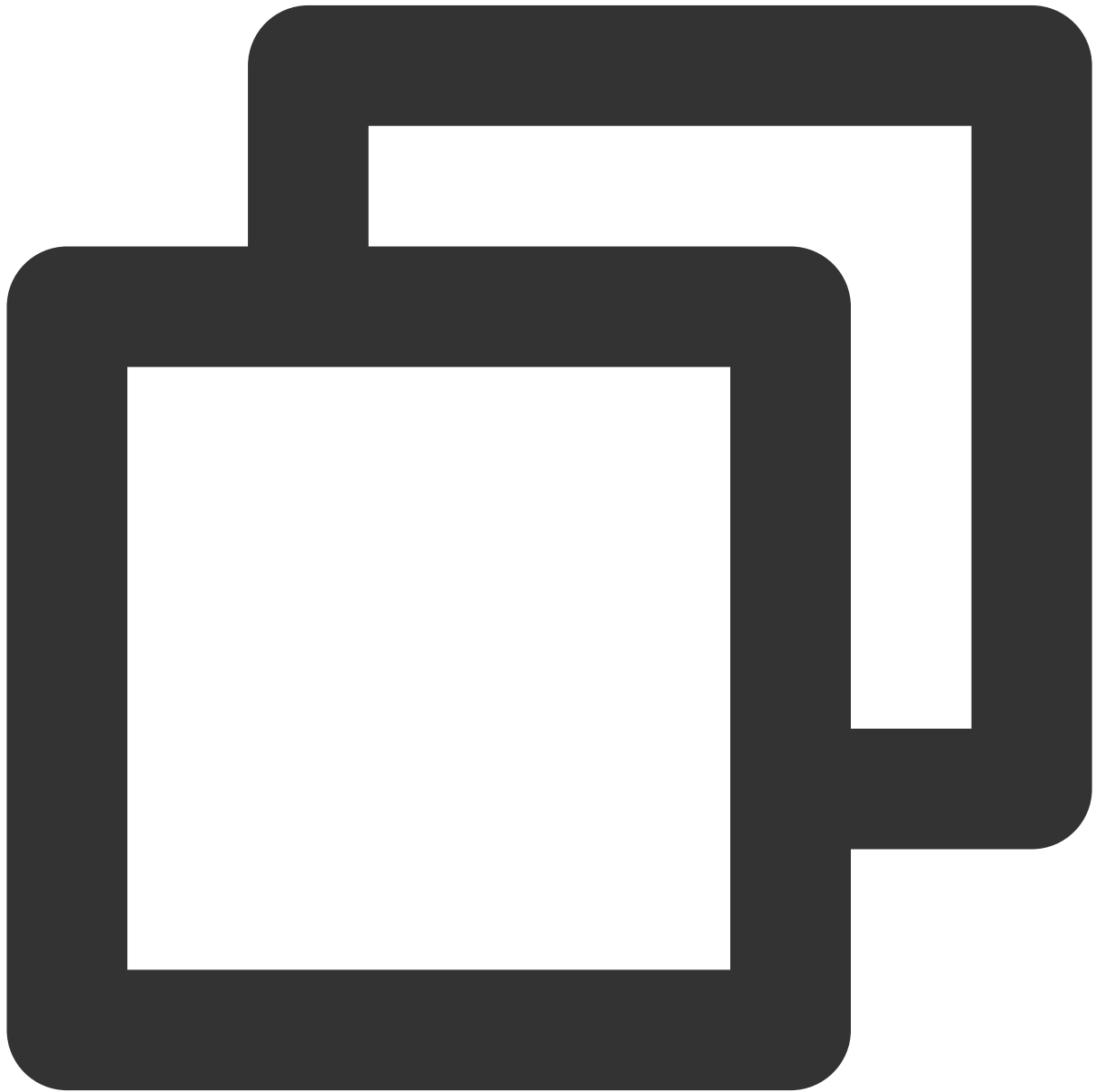
const userIDList: string[] = ['user1', 'user2'];
async function createGroupID() {
  const tim = TIM.create({ SDKAppID });
  const memberList: any[] = userIDList.map(userID => ({ userID: userID }));
  const res = await tim.createGroup({
    type: TIM.TYPES.GRP_PUBLIC, // Must be a public group
    name: 'WebSDK',
    memberList
  });
};
```

```
return res.data.group.groupID;  
}
```

## Group call

### Initiate a group call

Initiate a group call by calling the `groupCall` API.



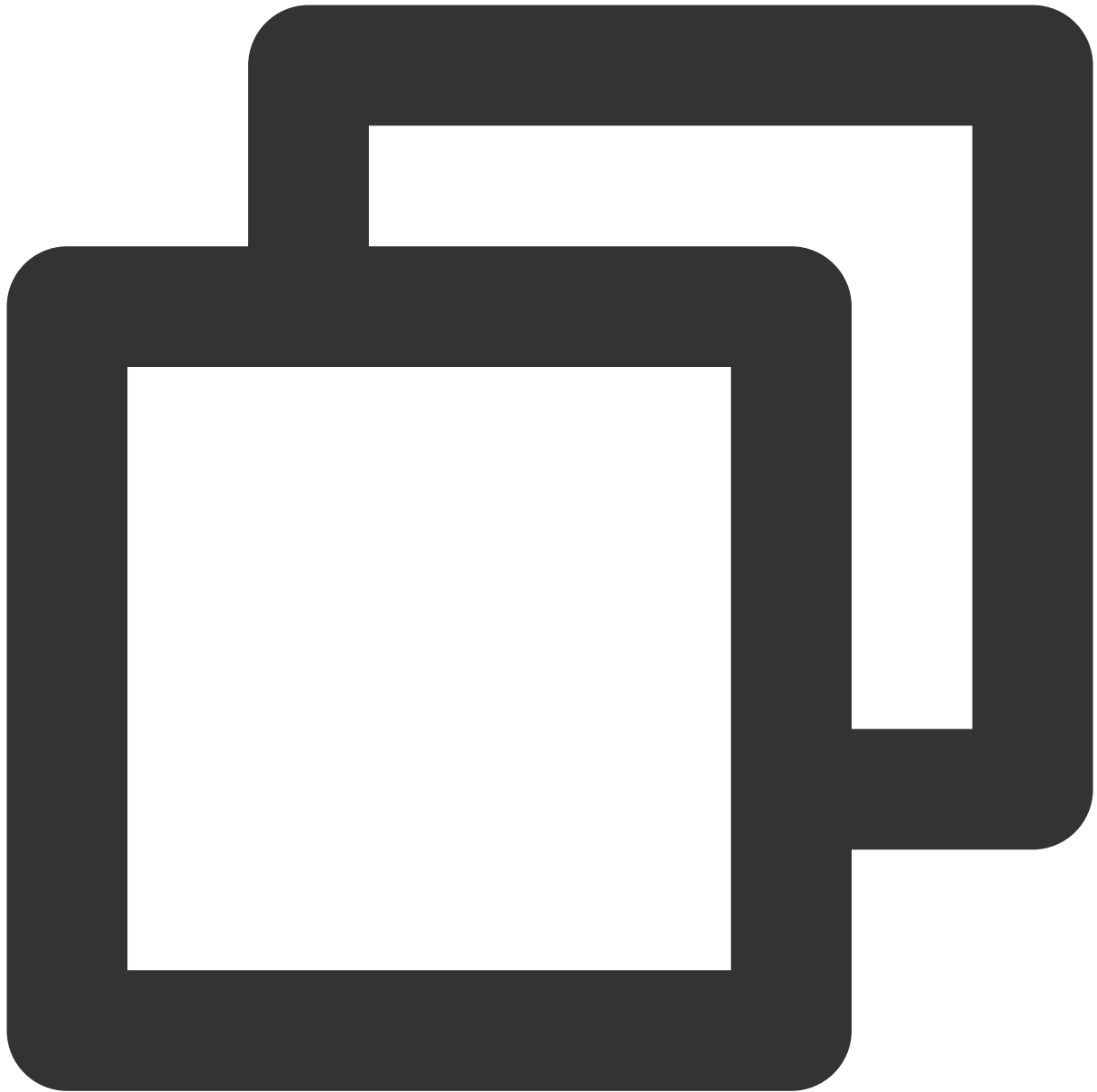
```
import { TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit-react";
// Replace it with the call-uikit npm package you are currently using

try {
  const params = {
    userIDList: ['user1', 'user2'],
    groupID: 'xxx',
    type: TUICallType.VIDEO_CALL,
  }
  await TUICallKitServer.groupCall(params);
} catch (error: any) {
  alert(`[TUICallKit] groupCall failed. Reason:${error}`);
}
```

## Join a group call

Join an existing audio and video call in the group by calling the `joinInGroupCall` API.

**Note:** Vue ≥ v3.1.2 is supported



```
import { TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit-react";
// Replace it with the call-uikit npm package you are currently using

try {
  const params = {
    type: TUICallType.VIDEO_CALL,
    groupID: "xxx",
    roomID: 0,
  };
  await TUICallKitServer.joinInGroupCall(params);
} catch (error: any) {
```

```
alert(`[TUICallKit] joinInGroupCall failed. Reason: ${error}`);  
}
```

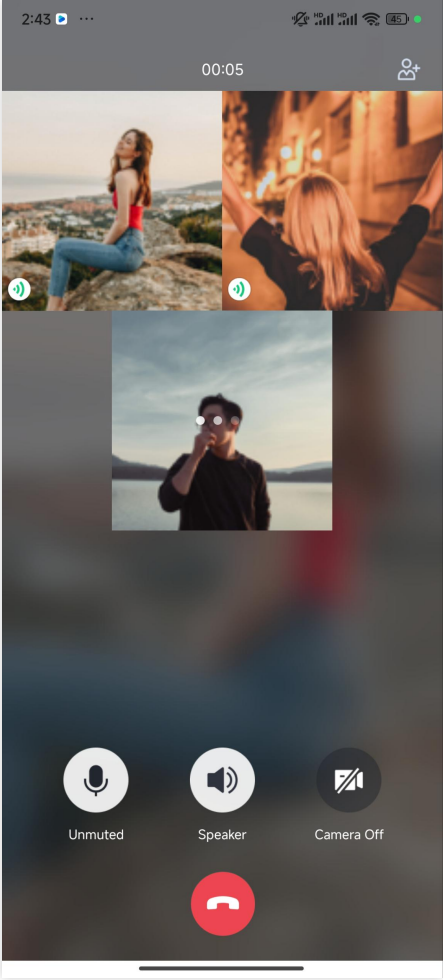
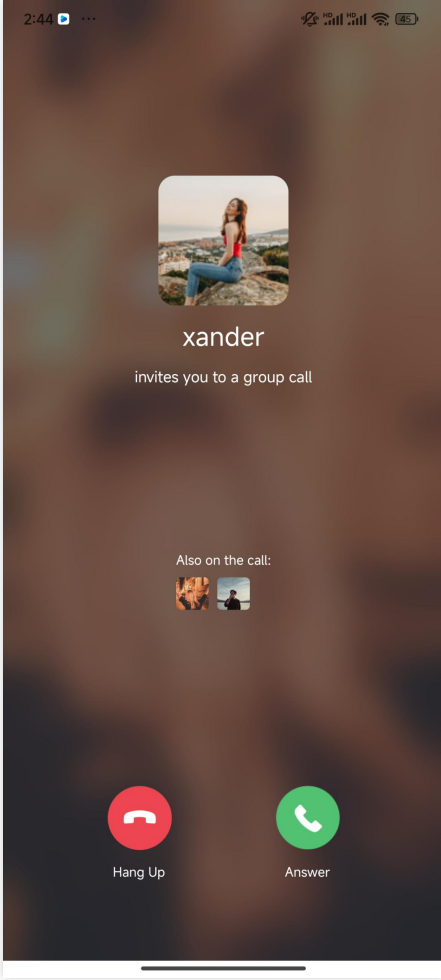

# uni-app (Anroid&iOS)

Last updated : 2024-04-03 17:23:11

This article introduces the use of the group call feature, such as initiating a group call and joining a group call.

## Expected outcome

TUICallKit supports group calls. The expected outcome is shown in the figure below.

| Initiate a group call  | Received Group Call Invitation  | Accept   |
|--|---|--|
|  |  |  |

## Create groupID

Before using the group call feature, you need to create a group first and initiate a group call in an existing group.

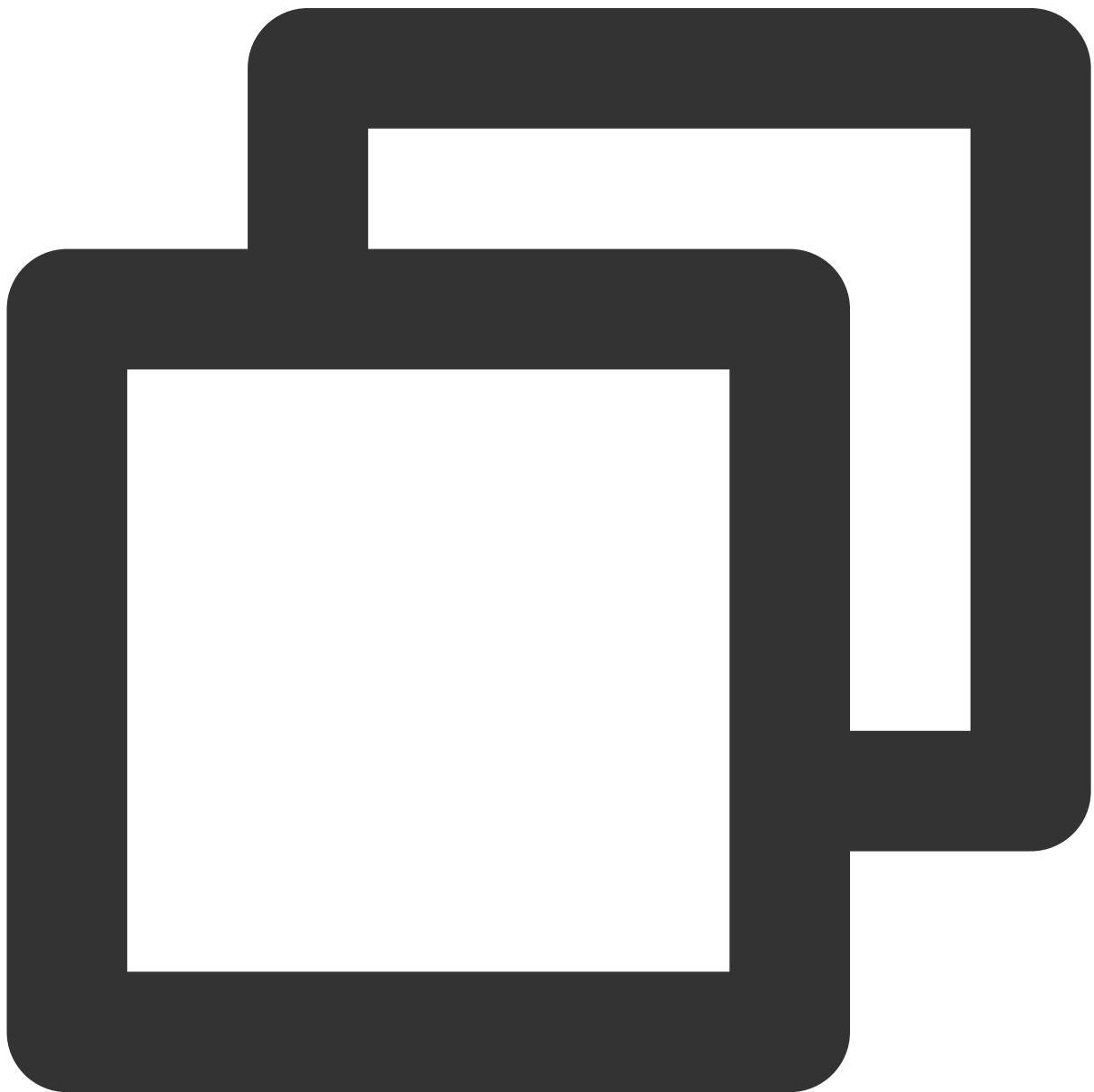
Method one: Create a group by calling the [IM API](#), see [IM Group Management](#) for details.

Method two: Manually create a group through the console, see [Console group management](#) for details.

## Group call

### Initiate a group call

Launch a group call using the groupCall API.

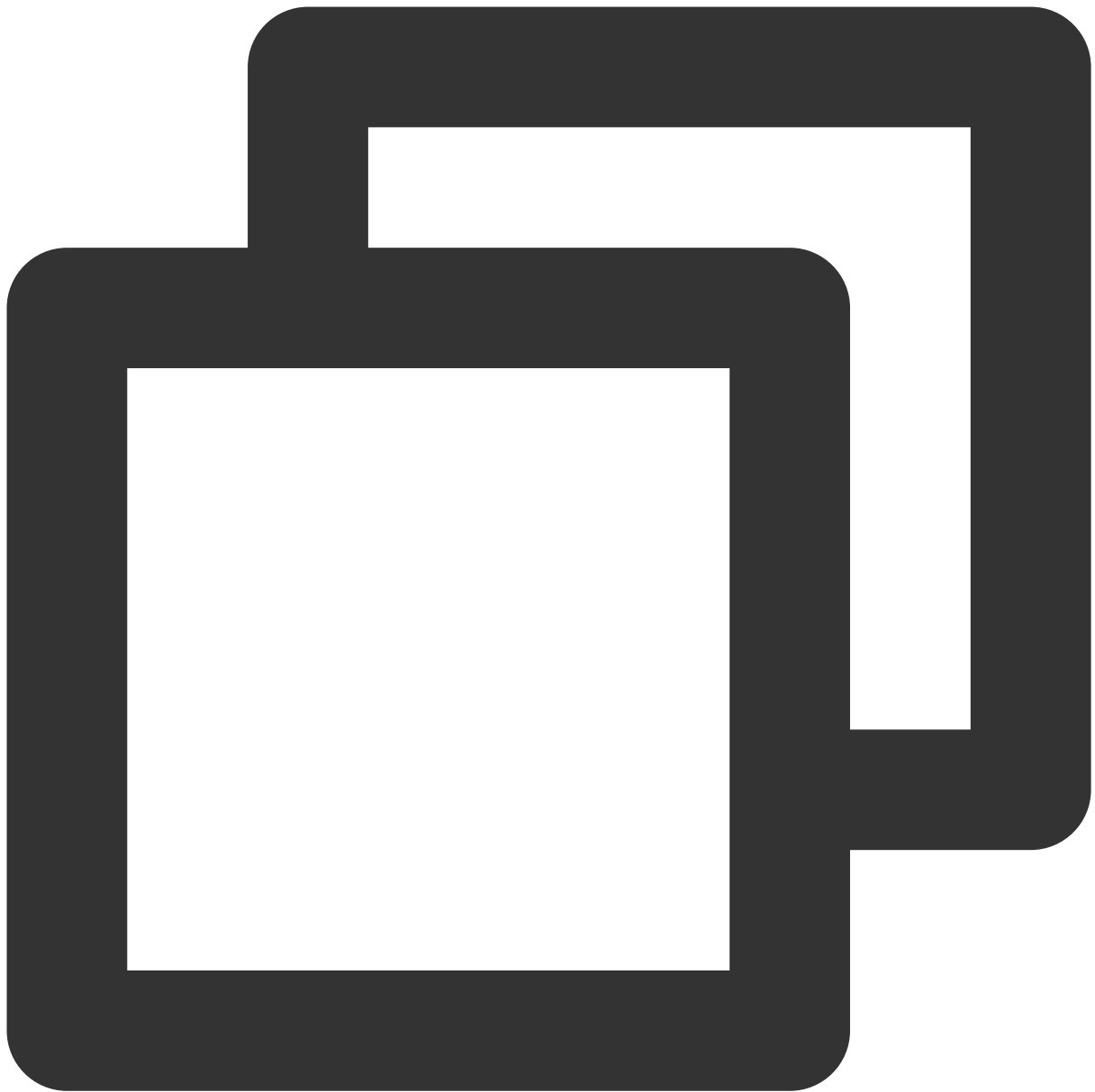




```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  groupID: 'myGroup',
  userIDList: ['mike', 'tom'],
  callMediaType: 1, // voice call(callMediaType = 1),video call(callMediaType = 2)
};
TUICallKit.groupCall(options, (res) => {
  if (res.code === 0) {
    console.log('groupCall success');
  } else {
    console.log(`groupCall failed, error message = ${res.msg}`);
  }
});
```

## Join a group call

Actively join an existing audio and video call in the group by calling the joinInGroupCall API.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  roomId: 9898,
  groupId: 'myGroup',
  callMediaType: 1, // voice call(callMediaType = 1),video call(callMediaType = 2)
};
TUICallKit.joinInGroupCall(options, (res) => {
  if (res.code === 0) {
    console.log('joinInGroupCall success');
  } else {
    console.log(`joinInGroupCall failed, error message = ${res.msg}`);
  }
});
```

```
}  
});
```

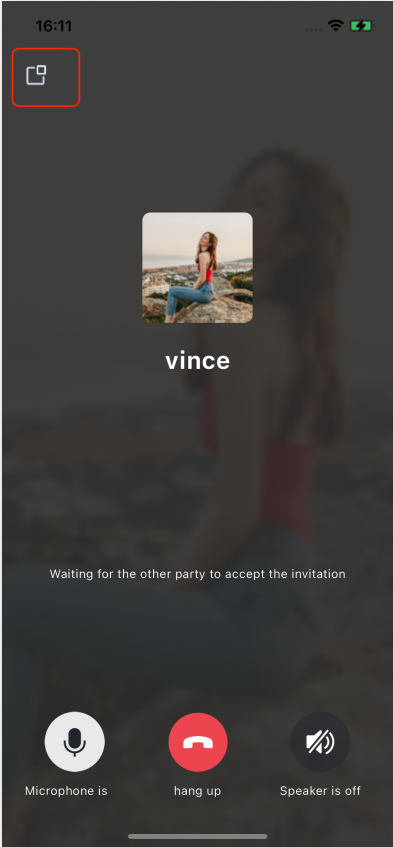
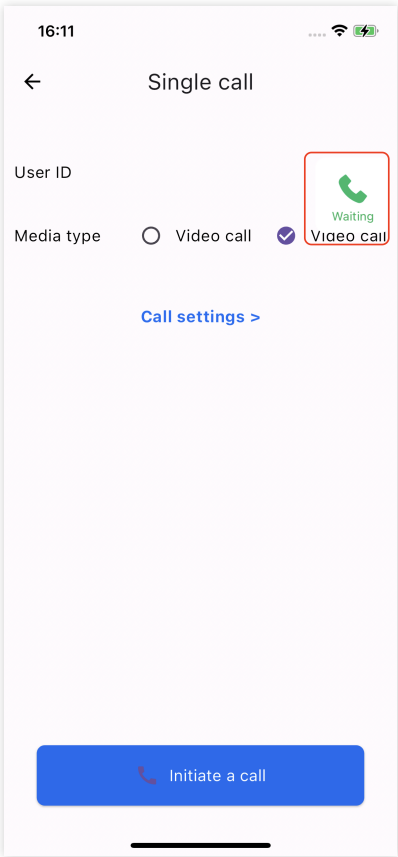

# Floating Window

## Android&iOS&Flutter

Last updated : 2024-04-03 17:23:11

This article explains how to use the Floating Window feature.

### Expected outcome

| Activate floating button   | Voice call floating window  | Video call floa  |
|--|---|--|
|  |  |  |

### Floating Window feature

TUICallKit allows users to minimize the call interface into a floating window using the floating window button at the top left corner of the call interface during a call.

If your business needs to enable this feature, you can use the `enableFloatWindow` method to activate this feature during the initialization of the `TUICallKit` component:

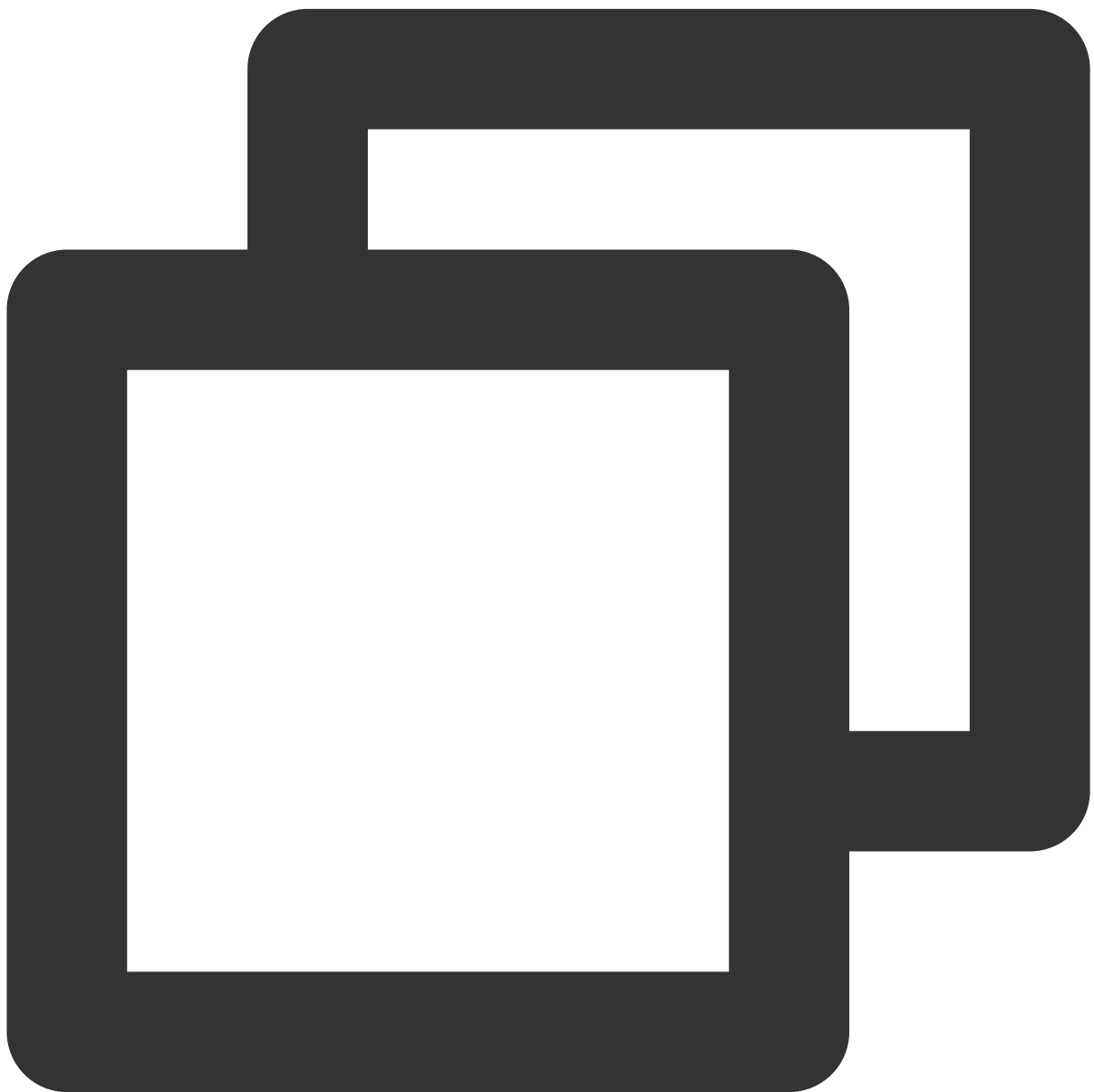
Android(Kotlin)

Android(Java)

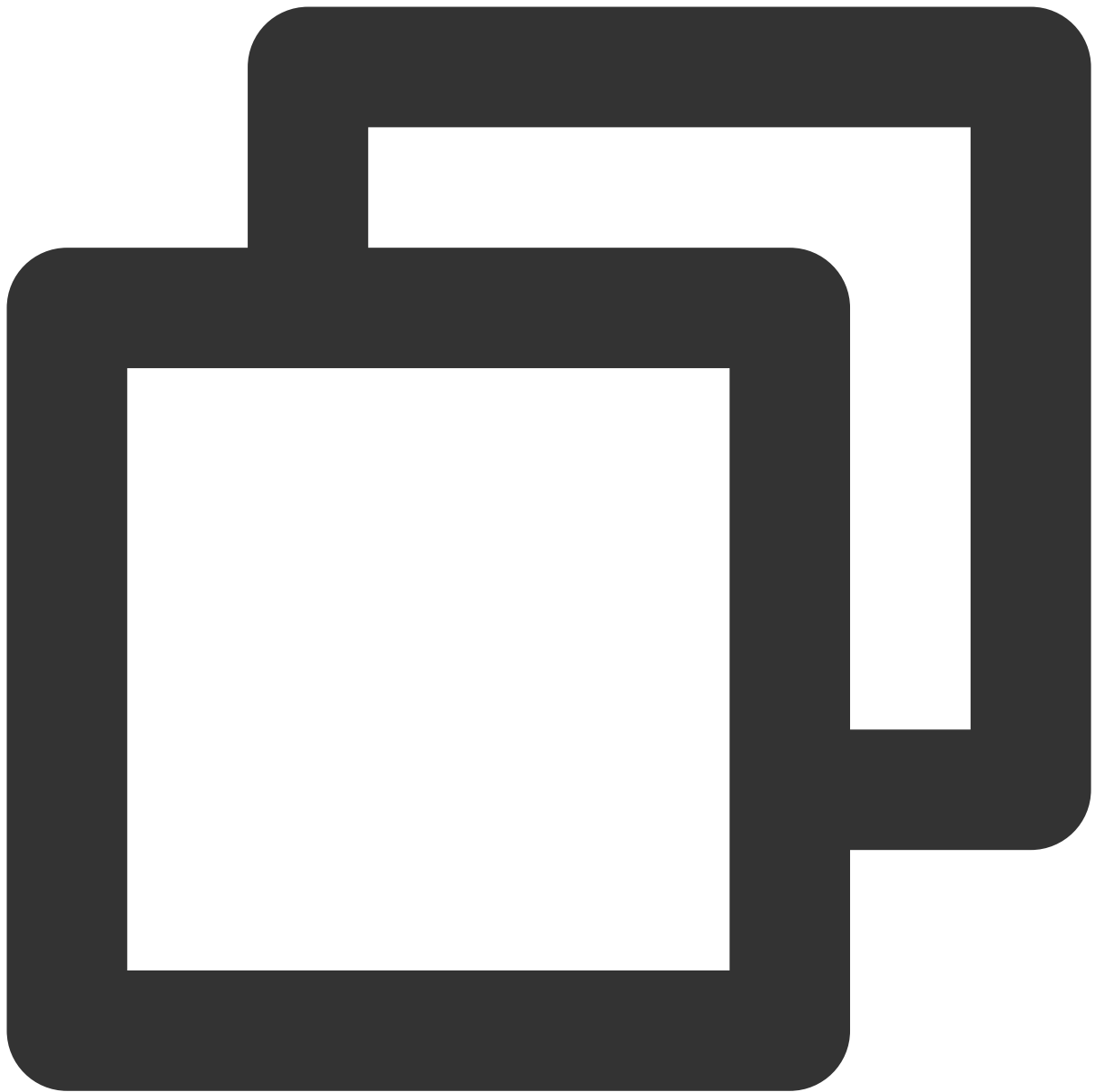
iOS(Swift)

iOS(Objective-C)

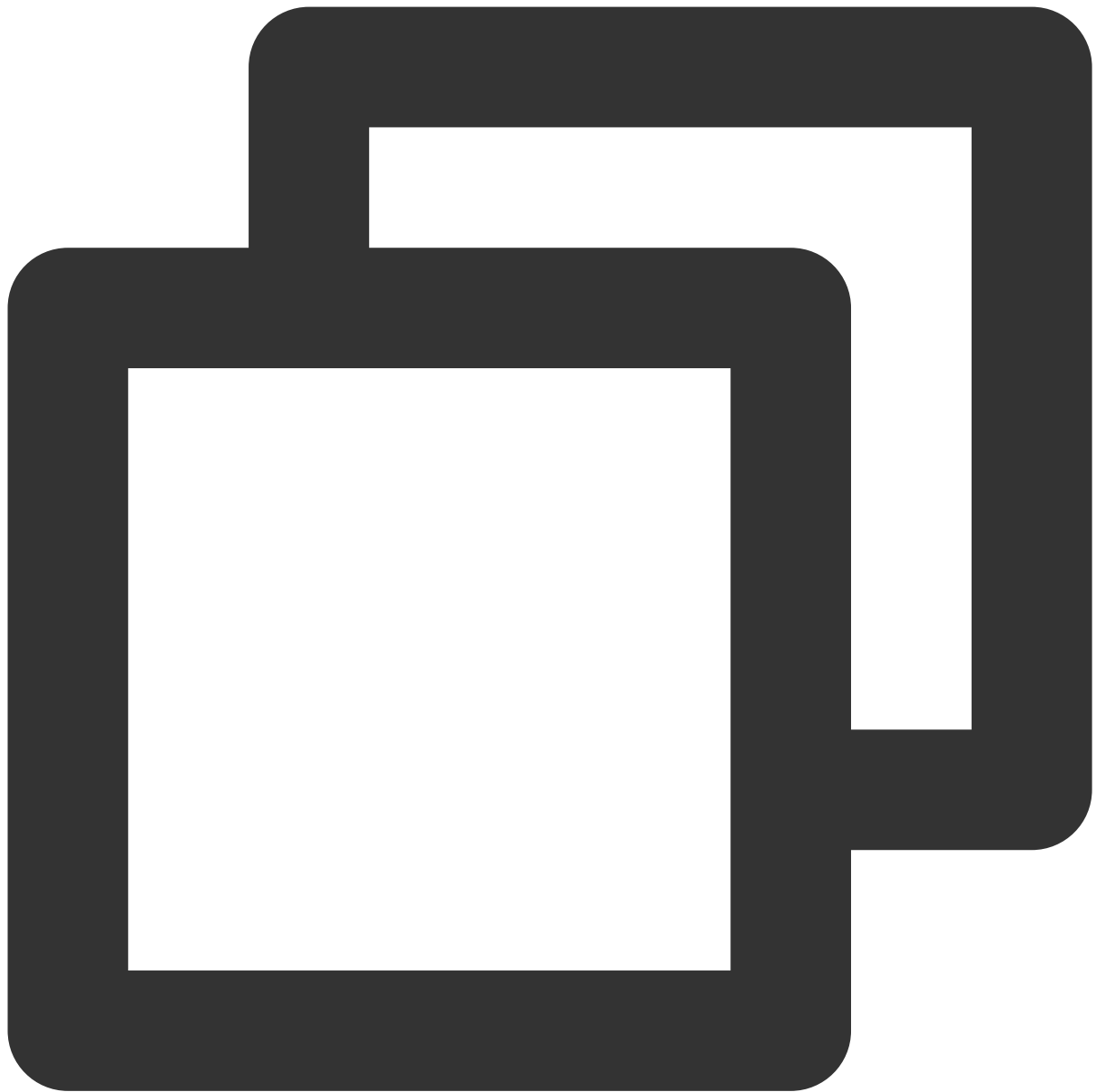
Flutter(Dart)



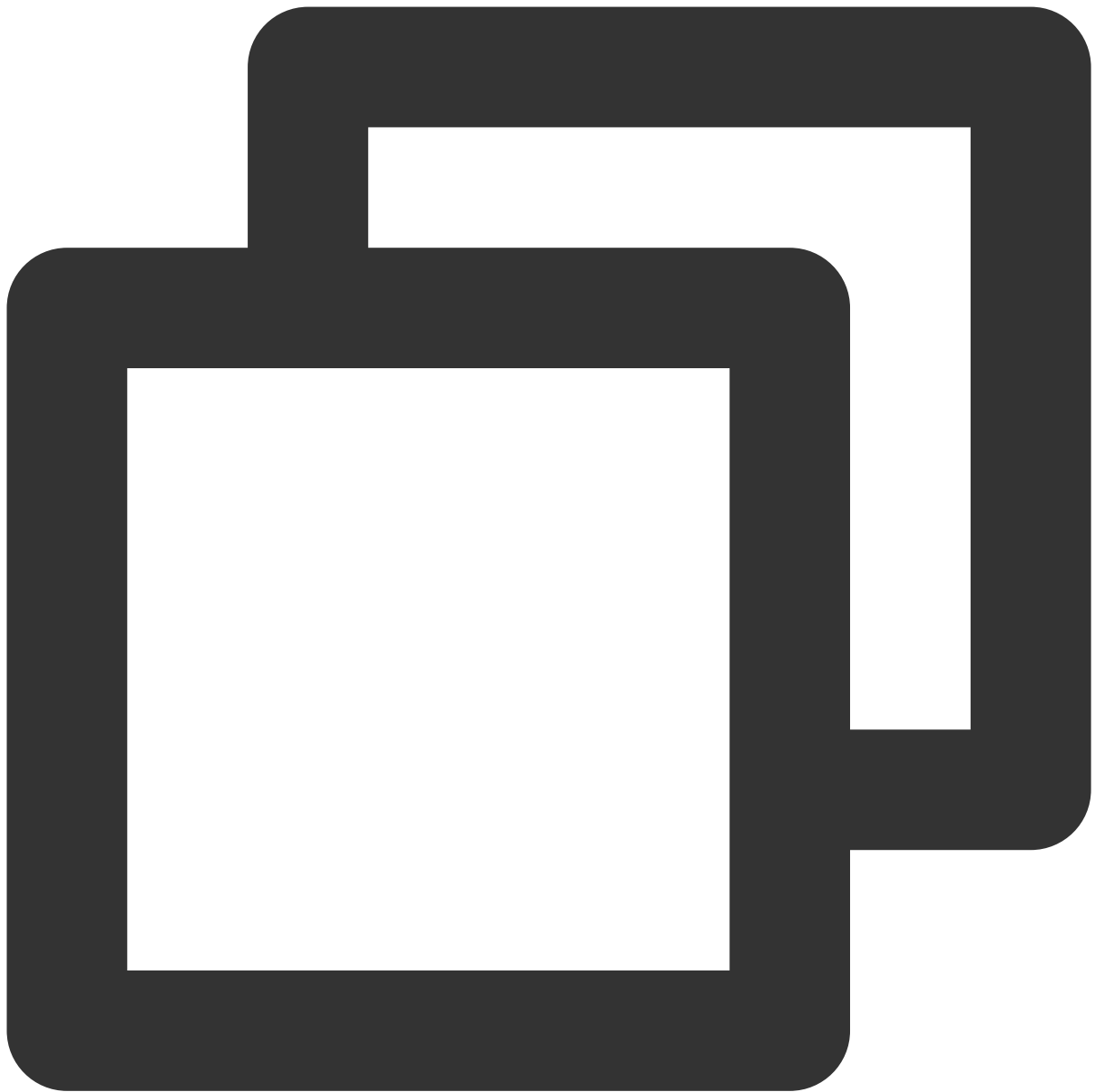
```
TUICallKit.createInstance(context).enableFloatWindow(true)
```



```
TUICallKit.createInstance(context).enableFloatWindow(true);
```

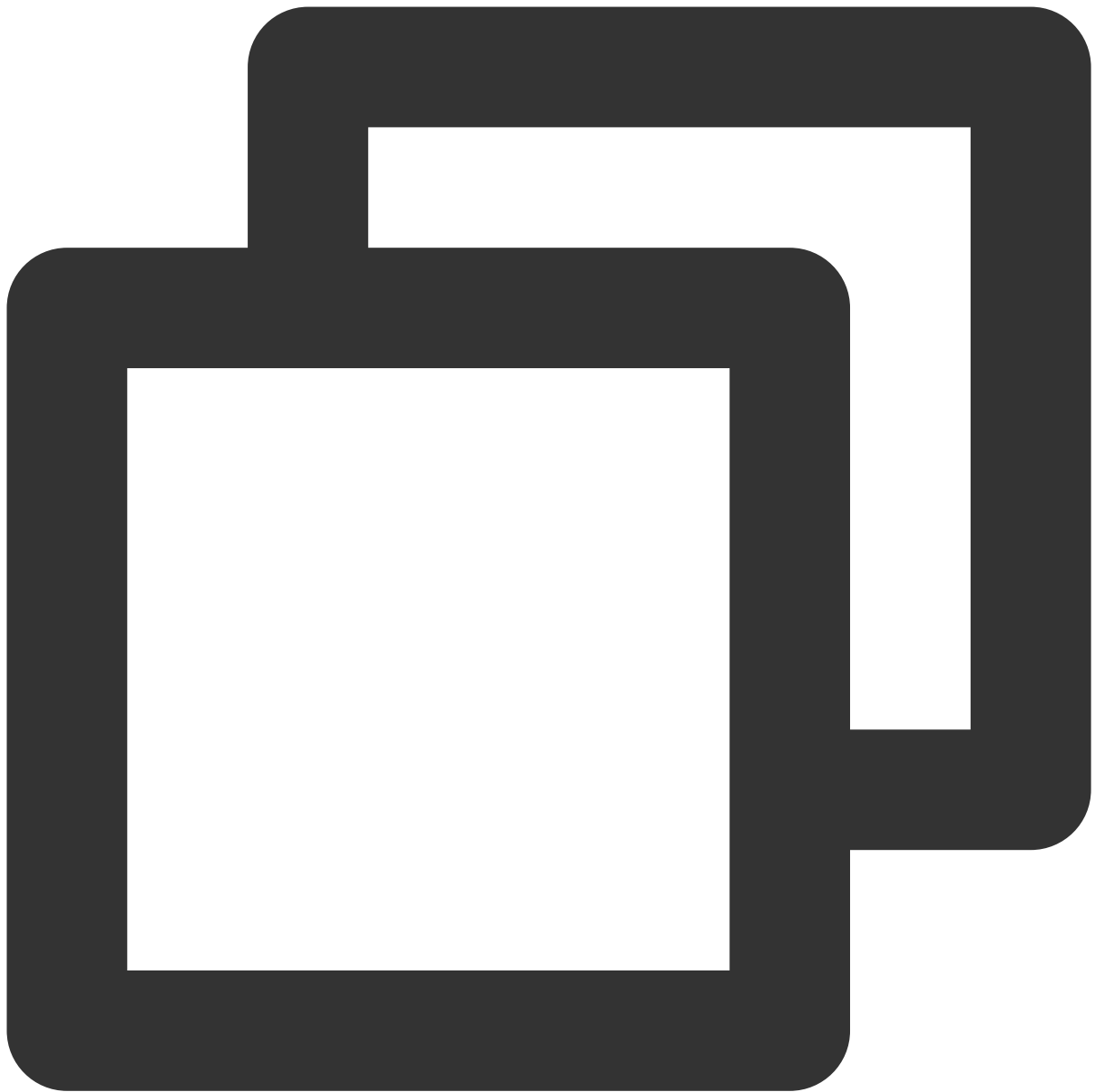


```
TUICallKit.createInstance().enableFloatWindow(true)
```



```
[[TUICallKit sharedInstance] enableFloatWindow:YES];
```





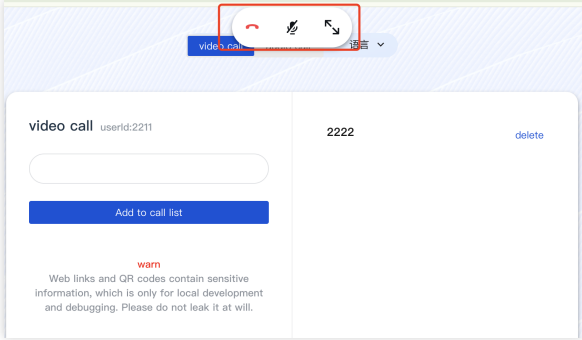
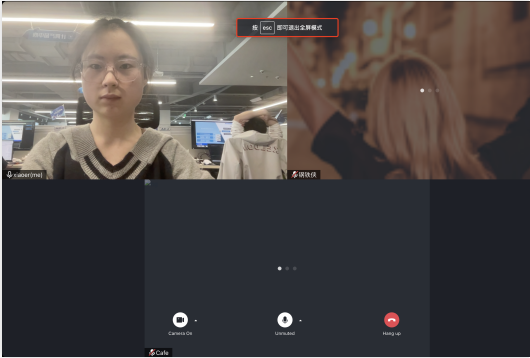
```
TUICallKit.instance.enableFloatWindow(true);
```

# Web&H5

Last updated : 2024-04-03 17:23:11

This article will introduce how to use the Floating Window feature.

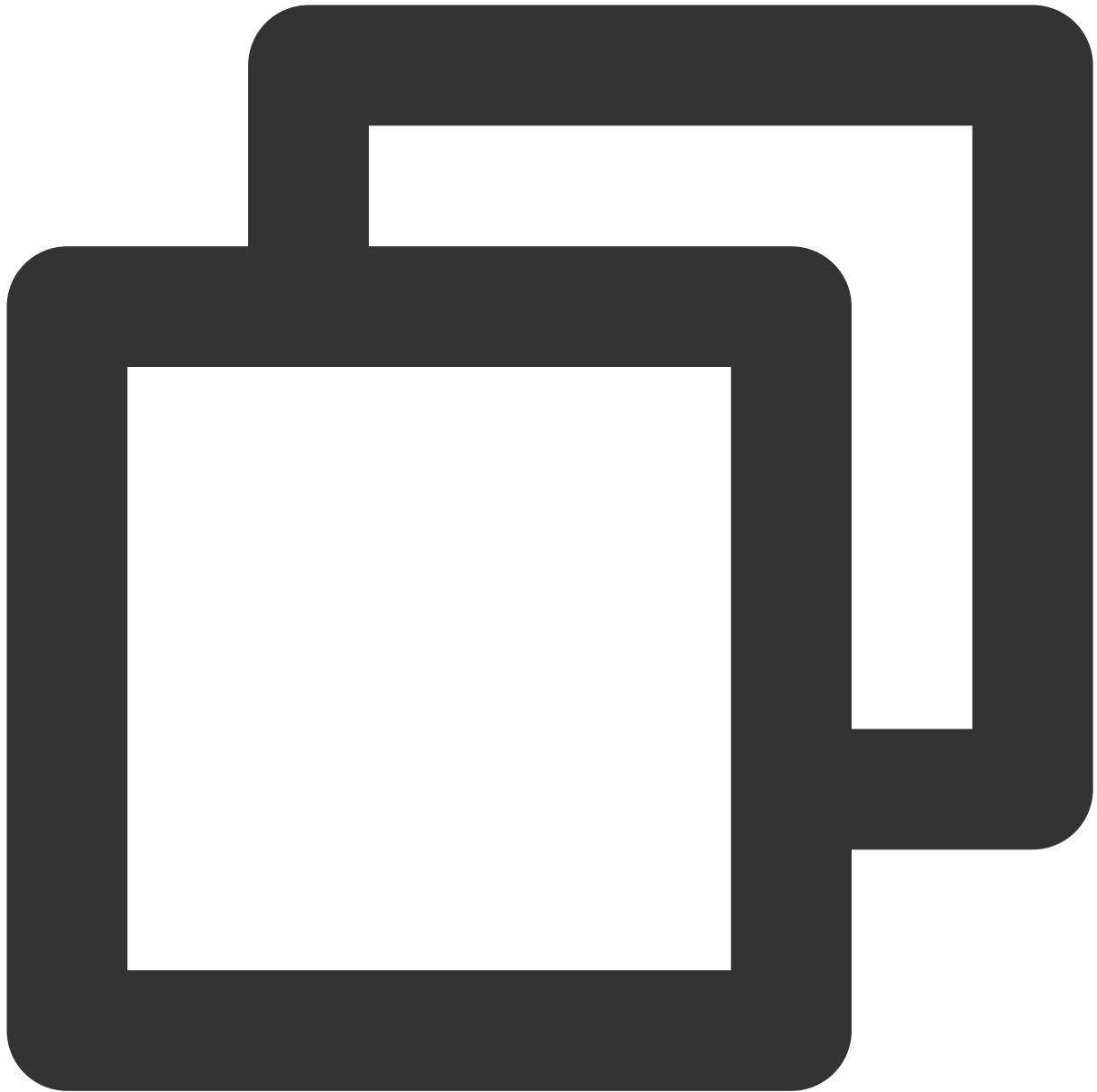
## Expected outcome

| Web Floating Window  | Web Full Screen   |
|--|---|
|  |  |

## Floating Window feature

Method 1: Use the `enableFloatWindow(enable: boolean)` API to enable/disable the Floating Window.

**Note:**Vue ≥ v3.1.0 is supported



```
try {  
  await TUICallKitServer.enableFloatWindow(enable: Boolean)  
} catch (error: any) {  
  alert(`[TUICallKit] enableFloatWindow failed. Reason: ${error}`);  
}
```

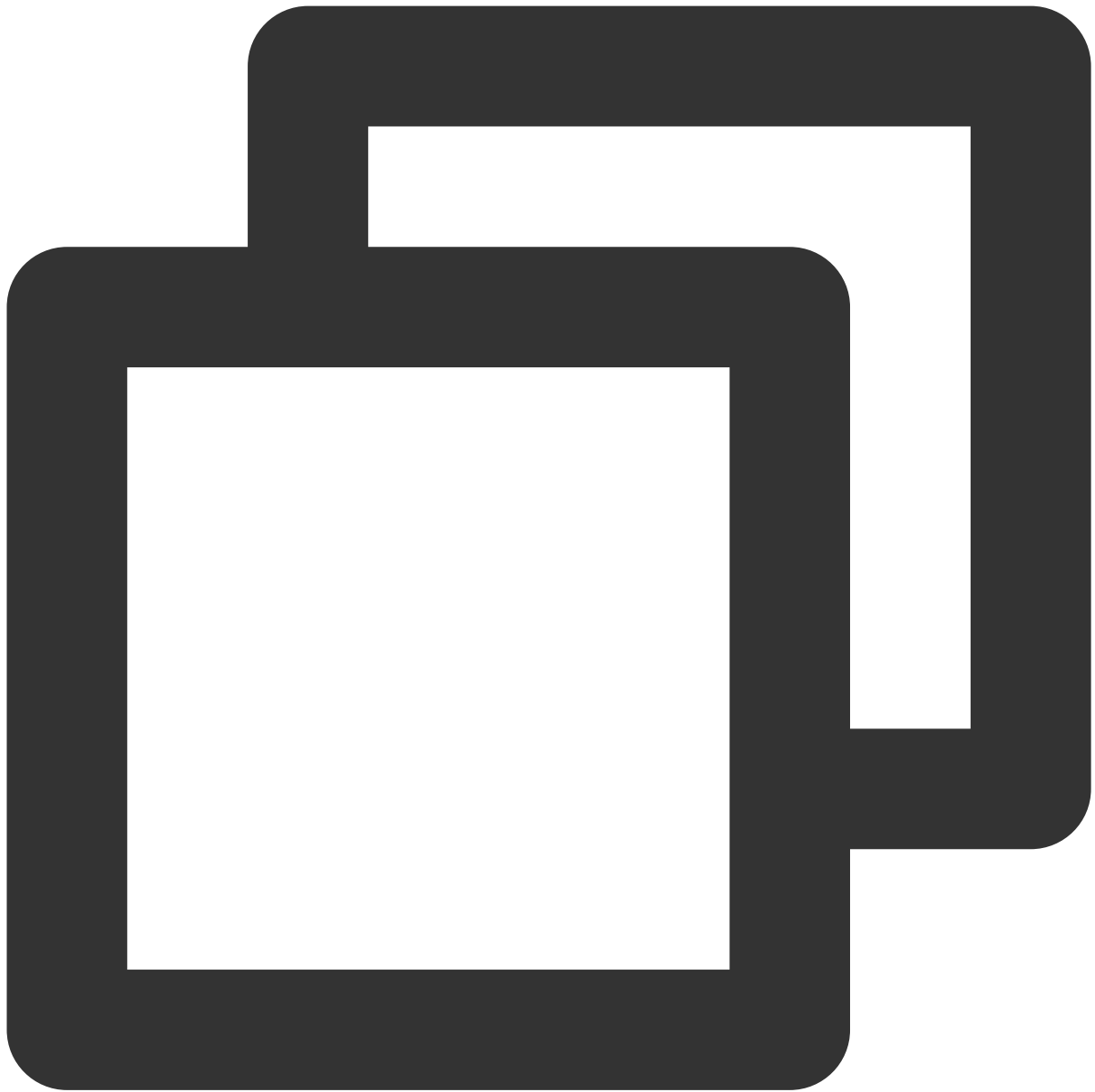
Method 2: Control the Floating Window and the Full Screen on/off through attribute control.

The `allowedMinimized` attribute controls the enabling/disabling of the Floating Window.

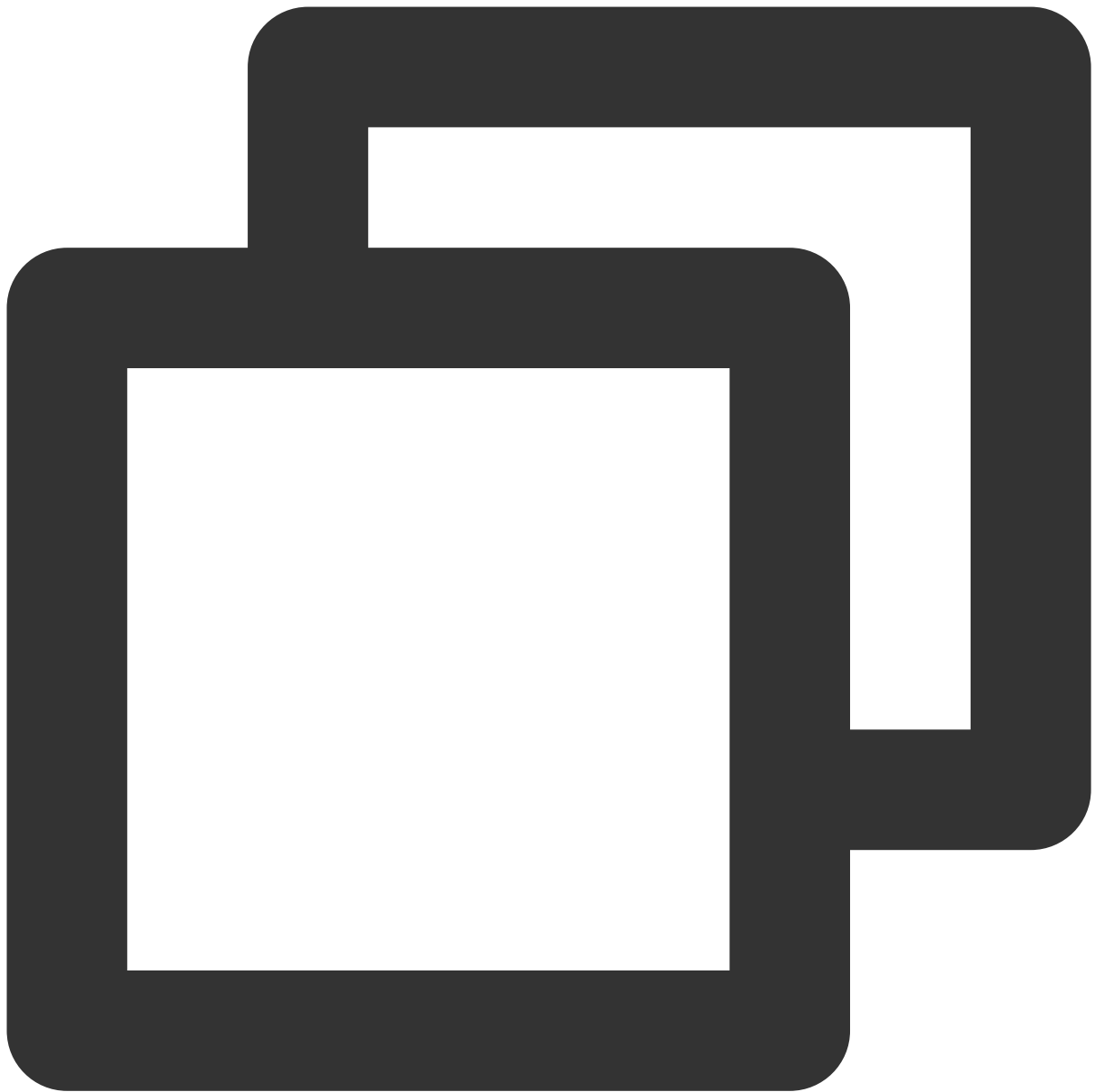
The `allowedMinimized` attribute controls the enabling/disabling of the Full Screen.

React

Vue



```
<TUICallKit
  allowedMinimized={true}
  allowedFullScree={true}
/>
```



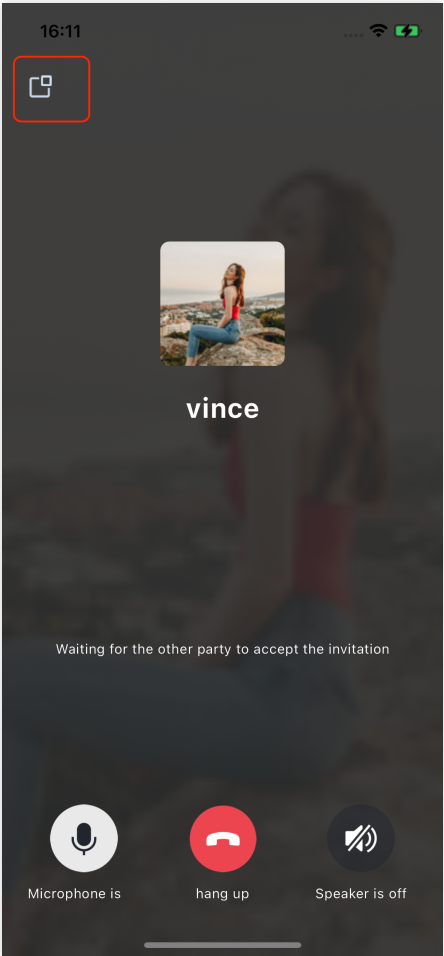
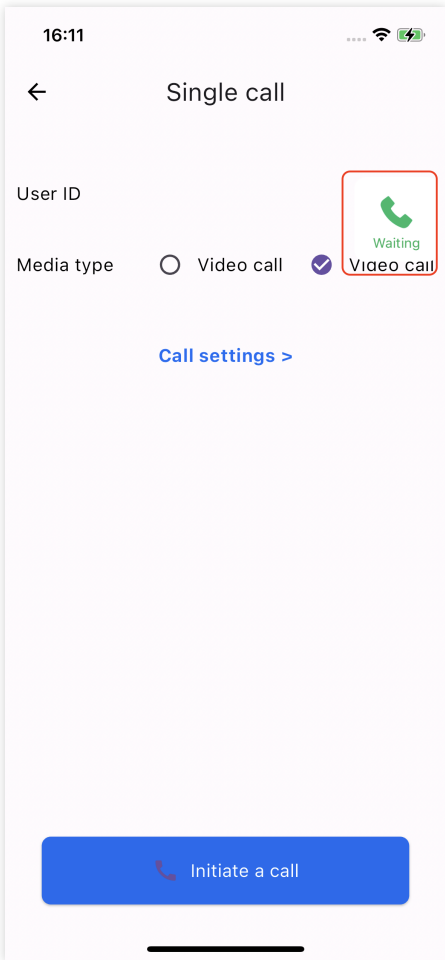
```
<TUICallKit
  :allowedMinimized="true"
  :allowedFullScreen="true"
/>
```

# uni-app (Anroid&iOS)

Last updated : 2024-04-03 17:23:11

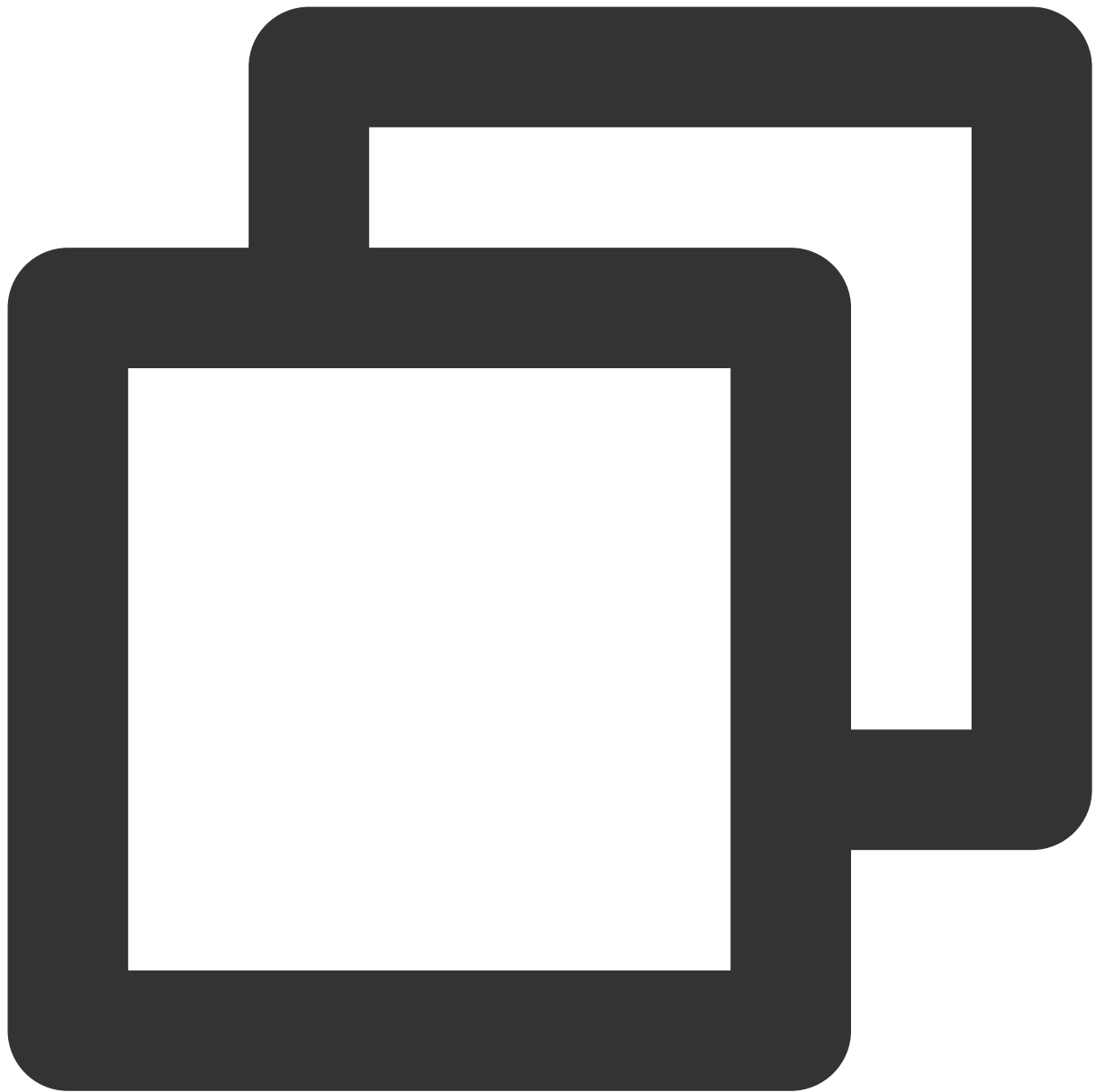
This article explains how to use the Floating Window feature.

## Expected outcome

| Activate floating button   | Voice call floating window  | Video call floating window |
|--|---|----------------------------|
| <div>A screenshot of a mobile app interface. At the top left, there is a small square button with a white icon on a dark background, highlighted by a red rectangle. Below it is a profile picture of a woman, the name 'vince', and the text 'Waiting for the other party to accept the invitation'. At the bottom, there are three circular buttons: 'Microphone is', 'hang up', and 'Speaker is off'.</div> | <div>A screenshot of a 'Single call' settings window. It shows 'User ID' and 'Media type' options. Under 'Media type', there are two radio buttons: 'Video call' (selected) and 'Voice call'. A red rectangle highlights the 'Video call' option. At the bottom, there is a blue button labeled 'Initiate a call'.</div> |                            |

## Floating Window feature

Invoke the `enableFloatWindow(enable: boolean)` API to enable/disable the floating window.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');  
const enable = true;  
TUICallKit.enableFloatWindow(enable);
```

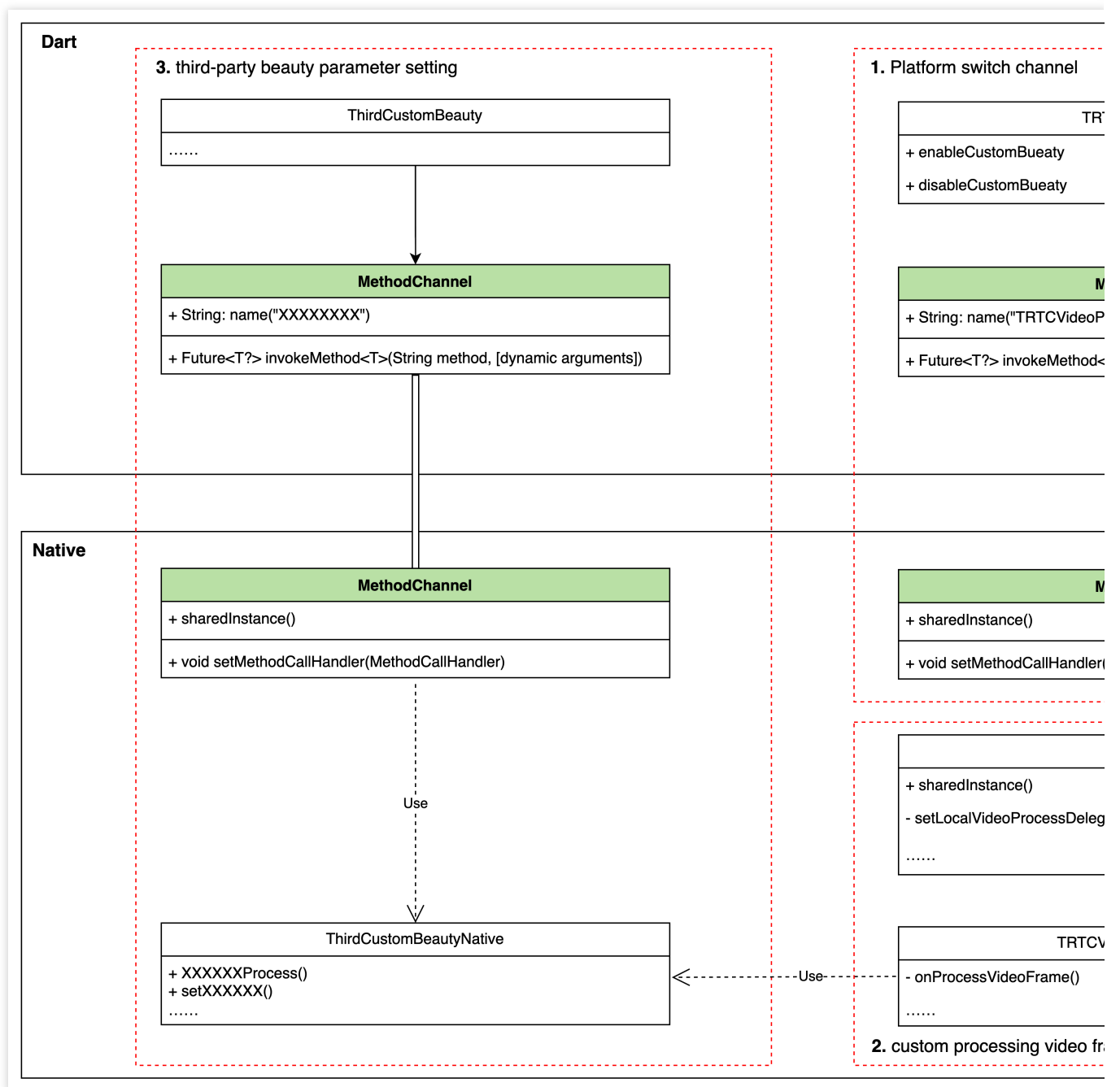
# Beauty Effects Flutter

Last updated : 2024-04-03 17:23:11

This document mainly introduces the method of integrating beauty effects in TUICallKit.

To complete custom beauty processing in Flutter, it needs to be done through TRTC custom video rendering. Due to Flutter's characteristics of not being good at handling a large amount of real-time data transmission, the part involving TRTC custom video rendering needs to be completed in the Native part. The specific plan is as follows:





The access plan is divided into 3 steps:

Step 1: Enable/disable TRTC custom rendering logic through the MethodChannel.

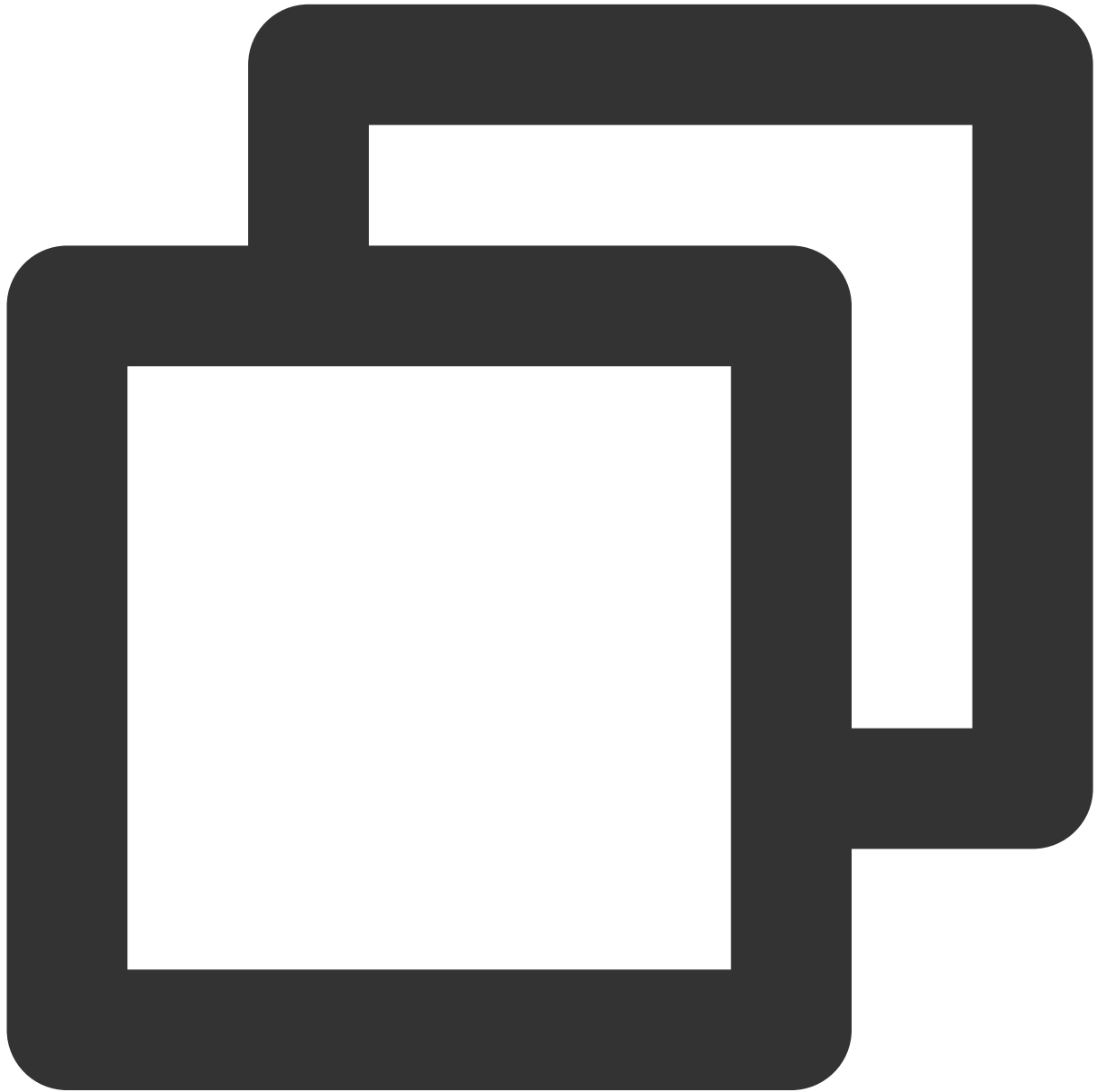
Step 2: Use the beauty processing module in TRTC's custom rendering processing logic onProcessVideoFrame() to process the original video frame.

Step 3: The customer's beauty processing module also needs to set the current beauty parameters through the interface in Dart. Customers can set beauty parameters through the MethodChannel method. This part can be customized by the customer according to their needs and the beauty they use.

# Integrate third-party beauty effects

## Step 1: Implement the start/end beauty control interface from Dart layer to Native

Implement Dart layer interface:



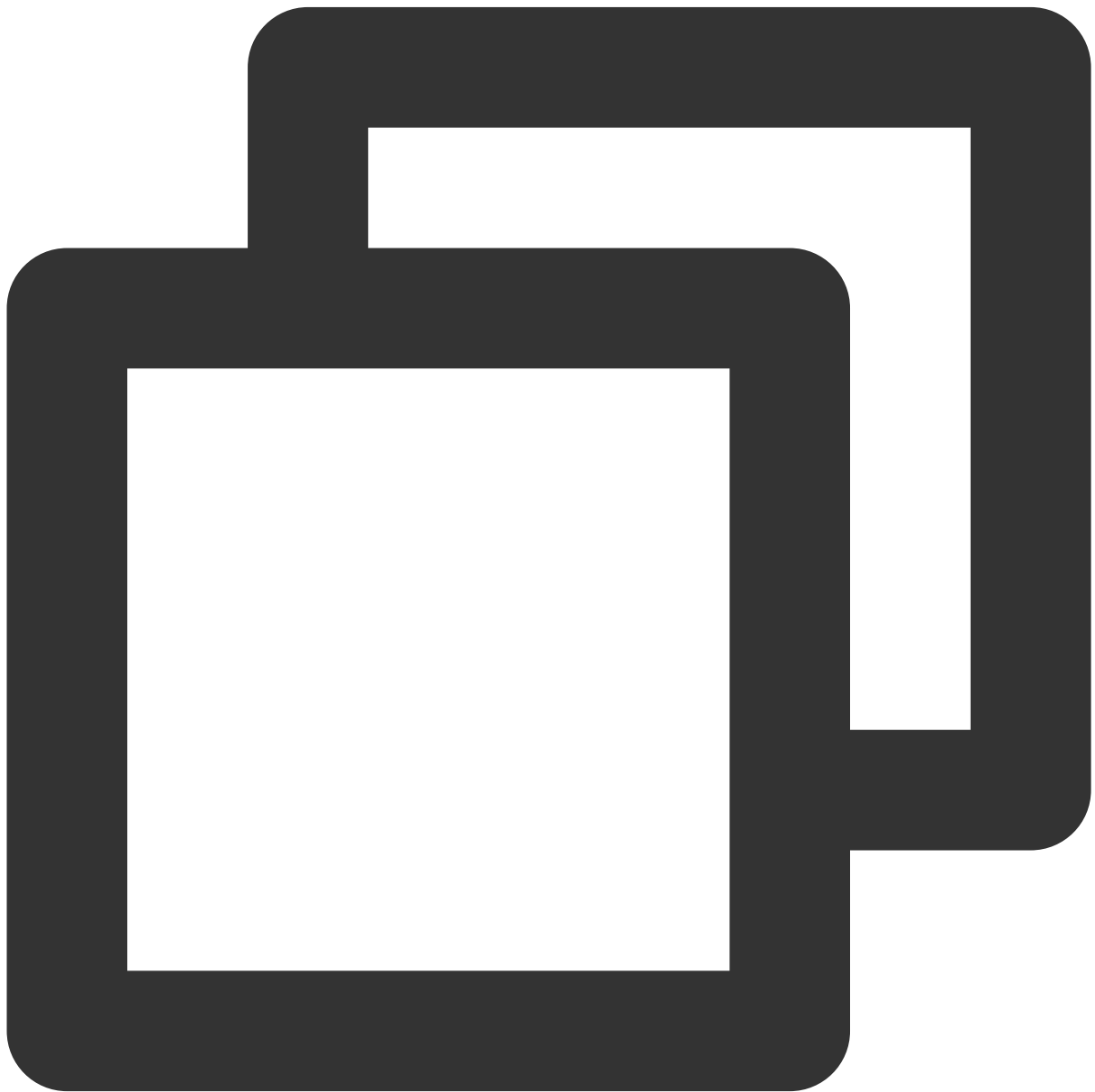
```
final channel = MethodChannel('TUICallKitCustomBeauty');

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

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

Implement the corresponding Native layer interface:

java  
swift



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

```
private MethodChannel channel;

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

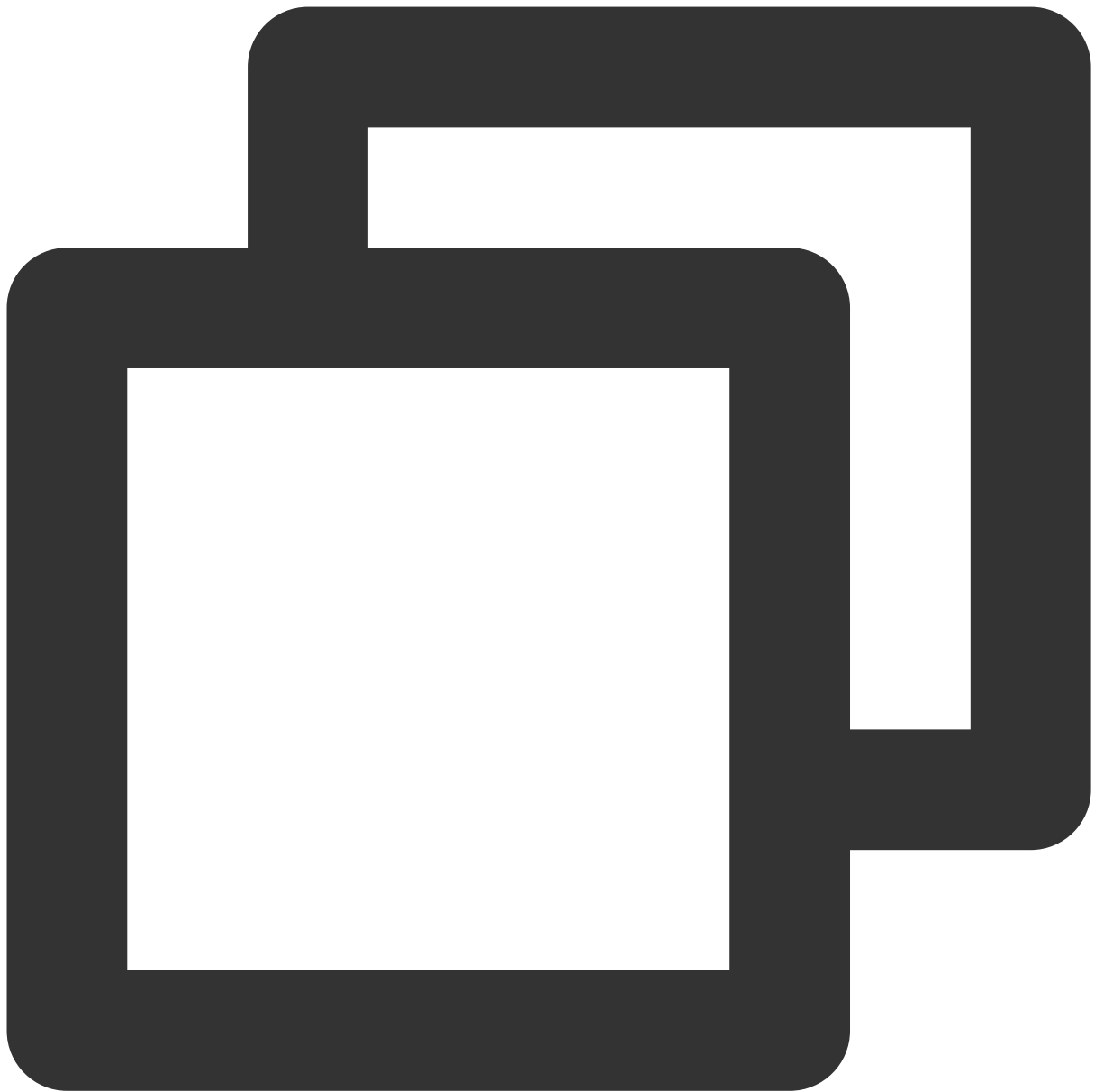
    channel = new MethodChannel(flutterEngine.getDartExecutor().getBinaryMessenger(),
        "flutter_tencent_tui_callkit_custom_beauty");
    channel.setMethodCallHandler(((call, result) -> {
        switch (call.method) {
            case "enableTUICallKitCustomBeauty":
                enableTUICallKitCustomBeauty();
                break;
            case "disableTUICallKitCustomBeauty":
                disableTUICallKitCustomBeauty();
                break;
            default:
                break;
        }
        result.success("");
    }));
}

public void enableTUICallKitCustomBeauty() {

}

public void disableTUICallKitCustomBeauty() {

}
}
```



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

  override func application(_ application: UIApplication,
                           didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) throws {
    GeneratedPluginRegistrant.register(with: self)

    guard let controller = window?.rootViewController as? FlutterViewController else {
      fatalError("Invalid root view controller")
    }
  }
}
```

```

        channel = FlutterMethodChannel(name: "TUICallKitCustomBeauty", binaryMessen
channel?.setMethodCallHandler({ [weak self] call, result in
    guard let self = self else { return }
    switch (call.method) {
    case "enableTUICallKitCustomBeauty":
        self.enableTUICallKitCustomBeauty()
        break
    case "disableTUICallKitCustomBeauty":
        self.disableTUICallKitCustomBeauty()
        break
    default:
        break
    }
}))
result(nil)
return super.application(application, didFinishLaunchingWithOptions: launch
}

func enableTUICallKitCustomBeauty() {

}

func disableTUICallKitCustomBeauty() {

}
}

```

## Step 2: Complete beauty processing in Native TRTC custom rendering logic

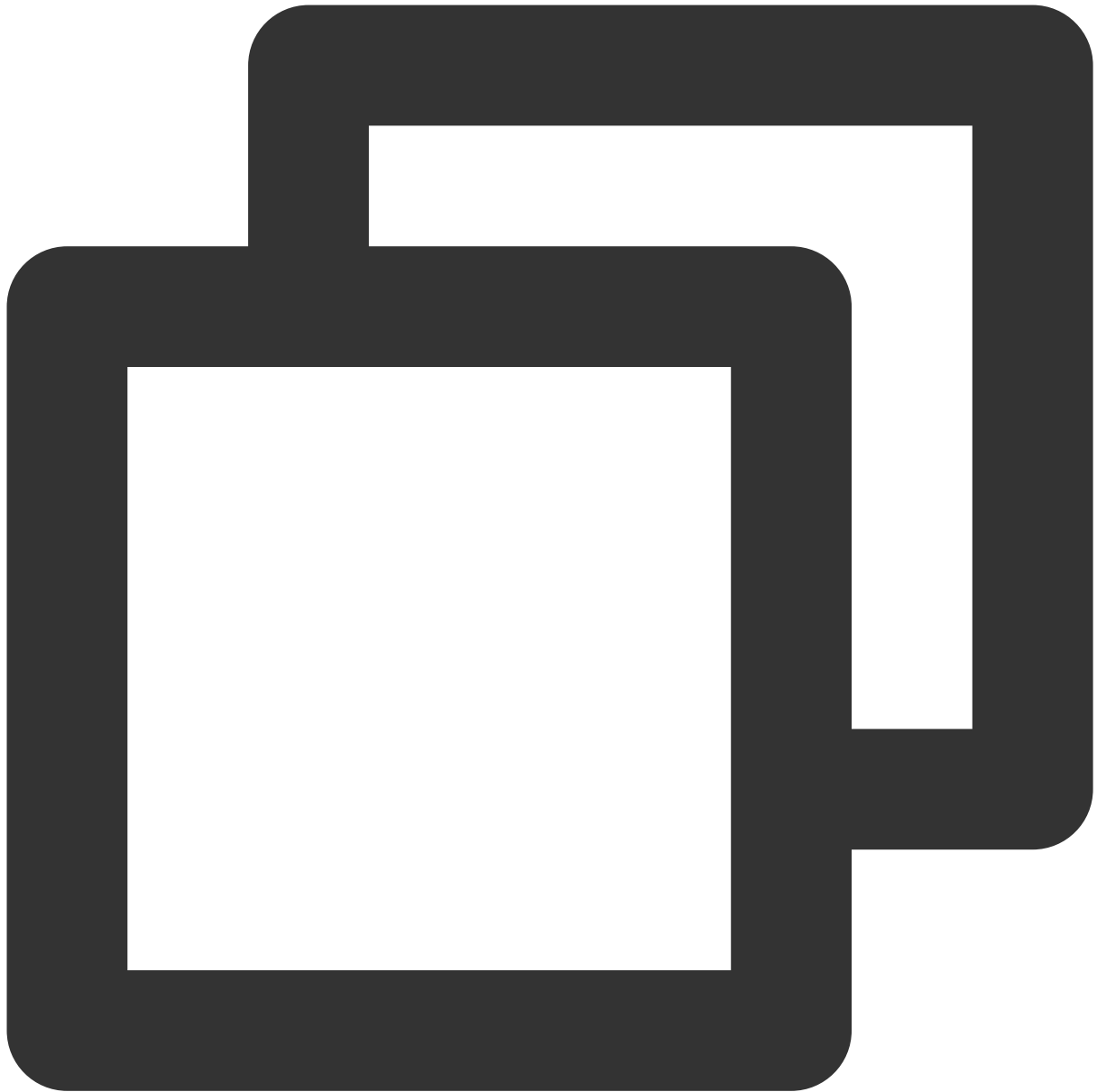
### Note :

Android needs to rely on `LiteAVSDK_Professional` first when accessing beauty. Add the following dependencies in the `app/build.gradle` of the Android project:

```

dependencies{
    api "com.tencent.liteav:LiteAVSDK_Professional:latest.release"
}
java
swift

```



```
void enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
        setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                                     TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new VideoFrameListerer());
}

void disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
        setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                                     TRTC_VIDEO_BUFFER_TYPE_TEXTURE, null);
}
```

```
}

class VideoFrameListerer implements TRTCCloudListener.TRTCVideoFrameListener {
    private XXXBeautyModel mBeautyModel = XXXBeautyModel.sharedInstance();

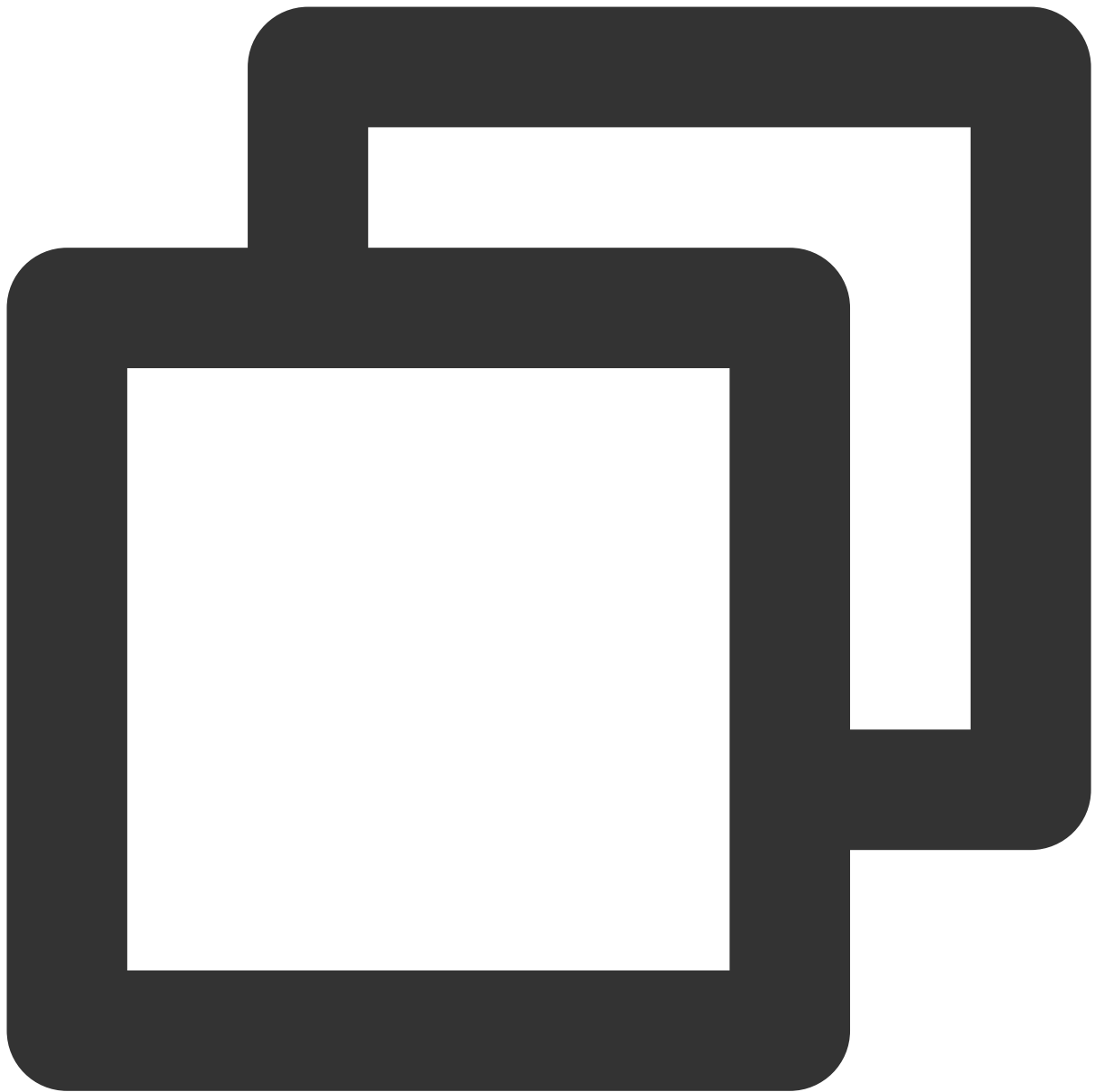
    @Override
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame trtcVideoFrame,
                                   TRTCCloudDef.TRTCVideoFrame trtcVideoFrame1) {
        // Beauty processing logic
        mBeautyModel.process(trtcVideoFrame, trtcVideoFrame1);
        .....

        return 0;
    }

    @Override
    public void onGLContextCreated() {
    }

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





```
let videoFrameListener: TRTCVideoFrameListener = TRTCVideoFrameListener()

func enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(videoFrameListener, pix

}

func disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(nil, pixelFormat: ._Tex
}

class TRTCVideoFrameListener: NSObject, TRTCVideoFrameDelegate {
```

```
let beautyModel = XXXXBeautyModel.shareInstance()
func onProcessVideoFrame(_ srcFrame: TRTCVideoFrame, dstFrame: TRTCVideoFrame)
    // Beauty processing logic
    beautyModel.onProcessVideoFrame(srcFrame, dstFrame)
    .....

    return 0
}
}
```

### Step 3: Customer-defined third-party beauty parameter control logic

In this part, customers can set beauty parameters according to their needs and the specific beauty module they use, referring to the implementation in [step 1](#). The specific implementation depends on the specific usage.

## Integrating Tencent Beauty Effects

The integration method of Tencent Beauty Effects also follows the above method. Now, taking Tencent Beauty Effects as an example, we will introduce the integration method in detail:

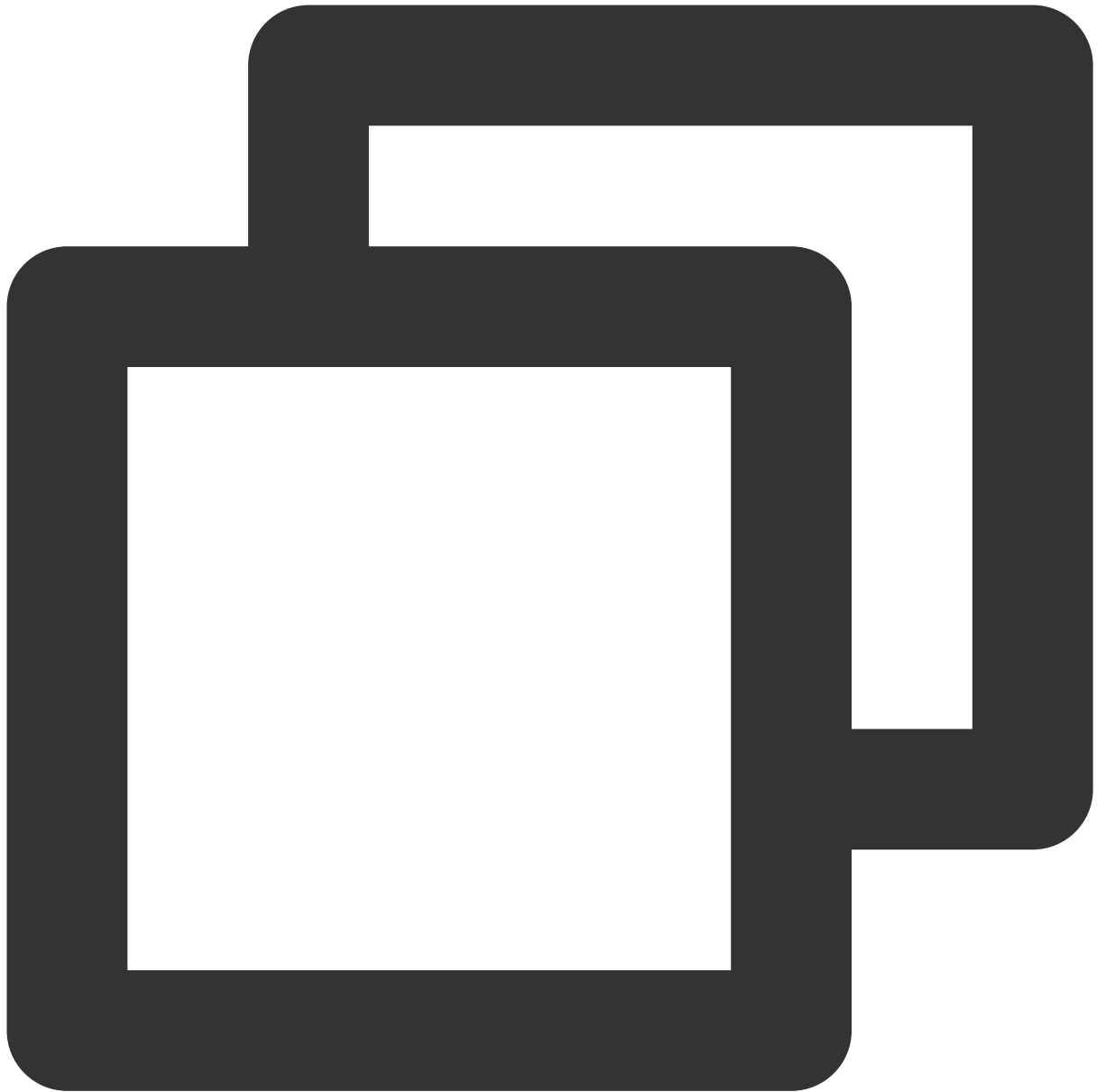
### Step 1: Beauty resource download and integration

1. [Download the SDK](#) according to the package you purchased.
2. Add files to your project:

Android

iOS

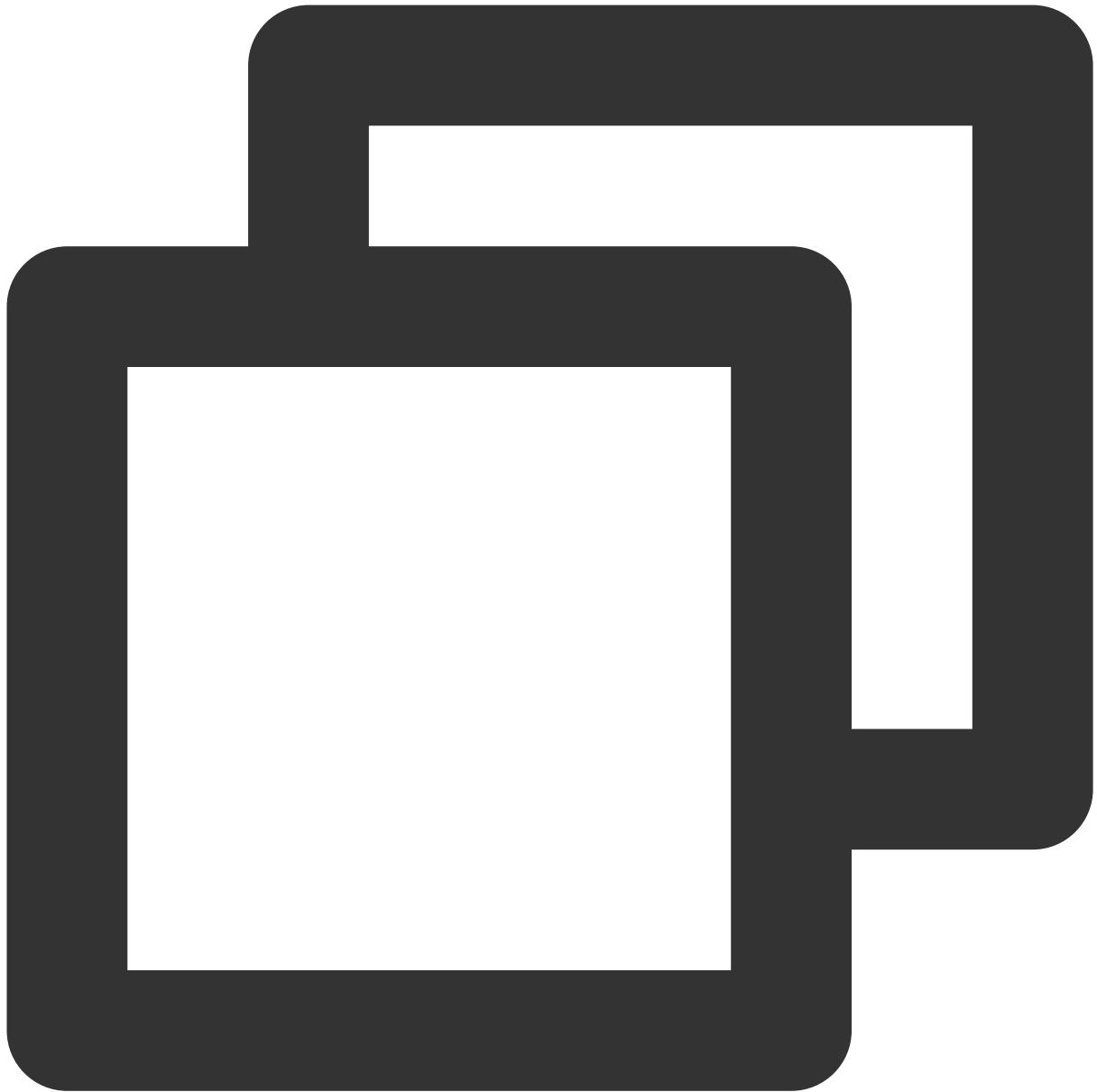
1. Find the build.gradle file under the app module and add the maven reference address for your corresponding package. For example, if you choose the S1-04 package, add the following:



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

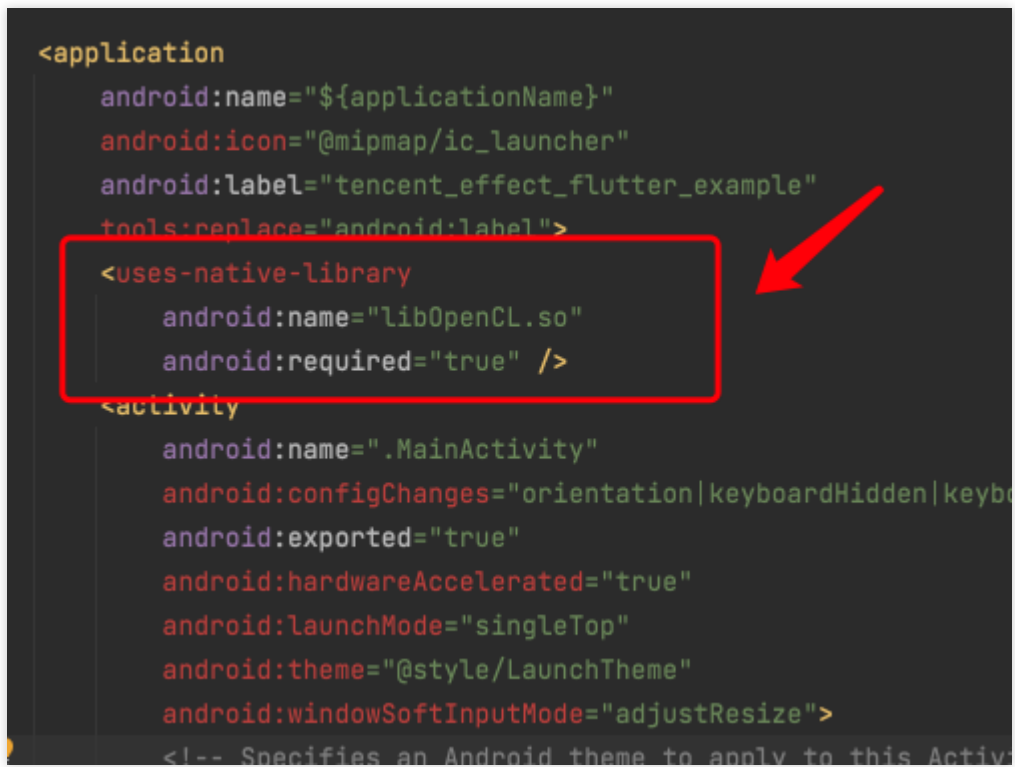
**For the maven addresses corresponding to each package, please refer to the [documentation](#).**

2. Find the `src/main/assets` folder under the app module. If it doesn't exist, create it. Check if there is a `MotionRes` folder in the downloaded SDK package. If so, copy this folder to the `../src/main/assets` directory.
3. Find the `AndroidManifest.xml` file under the app module and add the following tag in the application form



```
<uses-native-library
    android:name="libOpenCL.so"
    android:required="true" />
//The "true" here means that if this library is not present, the applicatio
// "false" means that the application can use this library (if it exists)
// Android official website introduction: https://developer.android.com/gu
```

Add as shown in the following figure:

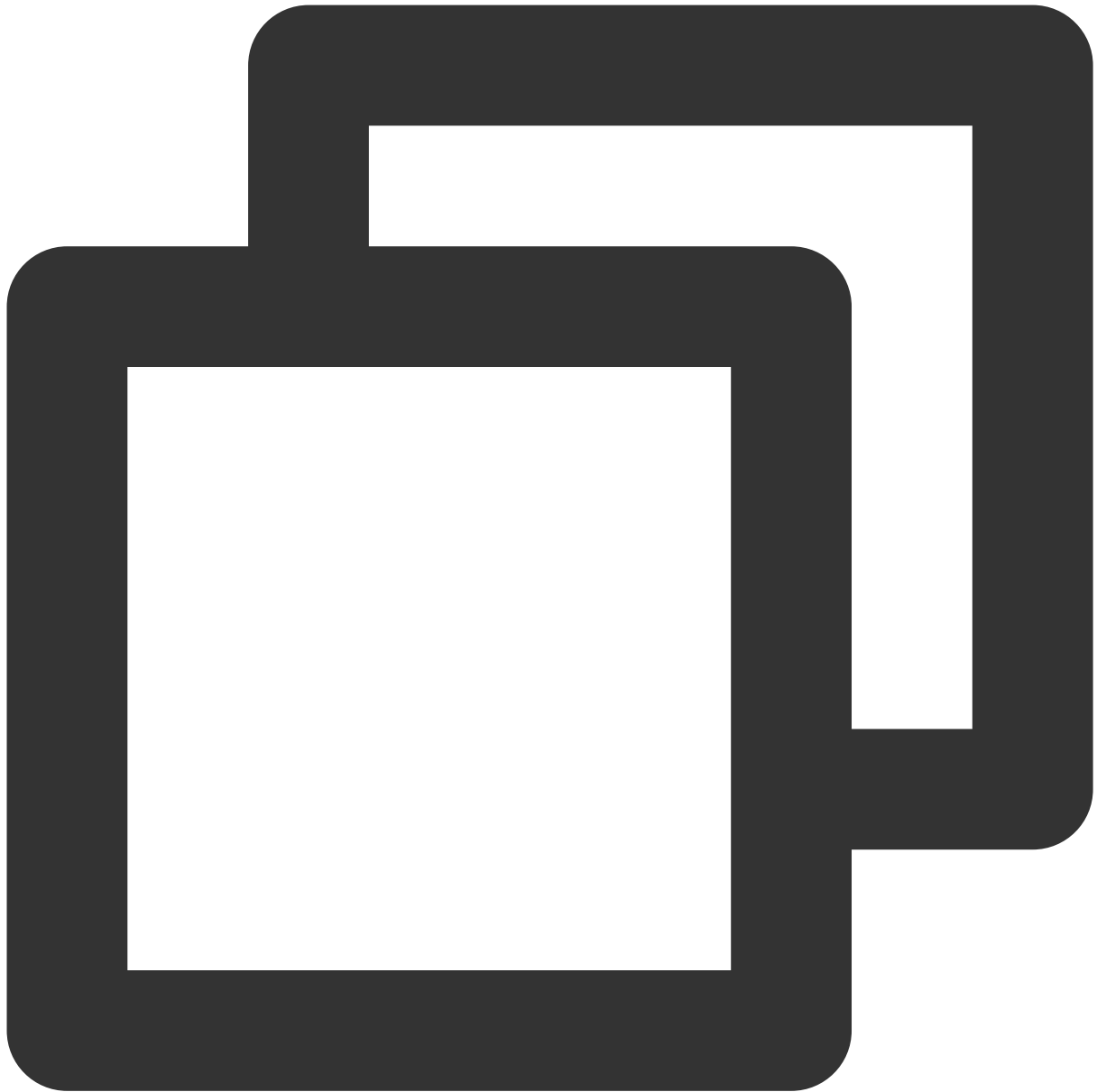


```
<application
    android:name="${applicationName}"
    android:icon="@mipmap/ic_launcher"
    android:label="tencent_effect_flutter_example"
    tools:replace="android:label">
    <uses-native-library
        android:name="libOpenCL.so"
        android:required="true" />
    <activity
        android:name=".MainActivity"
        android:configChanges="orientation|keyboardHidden|keybo
        android:exported="true"
        android:hardwareAccelerated="true"
        android:launchMode="singleTop"
        android:theme="@style/LaunchTheme"
        android:windowSoftInputMode="adjustResize">
        <!-- Specifies an Android theme to apply to this Activ
```

#### 4. Obfuscation configuration

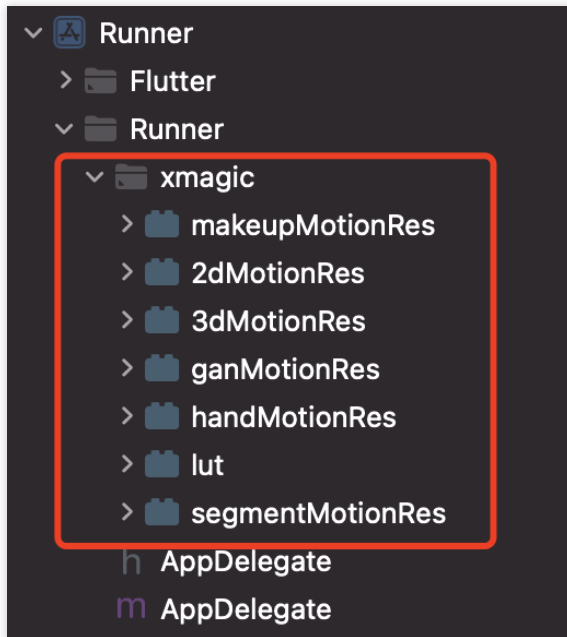
If you enable compile optimization when building a release package (set `minifyEnabled` to `true`), some code that is not called in the java layer will be cut off, and this code may be called by the native layer, causing a `no xxx method` exception.

If you enable such compile optimization, you need to add these keep rules to prevent xmagic code from being cut off:



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

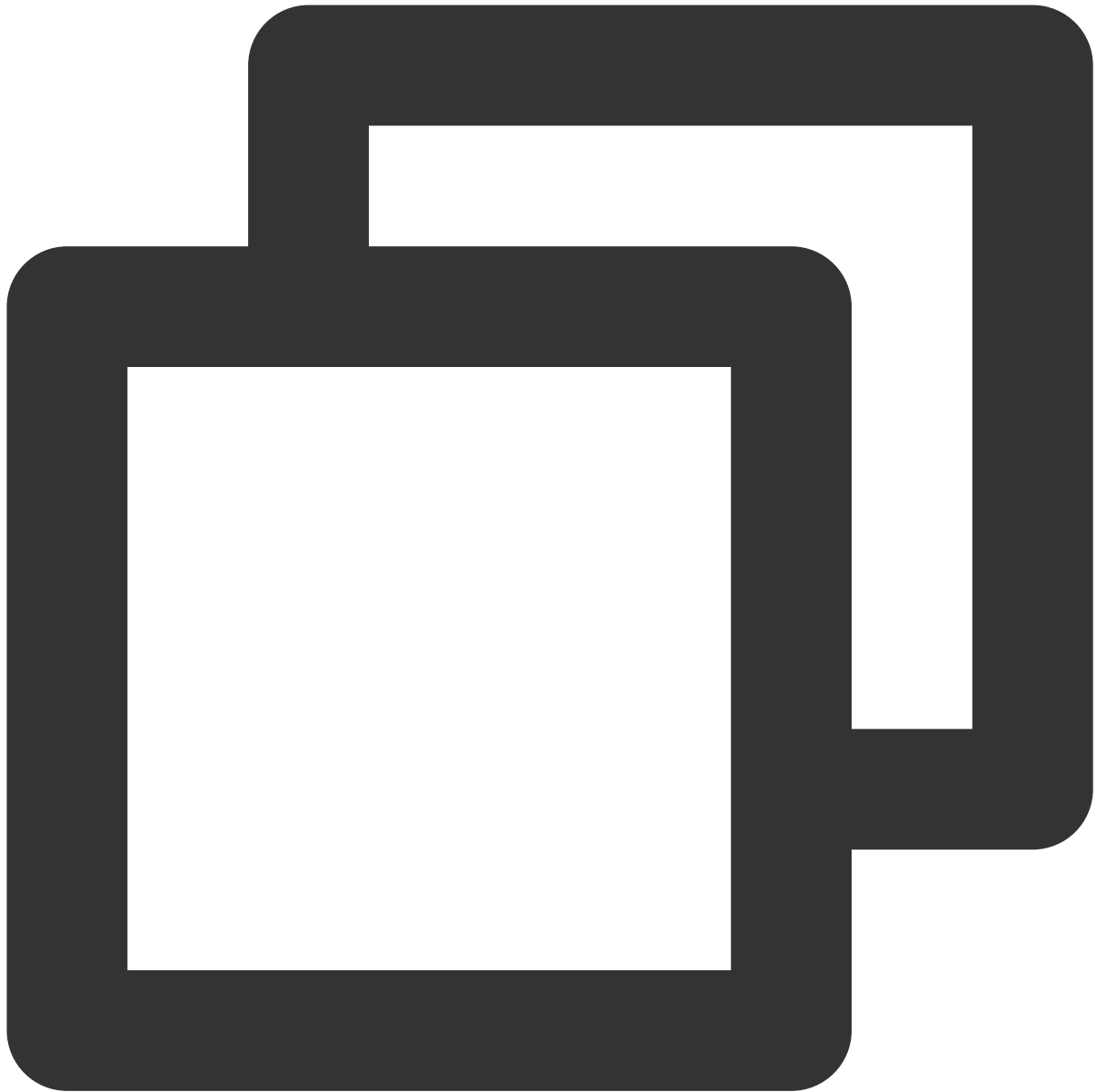
1. Add beauty resources to your project. After adding, it will be shown as in the following figure (the types of resources you have may not be exactly the same as in the figure):



2. In the demo, copy the 4 classes in demo/lib/producer: BeautyDataManager, BeautyPropertyProducer, BeautyPropertyProducerAndroid, and BeautyPropertyProducerIOS to your own Flutter project. These 4 classes are used to configure beauty resources and display beauty types on the beauty panel.

## Step 2: Reference the Flutter version SDK

**GitHub reference:** Add the following reference to the project's pubspec.yaml file:



```
tencent_effect_flutter:  
  git:  
    url: https://github.com/TencentCloud/tencenteffect-sdk-flutter
```

**Local reference:** Download the latest version of [tencent\\_effect\\_flutter](#) from tencent\_effect\_flutter, and then add the folders android, ios, lib, and the files pubspec.yaml, tencent\_effect\_flutter.iml to the project directory. Then, add the following reference to the project's pubspec.yaml file (refer to the demo):





```
tencent_effect_flutter:  
  path: ../
```

tencent\_effect\_flutter provides only a bridge, and the default version of the internally dependent XMagic is the latest. The real beauty effect is achieved by XMagic.

If you want to use the latest version of the beauty SDK, you can upgrade the SDK through the following steps:

Android

iOS

Execute the command `flutter pub upgrade` in the project directory, or click " Pub upgrade " in the upper right corner of the subspec.yaml page.

Execute the command `flutter pub upgrade` in the project directory, and then execute the command `pod update` in the iOS directory.

### Step 3: Implement the Dart layer to Native beauty control interface start/end

This section can refer to [Accessing third-party beauty effects/Step 1](#), and is not repeated here.

### Step 4: Complete the beauty processing in the TRTC custom rendering logic of Native

Android

iOS

Android needs to rely on `LiteAVSDK_Professional` first when accessing beauty. Add the following dependencies in the `app/build.gradle` of the Android project:

```
dependencies{  
    api "com.tencent.liteav:LiteAVSDK_Professional:latest.release"  
}
```



```
void enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
        setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                                     TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new VideoFrameListerer());
}

void disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
        setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                                     TRTC_VIDEO_BUFFER_TYPE_TEXTURE, null);
}
```

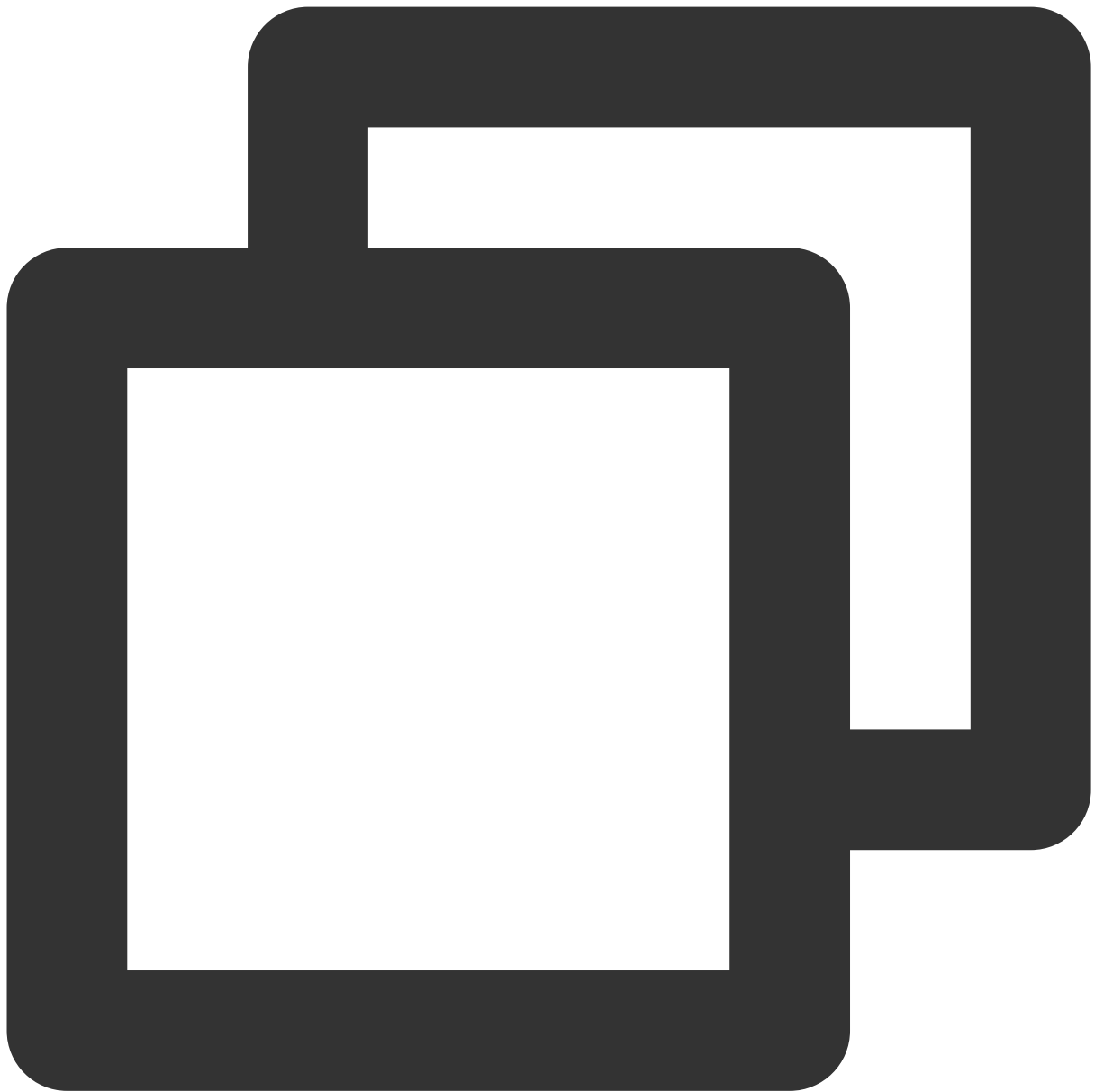
```
}

class VideoFrameListerer implements TRTCCloudListener.TRTCVideoFrameListener {
    private XXXBeautyModel mBeautyModel = XXXBeautyModel.sharedInstance();

    @Override
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame trtcVideoFrame,
                                   TRTCCloudDef.TRTCVideoFrame trtcVideoFrame1) {
        trtcVideoFrame1.texture.textureId = XmagicApiManager.getInstance()
            .process(trtcVideoFrame.texture.textureId, trtcVideoFrame.width, trtcVideoF
        return 0;
    }

    @Override
    public void onGLContextCreated() {
    }

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



```
import TXLiteAVSDK_Professional
import tencent_effect_flutter

let videoFrameListener: TRTCVideoFrameListener = TRTCVideoFrameListener()

func enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(videoFrameListener, pix
}

func disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(nil, pixelFormat: ._Tex
```

```
}

class TRTCVideoFrameListener: NSObject, TRTCVideoFrameDelegate {
    func onProcessVideoFrame(_ srcFrame: TRTCVideoFrame, dstFrame: TRTCVideoFrame)
        dstFrame.textureId = GLuint(XmagicApiManager.shareSingleton().getTextureId(
        return 0
    }
}

public class ConvertBeautyFrame: NSObject {
    public static func convertToTRTCPixelFormat(beautyPixelFormat: ITXCustomBeautyP
        switch beautyPixelFormat {
        case .Unknown:
            return ._Unknown
        case .I420:
            return ._I420
        case .Texture2D:
            return ._Texture_2D
        case .BGRA:
            return ._32BGRA
        case .NV12:
            return ._NV12
        }
    }

    public static func convertTRTCVideoFrame(trtcVideoFrame: TRTCVideoFrame) -> ITX
        let beautyVideoFrame = ITXCustomBeautyVideoFrame()
        beautyVideoFrame.data = trtcVideoFrame.data
        beautyVideoFrame.pixelBuffer = trtcVideoFrame.pixelBuffer
        beautyVideoFrame.width = UInt(trtcVideoFrame.width)
        beautyVideoFrame.height = UInt(trtcVideoFrame.height)
        beautyVideoFrame.textureId = trtcVideoFrame.textureId
        switch trtcVideoFrame.rotation {
        case ._0:
            beautyVideoFrame.rotation = .rotation_0
        case ._90:
            beautyVideoFrame.rotation = .rotation_90
        case ._180:
            beautyVideoFrame.rotation = .rotation_180
        case ._270:
            beautyVideoFrame.rotation = .rotation_270
        default:
            beautyVideoFrame.rotation = .rotation_0
        }
        switch trtcVideoFrame.pixelFormat {
        case ._Unknown:
```

```
        beautyVideoFrame.pixelFormat = .Unknown
    case ._I420:
        beautyVideoFrame.pixelFormat = .I420
    case ._Texture_2D:
        beautyVideoFrame.pixelFormat = .Texture2D
    case ._32BGRA:
        beautyVideoFrame.pixelFormat = .BGRA
    case ._NV12:
        beautyVideoFrame.pixelFormat = .NV12
    default:
        beautyVideoFrame.pixelFormat = .Unknown
    }
    beautyVideoFrame.bufferType = ITXCustomBeautyBufferType(rawValue: trtcVideo
    beautyVideoFrame.timestamp = trtcVideoFrame.timestamp
    return beautyVideoFrame
}
}
```

## Step 5: Enable beauty and set beauty parameters

After completing the above configuration, you can enable/disable beauty through `enableTUICallKitCustomBeauty()/disableTUICallKitCustomBeauty()`. You can set beauty parameters through the [Tencent beauty effects Flutter interface](#).

# Custom Ringtone

## Android

Last updated : 2024-04-03 17:23:11

This article explains how to replace the incoming call ringtone in TUICallKit. The incoming call ringtone is divided into **application ringtone** and **offline push ringtone**.

## Setting the application ringtone

There are two methods to set the application ringtone: replace ringtone audio, call Setting ringtone interface.

### 1. Replace Audio File

If you integrate the TUICallKit component via source code dependency, you can replace the three audio files in the `res\raw` folder to achieve the goal of replacing the ringtone:

| File Name         | Use                                    |
|-------------------|--|
| phone_dialing.mp3 | Ringtone when initiating a call        |
| phone_hangup.mp3  | Ringtone when the call is disconnected |
| phone_ringing.mp3 | Ringtone when receiving a call         |

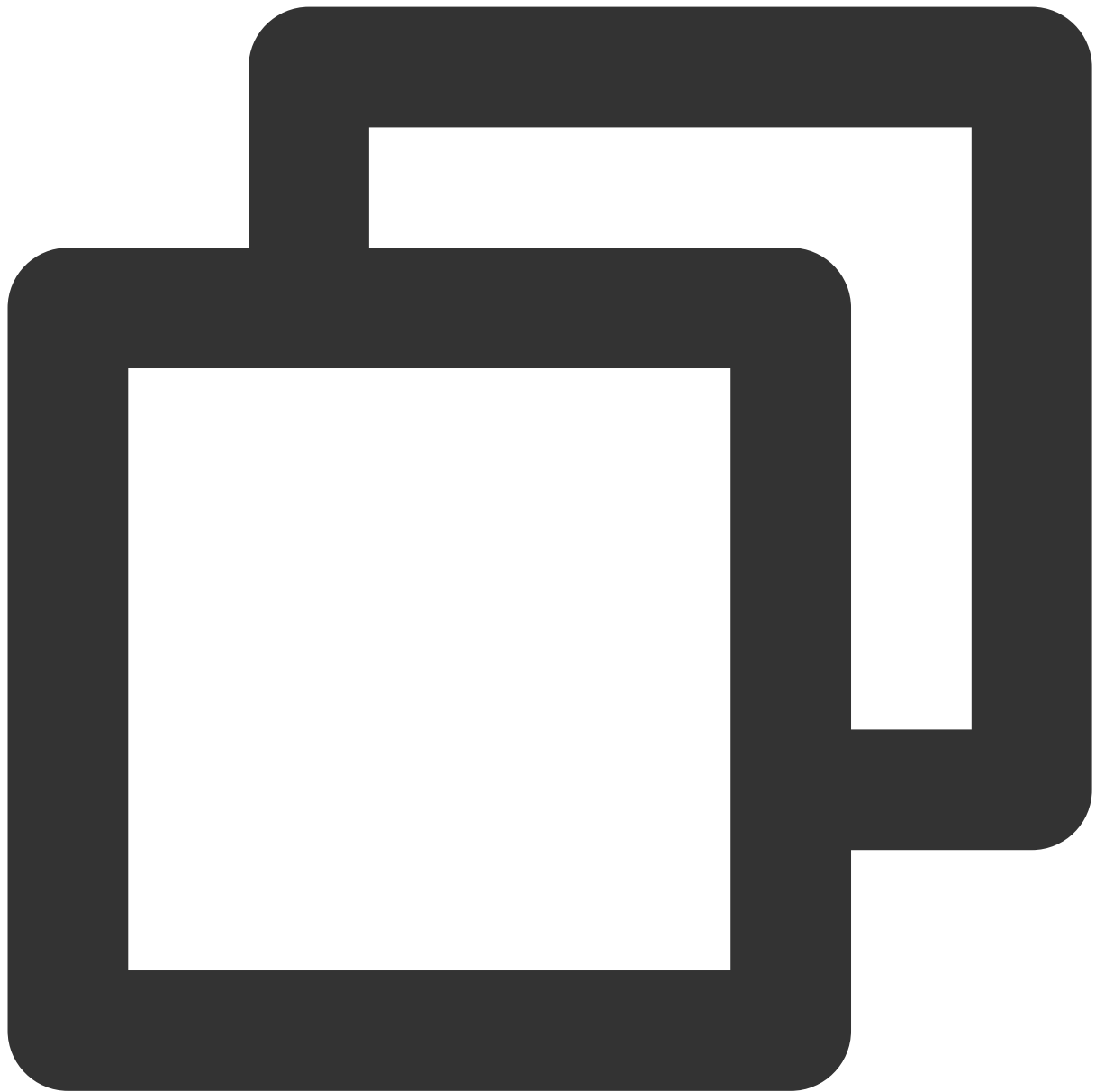
### 2. Setting Ringtone Interface

You can also customize the incoming call ringtone through the setCallingBell interface.

Kotlin

Java





```
TUICallKit.createInstance(context).setCallingBell(filePath)
```



```
TUICallKit.createInstance(context).setCallingBell(filePath);
```

### 3. Setting Mute Mode

If you do not need the phone to ring, you can enable the mute mode using the `enableMuteMode` setting.

Kotlin

Java



```
TUICallKit.createInstance(context).enableMuteMode(true)
```



```
TUICallKit.createInstance(context).enableMuteMode(true);
```

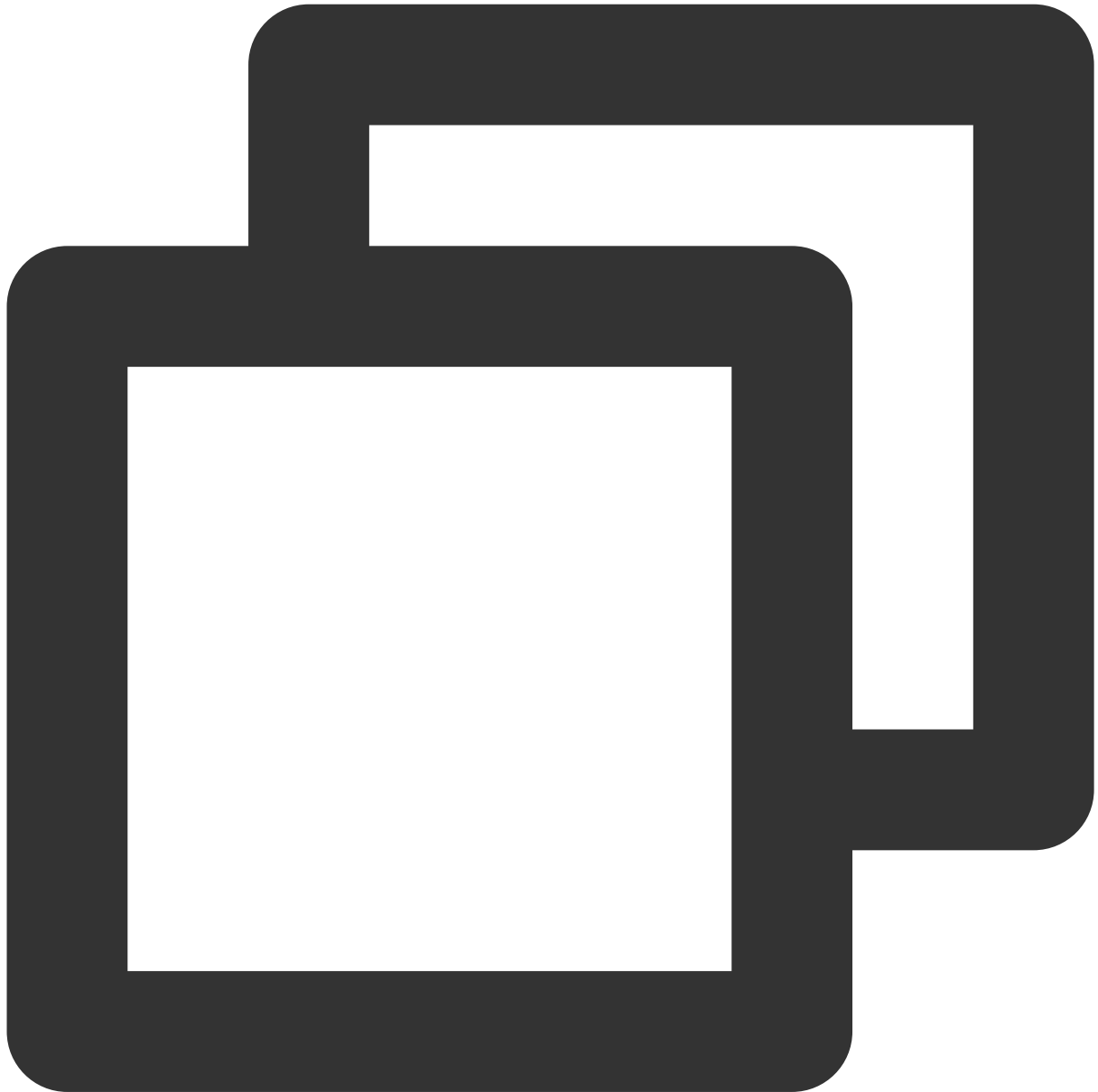
## Setting Offline Push Ringtone

Offline notification ringtones only support the following manufacturers for custom definitions: **Huawei, Xiaomi, FCM, and APNs**; other manufacturers such as OPPO, VIVO, Honor, etc., are not supported currently.

TUICallKit has introduced the TUIOfflinePush component. To enable offline notification custom ringtone capability, call the following method during application launch.

Kotlin

Java



```
class DemoApplication : Application() {  
    override fun onCreate() {  
        TUIOfflinePushConfig.getInstance().isAndroidPrivateRing = true  
    }  
}
```



```
public class DemoApplication extends Application {  
    @Override  
    public void onCreate() {  
        TUIOfflinePushConfig.getInstance().setAndroidPrivateRing(true);  
    }  
}
```

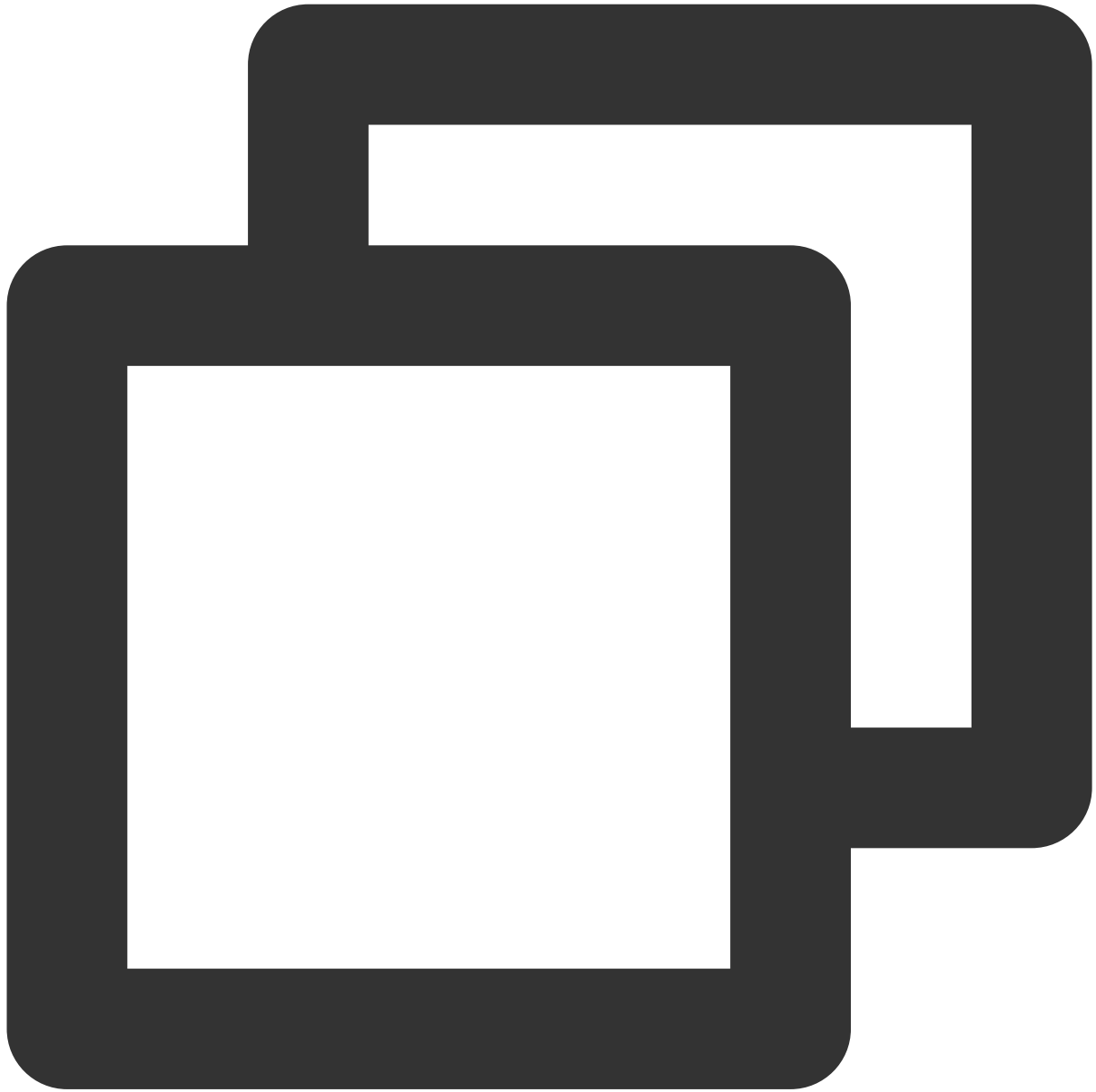
## 1. Huawei & APNs

Invoke the interfaces `setAndroidSound()` and `setIOSSound()`.

Customize the ringtone resource files, add the ringtone files to the local Android Studio project in the res/raw directory.

Kotlin

Java



```
val pushInfo = OfflinePushInfo()
pushInfo.iosSound = "ringtone name.mp3"
pushInfo.androidSound = "Ring Tone Name"
```



```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();  
pushInfo.setIOSSound("Ring Tone Name.mp3");  
pushInfo.setAndroidSound("Ring Tone Name");
```

#### Note

The IMSDK version 6.1.2155 and subsequent versions are supported.

The ringtone of Huawei's push message corresponds to the ringtone when the notification channel is first created, and subsequent settings for other ringtones do not take effect. If you want to update the user-defined notification ringtone, the application must be uninstalled and reinstalled. For details, see: [Custom Ringtones on Huawei](#).



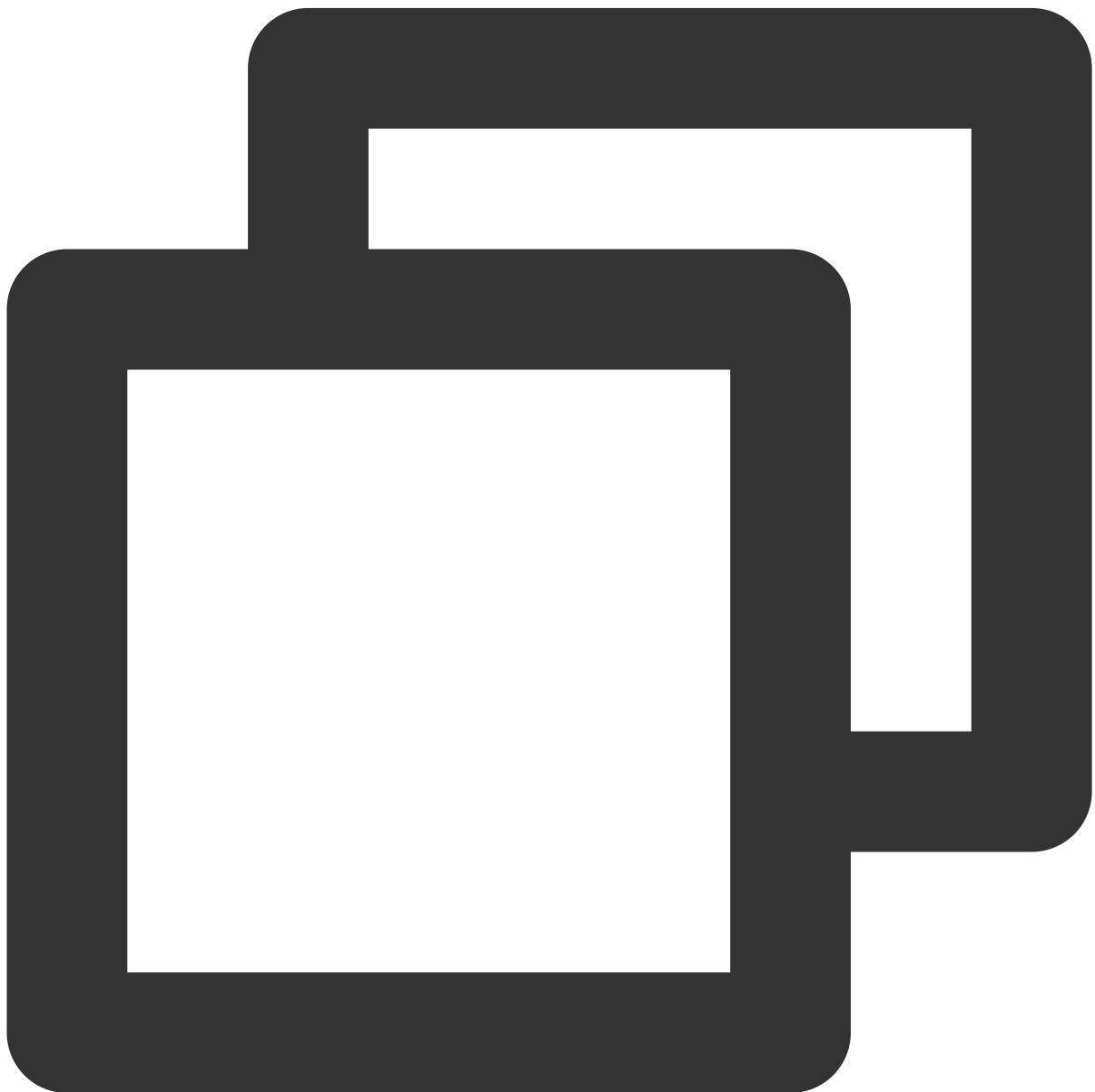
## 2. Xiaomi

(1) Before Android 8.0, use `setAndroidSound()` and `setIOSSound()` to customize the ringtone resource file by adding the ringtone file to the project's `res/raw` directory (refer to the Huawei invocation mentioned above).

(2) After Android 8.0, it's also necessary to sign in to the Mi Vendor Console and [create a channel and configure it](#), where the ringtone file needs to be added to the `res/raw` directory of the local Android Studio project.

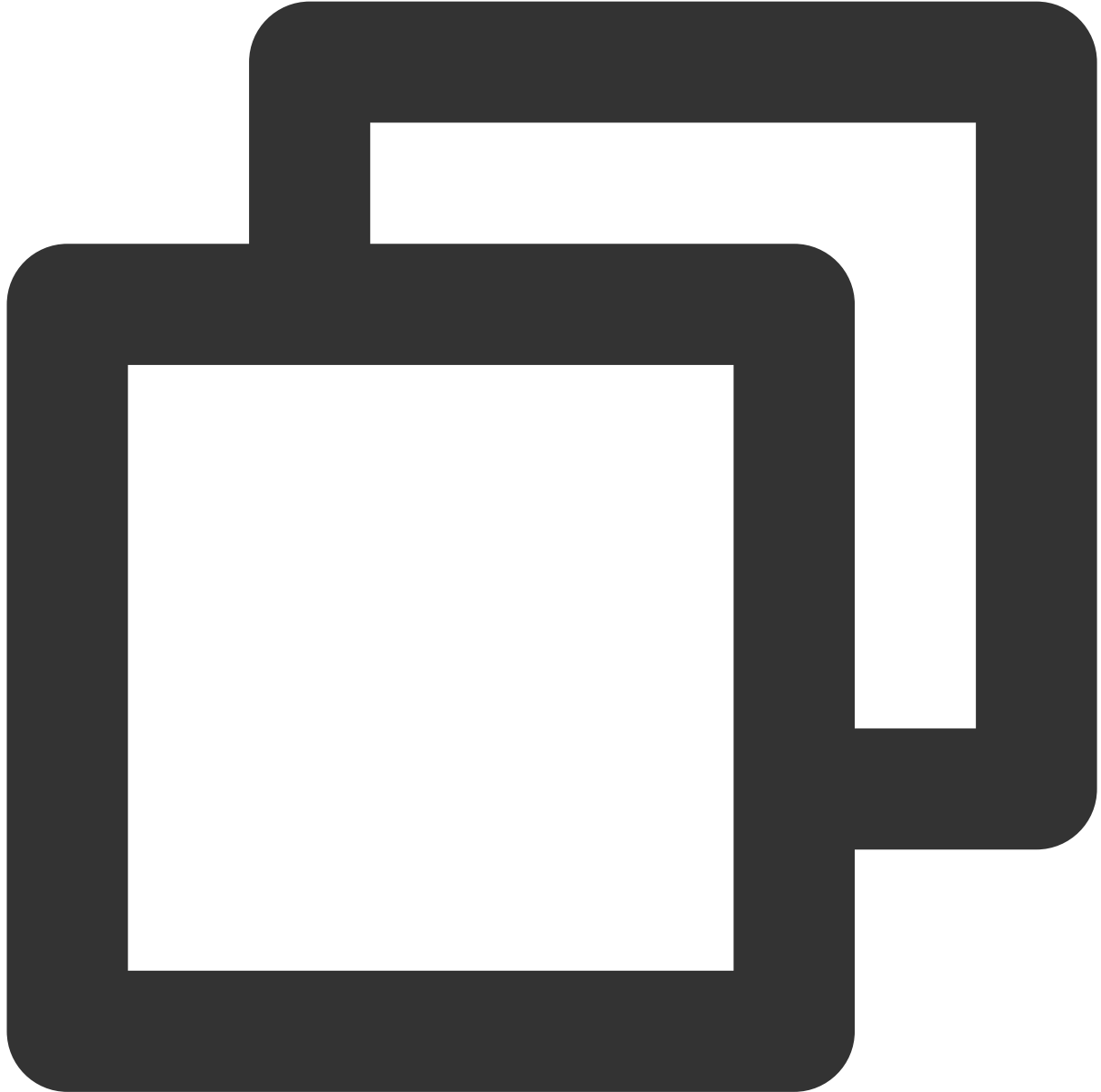
Kotlin

Java



```
val pushInfo = OfflinePushInfo()  
pushInfo.iosSound = "ringtone name.mp3"
```

```
pushInfo.androidSound = "Ring Tone Name"  
pushInfo.androidXiaoMiChannelID = "Vendor Requested Channel ID"
```



```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();  
pushInfo.setIOSSound("Ring Tone Name.mp3");  
pushInfo.setAndroidSound("Ring Tone Name");  
pushInfo.setAndroidXiaoMiChannelID("Vendor Requested Channel ID");
```

#### Note

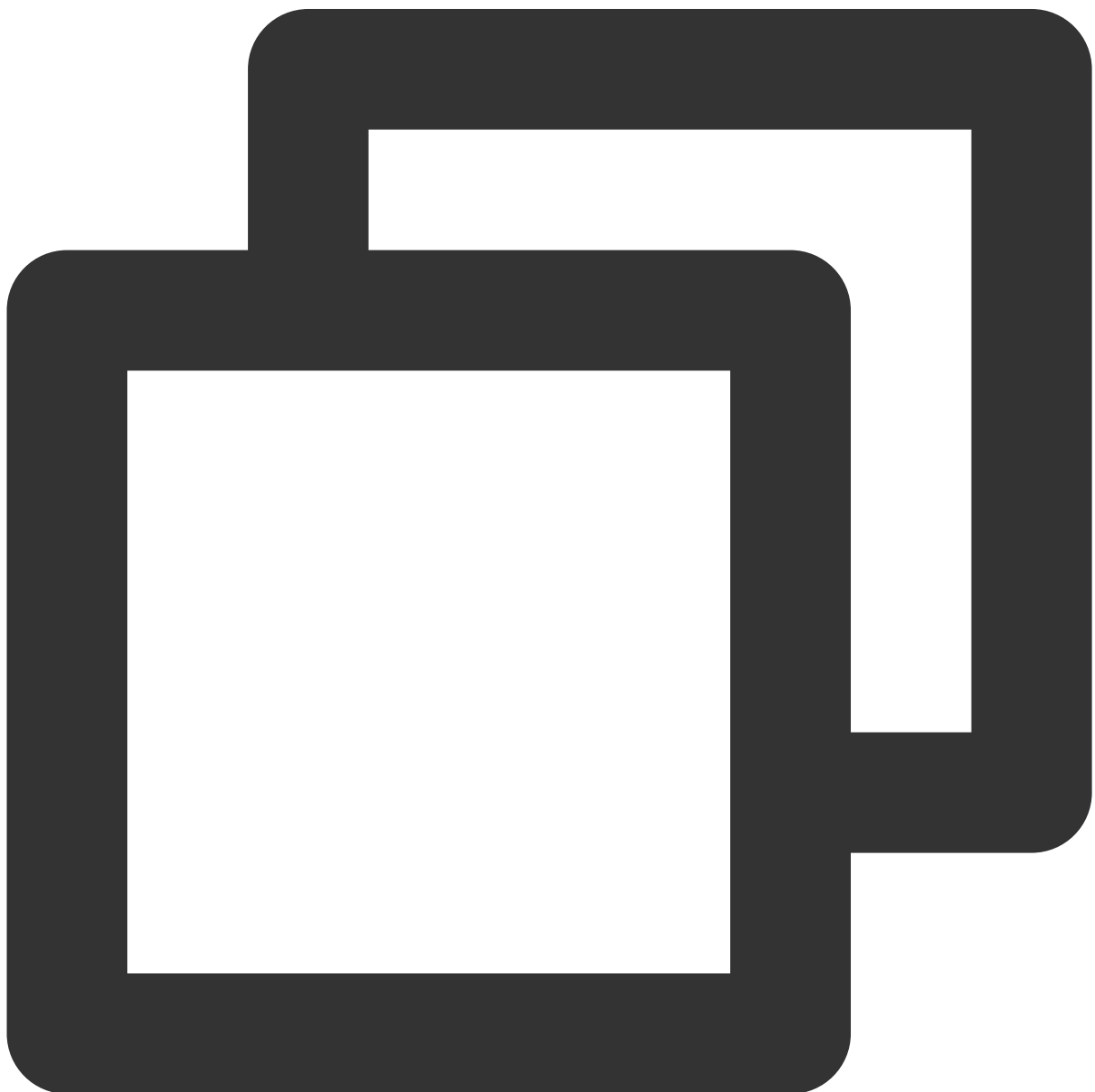
For versions before Android 8.0, supported by IMSDK version 6.1.2155 and above.

For versions after Android 8.0, supported by IMSDK version 7.0.3754 and above.

### 3. FCM

(1) Before Android 8.0, use `setAndroidSound()` and `setIOSSound()` to customize the ringtone resource file by adding the ringtone file to the project's `res/raw` directory (refer to the Huawei invocation mentioned above).

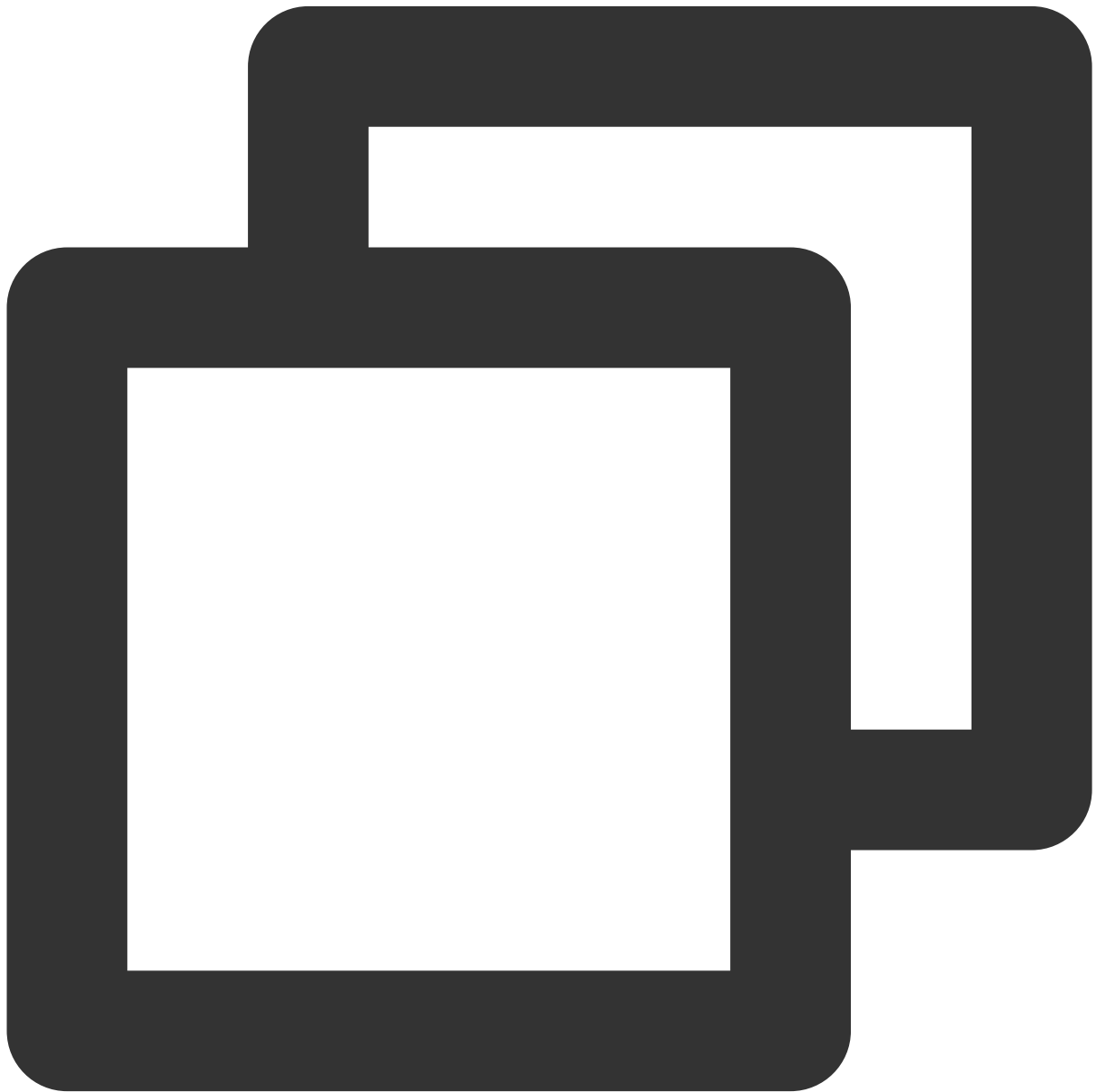
(2) After Android 8.0, FCM requires a configuration of `channelID` and ringtone resources. The `tuiofflinepush` component introduced by `TUICallKit` has already handled the playback of custom ringtones. The ringtone file needs to be added to the `res/raw` directory of your local Android Studio project, and the names of the ringtone and channel ID need to be specified. See [PrivateConstants](#) for more details.



```
public class PrivateConstants {  
    // For FCM channel, specify the channel ID  
    public static String fcmPushChannelId = "FCM ChannelID";  
    // For FCM, specify the push ringtone name of the channel, which should be cons  
    public static String fcmPushChannelSoundName = "ringtone name";  
}
```

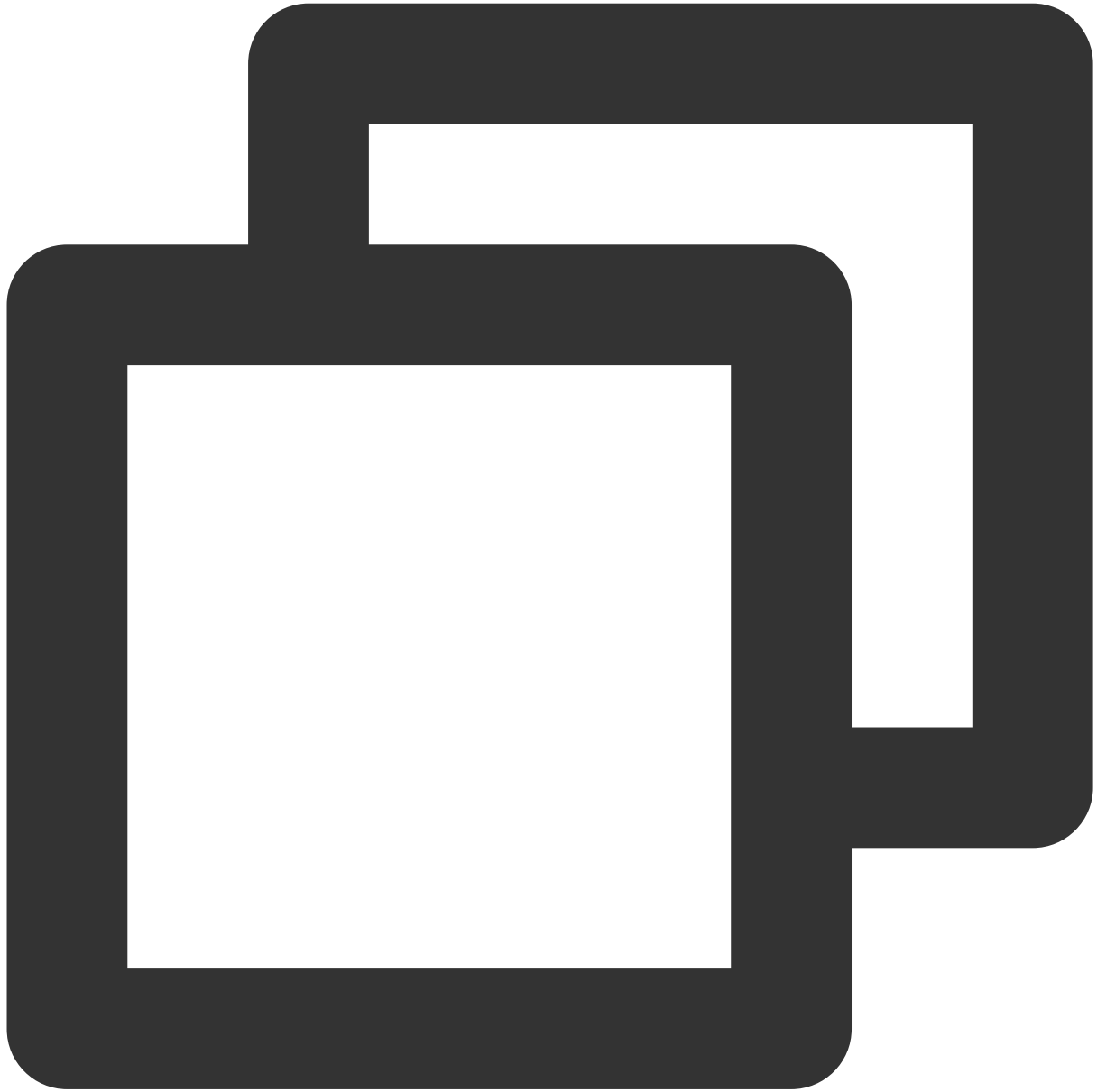
Kotlin

Java



```
val pushInfo = OfflinePushInfo()
```

```
pushInfo.iosSound = "ringtone name.mp3"  
pushInfo.androidSound = "Ring Tone Name"  
pushInfo.androidFCMChannelID = "Vendor Requested Channel ID"
```



```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();  
pushInfo.setIOSSound("Ring Tone Name.mp3");  
pushInfo.setAndroidSound("Ring Tone Name");  
pushInfo.setAndroidFCMChannelID("Vendor Requested Channel ID");
```

#### Note

For versions before Android 8.0, supported by IMSDK version 6.1.2155 and above.

For versions after Android 8.0, supported by IMSDK version 7.0.3754 and above.

FCM custom ringtone definition or setting channel id only supports certificate mode.

# iOS

Last updated : 2024-04-03 17:23:11

This article explains how to replace the incoming call ringtone for TUICallKit, which includes **application ringtone** and **offline push ringtone**.

## Setting application Ringtone

There are two ways to set the application ringtone: replace the ringtone audio, and call the Setting ringtone interface.

### 1. Replace Audio File

If you integrate the TUICallKit component via source code dependency, you can achieve the goal of replacing the ringtone by swapping out the three audio files under the `Resources\\AudioFile` folder:

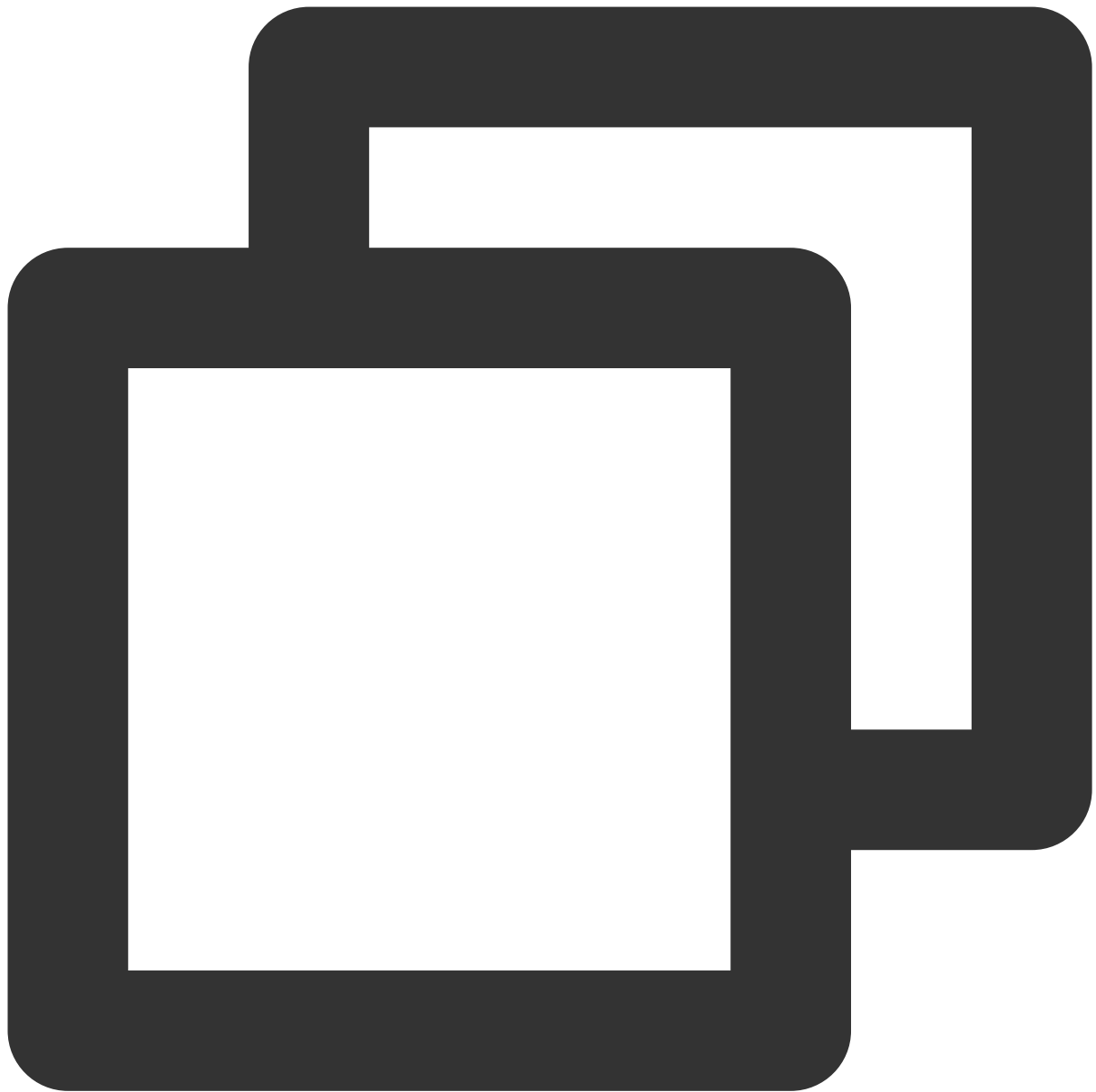
| File Name         | Use                                    |
|-------------------|--|
| phone_dialing.mp3 | Ringtone when initiating a call        |
| phone_hangup.mp3  | Ringtone when the call is disconnected |
| phone_ringing.mp3 | Ringtone when receiving a call         |

### 2. Set Ringtone Interface

You can also set the incoming call ringtone via the `setCallingBell` interface.

Swift

Objective-C



```
TUICallKit.createInstance().setCallingBell(filePath: " ")
```





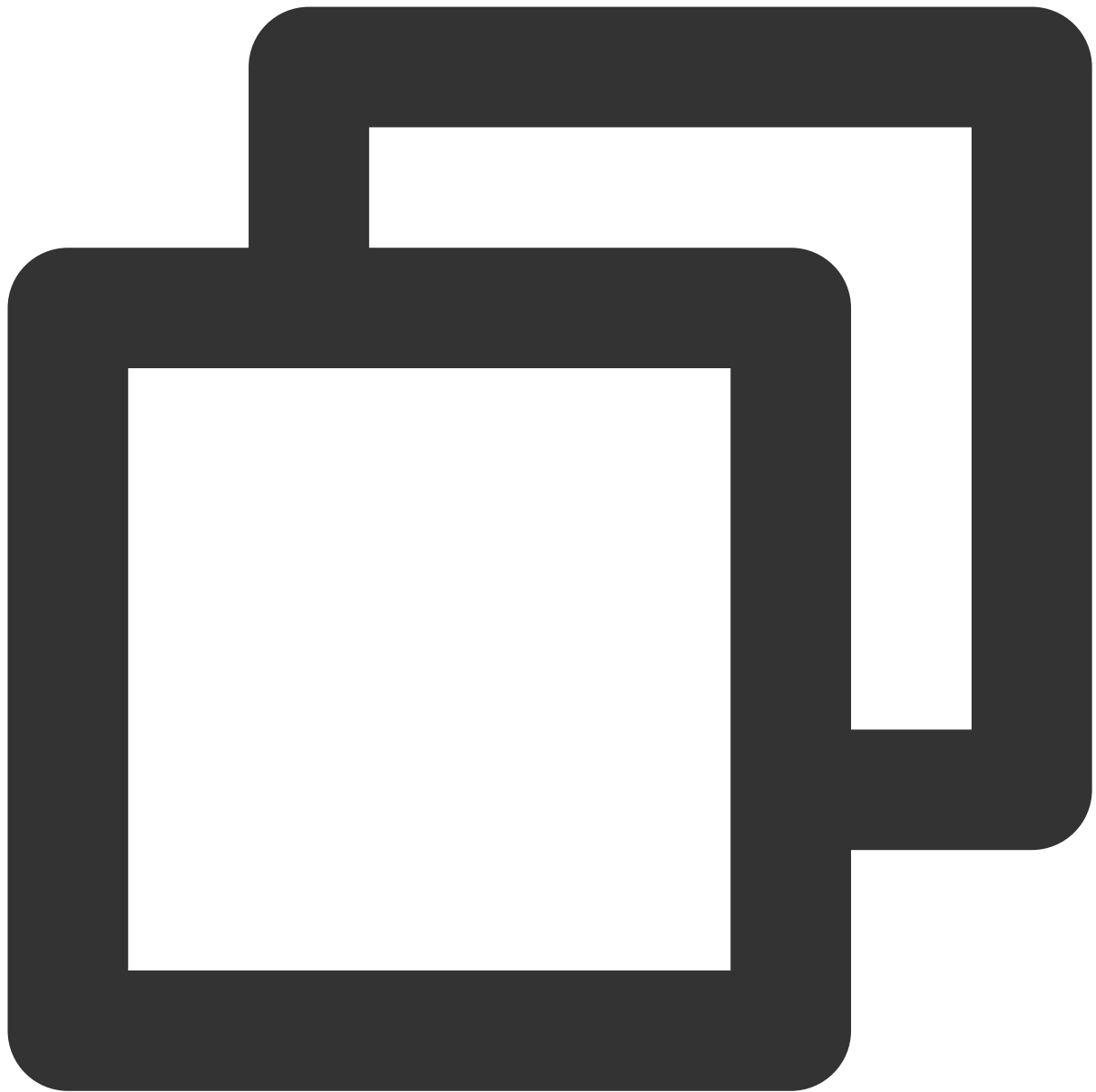
```
[[TUICallKit sharedInstance] setCallingBell:@" "];
```

### 3. Setting Mute Mode

If you do not require a ringtone, you can set the mute mode via `enableMuteMode`.

Swift

Objective-C



```
TUICallKit.createInstance().setCallingBell(enable: true)
```



```
[[TUICallKit sharedInstance] enableMuteMode: YES];
```

## Setting Offline Push Ringtone

VoIP push does not support custom push ringtones. APNs push can be set by specifying the `iOSSound` field in the `offlinePushInfo` params when making a call via the Call Interface. `iOSSound` should be passed the audio file name.

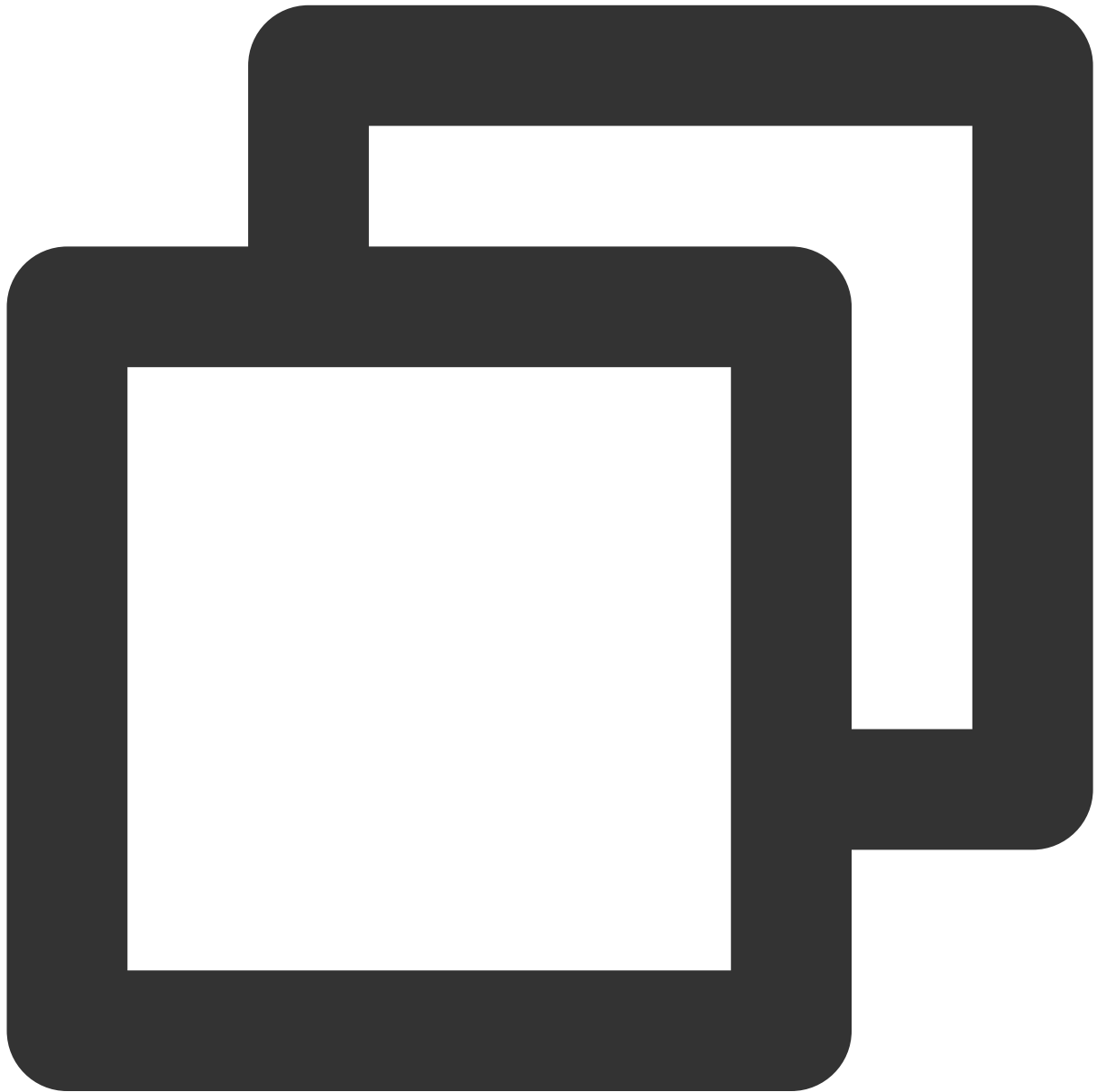
### Note:

Offline push sound settings (only effective for iOS), to customize iOSSound, you first need to link the audio file into the Xcode project, then set the audio file name (with extension) to iOSSound.

Ringtone duration should be less than 30s.

Objective-C

Swift



```
[[TUICallKit sharedInstance] call:@"mike's id" params:[self getCallParams] callMedia:  
- (TUICallParams *)getCallParams {
```

```
TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
TUICallParams *callParams = [TUICallParams new];
callParams.offlinePushInfo = offlinePushInfo;
callParams.timeout = 30;
return callParams;
}

+ (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = TUICallingLocalize(@"TUICallKit.have.new.invitation");
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeAPNs;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // For OPPO, you must set the `ChannelID` to receive push messages. The `Channe
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChanne
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery r
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/developmen
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}
```



```
let params = TUICallParams()
let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = "TUICallKit"
pushInfo.desc = "TUICallKit.have.new.invitation"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// For OPPO, you must set the `ChannelID` to receive push messages. The `ChannelID`
// OPPO must set a ChannelID to receive push messages. This channelID needs to be t
pushInfo.androidOPPOChannelID = "tuikit"
```

```
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HM
pushInfo.androidHuaWeiCategory = "IM"
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo
TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: p

} fail: {
    code, message in
}
```

# Web&H5

Last updated : 2024-04-03 17:23:11

This article introduces how to use the custom ringtone and silent incoming call ringtone feature from the definition.

## Setting incoming call ringtone

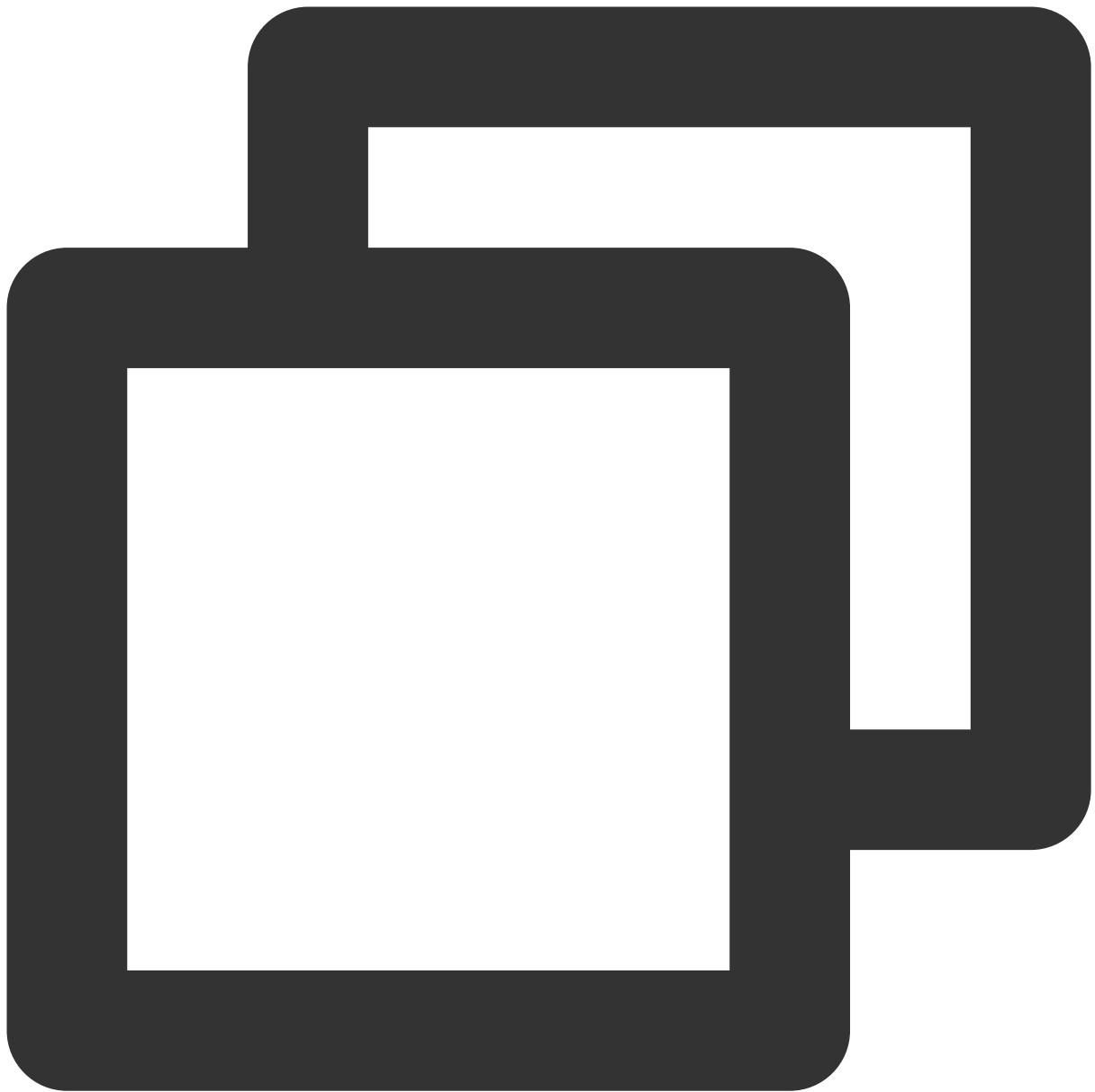
Only local MP3 format file addresses can be used, ensuring that the file is accessible.

To reset the ringtone, pass in an empty string for filePath.

Use the ES6 import method to import the ringtone file.

**Note:**Vue  $\geq$  v3.0.0 supported





```
import filePath from '../public/ring.mp3';

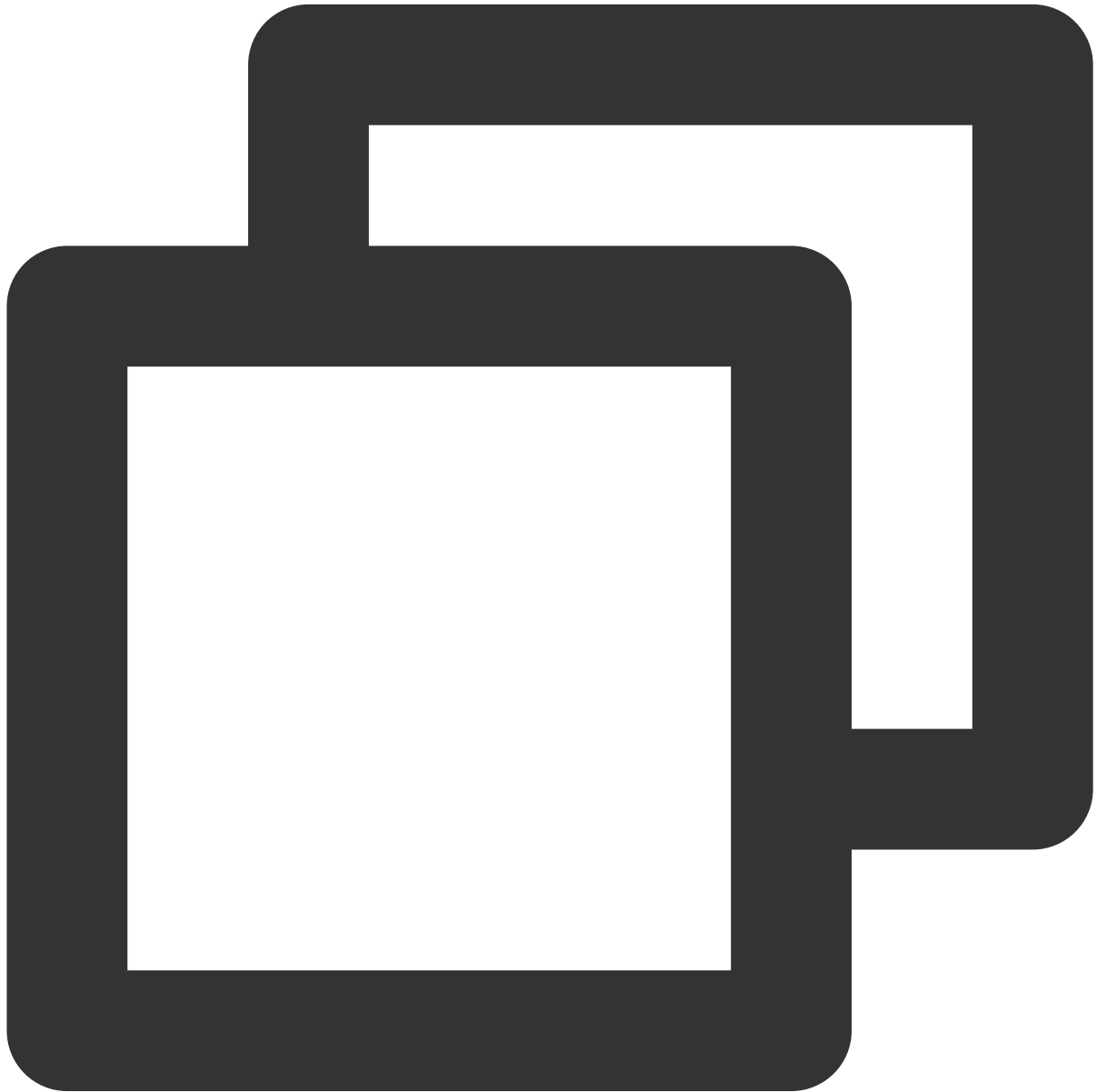
try {
  await TUICallKitServer.setCallingBell(filePath?: string);
} catch (error: any) {
  alert(`[TUICallKit] setCallingBell API failed. Reason: ${error}`);
}
```

## Silent incoming call ringtone

Enable/Disable incoming call ringtone.

After enabling, the incoming call ringtone will not be played when a call request is received.

**Note:**Vue ≥ v3.1.2 supported



```
try {  
  await TUICallKitServer.enableMuteMode(enable: boolean);  
} catch (error: any) {  
  alert(`[TUICallKit] enableMuteMode API failed. Reason: ${error}`);  
}
```

```
}
```

# uni-app (Anroid&iOS)

Last updated : 2024-04-03 17:23:11

This article introduces how to use the custom ringtone and silent incoming call ringtone feature from the definition.

## Customize Incoming Call Ringtone

Setting Custom incoming call ringtone, here **only local file addresses can be passed in, it is required to ensure the file directory is accessible by the application.**



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');

const tempFilePath = './static/rain.mp3'; // Locally stored audio files
let musicFilePath = '';
uni.saveFile({
  tempFilePath: tempFilePath,
  success: (res) => {
    console.warn(JSON.stringify(res));
    musicFilePath = res.savedFilePath;

    musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath);
```

```
// Set ringtone
TUICallKit.setCallingBell(musicFilePath, (res) => {
  if (res.code === 0) {
    console.log('setCallingBell success');
  } else {
    console.log(`setCallingBell failed, error message = ${res.msg}`);
  }
});
},
fail: (err) => {
  console.error(err);
},
});
```

## Silent incoming call ringtone

Enable/Disable incoming call ringtone.

After enabling, the incoming call ringtone will not be played when a call request is received.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');  
const enable = true;  
TUICallKit.enableMuteMode(enable);
```

# Flutter

Last updated : 2024-04-03 17:23:11

This article explains how to replace the incoming call ringtone of TUICallKit, which is divided into **application ringtone** and **offline push ringtone**.

## Setting application ringtone

There are two ways to set an application ringtone: replace the ringtone audio or call the Setting ringtone interface.

### 1. Replace Audio File

If you integrate the TUICallKit component via source code dependency, you can replace the three audio files in the **assets\audios** folder to achieve the purpose of ringtone replacement:

| File Name         | Use                                    |
|-------------------|--|
| phone_dialing.mp3 | Ringtone when initiating a call        |
| phone_hangup.mp3  | Ringtone when the call is disconnected |
| phone_ringing.mp3 | Ringtone when receiving a call         |

### 2. Set Ringtone Interface

You can also set the incoming call ringtone through the setCallingBell interface.





```
TUICallKit.instance.setCallingBell('flie path');
```

### 3. Setting Silent Mode

If you do not need ringing, you can enable the silent mode through `enableMuteMode`.



```
TUICallKit.instance.enableMuteMode(true);
```

## Setting Offline Push Ringtone

### 1. iOS

Voip push does not support custom Definition push ringtones. APNs push allows modifying the parameters `call` and `groupcall` in the interface `params` including `TUIOfflinePushInfo.ioSSound` Setting on the

`ios` platform for offline message ringtones.

## 2. Android

### Note:

The interface supports Huawei, Xiaomi, FCM, and APNs.

FCM's push ringtone is set as the application ringtone.

For Huawei, Xiaomi, and APNs push ringtone settings, please set the `TUIOfflinePushInfo.iOSound` 's `iosound` and `androidSound` fields when calling `Call` and `GroupCall`.

# Monitoring Call Status

## Android&iOS&Flutter

Last updated : 2024-04-03 17:23:11

This article describes how to use call status callbacks with the TUICallKit component.

## Call State Monitoring

If your business requires **monitoring call status**, such as the start and end of a call, and other events during the call, you can refer to the following code:

Android(Kotlin)

Android(Java)

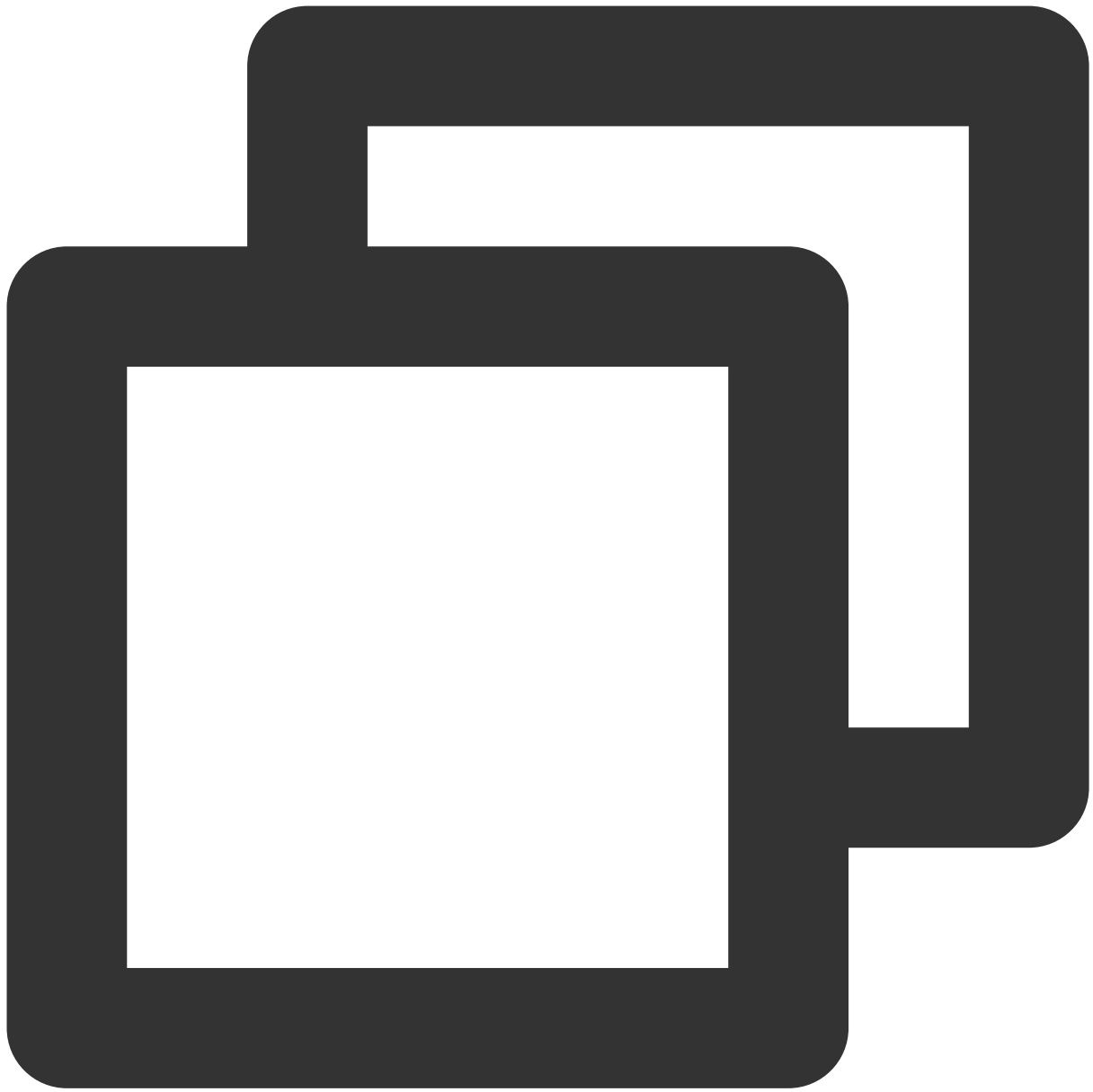
iOS(Swift)

iOS(Objective-C)

Flutter(Dart)

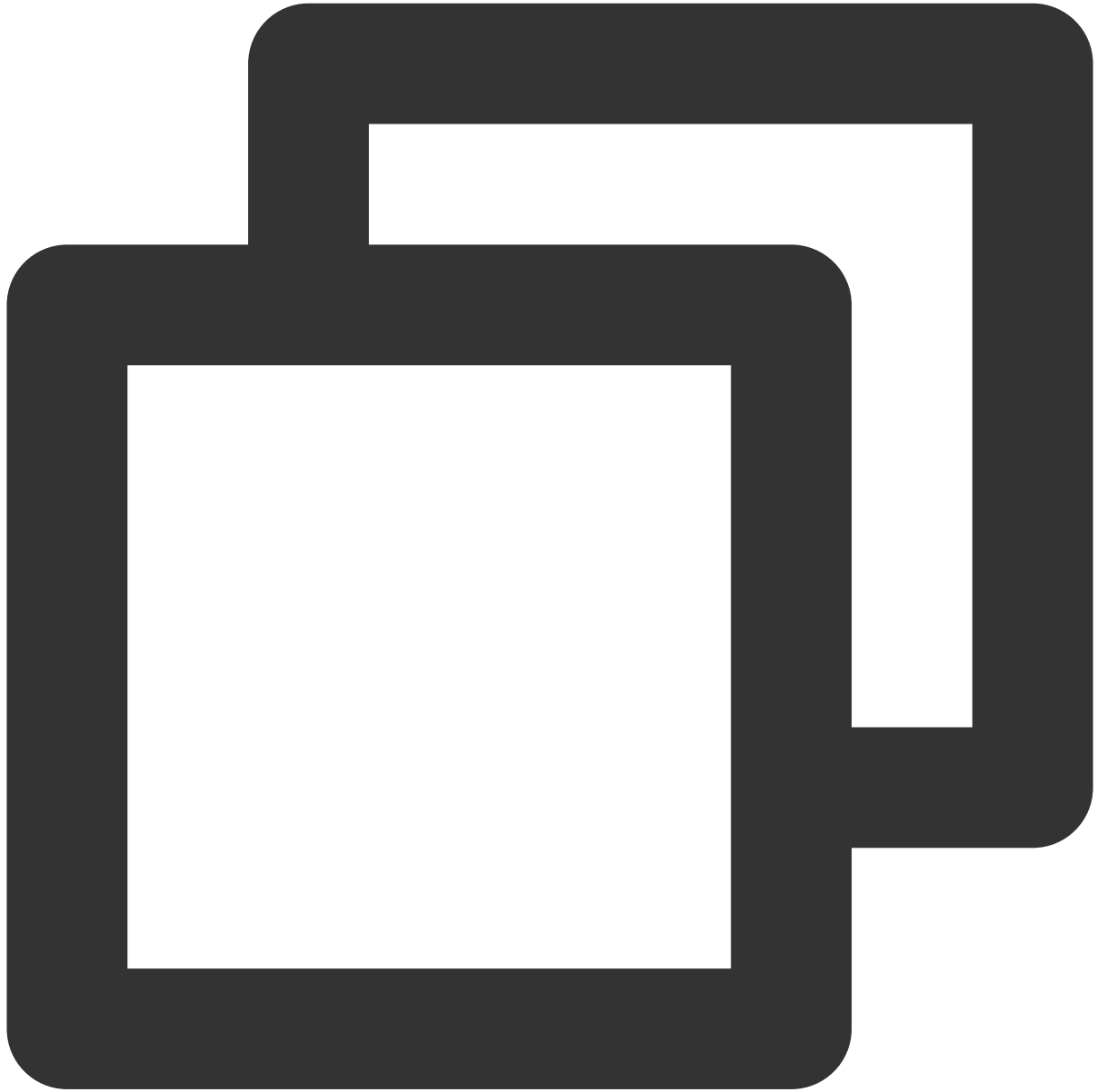


```
private val observer: TUICallObserver = object : TUICallObserver() {  
    override fun onCallBegin(roomId: TUICommonDefine.RoomId?, callMediaType: TUICal  
    }  
    override fun onCallEnd(roomId: TUICommonDefine.RoomId?, callMediaType: TUICalld  
    }  
    override fun onUserNetworkQualityChanged(networkQualityList: MutableList<TUICom  
    }  
}  
private fun initData() {  
    TUICallEngine.createInstance(context).addObserver(observer)  
}
```



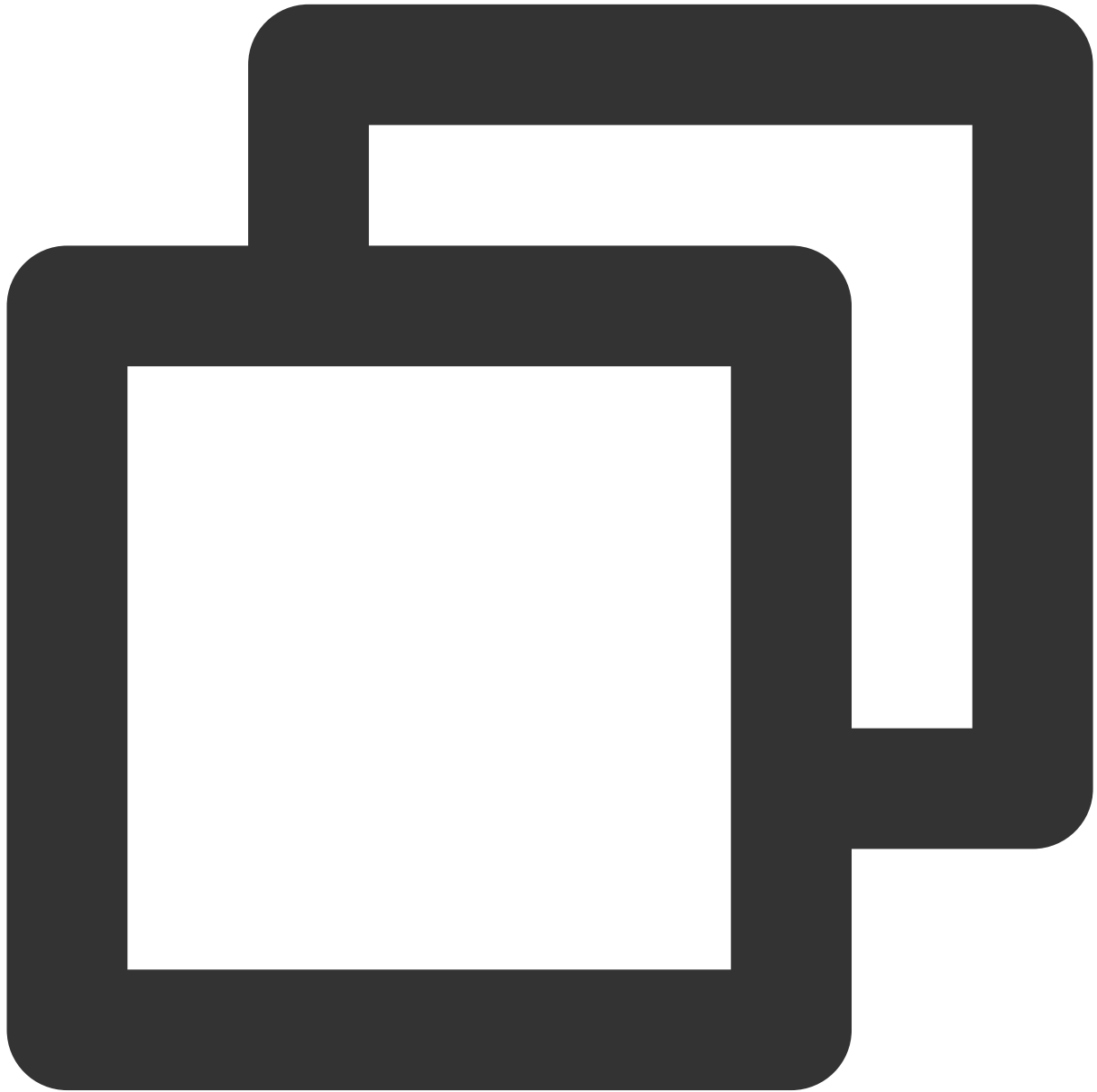
```
private TUICallObserver observer = new TUICallObserver() {  
    @Override  
    public void onCallBegin(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType ca  
    }  
    public void onCallEnd(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType ca  
    }  
    public void onUserNetworkQualityChanged(List<TUICommonDefine.NetworkQualityInfo  
    }  
};
```

```
private void initData(){  
    TUICallEngine.createInstance(context).addObserver(observer);  
}
```



```
import TUICallEngine  
  
TUICallEngine.createInstance().addObserver(self)  
  
public func onCallBegin(roomId: TUIRoomId, callMediaType: TUICallMediaType, callRol  
  
}
```

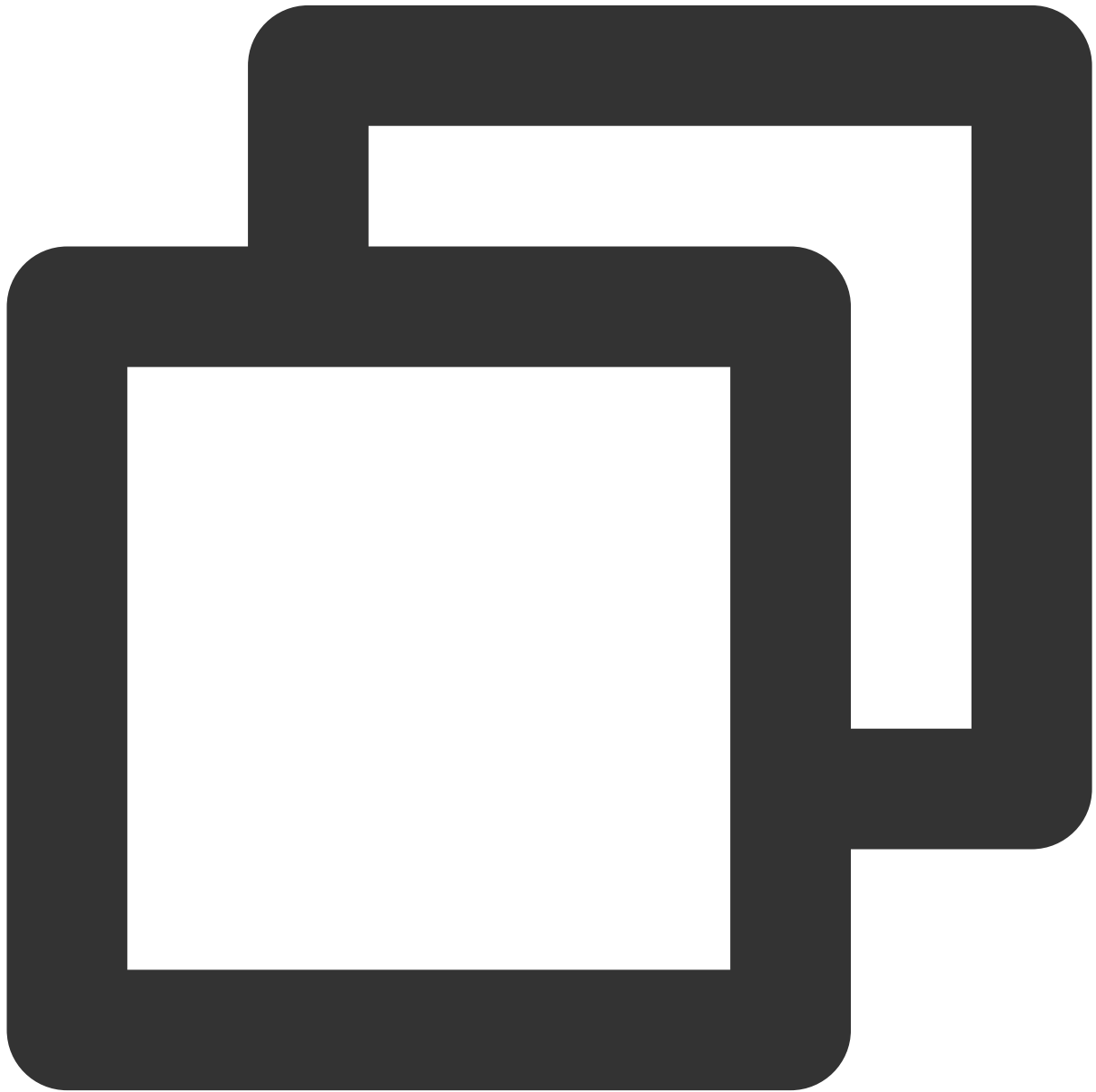
```
public func onCallEnd(roomId: TUIRoomId, callMediaType: TUICallMediaType, callRole:  
    }  
public func onUserNetworkQualityChanged(networkQualityList: [TUINetworkQualityInfo]  
    }
```



```
#import <TUICallEngine/TUICallEngine.h>  
  
[[TUICallEngine sharedInstance] addObserver:self];
```



```
- (void)onCallBegin:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaTy  
}  
- (void)onCallEnd:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaType  
}  
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkQualityInfo *> *)networkQual  
}
```



```
TUICallObserver observer = TUICallObserver(
```

```
onError: (int code, String message) {
    // Your callback handling logic
}, onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole role,
    // Your callback handling logic
}, onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole role,
    // Your callback handling logic
),, onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList,
    // Your callback handling logic
}, onCallReceived: (String callerId, List<String> calleeIdList, String groupId,
    // Your callback handling logic
}
...
))

TUICallEngine.instance.addObserver(observer);
```

**Note:**

On the Android platform, when setting `TUICallObserver` to listen for callbacks, ensure that the class hosting the callback will not be cleared. For example, it is not recommended to add a listener in `LoginActivity`, as when `LoginActivity` is destroyed, the callback will also be cleared; it is suggested to listen in the application's `Application` class or the main application interface.

# Web&H5

Last updated : 2024-04-03 17:23:11

This article describes how to use call status callbacks with the TUICallKit component.

## Call State Monitoring

If your business requires **monitoring the call status**, such as events during the call process ( [TUICallEvent](#) for details), you can refer to the following code:



```
import { TUICallEvent } from 'tuicall-engine-webrtc';

let handleUserEnter = function(event) {
  console.log('TUICallEvent.USER_ENTER: ', event);
};

TUICallKitServer.getTUICallEngineInstance().on(TUICallEvent.USER_ENTER, handleUserEnter);
TUICallKitServer.getTUICallEngineInstance().off(TUICallEvent.USER_ENTER, handleUserEnter);
```

## Component Callback Event

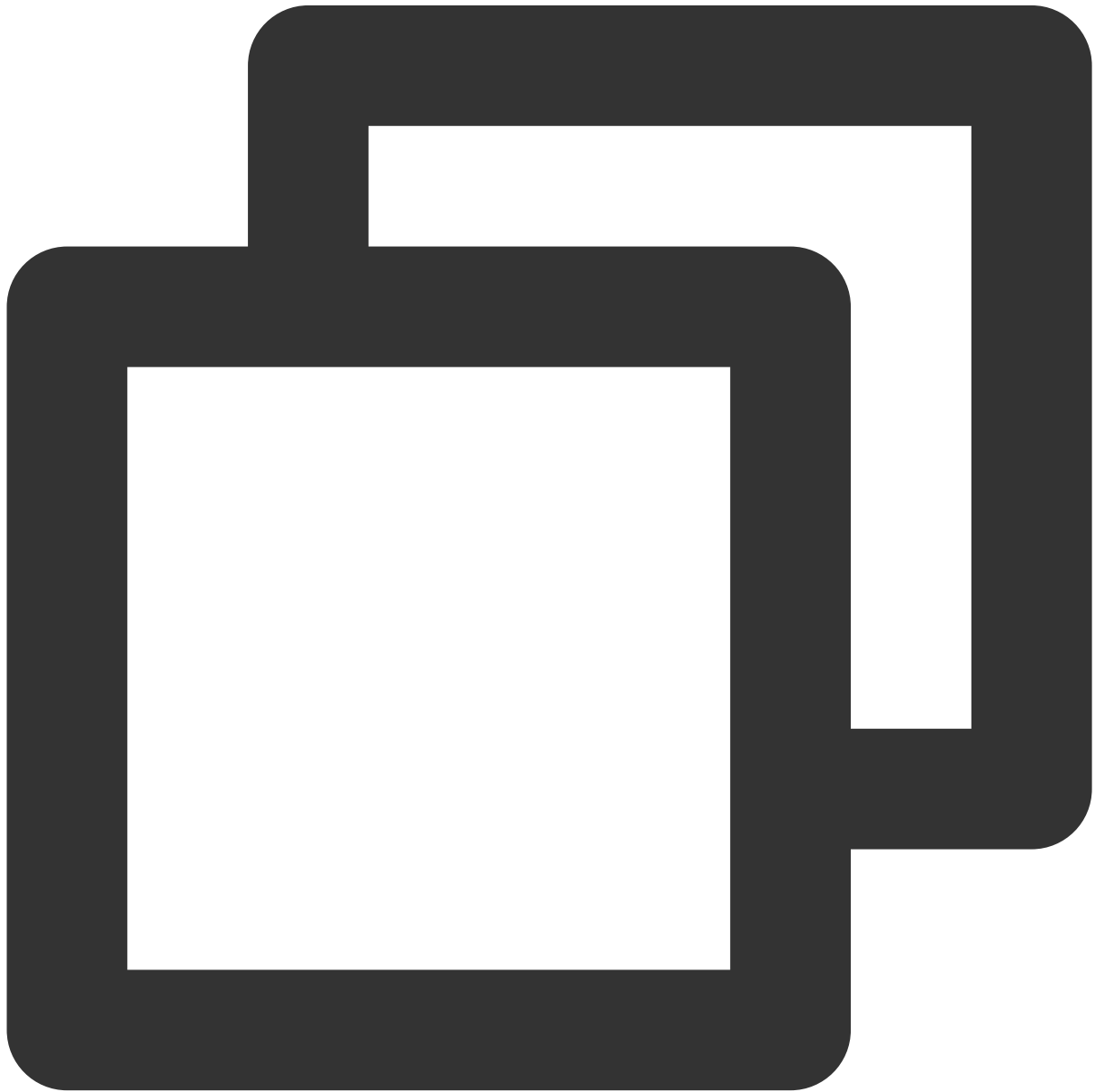
The TUICallKit component offers call status callbacks, which can be used to implement more interaction logic on the business side. Please refer to the [TUICallKit Attribute Overview](#) for more details.

`beforeCalling` : Executed prior to the commencement of the call.

`afterCalling` : Executed upon the completion of the call.

React

Vue



```
function App() {
  const handleBeforeCalling = () => {
    console.log("[TUICallKit Demo] beforeCalling");
  };
  const handleAfterCalling = () => {
    console.log("[TUICallKit Demo] afterCalling");
  };
  return (
    <TUICallKit
      beforeCalling={handleBeforeCalling}
      afterCalling={handleAfterCalling} />
  )
}
```



```
<template>
  <TUICallKit
    :beforeCalling="handleBeforeCalling"
    :afterCalling="handleAfterCalling" />
</template>
<script setup>
function handleBeforeCalling() {
  console.log("[TUICallKit Demo] beforeCalling");
}
function handleAfterCalling() {
  console.log("[TUICallKit Demo] afterCalling");
}
```

```
}  
</script>
```



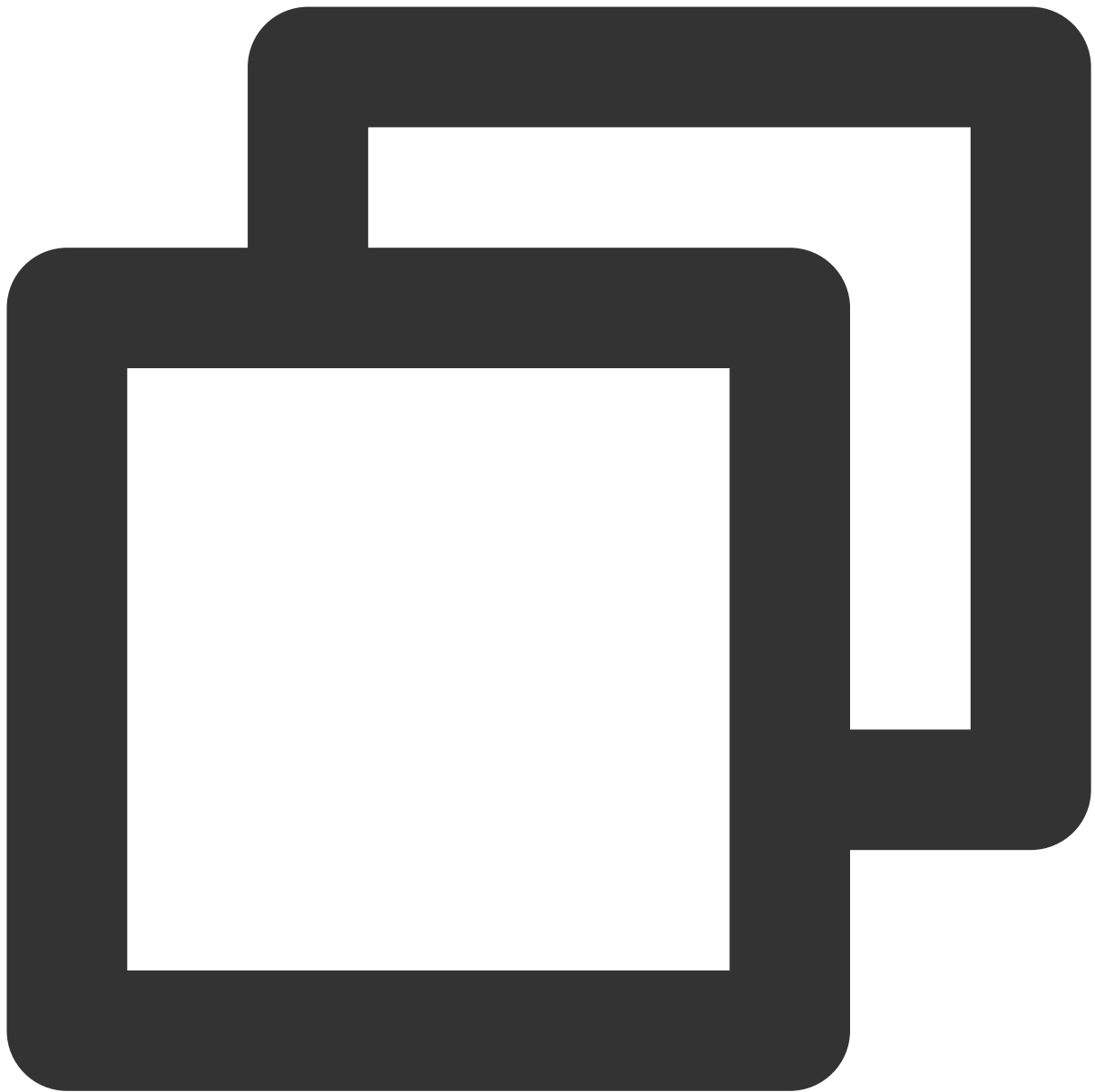
# uni-app (Android & iOS)

Last updated : 2024-04-03 17:23:11

This article describes how to use call status callbacks with the TUICallKit component.

## Call State Monitoring

If your business requires **monitoring call status**, such as events that occur during a call like starting and ending (see [Events](#) for details), please refer to the following code.



```
const TUICallKitEvent = uni.requireNativePlugin('globalEvent');

function handleError(res) {
  console.log('onError', JSON.stringify(res));
}
TUICallKitEvent.addEventListener('onError', handleError);
TUICallKitEvent.removeEventListener('onError', handleError);
```

# Client APIs (TUICallKit)

## Android

## API Overview

Last updated : 2023-09-19 15:19:59

### TUICallKit (UI Included)

TUICallKit is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat.

API	Description
<a href="#">createInstance</a>	Creates a TUICallKit instance (singleton mode).
<a href="#">setSelfInfo</a>	Sets the user nickname and profile photo.
<a href="#">call</a>	Makes a one-to-one call.
<a href="#">groupCall</a>	Makes a group call.
<a href="#">joinInGroupCall</a>	Joins a group call.
<a href="#">setCallingBell</a>	Sets the ringtone.
<a href="#">enableMuteMode</a>	Sets whether to turn on the mute mode.
<a href="#">enableFloatWindow</a>	Sets whether to enable floating windows.

### TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description
<a href="#">createInstance</a>	Creates a <code>TUICallEngine</code> instance (singleton).
<a href="#">destroyInstance</a>	Terminates a <code>TUICallEngine</code> instance (singleton).
<a href="#">init</a>	Authenticates the basic audio/video call capabilities.

<code>addObserver</code>	Registers an event listener.
<code>removeObserver</code>	Unregisters an event listener.
<code>call</code>	Makes a one-to-one call.
<code>groupCall</code>	Makes a group call.
<code>accept</code>	Answers a call.
<code>reject</code>	Declines a call.
<code>hangup</code>	Ends a call.
<code>ignore</code>	Ignores a call.
<code>inviteUser</code>	Invites users to the current group call.
<code>joinInGroupCall</code>	Joins a group call.
<code>switchCallMediaType</code>	Switches the call media type, such as from video call to audio call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.
<code>openCamera</code>	Turns the camera on.
<code>closeCamera</code>	Turns the camera off.
<code>switchCamera</code>	Switches the camera.
<code>openMicrophone</code>	Enables the mic.
<code>closeMicrophone</code>	Disables the mic.
<code>selectAudioPlaybackDevice</code>	Selects the audio playback device (receiver/speaker).
<code>setSelfInfo</code>	Sets the user nickname and profile photo.
<code>enableMultiDeviceAbility</code>	Sets whether to enable multi-device login for <code>TUICallEngine</code> (supported by the premium package).
<code>setVideoRenderParams</code>	Set the rendering mode of video image.
<code>setVideoEncoderParams</code>	Set the encoding parameters of video encoder.
<code>getTRTCCloudInstance</code>	Advanced features.
<code>setBeautyLevel</code>	Set beauty level, support turning off default beauty.

## TUICallObserver

`TUICallObserver` is the callback class of `TUICallEngine`. You can use it to listen for events.

API	Description
<code>onError</code>	An error occurred during the call.
<code>onCallReceived</code>	A call was received.
<code>onCallCancelled</code>	The call was canceled.
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user has a video stream.
<code>onUserAudioAvailable</code>	Whether a user has an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user was kicked offline.
<code>onUserSigExpired</code>	The user sig is expired.

## Definitions of Key Types

API	Description

<a href="#">TUICallDefine.MediaType</a>	The call type. Enumeration: Video call and audio call.
<a href="#">TUICallDefine.Role</a>	The call role. Enumeration: Caller and callee.
<a href="#">TUICallDefine.Status</a>	The call status. Enumeration: Idle, waiting, and answering.
<a href="#">TUICommonDefine.RoomId</a>	The room ID, which can be a number or string.
<a href="#">TUICommonDefine.Camera</a>	The camera type. Enumeration: Front camera and rear camera.
<a href="#">TUICommonDefine.AudioPlaybackDevice</a>	The audio playback device type. Enumeration: Speaker and receiver.
<a href="#">TUICommonDefine.NetworkQualityInfo</a>	The current network quality.

# TUICallKit

Last updated : 2023-09-19 15:27:34

## TUICallKit APIs

`TUICallKit` is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat. For directions on integration, see [Integrating TUICallKit](#).

## API Overview

API	Description
<a href="#">createInstance</a>	Creates a <code>TUICallKit</code> instance (singleton mode).
<a href="#">setSelfInfo</a>	Sets the alias and profile photo.
<a href="#">call</a>	Makes a one-to-one call.
<a href="#">call</a>	Makes a one-to-one call, Support for custom room ID, call timeout, offline push content, etc
<a href="#">groupCall</a>	Makes a group call.
<a href="#">groupCall</a>	Makes a group call, Support for custom room ID, call timeout, offline push content, etc
<a href="#">joinInGroupCall</a>	Joins a group call.
<a href="#">setCallingBell</a>	Sets the ringtone.
<a href="#">enableMuteMode</a>	Sets whether to turn on the mute mode.
<a href="#">enableFloatWindow</a>	Sets whether to enable floating windows.

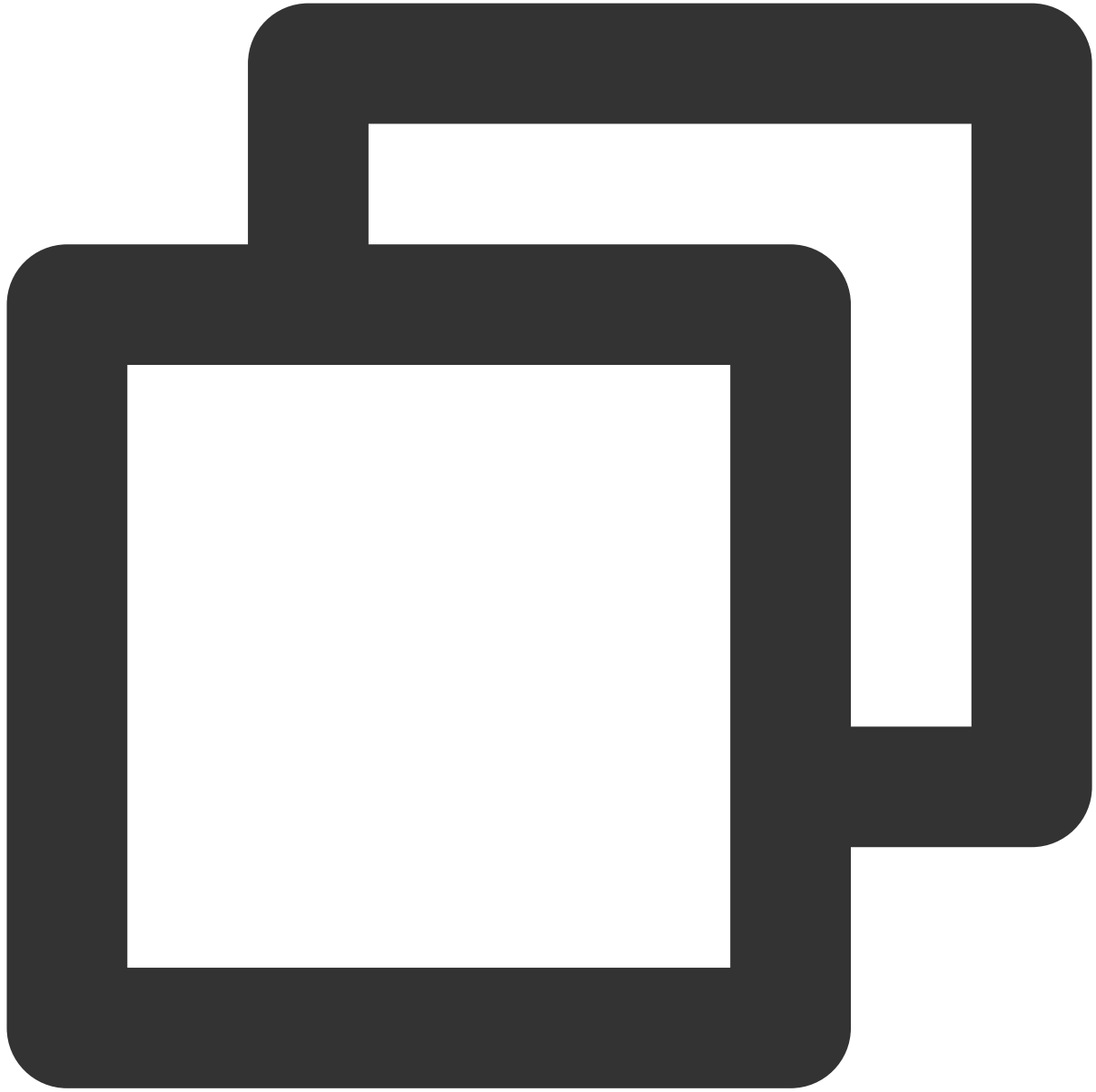
## Details

### createInstance

This API is used to create a `TUICallKit` singleton.

Kotlin

Java



```
fun createInstance(context: Context): TUICallKit
```





```
TUICallKit createInstance(Context context)
```

### setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.

Kotlin

Java



```
fun setSelfInfo(nickname: String?, avatar: String?, callback: TUICommonDefine.Callb
```



```
void setSelfInfo(String nickname, String avatar, TUICommonDefine.Callback callback)
```

The parameters are described below:

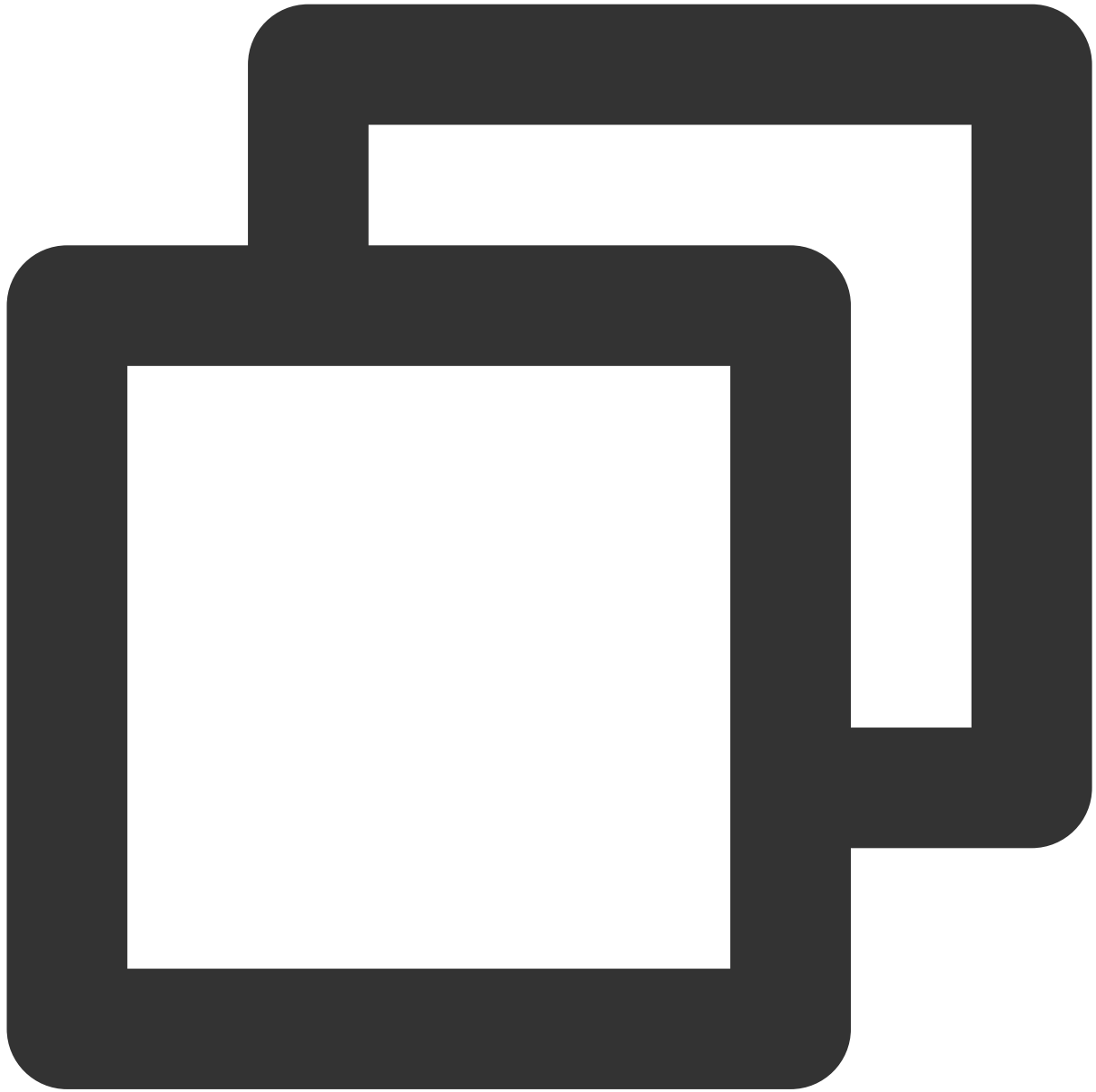
Parameter	Type	Description
nickname	String	The alias.
avatar	String	The profile photo.

## call

This API is used to make a (one-to-one) call.

Kotlin

Java



```
fun call(userId: String, callMediaType: TUICallDefine.MediaType)
```



```
void call(String userId, TUICallDefine.MediaType callMediaType)
```

The parameters are described below:

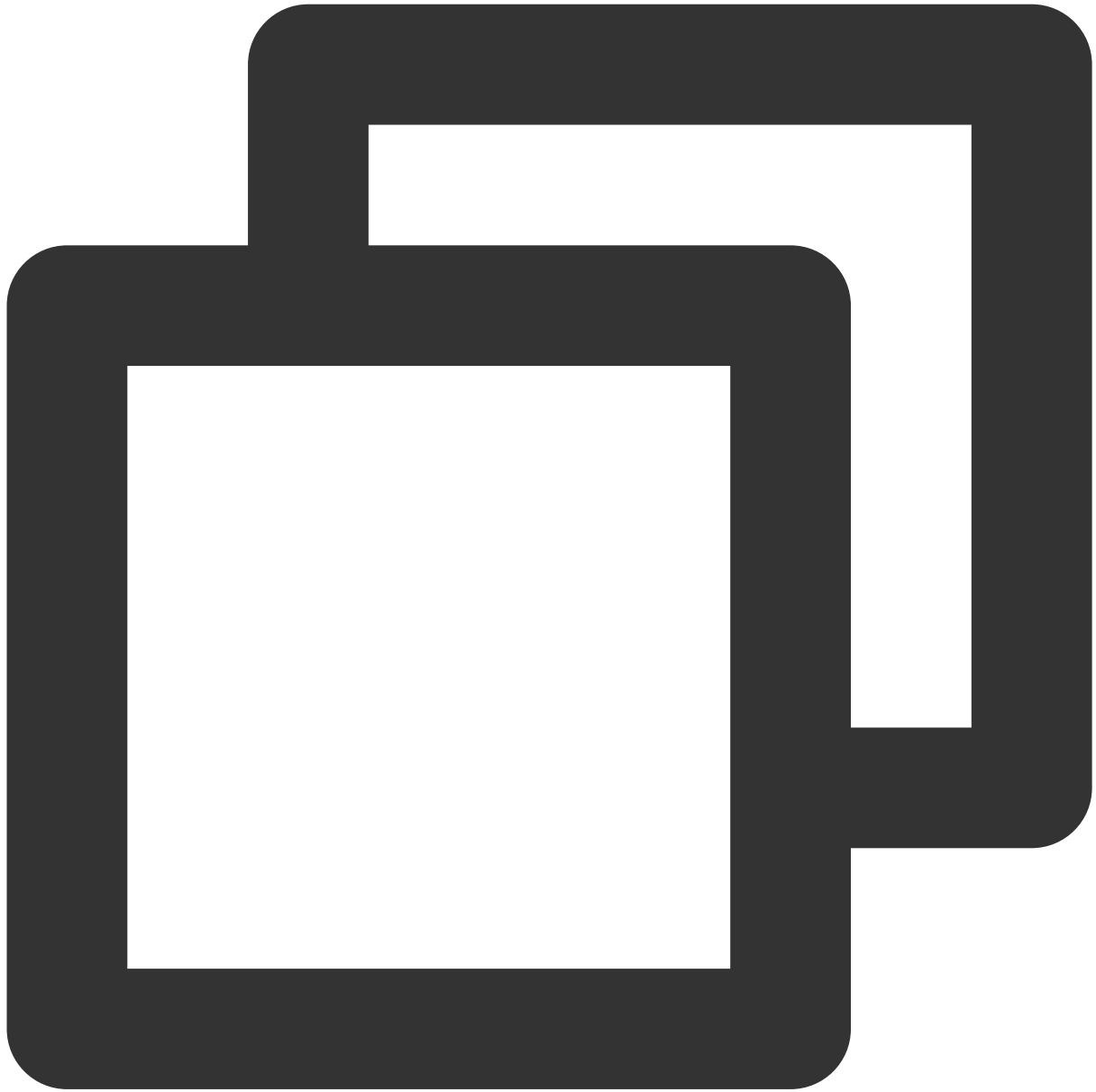
Parameter	Type	Description
userId	String	The target user ID.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.

## call

This API is used to make a (one-to-one) call, Support for custom room ID, call timeout, offline push content, etc.

Kotlin

Java



```
fun call(  
    userId: String, callMediaType: TUICallDefine.MediaType,  
    params: CallParams?, callback: TUICommonDefine.Callback?  
)
```



```
void call(String userId, TUICallDefine.MediaType callMediaType,  
          TUICallDefine.CallParams params, TUICommonDefine.Callback callback)
```

The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.

params	<a href="#">TUICallDefine.CallParams</a>	Call extension parameters, such as roomId, call timeout, offline push info,etc
--------	--	--

## groupCall

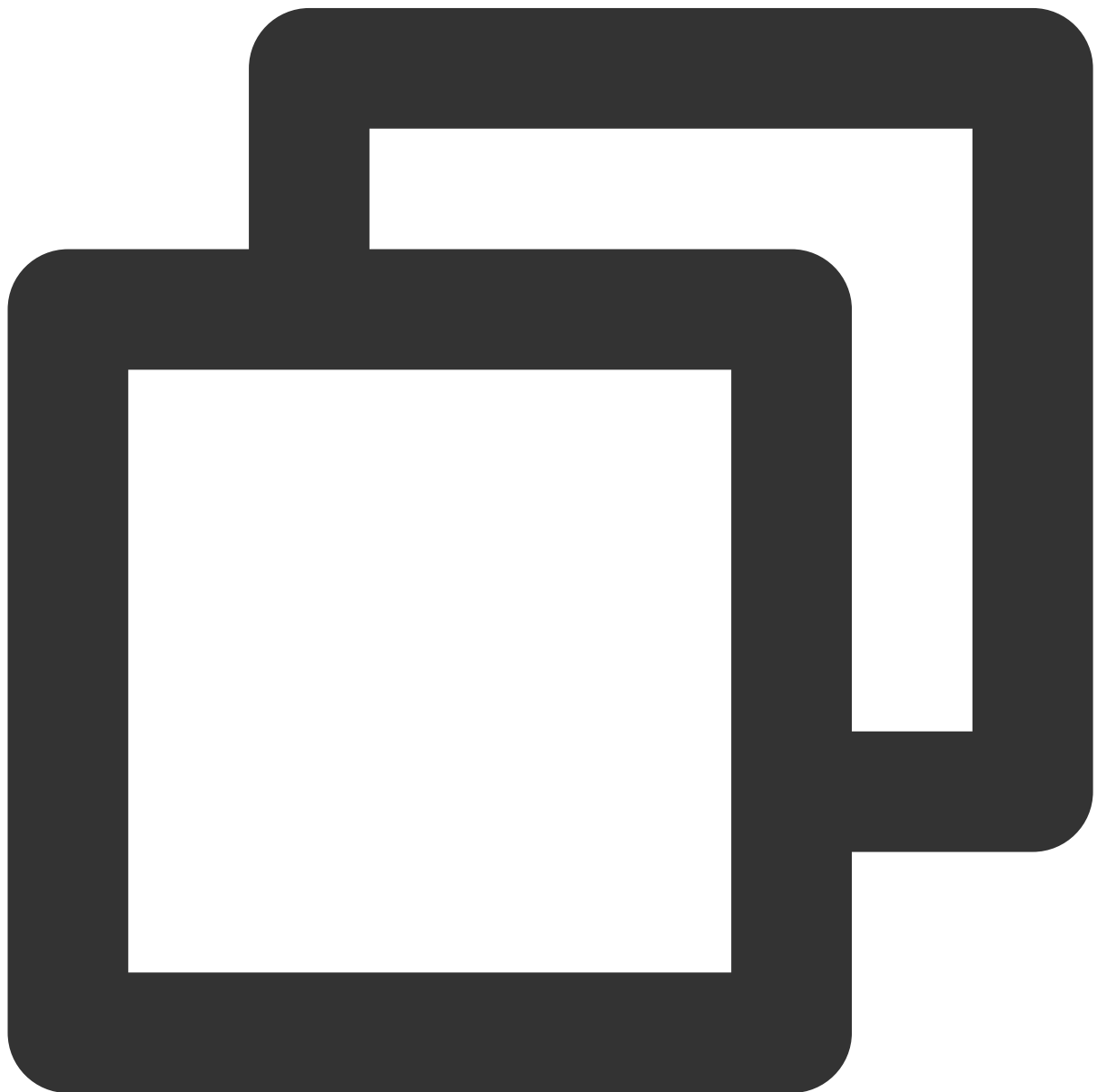
This API is used to make a group call.

### Notice:

Before making a group call, you need to create an IM group first.

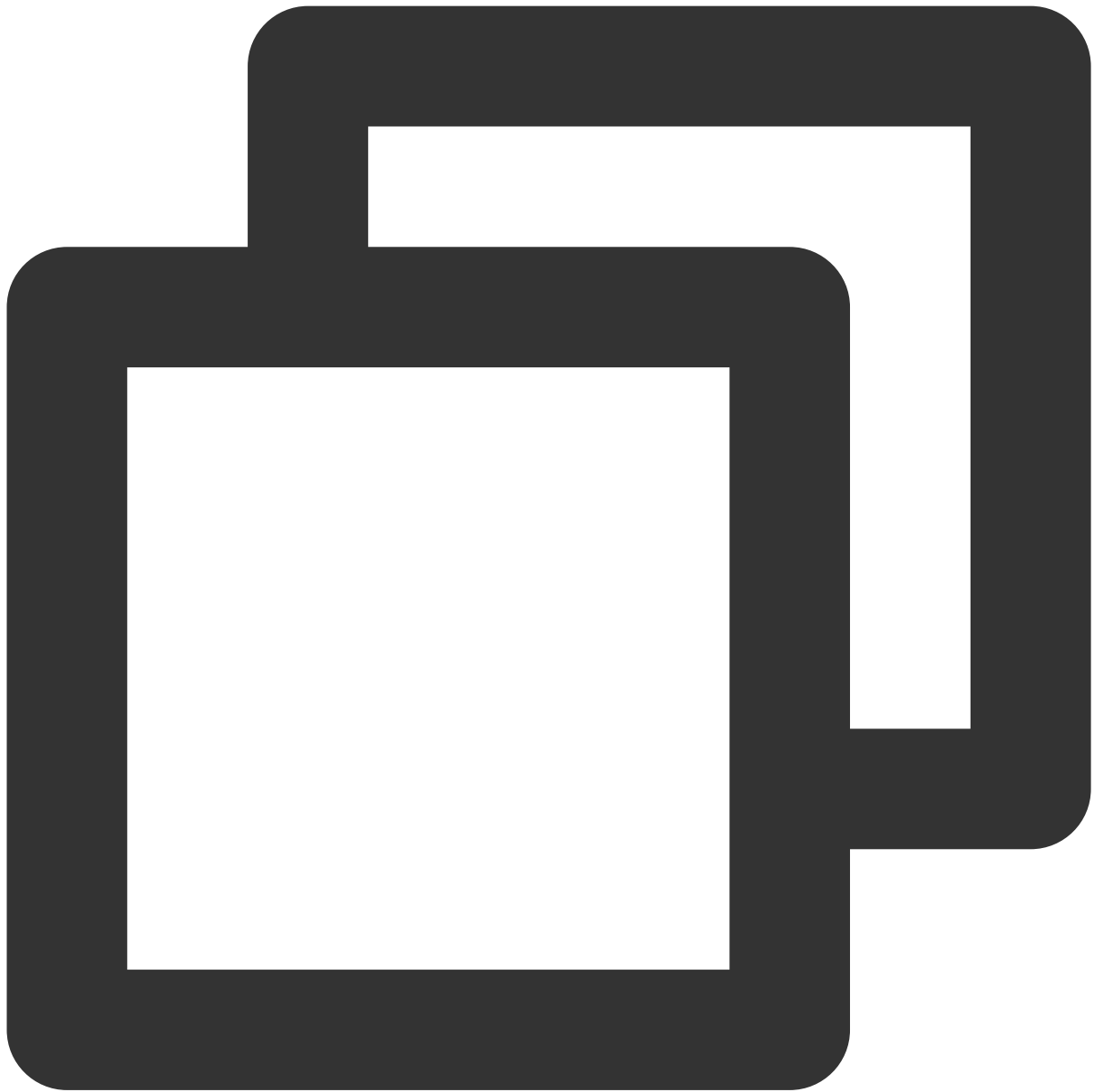
Kotlin

Java





```
fun groupCall(groupId: String, userIdList: List<String?>?, callMediaType: TUICallDe
```



```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal
```

The parameters are described below:

Parameter	Type	Description
groupId	String	The group ID.
userIdList	List	The target user IDs.

callMediaType

[TUICallDefine.MediaType](#)

The call type, which can be video or audio.

## groupCall

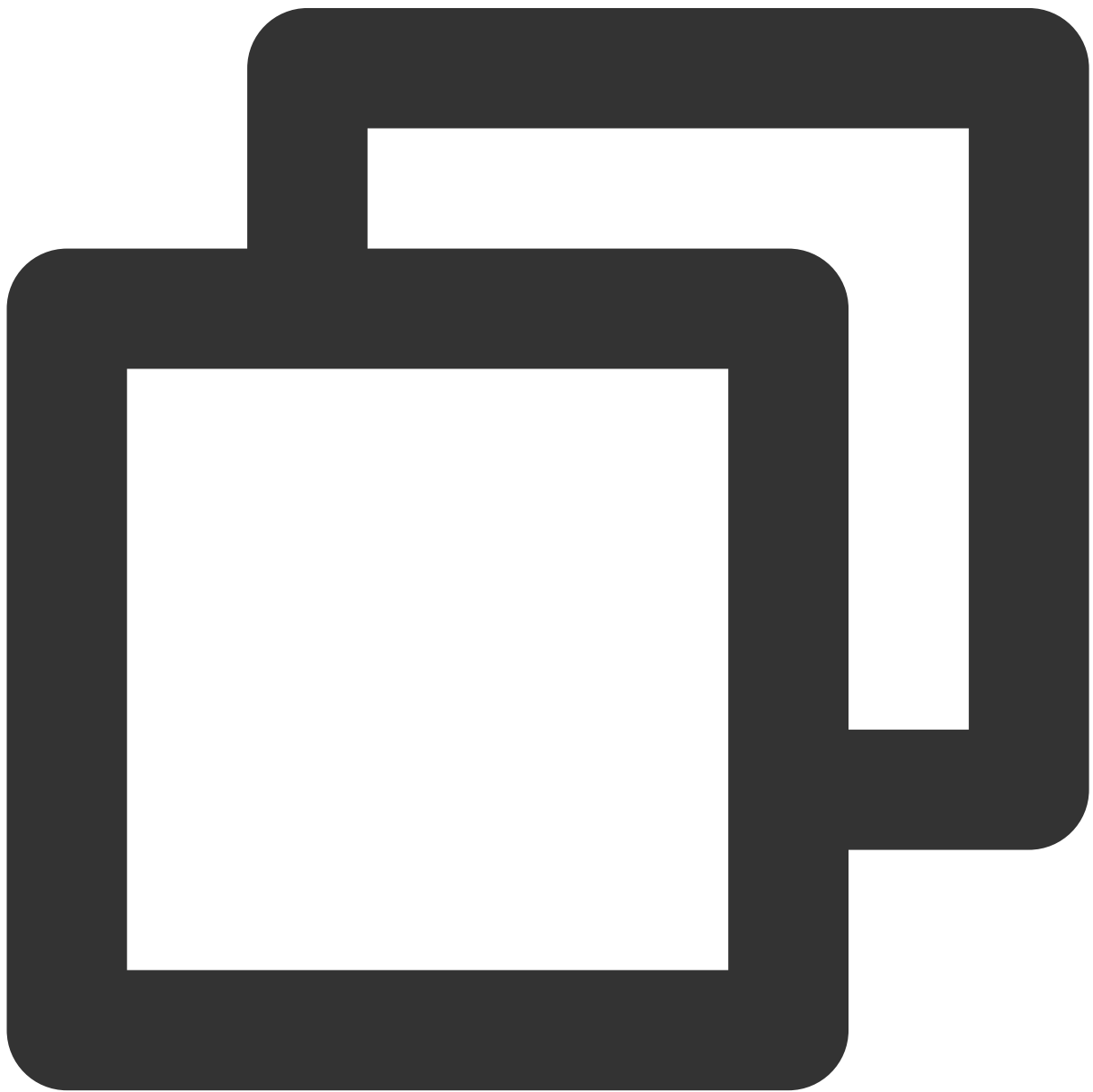
This API is used to make a group call, Support for custom room ID, call timeout, offline push content, etc.

### Notice:

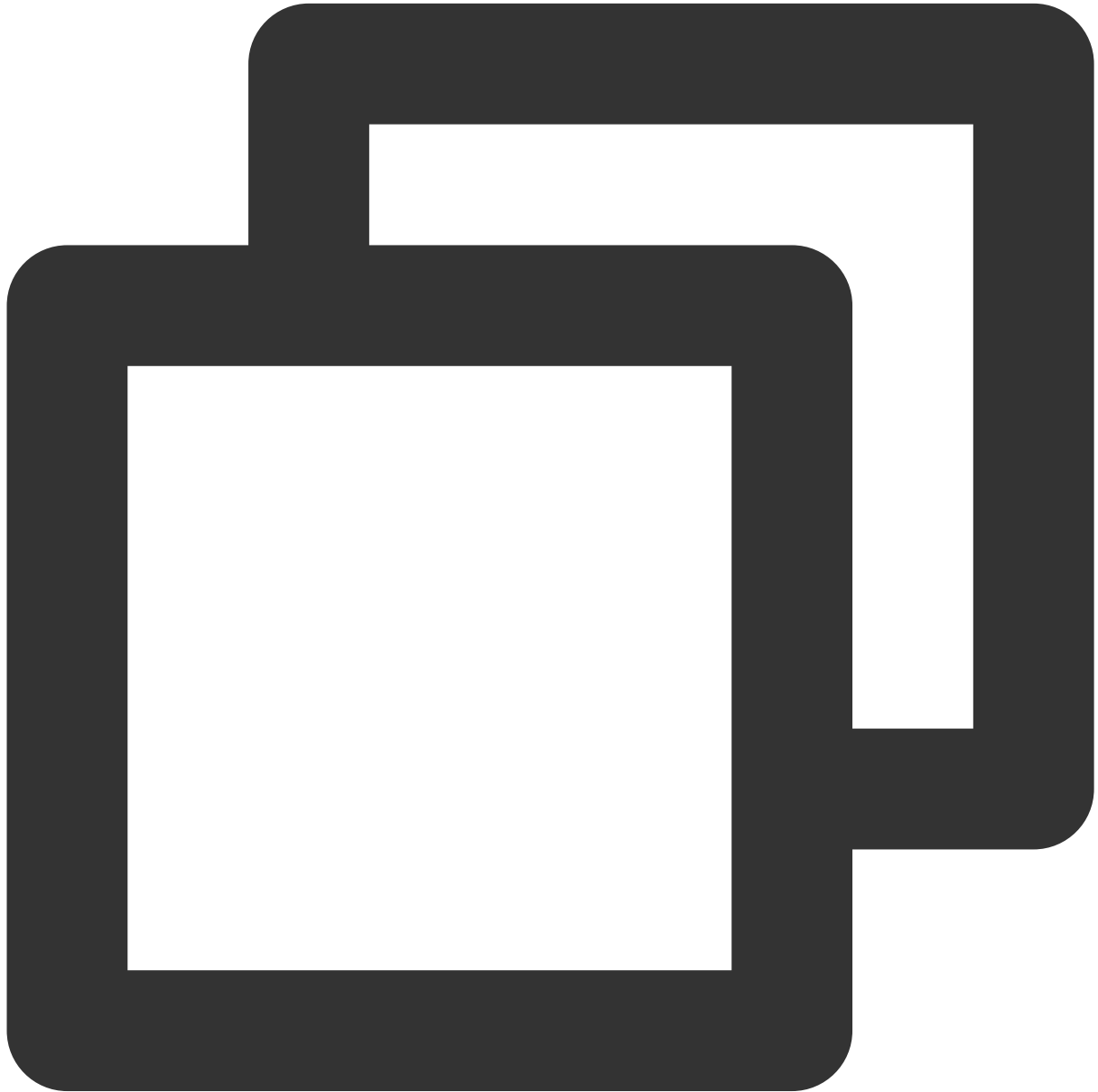
Before making a group call, you need to create an IM group first.

Kotlin

Java



```
fun groupCall(  
    groupId: String, userIdList: List<String?>?,  
    callMediaType: TUICallDefine.MediaType, params: CallParams?,  
    callback: TUICCommonDefine.Callback?  
)
```



```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal  
    TUICallDefine.CallParams params, TUICCommonDefine.Callback callback);
```

The parameters are described below:

Parameter	Type	Description
groupId	String	The group ID.
userIdList	List	The target user IDs.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.
params	<a href="#">TUICallDefine.CallParams</a>	Call extension parameters, such as roomId, call timeout, offline push info,etc

## joinInGroupCall

This API is used to join a group call.

### Notice:

Before joining a group call, you need to create or join an IM group in advance, and there are already users in the group who are in the call.

Kotlin

Java



```
fun joinInGroupCall(roomId: RoomId?, groupId: String?, callMediaType: TUICallDefine
```



```
void joinInGroupCall(TUICommonDefine.RoomId roomId, String groupId, TUICallDefine.M
```

The parameters are described below:

Parameter	Type	Description
roomId	<a href="#">TUICommonDefine.RoomId</a>	The room ID.
groupId	String	The group ID.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.

## setCallingBell

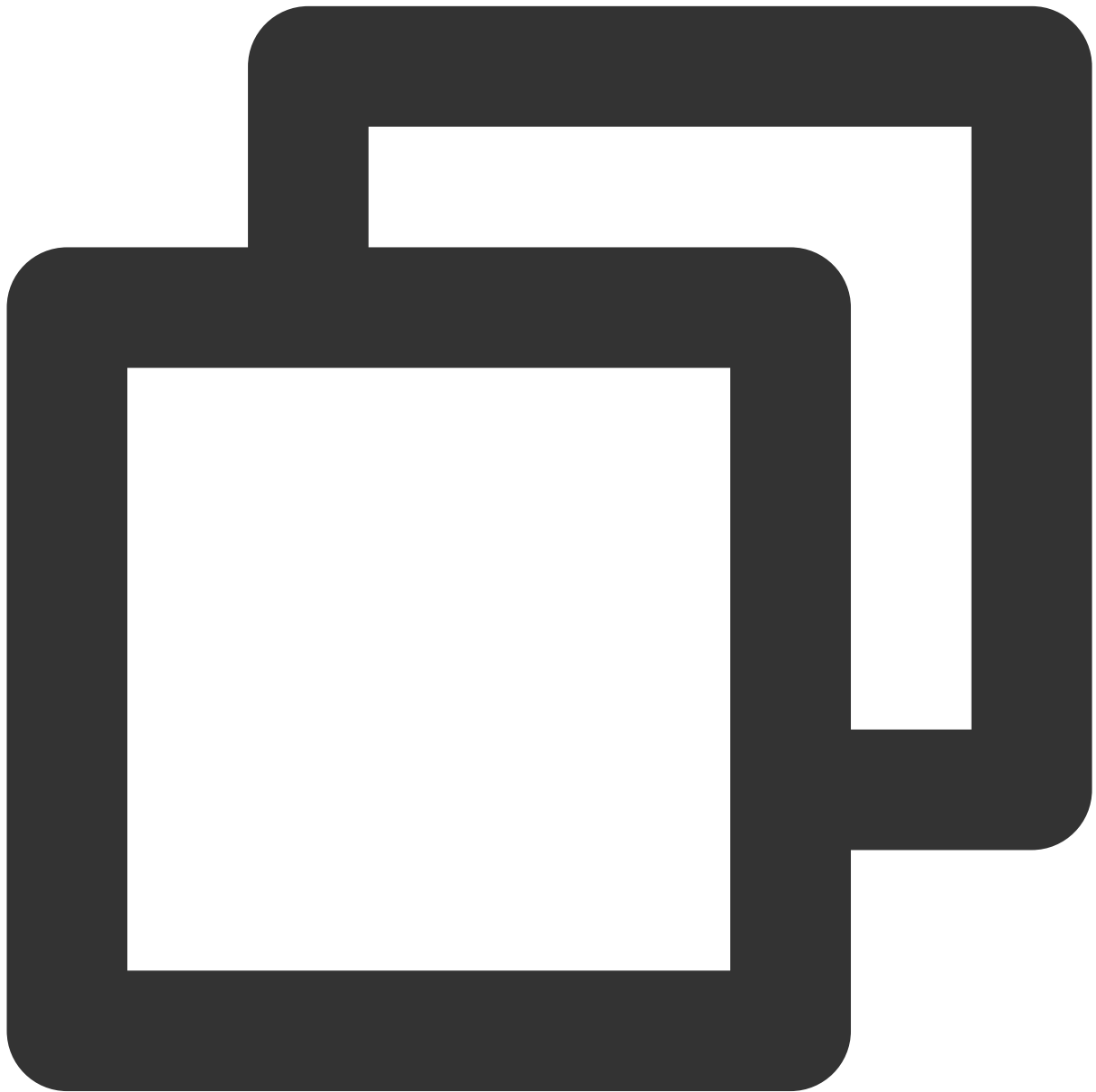
This API is used to set the ringtone. `filePath` must be an accessible local file URL.

The ringtone set is associated with the device and does not change with user.

To reset the ringtone, pass in an empty string for `filePath`.

Kotlin

Java



```
fun setCallingBell(filePath: String?)
```



```
void setCallingBell(String filePath);
```

### **enableMuteMode**

This API is used to set whether to turn on the mute mode.

Kotlin

Java





```
fun enableMuteMode(enable: Boolean)
```



```
void enableMuteMode(boolean enable);
```

## enableFloatWindow

This API is used to set whether to enable floating windows.

The default value is `false`, and the floating window button in the top left corner of the call view is hidden. If it is set to `true`, the button will become visible.

Kotlin

Java



```
fun enableFloatWindow(enable: Boolean)
```



```
void enableFloatWindow(boolean enable);
```

# TUICallEngine

Last updated : 2023-09-19 15:28:14

## TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

## Overview

API	Description
<code>createInstance</code>	Creates a <code>TUICallEngine</code> instance (singleton mode).
<code>destroyInstance</code>	Terminates a <code>TUICallEngine</code> instance (singleton mode).
<code>init</code>	Authenticates the basic audio/video call capabilities.
<code>addObserver</code>	Registers an event listener.
<code>removeObserver</code>	Unregisters an event listener.
<code>call</code>	Makes a one-to-one call.
<code>groupCall</code>	Makes a group call.
<code>accept</code>	Accepts a call.
<code>reject</code>	Rejects a call.
<code>hangup</code>	Ends a call.
<code>ignore</code>	Ignores a call.
<code>inviteUser</code>	Invites users to the current group call.
<code>joinInGroupCall</code>	Joins a group call.
<code>switchCallMediaType</code>	Changes the call type, for example, from video call to audio call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.

<code>openCamera</code>	Turns the camera on.
<code>closeCamera</code>	Turns the camera off.
<code>switchCamera</code>	Switches between the front and rear cameras.
<code>openMicrophone</code>	Turns the mic on.
<code>closeMicrophone</code>	Turns the mic off.
<code>selectAudioPlaybackDevice</code>	Selects the audio playback device (receiver or speaker).
<code>setSelfInfo</code>	Sets the alias and profile photo.
<code>enableMultiDeviceAbility</code>	Sets whether to enable multi-device login for <code>TUICallEngine</code> (supported by the premium package).
<code>setVideoRenderParams</code>	Set the rendering mode of video image.
<code>setVideoEncoderParams</code>	Set the encoding parameters of video encoder.
<code>getTRTCCloudInstance</code>	Advanced features.
<code>setBeautyLevel</code>	Set beauty level, support turning off default beauty.

## Details

### **createInstance**

This API is used to create a `TUICallEngine` singleton.



```
TUICallEngine createInstance(Context context)
```

### **destroyInstance**

This API is used to terminate a `TUICallEngine` singleton.



```
void destroyInstance();
```

## Init

This API is used to initialize `TUICallEngine` . Call it to authenticate the call service and perform other required actions before you call other APIs.





```
void init(int sdkAppId, String userId, String userSig, TUICommonDefine.Callback cal
```

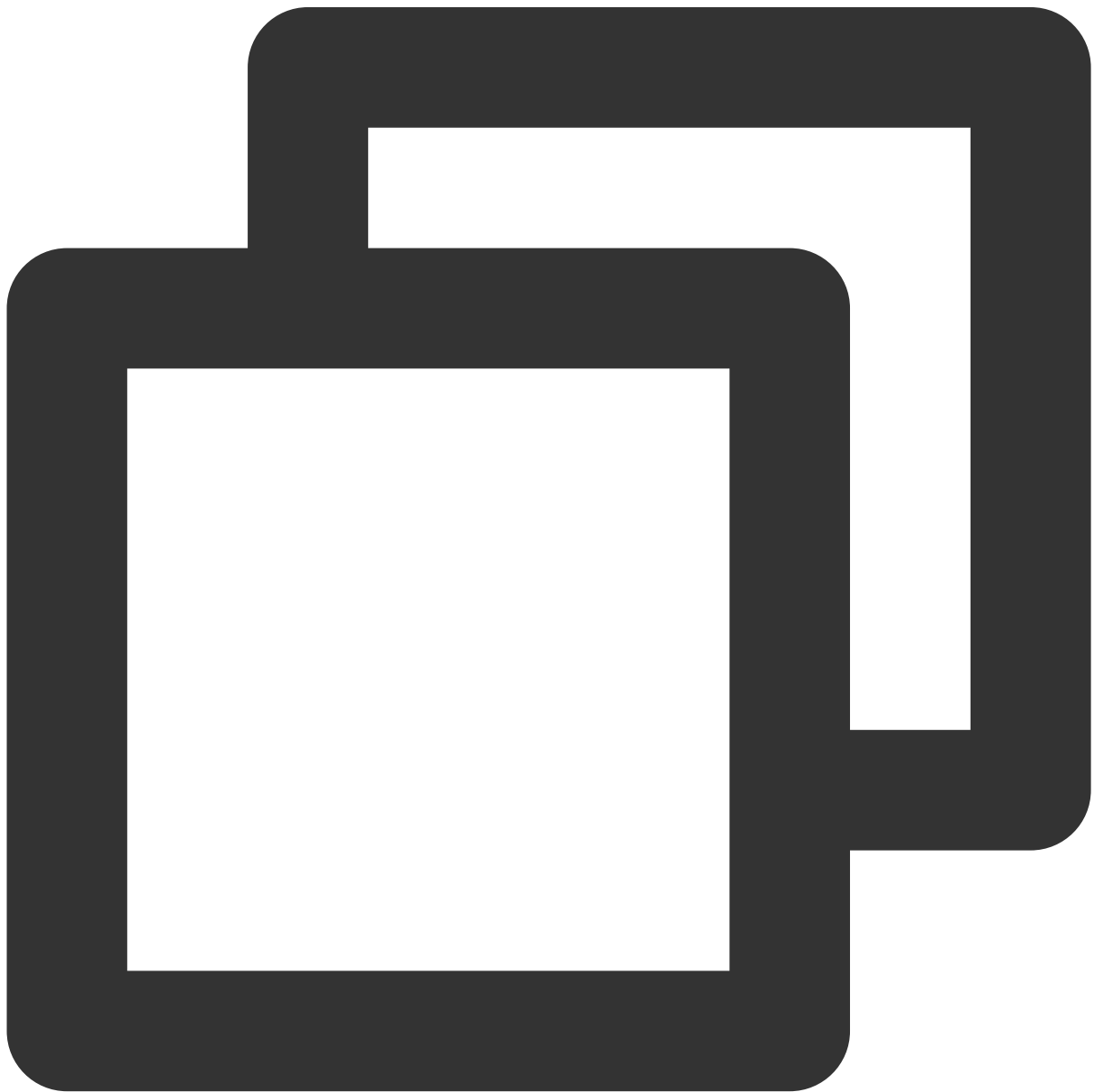
The parameters are described below:

Parameter	Type	Description
sdkAppId	int	You can view <code>SDKAppID</code> in <a href="#">Application Management</a> > <b>Application Info</b> of the TRTC console.
userId	String	The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and

		underscores (_).
userSig	String	Tencent Cloud's proprietary security signature. For how to calculate and use it, see <a href="#">UserSig</a> .
callback	TUICommonDefine.Callback	The initialization callback. <code>onSuccess</code> indicates initialization is successful.

## addObserver

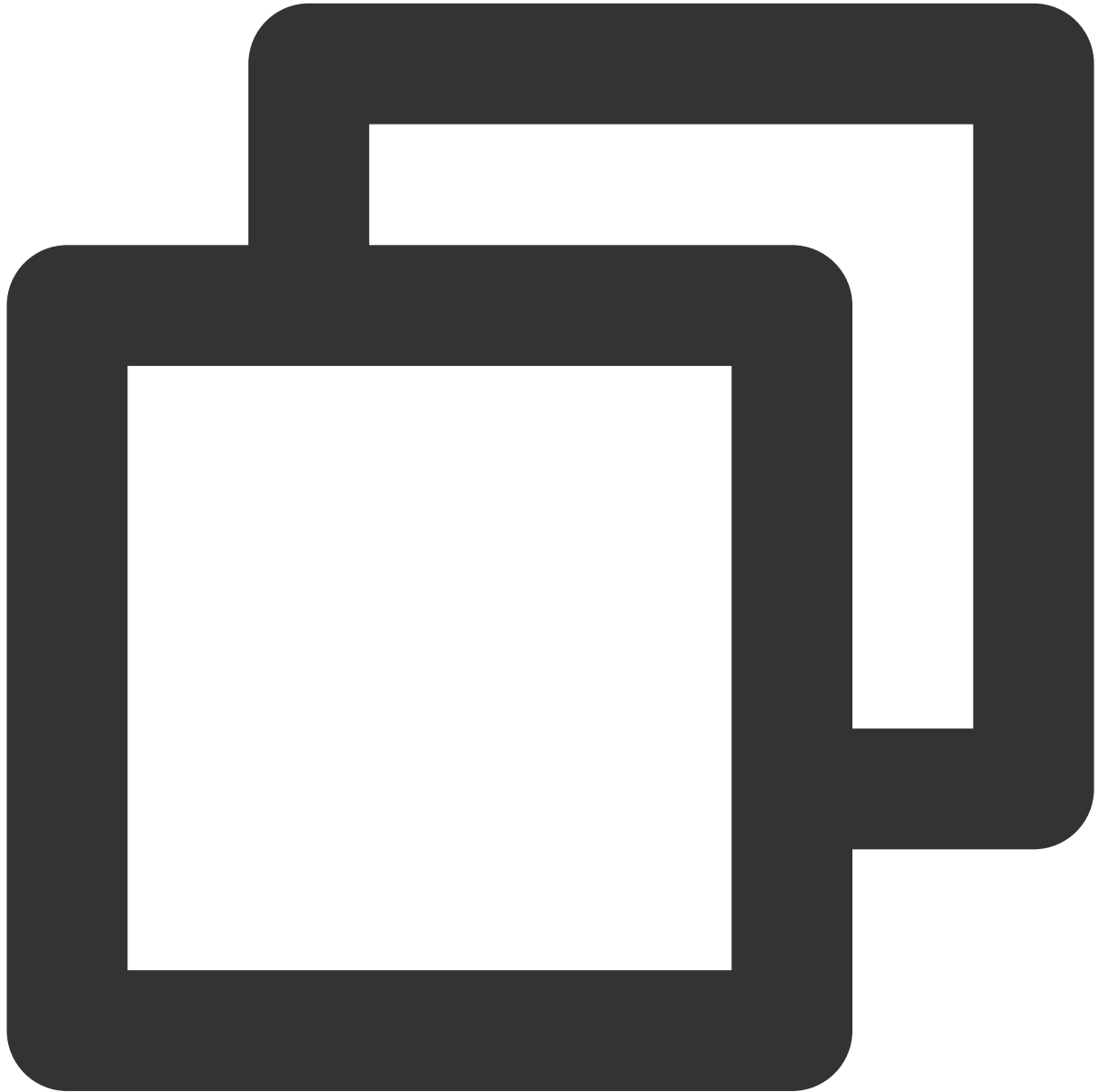
This API is used to register an event listener to listen for `TUICallObserver` events.



```
void addObserver(TUICallObserver observer);
```

## **removeObserver**

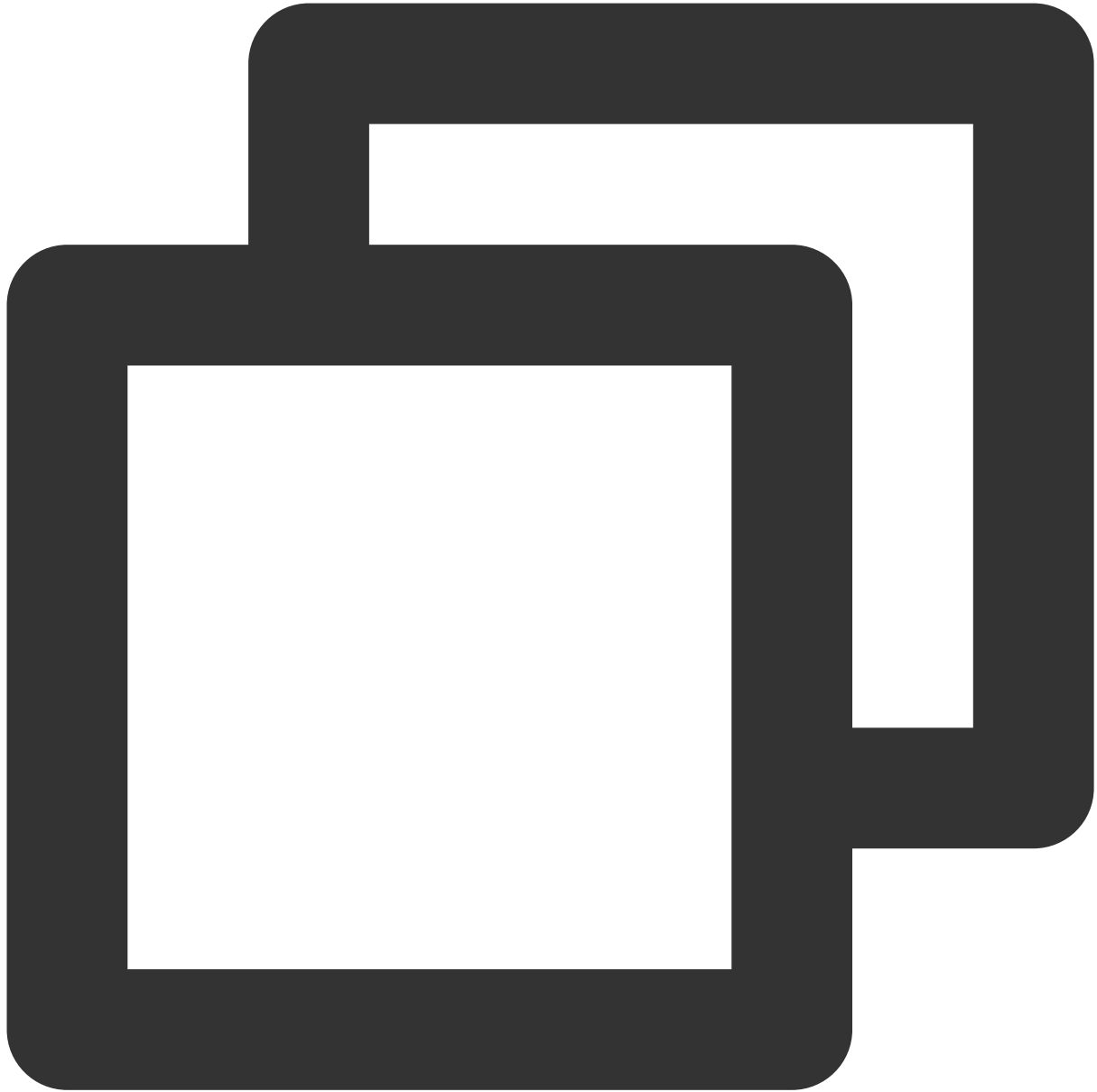
This API is used to unregister an event listener.



```
void removeObserver(TUICallObserver observer);
```

## **call**

This API is used to make a (one-to-one) call.



```
void call(String userId, TUICallDefine.MediaType callMediaType,  
          TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.

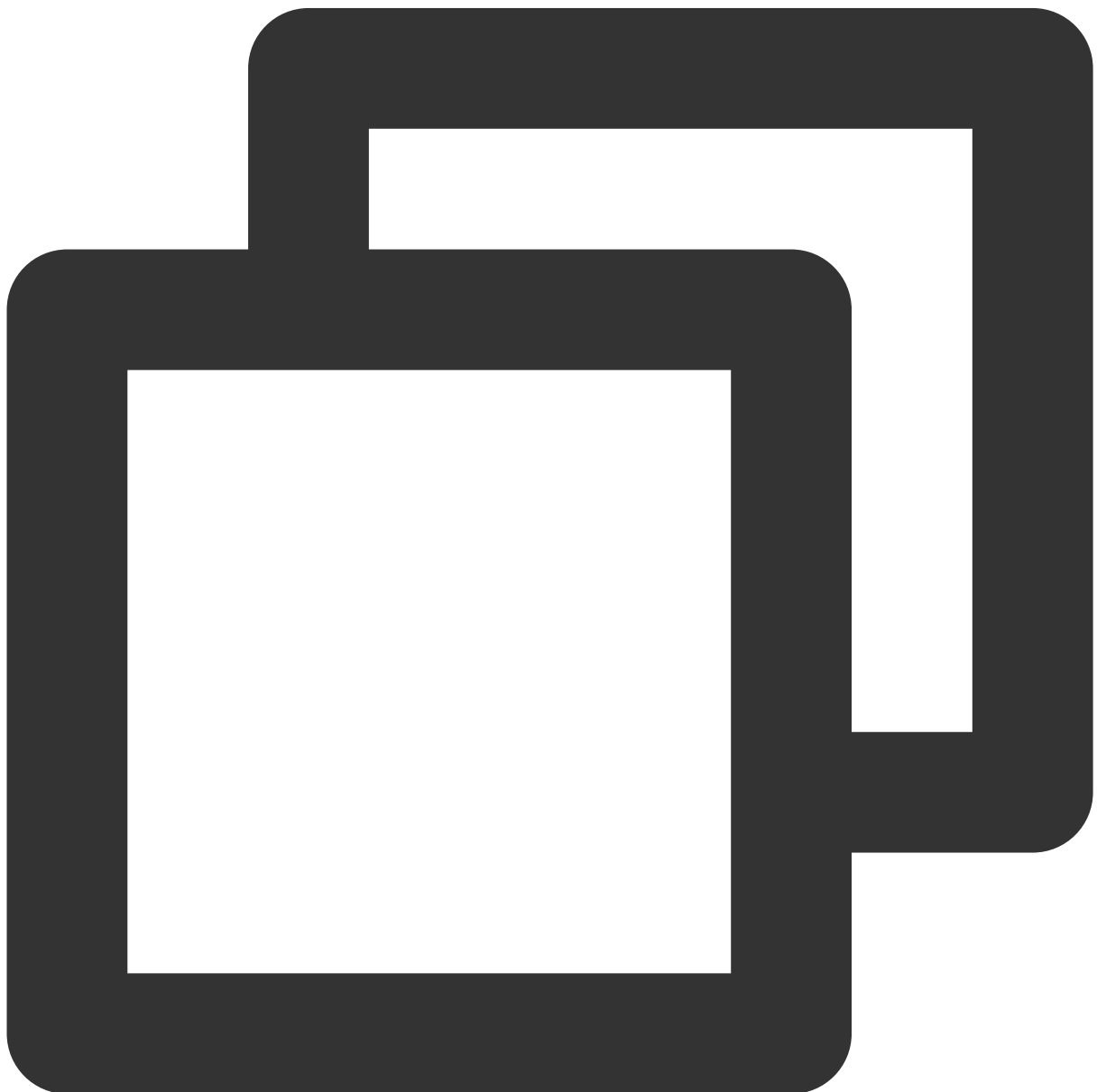
params	<a href="#">TUICallDefine.CallParams</a>	An additional parameter, such as roomId, call timeout, offline push info,etc
--------	--	--

## groupCall

This API is used to make a group call.

### Notice :

Before making a group call, you need to create an IM group first.



```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal
```

```
TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

The parameters are described below:

Parameter	Type	Description
groupId	String	The group ID.
userIdList	List	The target user IDs.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.
params	<a href="#">TUICallDefine.CallParams</a>	An additional parameter. such as roomId, call timeout, offline push info,etc

## accept

This API is used to accept a call. After receiving the `onCallReceived()` callback, you can call this API to accept the call.



```
void accept(TUICommonDefine.Callback callback);
```

## reject

This API is used to reject a call. After receiving the `onCallReceived()` callback, you can call this API to reject the call.



```
void reject(TUICommonDefine.Callback callback);
```

## ignore

This API is used to ignore a call. After receiving the `onCallReceived()` , you can call this API to ignore the call. The caller will receive the `onUserLineBusy` callback.

Note: If your project involves live streaming or conferencing, you can also use this API to implement the “in a meeting” or “on air” feature.





```
void ignore(TUICommonDefine.Callback callback);
```

## hangup

This API is used to end a call.

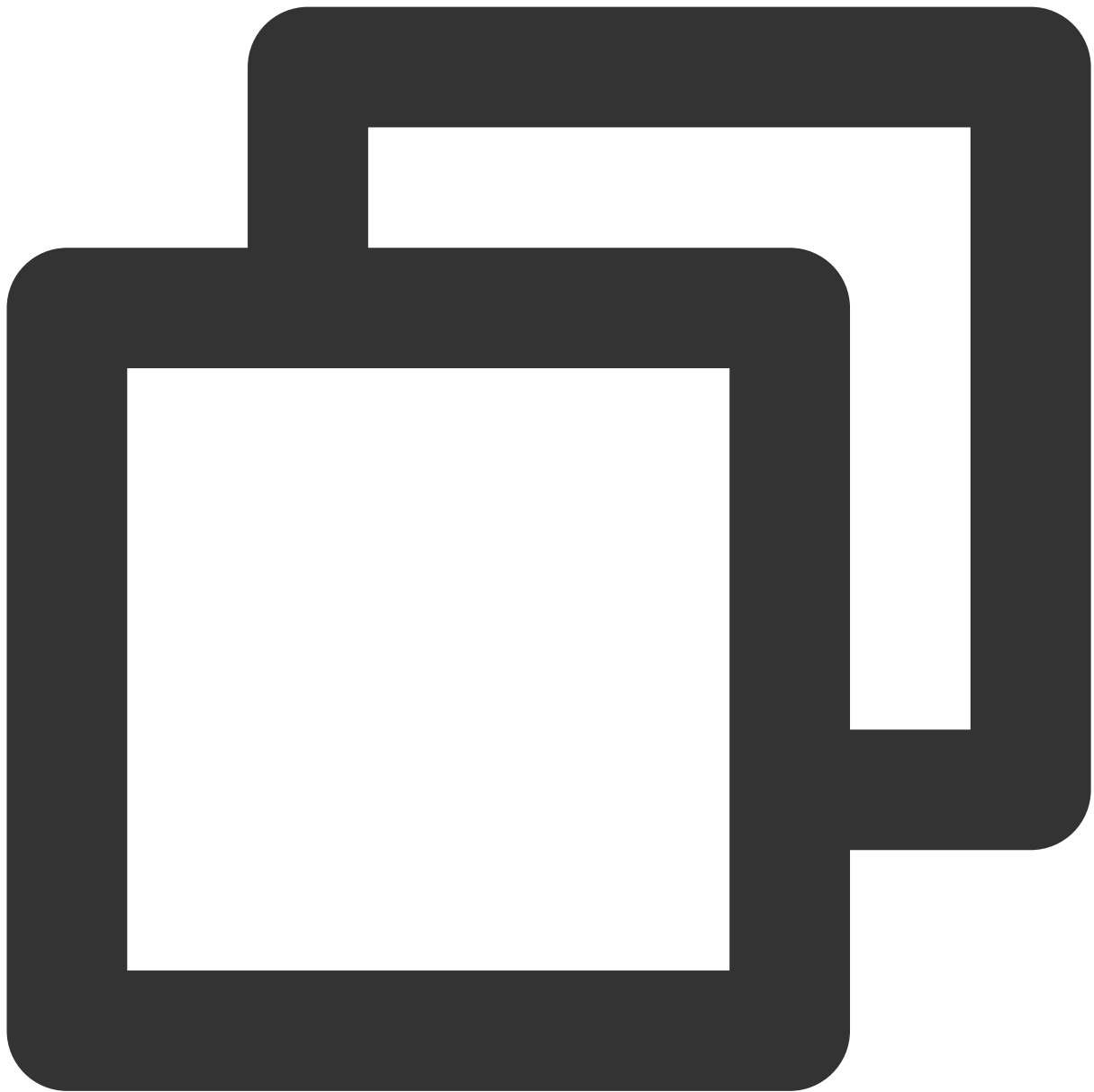


```
void hangup(TUICommonDefine.Callback callback);
```

### **inviteUser**

This API is used to invite users to the current group call.

This API is called by a participant of a group call to invite new users.



```
void inviteUser(List<String> userIdList, TUICallDefine.CallParams params,  
                TUICommonDefine.ValueCallback callback);
```

The parameters are described below:

Parameter	Type	Description
userIdList	List	The target user IDs.
params	<a href="#">TUICallDefine.CallParams</a>	An additional parameter. such as roomId, call timeout, offline push info,etc

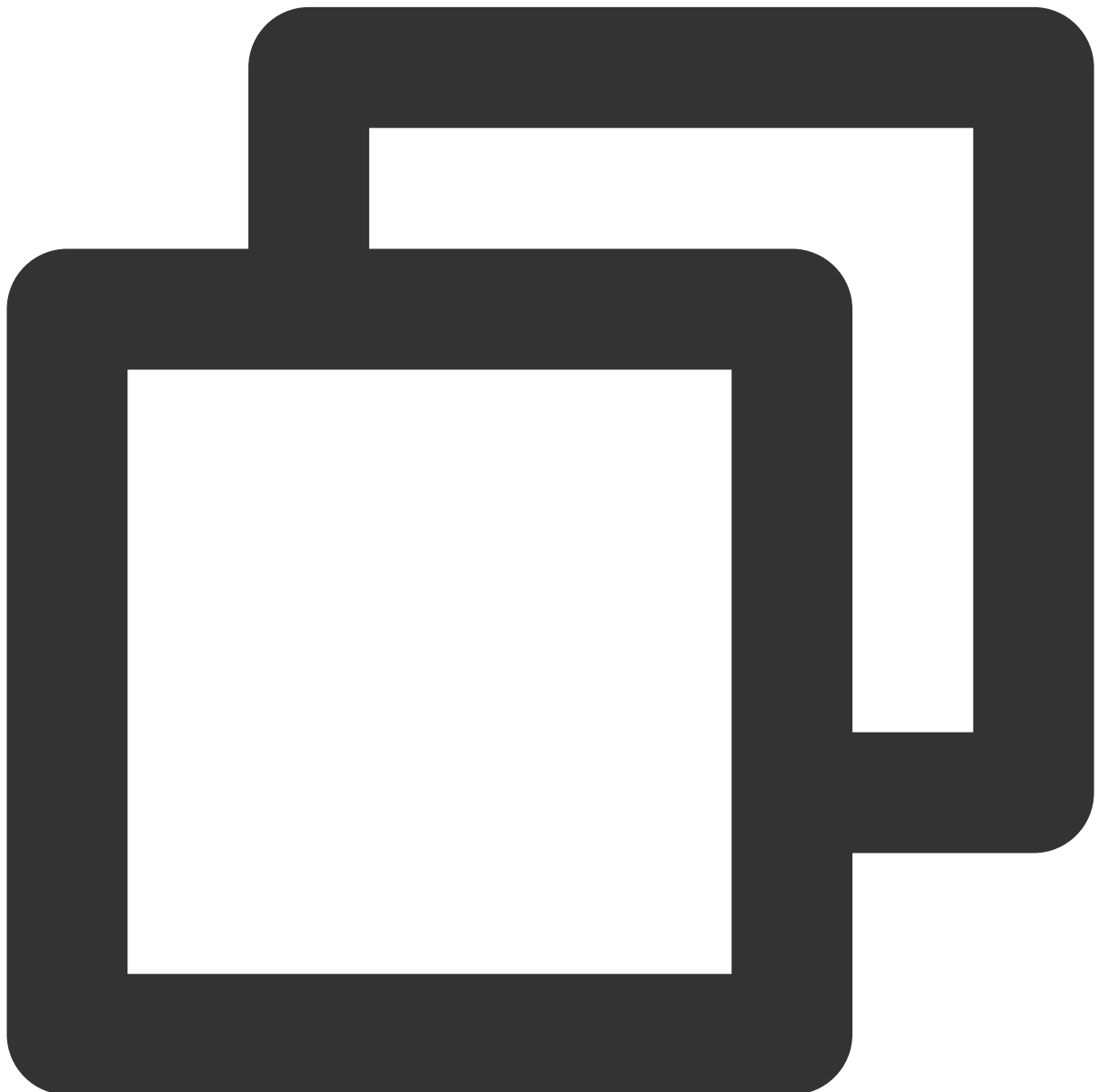
**Notice :**

In this case, the custom RoomId is invalid. The SDK will invite others to join the room where the inviter is currently located.

**joinInGroupCall**

This API is used to join a group call.

This API is called by a group member to join the group's call.



```
void joinInGroupCall(TUICommonDefine.RoomId roomId, String groupId,  
                    TUICallDefine.MediaType callMediaType, TUICommonDefine.Callbac
```

The parameters are described below:

Parameter	Type	Description
roomId	<a href="#">TUICommonDefine.RoomId</a>	The room ID.
groupId	String	The group ID.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.

### **switchCallMediaType**

This API is used to change the call type.



```
void switchCallMediaType(TUICallDefine.MediaType callMediaType);
```

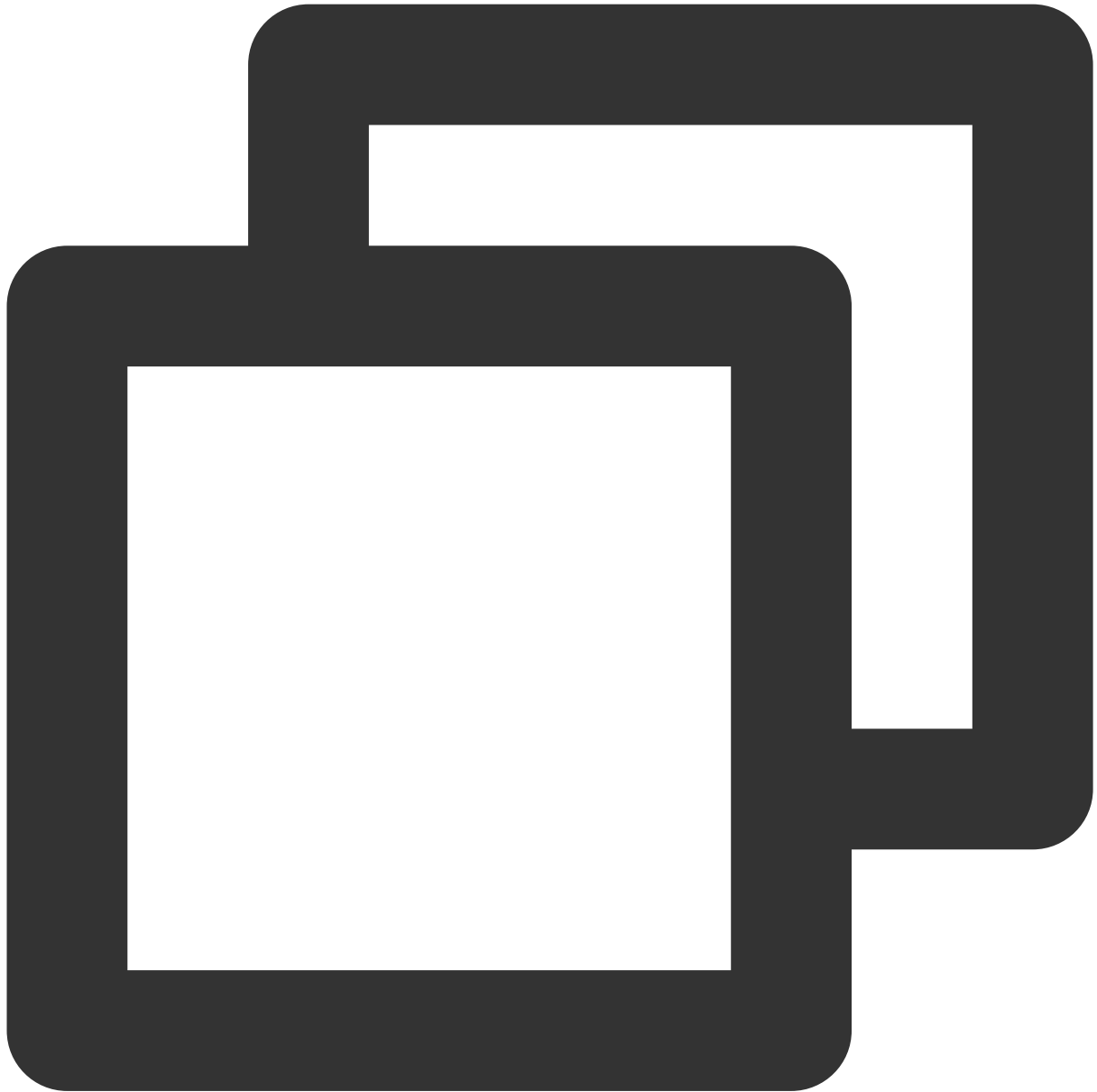
The parameters are described below:

Parameter	Type	Description
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.

## startRemoteView

This API is used to subscribe to the video stream of a remote user. For it to work, make sure you call it after

```
setRenderView .
```



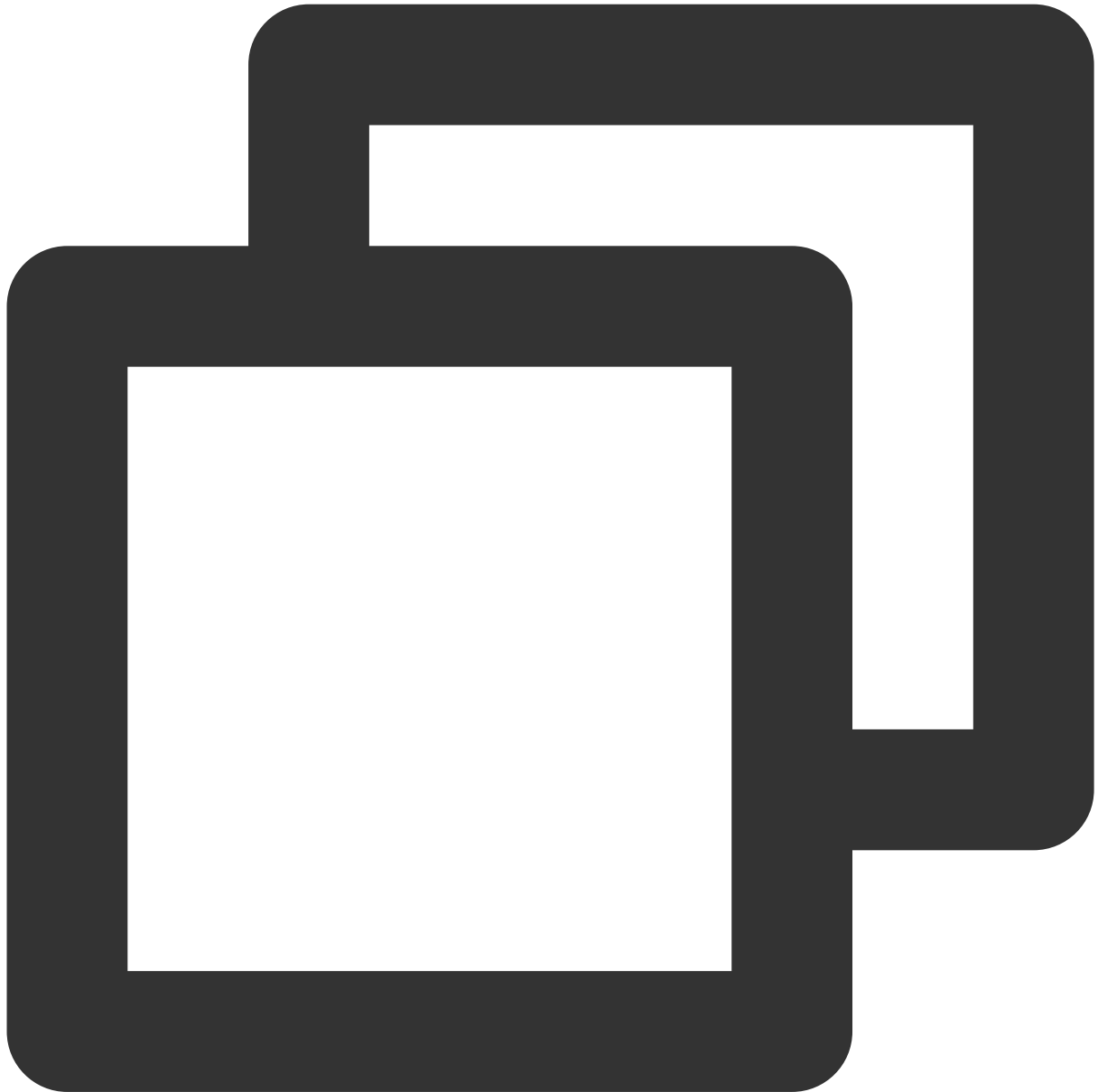
```
void startRemoteView(String userId, TUIVideoView videoView, TUICommonDefine.PlayCal
```

The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
videoView	TUIVideoView	The view to be rendered.

## stopRemoteView

This API is used to unsubscribe from the video stream of a remote user.



```
void stopRemoteView(String userId);
```

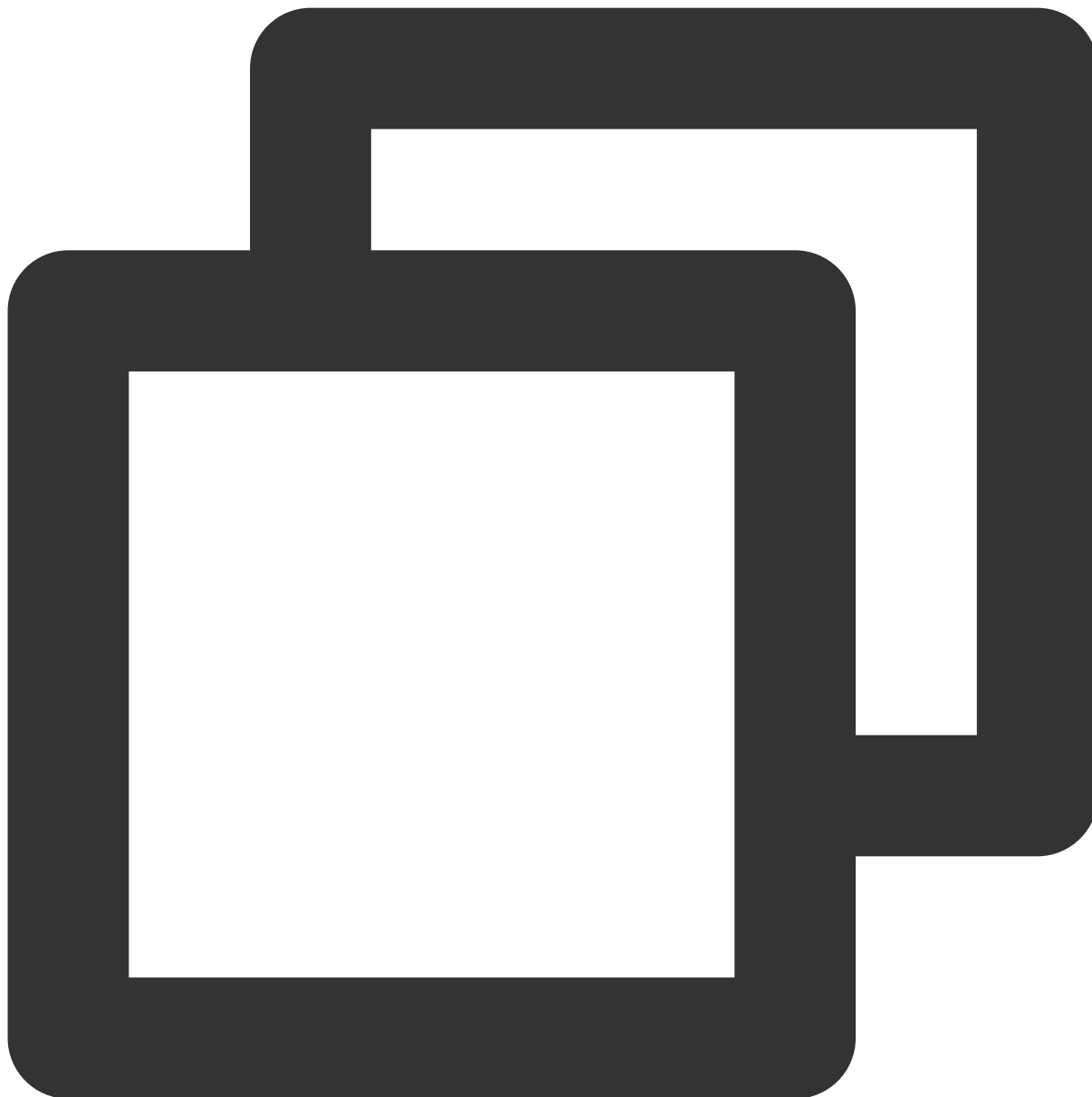
The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.



## openCamera

This API is used to turn the camera on.



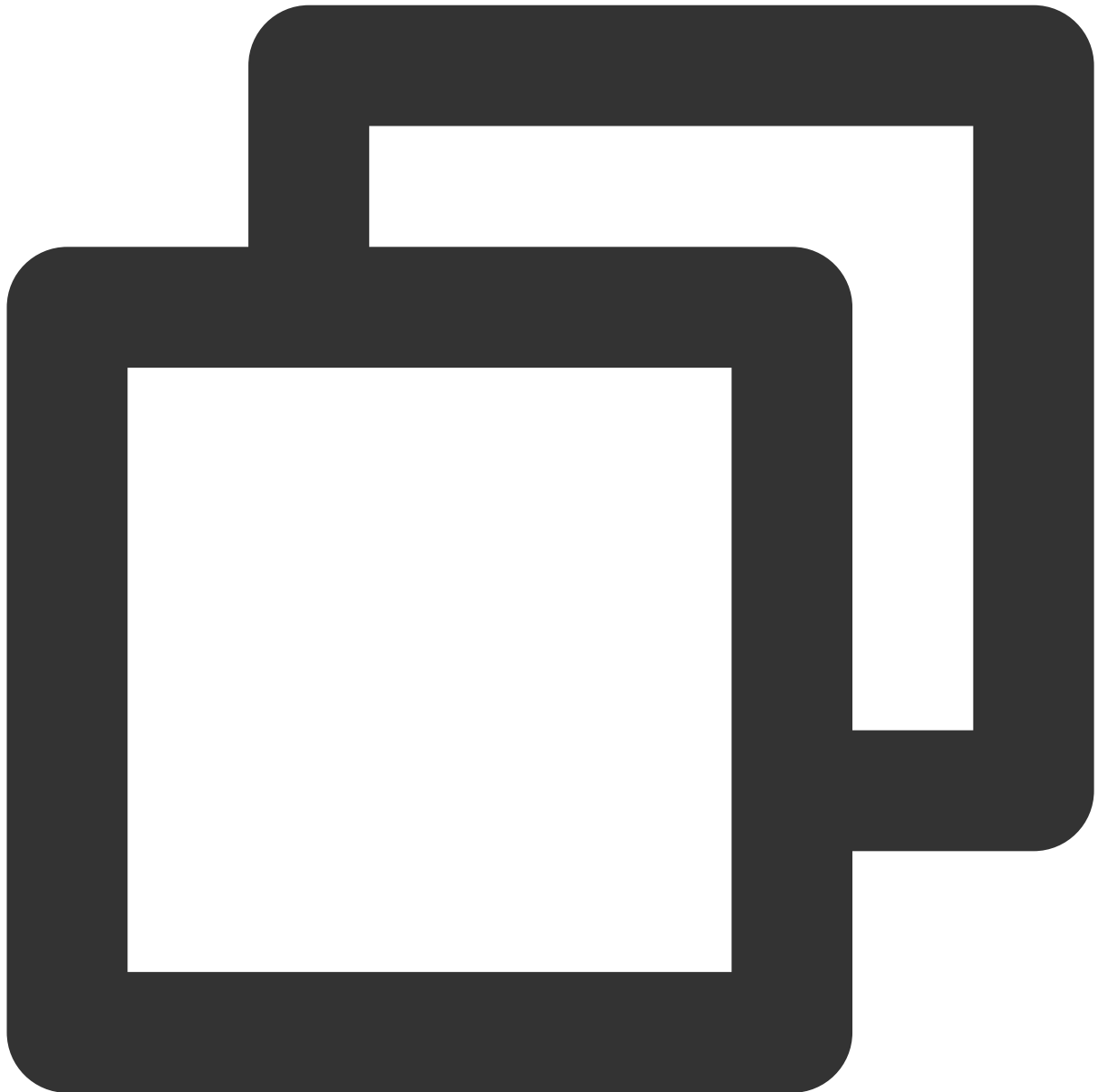
```
void openCamera(TUICommonDefine.Camera camera, TUIVideoView videoView, TUICommonDef
```

The parameters are described below:

Parameter	Type	Description
camera	<a href="#">TUICommonDefine.Camera</a>	The front or rear camera.
videoView	TUIVideoView	The view to be rendered.

## closeCamera

This API is used to turn the camera off.



```
void closeCamera();
```

## switchCamera

This API is used to switch between the front and rear cameras.



```
void switchCamera(TUICommonDefine.Camera camera);
```

The parameters are described below:

Parameter	Type	Description
camera	<a href="#">TUICommonDefine.Camera</a>	The front or rear camera.

## openMicrophone

This API is used to turn the mic on.



```
void openMicrophone(TUICommonDefine.Callback callback);
```

### **closeMicrophone**

This API is used to turn the mic off.



```
void closeMicrophone();
```

### **selectAudioPlaybackDevice**

This API is used to select the audio playback device (receiver or speaker). In call scenarios, you can use this API to turn on/off hands-free mode.



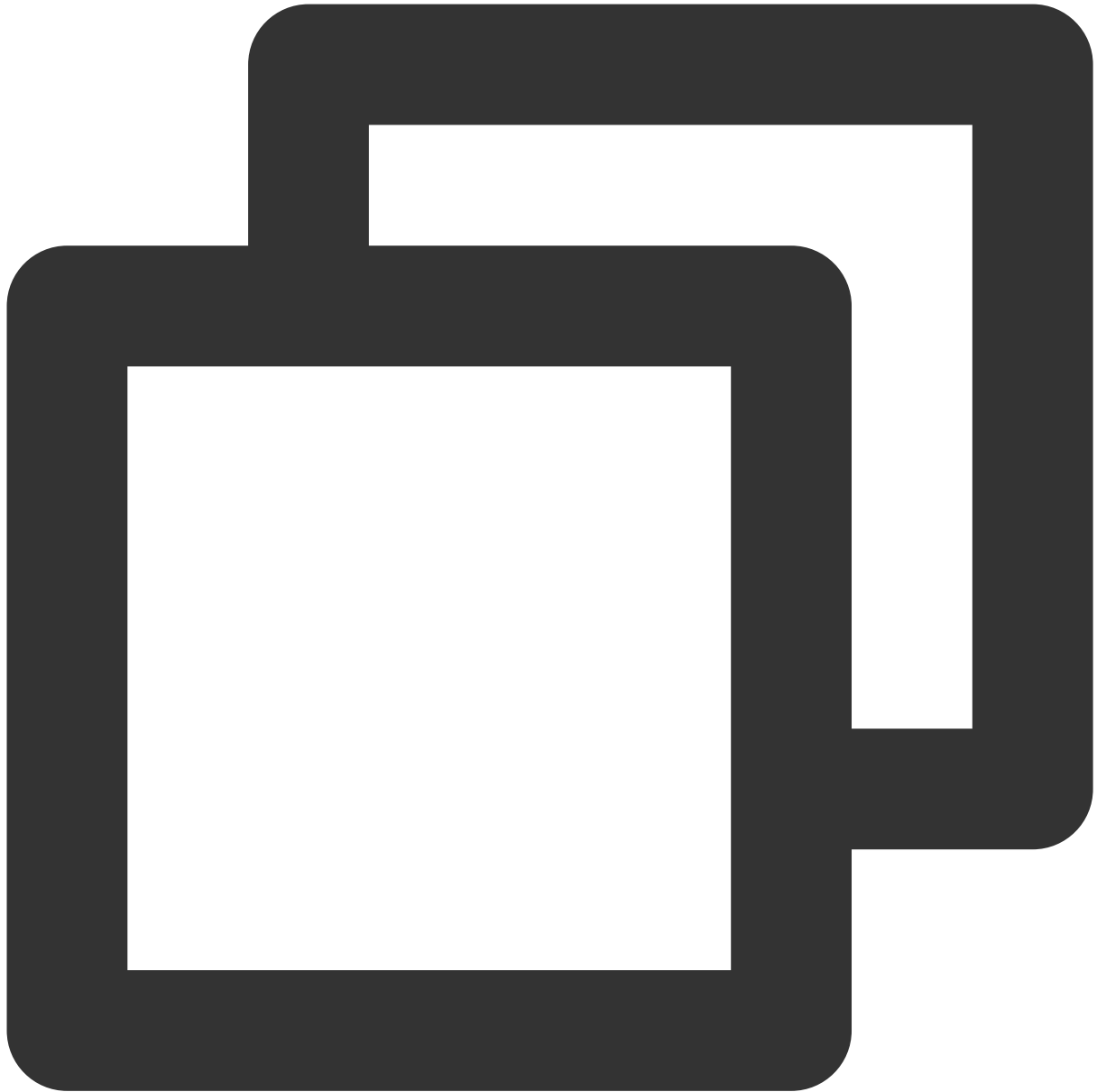
```
void selectAudioPlaybackDevice(TUICommonDefine.AudioPlaybackDevice device);
```

The parameters are described below:

Parameter	Type	Description
device	<a href="#">TUICommonDefine.AudioPlaybackDevice</a>	The speaker or receiver.

## setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.



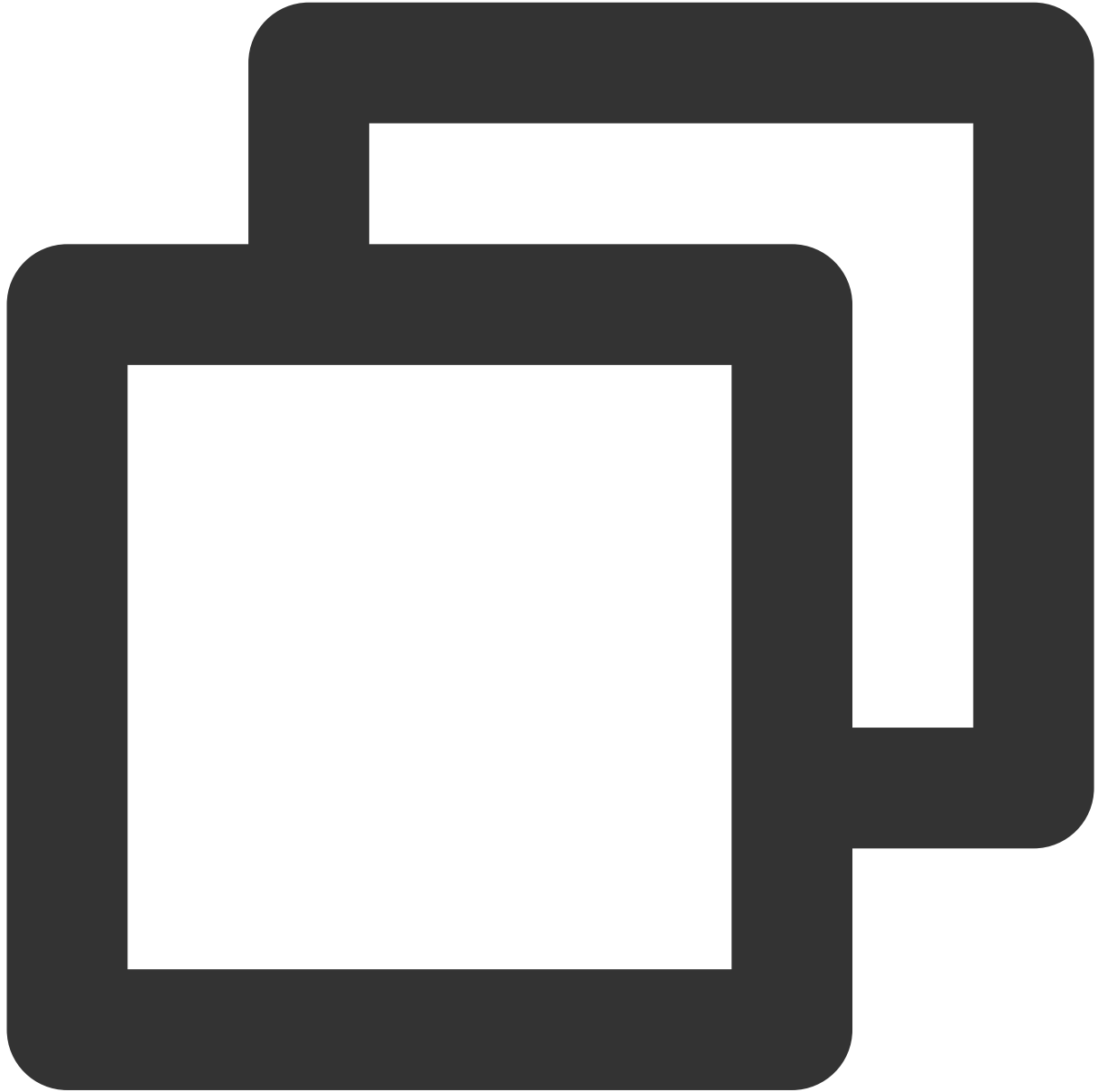
```
void setSelfInfo(String nickname, String avatar, TUICommonDefine.Callback callback)
```

The parameters are described below:

Parameter	Type	Description
nickname	String	The alias.
avatar	String	The URL of the profile photo.

## **enableMultiDeviceAbility**

This API is used to set whether to enable multi-device login for `TUICallEngine` (supported by the premium package).



```
void enableMultiDeviceAbility(boolean enable, TUICommonDefine.Callback callback);
```

## **setVideoRenderParams**

Set the rendering mode of video image.





```
void setVideoRenderParams(String userId, TUICommonDefine.VideoRenderParams params,
```

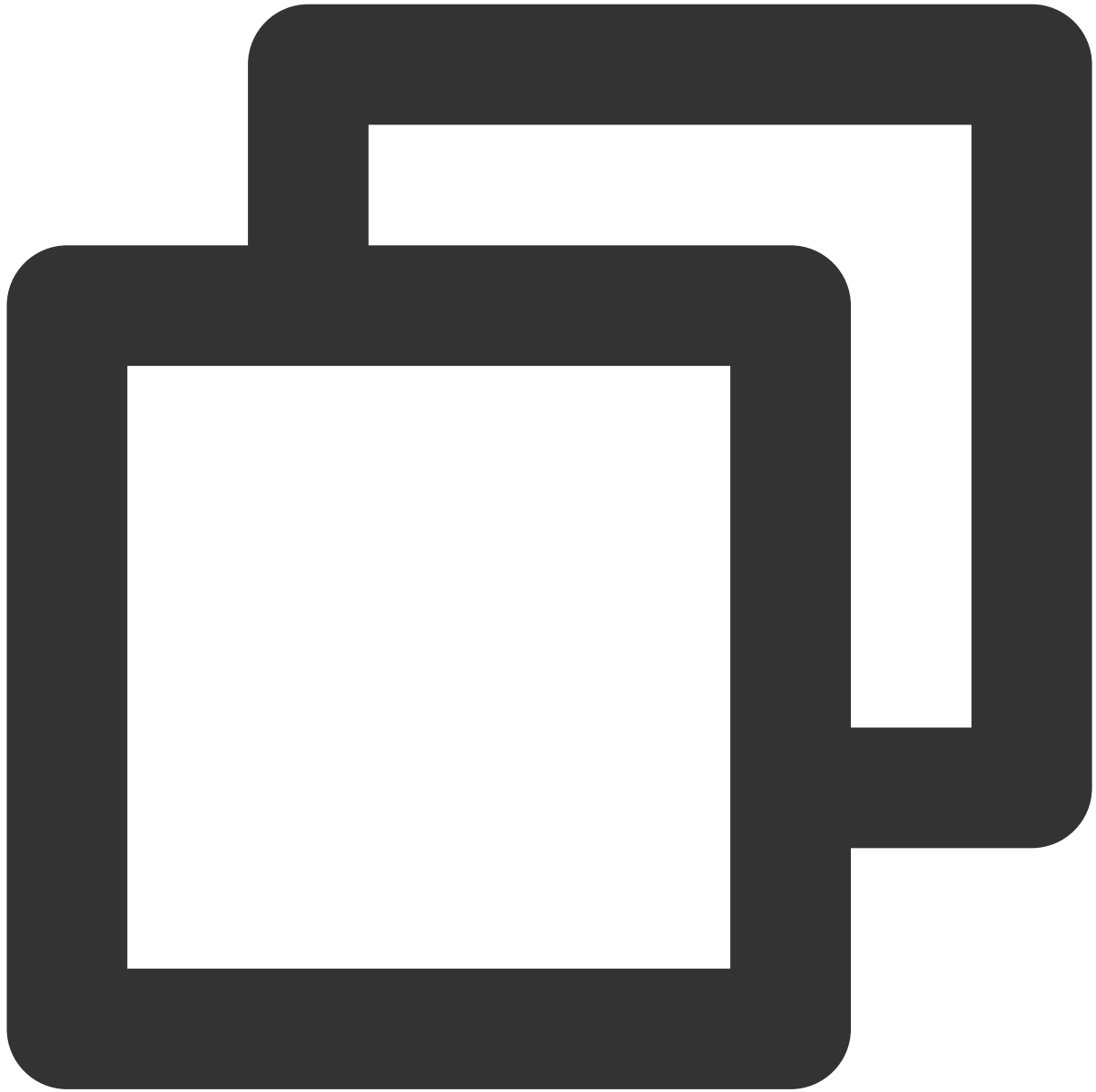
The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
params	<a href="#">TUICommonDefine.VideoRenderParams</a>	Video render parameters.

## setVideoEncoderParams

Set the encoding parameters of video encoder.

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.



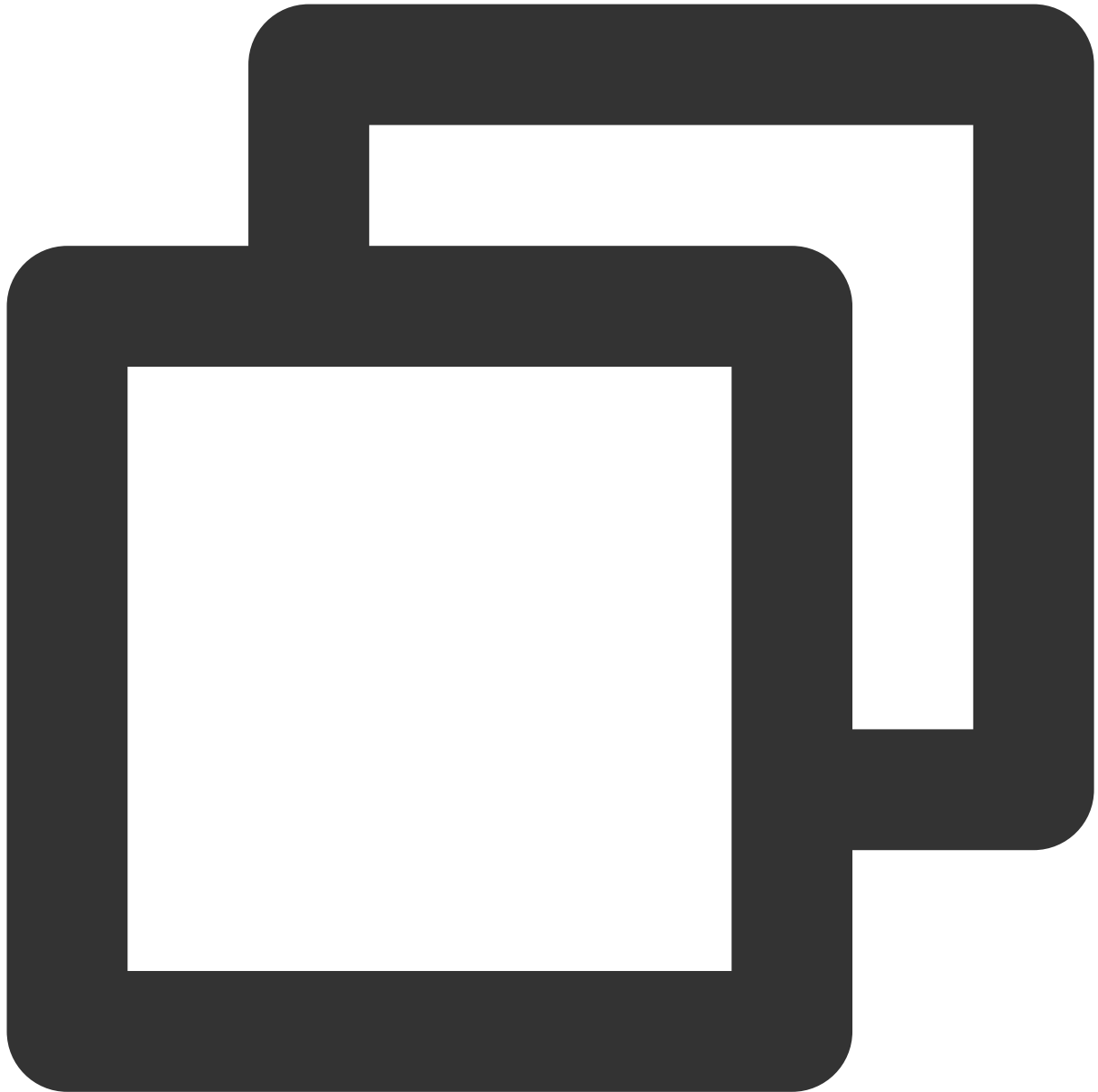
```
void setVideoEncoderParams(TUICommonDefine.VideoEncoderParams params, TUICommonDefine.VideoEncoderParams)
```

The parameters are described below:

Parameter	Type	Description
params	<a href="#">TUICommonDefine.VideoEncoderParams</a>	Video encoding parameters

## getTRTCCloudInstance

Advanced features.



```
TRTCCloud getTRTCCloudInstance();
```

## setBeautyLevel

Set beauty level, support turning off default beauty.



```
void setBeautyLevel(float level, TUICommonDefine.Callback callback);
```

The parameters are described below:

Parameter	Type	Description
level	float	Beauty level, range: 0 - 9 ; 0 means turning off the effect, 9 means the most obvious effect.



# TUICallObserver

Last updated : 2024-01-17 11:58:53

## TUICallObserver APIs

`TUICallObserver` is the callback class of `TUICallEngine`. You can use it to listen for events.

## Overview

API	Description
<code>onError</code>	A call occurred during the call.
<code>onCallReceived</code>	A call invitation was received.
<code>onCallCancelled</code>	The call was canceled.
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user had a video stream.
<code>onUserAudioAvailable</code>	Whether a user had an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user was kicked offline.

`onUserSigExpired`

The user sig is expired.

## Details

### **onError**

An error occurred.

#### **Note**

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



```
void onError(int code, String msg);
```

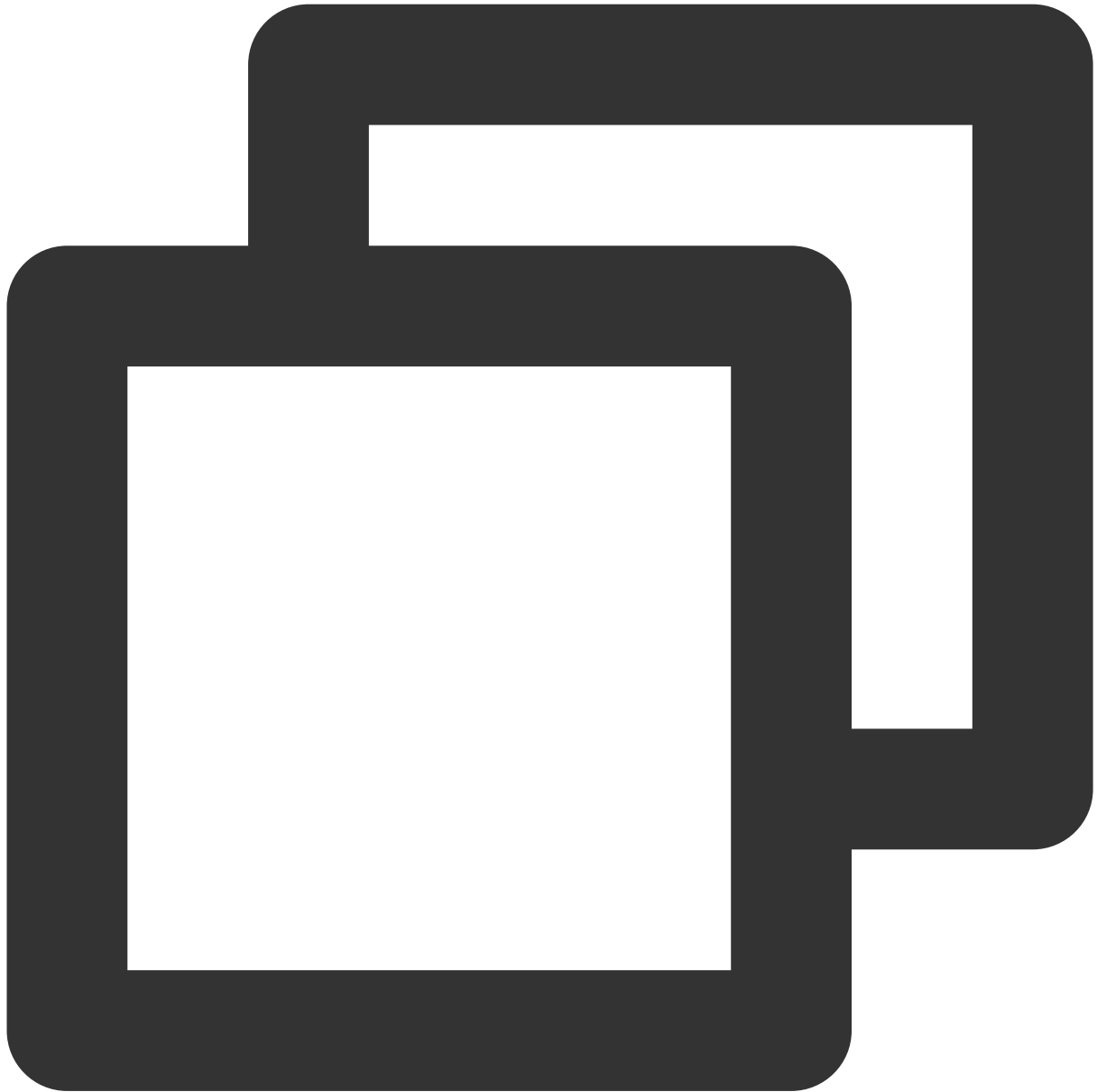
The parameters are described below:

Parameter	Type	Description
code	int	The error code.
msg	String	The error message.

## onCallReceived



A call invitation was received. This callback is received by an invitee. You can listen for this event to determine whether to display the incoming call view.



```
void onCallReceived(String callerId, List<String> calleeIdList, String groupId,  
    TUICallDefine.MediaType callMediaType, String userData);
```

The parameters are described below:

Parameter	Type	Description
callerId	String	The user ID of the inviter.

calleeIdList	List	The invitee list.
groupId	String	The group ID.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.
userData	String	User-added extended fields., Please refer to: <a href="#">TUICallDefine.CallParams</a>

## onCallCancelled

The call was canceled by the inviter or timed out. This callback is received by an invitee. You can listen for this event to determine whether to show a missed call message.

This indicates that the call was canceled by the caller, timed out by the callee, rejected by the callee, or the callee was busy. There are multiple scenarios involved. You can listen to this event to achieve UI logic such as missed calls and resetting UI status.

Call cancellation by the caller: The caller receives the callback (userId is himself); the callee receives the callback (userId is the ID of the caller)

Callee timeout: the caller will simultaneously receive the [onUserNoResponse](#) and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)

Callee rejection: The caller will simultaneously receive the [onUserReject](#) and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)

Callee busy: The caller will simultaneously receive the [onUserLineBusy](#) and onCallCancelled callbacks (userId is his own ID);

Abnormal interruption: The callee failed to receive the call , he receives this callback (userId is his own ID).



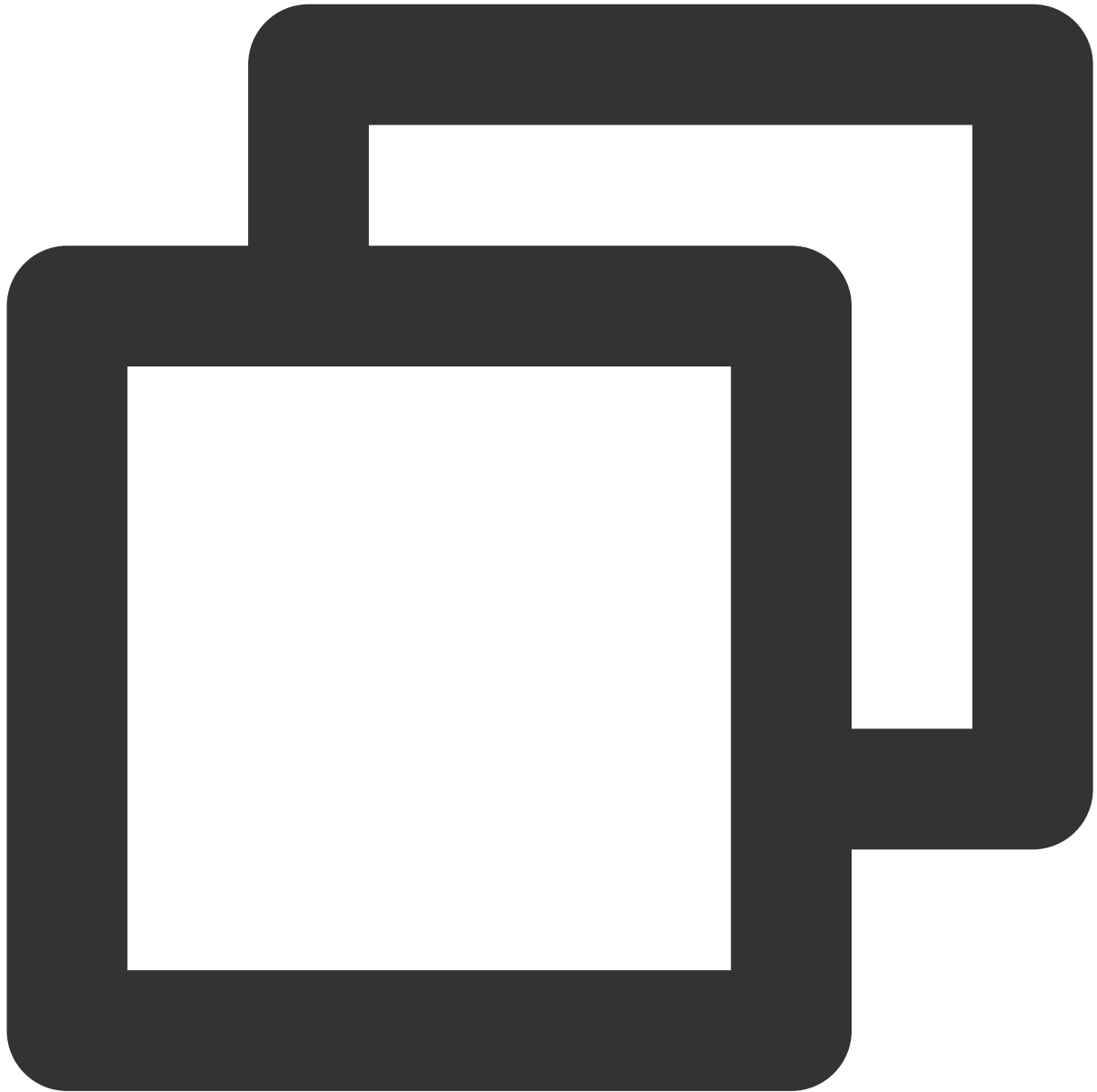
```
void onCallCancelled(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the inviter.

## onCallBegin

The call was connected. This callback is received by both the inviter and invitees. You can listen for this event to determine whether to start on-cloud recording, content moderation, or other tasks.



```
void onCallBegin(TUICCommonDefine.RoomId roomId, TUICallDefine.MediaType callMediaTy
```

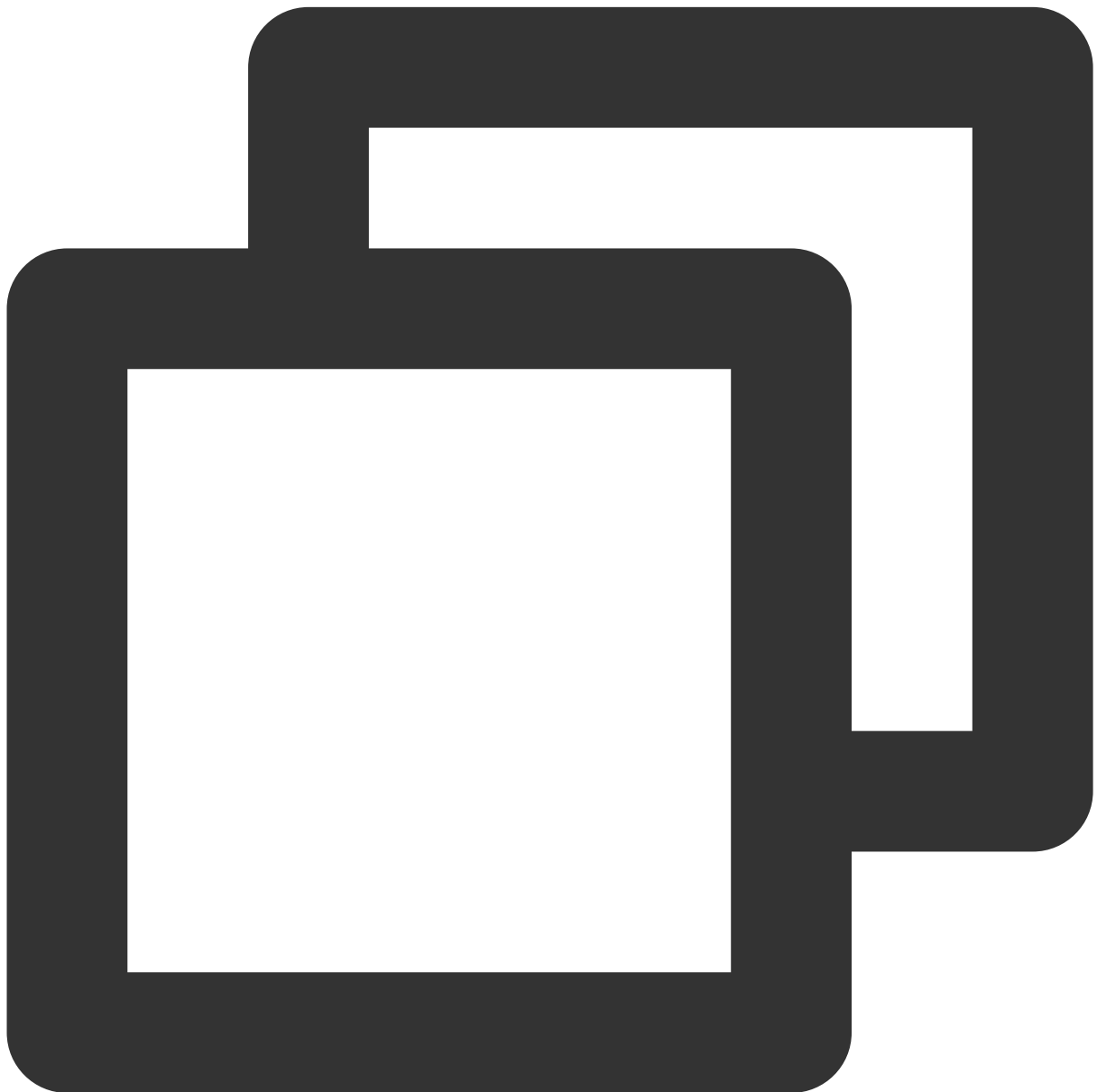
The parameters are described below:

Parameter	Type	Description
roomId	<a href="#">TUICCommonDefine.RoomId</a>	The room ID.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.

callRole	<a href="#">TUICallDefine.Role</a>	The role, which can be caller or callee.
----------	------------------------------------	--

## onCallEnd

The call ended. This callback is received by both the inviter and invitees. You can listen for this event to determine when to display call information such as call duration and call type, or stop on-cloud recording.



```
void onCallEnd(TUICCommonDefine.RoomId roomId, TUICallDefine.MediaType callMediaType
```

The parameters are described below:

--	--	--

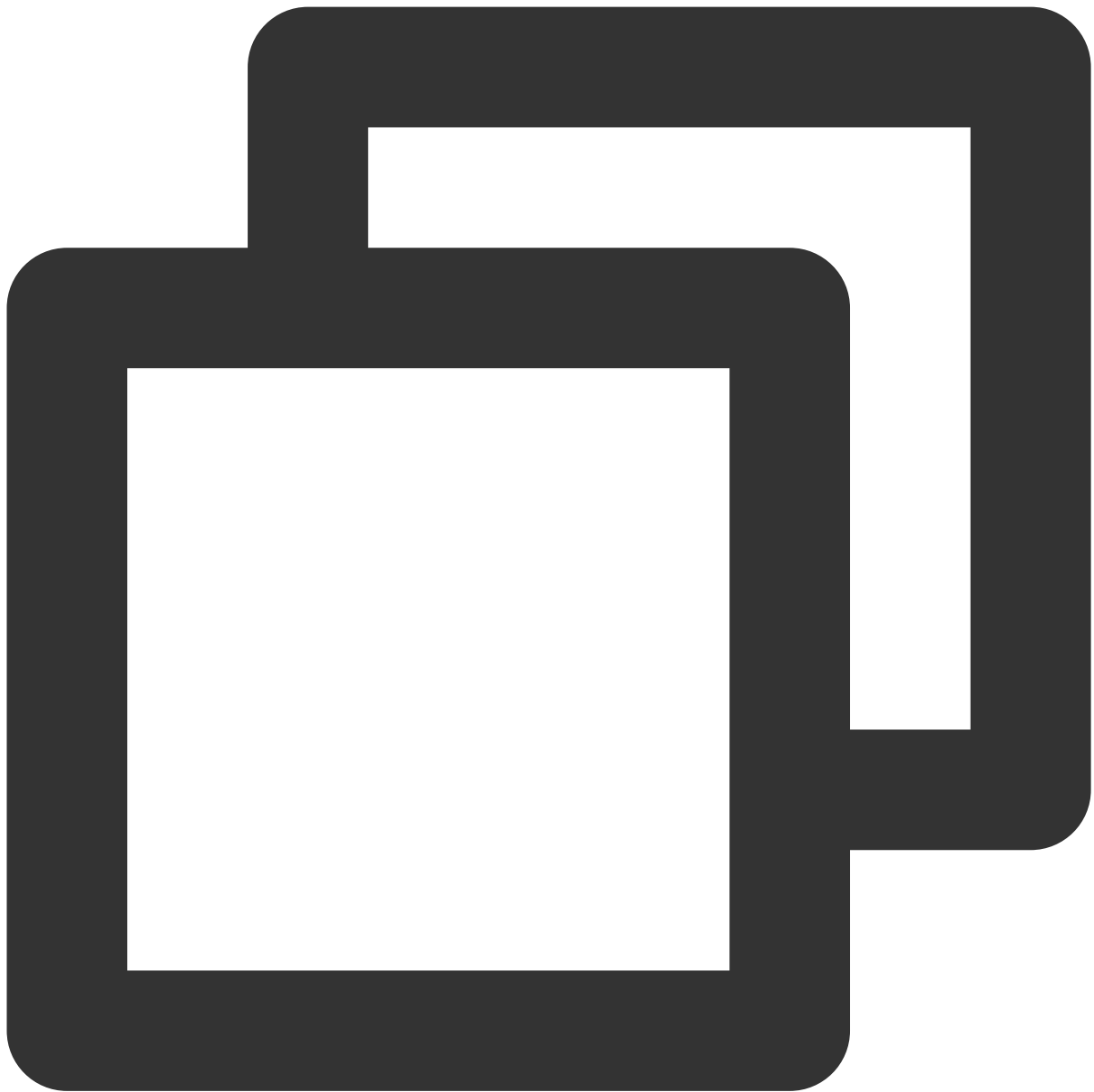
Parameter	Type	Description
roomId	<a href="#">TUICommonDefine.RoomId</a>	The room ID.
callMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type, which can be video or audio.
callRole	<a href="#">TUICallDefine.Role</a>	The role, which can be caller or callee.
totalTime	long	The call duration.

**Note :**

Client-side callbacks are often lost when errors occur, for example, when the process is closed. If you need to measure the duration of a call for billing or other purposes, we recommend you use the RESTful API.

**onCallMediaTypeChanged**

The call type changed.



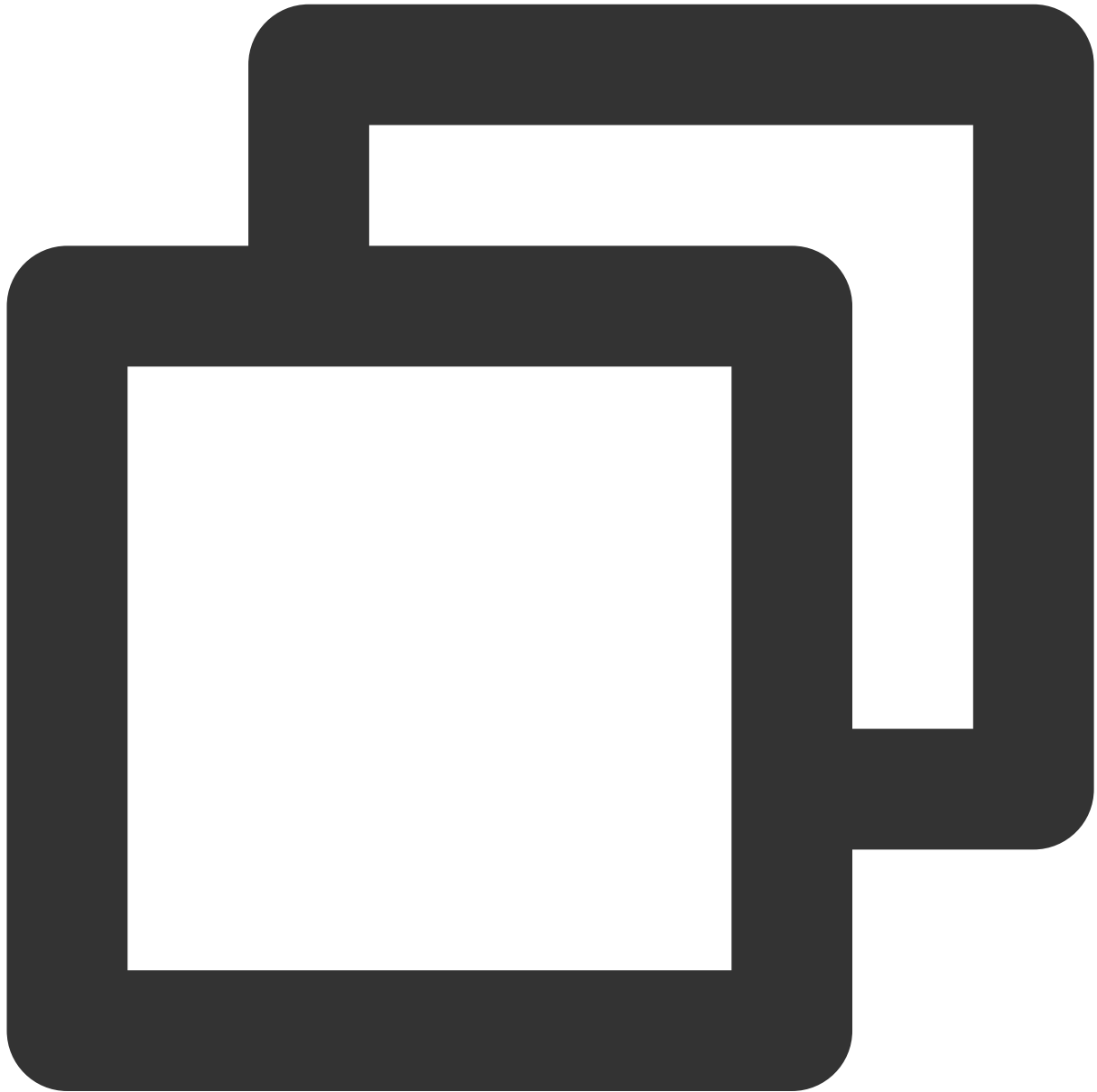
```
void onCallMediaTypeChanged(TUICallDefine.MediaType oldCallMediaType, TUICallDefine.
```

The parameters are described below:

Parameter	Type	Description
oldCallMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type before the change.
newCallMediaType	<a href="#">TUICallDefine.MediaType</a>	The call type after the change.

## onUserReject

The call was rejected. In a one-to-one call, only the inviter will receive this callback. In a group call, all invitees will receive this callback.



```
void onUserReject(String userId);
```

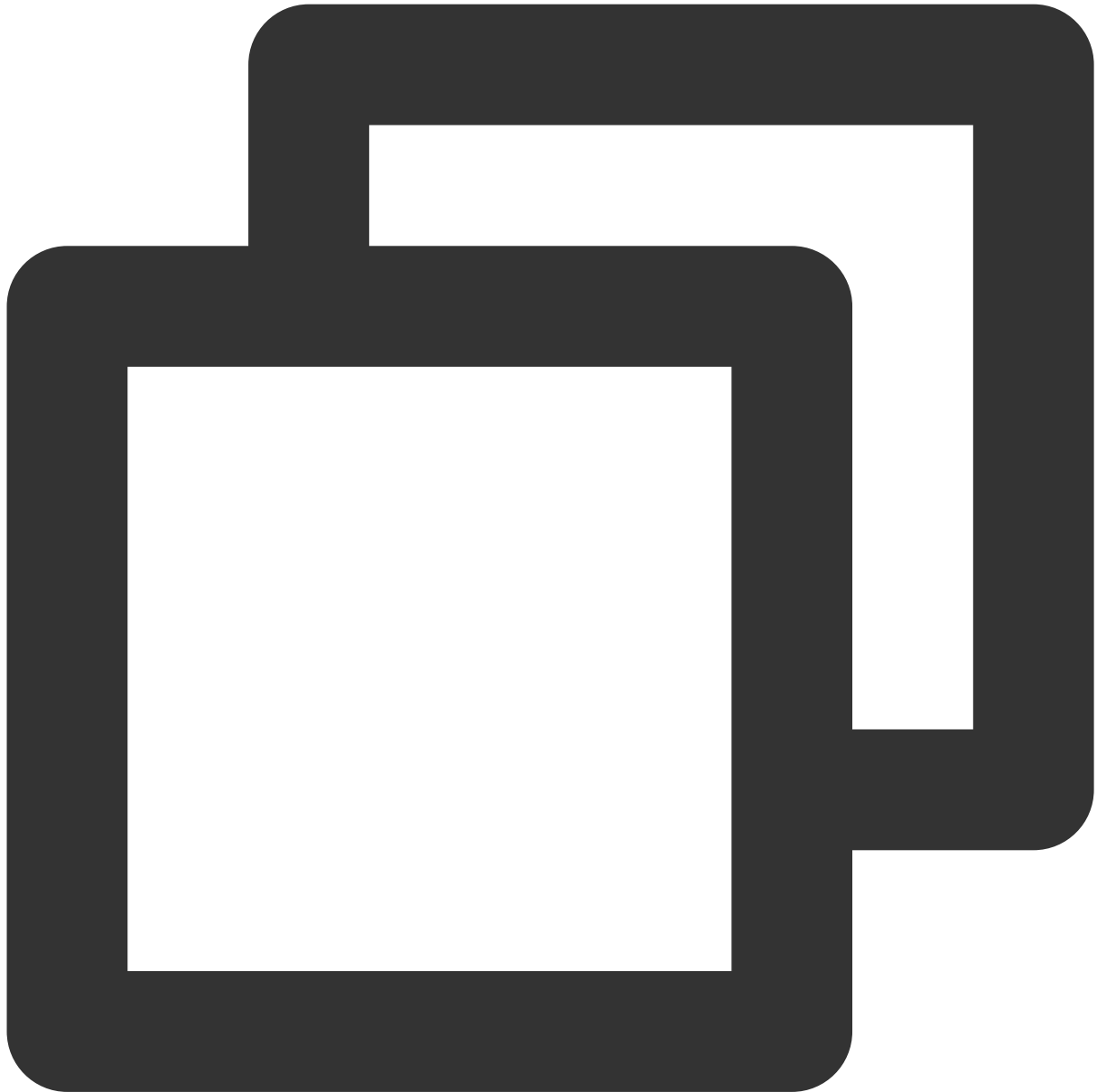
The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who rejected the call.



## onUserNoResponse

A user did not respond.



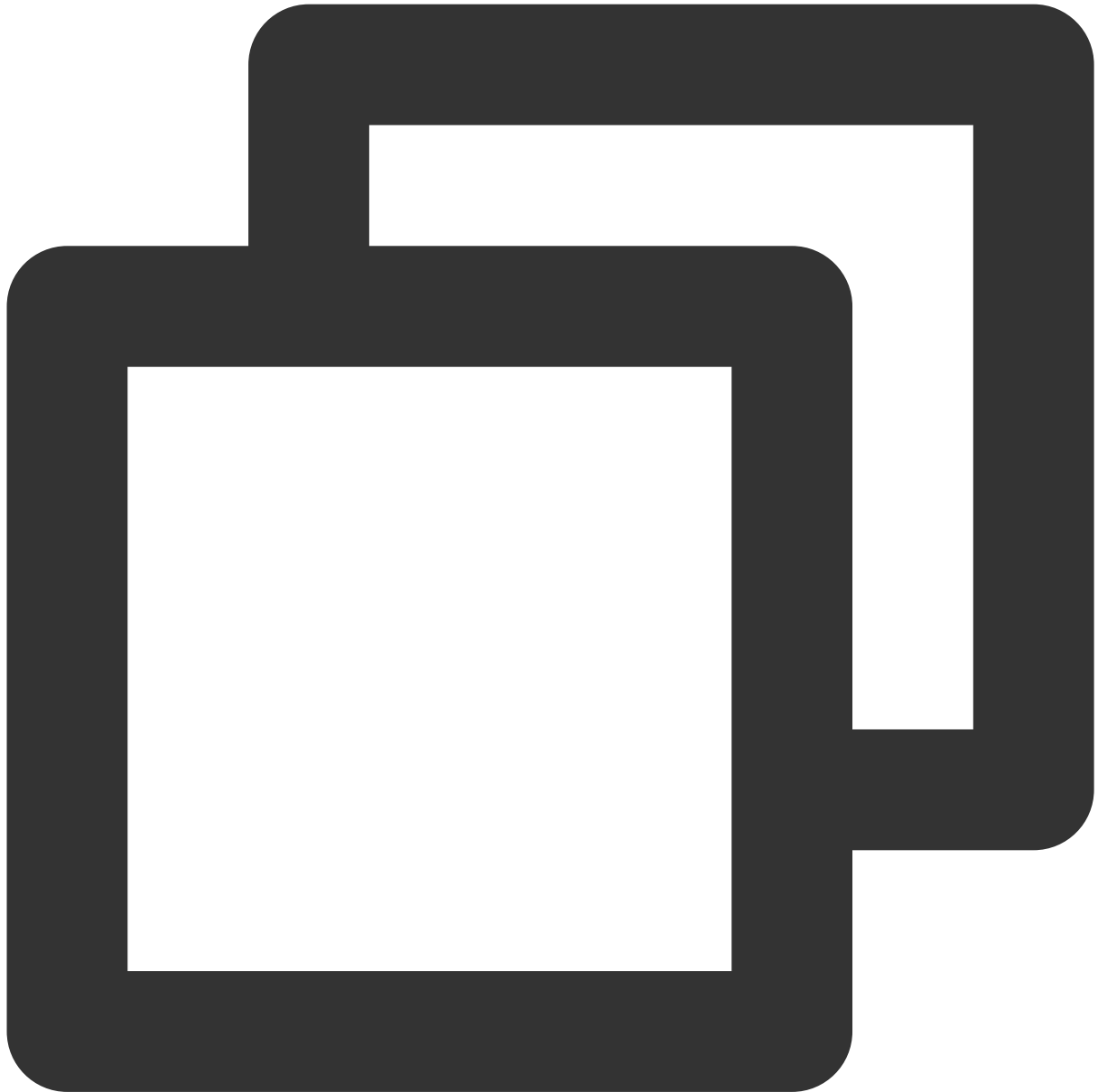
```
void onUserNoResponse(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who did not answer.

## onUserLineBusy

A user is busy.



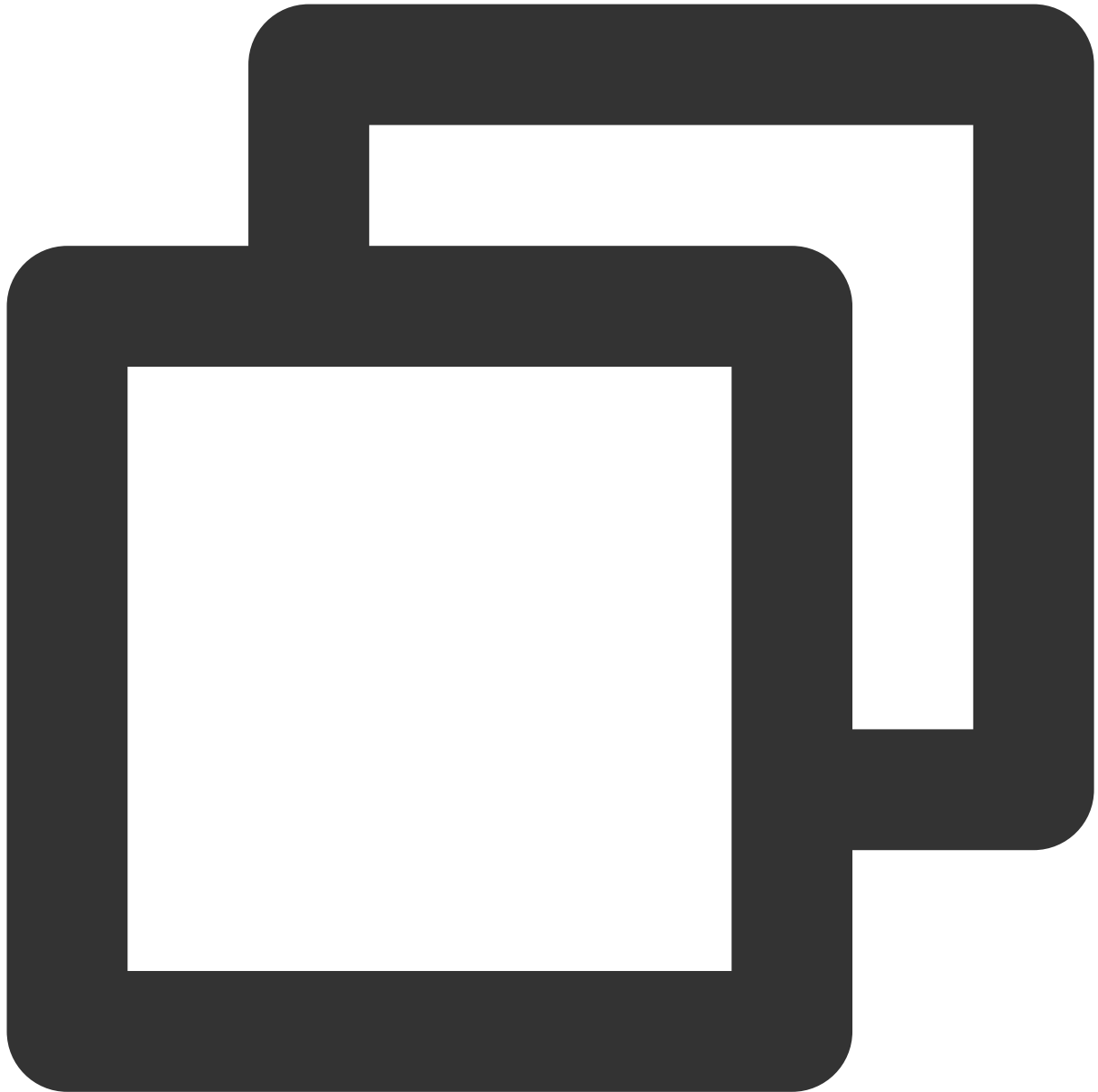
```
void onUserLineBusy(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who is busy.

## onUserJoin

A user joined the call.



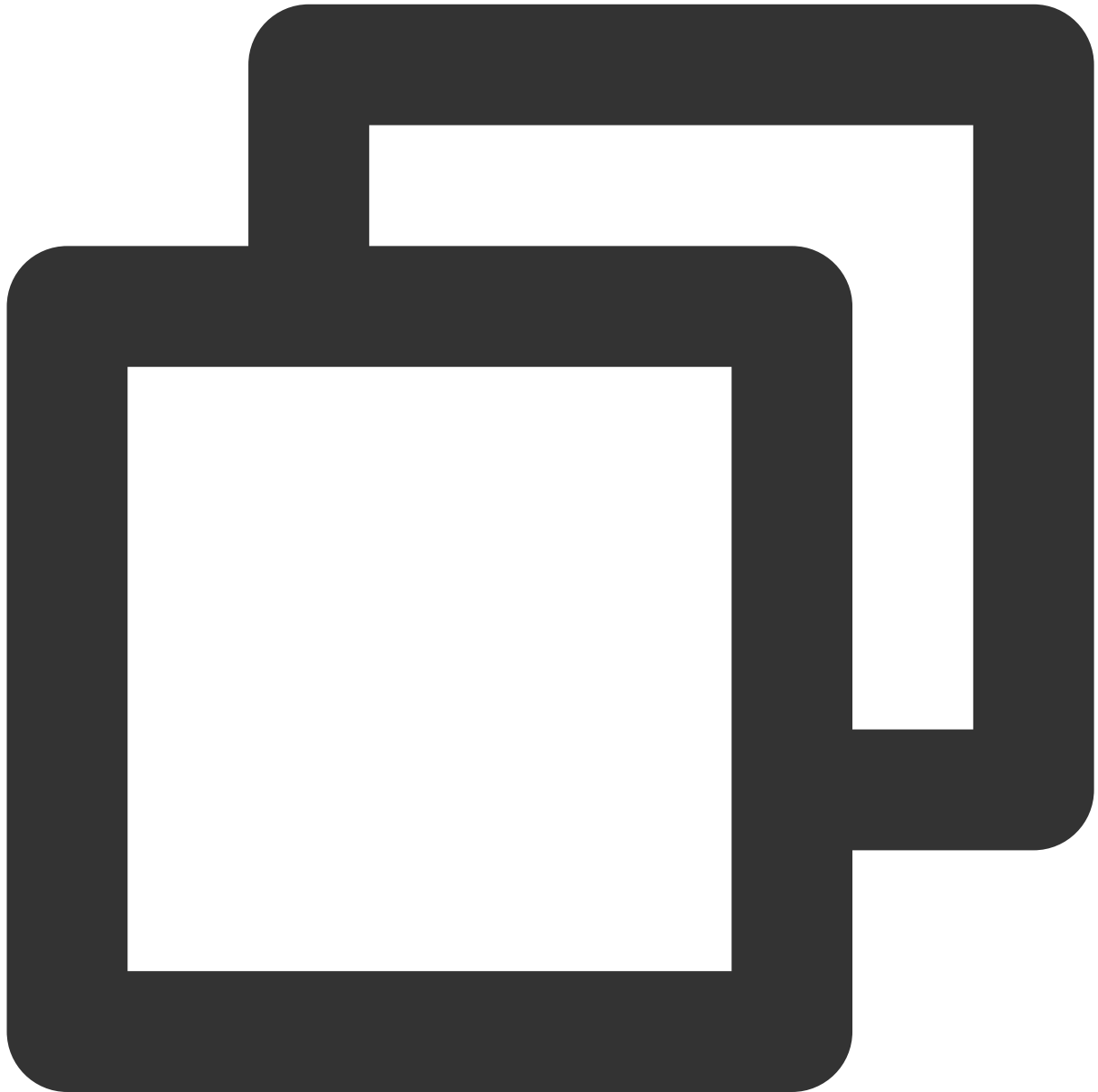
```
void onUserJoin(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The ID of the user who joined the call.

## onUserLeave

A user left the call.



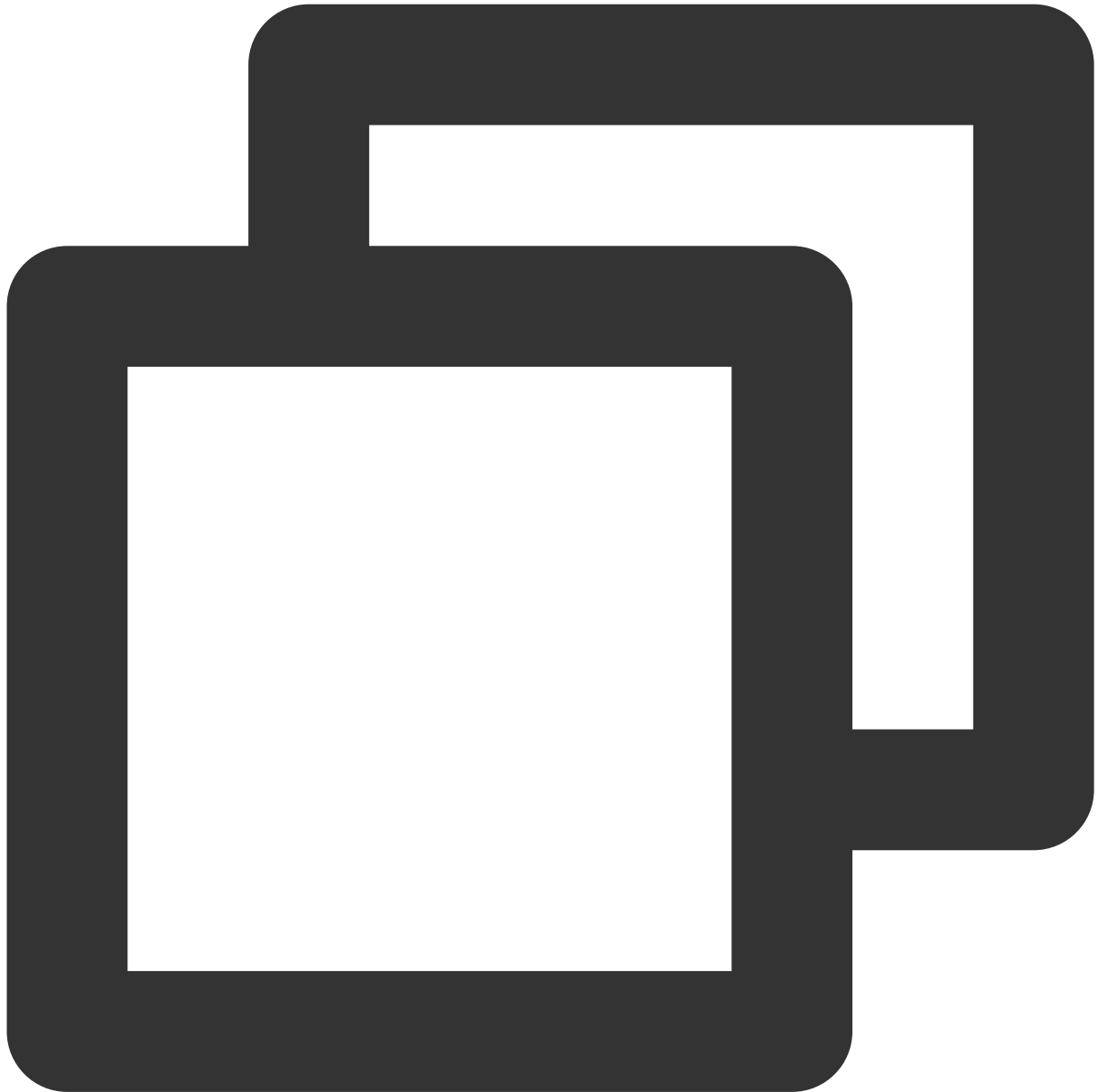
```
void onUserLeave(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The ID of the user who left the call.

## onUserVideoAvailable

Whether a user is sending video.



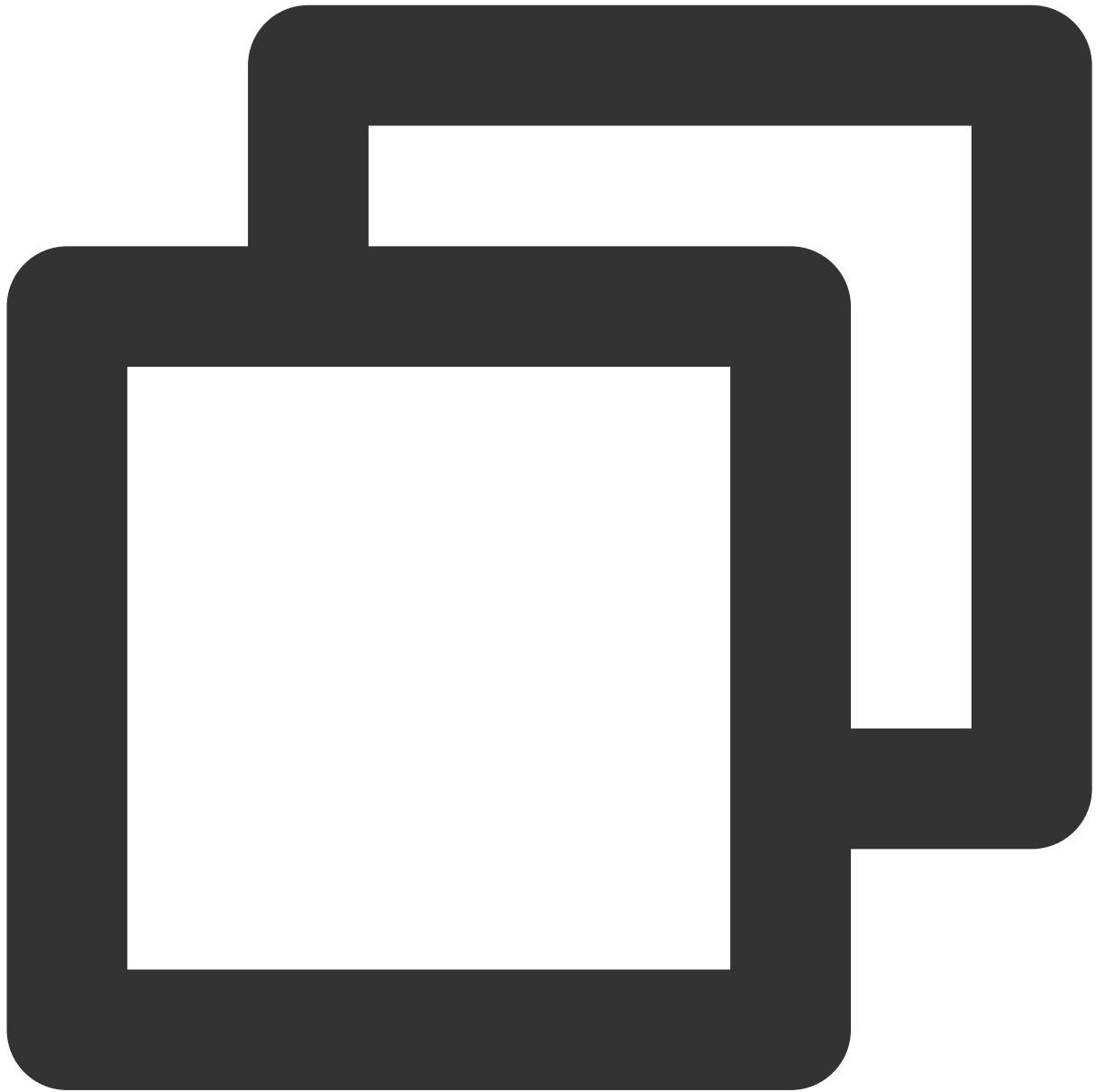
```
void onUserVideoAvailable(String userId, boolean isVideoAvailable);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
isVideoAvailable	boolean	Whether the user has video.

## onUserAudioAvailable

Whether a user is sending audio.



```
void onUserAudioAvailable(String userId, boolean isAudioAvailable);
```

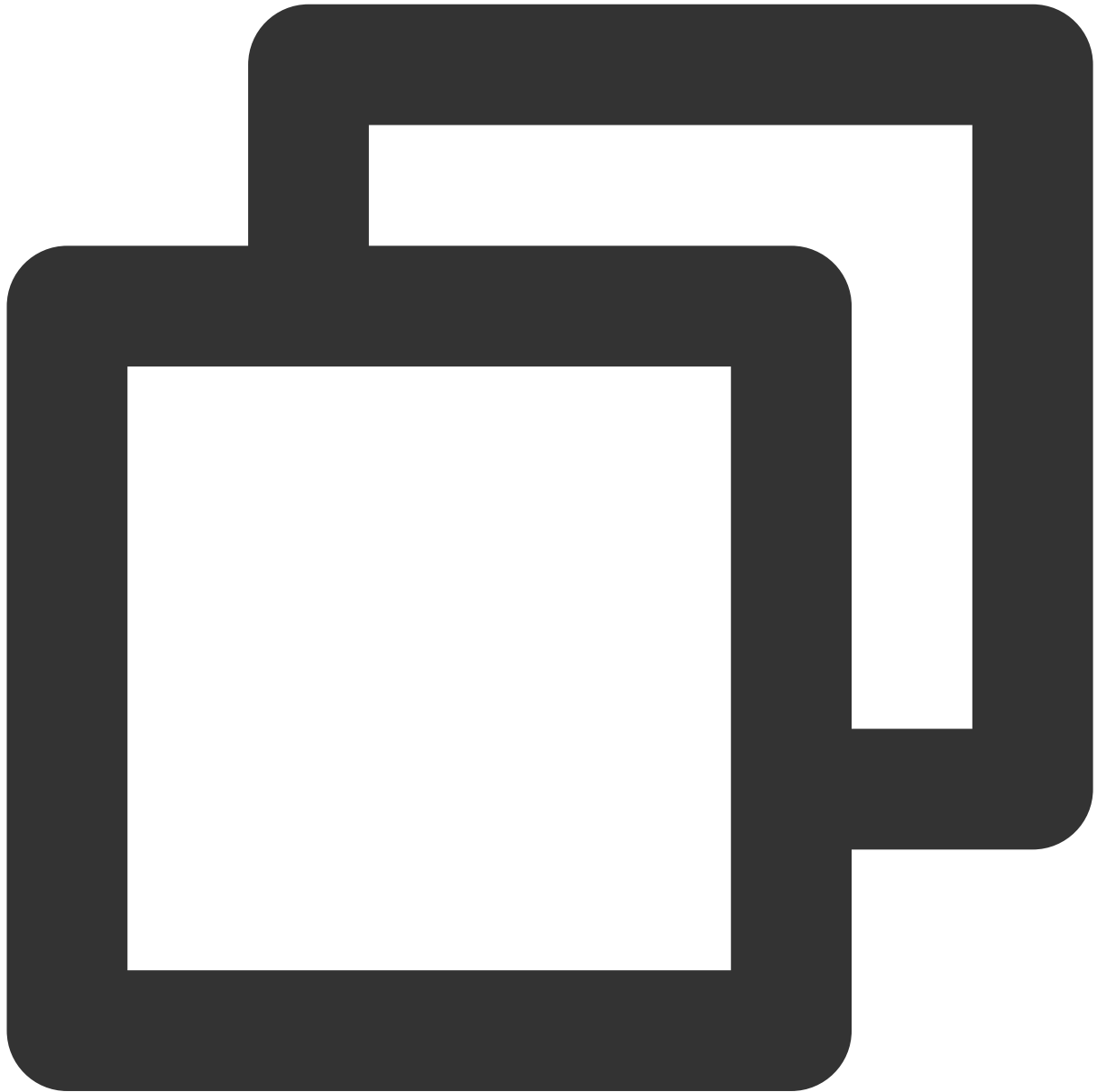
The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.

isAudioAvailable	boolean	Whether the user has audio.
------------------	---------	-----------------------------

## onUserVoiceVolumeChanged

The volumes of all users.



```
void onUserVoiceVolumeChanged(Map<String, Integer> volumeMap);
```

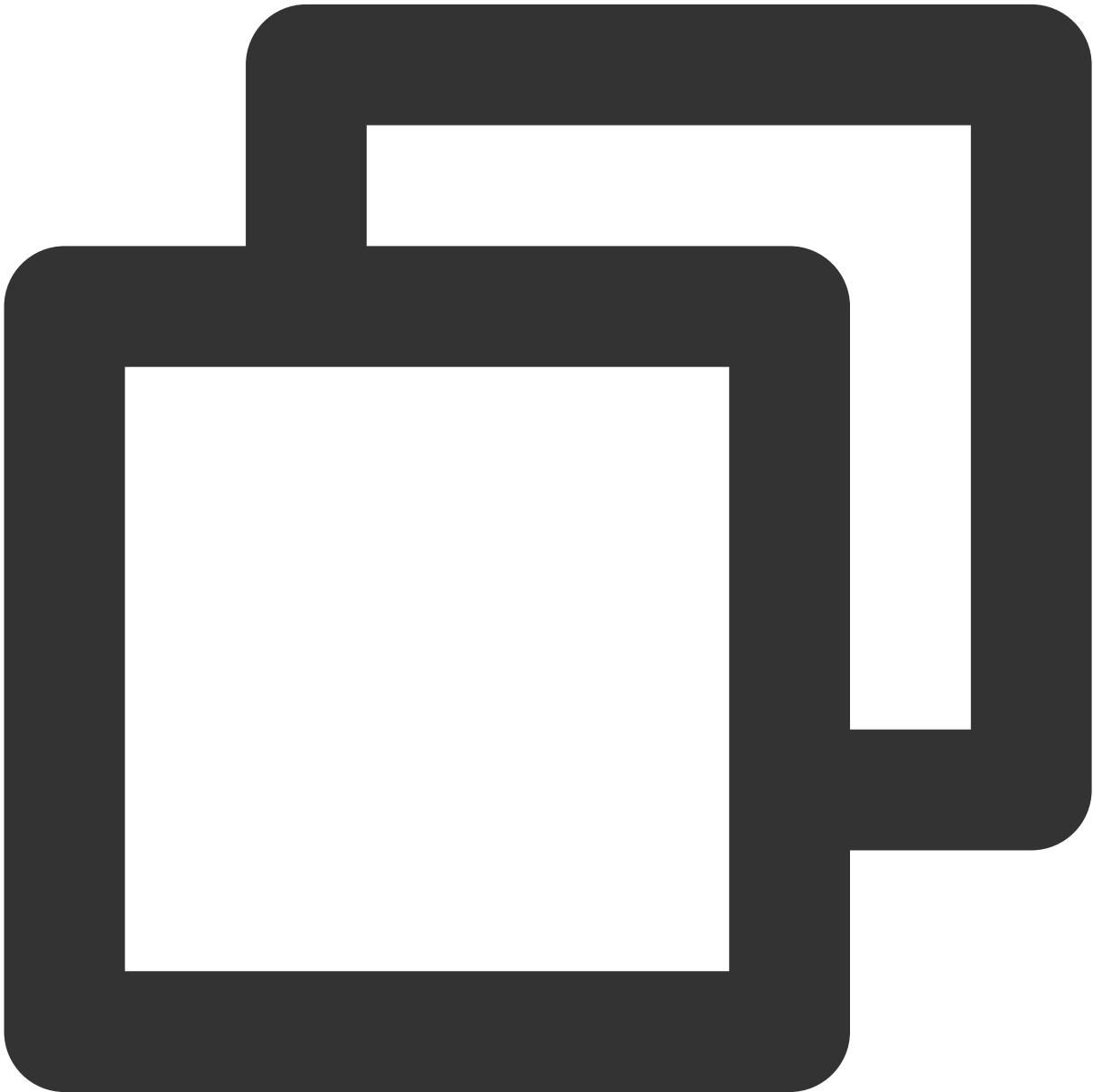
The parameters are described below:

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

volumeMap	Map<String, Integer>	The volume table, which includes the volume of each user ( <code>userId</code> ). Value range: 0-100.
-----------	----------------------	--

**onUserNetworkQualityChanged**

The network quality of all users.



```
void onUserNetworkQualityChanged(List<TUICallDefine.NetworkQualityInfo> networkQual
```

The parameters are described below:

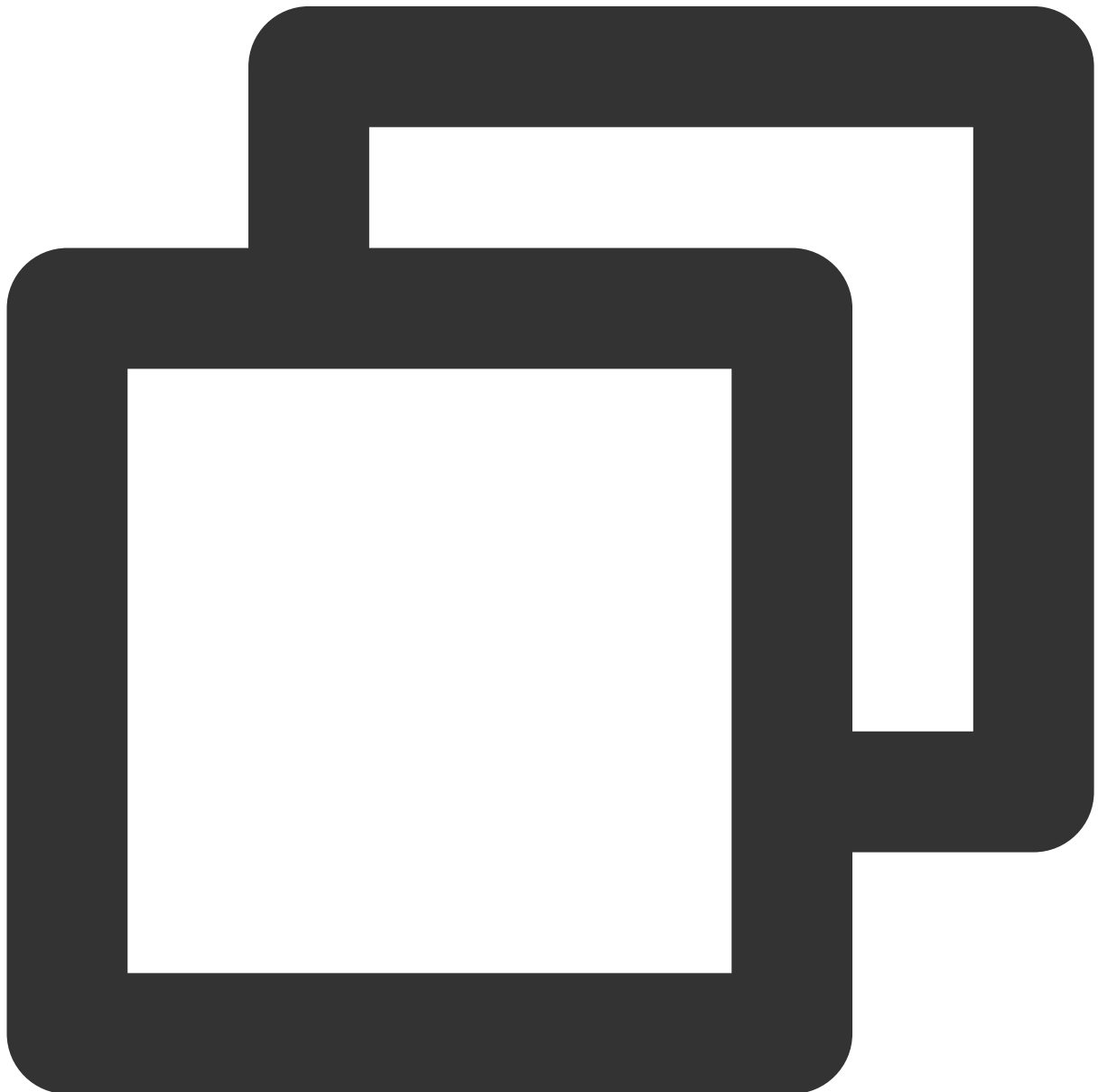
--	--	--



Parameter	Type	Description
networkQualityList	List	The current network conditions for all users ( <code>userId</code> ).

## onKickedOffline

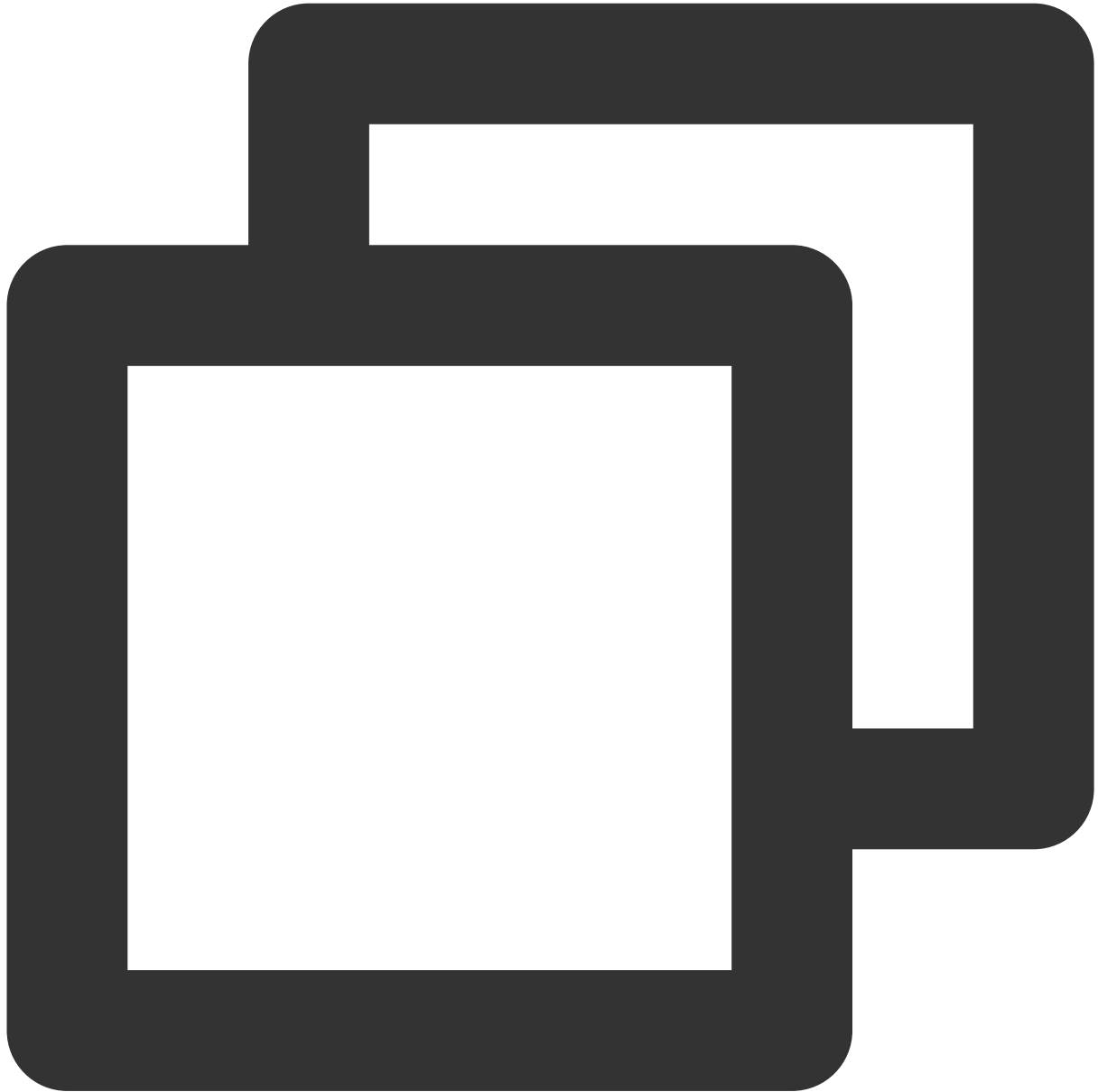
The current user was kicked offline : At this time, you can prompt the user with a UI message and then invoke `init` again.



```
void onKickedOffline();
```

## onUserSigExpired

The user sig is expired : At this time, you need to generate a new `userSig` , and then invoke `init` again.



```
void onUserSigExpired();
```

# Type Definition

Last updated : 2023-09-19 15:29:26

## Common structures

### TUICallDefine

Type	Description
<a href="#">CallParams</a>	An additional parameter.
<a href="#">OfflinePushInfo</a>	Offline push vendor configuration information.

### TUICommonDefine

Type	Description
<a href="#">RoomId</a>	Room ID for audio and video in a call.
<a href="#">NetworkQualityInfo</a>	Network quality information
<a href="#">VideoRenderParams</a>	Video render parameters
<a href="#">VideoEncoderParams</a>	Video encoding parameters

## Enum definition

### TUICallDefine

Type	Description
<a href="#">MediaType</a>	Media type in a call
<a href="#">Role</a>	Roles of individuals in a call.
<a href="#">Status</a>	The call status
<a href="#">Scene</a>	The call scene
<a href="#">IOSOfflinePushType</a>	iOS offline push type

### TUICommonDefine

Type	Description

<a href="#">AudioPlaybackDevice</a>	Audio route
<a href="#">Camera</a>	Camera type
<a href="#">NetworkQuality</a>	Network quality
<a href="#">FillMode</a>	Video image fill mode
<a href="#">Rotation</a>	Video image rotation direction
<a href="#">ResolutionMode</a>	Video aspect ratio mode
<a href="#">Resolution</a>	Video resolution

## CallParams

Call params

Value	Type	Description
offlinePushInfo	<a href="#">OfflinePushInfo</a>	Offline push vendor configuration information.
timeout	int	Call timeout period, default: 30s, unit: seconds.
userData	String	An additional parameter. Callback when the callee receives <a href="#">onCallReceived</a>

## OfflinePushInfo

Offline push vendor configuration information, please refer to : [Offline call push](#)

Value	Type	Description
title	String	offlinepush notification title
desc	String	offlinepush notification description
ignoreIOSBadge	boolean	Ignore badge count for offline push (only for iOS), if set to true, the message will not increase the unread count of the app icon on the iOS receiver's side.
iOSSound	String	Offline push sound setting (only for iOS). When <code>sound = IOS_OFFLINE_PUSH_NO_SOUND</code> , there will be no sound played when the message is received. When <code>sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND</code> , the system sound will be played when the message is received. If you want to customize the iOSSound, you need to link

		the audio file into the Xcode project first, and then set the audio file name (with extension) to the <code>iOSSound</code> .
<code>androidSound</code>	String	Offline push sound setting (only for Android, supported by IMSDK 6.1 and above). Only Huawei and Google phones support setting sound prompts. For Xiaomi phones, please refer to: <a href="#">Xiaomi custom ringtones</a> . In addition, for Google phones, in order to set sound prompts for FCM push on Android 8.0 and above systems, you must call <code>setAndroidFCMChannelID</code> to set the <code>channelID</code> for it to take effect.
<code>androidOPPOChannelID</code>	String	Set the channel ID for OPPO phones with Android 8.0 and above systems.
<code>androidVIVOClassification</code>	int	Classification of VIVO push messages (deprecated interface, VIVO push service will optimize message classification rules on April 3, 2023. It is recommended to use <code>setAndroidVIVOCategory</code> to set the message category). 0: Operational messages, 1: System messages. The default value is 1.
<code>androidXiaoMiChannelID</code>	String	Set the channel ID for Xiaomi phones with Android 8.0 and above systems.
<code>androidFCMChannelID</code>	String	Set the channel ID for google phones with Android 8.0 and above systems.
<code>androidHuaWeiCategory</code>	String	Classification of Huawei push messages, please refer to: <a href="#">Huawei message classification standard</a> .
<code>isDisablePush</code>	boolean	Whether to turn off push notifications (default is on).
<code>iOSPushType</code>	<a href="#">IOSOfflinePushType</a>	iOS offline push type, default is APNs

## RoomId

Room ID for audio and video in a call.

### Note :

(1) `intRoomId` and `strRoomId` are mutually exclusive. If you choose to use `strRoomId`, `intRoomId` needs to be filled in as 0. If both are filled in, the SDK will prioritize `intRoomId`.

(2) Do not mix `intRoomId` and `strRoomId` because they are not interchangeable. For example, the number 123 and the string "123" represent two completely different rooms.

Value	Type	Description
<code>intRoomId</code>	int	Numeric room ID. <b>range</b> : 1 - 2147483647( $2^{31}-1$ )
<code>strRoomId</code>	String	String room ID. <b>range</b> : Limited to 64 bytes in length. The supported character set range is as follows (a total of 89 Lowercase and uppercase English letters. (a-zA-Z) ; Number (0-9) ; Spaces 、 ! 、 # 、 \$ 、 % 、 & 、 ( 、 ) 、 + 、 - 、 : 、 ; 、 < °

### Note :

Currently, string room number is only supported on Android and iOS platforms. Support for other platforms such as Web, Mini Programs, Flutter, and Uniapp will be available in the future. Please stay tuned!

## NetworkQualityInfo

User network quality information

Value	Type	Description
<code>userId</code>	String	user ID
<code>quality</code>	<a href="#">NetworkQuality</a>	network quality

## VideoRenderParams

Video render parameters

Value	Type	Description
<code>fillMode</code>	<a href="#">VideoRenderParams.FillMode</a>	Video image fill mode
<code>rotation</code>	<a href="#">VideoRenderParams.Rotation</a>	Video image rotation direction

## VideoEncoderParams

Video encoding parameters

Value	Type	Description
<code>resolution</code>	<a href="#">VideoEncoderParams.Resolution</a>	Video resolution

resolutionMode

[VideoEncoderParams.ResolutionMode](#)

Video aspect ratio mode

## MediaType

Call media type

Value	Type	Description
Unknown	0	Unknown
Audio	1	Audio call
Video	2	Video call

## Role

Call role

Value	Type	Description
None	0	Unknown
Caller	1	Caller (inviter)
Called	2	Callee (invitee)

## Status

Call status

Value	Type	Description
None	0	Unknown
Waiting	1	The call is currently waiting
Accept	2	The call has been accepted

## Scene

Call scene

Value	Type	Description
GROUP_CALL	0	Group call
MULTI_CALL	1	Anonymous group calling (not supported at this moment, please stay tuned).

SINGLE_CALL	2	one to one call
-------------	---	-----------------

## IOSOfflinePushType

iOS offline push type

Type	Value	Description
APNs	0	APNs
VoIP	1	VoIP

## AudioPlaybackDevice

Audio route

Type	Value	Description
Earpiece	0	Earpiece
Speakerphone	1	Speakerphone

## Camera

Front/Back camera

Type	Value	Description
Front	0	Front camera
Back	1	Back camera

## NetworkQuality

Network quality

Type	Value	Description
Unknown	0	Unknown
Excellent	1	Excellent
Good	2	Good
Poor	3	Poor
Bad	4	Bad
Vbad	5	Vbad



Down	6	Down
------	---	------

## FillMode

If the aspect ratio of the video display area is not equal to that of the video image, you need to specify the fill mode:

Type	Value	Description
Fill	0	Fill mode: the video image will be centered and scaled to fill the entire display area, where parts that exceed the area will be cropped. The displayed image may be incomplete in this mode.
Fit	1	Fit mode: the video image will be scaled based on its long side to fit the display area, where the short side will be filled with black bars. The displayed image is complete in this mode, but there may be black bars.

## Rotation

We provides rotation angle setting APIs for local and remote images. The following rotation angles are all clockwise.

Type	Value	Description
Rotation_0	0	No rotation
Rotation_90	1	Clockwise rotation by 90 degrees
Rotation_180	2	Clockwise rotation by 180 degrees
Rotation_270	3	Clockwise rotation by 0 degrees

## ResolutionMode

Video aspect ratio mode

Type	Value	Description
Landscape	0	Landscape resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360.
Portrait	1	Portrait resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640.

## Resolution

## Video resolution

Type	Value	Description
Resolution_640_360	108	Aspect ratio: 16:9 ; resolution: 640x360 ; recommended bitrate: 500kbps
Resolution_640_480	62	Aspect ratio: 4:3 ; resolution: 640x480 ; recommended bitrate: 600kbps
Resolution_960_540	110	Aspect ratio: 16:9 ; resolution: 960x540 ; recommended bitrate: 850kbps
Resolution_960_720	64	Aspect ratio: 4:3 ; resolution: 960x720 ; recommended bitrate: 1000kbps
Resolution_1280_720	112	Aspect ratio: 16:9 ; resolution: 1280x720 ; recommended bitrate: 1200kbps
Resolution_1920_1080	114	Aspect ratio: 16:9 ; resolution: 1920x1080 ; recommended bitrate: 2000kbps

# iOS

## API Overview

Last updated : 2023-12-18 17:46:13

### TUICallKit (UI Included)

TUICallKit is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat.

API	Description
<a href="#">createInstance</a>	Creates a <code>TUICallKit</code> instance (singleton mode).
<a href="#">setSelfInfo</a>	Sets the user nickname and profile photo.
<a href="#">call</a>	Makes a one-to-one call.
<a href="#">call</a>	Makes a one-to-one call, Support for custom room ID, call timeout, offline push content, etc
<a href="#">groupCall</a>	Makes a group call.
<a href="#">groupCall</a>	Makes a group call, Support for custom room ID, call timeout, offline push content, etc
<a href="#">joinInGroupCall</a>	Joins a group call.
<a href="#">setCallingBell</a>	Sets the ringtone.
<a href="#">enableMuteMode</a>	Sets whether to turn on the mute mode.
<a href="#">enableFloatWindow</a>	Sets whether to enable floating windows.

### TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description
<a href="#">createInstance</a>	Creates a <code>TUICallEngine</code> instance (singleton).

<a href="#">destroyInstance</a>	Terminates a <code>TUICallEngine</code> instance (singleton).
<a href="#">init</a>	Authenticates the basic audio/video call capabilities.
<a href="#">addObserver</a>	Registers an event listener.
<a href="#">removeObserver</a>	Unregisters an event listener.
<a href="#">call</a>	Makes a one-to-one call.
<a href="#">groupCall</a>	Makes a group call.
<a href="#">accept</a>	Answers a call.
<a href="#">reject</a>	Declines a call.
<a href="#">hangup</a>	Ends a call.
<a href="#">ignore</a>	Ignores a call.
<a href="#">inviteUser</a>	Invites users to the current group call.
<a href="#">joinInGroupCall</a>	Joins a group call.
<a href="#">switchCallMediaType</a>	Switches the call media type, such as from video call to audio call.
<a href="#">startRemoteView</a>	Subscribes to the video stream of a remote user.
<a href="#">stopRemoteView</a>	Unsubscribes from the video stream of a remote user.
<a href="#">openCamera</a>	Turns the camera on.
<a href="#">closeCamera</a>	Turns the camera off.
<a href="#">switchCamera</a>	Switches the camera.
<a href="#">openMicrophone</a>	Enables the mic.
<a href="#">closeMicrophone</a>	Disables the mic.
<a href="#">selectAudioPlaybackDevice</a>	Selects the audio playback device (receiver/speaker).
<a href="#">setSelfInfo</a>	Sets the user nickname and profile photo.
<a href="#">enableMultiDeviceAbility</a>	Sets whether to enable multi-device login for <code>TUICallEngine</code> (supported by the premium package).
<a href="#">setVideoRenderParams</a>	Set the rendering mode of video image.
<a href="#">setVideoEncoderParams</a>	Set the encoding parameters of video encoder.

<a href="#">getTRTCCloudInstance</a>	Advanced features.
<a href="#">setBeautyLevel</a>	Set beauty level, support turning off default beauty.

## TUICallObserver

`TUICallObserver` is the callback class of `TUICallEngine`. You can use it to listen for events.

API	Description
<a href="#">onError</a>	An error occurred during the call.
<a href="#">onCallReceived</a>	A call was received.
<a href="#">onCallCancelled</a>	The call was canceled.
<a href="#">onCallBegin</a>	The call was connected.
<a href="#">onCallEnd</a>	The call ended.
<a href="#">onCallMediaTypeChanged</a>	The call type changed.
<a href="#">onUserReject</a>	A user declined the call.
<a href="#">onUserNoResponse</a>	A user didn't respond.
<a href="#">onUserLineBusy</a>	A user was busy.
<a href="#">onUserJoin</a>	A user joined the call.
<a href="#">onUserLeave</a>	A user left the call.
<a href="#">onUserVideoAvailable</a>	Whether a user had a video stream.
<a href="#">onUserAudioAvailable</a>	Whether a user had an audio stream.
<a href="#">onUserVoiceVolumeChanged</a>	The volume levels of all users.
<a href="#">onUserNetworkQualityChanged</a>	The network quality of all users.
<a href="#">onKickedOffline</a>	The current user was kicked offline.
<a href="#">onUserSigExpired</a>	The user sig is expired.

## Definitions of Key Types

--	--

API	Description
<a href="#">TUICallMediaType</a>	The call type. Enumeration: Video call and audio call.
<a href="#">TUICallRole</a>	The call role. Enumeration: Caller and callee.
<a href="#">TUICallStatus</a>	The call status. Enumeration: Idle, waiting, and answering.
<a href="#">TUIRoomId</a>	The room ID, which can be a number or string.
<a href="#">TUICallCamera</a>	The camera type. Enumeration: Front camera and rear camera.
<a href="#">TUIAudioPlaybackDevice</a>	The audio playback device type. Enumeration: Speaker and receiver.
<a href="#">TUINetworkQualityInfo</a>	The current network quality.

# TUICallKit

Last updated : 2023-06-21 16:42:24

## TUICallKit APIs

`TUICallKit` is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat. For directions on integration, see [Integrating TUICallKit](#).

## Overview

API	Description
<a href="#">createInstance</a>	Creates a <code>TUICallKit</code> instance (singleton mode).
<a href="#">setSelfInfo</a>	Sets the alias and profile photo.
<a href="#">call</a>	Makes a one-to-one call.
<a href="#">call</a>	Makes a one-to-one call, Support for custom room ID, call timeout, offline push content, etc
<a href="#">groupCall</a>	Makes a group call.
<a href="#">groupCall</a>	Makes a group call, Support for custom room ID, call timeout, offline push content, etc
<a href="#">joinInGroupCall</a>	Joins a group call.
<a href="#">setCallingBell</a>	Sets the ringtone.
<a href="#">enableMuteMode</a>	Sets whether to turn on the mute mode.
<a href="#">enableFloatWindow</a>	Sets whether to enable floating windows.

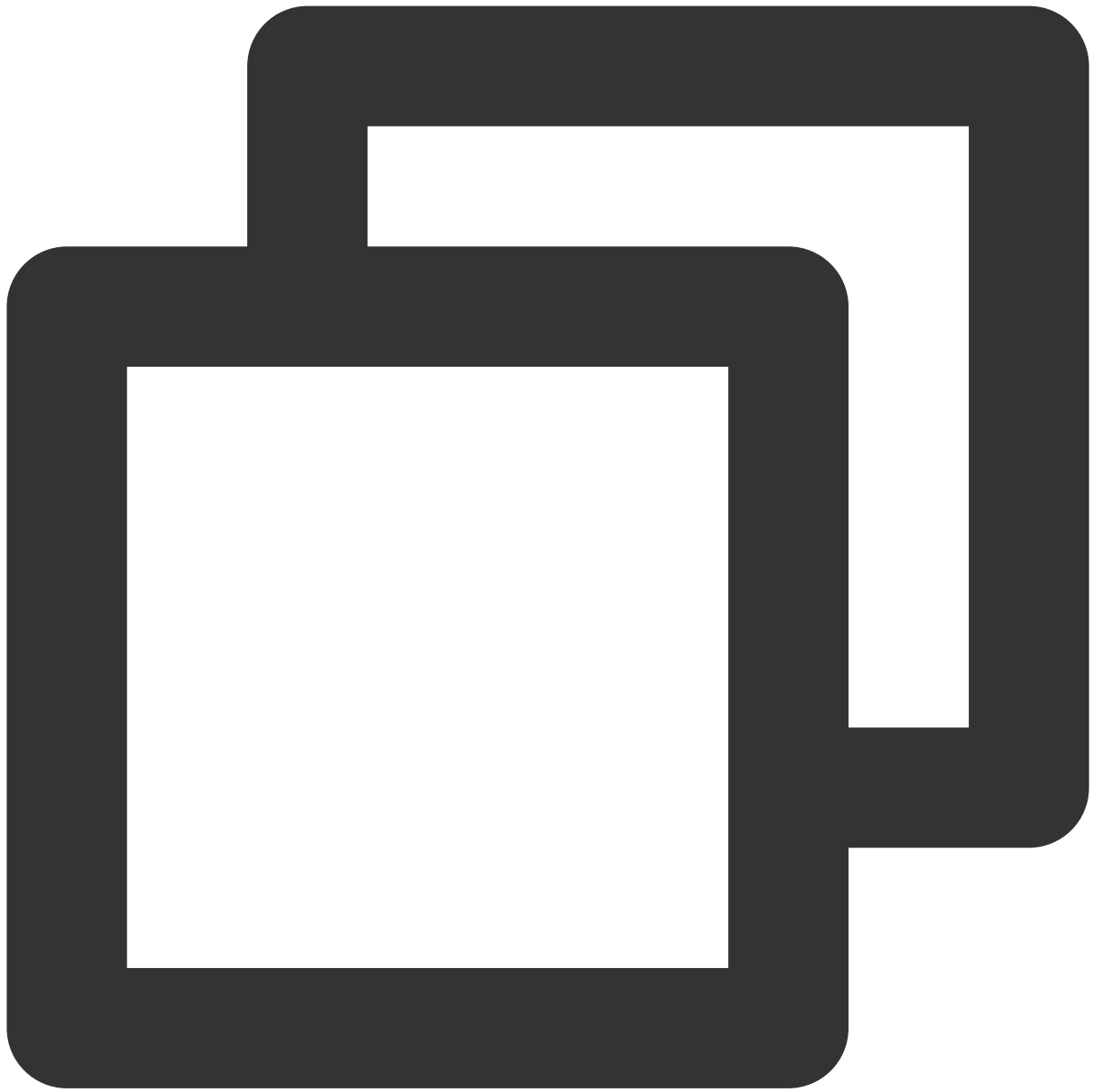
## Details

### createInstance

This API is used to create a `TUICallKit` singleton.

Objective-C

Swift



```
- (instancetype)createInstance;
```





```
public static func createInstance() -> TUICallKit
```

### setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.

Objective-C

Swift



```
- (void)setSelfInfo:(NSString * _Nullable)nickname avatar:(NSString * _Nullable)ava
```



```
public func setSelfInfo(nickname: String, avatar: String, succ:@escaping TUICallSuc
```

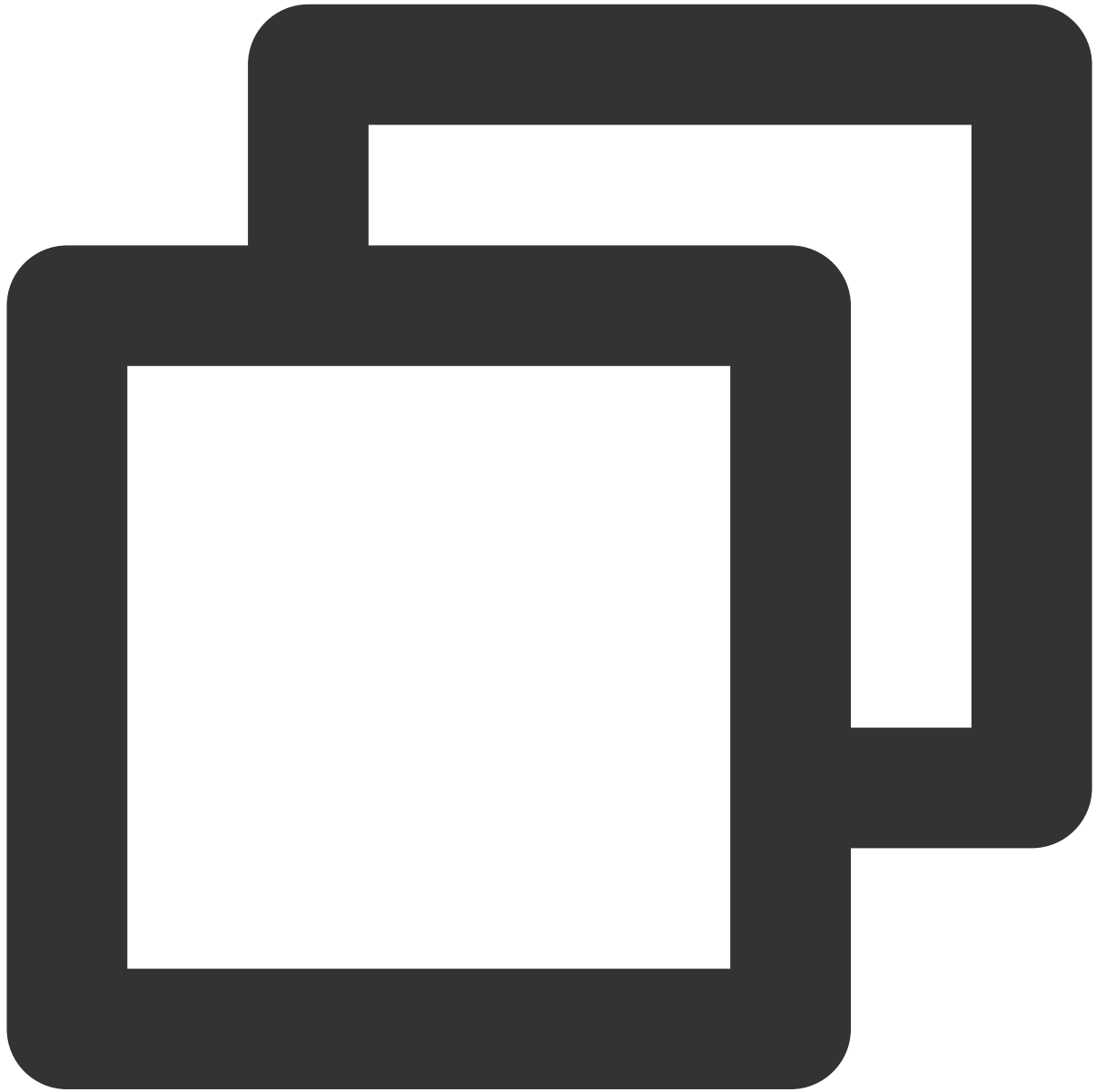
Parameter	Type	Description
nickName	NSString	The alias.
avatar	NSString	The profile photo.

## call

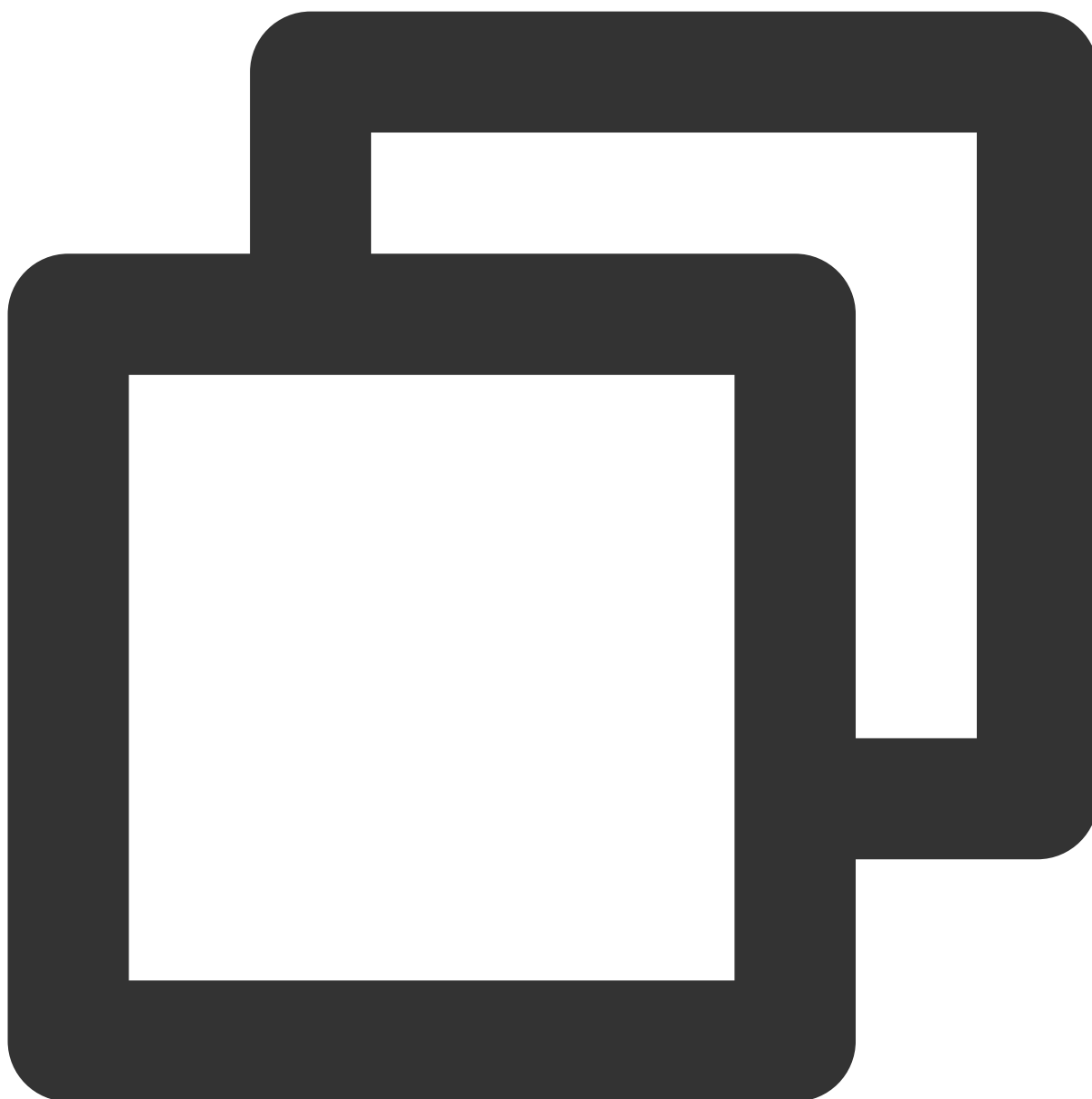
This API is used to make a (one-to-one) call.

Objective-C

Swift



```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType;
```



```
public func call(userId: String, callMediaType: TUICallMediaType)
```

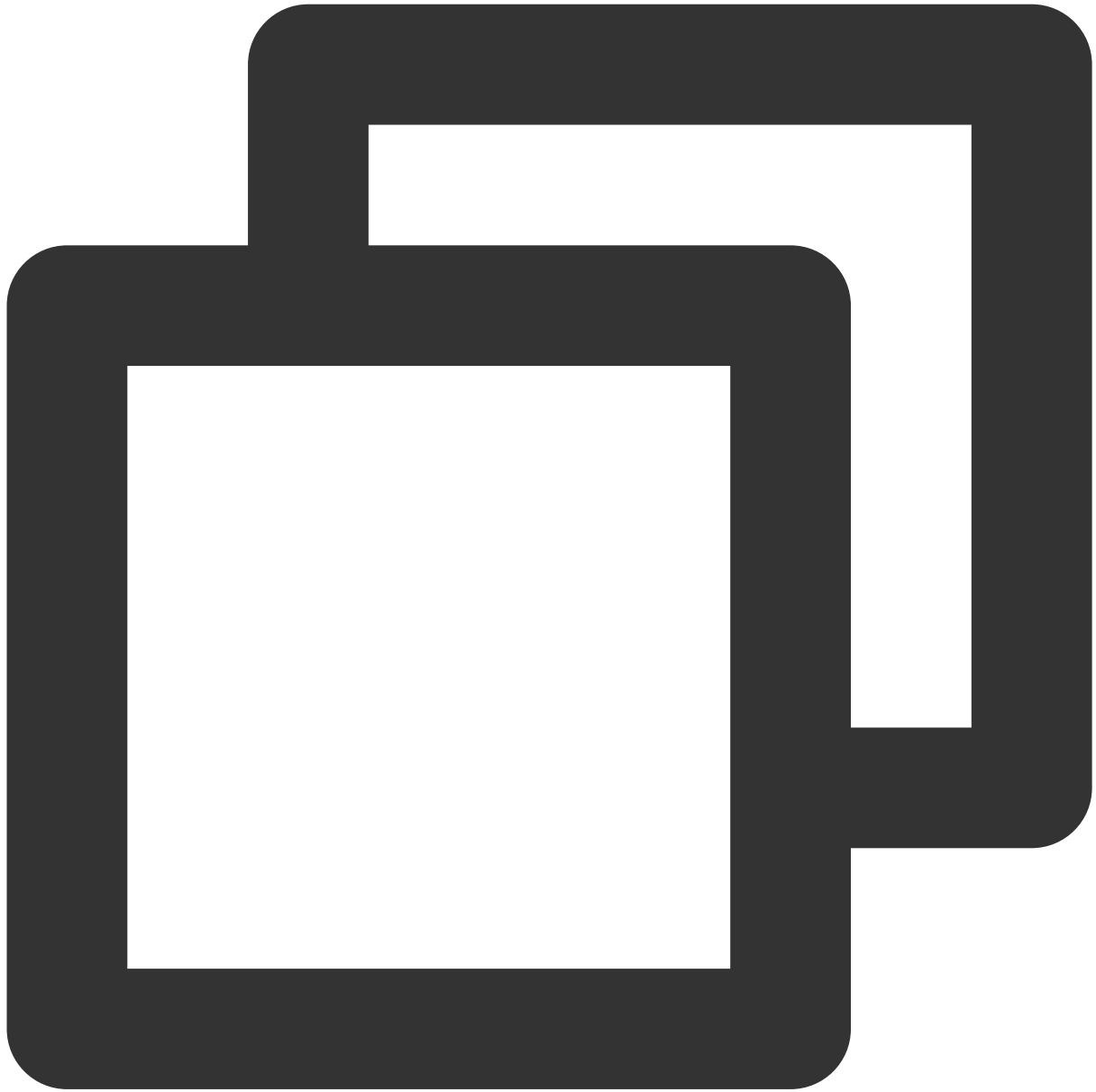
Parameter	Type	Description
userId	NSString	The target user ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.

## call

This API is used to make a (one-to-one) call, Support for custom room ID, call timeout, offline push content, etc.

Objective-C

Swift



```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType param
```



```
public func call(userId: String, callMediaType: TUICallMediaType, params: TUICallPa
succ: @escaping TUICallSucc, fail: @escaping TUICallFail)
```

Parameter	Type	Description
userId	NSString	The target user ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
params	<a href="#">TUICallParams</a>	Call extension parameters, such as roomId, call timeout, offline push info,etc

## groupCall

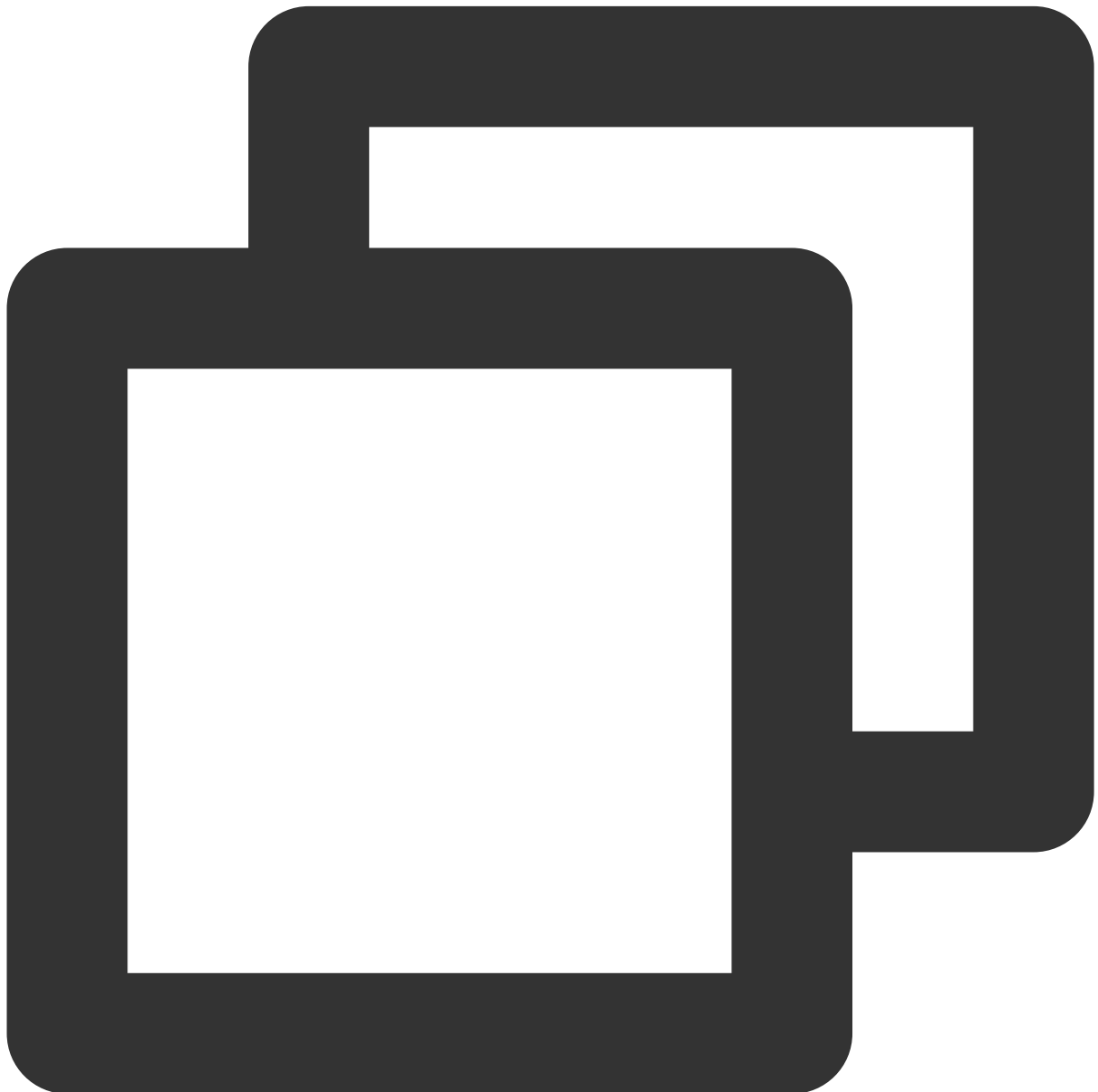
This API is used to make a group call.

### 注意：

Before making a group call, you need to create an IM group first.

Objective-C

Swift



```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray<NSString *> *)userIdList
```





```
public func groupCall(groupId: String, userIdList: [String], callMediaType: TUICall
```

Parameter	Type	Description
groupId	NSString	The group ID.
userIdList	NSArray	The target user IDs.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.

## groupCall

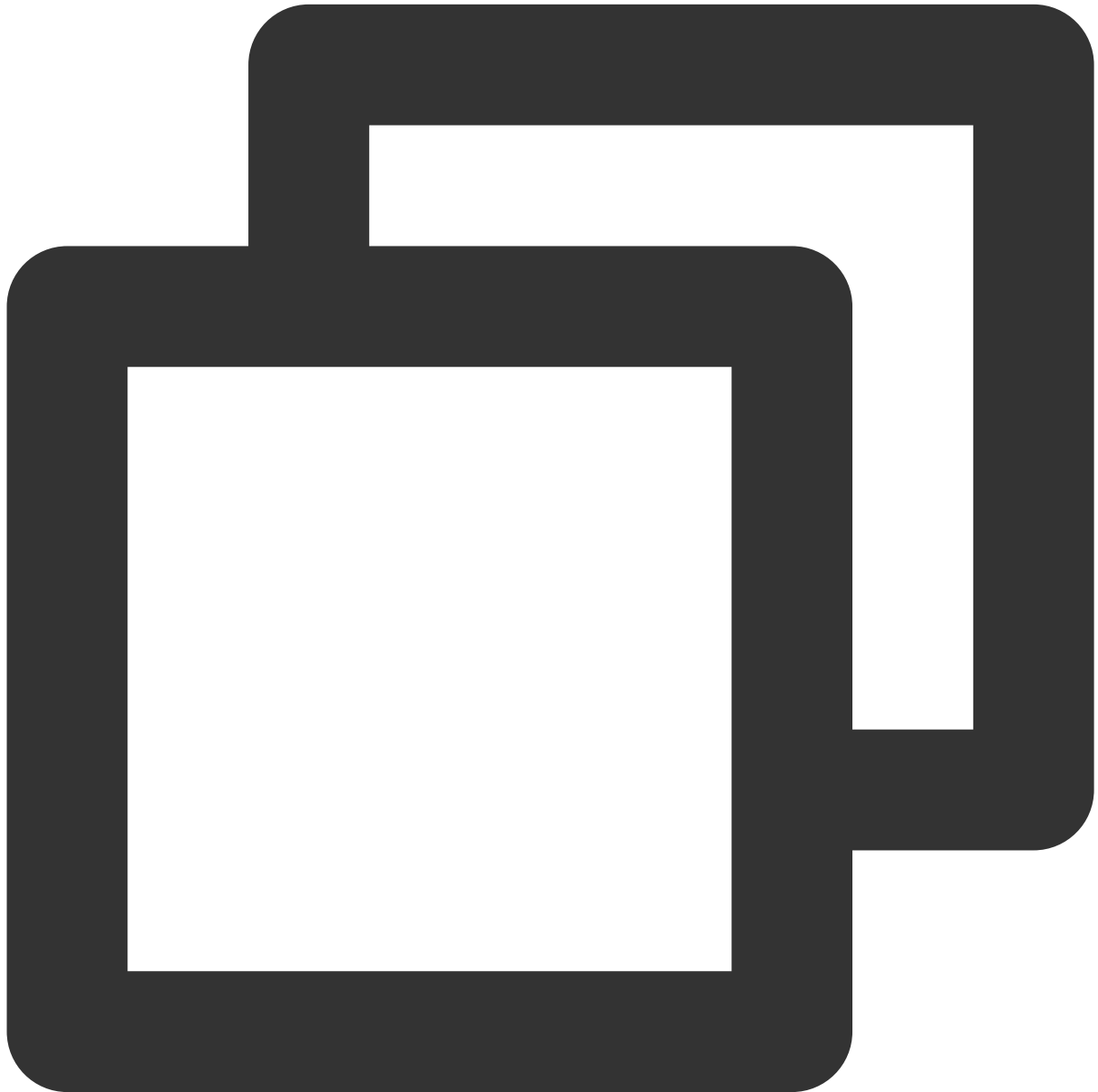
This API is used to make a group call, Support for custom room ID, call timeout, offline push content, etc.

### Notice :

Before making a group call, you need to create an IM group first.

Objective-C

Swift



```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray<NSString *> *)userIdList
```



```
public func groupCall(groupId: String, userIdList: [String], callMediaType: TUICall
```

Parameter	Type	Description
groupId	NSString	The group ID.
userIdList	NSArray	The target user IDs.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
params	<a href="#">TUICallParams</a>	Call extension parameters, such as roomId, call timeout, offline

		push info, etc
--	--	----------------

## joinInGroupCall

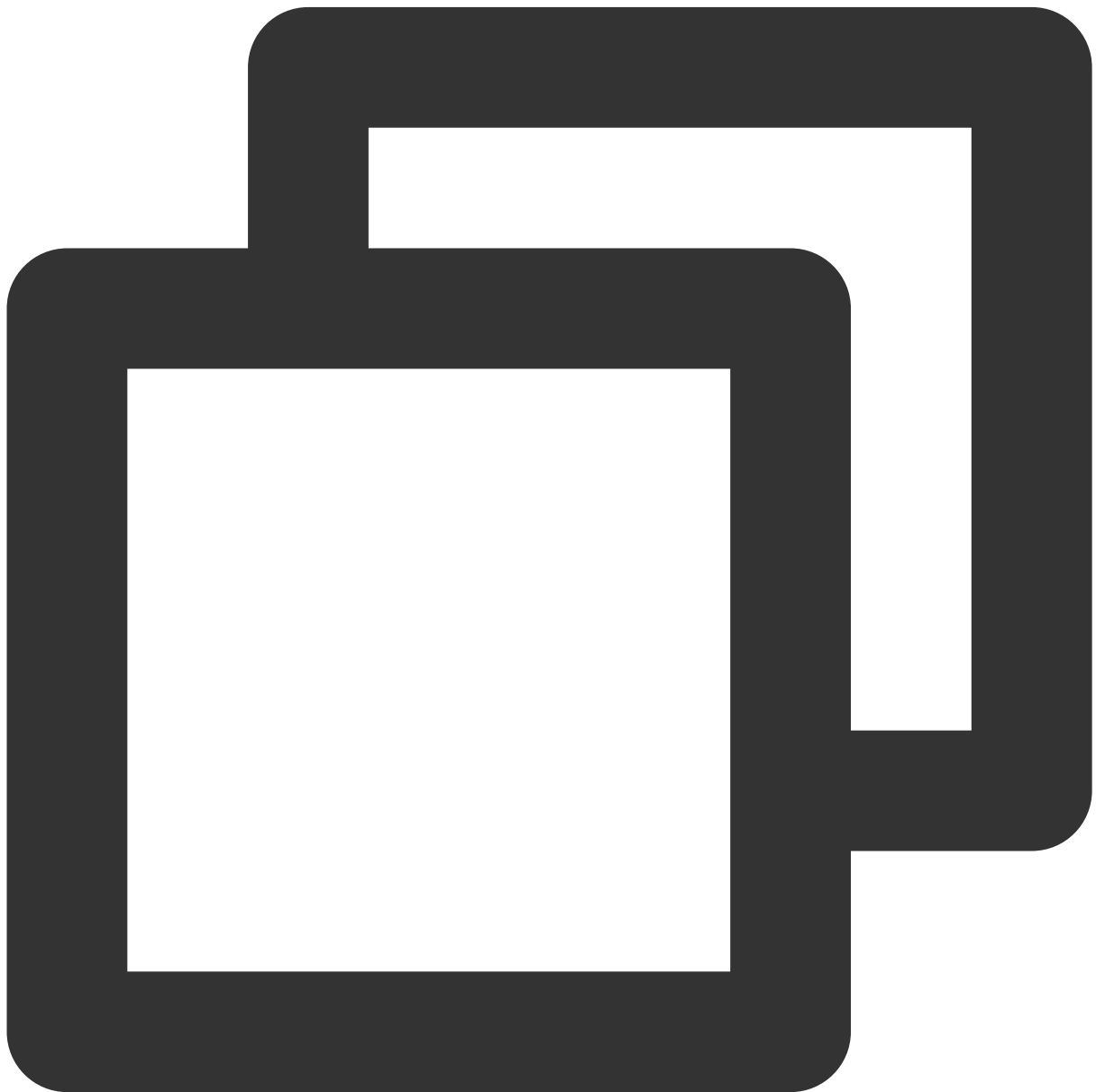
This API is used to join a group call.

### Notice:

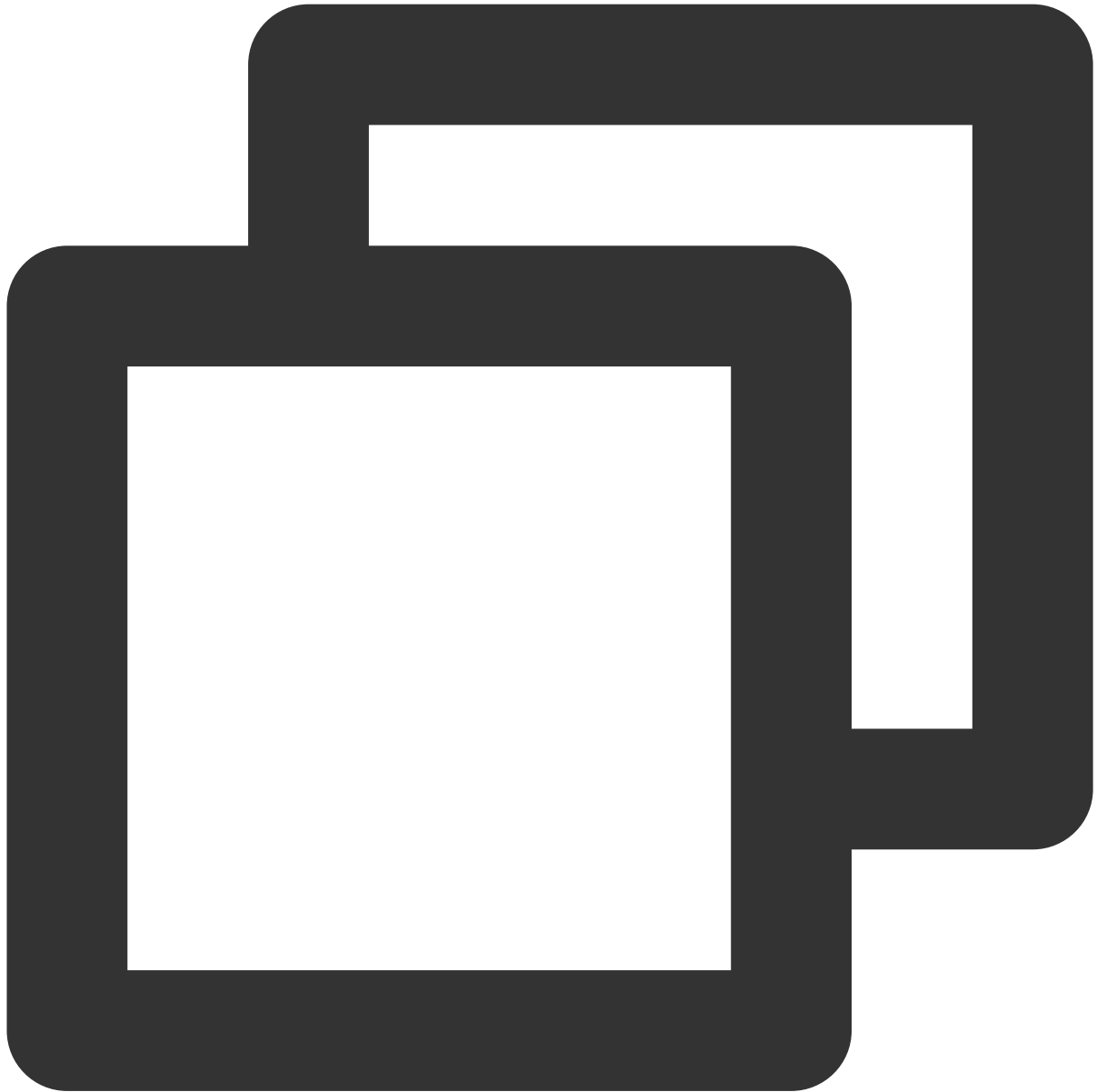
Before joining a group call, you need to create or join an IM group in advance, and there are already users in the group who are in the call.

Objective-C

Swift



```
- (void)joinInGroupCall:(TUIRoomId *)roomId groupId:(NSString *)groupId callMediaTy
```



```
public func joinInGroupCall(roomId: TUIRoomId, groupId: String, callMediaType: TUIC
```

Parameter	Type	Description
roomId	<a href="#">TUIRoomId</a>	The room ID.
groupId	NSString	The group ID.

callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
---------------	----------------------------------	---

## setCallingBell

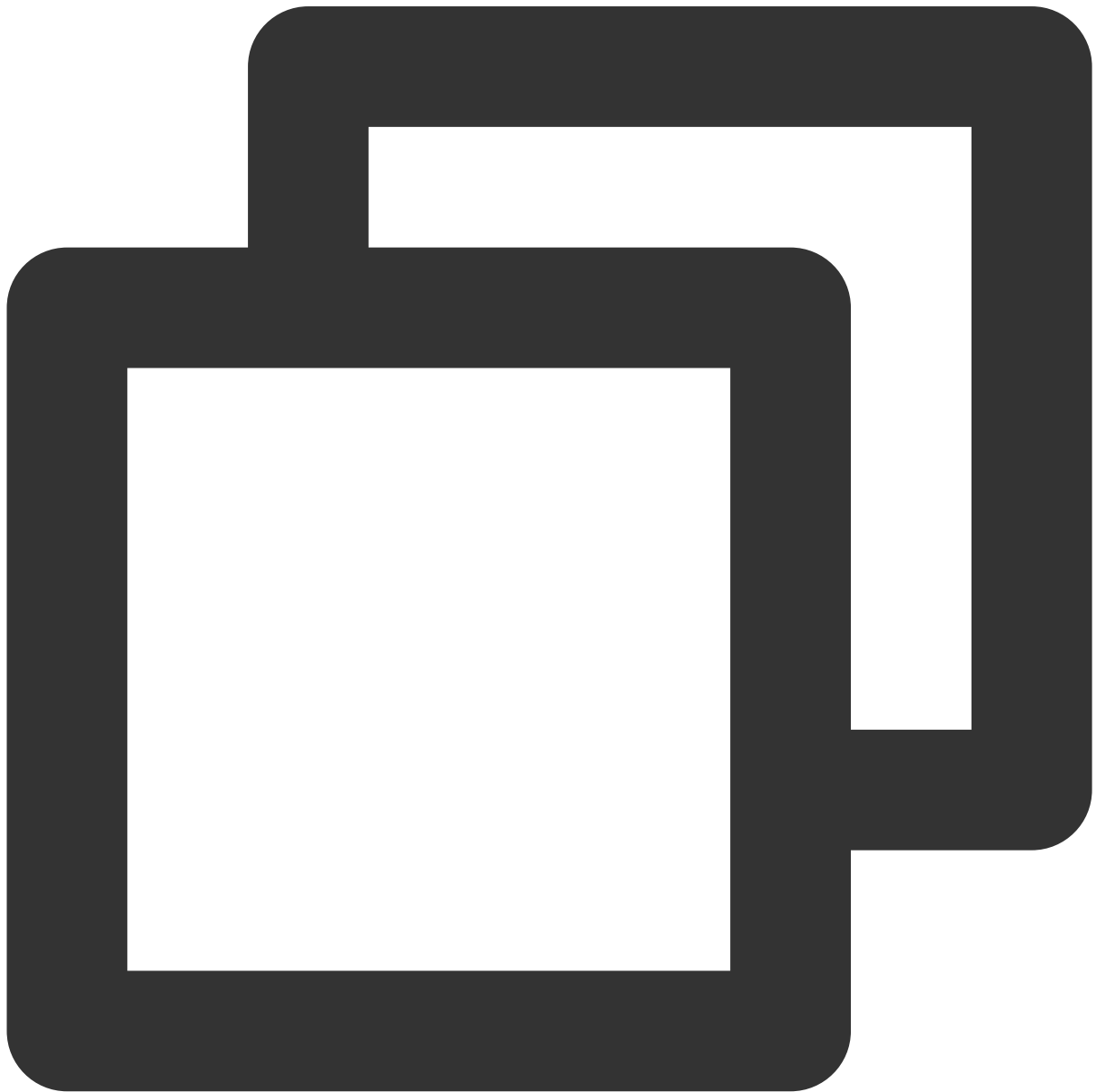
This API is used to set the ringtone. `filePath` must be an accessible local file URL.

The ringtone set is associated with the device and does not change with user.

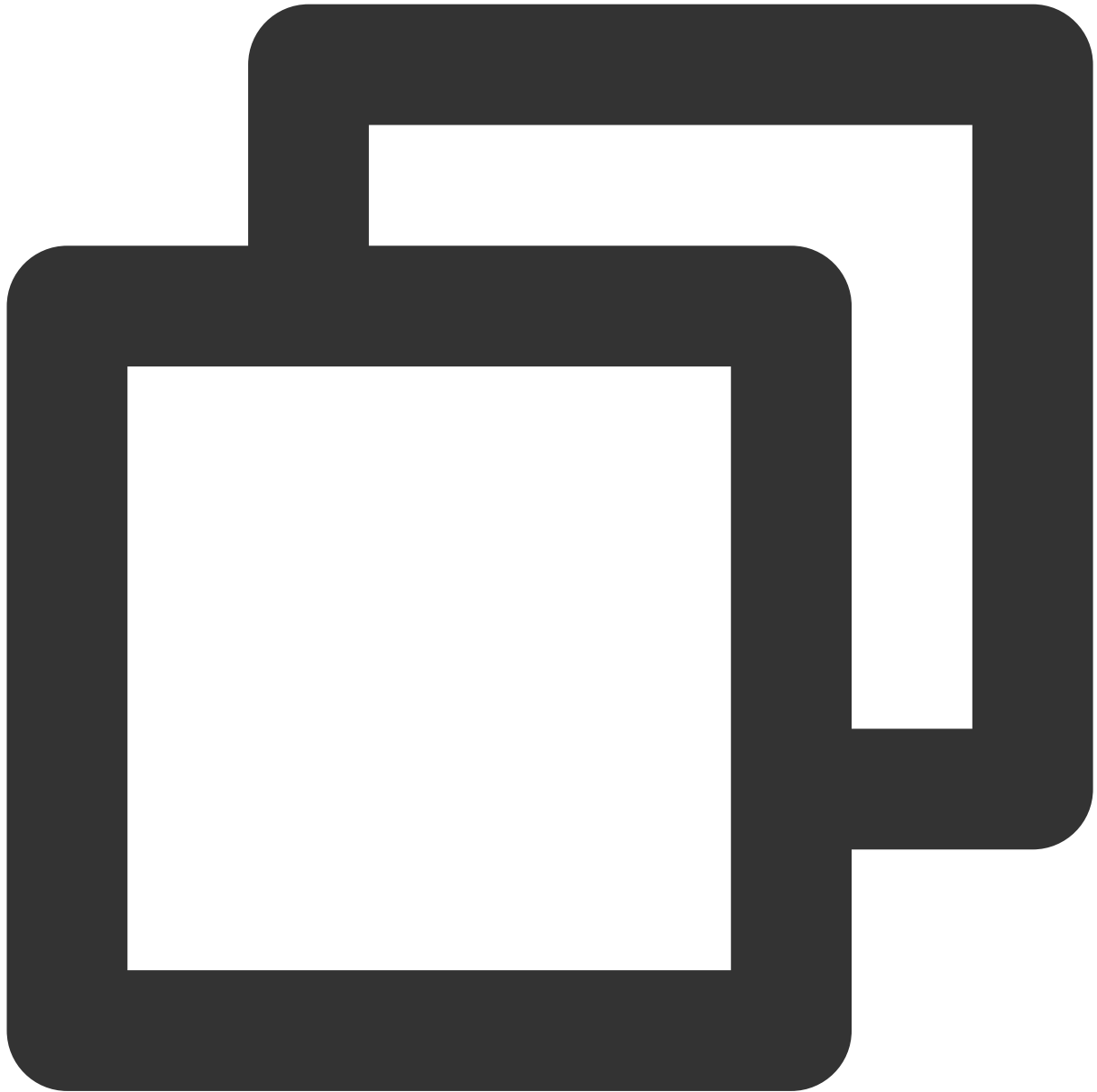
To reset the ringtone, pass in an empty string for `filePath`.

Objective-C

Swift



```
- (void)setCallingBell:(NSString *)filePath;
```



```
public func setCallingBell(filePath: String)
```

## **enableMuteMode**

This API is used to set whether to turn on the mute mode.

Objective-C

Swift



```
- (void)enableMuteMode:(BOOL)enable;
```





```
public func enableMuteMode(enable: Bool)
```

## **enableFloatWindow**

This API is used to set whether to enable floating windows.

The default value is `false`, and the floating window button in the top left corner of the call view is hidden. If it is set to `true`, the button will become visible.

Objective-C

Swift



```
- (void)enableFloatWindow:(BOOL)enable;
```



```
public func enableFloatWindow(enable: Bool)
```

# TUICallEngine

Last updated : 2024-01-18 11:29:33

## TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

### Overview

API	Description
<code>createInstance</code>	Creates a <code>TUICallEngine</code> instance (singleton mode).
<code>destroyInstance</code>	Terminates a <code>TUICallEngine</code> instance (singleton mode).
<code>init</code>	Authenticates the basic audio/video call capabilities.
<code>addObserver</code>	Registers an event listener.
<code>removeObserver</code>	Unregisters an event listener.
<code>call</code>	Makes a one-to-one call.
<code>groupCall</code>	Makes a group call.
<code>accept</code>	Accepts a call.
<code>reject</code>	Rejects a call.
<code>hangup</code>	Ends a call.
<code>ignore</code>	Ignores a call.
<code>inviteUser</code>	Invites users to the current group call.
<code>joinInGroupCall</code>	Joins a group call.
<code>switchCallMediaType</code>	Changes the call type, for example, from video call to audio call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.

<a href="#">openCamera</a>	Turns the camera on.
<a href="#">closeCamera</a>	Turns the camera off.
<a href="#">switchCamera</a>	Switches between the front and rear cameras.
<a href="#">openMicrophone</a>	Turns the mic on.
<a href="#">closeMicrophone</a>	Turns the mic off.
<a href="#">selectAudioPlaybackDevice</a>	Selects the audio playback device (receiver or speaker).
<a href="#">setSelfInfo</a>	Sets the alias and profile photo.
<a href="#">enableMultiDeviceAbility</a>	Sets whether to enable multi-device login for <code>TUICallEngine</code> (supported by the premium package).
<a href="#">setVideoRenderParams</a>	Set the rendering mode of video image.
<a href="#">setVideoEncoderParams</a>	Set the encoding parameters of video encoder.
<a href="#">getTRTCCloudInstance</a>	Advanced features.
<a href="#">setBeautyLevel</a>	Set beauty level, support turning off default beauty.

## Details

### **createInstance**

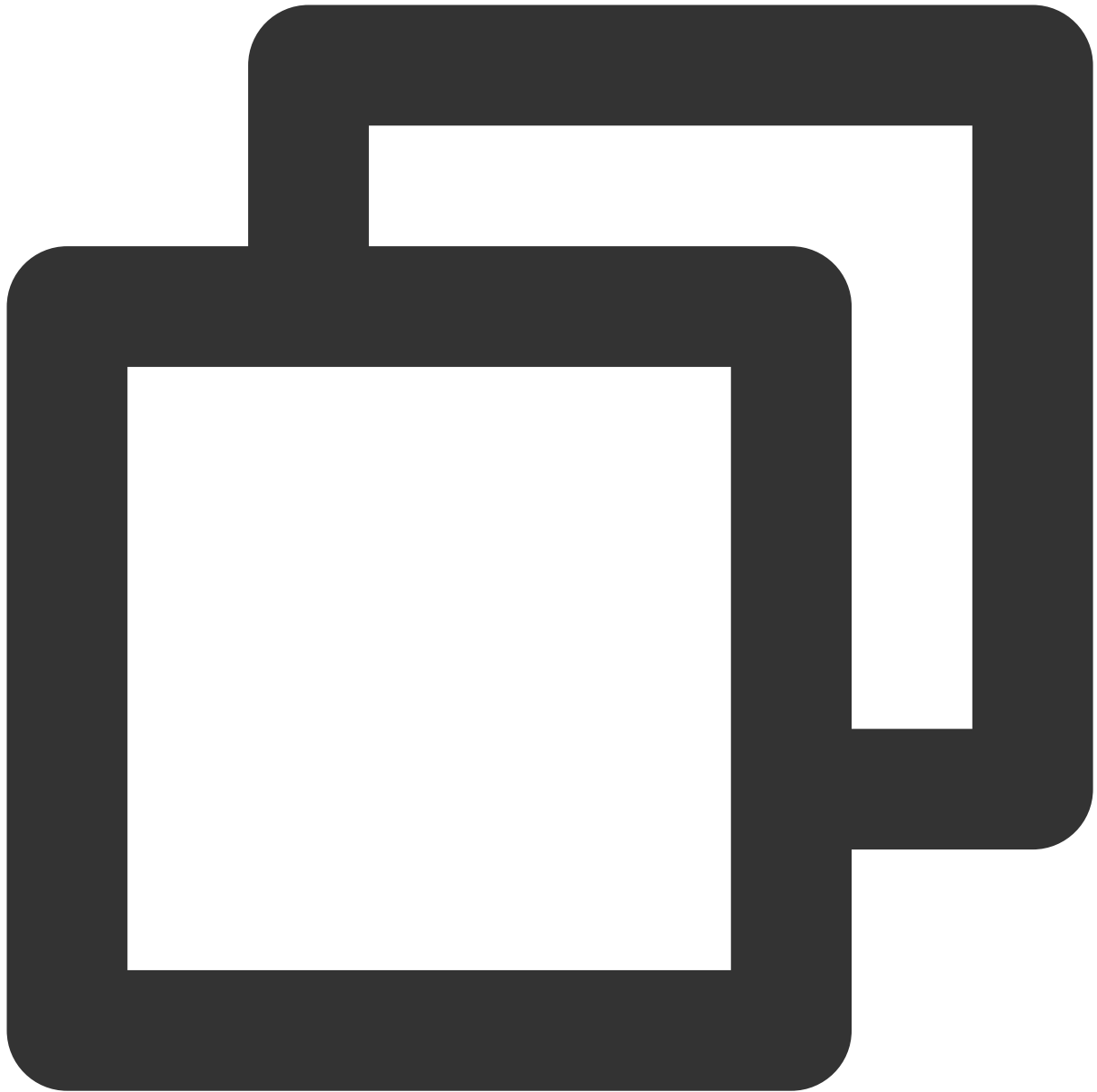
This API is used to create a `TUICallEngine` singleton.



```
- (TUICallEngine *)createInstance;
```

## **destroyInstance**

This API is used to terminate a `TUICallEngine` singleton.



```
- (void)destroyInstance;
```

## Init

This API is used to initialize `TUICallEngine` . Call it to authenticate the call service and perform other required actions before you call other APIs.



```
- (void)init:(NSString *)sdkAppID userId:(NSString *)userId userSig:(NSString *)userSig
```

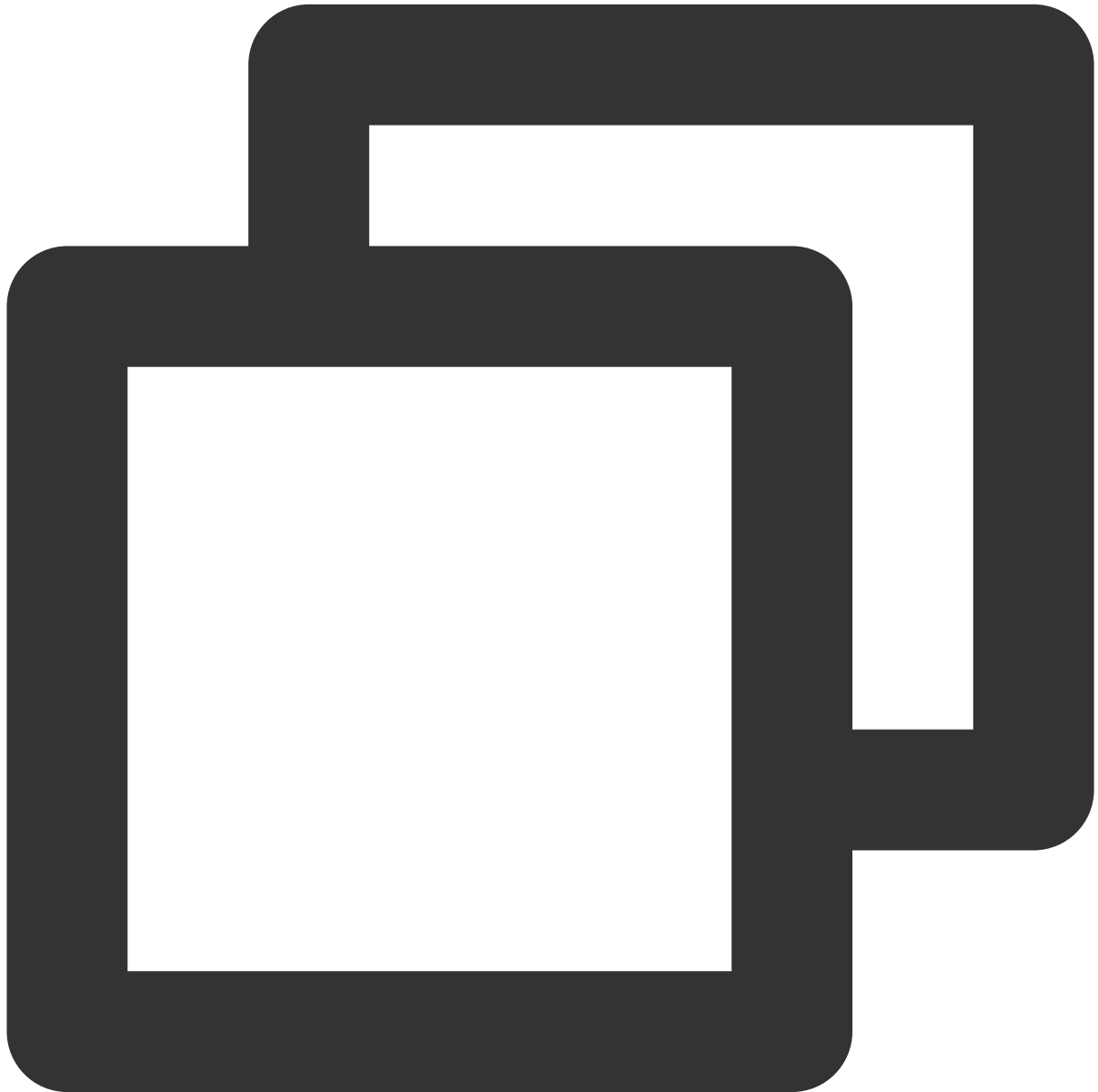
Parameter	Type	Description
sdkAppID	NSString	You can view <code>SDKAppID</code> in <a href="#">Application Management</a> > <b>Application Info</b> of the TRTC console.
userId	NSString	The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_).
userSig	NSString	Tencent Cloud's proprietary security signature. For how to calculate



and use it, see [UserSig](#).

## addObserver

This API is used to register an event listener to listen for `TUICallObserver` events.



```
- (void)addObserver:(id<TUICallObserver>)observer;
```

## removeObserver

This API is used to unregister an event listener.



```
- (void)removeObserver:(id<TUICallObserver>)observer;
```

## call

This API is used to make a (one-to-one) call.



```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType param
```

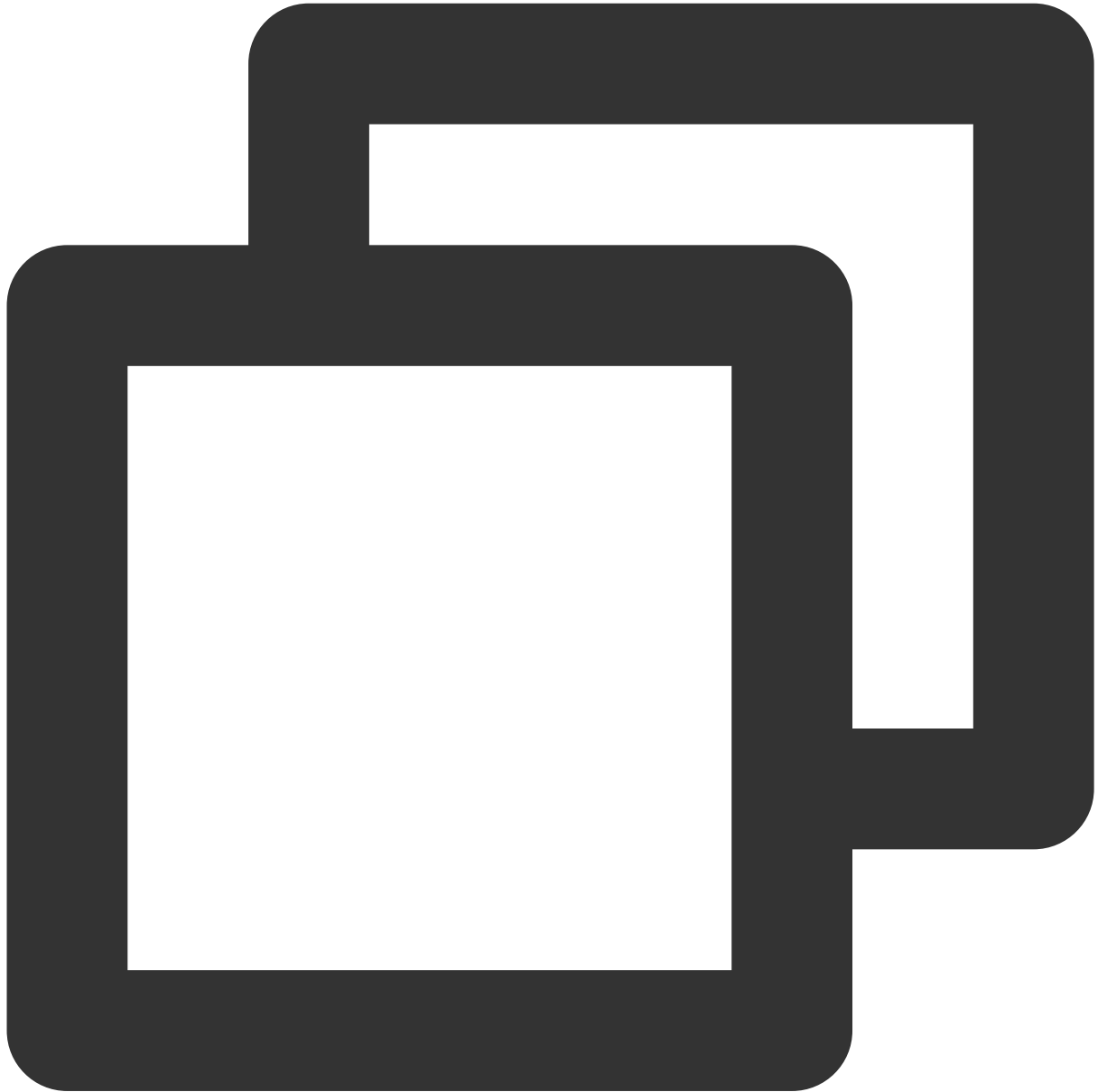
Parameter	Type	Description
userId	NSString	The target user ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
params	<a href="#">TUICallParams</a>	An additional parameter, such as roomId, call timeout, offline push info, etc

## groupCall

This API is used to make a group call.

### Notice :

Before making a group call, you need to create an IM group first.



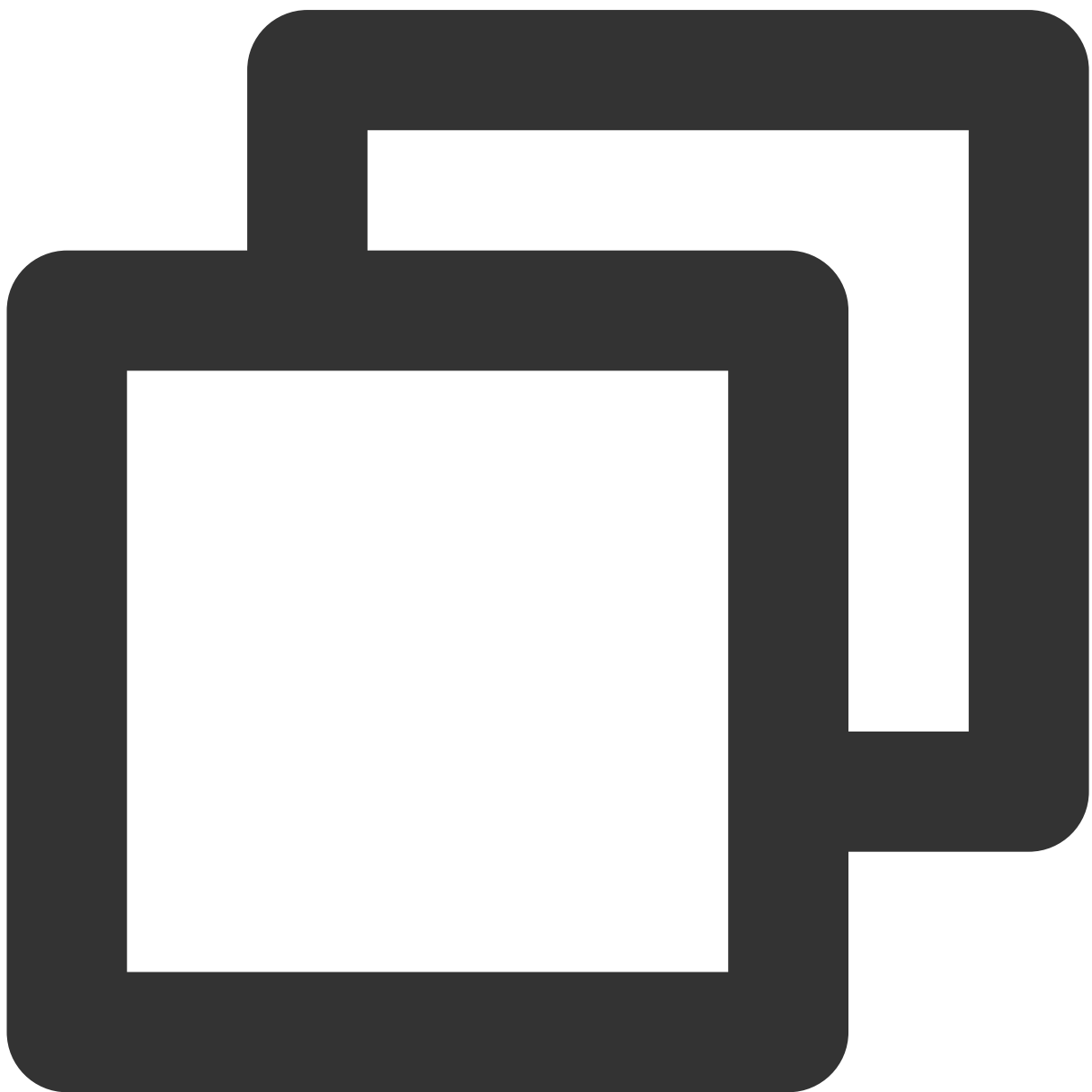
```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray <NSString *> *)userIdList
```

Parameter	Type	Description
groupId	NSString	The group ID.

userIdList	NSArray	The target user IDs.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
params	<a href="#">TUICallParams</a>	An additional parameter. such as roomId, call timeout, offline push info, etc

## accept

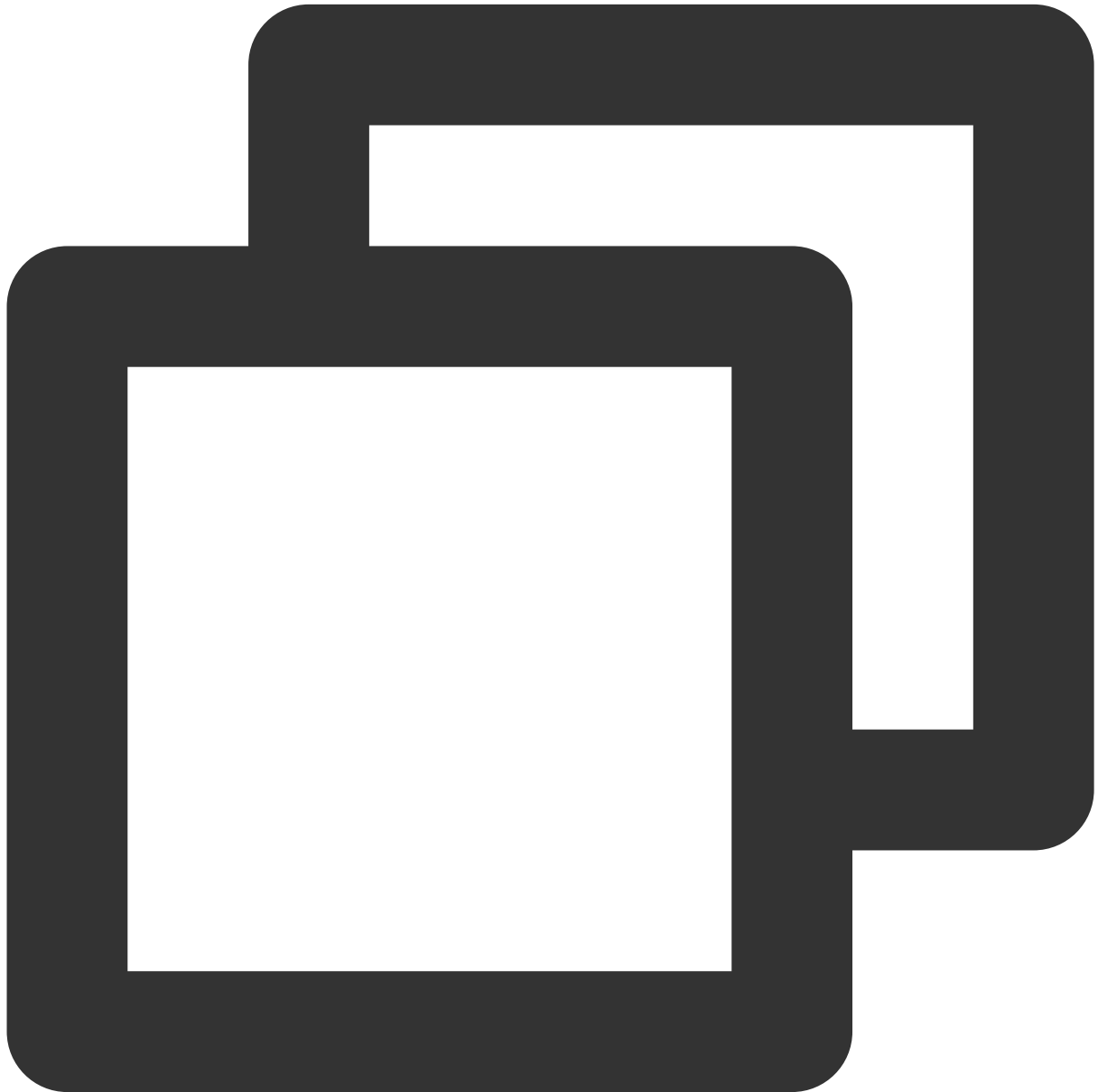
This API is used to accept a call. After receiving the `onCallReceived()` callback, you can call this API to accept the call.



```
- (void)accept:(TUICallSucc)succ fail:(TUICallFail)fail;
```

## reject

This API is used to reject a call. After receiving the `onCallReceived()` callback, you can call this API to reject the call.

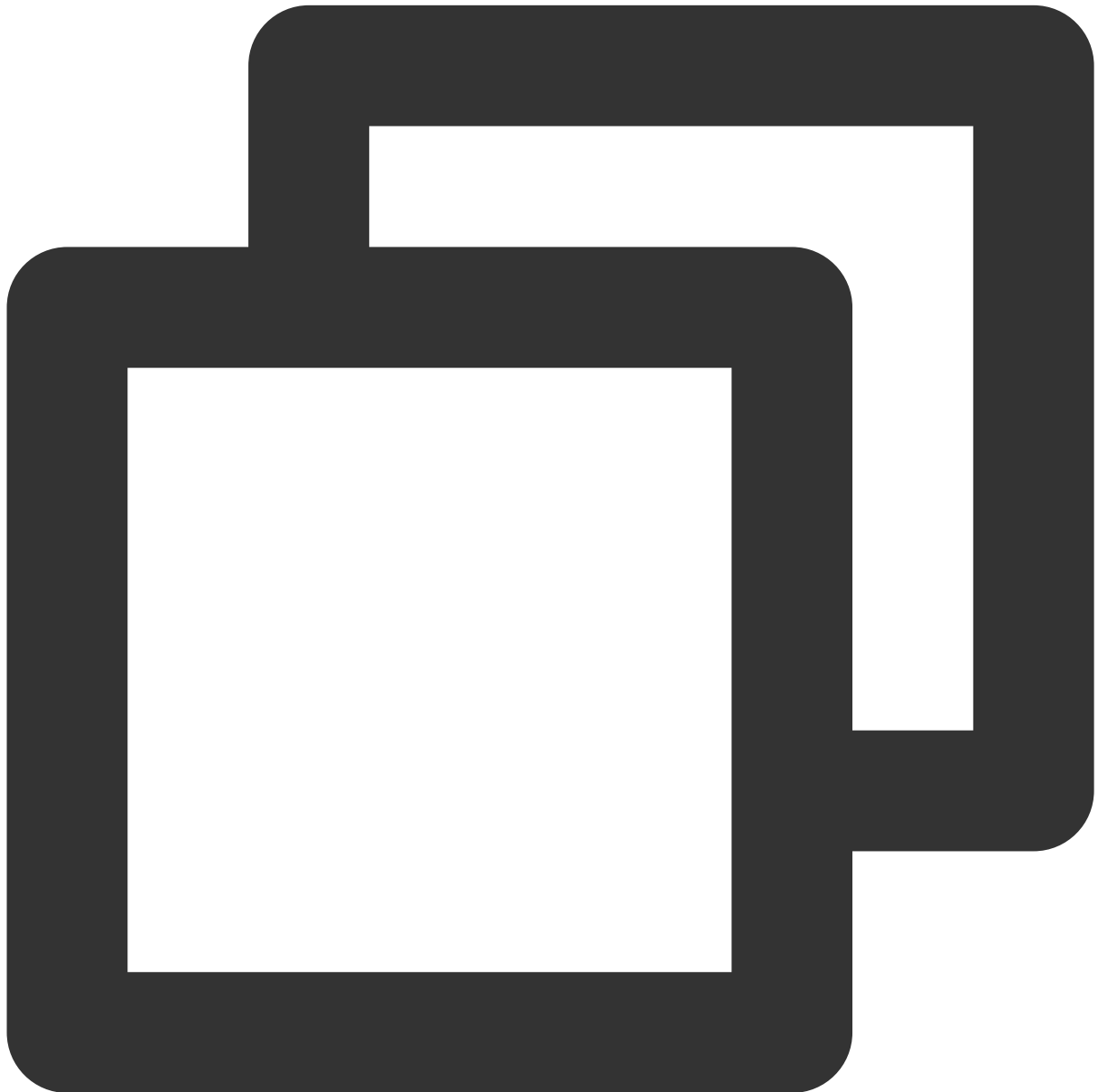


```
- (void)reject:(TUICallSucc)succ fail:(TUICallFail)fail;
```

## ignore

This API is used to ignore a call. After receiving the `onCallReceived()`, you can call this API to ignore the call. The caller will receive the `onUserLineBusy` callback.

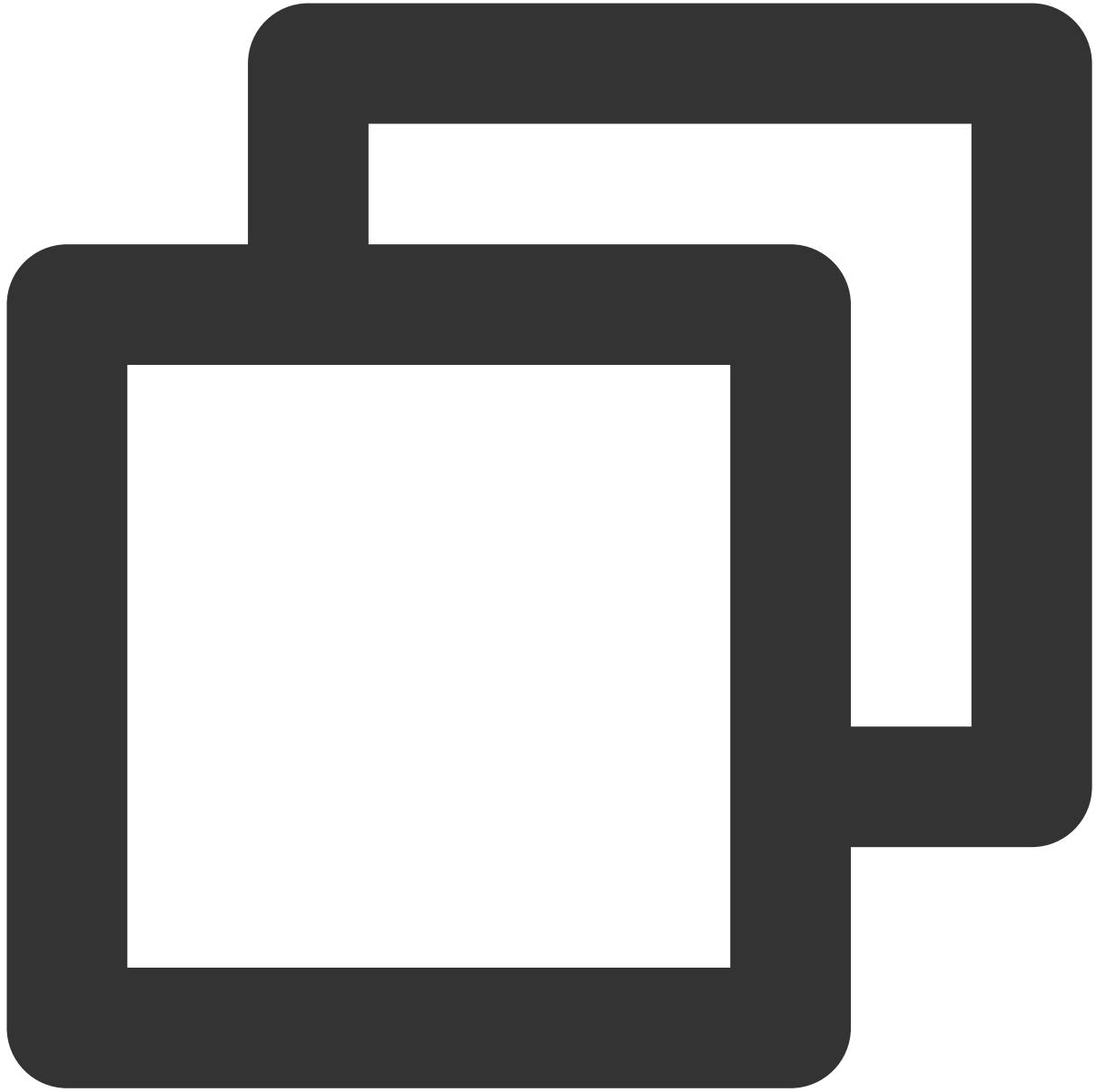
Note: If your project involves live streaming or conferencing, you can also use this API to implement the “in a meeting” or “on air” feature.



```
- (void)ignore:(TUICallSucc)succ fail:(TUICallFail)fail;
```

## hangup

This API is used to end a call.



```
- (void)hangup:(TUICallSucc)succ fail:(TUICallFail)fail;
```

## inviteUser

This API is used to invite users to the current group call.

This API is called by a participant of a group call to invite new users.





```
- (void)inviteUser:(NSArray<NSString *> *)userIdList params:(TUICallParams *)params
```

Parameter	Type	Description
userIdList	NSArray	The target user IDs.
params	<a href="#">TUICallParams</a>	An additional parameter. such as roomId, call timeout, offline push info, etc

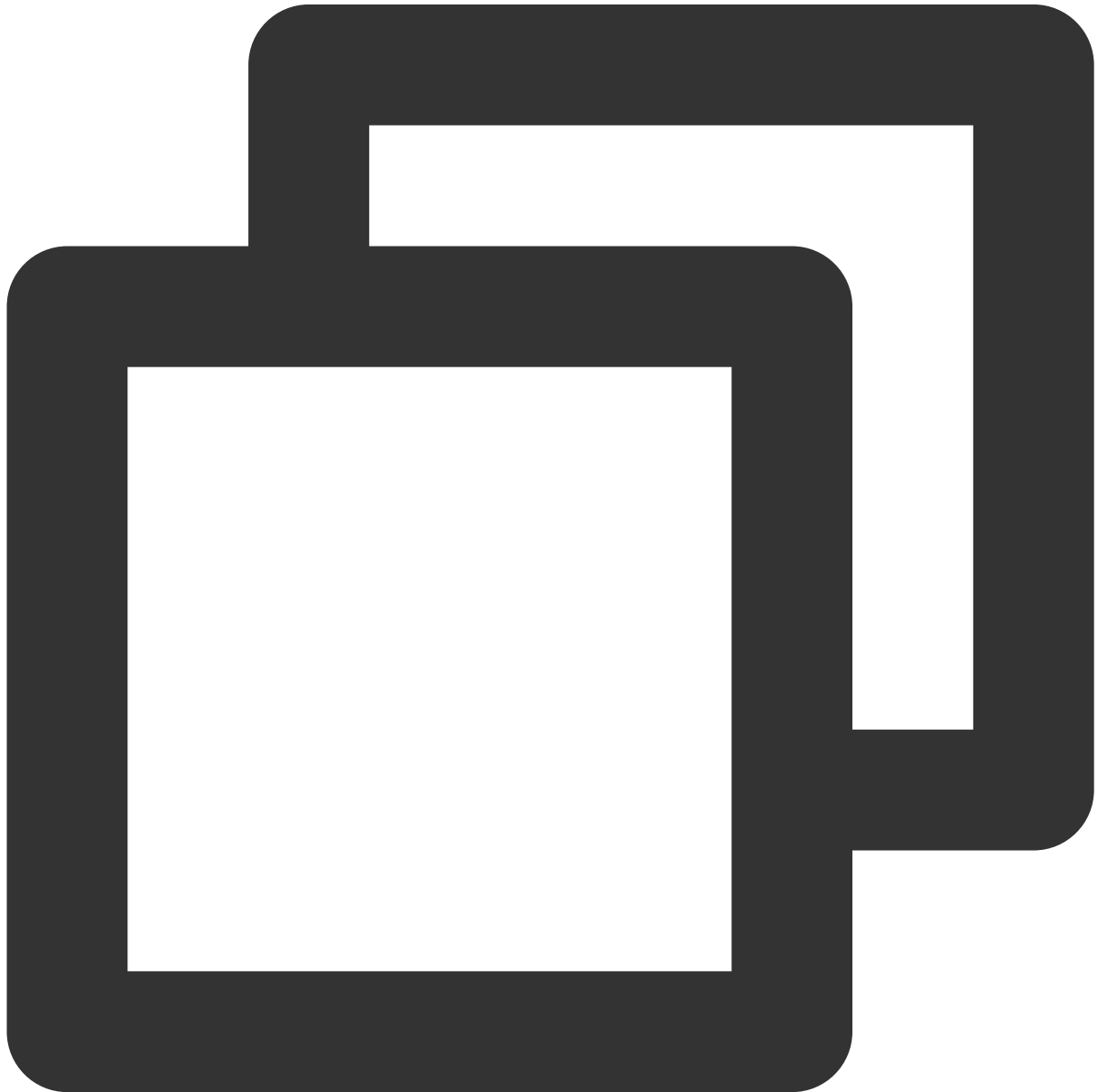
#### Notice :

In this case, the custom RoomId is invalid. The SDK will invite others to join the room where the inviter is currently located.

## joinInGroupCall

This API is used to join a group call.

This API is called by a group member to join the group's call.



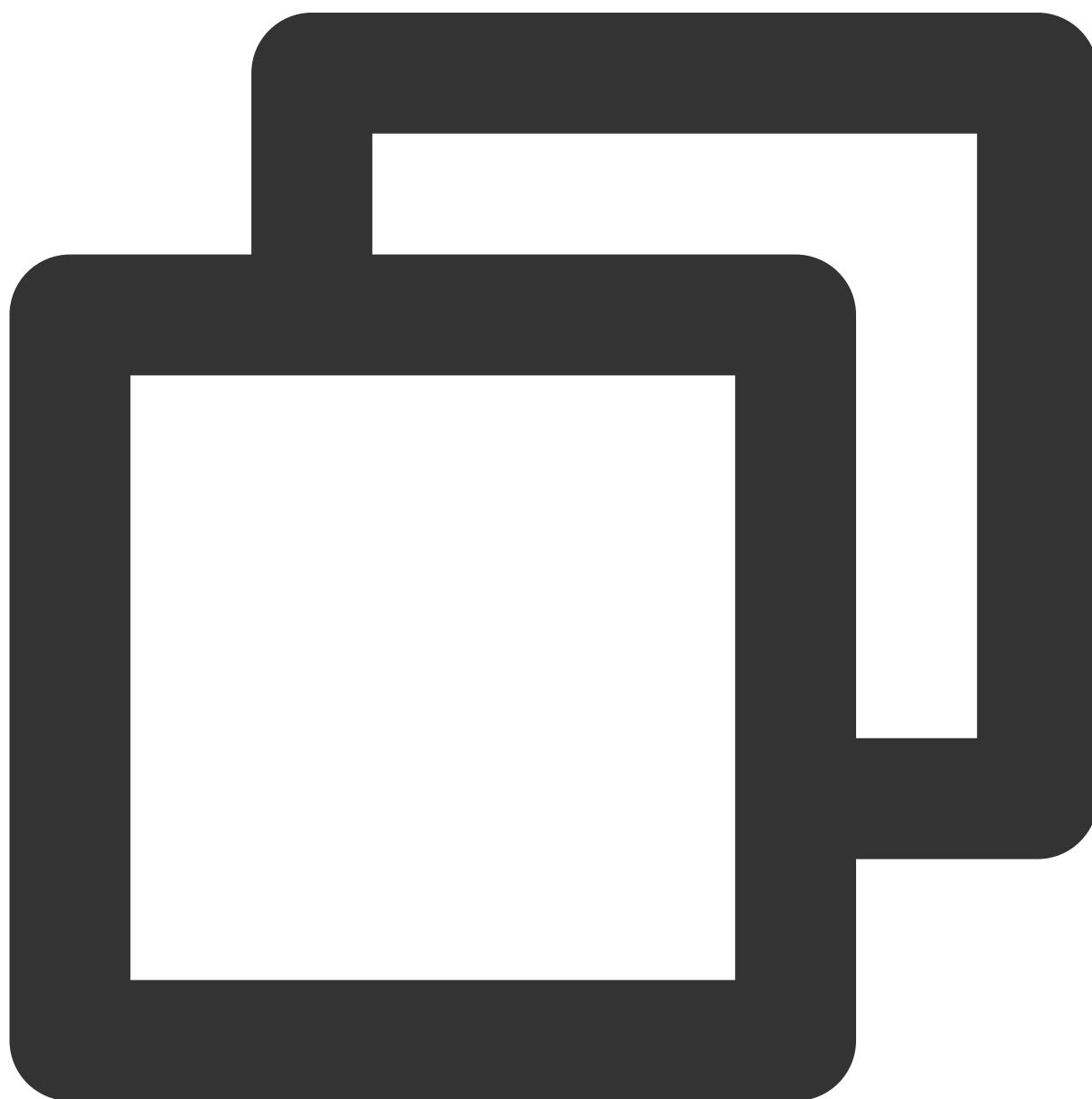
```
- (void)joinInGroupCall:(TUIRoomId *)roomId groupId:(NSString *)groupId callMediaTy
```

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

roomId	<a href="#">TUIRoomId</a>	The room ID.
groupId	NSString	The group ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.

## switchCallMediaType

This API is used to change the call type.

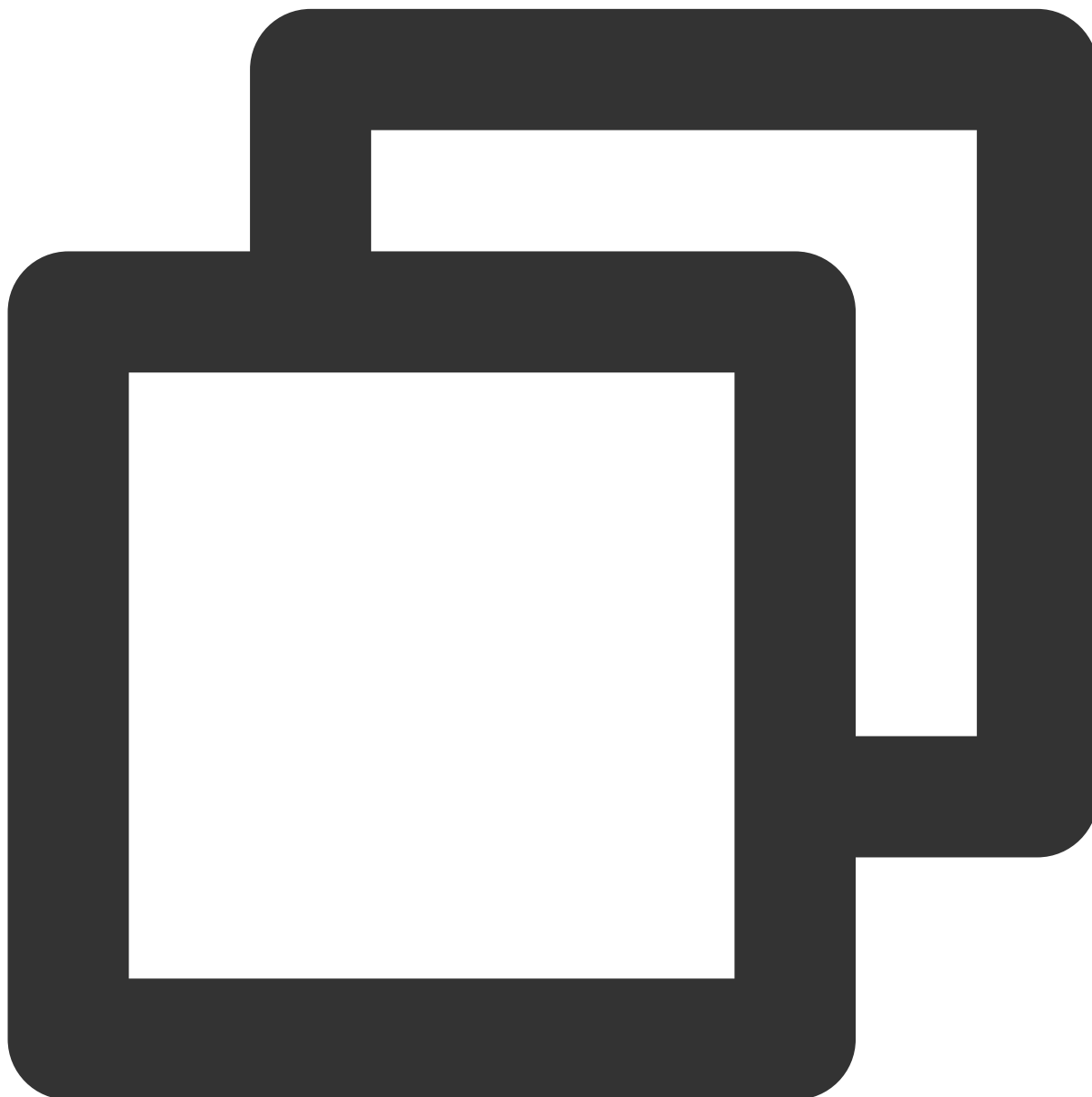


```
- (void)switchCallMediaType:(TUICallMediaType)newType;
```

Parameter	Type	Description
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.

## startRemoteView

This API is used to set the view object to display a remote video.

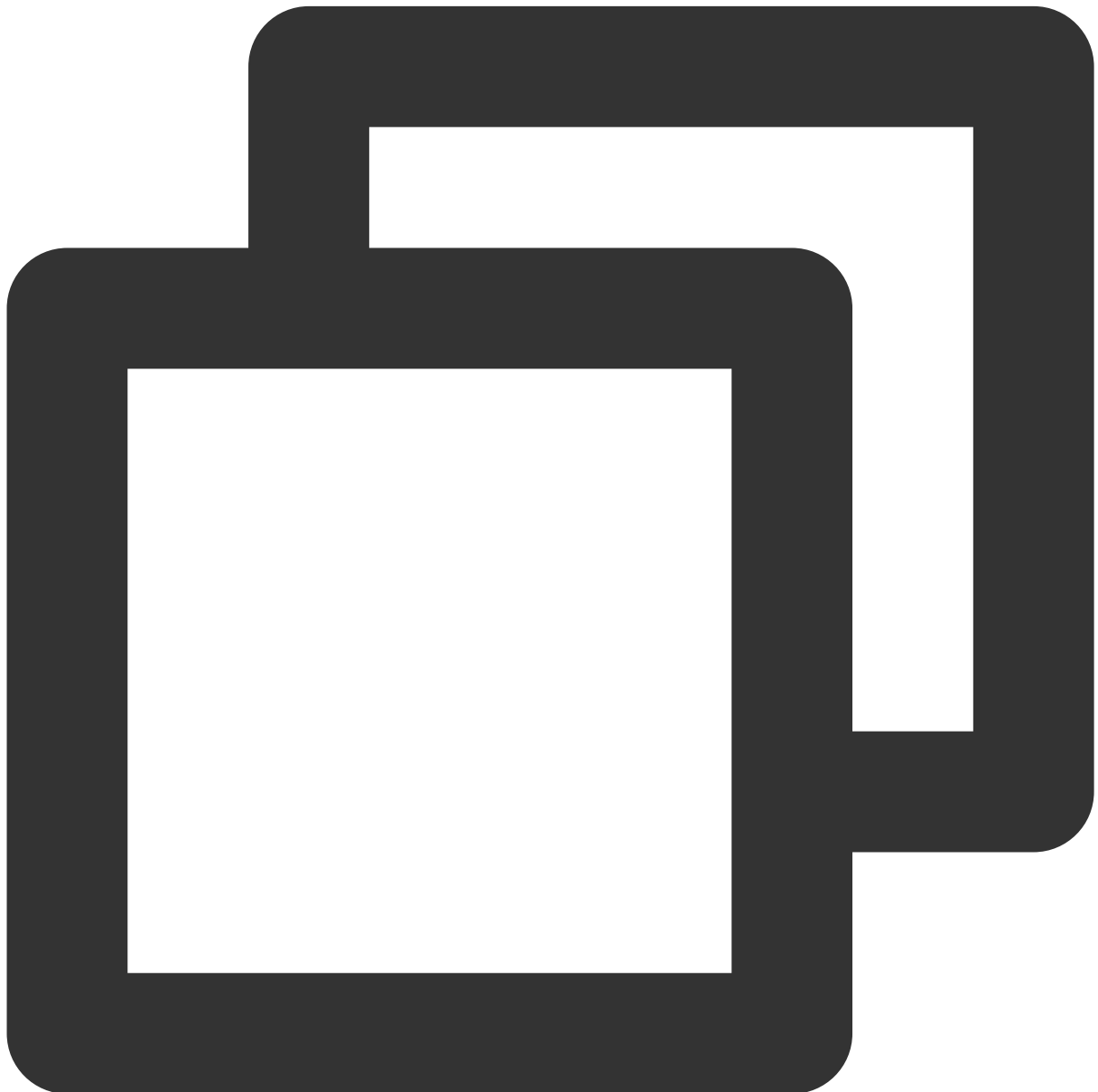


```
- (void)startRemoteView:(NSString *)userId videoView:(TUIVideoView *)videoView onPl
```

Parameter	Type	Description
userId	NSString	The target user ID.
videoView	TUIVideoView	The view to be rendered.

## stopRemoteView

This API is used to unsubscribe from the video stream of a remote user.

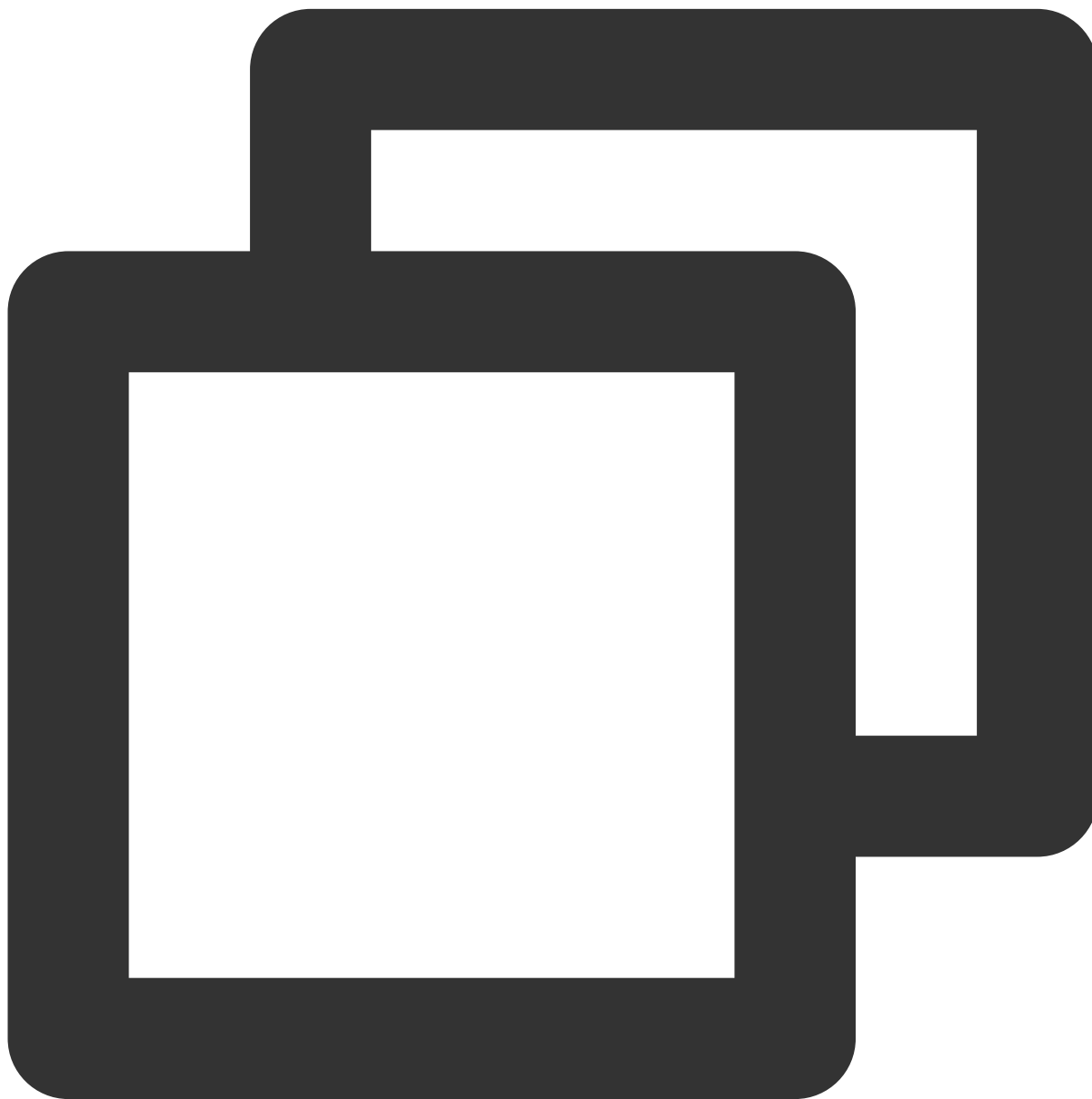


```
- (void)stopRemoteView:(NSString *)userId;
```

Parameter	Type	Description
userId	NSString	The target user ID.

## openCamera

This API is used to turn the camera on.



```
- (void)openCamera:(TUICamera)camera videoView:(TUIVideoView *)videoView succ:(TUIC
```

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

camera	<a href="#">TUICamera</a>	The front or rear camera.
videoView	TUIVideoView	The view to be rendered.

## closeCamera

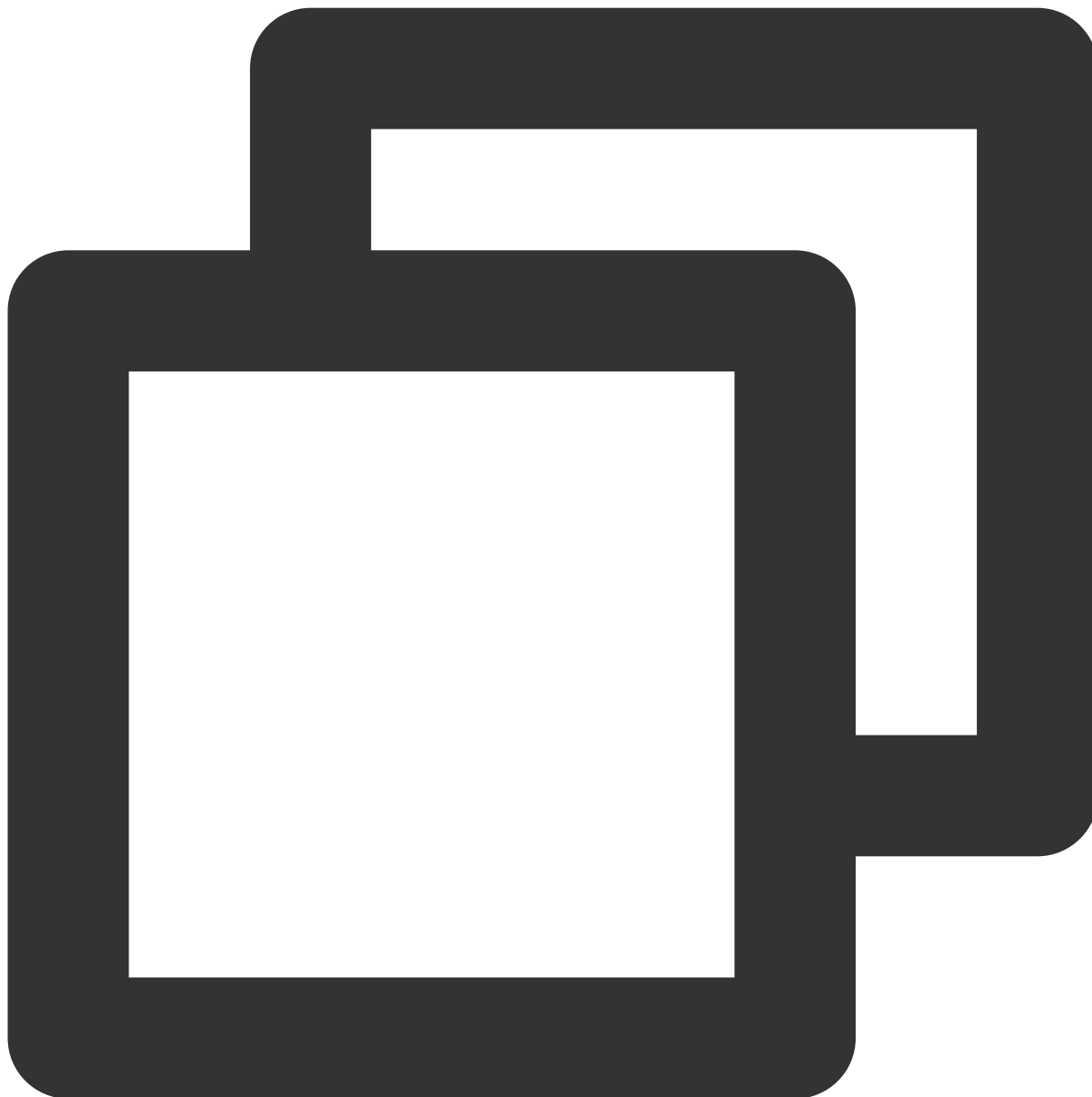
This API is used to turn the camera off.



```
- (void)closeCamera;
```

## switchCamera

This API is used to switch between the front and rear cameras.



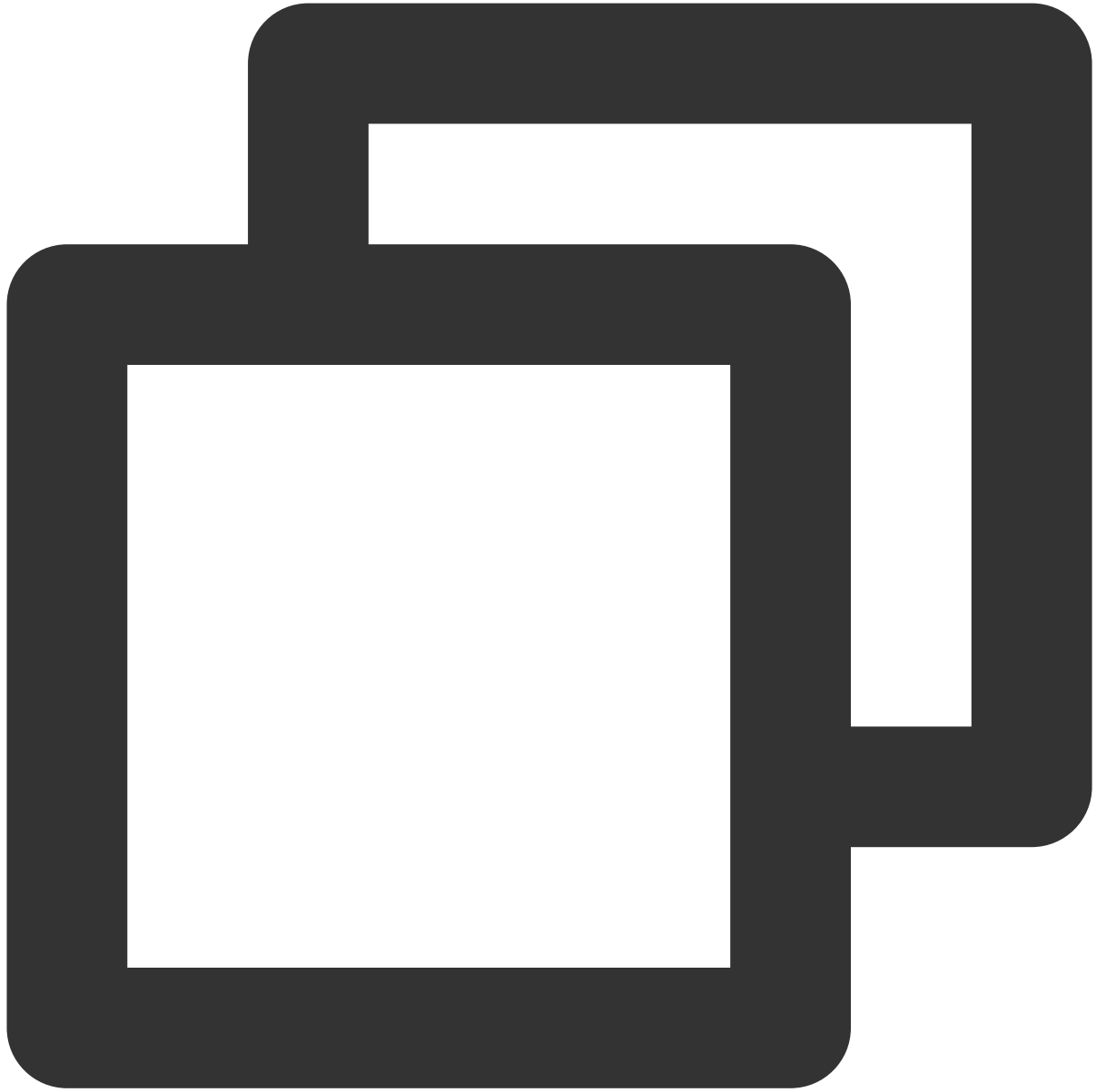
```
- (void)switchCamera:(TUICamera)camera;
```

Parameter	Type	Description
camera	<a href="#">TUICamera</a>	The front or rear camera.

## openMicrophone



This API is used to turn the mic on.



```
- (void)openMicrophone:(TUICallSucc)succ fail:(TUICallFail)fail;
```

## **closeMicrophone**

This API is used to turn the mic off.



```
- (void)closeMicrophone;
```

### **selectAudioPlaybackDevice**

This API is used to select the audio playback device (receiver or speaker). In call scenarios, you can use this API to turn on/off hands-free mode.



```
- (void)selectAudioPlaybackDevice:(TUIAudioPlaybackDevice) device;
```

Parameter	Type	Description
device	<a href="#">TUIAudioPlaybackDevice</a>	The speaker or receiver.

## setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.



```
- (void)setSelfInfo:(NSString * _Nullable)nickName avatar:(NSString * _Nullable)ava
```

### **enableMultiDeviceAbility**

This API is used to set whether to enable multi-device login for `TUICallEngine` (supported by the premium package).



```
- (void)enableMultiDeviceAbility:(BOOL)enable succ:(TUICallSucc)succ fail:(TUICallF
```

## setVideoRenderParams

Set the rendering mode of video image.



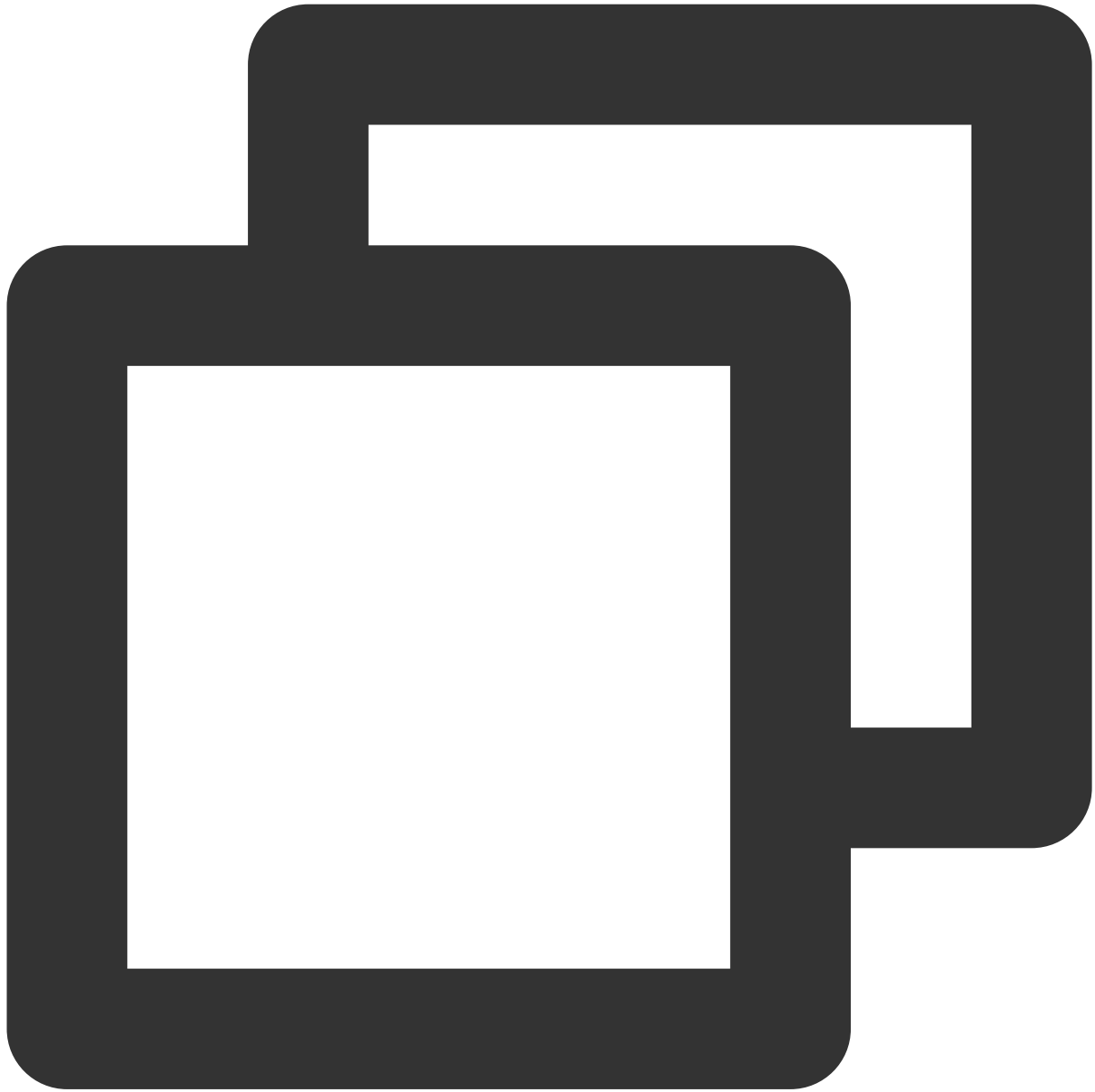
```
- (void)setVideoRenderParams:(NSString *)userId params:(TUIVideoRenderParams *)para
```

Parameter	Type	Description
userId	NSString	The target user ID.
params	<a href="#">TUIVideoRenderParams</a>	Video render parameters.

## setVideoEncoderParams

Set the encoding parameters of video encoder.

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.

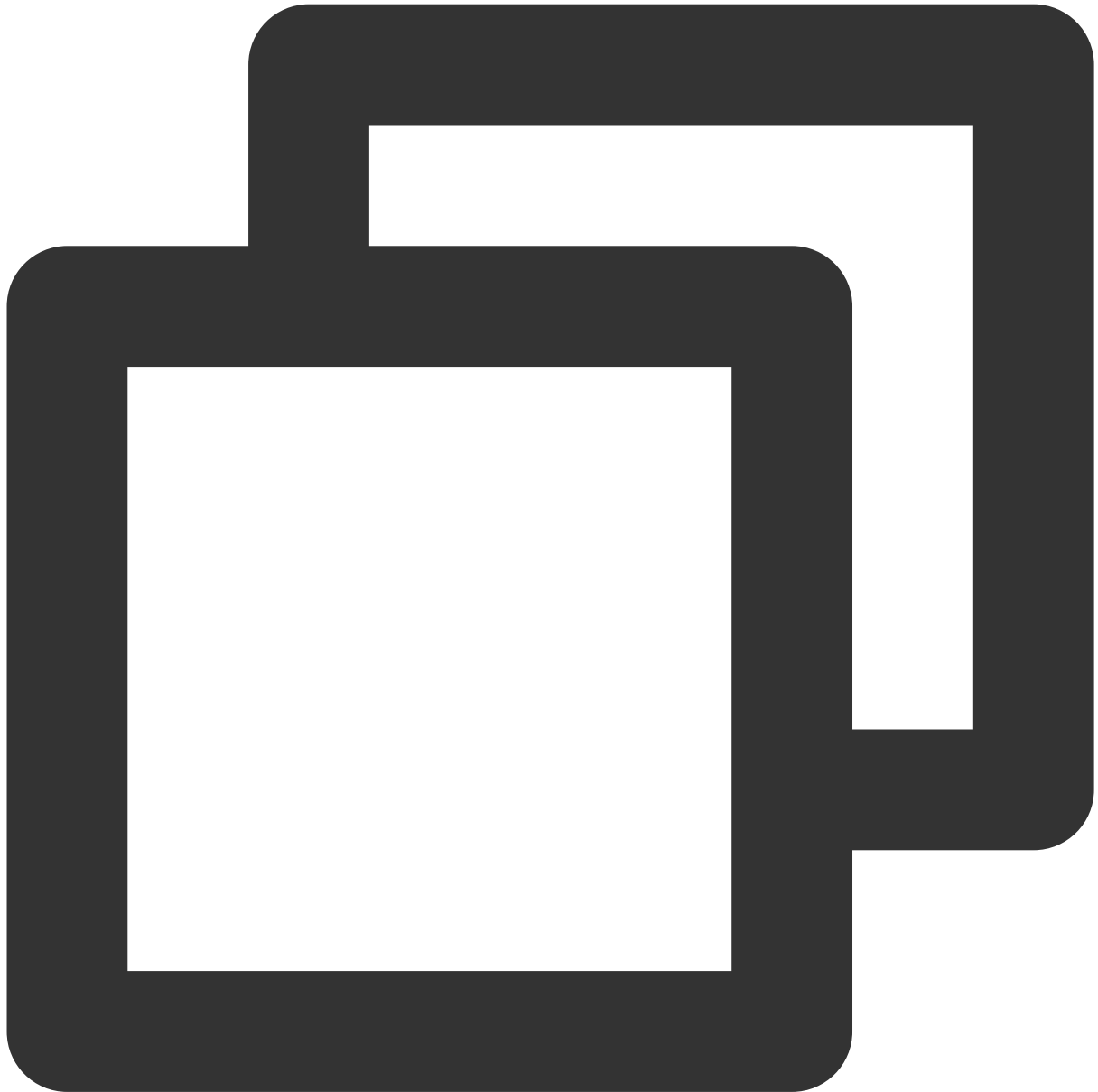


```
- (void)setVideoEncoderParams:(TUIVideoEncoderParams *)params succ:(TUICallSucc) suc
```

Parameter	Type	Description
params	<a href="#">TUIVideoEncoderParams</a>	Video encoding parameters

## getTRTCCloudInstance

Advanced features.



```
- (TRTCCloud *)getTRTCCloudInstance;
```

## setBeautyLevel

Set beauty level, support turning off default beauty.





```
- (void)setBeautyLevel:(CGFloat)level succ:(TUICallSucc)succ fail:(TUICallFail)fail
```

Parameter	Type	Description
level	CGFloat	Beauty level, range: 0 - 9 ; 0 means turning off the effect, 9 means the most obvious effect.

# TUICallObserver

Last updated : 2024-01-17 11:58:53

## TUICallObserver APIs

`TUICallObserver` is the callback class of `TUICallEngine`. You can use it to listen for events.

## Overview

API	Description
<code>onError</code>	A call occurred during the call.
<code>onCallReceived</code>	A call invitation was received.
<code>onCallCancelled</code>	The call was canceled.
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user had a video stream.
<code>onUserAudioAvailable</code>	Whether a user had an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.

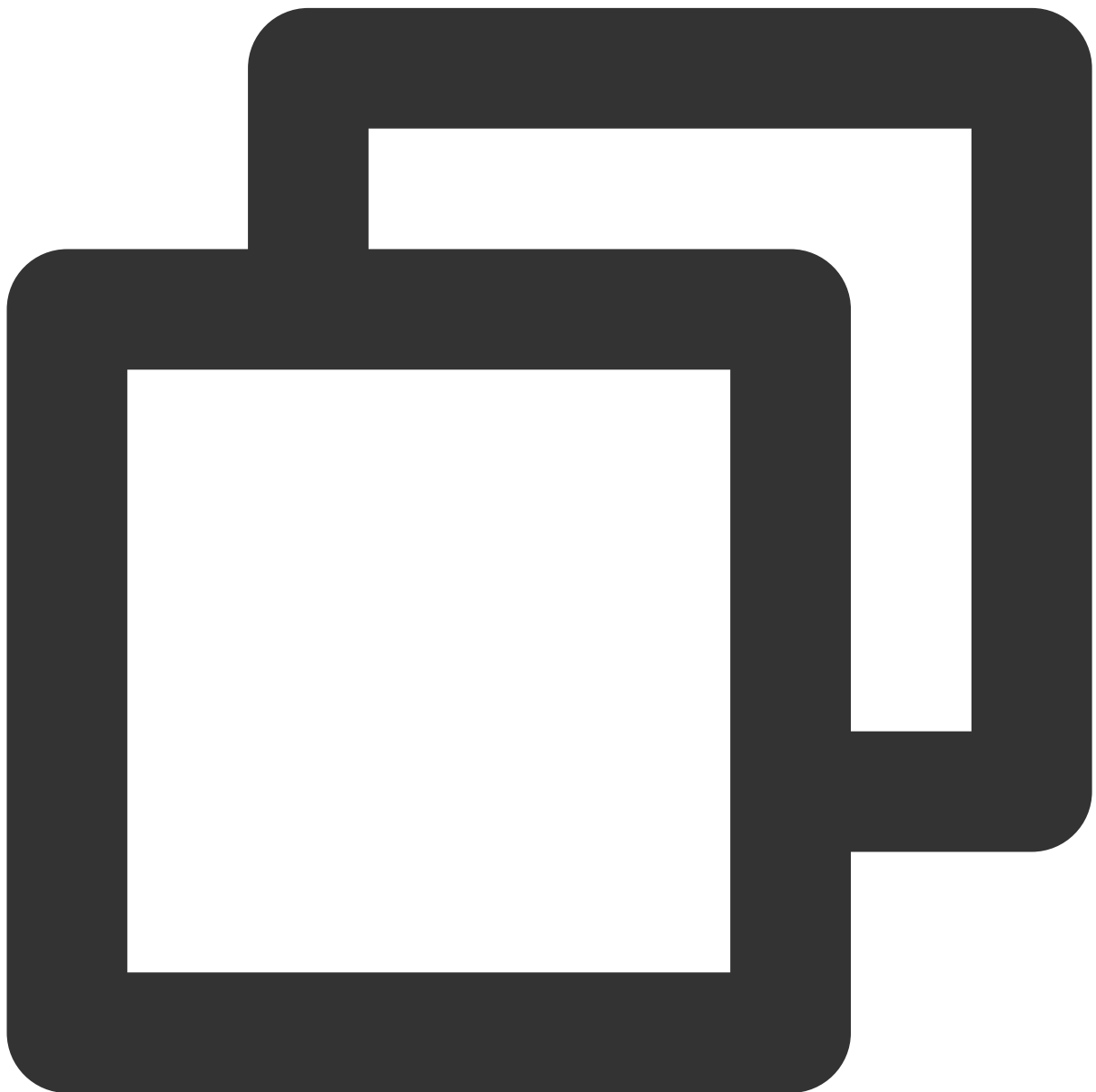
## Details

### onError

An error occurred.

#### Note

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



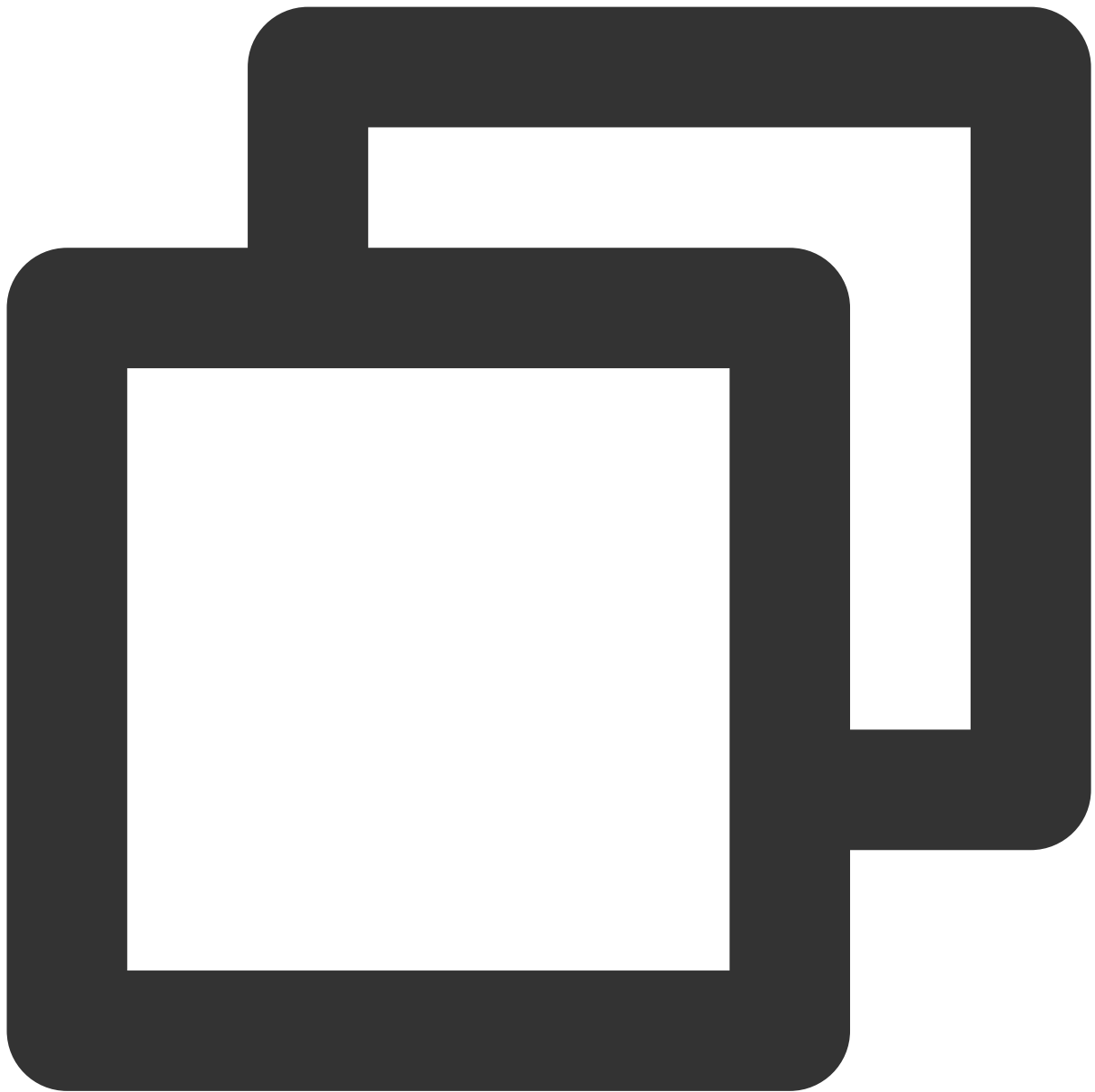
```
- (void)onError:(int)code message:(NSString * _Nullable)message;
```

The parameters are described below:

Parameter	Type	Description
code	int	The error code.
message	NSString	The error message.

## **onCallReceived**

A call invitation was received. This callback is received by an invitee. You can listen for this event to determine whether to display the incoming call view.



```
- (void)onCallReceived:(NSString *)callerId calleeIdList:(NSArray<NSString *> *)cal
```

The parameters are described below:

Parameter	Type	Description
callerId	NSString	The user ID of the inviter.
calleeldList	NSArray	The invitee list.
groupId	NSString	The group ID.

callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
userData	NSString	User-added extended fields., Please refer to:

## onCallCancelled

The call was canceled by the inviter or timed out. This callback is received by an invitee. You can listen for this event to determine whether to show a missed call message.

This indicates that the call was canceled by the caller, timed out by the callee, rejected by the callee, or the callee was busy. There are multiple scenarios involved. You can listen to this event to achieve UI logic such as missed calls and resetting UI status.

Call cancellation by the caller: The caller receives the callback (userId is himself); the callee receives the callback (userId is the ID of the caller)

Callee timeout: the caller will simultaneously receive the [onUserNoResponse](#) and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)

Callee rejection: The caller will simultaneously receive the [onUserReject](#) and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)

Callee busy: The caller will simultaneously receive the [onUserLineBusy](#) and onCallCancelled callbacks (userId is his own ID);

Abnormal interruption: The callee failed to receive the call , he receives this callback (userId is his own ID).



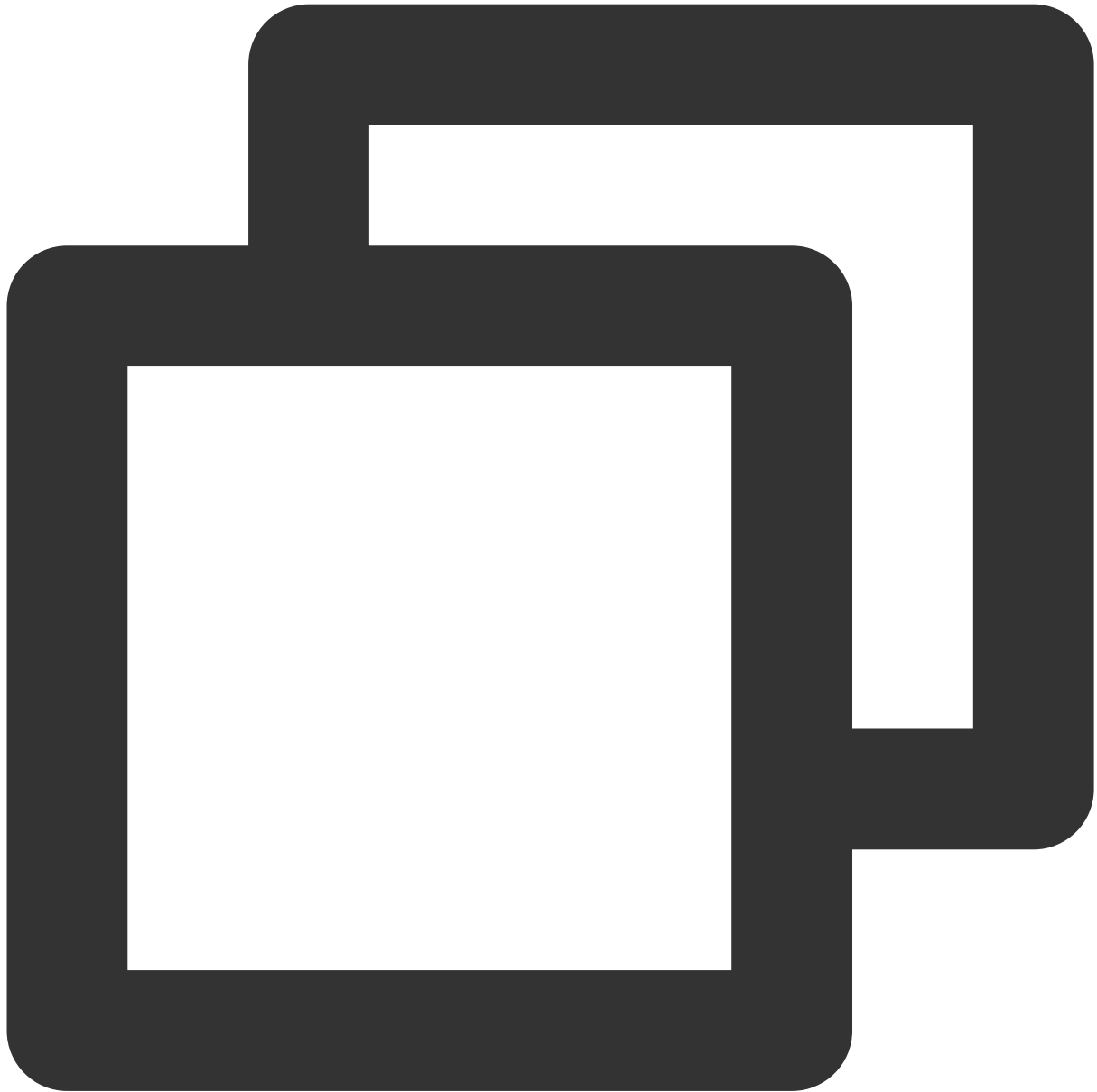
```
- (void)onCallCancelled:(NSString *)callerId;
```

The parameters are described below:

Parameter	Type	Description
callerId	NSString	The user ID of the inviter.

## onCallBegin

The call was connected. This callback is received by both the inviter and invitees. You can listen for this event to determine whether to start on-cloud recording, content moderation, or other tasks.



```
- (void)onCallBegin:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaTy
```

The parameters are described below:

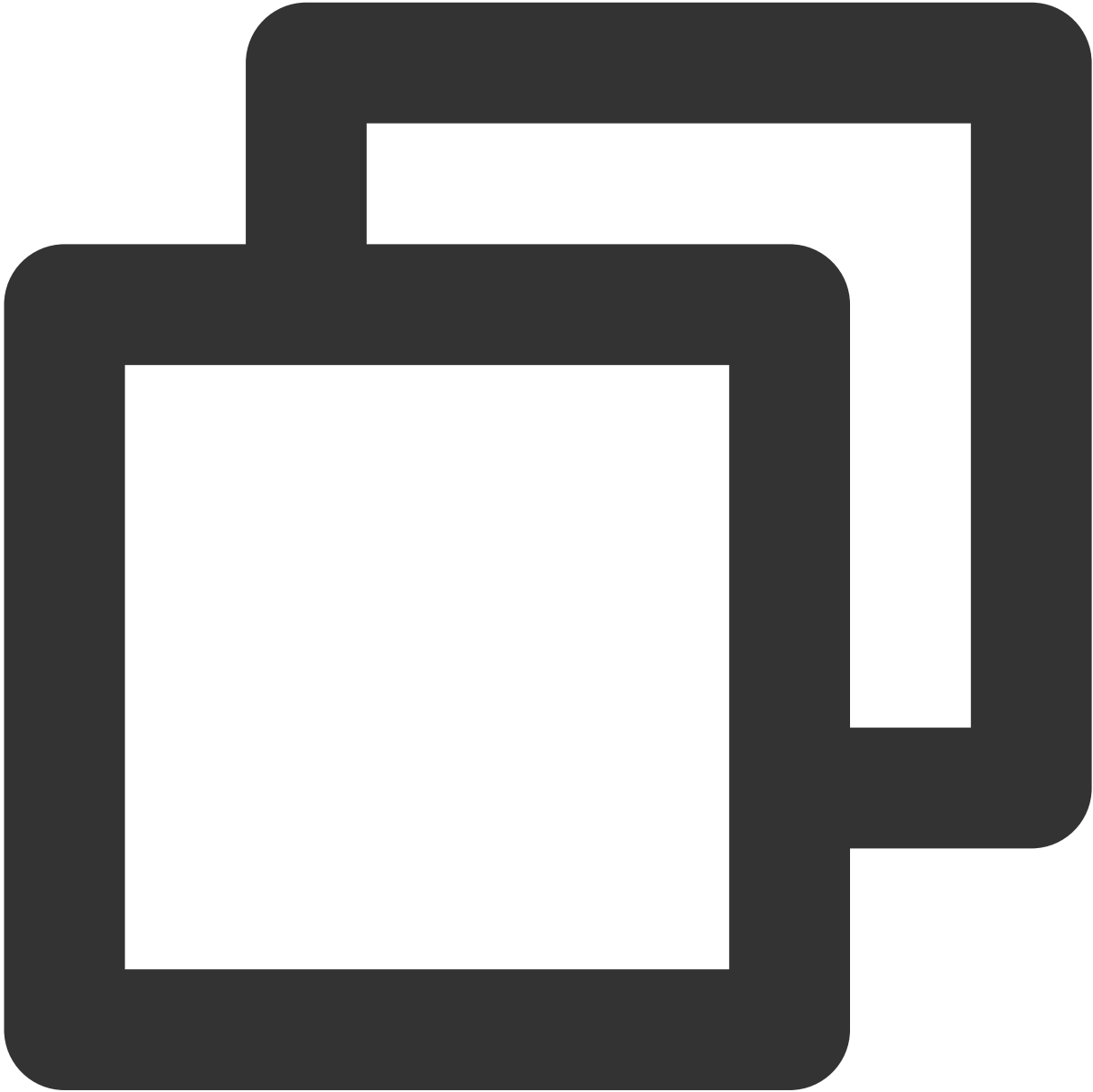
Parameter	Type	Description
roomId	<a href="#">TUIRoomId</a>	The room ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.



callRole	<a href="#">TUICallRole</a>	The role, which can be caller or callee.
----------	-----------------------------	--

onCallEnd

The call ended. This callback is received by both the inviter and invitees. You can listen for this event to determine when to display call information such as call duration and call type, or stop on-cloud recording.



```
- (void)onCallEnd:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaType
```

The parameters are described below:

--	--	--

Parameter	Type	Description
roomId	<a href="#">TUIRoomId</a>	The room ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
callRole	<a href="#">TUICallRole</a>	The role, which can be caller or callee.
totalTime	float	The call duration.

### Note

Client-side callbacks are often lost when errors occur, for example, when the process is closed. If you need to measure the duration of a call for billing or other purposes, we recommend you use the RESTful API.

### onCallMediaTypeChanged

The call type changed.



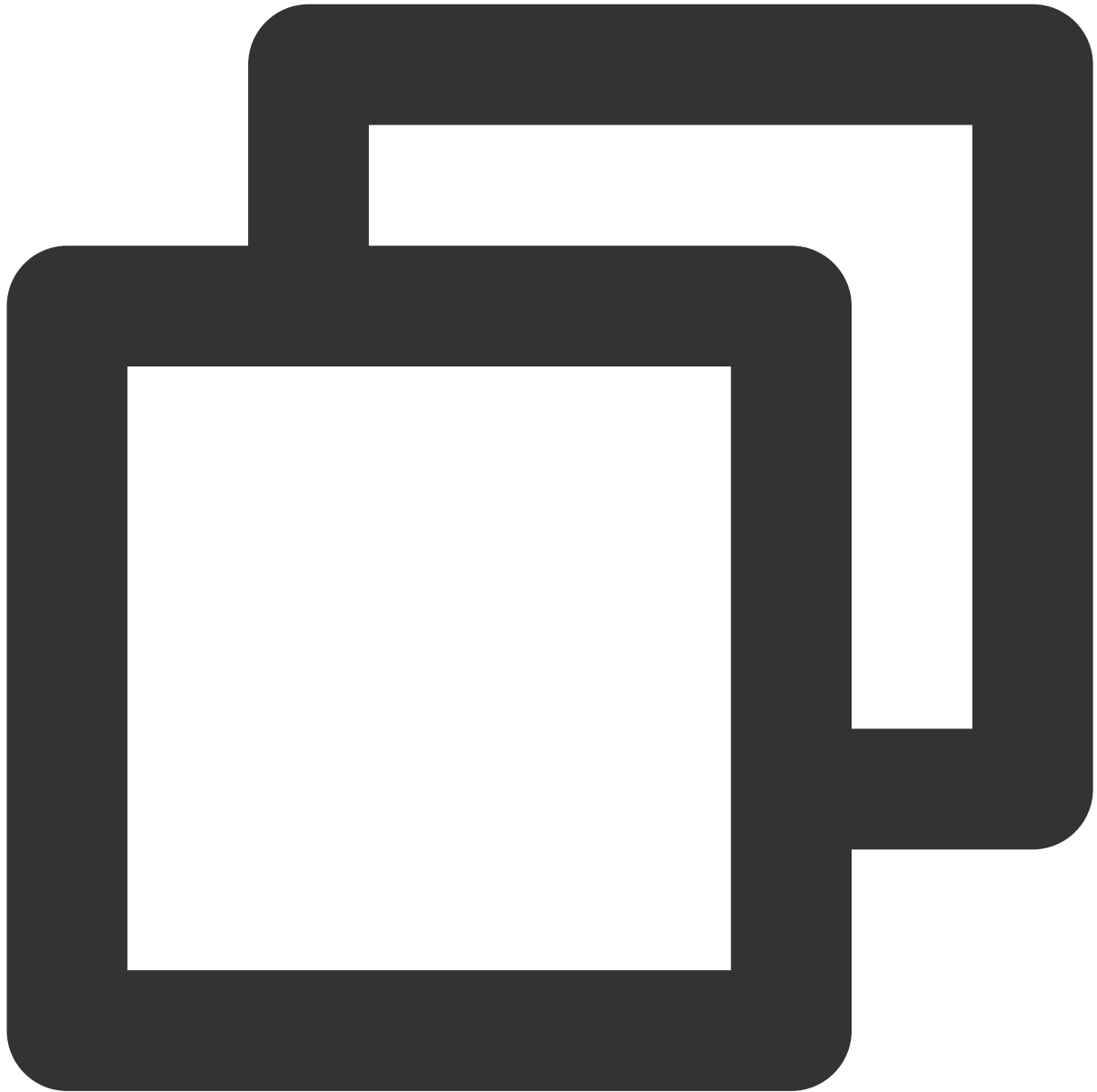
```
- (void)onCallMediaTypeChanged:(TUICallMediaType)oldCallMediaType newCallMediaType:
```

The parameters are described below:

Parameter	Type	Description
oldCallMediaType	<a href="#">TUICallMediaType</a>	The call type before the change.
newCallMediaType	<a href="#">TUICallMediaType</a>	The call type after the change.

## onUserReject

The call was rejected. In a one-to-one call, only the inviter will receive this callback. In a group call, all invitees will receive this callback.



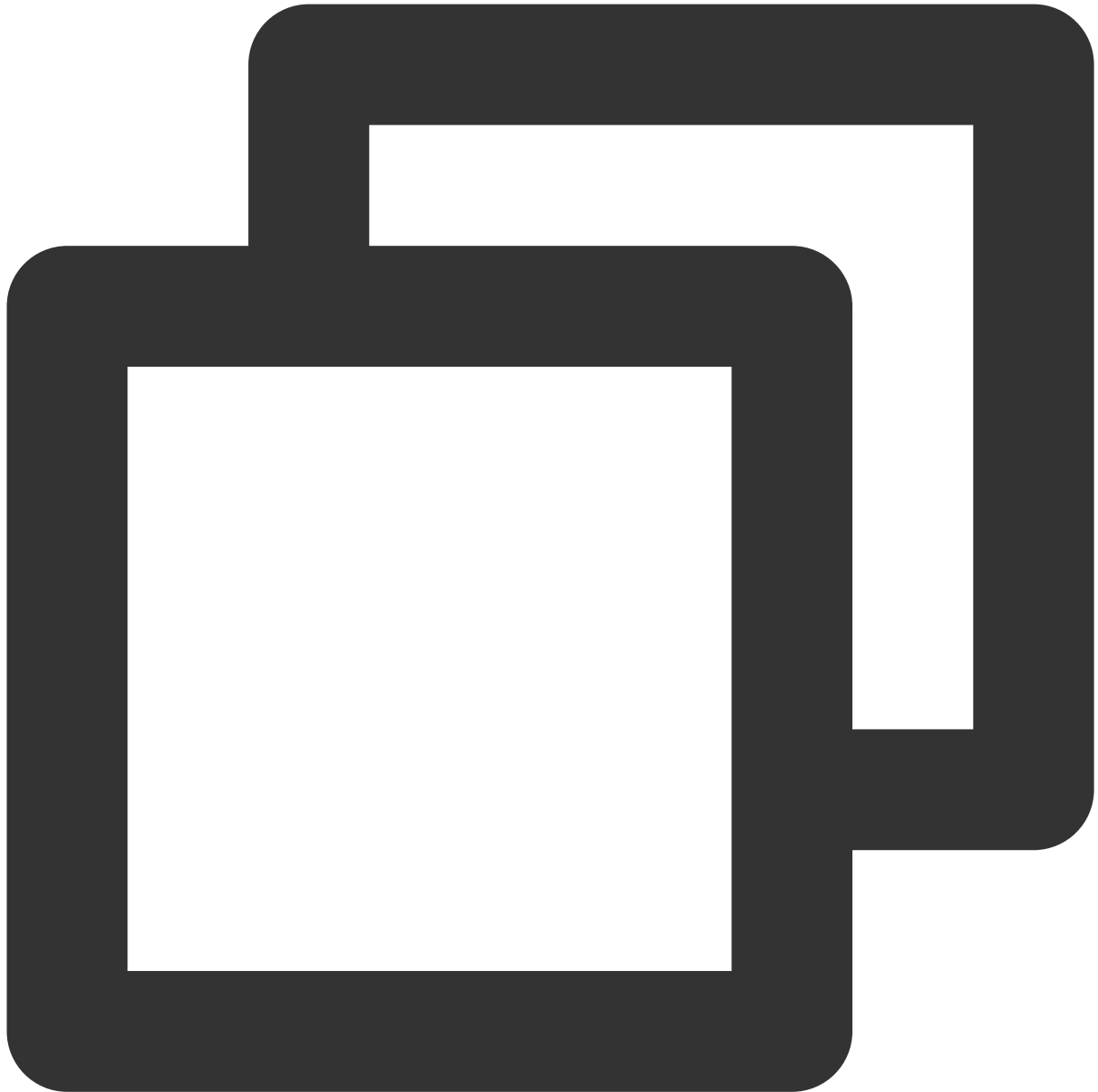
```
- (void)onUserReject:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID of the invitee who rejected the call.

## onUserNoResponse

A user did not respond.



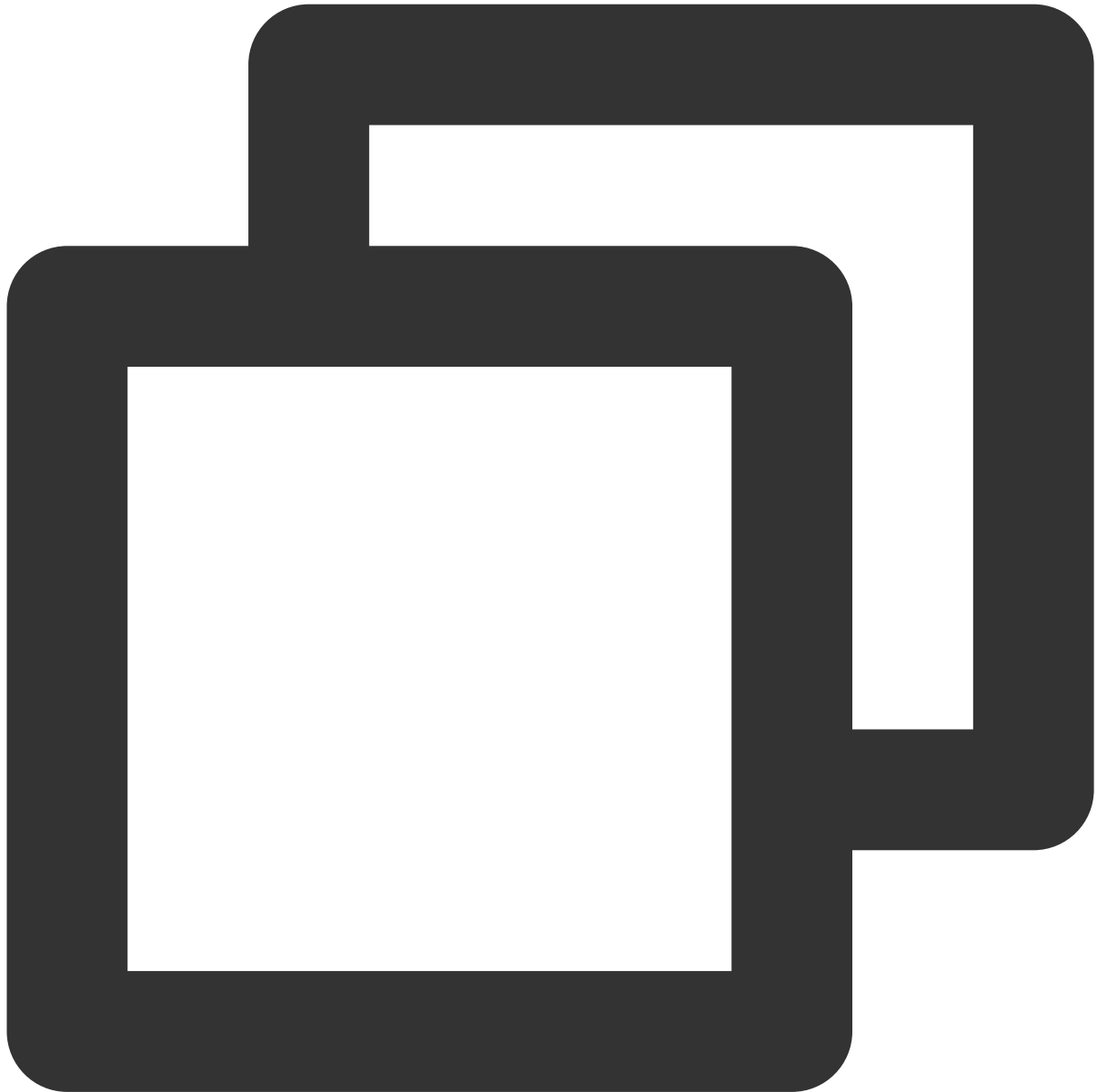
```
- (void)onUserNoResponse:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID of the invitee who did not answer.

## onUserLineBusy

A user is busy.



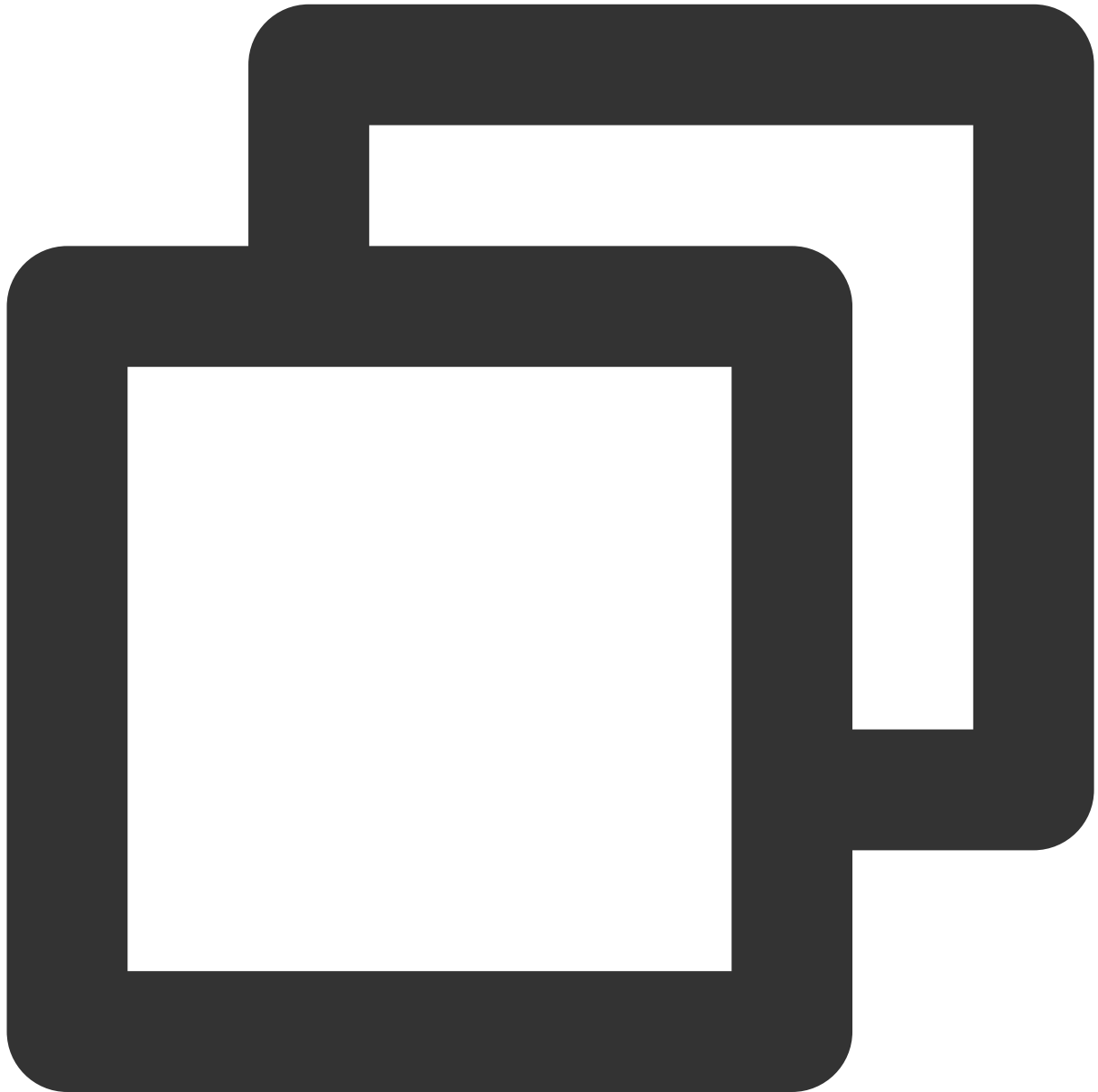
```
- (void)onUserLineBusy:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID of the invitee who is busy.

## onUserJoin

A user joined the call.



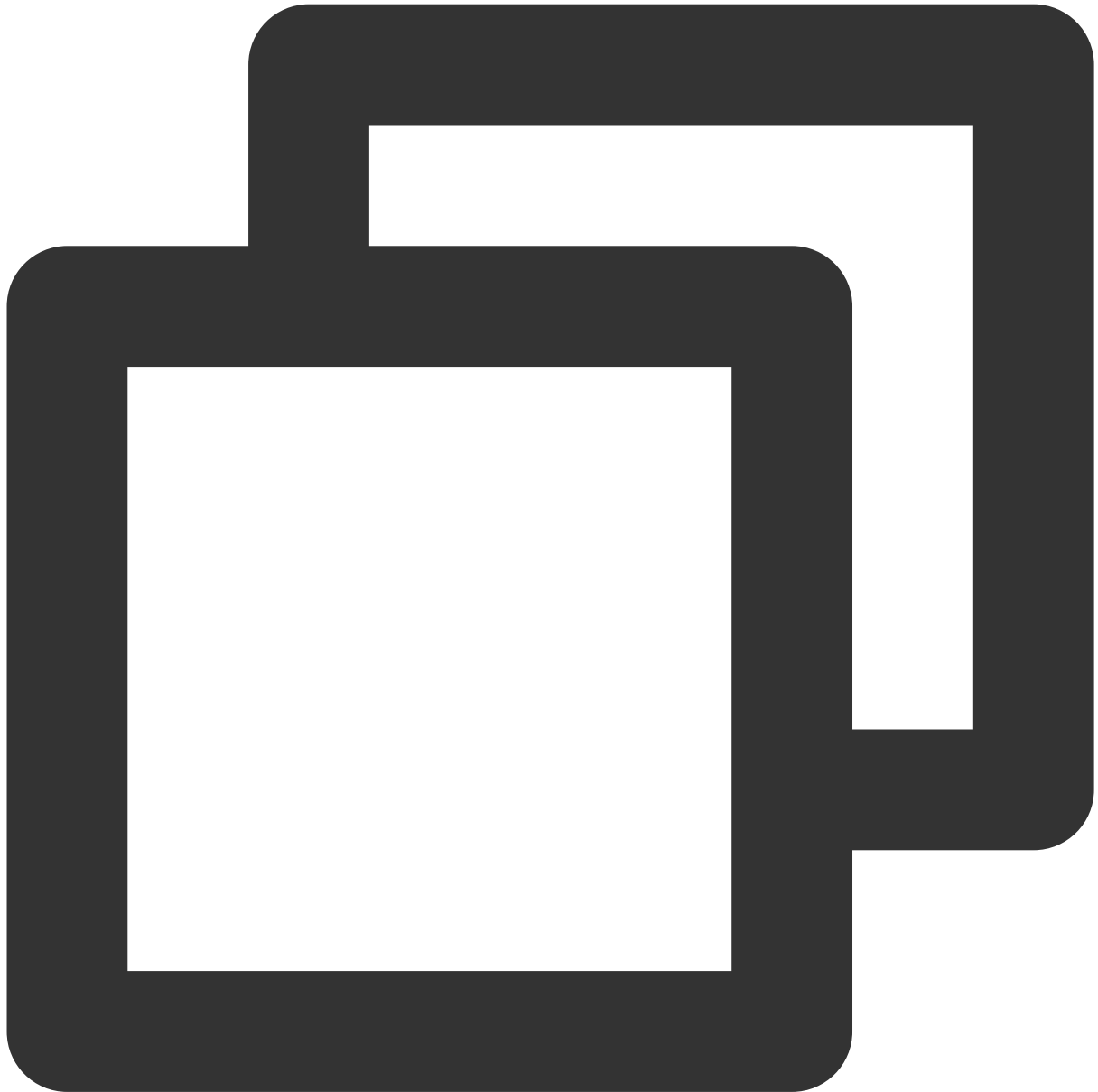
```
- (void)onUserJoin:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The ID of the user who joined the call.

## onUserLeave

A user left the call.



```
- (void)onUserLeave:(NSString *)userId;
```

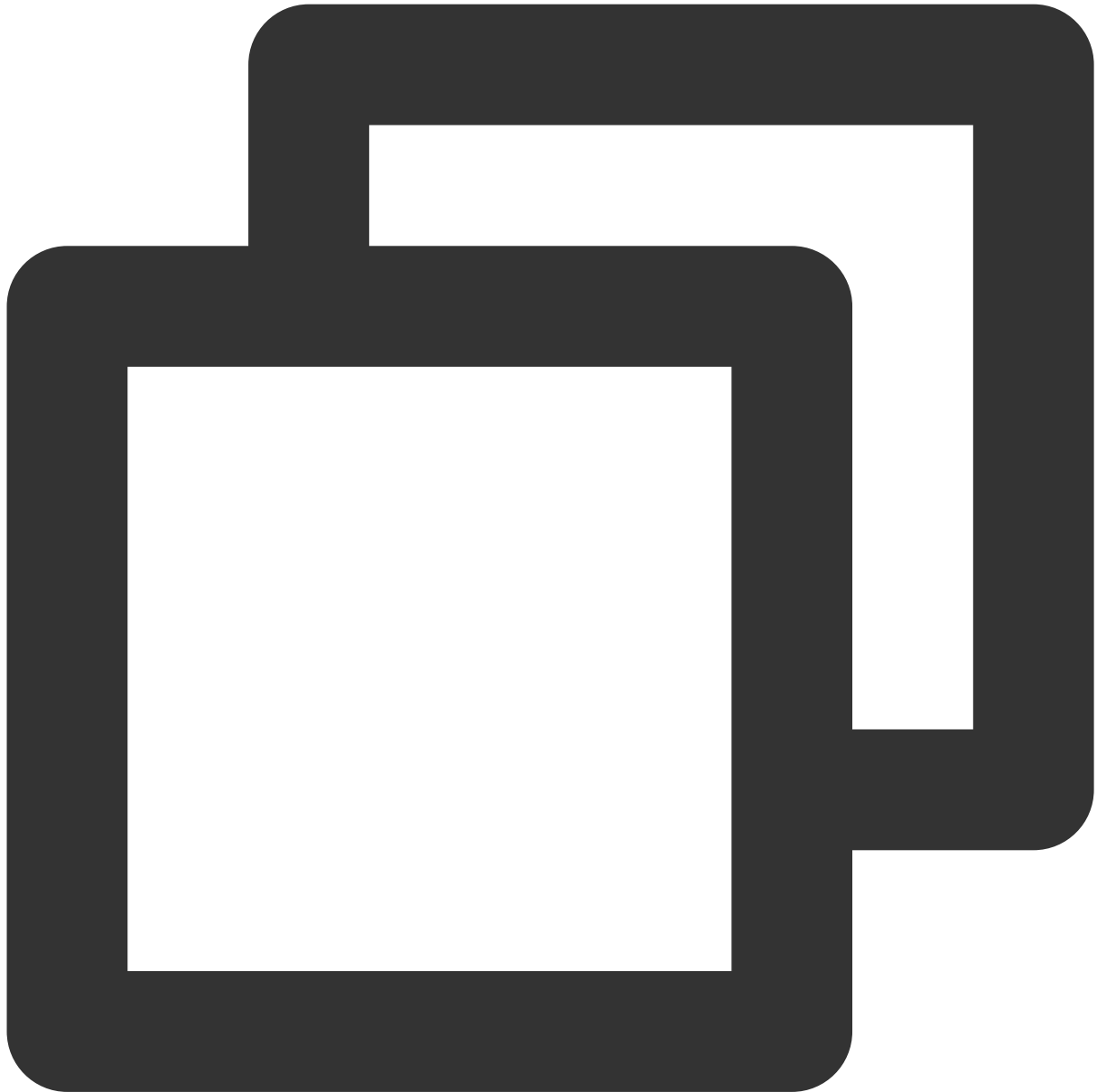
The parameters are described below:

Parameter	Type	Description
userId	NSString	The ID of the user who left the call.



## onUserAudioAvailable

Whether a user is sending audio.



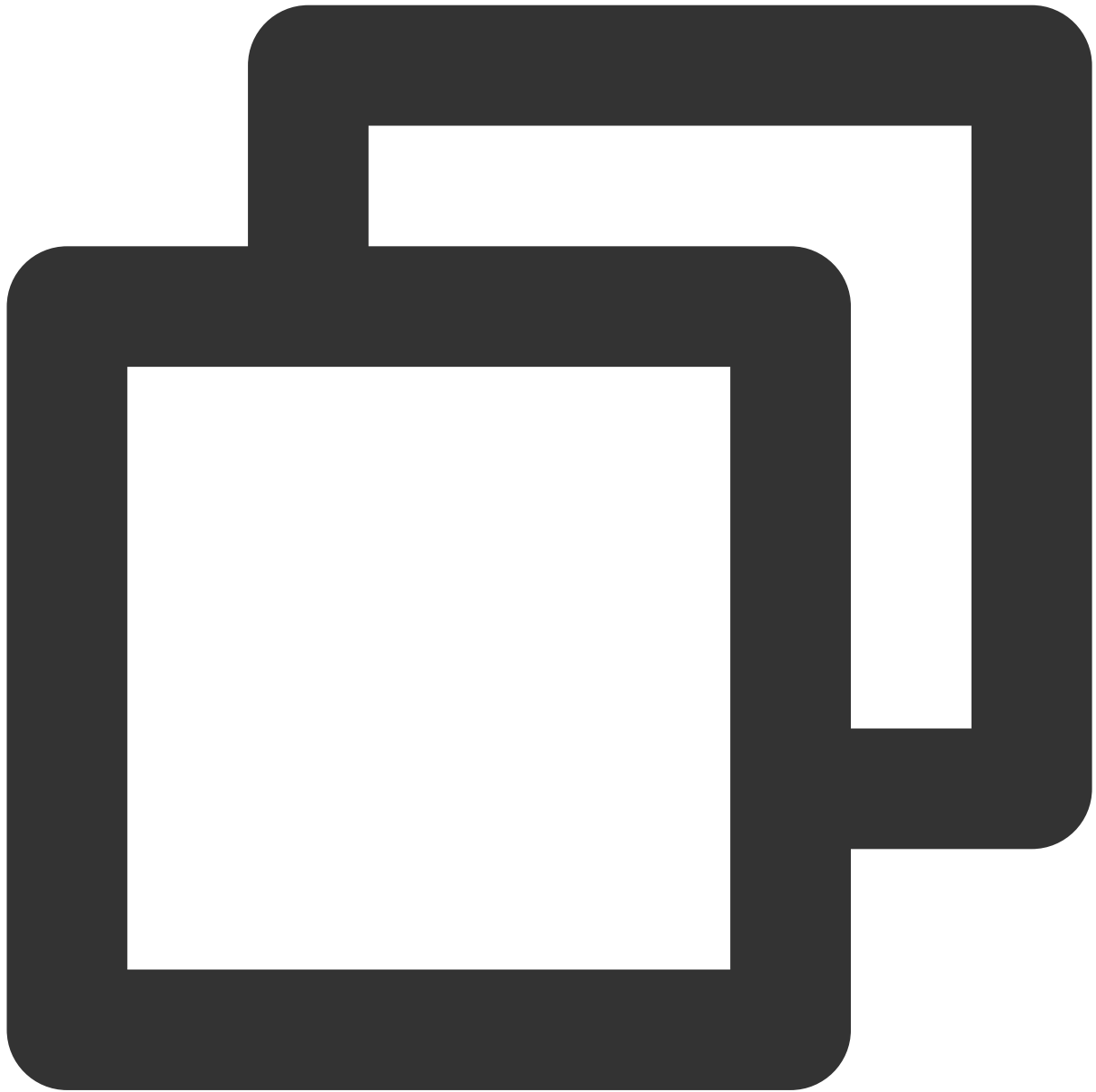
```
- (void)onUserAudioAvailable:(NSString *)userId isAudioAvailable:(BOOL)isAudioAvail
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.
isAudioAvailable	BOOL	Whether the user has audio.

## onUserVideoAvailable

Whether a user is sending video.



```
- (void)onUserVideoAvailable:(NSString *)userId isVideoAvailable:(BOOL)isVideoAvail
```

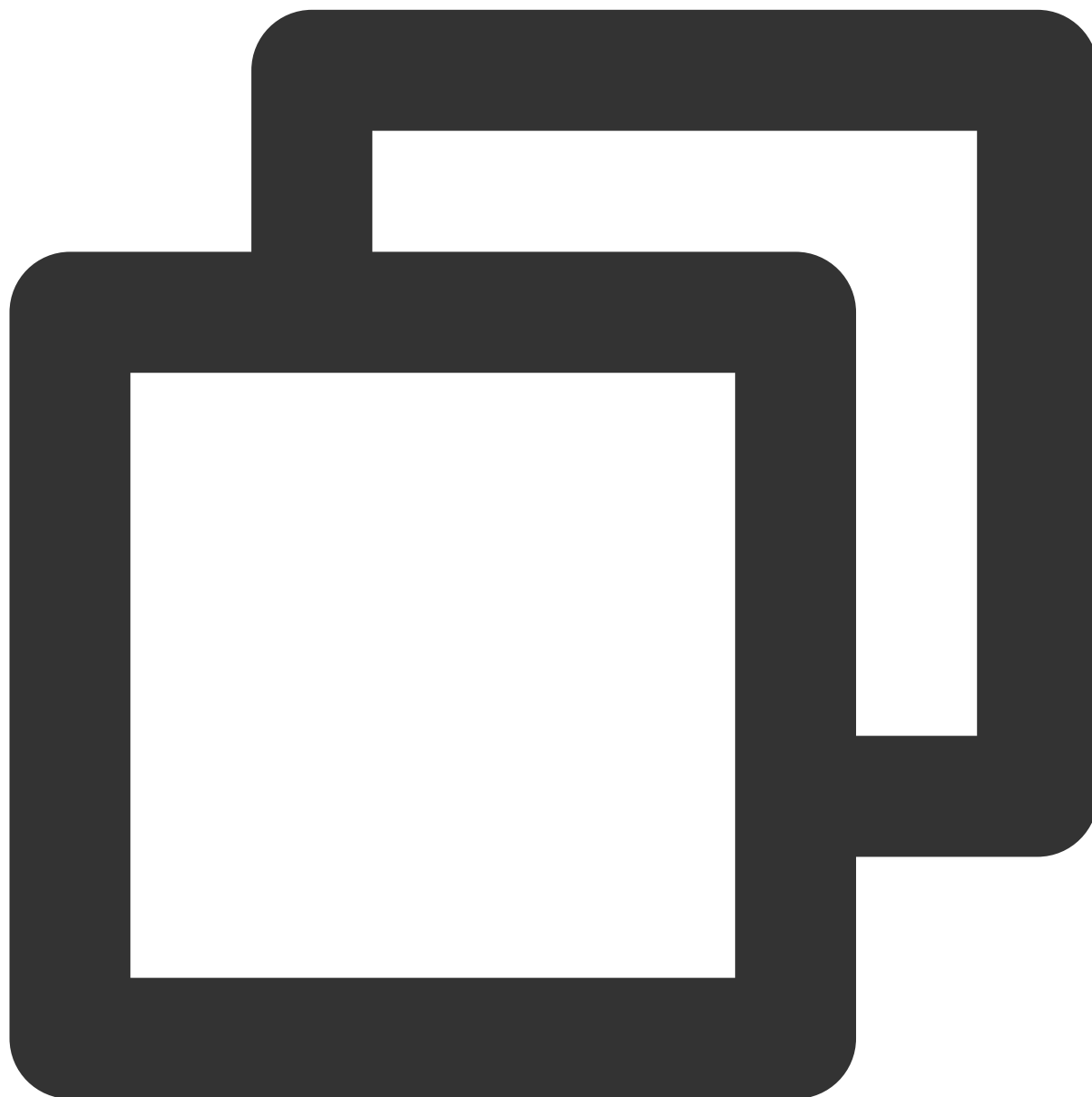
The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.

isVideoAvailable	BOOL	Whether the user has video.
------------------	------	-----------------------------

## onUserVoiceVolumeChanged

The volumes of all users.



```
- (void)onUserVoiceVolumeChanged:(NSDictionary <NSString *, NSNumber *> *)volumeMap
```

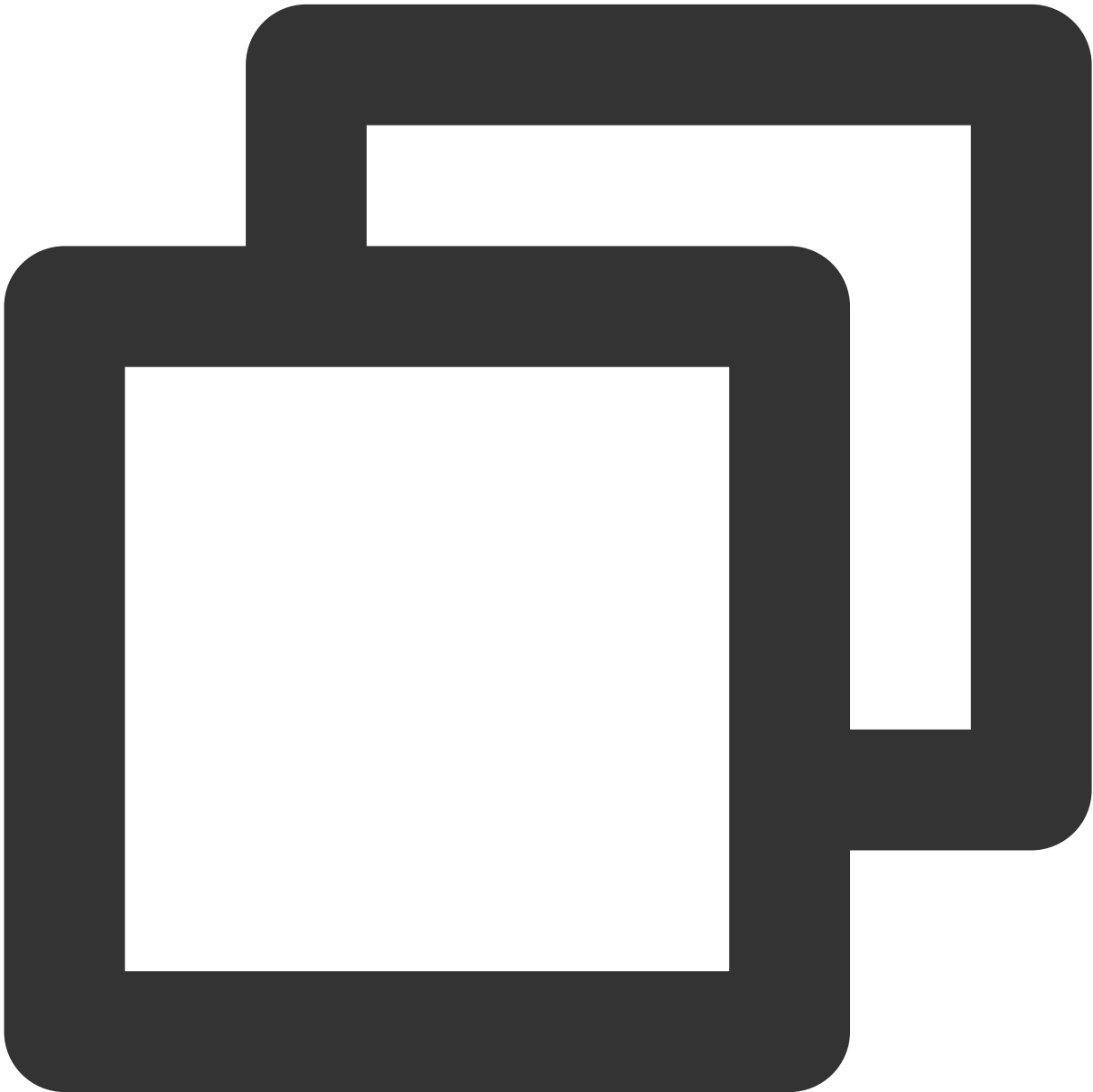
The parameters are described below:

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

volumeMap	NSDictionary	The volume table, which includes the volume of each user ( <code>userId</code> ). Value range: 0-100.
-----------	--------------	--

**onUserNetworkQualityChanged**

The network quality of all users.



```
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkQualityInfo *> *)networkQual
```

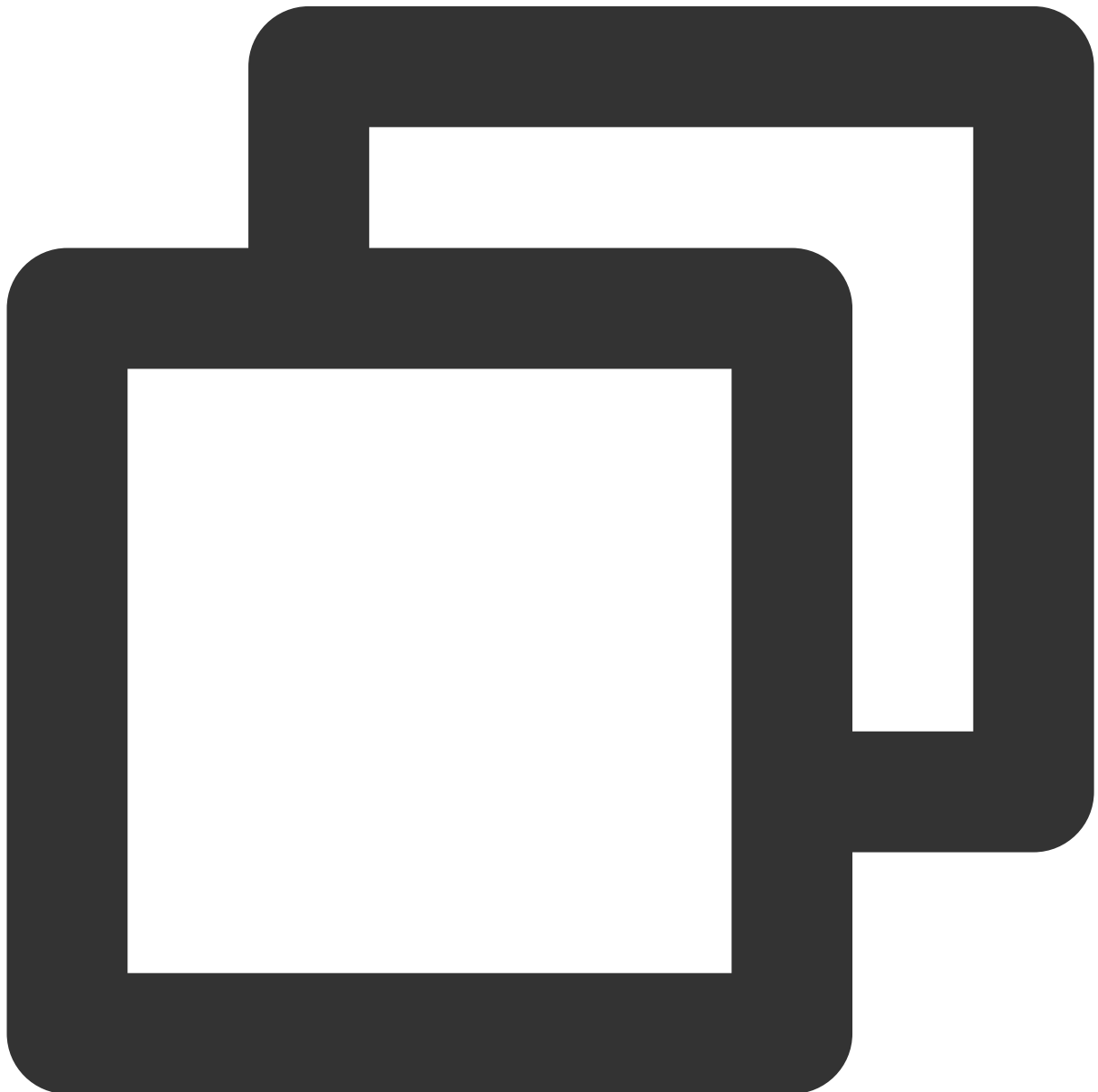
The parameters are described below:

--	--	--

Parameter	Type	Description
networkQualityList	NSArray	The current network conditions for all users ( <code>userId</code> ).

## onKickedOffline

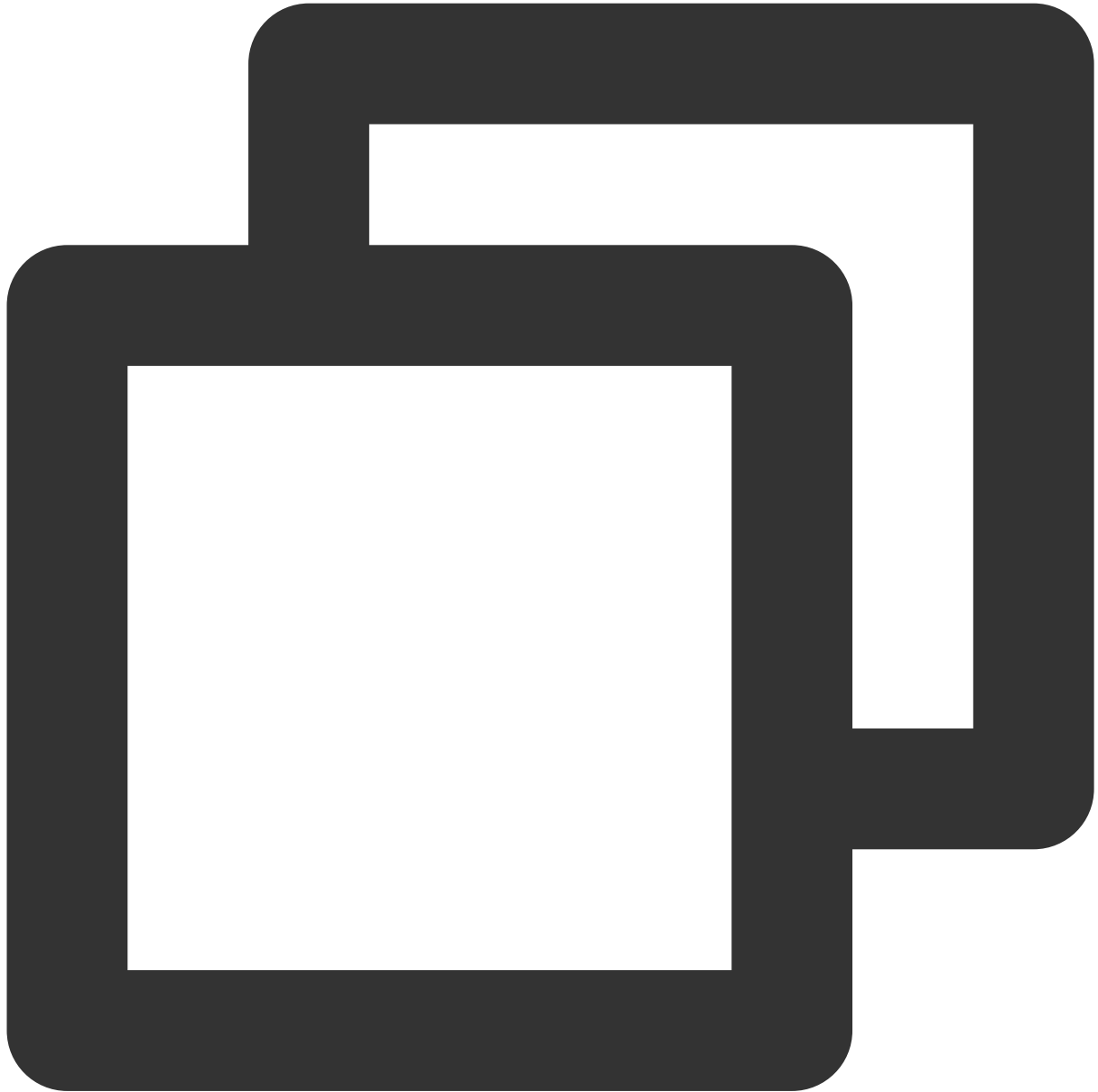
The current user was kicked offline : At this time, you can prompt the user with a UI message and then invoke `init` again.



```
- (void)onKickedOffline;
```

## onUserSigExpired

The user sig is expired : At this time, you need to generate a new `userSig` , and then invoke `init` again.



```
- (void)onUserSigExpired;
```

# Type Definition

Last updated : 2023-06-21 16:30:07

## Common structures

### TUICallDefine

Type	Description
<a href="#">TUICallParams</a>	An additional parameter.
<a href="#">TUIOfflinePushInfo</a>	Offline push vendor configuration information.

### TUICommonDefine

Type	Description
<a href="#">TUIRoomId</a>	Room ID for audio and video in a call.
<a href="#">TUINetworkQuality</a>	Network quality information
<a href="#">TUIVideoRenderParams</a>	Video render parameters
<a href="#">TUIVideoEncoderParams</a>	Video encoding parameters

## Enum definition

### TUICallDefine

Type	Description
<a href="#">TUICallMediaType</a>	Media type in a call
<a href="#">TUICallRole</a>	Roles of individuals in a call.
<a href="#">TUICallStatus</a>	The call status
<a href="#">TUICallScene</a>	The call scene
<a href="#">TUICallIOSOfflinePushType</a>	iOS offline push type

### TUICommonDefine

Type	Description

<a href="#">TUIAudioPlaybackDevice</a>	Audio route
<a href="#">TUICamera</a>	Camera type
<a href="#">TUINetworkQuality</a>	Network quality
<a href="#">TUIVideoRenderParamsFillMode</a>	Video image fill mode
<a href="#">TUIVideoRenderParamsRotation</a>	Video image rotation direction
<a href="#">TUIVideoEncoderParamsResolutionMode</a>	Video aspect ratio mode
<a href="#">TUIVideoEncoderParamsResolution</a>	Video resolution

## TUICallParams

Call params

参数	Type	Description
roomId	<a href="#">TUIRoomId</a>	Room ID for audio and video in a call.
offlinePushInfo	<a href="#">TUIOfflinePushInfo</a>	Offline push vendor configuration information.
timeout	int	Call timeout period, default: 30s, unit: seconds.
userData	NSString	An additional parameter. Callback when the callee receives <a href="#">onCallReceived</a>

## TUIOfflinePushInfo

Offline push vendor configuration information, please refer to : [Offline call push](#)

Value	Type	Description
title	NSString	offlinepush notification title
desc	NSString	offlinepush notification description
ignoreIOSBadge	BOOL	Ignore badge count for offline push (only for iOS), if set to true, the message will not increase the unread count of the app icon on the iOS receiver's side.
iOSSound	NSString	Offline push sound setting (only for iOS). When sound = <b>IOS_OFFLINE_PUSH_NO_SOUND</b> , there will be no sound played when the message is received. When sound = <b>IOS_OFFLINE_PUSH_DEFAULT_SOUND</b> ,



		the system sound will be played when the message is received. If you want to customize the iOSSound, you need to link the audio file into the Xcode project first, and then set the audio file name (with extension) to the iOSSound.
androidSound	NSString	Offline push sound setting (only for Android, supported by IMSDK 6.1 and above). Only Huawei and Google phones support setting sound prompts. For Xiaomi phones, please refer to: <a href="#">Xiaomi custom ringtones</a> . In addition, for Google phones, in order to set sound prompts for FCM push on Android 8.0 and above systems, you must call setAndroidFCMChannelID to set the channelId for it to take effect.
androidOPPOChannelID	NSString	Set the channel ID for OPPO phones with Android 8.0 and above systems.
androidVIVOClassification	NSInteger	Classification of VIVO push messages (deprecated interface, VIVO push service will optimize message classification rules on April 3, 2023. It is recommended to use setAndroidVIVOCategory to set the message category). 0: Operational messages, 1: System messages. The default value is 1.
androidXiaoMiChannelID	NSString	Set the channel ID for Xiaomi phones with Android 8.0 and above systems.
androidFCMChannelID	NSString	Set the channel ID for google phones with Android 8.0 and above systems.
androidHuaWeiCategory	NSString	Classification of Huawei push messages, please refer to: <a href="#">Huawei message classification standard</a> .
isDisablePush	BOOL	Whether to turn off push notifications (default is on).
iOSPushType	<a href="#">TUICallIOSOfflinePushType</a>	iOS offline push type, default is APNs

## TUIRoomId

Room ID for audio and video in a call.

**Note :**

(1) `intRoomId` and `strRoomId` are mutually exclusive. If you choose to use `strRoomId`, `intRoomId` needs to be filled in as 0. If both are filled in, the SDK will prioritize `intRoomId`.

(2) Do not mix `intRoomId` and `strRoomId` because they are not interchangeable. For example, the number 123 and the string "123" represent two completely different rooms.

Value	Type	Description
<code>intRoomId</code>	UInt32	Numeric room ID. <b>range:</b> 1 - 2147483647( $2^{31}-1$ )
<code>strRoomId</code>	NSString	String room ID. <b>range</b> : Limited to 64 bytes in length. The supported character set range is as follows (Lowercase and uppercase English letters. (a-zA-Z) Number (0-9) Spaces、 <code>!</code> 、 <code>#</code> 、 <code>\$</code> 、 <code>%</code> 、 <code>&amp;</code> 、 <code>(</code> 、 <code>)</code> 、 <code>+</code> 、 <code>-</code> 、 <code>:</code> 、 <code>;</code> 、 <code>&lt;</code> 、

#### Note :

Currently, string room number is only supported on Android and iOS platforms. Support for other platforms such as Web, Mini Programs, Flutter, and Uniapp will be available in the future. Please stay tuned!

## TUIVideoRenderParams

Video render parameters

Value	Type	Description
<code>fillMode</code>	<a href="#">TUIVideoRenderParamsFillMode</a>	Video image fill mode
<code>rotation</code>	<a href="#">TUIVideoRenderParamsRotation</a>	Video image rotation direction

## TUINetworkQualityInfo

User network quality information

Value	Type	Description
<code>userId</code>	NSString	user ID
<code>quality</code>	<a href="#">NetworkQuality</a>	network quality

## TUIVideoEncoderParams

Video encoding parameters

Value	Type	Description
resolution	<a href="#">TUIVideoEncoderParamsResolution</a>	Video resolution
resolutionMode	<a href="#">TUIVideoEncoderParamsResolutionMode</a>	Video aspect ratio mode

## TUICallMediaType

Call media type

Type	Value	Description
TUICallMediaTypeUnknown	0	Unknown
TUICallMediaTypeAudio	1	Audio call
TUICallMediaTypeVideo	2	Video call

## TUICallRole

Call role

Type	Value	Description
TUICallRoleNone	0	Unknown
TUICallRoleCall	1	Caller (inviter)
TUICallRoleCalled	2	Callee (invitee)

## TUICallStatus

Call status

Type	Value	Description
TUICallStatusNone	0	Unknown
TUICallStatusWaiting	1	The call is currently waiting
TUICallStatusAccept	2	The call has been accepted

## TUICallScene

Call scene

Type	Value	Description

TUICallSceneGroup	0	Group call
TUICallSceneMulti	1	Anonymous group calling (not supported at this moment, please stay tuned).
TUICallSceneSingle	2	one to one call

## TUICallIOSOfflinePushType

iOS offline push type

Type	Value	Description
TUICallIOSOfflinePushTypeAPNs	0	APNs
TUICallIOSOfflinePushTypeVoIP	1	VoIP

## TUIAudioPlaybackDevice

Audio route

Type	Value	Description
TUIAudioPlaybackDeviceSpeakerphone	0	Speakerphone
TUIAudioPlaybackDeviceEarpiece	1	Earpiece

## TUICamera

Front/Back camera

Type	Value	Description
TUICameraFront	0	Front camera
TUICameraBack	1	Back camera

## TUINetworkQuality

Network quality

Type	Value	Description
TUINetworkQualityUnknown	0	Unknown
TUINetworkQualityExcellent	1	Excellent

TUINetworkQualityGood	2	Good
TUINetworkQualityPoor	3	Poor
TUINetworkQualityBad	4	Bad
TUINetworkQualityVbad	5	Vbad
TUINetworkQualityDown	6	Down

### TUIVideoRenderParamsFillMode

If the aspect ratio of the video display area is not equal to that of the video image, you need to specify the fill mode:

Type	Value	Description
TUIVideoRenderParamsFillModeFill	0	Fill mode: the video image will be centered and scaled to fill the entire display area, where parts that exceed the area will be cropped. The displayed image may be incomplete in this mode.
TUIVideoRenderParamsFillModeFit	1	Fit mode: the video image will be scaled based on its long side to fit the display area, where the short side will be filled with black bars. The displayed image is complete in this mode, but there may be black bars.

### TUIVideoRenderParamsRotation

We provides rotation angle setting APIs for local and remote images. The following rotation angles are all clockwise.

Type	Value	Description
TUIVideoRenderParamsRotation_0	0	No rotation
TUIVideoRenderParamsRotation_90	1	Clockwise rotation by 90 degrees
TUIVideoRenderParamsRotation_180	2	Clockwise rotation by 180 degrees
TUIVideoRenderParamsRotation_270	3	Clockwise rotation by 0 degrees

### TUIVideoEncoderParamsResolutionMode

Video aspect ratio mode

Type	Value	Description

TUIVideoEncoderParamsResolutionModeLandscape	0	Landscape resolution, such as : TUIVideoEncoderParamsResolution_640_360 TUIVideoEncoderParamsResolutionModeLand: = 640 × 360
TUIVideoEncoderParamsResolutionModePortrait	1	Portrait resolution, such as : TUIVideoEncoderParamsResolution_640_360 TUIVideoEncoderParamsResolutionModePortr: 360 × 640

## TUIVideoEncoderParamsResolution

Video resolution

Type	Value	Description
TUIVideoEncoderParamsResolution_640_360	108	Aspect ratio: 16:9 ; resolution: 640x360 ; recommended bitrate: 500kbps
TUIVideoEncoderParamsResolution_640_480	62	Aspect ratio: 4:3 ; resolution: 640x480 ; recommended bitrate: 600kbps
TUIVideoEncoderParamsResolution_960_540	110	Aspect ratio: 16:9 ; resolution: 960x540 ; recommended bitrate: 850kbps
TUIVideoEncoderParamsResolution_960_720	64	Aspect ratio: 4:3 ; resolution: 960x720 ; recommended bitrate: 1000kbps
TUIVideoEncoderParamsResolution_1280_720	112	Aspect ratio: 16:9 ; resolution: 1280x720 ; recommended bitrate: 1200kbps
TUIVideoEncoderParamsResolution_1920_1080	114	Aspect ratio: 16:9 ; resolution: 1920x1080 ; recommended bitrate: 2000kbps

# Web

## API Overview

Last updated : 2024-04-03 17:23:11

### TUICallKit (Includes UI Components)

TUICallKit is an audio and video call component that **includes a UI component**. You can quickly implement a WhatsApp-like audio and video calling scenario with this component.

<TUICallKit/>: The core UI call component.

TUICallKitServer is the call instance, offering the following API interfaces.

API	Description
<a href="#">init</a>	Initializing TUICallKit.
<a href="#">call</a>	Initiate a one-on-one call
<a href="#">groupCall</a>	Initiate a group call
<a href="#">joinInGroupCall</a>	Actively join the ongoing group call
<a href="#">setCallingBell</a>	Establish a personalised ringtone for incoming calls
<a href="#">setSelfInfo</a>	Configure the user's nickname and profile photo
<a href="#">enableMuteMode</a>	Toggle On/Off the ringtone for incoming calls
<a href="#">enableFloatWindow</a>	Activate/Deactivate the floating window function
<a href="#">setLanguage</a>	Set the call language for the TUICallKit component
<a href="#">destroyed</a>	Terminating TUICallKit

### TUICallEngine (No UI)

TUICallEngine API is an audio and video call component that **offers a No UI interface**. You can use this set of APIs to custom encapsulate according to your business needs.

API	Description
<a href="#">createInstance</a>	Creating a TUICallEngine Instance (Singleton Pattern)

<a href="#">destroyInstance</a>	Terminating a TUICallEngine Instance (Singleton Pattern)
<a href="#">on</a>	Listening on events
<a href="#">off</a>	Canceling Event Listening
<a href="#">login</a>	Sign in Interface
<a href="#">logout</a>	Logout Interface
<a href="#">setSelfInfo</a>	Configure the user's nickname and profile photo
<a href="#">call</a>	Initiate a one-on-one call
<a href="#">groupCall</a>	Group Chat Invitation Call
<a href="#">accept</a>	Answer Calls
<a href="#">reject</a>	Decline Call
<a href="#">hangup</a>	End Calls
<a href="#">switchCallMediaType</a>	Switch Audio and Video Calls
<a href="#">startRemoteView</a>	Initiate Remote Screen Rendering
<a href="#">stopRemoteView</a>	Stop Remote Screen Rendering
<a href="#">startLocalView</a>	Start Local Screen Rendering, <b>Note: This will be deprecated; use openCamera instead</b>
<a href="#">stopLocalView</a>	Stop Local Screen Rendering, <b>Note: This will be deprecated; use closeCamera instead</b>
<a href="#">openCamera</a>	Enable the camera
<a href="#">closeCamara</a>	Turn Off Camera
<a href="#">openMicrophone</a>	Enable Microphone
<a href="#">closeMicrophone</a>	Turn off the microphone
<a href="#">setVideoQuality</a>	Set video quality
<a href="#">getDeviceList</a>	Access device list
<a href="#">switchDevice</a>	Switch camera or microphone devices
<a href="#">enableAIVoice</a>	Enable/disable AI noise reduction



## Event Types

TUICallEvent is the callback event class corresponding to TUICallEngine. Through this callback, you can listen to the callback events of interest.

EVENT	Description
<code>TUICallEvent.ERROR</code>	An error occurred inside the SDK
<code>TUICallEvent.SDK_READY</code>	This callback is received when the SDK enters the Ready State
<code>TUICallEvent.KICKED_OUT</code>	Duplicate Sign in, receiving this callback indicates being Kicked out of Room
<code>TUICallEvent.USER_ACCEPT</code>	If a user answers, this callback will be received
<code>TUICallEvent.USER_ENTER</code>	If a user agrees to join the call, this callback will be received
<code>TUICallEvent.USER_LEAVE</code>	If a user agrees to leave the call, this callback will be received
<code>TUICallEvent.REJECT</code>	User declines the call
<code>TUICallEvent.NO_RESP</code>	Invite user, no response
<code>TUICallEvent.LINE_BUSY</code>	Invitee Busy Line
<code>TUICallEvent.CALLING_TIMEOUT</code>	As an invitee, receiving this callback indicates that the call has timed out without an answer
<code>TUICallEvent.USER_VIDEO_AVAILABLE</code>	Remote User turns Camera On/Off, this callback will be received
<code>TUICallEvent.USER_AUDIO_AVAILABLE</code>	Remote User turns Microphone On/Off, this callback will be received
<code>TUICallEvent.USER_VOICE_VOLUME</code>	Remote User Speech Volume Adjustment, this callback will be received
<code>TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE</code>	Group Chat Update, Invitation List this callback will be received
<code>TUICallEvent.INVITED</code>	Invited for a call
<code>TUICallEvent.CALLING_CANCEL</code>	As an invitee, receiving this callback indicates that the call has been canceled

<a href="#">TUICallEvent.CALLING_END</a>	Receiving this callback indicates that the call has ended
<a href="#">TUICallEvent.DEVICED_UPDATED</a>	Device list update, this callback will be received
<a href="#">TUICallEvent.CALL_TYPE_CHANGED</a>	This callback is received when switching call types

## Document Link

[TUICallEngine](#)

[TUICallEvent](#)

# TUICallKit

Last updated : 2024-04-03 17:23:11

## API Introduction

The TUICallKit API is the **audio and video call component that includes a UI interface**. With the TUICallKit API, you can swiftly develop audio and video call scenarios reminiscent of WeChat through simple interfaces. For further comprehensive steps to access this, please refer to: [Swift Access to TUICallKit](#).

## API Overview

TUICallKit is the **audio and video call component with a UI**, which enables you to swiftly create scenarios akin to WeChat for voice and video calls.

<TUICallKit/>: The core UI call component.

TUICallKitServer is the call instance, offering the following API interfaces.

API	Description
<a href="#">init</a>	Initialize the `TUICallKit` component instance
<a href="#">call</a>	Initiate a one-on-one call
<a href="#">groupCall</a>	Initiate a group call
<a href="#">joinInGroupCall</a>	Actively join the ongoing group call
<a href="#">setCallingBell</a>	Establish a personalised ringtone for incoming calls
<a href="#">setSelfInfo</a>	Configure the user's nickname and profile photo
<a href="#">enableMuteMode</a>	Toggle On/Off the ringtone for incoming calls
<a href="#">enableFloatWindow</a>	Activate/Deactivate the floating window function
<a href="#">setLanguage</a>	Set the call language for the TUICallKit component
<a href="#">destroyed</a>	Terminate the TUICallKit component instance

## Introduction to `<TUICallKit/>` attributes

## Attribute Overview

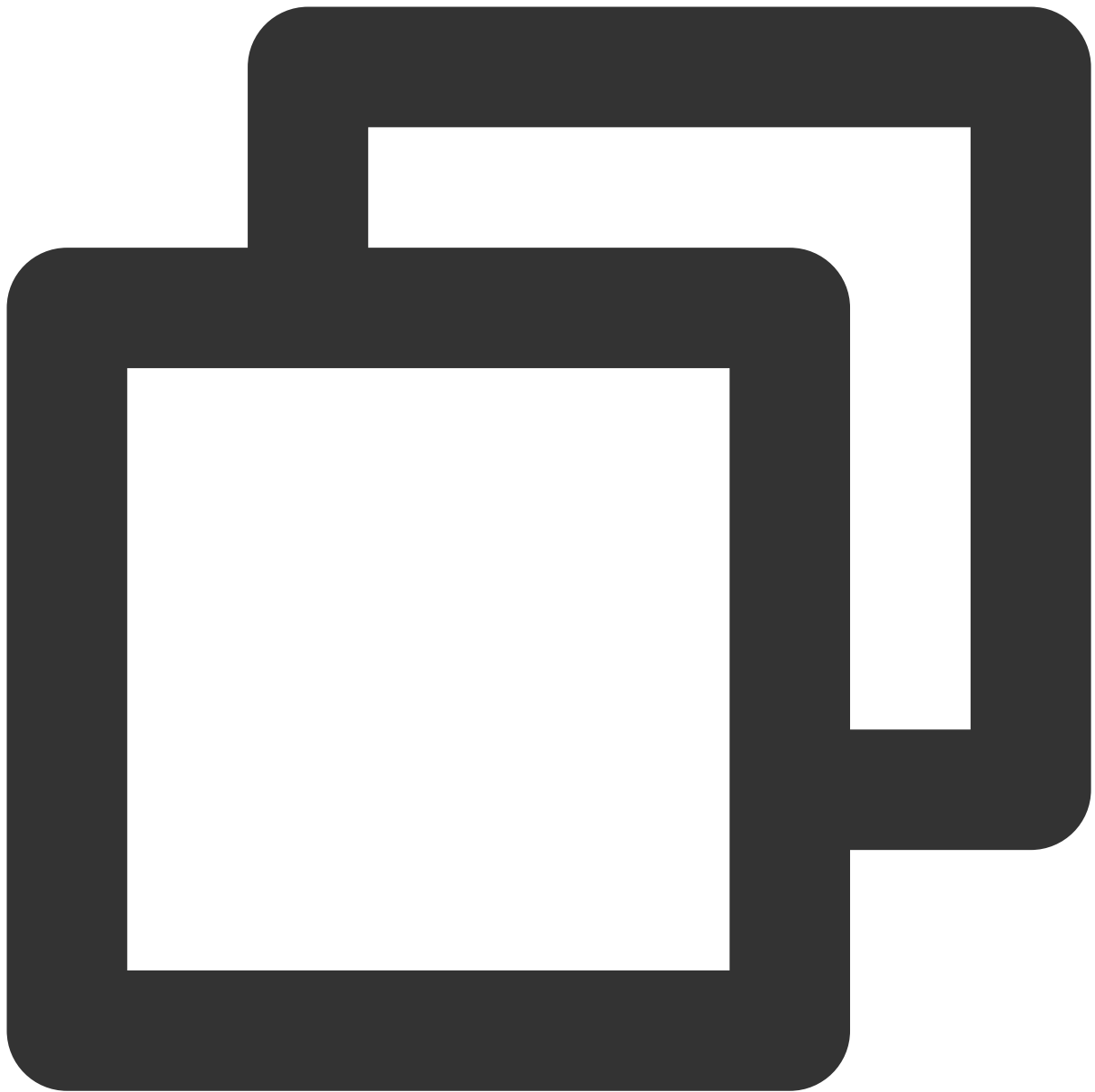
Attribute	Description	Type	Required	Default Value
allowedMinimized	Is the floating window permitted?	boolean	No	false
allowedFullScreen	Whether to permit full screen mode for the call interface	boolean	No	true
videoDisplayMode	Display mode for the call interface	VideoDisplayMode	No	VideoDisplayMode.COVER
videoResolution	Call Resolution	VideoResolution	No	VideoResolution.RESOLUTION_48
beforeCalling	This function will be executed prior to making a call and before receiving an invitation to talk	function(type, error)	No	-
afterCalling	This function will be executed after the termination of the call	function()	No	-
onMinimized	This function will be executed when the component switches to a minimized	function(oldStatus, newStatus)	No	-

	state. The explanation for the <a href="#">STATUS value</a> is			
kickedOut	The events thrown by the component occur when the current logged-in user is ejected. The call will also automatically terminate	function()	No	-
statusChanged	Event thrown by the component; this event is triggered when the call status changes. For detailed types of call status, refer to <a href="#">STATUS value description</a>	function({oldStatus, newStatus})	No	-

## Sample code

React

Vue

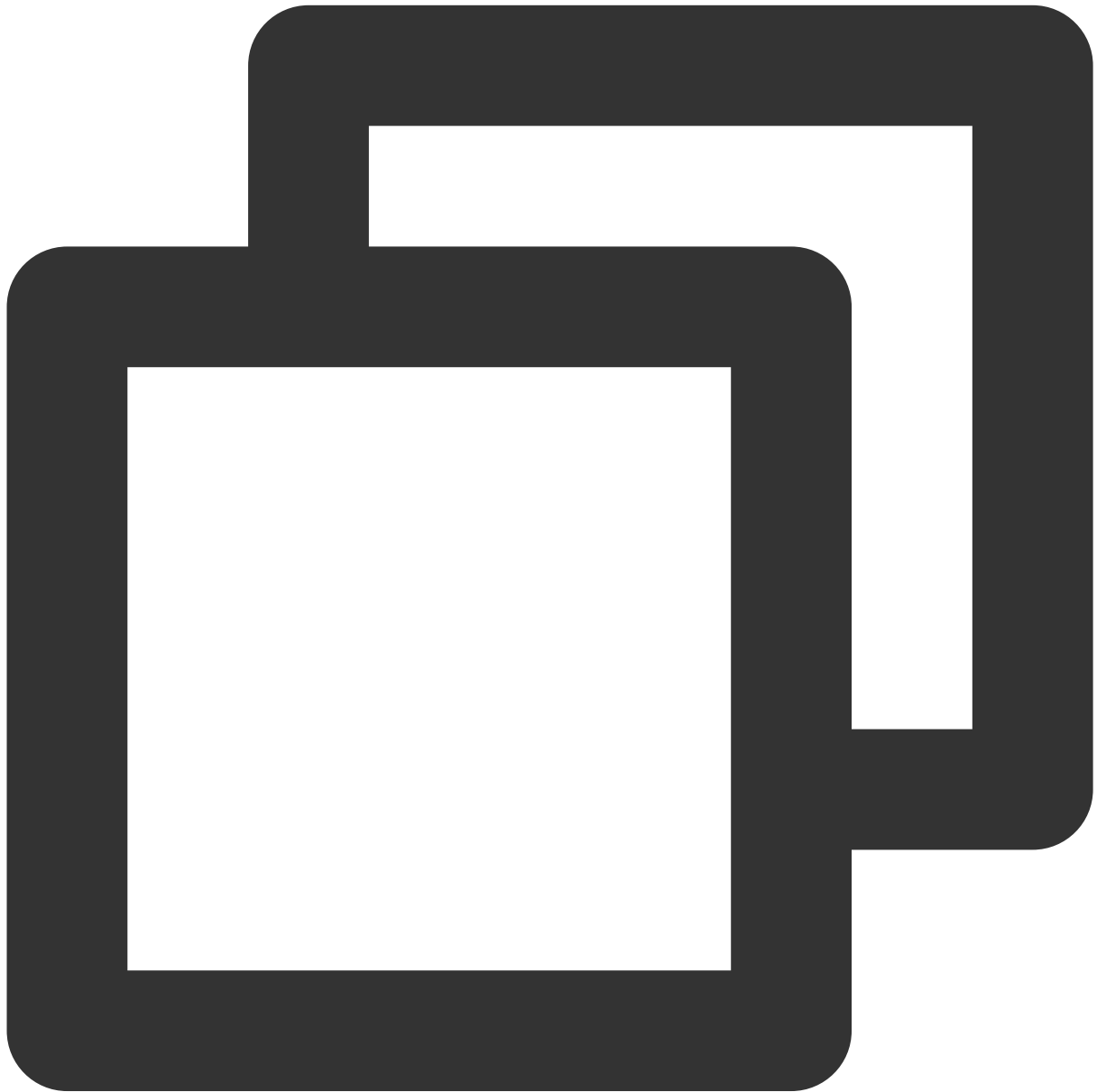


```
import { TUICallKit, VideoDisplayMode, VideoResolution } from "@tencentcloud/call-u

<TUICallKit
  videoDisplayMode={VideoDisplayMode.CONTAIN}
  videoResolution={VideoResolution.RESOLUTION_1080P}
  beforeCalling={handleBeforeCalling}
  afterCalling={handleAfterCalling}
/>

function handleBeforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] handleBeforeCalling:", type, error);
```

```
}  
function handleAfterCalling() {  
    console.log("[TUICallkit Demo] handleAfterCalling");  
}
```



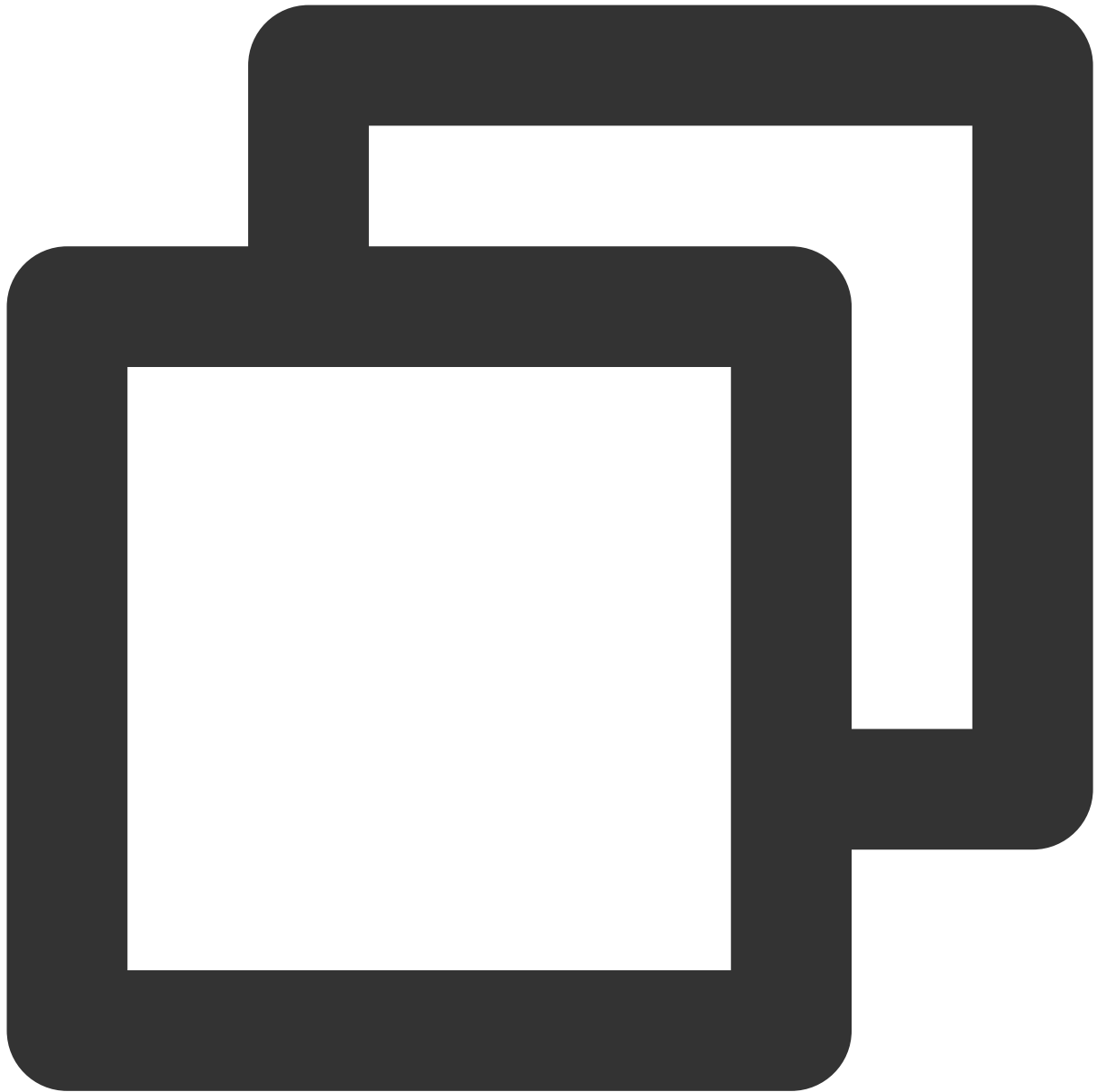
```
<template>  
  <TUICallKit  
    :allowedMinimized="true"  
    :allowedFullScreen="true"  
    :videoDisplayMode="VideoDisplayMode.CONTAIN"  
    :videoResolution="VideoResolution.RESOLUTION_1080P"
```

```
    :beforeCalling="beforeCalling"
    :afterCalling="afterCalling"
    :onMinimized="onMinimized"
    :kickedOut="handleKickedOut"
    :statusChanged="handleStatusChanged"
  />
</template>
<script lang="ts" setup>
import { TUICallKit, TUICallKitServer, VideoDisplayMode, VideoResolution, STATUS }

function beforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] beforeCalling:", type, error);
}
function afterCalling() {
  console.log("[TUICallkit Demo] afterCalling");
}
function onMinimized(oldStatus: string, newStatus: string) {
  console.log("[TUICallkit Demo] onMinimized: " + oldStatus + " -> " + newStatus);
}
function kickedOut() {
  console.log("[TUICallkit Demo] kickedOut");
}
function statusChanged(args: { oldStatus: string; newStatus: string; }) {
  const { oldStatus, newStatus } = args;
  if (newStatus === STATUS.CALLING_C2C_VIDEO) {
    console.log(`[TUICallkit Demo] statusChanged: ${oldStatus} -> ${newStatus}`);
  }
}
</script>
```

## Detailed information on TUICallKitServer API

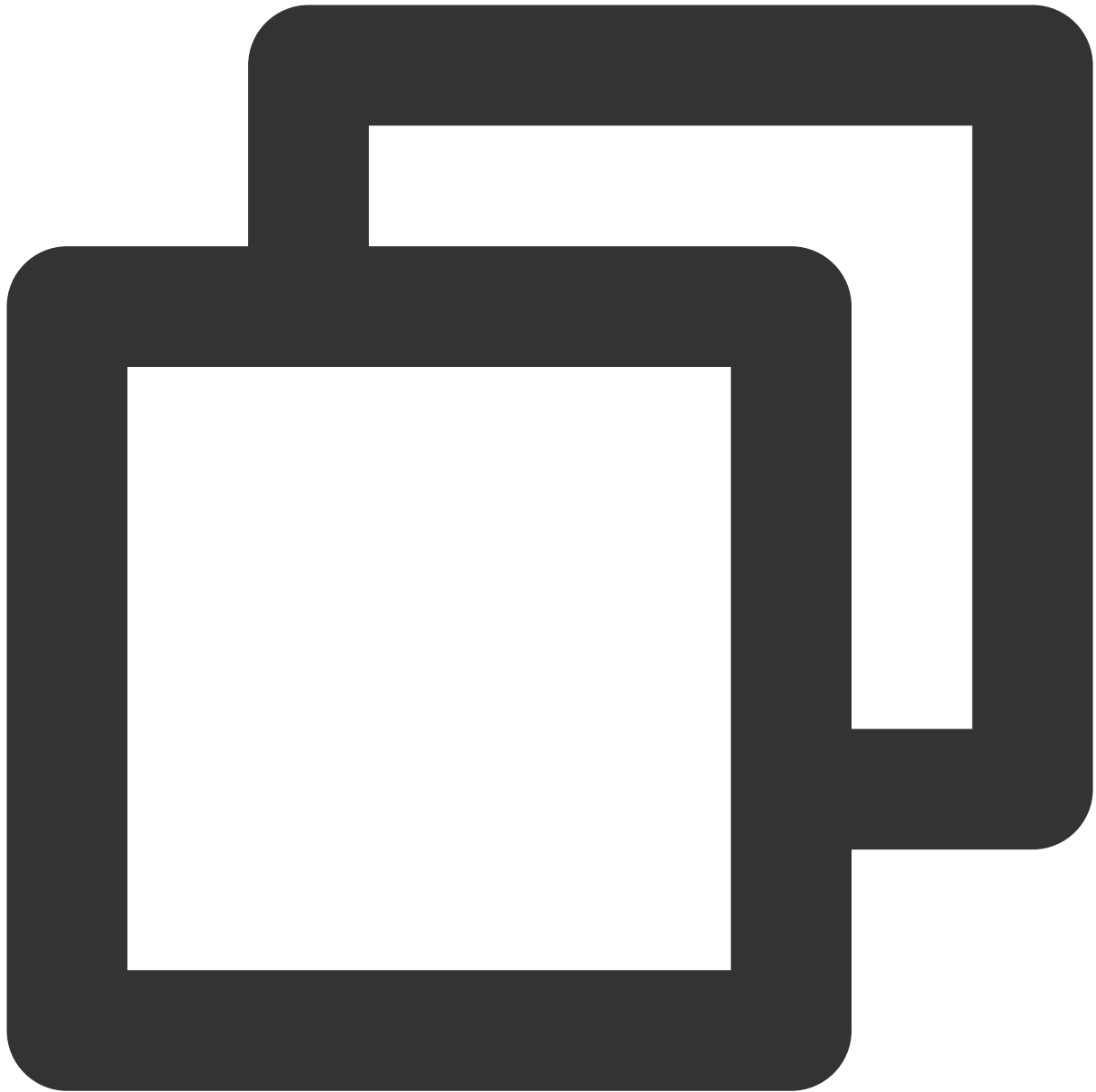




```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";  
// Replace it with the call-uikit npm package you are currently using
```

## init

Initialize TUICallKit, which should be accomplished prior to 'call' and 'groupCall'.



```
try {
  await TUICallKitServer.init({ SDKAppID, userID, userSig });
  // If you already have a tim instance in your project, you need to pass it in here
  // await TUICallKitServer.init({ tim, SDKAppID, userID, userSig });
  alert("[TUICallKit] Initialization succeeds.");
} catch (error: any) {
  alert(`[TUICallKit] Initialization failed. Reason: ${error}`);
}
```

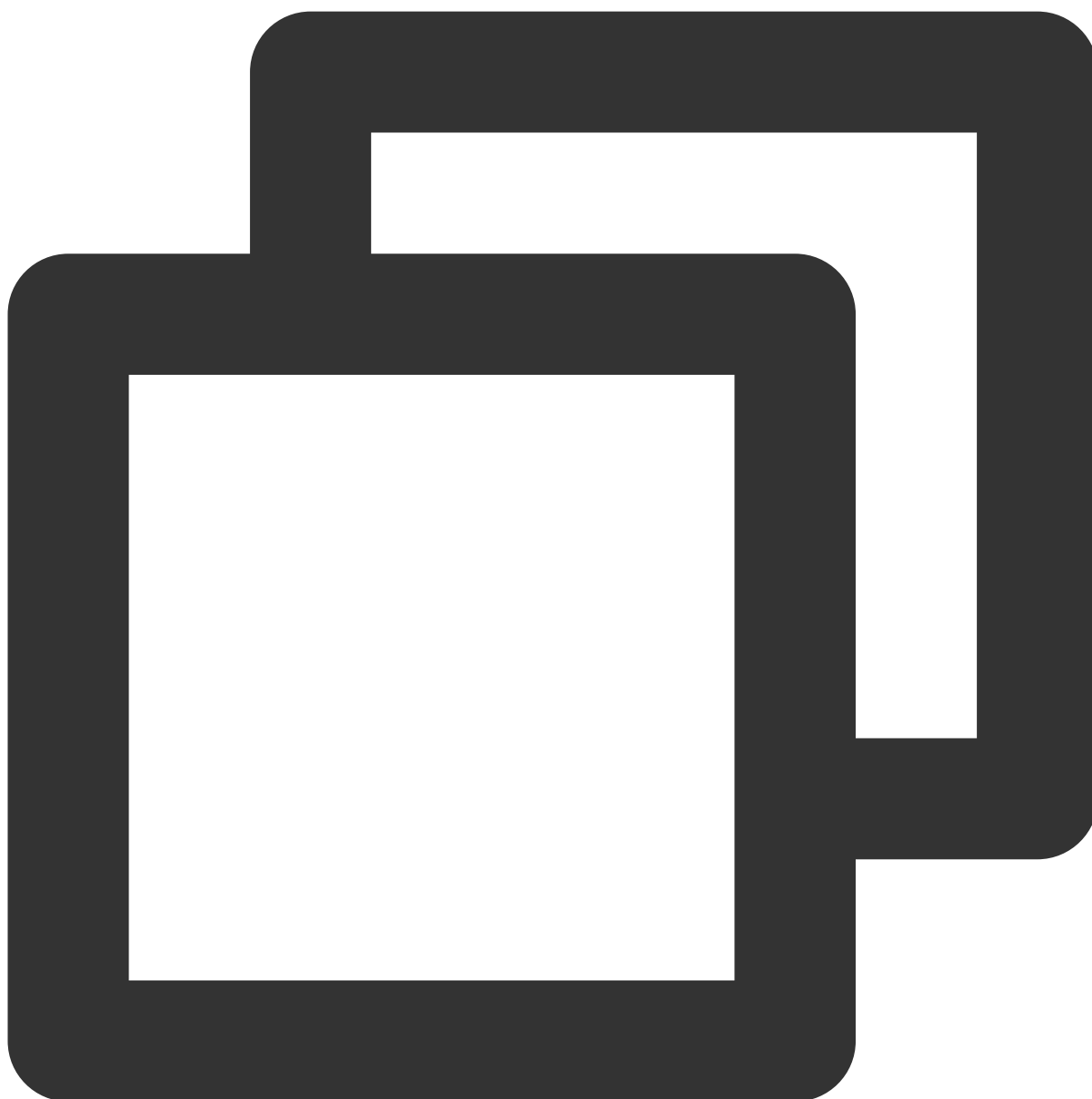
The parameters are described below:

--	--	--	--

Parameter	Type	Required	Meaning
SDKAppID	Number	Yes	SDKAppID of the cloud communication application
userID	String	Yes	The current user's ID is of string type, only allowing for the inclusion of English letters (a-z and A-Z), digits (0-9), hyphens (-) and underscores (_)
userSig	String	Yes	A safeguard signature designed by Tencent Cloud. For acquisition method, kindly refer to the calculation of <a href="#">UserSig</a>
tim	Any	No	The tim parameter is viable if a TIM instance already exists in the business, to ensure the uniqueness of the TIM instance

## call

Initiate a one-on-one call.



```
try {
  await TUICallKitServer.call({
    userID: callUserID,
    type: TUICallType.VIDEO_CALL,
  });
} catch (error: any) {
  alert(`[TUICallKit] Call failed. Reason: ${error}`);
}
```

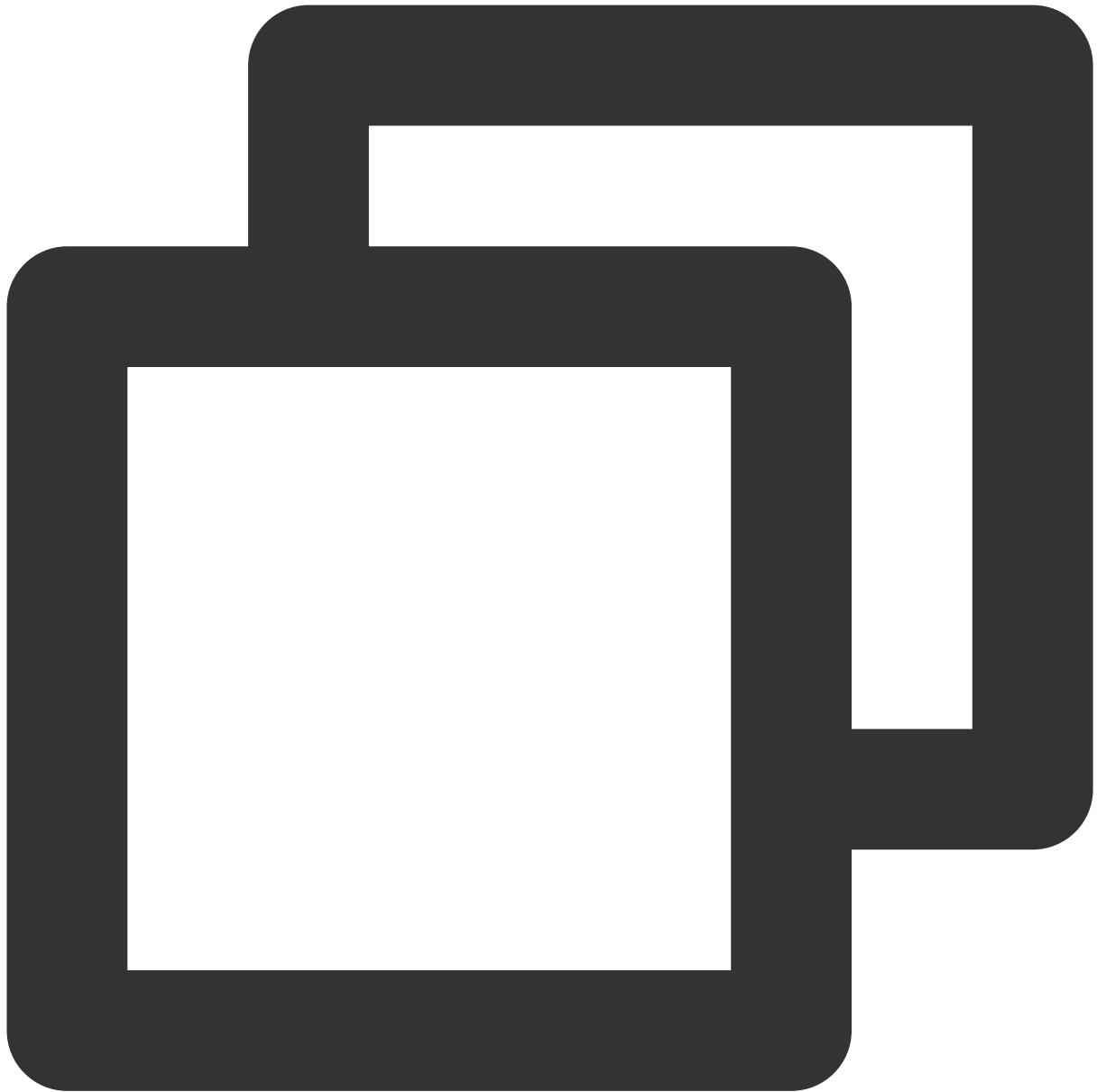
The parameters are described below:

--	--	--	--

Parameter	Type	Required	Meaning
userID	String	Yes	target user's userId
type	<a href="#">TUICallType</a>	Yes	The media type of the call, see <a href="#">TUICallType call type</a> for parameter value specifications
roomId	Number	No	Numerical Room ID, range [1, 2147483647]
timeout	Number	No	Call timeout duration, 0 signifies no timeout, unit s(seconds) (optional) - Default 30s
userData	String	No	Extension field: Used to enhance the invitation signal with additional information
<a href="#">offlinePushInfo</a>	Object	No	Customize offline message pushing

## groupCall

Initiate group communication.



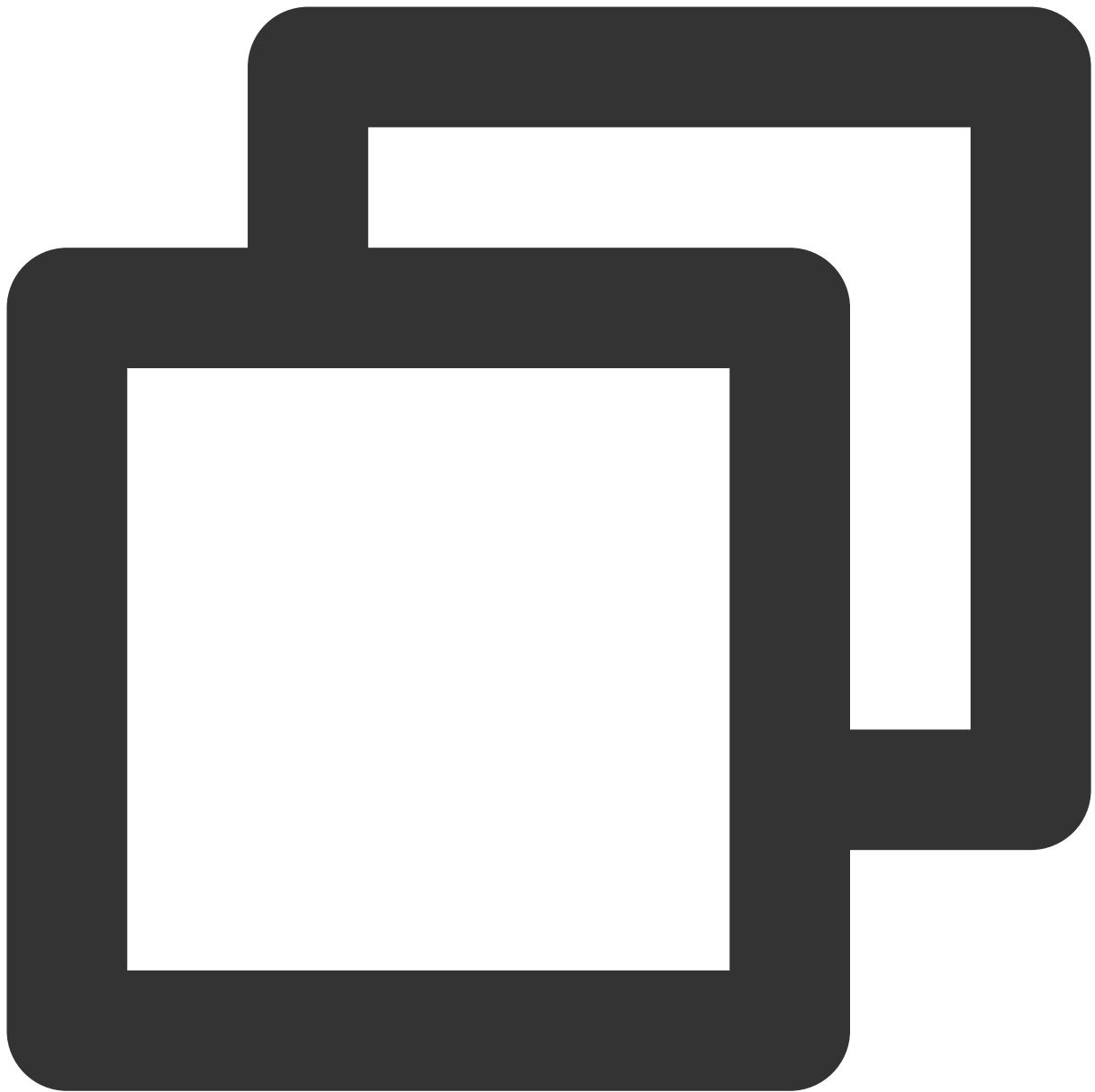
```
try {
  await TUICallKitServer.groupCall({
    userIDList: ['jack', 'tom'],
    groupId: "xxx",
    type: TUICallType.VIDEO_CALL
  });
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the groupCall API. Reason:${error}`);
}
```

The parameters are described below:

Parameter	Type	Required	Meaning
userIDList	Array<String>	Yes	Invitation list member lists
type	<a href="#">TUICallType</a>	Yes	The type of media for the call, you can refer to <a href="#">TUICallType for the explanation of parameter values</a>
groupID	String	Yes	Call group ID, the creation of groupID can be referred to <a href="#">chat-createGroup API</a>
roomID	Number	No	Numerical Room ID, range [1, 2147483647]
timeout	Number	No	Call timeout duration, 0 signifies no timeout, unit s(seconds) (optional) - Default 30s
userData	String	No	Extended field: Utilized for amplifying details in the invitation signaling
<a href="#">offlinePushInfo</a>	Object	No	Customize offline message pushing

## setLanguage

Language configuration.



```
TUICallKitServer.setLanguage("zh-cn"); // "en" | "zh-cn"
```

The parameters are described below:

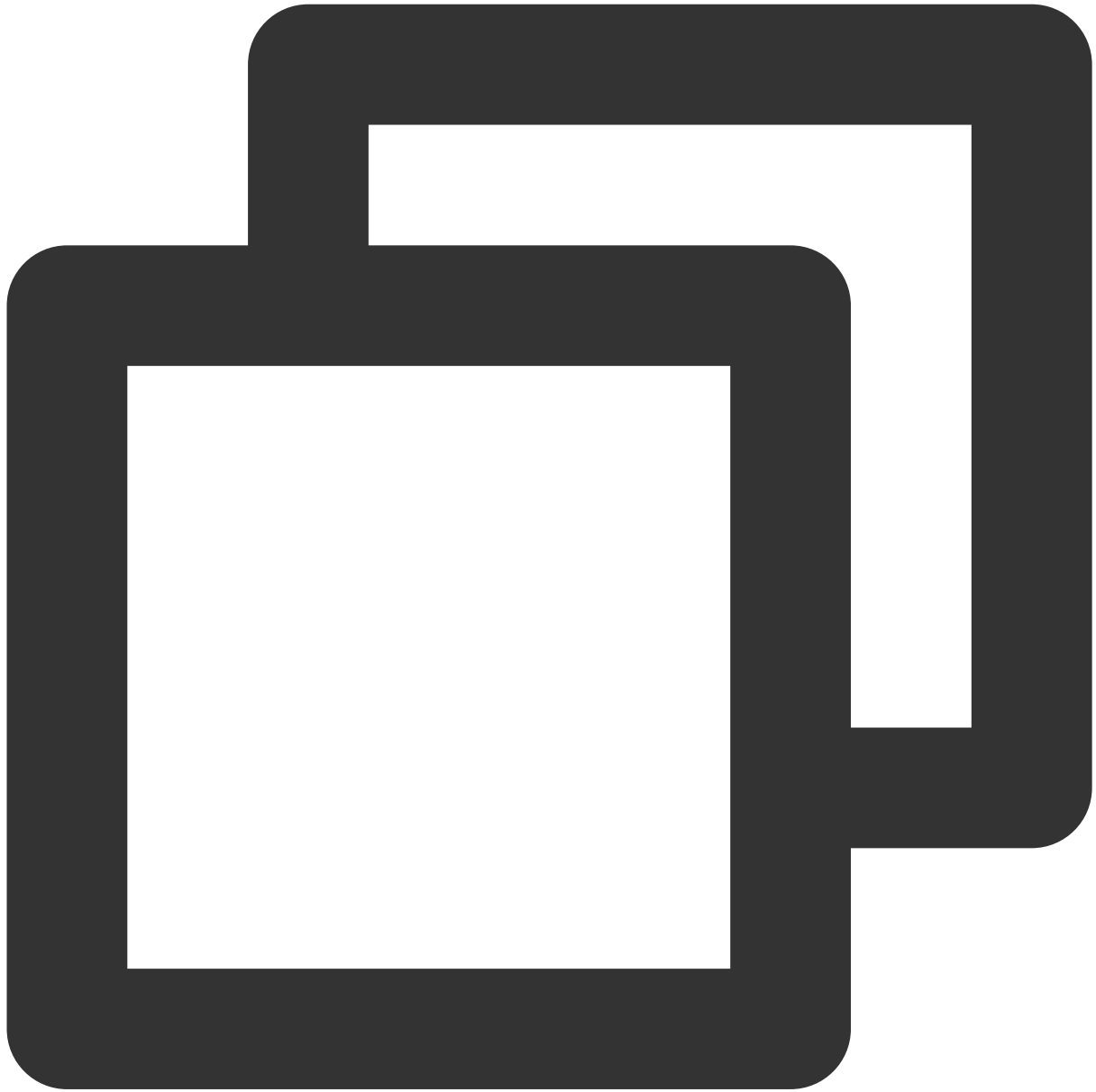
Parameter	Type	Required	Meaning
lang	String	Yes	Language type <code>en</code> or <code>zh-cn</code>

## setSelfInfo



**Note:** Vue  $\geq$  v2.2.0 is supported. Using this interface to modify user information during a call will not immediately update the UI. Changes will be visible in the next call.

Setting user nickname and profile picture.



```
try {
  await TUICallKitServer.setSelfInfo({ nickName: "xxx", avatar: "http://xxx" });
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the setSelfInfo API. Reason: ${error}`);
}
```

The parameters are described below:

Parameter	Type	Required	Meaning
nickName	String	Yes	User's nickname
avatar	String	Yes	User profile picture URL

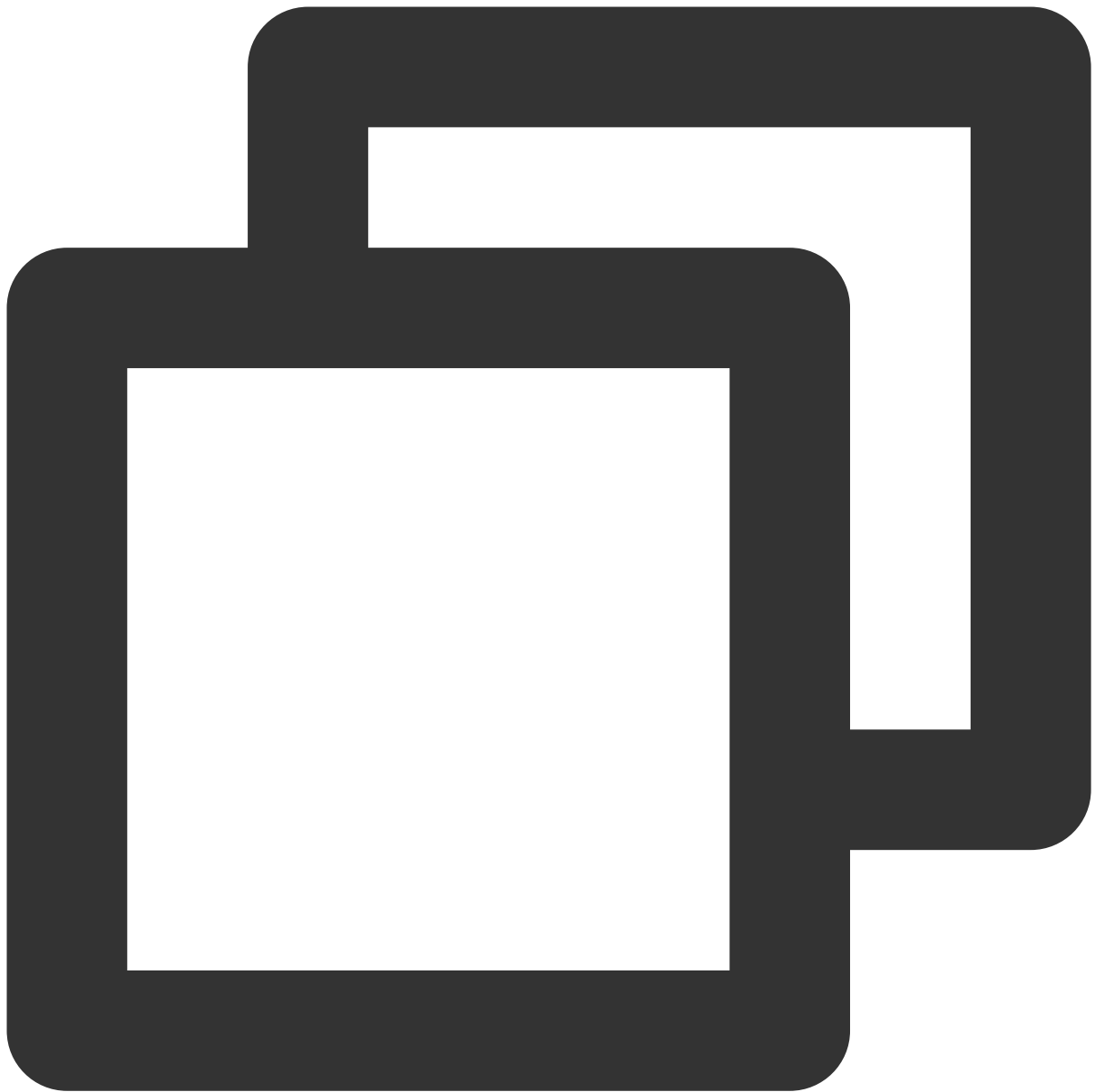
## setCallingBell

**Note:**Vue  $\geq$  v3.0.0 is supported

Customize the user's incoming call ringtone.

The input is restricted to the local MP3 format file address. It is imperative to ensure that the application has access to this file directory.

Use the ES6 import method to import the ringtone file.



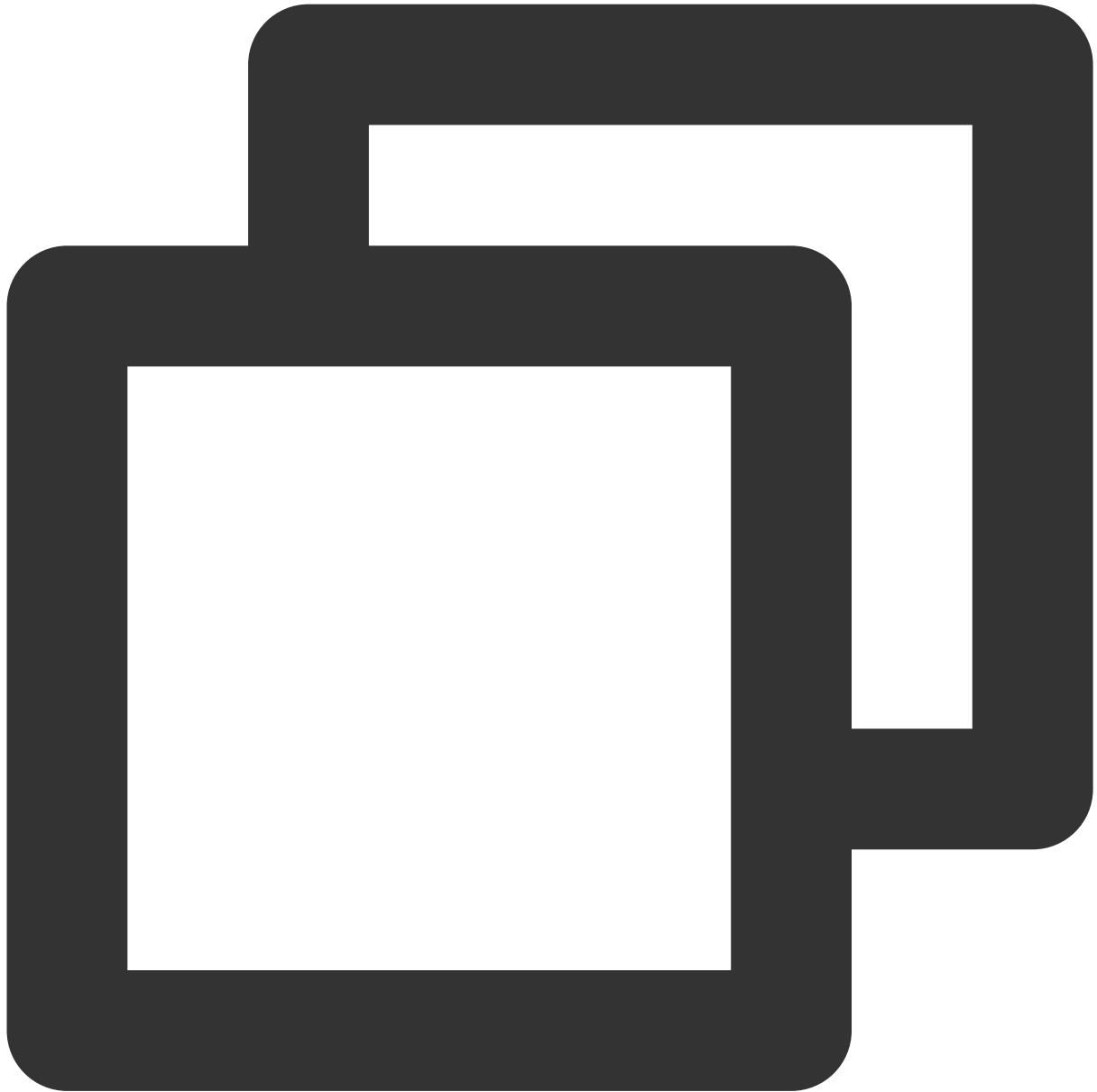
```
try {
  await TUICallKitServer.setCallingBell(filePath?: string)
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the setCallingBell API. Reason: ${error}`);
}
```

## enableFloatWindow

**Note:** Vue  $\geq$  v3.1.0 is supported

Enable/Disable floating window functionality.

By default, the floating window button in the top left corner of the call interface is hidden ( `false` ). It is displayed when set to true.



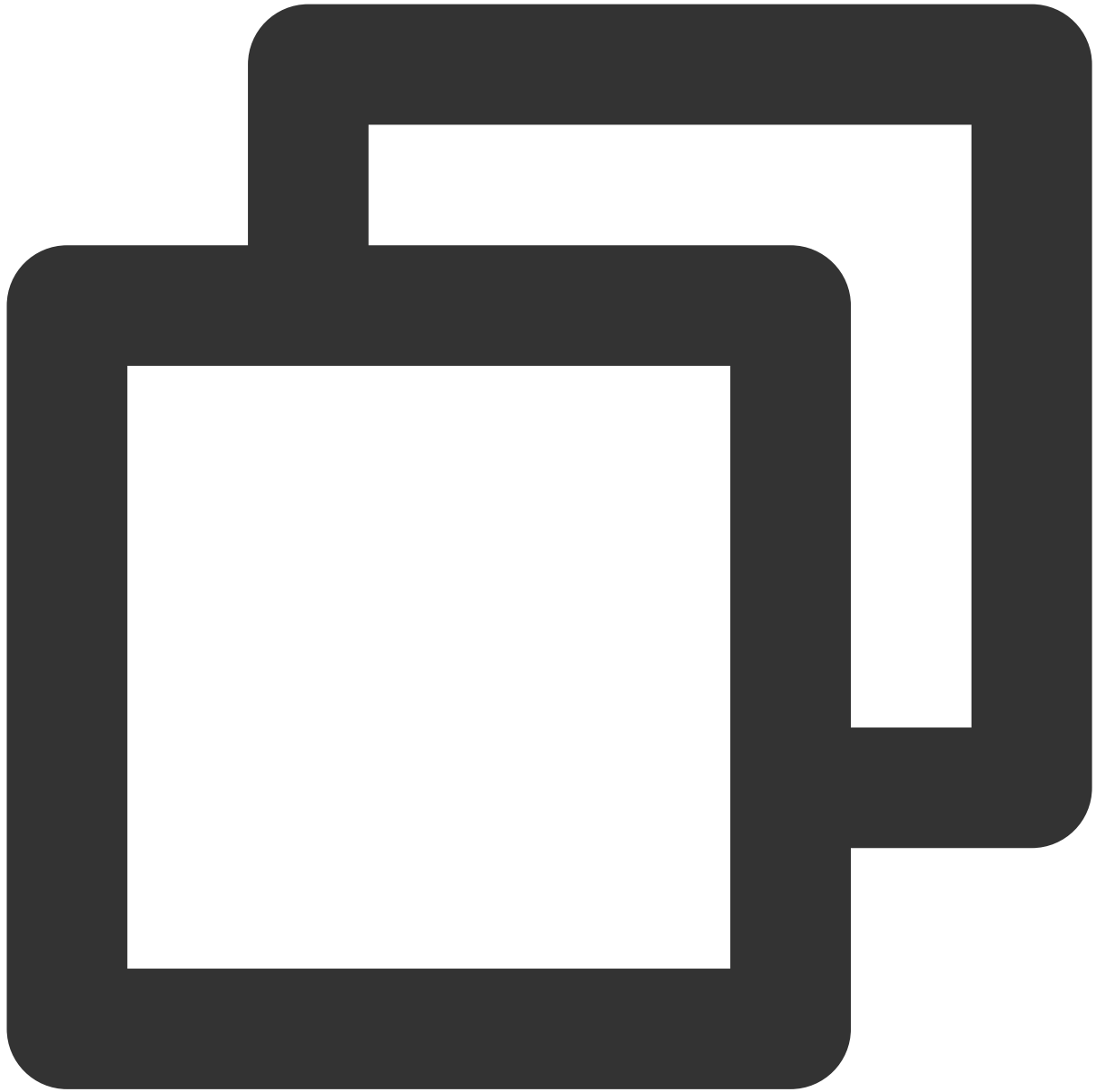
```
try {
  await TUICallKitServer.enableFloatWindow(enable: Boolean)
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the enableFloatWindow API. Reason: ${error}`);
}
```

## **enableMuteMode**

**Note:** Vue  $\geq$  v3.1.2 is supported

Enable/Disable incoming call ringtone.

After enabling, the incoming call ringtone will not be played when a call request is received.



```
try {
  await TUICallKitServer.enableMuteMode(enable: boolean)
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${error}`);
}
```

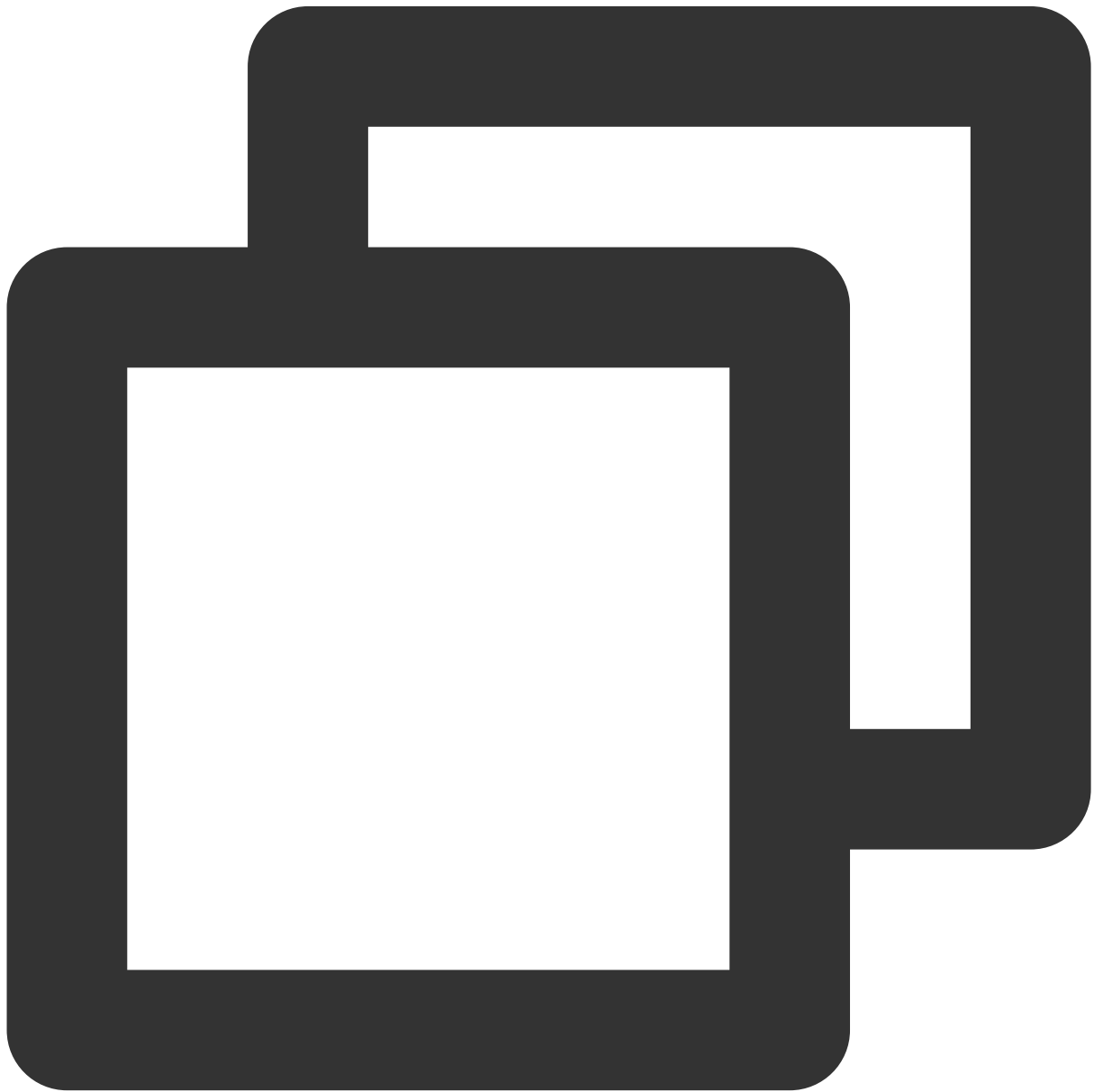
## joinInGroupCall

**Note:** Vue  $\geq$  v3.1.2 is supported

Join an existing audio-video call in a group.

**Note:**

Before joining an existing audio-video call in the group, an IM group must be pre-established or joined, and users in the group must already be engaged in a call. If the group has already been formed, please ignore this requirement. Instructions for creating a group can be found at [IM Group Management](#). Alternatively, you may directly utilize [IM TUIKit](#) for an all-in-one integration of chat, call and other scenarios.



```
try {
  const params = {
    type: TUICallType.VIDEO_CALL,
    groupID: "xxx",
    roomID: 0,
  };
  await TUICallKitServer.joinInGroupCall(params);
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${error}`);
}
```

Parameter list:

Parameter	Type	Required	Meaning
type	<a href="#">TUICallType</a>	Yes	The media type of communication, for instance, video calls, voice calls
groupID	string	Yes	Group ID associated with this group call
roomID	number	Yes	Audio-Video Room ID for this call

## destroyed

Terminate the TUICallKit instance.

This method won't automatically log out of `tim` , manual logging out is required: `tim.logout();` .



```
try {  
  await TUICallKitServer.destroyed();  
} catch (error: any) {  
  alert(`[TUICallKit] Failed to call the destroyed API. Reason: ${error}`);  
}
```

## TUICallKit Type Definition



## videoDisplayMode

There are three values for the `videoDisplayMode` display mode:

`VideoDisplayMode.CONTAIN`

`VideoDisplayMode.COVER`

`VideoDisplayMode.FILL` , the default value is `VideoDisplayMode.COVER` .

Attribute	Value	Description
videoDisplayMode	VideoDisplayMode.CONTAIN	Ensuring the full display of video content is our top priority. The dimensions of the video are scaled proportionally until one side aligns with the frame of the viewing window. In case of discrepancy in sizes between the video and the display window, the video is scaled - on the premise of maintaining the aspect ratio - to fill the window, resulting in a black border around the scaled video.
	VideoDisplayMode.COVER	Priority is given to ensure that the viewing window is filled. The video size is scaled proportionally until the entire window is filled. If the video's dimensions are different from those of the display window, the video stream will be cropped or stretched to match the window's ratio.
	VideoDisplayMode.FILL	Ensuring that the entire video content is displayed while filling the window does not guarantee preservation of the original video's proportion. The dimensions of the video will be stretched to match those of the window.

## videoResolution

The resolution `videoResolution` has three possible values:

`VideoResolution.RESOLUTION_480P`

`VideoResolution.RESOLUTION_720P`

`VideoResolution.RESOLUTION_1080P` , the default value is `VideoResolution.RESOLUTION_480P` .

### Resolution Explanation:

Video Profile	Resolution (W x H)	Frame Rate (fps)	Bitrate (Kbps)
480p	640 × 480	15	900

720p	1280 × 720	15	1500
1080p	1920 × 1080	15	2000

### Frequently Asked Questions:

iOS 13&14 does not support encoding videos higher than 720P. It is suggested to limit the highest collection to 720P on these two system versions. Refer to [iOS Safari known issue case 12](#).

Firefox does not permit the customization of video frame rates (default is set to 30fps).

Due to the influence of system performance usage, camera collection capabilities, browser restrictions, and other factors, the actual values of video resolution, frame rate, and bit rate may not necessarily match the set values exactly. In such scenarios, the browser will automatically adjust the Profile to get as close to the set values as feasible.

## STATUS

STATUS attribute value	Description
STATUS.IDLE	Idle status
STATUS.BE_INVITED	Received an Audio/Video Call Invite
STATUS.DIALING_C2C	Initiating a one-to-one call
STATUS.DIALING_GROUP	Initiating a group call
STATUS.CALLING_C2C_AUDIO	Engaged in a 1v1 Audio Call
STATUS.CALLING_C2C_VIDEO	In the midst of a one-to-one video call
STATUS.CALLING_GROUP_AUDIO	Engaged in Group Audio Communication
STATUS.CALLING_GROUP_VIDEO	Engaged in group video call

## TUICallType

TUICallType Type	Description
TUICallType.AUDIO_CALL	Audio Call
TUICallType.VIDEO_CALL	Video Call

## offlinePushInfo

Parameter	Type	Required	Meaning
offlinePushInfo.title	String	No	Offline Push Title (Optional)

offlinePushInfo.description	String	No	Offline Push Content (Optional)
offlinePushInfo.androidOPPOChannelID	String	No	Setting the channel ID for OPPO phones with 8.0 system and above for offline pushes (Optional)
offlinePushInfo.extension	String	No	Offline push through content (optional) (tsignaling version >= 0.9.0)

# TUICallEngine

Last updated : 2024-04-03 17:23:11

## TUICallEngine APIs

TUICallEngine API is the **No UI Interface** of the Audio and Video Call Components.

### API Overview

API	Description
<a href="#">createInstance</a>	Creating a TUICallEngine Instance (Singleton Pattern)
<a href="#">destroyInstance</a>	Terminating a TUICallEngine Instance (Singleton Pattern)
<a href="#">on</a>	Listening on events
<a href="#">off</a>	Canceling Event Listening
<a href="#">login</a>	Sign in Interface
<a href="#">logout</a>	Logout Interface
<a href="#">setSelfInfo</a>	Configure the user's nickname and profile photo
<a href="#">call</a>	C2C Invitation Call
<a href="#">groupCall</a>	Group Chat Invitation Call
<a href="#">accept</a>	Answer Calls
<a href="#">reject</a>	Decline Call
<a href="#">hangup</a>	End Calls
<a href="#">switchCallMediaType</a>	Switch Audio and Video Calls
<a href="#">inviteUser</a>	In Group Call, invite others to join
<a href="#">joinInGroupCall</a>	Actively join the ongoing group call
<a href="#">startRemoteView</a>	Initiate Remote Screen Rendering

<a href="#">stopRemoteView</a>	Stop Remote Screen Rendering
<a href="#">openCamera</a>	Enable camera, start local screen rendering
<a href="#">closeCamara</a>	Camera off, stop local screen rendering
<a href="#">openMicrophone</a>	Enable Microphone
<a href="#">closeMicrophone</a>	Turn off the microphone
<a href="#">setVideoQuality</a>	Set video quality
<a href="#">setVideoRenderParams</a>	Setting the display pattern of the user video display
<a href="#">getDeviceList</a>	Access device list
<a href="#">switchDevice</a>	Switch camera or microphone devices
<a href="#">enableAIVoice</a>	Enable/disable AI noise reduction
<a href="#">enableMultiDeviceAbility</a>	Enable/Disable TUICallEngine's multi-device Sign in pattern ( <b>Supported by Deluxe package</b> )

# TUICallEvent

Last updated : 2024-04-03 17:23:11

## TUICallEvent API Introduction

TUICallEvent API is the **Event Interface** of the Audio and Video Call Components.

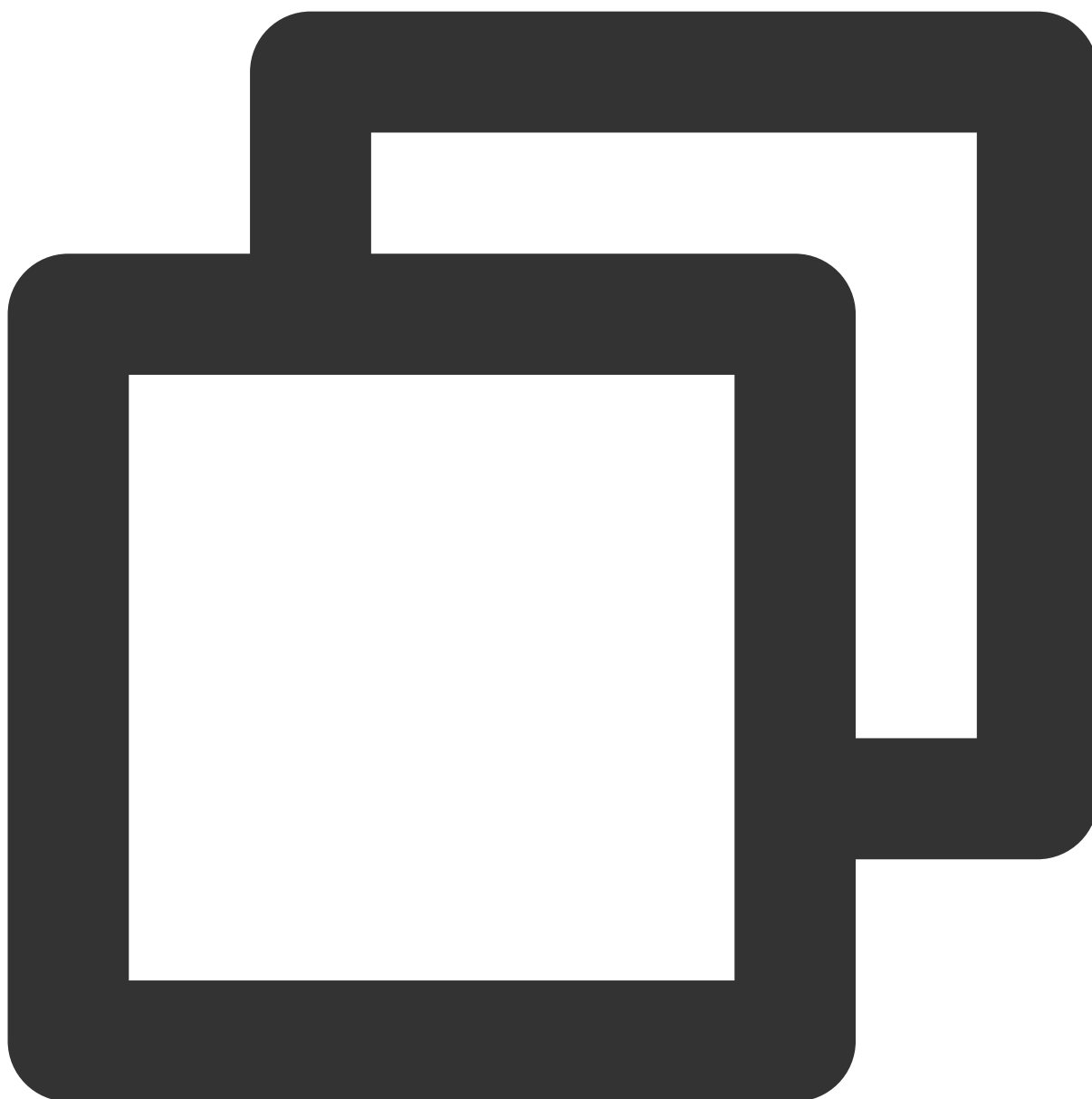
## Event List

EVENT	Description
<a href="#">TUICallEvent.ERROR</a>	An error occurred inside the SDK
<a href="#">TUICallEvent.SDK_READY</a>	This event is received when the SDK enters the ready state
<a href="#">TUICallEvent.KICKED_OUT</a>	Receiving this event after a duplicate sign-in indicates that the user has been removed from the room
<a href="#">TUICallEvent.USER_ACCEPT</a>	If a user answers, this event will be received
<a href="#">TUICallEvent.USER_ENTER</a>	If a user agrees to join the call, this event will be received
<a href="#">TUICallEvent.USER_LEAVE</a>	If a user agrees to leave the call, this event will be received
<a href="#">TUICallEvent.REJECT</a>	User declines the call, this event will be received
<a href="#">TUICallEvent.NO_RESP</a>	Invite user, no response, this event will be received
<a href="#">TUICallEvent.LINE_BUSY</a>	Invitee Busy Line, this event will be received
<a href="#">TUICallEvent.USER_VIDEO_AVAILABLE</a>	Remote user has enabled/disabled the camera, this event will be received
<a href="#">TUICallEvent.USER_AUDIO_AVAILABLE</a>	Remote user has enabled/disabled the microphone, this event will be received

<code>TUICallEvent.USER_VOICE_VOLUME</code>	Remote user adjusts speaking volume, this event will be received
<code>TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE</code>	Group chat update invitation list, this event will be received
<code>TUICallEvent.INVITED</code>	Being invited to a call, this event will be received
<code>TUICallEvent.ON_CALL_RECEIVED</code>	Event of Call Request
<code>TUICallEvent.CALLING_CANCEL</code>	If the call is not established, all platforms involved in the call will throw this event
<code>TUICallEvent.ON_CALL_BEGIN</code>	Event thrown when a call is connected
<code>TUICallEvent.CALLING_END</code>	Receiving this event indicates that the call has ended
<code>TUICallEvent.DEVICED_UPDATED</code>	Device list update, this event will be received
<code>TUICallEvent.CALL_TYPE_CHANGED</code>	Call type switching, this event will be received

## ERROR

An error occurred inside the SDK, which can be captured by listening for this event.



```
let onError = function(error) {  
    console.log(error.code, error.msg);  
};  
tuiCallEngine.on(TUICallEvent.ERROR, onError);
```

The parameters are described below:

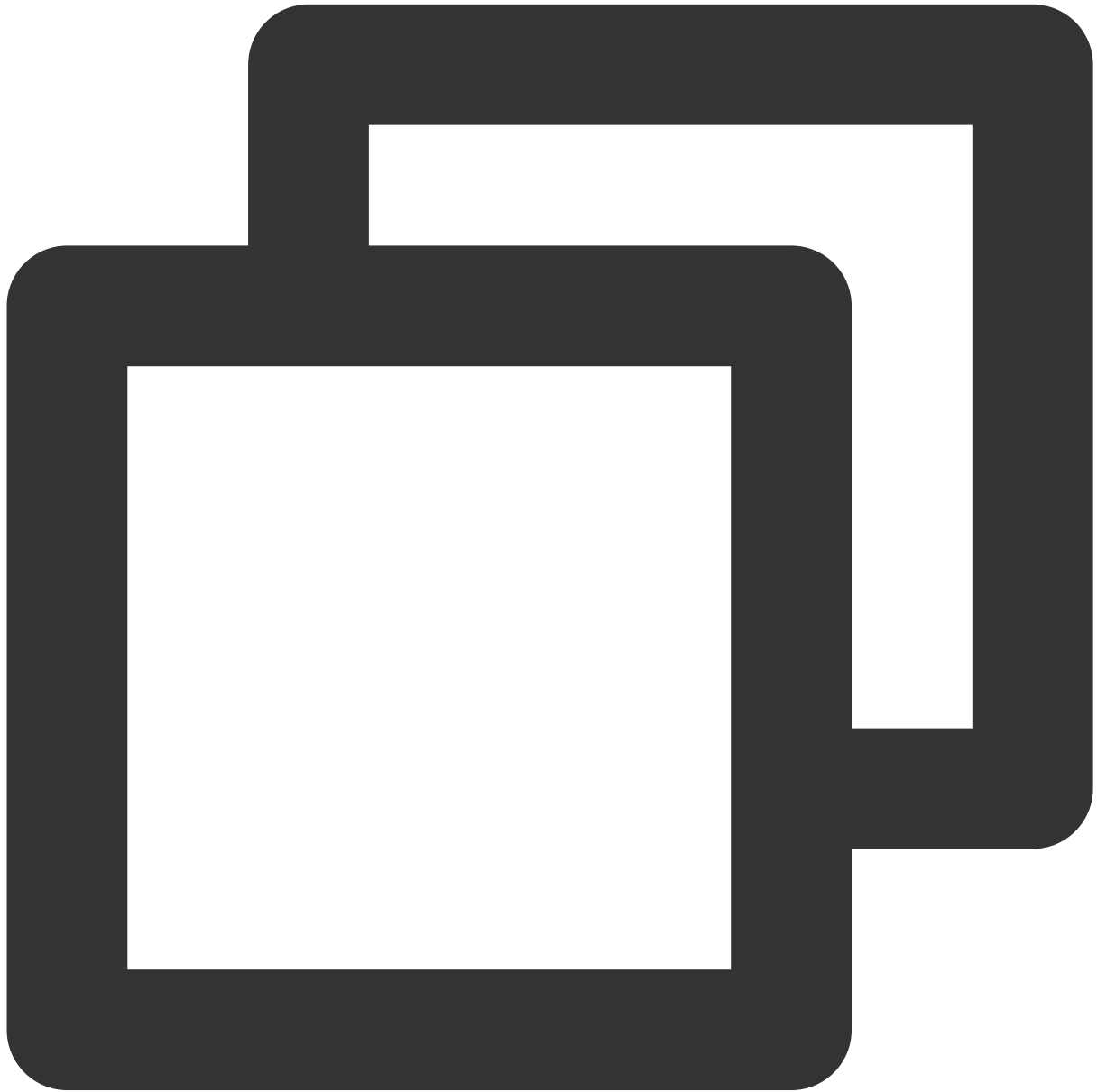
Parameter	Type	Meaning
code	int	<a href="#">Error Code</a>



msg	String	Error message
-----	--------	---------------

## SDK\_READY

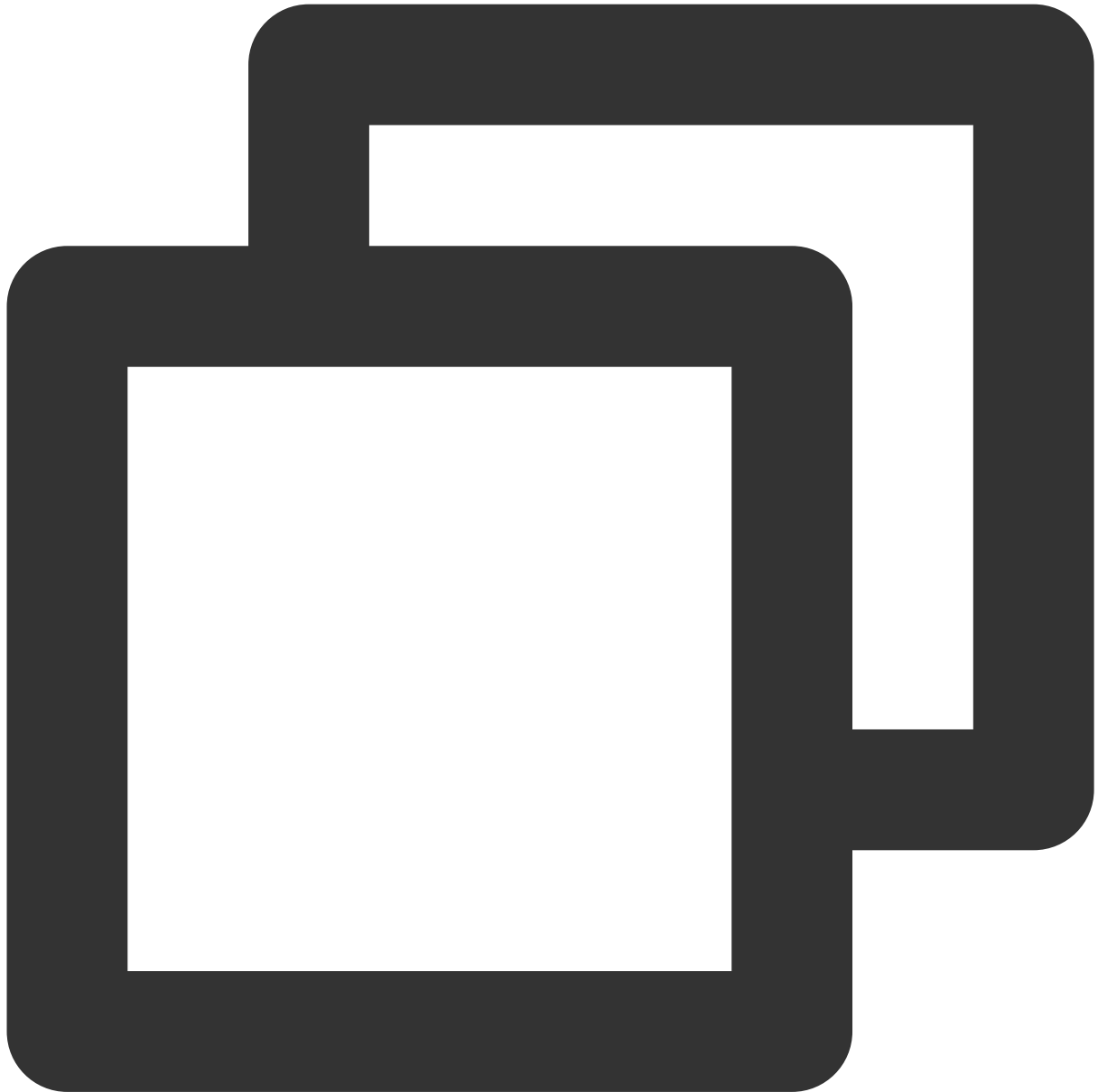
This event is received when the SDK enters the ready state.



```
let onSDKReady = function(event) {  
  console.log(event);  
};  
tuiCallEngine.on(TUICallEvent.SDK_READY, onSDKReady);
```

## KICKED\_OUT

Duplicate sign-in, receiving this event indicates that the user has been kicked out of the room.

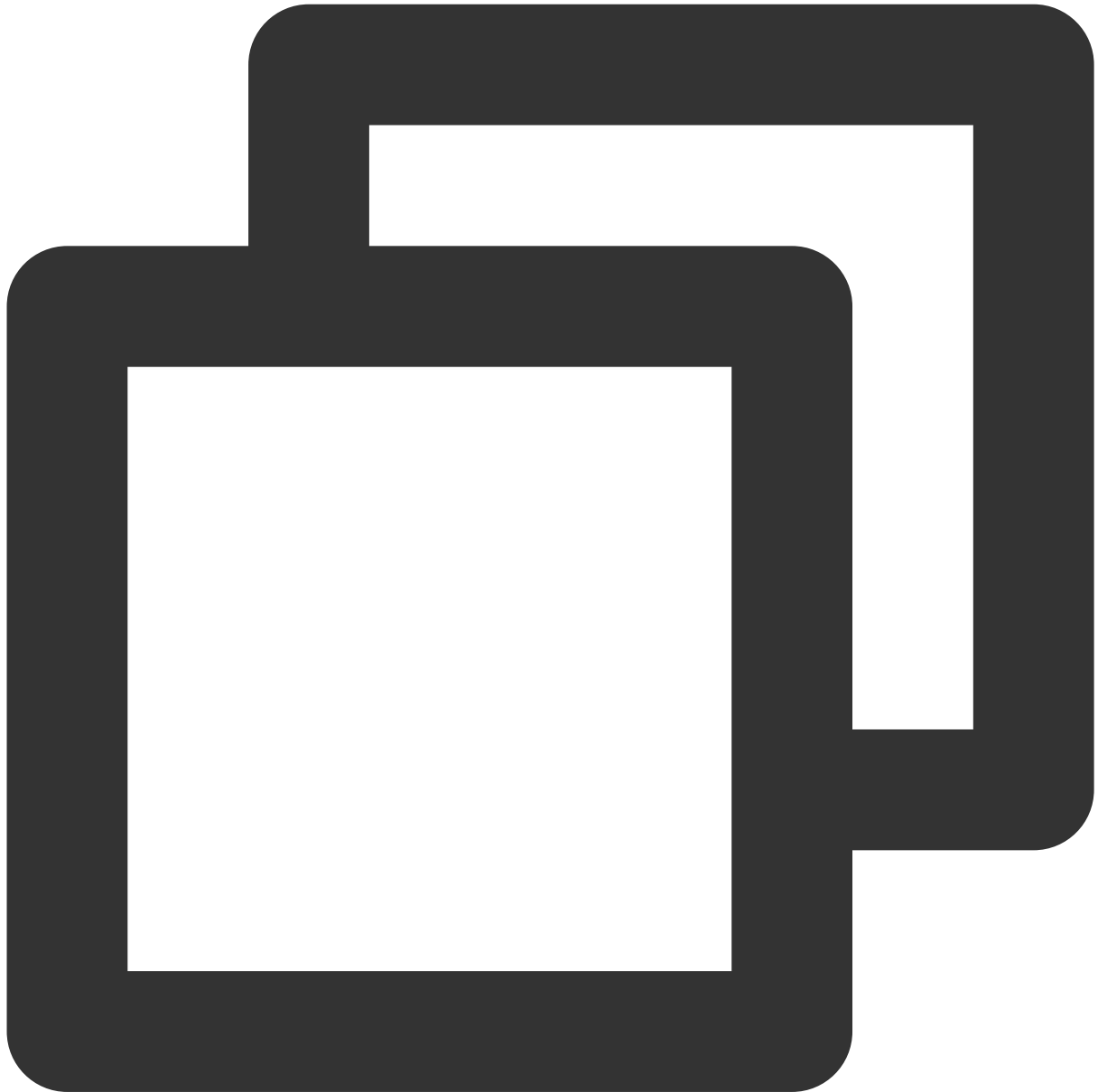


```
let handleOnKickedOut = function(event) {  
  console.log(event);  
};  
tuiCallEngine.on(TUICallEvent.KICKED_OUT, handleOnKickedOut);
```

## USER\_ACCEPT

If a user answers, all other users will receive this event, where `userID` is the user who answered.

1. In a 1v1 call: when the callee answers, the caller will throw this event.
2. In group calls: if A calls B and C, and B answers, both A and C will throw this event, with the event's `userID` being B. Similarly, if C answers, both A and B will throw this event, with the event's `userID` being C.



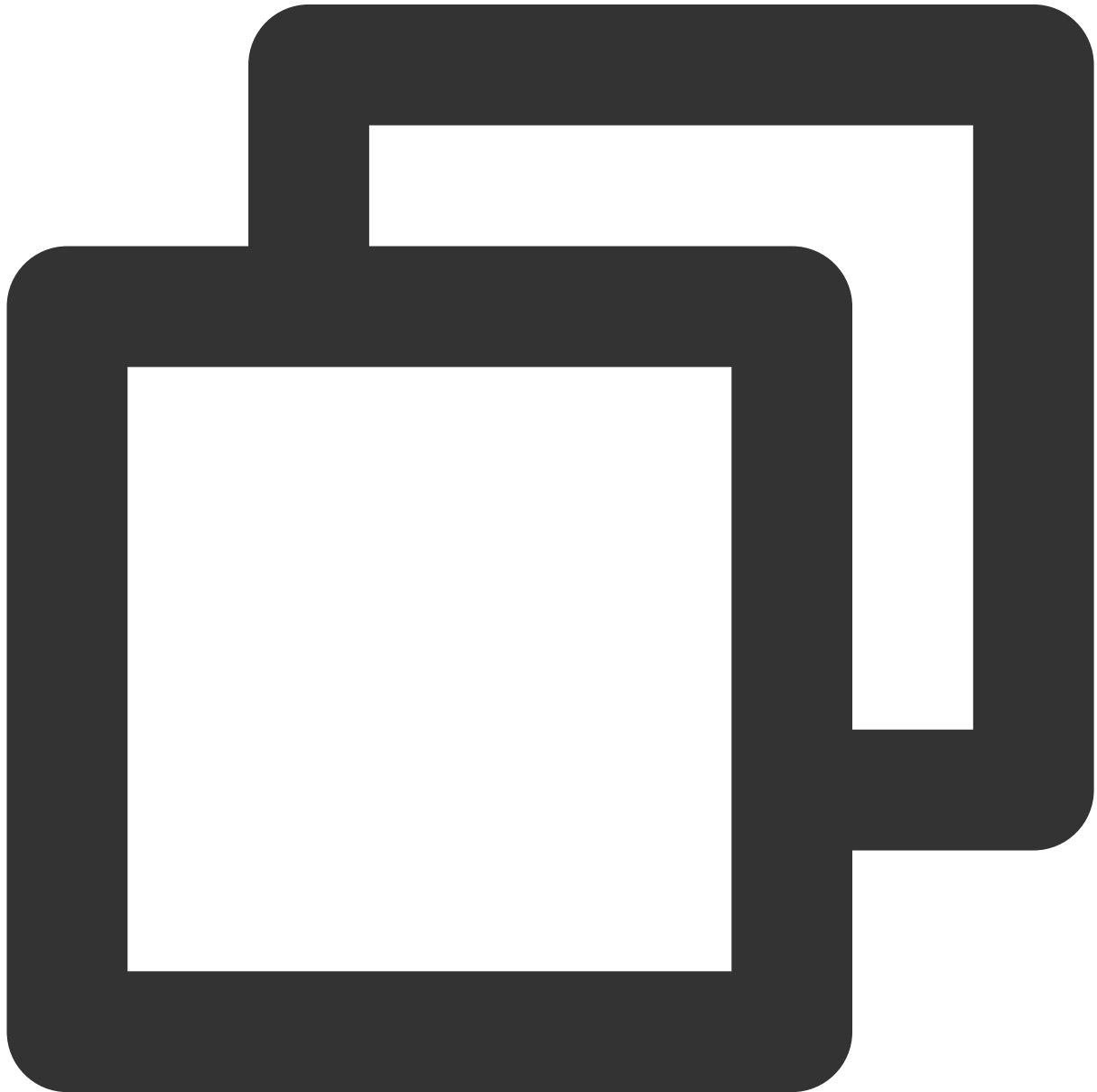
```
let handleUserAccept = function(event) {  
  console.log(event.userID);  
};  
tuiCallEngine.on(TUICallEvent.USER_ACCEPT, handleUserAccept);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Answering User ID

## USER\_ENTER

If a user agrees to join the call, all other users will receive this event, where `userID` is the user who agreed to join.



```
let handleUserEnter = function(event) {  
  console.log(event.userID);  
};
```

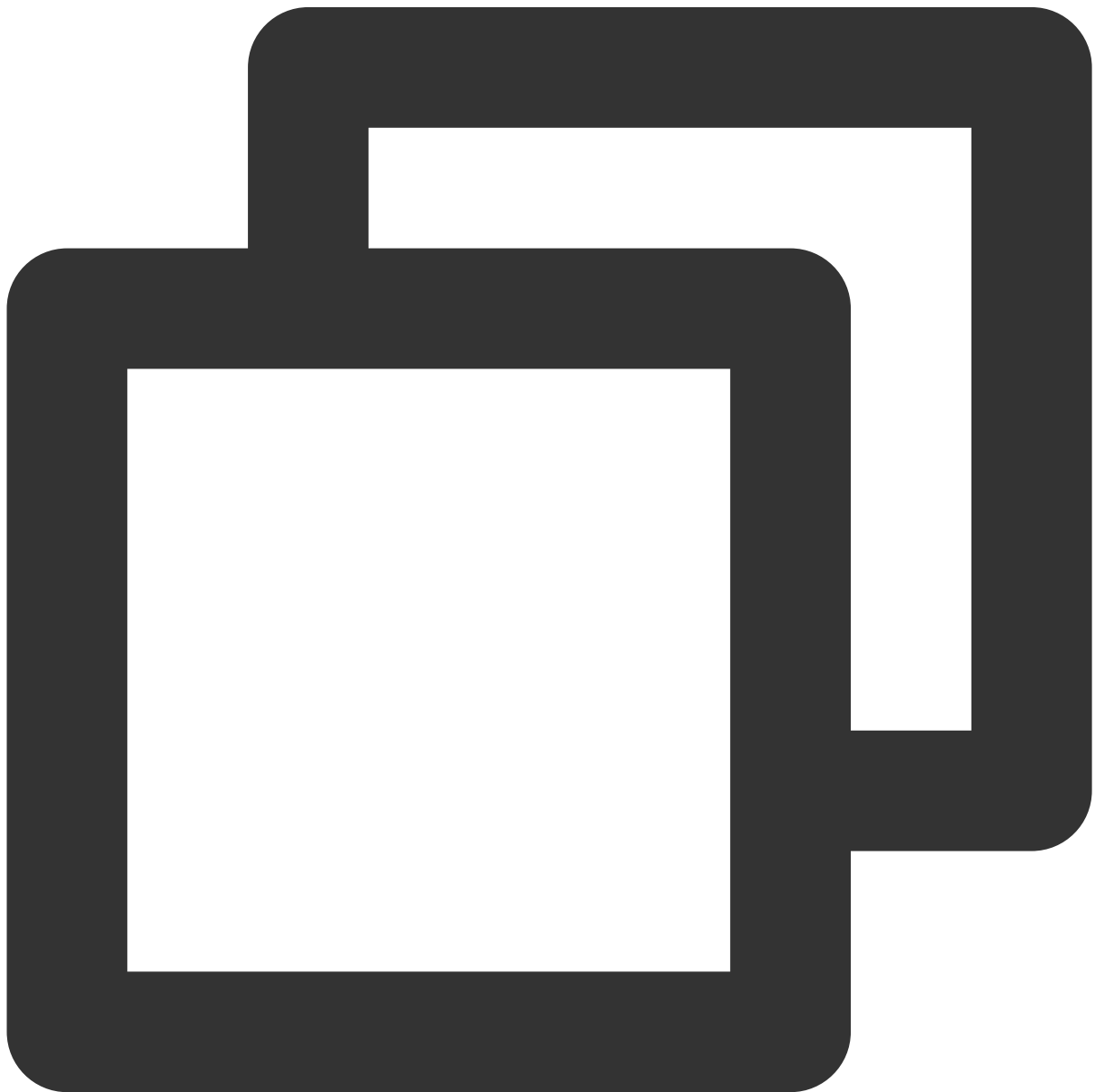
```
tuiCallEngine.on(TUICallEvent.USER_ENTER, handleUserEnter);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Entering User ID

## USER\_LEAVE

When a user leaves the call,**in the call** other users will receive this event, userID is the user who left the call.



```
let handleUserLeave = function(event) {  
    console.log(event.userID);  
};  
tuiCallEngine.on(TUICallEvent.USER_LEAVE, handleUserLeave);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Exiting User ID

## REJECT

### Call Rejection Event

1. In a 1v1 call, only the caller will receive the rejection event, where `userID` is the called party.
2. In a group call, all invitees can receive this event, where `userID` is the user who declined the call.



```
let handleInviteeReject = function(event) {  
    console.log(event.userID);  
};  
tuiCallEngine.on(TUICallEvent.REJECT, handleInviteeReject);
```

The parameters are described below:

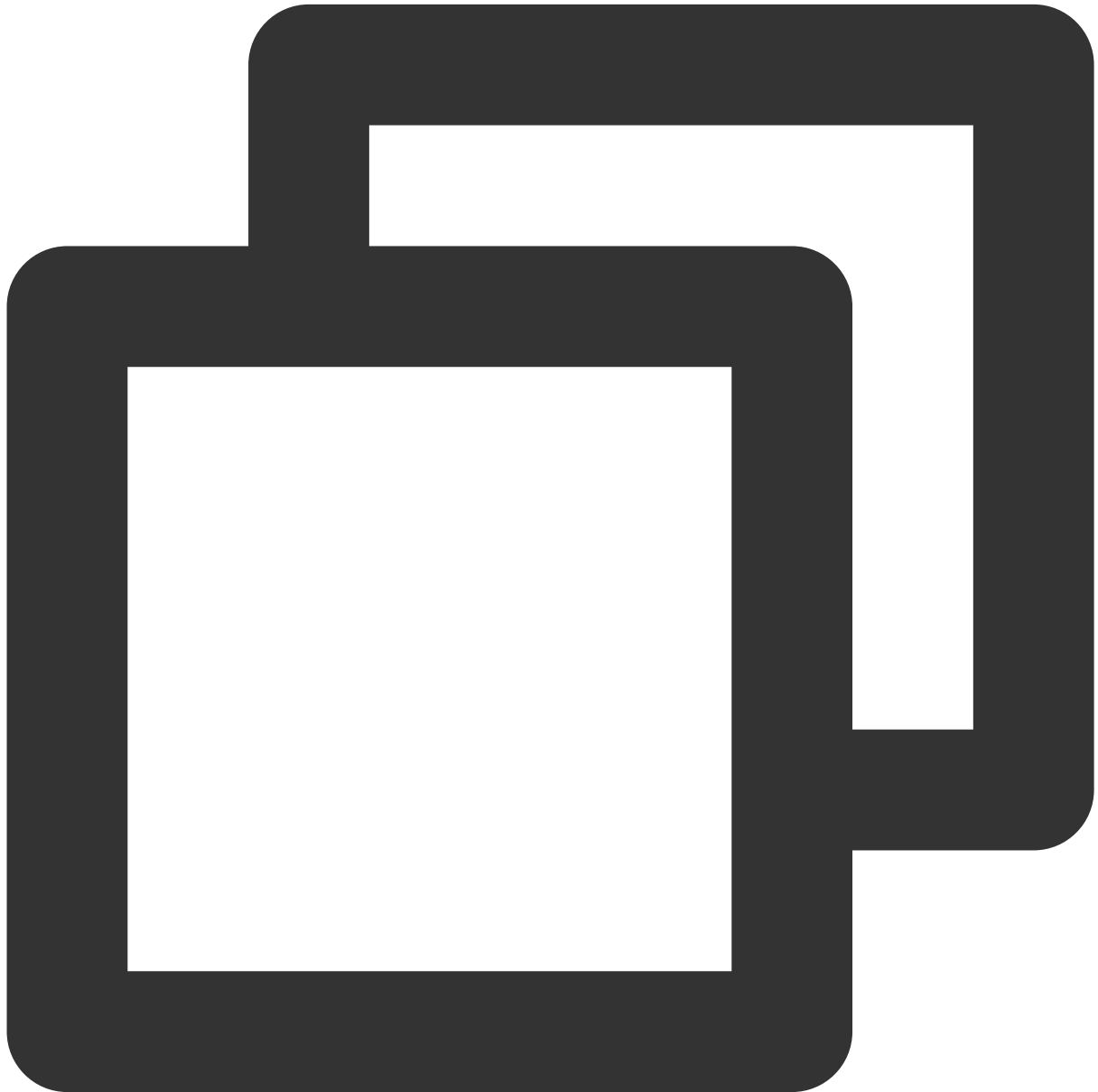
Parameter	Type	Meaning
userID	String	Declining User ID

## NO\_RESP

No Answer Event by Inviting User.

In a 1v1 call, only the initiator will receive the event of no answer. For example, A invites B, B does not answer, A can receive this event.

In a group call, all invitees can receive this event. For example, if A invites B and C to join the call and B does not respond, both A and C will receive this event.



```
let handleNoResponse = function(event) {  
  console.log(event.sponsor, event.userIDList);  
};
```



```
tuiCallEngine.on(TUICallEvent.NO_RESP, handleNoResponse);
```

The parameters are described below:

Parameter	Type	Meaning
sponsor	String	Caller's User ID
userIDList	Array<String>	List of Users Who Triggered Timeout Due to No Response

## LINE\_BUSY

Busy Call Event.



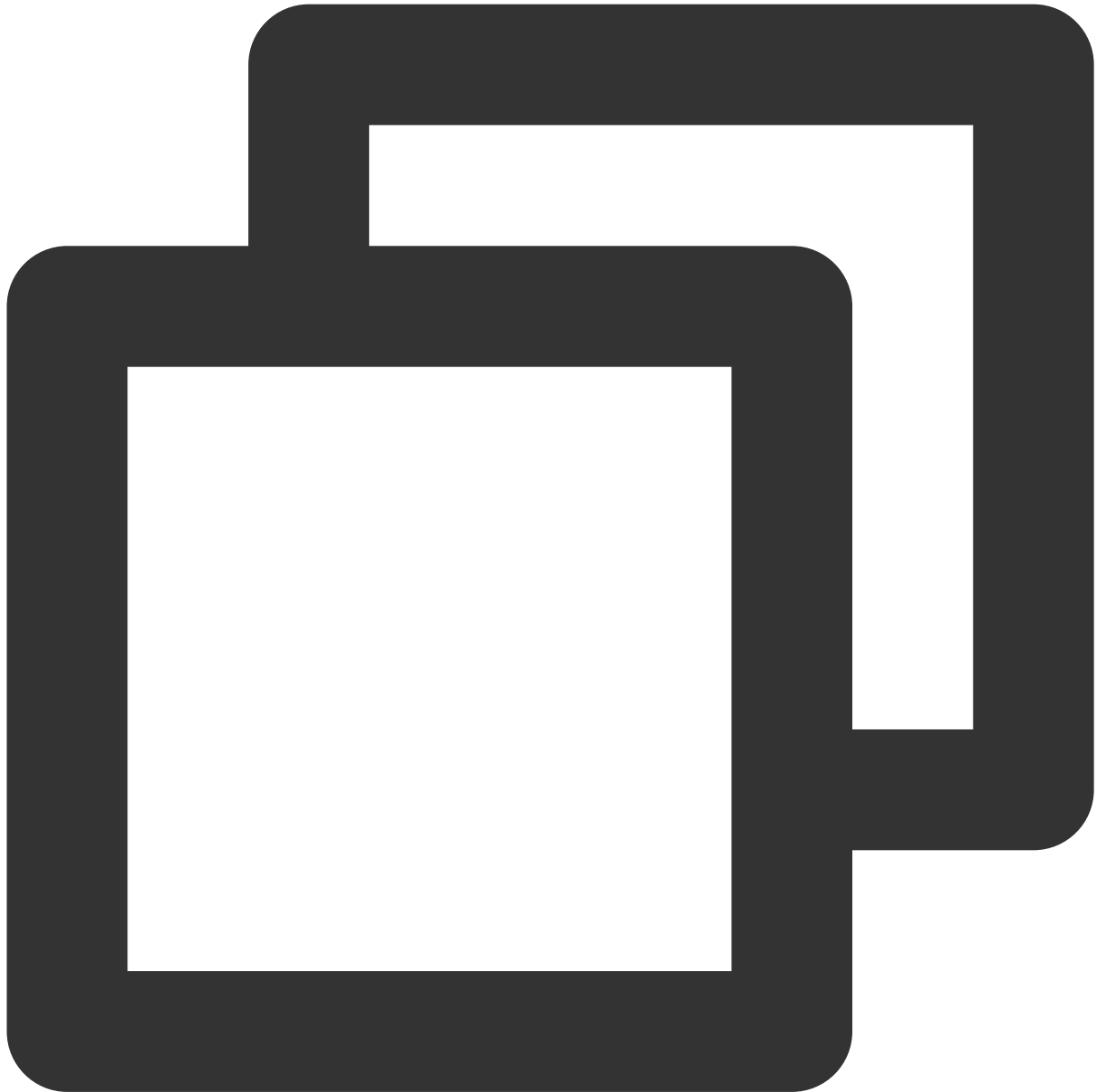
```
let handleLineBusy = function(event) {  
  console.log(event)  
};  
tuiCallEngine.on(TUICallEvent.LINE_BUSY, handleLineBusy);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Busy User ID

## USER\_VIDEO\_AVAILABLE

Remote user has enabled/disabled the camera, this event will be received.



```
let handleUserVideoChange = function(event) {  
    console.log(event.userID, event.isVideoAvailable);  
};  
tuiCallEngine.on(TUICallEvent.USER_VIDEO_AVAILABLE, handleUserVideoChange);
```

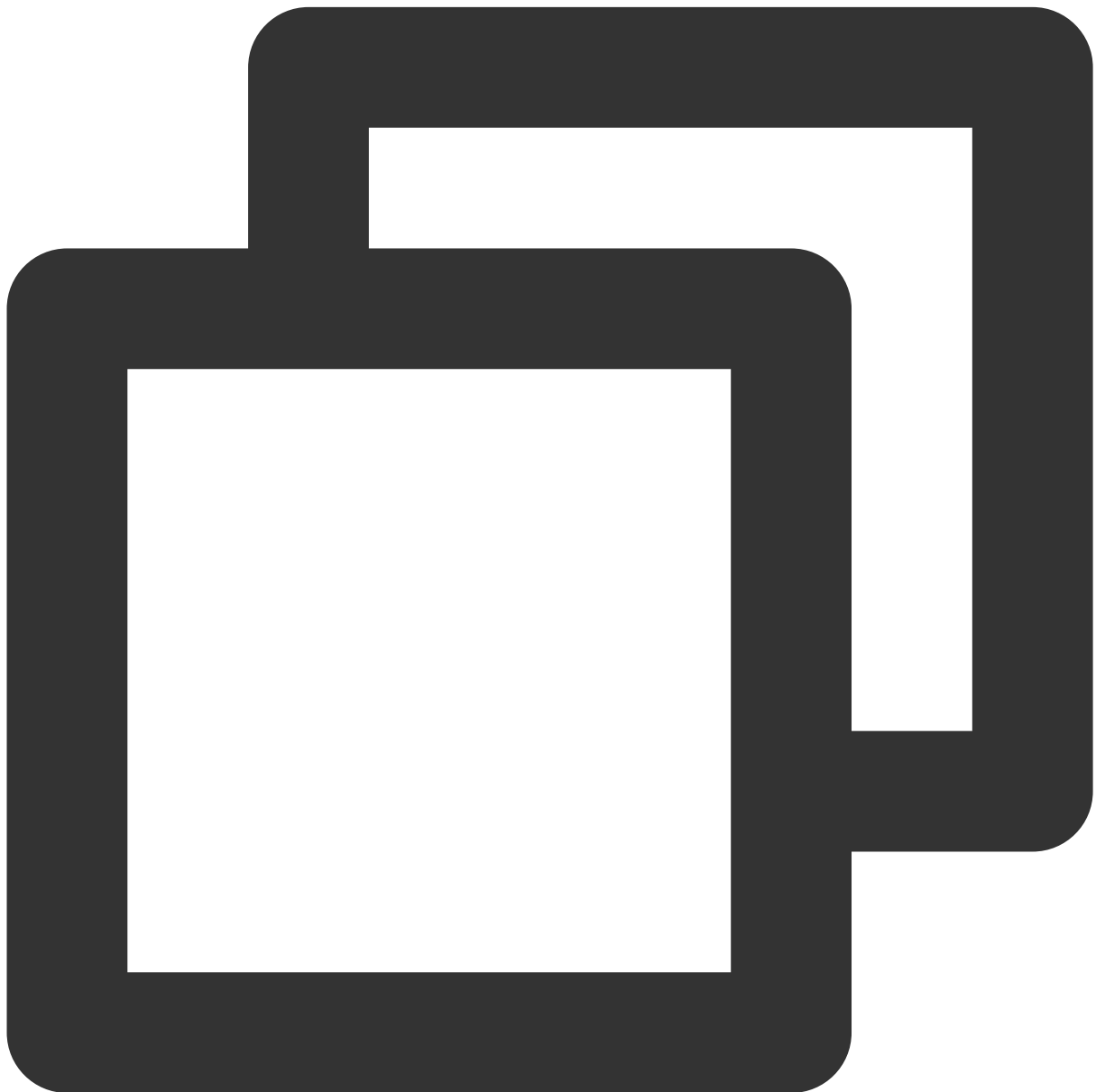
The parameters are described below:

Parameter	Type	Meaning

userID	String	Remote User ID
isVideoAvailable	Boolean	true: Remote User turns Camera On; false: Remote User turns Camera Off

## USER\_AUDIO\_AVAILABLE

Remote user has enabled/disabled the microphone, this event will be received.



```
let handleUserAudioChange = function(event) {  
  console.log(event.userID, event.isAudioAvailable);  
}
```

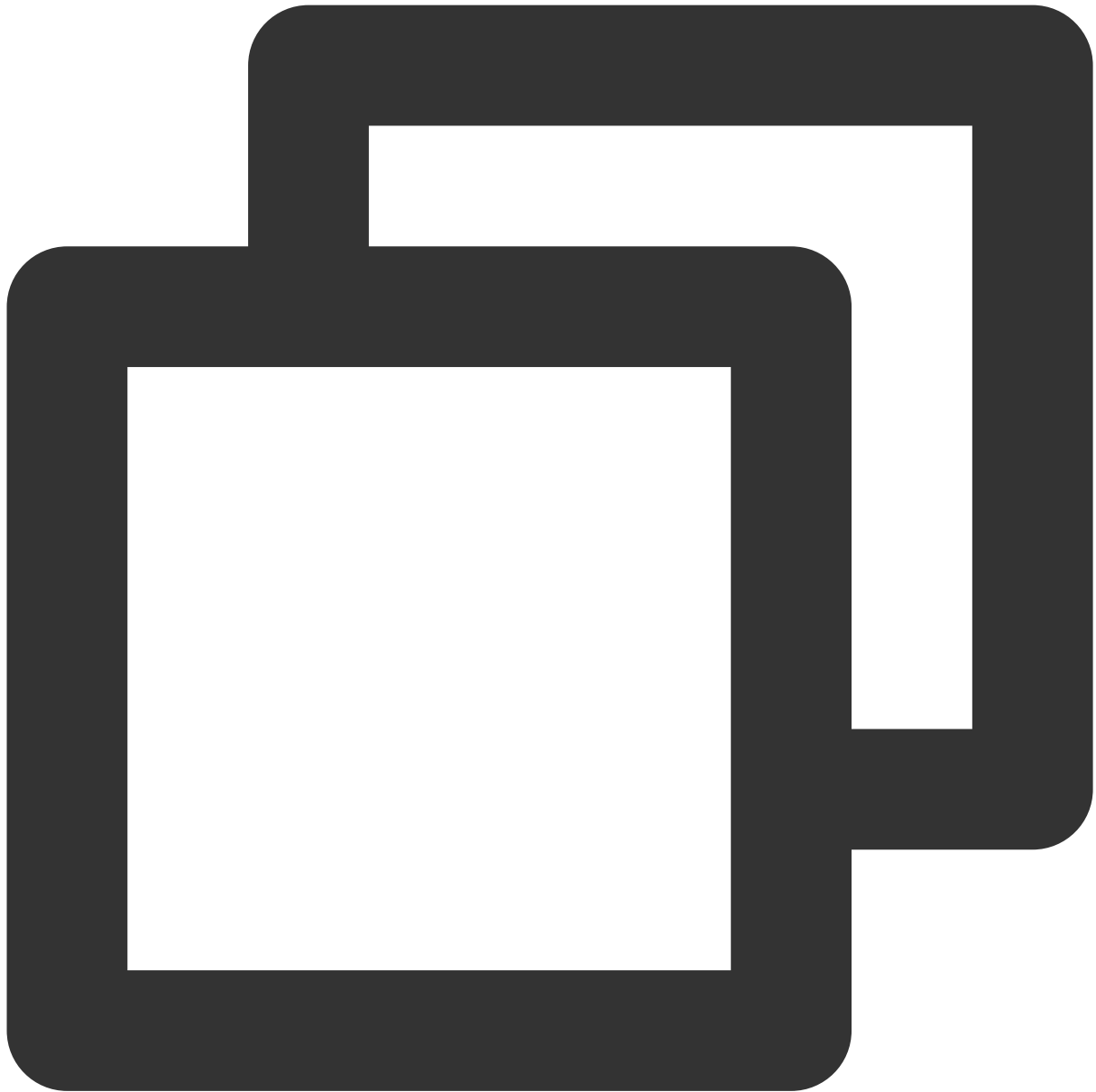
```
};  
tuiCallEngine.on(TUICallEvent.USER_AUDIO_AVAILABLE, handleUserAudioChange);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Remote User ID
isAudioAvailable	Boolean	true Remote user turns on microphone; false Remote user turns off microphone

## USER\_VOICE\_VOLUME

Remote user adjusts speaking volume, this event will be received.



```
let handleUserVoiceVolumeChange = function(event) {  
  console.log(event.volumeMap);  
};  
tuiCallEngine.on(TUICallEvent.USER_VOICE_VOLUME, handleUserVoiceVolumeChange);
```

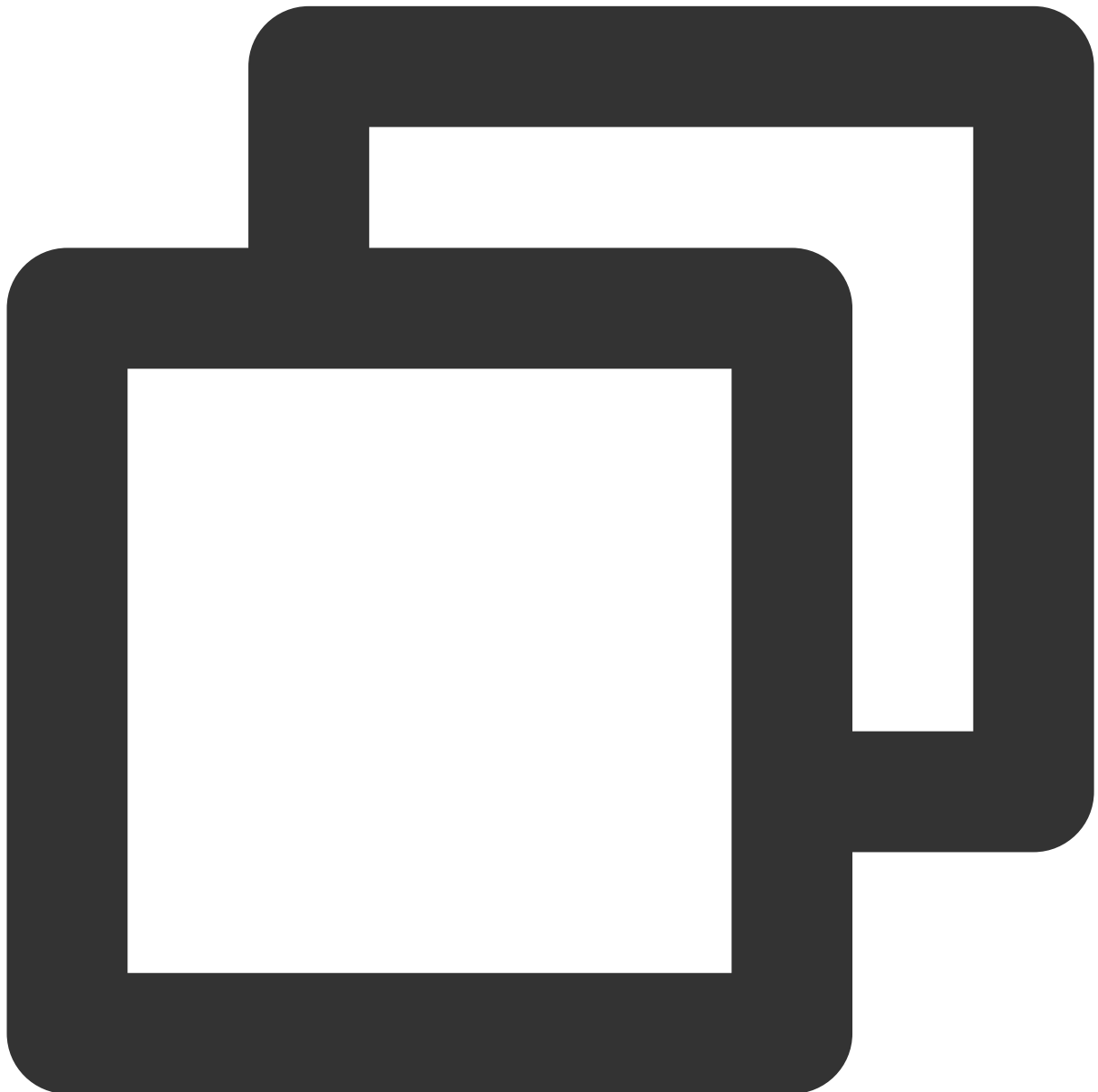
The parameters are described below:

Parameter	Type	Meaning
volumeMap	Array<Object>	Volume meter, access the volume level for each userid, with the minimum

	volume at 0 and the maximum volume at 100
--	---

## GROUP\_CALL\_INVITEE\_LIST\_UPDATE

Group chat update invitation list, this event will be received.



```
let handleGroupInviteeListUpdate = function(event) {  
  console.log(event.userIDList);  
};  
tuiCallEngine.on(TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE, handleGroupInviteeListUpdate);
```

The parameters are described below:

Parameter	Type	Meaning
userIDList	Array<String>	Group update invitation list

## INVITED

Receiving a new incoming call event, the called party will be notified. By listening to this event, you can decide whether to display the call answering interface.

### Please Note:

**Plan to deprecate in subsequent versions**





```
let handleNewInvitationReceived = function(event) {  
    console.log(event.sponsor, event.userIDList, event.isFromGroup, event.inviteData,  
    };  
tuiCallEngine.on(TUICallEvent.INVITED, handleNewInvitationReceived);
```

The parameters are described below:

Parameter	Type	Meaning
sponsor	String	Inviter

userIDList	Array<String>	Also Invited Persons
isFromGroup	Boolean	Is it a Group Call
inviteData	Object	Call Data
inviteID	String	Invitation ID, identifying one invitation
userData	String	Extended field: Utilized for amplifying details in the invitation signaling
callId	String	The unique ID for this call. <b>Supported in v1.4.6+</b>
roomId	Number	The Audio/Video Room ID for this call. <b>Supported in v1.4.6+</b>
callMediaType	Number	Media Type of the call, Video Call, Voice Call. <b>Supported from version v1.4.6+</b>
callRole	String	role, Enumeration Type: Caller, Called. <b>Supported from version v1.4.6+</b>

## ON\_CALL\_RECEIVED

Receiving a new incoming call event, the called party will be notified. By listening to this event, you can decide whether to display the call answering interface.

**Please Note:**

**Supported from version v1.4.6+**



```
let handleOnCallReceived = function(event) {  
    console.log(event)  
};  
tuiCallEngine.on(TUICallEvent.ON_CALL_RECEIVED, handleOnCallReceived);
```

The parameters are described below:

Parameter	Type	Meaning
sponsor	String	Inviter

userIDList	Array<String>	Also Invited Persons
isFromGroup	Boolean	Is it a Group Call
inviteData	Object	Call Data
inviteID	String	Invitation ID, identifying one invitation
userData	String	Extended field: Utilized for amplifying details in the invitation signaling
callId	String	Unique ID for this call
roomId	Number	Audio-Video Room ID for this call
callMediaType	Number	Media Type of the call, Video Call, Voice Call
callRole	String	role, Enumeration Type: Caller, Called

## CALLING\_CANCEL

**If the call is not established, this event will be thrown.** By listening to this event, you can implement display logic similar to missed calls, reset UI state, etc. Scenarios where the call is not established include:

Caller Cancelled: The caller throws this event, userID is the caller; the called also throws this event, userID is the called;

Callee Timeout: The caller will throw both [NO\\_RESP](#) and [CALLING\\_CANCEL](#) events, userID is the caller; the called throws the [CALLING\\_CANCEL](#) event, userID is the called;

Callee Rejected: The caller will throw both [REJECT](#) and [CALLING\\_CANCEL](#) events, userID is the caller; the called throws the [CALLING\\_CANCEL](#) event, userID is the called;

Callee Busy: The caller will throw both [LINE\\_BUSY](#) and [CALLING\\_CANCEL](#) events, userID is the caller; the callee throws the [CALLING\\_CANCEL](#) event, userID is the callee;

### Please Note:

**Plan to deprecate in subsequent versions**



```
let handleCallingCancel = function(event) {  
    console.log(event.userID);  
};  
tuiCallEngine.on(TUICallEvent.CALLING_CANCEL, handleCallingCancel);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Cancelled User ID

callId	String	The unique ID for this call. <b>Supported in v1.4.6+</b>
roomId	Number	The Audio/Video Room ID for this call. <b>Supported in v1.4.6+</b>
callMediaType	Number	Media Type of the call, Video Call, Voice Call. <b>Supported from version v1.4.6+</b>
callRole	String	role, Enumeration Type: Caller, Called. <b>Supported from version v1.4.6+</b>

## ON\_CALL\_CANCELED

**If the call is not established, this event will be thrown.** By listening to this event, you can implement display logic similar to missed calls, reset UI state, etc. Scenarios where the call is not established include:

Caller Cancelled: The caller throws this event, userID is the caller; the called also throws this event, userID is the called;

Callee Timeout: The caller will throw both [NO\\_RESP](#) and [CALLING\\_CANCEL](#) events, userID is the caller; the called throws the [CALLING\\_CANCEL](#) event, userID is the called;

Callee Rejected: The caller will throw both [REJECT](#) and [CALLING\\_CANCEL](#) events, userID is the caller; the called throws the [CALLING\\_CANCEL](#) event, userID is the called;

Callee Busy: The caller will throw both [LINE\\_BUSY](#) and [CALLING\\_CANCEL](#) events, userID is the caller; the callee throws the [CALLING\\_CANCEL](#) event, userID is the callee;

**Please Note:**

**Supported from version v1.4.6+**



```
let handleOnCallCanceled = function(event) {  
    console.log(event.userID);  
};  
tuiCallEngine.on(TUICallEvent.ON_CALL_CANCELED, handleOnCallCanceled);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Cancelled User ID

callId	String	Unique ID for this call
roomId	Number	Audio-Video Room ID for this call
callMediaType	Number	Media Type of the call, Video Call, Voice Call
callRole	String	role, Enumeration Type: Caller, Called

## ON\_CALL\_BEGIN

Indicates call connection. Both caller and called can receive it. You can start cloud recording, content review, etc., by listening to this event.

### Please Note:

**Supported from version v1.4.6+**





```
let handleOnCallBegin = function(event) {  
    console.log(event)  
};  
tuiCallEngine.on(TUICallEvent.ON_CALL_BEGIN, handleOnCallBegin);
```

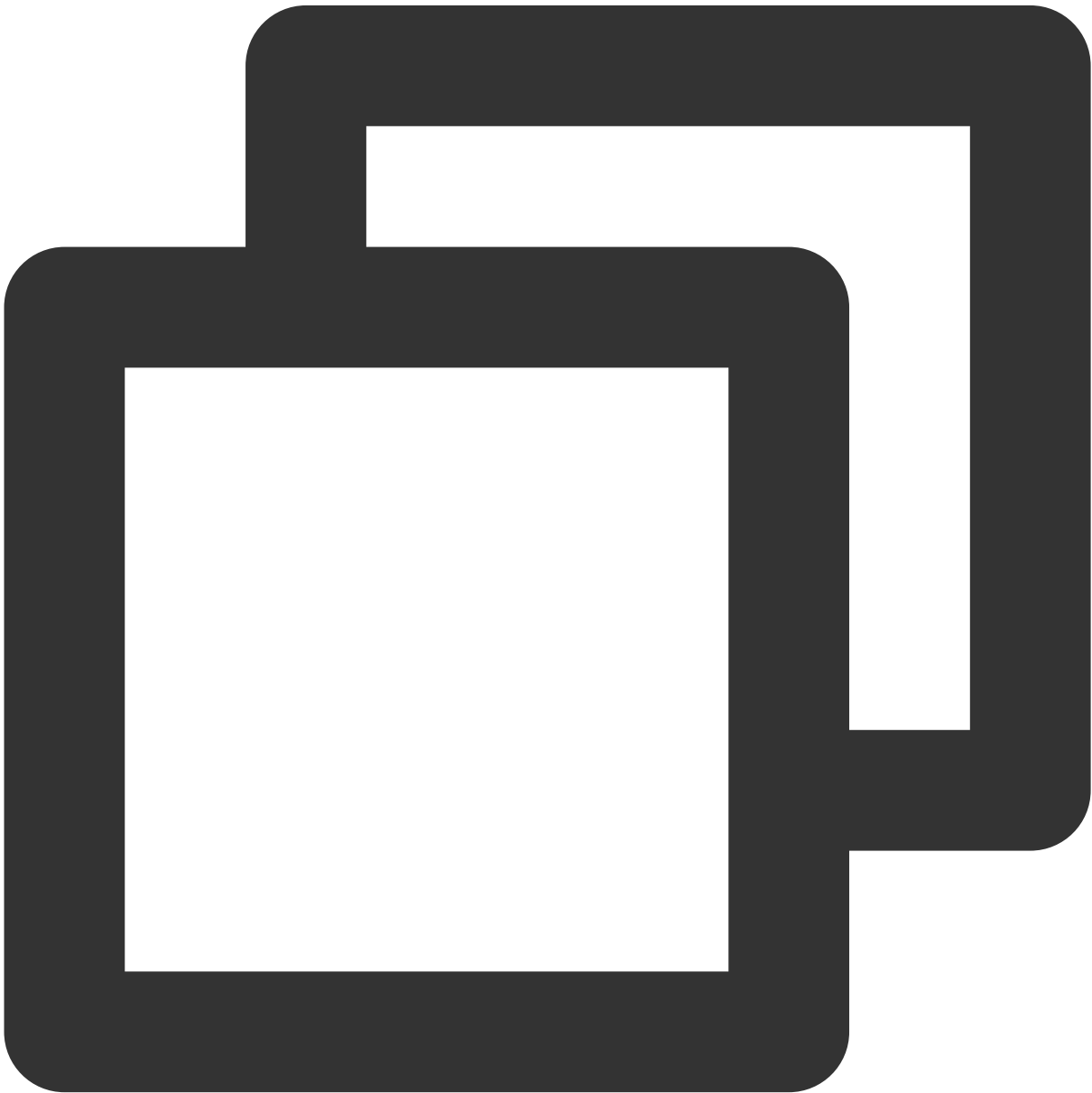
The parameters are described below:

Parameter	Type	Meaning
callId	String	Unique ID for this call

roomID	Number	Audio-Video Room ID for this call
callMediaType	Number	Media Type of the call, Video Call, Voice Call
callRole	String	role, Type: Caller, Called

CALLING\_END

Indicates call termination. Both caller and called can trigger this event. You can display information such as call duration, call type, or stop the cloud recording process by listening to this event.



```
let handleCallingEnd = function(event) {  
    console.log(event.userID, event.);  
};  
tuiCallEngine.on(TUICallEvent.CALLING_END, handleCallingEnd);
```

The parameters are described below:

Parameter	Type	Meaning
roomID	Number	Audio-Video Room ID for this call, currently only supports numeric room number, future versions will support character string room numbers
callMediaType	Number	Media Type of the call, Video Call, Voice Call
callRole	String	role, Enumeration Type: Caller ('inviter'), Called ('invitee'), Unknown ('')
totalTime	Number	The duration of this call in seconds
userID	String	userID of the call termination.
callId	String	The unique ID for this call. <b>Supported in v1.4.6+</b>
callEnd	Number	The duration of this call ( <b>will be deprecated</b> ) in seconds

## DEVICED\_UPDATED

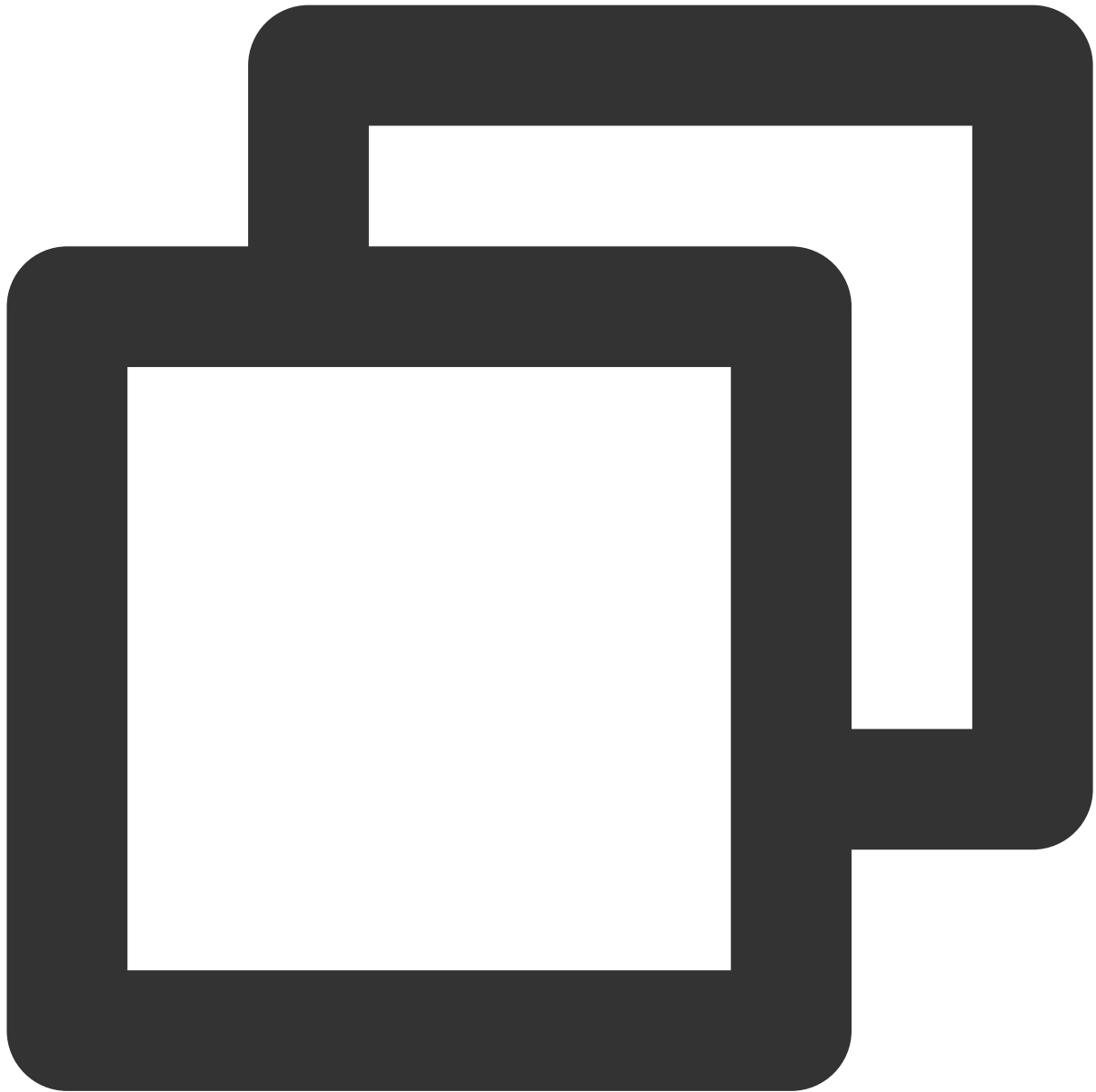
Device list update, this event will be received.



```
let handleDeviceUpdated = function({ microphoneList, cameraList, currentMicrophoneID, currentCameraID }) {  
    console.log(microphoneList, cameraList, currentMicrophoneID, currentCameraID);  
};  
tuiCallEngine.on(TUICallEvent.DEVICED_UPDATED, handleDeviceUpdated);
```

## CALL\_TYPE\_CHANGED

Call type switching, this event will be received.



```
let handleCallTypeChanged = function({ oldCallType, newCallType }) {  
  console.log(oldCallType, newCallType)  
};  
tuiCallEngine.on(TUICallEvent.CALL_TYPE_CHANGED, handleDeviceUpdated);
```

The parameters are described below:

Parameter	Type	Meaning
oldCallType	Number	Old call type

newCallType	Number	New call type
-------------	--------	---------------

# Flutter

## API Overview

Last updated : 2024-03-12 14:51:07

### TUICallKit (UI Included)

TUICallKit is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat.

API	Description
<a href="#">login</a>	Log in
<a href="#">logout</a>	Sign out
<a href="#">setSelfInfo</a>	Sets the user nickname and profile photo.
<a href="#">call</a>	Makes a one-to-one call.
<a href="#">groupCall</a>	Makes a group call.
<a href="#">joinInGroupCall</a>	Joins a group call.
<a href="#">enableMuteMode</a>	Sets whether to turn on the mute mode.
<a href="#">enableFloatWindow</a>	Sets whether to enable floating windows.
<a href="#">setCallingBell</a>	Custom ringtone.

### TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description
<a href="#">init</a>	Authenticates the basic audio/video call capabilities.
<a href="#">unInit</a>	The destructor function, which releases resources used by TUICallEngine.
<a href="#">addObserver</a>	Registers an event listener.

<code>removeObserver</code>	Unregisters an event listener.
<code>call</code>	Makes a one-to-one call.
<code>groupCall</code>	Makes a group call.
<code>accept</code>	Accepts a call.
<code>reject</code>	Rejects a call.
<code>hangup</code>	Ends a call.
<code>ignore</code>	Ignores a call.
<code>inviteUser</code>	Invites users to the current group call.
<code>joinInGroupCall</code>	Joins a group call.
<code>switchCallMediaType</code>	Changes the call type, for example, from video call to audio call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.
<code>openCamera</code>	Turns the camera on.
<code>closeCamera</code>	Turns the camera off.
<code>switchCamera</code>	Switches between the front and rear cameras.
<code>openMicrophone</code>	Turns the mic on.
<code>closeMicrophone</code>	Turns the mic off.
<code>selectAudioPlaybackDevice</code>	Selects the audio playback device (receiver or speaker).
<code>setSelfInfo</code>	Sets the alias and profile photo.
<code>enableMultiDeviceAbility</code>	Sets whether to enable multi-device login for TUICallEngine (supported by the premium package).
<code>setVideoRenderParams</code>	Set the rendering mode of video image.
<code>setVideoEncoderParams</code>	Set the encoding parameters of video encoder.
<code>queryRecentCalls</code>	Query call record.
<code>deleteRecordCalls</code>	Delete call record.
<code>setBeautyLevel</code>	Set beauty level, support turning off default beauty.



## TUICallObserver

`TUICallObserver` is the callback class of `TUICallEngine`. You can use it to listen for events.

API	Description
<code>onError</code>	An error occurred during the call.
<code>onCallReceived</code>	A call was received.
<code>onCallCancelled</code>	The call was canceled.
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user has a video stream.
<code>onUserAudioAvailable</code>	Whether a user has an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user is kicked offline
<code>onUserSigExpired</code>	Ticket expires while online

# TUICallKit

Last updated : 2024-03-12 14:51:07

## TUICallKit APIs

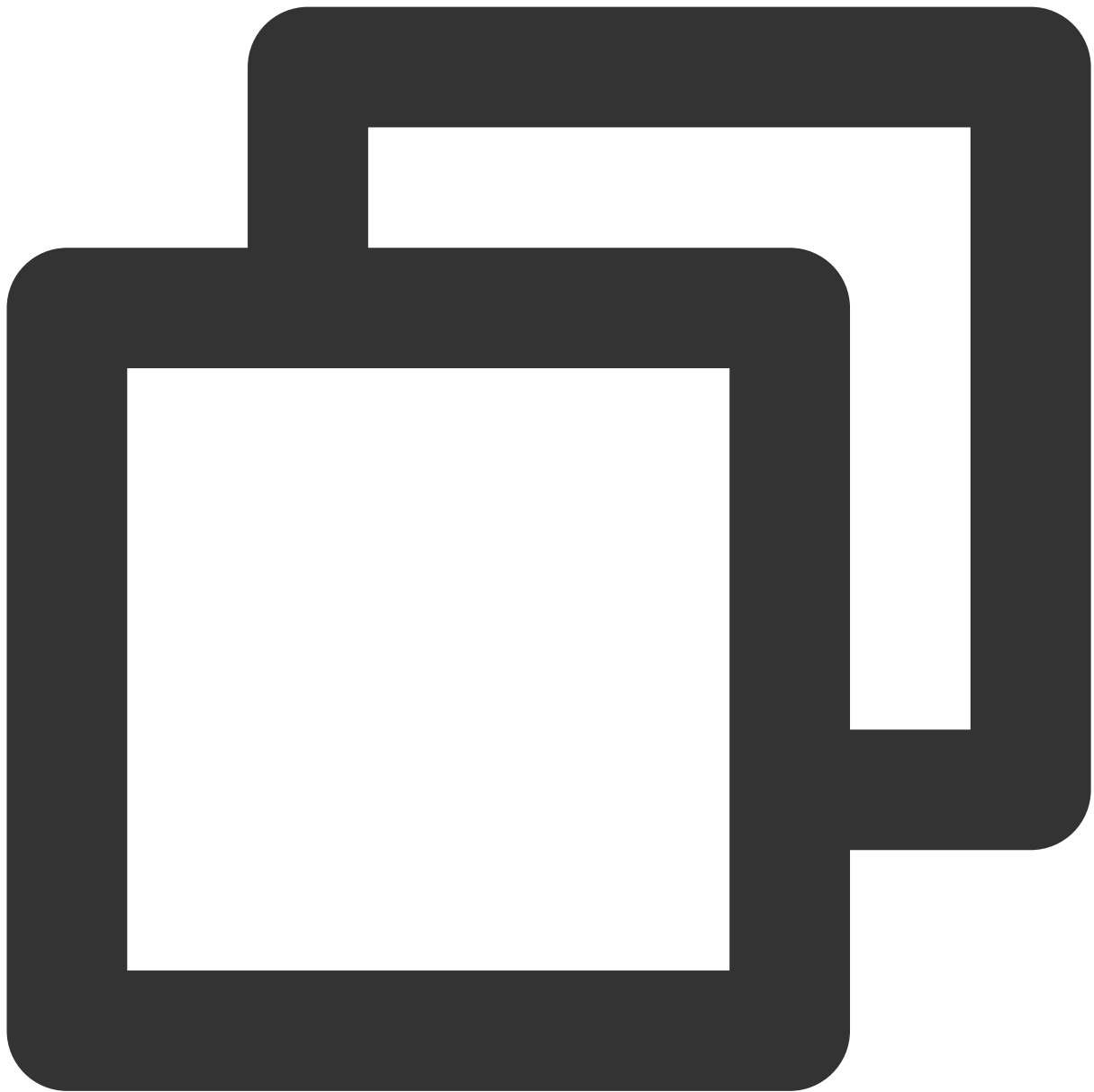
`TUICallKit` is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat. For directions on integration, see [Integrating TUICallKit](#).

## API overview

API	Description
<a href="#">login</a>	Log in
<a href="#">logout</a>	Sign out
<a href="#">setSelfInfo</a>	Sets the user nickname and profile photo.
<a href="#">call</a>	Makes a one-to-one call.
<a href="#">groupCall</a>	Makes a group call.
<a href="#">joinInGroupCall</a>	Joins a group call.
<a href="#">enableMuteMode</a>	Sets whether to turn on the mute mode.
<a href="#">enableFloatWindow</a>	Sets whether to enable floating windows.
<a href="#">setCallingBell</a>	Custom ringtone.

## Details

### login

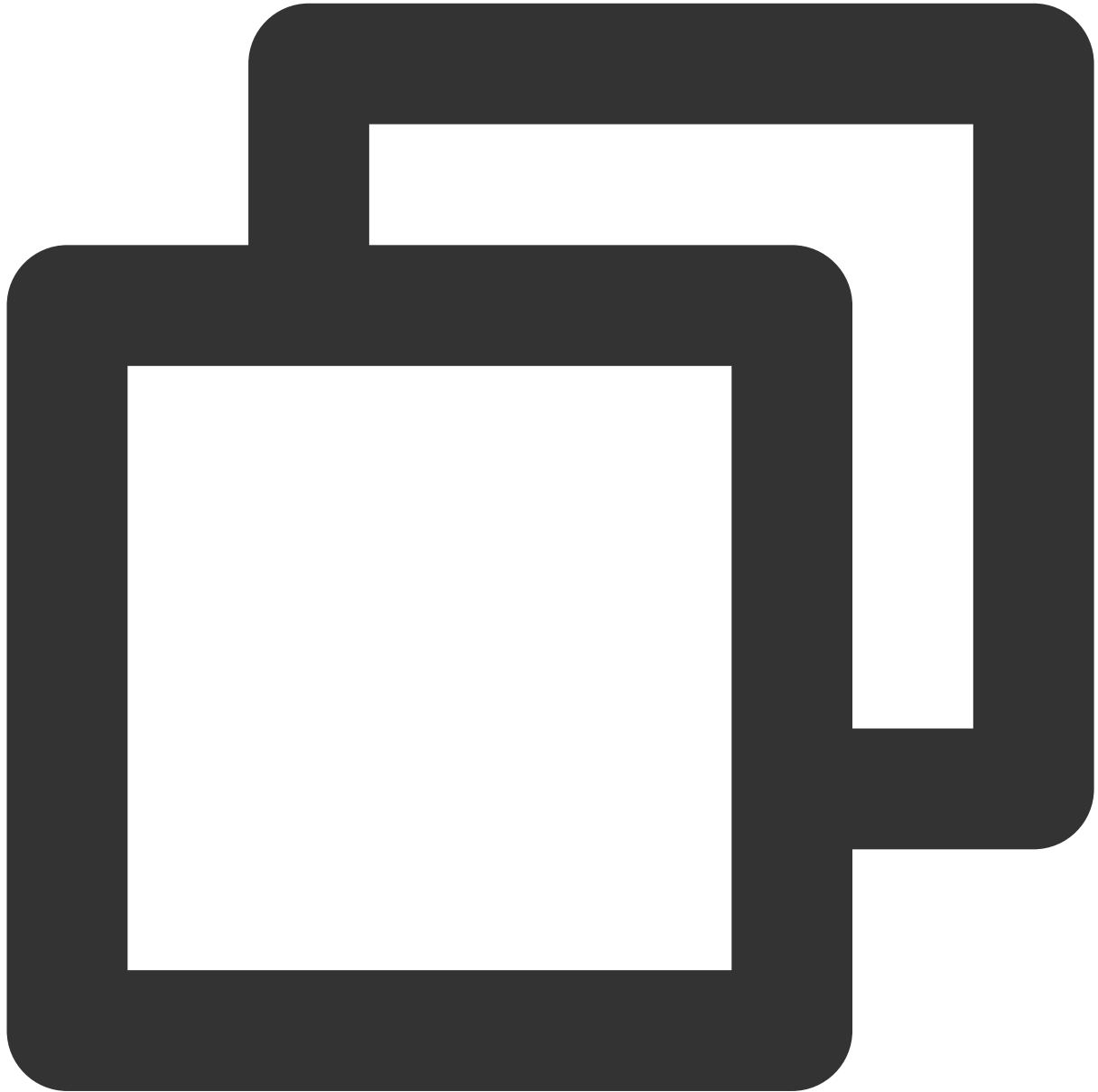


```
Future<TUIResult> login(int sdkAppId, String userId, String userSig)
```

Parameter	Type	Description
sdkAppId	int	User SDKAppID
userId	String	User ID
userSig	String	User Signature ( <a href="#">Signature calculation method</a> )
return value	<a href="#">TUIResult</a>	Contains code and message information: code = 0 means the call is

	successful; code != 0 means the call failed, see message for the reason of failure
--	--

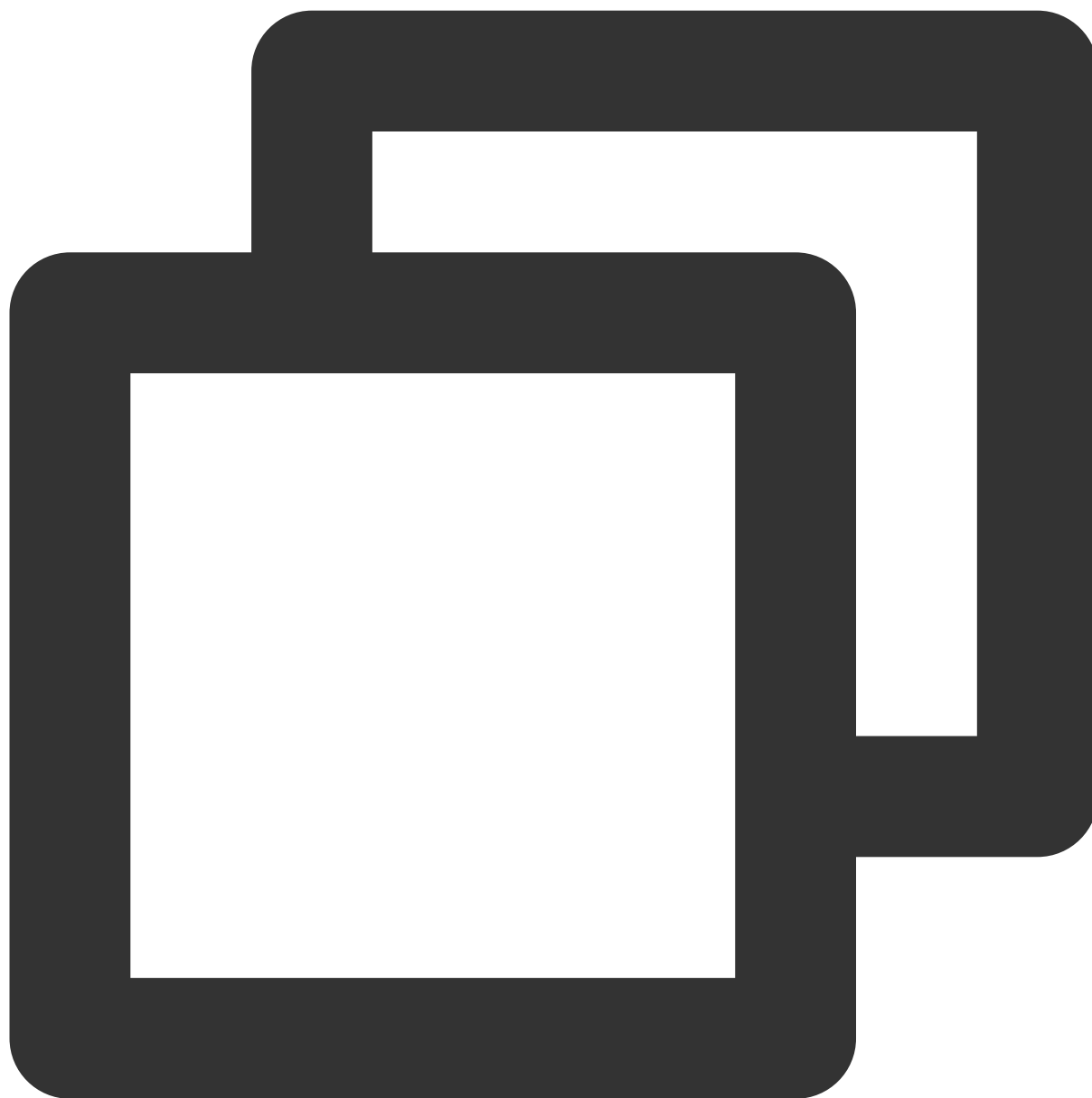
## logout



```
Future<void> logout ()
```

## setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.



```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

Parameter	Type	Description
nickName	String	The alias.
avatar	String	The profile photo.

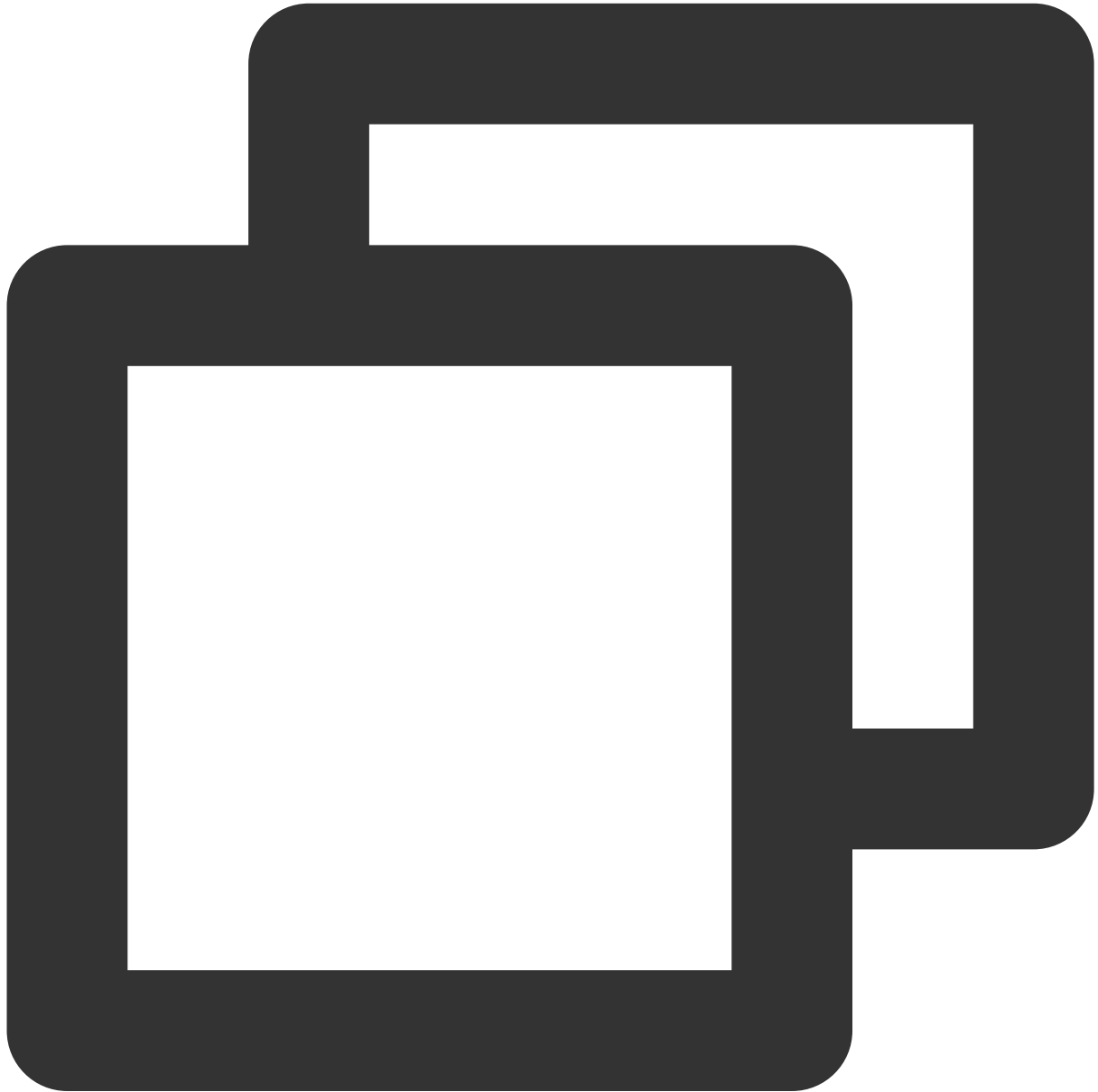
return value

[TUIResult](#)

Contains code and message information: code = 0 means the call is successful; code != 0 means the call failed, see message for the reason of failure

## call

This API is used to make a (one-to-one) call.



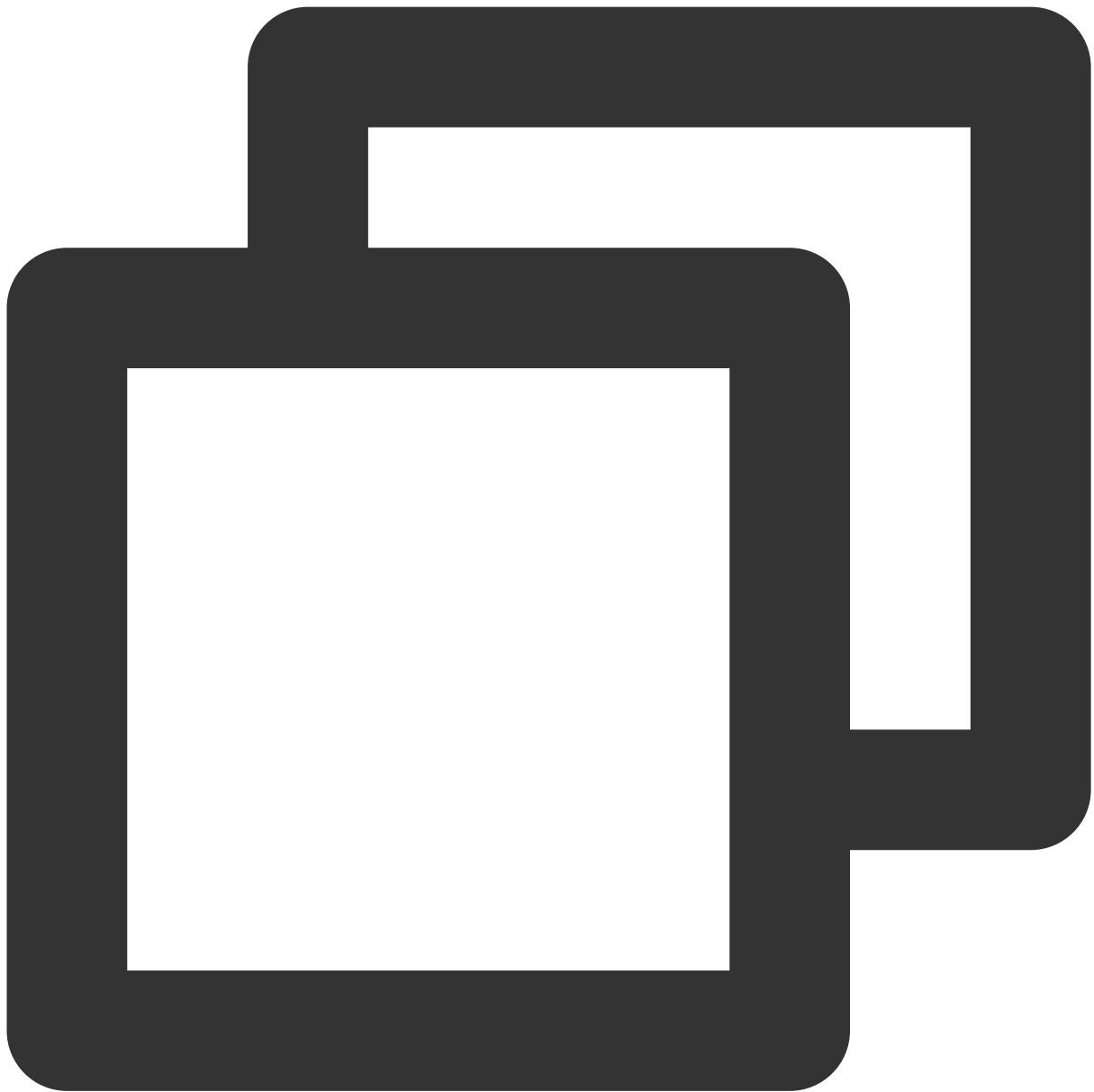
```
Future<void> call(String userId, TUICallMediaType callMediaType, [TUICallParams? pa
```

The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.

## groupCall

This API is used to make a group call.



```
Future<void> groupCall(String groupId, List<String> userIdList, TUICallMediaType ca
```

The parameters are described below:

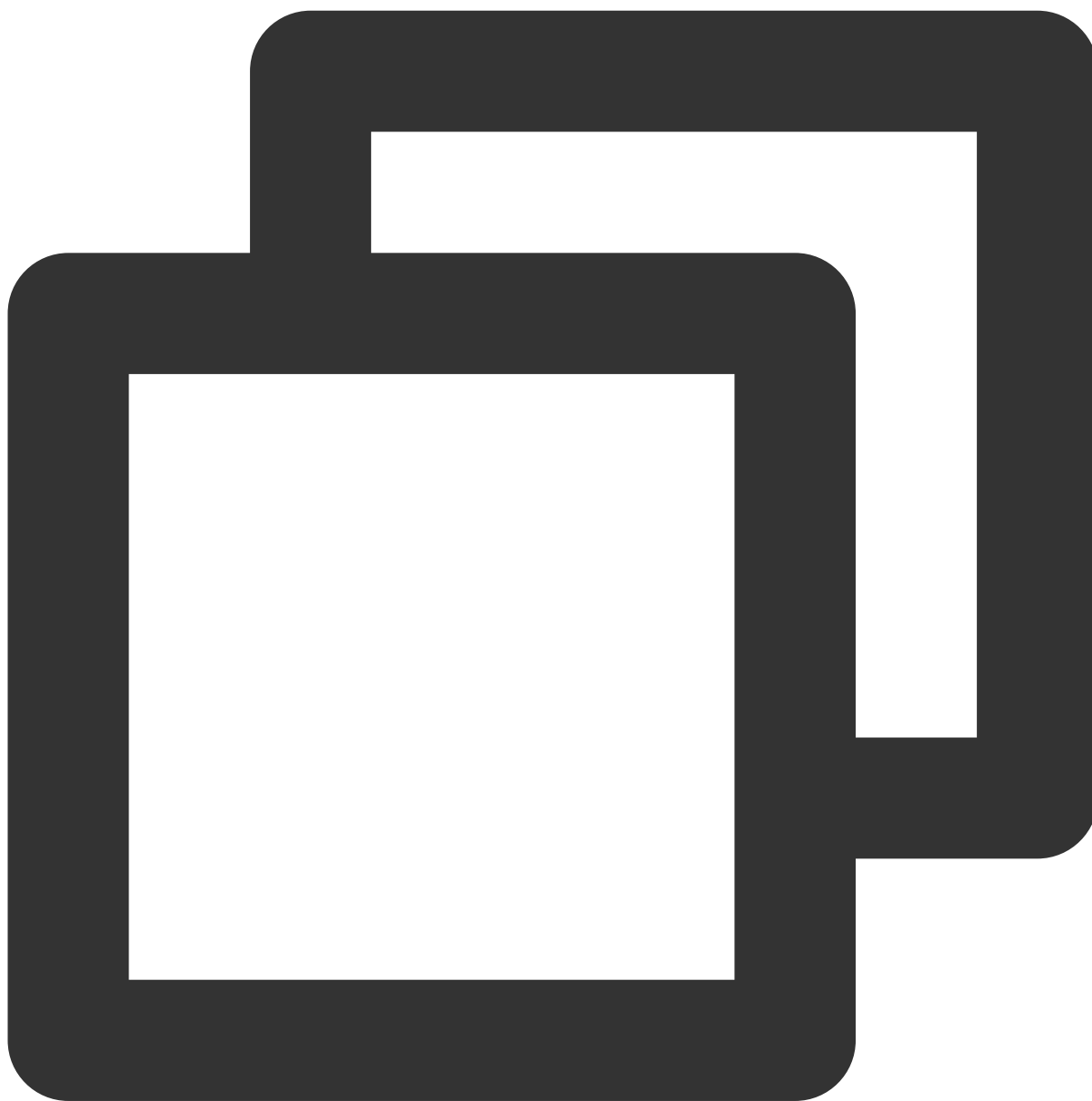
Parameter	Type	Description
groupId	String	The group ID.
userIdList	List<String>	The target user IDs.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or



```
audio.
```

## joinInGroupCall

This API is used to join a group call.



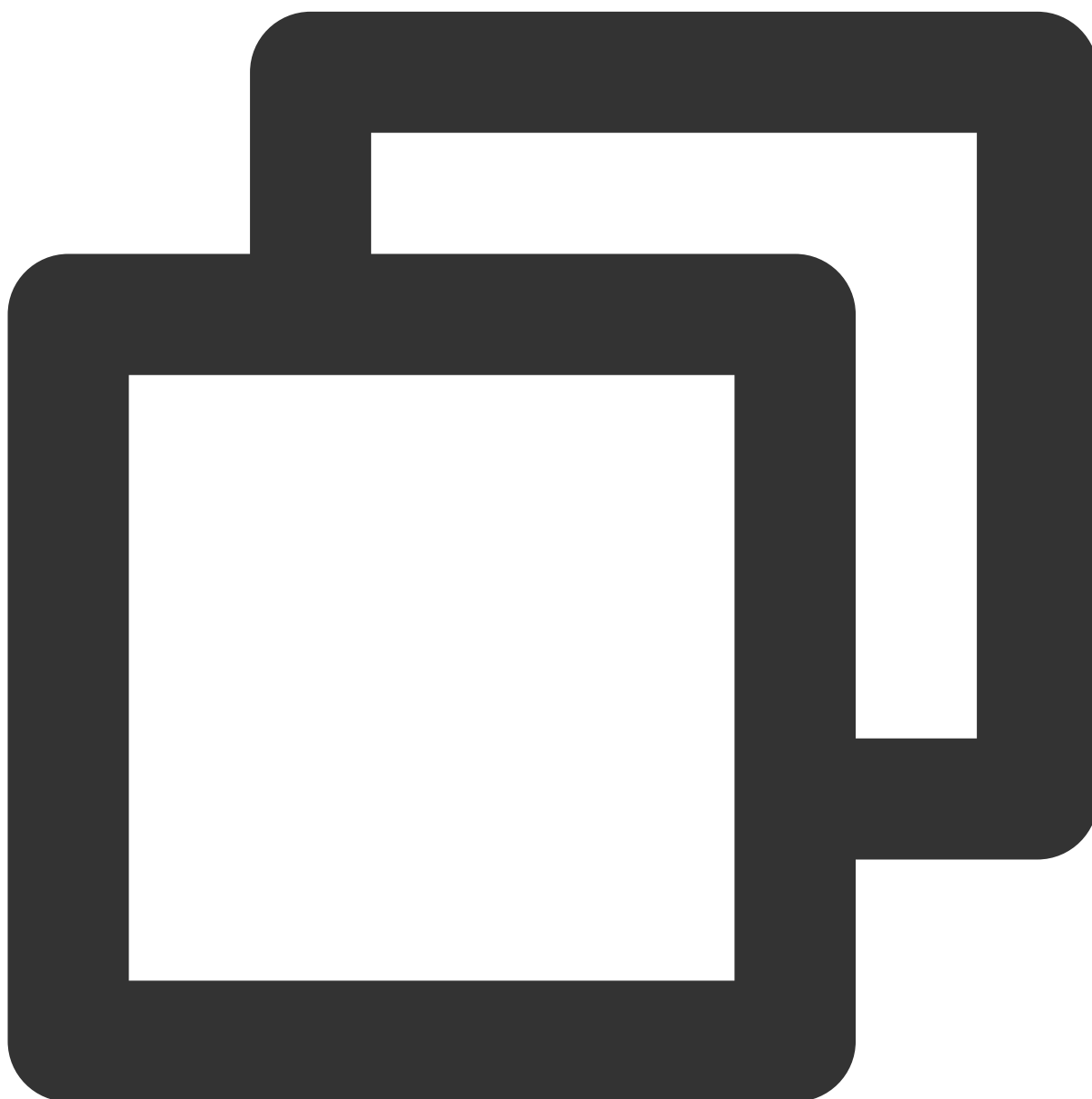
```
Future<void> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaType cal
```

Parameter	Type	Description
roomId	<a href="#">TUIRoomID</a>	The room ID.

groupId	String	The group ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.

## enableMuteMode

This API is used to set whether to turn on the mute mode.

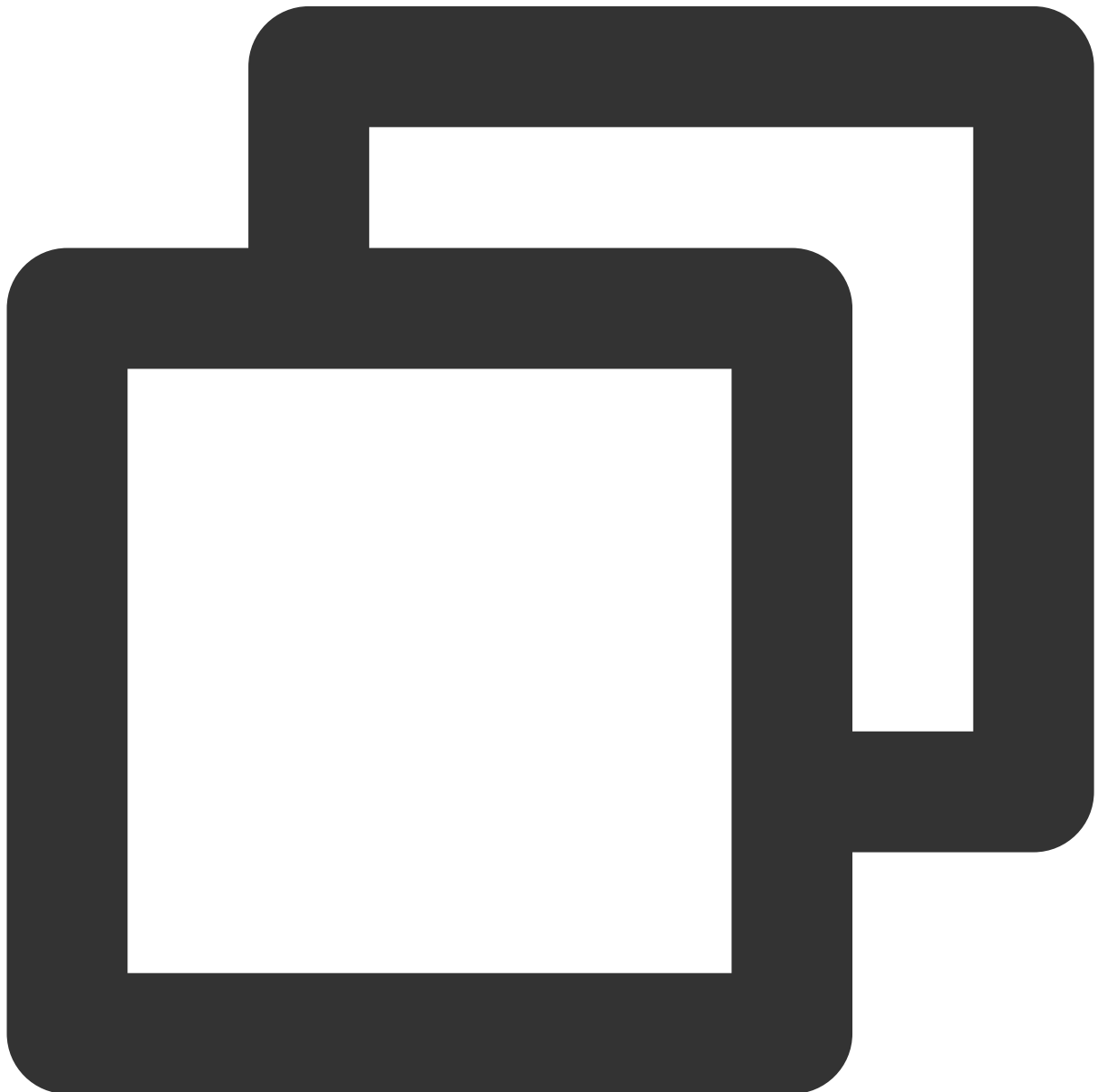


```
Future<void> enableMuteMode(bool enable)
```

Parameter	Type	Description
enable	Boolean	Turn on and off the mute; true means to turn on the mute

## enableFloatWindow

This API is used to set whether to enable floating windows. The default value is `false` , and the floating window button in the top left corner of the call view is hidden. If it is set to `true` , the button will become visible.

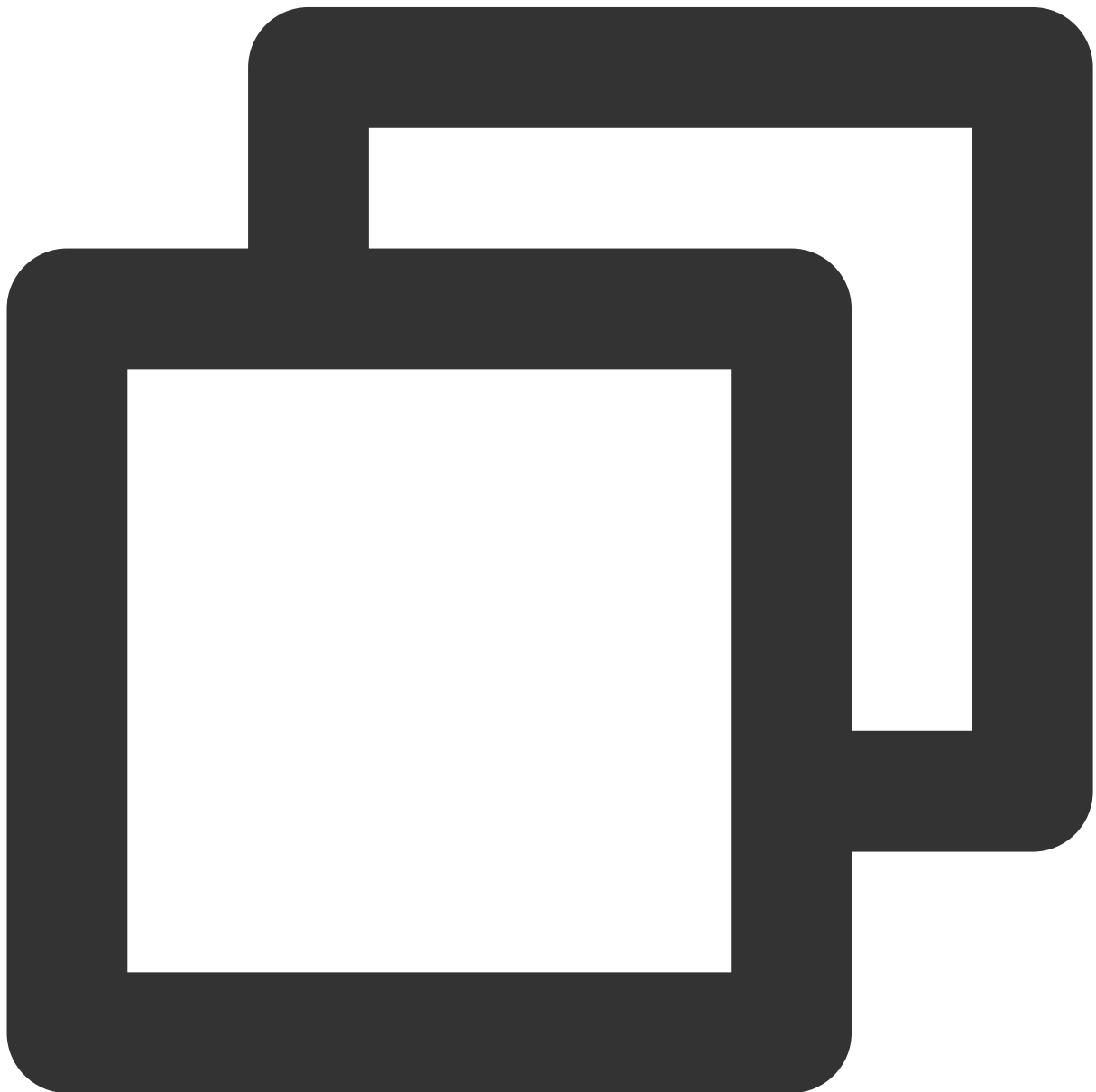


```
Future<void> enableFloatWindow(bool enable)
```

Parameter	Type	Description
enable	Boolean	The default value is false, and the floating window button in the top left corner of the call view is hidden. If it is set to true, the button will become visible.

## setCallingBell

Custom ringtone.



```
Future<void> setCallingBell(String assetName)
```

Parameter	Type	Description
assetName	String	The path of the ringtone. The ringtone file needs to be added to the assets resource of the main project.

# TUICallEngine

Last updated : 2023-09-19 16:18:05

## TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

## Overview

API	Description
<code>init</code>	Authenticates the basic audio/video call capabilities.
<code>unInit</code>	The destructor function, which releases resources used by TUICallEngine.
<code>addObserver</code>	Registers an event listener.
<code>removeObserver</code>	Unregisters an event listener.
<code>call</code>	Makes a one-to-one call.
<code>groupCall</code>	Makes a group call.
<code>accept</code>	Accepts a call.
<code>reject</code>	Rejects a call.
<code>hangup</code>	Ends a call.
<code>ignore</code>	Ignores a call.
<code>inviteUser</code>	Invites users to the current group call.
<code>joinInGroupCall</code>	Joins a group call.
<code>switchCallMediaType</code>	Changes the call type, for example, from video call to audio call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.

<a href="#">openCamera</a>	Turns the camera on.
<a href="#">closeCamera</a>	Turns the camera off.
<a href="#">switchCamera</a>	Switches between the front and rear cameras.
<a href="#">openMicrophone</a>	Turns the mic on.
<a href="#">closeMicrophone</a>	Turns the mic off.
<a href="#">selectAudioPlaybackDevice</a>	Selects the audio playback device (receiver or speaker).
<a href="#">setSelfInfo</a>	Sets the alias and profile photo.
<a href="#">enableMultiDeviceAbility</a>	Sets whether to enable multi-device login for TUICallEngine (supported by the premium package).
<a href="#">setVideoRenderParams</a>	Set the rendering mode of video image.
<a href="#">setVideoEncoderParams</a>	Set the encoding parameters of video encoder.
<a href="#">queryRecentCalls</a>	Query call record.
<a href="#">deleteRecordCalls</a>	Delete call record.
<a href="#">setBeautyLevel</a>	Set beauty level, support turning off default beauty.

## Details

### init

This API is used to initialize `TUICallEngine` . Call it to authenticate the call service and perform other required actions before you call other APIs.



```
Future<TUIResult> init(int sdkAppID, String userId, String userSig)
```

## **unInit**

The destructor function, which releases resources used by TUICallEngine.





```
Future<TUIResult> unInit ()
```

## addObserver

This API is used to register an event listener to listen for `TUICallObserver` events.



```
Future<void> addObserver(TUICallObserver observer)
```

### **removeObserver**

This API is used to unregister an event listener.



```
Future<void> removeObserver(TUICallObserver observer)
```

## call

This API is used to make a (one-to-one) call.



```
Future<TUIResult> call(String userId, TUICallMediaType mediaType, TUICallParams par
```

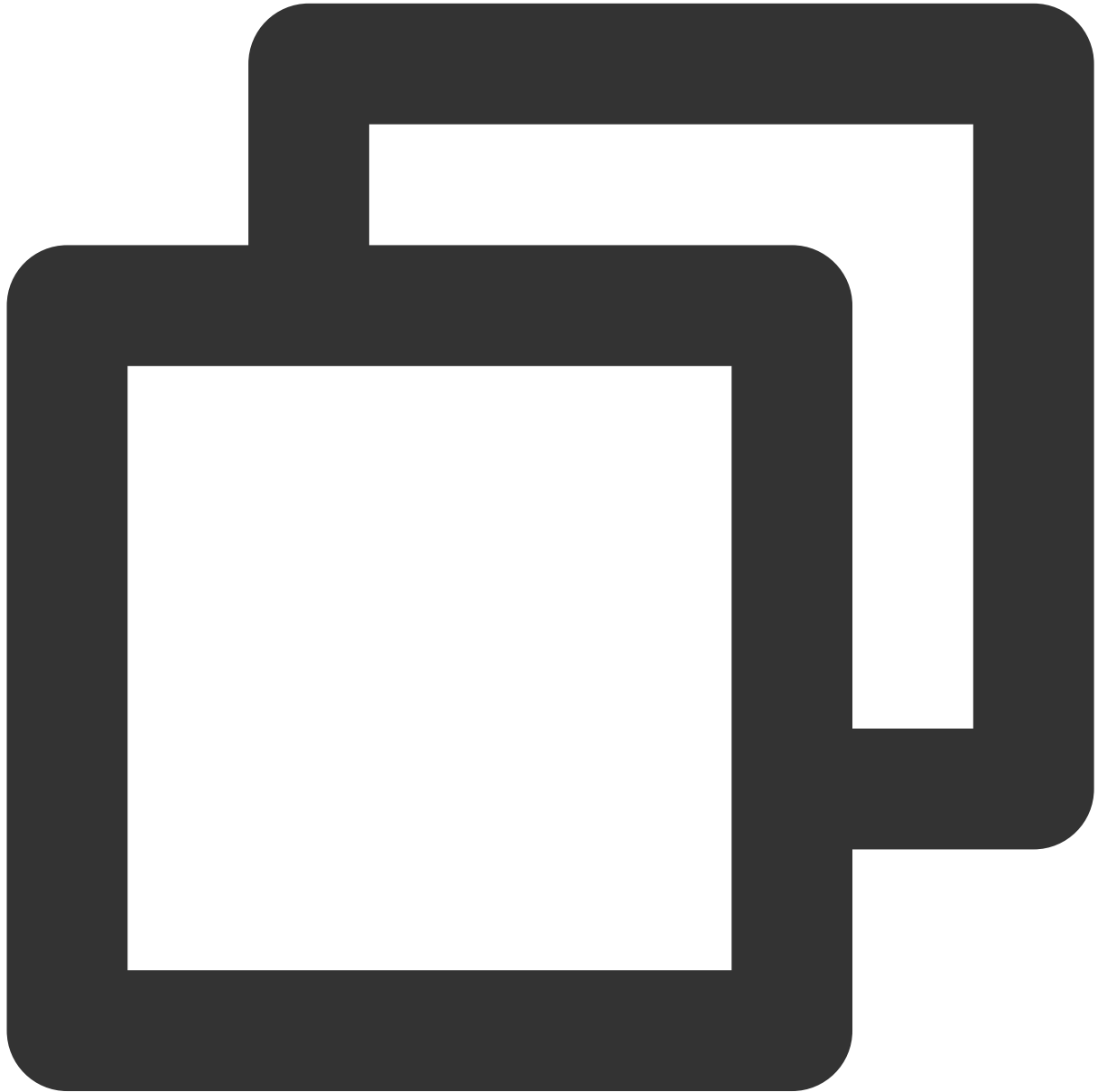
Parameter	Type	Description
userId	String	The target user ID.
mediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
params	<a href="#">TUICallParams</a>	An additional parameter, such as roomId, call timeout, offline push info, etc

## groupCall

This API is used to make a group call.

### Notice :

Before making a group call, you need to create an IM group first.



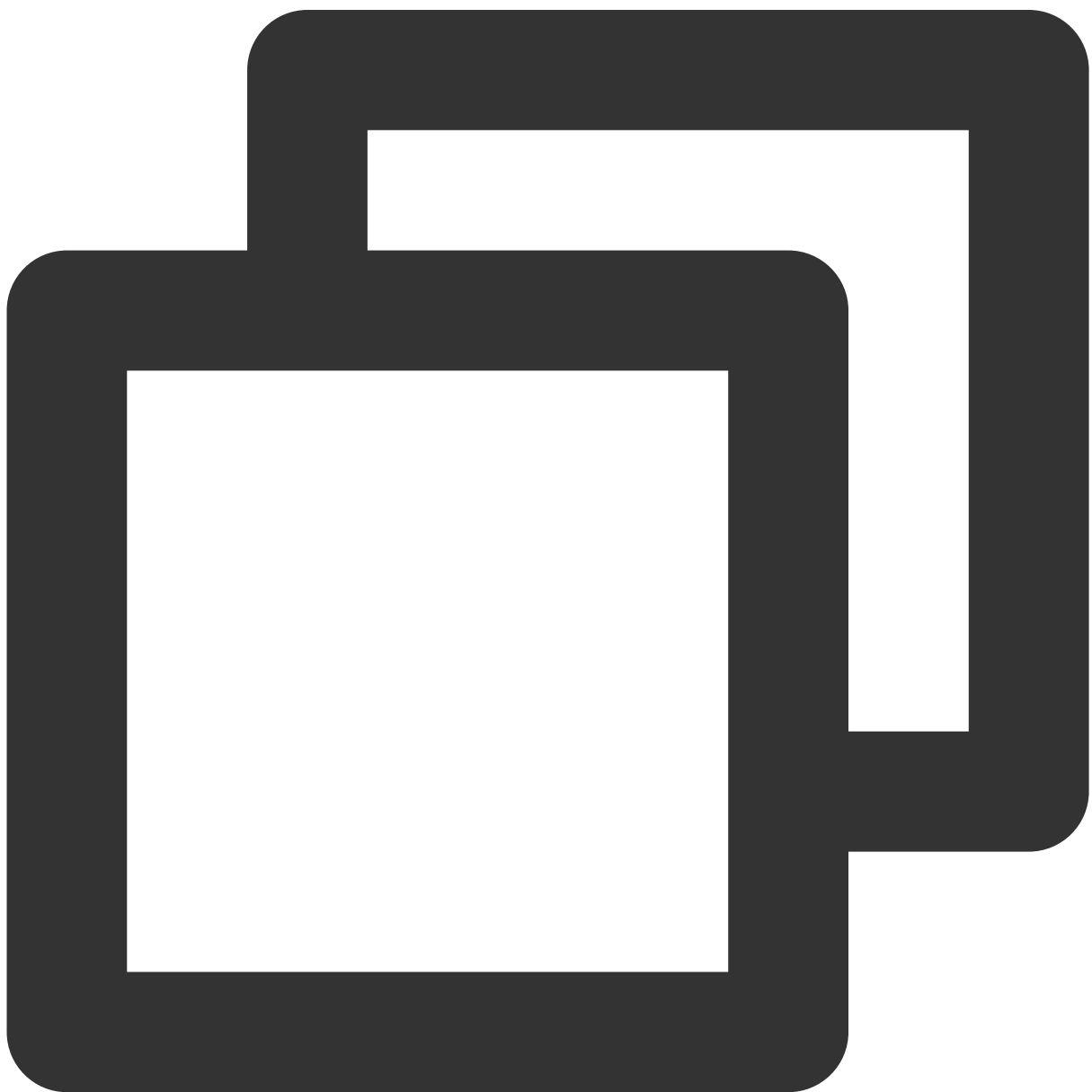
```
Future<TUIResult> groupCall(String groupId, List<String> userIdList, TUICallMediaTy
```

Parameter	Type	Description
groupId	String	The group ID.

userIdList	List<String>	The target user IDs.
mediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
params	<a href="#">TUICallParams</a>	An additional parameter. such as roomId, call timeout, offline push info, etc

## accept

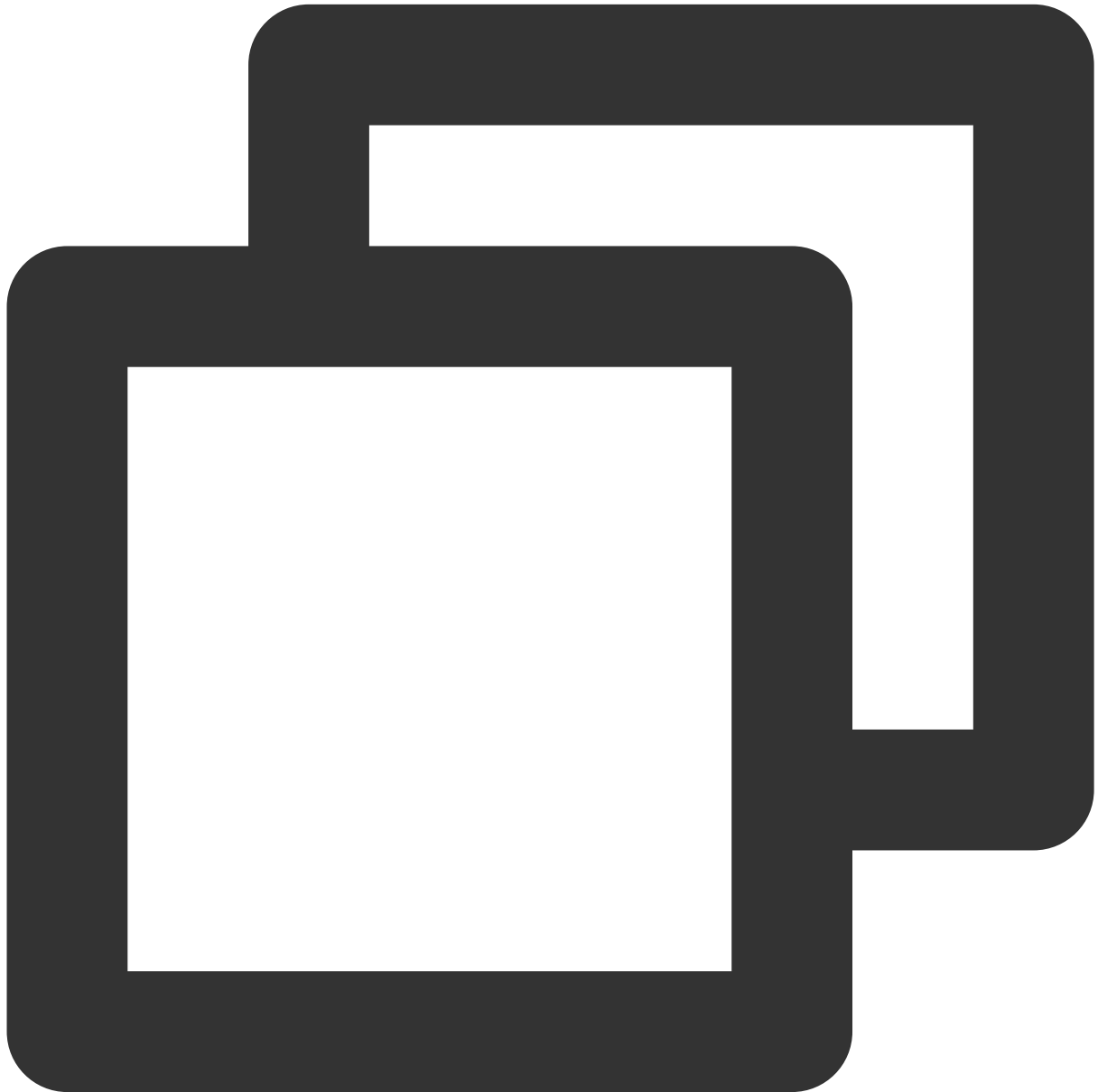
This API is used to accept a call. After receiving the `onCallReceived()` callback, you can call this API to accept the call.



```
Future<TUIResult> accept()
```

## reject

This API is used to reject a call. After receiving the `onCallReceived()` callback, you can call this API to reject the call.

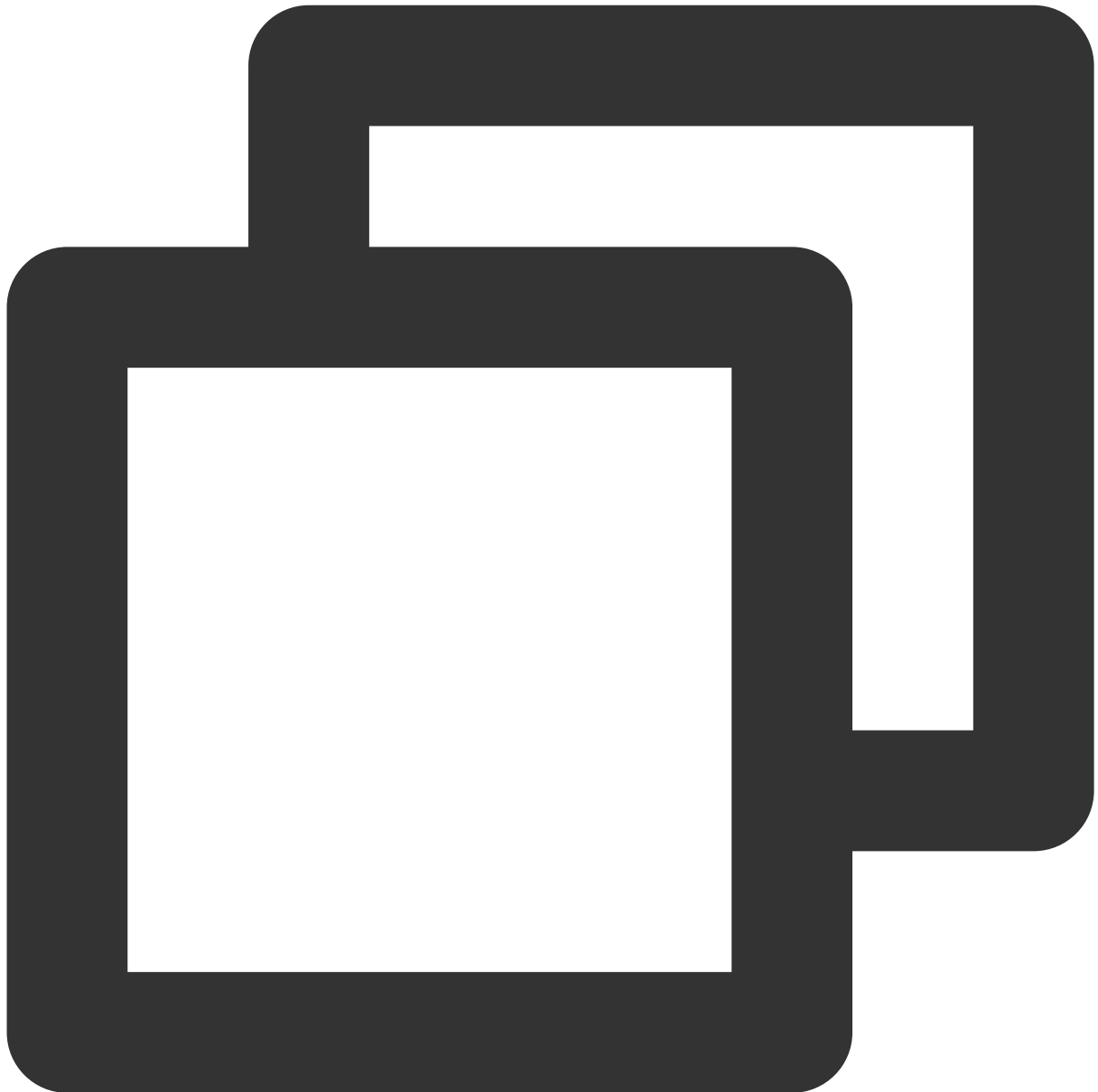


```
Future<TUIResult> reject()
```

## ignore

This API is used to ignore a call. After receiving the `onCallReceived()` , you can call this API to ignore the call. The caller will receive the `onUserLineBusy` callback.

Note: If your project involves live streaming or conferencing, you can also use this API to implement the “in a meeting” or “on air” feature.

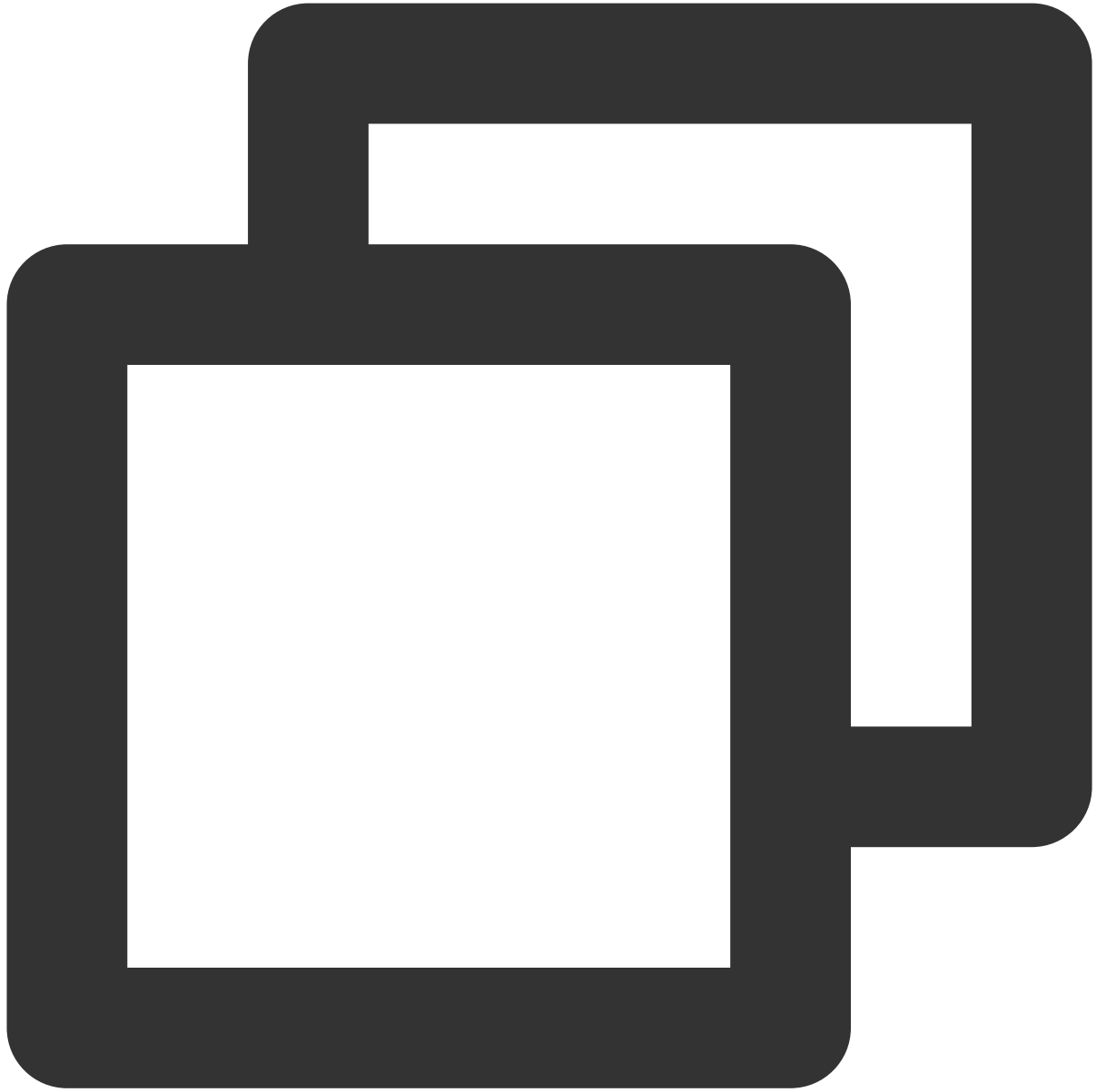


```
Future<TUIResult> ignore()
```

## hangup



This API is used to end a call.

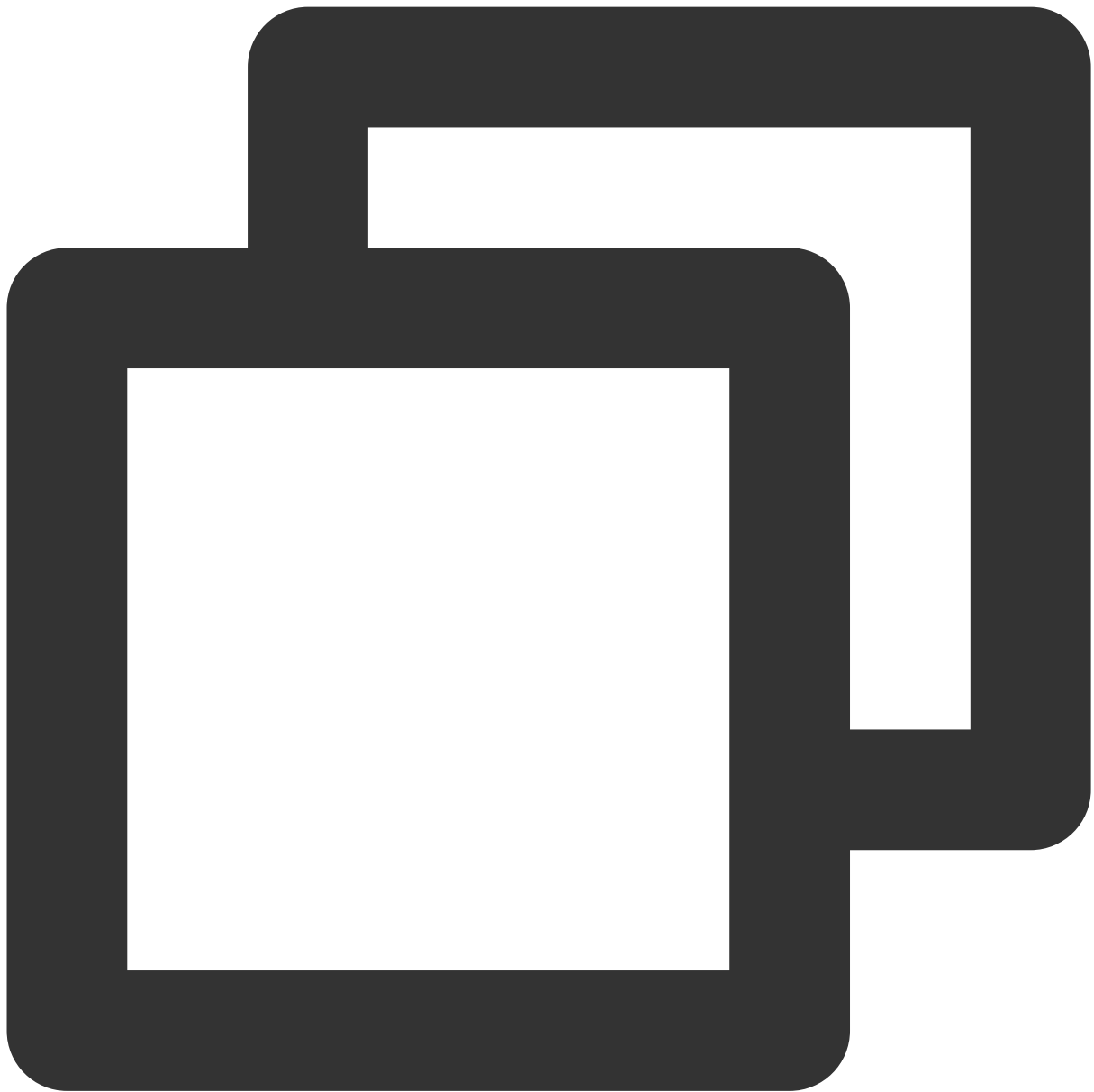


```
Future<TUIResult> hangup()
```

## inviteUser

This API is used to invite users to the current group call.

This API is called by a participant of a group call to invite new users.



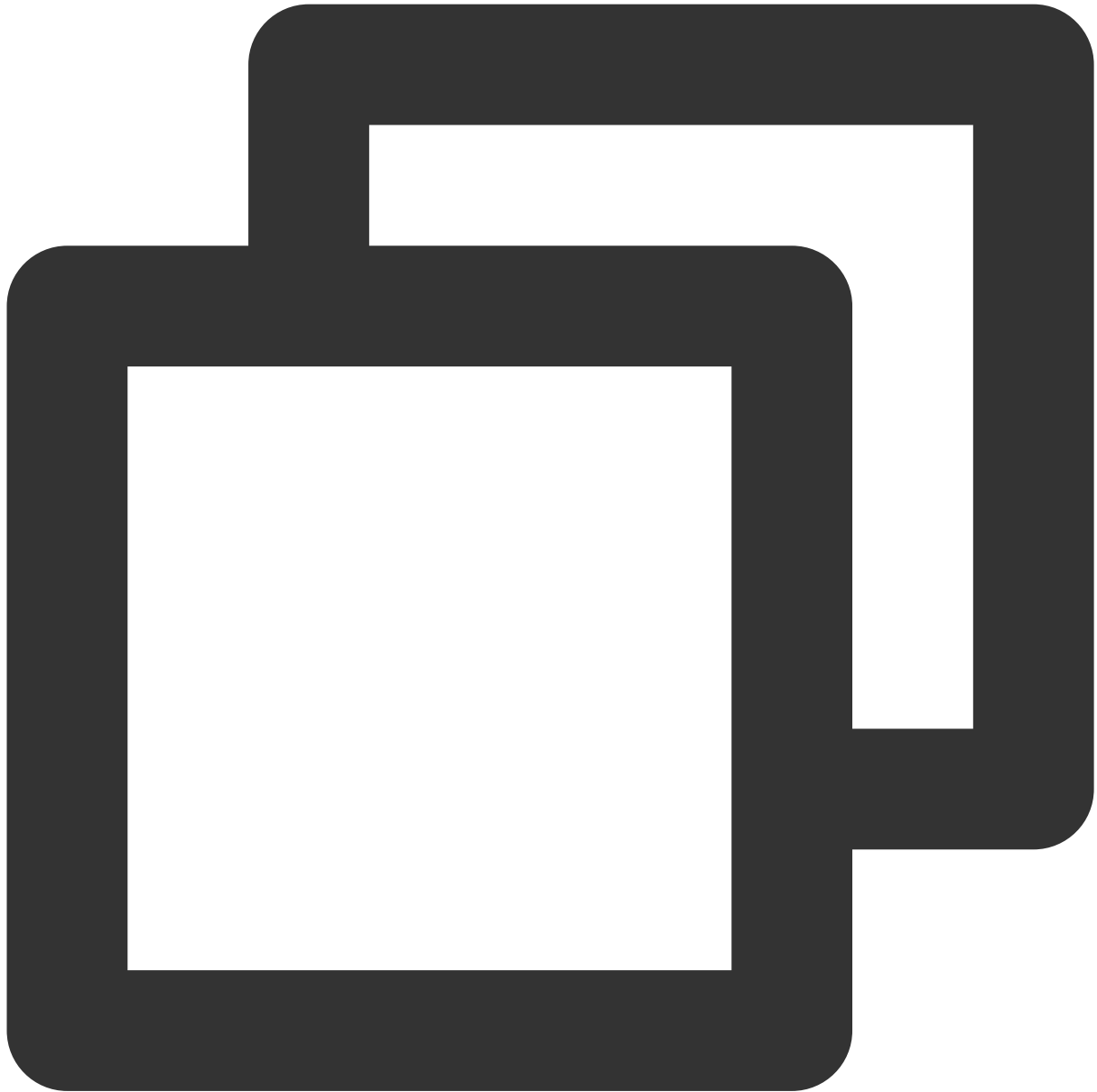
```
Future<void> ininviteUser(List<String> userIdList, TUICallParams params, TUIValueCal
```

Parameter	Type	Description
userIdList	List<String>	The target user IDs.
params	<a href="#">TUICallParams</a>	An additional parameter. such as roomId, call timeout, offline push info, etc.

## joinInGroupCall

This API is used to join a group call.

This API is called by a group member to join the group's call.



```
Future<TUIResult> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaTyp
```

Parameter	Type	Description
roomId	<a href="#">TUIRoomId</a>	The room ID.
groupId	String	The group ID.

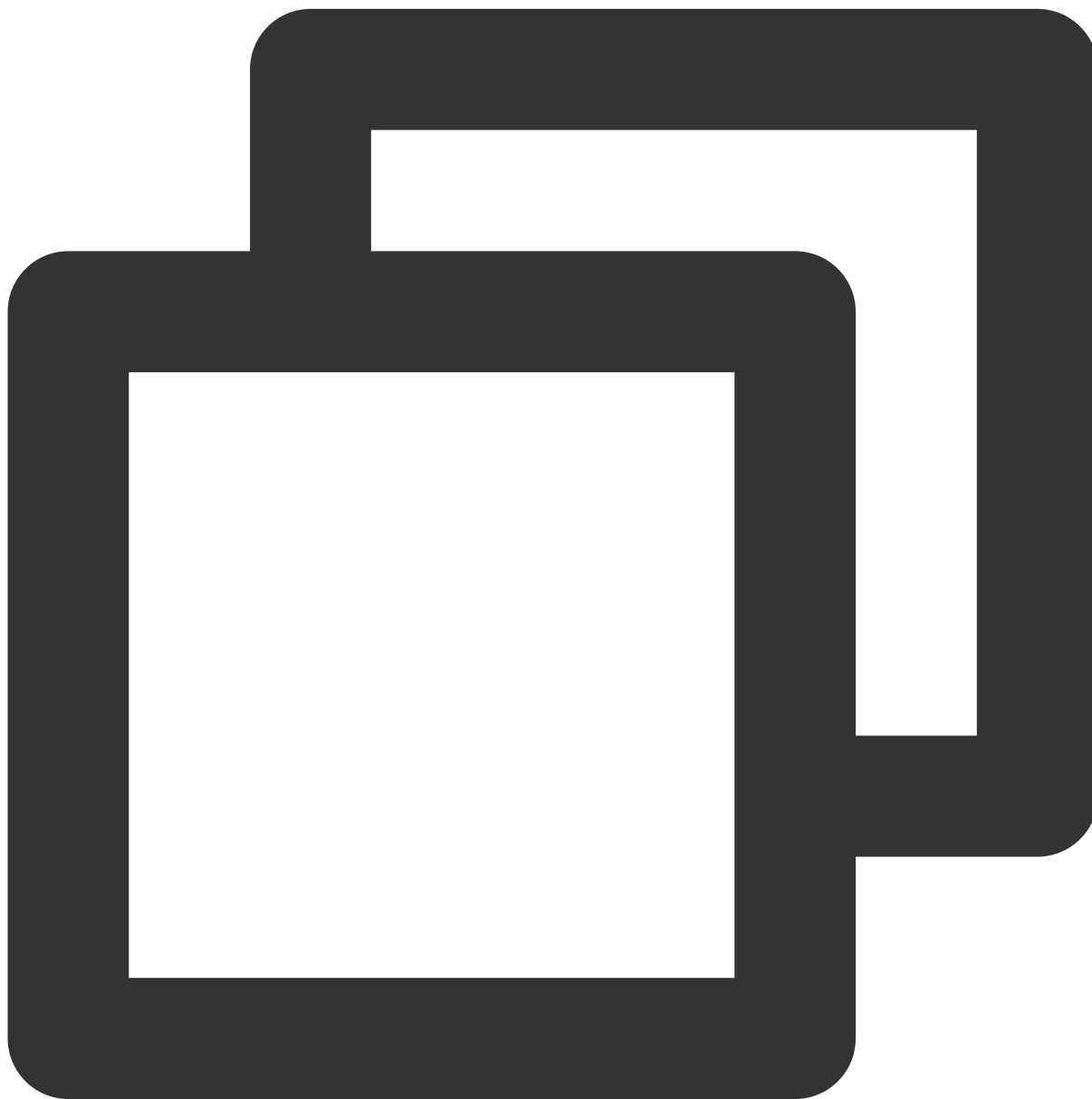
mediaType

[TUICallMediaType](#)

The call type, which can be video or audio.

## switchCallMediaType

This API is used to change the call type.

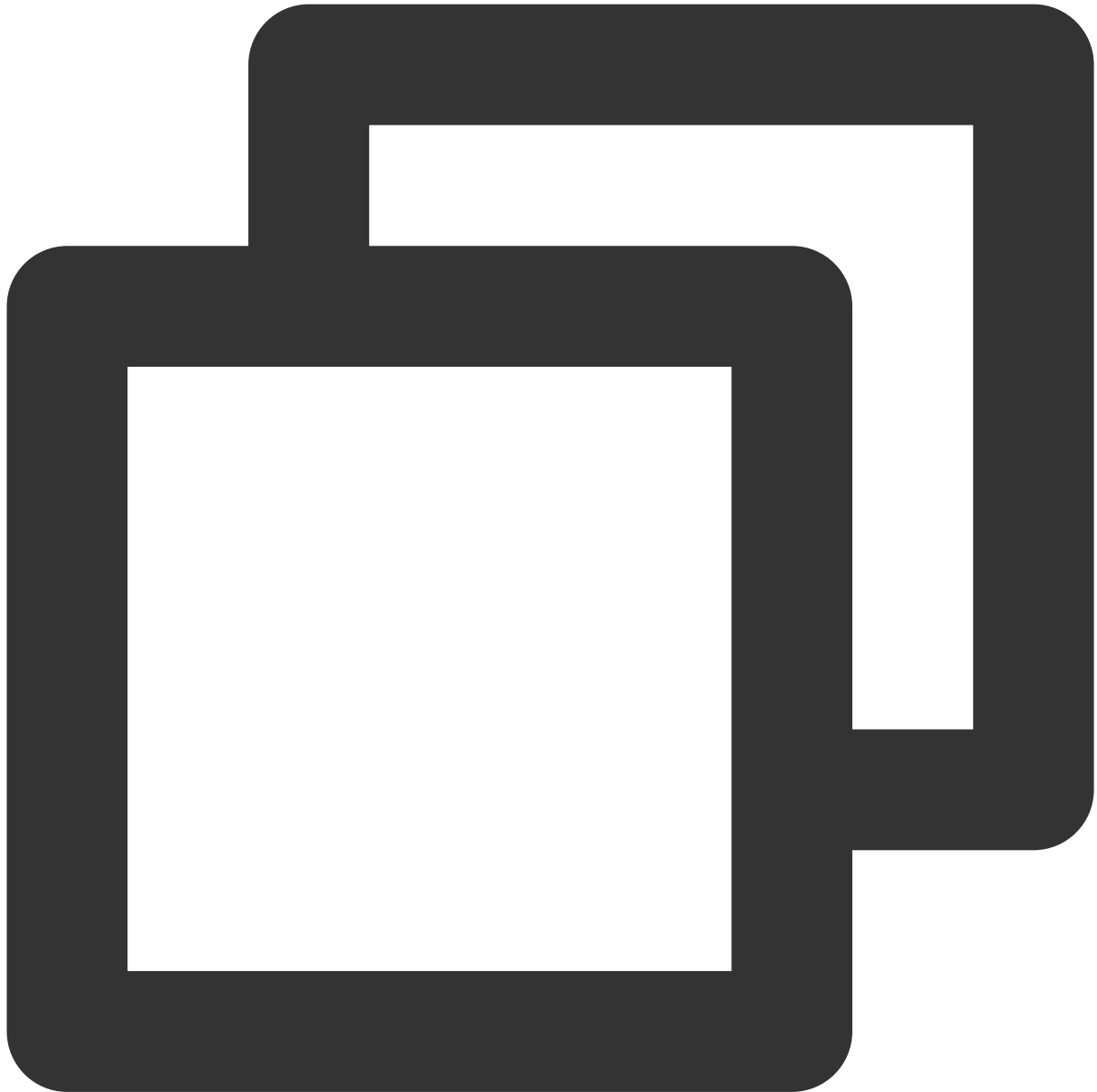


```
Future<void> switchCallMediaType(TUICallMediaType mediaType)
```

Parameter	Type	Description
mediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.

## startRemoteView

This API is used to set the view object to display a remote video.

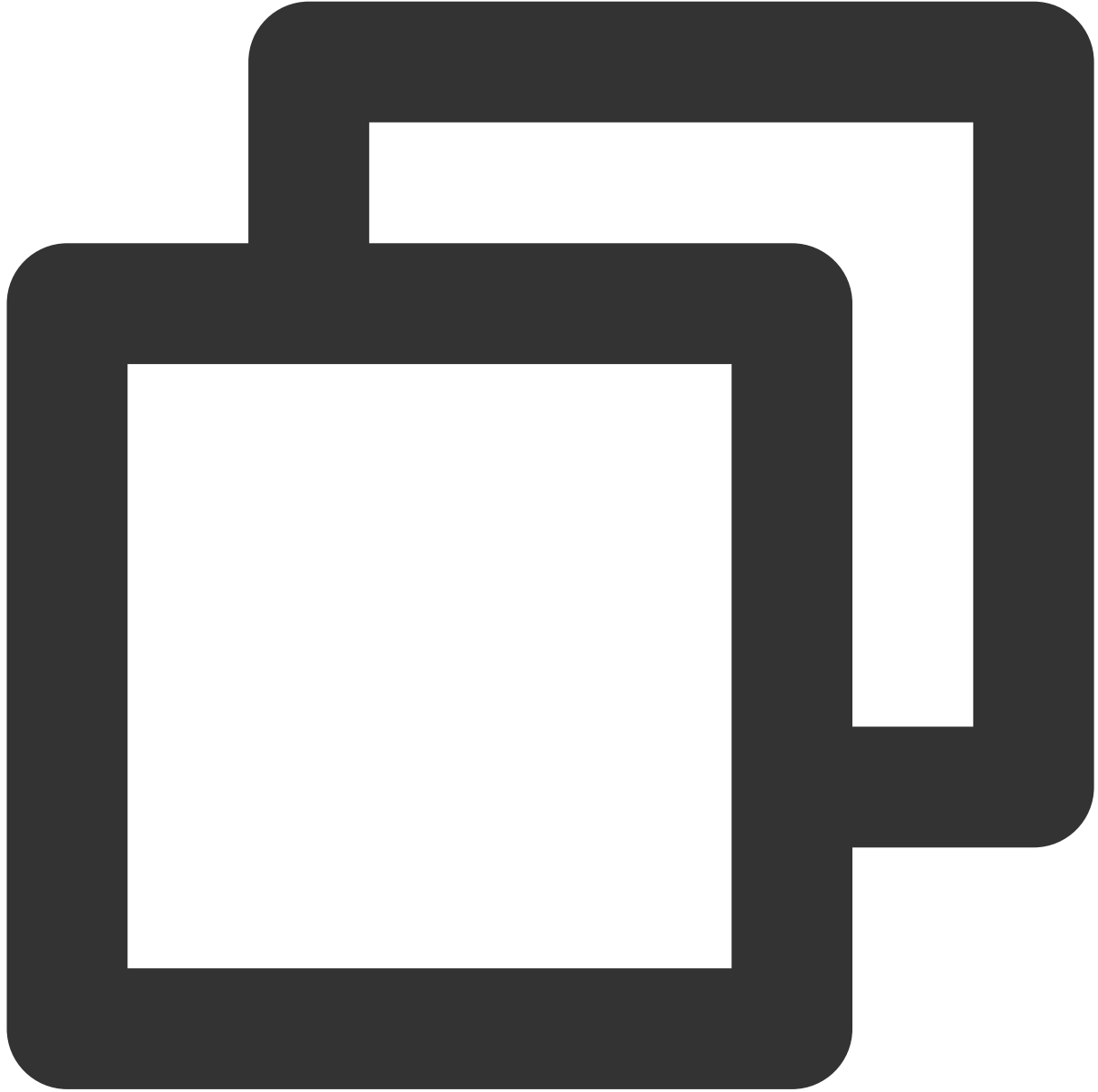


```
Future<void> startRemoteView(String userId, intviewId)
```

Parameter	Type	Description
userId	String	The target user ID.
intviewId	int	The ID of the widget in the video rendering screen

## stopRemoteview

This API is used to unsubscribe from the video stream of a remote user.

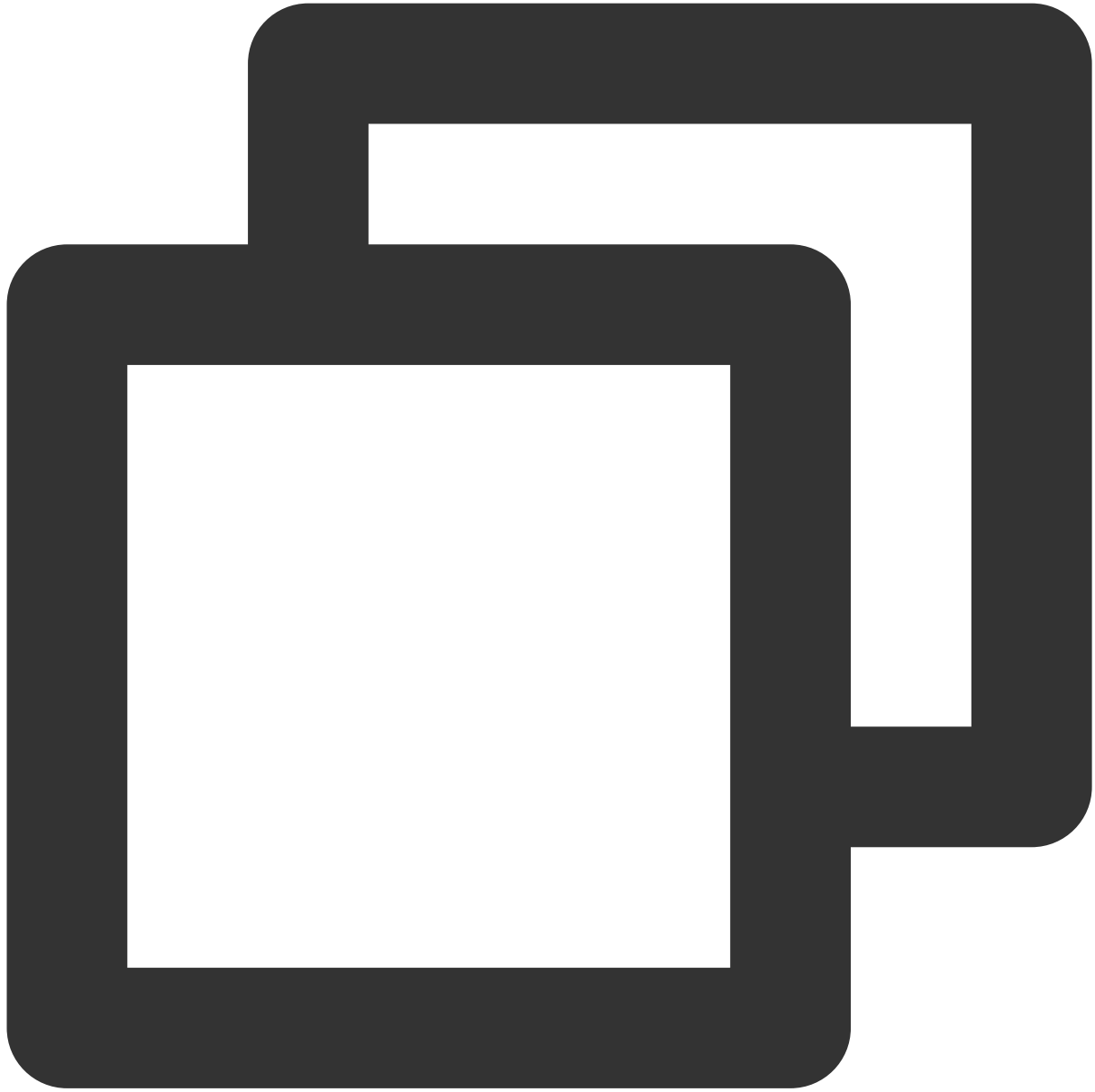


```
Future<void> stopRemoteView(String userId)
```

Parameter	Type	Description
userId	String	The target user ID.

## openCamera

This API is used to turn the camera on.

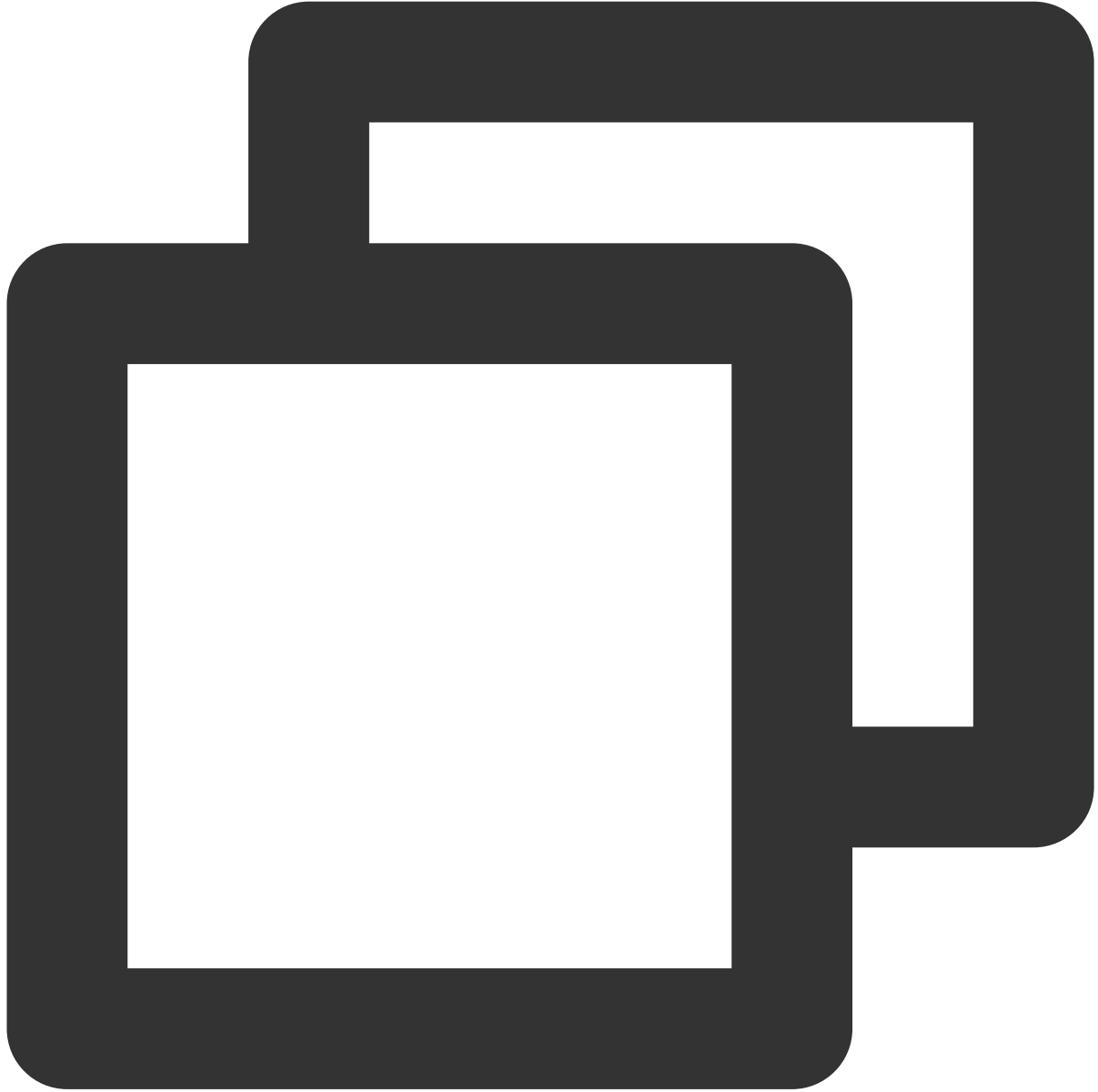


```
Future<TUIResult> openCamera(TUICamera camera, int? viewId)
```

Parameter	Type	Description
camera	<a href="#">TUICamera</a>	The front or rear camera.
viewId	int	The ID of the widget in the video rendering screen

## closeCamera

This API is used to turn the camera off.



```
Future<void> closeCamera ()
```

## switchCamera

This API is used to switch between the front and rear cameras.





```
Future<void> switchCamera(TUICamera camera)
```

Parameter	Type	Description
camera	<a href="#">TUICamera</a>	The front or rear camera.

## openMicrophone

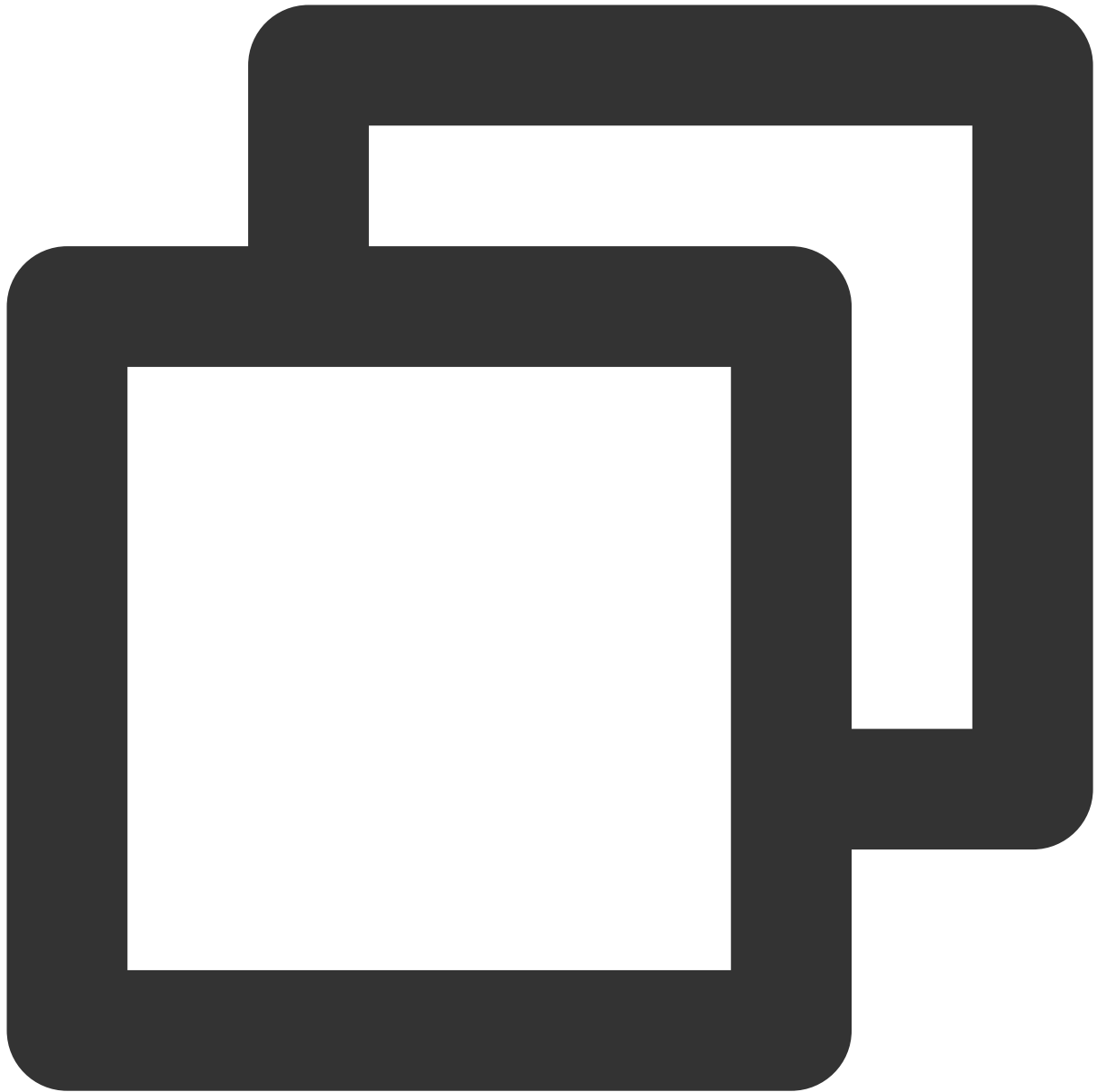
This API is used to turn the mic on.



```
Future<TUIResult> openMicrophone()
```

### **closeMicrophone**

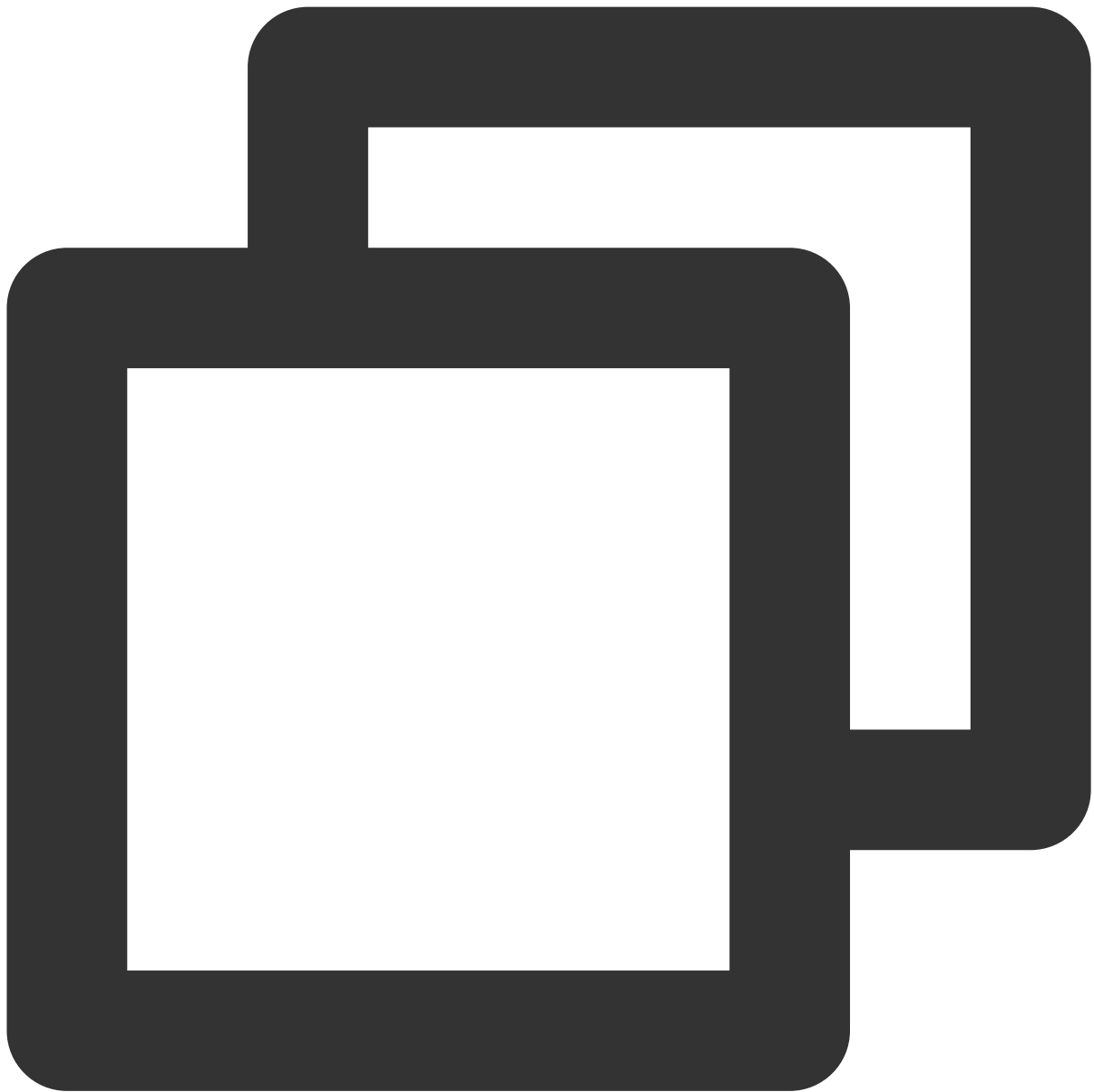
This API is used to turn the mic off.



```
Future<void> closeMicrophone()
```

### **selectAudioPlaybackDevice**

This API is used to select the audio playback device (receiver or speaker). In call scenarios, you can use this API to turn on/off hands-free mode.

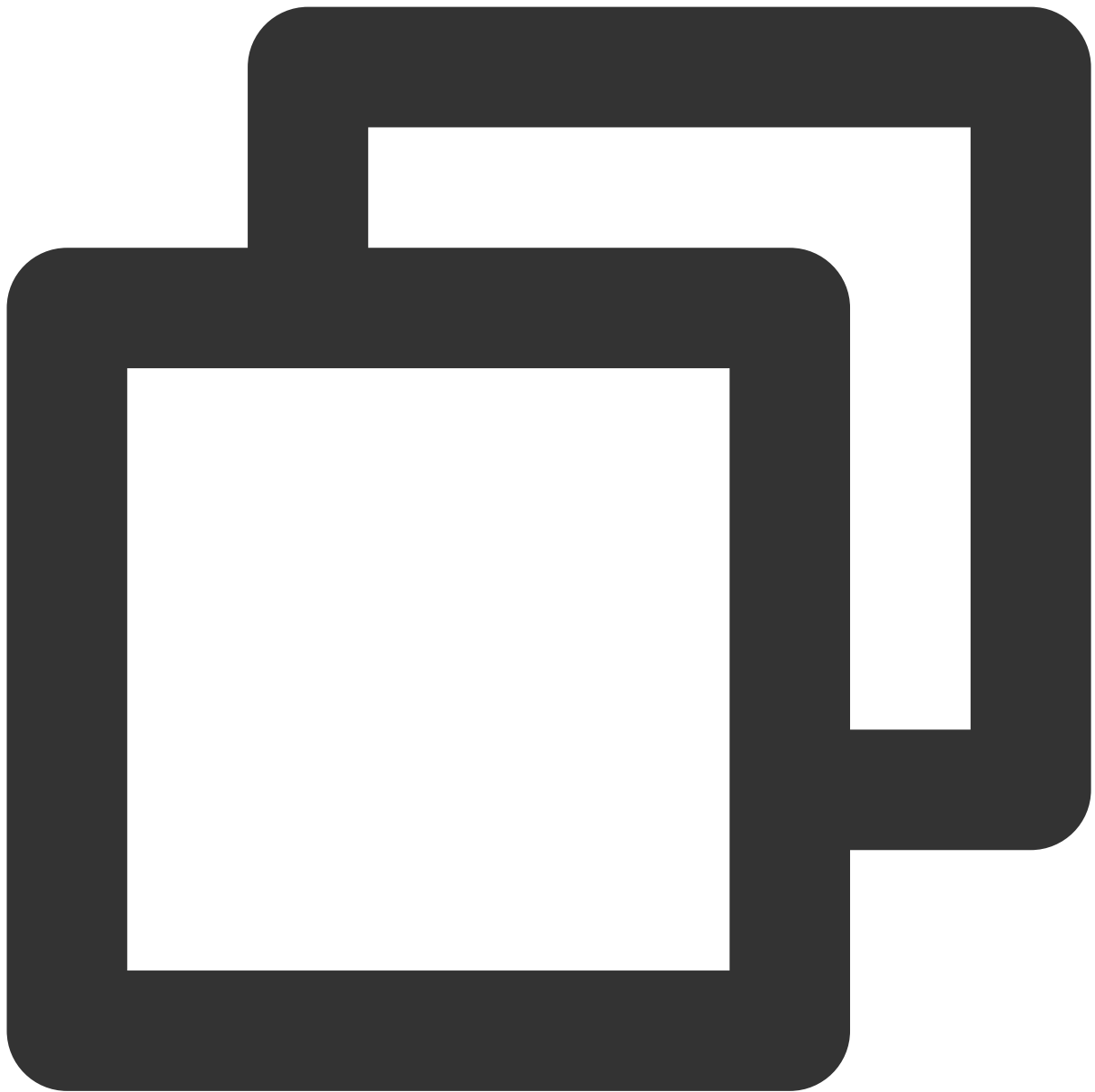


```
Future<void> selectAudioPlaybackDevice(TUIAudioPlaybackDevice device)
```

Parameter	Type	Description
device	<a href="#">TUIAudioPlaybackDevice</a>	The speaker or receiver.

## setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.



```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

### **enableMultiDeviceAbility**

This API is used to set whether to enable multi-device login for `TUICallEngine` (supported by the premium package).



```
Future<TUIResult> enableMultiDeviceAbility(bool enable)
```

### **setVideoRenderParams**

Set the rendering mode of video image.



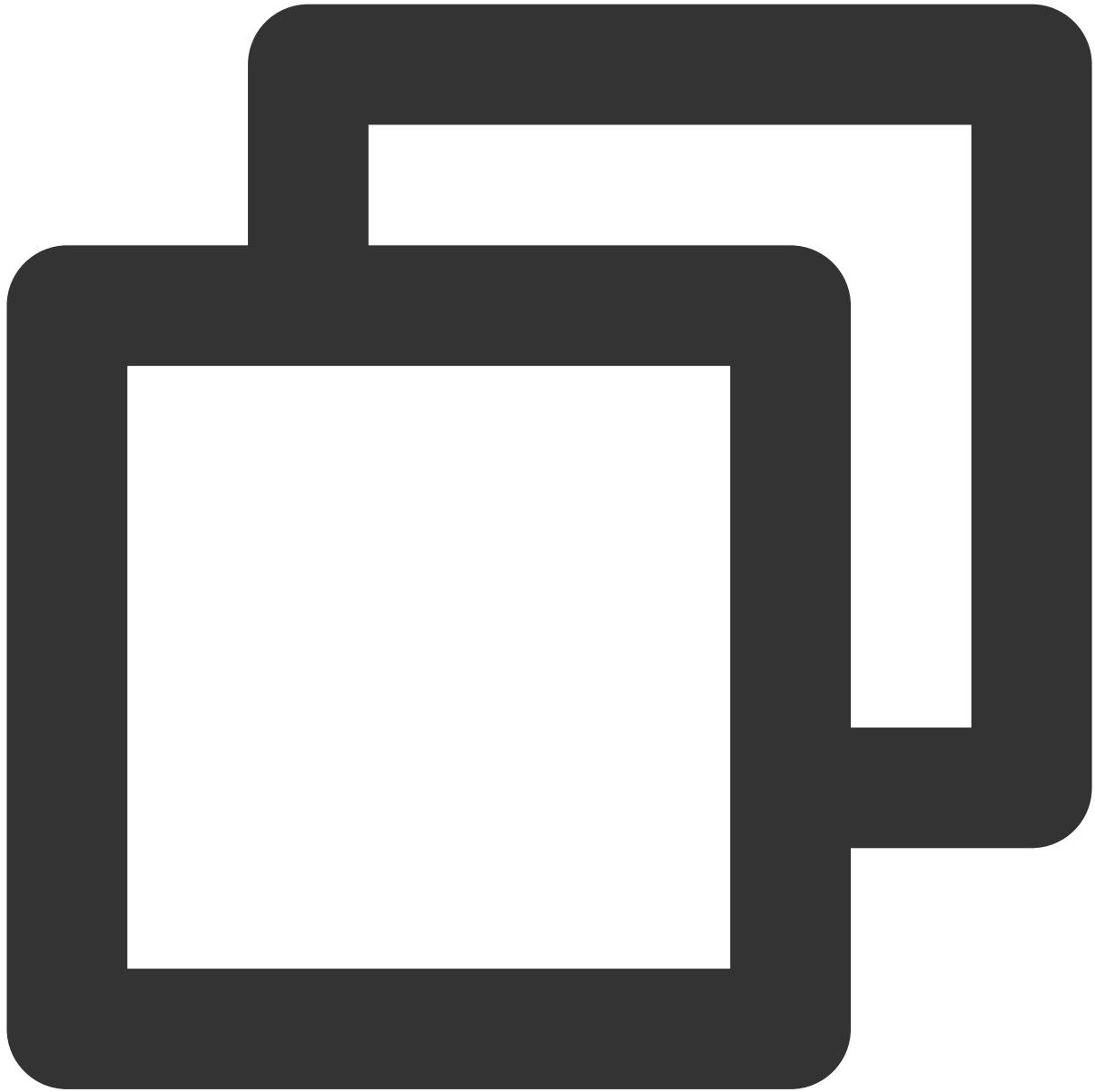
```
Future<TUIResult> setVideoRenderParams(String userId, VideoRenderParams params)
```

Parameter	Type	Description
userId	String	The target user ID.
params	<a href="#">VideoRenderParams</a>	Video render parameters.

## setVideoEncoderParams

Set the encoding parameters of video encoder.

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.



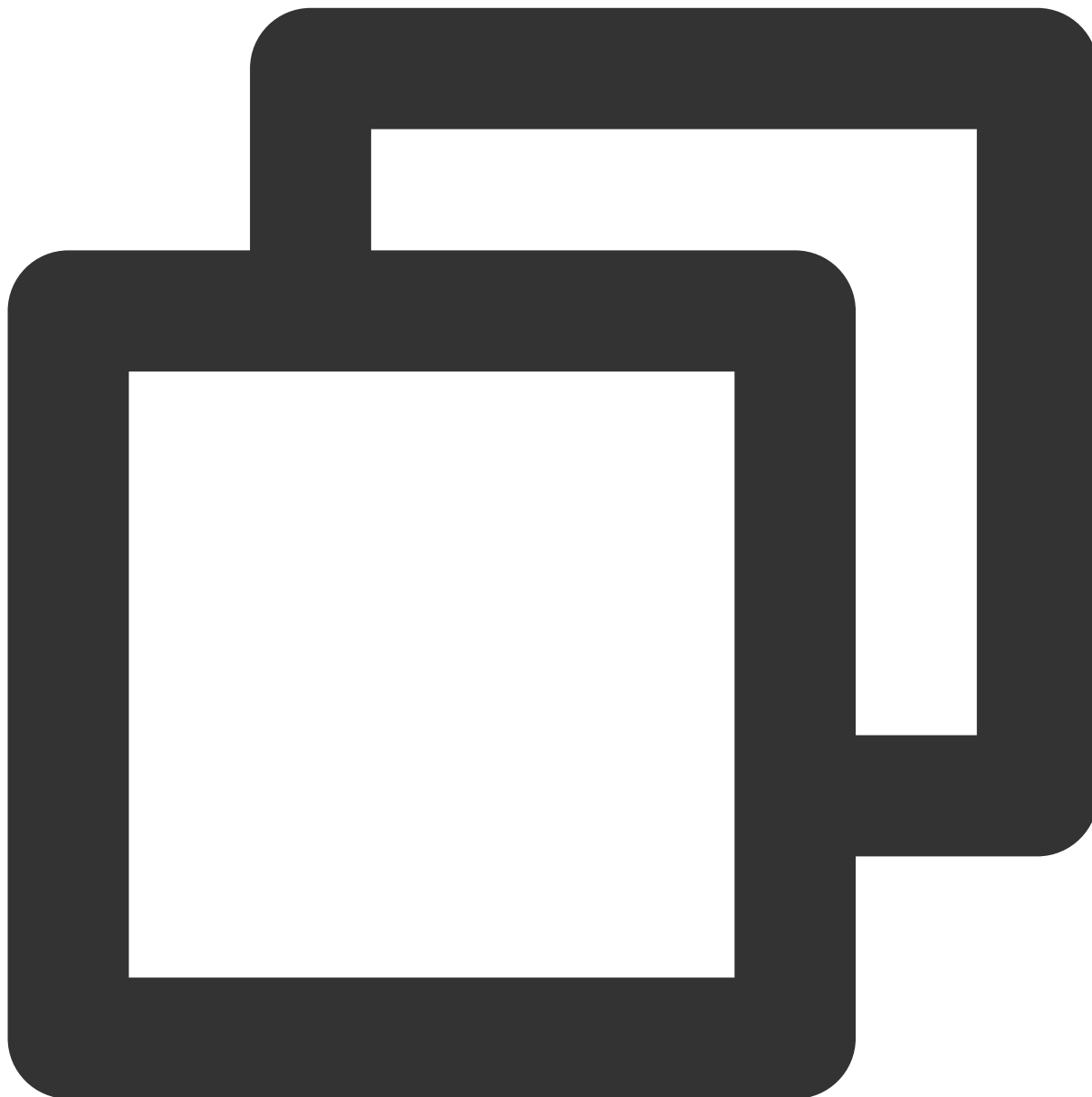
```
Future<TUIResult> setVideoEncoderParams (VideoEncoderParams params)
```

Parameter	Type	Description
params	<a href="#">VideoEncoderParams</a>	Video encoding parameters



## queryRecentCalls

Query call record.

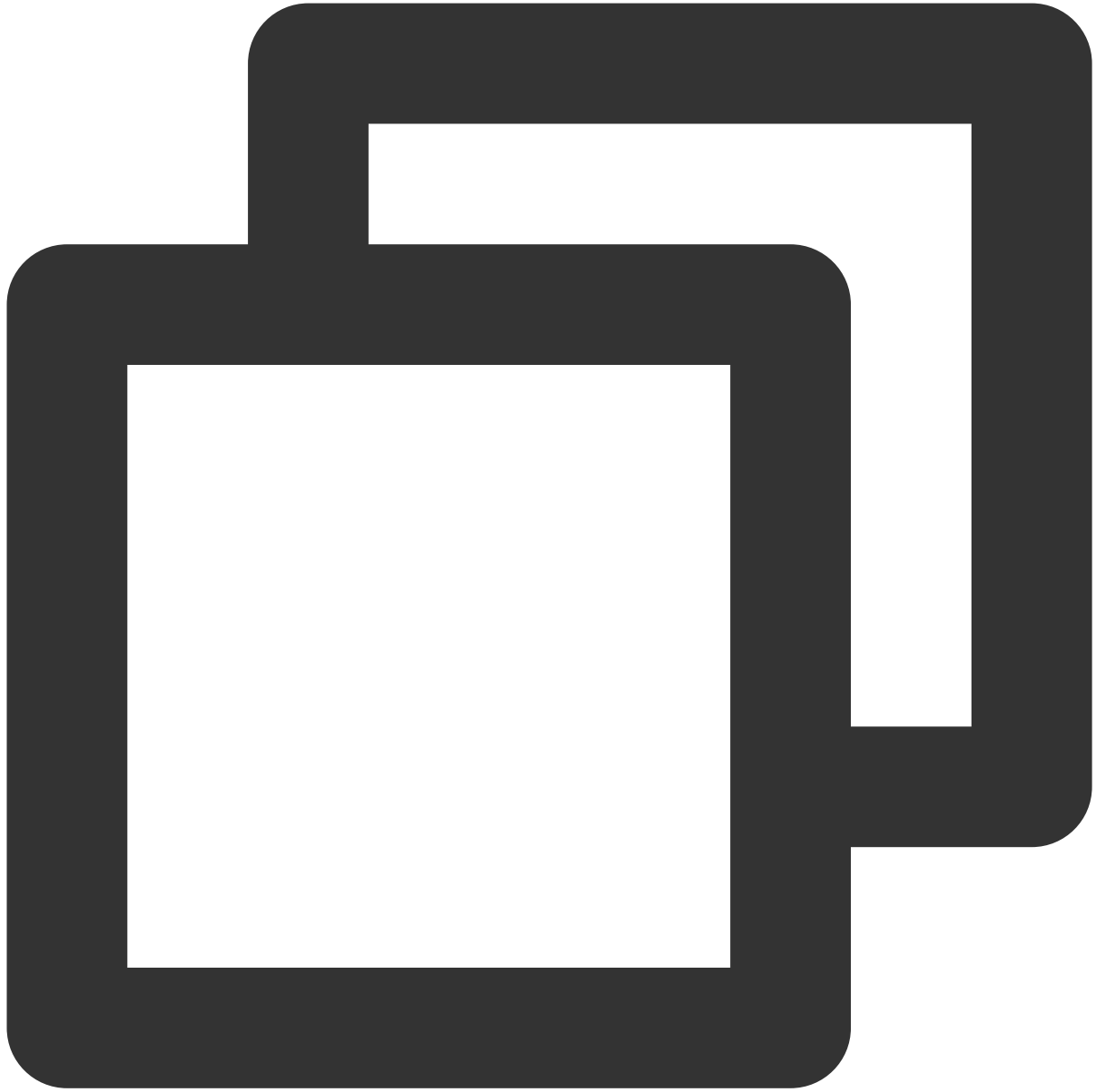


```
Future<void> queryRecentCalls(TUICallRecentCallsFilter filter, TUIValueCallback cal
```

Parameter	Type	Description
filter	<a href="#">TUICallRecentCallsFilter</a>	Filter condition

## deleteRecordCalls

Delete call record.



```
Future<void> deleteRecordCalls(List<String> callIdList, TUIValueCallback callback)
```

Parameter	Type	Description
callIdList	List<String>	List of IDs of records to be deleted.

### setBeautyLevel

Set beauty level, support turning off default beauty.



```
Future<TUIResult> setBeautyLevel(double level)
```

Parameter	Type	Description
level	double	Beauty level, range 0.0 to 9.0.

# TUICallObserver

Last updated : 2024-01-23 13:01:17

## TUICallObserver API

`TUICallObserver` is the callback class of `TUICallEngine`. You can use it to listen for events.

## Overview

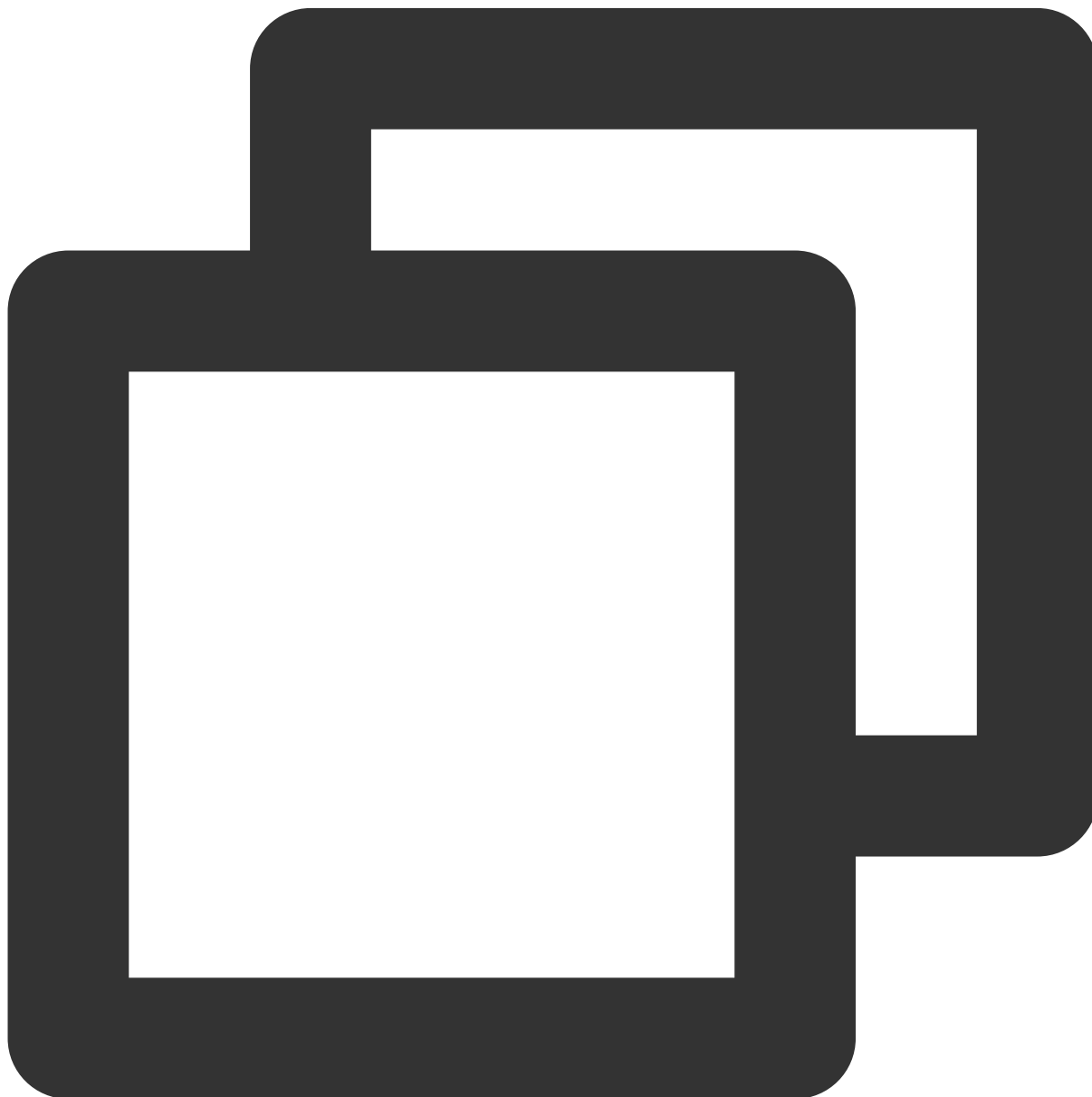
API	Description
<code>onError</code>	A call occurred during the call.
<code>onCallReceived</code>	A call invitation was received.
<code>onCallCancelled</code>	The call was canceled.
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user had a video stream.
<code>onUserAudioAvailable</code>	Whether a user had an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user was kicked offline.

`onUserSigExpired`

The user sig is expired.

## Details

Listen to the events thrown by the Flutter plugin through addObserver.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
  onError: (int code, String message) {
```

```
},onCallCancelled: (String callerId) {

}, onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole

}, onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole ca

}, onCallMediaTypeChanged: (TUICallMediaType oldCallMediaType, TUICallMediaType

}, onUserReject: (String userId) {

}, onUserNoResponse: (String userId) {

}, onUserLineBusy: (String onUserLineBusy) {

}, onUserJoin: (String userId) {

}, onUserLeave: (String userId) {

}, onUserVideoAvailable: (String userId, bool isVideoAvailable) {

}, onUserAudioAvailable: (String userId, bool isAudioAvailable) {

}, onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList

}, onCallReceived: (String callerId, List<String> calleeIdList, String groupId,

}, onUserVoiceVolumeChanged: (Map<String, int> volumeMap) {

}, onKickedOffline: () {

}, onUserSigExpired: () {

}

});
```

## onError

An error occurred.

### Note

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onError: (int code, String message) {  
    }  
));
```

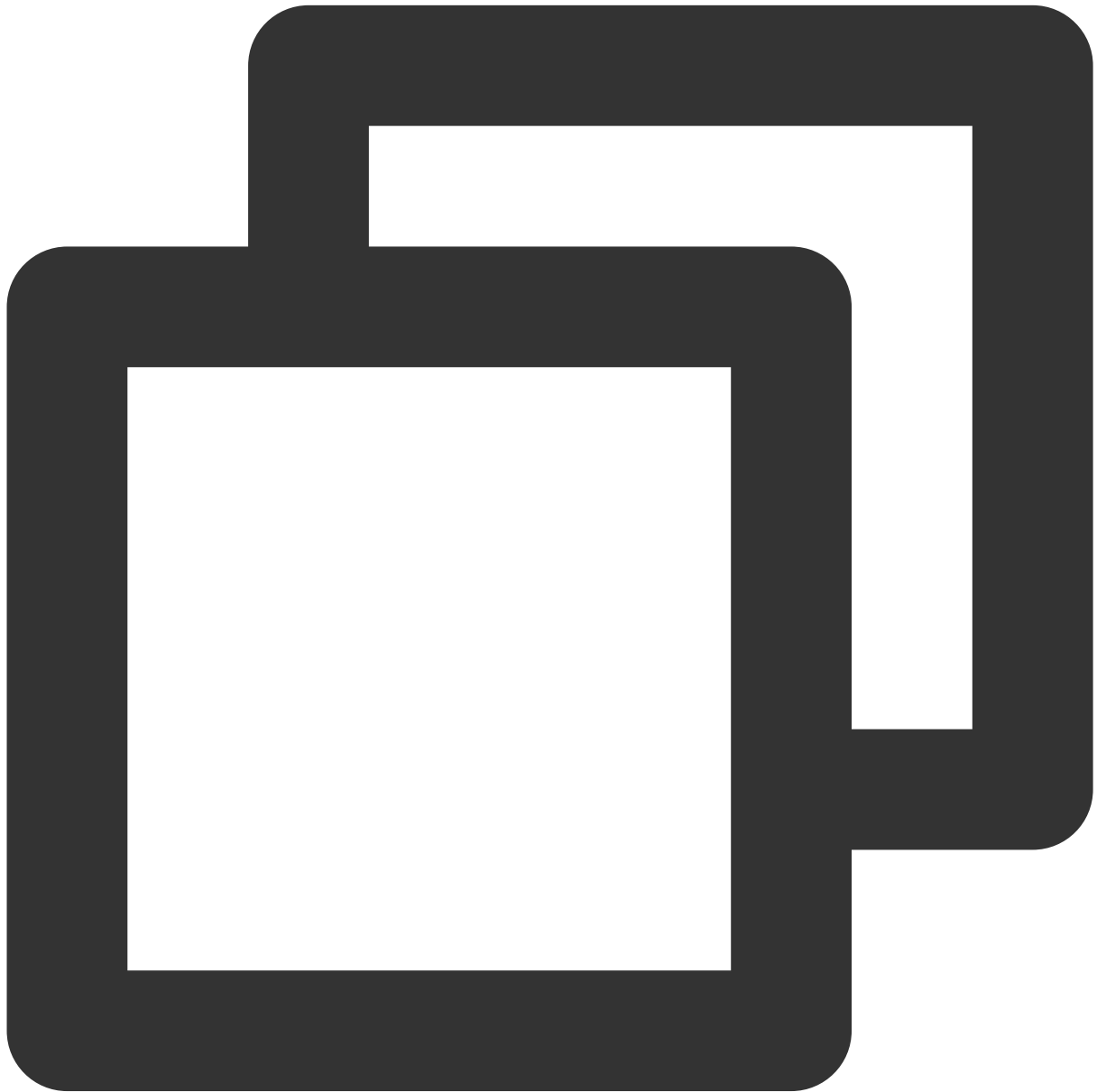
The parameters are described below:

Parameter	Type	Description
code	int	The error code.

message	String	The error message.
---------	--------	--------------------

## onCallReceived

A call invitation was received. This callback is received by an invitee. You can listen for this event to determine whether to display the incoming call view.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onCallReceived: (String callerId, List<String> calleeIdList, String groupId, TU  
    }  
));
```



The parameters are described below:

Parameter	Type	Description
callerId	String	The user ID of the inviter.
calleeIdList	List<String>	The invitee list.
groupId	String	The group ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.

## onCallCancelled

The call was canceled by the inviter or timed out. This callback is received by an invitee. You can listen for this event to determine whether to show a missed call message.

This indicates that the call was canceled by the caller, timed out by the callee, rejected by the callee, or the callee was busy. There are multiple scenarios involved. You can listen to this event to achieve UI logic such as missed calls and resetting UI status.

Call cancellation by the caller: The caller receives the callback (userId is himself); the callee receives the callback (userId is the ID of the caller) .

Callee timeout: the caller will simultaneously receive the [onUserNoResponse](#) and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID) .

Callee rejection: The caller will simultaneously receive the [onUserReject](#) and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID) .

Callee busy: The caller will simultaneously receive the [onUserLineBusy](#) and onCallCancelled callbacks (userId is his own ID);

Abnormal interruption: The callee failed to receive the call , he receives this callback (userId is his own ID).



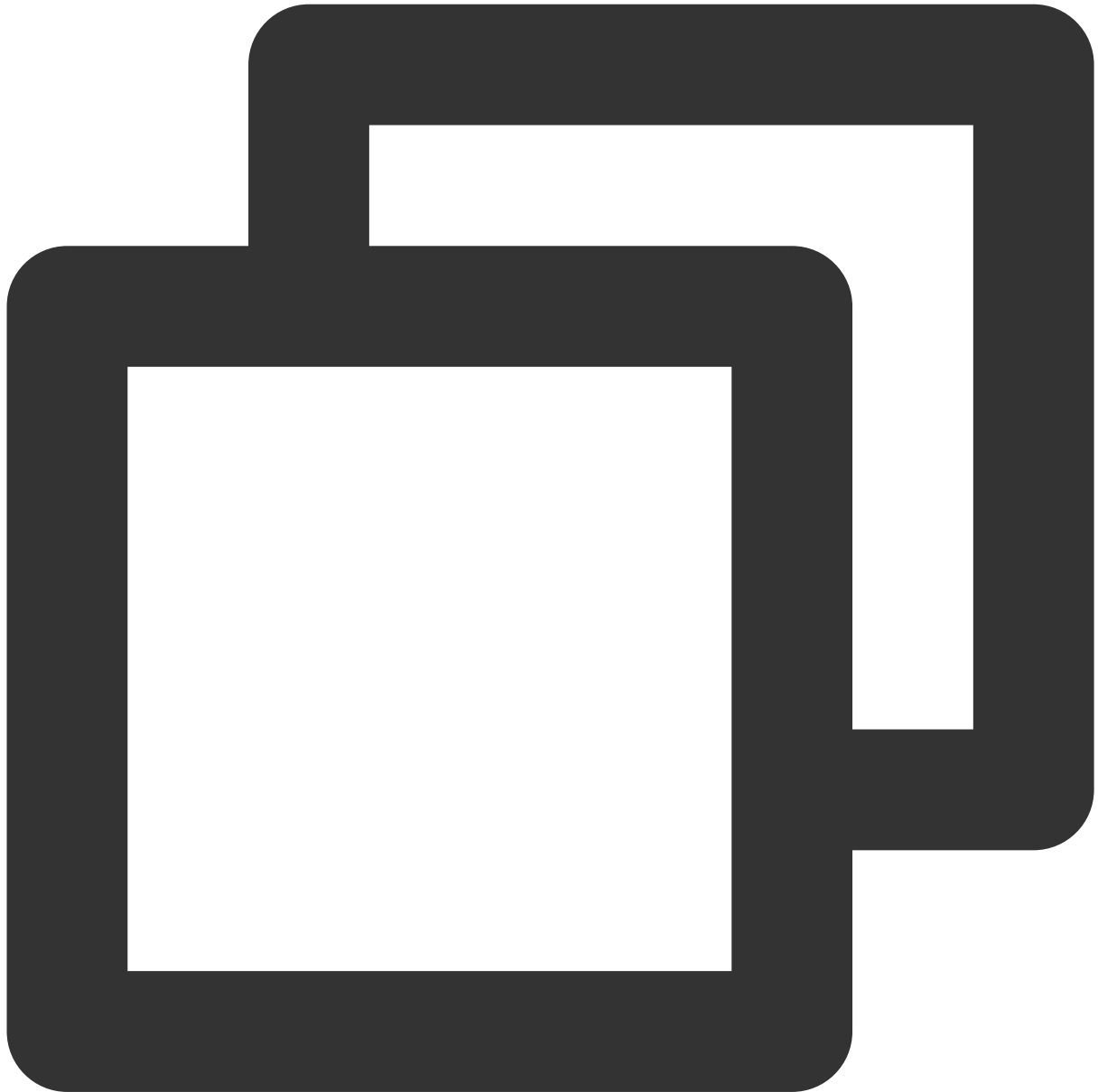
```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onCallCancelled: (String userId) {  
    }  
));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the inviter.

## onCallBegin

The call was connected. This callback is received by both the inviter and invitees. You can listen for this event to determine whether to start on-cloud recording, content moderation, or other tasks.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole cal  
})  
));
```

The parameters are described below:

--	--	--

Parameter	Type	Description
roomId	<a href="#">TUIRoomId</a>	The room ID.
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
callRole	<a href="#">TUICallRole</a>	The role, which can be caller or callee.

## onCallEnd

The call ended. This callback is received by both the inviter and invitees. You can listen for this event to determine when to display call information such as call duration and call type, or stop on-cloud recording.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole callRole)  
    )  
));
```

The parameters are described below:

Parameter	Type	Description
roomId	<a href="#">TUIRoomId</a>	The room ID.

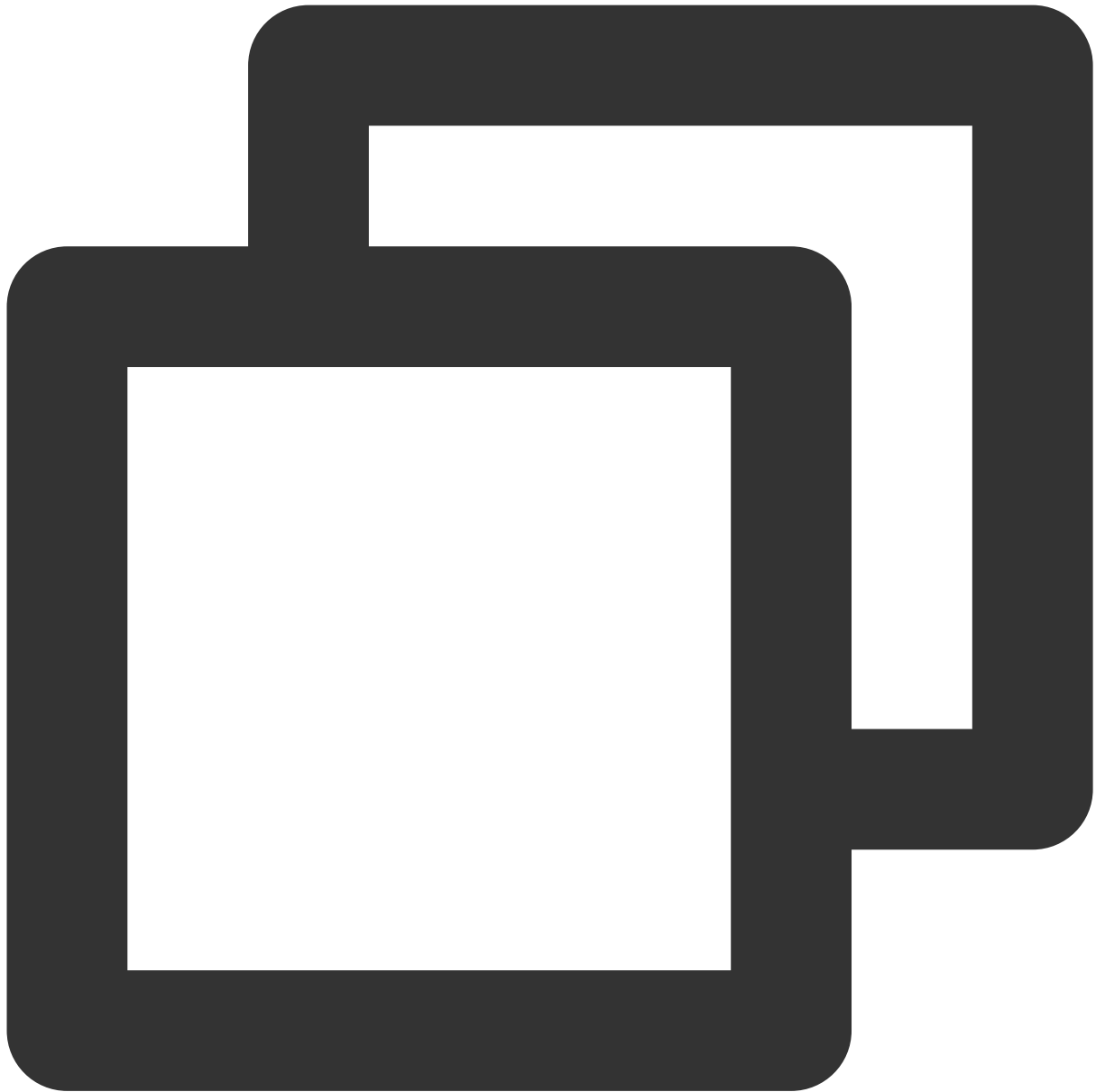
callMediaType	<a href="#">TUICallMediaType</a>	The call type, which can be video or audio.
callRole	Number	The role, which can be caller or callee.
totalTime	double	The call duration.

### Note

Client-side callbacks are often lost when errors occur, for example, when the process is closed. If you need to measure the duration of a call for billing or other purposes, we recommend you use the RESTful API.

### onCallMediaTypeChanged

The call type changed.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onCallMediaTypeChanged: (TUICallMediaType oldCallMediaType, TUICallMediaType ne  
    }  
));
```

The parameters are described below:

Parameter	Type	Description
oldCallMediaType	<a href="#">TUICallMediaType</a>	The call type before the change.

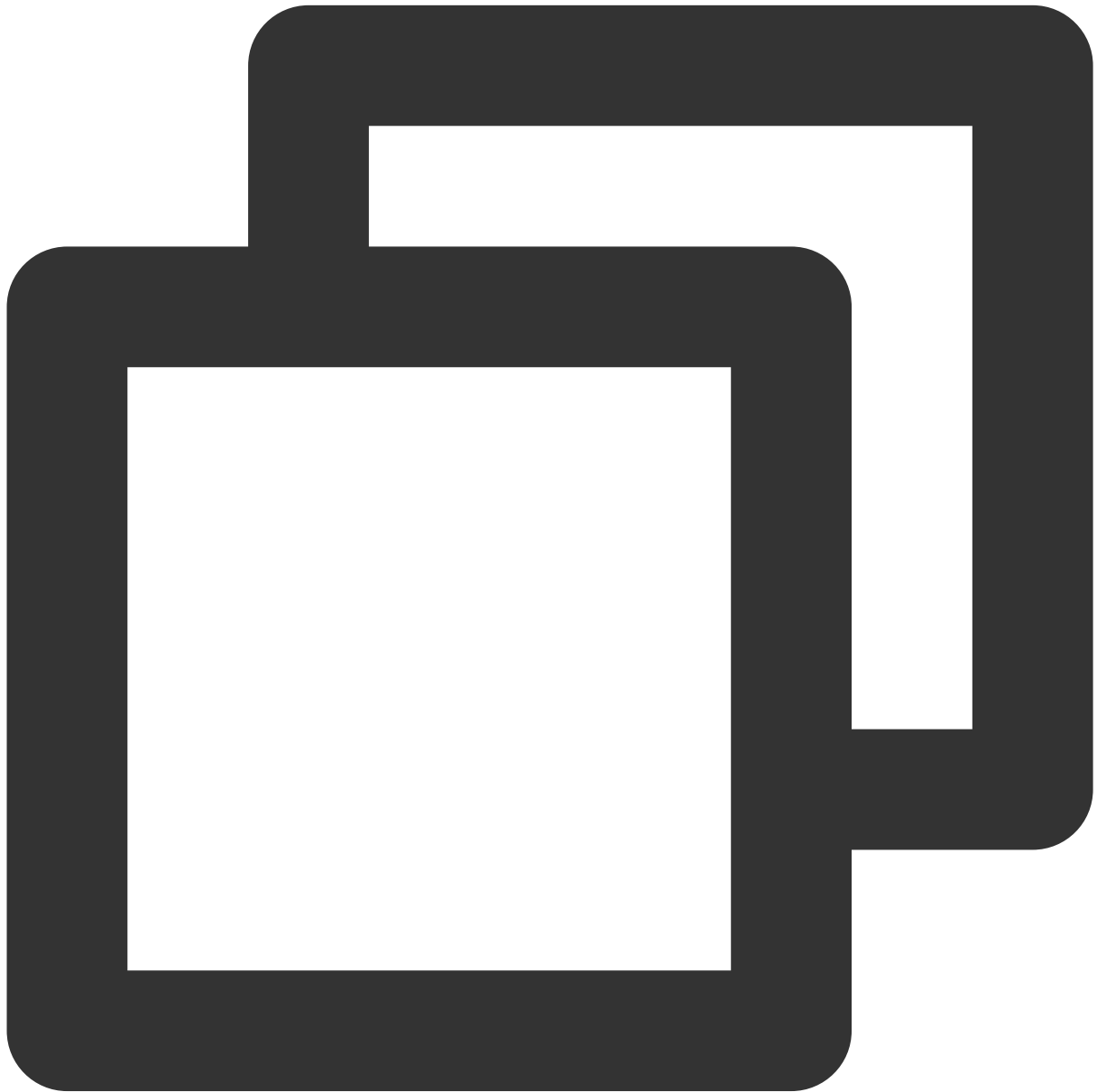
newCallMediaType

[TUICallMediaType](#)

The call type after the change.

## onUserReject

The call was rejected. In a one-to-one call, only the inviter will receive this callback. In a group call, all invitees will receive this callback.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onUserReject: (String userId) {  
    }  
));
```

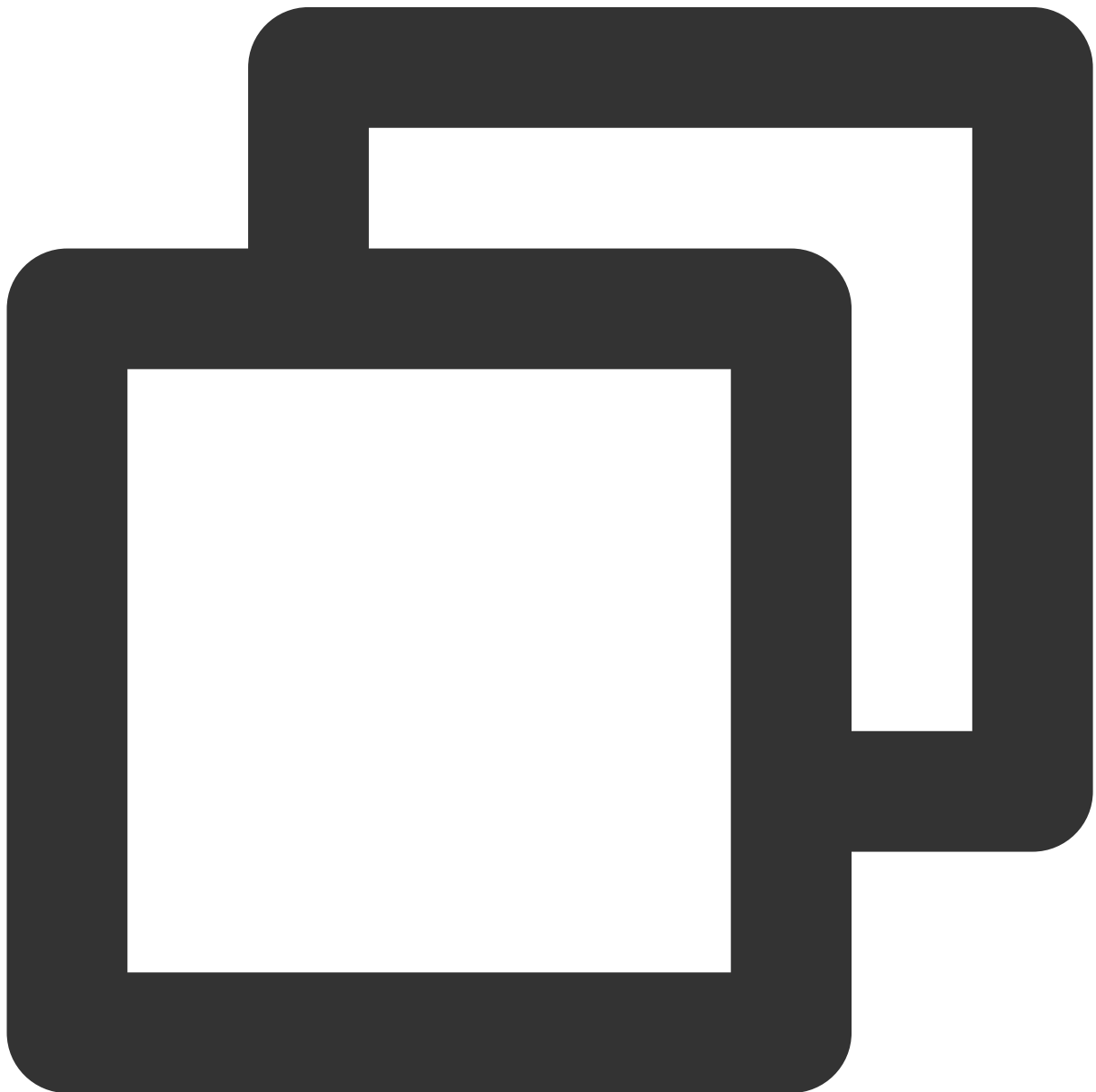


The parameters are described below:

Parameter	Type	Description
res.userId	String	The user ID of the invitee who rejected the call.

## onUserNoResponse

A user did not respond.



```
TUICallEngine.instance.addObserver(TUICallObserver(
```

```
    onUserNoResponse: (String userId) {  
    }  
});
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who did not answer.

## onUserLineBusy

A user is busy.



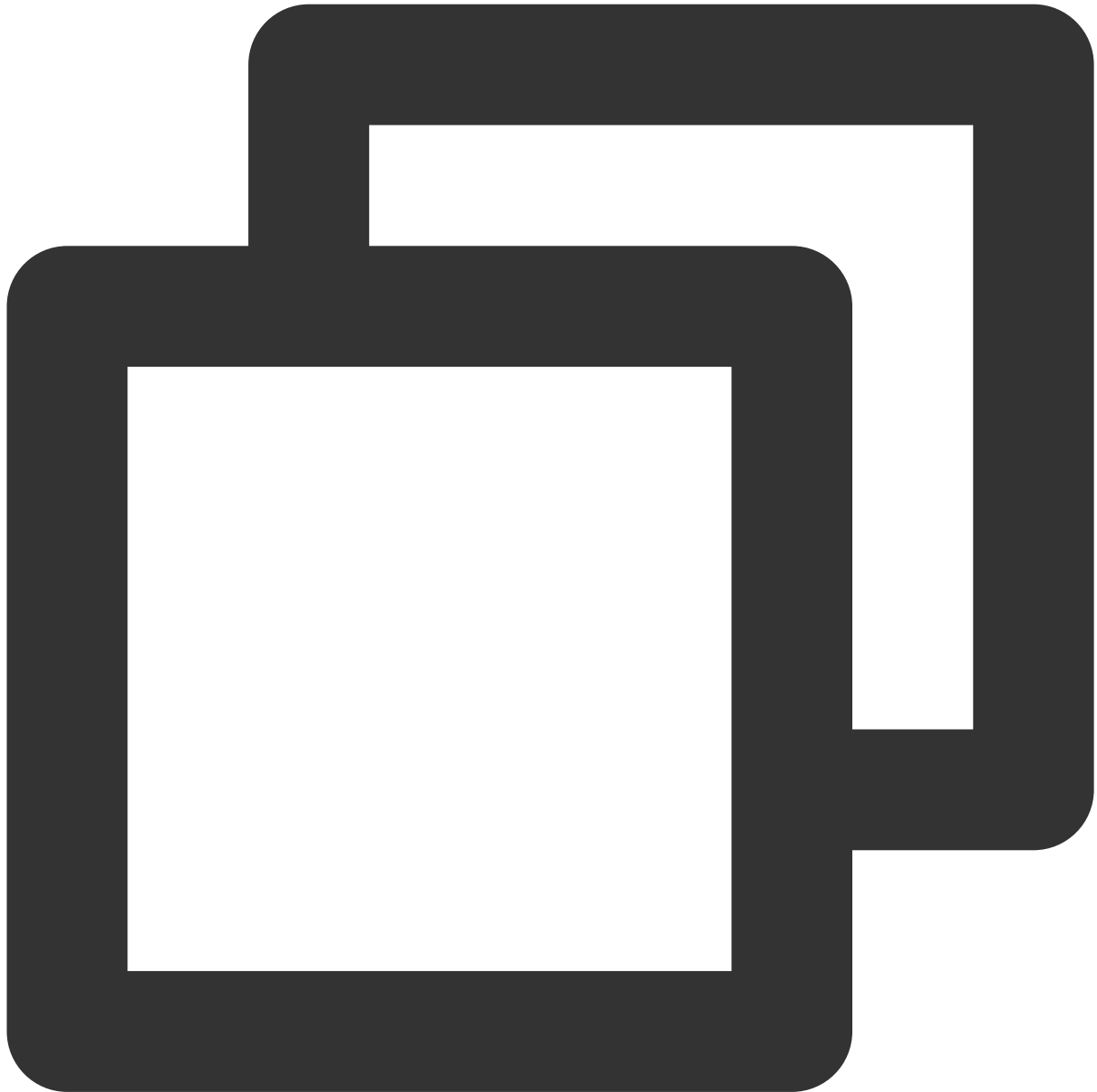
```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onUserLineBusy: (String onUserLineBusy) {  
        },  
));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who is busy.

## onUserJoin

A user joined the call.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onUserJoin: (String userId) {  
    }  
));
```

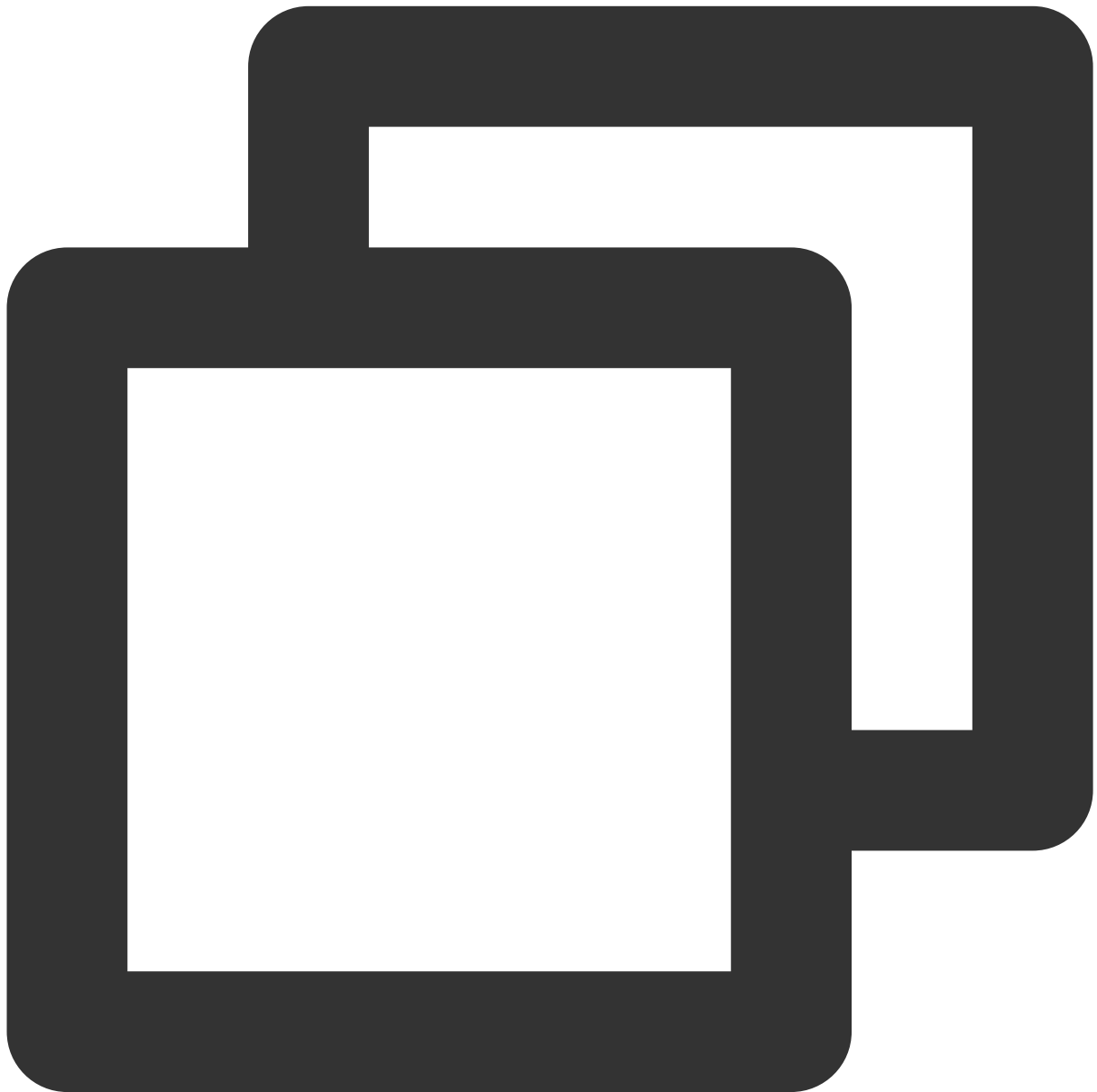
The parameters are described below:

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

userId	String	The ID of the user who joined the call.
--------	--------	---

## onUserLeave

A user left the call.



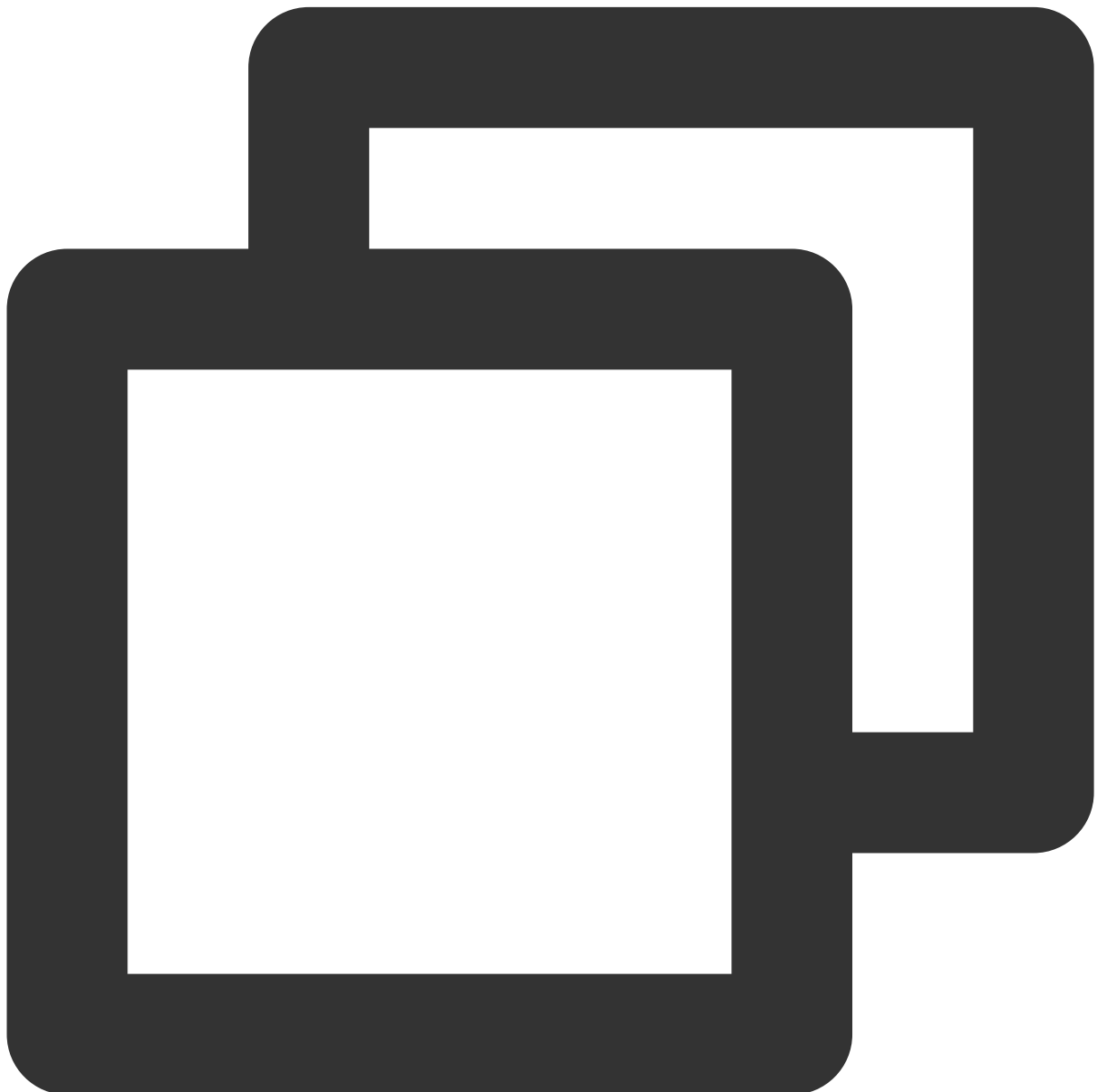
```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onUserLeave: (String userId) {  
    }  
));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The ID of the user who left the call.

### **onUserVideoAvailable**

Whether a user is sending video.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onUserVideoAvailable: (String userId, bool isVideoAvailable) {
```

```
}  
));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
isVideoAvailable	bool	Whether the user has video.

### **onUserAudioAvailable**

Whether a user is sending audio.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onUserAudioAvailable: (String userId, bool isAudioAvailable) {  
    }  
));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.



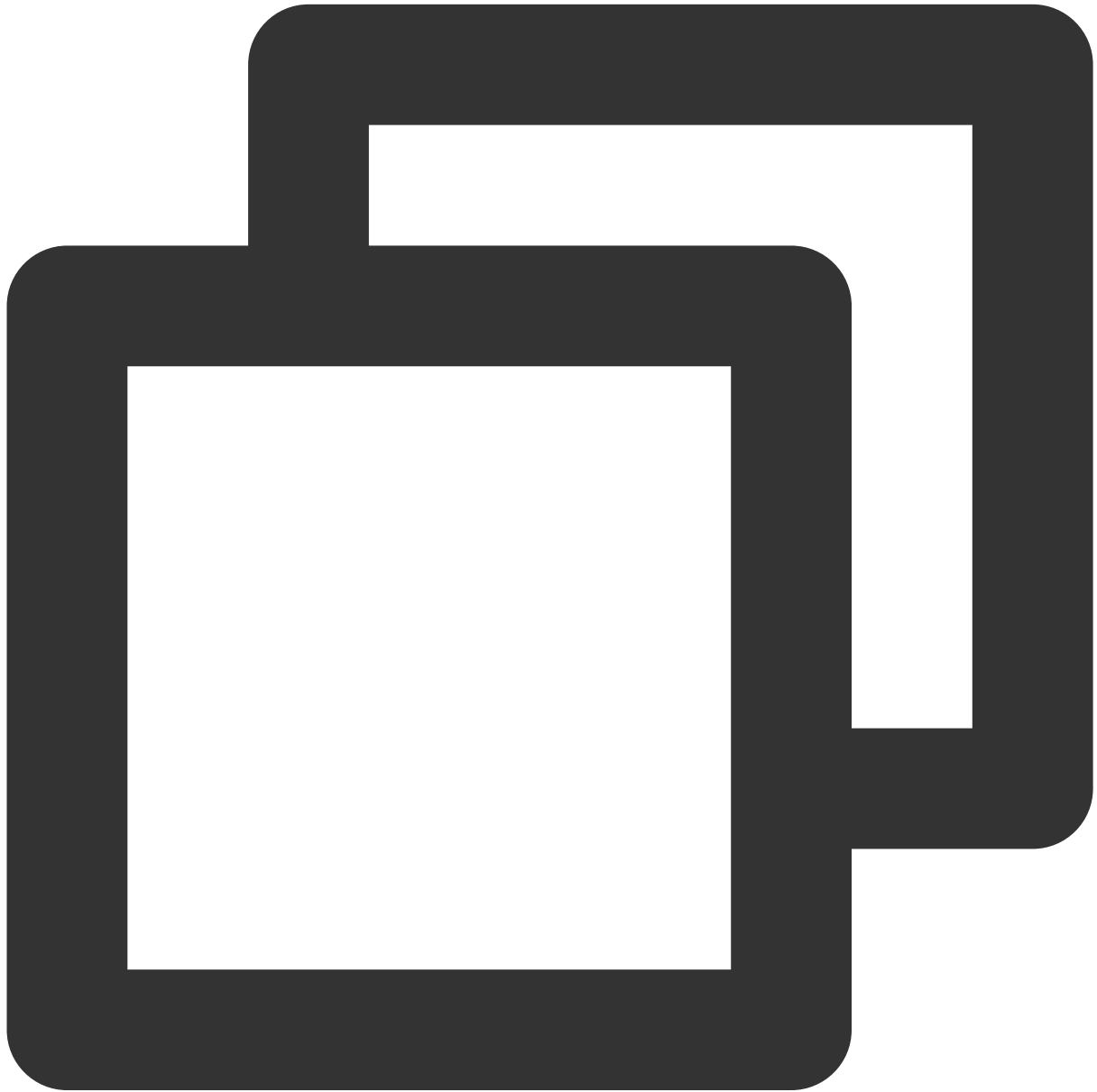
isAudioAvailable

bool

Whether the user has audio.

**onUserVoiceVolumeChanged**

The volumes of all users.



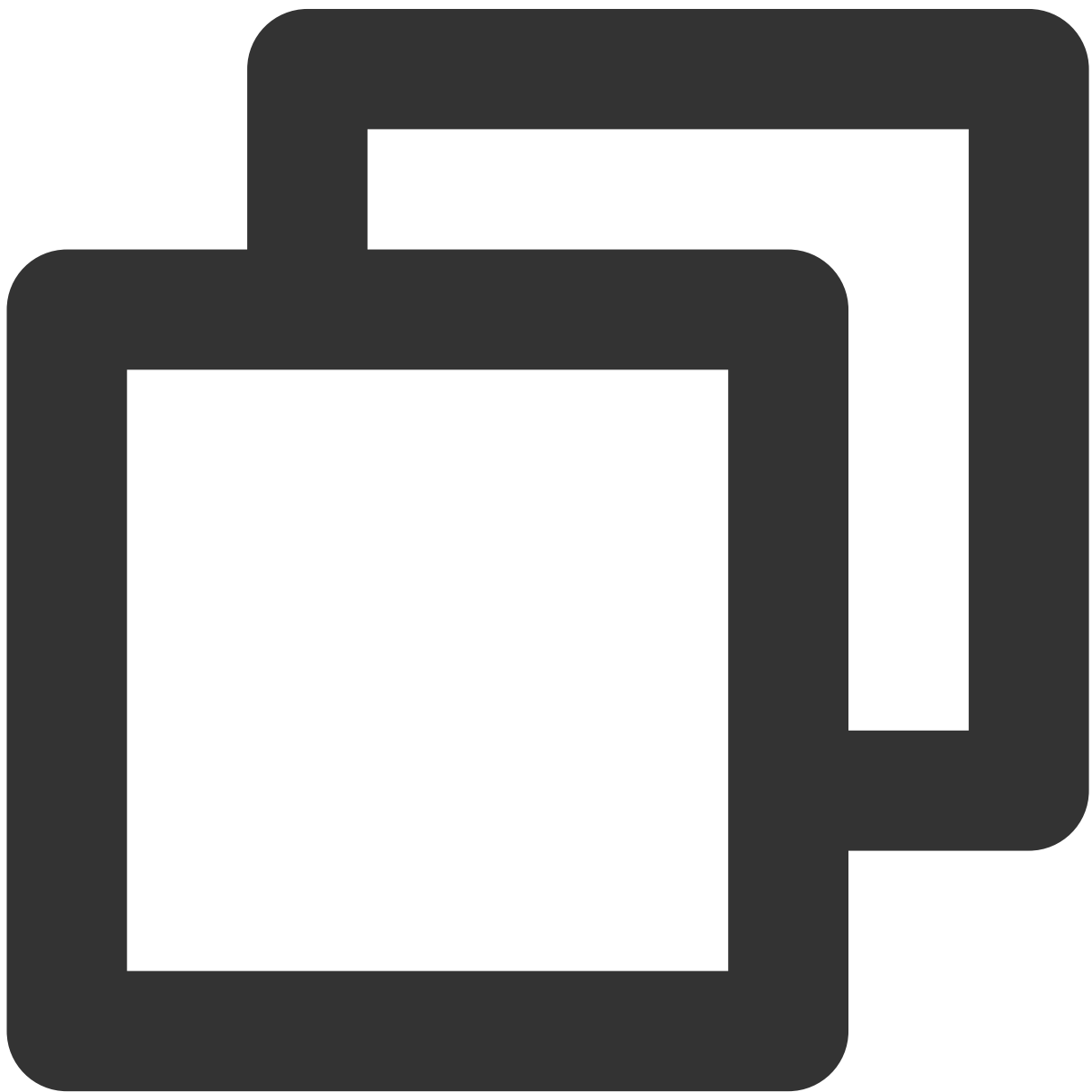
```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onUserVoiceVolumeChanged: (Map<String, int> volumeMap) {  
    }  
));
```

The parameters are described below:

Parameter	Type	Description
volumeMap	Map<String, int>	The volume table, which includes the volume of each user (userId). Value range: 0-100.

### onUserNetworkQualityChanged

The network quality of all users.



```
TUICallEngine.instance.addObserver(TUICallObserver (
```

```
onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList) {  
    }  
});  
  
class TUINetworkQualityInfo {  
    String userId;  
    TUINetworkQuality quality;  
    TUINetworkQualityInfo({required this.userId, required this.quality});  
}  
  
enum TUINetworkQuality {  
    unknown,  
    excellent,  
    good,  
    poor,  
    bad,  
    vBad,  
    down  
}
```

The parameters are described below:

Parameter	Type	Description
networkQualityList	List< <a href="#">TUINetworkQualityInfo</a> >	The current network conditions for all users (userId).

## onKickedOffline

The current user was kicked offline : At this time, you can prompt the user with a UI message and then invoke `init` again.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onKickedOffline: () {  
    }  
));
```

### onUserSigExpired

The user sig is expired : At this time, you need to generate a new `userSig` , and then invoke `init` again.



```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onUserSigExpired: () {  
    }  
));
```

# Type Definition

Last updated : 2023-09-21 15:37:01

## Common structures

### TUIResult

The return value of calling the API

Value	Type	Description
code	String	If the code is empty "", it means the call succeeded, if the code is not empty "", it means the call failed.
message	String?	Error message

### TUIRoomId

Room ID for audio and video in a call.

#### Note :

(1) `intRoomId` and `strRoomId` are mutually exclusive. If you choose to use `strRoomId`, `intRoomId` needs to be filled in as 0. If both are filled in, the SDK will prioritize `intRoomId`.

(2) Do not mix `intRoomId` and `strRoomId` because they are not interchangeable. For example, the number 123 and the string "123" represent two completely different rooms.

Value	Type	Description
intRoomId	int	Numeric room ID.
strRoomId	String	String room number.

### VideoRenderParams

Video render parameters

Value	Type	Description
fillMode	<a href="#">FillMode</a>	Video image fill mode
rotation	<a href="#">Rotation</a>	Video image rotation direction

## VideoEncoderParams

Video encoding parameters

Value	Type	Description
resolution	<a href="#">Resolution</a>	Video resolution
resolutionMode	<a href="#">ResolutionMode</a>	Video aspect ratio mode

## TUICallParams

Call params

Value	Type	Description
roomId	<a href="#">TUIRoomId</a>	Room Id.
offlinePushInfo	<a href="#">TUIOfflinePushInfo</a>	Offline push vendor configuration information.
timeout	String	Call timeout period, default: 30s, unit: seconds.
userData	String	An additional parameter.

## TUIOfflinePushInfo

Offline push vendor configuration information, please refer to:[Offline call push](#)

Value	Type	Description
title	String	offlinepush notification title
desc	String	offlinepush notification description
ignoreIOSBadge	bool	Ignore badge count for offline push (only for iOS), if set to true, the message will not increase the unread count of the app icon on the iOS receiver's side.
iOSSound	String	Offline push sound setting (only for iOS). When sound = IOS_OFFLINE_PUSH_NO_SOUND , there will be no sound played when the message is received. When sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND , the system sound will be played when the message is received. If you want to customize the iOSSound, you need to link the audio file into

		the Xcode project first, and then set the audio file name (with extension) to the iOSSound.
androidSound	String	Offline push sound setting (only for Android, supported by IMSDK 6.1 and above). Only Huawei and Google phones support setting sound prompts. For Xiaomi phones, please refer to: Xiaomi custom ringtones . In addition, for Google phones, in order to set sound prompts for FCM push on Android 8.0 and above systems, you must call setAndroidFCMChannelID to set the channelId for it to take effect.
androidOPPOChannelID	String	Set the channel ID for OPPO phones with Android 8.0 and above systems.
androidVIVOClassification	int	Classification of VIVO push messages (deprecated interface, VIVO push service will optimize message classification rules on April 3, 2023. It is recommended to use setAndroidVIVOCategory to set the message category). 0: Operational messages, 1: System messages. The default value is 1.
androidXiaoMiChannelID	String	Set the channel ID for Xiaomi phones with Android 8.0 and above systems.
androidFCMChannelID	String	Set the channel ID for google phones with Android 8.0 and above systems.
androidHuaWeiCategory	String	Classification of Huawei push messages, please refer to: <a href="#">Huawei message classification standard</a> .
isDisablePush	bool	Whether to turn off push notifications (default is on).
iOSPushType	<a href="#">TUICallIOSOfflinePushType</a>	iOS offline push type, default is APNs

## TUICallRecords

Call recording information

Value	Type	Description



callId	String	Call recording ID.
inviter	String	Inviter ID.
inviteList	List<String>	List of invited user IDs.
scene	<a href="#">TUICallScene</a>	Call scenario.
mediaType	<a href="#">TUICallMediaType</a>	Media type.
groupId	String	Group ID.
role	<a href="#">TUICallRole</a>	Role.
result	<a href="#">TUICallResultType</a>	Call result type.
beginTime	int	Start time.
totalTime	int	Total time.

## TUICallRecentCallsFilter

Call recording filtering conditions.

Value	Type	Description
begin	double	Start time.
end	double	End time.
resultType	<a href="#">TUICallResultType</a>	Call result type.

## enum definition

### TUICallMediaType

Call media type

Type	Value	Description
none	0	Unknown
audio	1	Audio call
video	2	Video call

### TUICallRole

## Call role

Type	Value	Description
none	0	Unknown
caller	1	Caller (inviter)
called	2	Callee (invitee)

**TUICallStatus**

## Call status

Type	Value	Description
none	0	Unknown
waiting	1	The call is currently waiting
accept	2	The call has been accepted

**TUICallScene**

## Call scene

Type	Value	Description
groupCall	0	Group call
multiCall	1	Anonymous group calling (not supported at this moment, please stay tuned).
singleCall	2	one to one call

**TUINetworkQuality**

## Network quality

Type	Value	Description
unknown	0	Unknown
excellent	1	Excellent
good	2	Good
poor	3	Poor

bad	4	Bad
vBad	5	Vbad
down	6	Down

## FillMode

If the aspect ratio of the video display area is not equal to that of the video image, you need to specify the fill mode:

Type	Value	Description
fill	0	Fill mode: the video image will be centered and scaled to fill the entire display area, where parts that exceed the area will be cropped. The displayed image may be incomplete in this mode.
fit	1	Fit mode: the video image will be scaled based on its long side to fit the display area, where the short side will be filled with black bars. The displayed image is complete in this mode, but there may be black bars.

## Rotation

We provides rotation angle setting APIs for local and remote images. The following rotation angles are all clockwise.

Type	Value	Description
rotation_0	0	No rotation
rotation_90	1	Clockwise rotation by 90 degrees
rotation_180	2	Clockwise rotation by 180 degrees
rotation_270	3	Clockwise rotation by 270 degrees

## ResolutionMode

Video aspect ratio mode

Type	Value	Description
landscape	0	Landscape resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360.
portrait	1	Portrait resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640.

## Resolution

Video resolution

Type	Value	Description
resolution_640_360	0	Aspect ratio: 16:9 ; resolution: 640x360 ; recommended bitrate: 500kbps
resolution_640_480	1	Aspect ratio: 4:3 ; resolution: 640x480 ; recommended bitrate: 600kbps
resolution_960_540	2	Aspect ratio: 16:9 ; resolution: 960x540 ; recommended bitrate: 850kbps
resolution_960_720	3	Aspect ratio: 4:3 ; resolution: 960x720 ; recommended bitrate: 1000kbps
resolution_1280_720	4	Aspect ratio: 16:9 ; resolution: 1280x720 ; recommended bitrate: 1200kbps
resolution_1920_1080	5	Aspect ratio: 16:9 ; resolution: 1920x1080 ; recommended bitrate: 2000kbps

## TUICallIOSOfflinePushType

iOS offline push type

Type	Value	Description
APNs	0	APNs
VoIP	1	VoIP

## TUICamera

Camera type.

Type	Value	Description
front	0	Front camera.
back	1	Rear camera.

## TUIAudioPlaybackDevice

Audio playback device.

Type	Value	Description
------	-------	-------------

speakerphone	0	Speaker
earpiece	1	Earpiece

## TUICallResultType

Call recording type.

Type	Value	Description
unknown	0	Unknown
missed	1	Missed
incoming	2	Incoming call
outgoing	3	Outgoing Call

# uniapp (Android&iOS)

## API Overview

Last updated : 2024-03-07 10:34:35

### TUICallKit (UI Included)

TUICallKit is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat.

API	Description
<a href="#">login</a>	login
<a href="#">logout</a>	logou
<a href="#">setSelfInfo</a>	Sets the user nickname and profile photo.
<a href="#">call</a>	Makes a one-to-one call.
<a href="#">groupCall</a>	Makes a group call.
<a href="#">joinInGroupCall</a>	Joins a group call.
<a href="#">setCallingBell</a>	Sets the ringtone.
<a href="#">enableMuteMode</a>	Sets whether to turn on the mute mode.
<a href="#">enableFloatWindow</a>	Sets whether to enable floating windows.

### Event

TUICallKit throws the following events.

API	Description
<a href="#">onError</a>	An error occurred during the call.
<a href="#">onCallReceived</a>	A call was received.
<a href="#">onCallCancelled</a>	The call was canceled.
<a href="#">onCallBegin</a>	The call was connected.

<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user has a video stream.
<code>onUserAudioAvailable</code>	Whether a user has an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user was kicked offline.
<code>onUserSigExpired</code>	The user sig is expired.

## TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description
<code>hangup</code>	Ends a call.
<code>accept</code>	Answers a call.
<code>setVideoRenderParams</code>	Set the rendering mode of video image.
<code>setVideoEncoderParams</code>	Set the encoding parameters of video encoder.

# TUICallKit

Last updated : 2024-03-07 18:03:37

## TUICallKit APIs

`TUICallKit` is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat. For directions on integration, see [Integrating TUICallKit](#).

## API Overview

API	Description
<a href="#">login</a>	login
<a href="#">logout</a>	logout
<a href="#">setSelfInfo</a>	Sets the user nickname and profile photo.
<a href="#">call</a>	Makes a one-to-one call.
<a href="#">groupCall</a>	Makes a group call.
<a href="#">joinInGroupCall</a>	Joins a group call.
<a href="#">setCallingBell</a>	Sets the ringtone.
<a href="#">enableMuteMode</a>	Sets whether to turn on the mute mode.
<a href="#">enableFloatWindow</a>	Sets whether to enable floating windows.

## API Detail

### login





```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  SDKAppID: 0,
  userID: 'mike',
  userSig: '',
};
TUICallKit.login(options, (res) => {
  if (res.code === 0) {
    console.log('login success');
  } else {
    console.log(`login failed, error message = ${res.msg}`);
  }
});
```

```
}  
});
```

Parameter	Type	Description
options	Object	Initialization parameters
options.SDKAppID	Number	User SDKAppID
options.userID	String	userID
options.userSig	String	User Signature
callback	Function	callback function, code = 0 means the call was successful; code != 0 means the call failed, see msg for the reason.

## logout



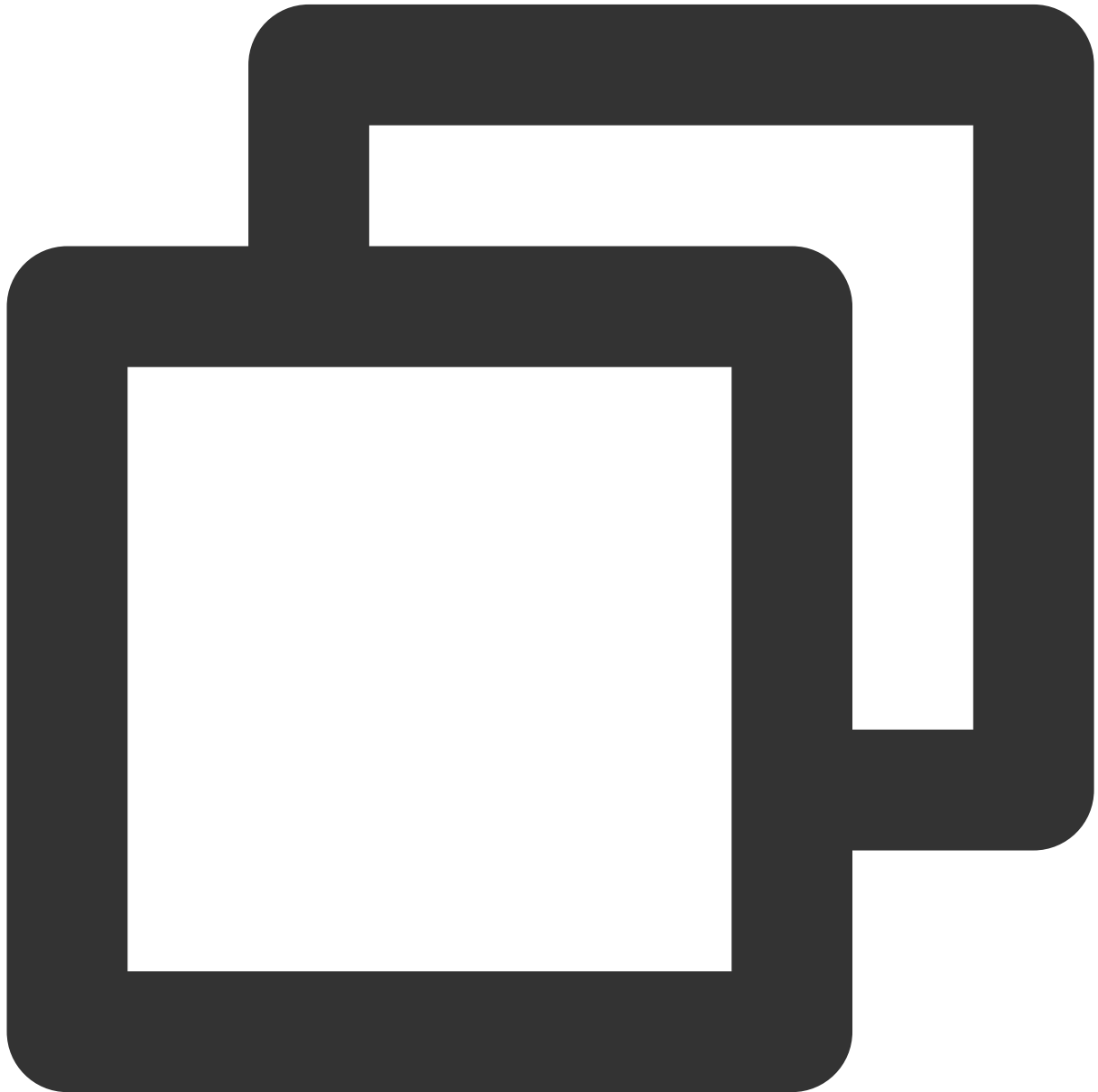
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
TUICallKit.logout((res) => {
  if (res.code === 0) {
    console.log('logout success');
  } else {
    console.log(`logout failed, error message = ${res.msg}`);
  }
});
```

Parameter	Type	Description

callback	Function	callback function, code = 0 means the call was successful; code != 0 means the call failed, see msg for the reason.
----------	----------	---

## setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  nickName: 'jack',
```

```
avatar: 'https://****/user_avatar.png'
}
TUICallKit.setSelfInfo(options, (res) => {
  if (res.code === 0) {
    console.log('setSelfInfo success');
  } else {
    console.log(`setSelfInfo failed, error message = ${res.msg}`);
  }
});
```

Parameter	Type	Description
options	Object	Initialization parameters
options.nickName	String	Nickname of the target user, not required
options.avatar	String	Target user's avatar, not required
callback	Function	callback function, code = 0 means the call was successful; code != 0 means the call failed, see msg for the reason.

## call

This API is used to make a (one-to-one) call.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  userID: 'mike',
  callMediaType: 1, // audio call(callMediaType = 1)、video call(callMediaType = 2)
  roomID: 0,
  strRoomID: '1223',
};
TUICallKit.call(options, (res) => {
  if (res.code === 0) {
    console.log('call success');
  } else {
```

```
    console.log(`call failed, error message = ${res.msg}`);  
  }  
});
```

The parameters are described below:

Parameter	Type	Description
options	Object	Initialization parameters
options.userID	String	The userID of the target user
options.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
options.roomID	Number	Customize the numeric room number. As long as the roomID is present, the numeric room number is used, even if strRoomID is present.
options.strRoomID	String	Customize the string room number. If you want to use a string room number, you need to set roomID = 0 after setting strRoomID.
callback	Function	callback function, code = 0 means the call was successful; code != 0 means the call failed, see msg for the reason.

## groupCall

This API is used to make a group call.

### Note:

Before making a group call, you need to create an IM group first.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  groupID: 'myGroup',
  userIDList: ['mike', 'tom'],
  callMediaType: 1, // audio call(callMediaType = 1)、video call(callMediaType = 2)
};
TUICallKit.groupCall(options, (res) => {
  if (res.code === 0) {
    console.log('call success');
  } else {
    console.log(`call failed, error message = ${res.msg}`);
  }
});
```



```
}  
});
```

Parameter	Type	Description
options	Object	Initialization parameters
options.groupID	String	Group ID for this group cal
options.userIDList	List	The target user IDs.
options.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
options.roomID	Number	Customize the numeric room number. As long as the roomID is present, the numeric room number is used, even if strRoomID is present.
options.strRoomID	String	Customize the string room number. If you want to use a string room number, you need to set roomID = 0 after setting strRoomID.

## joinInGroupCall

This API is used to join a group call.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  roomId: 9898,
  groupId: 'myGroup',
  callMediaType: 1, // audio call(callMediaType = 1)、video call(callMediaType = 2)
};
TUICallKit.joinInGroupCall(options, (res) => {
  if (res.code === 0) {
    console.log('joinInGroupCall success');
  } else {
    console.log(`joinInGroupCall failed, error message = ${res.msg}`);
  }
});
```

```
}  
});
```

Parameter	Type	Description
options	Object	Initialization parameters
options.roomID	Number	Customize the numeric room number. As long as the roomID is present, the numeric room number is used, even if strRoomID is present.
options.strRoomID	String	Customize the string room number. If you want to use a string room number, you need to set roomID = 0 after setting strRoomID.
options.groupID	String	Group ID for this group cal
options.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
callback	Function	callback function, code = 0 means the call was successful; code != 0 means the call failed, see msg for the reason.

## setCallingBell

To set a customized incoming call tone, here you are limited to passing in the local file address, and you need to make sure that the file directory is accessible to the application.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');

// 【1】 Save audio files locally through uni.saveFile. Reference.: https://zh.uniapp.cn/docs/uni-app-api/uni.saveFile
const tempFilePath = './static/rain.mp3';
let musicFilePath = '';
uni.saveFile({
  tempFilePath: tempFilePath,
  success: (res) => {
    musicFilePath = res.savedFilePath;

    // 【2】 Convert relative path to absolute path, otherwise access will not be successful
```

```
musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath);

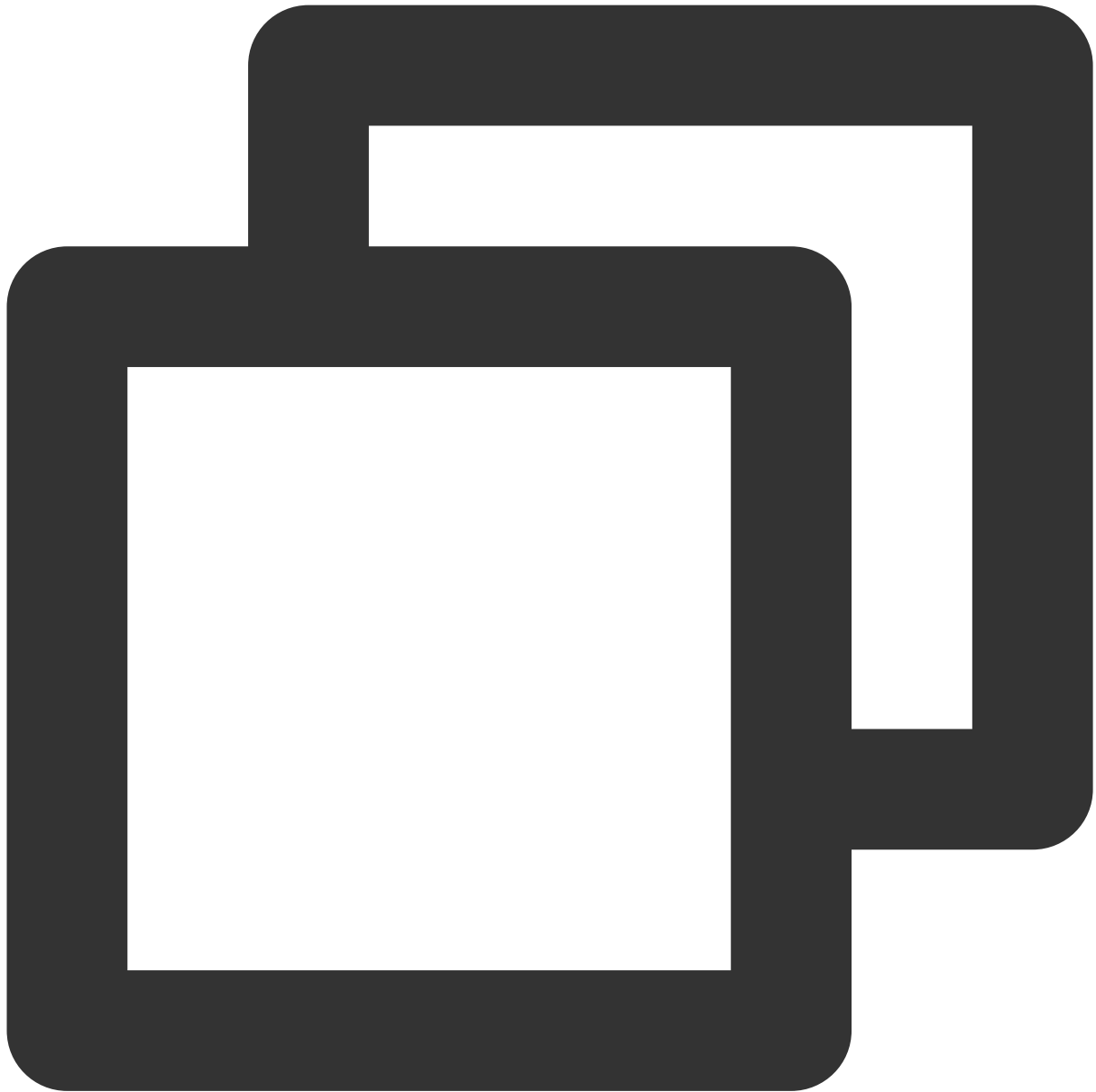
// 【3】 set ringtone
TUICallKit.setCallingBell(musicFilePath, (res) => {
  if (res.code === 0) {
    console.log('setCallingBell success');
  } else {
    console.log(`setCallingBell failed, error message = ${res.msg}`);
  }
});

},
fail: (err) => {
  console.error('save failed');
},
});
```

Parameter	Type	Description
filePath	String	Ringtone local file address
callback	Function	callback function, code = 0 means the call was successful; code != 0 means the call failed, see msg for the reason.

## enableMuteMode

This API is used to set whether to turn on the mute mode.



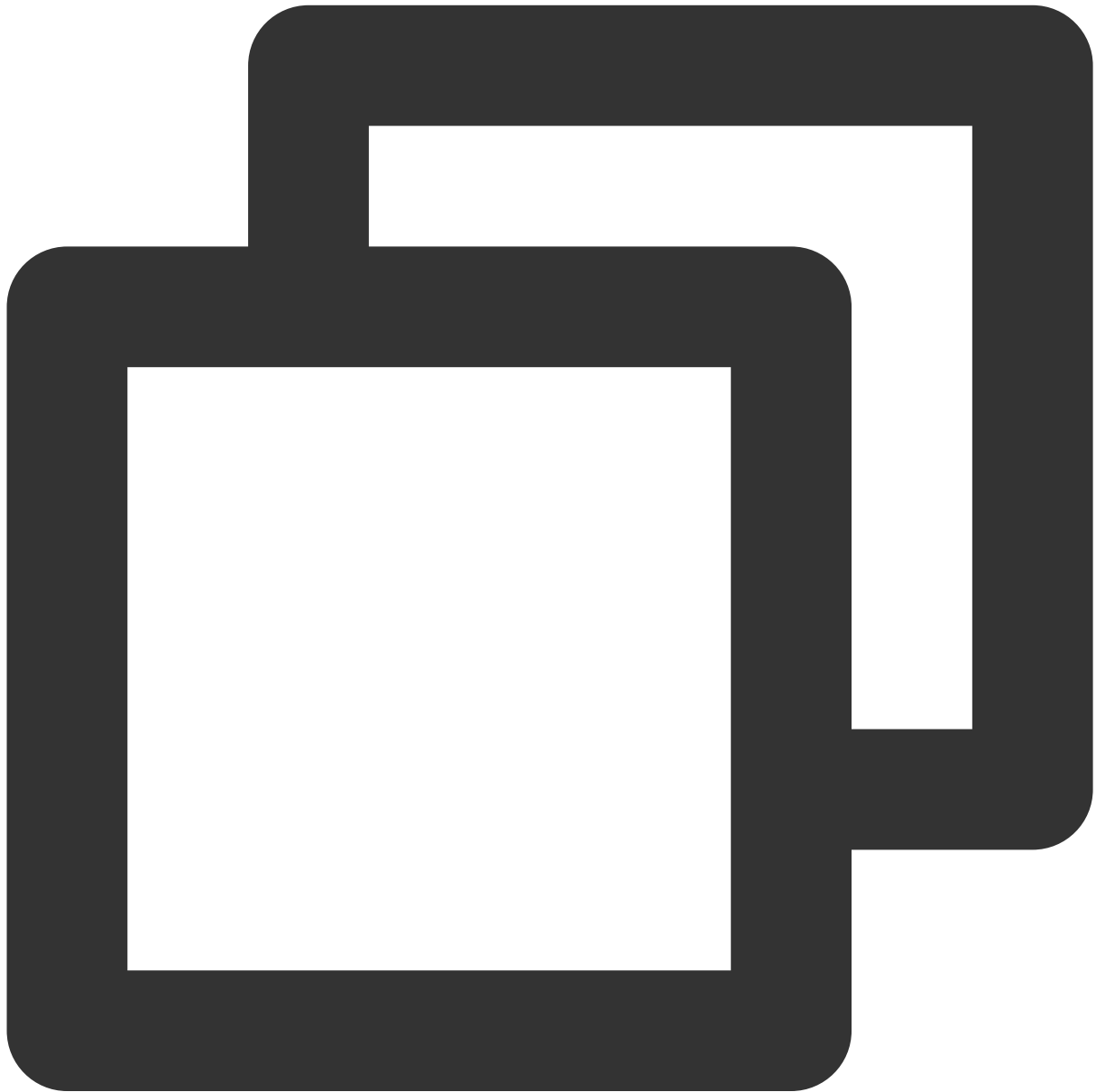
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');  
const enable = true;  
TUICallKit.enableMuteMode(enable);
```

Parameter	Type	Description
enable	Boolean	Mute on, mute off; true means mute on

## enableFloatWindow

This API is used to set whether to enable floating windows.

The default value is `false`, and the floating window button in the top left corner of the call view is hidden. If it is set to `true`, the button will become visible.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableFloatWindow(enable);
```

Parameter	Type	Description
enable	Boolean	Enable/disable the floating window function; true means floating window is enabled.





# TUICallEngine

Last updated : 2024-03-07 10:46:00

## TUICallEngine API

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

### Note

It needs to be used with [TencentCloud-TUICallKit](#) plugin and cannot be used alone.

## API Overview

API	Description
<a href="#">hangup</a>	Ends a call.
<a href="#">accept</a>	Accepts a call.
<a href="#">setVideoRenderParams</a>	Set the rendering mode of video image.
<a href="#">setVideoEncoderParams</a>	Set the encoding parameters of video encoder.

## API Details

### hangup

This API is used to end a call.



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngine');
TUICallEngine.hangup();
```

## accept

This API is used to accept a call. After receiving the `onCallReceived()` callback, you can call this API to accept the call.



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
TUICallEngine.accept();
```

### **setVideoRenderParams**

Set the rendering mode of video image.



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngine');
const params = {
  userID: '234',
  fillMode: 0, // 0-fill mode, 1-adapter mode
  rotation: 1, // 0:Rotation_0; 1: Rotation_90; 2: Rotation_180; 3: Rotation_270;
};
TUICallEngine.setVideoRenderParams(params, (res) => {
  console.warn('res = ', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
userID	String	target userId
params	Object	Video frame rendering parameters, e.g. frame rotation angle, fill mode

## setVideoEncoderParams

Set the encoding parameters of video encoder.

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngine');
const params = {
  resolution: 108,
  resolutionMode: 0, // 0--landscape, 1--portrait
};
TUICallEngine.setVideoEncoderParams(params, (res) => {
  console.warn('res = ', JSON.stringify(res));
});
```

The parameters are described below:

--	--	--

Parameter	Type	Description
resolution	Number	video resolution 62: aspect ratio 16:9 ; resolution 640x360 ; 64: aspect ratio 4:3 ; resolution 960x720 ; 108: aspect ratio 16:9 ; resolution 640x360 ; 110: aspect ratio 16:9 ; resolution 960x540 ; 112: aspect ratio 16:9 ; resolution 1280x720 ; 114: aspect ratio 16:9 ; resolution 1920x1080 ;
resolutionMode	Number	resolution mode 0: Landscape 1: Portrait

# Event

Last updated : 2024-03-07 10:35:23

You can listen to TUICallKit's corresponding events for prompts and other interactions.

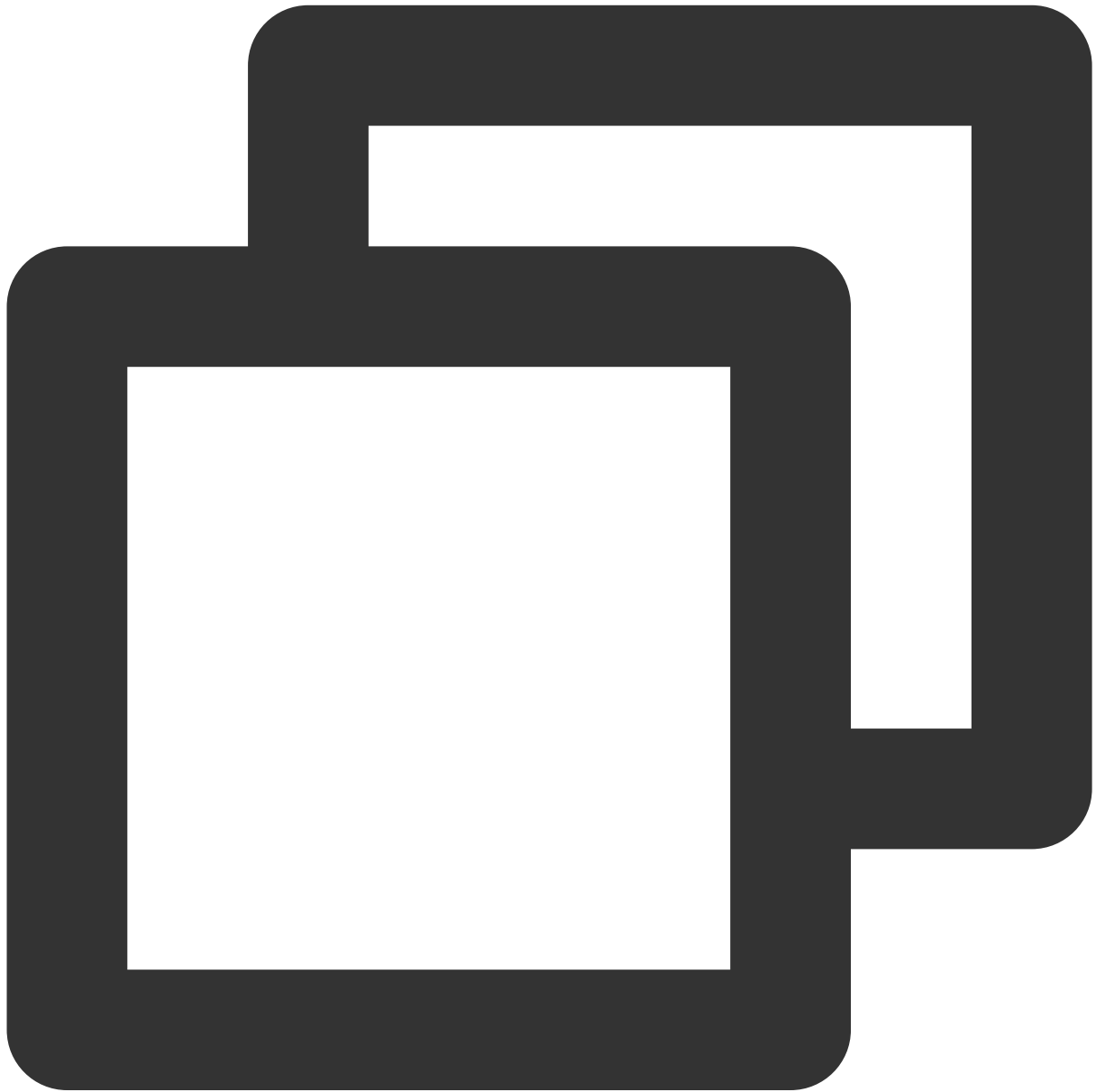
## Event Overview

API	Description
<a href="#">onError</a>	A call occurred during the call.
<a href="#">onCallReceived</a>	A call invitation was received.
<a href="#">onCallCancelled</a>	The call was canceled.
<a href="#">onCallBegin</a>	The call was connected.
<a href="#">onCallEnd</a>	The call ended.
<a href="#">onCallMediaTypeChanged</a>	The call type changed.
<a href="#">onUserReject</a>	A user declined the call.
<a href="#">onUserNoResponse</a>	A user didn't respond.
<a href="#">onUserLineBusy</a>	A user was busy.
<a href="#">onUserJoin</a>	A user joined the call.
<a href="#">onUserLeave</a>	A user left the call.
<a href="#">onUserVideoAvailable</a>	Whether a user had a video stream.
<a href="#">onUserAudioAvailable</a>	Whether a user had an audio stream.
<a href="#">onUserVoiceVolumeChanged</a>	The volume levels of all users.
<a href="#">onUserNetworkQualityChanged</a>	The network quality of all users.
<a href="#">onKickedOffline</a>	The current user was kicked offline.
<a href="#">onUserSigExpired</a>	The user sig is expired.



## Details

Listen to events thrown by the native plugin via globalEvent.



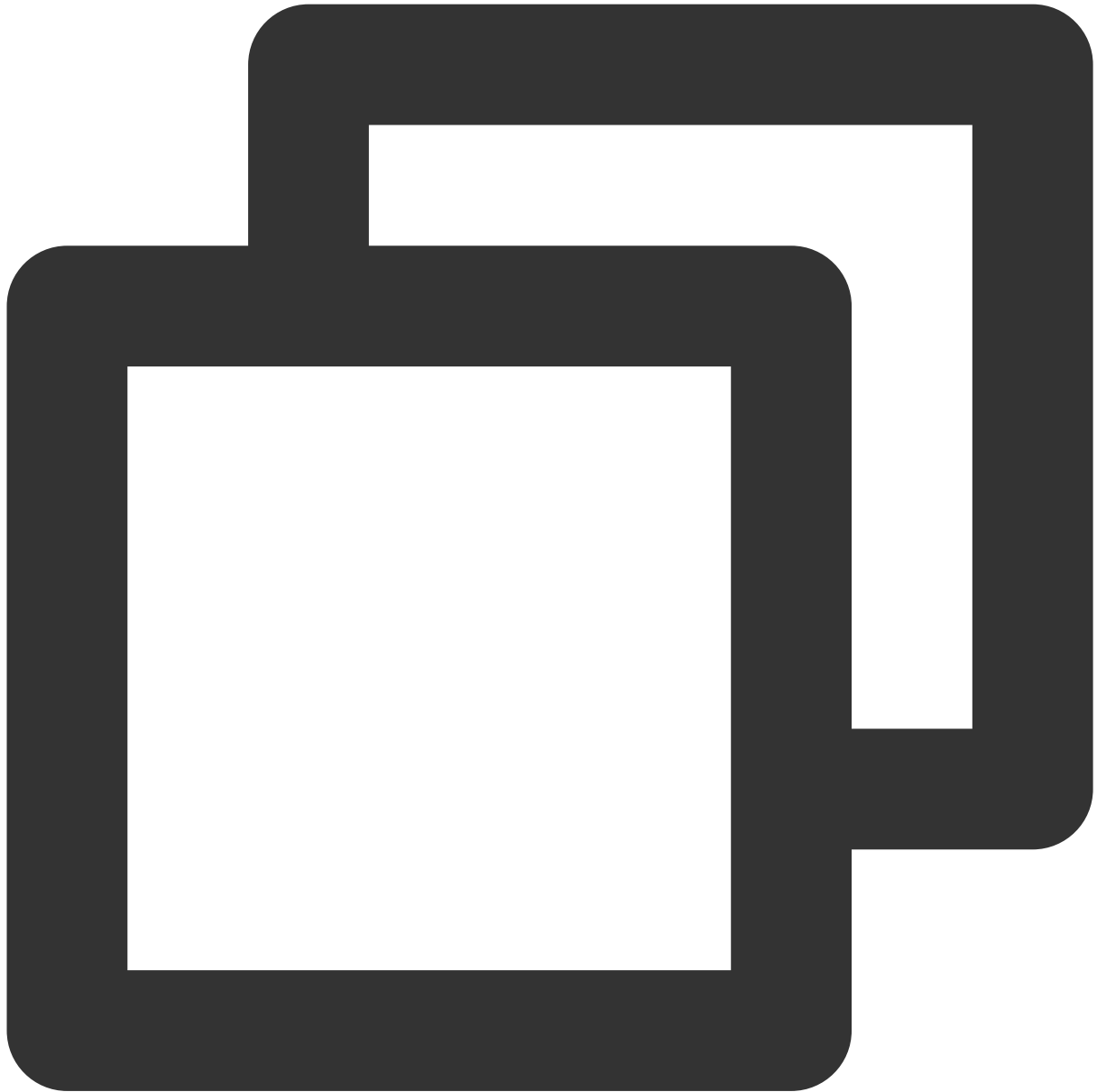
```
const TUICallingEvent = uni.requireNativePlugin('globalEvent');
```

### onError

An error occurred.

### Note

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



```
TUICallingEvent.addEventListener('onError', (res) => {  
  console.log('onError', JSON.stringify(res));  
});
```

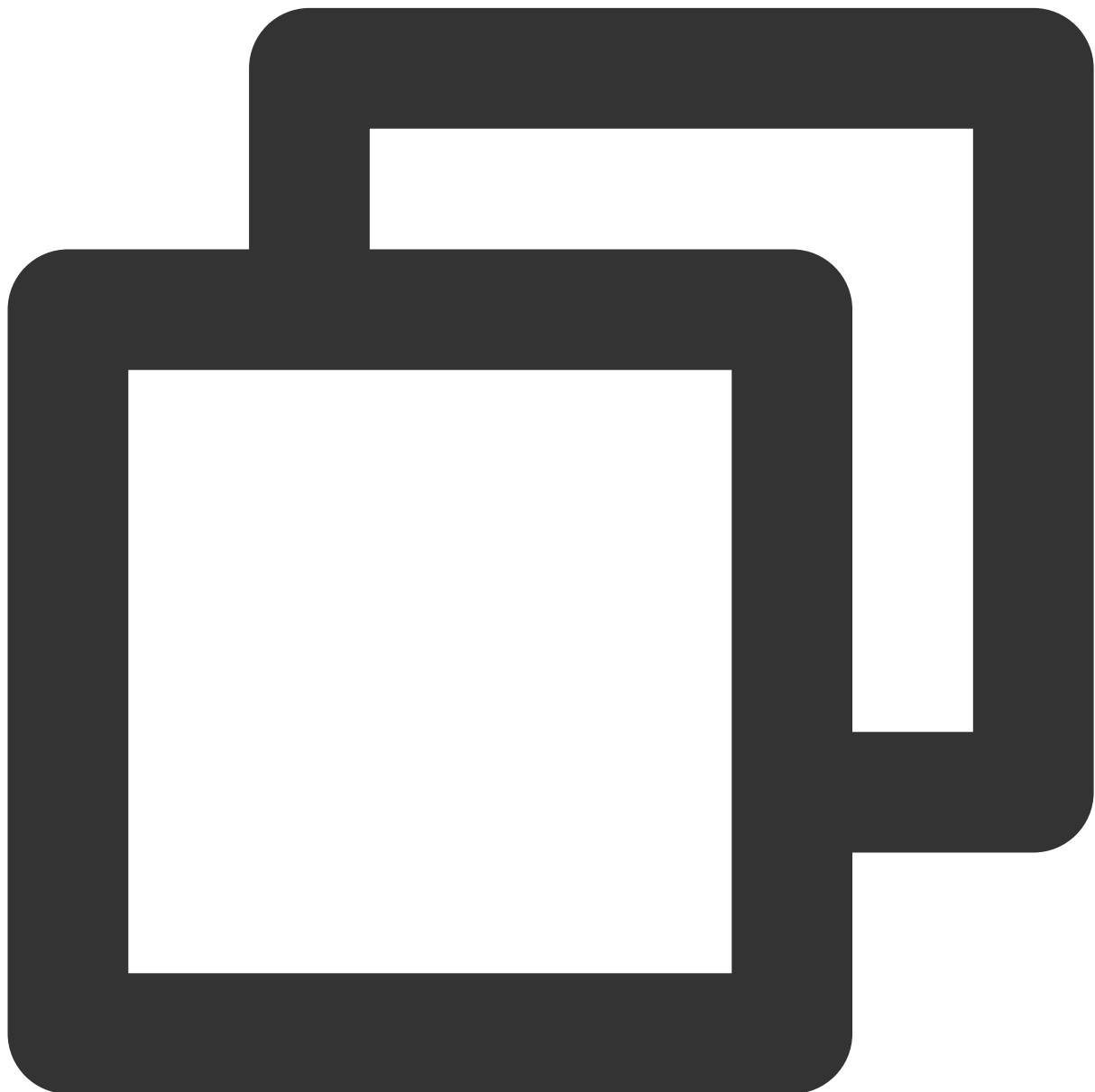
The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter

res.code	Number	The error code.
res.msg	String	The error message.

## onCallReceived

A call invitation was received. This callback is received by an invitee. You can listen for this event to determine whether to display the incoming call view.



```
TUICallingEvent.addListener('onCallReceived', (res) => {
```

```
console.log('onCallReceived', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.callerId	String	The user ID of the inviter.
res.calleeIdList	Array<String>	The invitee list.
res.groupId	String	The group ID.
res.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
res.userData	String	User-added extended fields.

## onCallCancelled

The call was canceled by the inviter or timed out. This callback is received by an invitee. You can listen for this event to determine whether to show a missed call message.

This indicates that the call was canceled by the caller, timed out by the callee, rejected by the callee, or the callee was busy. There are multiple scenarios involved. You can listen to this event to achieve UI logic such as missed calls and resetting UI status.

Call cancellation by the caller: The caller receives the callback (userId is himself); the callee receives the callback (userId is the ID of the caller)

Callee timeout: the caller will simultaneously receive the [onUserNoResponse](#) and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)

Callee rejection: The caller will simultaneously receive the [onUserReject](#) and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)

Callee busy: The caller will simultaneously receive the [onUserLineBusy](#) and onCallCancelled callbacks (userId is his own ID);

Abnormal interruption: The callee failed to receive the call , he receives this callback (userId is his own ID).



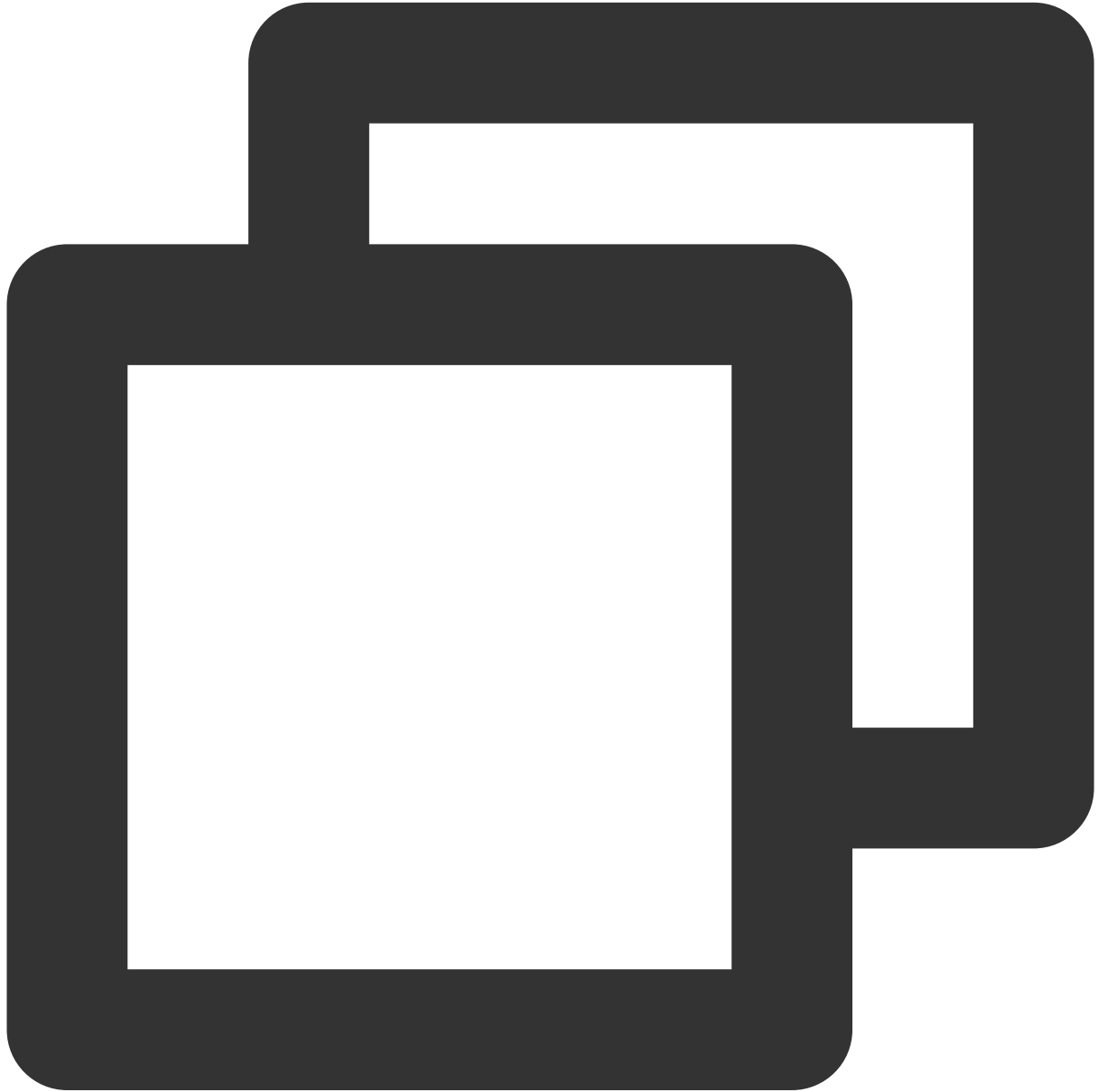
```
TUICallingEvent.addEventListener('onCallCancelled', (res) => {  
  console.log('onCallCancelled', res);  
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID of the inviter.

## onCallBegin

The call was connected. This callback is received by both the inviter and invitees. You can listen for this event to determine whether to start on-cloud recording, content moderation, or other tasks.



```
TUICallingEvent.addListener('onCallBegin', (res) => {  
  console.log('onCallBegin', JSON.stringify(res));  
});
```

The parameters are described below:

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

res	Object	callback parameter
res.roomID	Number	The room ID.
res.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
res.callRole	Number	The role, which can be caller or callee.

## onCallEnd

The call ended. This callback is received by both the inviter and invitees. You can listen for this event to determine when to display call information such as call duration and call type, or stop on-cloud recording.



```
TUICallingEvent.addEventListener('onCallEnd', (res) => {  
  console.log('onCallEnd', JSON.stringify(res));  
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.roomID	Number	The room ID.



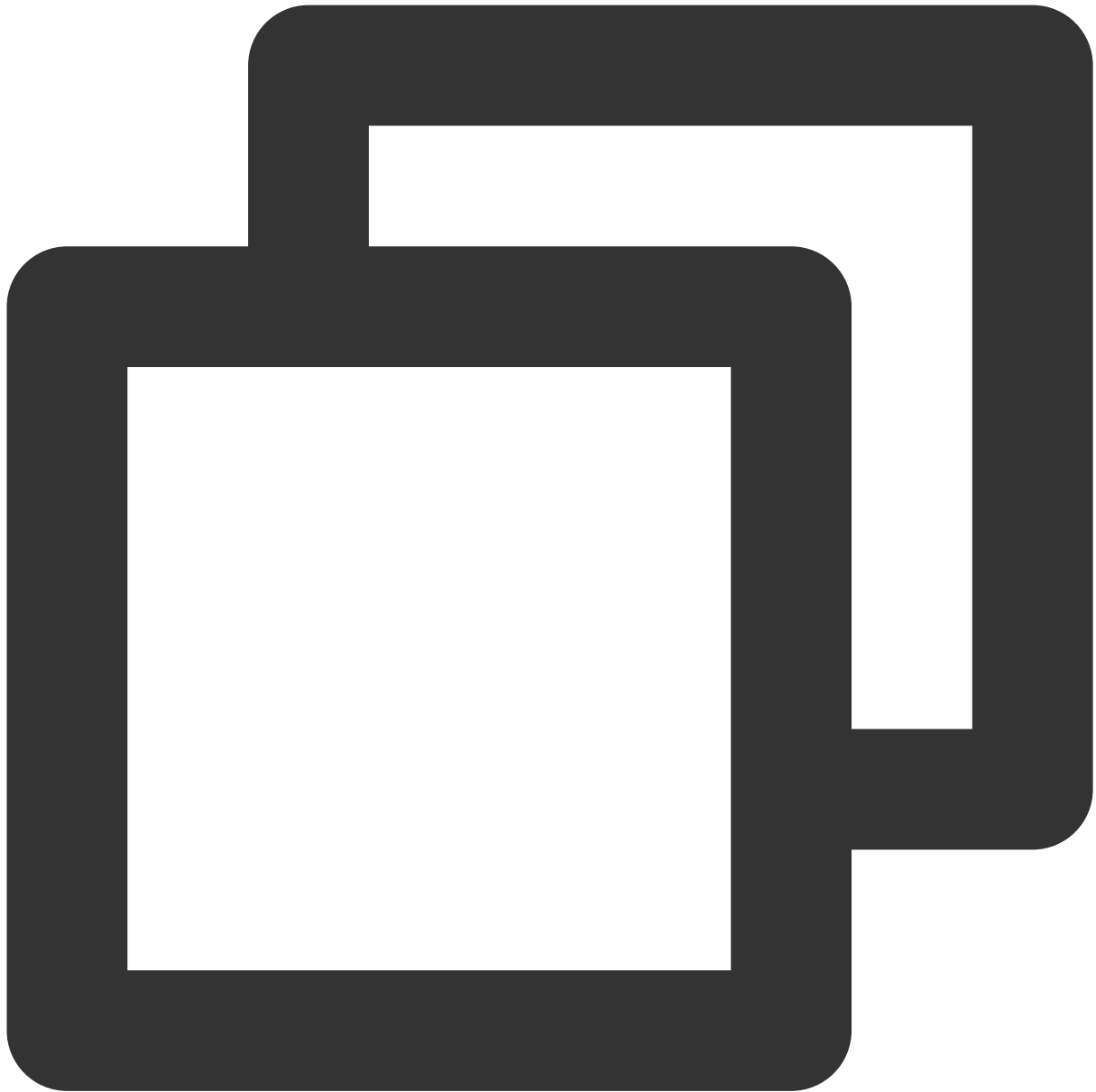
res.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
res.callRole	Number	The role, which can be caller or callee.
res.totalTime	Number	The call duration. unit: second

### Note

Client-side callbacks are often lost when errors occur, for example, when the process is closed. If you need to measure the duration of a call for billing or other purposes, we recommend you use the RESTful API.

### onCallMediaTypeChanged

The call type changed.



```
TUICallingEvent.addEventListener('onCallMediaTypeChanged', (res) => {  
  console.log('onCallMediaTypeChanged', JSON.stringify(res));  
});
```

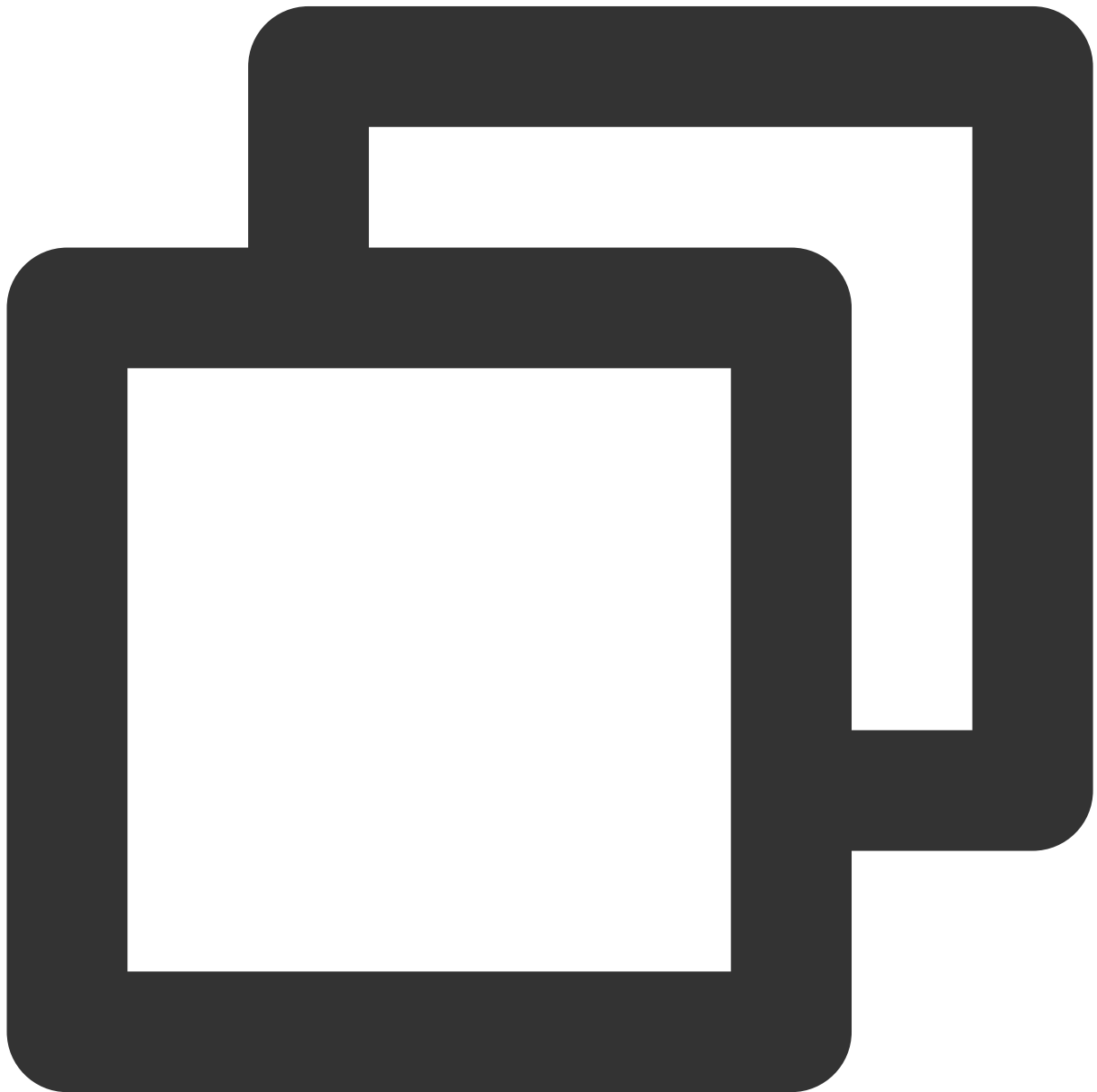
The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.oldCallMediaType	Number	The call type before the change. eg. audio call(callMediaType =

		1), video call(callMediaType = 2)
res.newCallMediaType	Number	The call type after the change. eg. audio call(callMediaType = 1), video call(callMediaType = 2)

### onUserReject

The call was rejected. In a one-to-one call, only the inviter will receive this callback. In a group call, all invitees will receive this callback.



```
TUICallingEvent.addListener('onUserReject', (res) => {
```

```
console.log('onUserReject', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID of the invitee who rejected the call.

## onUserNoResponse

A user did not respond.



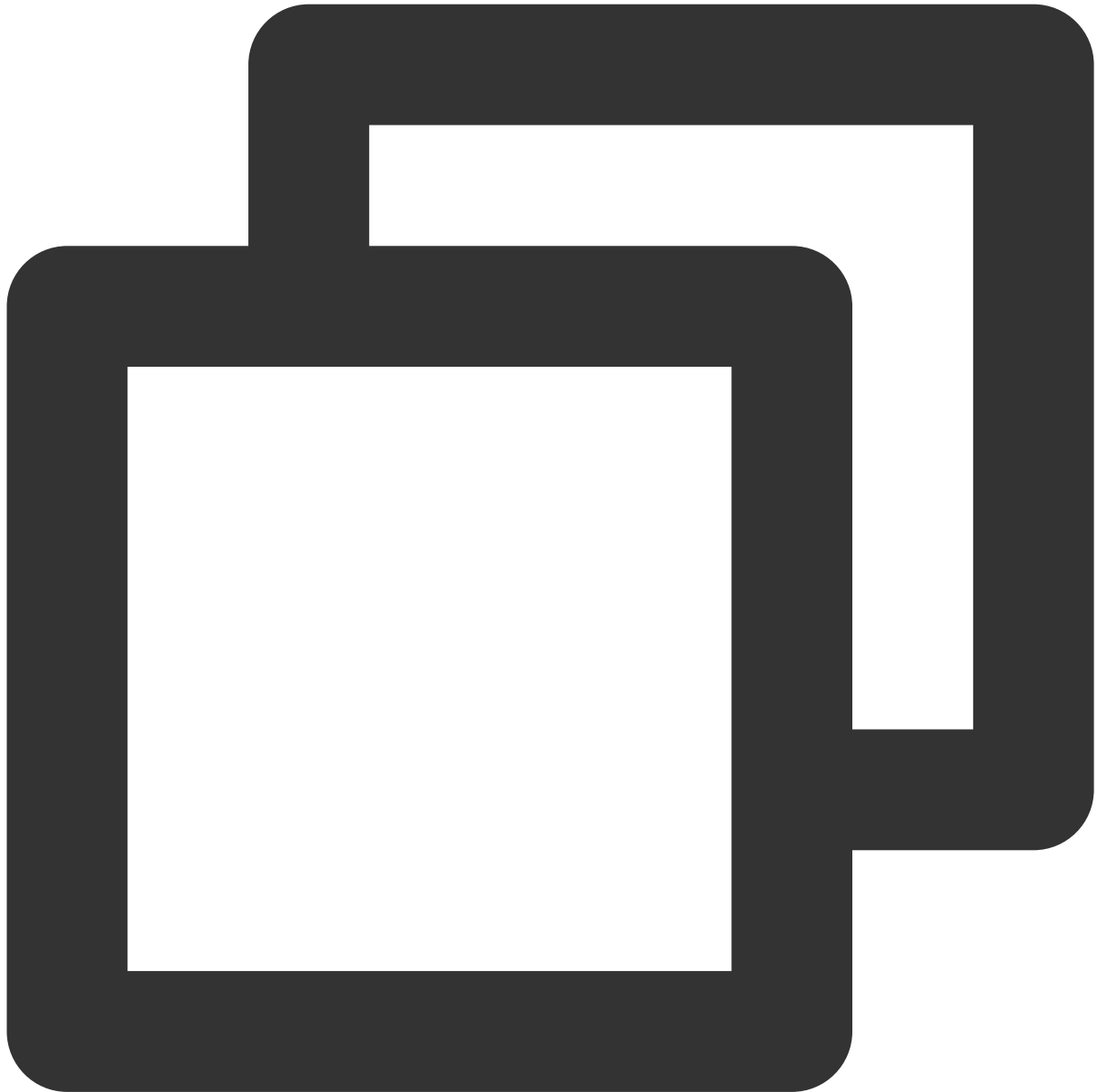
```
TUICallingEvent.addEventListener('onUserNoResponse', (res) => {  
  console.log('onUserNoResponse', JSON.stringify(res));  
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID of the invitee who did not answer.

## onUserLineBusy

A user is busy.



```
TUICallingEvent.addEventListener('onUserLineBusy', (res) => {  
  console.log('onUserLineBusy', JSON.stringify(res));  
});
```

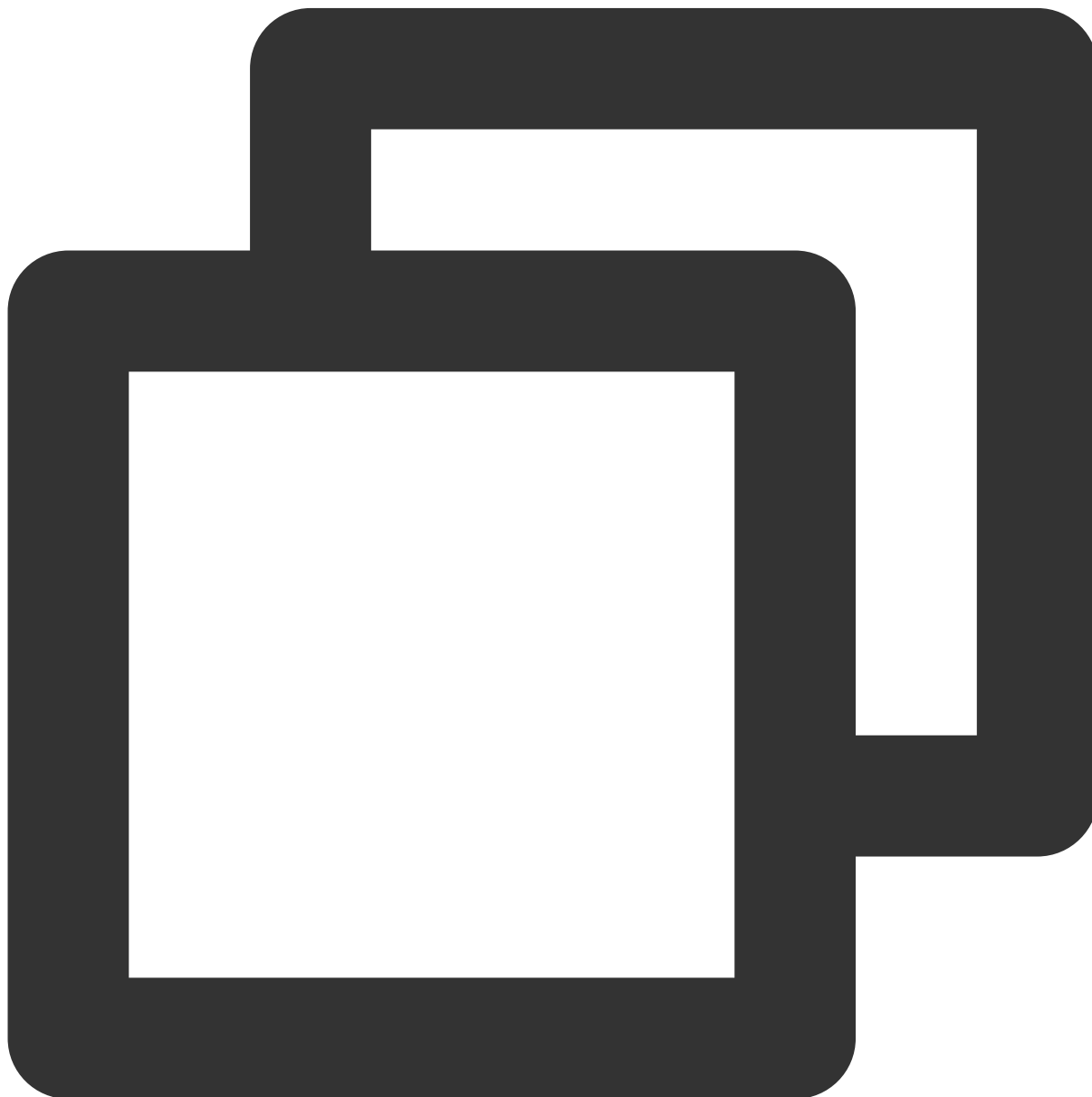
The parameters are described below:

Parameter	Type	Description

res	Object	callback parameter
res.userId	String	The user ID of the invitee who is busy.

## onUserJoin

A user joined the call.



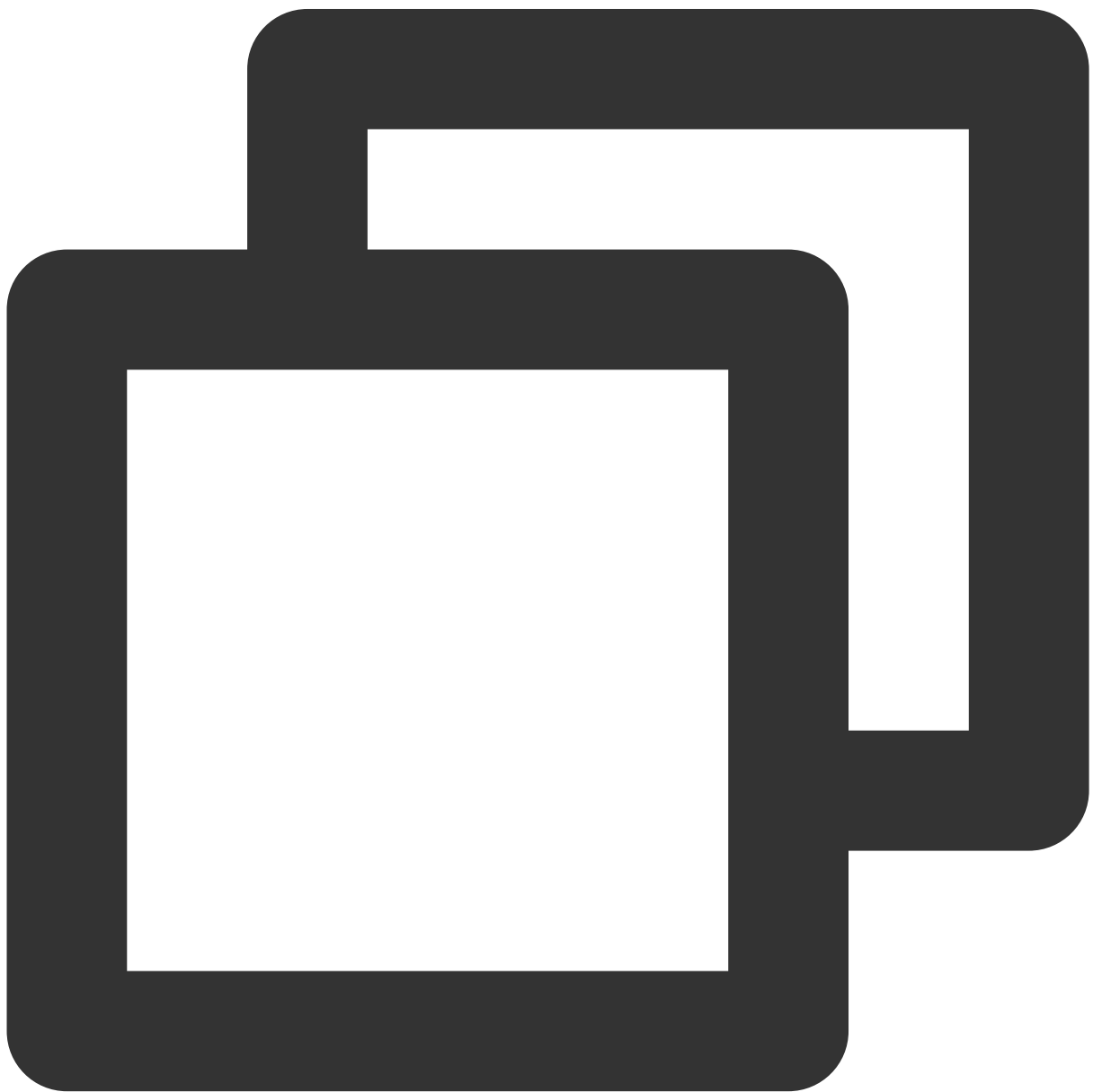
```
TUICallingEvent.addEventListener('onUserJoin', (res) => {  
  console.log('onUserJoin', JSON.stringify(res));  
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The ID of the user who joined the call.

### **onUserLeave**

A user left the call.





```
TUICallingEvent.addEventListener('onUserLeave', (res) => {  
  console.log('onUserLeave', JSON.stringify(res));  
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The ID of the user who left the call.

## onUserVideoAvailable

Whether a user is sending video.



```
TUICallingEvent.addEventListener('onUserVideoAvailable', (res) => {  
  console.log('onUserVideoAvailable', JSON.stringify(res));  
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID.

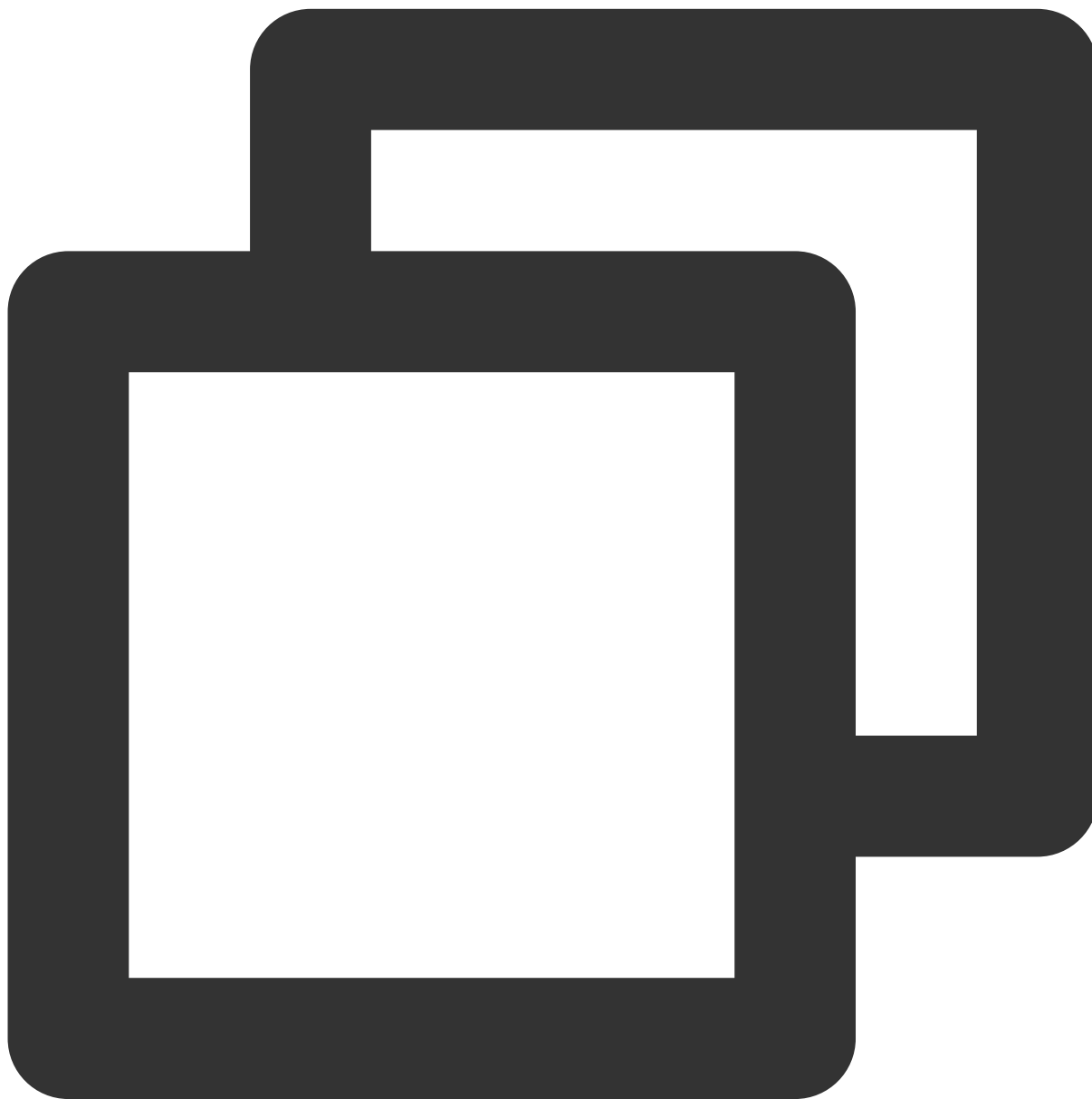
res.isVideoAvailable

boolean

Whether the user has video.

**onUserAudioAvailable**

Whether a user is sending audio.



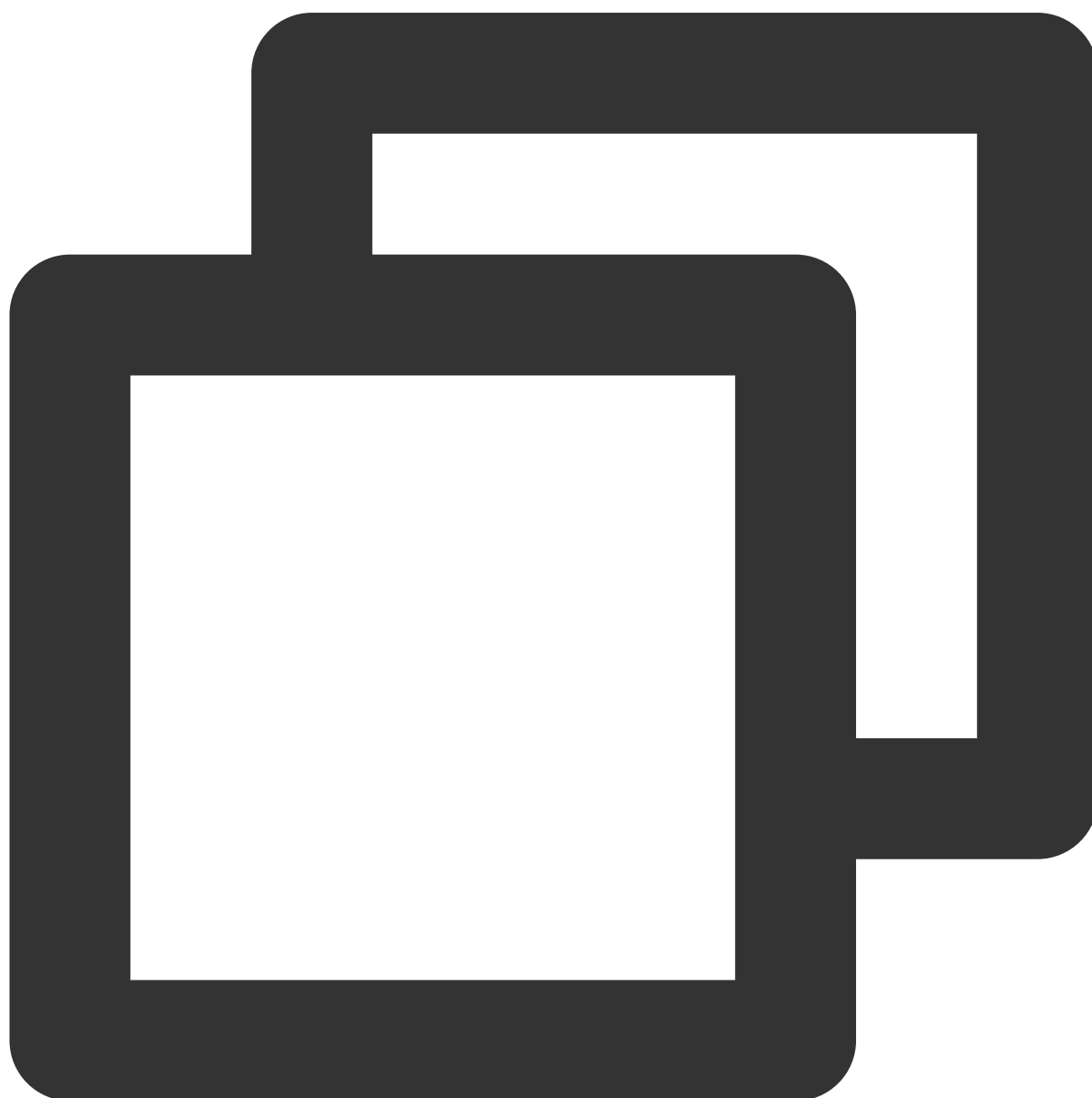
```
TUICallingEvent.addEventListener('onUserAudioAvailable', (res) => {  
  console.log('onUserAudioAvailable', JSON.stringify(res));  
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID.
res.isAudioAvailable	boolean	Whether the user has audio.

### onUserVoiceVolumeChanged

The volumes of all users.



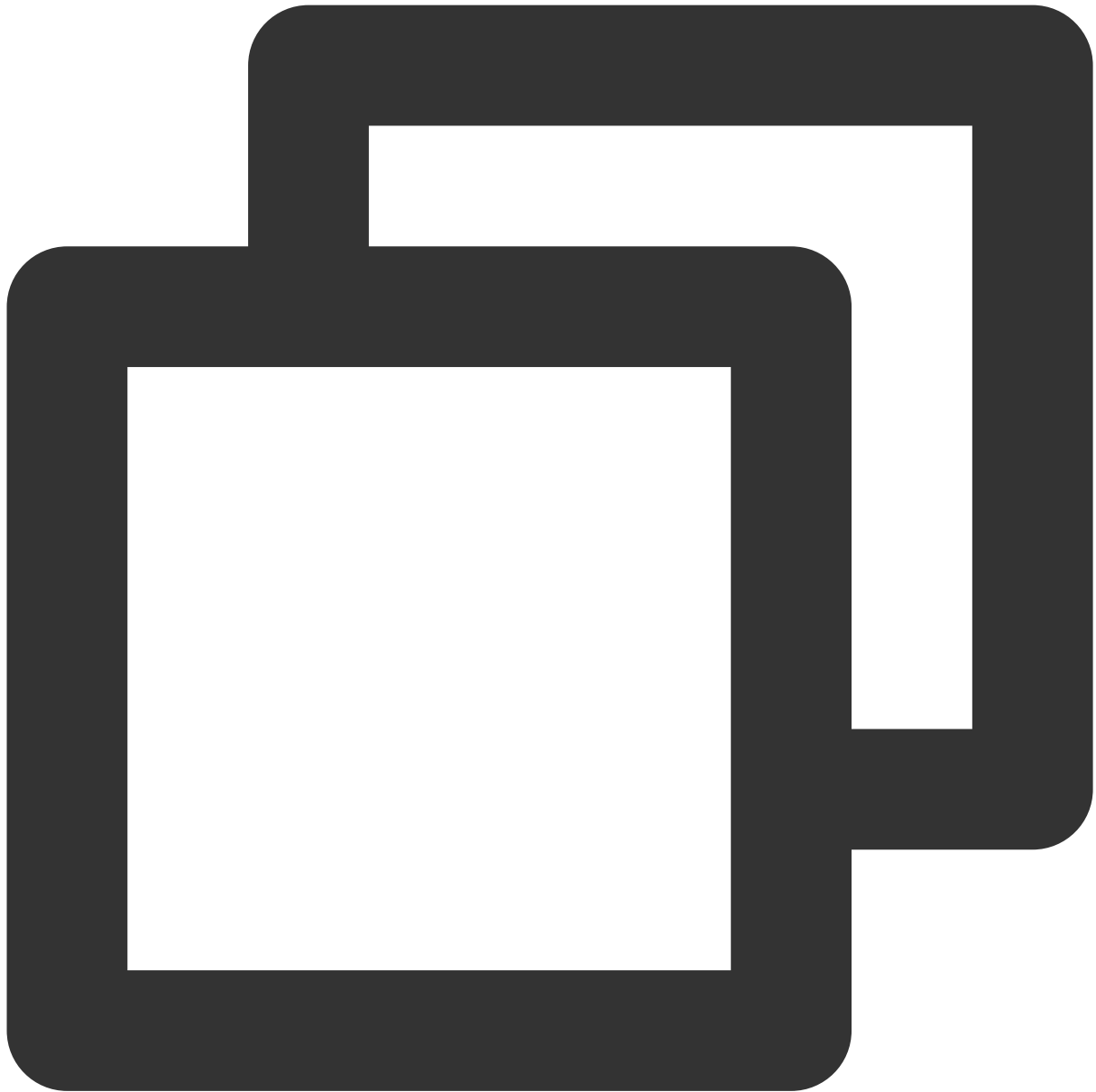
```
TUICallingEvent.addEventListener('onUserVoiceVolumeChanged', (res) => {  
    console.log('onUserVoiceVolumeChanged', JSON.stringify(res));  
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.volumeMap	Map<String, Integer>	The volume table, which includes the volume of each user (userId). Value range: 0-100.

### onUserNetworkQualityChanged

The network quality of all users.



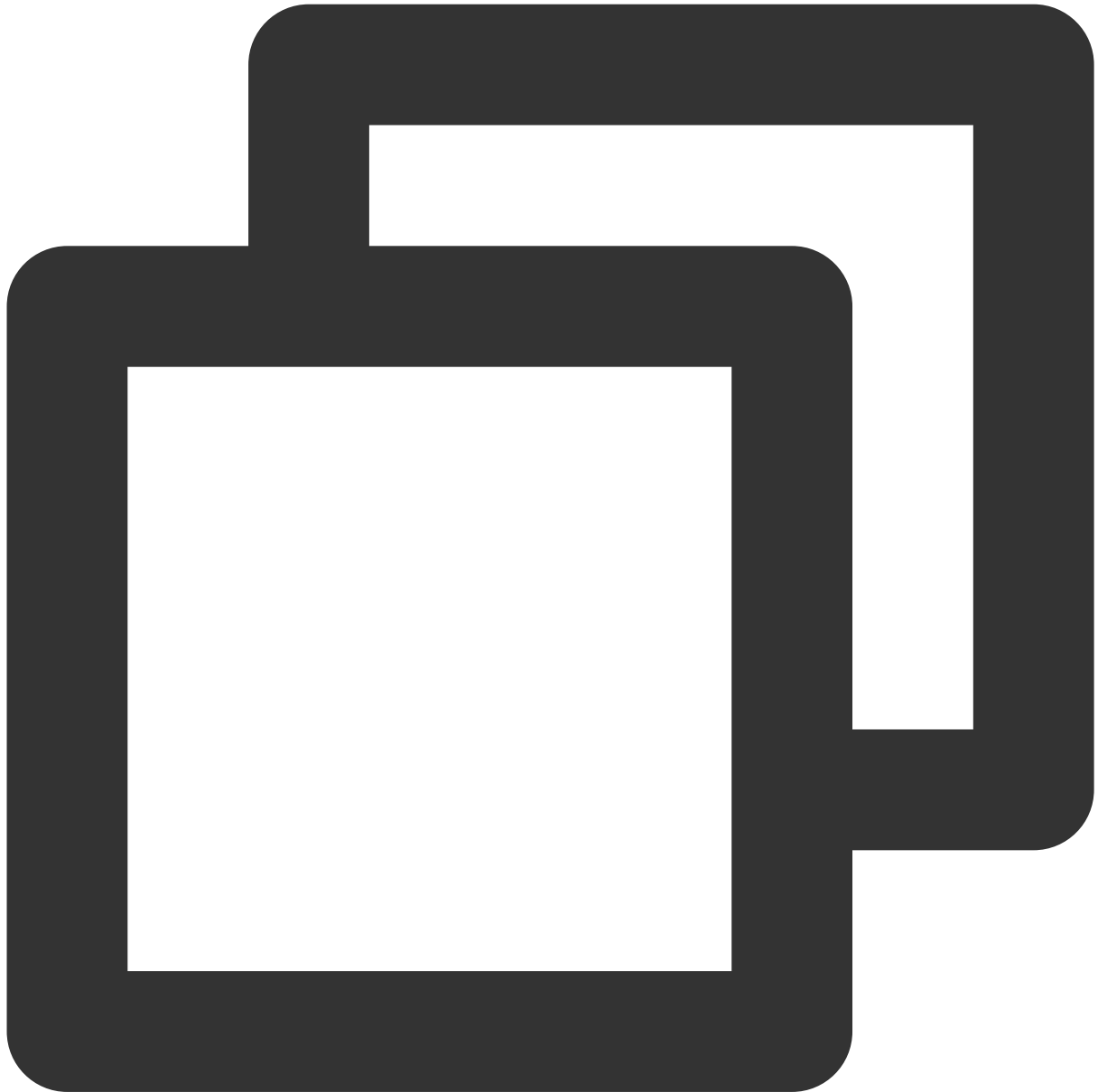
```
TUICallingEvent.addListener('onUserNetworkQualityChanged', (res) => {  
  console.log('onUserNetworkQualityChanged', JSON.stringify(res));  
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.networkQualityList	List	The current network conditions for all users (userId).

## onKickedOffline

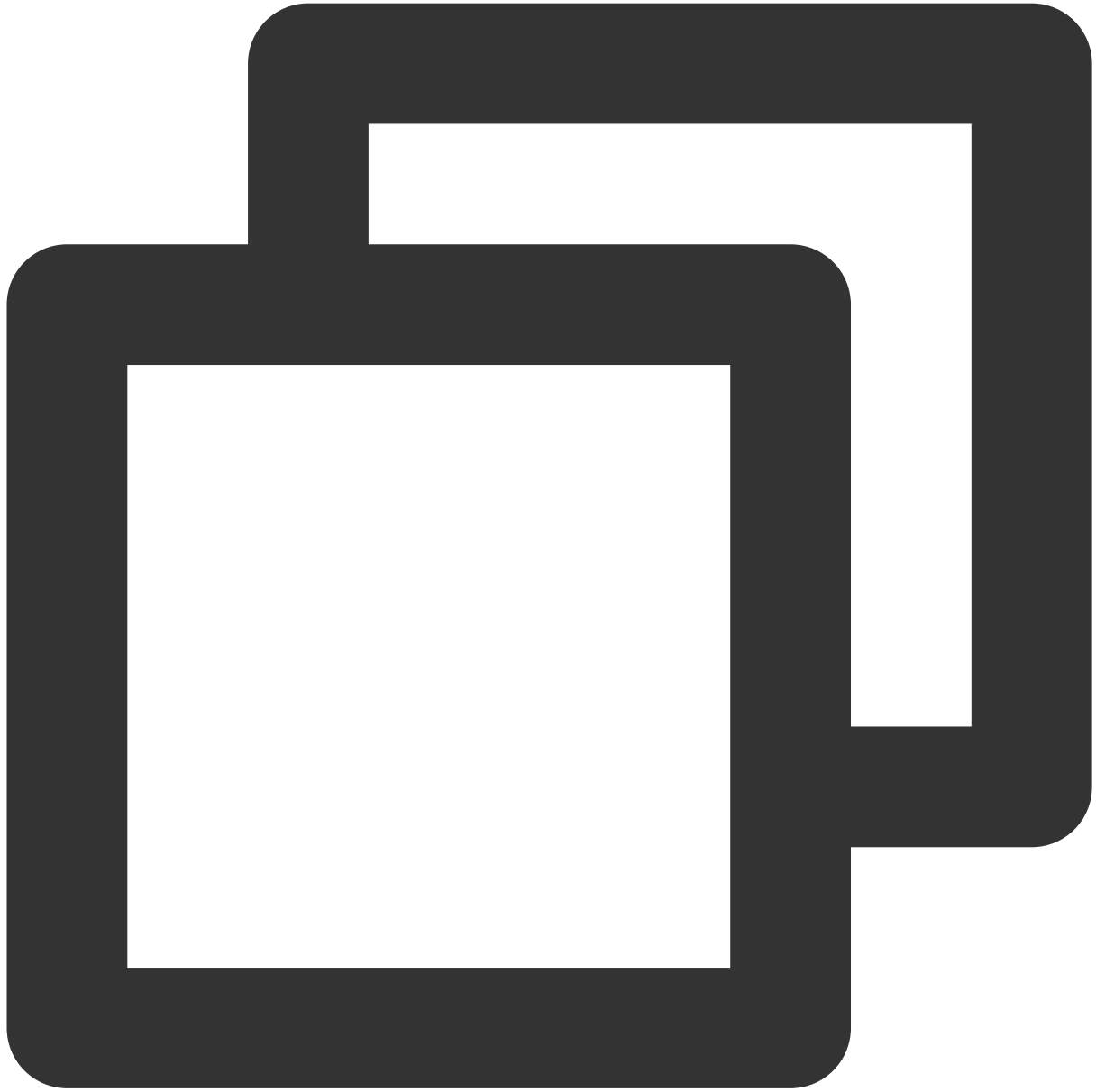
The current user was kicked offline : At this time, you can prompt the user with a UI message and then invoke `login` again.



```
TUICallingEvent.addListener('onKickedOffline', (res) => {  
  console.log('onKickedOffline', JSON.stringify(res));  
});
```

## onUserSigExpired

The user sig is expired : At this time, you need to generate a new `userSig` , and then invoke `login` again.



```
TUICallingEvent.addEventListener('onUserSigExpired', (res) => {  
  console.log('onUserSigExpired', JSON.stringify(res));  
});
```



# ErrorCode

Last updated : 2023-09-19 15:30:20

Notify users of warnings and errors that occur during audio and video calls.

## TUICallDefine Error Code

Definition	Value	Description
ERROR_PACKAGE_NOT_PURCHASED	-1001	You do not have TUICallKit package, please <a href="#">open the free experience</a> in the console or <a href="#">purchase the official package</a> .
ERROR_PACKAGE_NOT_SUPPORTED	-1002	The package you purchased does not support this ability. You can refer to console to purchase <a href="#">Upgrade package</a> .
ERROR_TIM_VERSION_OUTDATED	-1003	The IM SDK version is too low, please upgrade the IM SDK version to >= 6.6; Find and modify in the build.gradle file. : "com.tencent.imsdk:imsdk-plus:7.1.3925"
ERROR_PERMISSION_DENIED	-1101	Failed to obtain permission. The audio/video permission is not authorized. Check if the device permission is enabled.
ERROR_GET_DEVICE_LIST_FAIL	-1102	Failed to get the device list (only supported on web platform).
ERROR_INIT_FAIL	-1201	The <a href="#">init</a> method has not been called for initialization. The TUICallEngine API should be used after initialization.
ERROR_PARAM_INVALID	-1202	param is invalid.
ERROR_REQUEST_REFUSED	-1203	The current status can't use this function.
ERROR_REQUEST_REPEATED	-1204	The current status is waiting/accept, please do not call it repeatedly.
ERROR_SCENE_NOT_SUPPORTED	-1205	The current calling scene does not support this feature.
ERROR_SIGNALING_SEND_FAIL	-1406	Failed to send signaling. You can check the specific

		error message in the callback of the method.
--	--	--

## IM Error Code

Video and audio calls use Tencent Cloud's IM SDK as the basic service for communication, such as the core logic of call signaling and busy signaling. Common error codes are as follows:

Error Code	Description
6014	You have not logged in to the Chat SDK or have been forcibly logged out. Log in to the Chat SDK first and try again after a successful callback. To check whether you are online, use TIMManager.getLoginUser.
6017	Invalid parameter. Check the error information to locate the invalid parameter.
6206	UserSig has expired. Get a new valid UserSig and log in again. For more information about how to get a UserSig, see <a href="#">Generating UserSig</a> .
7013	The current package does not support this API. Please upgrade to the Flagship Edition package.
8010	The signaling request ID is invalid or has been processed.

Explanation :

More IM SDK error codes are available at : [IM Error Code](#)

## TRTC Error Code

Video and audio calls use Tencent Cloud's IM SDK as the basic service for calling, such as the core logic of switching camera and microphone on or off. Common error codes are as follows:

Enum	Value	Description
ERR_CAMERA_START_FAIL	-1301	Failed to turn the camera on. This may occur when there is a problem with the camera configuration program (driver) on Windows or macOS. Disable and reenale the camera, restart the camera, or update the configuration program.
ERR_CAMERA_NOT_AUTHORIZED	-1314	No permission to access to the camera. This usually occurs on mobile devices and may be because the user denied access.

ERR_CAMERA_SET_PARAM_FAIL	-1315	Incorrect camera parameter settings (unsupported values or others).
ERR_CAMERA_OCCUPY	-1316	The camera is being used. Try another camera.
ERR_MIC_START_FAIL	-1302	Failed to turn the mic on. This may occur when there is a problem with the mic configuration program (driver) on Windows or macOS. Disable and reenale the mic, restart the mic, or update the configuration program.
ERR_MIC_NOT_AUTHORIZED	-1317	No permission to access to the mic. This usually occurs on mobile devices and may be because the user denied access.
ERR_MIC_SET_PARAM_FAIL	-1318	Failed to set mic parameters.
ERR_MIC_OCCUPY	-1319	The mic is being used. The mic cannot be turned on when, for example, the user is having a call on the mobile device.
ERR_TRTC_ENTER_ROOM_FAILED	-3301	Failed to enter the room. For the reason, refer to the error message for -3301.

Explanation :

More TRTC SDK error codes are available at : [TRTC Error Code](#)

# Release Notes (TUICallKit)

## Web

Last updated : 2024-04-03 18:02:21

### Note:

TUICallKit Vue3 Github version address: [Github TUICallKit Web](#).

## Version 3.2.1 @2024.03.08

### Added

Language log reporting.

## Version 3.2.0 @2024.02.23

### Added

Added default offline push parameters.

### Fix

Fix the issue where group calls have no nickname.

## Version 3.1.9 @2024.01.30

### Fix

Fix the issue where group calls do not display user information.

Fixed the issue where the 'Confirm' button was still clickable in the selector component when there were no members available.

Fixed the issue where muting the microphone during a call would prevent the audio stream from being transmitted in subsequent calls (upgrade [trtc-cloud-js-sdk](#) to v2.2.7+).

## Version 3.1.8 @2024.01.19

### Fix

Fixed the impact of the selector component's style on the page.

## Version 3.1.7 @2024.01.12

### Fix

Added a retry mechanism for interfaces and fixed the playback failure issue due to the inability to find the DOM node.

Fixed the device list selection style issue on PC.

## Version 3.1.6 @2023.12.29

### Optimization

Optimized the prompt message during group calls.

Optimized the display issue when the nickname is too long.

### Fix

Fixed the camera permission issue for voice call requests.

Fixed the issue with 'destroyed'.

Fixed the hang-up issue in the floating window under different call scenarios.

Fixed the display remote issue during the caller call status.

Fixed the incomplete fill style issue on PC.

## Version 3.1.5 @2023.12.15

### Added

Optimized the timing of accessing device permissions. No longer access device permissions during initialization, only access when using call.

### Fix

Fixed [@tencentcloud/call-uikit-vue2](#), [@tencentcloud/call-uikit-vue2.6](#) components not having declare file issues.

## Version 3.1.4 @2023.12.01

### Added

Integrated into Chat, added isFromChat reporting.

**Fix**

Fixed the issue where the button is clickable under loading.

## Version 3.1.3 @2023.11.17

**Added**

Added parameter validation to the interface.

## Version 3.1.2 @2023.11.03

**Added**

Add the inviteUser feature for inviting others.

Added the feature to add people mid-call joinInGroupCall.

Introduced the feature to mute incoming call ringtones enableMuteMode.

**Fix**

Fixed the issue where remote stream microphone status was displayed incorrectly.

## Version 3.1.0 @2023.10.20

**Added**

Added Floating Window feature.

Added enableFloatWindow interface for enabling/disabling Floating Window feature.

Desktop Terminal supports switching Camera and Microphone devices.

Added failure prompt message for calling blocked users.

Added support for Japanese.

**Optimization**

During video calls, the big screen defaults to displaying the remote user.

## Version 3.0.8 @2023.10.10

**Added**

Added reporting for version number, framework, and other information.

## Version 3.0.7 @2023.10.08

### Added

Add desktop video call duration display.

### Optimization

Optimized desktop video stream preview of rounded corners and black edges.

Optimized display priority for remote stream user information: Remarks > Nickname > userId.

Optimized TUICallKit component package size (Removed unused images and code).

## Version 3.0.6 @2023.09.19

### Fix

Fixed the message display issue integrated into TUIKit.

## Version 3.0.5 @2023.09.15

### Optimization

Optimized mutual references in TUICallKit to avoid the stack overflow issue that occurs when packaging mini-programs in uniapp.

### Added

Added a prompt for desktop devices when there is no permission, guiding customers on how to authorize devices.

### Fix

Fixed setCallingBell where the called ringtone was overridden by the calling ringtone, leading to ringtone repetition issue.

Fixed styling issues on mobile devices.

## Version 3.0.4 @2023.09.01

### Fix

Fixed setCallingBell targeting incoming call ringtone (called ringtone).

Fixed destroyed error reporting problem.

Fixed the lack of Chinese and English in the error popup prompt.

Fixed the issue where it is impossible to switch between multi-screen support after turning off the camera during a 1v1 call.

## Version 3.0.3 @2023.8.25

### Added

Add [@tencentcloud/call-uikit-vue2.6](#) compatible with Vue 2.6 version.

### Optimization

Optimize the default language of the component to the system default language.

Optimize the log information printed.

Optimize error message thrown by [tuicall-engine-webrtc](#).

Optimize resource cleanup after component destruction.

### Fix

Fixed an issue where videoDisplayMode,videoResolution did not take effect when calling again after hanging up.

Fixed the issue where statusChanged was not triggered during the call.

Fixed the issue of init being called multiple times.

Fixed the issue of being unable to switch between full and split screens when turning off the camera during a call.

## Version 3.0.2 @2023.8.14

### Fix

Fixed styling issues of the called component on the H5 platform.

Fixed the styling issues that occurred after switching to a small window during another call.

## Version 3.0.1 @2023.8.8

### Fix

Fixed the issue of the caller's local preview failing during a group call, and modified the component layer's default reading mode from the data layer.

## Version 3.0.0 @2023.8.4



## Breaking Change

Upgraded the underlying dependency [tuicall-engine-webrtc](#) to ^2.0.0. It no longer supports creating tim instances with [tim-js-sdk](#). If you need to create a tim instance, please use [@tencentcloud/chat](#).

## Add

Add the custom ringtone feature `setCallingBell`.

## Version 2.4.2 @2023.11.03

### Added

Add the `inviteUser` feature for inviting others.

Added the feature to add people mid-call `joinInGroupCall`.

Introduced the feature to mute incoming call ringtones `enableMuteMode`.

### Fix

Fixed the issue where remote stream microphone status was displayed incorrectly.

## Version 2.4.0 @2023.10.20

### Added

Added Floating Window feature.

Added `enableFloatWindow` interface for enabling/disabling Floating Window feature.

Desktop Terminal supports switching Camera and Microphone devices.

Added failure prompt message for calling blocked users.

Added support for Japanese.

### Optimization

During video calls, the big screen defaults to displaying the remote user.

## Version 2.3.9 @2023.10.10

### Added

Added reporting for version number, framework, and other information.

## Version 2.3.8 @2023.10.08

### Added

Add desktop video call duration display.

### Optimization

Optimized desktop video stream preview of rounded corners and black edges.

Optimized display priority for remote stream user information: Remarks > Nickname > userId.

Optimized TUICallKit component package size (Removed unused images and code).

## Version 2.3.6 @2023.09.15

### Optimization

Optimized mutual references in TUICallKit to avoid the stack overflow issue that occurs when packaging mini-programs in uniapp.

### Added

Added a prompt for desktop devices when there is no permission, guiding customers on how to authorize devices.

### Fix

Fixed setCallingBell where the called ringtone was overridden by the calling ringtone, leading to ringtone repetition issue.

Fixed styling issues on mobile devices.

## Version 2.3.5 @2023.9.5

### Fix

Fixed the issue where the camera and microphone buttons were by default turned on before entering the room.

## Version 2.3.4 @2023.9.1

### Added

Add the feature to disable or enable the camera before answering a video call.

### Fix

Fixed the issue where it is impossible to switch screen sizes after turning off the camera during a 1v1 call.

Fixed the issue where `statusChanged` was not triggered when switching from a video call to a voice call.

## Version 2.3.3 @2023.8.22

### Fix

Fixed an issue where `videoDisplayMode`, `videoResolution` did not take effect when calling again after hanging up.

Fixed the issue where `statusChanged` was not triggered during the call.

## Version 2.3.2 @2023.7.26

### Breaking Change

Removed the `TUICallKitMini` floating window component, merged it into the `TUICallKit` component.

The thrown `@kicked-out` event has been adjusted to the affinity callback `:kickedOut`.

The thrown `@status-changed` event has been adjusted to the affinity callback `:statusChanged`.

### Add

Add animation effect when the call page appears.

Add group call layout on H5.

### Optimization

Optimize the problem prompt message during the call, prompt method.

Optimize support on H5 page, interaction.

Optimize the time it takes to bring up the call interface.

Optimize the [@tencentcloud/call-uikit-vue](#) package directory structure.

### Fix

Fixed call issues under boundary operations such as immediately hanging up after connecting.

Fixed styling issues on H5 for some models, browsers.

Fixed call anomaly issues caused by repeated clicks.

## Version 2.2.1 @2023.7.7

### Add

[@tencentcloud/call-uikit-vue2](#) Add detection and prompt for the Vue version.

## Fix

Fixed the issue where repeatedly clicking the "Answer" button on the incoming call page causes the answer to fail.

## Version 2.2.0 @2023.6.30

### Add

call,groupCall support custom roomId parameter for digital room numbers.

call,groupCall support custom userData parameter for extended fields (used to add additional information in the invitation signaling).

Add setSelfInfo interface, supporting user configuration of aliases and profile photos.

## Version 2.1.0 @2023.4.14

### Add

In the H5 voice chat pattern, while calling, it supports displaying the opposite party's nickname.

When initiating a call fails, "Call initiation failed" will be displayed on the calling page.

When answering a call fails, "Answer failed" will be shown on the incoming call page.

Support for monitoring whether the current user is kicked out (e.g., due to being logged out), see TUICallKit Method - @kicked-out.

Support for listening to TUICallKit call status, see TUICallKit Method - @status-changed.

Support for business-side code to control the answering, canceling, and hanging up of calls, see More Features - Auto-answer through Interface Setting.

The Vue2 version adds TypeScript type declaration files, allowing normal compilation of types in TypeScript projects.

## Fix

Fixed a warning about updating personal profile interface appearing in the console during component initialization.

Fixed background image misalignment issues of the callee answer button in the H5 pattern.

## Interface Change

`TUICallKitServer.destroy()` Added invocation limit, can only be called in non-call status.

## Version 2.0.1 @2023.03.31

### Add

Optimized the rendering logic of 1v1 and group call videos to improve performance and stability.

Optimized UI presentation, support for displaying corresponding UI during the execution of

`TUICallKitServer.call()`, which enables immediate UI display of `<TUICallKit/>` components upon clicking the call button.

### **Fix**

Fixed the issue of incorrect nickname display in group calls.

Fixed the issue of CSS not being scoped properly, leading to global style pollution.

## Version 2.0.0 @2023.03.21

### **Add**

Support for importing the packaged CallKit file from npm.

Support for Vue projects in JavaScript version.

Supports all versions of Vue2 projects, applicable to the npm package for Vue2: [call-uikit-vue2](#).

### **Fix**

Fixed the issue where calls could not be initiated due to the absence of a camera device or permission.

## Version 1.4.2 @2023.03.03

### **Add**

Supports setting call resolution. See API Documentation for details.

Supports changing the display pattern. See API Documentation for details.

Optimized the integration steps.

Optimized error throwing.

## Version 1.4.1 @2023.02.13

### **Add**

Optimized the logic for previewing the local camera.

Optimized the rendering logic for remote video streams.

## Version 1.4.0 @2023.01.06

**Add**

Supports importing in Vue2.7+ projects.

Call interface defaults to displaying nicknames. For setting nicknames, refer to [TIM#updateMyProfile](#).

## Version 1.3.3 @2022.12.27

**Add**

Added null value detection for the call list when making calls in the Basic Demo.

Added a loading icon when making calls in the Basic Demo.

Optimized the logic for device detection in the Basic Demo, no longer proactively popping up after manually skipping.

Optimized the reference method for component icons.

Changed the default package management tool to npm.

Optimized the rendering method for videos, reducing the number of iterative renderings.

**Fix**

Fixed an error in the Basic Demo caused by outdated dependencies in vue-CLI.

## Version 1.3.2 @2022.12.07

**Add**

Language switching is supported, see `setLanguage` for interface details.

Optimized the device detection logic in the Basic Demo; it will no longer pop up proactively after being manually skipped.

**Fix**

Fixed a warning caused by introducing `defineProps`.

## Version 1.3.1 @2022.11.29

**Note:**

This version depends on the SDK version [tuicall-engine-webrtc@1.2.1](#), please update promptly.

**Add**

Optimize style details.

Support monitoring the other party's modification of call type when the call is not answered.

Basic demo adds device detection feature.

**Fix**

Fixed errors caused by internal logic when hanging up the phone.

## Version 1.3.0 @2022.11.14

**Add**

Supports automatic switching to vertical screen style when using mobile H5.

Supports previewing the local camera when making a phone call.

Basic demo adds device detection before making a phone call.

**Fix**

Fixed the issue where the tim instance did not fully log out after calling `TUICallKitServer.destroyed()`.

Fixed the problem where a 'No response' message was displayed when the line was busy.

Fixed the issue where TypeScript types were not successfully packaged in a vite environment.

**Interface Change**

When actively calling `TUICallKitServer.call()` or `TUICallKitServer.groupCall()`, if an error occurs, the `beforeCalling` callback will not be invoked. Please use try catch to capture errors directly.

## Version 1.2.0 @2022.11.03

**Add**

Adaptation to new versions of TUICallEngine SDK.

## Version 1.1.0 @2022.10.21

**Add**

During a call, the call page can be displayed in full screen.

During a call, you can use `<TUICallKitMini/>` to minimize.

**Fix**

Fixed known issues, improved stability.

## Version 1.0.3 @2022.10.14

**Add**

Basic demo adds quick copy UserID, one-click open new window.

## Version 1.0.2 @2022.09.30

**Add**

Optimized access documentation, added demonstration images and detailed guides.

**Fix**

Fixed the issue where the device status bit became invalid when first entering the room.

Fixed the occasional failure of Icon loading when packaging with webpack.

Fixed known styling issues.

## Version 1.0.1 @2022.09.26

**Add**

Hide the other party's microphone icon during a phone call.

**Fix**

Fixed the issue where the SDKAppID input box in the basic demo should be numeric.

## Version 1.0.0 @2022.09.23

Quickly Run Through the TUICallKit Demo

Quick Integration of TUICallKit

TUICallKit API

TUICallKit Customizable Interface Guide

Frequently Asked Questions About TUICallKit (Web)



# Android and iOS

Last updated : 2023-11-29 18:21:52

## Version 2.0.0.750 Released November 3, 2023

### Functionality Enhancement

Android & iOS: The UIKit supports the Japanese language.

Android & iOS: Enhanced the display of call nicknames.

Android & iOS: Adjusted the default ringtone volume from 60% to 100%.

### Defect Rectification

iOS: Corrected the slow image loading issue in Swift version.

iOS: Fixed the issue where the call invitation was automatically cancelled after the caller initiated the call and moved the application to the background.

## Version 1.9.0.680 Released September 27, 2023

### Functionality Enhancement

Android & iOS: Added support for the Arabic language.

Android & iOS: Optimized the package purchase prompts, enabling redirection to corresponding package purchase pages based on the provided links.

Android & iOS: Enhanced the default bitrate at different resolutions to ensure clearer output at higher resolutions. For details, refer to [Resolution](#).

Android & iOS: The default bitrate for video calls has been set to 600kbps, with a beauty level of grade 4.

### Defect Rectification

Android & iOS: Resolved inconsistencies between the rejection prompt when initiating a call to a user on the blacklist and the rejection prompt when sending a private chat message.

iOS: Rectified an anomaly in the video placement for the initiator, which occurred on a 4-person group video call interface when one member declined the call.

iOS: Addressed an issue where the resolution would be reset if the beauty feature was enabled immediately after successful login.

## Version 1.8.0.620 Released August 14, 2023

## Functionality Enhancement

Android & iOS: By default, call messages are excluded from the unread count.

Android: Enhanced the redirection page for floating window permissions on Xiaomi smartphones.

## Defect Rectification

iOS: Remedied an issue where the onKickOffline callback interface became ineffective after being kicked offline.

iOS: Fixed an issue where, after clearing the call on the Missed Call interface, returning to the All Calls interface would result in an empty list.

# Version 1.7.2.570 Released July 20, 2023

## Functionality Enhancement

Android: Gravity sensor is turned off by default, optimizing the call experience on large-screen and customized devices.

## Defect Rectification

Android & iOS: Rectified an issue where, after User A (online) calls User B (offline) and cancels the call, User A calls back User B who logs in thereafter, leading to abnormal cloud call records for user B.

Android: Resolved the crash issue of TUICallKit after upgrading the TRTC SDK version to 11.3.

# Version 1.7.0.460 Released June 25, 2023

## Functionality Enhancement

Android & iOS: Includes UI integration solutions, optimizes sample projects, and improves call setting items.

Android: Reduced the status preservation level during a call to only show standby prompts in the status bar; removed notifications and vibrations.

# Version 1.6.1.410 Released on May 22, 2023

## New features:

Android & iOS: The UI interface [call\(\)](#) and [groupCall\(\)](#) now support custom room ID.

Android & iOS: When initiating a call, a string-type room ID can be passed in, see [CallParams](#) for details.

## Bug fixes:

Android: Fixed issue where an error would occur on the groupCall when generating list parameters using Arrays.asList.

Android: Fixed issue where the video call display was abnormal.

iOS: Fixed issue where it conflicted with TUIRoom component.

iOS: Fixed issue where initiating a call immediately after successful login would cause a crash.

iOS: Fixed issue where the invite page would not appear intermittently when clicking on a notification message to enter the app.

## Version 1.6.0.360 Released on April 27, 2023

### **New features:**

Android: TUICallKit added the Kotlin language version;

iOS: TUICallKit added the Swift language version;

Android & iOS: Added a page to display local call records.

### **Functional optimization:**

Android: Optimized the display of video call avatars.

Android & iOS: In group calls, other group members can be invited to join the call by default.

### **Bug fixes:**

Android: Fixed issues where devices running Android 12 or higher would have no sound after being connected to Bluetooth;

Android: Fixed intermittent issues where the muting setting on the callee side was not effective;

iOS: Fixed intermittent issues where devices could not receive incoming call invitations after relogging in;

iOS: Fixed the issue where the enableCustomViewRoute interface of TUICallKit was not valid;

iOS: Fixed the issue where the nickname was displayed incorrectly on the VoIP push page.

## Version 1.5.1.310 Released on April 17, 2023

### **New features:**

Android & iOS: Added VoIP message push function to provide a better call answering experience.

Android & iOS: Support custom extended fields when initiating a call, see TUICallDefine.CallParams parameter in the [call\(\)](#) method for details.

### **Functional optimization:**

Android & iOS: Optimized offline push capabilities for Huawei, Xiaomi, FCM, and other manufacturers, added manufacturer message categories, and channel ID settings.

**Bug fixes:**

Android & iOS: Fixed issue where the IM custom property was overwritten after initiating a call.

Android: Fixed issue where the totalTime unit in the [onCallEnd](#) callback was incorrect.

## Version 1.5.0.305 Released on March 09, 2023

**Functional optimization:**

Android & iOS: Optimized chat-message display.

Android: The ear-to-screen messaging function is turned off by default now.

Android: Upgraded gradle plugin and version.

Android: Optimized mediaPlayer class, supporting loop playback of ringtones.

**Bug fixes:**

Android & iOS: Fixed issue where the callee would not receive the onCallCancel callback when answering a call fails.

Android: Fixed issue where the caller would receive an exception when the callee fails to answer the call.

Android: Fixed issue where the caller cancels the call during the permission check of the first call and the callee pulls up the interface again.

Android: Fixed the issue where userId was empty when returning network quality to the upper callback.

## Version 1.4.0.255 Released on January 06, 2023

**New features:**

Android & iOS: Support custom call timeout time, see `TUICallDefine.CallParams` parameter in the `call()` method for details.

**Bug fixes:**

Android & iOS: Fixed issue where joining a room actively (`joinInGroupCall`) would result in abnormal termination of the call.

Android: Fixed issue where there were abnormalities with call status when you exited the audio and video call answer interface and came back to the foreground again.

## Version 1.3.0.205 Released on November 30, 2022

**New features:**

Android & iOS: Added beauty setting interface `setBeautyLevel()`, supporting turning off default beauty.

**Functional optimization:**

iOS: Optimized the framework size of `TUICallKit`.

**Bug fixes:**

Android & iOS: Fixed issue where the calling interface did not disappear when the server dissolves a room or kicks out a user.

Android: Fixed issue where if A called offline user B and then cancelled, then A called B again and B came online, the calling interface did not appear.

## Version 1.2.0.153 Released on November 14, 2022

**New features:**

Android & iOS: Support for custom video encoding resolutions.

Android & iOS: Support setting rendering parameters for video: rendering direction and filling mode.

Android & iOS: Support integration of third-party beauty features.

Android & iOS: `TUICallKit` has added overloaded interfaces `call()` and `groupCall()`, supporting custom offline messages (see API documentation for details).

**Functional optimization:**

Android & iOS: Optimized some `TUICallObserver` callback exception issues.

Android & iOS: Optimized the video-to-audio switching function, supporting switching in offline state.

Android & iOS: Improved error codes and error prompts for `TUICallKit`.

iOS: Standardized `TUICallEngine` and `TUICallKit` Swift API names.

**Bug fixes:**

Android & iOS: Fixed issue where you would still receive previously rejected incoming calls after logging back in to your account.

Android: Fixed issue where you would encounter abnormal hang-ups in invites from group chats in one-to-one chats.

Android: Fixed issue where there were abnormalities with multiple-scene exits from live rooms, preventing the initiation of calls.

Android: Fixed issue where contacting person A while B and C were calling each other at the same time would cause C to enter A's room at random.

## Version 1.1.0.103 Released on September 30, 2022

Android & iOS: Optimized the feature of inviting new members to the current group call.

Android & iOS: Optimized the call process to avoid the charging of recording, moderation, and other fees before a call is answered.

Android & iOS: Added support for custom offline notifications.

Android & iOS: Changed the parameters of some **TUICallEngine** APIs. For details, see [call\(\)](#), [groupCall\(\)](#), [inviteUser\(\)](#), and [onCallReceived\(\)](#).

Android & iOS: Fixed occasional callback errors during a group call.

Android & iOS: Fixed the status abnormal issue caused by repeated login or expired `UserSig` .

iOS: Fixed the issue where, when a mixed-language `TUICallKit` project is built with Objective-C and Swift, an error occurs when `init` is called.

Android: Fixed the issue where an error occurs when the floating window feature is integrated for a Kotlin project.

## Version 1.0.0.53 Released on August 15, 2022

### First release:

Android & iOS: Supports one-to-one and group audio/video calls.

Android & iOS: Supports offline call push for mainstream devices on the market.

Android & iOS: Supports custom profile photos and aliases.

Android & iOS: Supports floating call windows.

Android & iOS: Supports custom ringtones.

Android & iOS: Supports receiving calls when the user is logged in on multiple platforms.

# Flutter

Last updated : 2024-01-10 17:10:25

## Version 2.0.3 @2023.12.04

### Bug Fixes :

Android&iOS : Fixed the intermittent issue of incoming call page not displaying sometimes after clicking on an incoming call notification.

## Version 2.0.2 @2023.11.27

### Bug Fixes

Android : Fix the issue of call failure under multiple Flutter Engine conditions.

## Version 2.0.1 @2023.11.15

### Bug Fixes

Android & iOS : Fixed an incompatibility issue caused by Tencent RTC Observer when using Tencent RTC components with other components that also use Tencent RTC.

Android : Fixed an incompatibility issue when using TUIRoom in conjunction with other devices.

## Version 2.0.0 @2023.11.06

### Dependency Description

Upgrade the dependent client SDK version:

Android&iOS TUICore:7.6.5011.

Android&iOS TUICallEngine:2.0.0.750.

## Version 1.9.1 @2023.10.21

**New Features :**

iOS: Support Voip.

**Bug Fixes**

iOS: Fixed an issue where the call page would be abnormal when receiving a call in the background.

## Version 1.9.0 @2023.10.09

**New Features**

Android&iOS: Add an interface for setting ringtones.

**Function Optimization:**

Android & iOS: Optimize package purchasing prompts.

Android & iOS: Optimize default bitrates for different resolutions, [see details](#).

**Bug Fixes**

iOS: Fixed the issue where the same Observer object can be registered twice.

**Dependency Description**

Upgrade the dependent client SDK version:

Android&iOS TUICore:7.5.4852.

Android&iOS TUICallEngine:1.9.0.680.

## Version 1.8.3 @2023.08.25

**Bug Fixes**

Android&iOS: Fixed the problem of no call message display when using tencent\_cloud\_chat\_uikit.

Android&iOS: Fixed the problem of occasionally pulling up the group call page during a single-person call.

Android&iOS: Fixed the problem of occasionally pulling up the call page twice during a call.

Android&iOS: Fixed the problem of abnormal display of call duration.

Function Optimization:

Android: Optimized the problem of failing to pull up the interface in the background when receiving a call.

**Dependency Description :**

Upgrade tencent\_cloud\_uikit\_core to version 1.1.1.



## Version 1.8.2 @2023.08.19

### Bug Fixes

iOS: Fixed the problem of some compilers failing to compile due to the use of deprecated Swift interfaces.

## Version 1.8.1 @2023.08.18

### Bug Fixes :

Android&iOS: Fixed the problem of the video stream of the other party being displayed during a group call voice call.

## Version 1.8.0 @ 2023.08.17

### New Features:

Android&iOS : Build a new TUICalkit based on the Dart language, which makes it easier to customize your own UI style.

Android&iOS : TUICallEngine adds multiple business interfaces such as hangup, accept, and reject.

## Version 1.7.4 @ 2023.07.20

### Function Optimization:

Android : By default, the gravity sensor is turned off to optimize the call experience on large screens and customized devices.

### Bug Fixes:

Android&iOS : A calls B (offline) and then cancels, A calls B again, B logs in and goes online, and B's cloud call record is abnormal.

## Version 1.7.3 @ 2023.07.19

### Function Optimization:

Android: Supports development & debugging using the emulator.

**Dependency Notes:**

The client SDK version that the upgrade depends on: Android LiteAVSDK\_Professional: 11.3.0.13176.

## Version 1.7.2 @ 2023.07.09

**Bug Fixes:**

iOS: Upgrade the client SDK version to fix the problem of AppStore listing failure caused by Non-public API usage.

## Version 1.7.1 @ 2023.07.03

**New Features:**

Android&iOS : Added cloud call records, you can activate the service on the console for experience query.

**Function Optimization:**

Android : Reduce the level of system keep-alive during calls, only display the keep-alive reminder in the status bar, and remove notifications and vibrations.

## Version 1.6.3 @ 2023.06.03

**Bug Fixes:**

iOS: Fix the issue of an empty page when adding people halfway after calling joinInGroupCall.

iOS: Fix the issue of user screen overlap after calling joinInGroupCall.

## Version 1.6.2 @ 2023.05.30

**Bug Fixes:**

Android: Fix the crash issue caused by calling the joinInGroupCall API.

## Version 1.6.1 @ 2023.05.15

**Bug Fixes:**

Android: Fix the occasional crash when applying for floating window permissions on specific Vivo models.

**Dependency Notes:**

Upgrade the dependent client SDK version: Android LiteAVSDK\_Professional: 11.1.0.13111, iOS TXLiteAVSDK\_Professional: 11.1.14143.

## Version 1.6.0 @ 2023.05.03

**New Features:**

Android&iOS: Add hangup interface.

Android&iOS: Add user-defined fields and user-defined call timeout duration.

Android&iOS: Add midway join page for group calls.

**Function Optimization:**

Android: Optimize single-user video call avatar display.

Android&iOS: By default, support inviting other group members to join the call in group calls.

**Bug Fixes:**

Android: Fix the issue of no sound on Android 12 and above devices after connecting to Bluetooth.

Android: Fix the occasional issue of mute settings not taking effect on the called party's side.

iOS: Fix the occasional issue of the device not receiving incoming call invitations after re-login.

iOS: Fix the issue of incorrect nickname display on VoIP push page.

**Dependency Notes:**

Upgrade the dependent client SDK version: Android LiteAVSDK\_Professional: 11.1.0.13111, iOS TXLiteAVSDK\_Professional: 11.1.14143.

## Flutter Version 1.5.4 @ 2023.04.14

**New Features:**

Android&iOS: Add offline push parameters for Xiaomi, Huawei, and VIVO.

iOS: Supports VoIP message push function.

Android&iOS: Add resolution setting function.

**Function Optimization:**

Android: Optimize the unit of the totaltime parameter in the onCallEnd callback to milliseconds.

**Bug Fixes:**

Android&iOS: Fix onCallReceived callback exception issue.

iOS: Fix the issue of incomplete call page display when the screen is rotated.

## Version 1.5.3 @ 2023.03.17

### Bug Fixes:

Android: Fix the packaging failure issue.

Android&iOS: Fix the issue of throwing exceptions when callback methods are not implemented.

## Version 1.5.2 @ 2023.03.13

### Bug Fixes:

Android: Fix the compilation error caused by the API change of TUICallDefine.OfflinePushInfo.

## Version 1.5.1 @ 2023.03.13

### Bug Fixes:

Android&iOS: Fix the issue of incorrect version dependency of tencent\_calls\_engine.

## Version 1.5.0 @ 2023.03.13

### Function Optimization:

Android: Proximity wake-up function is turned off by default.

Android: Upgrade gradle plugin and version.

Android: Optimize the ringtone playback class, supporting loop playback.

### Bug Fixes:

Android&iOS: Fix the issue that the onCallCancel callback is missing when the called party fails to answer the call.

Android: Fix the abnormal problem of the caller when the called party fails to answer the call.

Android: Fix the issue that the interface is pulled up again by the called party when checking permissions for the first call and the calling party cancels the call.

Android: Fix the issue of userId being empty when calling back the network quality to the upper layer.

iOS: Fix the issue of incorrect Observer registration timing in Example.

## Version 1.4.2 @ 2023.02.24

### Bug Fixes:

Android&iOS: Fix the issue of abnormal calls caused by the wrong Observer registration timing in Example.

iOS: Fix the occasional invalid setting issue of removeObserver API.

## Version 1.4.1 @ 2023.02.20

### Bug Fixes:

Android: Fix the issue of the OnCallEnd event being lost after the call ends.

## Version 1.4.0 @ 2023.01.16

### Bug Fixes:

Android&iOS: Fix the issue of abnormal call termination when actively joining a room (joinInGroupCall).

Android: Fix the issue of abnormal call status when entering the background during audio and video call answering and returning to the foreground.

Android: Fix the issue of call initiation failure caused by login status when integrating the tencent\_cloud\_chat\_uikit plugin.

Android: Fix the issue of call initiation failure due to parameter check issues when initiating a group call.

## Version 1.3.1 @ 2022.12.27

### New Features:

Android&iOS: Support custom offline push message during a call.

### Bug Fixes:

Android: Fix the issue of sdkappid is invalid prompt when integrating the tencent\_cloud\_chat\_uikit plugin.

## Version 1.3.0 @ 2022.12.02

### Function Optimization:

iOS: Optimize the size of the TUICallKit Framework.

**Bug Fixes:**

Android&iOS: Fix the issue of the call interface not disappearing when the server-side dissolves the room or kicks out users.

Android: Fix the issue that the call interface does not display when user A calls offline user B, then cancels the call; A calls B again, and B's call interface does not display after going online.

## Version 1.2.2 @ 2022.11.17

**Bug Fixes:**

iOS: Fix the compilation issue caused by static library linking.

## Version 1.2.0 @ 2022.11.17

**New Features:**

Support 1v1 audio and video calls, group audio and video calls.

Support custom avatar and custom nickname.

Support setting custom ringtones.

Support enabling floating window during the call.

Support incoming call service for multi-platform login status.

# FAQs(TUICallKit)

## Web

Last updated : 2024-04-03 17:23:11

### What should I do if I receive the error message "The package you purchased does not support this ability"?

If you encounter the error message above, it's because the Audio and Video Calling Capability Package of your current application has either expired or not been activated. Please refer to [Activate Service](#) to claim or activate the audio and video calling capability, and continue to use the TUICallKit Component.

### How do I procure a package?

Please follow the link [Purchase Official Version](#).

### How can I generate UserSig?

UserSig is a type of security signature designed by Tencent Cloud for its cloud services. It serves as a login credential, derived from the encrypted combination of information such as SDKAppID and SecretKey.

**Method 1:** Accessing from the control panel, refer to [How to Obtain a Temporary UserSig](#).

**Method 2:** Deploying a temporary generation script.

#### Warning:

This approach involves configuring the SecretKey within the front-end code. Regrettably, in this method, the SecretKey can be easily decrypted through reverse engineering. In the event of your key being exposed, attackers can usurp your Tencent Cloud traffic. Therefore, **this method is only suitable for local functional debugging**. For a production environment, please refer to Method 3.

For easier initial debugging, `GenerateTestUserSig-es.js` in the `genTestUserSig(params)` function can be used temporarily to calculate userSig, for instance:



```
import { genTestUserSig } from "@tencentcloud/call-uikit-vue/debug/GenerateTestUserSig"
const { userSig } = genTestUserSig({ userID: "Alice", SDKAppID: 0, SecretKey: "YOUR_SECRET_KEY" })
```

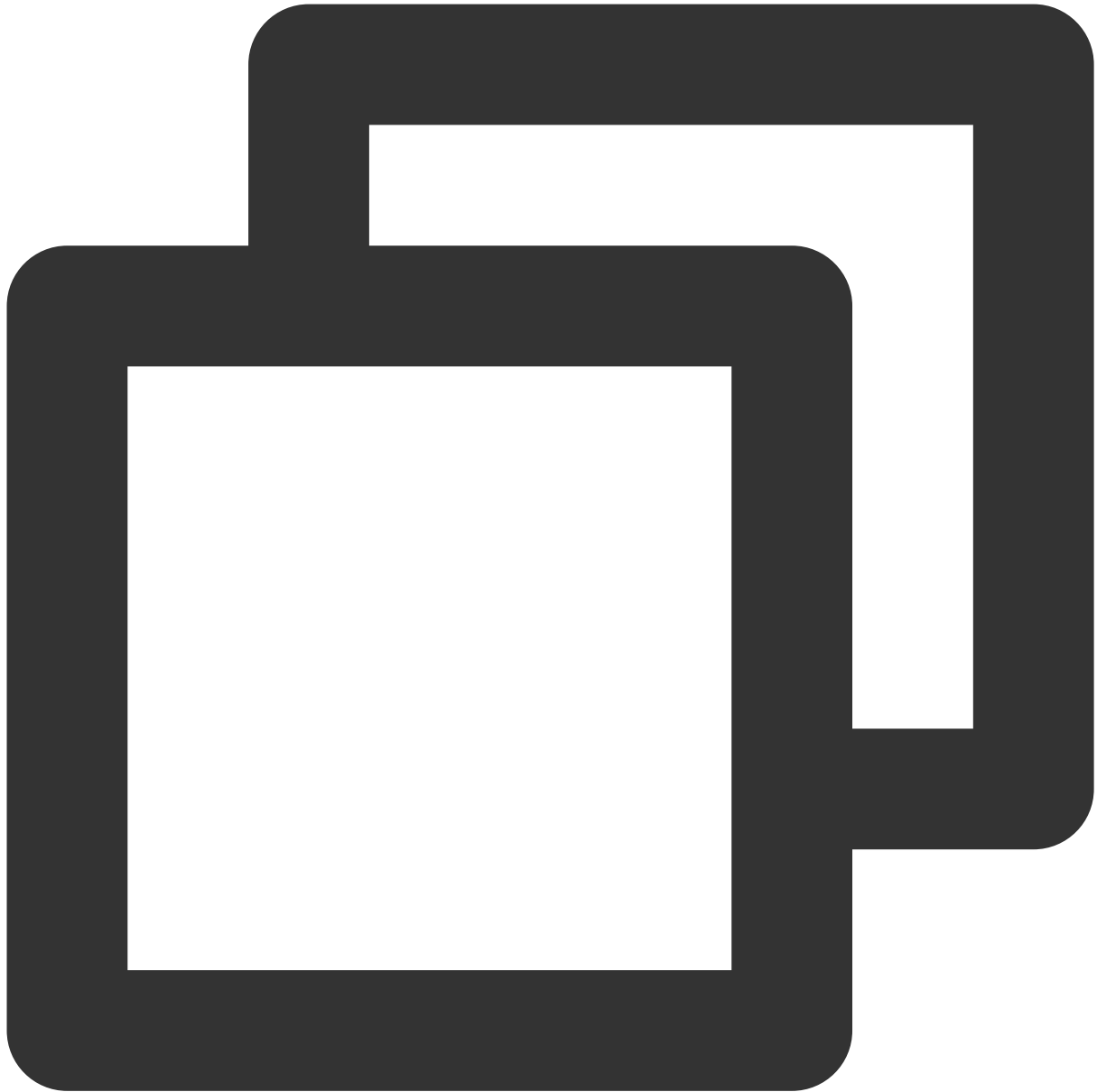
**Method Three:** Use in official environment.

The correct method of issuing UserSig is to integrate the calculation code of UserSig into your server-side, providing project-specific interfaces. When UserSig is needed, your project can launch requests to the business server to obtain dynamic UserSig. For detailed information, please see [Generating UserSig on Server-side](#).

**How is the groupID generated in group calls?**



The generation of groupID requires integration of the [@tencentcloud/chat](#) package. For specifics, refer to the [createGroup API](#); below is an example of the code to generate groupID.



```
import TIM from "@tencentcloud/chat"; // npm i @tencentcloud/chat

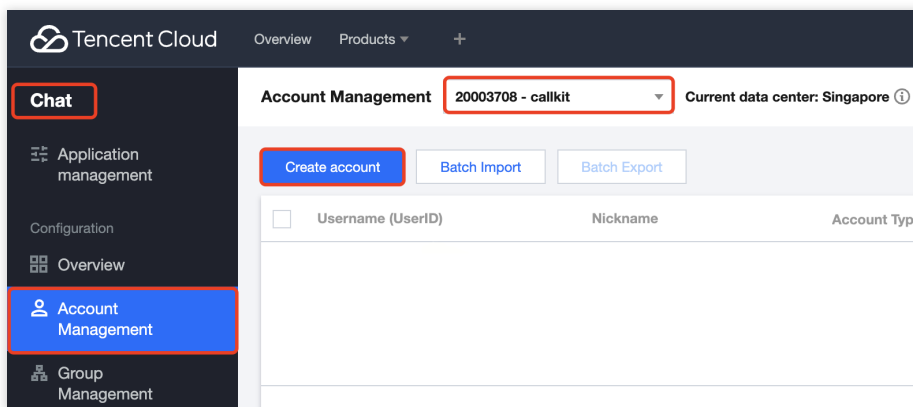
const userIDList: string[] = ['user1', 'user2', 'xxx']; // group member
async function createGroupID() {
  const tim = TIM.create({ SDKAppID });
  const memberList: any[] = userIDList.map(userID => ({ userID: userID }));
  const res = await tim.createGroup({
    type: TIM.TYPES.GRP_PUBLIC, // must be a public group
```

```
name: 'WebSDK',  
memberList  
});  
return res.data.group.groupID;  
}
```

## How can I create a userID?

Signing in once with userID and userSig will automatically create the user.

Create and get through the [Instant Messaging Console](#).



# All Platform

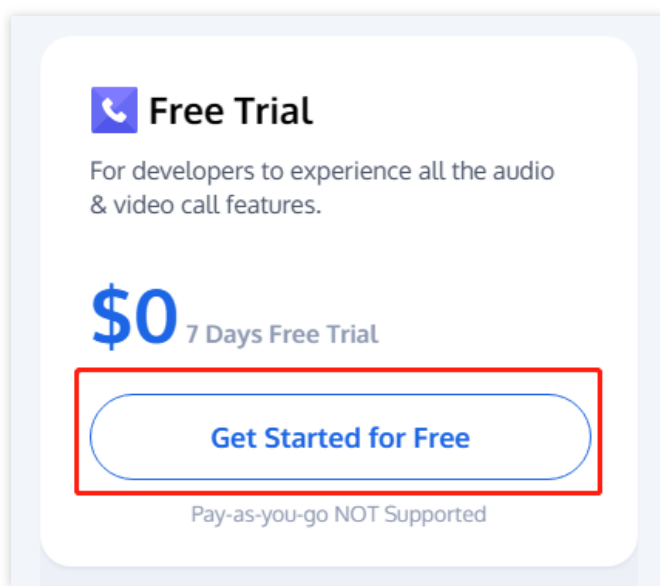
Last updated : 2023-09-19 16:27:46

## Is there an error message indicating that the audio and video call function is not enabled?

You can choose the Free Trial version (7 days trial) or purchase the official version.

### The Free Trail

1. Go to the [price page](#), read the specific services included in the Free Trail package, click **Get Started for Free**, and follow the documentation prompts to enter the console.



2. Read the **Call-Free Trail** package information in the console pop-up window and click **Activate Now** to activate the 7-day free trial of TUICallKit.

**Activate the Call trial version for free**

Application Name

Call\_230908

Call Version

**Call-Free Trail** **7 Days**

Deliver in-app video and voice calling within 30 minutes, [View UI examples](#)

✓ 100 MAU

✓ Voice, Video Call

✓ Call Notification and Offline Push

✓ Call Floating Window

✓ 10,000 Minutes Call Duration

✓ 1-to-1 and Group Call

✓ Customize Call Ringtone

✓ Multi-end Login Calls on the Same Platform

✓ 7 Days Validity

✓ Complete Call UI

✓ Call/Answer/Reject/Hang up

✓ AI Noise Suppression

Partial underlying services of Call are provided by Chat, so this activation will also activate Chat service for you.


Activate Now

**Notice :**

Partial underlying services of Call are provided by Chat, so this activation will also activate Chat service for you.

**The Official Version**

1. Go to the [price page](#), read the specific services included in the Free Trail package, click **Subscribe Now**, and follow the documentation prompts to enter the console.


**1-to-1 Call**

Get all essentials for 1-to-1 call apps and our engineer support.

**\$598** /month

**Subscribe Now**

Pay-as-you-go Supported\*

**Group Call**

Get more inspiring features such as group call, AI noise suppression, and more.

**\$1,298** /month

**Subscribe Now**

Pay-as-you-go Supported\*

©2013-2022 Tencent Cloud. All rights reserved.

Page 726 of 2527

2. Click **Create application** in the console, and select **1-to-1 Call** or **Group Call** in the pop-up window after successful creation to open official service of TUICallKit and become the official version.

✔ Created successfully

You need to choose a version to use Call properly. We recommend you to get the free trial version with all features, or you can directly purchase other versions.

<b>Trial</b> <span>Free</span>	<b>1-to-1 call</b> <span>\$598 per month</span>	<b>Group Call</b> <span>\$1298 per month</span>
<ul style="list-style-type: none"><li>✓ 7 days validity, gifted call duration 10,000 minutes/month</li><li>✓ Experience all features of <b>Audio/Video call</b></li></ul>	<ul style="list-style-type: none"><li>✓ Gifted call duration 200,000 minutes/month</li><li>✓ Complete call UI, supporting 1-to-1 video call, audio call, and other functions</li></ul>	<ul style="list-style-type: none"><li>✓ Gifted call duration 600,000 minutes/month</li><li>✓ Support group video/audio calls, multi-end login calls on the same platform, and other functions</li></ul>

Buy

## The error inviteID is invalid or the invitation has been processed

The inviteID has been cancelled by the calling party or the invitation has been accepted. Because signaling is affected by many factors such as network bandwidth, in extreme cases (repeated calls on the accept and reject interfaces), an error that inviteID is invalid may result.

## The message sender or receiver UserID is invalid or does not exist

The user you are calling has not logged in to the IM service, so the call fails. Please try to call a user who has logged in to the IM service.

## Can TUICallKit use TRTC without introducing IM SDK?

**No**, all components of TUIKit use Tencent Cloud IM SDK as the basic service for communication, such as call dialing signaling, call busy line signaling and other core logic, if you have purchased other IM products, you can also refer to TUICallKit logic to adapt.

## How to buy audio and video call plan?

---

Please refer to the purchase link [TRTC Call Monthly Packages](#), if you have other questions, please click the top of the page Talk to us for pre-sale package consultation.

## How do I get the Roomid of the call?

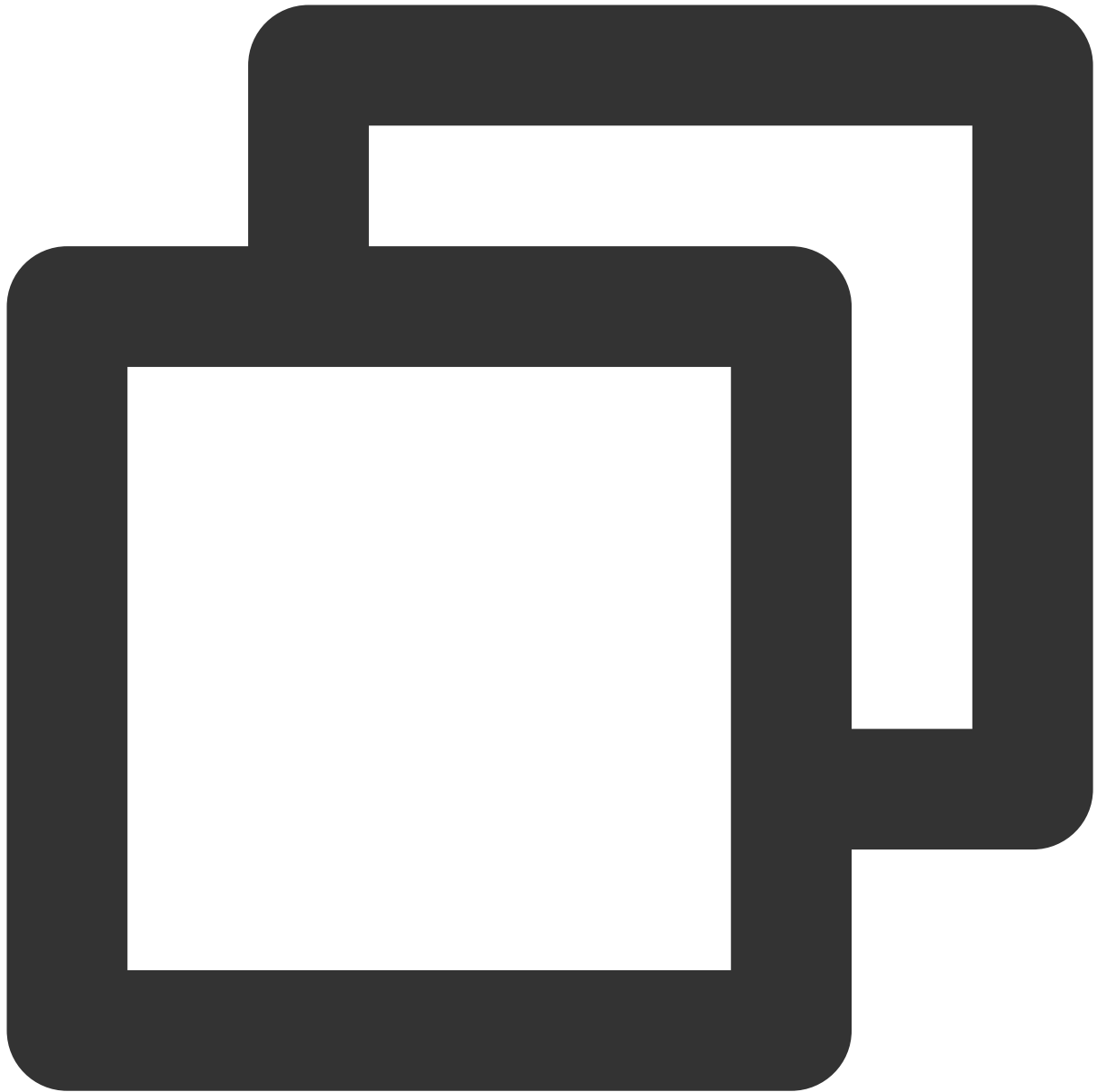
After the call is connected, you can use [onCallBegin](#) to return the roomid field to obtain the information.

# Flutter

Last updated : 2023-09-19 16:28:54

1. Integrate `tencent_calls_uikit` and `tencent_trtc_cloud` at the same time, or integrate `tencent_calls_uikit` and `live_flutter_plugin` at the same time, and the symbol conflict will occur, how to solve it?

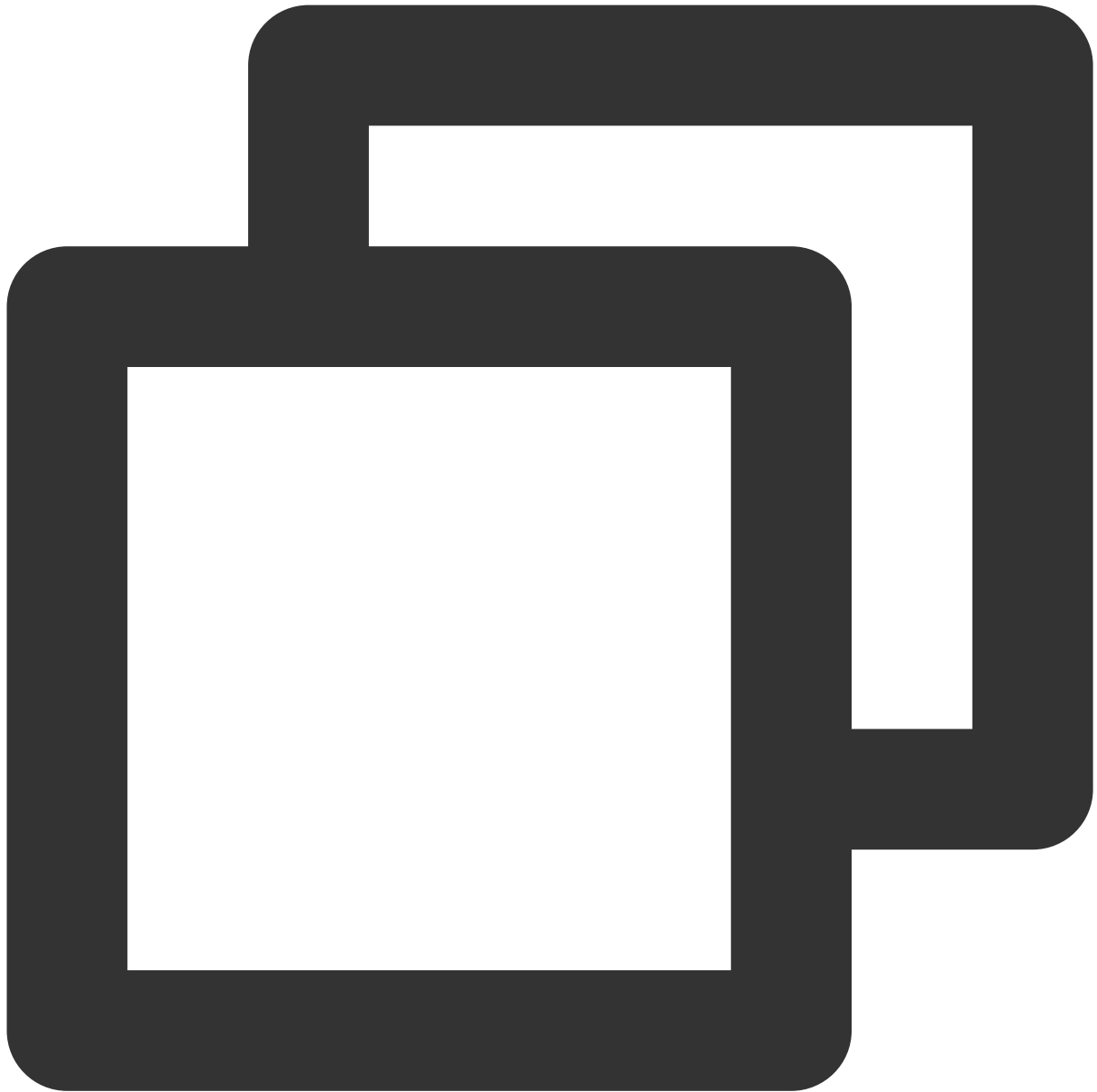
Details: When introducing flutter "tencent\_calls\_uikit" into our existing project, Android builds APK with the following error:



```
Duplicate class com.tencent.liteav.LiveSettingJni found in modules jetified-LiteAVS  
(com.tencent.liteav:LiteAVSDK_Professional:10.7.0.13053) and jetified-LiteAVSDK_TRT  
(com.tencent.liteav:LiteAVSDK_TRTC:10.3.0.11225)
```

The following error occurs when `pod install` is executed on iOS:







```
[!] The 'Pods-Runner' target has frameworks with conflicting names: txsoundtouch.xc
```

The reason for this problem is that `tencent_calls_uikit` and `tencent_trtc_cloud` rely on the standard and premium versions of TRTC Android SDK respectively (this problem will be gradually unified in subsequent versions). Currently, this problem can be fixed through local integration. The specific steps are as follows:

1. Go to the version management page of [tencent\\_calls\\_uikit](#) and [tencent\\_calls\\_engine](#) respectively (`tencent_calls_uikit` depends on `tencent_calls_engine`), click the download button, wait for the download to complete, and extract the file.





**tencent\_calls\_uikit 1.8.3** 


Published in the last hour •  mediaservices.com.cn Dart 3 compatible


[SDK](#) [FLUTTER](#) [PLATFORM](#) [ANDROID](#) [IOS](#) 👍 29

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

### Stable versions of tencent\_calls\_uikit

Version		Min Dart SDK	Uploaded	
<a href="#">1.8.3</a>	<span>Null safety</span>	2.12	in the last hour	 
<a href="#">1.8.2</a>	<span>Null safety</span>	2.12	18 days ago	
<a href="#">1.8.1</a>	<span>Null safety</span>	2.12	19 days ago	





**tencent\_calls\_engine 1.8.1** 

Published in the last hour •  mediaservices.com.cn Dart 3 compatible

[SDK](#) [FLUTTER](#) [PLATFORM](#) [ANDROID](#) [IOS](#) 👍 2

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

### Stable versions of tencent\_calls\_engine

Version		Min Dart SDK	Uploaded	
<a href="#">1.8.1</a>	<span>Null safety</span>	2.12	in the last hour	 
<a href="#">1.8.0</a>	<span>Null safety</span>	2.12	19 days ago	
<a href="#">1.7.4</a>	<span>Null safety</span>	2.12	46 days ago	

2. Change the TRTC SDK dependency in the [tencent\\_calls\\_engine](#) plug-in to the professional version.

Android: Go to the directory decompressed by `tencent_calls_engine` and change the dependency in the `android/build.gradle` file to the one displayed in the error message:  
`com.tencent.liteav:LiteAVSDK_Professional:10.7.0.13053` .

```

dependencies{
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    api 'com.google.code.gson:gson:2.9.1'
    api "com.tencent.liteav.tuikit:tuicore:1.1.16"
    api 'com.tencent.imsdk:imsdk-plus:6.8.3374'
    api "com.tencent.liteav:LiteAVSDK_TRTC:10.3.0.11225"
    api 'com.tencent.liteav.tuikit:tuicallengine:1.3.0.205'
}

```

iOS: Go to the directory where `tencent_calls_engine` is decompressed and change the dependencies in the `ios/tencent_calls_engine.podspec` file to: `s.dependency`  
`'TUICallEngine/Professional', '1.5.1.310' .`

```

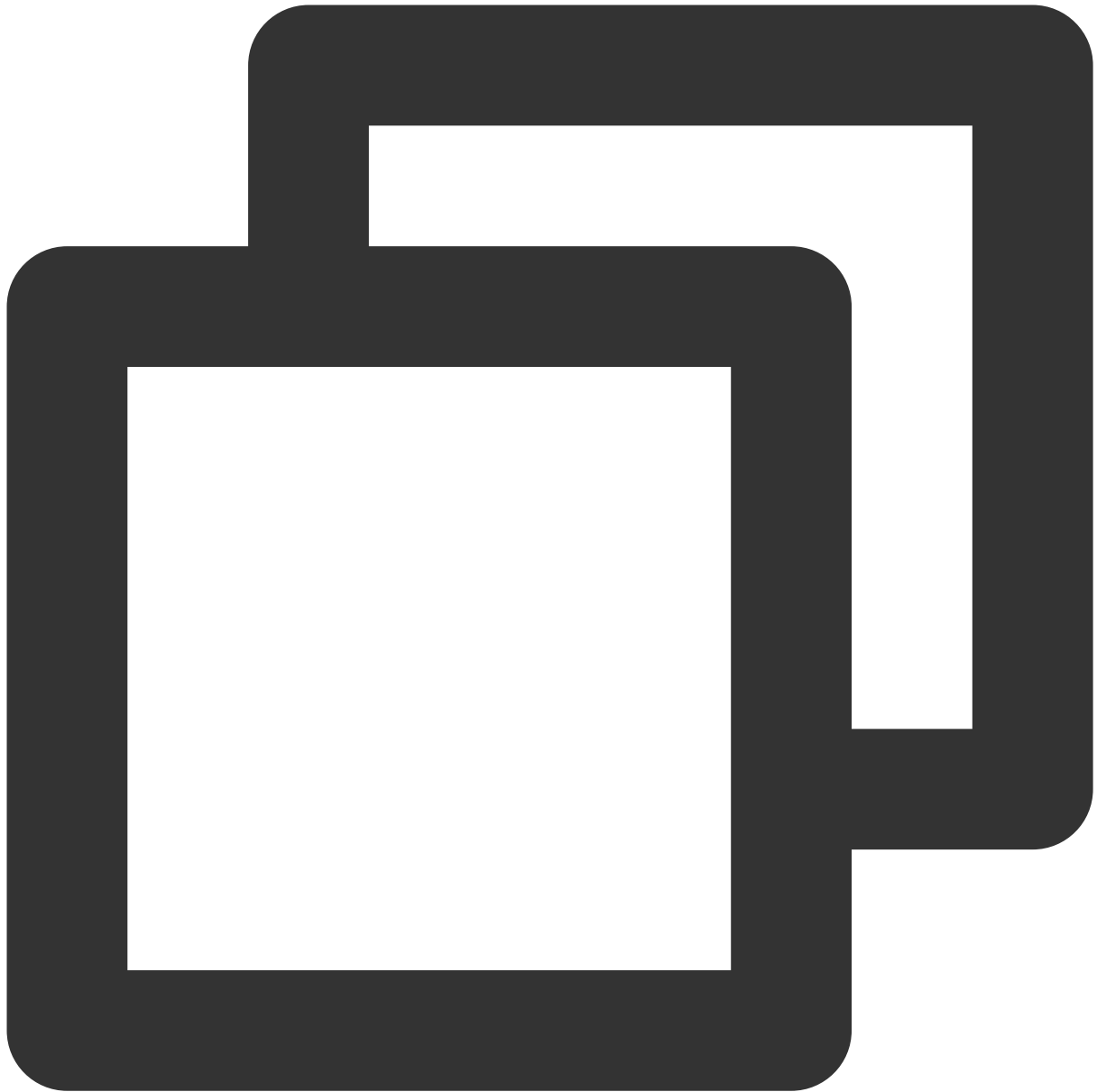
A new Flutter plugin project.
    DESC
s.homepage      = 'http://example.com'
s.license       = { :file => '../LICENSE' }
s.author        = { 'Your Company' => 'email@example.com' }
s.source        = { :path => '.' }
s.source_files  = 'Classes/**/*'
s.dependency    'Flutter'
s.dependency    'TUICallEngine/Professional', '1.5.1.310'
s.platform      = :ios, '9.0'
s.static_framework = true
# Flutter.framework does not contain a i386 slice.
s.pod_target_xcconfig = { 'DEFINES_MODULE' => 'YES', 'EXCLUDED_ARCHS[sdk=iphonesimulator]' => 'i386' }
s.swift_version  = '5.0'

```

Then go to the directory extracted by `tencent_calls_uikit` and change the dependencies in the `ios/Tencent_calls_Uikit.podspec` file to: `s.dependency 'TUICallKit/Professional',`  
`'1.5.1.310' .`

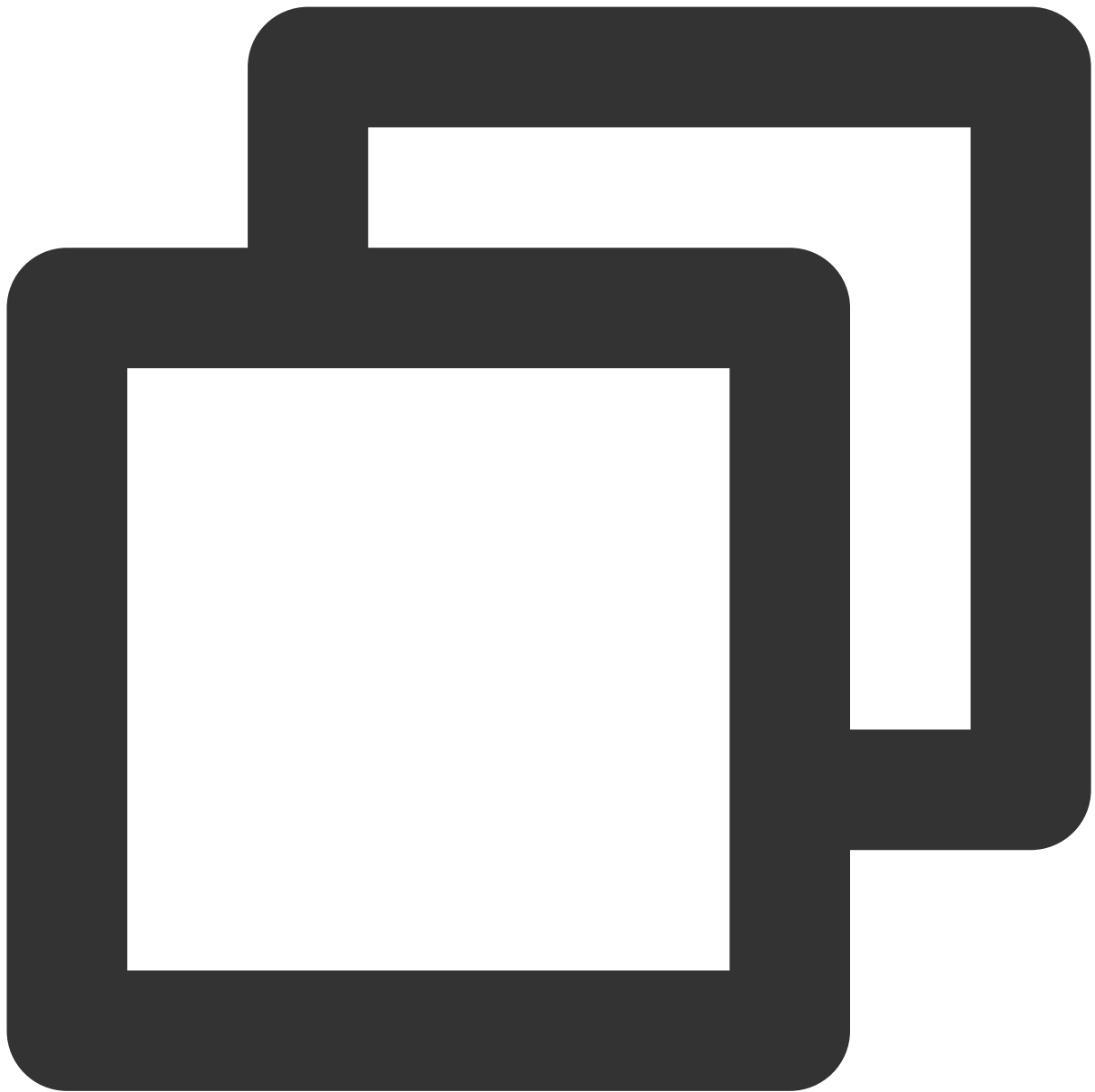
```
A new Flutter plugin project.
      DESC
s.homepage      = 'http://example.com'
s.license       = { :file => '../LICENSE' }
s.author        = { 'Your Company' => 'email@example.com' }
s.source        = { :path => '.' }
s.source_files  = 'Classes/**/*'
s.dependency    'Flutter'
s.dependency    'TUICallKit/Professional' '1.5.1.310'
s.platform      = :ios, '9.0'
s.static_framework = true
# Flutter.framework does not contain a i386 slice.
s.pod_target_xcconfig = { 'DEFINES_MODULE' => 'YES', 'EXCLUDED_ARCHS[sdk=iphonesimulator]' => 'i386' }
s.swift_version  = '5.0'
end
```

3. Modify `pubspec.yaml` in the `tencent_calls_uikit` project to adjust the `tencent_calls_engine` dependency to a local dependency as shown in the following example:



```
tencent_calls_engine:  
  path: tencent_calls_engine/ # The directory here corresponds to the plug-in di
```

4. Modify `pubspec.yaml` in your project to change the `tencent_calls_uikit` dependency to a local dependency as shown in the following example:

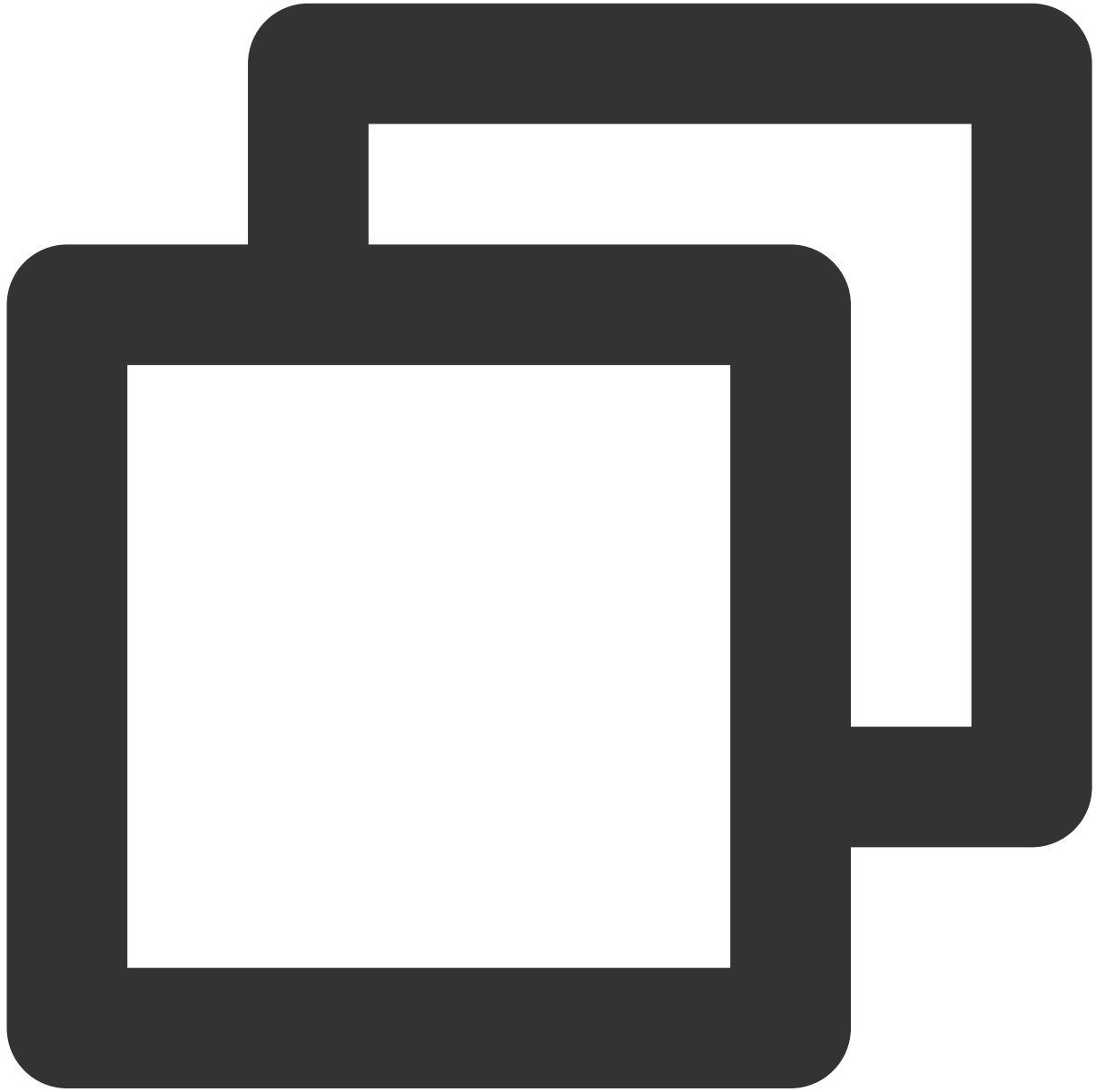


```
tencent_calls_uikit:  
  path: tencent_calls_uikit/ # The directory here corresponds to the plug-in dir
```

## 2. Flutter Android does not add confusion Settings, how to set?

If you need to compile and run on the Android platform, because we use the reflection feature of Java inside the SDK, we need to add some classes in the SDK to the list of non-obviation, so you need to add the following code in the

proguard-rules.pro file:

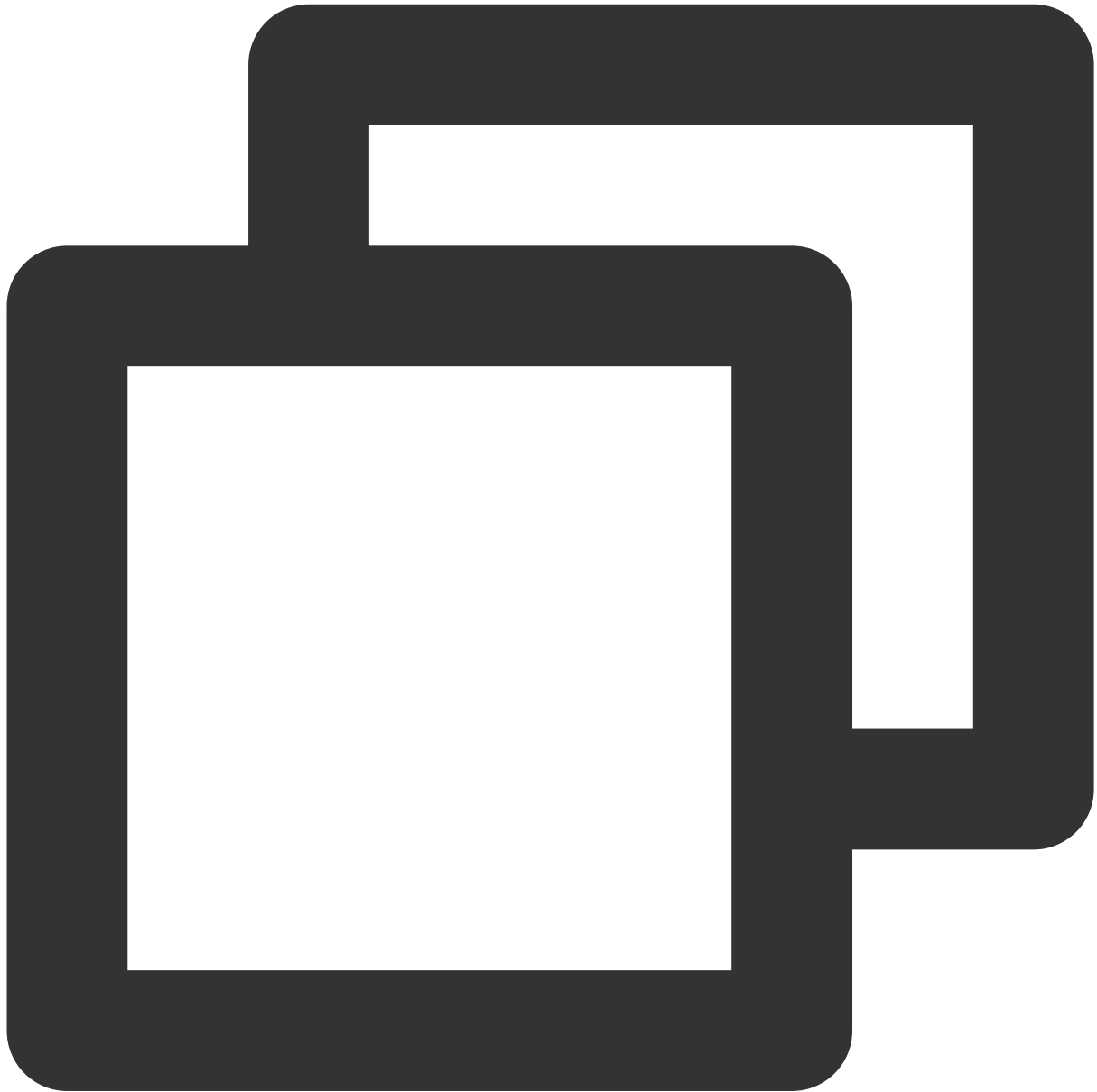


```
-keep class com.tencent.** { *; }
```

### 3. How can I fix a compilation error or page failure caused by an upgrade from a version earlier than 1.8.0 to 1.8.0 or later?

If you are upgrading from 1.8.0 or below to 1.8.0 or above, you need to check that the following steps are normal:

1. Add `navigatorObservers` to `MaterialApp`. The purpose is to navigate to the `TUICallKit` page when you receive a call invitation. Example code is as follows:

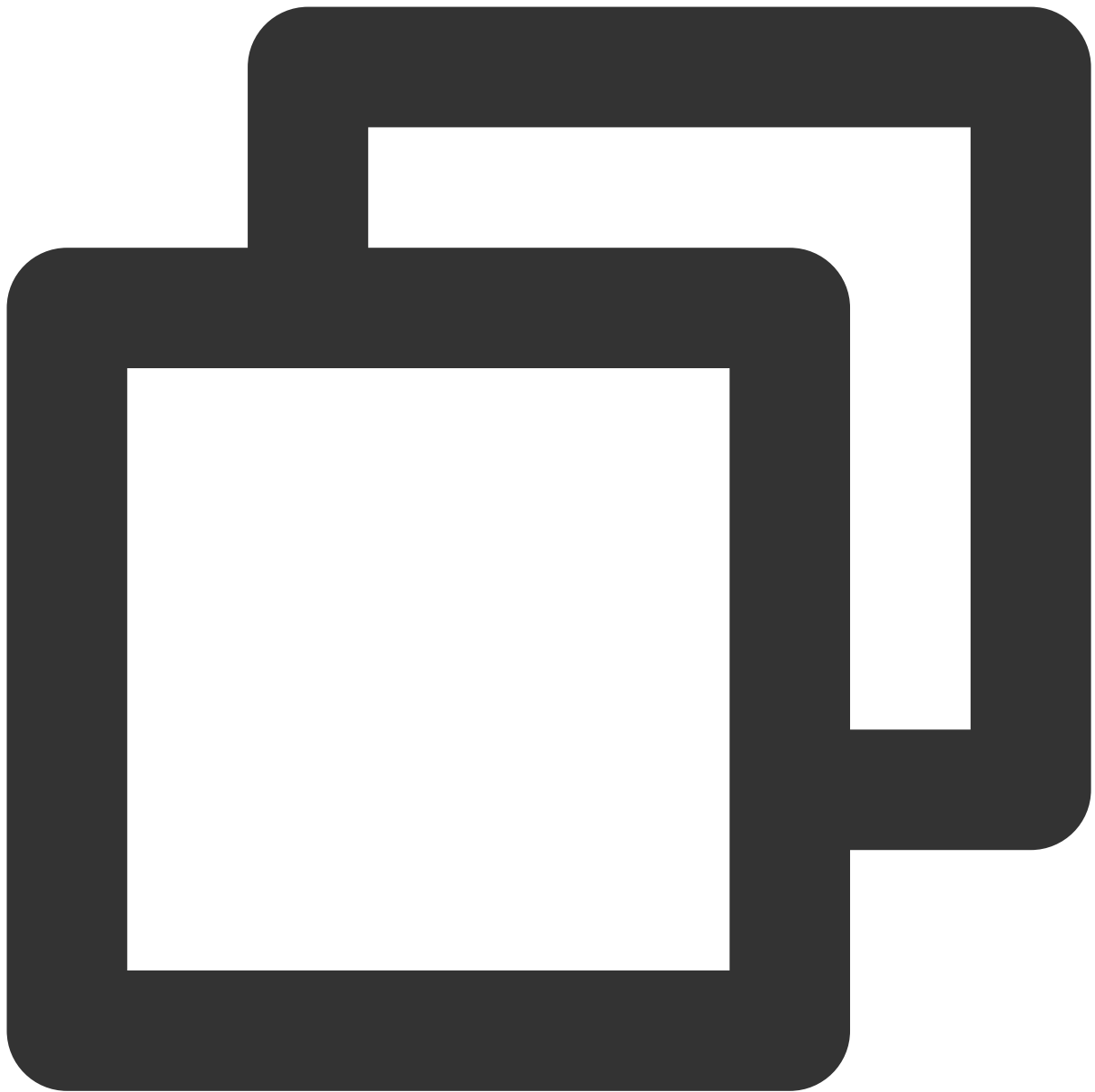


```
import 'package:tencent_calls_uikit/tuicall_kit.dart';

MaterialApp (
  navigatorObservers : [TUICallKit.navigatorObserver],
  ...
)
```

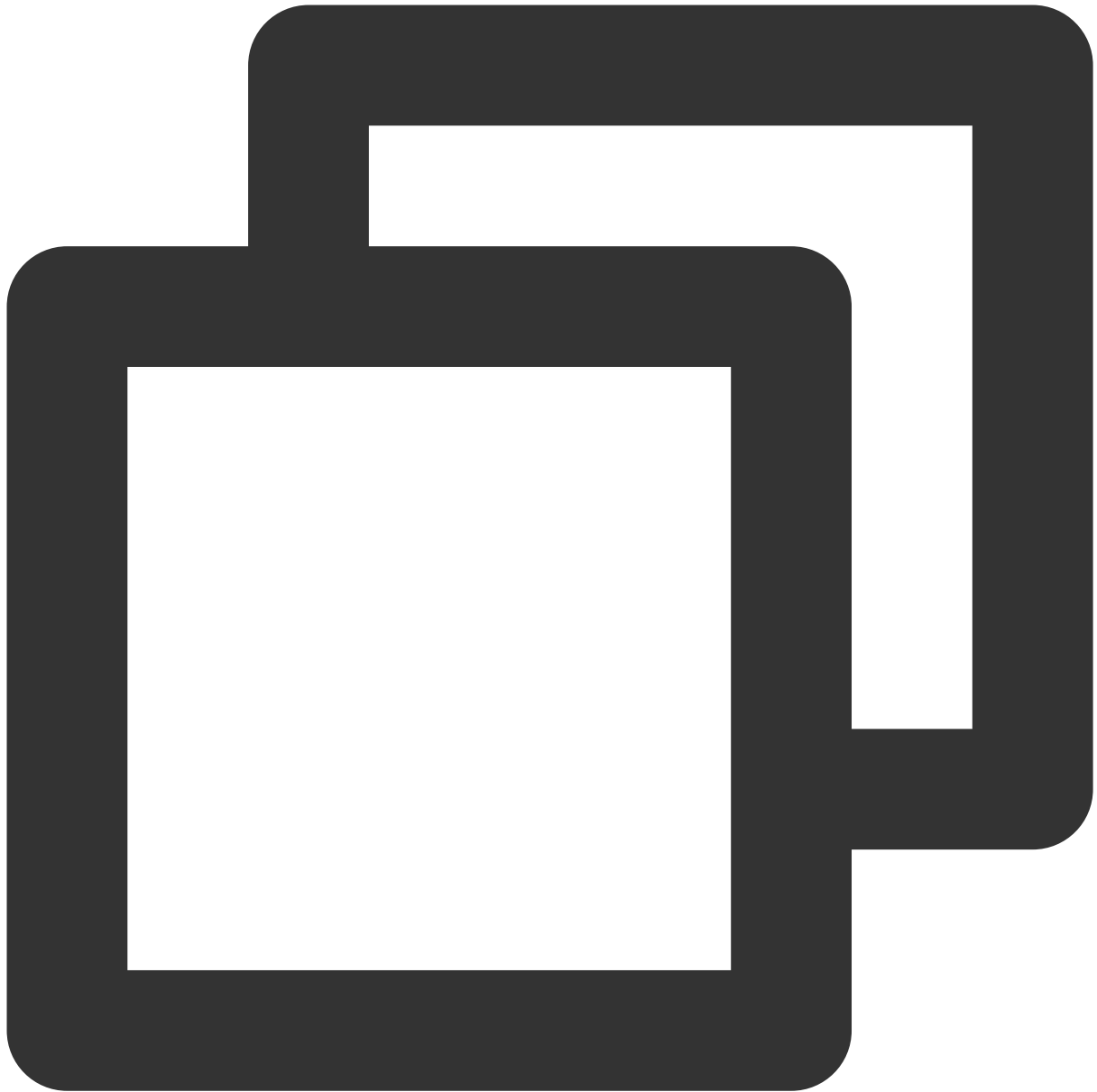
2. `tencent_calls_engine` Import files in plug-ins are uniformly replaced with new ones.





```
import 'package:tencent_calls_engine/tuicall_engine.dart';  
import 'package:tencent_calls_engine/tuicall_observer.dart';  
import 'package:tencent_calls_engine/tuicall_define.dart';
```

The above code block content is replaced with the following:



```
import 'package:tencent_calls_engine/tencent_calls_engine.dart';
```

3. The login API adjustment is more standardized and no parameters need to be specified.

```
return await callsUIKitPlugin.login(  
  sdkAppId: GenerateTestUserSig.sdkAppId,  
  userId: userInfo.userId,  
  userSig: GenerateTestUserSig.genTestSig(userInfo.userId));
```

161	164	return await callsUIKitF
162	165	userInfo.userId, Ger
163	166	}
164	167	For versions earlier than 1.8

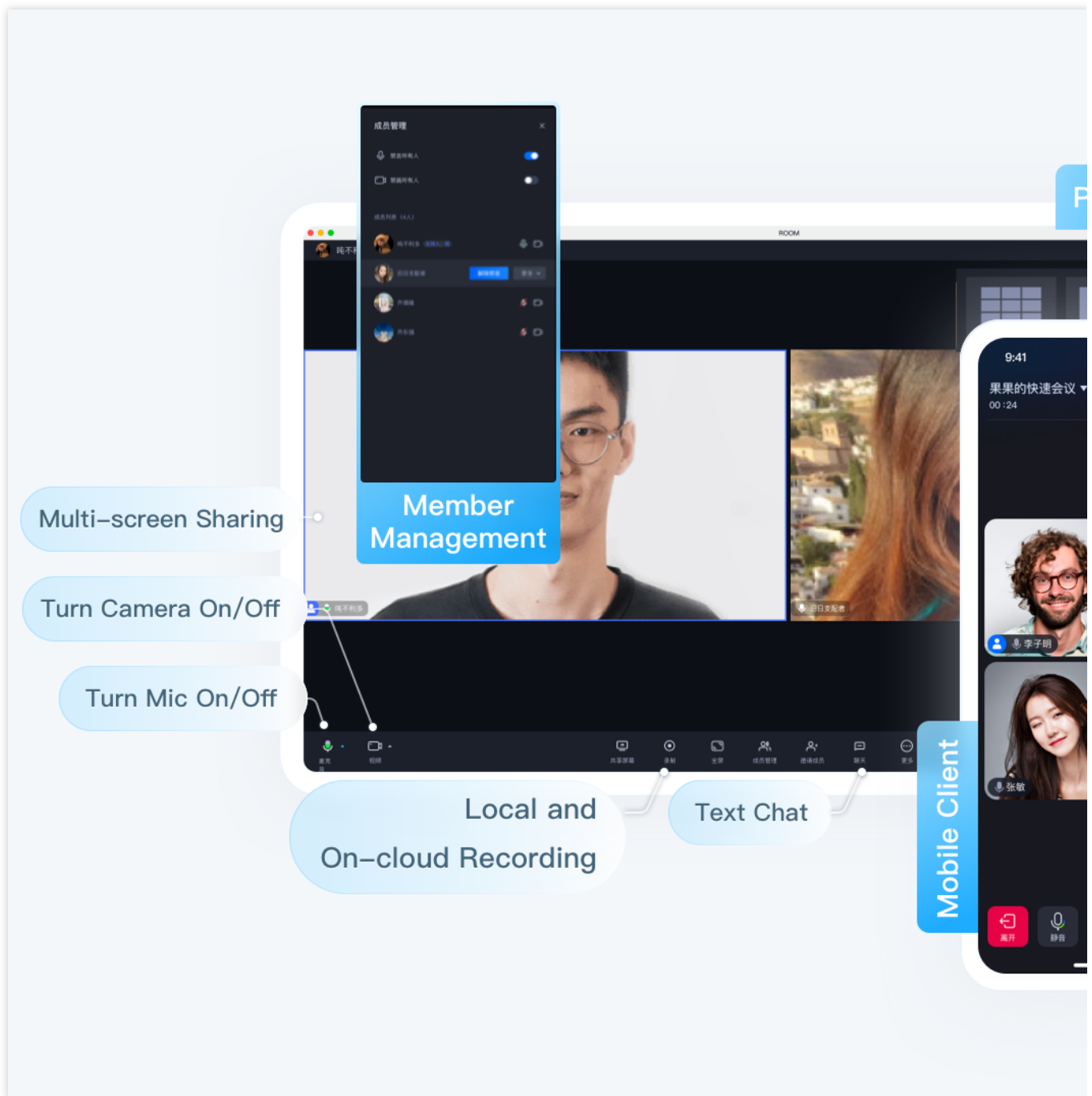
4. Optimization of offline push parameter construction.

# Audio/Video Conference Overview (TUIRoomKit)

Last updated : 2024-04-12 11:19:23

## Overview

Conference (TUIRoomKit) is a product suitable for multi-person audio and video conversation scenarios such as business meetings, webinars, and online education. By integrating this product, you can add room management, member management, screen sharing, and other functions to your App in just three steps within a day, quickly launching your business. The basic function display is as shown below:

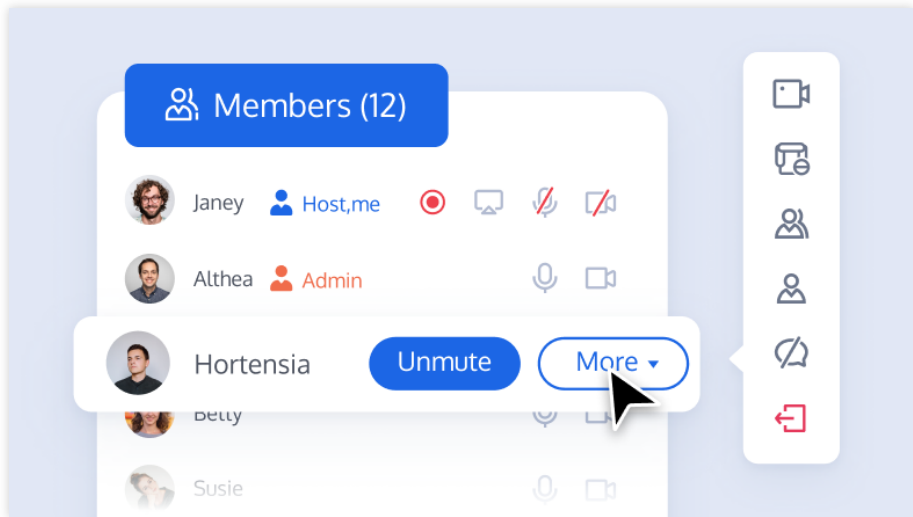
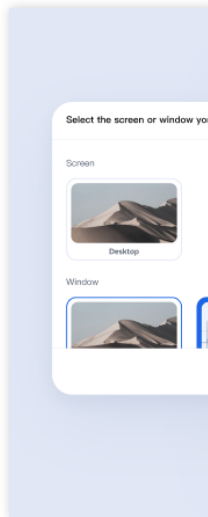


## Supported Platform

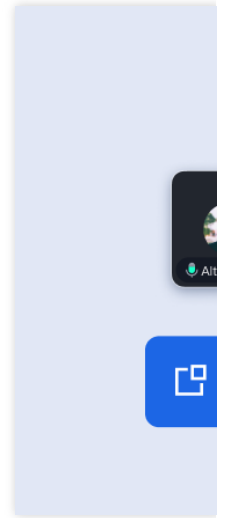
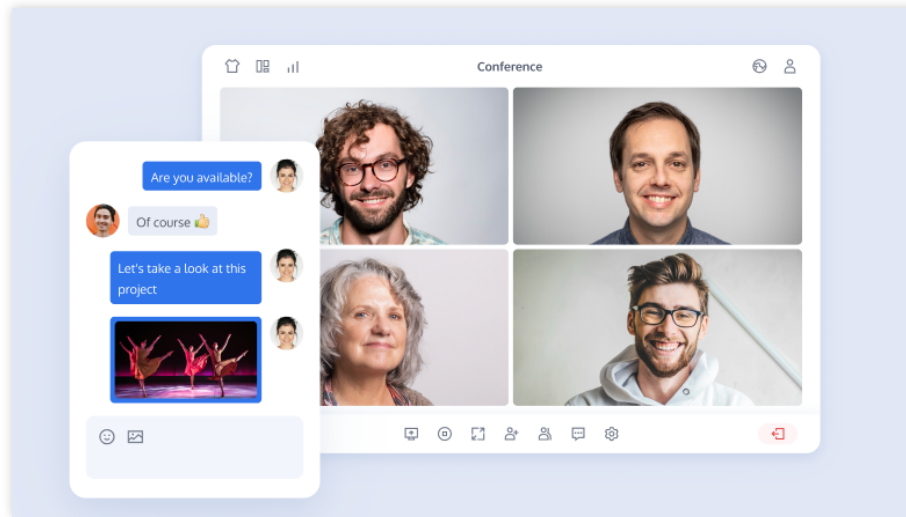
Platform	Android	iOS	Web	Flutter	Electron
Supported					
Supported Languages/Frameworks	Java	Swift	Vue3 Vue2	Dart	Vue3 Vue2

## Description of the Feature

Basic Feature	Advanced Feature	Feature Advantages
Multi-person video conversation Rich meeting control functions Support for on-stage speaking mode Enable/Disable Floating Window Audio and video settings Screen sharing	In-meeting chat On-cloud recording AI Noise Reduction	<b>Quick integration</b> <b>Comprehensive UI Interaction</b> <b>Cross-platform Interconnection Support</b> <b>Multi-device Login Support</b>

Member Management	Screen Sharing
	

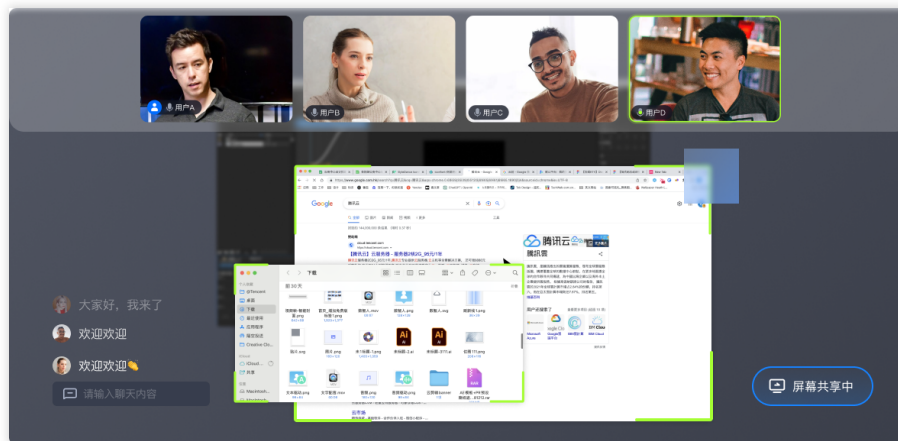
In-meeting Chat	Floating Window



## Applicable Scenario

Multi-person audio and video conversation interaction, covering business meetings, webinars, online education, online recruitment, and audio chat rooms.

### Audio/Video Conference



### Online Classroom



### Webinar

### Corporate Training



## Trying It Online

Platform	Web	Android	iOS	Others
Online Experience				/
Demo Integration	<a href="#">Github: web</a>	<a href="#">Github: Andorid</a>	<a href="#">Github: iOS</a>	<a href="#">Github</a>

## Exchange and Feedback

If you have any requirements or feedback, you can contact: [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).



# Activate the Service (TUIRoomKit)

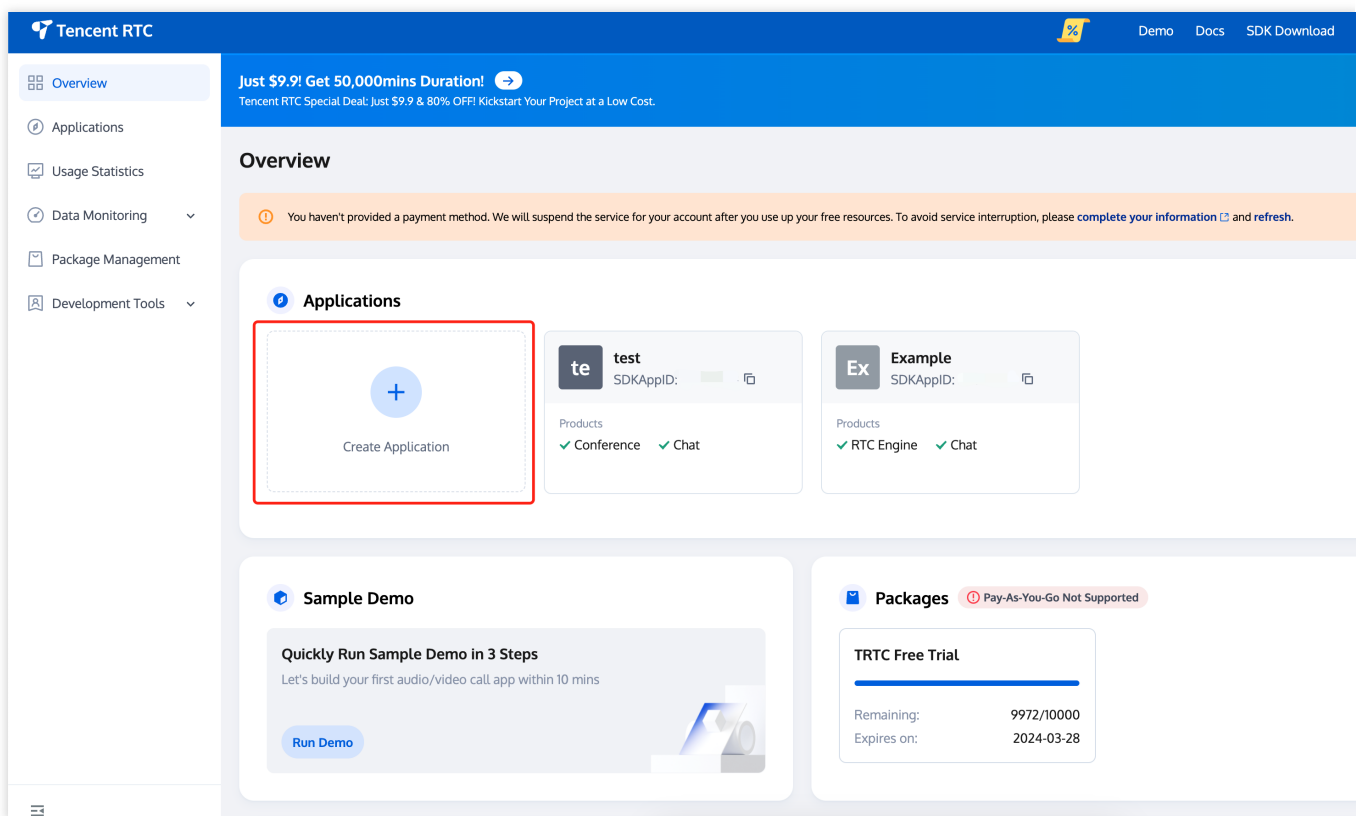
Last updated : 2024-04-12 11:40:48

## Activate and purchase guide

### Free trial

You can follow the steps below to activate the TRTC Conference product service and receive a free 14-day trial version.

1. Visit [TRTC Console > Applications](#), select **Create Application**.





2. In the Create application pop-up, select **Conference** and enter the application name, click **Create**.


### Create application


×

Select product Part of the underlying technology is supported by Chat, so we will enable it when you get started

**Call** UI Ex.  
Add high-quality video and voice calling quickly and easily

**Conference** UI Ex.  
Incorporate meetings for unlimited audience to collaborate together

**Live** UI Ex.  
Integrate live streaming for an engaging interactive experience

**RTC Engine**  
Integrate calling and interactive live streaming features

Application name

Conference


Create

3. After completing the application creation, you will default entry to the application details page, select the **Free Trail** in the floating window, and click to **Get started for free**.

### Select an Conference edition


×

You need to choose a version to use Conference properly. We recommend you to get the free trial version with all features, or you can directly purchase other versions.

**Free Trial**


- ✓ 14 days validity, gifted conference duration 10,000 minutes/month
- ✓ 100 Participants Per Room
- ✓ 10 Active Rooms
- ✓ Experience all features of conference

**\$0**

**Lite**


- ✓ Gifted conference duration 100,000 minutes/month
- ✓ 20 Participants Per Room
- ✓ 20 Active Rooms
- ✓ Basic functions

**\$299 /mo.**

**Standard**

- ✓ Gifted conference duration 300,000 minutes/month
- ✓ 50 Participants Per Room
- ✓ 100 Active Rooms
- ✓ On-stage speech mode, chat in conference and more interactive meeting features

**\$599 /mo.**

**Pro**

- ✓ Gifted conference duration 450,000 minutes/month
- ✓ 200 Participants Per Room
- ✓ 200 Active Rooms
- ✓ All features of conference

**\$899 /mo.**

Get started for free

4. After the activation is completed, you can view the edition information on the current page. The `SDKAppID` and `SDKSecretKey` here will be used in the integration guide.

The screenshot displays the 'Application Overview' page for a 'Conference' application. The left sidebar contains navigation links: 'All Applications', 'Application Overview' (selected), 'Advanced Features', 'Call', 'Conference', 'Live', 'RTC Engine', 'Chat', and 'In-game Voice Chat'. The main content area is divided into sections. The 'Basic Information' section shows the application name 'Conference', SDKAppID (highlighted with a red box), SDKSecretKey (highlighted with a red box), creation time '2024-03-11 21:08:42', region 'Global (excluding Chinese mainland)', and status 'Enabled'. The 'Products' section has a link to 'Quickly run sample demo in 3 steps >'. Below this, there are two product cards: 'Conference' and 'Chat'. The 'Conference' card shows the edition 'Conference : Trial >' (highlighted with a red box), expiration time '2024-03-25', and an auto-renewable status. The 'Chat' card shows the edition 'Chat : Development', expiration time '2024-04-11', and an auto-renewable status. Both product cards have 'Buy package' and 'Integrate' buttons.

Basic Information	
Application name	Conference
SDKAppID	[Redacted]
SDKSecretKey	*****
Description	--
Tags	No tags yet
Status	Enabled
Creation time	2024-03-11 21:08:42
Region	Global (excluding Chinese mainland)

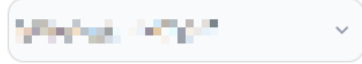
Products	
Quickly run sample demo in 3 steps >	
<b>Conference</b>	
Edition	Conference : Trial >
Expiration time	2024-03-25
Auto-renewable	--
<a href="#">Buy package</a> <a href="#">Integrate</a>	
<b>Chat</b>	
Edition	Chat : Development
Expiration time	2024-04-11
Auto-renewable	--
<a href="#">Buy package</a>	

## Purchase the official editions

1. Visit the [TRTC Conference Purchase page](#), select the Application (SDKAppID) and Conference Package you want to purchase. At the same time, **we recommend you to enable Auto-renewal to avoid affecting business use.** After Confirming purchase information and agreeing to the relevant agreement, check the agreement content and click **Subscribe now.**

## Conference Monthly Packages

Application (SDKAppID) ⓘ



[+ Create Application](#)

ⓘ Please select the correct SDKAppID, as it cannot be modified after purchase.

Package editions [Detail](#)

### Lite

- 20 participants per room
- 20 Active Rooms
- Minimize features
- 100,000 FREE Minutes Included

**\$299/mo.**

### Standard

- 50 participants per room
- 100 Active Rooms
- Advanced Features
- 300,000 FREE Minutes Included

**\$599/mo.**

**Most Popular**

### Pro

- 200 participants per room
- 200 Active Rooms
- All features included
- 450,000 FREE Minutes Included

**\$899/mo.**



**Automatic renewal**

Automatically renew monthly after expiration. You can cancel at any time, please feel free to check it.

☐ Data Monitoring [Detail](#)

Available for all application (SDKAppID). Providing comprehensive quality troubleshooting and real-time Quality Monitoring services, assisting you in quickly understanding the quality of your conference.

☒ I have read and agree to [TRTC Service Level Agreement](#)

2. Go to the order confirmation page to confirm product information.

## Please confirm the following product information [Go Back to Modify Configuration](#)

### Product List

sp\_rav\_conference

899.00 USD

Monthly package: TRTC Conference Pro

Unit Price: 899.00USD/month

Quantity: 1

Payment Mode: Prepaid

Term: 1month

### Discounts and Vouchers

☐ Use promo voucher

No available Promo voucher

Che

sp\_rav\_

Upfront

Tax:

Total

3. Go to the payment page to complete the payment. After the purchase is completed, you can go to the [TRTC Console](#) to view application edition information, and refer to the [Integration guide](#) for integration.

## Renew the official edition

Refer to the [Purchasing the official edition](#) and purchase the same edition of the Conference monthly package again to complete the renewal. It is recommended that you renew Conference by enabling auto-renewal. **When the account balance is sufficient, it will automatically renew monthly after expiration.**

## Conference Monthly Packages

Application (SDKAppID) ⓘ



[+ Create Application](#)

ⓘ Please select the correct SDKAppID, as it cannot be modified after purchase.

Package editions [Detail](#)

### Lite

- 20 participants per room
- 20 Active Rooms
- Minimize features
- 100,000 FREE Minutes Included

**\$299/mo.**

### Standard

- 50 participants per room
- 50 Active Rooms
- Advanced Features
- 300,000 FREE Minutes Included

**\$599/mo.**

**Most Popular**

### Pro

- 200 participants per room
- 200 Active Rooms
- All Features Included
- 400,000 FREE Minutes Included

**\$899/mo.**

☒ Automatic renewal

Automatically renew monthly after expiration. You can cancel at any time, please feel free to check it.

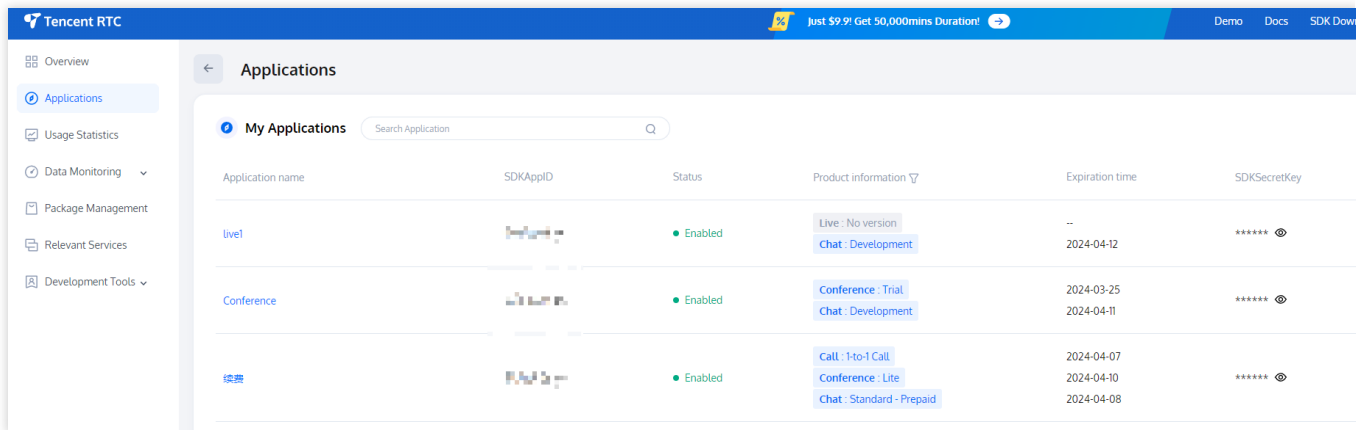
☐ Data Monitoring [Detail](#)

Available for all application (SDKAppID). Providing comprehensive quality troubleshooting and real-time Quality Monitoring services, assisting you in quickly understanding the quality of your application.

☒ I have read and agree to [TRTC Service Level Agreement](#)

The specific steps to enable auto-renewal in the console are as follows:

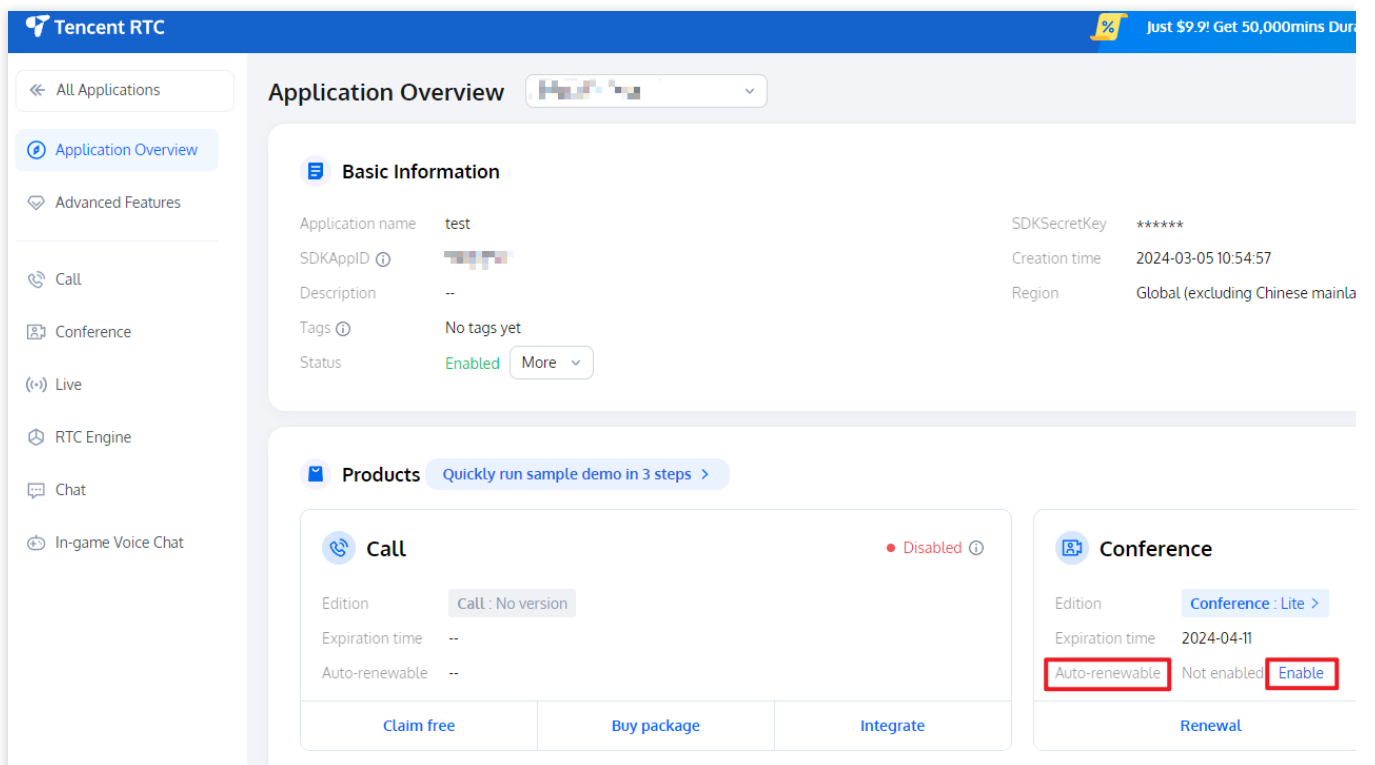
1. Access the [TRTC Console > Applications](#), find the application and click **manage** to enter the application details page.



The screenshot shows the 'Applications' page in the Tencent RTC console. A sidebar on the left contains navigation links: Overview, Applications (selected), Usage Statistics, Data Monitoring, Package Management, Relevant Services, and Development Tools. The main area is titled 'My Applications' and features a search bar. Below is a table with columns: Application name, SDKAppID, Status, Product information, Expiration time, and SDKSecretKey. Three applications are listed: 'live1', 'Conference', and '语音' (Voice). The 'Conference' application is highlighted with a blue background.

Application name	SDKAppID	Status	Product information	Expiration time	SDKSecretKey
live1	[redacted]	Enabled	Live : No version Chat : Development	-- 2024-04-12	*****
Conference	[redacted]	Enabled	Conference : Trial Chat : Development	2024-03-25 2024-04-11	*****
语音	[redacted]	Enabled	Call : 1-to-1 Call Conference : Lite Chat : Standard - Prepaid	2024-04-07 2024-04-10 2024-04-08	*****

2. In the Conference product information, click the **Enable auto-renewable** button, a confirmation pop-up will appear, click **Enable**.



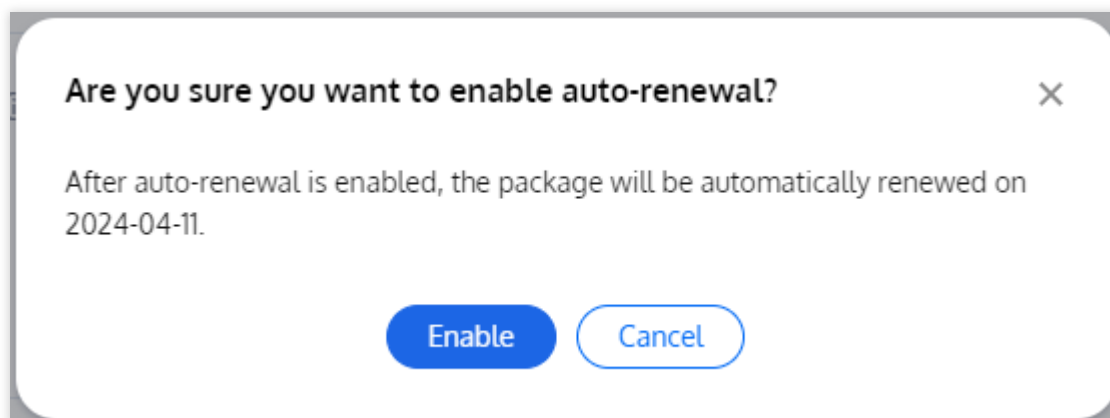
The screenshot shows the 'Application Overview' page for the 'Conference' product. The left sidebar lists navigation options: All Applications, Application Overview (selected), Advanced Features, Call, Conference, Live, RTC Engine, Chat, and In-game Voice Chat. The main area is titled 'Application Overview' and shows basic information for the application 'test'. Below this, the 'Products' section displays details for the 'Call' and 'Conference' products. The 'Conference' product is highlighted, and the 'Auto-renewable' button is circled in red.

**Basic Information**

Application name	test	SDKSecretKey	*****
SDKAppID	[redacted]	Creation time	2024-03-05 10:54:57
Description	--	Region	Global (excluding Chinese mainla
Tags	No tags yet		
Status	Enabled		

**Products** Quickly run sample demo in 3 steps >

Product	Edition	Expiration time	Auto-renewable	Buttons
Call	Call : No version	--	--	Claim free, Buy package, Integrate
Conference	Conference : Lite	2024-04-11	Auto-renewable (circled in red), Not enabled, Enable (circled in red)	Renewal



## Function and Billing Overview

To use TUIRoomKit, you need to purchase the Conference monthly package. The pricing and feature comparison of the Conference monthly package can be found in the table. For detailed billing information, please refer to [the Conference monthly package billing instructions](#).

Item		Free Trial	Lite	Standard	Pro
Price		As low as 0 USD for 14 days	<a href="#">Subscribe Now</a>	<a href="#">Subscribe Now</a>	<a href="#">Subscribe Now</a>
Package duration	Free minutes	10,000 minutes/month	10,000 minutes/month	10,000 minutes/month	10,000 minutes/month
	Package bonus minutes	-	100,000 minutes/month	300,000 minutes/month	450,000 minutes/month
	Pay-as-you-go upon exhaustion (within the validity of the package)	Services become unavailable after exhaustion	✓	✓	✓
Conference Features	Participants per room	100	20	50	200
	Active Rooms	10	20	100	200
	Quota Of Free Monthly Active	100/month	10,000/month	10,000/month	10,000/month



User (MAU)				
Video Quality	HD, FHD, 2K, 4K	HD, FHD	HD, FHD	HD, FHD, 2K, 4K
Complete UI	✓	✓	✓	✓
Custom UI	✓	✓	✓	✓
Floating Window	✓	✓	✓	✓
Screen Sharing	✓	✓	✓	✓
Member Management	✓	✓	✓	✓
Room Management	✓	✓	✓	✓
Free Speech Mode	✓	✓	✓	✓
Audio mix by default	✓	✓	✓	✓
On-stage Speech Mode	✓	-	✓	✓
Microphone Management	✓	-	✓	✓
Microphone Quantity	20	-	9	20
Chat in Conference	✓	-	✓	✓
AI noise suppression	✓	-	✓	✓
Less stutter under poor network conditions	✓	-	✓	✓
Multi-terminal login	✓	-	-	✓

TRTC Basic Features	On-Cloud Recording	✓	✓	✓	✓
	On-Cloud Mix-Transcoding	✓	✓	✓	✓
	Relay To CSS	✓	✓	✓	✓
Supported Platforms		iOS, Android, Web, Flutter, Electron, Windows	iOS, Android, Web, Flutter, Electron, Windows	iOS, Android, Web, Flutter, Electron, Windows	iOS, Android, Web, Flutter, Electron, Windows
Technical Support		-	Response time: 5 day/12 hour P1 - 2 hours P2 - 6 hours P3 - 12 hours	Response time: 5 day/12 hour P1 - 2 hours P2 - 6 hours P3 - 12 hours	Response time: 7 day/24 hour P1 - 1 hour P2 - 4 hours P3 - 8 hours
		-	Ticket and email support	Ticket and email support	Ticket and email support
		-	Telegram group	Telegram group	Telegram group

**Notes:**

1. Package duration: The duration of a package can be used to deduct your conference durations. If you use the trial edition, services will become unavailable for your application after you use up the package. If you use Lite, Standard or Pro, after you use up the package, your additional usage will be charged at pay-as-you-go rates.
2. Free minutes: Each Tencent Cloud account will get 10,000 free minutes per usage cycle (a usage cycle is one month) after it buys a TRTC Conference package. The free minutes can deduct usage of both TRTC Conference features and TRTC basic features such as on-cloud recording and on-cloud mixtranscoding. To learn more, see [Free Minutes](#).
3. Package bonus minutes: Bonus minutes can deduct your conference durations. They are valid for one month and will expire at the end of each usage cycle.
4. TRTC basic features: In addition to TRTC Conference features, you can also use TRTC's basic features, which will incur additional fees. For the billing details, see [Billing of On-Cloud Recording](#) and [Billing of MixTranscoding and Relay to CDN](#).

# Integration (TUIRoomKit)

## iOS

Last updated : 2024-04-12 11:29:00

This article will introduce how to complete the integration of the `TUIRoomKit` Component in the shortest time. By following this document, you will complete the following key steps within an hour and ultimately obtain an audio/video conference function with a complete UI interface.

## Environment preparation

iOS 13.0 and higher.

Xcode 12.0 and higher.

Swift 4.2 and higher.

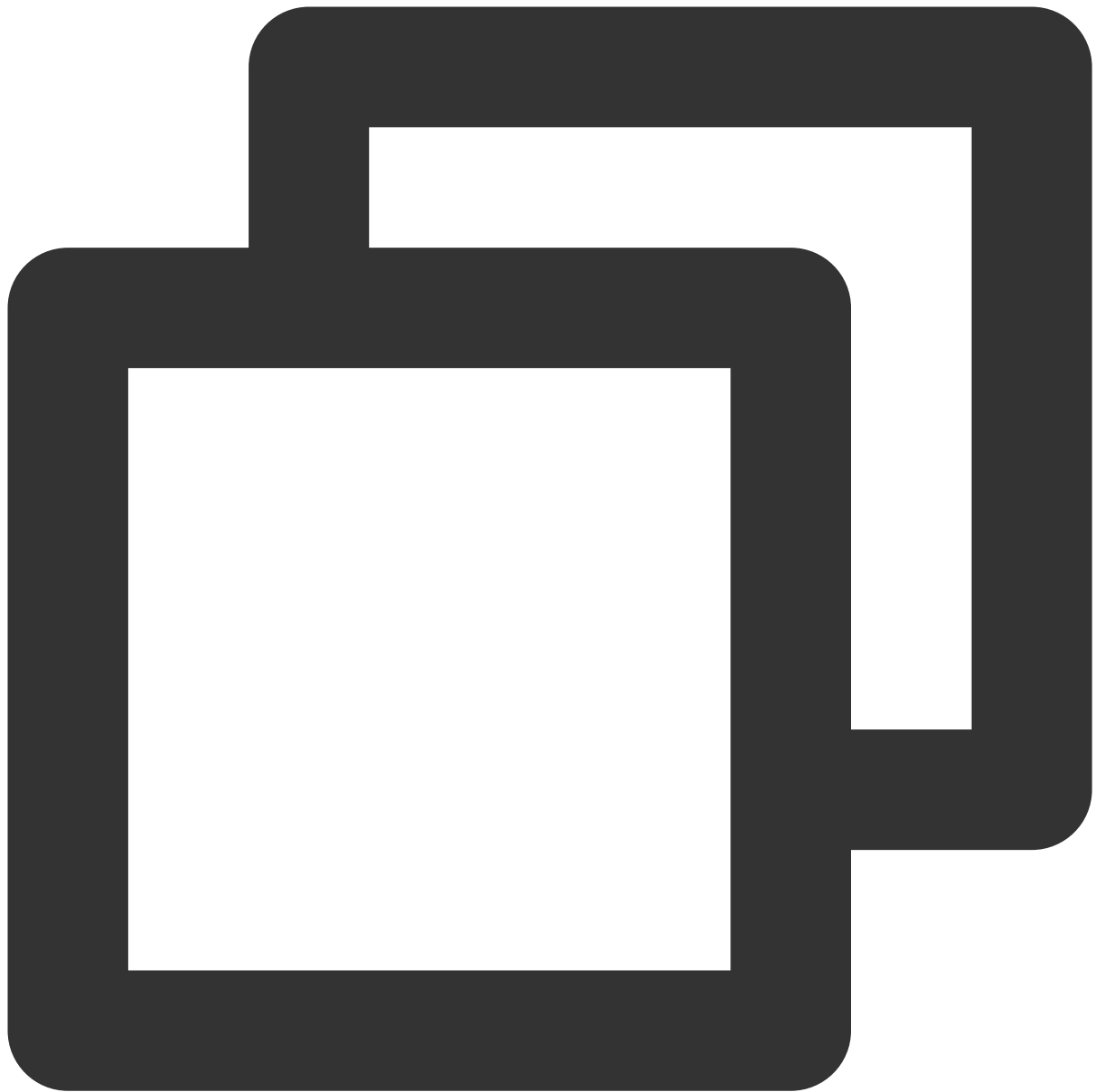
## Activate the service

Before initiating a meeting with TUIRoomKit, you need to activate the exclusive multi-person audio and video interaction service for TUIRoomKit on the console. For specific steps, please refer to [Activate Service](#).

## Integrate the TUIRoomKit Component

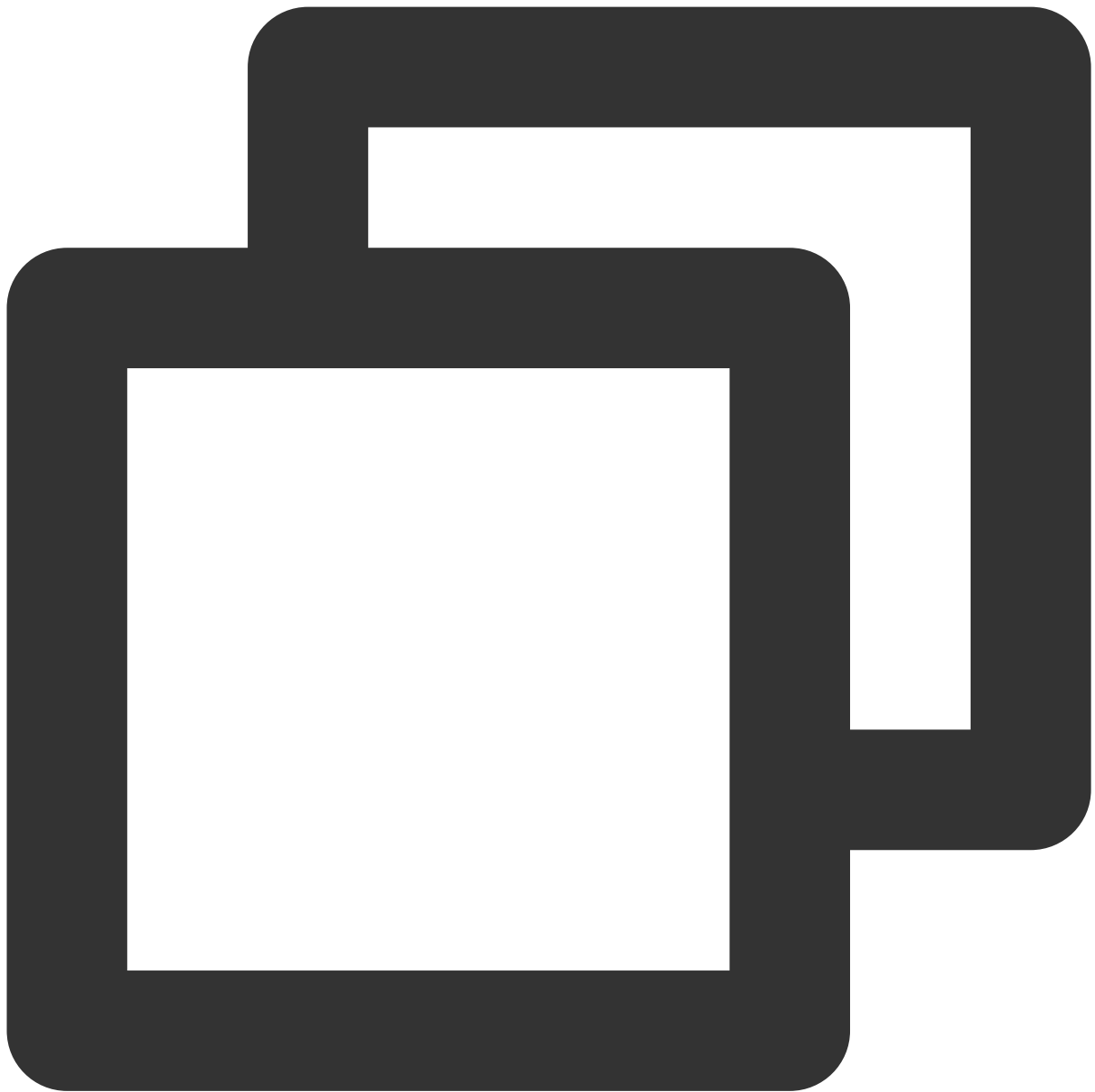
### **Pull remote CocoaPods integration (recommended when there is no source code modification)**

1. Add the following dependencies to your Podfile file.



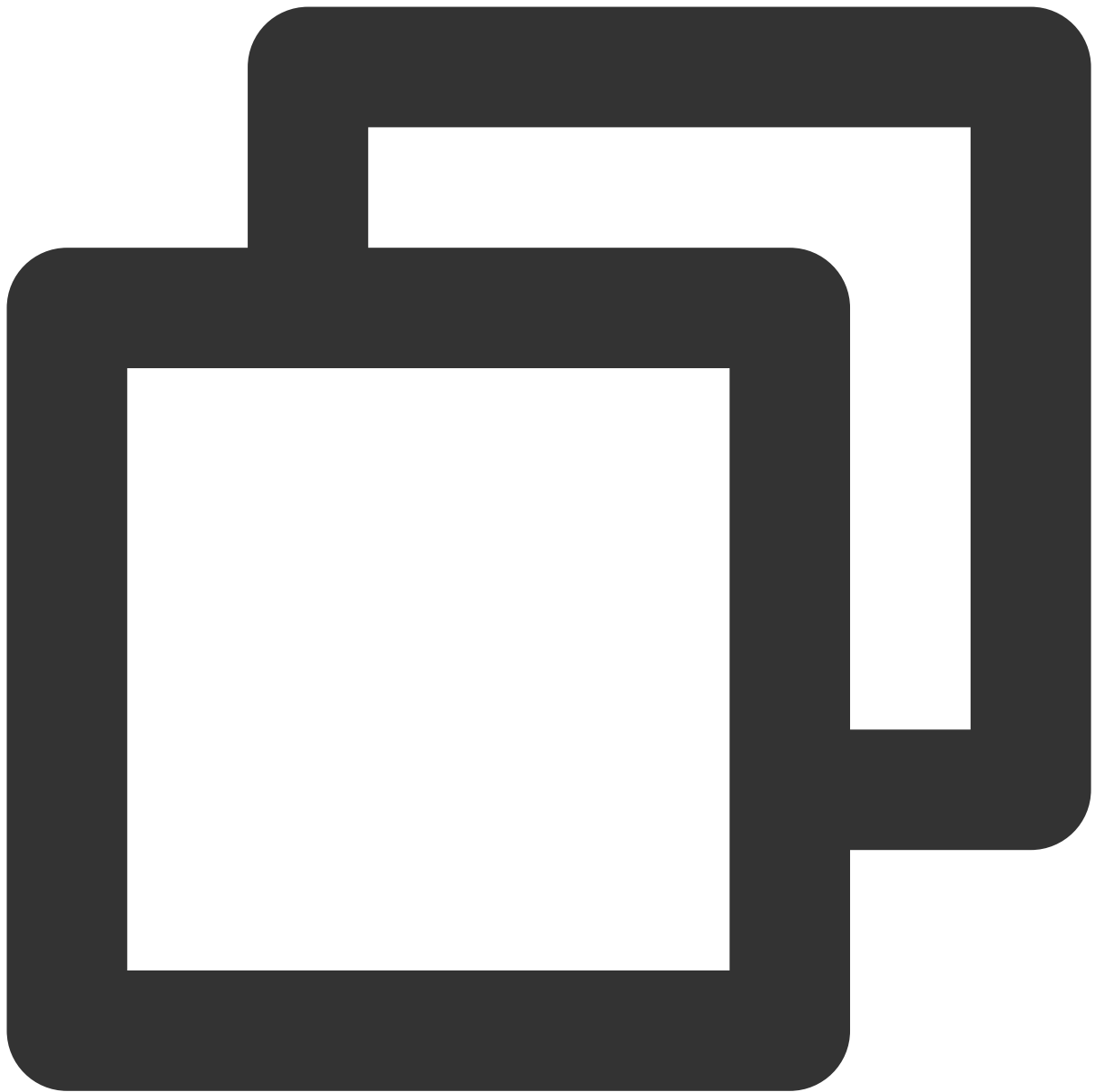
```
pod 'TUIRoomKit'
```

2. Execute the following command to install the Component.



```
pod install
```

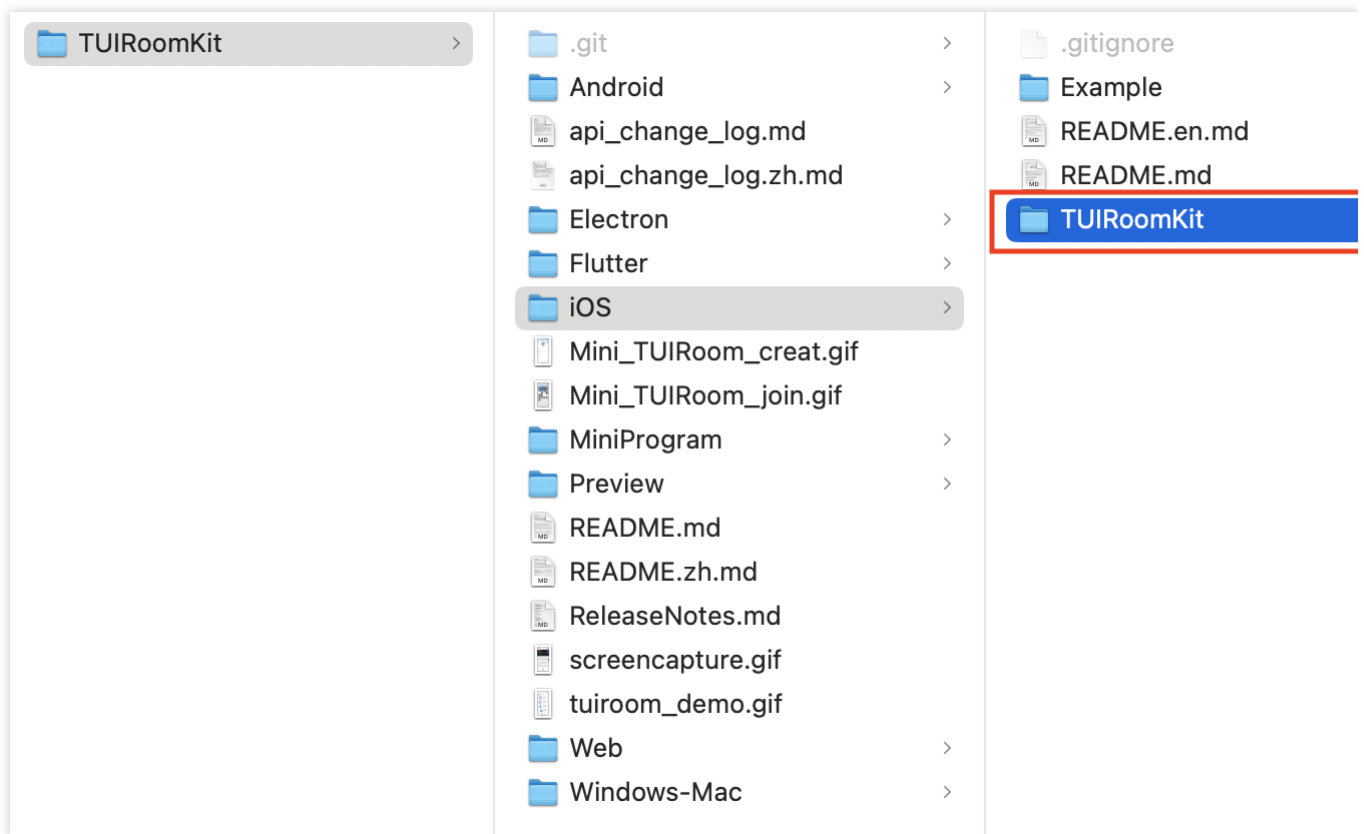
3. If you cannot install the latest version of TUIRoomKit, execute the following command to update the local CocoaPods repository list.



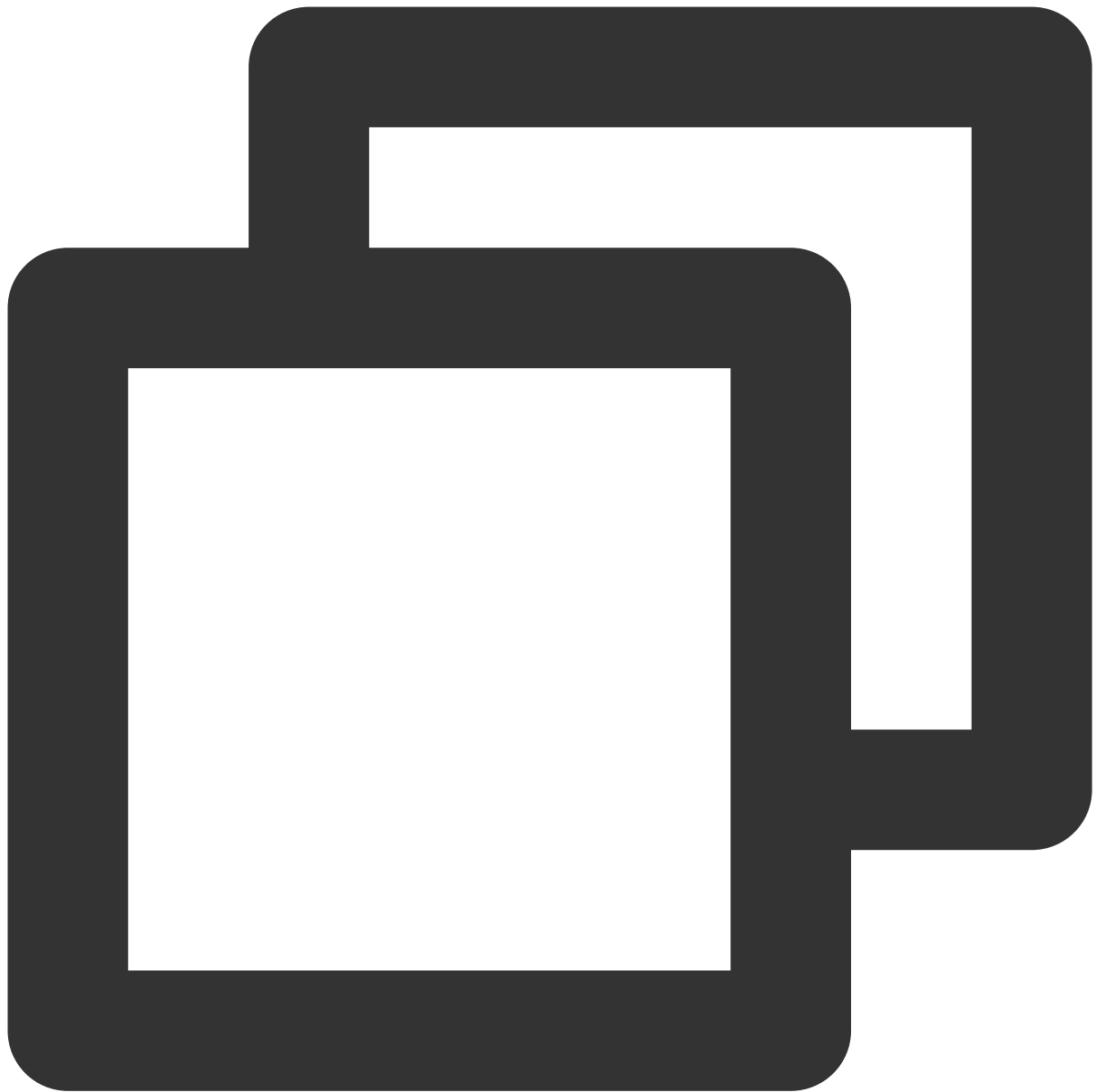
```
pod repo update
```

### Source code integration (recommended when there are source code modifications)

1. Download the TUIRoomKit source code from GitHub. Drag directly into your project directory:  
TUIRoomKit/iOS/TUIRoomKit.



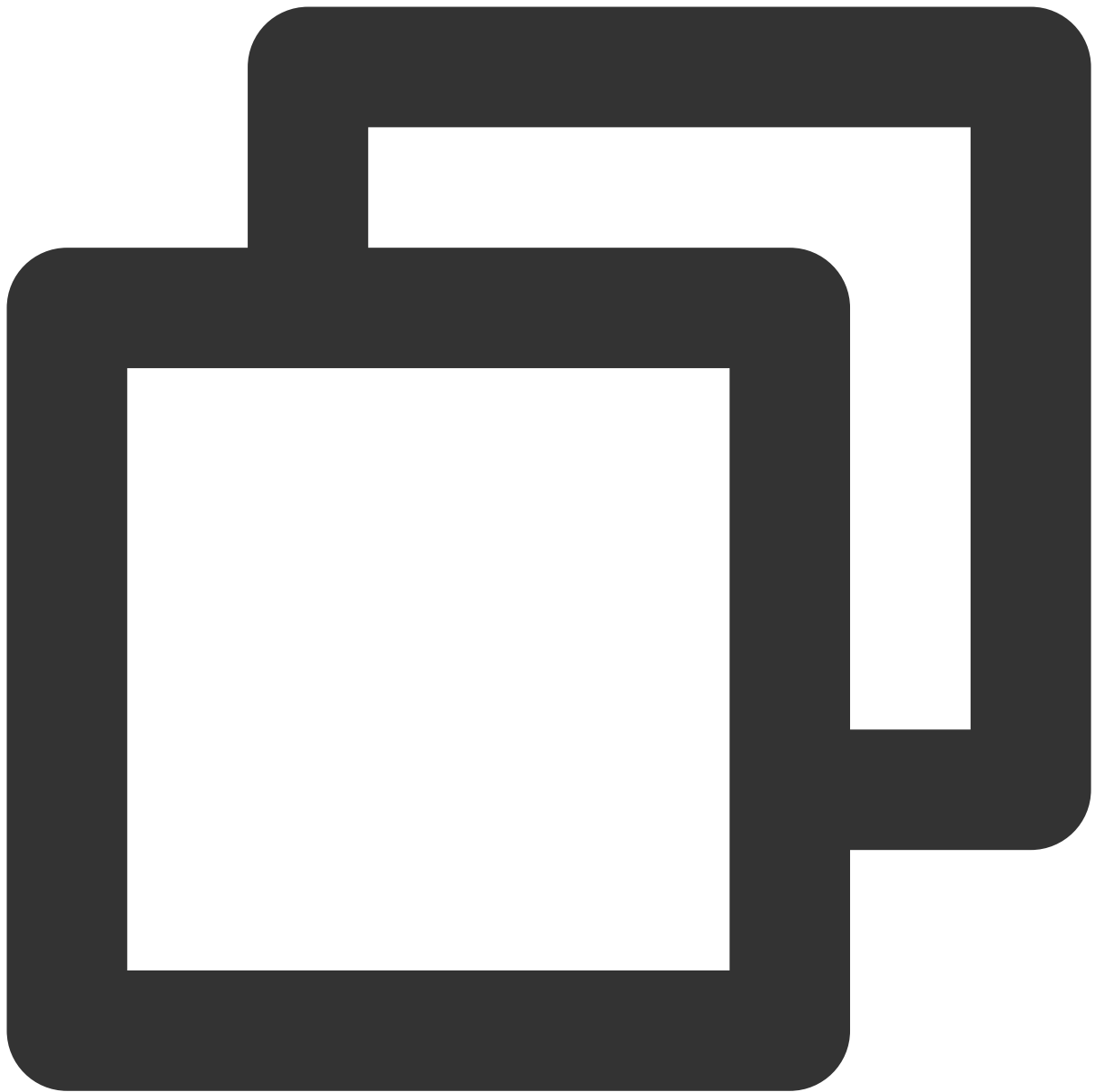
2. Modify the path of the TUIRoomKit component in your Podfile, for example:



```
pod 'TUIRoomKit', :path => "../TUIRoomKit/"
```

3. Execute installation command.





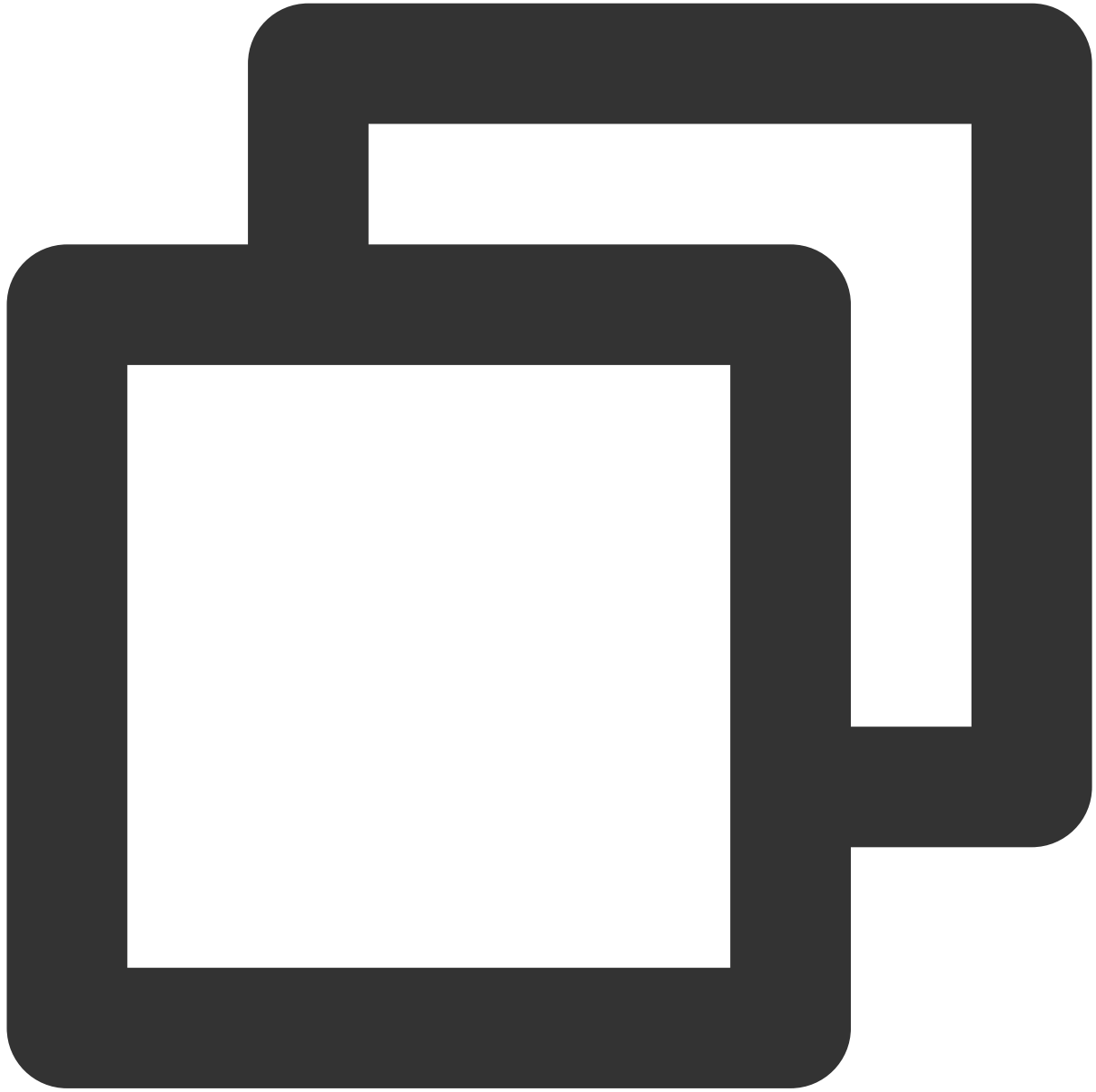
```
pod install
```

## Quick setup

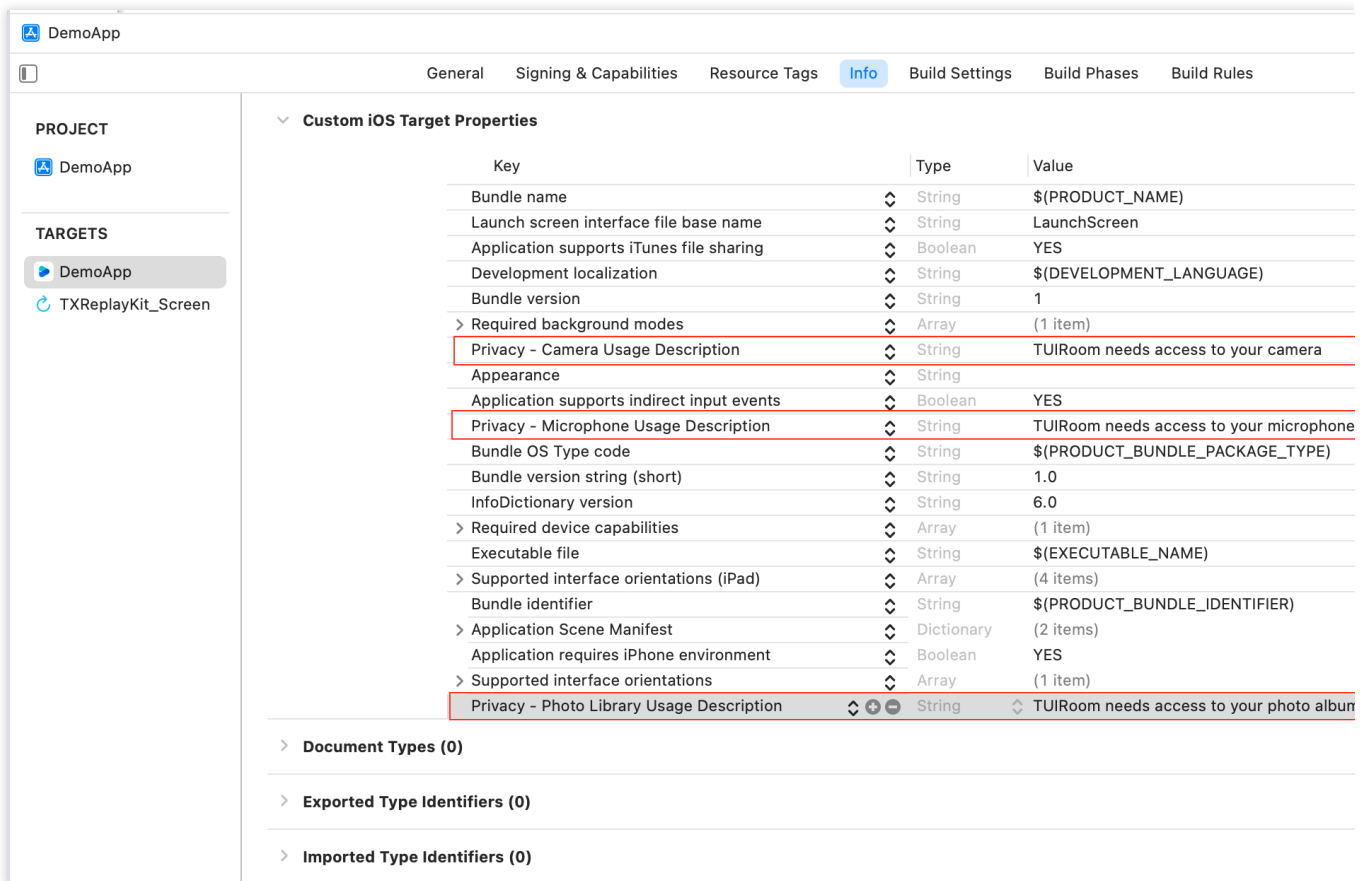
### Step 1: Project Configuration

To use the audio and video functions, you need to authorize the use of microphone, camera and photo album. Add the following items to the App's Info.plist, corresponding to the prompt messages for the microphone, camera, and photo

album when the system pops up the authorization dialog box.

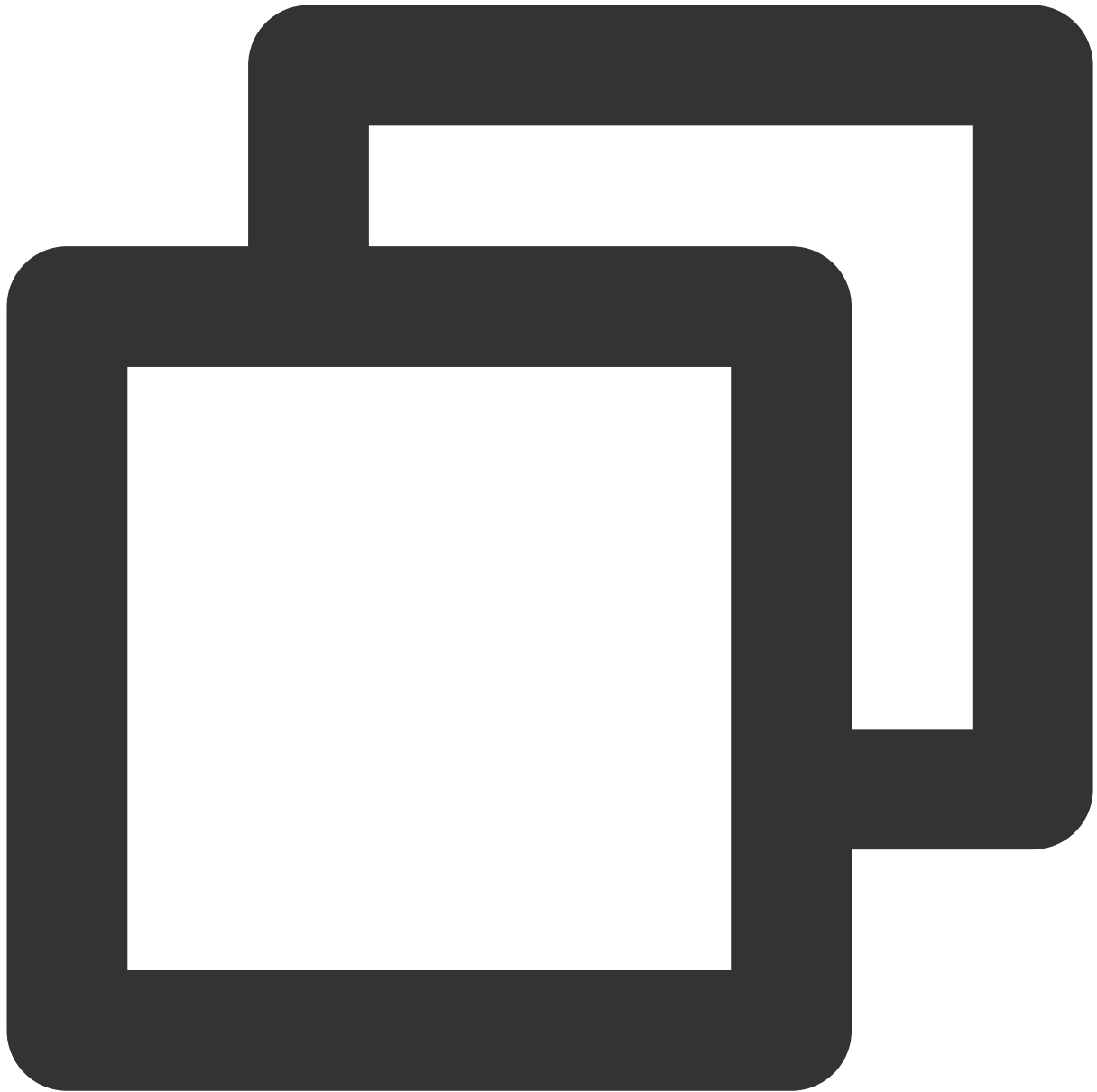


```
<key>NSCameraUsageDescription</key>
<string>TUIRoom needs access to your Camera permission</string>
<key>NSMicrophoneUsageDescription</key>
<string>TUIRoom needs access to your Mic permission</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>TUIRoom needs access to your Photo Library</string>
```



## Step 2: Log in TUI Component

Add the following code to your project. Its function is to complete the initialization of TUI components by calling the relevant interfaces in TUICore. This step is very critical, because all functions of TUIRoomKit can only be used normally after successful login, so please be patient and check whether the relevant parameters are configured correctly:



```
import TUICore

TUILogin.login(1400000001,                                // Please replace with the SDKApp
               userID: "998",                               // Please replace with your UserI
               userSig: "xxxxxxxxxx") {                     // You can calculate a UserSig in
    print("login success")
} fail: { (code, message) in
    print("login failed, code: \ \(code), error: \ \(message ?? "nil")")
}
```

## Parameter Description

Here is a detailed introduction to the key parameters used in the login function:

**SDKAppID** : You have already obtained it in [Activate the service](#), so it will not be repeated here.

**UserID** : The ID of the current user, string type, only allows to contain English letters (a-z and A-Z), numbers (0-9), hyphens (-), and underscores (\_).

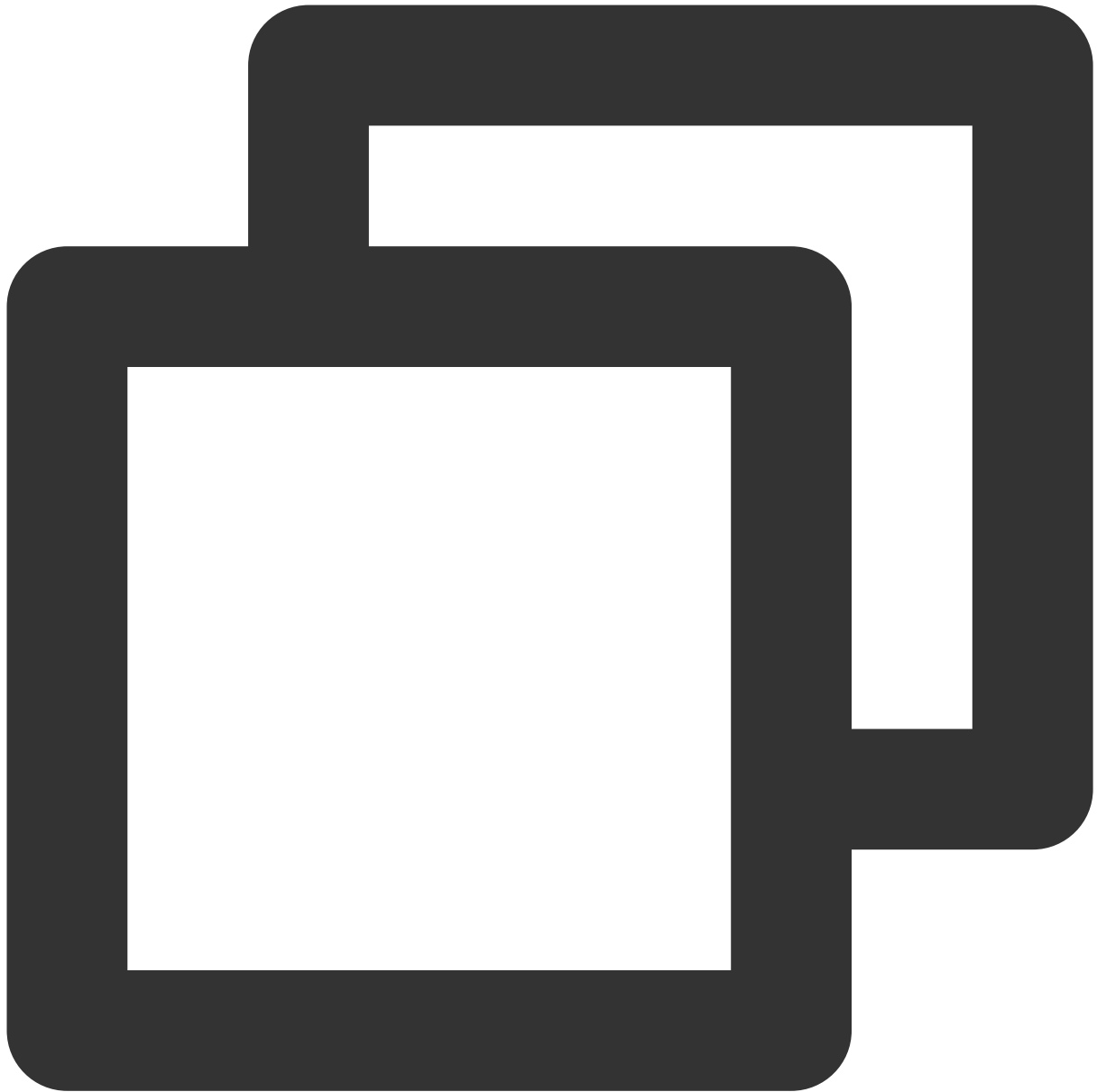
**UserSig** : Encrypt the SDKAppID, UserID, etc. with the SDKSecretKey obtained in [Activate the service](#) to get the UserSig, which is a ticket for authorization and is used for Tencent Cloud to recognize whether the current user can use the TRTC service. You can create a temporarily available UserSig through the [UserSig Tools](#) through the project sidebar in the console.

The screenshot displays the 'UserSig Tools' interface within the Tencent Cloud console. On the left, a sidebar lists navigation options: Overview, Applications, Usage Statistics, Data Monitoring, Package Management, and Development Tools. The 'Development Tools' section is expanded, and 'UserSig Tools' is highlighted. The main content area is titled 'UserSig Tools' and features a warning banner at the top: 'You haven't provided a payment method. We will suspend the service for your account after you use up your free resources. To avoid service interruption, please complete your information and refresh.' Below the banner, there are two primary tool panels. The 'Signature (UserSig) Generator' panel on the left includes a description: 'This tool can quickly generate a UserSig, which can be used to run through demos and to debug features.' It contains input fields for 'Application (SDKAppID)' (a dropdown menu), 'Username (UserID)' (a text input), and 'Secret key' (a text input with a note 'Auto-generated after you select an application'). A 'Generate' button is present, followed by a 'Generate result' section with a large text area and a 'Copy' button. The 'Signature (UserSig) Verifier' panel on the right has a description: 'This tool is used to verify the validity of the UserSig you use.' It includes similar input fields for 'Application (SDKAppID)', 'Username (UserID)', and 'Secret key', along with a 'Verify' button. The 'UserSig' field has a placeholder 'Please enter'.

For more information, please refer to the [UserSig related](#).

## Step 3: Set user nickname and avatar

The user's username and avatar can be set by calling `setSelfInfo` of `TUIRoomEngine`, which can be set in the callback of successful login.

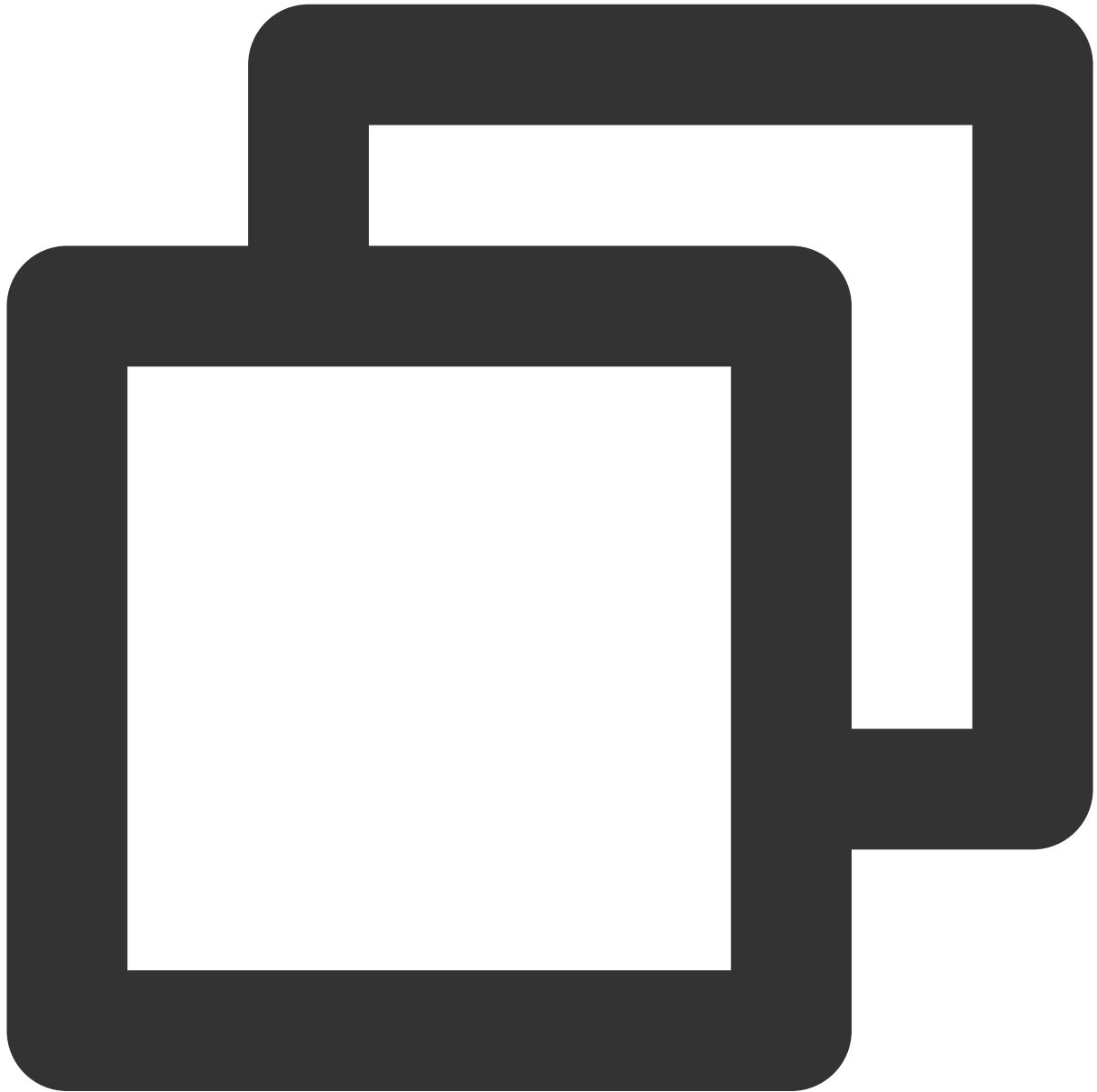


```
import TUIRoomEngine

TUIRoomEngine.setSelfInfo(userName: "xxx", avatarUrl: "xxx") {
    print("setSelfInfo success")
} onError: { code, message in
    print("setSelfInfo failed, code:\\(code),message:\\(message)")
}
```

#### Step 4: The host start a quick conference

The main conference page is `ConferenceMainViewController`. Just call `quickStartConference` to initiate a quick conference and jump to `ConferenceMainViewController` in the `onConferenceStarted` callback.



```
import TUIRoomKit

// CreateRoomViewController for your own ViewController
class CreateConferenceViewController: UIViewController {
    private var conferenceViewController: ConferenceMainViewController?
    func quickStartConferenceAction() {
        conferenceViewController = ConferenceMainViewController()
        conferenceViewController?.setConferenceObserver(observer: self)
    }
}
```

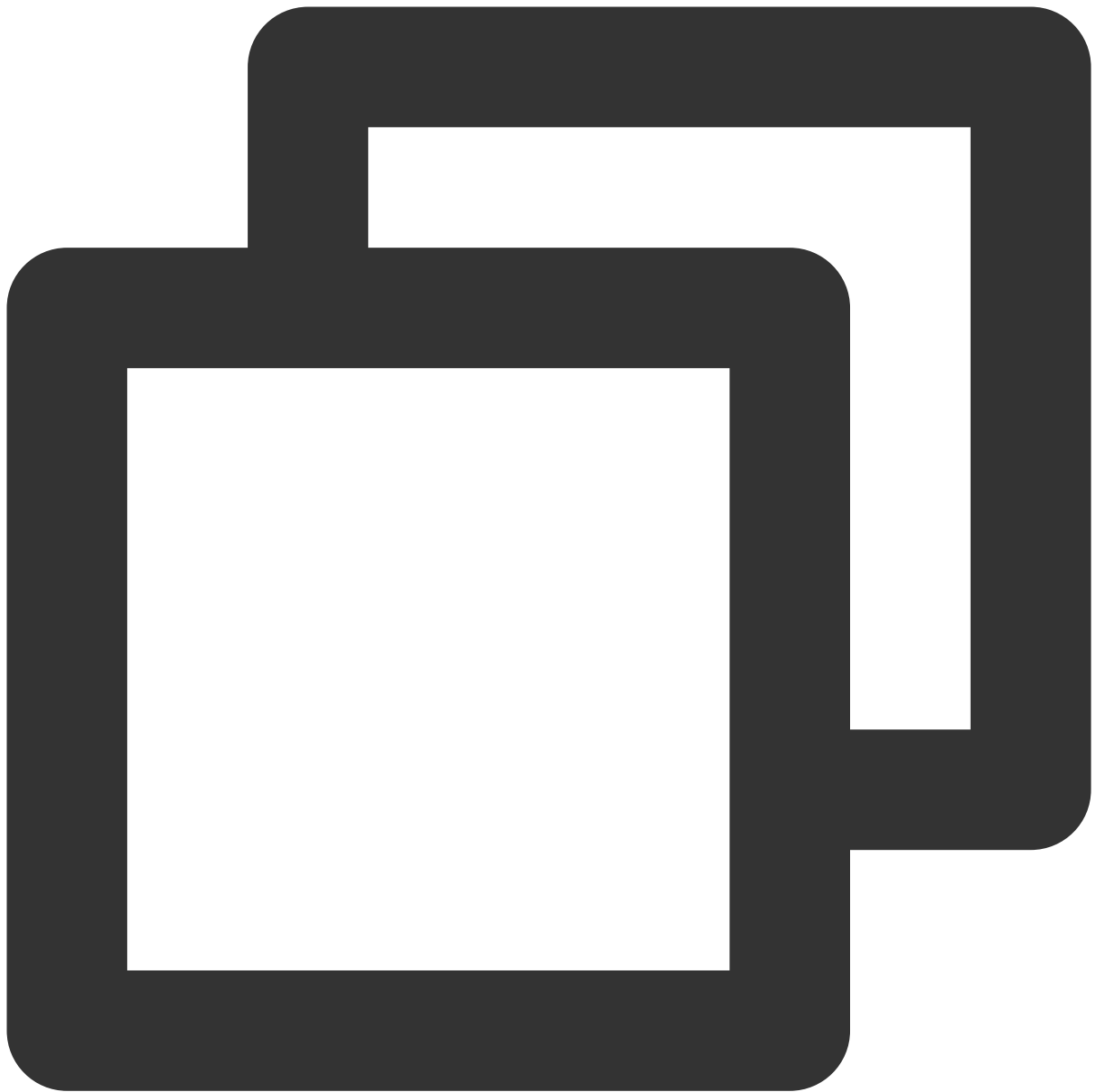
```
        conferenceViewController?.quickStartConference(conferenceId: "123456")
    }
}

extension CreateConferenceViewController: ConferenceObserver {
    func onConferenceStarted(conferenceId: String, error: ConferenceError) {
        if error == .success, let vc = conferenceViewController {
            navigationController?.pushViewController(vc, animated: true)
        }
        conferenceViewController = nil
    }
}
```

## Step 5: Members join the conference

The main conference page is `ConferenceMainViewController`. Just call `joinConference` to join the conference and jump to `ConferenceMainViewController` in the `onConferenceJoined` callback.





```
import TUIRoomKit

// EnterRoomViewController for your own ViewController
class EnterConferenceViewController: UIViewController {
    private var conferenceViewController: ConferenceMainViewController?
    private func joinConferenceAction() {
        conferenceViewController = ConferenceMainViewController()
        conferenceViewController?.setConferenceObserver(observer: self)
        conferenceViewController?.joinConference(conferenceId: "123456")
    }
}
```

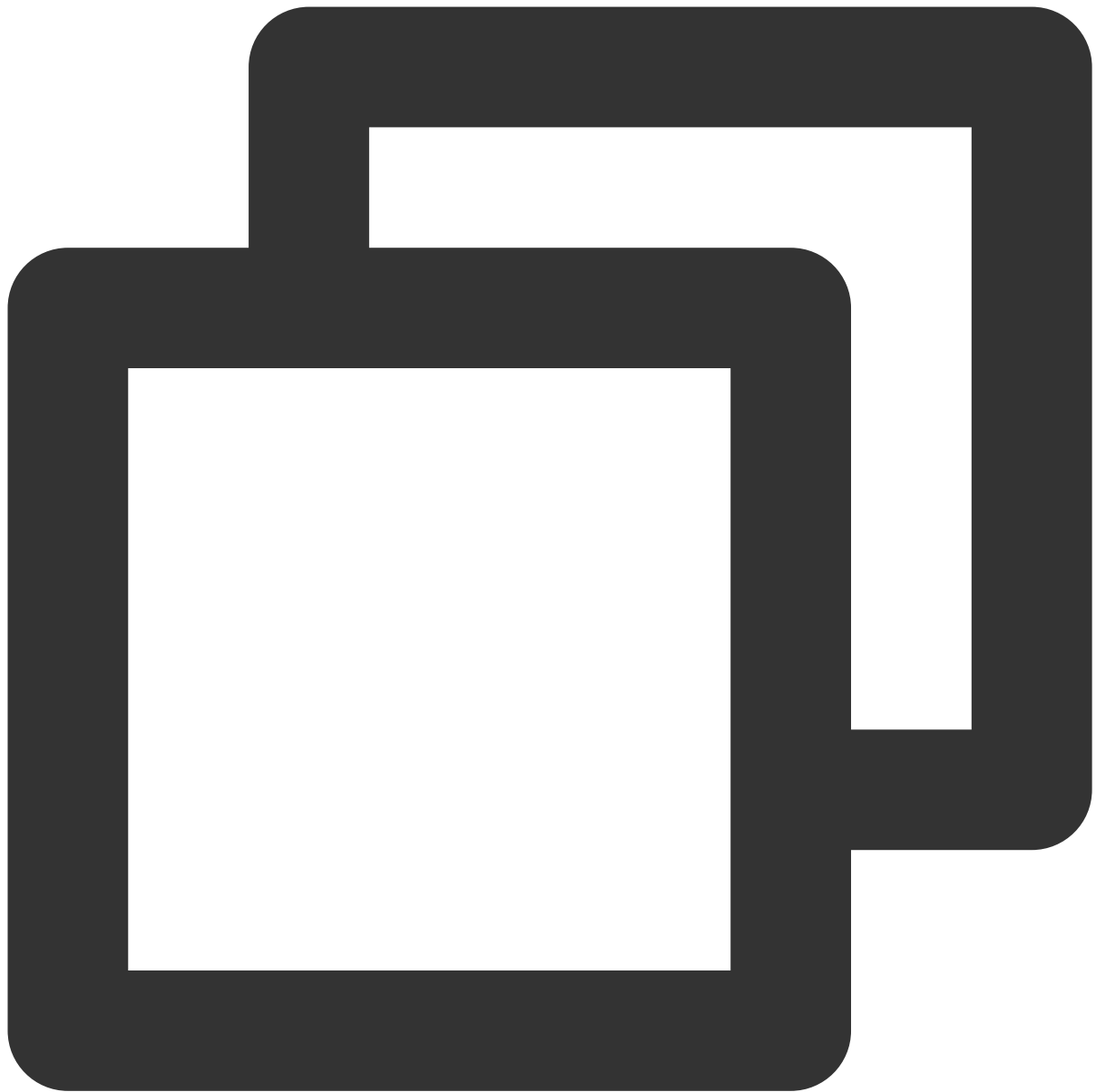
```
extension EnterConferenceViewController: ConferenceObserver {
    func onConferenceJoined(conferenceId: String, error: ConferenceError) {
        if error == .success, let vc = conferenceViewController {
            navigationController?.pushViewController(vc, animated: true)
        }
        conferenceViewController = nil
    }
}
```

## More Features

### UI Widget Access

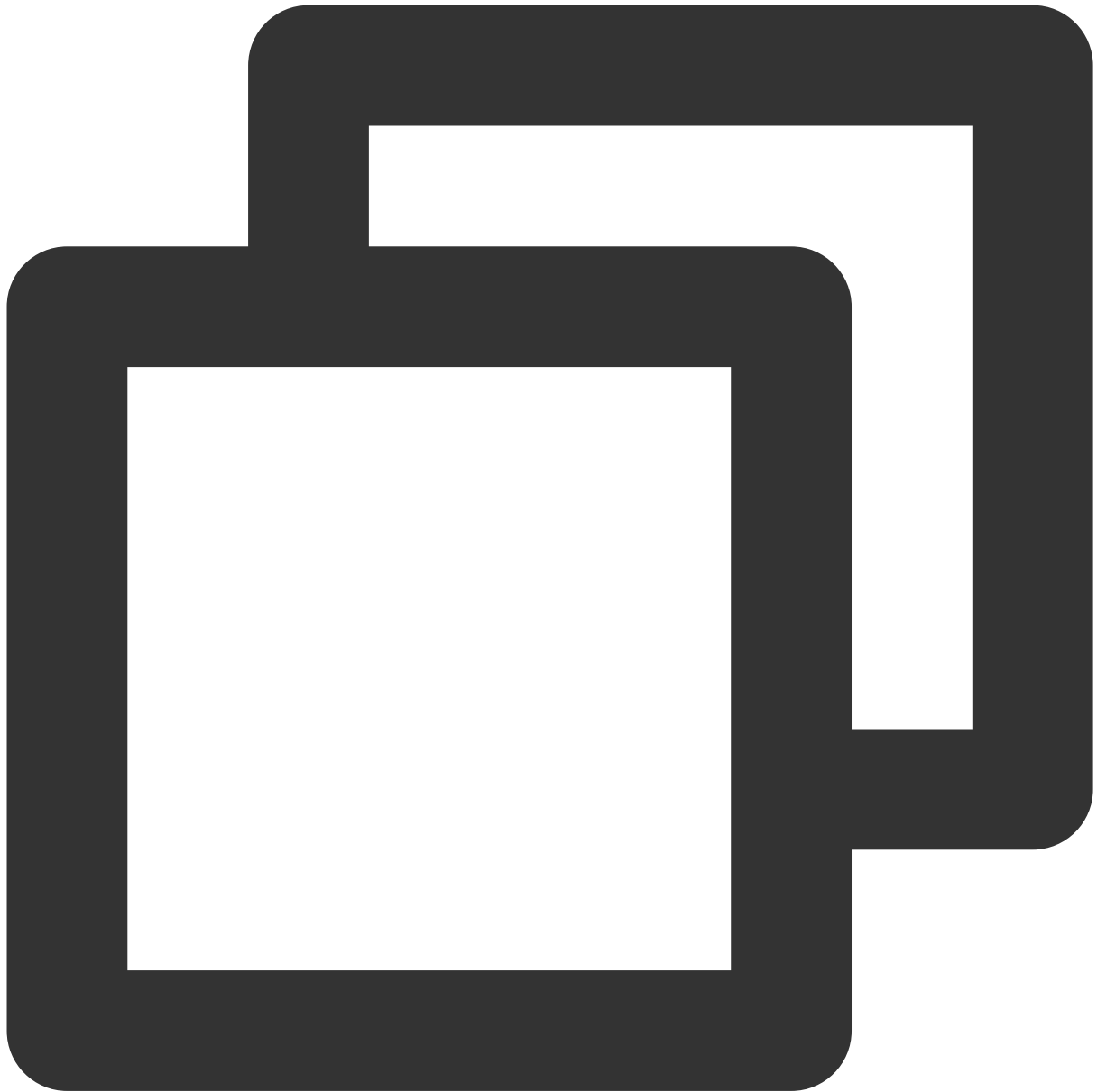
Import the chat widget using CocoaPods, with the following specific steps:

1. Add the following dependencies to your `Podfile` file.



```
pod 'TUIChat'           # [Optional] TUIChat Widget
```

2. Execute the following command to install the component.



```
pod install
```

## Common Problems

If you encounter problems when using TUIRoomKit, please check the [FAQ document](#) first.

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# Android

Last updated : 2024-04-12 11:29:03

This article will introduce how to complete the integration of the `TUIRoomKit` Component in the shortest time. By following this document, you will complete the following key steps within an hour and ultimately obtain an audio/video conference function with a complete UI interface.

## Environment preparation

Minimum compatibility with Android 4.4 (SDK API Level 19), recommended to use Android 5.0 (SDK API Level 21) and above.

Android Studio 3.5 and above (Gradle 3.5.4 and above).

Mobile devices with Android 4.4 and above.

## Activate the service

Before initiating a meeting with TUIRoomKit, you need to activate the exclusive multi-person audio and video interaction service for TUIRoomKit on the console. For specific steps, please refer to [Activate Service](#).

## module source code integration

1. Clone/download the code in [Github](#), and then copy the **debug**, **timcommon**, **tuichat**, and **tuiroomkit** subdirectories in the **Android** directory to the app directory in your current project.
2. Find the **setting.gradle** file in the project root directory and add the following code to it. Its function is to import **tuiroomkit** as a local module into your current project.



```
include ':debug'
include ':timcommon'
include ':tuichat'
include ':tuiroomkit'
```

3. Find the **build.gradle** file in the app directory and add the following code to it. Its function is to declare the dependence of the current app on the newly added **tuiroomkit** component.



```
api project(':tuiroomkit')
```

4. Add the following line to the **gradle.properties** file to automatically convert third-party libraries to be compatible with AndroidX:





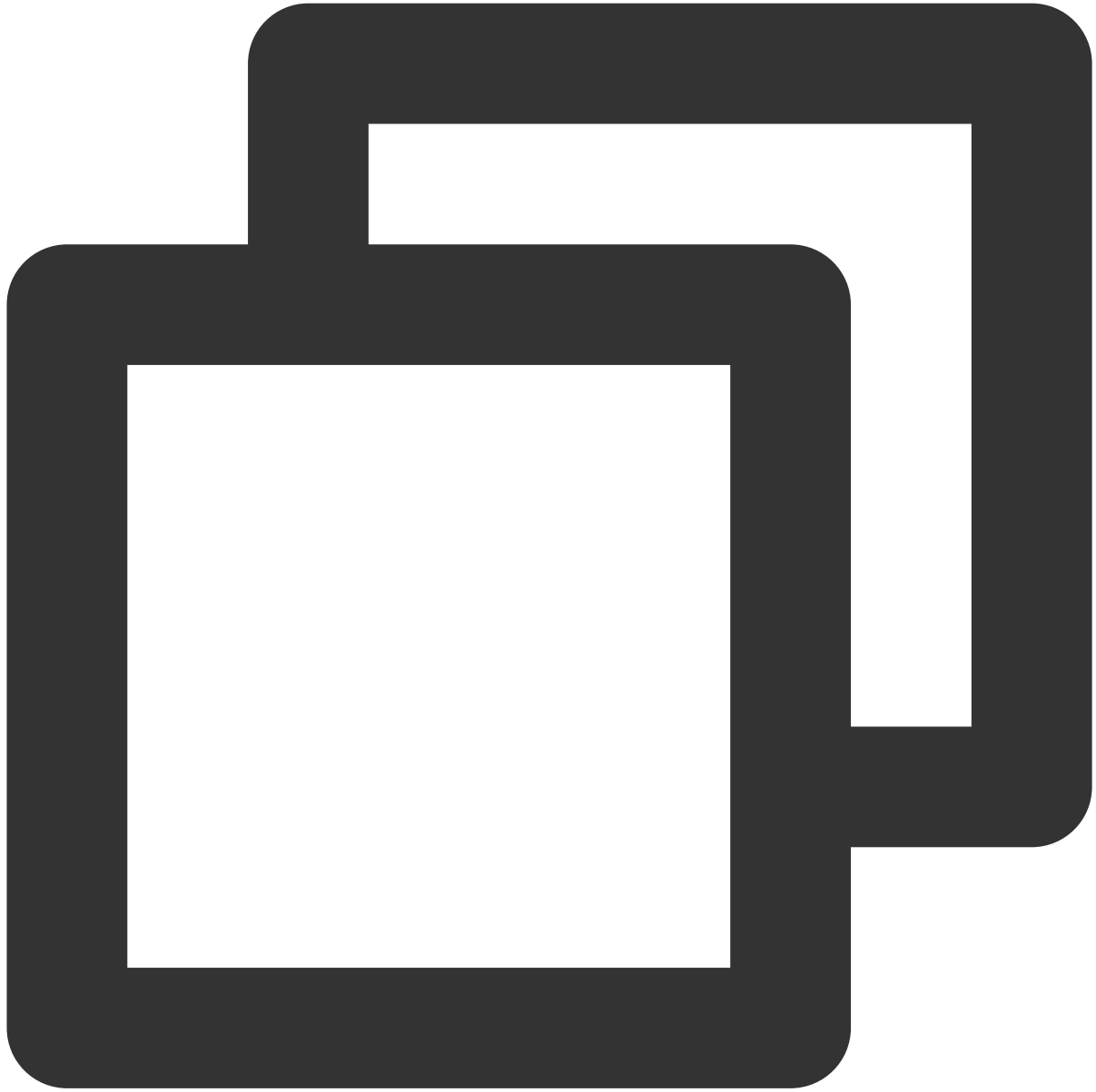
```
android.enableJetifier=true
```

5. To add maven repository and Kotlin support, add the following in the **build.gradle** file of the root project (same level as settings.gradle):



```
buildscript {  
    repositories {  
        mavenCentral()  
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }  
    }  
    dependencies {  
        classpath 'com.android.tools.build:gradle:7.0.0'  
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.5.31"  
    }  
}
```

If you are using Gradle 8.x, you need to add the following code.



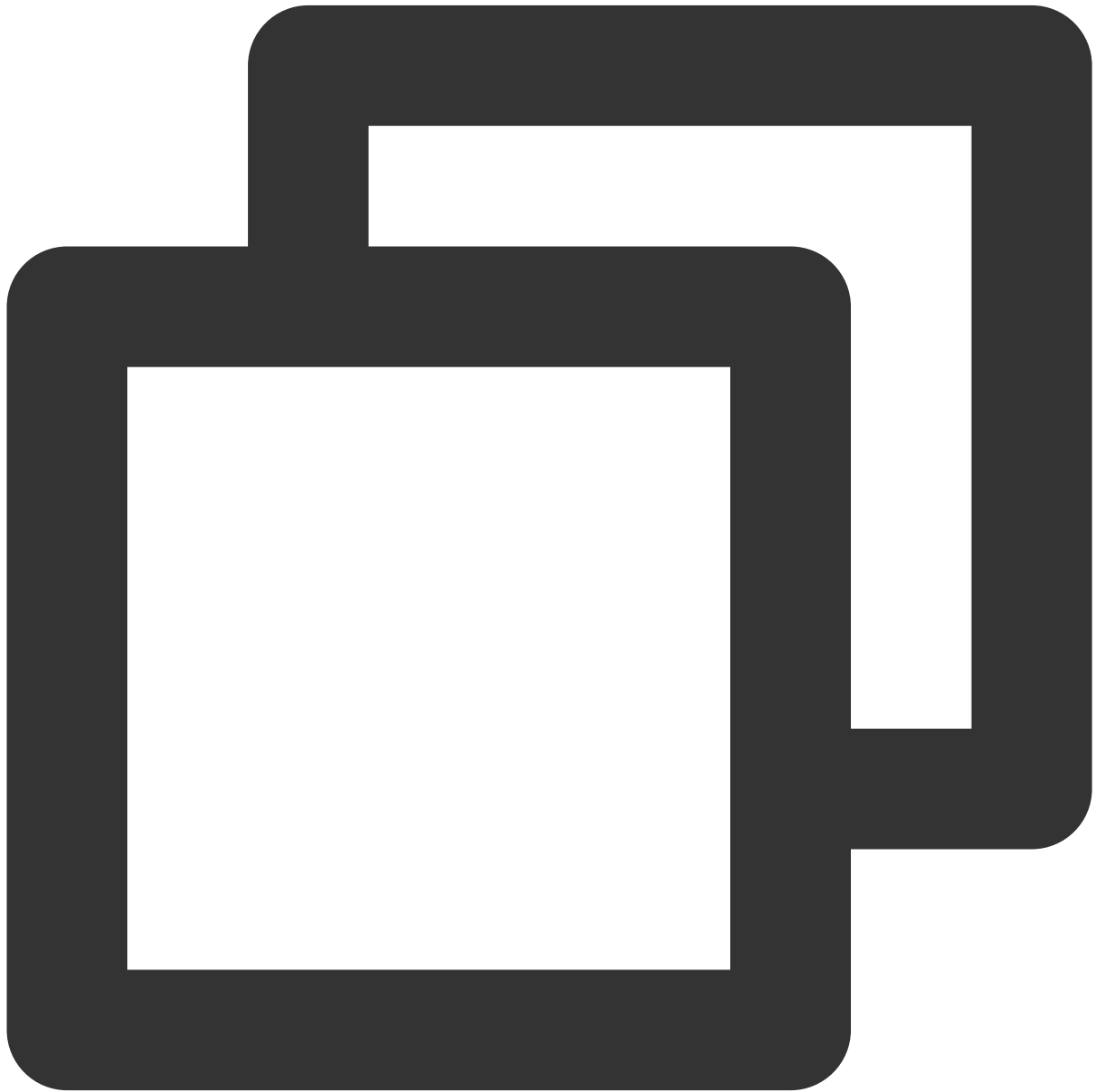
```
buildscript {  
    repositories {  
        mavenCentral()  
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }  
    }  
    dependencies {  
        classpath 'com.android.tools.build:gradle:8.0.2'  
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.9.0"  
    }  
}
```

```
}
```

**Note:**

The version correspondence between Kotlin, Gradle and AGP can be [viewed here](#).

6. Since we use the reflection feature of Java inside the SDK, we need to add some classes in the SDK to the unobfuscated list, so you need to add the following code to the **proguard-rules.pro** file:



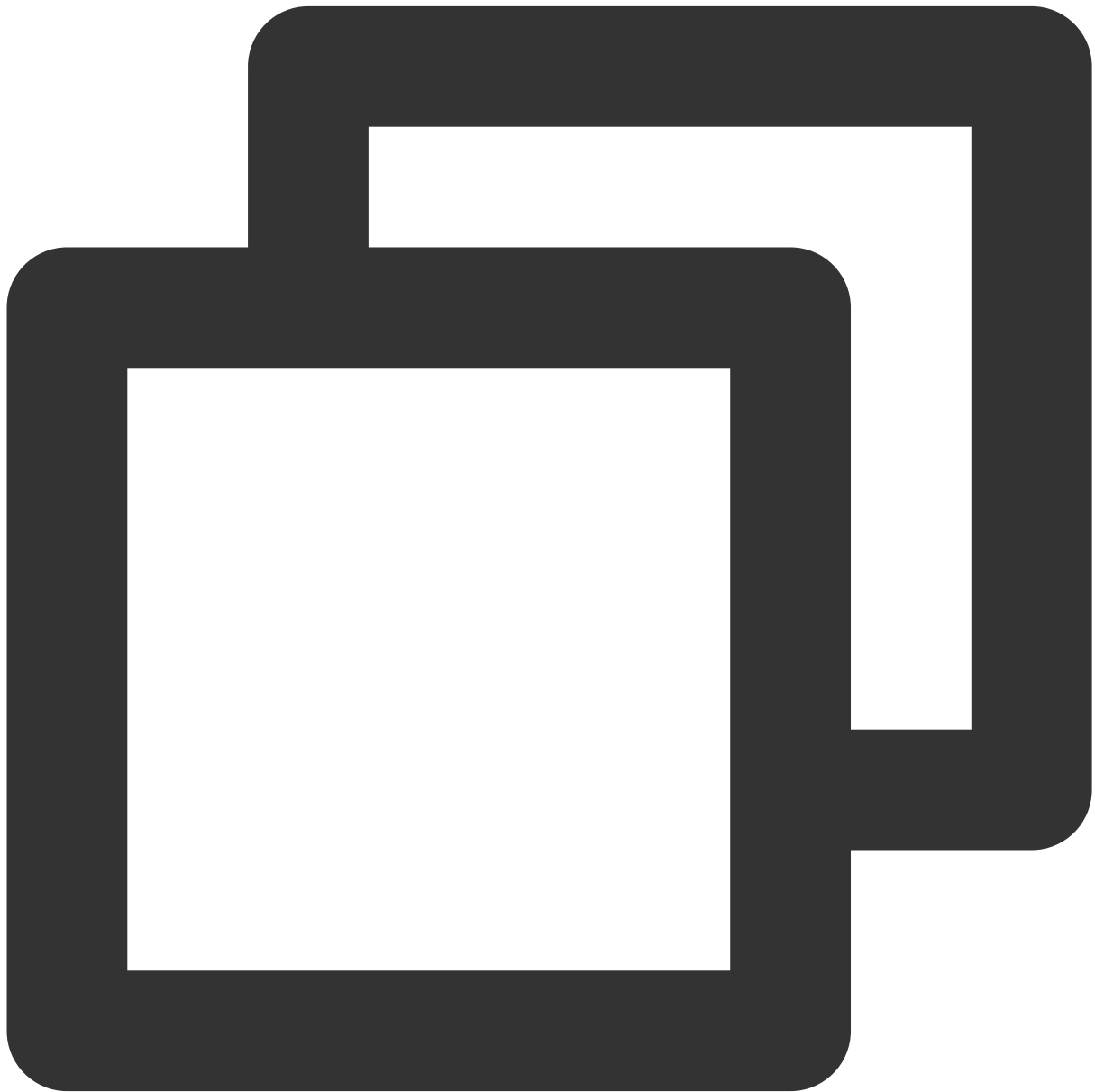
```
-keep class com.tencent.** { *; }
```

## Quick setup

By referring to the steps below, you can quickly build the main interface of the meeting in your project.

### Step 1: Component Login

Add the following code to your project. Its function is to complete the component login by calling the relevant interface in TUILogin. This step is extremely critical, because you can only use the various functions of TUIRoomKit normally after logging in, so please be patient and check whether the relevant parameters are configured correctly:



```
TUILogin.login(context,
                1400000001,    // Please replace it with the SDKAppID obtained from
                "998",         // Please replace with your UserID
                "xxxxxxxxxx",   // You can calculate a UserSig in the console and fil
                callback);     // Callback for successful login
```

### Parameter Description

Here is a detailed introduction to several key parameters needed in the login function:

**SDKAppID** : Get it in the last step of [activating the service](#).

**UserID** : The ID of the current user, a string type, only allowed to contain English letters (a-z and A-Z), numbers (0-9), hyphens (-) and underscores (\_).

**UserSig** : Use the SDKSecretKey obtained in step 4 of [activating the service](#) to encrypt the SDKAppID, UserID and other information to obtain the UserSig, which is an authentication ticket used by Tencent Cloud to identify whether the current user can use TRTC services. You can generate a temporarily available UserSig through the auxiliary tools in the [console](#).

For more information, please refer to the [UserSig related](#).

### Step 2: Set user nickname and avatar

The user's username and avatar can be set by calling `setSelfInfo` of `TUIRoomEngine`, which can be set in the callback of successful login.



```
TUIRoomEngine.setSelfInfo("nickName", "faceUrl", null);
```

### Step 3: The host initiates a quick meeting

1. Add interface layout in activity\_conference\_owner.xml:



```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/conference_owner_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

2. ConferenceMainFragment is the main interface of the conference. You only need to call quickStartConference to initiate a quick conference, and add ConferenceMainFragment to the current Activity in the quick conference callback onConferenceStarted to initiate a quick conference.





```
public class ConferenceOwnerActivity extends AppCompatActivity {  
    private static final String TAG = "ConferenceOwnerActivity";  
  
    private ConferenceObserver mConferenceObserver;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_conference_owner);  
    }  
}
```

```
ConferenceMainFragment fragment = new ConferenceMainFragment();
setConferenceObserver(fragment);
// Replace "123456" with the corresponding conference number
fragment.quickStartConference("123456");
}

private void setConferenceObserver(ConferenceMainFragment fragment) {
    mConferenceObserver = new ConferenceObserver() {
        @Override
        public void onConferenceStarted(String conferenceId, ConferenceError error) {
            super.onConferenceStarted(conferenceId, error);
            if (error != ConferenceError.SUCCESS) {
                Log.e(TAG, "Error : " + error);
                return;
            }
            FragmentManager manager = getSupportFragmentManager();
            FragmentTransaction transaction = manager.beginTransaction();
            transaction.add(R.id.conference_owner_container, fragment);
            transaction.commitAllowingStateLoss();
        }
    };
    fragment.setConferenceObserver(mConferenceObserver);
}
}
```

### Note

ConferenceOwnerActivity must inherit FragmentActivity or its subclass, otherwise the interface will not be displayed.

## Step 4: General members join the meeting

1. Add interface layout in activity\_conference\_general.xml:



```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/conference_general_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

2. ConferenceMainFragment is the main interface of the conference. You only need to call `joinConference` to join the conference and add `ConferenceMainFragment` to the current Activity in the `onConferenceJoined` callback to join the conference to participate in the current conference.



```
public class ConferenceGeneralActivity extends AppCompatActivity {  
    private static final String TAG = "ConferenceGeneralActivity";  
  
    private ConferenceObserver mConferenceObserver;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_conference_general);  
    }  
}
```

```
ConferenceMainFragment fragment = new ConferenceMainFragment();
setConferenceObserver(fragment);
// Replace "123456" with the corresponding conference number
fragment.joinConference("123456");
}

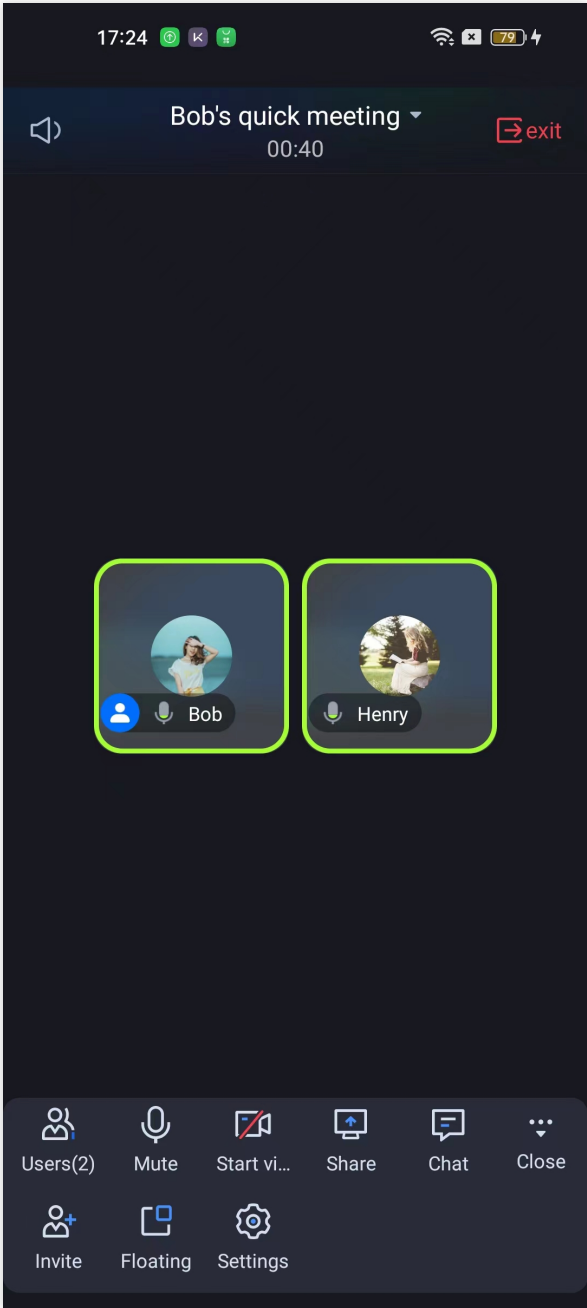
private void setConferenceObserver(ConferenceMainFragment fragment) {
    mConferenceObserver = new ConferenceObserver() {
        @Override
        public void onConferenceJoined(String conferenceId, ConferenceError error) {
            super.onConferenceJoined(conferenceId, error);
            if (error != ConferenceError.SUCCESS) {
                Log.e(TAG, "Error : " + error);
                return;
            }
            FragmentManager manager = getSupportFragmentManager();
            FragmentTransaction transaction = manager.beginTransaction();
            transaction.add(R.id.conference_general_container, fragment);
            transaction.commitAllowingStateLoss();
        }
    };
    fragment.setConferenceObserver(mConferenceObserver);
}
}
```

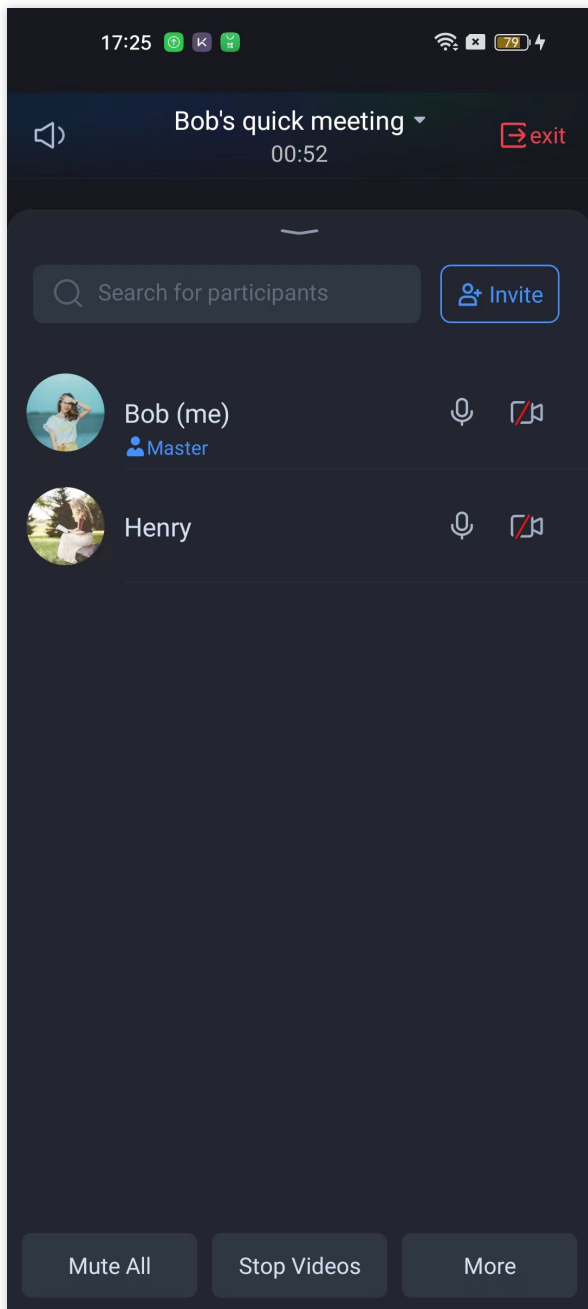
### Note

ConferenceGeneralActivity must inherit FragmentActivity or its subclass, otherwise the interface will not be displayed.

### Step 5: Run

After running the code, you will see the meeting interface as shown below.





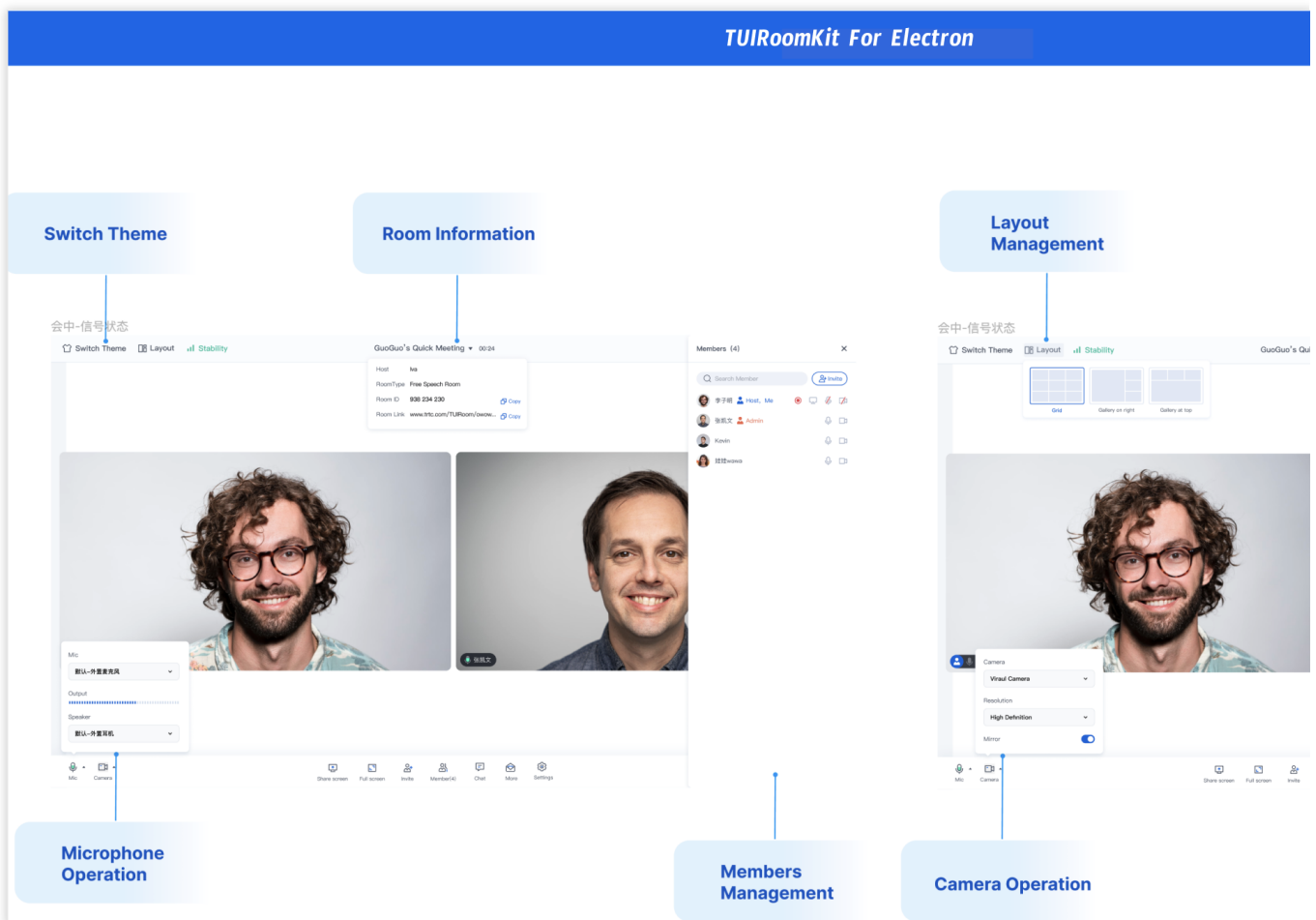
## Suggestions and Feedback

If you have any suggestions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# Electron

Last updated : 2024-04-12 11:29:03

TUIRoomKit is a multiplayer audio/video UI component launched by Tencent Cloud. The component provides room management, audio/video control, screen sharing, member management, mike management, live chat, custom layout switching and other rich functional interactions, as well as support for Chinese and English switching, one-click skinning and other capabilities.



You can click on the Online Experience link: [Mac OS Version](#) and [Windows Version](#) to download and experience more features of TUIRoomKit Electron.

You can also click on [Github](#) to download the TUIRoomKit code and refer to [the TUIRoomKit Electron sample project](#) and run the TUIRoomKit Electron sample project.

If you need to integrate the Electron-side TUIRoomKit Component into your existing business, please refer to this document.



## Supported Platforms

Windows (PC)

Mac

## Step 1: Activate the service

Before initiating a meeting with TUIRoomKit, you need to activate the exclusive multi-person audio and video interaction service for TUIRoomKit on the console. For specific steps, please refer to [Activate Service](#).

## Step 2: Prepare Vue Project Code

Vue3 + Vite + TS

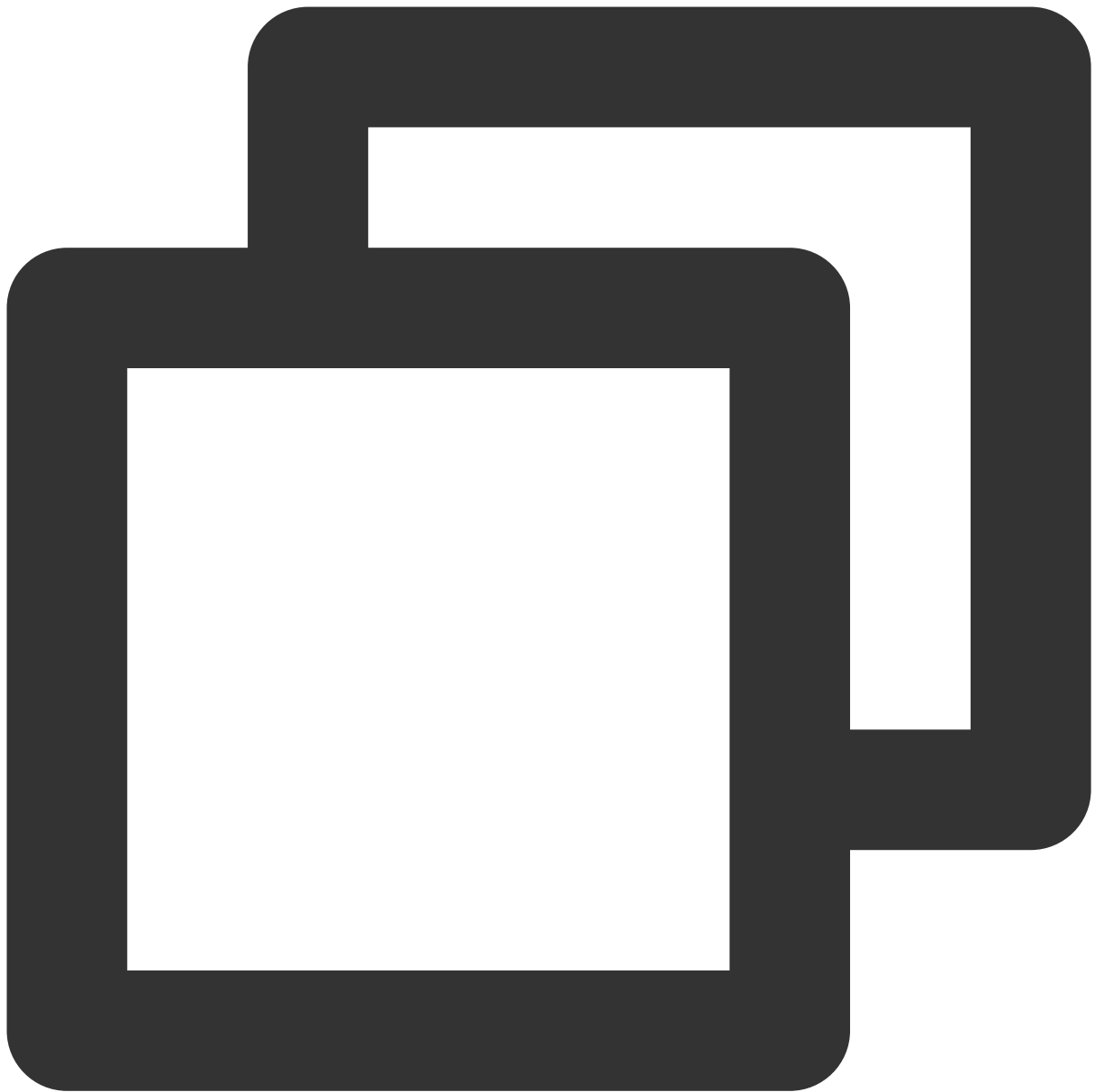
Vue2 + Webpack +TS

1. Open the existing Electron + Vue3 + TS project on the business side. If there is no Electron + Vue3 + TS project, you can generate an Electron + Vue3 + TS template project through this template [Github](#), with a nodejs version greater than 14.17.0.

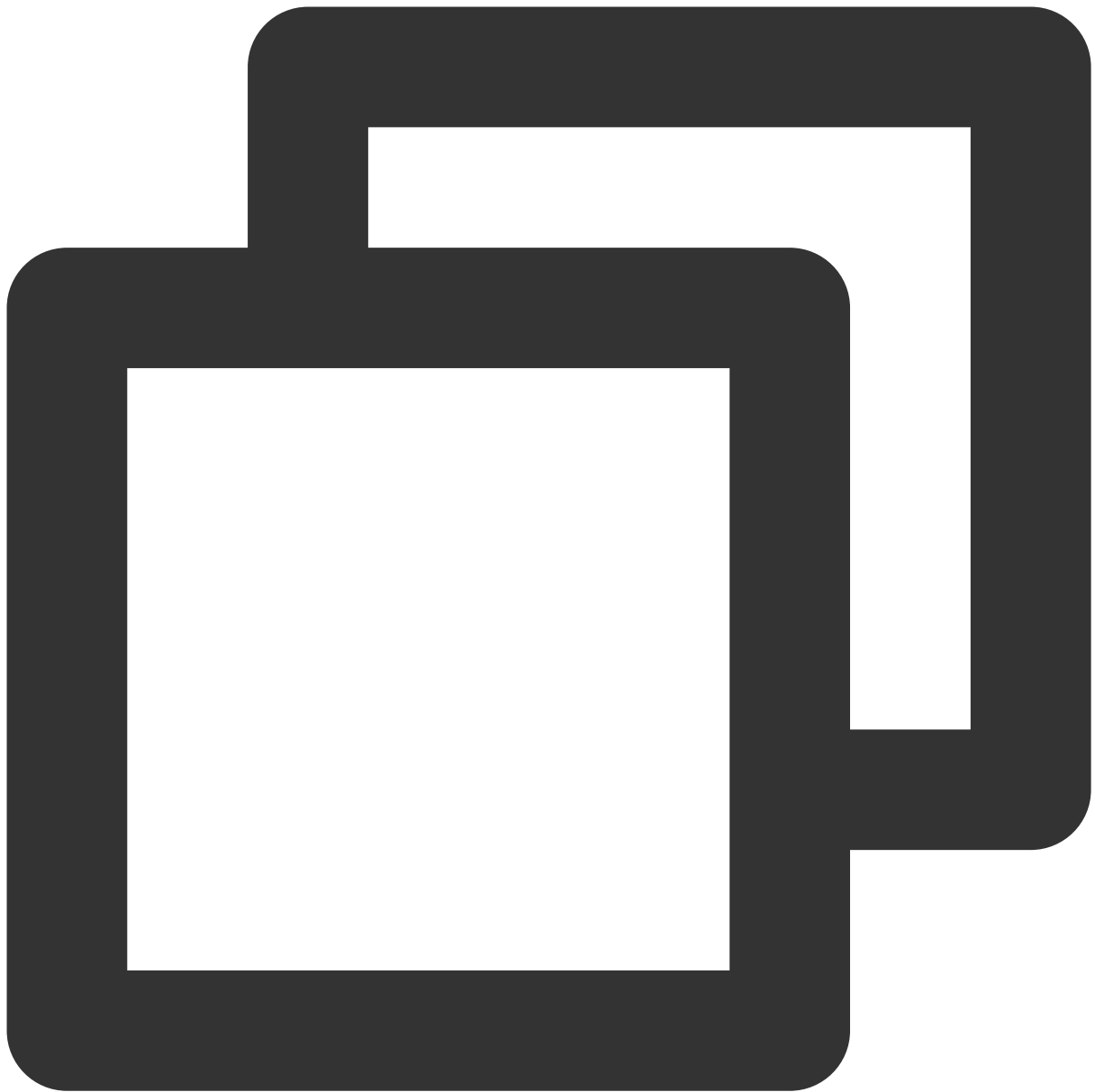
### Note:

The integration steps described in this document are based on the electron-vite-vue template project version 1.0.0. The latest version of the electron-vite-vue template project has adjusted the directory structure. If you need to use the latest version, you can refer to this document to adjust the directory and configuration yourself.

2. After cloning the template project, execute the following Script:



```
git clone https://github.com/electron-vite/electron-vite-vue.git
cd electron-vite-vue
git checkout v1.0.0
git switch -c TUIRoomKit-quick-start
npm install
npm run dev
```



```
// Install vue-cli, note that Vue CLI 4.x requires Node.js to be v10 or higher
npm install -g @vue/cli
// Create a Vue2 + Webpack + TS Template Project
vue create tuiroomkit-demo
```

**Note:**

During the process of executing the template project generation script, select `Manually select features` for the template generation method, and refer to the image for other configuration options.

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, TS, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Use class-style component syntax? No
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)? Yes
? Pick a linter / formatter config: Standard
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
```

After successfully generating the Vue2 + Webpack + TS Template Project, refer to [the Electron UI-less integration document](#) to integrate Electron into the template project.

#### Note:

Since the TUIRoom Component has already introduced trtc-electron-sdk, you can ignore this step when referring to [the Electron UI-less integration document](#).

2. In the project script, import and use the module:

```
const TRTCCloud = require('trtc-electron-sdk').default;
// import TRTCCloud from 'trtc-electron-sdk';
this.rtcCloud = new TRTCCloud();
// Get the SDK version number
this.rtcCloud.getSDKVersion();
```

Since v7.9.348, the TRTC Electron SDK has integrated `trtc.d.ts` for developers using TypeScript.

```
import TRTCCloud from 'trtc-electron-sdk';
const rtcCloud: TRTCCloud = new TRTCCloud();
// Get the SDK version number
rtcCloud.getSDKVersion();
```

## Step 3: Download and Reference TUIRoomkit Component

### 1. Download TUIRoom Component code

Click [Github](#), clone or download the TUIRoomKit repository code.

### 2. Reference TUIRoom Component

Introducing TUIRoom Component in Vue3 Project

## Introducing TUIRoom Component in Vue2 Project

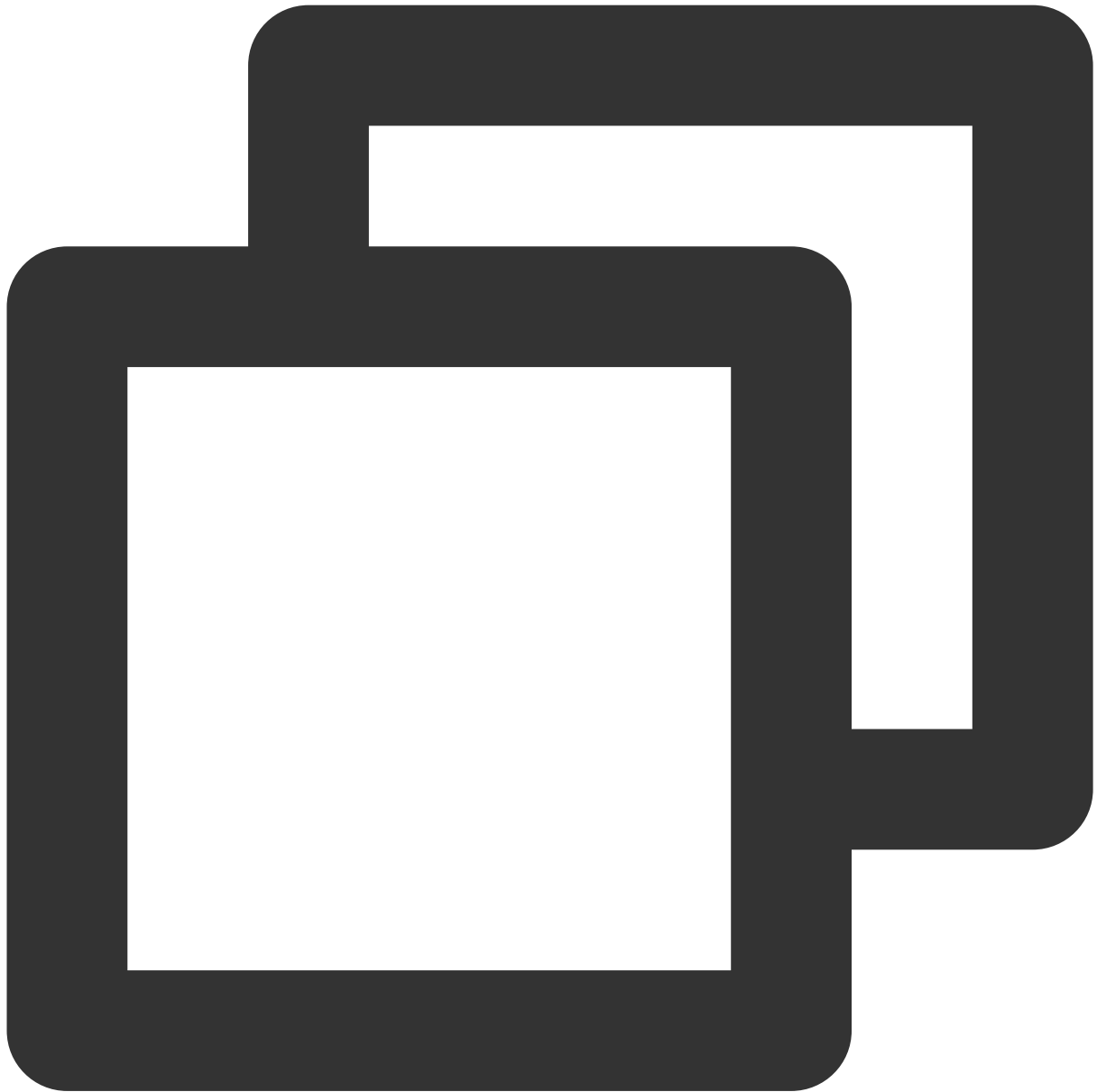
1. Copy the `TUIRoomKit/Electron/vue3/packages/renderer/src/TUIRoom` folder to the existing project `packages/renderer/src/` directory, and copy the

`TUIRoomKit/Electron/vue3/packages/renderer/index.html` folder to the existing project `packages/renderer/` directory.

2. Reference the TUIRoom Component in the page. For example, import the TUIRoom Component in the `App.vue` component.

The TUIRoom Component divides users into Host and Regular Member roles. The Component provides 'init', `createRoom`, and `enterRoom` methods externally.

Both the Host and Regular Members can initialize the application and user data to the TUIRoom Component through the `init` method, the Host can create and join a Room through the `createRoom` method, and Regular Members can join a Room created by the Host through the `enterRoom` method.



```
<template>
  <room ref="TUIRoomRef"></room>
</template>

<script setup lang="ts">
import { ref, onMounted } from 'vue';
// Import TUIRoom Component, make sure the import path is correct
import Room from './TUIRoom/index.vue';
// Get the TUIRoom Component element for calling TUIRoom Component methods
const TUIRoomRef = ref();
```

```
onMounted(async () => {
  // Initialize TUIRoom Component
  // The host needs to initialize the TUIRoom Component before creating a room
  // Ordinary members need to initialize the TUIRoom Component before entering a room
  await TUIRoomRef.value.init({
    // Get sdkAppId, please refer to Step 1
    sdkAppId: 0,
    // The unique identifier of the user in your business
    userId: '',
    // Local development and debugging can quickly generate userSig on https://console.cloud.tencent.com/rtc
    userSig: '',
    // The nickname used by the user in your business
    userName: '',
    // The avatar URL used by the user in your business
    avatarUrl: '',
    // Skin theme colours that users need in your business and whether or not switch theme
    theme: {
      defaultTheme: 'black',
      isSupportSwitchTheme: true
    }
  })
  // By default, create a room. In actual access, you can choose to execute the handleEnterRoom method
  await handleCreateRoom();
})

// The host creates a room, this method is only called when creating a room
async function handleCreateRoom() {
  // roomId is the room number entered by the user, and roomId is required to be of type string
  // roomMode includes 'FreeToSpeak' (Free-to-speak mode) and 'SpeakAfterTakingSeat' (Speak after taking seat mode)
  // roomParam specifies the default behavior of the user when entering the room (when the user enters the room)
  const roomId = '123456';
  const roomMode = 'FreeToSpeak';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
  }
  try {
    await TUIRoomRef.value.createRoom({ roomId, roomName: roomId, roomMode, roomParam })
  } catch (error: any) {
    alert('TUIRoomKit.createRoom error: ' + error.message);
  }
}

// Regular members enter the room, this method is called when regular members enter the room
async function handleEnterRoom() {
  // roomId is the room number entered by the user, and roomId is required to be of type string
  // roomParam specifies the default behavior of the user when entering the room (when the user enters the room)
```

```
const roomId = '123456';
const roomParam = {
  isOpenCamera: true,
  isOpenMicrophone: true,
}
try {
  await TUIRoomRef.value.enterRoom({ roomId, roomParam });
} catch (error: any) {
  alert('TUIRoomKit.enterRoom error: ' + error.message);
}
}
</script>

<style lang="scss">
html, body {
  width: 100%;
  height: 100%;
  margin: 0;
}

#app {
  width: 100%;
  height: 100%;
}
</style>
```

1. Copy the `TUIRoomKit/Electron/vue2/src/TUIRoom` folder to the existing project `src/` directory.
2. Reference the TUIRoom Component in the page. For example, import the TUIRoom Component in the `App.vue` component.

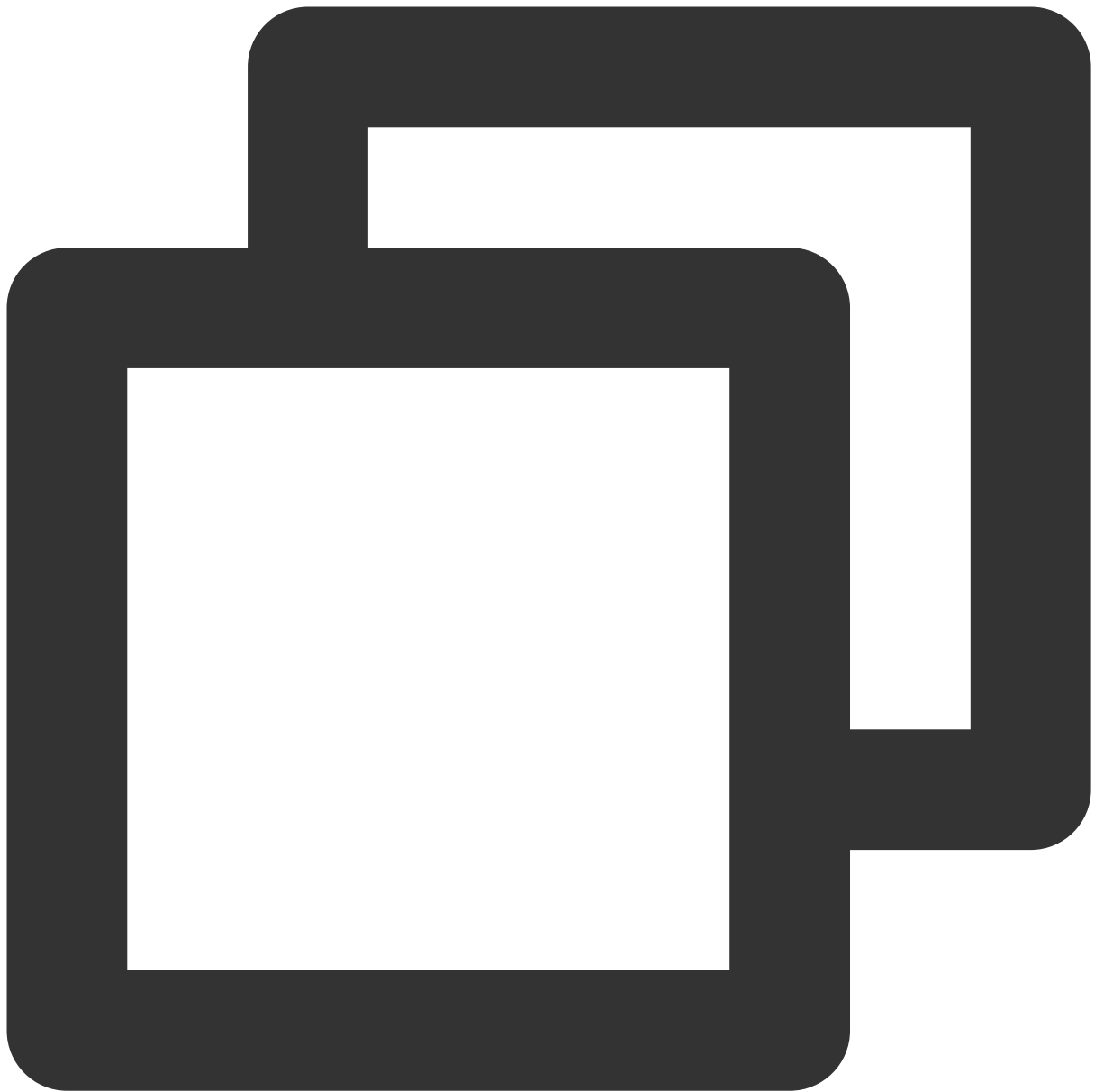
The TUIRoom Component divides users into Host and Regular Member roles. The Component provides [init](#), [createRoom](#), and [enterRoom](#) methods externally.

Both the Host and Regular Members can initialize the application and user data to the TUIRoom Component through the [init](#) method, the Host can create and join a Room through the [createRoom](#) method, and Regular Members can join a Room created by the Host through the [enterRoom](#) method.

**Note:**

After copying the following code to the page, you need to modify the parameters of the TUIRoom interface to the actual data.





```
<template>
  <div id="app">
    <room-container ref="TUIRoomRef"></room-container>
  </div>
</template>

<script>
import RoomContainer from '@/TUIRoom/index.vue';
export default {
  name: 'App',
  components: { RoomContainer },
```

```
data() {
  return {};
},
async mounted() {
  // Initialize TUIRoom Component
  // The host needs to initialize the TUIRoom component before creating a room
  // Regular members need to initialize the TUIRoom component before entering a room
  await this.$refs.TUIRoomRef.init({
    // Get sdkAppId, please refer to Step 1
    sdkAppId: 0,
    // User unique identifier in your business
    userId: '',
    // Local development and debugging can quickly generate userSig on https://console.cloud.tencent.com/rtc
    userSig: '',
    // The nickname used by the user in your business
    userName: '',
    // The avatar link used by the user in your business
    avatarUrl: '',
  });
  // By default, create a room, and actually access it according to the needs of the user
  await this.handleCreateRoom();
},
methods: {
  // The host creates a room, this method is only called when creating a room
  async handleCreateRoom() {
    // roomId is the room number entered by the user, and roomId is required to be a string
    // roomMode includes 'FreeToSpeak' (Free-to-speak mode) and 'SpeakAfterTaking' (Speak after taking mode)
    // roomParam specifies the default behavior of the user when entering the room
    const roomId = '123456';
    const roomMode = 'FreeToSpeak';
    const roomParam = {
      isOpenCamera: true,
      isOpenMicrophone: true,
    };
    try {
      await this.$refs.TUIRoomRef.createRoom({ roomId, roomName: roomId, roomMode, roomParam });
    } catch (error) {
      alert('TUIRoomKit.createRoom error: ' + error.message);
    }
  },
  // Regular members enter the room, this method is called when regular members enter the room
  async handleEnterRoom() {
    // roomId is the room number entered by the user, and roomId is required to be a string
    // roomParam specifies the default behavior of the user when entering the room
    const roomId = '123456';
    const roomParam = {
      isOpenCamera: true,
```

```
        isOpenMicrophone: true,
    }
    try {
        await this.$refs.TUIRoomRef.enterRoom({ roomId, roomParam });
    } catch (error) {
        alert('TUIRoomKit.enterRoom error: ' + error.message);
    }
}
},
};

</script>

<style lang="scss">
html, body {
    width: 100%;
    height: 100%;
    margin: 0;
}

#app {
    width: 100%;
    height: 100%;
    * {
        box-sizing: border-box;
    }
}
</style>
```

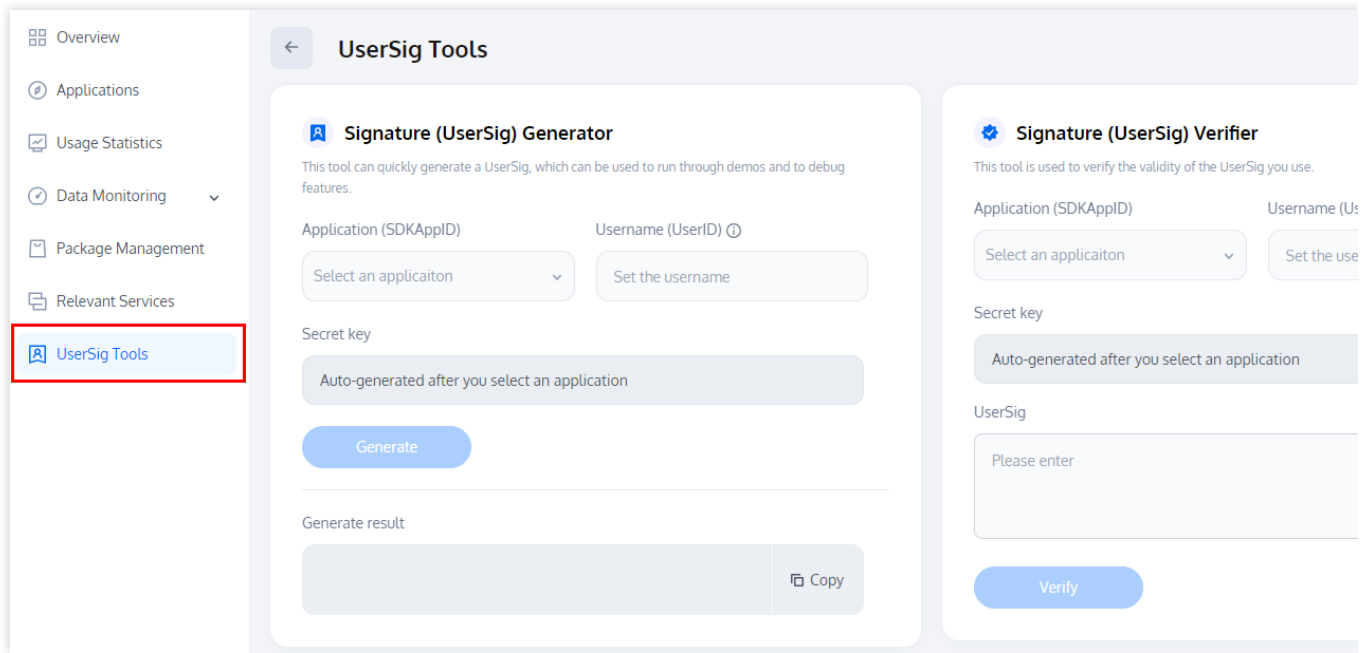
### Parameter Description

Here is a detailed introduction to the key parameters used in the login function:

**SDKAppID** : You have already obtained it in [Step 1](#), so it will not be repeated here.

**UserID** : The ID of the current user, string type, only allows to contain English letters (a-z and A-Z), numbers (0-9), hyphens (-), and underscores (\_).

**UserSig** : Encrypt the SDKAppID, UserID, etc. with the SDKSecretKey obtained in [Step 1](#) to get the UserSig, which is a ticket for authorization and is used for Tencent Cloud to recognize whether the current user can use the TRTC service. You can create a temporarily available UserSig through the UserSig Tools through the project sidebar in the console.



For more information, please refer to the [UserSig related](#).

#### Note:

**This step is also the step with the most feedback from developers we have received so far. Common problems are as follows:**

SDKAppID is set incorrectly. Please use the SDKAppID of the international site correctly, otherwise, you will not be able to access it.

UserSig is misconfigured as an encryption key (SDKSecretKey). UserSig is obtained by encrypting the SDKAppID, UserID, and expiration time with the SDKSecretKey, not by directly configuring the SDKSecretKey as UserSig.

UserID is set to simple strings like "1", "123", "111", etc. **Since TRTC does not support multi-terminal login with the same UserID**, simple UserIDs like "1", "123", "111" are easily occupied by your colleagues, causing login failure. Therefore, we recommend that you set some UserIDs with high identifiability when debugging.

The sample code in Github uses the `genTestUserSig` function to calculate UserSig locally to quickly get you through the current access process. However, this solution exposes your SDKSecretKey in the App code, which is not conducive to your subsequent upgrades and protection of your SDKSecretKey. Therefore, we strongly recommend that you put the calculation logic of UserSig on the server side and have the app request the real-time calculated UserSig from your server every time it uses the TUIRoomKit Component.

## Step 4: Configure the Development Environment

After the TUIRoom Component is introduced, in order to ensure that the project can run normally, the following configurations need to be made:

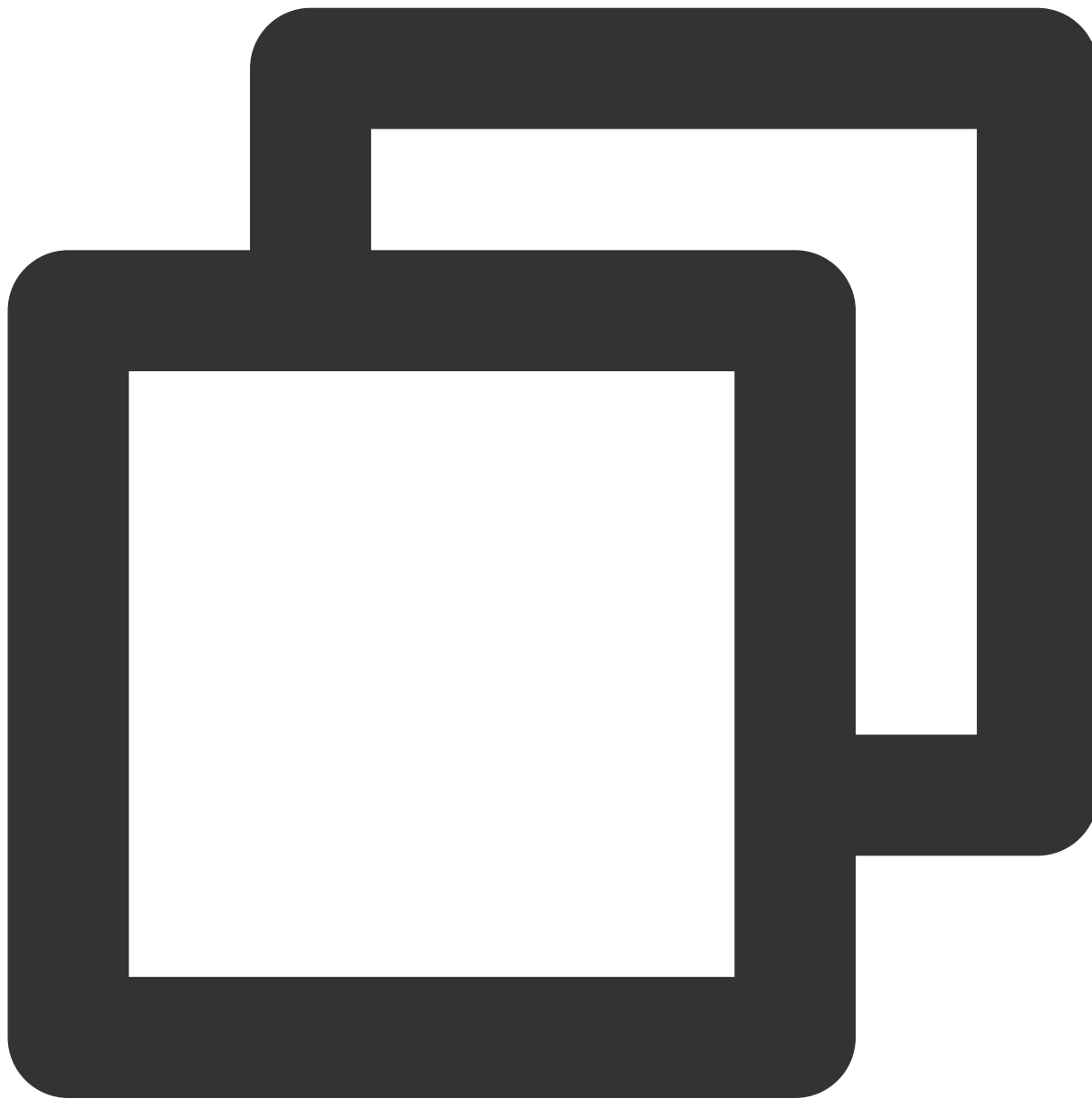
Setting up Vue3 + Vite + TS Development Environment

Setting up Vue2 + Webpack + TS Development Environment

Setting up Vue2 + Webpack + JS Development Environment

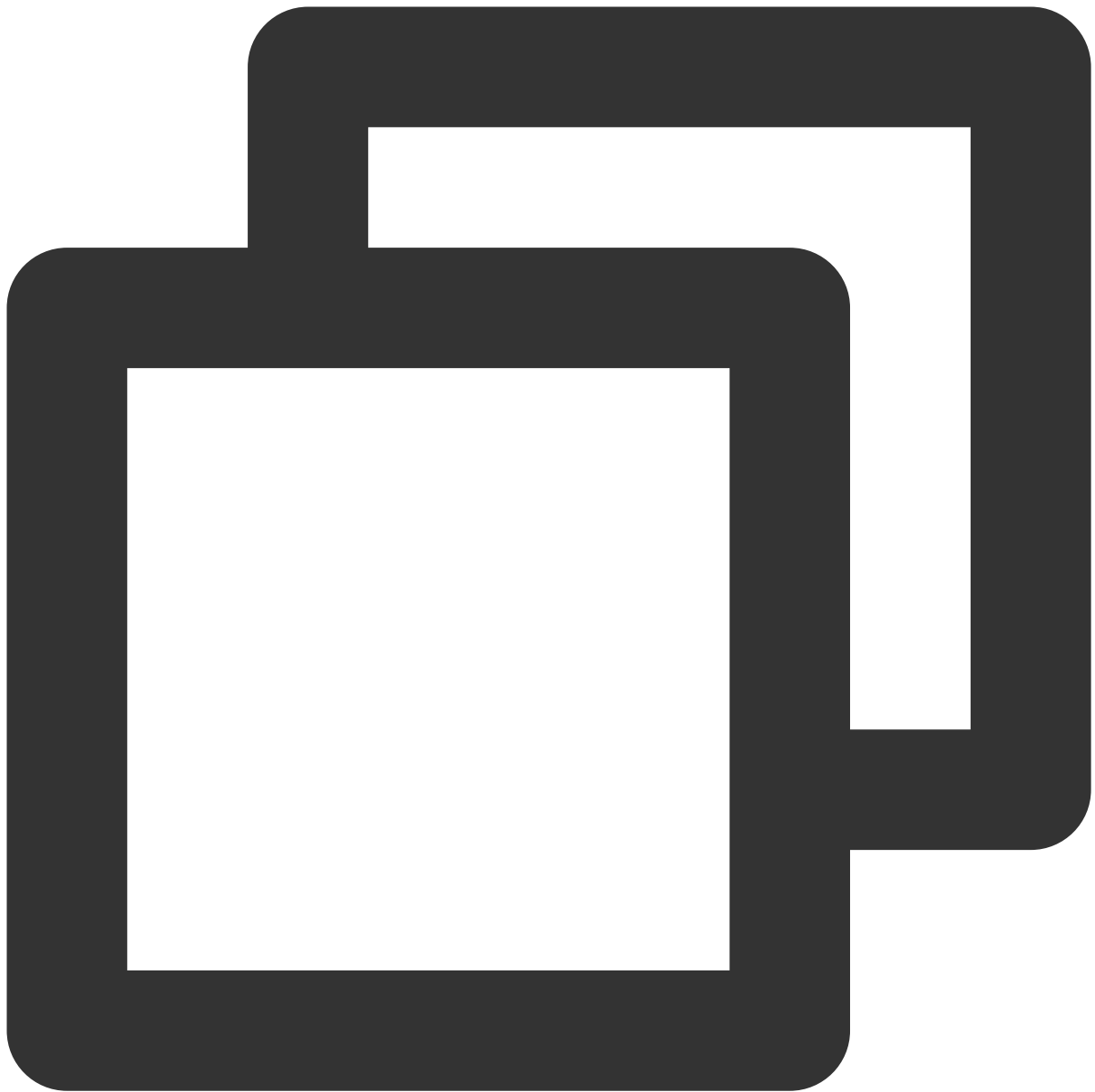
## 1. Install Dependencies

Install Development Environment Dependencies:



```
npm install sass typescript unplugin-auto-import unplugin-vue-components --save-dev
```

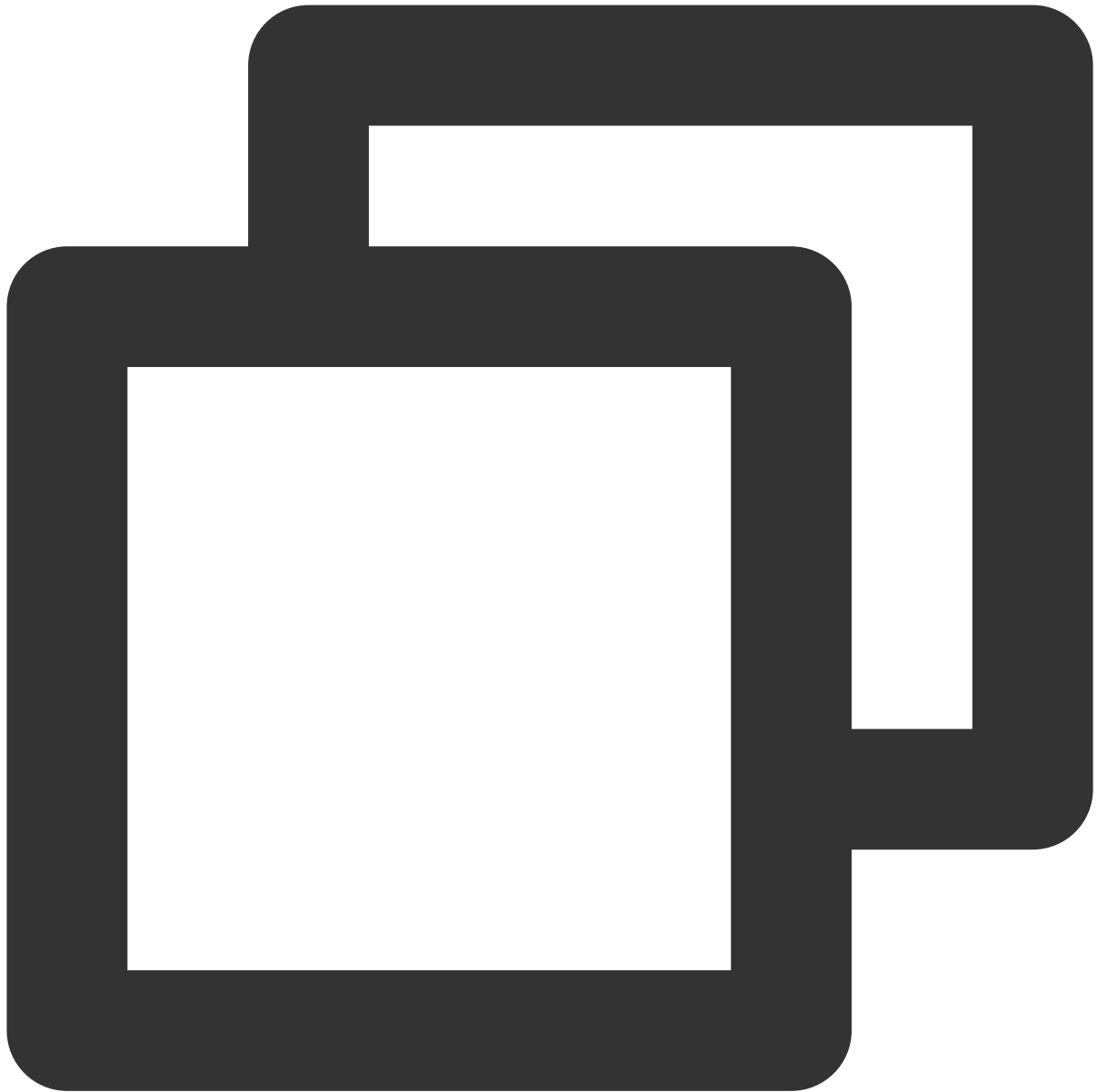
Install Production Environment Dependencies:



```
npm install events mitt pinia trtc-electron-sdk tim-js-sdk tsignaling vue-i18n @ten
```

## 2. Register Pinia

TUIRoom uses Pinia for room data management, you need to register Pinia in the project entry file, which is the `packages/renderer/src/main.ts` file.



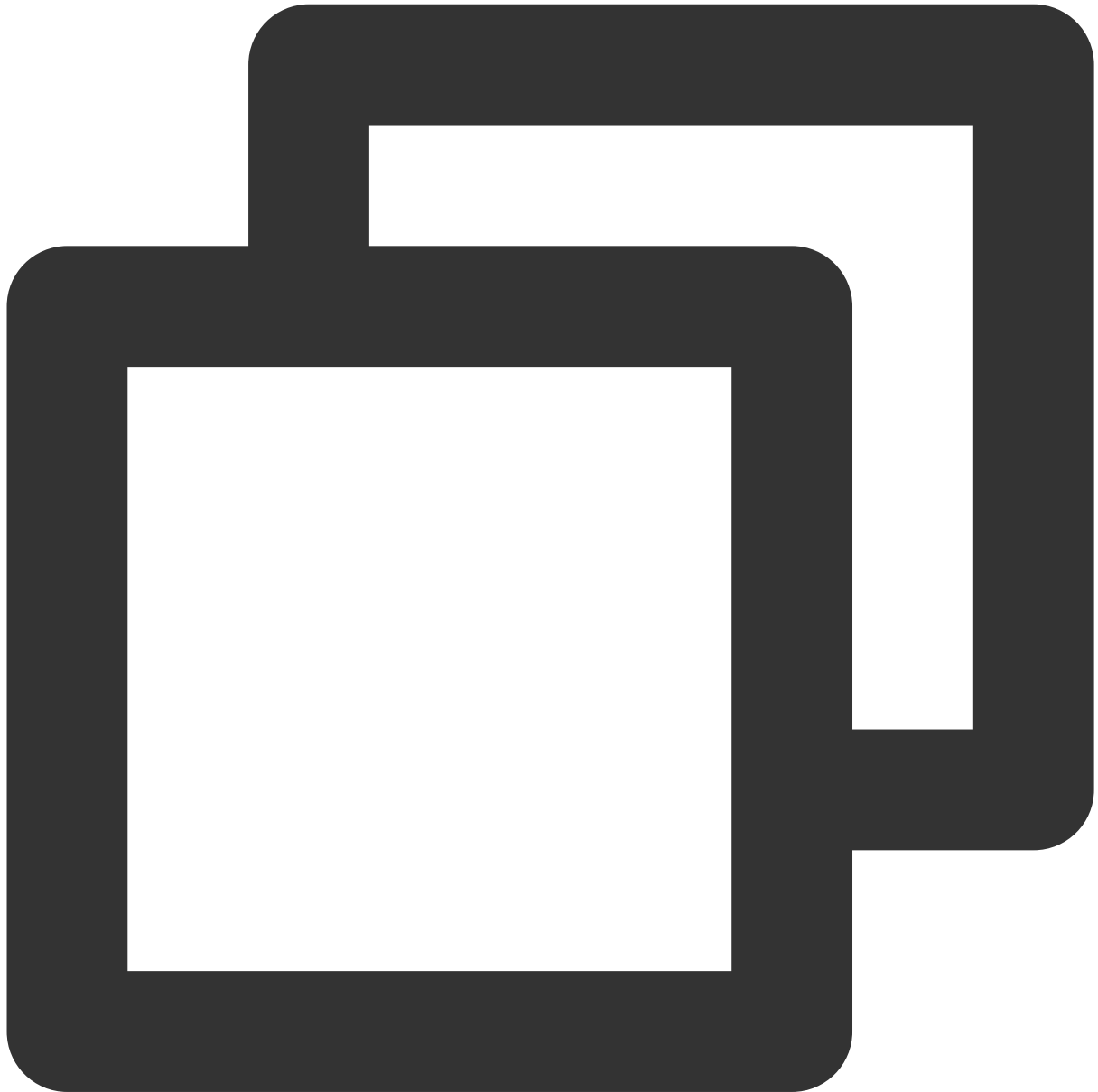
```
// src/main.ts file
import { createPinia } from 'pinia';

const app = createApp(App);
// Register Pinia
createApp(App)
  .use(createPinia())
  .mount('#app')
  .$nextTick(window.removeLoading)
```

### 3. Import trtc-electron-sdk

In order to import trtc-electron-sdk in the UI layer using the import method, to unify the code style, otherwise, it must be imported using the require method, you need to configure it in packages/renderer/vite.config.ts. Replace the content in resolve with the following configuration items, you can refer to

packages/renderer/vite.config.ts [file](#).



```
// vite.config.ts

export default defineConfig({
  // ...
  plugins: [
    resolve(
```

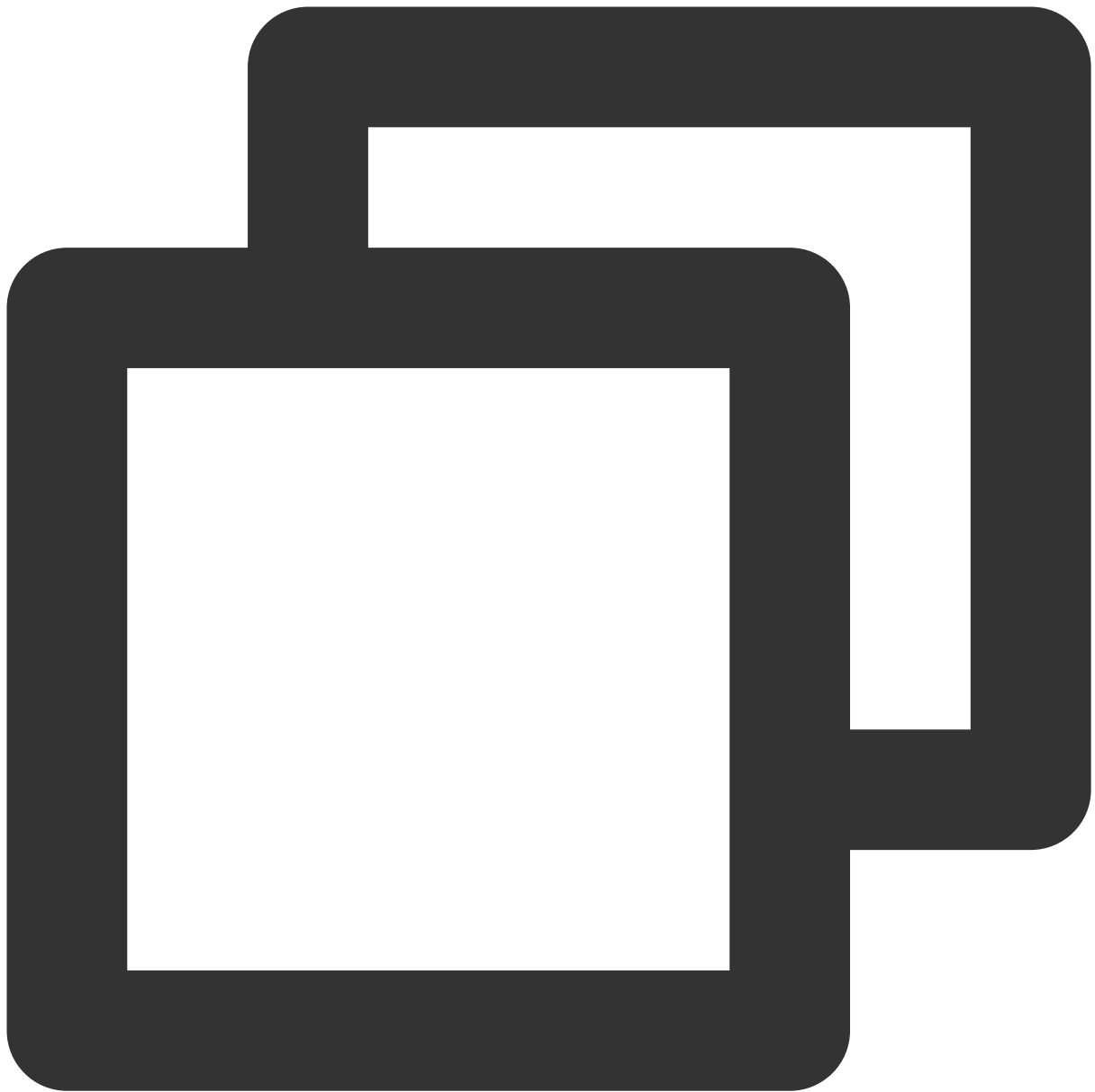


```
{
  "trtc-electron-sdk": `
    const TRTCCloud = require("trtc-electron-sdk");
    const TRTCParams = TRTCCloud.TRTCParams;
    const TRTCAppScene = TRTCCloud.TRTCAppScene;
    const TRTCVideoStreamType = TRTCCloud.TRTCVideoStreamType;
    const TRTCScreenCaptureSourceType = TRTCCloud.TRTCScreenCaptureSourceType;
    const TRTCVideoEncParam = TRTCCloud.TRTCVideoEncParam;
    const Rect = TRTCCloud.Rect;
    const TRTCAudioQuality = TRTCCloud.TRTCAudioQuality;
    const TRTCScreenCaptureSourceInfo = TRTCCloud.TRTCScreenCaptureSourceInfo;
    const TRTCDeviceInfo = TRTCCloud.TRTCDeviceInfo;
    const TRTCVideoQosPreference = TRTCCloud.TRTCVideoQosPreference;
    const TRTCQualityInfo = TRTCCloud.TRTCQualityInfo;
    const TRTCQuality = TRTCCloud.TRTCQuality;
    const TRTCStatistics = TRTCCloud.TRTCStatistics;
    const TRTCVolumeInfo = TRTCCloud.TRTCVolumeInfo;
    const TRTCDeviceType = TRTCCloud.TRTCDeviceType;
    const TRTCDeviceState = TRTCCloud.TRTCDeviceState;
    const TRTCBeautyStyle = TRTCCloud.TRTCBeautyStyle;
    const TRTCVideoResolution = TRTCCloud.TRTCVideoResolution;
    const TRTCVideoResolutionMode = TRTCCloud.TRTCVideoResolutionMode;
    const TRTCVideoMirrorType = TRTCCloud.TRTCVideoMirrorType;
    const TRTCVideoRotation = TRTCCloud.TRTCVideoRotation;
    const TRTCVideoFillMode = TRTCCloud.TRTCVideoFillMode;
    const TRTCRoleType = TRTCCloud.TRTCRoleType;
    const TRTCScreenCaptureProperty = TRTCCloud.TRTCScreenCaptureProperty;
    export {
      TRTCParams,
      TRTCAppScene,
      TRTCVideoStreamType,
      TRTCScreenCaptureSourceType,
      TRTCVideoEncParam,
      Rect,
      TRTCAudioQuality,
      TRTCScreenCaptureSourceInfo,
      TRTCDeviceInfo,
      TRTCVideoQosPreference,
      TRTCQualityInfo,
      TRTCStatistics,
      TRTCVolumeInfo,
      TRTCDeviceType,
      TRTCDeviceState,
      TRTCBeautyStyle,
      TRTCVideoResolution,
      TRTCVideoResolutionMode,
      TRTCVideoMirrorType,
```

```
        TRTCVideoRotation,  
        TRTCVideoFillMode,  
        TRTCRoleType,  
        TRTCQuality,  
        TRTCScreenCaptureProperty,  
    };  
    export default TRTCCloud.default;  
    ,  
    }  
    ),  
    // ...  
    ]  
    // ...  
    });
```

#### 4. env.d.ts File Configuration

env.d.ts file configuration needs to be configured in `packages/renderer/src/env.d.ts`. The following configuration items are incremental configurations, do not delete the existing `env.d.ts` file configuration.



```
// env.d.ts
declare module '*.vue' {
  import { DefineComponent } from 'vue'
  // eslint-disable-next-line @typescript-eslint/no-explicit-any, @typescript-eslint-
  const component: DefineComponent<{}, {}, any>
  export default component
}

declare module 'tsignaling/tsignaling-js' {
  import TSignaling from 'tsignaling/tsignaling-js';
  export default TSignaling;
```

```
}

declare module 'tim-js-sdk' {
  import TIM from 'tim-js-sdk';
  export default TIM;
}

declare const Aegis: any;
```

## 5. Configure Chinese-English language switching

TUIRoom currently supports Chinese-English language switching, you need to register the i18n instance in the main.ts file.

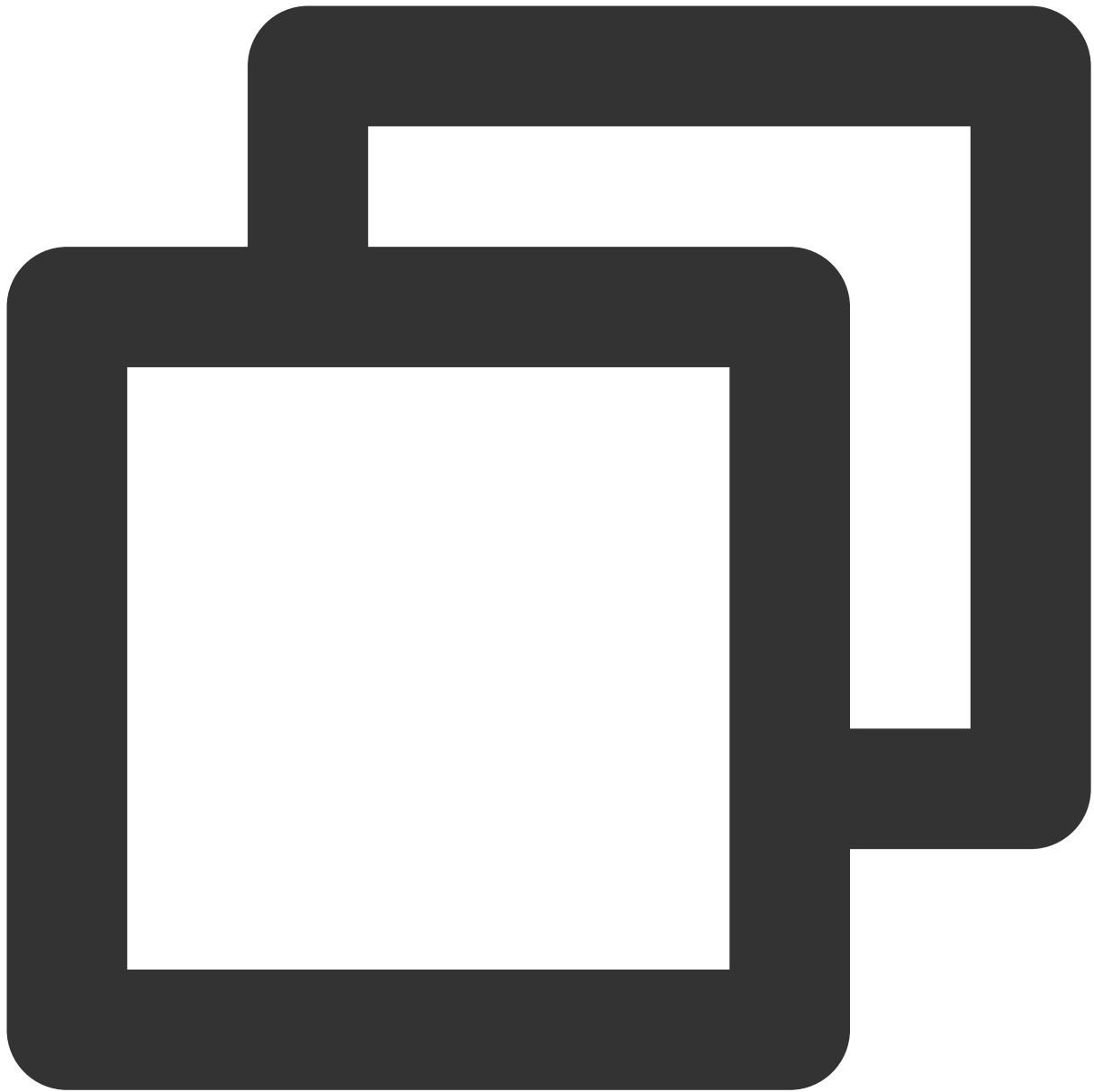


```
// src/main.ts
// Import locales/index.ts file, Please confirm whether the import path is correct
import VueI18n from './TUIRoom/locales/index';

createApp(App)
  .use(createPinia())
  .use(VueI18n)
  .mount('#app')
  .$nextTick(window.removeLoading);
```

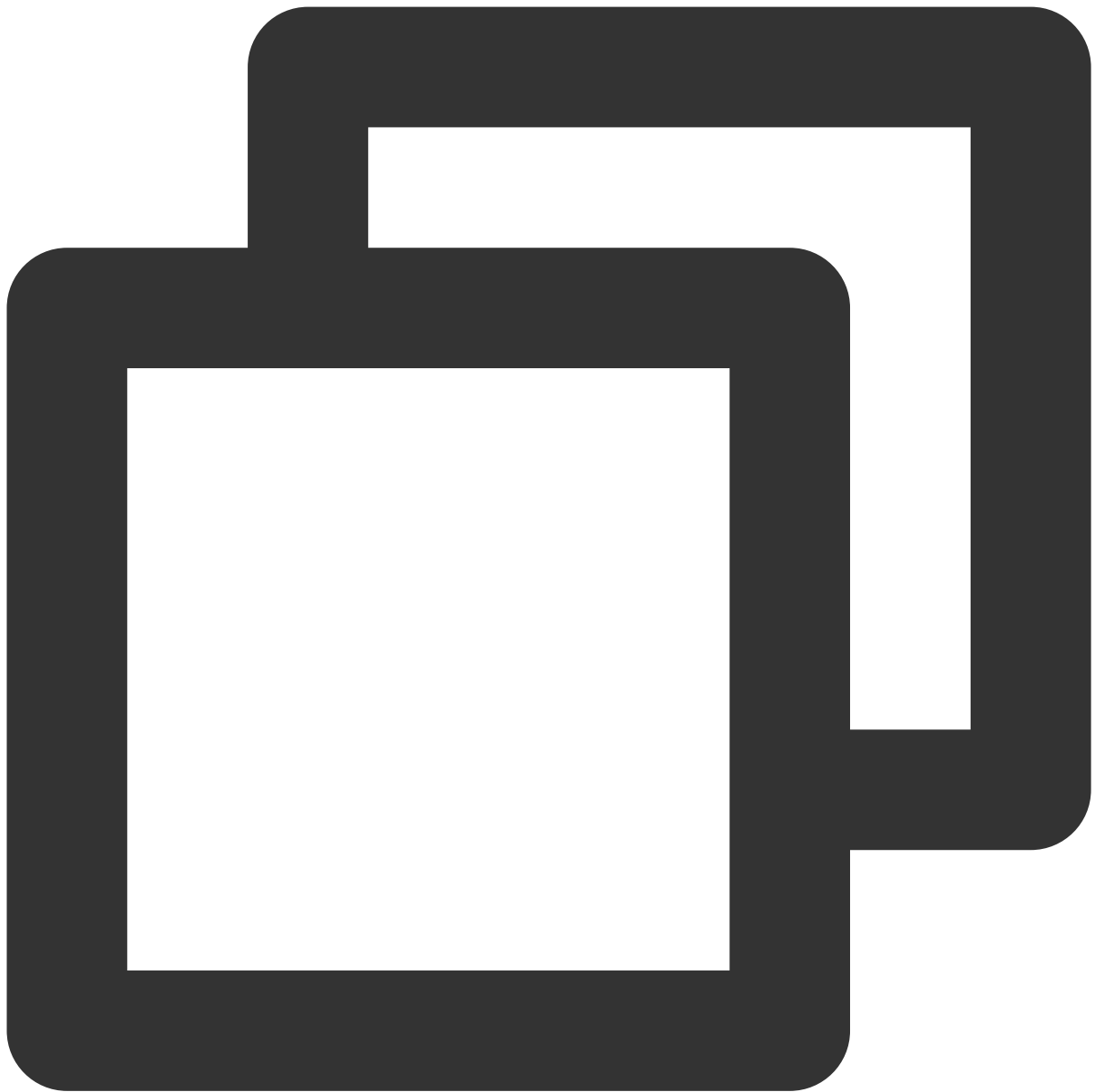
## 1. Install Dependencies

## Install Development Environment Dependencies



```
npm install sass sass-loader -S -D
```

## Install Production Environment Dependencies

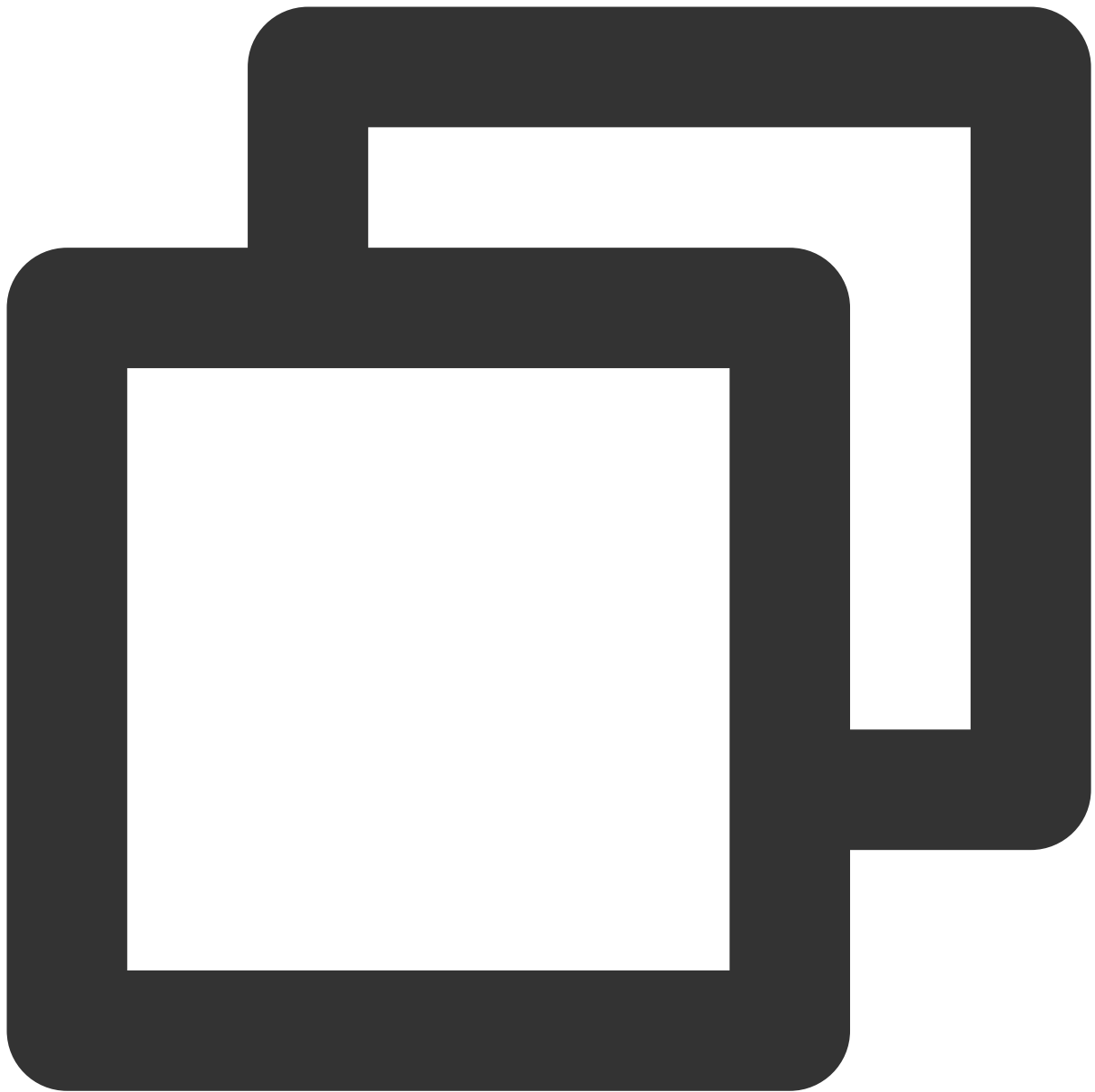


```
npm install vue@^2.7 pinia trtc-electron-sdk tim-js-sdk tsignaling @tencentcloud/tu
```

## 2. Register Pinia

TUIRoom uses Pinia for room data management, you need to register Pinia in the project entry file, which is the

`src/main.ts` file.



```
import { createPinia, PiniaVuePlugin } from 'pinia';

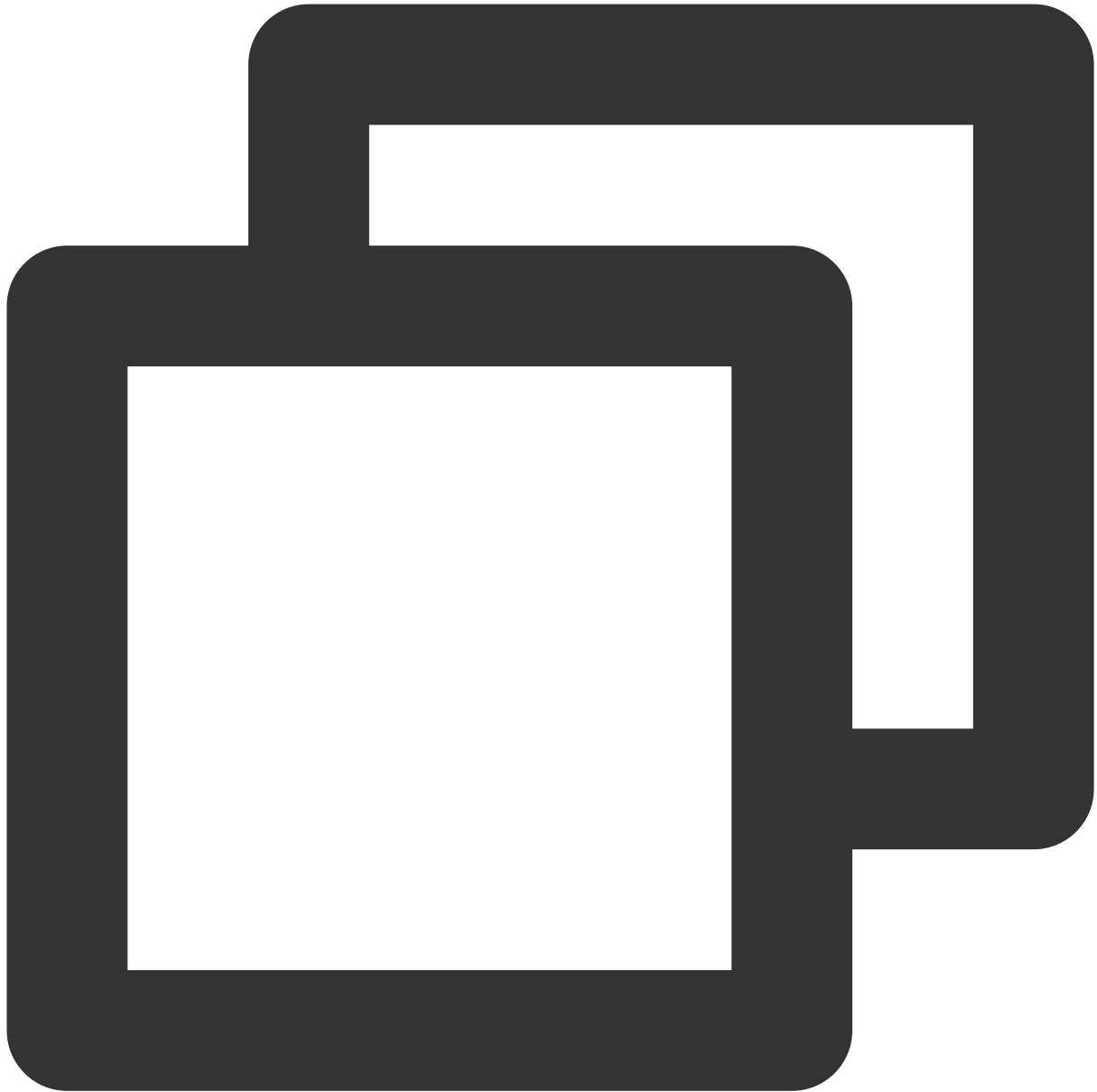
Vue.use(PiniaVuePlugin);
const pinia = createPinia();

new Vue({
  pinia,
  render: h => h(App),
}).$mount('#app');
```

### 3. Configure Chinese-English language switching



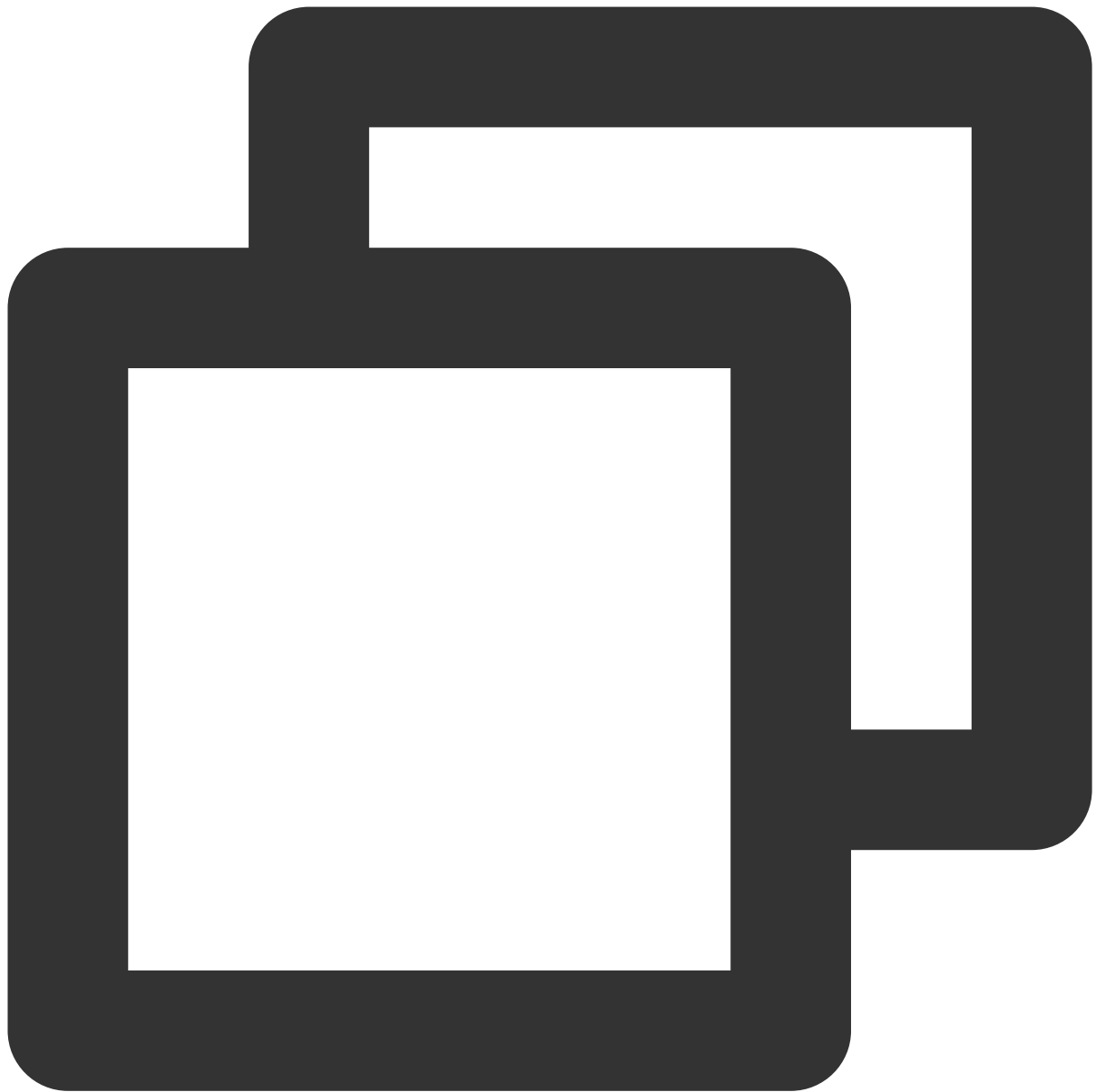
TUIRoom currently supports Chinese-English language switching, you need to register the i18n instance in the main.ts file.



```
import i18n from './TUIRoom/locales/';  
  
Vue.use(i18n);
```

TUIRoom's default support environment is Vue2 + Webpack + TS. If you want to integrate TUIRoom Component in a Vue2 + Webpack + JS environment, you need to do the following setup.

1. Set up typescript to support TUIRoom Component loading.



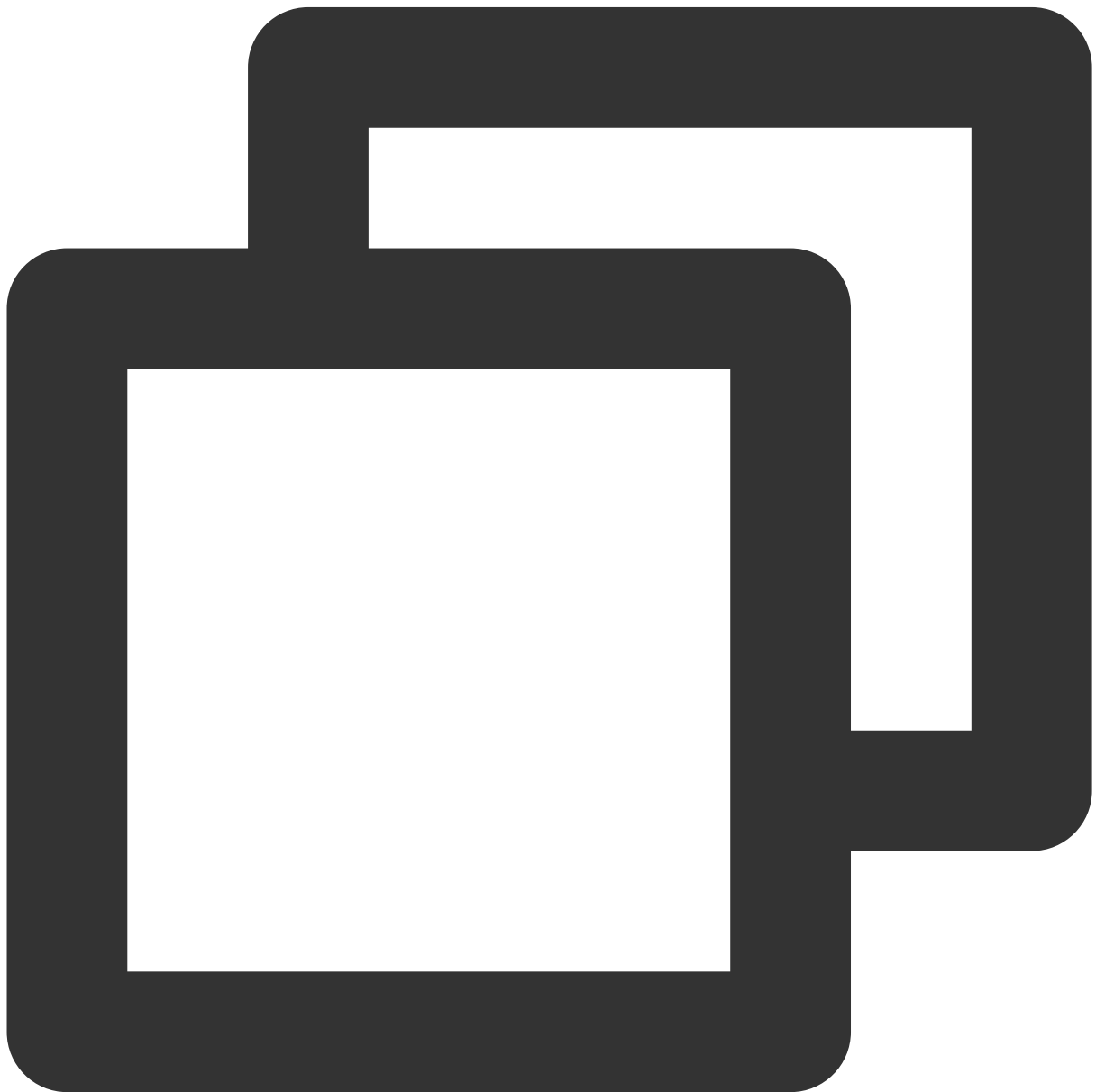
```
vue add typescript
```

Options for setting up the TS development environment can be referred to the image.

```
? Use class-style component syntax? No
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)? Yes
? Convert all .js files to .ts? No
? Allow .js files to be compiled? Yes
? Skip type checking of all declaration files (recommended for apps)? No
```

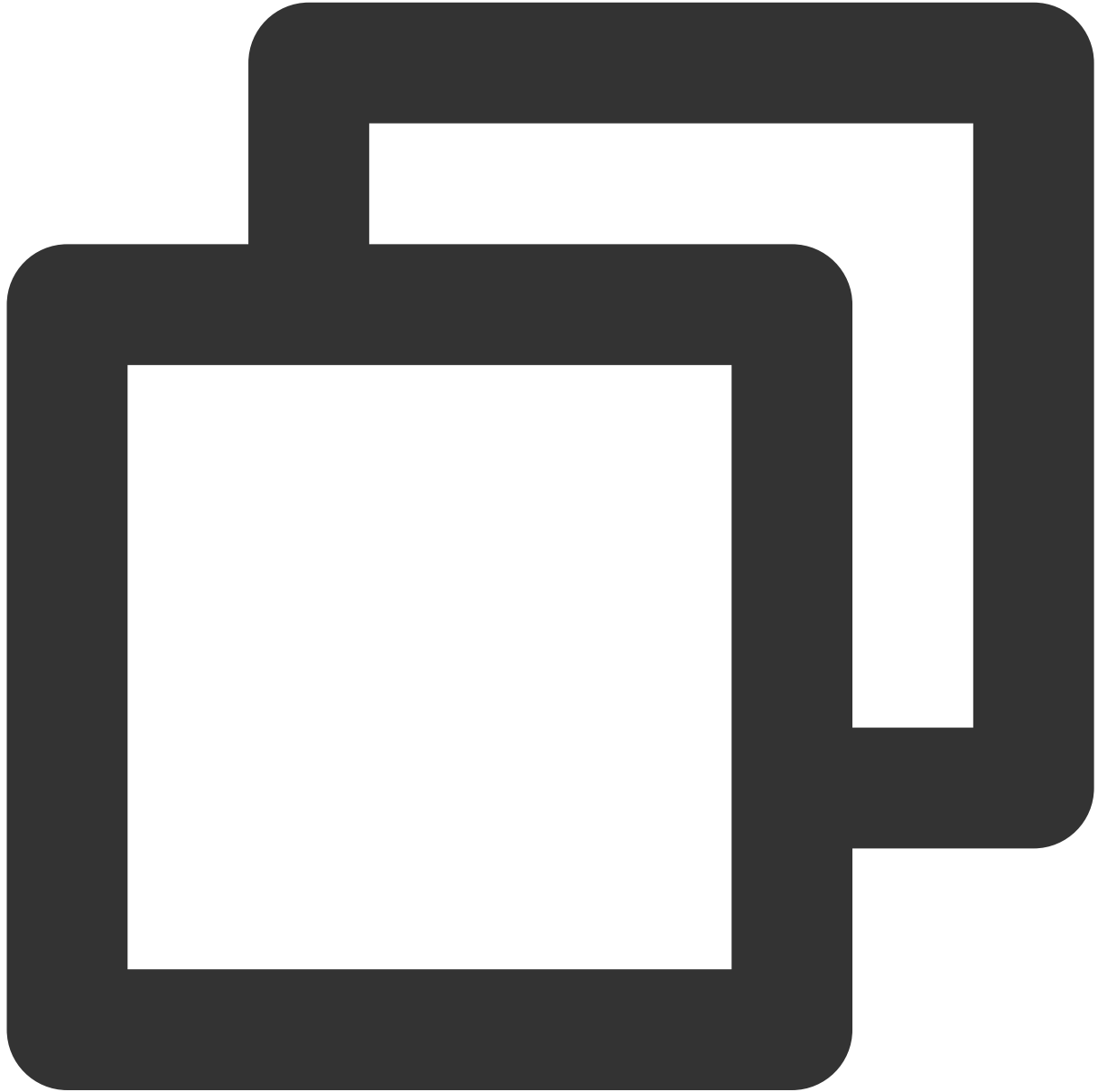
## 2. Install Dependencies

Install Development Environment Dependencies



```
npm install sass sass-loader -S -D
```

Install Production Environment Dependencies

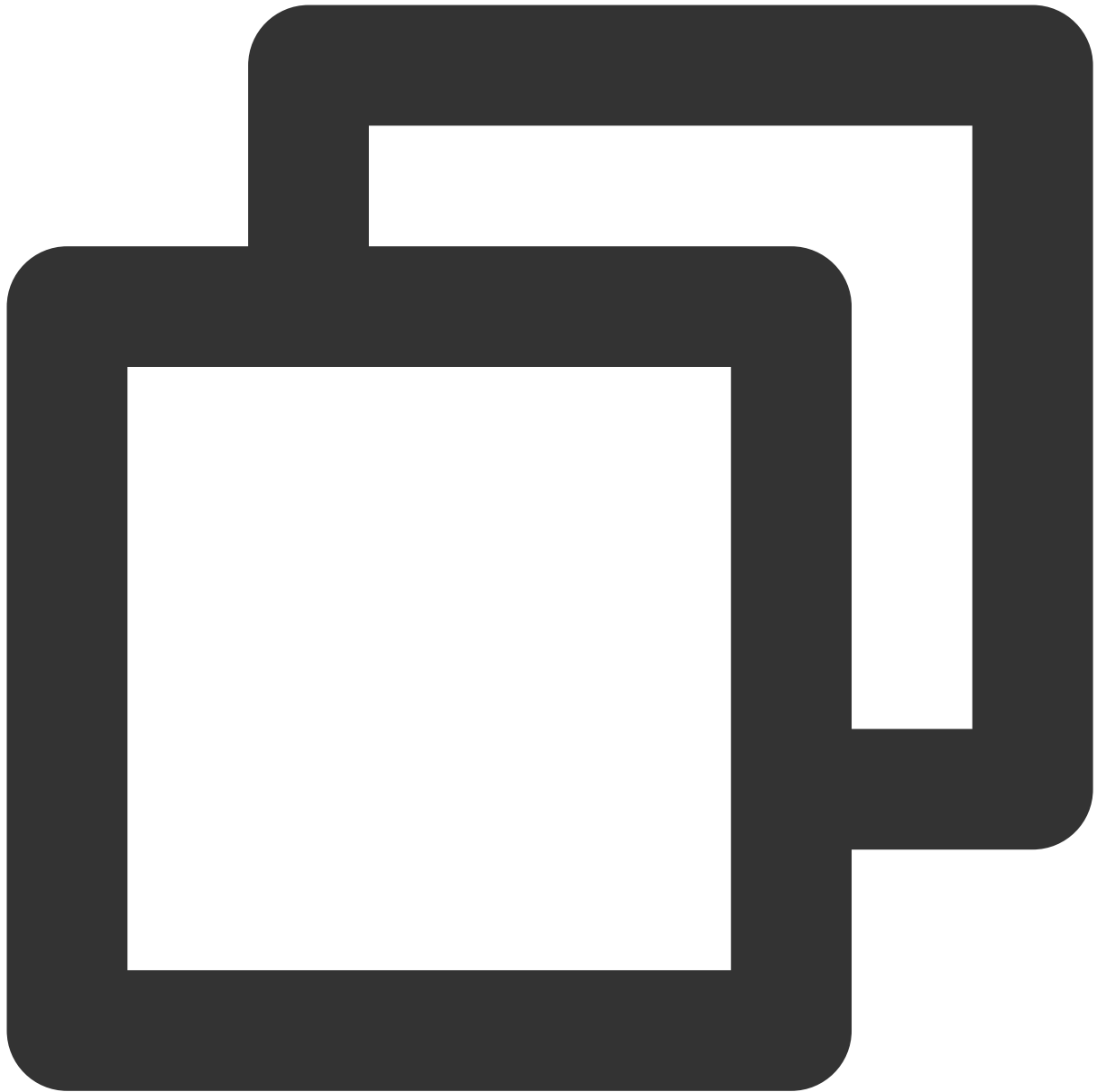


```
npm install vue@^2.7 pinia trtc-electron-sdk tim-js-sdk tsignaling @tencentcloud/tu
```

### 3. Register Pinia

TUIRoom uses Pinia for room data management, you need to register Pinia in the project entry file, which is the

`src/main.ts` file.



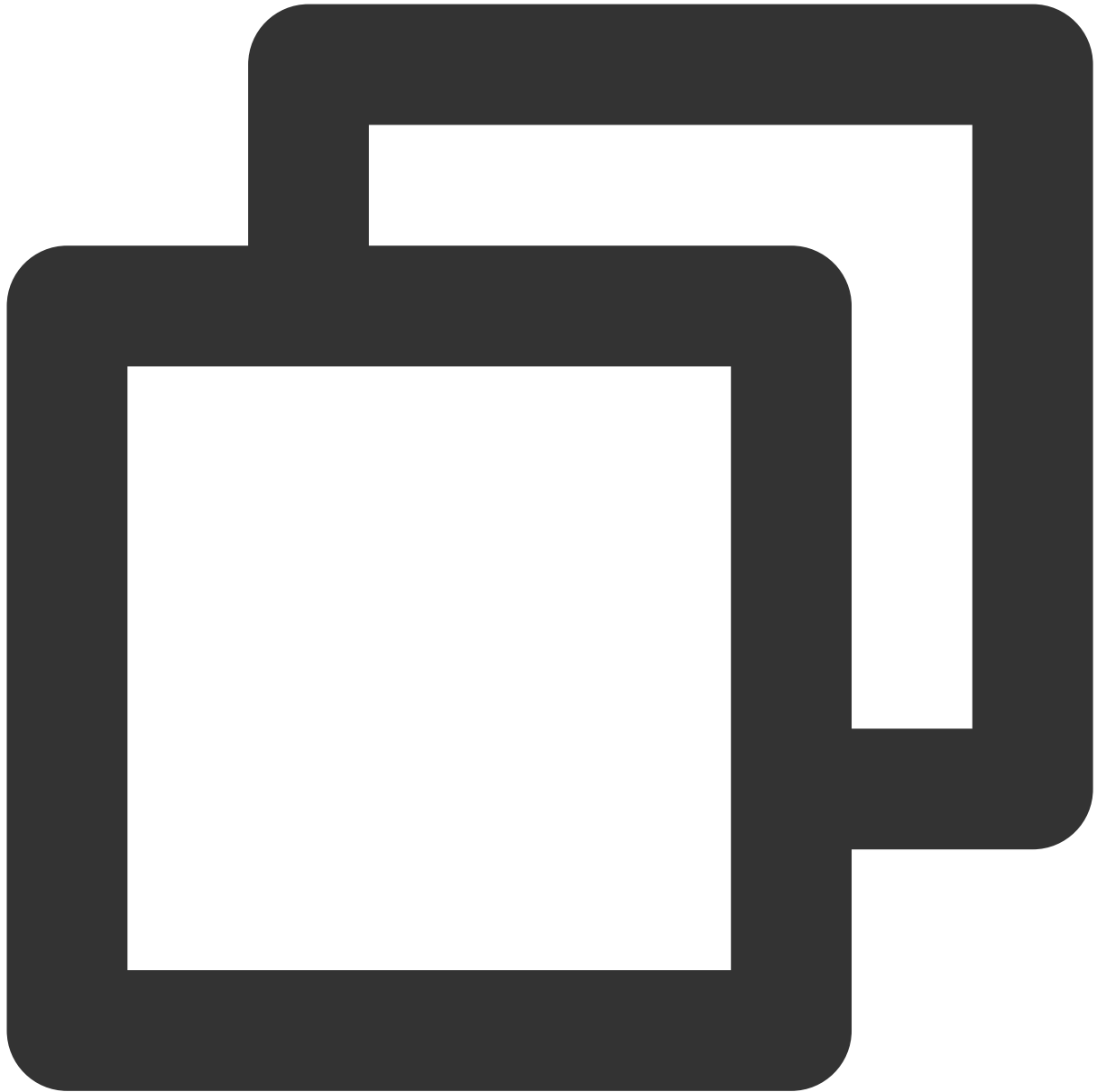
```
import { createPinia, PiniaVuePlugin } from 'pinia';

Vue.use(PiniaVuePlugin);
const pinia = createPinia();

new Vue({
  pinia,
  render: h => h(App),
}).$mount('#app');
```

#### 4. Configure Chinese-English language switching

TUIRoom currently supports Chinese-English language switching, you need to register the i18n instance in the main.ts file.



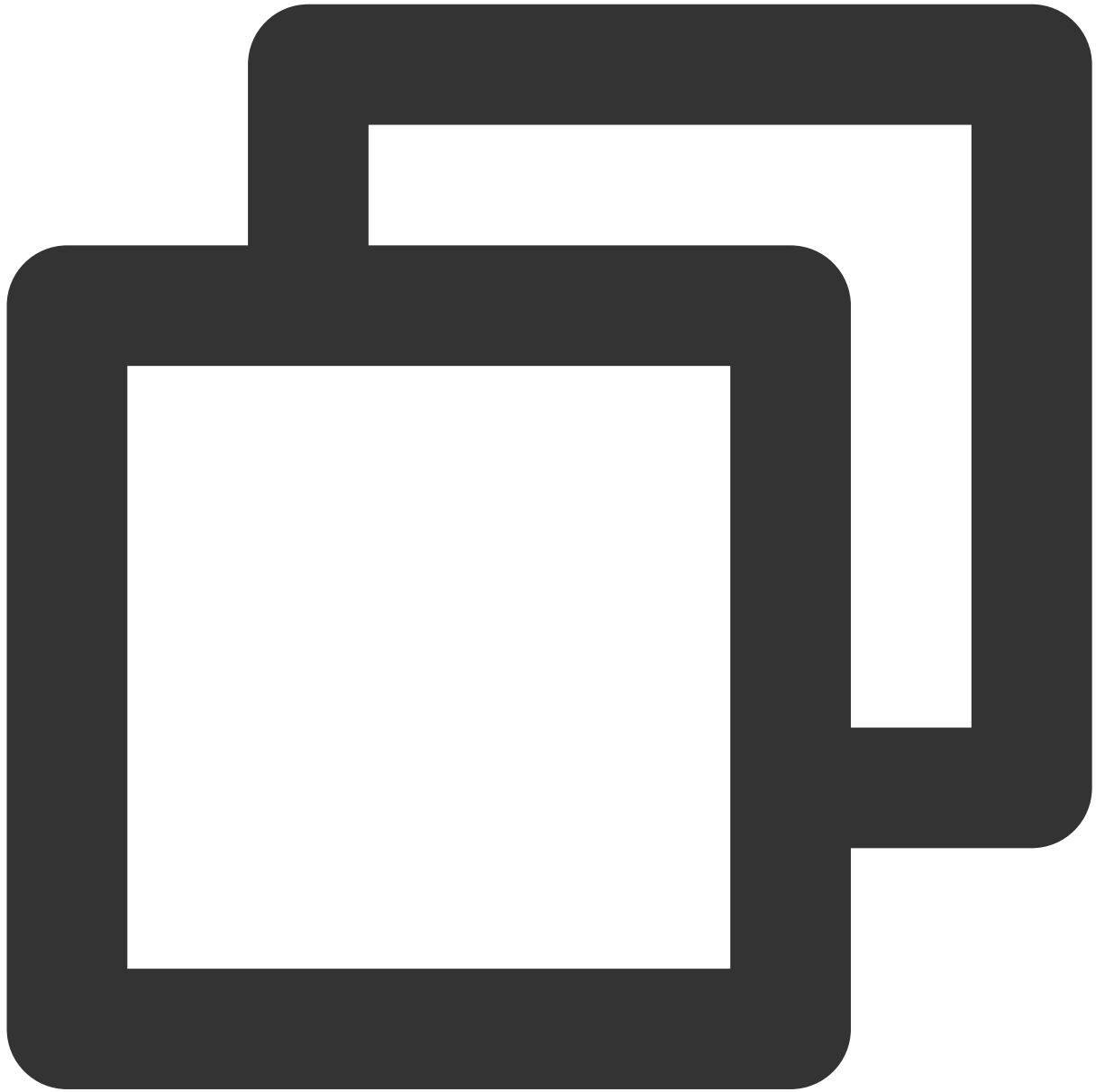
```
import i18n from './TUIRoom/locales/';  
  
Vue.use(i18n);
```

## Step 5: Run the development environment

Vue3 + Vite + TS Project

Vue2 + Webpack + TS Project

1. Execute the development environment command



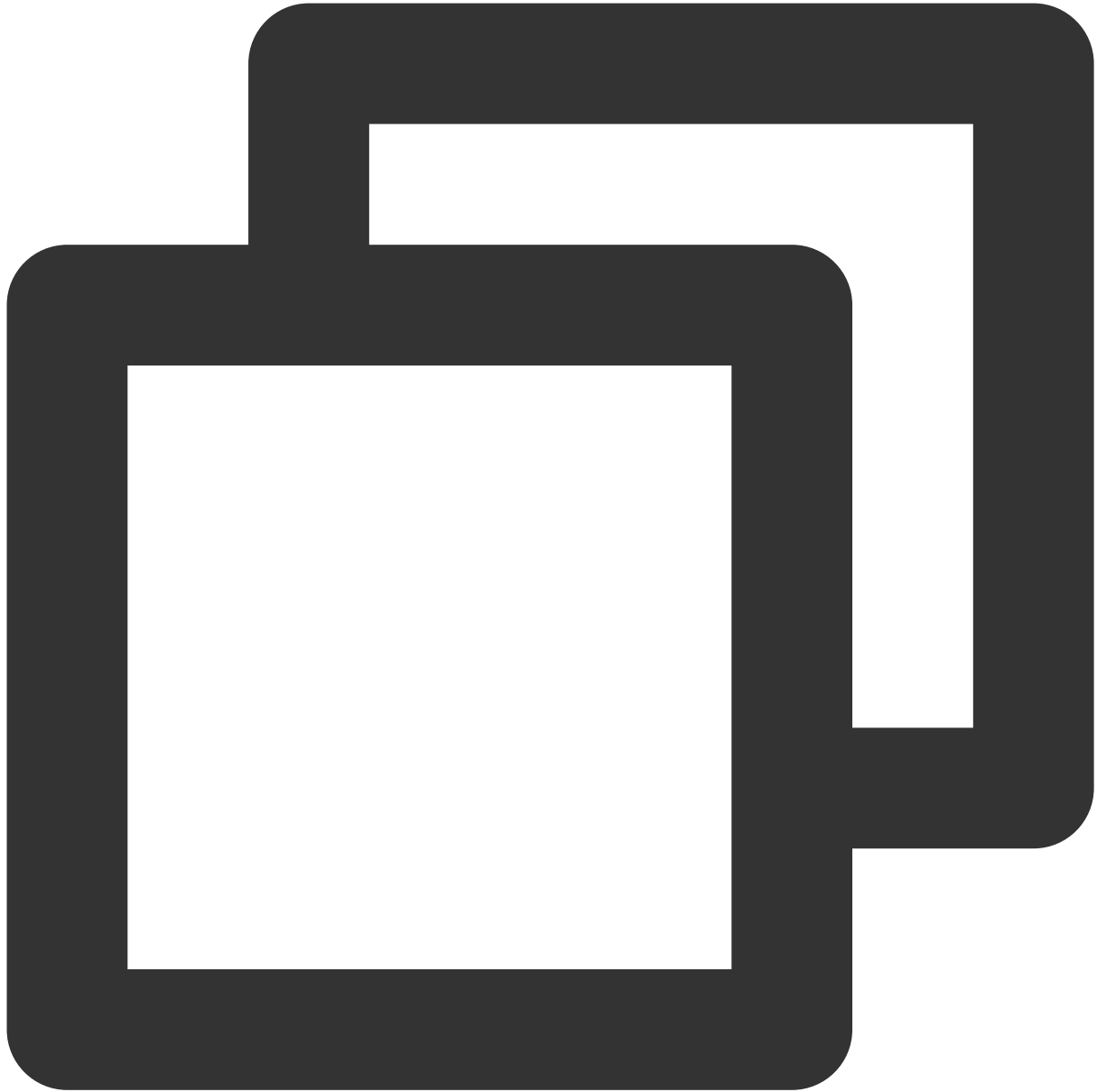
```
npm run dev
```

2. Experience the TUIRoom Component functions

**Note :**

If there are eslint errors in src/TUIRoom directory during runtime, you can add /src/TUIRoom/ path to block eslint checking in .eslintignore file.

## 1. Execute the development environment command



```
npm run start
```

### Note:

Electron's default listening port number is 8080. If the UI is not rendered after packaging, it may be because the default port is occupied. Modify the corresponding port number in main.electron.js.

## 2. Experience the TUIRoom Component functions

### Note :



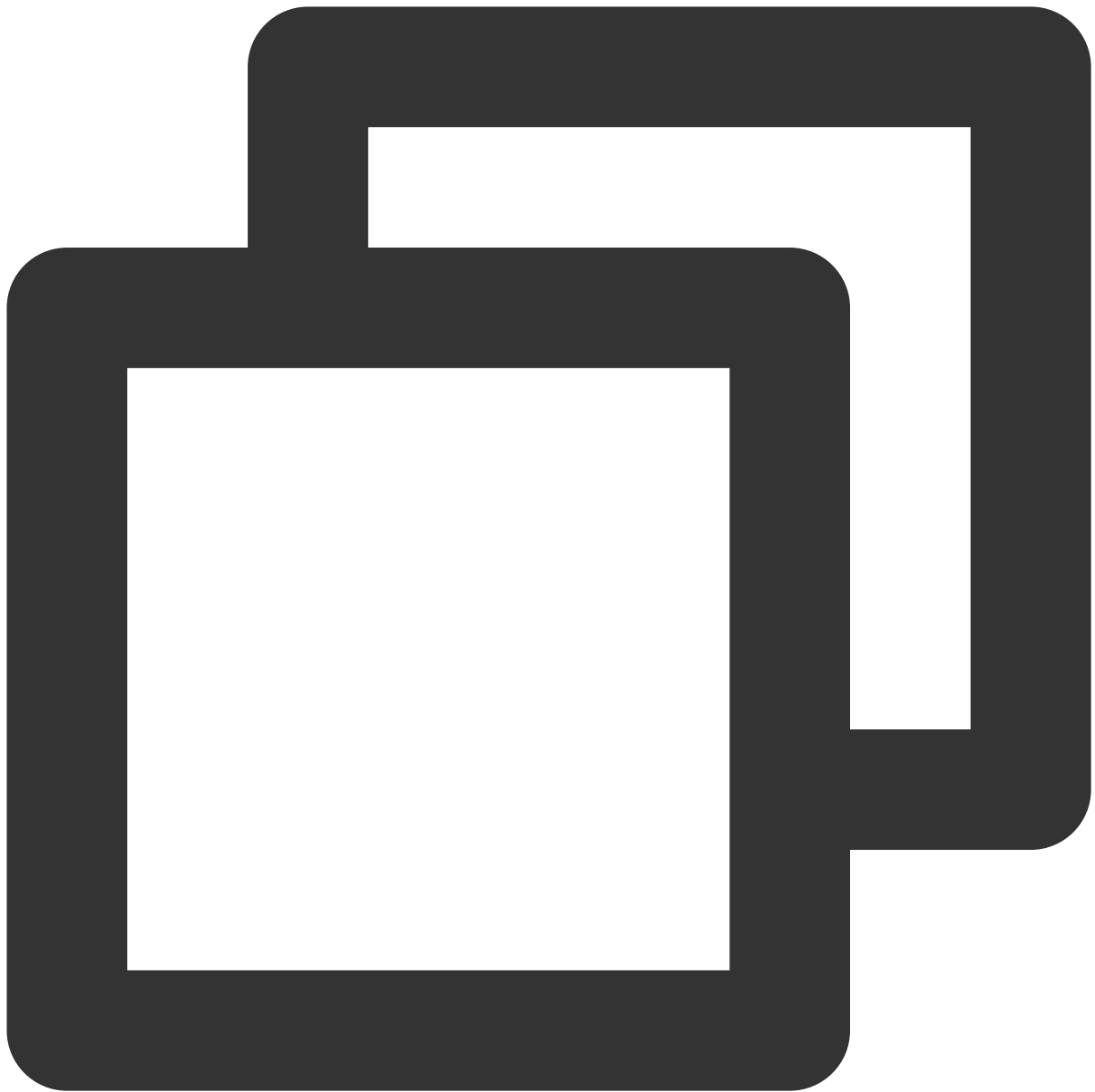
If there are eslint errors in `src/TUIRoom` directory during runtime, you can add `/src/TUIRoom/` path to block eslint checking in `.eslintignore` file.

## Step 6: Build the installation package and run

Vue3 + Vite + TS Project

Vue2 + Webpack + TS Project

In the command line terminal, execute the following command to build the installation package. The built installation package is located in the `release` directory and can be installed and run.



```
npm run build
```

In the command line terminal, execute the corresponding build command in package.json. The built installation package is located in the bin directory and can be installed and run.

**Note:**

Only Mac computers can build Mac installation packages, and Windows computers can build Windows installation packages. Cross-compilation is temporarily not supported.

## Common Problems

### 1. Running Times Error : "does not provide an export named 'createBlock' " ?

If you get the following error when running after following the above steps to access, please make sure you fix the Vite version below 3.0.0 and the Vue version below 3.4.9.

```
/node_modules/.vite/deps/vue.js?v=b083d236' does not provide an export named 'create
```

The reason is that the version of vite supported by the electron project template provided by the project is specified to be under 3.0.0, but the attributes in vue version 3.4.9 and above need to be upgraded in order to get the support, and an [issue](#) has been raised with the vue official here.。

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# Windows

Last updated : 2024-04-12 11:29:00

This article will introduce how to complete the integration of the `TUIRoomKit` Component in the shortest time. By following this document, you will complete the following key steps within an hour and finally obtain a Real-time Conference SDK with a complete UI interface. This includes Member Management, Screen Sharing, chat, Free Speech, Request to Speak mode, and other functions.

## Step 1: Activate the service

## Step 2: Download TUIRoomKit Project Code

### 1. Download TUIRoomKit Component Code

Click [Github](#), clone or download TUIRoomKit repository code, Windows-Mac directory is the directory for Windows project.

### 2. Modify User Configuration

Enter the Windows-Mac directory, open the `utils\usersig\win\GenerateTestUserSig.h` file, and modify the SDKAppID and Secret Key.

## Step 3: Configure Development Environment

### Windows Environment:

Visual Studio 2015 or higher integrated development environment.

QT5.9.1 or higher version of Qt development library.

QT development plugin Qt Visual Studio Tools 2.2.0 or higher for VS.

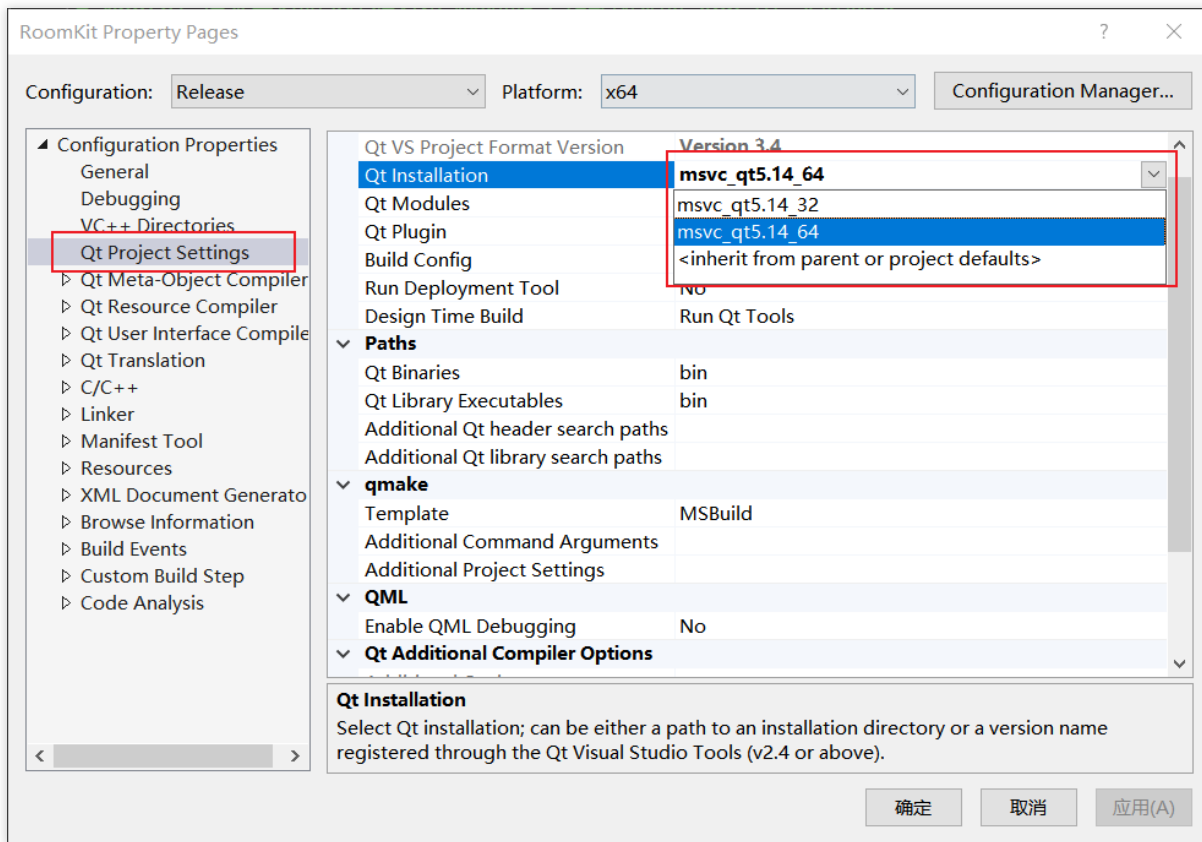
Minimum supported system: Windows 8.1.

Please ensure that your integrated development environment can develop normally.

## Step 4: Configure TUIRoomKit Project

1. Enter the RoomKit directory, use Visual Studio to open the `RoomKit.vcxproj` file, and open the TUIRoomKit project.

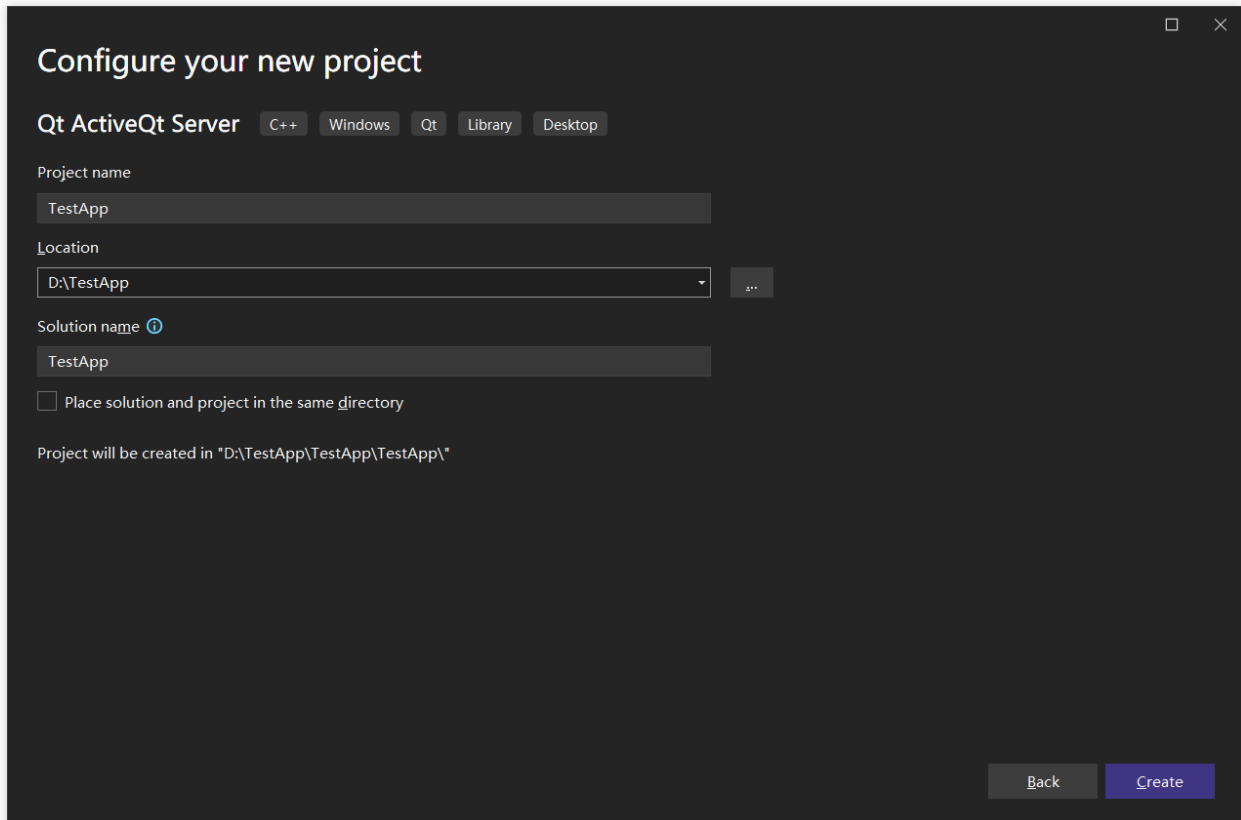
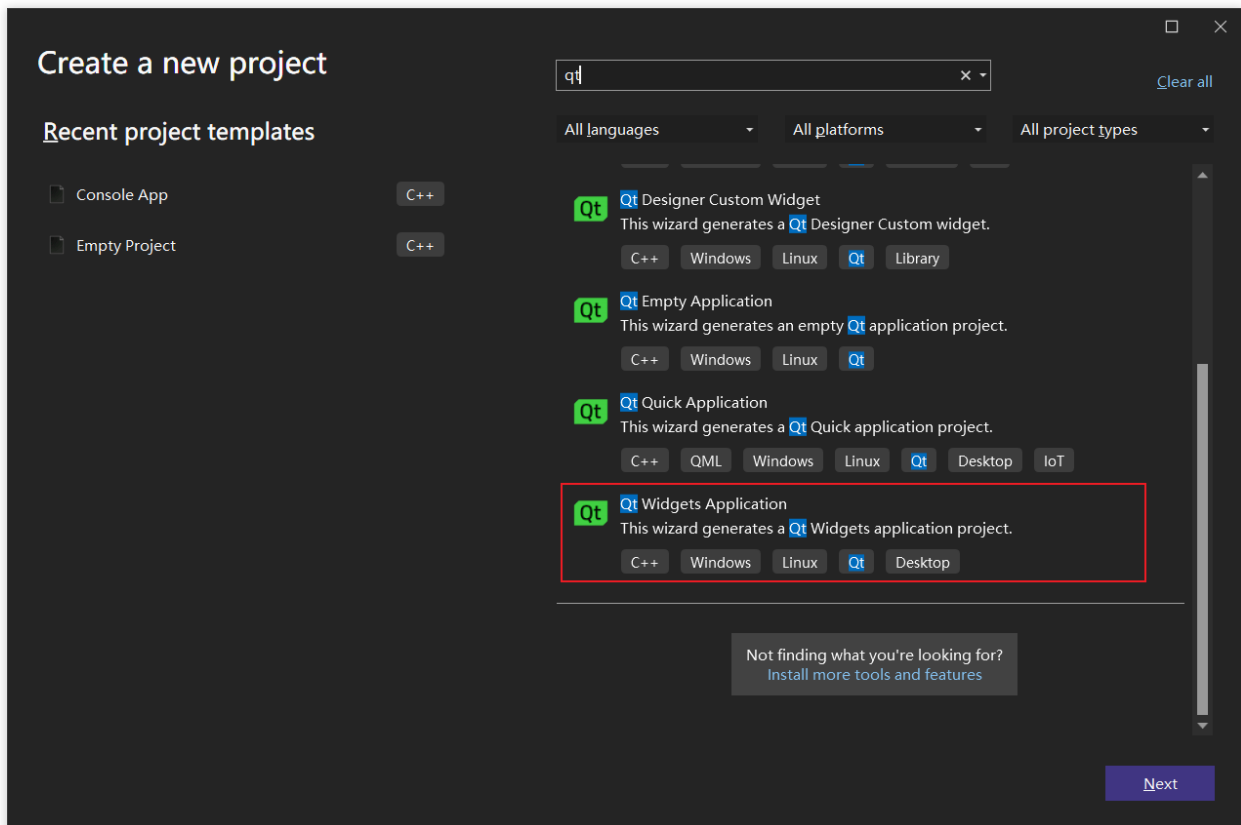
## 2. Configure QT development settings:



## 3. Compile the TUIRoomKit project and generate the RoomKit.exe program.

# Step 5: Externally Launch TUIRoomKit Program

## 1. Open VS, select Qt Widgets Application project type, and create TestApp project.



2. Write the program to start the process and invoke the LoadRoomApp function at the appropriate location.



```
#include <QProcess>
#include <QApplication>
void LoadRoomApp() {
    QString executable_file_path = QApplication::applicationDirPath();
    QString app_path = executable_file_path + "/RoomKit.exe";
    QProcess::startDetached(app_path);
}
```

```
#include "TestApp.h"

#include <QProcess>           Launch RoomApp
#include <QApplication>

void LoadRoomApp() {
    QString executable_file_path = QApplication::applicationDirPath();
    QString app_path = executable_file_path + "/RoomApp.exe";
    QProcess::startDetached(app_path);
}

TestApp::TestApp(QWidget *parent)
    : QMainWindow(parent) {
    ui.setupUi(this);
    LoadRoomApp();           When TestApp launches, launch RoomApp
}
```

3. Compile the project and copy the output of RoomKit compilation to the current executable program directory, taking the release x86 program as an example:

Copy all files in the `TUIRoomKit\\Windows-Mac\\RoomKit\\bin\\Win32\\Release` directory to the current program directory.

4. Execute the program, and simultaneously launch RoomKit with TestApp.

## Suggestions and Feedback

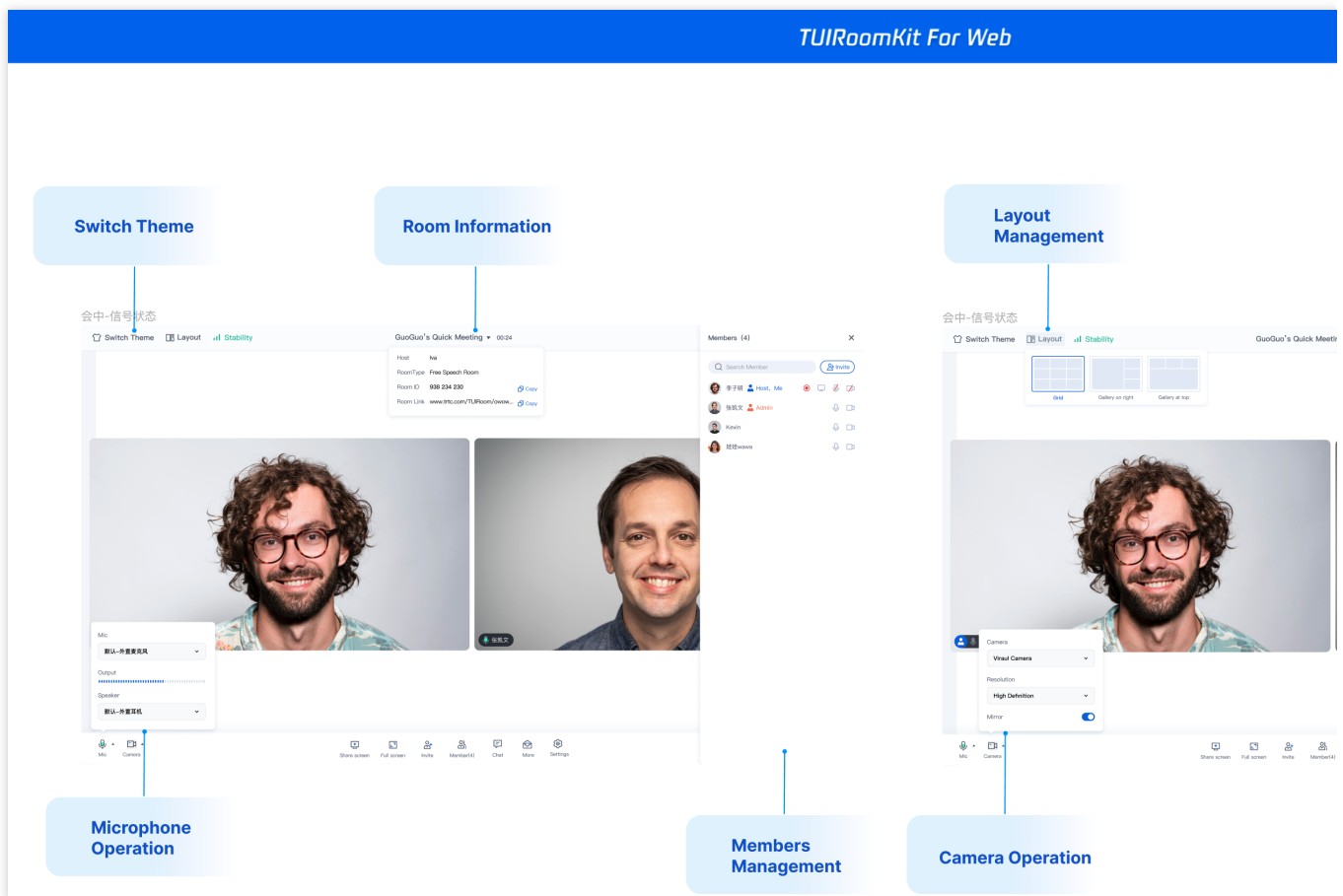
If you have any suggestions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).



# Web

Last updated : 2024-04-12 11:29:00

TUIRoomKit is a multi-person audio and video UI component launched by Tencent Cloud. The component provides rich functional interactions such as room management, audio and video control, screen sharing, member management, microphone management, instant messaging, custom layout switching, etc., and supports Chinese and English switching, one-click skin change, and other capabilities.



This article introduces the access guide of TUIRoomKit(Web), helping you quickly launch business scenarios such as enterprise meetings, online education, medical consultations, online inspections, and remote loss assessments.

You can click on the [TUIRoomKit online experience link](#) to experience more features of TUIRoomKit.

You can click on [Github](#) to download the TUIRoomKit code and refer to the README.md document in the code repository to run the TUIRoomKit Web sample project.

## Step 1: Activate the service

Before initiating a meeting with TUIRoomKit, you need to activate the exclusive multi-person audio and video interaction service for TUIRoomKit on the console. For specific steps, please refer to [Activate Service](#).

## Step 2: Prepare Vue project code

If you already have a Vue project, you can skip this step.

If you don't have a Vue project, you can choose one of the following ways to generate a template project.

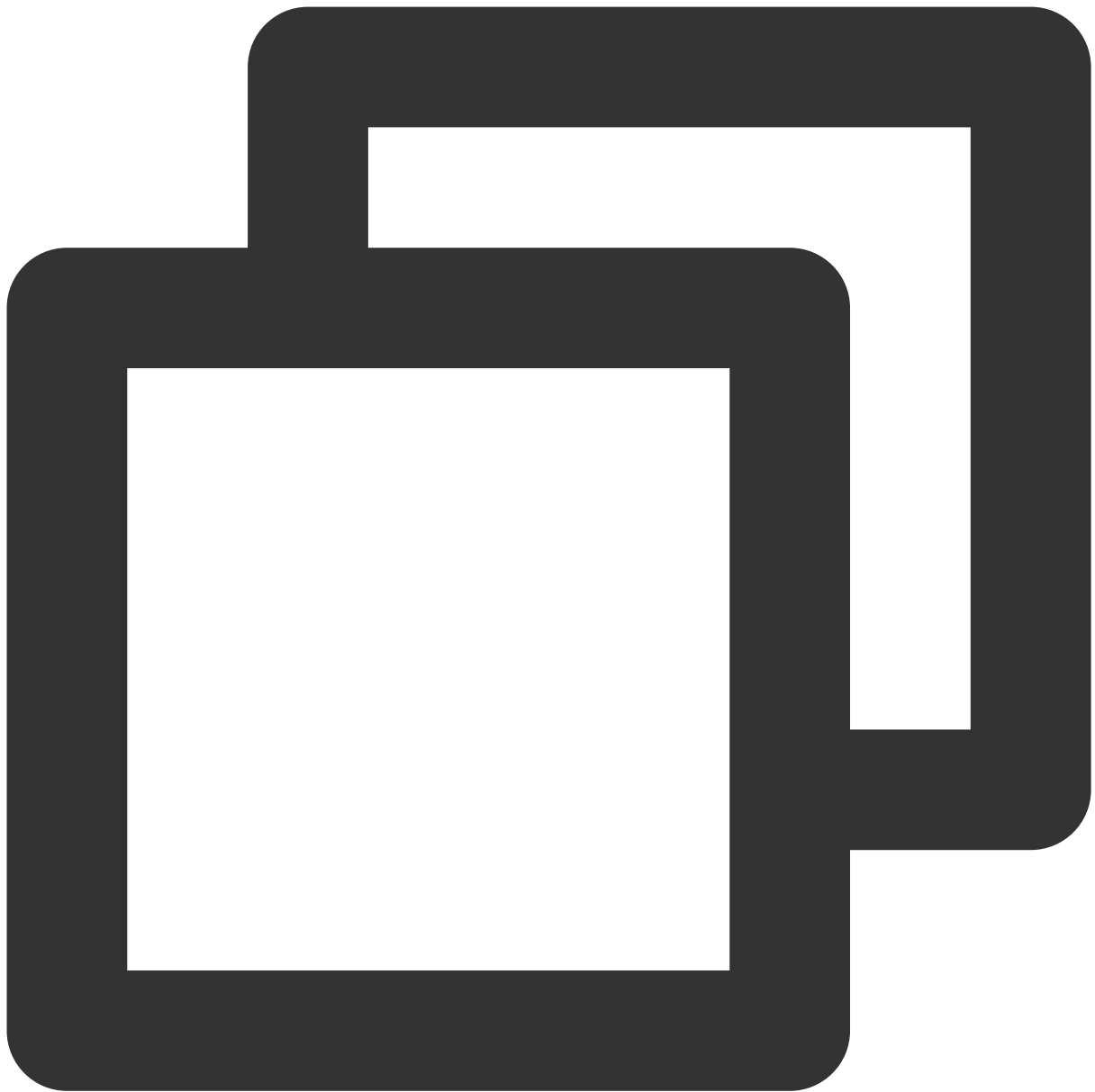
Vue3 + Vite + TS Template Project

Vue3 + Webpack + TS Template Project

Vue2 + Webpack + JS Template Project

### Note :

Vite requires Node.js version 14.18+, 16+. However, some templates depend on a higher Node version to work properly, so be careful to upgrade your Node version when your package manager warns you.



```
# npm 6.x
npm create vite@latest TUIRoom-demo --template vue-ts

# npm 7+, extra double-dash is needed:
npm create vite@latest TUIRoom-demo -- --template vue-ts

# yarn
yarn create vite TUIRoom-demo --template vue-ts

# pnpm
pnpm create vite TUIRoom-demo --template vue-ts
```

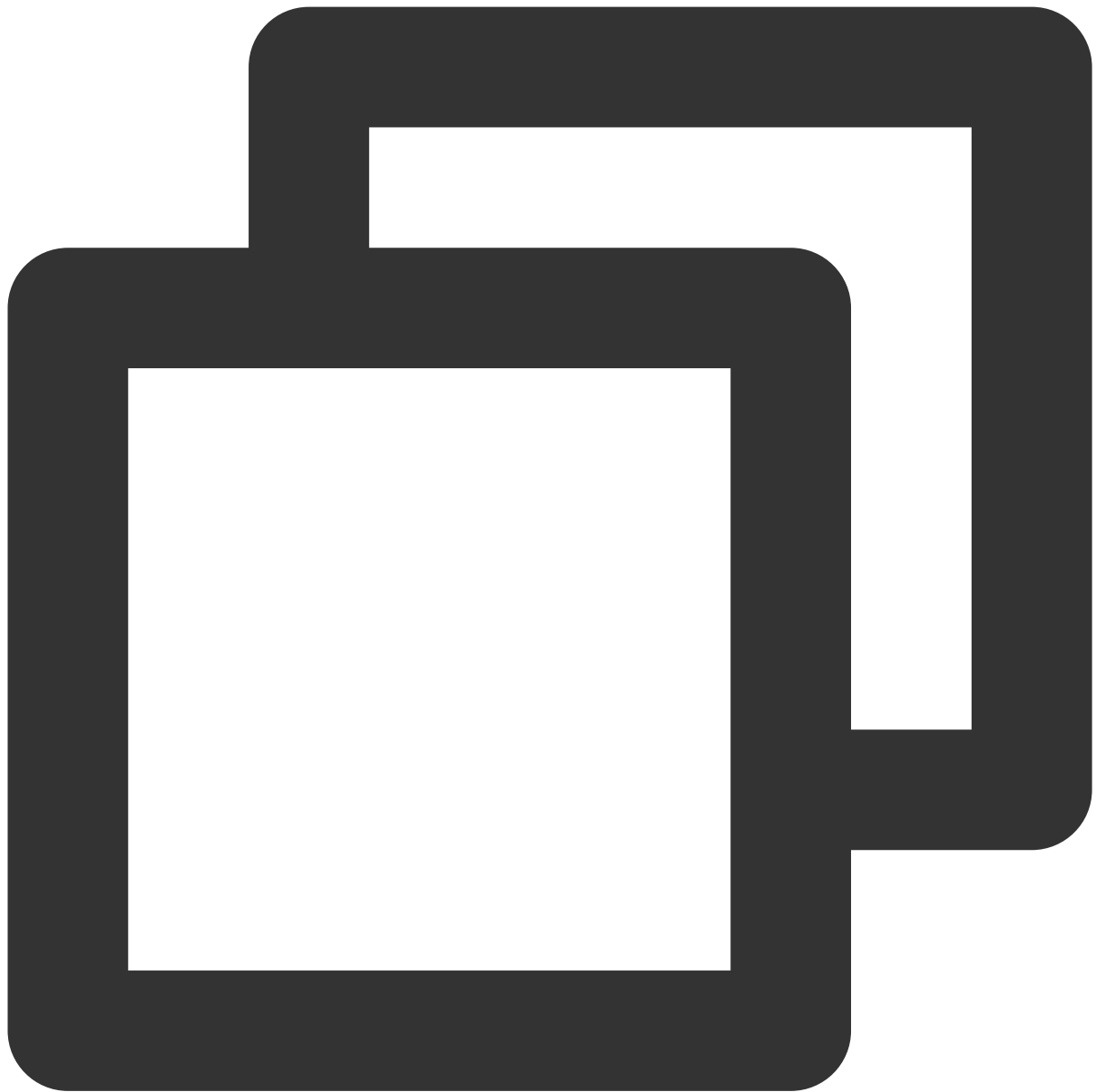
**Note:**

Taking the creation method of npm6.x as an example, the operation flowchart of generating the template project is as follows: (The marked highlight area needs to be input or selected)

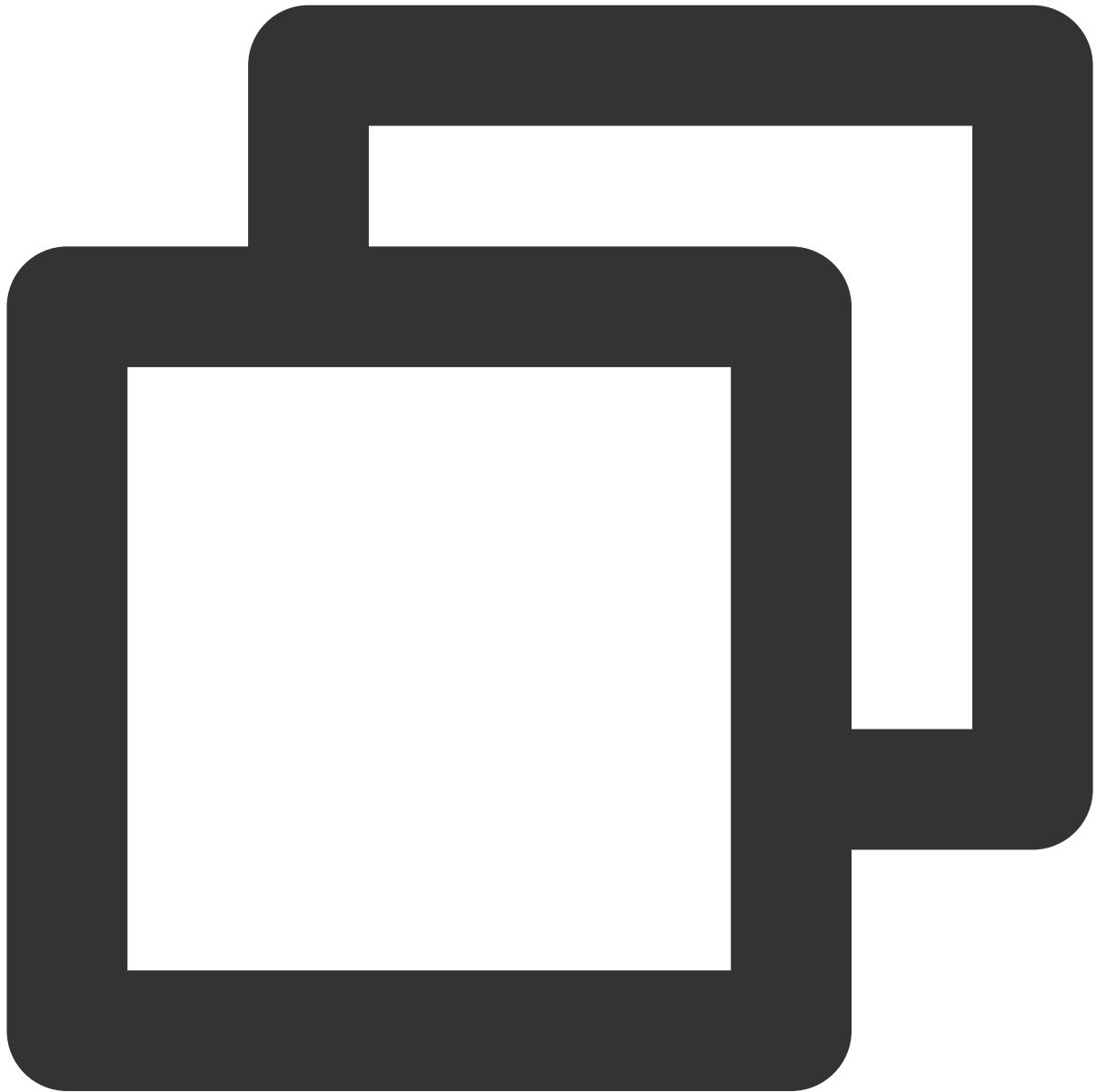
```
x npm create vite@latest TUIRoom-demo --template vue-ts
✓ Package name: ... tuiroom-demo Step2. Project Naming Step1. Create Empty Project
? Select a framework: > - Use arrow-keys. Return to submit.
  Vanilla
> Vue Step3. Choose Vue
  React
  Preact
  Lit
  Svelte
  Others

? Select a variant: > - Use arrow-keys. Return to submit.
  JavaScript
> TypeScript Step4. Choose TypeScript
  Customize with create-vue ↗
  Nuxt ↗
```

After successfully generating the Vue3 + Vite + TS template project, execute the following script:



```
cd TUIRoom-demo  
npm install  
npm run dev
```



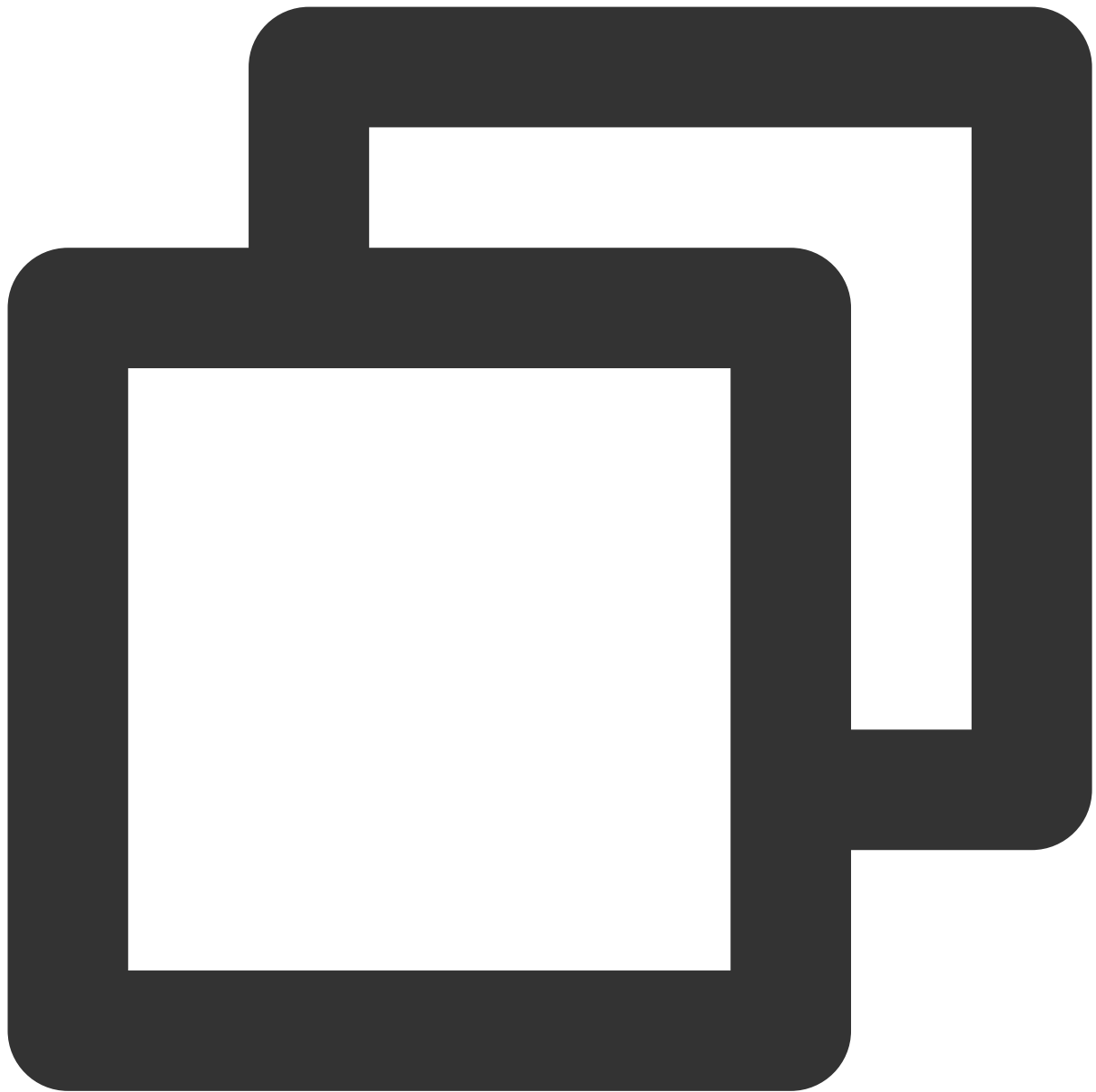
```
// Install vue-cli, note that Vue CLI 4.x requires Node.js to be v10 or above
npm install -g @vue/cli
// Create Vue3 + Webpack + TS Template Project
vue create tuiroomkit-demo
```

**Note:**

During the process of executing the generate template project script, choose the template generation method as Manually select features, and refer to the reference image for other configuration options.

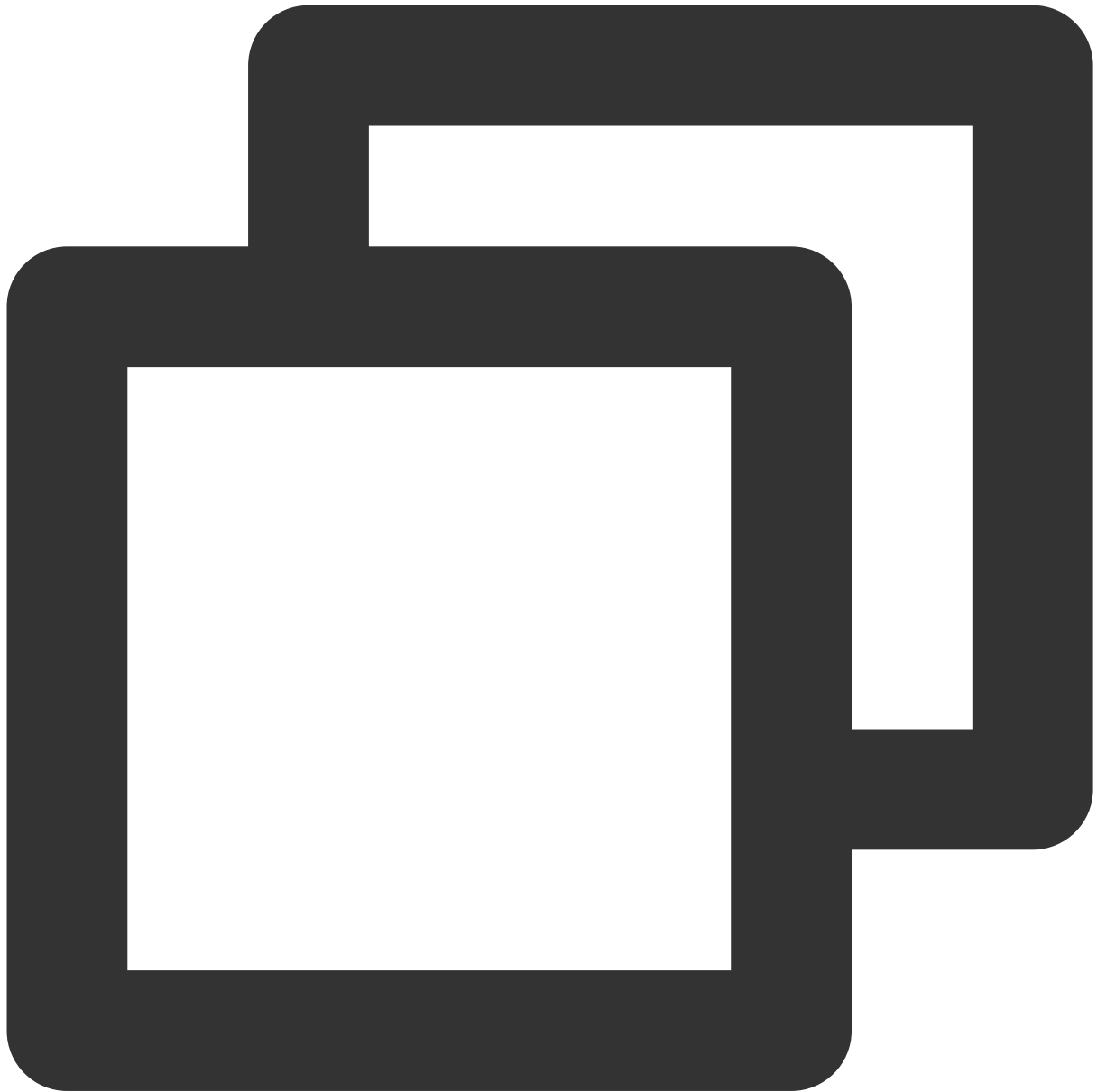
```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, TS, Linter
? Choose a version of Vue.js that you want to start the project with 3.x
? Use class-style component syntax? No
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)? Yes
? Pick a linter / formatter config: Standard
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? No
```

After successfully generating the Vue3 + Webpack + TS template project, execute the following script:



```
cd tuiroomkit-demo  
npm run serve
```





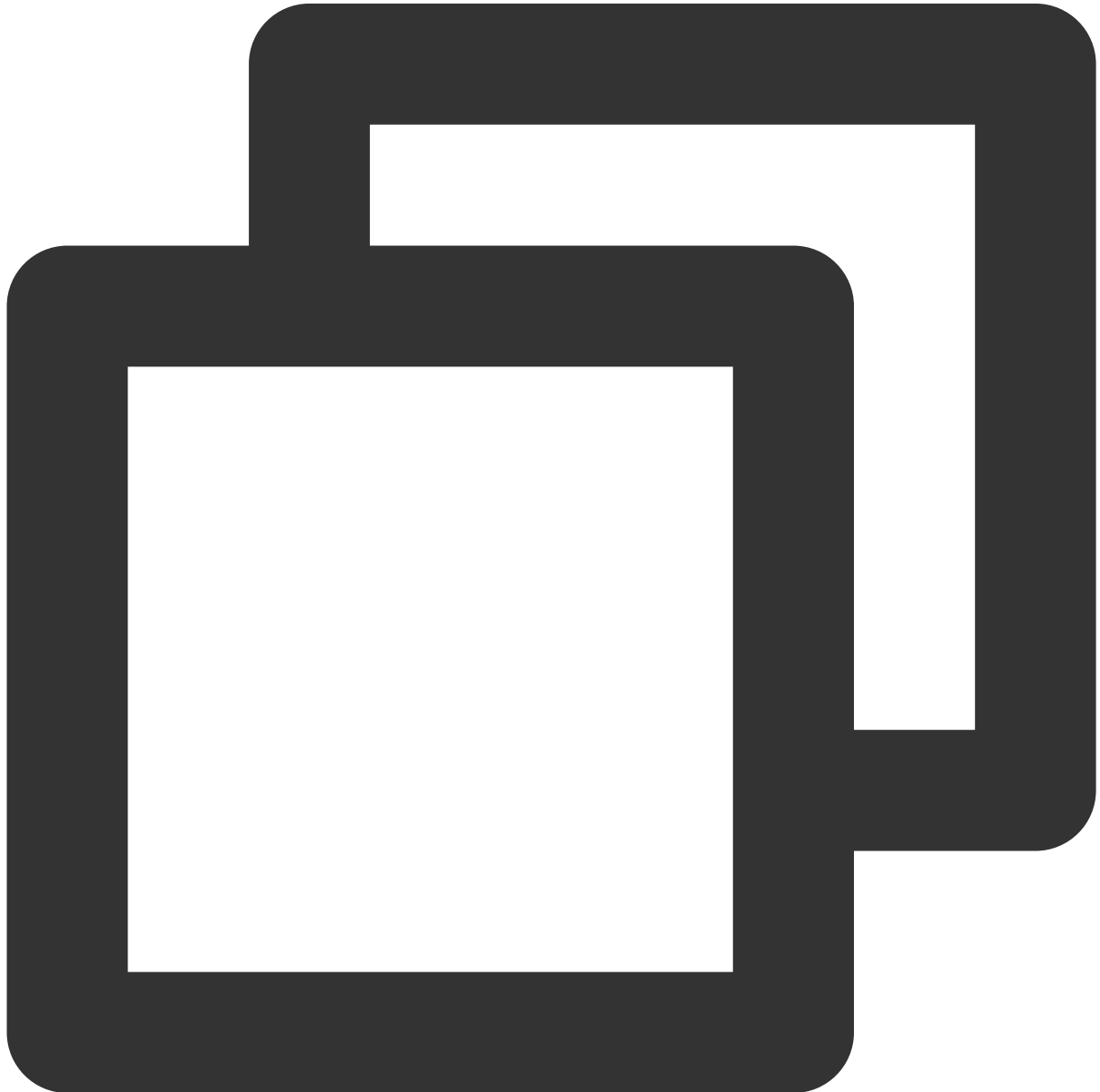
```
// Install vue-cli, note that Vue CLI 4.x requires Node.js to be v10 or above
npm install -g @vue/cli
// Create Vue2 + Webpack + Js Template Project
vue create tuiroomkit-demo
```

**Note:**

During the process of executing the generate template project script, choose the template generation method as Default ([Vue 2] babel, eslint).

```
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

After successfully generating the Vue2 + Webpack + JS template project, execute the following script:



```
cd tuiroomkit-demo
npm run serve
```

## Step 3: Download and reference TUIRoom component

### 1. Download TUIRoom component code

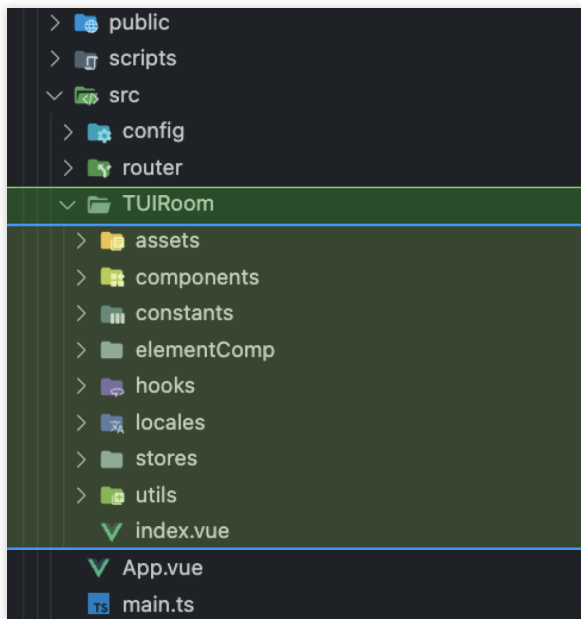
Click on [Github](#), clone or download the TUIRoomKit repository code.

### 2. Reference TUIRoom component

Introducing TUIRoom Component in Vue3 Project

Introducing TUIRoom Component in Vue2 Project

Copy the `TUIRoomKit/Web/vue3/src/TUIRoom` folder to the existing project `src/` directory,



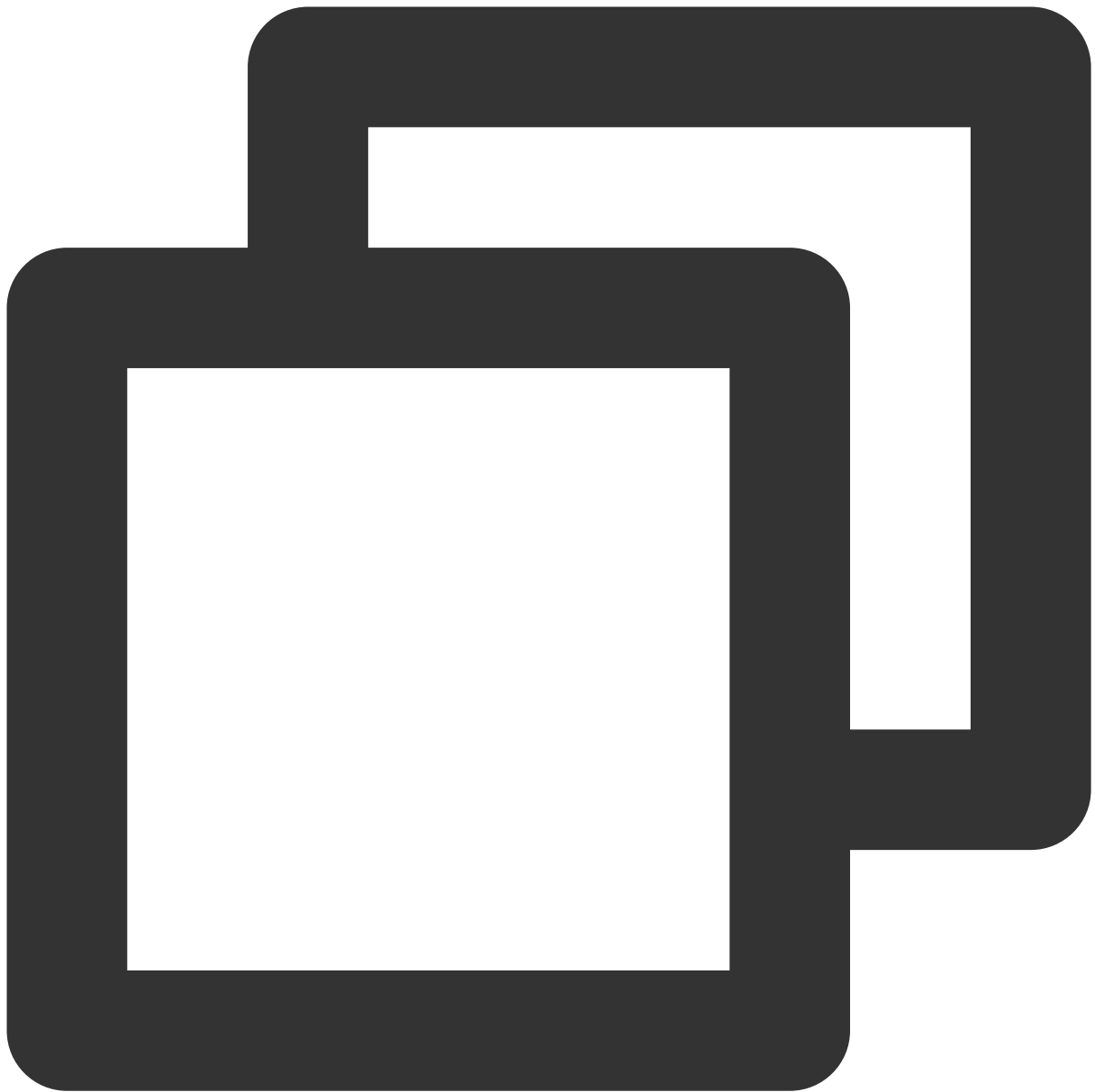
Reference the TUIRoom Component in the page. For example, import the TUIRoom Component in the `App.vue` component.

The TUIRoom Component divides users into Host and Regular Member roles. The Component provides `init`, `createRoom`, and `enterRoom` methods externally.

Both the Host and Regular Members can initialize the application and user data to the TUIRoom Component through the `init` method, the Host can create and join a Room through the `createRoom` method, and Regular Members can join a Room created by the Host through the `enterRoom` method.

#### Note:

After copying the following code in the page, you need to modify the parameters of the TUIRoom interface to the actual data.



```
<template>
  <room ref="TUIRoomRef"></room>
</template>

<script setup lang="ts">
import { ref, onMounted } from 'vue';
// Import TUIRoom Component, make sure the import path is correct
import Room from './TUIRoom/index.vue';
// Get the TUIRoom Component element for calling TUIRoom Component methods
const TUIRoomRef = ref();
```

```
onMounted(async () => {
  // Initialize TUIRoom Component
  // The host needs to initialize the TUIRoom Component before creating a room
  // Ordinary members need to initialize the TUIRoom Component before entering a room
  await TUIRoomRef.value.init({
    // Get sdkAppId, please refer to Step 1
    sdkAppId: 0,
    // The unique identifier of the user in your business
    userId: '',
    // Local development and debugging can quickly generate userSig on https://console
    userSig: '',
    // The nickname used by the user in your business
    userName: '',
    // The avatar URL used by the user in your business
    avatarUrl: '',
    // Skin theme colours that users need in your business and whether or not switch
    theme: {
      defaultTheme: 'black',
      isSupportSwitchTheme: true
    }
  })
  // By default, create a room. In actual access, you can choose to execute the handleEnterRoom method
  await handleCreateRoom();
})

// The host creates a room, this method is only called when creating a room
async function handleCreateRoom() {
  // roomId is the room number entered by the user, and roomId is required to be of type string
  // roomMode includes 'FreeToSpeak' (Free-to-speak mode) and 'SpeakAfterTakingSeat' (Speak after taking seat mode)
  // roomParam specifies the default behavior of the user when entering the room (roomParam is optional)
  const roomId = '123456';
  const roomMode = 'FreeToSpeak';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
  }
  try {
    await TUIRoomRef.value.createRoom({ roomId, roomName: roomId, roomMode, roomParam })
  } catch (error: any) {
    alert('TUIRoomKit.createRoom error: ' + error.message);
  }
}

// Regular members enter the room, this method is called when regular members enter the room
async function handleEnterRoom() {
  // roomId is the room number entered by the user, and roomId is required to be of type string
  // roomParam specifies the default behavior of the user when entering the room (roomParam is optional)
```

```
const roomId = '123456';
const roomParam = {
  isOpenCamera: true,
  isOpenMicrophone: true,
}
try {
  await TUIRoomRef.value.enterRoom({ roomId, roomParam });
} catch (error: any) {
  alert('TUIRoomKit.enterRoom error: ' + error.message);
}
</script>

<style lang="scss">
html, body {
  width: 100%;
  height: 100%;
  margin: 0;
}

#app {
  width: 100%;
  height: 100%;
}
</style>
```

Parameter descriptions:

Here is a detailed description of the key parameters used in init:

**sdkAppId:** you have already got it in the last step of step 1, so I won't repeat it here.

**userId:** ID of the current user, string type, only allow alphabets (a-z and A-Z), numbers (0-9), hyphens (-) and underscores (\_).

**userSig:** Use SDKSecretKey to encrypt information such as sdkAppId, userId, etc. to get userSig, which is an authentication ticket used by Tencent Cloud to identify whether the current user is able to use TRTC's services. You can get a temporary userSig by visiting [Real-time Audio/Video TRTC Console -> UserSig Tools](#).

The screenshot shows the 'UserSig Tools' interface. At the top, there is a warning banner: 'You haven't provided a payment method. We will suspend the service for your account after you use up your free resources. To avoid service interruption, please [complete your information](#) and [refresh](#).' Below this, the interface is divided into two main sections: 'Signature (UserSig) Generator' and 'Signature (UserSig) Verifier'. The 'Generator' section includes fields for 'Application (SDKAppID)' (a dropdown menu), 'Username (UserID)' (a text input), and 'Secret key' (a text area with a placeholder 'Auto-generated after you select an application'). There is a 'Generate' button and a 'Generate result' section with a 'Copy' button. The 'Verifier' section includes fields for 'Application (SDKAppID)' (a dropdown menu), 'Username (UserID)' (a text input), and 'Secret key' (a text area with a placeholder 'Auto-generated after you select an application'). There is a 'Verify' button and a 'UserSig' text input field with a placeholder 'Please enter'.

**Note :**

This step is also by far the one we receive the most feedback on from developers, with the following common problems.

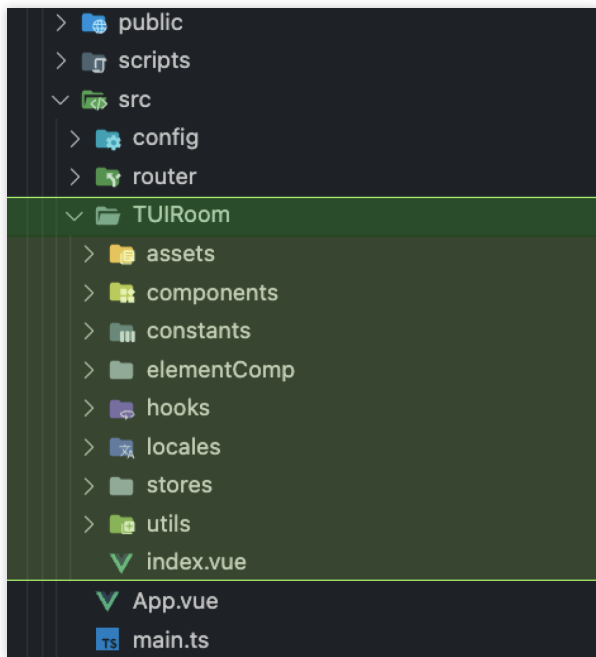
sdkAppId is set incorrectly.

userSig is misconfigured as a SDKSecretkey. userSig is obtained by encrypting sdkAppId, userId and expiration time with the SDKSecretkey, instead of configuring the SDKSecretkey as userSig directly.

UserSig is set to a simple string such as "1", "123", "111", etc. Since TRTC doesn't support multi-login with the same UserID, it is not possible to use the same UserID for multi-login, so it is not possible to use the same UserID for multi-login. Since TRTC does not support multi-location login with the same userID, userIDs like "1", "123", "111" will easily be occupied by your colleagues during multi-user development, resulting in login failure. recommend that you set some highly recognizable userID during debugging.

The sample code in Github uses the `genTestUserSig` function to compute the userSig locally in order to get you through the current access process faster, but this solution exposes your SDKSecretkey to the App's code, which is not conducive to upgrading and securing your SDKSecretkey, so we strongly recommend that you place the Therefore, we strongly recommend that you put the UserSig calculation logic on the server side and have the App request the real-time calculated userSig from your server every time you use the TUIRoomKit component.

Copy the `TUIRoomKit/Web/vue2/src/TUIRoom` folder to the existing project `src/` directory,



Reference the TUIRoom Component in the page. For example, import the TUIRoom Component in the `App.vue` component.

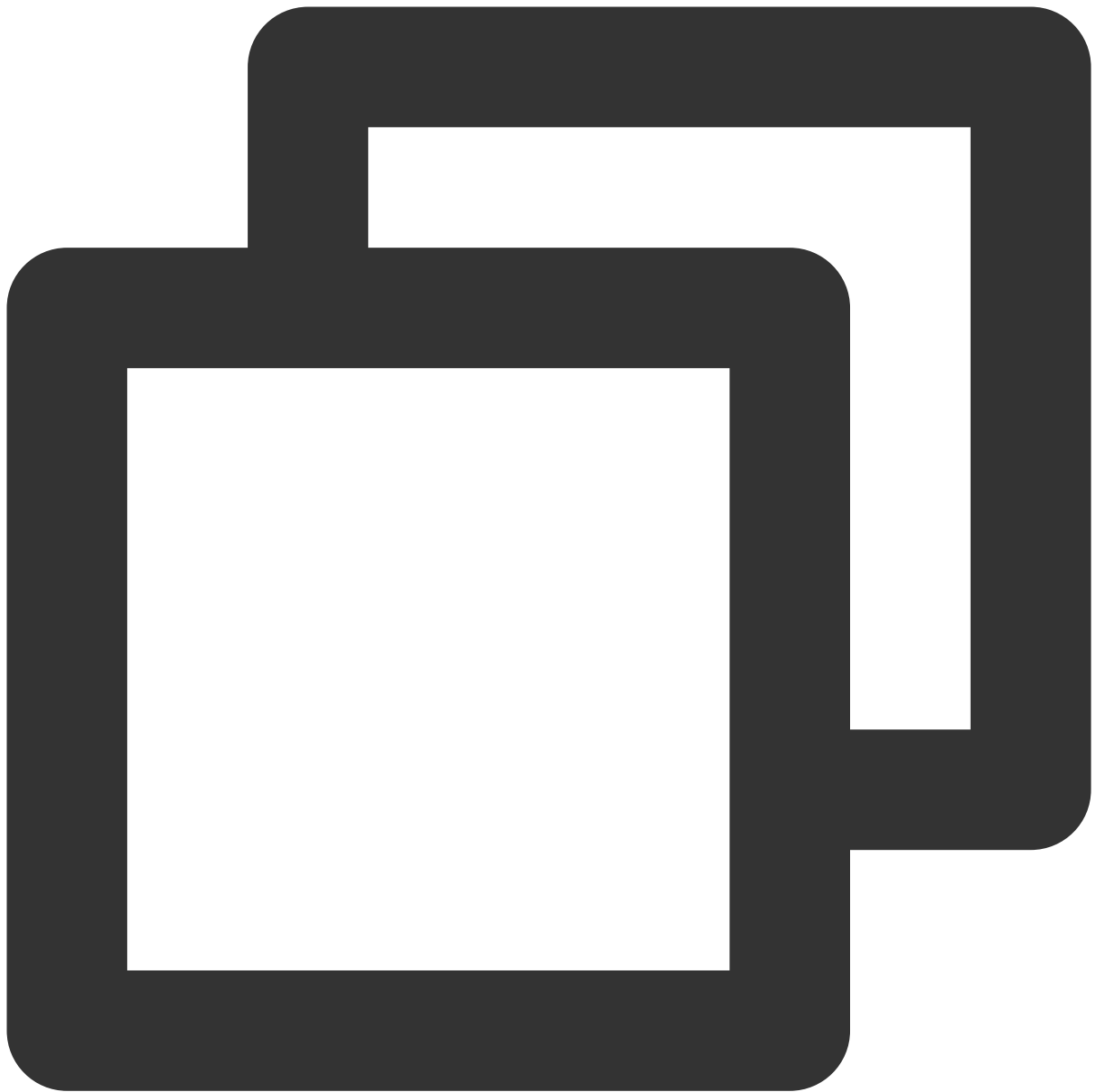
The TUIRoom Component divides users into Host and Regular Member roles. The Component provides `init`, `createRoom`, and `enterRoom` methods externally.

Both the Host and Regular Members can initialize the application and user data to the TUIRoom Component through the `init` method, the Host can create and join a Room through the `createRoom` method, and Regular Members can join a Room created by the Host through the `enterRoom` method.

**Note:**

After copying the following code in the page, you need to modify the parameters of the TUIRoom interface to the actual data.





```
<template>
  <div id="app">
    <room-container ref="TUIRoomRef"></room-container>
  </div>
</template>

<script>
import RoomContainer from '@/TUIRoom/index.vue';
export default {
  name: 'App',
  components: { RoomContainer },
```

```
data() {
  return {};
},
async mounted() {
  // Initialize TUIRoom Component
  // The host needs to initialize the TUIRoom component before creating a room
  // Regular members need to initialize the TUIRoom component before entering a room
  await this.$refs.TUIRoomRef.init({
    // Get sdkAppId, please refer to Step 1
    sdkAppId: 0,
    // User unique identifier in your business
    userId: '',
    // Local development and debugging can quickly generate userSig on https://console.cloud.tencent.com/rtc
    userSig: '',
    // The nickname used by the user in your business
    userName: '',
    // The avatar link used by the user in your business
    avatarUrl: '',
  });
  // By default, create a room, and actually access it according to the needs of the user
  await this.handleCreateRoom();
},
methods: {
  // The host creates a room, this method is only called when creating a room
  async handleCreateRoom() {
    // roomId is the room number entered by the user, and roomId is required to be a string
    // roomMode includes 'FreeToSpeak' (Free-to-speak mode) and 'SpeakAfterTaking' (Speak after taking mode)
    // roomParam specifies the default behavior of the user when entering the room
    const roomId = '123456';
    const roomMode = 'FreeToSpeak';
    const roomParam = {
      isOpenCamera: true,
      isOpenMicrophone: true,
    }
    try {
      await this.$refs.TUIRoomRef.createRoom({ roomId, roomName: roomId, roomMode, roomParam });
    } catch (error) {
      alert('TUIRoomKit.createRoom error: ' + error.message);
    }
  },
  // Regular members enter the room, this method is called when regular members enter the room
  async handleEnterRoom() {
    // roomId is the room number entered by the user, and roomId is required to be a string
    // roomParam specifies the default behavior of the user when entering the room
    const roomId = '123456';
    const roomParam = {
      isOpenCamera: true,
```

```
        isOpenMicrophone: true,
    }
    try {
        await this.$refs.TUIRoomRef.enterRoom({ roomId, roomParam });
    } catch (error) {
        alert('TUIRoomKit.enterRoom error: ' + error.message);
    }
}
},
};

</script>

<style lang="scss">
html, body {
    width: 100%;
    height: 100%;
    margin: 0;
}

#app {
    width: 100%;
    height: 100%;
    * {
        box-sizing: border-box;
    }
}
</style>
```

#### Parameter descriptions:

Here is a detailed description of the key parameters used in init:

**sdkAppId:** you have already got it in the last step of step 1, so I won't repeat it here.

**userId:** ID of the current user, string type, only allow alphabets (a-z and A-Z), numbers (0-9), hyphens (-) and underscores (\_).

**userSig:** Use SDKSecretKey to encrypt information such as sdkAppId, userId, etc. to get userSig, which is an authentication ticket used by Tencent Cloud to identify whether the current user is able to use TRTC's services. You can get a temporary userSig by visiting [Real-time Audio/Video TRTC Console -> UserSig Tools](#).

The screenshot shows the 'UserSig Tools' interface. At the top, there is a warning banner: 'You haven't provided a payment method. We will suspend the service for your account after you use up your free resources. To avoid service interruption, please [complete your information](#) and [refresh](#).' Below this, the interface is split into two main sections: 'Signature (UserSig) Generator' and 'Signature (UserSig) Verifier'. The Generator section includes fields for 'Application (SDKAppID)' (a dropdown menu), 'Username (UserID)' (a text input), and 'Secret key' (a text input with a placeholder 'Auto-generated after you select an application'). There is a 'Generate' button and a 'Generate result' section with a large text area and a 'Copy' button. The Verifier section includes fields for 'Application (SDKAppID)' (a dropdown menu), 'Secret key' (a text input with a placeholder 'Auto-generated after you select an application'), and 'UserSig' (a text input with a placeholder 'Please enter'). There is a 'Verify' button.

#### Note :

This step is also by far the one we receive the most feedback on from developers, with the following common problems.

sdkAppId is set incorrectly.

userSig is misconfigured as a Secretkey. userSig is obtained by encrypting sdkAppId, userId and expiration time with the SecretKey, instead of configuring the SecretKey as userSig directly.

UserSig is set to a simple string such as "1", "123", "111", etc. Since TRTC doesn't support multi-login with the same UserID, it is not possible to use the same UserID for multi-login, so it is not possible to use the same UserID for multi-login. Since TRTC does not support multi-location login with the same userID, userIDs like "1", "123", "111" will easily be occupied by your colleagues during multi-user development, resulting in login failure. recommend that you set some highly recognizable userID during debugging.

The sample code in Github uses the genTestUserSig function to compute the userSig locally in order to get you through the current access process faster, but this solution exposes your SecretKey to the App's code, which is not conducive to upgrading and securing your SecretKey, so we strongly recommend that you place the Therefore, we strongly recommend that you put the UserSig calculation logic on the server side and have the App request the real-time calculated userSig from your server every time you use the TUIRoomKit component.

## Step 4: Configure the development environment

After the TUIRoom component is introduced, in order to ensure that the project can run normally, the following configurations need to be made:

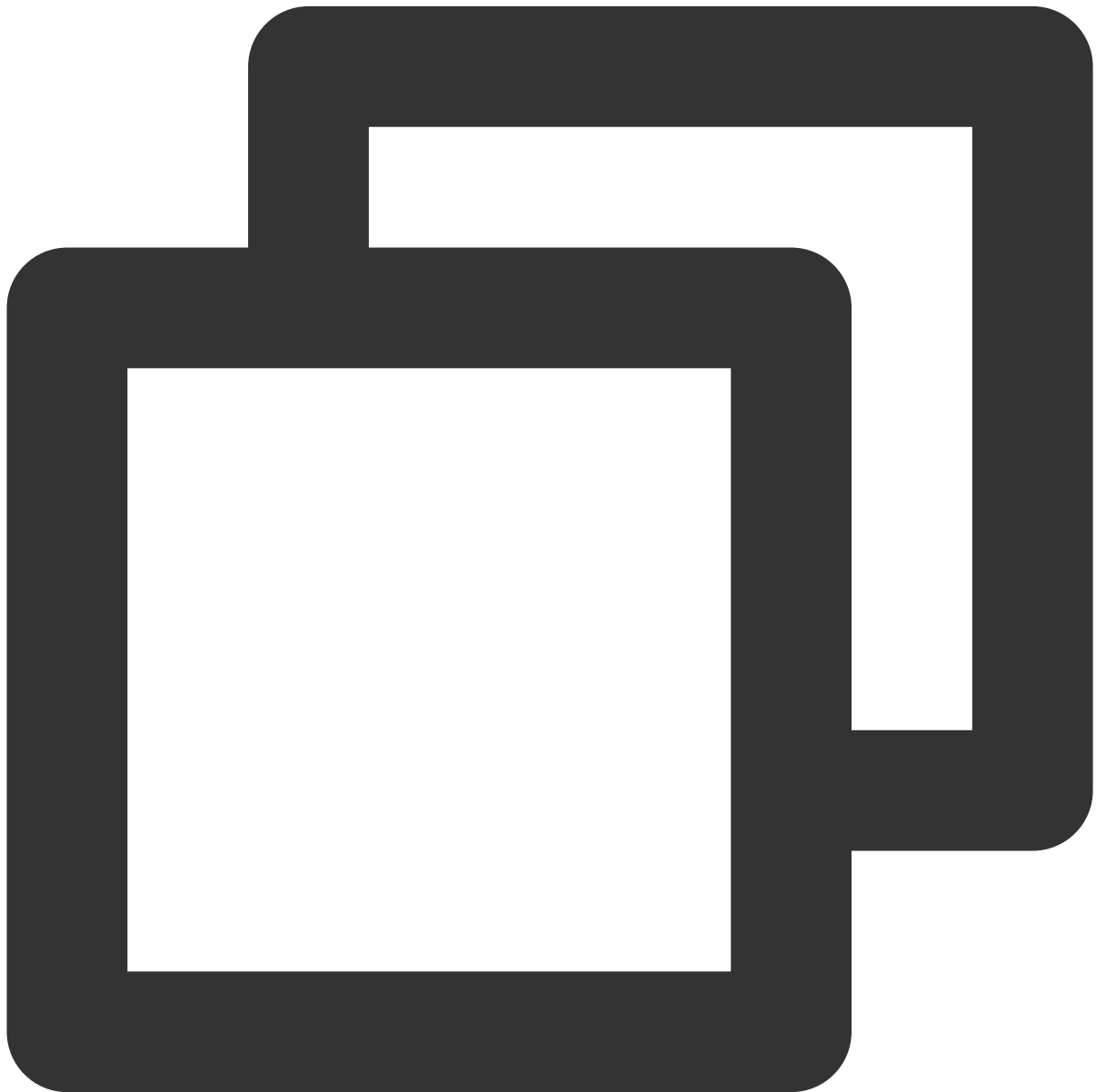
Setting up Vue3 + Vite + TS Environment

Setting up Vue3 + Webpack + TS Environment

Setting up Vue2 + Webpack + TS Environment

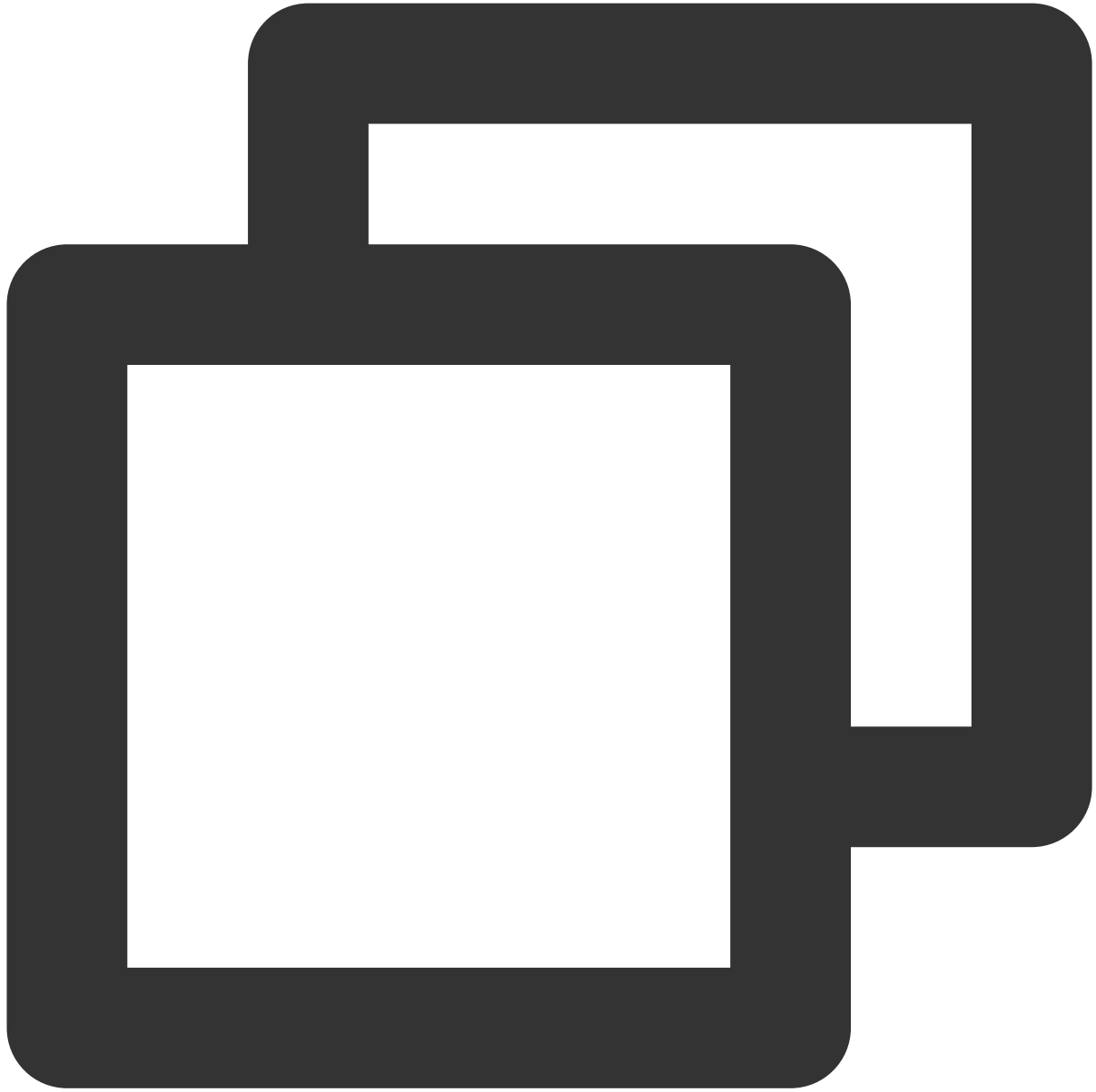
### 1. Install Dependencies

Install Development Environment Dependencies:



```
npm install sass typescript -S -D
```

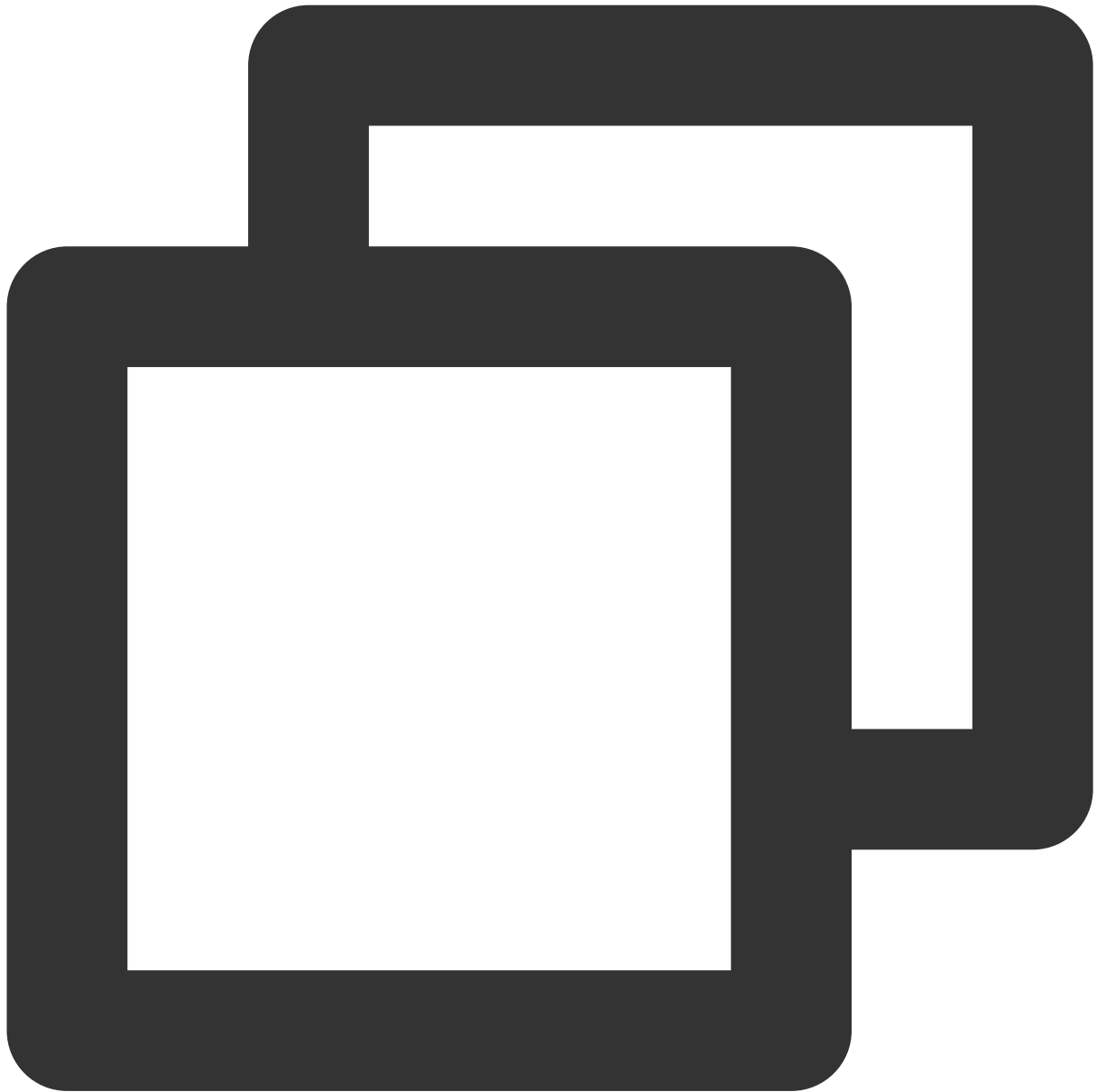
Install Production Environment Dependencies :



```
npm install mitt pinia @tencentcloud/tuiroom-engine-js vue-i18n rtc-detect -S
```

## 2. Register Pinia

TUIRoom uses Pinia for room data management, you need to register Pinia in the project entry file, which is the `src/main.ts` file.

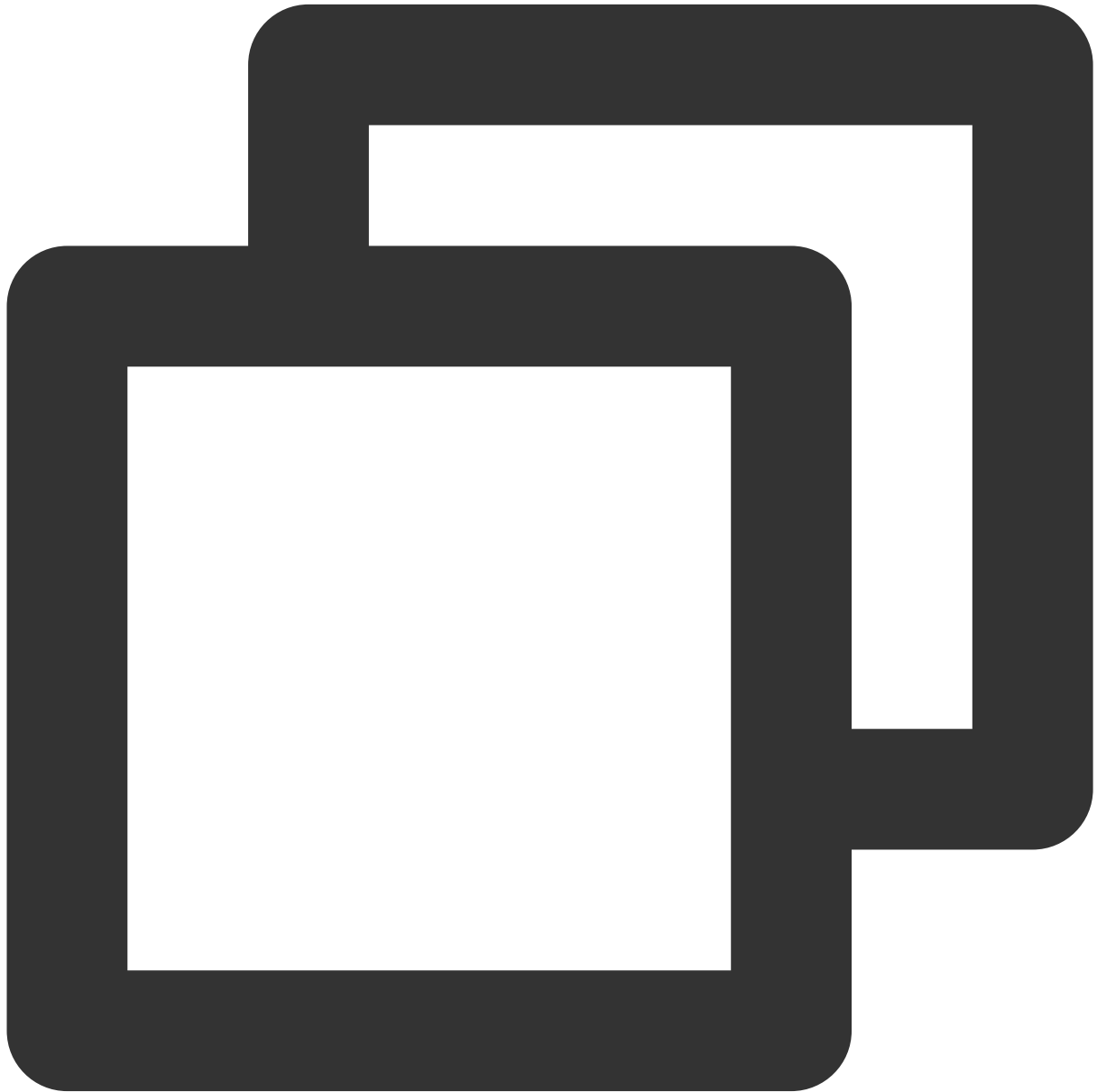


```
// src/main.ts file
import { createPinia } from 'pinia';

const app = createApp(App);
// Register pinia
app.use(createPinia());
app.mount('#app');
```

### 3. Configure Chinese-English language switching

TUIRoom currently supports Chinese-English language switching, you need to register the i18n instance in the main.ts file.



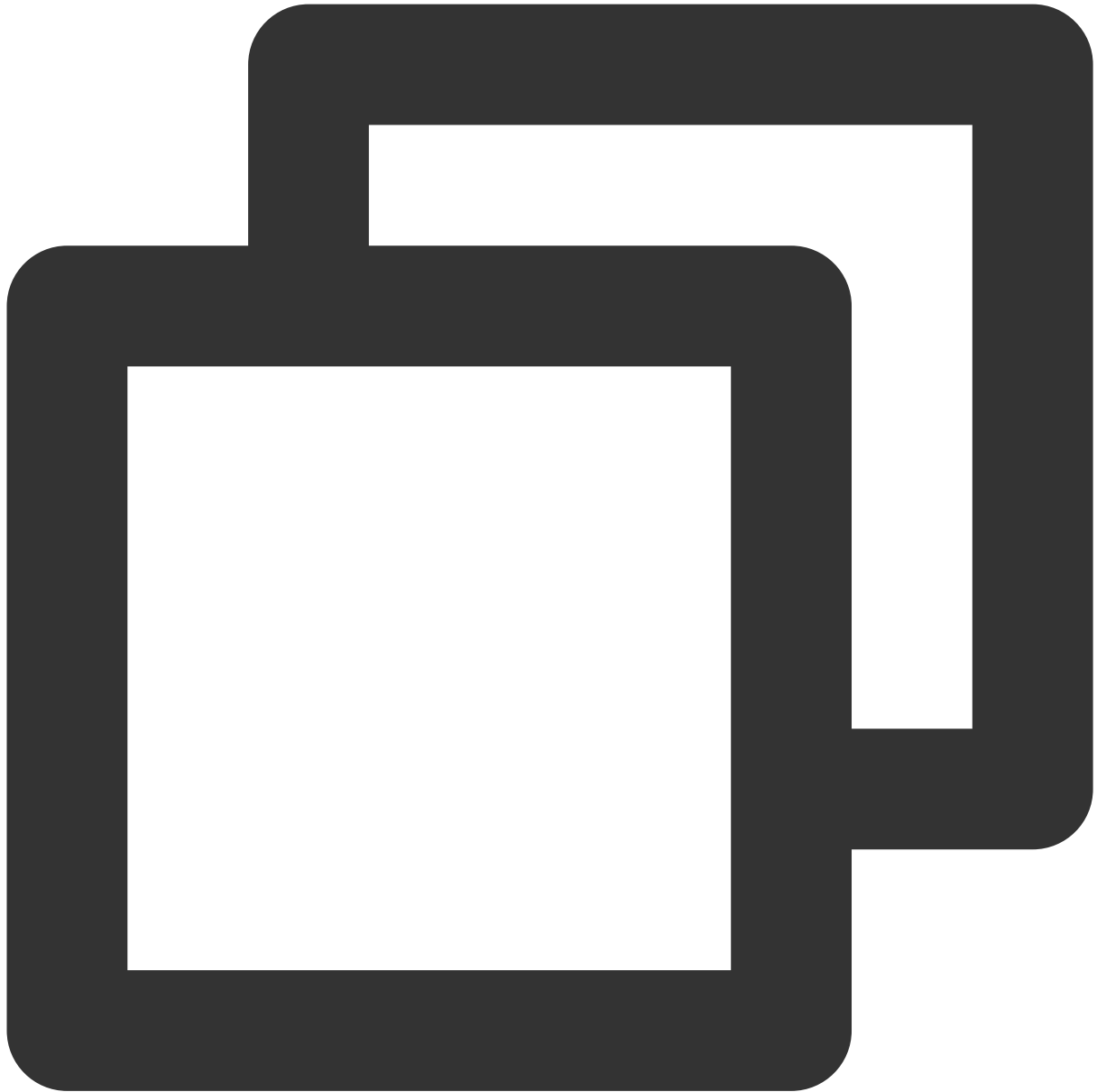
```
// src/main.ts
// Import locales/index.ts file, Please confirm whether the import path is correct
import VueI18n from './TUIRoom/locales/index';

app.use(VueI18n);
```

#### 4. Configuring esLint Checksums



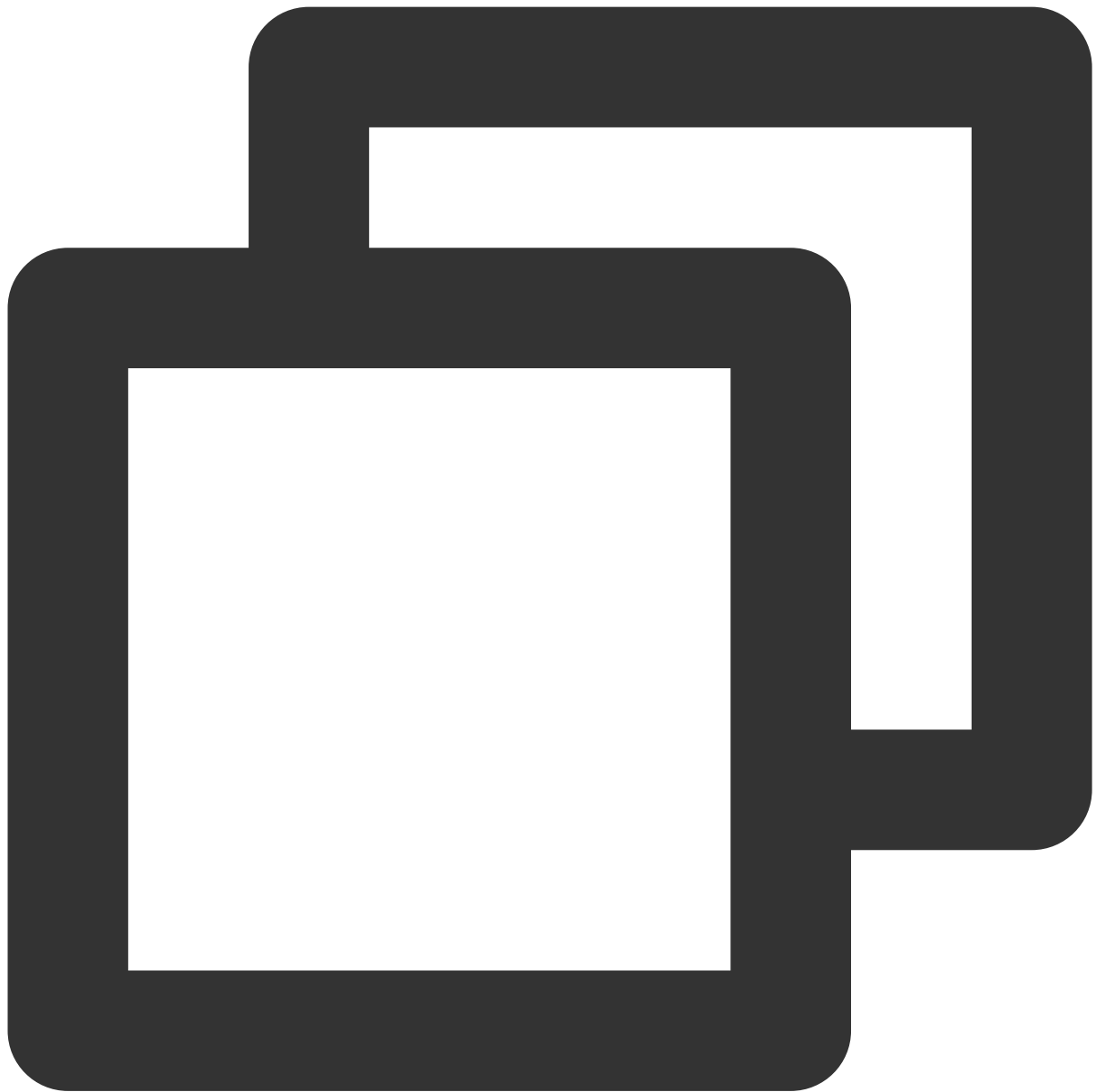
If you don't want conflicts between the esLint rules of the TUIRoomKit component and your local rules, you can add an ignore TUIRoom folder to .eslintignore.



```
src/TUIRoom
```

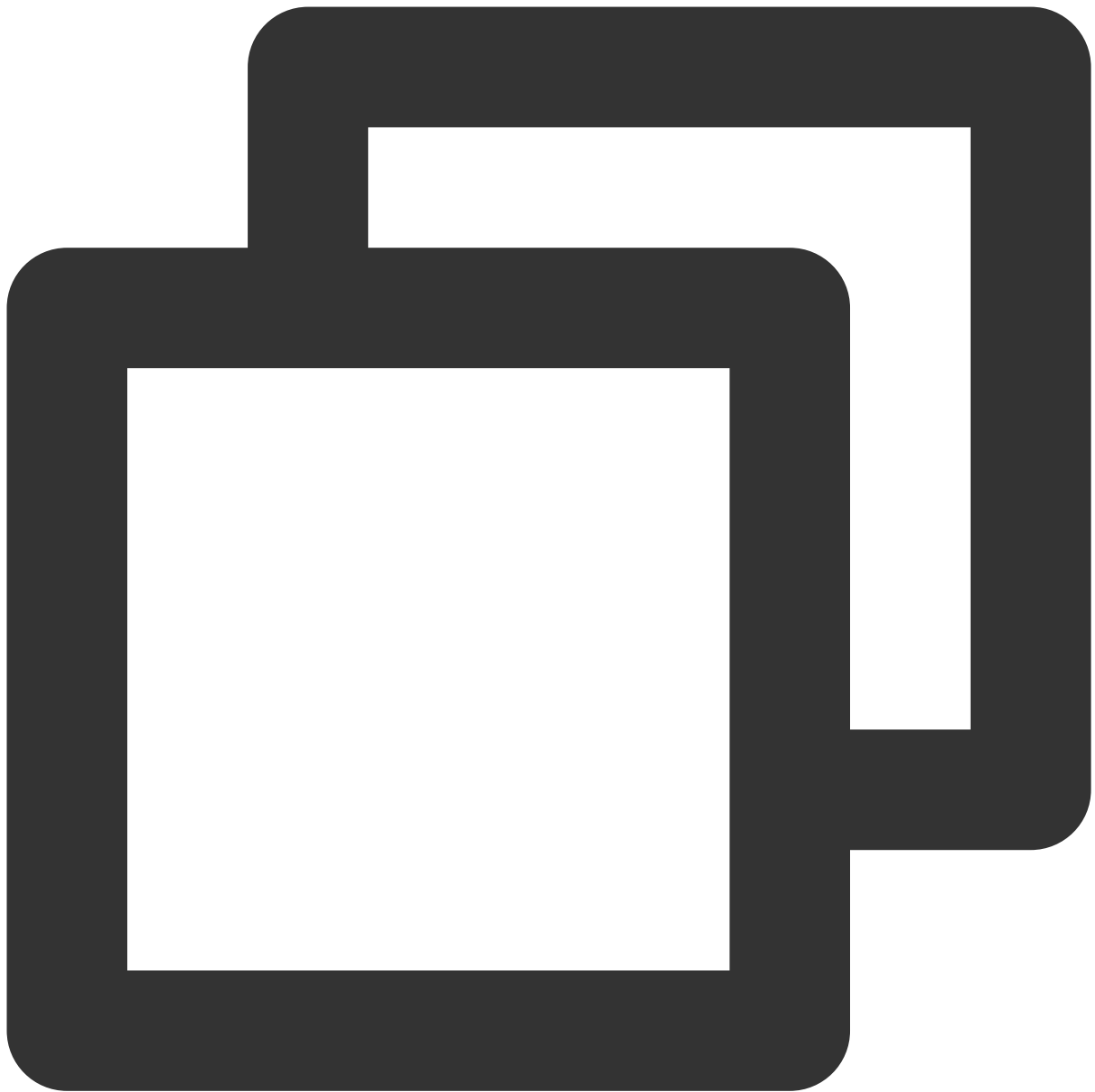
## 1. Install Dependencies

Install Development Environment Dependencies:



```
npm install sass sass-loader typescript -S -D
```

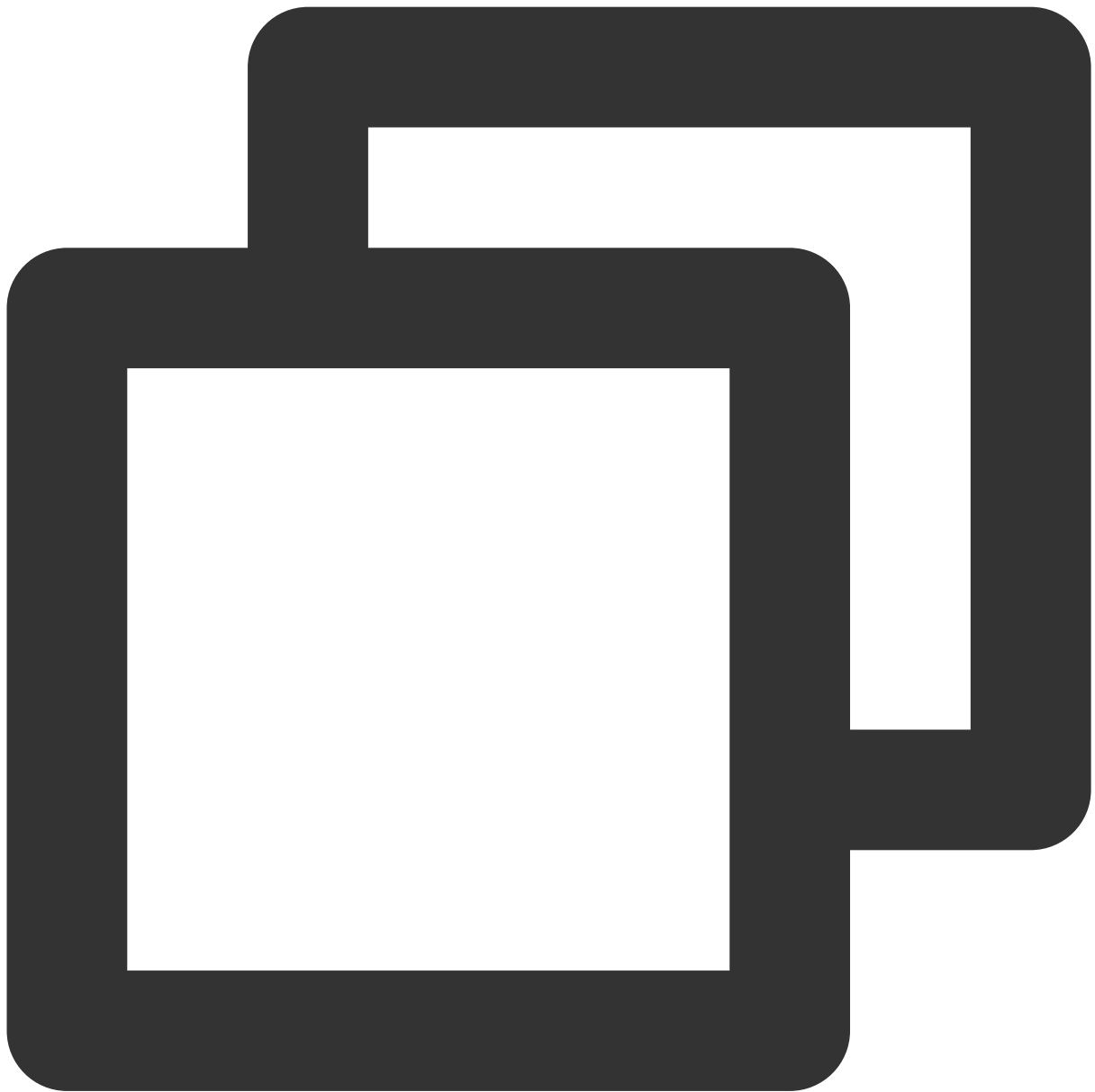
Install Production Environment Dependencies:



```
npm install mitt pinia @tencentcloud/tuiroom-engine-js vue-i18n rtc-detect -S
```

## 2. Register Pinia

TUIRoom uses Pinia for room data management, you need to register Pinia in the project entry file, which is the `src/main.ts` file.

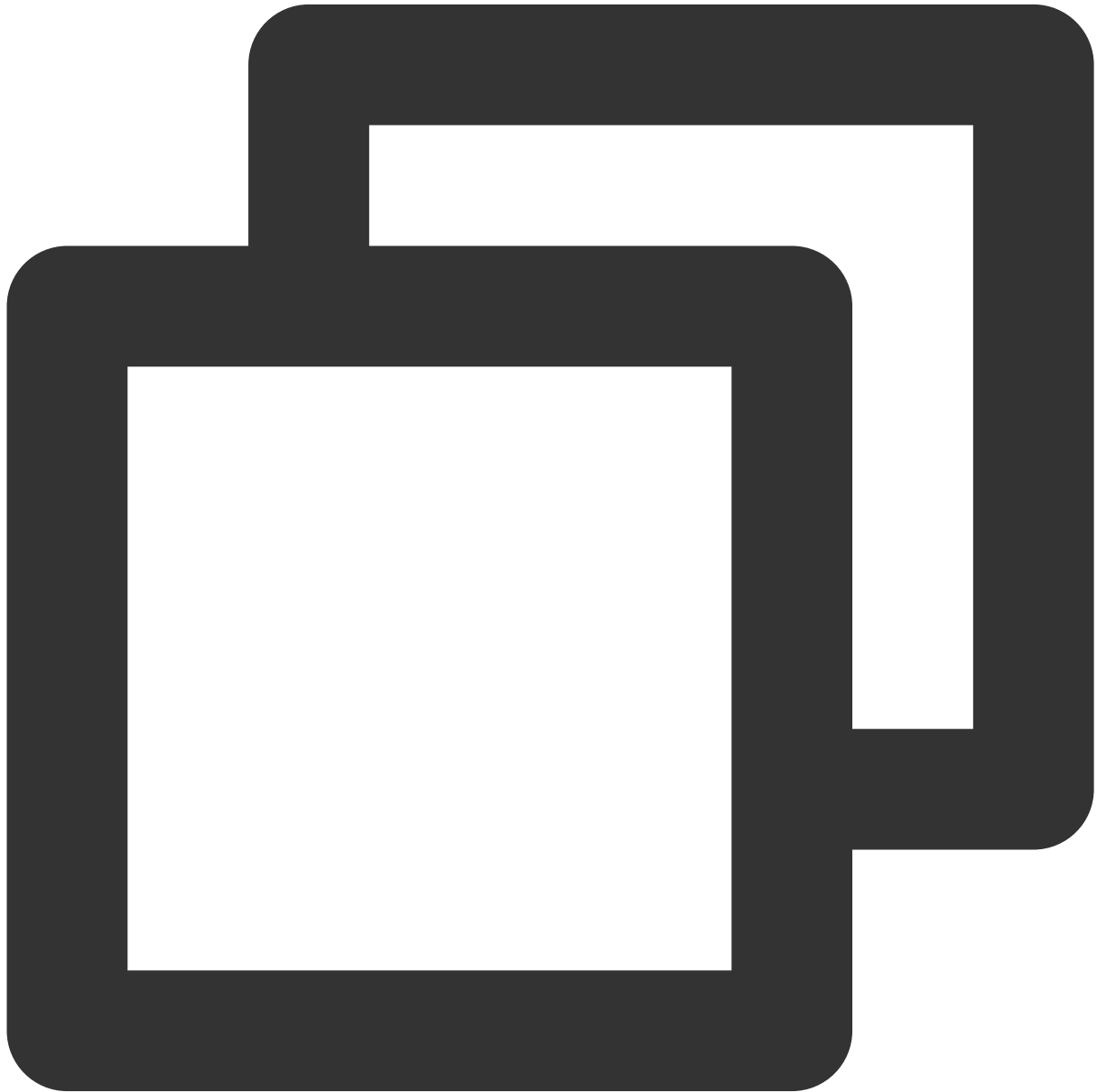


```
// src/main.ts file
import { createPinia } from 'pinia';

const app = createApp(App);
// Register pinia
app.use(createPinia());
app.mount('#app');
```

### 3. Configure Chinese-English language switching

TUIRoom currently supports Chinese-English language switching, you need to register the i18n instance in the main.ts file.

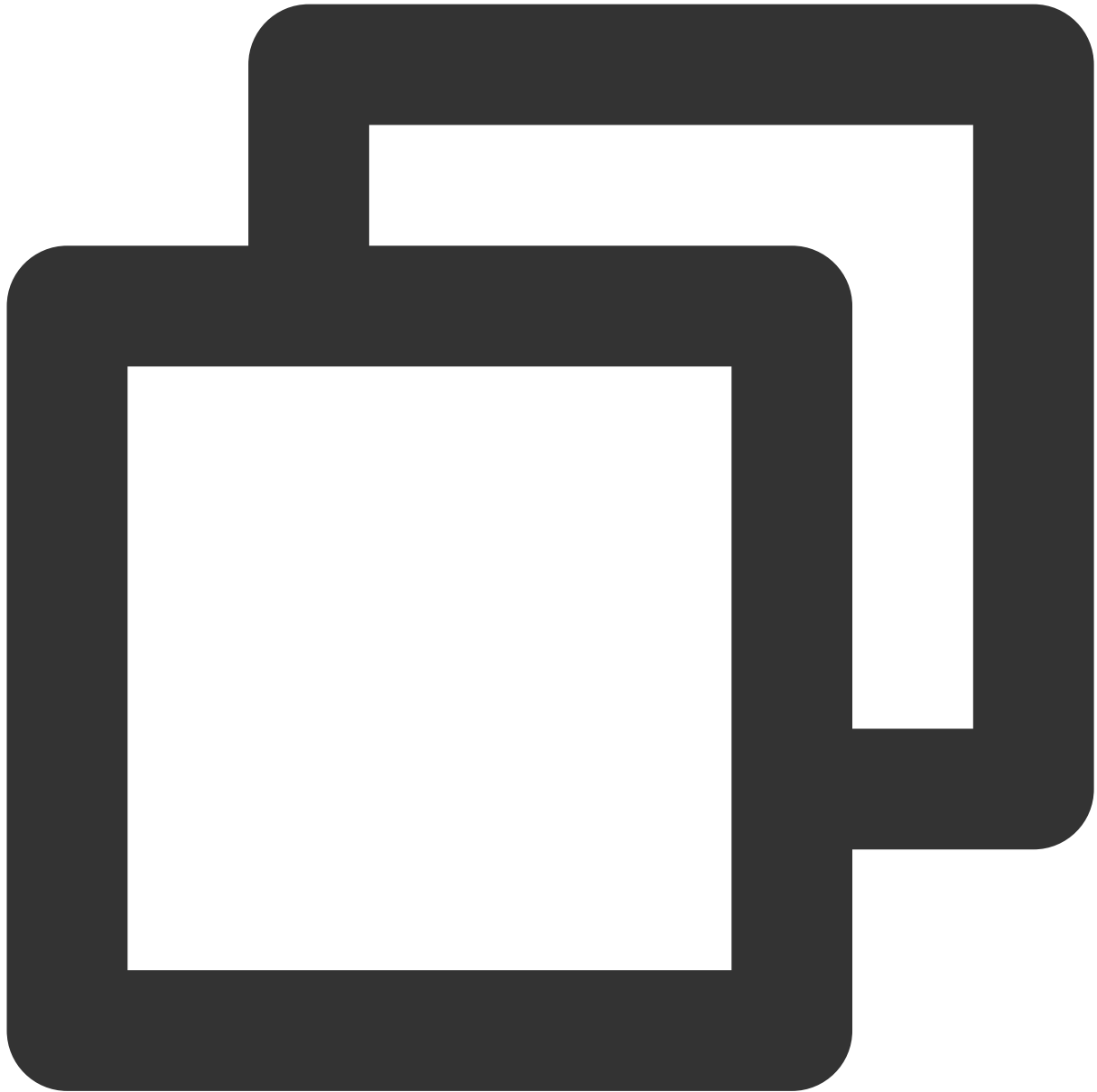


```
// src/main.ts
// Import locales/index.ts file, Please confirm whether the import path is correct
import VueI18n from './TUIRoom/locales/index';

app.use(VueI18n);
```

#### 4. Configure esLint Checksums

If you don't want conflicts between the esLint rules of the TUIRoomKit component and your local rules, you can add an ignore TUIRoom folder to .eslintignore.



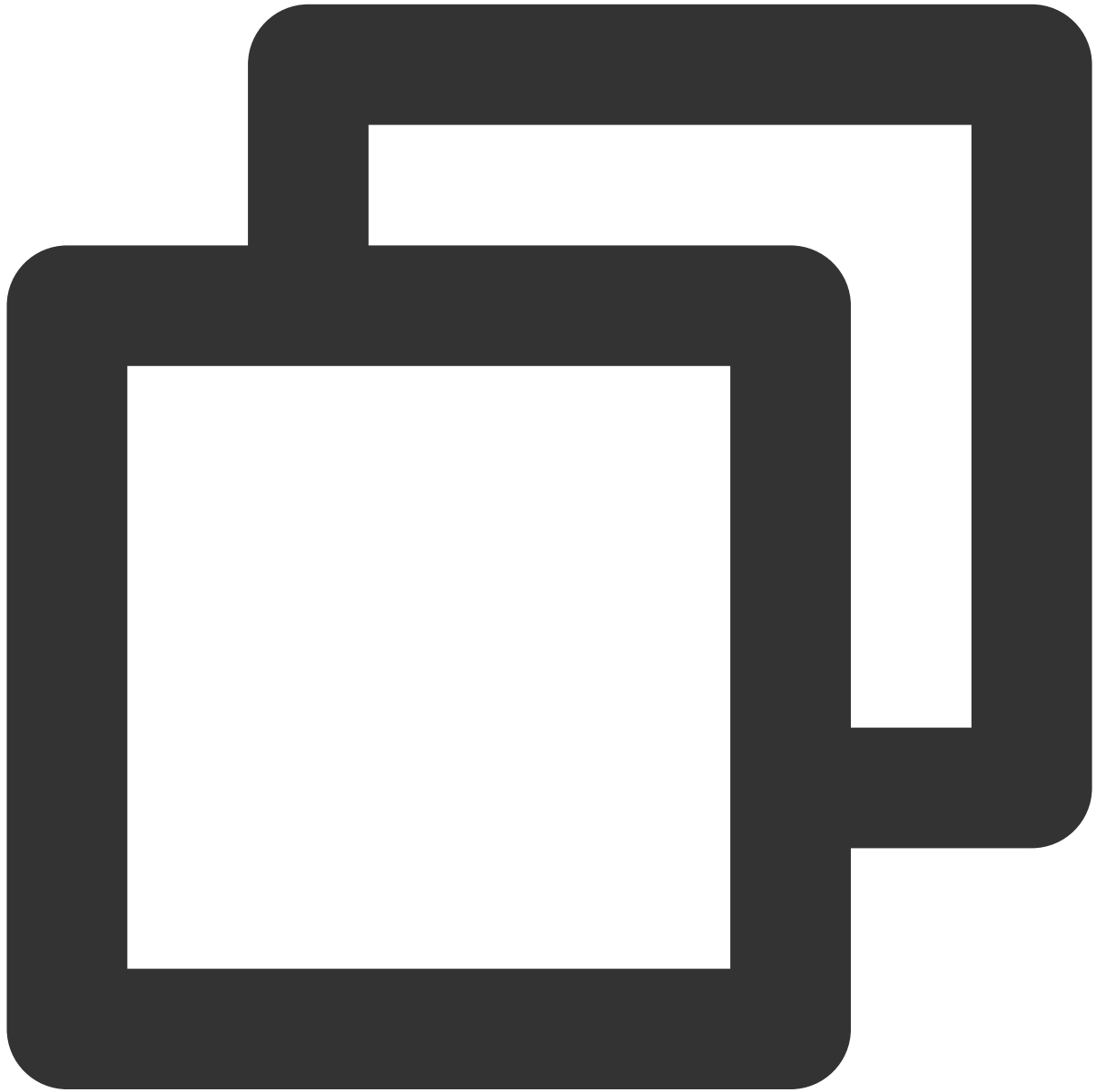
```
src/TUIRoom  
src/App.vue  
src/main.ts
```

**Note :**

TUIRoomKit component requires vue2 project to have vue2.7 installed and supports typescript environment. If your current project is on vue2.6 + js environment, follow the steps below to upgrade to vue2.7 + ts environment.

Upgrading from vue2.6 to vue2.7 is a smooth upgrade and has no impact on existing code. After configuring the ts environment, there is no impact on existing js code. Please feel free to upgrade.

### 1. Configure TypeScript, support TUIRoom Component loading



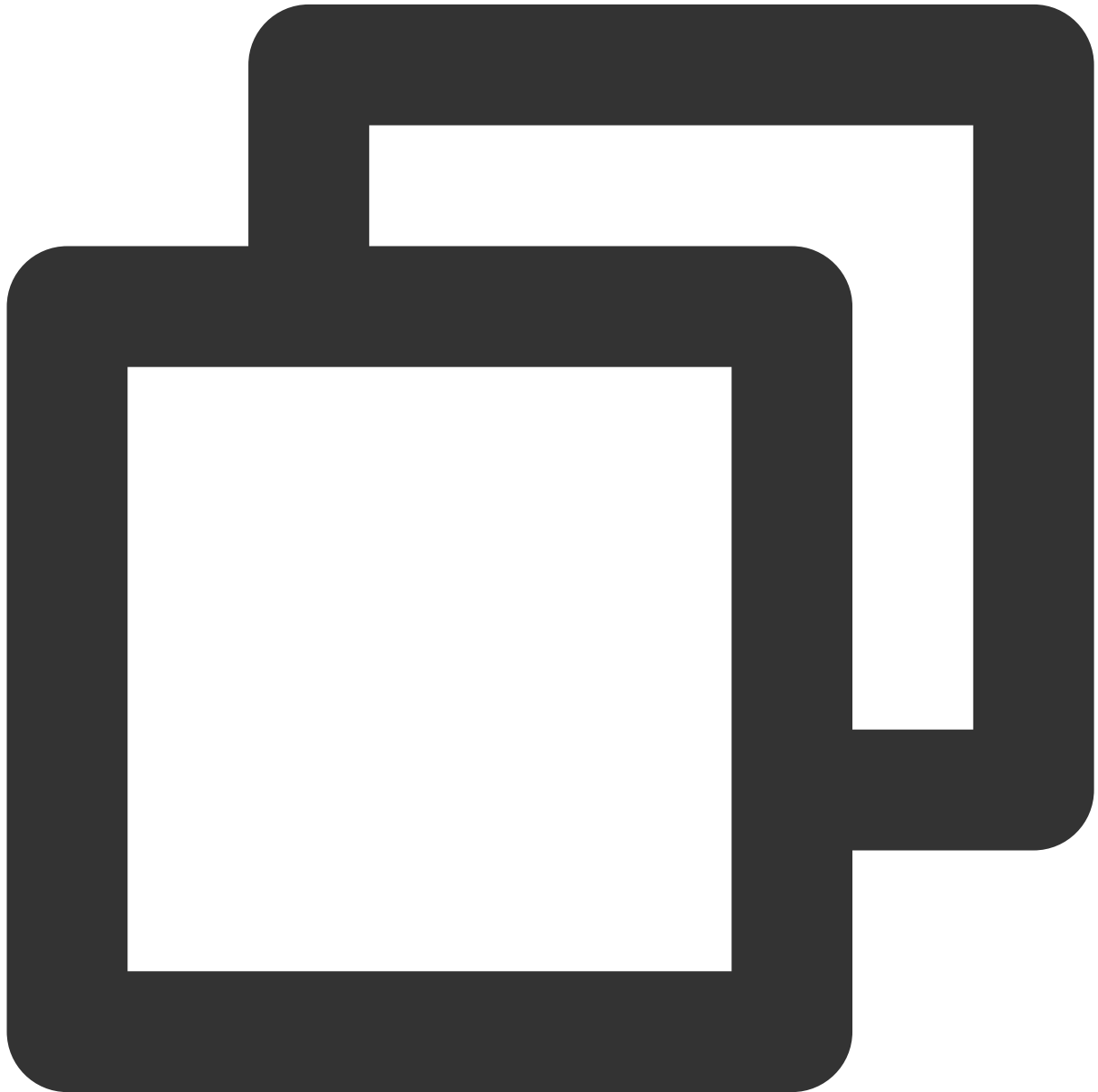
```
vue add typescript
```

Refer to the image for options to configure the TS development environment:

```
? Use class-style component syntax? No
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)? Yes
? Convert all .js files to .ts? No
? Allow .js files to be compiled? Yes
? Skip type checking of all declaration files (recommended for apps)? No
```

## 2. Install Dependencies

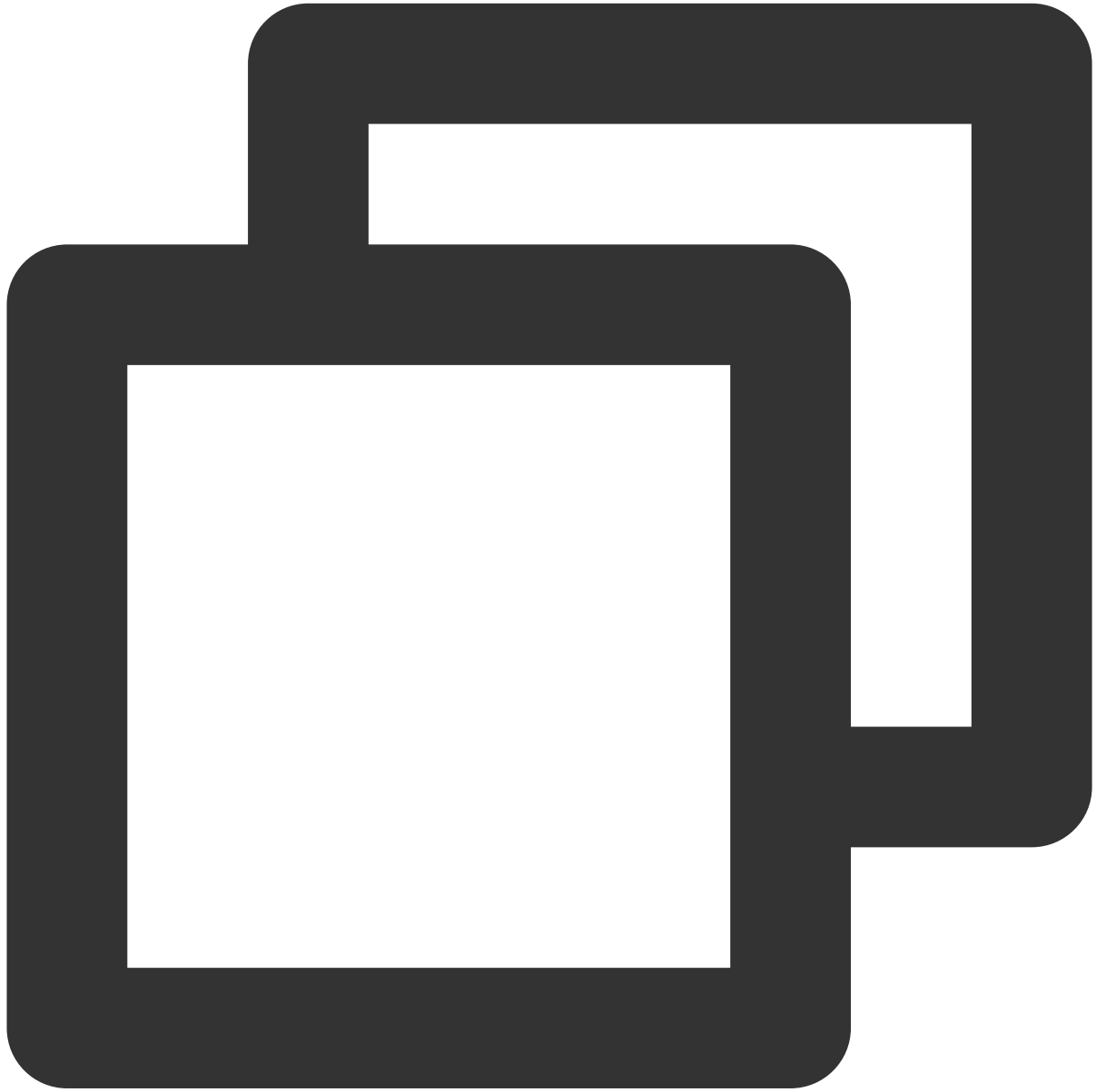
Install Development Environment Dependencies:



```
npm install sass sass-loader -S -D
```



Install Production Environment Dependencies:

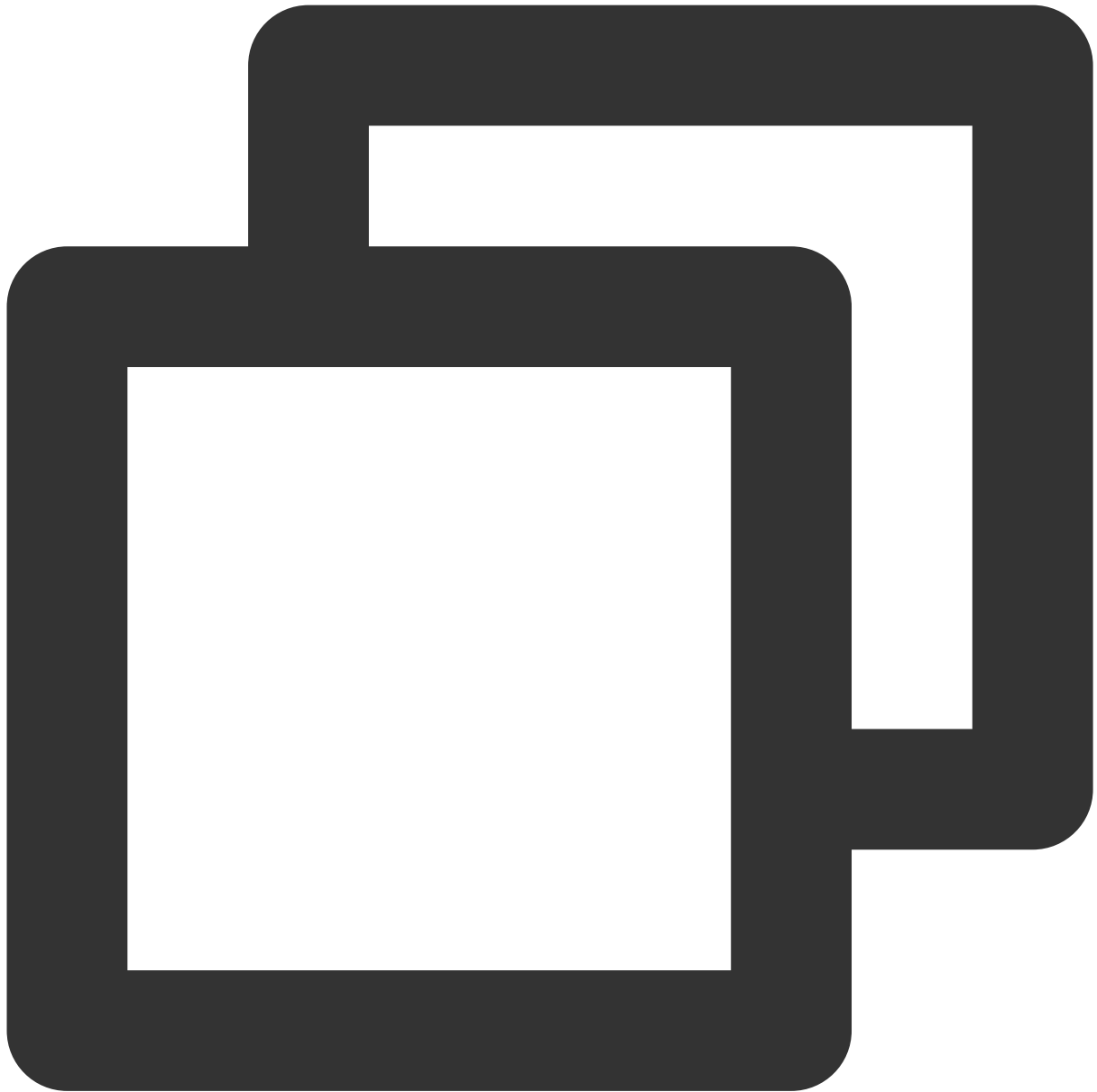


```
npm install vue@^2.7 pinia @tencentcloud/tuiroom-engine-js vue-i18n@^8 vue-i18n-bri
```

### 3. Register Pinia

TUIRoom uses Pinia for room data management, you need to register Pinia in the project entry file, which is the

`src/main.ts` or `src/main.js` file.



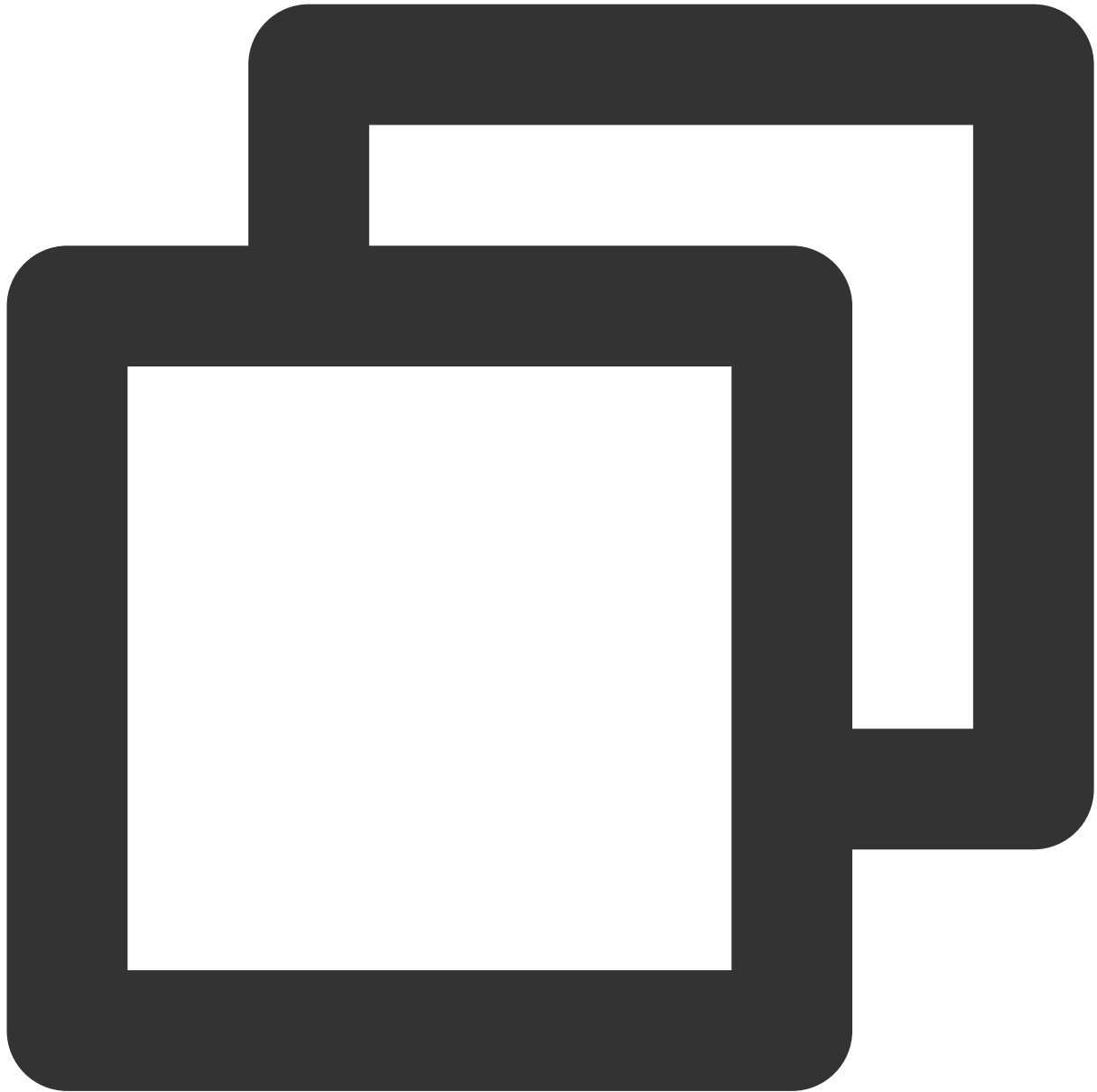
```
import { createPinia, PiniaVuePlugin } from 'pinia';

Vue.use(PiniaVuePlugin);
const pinia = createPinia();

new Vue({
  pinia,
  render: h => h(App),
}).$mount('#app');
```

#### 4. Configure Chinese-English language switching

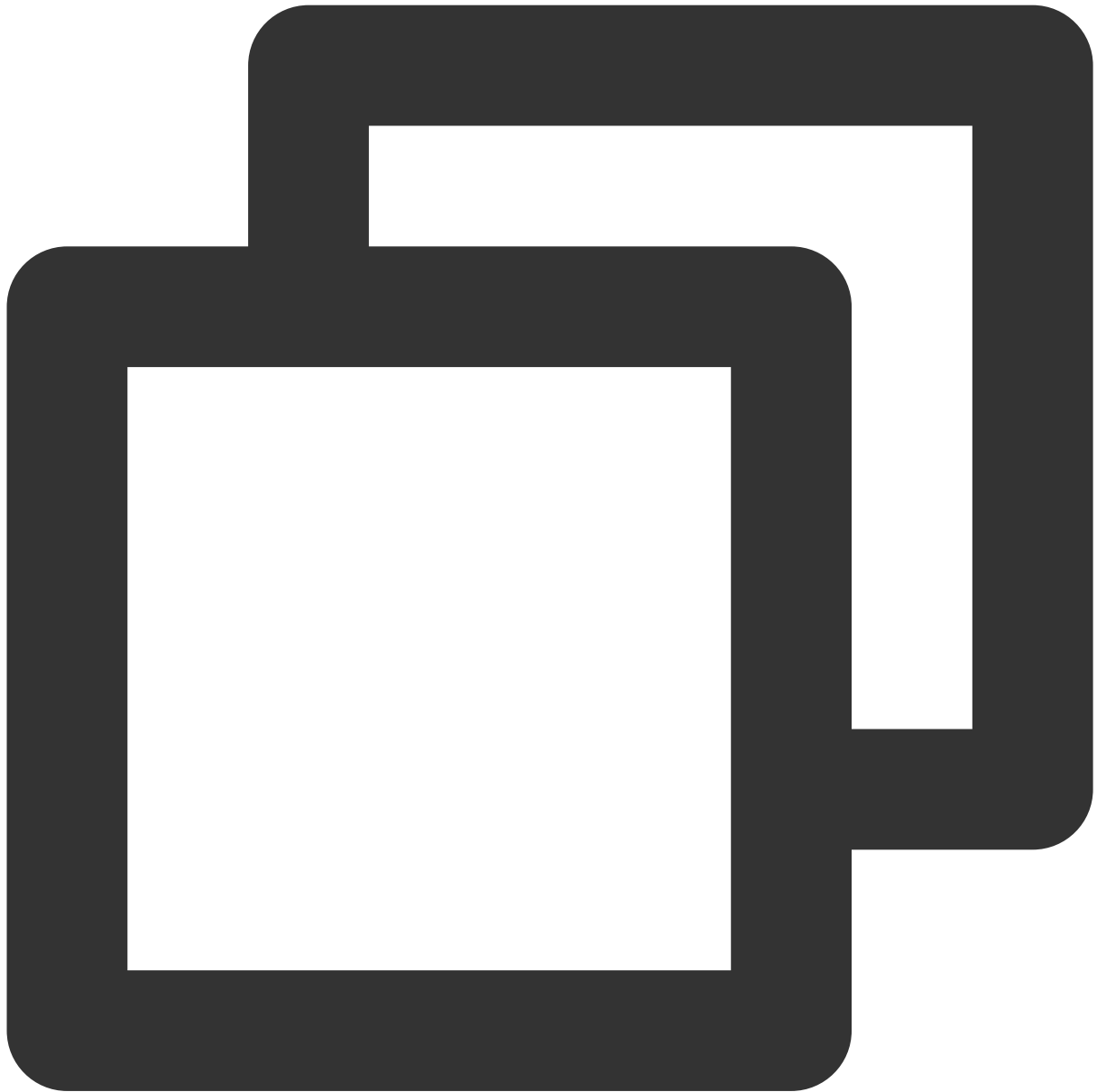
TUIRoom currently supports Chinese-English language switching, you need to register the i18n instance in the main.ts file.



```
import i18n from './TUIRoom/locales/';  
  
Vue.use(i18n);
```

### 5.Configure esLint Checksums

If you don't want conflicts between the esLint rules of the TUIRoomKit component and your local rules, you can add an ignore TUIRoom folder to .eslintignore.



```
src/TUIRoom
```

## Step 5: Run in the development environment

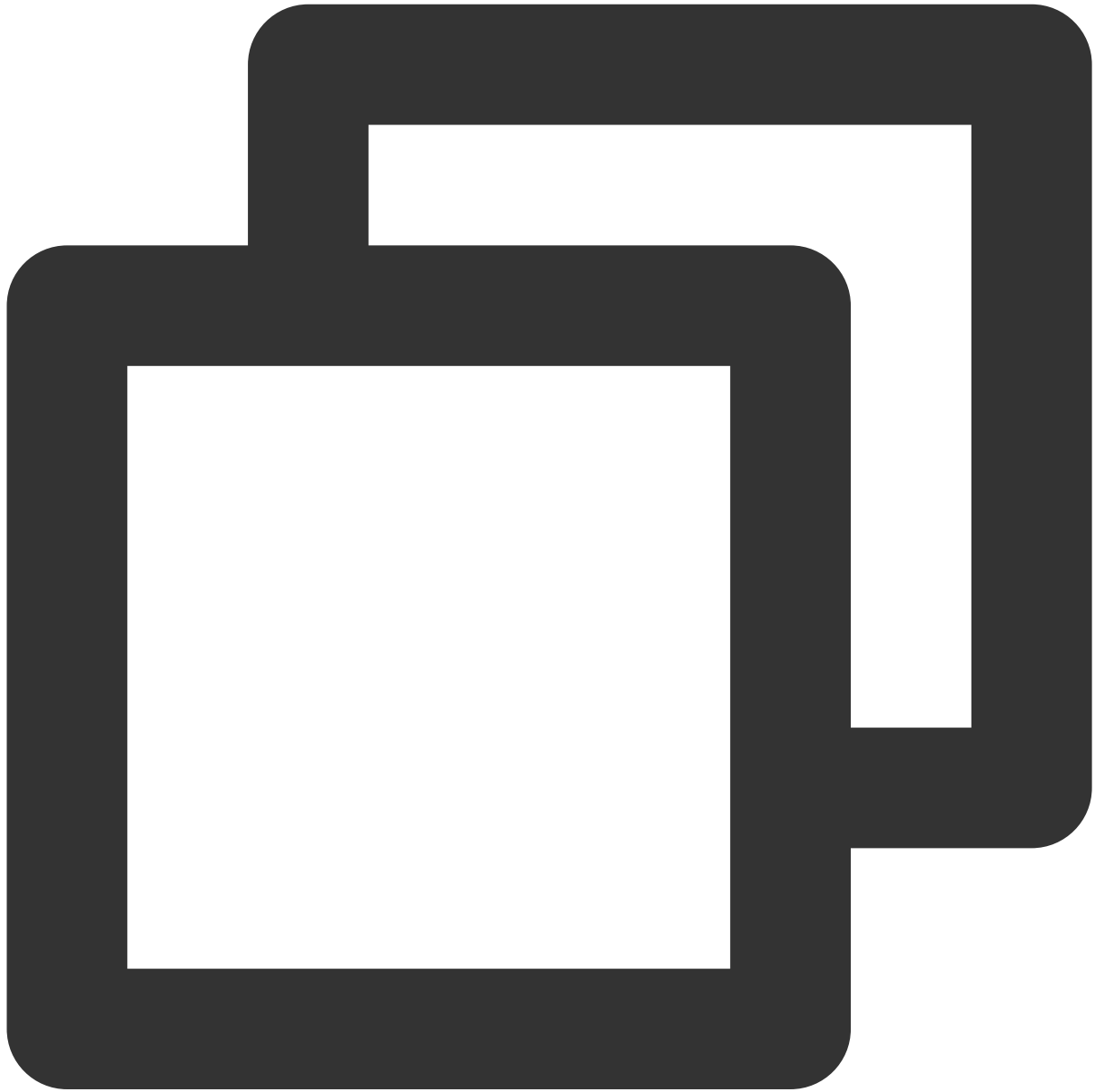
Execute the development environment running script in the console, open the page containing TUIRoomKit with a browser, and you can use the TUIRoomKit component in the page.

Vue3 + Vite + TS Project

Vue3 + Webpack + TS Project

Vue2 + Webpack + TS Project

1. Execute development environment command.



```
npm run dev
```

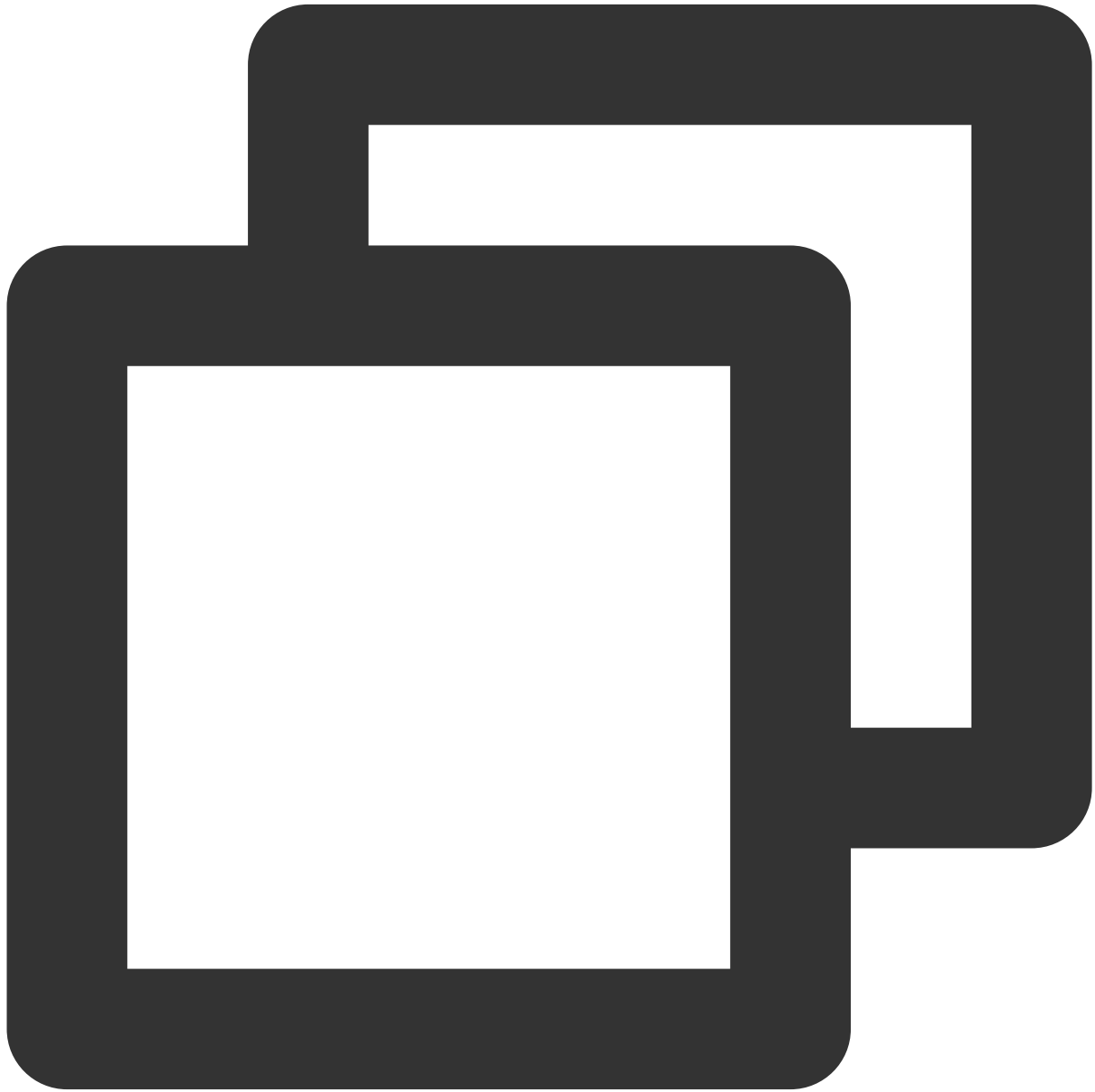
2. Open the page: `http://localhost:3000/` in the browser.

#### Note

Due to the on-demand import of element-plus components by TUIRoom, the response of the routing page in the development environment will be slow when it is loaded for the first time. Wait for the element-plus on-demand loading

to complete, and it can be used normally. The on-demand loading of element-plus will not affect the loading of the page after packaging.

1. Execute development environment command



```
npm run serve
```

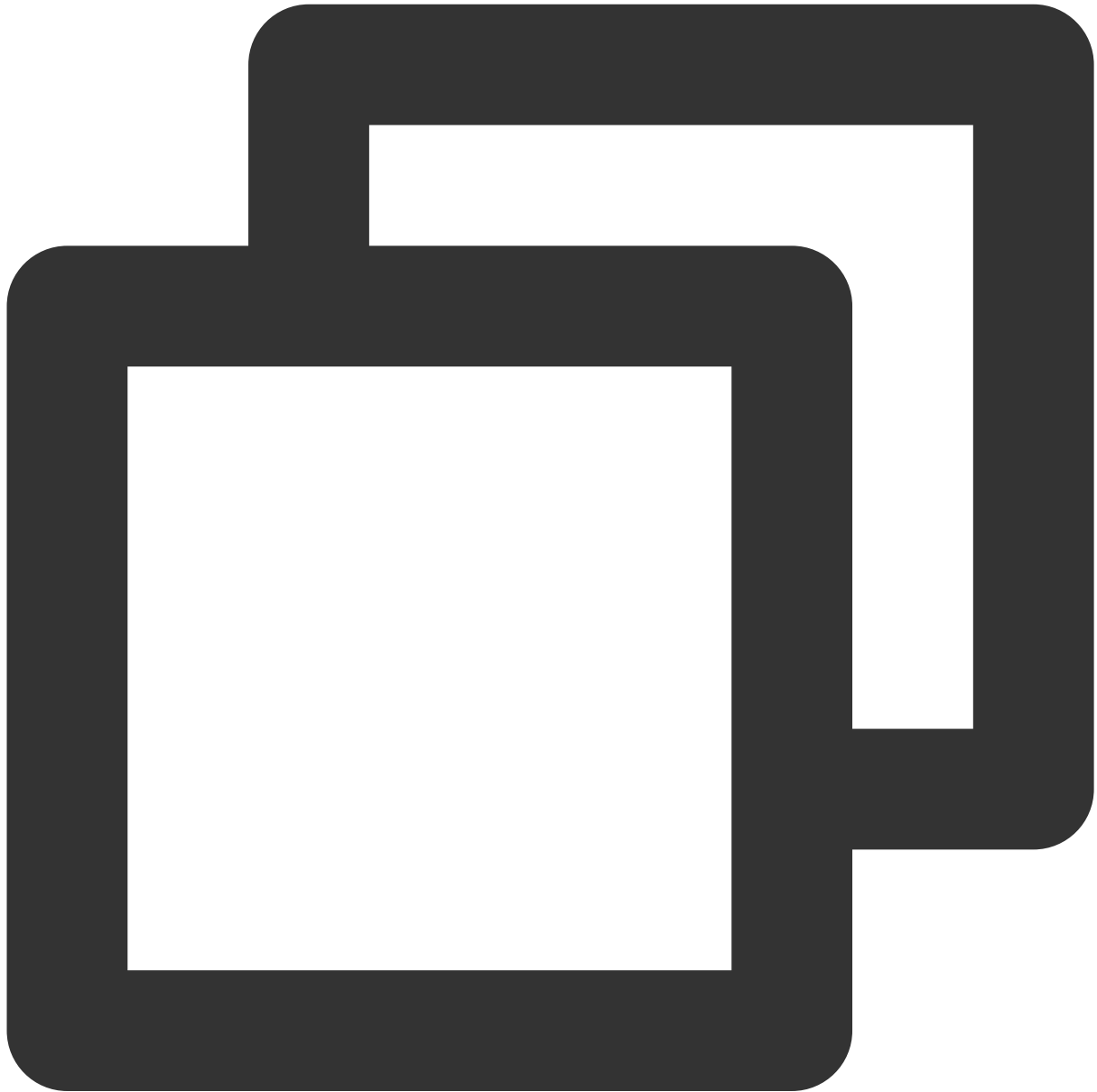
2. Open the page: `http://localhost:8080/` in the browser.

#### Note

If there are eslint errors in the `src/TUIRoom` directory during the running process, you can add the `/src/TUIRoom/` path to the `.eslintignore` file to shield the eslint check.

### 3. Experience the functions of the TUIRoom component

#### 1. Execute development environment command



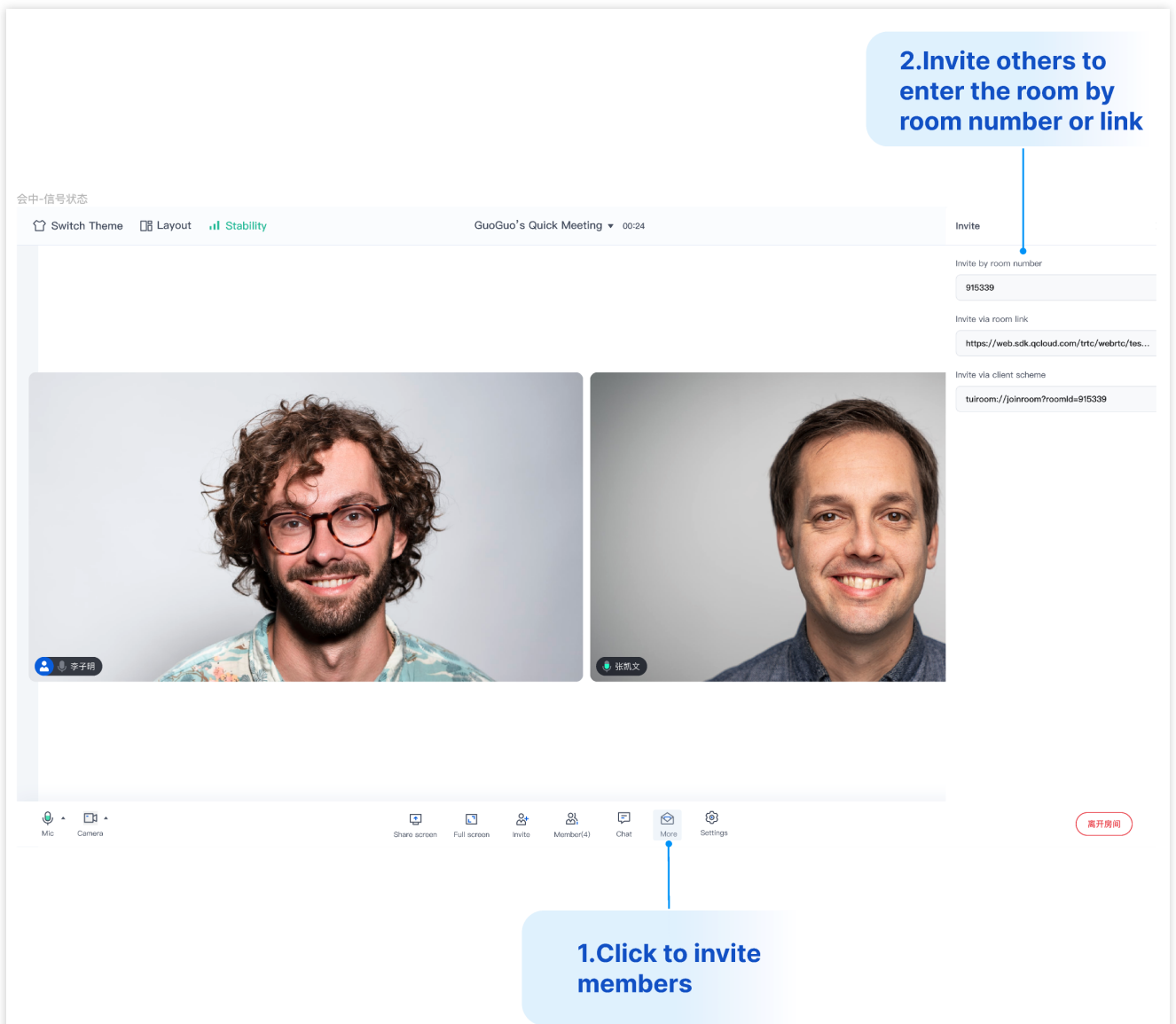
```
npm run serve
```

#### 2. Open the page: `http://localhost:8080/` in the browser.

#### Note

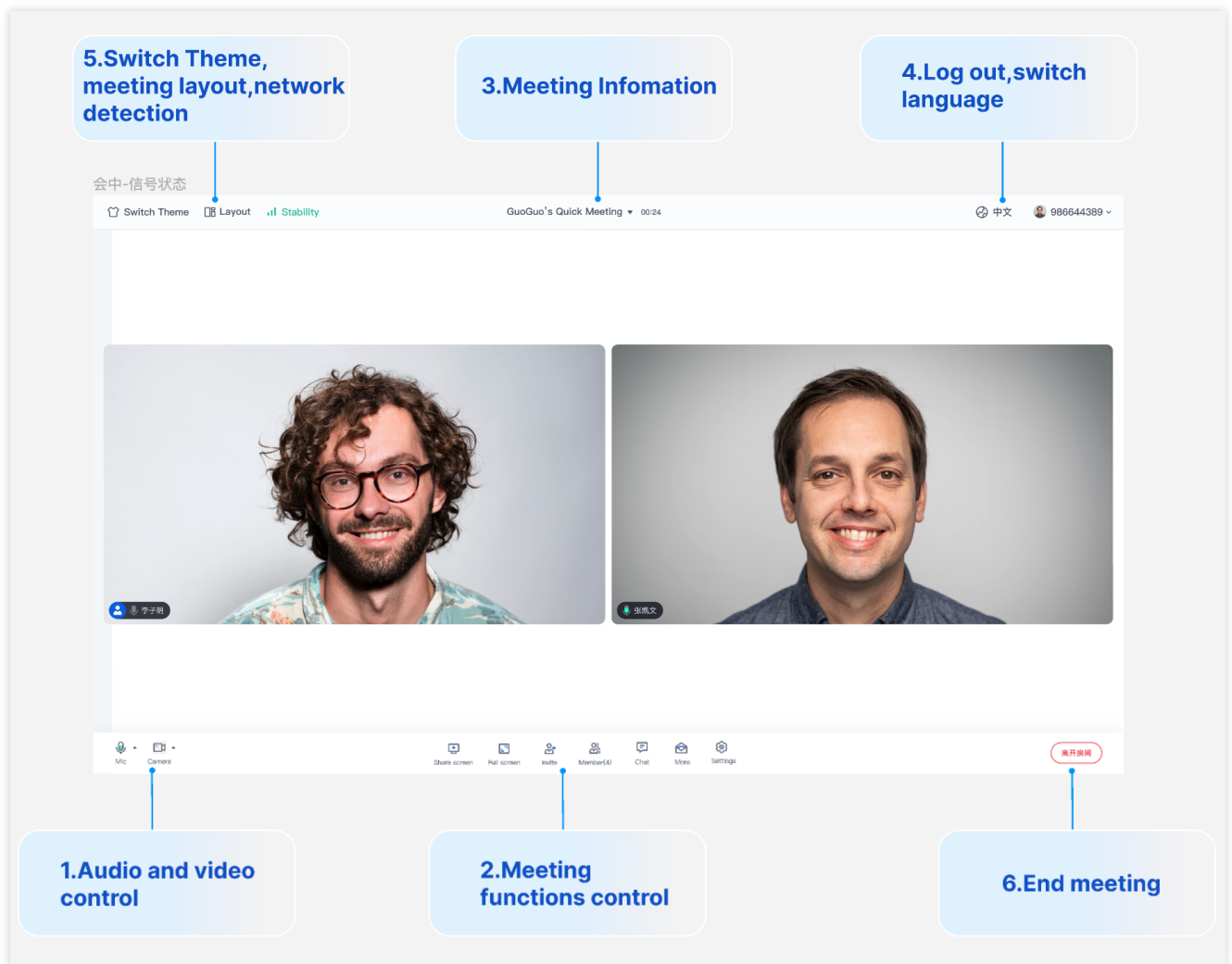
If there are eslint errors in the `src/TUIRoom` directory during the running process, you can add the `/src/TUIRoom/` path to the `.eslintignore` file to shield the eslint check.

After the project starts, click on Invite Members, and experience multi-user room interaction locally through the room link and client-side scheme invitation method.



Start your first meeting.





## Step 6: Deploy in the production environment

### 1. Package dist files



```
npm run build
```

**Note**

Please check the `package.json` file for the actual packaging command

2. Deploy dist files to the server

The production environment requires the use of an HTTPS domain name:

## Domain Names and Protocols

For security and privacy reasons, only HTTPS URLs can access all features of the TRTC SDK for web (WebRTC). Therefore, please use the HTTPS protocol for the web page of your commercial application.

Note: You can use `http://localhost` or `file://` URLs for local development.

The table below lists the supported URL domain names and protocols.

Scenario	Protocol	Receive (Playback)	Send (Publish)	Share Screen
Commercial	HTTPS	Supported	Supported	Supported
Commercial	HTTP	Supported	Not supported	Not supported
Local development	<code>http://localhost</code>	Supported	Supported	Supported
Local development	<code>http://127.0.0.1</code>	Supported	Supported	Supported
Local development	<code>http://[local IP address]</code>	Supported	Not supported	Not supported
Local development	<code>file://</code>	Supported	Supported	Supported

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# Flutter

Last updated : 2024-04-12 11:29:00

This article will introduce how to complete the integration of the `TUIRoomKit` component in the shortest time. By following this document, you will complete the following key steps within an hour and ultimately obtain an audio/video conference function with a complete UI interface.

## Environment Preparation

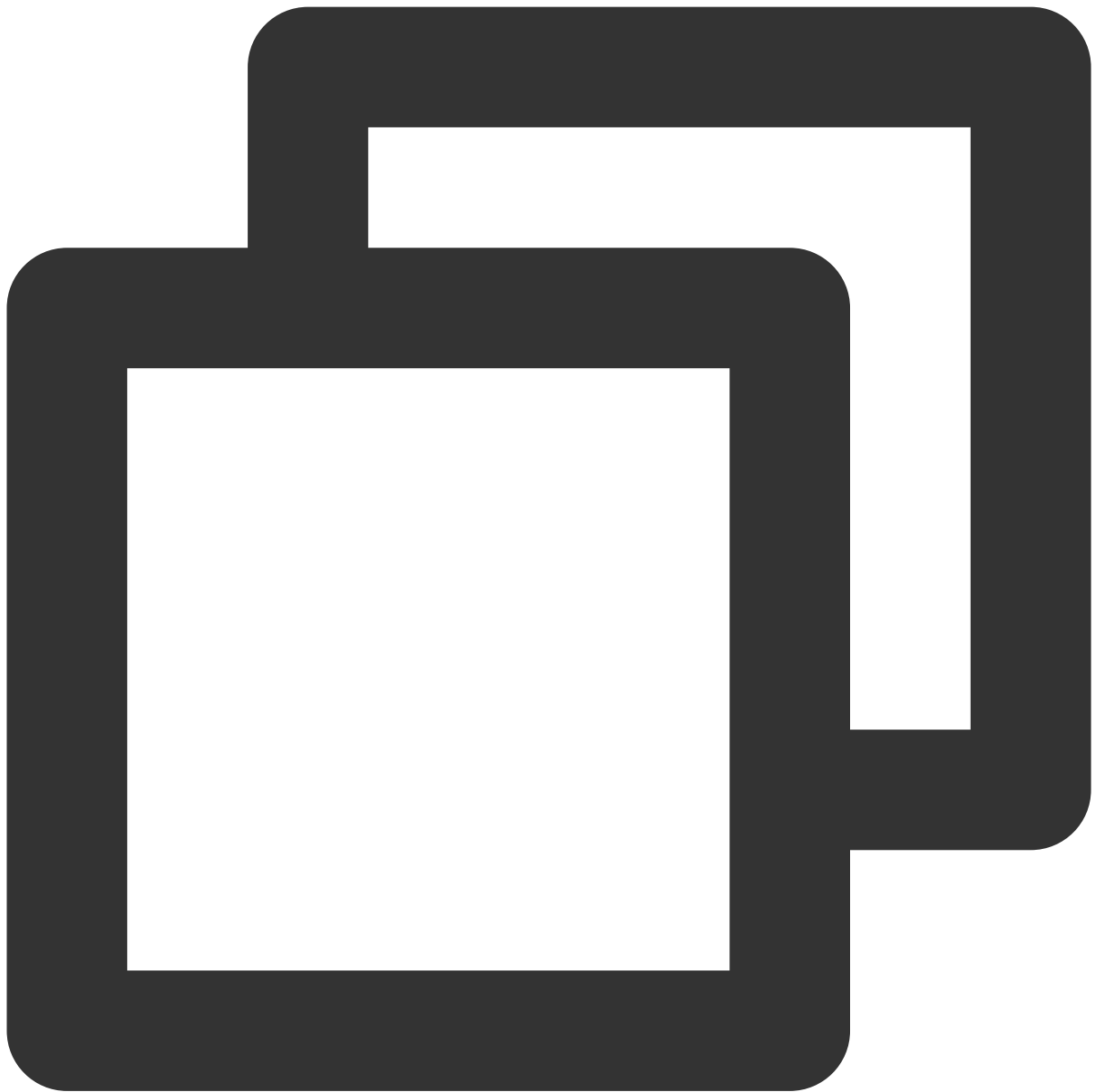
Platform	Version
Flutter	3.0.0 and above versions.
Android	Minimum compatibility with Android 4.1 (SDK API Level 16), recommended to use Android 5.0 (SDK API Level 21) and above. Android Studio 3.5 and above (Gradle 3.5.4 and above). Mobile devices with Android 4.1 and above.
iOS	iOS 12.0 and higher.

## Step 1: Activate the Service

Before initiating a meeting with TUIRoomKit, you need to activate the exclusive multi-person audio and video interaction service for TUIRoomKit on the console. For specific steps, please refer to [Activate Service](#).

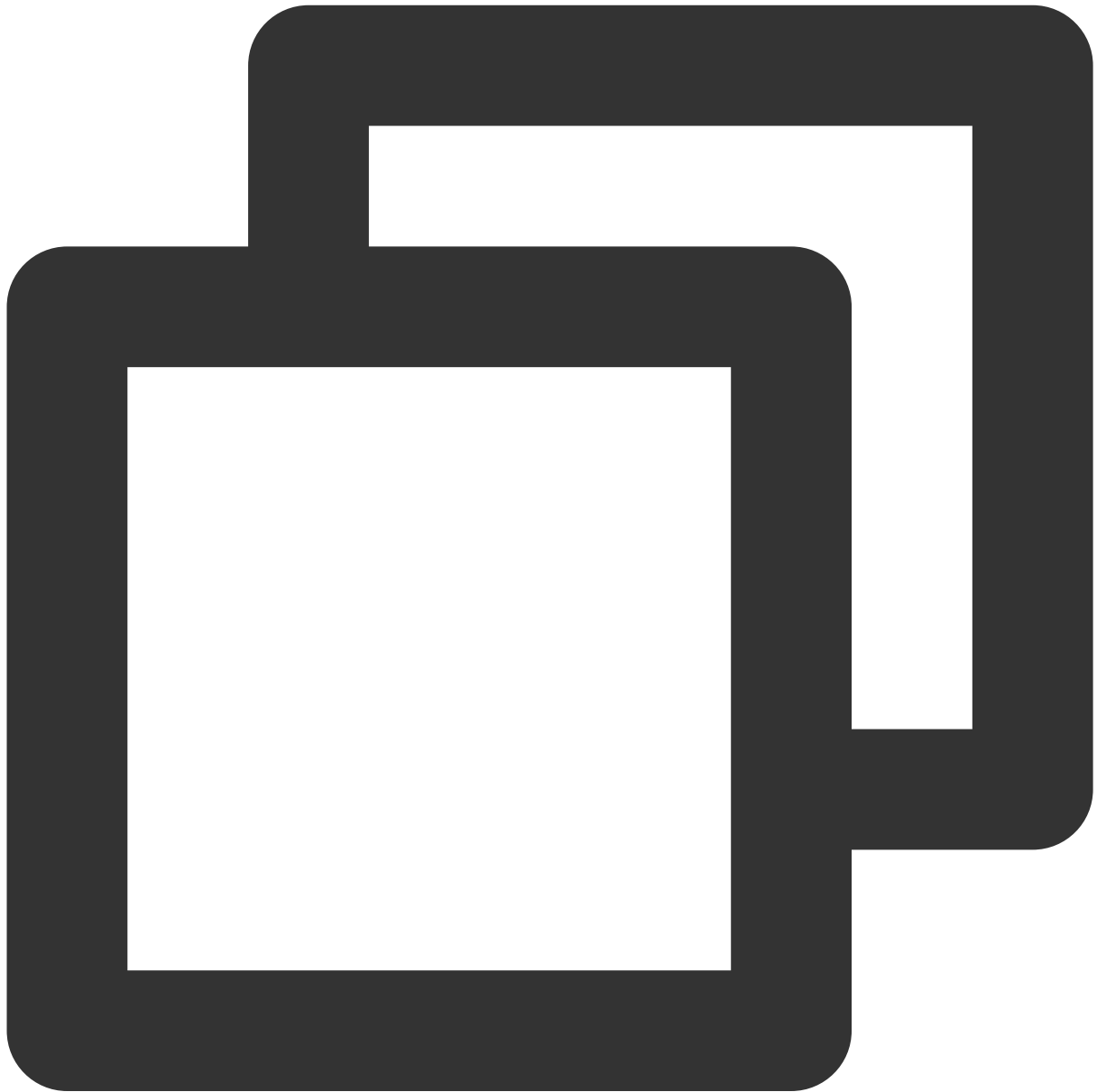
## Step 2: Integrate the TUIRoomKit Component

Add the `rtc_conference_tuikit` plugin dependency in pubspec.yaml file in your project



```
dependencies:  
  rtc_conference_tui_kit: The latest version
```

Execute the following command to install the plugin



```
flutter pub get
```

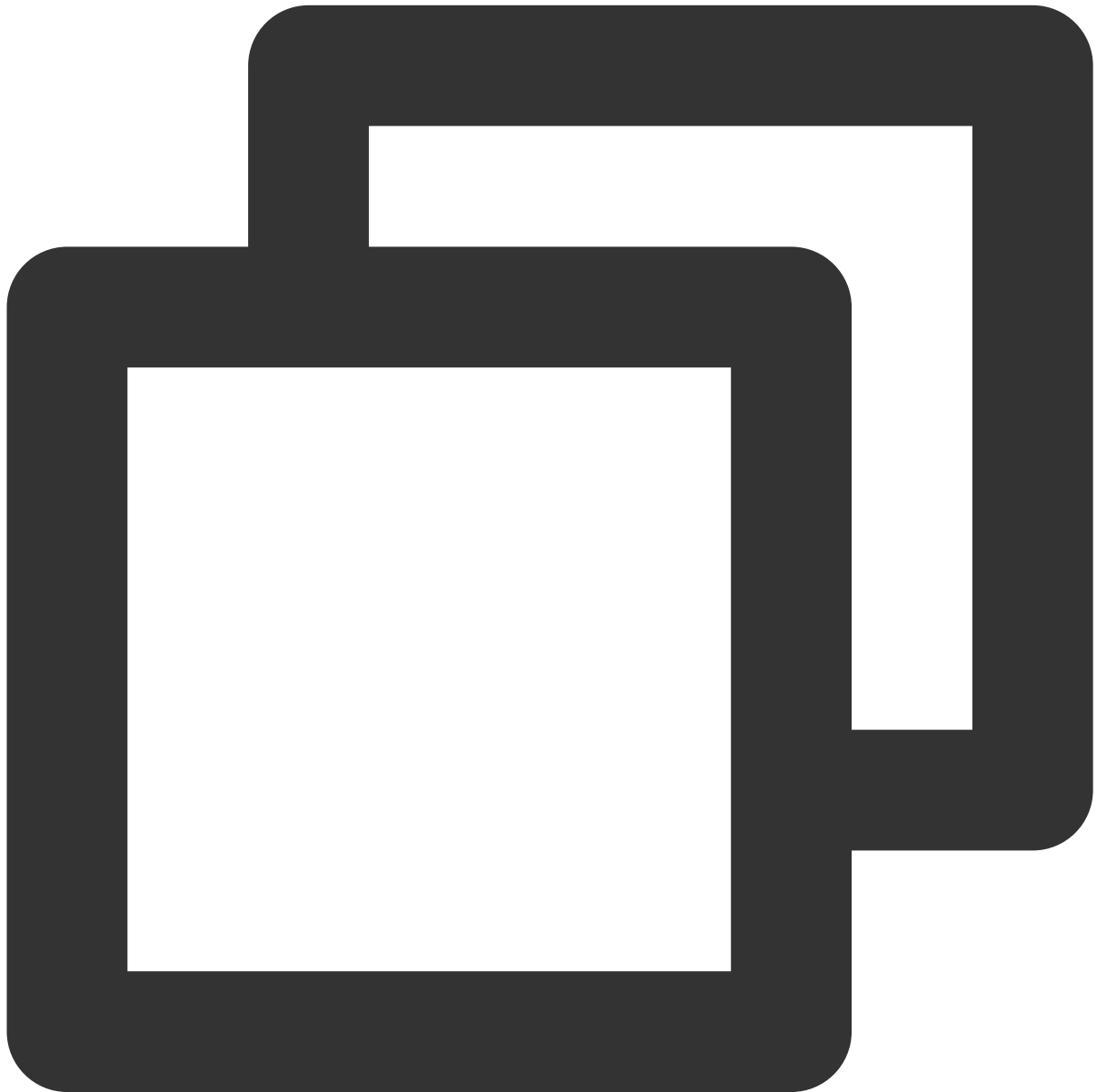
## Step 3: Complete Project Configuration

Since the `rtc_conference_tui_kit` has utilized the relevant features of the `GetX` state management library, you need to replace `MaterialApp` with `GetMaterialApp` in your application. Alternatively, you can set the `navigatorKey` attribute in your `MaterialApp` to `Get.key` for the same effect.

Open your project with **Xcode**, choose **Project > Building Settings > Deployment**, and set **Strip Style** to **Non-Global Symbols** to maintain the necessary global symbol information.

If you need to use audio and video features on **iOS**, authorize the microphone and camera usage rights ( **Android** has declared the relevant permissions in the SDK, and no manual configuration is necessary).

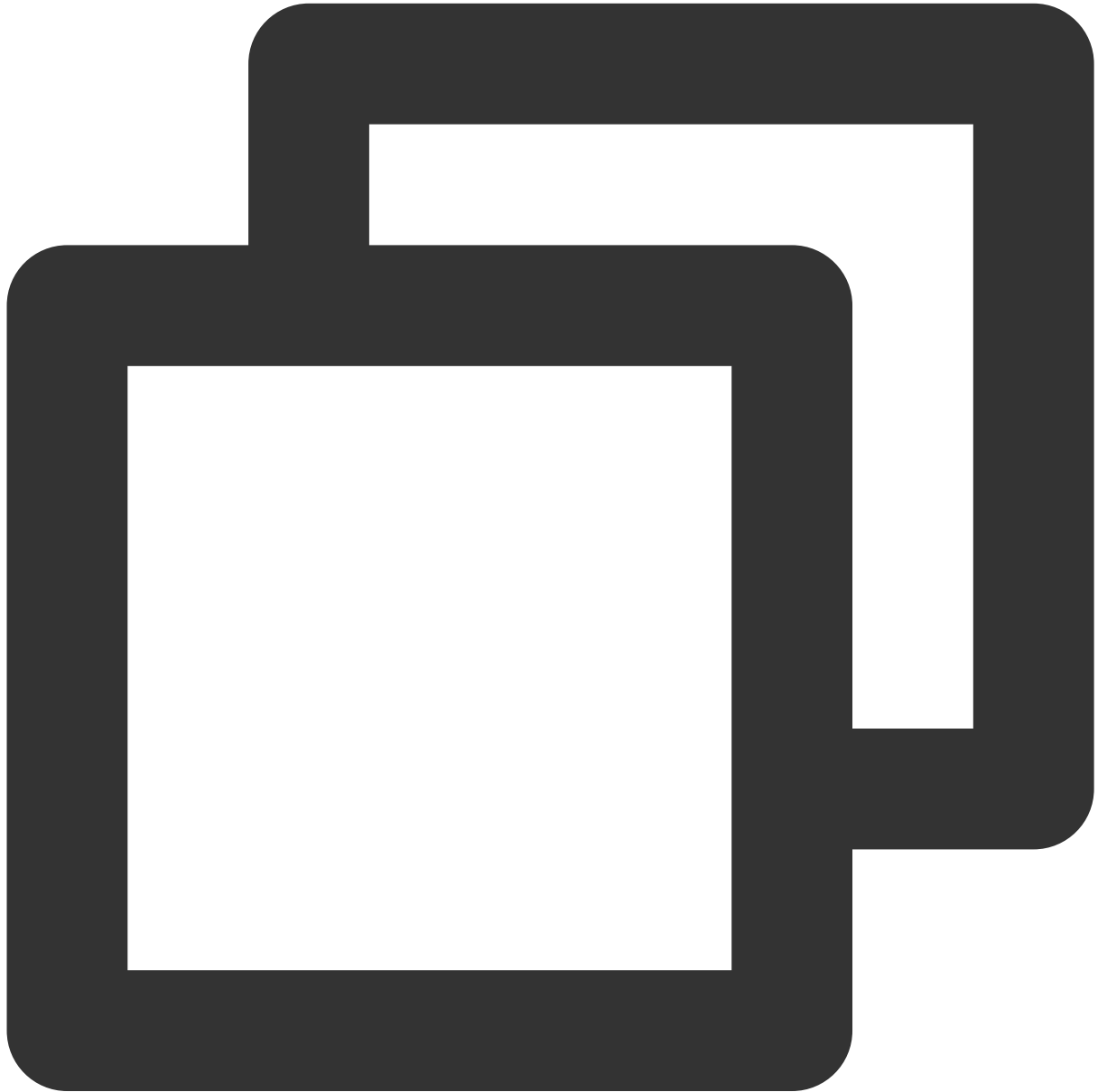
Add the following two items to your application's **Info.plist**. They correspond to the messages in the popup licensing dialog box while microphone and camera permissions are requested.



```
<key>NSCameraUsageDescription</key>
<string>TUIRoom requires access to your camera.</string>
<key>NSMicrophoneUsageDescription</key>
```

```
<string>TUIRoom requires access to your microphone.</string>
```

After the above are added, add the following preprocessor definitions in your **ios/Podfile** to enable camera and microphone permissions.



```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
    target.build_configurations.each do |config|
      config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||= [
        '$(inherited) ',
```

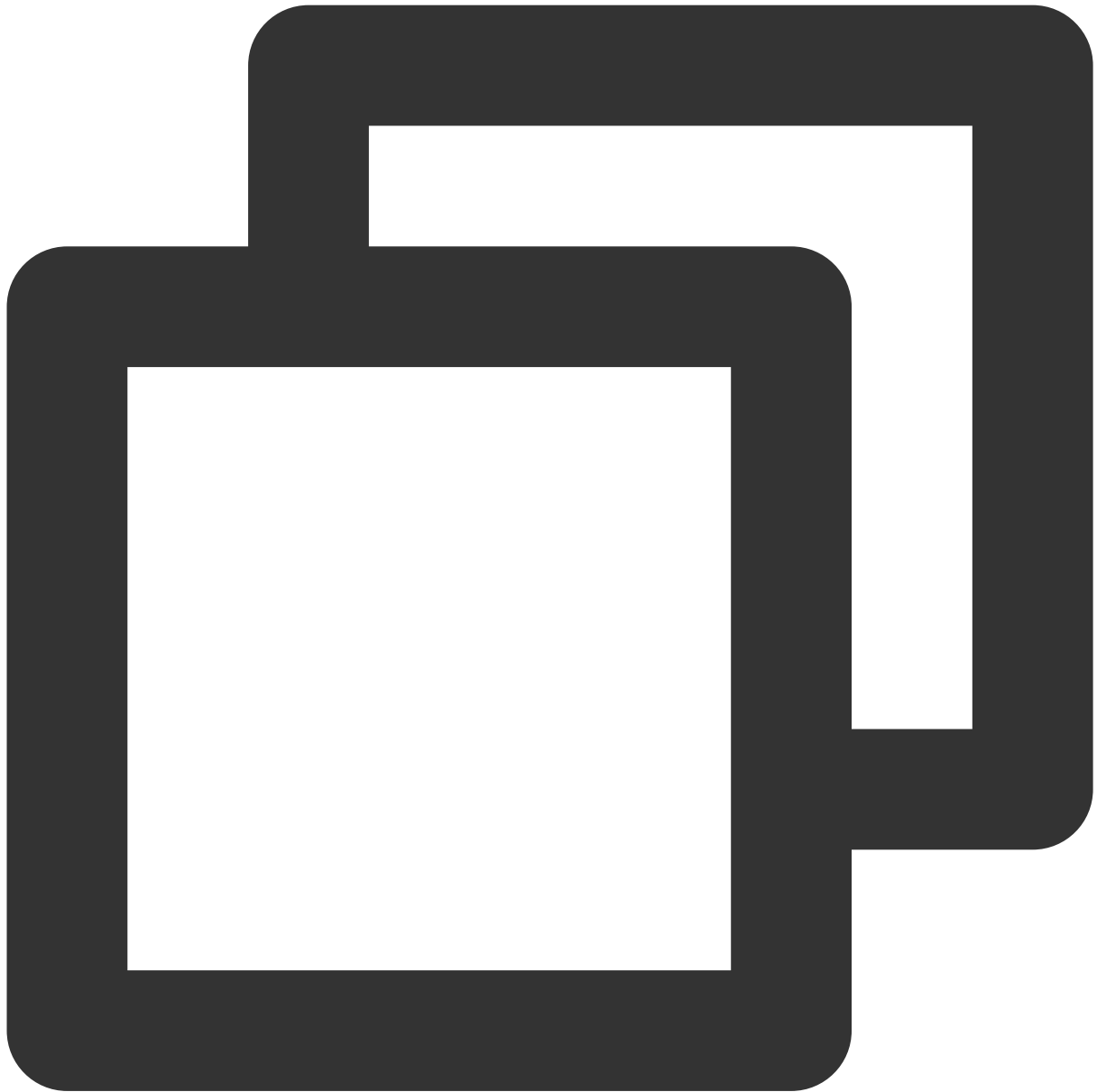


```
        'PERMISSION_MICROPHONE=1 ',  
        'PERMISSION_CAMERA=1 ',  
    ]  
end  
end  
end
```

## Step 4: Log in to TUIRoomKit Component

Add the following code to your project, which serves to log in to the component by calling the relevant interfaces in

`TUIRoomKit`. This step is extremely critical, as only after logging in can you use the various functions of `TUIRoomKit`, so please be patient and check if the relevant parameters are configured correctly:



```
import 'package:rtc_room_engine/rtc_room_engine.dart';

var result = await TUIRoomEngine.login(
  SDKAPPID,    // Please replace with your SDKAPPID
  'userId',    // Please replace with your user ID
  'userSig',   // Please replace with your userSig
);

if (result.code == TUIError.success) {
  // login success
} else {
```

```
// login error
}
```

## Parameter Description

Here is a detailed introduction to the key parameters used in the login function:

**SDKAppID** : You have already obtained it in [Step 1](#), so it will not be repeated here.

**UserID** : The ID of the current user, string type, only allows to contain English letters (a-z and A-Z), numbers (0-9), hyphens (-), and underscores (\_).

**UserSig** : Encrypt the SDKAppID, UserID, etc. with the SDK Secret Key obtained in [Step 1](#) to get the UserSig, which is a ticket for authorization and is used for Tencent Cloud to recognize whether the current user can use the TRTC service. You can create a temporarily available UserSig through the [UserSig Tools](#) through the project sidebar in the console.

For more information, please refer to the [UserSig related](#).

### Note:

**This step is also the step with the most feedback from developers we have received so far. Common problems are as follows:**

`SDKAppID` is set incorrectly. Please use the `SDKAppID` of the international site correctly, otherwise, you will not be able to access it.

`UserSig` is misconfigured as an encryption key ( `SDKSecretKey` ). `UserSig` is obtained by encrypting the `SDKAppID` , `UserID` , and expiration time with the `SDKSecretKey`, not by directly configuring the `SDKSecretKey` as `UserSig` .

`UserID` is set to simple strings like "1", "123", "111", etc. **Since TRTC does not support multi-terminal login with the same UserID**, simple UserIDs like "1", "123", "111" are easily occupied by your colleagues, causing login

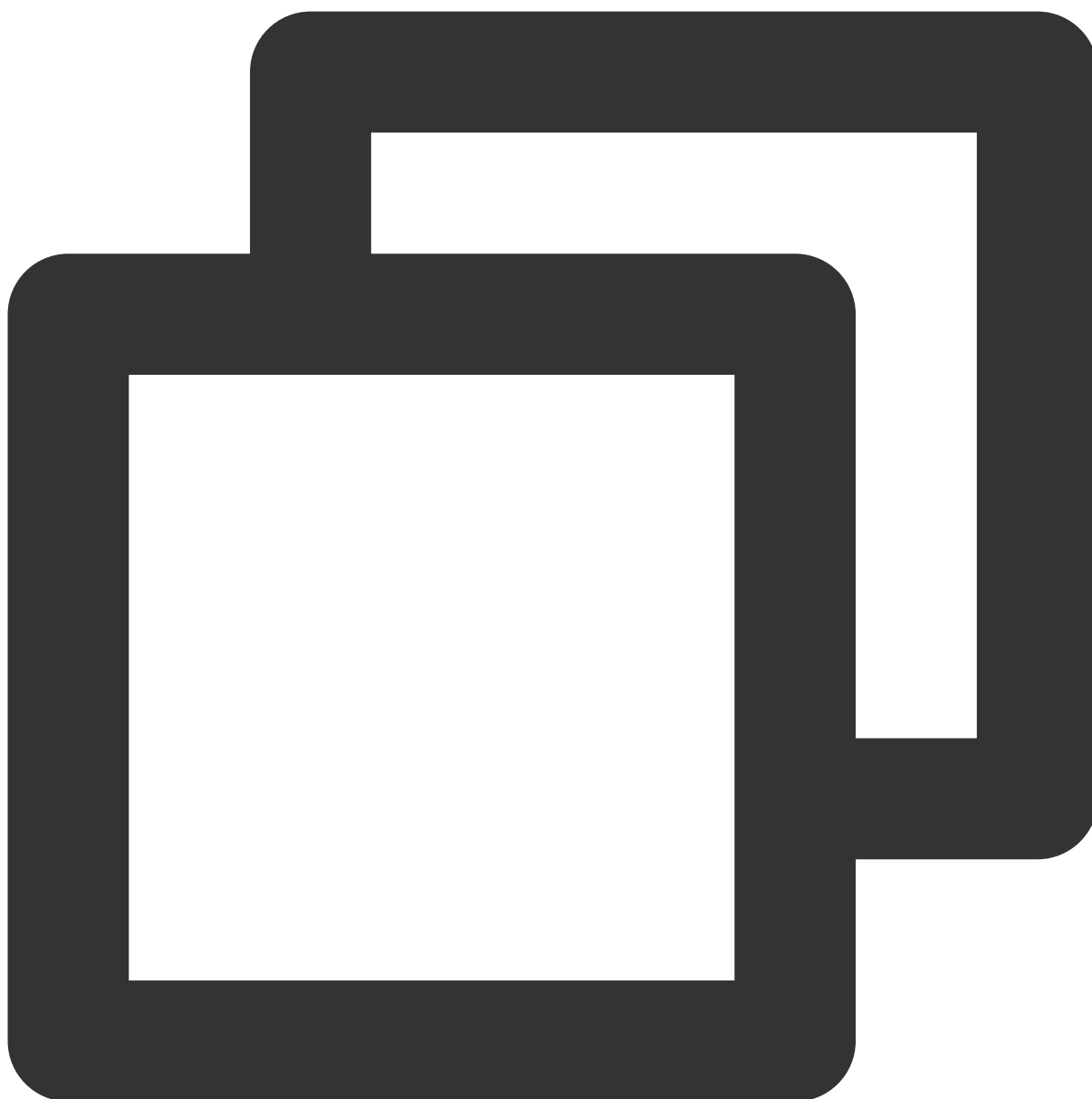
failure. Therefore, we recommend that you set some UserIDs with high identifiability when debugging.

The [sample code](#) in Github uses the `genTestUserSig` function to calculate UserSig locally to quickly get you through the current access process. However, this solution exposes your SDKSecretKey in the App code, which is not conducive to your subsequent upgrades and protection of your `SDKSecretKey`. Therefore, we strongly recommend that you put the calculation logic of `UserSig` on the server side and have the app request the real-time calculated UserSig from your server every time it uses the TUIRoomKit Component.

## Step 5: Use TUIRoomKit

### Set User Information (optional)

You can set the username and avatar by calling TUIRoomEngine's `setSelfInfo`.



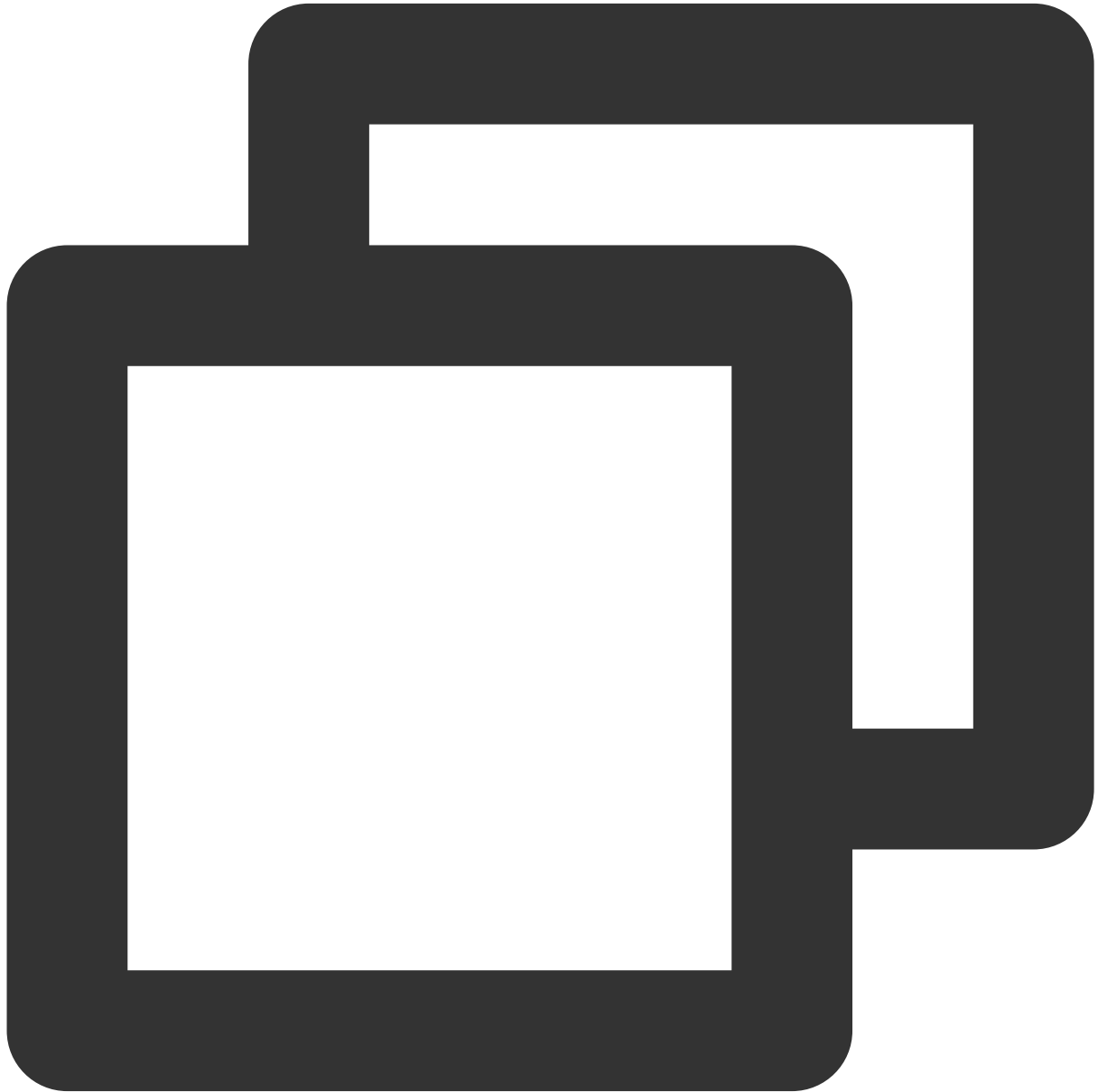
```
import 'package:rtc_room_engine/rtc_room_engine.dart';
```

```
TUIRoomEngine.setSelfInfo(userName, avatarURL);
```

Parameter	Type	Meaning
userName	String	User Name
avatarURL	String	User Avatar URL

## Start a quick conference

By calling the `quickStart` method of `ConferenceSession`, you can start a quick conference.



```
import 'package:rtc_conference_tui_kit/rtc_conference_tui_kit.dart';

ConferenceSession.newInstance('your roomId')
  ..onActionSuccess = _quickStartSuccess
  ..onActionError = _quickStartError
  ..quickStart();

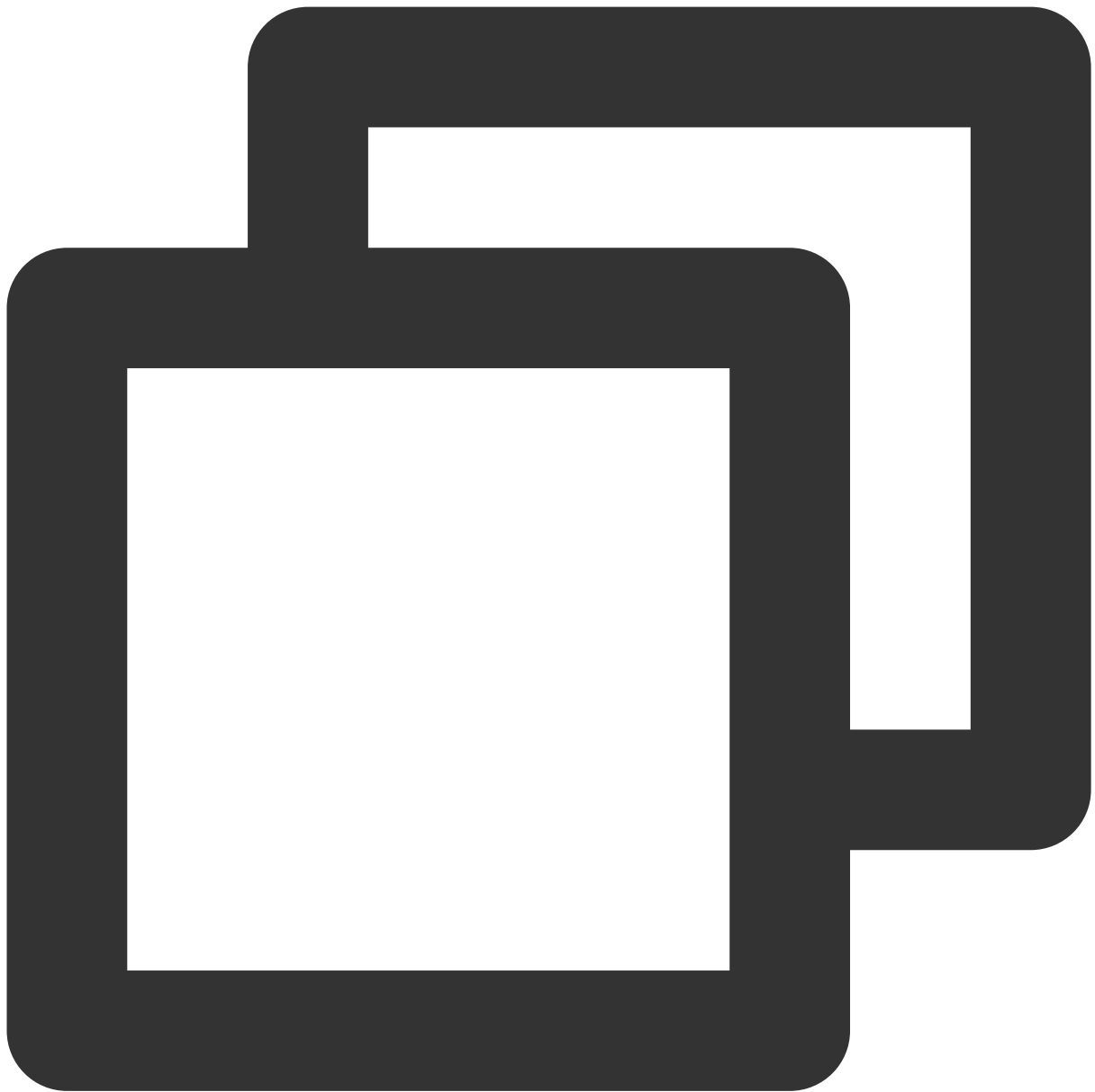
void _quickStartSuccess() {
```

```
// You can navigate to the conference page on your own in this success callback o
Navigator.push(
  context,
  MaterialPageRoute(
    builder: (context) => ConferenceMainPage(),
  ),
);
}

void _quickStartError(ConferenceError error, String message) {
  debugPrint("code: $error message: $message");
}
```

## Join a conference

By calling the `join` method of `ConferenceSession`, you can join the specified conference.



```
import 'package:rtc_conference_tui_kit/rtc_conference_tui_kit.dart';

ConferenceSession.newInstance('your roomId')
  ..onActionSuccess = _joinSuccess
  ..onActionError = _joinError
  ..join();

void _joinSuccess() {
  //You can navigate to the conference page on your own in this success callback of
  Navigator.push(
    context,
```



```
MaterialPageRoute(  
  builder: (context) => ConferenceMainPage(),  
),  
);  
}  
  
void _joinError(ConferenceError error, String message) {  
  debugPrint("code: $error message: $message");  
}
```

## More Features

TUIRoomEngine SDK Provided Rich Audio Video Room Features, Specific Content Please Refer [API Overview](#).

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [info\\_rtc@tencent.com](mailto:info_rtc@tencent.com).

# UI Customization (TUIRoomKit) Electron

Last updated : 2023-09-25 10:49:12

This article will introduce how to customize the User Interface of TUIRoomkit. We provide two options for you to choose from: Skinning Scheme and Custom UI Scheme.

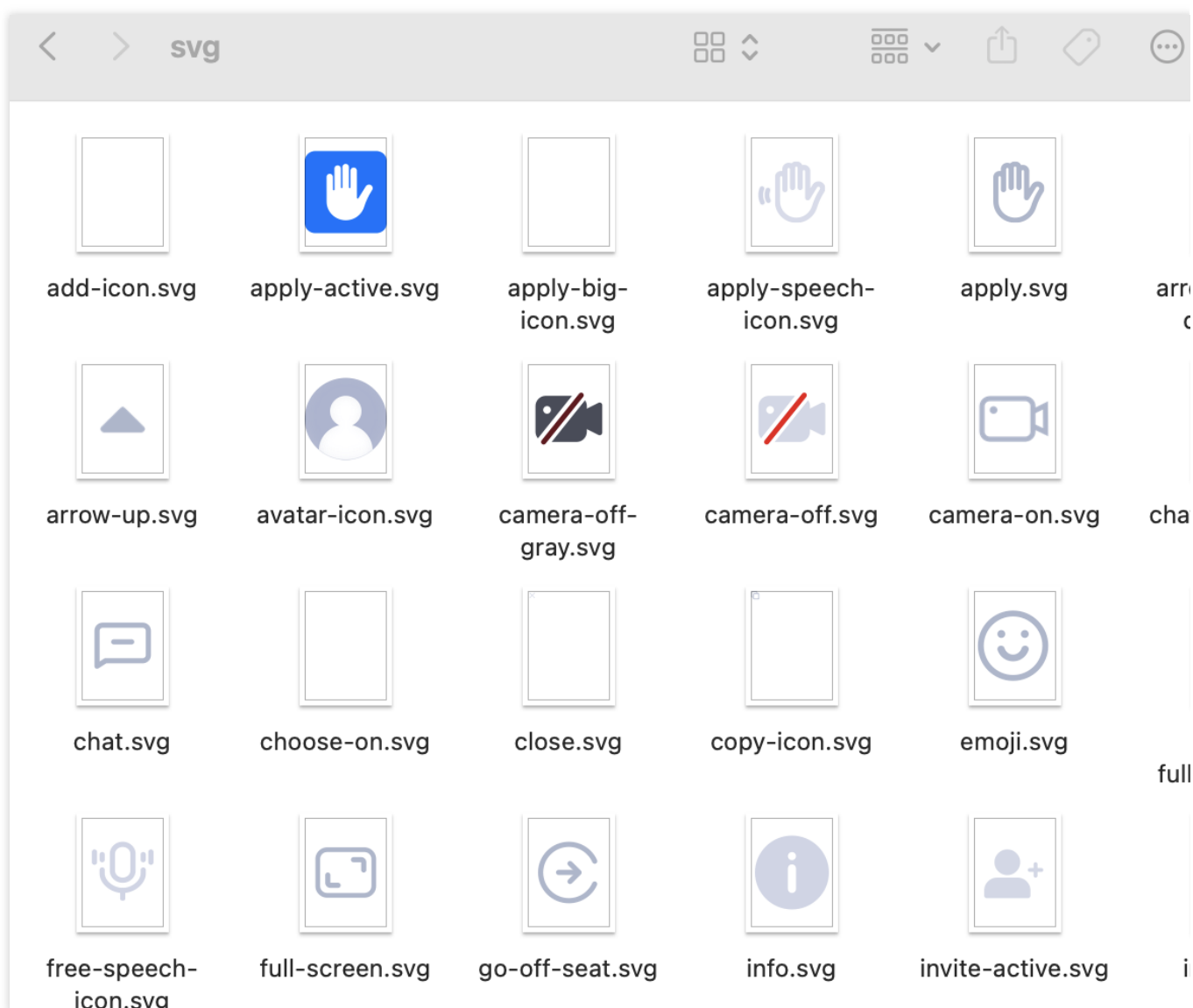
## Option 1: Interface Fine-tuning Scheme

By directly modifying the Source Code of the UI we provide, you can adjust the User Interface of TUIRoomKit. The Source Code of TUIRoomKit's interface is located in the Electron/ folder on [Github](#):

### Replace Icons

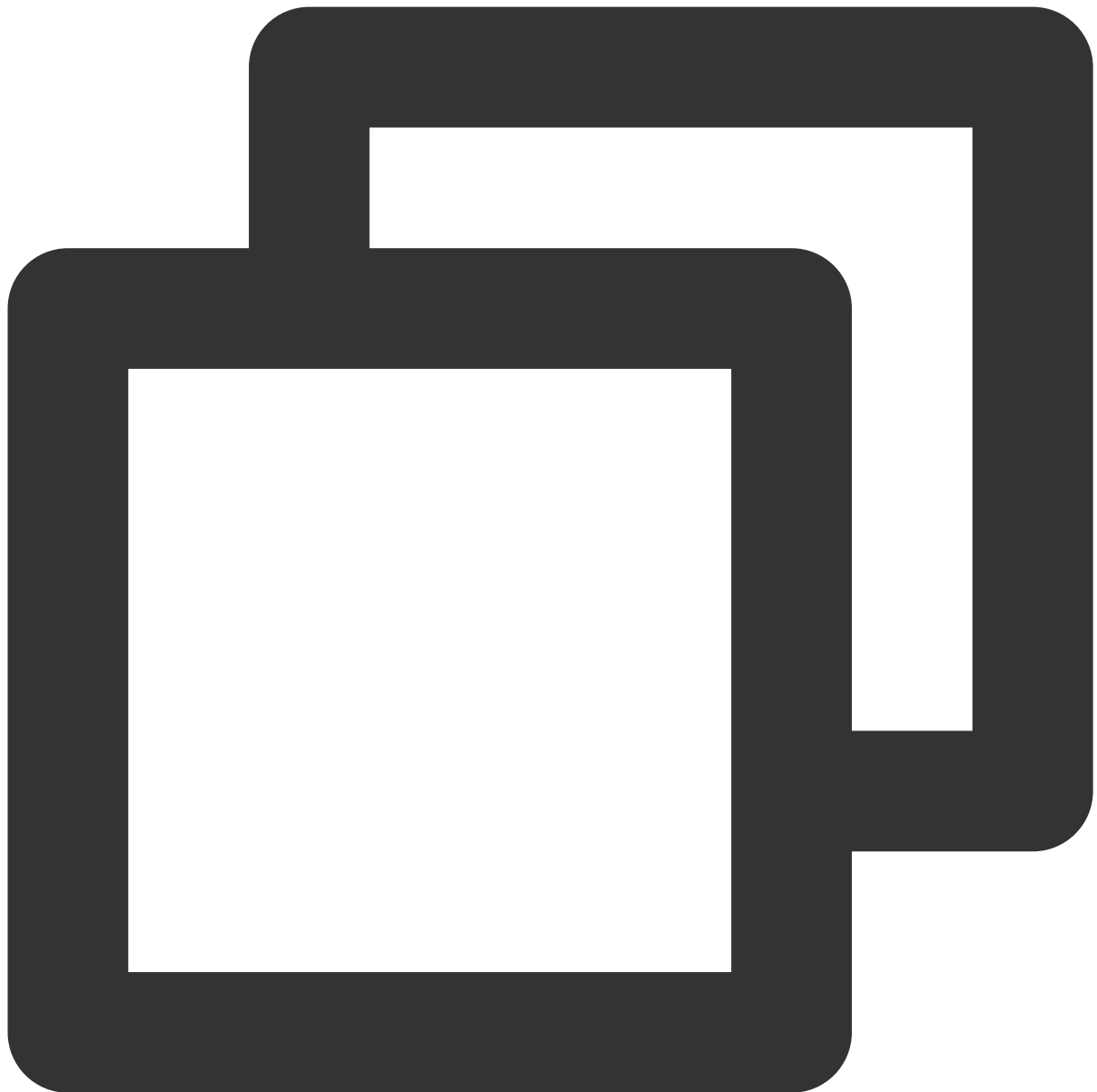
You can directly modify the icon components in the

`packages/renderer/src/TUIRoom/assets/icons/svg` Folder to ensure that the color tone style of the icons throughout the app remains consistent. Please keep the icon file names unchanged when replacing.



## Adjust UI Layout

You can adjust the UI layout of the Multi-person Video Conference interface by modifying the different pages in the `packages/renderer/src/TUIRoom/components/` folder.



```
- components/  
  - base  
  - Chat          Chat  
  - common        Common Icon Component  
  - ManageMember  Member Management  
  - RoomContent   Room Video  
  - RoomFooter    Room Page Footer  
  - RoomHeader    Room Page Header  
  - RoomHome      Home Page  
  - RoomInvite    Invite Members  
  - RoomLogin     Login Page
```

- RoomMore More
- RoomSetting Set Up
- RoomSidebar Drawer Component

## Option 2: Custom UI Scheme

The overall functionality of TUIRoom is based on the TUIRoomEngine, a UI-less SDK. You can completely implement your own User Interface based on TUIRoomEngine. For details, please see:

[TUIRoomEngine Access Guide](#)

[TUIRoomEngine API Interface Address](#)

# Android

Last updated : 2023-09-25 10:49:44

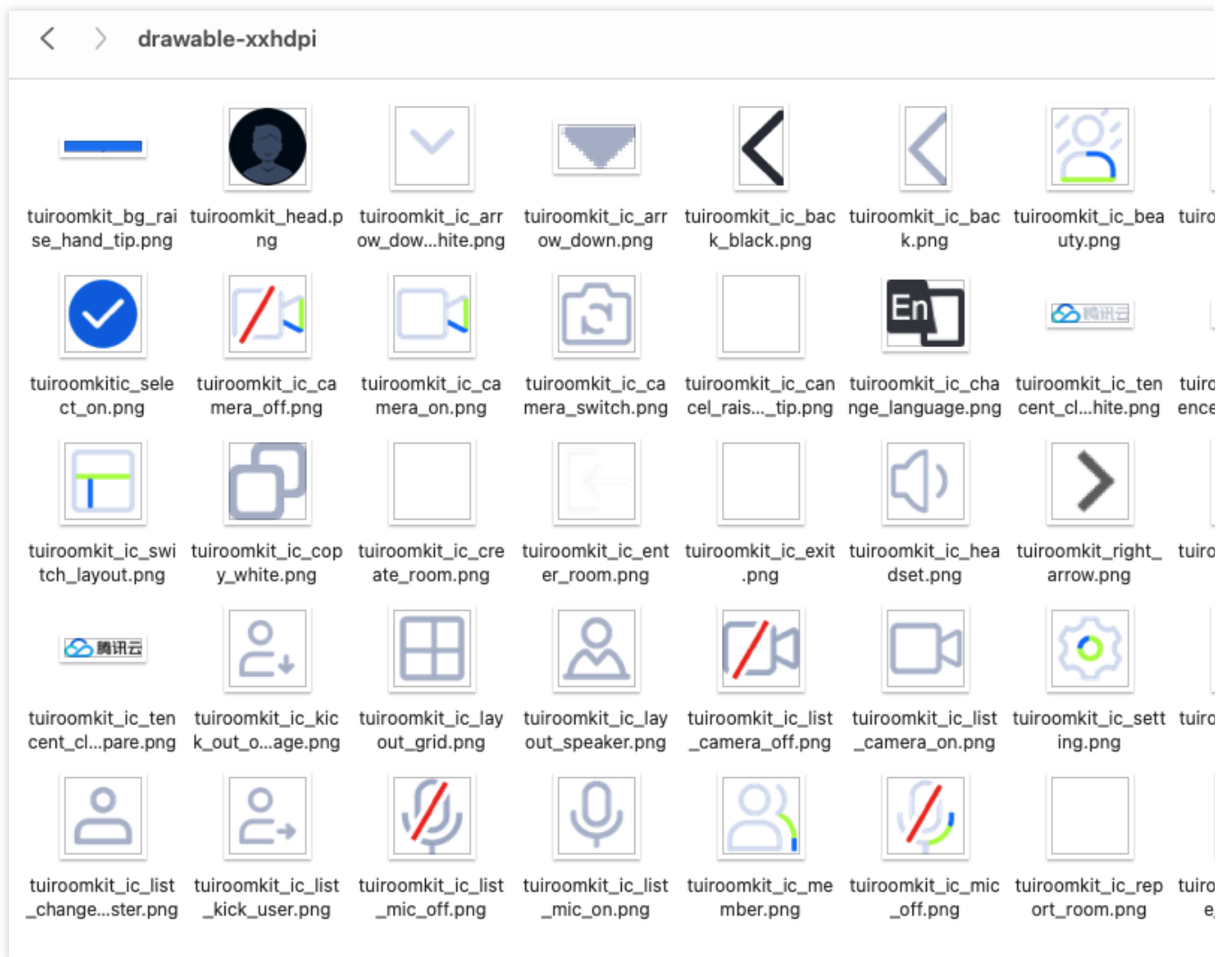
This article will introduce how to customize the User Interface of TUIRoomKit. We provide two solutions for you to choose from: **Fine-tuning Solution** and **Self-implemented UI Solution**.

## Solution 1: Fine-tuning Solution

By directly modifying the UI source code we provide, you can adjust the User Interface of TUIRoomKit. The source code of TUIRoomKit's interface is located in the `Android/tuiroomkit` folder on Github:

### Replace Icon

You can directly replace the icons in the `src/res/drawable-xxhdpi` folder to ensure that the color tone and style of the icons in the entire App remain consistent. Please keep the name of the icon file unchanged when replacing.



## Replace Copywriting

You can modify the string content of the video conference interface by modifying the `strings.xml` files in `values-zh` and `values-en`.

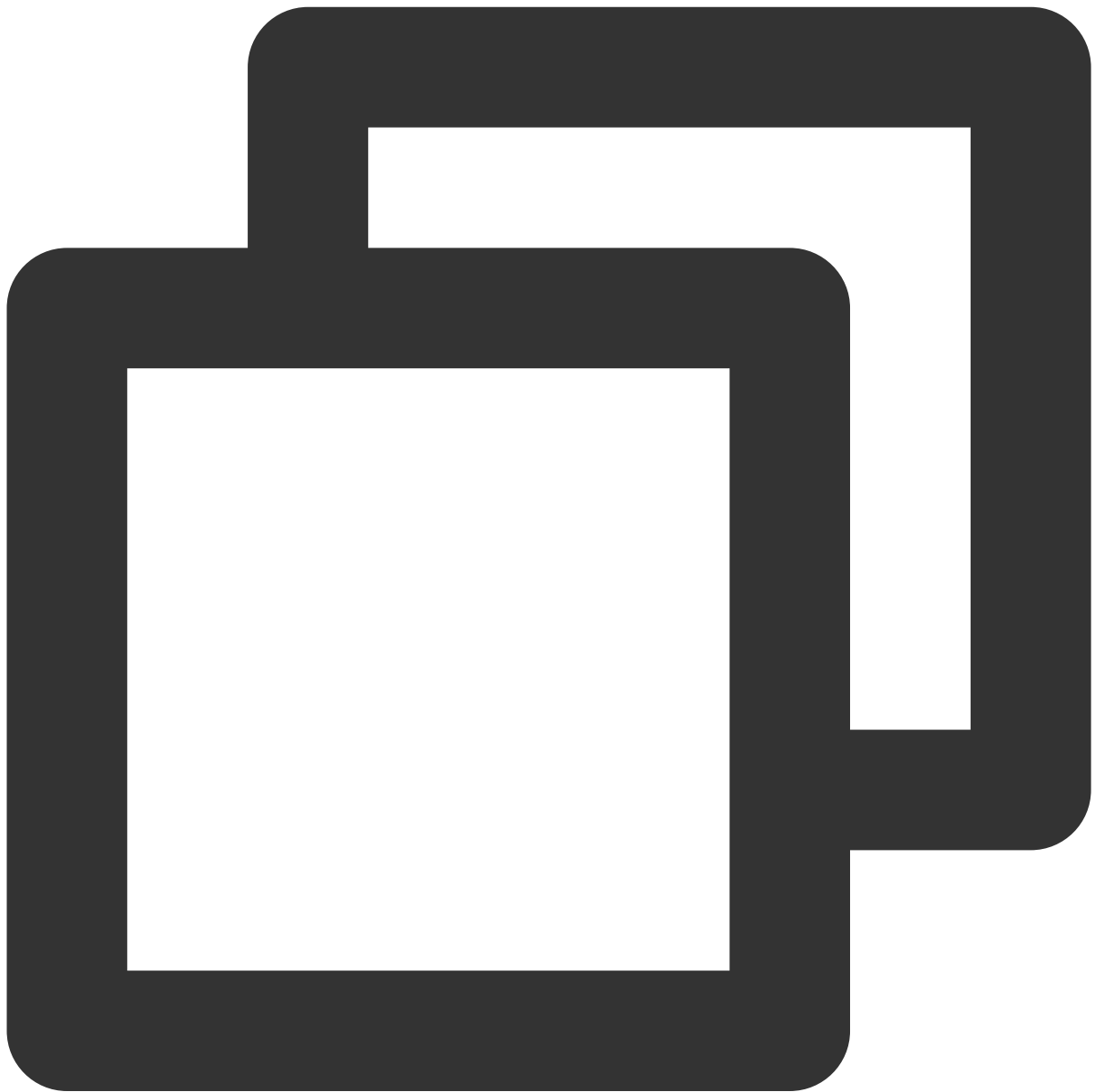
## Solution 2: Custom Partial UI Solution

The UI code of `TUIRoomKit` is located in the

`src/main/java/com/tencent/cloud/tuikit/roomkit/view` directory, and the screen view is in the

`TUIVideoSeat` Component.

The key files of `TUIRoomKit`'s View are as follows. You can change the corresponding view according to your needs and adjust your UI.



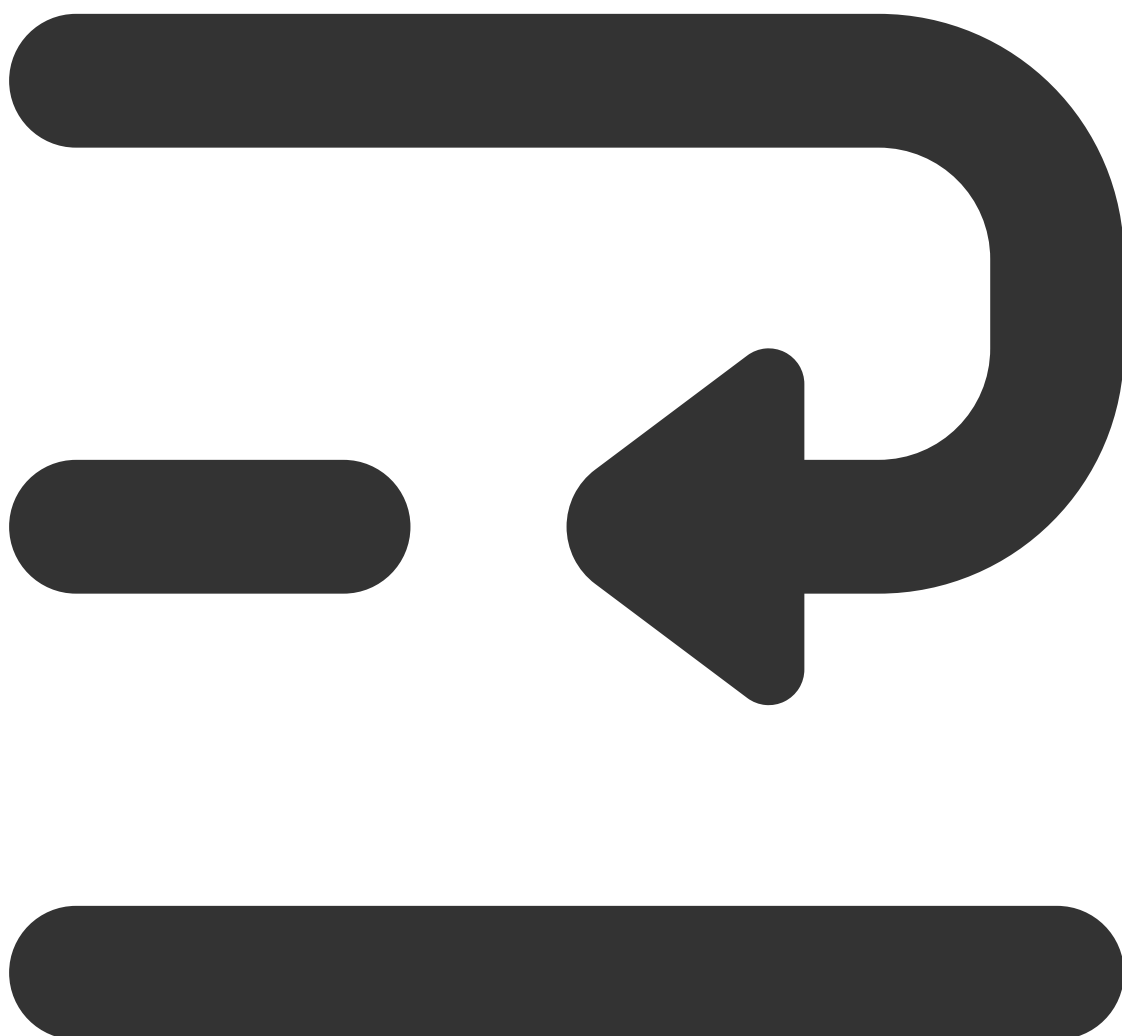
```
        |— PrepareActivity    // Preparation page Activity
        |
        |— CreateRoomActivity // Create room Activity
        |
        |— EnterRoomActivity  // Enter room Activity
        |
activity —|— RoomMainActivity // Room main page Activity
        |
view —|
        |
component—|— PrepareView      // Preparation page View
```

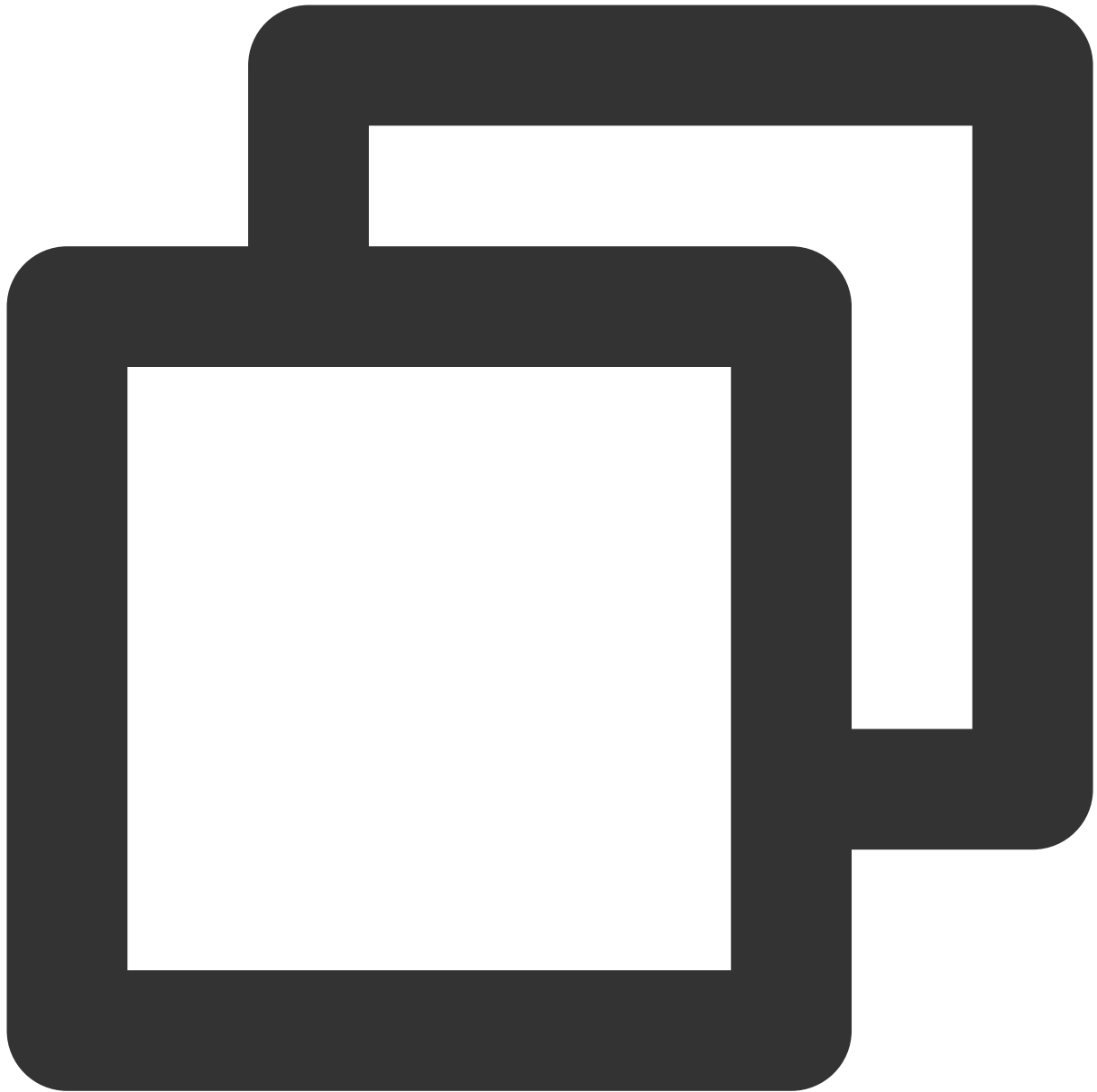


```
|
|— CreateRoomView      // Create room View
|
|— EnterRoomView       // Enter room View
|
|— RoomMainView        // Audio/video conference main page View
|
|— TopView             // Top button view, including: Speaker/Receiv
|
|— BottomView          // Bottom button view, including: Camera, Mic
|
|— UseListView         // User List view
|
|— RaiseHandApplicationListView // Raise Hand to Speak mode, R
|
|— TransferMasterView // Room Owner Transfer page
|
|— MoreFunctionView    // "More" function view, including Chat, Beau
|
|— MeetingActivity     // Audio/video conference main activity
```

## Modification of BottomView's Bottom Button

To make it easier for you to customize the bottom function buttons, our BottomView is automatically constructed by reading the list. Taking the video switch button as an example, the code is as follows.





```
private BottomItemData createCameraItem() {
    BottomItemData cameraItemData = new BottomItemData();
    //Set button type to differentiate different buttons
    cameraItemData.setType(BottomItemData.Type.VIDEO);
    //Set whether the button is clickable
    if (isOwner()) {
        cameraItemData.setEnable(true);
    } else if (mRoomStore.roomInfo.enableSeatControl) {
        cameraItemData.setEnable(false);
    } else {
        cameraItemData.setEnable(mRoomStore.roomInfo.enableVideo);
    }
}
```

```

}
//Set the default icon of the button
cameraItemData.setIconId(R.drawable.tuiroomkit_ic_camera_off);
//Set the background image of the button
cameraItemData.setBackground(R.drawable.tuiroomkit_bg_bottom_item_black);
//Set the icon of the button when it is not clickable
cameraItemData.setDisableIconId(R.drawable.tuiroomkit_ic_camera_off);
//Set the default icon of the button
cameraItemData.setName(mContext.getString(R.string.tuiroomkit_item_open_camera))

//Button click effect, if your button needs to switch images/names, etc. when cl
BottomSelectItemData camaraSelectItemData = new BottomSelectItemData();
//Set the name of the button when selected
camaraSelectItemData.setSelectedName(mContext.getString(R.string.tuiroomkit_item
//Set the name of the button when not selected
camaraSelectItemData.setUnSelectedName(mContext.getString(R.string.tuiroomkit_it
//Set whether the button is selected
camaraSelectItemData.setSelected(false);
//Set the icon of the button when selected
camaraSelectItemData.setSelectedIconId(R.drawable.tuiroomkit_ic_camera_on);
//Set the icon of the button when not selected
camaraSelectItemData.setUnSelectedIconId(R.drawable.tuiroomkit_ic_camera_off);
//Set the click event of the button when selected/unselected
camaraSelectItemData.setOnItemSelectedListener(new BottomSelectItemData.OnItemSele
    @Override
    public void onItemSelected(boolean isSelected) {
        enableCamera(isSelected);
    }
});
cameraItemData.setSelectItemData(camaraSelectItemData);
return cameraItemData;
}

```

## Solution 3: Custom All UI Solution

The overall function of TUIRoomKit is based on the TUIRoomEngine, a UI-less SDK. You can completely implement your own UI interface based on TUIRoomEngine. For details, please refer to the TUIRoomEngine API interface address.

# iOS

Last updated : 2024-03-15 16:06:53

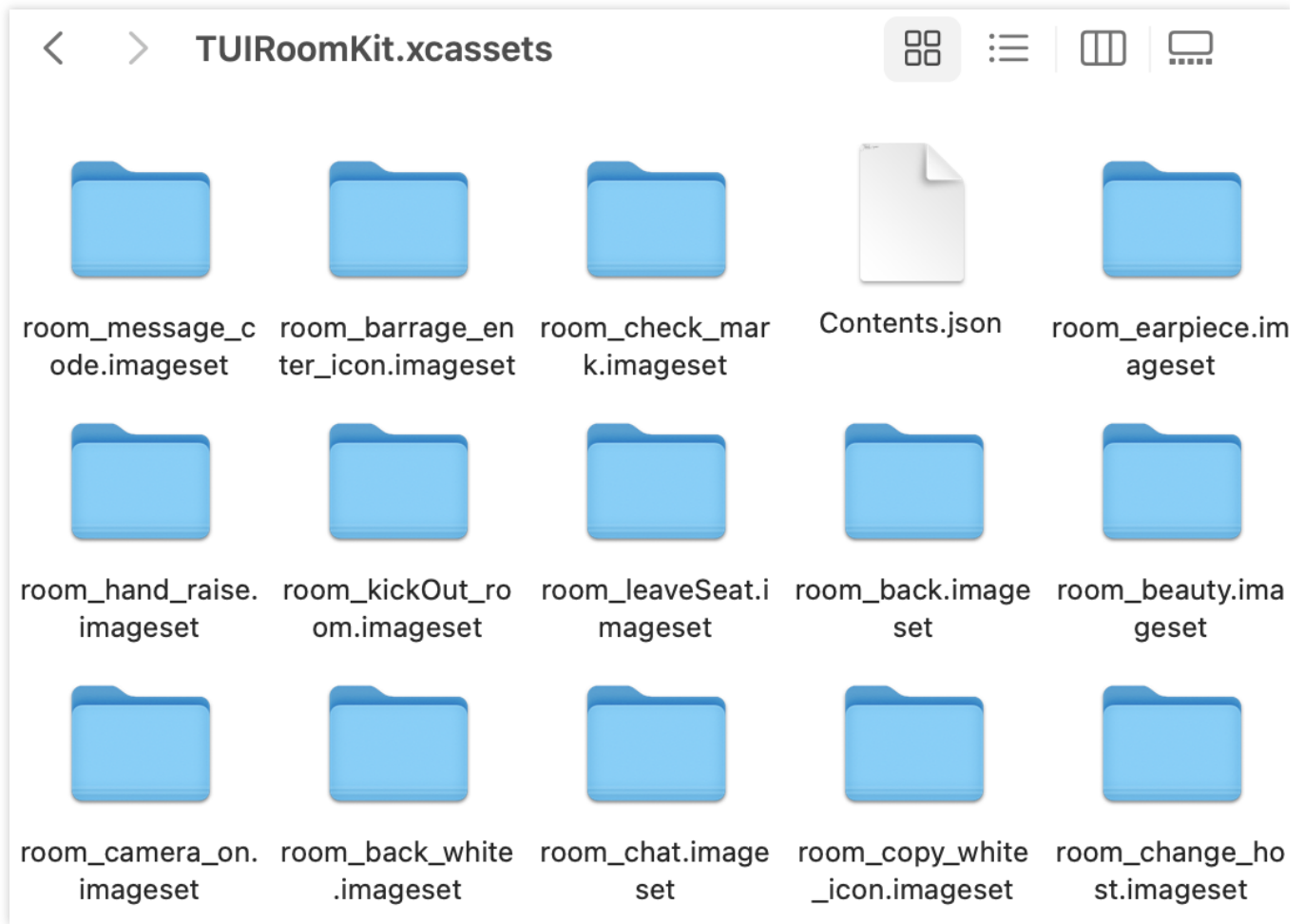
This article will introduce how to customize the user interface of TUIRoomKit. We provide two solutions for you to choose from: **fine-tuning solution** and **custom UI solution**.

## Solution 1: Fine-tuning solution

By directly modifying the UI source code we provide, you can adjust the user interface of TUIRoomKit. The interface source code of TUIRoomKit is located in the iOS/ folder on Github:

### Replace icons

You can directly modify the icon components in the `TUIRoomKit/Resources/TUIRoomKit.xcassets` folder to ensure that the icon color tone style is consistent throughout the app. Please keep the icon file name unchanged when replacing.

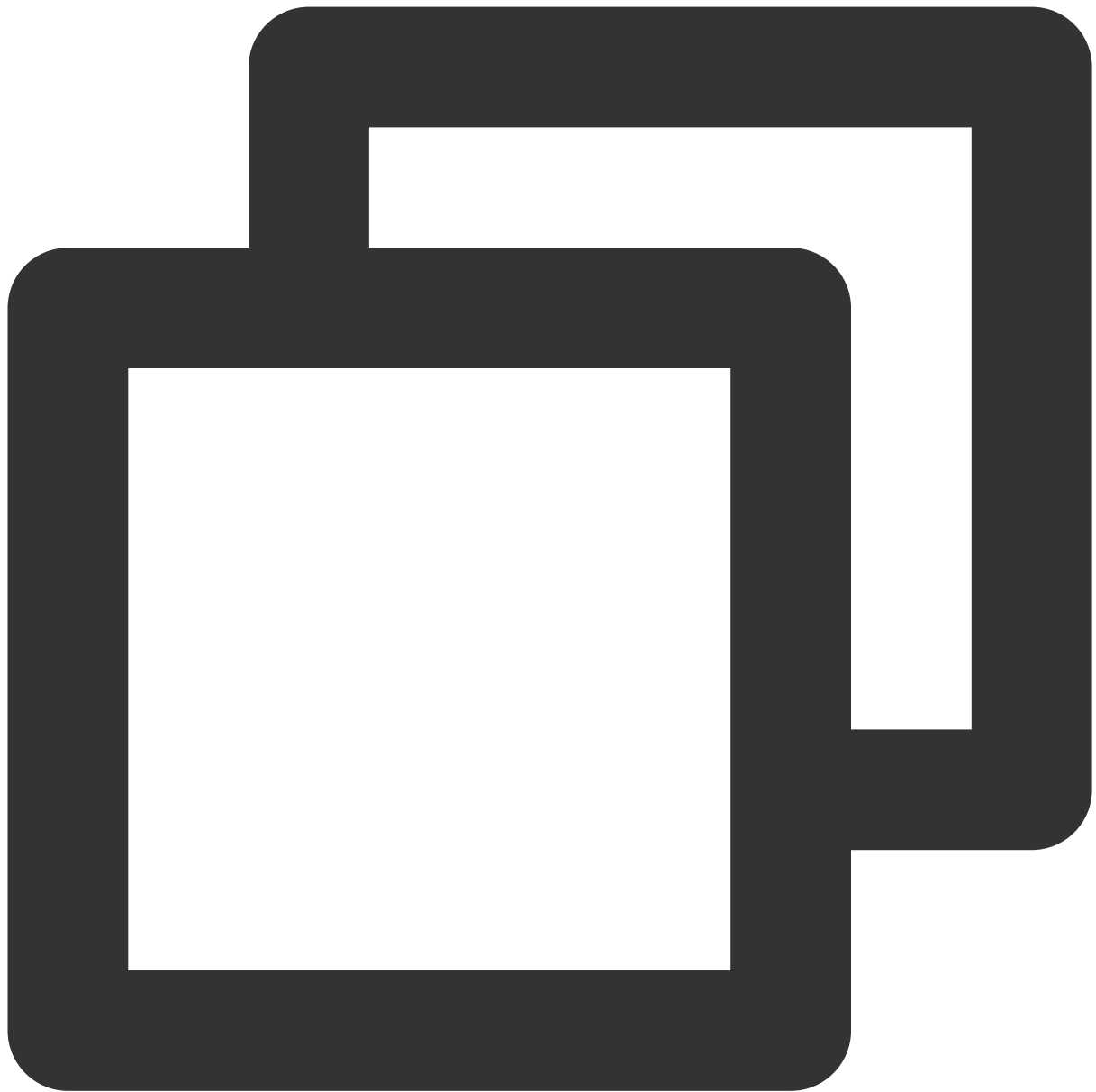


## Replace copywriting

You can modify the `strings.xml` files in the `values-zh` and `values-en` of the `TUIRoomKit/Resources/Localized` folder to change the string content of the video conference interface.

## Adjust UI layout

You can adjust the UI layout of the multi-person video conference interface by modifying the different pages under the `TUIRoomKit/Source/View` file:



```
|— Component
|   |— ButtonItemView.swift           //Universal view
|   |— ListCellItemView.swift        //Universal view
|— Page
|   |— RoomMainView.swift             //Room main view
|   |— RoomMainViewController.swift
|   |— RoomRouter.swift               //Routing
|   |— Widget
|       |— BottomNavigationBar         //Bottom bar
|       |   |— BottomItemView.swift
|       |   |— BottomView.swift
```

```

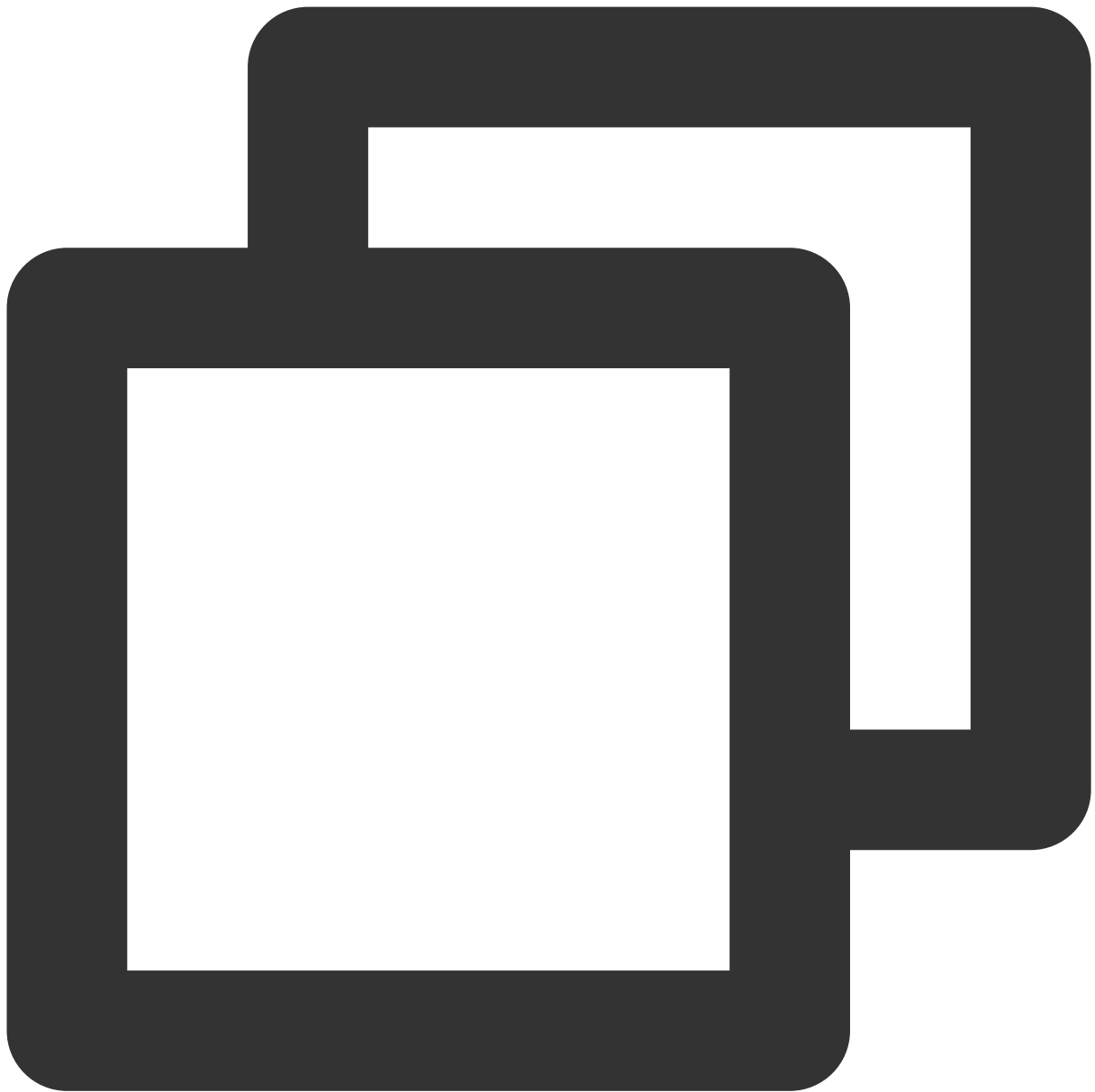
├─ FloatWindow                                //Floating window
|   ├─ RoomUserStatusView.swift
|   └─ RoomVideoFloatView.swift
├─ MediaSettings                             //Set up
|   ├─ MediaSettingView.swift
|   └─ VideoChoicePanel.swift
├─ TopNavigationBar                         //Top bar
|   ├─ TopItemView.swift
|   └─ TopView.swift
├─ UserControlPanel                         //User list
|   ├─ UserListManagerView.swift
|   └─ UserListView.swift
├─ RaiseHandControlPanel                   //Raise hand user management lis
|   └─ RaiseHandApplicationListView.swift
├─ TransferOwnerControlPanel               //Room transfer list
|   └─ TransferMasterView.swift
├─ PopUpControlPanel                       //Universal pop-up box
|   ├─ PopUpView.swift
|   └─ PopUpViewController.swift
└─ VideoSeat                              //Video display
    ├─ ScreenCaptureMaskView.swift
    ├─ TUIVideoSeatCell.swift
    ├─ TUIVideoSeatLayout.swift
    ├─ TUIVideoSeatUserStatusView.swift
    └─ TUIVideoSeatView.swift

```

## Modification of BottomView at the bottom

To make it easier for you to customize the function buttons at the bottom, our BottomView is automatically constructed by reading the list. Taking the video switch button as an example, the code is as follows:





```
func createBottomData() {  
    let muteVideoItem = ButtonItemData()  
    //Set the default button title  
    muteVideoItem.normalTitle = .unMuteVideoText  
    //Set the button title after clicking  
    muteVideoItem.selectedTitle = .muteVideoText  
    //Set the default button icon  
    muteVideoItem.normalIcon = "room_camera_on"  
    //Set the button icon after clicking  
    muteVideoItem.selectedIcon = "room_camera_off"
```

```
//Set the button image resource acquisition location
muteVideoItem.resourceBundle = tuiRoomKitBundle()
//Set whether the button is clicked
muteVideoItem.isSelect = !(roomInfo.isOpenCamera)
//Set the button type to differentiate different buttons
muteVideoItem.buttonType = .muteVideoItemType
//Set the button click event
muteVideoItem.action = { [weak self] sender in
    guard let self = self, let button = sender as? UIButton else { return }
    self.muteVideoAction(sender: button)
}
```

## Solution 2: Custom UI solution

The overall function of TUIRoomKit is based on the TUIRoomEngine, a UI-less SDK. You can completely implement your own UI interface based on TUIRoomEngine. For details, please see [TUIRoomEngine API](#) interface address.

# Windows

Last updated : 2023-09-25 10:50:35

This article will introduce how to customize the user interface of TUIRoomKit. We provide two options for you to choose from: **the skin replacement scheme** and **the custom UI scheme**.

## Option 1: Interface Fine-tuning Scheme

You can directly modify and adapt based on the App we provide

The App directory contains the design and logic related to the UI. You can modify the RoomApp source code for secondary development according to your needs. The main features are as follows:

Feature	File Directory
Home Login	Windows-Mac\\RoomApp\\App\\LoginViewController.cpp
Device Detection	Windows-Mac\\RoomApp\\App\\PresetDeviceController.cpp
Main Page	Windows-Mac\\RoomApp\\App\\MainWindow.cpp
Mic-on List	Windows-Mac\\RoomApp\\App\\StageListController.cpp
Member List	Windows-Mac\\RoomApp\\App\\MemberListViewController.cpp
Settings Page	Windows-Mac\\RoomApp\\App\\SettingViewController.cpp
Chat Room	Windows-Mac\\RoomApp\\App\\ChatRoomViewController.cpp
Screen Sharing	Windows-Mac\\RoomApp\\App\\ScreenShareWindow.cpp
Bottom Toolbar	Windows-Mac\\RoomApp\\App\\BottomBarController.cpp

## Option 2: Custom UI Scheme

The overall function of TUIRoomKit is based on the TUIRoomEngine, a UI-less SDK. You can completely implement your own UI interface based on TUIRoomEngine. For details, please refer to the TUIRoomEngine API interface address.

# Web

Last updated : 2023-09-25 10:51:00

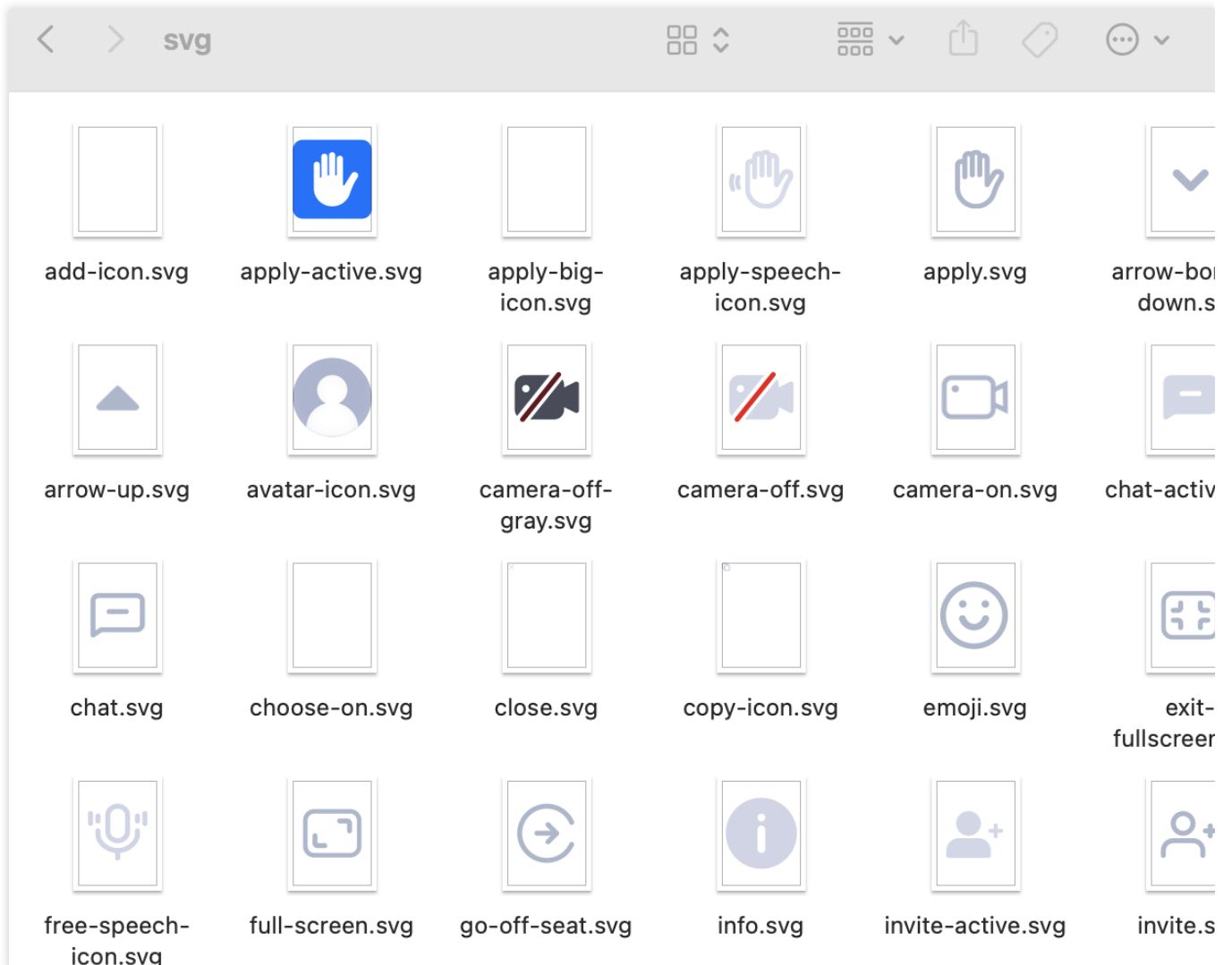
This article will introduce how to customize the User Interface of TUIRoomkit. We provide two options for you to choose from: Skinning Scheme and Custom UI Scheme.

## Option 1: Interface Fine-tuning Scheme

By directly modifying the Source Code of the UI we provide, you can adjust the User Interface of TUIRoomKit. The Source Code of TUIRoomKit's interface is located in the Electron/ folder on [Github](#).

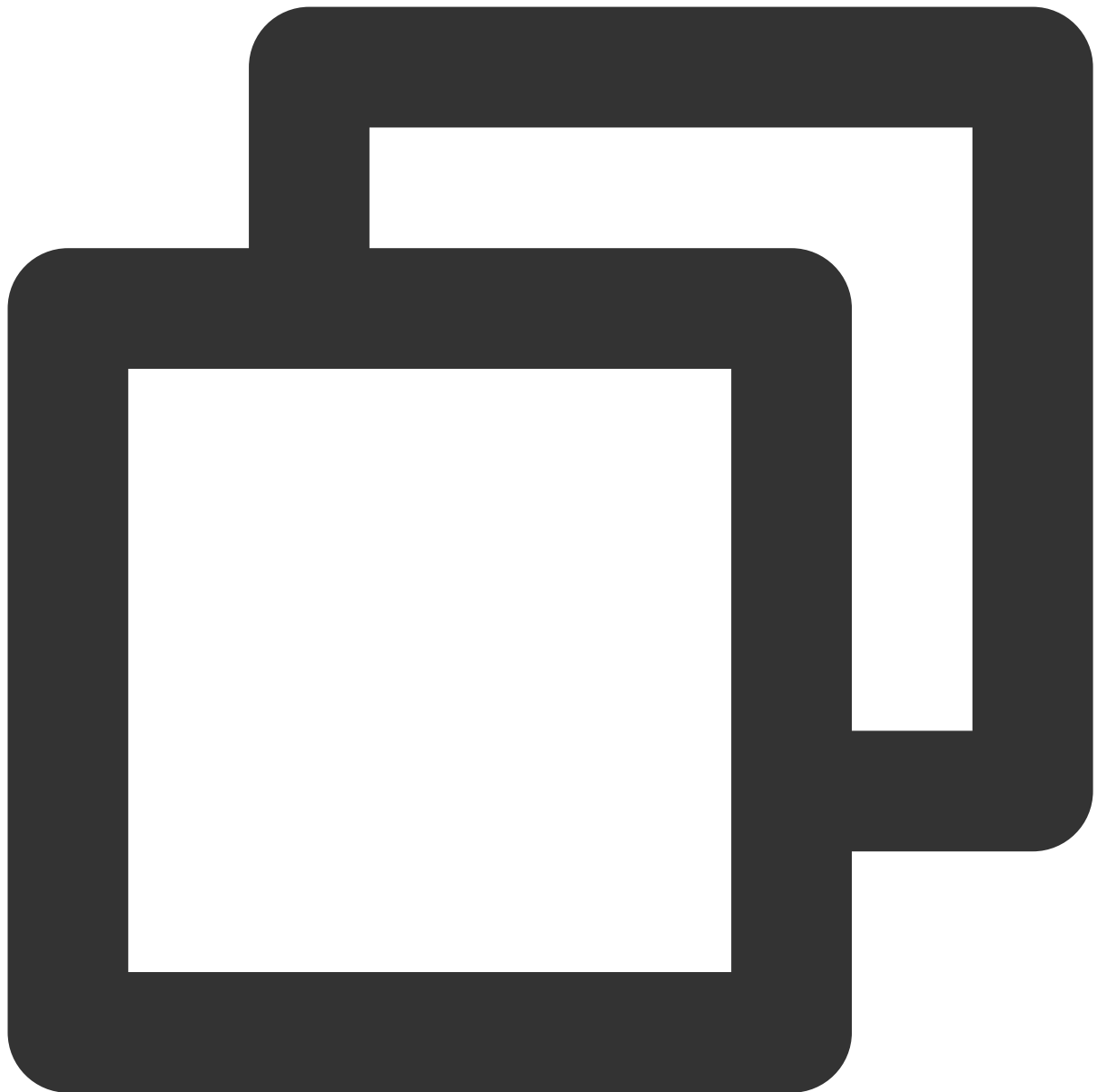
### Replace Icons

You can directly modify the icon components in the `web/vue3/src/TUIRoom/assests/icons/svg` Folder to ensure that the color tone style of the icons throughout the app remains consistent. Please keep the icon file names unchanged when replacing.



## Adjust UI Layout

You can adjust the UI layout of the Multi-person Video Conference interface by modifying the different pages in the `src/components/` folder.



```
- components/  
  - base  
  - Chat          Chat  
  - common        Common Icon Component  
  - ManageMember  Member Management  
  - RoomContent   Room Video  
  - RoomFooter    Room Page Footer  
  - RoomHeader    Room Page Header  
  - RoomHome      Home Page  
  - RoomInvite    Invite Members  
  - RoomLogin     Login Page
```

- RoomMore More
- RoomSetting Set Up
- RoomSidebar Drawer Component

## Option 2: Custom UI Scheme

The overall functionality of TUIRoom is based on the TUIRoomEngine, a UI-less SDK. You can completely implement your own User Interface based on TUIRoomEngine. For details, please see:

[TUIRoomEngine Access Guide](#)

[TUIRoomEngine API Interface Address](#)

# Flutter

Last updated : 2024-04-12 11:22:08

This article will introduce how to customize the user interface of `TUIRoomKit` . We provide two solutions for you to choose from: **fine-tuning solution** and **custom UI solution**.

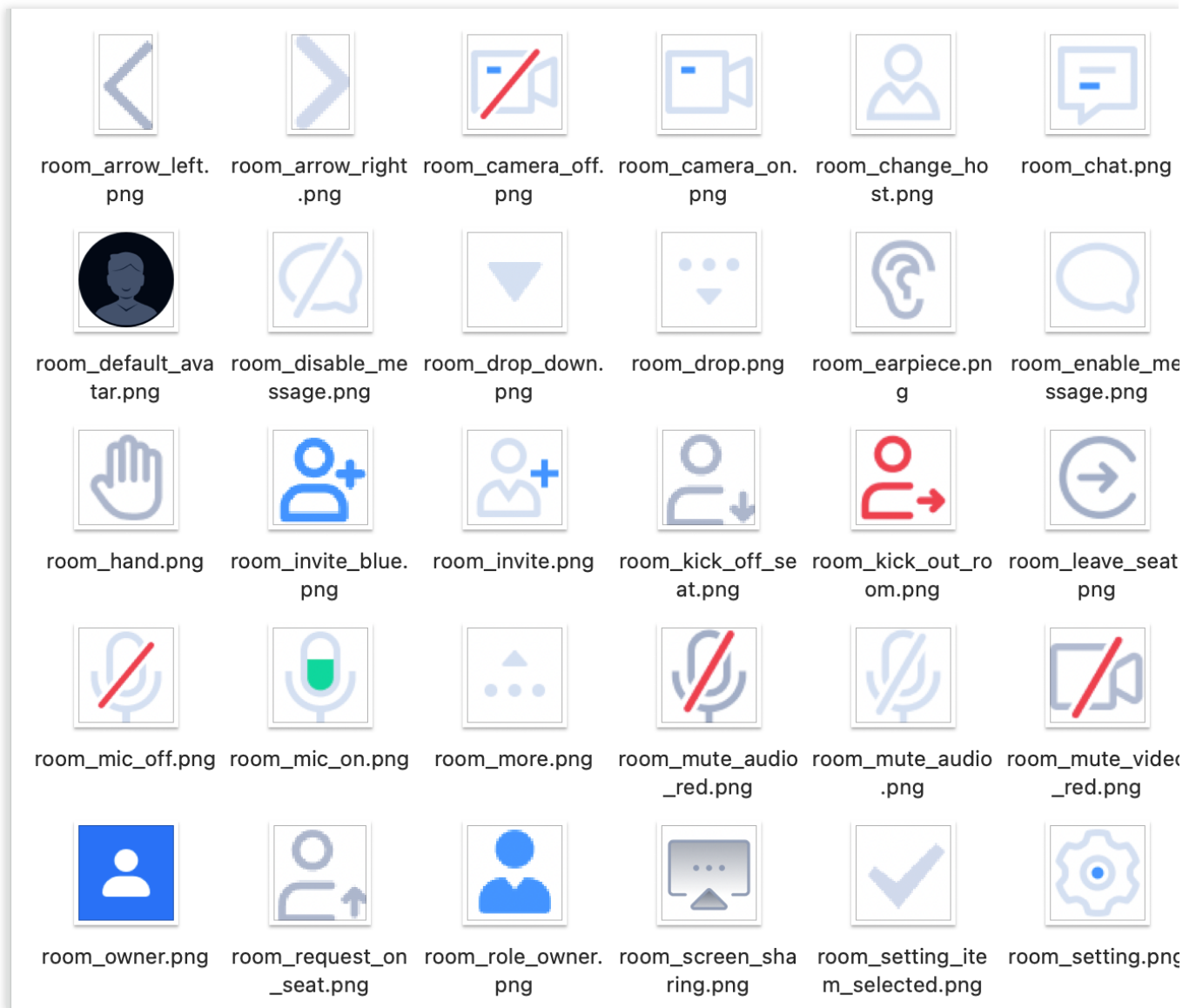
## Solution 1: Fine-tuning solution

By directly modifying the UI source code we provide, you can adjust the user interface of `TUIRoomKit` .

### Replace icons

You can directly modify the icon components in the `rtc_conference_tui_kit/assets/images` folder to ensure that the icon color tone style is consistent throughout the app. Please keep the icon file name unchanged when replacing.



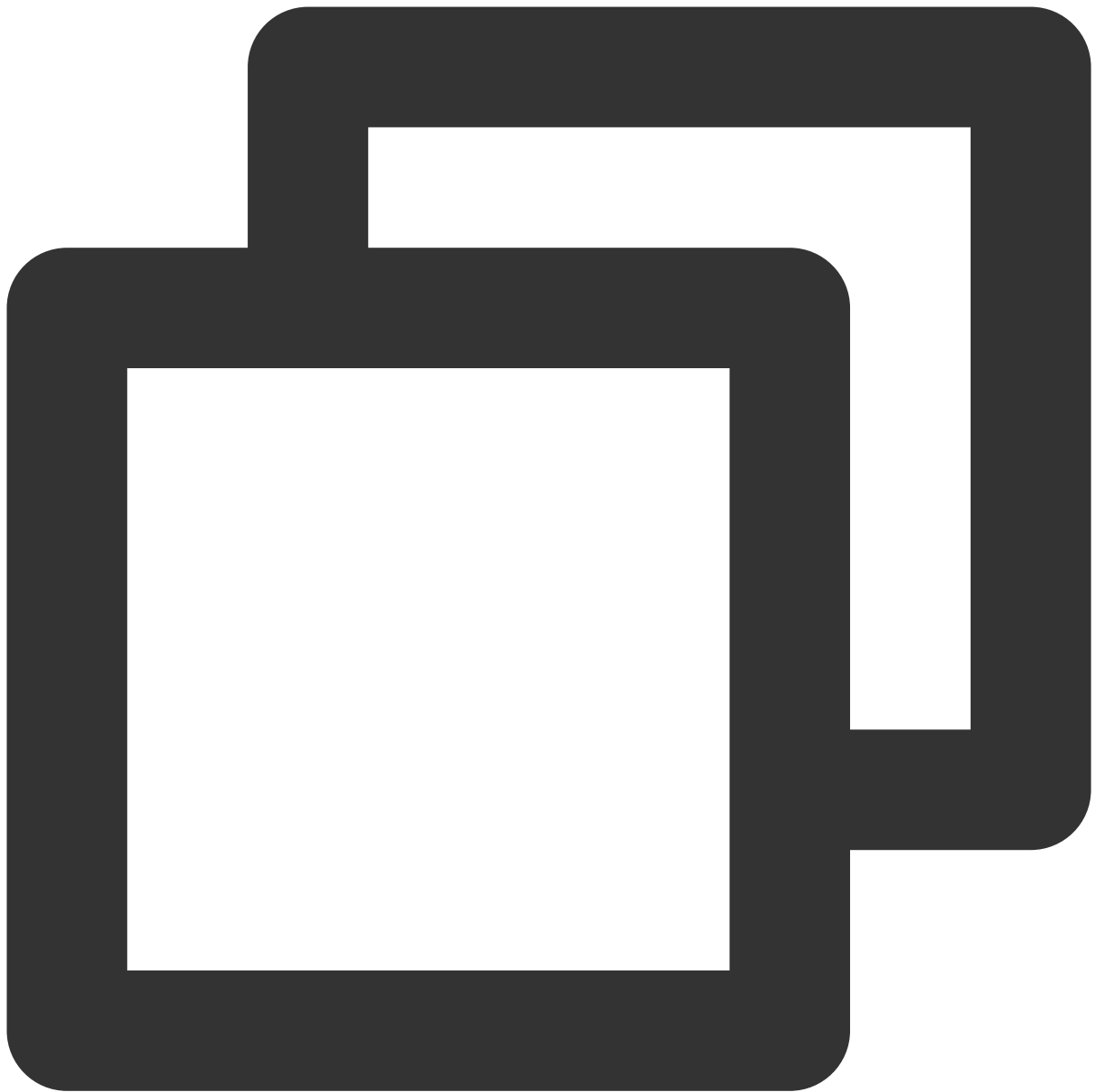


## Replace copywriting

You can modify the `en_us.dart` and `zh_cn.dart` files in the `rtc_conference_tui_kit/lib/common/languages` folder to change the string content of the video conference interface.

## Solution 2: Custom part of UI solution

The UI component folders of `TUIRoomKit` are as follows. You can modify the `view.dart` in the corresponding folder to customize the UI according to your actual business needs.



```
├─ lib
├─ common
│   └─ widgets
│       ├── bottom_sheet.dart (1 KB)           // Universal bottom sheet
│       ├── copy_text_button.dart (1 KB)       // Universal copy button
│       ├── dialog.dart (4 KB)                 // Universal dialog
│       ├── drop_down_button.dart (<1 KB)      // Universal dropdown butto
│       ├── info_list.dart (1 KB)              // Universal information di
│       ├── rounded_container.dart (<1 KB)     // Universal rounded corner
│       ├── search_bar.dart (1 KB)             // Universal search bar
│       └── single_select_list.dart (1 KB)     // Universal single select
```

```

|       |— slider.dart (1 KB) // Universal slider
|       |— switch.dart (<1 KB) // Universal switcher
|       |— toast.dart (<1 KB) // Universal toast
|       |— user_info.dart (3 KB) // Universal user info item
|       |— volume_bar.dart (2 KB) // Universal dynamic microp
|— pages
|   |— conference_main
|       |— view.dart (3 KB)
|       |— widgets
|           |— bottom_view
|               |— view.dart (1 KB) // Function button of bott
|               |— widgets
|                   |— base_button.dart (10 KB) // Unexpanded basic bottom
|                   |— bottom_button_item.dart (2 KB) // Encapsulated universal
|                   |— mic_button.dart (1 KB) // Microphone button that
|                   |— more_button.dart (3 KB) // All bottom function but
|           |— exit
|               |— view.dart (2 KB) // Bottom sheet of exit ro
|           |— invite_sheet
|               |— invite_sheet.dart (1 KB) // Invite popup
|           |— local_screen_sharing
|               |— local_screen_sharing.dart (1 KB) // Prompt component during
|           |— raise_hand_list
|               |— view.dart (3 KB) // Raise hand application
|               |— widgets
|                   |— title.dart (<1 KB) // List title bar
|                   |— user_item.dart (1 KB) // A single item of the en
|                   |— user_table.dart (1 KB) // List component
|           |— setting
|               |— view.dart (1 KB) // Settings panel
|               |— widgets
|                   |— audio_setting.dart (1 KB) // Audio setting component
|                   |— setting_info_select.dart (1 KB) // Radio value setting com
|                   |— setting_item.dart (<1 KB) // Settings panel single s
|                   |— video_setting.dart (4 KB) // Video setting component
|           |— top_view
|               |— view.dart (3 KB) // Function button of top
|               |— widgets
|                   |— meeting_title.dart (2 KB) // Top meeting information
|                   |— room_info_sheet.dart (2 KB) // Detailed meeting inform
|                   |— top_button_item.dart (1 KB) // Top function single uni
|           |— transfer_host
|               |— view.dart (1 KB) // Transfer homeowners pan
|               |— widgets
|                   |— title.dart (<1 KB) // Title bar of transfer h
|                   |— user_item.dart (1 KB) // Transferable homeowner
|                   |— user_table.dart (1 KB) // Transferable homeowner

```

```

|      |─ user_list
|      |   |─ view.dart (3 KB)                // User list Page
|      |   └─ widgets
|      |       |─ button_item.dart (1 KB)      // Button item at the bott
|      |       |─ user_control.dart (7 KB)     // The member management p
|      |       |─ user_control_item.dart (1 KB) // The item of member mana
|      |       |─ user_item.dart (3 KB)        // User list item
|      |       └─ user_table.dart (3 KB)       // User list
|      |─ video_seat
|      |   |─ video_layout
|      |   |   |─ view.dart (3 KB)            // The layout of video ite
|      |   |   └─ widgets
|      |   |       |─ video_item              // Single video frame item
|      |   |       |   |─ view.dart (4 KB)    // Single video frame item
|      |   |       |   └─ widgets
|      |   |       |       |─ video_user_info.dart (2 KB)    // User inform
|      |   |       |       └─ volume_bar.dart (<1 KB)        // Microphone
|      |   |       └─ with_draggable_window_widget.dart (1 KB) // Two-person
|      |   └─ video_page_turning
|      |       |─ view.dart (2 KB)            // Video scree
|      |       └─ widgets
|      |           └─ page_indicator.dart (1 KB) // Page indica

```

## Solution 3: Custom all UI solution

The overall function of `TUIRoomKit` is based on the `TUIRoomEngine`, a UI-less SDK. You can completely implement your own UI interface based on `TUIRoomEngine`. For details, please see [TUIRoomEngine API interface address](#).

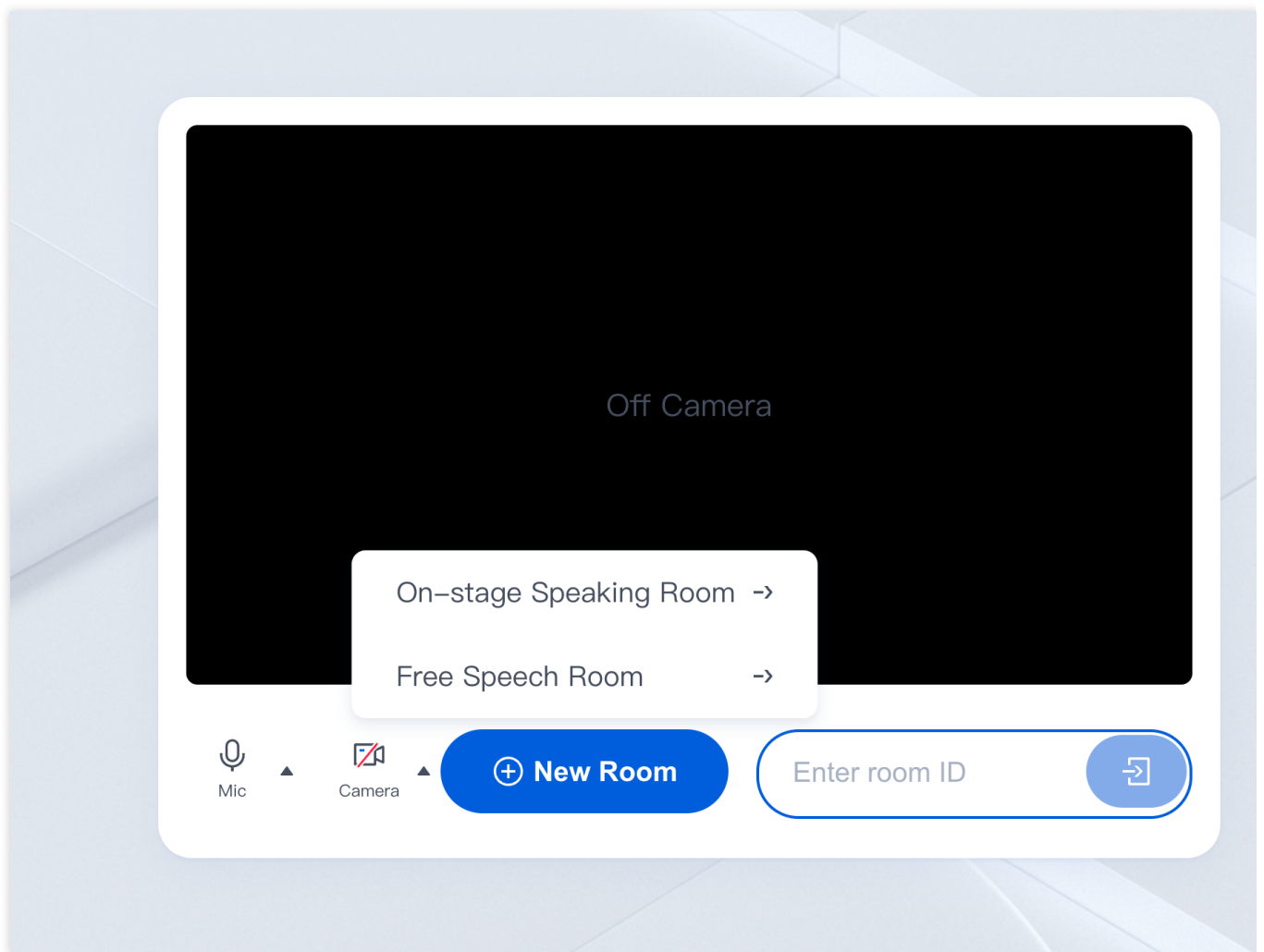
# Meeting Control (TUIRoomKit)

Last updated : 2024-04-12 11:20:53

This document will provide a detailed introduction to the pre-conference control, in-conference control, and other aspects of `TUIRoomKit` to help you better master the conference control features of `TUIRoomKit`. Through this document, you will be able to fully utilize the functions of `TUIRoomKit` to achieve high-quality audio and video conferences.

## Pre-conference Control

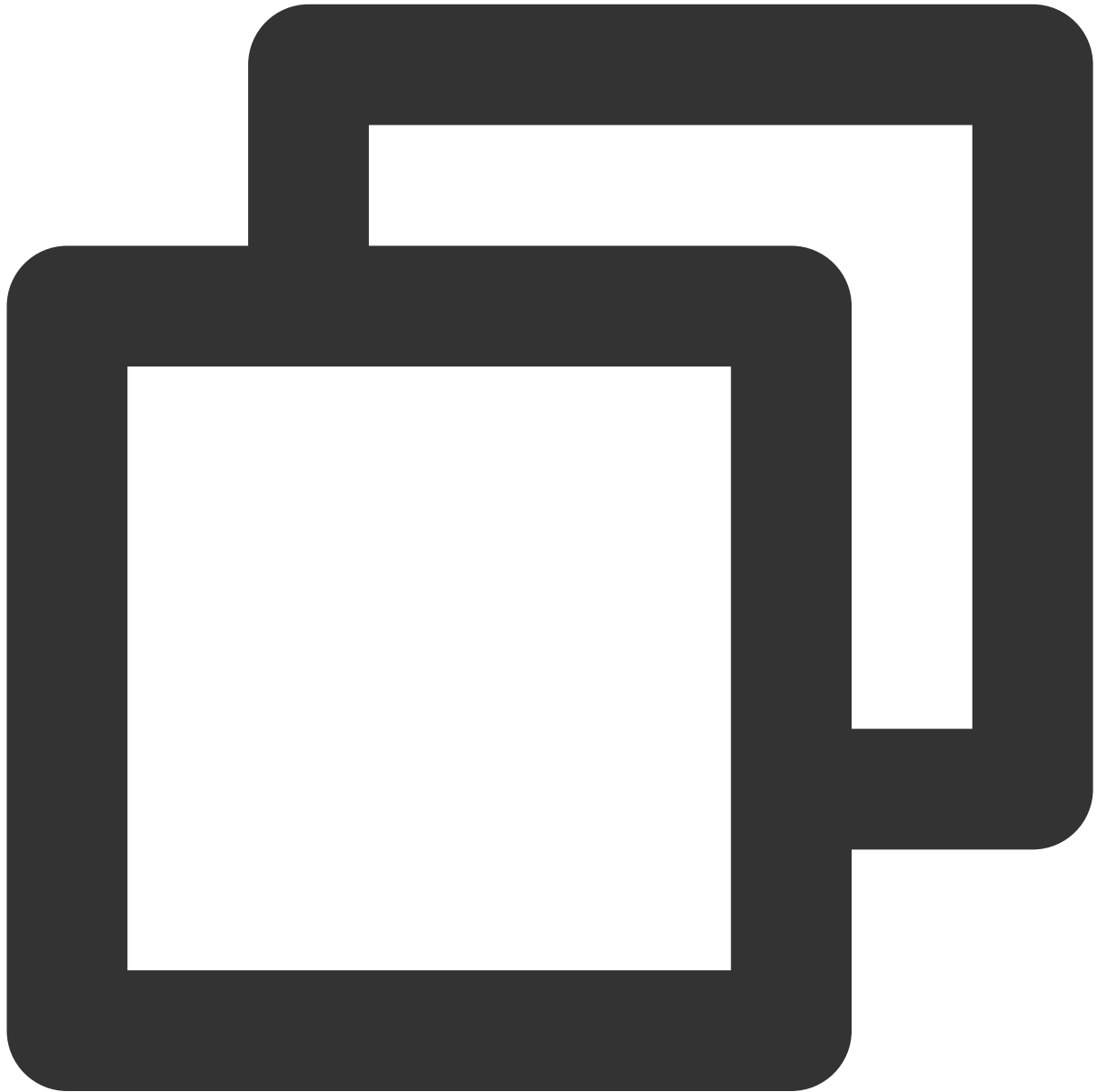
Before entering the conference, you can use the pre-conference control features of `TUIRoomKit` to set the relevant parameters of the conference in advance, ensuring the smooth progress of the conference.



iOS (Swift)

Android (Java)

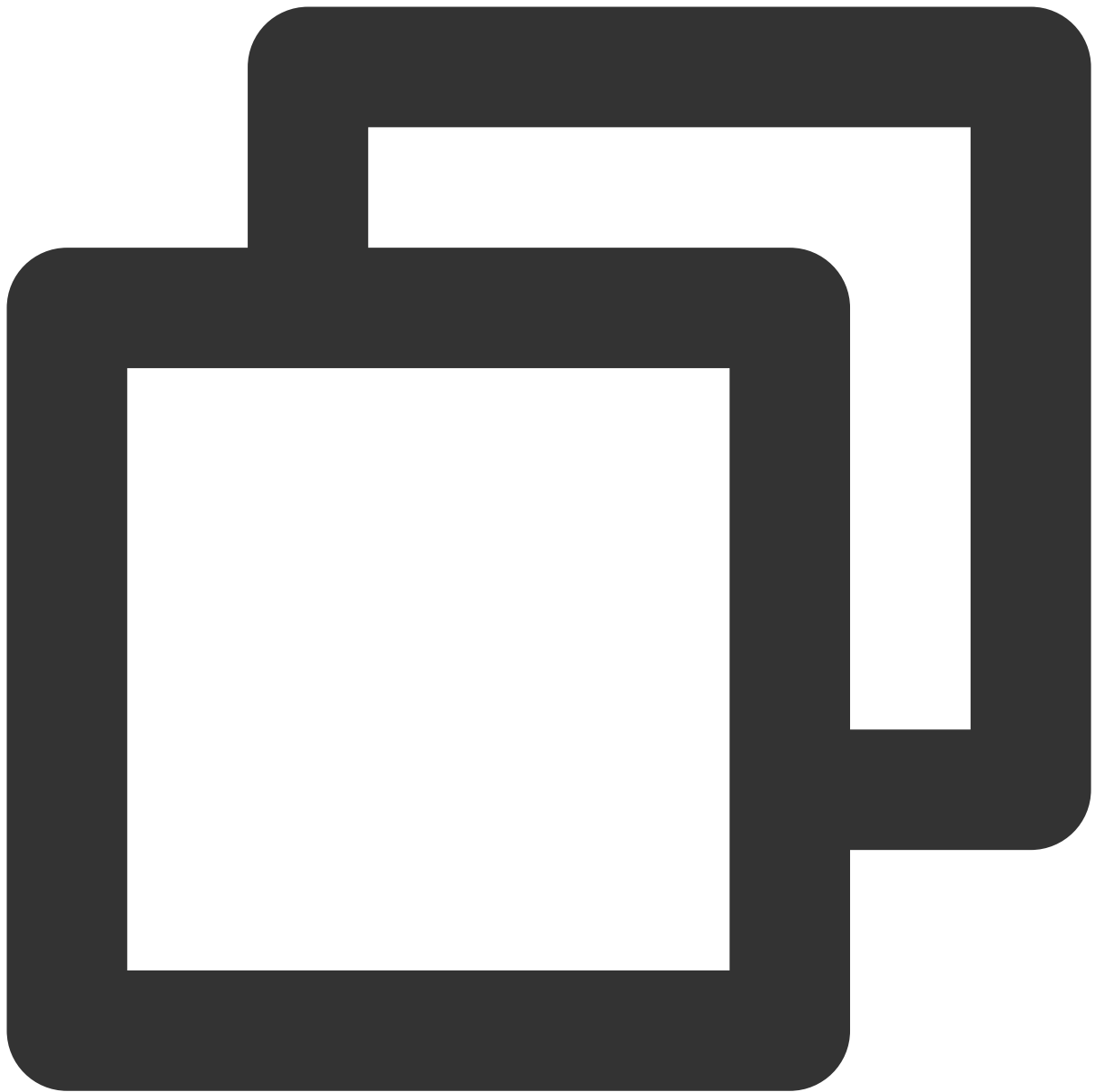
Flutter (Dart)



```
// CreateRoomViewController for your own ViewController
class CreateConferenceViewController: UIViewController {
    private var conferenceViewController: ConferenceMainViewController?
    func quickStartConferenceAction() {
        conferenceViewController = ConferenceMainViewController()
        // Implement the pre-conference control features by setting the parameters in
        let params = ConferenceParams()
        params.isMuteMicrophone = false
    }
}
```

```
params.isOpenCamera = false
params.isSoundOnSpeaker = true
params.name = "your conference name"
params.enableMicrophoneForAllUser = true
params.enableCameraForAllUser = true
params.enableMessageForAllUser = true
params.enableSeatControl = false
conferenceViewController?.setConferenceParams(params: params)
conferenceViewController?.setConferenceObserver(observer: self)
// After completing the settings, call the interface to start or join a conference
conferenceViewController?.quickStartConference(conferenceId: "your conference")
}
}

extension CreateConferenceViewController: ConferenceObserver {
    func onConferenceStarted(conferenceId: String, error: ConferenceError) {
        if error == .success, let vc = conferenceViewController {
            navigationController?.pushViewController(vc, animated: true)
        }
        conferenceViewController = nil
    }
}
```



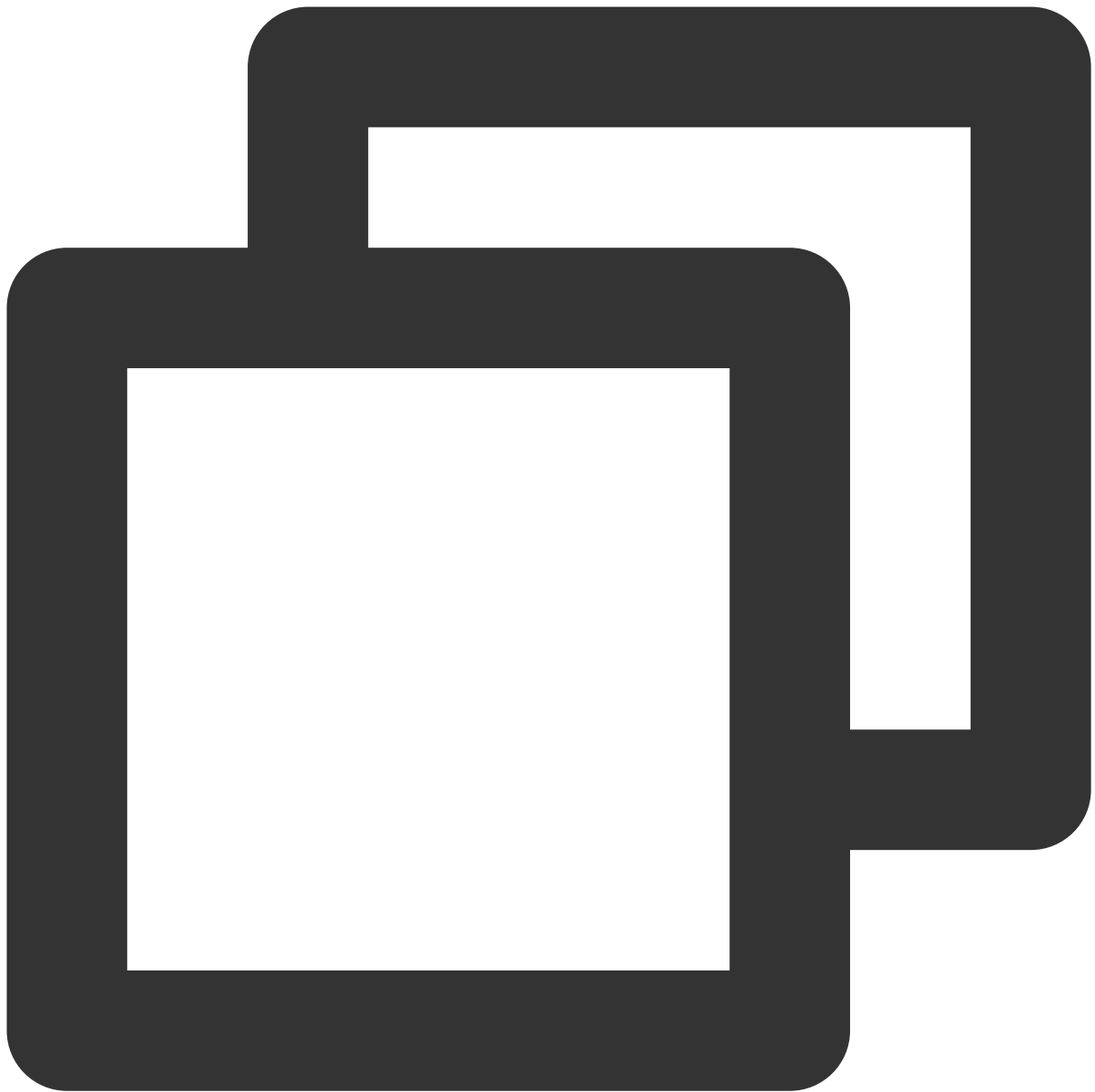
```
public class ConferenceOwnerActivity extends AppCompatActivity {
    private static final String TAG = "ConferenceOwnerActivity";

    private ConferenceObserver mConferenceObserver;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.app_activity_conference_main);
        // Implement the pre-conference control features by setting the parameters
        ConferenceParams params = new ConferenceParams();
        params.setMuteMicrophone(false);
    }
}
```



```
params.setOpenCamera(false);
params.setSoundOnSpeaker(true);
params.setName("your conference name");
params.setEnableMicrophoneForAllUser(true);
params.setEnableCameraForAllUser(true);
params.setEnableMessageForAllUser(true);
params.setEnableSeatControl(false);
ConferenceMainFragment fragment = new ConferenceMainFragment();
fragment.setConferenceParams(params);
setConferenceObserver(fragment);
fragment.quickStartConference("your conferenceId"); // After completing th
}

private void setConferenceObserver(ConferenceMainFragment fragment) {
    mConferenceObserver = new ConferenceObserver() {
        @Override
        public void onConferenceStarted(String conferenceId, ConferenceError er
            super.onConferenceStarted(conferenceId, error);
            if (error != ConferenceError.SUCCESS) {
                Log.e(TAG, "Error : " + error);
                return;
            }
            FragmentManager manager = getSupportFragmentManager();
            FragmentTransaction transaction = manager.beginTransaction();
            transaction.add(R.id.conference_owner_container, fragment);
            transaction.commitAllowingStateLoss();
        }
    };
    fragment.setConferenceObserver(mConferenceObserver);
}
}
```



```
var conferenceSession = ConferenceSession.newInstance("your conferenceId")
    ..isMuteMicrophone = false
    ..isOpenCamera = false
    ..isSoundOnSpeaker = true
    ..name = "your conference name"
    ..enableMicrophoneForAllUser = true
    ..enableCameraForAllUser = true
    ..enableMessageForAllUser = true
    ..enableSeatControl = false
    ..onActionSuccess = () { // Successful operation callback, you can navigate
        Navigator.push(
```

```
        context,
        MaterialPageRoute(
            builder: (context) => ConferenceMainPage(),
        ),
    );
}

..onActionError = (ConferenceError error, String message) {} // Failure oper
..quickStart(); // After completing the settings, call the interface
```

Here is a detailed introduction to the parameters in the above code.

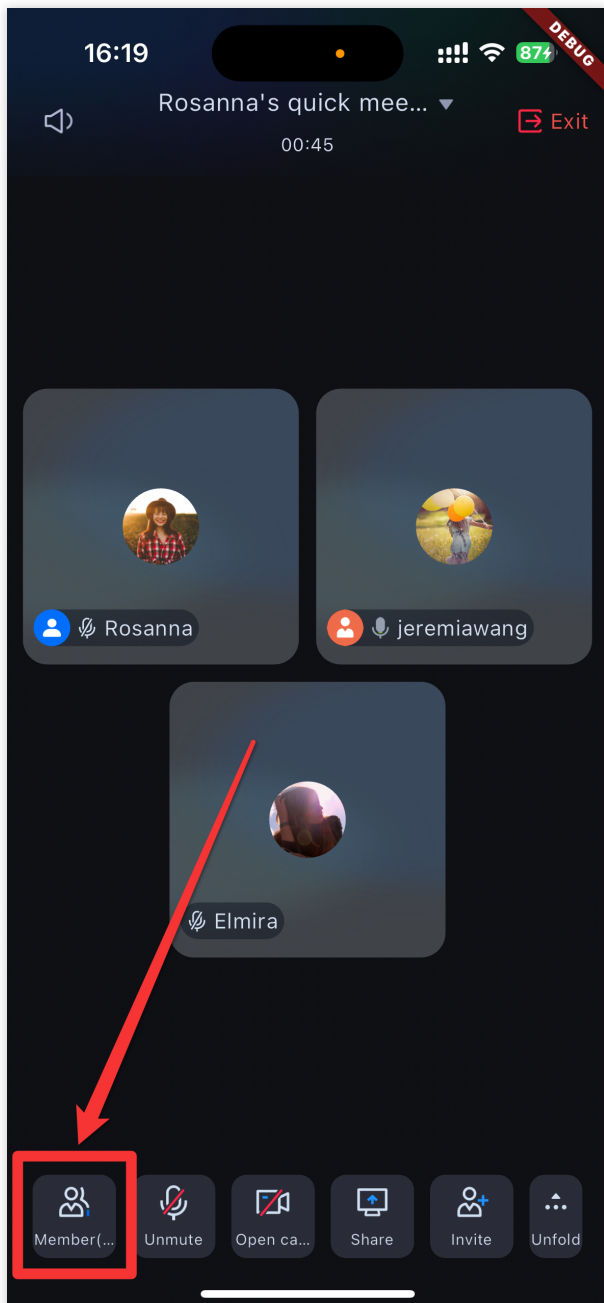
Parameter	Type	Meaning
isMuteMicrophone	bool	Whether to mute the microphone (default is false)
isOpenCamera	bool	Whether to open the camera (default is false)
isSoundOnSpeaker	bool	Whether to turn on the speakers (default is true)
name	String	conference name (default is your conferenceld)
enableMicrophoneForAllUser	bool	Whether to enable microphone permission for all members (default is true)
enableCameraForAllUser	bool	Whether to enable camera permission for all members (default is true)
enableMessageForAllUser	bool	Whether to enable message permission for all members (default is true)
enableSeatControl	bool	Whether to enable on-stage speaking mode (default is false)

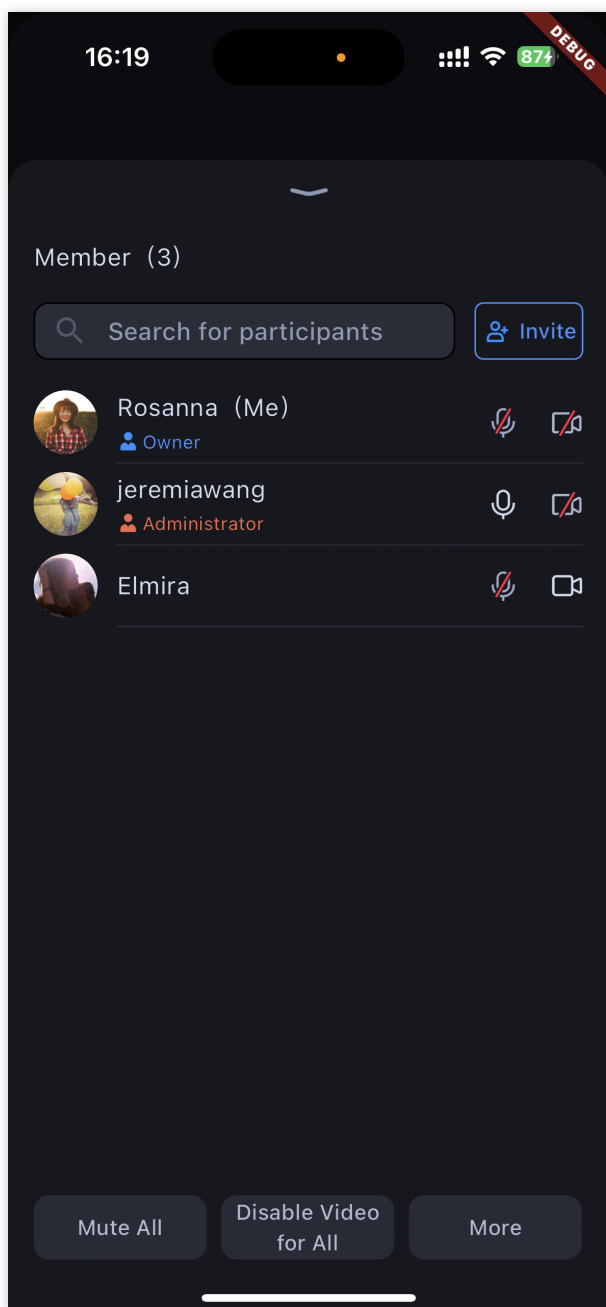
## In-conference Control

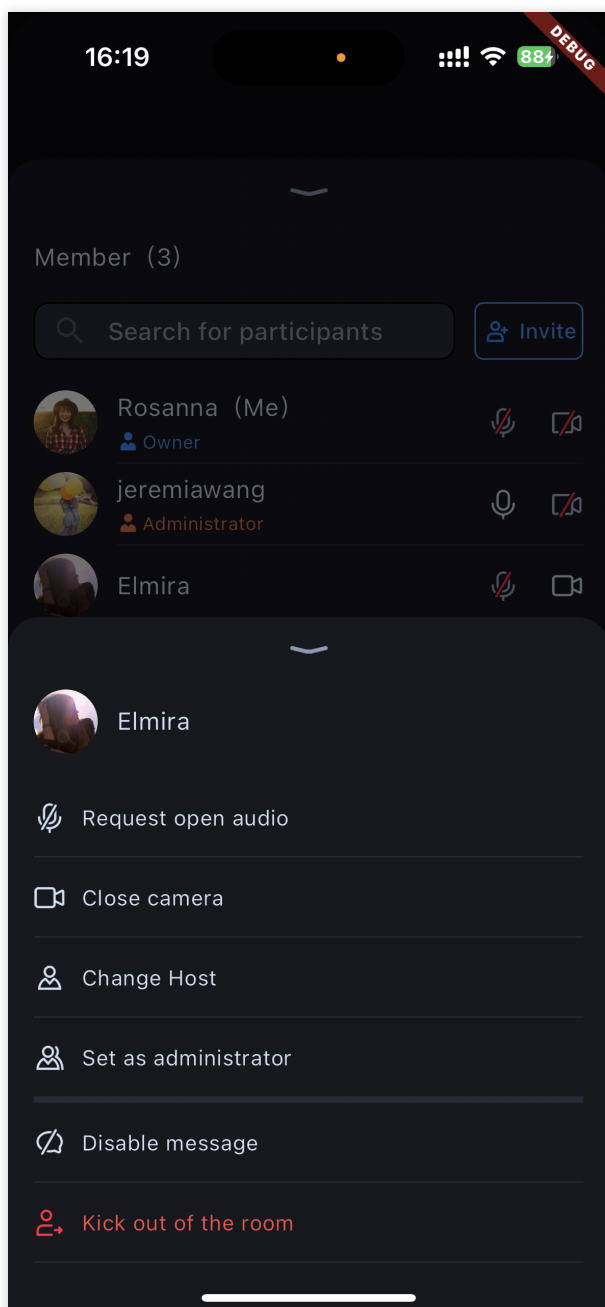
During a conference, you may need to control the conference at any time to meet the needs of different scenarios.

`TUIRoomKit` provides a wealth of in-conference control functions, allowing you to easily handle various conference situations. Through this section, you will intuitively understand how to effectively control the conference during the conference.

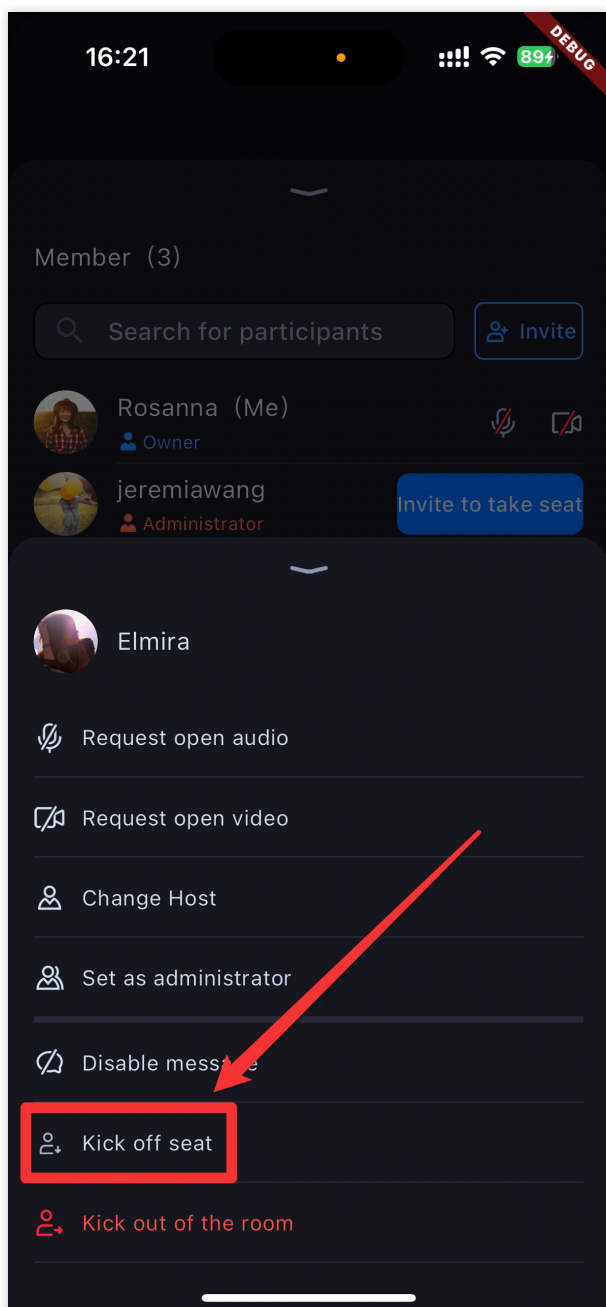
When you are the host or administrator, click the member button in the lower left corner to open the member list for related conference control, as shown in the second picture from the below. In the member list, click on the user you need to operate, and the interface shown in the first picture on the right will appear.

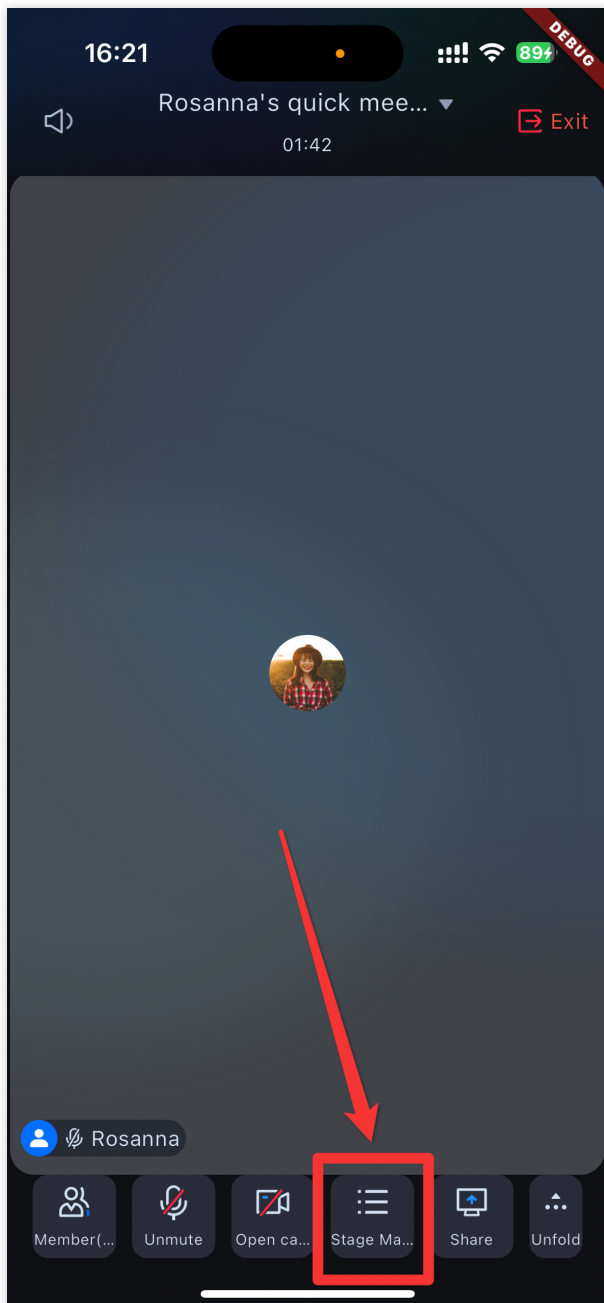




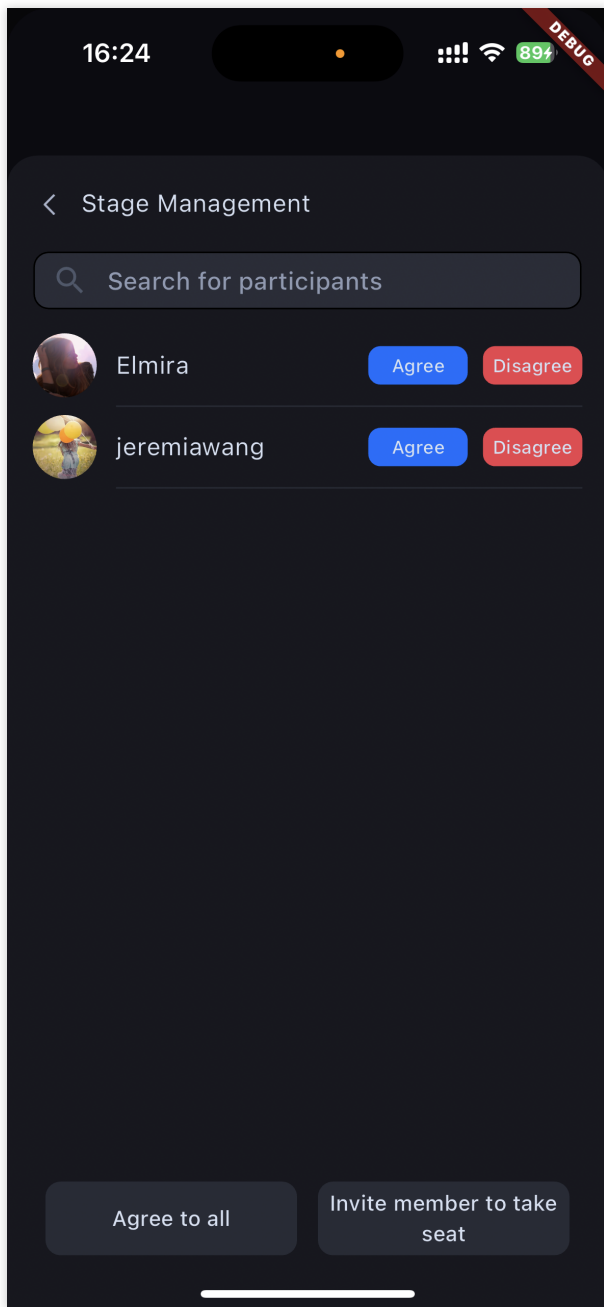


In the on-stage speaking mode, click on the user you need to operate in the member list, and the interface that appears is as shown in the first picture on the left below. Click the `Stage Management` button on the conference bottom toolbar to open the stage management interface, as shown in the first picture on the right below.









Function	Corresponding interface (Android as an example)
Disable/Enable video for all	<a href="#">disableDeviceForAllUserByAdmin</a>
Turn off/Request to turn on user's audio/video	<a href="#">closeRemoteDeviceByAdmin</a> / <a href="#">openRemoteDeviceByAdmin</a>
Transfer host	<a href="#">changeUserRole</a>
Set/Cancel administrator	<a href="#">changeUserRole</a>
Disable/enable message for user	<a href="#">disableSendingMessageByAdmin</a>

Kick out of the room	<a href="#">kickRemoteUserOutOfRoom</a>
Invite to stage/Ask to step down	<a href="#">takeUserOnSeatByAdmin</a> / <a href="#">kickUserOffSeatByAdmin</a>
Agree/Reject user's stage application	<a href="#">responseRemoteRequest</a>

# API Documentation(TUIRoomKit)

## iOS&Mac

## API Overview

Last updated : 2023-12-18 18:01:32

### TUIRoomEngine API List

TUIRoomEngine API is the UI-free interface of the Conference Component, which allows you to customize the encapsulation according to your business needs.

TUIRoomEngine

#### TUIRoomEngine Core Methods

API	Description
<a href="#">init</a>	Create Instance of TUIRoomEngine.
<a href="#">login</a>	Login Interface, you need to initialize user information before entering the room and perform a series of operations.
<a href="#">logout</a>	Logout Interface, there will be active room leaving operation and resources destruction.
<a href="#">setSelfInfo</a>	Set local user name and avatar.
<a href="#">getSelfInfo</a>	Get the basic information of the local user login.
<a href="#">addObserver</a>	Set Event Callback.
<a href="#">removeObserver</a>	Remove Event Callback.

#### Active Interface related to the room

API	Description
<a href="#">createRoom</a>	Create a room.
<a href="#">destroyRoom</a>	Close the room.
<a href="#">enterRoom</a>	Entered room.

<a href="#">exitRoom</a>	Leave the room.
<a href="#">connectOtherRoom</a>	Connect to another room.
<a href="#">disconnectOtherRoom</a>	Disconnect from another room.
<a href="#">fetchRoomInfo</a>	Get Room data.
<a href="#">updateRoomNameByAdmin</a>	Update Room ID (only administrator or group owner can call).
<a href="#">updateRoomSpeechModeByAdmin</a>	Set Mic Control Mode for the room (only administrator or group owner can call).

## Local user view rendering and video management

API	Description
<a href="#">setLocalVideoView</a>	Set the View Control for local user video rendering.
<a href="#">openLocalCamera</a>	Open local Camera.
<a href="#">closeLocalCamera</a>	Close local Camera.
<a href="#">updateVideoQuality</a>	Update Encoding Quality settings for local video.
<a href="#">startPushLocalVideo</a>	Start pushing local video.
<a href="#">stopPushLocalVideo</a>	Stop pushing local video.
<a href="#">startScreenCapture</a>	Start Screen Sharing (this interface is only supported on mobile devices).
<a href="#">startScreenCapture</a>	Start Screen Sharing (this interface is only supported on Mac OS desktop systems).
<a href="#">stopScreenCapture</a>	End Screen Sharing.
<a href="#">getScreenCaptureSources</a>	Enumerate shareable screens and windows (this interface is only supported on Mac OS systems).
<a href="#">selectScreenCaptureTarget</a>	Select the screen or window to share (this interface is only supported on Mac OS systems).

## Local user audio management

API	Description
<a href="#">openLocalMicrophone</a>	Open local mic.

<a href="#">closeLocalMicrophone</a>	Close local mic.
<a href="#">updateAudioQuality</a>	Update local Audio Encoding Quality settings.
<a href="#">startPushLocalAudio</a>	Start pushing local audio.
<a href="#">stopPushLocalAudio</a>	Stop pushing local audio.

## Remote user view rendering and video management

API	Description
<a href="#">setRemoteVideoView</a>	Set the View Control for remote user video rendering.
<a href="#">startPlayRemoteVideo</a>	Start Playback of remote user video.
<a href="#">stopPlayRemoteVideo</a>	Stop Playback of remote user video.
<a href="#">muteRemoteAudioStream</a>	Mute remote user.

## Room user information

API	Description
<a href="#">getUserList</a>	Get the member list in the room.
<a href="#">getUserInfo</a>	Get member information.

## Room user management

API	Description
<a href="#">changeUserRole</a>	Modify user role (only administrator or group owner can call).
<a href="#">kickRemoteUserOutOfRoom</a>	Kick remote user out of the room (only administrator or group owner can call).

## Room user speech management

API	Description
<a href="#">disableDeviceForAllUserByAdmin</a>	Control the permission status of all users in the current room to open audio streams, video streams, and capture devices, such as: all users are prohibited from opening mics, all users are prohibited from opening cameras, all users are prohibited from

	opening screen sharing (currently only available in conference scenarios, and only administrators or group owners can call).
<a href="#">openRemoteDeviceByAdmin</a>	Request remote user to open media device (only administrator or group owner can call).
<a href="#">closeRemoteDeviceByAdmin</a>	Close remote user media device (only administrator or group owner can call).
<a href="#">applyToAdminToOpenLocalDevice</a>	Request to open local media device (available for ordinary users).

## Room mic seat management

API	Description
<a href="#">setMaxSeatCount</a>	Set the maximum number of mic seats (only supported when entering the room and creating the room).
<a href="#">getSeatList</a>	Get the list of mic seats.
<a href="#">lockSeatByAdmin</a>	Lock the mic seat (only administrator or group owner can call, including position lock, audio status lock, and video status lock).
<a href="#">takeSeat</a>	Apply to Go Live (no need to apply in free speech mode).
<a href="#">leaveSeat</a>	Apply to leave the mic (no need to apply in free speech mode).
<a href="#">takeUserOnSeatByAdmin</a>	Host/Administrator invites user to Go Live.
<a href="#">kickUserOffSeatByAdmin</a>	Host/Administrator kicks user off the mic.

## Signaling management

API	Description
<a href="#">cancelRequest</a>	Cancel Request.
<a href="#">responseRemoteRequest</a>	Reply Request.

## Send message

API	Description

<a href="#">sendMessage</a>	Send Text Message.
<a href="#">sendCustomMessage</a>	Send Custom Message.
<a href="#">disableSendingMessageByAdmin</a>	Disable remote user's ability to send text messages (only administrator or group owner can call).
<a href="#">disableSendingMessageForAllUser</a>	Disable all users' ability to send text messages (only administrator or group owner can call).

### Advanced features: Get TRTC instance

API	Description
<a href="#">getDeviceManager</a>	Get native TRTC Device Management class.
<a href="#">getAudioEffectManager</a>	Get native TRTC Sound Effect Class.
<a href="#">getBeautyManager</a>	Get native TRTC Beauty Class.
<a href="#">getTRTCCloud</a>	Get native TRTC Instance Class.

## TUIRoomObserver Callback Events

TUIRoomObserver is the callback event class corresponding to TUIRoomEngine. You can listen to the callback events you need through this callback.

TUIRoomObserver

## TUIRoomObserver

### Error callback

API	Description
<a href="#">onError</a>	Error Event Callback.

### Login status event callback

API	Description

<a href="#">onKickedOffLine</a>	Other terminal login is kicked offline.
<a href="#">onUserSigExpired</a>	User Credential Timeout Event.

## Room event callback

API	Description
<a href="#">onRoomNameChanged</a>	Room ID change event.
<a href="#">onAllUserMicrophoneDisableChanged</a>	All users in the room have their mics disabled event.
<a href="#">onAllUserCameraDisableChanged</a>	All users in the room have their cameras disabled event.
<a href="#">onSendMessageForAllUserDisableChanged</a>	All users in the room have their text message sending disabled event.
<a href="#">onKickedOutOfRoom</a>	Kicked out of the room event.
<a href="#">onRoomDismissed</a>	Room closed event.
<a href="#">onRoomSpeechModeChanged</a>	Room Mic Control Mode changed.

## Room user event callback

API	Description
<a href="#">onRemoteUserEnterRoom</a>	Remote user entered room event.
<a href="#">onRemoteUserLeaveRoom</a>	Remote user left the room event.
<a href="#">onUserRoleChanged</a>	User role changed event.
<a href="#">onUserVideoStateChanged</a>	User video status changed event.
<a href="#">onUserAudioStateChanged</a>	User audio status changed event.
<a href="#">onUserScreenCaptureStopped</a>	User screen capture stopped event.
<a href="#">onUserVoiceVolumeChanged</a>	User volume change event.
<a href="#">onSendMessageForUserDisableChanged</a>	User text message sending ability changed event.
<a href="#">onUserNetworkQualityChanged</a>	User network status change event.

## Room mic seat event callback

--	--



API	Description
<a href="#">onRoomMaxSeatCountChanged</a>	Maximum mic seat changed event in the room (only effective in conference type rooms).
<a href="#">onSeatListChanged</a>	Mic seat list changed event.
<a href="#">onKickedOffSeat</a>	Received user kicked off mic event.

### Request signaling event callback

API	Description
<a href="#">onRequestReceived</a>	Received request message event.
<a href="#">onRequestCancelled</a>	Received request cancelled event.

### Room message event callback

API	Description
<a href="#">onReceiveTextMessage</a>	Received normal text message event.
<a href="#">onReceiveCustomMessage</a>	Received custom message event.

# TUIRoomKit

Last updated : 2024-03-15 16:07:57

## Introduction

TUIRoomKit is an open source UI layer suite for conference SDK, currently only supports Swift language on the iOS platform. The conference UI can be called up through simple API calls.

## TUIRoomKit Interface

API	Description
<a href="#">createInstance</a>	Initialize the TUIRoomKit singleton object
<a href="#">destroyInstance</a>	Destroy the TUIRoomKit singleton object
<a href="#">setSelfInfo</a>	Set user information (avatar, nickname) (optional)
<a href="#">createRoom</a>	Create a room
<a href="#">enterRoom</a>	Enter the room

### **createInstance**

Initialize the TUIRoomKit singleton object.



```
public class func createInstance() -> TUIRoomKit
```

### **destroyInstance**

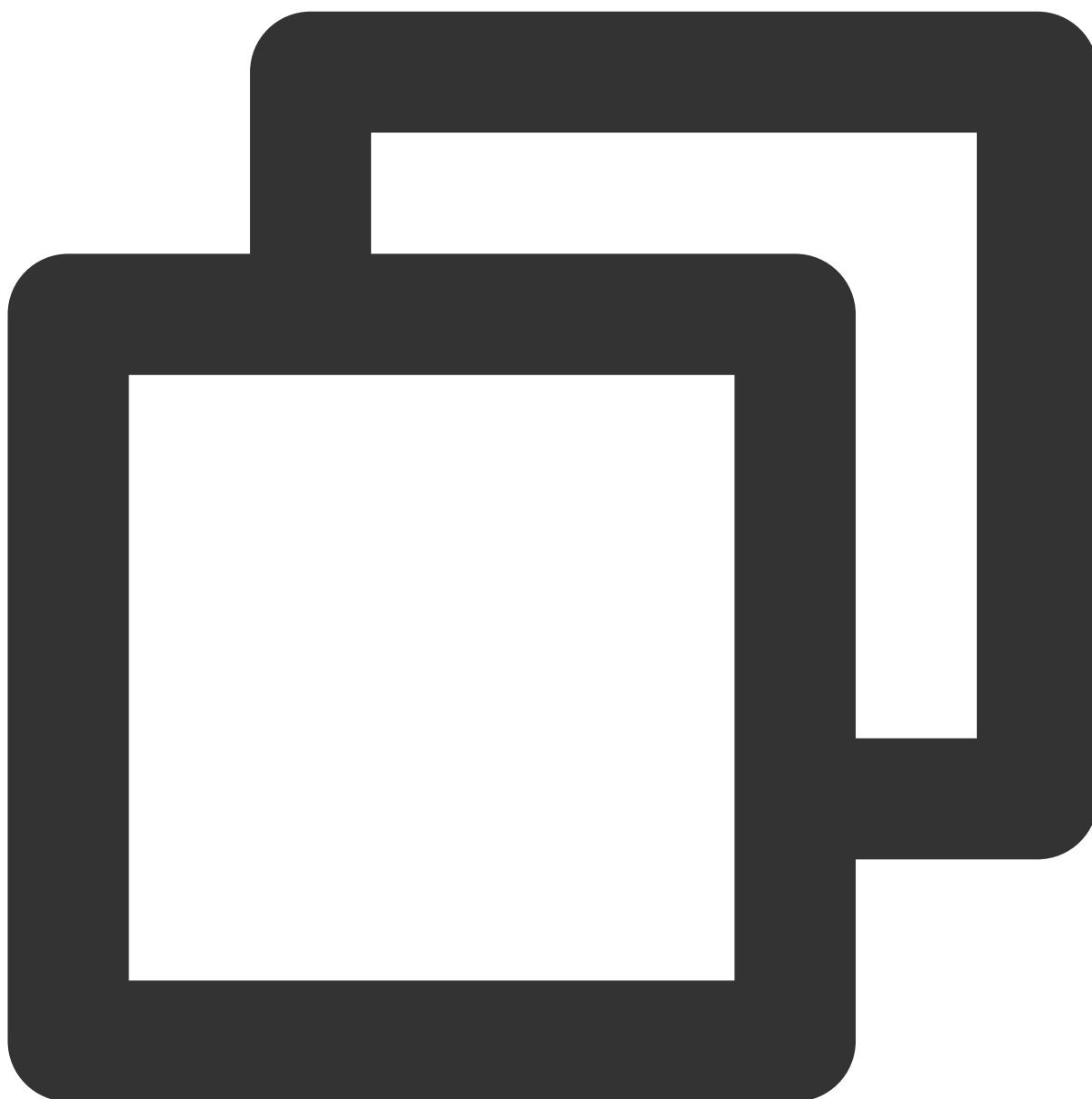
Destroy the TUIRoomKit singleton object.



```
public class func destroyInstance() -> Void
```

### **setSelfInfo(optional)**

Set user information (avatar, nickname).



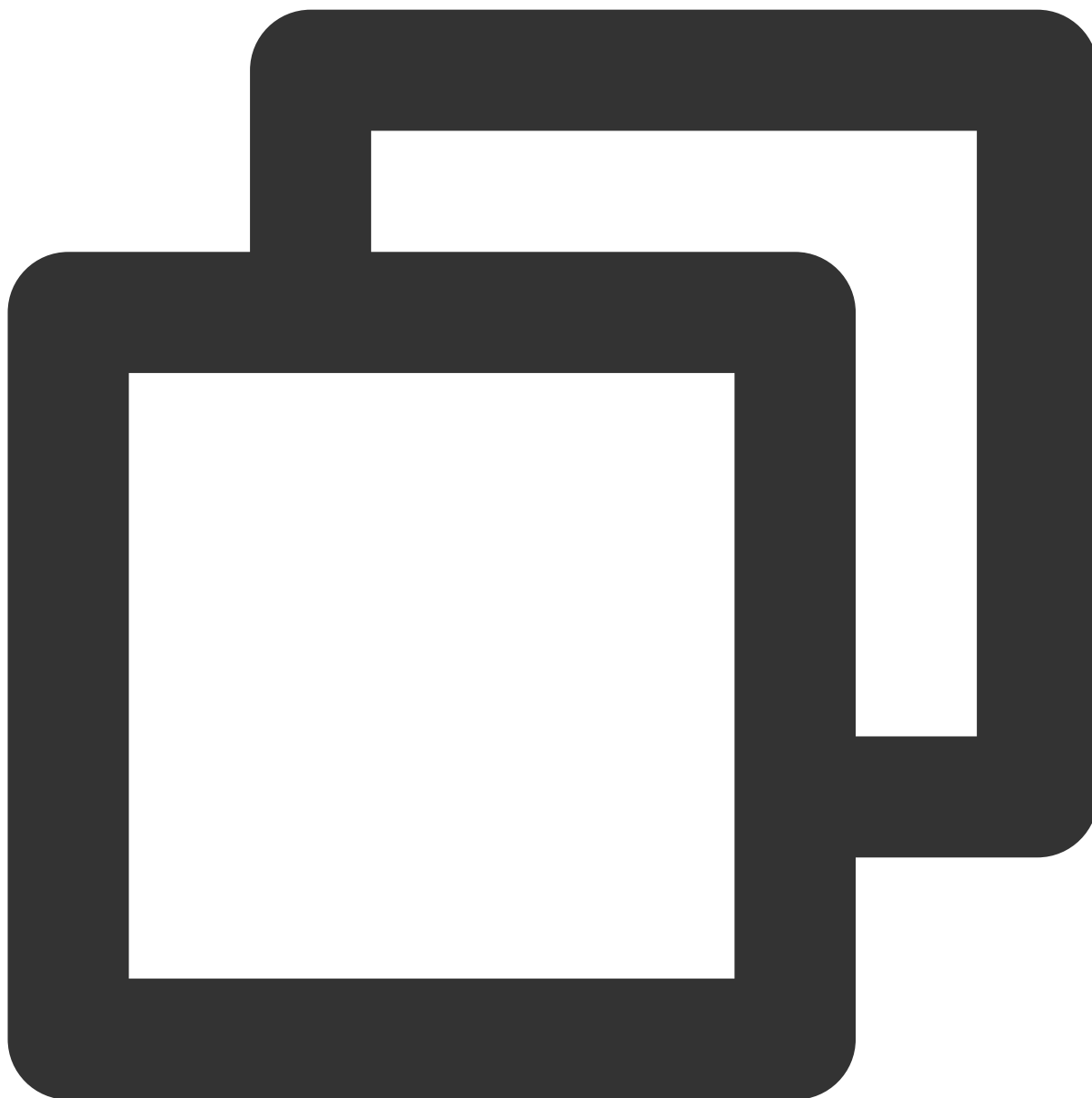
```
public func setSelfInfo(userName: String,  
                        avatarURL: String,  
                        onSuccess: @escaping TUISuccessBlock,  
                        onError: @escaping TUIErrorBlock) -> Void
```

Parameter	Type	Meaning
userName	String	Username
avatarURL	String	User avatar URL

onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failure callback

## createRoom

Create a room.



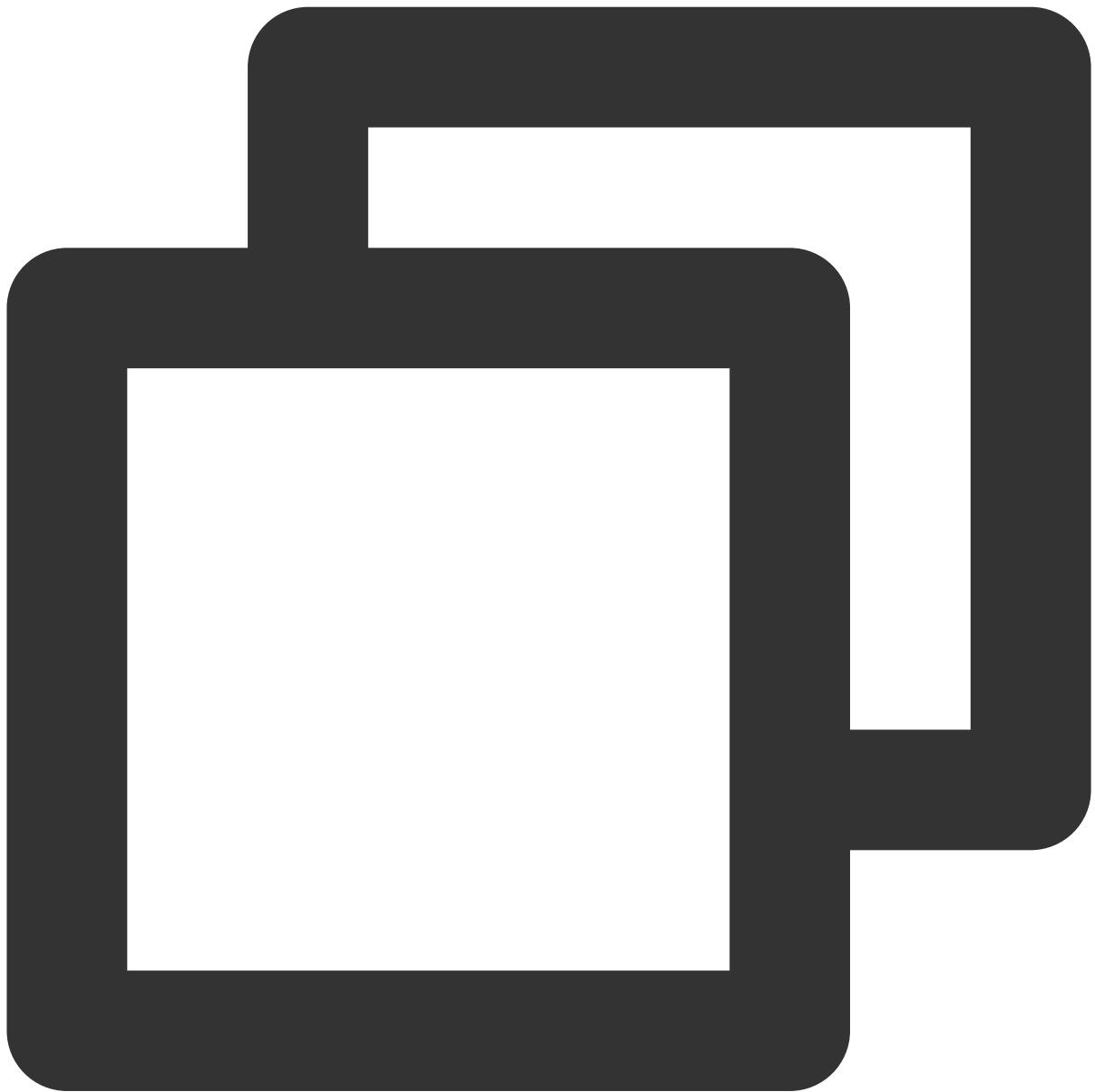
```
public func createRoom(roomInfo: TUIRoomInfo,  
                        onSuccess: @escaping TUISuccessBlock,  
                        onError: @escaping TUIErrorBlock) -> Void
```

The parameters are as follows:

Parameter	Type	Meaning
roomInfo	<a href="#">TUIRoomInfo</a>	Room data
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failure callback

## enterRoom

Enter the room.



```
public func enterRoom(roomId: String,  
    enableAudio: Bool,  
    enableVideo: Bool,  
    isSoundOnSpeaker: Bool,  
    onSuccess: @escaping TUISuccessBlock,  
    onError: @escaping TUIErrorBlock) -> Void
```

The parameters are as follows:

Parameter	Type	Meaning



roomId	String	Room ID string
enableAudio	Bool	Whether to turn on the audio when entering the room
enableVideo	Bool	Whether to turn on the video when entering the room
isSoundOnSpeaker	Bool	Whether to turn on the speakers when entering the room
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failure callback

# TUIRoomEngine

Last updated : 2023-12-18 17:57:07

## TUIRoomEngine API Introduction

TUIRoomEngine API is a No UI Interface for Conference Component, you can use this API to custom encapsulate according to your business needs.

### **init**

Create TUIRoomEngine instance

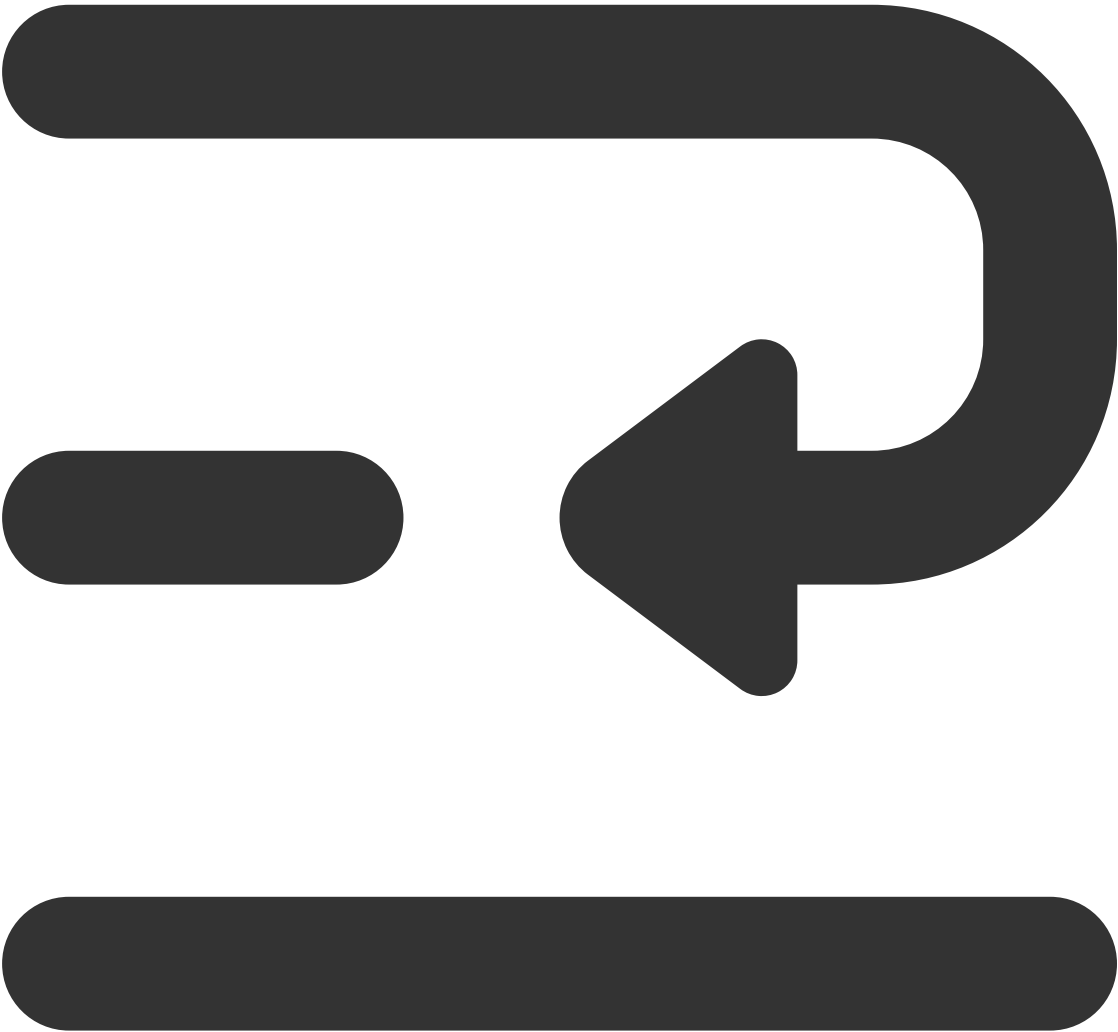


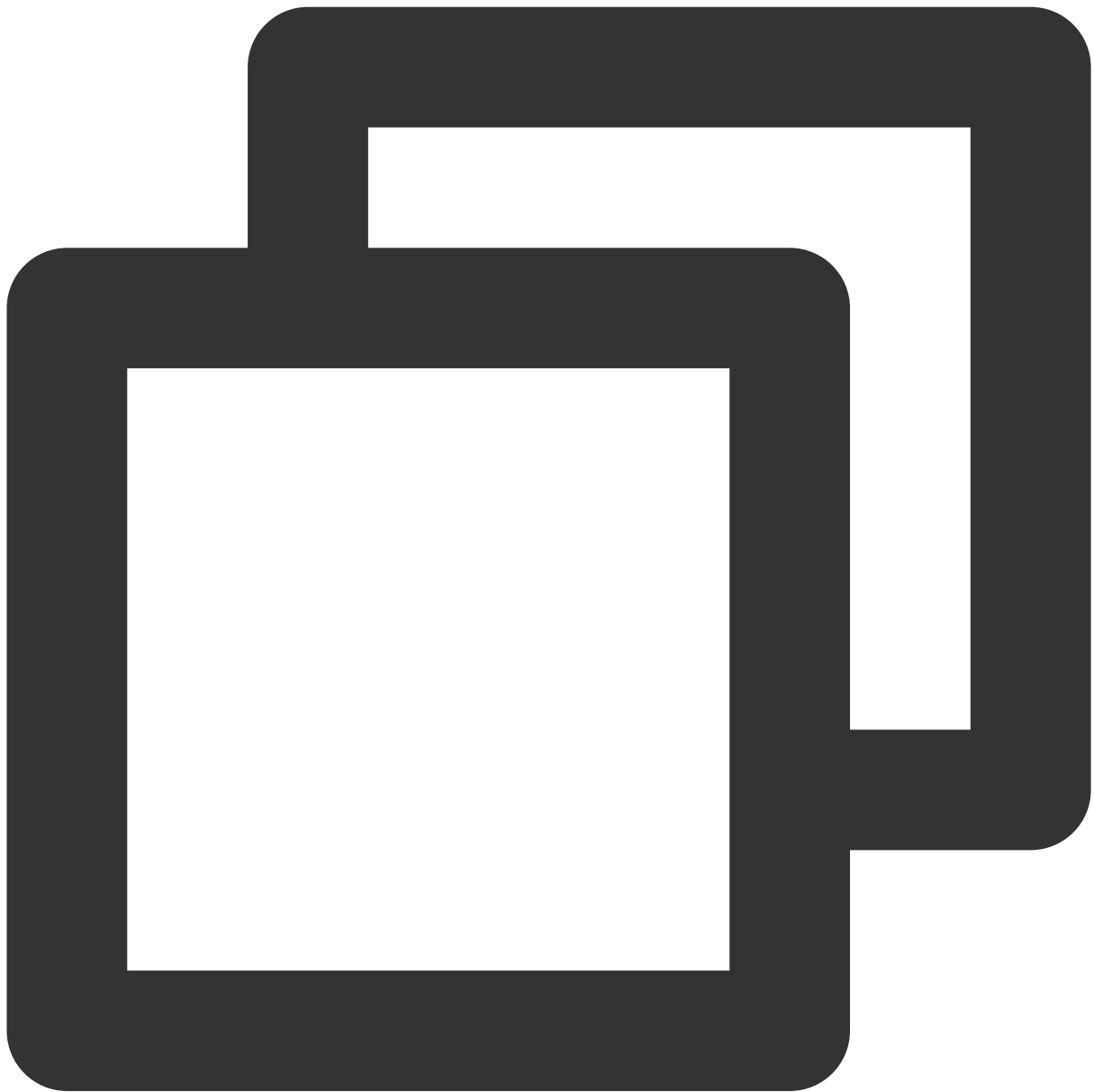
```
TUIRoomEngine *roomEngine = [[TUIRoomEngine alloc] init];
```

## login

Login interface, you need to initialize user information before entering the room and perform a series of operations

**Note:** In v1.0.0, this interface is named setup, and in v1.0.1 and above, please use TUIRoomEngine.login to log in to TUIRoomEngine.





```
+ (void)loginWithSDKAppId:(NSInteger) sdkAppId
    userId:(NSString *)userId
    userSig:(NSString *)userSig
    onSuccess:(TUISuccessBlock) onSuccess
    onError:(TUIErrorBlock) onError;
```

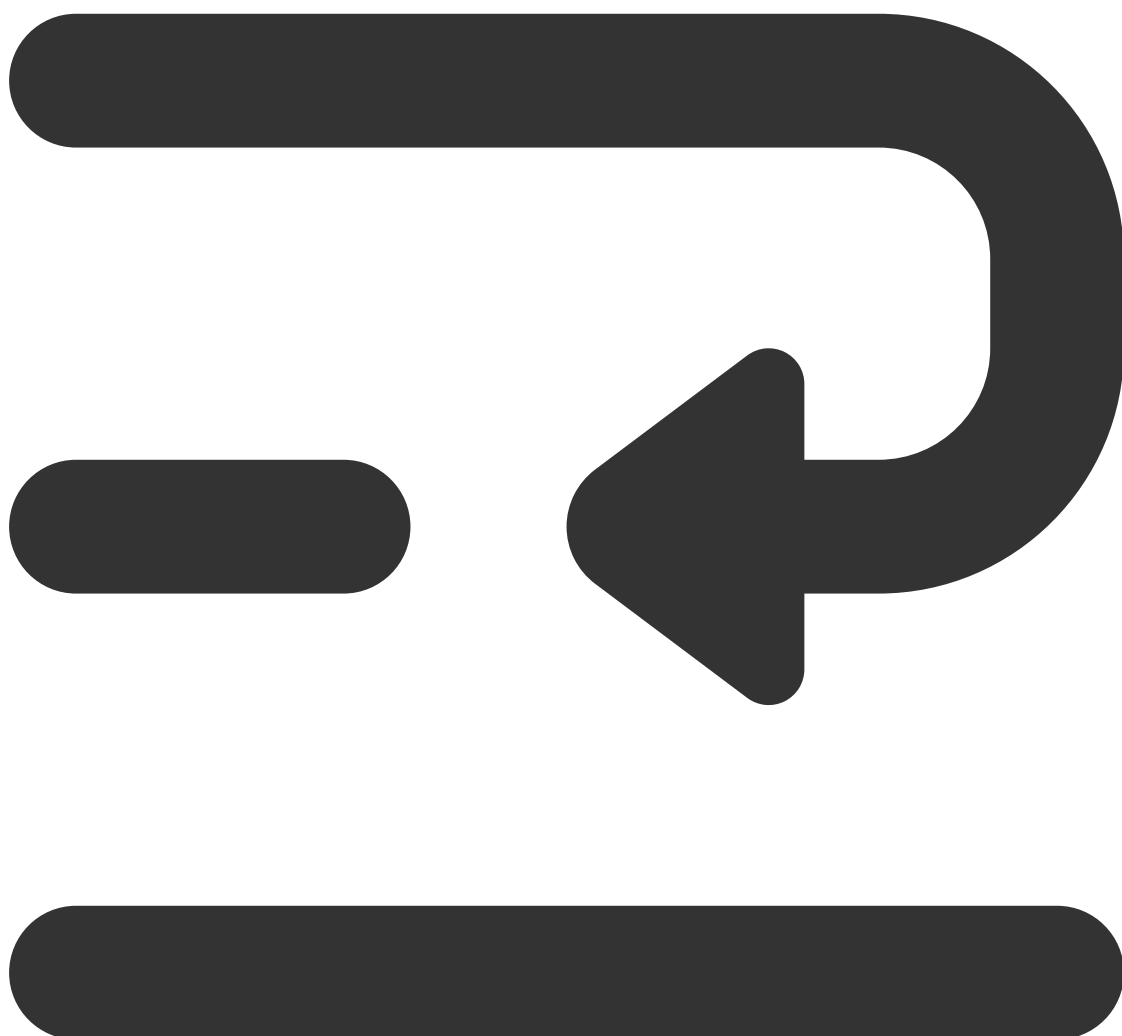
**The parameters are as follows:**

Parameter	Type	Meaning
sdkAppId	NSInteger	SDKAppID of Tencent Cloud communication

		application
userId	NSString *	User ID for Differentiate different users
userSig	NSString *	UserSig for Tencent Cloud flow authentication
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## logout

Logout interface, there will be actively leave the room operation, destroy resources





```
+ (void)logout:(TUISuccessBlock)onSuccess  
    onError:(TUIErrorBlock)onError;
```

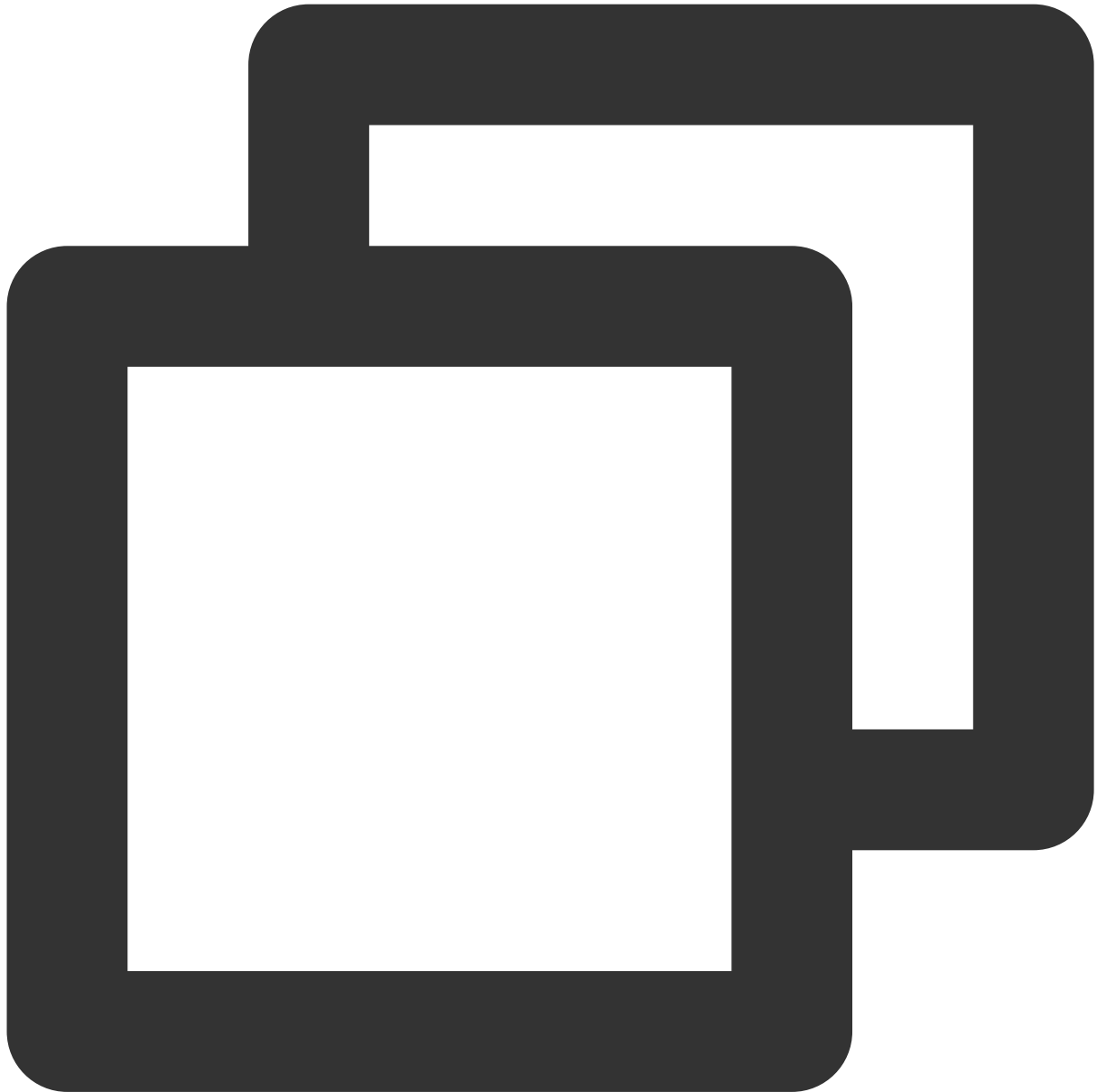
**The parameters are as follows:**

Parameter	Type	Meaning
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback



## setSelfInfo

Set local user name and avatar.



```
+ (void)setSelfInfoWithUserName:(NSString *)userName
      avatarUrl:(NSString *)avatarUrl
    onSuccess:(TUISuccessBlock)onSuccess
    onError:(TUIErrorBlock)onError;
```

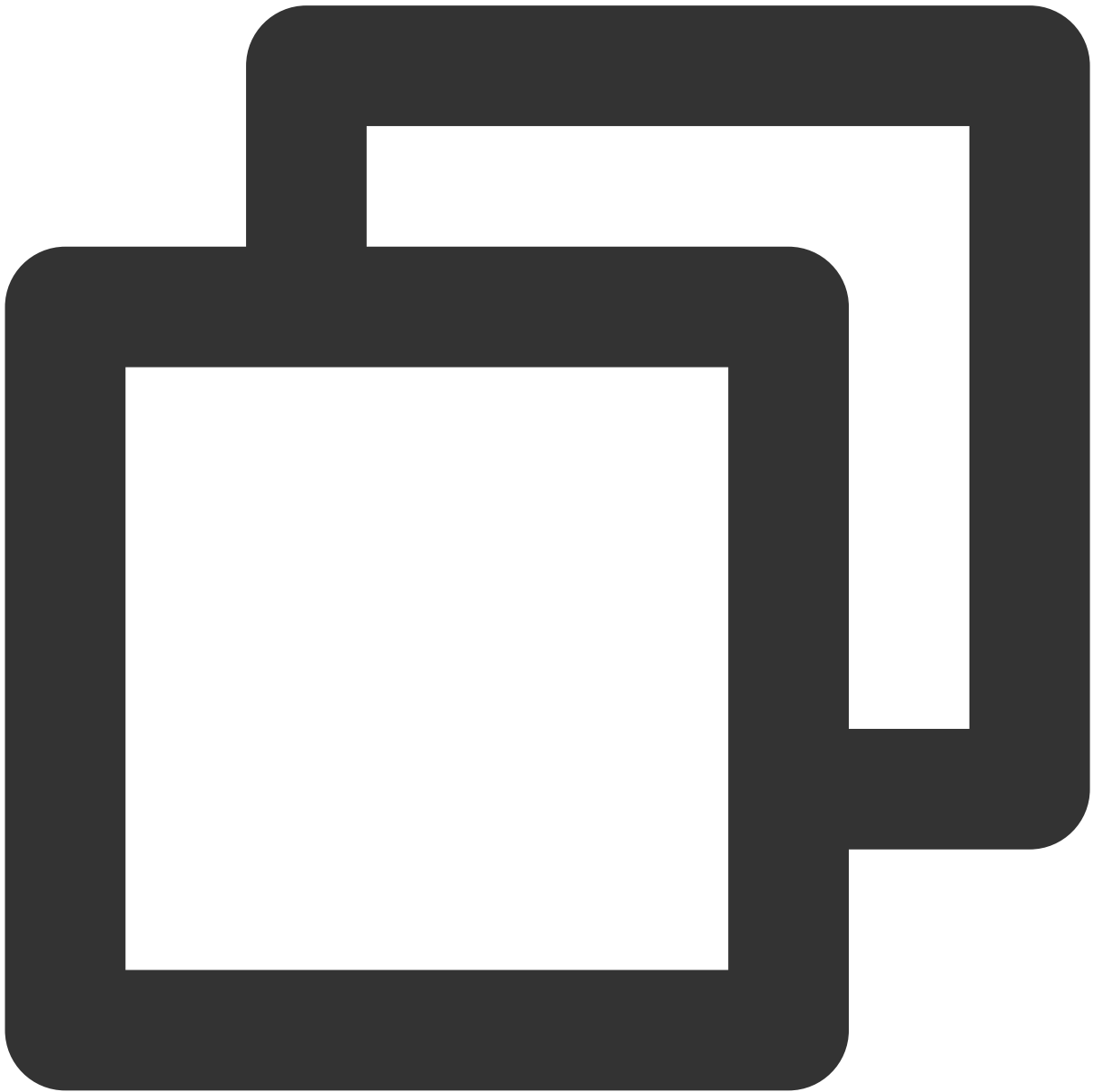
The parameters are as follows:

Parameter	Type	Meaning

userName	NSString *	User name
avatarUrl	NSString *	User avatar URL address
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

### getSelfInfo

Get the basic information of the local user login.



```
+ (TUILoginUserInfo *)getSelfInfo;
```

The parameters are as follows:

Parameter	Type	Meaning
selfInfo	<a href="#">TUILoginUserInfo</a> *	The basic information of the local user login, the detailed definition can be referred to <code>TUIRoomDefine.h</code> in <a href="#">TUILoginUserInfo</a> .

## addObserver

Set event callback.



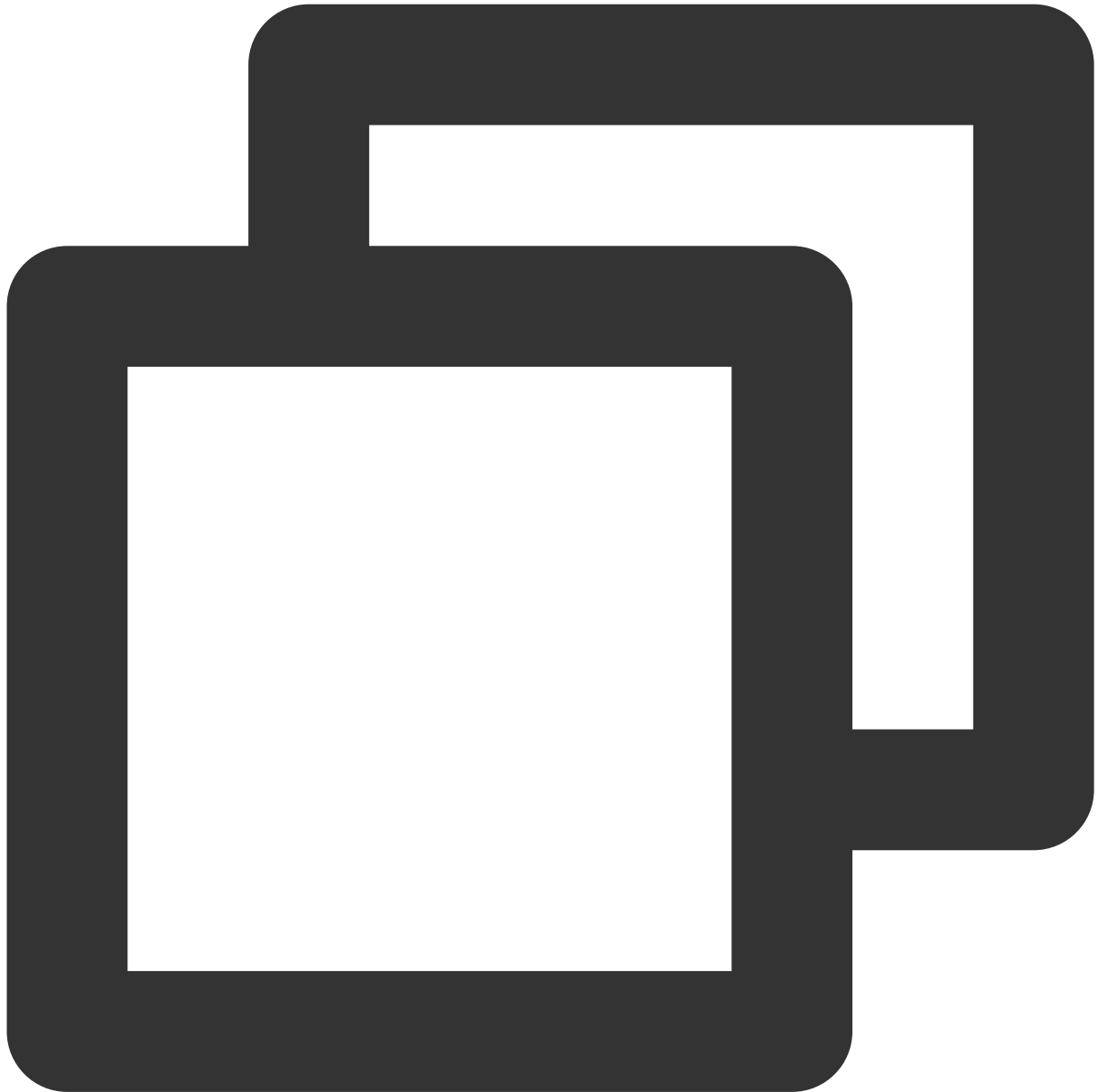
```
- (void)addObserver:(id<TUIRoomObserver>) observer;
```

**The parameters are as follows:**

Parameter	Type	Meaning
observer	TUIRoomObserver *	The pointer of the callback instance, you can get various event notifications (such as: error code, remote user entered room, audio and video status parameters, etc.) through TUIRoomObserver

## removeObserver

Remove event callback.



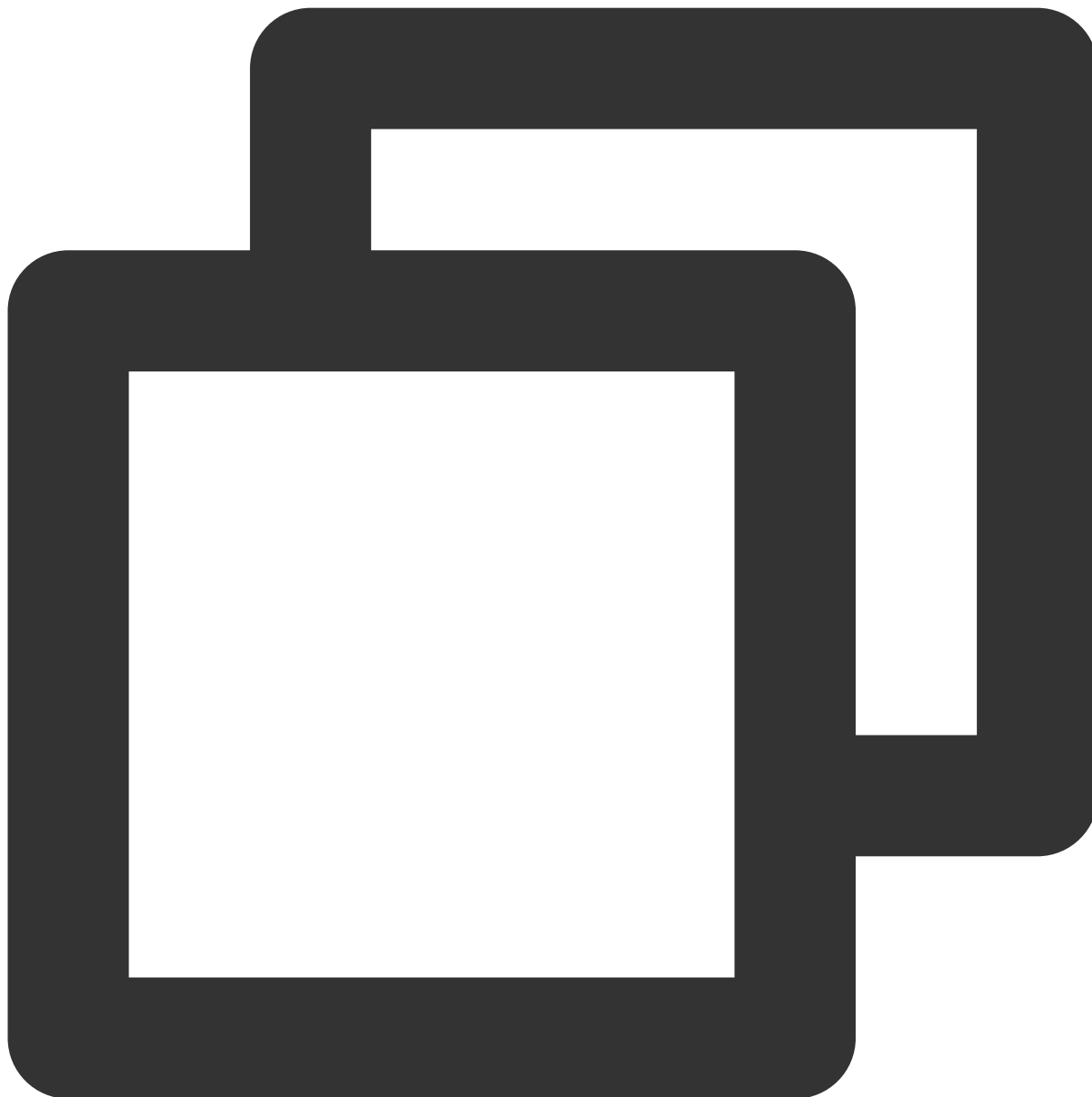
```
- (void)removeObserver:(id<TUIRoomObserver>)observer;
```

The parameters are as follows:

Parameter	Type	Meaning
observer	TUIRoomObserver *	The pointer of the callback instance

## createRoom

Create room.



```
- (void)createRoom:(TUIRoomInfo *)roomInfo
    onSuccess:(TUISuccessBlock)onSuccess
    onError:(TUIErrorBlock)onError;
```

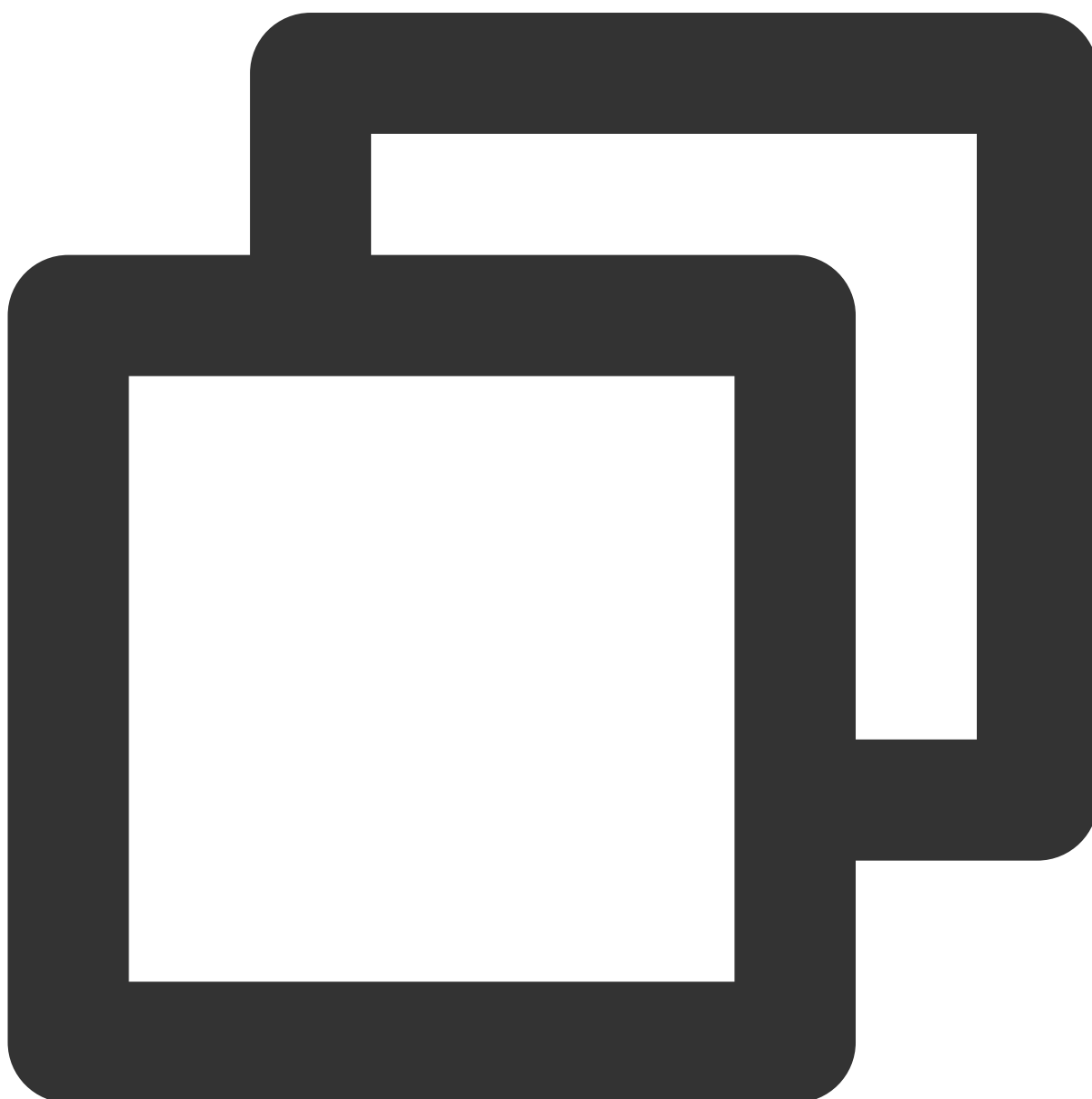
The parameters are as follows:

Parameter	Type	Meaning

roomInfo	<a href="#">TUIRoomInfo</a> *	Room data, you can initialize some room settings
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## destroyRoom

close the room.



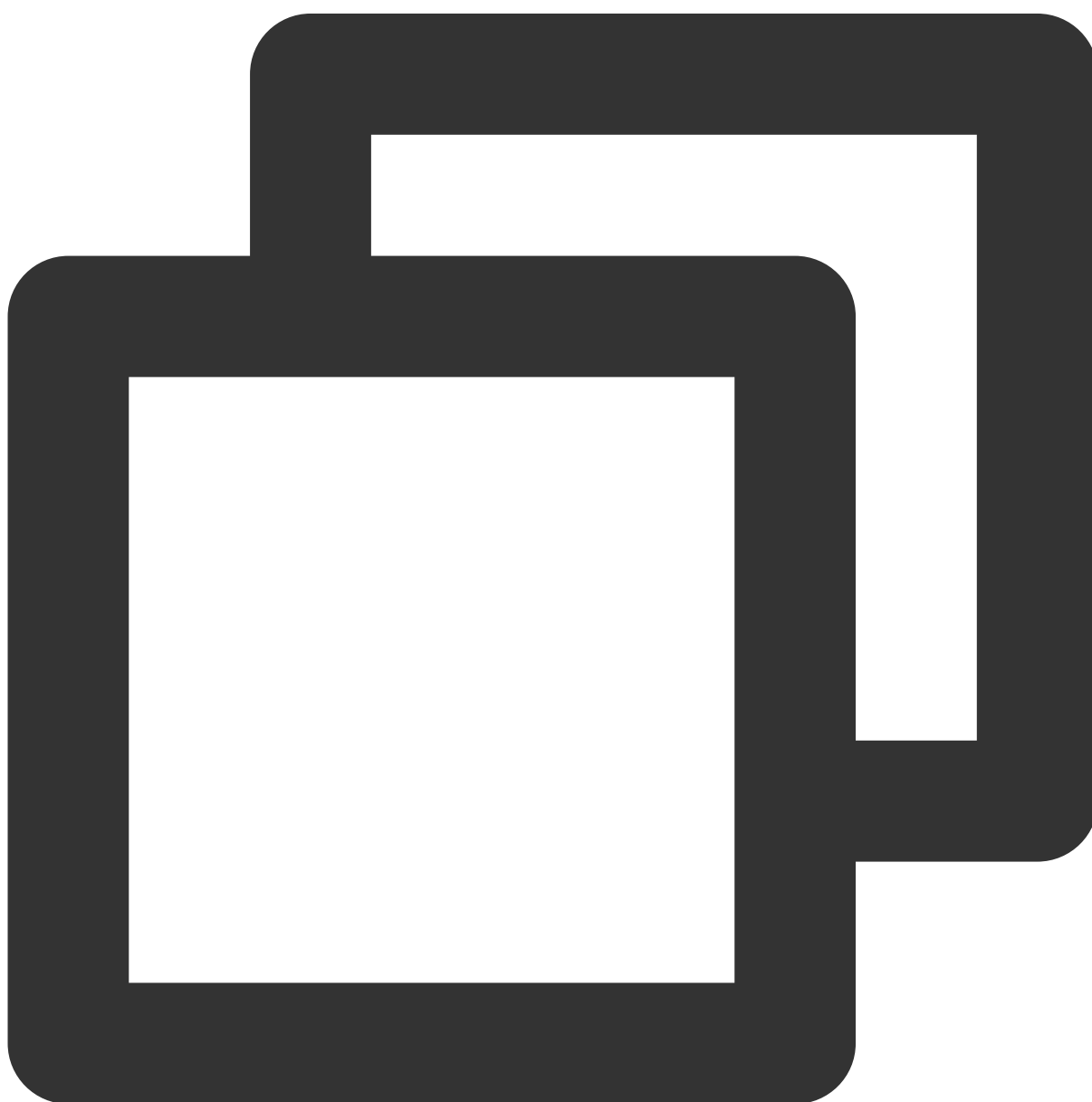
```
- (void)destroyRoom:(TUISuccessBlock)onSuccess onError:(TUIErrorBlock)onError;
```

The parameters are as follows:

Parameter	Type	Meaning
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## enterRoom

Entered room.





```
- (void)enterRoom:(NSString *)roomId  
    onSuccess:(TUIRoomInfoBlock)onSuccess  
    onError:(TUIErrorBlock)onError;
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	NSString *	Room ID
onSuccess	TUIRoomInfoBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## exitRoom

Leave room.



```
- (void)exitRoom:(BOOL)syncWaiting  
    onSuccess:(TUISuccessBlock)onSuccess  
    onError:(TUIErrorBlock)onError;
```

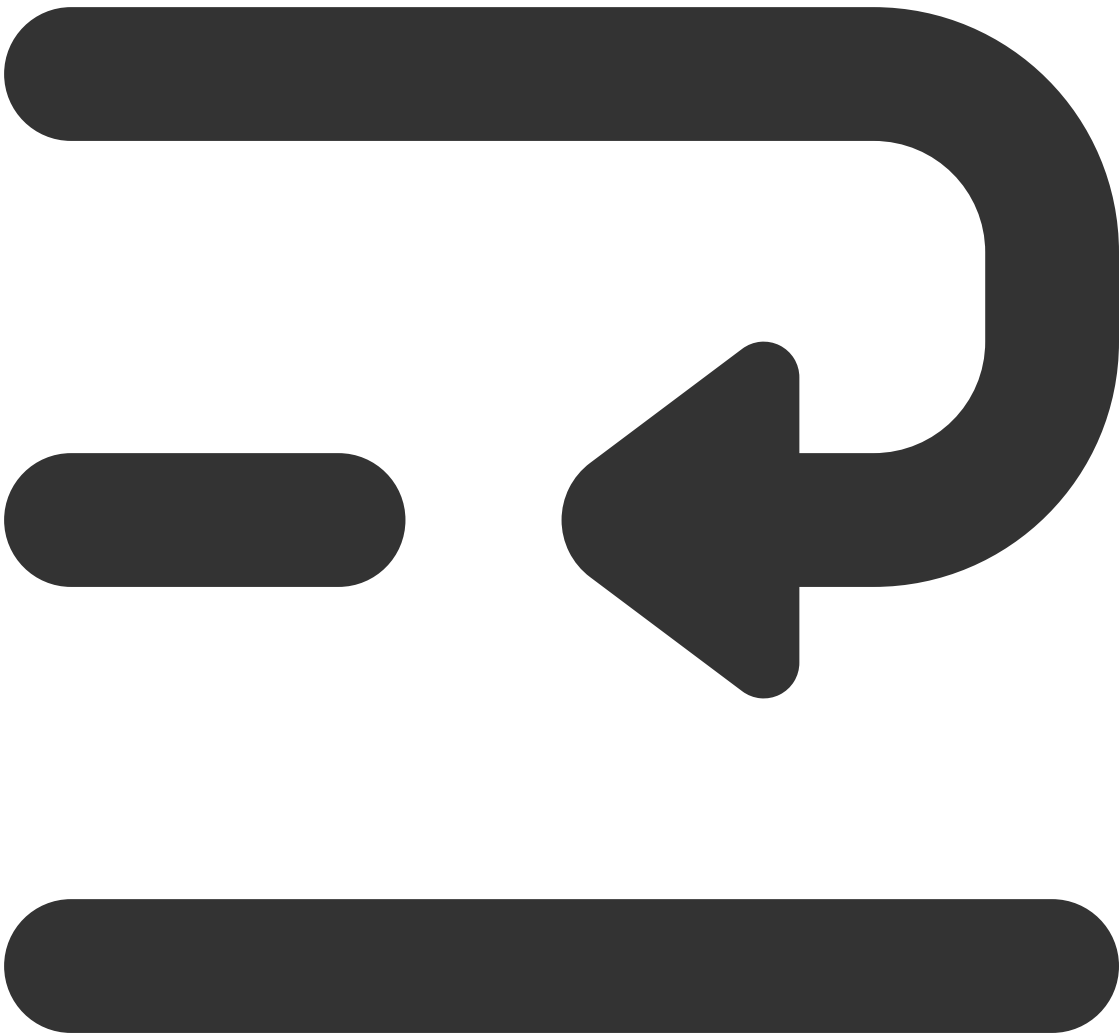
**The parameters are as follows:**

Parameter	Type	Meaning
syncWaiting	BOOL	Whether to synchronize and wait for the interface to return

onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

**connectOtherRoom**

Connect to other rooms (used for streaming scenario to apply for cross-room streaming).





```
- (TUIRequest *)connectOtherRoom:(NSString *)roomId
    userId:(NSString *)userId
    timeout:(NSTimeInterval)timeout
    onAccepted:(TUIRequestAcceptedBlock)onAccepted
    onRejected:(TUIRequestRejectedBlock)onRejected
    onCancelled:(TUIRequestCancelledBlock)onCancelled
    onTimeout:(TUIRequestTimeoutBlock)onTimeout
    onError:(TUIRequestErrorBlock)onError;
```

The parameters are as follows:

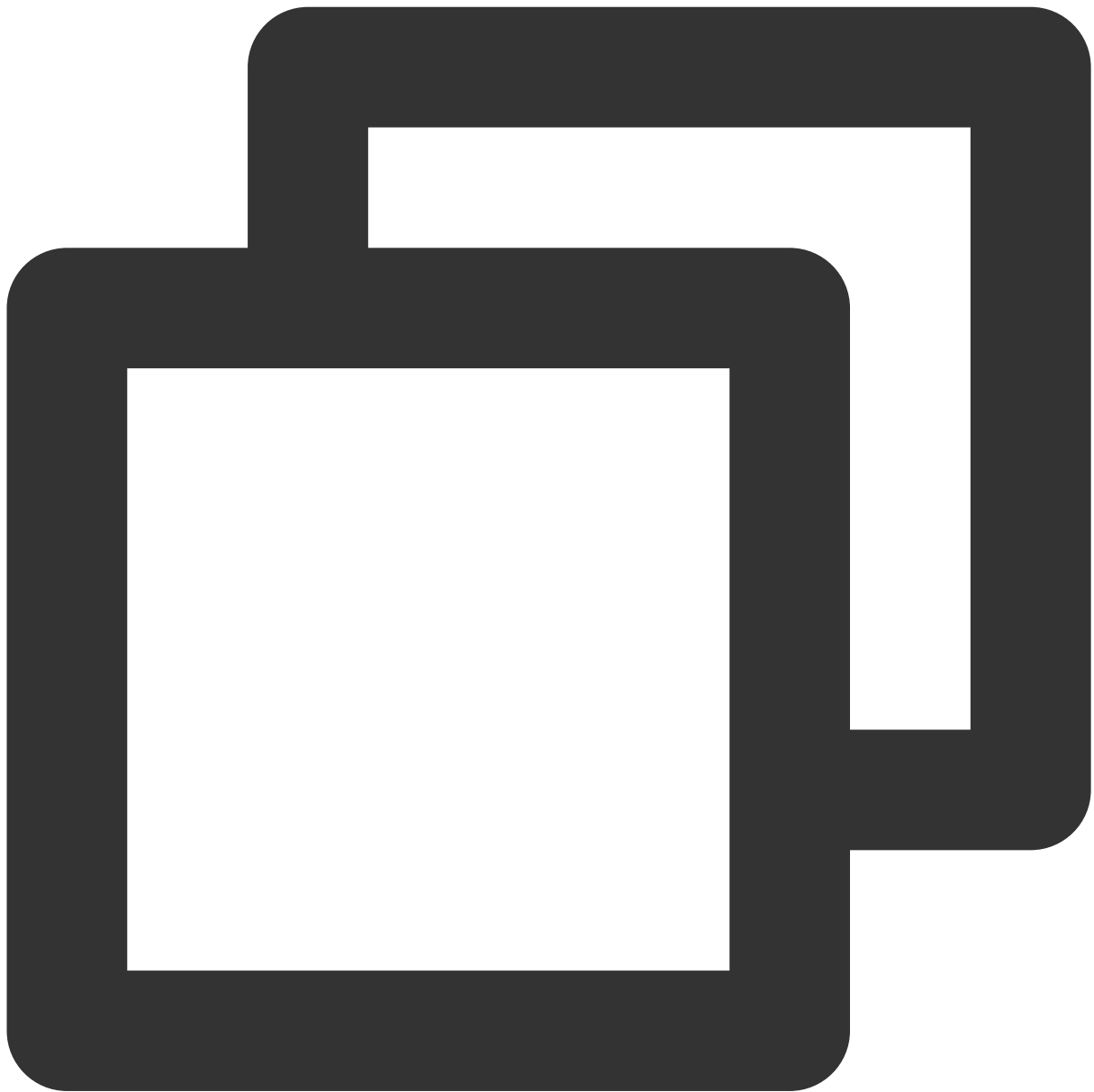
--	--	--

Parameter	Type	Meaning
roomId	NSString *	Room ID
userId	NSString *	User ID
timeout	NSTimeInterval	Timeout period (Unit seconds, if set to 0, SDK will not perform timeout detection and will not trigger timeout Callback)
onAccepted	TUIRequestAcceptedBlock	Invitation accepted Callback
onRejected	TUIRequestRejectedBlock	Invitation rejected Callback
onCancelled	TUIRequestCancelledBlock	Invitation canceled Callback
onTimeout	TUIRequestTimeoutBlock	Invitation timeout unprocessed Callback
onError	TUIRequestErrorBlock	Invitation error occurred Callback

Return value	Type	Meaning
request	TUIRequest	Request body

## disconnectOtherRoom

Disconnect from other rooms (used for disconnecting cross-room streaming in live streaming scenarios).



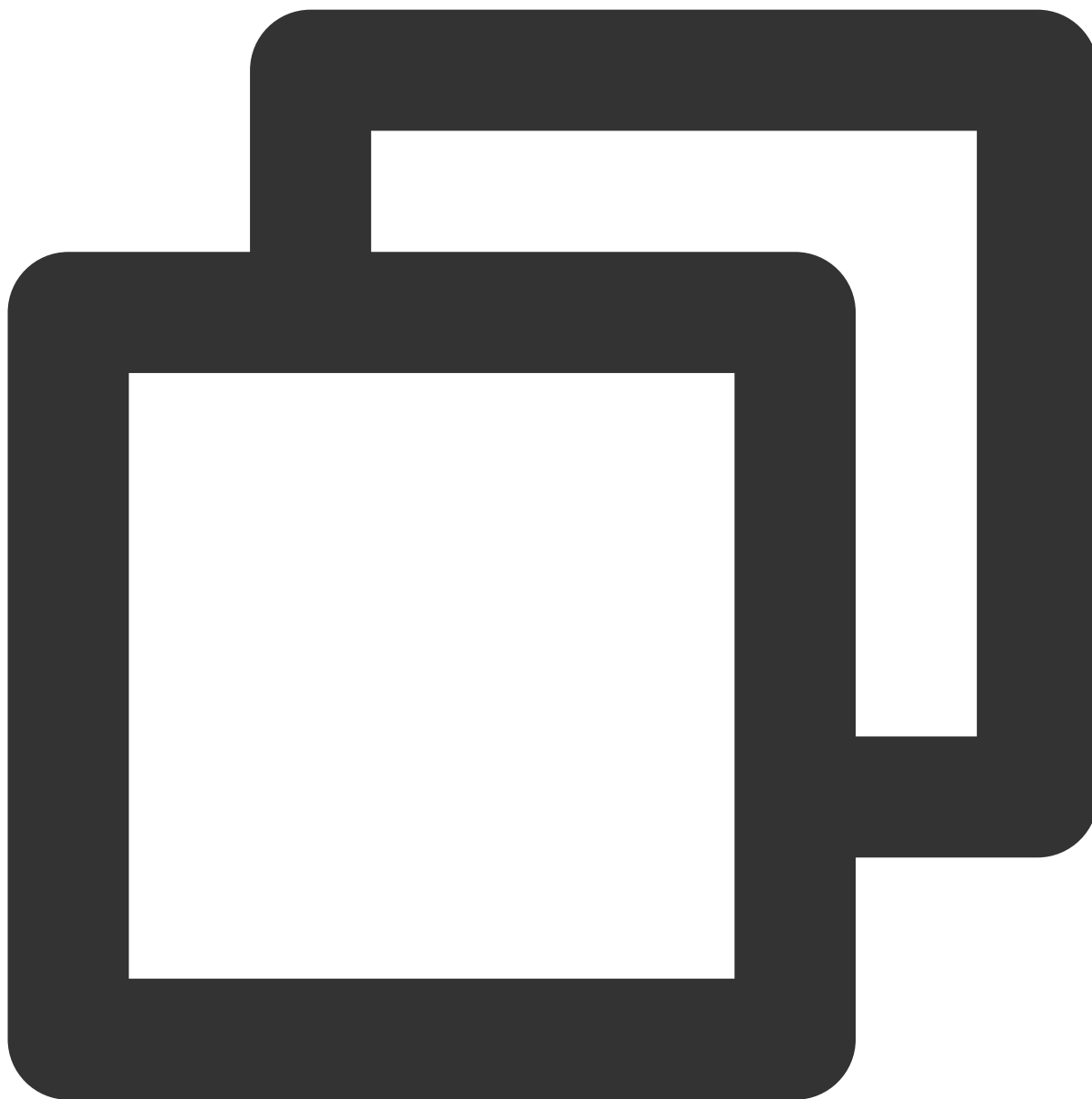
```
- (void)disconnectOtherRoom:(TUISuccessBlock)onSuccess onError:(TUIErrorBlock)onErr
```

The parameters are as follows:

Parameter	Type	Meaning
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## fetchRoomInfo

Get Room data.



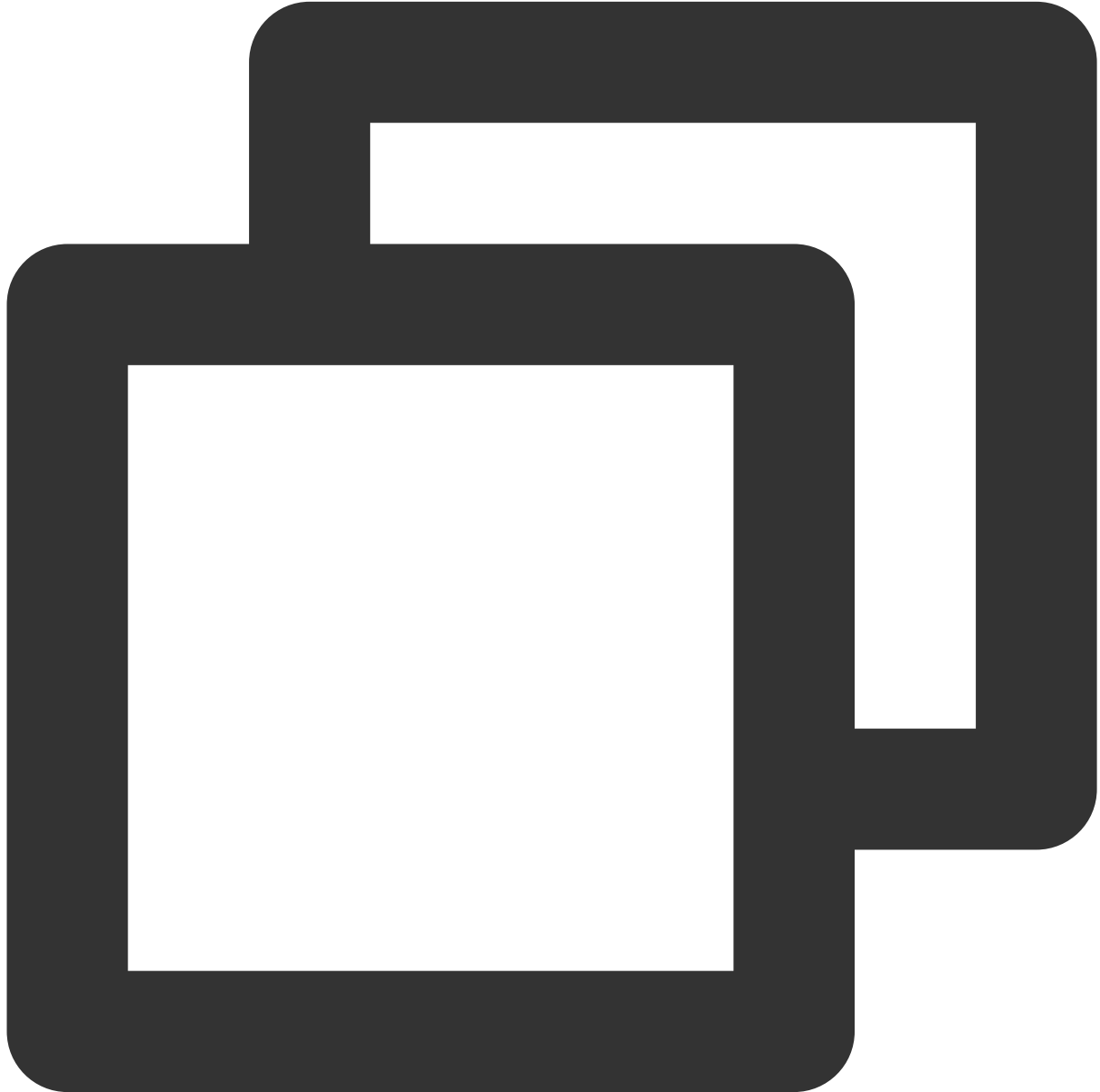
```
- (void)fetchRoomInfo:(TUIRoomInfoBlock)onSuccess onError:(TUIErrorBlock)onError;
```

The parameters are as follows:

Parameter	Type	Meaning
onSuccess	TUIRoomInfoBlock	Success Callback with <a href="#">TUIRoomInfo</a> Room data
onError	TUIErrorBlock	Failed callback

## updateRoomNameByAdmin

Update Room ID (Only Administrator or Group owner can call).



```
- (void)updateRoomNameByAdmin:(NSString *)roomName
    onSuccess:(TUISuccessBlock)onSuccess
    onError:(TUIErrorBlock)onError;
```

The parameters are as follows:

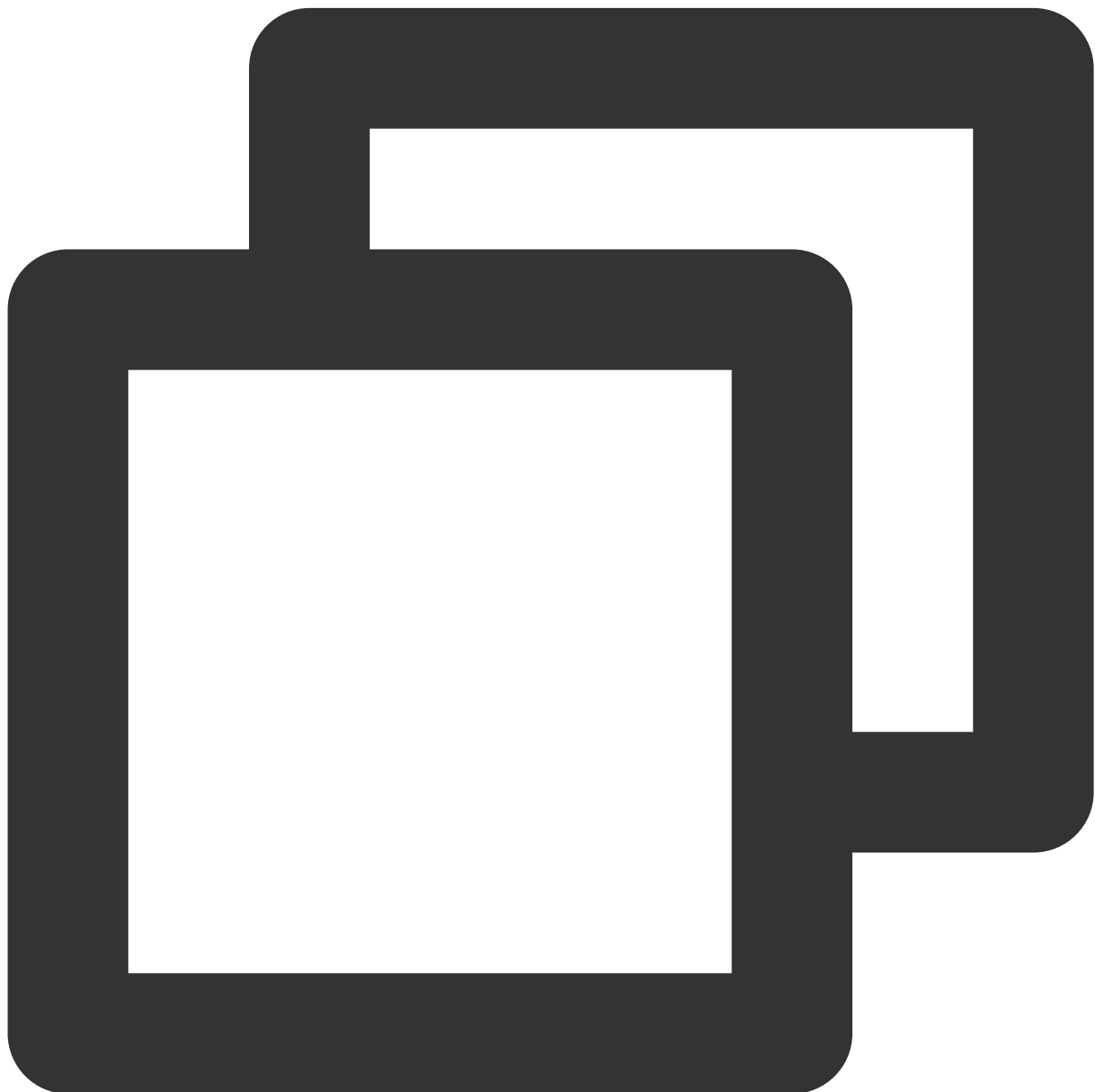
Parameter	Type	Meaning



roomName	NSString *	Room Name
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## updateRoomSpeechModeByAdmin

Set up the room's mic control mode (only administrators or group owners can call).



```
- (void)updateRoomSpeechModeByAdmin:(TUISpeechMode)mode
```

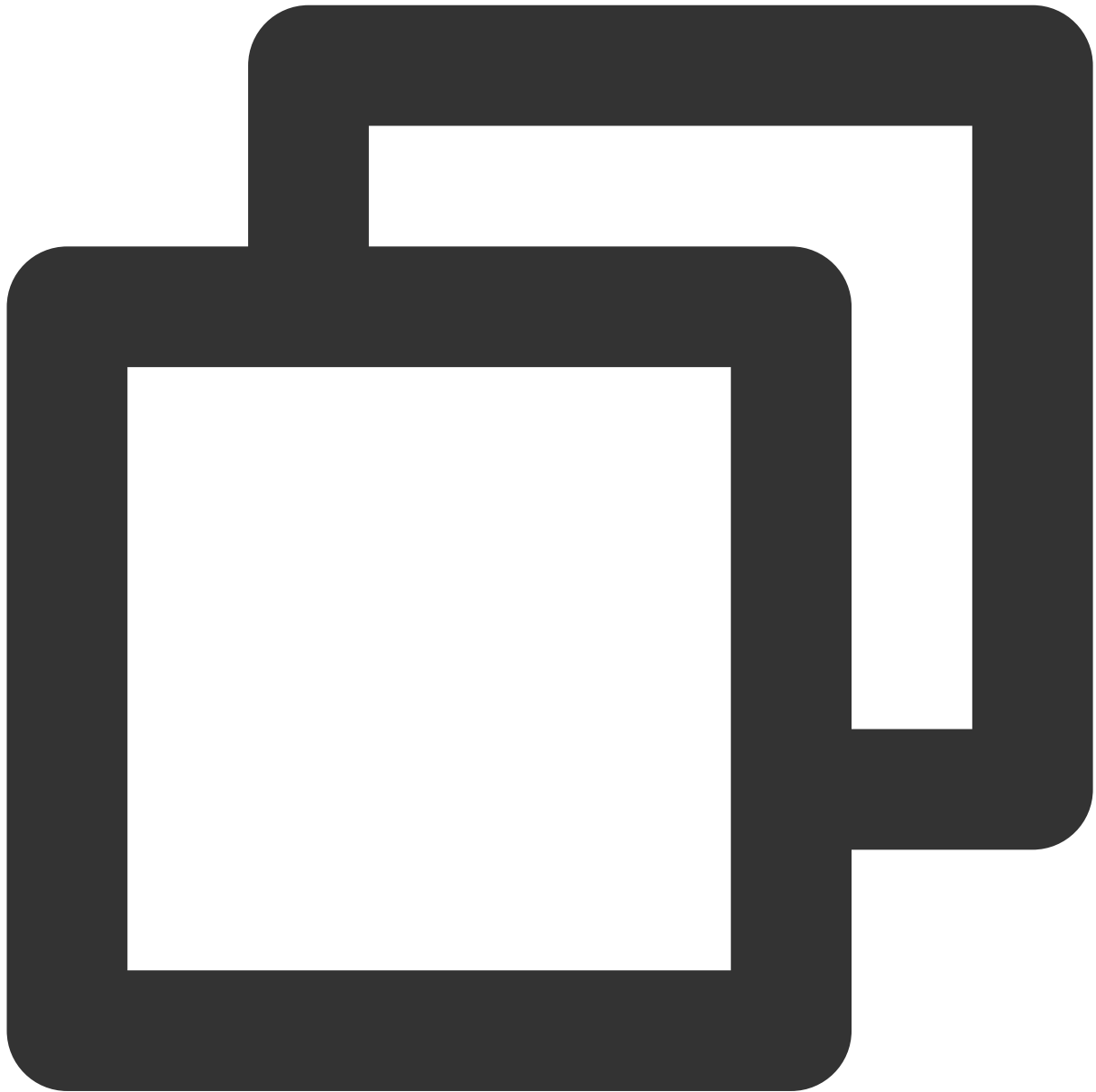
```
onSuccess: (TUISuccessBlock) onSuccess  
onError: (TUIErrorBlock) onError;
```

The parameters are as follows:

Parameter	Type	Meaning
mode	<a href="#">TUISpeechMode</a>	Speech Mode, detailed Definition can be found in TUIRoomDefine.h file's <a href="#">TUISpeechMode</a> .
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## setLocalVideoView

Set Local User Video Rendering Control.



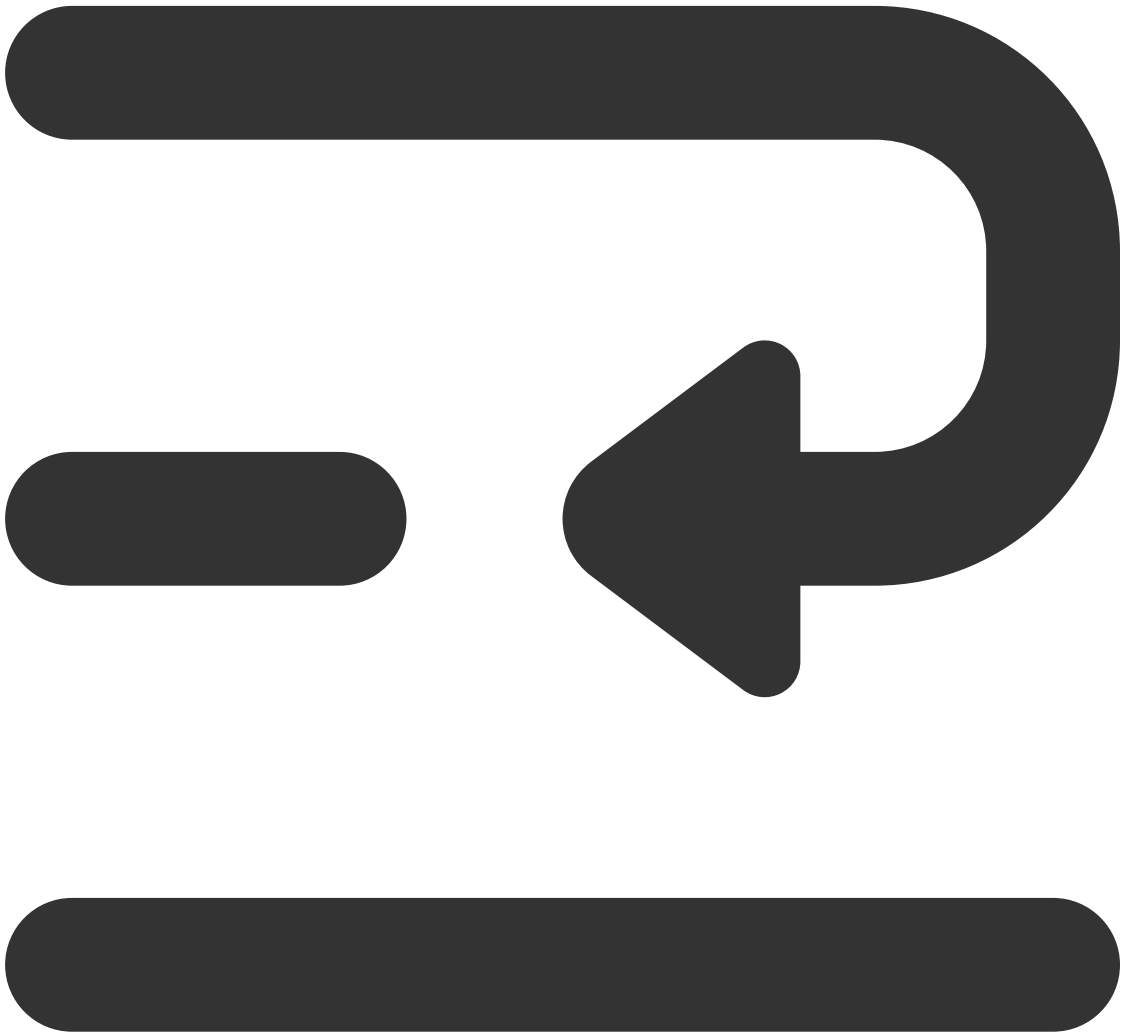
```
- (void)setLocalVideoView:(TUIVideoStreamType)streamType view:(TUIVideoView *)view;
```

The parameters are as follows:

Parameter	Type	Meaning
streamType	<a href="#">TUIVideoStreamType</a>	Video streams Type, detailed Definition can be found in TUIRoomDefine.h file's <a href="#">TUIVideoStreamType</a> .
view	TUIVideoView *	Video Rendering View

## setRemoteVideoView

Set Remote User Video Rendering Control.





```
- (void)setRemoteVideoView:(NSString *)userId
      streamType:(TUIVideoStreamType) streamType
      view:(TUIVideoView * __nullable) view;
```

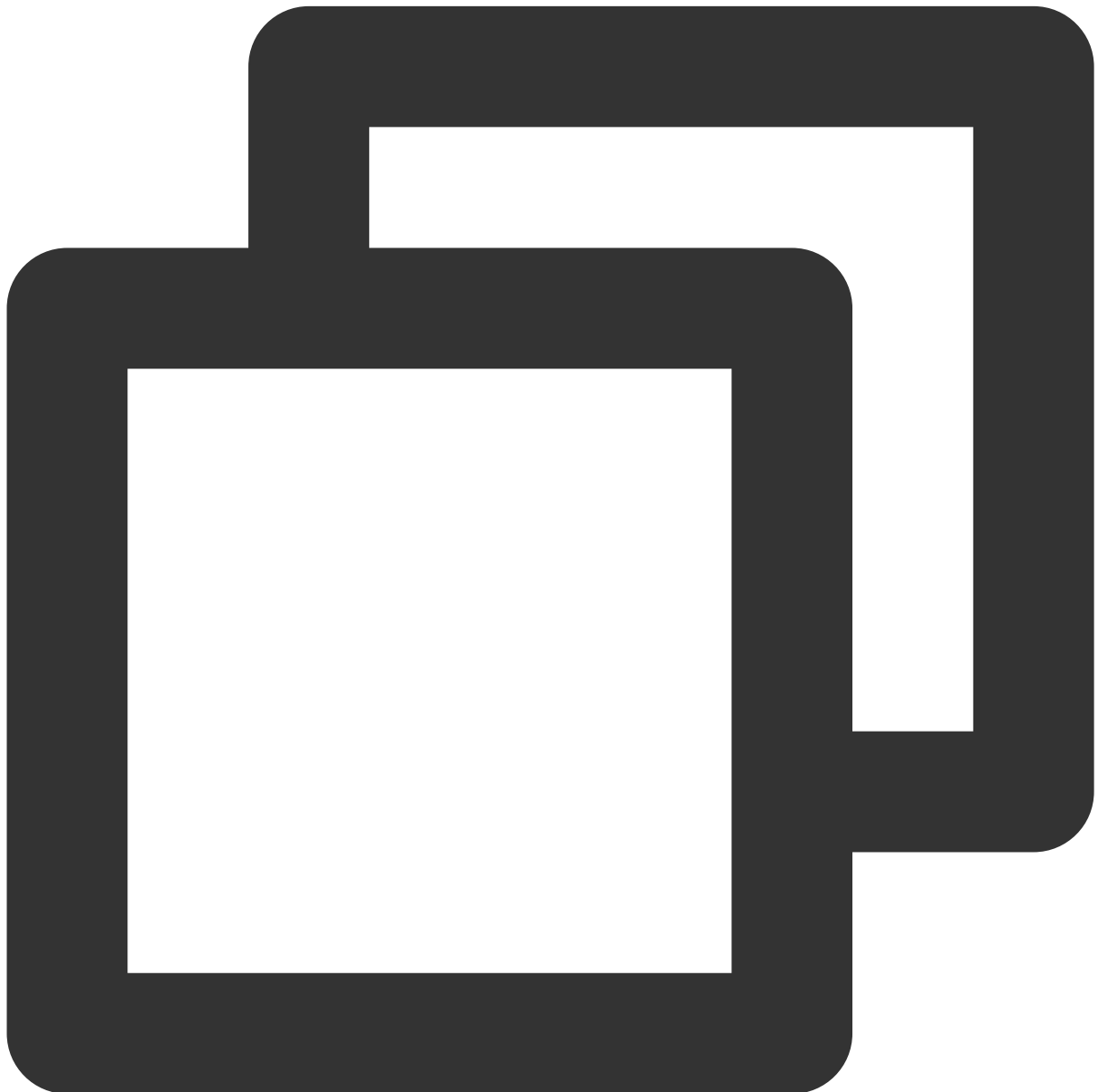
The parameters are as follows:

Parameter	Type	Meaning
userId	NSString	Remote User ID
streamType	<a href="#">TUIVideoStreamType</a>	Video streams Type, detailed Definition can be found in

		TUIRoomDefine.h file's TUIVideoStreamType.
view	TUIVideoView *	Video Rendering View

## openLocalCamera

Open Local Camera.



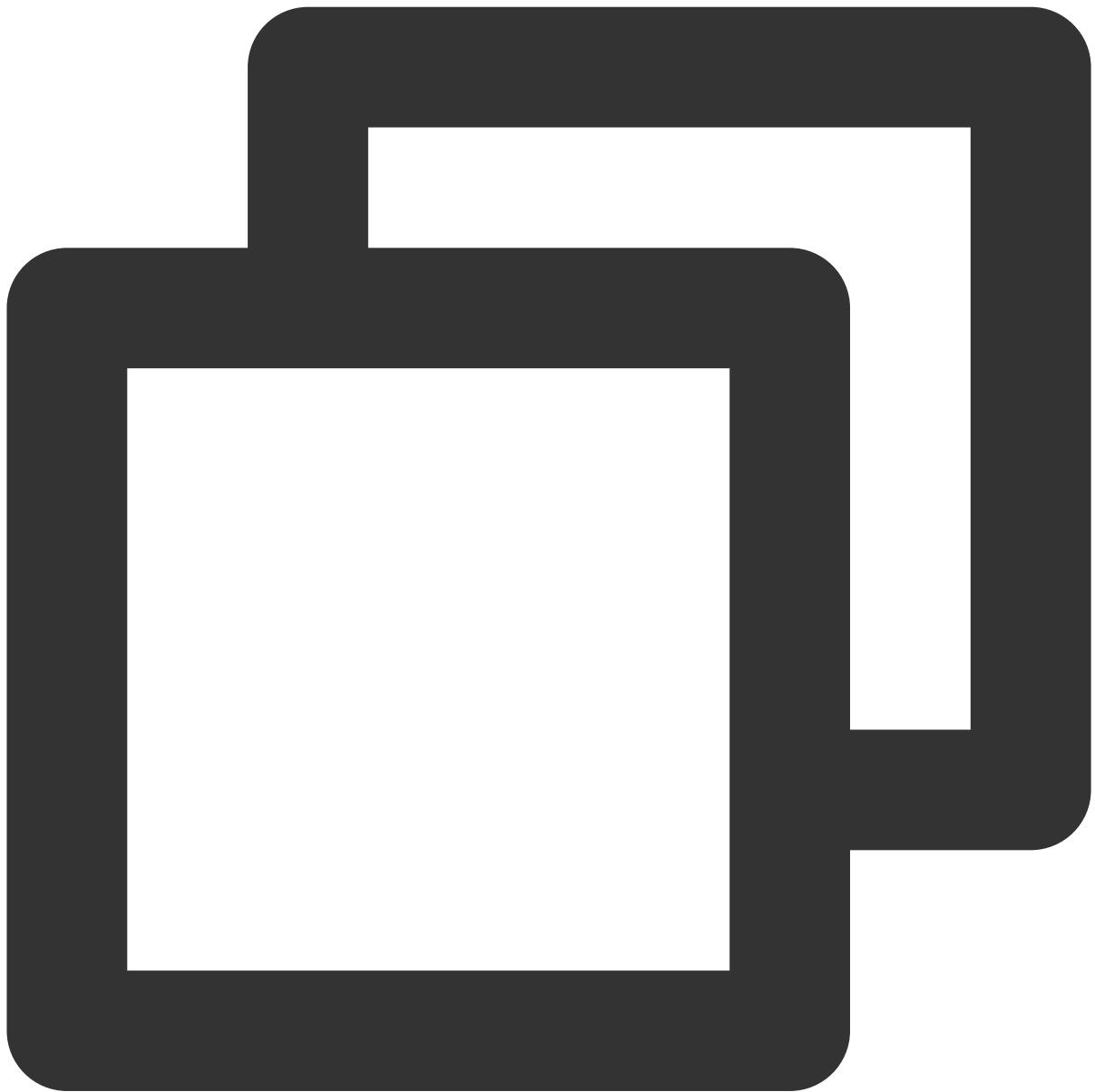
```
- (void)openLocalCamera:(BOOL)isFront  
    quality:(TUIVideoQuality)quality
```

```
onSuccess:(TUISuccessBlock) onSuccess  
onError:(TUIErrorBlock) onError;
```

Parameter	Type	Meaning
isFront	BOOL	Front or not
quality	<a href="#">TUIVideoQuality</a>	Video Quality, detailed Definition can be found in TUIRoomDefine.h file's <a href="#">TUIVideoQuality</a> .
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## closeLocalCamera

Close Local Camera.

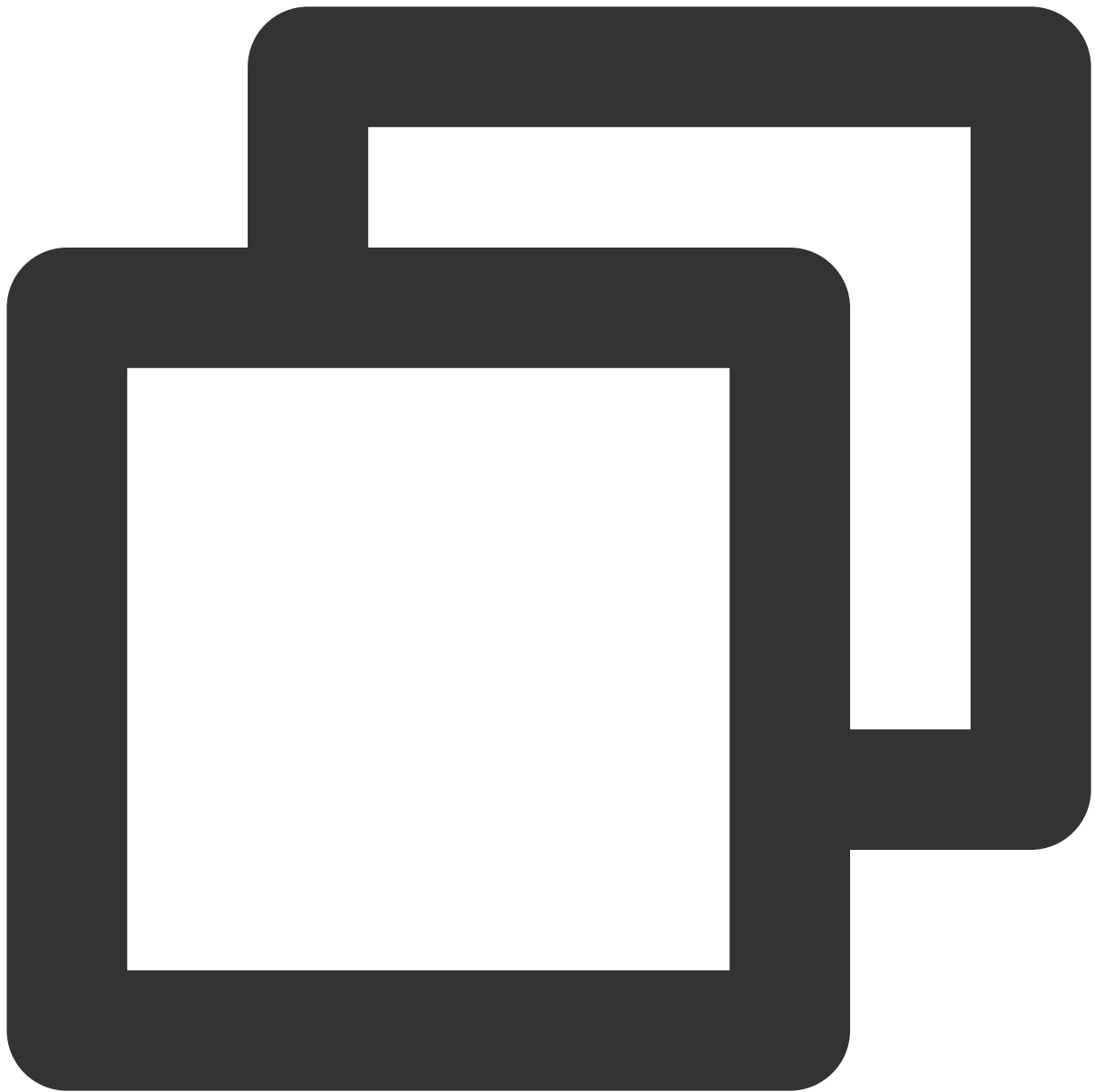


```
- (void)closeLocalCamera;
```

### **updateVideoQuality**

Update Local Video Codec Quality Setting.





```
- (void)updateVideoQuality:(TUIVideoQuality)quality;
```

Parameter	Type	Meaning
quality	<a href="#">TUIVideoQuality</a>	Video Quality

## openLocalMicrophone

Open Local mic.



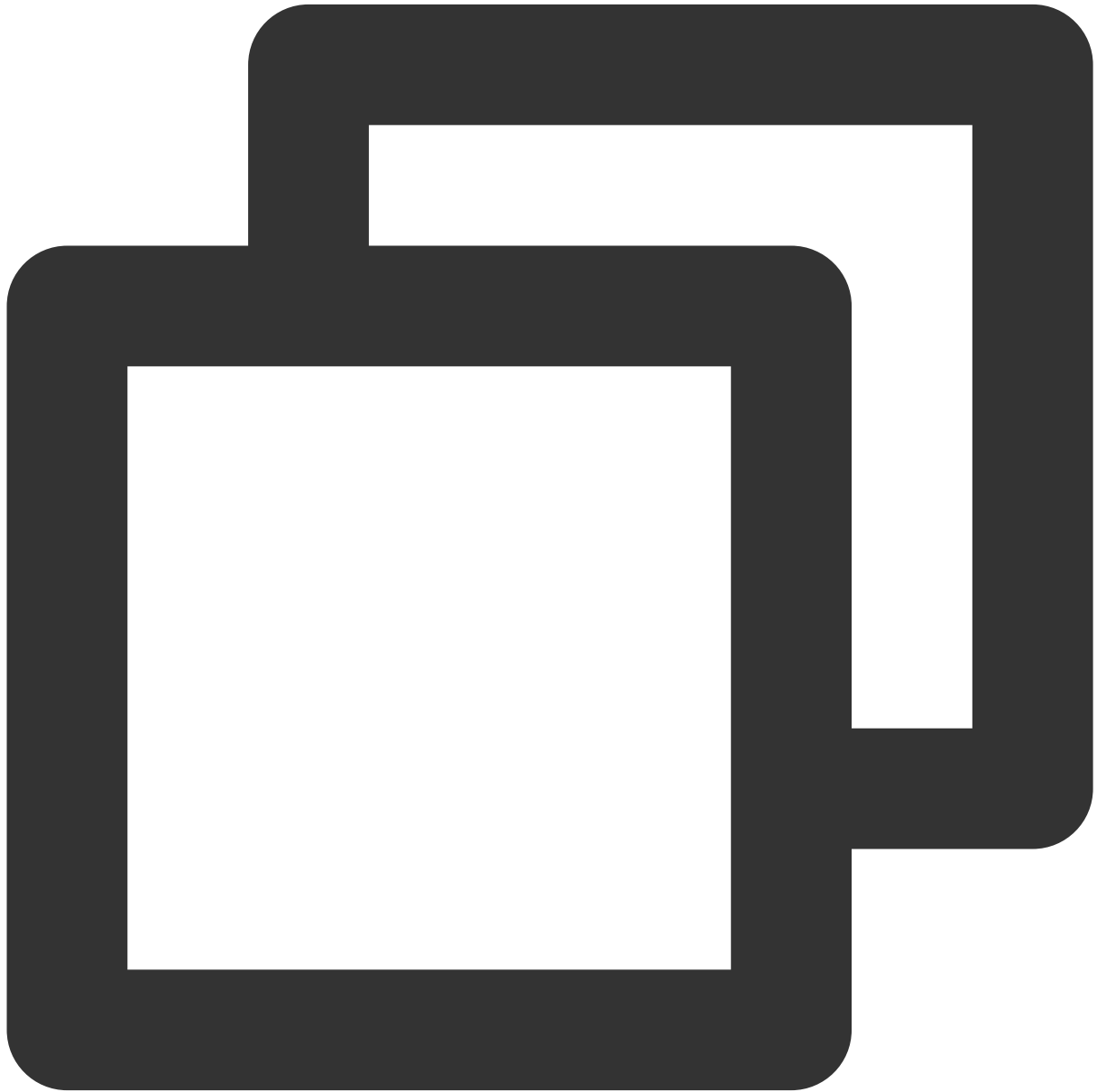
```
- (void)openLocalMicrophone:(TUIAudioQuality) quality  
    onSuccess:(TUISuccessBlock) onSuccess  
    onError:(TUIErrorBlock) onError;
```

Parameter	Type	Meaning
quality	<a href="#">TUIAudioQuality</a>	Audio Quality, detailed Definition can be found in TUIRoomDefine.h file's <a href="#">TUIAudioQuality</a>
onSuccess	TUISuccessBlock	Successful callback

onError	TUIErrorBlock	Failed callback
---------	---------------	-----------------

## closeLocalMicrophone

Close Local mic.



```
- (void)closeLocalMicrophone;
```

## updateAudioQuality

Update Local Audio Codec Quality Setting.



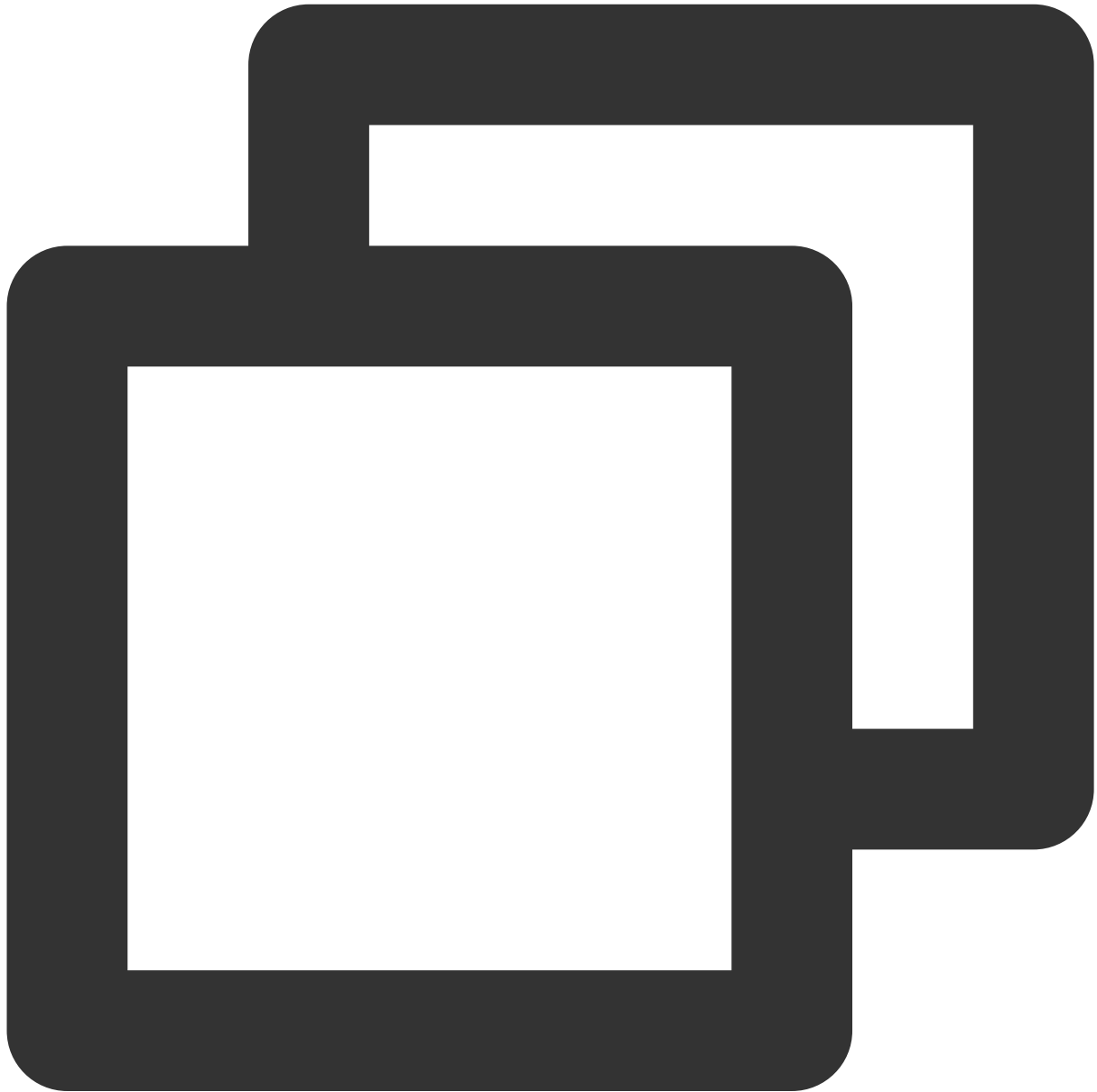
```
- (void)updateAudioQuality:(TUIAudioQuality)quality;
```

Parameter	Type	Meaning
quality	<a href="#">TUIAudioQuality</a>	Audio Quality

## startScreenCapture

Start Screen Sharing (Only supports mobile iOS 11.0 and above systems)

This interface supports capturing the entire iOS system screen, similar to Tencent Meeting's full system-level Screen Sharing.



```
- (void)startScreenCaptureByReplaykit:(NSString *)appGroup API_AVAILABLE(ios(11.0))
```

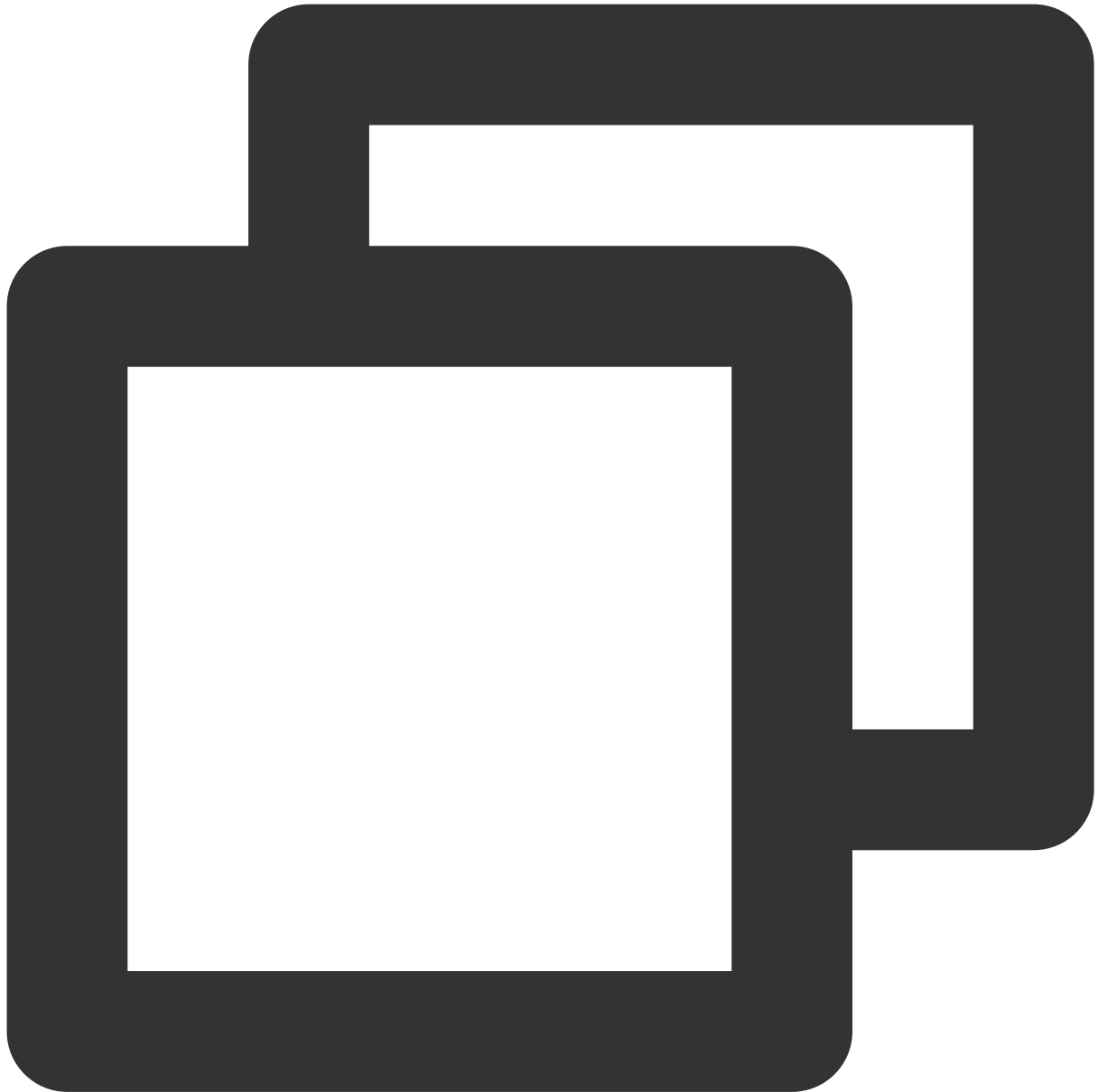
**The parameters are as follows:**

Parameter	Type	Meaning
appGroup	NSString *	Specify the Application Group Identifier shared by your app and the Screen recording Process, you can set this parameter to nil, but it is recommended to set it according to the documentation for better Reliability.

## startScreenCapture

Start Screen Sharing (This interface only supports desktop Mac OS systems)

This interface can capture the entire Mac OS system screen or the window content of a specified app and share it with other users in the same room.



```
- (void)startScreenCapture:(TUIVideoView *)view
    onSuccess:(TUISuccessBlock)onSuccess
    onError:(TUIErrorBlock)onError;
```

The parameters are as follows:

Parameter	Type	Meaning
view	TUIVideoView *	Parent Control of the Rendering Control, can be set to null, indicating no preview effect of Screen Sharing.
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## stopScreenCapture

End Screen Sharing.



```
- (void)stopScreenCapture;
```

## getScreenCaptureSources

Enumerate available screens and windows for sharing (This interface only supports Mac OS systems)

When you integrate the desktop system's Screen Sharing function, you generally need to display a Target selection Interface so that users can use this Interface to choose whether to share the entire screen or a specific window.

Through this interface, you can query the ID, name, and thumbnail of the windows available for sharing in the current system. We provide a default Interface implementation in the Demo for your reference.





```
- (NSArray<TUIShareTarget *> *)getScreenCaptureSources;
```

Return Value	Type	Meaning
screenCaptureSources	NSArray< <a href="#">TUIShareTarget</a> *> *	Window List including screens

### **selectScreenCaptureTarget**

Select Screen Sharing Target.



```
- (void)selectScreenCaptureTarget:(NSString *)targetId;
```

The parameters are as follows:

Parameter	Type	Meaning
targetId	NSString *	Designated Sharing Source

## startPushLocalVideo

Start Pushing Local Video.



```
- (void)startPushLocalVideo;
```

## **stopPushLocalVideo**

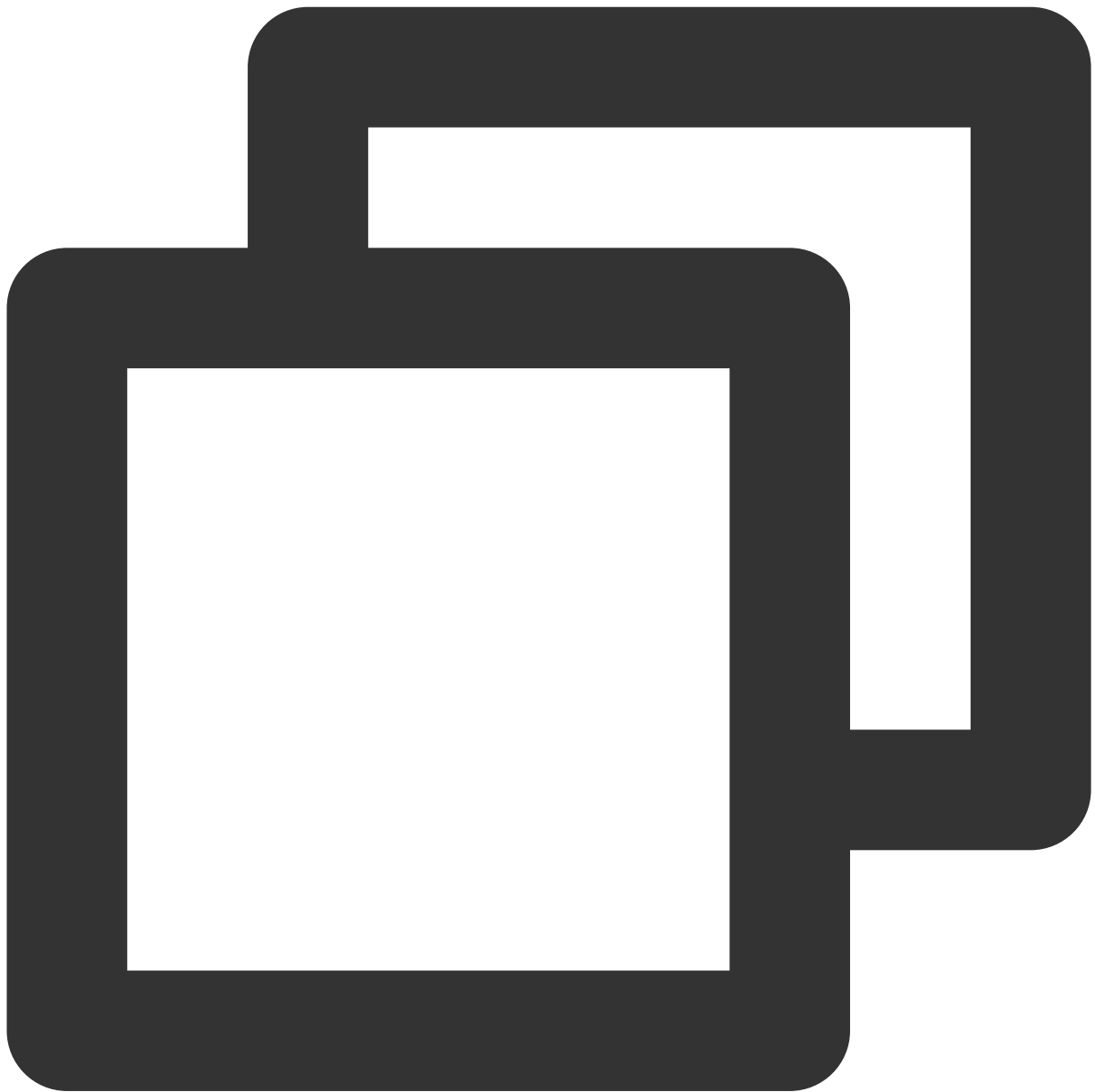
Stop Pushing Local Video.



```
- (void)stopPushLocalVideo;
```

### **startPushLocalAudio**

Start Pushing Local Audio.



```
- (void)startPushLocalAudio;
```

## **stopPushLocalAudio**

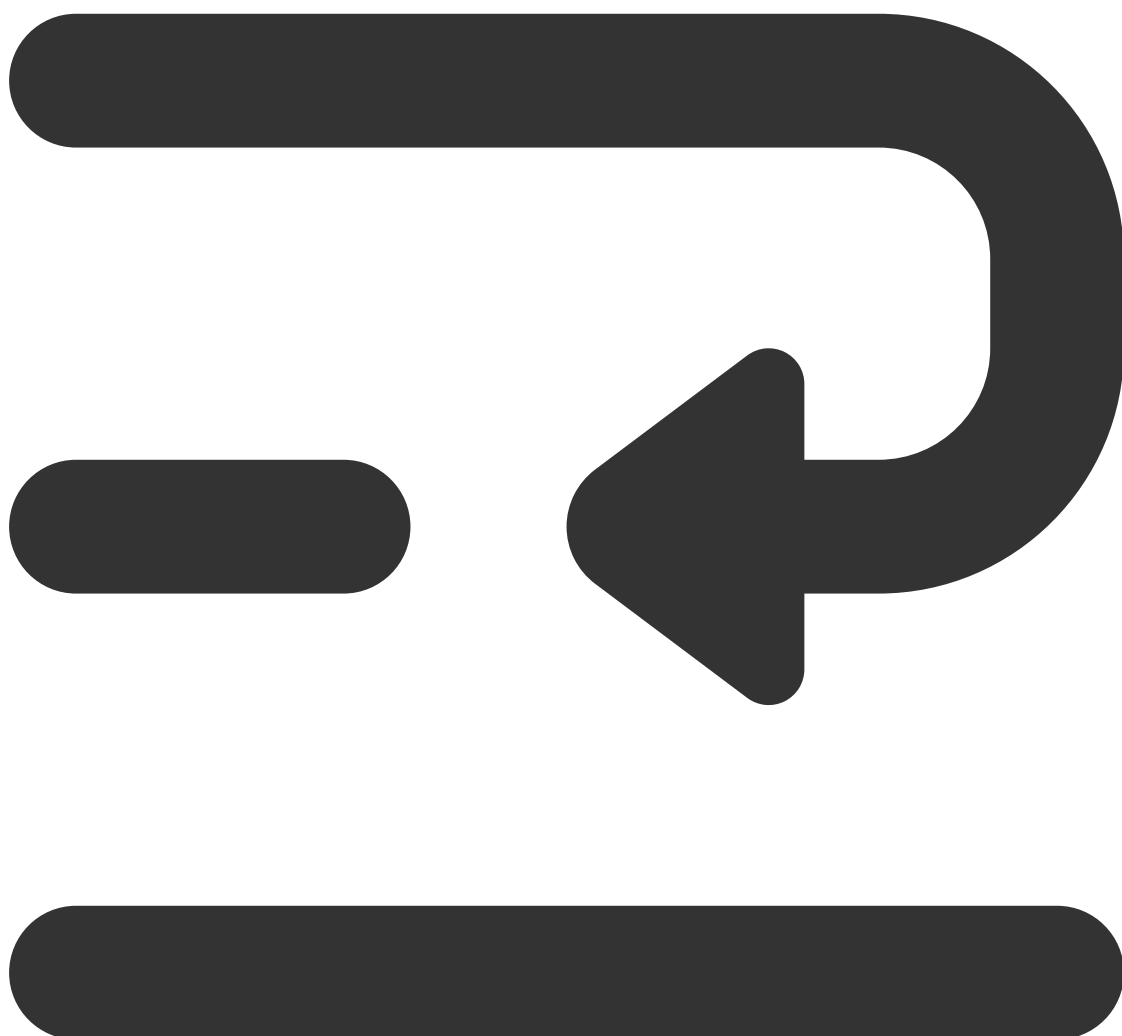
Stop Pushing Local Audio.

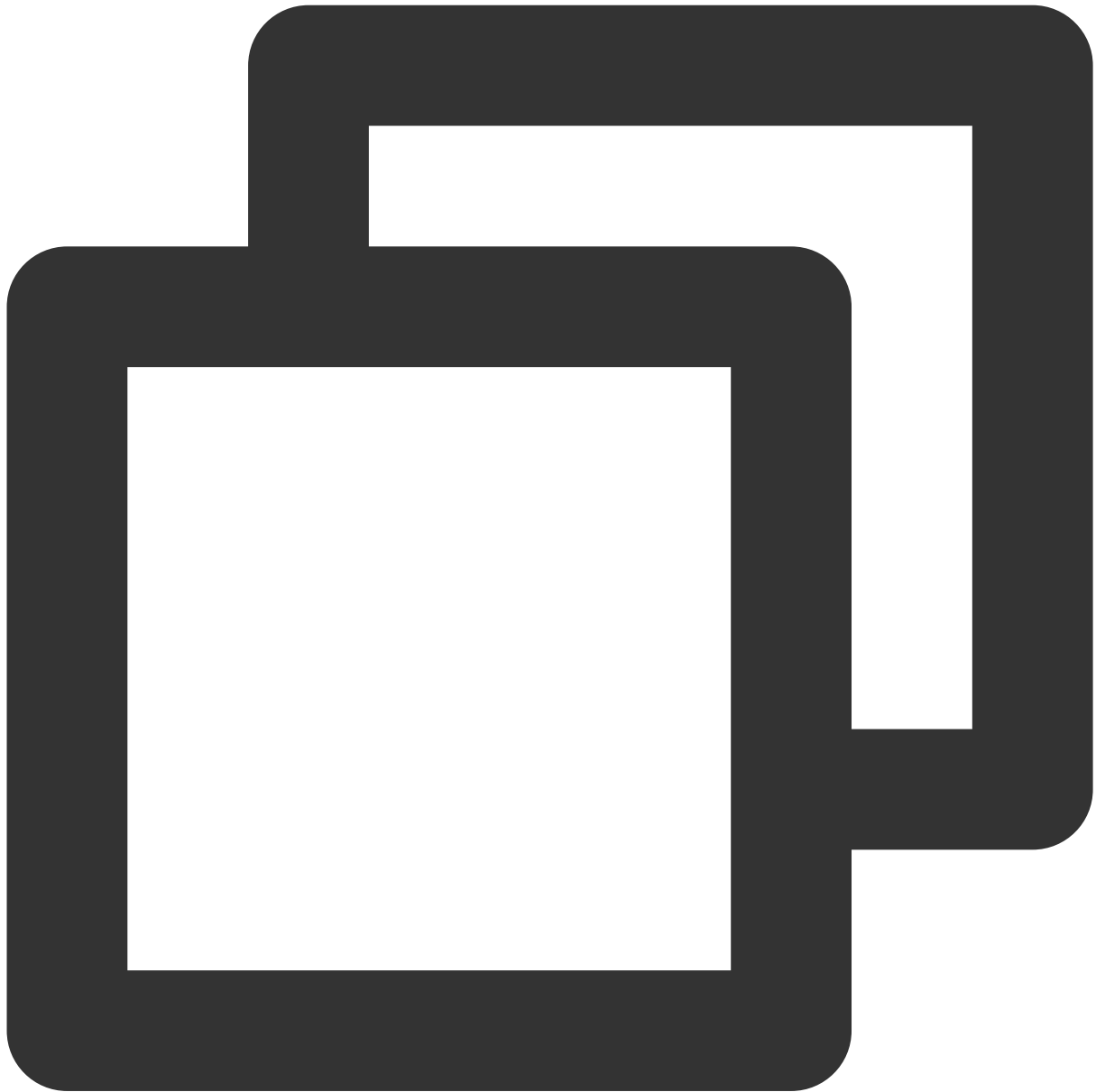


```
- (void)stopPushLocalAudio;
```

### **startPlayRemoteVideo**

Start Playback Remote User Video.





```
- (void)startPlayRemoteVideo:(NSString *)userId
    streamType:(TUIVideoStreamType)streamType
    onPlaying:(TUIPlayOnPlayingBlock)onPlaying
    onLoading:(TUIPlayOnLoadingBlock)onLoading
    onError:(TUIPlayOnErrorBlock)onError;
```

**The parameters are as follows:**

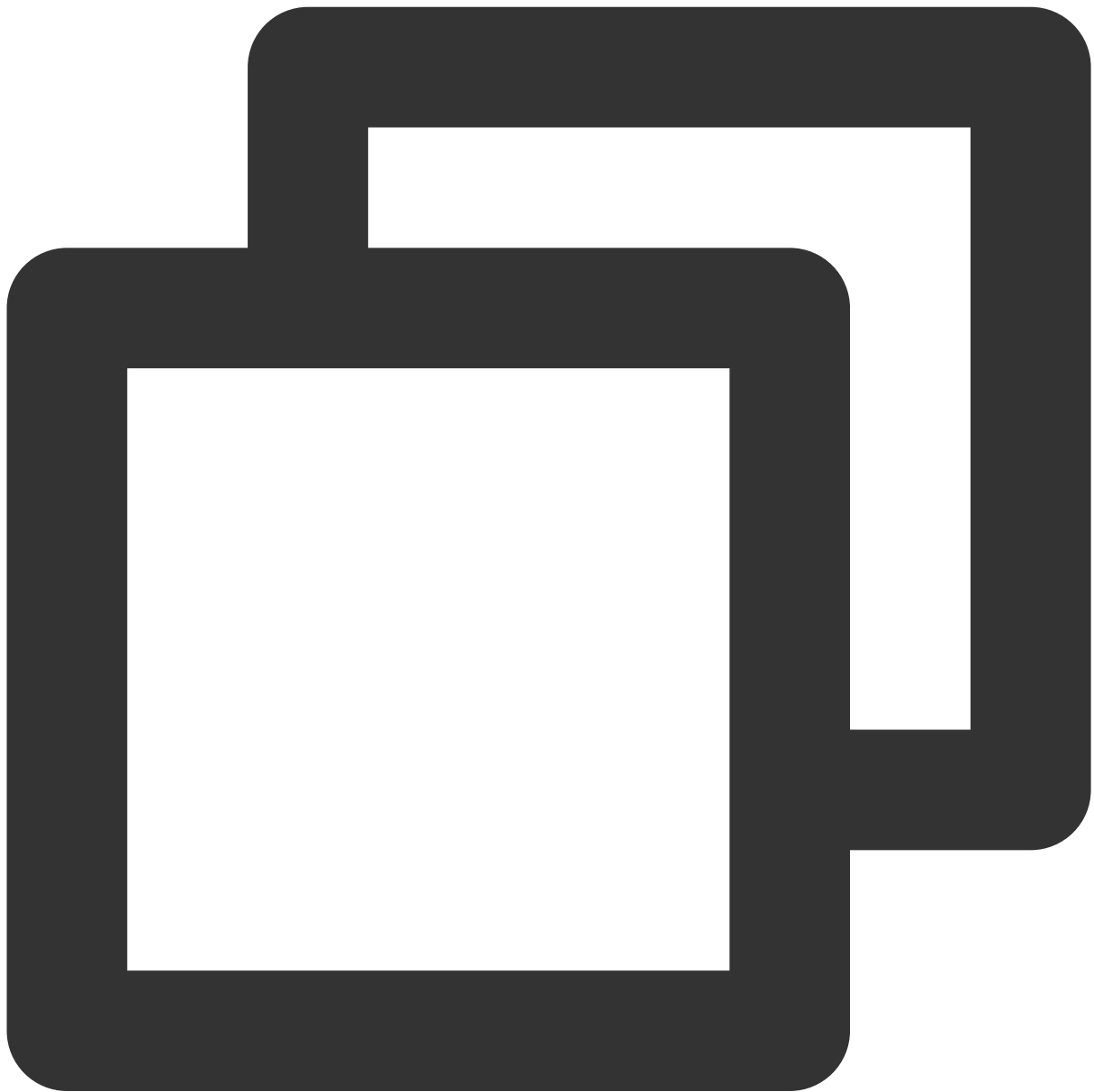
Parameter	Type	Meaning
userId	NSString *	User ID



streamType	<a href="#">TUIVideoStreamType</a>	Type of Video streams. Detailed Definition can be found in TUIRoomDefine.h file's <a href="#">TUIVideoStreamType</a> .
onPlaying	TUIPlayOnPlayingBlock	Playback Callback
onLoading	TUIPlayOnLoadingBlock	Load Callback
onError	TUIPlayOnErrorBlock	Error Callback

## stopPlayRemoteVideo

Stop Playback Remote User Video.



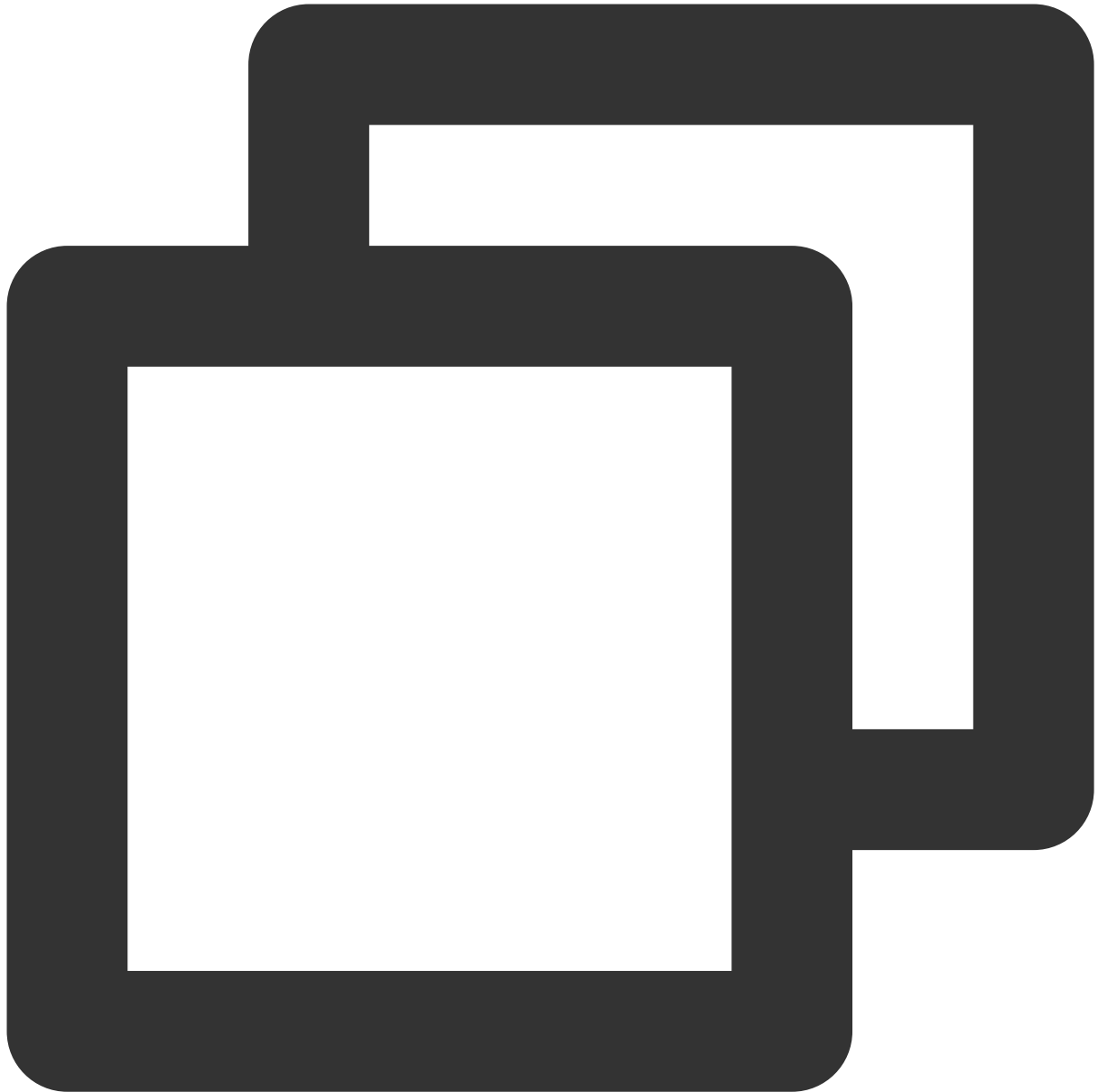
```
- (void)stopPlayRemoteVideo:(NSString *)userId streamType:(TUIVideoStreamType)streamType
```

The parameters are as follows:

Parameter	Type	Meaning
userId	NSString *	User ID
streamType	<a href="#">TUIVideoStreamType</a>	Type of Video streams. Detailed Definition can be found in TUIRoomDefine.h file's <a href="#">TUIVideoStreamType</a> .

## **muteRemoteAudioStream**

Mute Remote User

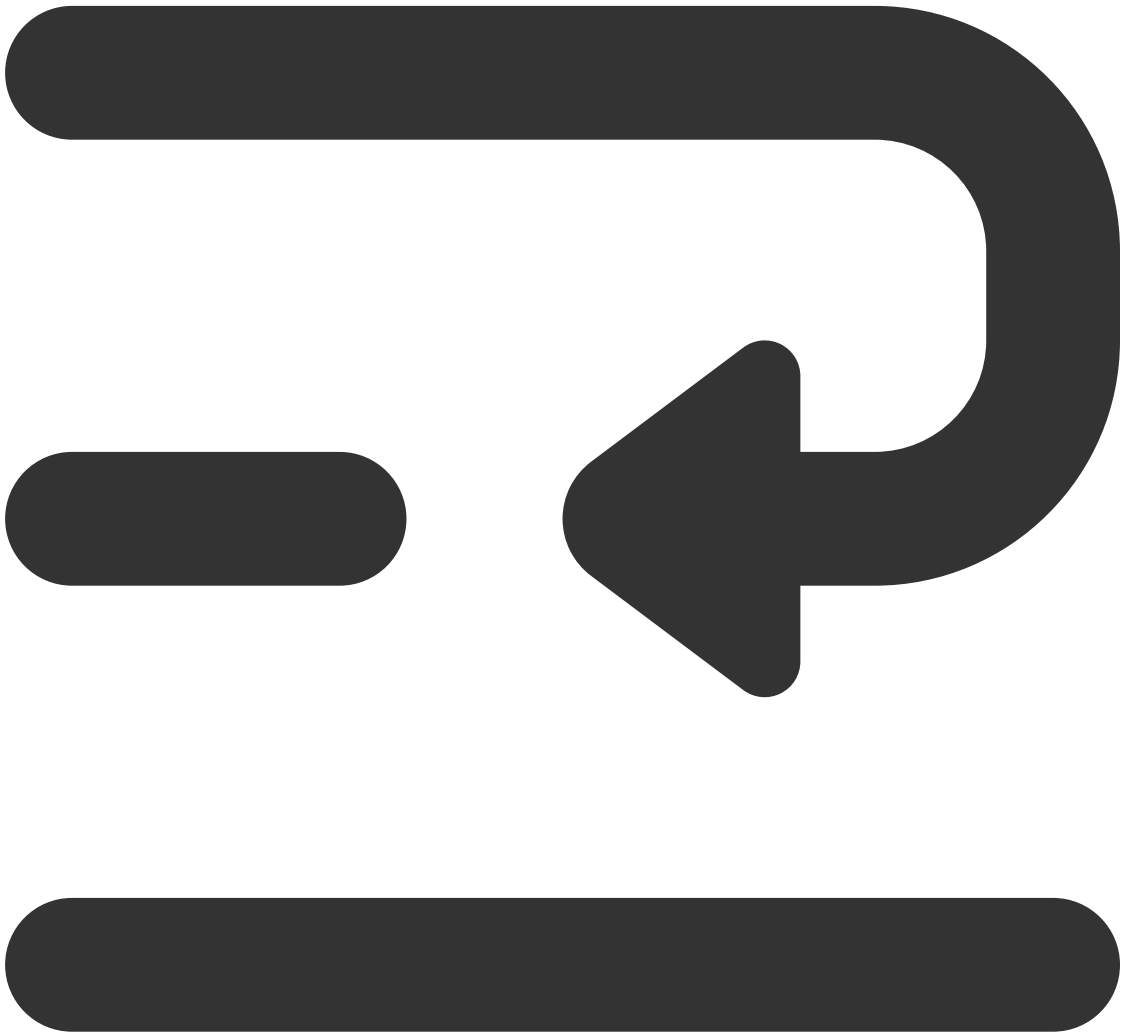


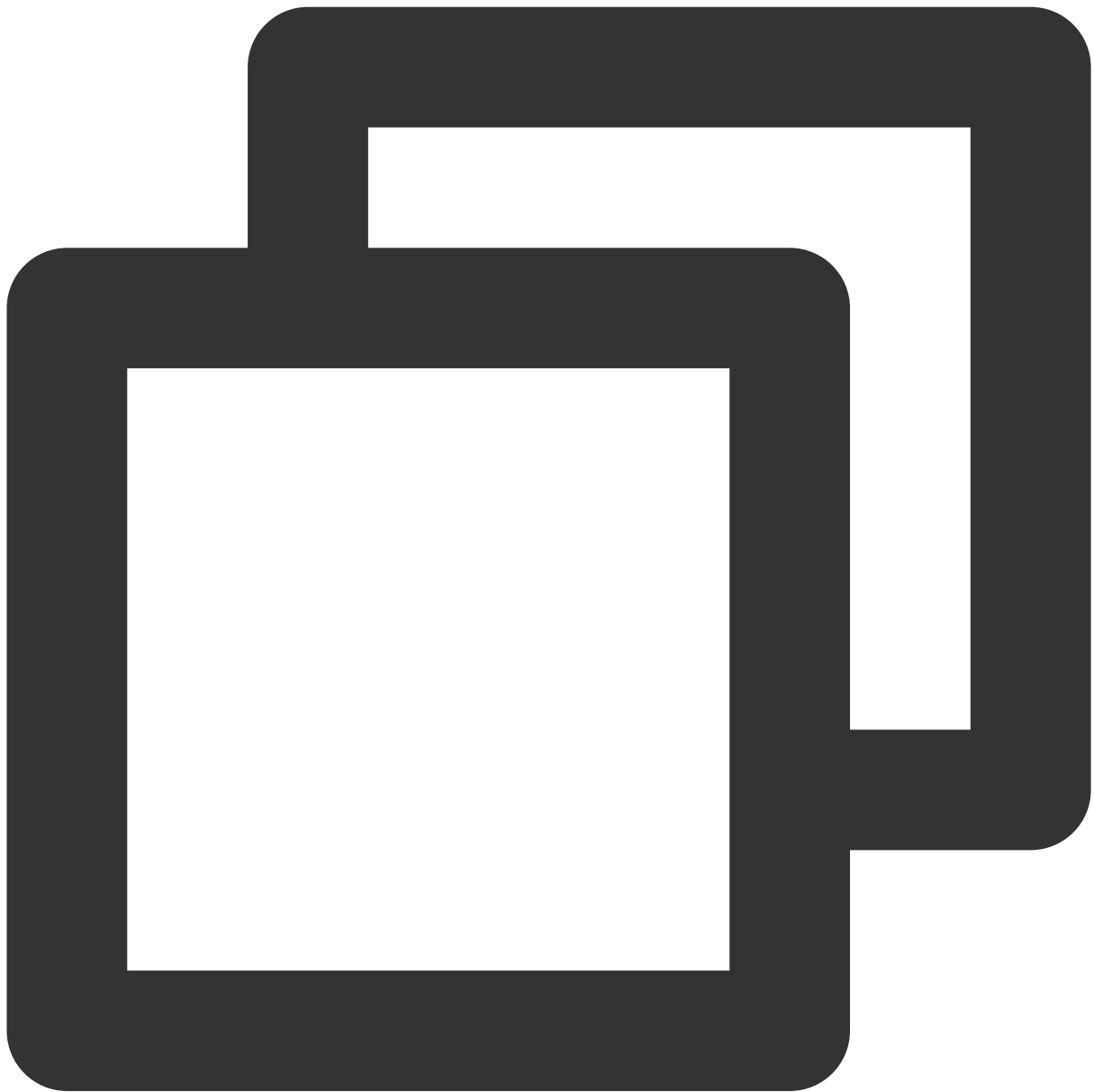
```
- (void)muteRemoteAudioStream:(NSString *)userId isMute:(BOOL)isMute;
```

Parameter	Type	Meaning
userId	NSString *	User ID
isMute	BOOL	Mute or not

## getUserList

Get Member List in the Room.





```
- (void)getUserList:(NSInteger)nextSequence
    onSuccess:(TUIUserListResponseBlock) onSuccess
    onError:(TUIErrorBlock) onError;
```

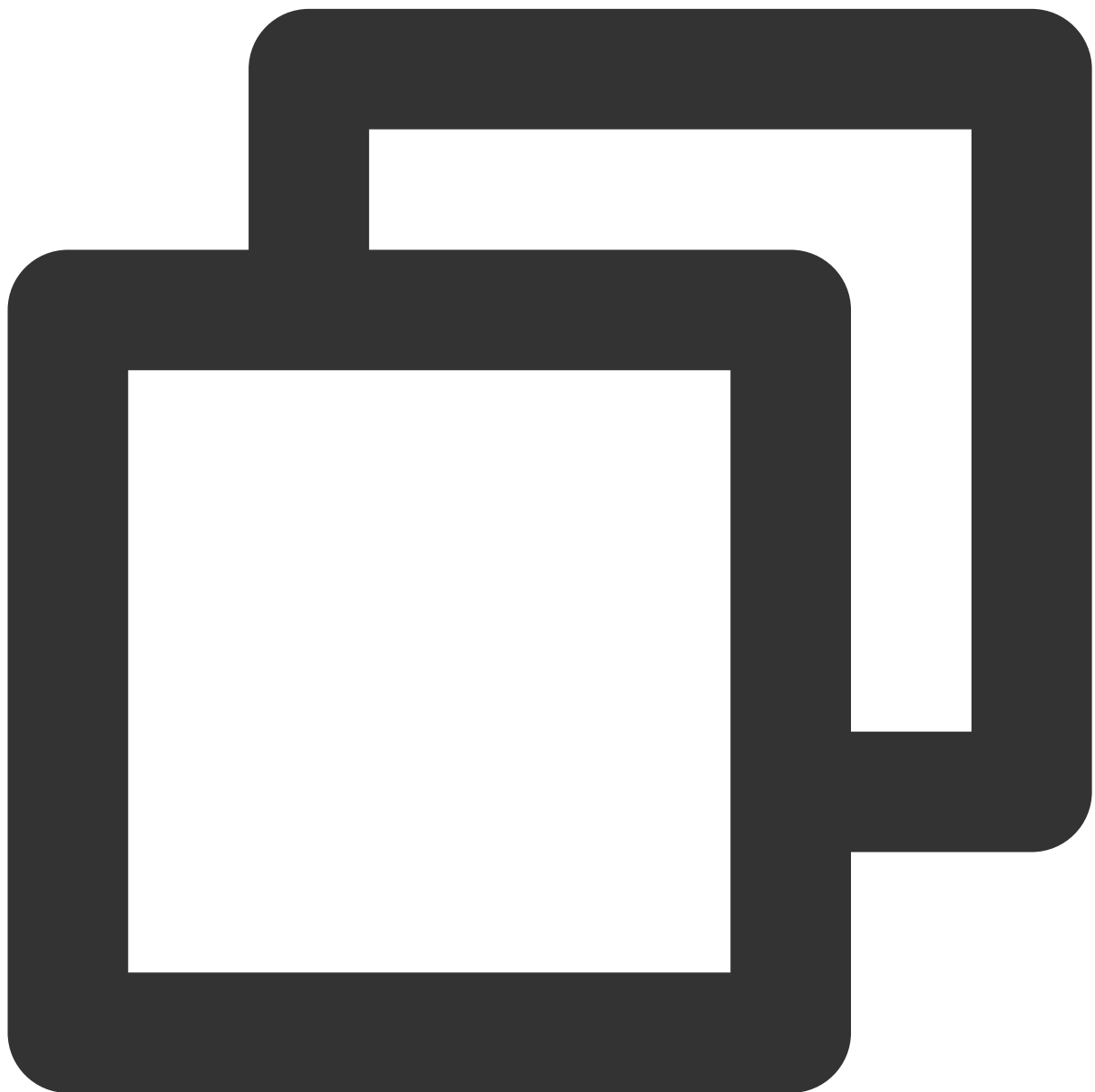
**The parameters are as follows:**

Parameter	Type	Meaning
nextSequence	NSInteger	Pagination Fetch flag, fill in 0 for the first Fetch, if the Callback returns TUIUserListResult with next_sequence not equal to zero, Pagination is required, pass in again to Fetch until it is 0

onSuccess	TUIUserListResponseBlock	Success Callback. Detailed Definition can be found in TUIRoomDefine.h file's TUIUserListResponseBlock
onError	TUIErrorBlock	Failed callback

## getUserInfo

Get Member data.



```
- (void)getUserInfo:(NSString *)userId
    onSuccess:(TUIUserInfoBlock)onSuccess
```

```
onError: (TUIErrorBlock) onError;
```

The parameters are as follows:

Parameter	Type	Meaning
userId	NSString *	User ID
onSuccess	TUIUserInfoBlock	Success Callback. Detailed Definition can be found in TUIRoomDefine.h file's TUIUserInfoBlock
onError	TUIErrorBlock	Error callback

## changeUserRole

Modify User Role (Only Administrator or Group owner can call).



```
- (void)changeUserRoleWithUserId:(NSString *)userId
    role:(TUIRole)role
    onSuccess:(TUISuccessBlock)onSuccess
    onError:(TUIErrorBlock)onError;
```

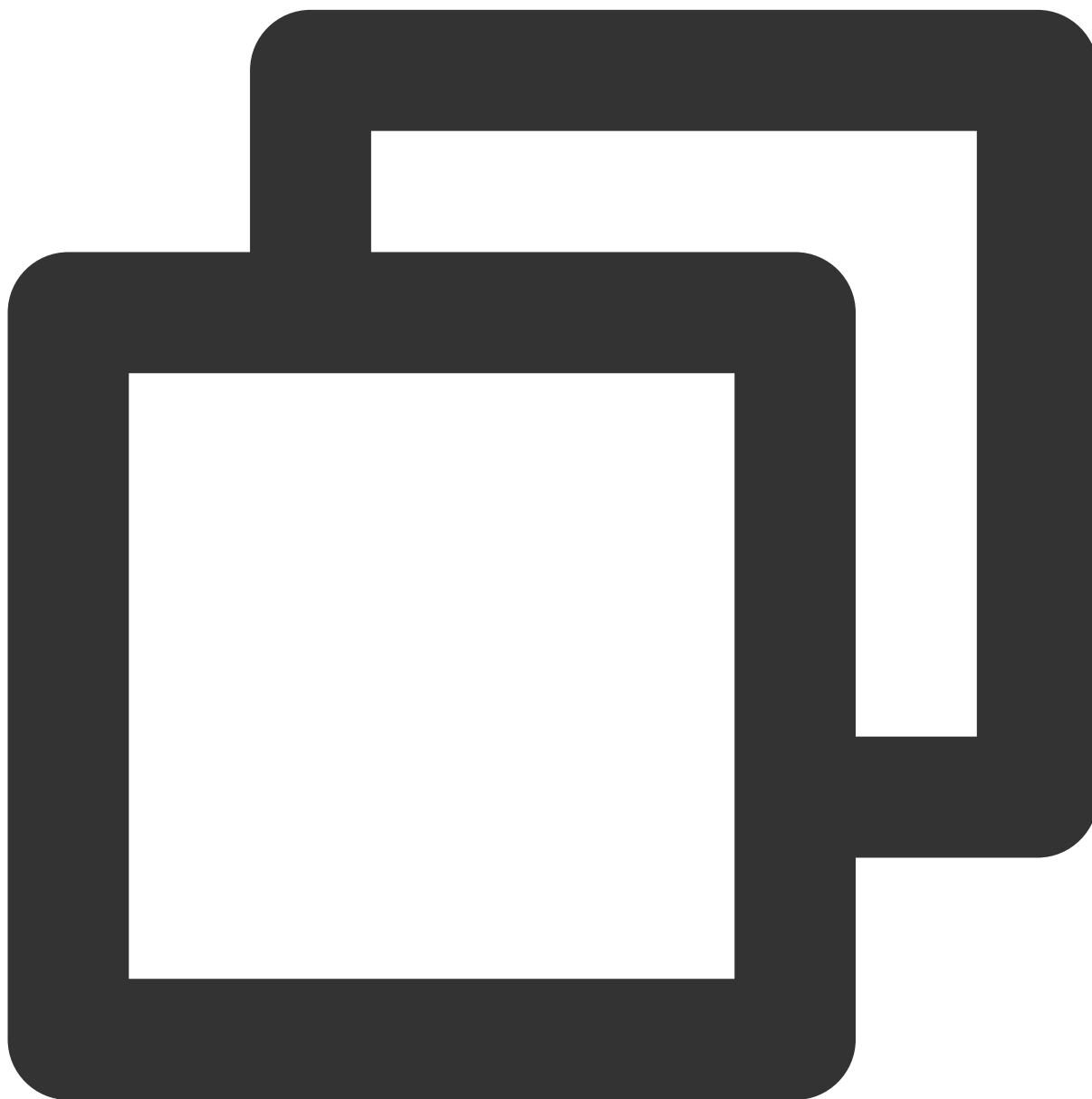
The parameters are as follows:

Parameter	Type	Meaning
userId	NSString *	User ID



role	<a href="#">TUIRole</a>	Role to switch to
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

## kickRemoteUserOutOfRoom



```
- (void)kickRemoteUserOutOfRoom:(NSString *)userId
    onSuccess:(TUISuccessBlock) onSuccess
    onError:(TUIErrorBlock) onError;
```

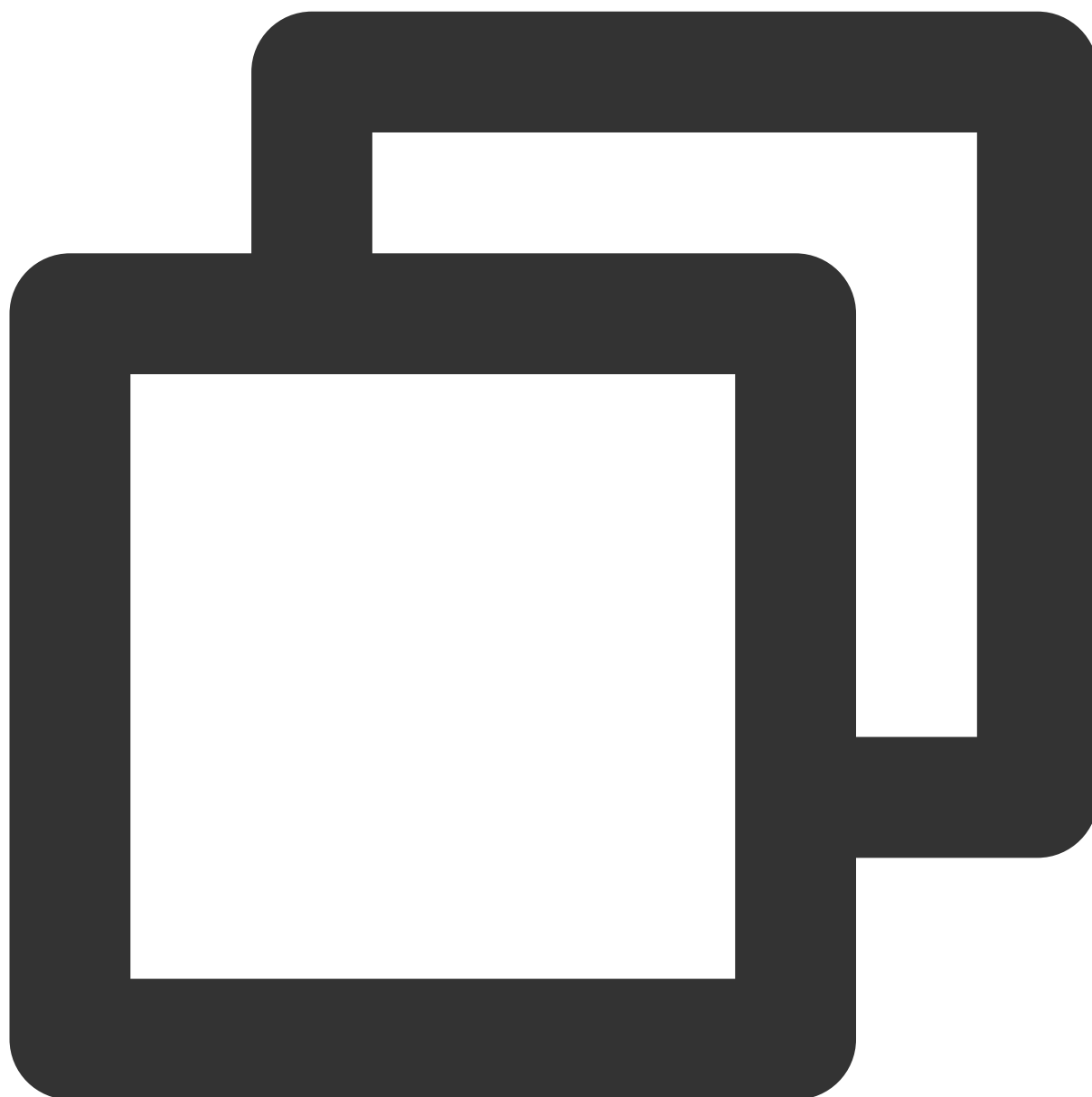
Kick Remote User out of the Room (Only Administrator or Group Owner can call).

Parameter	Type	Meaning
userId	NSString *	User ID
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Failed callback

### **disableDeviceForAllUserByAdmin**

Control the permission status of all users in the current room to open Audio and Video Capturing devices, such as: Disable all users from opening mic, Disable all users from opening Camera, Disable all users from opening Screen Sharing

(Currently only available in conference scenes, and only Administrator or Group Owner can call).



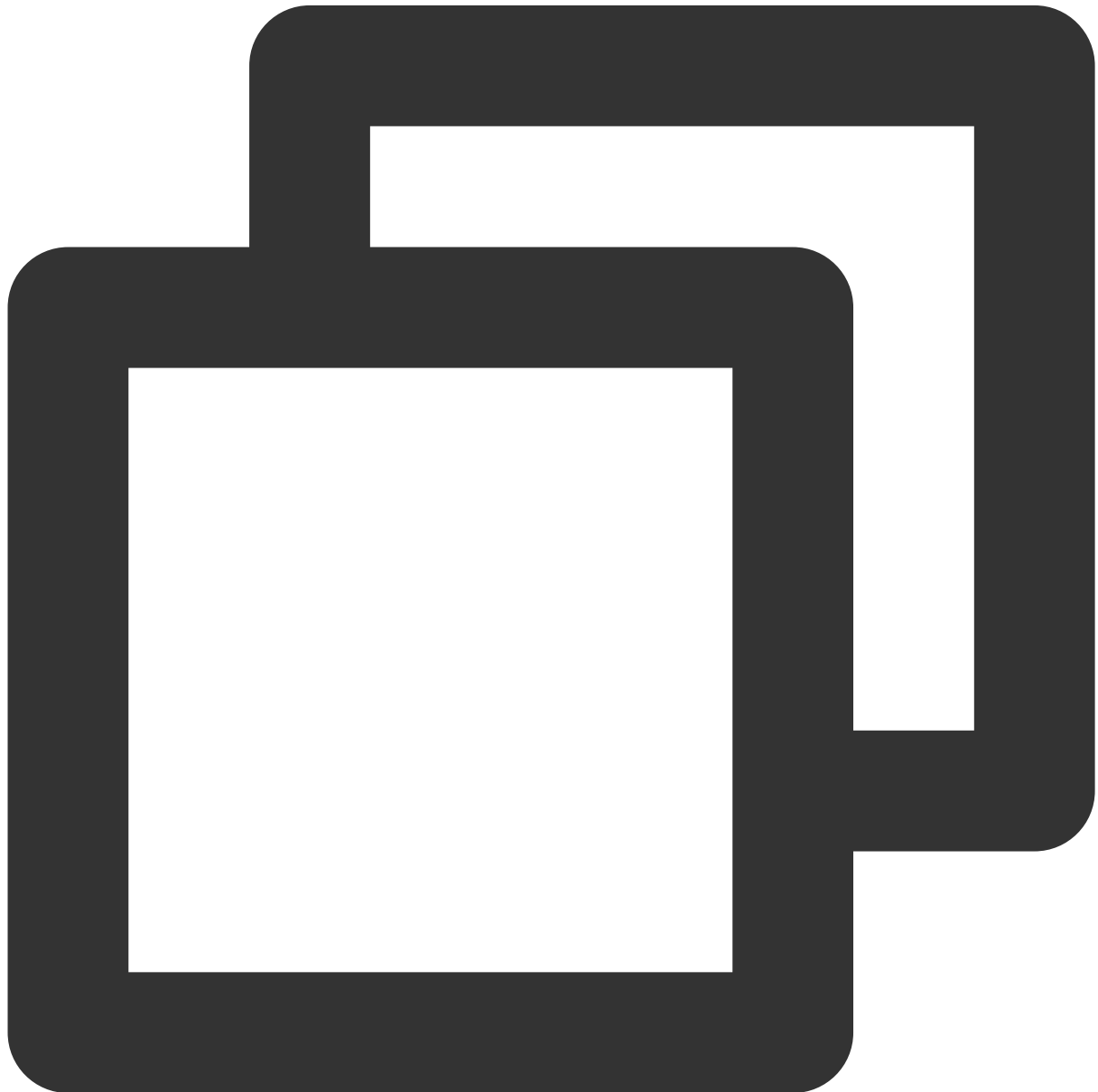
```
- (void)disableDeviceForAllUserByAdmin:(TUIMediaDevice)device
    isDisable:(BOOL)isDisable
    onSuccess:(TUISuccessBlock)onSuccess
    onError:(TUIErrorBlock)onError;
```

Parameter	Type	Meaning
device	<a href="#">TUIMediaDevice</a>	Device Type
isDisable	BOOL	Disable or Not

onSuccess	TUISuccessBlock	Operation Success Callback
onError	TUIErrorBlock	Operation Failure Callback

## openRemoteDeviceByAdmin

Request Remote User to Open Media Device (Only Administrator or Group Owner can call).



```
- (TUIRequest *)openRemoteDeviceByAdmin:(NSString *)userId
```

```

        device: (TUIMediaDevice) device
        timeout: (NSTimeInterval) timeout
        onAccepted: (nullable TUIRequestAcceptedBlock) onAccepted
        onRejected: (nullable TUIRequestRejectedBlock) onRejected
        onCancelled: (nullable TUIRequestCancelledBlock) onCancelled
        onTimeout: (nullable TUIRequestTimeoutBlock) onTimeout
        onError: (nullable TUIRequestErrorBlock) onError;

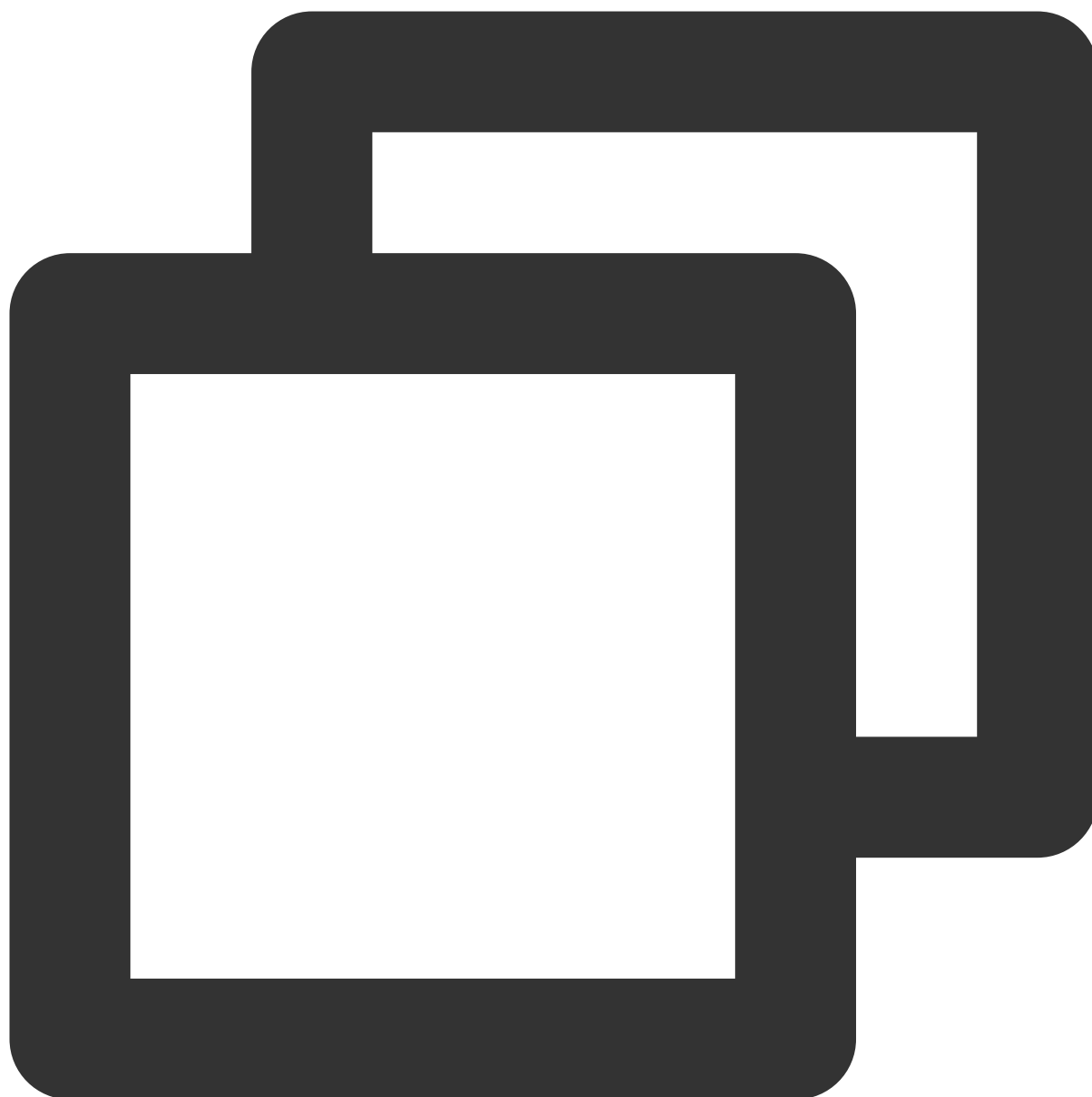
```

Parameter	Type	Meaning
userId	NSString *	User ID
device	<a href="#">TUIMediaDevice</a>	Media Device
timeout	NSTimeInterval	Timeout Duration, Unit Second, if set to 0, SDK will not perform timeout detection and will not trigger timeout callback
onAccepted	nullable TUIRequestAcceptedBlock	Invitation Accepted Callback
onRejected	nullable TUIRequestRejectedBlock	Invitation Rejected Callback
onCancelled	nullable TUIRequestCancelledBlock	Invitation Canceled Callback
onTimeout	nullable TUIRequestTimeoutBlock	Invitation Timeout Unhandled Callback
onError	nullable TUIRequestErrorBlock	Invitation Error Callback

Return Value	Type	Meaning
request	<a href="#">TUIRequest</a>	Request Body

## closeRemoteDeviceByAdmin

Close Remote User's Media Device (Only Administrator or Group Owner can call).



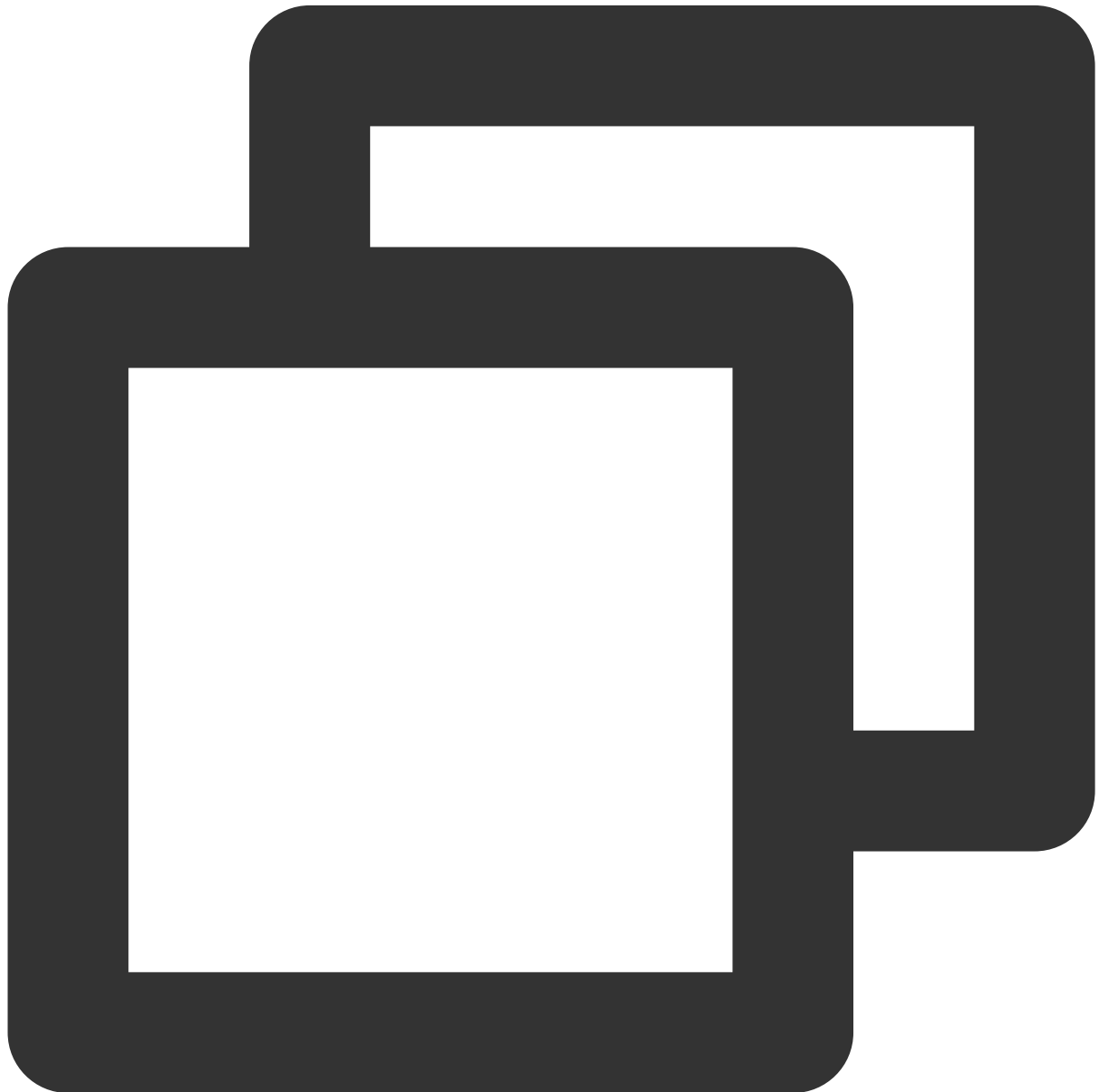
```
- (void)closeRemoteDeviceByAdmin:(NSString *)userId
    device:(TUIMediaDevice) device
    onSuccess:(TUISuccessBlock) onSuccess
    onError:(TUIErrorBlock) onError;
```

Parameter	Type	Meaning
userId	NSString *	User ID
device	<a href="#">TUIMediaDevice</a>	Media Device

onSuccess	TUISuccessBlock	Call Success Callback
onError	TUIErrorBlock	Call Failure Callback

## **applyToAdminToOpenLocalDevice**

Request to Open Local Media Device (Available for ordinary users).



```
- (TUIRequest *)applyToAdminToOpenLocalDevice:(TUIMediaDevice) device
```

```

        timeout:(NSTimeInterval)timeout
        onAccepted:(nullable TUIRequestAcceptedBlock)onA
        onRejected:(nullable TUIRequestRejectedBlock)onR
        onCancelled:(nullable TUIRequestCancelledBlock)on
        onTimeout:(nullable TUIRequestTimeoutBlock)onTi
        onError:(nullable TUIRequestErrorBlock)onErro

```

Parameter	Type	Meaning
device	<a href="#">TUIMediaDevice</a>	Media Device
timeout	NSTimeInterval	Timeout Duration, Unit Second, if set to 0, SDK will not perform timeout detection and will not trigger timeout callback
onAccepted	nullable TUIRequestAcceptedBlock	Invitation Accepted Callback
onRejected	nullable TUIRequestRejectedBlock	Invitation Rejected Callback
onCancelled	nullable TUIRequestCancelledBlock	Invitation Canceled Callback
onTimeout	nullable TUIRequestTimeoutBlock	Invitation Timeout Unhandled Callback
onError	nullable TUIRequestErrorBlock	Invitation Error Callback

Return Value	Type	Meaning
request	<a href="#">TUIRequest</a>	Request Body

## setMaxSeatCount

Set Maximum Seat Count (Only supported when entering the room and creating the room).

When roomType is TUIRoomTypeConference (Educational and Conference Scenes), maxSeatCount value is not limited;

When roomType is TUIRoomTypeLivingRoom (Live Streaming Scene), maxSeatCount is limited to 16;





```
- (void)setMaxSeatCount:(NSUInteger)maxSeatCount  
    onSuccess:(TUISuccessBlock)onSuccess  
    onError:(TUIErrorBlock)onError;
```

Parameter	Type	Meaning
maxSeatCount	NSUInteger	Maximum microphone positions
onSuccess	TUISuccessBlock	Successful callback

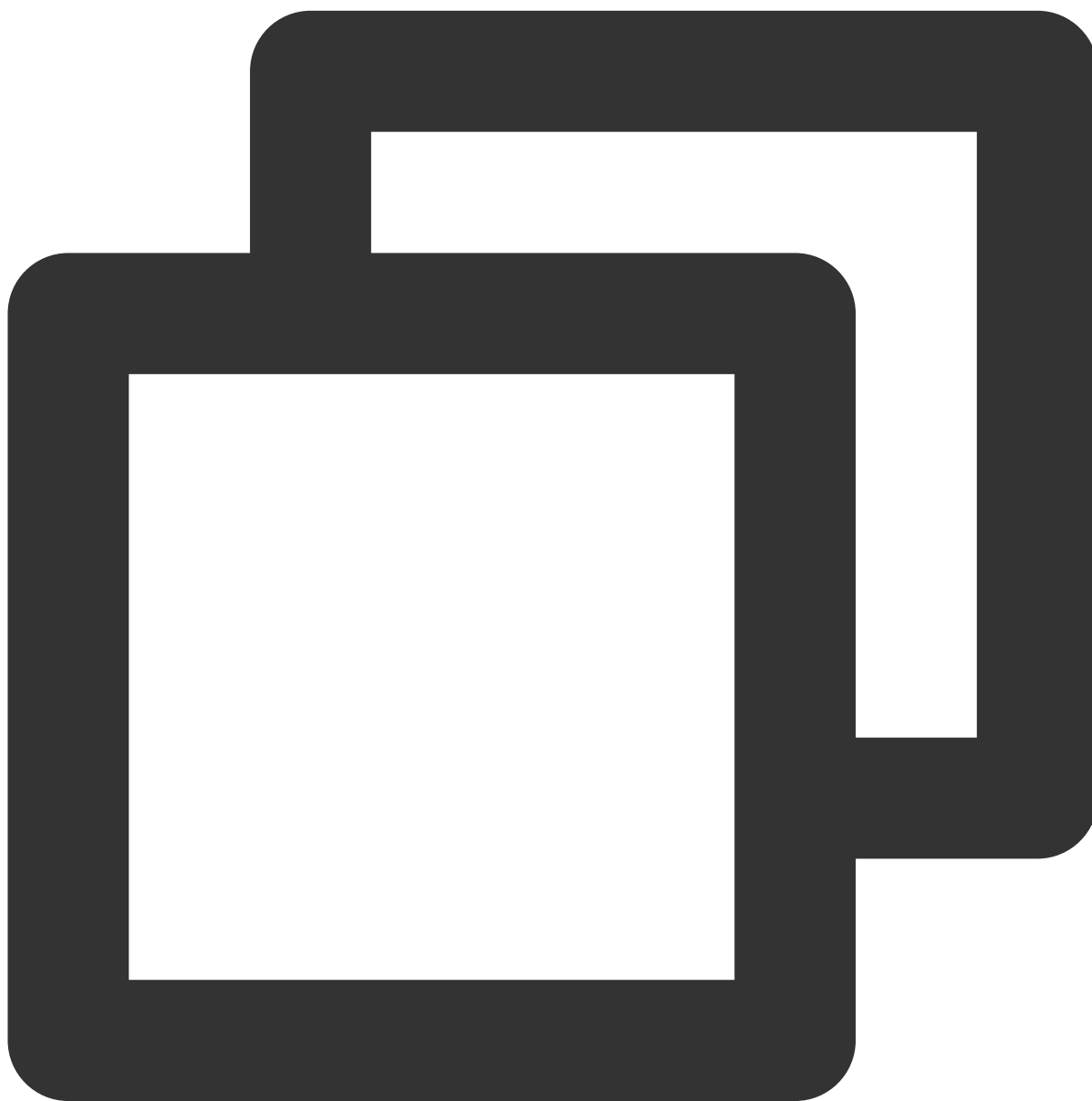
onError

TUIErrorBlock

Error callback

## getSeatList

Get Seat List.



```
- (void) getSeatList: (TUISeatListResponseBlock) onSuccess  
    onError: (TUIErrorBlock) onError;
```

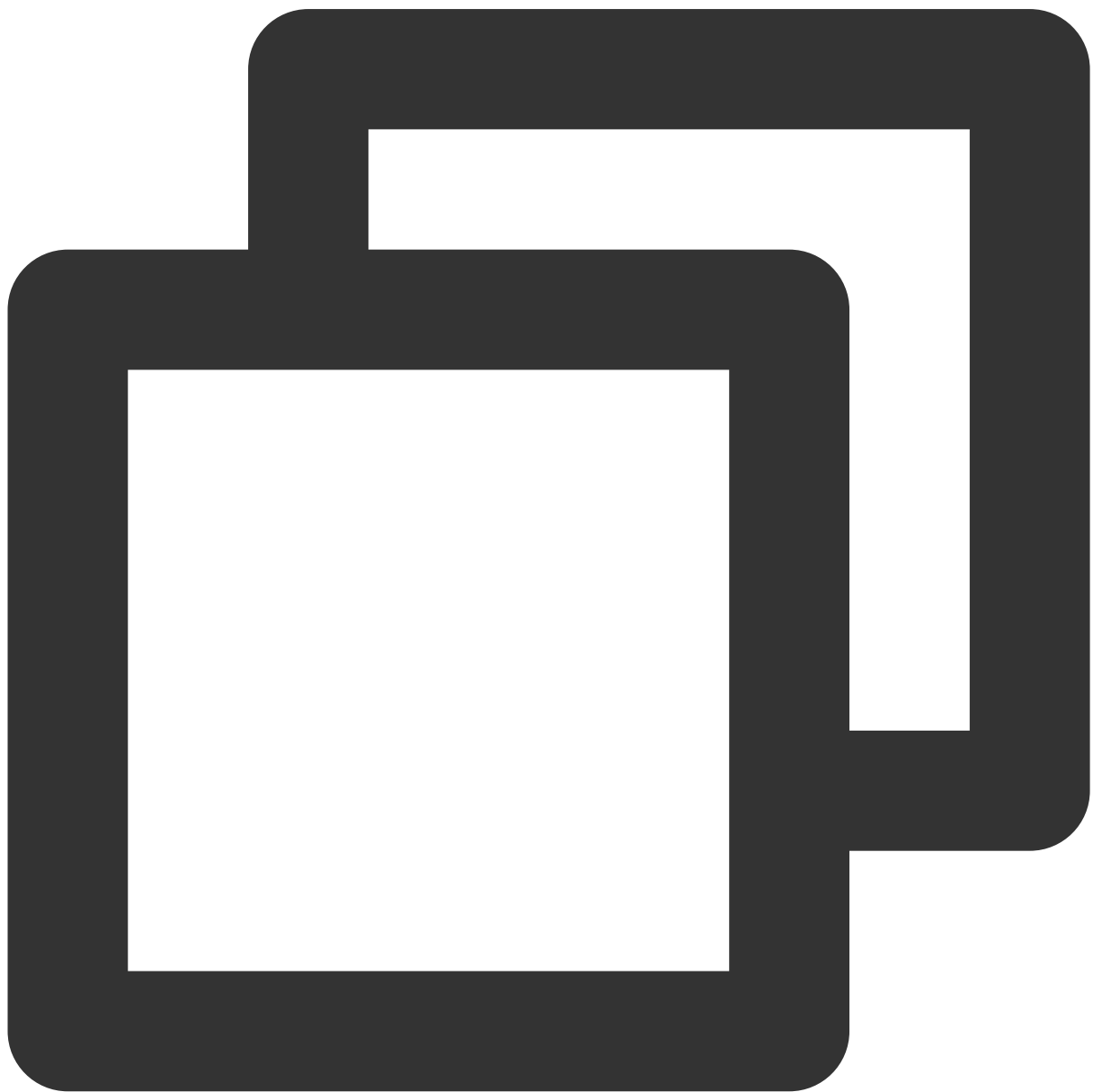
The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

onSuccess	TUISeatListResponseBlock	Success Callback. For detailed definition, please refer to TUIRoomDefine.h for TUISeatListResponseBlock and <a href="#">TUISeatInfo</a> definition
onError	TUIErrorBlock	Error callback

## lockSeatByAdmin

Lock Seat (Only Administrator or Group Owner can call, including position lock, Audio status lock and Video status lock).



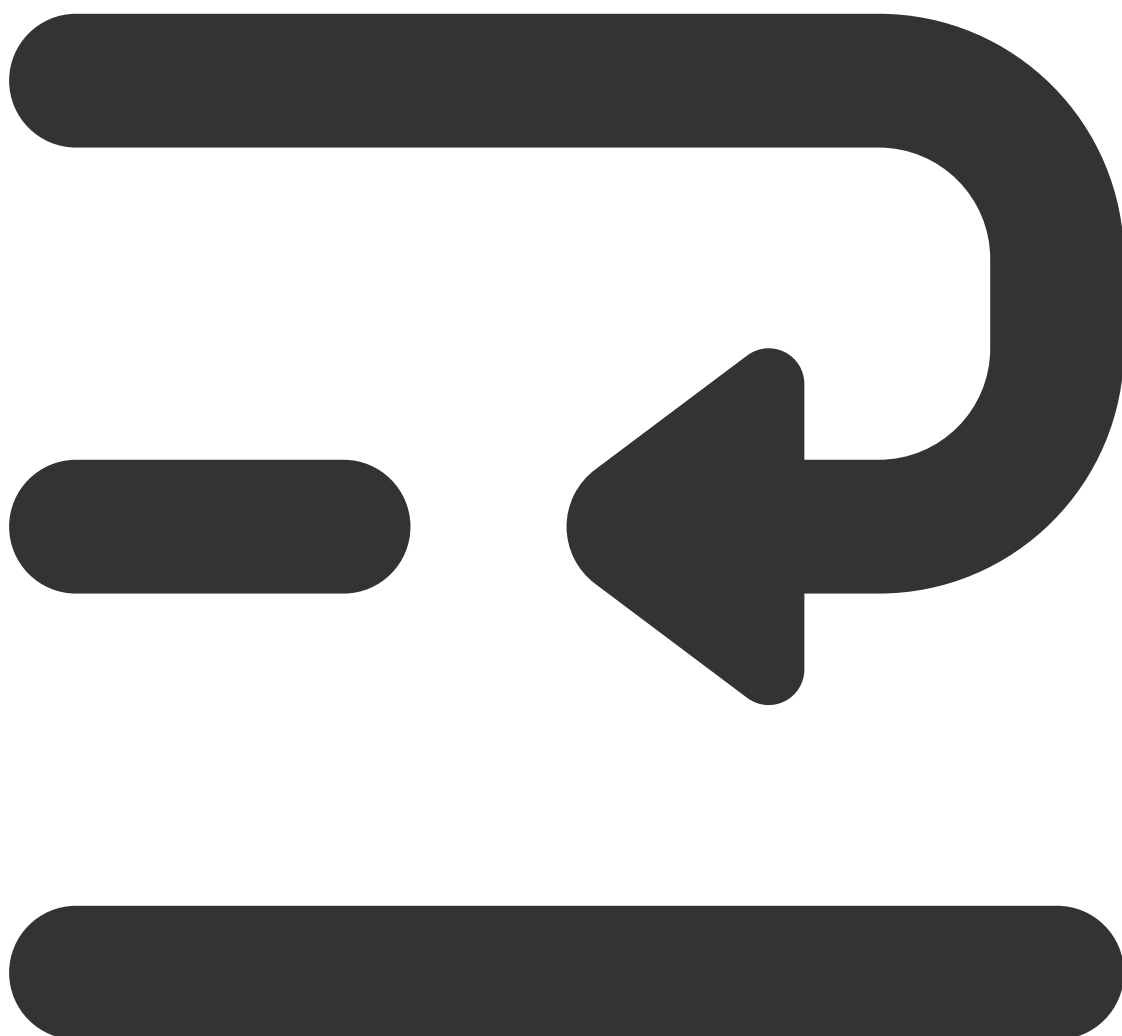
```
- (void)lockSeatByAdmin:(NSInteger)seatIndex
    lockMode:(TUISeatLockParams *)lockParams
    onSuccess:(TUISuccessBlock)onSuccess
    onError:(TUIErrorBlock)onError;
```

The parameters are as follows:

Parameter	Type	Meaning
seatIndex	NSInteger	Seat Number
lockParams	<a href="#">TUISeatLockParams</a> *	Lock Seat Parameters, for detailed definition, please refer to TUIRoomDefine.h for <a href="#">TUISeatLockParams</a> definition
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Error callback

## takeSeat

Local User Go Live (In Free Speech Mode, no application is required, in Mic Control Mode, application to the host or administrator is required to allow Go Live, in Live Streaming Scene, users can Go Live freely, after Go Live, users can speak, in Conference Scene, no need to call this interface to speak).





```
- (TUIRequest *)takeSeat:(NSInteger)seatIndex
    timeout:(NSTimeInterval)timeout
    onAccepted:(TUIRequestAcceptedBlock)onAccepted
    onRejected:(TUIRequestRejectedBlock)onRejected
    onCancelled:(TUIRequestCancelledBlock)onCancelled
    onTimeout:(TUIRequestTimeoutBlock)onTimeout
    onError:(TUIRequestErrorBlock)onError;
```

The parameters are as follows:

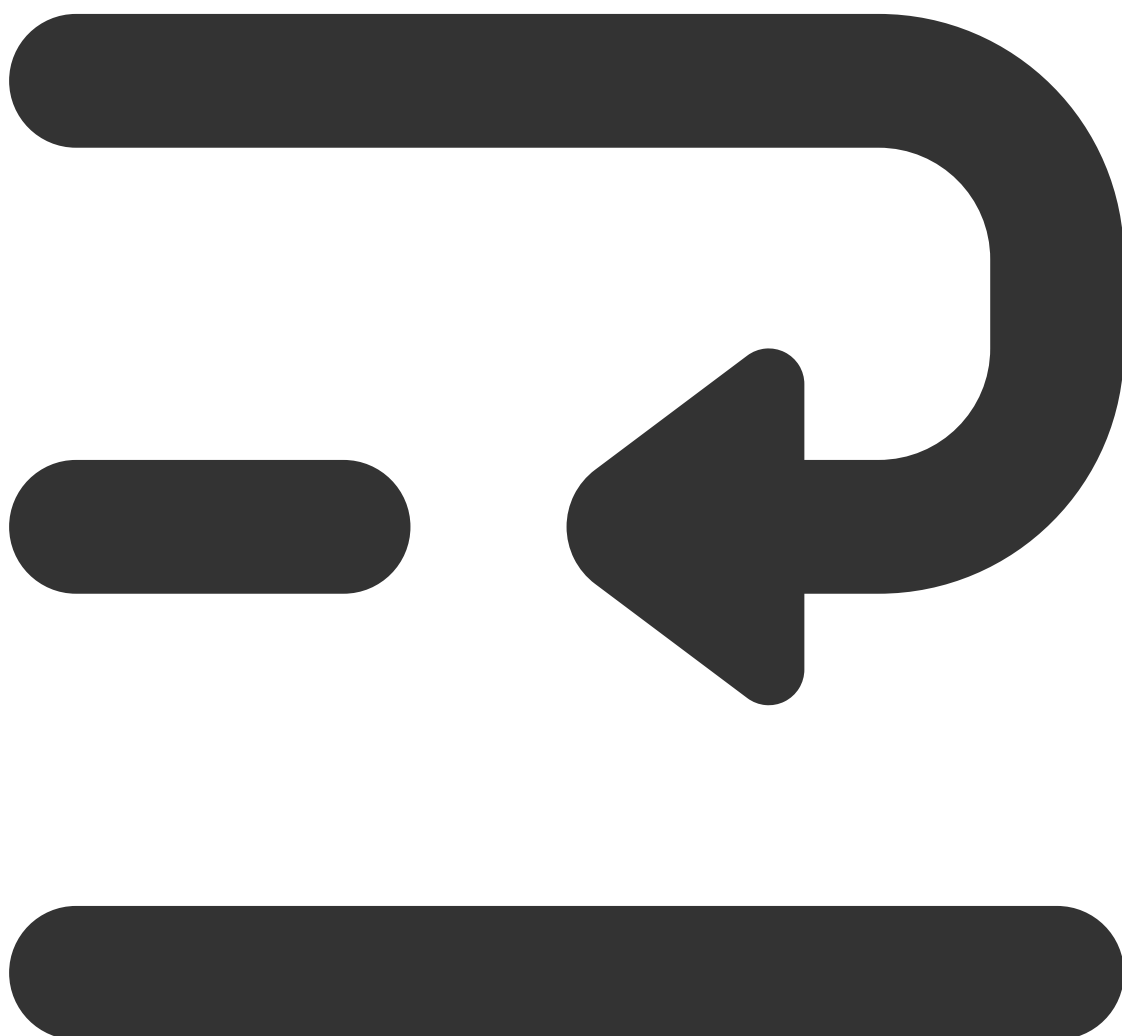
Parameter	Type	Meaning
-----------	------	---------

seatIndex	NSInteger	Seat Number
timeout	NSTimeInterval	Timeout Duration
onAccepted	TUIRequestAcceptedBlock	Signaling Accepted Callback
onRejected	TUIRequestRejectedBlock	Signaling Rejected Callback
onCancelled	TUIRequestCancelledBlock	Signaling Canceled Callback
onTimeout	TUIRequestTimeoutBlock	Signaling Timeout Callback
onError	TUIRequestErrorBlock	Error Callback

Return Value	Type	Meaning
request	<a href="#">TUIRequest</a>	Request Body

## leaveSeat

Local User Get Off the Mic (In Free Speech Mode, no application is required).







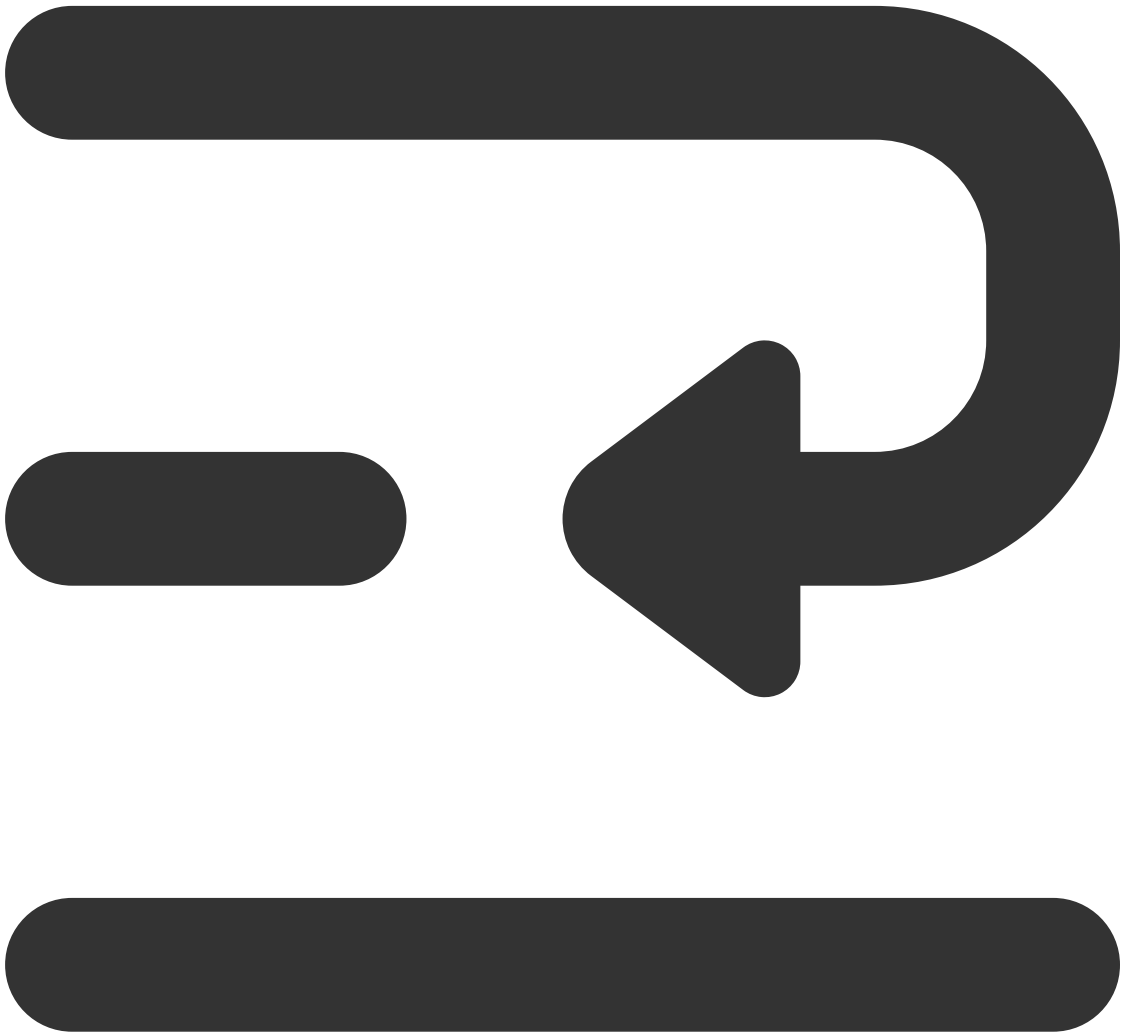
```
- (void)leaveSeat:(TUISuccessBlock) onSuccess  
    onError:(TUIErrorBlock) onError;
```

**The parameters are as follows:**

Parameter	Type	Meaning
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Error callback

## takeUserOnSeatByAdmin

Host/Administrator Invite User to Go Live.





```
- (TUIRequest *)takeUserOnSeatByAdmin:(NSInteger)seatIndex
    userId:(NSString *)userId
    timeout:(NSTimeInterval)timeout
    onAccepted:(TUIRequestAcceptedBlock)onAccepted
    onRejected:(TUIRequestRejectedBlock)onRejected
    onCancelled:(TUIRequestCancelledBlock)onCancelled
    onTimeout:(TUIRequestTimeoutBlock)onTimeout
    onError:(TUIRequestErrorBlock)onError;
```

The parameters are as follows:

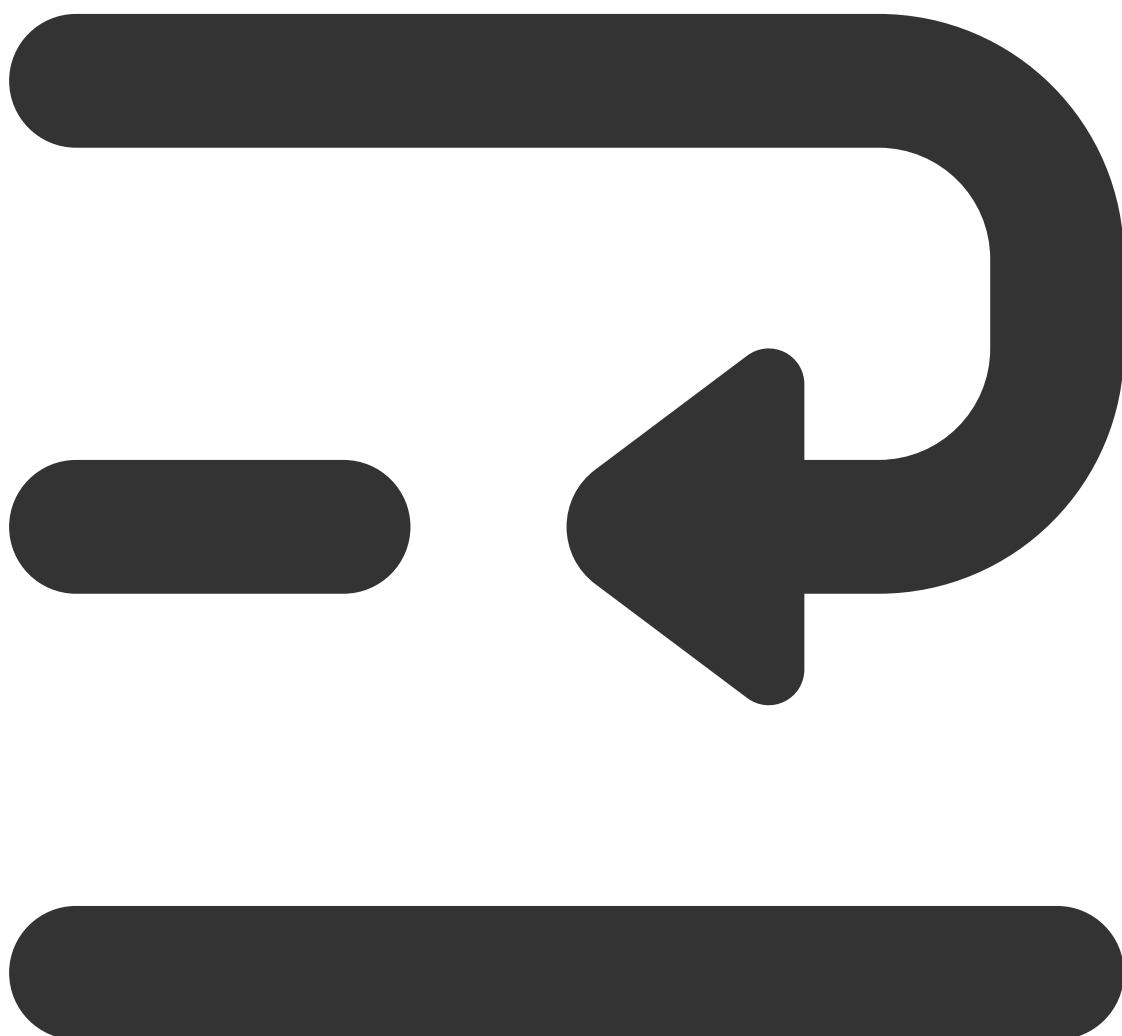
--	--	--

Parameter	Type	Meaning
seatIndex	NSInteger	Seat Number
userId	NSString *	User ID
timeout	NSTimeInterval	Timeout Duration (Unit Second, if set to 0, SDK will not perform timeout detection and will not trigger timeout callback)
onAccepted	TUIRequestAcceptedBlock	Invitation Accepted Callback
onRejected	TUIRequestRejectedBlock	Invitation Rejected Callback
onCancelled	TUIRequestCancelledBlock	Invitation Canceled Callback
onTimeout	TUIRequestTimeoutBlock	Invitation Timeout Unhandled Callback
onError	TUIRequestErrorBlock	Invitation Error Callback

Return Value	Type	Meaning
request	<a href="#">TUIRequest</a>	Request Body

## kickUserOffSeatByAdmin

Host/Administrator Kick User Off the Mic.





```
- (void)kickUserOffSeatByAdmin:(NSInteger)seatIndex
    userId:(NSString *)userId
    onSuccess:(TUISuccessBlock)onSuccess
    onError:(TUIErrorBlock)onError;
```

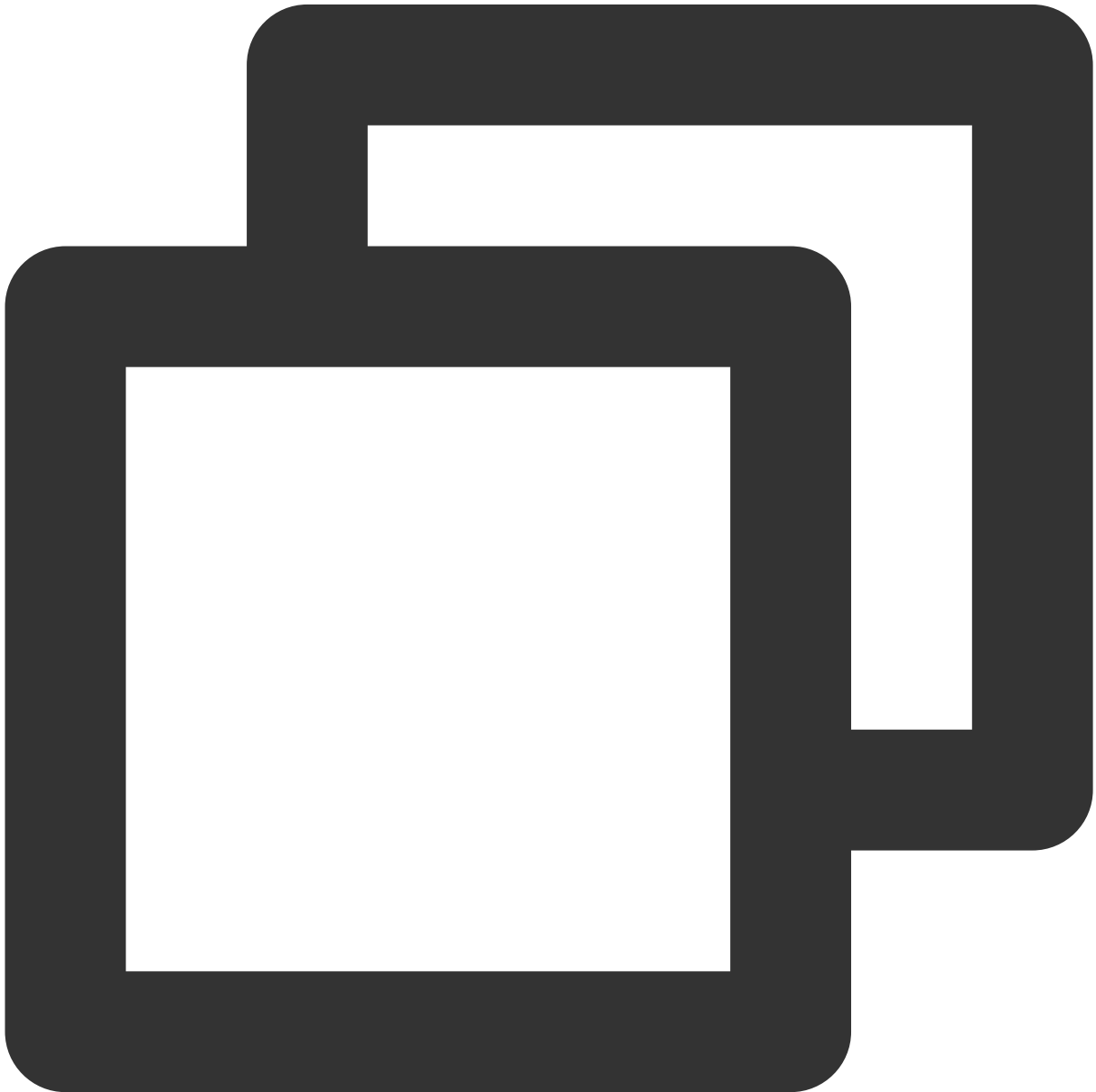
The parameters are as follows:

Parameter	Type	Meaning
seatIndex	NSInteger	Seat Number

userId	NSString *	User ID
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Error callback

cancelRequest

Cancel Request.



```
- (void)cancelRequest:(NSInteger)requestId
```

```
onSuccess:(TUISuccessBlock)onSuccess  
onError:(TUIErrorBlock)onError;
```

The parameters are as follows:

Parameter	Type	Meaning
requestId	NSString *	Request ID. Returned by the sending request interface
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Error callback

## responseRemoteRequest

Reply Request.





```
- (void)responseRemoteRequest:(NSString *)requestId  
    agree:(BOOL)agree  
    onSuccess:(TUISuccessBlock)onSuccess  
    onError:(TUIErrorBlock)onError;
```

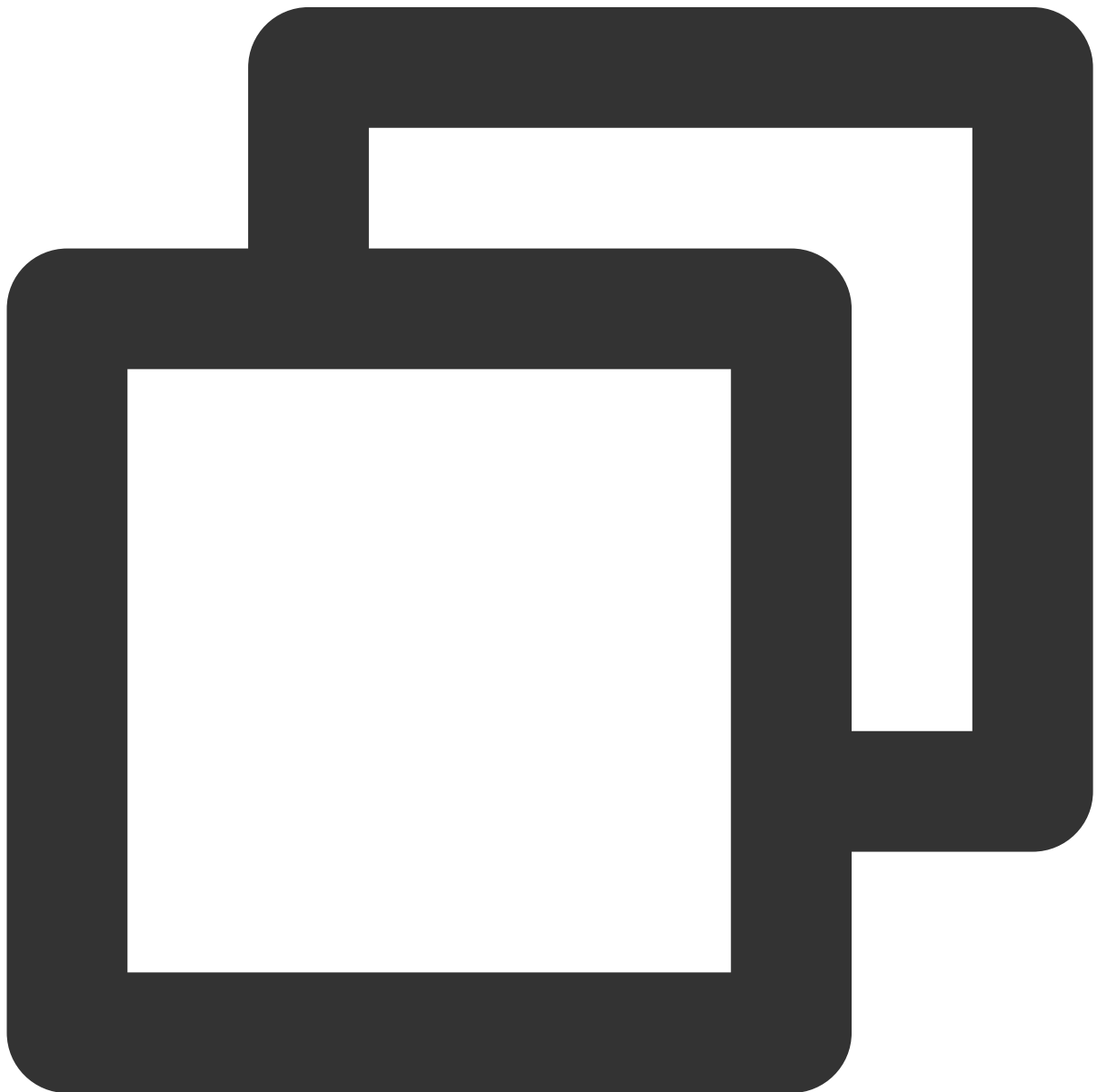
The parameters are as follows:

Parameter	Type	Meaning
requestId	NSString *	Request ID. Returned by the sending request interface or OnRequestReceived event notification

agree	BOOL	Agree or not. YES: Agree, NO: Reject
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Error callback

## sendMessage

Send Text Message.



```
- (void)sendMessage:(NSString *)message
```

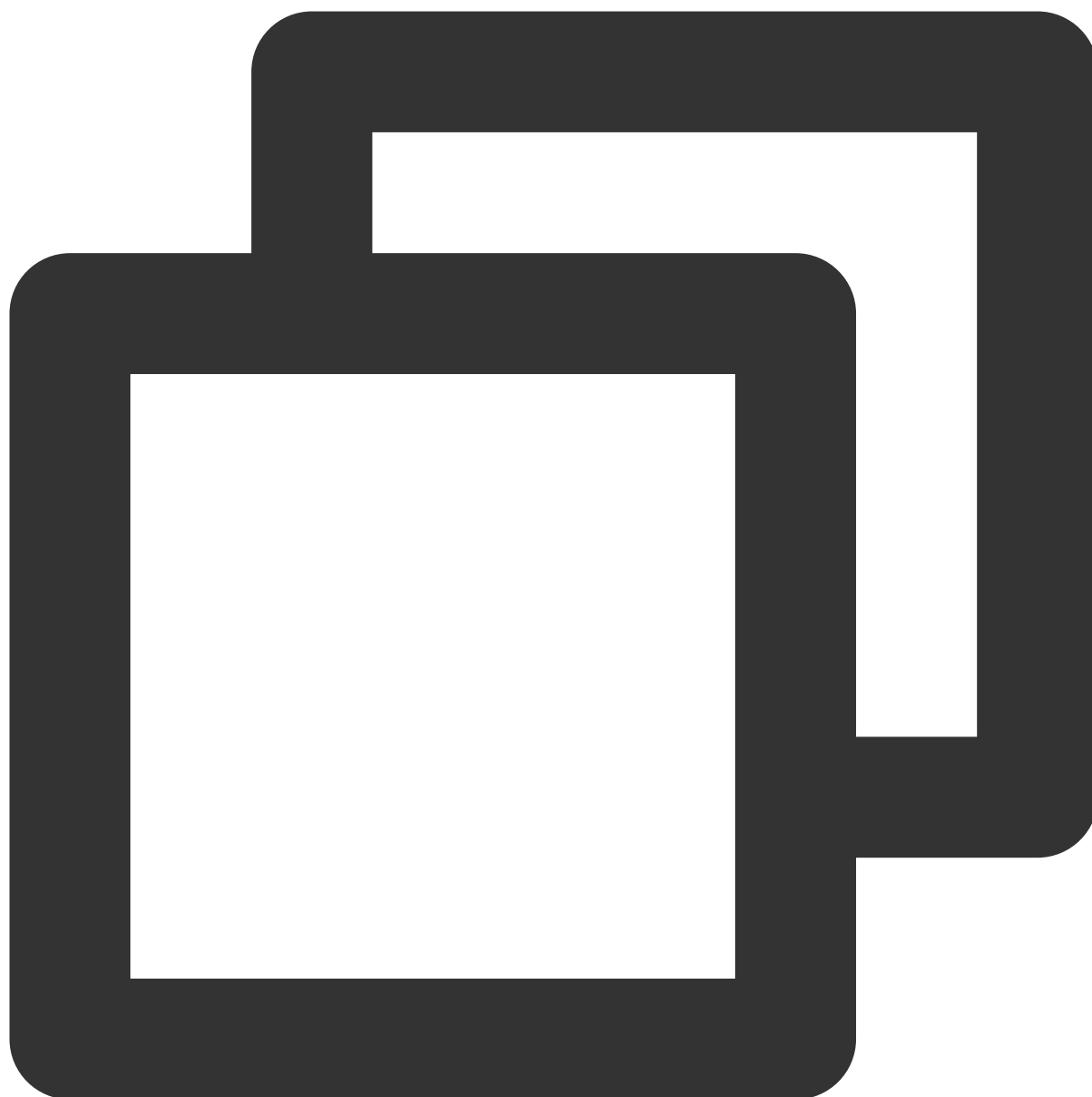
```
onSuccess:(TUISuccessBlock) onSuccess  
onError:(TUIErrorBlock) onError;
```

The parameters are as follows:

Parameter	Type	Meaning
message	NSString *	Message Content
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Error callback

## sendCustomMessage

Send Custom Message.



```
- (void)sendCustomMessage:(NSString *)message  
    onSuccess:(TUISuccessBlock)onSuccess  
    onError:(TUIErrorBlock)onError;
```

**The parameters are as follows:**

Parameter	Type	Meaning
message	NSString *	Message Content
onSuccess	TUISuccessBlock	Successful callback

onError

TUIErrorBlock

Error callback

## disableSendingMessageByAdmin

Disable Remote User's Text Message Sending Ability (Only Administrator or Group Owner can call).



```
- (void)disableSendingMessageByAdmin:(NSString *)userId
    isDisable:(BOOL)isDisable
    onSuccess:(TUISuccessBlock)onSuccess
    onError:(TUIErrorBlock)onError;
```

Parameter	Type	Meaning
userId	NSString *	User ID
isDisable	BOOL	Disable or Not
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Error callback

### **disableSendingMessageForAllUser**

Disable All Users' Text Message Sending Ability (Only Administrator or Group Owner can call).

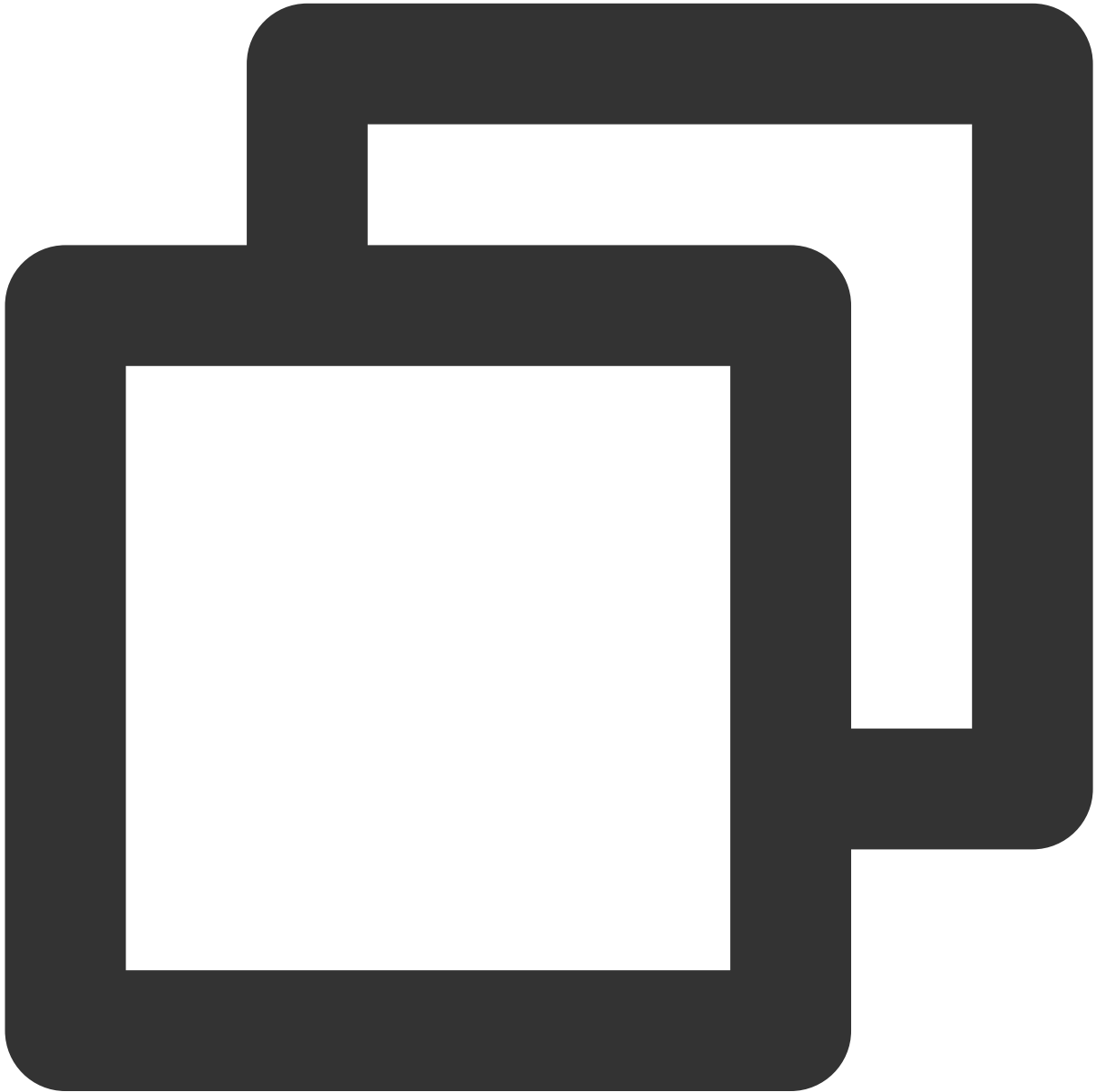


```
- (void)disableSendingMessageForAllUser:(BOOL)isDisable  
    onSuccess:(TUISuccessBlock)onSuccess  
    onError:(TUIErrorBlock)onError;
```

Parameter	Type	Meaning
isDisable	BOOL	Disable or Not
onSuccess	TUISuccessBlock	Successful callback
onError	TUIErrorBlock	Error callback

## getManager

Get Device Manager Object.



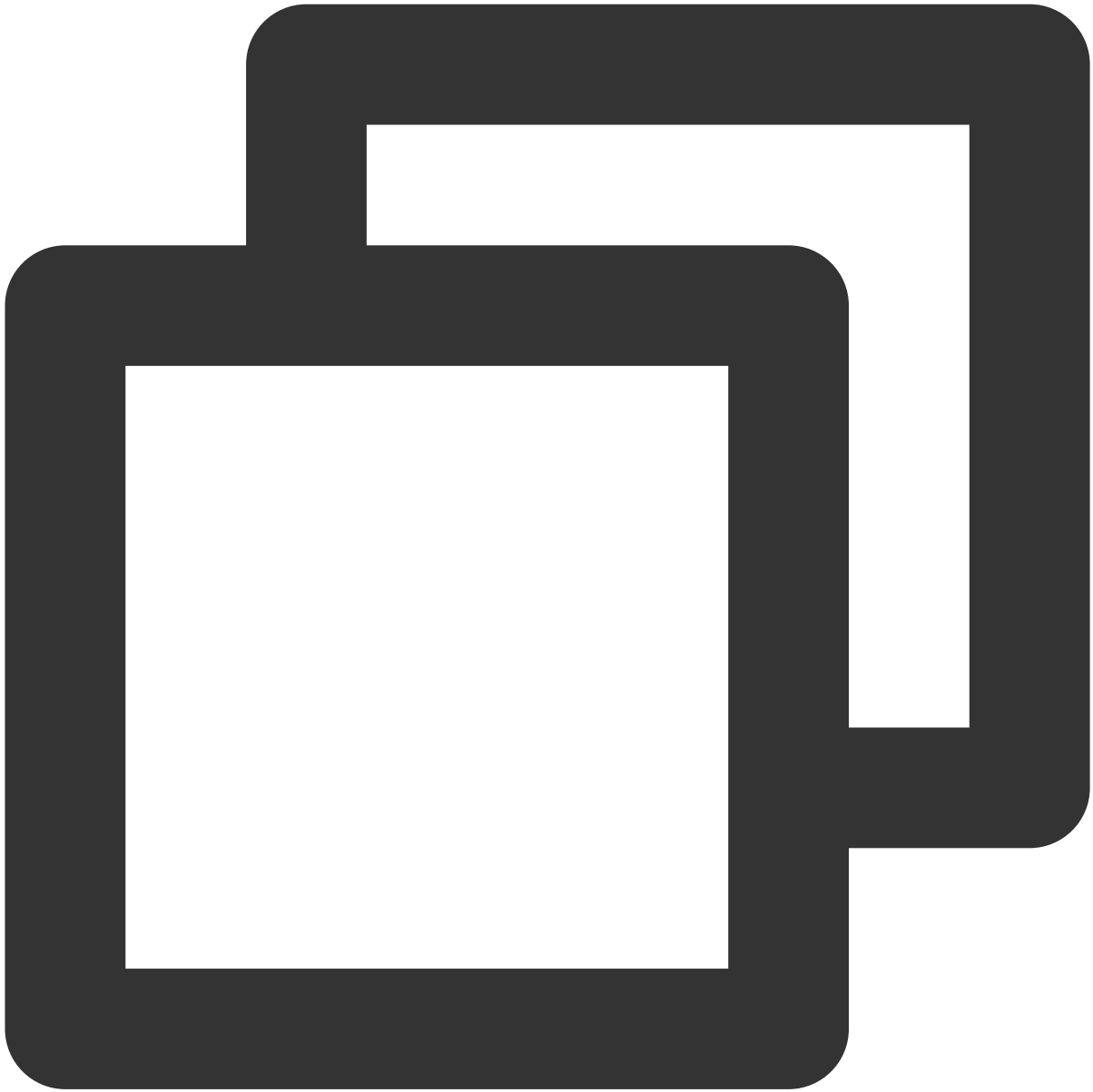
```
- (TXDeviceManager *)getManager;
```

Return Value	Type	Meaning
manager	TXDeviceManager *	TXDeviceManager Object



## getBeautyManager

Get Sound Effect Manager Object.

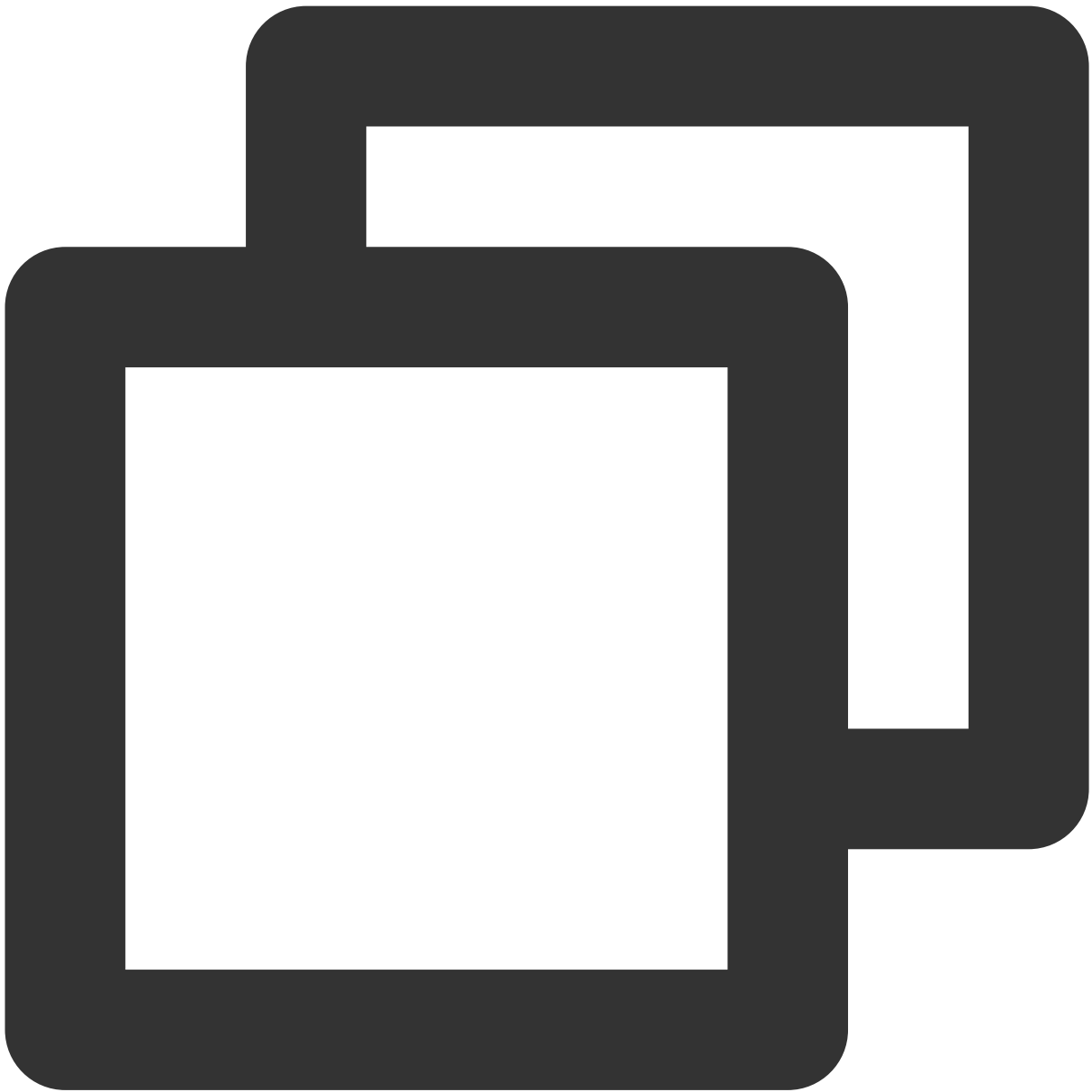


```
- (TXBeautyManager *)getBeautyManager;
```

Return Value	Type	Meaning
manager	TXBeautyManager *	TXBeautyManager Object

## getAudioEffectManager

Get Sound Effect Manager Object.

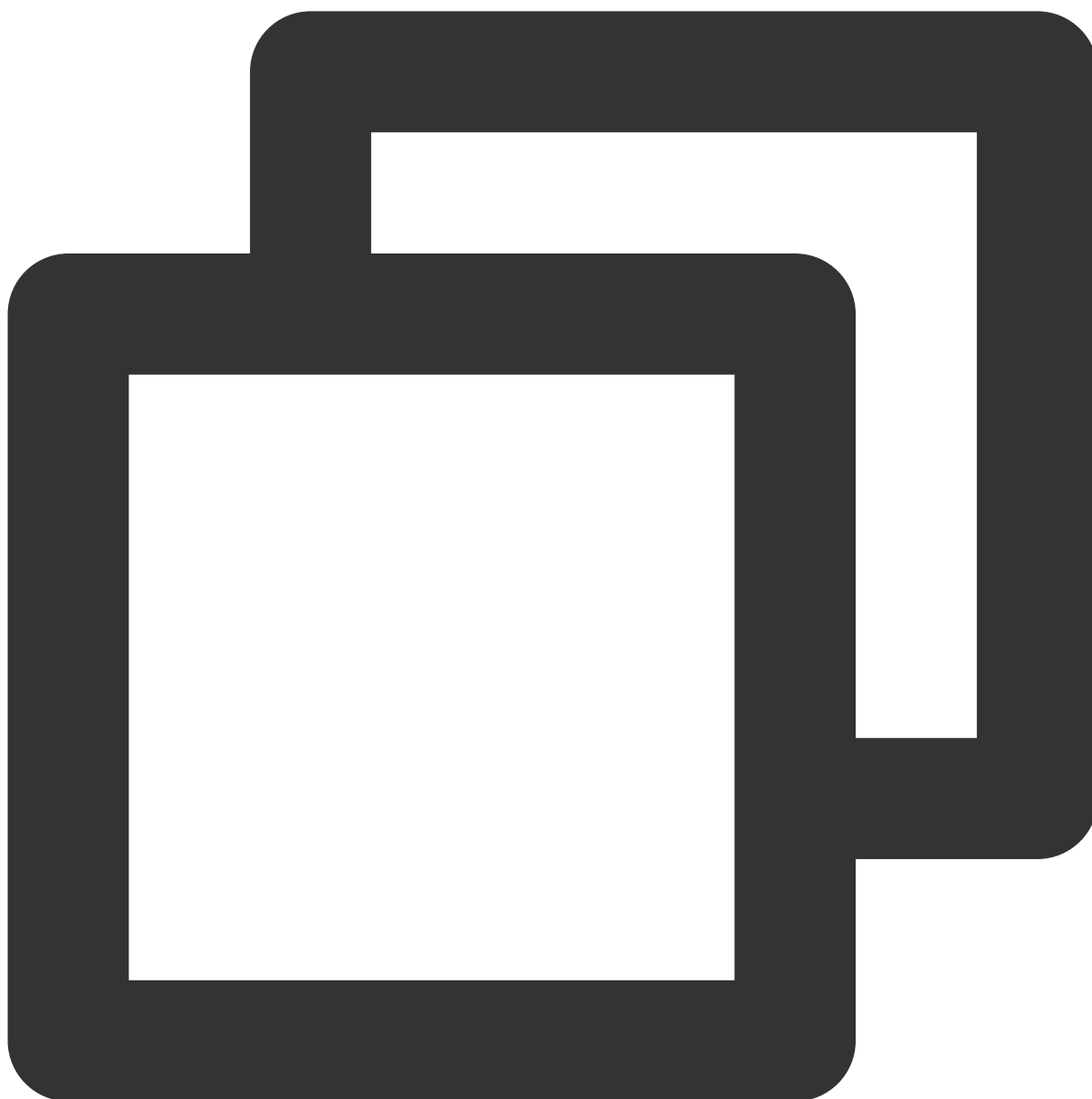


```
- (TXAudioEffectManager *)getAudioEffectManager;
```

Return Value	Type	Meaning
manager	TXAudioEffectManager *	TXAudioEffectManager Object

getTRTCCloud

Get TRTC Instance Object.



```
- (TRTCCloud *)getTRTCCloud;
```

Return Value	Type	Meaning
manager	TRTCCloud *	TRTCCloud Object

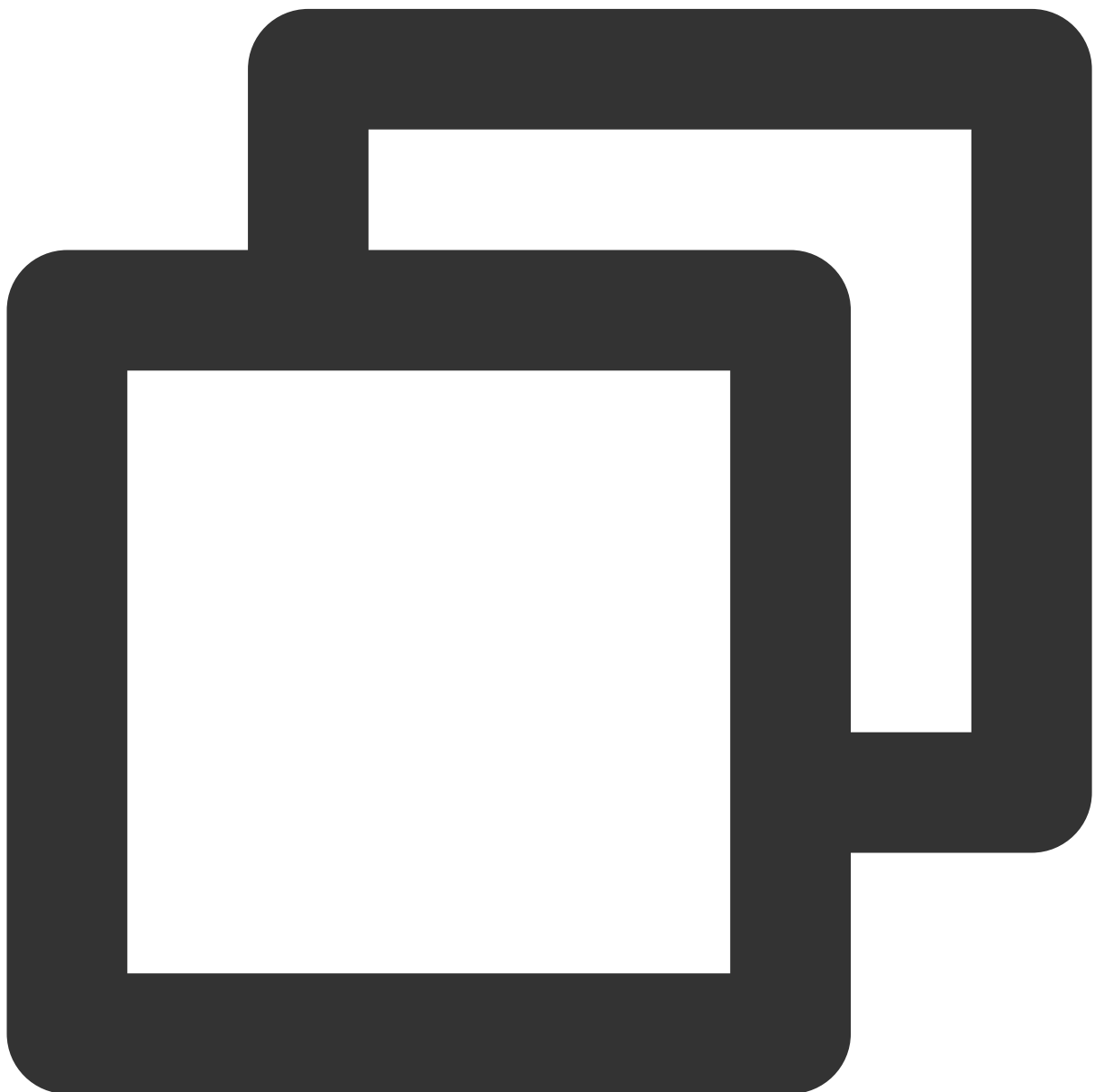
# TUIRoomObserver

Last updated : 2023-12-18 17:52:47

The TUIRoomObserver class is the Callback Event class corresponding to the TUIRoomEngine. You can listen to the Callback Events you are interested in through this interface.

## **onError**

Error Event Callback.



```
- (void)onError:(TUIError)error message:(NSString *)message;
```

The parameters are as follows:

Parameter	Type	Meaning
error	<a href="#">TUIError</a>	Error Code
message	NSString *	Error Message

## onKickedOffline

Other terminals login and get kicked off event.



```
- (void)onKickedOffLine:(NSString *)message;
```

The parameters are as follows:

Parameter	Type	Meaning
message	NSString *	Kicked out description

### onUserSigExpired

User credential timeout event.



```
- (void)onUserSigExpired;
```

### **onRoomNameChanged**

Room name change event.



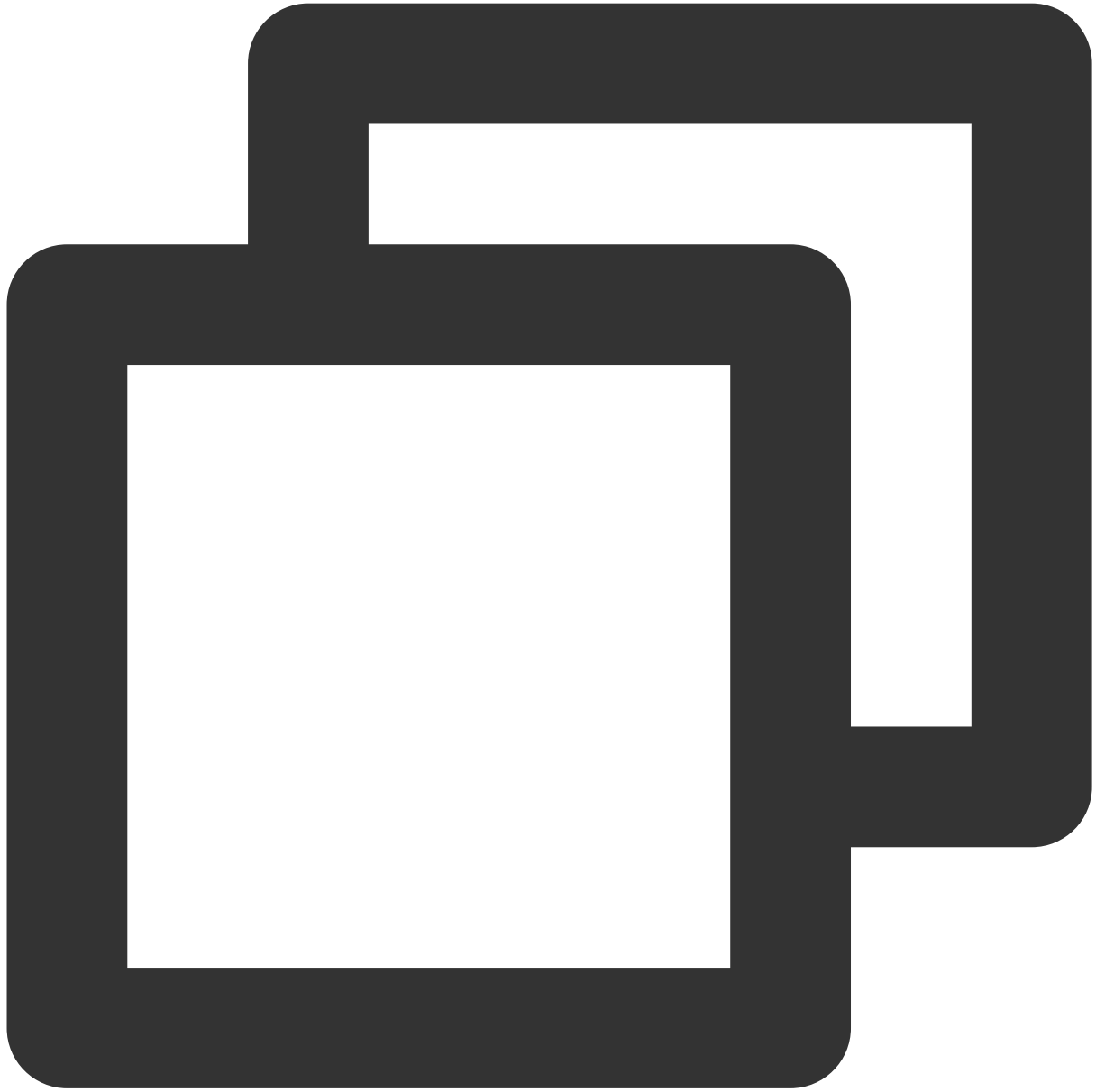
```
- (void)onRoomNameChanged:(NSString *)roomId roomName:(NSString *)roomName;
```

Parameter	Type	Meaning
roomId	NSString *	Room ID
roomName	NSString *	Room Name

### onAllUserMicrophoneDisableChanged



Inside the room, all users' mic is disabled event.

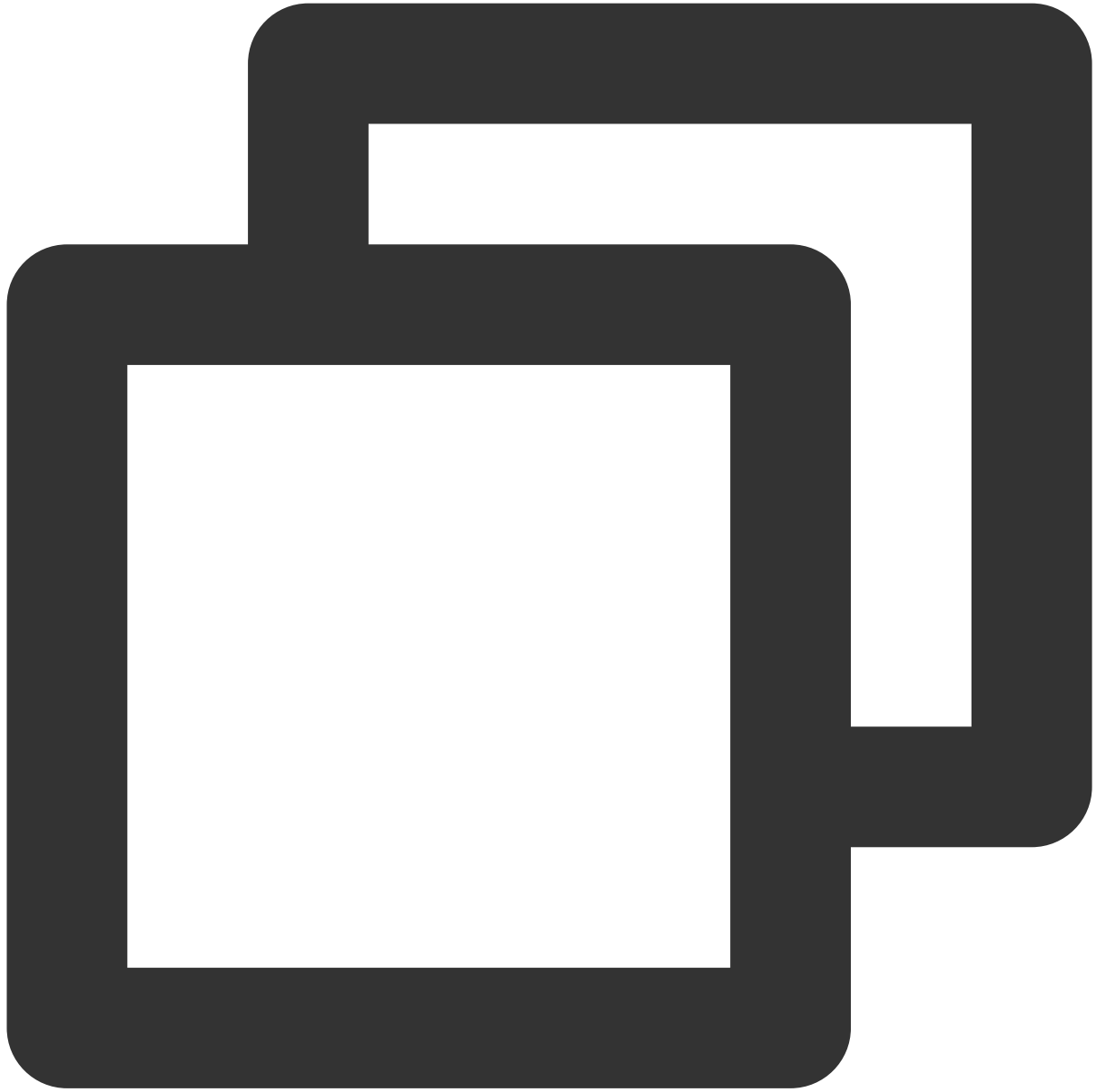


```
- (void)onAllUserMicrophoneDisableChanged:(NSString *)roomId isDisable:(BOOL)isDisa
```

Parameter	Type	Meaning
roomId	NSString *	Room ID
isDisable	BOOL	Whether it is disabled

### onAllUserCameraDisableChanged

All users' Camera in the Room are disabled event.

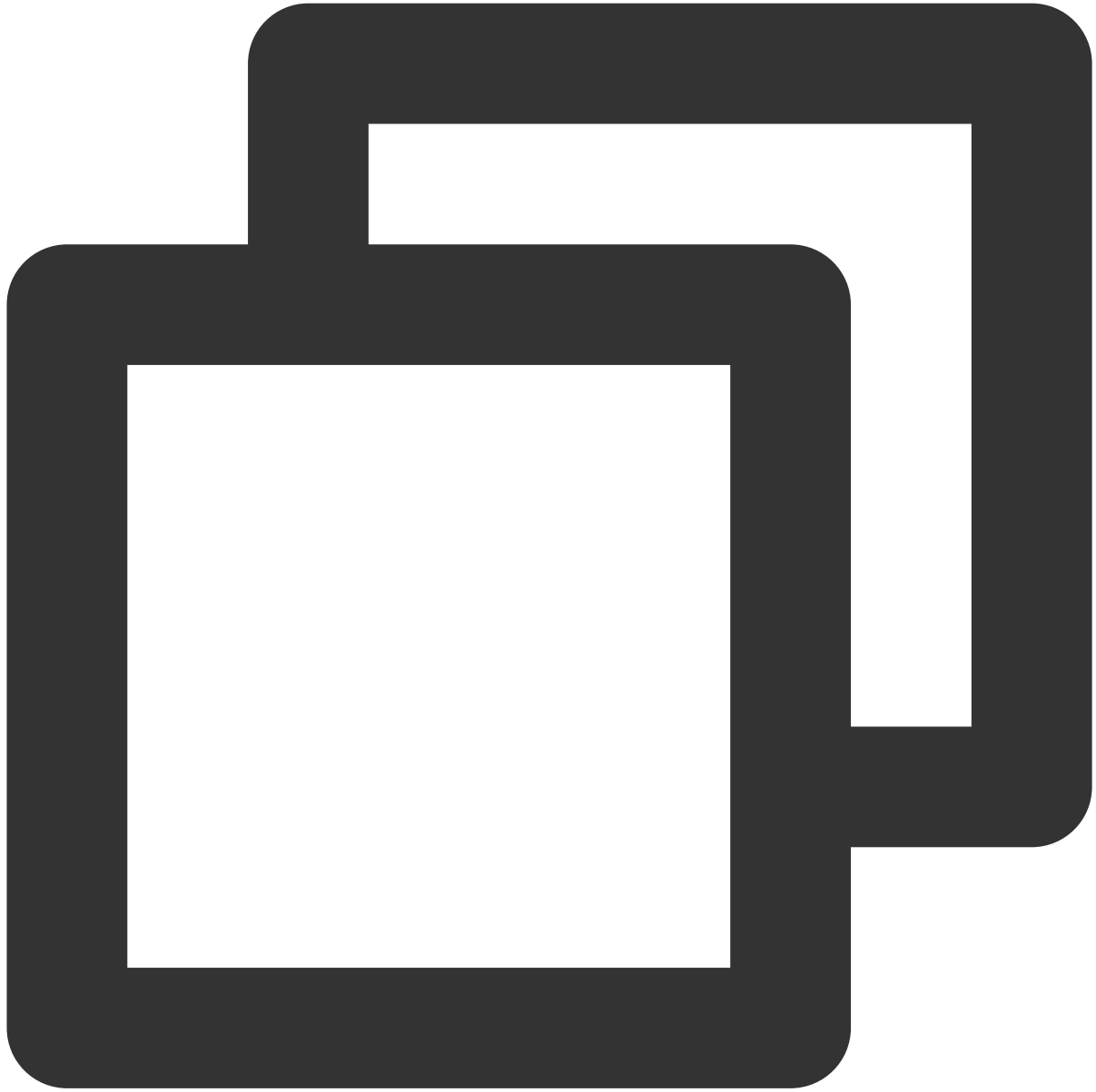


```
- (void)onAllUserCameraDisableChanged:(NSString *)roomId isDisable:(BOOL)isDisable;
```

Parameter	Type	Meaning
roomId	NSString *	Room ID
isDisable	BOOL	Whether it is disabled

### onSendMessageForAllUserDisableChanged

Inside the room, all users' text message sending is disabled event.



```
- (void)onSendMessageForAllUserDisableChanged:(NSString *)roomId isDisable:(BOOL)is
```

### **onKickedOutOfRoom**

Kicked out of the room event.



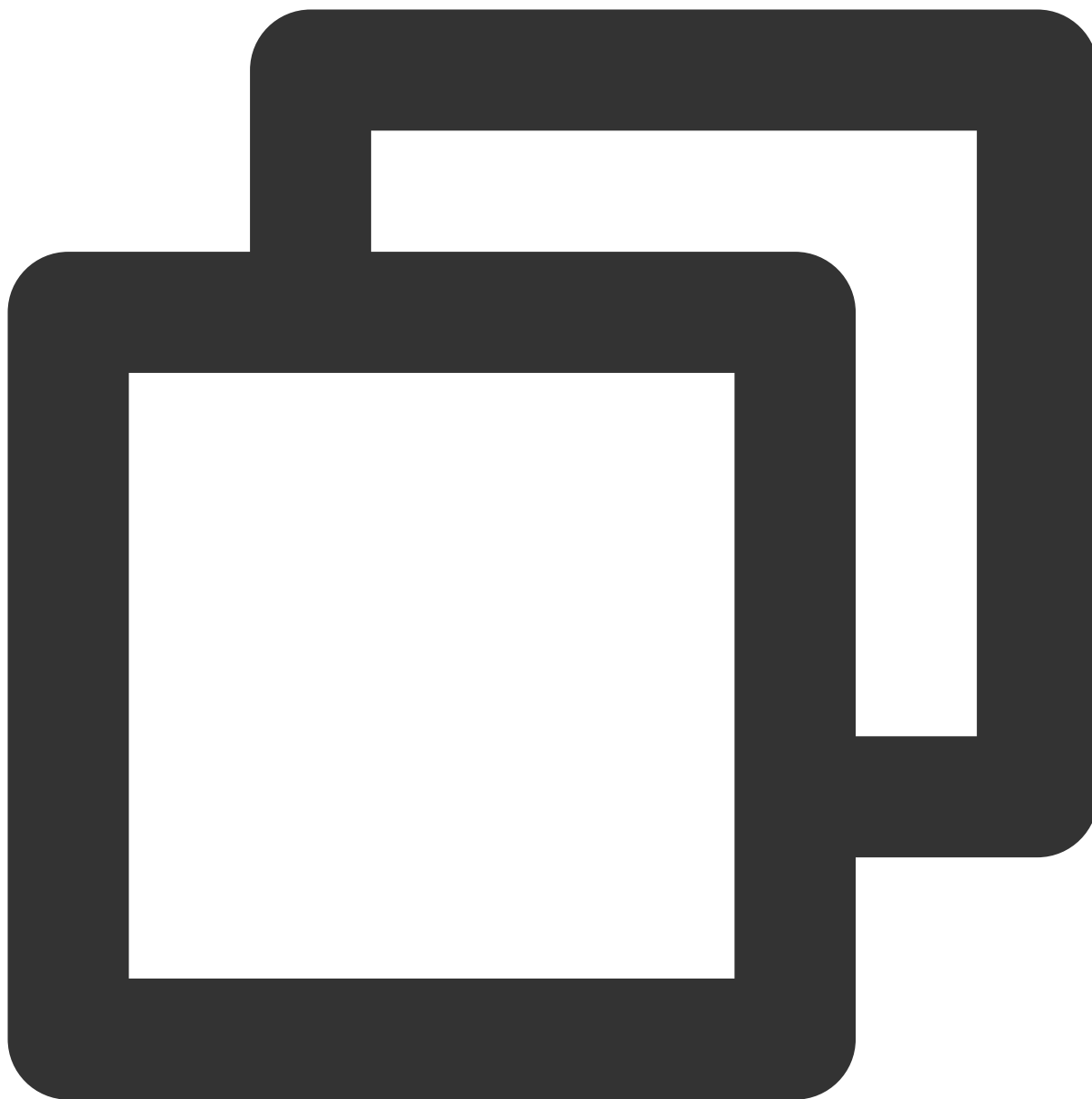
```
- (void)onKickedOutOfRoom:(NSString *)roomId message:(NSString *)message;
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	NSString *	Room ID
message	NSString *	Description of being kicked out

## onRoomDismissed

Room dissolution event.



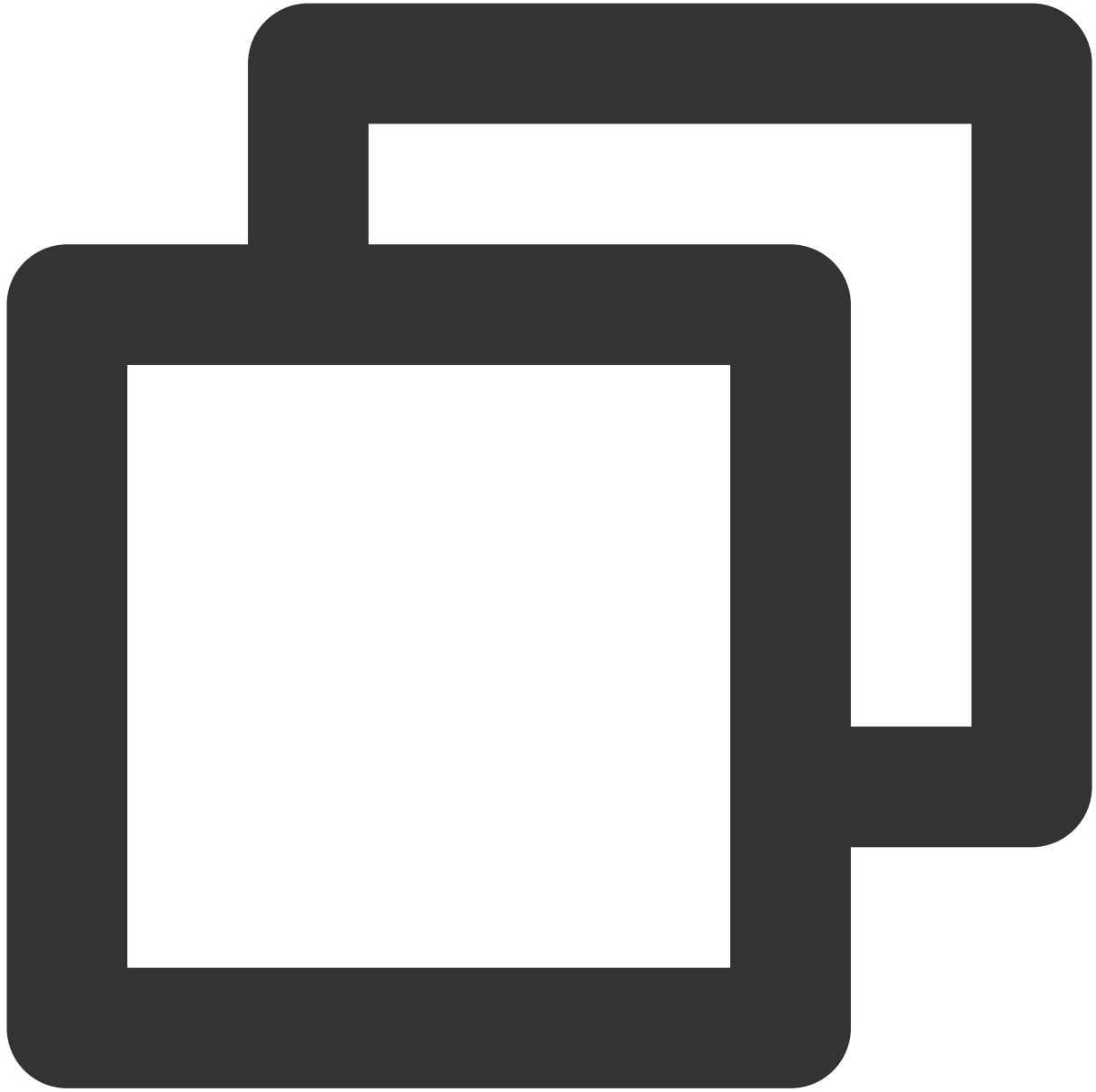
```
- (void)onRoomDismissed:(NSString *)roomId;
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	NSString *	Room ID

**onRoomSpeechModeChanged**

Mic control mode changes in the room.



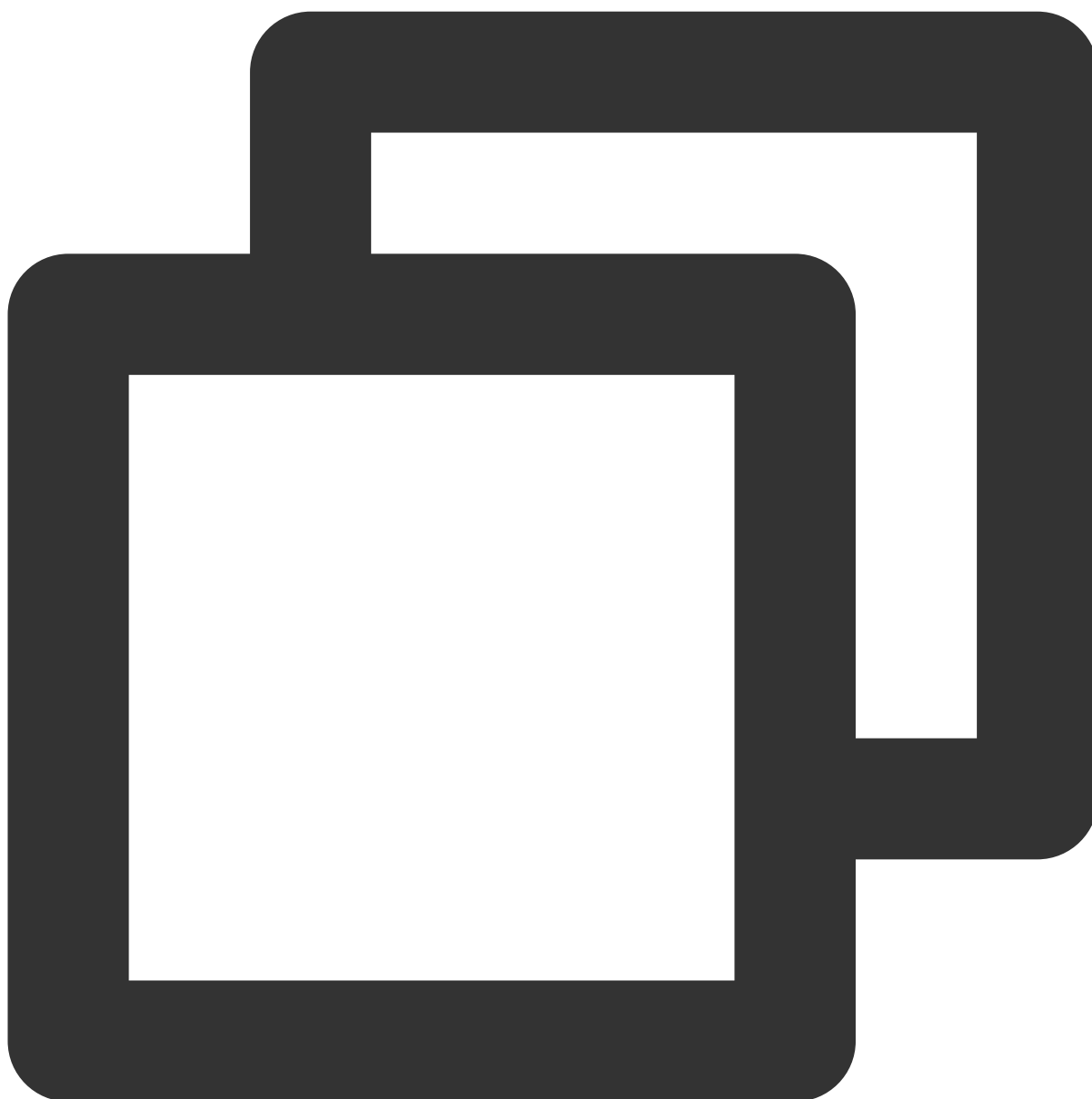
```
- (void)onRoomSpeechModeChanged:(NSString *)roomId speechMode:(TUISpeechMode)mode;
```

Parameter	Type	Meaning
roomId	NSString *	Room ID
mode	<a href="#">TUISpeechMode</a>	Mic control mode

## onRemoteUserEnterRoom

Remote user enters the room event.





```
- (void)onRemoteUserEnterRoom:(NSString *)roomId userInfo:(TUIUserInfo *)userInfo;
```

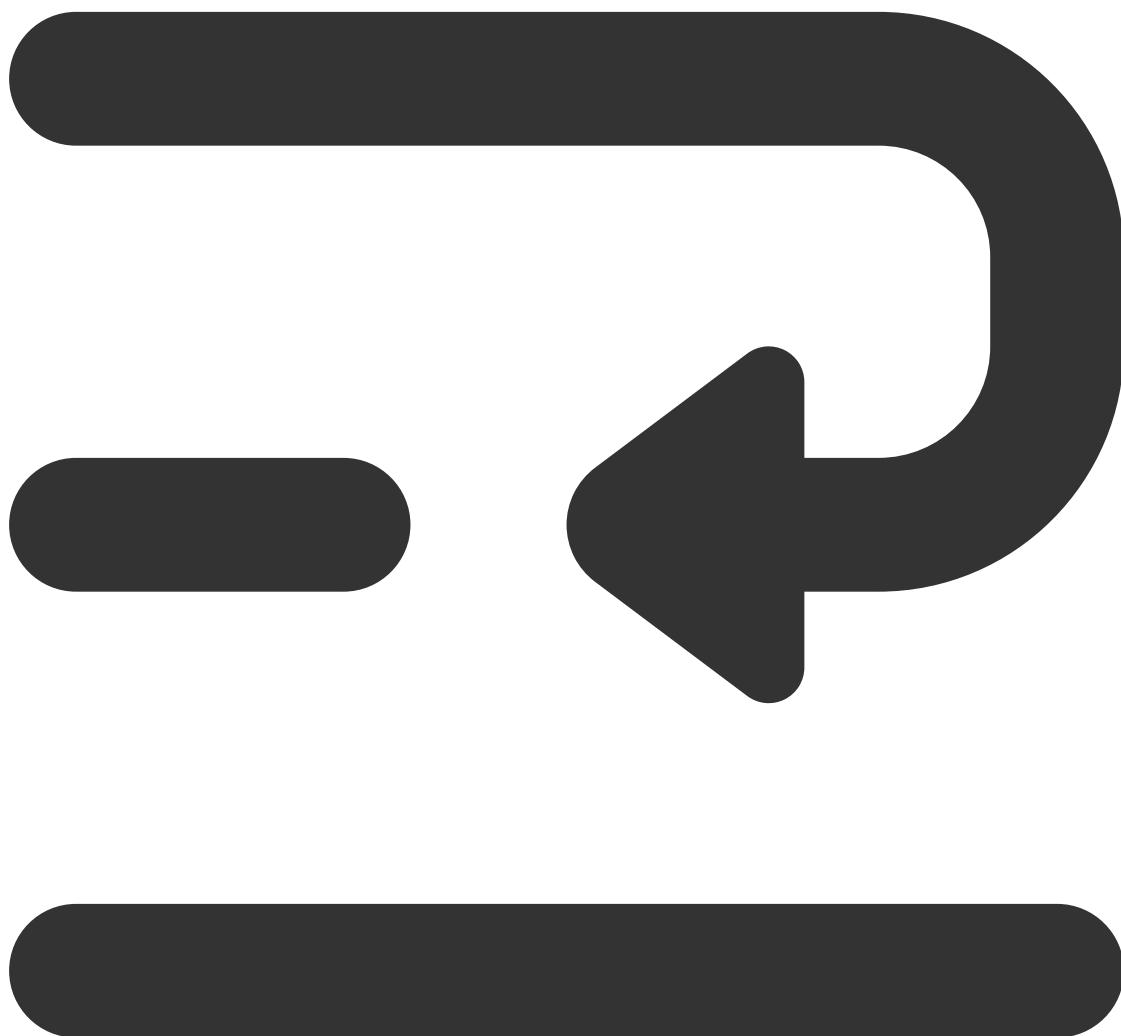
The parameters are as follows:

Parameter	Type	Meaning
roomId	NSString *	Room ID
userInfo	<a href="#">TUIUserInfo</a> *	User information

## onRemoteUserLeaveRoom



Remote user leaves the room event.





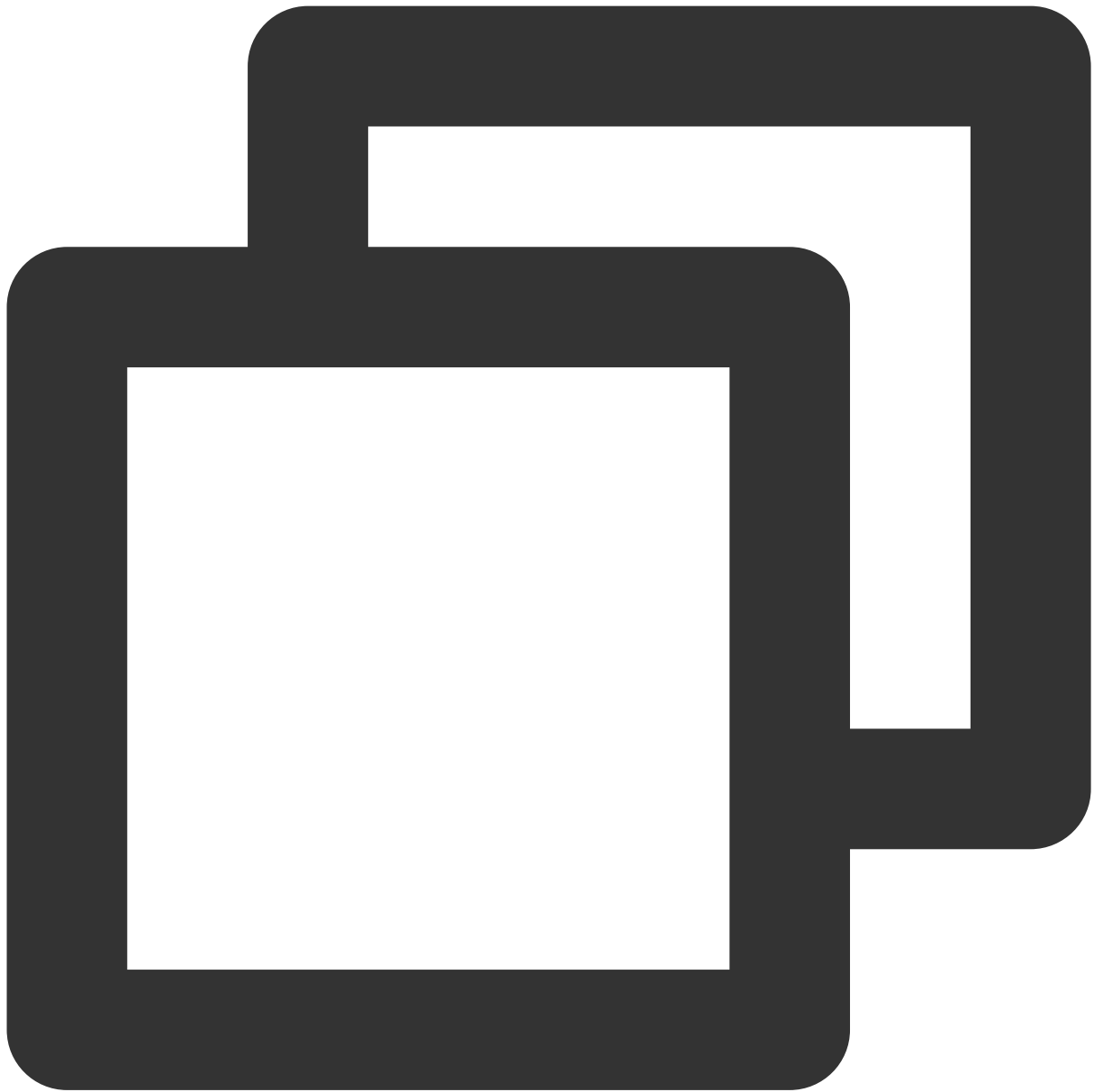
```
- (void)onRemoteUserLeaveRoom:(NSString *)roomId userInfo:(TUIUserInfo *)userInfo;
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	NSString *	Room ID
userInfo	<a href="#">TUIUserInfo</a> *	User information

## onUserRoleChanged

User role changes event.



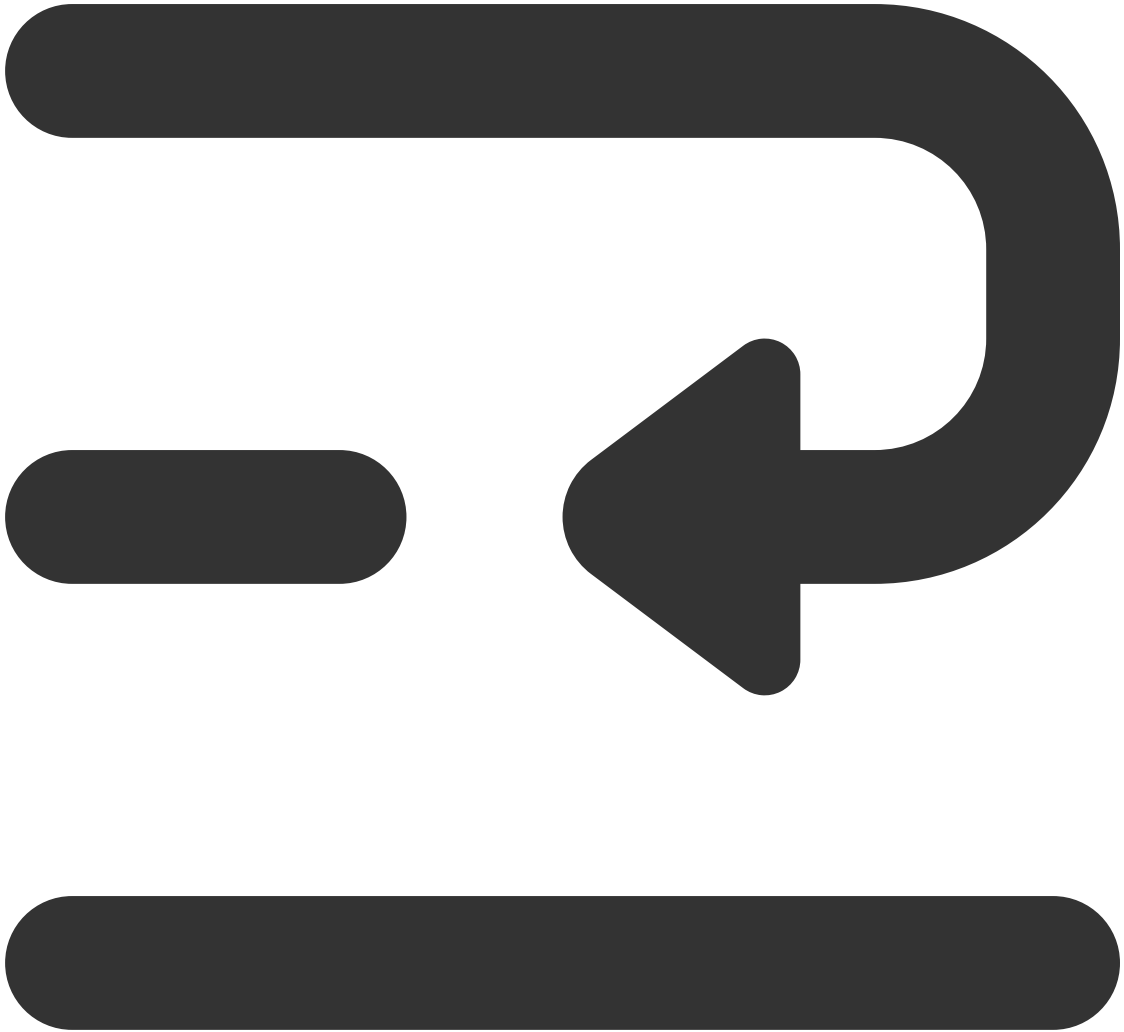
```
- (void)onUserRoleChanged:(NSString *)userId userRole:(TUIRole)userRole;
```

The parameters are as follows:

Parameter	Type	Meaning
userId	NSString *	User ID
userRole	<a href="#">TUIRole</a>	User Role

## onUserVideoStateChanged

User Video status changes event.





```
- (void)onUserVideoStateChanged:(NSString *)userId
    streamType:(TUIVideoStreamType)streamType
    hasVideo:(BOOL)hasVideo
    reason:(TUIChangeReason)reason;
```

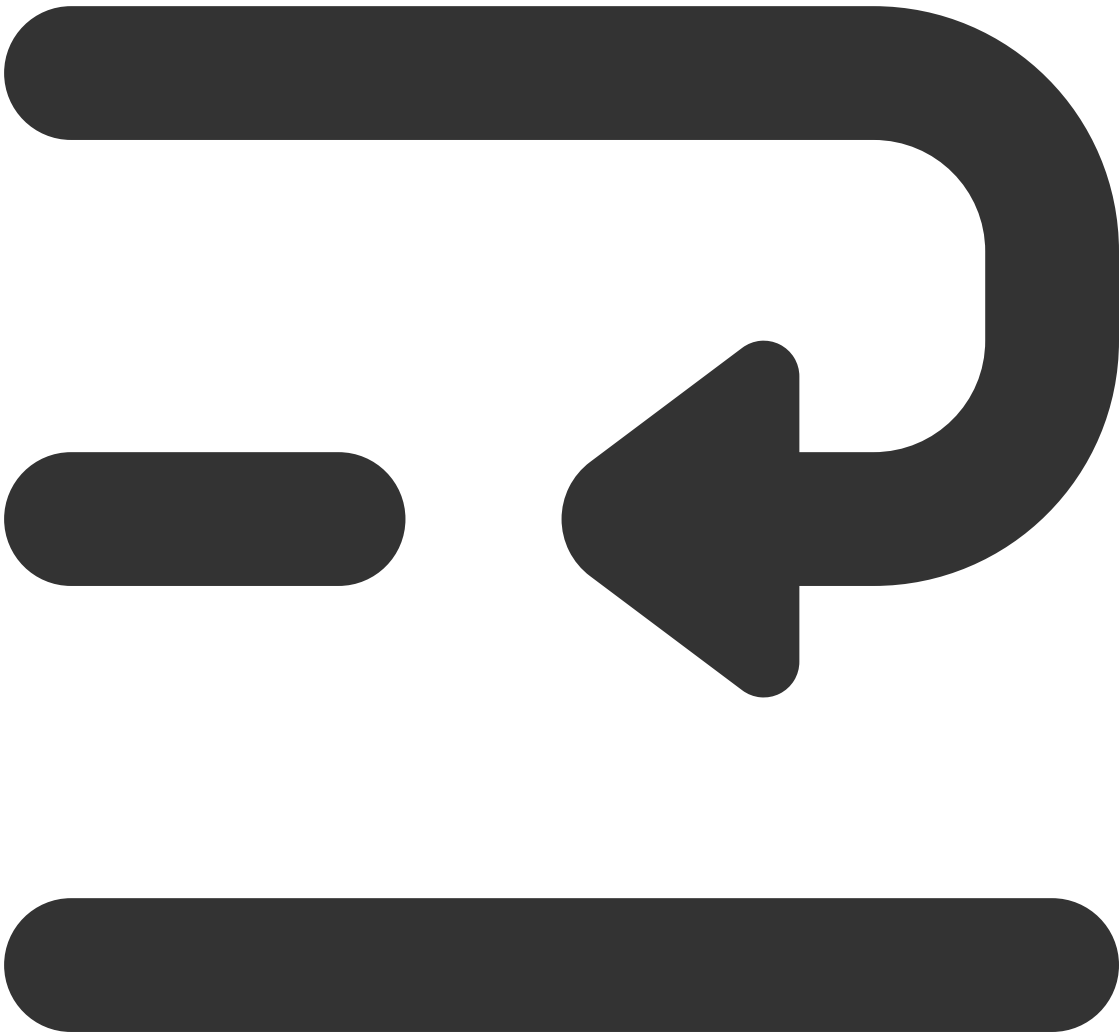
The parameters are as follows:

Parameter	Type	Meaning
userId	NSString *	User ID

streamType	<a href="#">TUIVideoStreamType</a>	Streams type
hasVideo	BOOL	Whether there are streams
reason	<a href="#">TUIChangeReason</a>	Reason for streams change

### onUserAudioStateChanged

User Audio status changes event.





```
- (void)onUserAudioStateChanged:(NSString *)userId
    hasAudio:(BOOL)hasAudio
    reason:(TUIChangeReason) reason;
```

**The parameters are as follows:**

Parameter	Type	Meaning
userId	NSString *	User ID
hasAudio	BOOL	Whether there are Audio streams

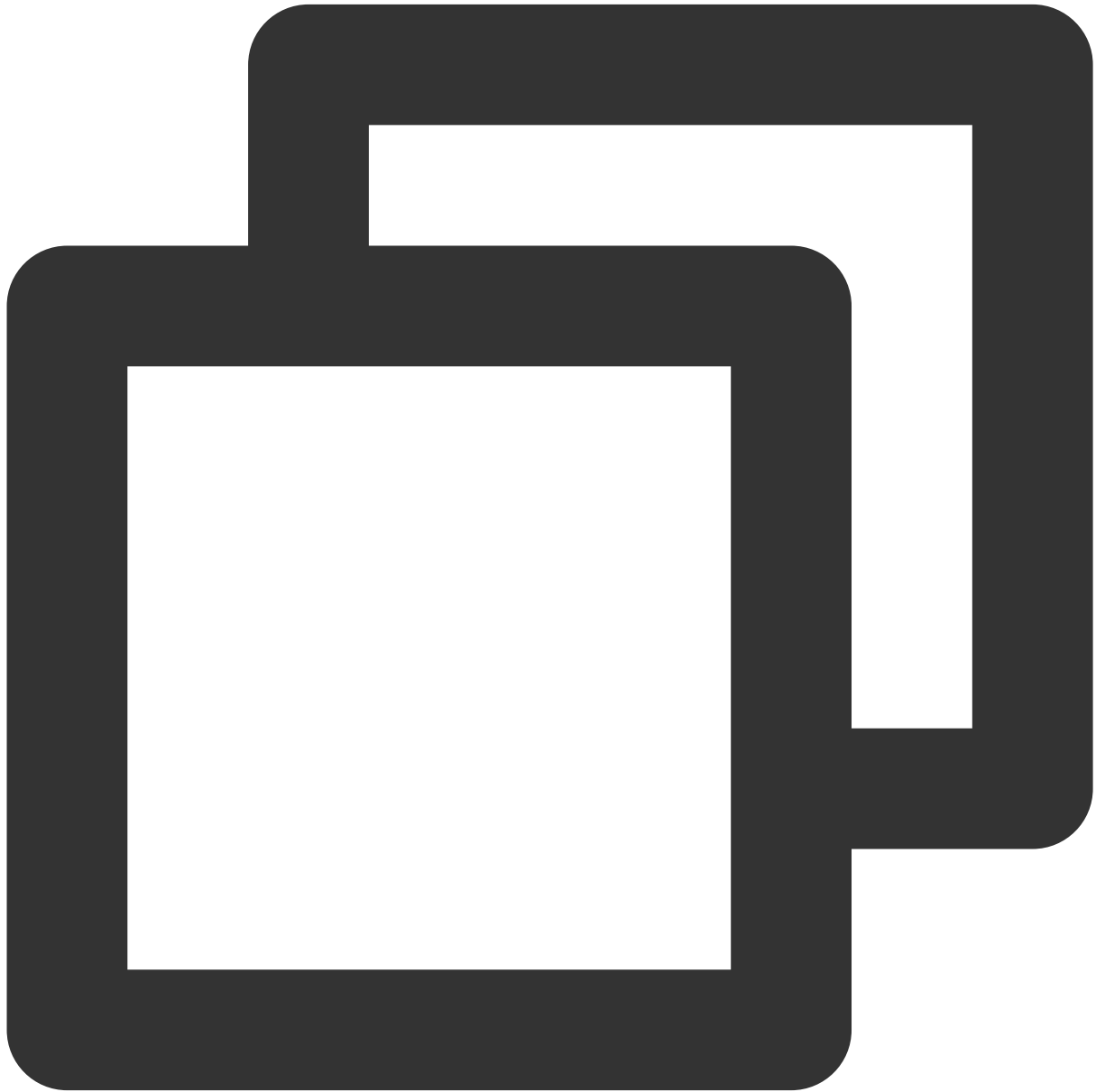
reason

TUIChangeReason

Reason for Audio streams change

## onUserScreenCaptureStopped

Screen Sharing stopped Callback event.



```
- (void)onUserScreenCaptureStopped: (NSInteger) reason;
```

The parameters are as follows:

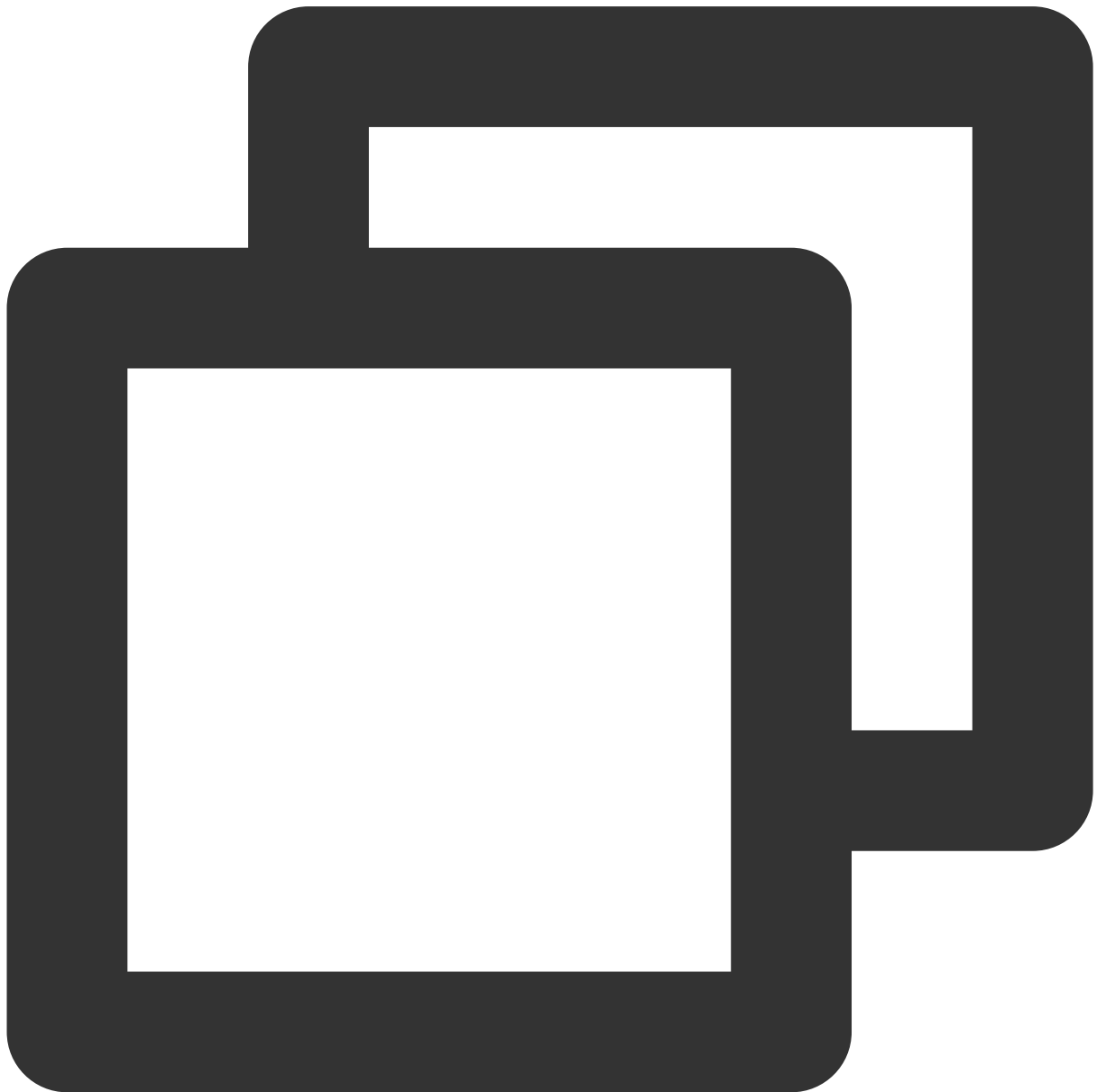
Parameter	Type	Meaning



reason	NSInteger	Stop reason: 0: User actively stops 1: Screen window closing causes the stop 2: Screen Sharing display screen status change (such as interface being unplugged, Projection mode change, etc.)
--------	-----------	--

## onRoomMaxSeatCountChanged

Maximum number of mic slots changes event in the room (only effective in meeting type rooms).



```
- (void)onRoomMaxSeatCountChanged:(NSString *)roomId maxSeatNumber:(NSInteger)maxSe
```

Parameter	Type	Meaning
roomId	NSString *	Room ID
maxSeatNumber	NSInteger	Maximum number of mic slots in the room

### **onUserVoiceVolumeChanged**

User volume change event.



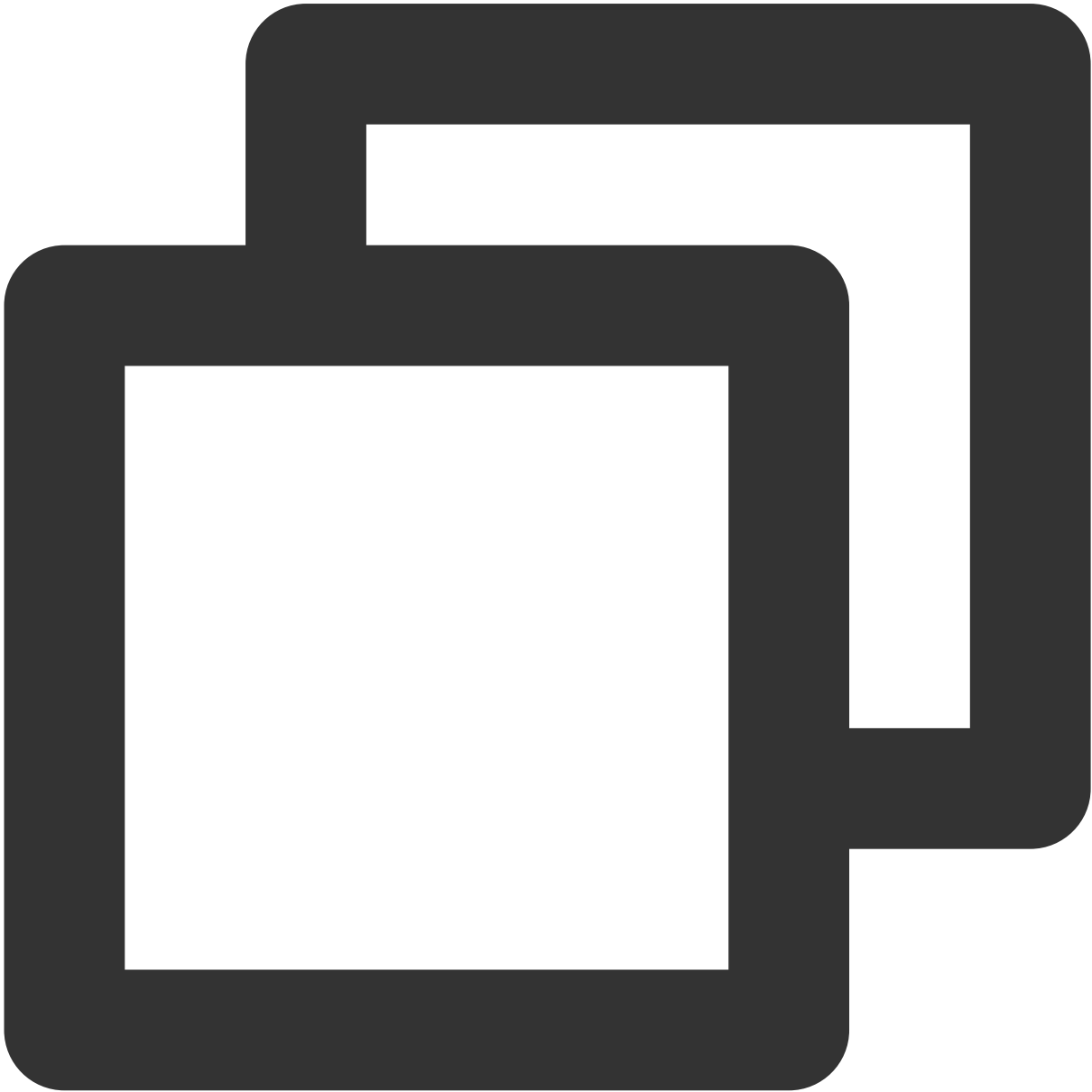
```
- (void)onUserVoiceVolumeChanged:(NSDictionary<NSString *, NSNumber *> *)volumeMap;
```

The parameters are as follows:

Parameter	Type	Meaning
volumeMap	NSDictionary<NSString *, NSNumber *> *	Volume map

onSendMessageForUserDisableChanged

User text message sending ability changes event.



```
- (void)onSendMessageForUserDisableChanged:(NSString *)roomId
    userId:(NSString *)userId
    isDisable:(BOOL)muted;
```

Parameter	Type	Meaning
roomId	NSString *	Room ID
userId	NSString *	User ID
muted	BOOL	Whether it is prohibited to send text messages.

### onUserNetworkQualityChanged

User network status change event.



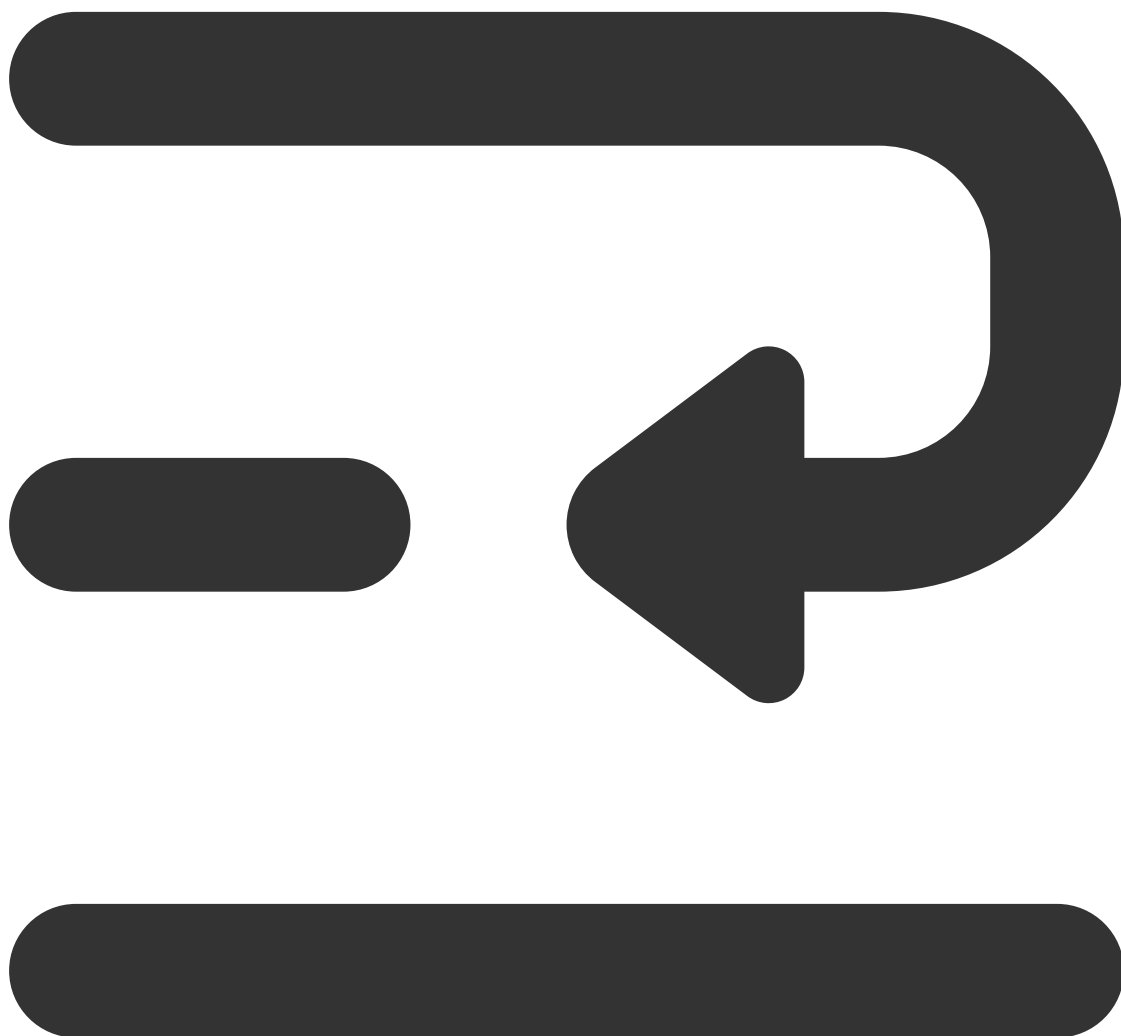
```
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkInfo *> *)networkList;
```

The parameters are as follows:

Parameter	Type	Meaning
networkList	NSArray<TUINetworkInfo>*	Network status array, you can refer to <code>TUINetworkInfo</code> object

## onSeatListChanged

Mic slot list changes event.





```
- (void)onSeatListChanged:(NSArray<TUISeatInfo *> *)seatList  
    seated:(NSArray<TUISeatInfo *> *)seatedList  
    left:(NSArray<TUISeatInfo *> *)leftList;
```

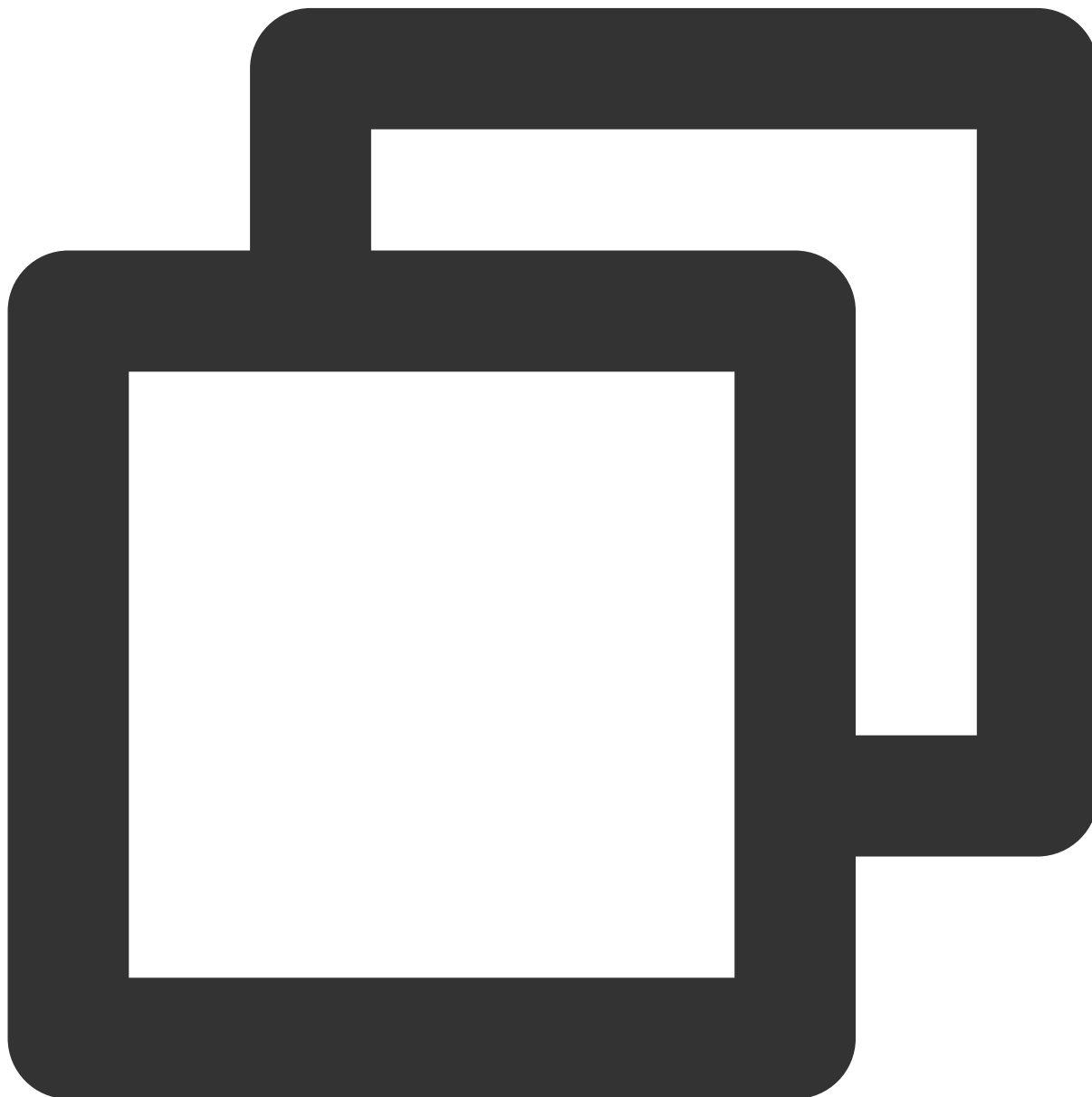
The parameters are as follows:

Parameter	Type	Meaning
seatList	NSArray<TUISeatInfo *> *	The latest user list on the mic, including newly on mic users

seatedList	NSArray<TUISeatInfo *> *	Newly on mic user list
leftList	NSArray<TUISeatInfo *> *	Newly off mic user list

## onKickedOffSeat

Received the event of user being kicked off mic.



```
- (void)onKickedOffSeat:(NSString *)userId;
```

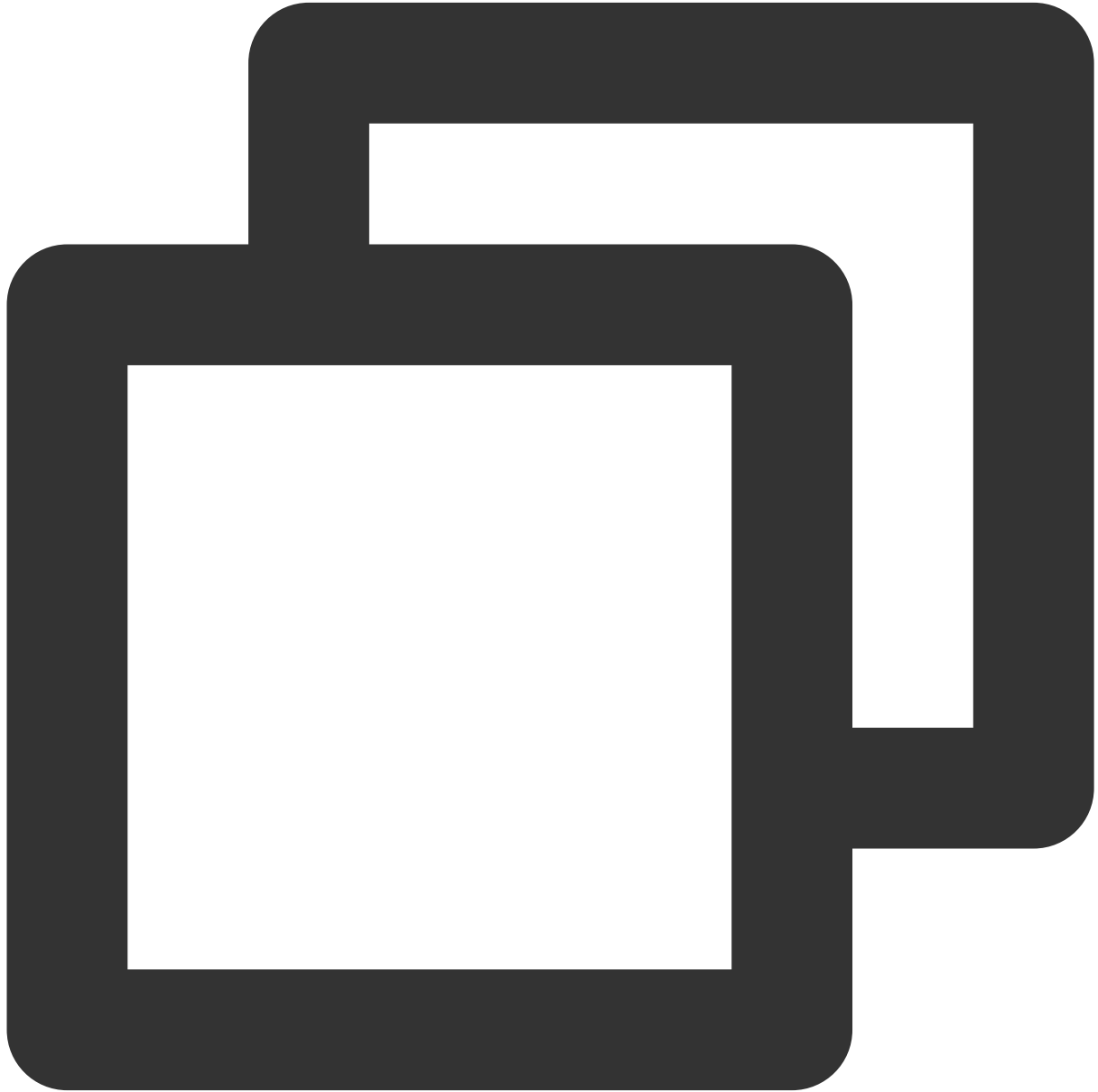
Parameter	Type	Meaning
-----------	------	---------



userId	NSString *	User ID
--------	------------	---------

## onRequestReceived

Received request message event.



```
- (void)onRequestReceived:(TUIRequest *)request;
```

The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

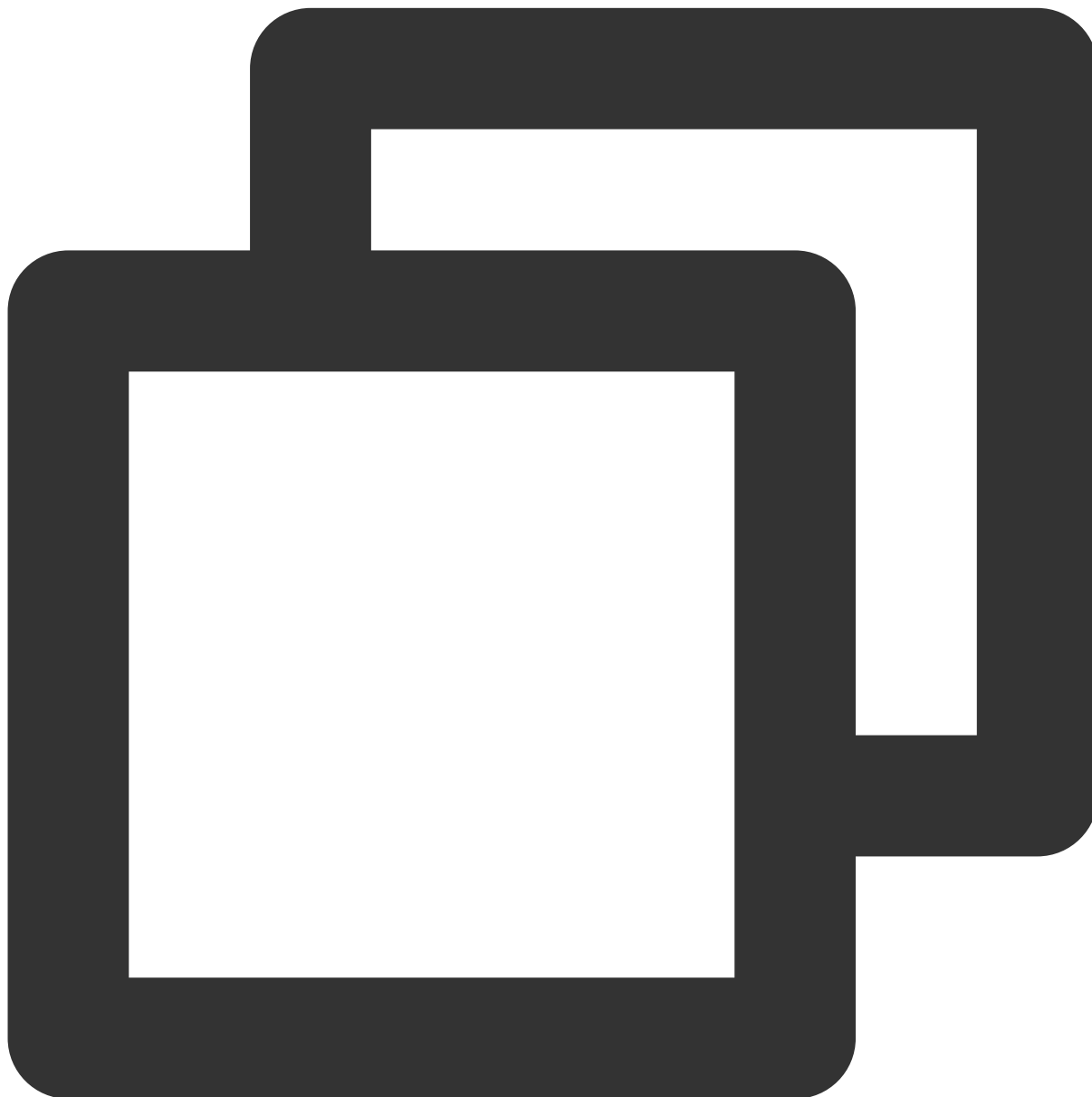
request

TUIRequest \*

Request content

## onRequestCancelled

Received request cancellation event.



```
- (void)onRequestCancelled:(NSString *)requestId;
```

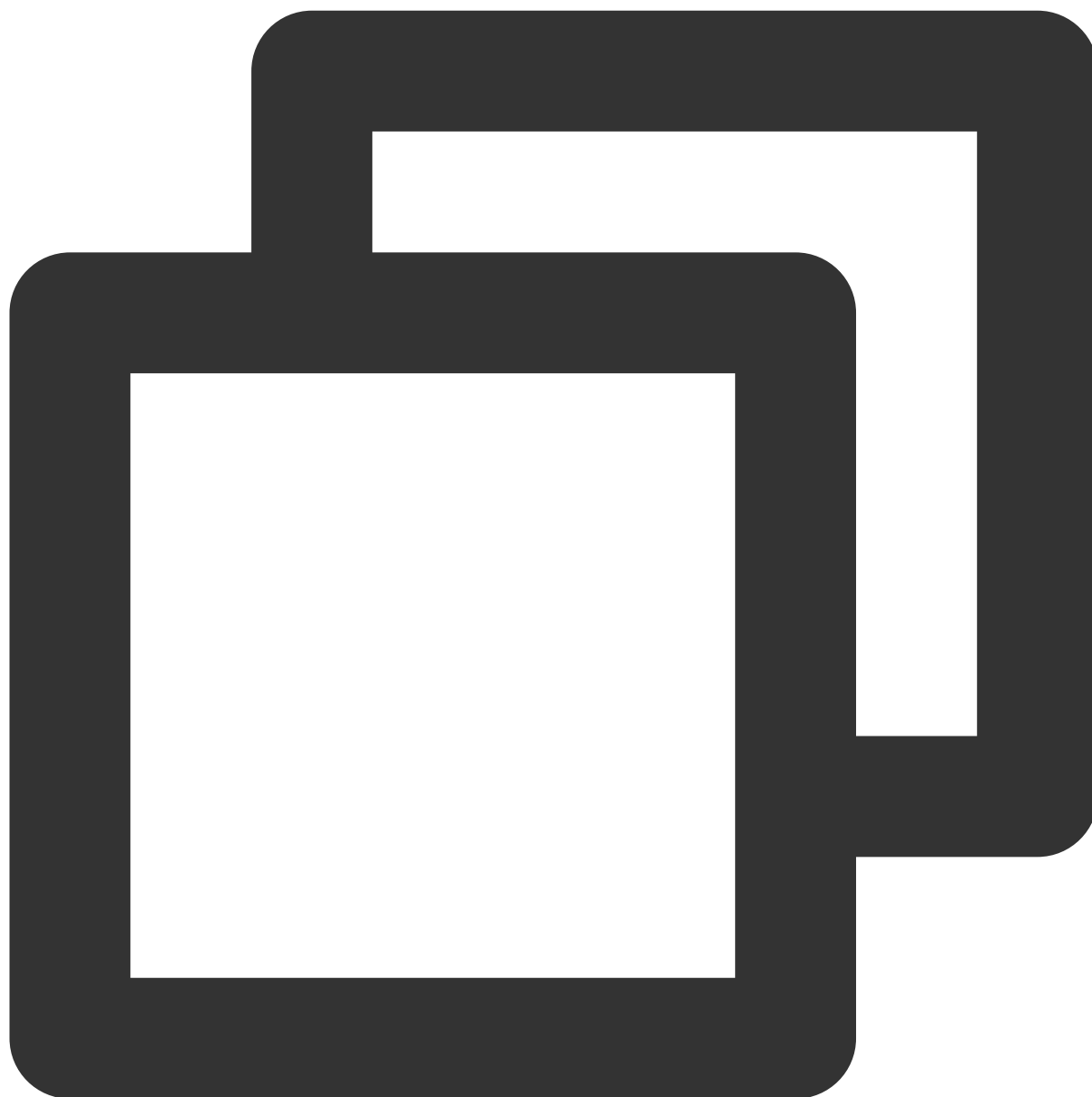
The parameters are as follows:

Parameter	Type	Meaning

requestId	NSString *	Request ID
-----------	------------	------------

## onReceiveTextMessage

Received ordinary text message event.



```
- (void)onReceiveTextMessage:(NSString *)roomId  
    message:(TUIMessage *)message;
```

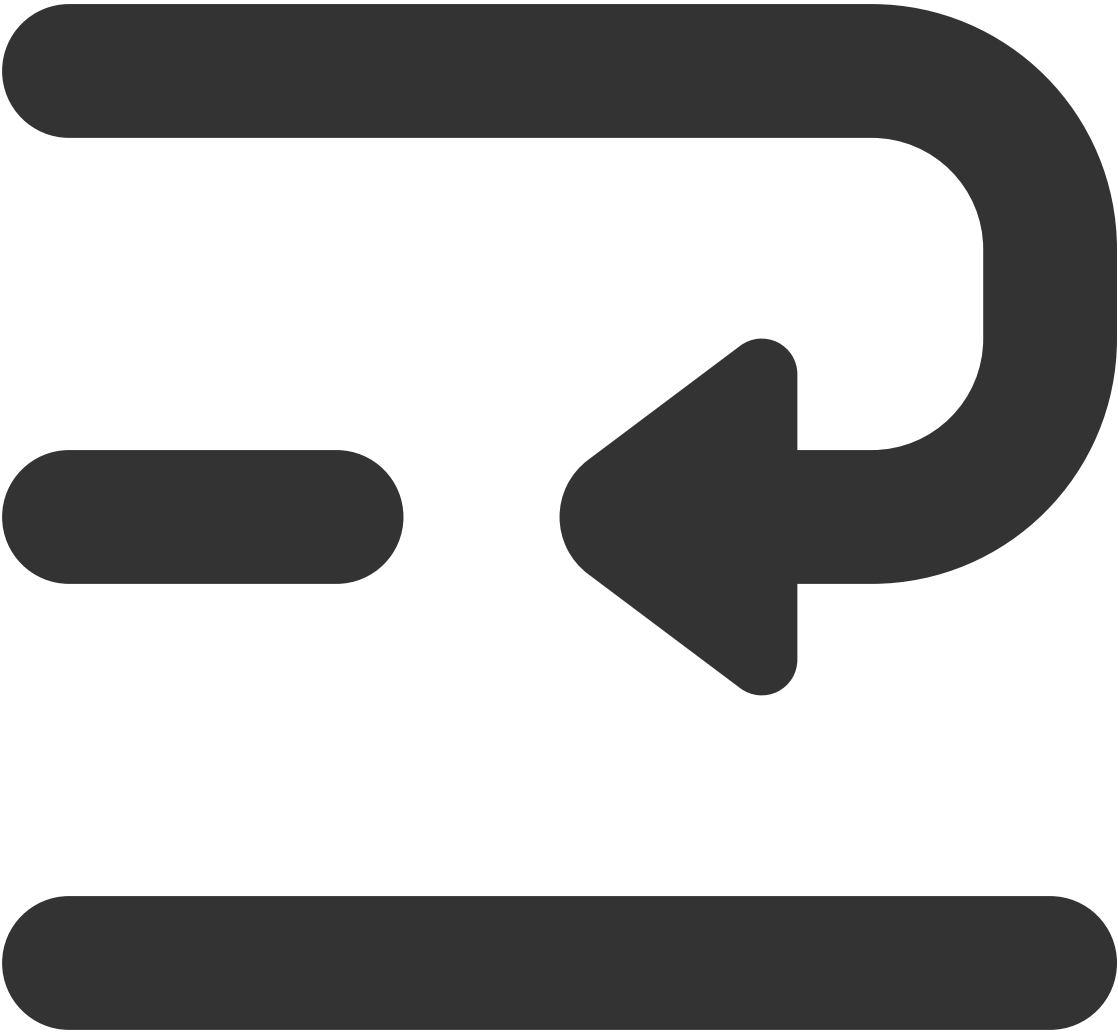
The parameters are as follows:

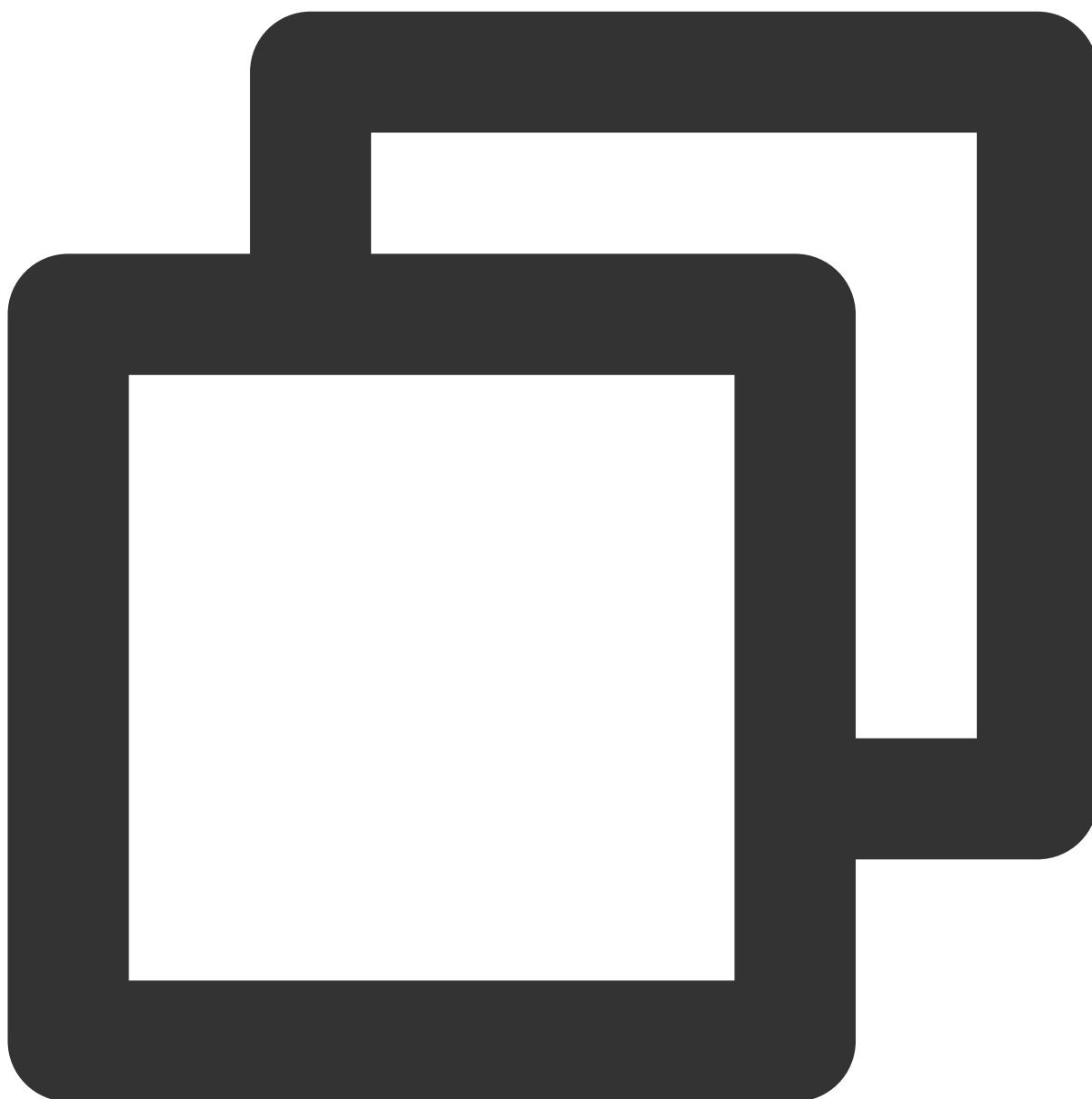
Parameter	Type	Meaning
-----------	------	---------

roomId	NSString *	Room ID
message	<a href="#">TUIMessage</a>	Message content, detailed definition can refer to <code>TUIRoomDefine.h</code> file in TUIMessage definition

**onReceiveCustomMessage**

Received custom message event.





```
- (void)onReceiveCustomMessage:(NSString *)roomId  
    message:(TUIMessage *)message;
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	NSString *	Room ID
message	<a href="#">TUIMessage</a>	Message content



# Type Definition

Last updated : 2023-12-18 18:05:05

## Enumeration Definition

### TUIRoomDefine

Type	Description
<a href="#">TUIRoomType</a>	Room Type
<a href="#">TUISpeechMode</a>	Mic Control Mode
<a href="#">TUIMediaDevice</a>	Room Media Device Type
<a href="#">TUIRole</a>	Room Role Type
<a href="#">TUIVideoQuality</a>	Video Quality
<a href="#">TUIAudioQuality</a>	Audio Quality
<a href="#">TUIVideoStreamType</a>	Video Stream Type
<a href="#">TUIChangeReason</a>	Change Reason (User audio and video status change operation reason: self-modification or modified by room owner/administrator)
<a href="#">TUICaptureSourceType</a>	Screen Sharing Capture Source Type
<a href="#">TUIRequestAction</a>	Request Type

### TUICommonDefine

Type	Description
<a href="#">TUINetworkQuality</a>	Network Quality

## Common Structure

### TUIRoomDefine

Type	Description
<a href="#">TUIRoomInfo</a>	Room data
<a href="#">TUILoginUserInfo</a>	User Login Information

<a href="#">TUIUserInfo</a>	Room User Information
<a href="#">TUISeatInfo</a>	Room Seat Information
<a href="#">TUISeatLockParams</a>	Lock Seat Operation Parameters
<a href="#">TUIUserVoiceVolume</a>	Room User Volume
<a href="#">TUIRequest</a>	Signaling Request
<a href="#">TUIShareTarget</a>	Screen Sharing Capture Source Information

### TUICommonDefine

Type	Description
<a href="#">TUINetworkInfo</a>	Network Quality Information
<a href="#">TUIMessage</a>	Message

### TUIRoomType

Room Type

Enumeration	Value	Description
TUIRoomTypeConference	1	Conference Type Room, suitable for conference and education scenarios, this room can enable free speech, apply for speech, go live and other modes.
TUIRoomTypeLivingRoom	2	Live Type Room, suitable for live broadcast scenarios, this room can enable free speech, mic control mode, and the seats in this room are numbered.

### TUISpeechMode

Mic Control Mode

Enumeration	Value	Description
TUISpeechModeFreeToSpeak	1	Free speech mode.
TUISpeechModeApplyToSpeak	2	Apply to speak mode. (Only effective in conference type room)
TUISpeechModeApplySpeakAfterTakingSeat	3	Go Live mode.



## TUIMediaDevice

### Room Media Device Type

Enumeration	Value	Description
TUIMediaDeviceMicrophone	1	Mic
TUIMediaDeviceCamera	2	Camera
TUIMediaDeviceApplyScreenSharing	3	Screen Sharing

## TUIRole

### Room Role Types

Enumeration	Value	Description
TUIRoleRoomOwner	0	Room Owner, generally refers to the creator of the room, the highest authority holder in the room
TUIRoleAdministrator	1	Room Administrator
TUIRoleGeneralUser	2	General Member in the room

## TUIVideoQuality

### Video Quality

Enumeration	Value	Description
TUIVideoQuality360P	1	Low-quality 360P
TUIVideoQuality540P	2	Standard Definition 540P
TUIVideoQuality720P	3	High Definition 720P
TUIVideoQuality1080P	4	Ultra-clear 1080P

## TUIAudioQuality

### Audio Quality

Enumeration	Value	Description
TUIAudioQualitySpeech	0	Speech Mode

TUIAudioQualityDefault	1	Default Mode
TUIAudioQualityMusic	2	Music Mode

## TUIVideoStreamType

Video Stream Type

Enumeration	Value	Description
TUIVideoStreamTypeCameraStream	0	High-quality Camera Video Stream
TUIVideoStreamTypeScreenStream	1	Screen Sharing Video Stream
TUIVideoStreamTypeCameraStreamLow	2	Low-quality Camera Video Stream

## TUIChangeReason

Change Reason (User audio and video status change operation reason: self-modification or modification by room owner/administrator)

Enumeration	Value	Description
TUIChangeReasonBySelf	0	Self-operation
TUIChangeReasonByAdmin	1	Room Owner or Administrator Operation

## TUICaptureSourceType

Screen Sharing Capture Source Type

Enumeration	Value	Description
TUICaptureSourceTypeUnknown	-1	Undefined
TUICaptureSourceTypeWindow	0	Window
TUICaptureSourceTypeScreen	1	Screen

## TUIRequestAction

Request Type

Enumeration	Value	Description
TUIRequestActionInvalidAction	0	Invalid Request
TUIRequestActionOpenRemoteCamera	1	Request Remote User to

		Open Camera
TUIRequestActionOpenRemoteMicrophone	2	Request Remote User to Open Microphone
TUIRequestActionConnectOtherRoom	3	Request to Connect to Other Room
TUIRequestActionTakeSeat	4	Request to Go Live
TUIRequestActionRemoteUserOnSeat	5	Request Remote User to Go Live
TUIRequestActionApplyToAdminToOpenLocalCamera	6	Request to Admin to Open Local Camera
TUIRequestActionApplyToAdminToOpenLocalMicrophone	7	Request to Admin to Open Local Microphone

## TUINetworkQuality

### Network Quality

Enumeration	Value	Description
TUINetworkQualityUnknown	0	Undefined
TUINetworkQualityExcellent	1	Current Network is Excellent
TUINetworkQualityGood	2	Current Network is Good
TUINetworkQualityPoor	3	Current Network is Average
TUINetworkQualityBad	4	Current Network is Poor
TUINetworkQualityVeryBad	5	Current Network is Very Poor
TUINetworkQualityDown	6	Current Network Does Not Meet TRTC's Minimum Requirements

## TUIRoomInfo

### Room Information

Field	Type	Description
roomId	NSString *	Room ID
roomType	<a href="#">TUIRoomType</a>	Room Type

ownerId	NSString *	Host ID, default is the room creator (read-only)
name	NSString *	Room Name, default is the room ID
speechMode	TUISpeechMode	Mic Control Mode
createTime	NSUInteger	Room Creation Time (read-only)
memberCount	NSInteger	Number of Members in the Room (read-only)
maxSeatCount	NSUInteger	Maximum Number of Mic Seats (only supported when entering the room and creating the room)
isCameraDisableForAllUser	BOOL	Whether to Disable Opening Camera (optional when creating a room), default value: false
isMicrophoneDisableForAllUser	BOOL	Whether to Disable Opening Microphone (optional when creating a room), default value: false
isMessageDisableForAllUser	BOOL	Whether to Disable Sending Messages (optional when creating a room), default value: false
enableCDNStreaming	BOOL	Whether to Enable CDN Live Streaming (optional when creating a room, for live streaming rooms), default value: false
cdnStreamDomain	NSString*	Live Streaming Push Domain (optional when creating a room, for live streaming rooms), default value: empty

## TUILoginUserInfo

User Login Information

Field	Type	Meaning
userId	NSString *	User ID
userName	NSString *	User Name
avatarUrl	NSString *	User Avatar URL

## TUIUserInfo

User Information in the Room

Field	Type	Description
userId	NSString *	User ID
userName	NSString *	User Name
avatarUrl	NSString *	User Avatar URL
userRole	TUIRole	User Role Type
hasAudioStream	BOOL	Whether There is Audio Stream, default value: false
hasVideoStream	BOOL	Whether There is Video Stream, default value: false
hasScreenStream	BOOL	Whether There is Screen Sharing Stream, default value: false

## TUISeatInfo

Seat Information in the Room

Field	Type	Description
index	NSInteger	Mic Seat Number
userId	NSString *	User ID
isLocked	BOOL	Whether the Mic Seat is Locked, default false
isVideoLocked	BOOL	Whether the Mic Seat is Prohibited from Opening Camera, default false
isAudioLocked	BOOL	Whether the Mic Seat is Prohibited from Opening Microphone, default false

## TUISeatLockParams

Lock Seat Operation Parameters

Field	Type	Meaning

lockSeat	BOOL	Lock Mic Seat, default false
lockVideo	BOOL	Lock Mic Seat Camera, default false
lockAudio	BOOL	Lock Mic Seat Microphone, default false

## TUIUserVoiceVolume

User Voice Volume in the Room

Field	Type	Description
userId	NSString *	User ID
volume	NSUInteger	Volume Size, Value range 0 - 100

## TUIRequest

Signaling Request

Field	Type	Description
requestId	NSString *	Request ID
requestAction	<a href="#">TUIRequestAction</a>	Request Type
userId	NSString *	User ID
content	NSString *	Signaling Content
timestamp	NSUInteger	Timestamp

## TUIShareTarget

Screen Sharing Capture Source Information

Field	Type	Description
targetId	NSString *	Capturing Source ID, for windows, this field represents the window ID; for screens, this field represents the display ID
sourceType	<a href="#">TUICaptureSourceType</a>	Capturing Source Type
sourceName	NSString *	Capturing Source Name
thumbnailImage	TUIImage *	Thumbnail

iconImage	TUIImage *	Icon
extInfo	NSDictionary *	Window's Extended Information

## TUINetworkInfo

Network Quality Information

Field	Type	Description
userId	NSString *	User ID
quality	<a href="#">TUINetworkQuality</a>	Network Quality
upLoss	uint32_t	Upstream Packet Loss Rate
downLoss	uint32_t	Downstream Packet Loss Rate
delay	uint32_t	Network Delay

## TUIMessage

Message

Field	Type	Description
messageId	NSString *	Message ID
message	NSString *	Message Text
timestamp	uint64_t	Message Time
userId	NSString *	Message Sender
userName	NSString *	Message Sender Nickname
avatarUrl	NSString *	Message Sender Avatar

# Android

## API Overview

Last updated : 2023-10-24 17:16:07

### TUIRoomEngine (No UI Interface)

TUIRoomEngine API is the Audio/Video call Component's No UI Interface, you can use this set of API to customize packaging according to your business needs.

TUIRoomEngine

#### TUIRoomEngine Core Methods

API	Description
<a href="#">createInstance</a>	Create TUIRoomEngine Instance
<a href="#">destroyInstance</a>	Destroy TUIRoomEngine Instance
<a href="#">login</a>	Login interface, you need to initialize user information before entering the room and perform a series of operations.
<a href="#">logout</a>	Logout interface, there will be actively leave room operation, destroy resources
<a href="#">setSelfInfo</a>	Set local user name and avatar
<a href="#">getSelfInfo</a>	Get local user basic information
<a href="#">addObserver</a>	Set event callback
<a href="#">removeObserver</a>	Remove event callback

#### Room Related Active Interface

API	Description
<a href="#">createRoom</a>	Create room
<a href="#">destroyRoom</a>	Close the room
<a href="#">enterRoom</a>	Entered room
<a href="#">exitRoom</a>	Leave room



<a href="#">connectOtherRoom</a>	Connect to other room
<a href="#">disconnectOtherRoom</a>	Disconnect from other room
<a href="#">fetchRoomInfo</a>	Get room data
<a href="#">updateRoomNameByAdmin</a>	Update room name
<a href="#">updateRoomSpeechModeByAdmin</a>	Set room management mode (only administrator or group owner can call)

## Local User View Rendering, Video Management

API	Description
<a href="#">setLocalVideoView</a>	Set the view control for local user video rendering
<a href="#">openLocalCamera</a>	Open local camera
<a href="#">closeLocalCamera</a>	Close local camera
<a href="#">updateVideoQuality</a>	Update local video codec quality settings
<a href="#">startScreenSharing</a>	Start screen sharing
<a href="#">stopScreenSharing</a>	End screen sharing
<a href="#">startPushLocalVideo</a>	Start pushing local video
<a href="#">stopPushLocalVideo</a>	Stop pushing local video

## Local User Audio Management

API	Description
<a href="#">openLocalMicrophone</a>	Open local microphone
<a href="#">closeLocalMicrophone</a>	Close local microphone
<a href="#">updateAudioQuality</a>	Update local audio codec quality settings
<a href="#">startPushLocalAudio</a>	Start pushing local audio
<a href="#">stopPushLocalAudio</a>	Stop pushing local audio

## Remote User View Rendering, Video Management

API	Description
-----	-------------

<a href="#">setRemoteVideoView</a>	Set the view control for remote user video rendering
<a href="#">startPlayRemoteVideo</a>	Start playing remote user video
<a href="#">stopPlayRemoteVideo</a>	Stop playing remote user video
<a href="#">muteRemoteAudioStream</a>	Mute remote user

## Room User Information

API	Description
<a href="#">getUserList</a>	Get the member list in the room
<a href="#">getUserInfo</a>	Get member information

## Room User Management

API	Description
<a href="#">changeUserRole</a>	Modify user role (only administrator or group owner can call)
<a href="#">kickRemoteUserOutOfRoom</a>	Kick Remote User out of the Room (Only Administrator or Group Owner can call)

## Speech Management in Room

API	Description
<a href="#">disableDeviceForAllUserByAdmin</a>	Media Device Management for All Users (Only Administrator or Group Owner can call)
<a href="#">openRemoteDeviceByAdmin</a>	Request Remote User to Open Media Device (Only Administrator or Group Owner can call)
<a href="#">closeRemoteDeviceByAdmin</a>	Close Remote User's Media Device (Only Administrator or Group Owner can call)
<a href="#">applyToAdminToOpenLocalDevice</a>	Request to Open Local Media Device (Available for Ordinary Users)

## Microphone Seat Management in Room

API	Description
<a href="#">setMaxSeatCount</a>	Set Maximum Number of Microphone Seats (Only supported when entering the

	room and creating the room)
<a href="#">getSeatList</a>	Get Microphone Seat List
<a href="#">lockSeatByAdmin</a>	Lock Microphone Seat (Including Position Lock, Audio State Lock, Video State Lock)
<a href="#">takeSeat</a>	Go Live Locally Conference Scene: <a href="#">SPEAK_AFTER_TAKING_SEAT</a> mode requires application to the host or administrator to allow going live, other modes do not support going live. Live Broadcast Scene: <a href="#">FREE_TO_SPEAK</a> mode allows free going live, and speak after going live; <a href="#">SPEAK_AFTER_TAKING_SEAT</a> mode requires application to the host or administrator to allow going live; other modes do not support going live.
<a href="#">leaveSeat</a>	Leave Microphone Seat Locally
<a href="#">takeUserOnSeatByAdmin</a>	Host/Administrator invites user to go live
<a href="#">kickUserOffSeatByAdmin</a>	Host/Administrator kicks user off the microphone seat

## Signaling Management

API	Description
<a href="#">cancelRequest</a>	Cancel Request
<a href="#">responseRemoteRequest</a>	Reply to Request

## Send Message

API	Description
<a href="#">sendTextMessage</a>	Send Text Message
<a href="#">sendCustomMessage</a>	Send Custom Message
<a href="#">disableSendingMessageByAdmin</a>	Disable Remote User's Text Message Sending Ability (Only Administrator or Group Owner can call)
<a href="#">disableSendingMessageForAllUser</a>	Disable All Users' Text Message Sending Ability (Only Administrator or Group Owner can call) Advanced Feature: Get TRTC Instance

## Advanced Feature: Get TRTC Instance

API	Description
-----	-------------

<a href="#">getTRTCCloud</a>	Get TRTC Instance Object
<a href="#">getDeviceManager</a>	Get Device Management Object
<a href="#">getAudioEffectManager</a>	Get Audio Effect Management Object
<a href="#">getBeautyManager</a>	Get Beauty Management Object

## Event Type Definition

TUIRoomObserver is the Callback Event class corresponding to TUIRoomEngine. You can listen to the callback events you need through this callback.

TUIRoomObserver

### Error Callback

Event	Description
<a href="#">onError</a>	Error Callback Event

### Login Status Event Callback

API	Description
<a href="#">onKickedOffLine</a>	User Kicked Offline Event
<a href="#">onUserSigExpired</a>	User Credential Timeout Event

### Room Event Callback

API	Description
<a href="#">onRoomNameChanged</a>	Room Name Change Event
<a href="#">onAllUserMicrophoneDisableChanged</a>	All Users' Microphones Disabled in Room Event
<a href="#">onAllUserCameraDisableChanged</a>	All Users' Cameras Disabled in Room Event
<a href="#">onSendMessageForAllUserDisableChanged</a>	All Users' Text Message Sending Disabled in Room Event
<a href="#">onRoomDismissed</a>	Room Dismissed Event

<a href="#">onKickedOutOfRoom</a>	Kicked Out of Room Event
<a href="#">onRoomSpeechModeChanged</a>	Room Microphone Control Mode Change

### Room User Event Callback

API	Description
<a href="#">onRemoteUserEnterRoom</a>	Remote User Entering Room Event
<a href="#">onRemoteUserLeaveRoom</a>	Remote User Leaving Room Event
<a href="#">onUserRoleChanged</a>	User Role Change Event
<a href="#">onUserVideoStateChanged</a>	User Video State Change Event
<a href="#">onUserAudioStateChanged</a>	User Audio State Change Event
<a href="#">onUserVoiceVolumeChanged</a>	User Volume Change Event
<a href="#">onSendMessageForUserDisableChanged</a>	User Text Message Sending Ability Change Event
<a href="#">onUserNetworkQualityChanged</a>	User Network Status Change Event
<a href="#">onUserScreenCaptureStopped</a>	Screen Sharing End Event

### Room Microphone Seat Event Callback

API	Description
<a href="#">onRoomMaxSeatCountChanged</a>	Room Maximum Microphone Seat Number Change Event (Only effective in conference type rooms)
<a href="#">onSeatListChanged</a>	Microphone Seat List Change Event
<a href="#">onKickedOffSeat</a>	Received User Kicked Off Microphone Event

### Request Signaling Event Callback

API	Description
<a href="#">onRequestReceived</a>	Received Request Message Event
<a href="#">onRequestCancelled</a>	Received Request Cancellation Event

### Room Message Event Callback

API	Description
<a href="#">onReceiveTextMessage</a>	Received Normal Text Message Event
<a href="#">onReceiveCustomMessage</a>	Received Custom Message Event

# TUIRoomKit

Last updated : 2024-03-29 17:53:52

Module: TUIRoomKit

Function: Multi-person audio and video main function interface

Version: 2.1.0

## TUIRoomKit

## TUIRoomKit

function list	describe
<a href="#">createInstance</a>	Create a TUIRoomKit instance (singleton mode).
<a href="#">destroyInstance</a>	Destroy the TUIRoomKit instance.
<a href="#">setSelfInfo</a>	Set up personal information, including username and avatar.
<a href="#">createRoom</a>	Create a room.
<a href="#">enterRoom</a>	Enter the room.

## createInstance

Initialize the TUIRoomKit singleton object.



```
public static TUIRoomKit createInstance();
```

### **destroyInstance**

Destroy the TUIRoomKit instance.

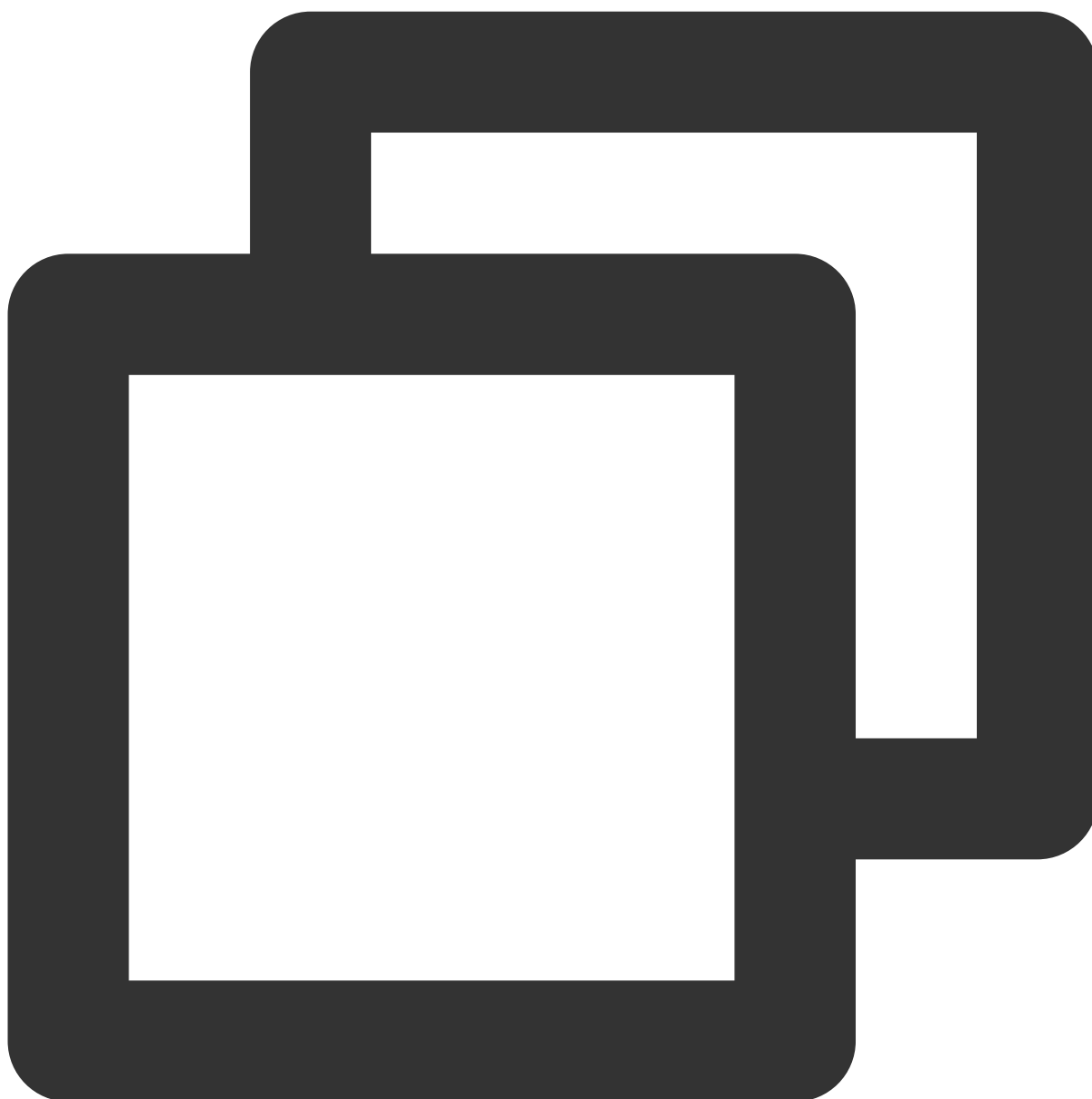




```
public static void destroyInstance();
```

## setSelfInfo

Set up personal information, including username and avatar.

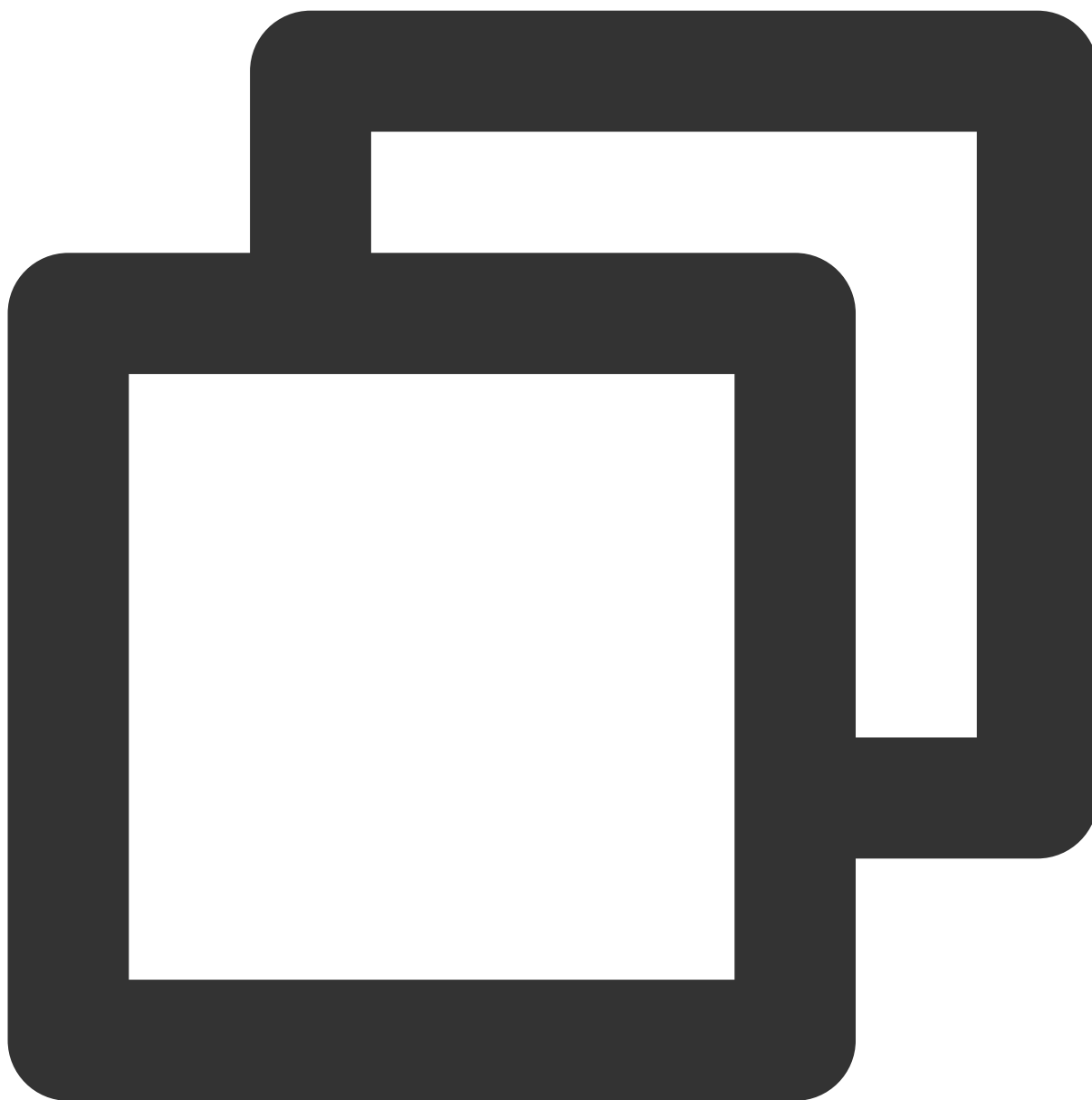


```
public abstract void setSelfInfo(String userName, String avatarURL,  
                                TUIRoomDefine.ActionCallback callback);
```

parameter	describe
userName	The individual's username.
avatarURL	Personal avatar link.
callback	Callback for success in setting personal information.

## createRoom

Create a room.



```
public abstract void createRoom(TUIRoomDefine.RoomInfo roomInfo, TUIRoomDefine.Acti
```

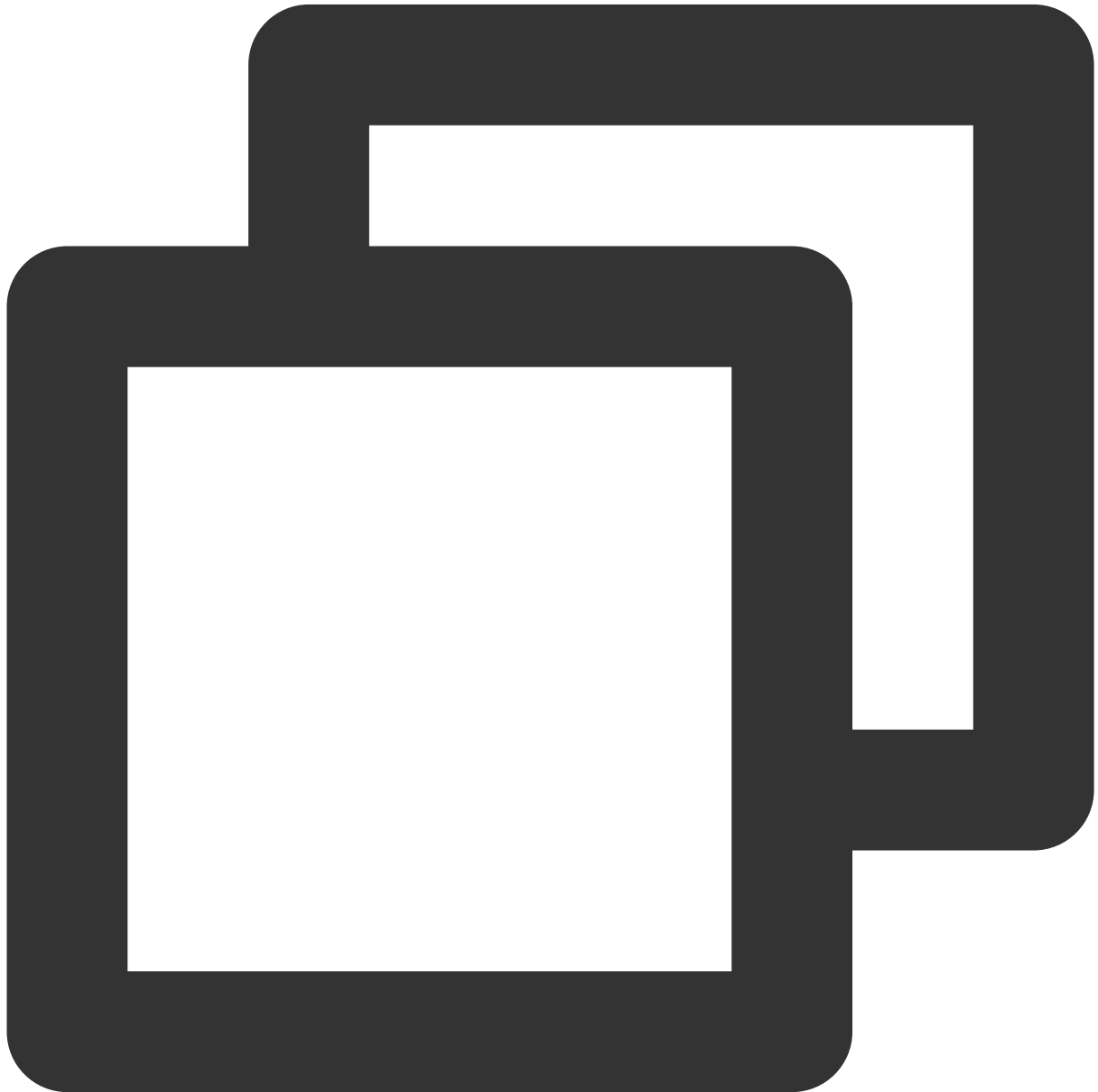
parameter	describe
roomInfo	Parameters for creating a room, including room number, room name, etc., where roomId is required and the rest can be default values.

callback

Callback to determine whether the room is successfully created.

## enterRoom

Enter the room.



```
public abstract void enterRoom(String roomId,  
                                boolean enableAudio,  
                                boolean enableVideo,
```

```
boolean isSoundOnSpeaker,  
TUIRoomDefine.GetRoomInfoCallback callback);
```

parameter	describe
roomId	Room number to enter the room.
enableAudio	true: When entering the room, turn on the microphone and push local audio data to the remote end. Other members can hear the local sound normally; false: When entering the room, only the microphone is turned on and local audio data is not pushed to the remote end. Other members cannot hear the local sound.
enableVideo	true: Enter the room, turn on the camera, and push the local video data to the remote end. Other members can see the local picture normally; false: When entering the room, the camera will not be turned on and local video data will not be pushed to the remote end. Other members will not be able to see the local video.
isSoundOnSpeaker	Whether to use the speaker to play sound, true to use the speaker, false to use the earpiece.
callback	Callback whether the room entry is successful.

# TUIRoomEngine

Last updated : 2023-10-23 11:32:46

## TUIRoomEngine (No UI Interface)

TUIRoomEngine API is a No UI Interface for Conference Component, you can use this API to custom encapsulate according to your business needs.

### **createInstance**

Create TUIRoomEngine Instance



```
static TUIRoomEngine createInstance()
```

**return:**TUIRoomEngine Instance

### **destroyInstance**

Destroy TUIRoomEngine Instance



```
void destroyInstance()
```

## login

Login interface, you need to initialize user information before entering the room and perform a series of operations

**Note :** In v1.0.0, this interface is named `setup`, and in v1.0.1 and above, please use `TUIRoomEngine.login` to log in to `TUIRoomEngine`.





```
void static login(Context context,  
    int sdkAppId,  
    String userId,  
    String userSig,  
    TUIRoomDefine.ActionCallback callback)
```

**Parameters:**

Parameter	Type	Meaning
context	Context	Android Context

sdkAppld	int	Get sdkAppld information from Application Info
userId	String	User ID
userSig	String	UserSig
callback	TUIRoomDefine.ActionCallback	API Callback for notifying the success or failure of the interface call

## logout

Logout interface, there will be actively leave the room operation, destroy resources



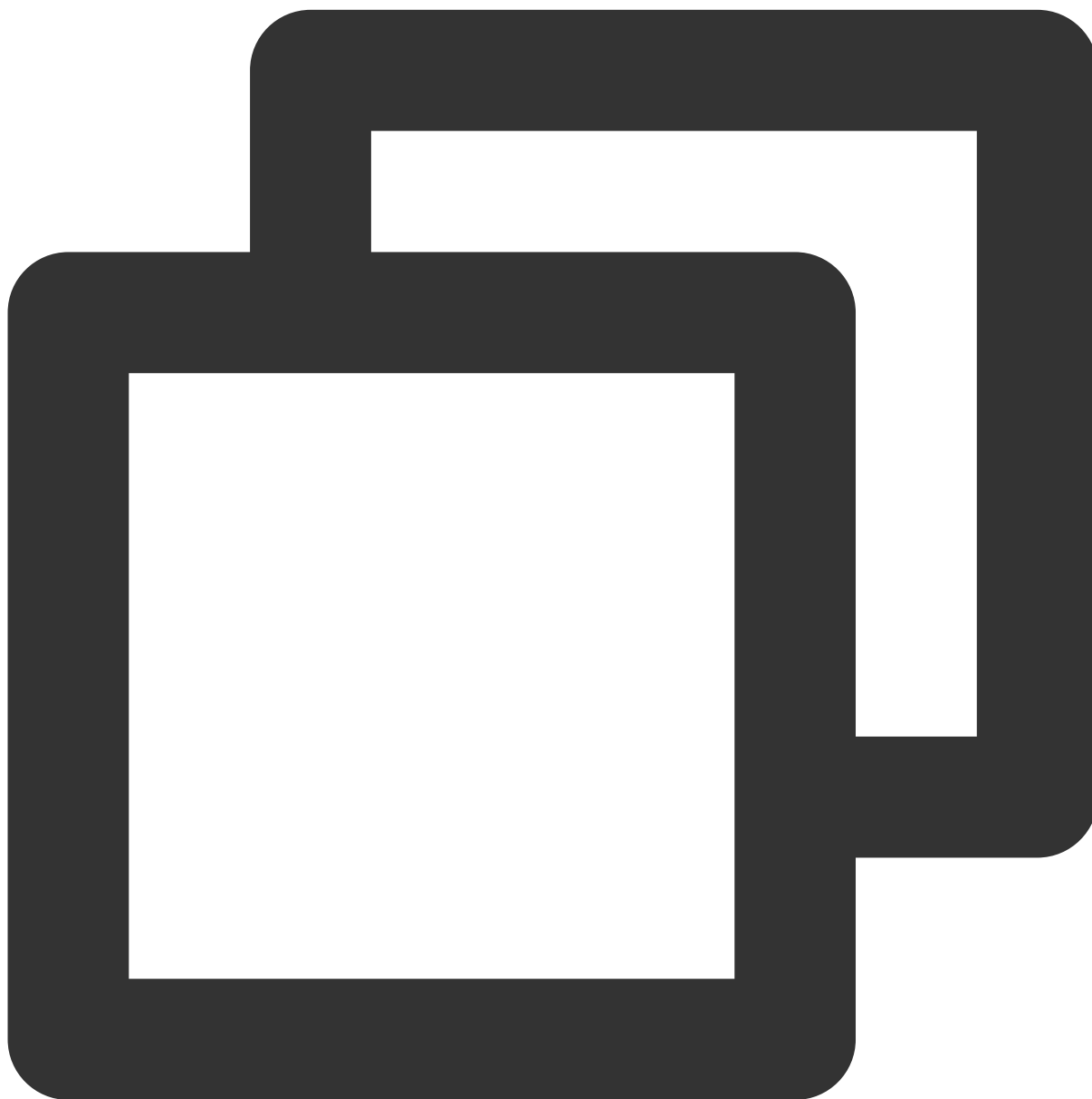
```
static void logout(TUIRoomDefine.ActionCallback callback)
```

**Parameters:**

Parameter	Type	Meaning
callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call

**setSelfInfo**

Set local user name and avatar



```
static void setSelfInfo(String userName, String avatarURL, TUIRoomObserver.ActionCa
```

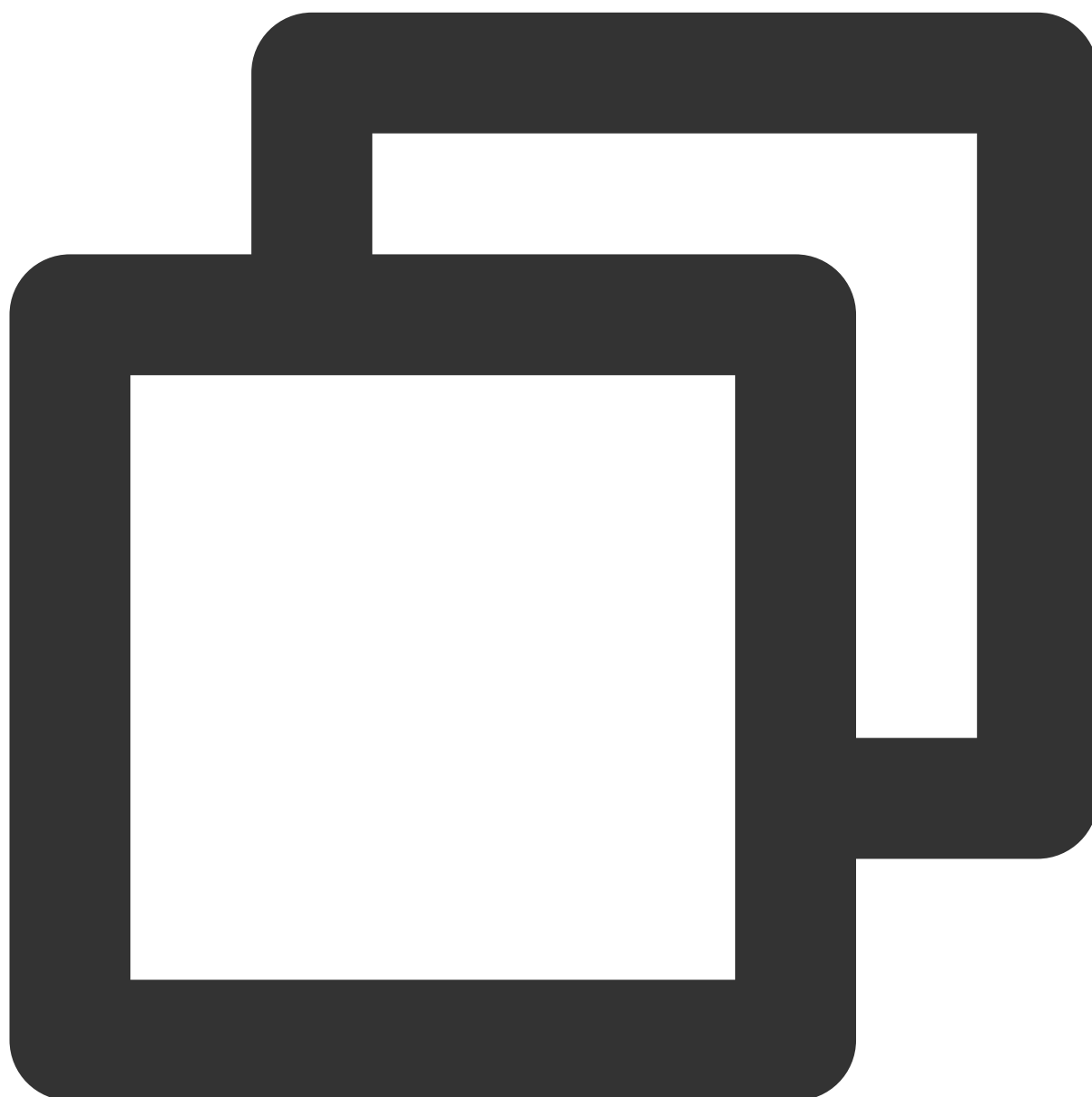
**Parameters:**

Parameter	Type	Meaning
userName	int	User Name
avatarUrl	String	User avatar URL address

callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call
----------	------------------------------	--

## getSelfInfo

Get the basic information of the local user login

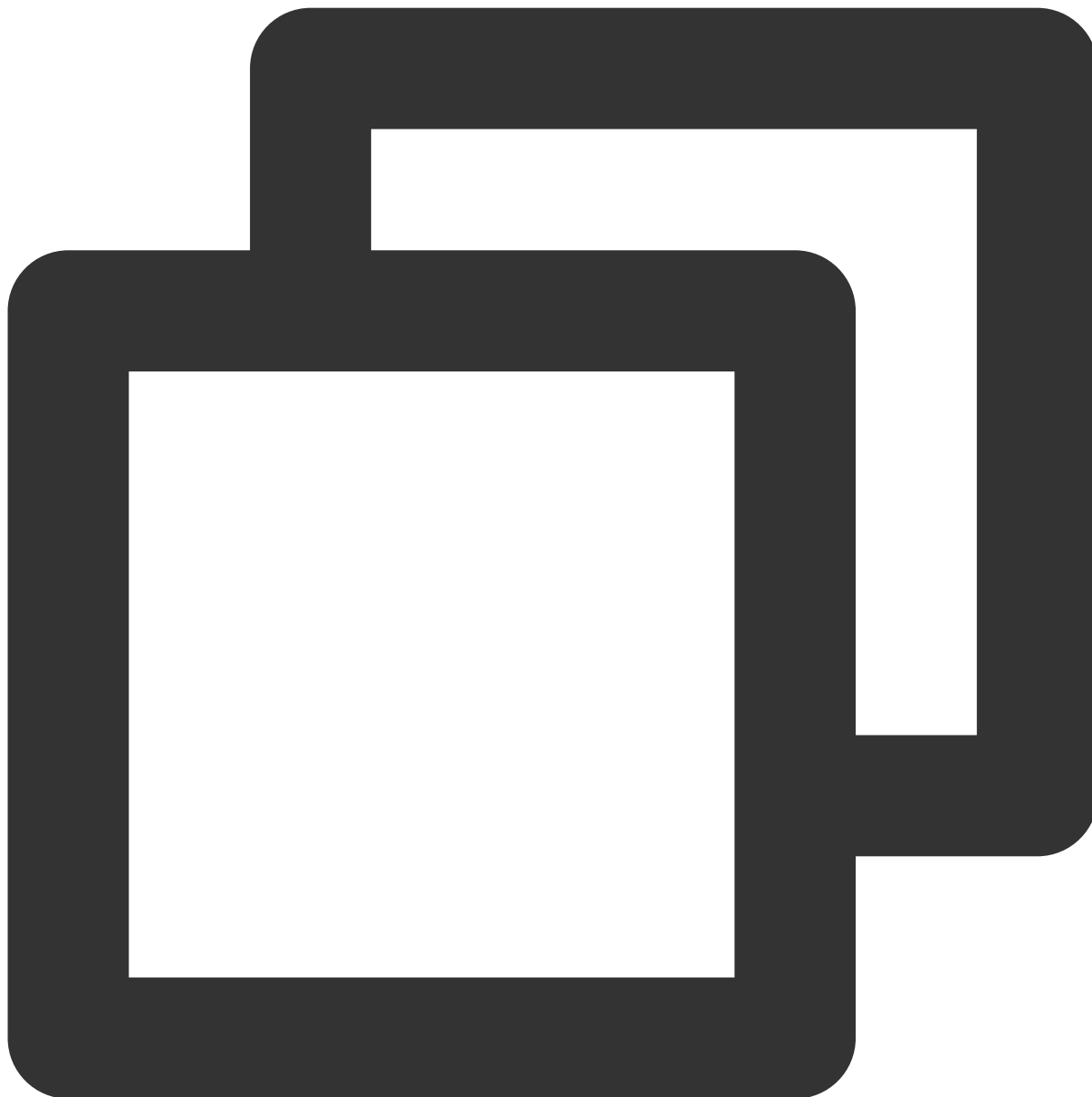


```
static TUIRoomDefine.LoginUserInfo getSelfInfo()
```

**return:**the basic information of the local user login

## addObserver

Add TUIRoomEngine Event Callback



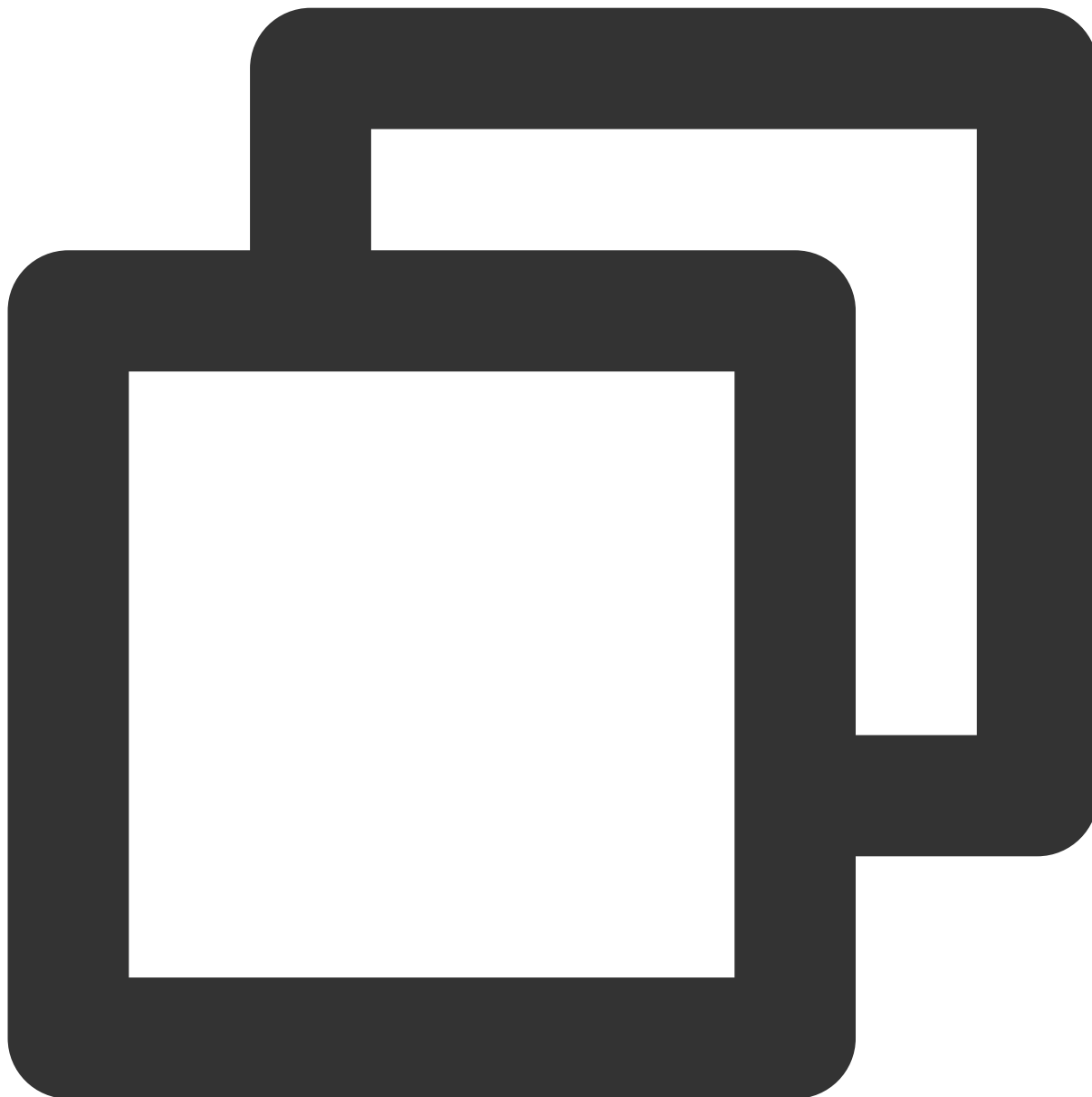
```
void addObserver(TUIRoomObserver observer)
```

### Parameters:

Parameter	Type	Meaning
observer	TUIRoomObserver	TUIRoomEngine Event Callback

## removeObserver

Remove TUIRoomEngine Event Callback

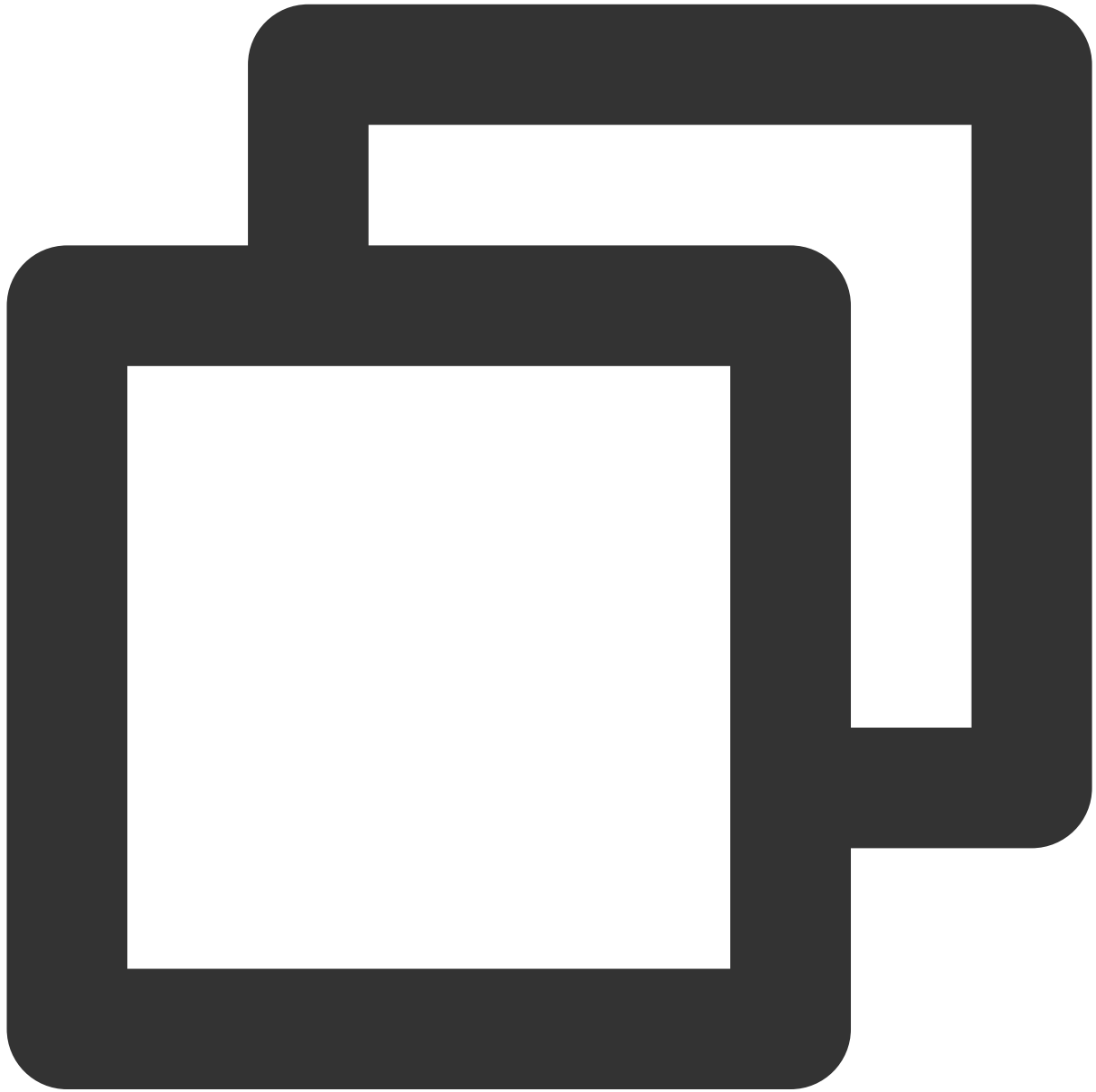


```
void removeObserver(TUIRoomObserver observer)
```

Parameter	Type	Meaning
observer	TUIRoomObserver	TUIRoomEngine Event Callback

## createRoom

The host creates a room, and the user who calls `createRoom` is the owner of the room. When creating a room, you can set the Room ID, room name, and whether the room allows users to join, enable video and audio, send messages, and other functions.



```
void createRoom(TUIRoomDefine.RoomInfo roomInfo, TUIRoomDefine.ActionCallback callb
```

**Parameters:**

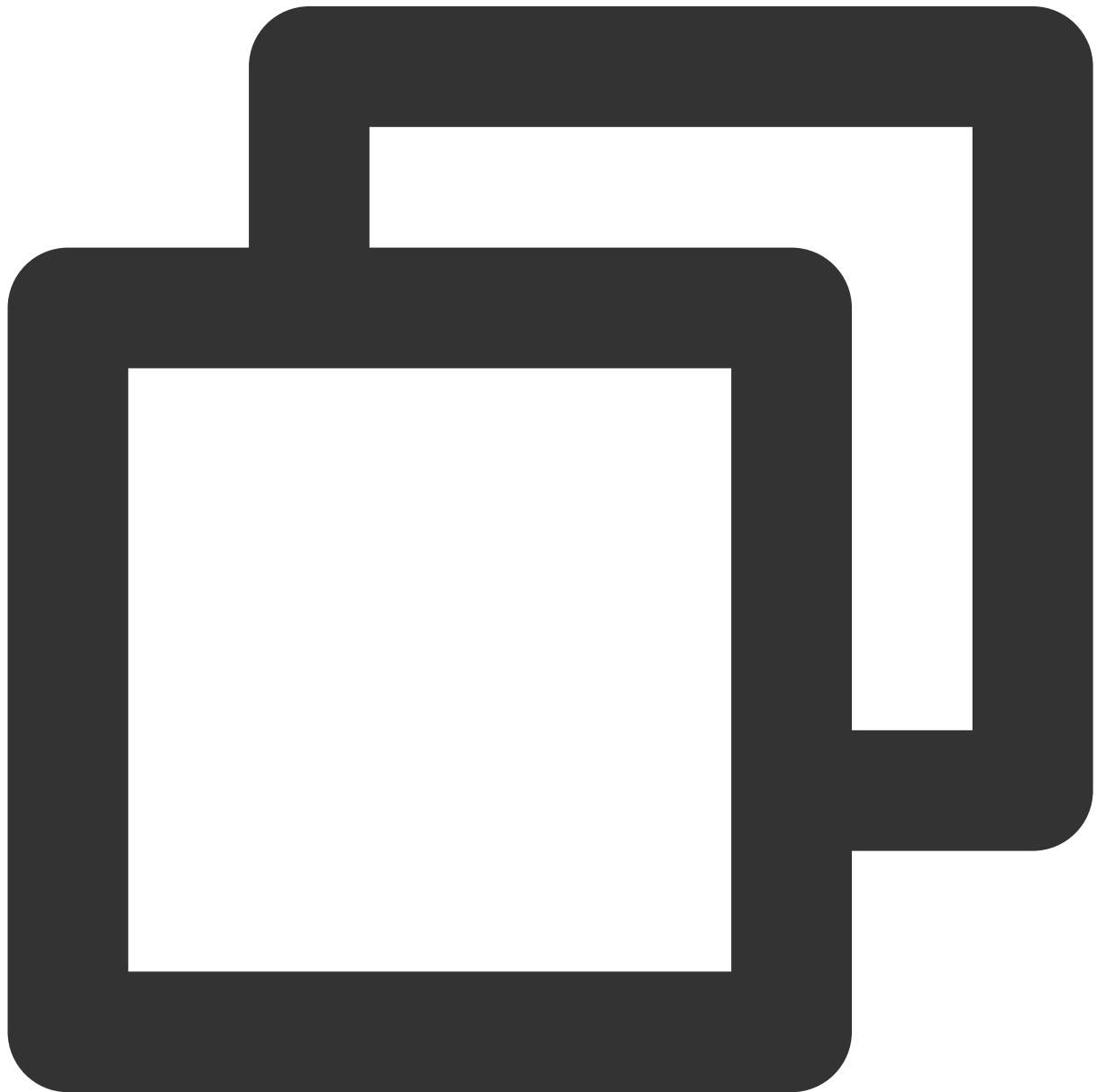
Parameter	Type	Meaning
roomInfo	<a href="#">TUIRoomDefine.RoomInfo</a>	Room data



callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call
----------	------------------------------	--

## destroyRoom

Destroy Room Interface, the room owner must initiate the destruction of the room. After the room is destroyed, it is unavailable for entry.



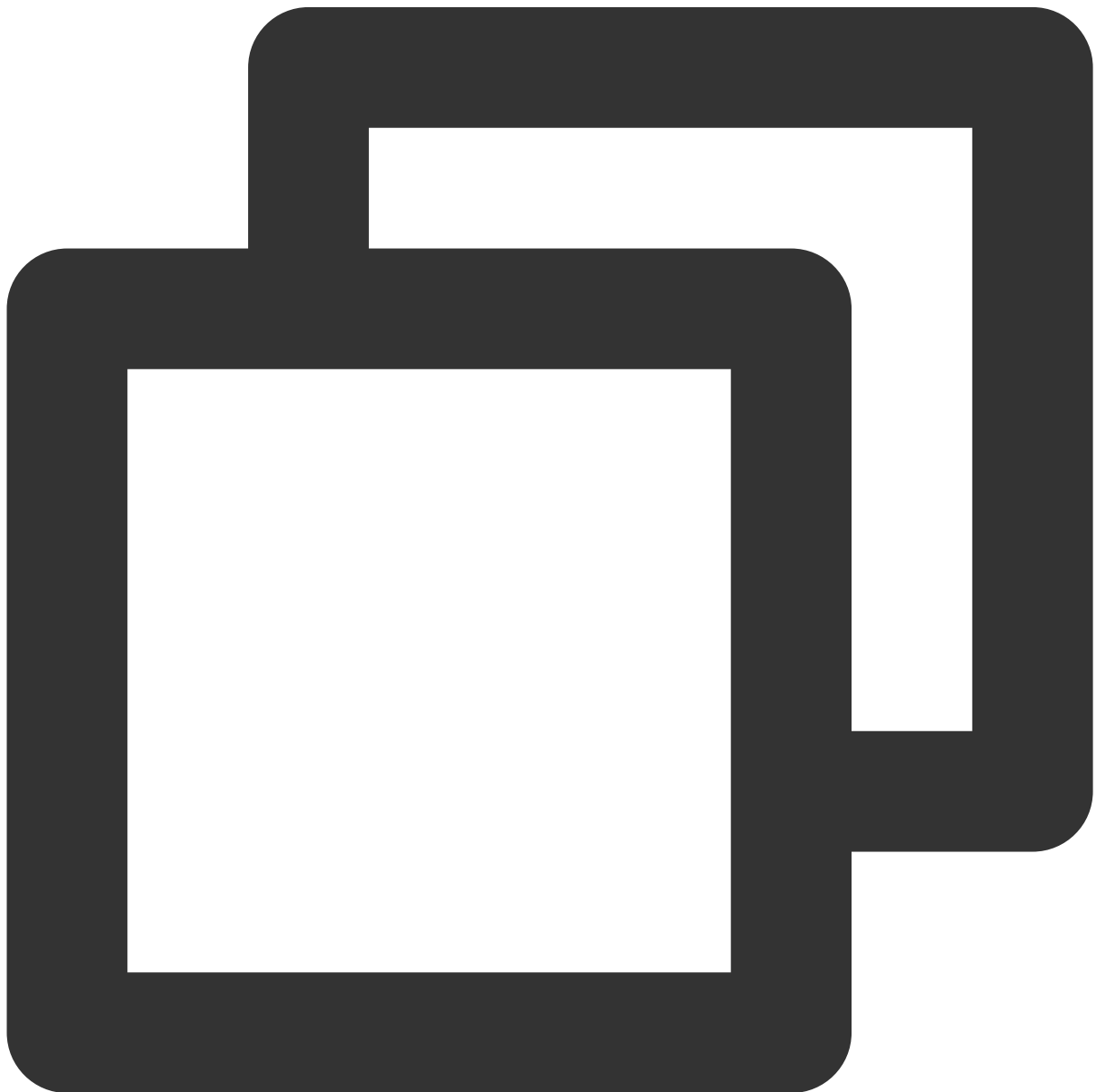
```
void destroyRoom(TUIRoomDefine.ActionCallback callback)
```

### Parameters:

Parameter	Type	Meaning
callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call

## enterRoom

Entered room.



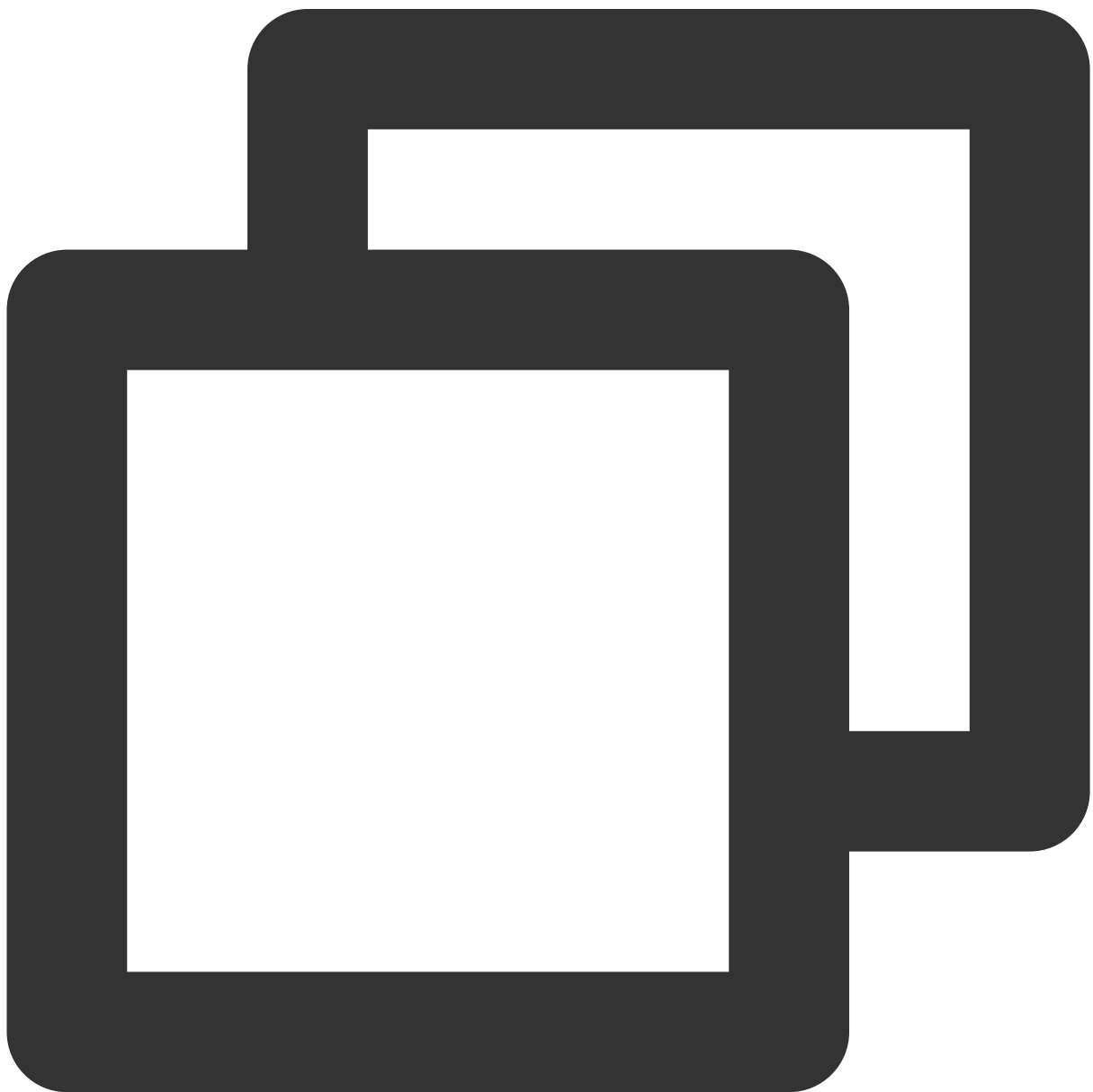
```
void enterRoom(String roomId, TUIRoomDefine.GetRoomInfoCallback callback)
```

**Parameters:**

Parameter	Type	Meaning
roomId	String	Room ID
callback	TUIRoomDefine.GetRoomInfoCallback	Get the entered Room data Callback

**exitRoom**

Exit Room Interface, after the user executes Enter Room, they can leave the room through Leave Room.



```
void exitRoom(boolean isSyncWaiting, TUIRoomDefine.ActionCallback callback);
```

**Parameters:**

Parameter	Type	Meaning
isSyncWaiting	boolean	Whether to synchronize leaving the room
callback	TUIRoomDefine.ActionCallback	Leave Room Result Callback

**connectOtherRoom**

Connect to other rooms

**Description:**

Used for live streaming scenario to apply for cross-room streaming



```
TUIRoomDefine.Request connectOtherRoom(String roomId,  
                                         String userId,  
                                         int timeout,  
                                         TUIRoomDefine.RequestCallback callback);
```

**Parameters:**

Parameter	Type	Meaning
roomId	String	Room ID

userId	String	User ID
timeout	int	Time
callback	TUIRoomDefine.RequestCallback	Connect to other rooms Callback

**Return :** Request body

### **disconnectOtherRoom**

Disconnect from other rooms

**Description:**

Used for live streaming scenario to disconnect cross-room streaming



```
void disconnectOtherRoom(TUIRoomDefine.ActionCallback callback);
```

**Parameters:**

Parameter	Type	Meaning
callback	TUIRoomDefine.ActionCallback	Disconnect with other rooms Connection result Callback

## fetchRoomInfo

Get Room data



```
void fetchRoomInfo(TUIRoomDefine.GetRoomInfoCallback callback);
```

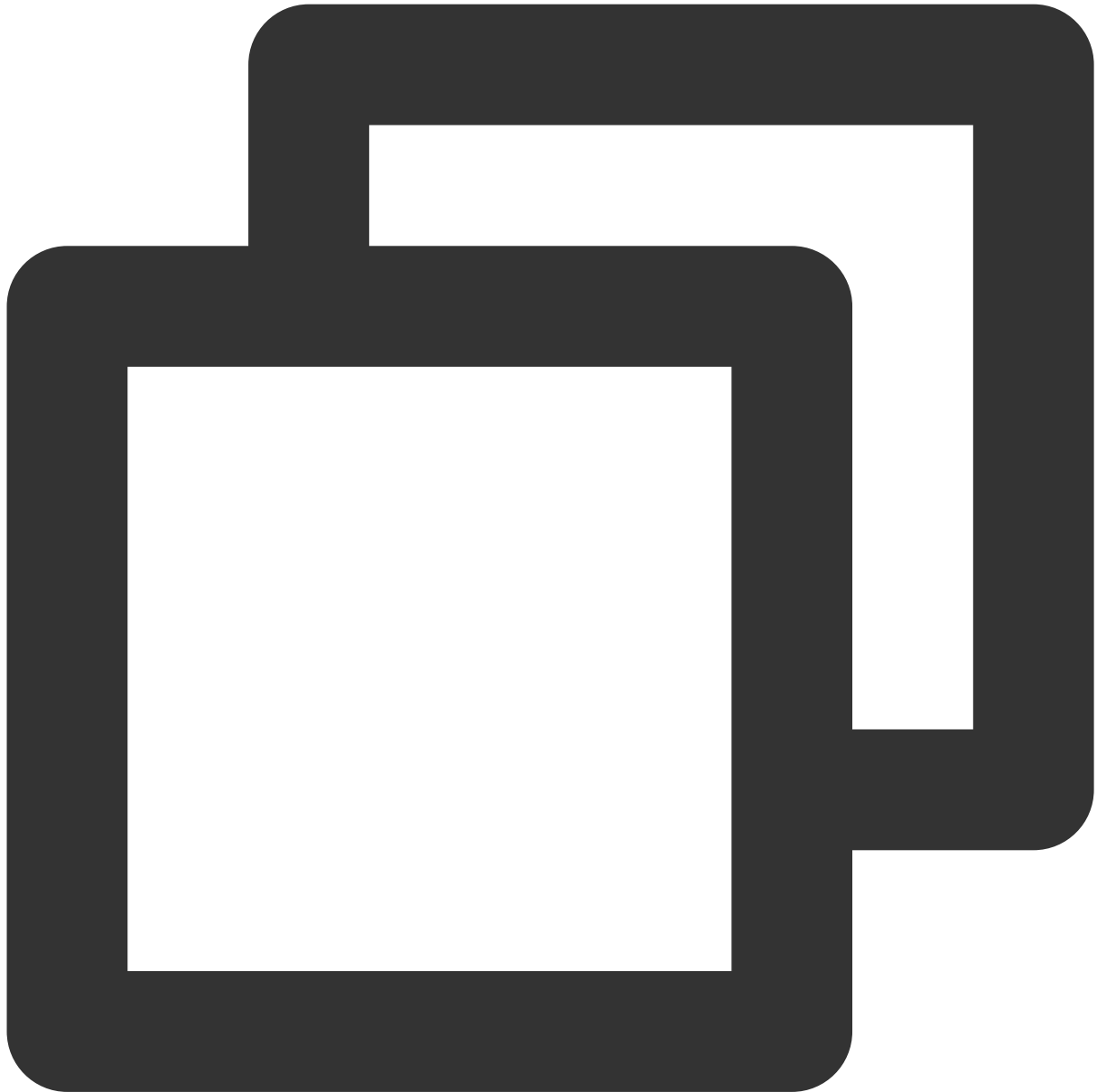
### Parameters:

Parameter	Type	Meaning
callback	TUIRoomDefine.GetRoomInfoCallback	Get Room data Callback



## updateRoomNameByAdmin

Update Room ID



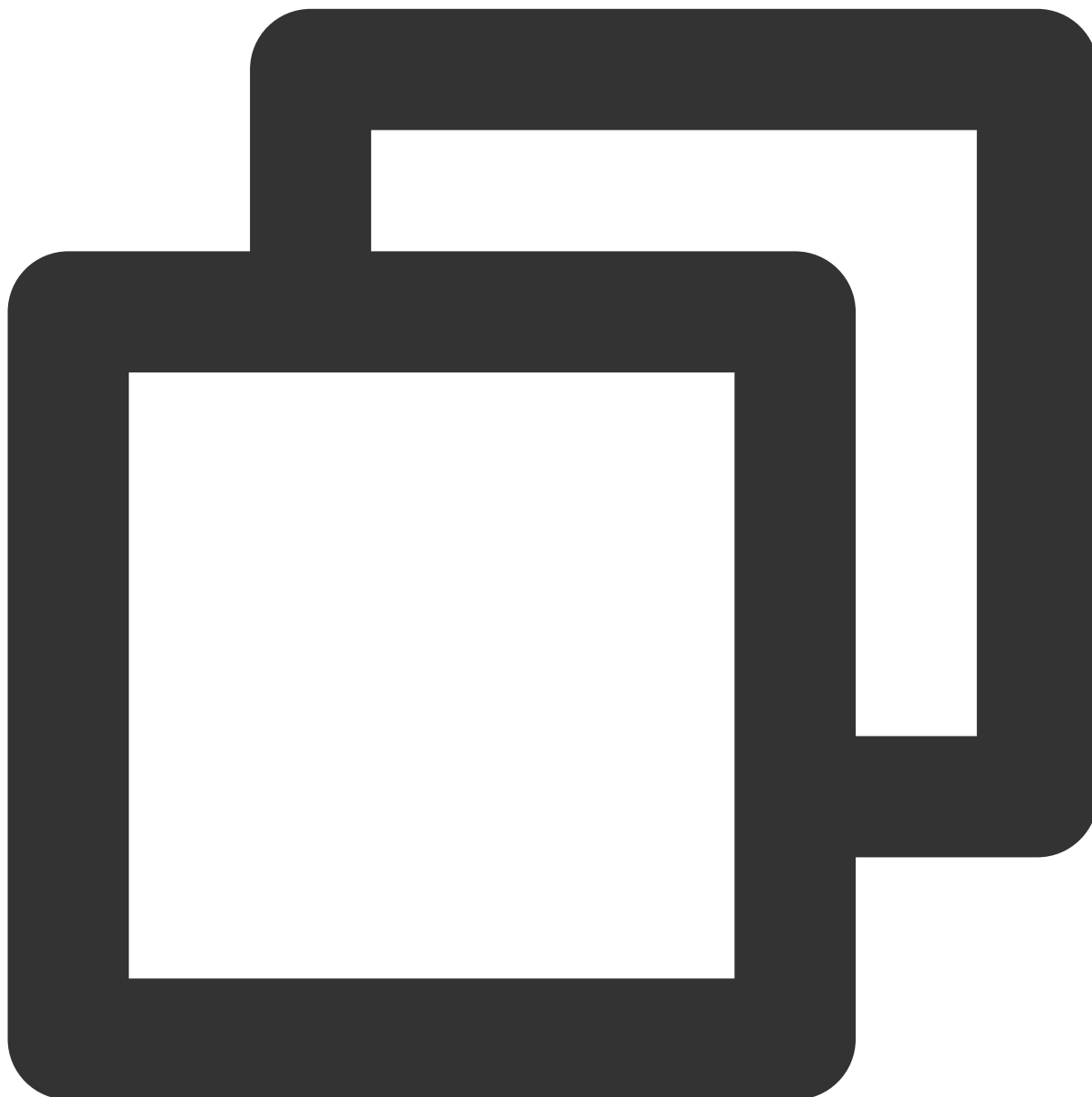
```
void updateRoomNameByAdmin(String roomName, TUIRoomDefine.ActionCallback callback);
```

### Parameters:

Parameter	Type	Meaning
roomName	String	Room ID
callback	TUIRoomDefine.ActionCallback	Update Operation result Callback

## updateRoomSpeechModeByAdmin

Set Management mode (only Administrator or Group owner can call)



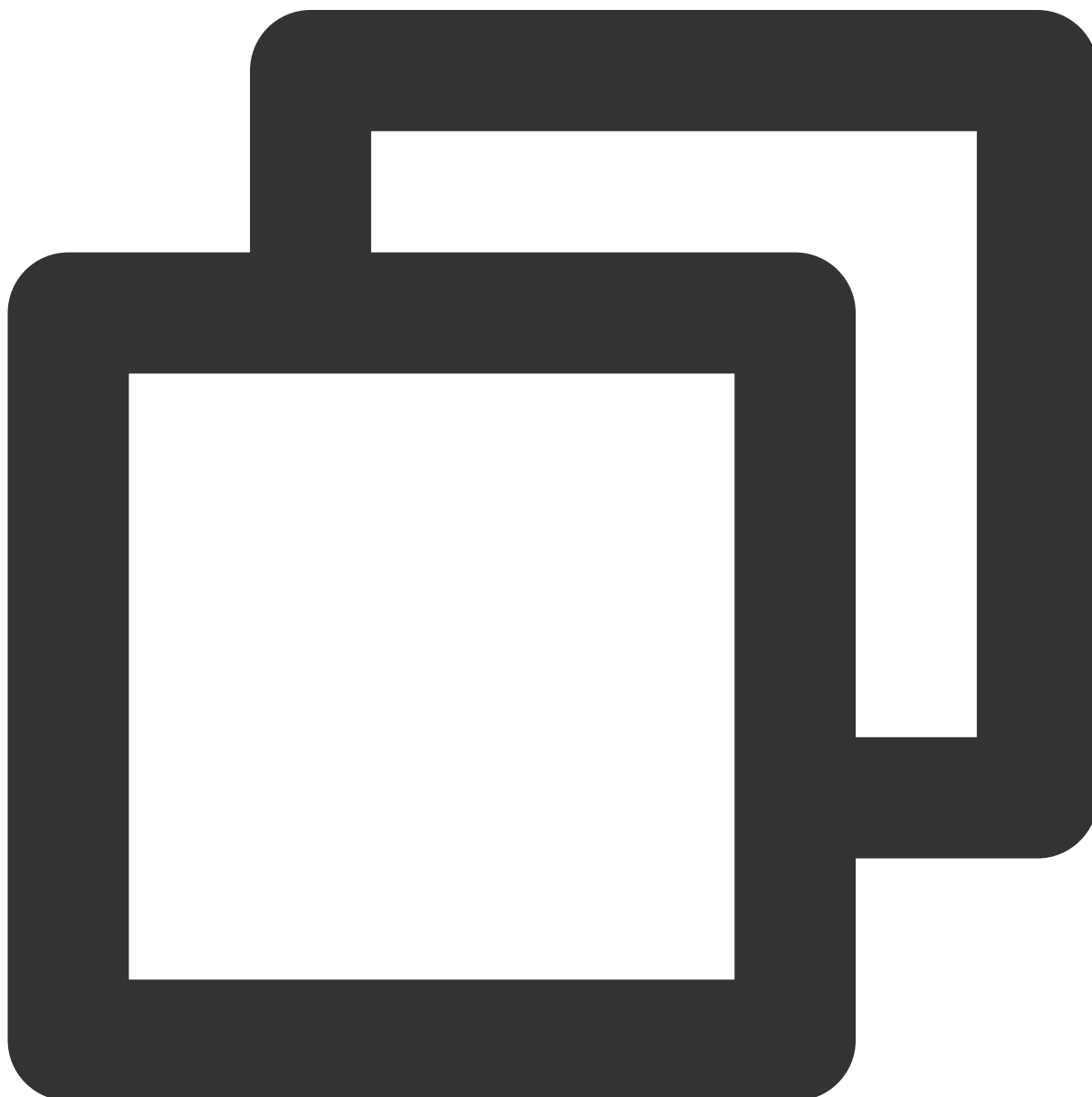
```
void updateRoomSpeechModeByAdmin(TUIRoomDefine.SpeechMode mode, TUIRoomDefine.Actio
```

Parameter	Type	Meaning
mode	<a href="#">TUIRoomDefine.SpeechMode</a>	Management mode

callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call
----------	------------------------------	--

## setLocalVideoView

Set Local user Video Rendering View control



```
void setLocalVideoView(TUIRoomDefine.VideoStreamType streamType,  
                        TUIVideoView view);
```

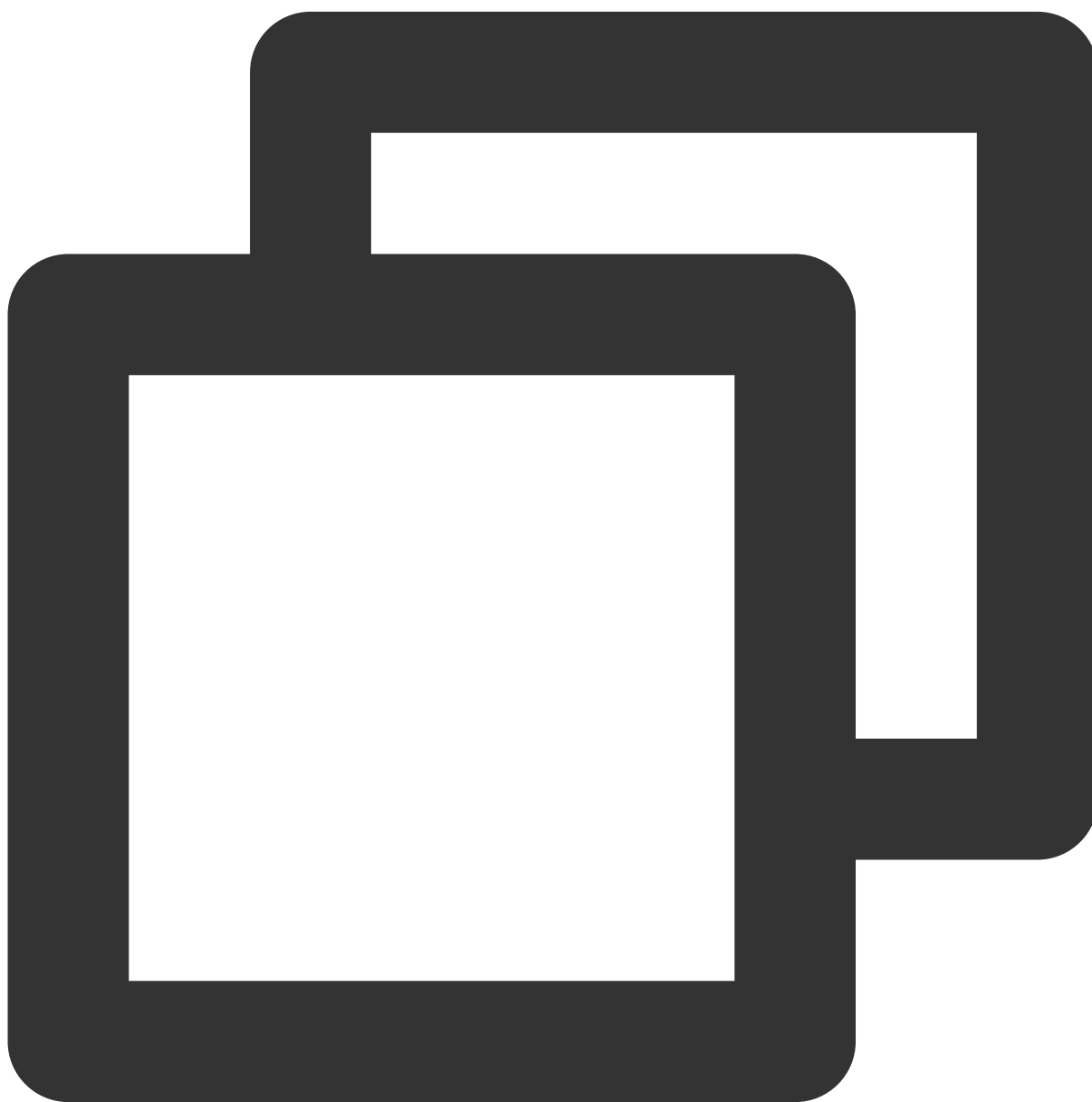
### Parameters:

--	--	--

Parameter	Type	Meaning
streamType	<a href="#">TUIRoomDefine.VideoStreamType</a>	Local streams type
view	TUIVideoView	To be rendered view, Video Rendering on this view

## openLocalCamera

Open Local Camera, Start Video Capturing



```
void openLocalCamera(boolean isFront,  
                    TUIRoomDefine.VideoQuality quality,  
                    TUIRoomDefine.ActionCallback callback);
```

**Parameters:**

Parameter	Type	Meaning
isFront	boolean	Whether to use Front Camera
quality	<a href="#">TUIRoomDefine.VideoQuality</a>	Video Quality
callback	TUIRoomDefine.ActionCallback	Open Camera Operation result Callback

**closeLocalCamera**

Close Local Camera



```
void closeLocalCamera();
```

## updateVideoQuality

Set Local Video Parameter



```
void updateVideoQuality(TUIRoomDefine.VideoQuality quality);
```

**Parameters:**

Parameter	Type	Meaning
quality	<a href="#">TUIRoomDefine.VideoQuality</a>	Video Quality

**startScreenSharing**

Start Screen sharing



```
void startScreenSharing();
```

## stopScreenSharing

Stop Screen sharing





```
void stopScreenSharing();
```

### **startPushLocalVideo**

Start pushing Local Video streams to Remote



```
void startPushLocalVideo();
```

## **stopPushLocalVideo**

Stop pushing Local Video streams to Remote



```
void stopPushLocalVideo();
```

## **openLocalMicrophone**

Open Local mic

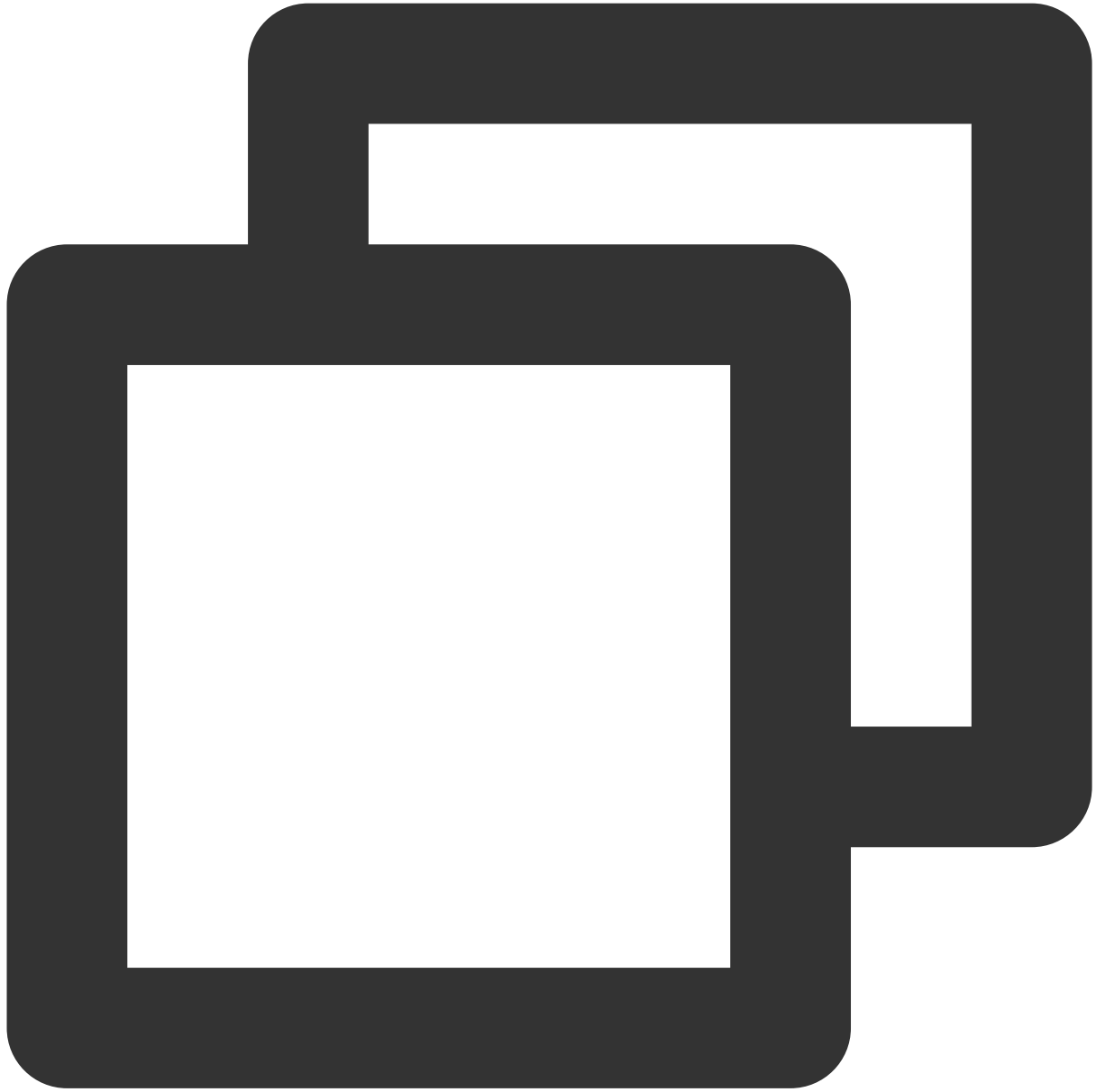


```
void openLocalMicrophone(TUIRoomDefine.AudioQuality quality, TUIRoomDefine.ActionCa
```

Parameter	Type	Meaning
quality	<a href="#">TUIRoomDefine.AudioQuality</a>	Audio Quality
callback	TUIRoomDefine.ActionCallback	Open mic Operation result Callback

## closeLocalMicrophone

Close Local mic



```
void closeLocalMicrophone();
```

### **updateAudioQuality**

Update Local Audio Codec Quality setting



```
void updateAudioQuality(TUIRoomDefine.AudioQuality quality);
```

**Parameters:**

Parameter	Type	Meaning
quality	<a href="#">TUIRoomDefine.AudioQuality</a>	Audio Quality

**startPushLocalAudio**

Start pushing Local Audio streams to Remote



```
void startPushLocalAudio();
```

## **stopPushLocalAudio**

Stop pushing Local Audio streams to Remote



```
void stopPushLocalAudio();
```

### **setRemoteVideoView**

Set Remote user Video Rendering View control





```
void setRemoteVideoView(String userId,
                        TUIRoomDefine.VideoStreamType streamType,
                        TUIVideoView view);
```

**Parameters:**

Parameter	Type	Meaning
userId	String	User ID
streamType	<a href="#">TUIRoomDefine.VideoStreamType</a>	User streams type

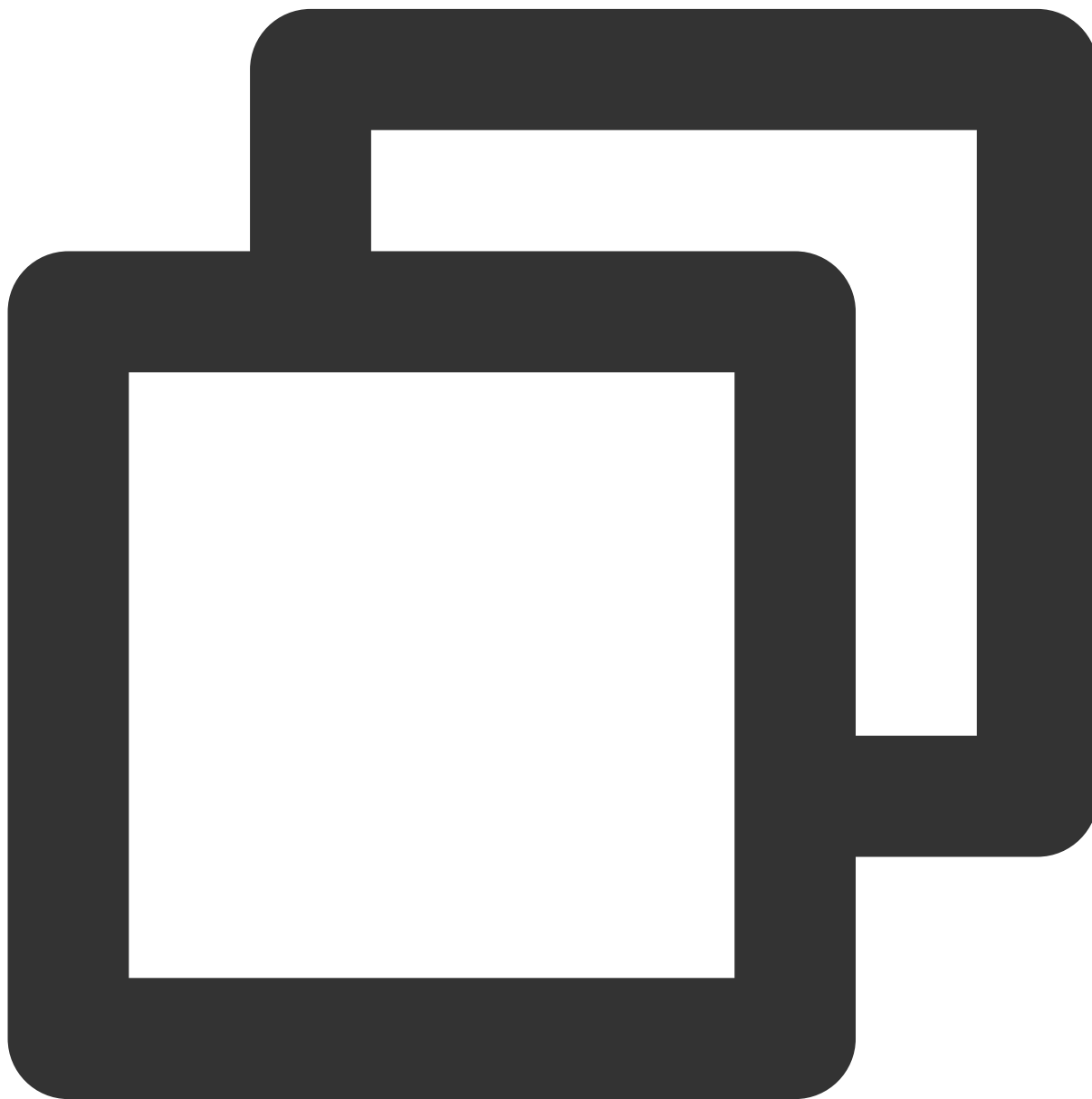
view

TUIVideoView

Playback Remote user streams view

## startPlayRemoteVideo

Start Playback Remote user Video streams



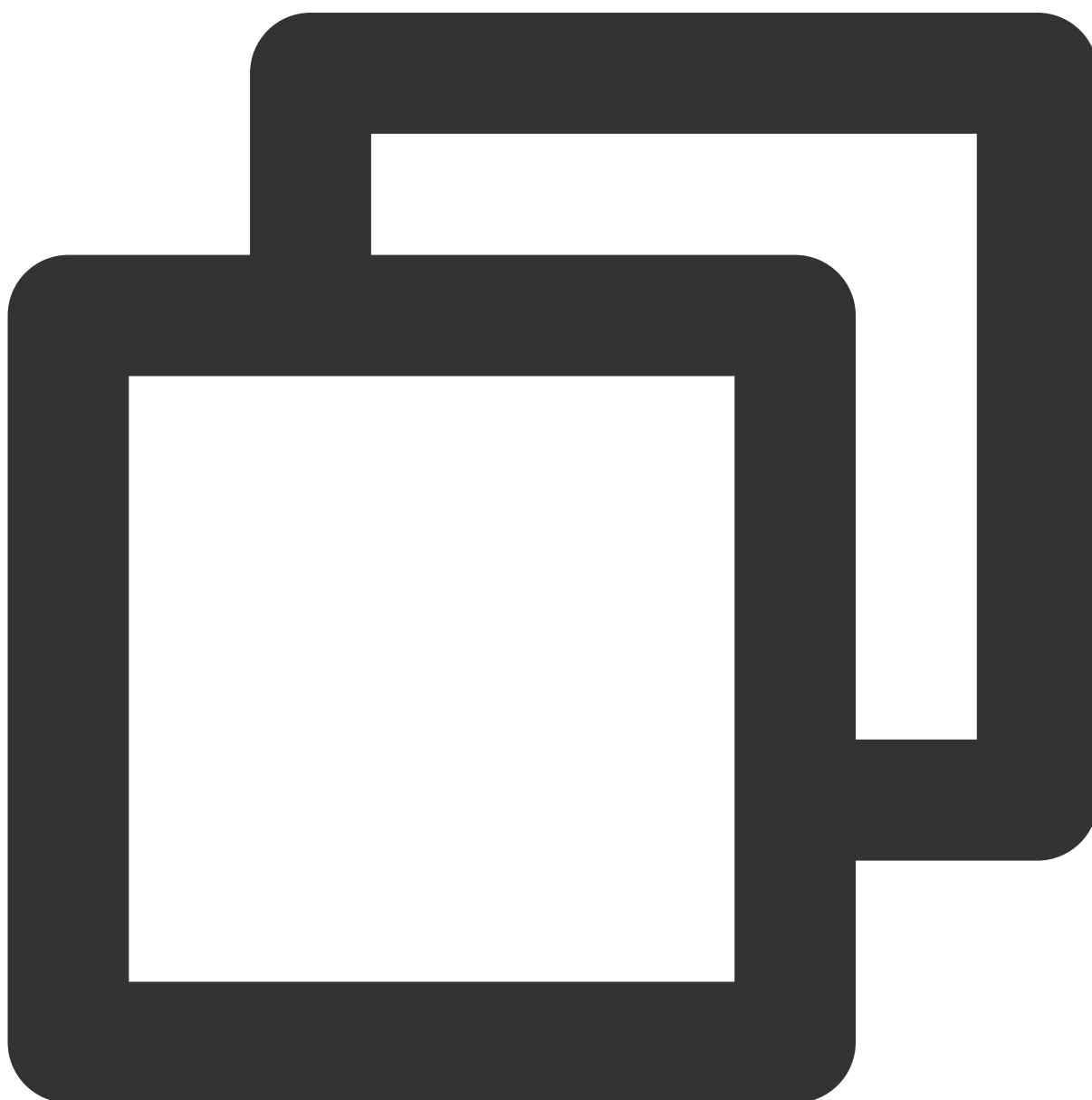
```
void startPlayRemoteVideo(String userId,
                          TUIRoomDefine.VideoStreamType streamType,
                          TUIRoomDefine.PlayCallback callback);
```

### Parameters:

Parameter	Type	Meaning
userId	String	User ID
streamType	<a href="#">TUIRoomDefine.VideoStreamType</a>	User streams type
callback	TUIRoomDefine.PlayCallback	Playback Operation result Callback

## stopPlayRemoteVideo

Stop Playback Remote user Video streams



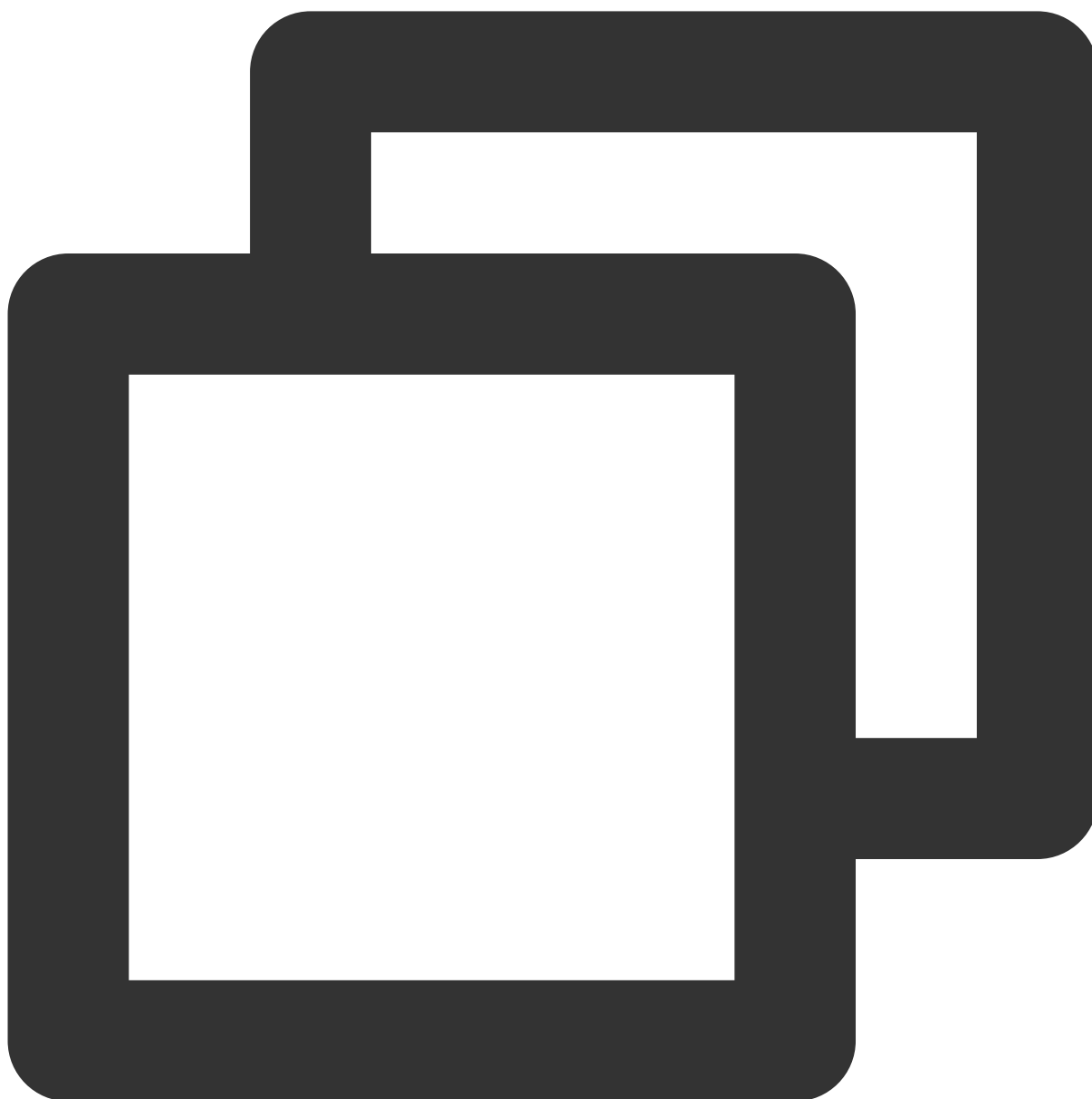
```
void stopPlayRemoteVideo(String userId, TUIRoomDefine.VideoStreamType streamType);
```

**Parameters:**

Parameter	Type	Meaning
userId	String	User ID
streamType	<a href="#">TUIRoomDefine.VideoStreamType</a>	User streams type

**muteRemoteAudioStream**

Mute Remote user



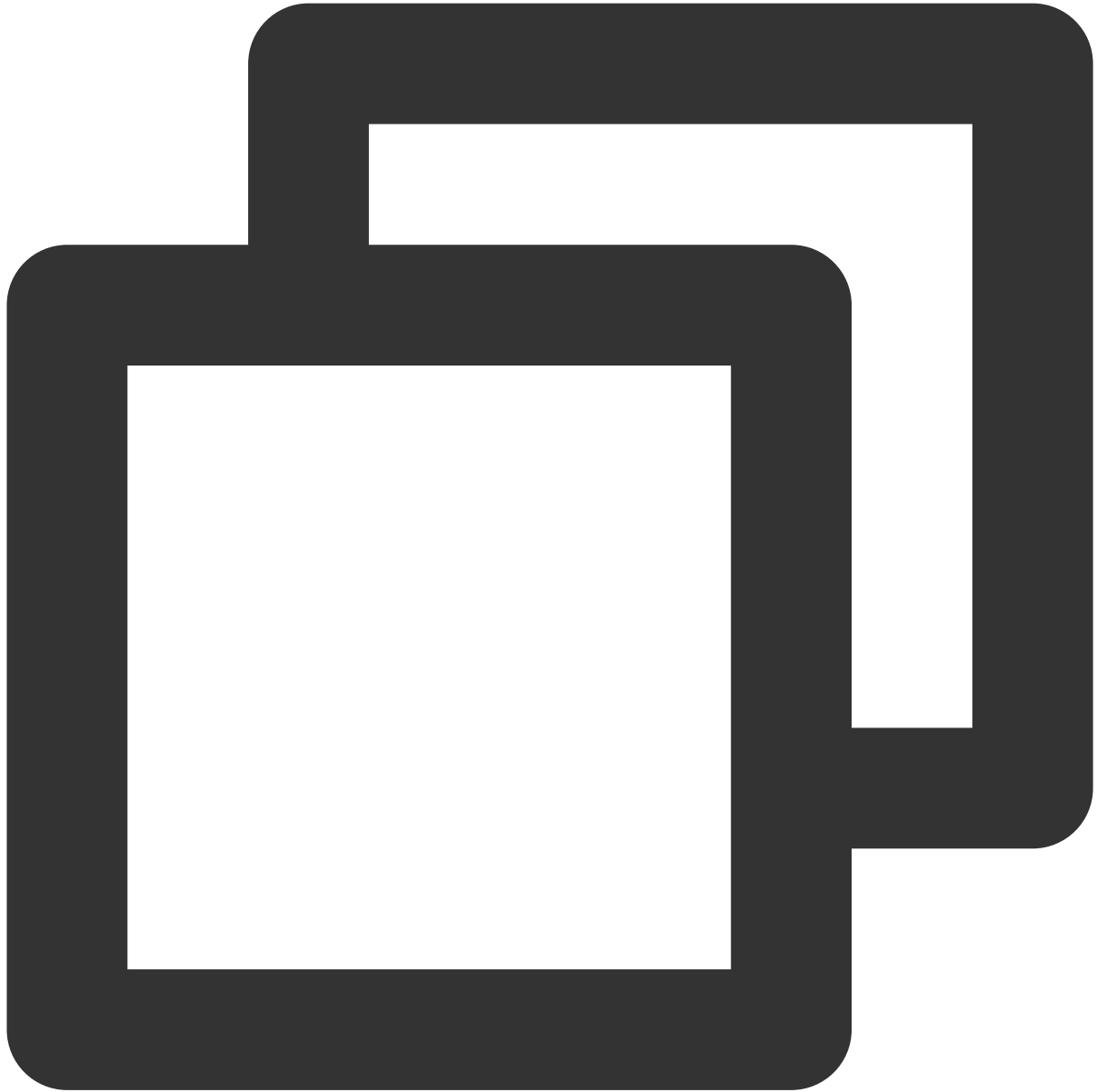
```
void muteRemoteAudioStream(String userId, boolean isMute);
```

**Parameters:**

Parameter	Type	Meaning
userId	String	User ID
isMute	boolean	Whether to mute

**getUserList**

Get current User list in the room, Note that the maximum number of User list fetched by this Interface is 100



```
void getUserList(long nextSequence, TUIRoomDefine.GetUserListCallback callback);
```

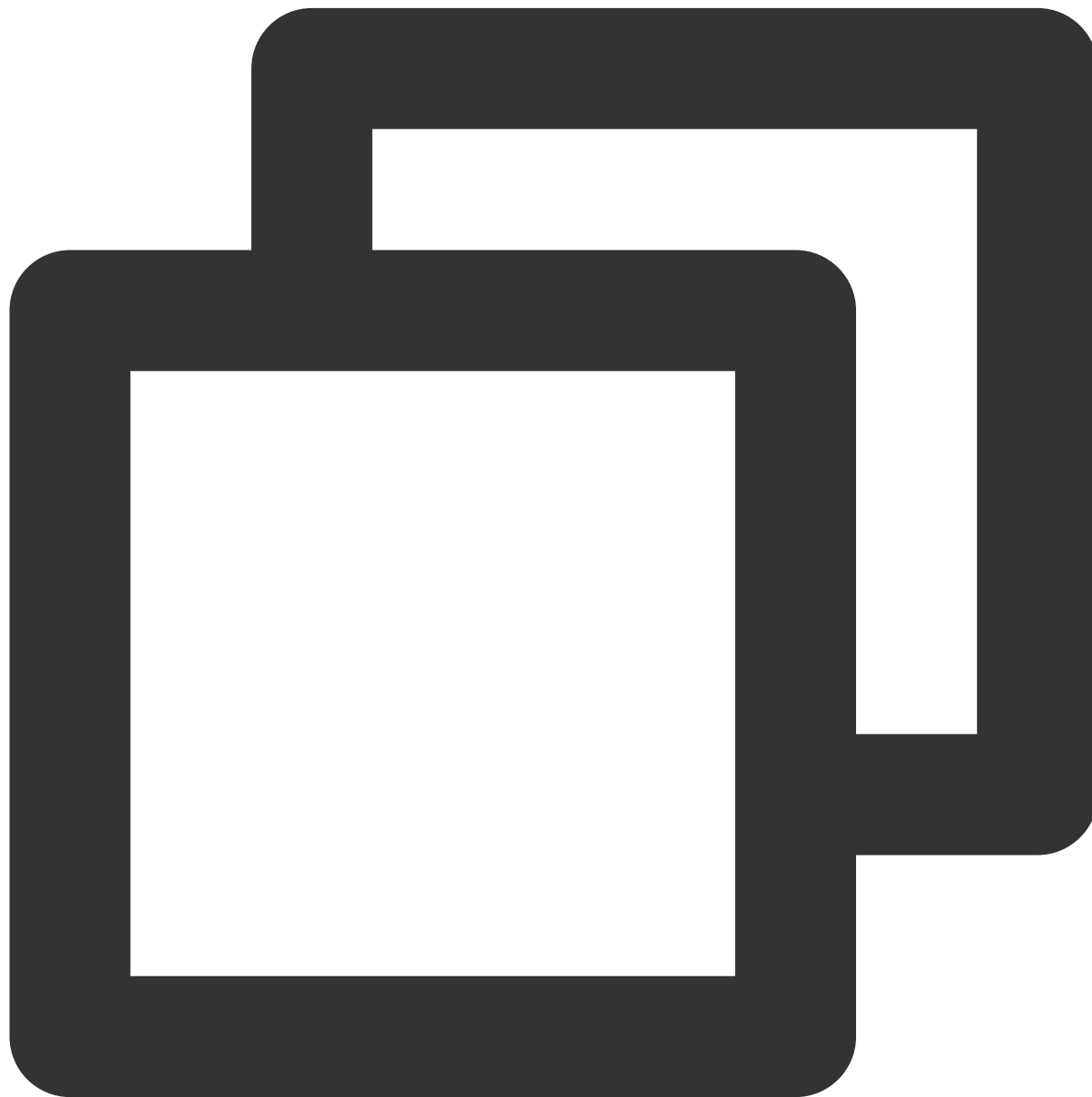
**Parameters:**

Parameter	Type	Meaning
nextSequence	long	Pagination Fetch Flag, fill in 0 for the first Fetch, if the nextSeq in the Callback is not 0, you need to do Pagination, pass in the nextSeq to Fetch again until the nextSeq in the Callback is 0

callback	TUIRoomDefine.GetUserListCallback	Get User list Callback
----------	-----------------------------------	------------------------

## getUserInfo

Get User [Learn more](#)



```
void getUserInfo(String userId, TUIRoomEngineDef.GetUserInfoCallback callback);
```

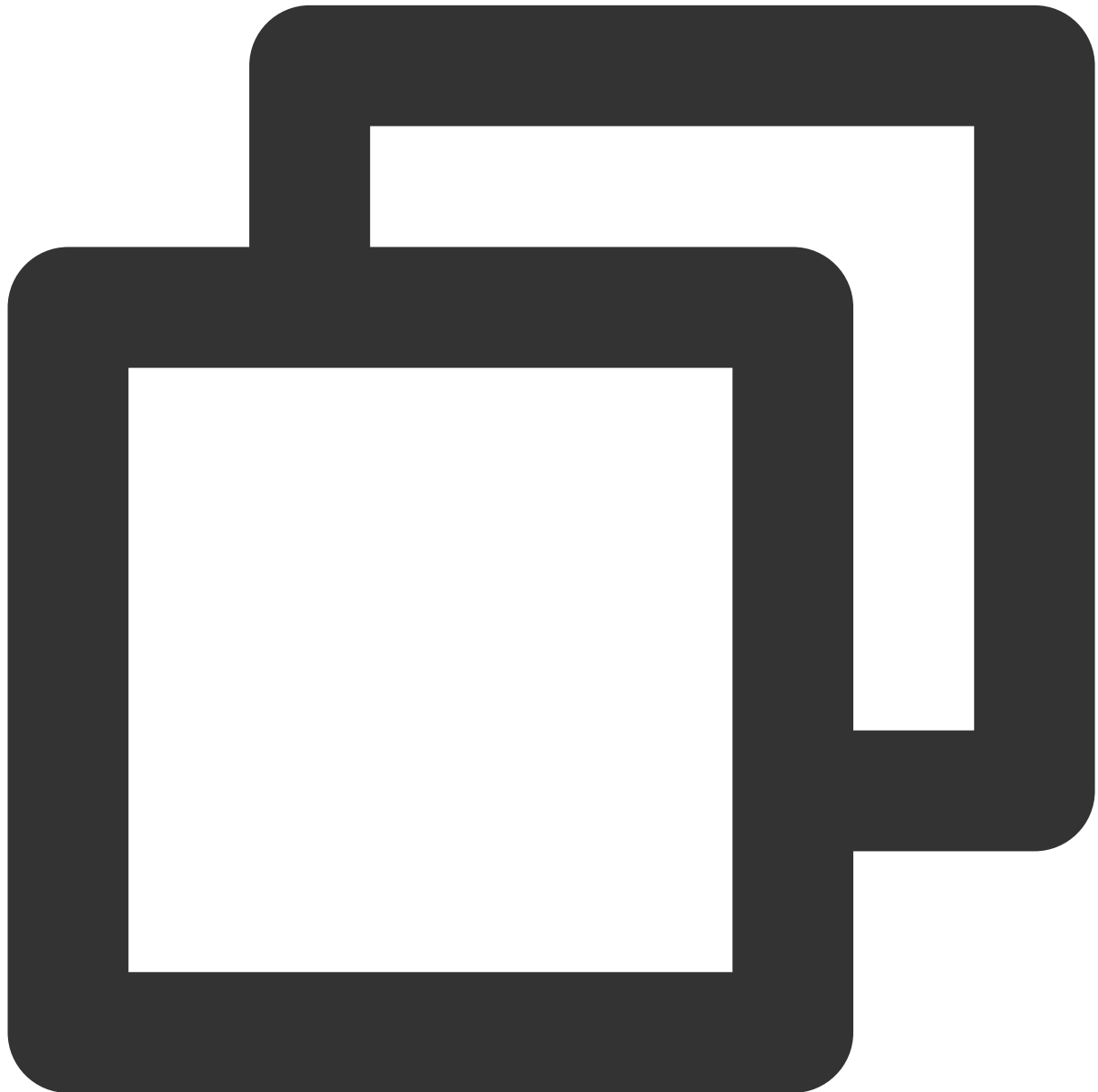
### Parameters:

Parameter	Type	Meaning
-----------	------	---------

userId	String	Get Learn more of the user by userId
callback	TUIRoomDefine.GetUserInfoCallback	Get User Learn more Callback

## changeUserRole

Change user Role, only Administrator or Group owner can call



```
void changeUserRole(String userId,  
                    TUIRoomDefine.Role role,  
                    TUIRoomDefine.ActionCallback callback);
```

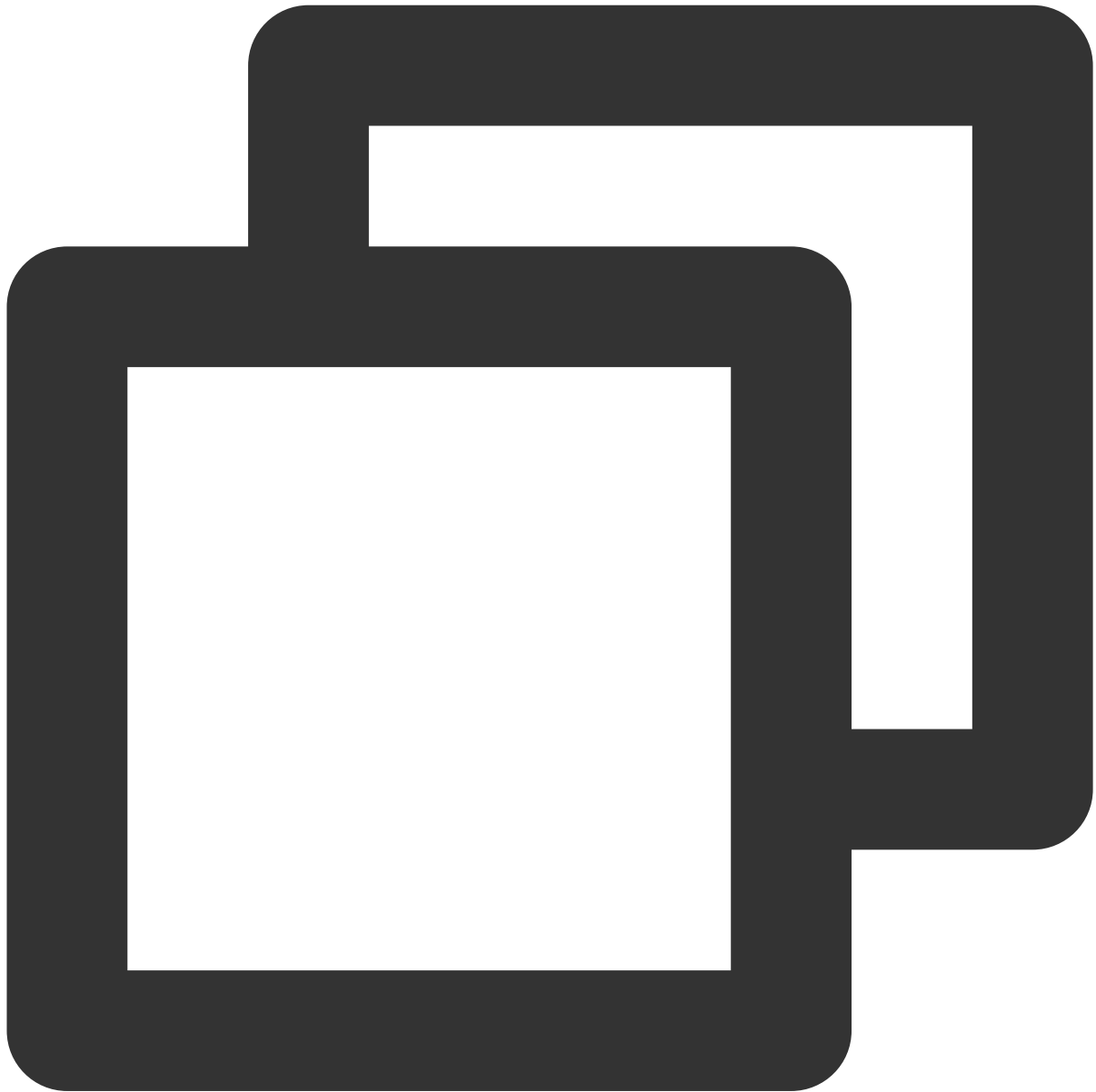


**Parameters:**

Parameter	Type	Meaning
userId	String	User ID
role	<a href="#">TUIRoomDefine.Role</a>	User Role
callback	TUIRoomDefine.ActionCallback	Change Role Operation result Callback

**kickRemoteUserOutOfRoom**

Kick user out of the room, only Administrator or Group owner can call



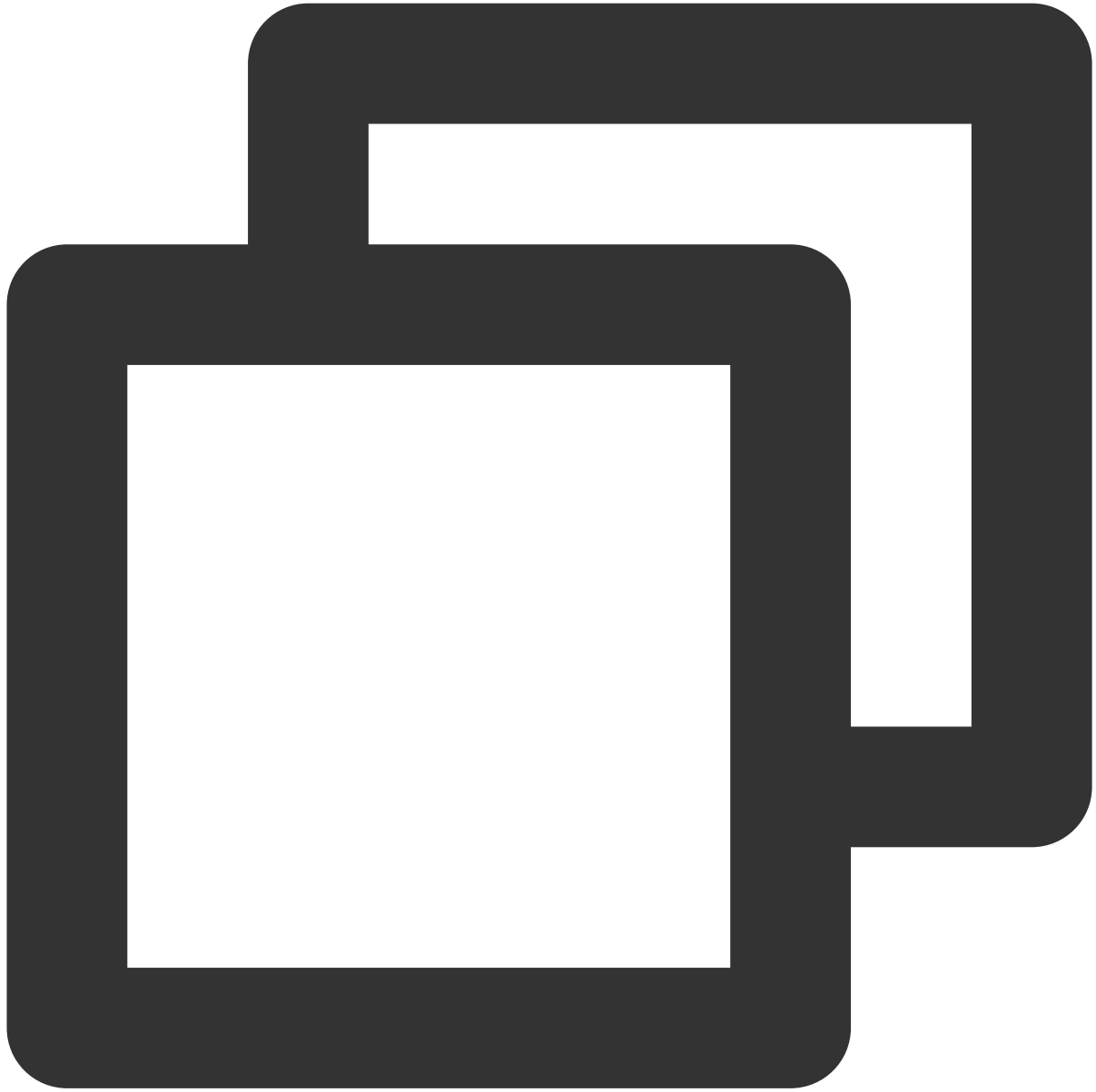
```
void kickRemoteUserOutOfRoom(String userId, TUIRoomDefine.ActionCallback callback);
```

**Parameters:**

Parameter	Type	Meaning
userId	String	User ID
callback	TUIRoomDefine.ActionCallback	Operation result Callback

**disableDeviceForAllUserByAdmin**

Manage Media device for all users, only Administrator or Group owner can call



```
void disableDeviceForAllUserByAdmin(TUIRoomDefine.MediaDevice device,  
                                     boolean isDisable,  
                                     TUIRoomDefine.ActionCallback callback);
```

Parameter	Type	Meaning
device	<a href="#">TUIRoomDefine.MediaDevice</a>	Device
isDisable	boolean	Whether to Disable

callback

TUIRoomDefine.ActionCallback

Operation result Callback

## openRemoteDeviceByAdmin

Request Remote user to open Media device, only Administrator or Group owner can call

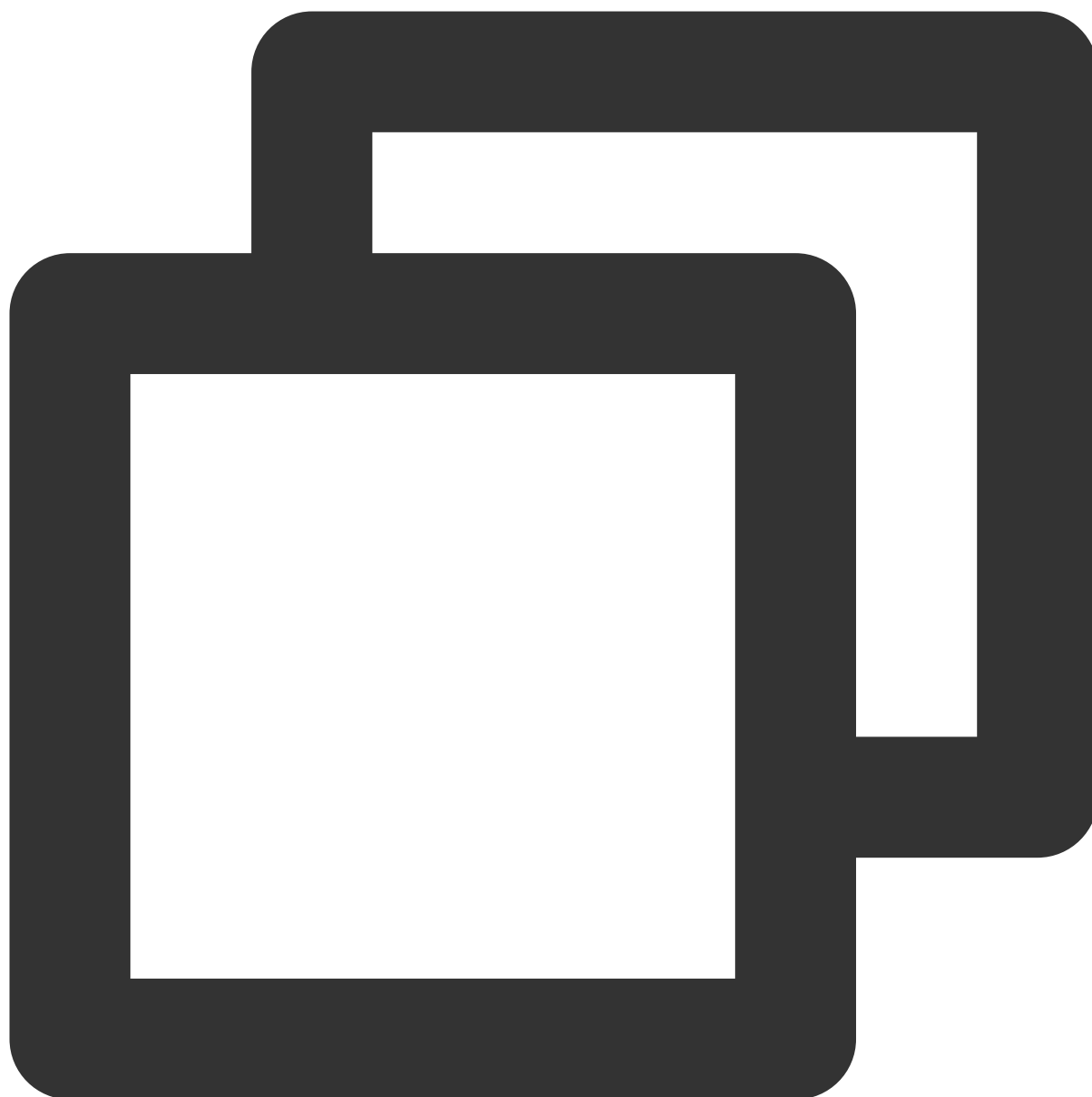


```
TUIRoomDefine.Request openRemoteDeviceByAdmin(String userId,  
                                                TUIRoomDefine.MediaDevice device,  
                                                int timeout,  
                                                TUIRoomDefine.RequestCallback callbac
```

Parameter	Type	Meaning
userId	String	User ID
device	<a href="#">TUIRoomDefine.MediaDevice</a>	Device
timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout detection and will not trigger timeout Callback
callback	TUIRoomDefine.ActionCallback	Operation result Callback

### closeRemoteDeviceByAdmin

Close Remote user Media device, only Administrator or Group owner can call

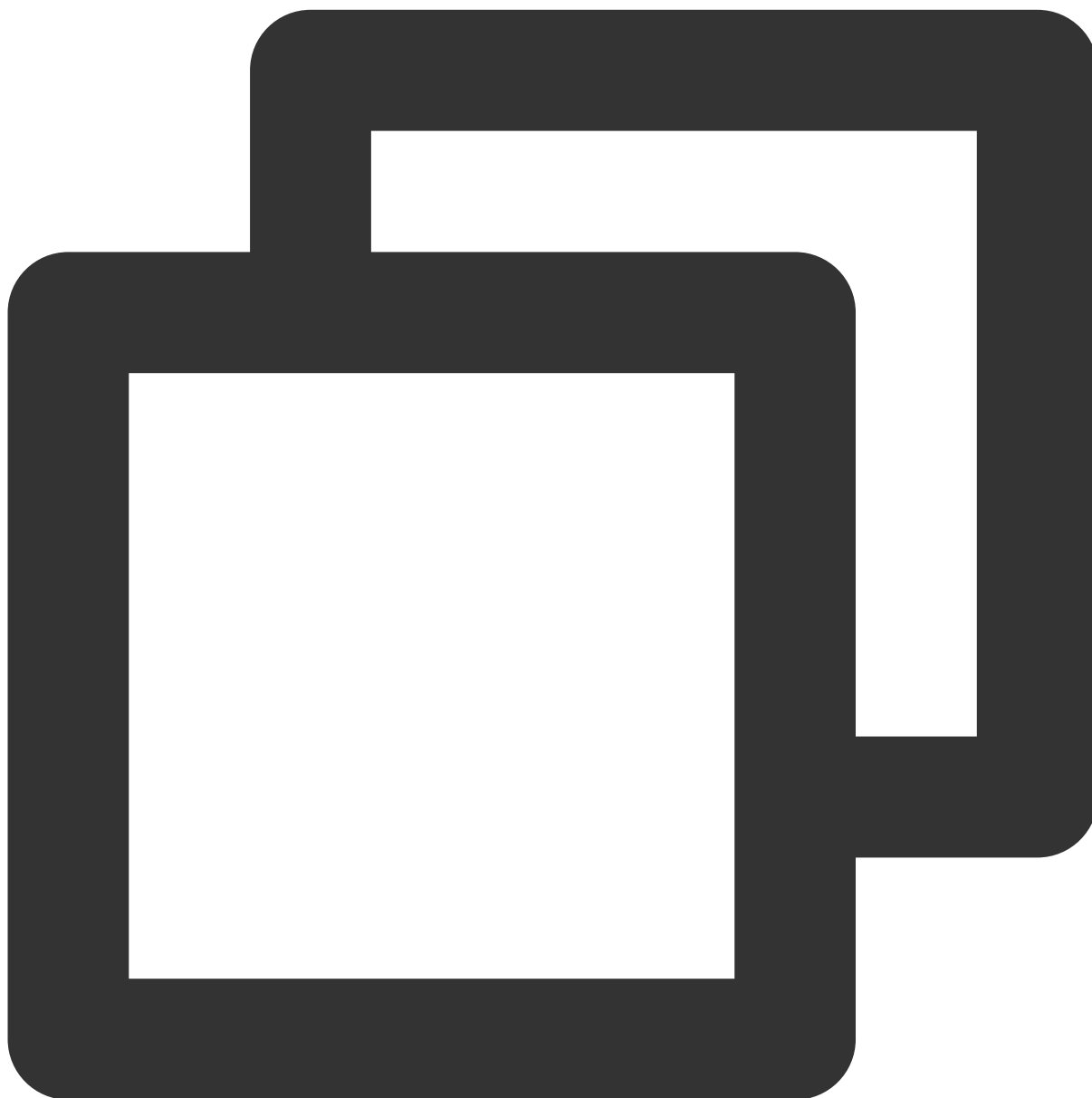


```
void closeRemoteDeviceByAdmin(String userId,  
                               TUIRoomDefine.MediaDevice device,  
                               TUIRoomDefine.ActionCallback callback);
```

Parameter	Type	Meaning
device	<a href="#">TUIRoomDefine.MediaDevice</a>	Device
isDisable	boolean	Whether to Disable
callback	TUIRoomDefine.ActionCallback	Operation result Callback

## applyToAdminToOpenLocalDevice

Lock all users' Media device management



```
TUIRoomDefine.Request applyToAdminToOpenLocalDevice (TUIRoomDefine.MediaDevice device,
int timeout,
TUIRoomDefine.RequestCallback callback)
```

Parameter	Type	Meaning

device	<a href="#">TUIRoomDefine.MediaDevice</a>	Device
timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout detection and will not trigger timeout Callback
callback	TUIRoomDefine.ActionCallback	Operation result Callback

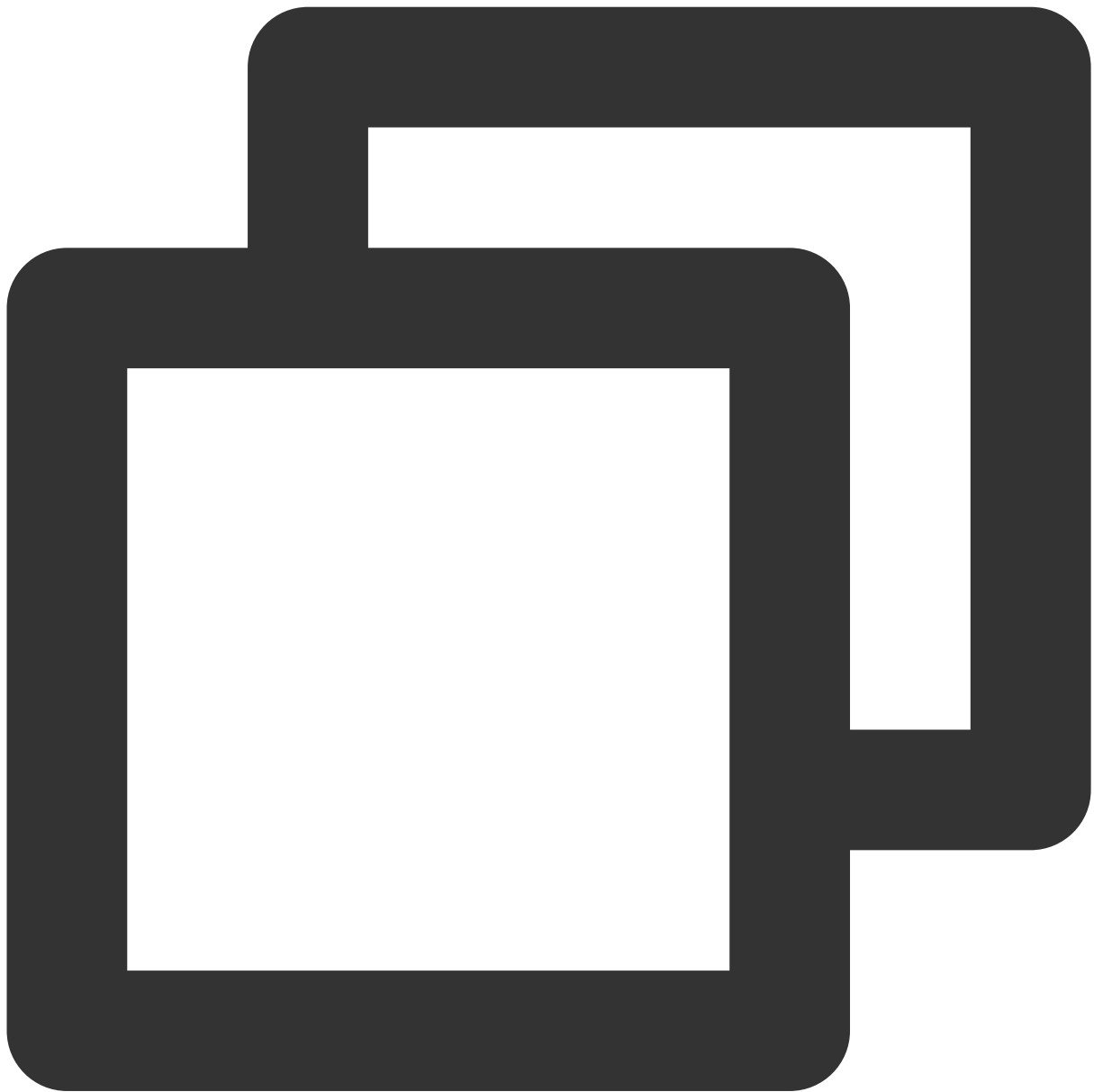
## setMaxSeatCount

Set Maximum number of seats, only supported when entering the room and creating the room

When roomType is RoomType.CONFERENCE (Education and Conference scene), maxSeatCount value is not limited;

When roomType is RoomType.LIVE\_ROOM (Live broadcast scene), maxSeatCount is limited to 16;



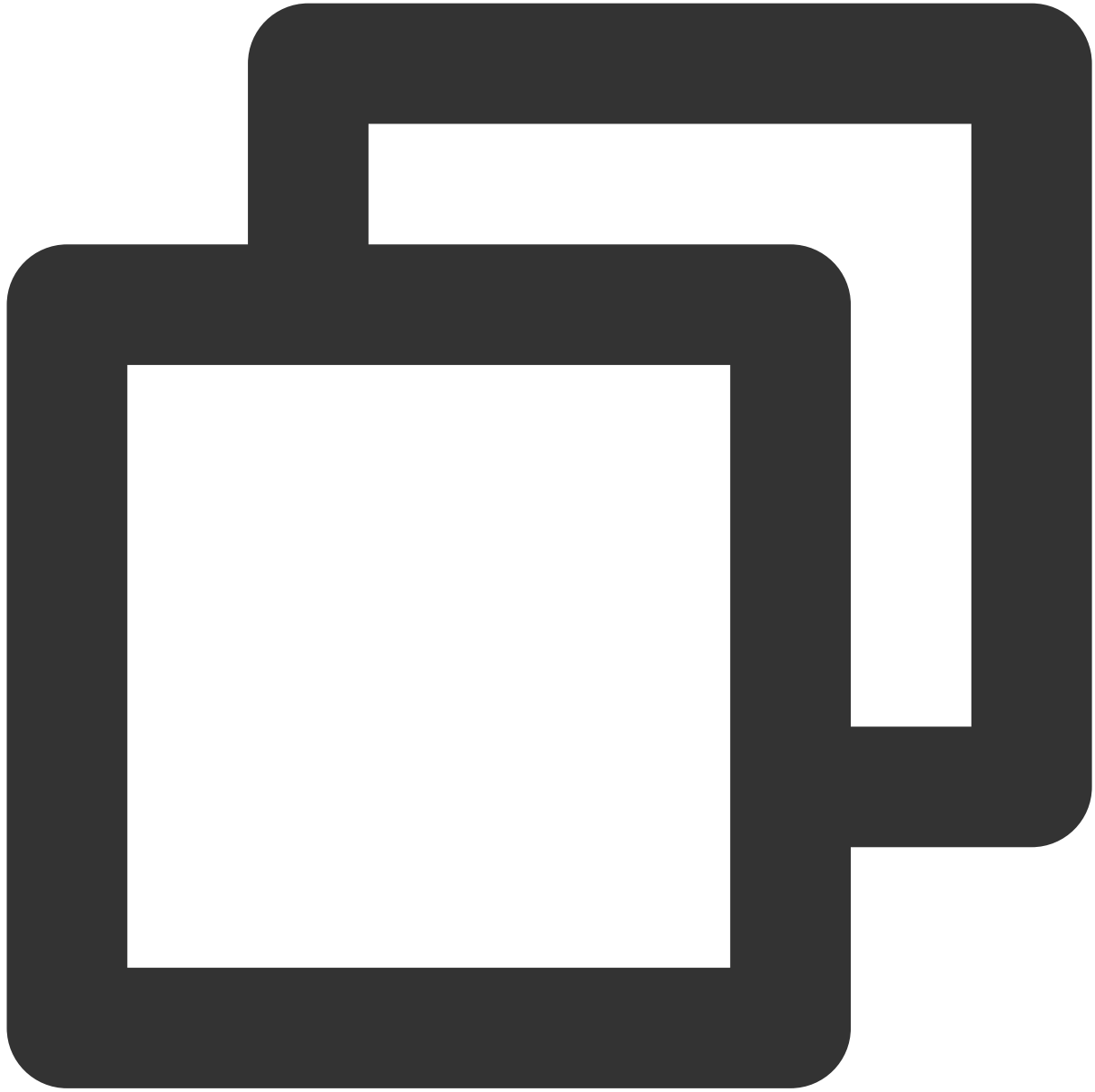


```
void setMaxSeatCount(int maxSeatCount, TUIRoomDefine.ActionCallback callback);
```

Parameter	Type	Meaning
maxSeatCount	int	Maximum number of seats
callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call

### lockSeatByAdmin

Lock seat (including position lock, audio status lock, video status lock)



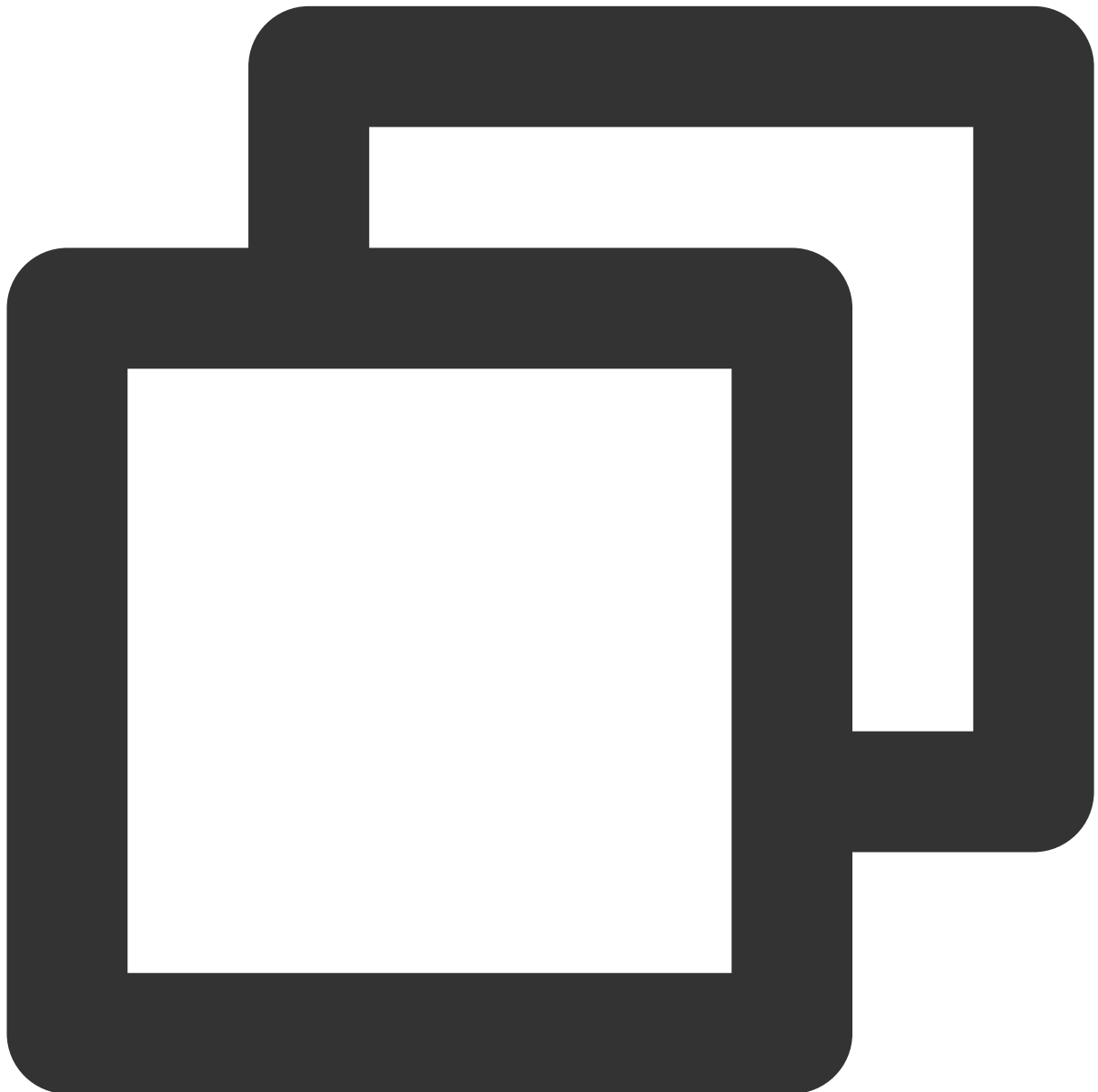
```
void lockSeatByAdmin(int seatIndex,  
                    TUIRoomDefine.SeatLockParams lockParams,  
                    TUIRoomDefine.ActionCallback callback);
```

Parameter	Type	Meaning
seatIndex	int	Seat number
lockParams	<a href="#">TUIRoomDefine.SeatLockParams</a>	Lock microphone parameter

callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call
----------	------------------------------	--

## getSeatList

Get seat list



```
void getSeatList(TUIRoomDefine.GetSeatListCallback callback);
```

Parameter	Type	Meaning

callback	TUIRoomDefine.GetSeatListCallback	Get seat list Callback
----------	-----------------------------------	------------------------

## takeSeat

Local on microphone

### Explanation:

Conference scene: [SPEAK\\_AFTER\\_TAKING\\_SEAT](#) mode requires an application to the host or administrator to allow on microphone, other modes do not support on microphone.

Live broadcast scene: [FREE\\_TO\\_SPEAK](#) mode can freely on microphone, after on microphone, open microphone to speak; [SPEAK\\_AFTER\\_TAKING\\_SEAT](#) mode requires an application to the host or administrator to allow on microphone; other modes do not support on microphone.



```
TUIRoomDefine.Request takeSeat(int seatIndex,  
                                int timeout,  
                                TUIRoomDefine.RequestCallback callback);
```

**Parameters:**

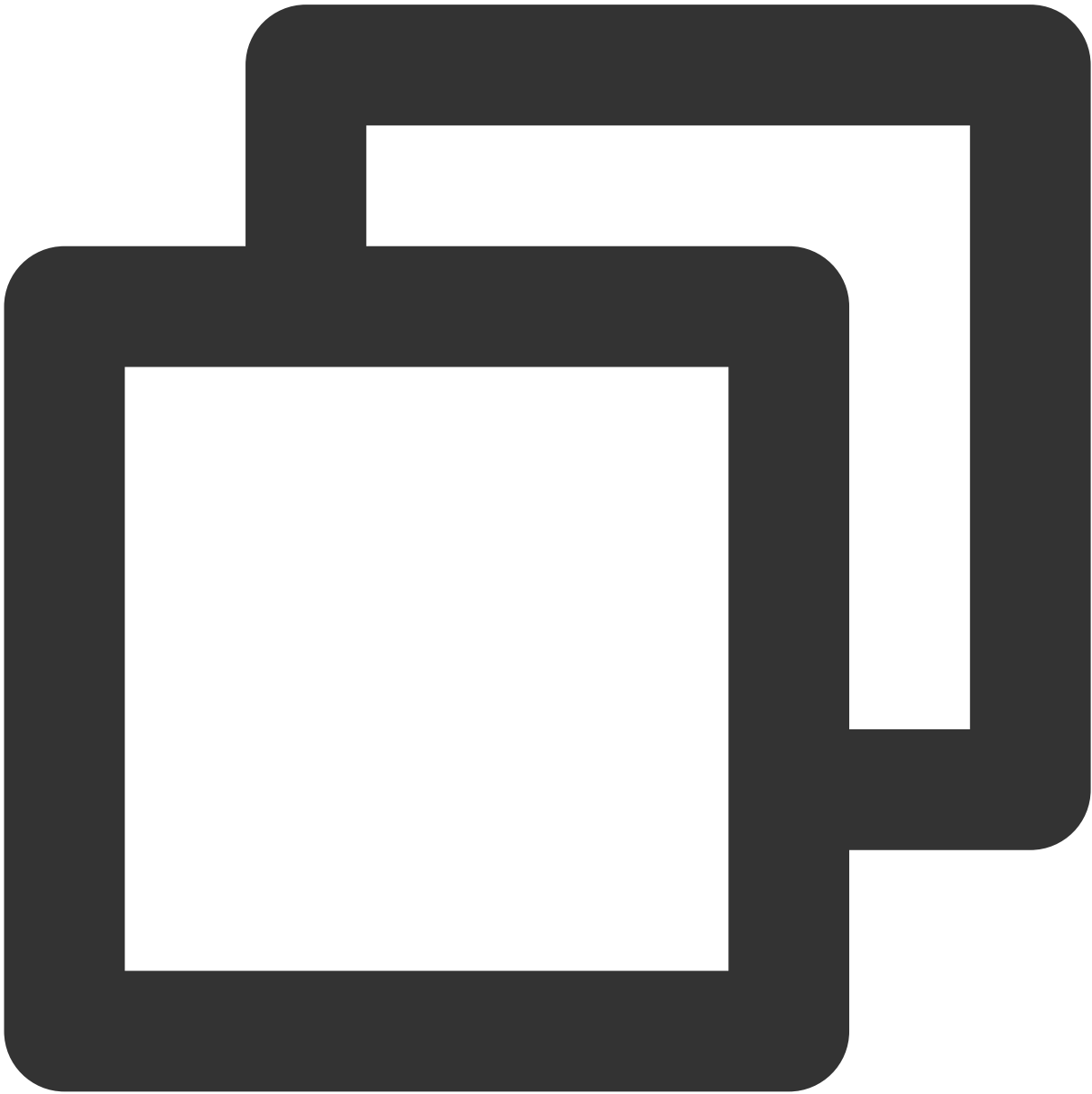
Parameter	Type	Meaning
seatIndex	int	Seat number
timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout

		detection and will not trigger timeout Callback
callback	TUIRoomDefine.RequestCallback	Call interface Callback, used to notify the request Callback status

**Return:** Request body

**leaveSeat**

Local off microphone



```
void leaveSeat(TUIRoomDefine.ActionCallback callback);
```

**Parameters:**

Parameter	Type	Meaning
callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call

**takeUserOnSeatByAdmin**

Host/Administrator Invite User on microphone



```
TUIRoomDefine.Request takeUserOnSeatByAdmin(int seatIndex,  
                                              String userId,  
                                              int timeout,  
                                              TUIRoomDefine.RequestCallback callback)
```

Parameter	Type	Meaning
seatIndex	int	Seat number
userId	String	User ID

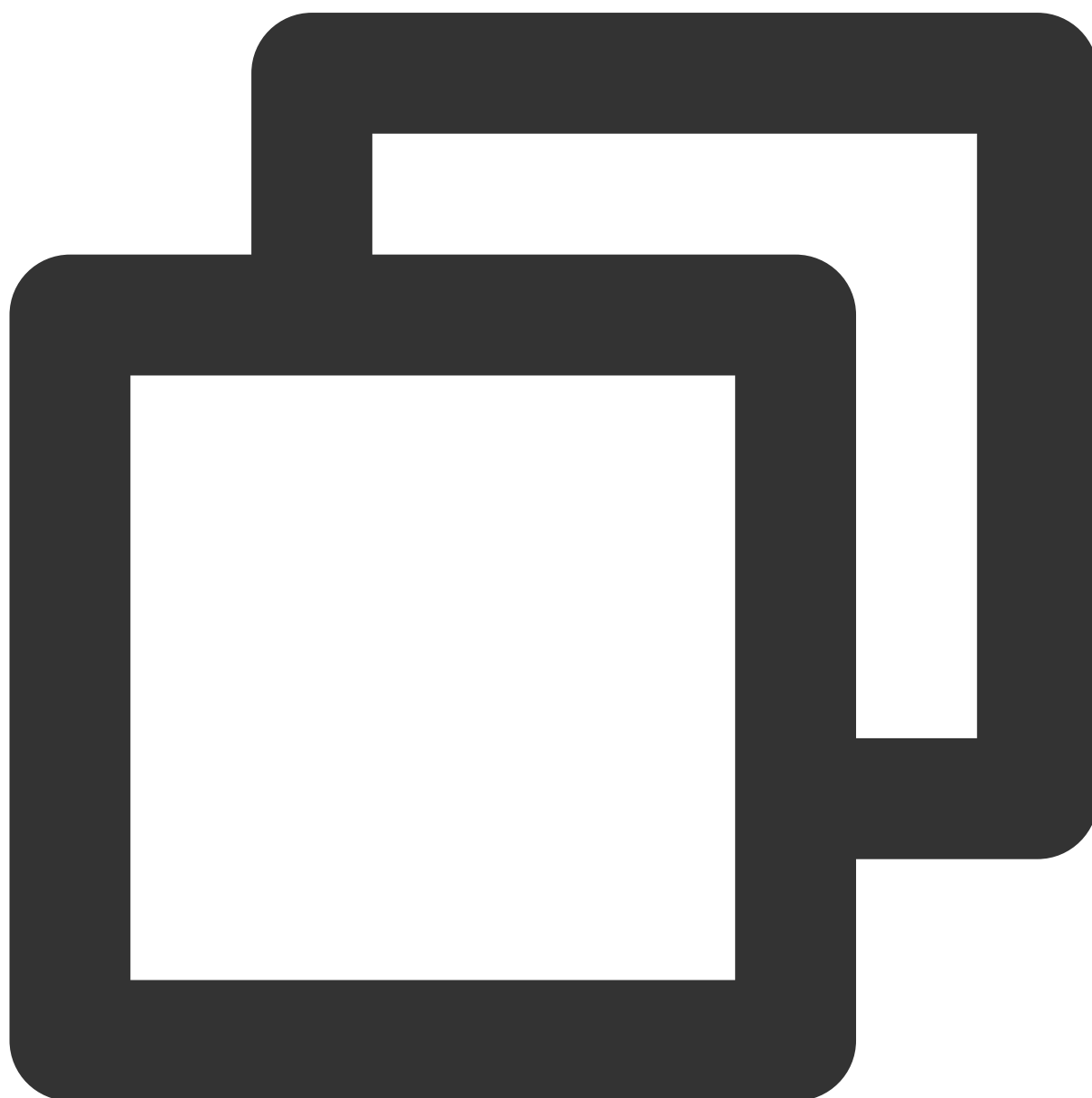


timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout detection and will not trigger timeout Callback
callback	TUIRoomDefine.RequestCallback	Call interface Callback, used to notify the request Callback status

**Return:** Request body

### **kickUserOffSeatByAdmin**

Host/Administrator Take User off microphone

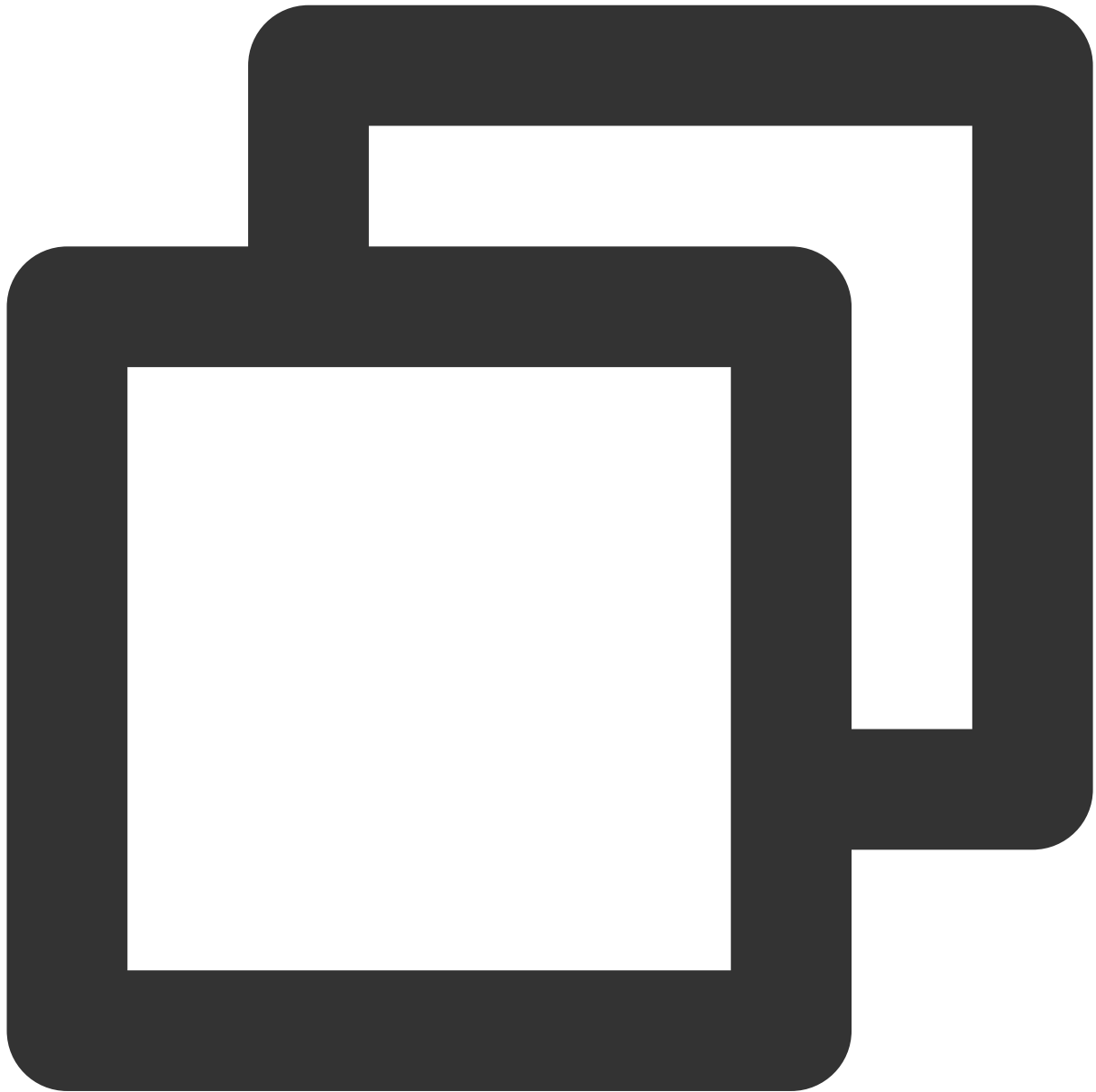


```
void kickUserOffSeatByAdmin(int seatIndex, String userId, TUIRoomDefine.ActionCallb
```

Parameter	Type	Meaning
seatIndex	int	Seat number
userId	String	User ID
callback	TUIRoomDefine.RequestCallback	Call interface Callback, used to notify the request Callback status

## sendTextMessage

Send Text message



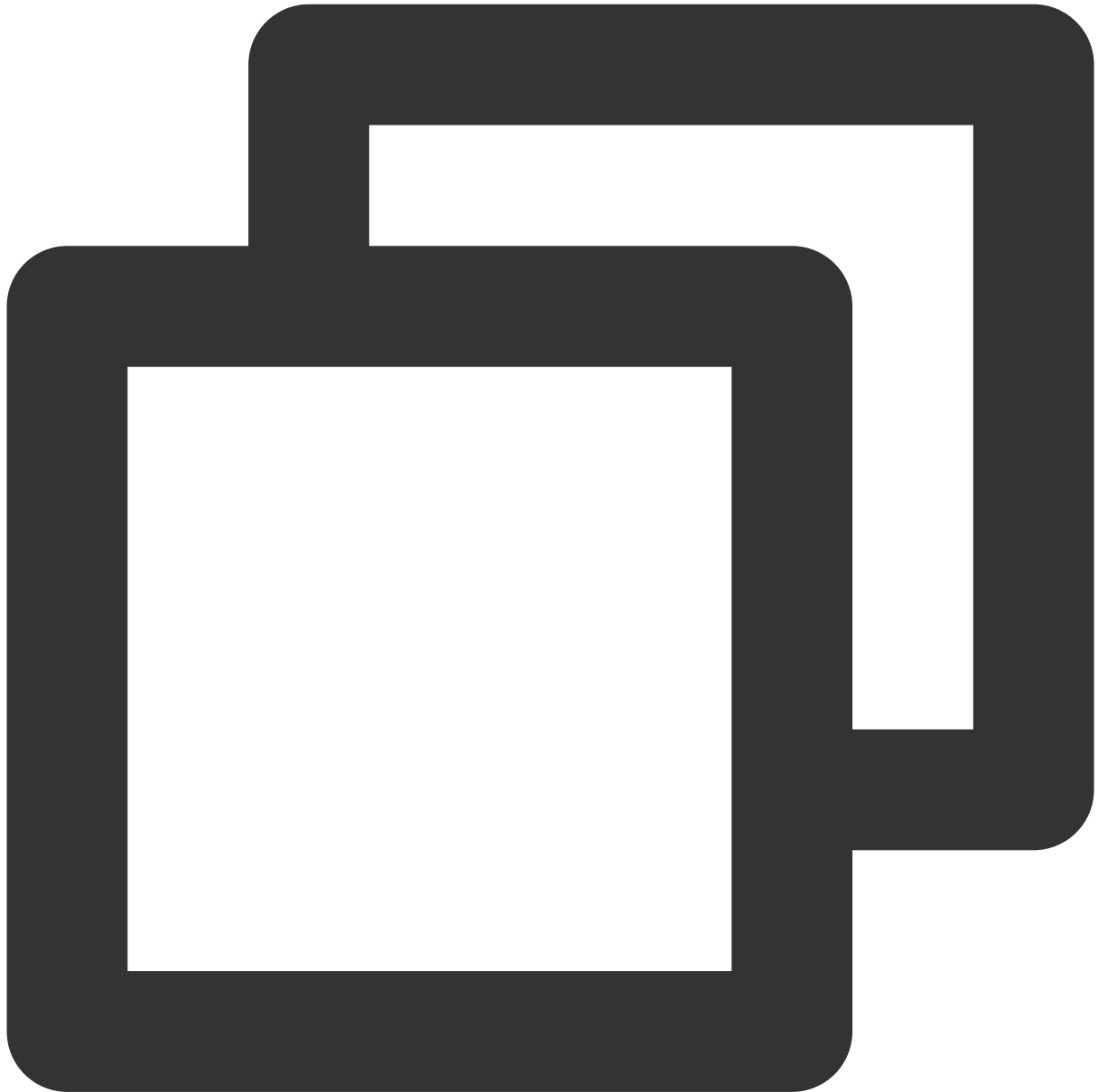
```
void sendTextMessage(String message,  
                     TUIRoomDefine.ActionCallback callback);
```

**Parameters:**

Parameter	Type	Meaning
message	String	Text message Content
callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call

## sendCustomMessage

Send Custom message



```
void sendCustomMessage(String message,  
                        TUIRoomDefine.ActionCallback callback);
```

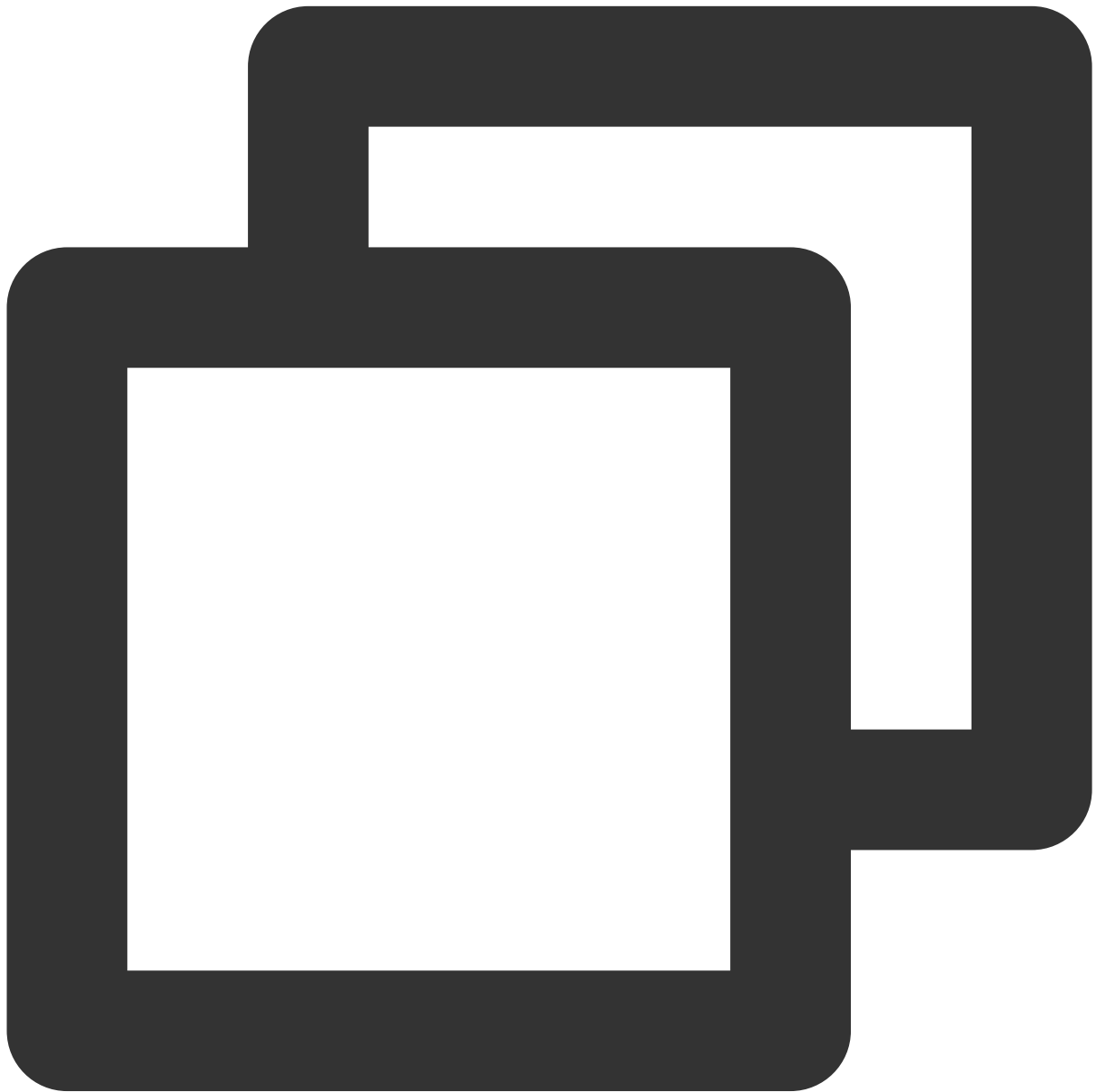
### Parameters:

Parameter	Type	Meaning
message	String	Custom message Content

callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call
----------	------------------------------	--

### **disableSendingMessageByAdmin**

Disable Remote user's Text message sending Ability (only Administrator or Group owner can call)

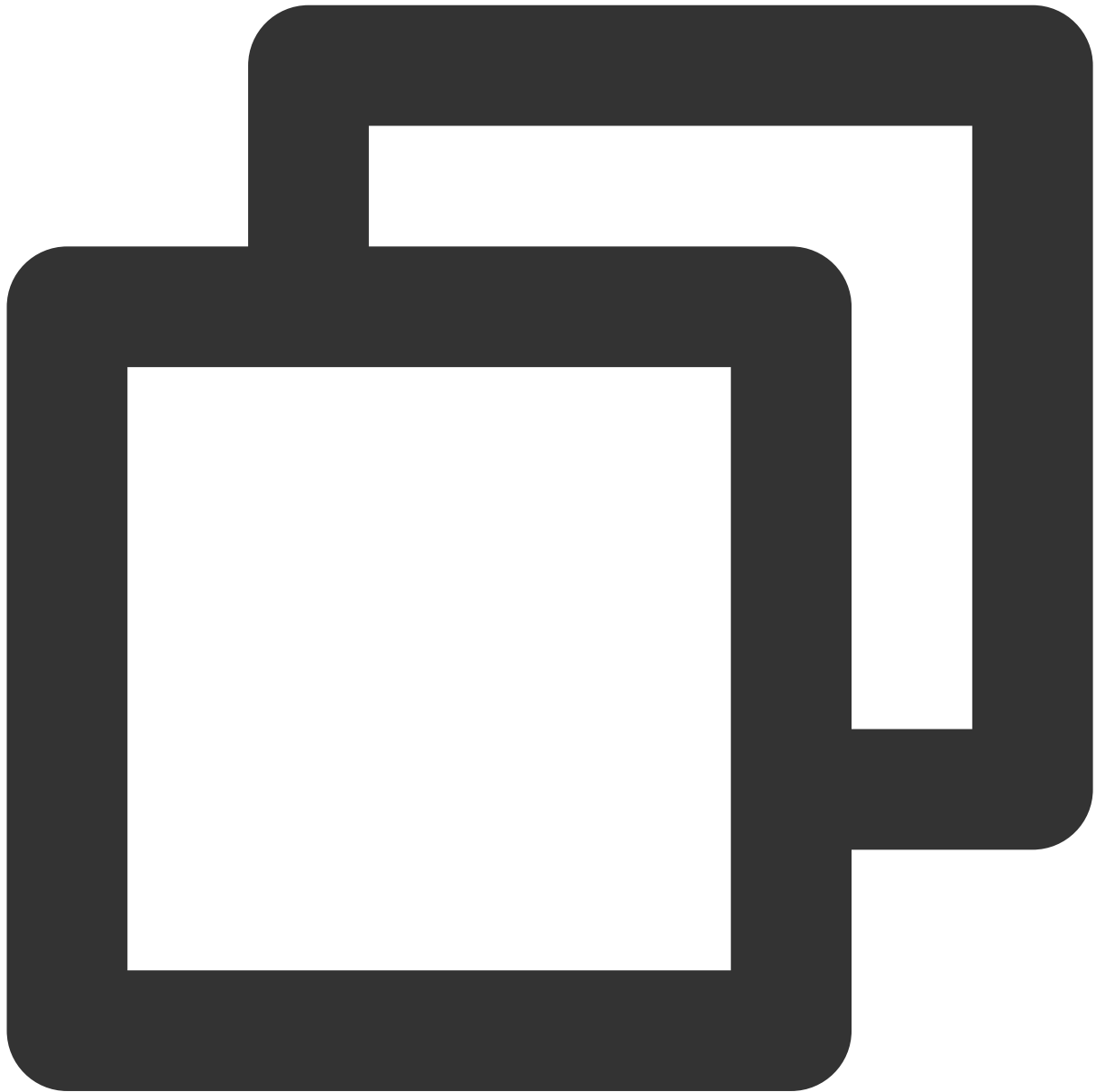


```
void disableSendingMessageByAdmin(String userId,  
                                   boolean isDisable,  
                                   TUIRoomDefine.ActionCallback callback);
```

Parameter	Type	Meaning
userId	String	User ID
isDisable	boolean	Whether to Disable
callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call

### **disableSendingMessageForAllUser**

Disable all users' Text message sending Ability (only Administrator or Group owner can call)

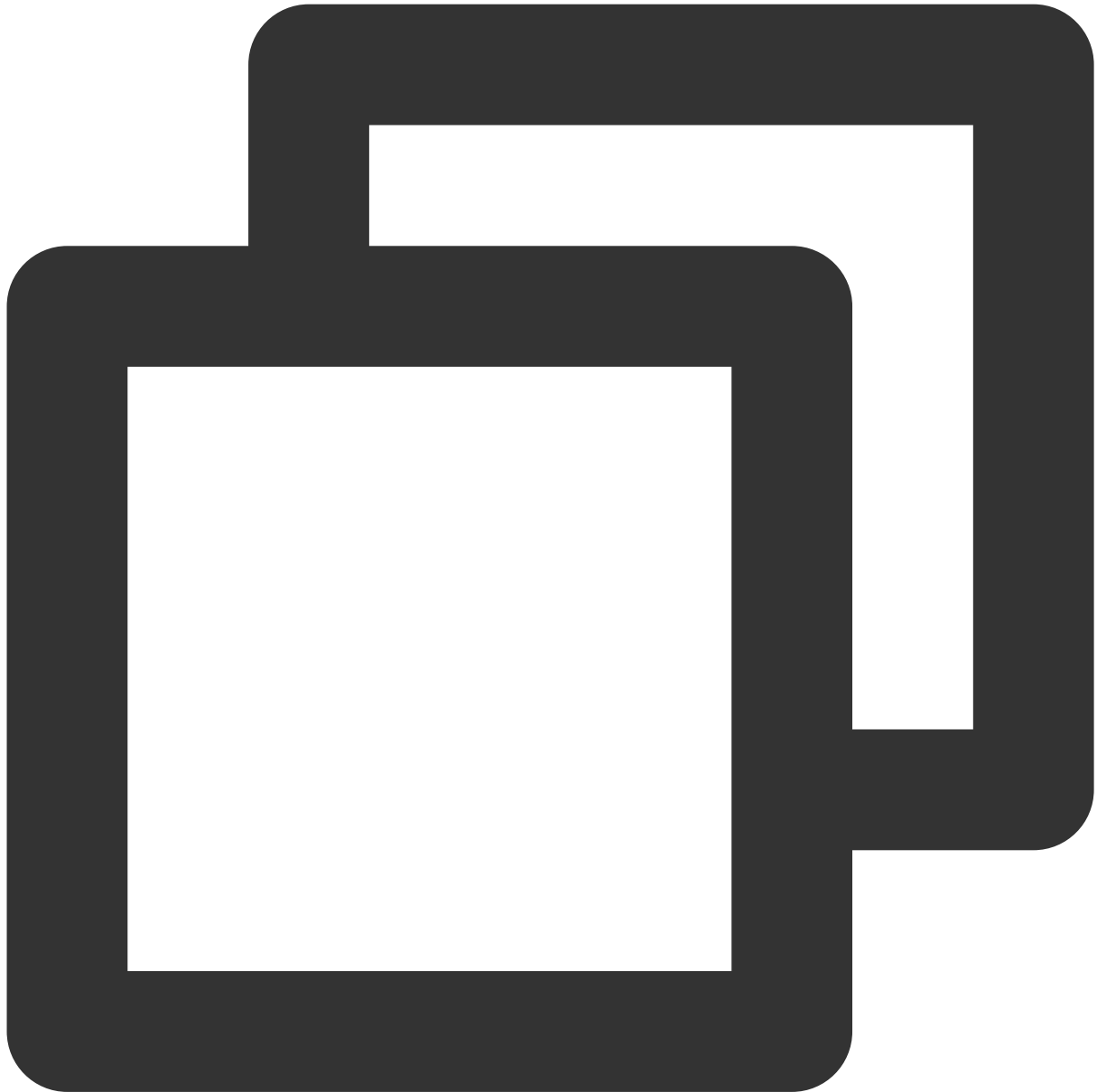


```
void disableSendingMessageForAllUser(boolean isDisable,  
                                     TUIRoomDefine.ActionCallback callback);
```

Parameter	Type	Meaning
isDisable	boolean	Whether to Disable
callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call

## cancelRequest

Cancel sent Request



```
void cancelRequest(String requestId, TUIRoomDefine.ActionCallback callback);
```

### Parameters:

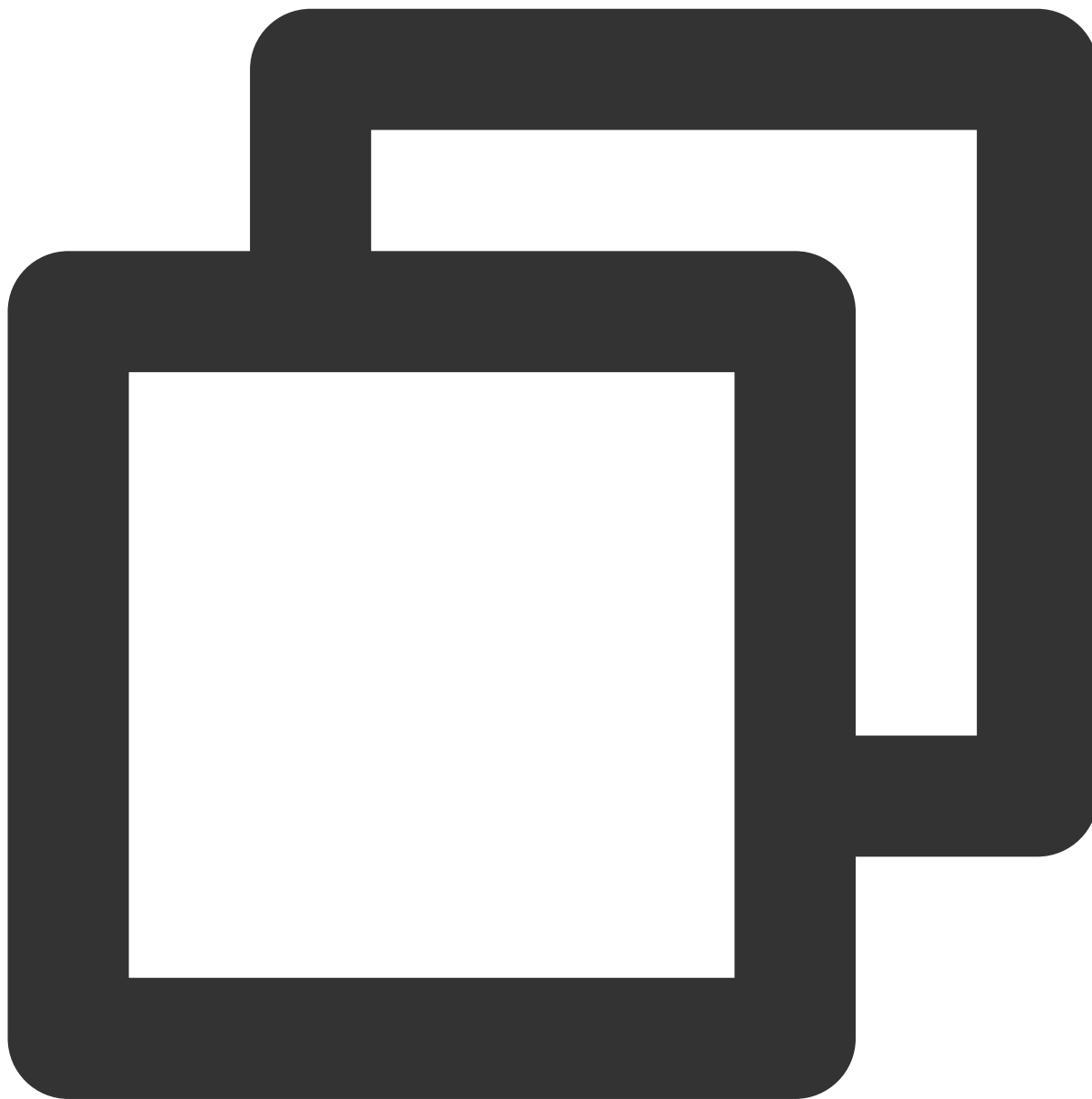
Parameter	Type	Meaning
requestId	String	Request ID
callback	TUIRoomDefine.ActionCallback	Callback of API, used to Notify the Success



		or Failure of the API call
--	--	----------------------------

## responseRemoteRequest

Reply to Remote user's Request



```
void responseRemoteRequest(String requestId,  
                           boolean agree,  
                           TUIRoomDefine.ActionCallback callback);
```

### Parameters:

--	--	--

Parameter	Type	Meaning
requestId	String	Request ID
agree	boolean	Whether to agree
callback	TUIRoomEngineDef.ActionCallback	Callback of API, used to Notify the Success or Failure of the API call

## getTRTCCloud

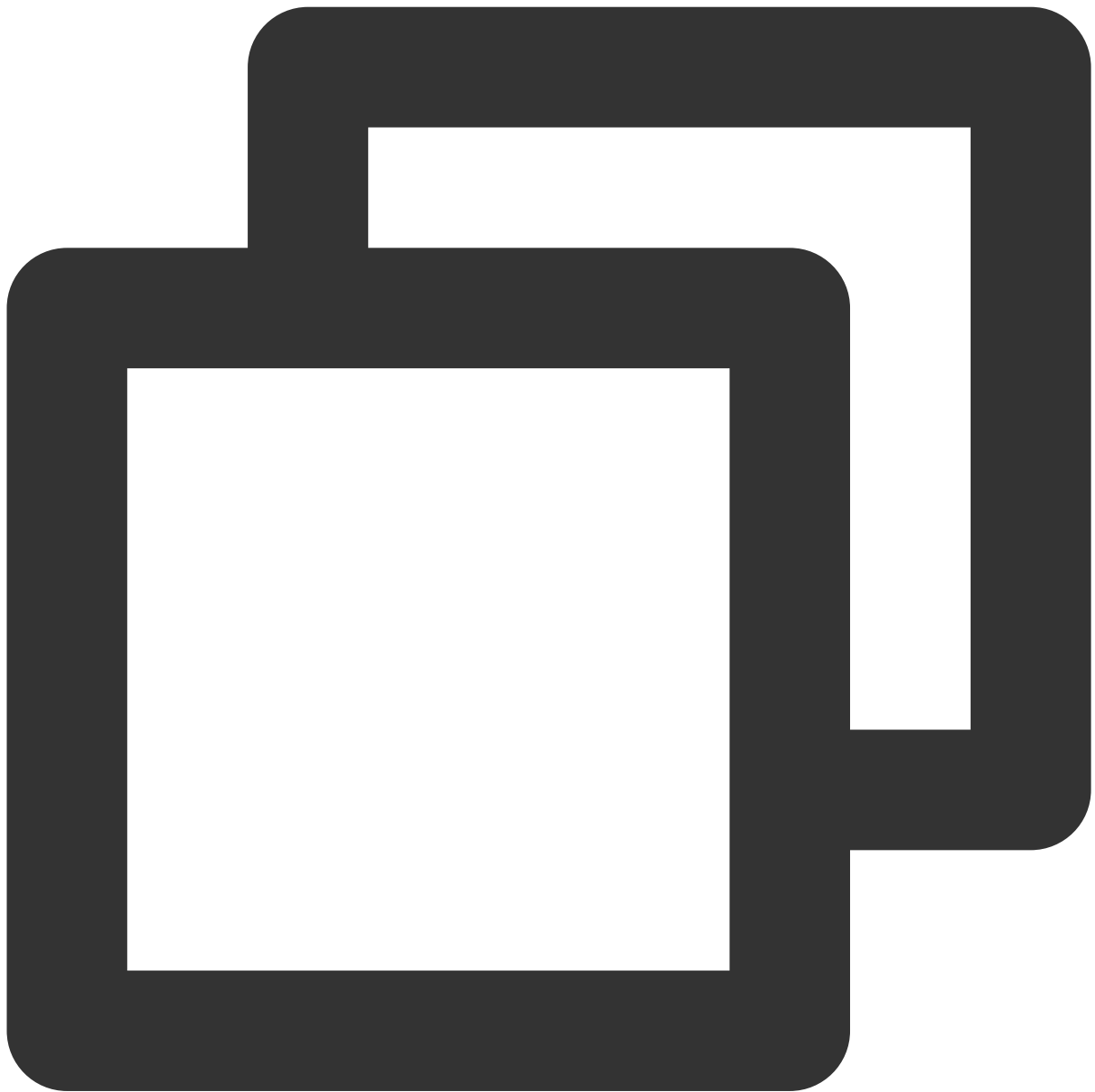
Get TRTCCloud Instance



```
TRTCCloud getTRTCCloud();
```

## getDeviceManager

Get deviceManager, you can use deviceManager's method to get Device list, even Device, switch Device and other functions.



```
TXDeviceManager getDeviceManager();
```

## **getAudioEffectManager**

Get Audio management



```
TXAudioEffectManager getAudioEffectManager();
```

## **getBeautyManager**

Get Beauty management object



```
TXBeautyManager getBeautyManager();
```

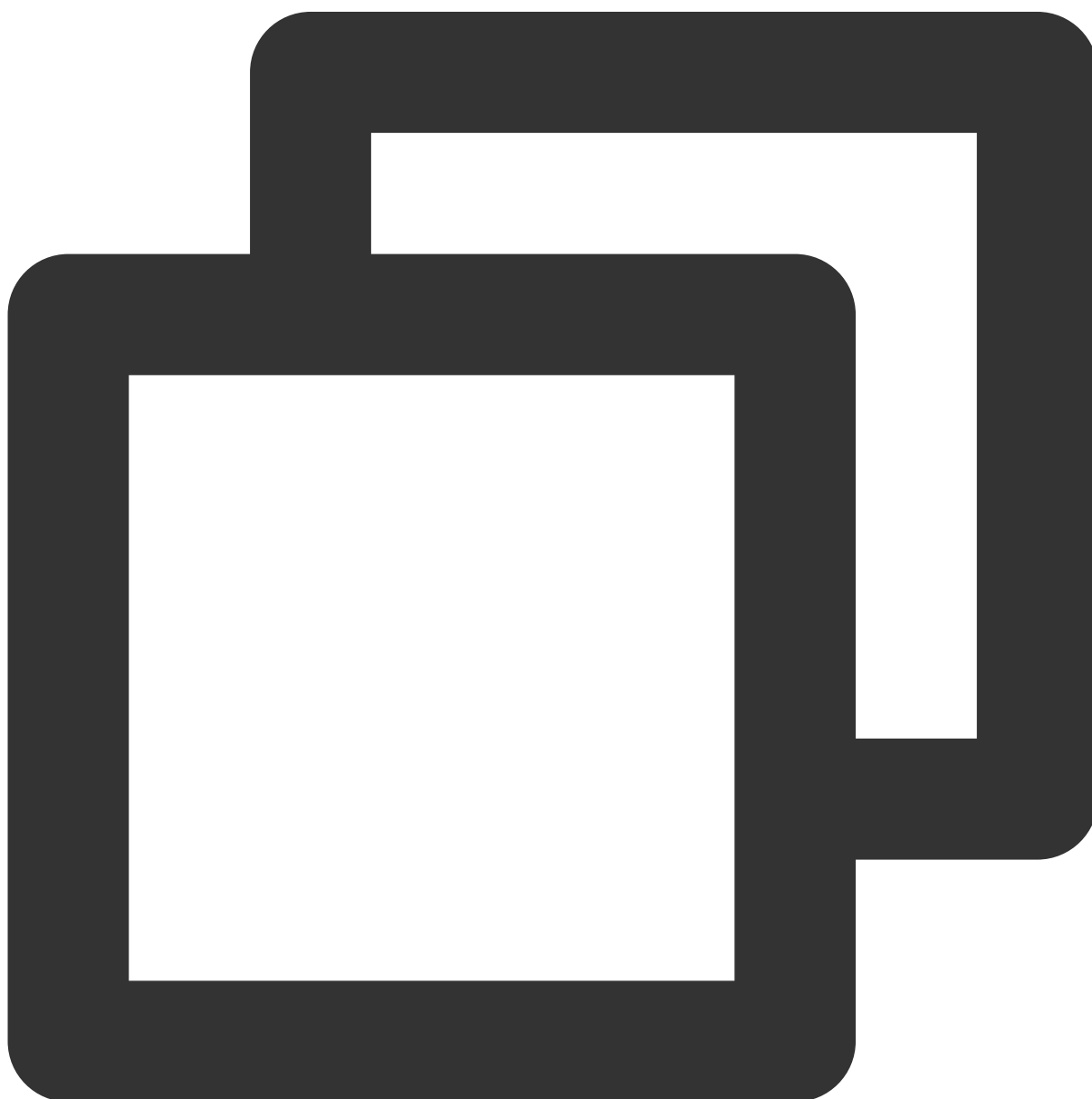
# TUIRoomObserver

Last updated : 2023-10-24 17:07:24

## TUIRoomEngine Event Callback

### **onError**

Error Event Callback

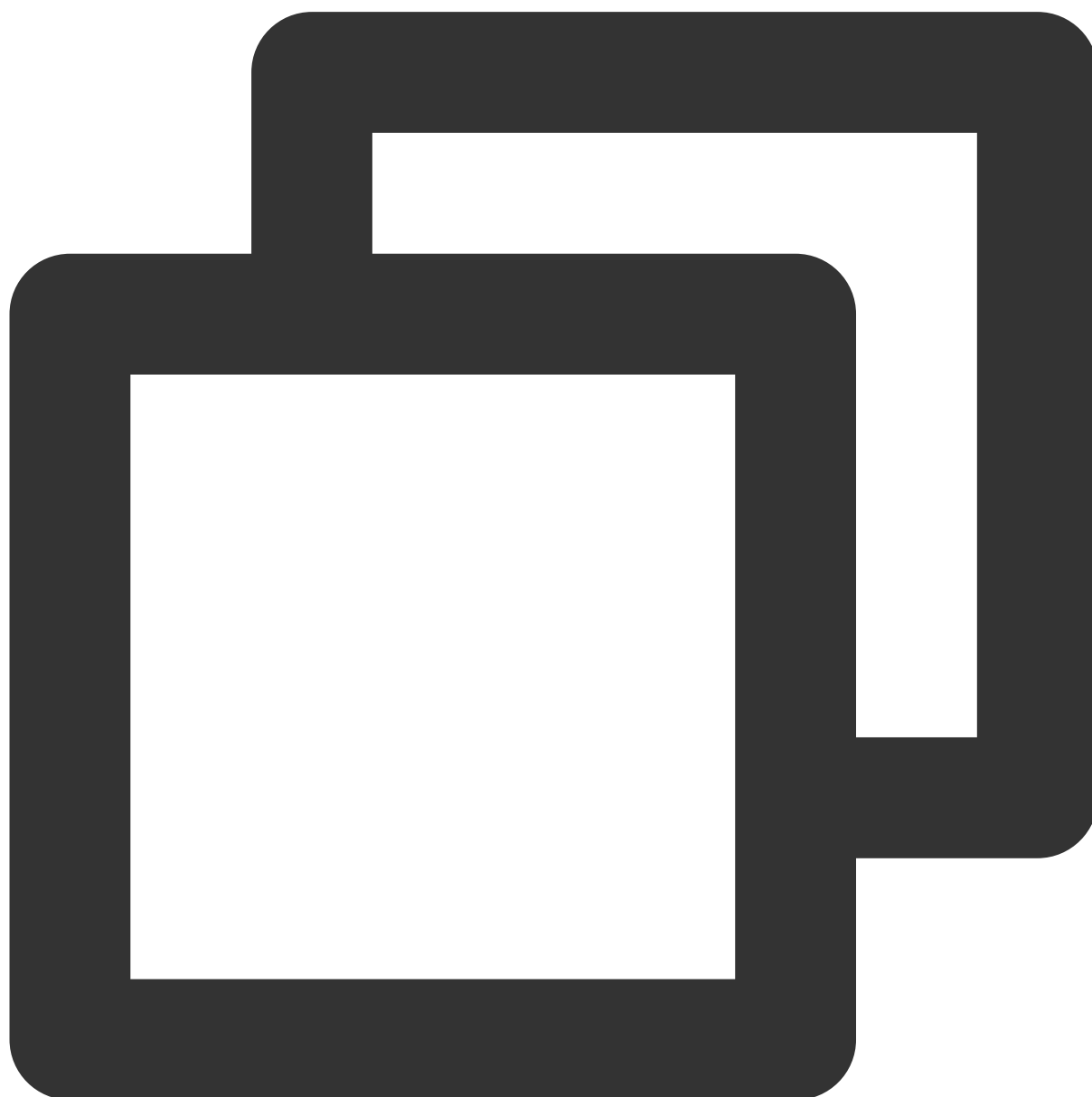


```
void onError(TUICommonDefine.Error errorCode, String message)
```

Parameter	Type	Description
errorCode	TUICommonDefine.Error	Error Code
message	String	Error Message

## onKickedOffLine

Other terminals login and get kicked off event.



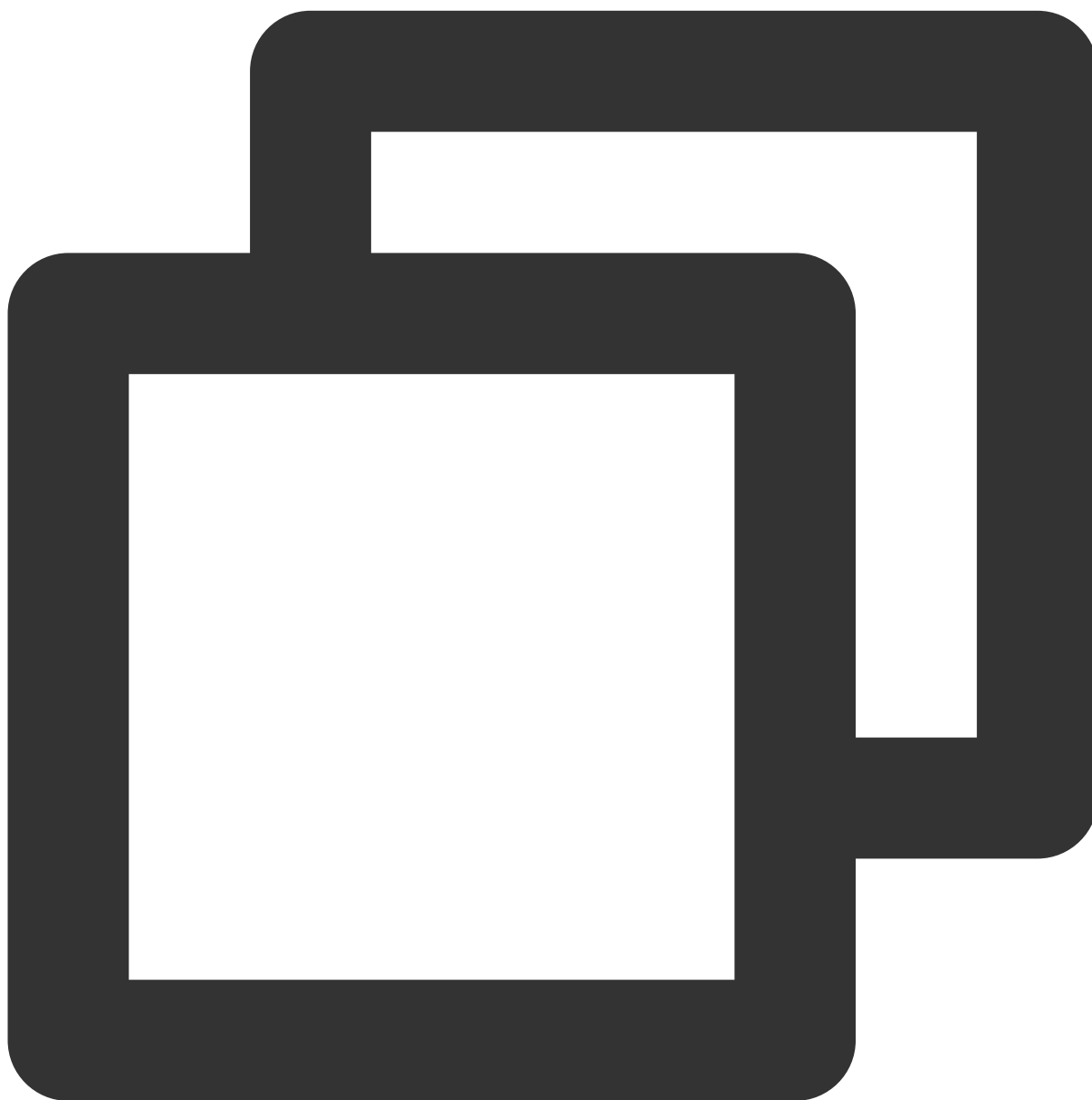


```
void onKickedOffLine(String message)
```

Parameter	Type	Description
message	String	Kicked out description

## onUserSigExpired

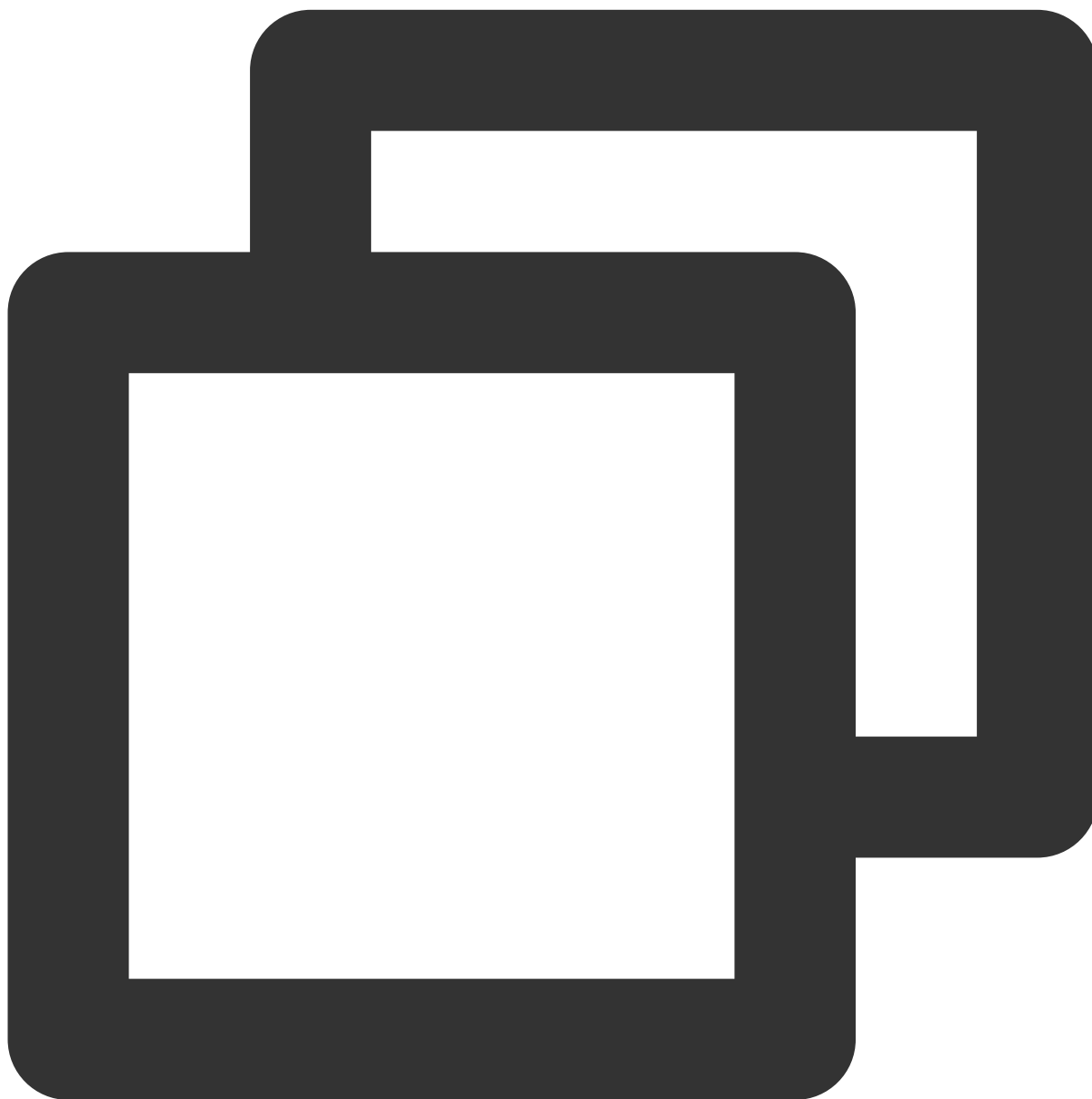
User credential timeout event.



```
void onUserSigExpired()
```

## onRoomNameChanged

Room name change event.



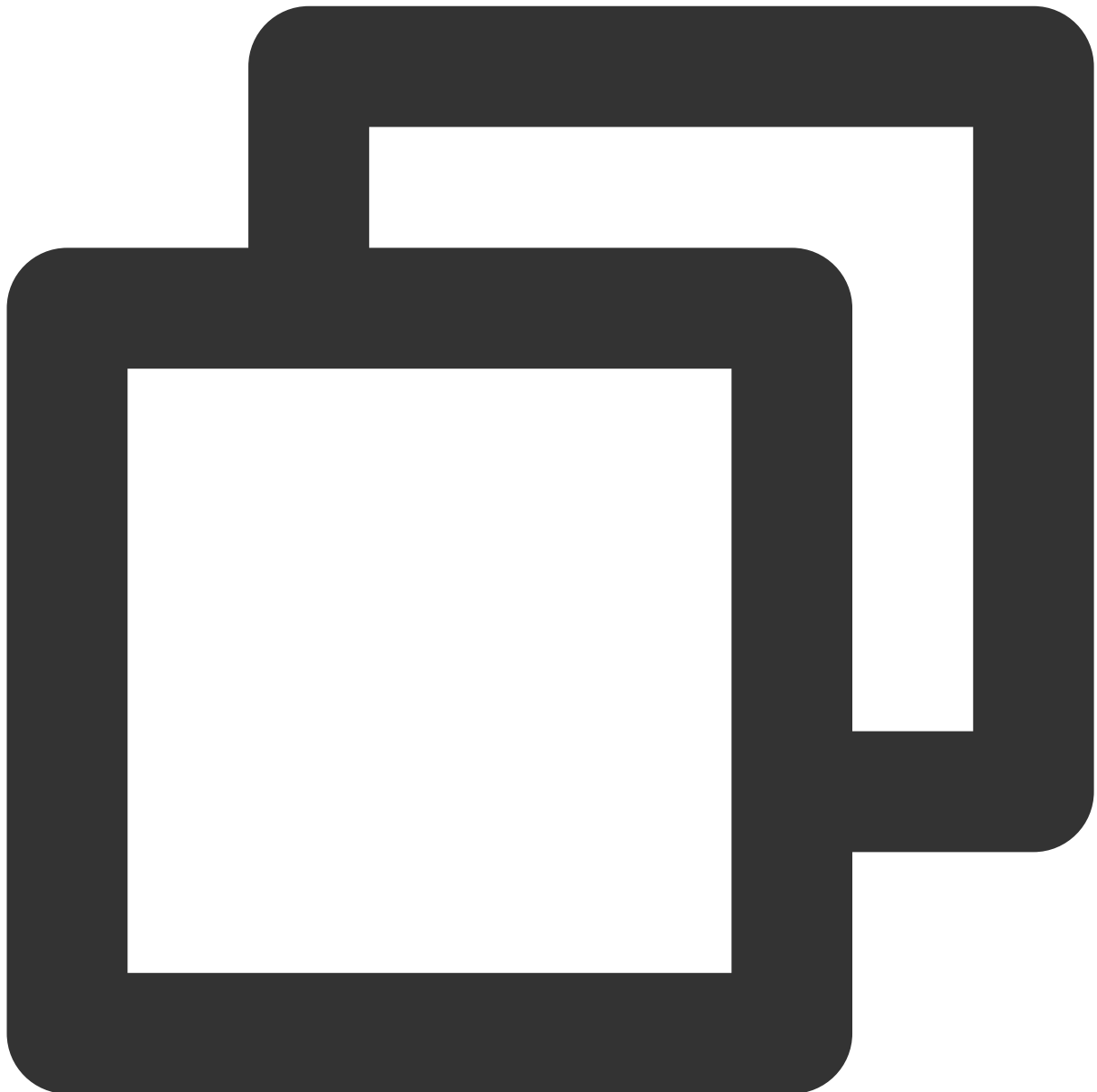
```
void onRoomNameChanged(String roomId, String roomName)
```

Parameter	Type	Description

roomId	String	Room ID
roomName	String	Room Name

### **onAllUserMicrophoneDisableChanged**

Inside the room, all users' mic is disabled event.



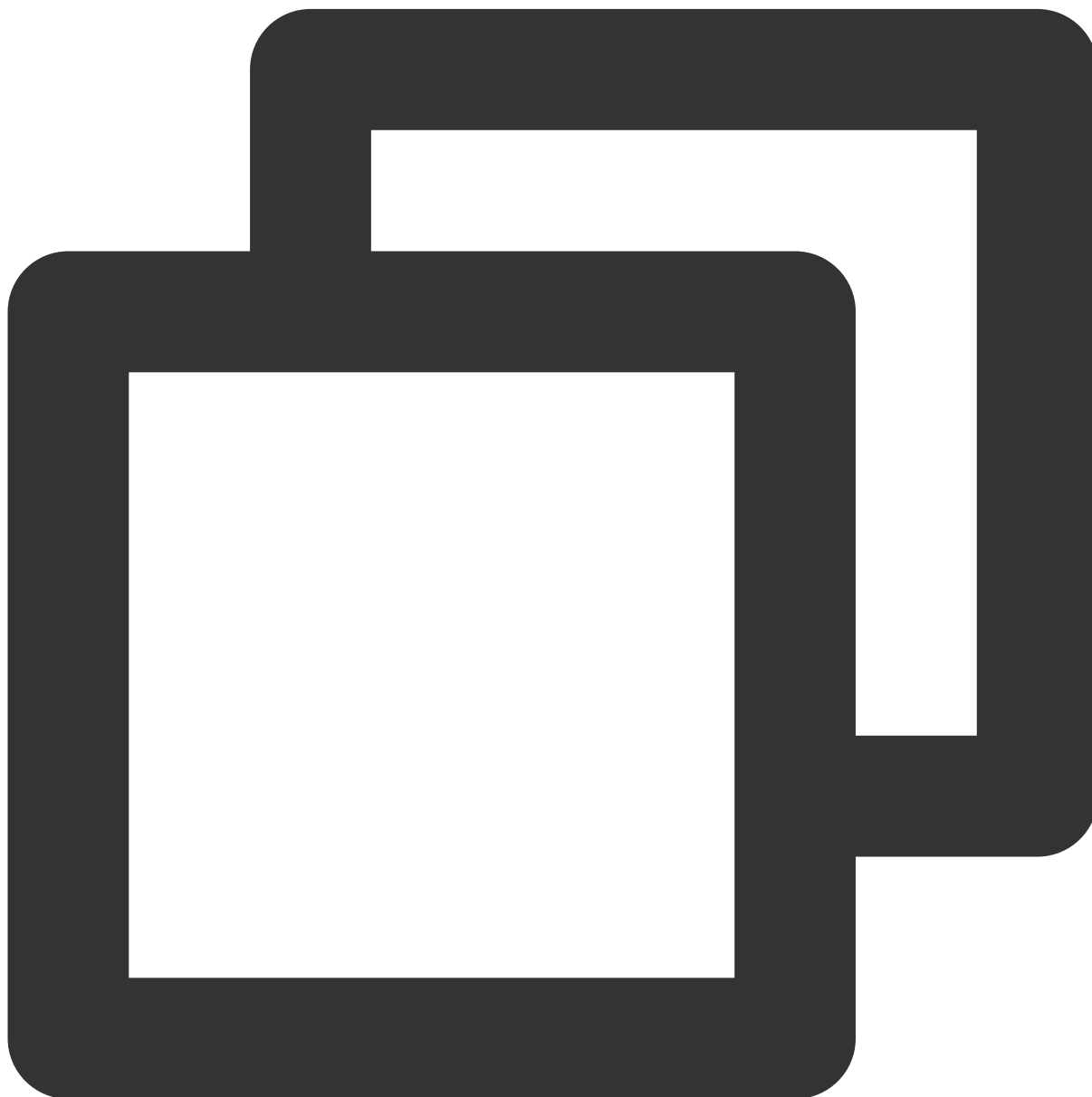
```
void onAllUserMicrophoneDisableChanged(String roomId, boolean isDisable)
```

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

roomId	String	Room ID
isDisable	boolean	Whether it is disabled

### onAllUserCameraDisableChanged

All users' Camera in the Room are disabled event.



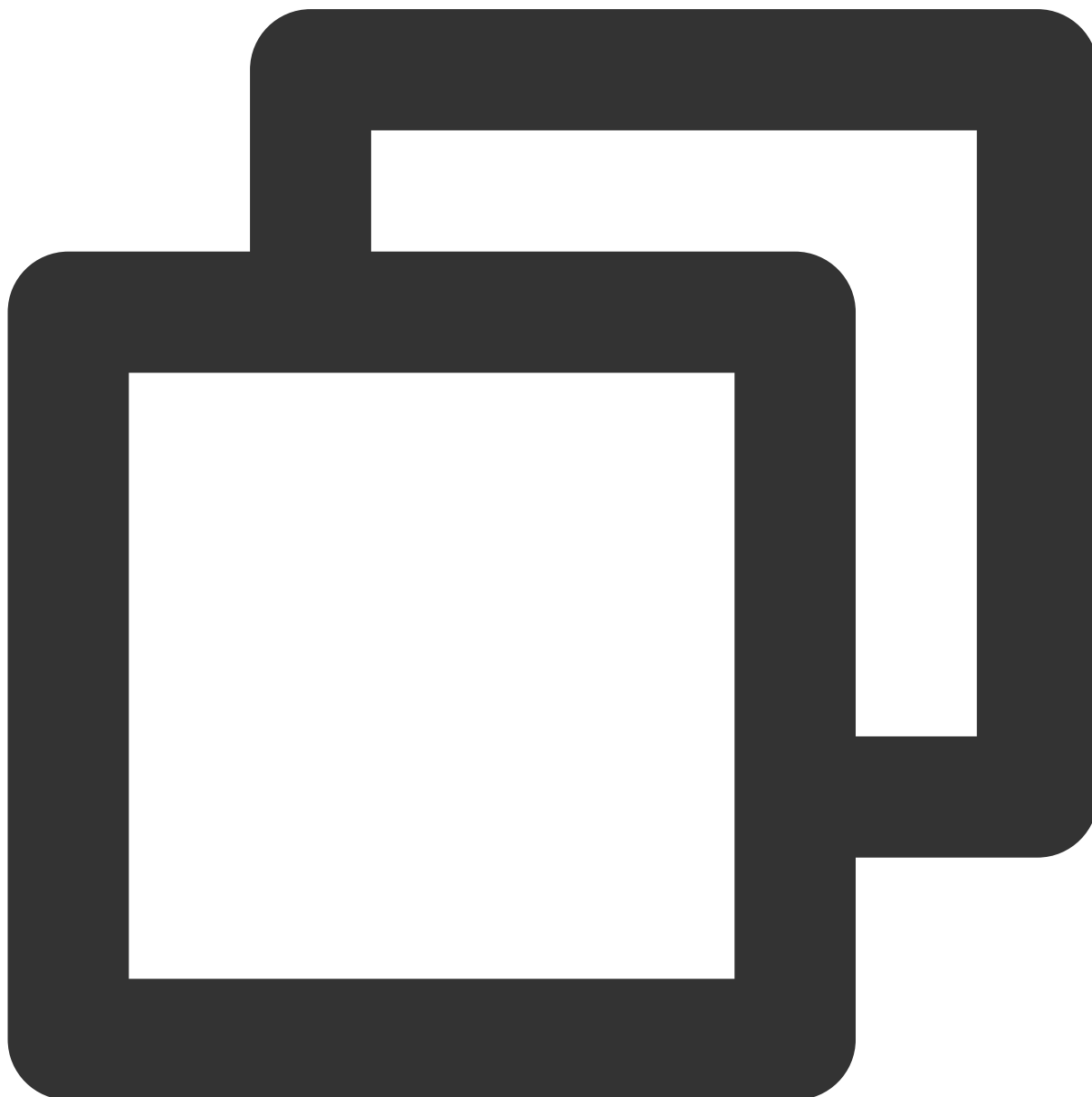
```
void onAllUserCameraDisableChanged(String roomId, boolean isDisable)
```

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

roomId	String	Room ID
isDisable	boolean	Whether it is disabled

### onSendMessageForAllUserDisableChanged

Inside the room, all users' text message sending is disabled event.

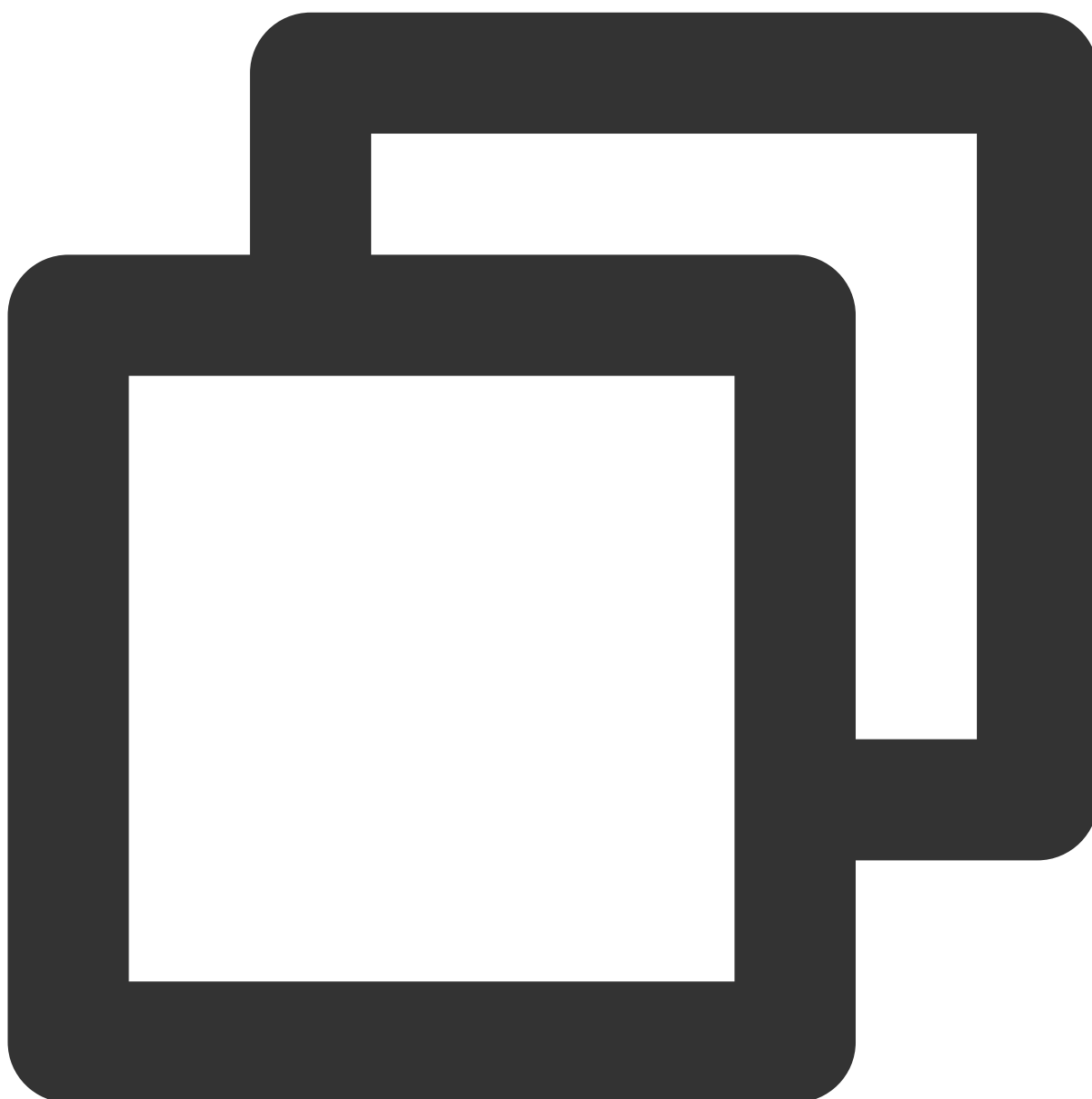


```
void onSendMessageForAllUserDisableChanged(String roomId, boolean isDisable)
```

Parameter	Type	Description
roomId	String	Room ID
isDisable	boolean	Whether it is disabled

## onRoomDismissed

Room dissolution event.

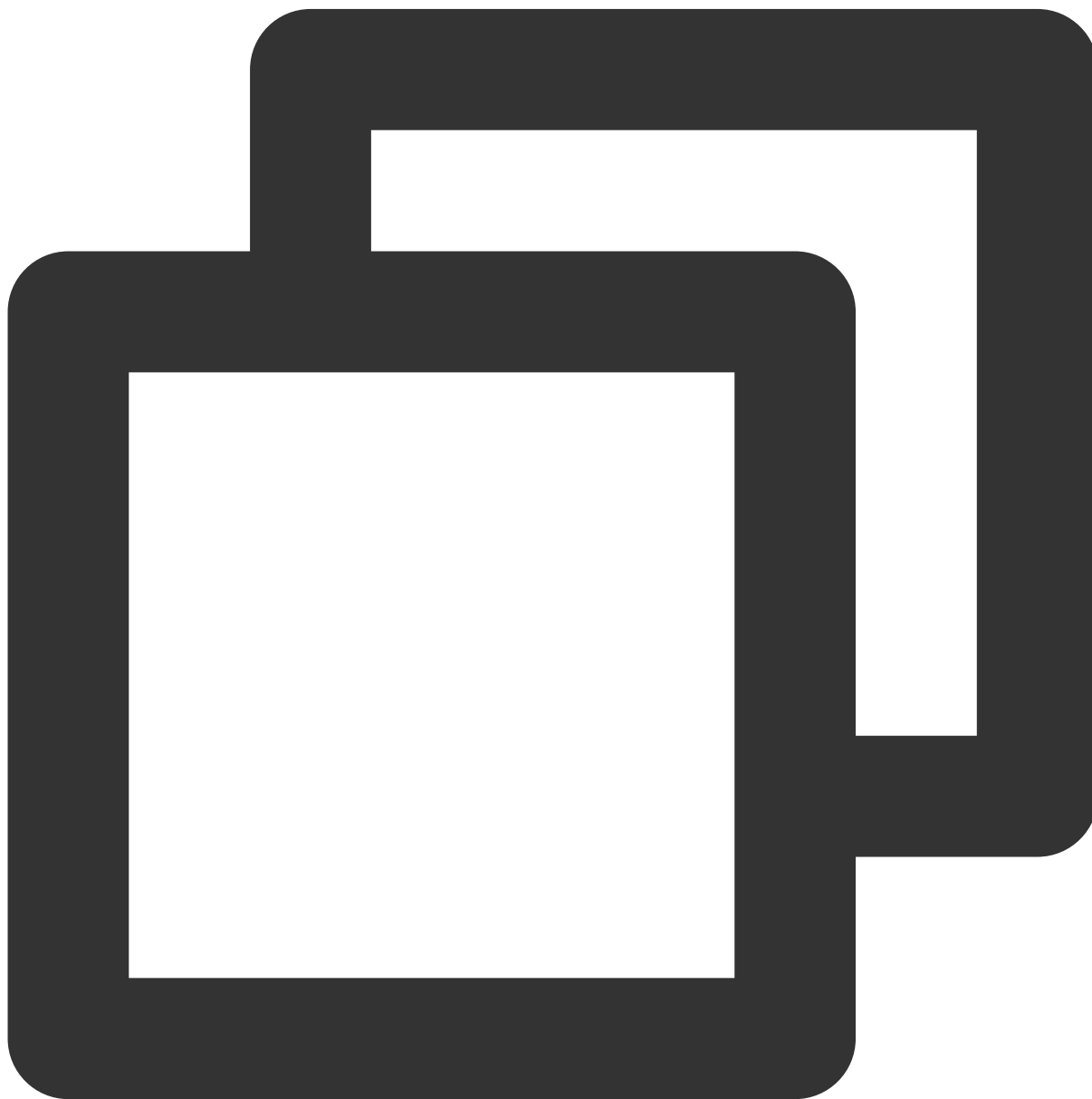


```
void onRoomDismissed(String roomId)
```

Parameter	Type	Description
roomId	String	Room ID

## onKickedOutOfRoom

Kick out of the room event



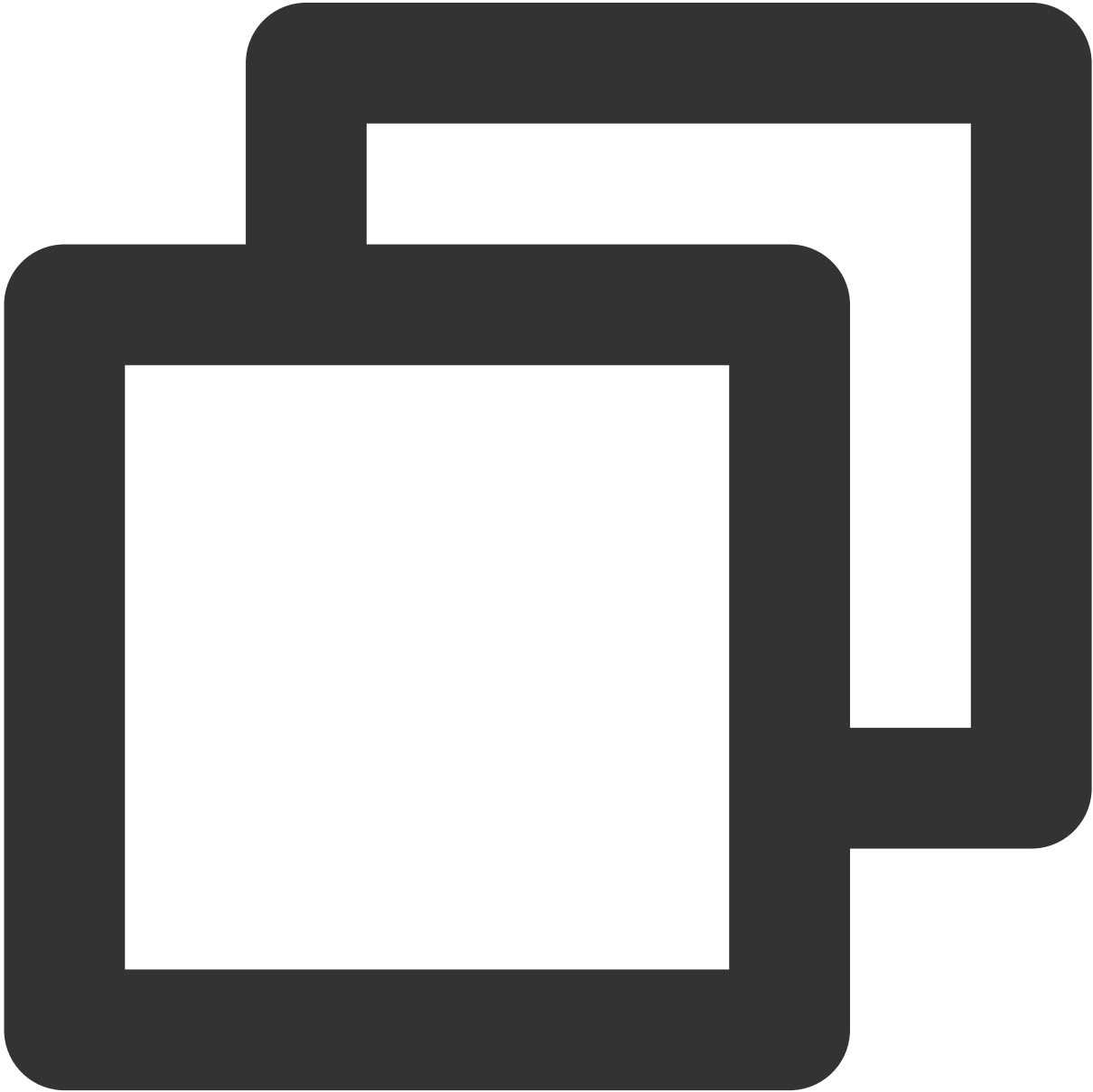
```
void onKickedOutOfRoom(String roomId, String message)
```

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

roomId	String	Room ID
message	String	Description of being kicked out

**onRoomSpeechModeChanged**

Mic control mode changes in the room.



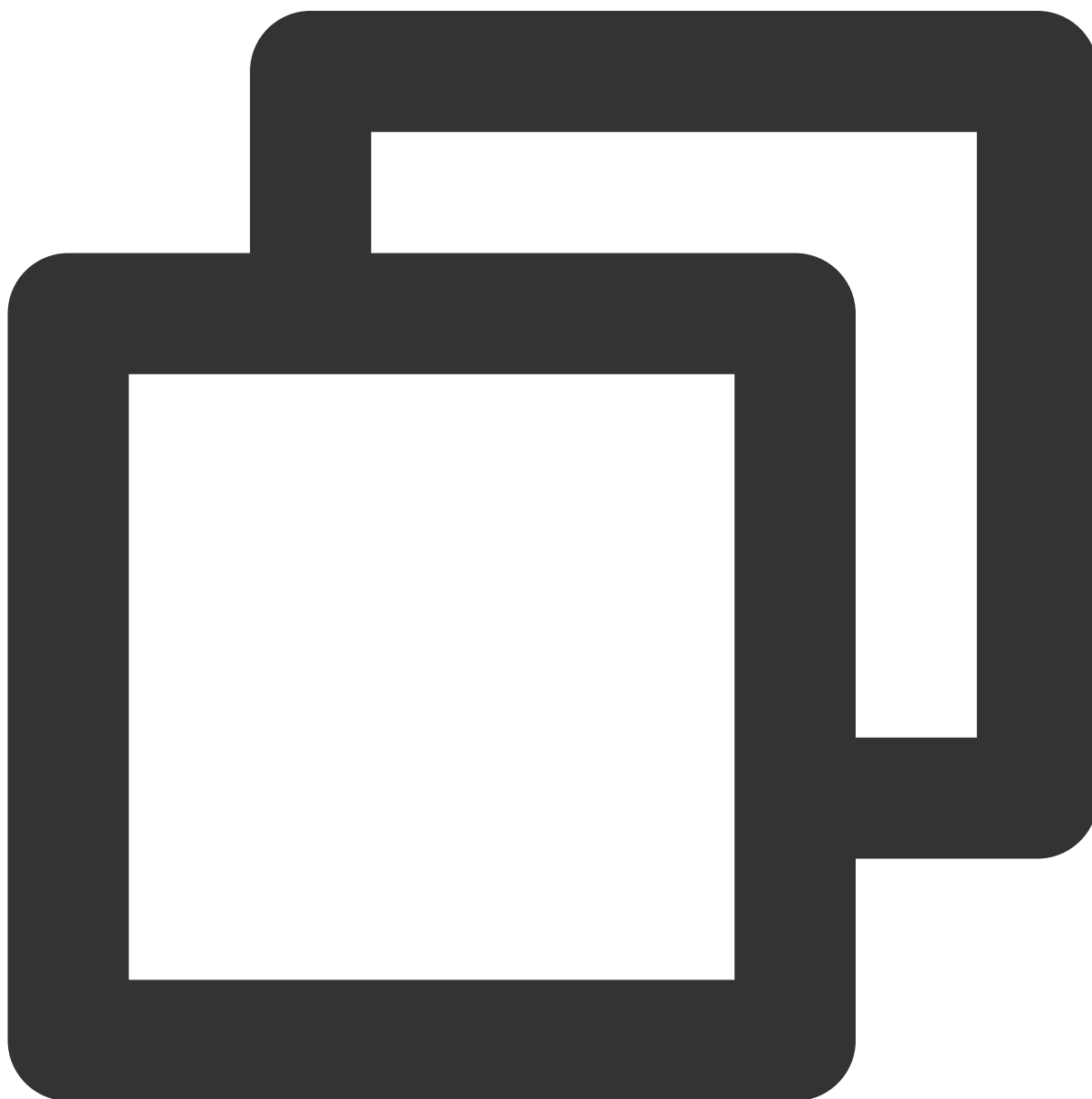
```
void onRoomSpeechModeChanged(String roomId, TUIRoomDefine.SpeechMode speechMode)
```



Parameter	Type	Description
roomId	String	Room ID
speechMode	<a href="#">TUIRoomDefine.SpeechMode</a>	Mic control mode

## onRemoteUserEnterRoom

Remote user enters the room event.

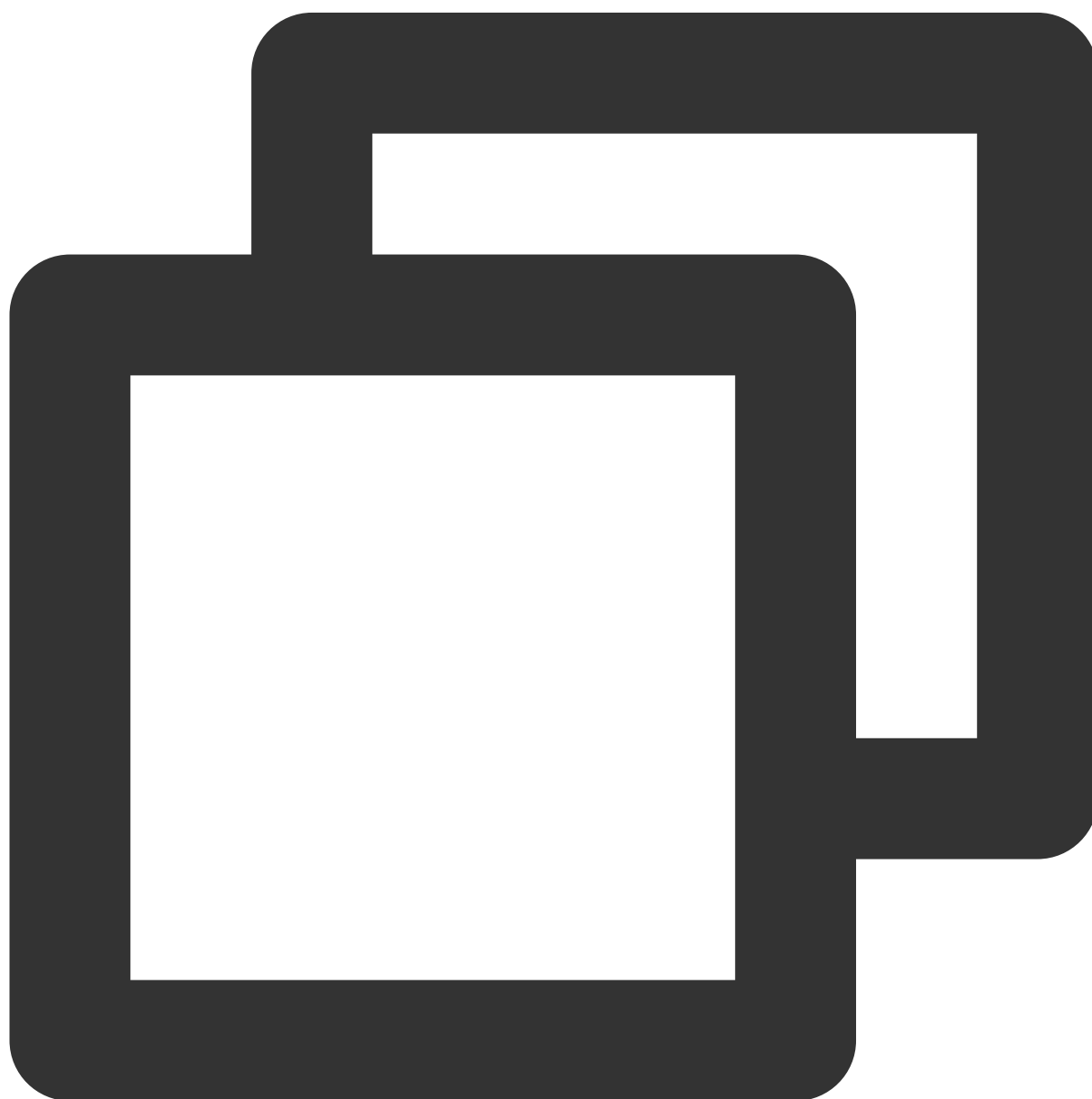


```
void onRemoteUserEnterRoom(String roomId, TUIRoomDefine.UserInfo userInfo)
```

Parameter	Type	Description
roomId	String	Room ID
userInfo	<a href="#">TUIRoomDefine.UserInfo</a>	User information

## onRemoteUserLeaveRoom

Remote user leaves the room event.

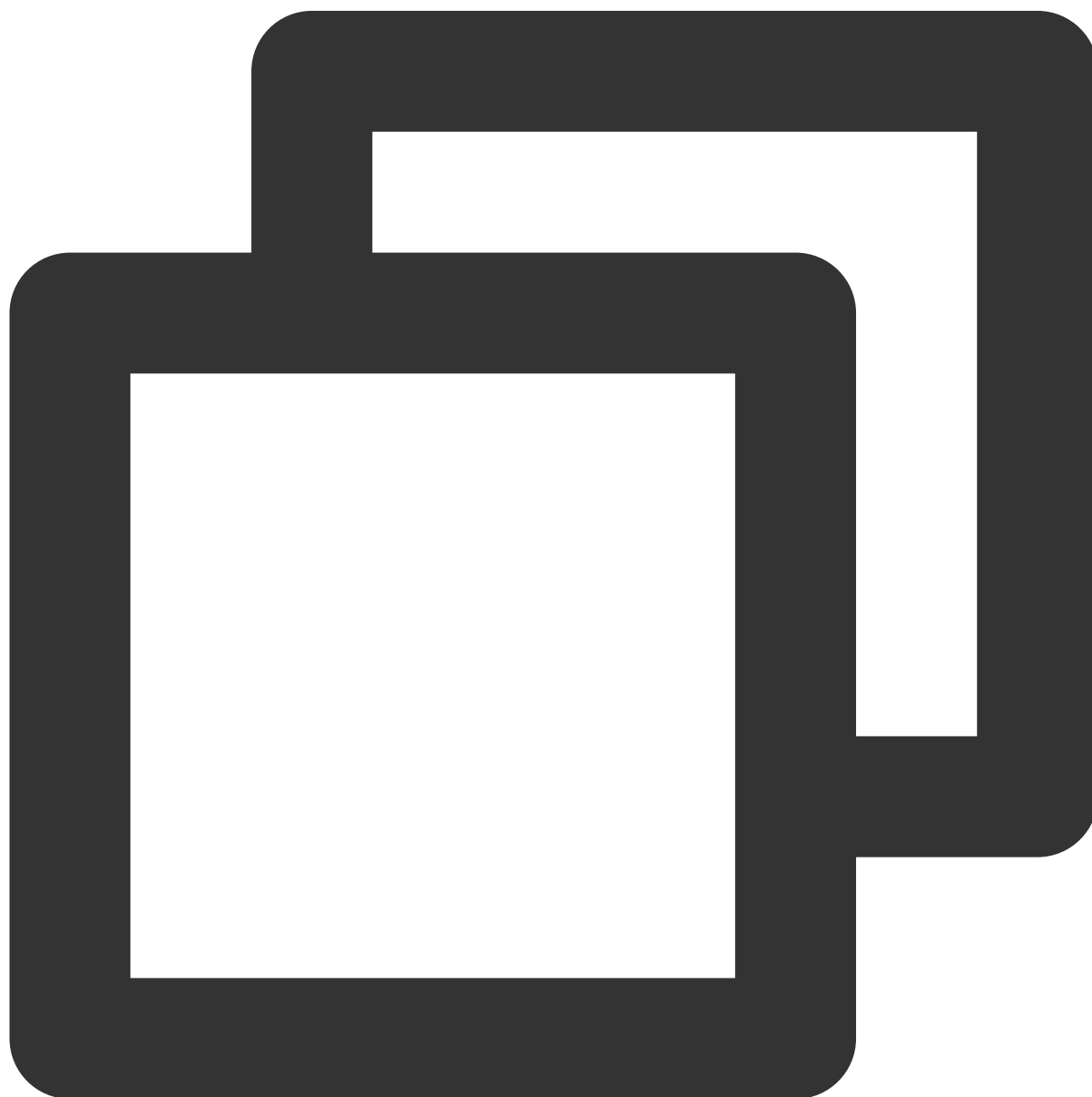


```
void onRemoteUserLeaveRoom(String roomId, TUIRoomDefine.UserInfo userInfo)
```

Parameter	Type	Description
roomId	String	Room ID
userInfo	<a href="#">TUIRoomDefine.UserInfo</a>	User information

## onUserRoleChanged

User role changes event.

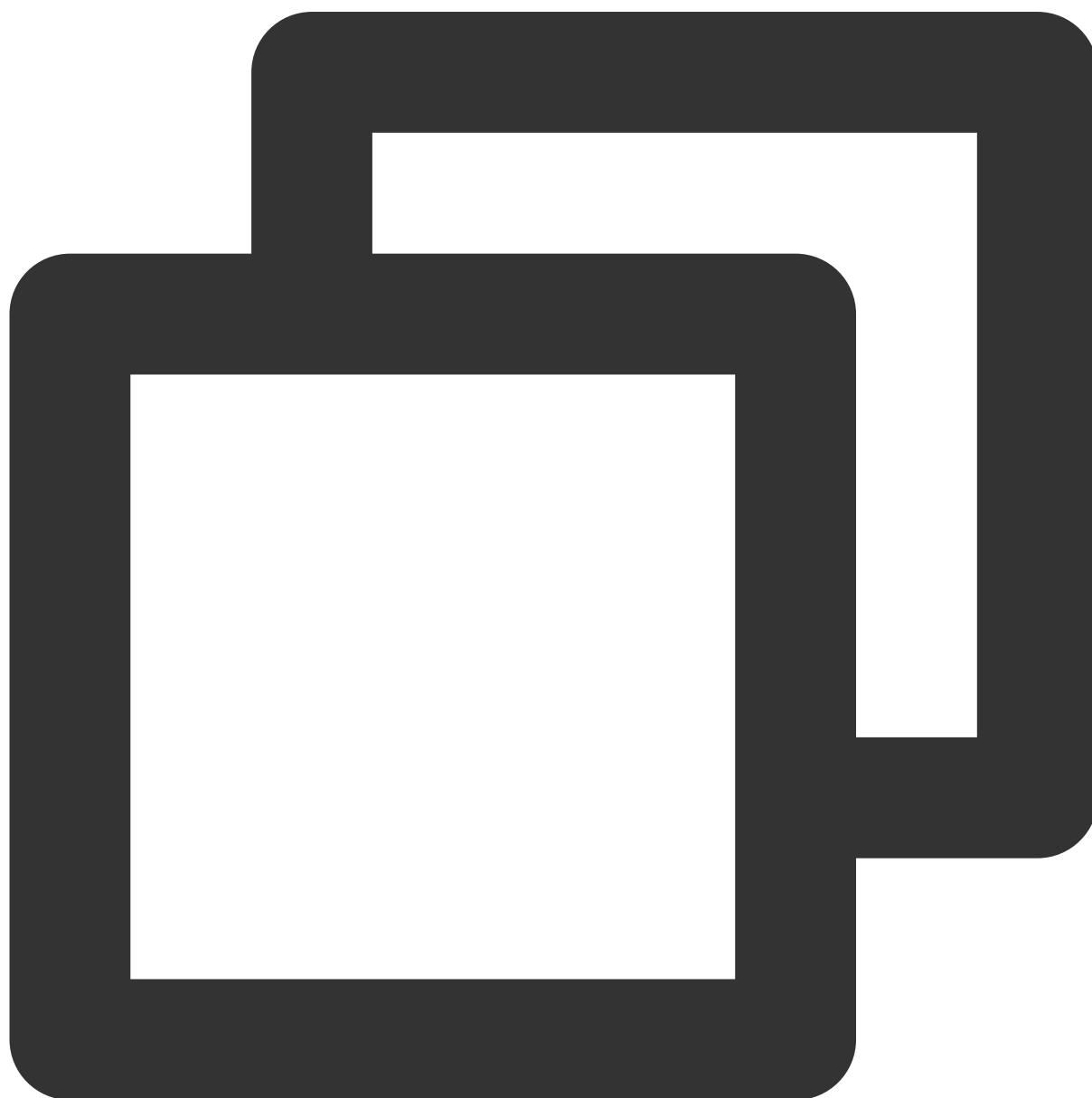


```
void onUserRoleChanged(String userId, TUIRoomDefine.Role role)
```

Parameter	Type	Description
userId	String	User ID
role	<a href="#">TUIRoomDefine.Role</a>	User Role

## onUserVideoStateChanged

User Video status changes event.

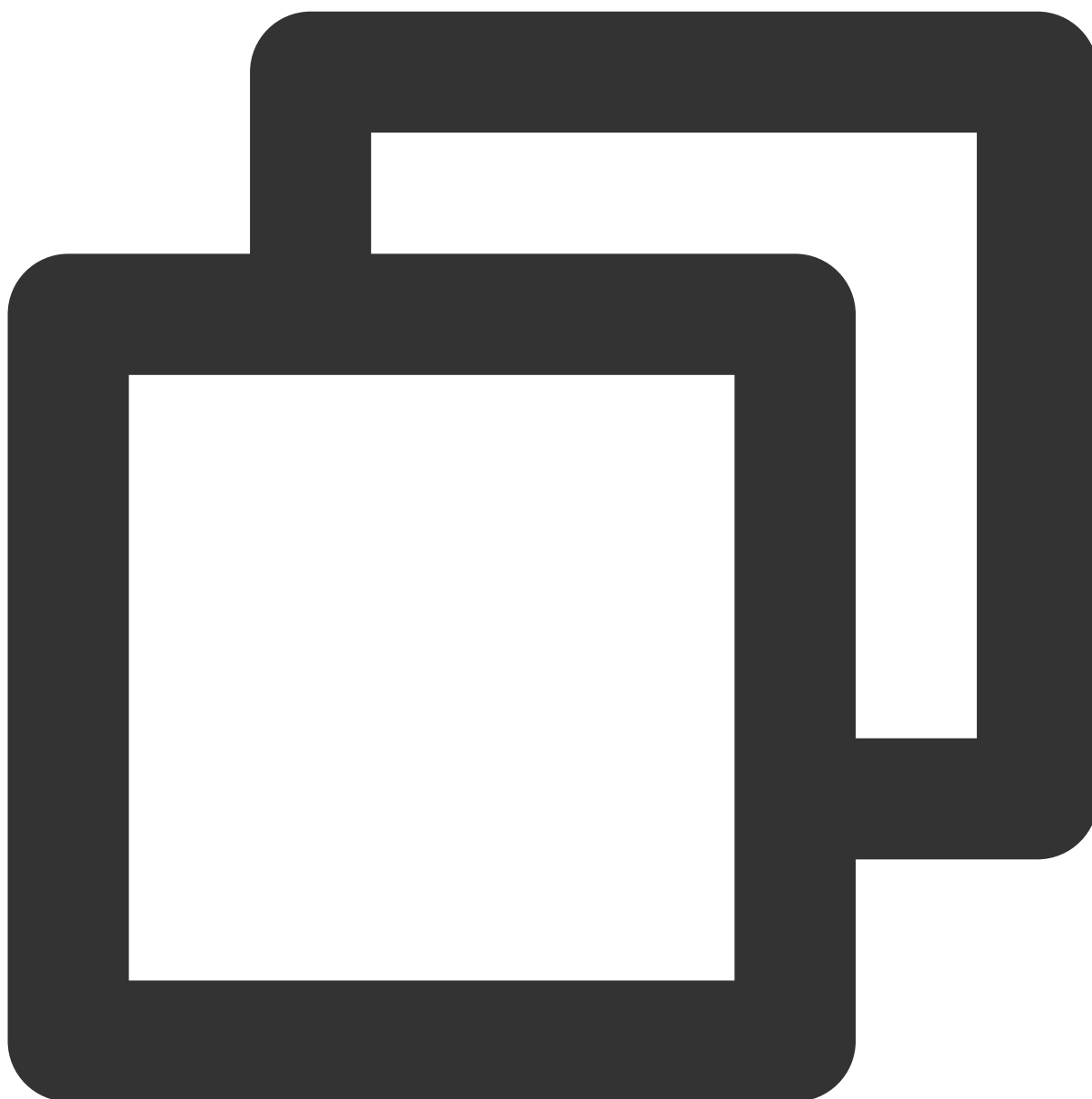


```
void onUserVideoStateChanged(String userId,
                             TUIRoomDefine.VideoStreamType streamType,
                             boolean hasVideo,
                             TUIRoomDefine.ChangeReason reason)
```

Parameter	Type	Description
userId	String	User ID
streamType	<a href="#">TUIRoomDefine.VideoStreamType</a>	Streams type
hasVideo	boolean	Whether there are streams
reason	<a href="#">TUIRoomDefine.ChangeReason</a>	Reason for streams change

## onUserAudioStateChanged

User Audio status changes event.

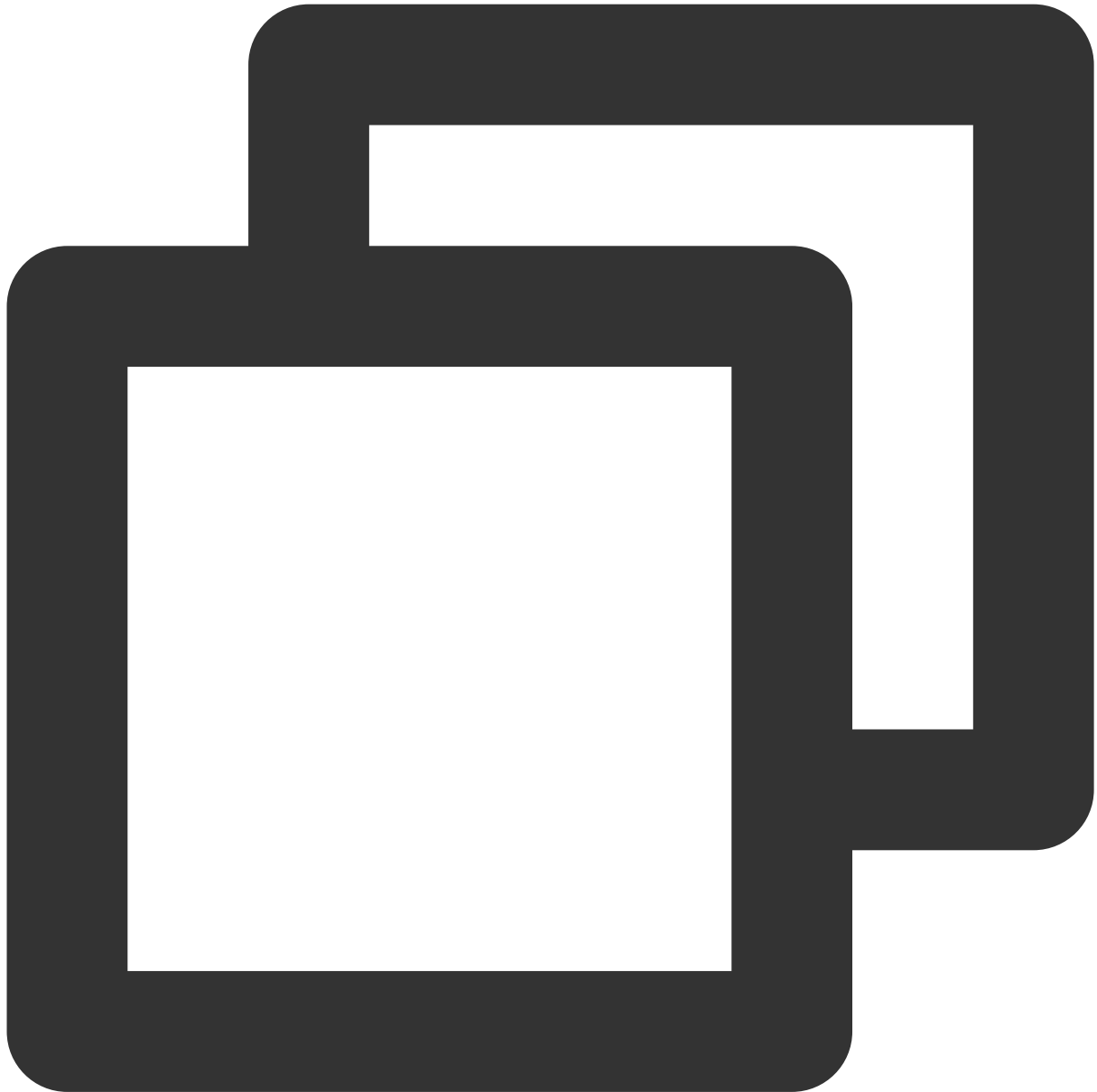


```
void onUserAudioStateChanged(String userId, boolean hasAudio, TUIRoomDefine.ChangeReason reason)
```

Parameter	Type	Description
userId	String	User ID
hasAudio	boolean	Whether there are Audio streams
reason	<a href="#">TUIRoomDefine.ChangeReason</a>	Reason for Audio streams change

## onUserVoiceVolumeChanged

User volume change event.

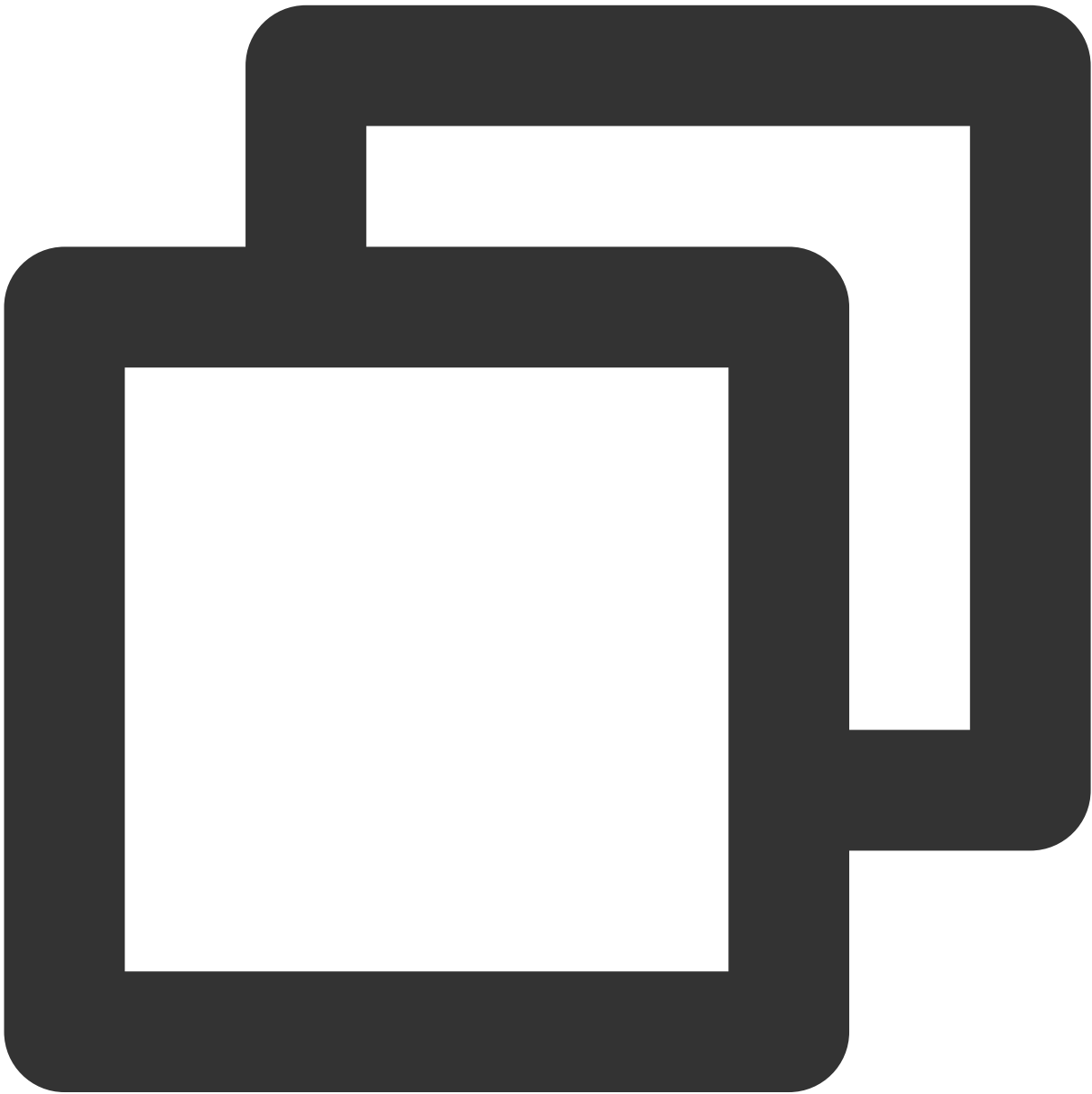


```
void onUserVoiceVolumeChanged(Map<String, Integer> volumeMap)
```

Parameter	Type	Description
volumeMap	Map	User Volume Map key: userId value: Used for carrying the volume size of all speaking users, Value range 0 - 100

onSendMessageForUserDisableChanged

User text message sending ability changes event.



```
void onSendMessageForUserDisableChanged(String roomId, String userId, boolean isDis
```

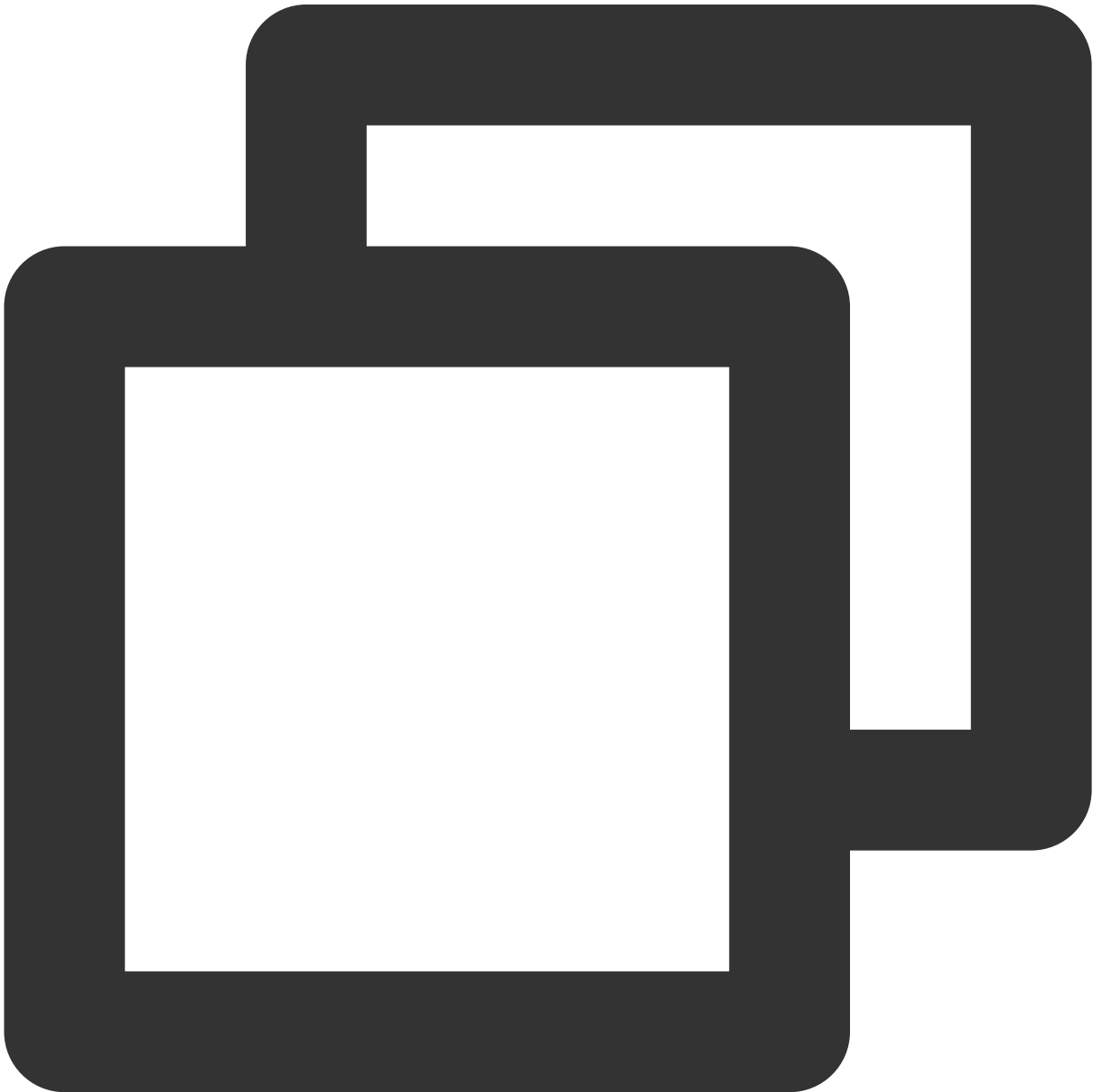
Parameter	Type	Description
roomId	String	Room ID



userId	String	User ID
isDisable	boolean	Whether it is prohibited to send text messages.

onUserNetworkQualityChanged

User network status change event.



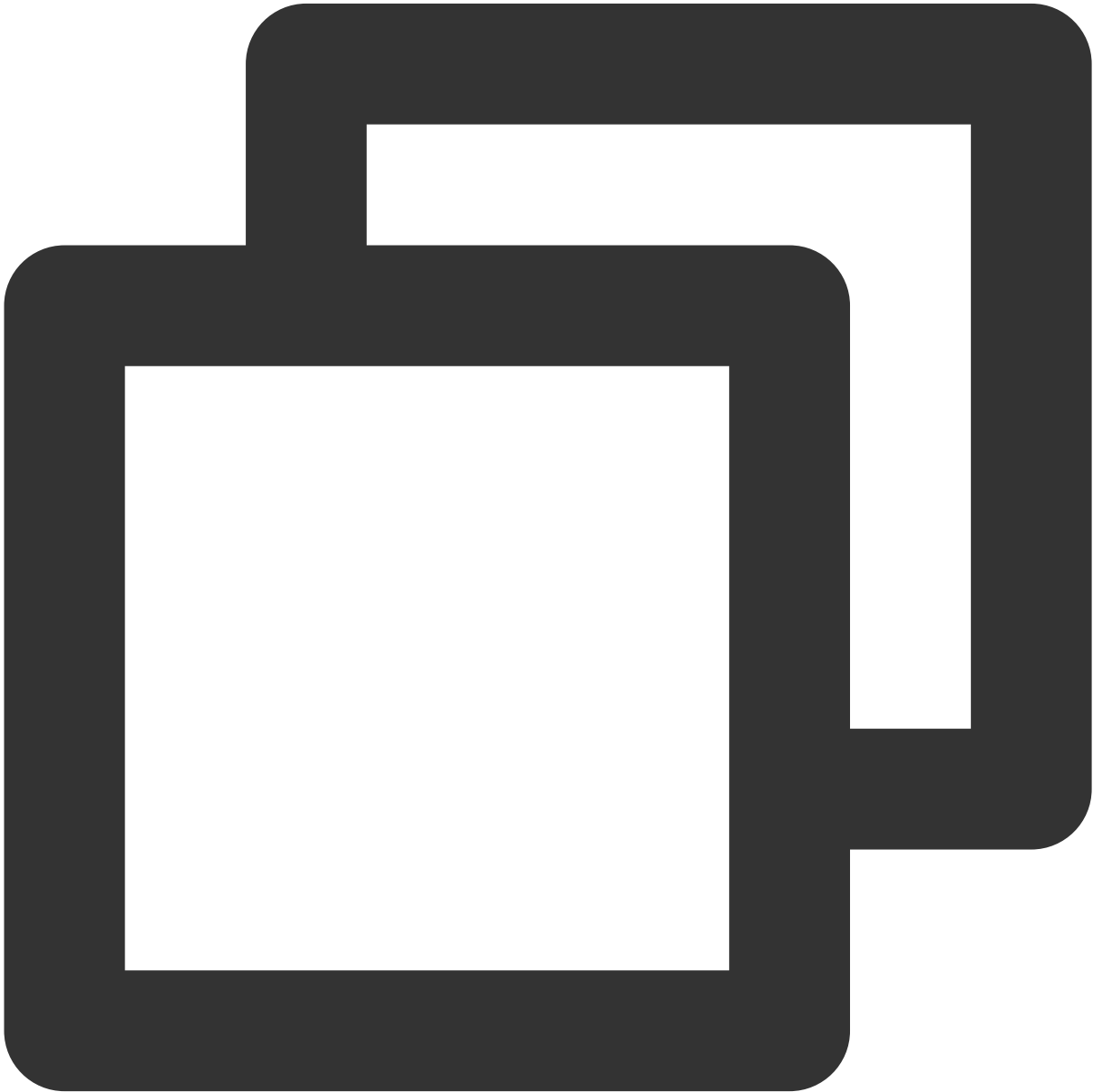
```
void onUserNetworkQualityChanged (Map<String, TUICommonDefine.NetworkInfo> networkMa
```

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

networkMap	Map	User Network Status Map key: userId value: Network Condition
------------	-----	--

**onUserScreenCaptureStopped**

Screen Sharing stopped Callback event.



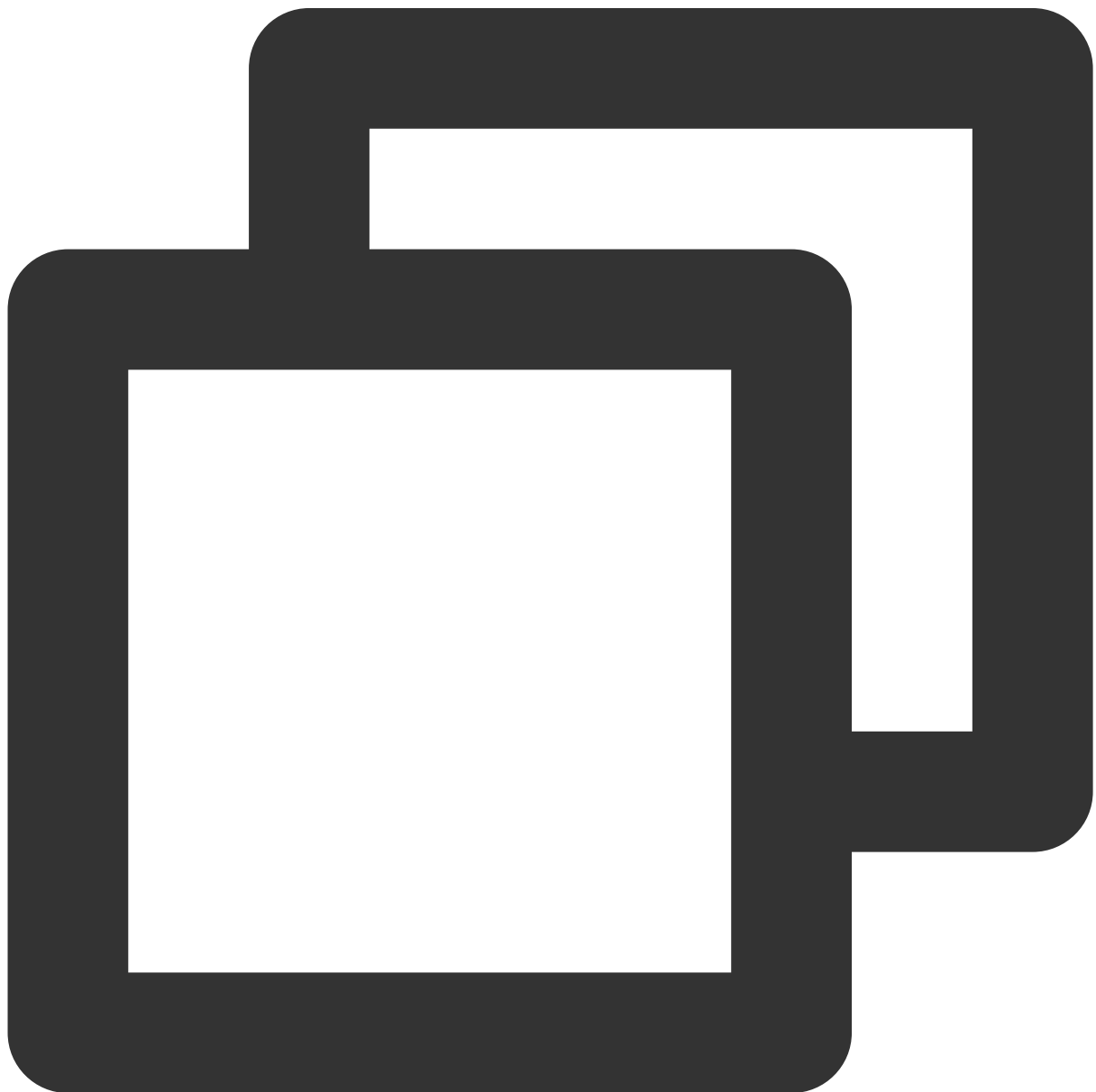
```
void onUserScreenCaptureStopped(int reason)
```

--	--	--

Parameter	Type	Description
reason	int	Stop reason: 0: User actively stops 1: Screen window closing causes the stop 2: Screen Sharing display screen status change (such as interface being unplugged, Projection mode change, etc.)

### **onRoomMaxSeatCountChanged**

Maximum number of mic slots changes event in the room (only effective in meeting type rooms).

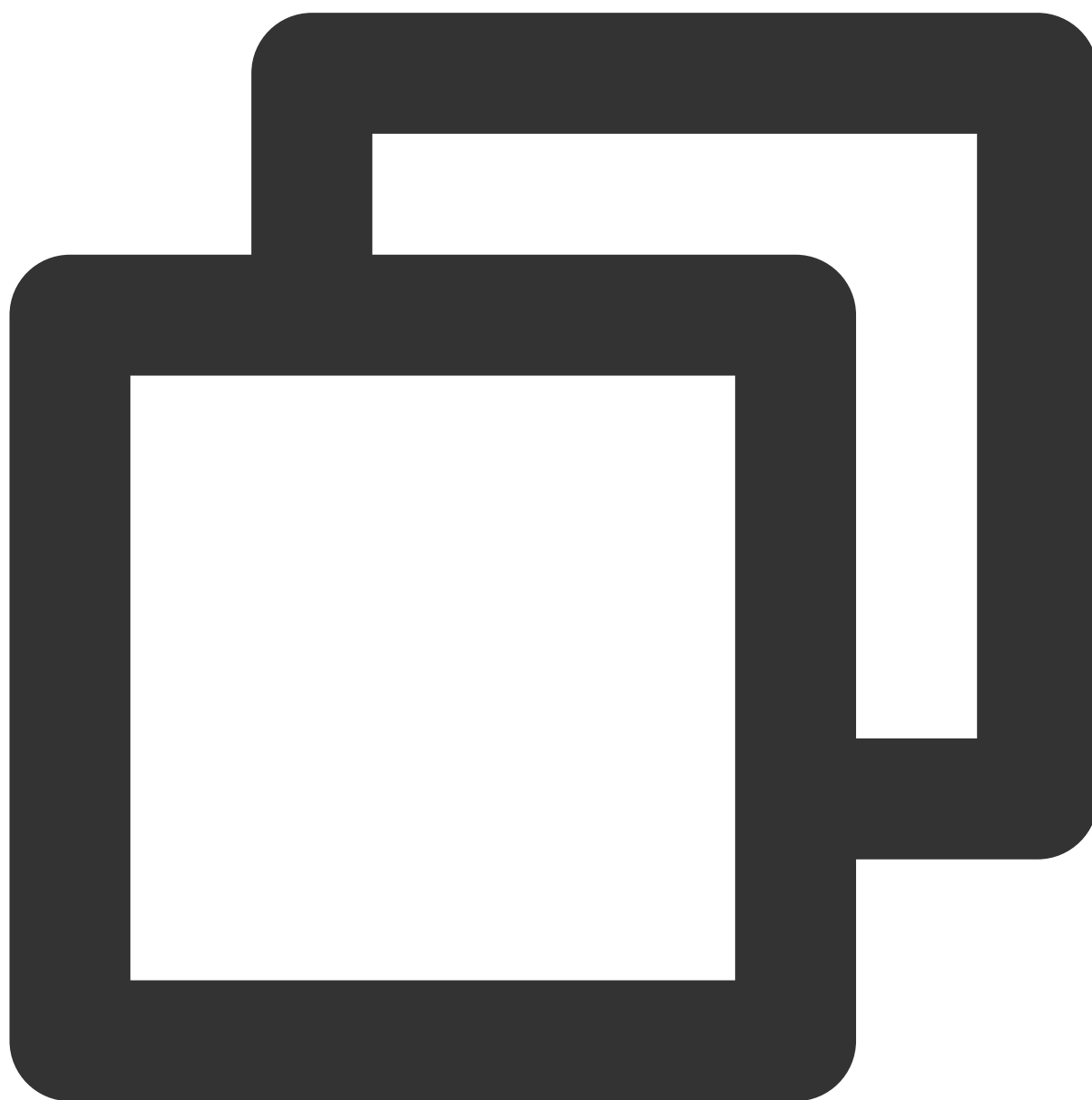


```
void onRoomMaxSeatCountChanged(String roomId, int maxSeatCount)
```

Parameter	Type	Description
roomId	String	Room ID
maxSeatCount	int	Maximum number of mic slots in the room

## onSeatListChanged

Mic slot list changes event.

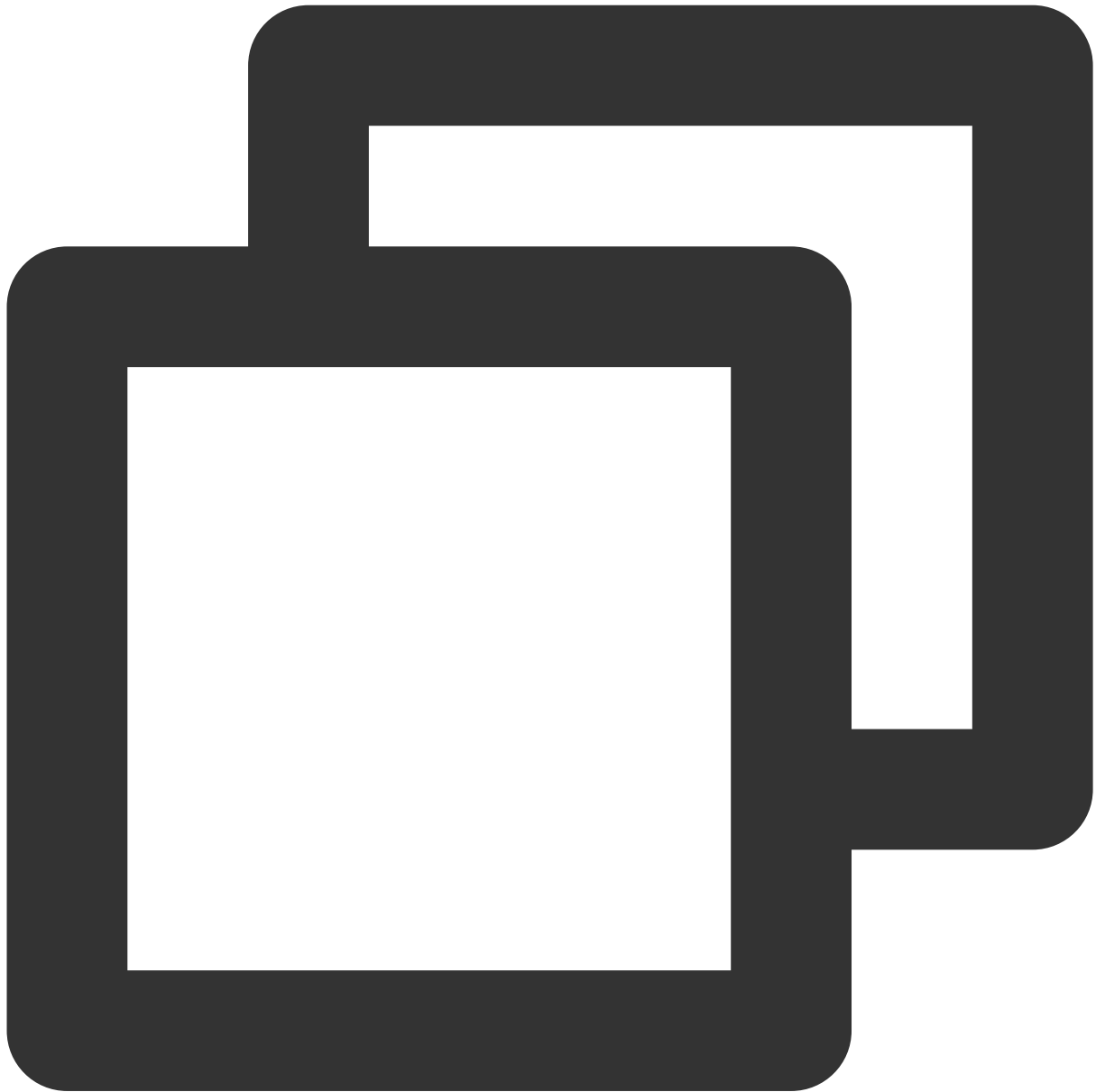


```
void onSeatListChanged(List<TUIRoomDefine.SeatInfo> seatList,  
                      List<TUIRoomDefine.SeatInfo> seatedList,  
                      List<TUIRoomDefine.SeatInfo> leftList)
```

Parameter	Type	Description
seatList	List< <a href="#">TUIRoomDefine.SeatInfo</a> >	The latest user list on the mic, including newly on mic users
seatedList	List< <a href="#">TUIRoomDefine.SeatInfo</a> >	Newly on mic user list
leftList	List< <a href="#">TUIRoomDefine.SeatInfo</a> >	Newly off mic user list

### onKickedOffSeat

Received the event of user being kicked off mic.

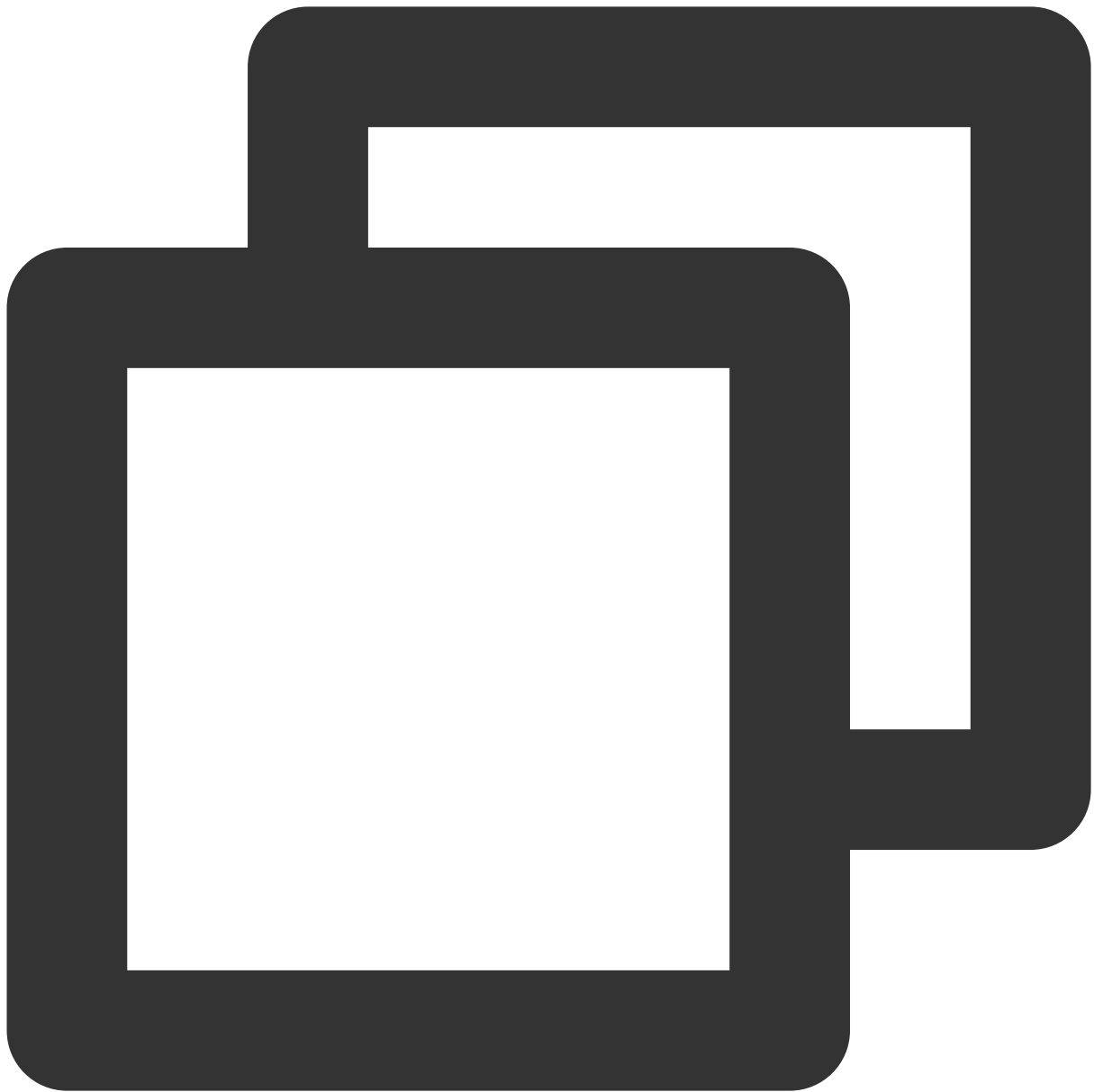


```
void onKickedOffSeat (String userId)
```

Parameter	Type	Description
userId	String	Operate Kick-out of the (Host/Administrator) User ID

## onRequestReceived

Received request message event.



```
void onRequestReceived(TUIRoomDefine.Request request)
```

Parameter	Type	Description
request	<a href="#">TUIRoomDefine.Request</a>	Request content

## onRequestCancelled

Received request cancellation event.



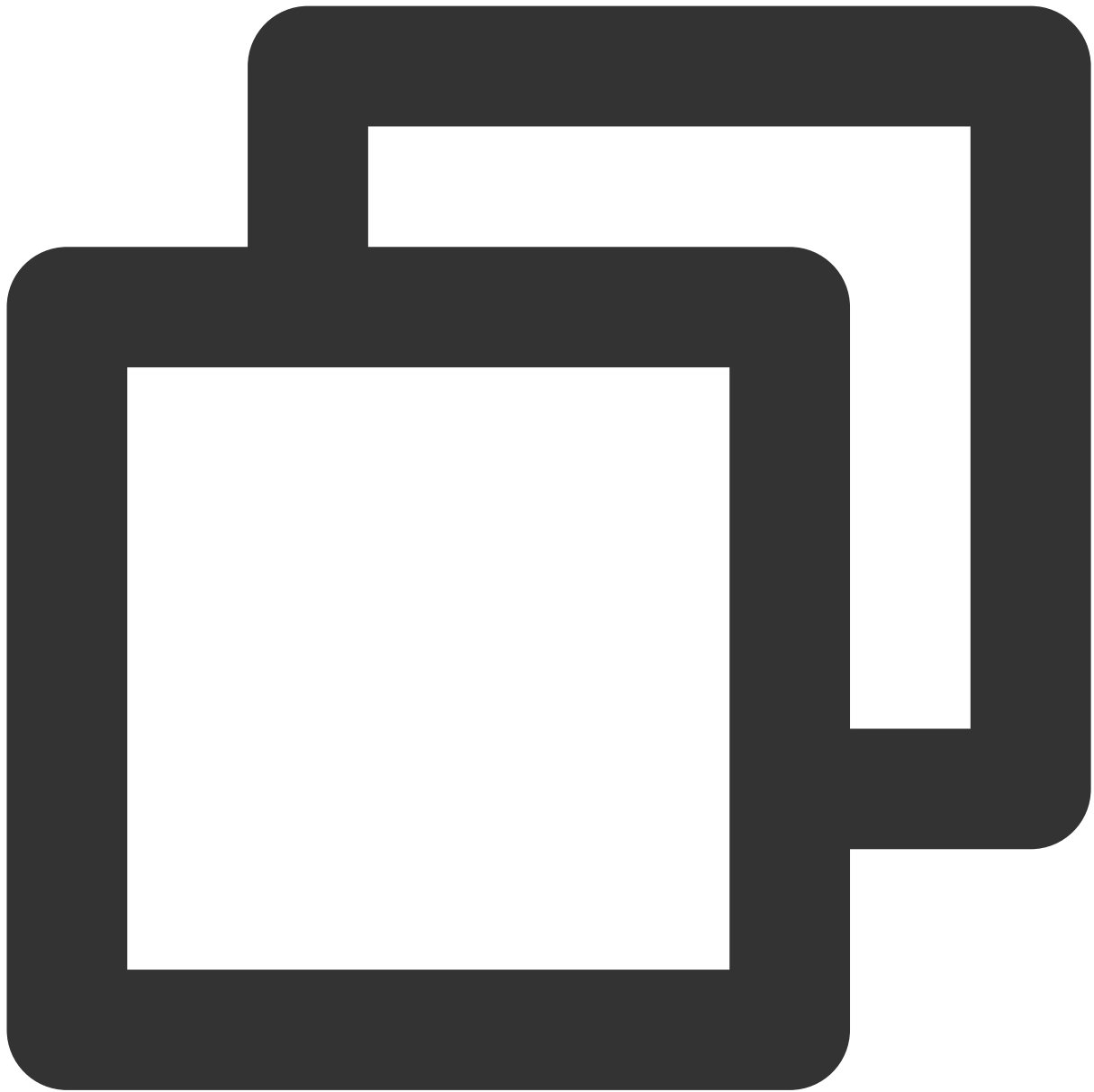
```
void onRequestCancelled(String requestId, String userId)
```

Parameter	Type	Description
requestId	String	Request ID
userId	String	Cancel signaling user ID

## onReceiveTextMessage



Received ordinary text message event.

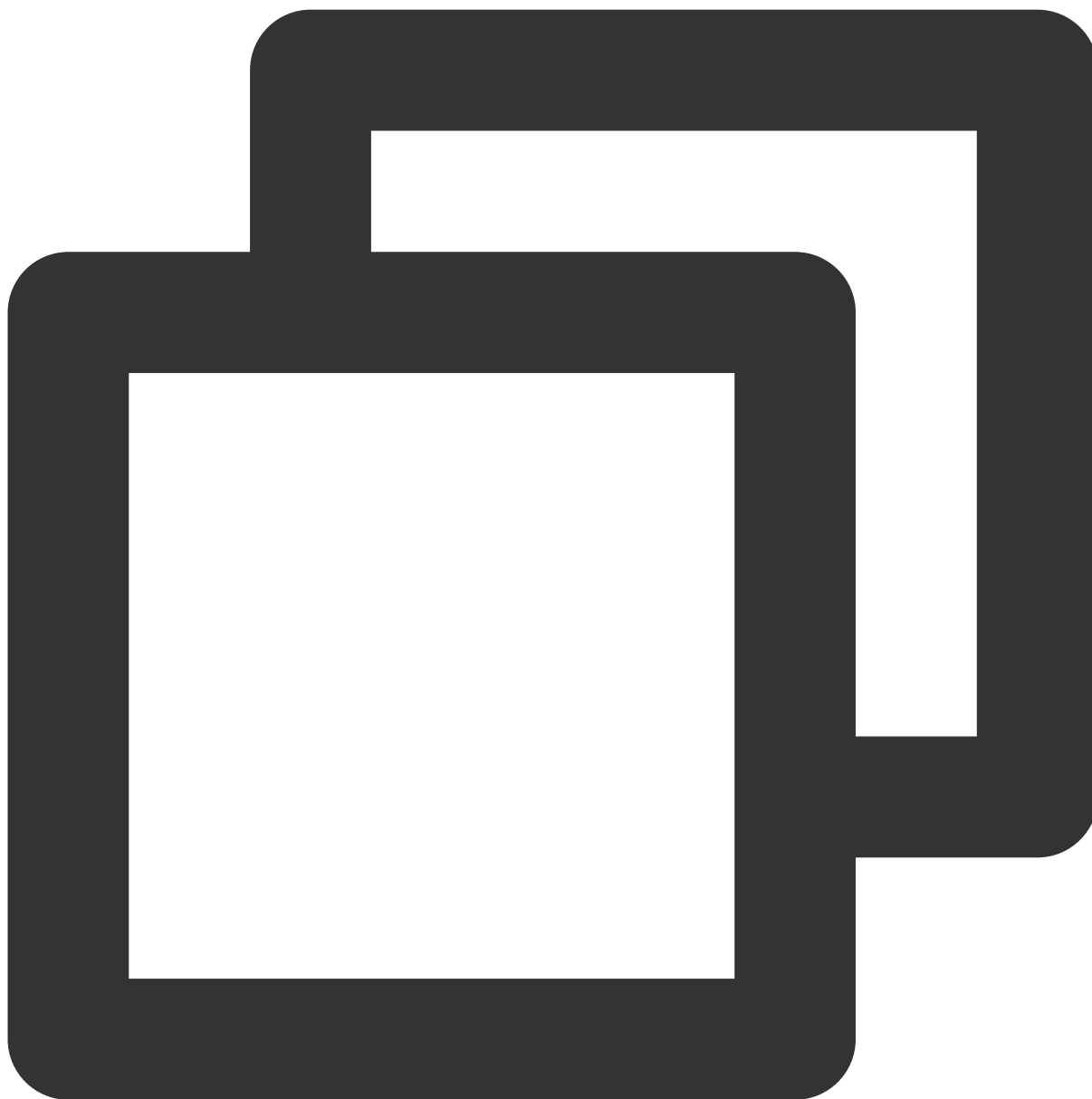


```
void onReceiveTextMessage(String roomId, TUICommonDefine.Message message)
```

Parameter	Type	Description
roomId	String	Room ID
message	<a href="#">TUICommonDefine.Message</a>	Message content

### onReceiveCustomMessage

Received custom message event.



```
void onReceiveCustomMessage(String roomId, TUICommonDefine.Message message)
```

Parameter	Type	Description
roomId	String	Room ID
message	<a href="#">TUICommonDefine.Message</a>	Message content



# Type Definition

Last updated : 2023-10-23 11:30:42

## Enumeration Definition

### TUIRoomDefine

Type	Description
<a href="#">RoomType</a>	Room Type
<a href="#">SpeechMode</a>	Room Mode
<a href="#">MediaDevice</a>	Room Media Device Type
<a href="#">Role</a>	Room Role Type
<a href="#">VideoQuality</a>	Video Quality
<a href="#">AudioQuality</a>	Audio Quality
<a href="#">VideoStreamType</a>	Video Stream Type
<a href="#">ChangeReason</a>	Change Reason (User audio and video status change operation reason: self-modification or modified by room owner/administrator)
<a href="#">RequestAction</a>	Request Type

### TUICommonDefine

Type	Description
<a href="#">NetworkQuality</a>	Network Quality

## Common Structure

### TUIRoomDefine

Type	Description
<a href="#">RoomInfo</a>	Room data
<a href="#">LoginUserInfo</a>	User Login Information
<a href="#">UserInfo</a>	Room User Information

<a href="#">SeatInfo</a>	Room Seat Information
<a href="#">SeatLockParams</a>	Lock Seat Operation Parameters
<a href="#">UserVoiceVolume</a>	Room User Volume
<a href="#">Request</a>	Signaling Request

## TUICommonDefine

Type	Description
<a href="#">NetworkInfo</a>	Network Quality Information
<a href="#">Message</a>	Message

## RoomType

Room Type

Enumeration	Value	Description
CONFERENCE	1	Conference Type Room, suitable for conference and education scenarios, this room can enable free speech, apply for speech, go live and other modes.
LIVE_ROOM	2	Live Type Room, suitable for live broadcast scenarios, this room can enable free speech, mic control mode, and the seats in this room are numbered.

## SpeechMode

Mic Control Mode

Enumeration	Value	Description
FREE_TO_SPEAK	1	Free speech mode.
APPLY_TO_SPEAK	2	Apply to speak mode. (Only effective in conference type room)
SPEAK_AFTER_TAKING_SEAT	3	Go Live mode.

## MediaDevice

Room Media Device Type

--	--	--

Enumeration	Value	Description
MICROPHONE	1	Mic
CAMERA	2	Camera
SCREEN_SHARING	3	Screen Sharing

## Role

### Room Role Types

Enumeration	Value	Description
ROOM_OWNER	0	Room Owner, generally refers to the creator of the room, the highest authority holder in the room
MANAGER	1	Room Administrator
GENERAL_USER	2	General Member in the room

## VideoQuality

### Video Quality

Enumeration	Value	Description
Q_360P	1	Low-quality 360P
Q_540P	2	Standard Definition 540P
Q_720P	3	High Definition 720P
Q_1080P	4	Ultra-clear 1080P

## AudioQuality

### Audio Quality

Enumeration	Value	Description
SPEECH	0	Speech Mode
DEFAULT	1	Default Mode
MUSIC	2	Music Mode

## VideoStreamType

Video Stream Type

Enumeration	Value	Description
CAMERA_STREAM	0	High-quality Camera Video Stream
SCREEN_STREAM	1	Screen Sharing Video Stream
CAMERA_STREAM_LOW	2	Low-quality Camera Video Stream

## ChangeReason

Change Reason (User audio and video status change operation reason: self-modification or modification by room owner/administrator)

Enumeration	Value	Description
BY_SELF	0	Self-operation
BY_ADMIN	1	Room Owner or Administrator Operation

## RequestAction

Request Type

Enumeration	Value	Description
INVALID_ACTION	0	Invalid Request
REQUEST_TO_OPEN_REMOTE_CAMERA	1	Request Remote User to Open Camera
REQUEST_TO_OPEN_REMOTE_MICROPHONE	2	Request Remote User to Open Microphone
REQUEST_TO_CONNECT_OTHER_ROOM	3	Request to Connect to Other Room
REQUEST_TO_TAKE_SEAT	4	Request to Go Live
REQUEST_REMOTE_USER_ON_SEAT	5	Request Remote User to Go Live
REQUEST_APPLY_TO_ADMIN_TO_OPEN_LOCAL_CAMERA	6	Request to Admin to Open Local Camera
REQUEST_APPLY_TO_ADMIN_TO_OPEN_LOCAL_MICROPHONE	7	Request to Admin to

[Open Local Microphone](#)

## NetworkQuality

### Network Quality

Enumeration	Value	Description
UNKNOWN	0	Undefined
EXCELLENT	1	Current Network is Excellent
GOOD	2	Current Network is Good
POOR	3	Current Network is Average
BAD	4	Current Network is Poor
VERY_BAD	5	Current Network is Very Poor
DOWN	6	Current Network Does Not Meet TRTC's Minimum Requirements

## RoomInfo

### Room Information

Field	Type	Description
roomId	String	Room ID
ownerId	String	Host ID, default is the room creator (read-only)
roomType	<a href="#">RoomType</a>	Room Type
name	String	Room Name, default is the room ID
speechMode	<a href="#">SpeechMode</a>	Mic Control Mode
isCameraDisableForAllUser	boolean	Whether to Disable Opening Camera (optional when creating a room), default value: false
isMicrophoneDisableForAllUser	boolean	Whether to Disable Opening Microphone (optional when creating a room), default value: false
isMessageDisableForAllUser	boolean	Whether to Disable Sending Messages (optional when creating a room), default value: false
maxSeatCount	int	Maximum Number of Mic Seats (only supported when



		entering the room and creating the room)
enableCDNStreaming	boolean	Whether to Enable CDN Live Streaming (optional when creating a room, for live streaming rooms), default value: false
cdnStreamDomain	String	Live Streaming Push Domain (optional when creating a room, for live streaming rooms), default value: empty
createTime	long	Room Creation Time (read-only)
memberCount	int	Number of Members in the Room (read-only)

## LoginUserInfo

User Login Information

Field	Type	Description
userId	String	User ID
userName	String	User Name
avatarUrl	String	User Avatar URL

## UserInfo

User Information in the Room

Field	Type	Description
userId	String	User ID
userName	String	User Name
avatarUrl	String	User Avatar URL
userRole	<a href="#">Role</a>	User Role Type
hasAudioStream	boolean	Whether There is Audio Stream, default value: false
hasVideoStream	boolean	Whether There is Video Stream, default value: false
hasScreenStream	boolean	Whether There is Screen Sharing Stream, default value: false

## SeatInfo

## Seat Information in the Room

Field	Type	Description
index	int	Mic Seat Number
userId	String	User ID
isLocked	boolean	Whether the Mic Seat is Locked, default false
isVideoLocked	boolean	Whether the Mic Seat is Prohibited from Opening Camera, default false
isAudioLocked	boolean	Whether the Mic Seat is Prohibited from Opening Microphone, default false

**SeatLockParams**

## Lock Seat Operation Parameters

Field	Type	Description
lockSeat	boolean	Lock Mic Seat, default false
lockVideo	boolean	Lock Mic Seat Camera, default false
lockAudio	boolean	Lock Mic Seat Microphone, default false

**UserVoiceVolume**

## User Voice Volume in the Room

Field	Type	Description
userId	String	User ID
volume	int	Volume Size, Value range 0 - 100

**Request**

## Signaling Request

Field	Type	Description
requestId	String	Request ID

requestAction	<a href="#">RequestAction</a>	Request Type
userId	String	User ID
content	String	Signaling Content
timestamp	int	Timestamp

## NetworkInfo

### Network Quality Information

Field	Type	Description
userId	String	User ID
quality	<a href="#">NetworkQuality</a>	Network Quality
upLoss	int	Upstream Packet Loss Rate
downLoss	int	Downstream Packet Loss Rate
delay	int	Network Delay

## Message

### Message

Field	Type	Description
messageId	String	Message ID
message	String	Message Text
timestamp	long	Message Time
userId	String	Message Sender
userName	String	Message Sender Nickname
avatarUrl	String	Message Sender Avatar

# Windows

## API Overview

Last updated : 2023-10-30 11:01:59

### TUIRoomEngine API List

TUIRoomEngine API is the UI-free interface of the Conference Component, which allows you to customize the encapsulation according to your business needs.

TUIRoomEngine

#### TUIRoomEngine Core Methods

API	Description
<a href="#">createTUIRoomEngine</a>	Create Instance of TUIRoomEngine
<a href="#">destroyTUIRoomEngine</a>	Destroy TUIRoomEngine Instance
<a href="#">login</a>	Login interface, you need to initialize user information before entering the room and perform a series of operations.
<a href="#">logout</a>	Logout interface, there will be actively leave room operation, destroy resources
<a href="#">setSelfInfo</a>	Set local user name and avatar
<a href="#">getSelfInfo</a>	Get local user basic information
<a href="#">addObserver</a>	Set event callback
<a href="#">removeObserver</a>	Remove event callback

#### Room Related Active Interface

API	Description
<a href="#">createRoom</a>	Create room
<a href="#">destroyRoom</a>	Close the room
<a href="#">enterRoom</a>	Entered room

<a href="#">exitRoom</a>	Leave room
<a href="#">connectOtherRoom</a>	Connect to other room
<a href="#">disConnectOtherRoom</a>	Disconnect from other room
<a href="#">fetchRoomInfo</a>	Get room data
<a href="#">updateRoomNameByAdmin</a>	Update room name(only administrator or group owner can call)
<a href="#">updateRoomSpeechModeByAdmin</a>	Set room management mode (only administrator or group owner can call)

## Local User View Rendering, Video Management

API	Description
<a href="#">setLocalVideoView</a>	Set the view control for local user video rendering
<a href="#">openLocalCamera</a>	Open local camera
<a href="#">closeLocalCamera</a>	Close local camera
<a href="#">updateVideoQuality</a>	Update local video codec quality settings
<a href="#">startScreenSharing</a>	Start screen sharing
<a href="#">stopScreenSharing</a>	End screen sharing
<a href="#">getScreenSharingTargetList</a>	Enumerate shareable screens and windows
<a href="#">selectScreenSharingTarget</a>	Select the screen or window to share
<a href="#">startPushLocalVideo</a>	Start pushing local video
<a href="#">stopPushLocalVideo</a>	Stop pushing local video

## Local User Audio Management

API	Description
<a href="#">openLocalMicrophone</a>	Open local microphone
<a href="#">closeLocalMicrophone</a>	Close local microphone
<a href="#">updateAudioQuality</a>	Update local audio codec quality settings

<a href="#">startPushLocalAudio</a>	Start pushing local audio
<a href="#">stopPushLocalAudio</a>	Stop pushing local audio

## Remote User View Rendering, Video Management

API	Description
<a href="#">setRemoteVideoView</a>	Set the view control for remote user video rendering
<a href="#">startPlayRemoteVideo</a>	Start playing remote user video
<a href="#">stopPlayRemoteVideo</a>	Stop playing remote user video
<a href="#">muteRemoteAudioStream</a>	Mute remote user

## Room User Information

API	Description
<a href="#">getUserList</a>	Get the member list in the room
<a href="#">getUserInfo</a>	Get member information

## Room User Management

API	Description
<a href="#">changeUserRole</a>	Modify user role (only administrator or group owner can call)
<a href="#">kickRemoteUserOutOfRoom</a>	Kick Remote User out of the Room (Only Administrator or Group Owner can call)

## Speech Management in Room

API	Description
<a href="#">disableDeviceForAllUserByAdmin</a>	Control the permission status of whether all users in the current room can open Audio and Video streams capturing devices, such as: Prohibit All from turning on the mic, Prohibit All from turning on the Camera, Prohibit All from turning on Screen Sharing (currently only available in Meeting Scenario, and only Administrator or Group Owner can invoke).

<a href="#">openRemoteDeviceByAdmin</a>	Request Remote User to Open Media Device (Only Administrator or Group Owner can call)
<a href="#">closeRemoteDeviceByAdmin</a>	Close Remote User's Media Device (Only Administrator or Group Owner can call)
<a href="#">applyToAdminToOpenLocalDevice</a>	Request to Open Local Media Device (Available for Ordinary Users)

## Microphone Seat Management in Room

API	Description
<a href="#">setMaxSeatCount</a>	Set Maximum Number of Microphone Seats (Only supported when entering the room and creating the room)
<a href="#">getSeatList</a>	Get Microphone Seat List
<a href="#">lockSeatByAdmin</a>	Lock Microphone Seat (Including Position Lock, Audio State Lock, Video State Lock)
<a href="#">takeSeat</a>	Apply to Go Live (Not Required in Free Speech Mode)
<a href="#">leaveSeat</a>	Apply to leave Live (Not Required in Free Speech Mode)
<a href="#">takeUserOnSeatByAdmin</a>	Host/Administrator invites user to go live
<a href="#">kickUserOffSeatByAdmin</a>	Host/Administrator kicks user off the microphone seat

## Send Message

API	Description
<a href="#">sendTextMessage</a>	Send Text Message
<a href="#">sendCustomMessage</a>	Send Custom Message
<a href="#">disableSendingMessageByAdmin</a>	Disable Remote User's Text Message Sending Ability (Only Administrator or Group Owner can call)
<a href="#">disableSendingMessageForAllUser</a>	Disable All Users' Text Message Sending Ability (Only Administrator or Group Owner can call) Advanced Feature: Get TRTC Instance

## Signaling Management

--	--

API	Description
<a href="#">cancelRequest</a>	Cancel Request
<a href="#">responseRemoteRequest</a>	Reply to Request

### Advanced Feature: Get TRTC Instance

API	Description
<a href="#">getTRTCCloud</a>	Get TRTC Instance Object
<a href="#">getDeviceManager</a>	Get Device Management Object
<a href="#">getAudioEffectManager</a>	Get Audio Effect Management Object

## TTUIRoomObserver Callback Event

TUIRoomObserver is the Callback Event class corresponding to TUIRoomEngine. You can monitor the Callback Events you need through this Callback.

TUIRoomObserver

## TUIRoomObserver

### Error Callback

API	Description
<a href="#">onError</a>	Error Callback Event

### Login Status Event Callback

API	Description
<a href="#">onKickedOffLine</a>	User Kicked Offline Event
<a href="#">onUserSigExpired</a>	User Credential Timeout Event

### Room Event Callback



API	Description
<a href="#">onRoomNameChanged</a>	Room Name Change Event
<a href="#">onAllUserMicrophoneDisableChanged</a>	All Users' Microphones Disabled in Room Event
<a href="#">onAllUserCameraDisableChanged</a>	All Users' Cameras Disabled in Room Event
<a href="#">onSendMessageForAllUserDisableChanged</a>	All Users' Text Message Sending Disabled in Room Event
<a href="#">onKickedOutOfRoom</a>	Kicked Out of Room Event
<a href="#">onRoomDismissed</a>	Room Dismissed Event
<a href="#">onRoomSpeechModeChanged</a>	Room Microphone Control Mode Change

### Room User Event Callback

API	Description
<a href="#">onRemoteUserEnterRoom</a>	Remote User Entering Room Event
<a href="#">onRemoteUserLeaveRoom</a>	Remote User Leaving Room Event
<a href="#">onUserRoleChanged</a>	User Role Change Event
<a href="#">onUserVideoStateChanged</a>	User Video State Change Event
<a href="#">onUserAudioStateChanged</a>	User Audio State Change Event
<a href="#">onUserVoiceVolumeChanged</a>	User Volume Change Event
<a href="#">onSendMessageForUserDisableChanged</a>	User Text Message Sending Ability Change Event
<a href="#">onUserNetworkQualityChanged</a>	User Network Status Change Event
<a href="#">onUserScreenCaptureStopped</a>	Screen Sharing End Event

### Room Microphone Seat Event Callback

API	Description
<a href="#">onRoomMaxSeatCountChanged</a>	Room Maximum Microphone Seat Number Change Event (Only effective in conference type rooms)
<a href="#">onSeatListChanged</a>	Microphone Seat List Change Event

[onKickedOffSeat](#)

Received User Kicked Off Microphone Event

**Request Signaling Event Callback**

API	Description
<a href="#">onRequestReceived</a>	Received Request Message Event
<a href="#">onRequestCancelled</a>	Received Request Cancellation Event

**Room Message Event Callback**

API	Description
<a href="#">onReceiveTextMessage</a>	Received Normal Text Message Event
<a href="#">onReceiveCustomMessage</a>	Received Custom Message Event

# TUIRoomEngine

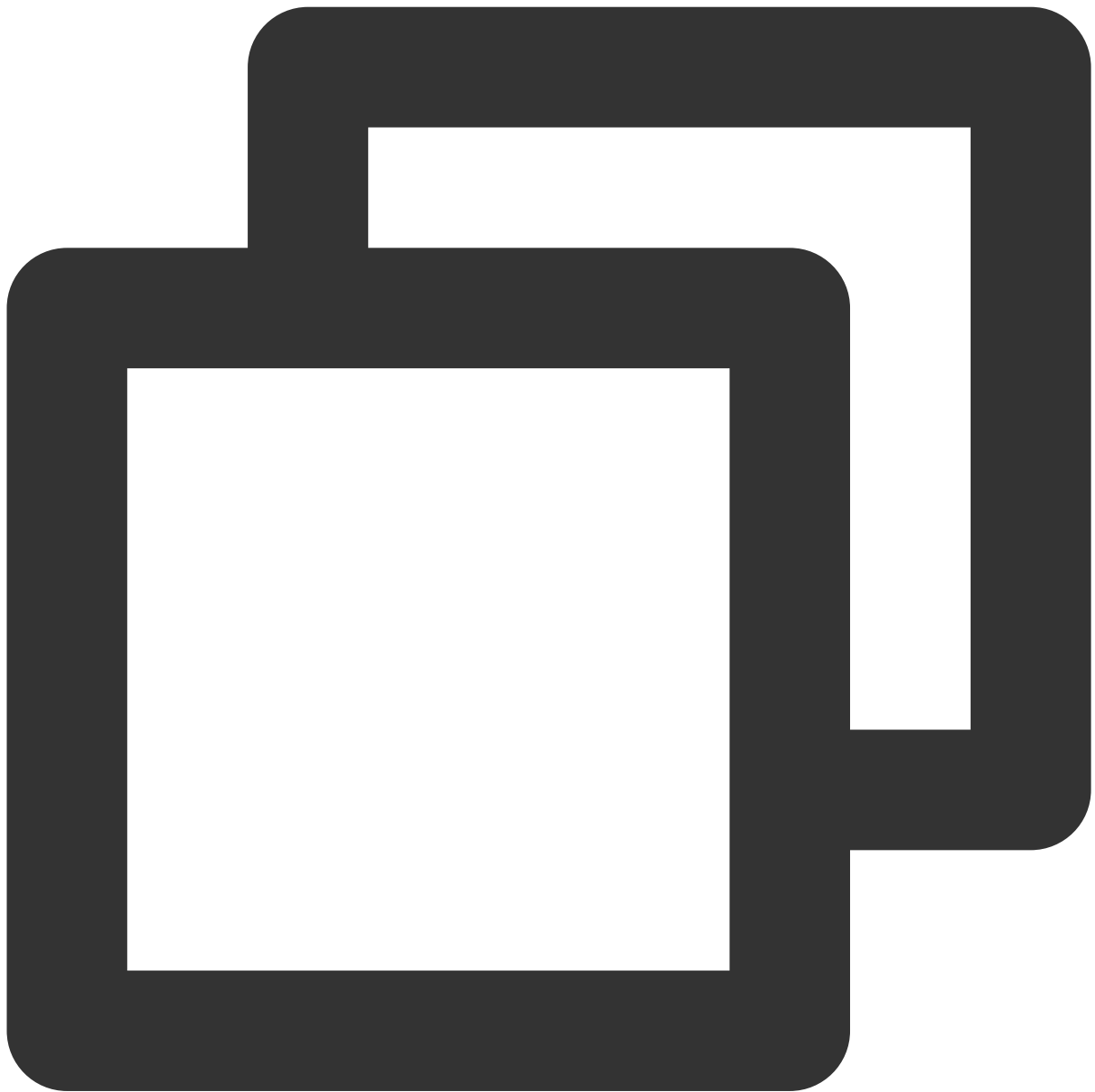
Last updated : 2023-10-30 10:57:09

## TUIRoomEngine API Introduction

TUIRoomEngine API is a No UI Interface for Conference Component, you can use this API to custom encapsulate according to your business needs.

### **createTUIRoomEngine**

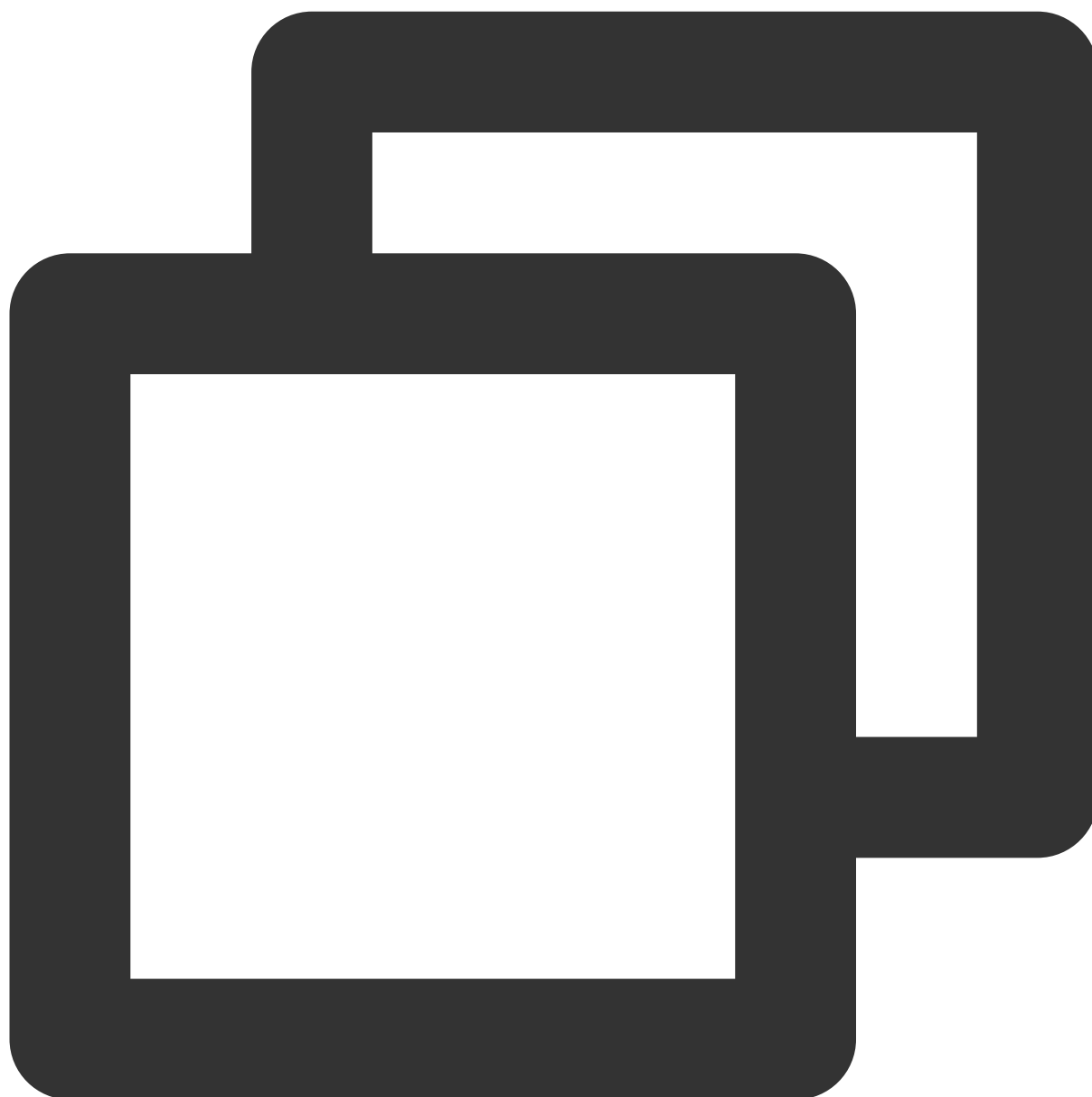
Create TUIRoomEngine Instance



```
tuikit::TUIRoomEngine* createTUIRoomEngine();
```

## **destroyTUIRoomEngine**

Destroy TUIRoomEngine Instance

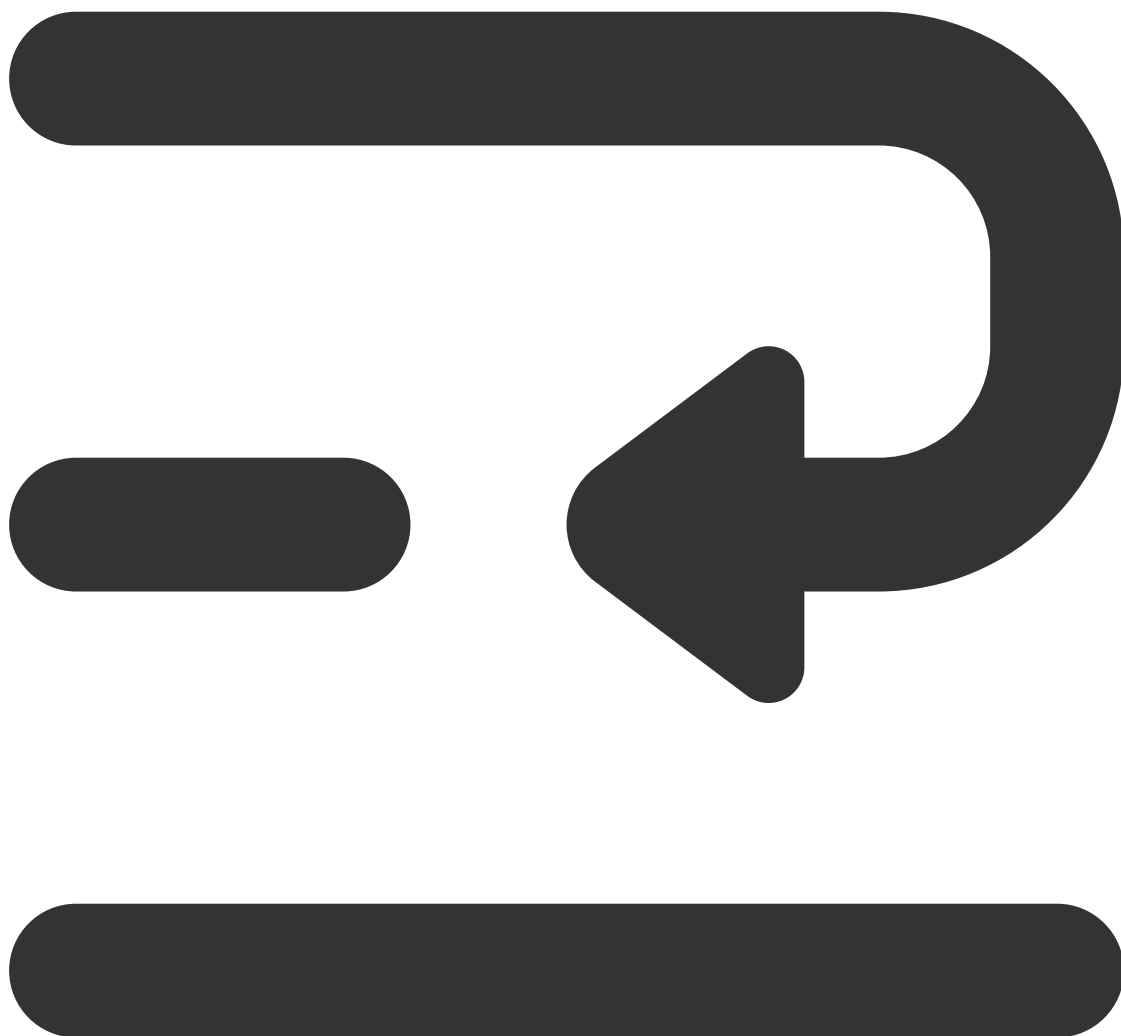


```
void destroyTUIRoomEngine(tuikit::TUIRoomEngine* roomEngine);
```

Parameter	Type	Meaning
roomEngine	tuikit::TUIRoomEngine*	TUIRoomEngine Instance Pointer, this pointer can only be obtained through the CreateTUIRoomEngine Interface.

## login

Login interface, you need to initialize user information before entering the room and perform a series of operations.





```
static void login(int sdkAppId, const char* userId, const char* userSig, TUICallbac
```

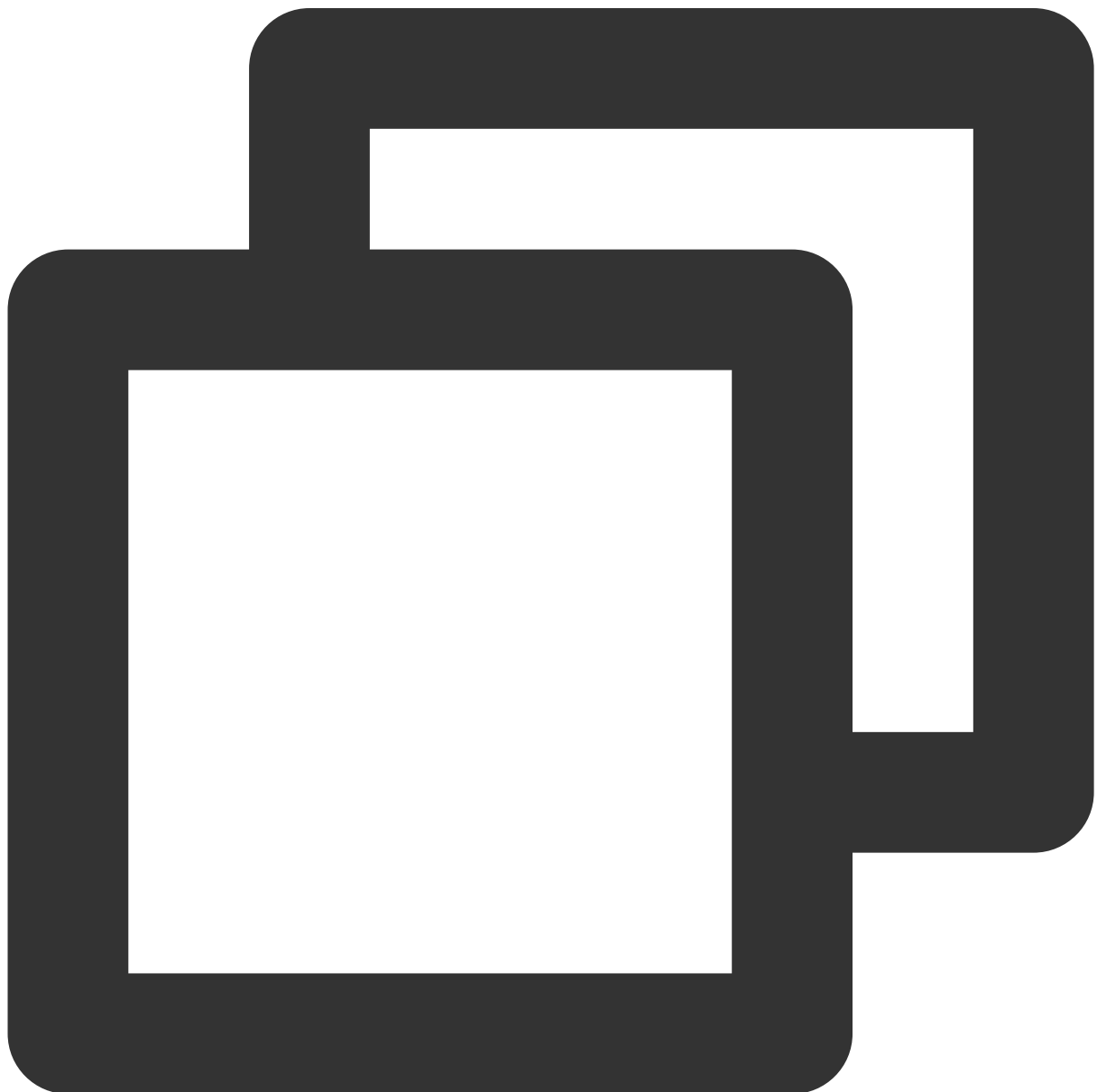
**The parameters are as follows:**

Parameter	Type	Meaning
sdkAppId	int	SDKAppID of Tencent Cloud communication application
userId	const char*	User ID for Differentiate different users

userSig	const char*	UserSig for Tencent Cloud flow authentication
callback	TUICallback*	API Callback for notifying the success or failure of the interface call

## logout

Logout interface, there will be actively leave the room operation, destroy resources.

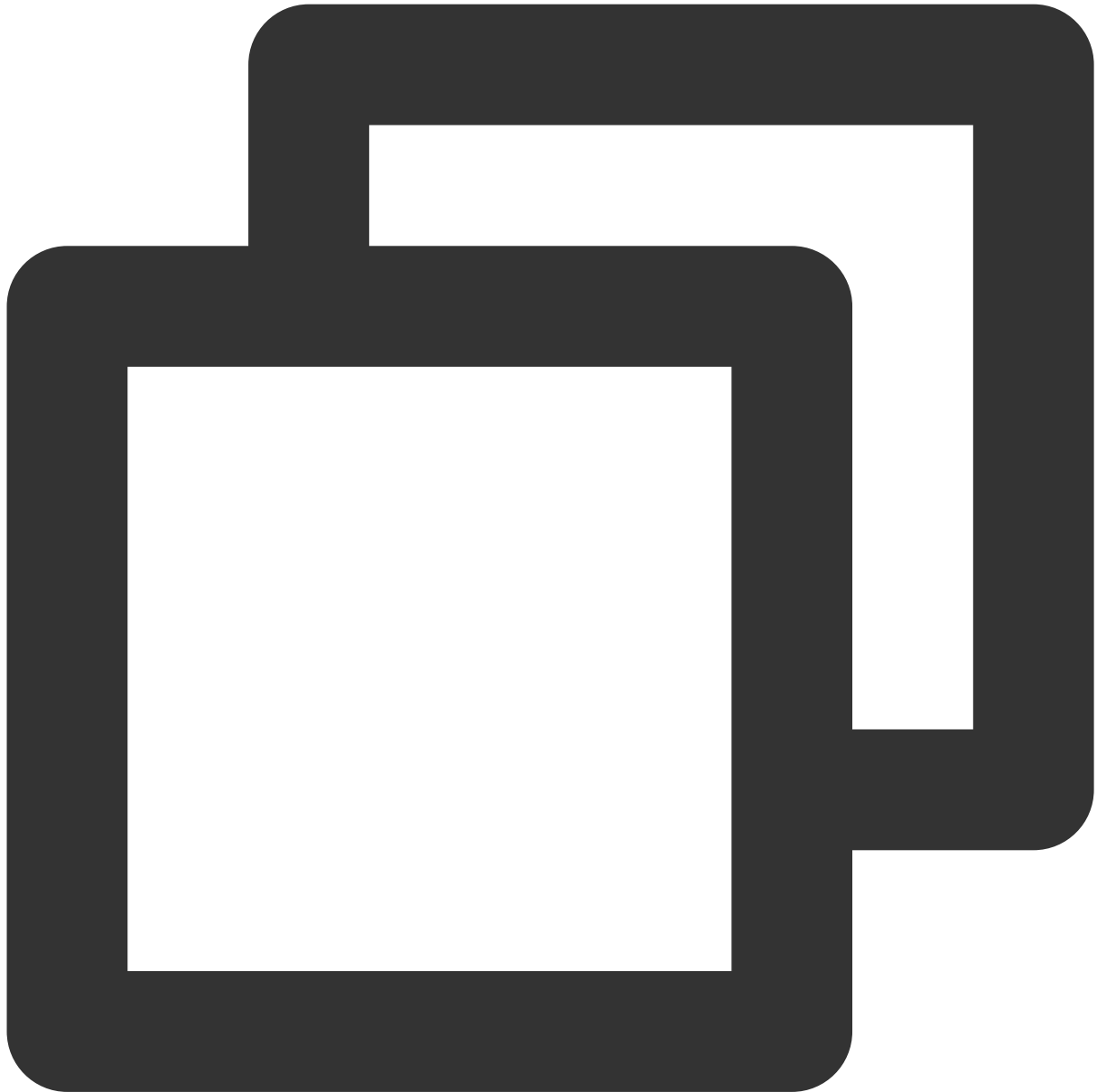


```
static void logout(TUICallback* callback);
```



## setSelfInfo

Set local user name and avatar.



```
static void SetSelfInfo(const char* userName, const char* avatarUrl, TUICallback* c
```

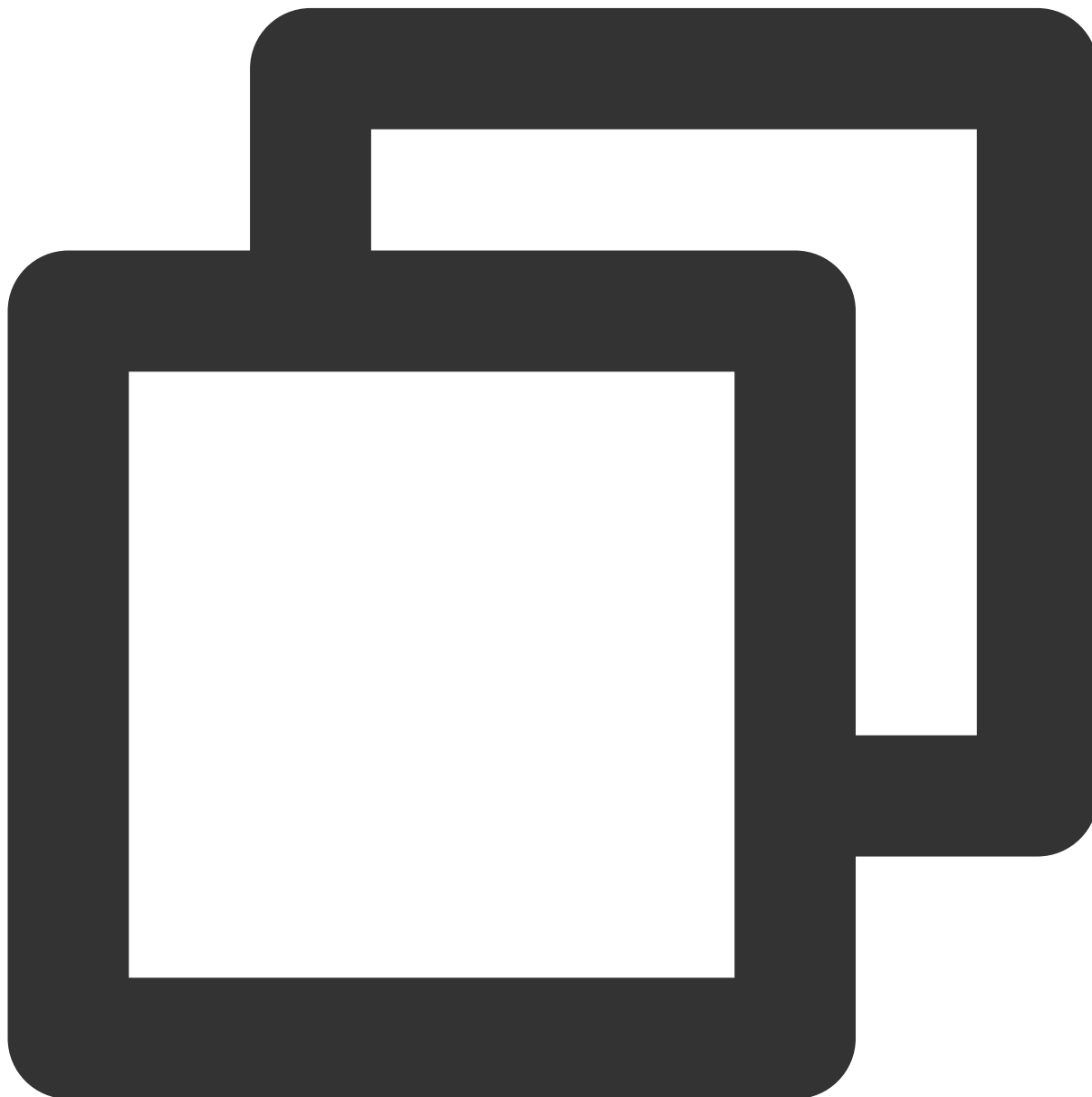
The parameters are as follows:

Parameter	Type	Meaning
userName	const char*	User Name
avatarUrl	const char*	User avatar URL address

callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call
----------	--------------	--

## getSelfInfo

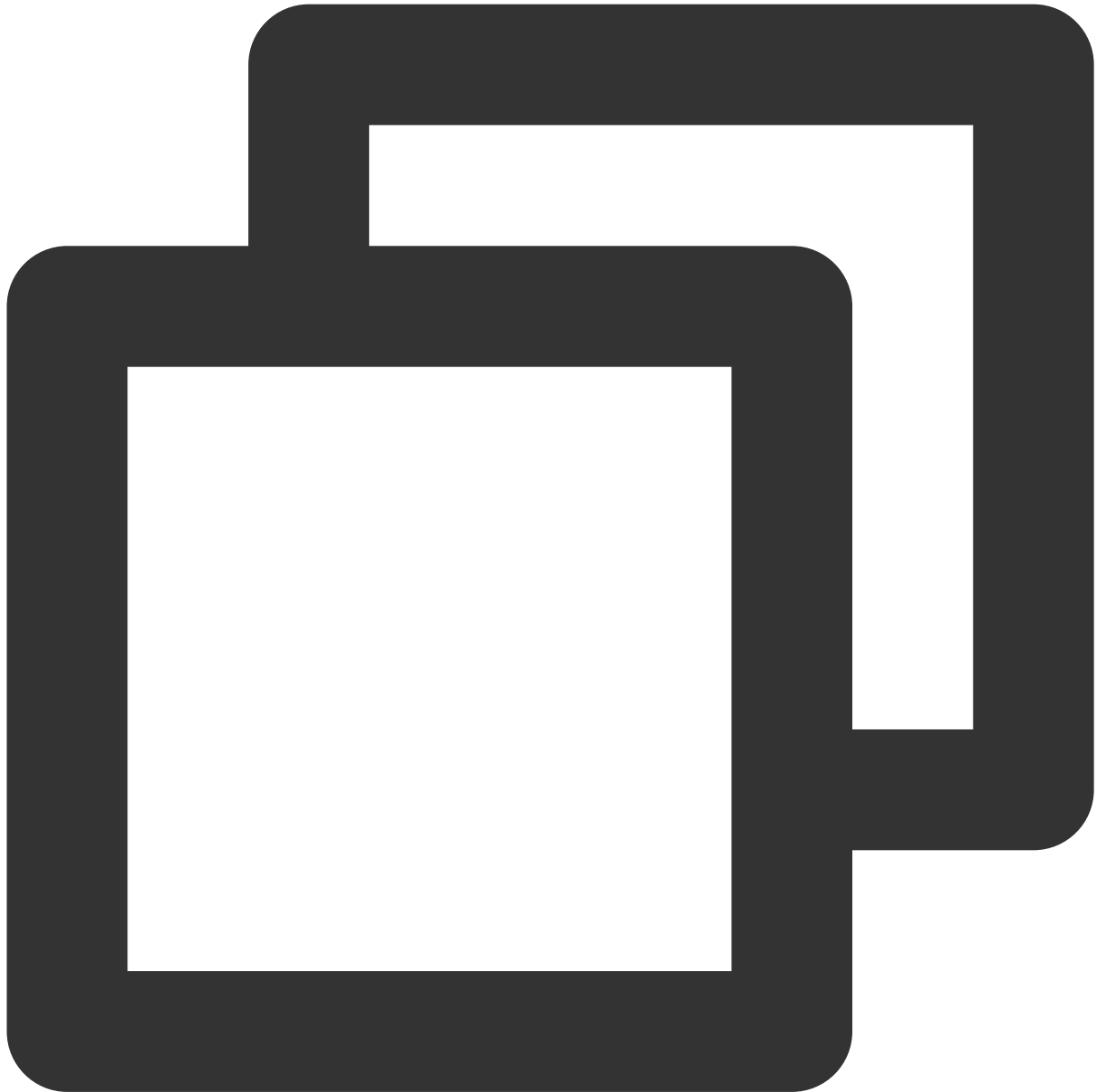
Get the basic information of the local user login.



```
static TUILoginUserInfo GetSelfInfo();
```

## addObserver

Add TUIRoomEngine Event Callback.



```
virtual void AddObserver(TUIRoomObserver* observer) = 0;
```

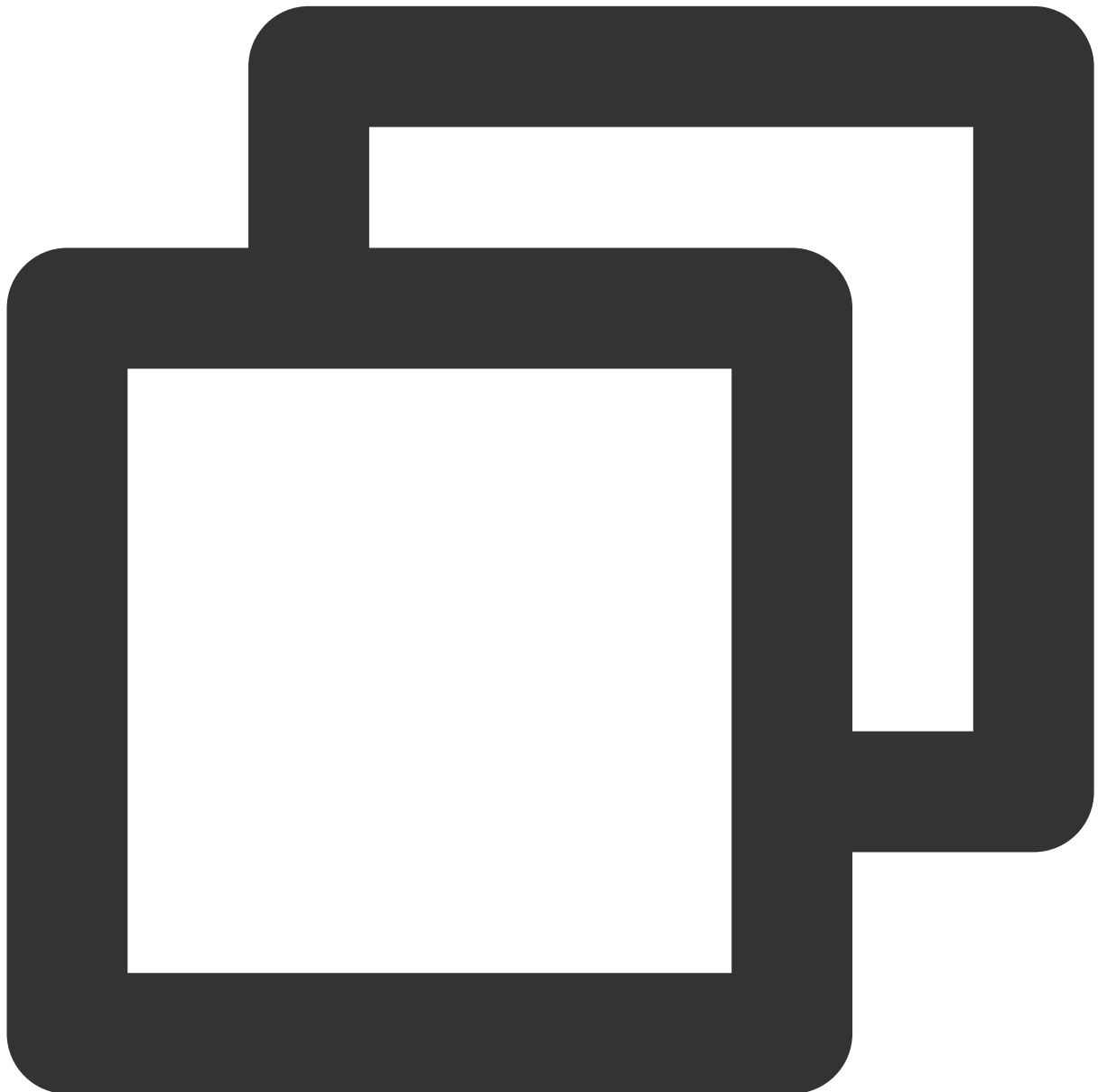
The parameters are as follows:

Parameter	Type	Meaning
observer	TUIRoomObserver*	Callback Instance Pointer, you can get various event notifications (such as: Error Code, Remote User Enter

		Room, Audio and Video Status Parameters, etc.) through TUIRoomObserver.
--	--	---

## removeObserver

Remove TUIRoomEngine Event Callback.



```
virtual void removeObserver(TUIRoomObserver* observer) = 0;
```

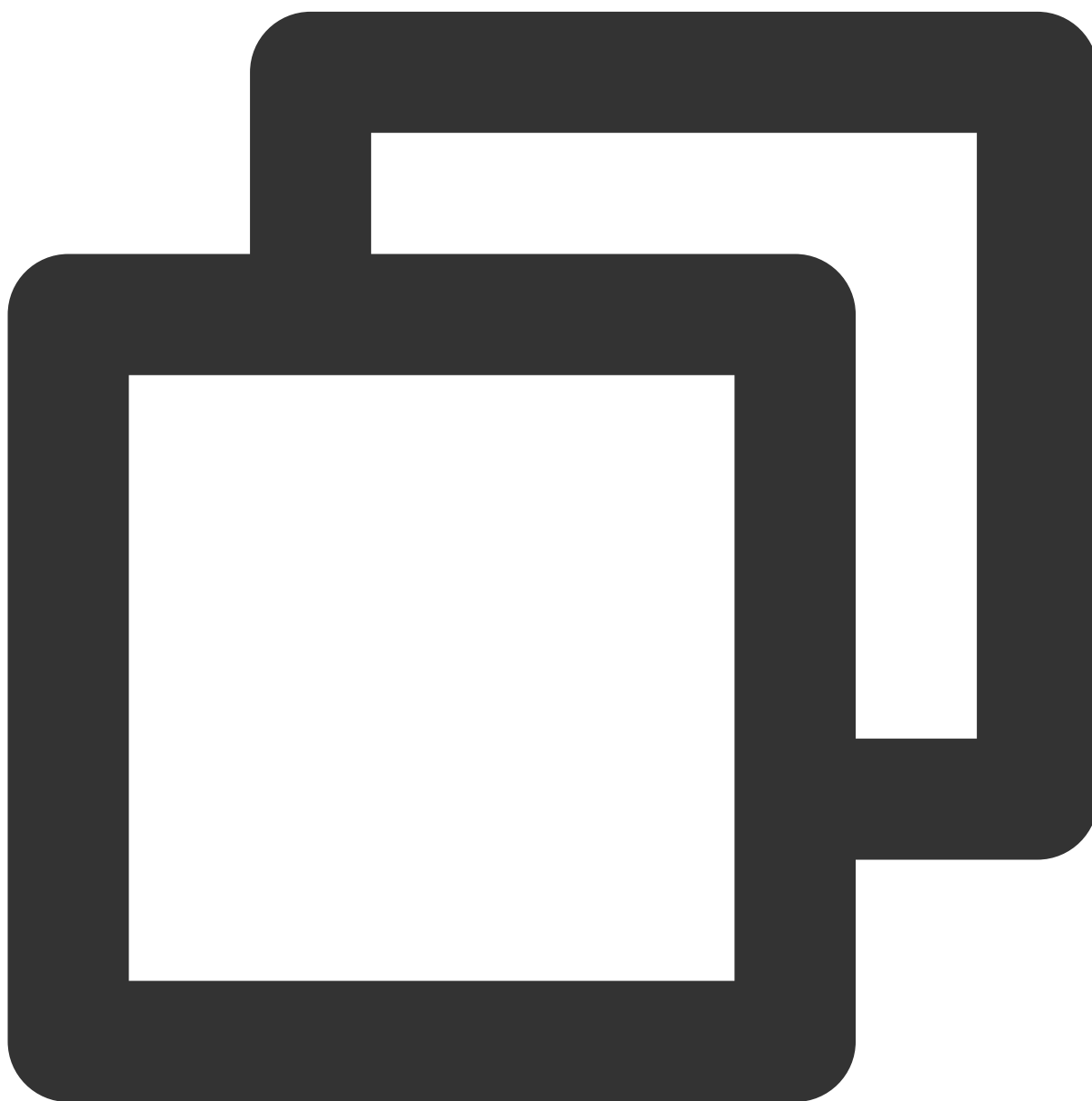
The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

observer	TUIRoomObserver*	TUIRoomEngine Event Callback
----------	------------------	------------------------------

## createRoom

Create room.



```
virtual void createRoom(const TUIRoomInfo& roomInfo, TUICallback* callback) = 0;
```

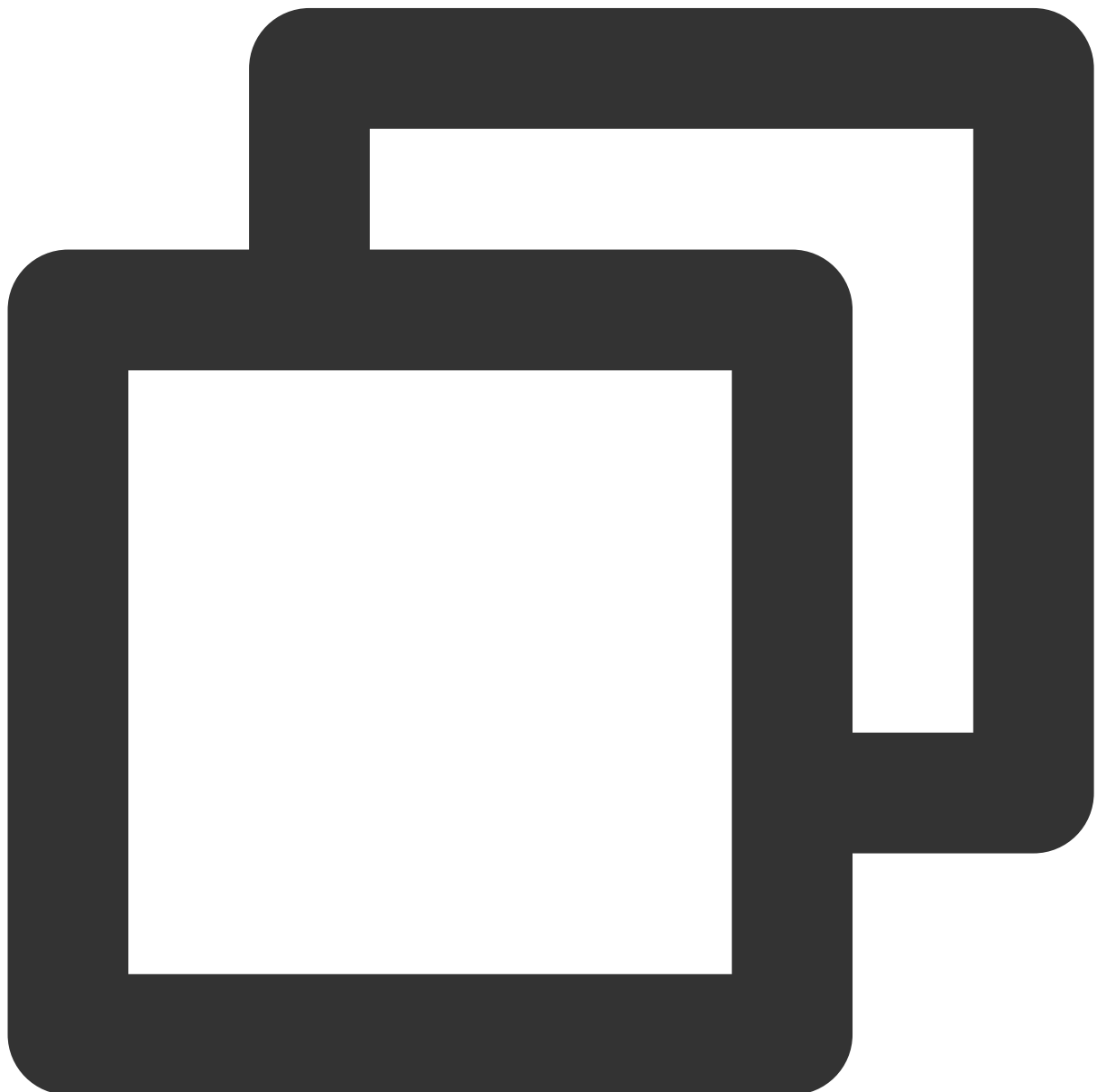
The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

roomInfo	const TUIRoomInfo&	Room data
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## destroyRoom

Destroy Room.



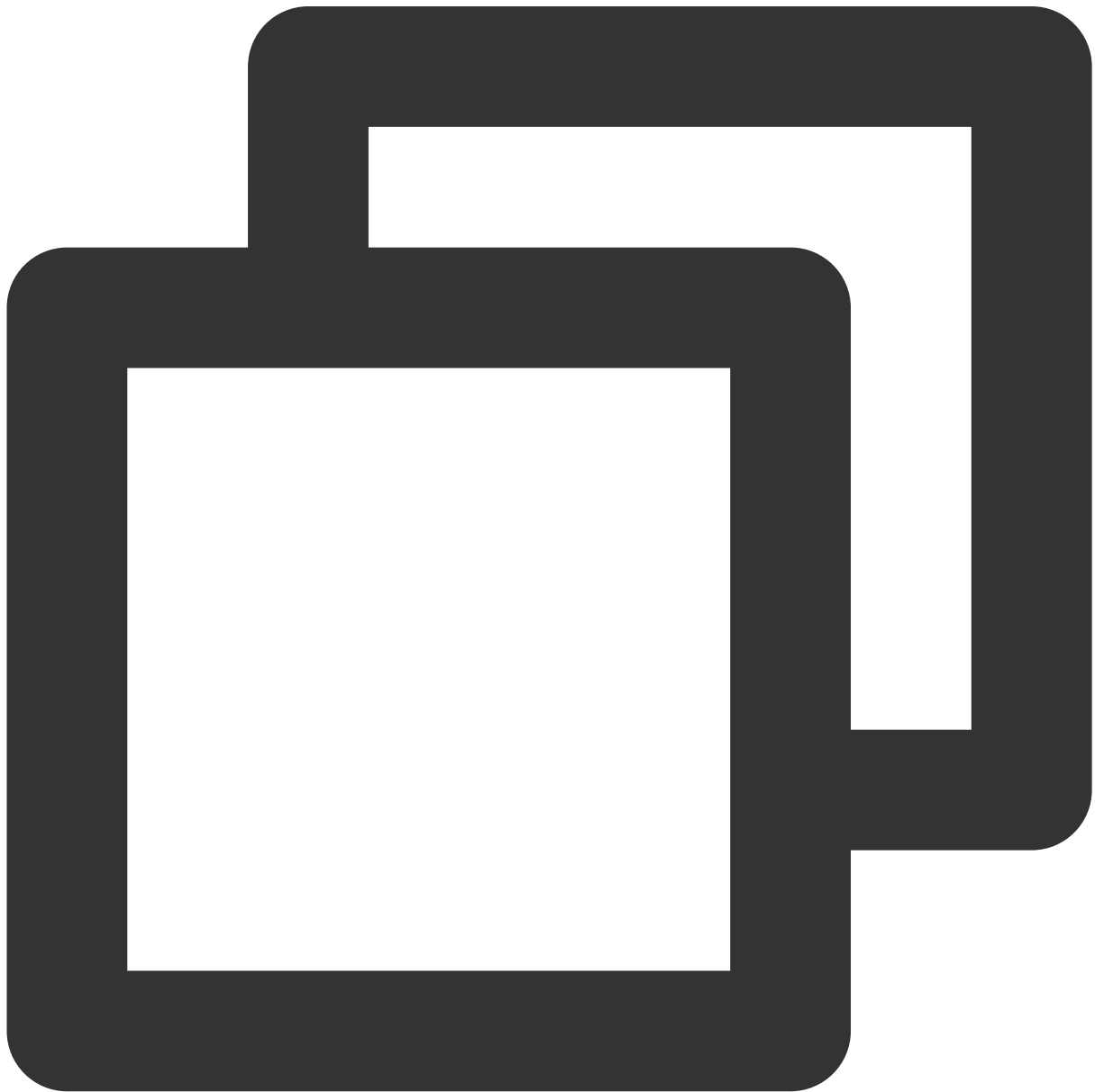
```
virtual void destroyRoom(TUICallback* callback) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## enterRoom

Entered room.



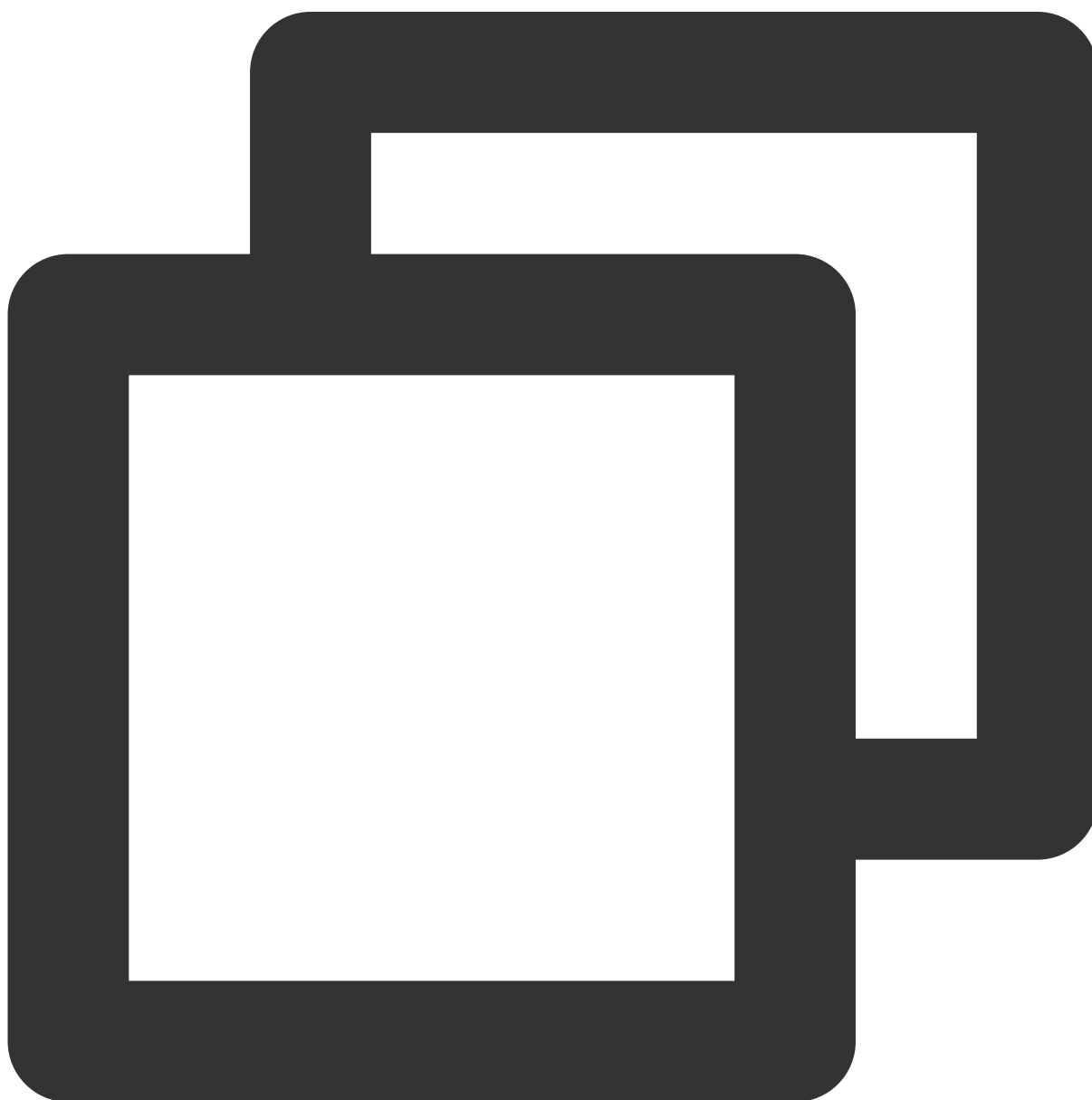
```
virtual void enterRoom(const char* roomId, TUICallback<TUIRoomInfo>* callback)
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	const char*	Room ID
callback	TUIValueCallback<TUIRoomInfo>*	Get the entered Room data Callback

## exitRoom

Exit Room.





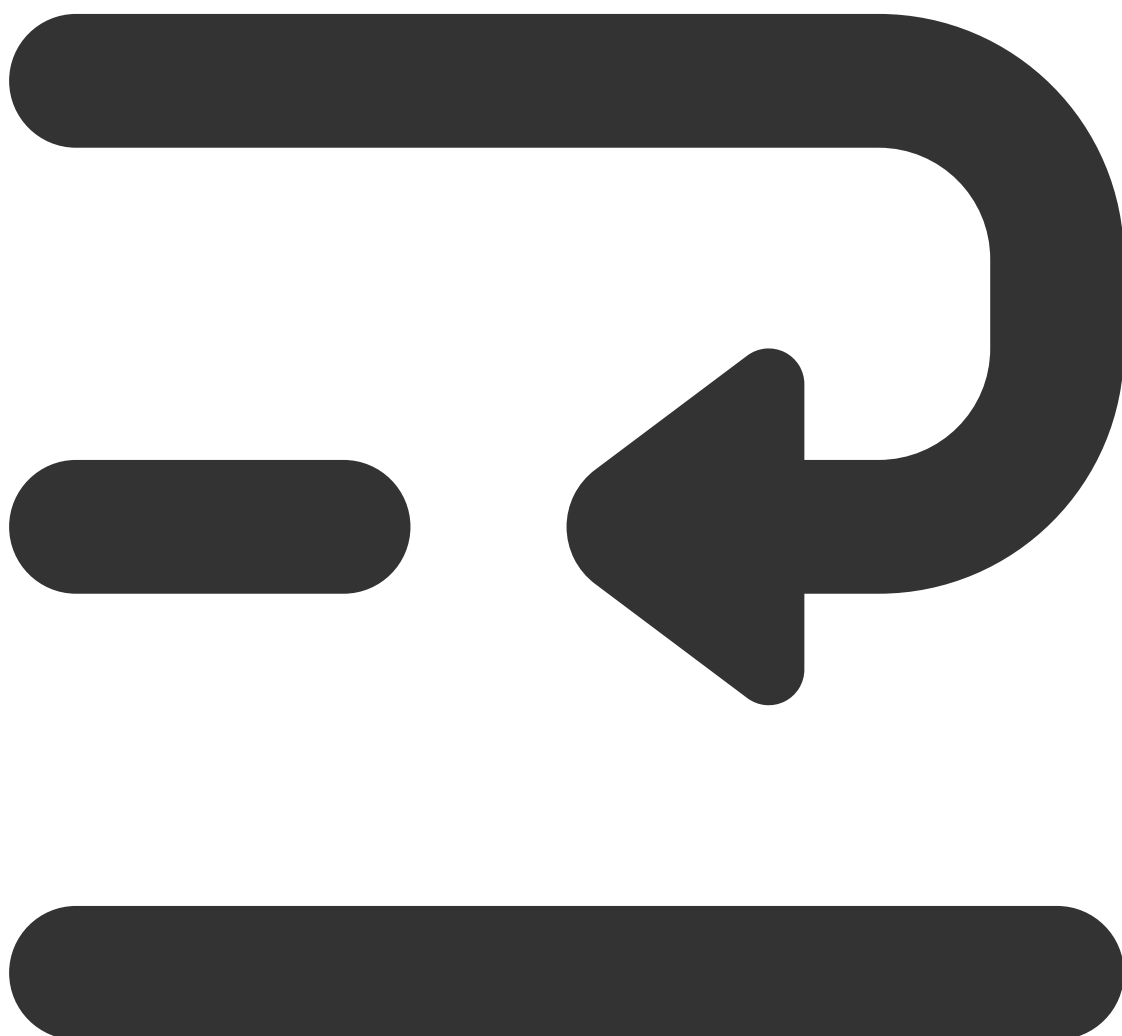
```
virtual void exitRoom(bool syncWaiting, TUICallback* callback) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
syncWaiting	bool	Whether to synchronize leaving the room
callback	TUICallback*	Leave Room Result Callback

## connectOtherRoom

Connect to other rooms.





```
virtual TUIRequest connectOtherRoom(const char* roomId, const char* userId, int tim
```

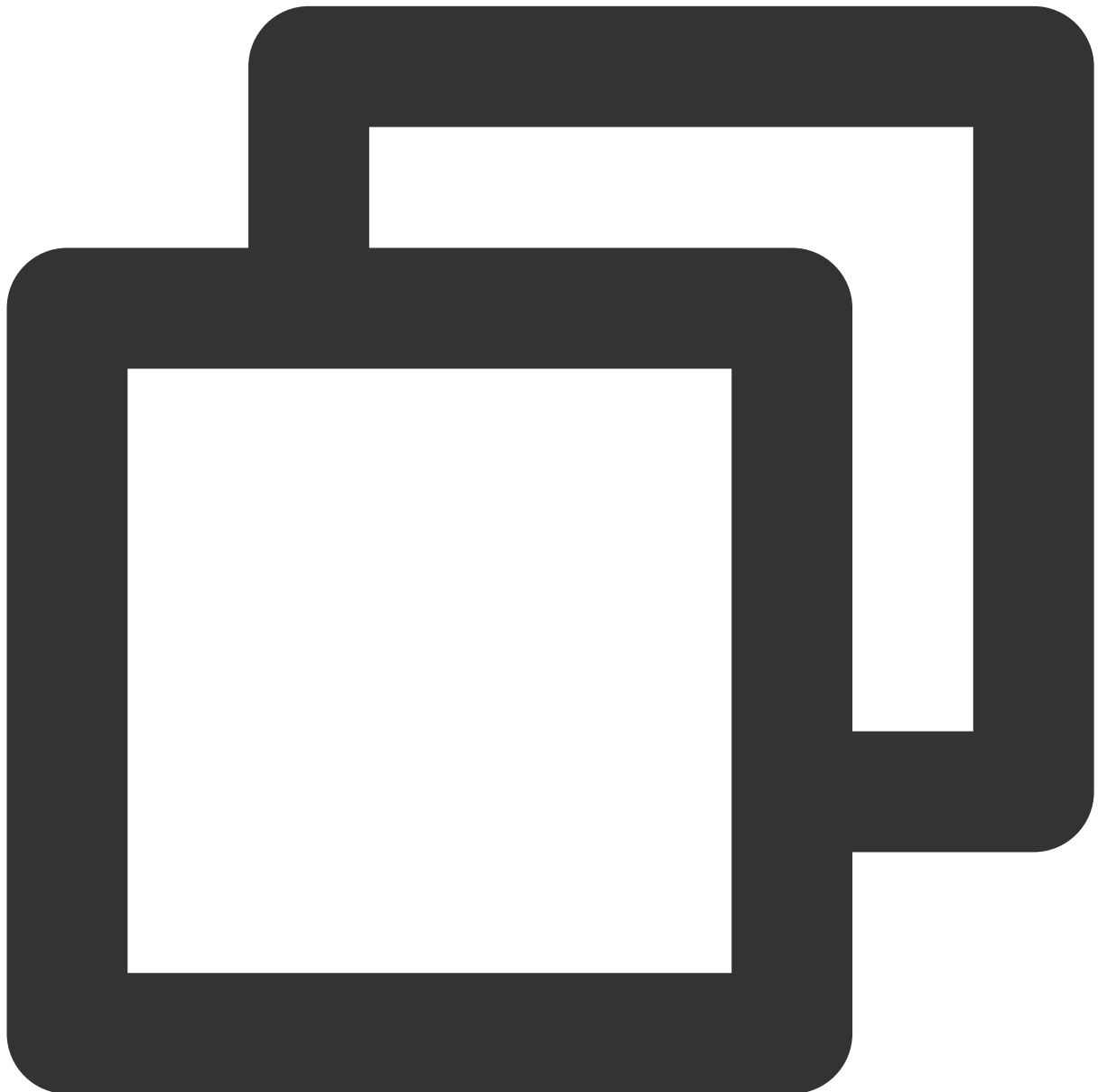
**The parameters are as follows:**

Parameter	Type	Meaning
roomId	const char*	Room ID
userId	const char*	User ID
timeout	int	Time

callback	TUIRequestCallback*	Callback of API, used to Notify the Success or Failure of the API call
----------	---------------------	--

## disconnectOtherRoom

Disconnect from other rooms



```
virtual void disconnectOtherRoom(TUICallback* callback) = 0;
```

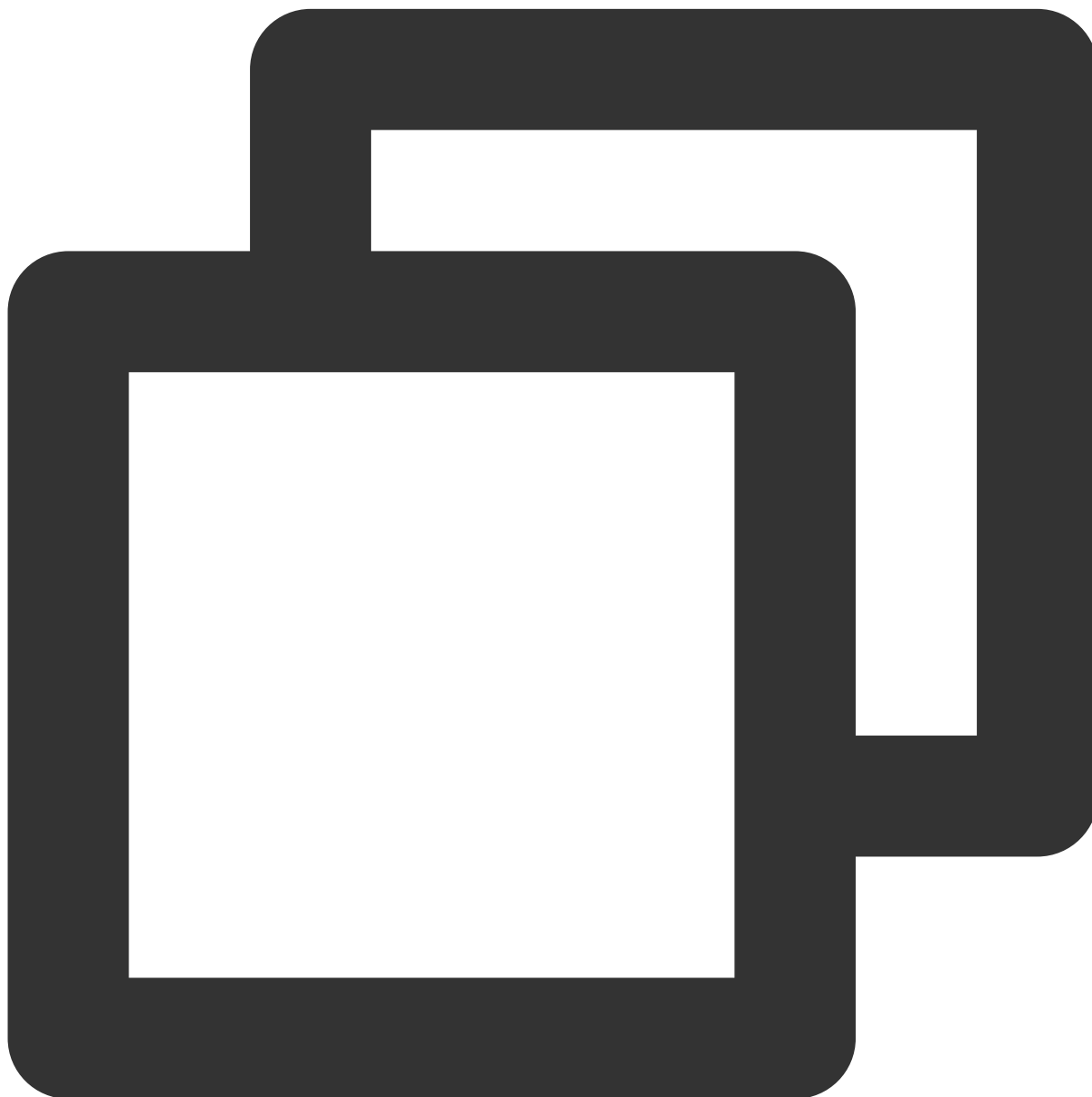
The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call
----------	--------------	--

## fetchRoomInfo

Get Room data.



```
virtual void fetchRoomInfo(TUIValueCallback<TUIRoomInfo>* callback) = 0;
```

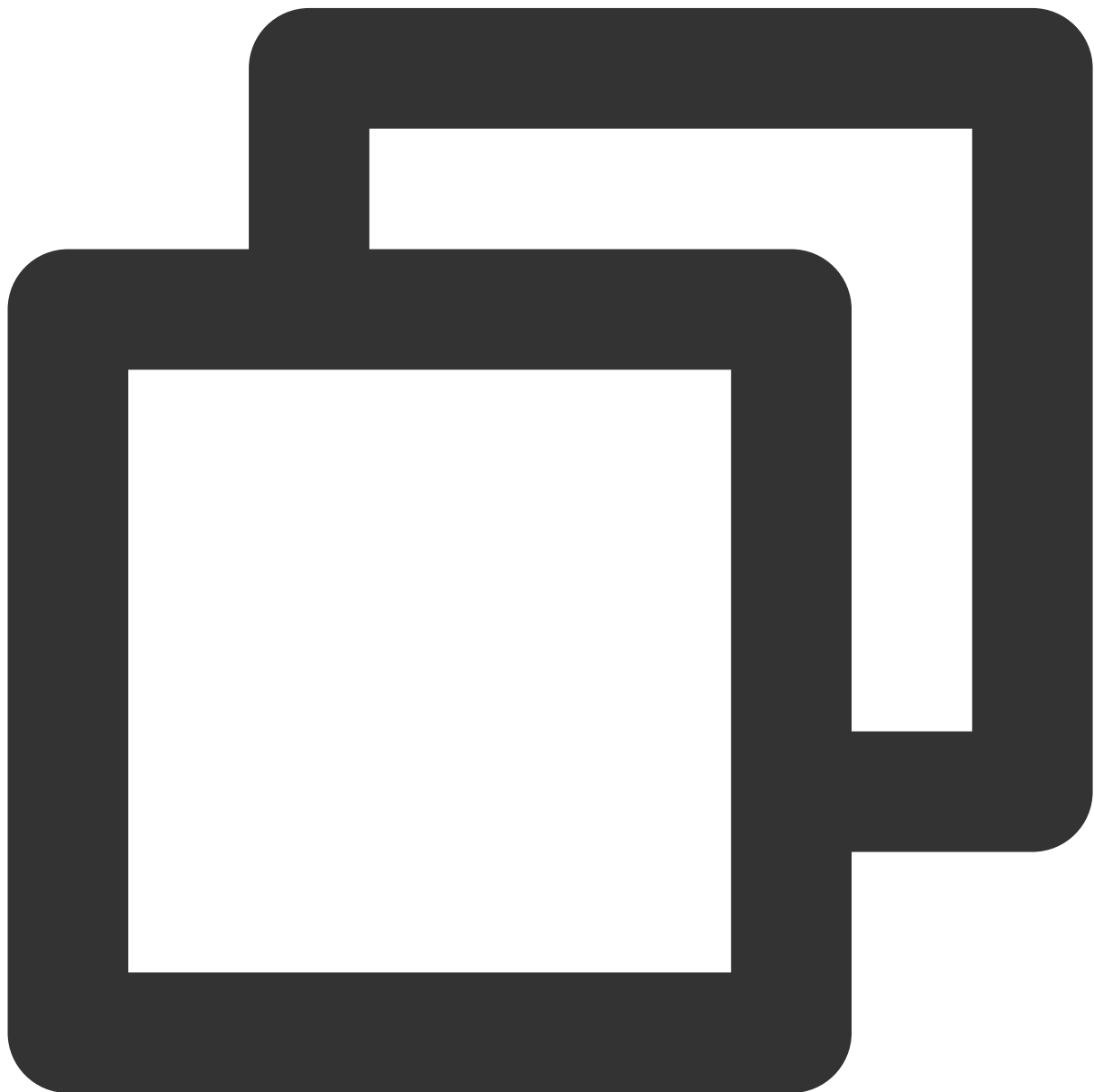
The parameters are as follows:

--	--	--

Parameter	Type	Meaning
callback	TUIValueCallback<TUIRoomInfo>*	Callback of API, used to Notify the Success or Failure of the API call

## updateRoomNameByAdmin

Update Room ID



```
virtual void updateRoomNameByAdmin(const char* roomName, TUICallback* callback) = 0
```

The parameters are as follows:

Parameter	Type	Meaning
roomName	const char*	Room ID
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## updateRoomSpeechModeByAdmin

Set Management mode.



```
virtual void updateRoomSpeechModeByAdmin(TUISpeechMode mode, TUICallback* callback)
```

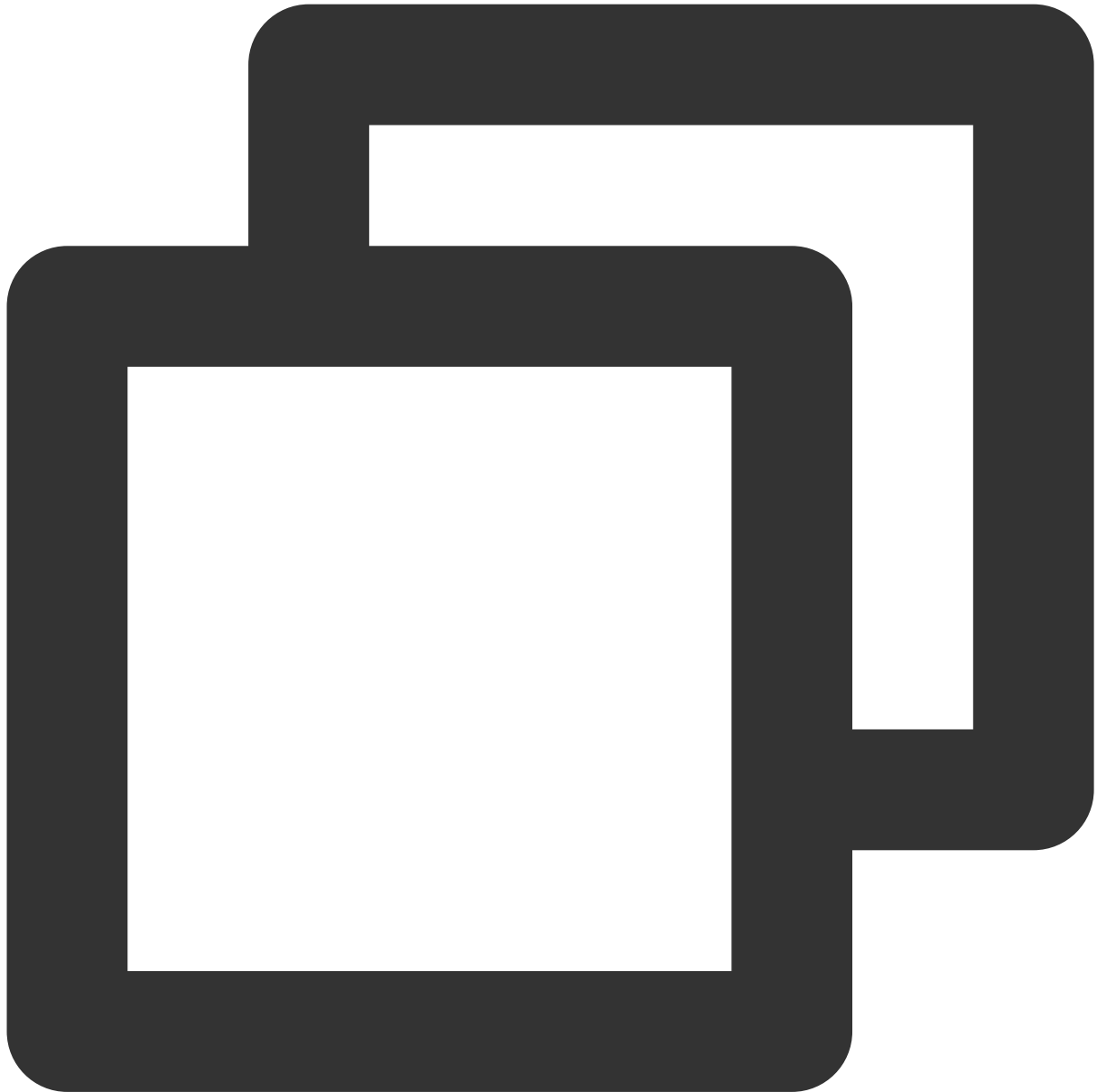
**The parameters are as follows:**

Parameter	Type	Meaning
mode	TUISpeechMode	Management mode
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call



## setLocalVideoView

Set Local user Video Rendering View control



```
virtual void setLocalVideoView(TUIVideoStreamType streamType, const TUIVideoView& v
```

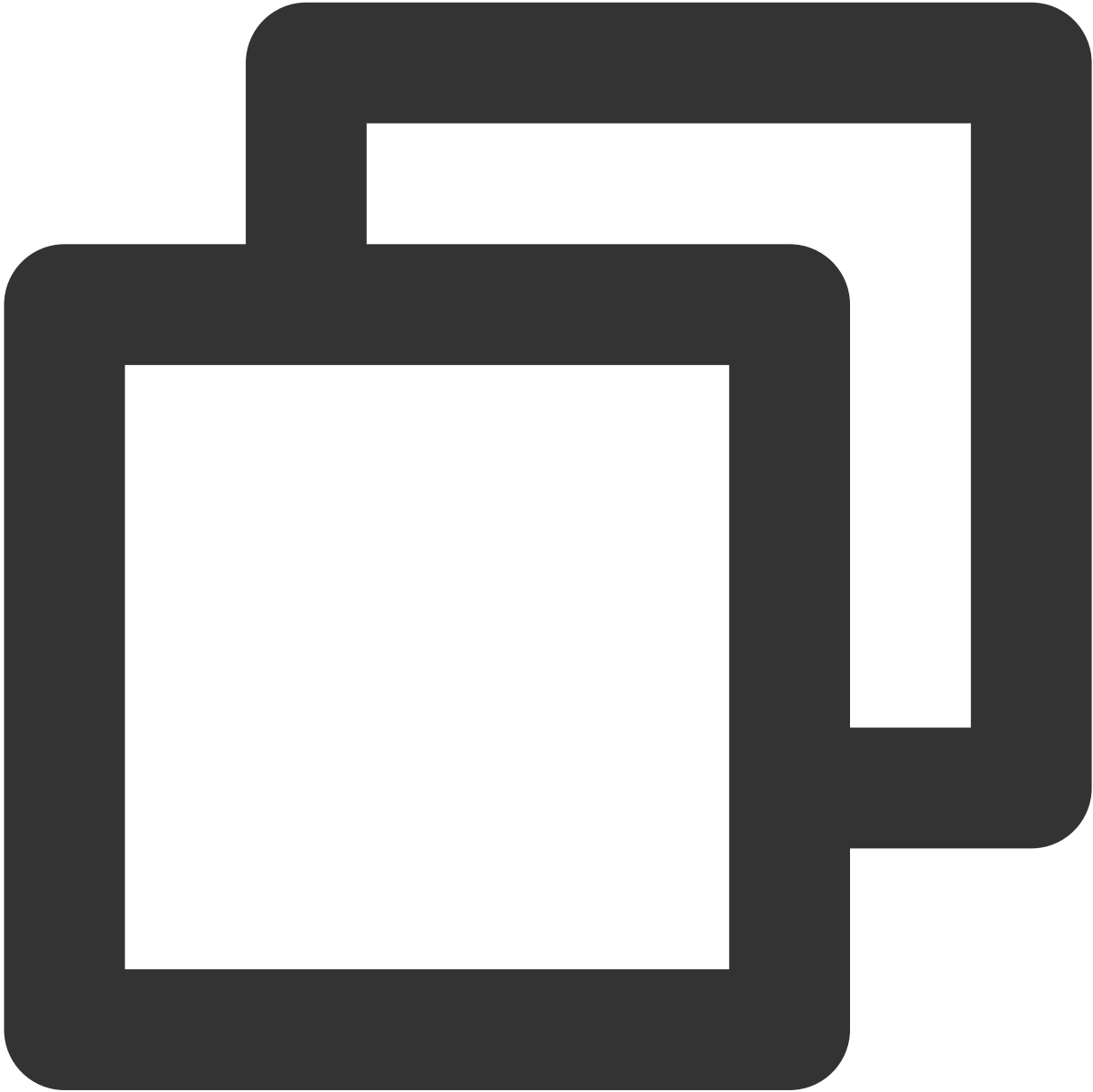
The parameters are as follows:

Parameter	Type	Meaning
streamType	TUIVideoStreamType	Local streams type
view	const	To be rendered view, Video Rendering on this view

	TUIVideoView&	
--	---------------	--

### openLocalCamera

Open Local Camera, Start Video Capturing.



```
virtual void openLocalCamera(bool isFront, TUIVideoQuality quality, TUICallback* ca
```

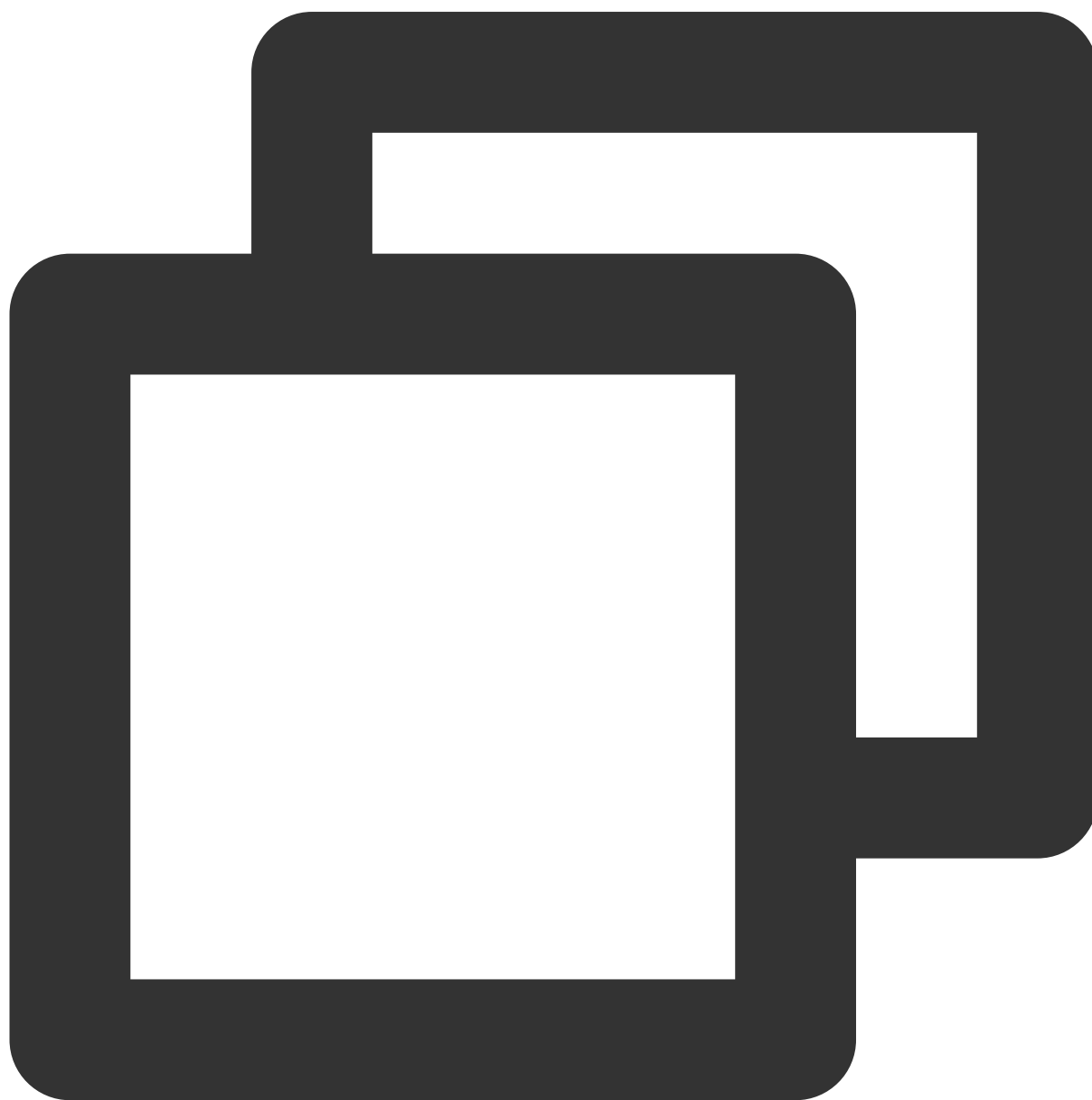
The parameters are as follows:

Parameter	Type	Meaning

isFront	bool	Is it a front Camera (Invalid on Windows)
quality	TUIVideoQuality	Video Quality
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## closeLocalCamera

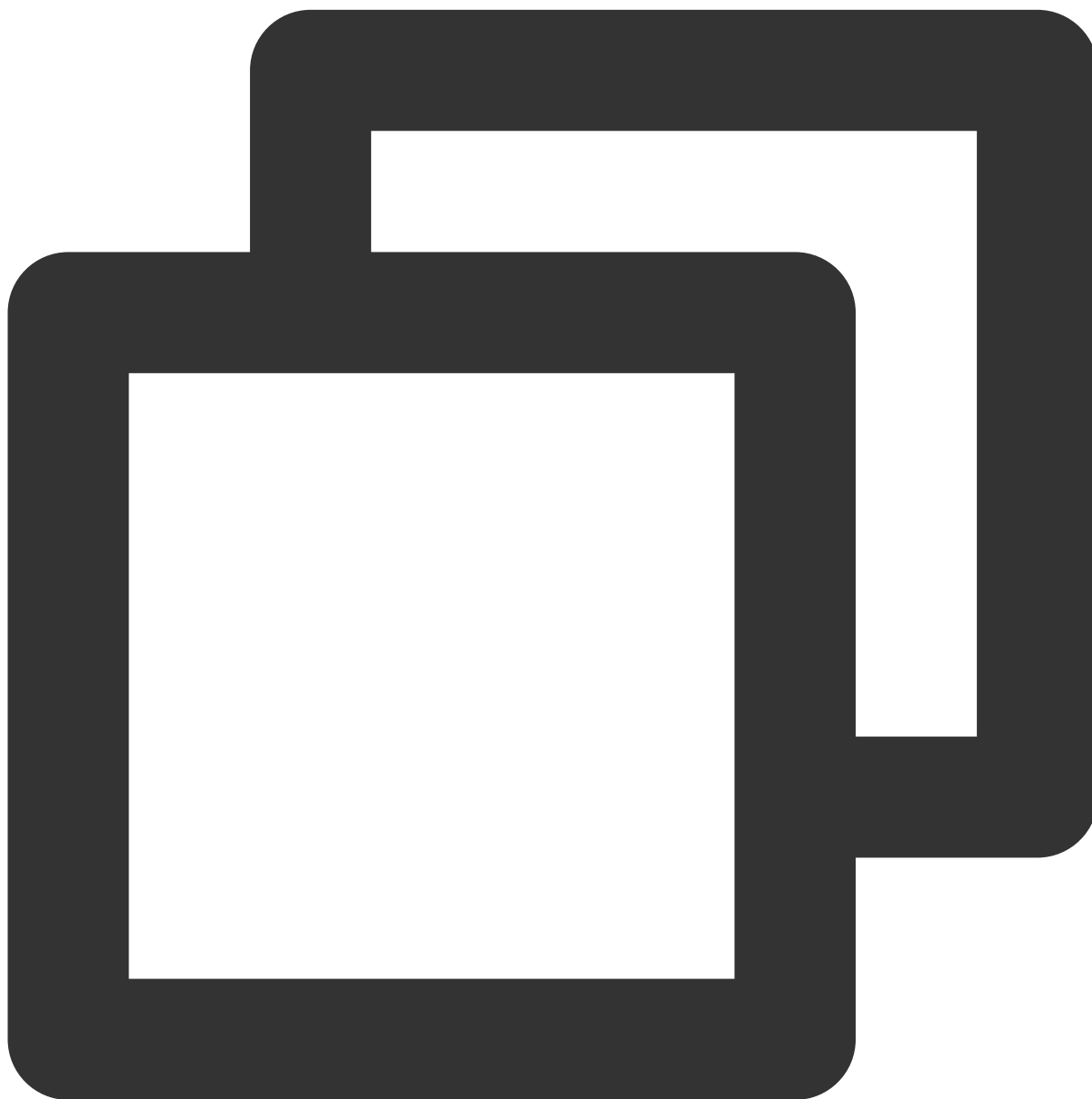
Close Local Camera.



```
virtual void closeLocalCamera() = 0;
```

## updateVideoQuality

Set Local Video Parameter.



```
virtual void updateVideoQuality(TUIVideoQuality quality) = 0;
```

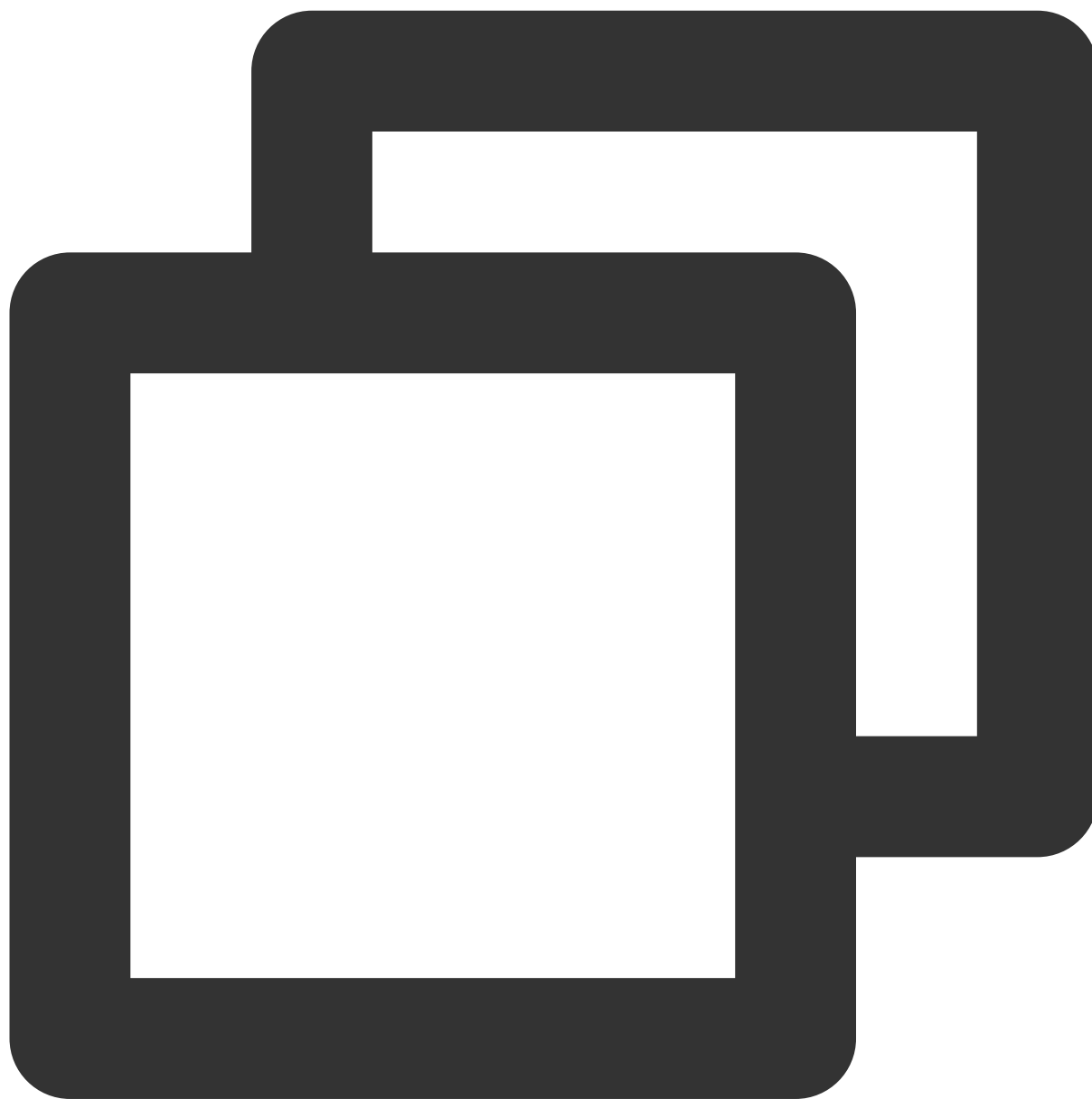
The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

quality	TUIVideoQuality	Video Quality
---------	-----------------	---------------

## startScreenSharing

Start Screen sharing.



```
virtual void startScreenSharing(const TUISourceId& sourceId, TUICallback* callback)
```

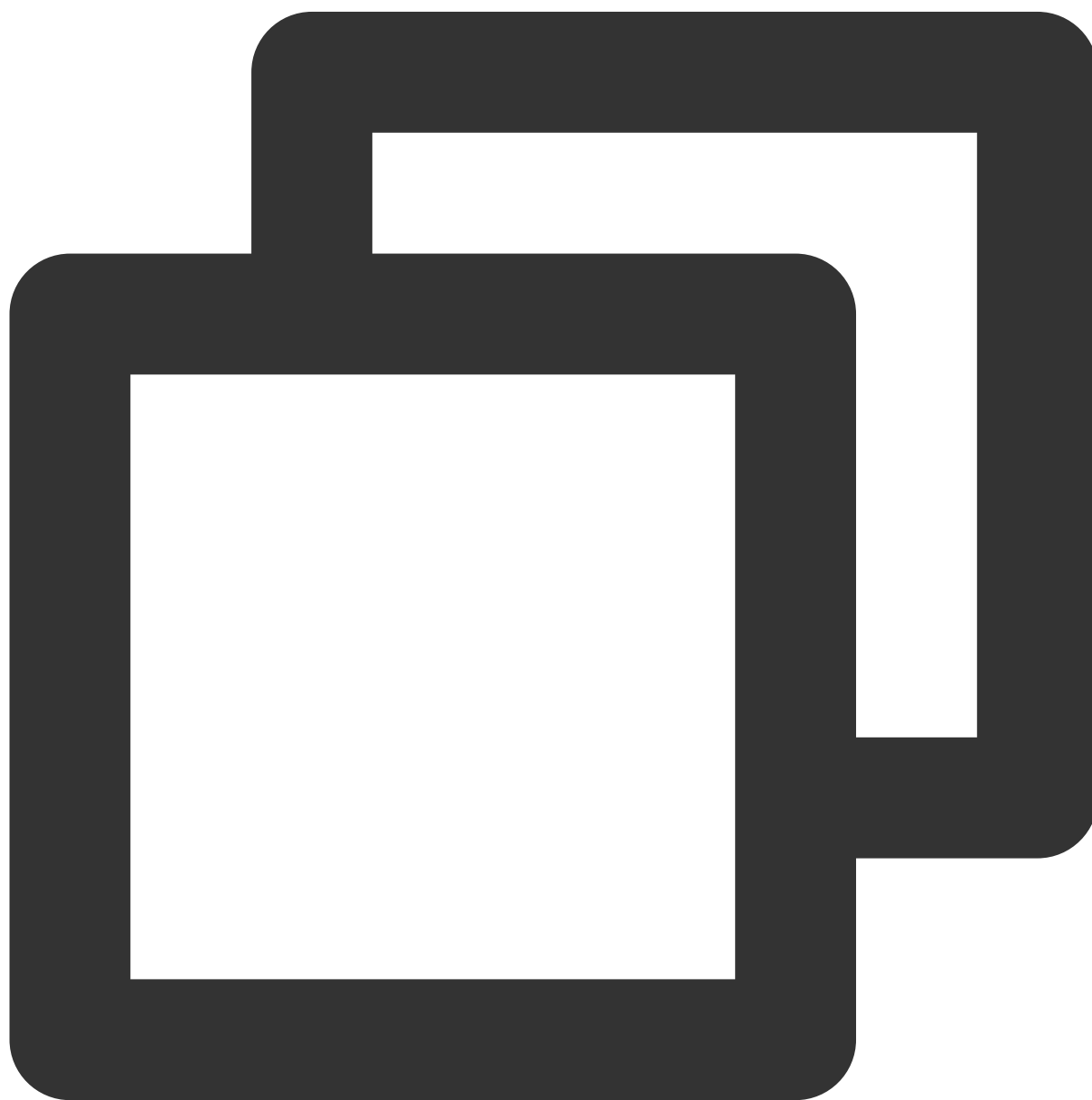
The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

sourceId	const TUISourceId&	Handle of the Screen Sharing window or screen, which can be obtained by calling GetScreenSharingTargetList
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## stopScreenSharing

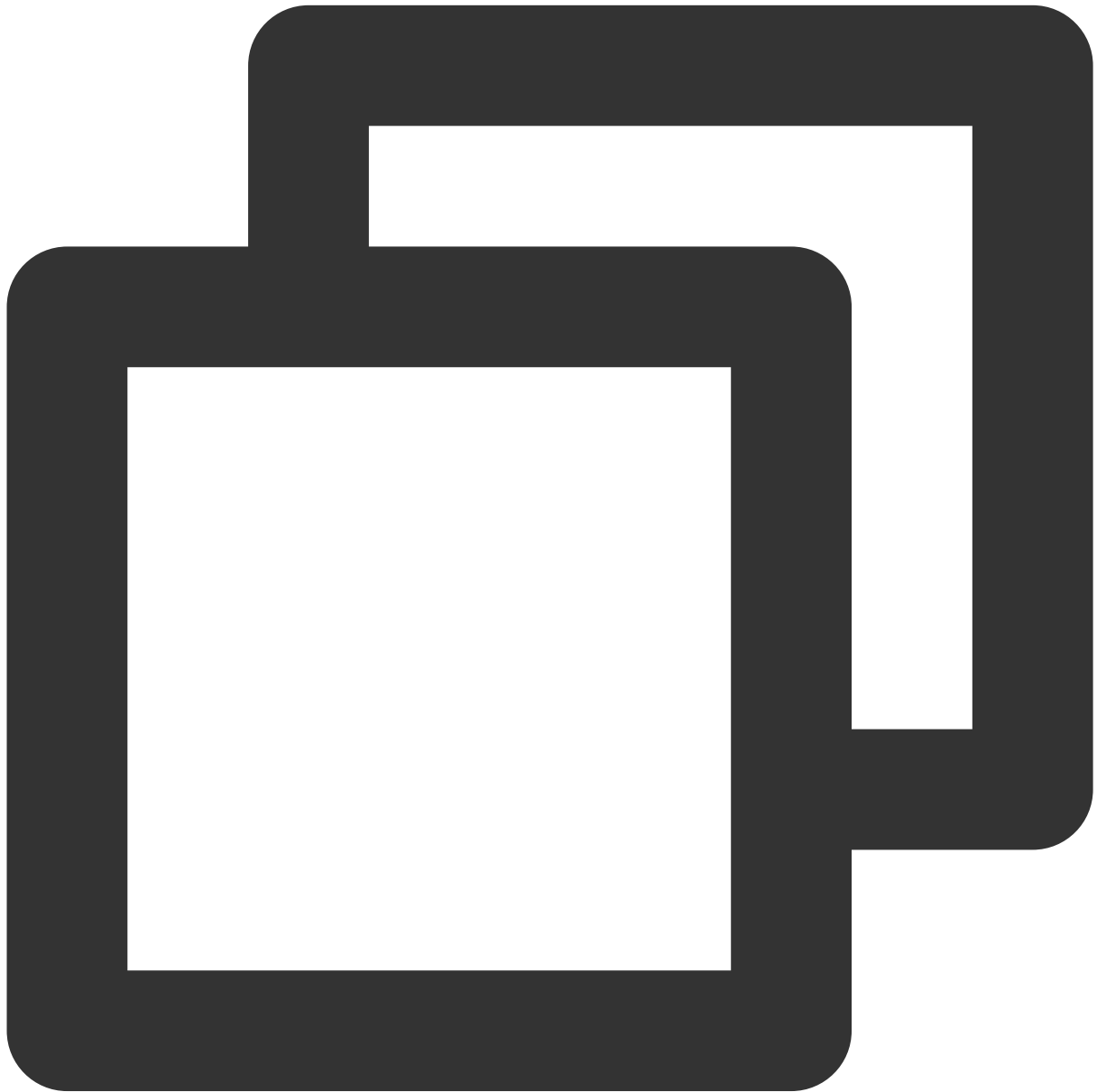
Stop Screen sharing.



```
virtual void stopScreenSharing() = 0;
```

## getScreenSharingTargetList

Get Sharing Object List.



```
virtual void getScreenSharingTargetList(TUICollectionCallback<TUIShareTarget>* list) = 0;
```

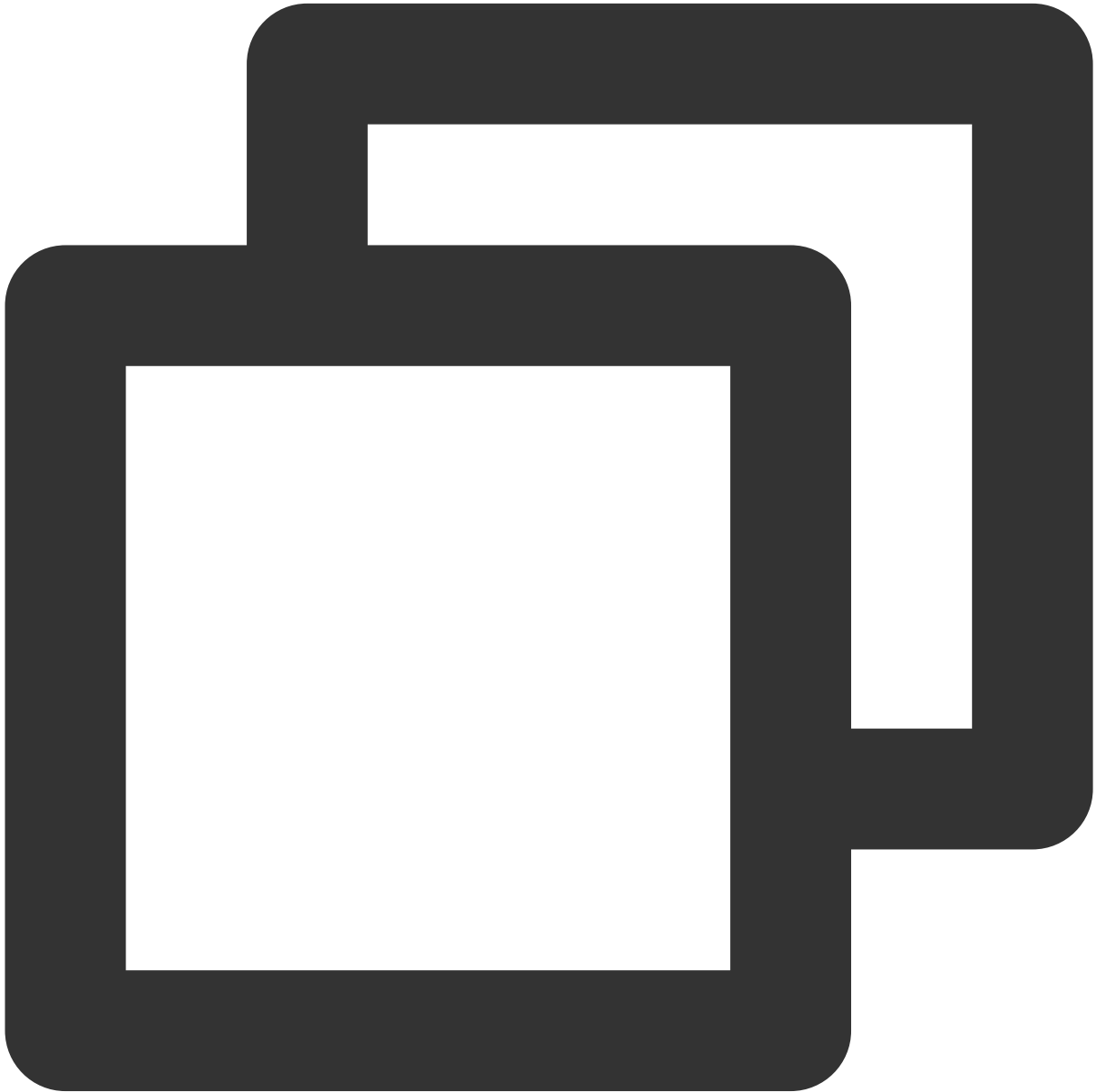
The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

list	TUIListCallback<TUIShareTarget>*	Callback of API, used to Notify the Success or Failure of the API call
------	----------------------------------	--

**selectScreenSharingTarget**

Select Screen Sharing Target.



```
virtual void selectScreenSharingTarget(const TUISourceId& sourceId) = 0;
```

The parameters are as follows:

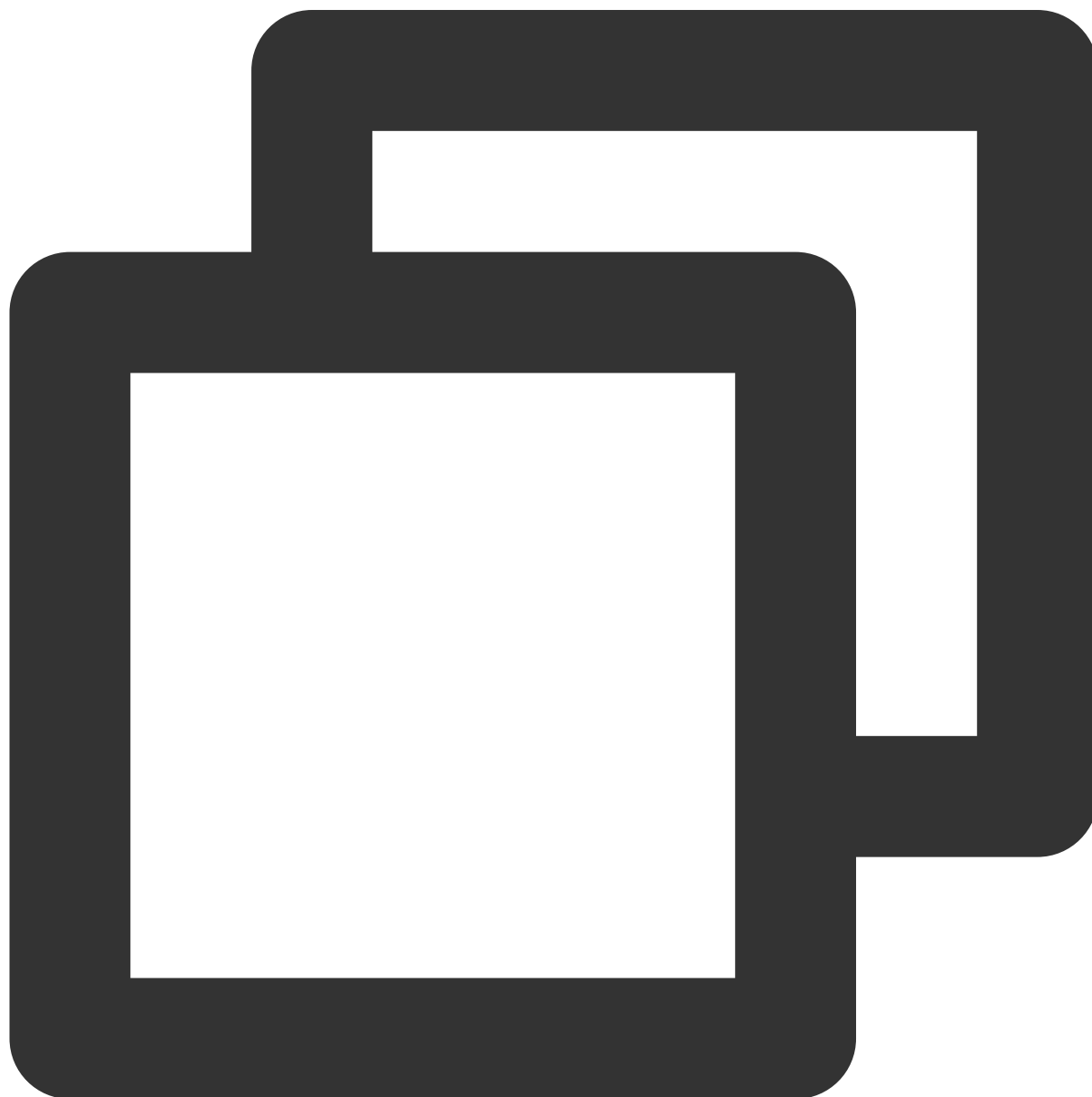
--	--	--



Parameter	Type	Meaning
sourceId	const TUISourceId&	Screen Sharing window or screen handle, can call GetScreenSharingTargetList to get.

## startPushLocalVideo

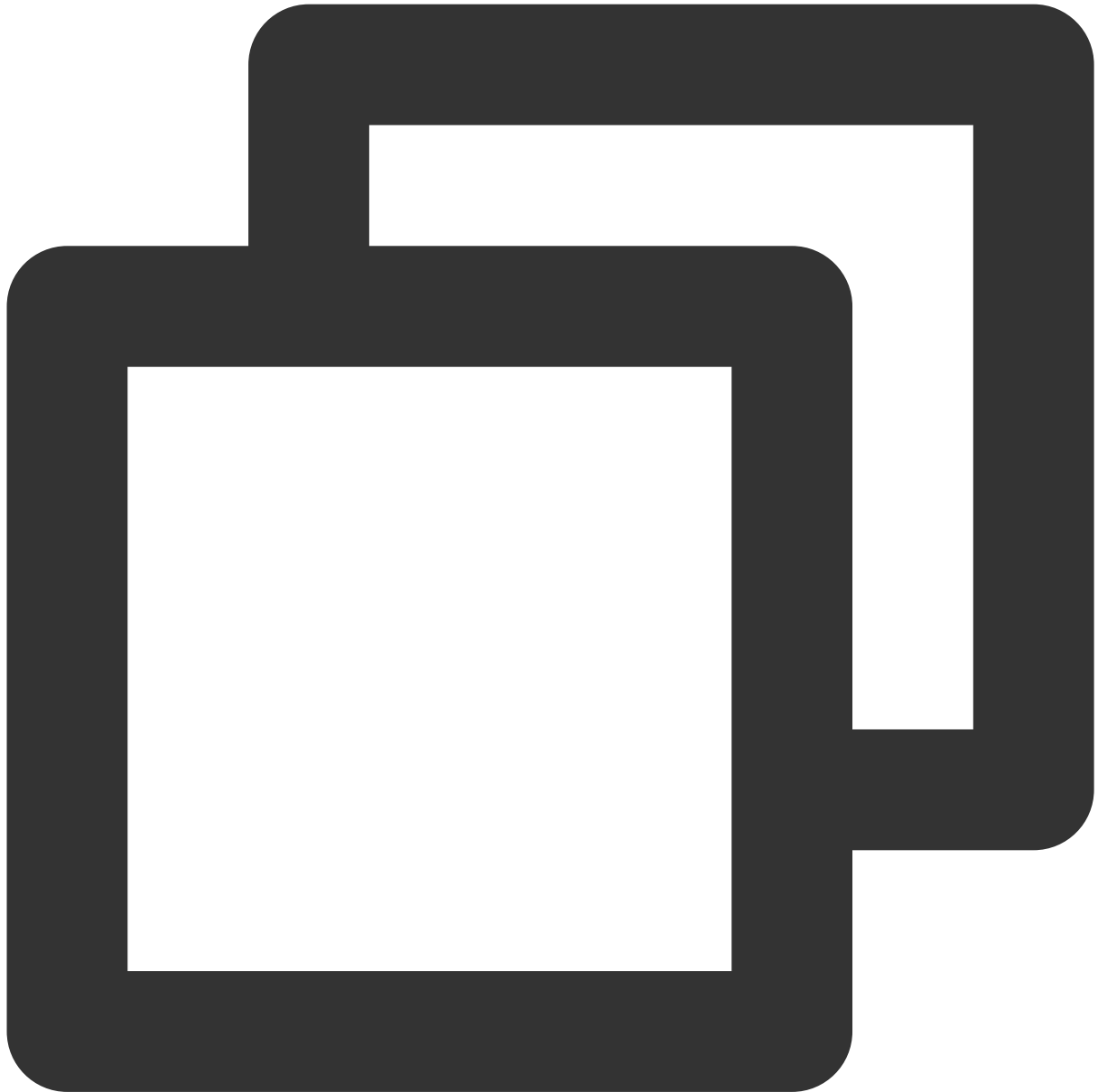
Start Pushing Local Video.



```
virtual void startPushLocalVideo() = 0;
```

## **stopPushLocalVideo**

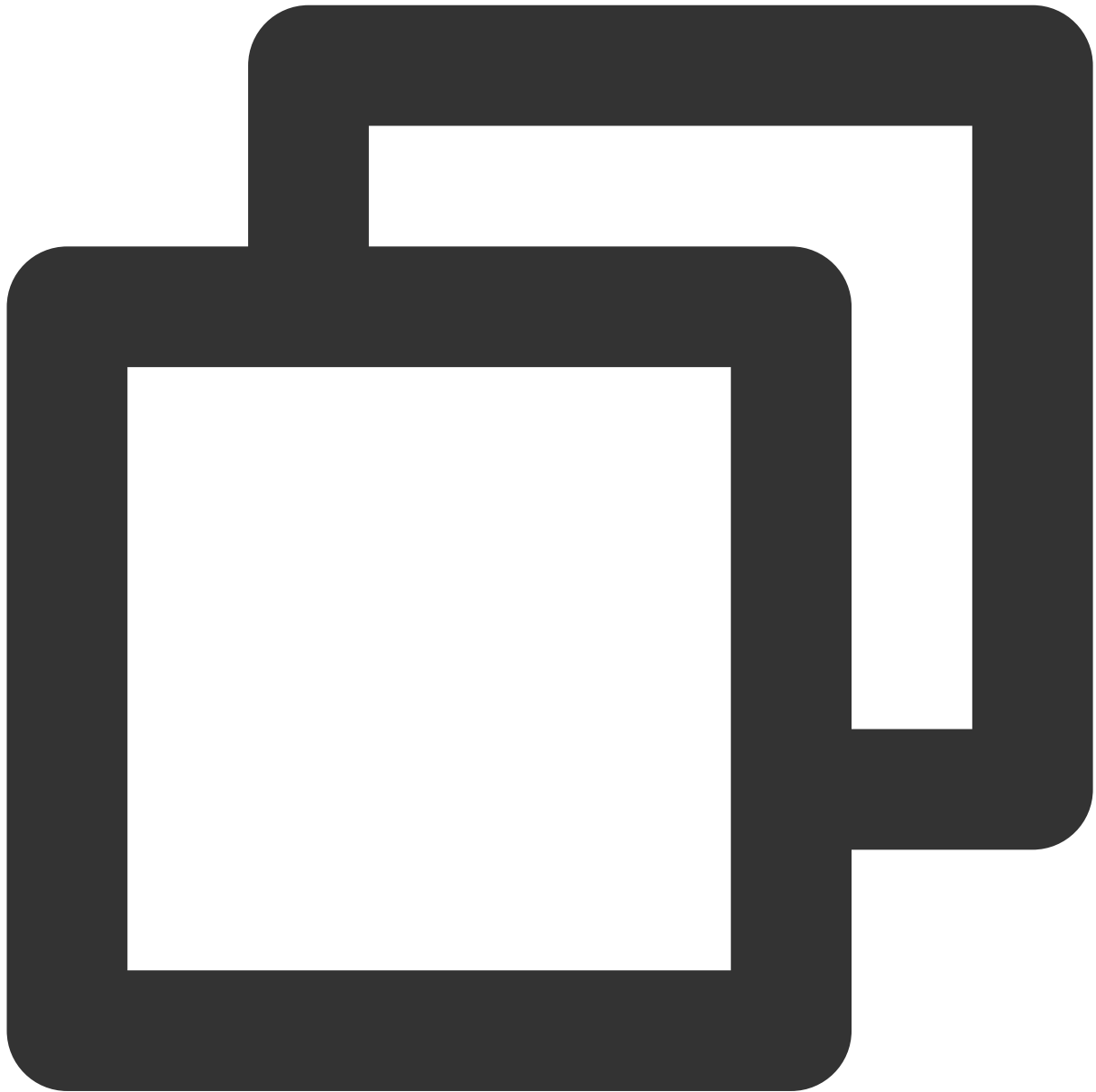
Stop Pushing Local Video.



```
virtual void stopPushLocalVideo() = 0;
```

## **openLocalMicrophone**

Open Local mic.



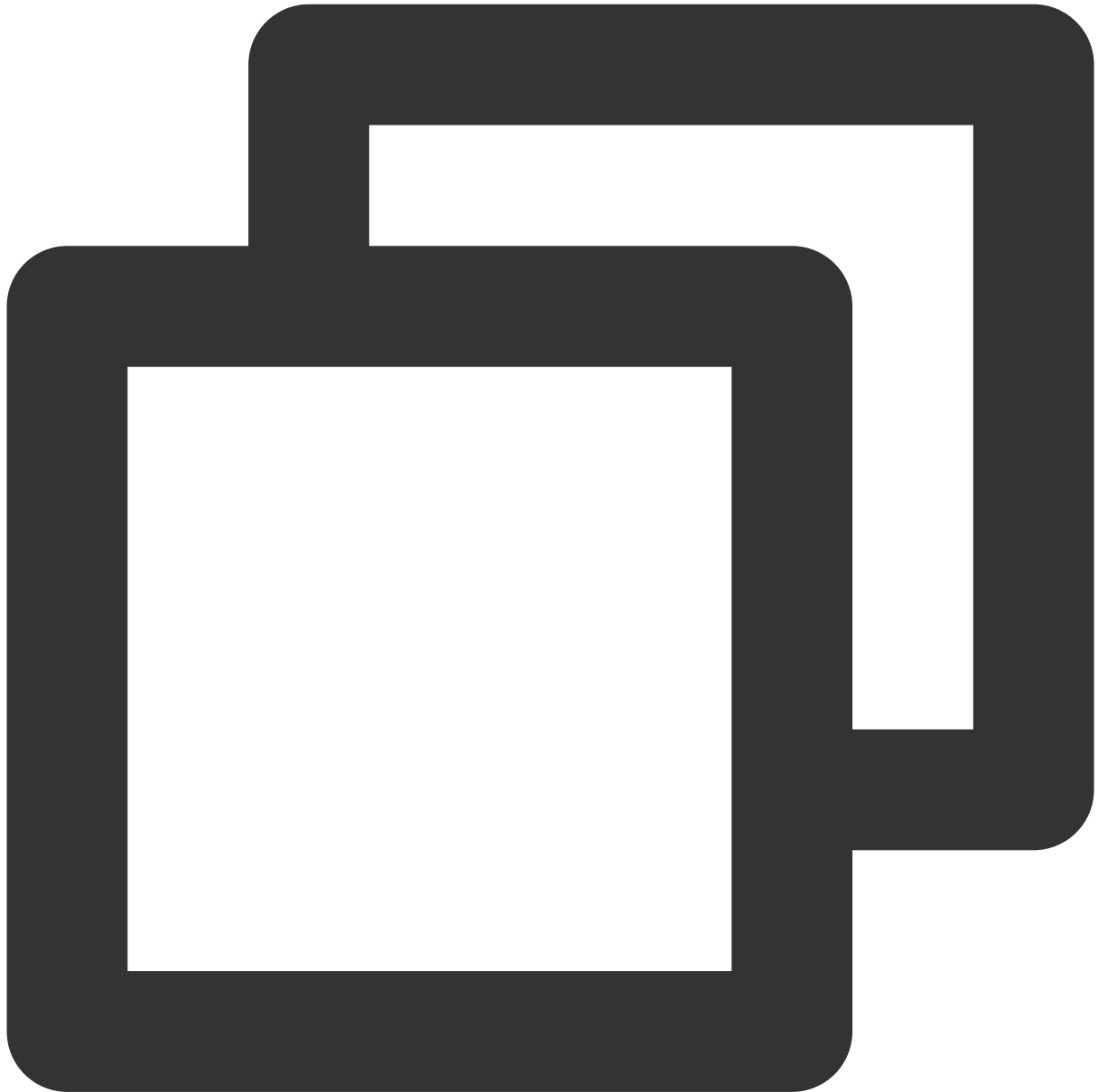
```
virtual void openLocalMicrophone(TUIAudioQuality quality, TUICallback* callback) =
```

**The parameters are as follows:**

Parameter	Type	Meaning
quality	TUIAudioQuality	Audio Quality
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## closeLocalMicrophone

Close Local mic.



```
virtual void closeLocalMicrophone() = 0;
```

## updateAudioQuality

Update Local Audio Codec Quality setting.



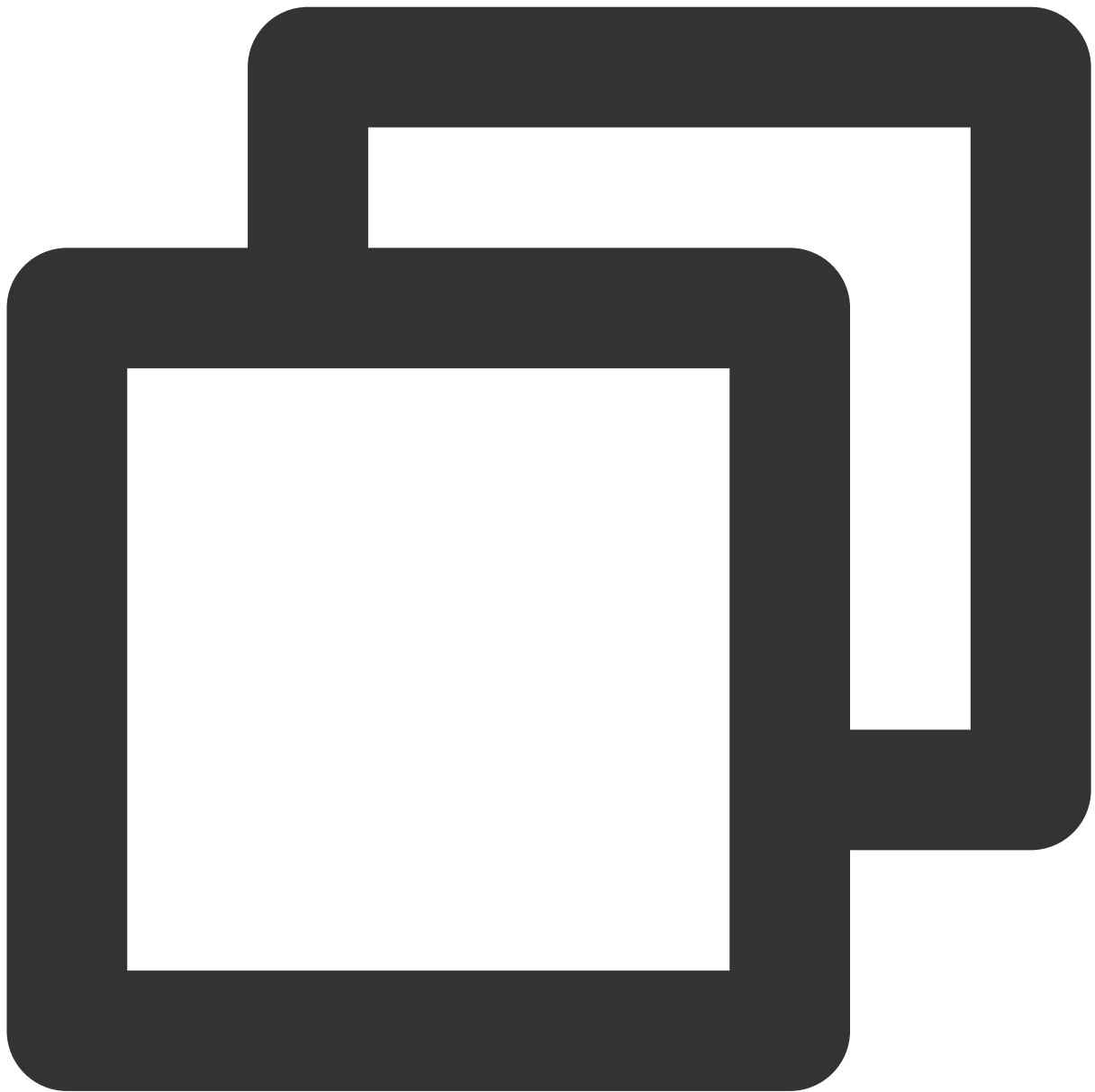
```
virtual void updateAudioQuality(TUIAudioQuality quality) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
quality	TUIAudioQuality	Audio Quality

### startPushLocalAudio

Start Pushing Local Audio.



```
virtual void startPushLocalAudio() = 0;
```

## **stopPushLocalAudio**

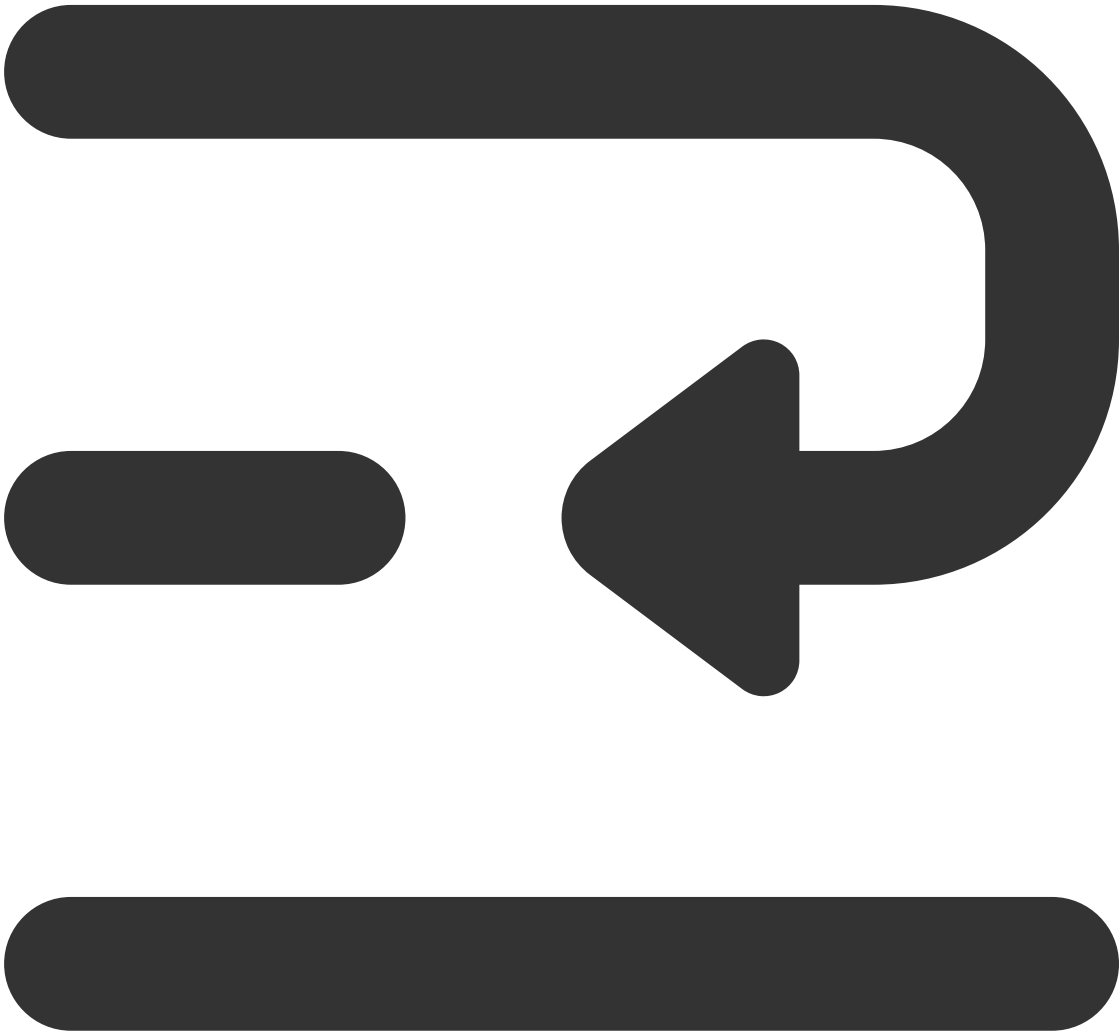
Stop Pushing Local Audio.



```
virtual void stopPushLocalAudio() = 0;
```

### **setRemoteVideoView**

Set Remote user Video Rendering View control.







```
virtual void setRemoteVideoView(const char* userId, TUIVideoStreamType streamType,
```

**The parameters are as follows:**

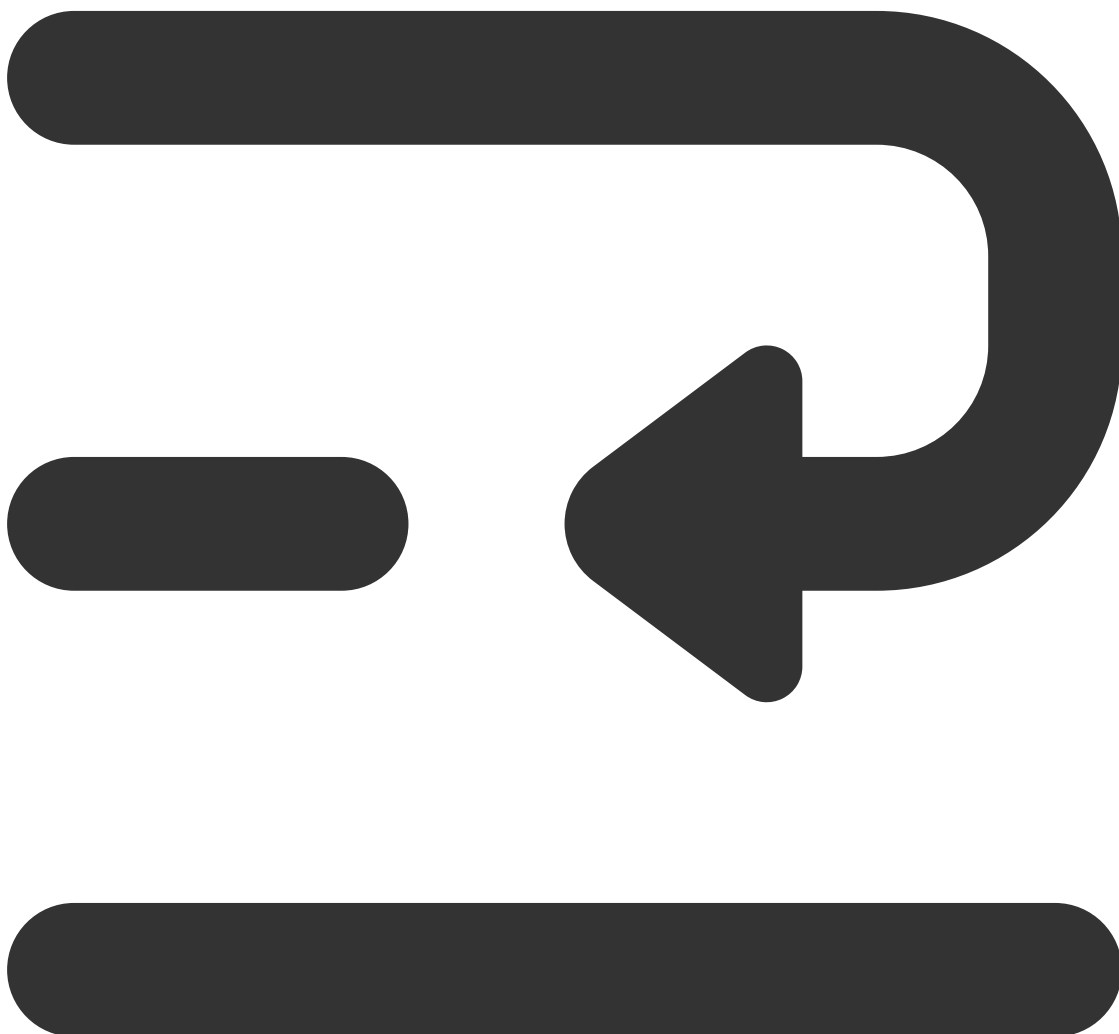
Parameter	Type	Meaning
userId	const char*	Remote User ID
streamType	TUIVideoStreamType	Stream Type
view	const	const TUIVideoView&

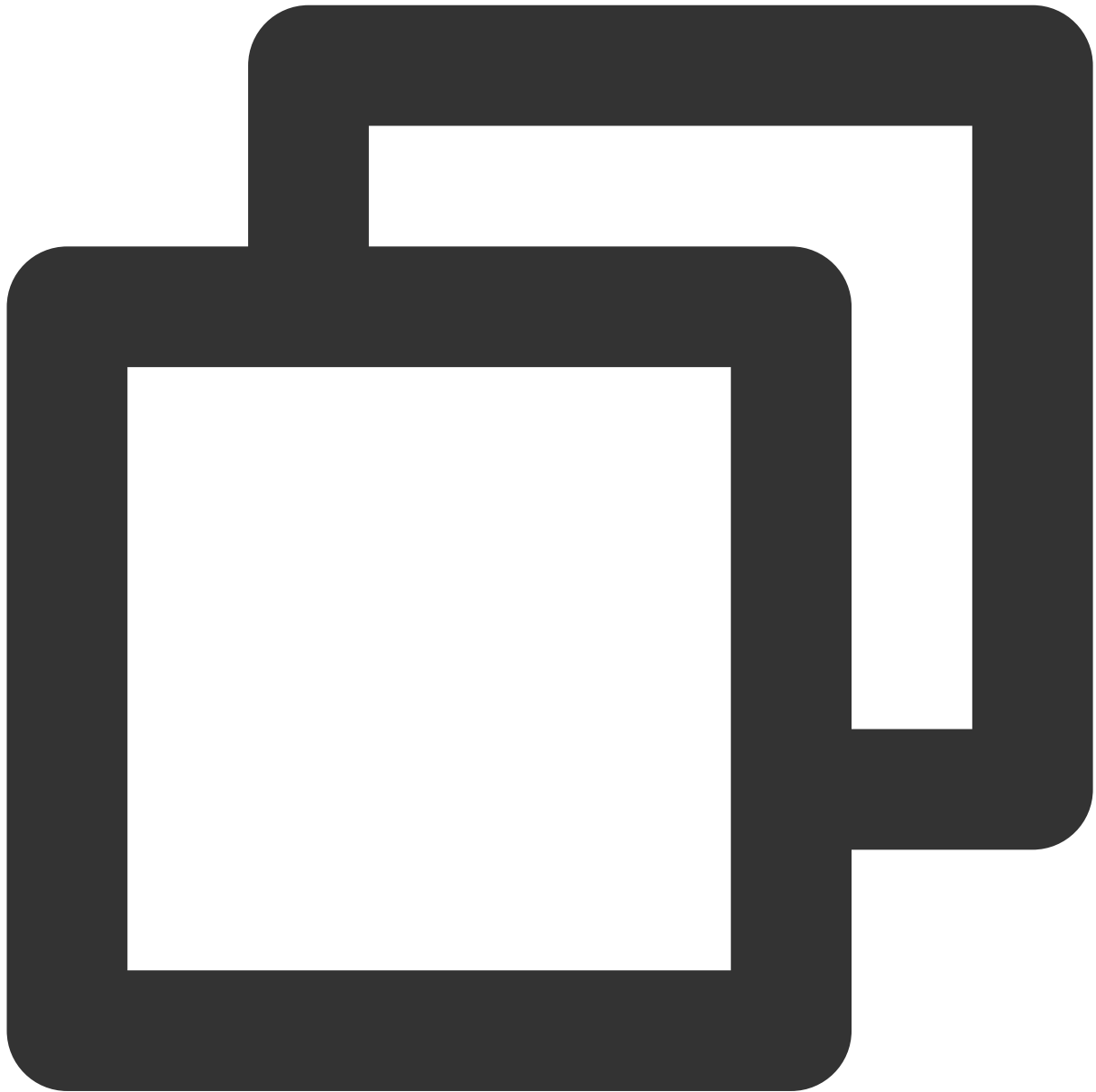
TUIVideoView&amp;

Rendering Window Handle

**startPlayRemoteVideo**

Start Playback Remote user Video streams





```
virtual void startPlayRemoteVideo(const char* userId, TUIVideoStreamType streamType
```

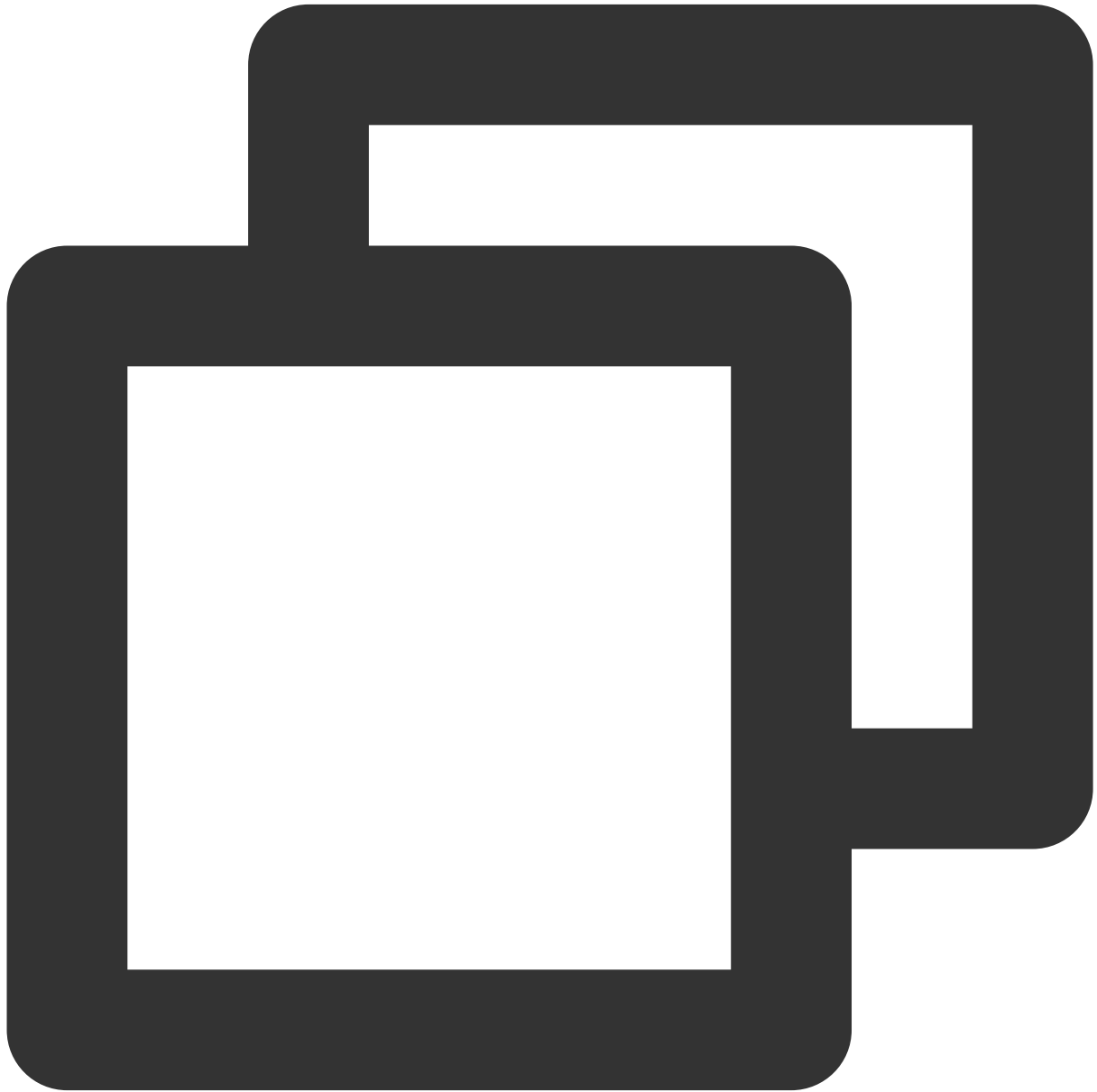
**The parameters are as follows:**

Parameter	Type	Meaning
userId	const char*	User ID
streamType	TUIVideoStreamType	User streams type
callback	TUIPlayCallback*	Callback of API, used to Notify the Success or Failure

of the API call

## stopPlayRemoteVideo

Stop Playback Remote user Video streams.



```
virtual void stopPlayRemoteVideo(const char* userId, TUIVideoStreamType streamType)
```

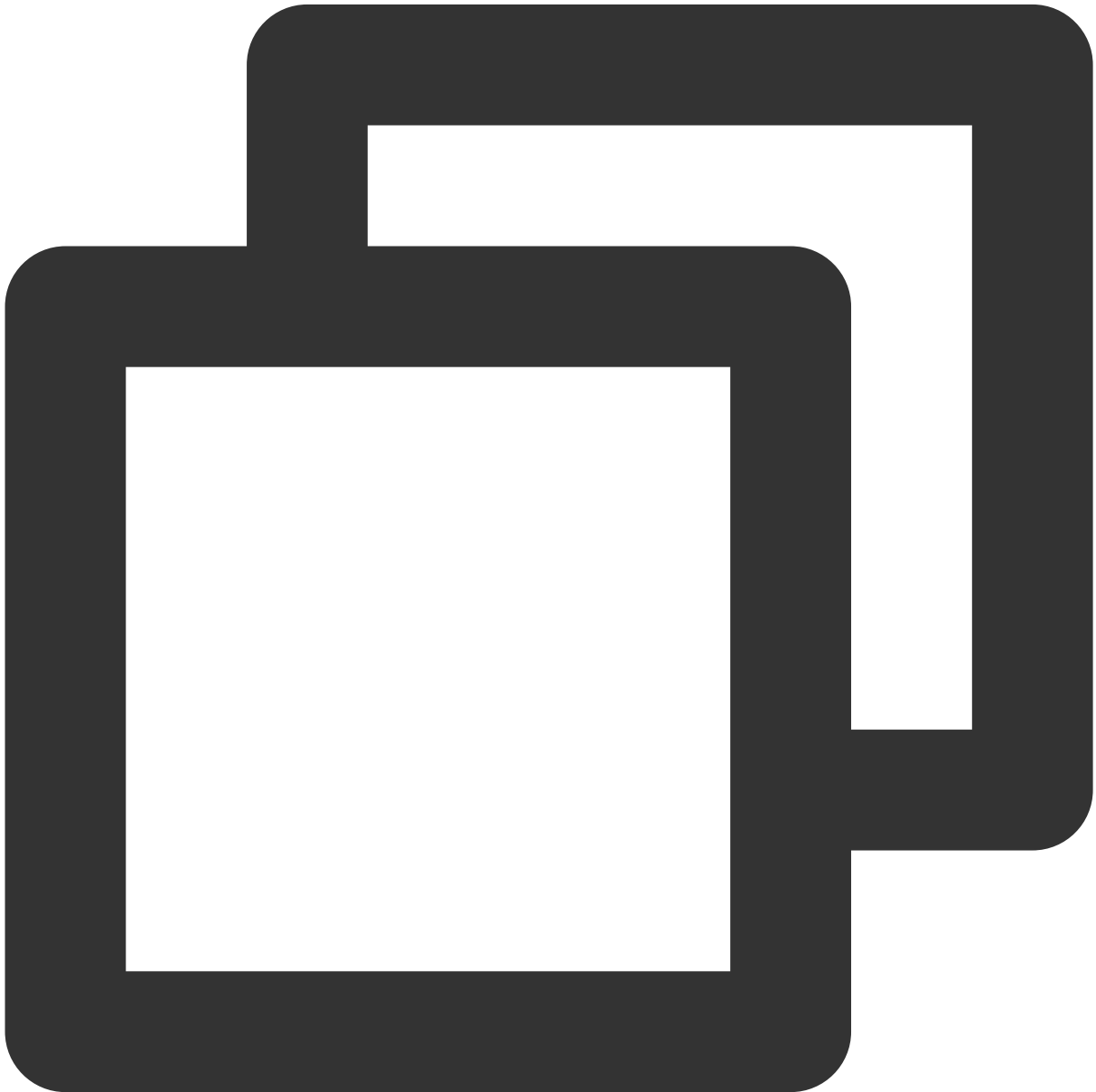
The parameters are as follows:

Parameter	Type	Meaning

userId	const char*	User ID
streamType	TUIVideoStreamType	User streams type

### **muteRemoteAudioStream**

Mute Remote user



```
virtual void muteRemoteAudioStream(const char* userId, bool isMute) = 0;
```

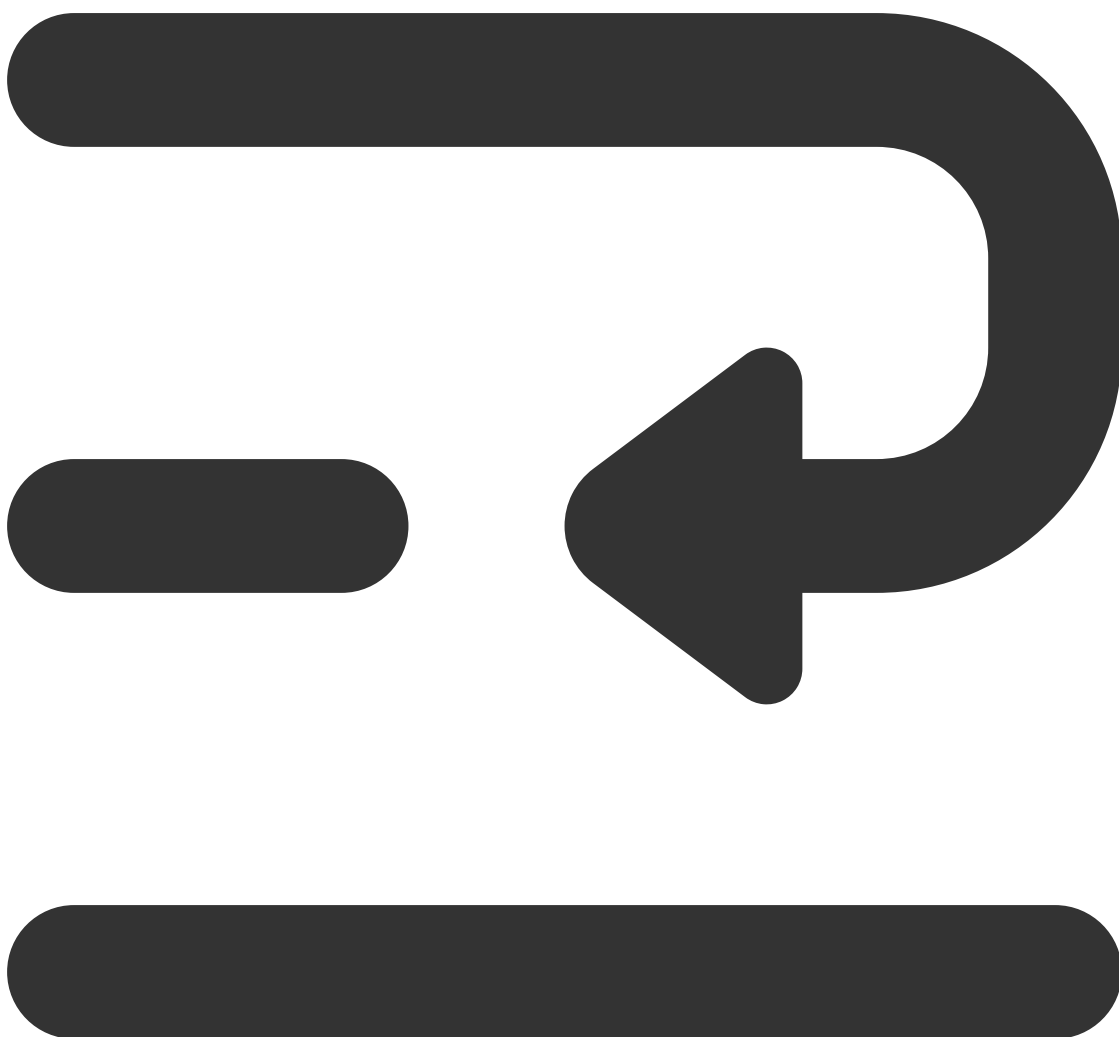
The parameters are as follows:

--	--	--

Parameter	Type	Meaning
userId	const char*	User ID
isMute	bool	Whether to mute

## getUserList

Get current User list in the room,.





```
virtual void getUserList(uint64_t nextSequence, TUIValueCallback<TUIUserListResult>
```

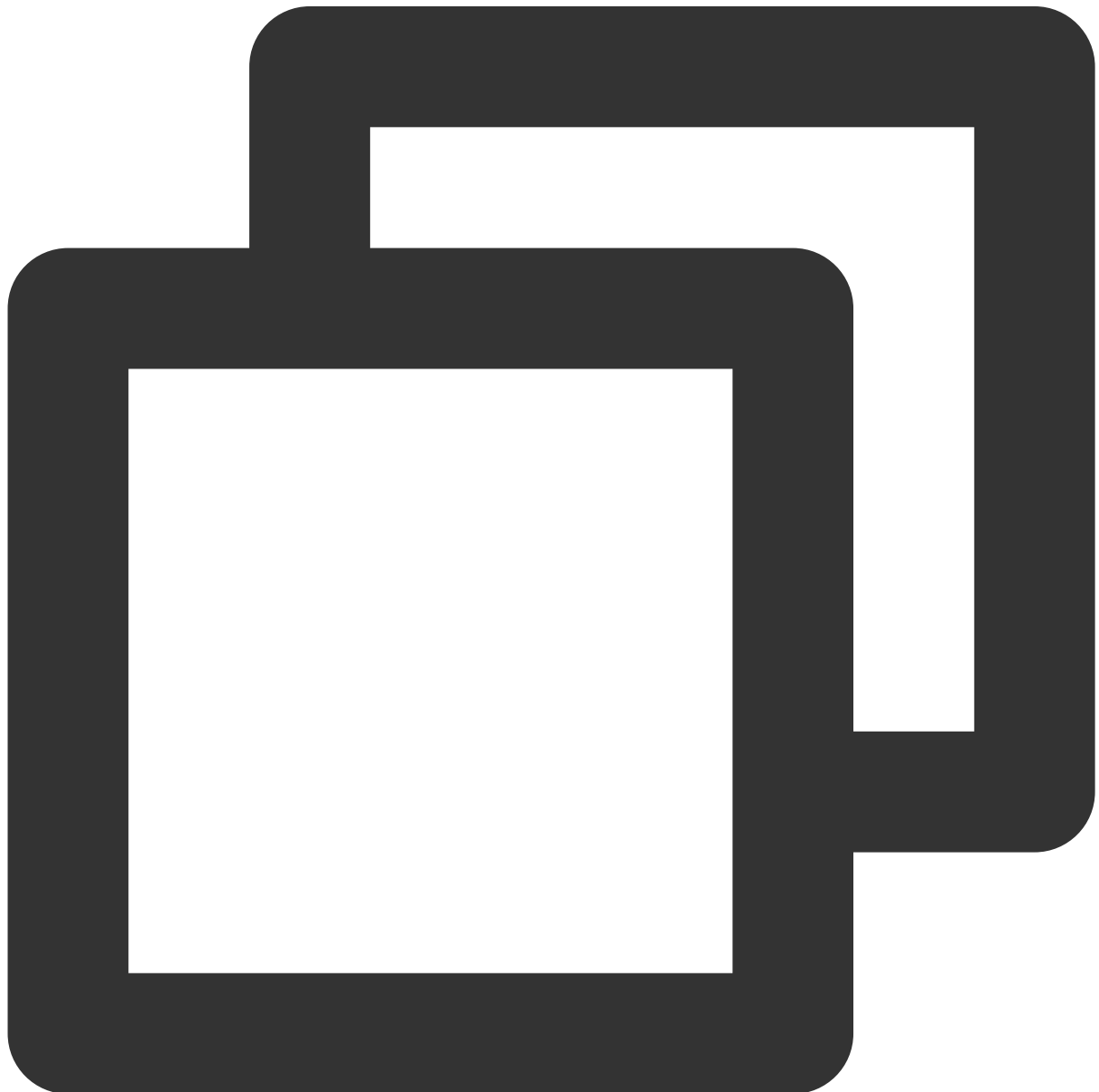
**The parameters are as follows:**

Parameter	Type	Meaning
nextSequence	uint64_t	Pagination Fetch Flag, fill in 0 for the first Fetch, if the nextSeq in the Callback is not 0, you need to do Pagination, pass in the nextSeq to Fetch again until the nextSeq in the Callback is 0

callback	TUIValueCallback<TUIUserListResult>*	Callback of API, used to Notify the Success or Failure of the API call
----------	--------------------------------------	--

## getUserInfo

Get User [Learn more](#).



```
virtual void getUserInfo(const char* userId, TUIValueCallback<TUIUserInfo>* callback
```

The parameters are as follows:

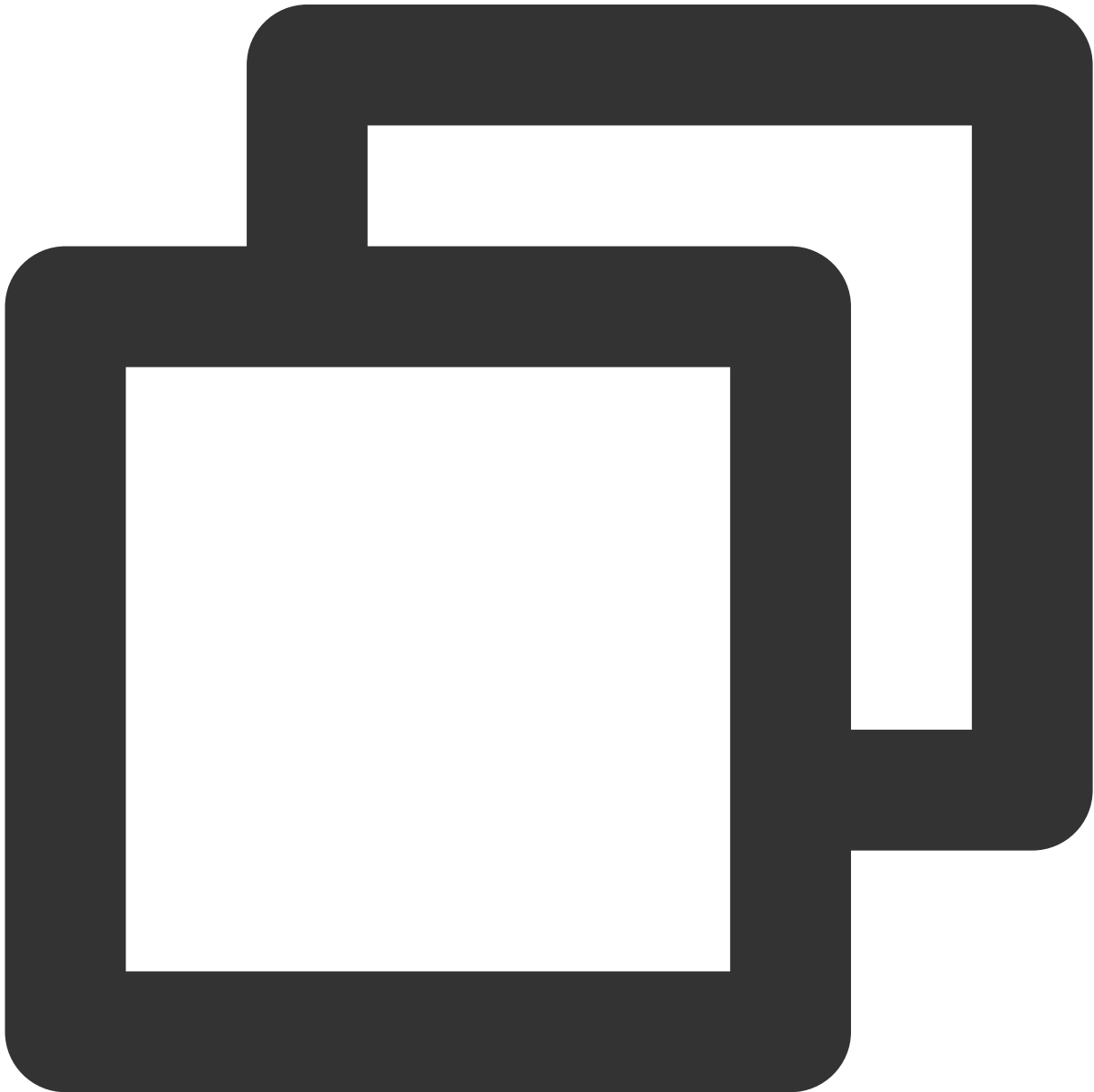
Parameter	Type	Meaning
-----------	------	---------



userId	const char*	User ID
callback	TUIValueCallback<TUIUserInfo>*	Callback of API, used to Notify the Success or Failure of the API call

changeUserRole

Change user Role, only Administrator or Group owner can call



```
virtual void changeUserRole(const char* userId, TUIRole role, TUICallback* callback
```

The parameters are as follows:

Parameter	Type	Meaning
userId	const char*	User ID
role	TUIRole	User Role
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## kickRemoteUserOutOfRoom

Kick user out of the room.



```
virtual void kickRemoteUserOutOfRoom(const char* userId, TUICallback* callback) = 0
```

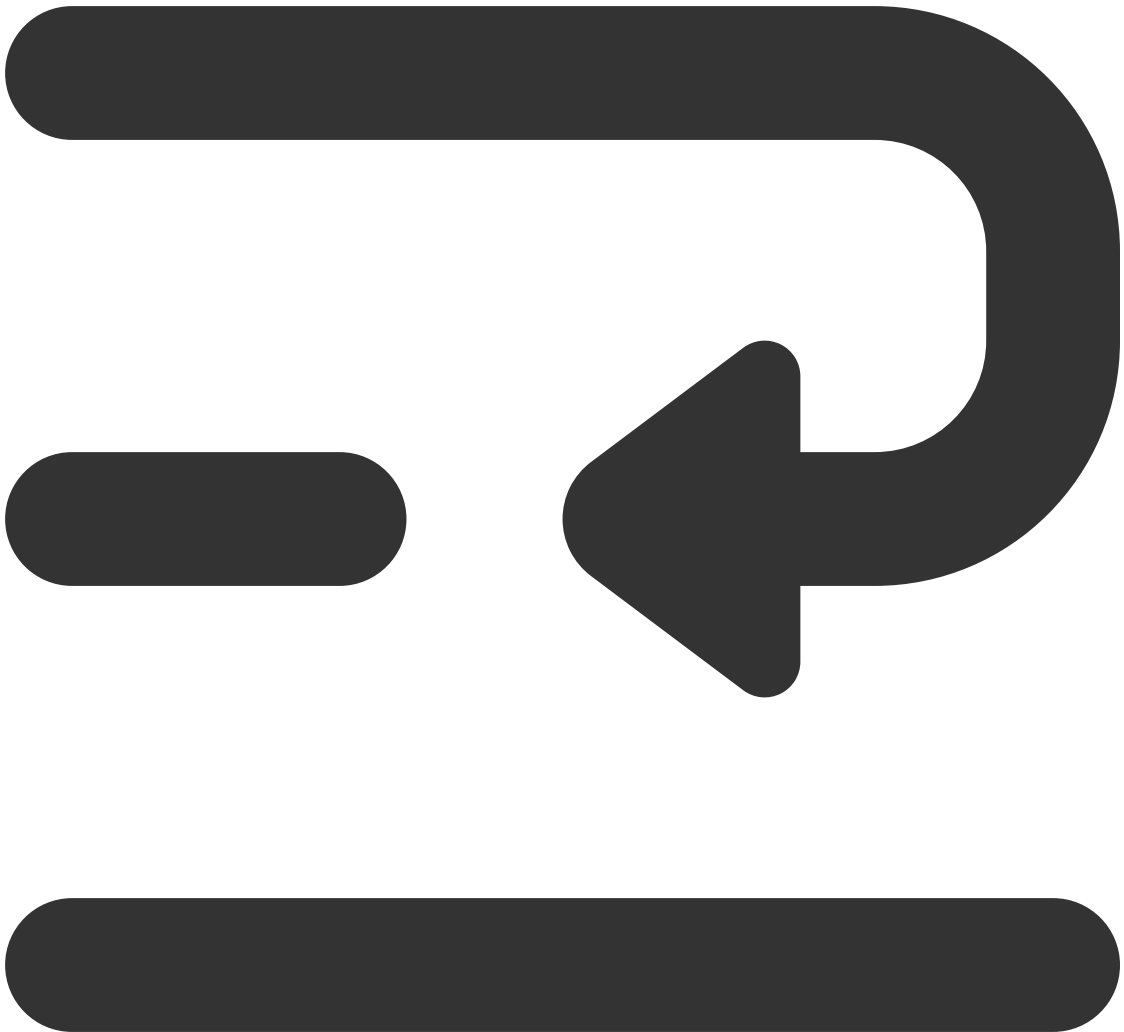
**The parameters are as follows:**

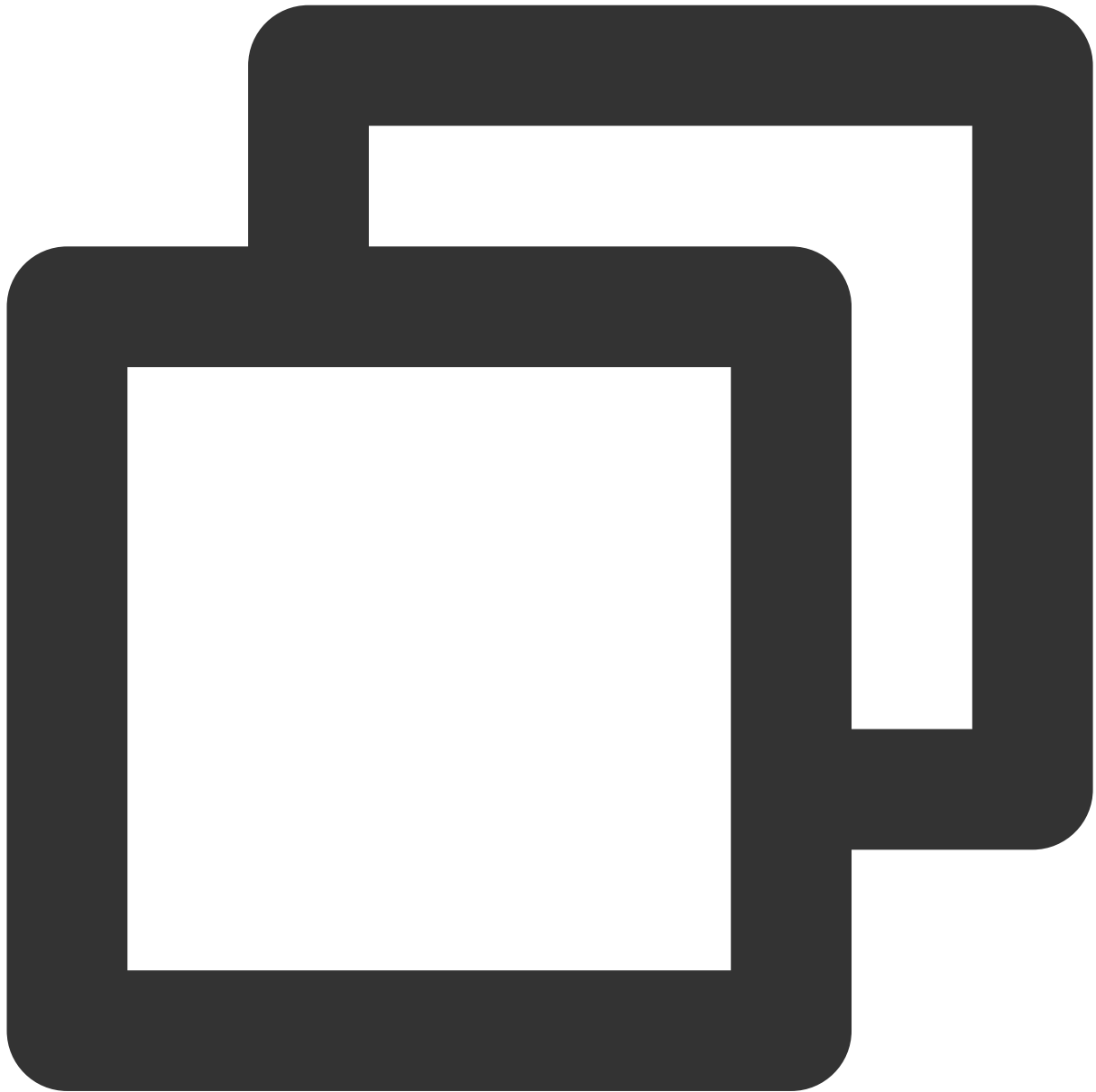
Parameter	Type	Meaning
userId	const char*	User ID
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## **disableDeviceForAllUserByAdmin**

Control the permission status of whether all users in the current room can open Audio and Video streams capturing devices, such as: Prohibit All from turning on the mic, Prohibit All from turning on the Camera, Prohibit All from turning on Screen Sharing

(Currently only available in Meeting Scenario, and only Administrator or Group Owner can invoke).





```
virtual void disableDeviceForAllUserByAdmin(TUIMediaDevice device, bool isDisable,
```

**The parameters are as follows:**

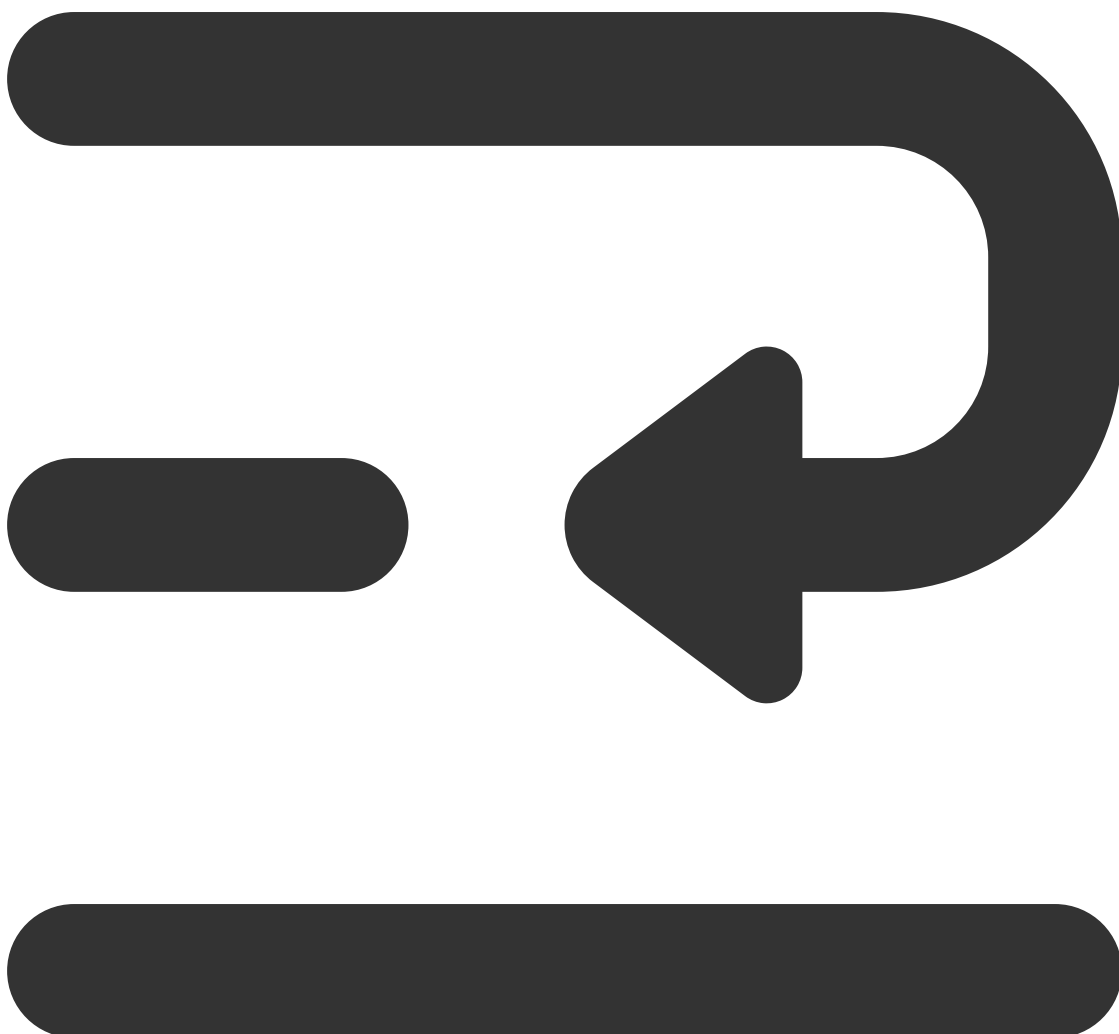
Parameter	Type	Meaning
device	TUIMediaDevice	Device
isDisable	bool	Whether to Disable
callback	TUICallback*	Callback of API, used to Notify the Success or Failure

		of the API call
--	--	-----------------

## openRemoteDeviceByAdmin

Request remote users to open media devices

(Currently only available in the conference scene, and only administrators or group owners can call).





```
virtual TUIRequest openRemoteDeviceByAdmin(const char* userId, TUIMediaDevice device, int timeout)
```

**The parameters are as follows:**

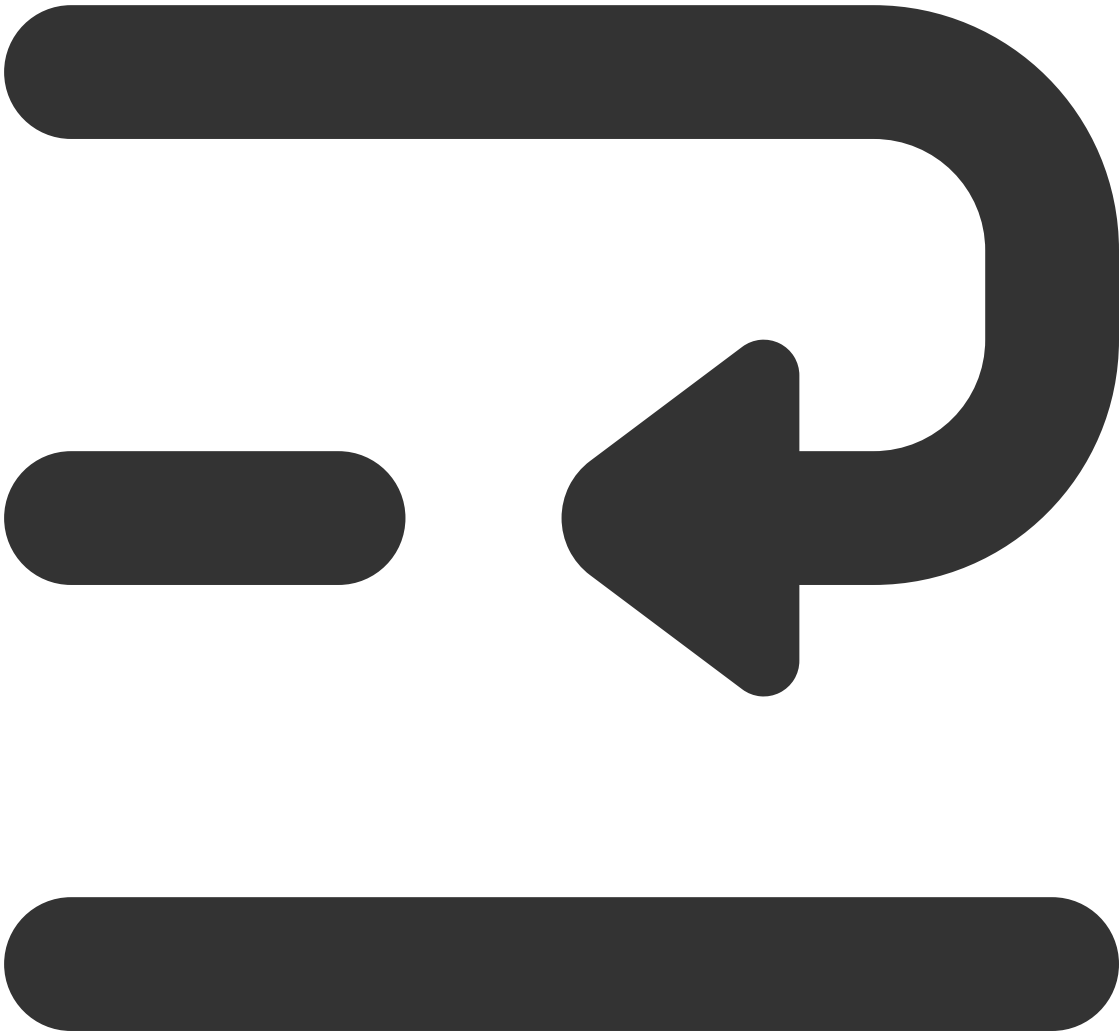
Parameter	Type	Meaning
userId	const char*	User ID
device	TUIMediaDevice	Device
timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout

		detection and will not trigger timeout Callback
callback	TUIRequestCallback*	Callback of API, used to Notify the Success or Failure of the API call

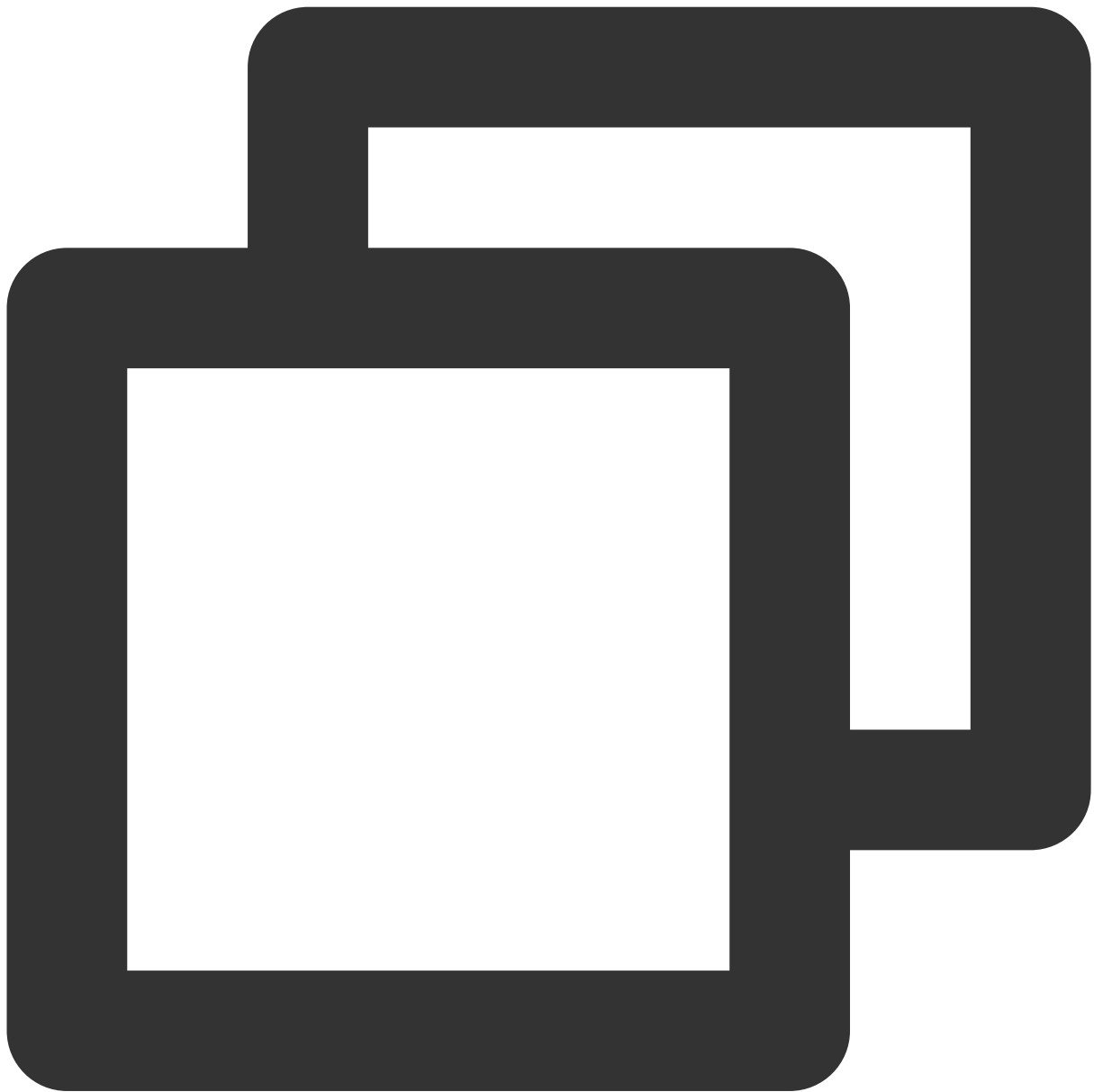
closeRemoteDeviceByAdmin

Close remote users' media devices

(Currently only available in the conference scene, and only administrators or group owners can call).







```
virtual void closeRemoteDeviceByAdmin(const char* userId, TUIMediaDevice device, TU
```

**The parameters are as follows:**

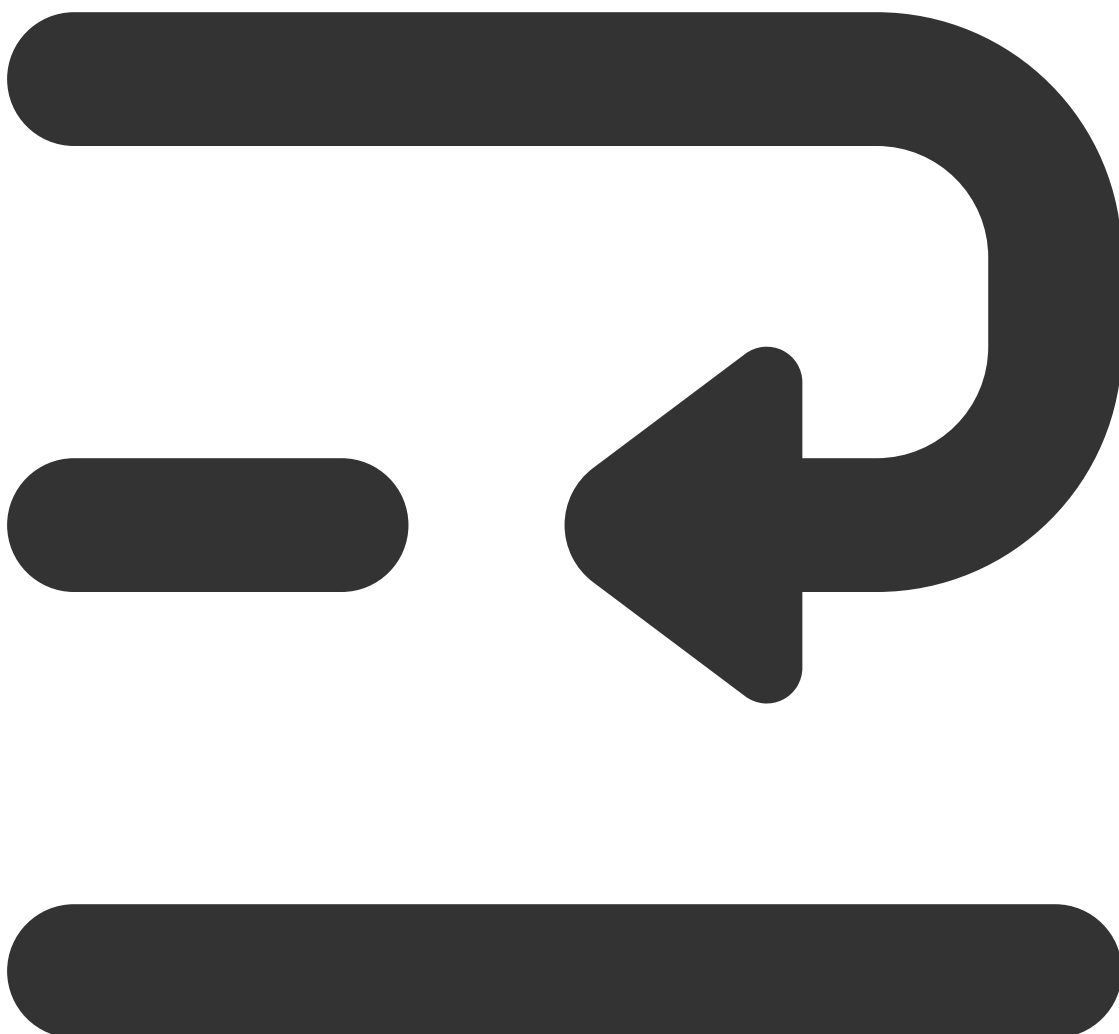
Parameter	Type	Meaning
userId	const char*	User ID
device	TUIMediaDevice	Device
callback	TUICallback*	Callback of API, used to Notify the Success or Failure

		of the API call
--	--	-----------------

## **applyToAdminToOpenLocalDevice**

Request to open local media devices

(Currently only available in the conference scene, and only ordinary members can call).





```
virtual TUIRequest applyToAdminToOpenLocalDevice(TUIMediaDevice device, int timeout
```

**The parameters are as follows:**

Parameter	Type	Meaning
device	TUIMediaDevice	Device
timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout detection and will not trigger timeout Callback

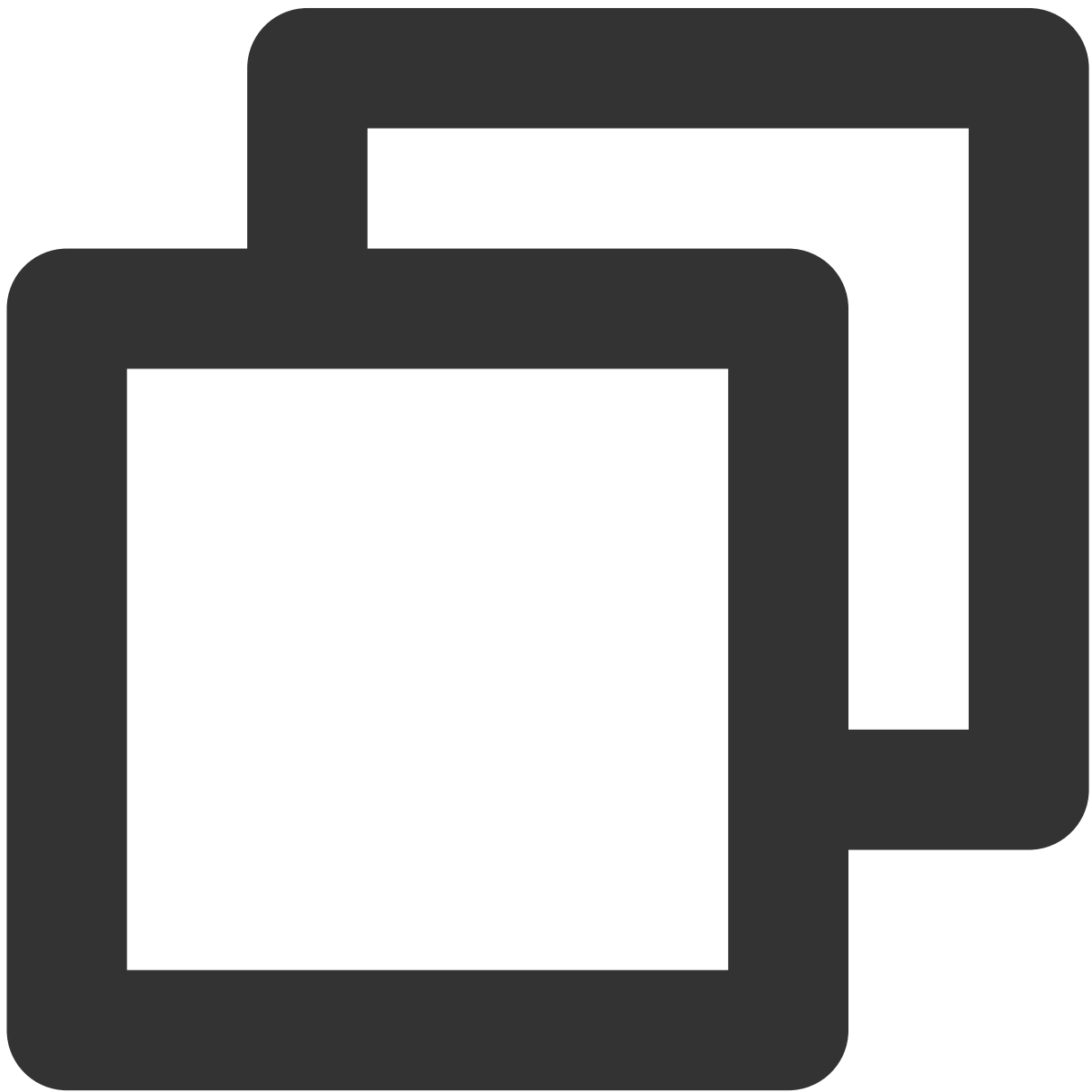
callback	TUIRequestCallback*	Callback of API, used to Notify the Success or Failure of the API call
----------	---------------------	--

## setMaxSeatCount

Set Maximum number of seats, only supported when entering the room and creating the room

When roomType is RoomType.CONFERENCE (Education and Conference scene), maxSeatCount value is not limited;

When roomType is RoomType.LIVE\_ROOM (Live broadcast scene), maxSeatCount is limited to 16;



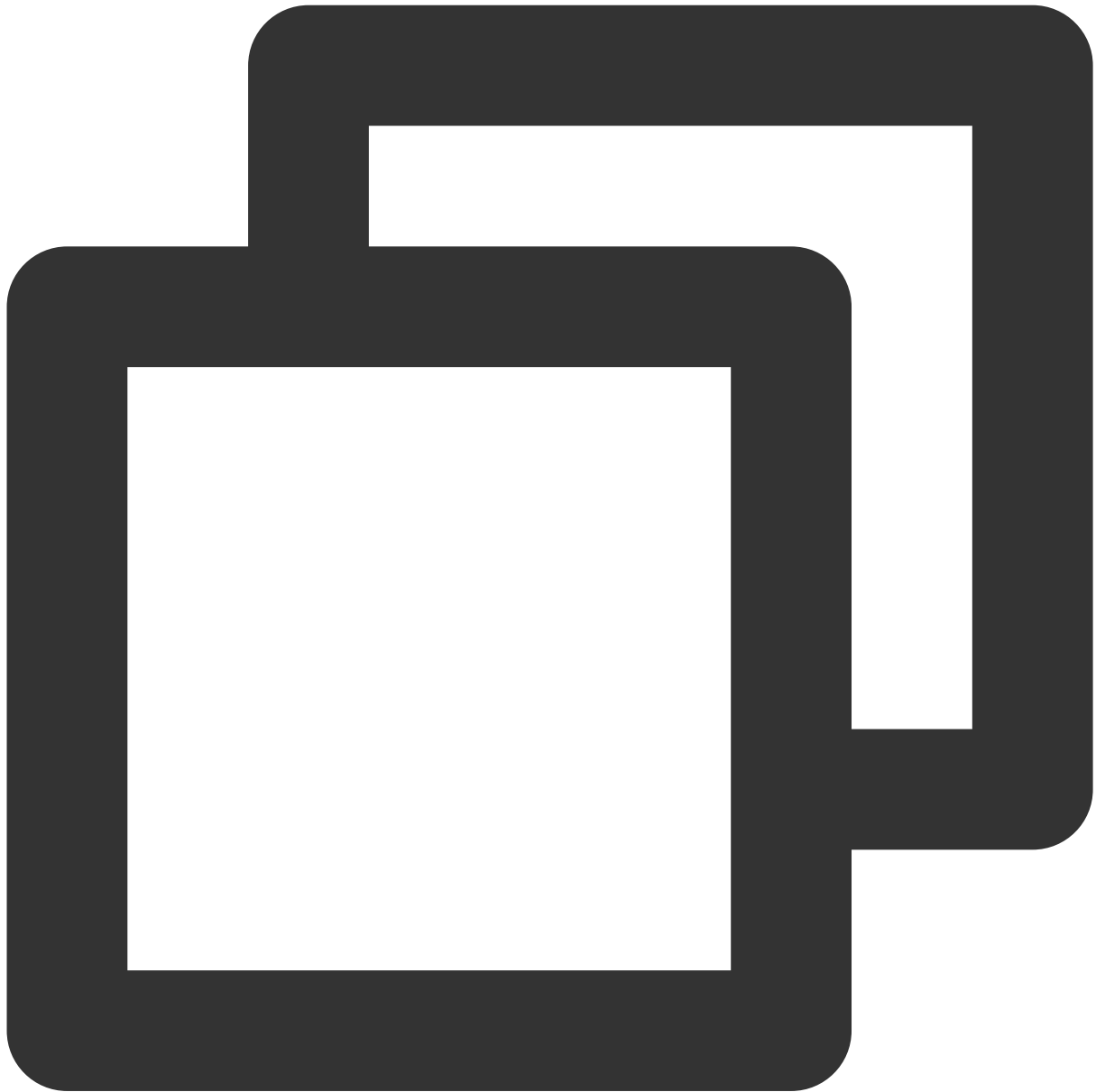
```
virtual void setMaxSeatCount(uint32_t maxSeatCount, TUICallback* callback) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
maxSeatCount	uint32_t	Maximum number of seats
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## getSeatList

Get seat list



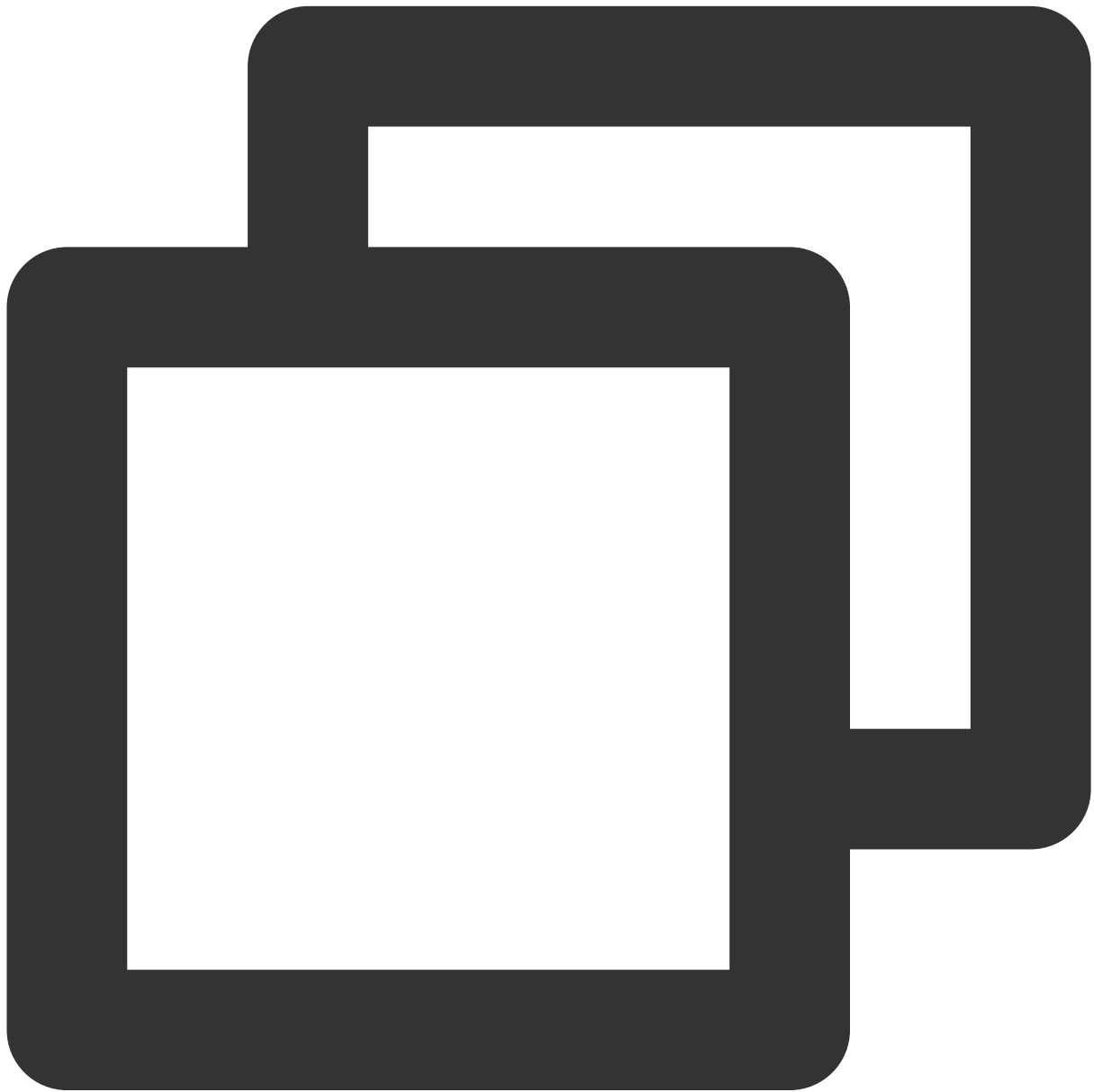
```
virtual void getSeatList(TUICollectionCallback<TUISeatInfo>* callback) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
callback	TUICollectionCallback<TUISeatInfo>*	Callback of API, used to Notify the Success or Failure of the API call

## lockSeatByAdmin

Lock seat.



```
virtual void lockSeatByAdmin(int seatIndex, const TUISeatLockParams& lockParams, TU
```

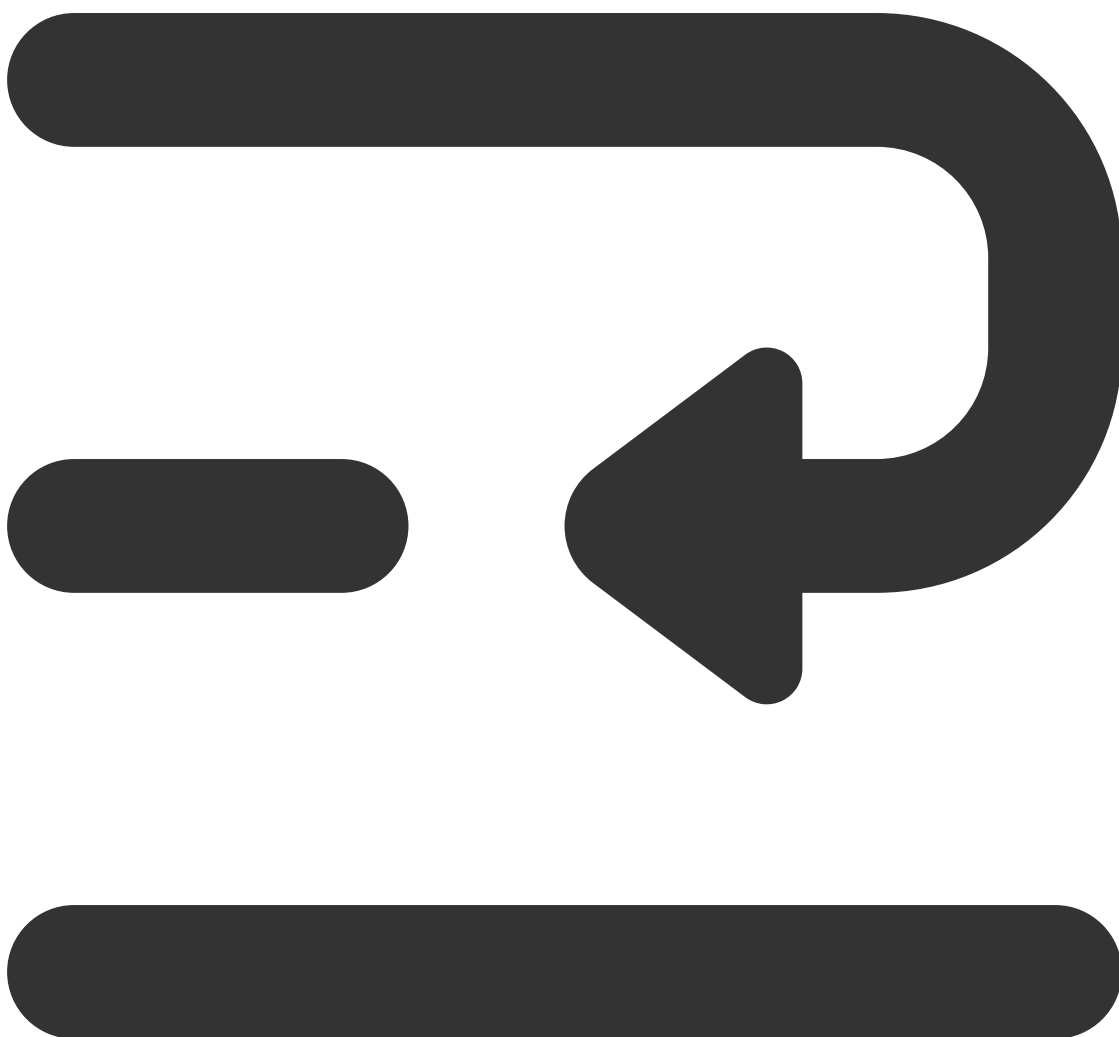
The parameters are as follows:

Parameter	Type	Meaning
seatIndex	int	Seat number
lockParams	TUISeatLockParams	Lock microphone parameter

callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call
----------	--------------	--

## takeSeat

Get seat list.







```
virtual TUIRequest takeSeat(int seatIndex, int timeout, TUIRequestCallback* callback
```

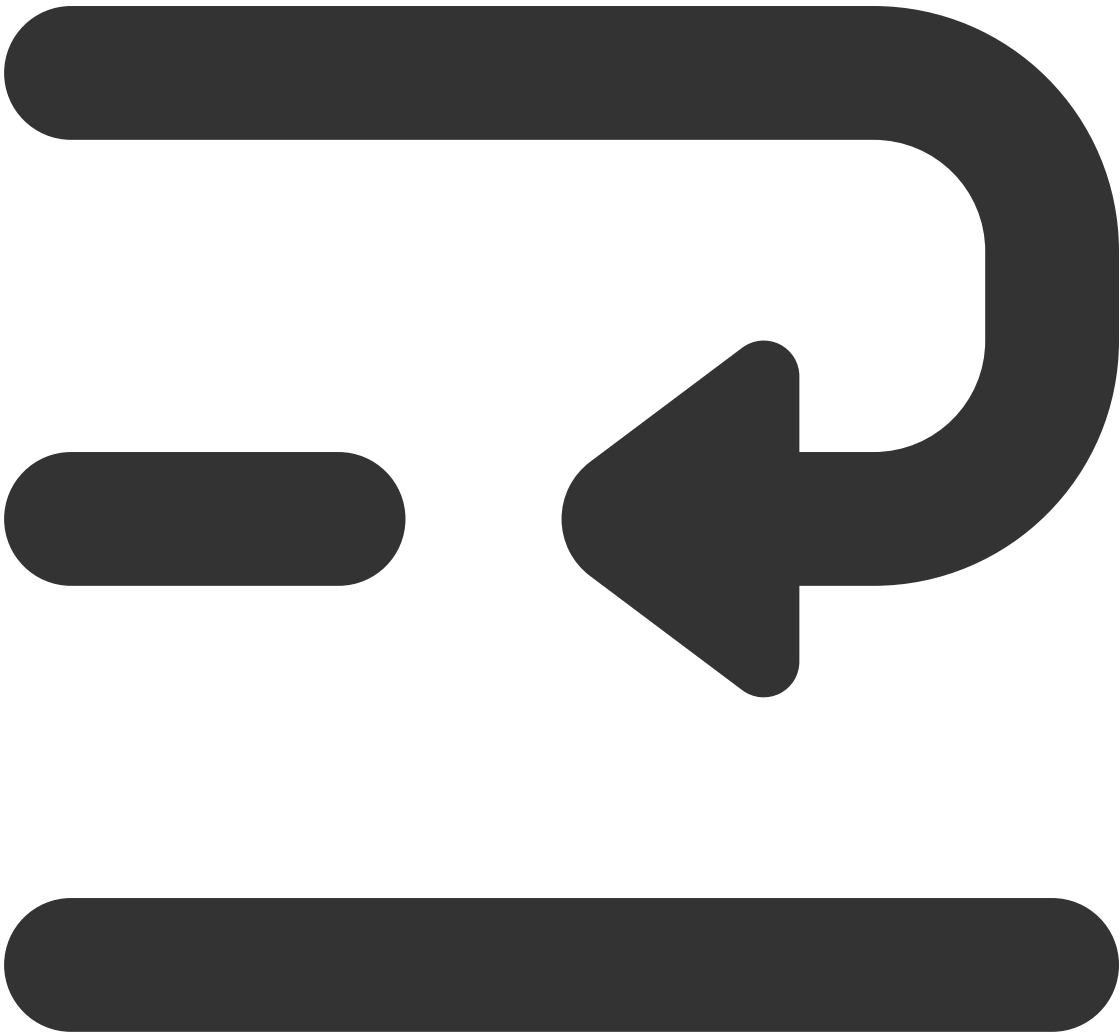
**The parameters are as follows:**

Parameter	Type	Meaning
seatIndex	int	Seat number
timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout detection and will not trigger timeout Callback

callback	TUIRequestCallback*	Callback of API, used to Notify the Success or Failure of the API call
----------	---------------------	--

leaveSeat

Local off microphone.





```
virtual void leaveSeat(TUICallback* callback) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

**takeUserOnSeatByAdmin**

Host/Administrator Invite User on microphone.





```
virtual TUIRequest takeUserOnSeatByAdmin(int seatIndex, const char* userId, int tim
```

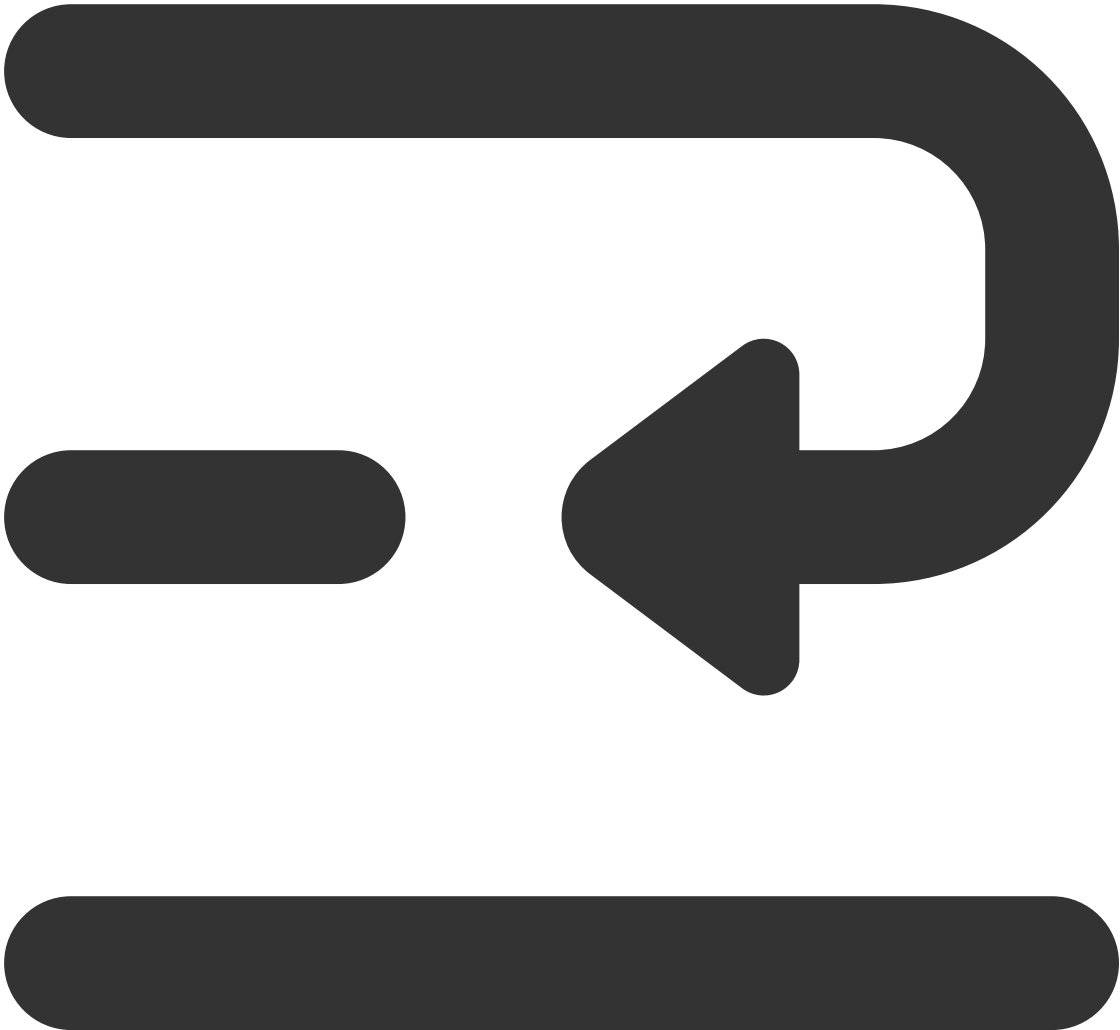
**The parameters are as follows:**

Parameter	Type	Meaning
seatIndex	int	Seat number
userId	const char*	User ID
timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout

		detection and will not trigger timeout Callback
callback	TUIRequestCallback*	Callback of API, used to Notify the Success or Failure of the API call

**kickUserOffSeatByAdmin**

Host/Administrator Take User off microphone.





```
virtual void kickUserOffSeatByAdmin(int seatIndex, const char* userId, TUICallback*
```

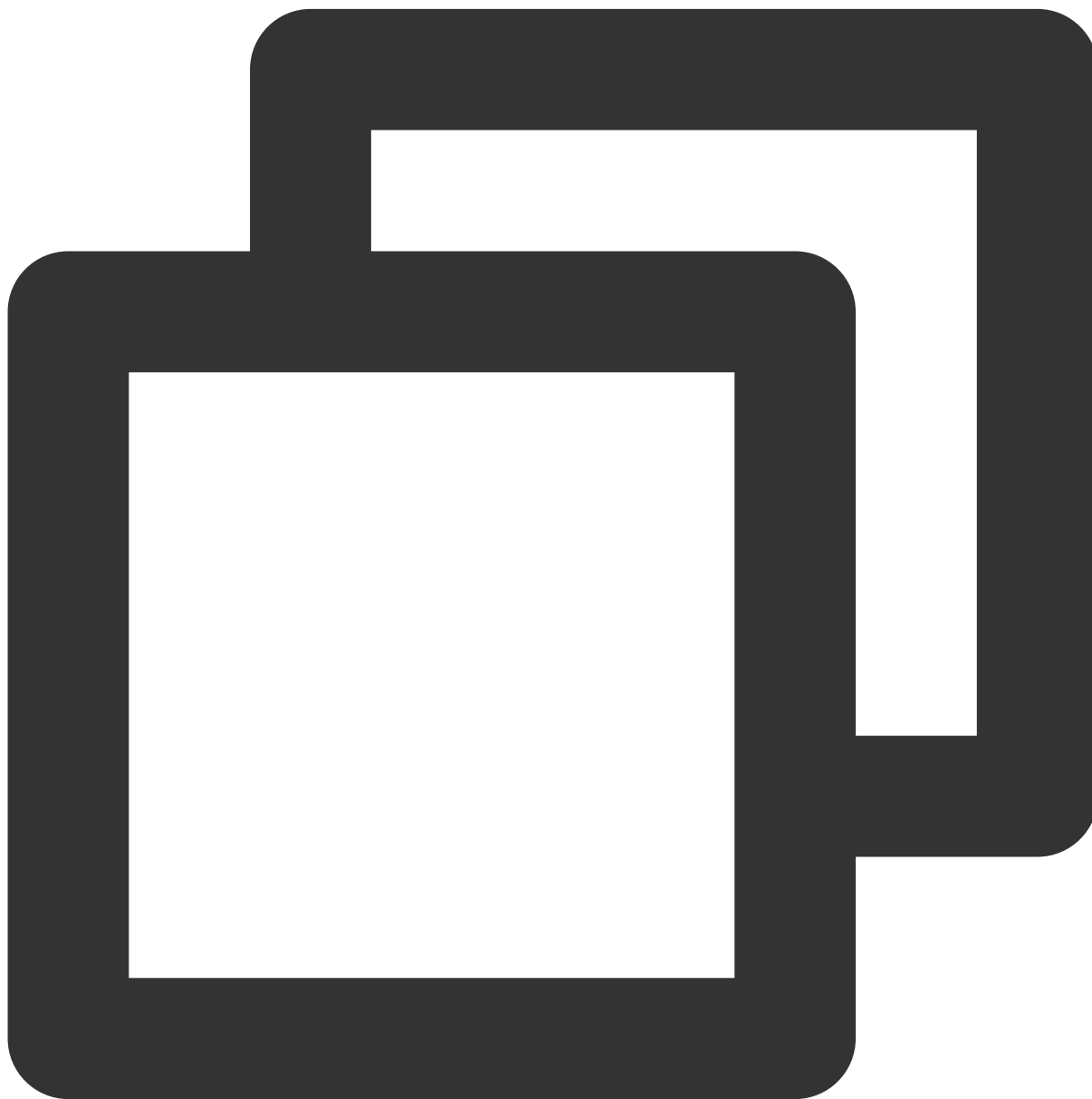
**The parameters are as follows:**

Parameter	Type	Meaning
seatIndex	int	Seat number
userId	const char*	User ID
callback	TUICallback*	Callback of API, used to Notify the Success or Failure

of the API call

## sendMessage

Send Text message.



```
virtual void sendMessage(const char* message, TUICallback* callback) = 0;
```

The parameters are as follows:

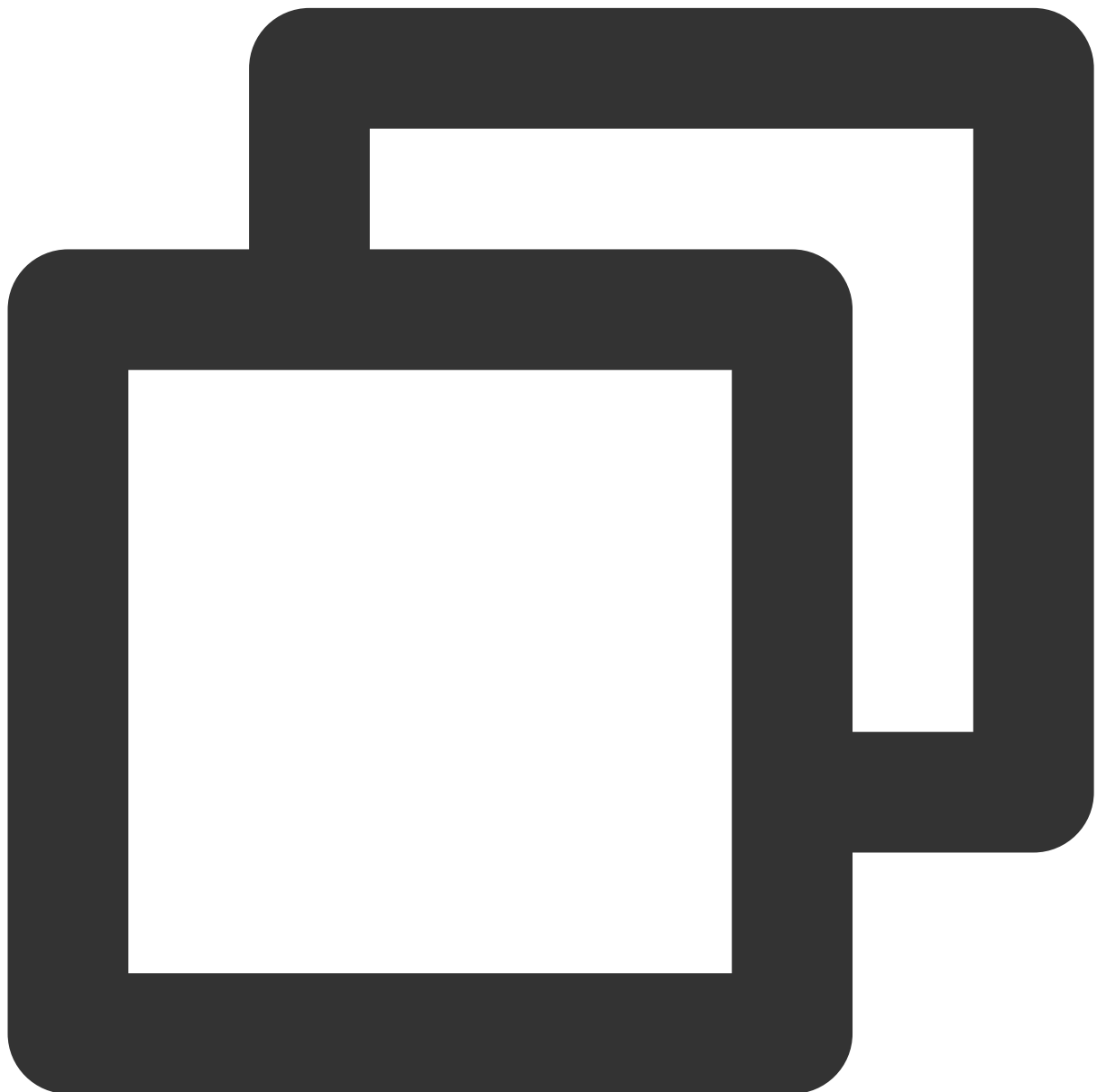
Parameter	Type	Meaning



message	const char*	Text message Content
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## sendCustomMessage

Send Custom message.



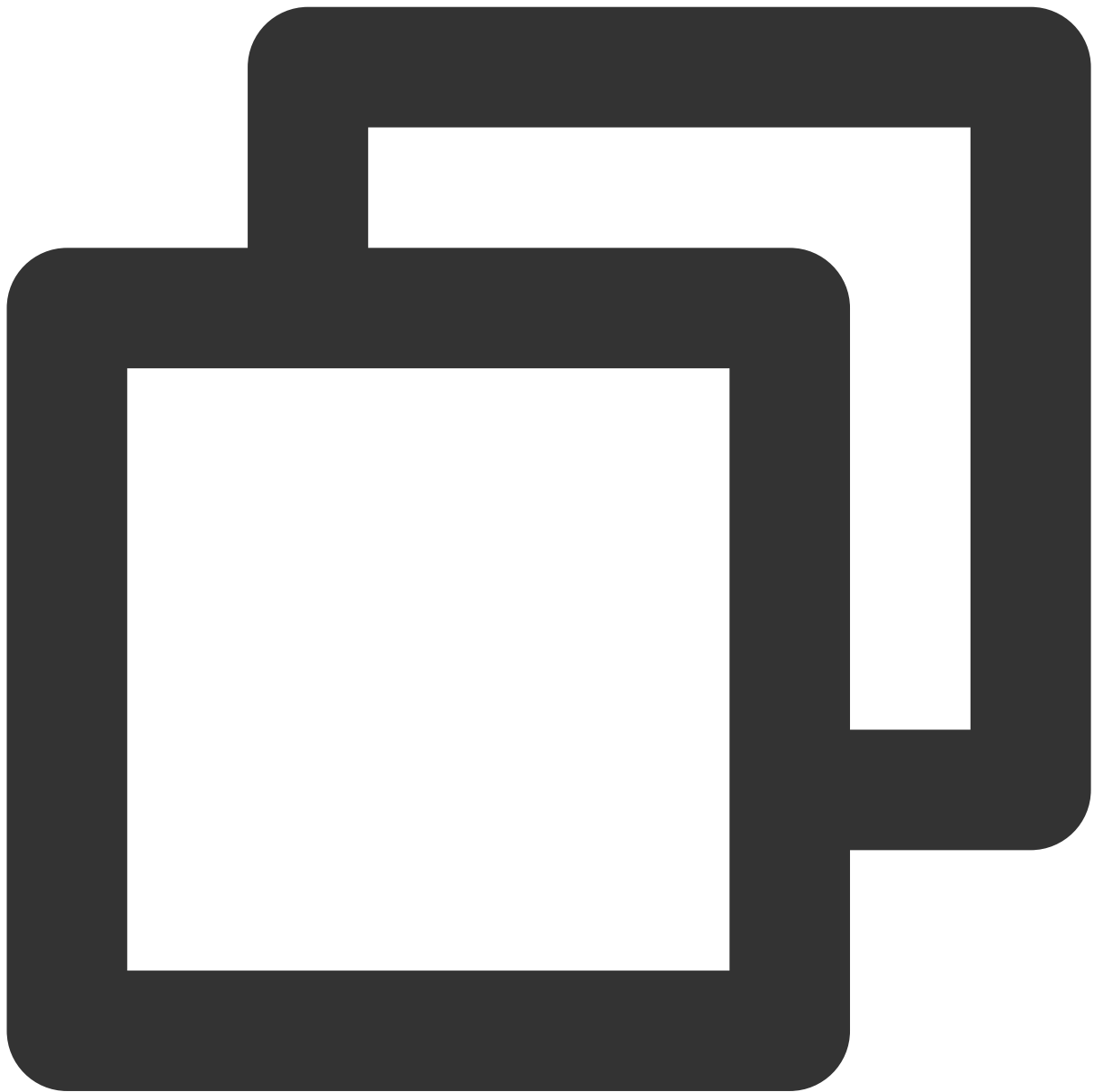
```
virtual void sendCustomMessage(const char* message, TUICallback* callback) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
message	const char*	Custom message Content
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

### **disableSendingMessageByAdmin**

Disable Remote user's Text message sending Ability (only Administrator or Group owner can call).



```
virtual void disableSendingMessageByAdmin(const char* userId, bool isDisable, TUICa
```

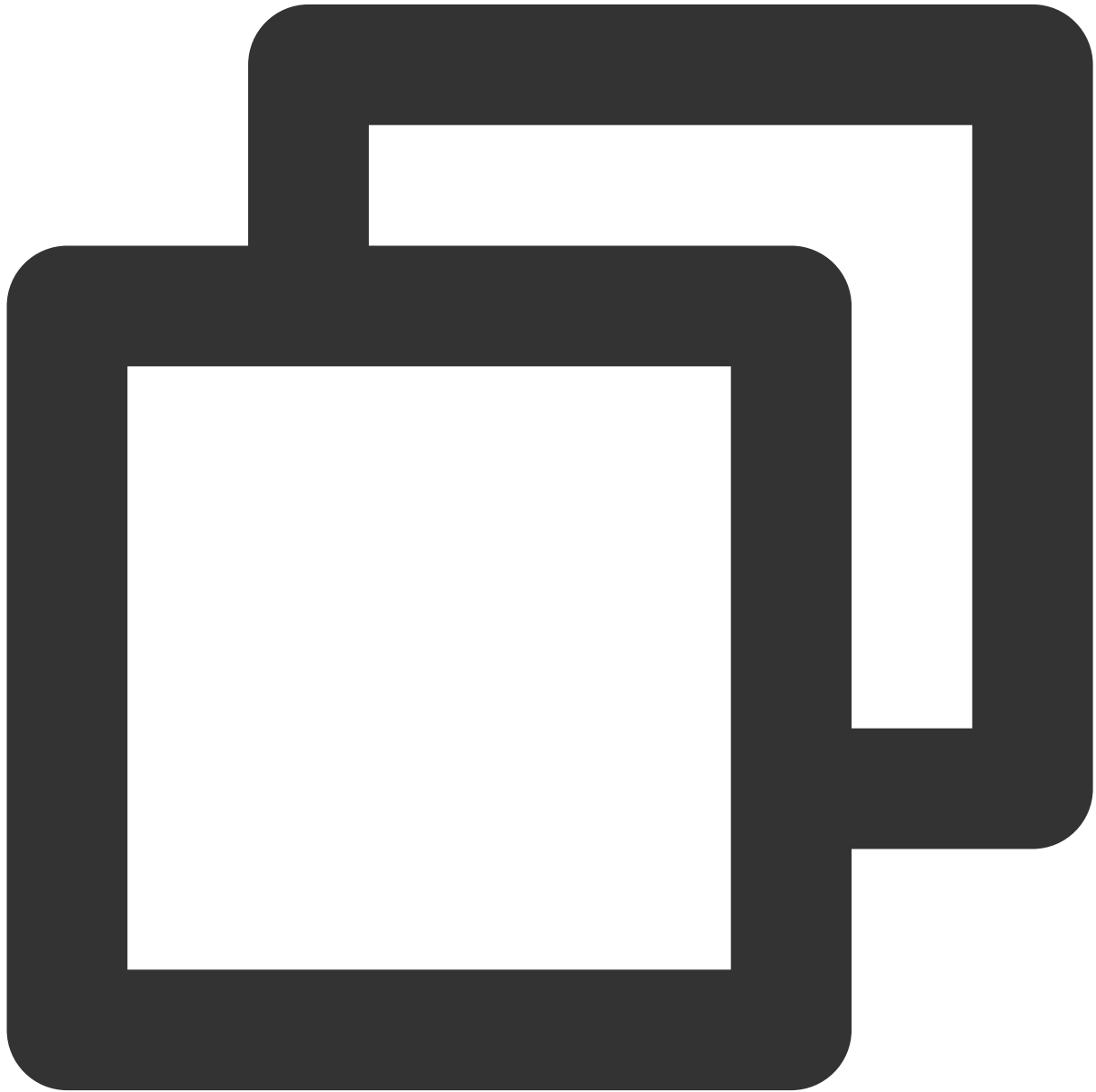
**The parameters are as follows:**

Parameter	Type	Meaning
userId	const char*	User ID
isDisable	bool	Whether to Disable
callback	TUICallback*	Callback of API, used to Notify the Success or Failure

		of the API call
--	--	-----------------

## disableSendingMessageForAllUser

Disable all users' Text message sending Ability (only Administrator or Group owner can call)



```
virtual void disableSendingMessageForAllUser(bool isDisable, TUICallback* callback)
```

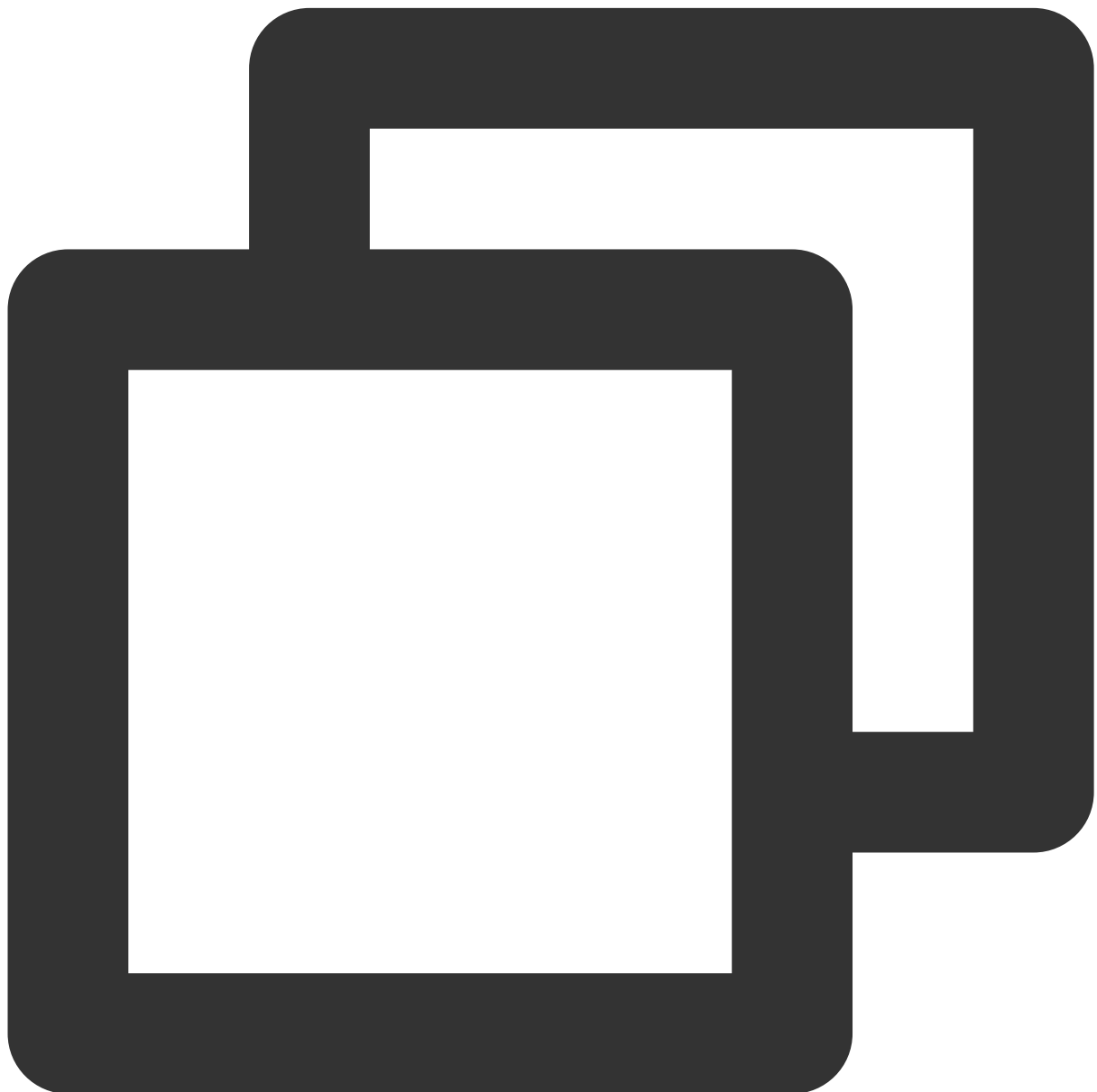
The parameters are as follows:

Parameter	Type	Meaning

isDisable	bool	Whether to Disable
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## cancelRequest

Cancel sent Request.



```
virtual void cancelRequest(const char* requestId, TUICallback* callback) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
requestId	const char*	Request ID
callback	TUICallback*	Callback of API, used to Notify the Success or Failure of the API call

## responseRemoteRequest

Reply to Remote user's Request.



```
virtual void responseRemoteRequest(const char* requestId, bool agree, TUICallback*
```

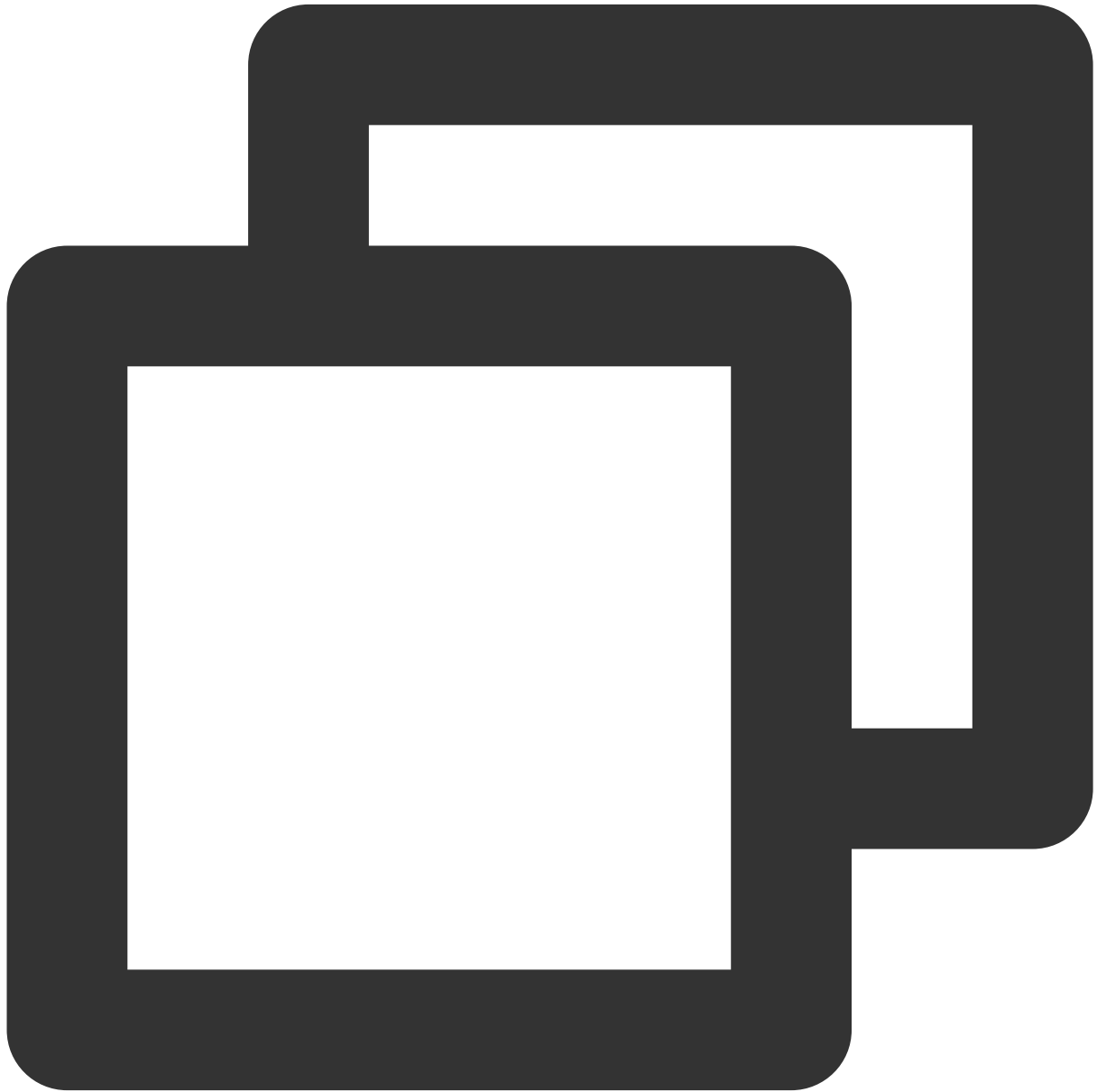
**The parameters are as follows:**

Parameter	Type	Meaning
requestId	const char*	Request ID
agree	bool	Whether to agree
callback	TUICallback*	Callback of API, used to Notify the Success or Failure

		of the API call
--	--	-----------------

## getTRTCCloud

Get TRTCCloud Instance.



```
virtual void* getTRTCCloud() = 0;
```

## getDeviceManager

Get deviceManager.





```
virtual liteav::ITXDeviceManager* getDeviceManager() = 0;
```

## **getAudioEffectManager**

Get Audio management.



```
virtual liteav::ITXAudioEffectManager* getAudioEffectManager() = 0;
```

# TUIRoomObserver

Last updated : 2023-10-30 11:01:03

## TUIRoomObserver Callback Event Details

The TUIRoomObserver class is the corresponding Callback Event class for TUIRoomEngine. You can listen to the Callback Events you are interested in through this Callback Interface.

### **onError**

Error Event Callback



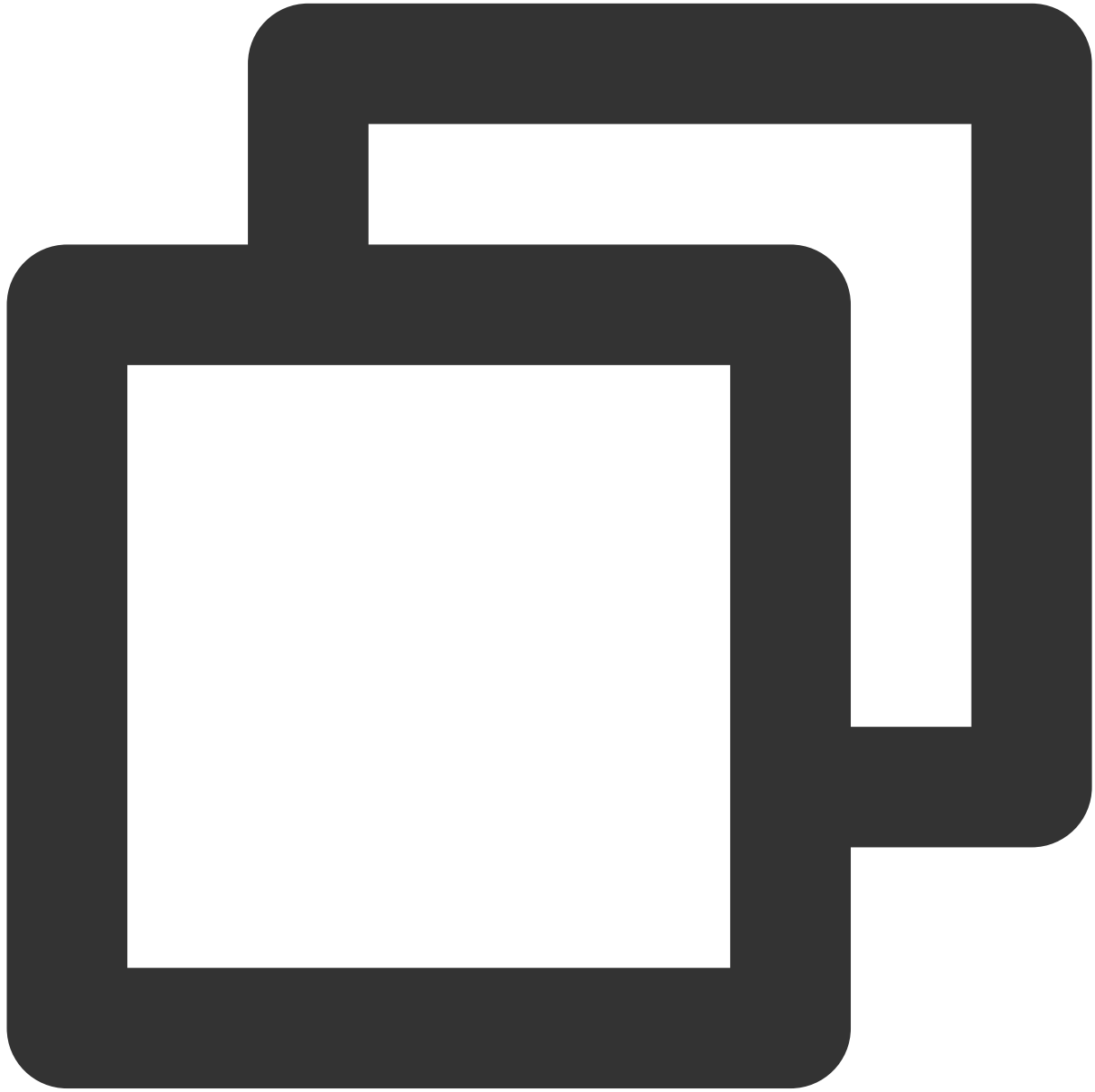
```
virtual void onError(TUIError errorCode, const char* message) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
errorCode	TUIError	Error Code
message	const char*	Error Message

**onKickedOffline**

Other terminals login and get kicked off event.



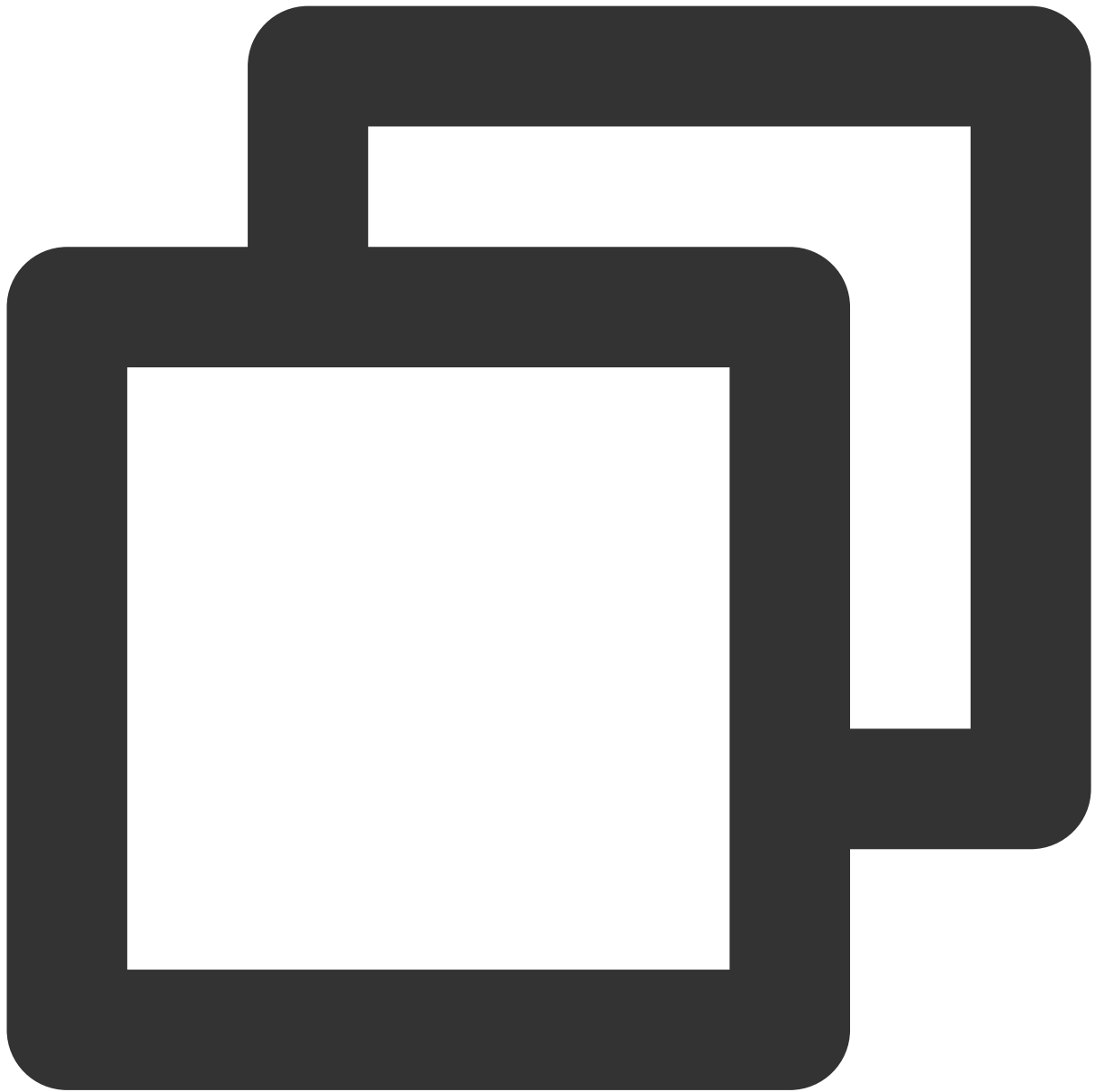
```
virtual void onKickedOffLine(const char* message) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
message	const char*	Kicked out description

**onUserSigExpired**

User credential timeout event.



```
virtual void onUserSigExpired() = 0;
```

### **onRoomNameChanged**

Room name change event.



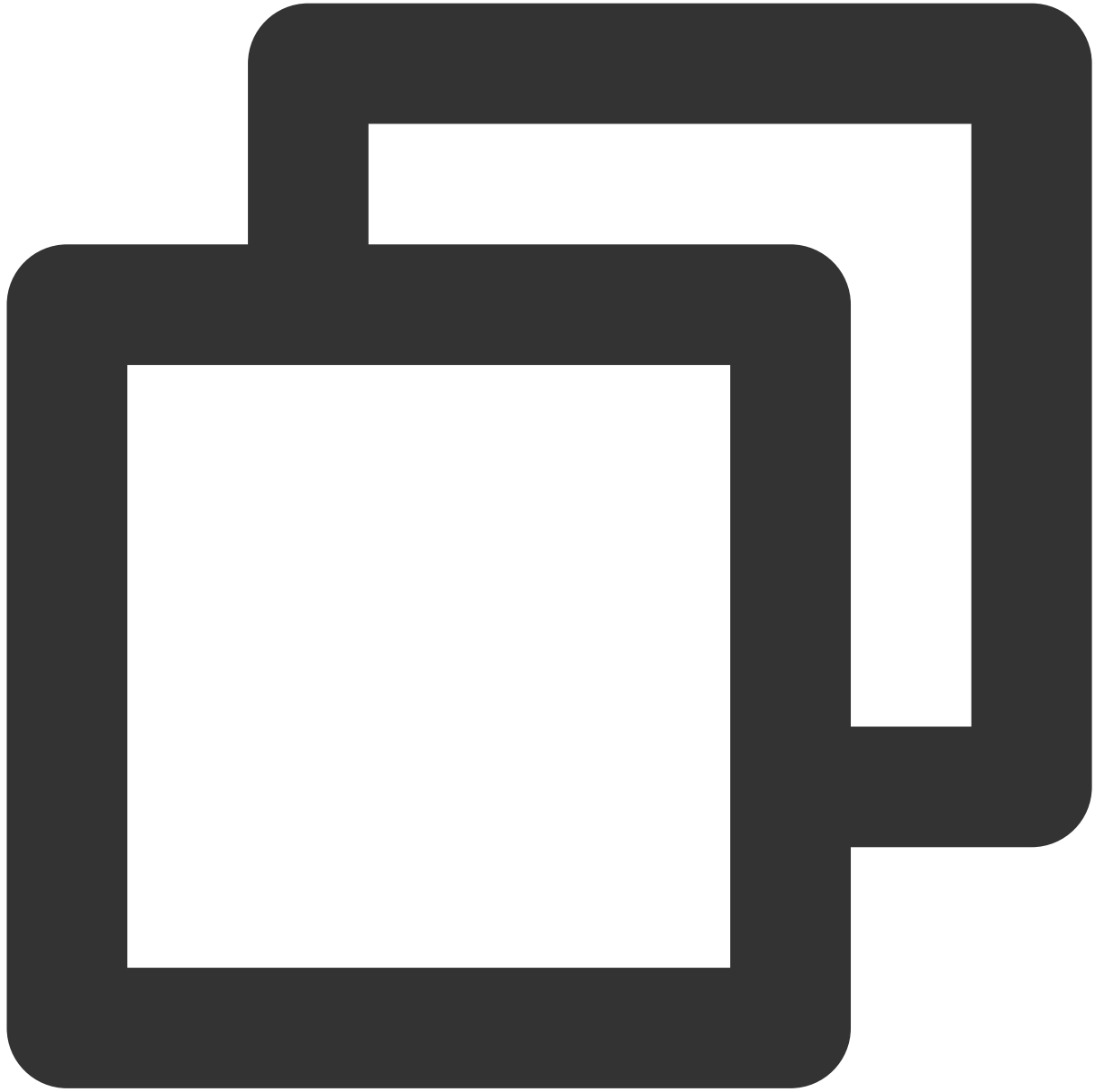
```
virtual void onRoomNameChanged(const char* roomId, const char* roomName) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	const char*	Room ID
roomName	const char*	Room Name

### onAllUserMicrophoneDisableChanged

Inside the room, all users' mic is disabled event.



```
virtual void onAllUserMicrophoneDisableChanged(const char* roomId, bool isDisable)
```

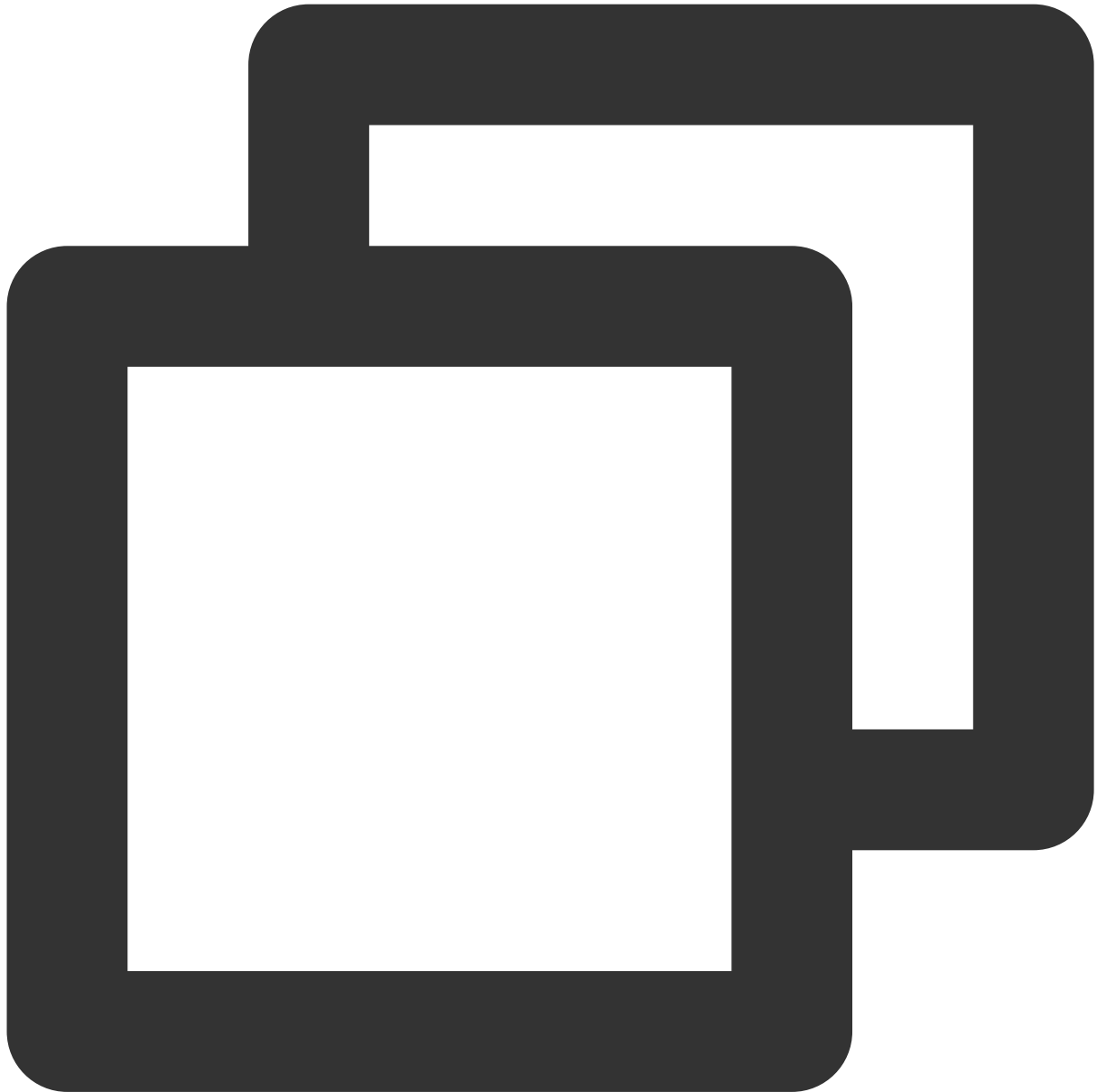
The parameters are as follows:

Parameter	Type	Meaning
roomId	const char*	Room ID
isDisable	bool	Whether it is disabled



## onAllUserCameraDisableChanged

All users' Camera in the Room are disabled event.



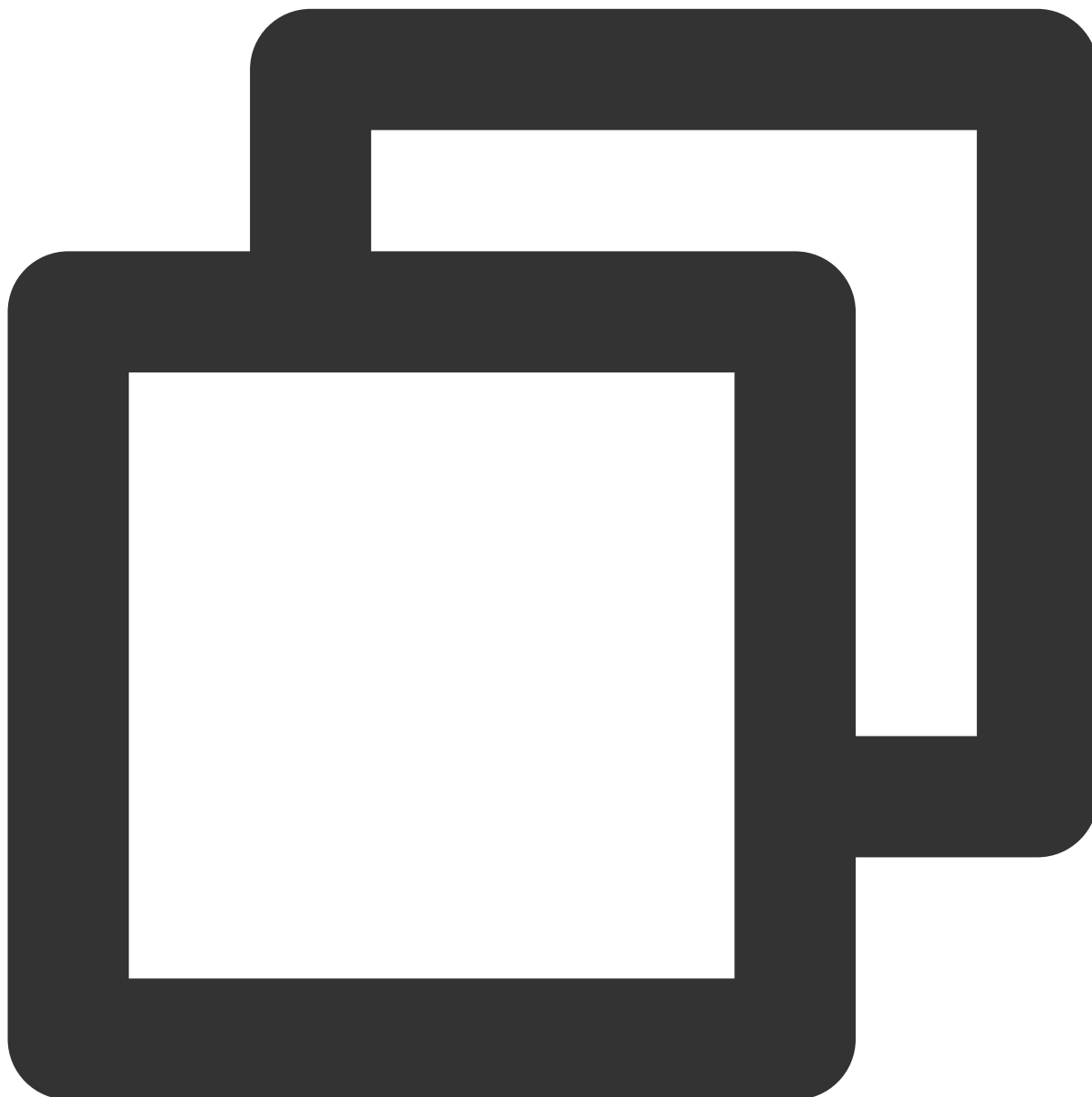
```
virtual void onAllUserCameraDisableChanged(const char* roomId, bool isDisable) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	const char*	Room ID
isDisable	bool	Whether it is disabled

## onSendMessageForAllUserDisableChanged

Inside the room, all users' text message sending is disabled event.



```
virtual void onSendMessageForAllUserDisableChanged(const char* roomId, bool isDisab
```

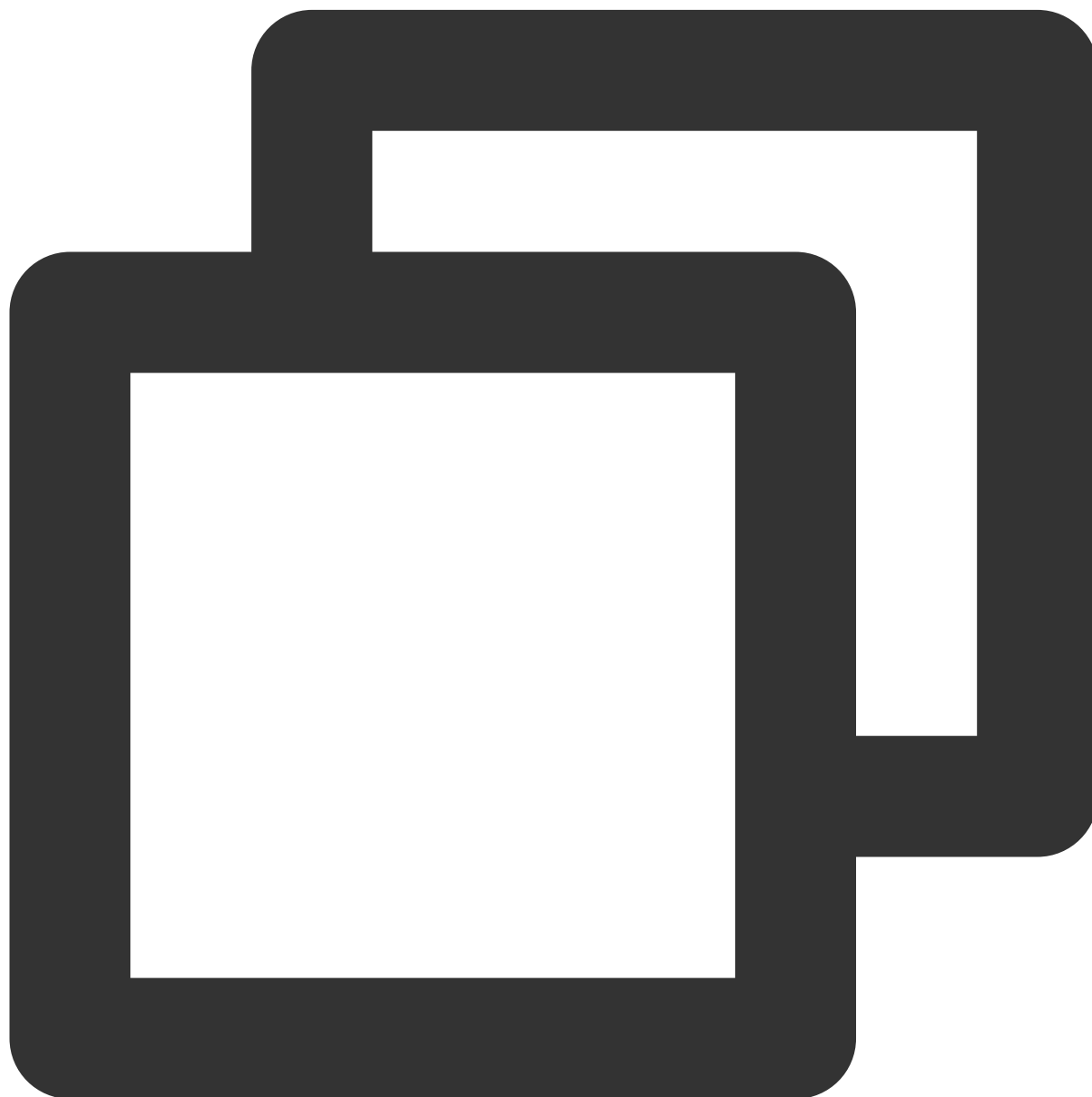
The parameters are as follows:

Parameter	Type	Meaning
roomId	const char*	Room ID

isDisable	bool	Whether it is disabled
-----------	------	------------------------

## onRoomDismissed

Room dissolution event.



```
virtual void onRoomDismissed(const char* roomId) = 0;
```

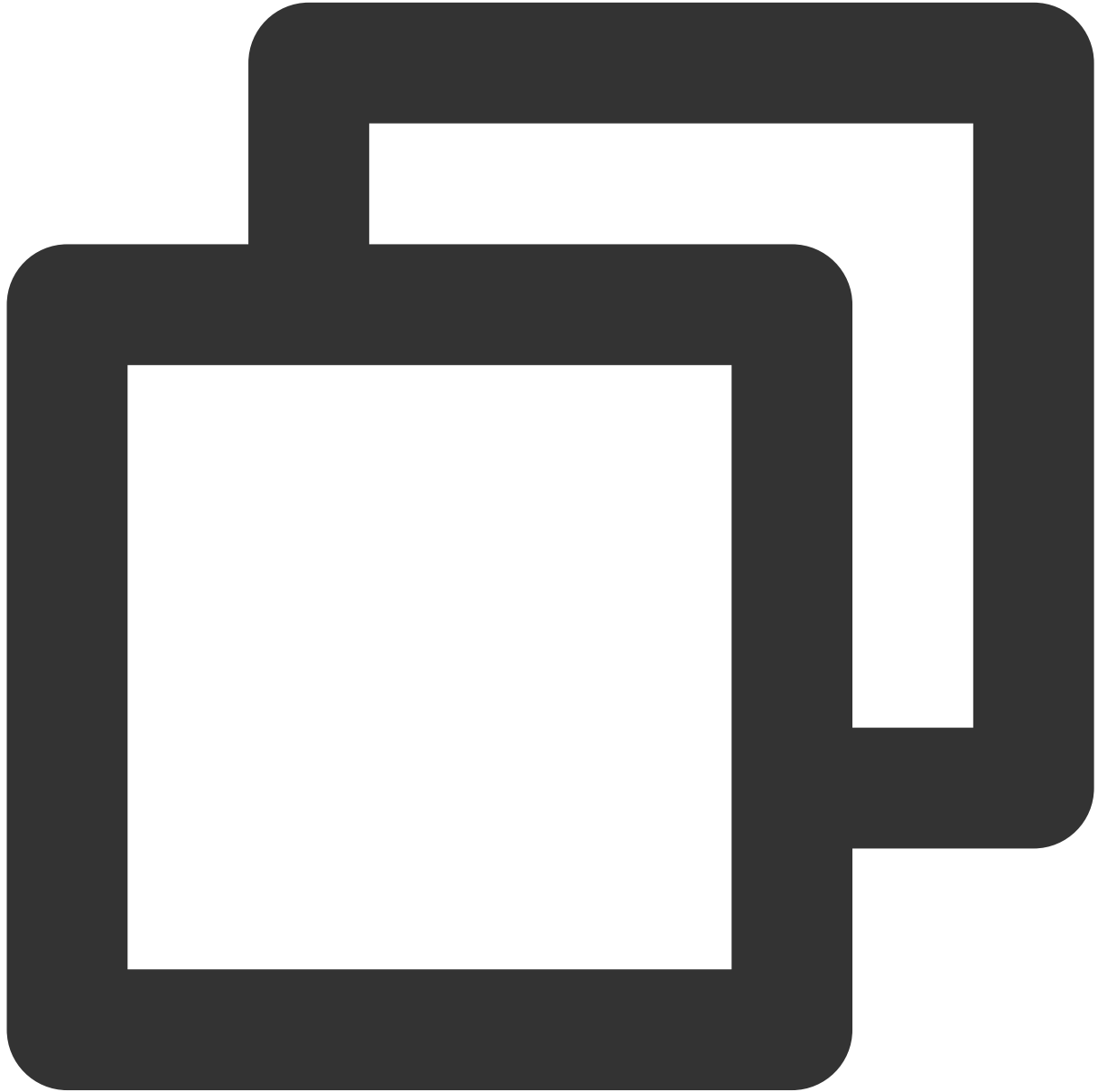
The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

roomId	const char*	Room ID
--------	-------------	---------

### onKickedOutOfRoom

Kicked out of the room event.



```
virtual void onKickedOutOfRoom(const char* roomId, const char* message) = 0;
```

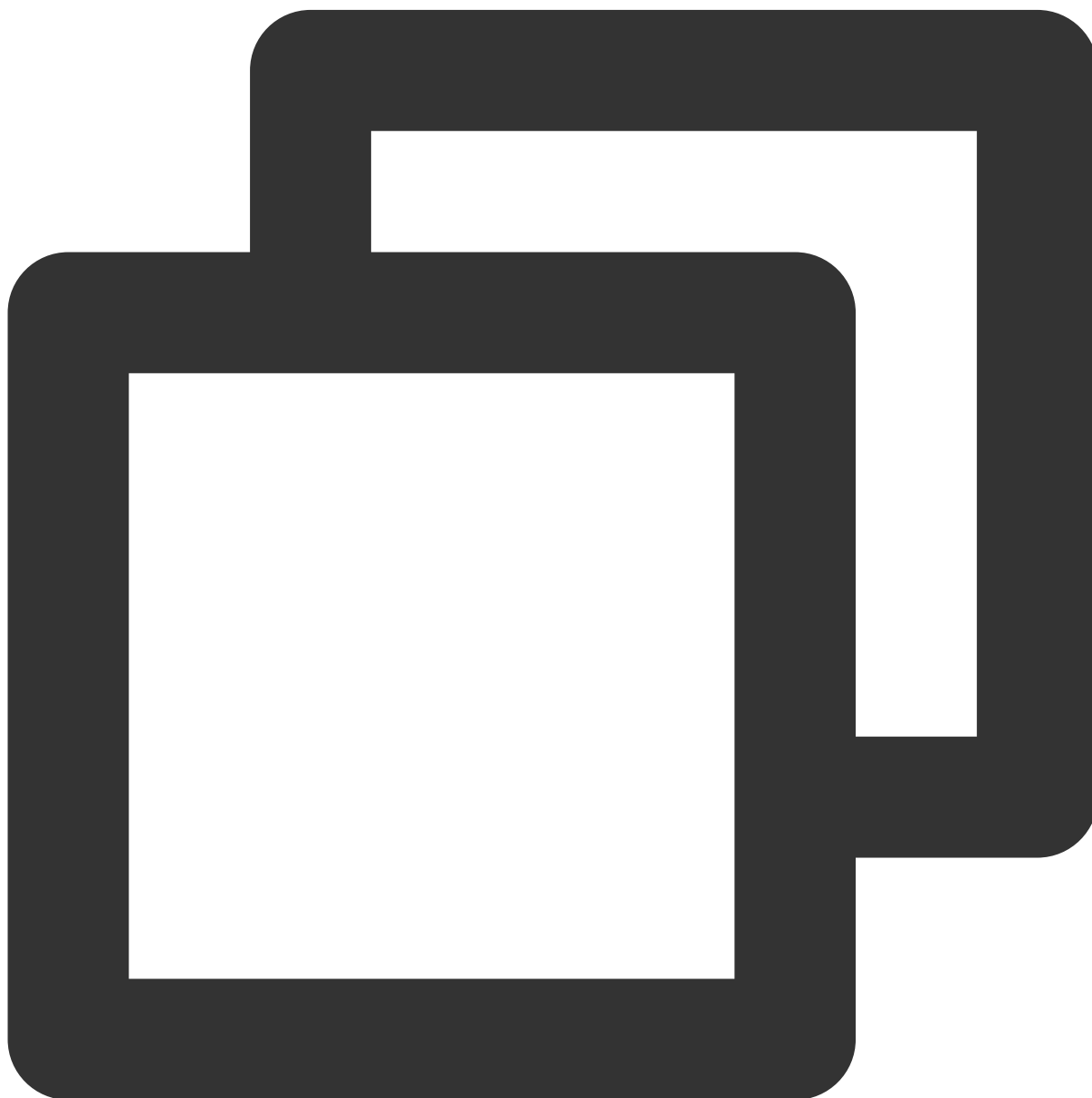
The parameters are as follows:

Parameter	Type	Meaning
-----------	------	---------

roomId	const char*	Room ID
message	const char*	Description of being kicked out

## onRoomSpeechModeChanged

Mic control mode changes in the room.



```
virtual void onRoomSpeechModeChanged(const char* roomId, TUISpeechMode speechMode)
```

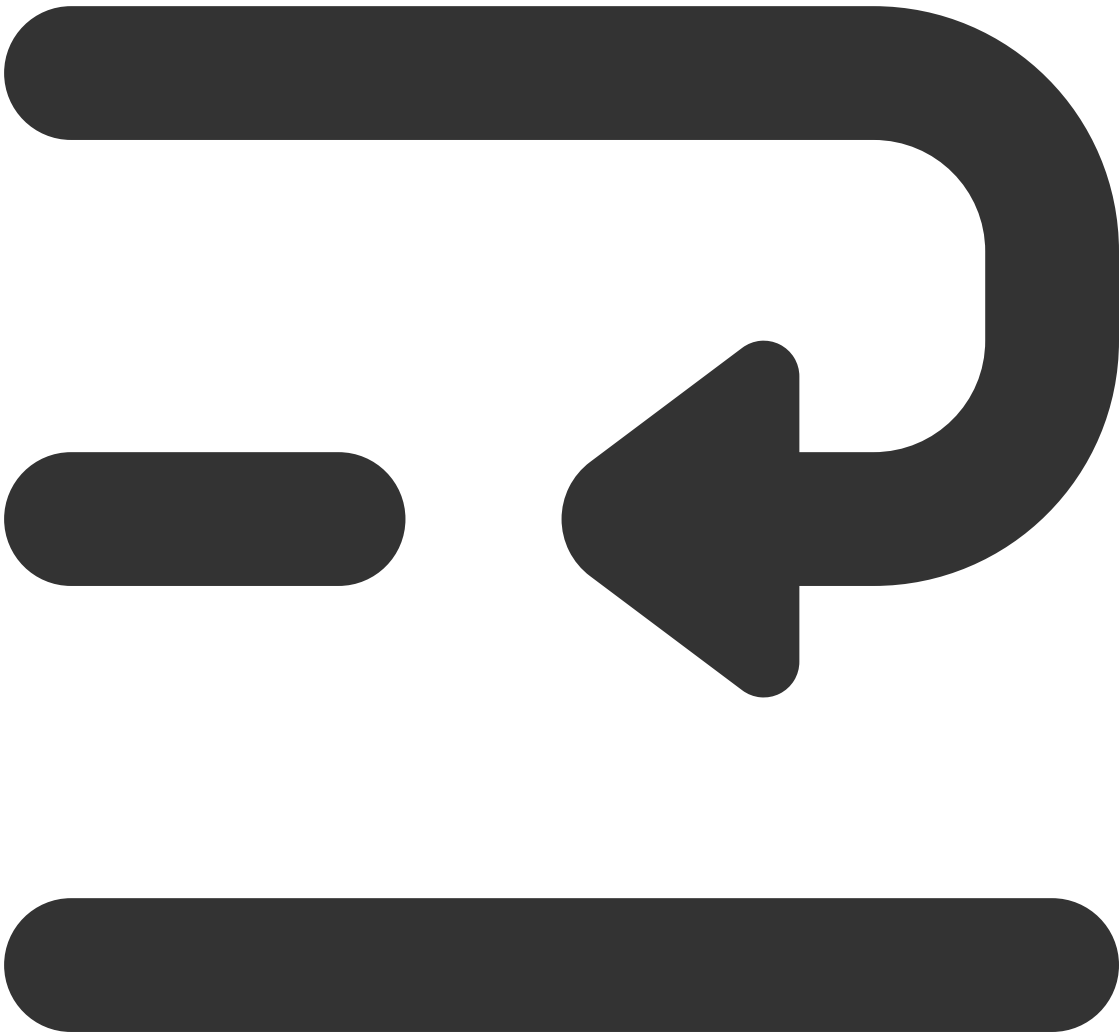
The parameters are as follows:

--	--	--

Parameter	Type	Meaning
roomId	const char*	Room ID
speechMode	TUISpeechMode	Mic control mode

### onRemoteUserEnterRoom

Remote user enters the room event.





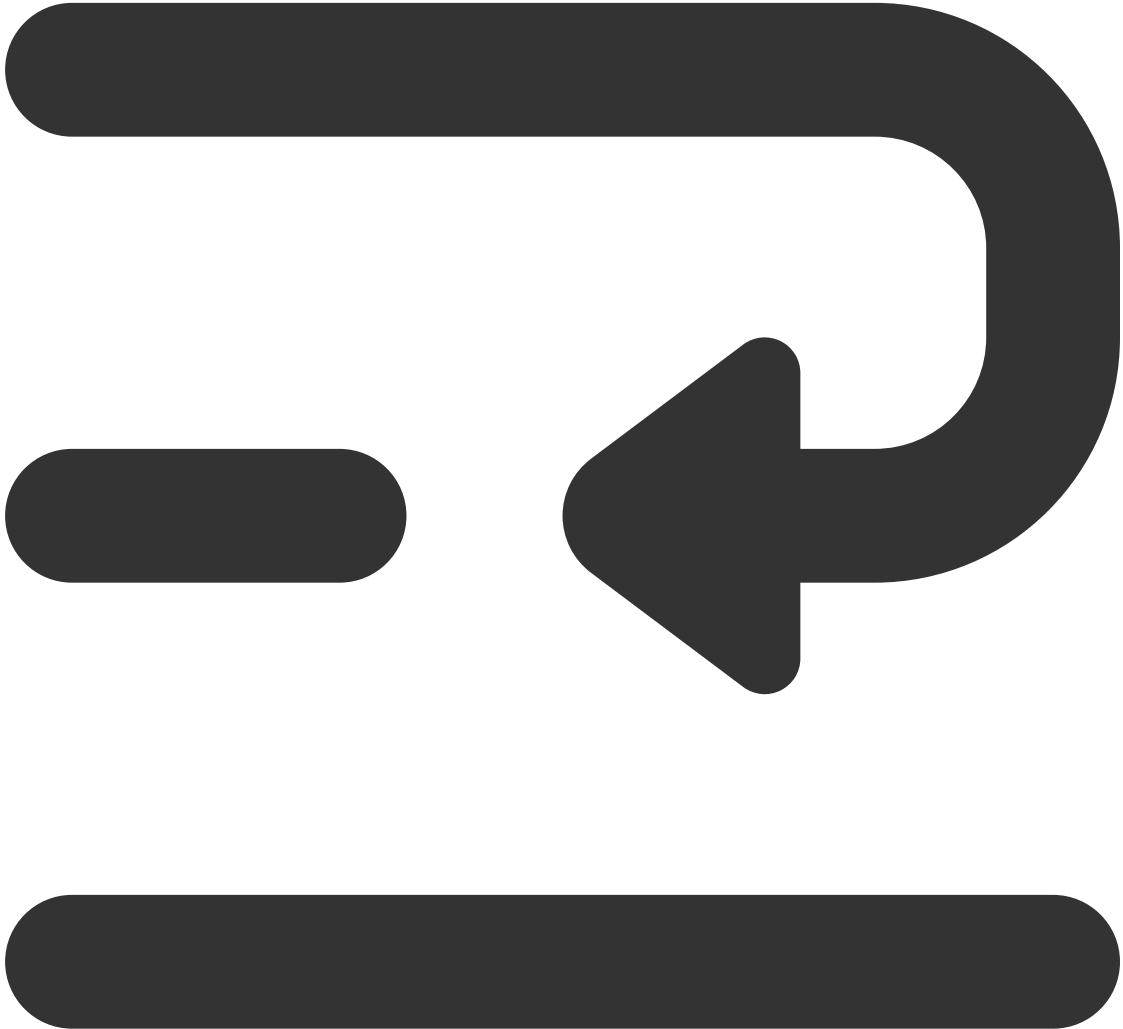
```
virtual void onRemoteUserEnterRoom(const char* roomId, const TUIUserInfo& userInfo)
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	const char*	Room ID
userInfo	const TUIUserInfo&	User information

**onRemoteUserLeaveRoom**

Remote user leaves the room event.







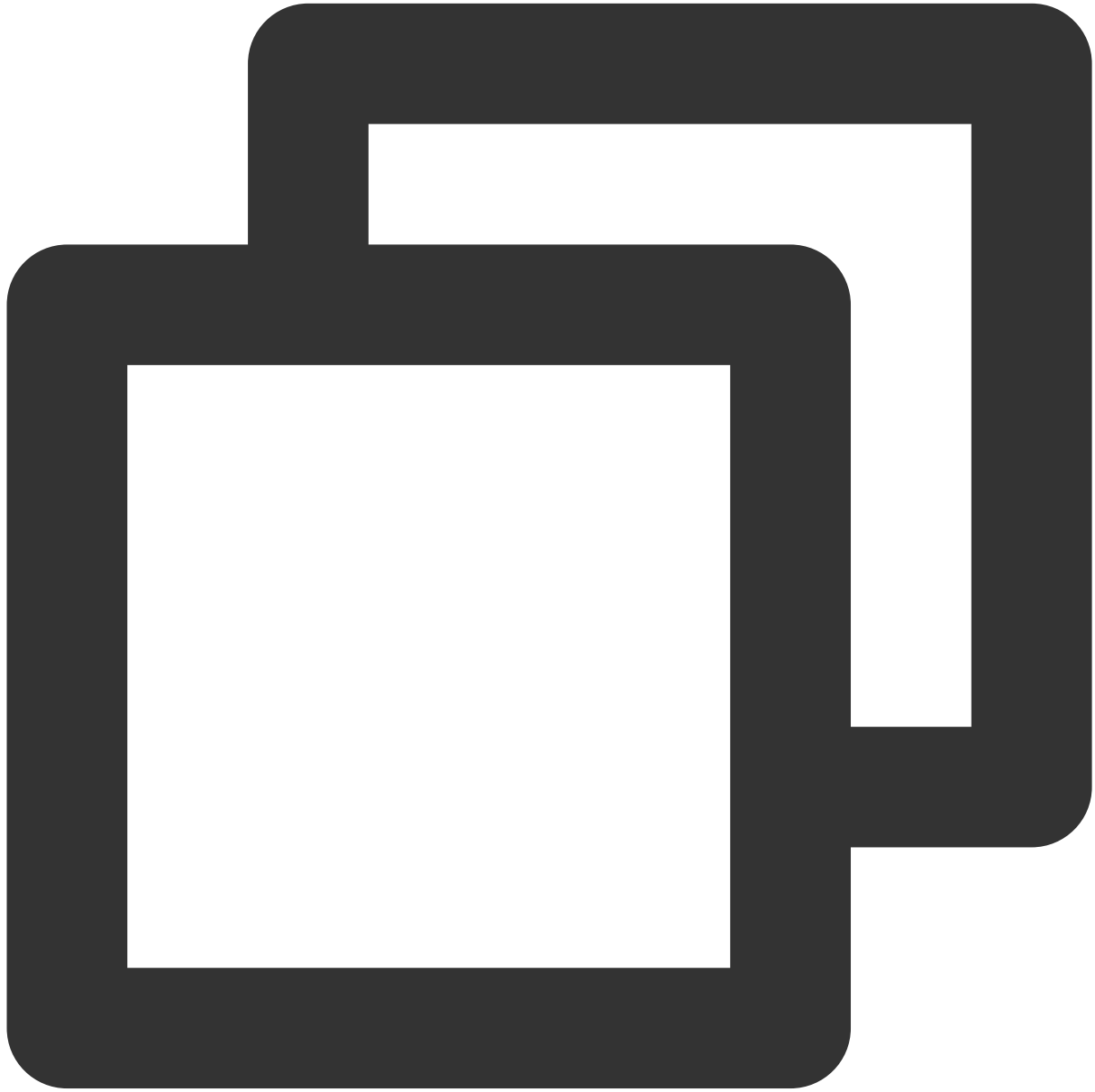
```
virtual void OnRemoteUserLeaveRoom(const char* roomId, const TUIUserInfo& userInfo)
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	const char*	Room ID
userInfo	const TUIUserInfo&	User information

## onUserRoleChanged

User role changes event.



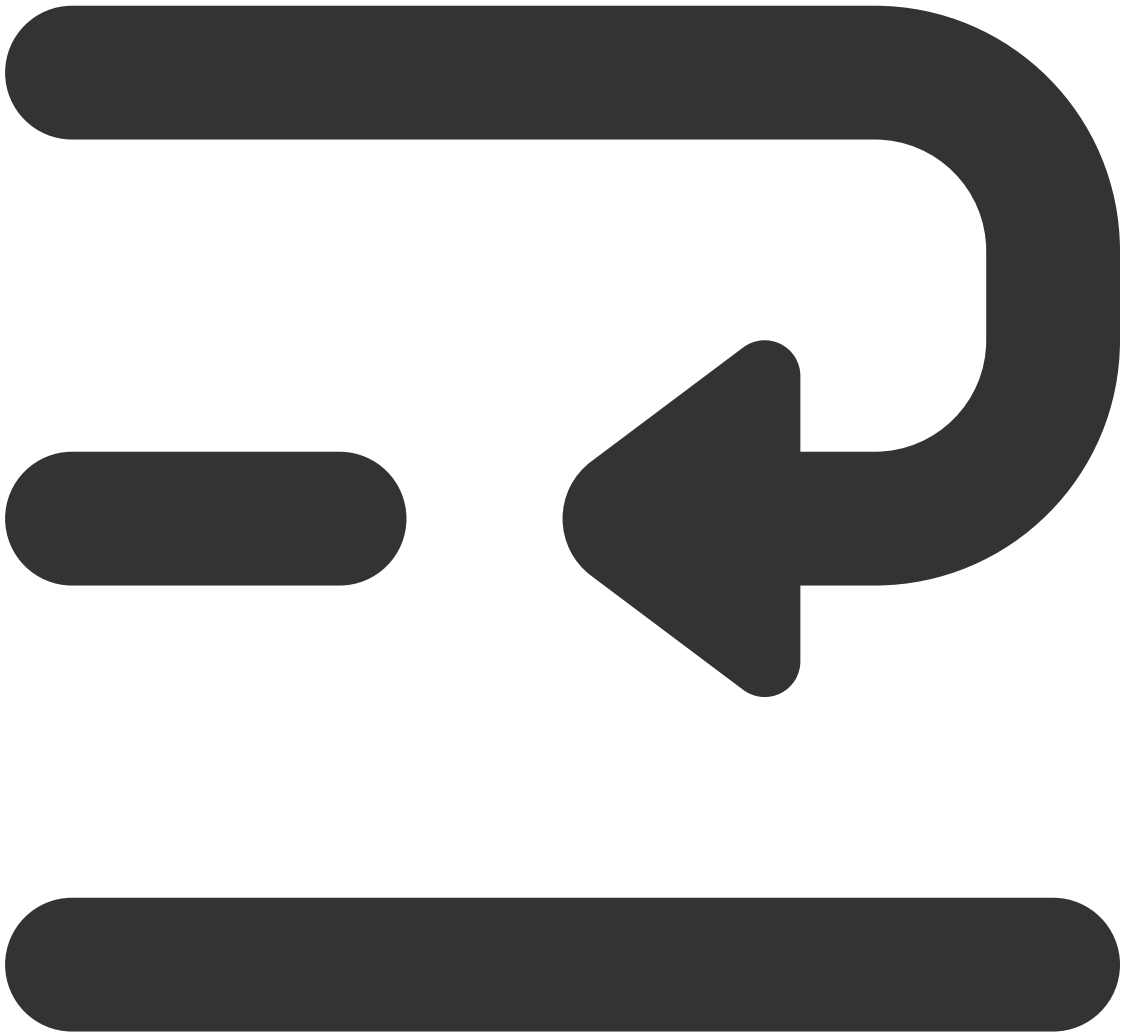
```
virtual void onUserRoleChanged(const char* userId, const TUIRole& userRole) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
userId	const char*	User ID
userRole	const TUIRole&	User Role

## onUserVideoStateChanged

User Video status changes event.





```
virtual void onUserVideoStateChanged(const char* userId, TUIVideoStreamType streamT
```

**The parameters are as follows:**

Parameter	Type	Meaning
userId	const char*	User ID
streamType	TUIVideoStreamType	Streams type
hasVideo	bool	Whether there are streams

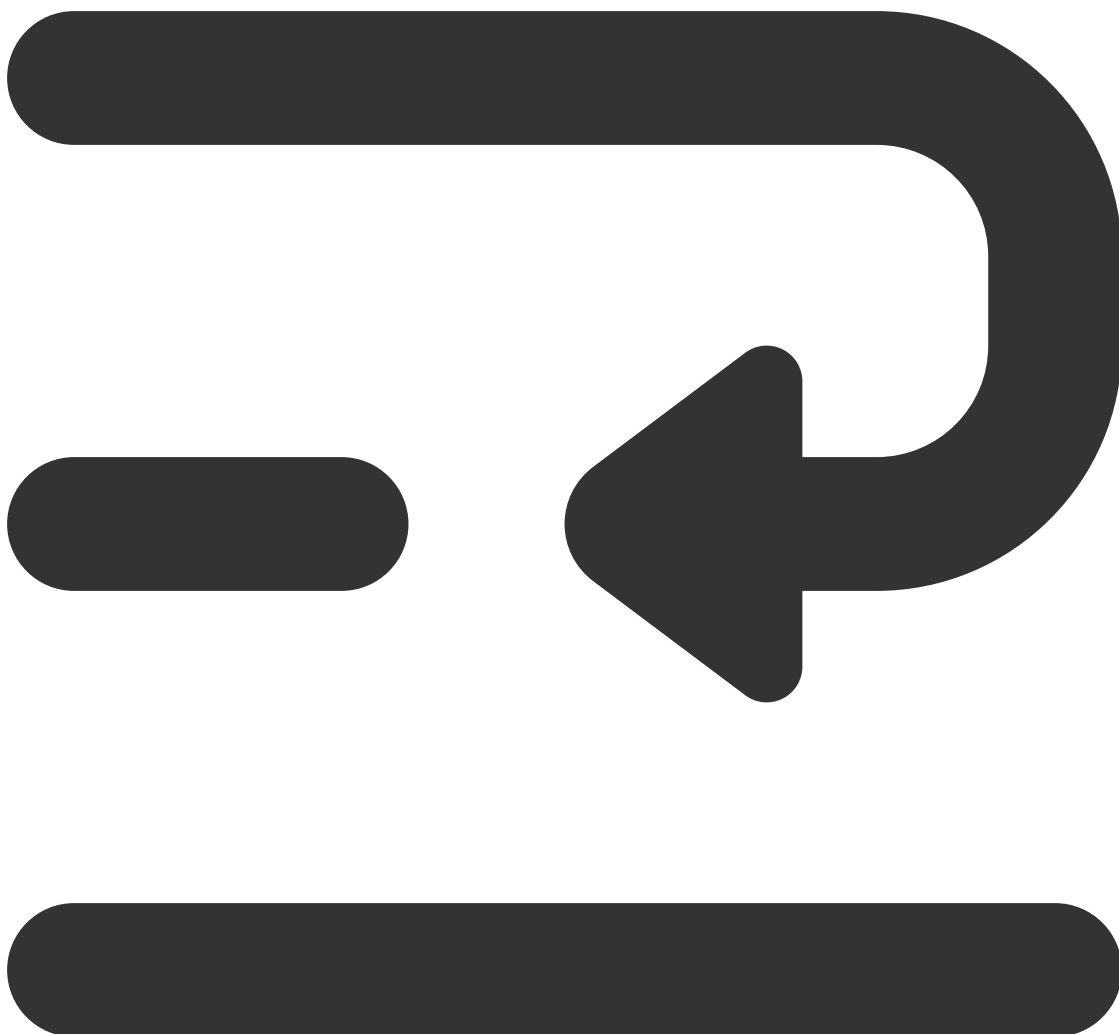
reason

TUICChangeReason

Reason for streams change

**onUserAudioStateChanged**

User Audio status changes event.





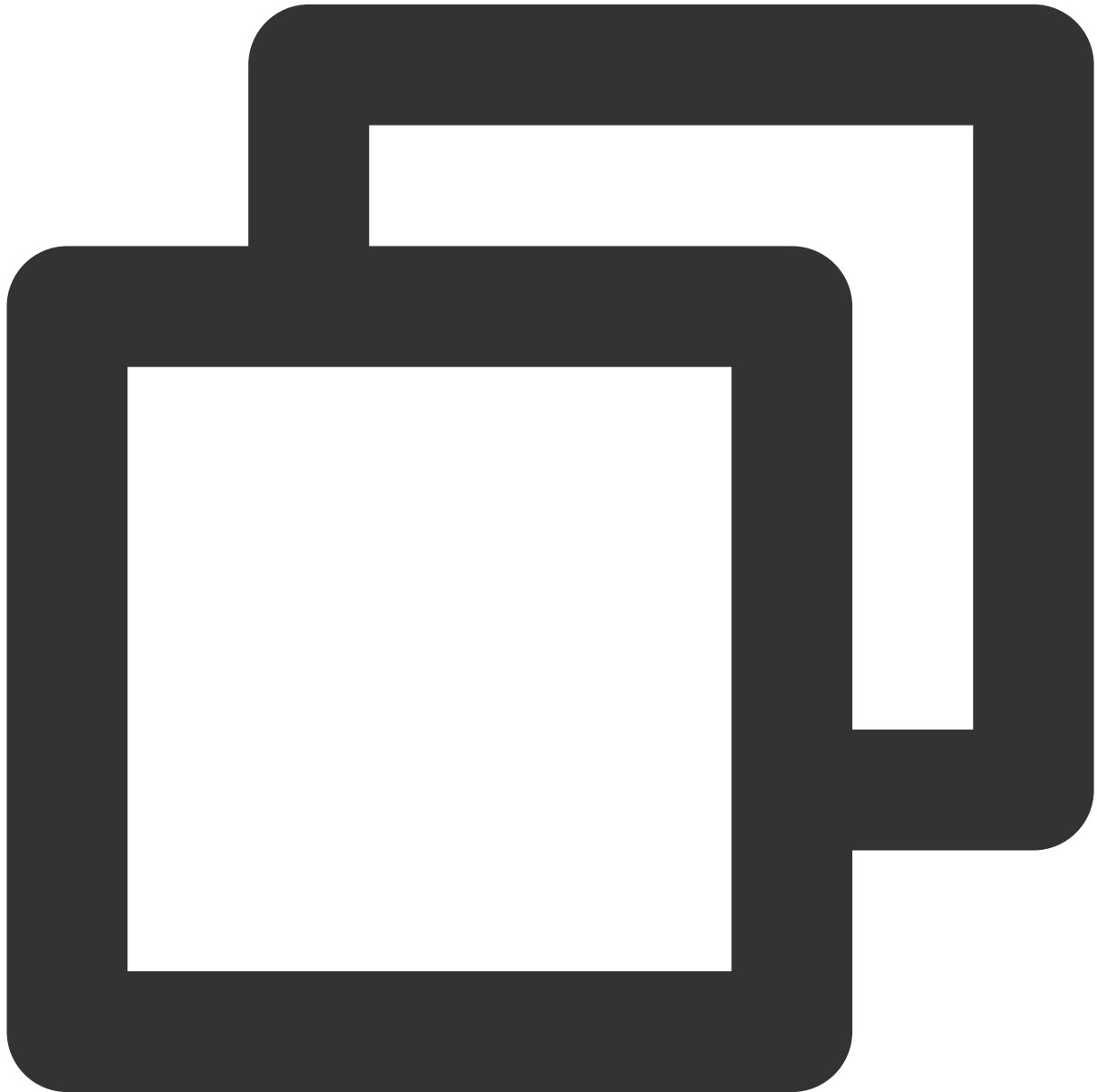
```
virtual void onUserAudioStateChanged(const char* userId, bool hasAudio, TUIChangeReason reason)
```

**The parameters are as follows:**

Parameter	Type	Meaning
userId	const char*	User ID
hasAudio	bool	Whether there are Audio streams
reason	TUIChangeReason	Reason for Audio streams change

## onUserVoiceVolumeChanged

User volume change event.



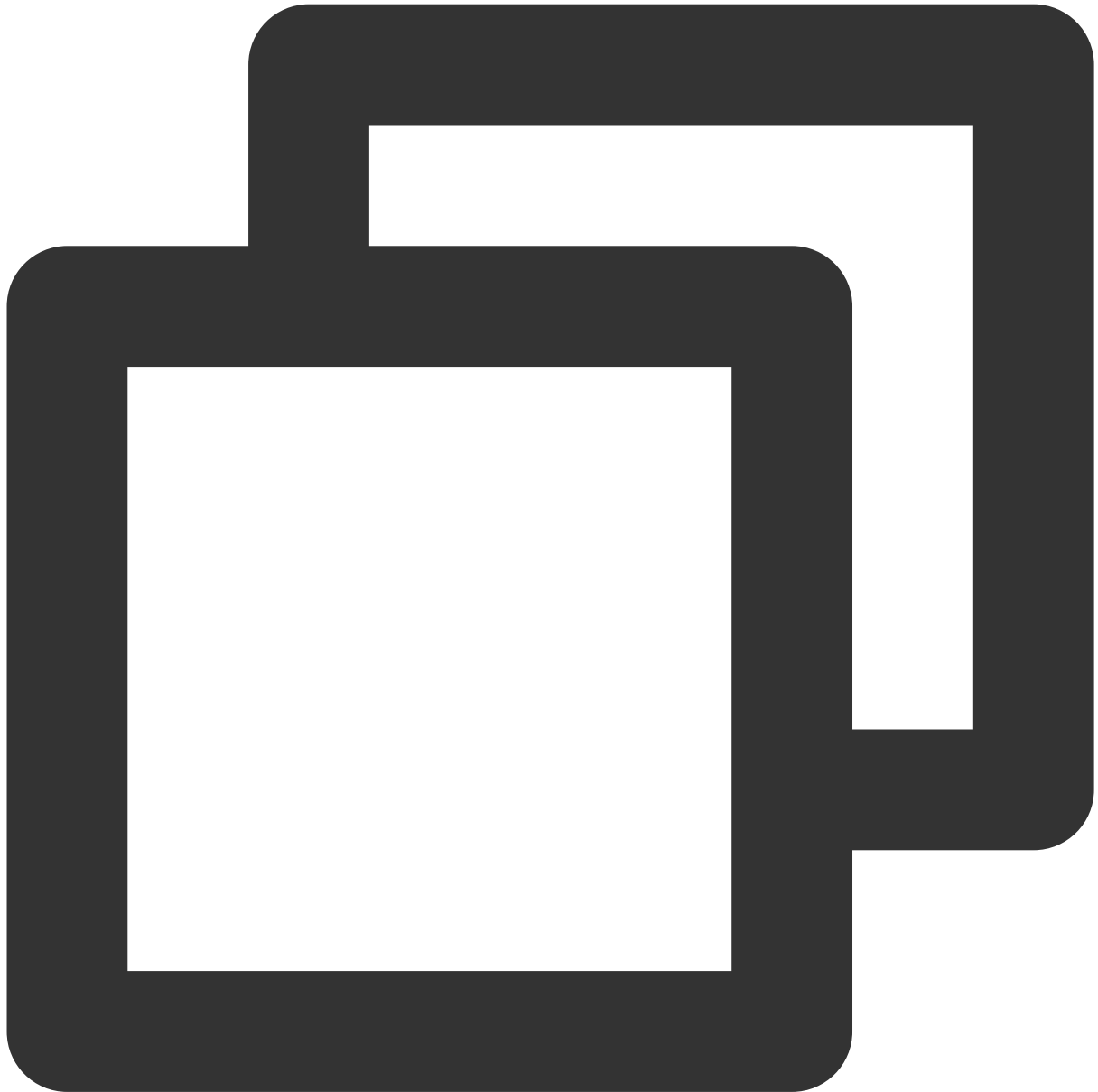
```
virtual void onUserVoiceVolumeChanged(TUIMap<const char*, int>* volumeMap) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
volumeMap	TUIMap<const char*, int>*	User Volume Map

## onSendMessageForUserDisableChanged

User text message sending ability changes event.



```
virtual void onSendMessageForUserDisableChanged(const char* roomId, const char* use
```

The parameters are as follows:

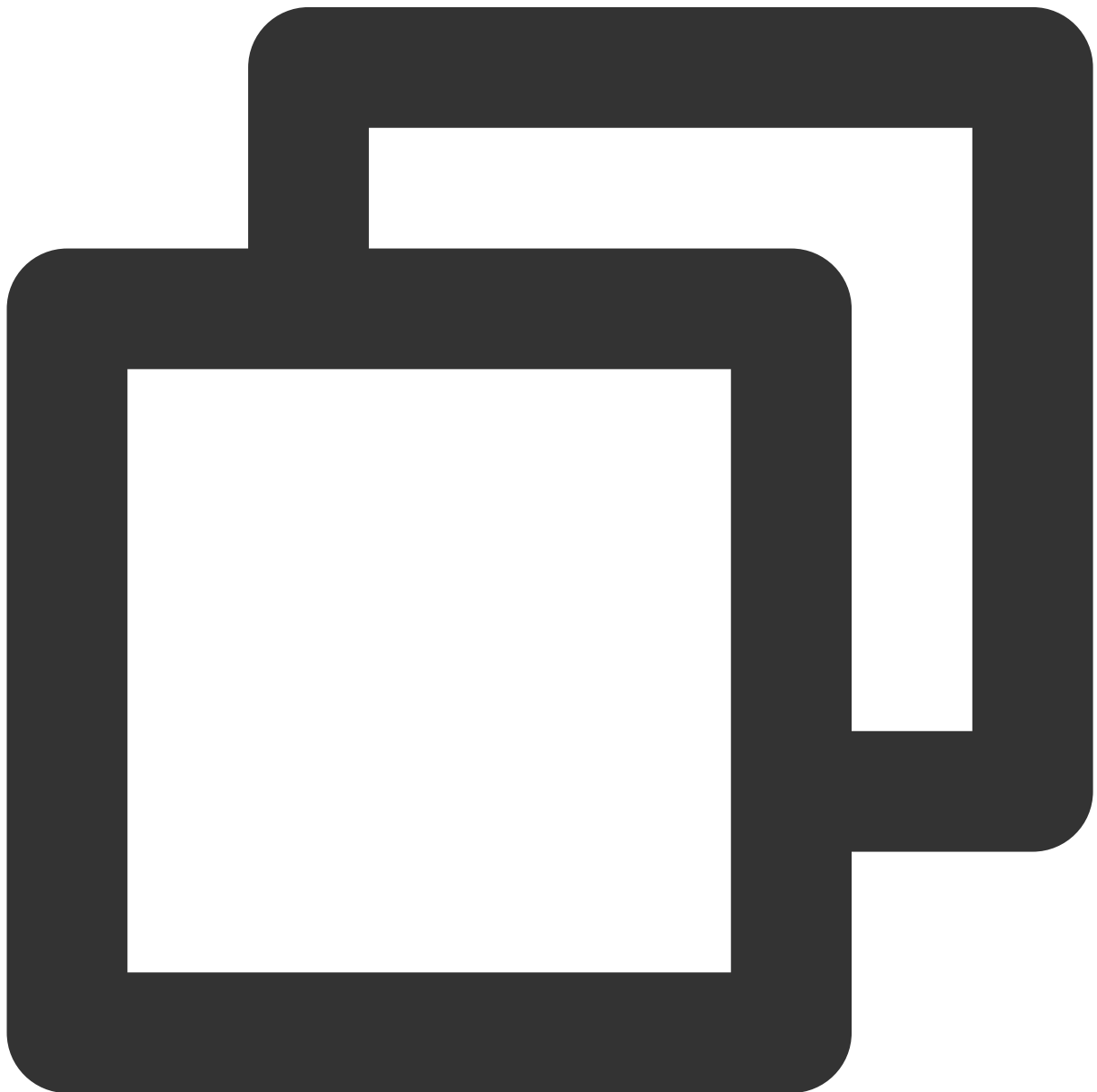
Parameter	Type	Meaning
roomId	const char*	Room ID
userId	const char*	User ID



isDisable	bool	Whether the user is prohibited from sending messages, true: The user is prohibited from sending text and custom messages; false: The user is allowed to send text and custom messages.
-----------	------	--

### onUserNetworkQualityChanged

User network status change event.



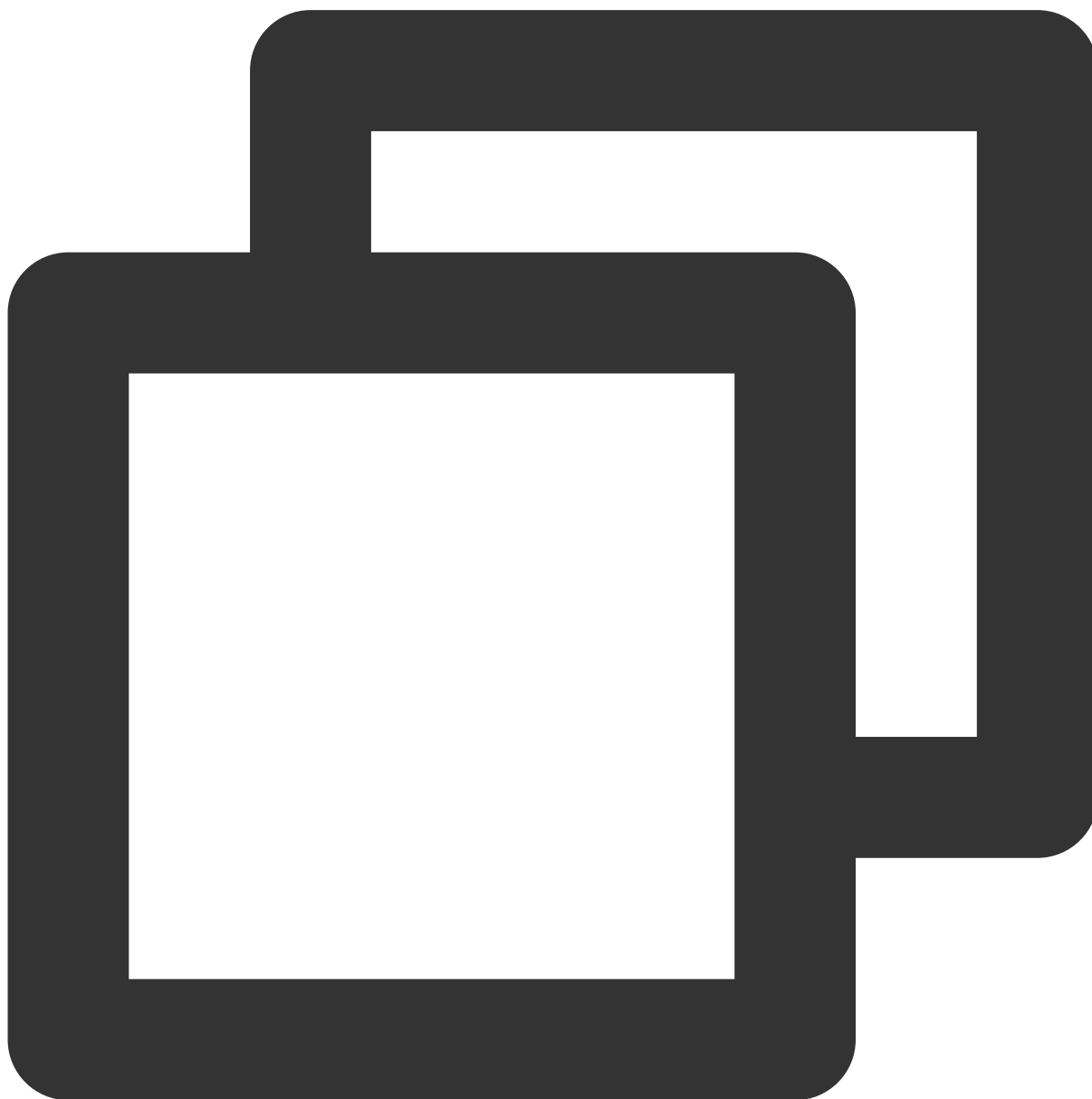
```
virtual void onUserNetworkQualityChanged(TUICollection<TUINetwork>* networkList) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
networkList	TUICollection<TUINetwork>*	User network status list

### onUserScreenCaptureStopped

Screen Sharing stopped Callback event.



```
virtual void onUserScreenCaptureStopped(int reason) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
reason	int	Stop reason: 0: User actively stops 1: Screen window closing causes the stop 2: Screen Sharing display screen status change (such as interface being unplugged, Projection mode change, etc.)

### onRoomMaxSeatCountChanged

Maximum number of mic slots changes event in the room (only effective in meeting type rooms).



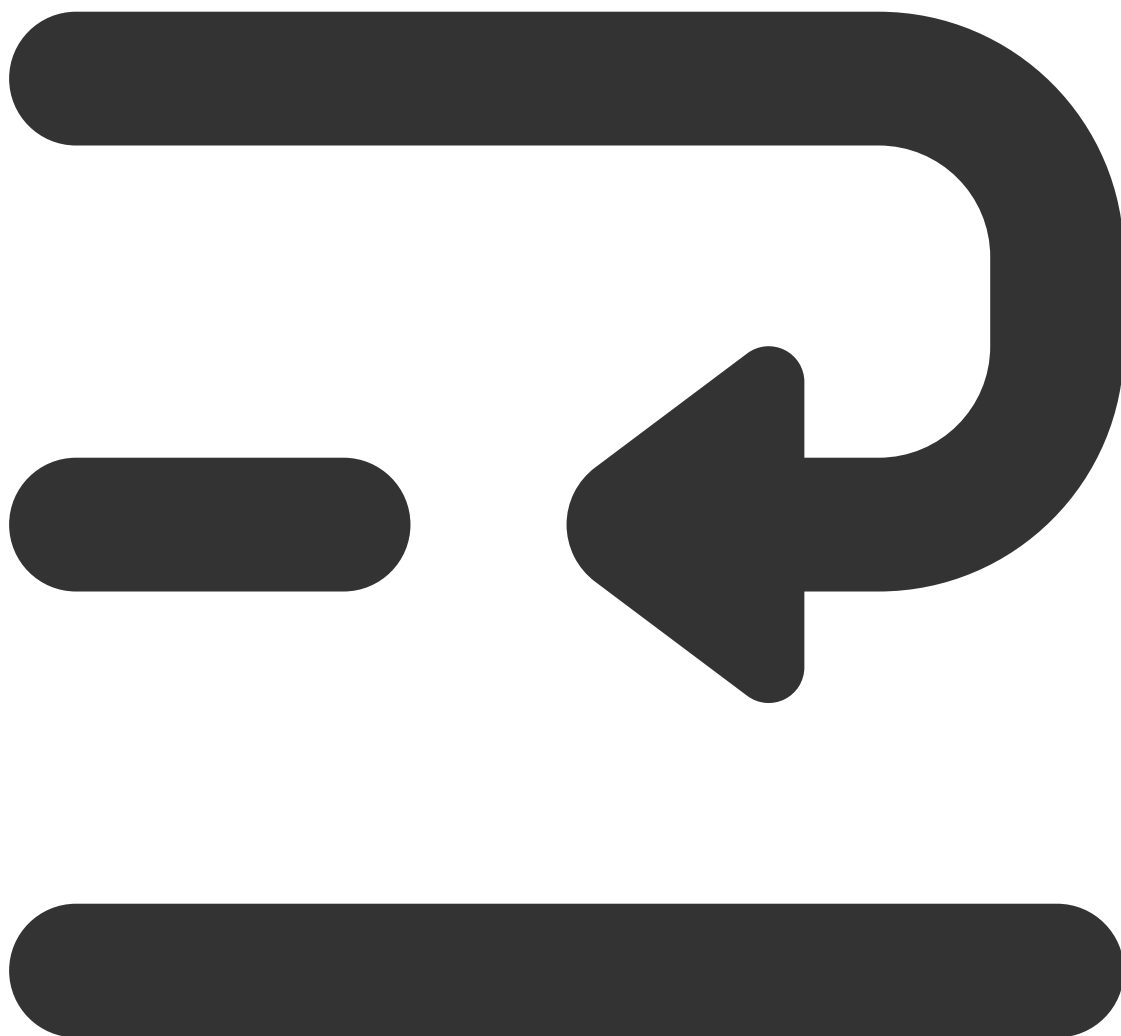
```
virtual void onRoomMaxSeatCountChanged(const char* roomId, int maxSeatCount) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning
roomId	const char*	Room ID
maxSeatCount	int	Maximum number of mic slots in the room

### onSeatListChanged

Mic slot list changes event.





```
virtual void onSeatListChanged(TUICollection<TUISeatInfo>* seatList, TUICollection<TUISeatInfo>
```

The parameters are as follows:

Parameter	Type	Meaning
seatList	TUICollection<TUISeatInfo>*	The latest user list on the mic, including newly on mic users
usersSeated	TUICollection<TUISeatInfo>*	Newly on mic user list

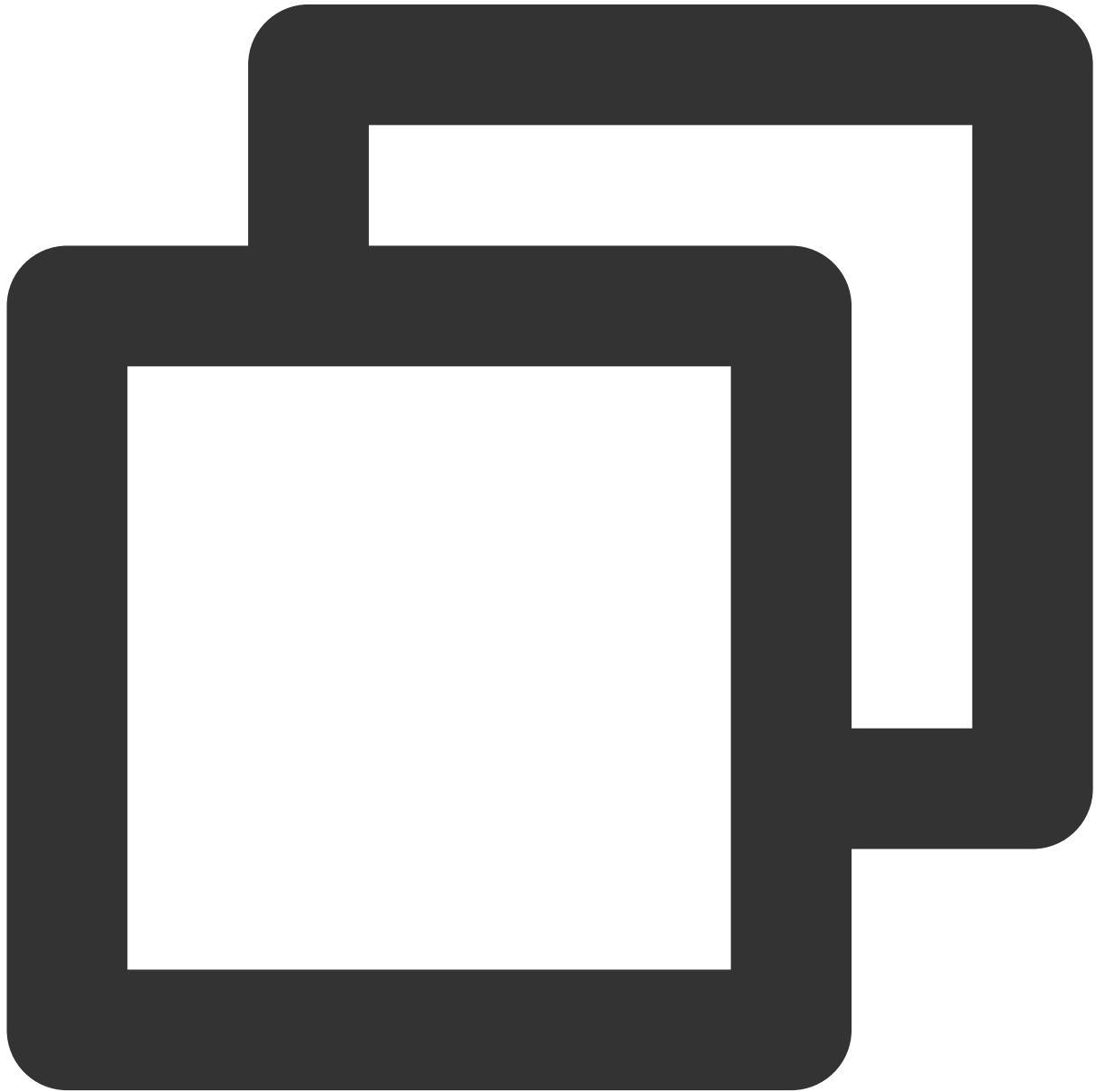
usersLeft

TUIList&lt;TUISeatInfo&gt;\*

Newly off mic user list

## onKickedOffSeat

Received the event of user being kicked off mic.



```
virtual void onKickedOffSeat(const char* userId) = 0;
```

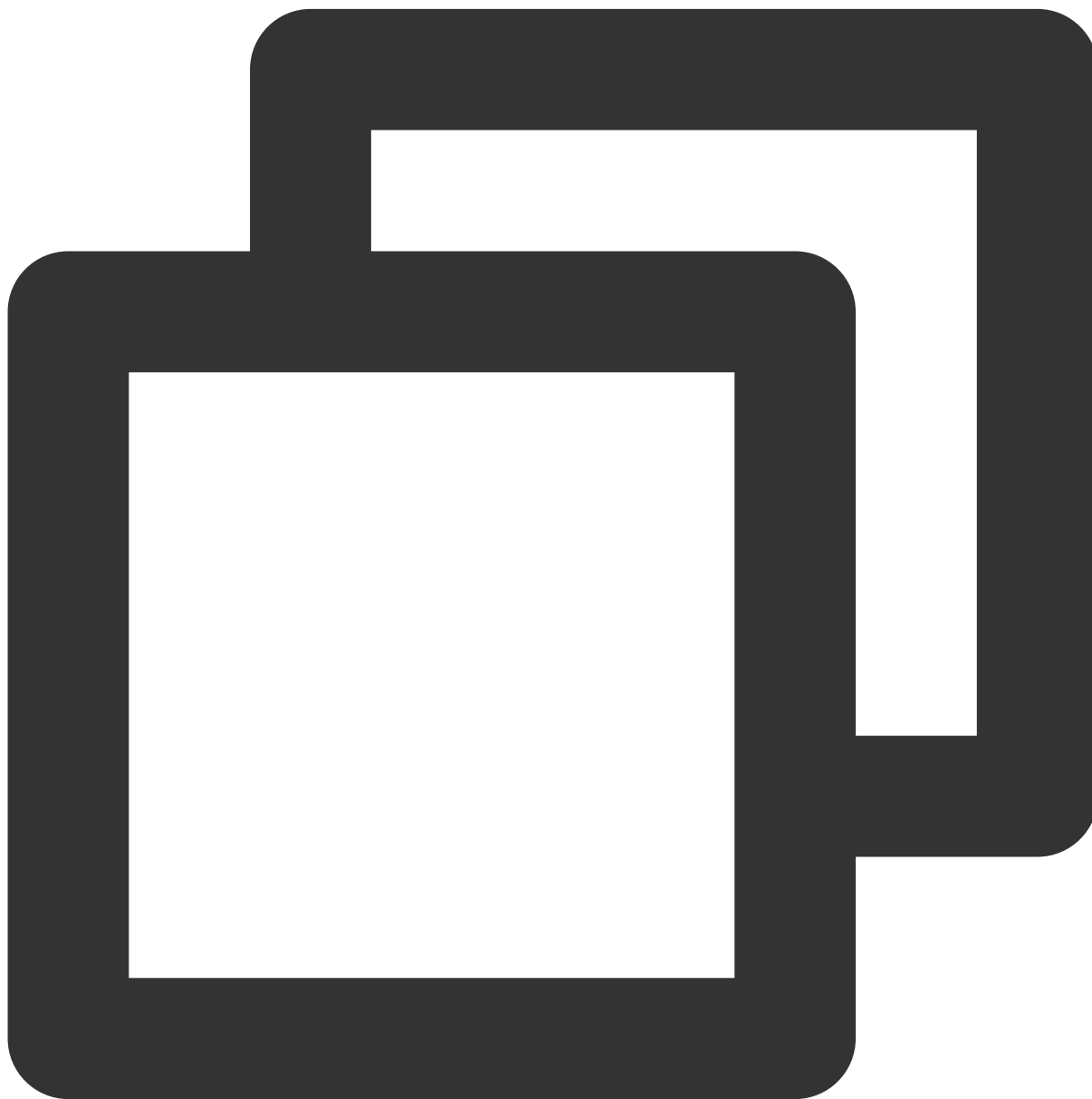
The parameters are as follows:

Parameter	Type	Meaning

userId	const char*	User ID
--------	-------------	---------

## onRequestReceived

Received request message event.



```
virtual void onRequestReceived(const TUIRequest* request) = 0;
```

The parameters are as follows:

Parameter	Type	Meaning



request	const TUIRequest*	Request content
---------	-------------------	-----------------

## onRequestCancelled

Received request cancellation event.



```
virtual void OnRequestCancelled(const char* requestId, const char* userId) = 0;
```

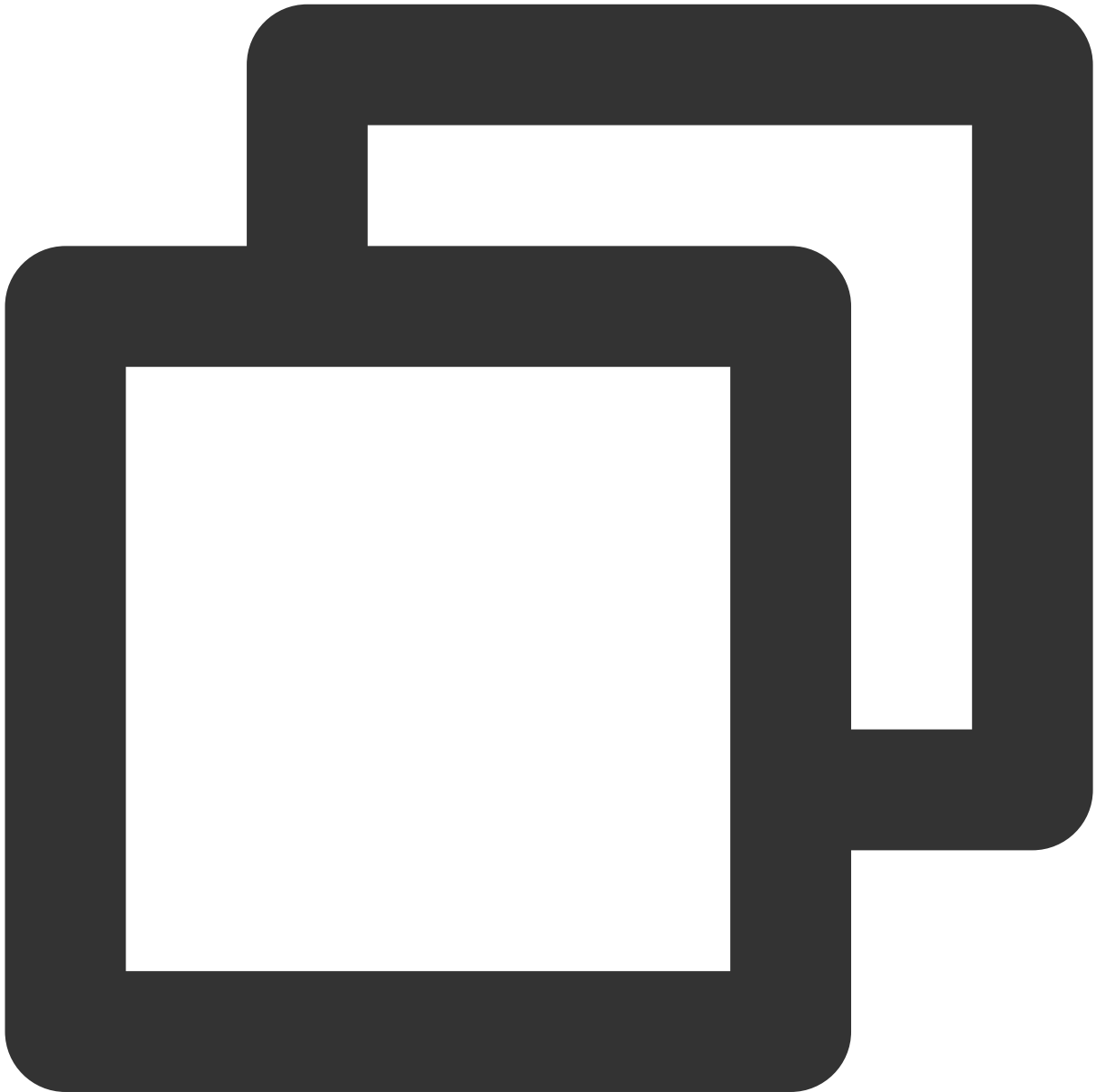
The parameters are as follows:

Parameter	Type	Meaning

requestId	const char*	Request ID
userId	const char*	User ID

onReceiveTextMessage

Received ordinary text message event.



```
virtual void onReceiveTextMessage(const char* roomId, const TUIMessage& message) =
```

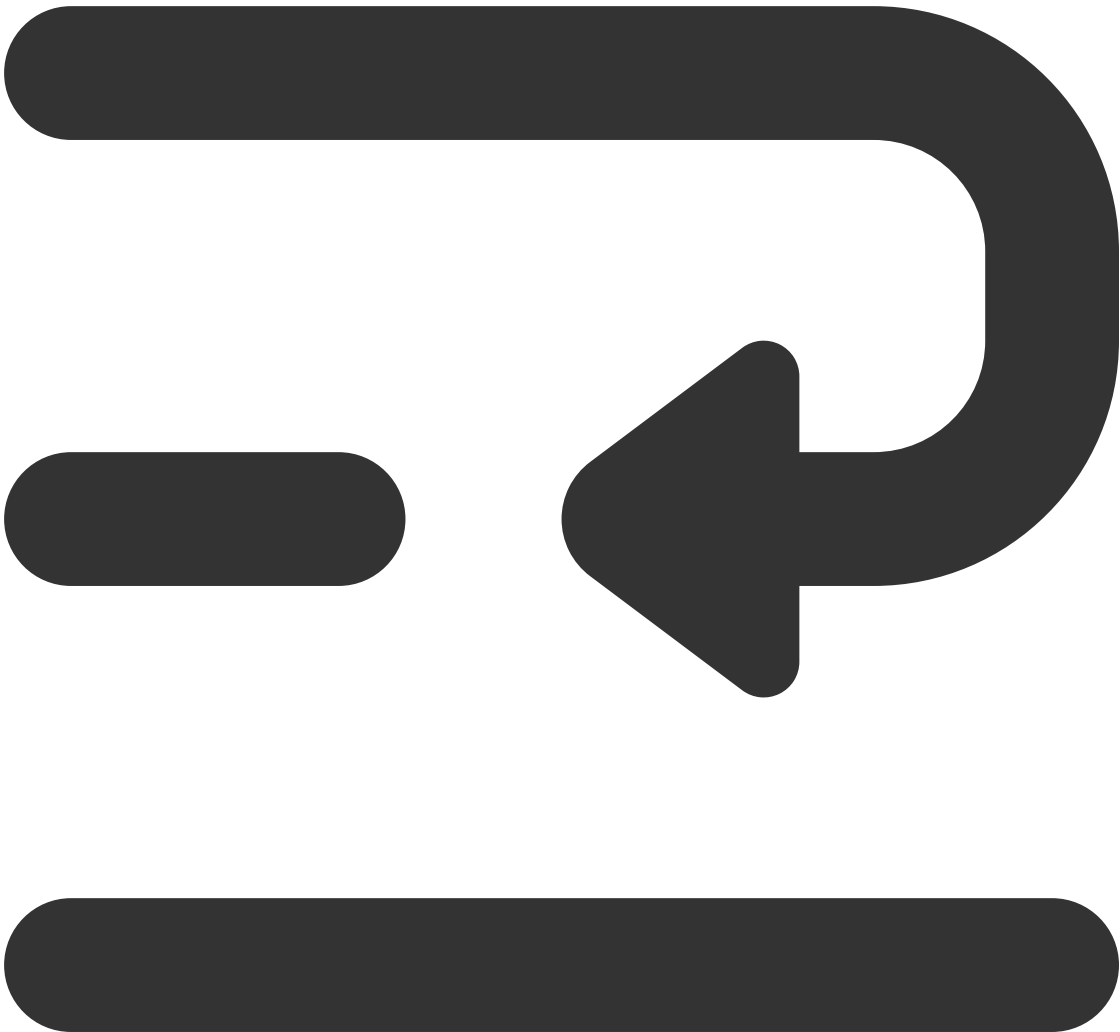
The parameters are as follows:

--	--	--

Parameter	Type	Meaning
roomId	const char*	Room ID
message	const TUIMessage&	Message content

onReceiveCustomMessage

Received custom message event.





```
virtual void onReceiveCustomMessage(const char* roomId, const TUIMessage& message)
```

**The parameters are as follows:**

Parameter	Type	Meaning
roomId	const char*	Room ID
message	const TUIMessage&	Message content



# Type Definition

Last updated : 2023-10-30 10:55:29

## Enumeration Definition

### TUIRoomDefine

Type	Description
<a href="#">TUIRoomType</a>	Room Type
<a href="#">TUISpeechMode</a>	Mic Control Mode
<a href="#">TUIMediaDevice</a>	Room Media Device Type
<a href="#">TUIRole</a>	Room Role Type
<a href="#">TUIVideoQuality</a>	Video Quality
<a href="#">TUIAudioQuality</a>	Audio Quality
<a href="#">TUIVideoStreamType</a>	Video Stream Type
<a href="#">TUIChangeReason</a>	Change Reason (User audio and video status change operation reason: self-modification or modified by room owner/administrator)
<a href="#">TUICaptureSourceType</a>	Screen Sharing Capture Source Type
<a href="#">TUIRequestAction</a>	Request Type

### TUICommonDefine

Type	Description
<a href="#">TUINetworkQuality</a>	Network Quality

## Common Structure

### TUIRoomDefine

Type	Description
<a href="#">TUIRoomInfo</a>	Room data
<a href="#">TUILoginUserInfo</a>	User Login Information

<a href="#">TUIUserInfo</a>	Room User Information
<a href="#">TUISeatInfo</a>	Room Seat Information
<a href="#">TUISeatLockParams</a>	Lock Seat Operation Parameters
<a href="#">TUIUserVoiceVolume</a>	Room User Volume
<a href="#">TUIRequest</a>	Signaling Request
<a href="#">TUIShareTarget</a>	Screen Sharing Capture Source Information

### TUICommonDefine

Type	Description
<a href="#">TUINetworkInfo</a>	Network Quality Information
<a href="#">TUIMessage</a>	Message
<a href="#">TUIImageBuffer</a>	Image Info

### TUIRoomType

#### Room Type

Enumeration	Value	Description
kConference	1	Conference Type Room, suitable for conference and education scenarios, this room can enable free speech, apply for speech, go live and other modes.
kLivingRoom	2	Live Type Room, suitable for live broadcast scenarios, this room can enable free speech, mic control mode, and the seats in this room are numbered.

### TUISpeechMode

#### Mic Control Mode

Enumeration	Value	Description
kFreeToSpeak	1	Free speech mode.
kApplyToSpeak	2	Apply to speak mode. (Only effective in conference type room)

kApplySpeakAfterTakingSeat	3	Go Live mode.
----------------------------	---	---------------

## TUIMediaDevice

### Room Media Device Type

Enumeration	Value	Description
kMicrophone	1	Mic
kCamera	2	Camera
kApplyScreenSharing	3	Screen Sharing

## TUIRole

### Room Role Types

Enumeration	Value	Description
kRoomOwner	0	Room Owner, generally refers to the creator of the room, the highest authority holder in the room
kAdministrator	1	Room Administrator
kGeneralUser	2	General Member in the room

## TUIVideoQuality

### Video Quality

Enumeration	Value	Description
kVideoQuality_360P	1	Low-quality 360P
kVideoQuality_540P	2	Standard Definition 540P
kVideoQuality_720P	3	High Definition 720P
kVideoQuality_1080P	4	Ultra-clear 1080P

## TUIAudioQuality

### Audio Quality

Enumeration	Value	Description



kAudioQualitySpeech	0	Speech Mode
kAudioQualityDefault	1	Default Mode
kAudioQualityMusic	2	Music Mode

## TUIVideoStreamType

Video Stream Type

Enumeration	Value	Description
kCameraStream	0	High-quality Camera Video Stream
kScreenStream	1	Screen Sharing Video Stream
kCameraStreamLow	2	Low-quality Camera Video Stream

## TUIChangeReason

Change Reason (User audio and video status change operation reason: self-modification or modification by room owner/administrator)

Enumeration	Value	Description
kChangedBySelf	0	Self-operation
kChangedByAdmin	1	Room Owner or Administrator Operation

## TUICaptureSourceType

Screen Sharing Capture Source Type

Enumeration	Value	Description
kUnknown	-1	Undefined
kWindow	0	Window
kScreen	1	Screen

## TUIRequestAction

Request Type

Enumeration	Value	Description
kInvalidAction	0	Invalid Request

kRequestToOpenRemoteCamera	1	Request Remote User to Open Camera
kRequestToOpenRemoteMicrophone	2	Request Remote User to Open Microphone
kRequestToConnectOtherRoom	3	Request to Connect to Other Room
kRequestToTakeSeat	4	Request to Go Live
kRequestRemoteUserOnSeat	5	Request Remote User to Go Live
kApplyToAdminToOpenLocalCamera	6	Request to Admin to Open Local Camera
kApplyToAdminToOpenLocalMicrophone	7	Request to Admin to Open Local Microphone

## TUINetworkQuality

### Network Quality

Enumeration	Value	Description
kQualityUnknown	0	Undefined
kQualityExcellent	1	Current Network is Excellent
kQualityGood	2	Current Network is Good
kQualityPoor	3	Current Network is Average
kQualityBad	4	Current Network is Poor
kQualityVeryBad	5	Current Network is Very Poor
kQualityDown	6	Current Network Does Not Meet TRTC's Minimum Requirements

## TUIRoomInfo

### Room Information

Field	Type	Description
roomId	const char*	Room ID
ownerId	const char*	Host ID, default is the room creator (read-only)

roomType	<a href="#">TUIRoomType</a>	Room Type
name	const char*	Room Name, default is the room ID
speechMode	<a href="#">TUISpeechMode</a>	Mic Control Mode
isCameraDisableForAllUser	bool	Whether to Disable Opening Camera (optional when creating a room), default value: false
isMicrophoneDisableForAllUser	bool	Whether to Disable Opening Microphone (optional when creating a room), default value: false
isMessageDisableForAllUser	bool	Whether to Disable Sending Messages (optional when creating a room), default value: false
maxSeatCount	int	Maximum Number of Mic Seats
createTime	long	Room Creation Time (read-only)
memberCountt	int	Number of Members in the Room (read-only)

## TUILoginUserInfo

User Login Information

Field	Type	Description
userId	const char*	User ID
userName	const char*	User Name
avatarUrl	const char*	User Avatar URL

## TUIUserInfo

User Information in the Room

Field	Type	Description
userId	const char*	User ID
userName	const char*	User Name
avatarUrl	const char*	User Avatar URL

userRole	<b>TUIRole</b>	User Role Type
hasAudioStream	bool	Whether There is Audio Stream, default value: false
hasVideoStream	bool	Whether There is Video Stream, default value: false
hasScreenStream	bool	Whether There is Screen Sharing Stream, default value: false

## TUISeatInfo

Seat Information in the Room

Field	Type	Description
index	int	Mic Seat Number
userId	const char*	User ID
isLocked	bool	Whether the Mic Seat is Locked, default false
isVideoLocked	bool	Whether the Mic Seat is Prohibited from Opening Camera, default false
isAudioLocked	bool	Whether the Mic Seat is Prohibited from Opening Microphone, default false

## TUISeatLockParams

Lock Seat Operation Parameters

Field	Type	Description
lockSeat	bool	Lock Mic Seat, default false
lockVideo	bool	Lock Mic Seat Camera, default false
lockAudio	bool	Lock Mic Seat Microphone, default false

## TUIUserVoiceVolume

User Voice Volume in the Room

--	--	--

Field	Type	Description
userId	const char*	User ID
volume	int	Volume Size, Value range 0 - 100

## TUIRequest

Signaling Request

Field	Type	Description
requestId	uint32_t	Request ID
requestAction	<a href="#">TUIRequestAction</a>	Request Type
userId	const char*	User ID
content	const char*	Signaling Content
timestamp	uint32_t	Timestamp

## TUIShareTarget

Screen Sharing Capture Source Information

Field	Type	Description
id	TUISourceId	Capturing Source ID, for windows, this field represents the window ID; for screens, this field represents the display ID
sourceType	<a href="#">TUICaptureSourceType</a>	Capturing Source Type
sourceName	const char*	Capturing Source Name
thumbnailImage	<a href="#">TUIImageBuffer</a>	Thumbnail
iconImage	<a href="#">TUIImageBuffer</a>	Icon
isMinimized	bool	Whether to Minimize

## TUINetworkInfo

Network Quality Information

Field	Type	Description

userId	const char*	User ID
quality	<a href="#">TUINetworkQuality</a>	Network Quality
upLoss	int	Upstream Packet Loss Rate
downLoss	int	Downstream Packet Loss Rate
delay	int	Network Delay

## TUIMessage

### Message

Field	Type	Description
messageId	const char*	Message ID
message	const char*	Message Text
timestamp	int64_t	Message Time
userId	const char*	Message Sender
userName	const char*	Message Sender Nickname
avatarUrl	const char*	Message Sender Avatar

## TUIImageBuffer

### Image Info

Field	Type	Description
buffer	const char*	Image Data Cache Address
length	uint32_t	Length
width	uint32_t	Width
height	uint32_t	Height

# Web

## API Overview

Last updated : 2023-11-16 14:29:38

### TUIRoomKit (UI included Component)

#### TUIRoomKit API List

API	Description
<a href="#">init</a>	Initialize TUIRoomKit Component
<a href="#">createRoom</a>	Host creates room
<a href="#">enterRoom</a>	Regular member enters room

#### TUIRoomKit Event type

Event	Description
<a href="#">onCreateRoom</a>	Room event callback for creating room
<a href="#">onEnterRoom</a>	Entered room callback
<a href="#">onDestroyRoom</a>	Destroy room callback
<a href="#">onExitRoom</a>	Regular member exits room callback
<a href="#">onKickedOutOfRoom</a>	Regular member is kicked out of room by host notification
<a href="#">onKickedOffLine</a>	User account logged in elsewhere, kicked off line
<a href="#">onUserSigExpired</a>	User UserSig expired, please re-obtain UserSig

### TUIRoomEngine (No UI interface)

#### TUIRoomEngine API List

#### TUIRoomEngine Static method

API	Description

<a href="#">once</a>	Listen to TUIRoomEngine ready event. <b>Note: All methods except TUIRoomEngine.init must be executed after listening to the TUIRoomEngine ready event and the TUIRoomEngine.init method is executed successfully.</b>
<a href="#">login</a>	Login to TUIRoomEngine
<a href="#">setSelfInfo</a>	Set the current user's basic information (username, user avatar)
<a href="#">getSelfInfo</a>	Get the current user's basic information (username, user avatar)
<a href="#">logout</a>	Logout from TUIRoomEngine

### RoomEngine Room Management API

API	Description
<a href="#">createRoom</a>	Create room
<a href="#">enterRoom</a>	Entered room
<a href="#">destroyRoom</a>	Close the room
<a href="#">exitRoom</a>	Leave room
<a href="#">fetchRoomInfo</a>	Get room data
<a href="#">updateRoomNameByAdmin</a>	Update the room's name (Only group owner or administrator can call)
<a href="#">updateRoomSpeechModeByAdmin</a>	Update the room's speech mode (Only group owner or administrator can call)
<a href="#">getUserList</a>	Get the current room user list
<a href="#">getUserInfo</a>	Get user Learn more

### roomEngine Audio Video API

API	Description
<a href="#">setLocalVideoView</a>	Set the view control for local user video rendering
<a href="#">openLocalCamera</a>	Open local camera



<a href="#">closeLocalCamera</a>	Close local camera
<a href="#">openLocalMicrophone</a>	Open local microphone
<a href="#">closeLocalMicrophone</a>	Close local microphone
<a href="#">updateVideoQuality</a>	Update local video codec quality settings
<a href="#">updateAudioQuality</a>	Update local audio codec quality settings
<a href="#">startScreenSharing</a>	Start screen sharing
<a href="#">stopScreenSharing</a>	End screen sharing
<a href="#">startPushLocalVideo</a>	Start pushing local video
<a href="#">stopPushLocalVideo</a>	Stop pushing local video
<a href="#">startPushLocalAudio</a>	Start pushing local audio
<a href="#">stopPushLocalAudio</a>	Stop pushing local audio
<a href="#">setRemoteVideoView</a>	Set the view control for remote user video rendering
<a href="#">startPlayRemoteVideo</a>	Start playing remote user video
<a href="#">stopPlayRemoteVideo</a>	Stop playing remote user video
<a href="#">muteRemoteAudioStream</a>	Mute remote user

### roomEngine Member Management API

API	Description
<a href="#">openRemoteDeviceByAdmin</a>	Request remote user to open media device
<a href="#">applyToAdminToOpenLocalDevice</a>	Participant applies to the host to open the device
<a href="#">closeRemoteDeviceByAdmin</a>	Close remote user's media device
<a href="#">cancelRequest</a>	Cancel the sent request
<a href="#">responseRemoteRequest</a>	Reply to remote user's request
<a href="#">changeUserRole</a>	Change user's role
<a href="#">kickRemoteUserOutOfRoom</a>	Kick user out of the room
<a href="#">disableDeviceForAllUserByAdmin</a>	Disable/Enable all users' media devices

<a href="#">disableSendingMessageForAllUser</a>	Disable/Enable all users to send messages
<a href="#">disableSendingMessageByAdmin</a>	Disable/Enable a user to send messages

### roomEngine Microphone Management API

API	Description
<a href="#">setMaxSeatCount</a>	Set room microphone position maximum value
<a href="#">getSeatList</a>	Get microphone position information
<a href="#">takeSeat</a>	Get microphone position
<a href="#">leaveSeat</a>	Release microphone position
<a href="#">takeUserOnSeatByAdmin</a>	Invite others to go live (only room host and administrator can call this method)
<a href="#">kickUserOffSeatByAdmin</a>	Kick others off the microphone position (only room host and administrator can call this method)
<a href="#">lockSeatByAdmin</a>	Lock a microphone position status (only room host and administrator can call this method)

### roomEngine Message Sending API

API	Description
<a href="#">sendTextMessage</a>	Send text message
<a href="#">sendCustomMessage</a>	Send custom message

### roomEngine Device Management API

API	Description
<a href="#">getCameraDevicesList</a>	Get camera device list
<a href="#">getMicDevicesList</a>	Get mic device list
<a href="#">getSpeakerDevicesList</a>	Get speaker device list
<a href="#">setCurrentCameraDevice</a>	Set the camera device to use
<a href="#">setCurrentMicDevice</a>	Set the mic device to use
<a href="#">setCurrentSpeakerDevice</a>	Set the speaker device to use

<a href="#">getCurrentCameraDevice</a>	Get the currently used camera device
<a href="#">getCurrentMicDevice</a>	Get the currently used mic device
<a href="#">getCurrentSpeakerDevice</a>	Get the currently used speaker device
<a href="#">startCameraDeviceTest</a>	Start camera device test
<a href="#">stopCameraDeviceTest</a>	Stop camera device test

### roomEngine Event Listening API

API	Description
<a href="#">on</a>	Listen to <a href="#">TUIRoomEvents</a> event
<a href="#">off</a>	Cancel listening to <a href="#">TUIRoomEvents</a> event

### roomEngine Other API

API	Description
<a href="#">getTRTCCloud</a>	Get trtcCloud instance
<a href="#">getTIM</a>	Get tim instance

### TUIRoomEngine Event Type

TUIRoomEvent is the Callback Event class corresponding to TUIRoomEngine. You can listen to the Callback Events you are interested in through this callback.

EVENT	Description
<a href="#">TUIRoomEvents.onError</a>	Error Event
<a href="#">TUIRoomEvents.onKickedOutOfRoom</a>	Kick out of room event
<a href="#">TUIRoomEvents.onKickedOffline</a>	User Kicked Offline Event
<a href="#">TUIRoomEvents.onUserSigExpired</a>	User Credential Timeout Event
<a href="#">TUIRoomEvents.onRoomDismissed</a>	Room Dismissed Event
<a href="#">TUIRoomEvents.onRoomNameChanged</a>	Room Name Change Event
<a href="#">TUIRoomEvents.onRoomSpeechModeChanged</a>	Room Microphone Control Mode Change

<a href="#">TUIRoomEvents.onAllUserCameraDisableChanged</a>	All Users' Cameras Disabled in Room Event
<a href="#">TUIRoomEvents.onAllUserMicrophoneDisableChanged</a>	All Users' Microphones Disabled in Room Event
<a href="#">TUIRoomEvents.onSendMessageForAllUserDisableChanged</a>	All Users' Text Message Sending Disabled in Room Event
<a href="#">TUIRoomEvents.onRoomMaxSeatCountChanged</a>	Maximum number of microphone slots in the room modification event
<a href="#">TUIRoomEvents.onRemoteUserEnterRoom</a>	Remote user Entered room event
<a href="#">TUIRoomEvents.onRemoteUserLeaveRoom</a>	Remote user leaving room event
<a href="#">TUIRoomEvents.onUserRoleChanged</a>	Role change event
<a href="#">TUIRoomEvents.onUserVideoStateChanged</a>	Video Status change event
<a href="#">TUIRoomEvents.onUserAudioStateChanged</a>	Audio Status change event
<a href="#">TUIRoomEvents.onSendMessageForUserDisableChanged</a>	Send message status event
<a href="#">TUIRoomEvents.onUserVoiceVolumeChanged</a>	Volume change event
<a href="#">TUIRoomEvents.onUserNetworkQualityChanged</a>	Network quality change event
<a href="#">TUIRoomEvents.onSeatListChanged</a>	Mic position list change event
<a href="#">TUIRoomEvents.onKickedOffSeat</a>	Kicked off the mic event
<a href="#">TUIRoomEvents.onRequestReceived</a>	Request received event
<a href="#">TUIRoomEvents.onRequestCancelled</a>	Request cancelled event
<a href="#">TUIRoomEvents.onReceiveTextMessage</a>	Receive text message event
<a href="#">TUIRoomEvents.onReceiveCustomMessage</a>	Receive custom message event
<a href="#">TUIRoomEvents.onDeviceChange</a>	Device change event
<a href="#">TUIRoomEvents.onUserScreenCaptureStopped</a>	Screen sharing stopped event When a user uses the built-in browser's stop sharing button to end Screen Sharing, the user will receive the 'onUserScreenCaptureStopped' event to modify the Sharing status.



# TUIRoomKit

Last updated : 2023-11-16 14:21:03

## API Introduction

TUIRoomKit API is a human video conference component with UI interface. By using TUIRoomKit API, you can quickly implement a WeChat-like audio/video call scenario through a simple interface. If you want to experience and debug the video conference effect, please read [the Quick Start Demo](#). If you want to embed our features directly into your project, please read [the Quick Access \(TUIRoomKit\)](#).

## API Overview

Reference the TUIRoomKit component in the page. For example, import the TUIRoomKit component in the `App.vue` component.

TUIRoomKit component divides users into host role and regular member role. The component provides [init](#), [createRoom](#), and [enterRoom](#) methods externally.

Hosts and regular members can initialize the application and user data to the TUIRoomKit component through the [init](#) method. The host can create and join a room through the [createRoom](#) method, and regular members can join the room created by the host through the [enterRoom](#) method.



```
<template>
  <room ref="TUIRoomRef"></room>
</template>

<script setup lang="ts">
import { ref, onMounted } from 'vue';
// Import the TUIRoom Component, make sure the import path is correct
import Room from './TUIRoom/index.vue';
// Get the TUIRoom Component element, used to call the TUIRoom Component method
const TUIRoomRef = ref();
```

```
onMounted(async () => {
  // Initialize the TUIRoom component
  // The host needs to initialize the TUIRoom component before creating a room
  // Regular members need to initialize the TUIRoom component before entering a room
  await TUIRoomRef.value.init({
    // To get the sdkAppId, please refer to Step 1
    sdkAppId: 0,
    // The unique identifier of the user in your business
    userId: '',
    // For local development and debugging, you can quickly generate userSig on the
    // Note that userSig and userId have a one-to-one correspondence
    userSig: '',
    // The nickname used by the user in your business
    userName: '',
    // The avatar URL used by the user in your business
    avatarUrl: '',
  })
  // By default, execute the create room operation, and you can choose to execute t
  await handleCreateRoom();
})

// The host creates a room, and this method is called only when creating a room
async function handleCreateRoom() {
  // roomId is the room number entered by the user, and roomId must be of type stri
  // roomMode includes 'FreeToSpeak' (free speech mode) and 'SpeakAfterTakingSeat'
  // roomParam specifies the default behavior of the user when entering the room (w
  const roomId = '123456';
  const roomMode = 'FreeToSpeak';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
  }
  try {
    await TUIRoomRef.value.createRoom({ roomId, roomName: roomId, roomMode, roomPar
  } catch (error: any) {
    alert('TUIRoomKit.createRoom error: ' + error.message);
  }
}

// Regular members enter the room, and this method is called when regular members e
async function handleEnterRoom() {
  // roomId is the room number entered by the user, and roomId must be of type str
  // roomParam specifies the default behavior of the user when entering the room (
  const roomId = '123456';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
  }
}
```



```
    }
    try {
      await TUIRoomRef.value.enterRoom({ roomId, roomParam });
    } catch (error: any) {
      alert('TUIRoomKit.enterRoom error: ' + error.message);
    }
  }
}
</script>

<style lang="scss">
html, body {
  width: 100%;
  height: 100%;
  margin: 0;
}

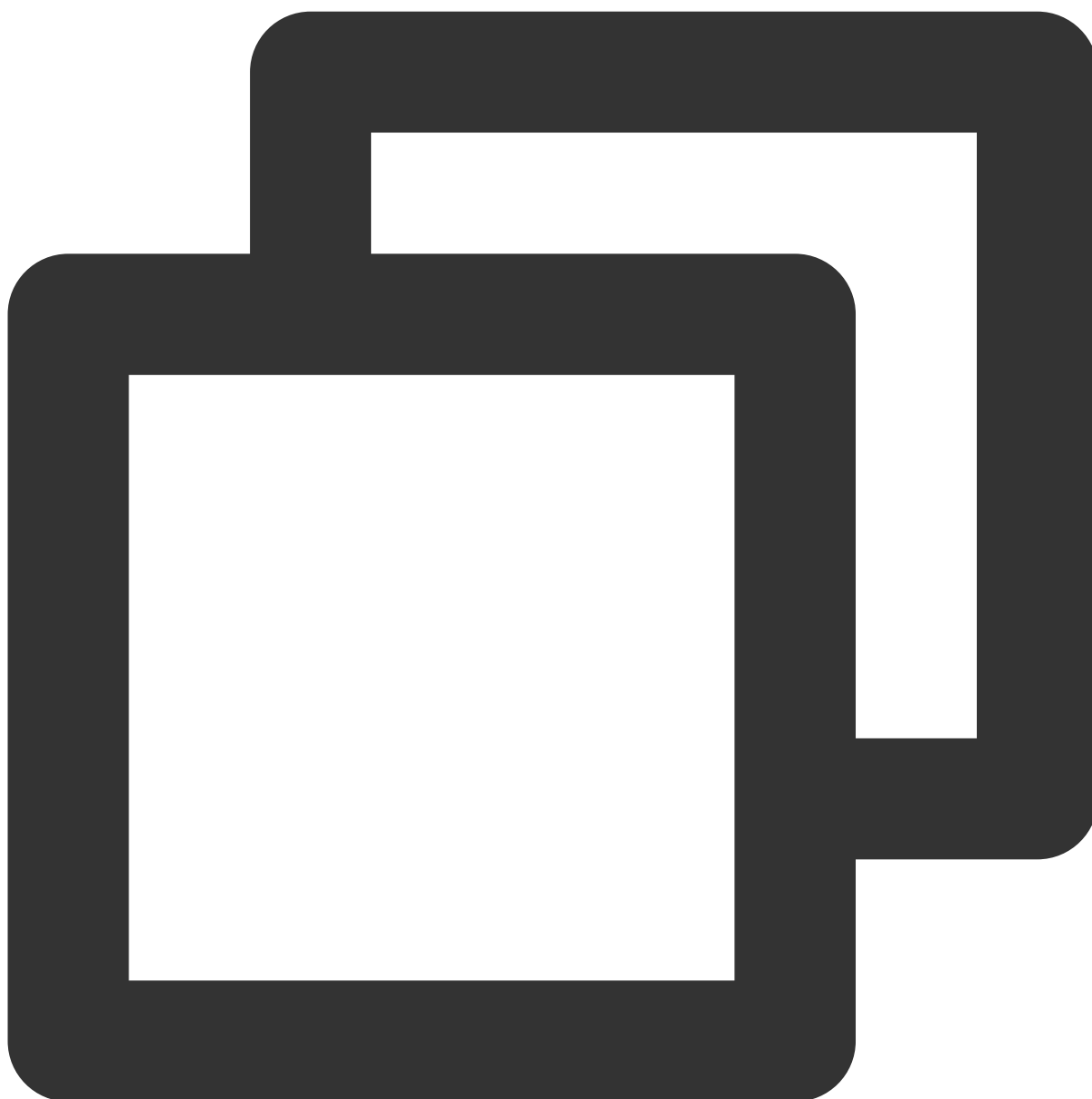
#app {
  width: 100%;
  height: 100%;
  * {
    box-sizing: border-box;
  }
}
</style>
```

## API Details

### TUIRoomkit Interface

#### init

Initialize TUIRoomKit Data, any user using TUIRoomKit needs to call the init method.



```
TUIRoomRef.value.init(roomData);
```

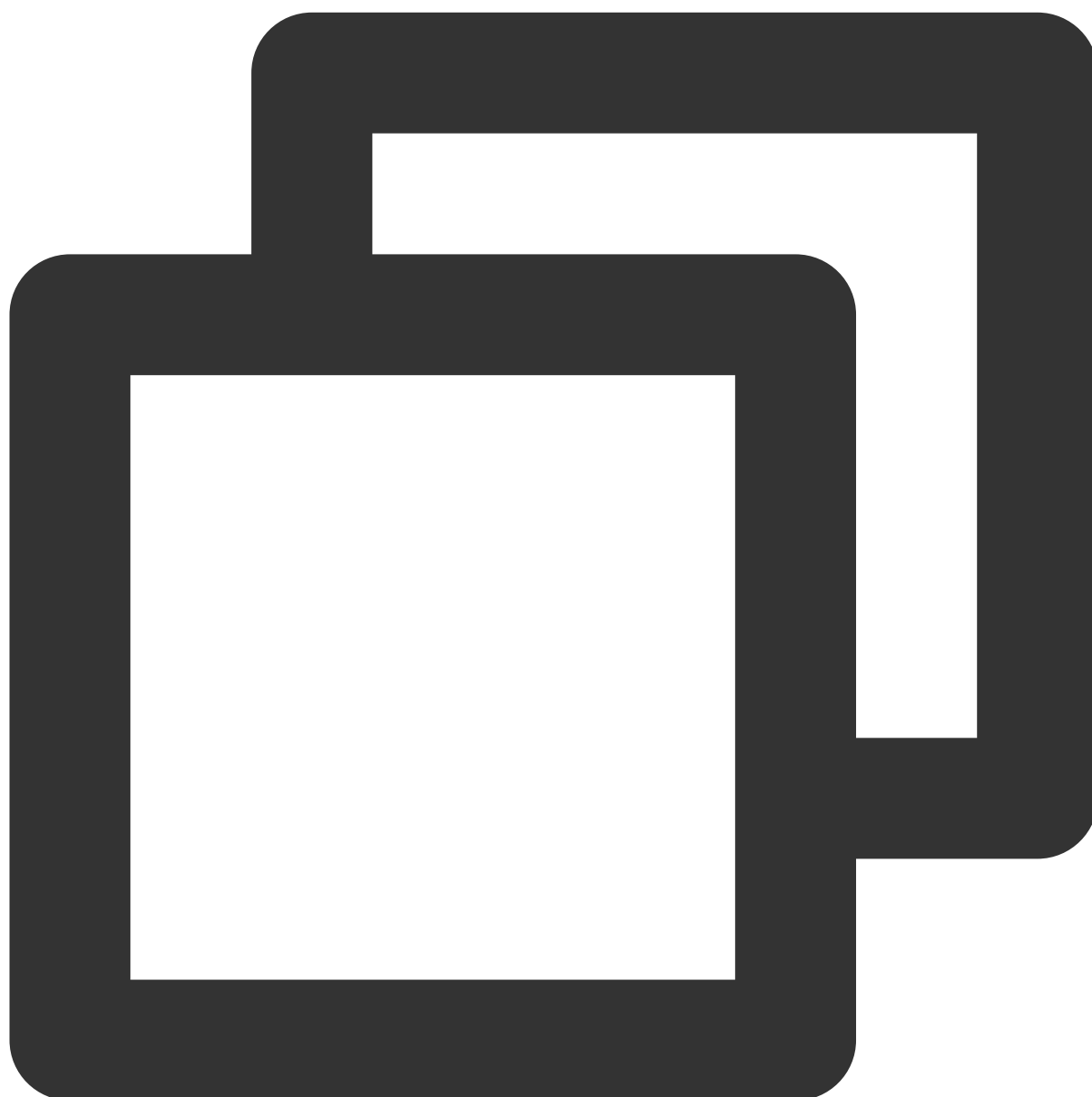
The parameters are shown in the table below:

Parameter	Type	Meaning
roomData	object	-
roomData.sdkAppld	number	Customer's SDKAppld
roomData.userId	string	User's Unique ID

roomData.userSig	string	User's UserSig
roomData.userName	string	User's Nickname
roomData.avatarUrl	string	User's Avatar URL

## createRoom

Host creates a room.



```
TUIRoomRef.value.createRoom(roomId, roomMode, roomParam);
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
roomMode	string	Room mode, FreeToSpeak (Free Speech Mode) and SpeakAfterTakingSeat (Onstage Speech Mode), default is FreeToSpeak
roomParam	Object	Optional
roomParam.isOpenCamera	string	Optional, whether to open the camera when entering the room, default is closed
roomParam.isOpenMicrophone	string	Optional, whether to open the mic when entering the room, default is closed
roomParam.defaultCameraId	string	Optional, default Camera device ID
roomParam.defaultMicrophoneId	string	Optional, default mic device ID
roomParam.defaultSpeakerId	String	Optional, default Speaker device ID

## enterRoom

Regular member joins the room.



```
TUIRoomRef.value.enterRoom(roomId, roomParam);
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
roomParam	Object	Optional
roomParam.isOpenCamera	string	Optional, whether to open the camera when entering the room,

		default is closed
roomParam.isOpenMicrophone	string	Optional, whether to open the mic when entering the room, default is closed
roomParam.defaultCameraId	string	Optional, default Camera device ID
roomParam.defaultMicrophoneId	string	Optional, default mic device ID
roomParam.defaultSpeakerId	String	Optional, default Speaker device ID

## TUIRoomkit Event

### onCreateRoom

Create room Callback.



```
<template>
  <room ref="TUIRoomRef" @on-create-room="onCreateRoom"></room>
</template>

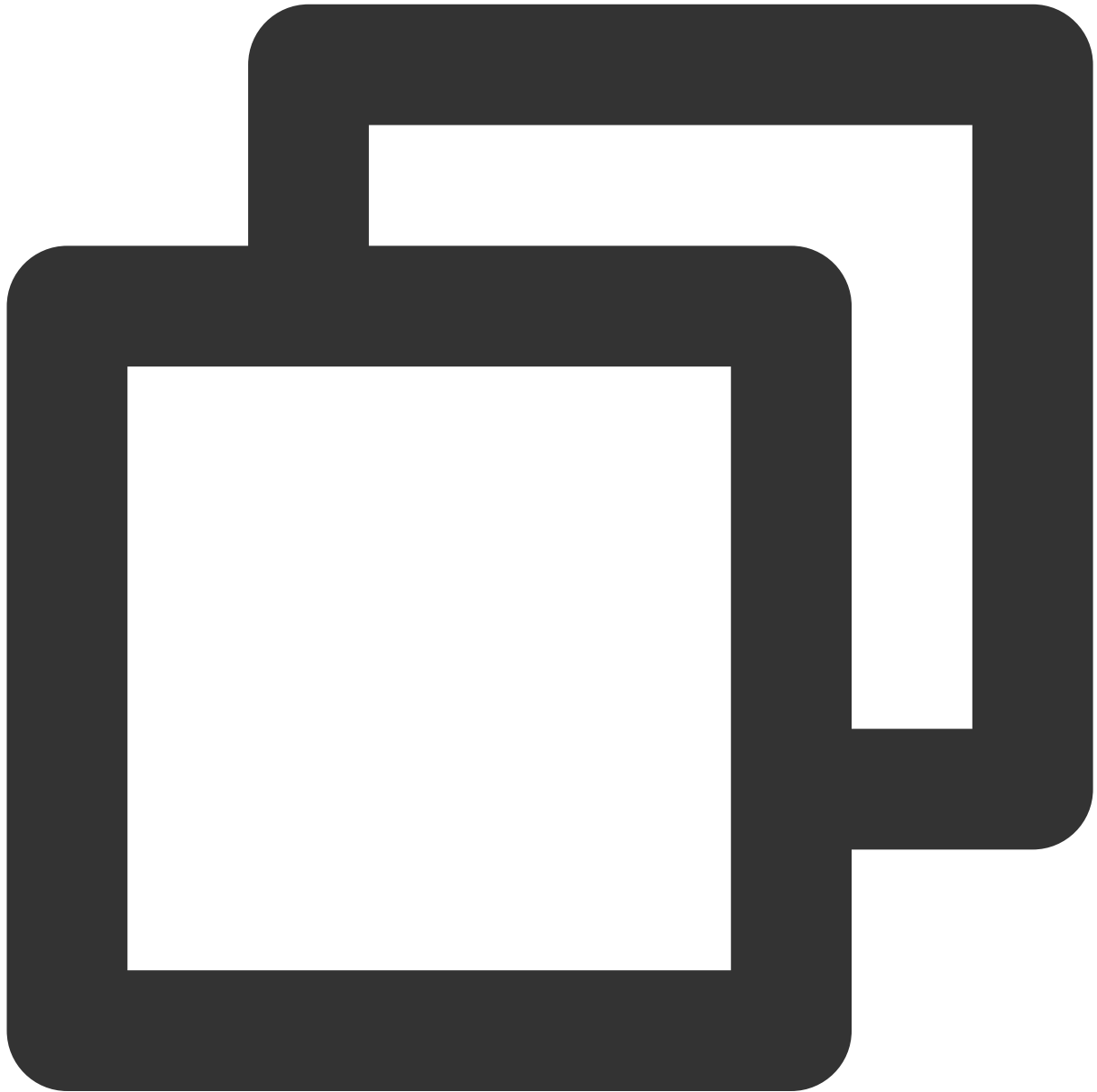
<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onCreateRoom(info) {
    if (info.code === 0) {
      console.log('Create room Success')
    }
  }
</script>
```

```
    }  
  }  
</script>
```

## onEnterRoom

Entered room Callback.



```
<template>  
  <room ref="TUIRoomRef" @on-enter-room="onEnterRoom"></room>  
</template>
```



```
<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onEnterRoom(info) {
    if (info.code === 0) {
      console.log('Entered room Success')
    }
  }
</script>
```

## onDestroyRoom

Host destroys room Callback.



```
<template>
  <room ref="TUIRoomRef" @on-destroy-room="onDestroyRoom"></room>
</template>

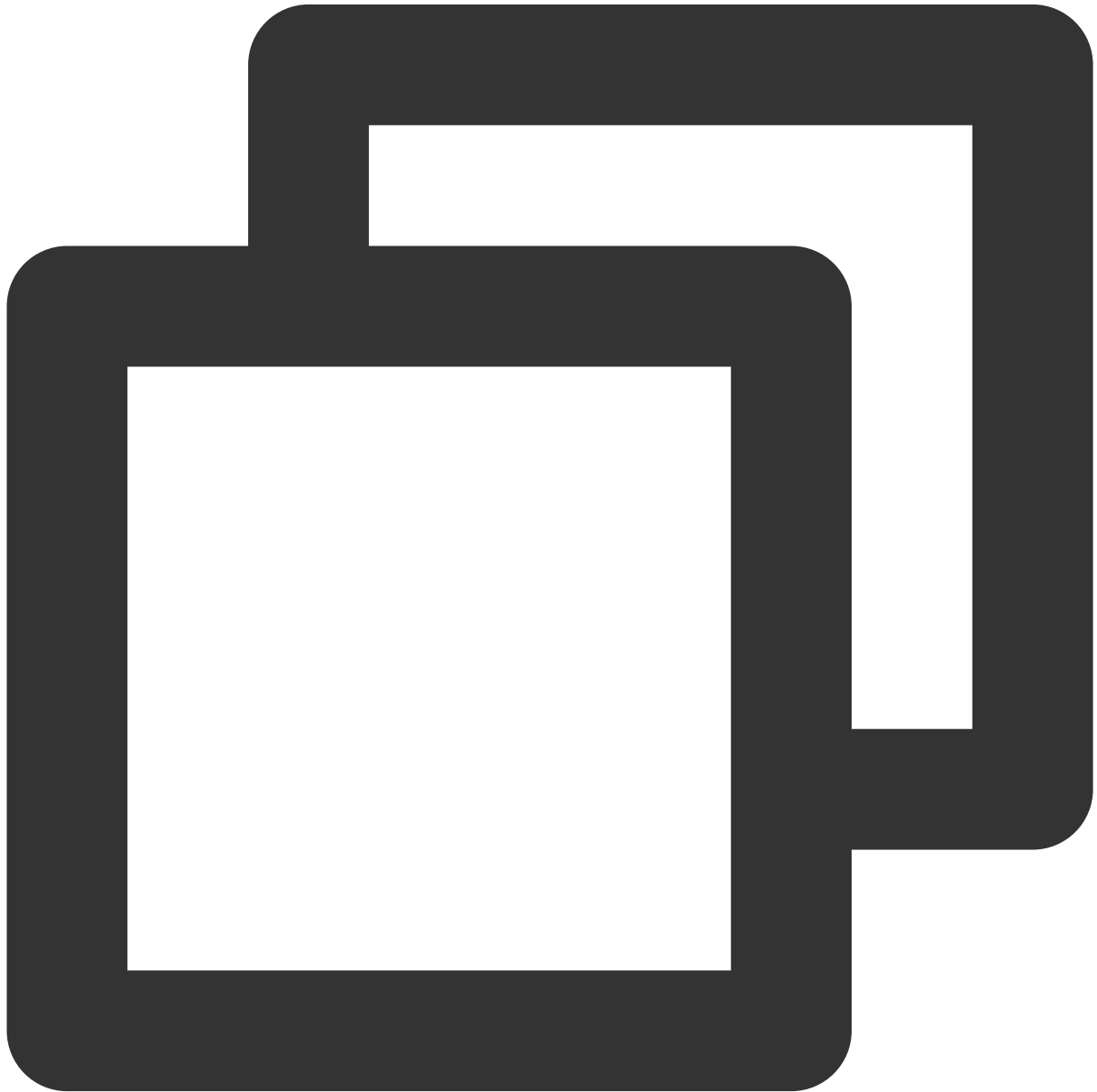
<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onDestroyRoom(info) {
    if (info.code === 0) {
      console.log('Host destroys room Success')
    }
  }
</script>
```

```
}  
}  
</script>
```

## onExitRoom

Regular member exits room Callback.



```
<template>  
  <room ref="TUIRoomRef" @on-exit-room="onExitRoom"></room>  
</template>
```

```
<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onExitRoom(info) {
    if (info.code === 0) {
      console.log('Regular member exits room Success')
    }
  }
</script>
```

## onKickedOutOfRoom

Regular member is kicked out of room by host Notification.



```
<template>
  <room ref="TUIRoomRef" @on-kicked-out-of-room="onKickedOutOfRoom"></room>
</template>

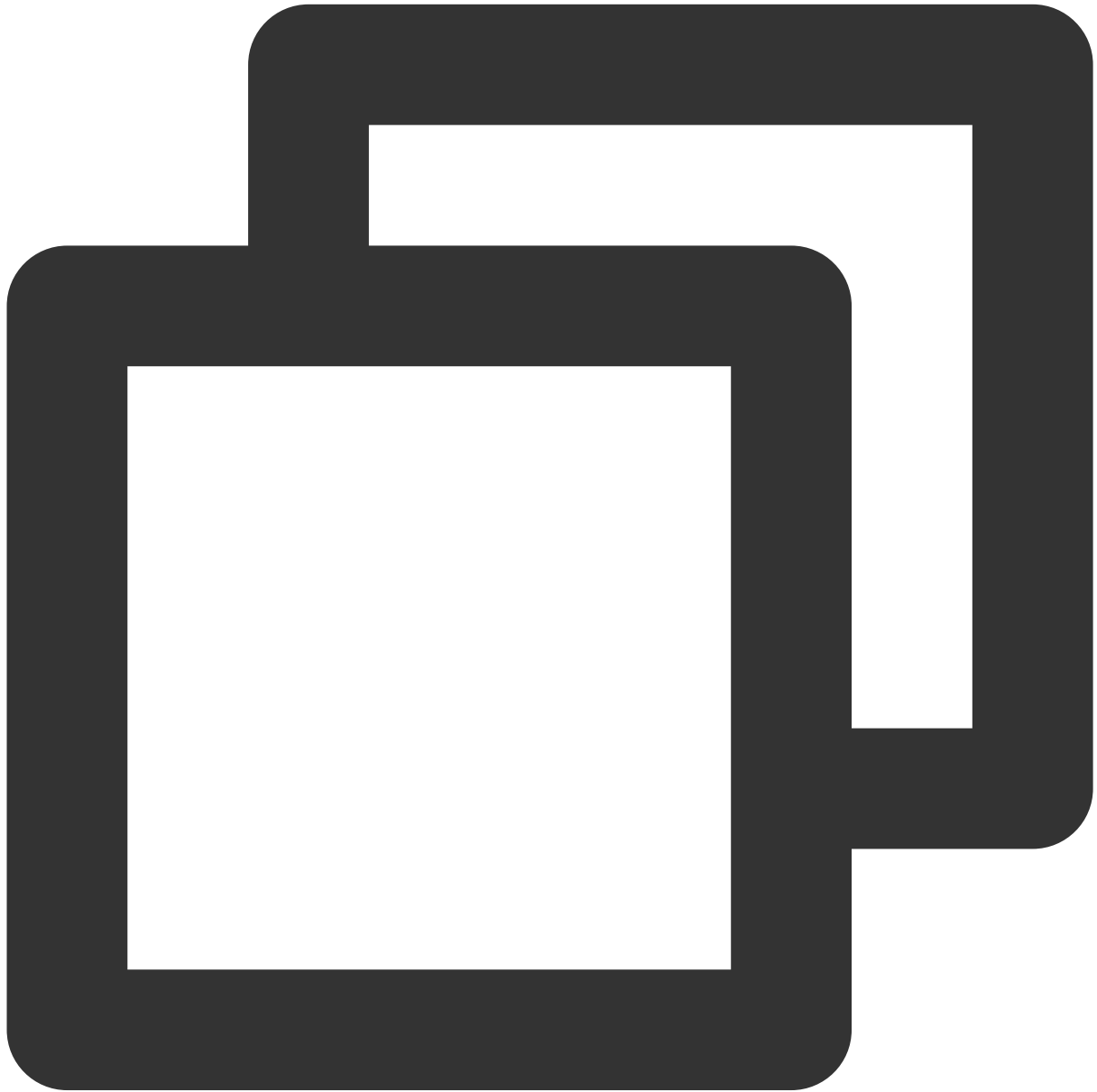
<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onKickedOutOfRoom({ roomId, message }) {
    console.log('Regular member is kicked out of room by host', roomId, message);
  }
</script>
```

```
</script>
```

### onKickedOffLine

User account logged in elsewhere, kicked offline.



```
<template>
  <room ref="TUIRoomRef" @on-kick-off-line="onKickedOffLine"></room>
</template>

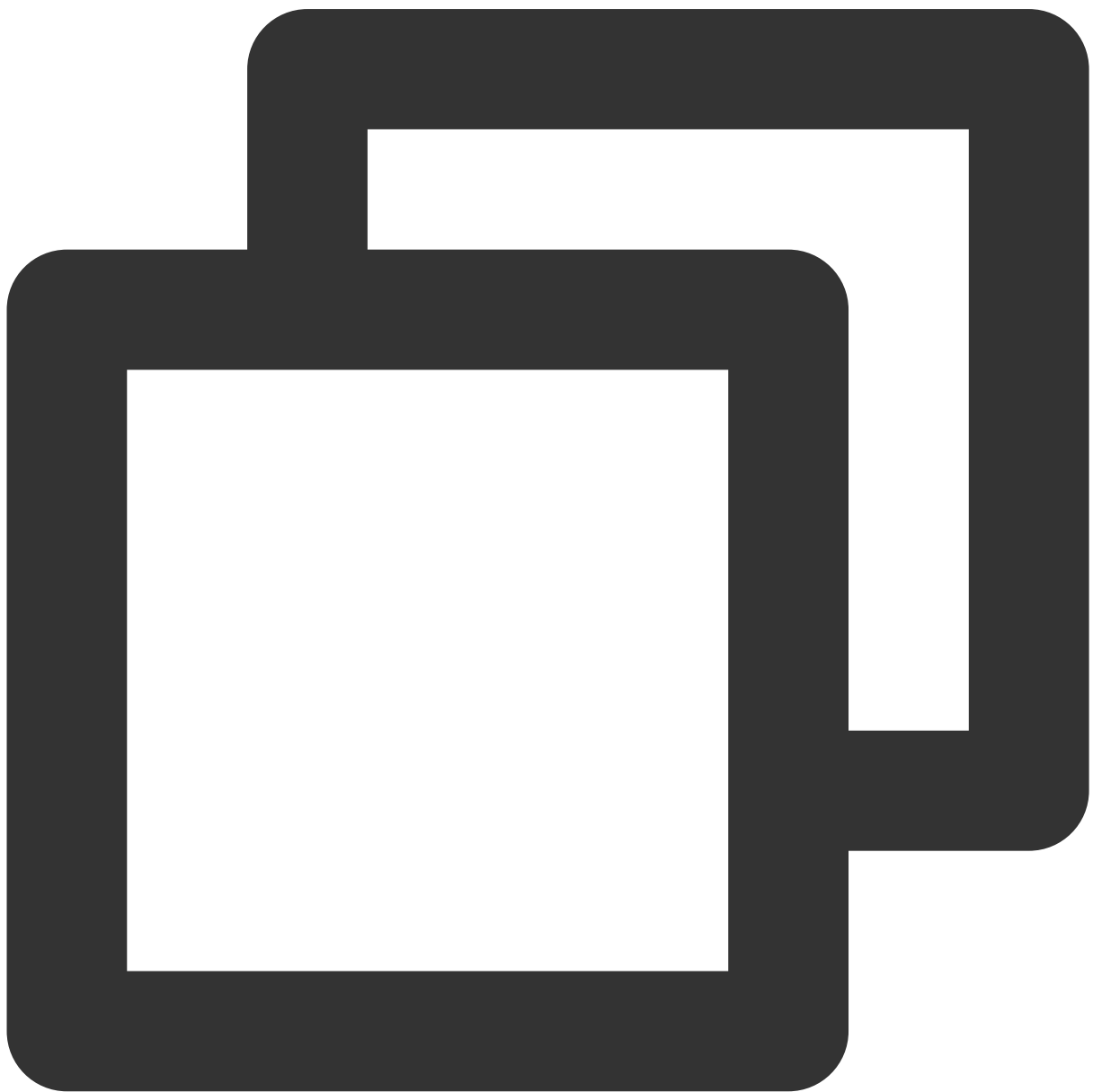
<script setup lang="ts">
```

```
// Import TUIRoom Component, make sure the import path is correct
import Room from './TUIRoom/index.vue';

function onKickedOffline({ message }) {
  console.log('Member kicked offline', message);
}
</script>
```

## onUserSigExpired

User userSig expired, please regenerate userSig.



```
<template>
  <room ref="TUIRoomRef" @on-user-sig-expired="onUserSigExpired"></room>
</template>

<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onUserSigExpired() {
    console.log('User userSig expired, please regenerate userSig.');
```



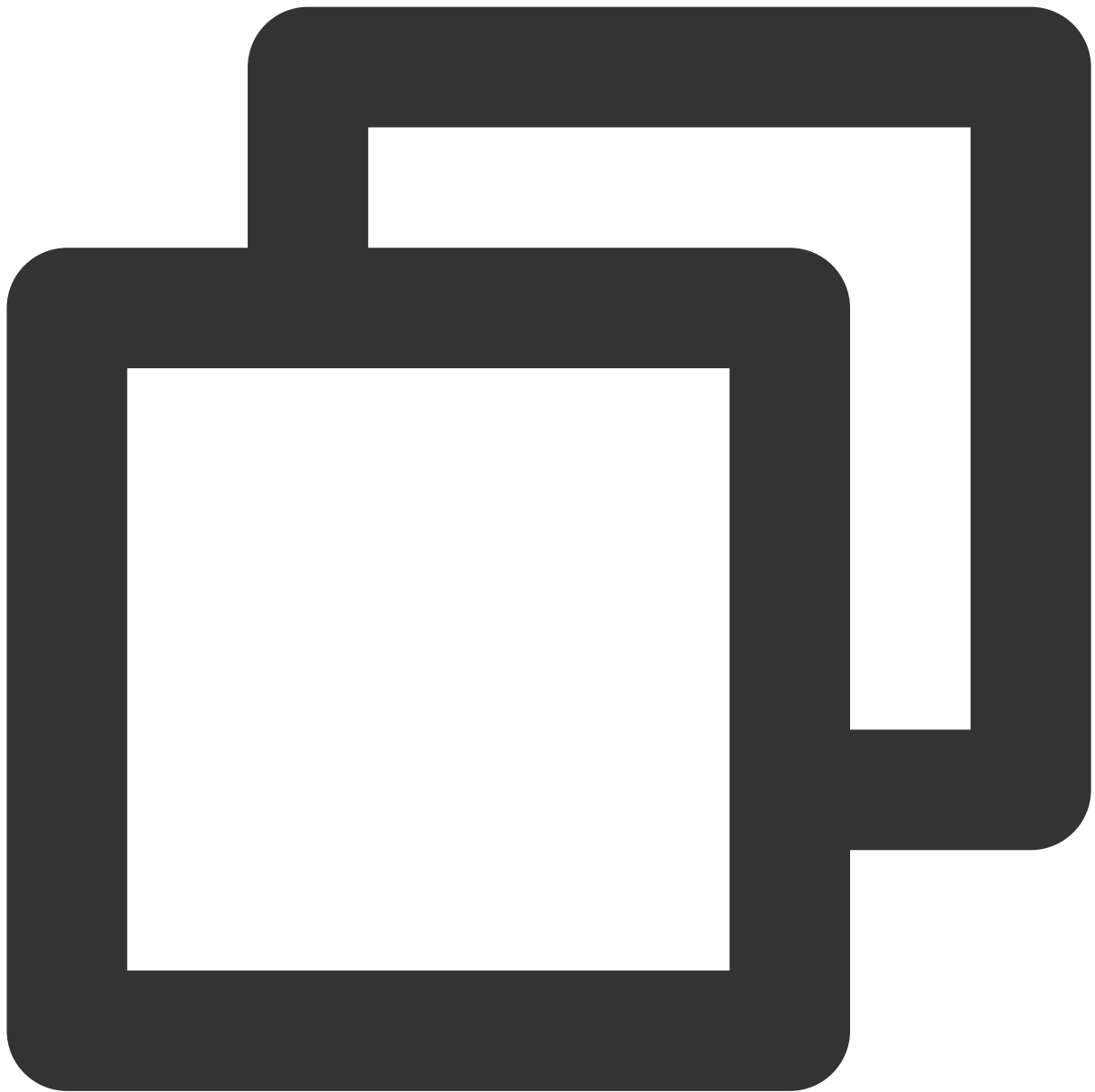
# TUIRoomEngine

Last updated : 2024-01-10 17:26:33

## TUIRoomEngine Introduction

TUIRoomEngine SDK provides Room Management, Multi-person Tencent Real-Time Communication, Screen Sharing, Member Management, Instant Messaging, and other features.

**Installation Method:**



```
// Use npm
npm i @tencentcloud/tuiroom-engine-js --save

// Use pnpm
pnpm i @tencentcloud/tuiroom-engine-js --save

// Use yarn
yarn add @tencentcloud/tuiroom-engine-js
```

# TUIRoomEngine API

## TUIRoomEngine Static Method

API	Description
<a href="#">once</a>	<b>Listening to the TUIRoomEngine ready Event.</b> <b>Note: All methods other than TUIRoomEngine.login must be executed after listening to the TUIRoomEngine ready event and the successful execution of the TUIRoomEngine.login method.</b>
<a href="#">login</a>	Login to TUIRoomEngine
<a href="#">setSelfInfo</a>	Setting the current user's Basic information (Username, User Avatar)
<a href="#">getSelfInfo</a>	Get the current user's Basic information (Username, User Avatar)
<a href="#">logout</a>	Logout of TUIRoomEngine

## roomEngine Room Management API

API	Description
<a href="#">createRoom</a>	Create Room
<a href="#">enterRoom</a>	Enter Room
<a href="#">destroyRoom</a>	Destroy Room
<a href="#">exitRoom</a>	Exit Room
<a href="#">fetchRoomInfo</a>	Get Room data
<a href="#">updateRoomNameByAdmin</a>	Update Name (Call by Group Owner or Administrator only)
<a href="#">updateRoomSpeechModeByAdmin</a>	Update Speaking Mode (Call by Group Owner or Administrator only)
<a href="#">getUserList</a>	Get User List
<a href="#">getUserInfo</a>	Learn more about the user

## roomEngine Audio Video API

API	Description

<a href="#">setLocalVideoView</a>	Set Local Stream Rendering Position
<a href="#">openLocalCamera</a>	Capturing Local Camera Video streams
<a href="#">closeLocalCamera</a>	Close Local Camera
<a href="#">openLocalMicrophone</a>	Open Local mic
<a href="#">closeLocalMicrophone</a>	Close Local mic
<a href="#">updateVideoQuality</a>	Set Local Video Parameters
<a href="#">updateAudioQuality</a>	Set Local Audio Parameters
<a href="#">startScreenSharing</a>	Start Screen Sharing
<a href="#">stopScreenSharing</a>	Stop Screen Sharing
<a href="#">startPushLocalVideo</a>	Start Pushing Local Video streams to Remote
<a href="#">stopPushLocalVideo</a>	Stop Pushing Local Video streams to Remote
<a href="#">startPushLocalAudio</a>	Start Pushing Local Audio Stream to Remote
<a href="#">stopPushLocalAudio</a>	Stop Pushing Local Audio Stream to Remote
<a href="#">setRemoteVideoView</a>	Set Remote Stream Rendering Area
<a href="#">startPlayRemoteVideo</a>	Start Playback Remote User Video streams
<a href="#">stopPlayRemoteVideo</a>	Stop Playback Remote User Video streams
<a href="#">muteRemoteAudioStream</a>	Stop Remote User Audio Stream

## roomEngine Member Management API

API	Description
<a href="#">openRemoteDeviceByAdmin</a>	Request Remote User to Open Media Device
<a href="#">applyToAdminToOpenLocalDevice</a>	Participant Apply to Host to Open Device
<a href="#">closeRemoteDeviceByAdmin</a>	Close Remote User Media Device
<a href="#">cancelRequest</a>	Cancel Sent Request
<a href="#">responseRemoteRequest</a>	Reply to Remote User Request
<a href="#">changeUserRole</a>	Change User Role

<a href="#">kickRemoteUserOutOfRoom</a>	Kick Out User from Room
<a href="#">disableDeviceForAllUserByAdmin</a>	Disable/Enable All Users' Media Device
<a href="#">disableSendingMessageForAllUser</a>	Disallow/Allow All Users to Send Message
<a href="#">disableSendingMessageByAdmin</a>	Disallow/Allow Specific User to Send Message

### roomEngine Mic Position Management API

API	Description
<a href="#">setMaxSeatCount</a>	Set Room Maximum Value
<a href="#">getSeatList</a>	Get Mic Position Information
<a href="#">takeSeat</a>	Get Mic Position
<a href="#">leaveSeat</a>	Release Mic Position
<a href="#">takeUserOnSeatByAdmin</a>	Invite Others to Go Live (Only Room Host and Administrator can invoke this method)
<a href="#">kickUserOffSeatByAdmin</a>	Kick Others Off the Mic (Only Room Host and Administrator can invoke this method)
<a href="#">lockSeatByAdmin</a>	Lock a Specific Mic Position Status (Only Room Host and Administrator can invoke this method)

### roomEngine Message Sending API

API	Description
<a href="#">sendTextMessage</a>	Send Text Message
<a href="#">sendCustomMessage</a>	Send Custom Message

### roomEngine Device Management API

API	Description
<a href="#">getCameraDevicesList</a>	Get Camera Device List
<a href="#">getMicDevicesList</a>	Get Mic Device List
<a href="#">getSpeakerDevicesList</a>	Get Speaker Device List

<a href="#">setCurrentCameraDevice</a>	Set to Use Camera
<a href="#">setCurrentMicDevice</a>	Set to Use Mic
<a href="#">setCurrentSpeakerDevice</a>	Set to Use Speaker
<a href="#">getCurrentCameraDevice</a>	Get the Currently Used Camera
<a href="#">getCurrentMicDevice</a>	Get the Currently Used Mic
<a href="#">getCurrentSpeakerDevice</a>	Get the Currently Used Speaker
<a href="#">startCameraDeviceTest</a>	Start Camera Test
<a href="#">stopCameraDeviceTest</a>	Stop Camera Test

### roomEngine Event Listening API

API	Description
<a href="#">on</a>	Listen to <a href="#">TUIRoomEvents</a>
<a href="#">off</a>	Cancel Listening to <a href="#">TUIRoomEvents</a>

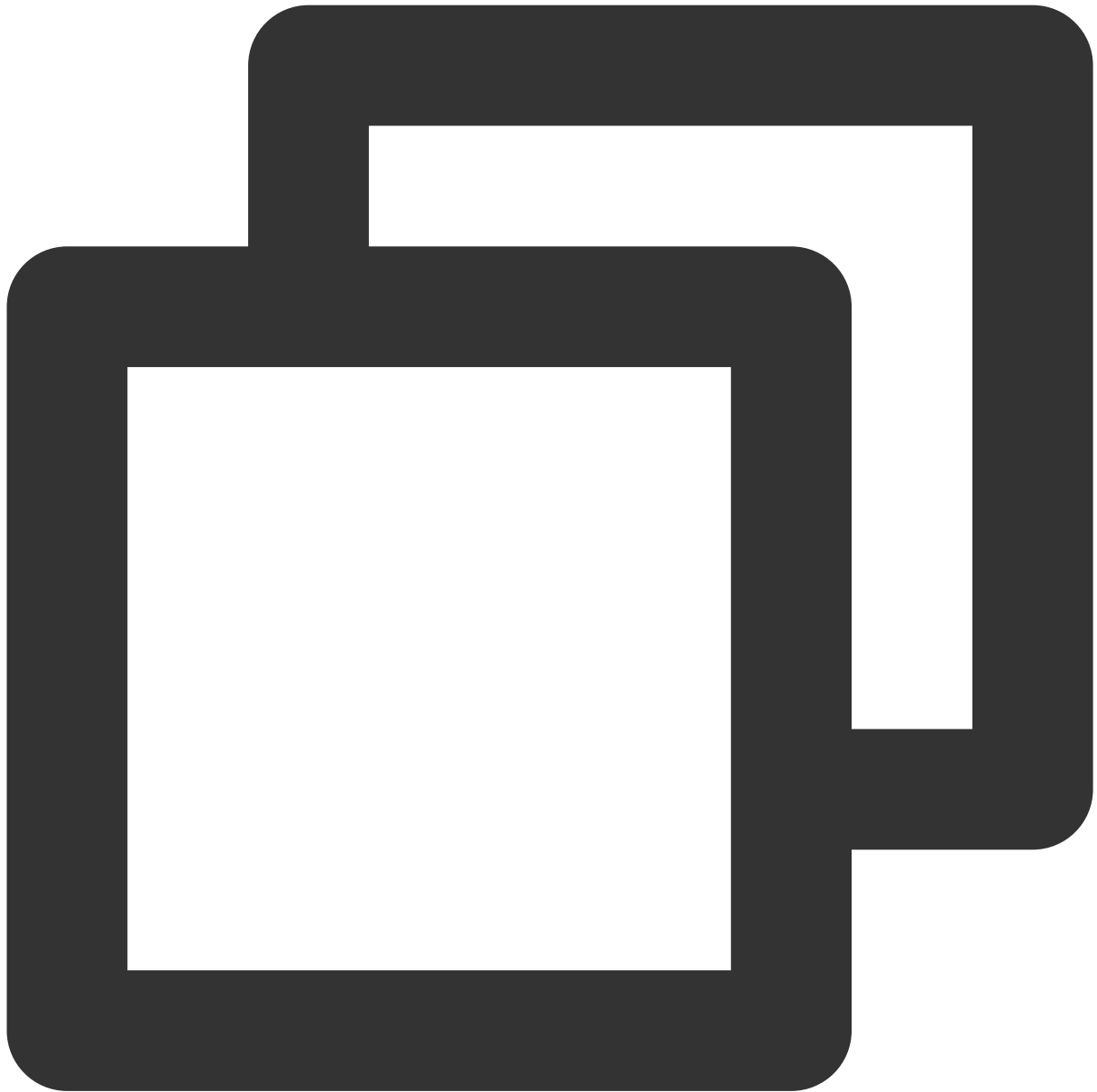
### roomEngine Other API

API	Description
<a href="#">getTRTCCloud</a>	Get trtcCloud Instance
<a href="#">getTIM</a>	Get tim Instance

## API Details

### once

Monitor TUIRoomEngine 'ready' Event



```
TUIRoomEngine.once('ready', () => {
  const roomEngine = new TUIRoomEngine();

  await TUIRoomEngine.login({
    sdkAppId: 0,    // Fill in the sdkAppId you applied for
    userId: '',     // Fill in the userId corresponding to your business
    userSig: '',    // Fill in the userSig calculated by the server or locally
  });

  await roomEngine.createRoom({
    roomId: '12345', // Enter your Room ID, note that the Room ID is required to
```

```
name: 'Test Room',      // Enter your room name, the default room name is roomId
roomType: TUIRoomType.kGroup, // Set the room type to TUIRoomType.kGroup type
});
});
```

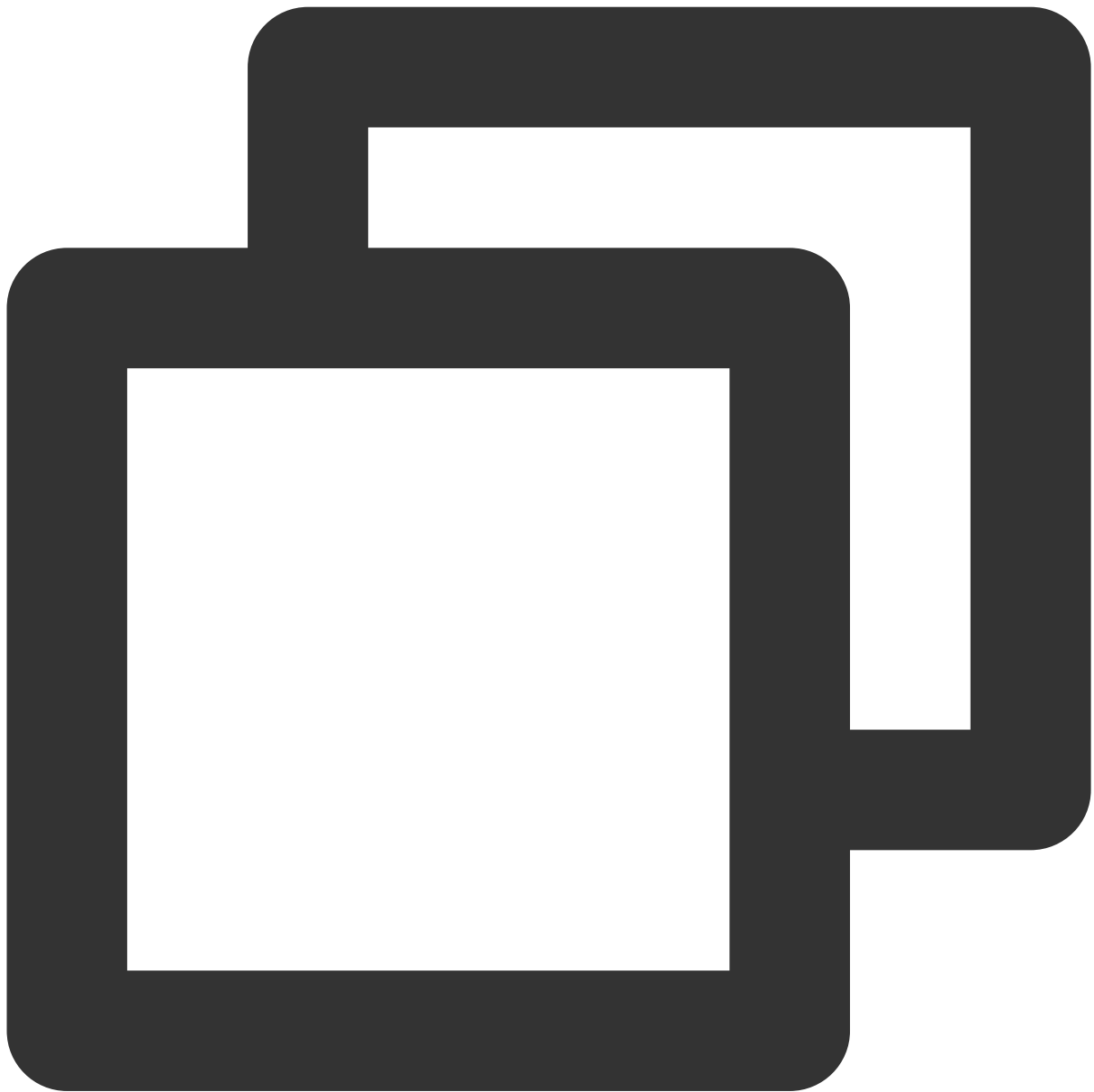
## login

### Description:

In version v1.0.0, this interface is named TUIRoomEngine.init, please use TUIRoomEngine.login to log in TUIRoomEngine in v1.0.1 and above versions.

You must log in to TUIRoomEngine before you can call other methods of TUIRoomEngine and its instances.





```
// Login TUIRoomEngine
await TUIRoomEngine.login({
  sdkAppId: 0,    // Fill in the sdkAppId you applied for
  userId: '',     // Fill in the userId corresponding to your business
  userSig: '',    // Fill in the userSig calculated by the server or locally
});
```

Parameter:

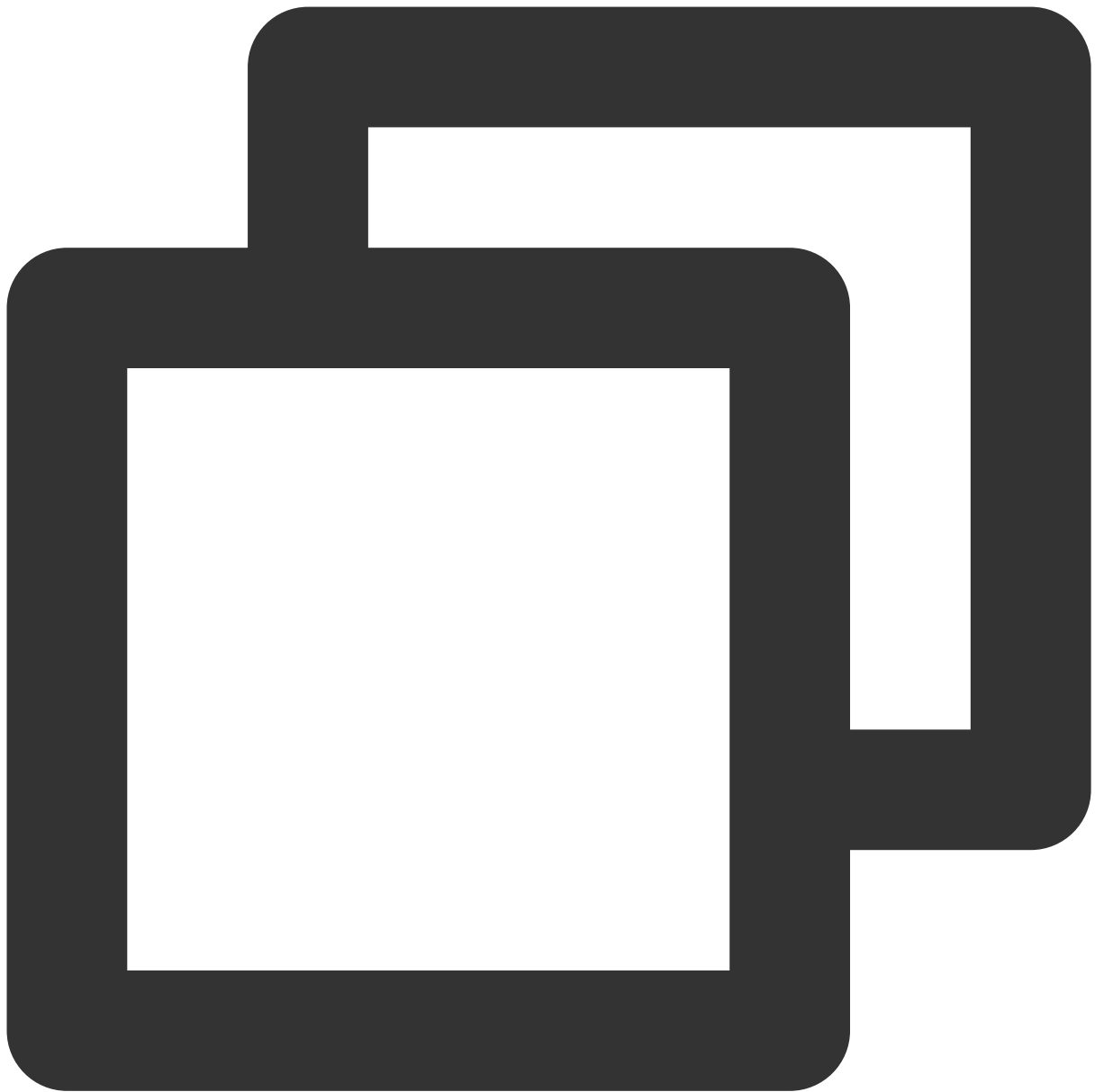
Parameter	Type	Description	Default Value	Meaning
-----------	------	-------------	---------------	---------

sdkAppld	number	Required	-	After clicking Application Management > Create Application in the Tencent Real-Time Communication Console, you can get the sdkAppld information in Application Info.
userId	string	Required	-	It is recommended to limit the user ID length to 32 bytes, and only allow uppercase and lowercase English letters (a-zA-Z), numbers (0-9), underscores, and hyphens.
userSig	string	Required	-	UserSig signature Please refer to <a href="#">UserSig related</a> for the method of calculating userSig.
tim	TIM	Not Required	-	If you want to use more capabilities of the Chat SDK while accessing roomEngine, you can pass the created tim instance into TUIRoomEngine. For the creation method of tim instance, please refer to <a href="#">TIM.create</a> .

Returns *Promise<void>*

## setSelfInfo

Set current user Basic information (Username, User avatar)



```
// Set current user Username and User avatar
await TUIRoomEngine.setSelfInfo({
  userName: '',      // Enter your New username
  avatarUrl: '',     // Enter your New avatar URL
});
```

Parameter:

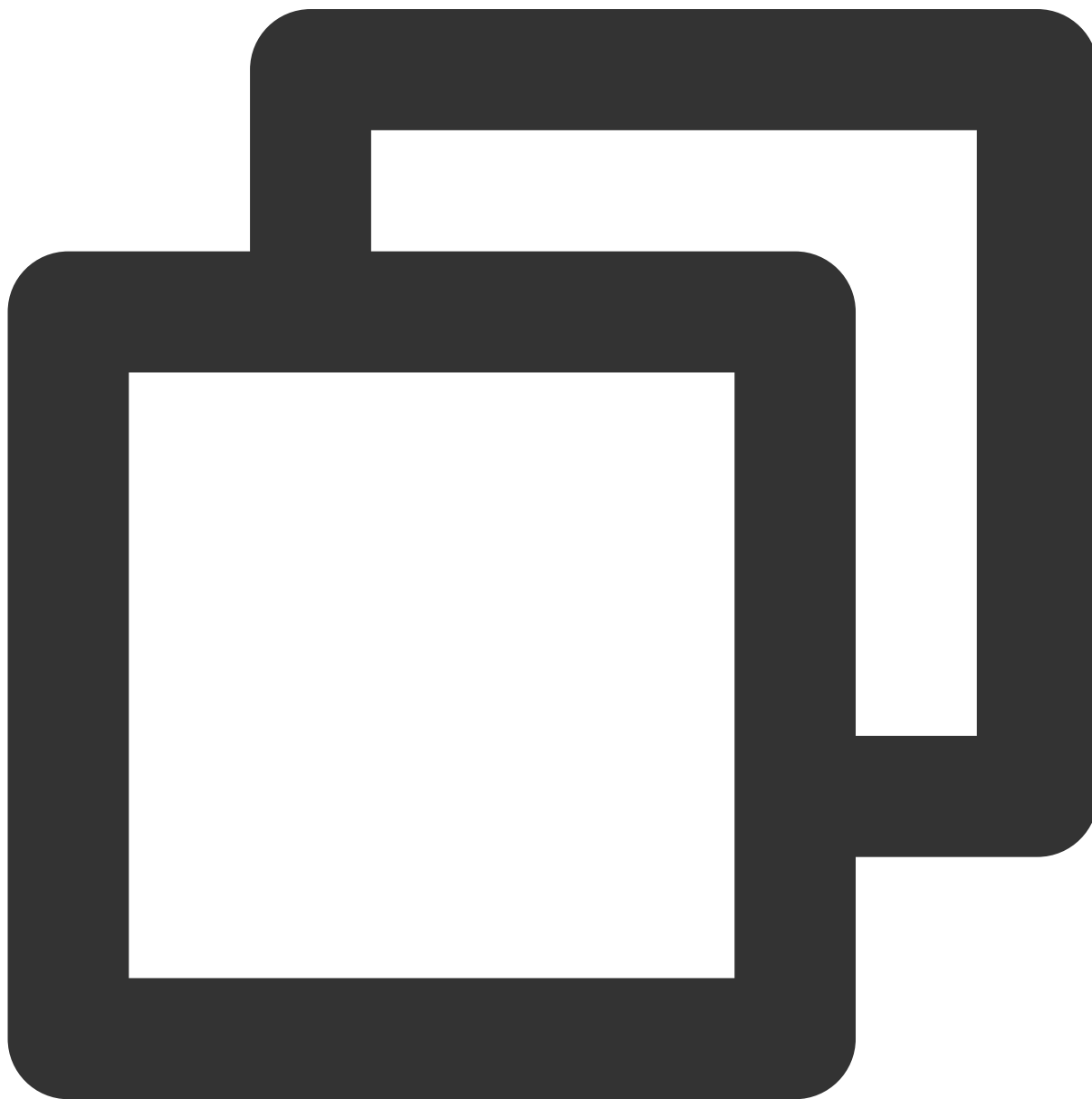
Parameter	Type	Description	Default Value	Meaning
userName	string	Required	-	User Name

avatarUrl	string	Required	-	User avatar
-----------	--------	----------	---	-------------

Returns *Promise<void>*

## getSelfInfo

Get current user Basic information (Username, User avatar)



```
// Get current user Username and User avatar
```

```
const loginUserInfo = await TUIRoomEngine.getSelfInfo();
```

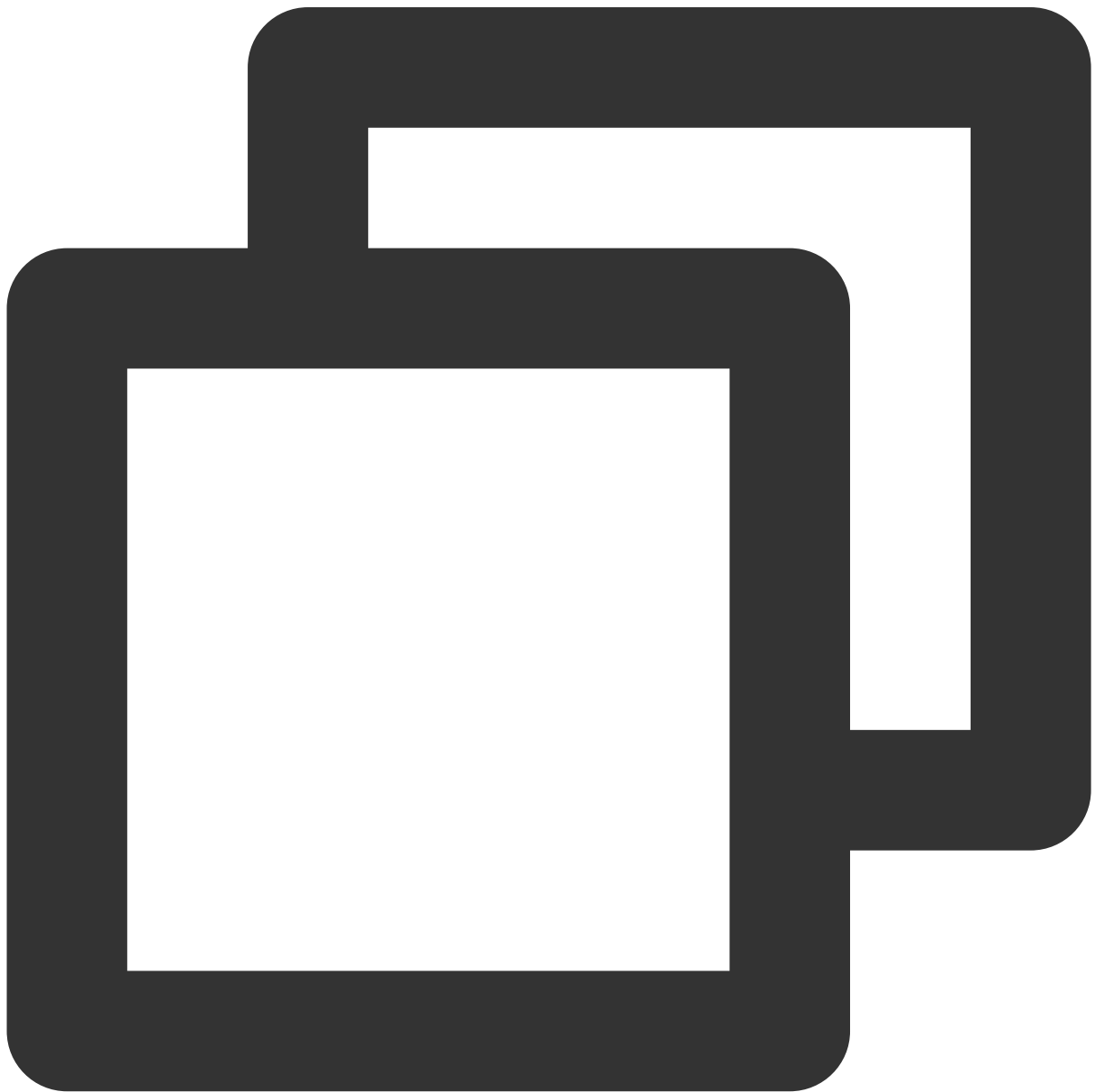
Returns *Promise*<[TUILoginUserInfo](#)> *loginUserInfo*

## logout

### Description:

Interface since v1.0.1 Version support.

Logout TUIRoomEngine

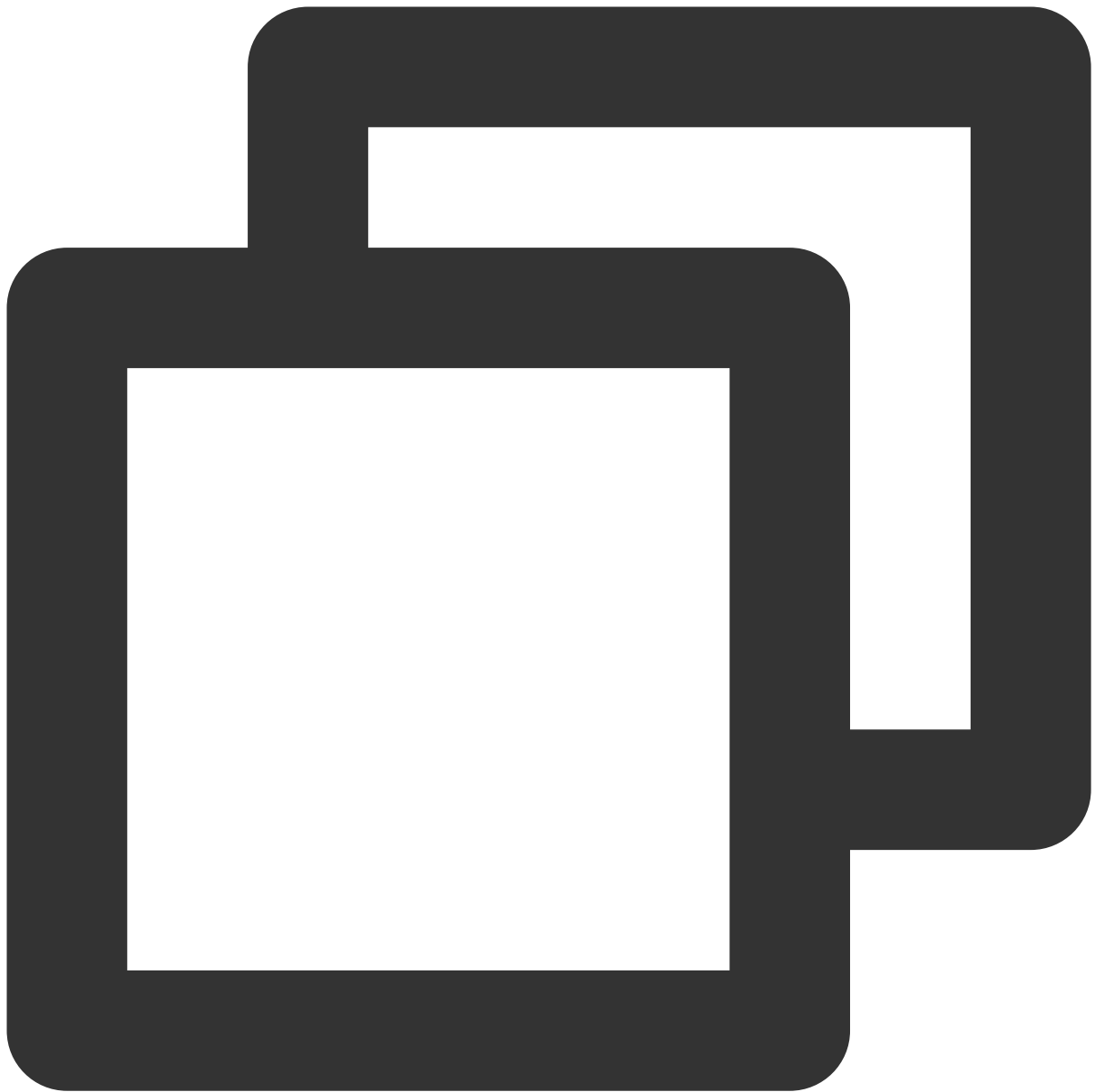


```
// Logout TUIRoomEngine  
await TUIRoomEngine.logout();
```

Returns *Promise<void>*

## createRoom

Host creates room, Call createRoom User is the room owner. When creating a room, you can set Room ID, Room name, Room type, Speaking mode, Whether to allow users to join and enable audio and video, Send message and other functions.



```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({
  roomId: '12345',    // Enter your Room ID, note that the Room ID must be a string
  roomName: 'Test Room', // Enter your Room Name, the default Room Name is roomName
  roomType: TUIRoomType.kConference, // Set the Room Type to TUIRoomType.kConference
  speechMode: TUISpeechMode.kFreeToSpeak, // Set the speech mode to free speech mode
  isMicrophoneDisableForAllUser: false, // Allow users to turn on their mic when join
  isCameraDisableForAllUser: false, // Allow users to turn on their Camera when join
  isMessageDisableForAllUser: false, // Allow users to send messages when join
});
```

Parameter:

Parameter	Type	Description	Default Value	Mea
roomId	string	Required	-	Roo byte char Engl Nurr Spa { }   ·
roomName	string	Optional	roomId	Roo canr
roomType	<a href="#">TUIRoomType</a>	Optional	TUIRoomType.kConference	Roo Offic rem room TUI E-cc room TUI
speechMode	<a href="#">TUISpeechMode</a>	Optional	TUISpeechMode.kFreeToSpeak	Spe For (edu Set : TUI: can defa Set : TUI: do n defa to a or m and mod Set : TUI: user get p and For broa

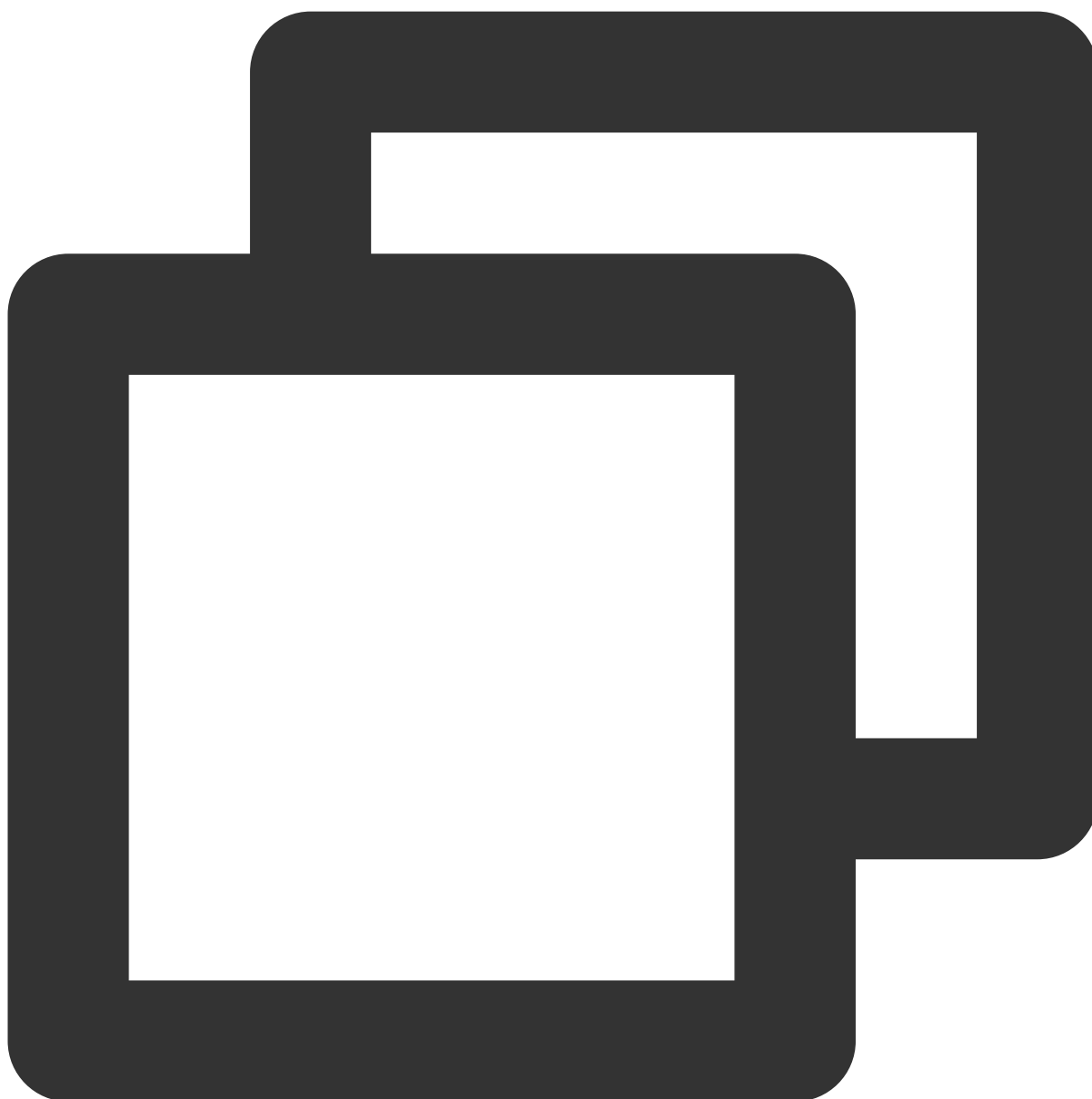


				Set : TUI: for h Set : TUI: host TUI: TUI: mod
isMicrophoneDisableForAllUser	boolean	Optional	false	Enal mute
isCameraDisableForAllUser	boolean	Optional	false	Enal not e
isMessageDisableForAllUser	boolean	Optional	false	Allov proh
maxSeatCount	number	Optional	-	Max For` (edu is nc For` broa max
enableCDNStreaming	boolean	Optional	false	Enal
cdnStreamDomain	string	Optional	"	Pusl

Returns *Promise<void>*

## enterRoom

Entered room interface



```
const roomEngine = new TUIRoomEngine();
const roomInfo = await roomEngine.enterRoom({
  roomId: '12345',
});
```

Parameter:

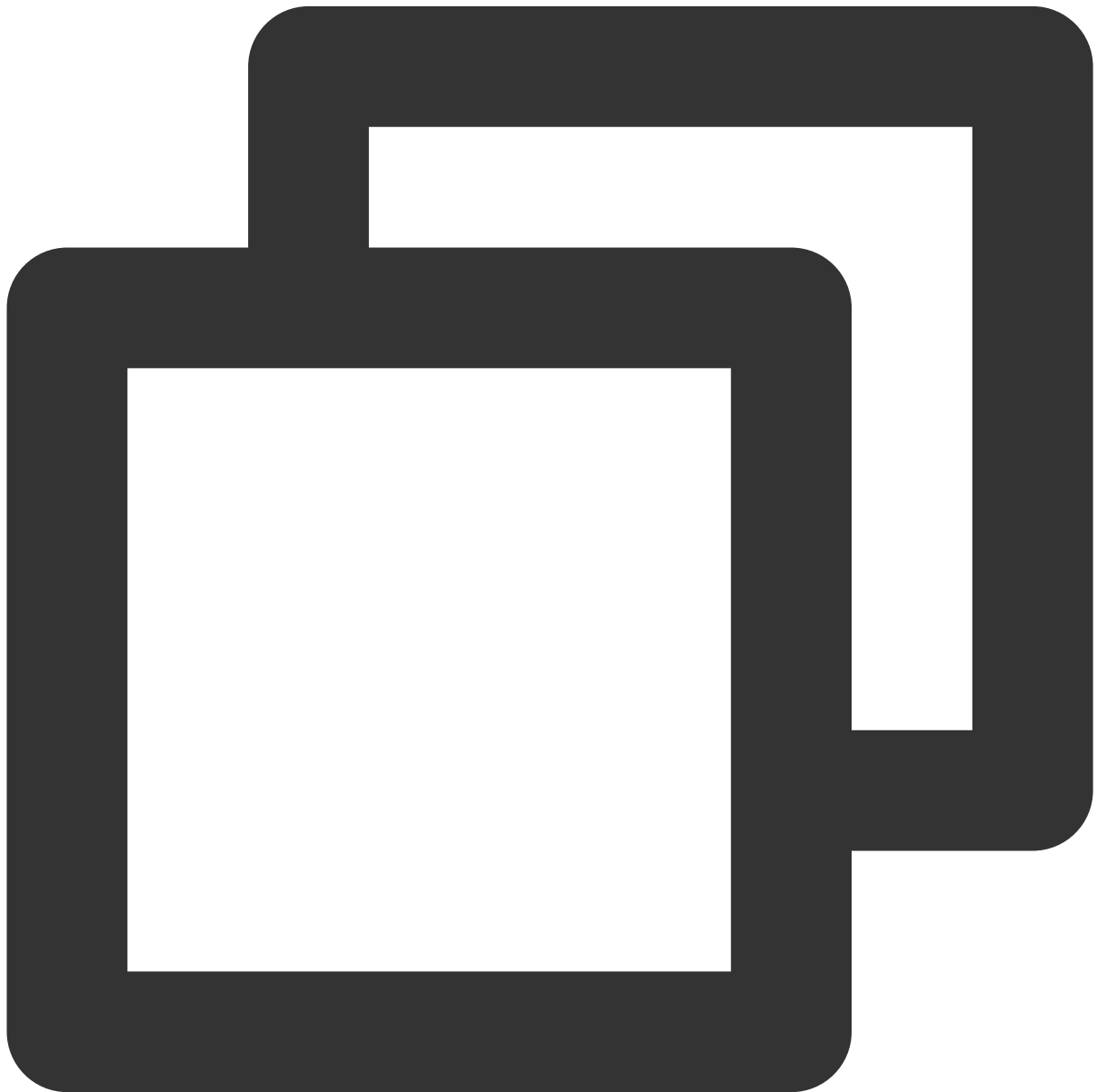
Parameter	Type	Description	Default Value	Meaning
roomId	string	Required	-	Room ID

Returns Promise<[TUIRoomInfo](#)> roomInfo

This interface returns the current Room data

## destroyRoom

Close the room interface, the room must be closed by the room owner, and the room cannot be entered after it is closed.



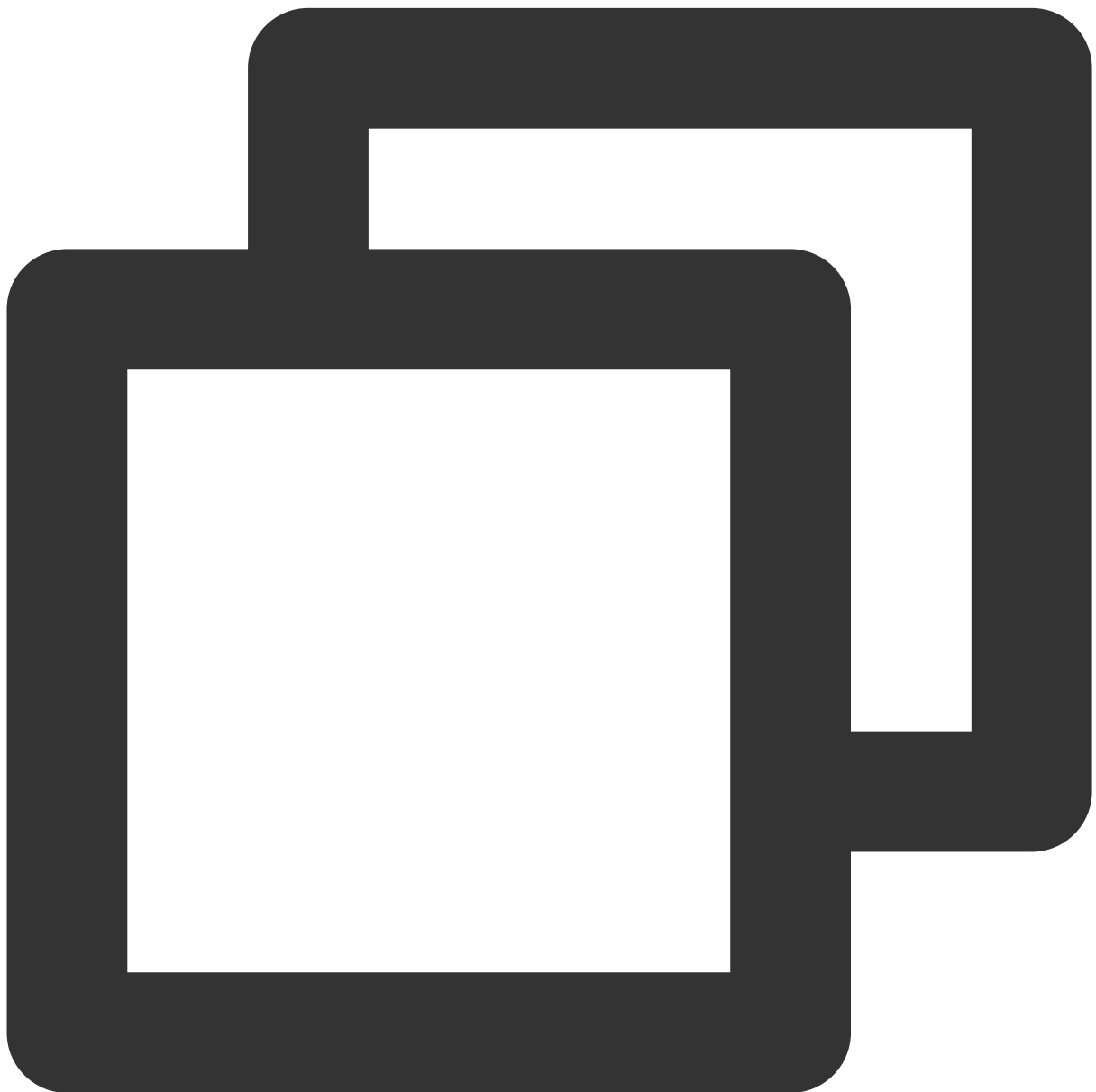
```
const roomEngine = new TUIRoomEngine();
```

```
await roomEngine.destroyRoom();
```

Returns *Promise<void>*

## exitRoom

Leave the room interface, users can leave the room through exitRoom after executing enterRoom.

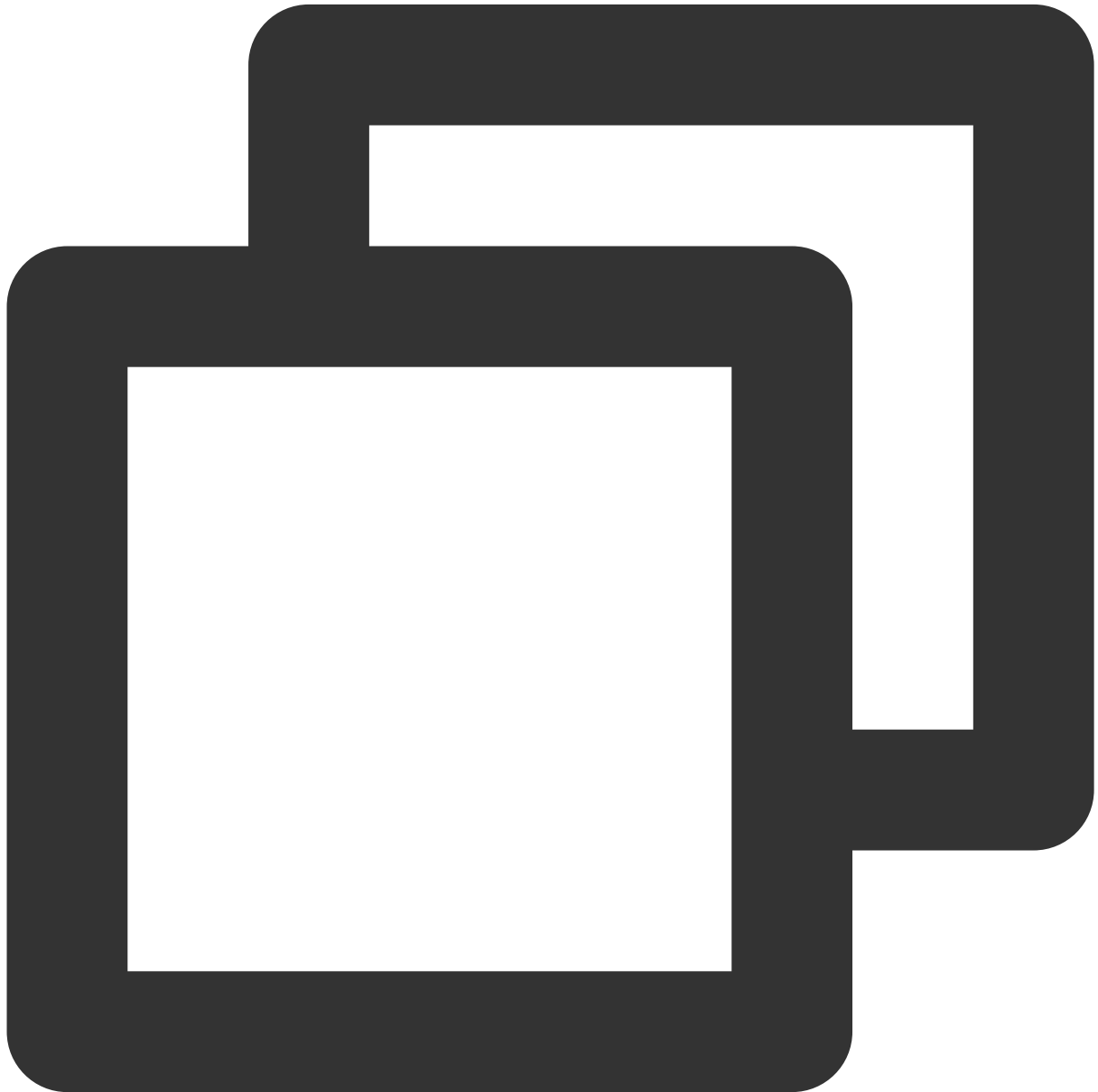


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.exitRoom();
```

Returns *Promise<void>*

## fetchRoomInfo

Get Room information

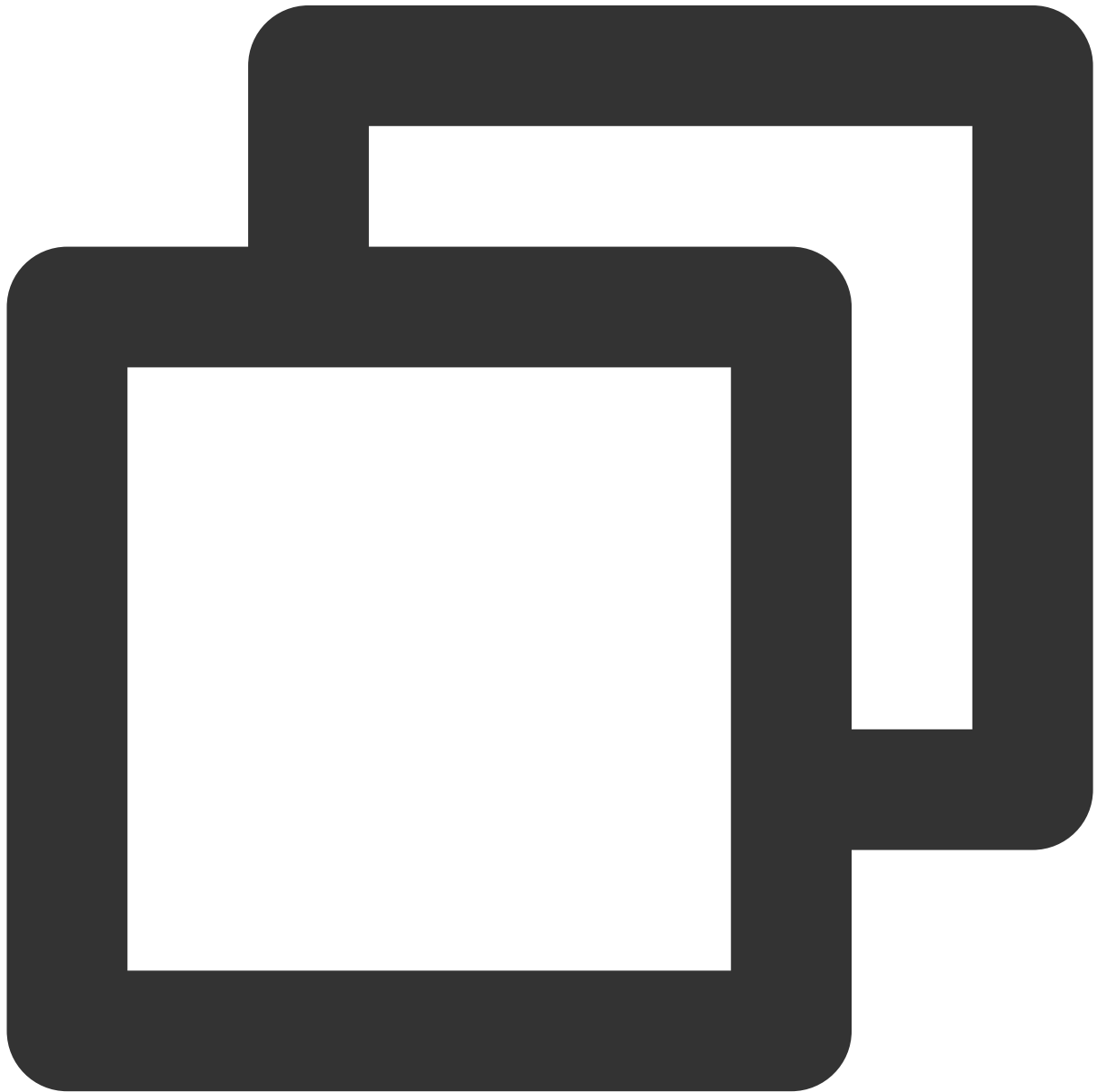


```
const roomEngine = new TUIRoomEngine();  
const roomInfo = roomEngine.fetchRoomInfo();
```

Returns : *Promise*<[TUIRoomInfo](#)> *roomInfo*

## updateRoomNameByAdmin

Update the current room's name (only group owner or admin can invoke)



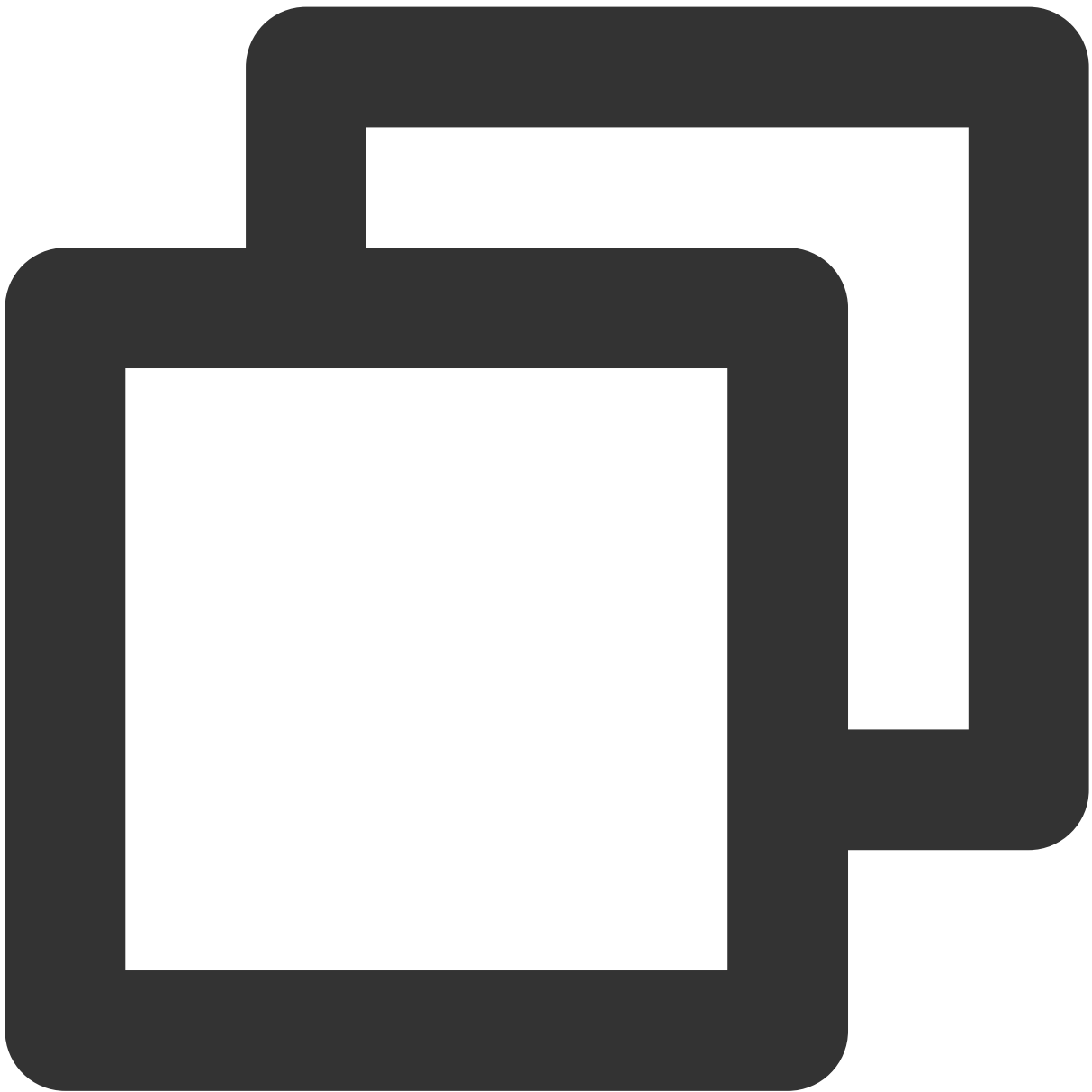
```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.updateRoomNameByAdmin({ roomName: 'NewName' });
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
roomName	string	Required	-	Update the room's name, with the requirement that roomName is not an empty string

## updateRoomSpeechModeByAdmin

Update the room's speaking mode (only group owner or admin can invoke)



```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.updateRoomSpeechModeByAdmin({
  speechMode: TUISpeechMode.kSpeakAfterTakingSeat // Update to Go Live Speaking mode
});
```

Parameter:

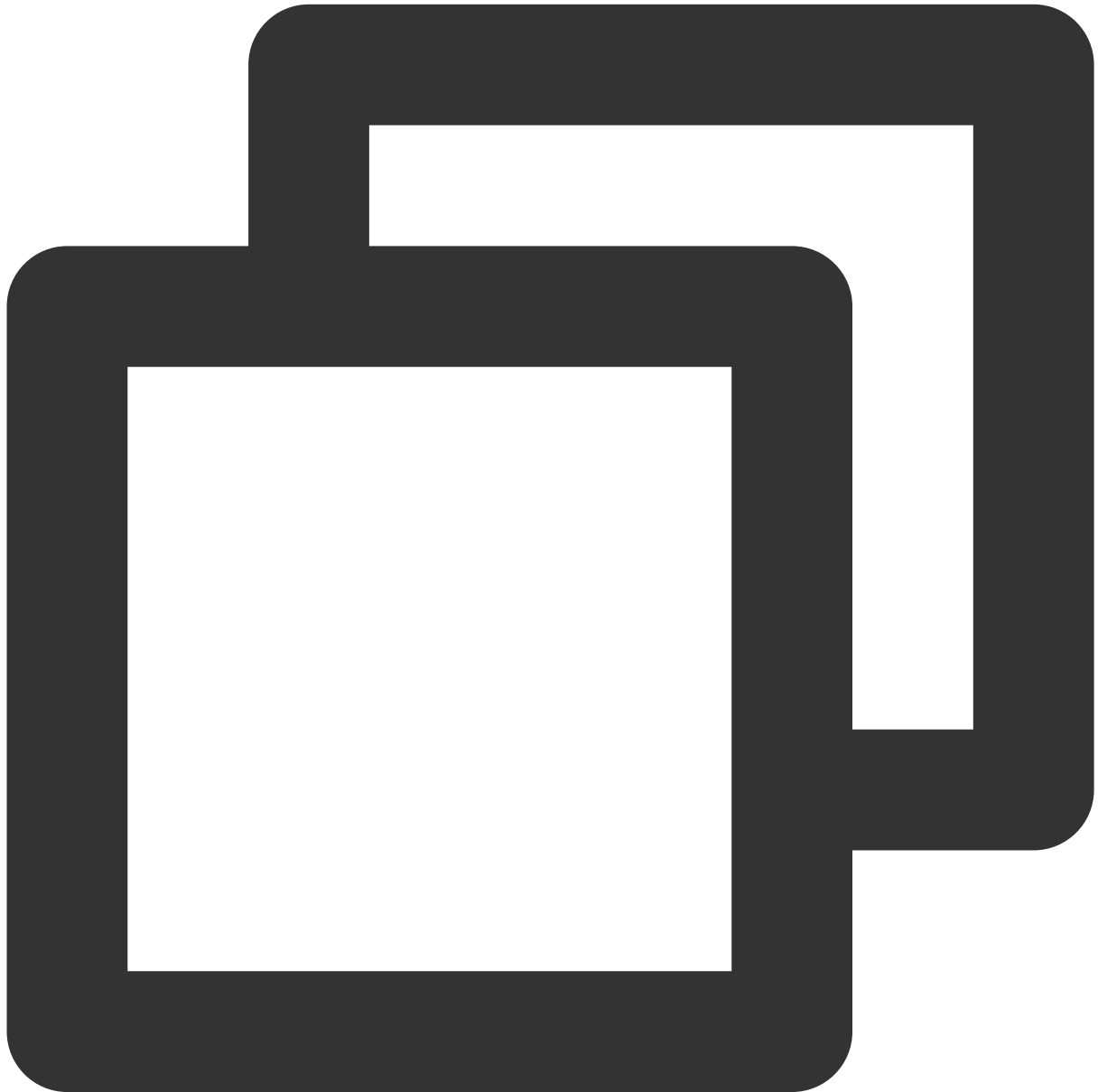
Parameter	Type	Description	Default Value	Meaning



speechMode	TUISpeechMode	Required	-	Update the room's speaking mode
------------	---------------	----------	---	---------------------------------

## getUserList

Get the current room's user list, note that the maximum number of user lists fetched by this interface is 100



```
const roomEngine = new TUIRoomEngine();
const userList = [];
let result;
```

```
do {  
  result = await globalProperties.roomEngine.getUserList();  
  userList.push(...result.userInfoList);  
} while (result.nextSequence !== 0)
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
nextSequence	number	Optional	0	Offset, default is to start fetching users from 0

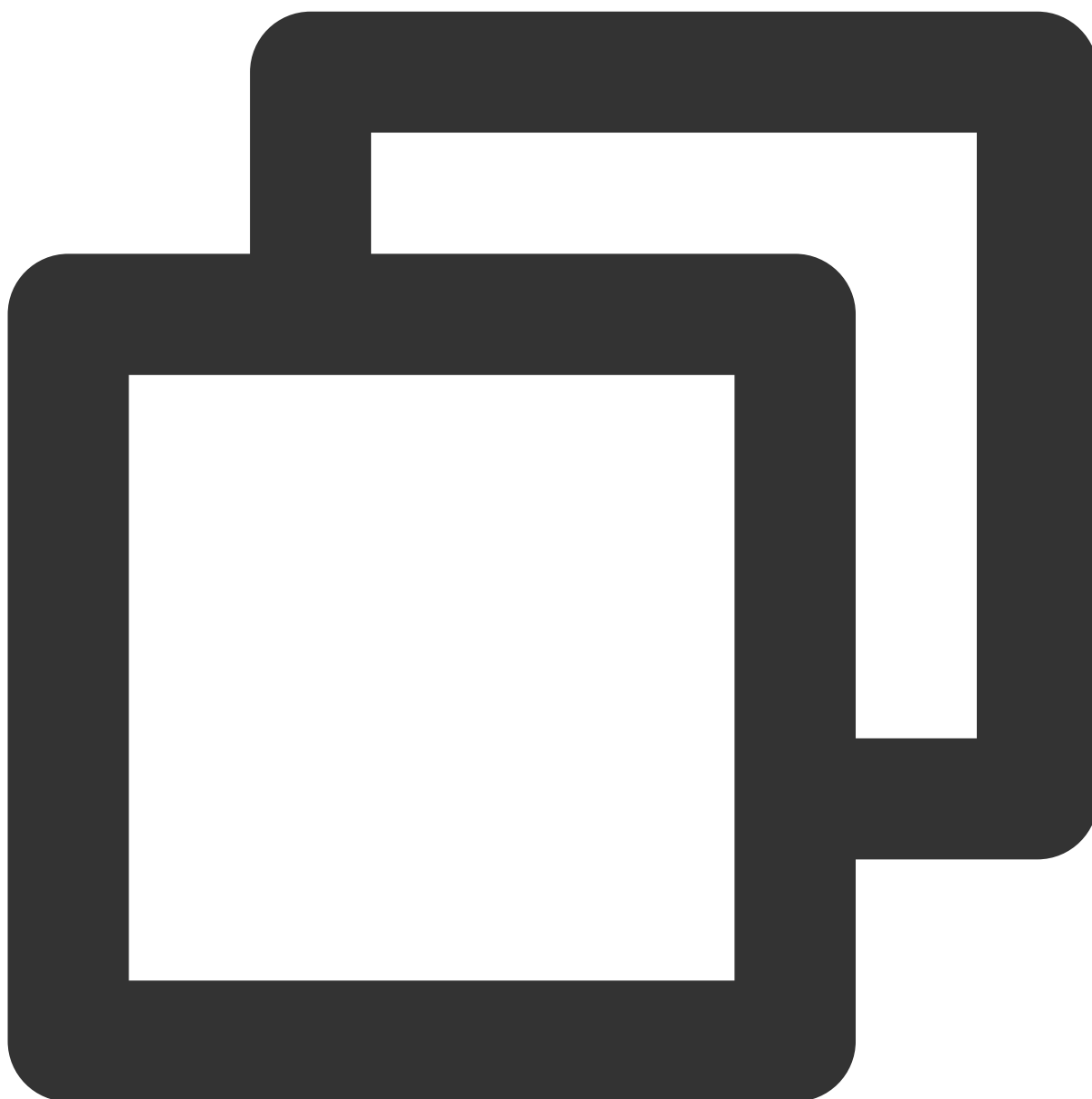
Returns : *Promise<Array> result*

result.nextSequence is the offset for fetching group users next time, if result.nextSequence is 0, it means that all userList have been fetched

result.userInfoList is the userList fetched this time

## getUserInfo

Get the detailed information of the user



```
const roomEngine = new TUIRoomEngine();
const userList = [];
const userInfo = await roomEngine.getUserInfo({
  userId: 'user_12345',
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
-----------	------	-------------	---------------	---------

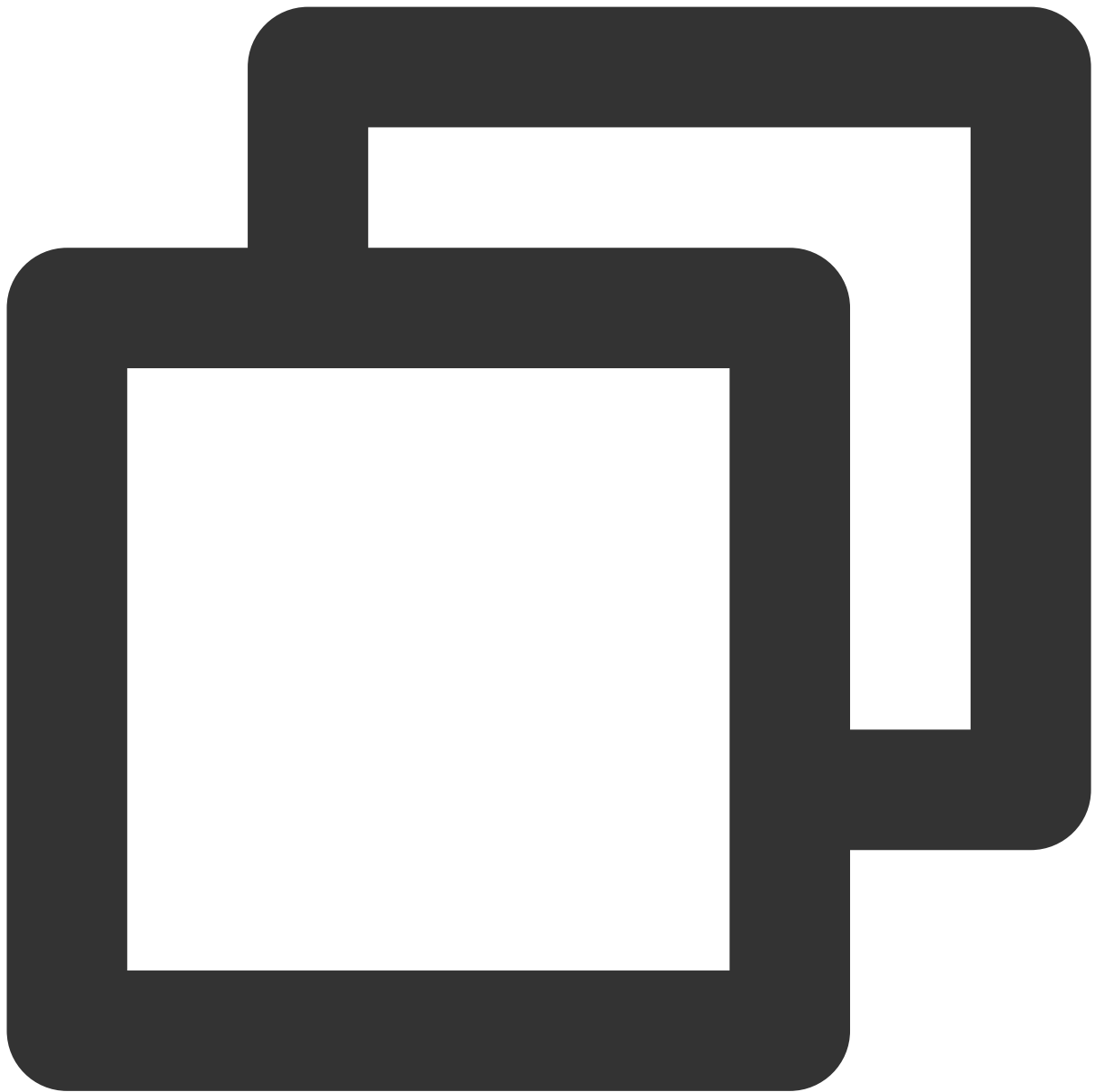
userId	string	Required	-	Get the detailed information of the user according to userId
--------	--------	----------	---	--

Returns : *Promise<Array<[TUIUserInfo](#)>> userInfoList*

This interface returns the user information of the specified user

## setLocalVideoView

Set the rendering position of the local stream



```
const roomEngine = new TUIRoomEngine();

// Set the playback area of the local camera stream to the div element with id 'pre
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'preview-camera',
});

// Set the playback area of the local screen sharing stream to the div element with
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kScreenStream,
```

```
view: 'preview-screen',  
});
```

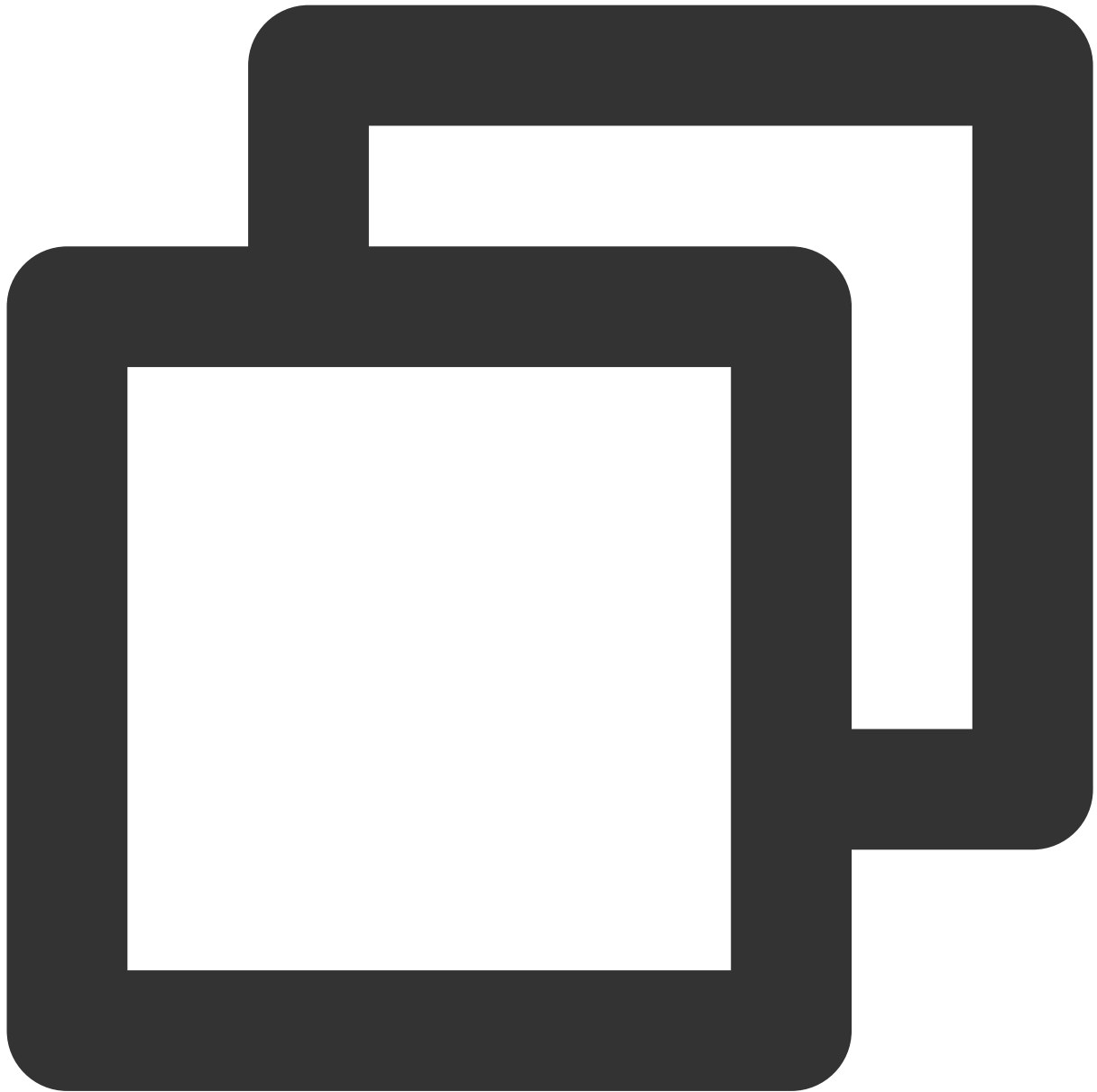
Parameter:

Parameter	Type	Description	Default Value	Meaning
streamType	<a href="#">TUIVideoStreamType</a>	Required	-	Local stream type
view	string	Required	-	The id of the div element corresponding to the streamType

Returns : *Promise<void>*

## openLocalCamera

Open the local camera and start capturing video streams

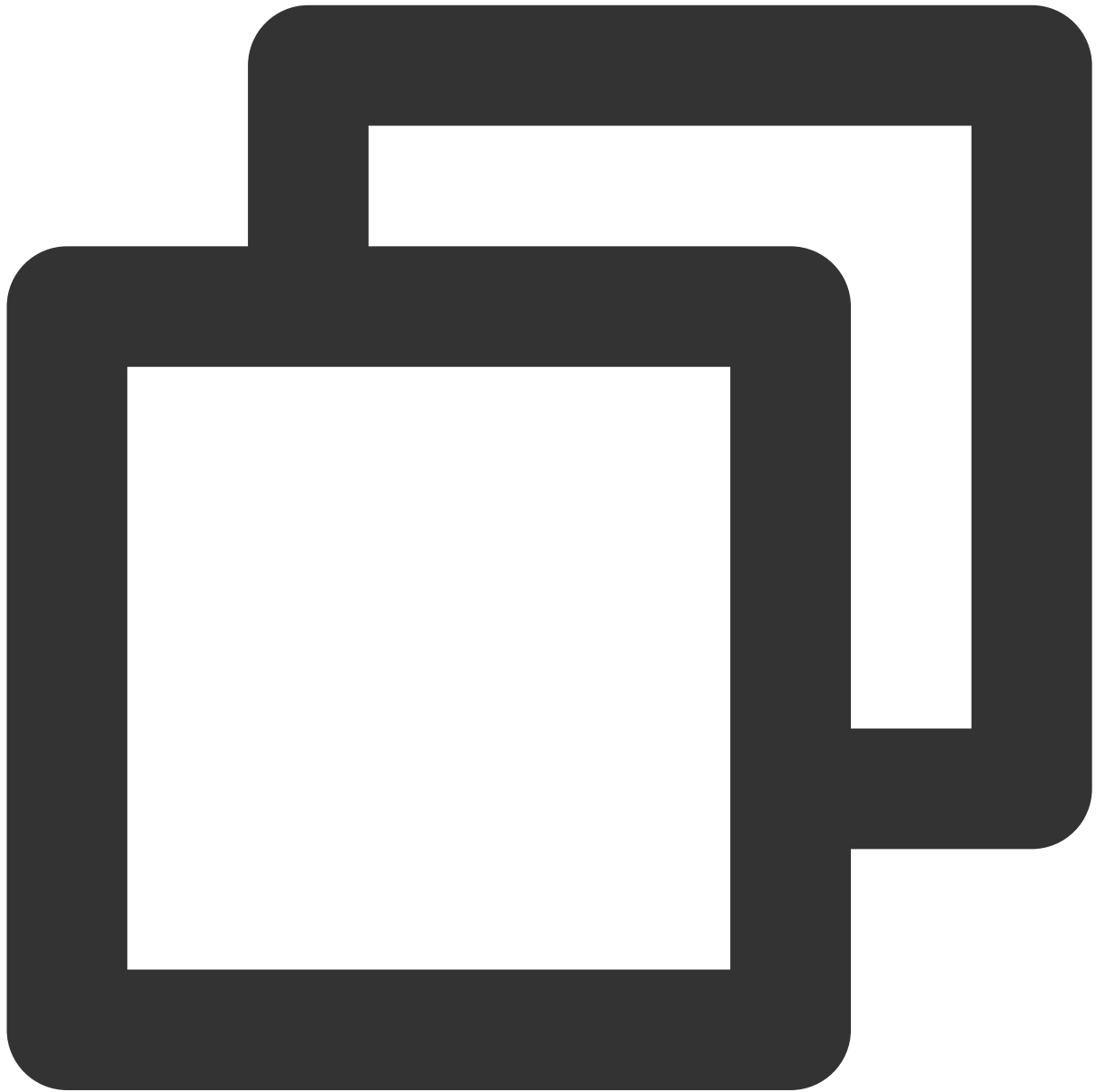


```
const roomEngine = new TUIRoomEngine();
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'preview-camera',
});
await roomEngine.openLocalCamera();
```

Returns : *Promise<void>*

## closeLocalCamera

Close the local camera



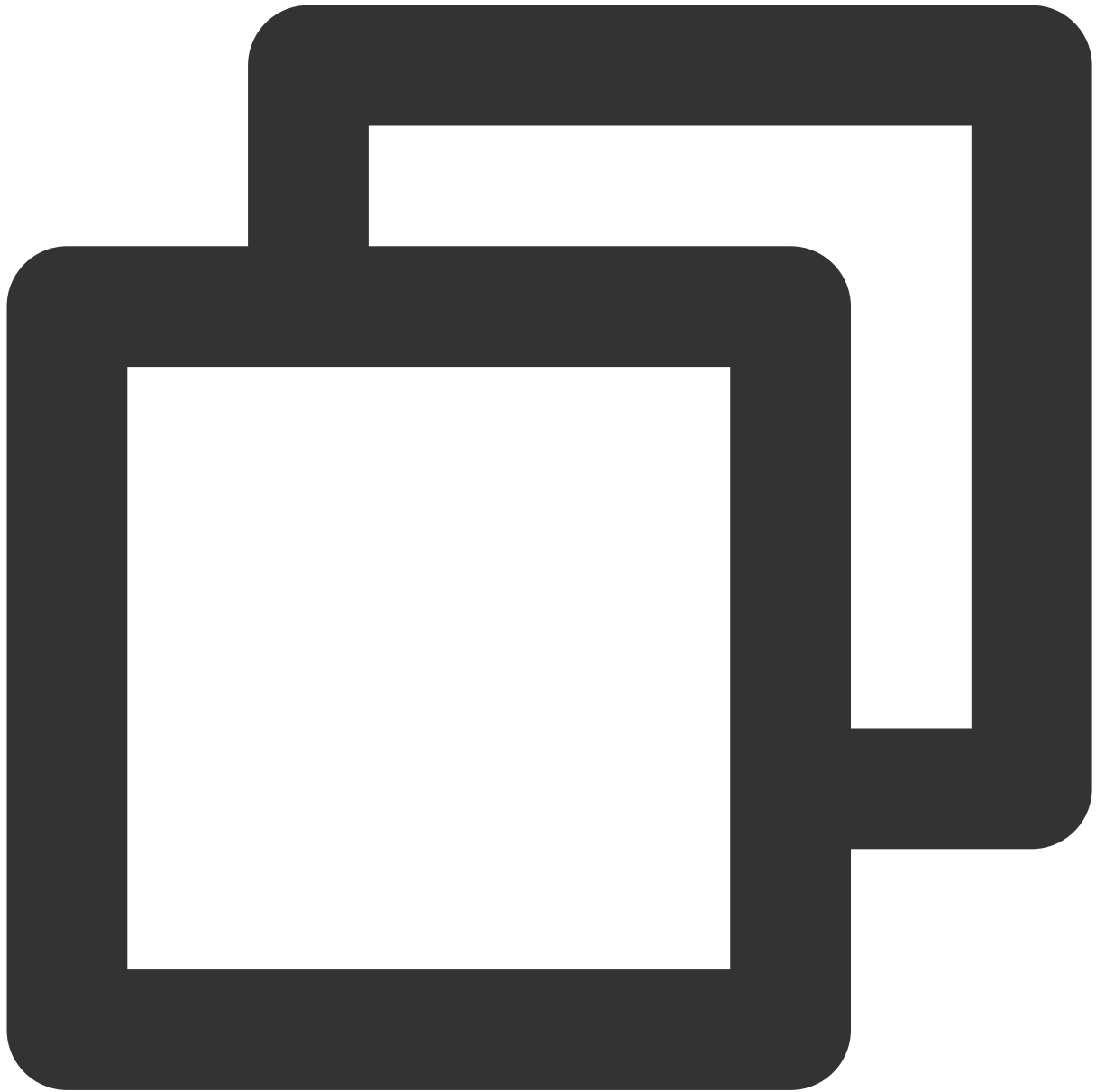
```
const roomEngine = new TUIRoomEngine();  
await roomEngine.closeLocalCamera();
```

Returns : *Promise<void>*



## openLocalMicrophone

Open the local mic and start capturing audio streams

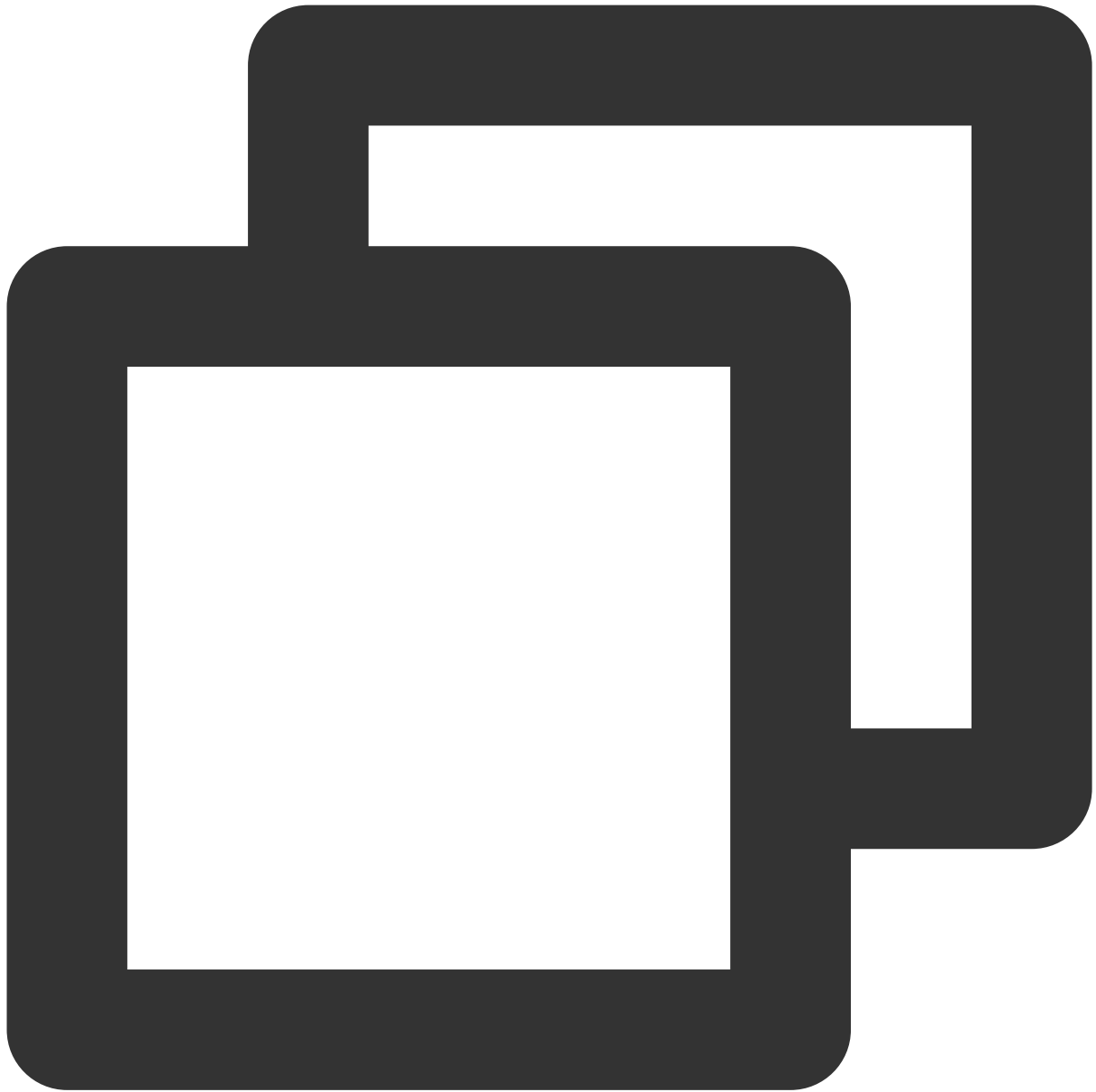


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.openLocalMicrophone();
```

Returns : *Promise<void>*

## closeLocalMicrophone

Close the local mic

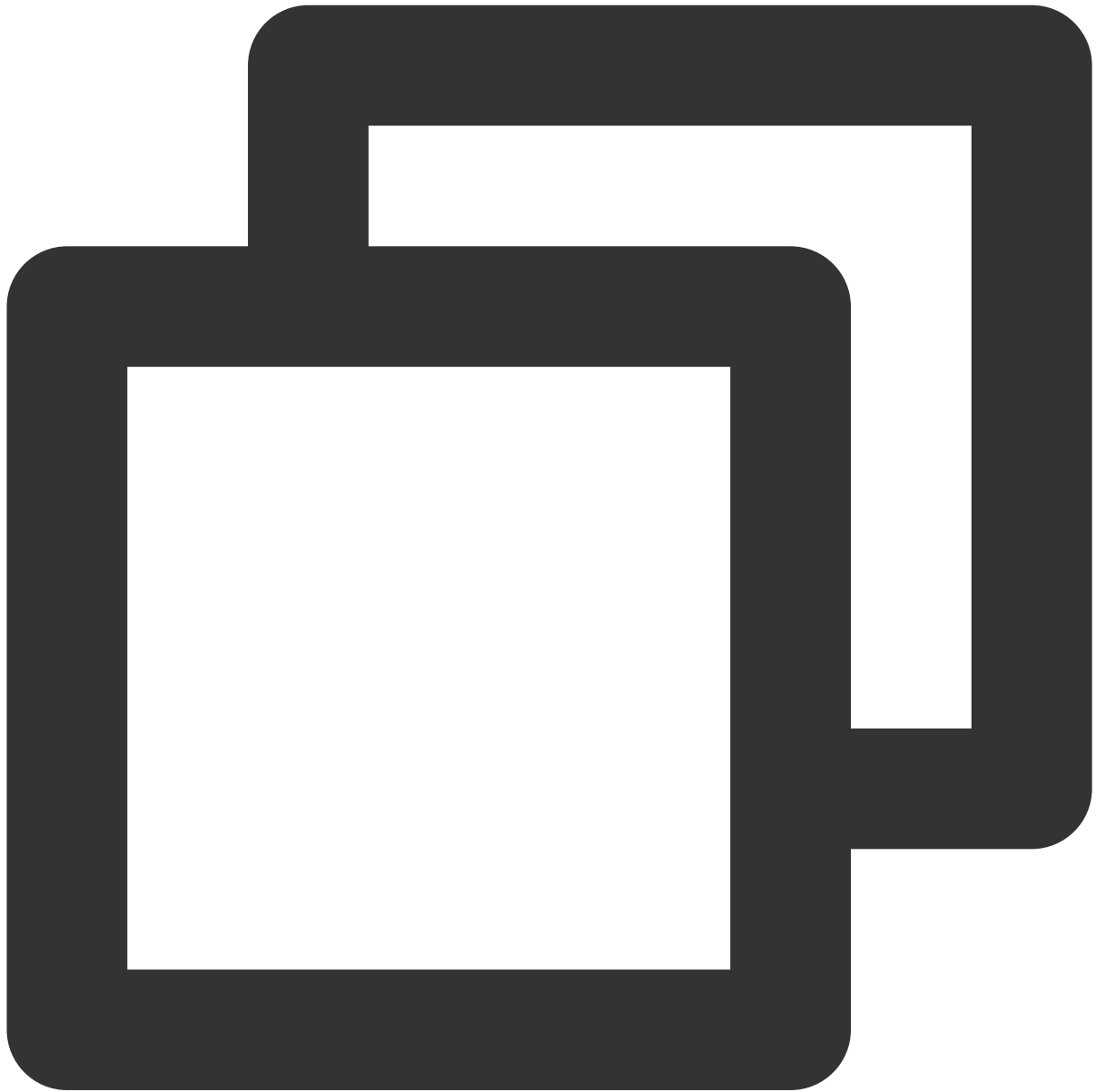


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.closeLocalMicrophone();
```

Returns : *Promise<void>*

## updateVideoQuality

Set the codec parameters of the local video stream, default is `TUIVideoProfile.kVideoQuality_720P`



```
const roomEngine = new TUIRoomEngine();
await roomEngine.updateVideoQuality({
  quality: TUIVideoQuality.kVideoQuality_540p,
});
```

Parameter:

--	--	--	--	--

Parameter	Type	Description	Default Value	Meaning
quality	<a href="#">TUIVideoQuality</a>	Required	-	Clear TUIVideoProfile.kVideoQuality_360P SD TUIVideoProfile.kVideoQuality_540P HD TUIVideoProfile.kVideoQuality_720P Full HD TUIVideoProfile.kVideoQuality_1080P

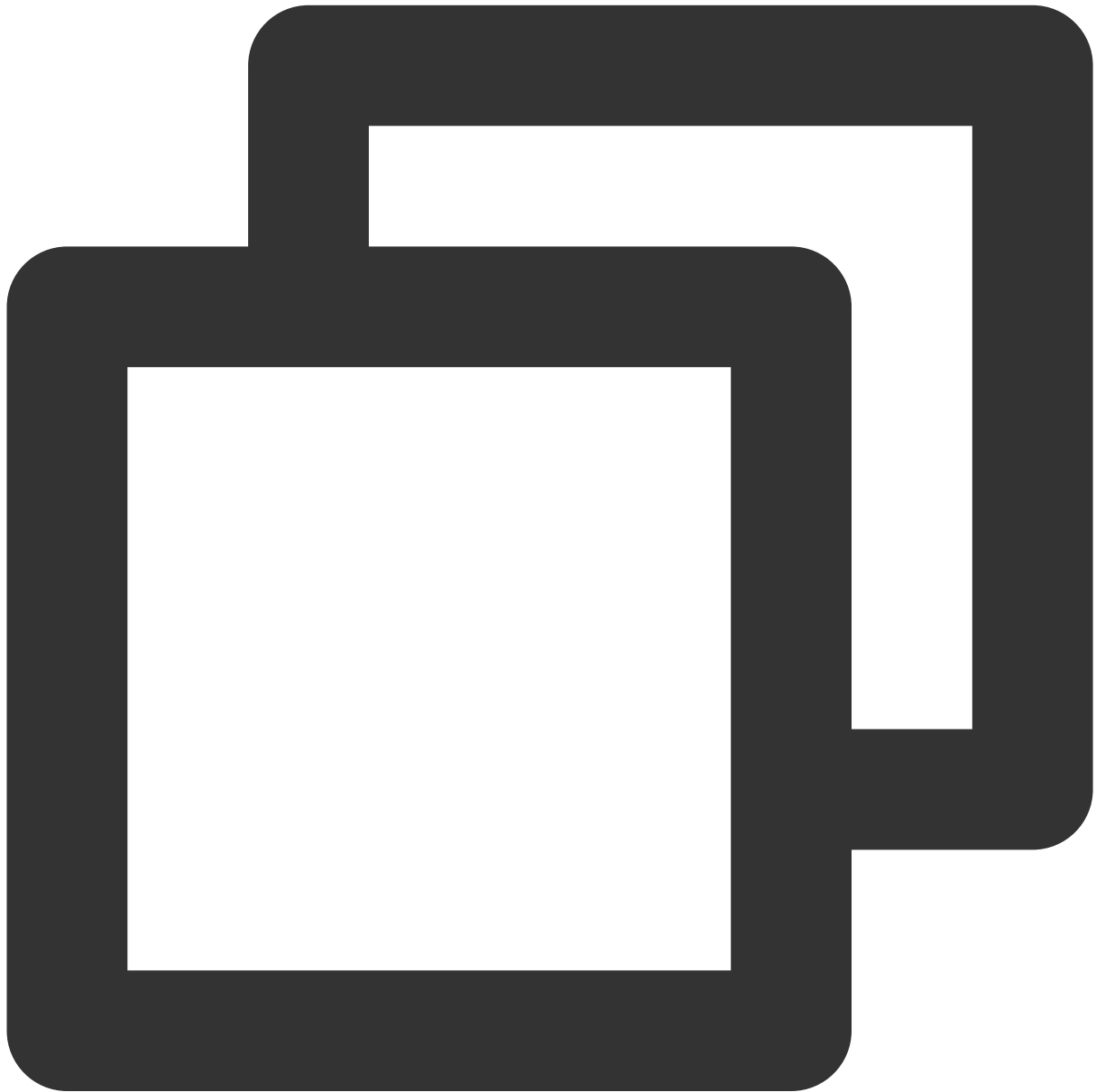
Returns : *Promise<void>*

## updateAudioQuality

Set Local Audio Parameters

### Note:

This method needs to be set before openLocalMicrophone, otherwise it will not take effect.



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setLocalAudioProfile({
  audioProfile: TUIAudioProfile.kAudioProfileSpeech,
});
```

Parameter:

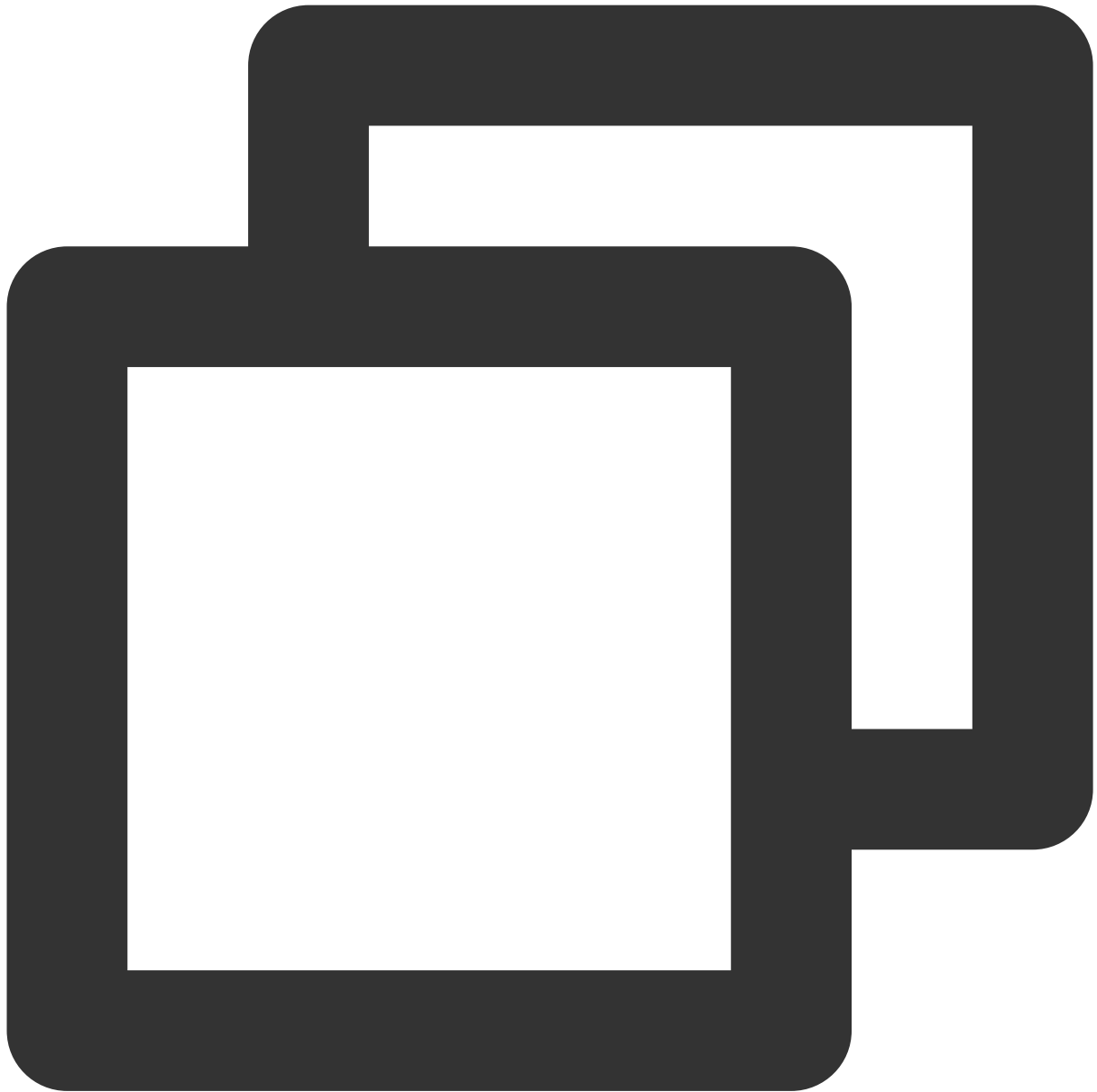
Parameter	Type	Description	Default Value	Meaning
audioProfile	<a href="#">TUIAudioProfile</a>	Required	-	TUIAudioProfile.kAudioProfileSpeech: Speech

				Mode; Sample rate: 16k TUIAudioProfile.kAudioProfileDefault: Standard Mode (or Default Mode); Sample rate: 48k TUIAudioProfile.kAudioProfileMusic: Music Mode; Sample rate: 48k
--	--	--	--	--

Returns : *Promise*<void>

## startScreenSharing

Start Screen Sharing



```
const roomEngine = new TUIRoomEngine();
// Set the playback area of the local screen sharing stream to the div element with the id 'preview-screen'
await roomEngine.setLocalRenderView({
  streamType: TUIVideoStreamType.kScreenStream,
  view: 'preview-screen',
});
// example 1: Start Screen Sharing
await roomEngine.startScreenSharing();
// example 2: Start Screen Sharing(Capturing System Audio)
await roomEngine.startScreenSharing({ screenAudio: true });
```

Parameter:

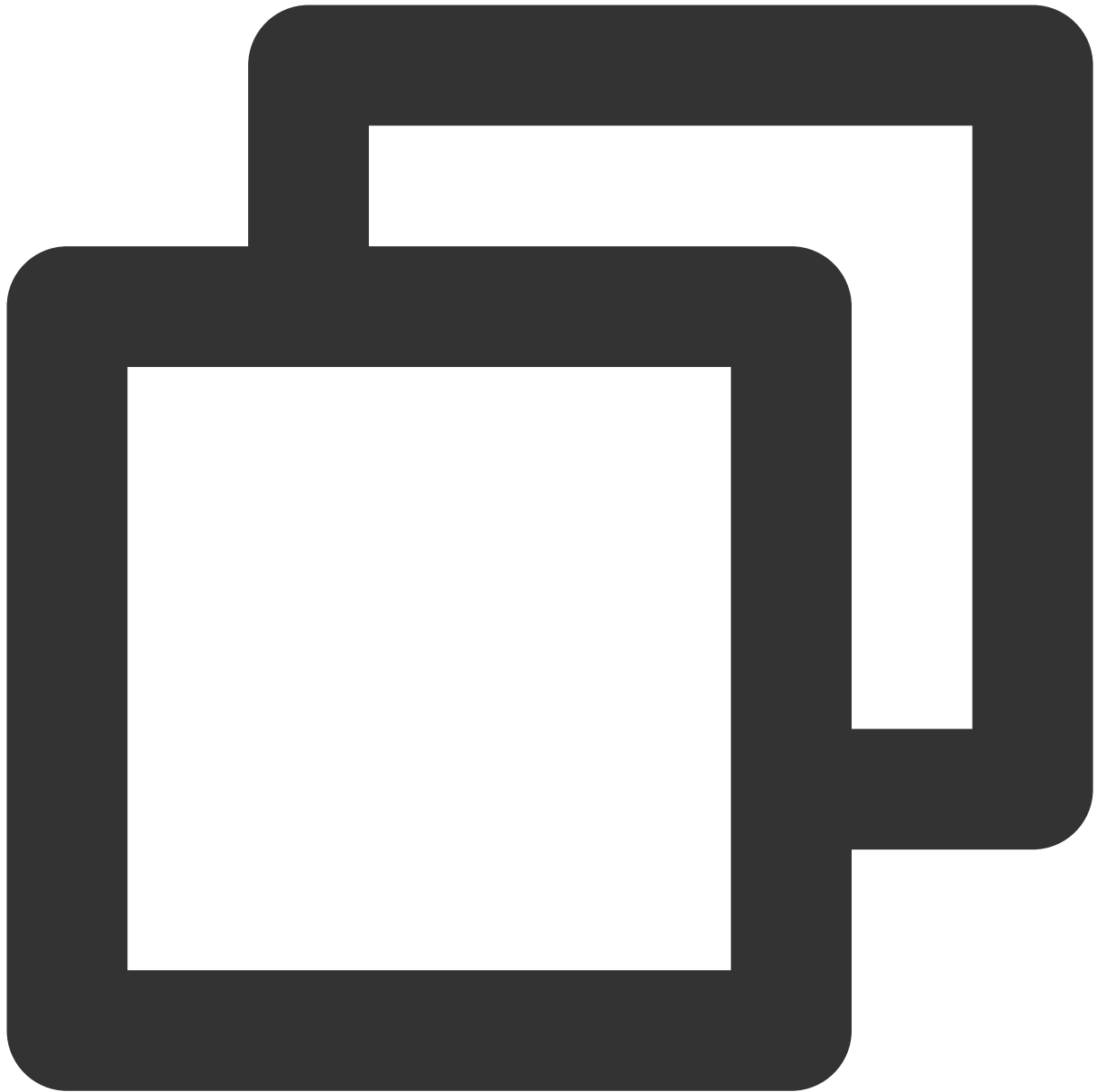
Parameter	Type	Description	Default Value	Meaning
screenAudio	boolean	Optional	false	Whether web screen sharing can optionally share system sound, screenAudio default value is false

Returns : *Promise<void>*

## stopScreenSharing

Stop Screen Sharing



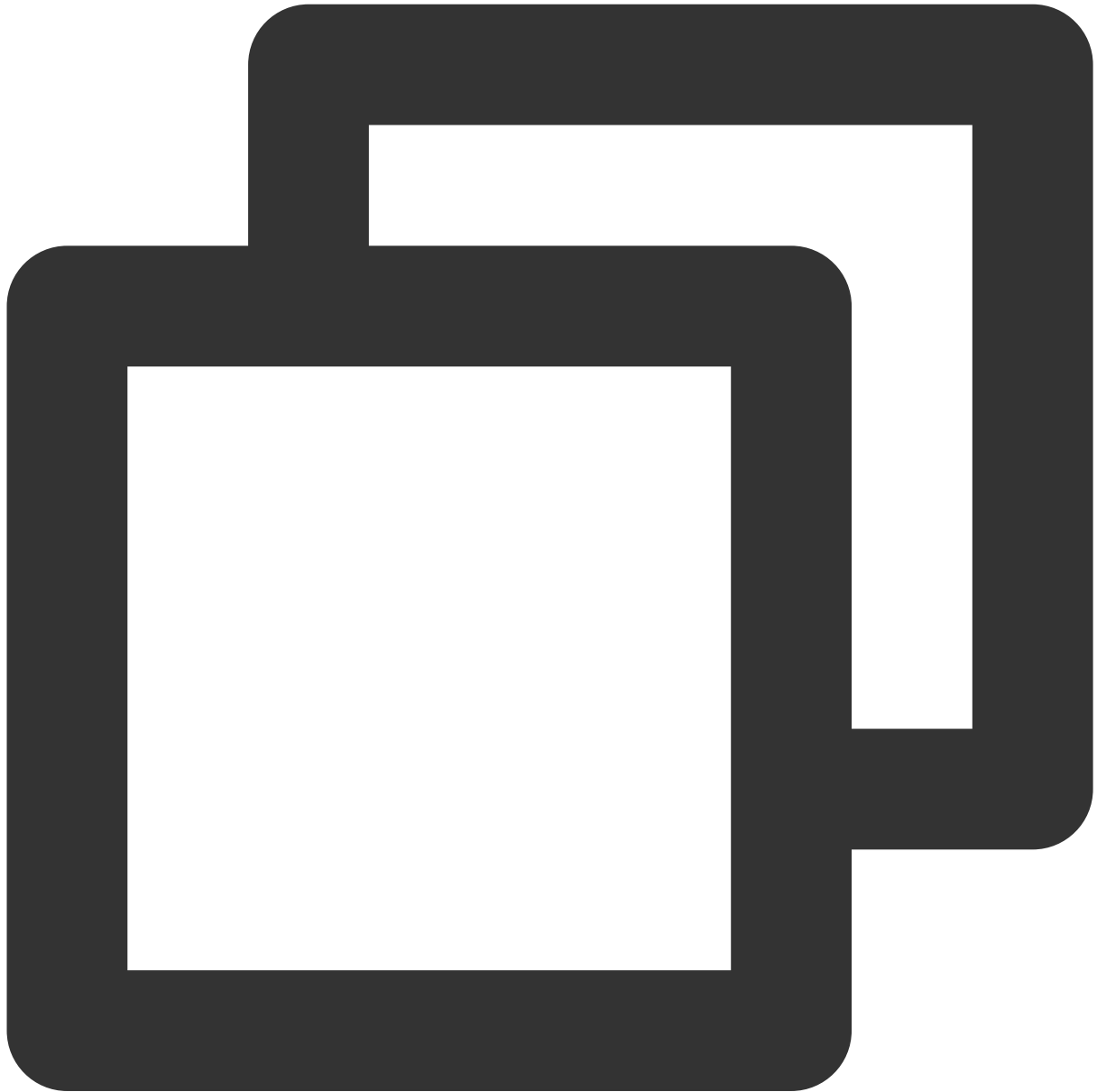


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.stopScreenSharing();
```

Returns : *Promise<void>*

## startPushLocalVideo

After entering the room, the local video stream will be pushed to the remote by default. This interface is used to re-push the local video stream to the remote after stopping the push.

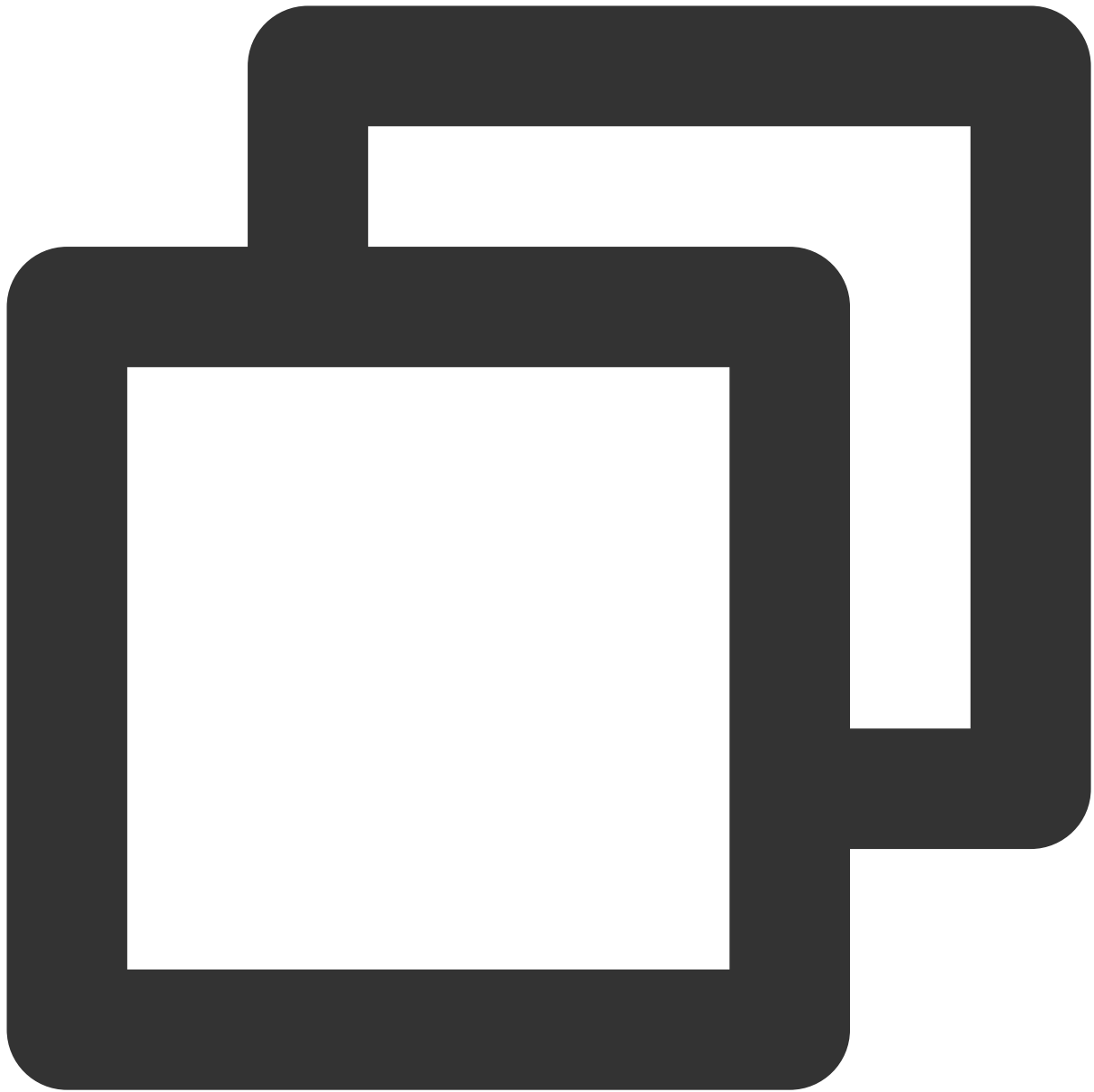


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.startPushLocalVideo();
```

Returns : *Promise<void>*

## stopPushLocalVideo

## Stop Pushing Local Video Stream to Remote

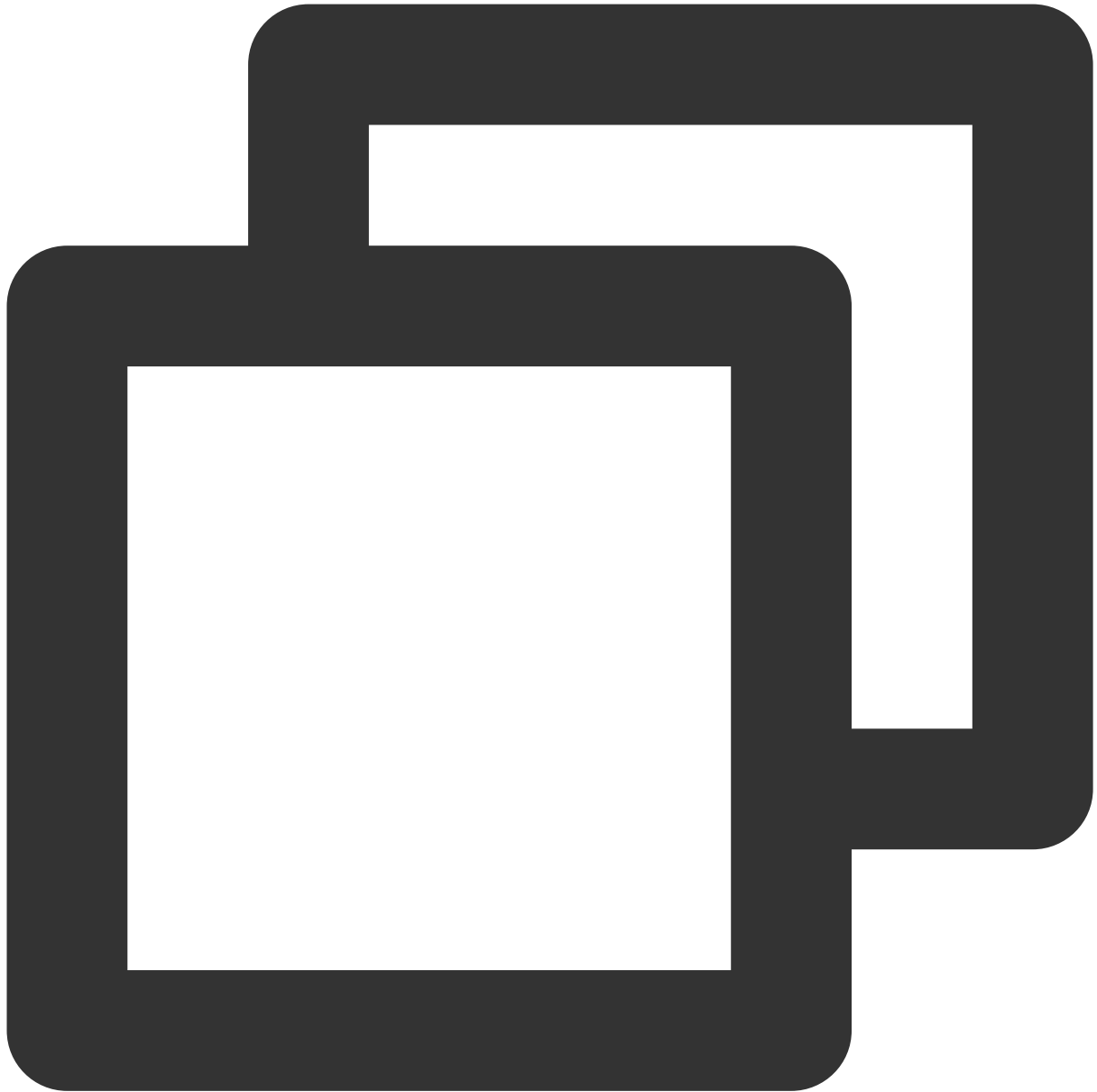


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.stopPushLocalVideo();
```

Returns : *Promise<void>*

## startPushLocalAudio

After entering the room, the local audio stream will be pushed to the remote by default. This interface is used to re-push the local audio stream to the remote after stopping the push.

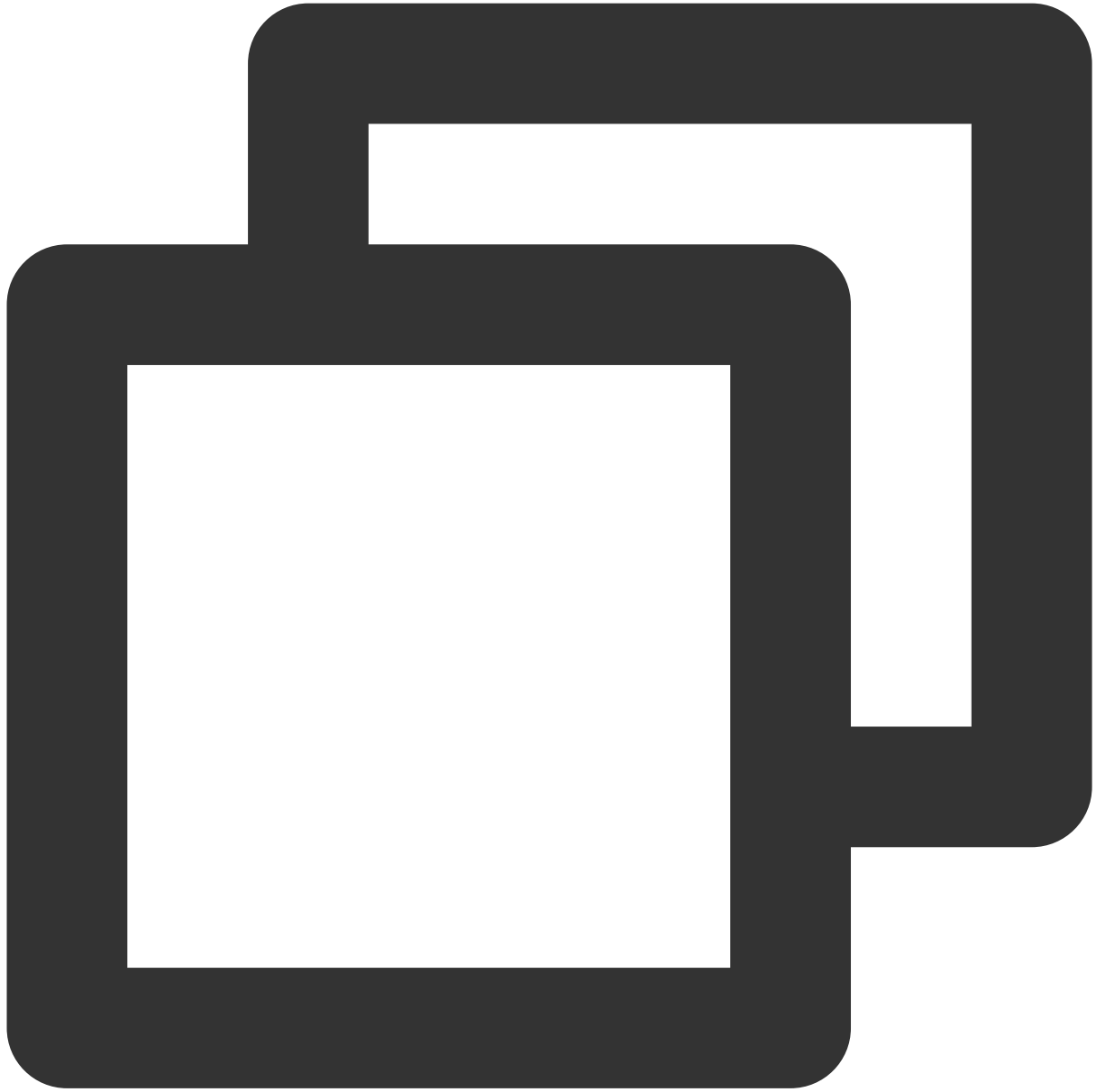


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.startPushLocalAudio();
```

Returns : *Promise<void>*

## stopPushLocalAudio

Stop Pushing Local Audio Stream to Remote

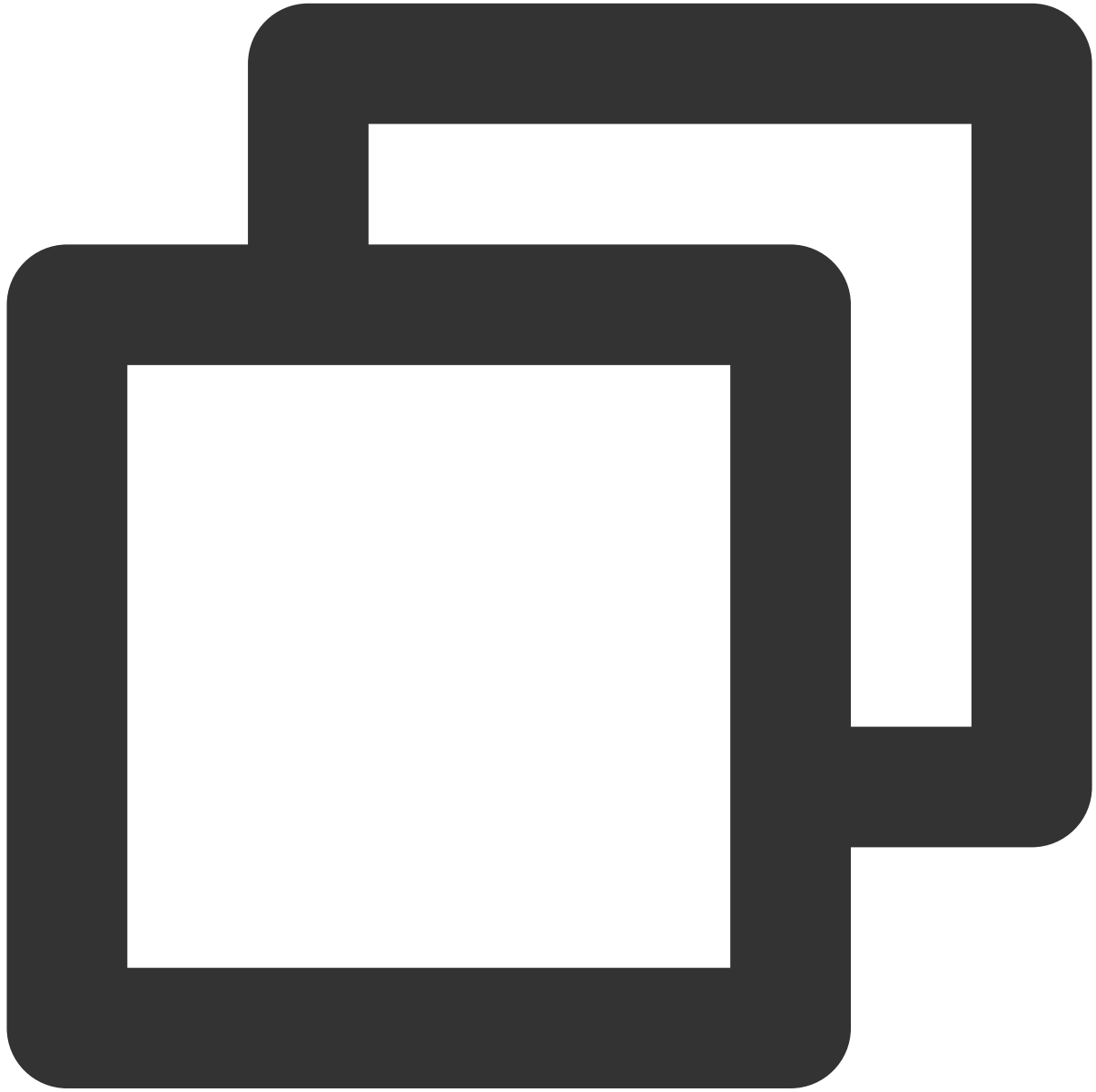


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.stopPushLocalAudio();
```

Returns : *Promise<void>*

## setRemoteVideoView

## Set Remote Stream Rendering Area



```
const roomEngine = new TUIRoomEngine();

// Set the remote user's video stream to play in the area with id 'remote_preview_c
await roomEngine.setRemoteVideoView({
  userId: 'user_1234',
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'remote_preview_camera',
});
// Set the remote user's screen sharing stream to play in the area with id 'remote_
```

```
await roomEngine.setRemoteVideoView({
  userId: 'user_1234',
  streamType: TUIVideoStreamType.kScreenStream,
  view: 'remote_preview_screen',
});
```

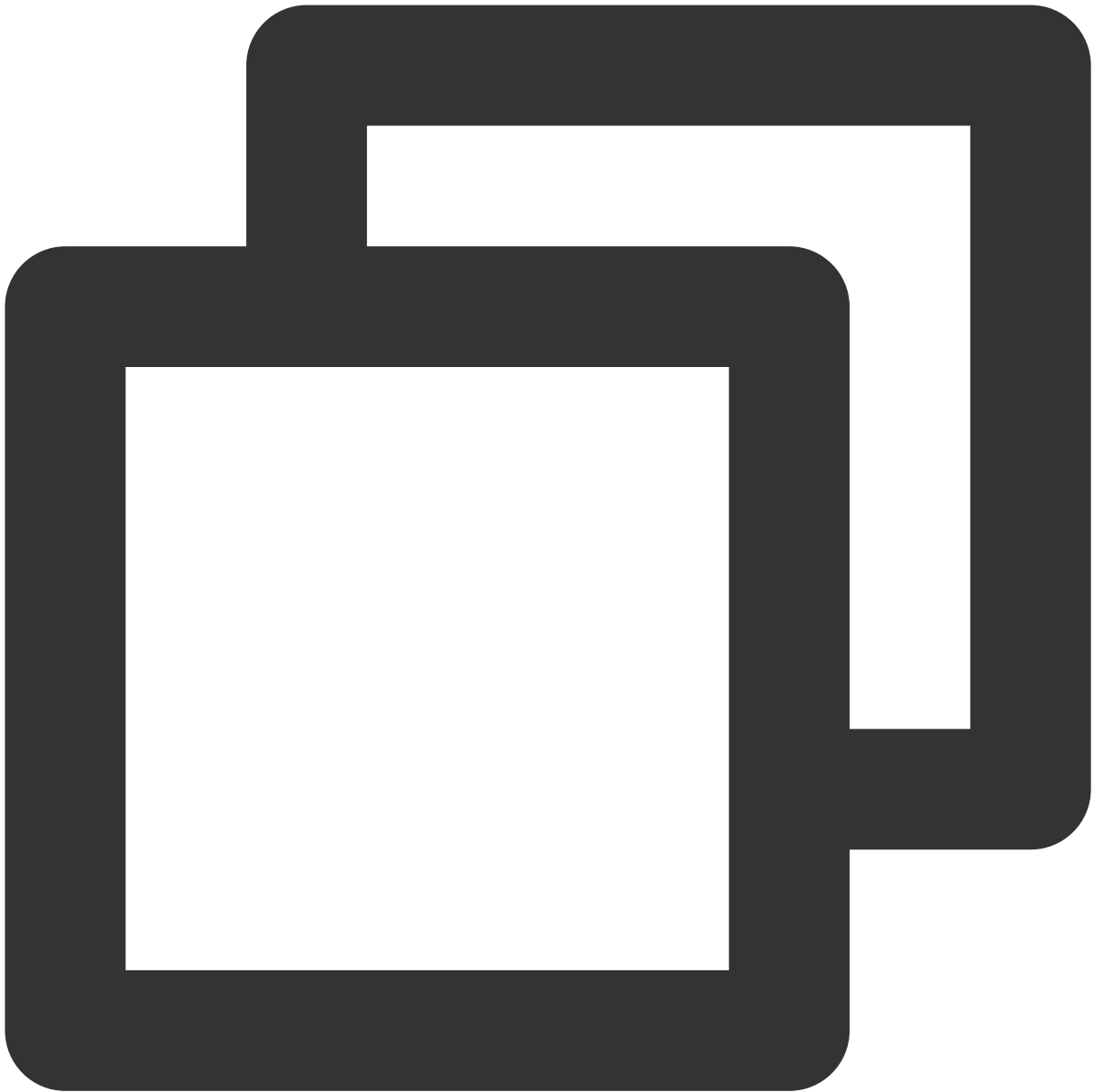
Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID
streamType	<a href="#">TUIVideoStreamType</a>	Required	-	User Stream Type
view	string	Required	-	The id of the div element playing the remote user's stream

Returns : *Promise<void>*

## startPlayRemoteVideo

Start Playback of Remote User Video Stream



```
const roomEngine = new TUIRoomEngine();
await roomEngine.startPlayRemoteVideo({
  userId: 'user_1234',
  streamType: TUIVideoStreamType.kCameraStream,
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning

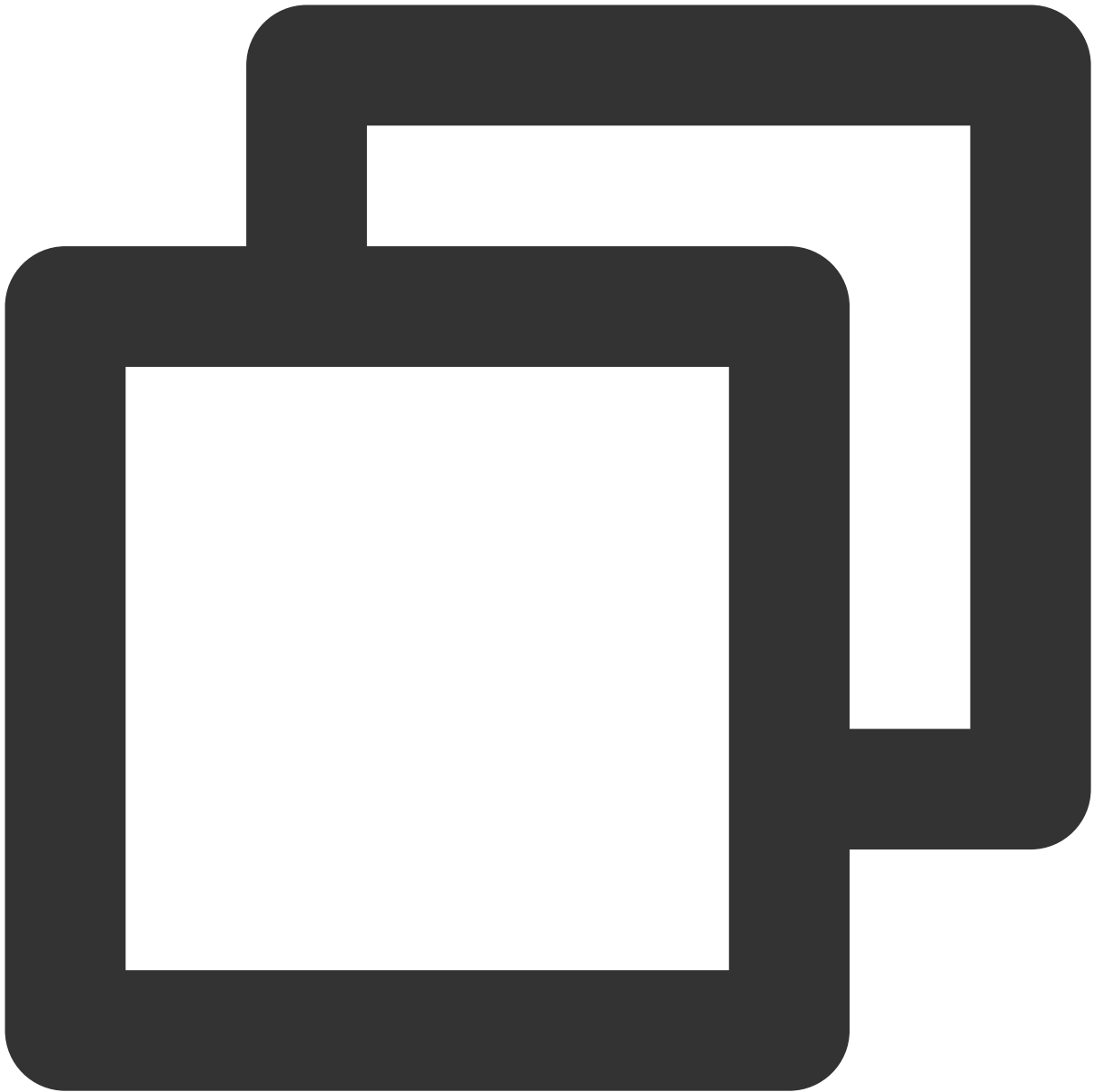


userId	string	Required	-	User ID
streamType	<a href="#">TUIVideoStreamType</a>	Required	-	User Stream Type TUIVideoStreamType.kCameraStream Video Stream TUIVideoStreamType.kScreenStream Screen Sharing Stream TUIVideoStreamType.kCameraStreamLow Low Definition Video Stream

Returns : *Promise<void>*

## stopPlayRemoteVideo

Stop Playback of Remote User Video Stream



```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopPlayRemoteVideo({
  userId: 'user_1234',
  streamType: TUIVideoStreamType.kCameraStream,
});
```

Parameter:

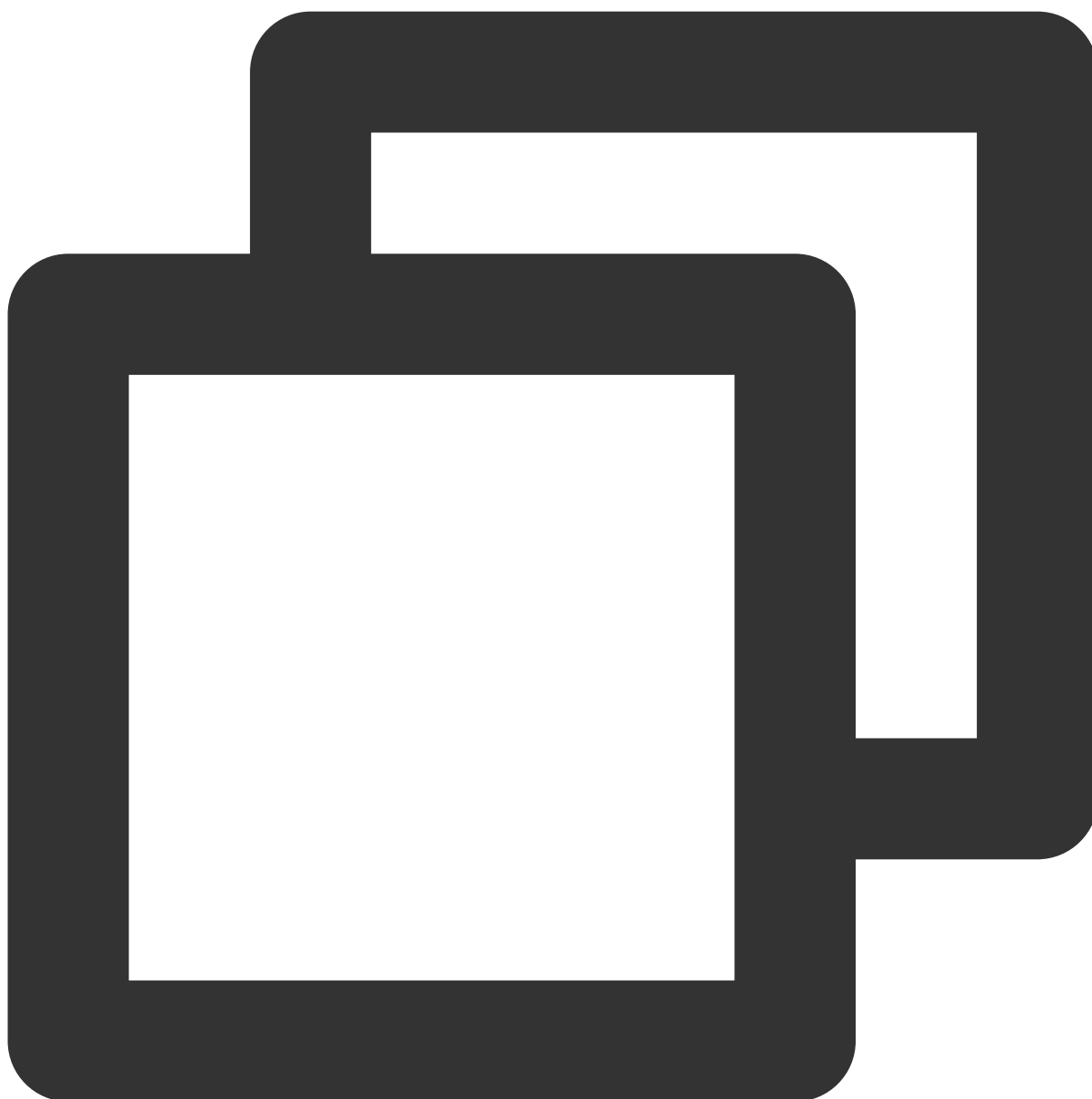
Parameter	Type	Description	Default Value	Meaning

userId	string	Required	-	User ID
streamType	<a href="#">TUIVideoStreamType</a>	Required	-	User Stream Type TUIVideoStreamType.kCameraStream Video Stream TUIVideoStreamType.kScreenStream Screen Sharing Stream TUIVideoStreamType.kCameraStreamLow Low Definition Video Stream

Returns : *Promise<void>*

## muteRemoteAudioStream

Stop Remote User's Audio Stream



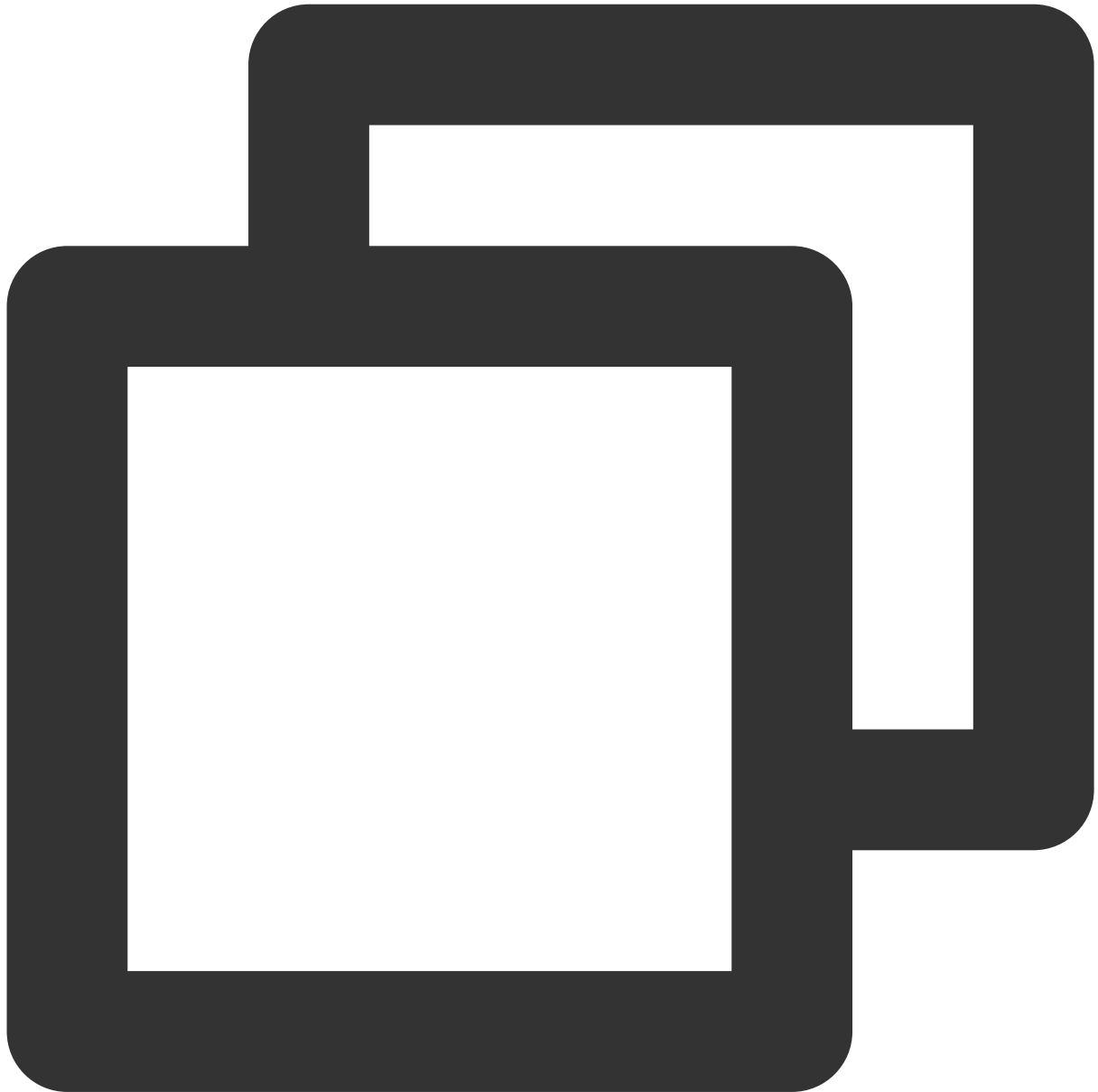
```
const roomEngine = new TUIRoomEngine();
await roomEngine.muteRemoteAudioStream({
  userId: 'user_1234',
  isMute: true,
});
```

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID

isMute	boolean	Required	-	Whether to Stop Remote User's Audio
--------	---------	----------	---	-------------------------------------

## openRemoteDeviceByAdmin

Request Remote User to Open Media Device



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.openRemoteDeviceByAdmin({
  userId: 'user_1234',
```

```

device: TUIMediaDevice.kMicrophone //The requested device is a mic
timeout: 0,
requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
  switch (requestCallbackType) {
    case TUIRequestCallbackType.kRequestAccepted:
      // Request Accepted
      break;
    case TUIRequestCallbackType.kRequestRejected:
      // Request Rejected
      break;
    case TUIRequestCallbackType.kRequestCancelled:
      // Request Canceled
      break;
    case TUIRequestCallbackType.kRequestTimeout:
      // Request Timeout
      break;
    case TUIRequestCallbackType.kRequestError:
      // Request Error
      break;
    default:
      break;
  }
},
});

```

**Parameter:**

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID
device	TUIMediaDevice	Required	-	Media Device Type (Camera/Mic/Screen Sharing)
timeout	number	Required	-	Timeout Time. If timeout is set to 0, there is no timeout time
requestCallback	Function	Required	Empty Function	Request Callback, used to notify the initiator of the request being accepted/rejected/canceled/timeout/error

Returns : *Promise<string> requestId*

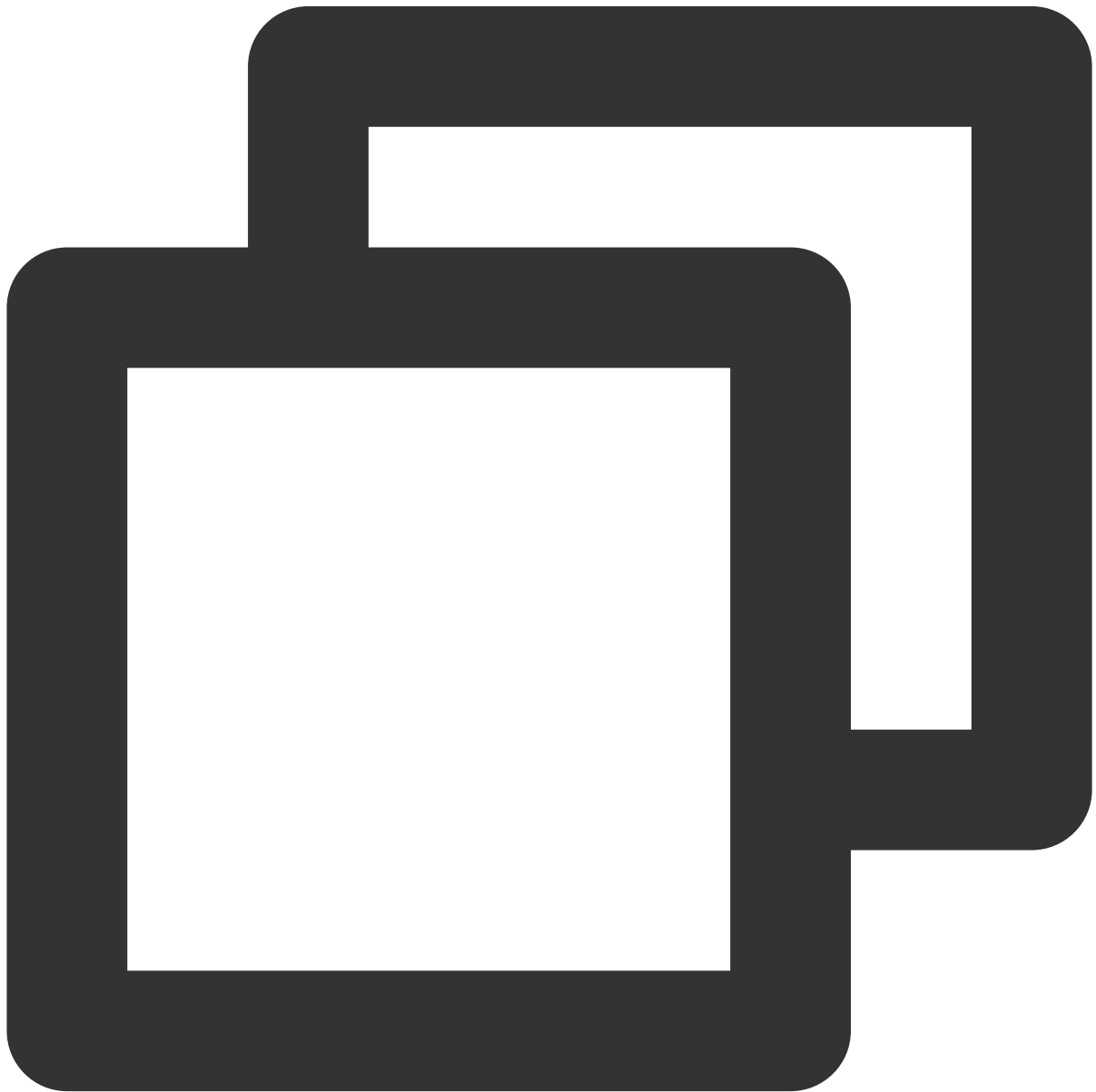
This interface returns requestId, users can use this requestId to call cancelRequest interface to cancel the request.

**Description:**

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number;

## applyToAdminToOpenLocalDevice

Participant applies to the host to open the device



```
const roomEngine = new TUIRoomEngine();  
const requestId = roomEngine.applyToAdminToOpenLocalDevice({
```

```
device: TUIMediaDevice.kMicrophone //The requested device is a mic
timeout: 0,
requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
  switch (requestCallbackType) {
    case TUIRequestCallbackType.kRequestAccepted:
      // Request Accepted
      break;
    case TUIRequestCallbackType.kRequestRejected:
      // Request Rejected
      break;
    case TUIRequestCallbackType.kRequestCancelled:
      // Request Canceled
      break;
    case TUIRequestCallbackType.kRequestTimeout:
      // Request Timeout
      break;
    case TUIRequestCallbackType.kRequestError:
      // Request Error
      break;
    default:
      break;
  }
},
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
device	TUIMediaDevice	Required	-	Media Device Type (Camera/Mic/Screen Sharing)
timeout	number	Required	-	Timeout Time. If timeout is set to 0, there is no timeout time
requestCallback	Function	Optional	Empty Function	Request Callback, used to notify the initiator of the request being accepted/rejected/canceled/timeout/error

Returns : *Promise<string> requestId*

This interface returns requestId, users can use this requestId to call cancelRequest interface to cancel the request

## closeRemoteDeviceByAdmin



## Close Remote User Media Device



```
const roomEngine = new TUIRoomEngine();
await roomEngine.closeRemoteDeviceByAdmin({
  userId: 'user_1234',
  device: TUIMediaDevice.kMicrophone, //Close mic
});
```

## Parameter:

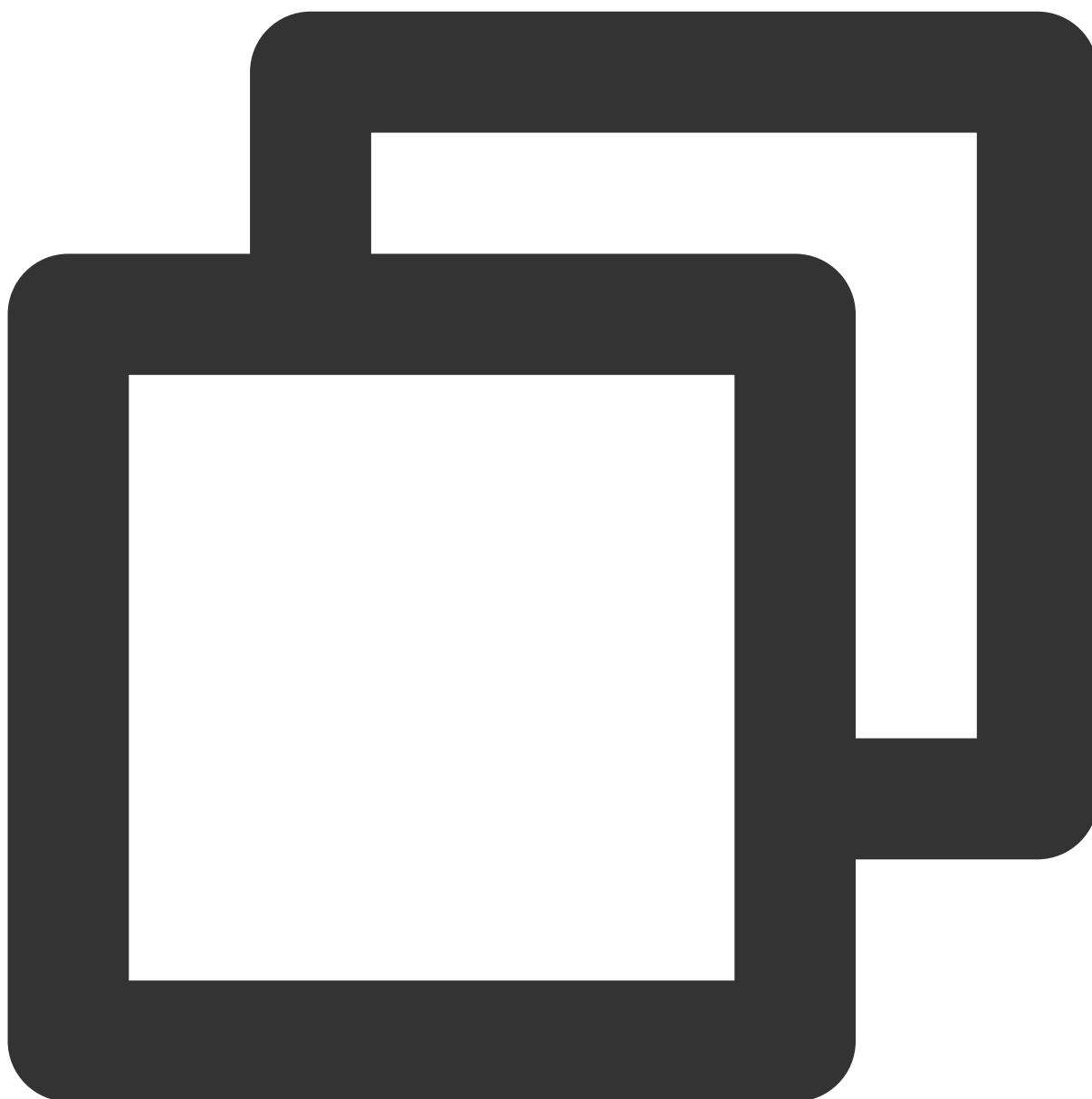
Parameter	Type	Description	Default Value	Meaning
-----------	------	-------------	---------------	---------

userId	string	Required	-	User ID
device	TUIMediaDevice	Required	-	Media Device Type (Camera/Mic/Screen Sharing)

Returns : *Promise<void>*

## cancelRequest

Cancel Already Sent Request



```
const roomEngine = new TUIRoomEngine();
await roomEngine.cancelRequest({
  requestId: '',    // Please use Actual requestId
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
requestId	string	Required	-	Request ID

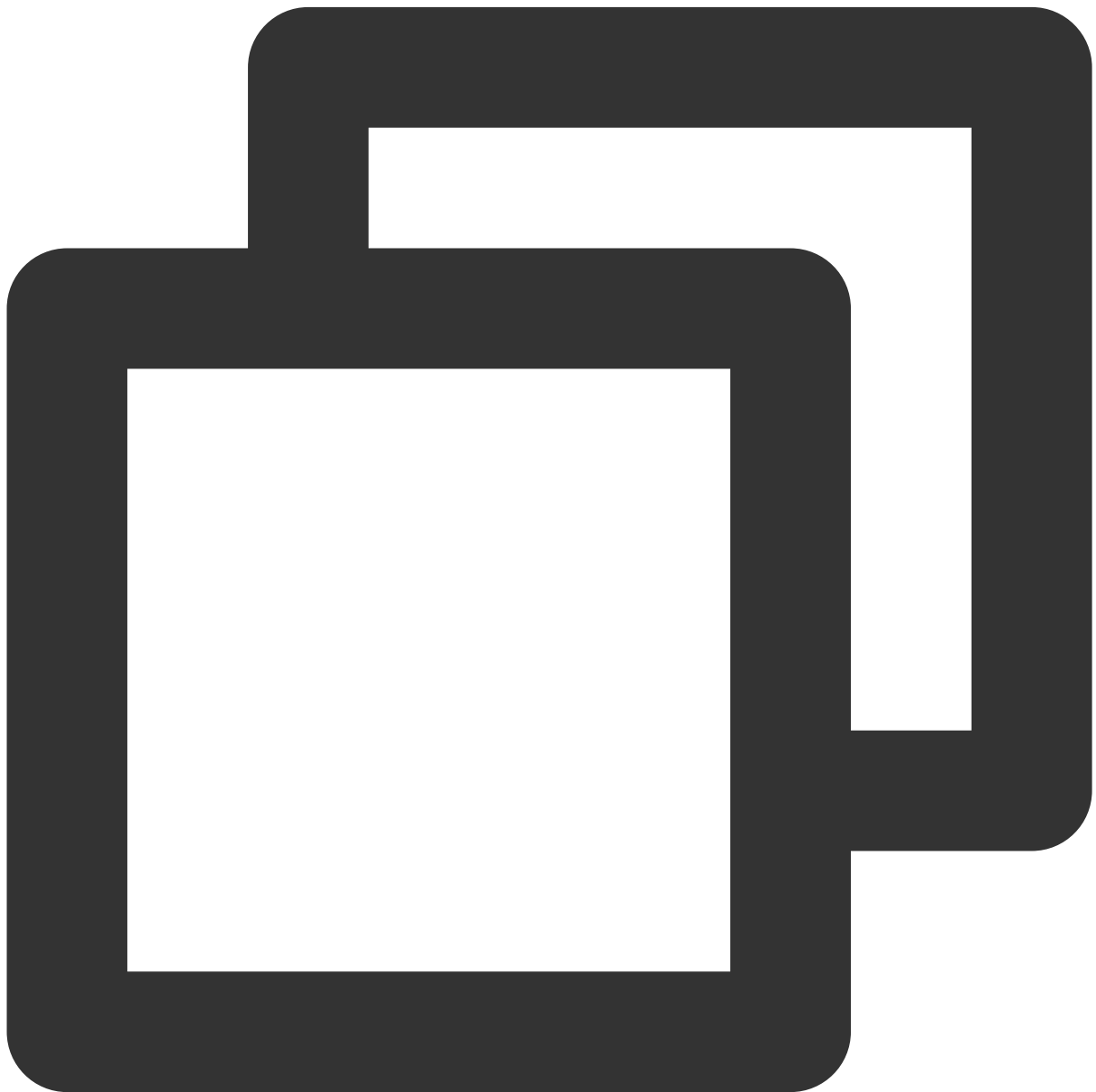
Returns : *Promise<void>*

**Note:**

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number;

## responseRemoteRequest

Reply to Remote User's Request



```
const roomEngine = new TUIRoomEngine();
// Agree to Remote User's Request
await roomEngine.responseRemoteRequest({
  requestId: '',    // Please use Actual requestId
  agree: true,
});
// Reject Remote User's Request
await roomEngine.responseRemoteRequest({
  requestId: '',    // Please use Actual requestId
  agree: false,
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
requestId	string	Required	-	Request ID
agree	boolean	Required	-	Whether to Agree

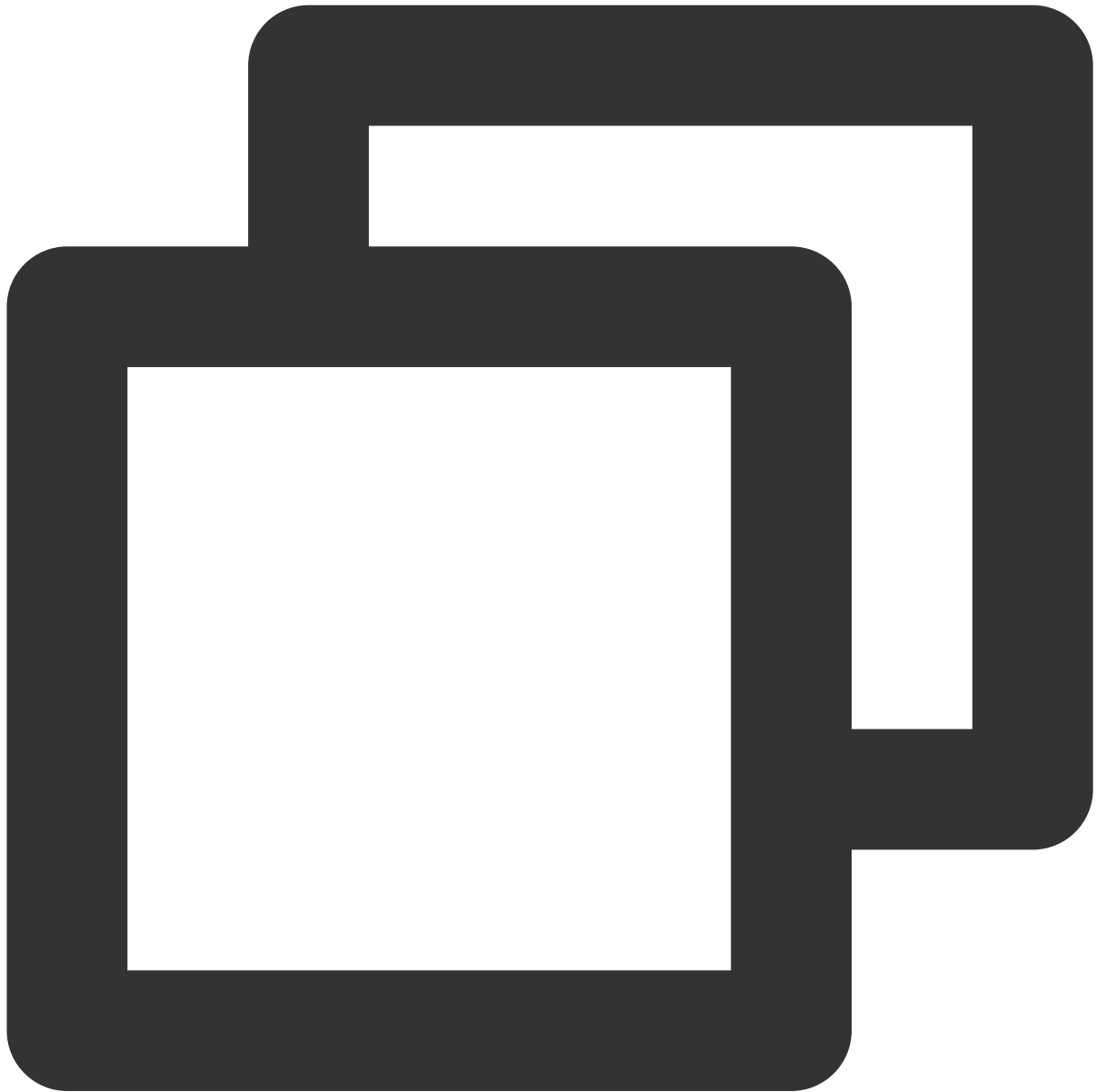
Returns : *Promise<void>*

**Note:**

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number;

## disableDeviceForAllUserByAdmin

Prohibit/Allow All Users to Open Media Device (This Interface is invalid for Room Owner and Administrator)



```
// Example 1: Prohibit All Users to Open Mic
await roomEngine.disableDeviceForAllUserByAdmin({
  device: TUIMediaDevice.kMicrophone,
  isDisable: true,
})
// Example 2: Allow All Users to Open Mic
await roomEngine.disableDeviceForAllUserByAdmin({
  device: TUIMediaDevice.kMicrophone,
  isDisable: false,
})
```

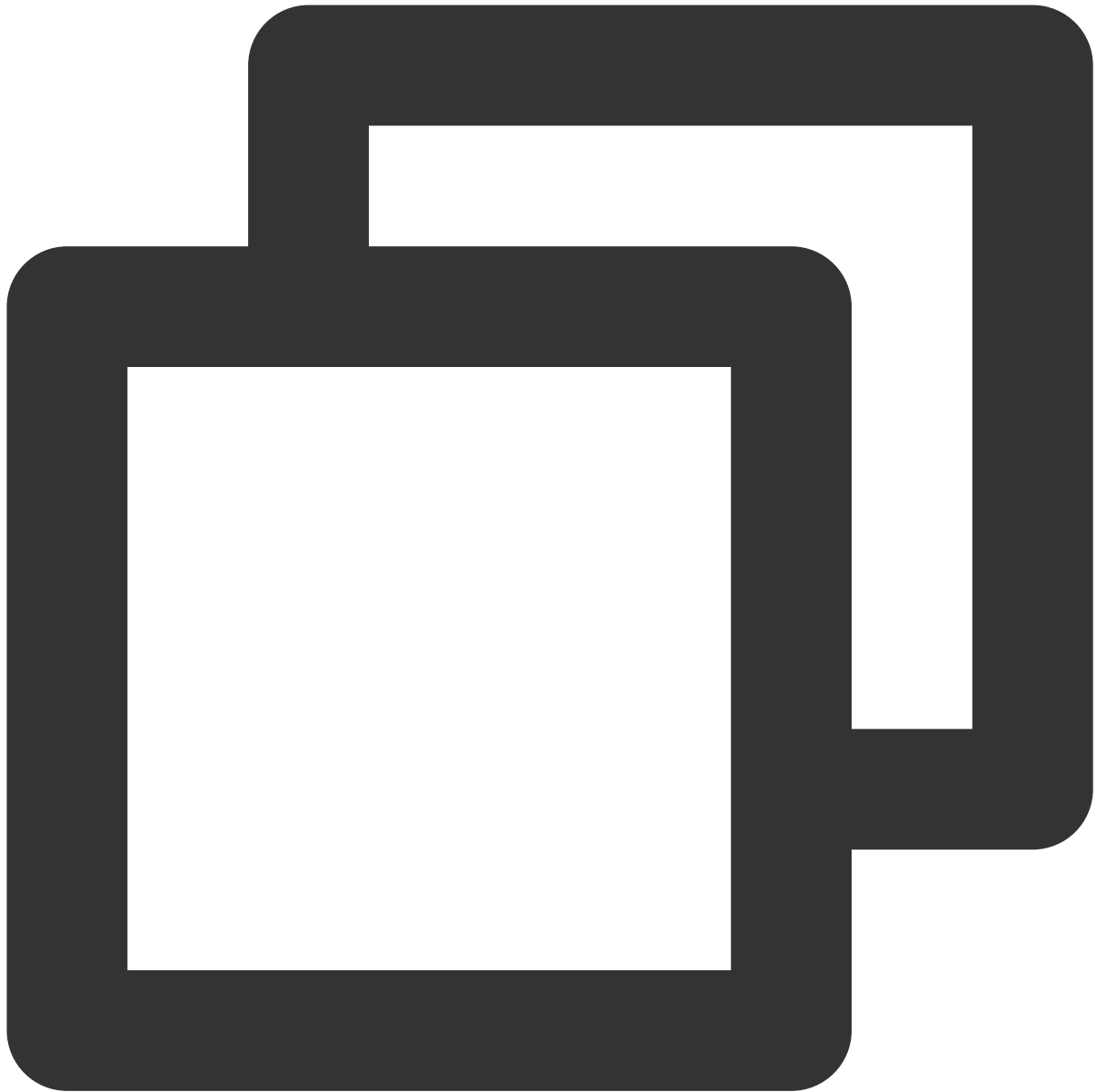
Parameter:

Parameter	Type	Description	Default Value	Meaning
device	TUIMediaDevice	Required	-	Disabled Media Device Type (Camera/Mic/Screen Sharing)
isDisable	boolean	Required	-	Whether it is Prohibited

Returns : *Promise<void>*

## disableSendingMessageForAllUser

Whether All Users are Allowed to Send Messages (This Interface is invalid for Room Owner and Administrator)



```
await roomEngine.disableSendingMessageForAllUser({  
  isDisable: true,  
});
```

Parameter:

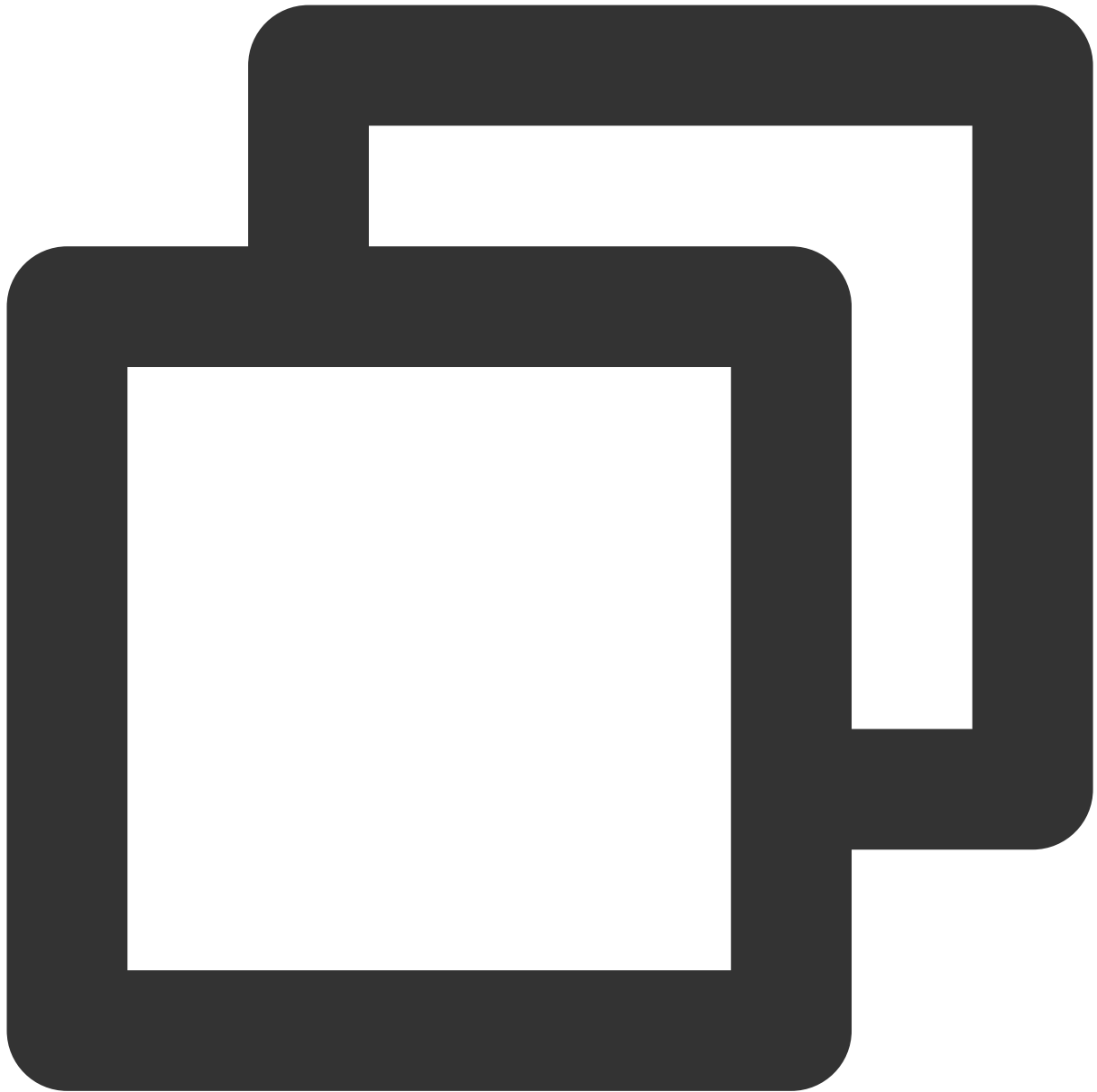
Parameter	Type	Description	Default Value	Meaning
isDisable	boolean	Required	-	Whether it is Disabled



Returns : *Promise<void>*

## disableSendingMessageByAdmin

Whether Specific User is Allowed to Send Messages



```
await roomEngine.disableSendingMessageByAdmin({
  userId: 'user_1234',
  isDisable: true,
```

```
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID
isDisable	boolean	Required	-	Whether it is Disabled

Returns : *Promise<void>*

## changeUserRole

Change User's Role (Only the Host can call this Interface)



```
const roomEngine = new TUIRoomEngine();

// Transfer the Room to User user_1234
await roomEngine.changeUserRole({
  userId: 'user_1234',
  role: TUIRole.kRoomOwner,
});

// Set user_1234 as Room Administrator
await roomEngine.changeUserRole({
  userId: 'user_1234',
```

```
    userRole: TUIRole.kAdministrator,  
  });
```

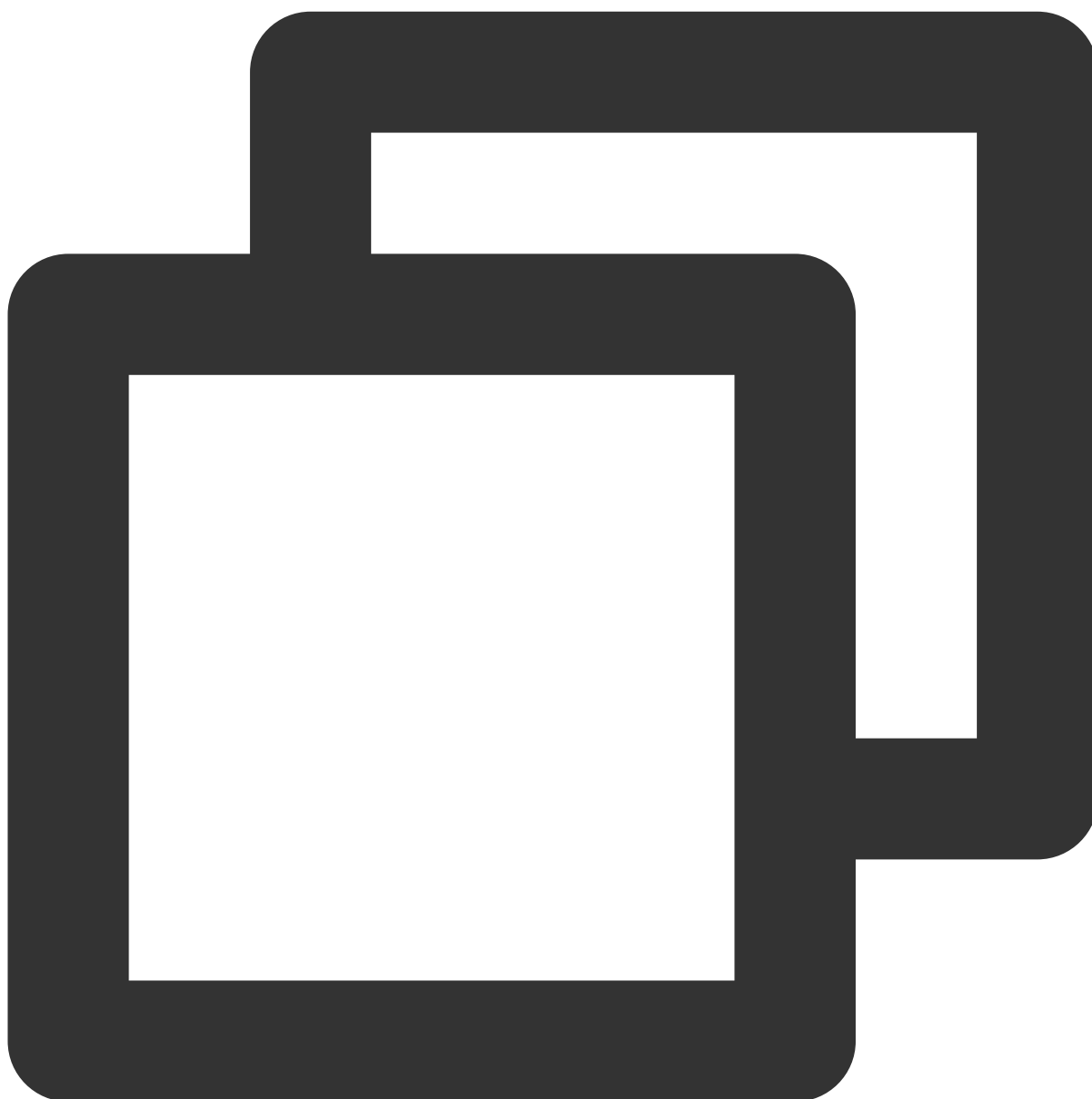
Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID
userRole	<a href="#">TUIRole</a>	Required	-	User Role Host TUIRole.kRoomOwner Administrator TUIRole.kAdministrator General Member TUIRole.kGeneralUser

Returns : *Promise<void>*

## kickRemoteUserOutOfRoom

Kick Out User from Room (Only Host and Administrator can call this Interface)



```
const roomEngine = new TUIRoomEngine();
await roomEngine.kickRemoteUserOutOfRoom({
  userId: 'user_1234',
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID

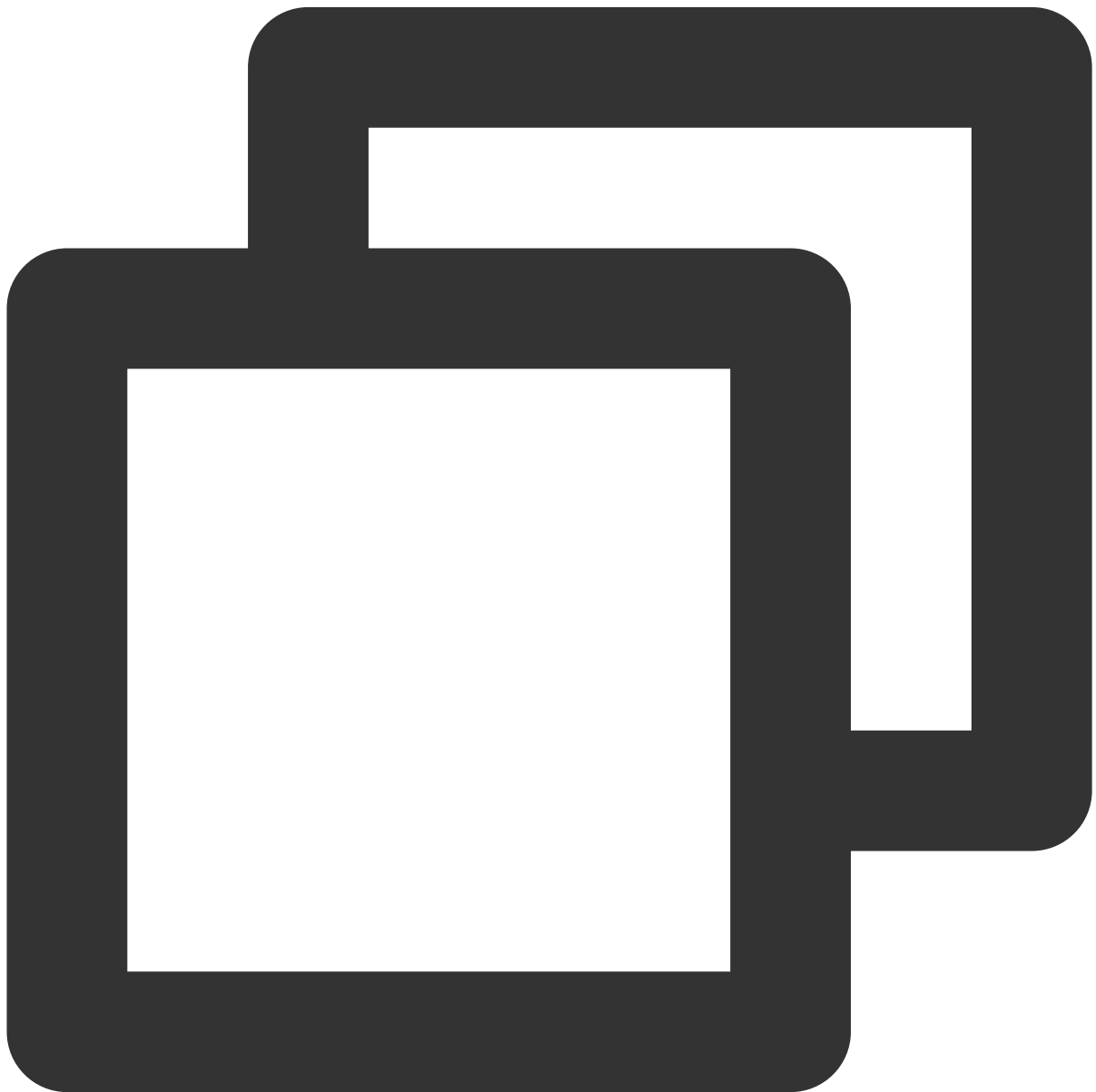
Returns : *Promise<void>*

## setMaxSeatCount

Set Room Seat Maximum Value

When roomType is TUIRoomType.kConference (Education and Conference Scene), maxSeatCount value is not limited;

When roomType is TUIRoomType.kLivingRoom (Live Scene), maxSeatCount is limited to 16;



```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.setMaxSeatCount({ maxSeatCount: 16 })
```

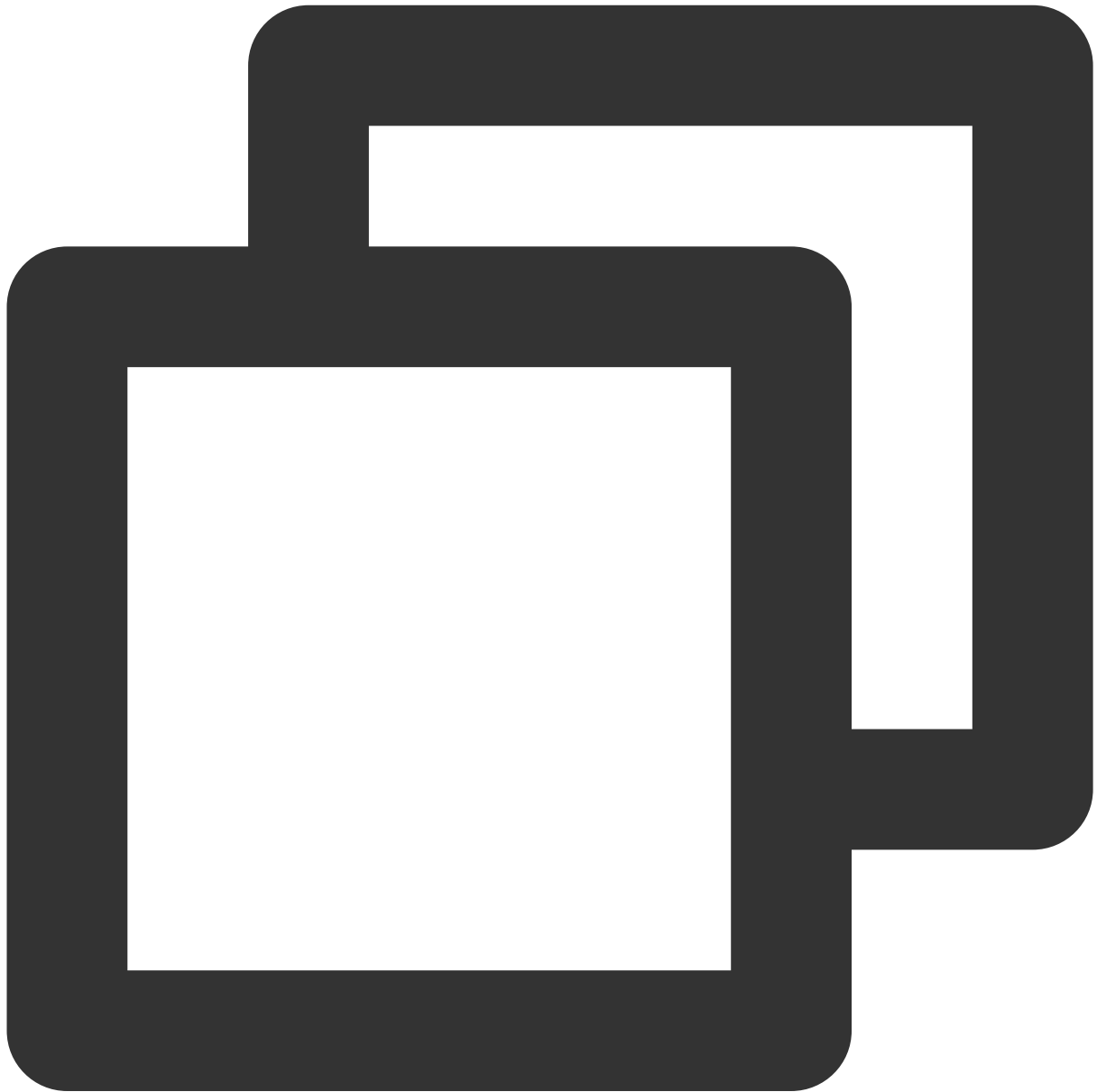
Parameter:

Parameter	Type	Description	Default Value	Meaning
maxSeatCount	number	Required	-	Set Room Seat Maximum Value

Returns : Promise<void>

## getSeatList

Get Seat List



```
const roomEngine = new TUIRoomEngine();  
const seatList = await roomEngine.getSeatList();
```

Returns : *Promise*<[TUISeatInfo](#)[]> seatList  
seatList for the Current Room's All Seat List

## takeSeat

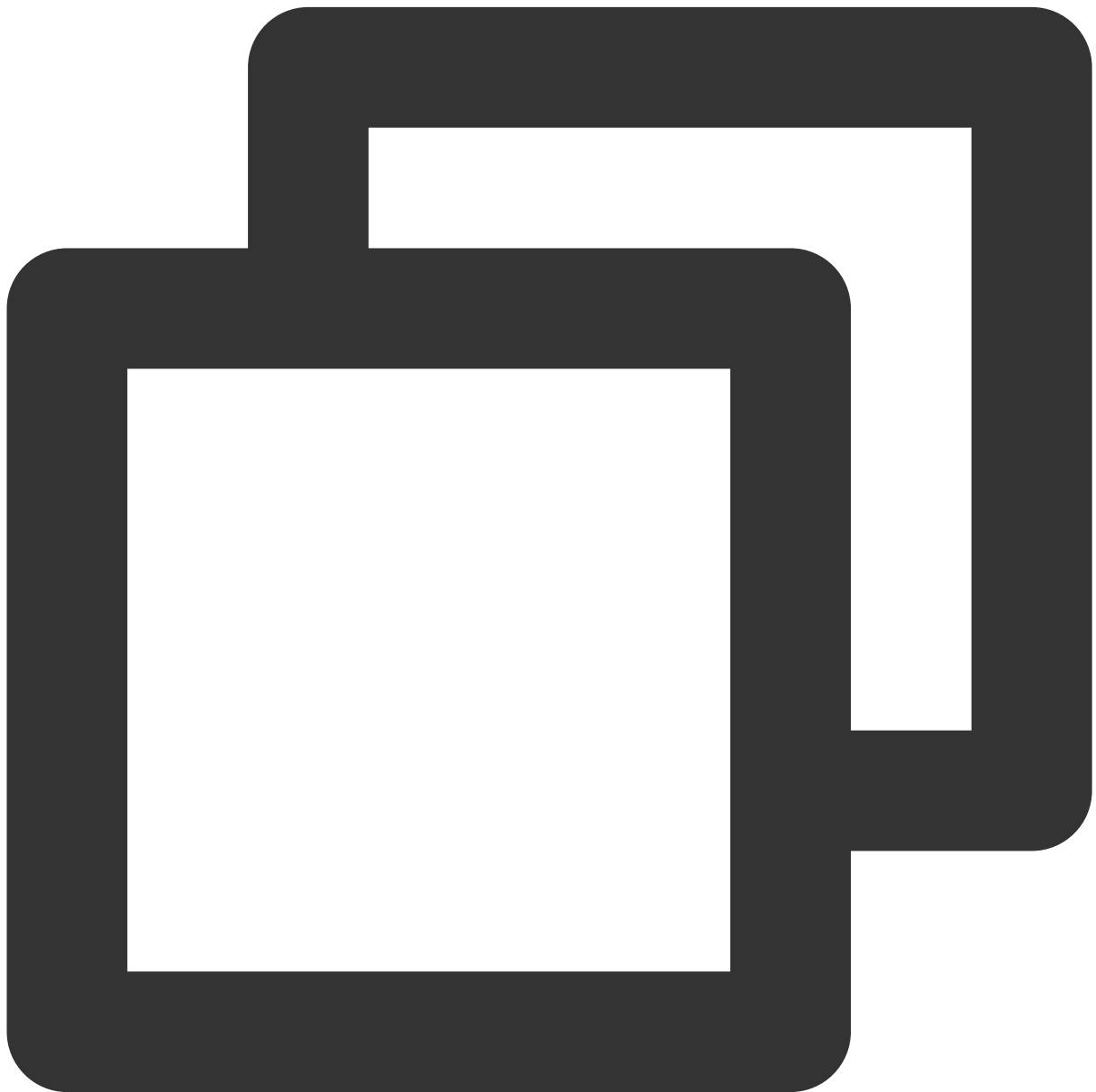


Mic-off Users can call `takeSeat` to become Mic-on Users, only Mic-on Users can Publish Local stream.

When `roomInfo.roomType` is `TUIRoomType.kConference` and `roomInfo.speechMode` is `TUISpeechMode.kSpeakAfterTakingSeat`, General Users need to Wait for the Host/Administrator's Agreement to become Mic-on Users after calling `takeSeat` method.

When `roomInfo.roomType` is `TUIRoomType.kLivingRoom` and `roomInfo.speechMode` is `TUISpeechMode.kFreeToSpeak`, General Users become Mic-on Users after calling `takeSeat` method Successfully. Host & Administrator become Mic-on Users after calling `takeSeat` Successfully.

Changes of Mic-on Users are Notified to All Users through `TUIRoomEvents.onSeatListChanged`.



```
const roomEngine = new TUIRoomEngine();
```

```
// Scenario 1: Host/Administrator Go Live
// Scenario 2: When roomInfo.roomType is TUIRoomType.kConference
// and roomInfo.speechMode is TUISpeechMode.kSpeakAfterTakingSeat, General Users Go
await roomEngine.takeSeat({
  seatIndex: -1,
  timeout: 0,
});

// Scenario 3: When roomInfo.enableSeatControl is true, General Users Go Live
const requestId = await roomEngine.instance?.takeSeat({
  seatIndex: -1,
  timeout: 0,
  requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
    switch (requestCallbackType) {
      case TUIRequestCallbackType.kRequestAccepted:
        // Request Accepted
        break;
      case TUIRequestCallbackType.kRequestRejected:
        // Request Rejected
        break;
      case TUIRequestCallbackType.kRequestCancelled:
        // Request Canceled
        break;
      case TUIRequestCallbackType.kRequestTimeout:
        // Request Timeout
        break;
      case TUIRequestCallbackType.kRequestError:
        // Request Error
        break;
      default:
        break;
    }
  },
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
seatIndex	number	Required	-	Seat index, set to -1 when there is No Serial Number
timeout	number	Required	-	Timeout. If timeout is set to 0, there is no Timeout
requestCallback	Function	Optional	Empty Function	Callback, used to Notify Initiator of Request being Accepted/Rejected/Cancelled/Timeout/Error

Returns : Promise<string> requestId

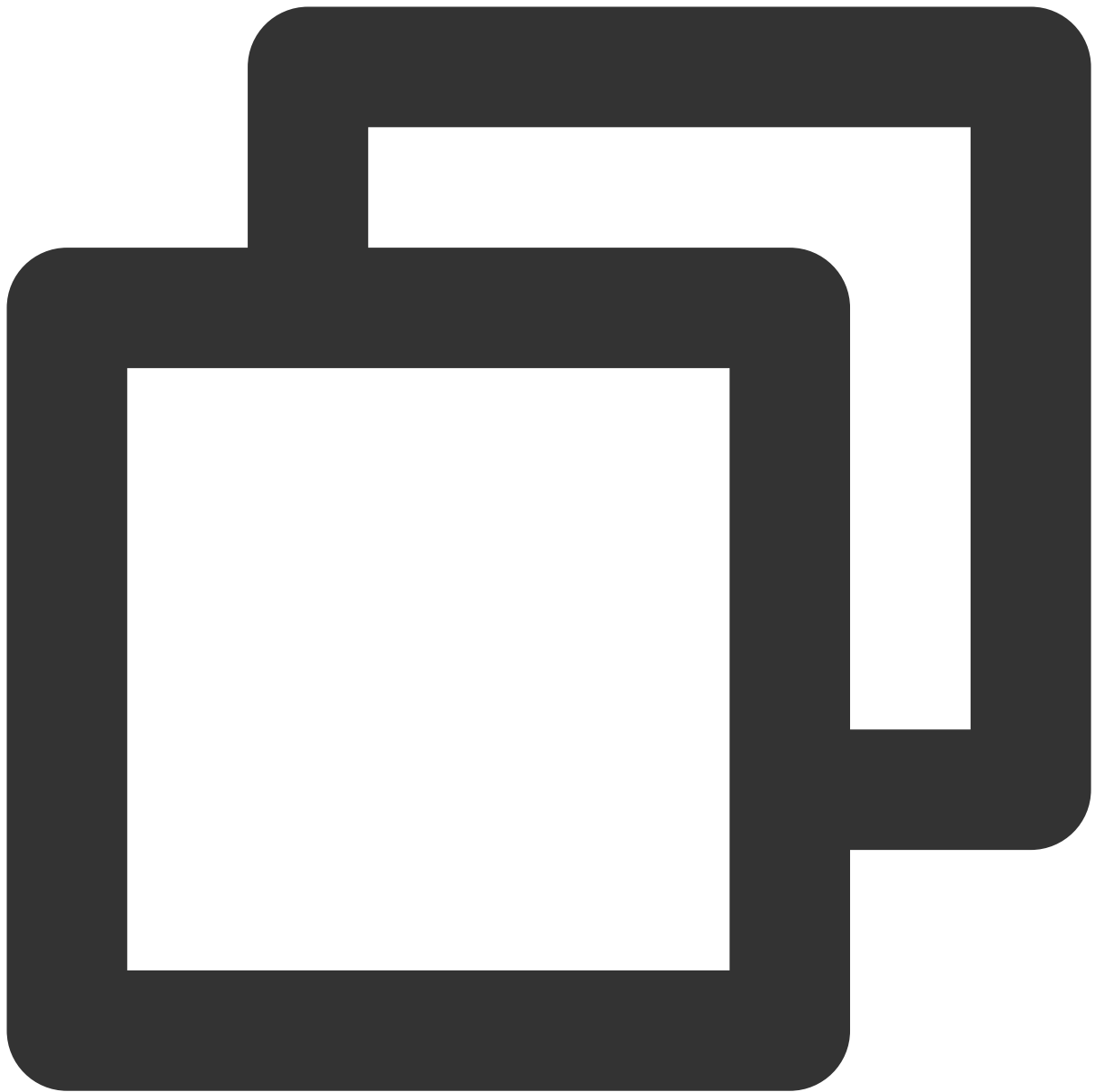
When roomInfo.enableSeatControl is true, General Users call this Interface to return requestId, General Users can use this requestId to call cancelRequest Interface to cancel Go Live Request.

**Note:**

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number;

## leaveSeat

Release Seat



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.leaveSeat();
```

Returns : *Promise<void>*

## takeUserOnSeatByAdmin

Invite Others to Go Live



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.takeUserOnSeatByAdmin({
  seatIndex: 0,
  userId: 'user_1234',
  timeout: 0,
  requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
    switch (requestCallbackType) {
      case TUIRequestCallbackType.kRequestAccepted:
        // Request Accepted
        break;
      case TUIRequestCallbackType.kRequestRejected:
```

```
// Request Rejected
break;
case TUIRequestCallbackType.kRequestCancelled:
    // Request Canceled
    break;
case TUIRequestCallbackType.kRequestTimeout:
    // Request Timeout
    break;
case TUIRequestCallbackType.kRequestError:
    // Request Error
    break;
default:
    break;
}
},
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
seatIndex	number	Required	-	Seat index
userId	string	Required	-	User ID
timeout	number	Required	-	Timeout. If timeout is set to 0, there is no Timeout
requestCallback	Function	Optional	Empty Function	Callback, used to Notify Initiator of Request being Accepted/Rejected/Cancelled/Timeout/Error

Returns : *Promise<string>* requestId

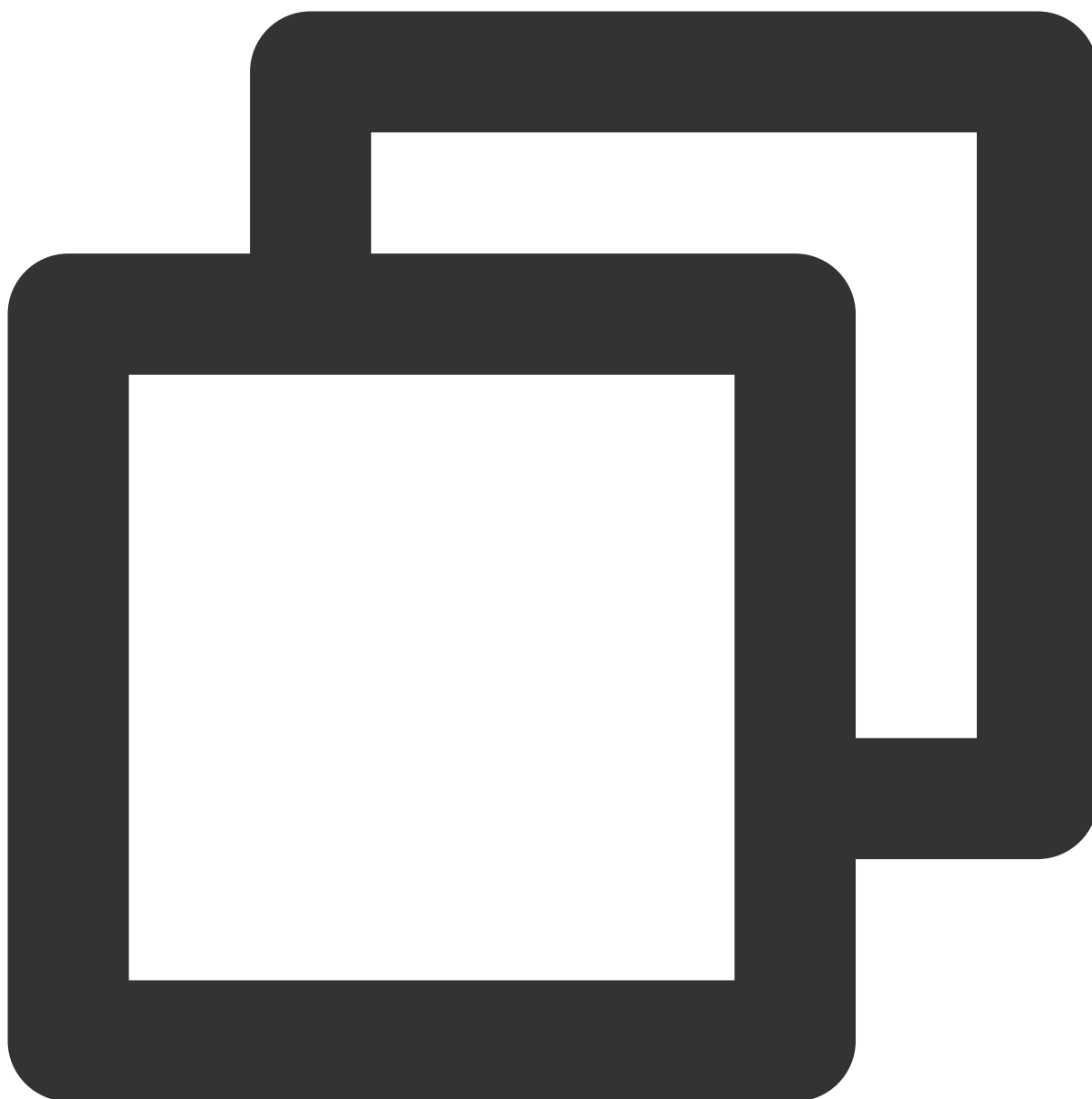
This Interface returns requestId, Users can use this requestId to call cancelRequest Interface to cancel Request

**Note:**

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number

## kickUserOffSeatByAdmin

Ask others to get off the mic



```
const roomEngine = new TUIRoomEngine();
const requestId = await roomEngine.kickUserOffSeatByAdmin({
  seatIndex: 0,
  userId: 'user_1234',
});
```

Parameter:

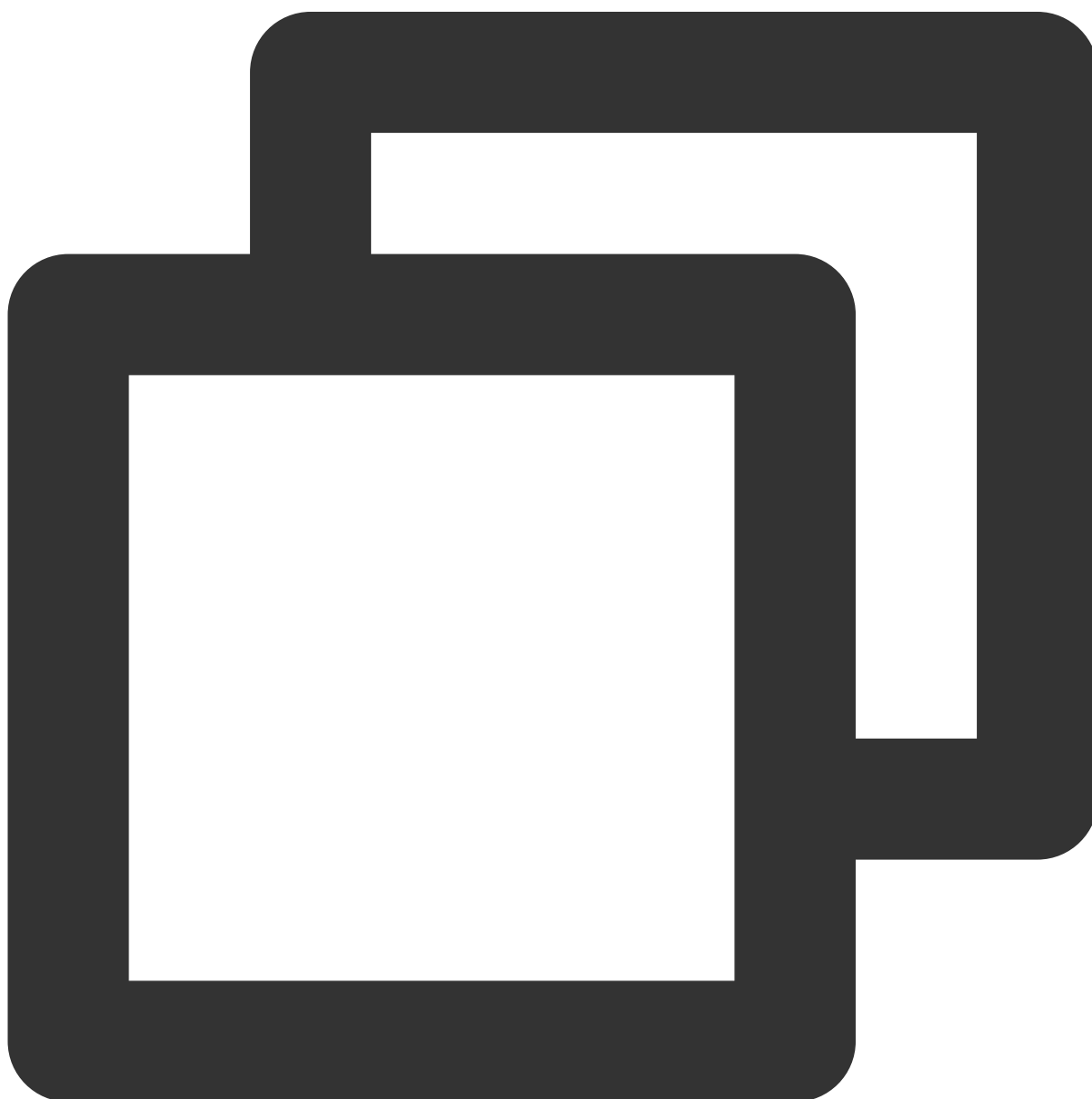
Parameter	Type	Description	Default Value	Meaning

seatIndex	number	Required	-	Seat index
userId	string	Required	-	User ID

Returns : *Promise<void>*

## lockSeatByAdmin

Lock a certain mic position status (Only the room host and administrator can call this method)





```
const roomEngine = new TUIRoomEngine();
await roomEngine.lockSeatByAdmin({
  seatIndex: 0,
  lockParams: {
    lockSeat: true,
    lockVideo: true,
    lockAudio: true,
  },
});
```

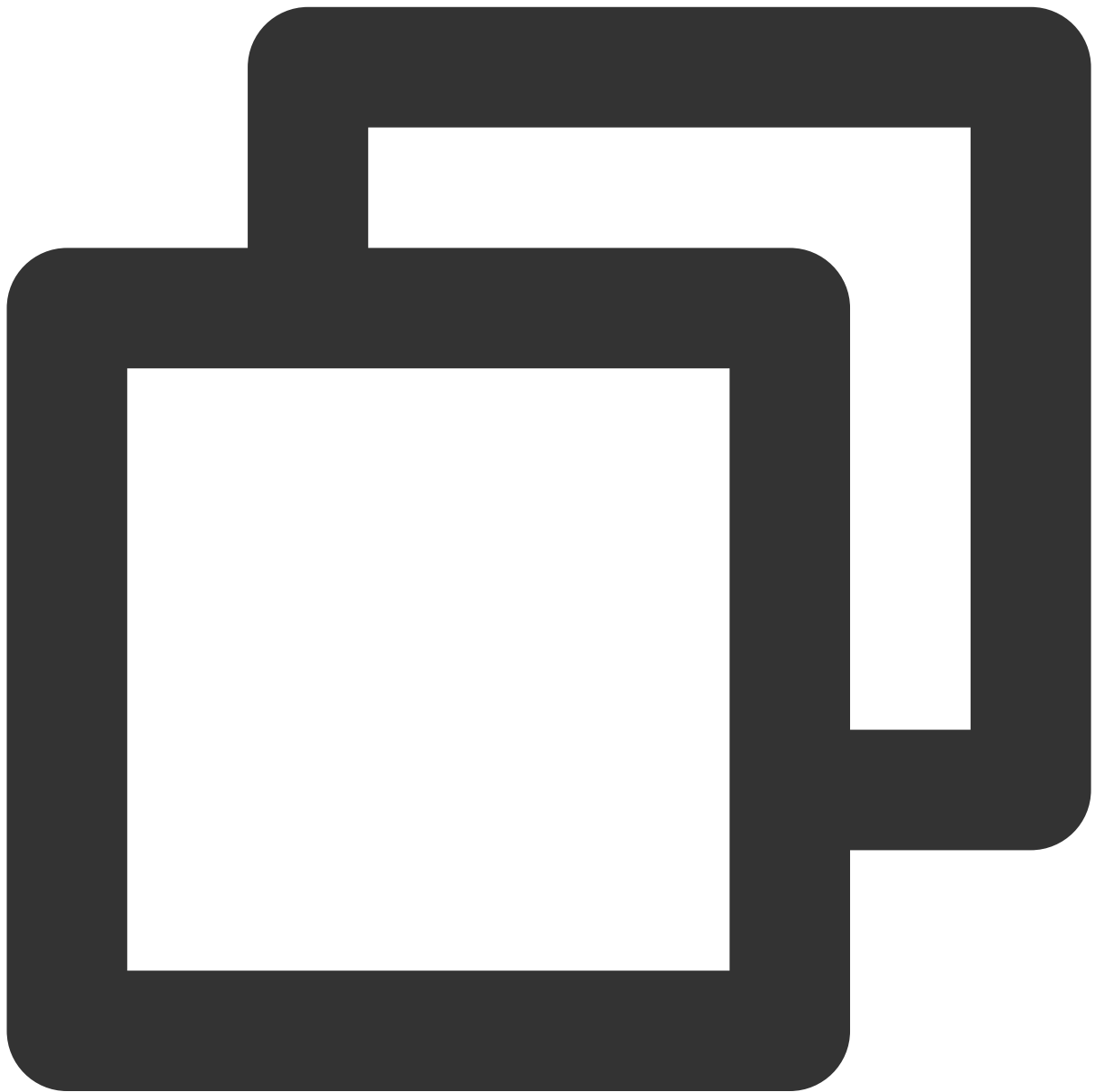
Parameter:

Parameter	Type	Description	Default Value	Meaning
seatIndex	number	Required	-	Mic position index
lockParams	TUISeatLockParams	Required	-	Lock mic parameter

Returns : *Promise<void>*

## sendTextMessage

Send text message



```
const roomEngine = new TUIRoomEngine();
await roomEngine.sendTextMessage({
  messageText: 'hello, everyone',
});
```

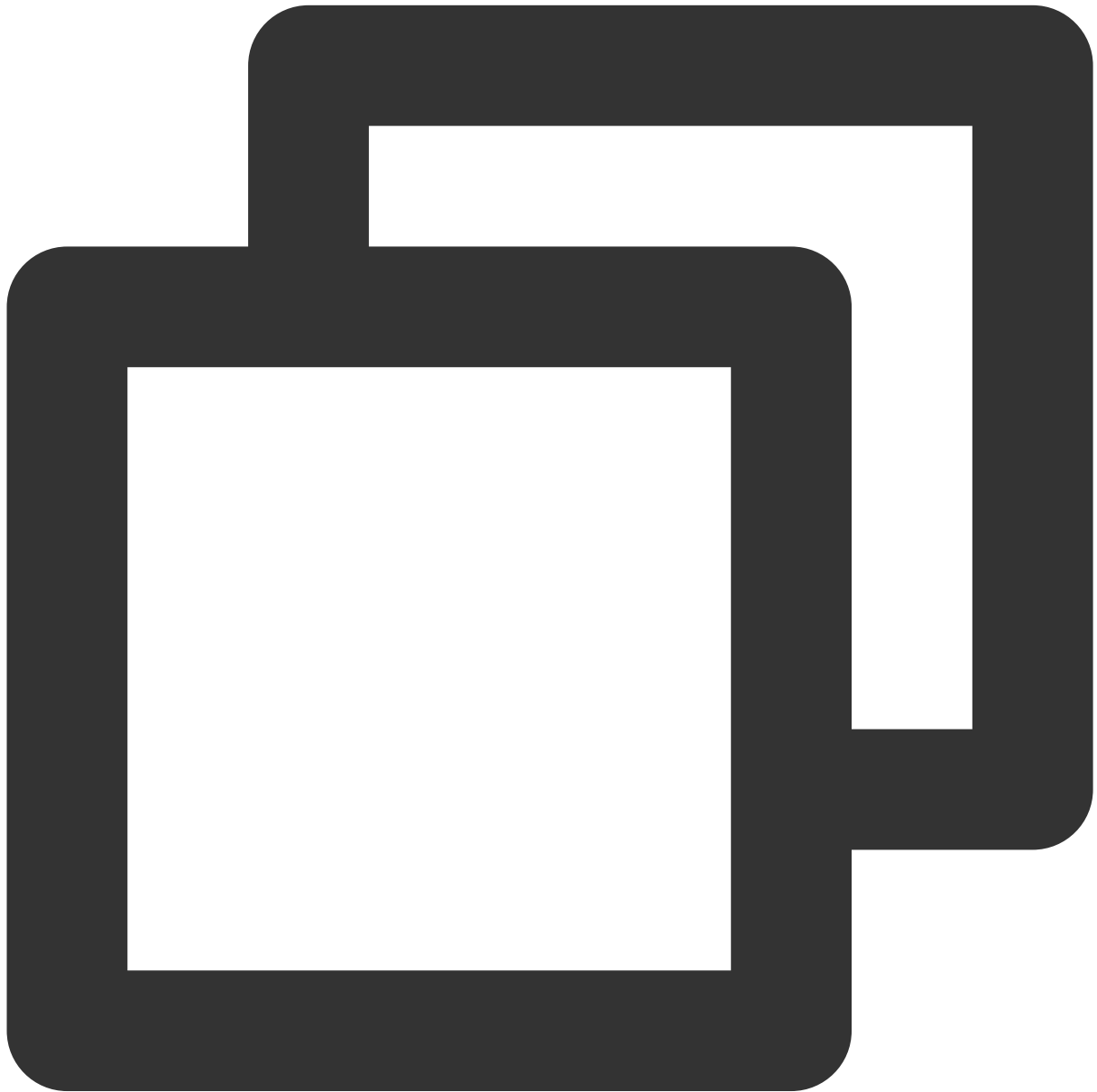
Parameter:

Parameter	Type	Description	Default Value	Meaning
messageText	string	Required	-	Text message content

Returns : *Promise<void>*

## sendCustomMessage

Send custom message



```
const roomEngine = new TUIRoomEngine();
await roomEngine.sendCustomMessage({
  messageText: '{ data:', description: '}',
```

```
});
```

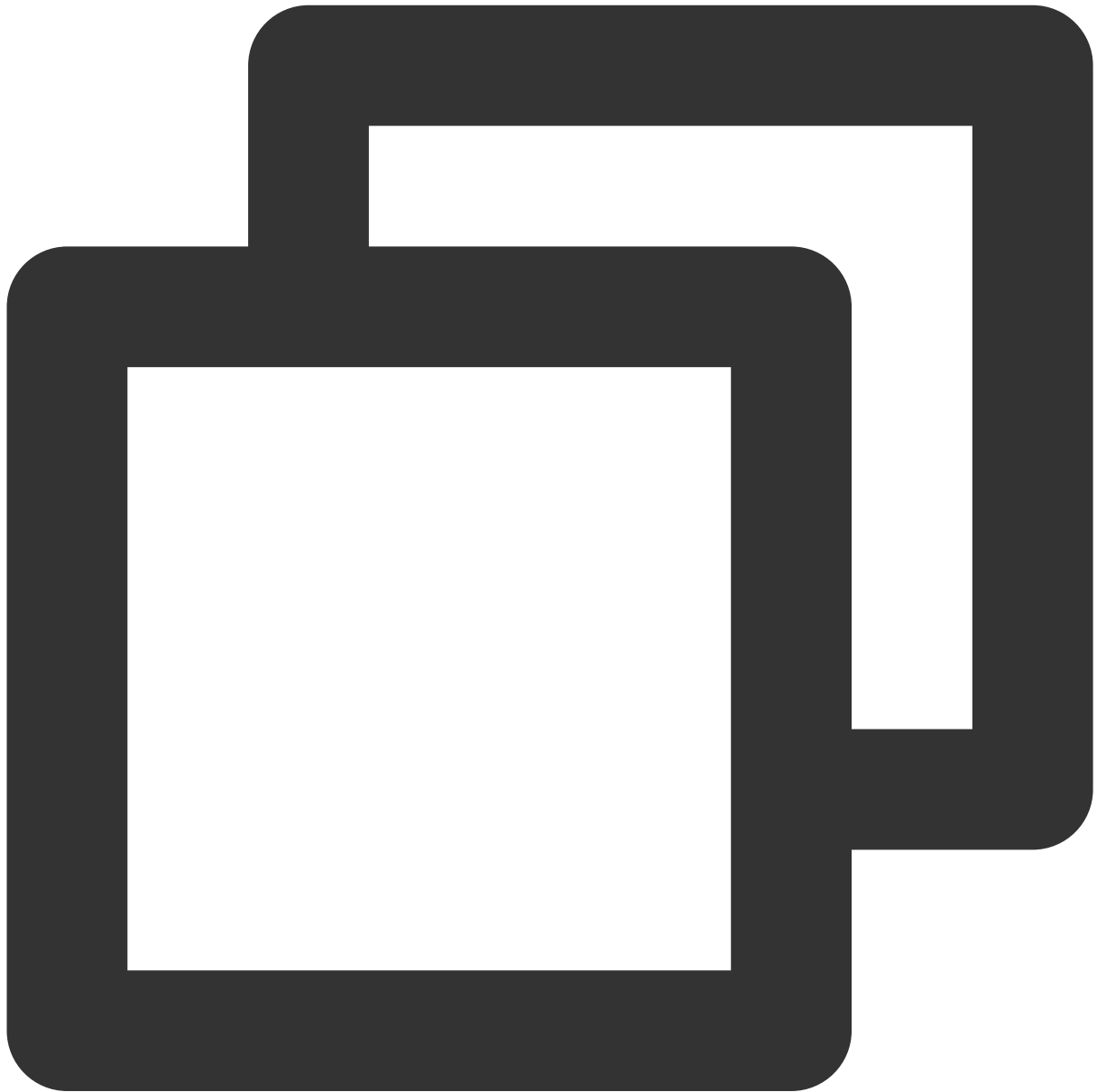
Parameter:

Parameter	Type	Description	Default Value	Meaning
messageText	string	Required	-	Custom message content

Returns : *Promise<void>*

## getCameraDevicesList

Get camera device list

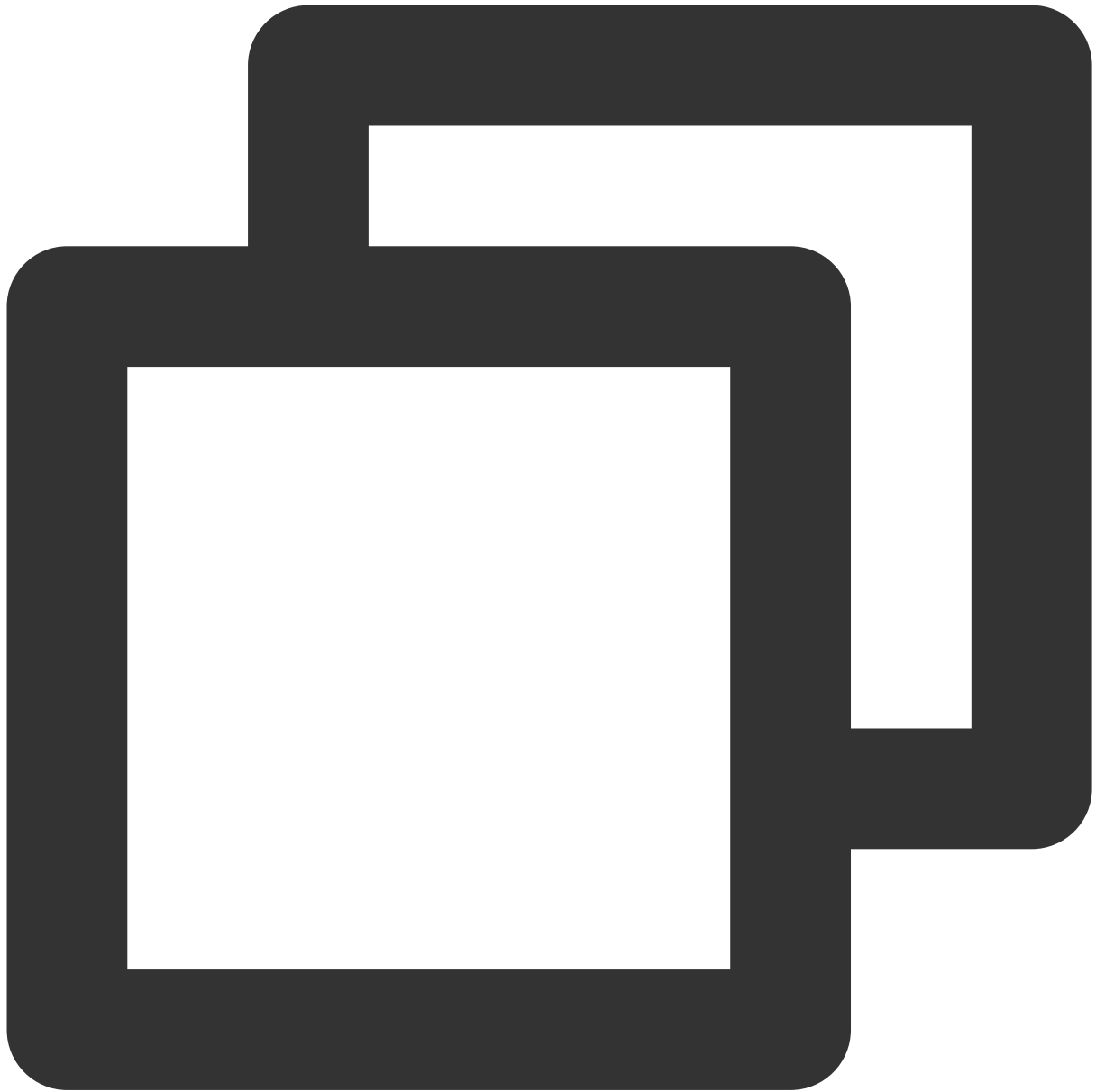


```
const roomEngine = new TUIRoomEngine();
const cameraList = await roomEngine.getCameraDevicesList();
for (i = 0; i < cameraList.length; i++) {
  var camera = cameraList[i];
  console.info("camera deviceName: " + camera.deviceName + " deviceId:" + camera.
}
```

Returns: *Promise*<[TRTCDeviceInfo\[\]](#)> cameraList

## getMicDevicesList

Get mic device list

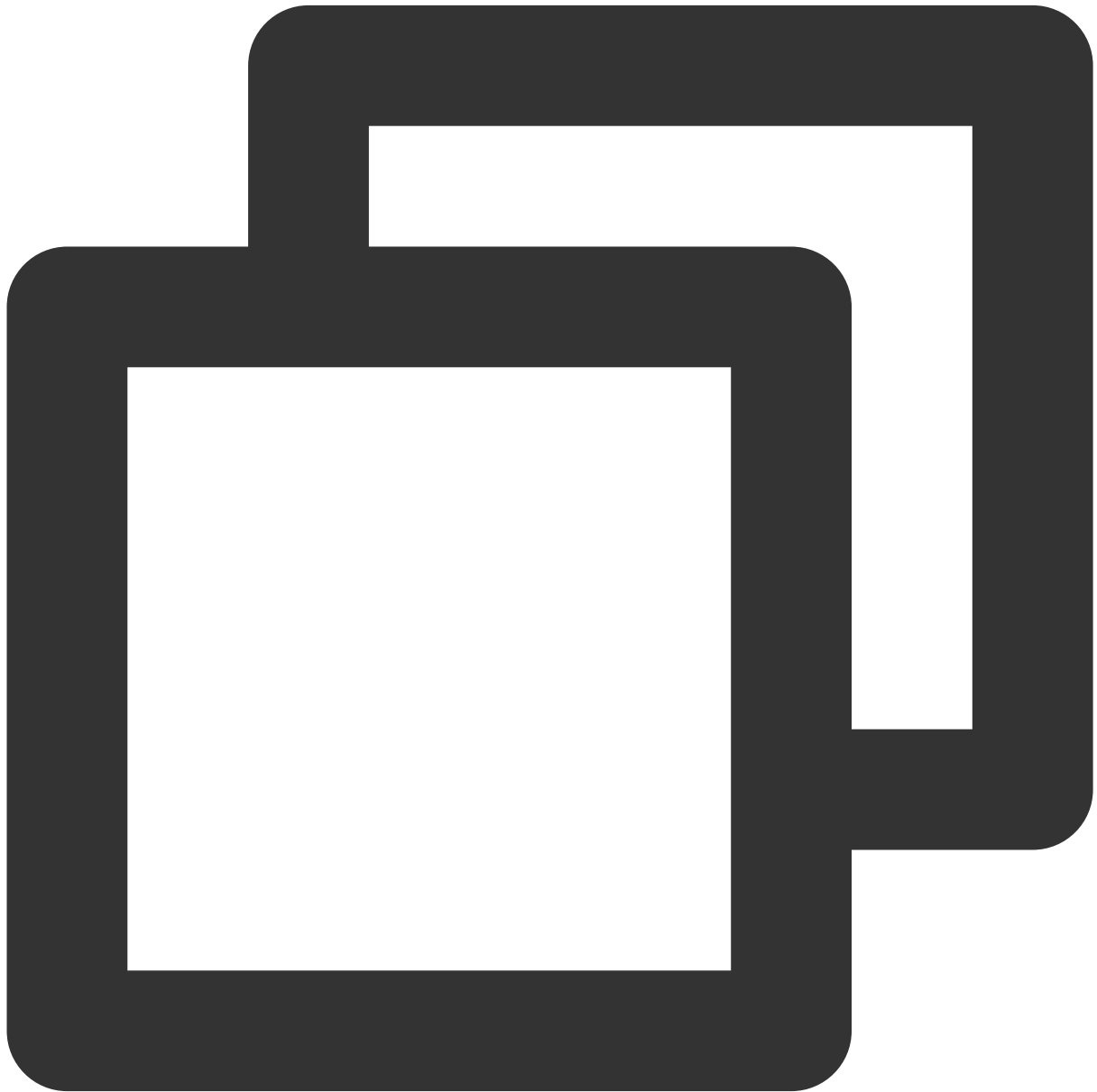


```
const roomEngine = new TUIRoomEngine();
const micList = await roomEngine.getMicDevicesList();
for (i = 0; i < micList.length; i++) {
  var mic = micList[i];
  console.info("mic deviceName: " + mic.deviceName + " deviceId:" + mic.deviceId)
}
```

Returns: *Promise*<[TRTCDDeviceInfo](#)[]> micList

## getSpeakerDevicesList

Get speaker device list



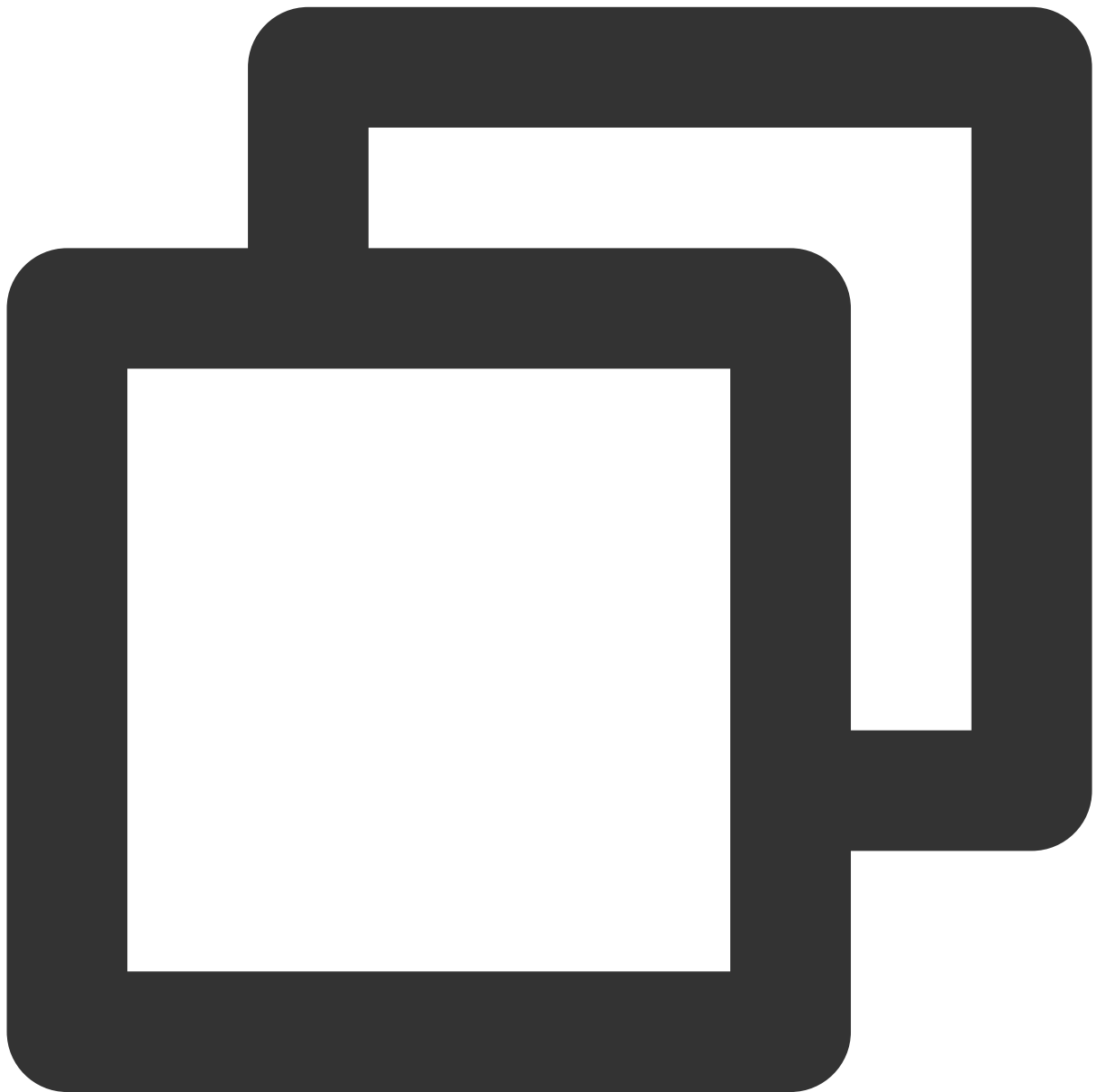
```
const roomEngine = new TUIRoomEngine();
const speakerList = await roomEngine.getSpeakerDevicesList();
for (i = 0; i < speakerList.length; i++) {
```

```
var speaker = speakerList[i];  
console.info("speaker deviceName: " + speaker.deviceName + " deviceId:" + speaker.deviceId);  
}
```

Returns: *Promise*<[TRTCDeviceInfo](#)[]> speakerList

## setCurrentCameraDevice

Set the camera device to be used





```
const roomEngine = new TUIRoomEngine();
await roomEngine.setCurrentCameraDevice({ deviceId: '' });
```

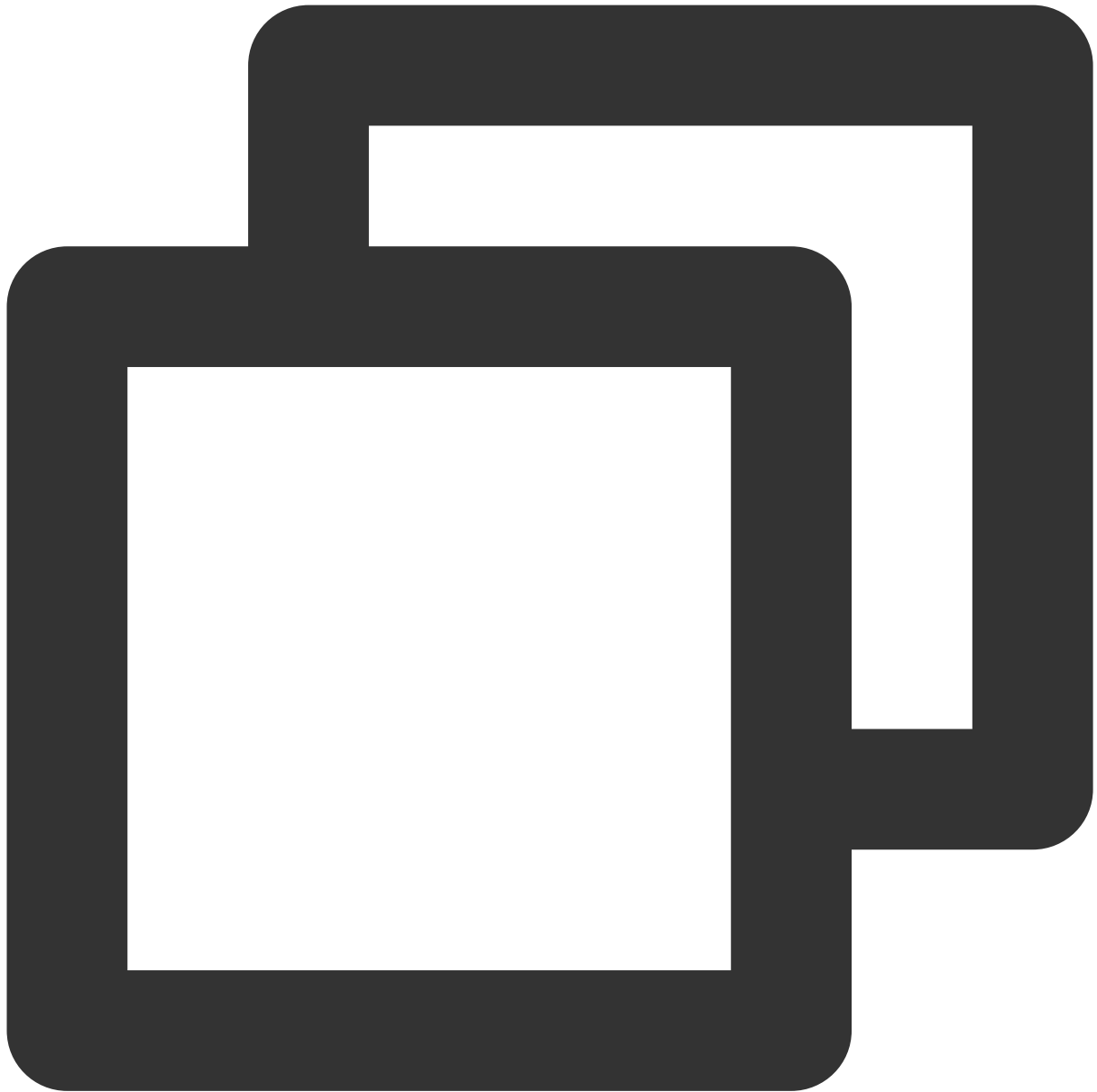
Parameter:

Parameter	Type	Description	Default Value	Meaning
deviceId	string	Required	-	Device ID obtained from getCameraDevicesList

Returns : *void*

## setCurrentMicDevice

Set the mic device to be used



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.setCurrentMicDevice({ deviceId: '' });
```

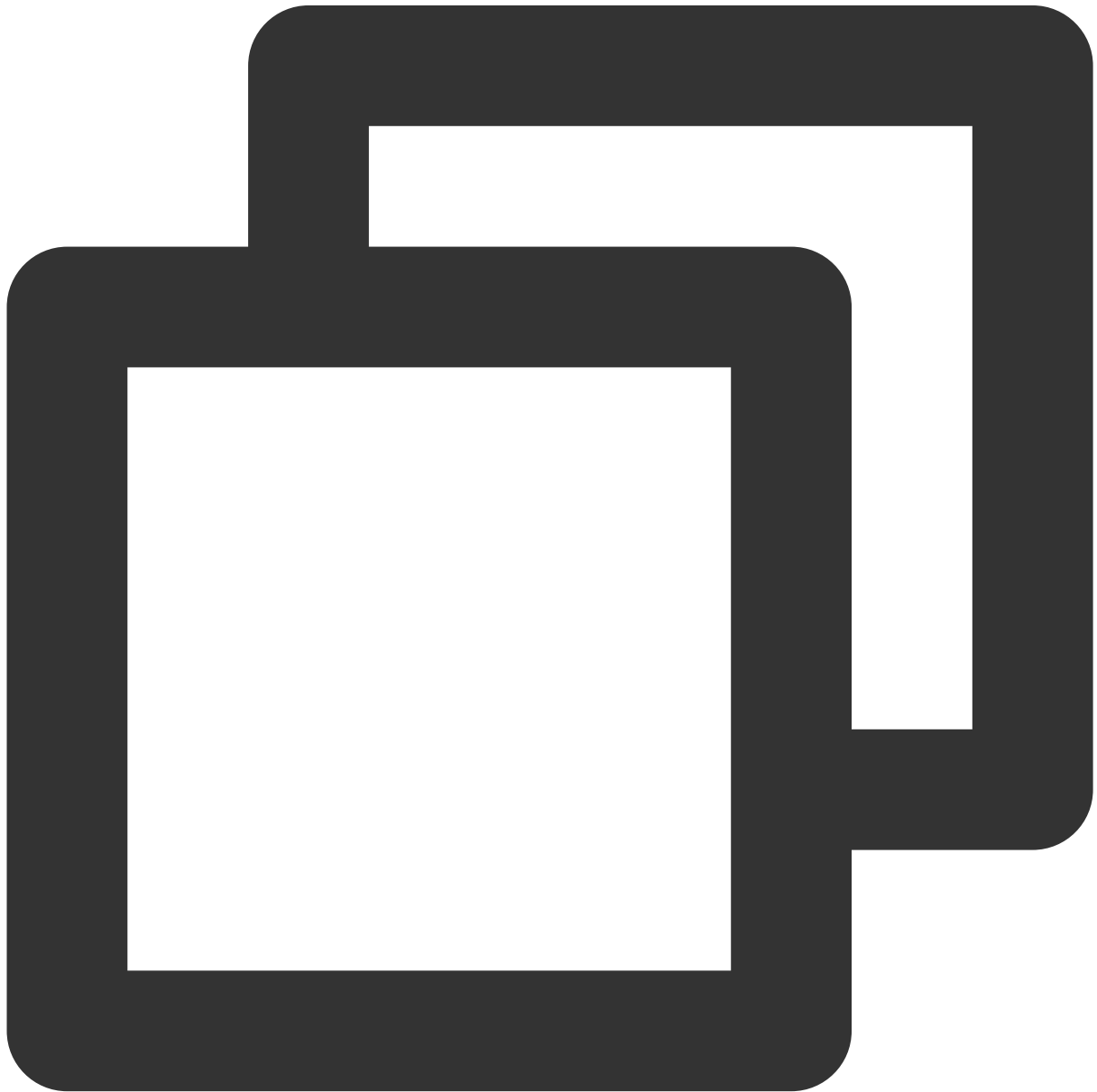
Parameter:

Parameter	Type	Description	Default Value	Meaning
deviceId	string	Required	-	Device ID obtained from getMicDevicesList

Returns : *void*

## setCurrentSpeakerDevice

Set the speaker device to be used



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.setCurrentSpeakerDevice({ deviceId: '' });
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
deviceId	string	Required	-	Device ID obtained from getSpeakerDevicesList

Returns : *void*

## getCurrentCameraDevice

Get the currently used camera device



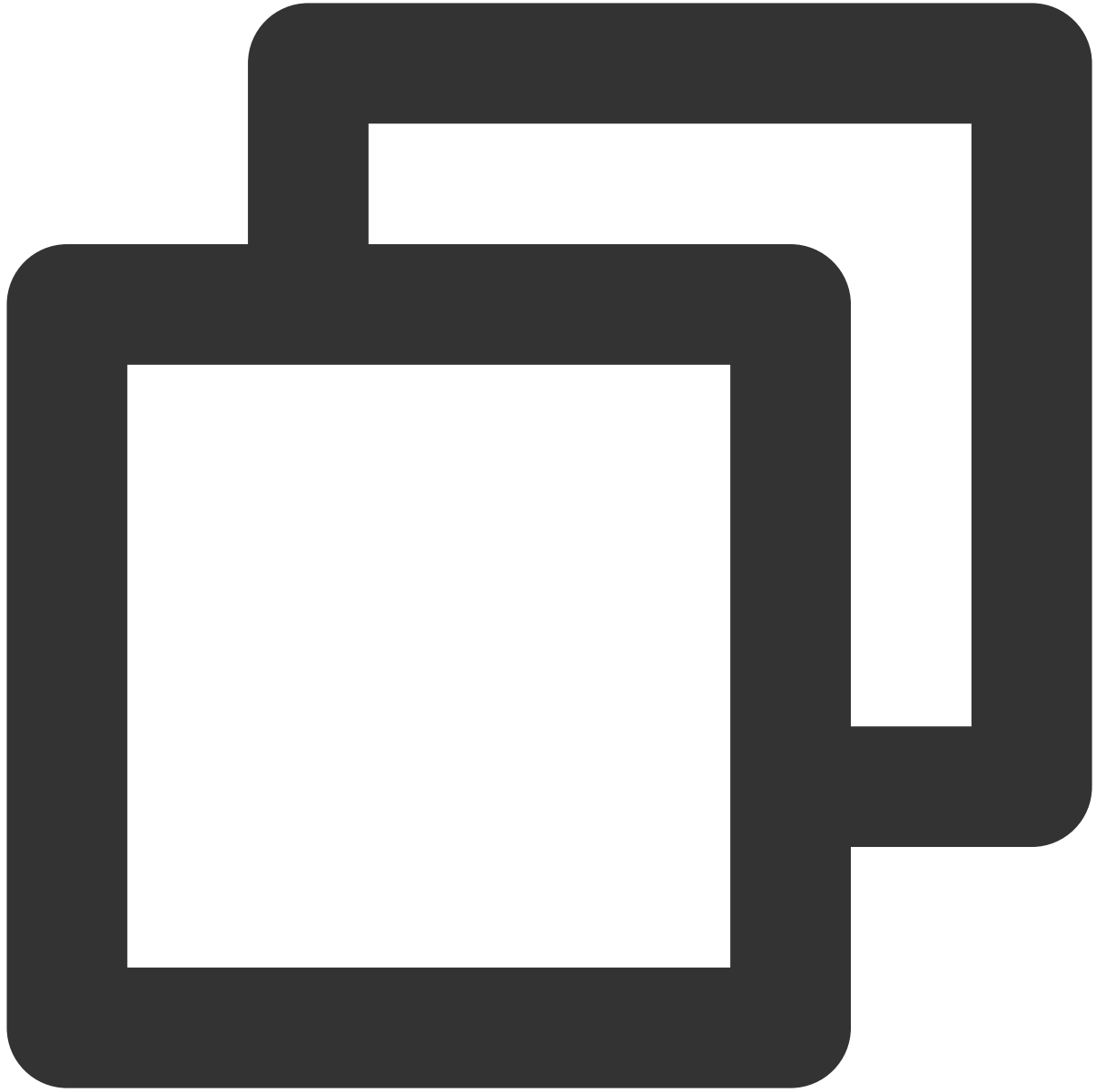
```
const roomEngine = new TUIRoomEngine();  
const currentCameraDevice = roomEngine.getCurrentCameraDevice();
```

Returns : [TRTCDeviceInfo](#)

Device information, can get device ID and device name

## getCurrentMicDevice

Get the currently used mic device



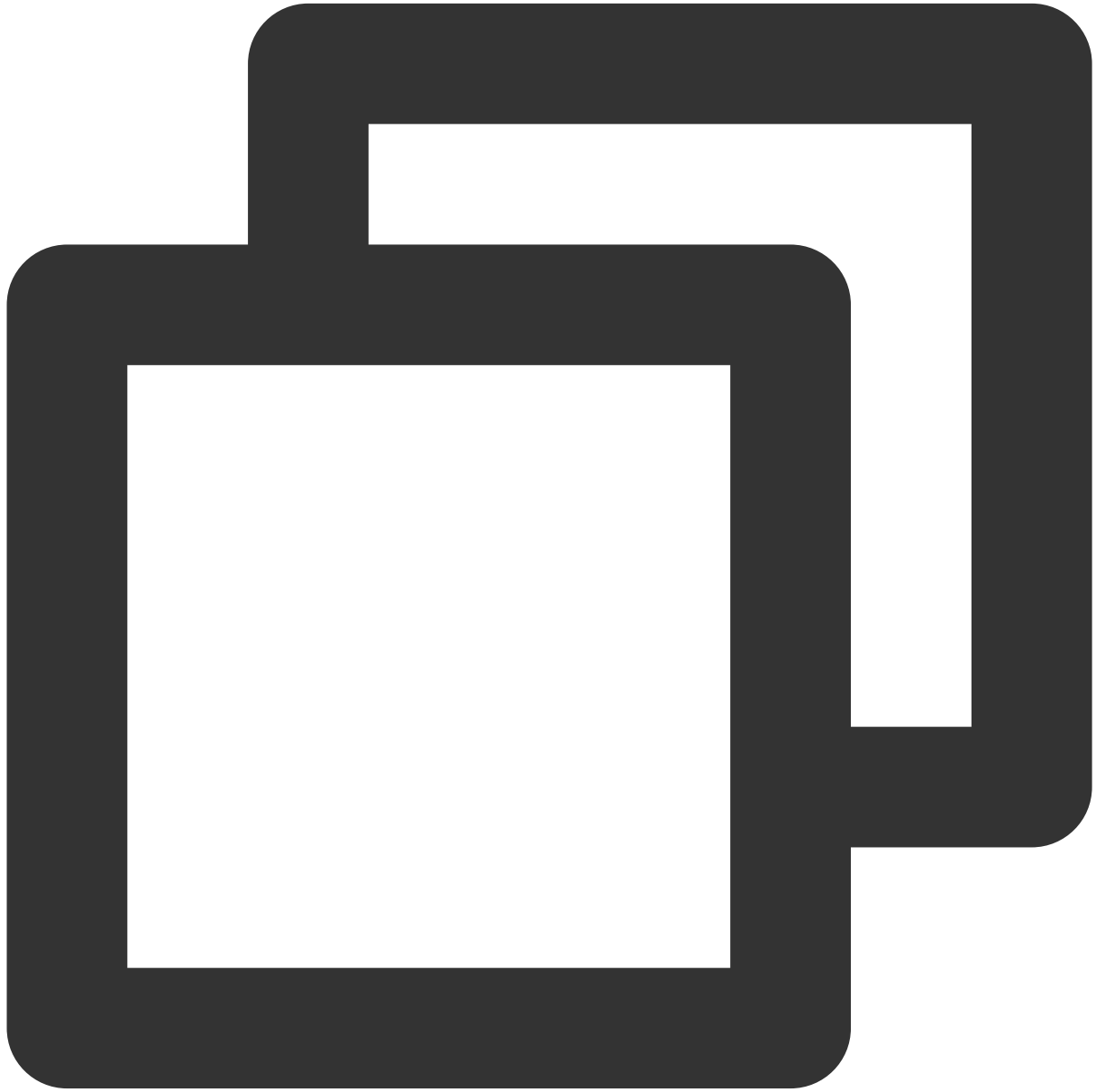
```
const roomEngine = new TUIRoomEngine();  
const currentMicDevice = roomEngine.getCurrentMicDevice();
```

Returns : [TRTCDeviceInfo](#)

Device information, can get device ID and device name

## getCurrentSpeakerDevice

Get the currently used speaker device



```
const roomEngine = new TUIRoomEngine();  
const currentSpeakerDevice = roomEngine.getCurrentSpeakerDevice();
```

Returns : [TRTCDeviceInfo](#)

Device information, can get device ID and device name

## startCameraDeviceTest

Start camera test



```
const roomEngine = new TUIRoomEngine();
await roomEngine.startCameraDeviceTest({ view: 'test-preview' });
```

Parameter:

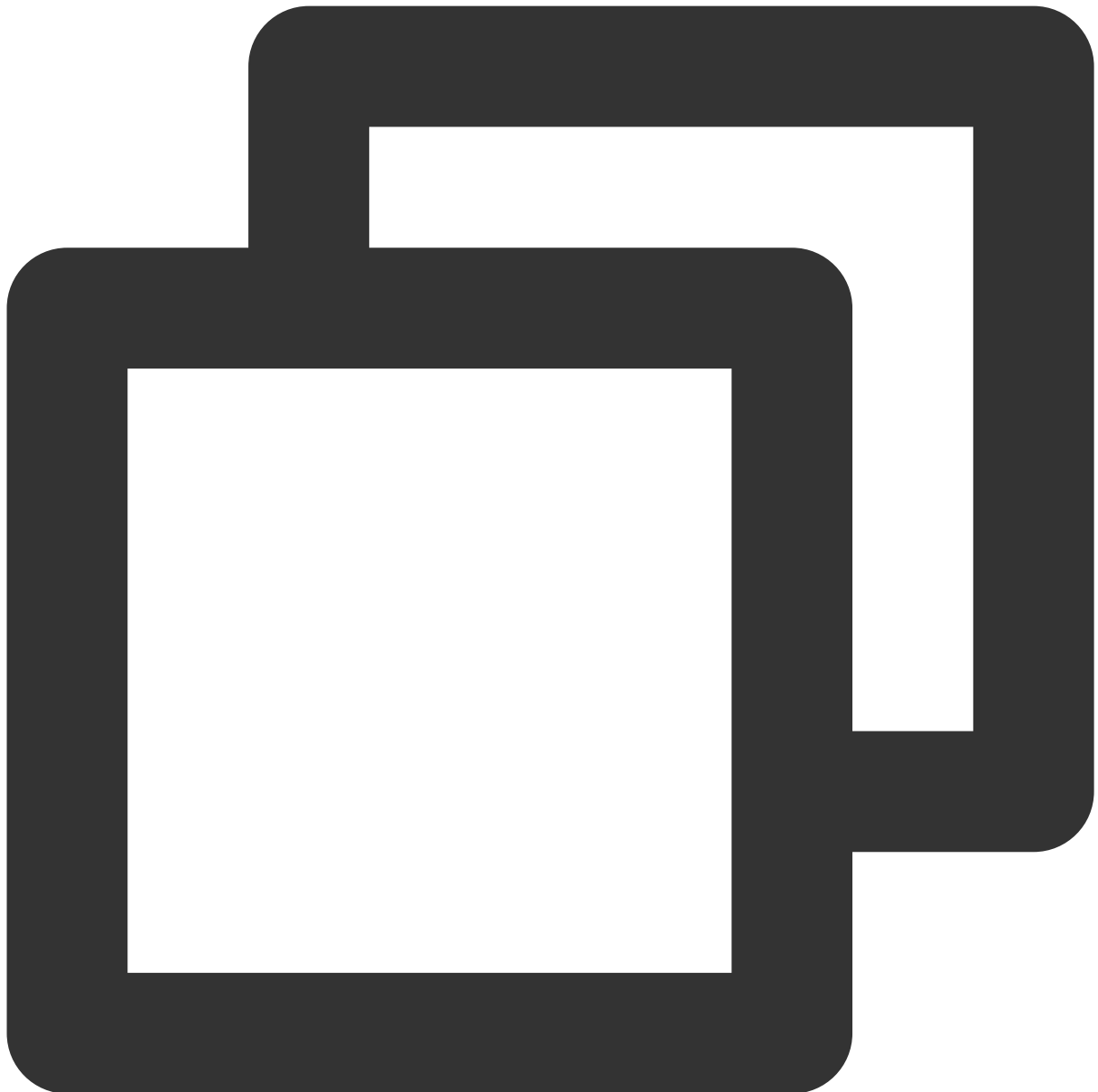
Parameter	Type	Description	Default Value	Meaning
view	string	Required	-	Display the video area of the camera test, the input view is the Id of the div element carrying the preview screen



Returns : *Promise<void>*

## stopCameraDeviceTest

Stop camera test

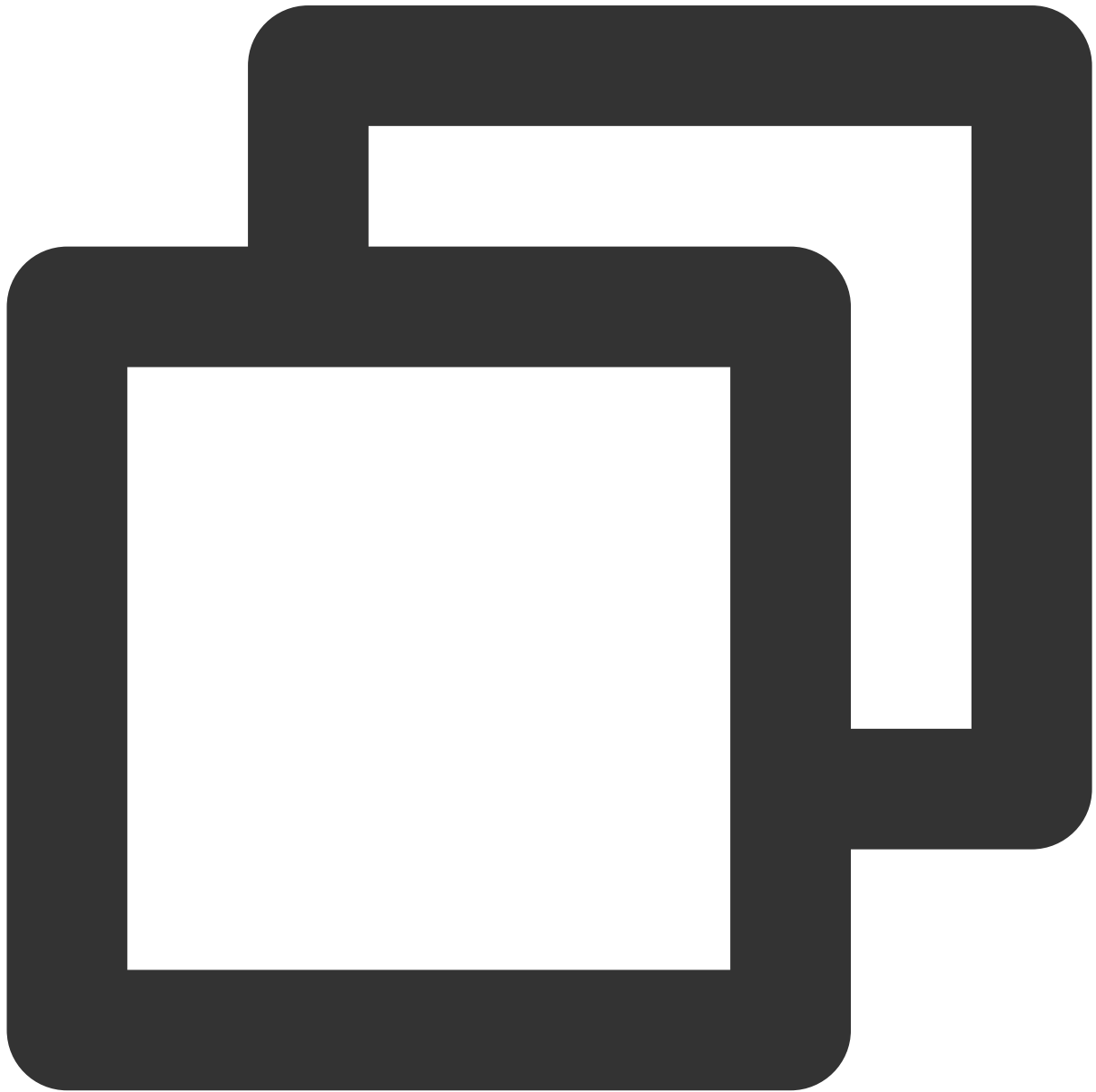


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.stopCameraDeviceTest();
```

Returns : *Promise<void>*

## on

Listen to roomEngine events



```
const roomEngine = new TUIRoomEngine();  
roomEngine.on(event, func);
```

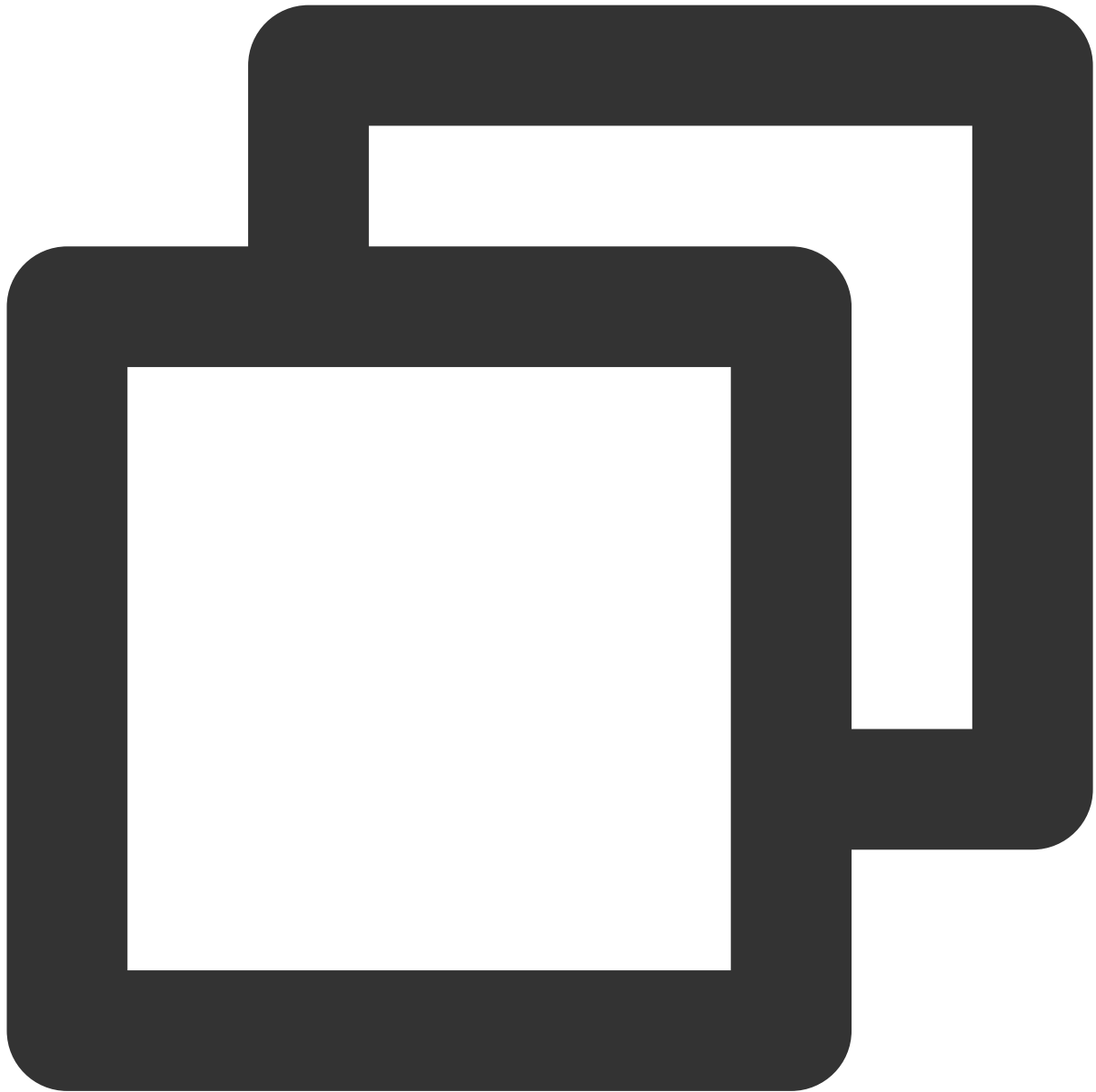
Parameter:

Parameter	Type	Description	Default Value	Meaning
event	<a href="#">TUIRoomEvents</a>	Required	-	TUIRoomEngine event list
func	Function	Required	-	Event callback function

Returns : *void*

## off

Cancel listening to roomEngine events



```
const roomEngine = new TUIRoomEngine();  
roomEngine.off(event, func);
```

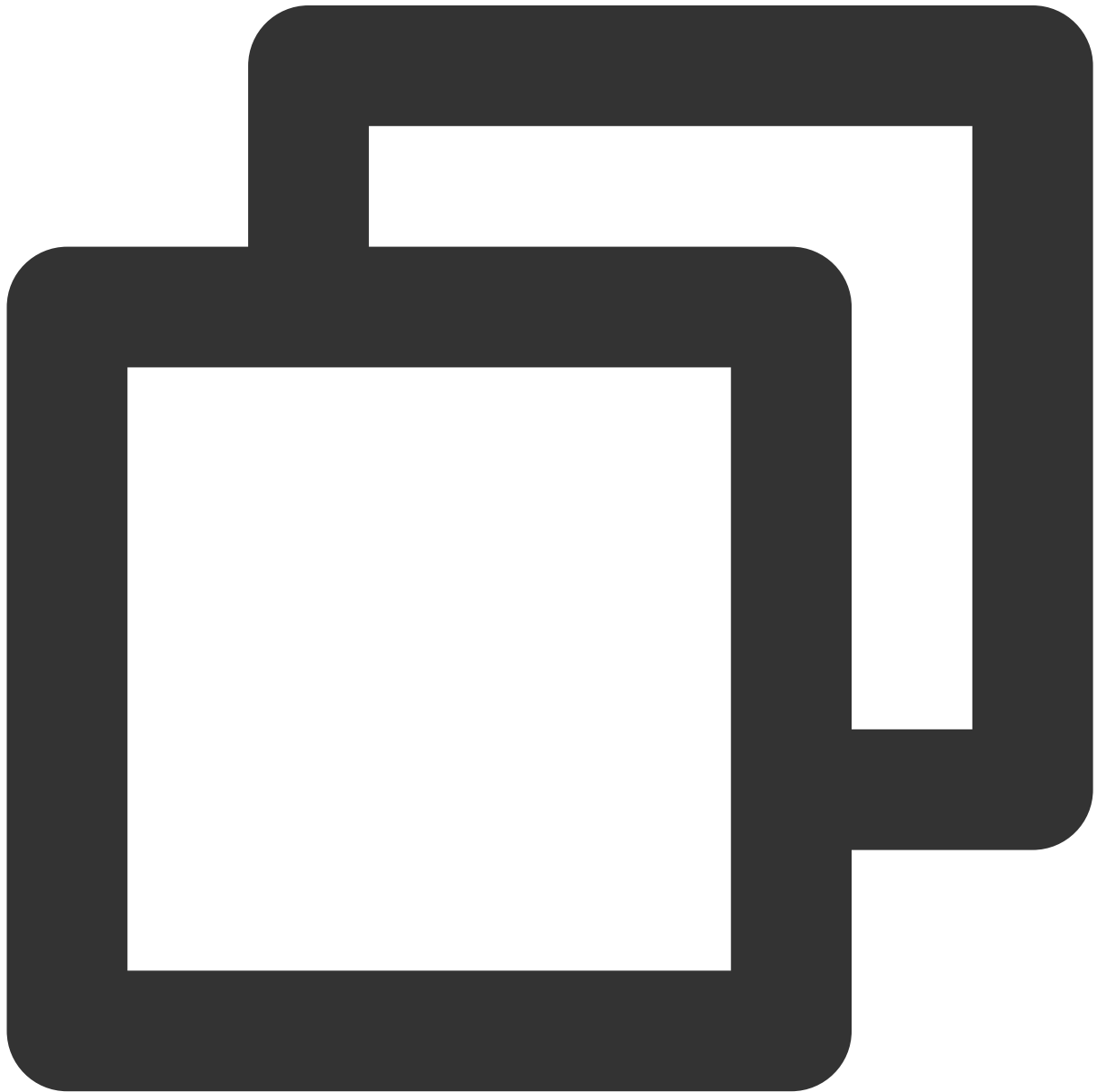
Parameter:

Parameter	Type	Description	Default Value	Meaning
event	<a href="#">TUIRoomEvents</a>	Required	-	TUIRoomEngine event list
func	Function	Required	-	Event callback function

Returns : *void*

## getTRTCCloud

Get trtcCloud instance, for web-side trtcCloud capabilities, please check: [TRTCCloud API documentation](#)

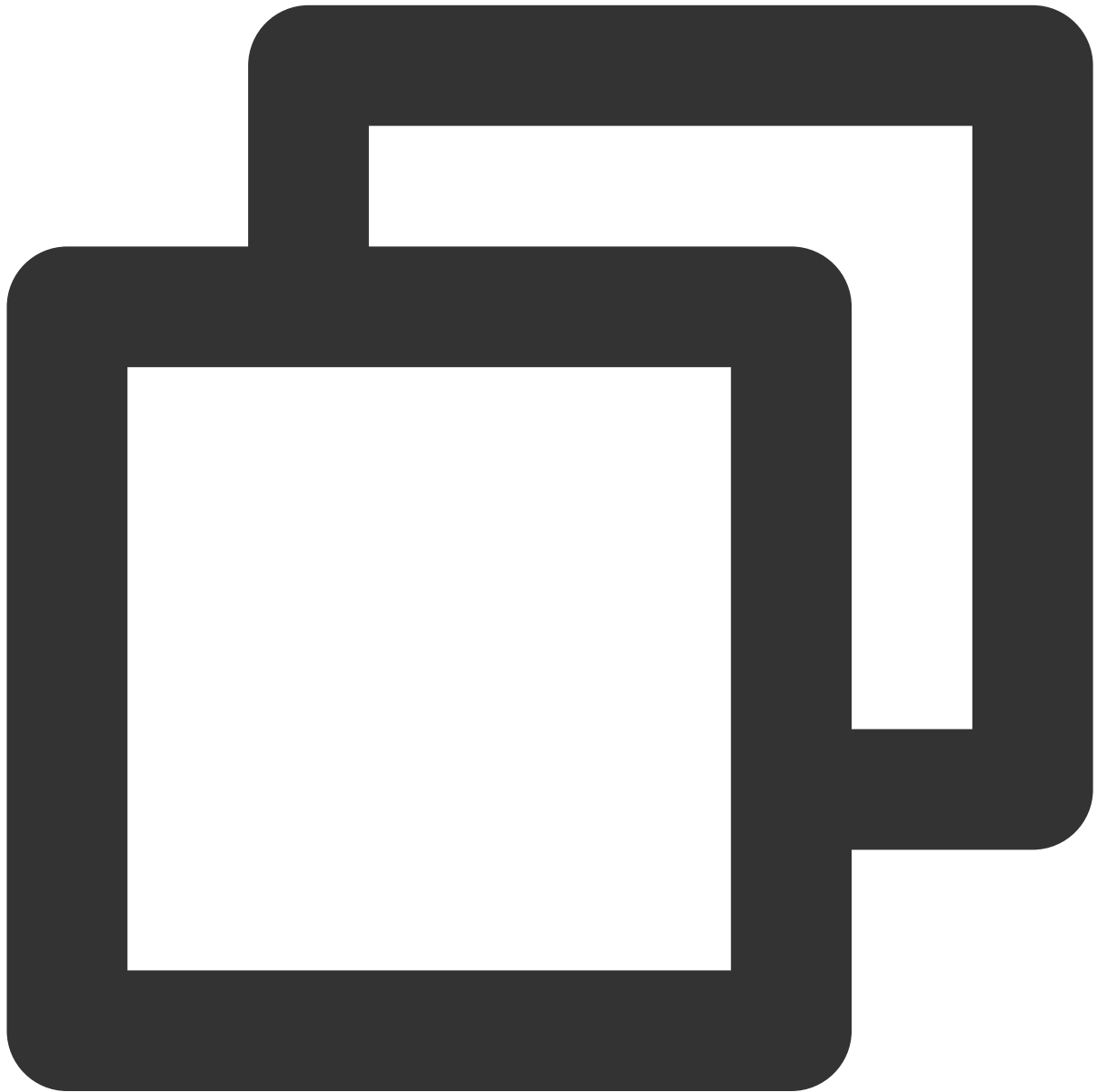


```
const roomEngine = new TUIRoomEngine();  
const trtcCloud = roomEngine.getTRTCCloud();
```

Returns : [TRTCCloud](#)

## getTIM

Get tim instance, for web-side tim capabilities, please check: [IM API documentation](#)



```
const roomEngine = new TUIRoomEngine();  
const trtcCloud = roomEngine.getTIM();
```

Returns : *TIM*

# TUIRoomEvents

Last updated : 2023-11-14 17:25:21

## TUIRoomEvent API Introduction

TUIRoomEvent API is the Event Interface for Audio/Video call Component.

### Event list

EVENT	Description
<a href="#">TUIRoomEvents.onError</a>	Error Event
<a href="#">TUIRoomEvents.onKickedOutOfRoom</a>	Kicked out of room event
<a href="#">TUIRoomEvents.onKickedOffLine</a>	Current user Kicked off line event
<a href="#">TUIRoomEvents.onUserSigExpired</a>	UserSig Expiration Event
<a href="#">TUIRoomEvents.onRoomDismissed</a>	Host Destroy Room Event
<a href="#">TUIRoomEvents.onRoomNameChanged</a>	Room Name Change Event
<a href="#">TUIRoomEvents.onRoomSpeechModeChanged</a>	Room Speech Mode Change Event
<a href="#">TUIRoomEvents.onAllUserCameraDisableChanged</a>	All members Camera Usage Permission Change Event
<a href="#">TUIRoomEvents.onAllUserMicrophoneDisableChanged</a>	All members Mic Usage Permission Change Event
<a href="#">TUIRoomEvents.onSendMessageForAllUserDisableChanged</a>	All members Send Message Status Change Event
<a href="#">TUIRoomEvents.onRoomMaxSeatCountChanged</a>	Room Maximum Seat Count Change Event
<a href="#">TUIRoomEvents.onRemoteUserEnterRoom</a>	Remote User Entered Room Event
<a href="#">TUIRoomEvents.onRemoteUserLeaveRoom</a>	Remote User Leave Room Event
<a href="#">TUIRoomEvents.onUserRoleChanged</a>	User Role Change Event
<a href="#">TUIRoomEvents.onUserVideoStateChanged</a>	User Video Status Change Event



<a href="#">TUIRoomEvents.onUserAudioStateChanged</a>	User Audio Status Change Event
<a href="#">TUIRoomEvents.onSendMessageForUserDisableChanged</a>	User Send Message Status Event
<a href="#">TUIRoomEvents.onUserVoiceVolumeChanged</a>	User Volume Change Event
<a href="#">TUIRoomEvents.onUserNetworkQualityChanged</a>	User Network Quality Change Event
<a href="#">TUIRoomEvents.onSeatListChanged</a>	Seat List Change Event
<a href="#">TUIRoomEvents.onKickedOffSeat</a>	User Kicked off Seat Event
<a href="#">TUIRoomEvents.onRequestReceived</a>	Request Received Event
<a href="#">TUIRoomEvents.onRequestCancelled</a>	Request Cancelled Event
<a href="#">TUIRoomEvents.onReceiveTextMessage</a>	Receive Text Message Event
<a href="#">TUIRoomEvents.onReceiveCustomMessage</a>	Receive Custom Message Event
<a href="#">TUIRoomEvents.onDeviceChange</a>	Device Change Event
<a href="#">TUIRoomEvents.onUserScreenCaptureStopped</a>	Screen Sharing Stopped Event When the user uses the built-in browser stop sharing button to end screen sharing, the user will receive the 'onUserScreenCaptureStopped' event to modify the screen sharing status.

## onError

Error Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onError, (error) => {
  console.log('TUIRoomError error', error);
})
```

The parameters are shown in the table below:

Parameter	Type	Meaning
code	number	Error Code

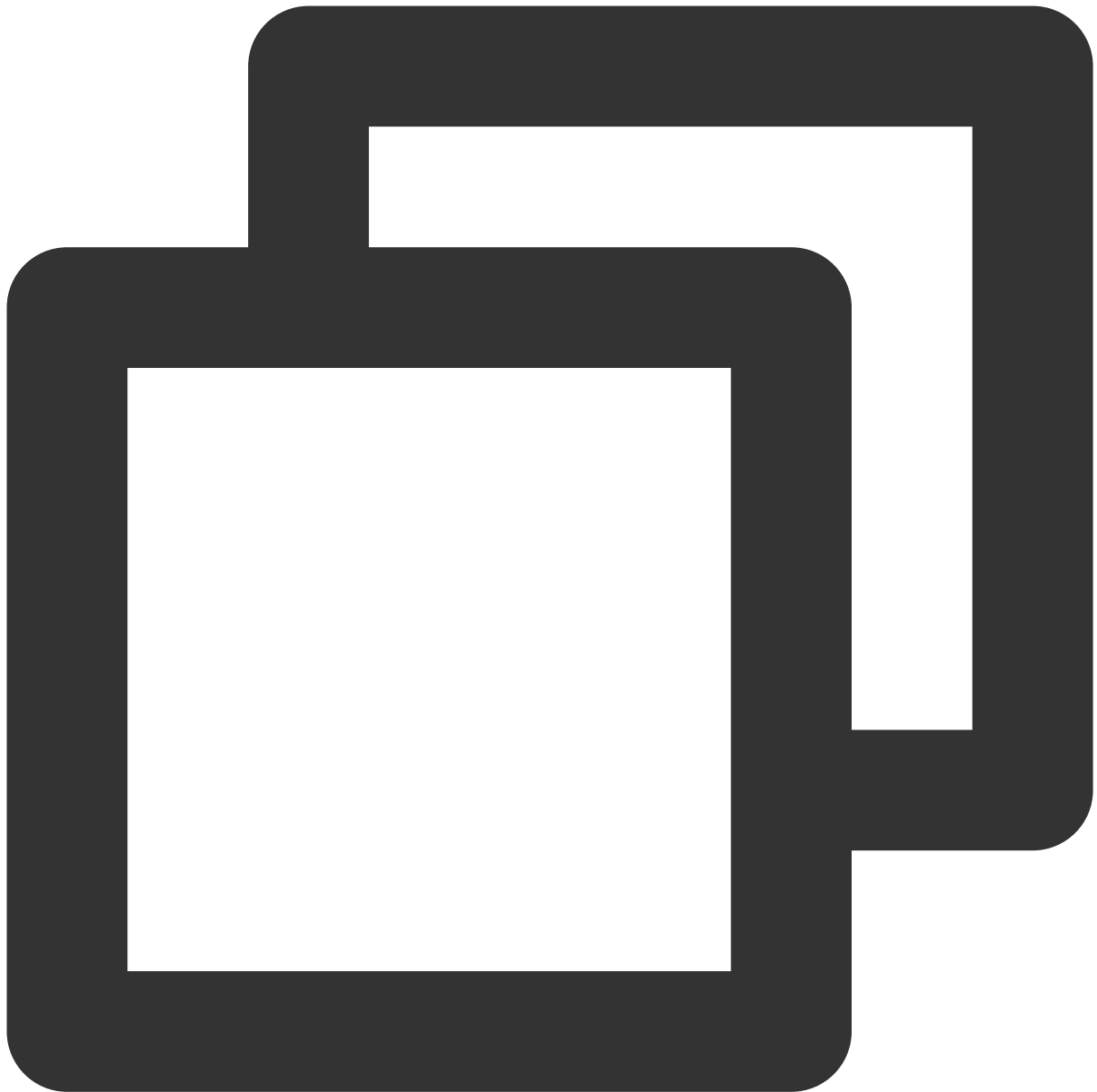
message

string

Error Information

## onKickedOutOfRoom

Kicked out of room event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onKickedOutOfRoom, ({ roomId, message }) => {
  console.log('roomEngine.onKickedOutOfRoom', roomId, message);
});
```

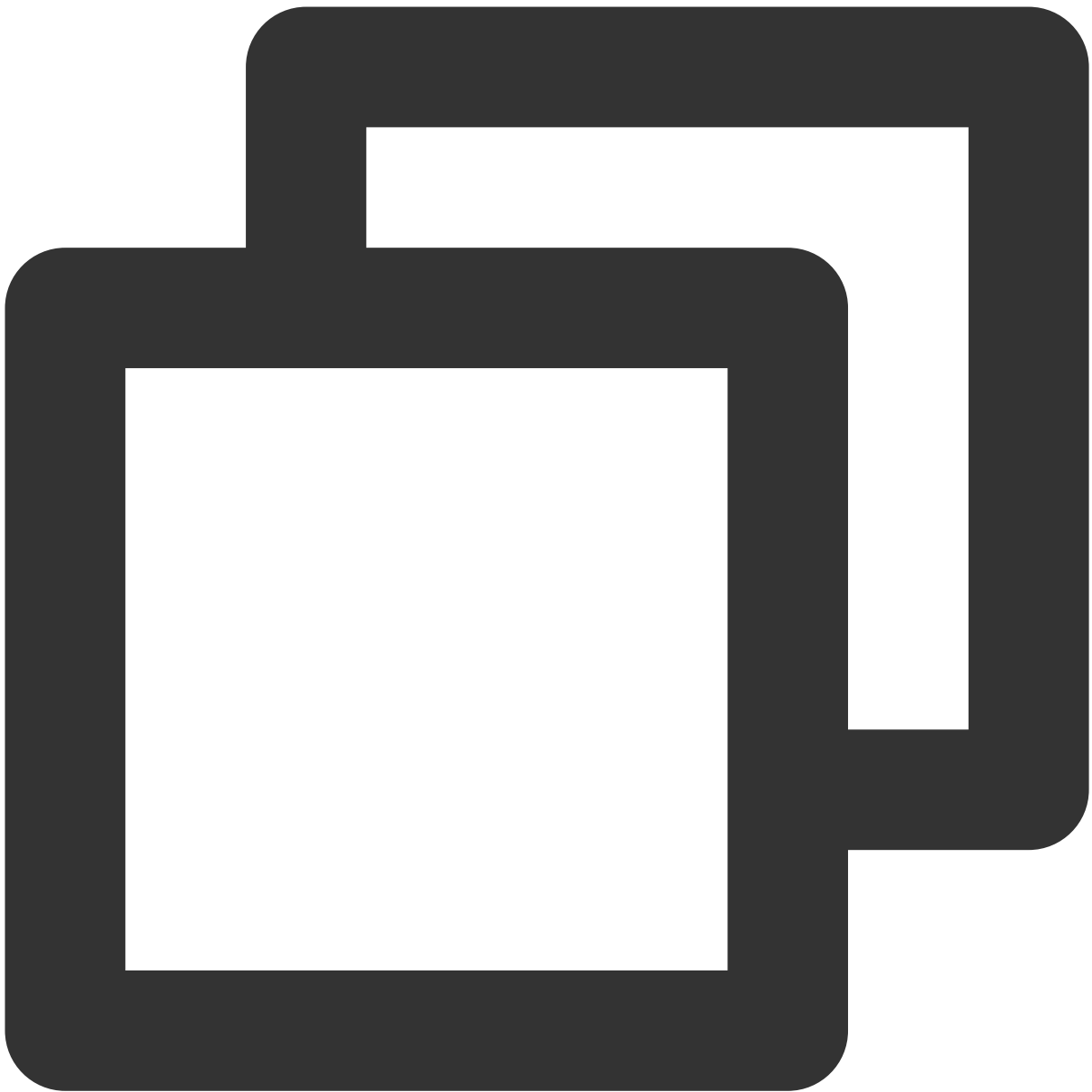
```
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
message	string	Kicked out of room information

## onKickedOffLine

Current user Kicked off line event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onKickedOffLine, ({ message }) => {
  console.log('roomEngine.onKickedOffLine', message);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID

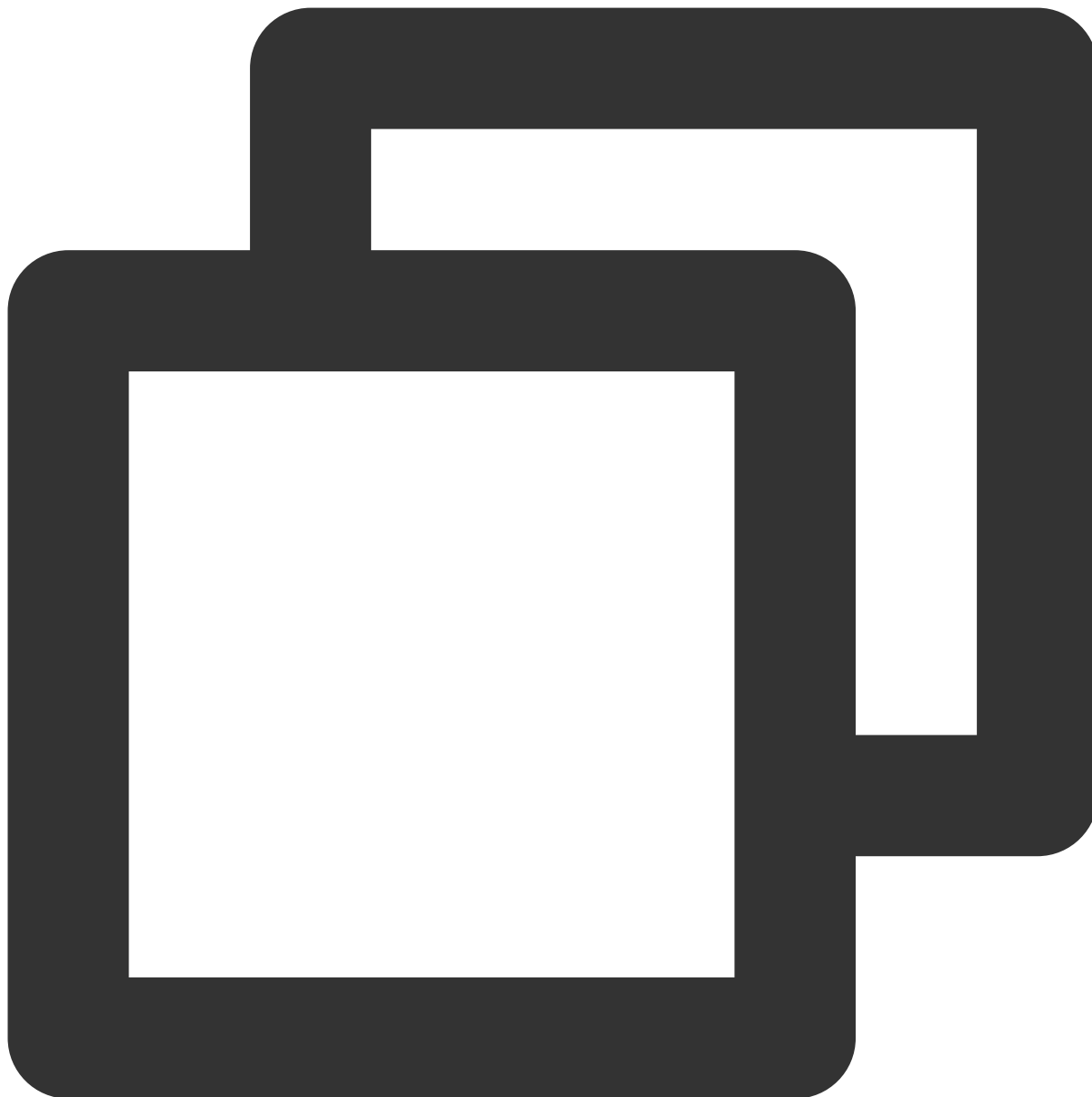
message

string

User logged in on other end information

## onUserSigExpired

UserSig Expiration Event

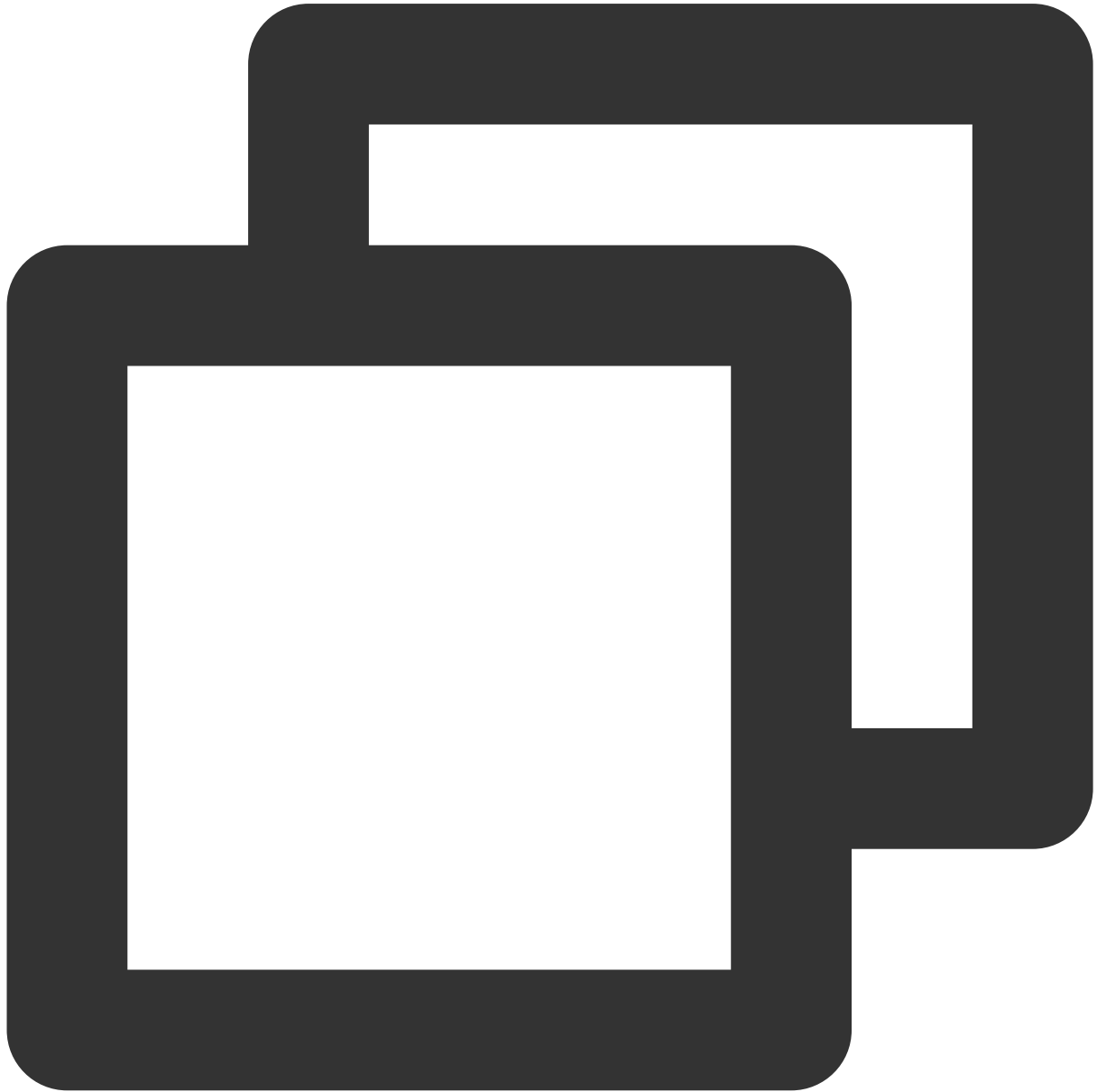


```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserSigExpired, () => {
  console.log('roomEngine.onUserSigExpired');
});
```

```
});
```

## onRoomDismissed

Host Destroy Room Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomDismissed, ({ roomId }) => {
  console.log('roomEngine.onRoomDismissed', roomId);
});
```

```
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID

## onRoomNameChanged

Room ID Modification Event





```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomNameChanged, ({ roomId, roomName }) => {
  console.log('roomEngine.onRoomNameChanged', roomId, roomName);
});
```

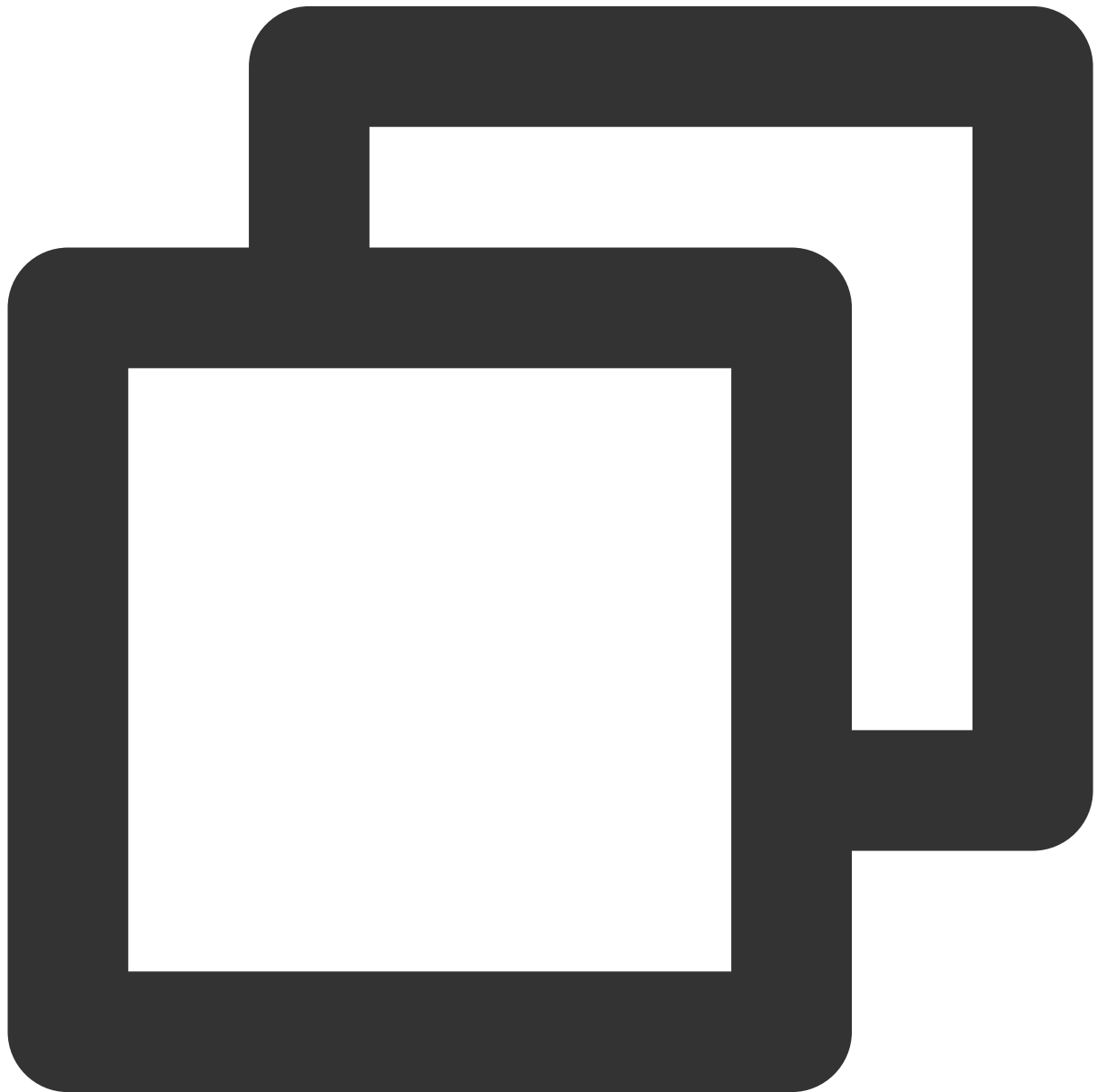
The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID

roomName	string	Room Name
----------	--------	-----------

## onRoomSpeechModeChanged

Room Name Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomSpeechModeChanged, ({ roomId, speechMode }) => {
  console.log('roomEngine.onRoomSpeechModeChanged', roomId, speechMode);
});
```

```
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
speechMode	<a href="#">TUISpeechMode</a>	Speech Mode

## onAllUserCameraDisableChanged

All members Camera Usage Permission Change Event



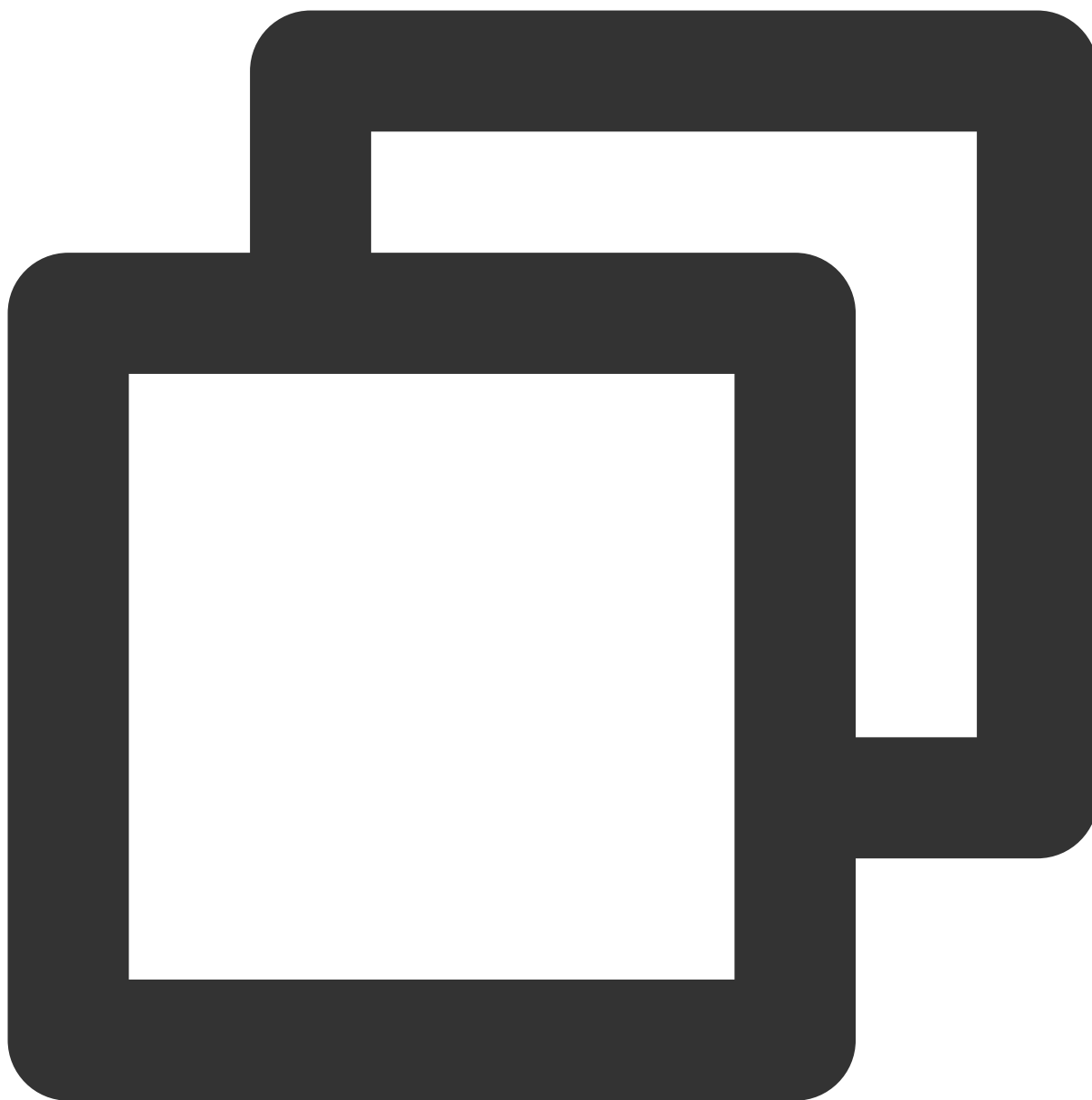
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onAllUserCameraDisableChanged, ({ isDisable }) => {
  console.log('roomEngine.onAllUserCameraDisableChanged', isDisable);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
isDisable	boolean	Allow Camera Usage

## onAllUserMicrophoneDisableChanged

All members Mic Usage Permission Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onAllUserMicrophoneDisableChanged, ({ isDisable }) => {
  console.log('roomEngine.onAllUserMicrophoneDisableChanged', isDisable);
});
```

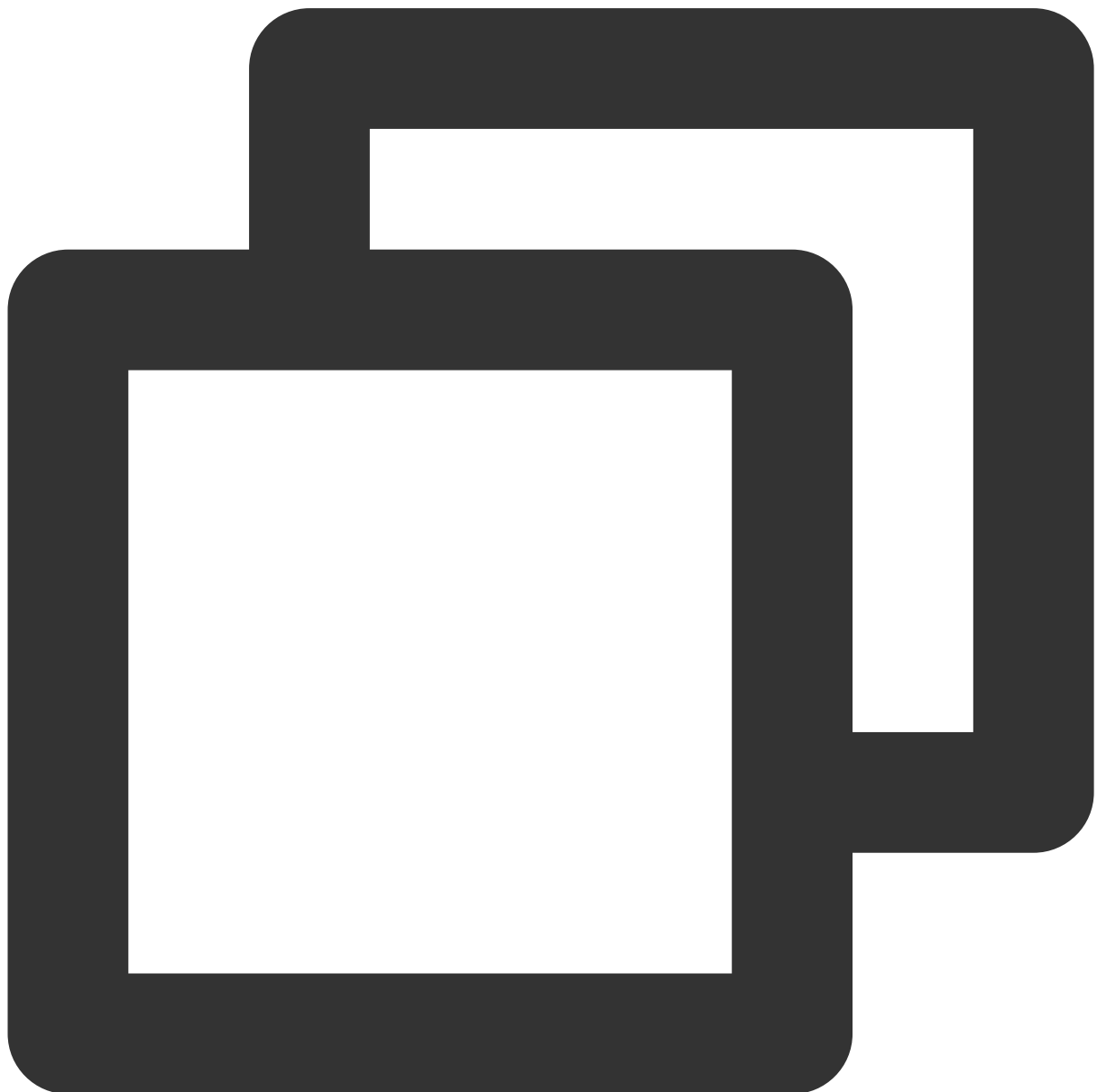
The parameters are shown in the table below:

--	--	--

Parameter	Type	Meaning
isDisable	boolean	Allow Mic Usage

## onSendMessageForAllUserDisableChanged

All members Send Message Permission Change Event



```
const roomEngine = new TUIRoomEngine();
```

```
roomEngine.on(TUIRoomEvents.onSendMessageForAllUserDisableChanged, ({ isDisable })  
    console.log('roomEngine.onSendMessageForAllUserDisableChanged', isDisable);  
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
isDisable	boolean	Allow Sending Text Message

## onRoomMaxSeatCountChanged

Room Maximum Seat Count Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomMaxSeatCountChanged, ({ maxSeatNumber }) => {
  console.log('roomEngine.onRoomMaxSeatCountChanged', maxSeatNumber);
});
```

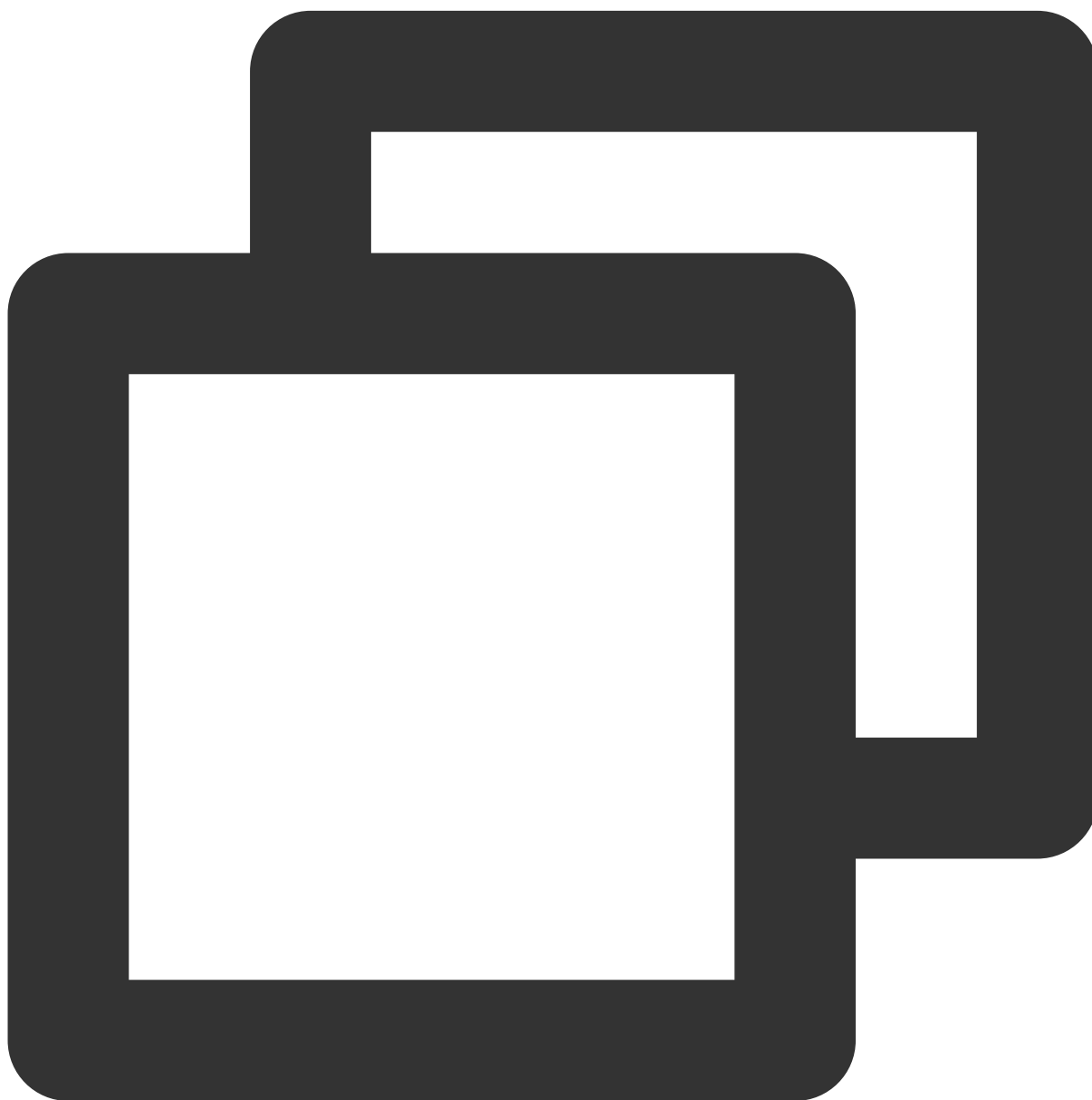
The parameters are shown in the table below:

Parameter	Type	Meaning
maxSeatNumber	number	Maximum Seat Count



## onRemoteUserEnterRoom

Remote User Entered Room Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRemoteUserEnterRoom, ({ roomId, userInfo }) => {
  console.log('roomEngine.onRemoteUserEnterRoom', roomId, userInfo);
});
```

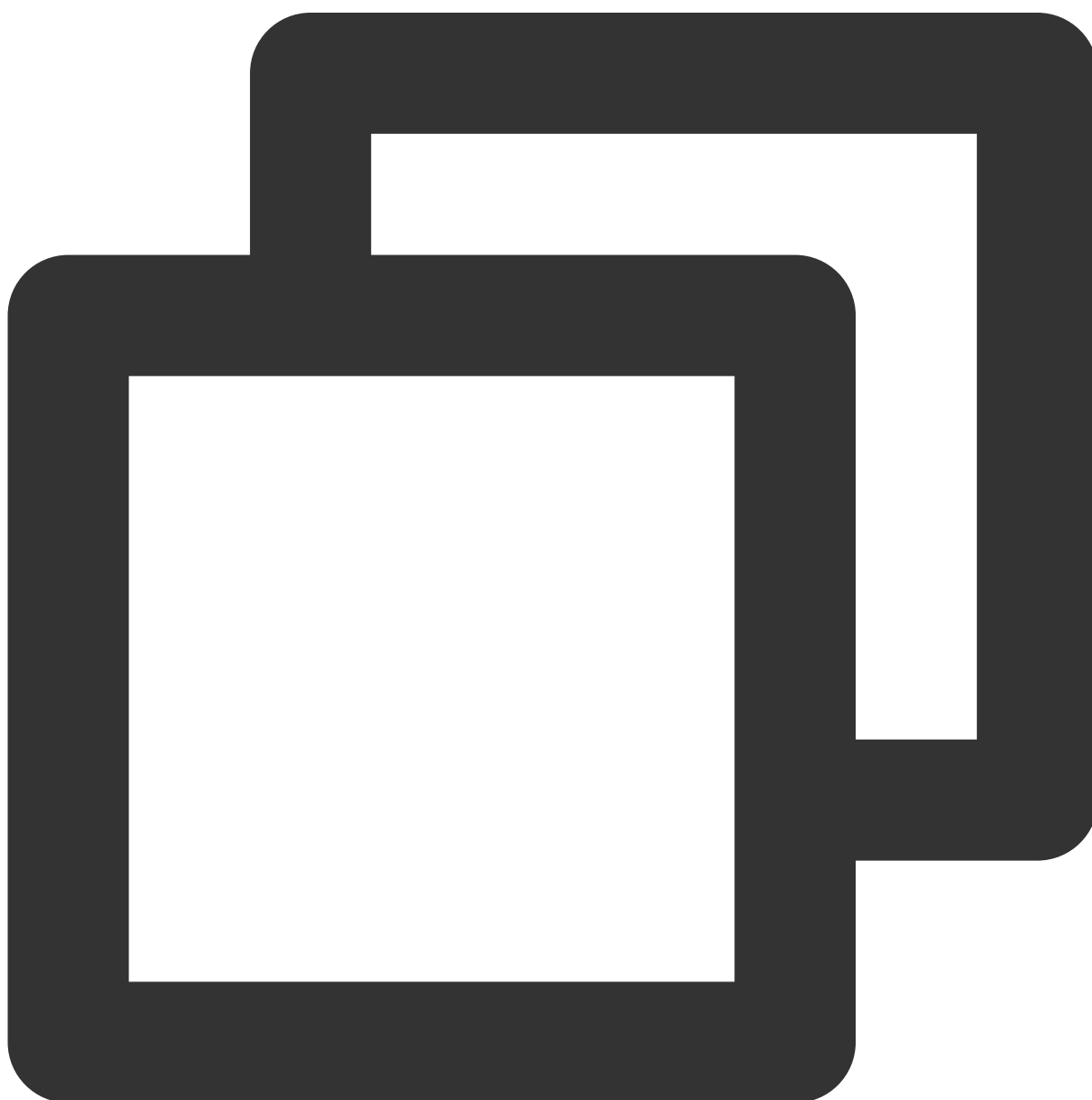
The parameters are shown in the table below:

--	--	--

Parameter	Type	Meaning
roomId	string	Room ID
userInfo	<a href="#">TUIUserInfo</a>	User Information

## onRemoteUserLeaveRoom

Remote User Leave Room Event



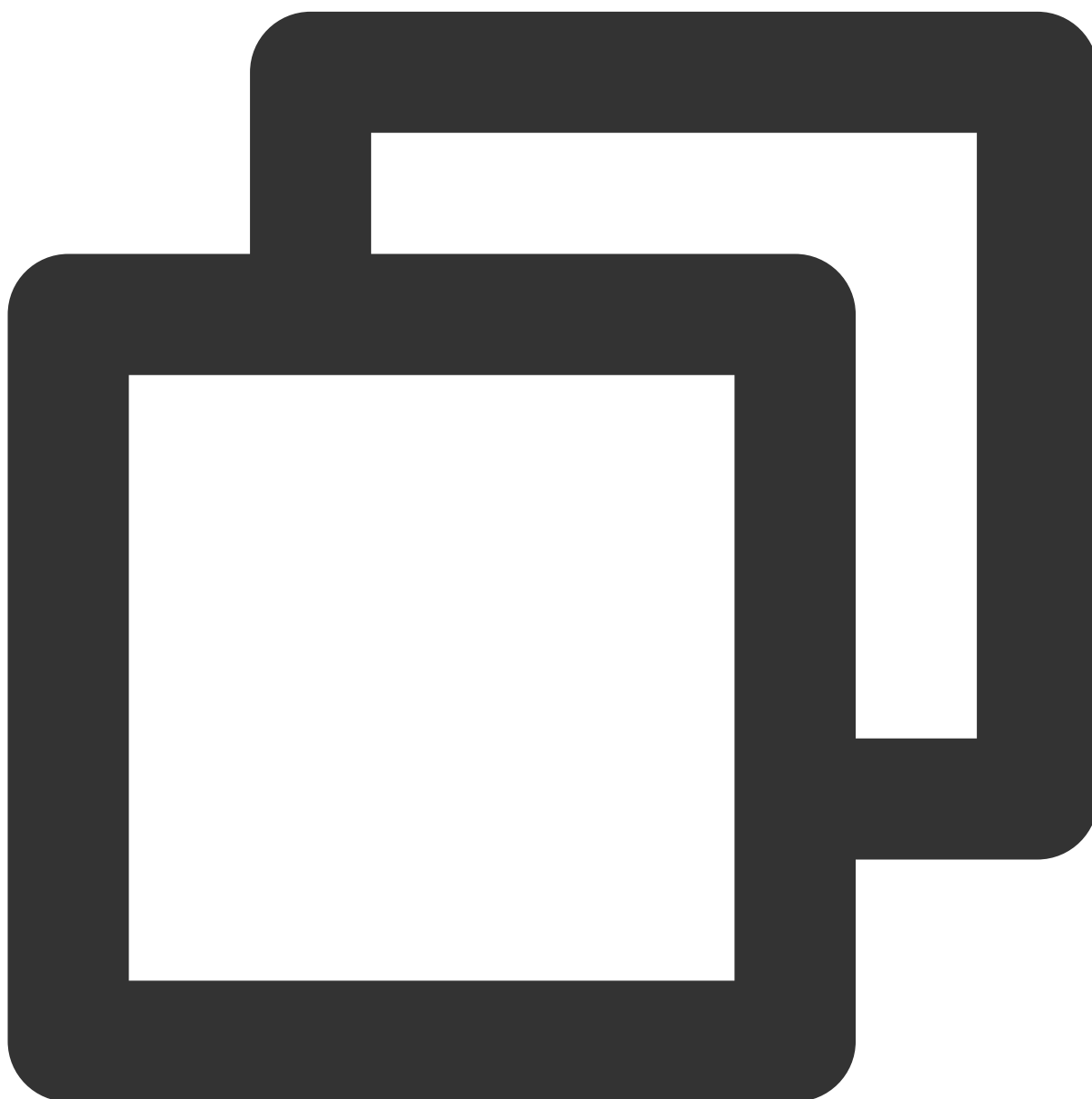
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRemoteUserLeaveRoom, ({ roomId, userInfo }) => {
  console.log('roomEngine.onRemoteUserLeaveRoom', roomId, userInfo);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
userInfo	<a href="#">TUIUserInfo</a>	User Information

## onUserRoleChanged

User Role Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserRoleChanged, ({ userId, userRole }) => {
  console.log('roomEngine.onUserRoleChanged', userId, userRole);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
userId	string	User Id

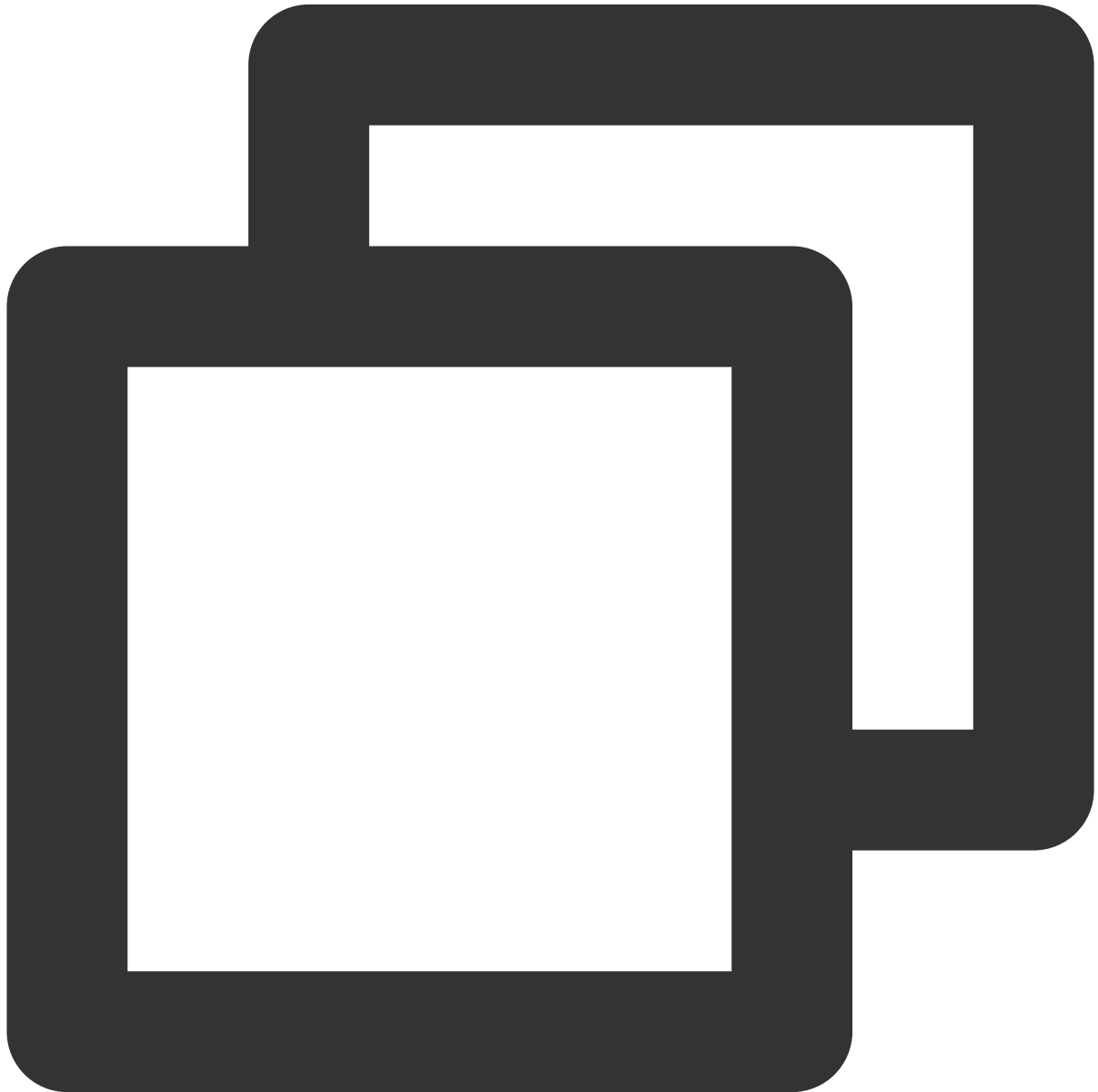
userRole

TUIRole

User Changed Role

## onUserVideoStateChanged

User Video Status Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserVideoStateChanged, ({ userId, streamType, hasVideo, ...data }) => {
  console.log('roomEngine.onUserVideoStateChanged', userId, streamType, hasVideo, ...data);
});
```

```
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
userId	string	User Id
streamType	<a href="#">TUIVideoStreamType</a>	User Stream Type
hasVideo	boolean	Has Video streams
reason	<a href="#">TUIChangeReason</a>	Change Reason, Self-operation/Host-operation

## onUserAudioStateChanged

User Audio Status Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserAudioStateChanged, ({ userId, hasAudio, reason })
  console.log('roomEngine.onUserAudioStateChanged', userId, hasAudio, reason);
});
```

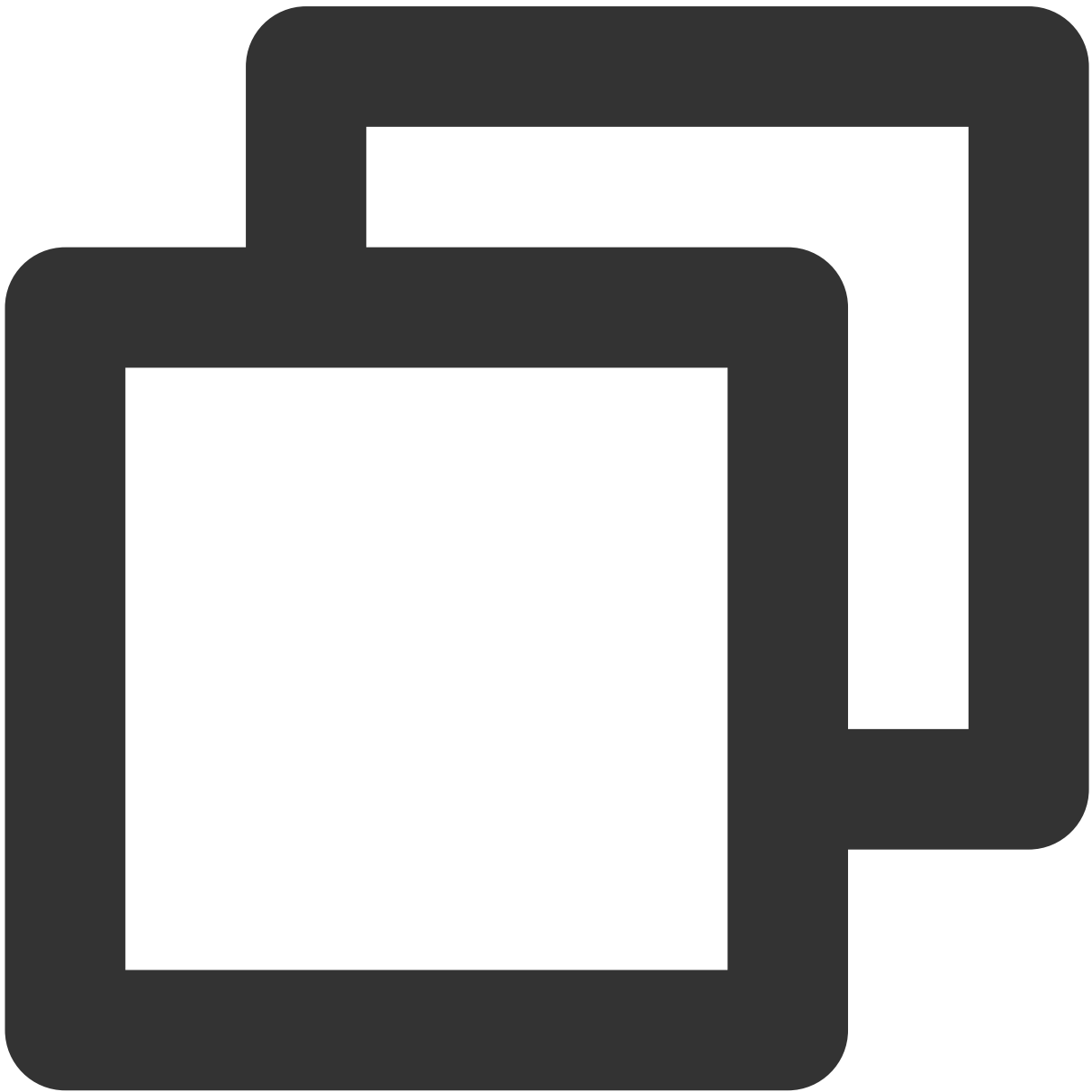
The parameters are shown in the table below:

Parameter	Type	Meaning
userId	string	User Id

hasVideo	boolean	Has Audio streams
reason	<a href="#">TUIChangeReason</a>	Change Reason, Self-operation/Host-operation

## onSendMessageForUserDisableChanged

Member Camera Usage Permission Change Event





```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onSendMessageForAllUserDisableChanged, ({ isDisable })
  console.log('roomEngine.onSendMessageForAllUserDisableChanged', isDisable);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
isDisable	boolean	Allow Sending Text Message

## onUserVoiceVolumeChanged

User Volume Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserVoiceVolumeChanged, ({ userVolumeList }) => {
  userVolumeList.forEach(userVolume => {
    console.log('roomEngine.onUserVoiceVolumeChanged', userVolume.userId, userVolume);
  })
});
```

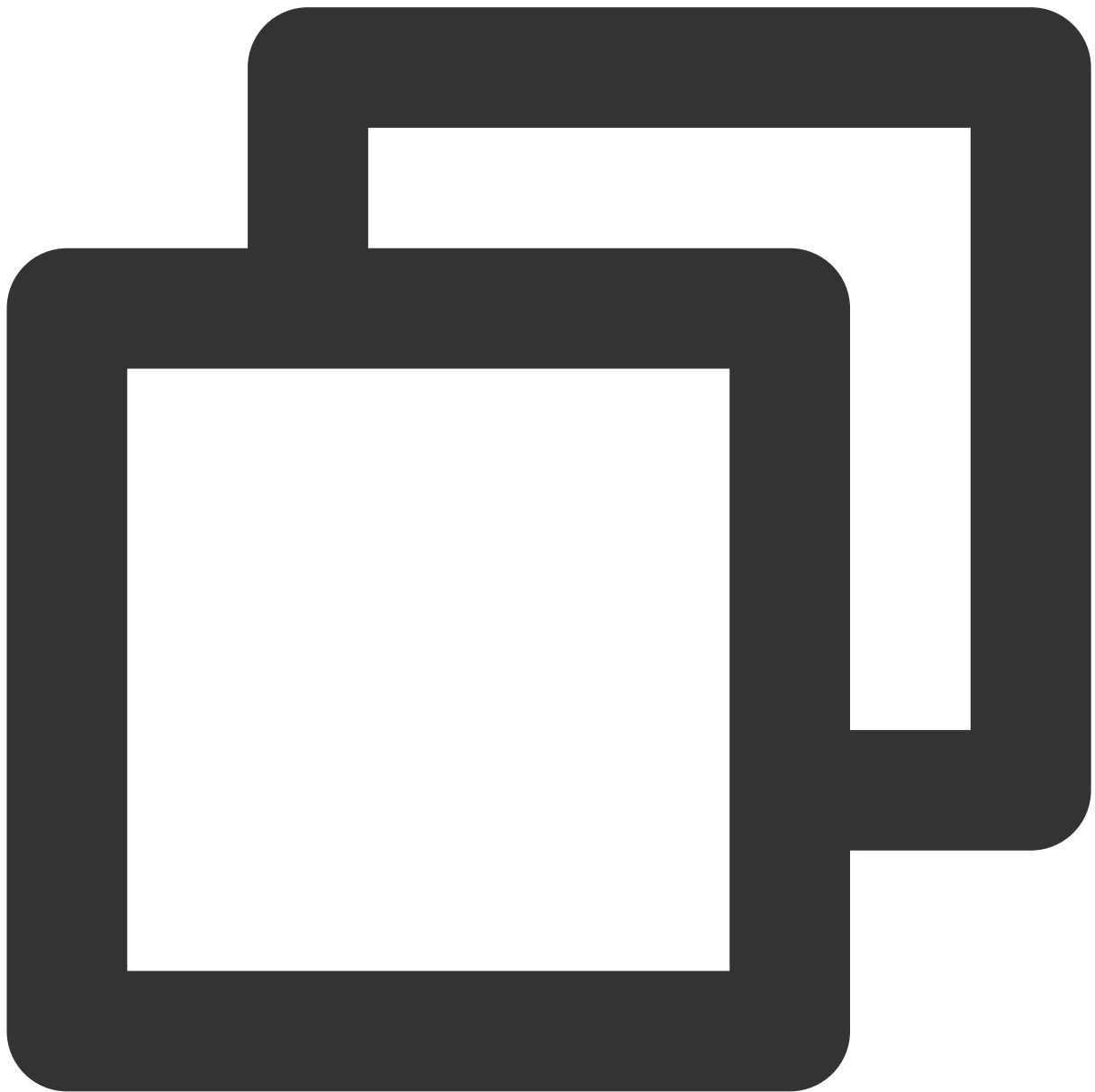
The parameters are shown in the table below:

Parameter	Type	Meaning

userVolumeList	Array.<object>	Volume of all users in the room, including userId and volume information, volume range is 1-100
----------------	----------------	---

## onUserNetworkQualityChanged

User Network Quality Change Event



```
const roomEngine = new TUIRoomEngine();
```

```
roomEngine.on(TUIRoomEvents.onUserNetworkQualityChanged, ({ userNetworkList }) => {
  userNetworkList.forEach(userNetwork => {
    console.log('roomEngine.onUserNetworkQualityChanged', userNetwork.userId, userN
  })
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
networkMap	<a href="#">TUINetworkQuality</a>	Traverse Network Quality Level

## onSeatListChanged

Seat List Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onSeatListChanged, ({ seatList, seatedList, leftList })
  console.log('roomEngine.onSeatListChanged',seatList, seatedList, leftList);
});
```

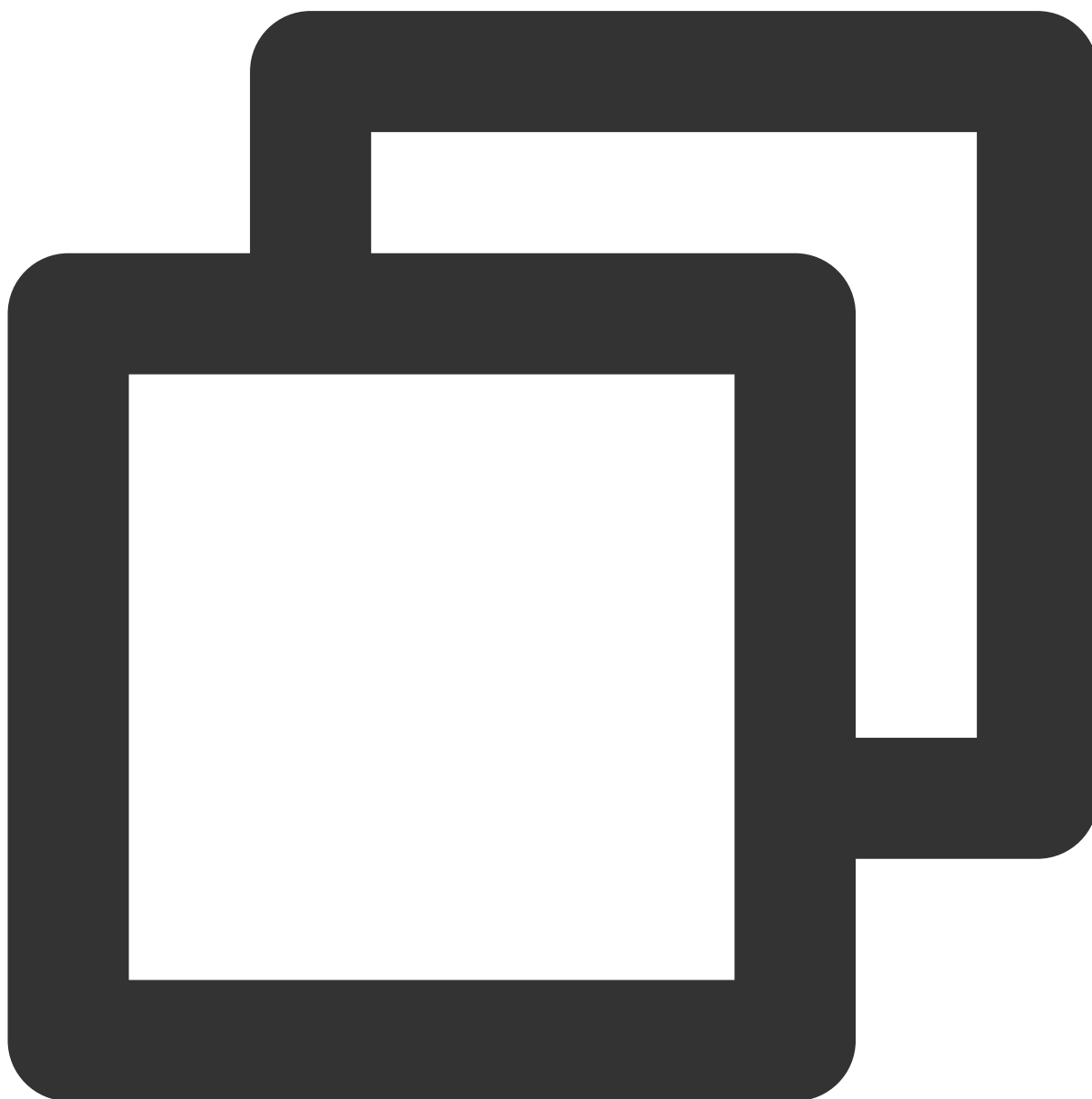
The parameters are shown in the table below:

Parameter	Type	Meaning
seatList	Array.< <a href="#">TUISeatInfo</a> >	Seat List

seatedList	Array.<TUISeatInfo>	New Seat Information
leftList	Array.<TUISeatInfo>	Left Seat Information

## onKickedOffSeat

Seat List Change Event



```
const roomEngine = new TUIRoomEngine();
```

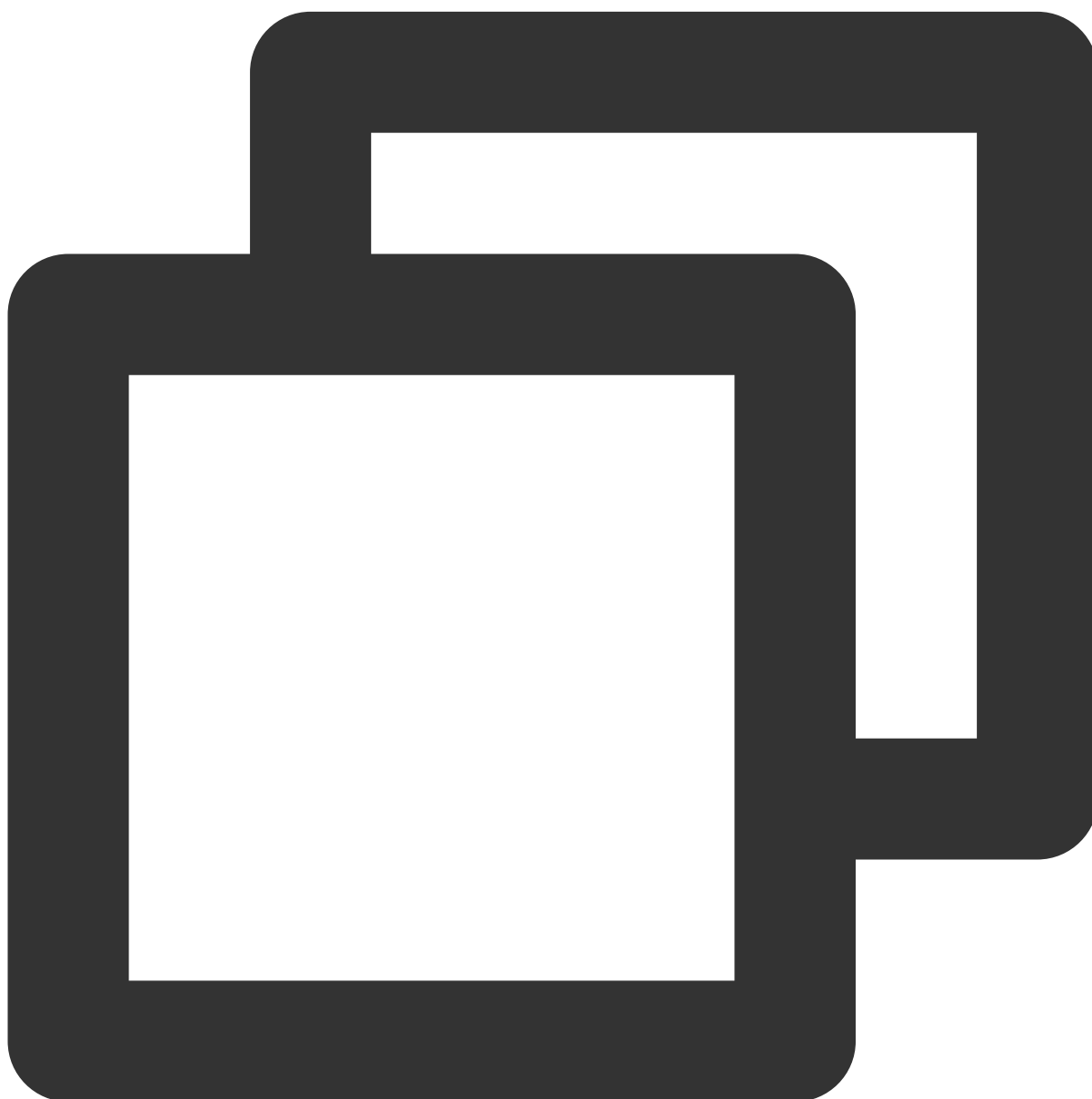
```
roomEngine.on(TUIRoomEvents.onKickedOffSeat, ({ userId }) => {  
    console.log('roomEngine.onKickedOffSeat', userId);  
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
userId	String	Kicked off Seat User Id

## onRequestReceived

Error Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRequestReceived, ({ request }) => {
  console.log('roomEngine.onRequestReceived', request);
});
```

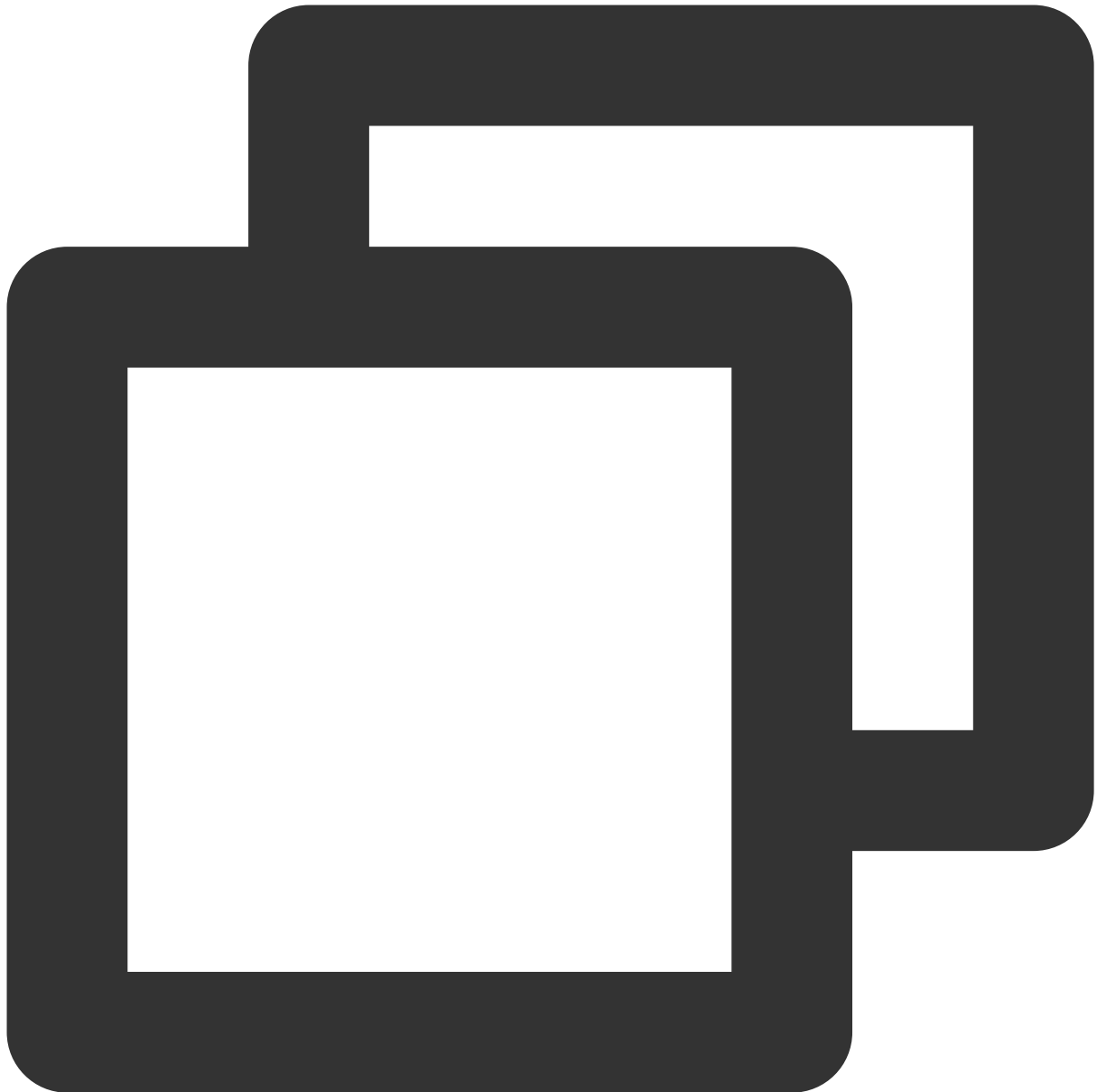
The parameters are shown in the table below:

Parameter	Type	Meaning
request	<a href="#">TUIRequest</a>	Request Received



# onRequestCancelled

Request Cancelled Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRequestCancelled, ({ requestId, userId }) => {
  console.log('roomEngine.onRequestCancelled', requestId, userId);
});
```

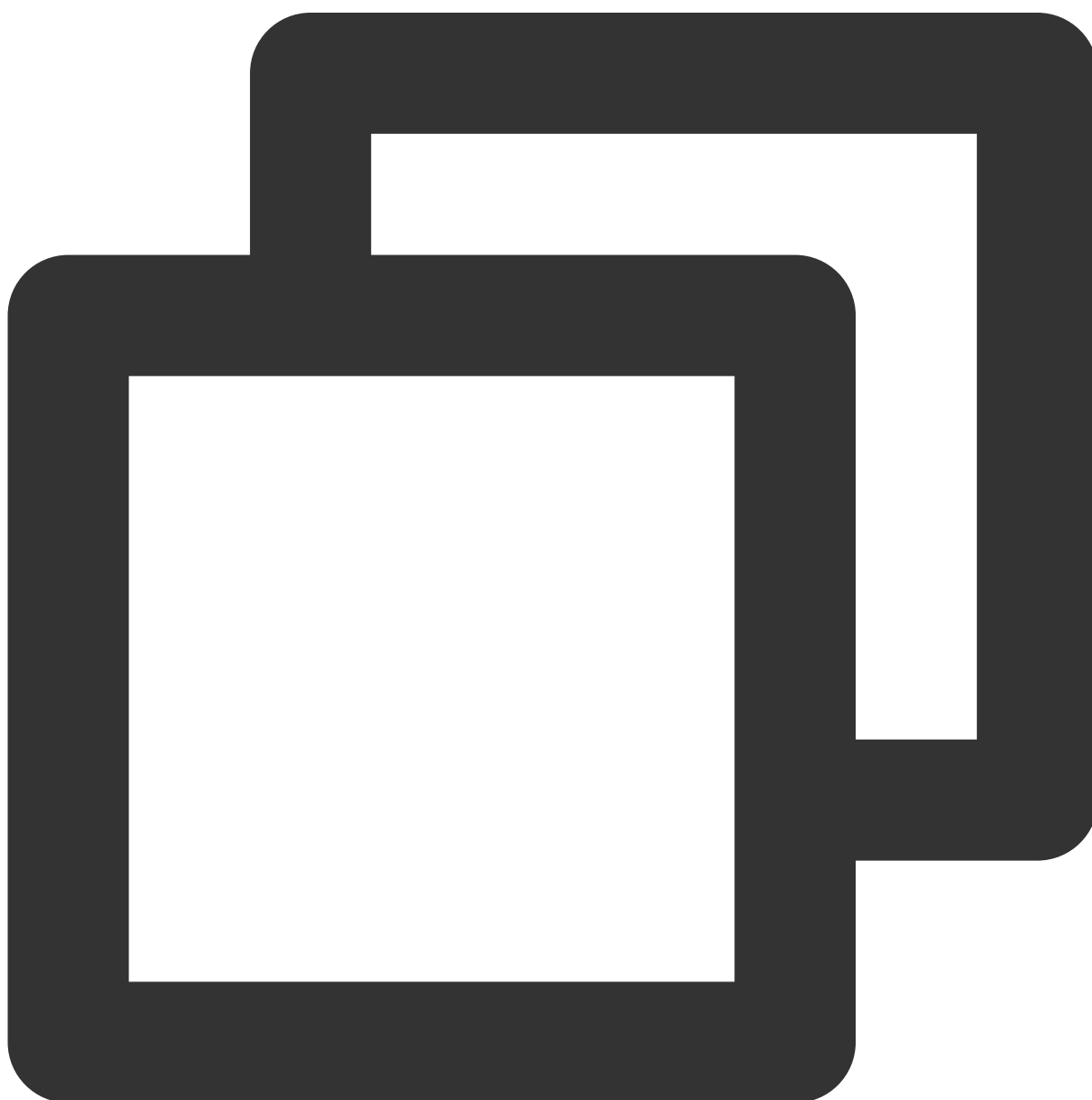
The parameters are shown in the table below:

--	--	--

Parameter	Type	Meaning
requestId	string	Request Id
userId	string	User Id of Cancelled Request

## onReceiveTextMessage

Receive Text Message Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onReceiveTextMessage, ({ roomId, message }) => {
  console.log('roomEngine.onReceiveTextMessage', roomId, message);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room Id
message	<a href="#">TUIMessage</a>	Receive Text Message

## onReceiveCustomMessage

Receive Custom Message Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onReceiveCustomMessage, ({ roomId, message }) => {
  console.log('roomEngine.onReceiveCustomMessage', roomId, message);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room Id

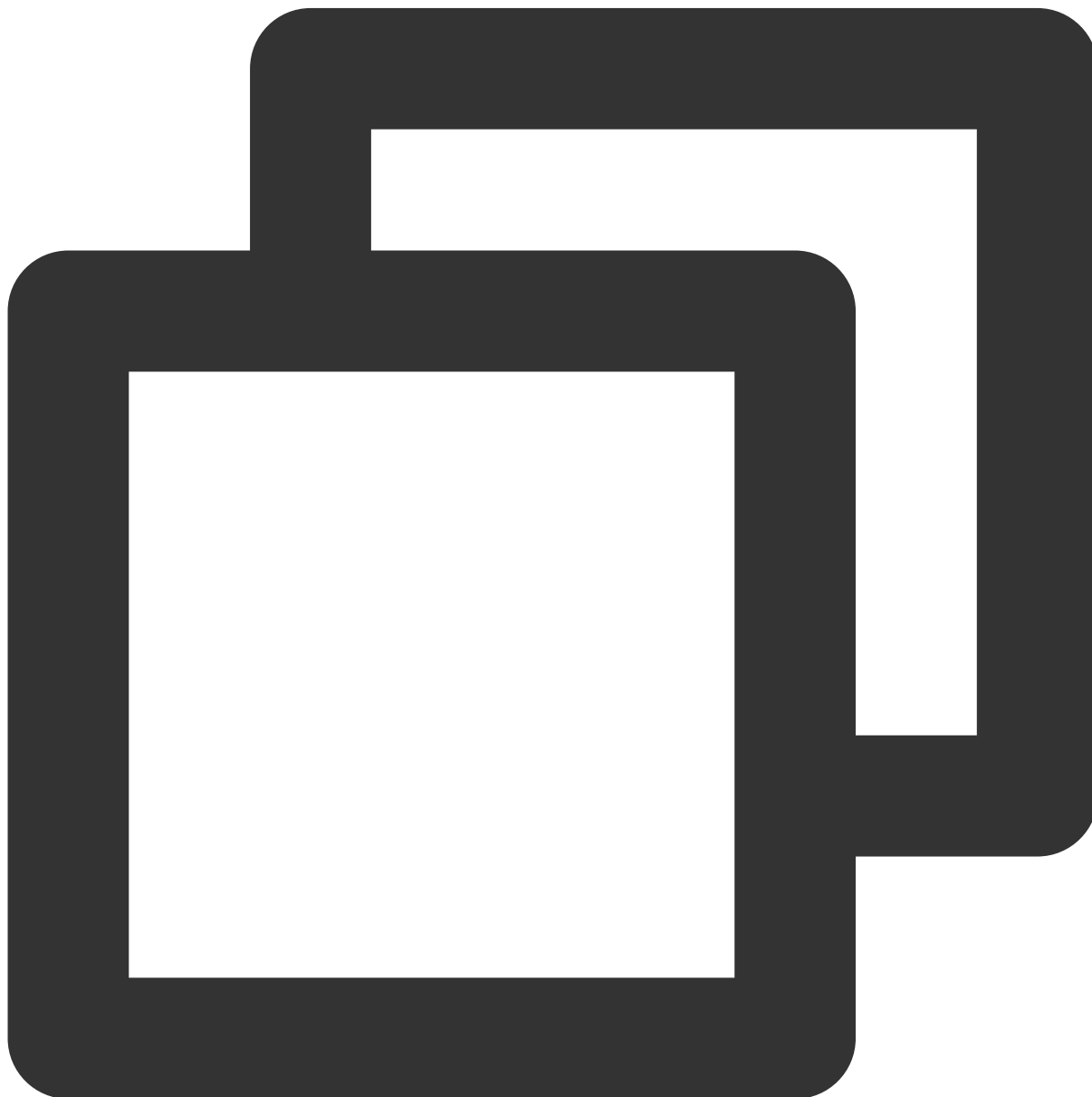
message

TUIMessage

Receive Text Message

## onDeviceChange

Device Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onDeviceChange, ({ deviceId, type, state }) => {
  console.log('roomEngine.onDeviceChange', deviceId, type, state);
});
```

```
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
deviceId	string	Device Id
type	<a href="#">TRTCDeviceType</a>	Device Type
state	<a href="#">TRTCDeviceState</a>	Device Change Status

## onUserScreenCaptureStopped

Screen Sharing Stopped Event, when the user uses the built-in browser stop sharing button to end screen sharing, the user will receive the ' `onUserScreenCaptureStopped` ' event to modify the screen sharing status.



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserScreenCaptureStopped, () => {
  console.log('roomEngine.onUserScreenCaptureStopped', deviceId, type, state);
});
```

# TUIRoomEngine Defines

Last updated : 2023-11-14 17:05:09

Introduction to Key Type Definition of TUIRoomEngine web side.

## Enumeration Value

### TUIRole

User Role, TUIRoomEngine provides three user roles, which are Host, Administrator, and Regular User.

Field	Type	Description
kRoomOwner	number	Host Role
kAdministrator	number	Administrator Role
kGeneralUser	number	Regular User Role

### TUIVideoProfile

Video Resolution

Field	Type	Description
kLowDefinition	number	Low Resolution
kStandardDefinition	number	SD
kHighDefinition	number	HD
kSuperDefinition	number	Ultra HD

### TUIAudioProfile

Audio Resolution

Field	Type	Description
kAudioProfileSpeech	number	Voice Mode
kAudioProfileDefault	number	Standard Mode (Default Mode)
kAudioProfileMusic	number	Music Mode



## TUIVideoStreamType

### Streams Type

Field	Type	Description
kCameraStream	number	Camera Streams
kScreenStream	number	Screen Sharing Streams
kCameraStreamLow	number	Low Resolution Camera Streams

## TUINetworkQuality

### Network Status

Field	Type	Description
kQualityUnknown	number	Network Condition Unknown
kQualityExcellent	number	Network Condition Excellent
kQualityGood	number	Network Condition Good
kQualityPoor	number	Network Condition Fair
kQualityBad	number	Network Condition Poor
kQualityVeryBad	number	Network Condition Very Poor
kQualityDown	number	Network Connection Disconnected

## TUIRoomType

### Room Type

Field	Type	Description
kGroup	number	Group Type Room, suitable for conference and educational scene, the microphone position in this room is unordered and has no quantity limit
kOpen	number	Open Type Room, suitable for live streaming scene, the microphone position in this room is ordered and has a quantity limit

## TUISpeechMode

### Speech Type

Field	Type	Description
kFreeToSpeak	number	Free Speech Mode
kApplyToSpeak	number	Hand-raising Speech Mode
kSpeakAfterTakingSeat	number	Speak After Sitting (Grab Microphone Position)

## TUICaptureSourceType

Screen Sharing Type

Field	Type	Description
kWindow	number	Sharing Target is a specific Windows or Mac window todo (only for electron)
kScreen	number	Sharing Target is the entire Windows desktop or Mac desktop

## TUIChangeReason

Change Reason (Audio and Video Status Change Operation Reason: Self-initiated modification or modified by room owner/administrator)

Field	Type	Description
kChangedBySelf	number	Self-operation
kChangedByAdmin	number	Room Owner or Administrator Operation

## TUIRequestAction

Room Type

Field	Type	Description
kInvalidAction	number	Invalid Operation
kRequestToOpenRemoteCamera	number	Request Remote Camera On
kRequestToOpenRemoteMicrophone	number	Request Remote Mic On
kRequestToConnectOtherRoom	number	Request Remote Cross-room Streaming, web side temporarily unsupported
kRequestToTakeSeat	number	Request Go Live

kRequestRemoteUserOnSeat	number	Request Remote Go Live
--------------------------	--------	------------------------

## TUIRequestCallbackType

Request Type

Field	Type	Description
kRequestAccepted	number	Peer Accepted
kRequestRejected	number	Peer Rejected
kRequestCancelled	number	Request Canceled
kRequestTimeout	number	Request Timeout
kRequestError	number	Request Error

## Type Definition

### TUIRoomInfo

Room data, user can use `roomEngine.getRoomInfo` to get room data.

Field	Type	Description
roomId	string	Room Number, String Type Room Number
roomType	<a href="#">TUIRoomType</a>	Room Type
owner	string	Host's userId
name	string	Room ID
createTime	string	Creation time
roomMemberCount	number	Current total number of people in the room
maxSeatCount	number	Maximum number of microphone positions in the room
enableVideo	boolean	Allow users to join and turn on Audio
enableAudio	boolean	Allow users to join and turn on Video
enableMessage	boolean	Allow users to join and send messages
enableSeatControl	boolean	Enable microphone position control

## TUIUserInfo

### User Information

Field	Type	Description
userId	string	User Id
userName	string	User Name
avatarUrl	string	User Avatar
userRole	<a href="#">TUIRole</a>	User Role
hasAudioStream	boolean	Whether there are Audio streams
hasVideoStream	boolean	Whether there are Video streams
hasScreenStream	boolean	Whether there is Screen Sharing stream

## TUIMessage

### Message Information

Field	Type	Description
messageId	string	Message Id
message	string	Message
timestamp	number	Timestamp information, accurate to seconds
userId	string	User Id
userName	string	User name
avatarUrl	string	User Avatar

## TUIRequest

### Request Information

Field	Type	Description
requestAction	<a href="#">TUIRequestAction</a>	Request Type
timestamp	number	Request Initiation Time
requestId	string	Request Id v1.0.2 and above versions requestId type is string;

		v1.0.0 and v1.0.1 versions requestId type is number;
userId	string	Initiate Request's User Id
content	string	Other Content

## TUIRequestCallback

### Request Callback Information

Field	Type	Description
requestCallbackType	<a href="#">TUIRequestCallbackType</a>	Request Callback Type, Accept/Reject/Cancel/Timeout/Error
requestId	string	Request Id v1.0.2 and above versions requestId type is string; v1.0.0 and v1.0.1 versions requestId type is number;
userId	string	User Id
code	number	Request Response Code
message	string	Request Status Supplemental Description

## TUISeatInfo

### Microphone Position Information

Field	Type	Description
index	number	Microphone Position Number
userId	string	Microphone Position Correspondence's User Id
locked	boolean	Whether the current microphone position is locked
videoMuted	boolean	Whether the current microphone position prohibits Video
audioMuted	boolean	Whether the current microphone position prohibits Audio

# Electron

## API Overview

Last updated : 2024-02-29 10:57:30

### TUIRoomEngine (No UI interface)

#### TUIRoomEngine Static method

API	Description
<a href="#">once</a>	<b>Listen to TUIRoomEngine ready event.</b> <b>Note: All methods except TUIRoomEngine.init must be executed after listening to the TUIRoomEngine ready event and the TUIRoomEngine.init method is executed successfully.</b>
<a href="#">login</a>	Login to TUIRoomEngine
<a href="#">setSelfInfo</a>	Set the current user's basic information (username, user avatar)
<a href="#">getSelfInfo</a>	Get the current user's basic information (username, user avatar)
<a href="#">logout</a>	Logout from TUIRoomEngine

#### RoomEngine Room Management API

API	Description
<a href="#">createRoom</a>	Create room
<a href="#">enterRoom</a>	Entered room
<a href="#">destroyRoom</a>	Close the room
<a href="#">exitRoom</a>	Leave room
<a href="#">fetchRoomInfo</a>	Get room data
<a href="#">updateRoomNameByAdmin</a>	Update the room's name (Only group owner or administrator can call)
<a href="#">updateRoomSpeechModeByAdmin</a>	Update the room's speech mode (Only group owner or administrator can call)
<a href="#">getUserList</a>	Get the current room user list

[getUserInfo](#)[Get user](#) [Learn more](#)**roomEngine Audio Video API**

API	Description
<a href="#">setLocalVideoView</a>	Set the view control for local user video rendering
<a href="#">openLocalCamera</a>	Open local camera
<a href="#">closeLocalCamera</a>	Close local camera
<a href="#">openLocalMicrophone</a>	Open local microphone
<a href="#">closeLocalMicrophone</a>	Close local microphone
<a href="#">updateVideoQuality</a>	Update local video codec quality settings
<a href="#">updateAudioQuality</a>	Update local audio codec quality settings
<a href="#">startPushLocalVideo</a>	Start pushing local video
<a href="#">stopPushLocalVideo</a>	Stop pushing local video
<a href="#">startPushLocalAudio</a>	Start pushing local audio
<a href="#">stopPushLocalAudio</a>	Stop pushing local audio
<a href="#">setRemoteVideoView</a>	Set the view control for remote user video rendering
<a href="#">startPlayRemoteVideo</a>	Start playing remote user video
<a href="#">stopPlayRemoteVideo</a>	Stop playing remote user video
<a href="#">muteRemoteAudioStream</a>	Mute remote user

**roomEngine Member Management API**

API	Description
<a href="#">openRemoteDeviceByAdmin</a>	Request remote user to open media device
<a href="#">applyToAdminToOpenLocalDevice</a>	Participant applies to the host to open the device
<a href="#">closeRemoteDeviceByAdmin</a>	Close remote user's media device
<a href="#">cancelRequest</a>	Cancel the sent request

<a href="#">responseRemoteRequest</a>	Reply to remote user's request
<a href="#">changeUserRole</a>	Change user's role
<a href="#">kickRemoteUserOutOfRoom</a>	Kick user out of the room
<a href="#">disableDeviceForAllUserByAdmin</a>	Disable/Enable all users' media devices
<a href="#">disableSendingMessageForAllUser</a>	Disable/Enable all users to send messages
<a href="#">disableSendingMessageByAdmin</a>	Disable/Enable a user to send messages

### roomEngine Screen Sharing API

API	Description
<a href="#">startScreenSharingElectron</a>	Start screen sharing
<a href="#">stopScreenSharingElectron</a>	End screen sharing
<a href="#">getScreenSharingTarget</a>	Get Screen Sharing List
<a href="#">selectScreenSharingTarget</a>	Switch Screen Sharing Window

### roomEngine Microphone Management API

API	Description
<a href="#">setMaxSeatCount</a>	Set Room Microphone Maximum
<a href="#">getSeatList</a>	Get Microphone Information
<a href="#">takeSeat</a>	Get Microphone
<a href="#">leaveSeat</a>	Release Microphone
<a href="#">takeUserOnSeatByAdmin</a>	Invite Others to Go Live (Only the Host and Administrator can call this method)
<a href="#">kickUserOffSeatByAdmin</a>	Kick Off Microphone (Only the Host and Administrator can call this method)
<a href="#">lockSeatByAdmin</a>	Lock a Microphone Status (Only the Host and Administrator can call this method)

### roomEngine Message Sending API

API	Description
<a href="#">sendTextMessage</a>	Send text message



[sendCustomMessage](#)

Send custom message

**roomEngine Device Management API**

API	Description
<a href="#">getCameraDevicesList</a>	Get camera device list
<a href="#">getMicDevicesList</a>	Get mic device list
<a href="#">getSpeakerDevicesList</a>	Get speaker device list
<a href="#">setCurrentCameraDevice</a>	Set the camera device to use
<a href="#">setCurrentMicDevice</a>	Set the mic device to use
<a href="#">setCurrentSpeakerDevice</a>	Set the speaker device to use
<a href="#">getCurrentCameraDevice</a>	Get the currently used camera device
<a href="#">getCurrentMicDevice</a>	Get the currently used mic device
<a href="#">getCurrentSpeakerDevice</a>	Get the currently used speaker device
<a href="#">startCameraDeviceTest</a>	Start camera device test
<a href="#">stopCameraDeviceTest</a>	Stop camera device test

**roomEngine Event Listening API**

API	Description
<a href="#">on</a>	Listen to <a href="#">TUIRoomEvents</a> event
<a href="#">off</a>	Cancel listening to <a href="#">TUIRoomEvents</a> event

**roomEngine Other API**

API	Description
<a href="#">getTRTCCloud</a>	Get trtcCloud instance
<a href="#">getTIM</a>	Get tim instance

## Event Type Definition

TUIRoomEvent is the Callback Event class corresponding to TUIRoomEngine. You can listen to Callback Events of interest through this callback.

EVENT	Description
<a href="#">TUIRoomEvents.onError</a>	Error Event
<a href="#">TUIRoomEvents.onKickedOutOfRoom</a>	Kick out of room event
<a href="#">TUIRoomEvents.onKickedOffline</a>	User Kicked Offline Event
<a href="#">TUIRoomEvents.onUserSigExpired</a>	User Credential Timeout Event
<a href="#">TUIRoomEvents.onRoomDismissed</a>	Room Dismissed Event
<a href="#">TUIRoomEvents.onRoomNameChanged</a>	Room Name Change Event
<a href="#">TUIRoomEvents.onRoomSpeechModeChanged</a>	Room Microphone Control Mode Change
<a href="#">TUIRoomEvents.onAllUserCameraDisableChanged</a>	All Users' Cameras Disabled in Room Event
<a href="#">TUIRoomEvents.onAllUserMicrophoneDisableChanged</a>	All Users' Microphones Disabled in Room Event
<a href="#">TUIRoomEvents.onSendMessageForAllUserDisableChanged</a>	All Users' Text Message Sending Disabled in Room Event
<a href="#">TUIRoomEvents.onRoomMaxSeatCountChanged</a>	Maximum number of microphone slots in the room modification event
<a href="#">TUIRoomEvents.onRemoteUserEnterRoom</a>	Remote user Entered room event
<a href="#">TUIRoomEvents.onRemoteUserLeaveRoom</a>	Remote user leaving room event
<a href="#">TUIRoomEvents.onUserRoleChanged</a>	Role change event
<a href="#">TUIRoomEvents.onUserVideoStateChanged</a>	Video Status change event
<a href="#">TUIRoomEvents.onUserAudioStateChanged</a>	Audio Status change event
<a href="#">TUIRoomEvents.onSendMessageForUserDisableChanged</a>	Send message status event
<a href="#">TUIRoomEvents.onUserVoiceVolumeChanged</a>	Volume change event
<a href="#">TUIRoomEvents.onUserNetworkQualityChanged</a>	Network quality change event
<a href="#">TUIRoomEvents.onSeatListChanged</a>	Mic position list change event
<a href="#">TUIRoomEvents.onKickedOffSeat</a>	Kicked off the mic event

<a href="#">TUIRoomEvents.onRequestReceived</a>	Request received event
<a href="#">TUIRoomEvents.onRequestCancelled</a>	Request cancelled event
<a href="#">TUIRoomEvents.onReceiveTextMessage</a>	Receive text message event
<a href="#">TUIRoomEvents.onReceiveCustomMessage</a>	Receive custom message event
<a href="#">TUIRoomEvents.onDeviceChange</a>	Device change event
<a href="#">TUIRoomEvents.onUserScreenCaptureStopped</a>	Screen sharing stopped event When a user uses the built-in browser's stop sharing button to end Screen Sharing, the user will receive the 'onUserScreenCaptureStopped' event to modify the Sharing status.

# TUIRoomKit

Last updated : 2023-11-16 14:48:02

## API API Introduction

TUIRoomKit API is a human video conference component with UI interface. By using TUIRoomKit API, you can quickly implement a WeChat-like audio/video call scenario through a simple interface. If you want to experience and debug the video conference effect, please read [the Quick Start Demo](#). If you want to embed our features directly into your project, please read [the Quick Access \(TUIRoomKit\)](#).

## API Overview

Reference the TUIRoomKit component in the page. For example, import the TUIRoomKit component in the `App.vue` component.

TUIRoomKit component divides users into host role and regular member role. The component provides [init](#), [createRoom](#), and [enterRoom](#) methods externally.

Hosts and regular members can initialize the application and user data to the TUIRoomKit component through the [init](#) method. The host can create and join a room through the [createRoom](#) method, and regular members can join the room created by the host through the [enterRoom](#) method.



```
<template>
  <room ref="TUIRoomRef"></room>
</template>

<script setup lang="ts">
  import { ref, onMounted } from 'vue';
  // Import the TUIRoom Component, make sure the import path is correct
  import Room from '../TUIRoom/index.vue';
  // Get the TUIRoom Component element, used to call the TUIRoom Component method
  const TUIRoomRef = ref();
```

```
onMounted(async () => {
// Initialize the TUIRoom component
// The host needs to initialize the TUIRoom component before creating a room
// Regular members need to initialize the TUIRoom component before entering
await TUIRoomRef.value.init({
  // To get the sdkAppId, please refer to Step 1
  sdkAppId: 0,
  // The unique identifier of the user in your business
  userId: '',
  // For local development and debugging, you can quickly generate userSig
  // Note that userSig and userId have a one-to-one correspondence
  userSig: '',
  // The nickname used by the user in your business
  userName: '',
  // The avatar URL used by the user in your business
  avatarUrl: '',
})
// By default, execute the create room operation, and you can choose to execute
await handleCreateRoom();
})

// The host creates a room, this method is only called when creating a room
async function handleCreateRoom() {
  // roomId is the room number entered by the user, and roomId must be of type string
  // roomMode includes 'FreeToSpeak' (free speech mode) and 'SpeakAfterTaking'
  // roomParam specifies the default behavior of the user when entering the room
  const roomId = '123456';
  const roomMode = 'FreeToSpeak';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
  };
  const createRoomInfo = {
    roomId,
    roomName: 'User defined room name' || roomId,
    roomMode,
    roomParam
  };
  await TUIRoomRef.value.createRoom(createRoomInfo);
}

// Regular members enter the room, and this method is called when regular members enter
async function handleEnterRoom() {
  // roomId is the room number entered by the user, and roomId must be of type string
  // roomParam specifies the default behavior of the user when entering the room
  const roomId = '123456';
  const roomParam = {
```

```
        isOpenCamera: true,
        isOpenMicrophone: true,
    };
    const enterRoomInfo = {
        roomId,
        roomParam
    };
    await TUIRoomRef.value.enterRoom(enterRoomInfo);
}
</script>

<style>
html, body {
    width: 100%;
    height: 100%;
    margin: 0;
}

#app {
    width: 100%;
    height: 100%;
}
</style>
```

## API Details

### TUIRoomkit Interface

#### init

Initialize TUIRoomKit Data, any user using TUIRoomKit needs to call the init method.



```
TUIRoomRef.value.init(roomData);
```

The parameters are shown in the table below:

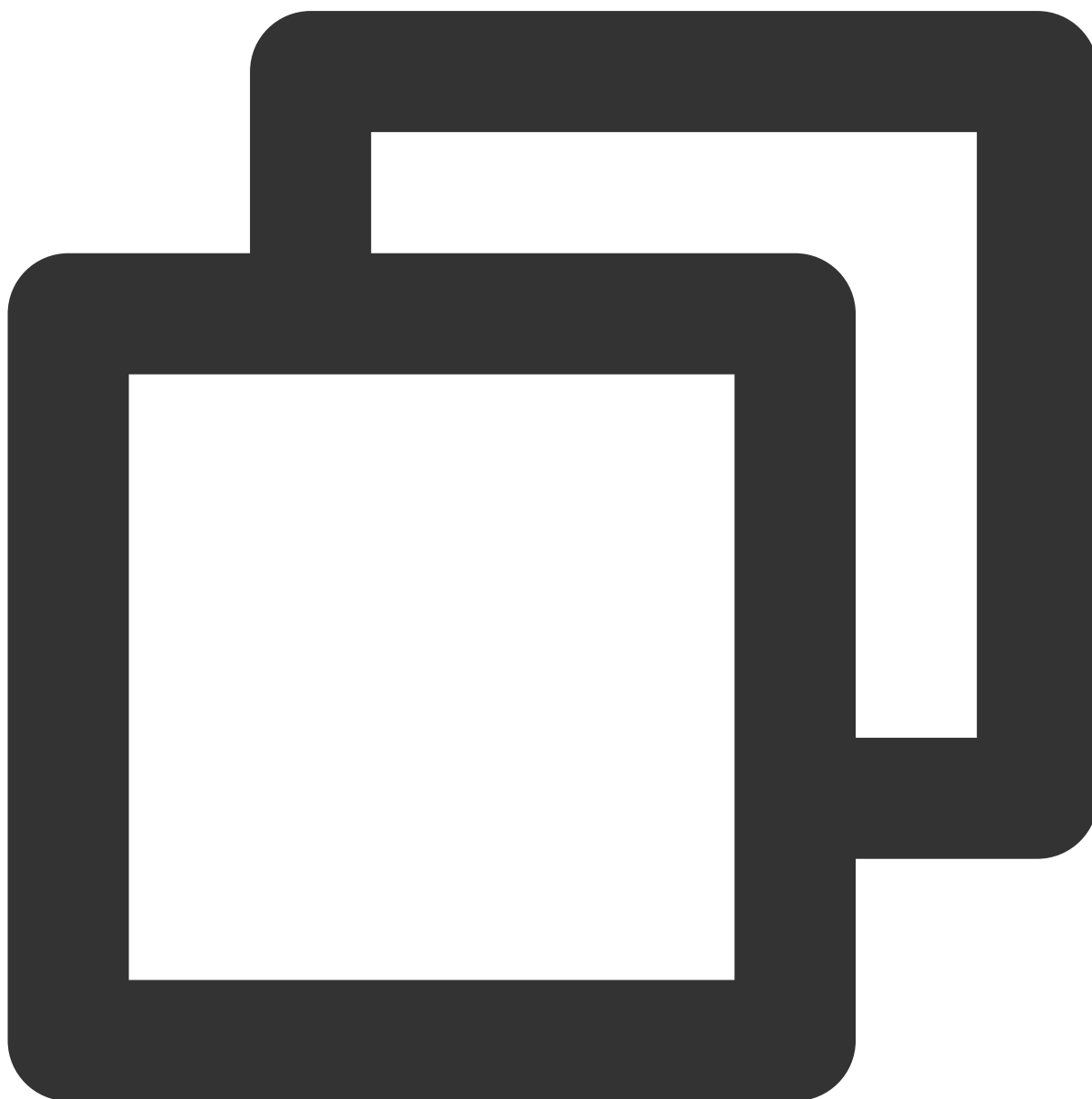
Parameter	Type	Meaning
roomData	object	-
roomData.sdkAppId	number	Customer's SDKAppId
roomData.userId	string	User's Unique ID



roomData.userSig	string	User's UserSig
roomData.userName	string	User's Nickname
roomData.avatarUrl	string	User's Avatar URL

## createRoom

Host creates a room.



```
TUIRoomRef.value.createRoom(roomId, roomMode, roomParam);
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
roomMode	string	Room mode, FreeToSpeak (Free Speech Mode) and SpeakAfterTakingSeat (Onstage Speech Mode), default is FreeToSpeak
roomParam	Object	Optional
roomParam.isOpenCamera	string	Optional, whether to open the camera when entering the room, default is closed
roomParam.isOpenMicrophone	string	Optional, whether to open the mic when entering the room, default is closed
roomParam.defaultCameraId	string	Optional, default Camera device ID
roomParam.defaultMicrophoneId	string	Optional, default mic device ID
roomParam.defaultSpeakerId	String	Optional, default Speaker device ID

## enterRoom

Regular member joins the room.



```
TUIRoomRef.value.enterRoom(roomId, roomParam);
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
roomParam	Object	Optional
roomParam.isOpenCamera	string	Optional, whether to open the camera when entering the

		room, default is closed
roomParam.isOpenMicrophone	string	Optional, whether to open the mic when entering the room, default is closed
roomParam.defaultCameraId	string	Optional, default Camera device ID
roomParam.defaultMicrophoneId	string	Optional, default mic device ID
roomParam.defaultSpeakerId	String	Optional, default Speaker device ID

## TUIRoomkit Event

### onCreateRoom

Create room Callback.



```
<template>
  <room ref="TUIRoomRef" @on-create-room="onCreateRoom"></room>
</template>

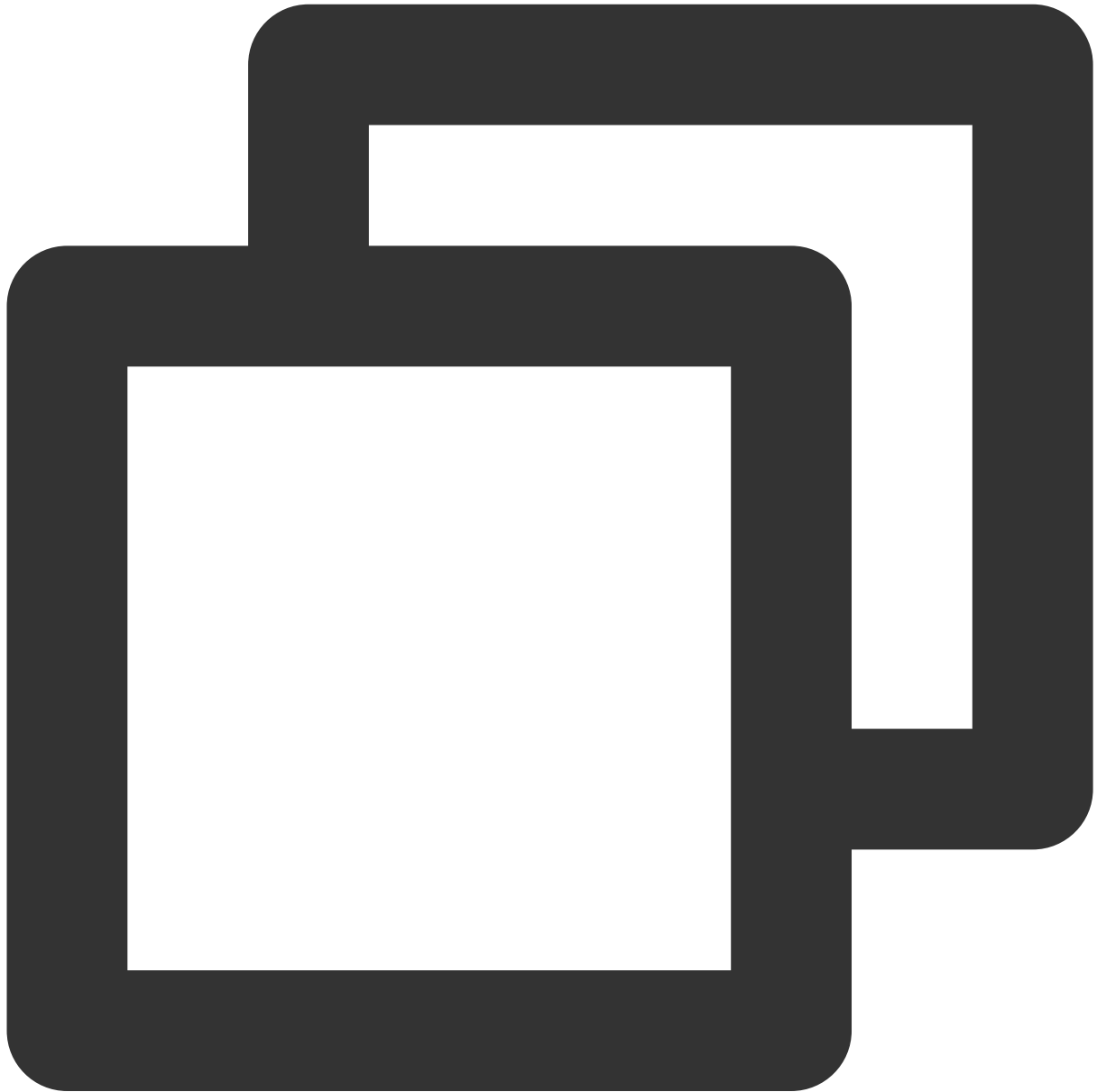
<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onCreateRoom(info) {
    if (info.code === 0) {
      console.log('Room created successfully')
    }
  }
</script>
```

```
    }  
  }  
</script>
```

## onEnterRoom

Entered room Callback.



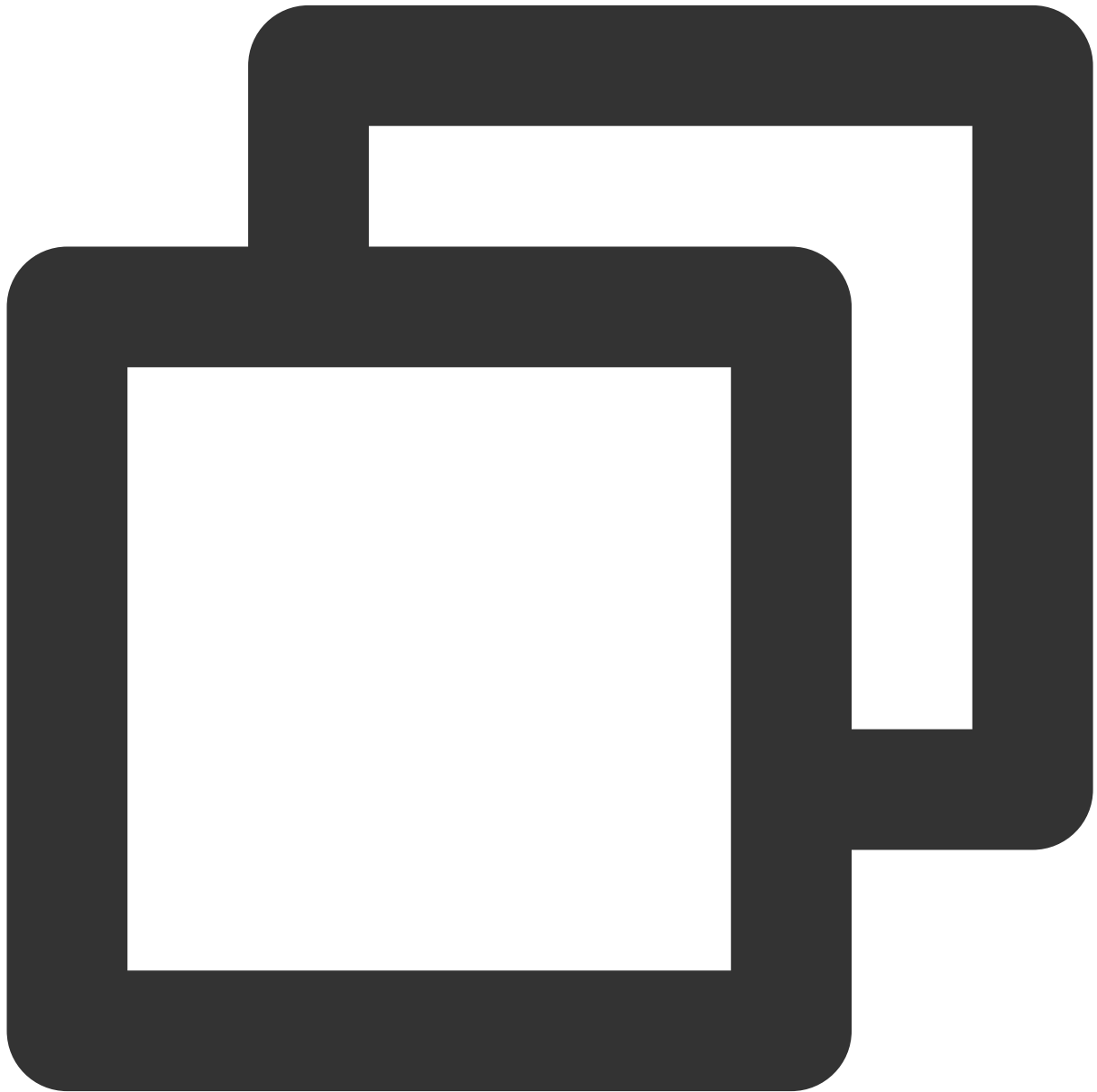
```
<template>  
  <room ref="TUIRoomRef" @on-enter-room="onEnterRoom"></room>  
</template>
```

```
<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onEnterRoom(info) {
    if (info.code === 0) {
      console.log('Entered room Success')
    }
  }
</script>
```

## onDestroyRoom

Host destroys room Callback.



```
<template>
  <room ref="TUIRoomRef" @on-destroy-room="onDestroyRoom"></room>
</template>

<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

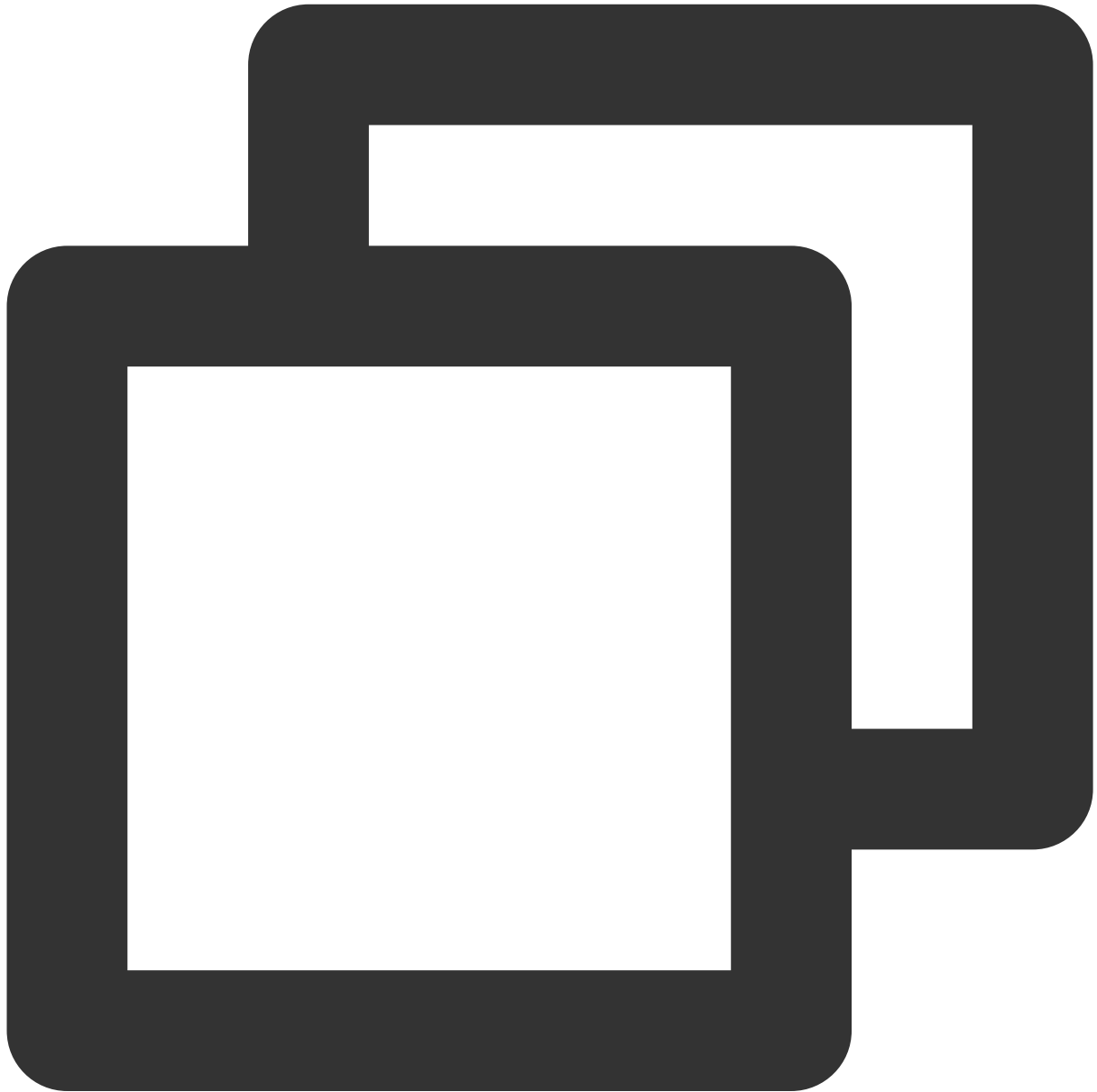
  function onDestroyRoom(info) {
    if (info.code === 0) {
      console.log('Host destroys room Success')
    }
  }
</script>
```



```
}  
}  
</script>
```

## onExitRoom

Regular member exits room Callback.



```
<template>  
  <room ref="TUIRoomRef" @on-exit-room="onExitRoom"></room>  
</template>
```

```
<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onExitRoom(info) {
    if (info.code === 0) {
      console.log('Regular member exits room Success')
    }
  }
</script>
```

## onKickedOutOfRoom

Regular member is kicked out of room by host Notification.



```
<template>
  <room ref="TUIRoomRef" @on-kicked-out-of-room="onKickedOutOfRoom"></room>
</template>

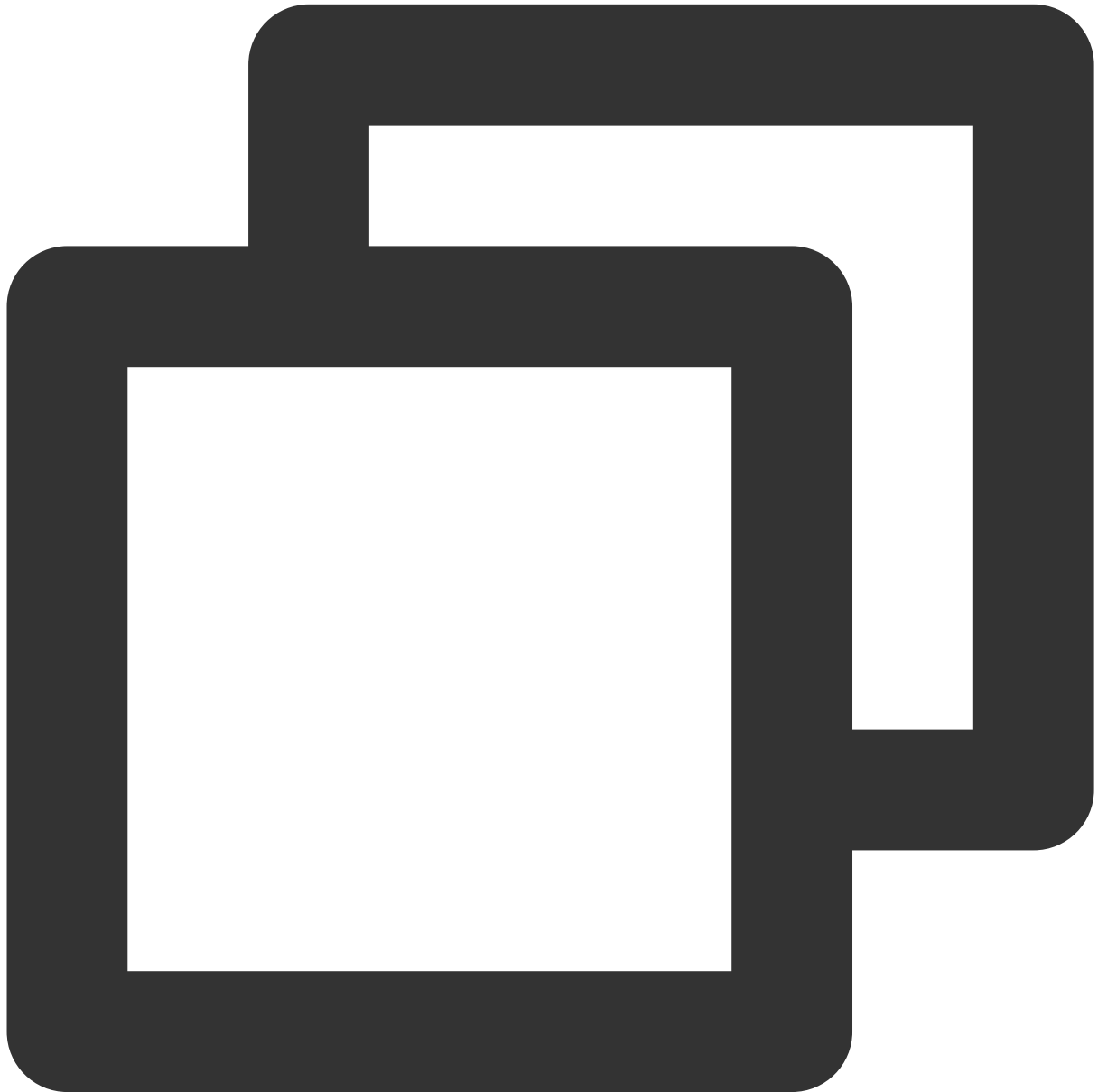
<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onKickedOutOfRoom({ roomId, message }) {
    console.log('Regular member is kicked out of room by host', roomId, message);
  }
</script>
```

```
</script>
```

### onKickedOffLine

User account logged in elsewhere, kicked offline.



```
<template>
  <room ref="TUIRoomRef" @on-kick-off-line="onKickedOffLine"></room>
</template>

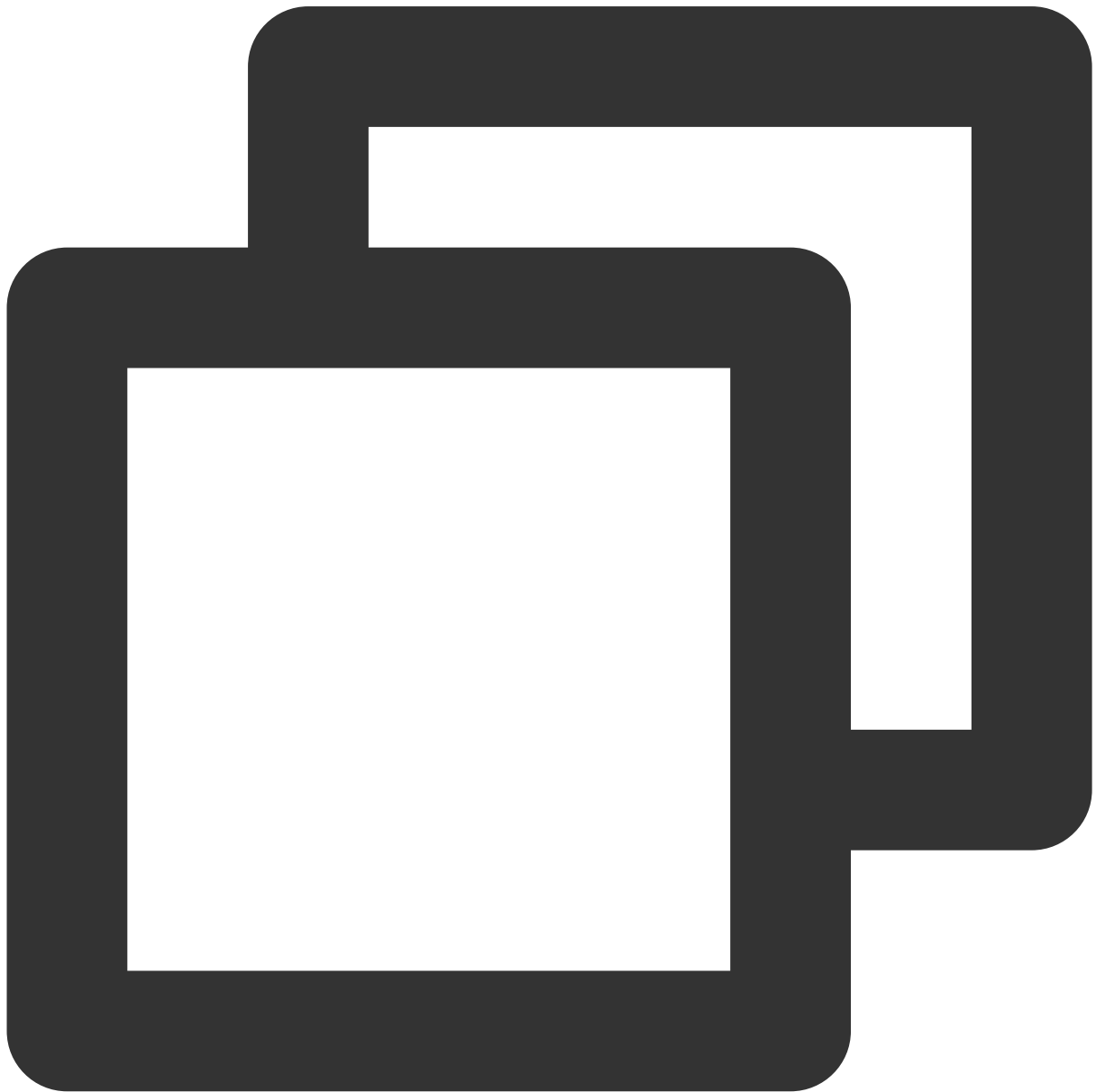
<script setup lang="ts">
```

```
// Import TUIRoom Component, make sure the import path is correct
import Room from './TUIRoom/index.vue';

function onKickedOffline({ message }) {
  console.log('Member kicked offline', message);
}
</script>
```

## onUserSigExpired

User userSig expired, please regenerate userSig.



```
<template>
  <room ref="TUIRoomRef" @on-user-sig-expired="onUserSigExpired"></room>
</template>

<script setup lang="ts">
  // Import TUIRoom Component, make sure the import path is correct
  import Room from './TUIRoom/index.vue';

  function onUserSigExpired() {
    console.log('User userSig expired, please regenerate userSigg.');
```

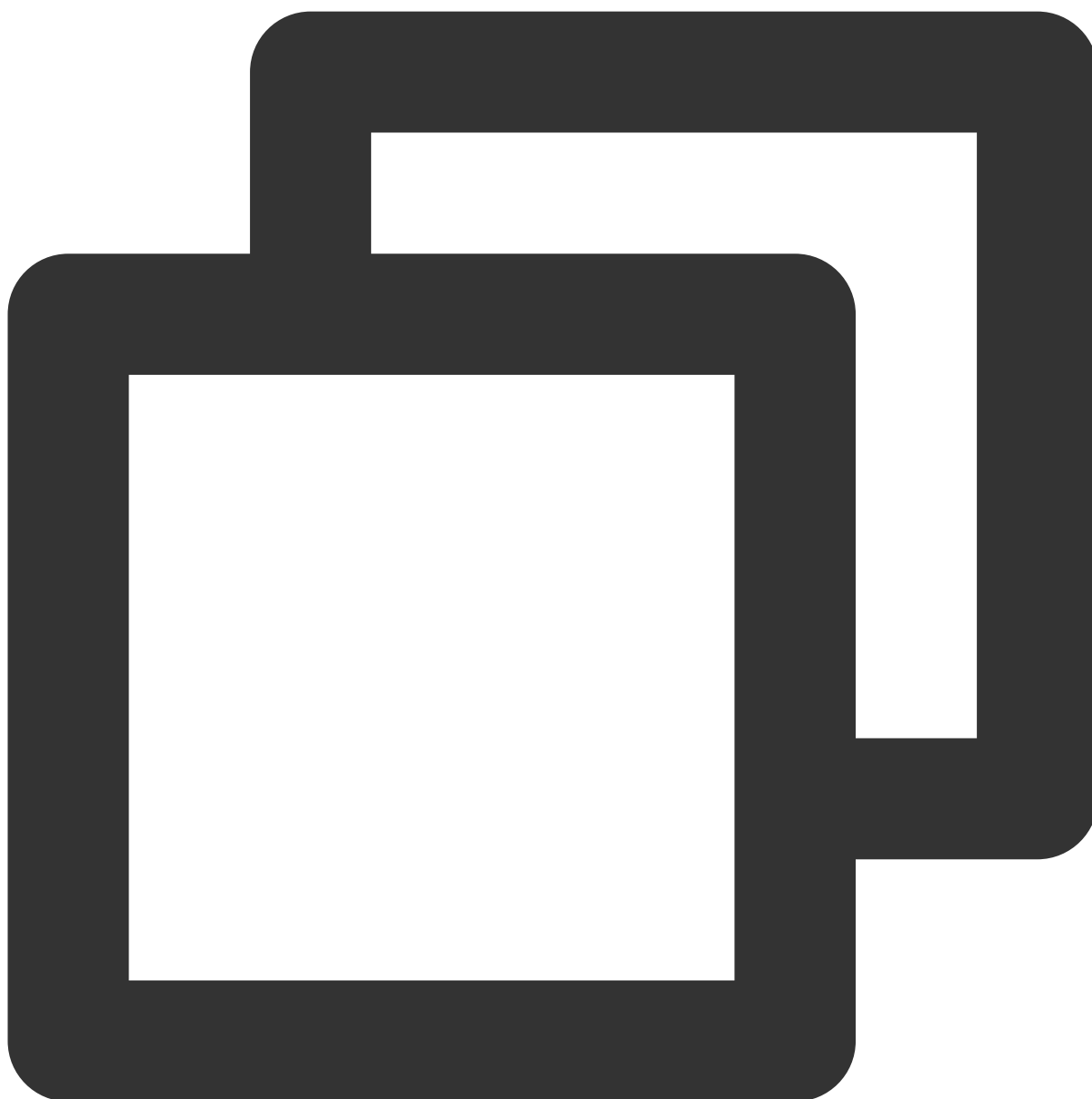
# TUIRoomEngine

Last updated : 2023-11-16 14:46:30

## TUIRoomEngine Introduction

TUIRoomEngine SDK provides Room Management, Multi-person Tencent Real-Time Communication, Screen Sharing, Member Management, Instant Messaging, and other features.

### **Installation Method:**



```
// Use npm
npm i @tencentcloud/tuiroom-engine-electron --save

// Use pnpm
pnpm i @tencentcloud/tuiroom-engine-electron --save

// Use yarn
yarn add @tencentcloud/tuiroom-engine-electron
```



# TUIRoomEngine API

## TUIRoomEngine Static Method

API	Description
<a href="#">once</a>	<b>Listening to the TUIRoomEngine ready Event.</b> <b>Note: All methods other than TUIRoomEngine.login must be executed after listening to the TUIRoomEngine ready event and the successful execution of the TUIRoomEngine.login method.</b>
<a href="#">login</a>	Login to TUIRoomEngine
<a href="#">setSelfInfo</a>	Setting the current user's Basic information (Username, User Avatar)
<a href="#">getSelfInfo</a>	Get the current user's Basic information (Username, User Avatar)
<a href="#">logout</a>	Logout of TUIRoomEngine

## roomEngine Room Management API

API	Description
<a href="#">createRoom</a>	Create Room
<a href="#">enterRoom</a>	Enter Room
<a href="#">destroyRoom</a>	Destroy Room
<a href="#">exitRoom</a>	Exit Room
<a href="#">fetchRoomInfo</a>	Get Room data
<a href="#">updateRoomNameByAdmin</a>	Update Name (Call by Group Owner or Administrator only)
<a href="#">updateRoomSpeechModeByAdmin</a>	Update Speaking Mode (Call by Group Owner or Administrator only)
<a href="#">getUserList</a>	Get User List
<a href="#">getUserInfo</a>	Learn more about the user

## roomEngine Audio Video API

API	Description
<a href="#">setLocalVideoView</a>	Set Local Stream Rendering Position

<a href="#">openLocalCamera</a>	Capturing Local Camera Video streams
<a href="#">closeLocalCamera</a>	Close Local Camera
<a href="#">openLocalMicrophone</a>	Open Local mic
<a href="#">closeLocalMicrophone</a>	Close Local mic
<a href="#">updateVideoQuality</a>	Set Local Video Parameters
<a href="#">updateAudioQuality</a>	Set Local Audio Parameters
<a href="#">startPushLocalVideo</a>	Start Pushing Local Video streams to Remote
<a href="#">stopPushLocalVideo</a>	Stop Pushing Local Video streams to Remote
<a href="#">startPushLocalAudio</a>	Start Pushing Local Audio Stream to Remote
<a href="#">stopPushLocalAudio</a>	Stop Pushing Local Audio Stream to Remote
<a href="#">setRemoteVideoView</a>	Set Remote Stream Rendering Area
<a href="#">startPlayRemoteVideo</a>	Start Playback Remote User Video streams
<a href="#">stopPlayRemoteVideo</a>	Stop Playback Remote User Video streams
<a href="#">muteRemoteAudioStream</a>	Stop Remote User Audio Stream

## roomEngine Member Management API

API	Description
<a href="#">openRemoteDeviceByAdmin</a>	Request Remote User to Open Media Device
<a href="#">applyToAdminToOpenLocalDevice</a>	Participant Apply to Host to Open Device
<a href="#">closeRemoteDeviceByAdmin</a>	Close Remote User Media Device
<a href="#">cancelRequest</a>	Cancel Sent Request
<a href="#">responseRemoteRequest</a>	Reply to Remote User Request
<a href="#">changeUserRole</a>	Change User Role
<a href="#">kickRemoteUserOutOfRoom</a>	Kick Out User from Room
<a href="#">disableDeviceForAllUserByAdmin</a>	Disable/Enable All Users' Media Device
<a href="#">disableSendingMessageForAllUser</a>	Disallow/Allow All Users to Send Message

[disableSendingMessageByAdmin](#)

Disallow/Allow Specific User to Send Message

## roomEngine Screen Sharing API

API	Description
<a href="#">startScreenSharingElectron</a>	Start Screen Sharing
<a href="#">stopScreenSharingElectron</a>	Stop Screen Sharing
<a href="#">getScreenSharingTarget</a>	Obtain Screen Sharing List
<a href="#">selectScreenSharingTarget</a>	Switch Screen Sharing Window

## roomEngine Mic Position Management API

API	Description
<a href="#">setMaxSeatCount</a>	Set Room Maximum Value
<a href="#">getSeatList</a>	Get Mic Position Information
<a href="#">takeSeat</a>	Get Mic Position
<a href="#">leaveSeat</a>	Release Mic Position
<a href="#">takeUserOnSeatByAdmin</a>	Invite Others to Go Live (Only Room Host and Administrator can invoke this method)
<a href="#">kickUserOffSeatByAdmin</a>	Kick Others Off the Mic (Only Room Host and Administrator can invoke this method)
<a href="#">lockSeatByAdmin</a>	Lock a Specific Mic Position Status (Only Room Host and Administrator can invoke this method)

## roomEngine Message Sending API

API	Description
<a href="#">sendTextMessage</a>	Send Text Message
<a href="#">sendCustomMessage</a>	Send Custom Message

## roomEngine Device Management API

--	--

API	Description
<a href="#">getCameraDevicesList</a>	Get Camera Device List
<a href="#">getMicDevicesList</a>	Get Mic Device List
<a href="#">getSpeakerDevicesList</a>	Get Speaker Device List
<a href="#">setCurrentCameraDevice</a>	Set to Use Camera
<a href="#">setCurrentMicDevice</a>	Set to Use Mic
<a href="#">setCurrentSpeakerDevice</a>	Set to Use Speaker
<a href="#">getCurrentCameraDevice</a>	Get the Currently Used Camera
<a href="#">getCurrentMicDevice</a>	Get the Currently Used Mic
<a href="#">getCurrentSpeakerDevice</a>	Get the Currently Used Speaker
<a href="#">startCameraDeviceTest</a>	Start Camera Test
<a href="#">stopCameraDeviceTest</a>	Stop Camera Test

## roomEngine Event Listening API

API	Description
<a href="#">on</a>	Listen to <a href="#">TUIRoomEvents</a>
<a href="#">off</a>	Cancel Listening to <a href="#">TUIRoomEvents</a>

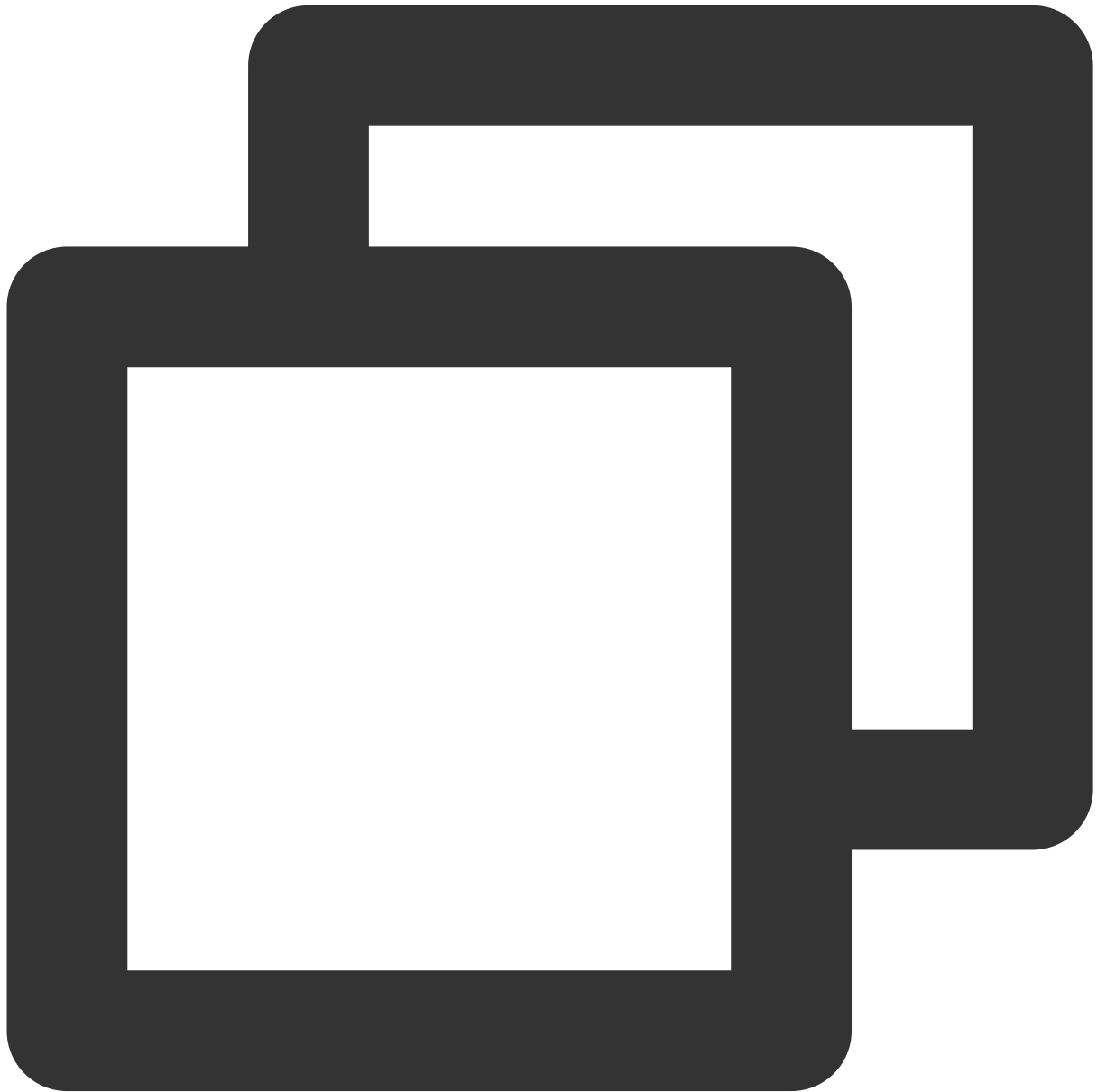
## roomEngine Other API

API	Description
<a href="#">getTRTCCloud</a>	Get trtcCloud Instance
<a href="#">getTIM</a>	Get tim Instance

## API Details

### once

Monitor TUIRoomEngine 'ready' Event



```
TUIRoomEngine.once('ready', () => {
  const roomEngine = new TUIRoomEngine();

  await TUIRoomEngine.init({
    sdkAppId: 0,    // Fill in the sdkAppId you applied for
    userId: '',     // Fill in the userId corresponding to your business
    userSig: '',    // Fill in the userSig calculated by the server or locally
  });

  await roomEngine.createRoom({
    roomId: '12345', // Enter your Room ID, note that the Room ID is required to
```

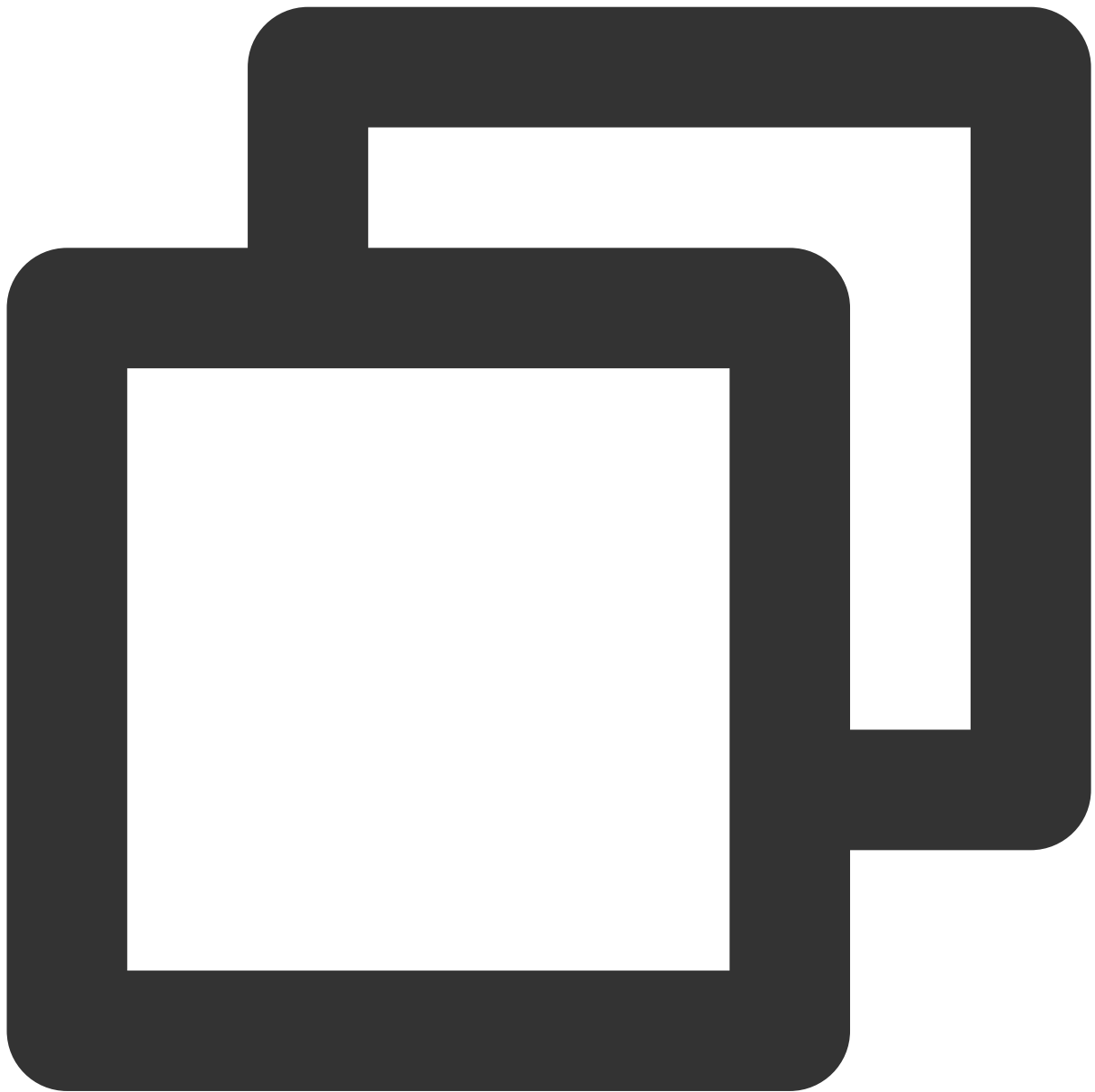
```
name: 'Test Room', // Enter your room name, the default room name is roomId
roomType: TUIRoomType.kGroup, // Set the room type to TUIRoomType.kGroup type
});
});
```

## login

### Description:

In version v1.0.0, this interface is named TUIRoomEngine.init, please use TUIRoomEngine.login to log in TUIRoomEngine in v1.0.1 and above versions.

You must log in to TUIRoomEngine before you can call other methods of TUIRoomEngine and its instances.



```
// Login TUIRoomEngine
await TUIRoomEngine.login({
  sdkAppId: 0,    // Fill in the sdkAppId you applied for
  userId: '',     // Fill in the userId corresponding to your business
  userSig: '',    // Fill in the userSig calculated by the server or locally
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning

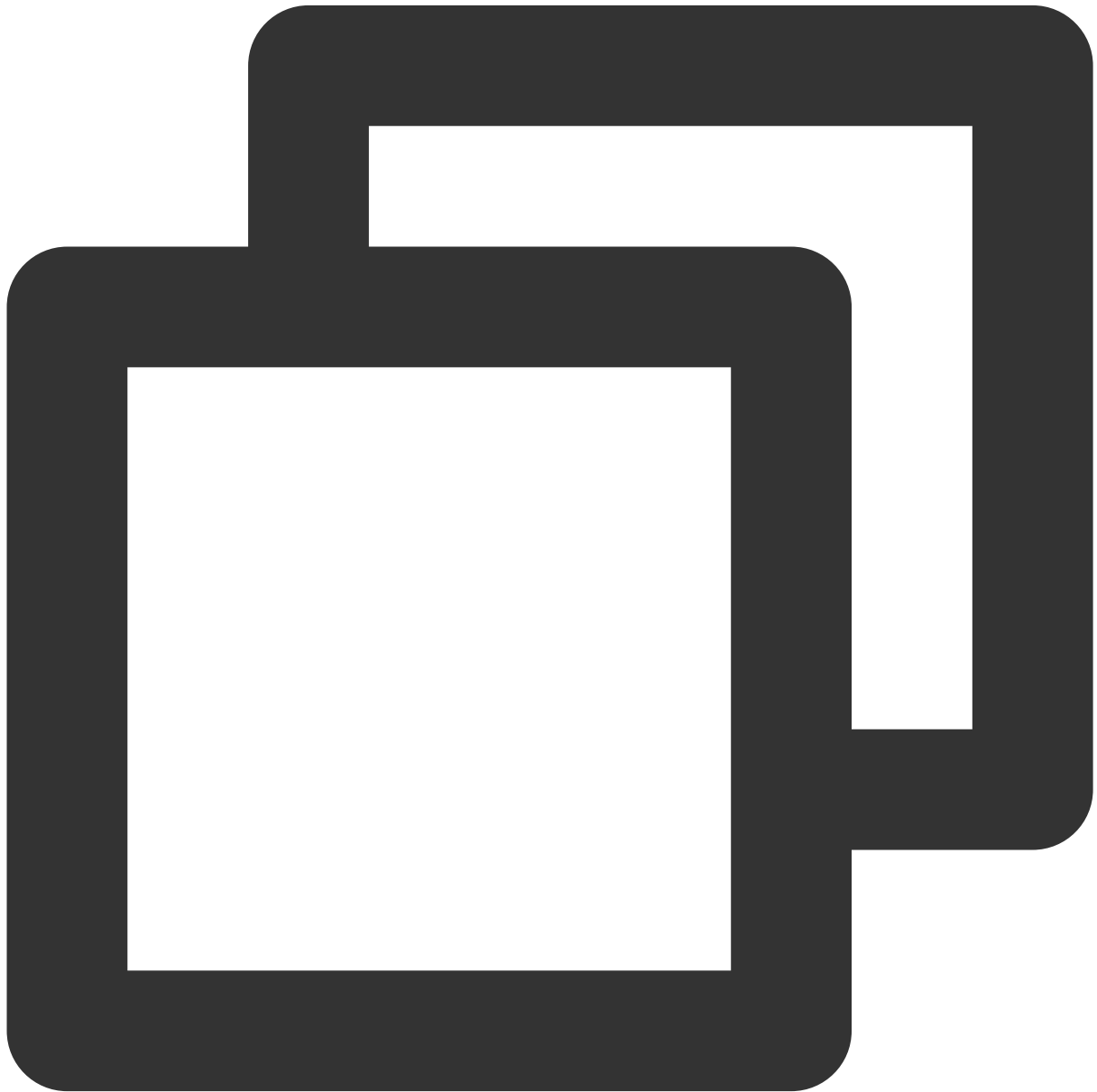
sdkAppld	number	Required	-	After clicking Application Management > Create Application in the Tencent Real-Time Communication Console, you can get the sdkAppld information in Application Info.
userId	string	Required	-	It is recommended to limit the user ID length to 32 bytes, and only allow uppercase and lowercase English letters (a-zA-Z), numbers (0-9), underscores, and hyphens.
userSig	string	Required	-	UserSig signature Please refer to <a href="#">UserSig related</a> for the method of calculating userSig.
tim	TIM	Not Required	-	If you want to use more capabilities of the Chat SDK while accessing roomEngine, you can pass the created tim instance into TUIRoomEngine. For the creation method of tim instance, please refer to <a href="#">TIM.create</a> .

**Returns** *Promise<void>*

### setSelfInfo

Set current user Basic information (Username, User avatar).





```
// Set current user Username and User avatar
await TUIRoomEngine.setSelfInfo({
  userName: '',      // Enter your New username
  avatarUrl: '',     // Enter your New avatar URL
});
```

**Parameter:**

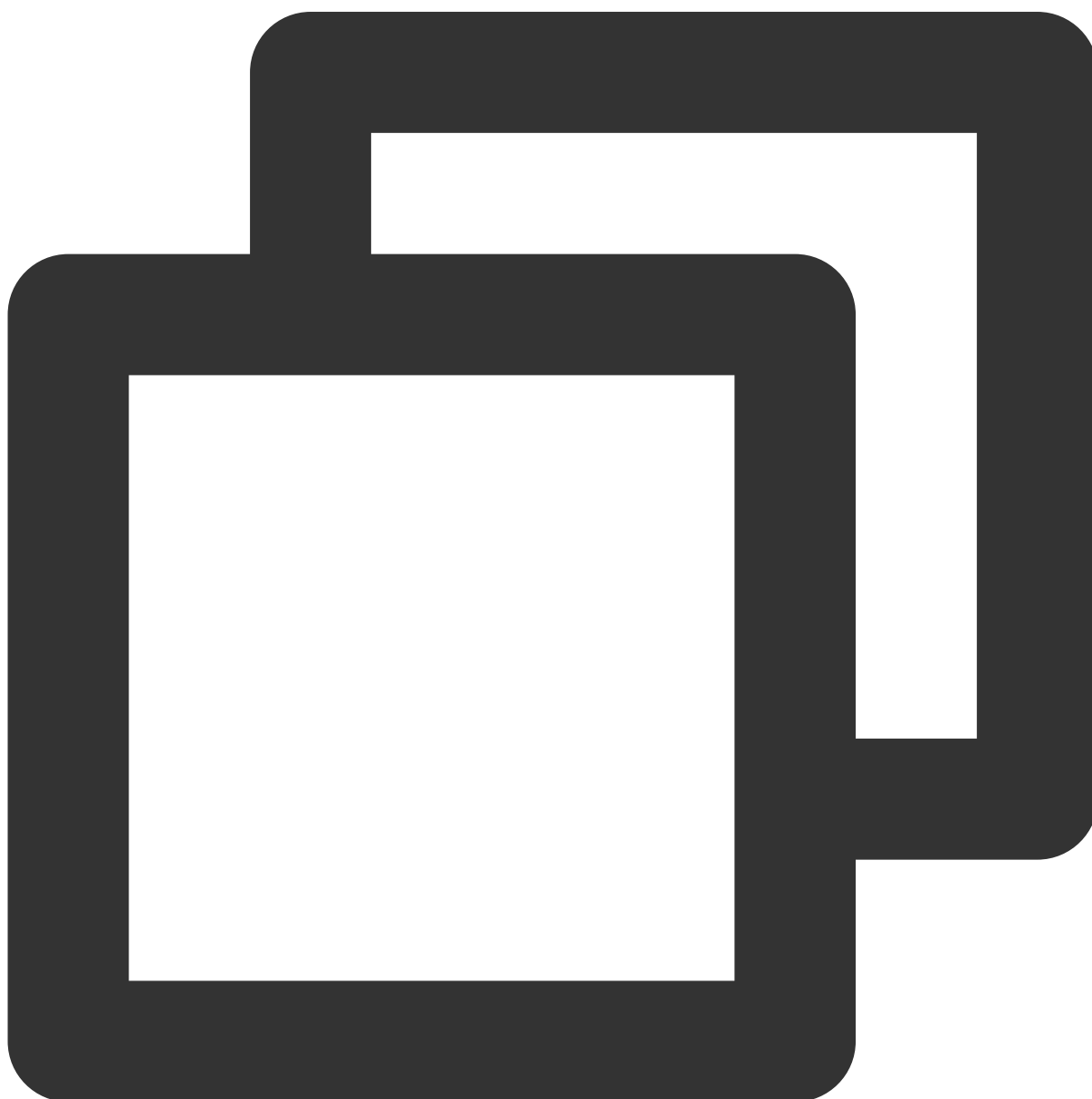
Parameter	Type	Description	Default Value	Meaning

userName	string	Required	-	User Name
avatarUrl	string	Required	-	User avatar

**Returns** *Promise<void>*

## getSelfInfo

Get current user Basic information (Username, User avatar).



```
// Get current user Username and User avatar
```

```
const loginUserInfo = await TUIRoomEngine.getSelfInfo();
```

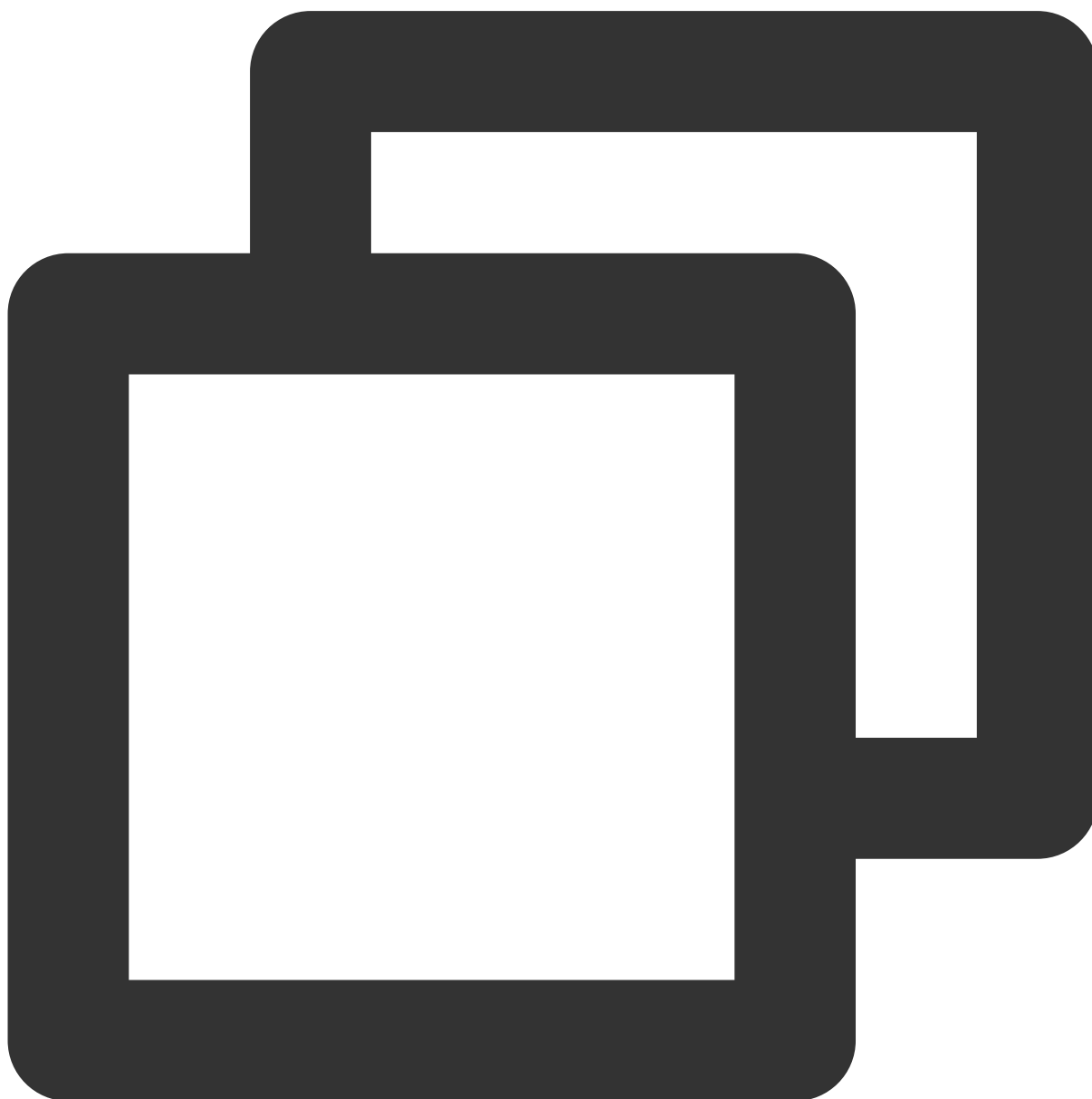
Returns *Promise*<[TUILoginUserInfo](#)> *loginUserInfo*

## logout

### Description:

Interface since v1.0.1 Version support.

Logout TUIRoomEngine



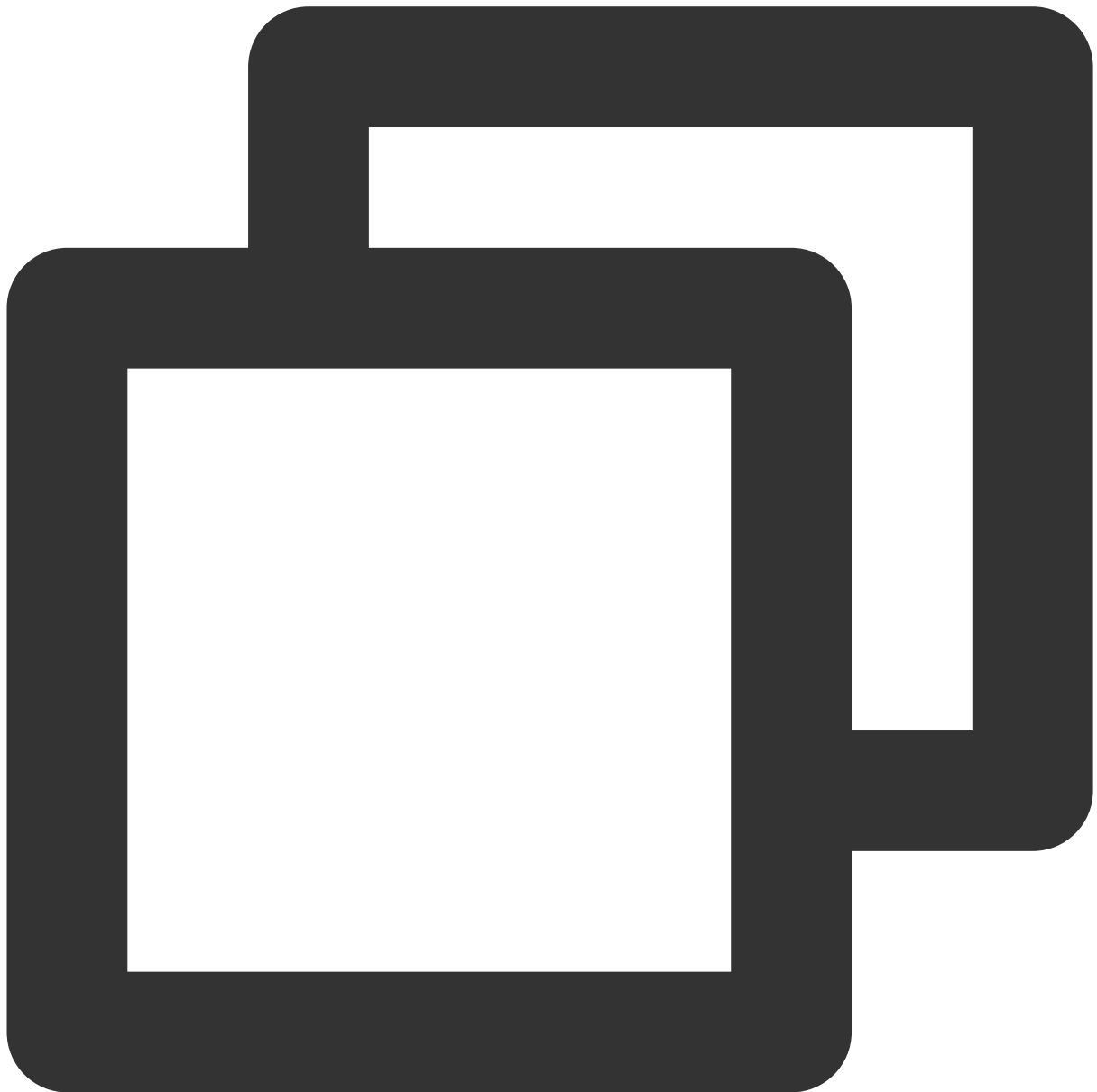
```
// Logout TUIRoomEngine
```

```
await TUIRoomEngine.logout();
```

Returns *Promise<void>*

## createRoom

Host creates room, Call createRoom User is the room owner. When creating a room, you can set Room ID, Room name, Room type, Speaking mode, Whether to allow users to join and enable audio and video, Send message and other functions.



```
const roomEngine = new TUIRoomEngine();
```

Parameter:

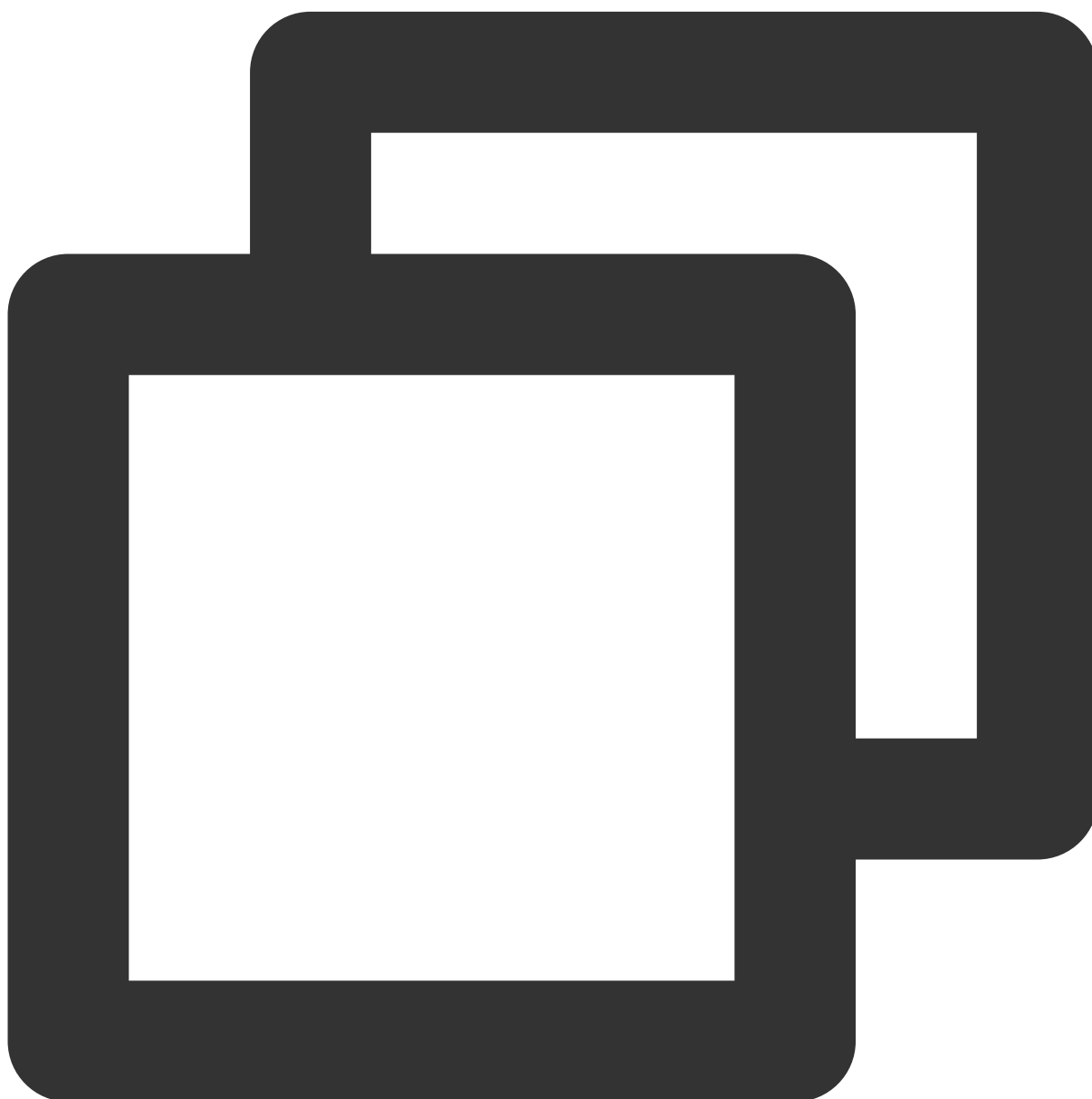
Parameter	Type	Description	Default Value	Meaning
roomId	string	Required	-	Room ID (byte char Engl Num Spa {}   ,
roomName	string	Optional	roomId	Room name
roomType	TUIRoomType	Optional	TUIRoomType.kConference	Room type (Office room TUI E-con room TUI
speechMode	TUISpeechMode	Optional	TUISpeechMode.kFreeToSpeak	Speech mode (edu Set : TUI: can defa Set : TUI: do n defa to ap or m

				and mod Set : TUI: user get p and For` broa Set : TUI: for h Set : TUI: host TUI: TUI: mod
isMicrophoneDisableForAllUser	boolean	Optional	false	Enal mute
isCameraDisableForAllUser	boolean	Optional	false	Enal not e
isMessageDisableForAllUser	boolean	Optional	false	Allov proh
maxSeatCount	number	Optional	-	Max For` (edu is nc For` broa max
enableCDNStreaming	boolean	Optional	false	Enal
cdnStreamDomain	string	Optional	"	Pustl

Returns *Promise<void>*

## enterRoom

Entered room interface.



```
const roomEngine = new TUIRoomEngine();
const roomInfo = await roomEngine.enterRoom({
  roomId: '12345',
});
```

Parameter:

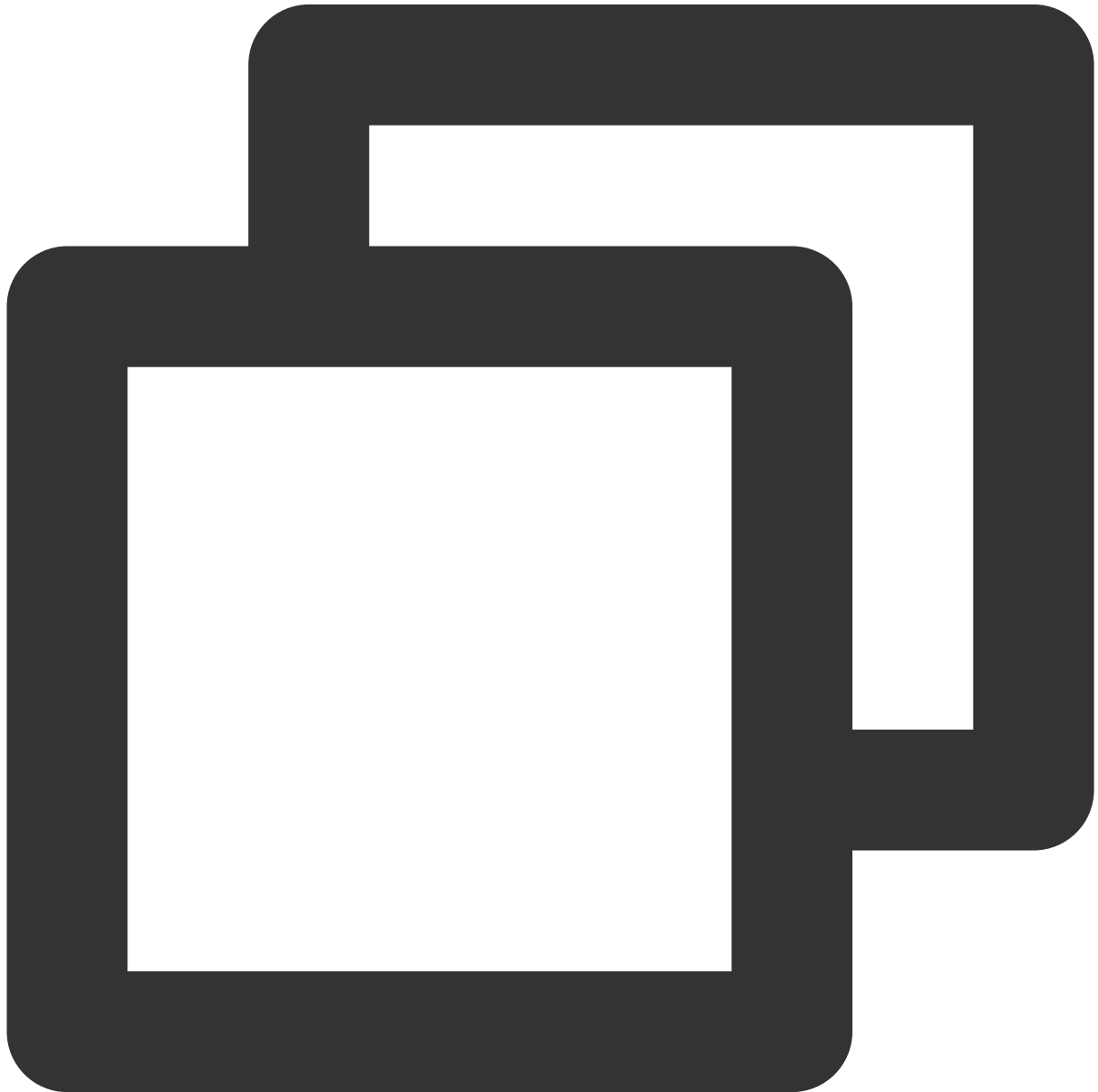
Parameter	Type	Description	Default Value	Meaning
roomId	string	Required	-	Room ID

**Returns** *Promise*<[TUIRoomInfo](#)> *roomInfo*

This interface returns the current Room data

## **destroyRoom**

Close the room interface, the room must be closed by the room owner, and the room cannot be entered after it is closed.



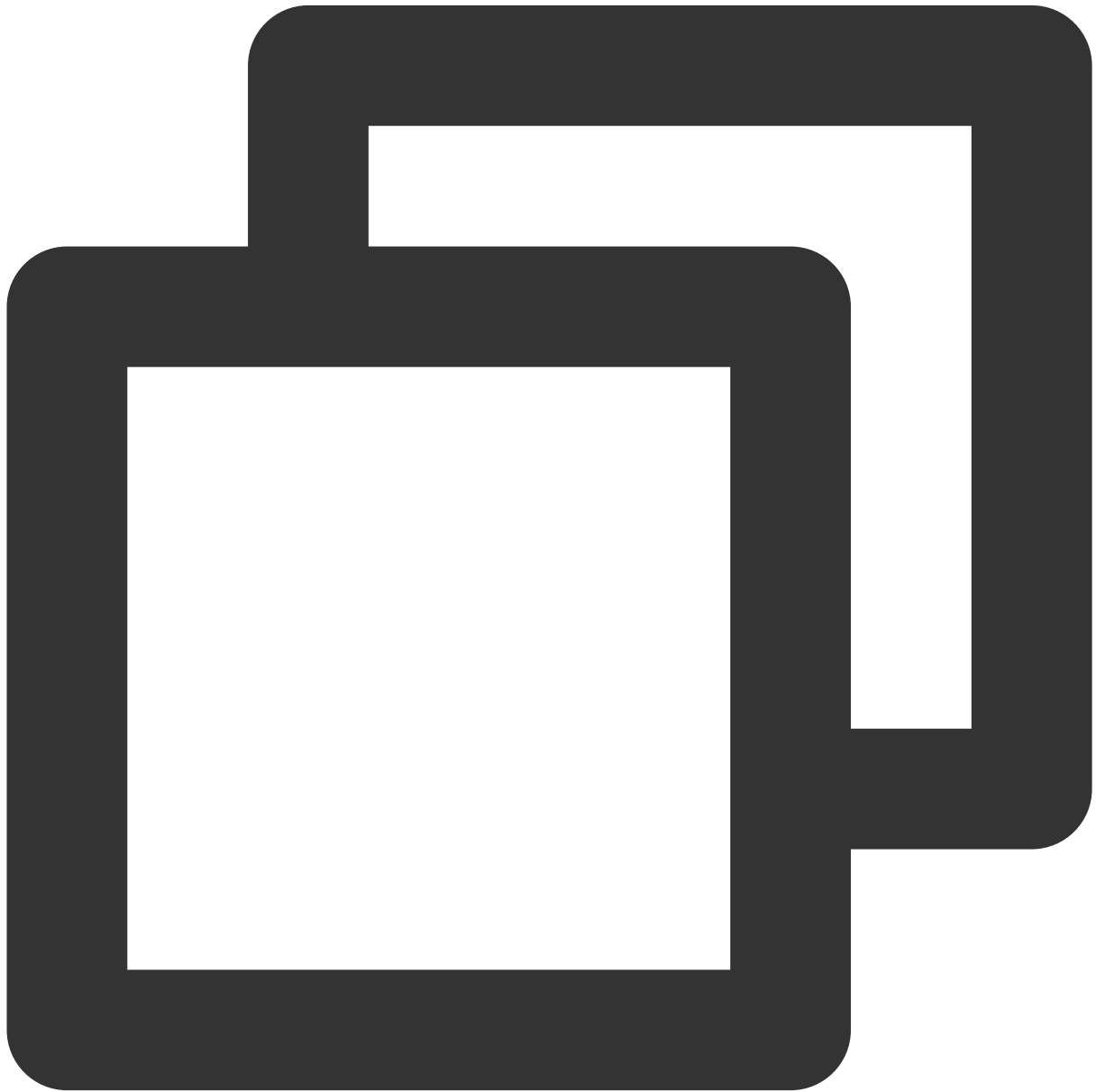
```
const roomEngine = new TUIRoomEngine();  
await roomEngine.destroyRoom();
```



**Returns** *Promise<void>*

## exitRoom

Leave the room interface, users can leave the room through exitRoom after executing enterRoom.

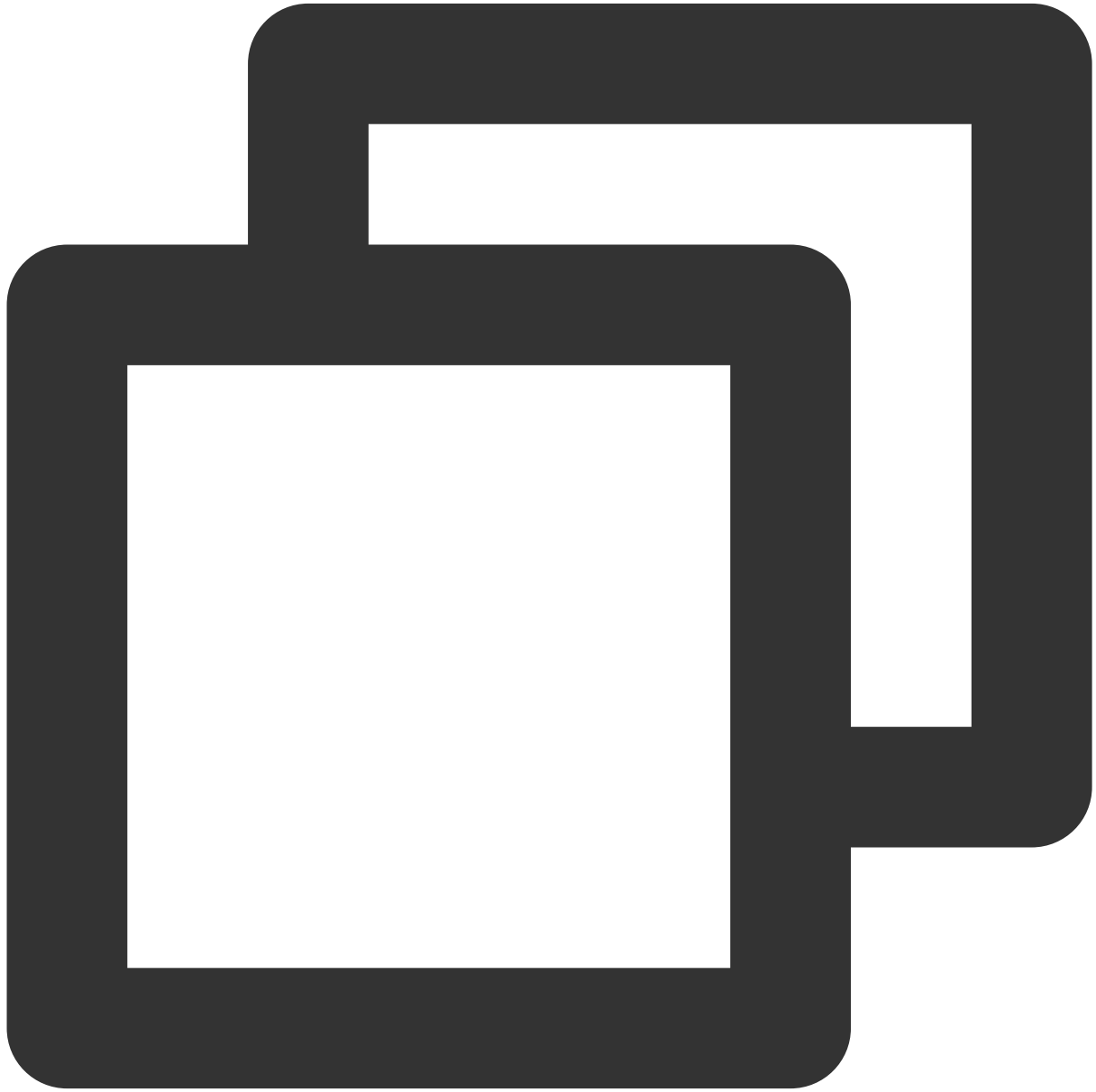


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.exitRoom();
```

**Returns** *Promise<void>*

## fetchRoomInfo

Get Room information.

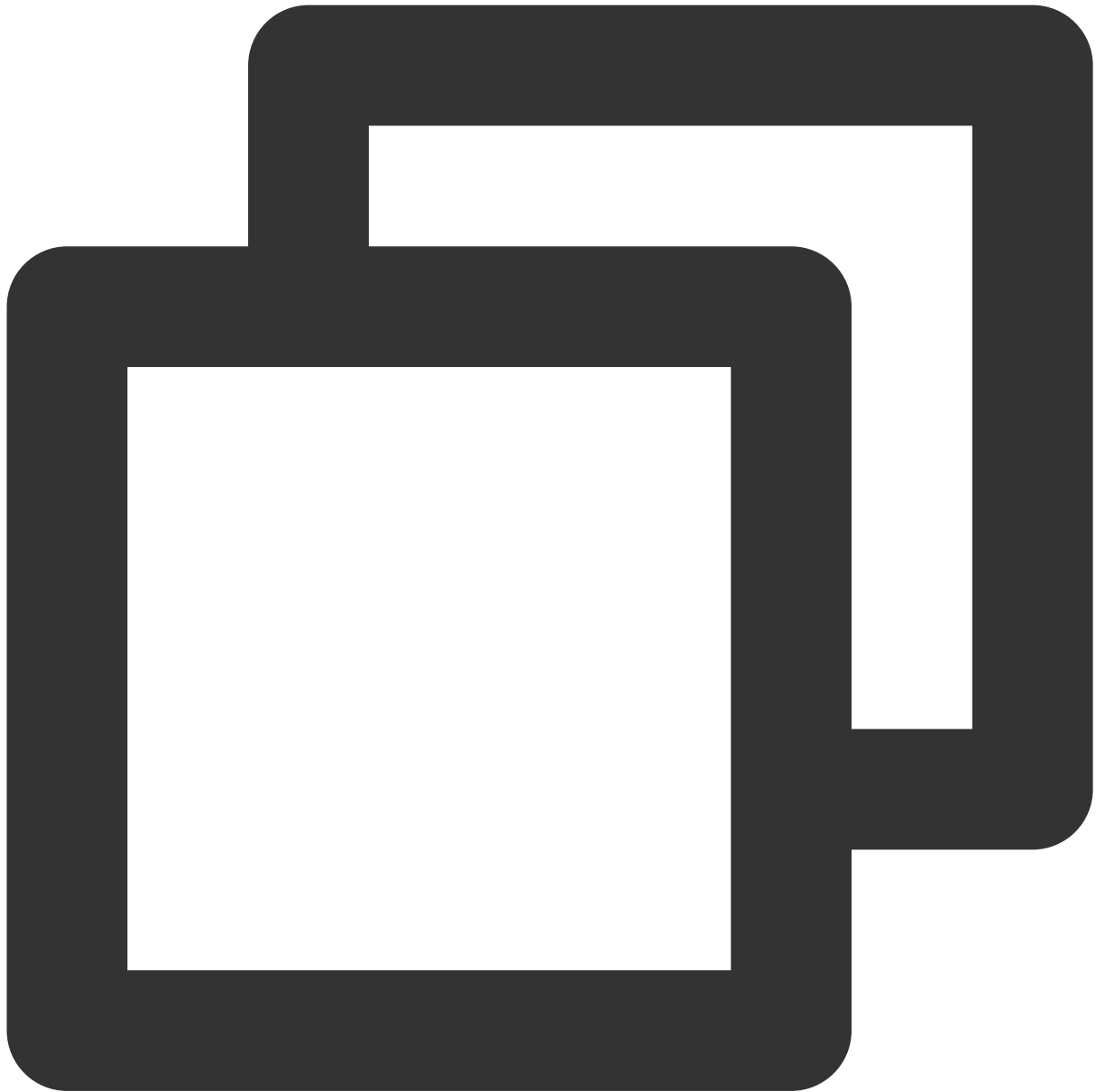


```
const roomEngine = new TUIRoomEngine();  
const roomInfo = roomEngine.fetchRoomInfo();
```

Returns : *Promise*<[TUIRoomInfo](#)> *roomInfo*

### **updateRoomNameByAdmin**

Update the current room's name (only group owner or admin can invoke).



```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.updateRoomNameByAdmin({ roomName: 'new name' });
```

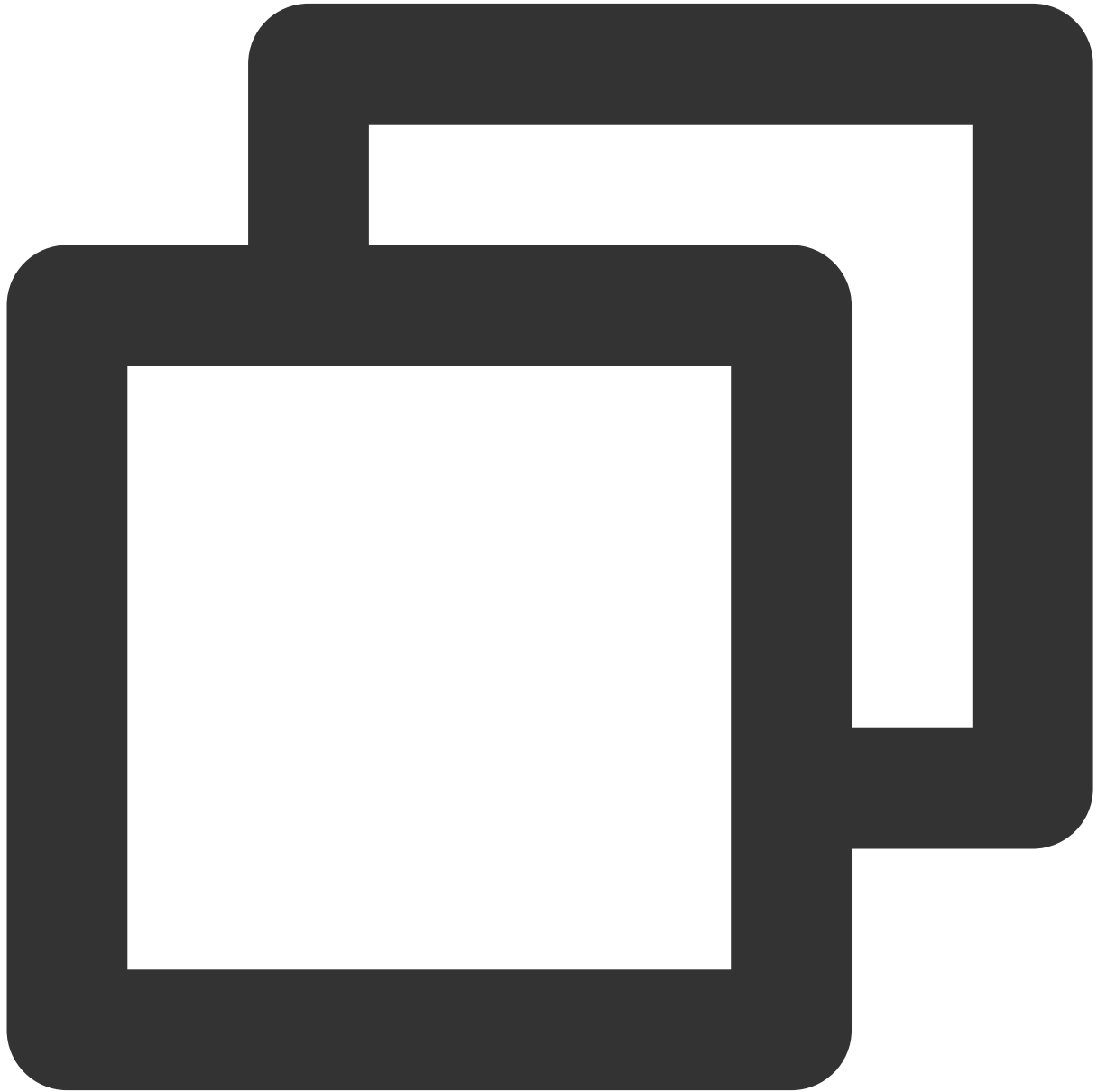
Parameter:

Parameter	Type	Description	Default Value	Meaning
roomName	string	Required	-	Update the room's name, with the requirement that roomName is not an empty string

**Returns** *Promise<void>*

### **updateRoomSpeechModeByAdmin**

Update the room's speaking mode (only group owner or admin can invoke).



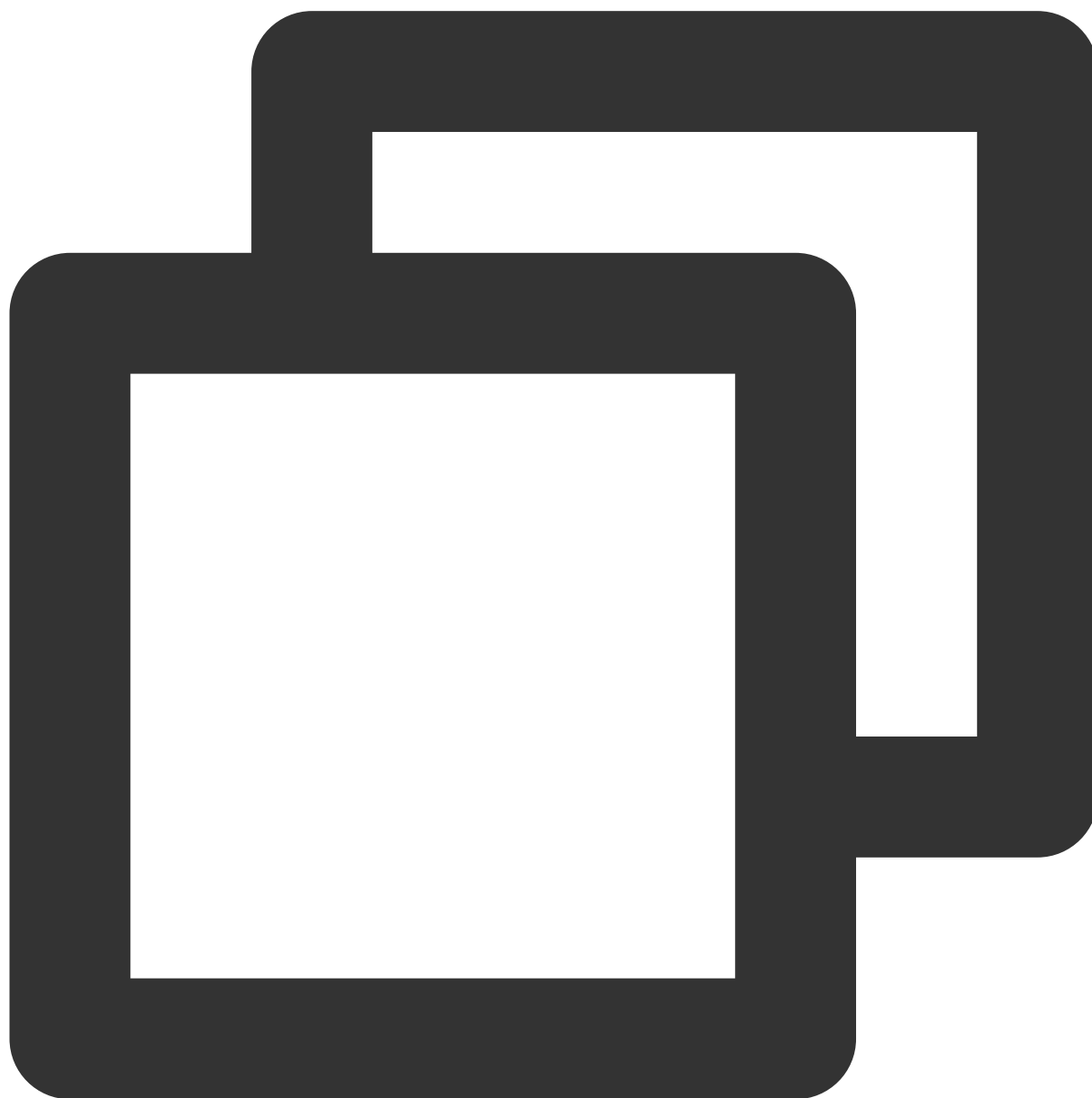
```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.updateRoomSpeechModeByAdmin({
  speechMode: TUISpeechMode.kSpeakAfterTakingSeat // Update to Go Live Speaking mo
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
speechMode	TUISpeechMode	Required	-	Update the room's speaking mode

## getUserList

Get the current room's user list, note that the maximum number of user lists fetched by this interface is 100.



```
const roomEngine = new TUIRoomEngine();
```

```
const userList = [];  
let result;  
do {  
    result = await globalProperties.roomEngine.getUserList();  
    userList.push(...result.userInfoList);  
} while (result.nextSequence !== 0)
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
nextSequence	number	Optional	0	Offset, default is to start fetching users from 0

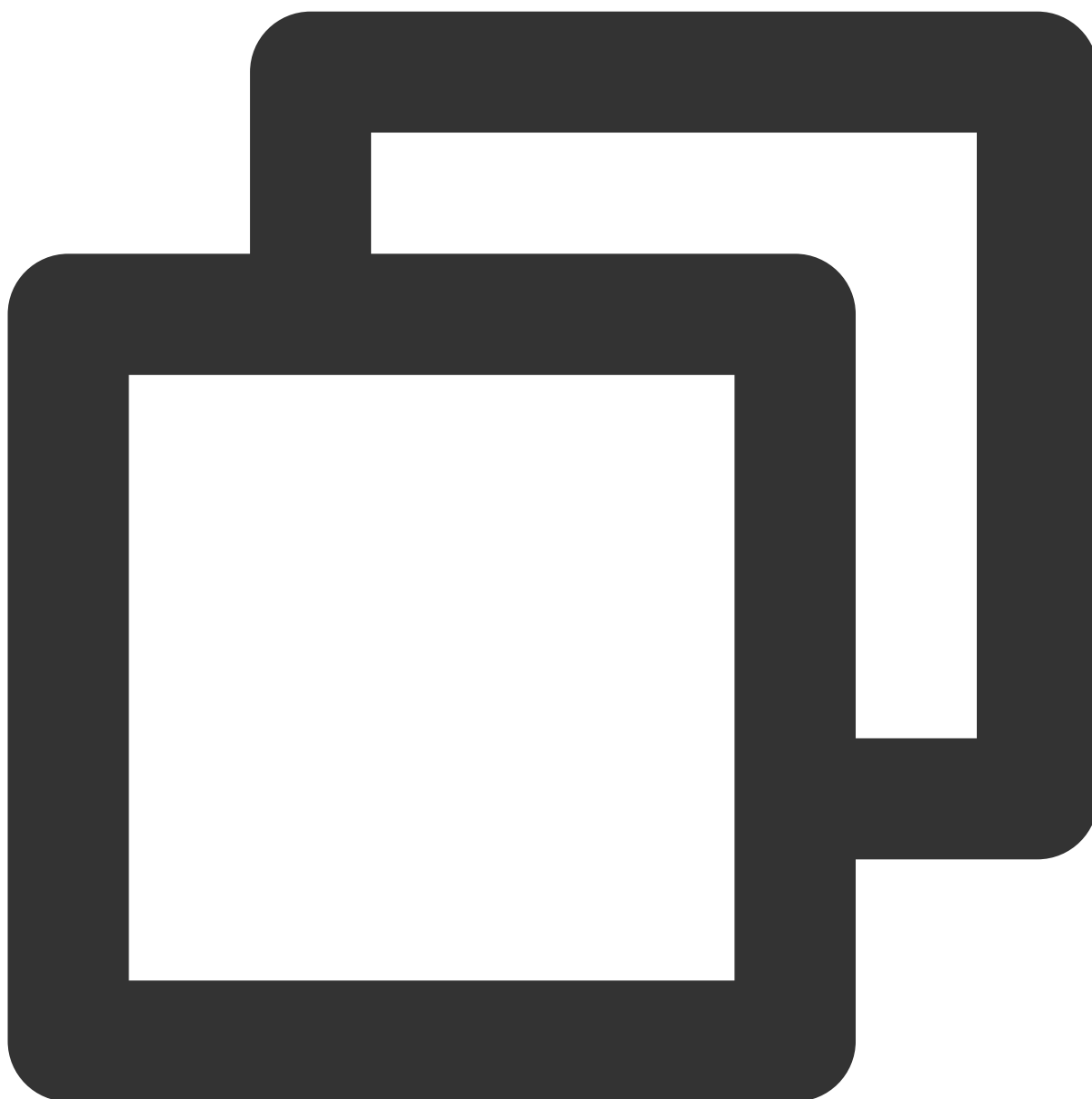
**Returns** : *Promise<object> result*

result.nextSequence is the offset for fetching group users next time, if result.nextSequence is 0, it means that all userList have been fetched

result.userInfoList is the userList fetched this time

## getUserInfo

Get the detailed information of the user.



```
const roomEngine = new TUIRoomEngine();
const userList = [];
const userInfo = await roomEngine.getUserInfo({
  userId: 'user_12345',
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	Get the detailed information of the user

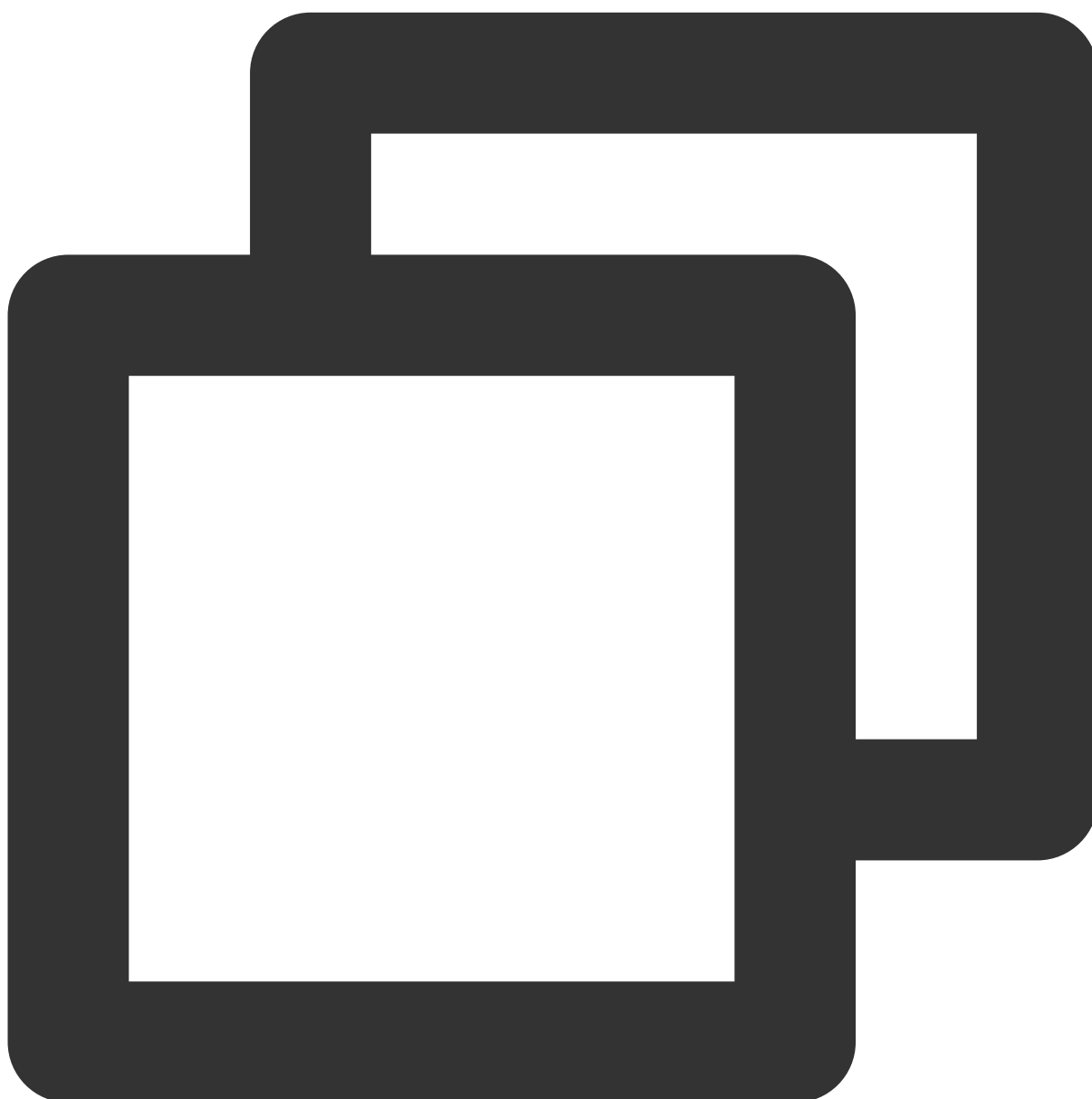
				according to userId
--	--	--	--	---------------------

**Returns** : *Promise*<[TUIUserInfo](#)> *userInfo*

This interface returns the user information of the specified user

### **setLocalVideoView**

Set the rendering position of the local stream.



```
const roomEngine = new TUIRoomEngine();
```



```
// Set the playback area of the local camera stream to the div element with id 'pre
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'preview-camera',
});

// Set the playback area of the local screen sharing stream to the div element with
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kScreenStream,
  view: 'preview-screen',
});
```

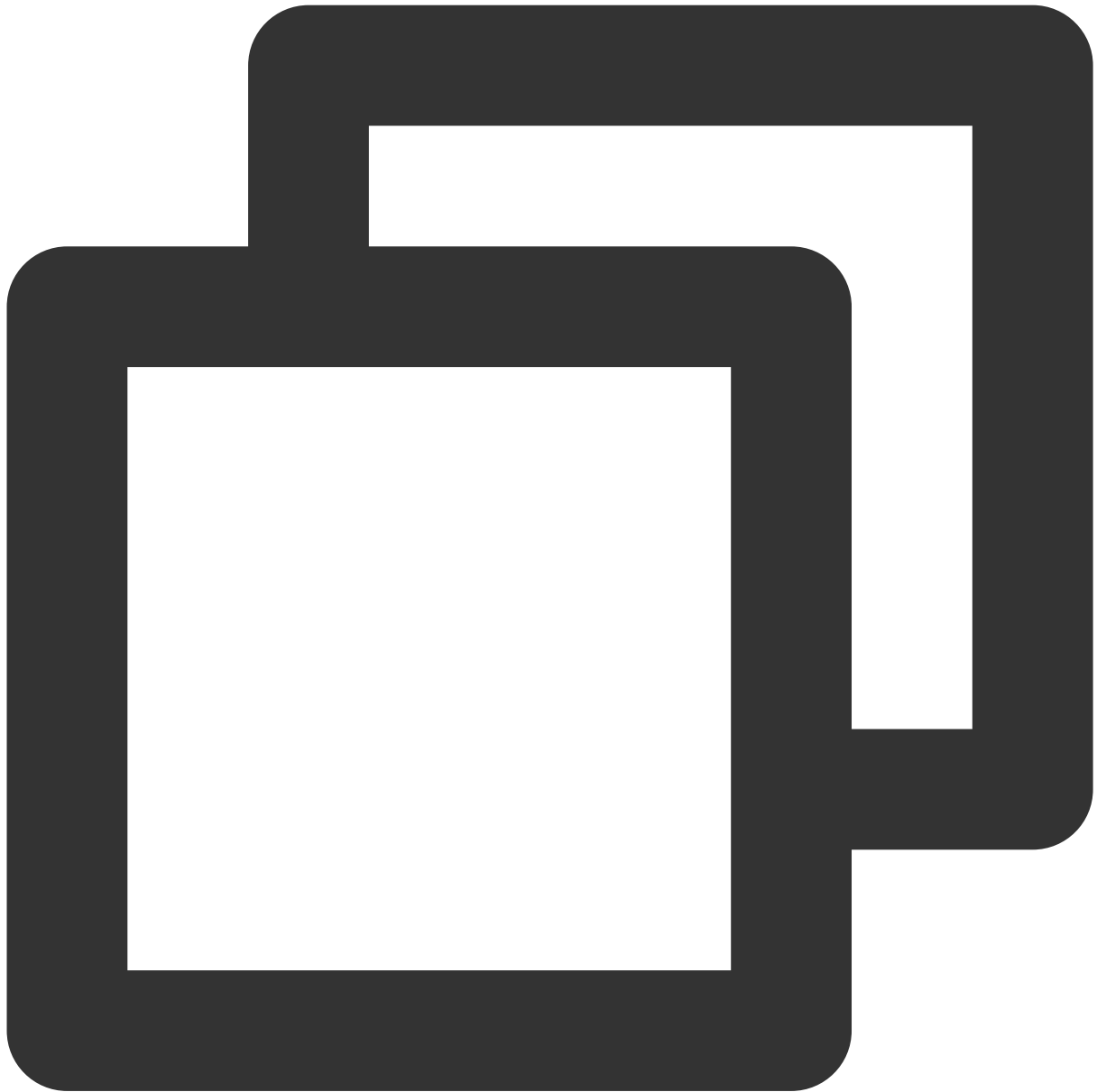
Parameter:

Parameter	Type	Description	Default Value	Meaning
streamType	<a href="#">TUIVideoStreamType</a>	Required	-	Local stream type
view	string	Required	-	The id of the div element corresponding to the streamType

Returns : *Promise<void>*

## openLocalCamera

Open the local camera and start capturing video streams.

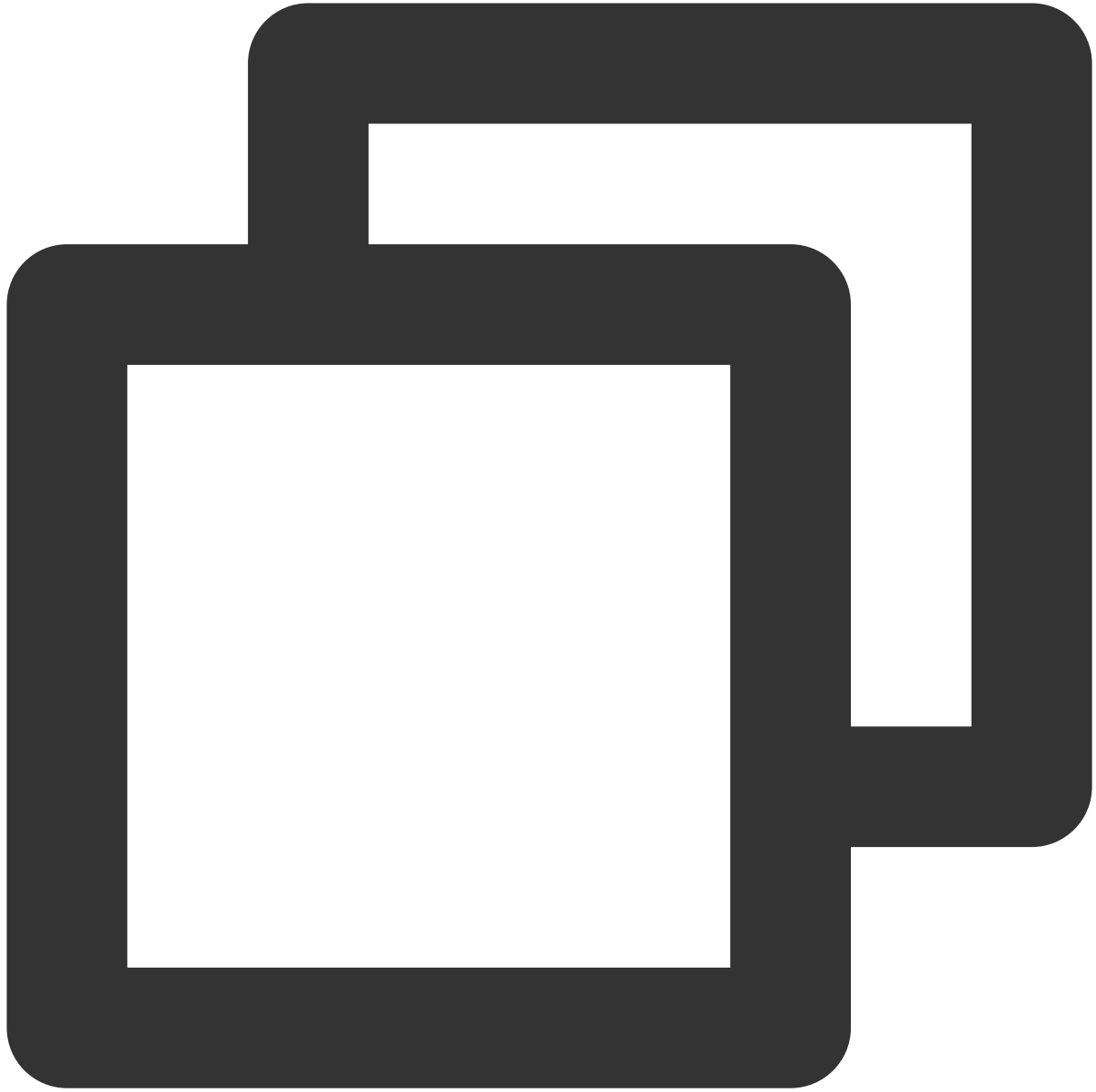


```
const roomEngine = new TUIRoomEngine();
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'preview-camera',
});
await roomEngine.openLocalCamera();
```

Returns : *Promise<void>*

### **closeLocalCamera**

Close the local camera.

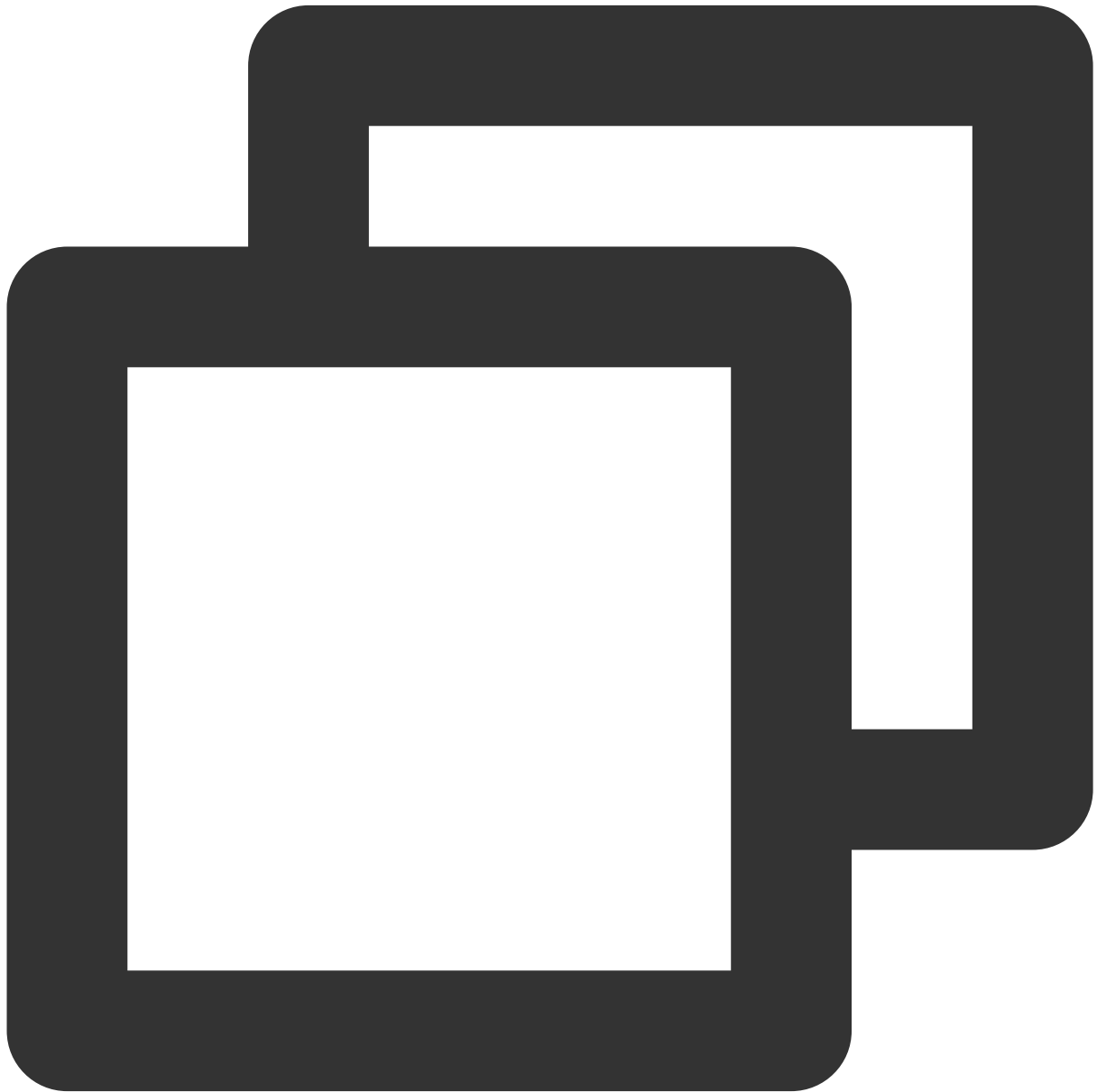


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.closeLocalCamera();
```

Returns : *Promise<void>*

### **openLocalMicrophone**

Open the local mic and start capturing audio streams.

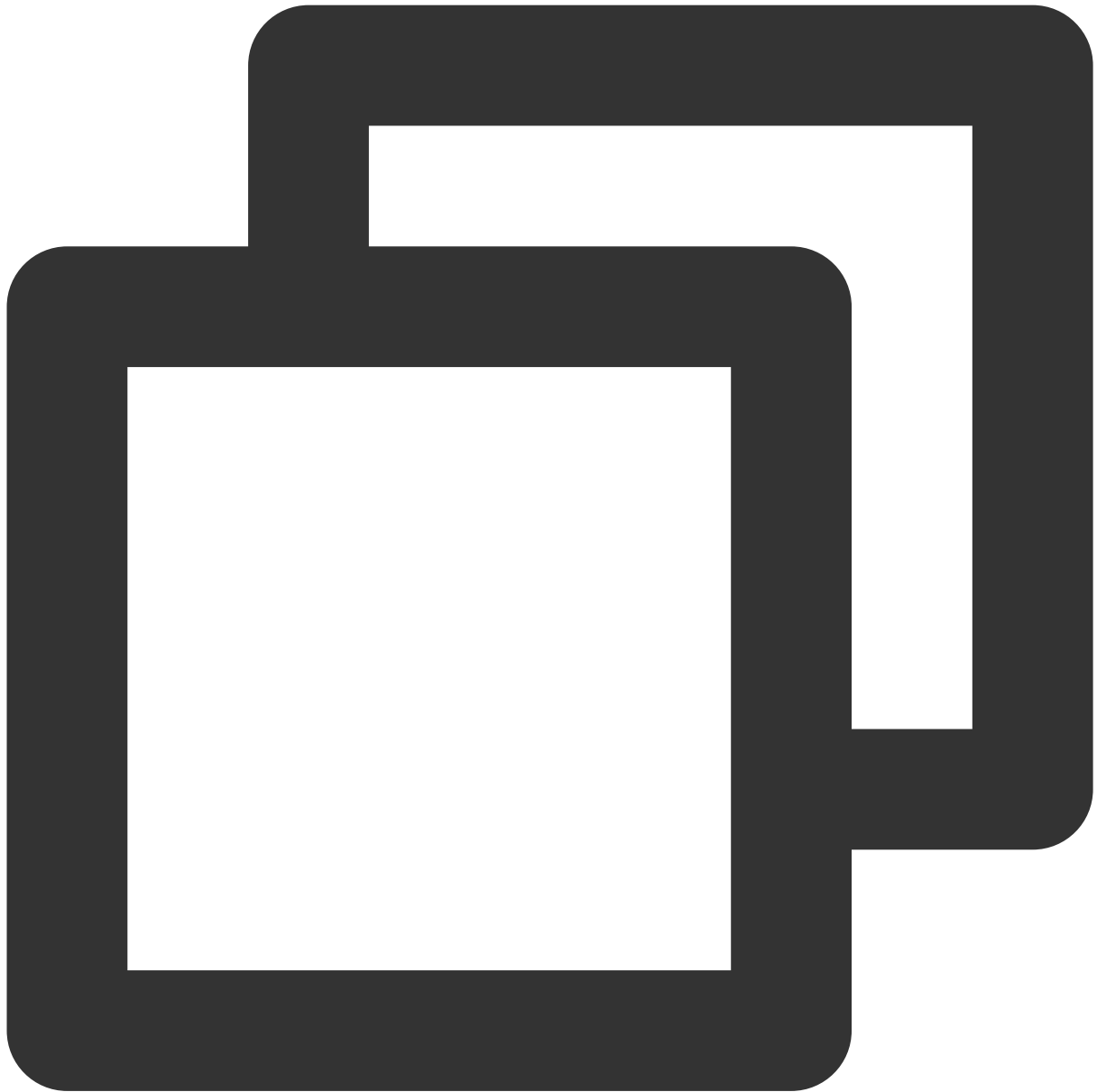


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.openLocalMicrophone();
```

Returns : *Promise<void>*

### **closeLocalMicrophone**

Close the local mic.

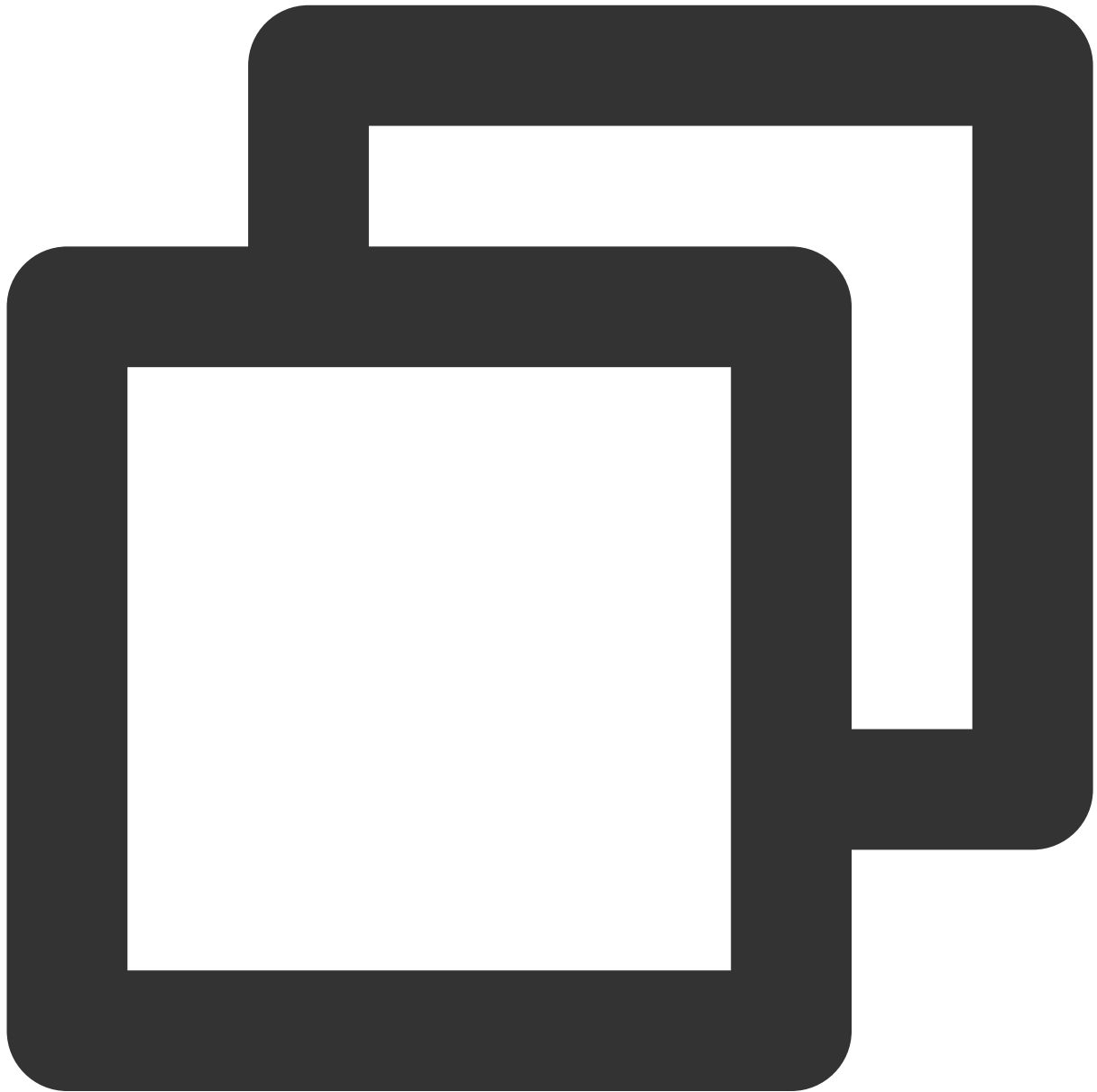


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.closeLocalMicrophone();
```

Returns : *Promise<void>*

### **updateVideoQuality**

Set the codec parameters of the local video stream, default is `TUIVideoProfile.kVideoQuality_720P`.



```
const roomEngine = new TUIRoomEngine();
await roomEngine.updateVideoQuality({
  quality: TUIVideoQuality.kVideoQuality_540p,
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
quality	<a href="#">TUIVideoQuality</a>	Required	-	Clear TUIVideoProfile.kVideoQuality_360P

				SD TUIVideoProfile.kVideoQuality_540P HD TUIVideoProfile.kVideoQuality_720P Full HD TUIVideoProfile.kVideoQuality_1080P
--	--	--	--	---

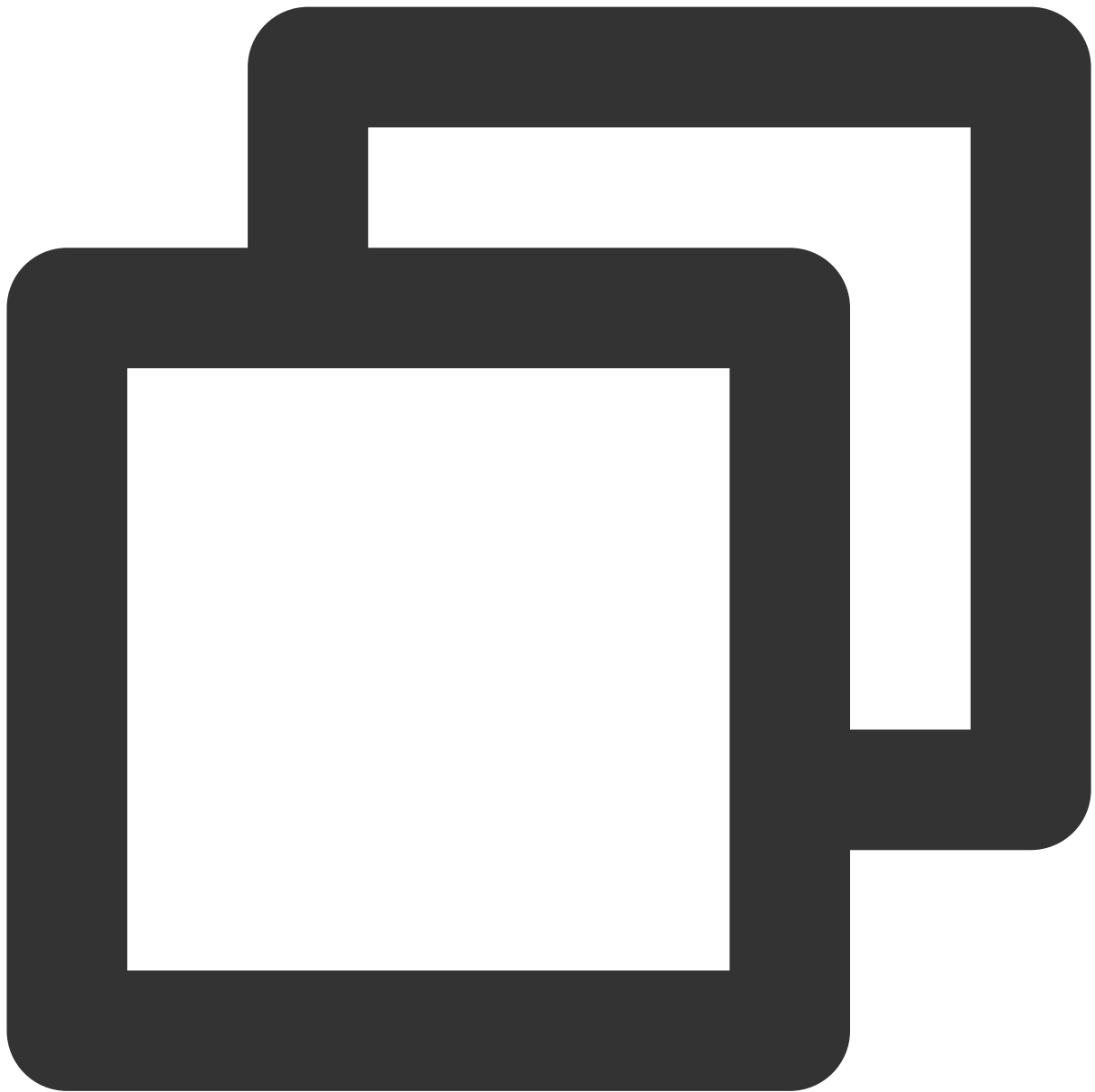
Returns : *Promise<void>*

## updateAudioQuality

Set Local Audio Parameters.

### Note:

This method needs to be set before openLocalMicrophone, otherwise it will not take effect.



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setLocalAudioProfile({
  audioProfile: TUIAudioProfile.kAudioProfileSpeech,
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
audioProfile	<a href="#">TUIAudioProfile</a>	Required	-	TUIAudioProfile.kAudioProfileSpeech: Speech

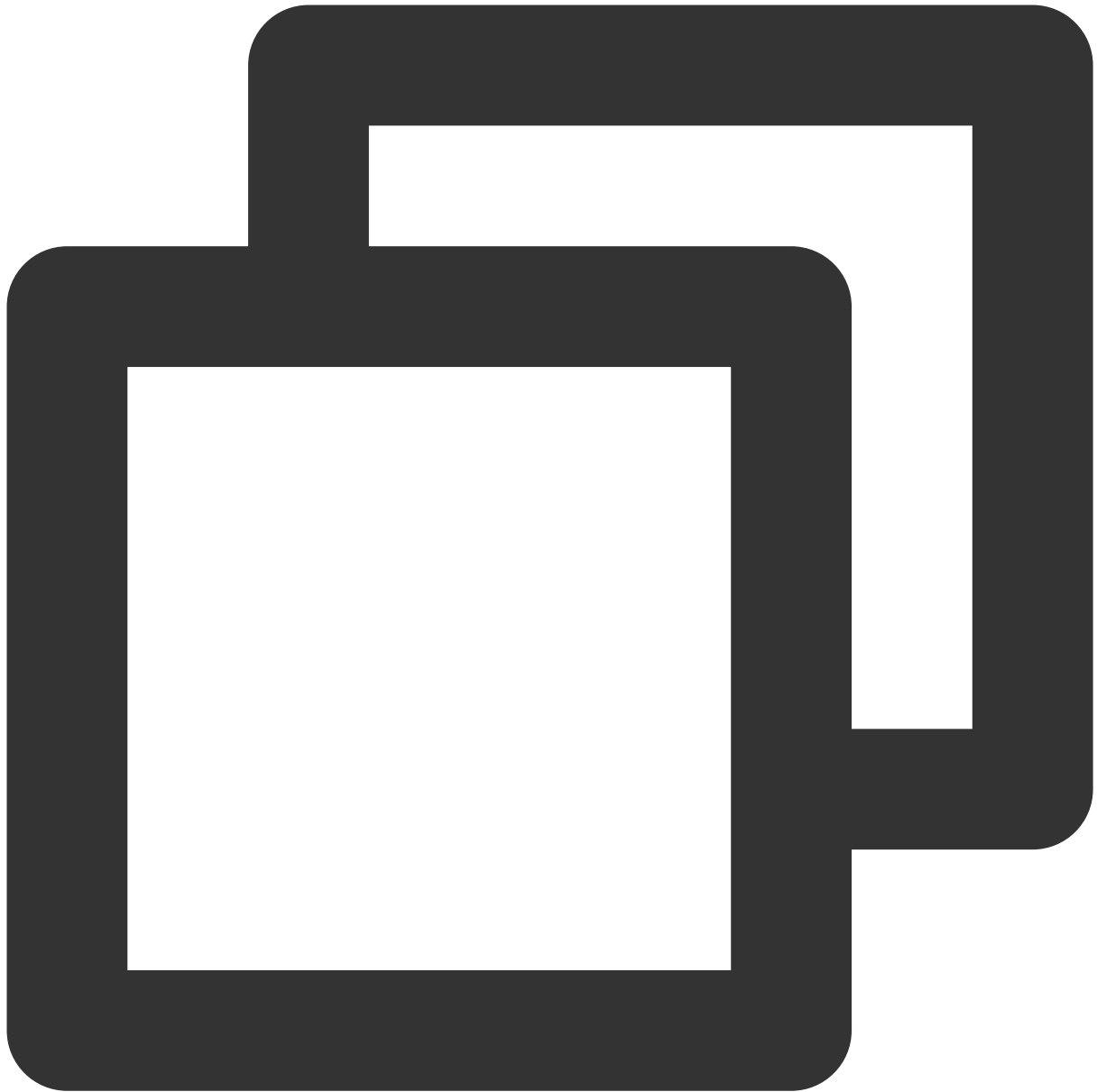


				Mode; Sample rate: 16k TUIAudioProfile.kAudioProfileDefault: Standard Mode (or Default Mode); Sample rate: 48k TUIAudioProfile.kAudioProfileMusic: Music Mode; Sample rate: 48k
--	--	--	--	--

Returns : *Promise<void>*

## **startPushLocalVideo**

After entering the room, the local video stream will be pushed to the remote by default. This interface is used to re-push the local video stream to the remote after stopping the push.

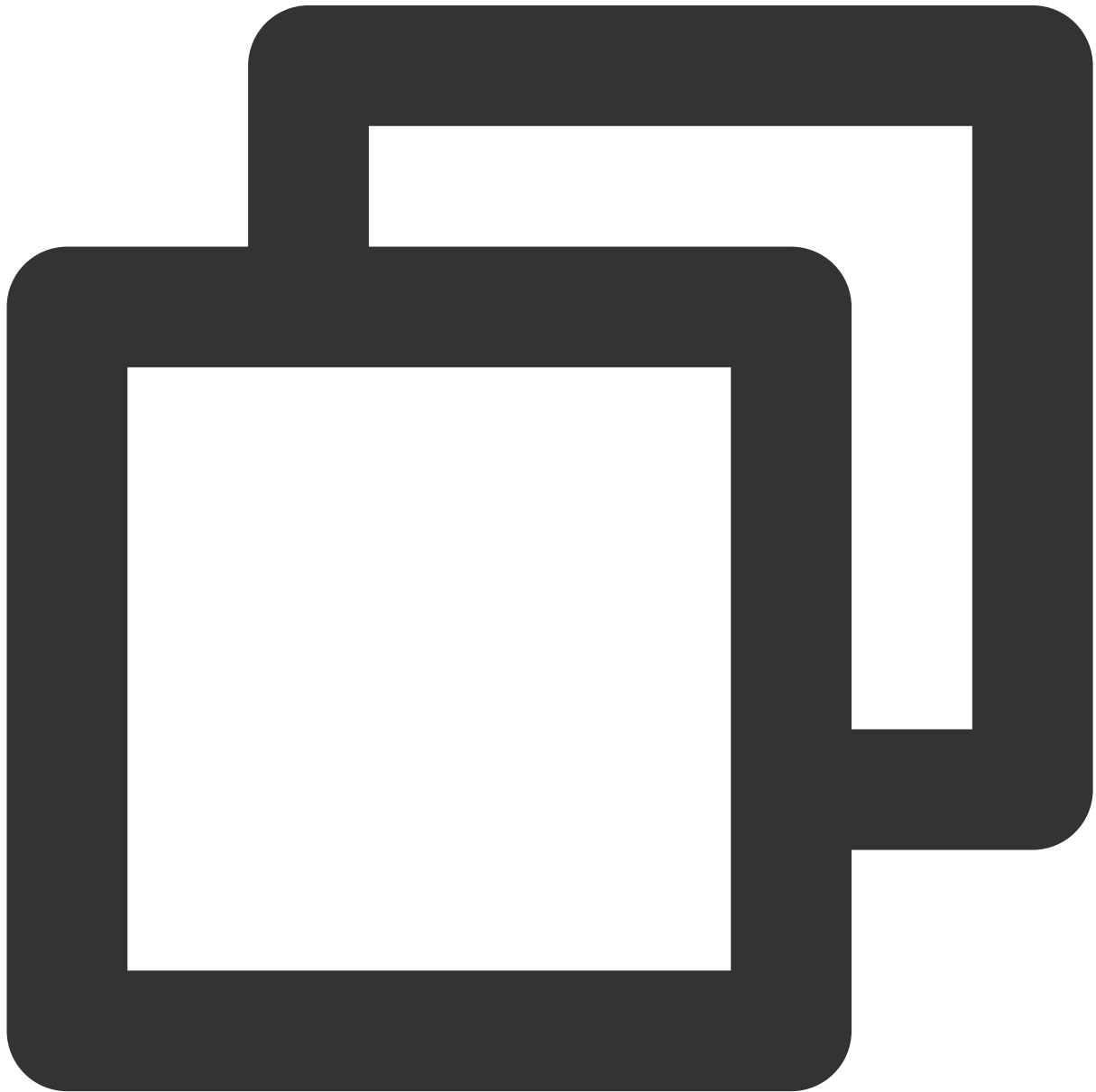


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.startPushLocalVideo();
```

Returns : *Promise<void>*

### **stopPushLocalVideo**

Stop Pushing Local Video Stream to Remote.



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.stopPushLocalVideo();
```

Returns : *Promise<void>*

### **startPushLocalAudio**

After entering the room, the local audio stream will be pushed to the remote by default. This interface is used to re-push the local audio stream to the remote after stopping the push.

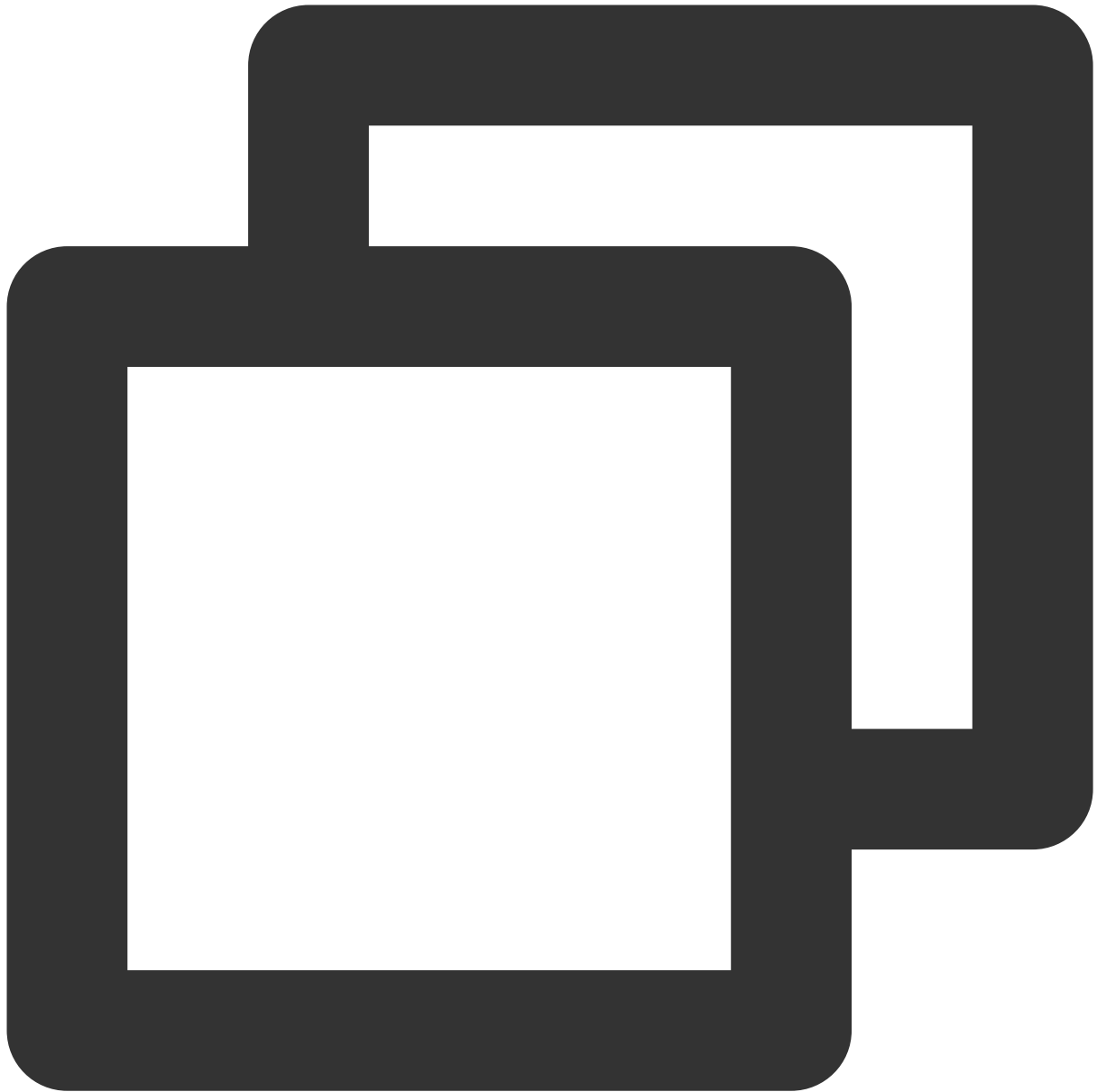


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.startPushLocalAudio();
```

Returns : *Promise<void>*

### **stopPushLocalAudio**

Stop Pushing Local Audio Stream to Remote.

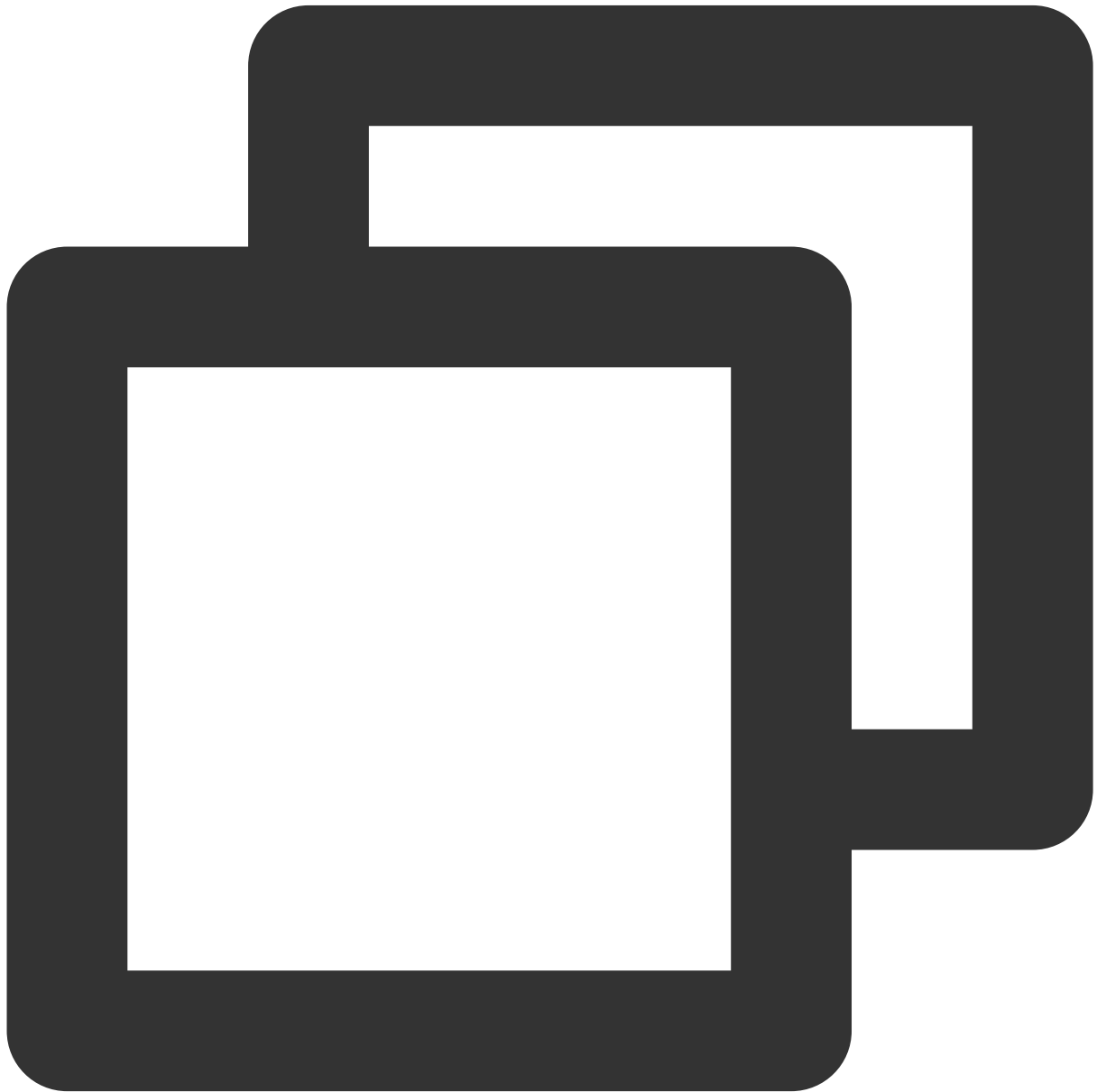


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.stopPushLocalAudio();
```

Returns : *Promise<void>*

### **setRemoteVideoView**

Set Remote Stream Rendering Area.



```
const roomEngine = new TUIRoomEngine();

// Set the remote user's video stream to play in the area with id 'remote_preview_c
await roomEngine.setRemoteVideoView({
  userId: 'user_1234',
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'remote_preview_camera',
});
// Set the remote user's screen sharing stream to play in the area with id 'remote_
await roomEngine.setRemoteVideoView({
  userId: 'user_1234',
```

```
streamType: TUIVideoStreamType.kScreenStream,  
view: 'remote_preview_screen',  
});
```

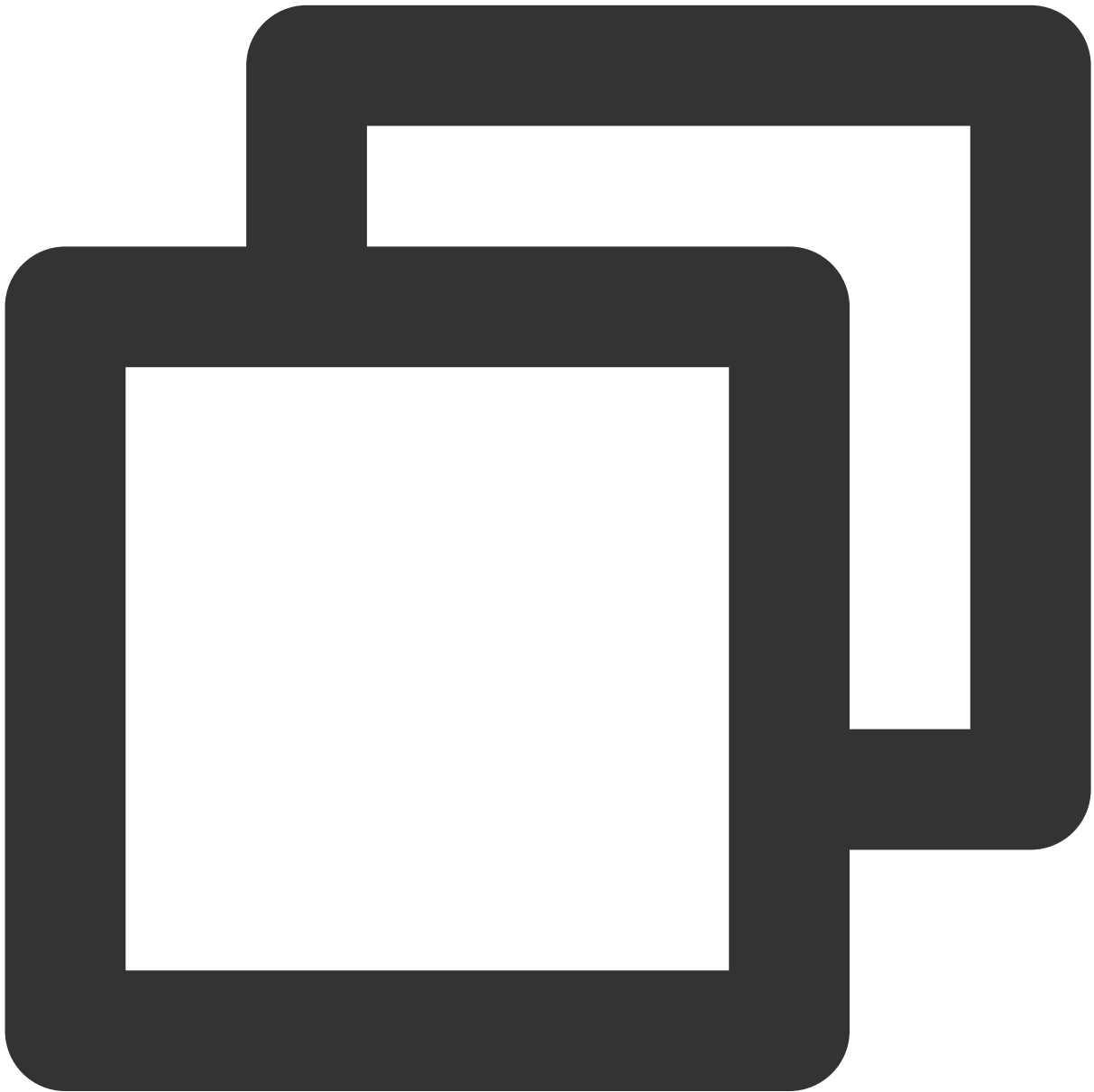
Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID
streamType	<a href="#">TUIVideoStreamType</a>	Required	-	User Stream Type
view	string	Required	-	The id of the div element playing the remote user's stream

Returns : *Promise<void>*

### **startPlayRemoteVideo**

Start Playback of Remote User Video Stream.



```
const roomEngine = new TUIRoomEngine();
await roomEngine.startPlayRemoteVideo({
  userId: 'user_1234',
  streamType: TUIVideoStreamType.kCameraStream,
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning

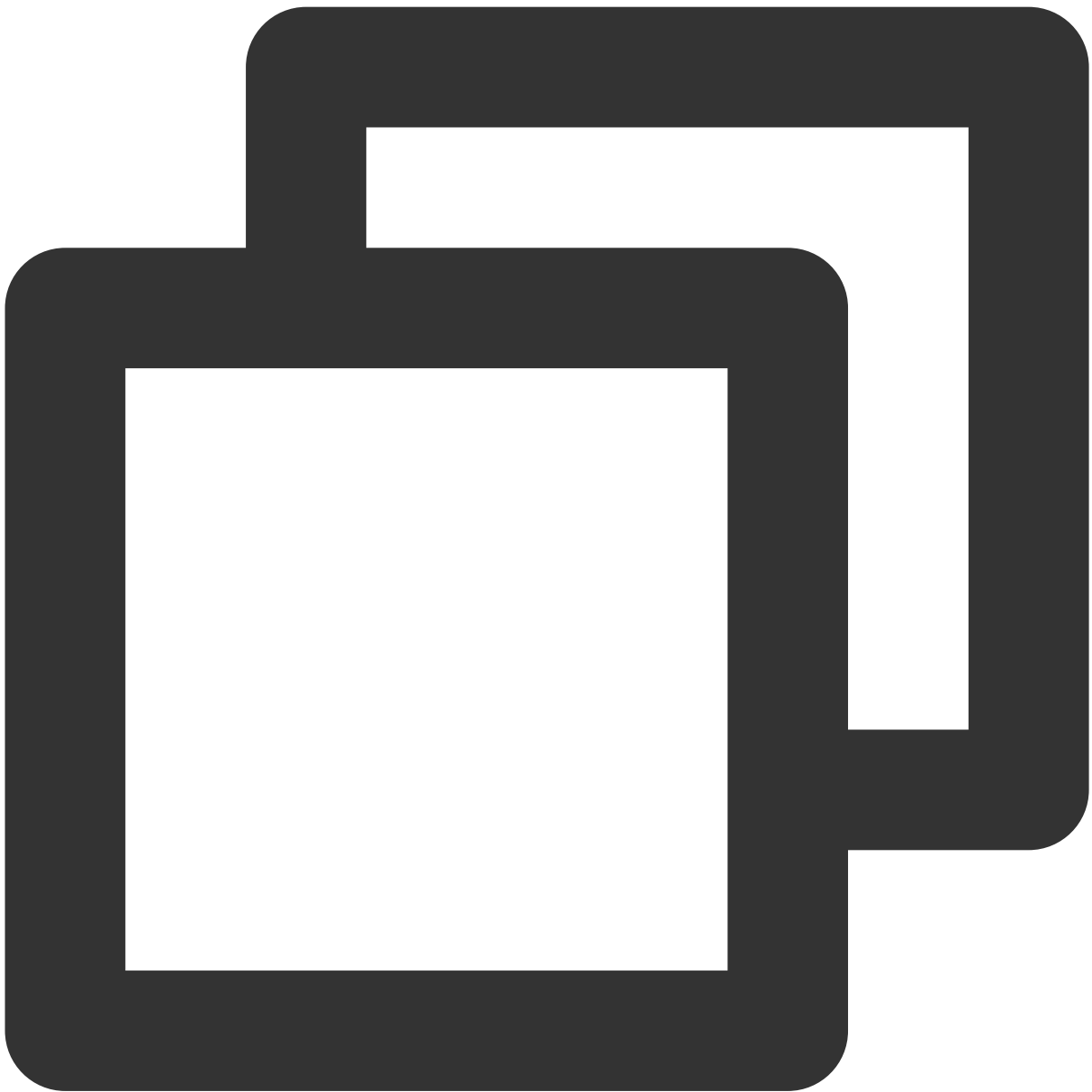


userId	string	Required	-	User ID
streamType	<a href="#">TUIVideoStreamType</a>	Required	-	User Stream Type TUIVideoStreamType.kCameraStream Video Stream TUIVideoStreamType.kScreenStream Screen Sharing Stream TUIVideoStreamType.kCameraStreamLow Low Definition Video Stream

Returns : *Promise<void>*

## stopPlayRemoteVideo

Stop Playback of Remote User Video Stream.



```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopPlayRemoteVideo({
  userId: 'user_1234',
  streamType: TUIVideoStreamType.kCameraStream,
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning

userId	string	Required	-	User ID
streamType	<a href="#">TUIVideoStreamType</a>	Required	-	User Stream Type TUIVideoStreamType.kCameraStream Video Stream TUIVideoStreamType.kScreenStream Screen Sharing Stream TUIVideoStreamType.kCameraStreamLow Low Definition Video Stream

Returns : *Promise<void>*

### **muteRemoteAudioStream**

Stop Remote User's Audio Stream.

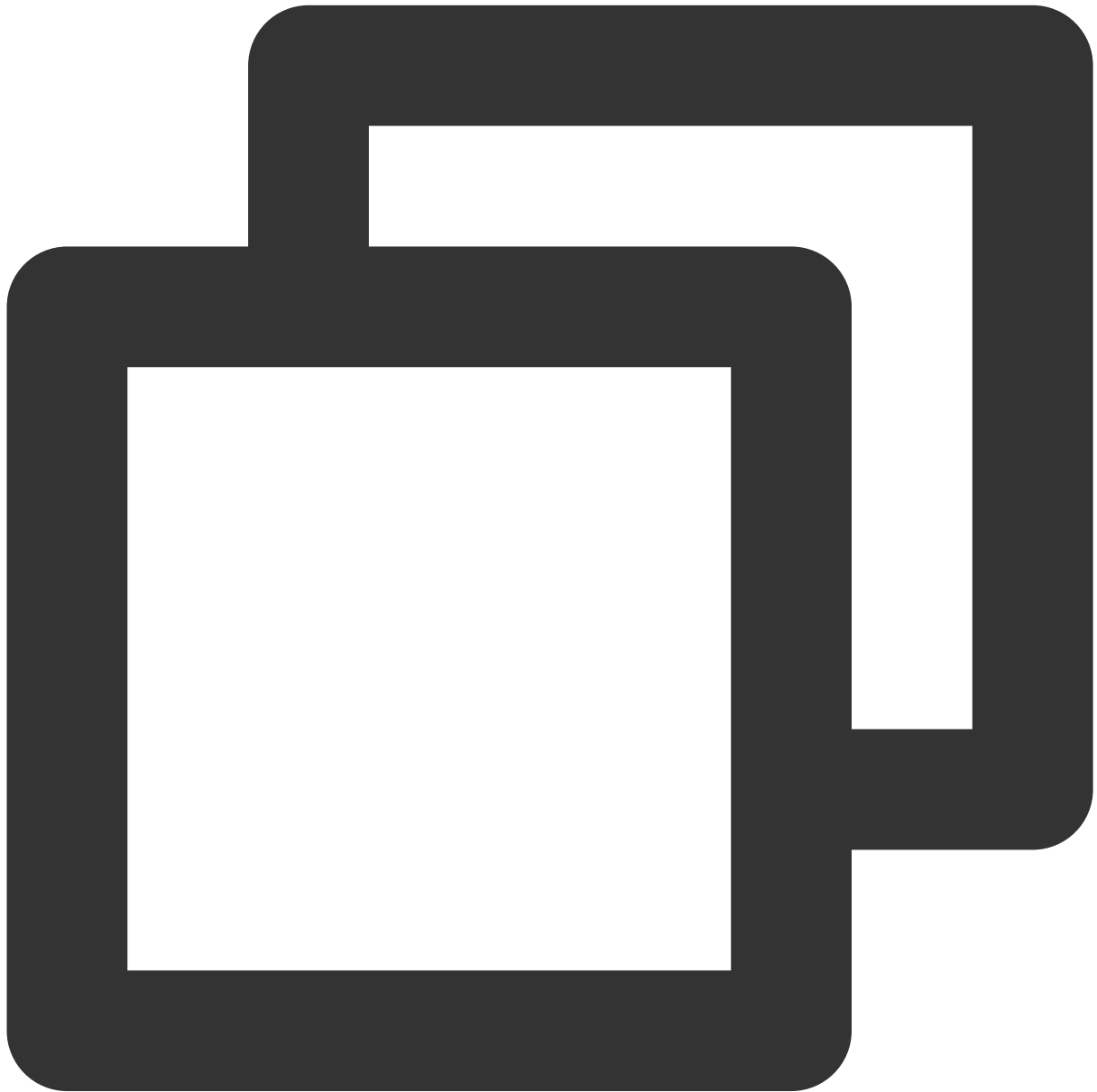


```
const roomEngine = new TUIRoomEngine();
await roomEngine.muteRemoteAudioStream({
  userId: 'user_1234',
  isMute: true,
});
```

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID
isMute	boolean	Required	-	Whether to Stop Remote User's Audio

## openRemoteDeviceByAdmin

Request Remote User to Open Media Device.



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.openRemoteDeviceByAdmin({
  userId: 'user_1234',
  device: TUIMediaDevice.kMicrophone //The requested device is a mic
  timeout: 0,
  requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
```

```

switch (requestCallbackType) {
    case TUIRequestCallbackType.kRequestAccepted:
        // Request Accepted
        break;
    case TUIRequestCallbackType.kRequestRejected:
        // Request Rejected
        break;
    case TUIRequestCallbackType.kRequestCancelled:
        // Request Canceled
        break;
    case TUIRequestCallbackType.kRequestTimeout:
        // Request Timeout
        break;
    case TUIRequestCallbackType.kRequestError:
        // Request Error
        break;
    default:
        break;
}
},
});

```

Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID
device	<a href="#">TUIMediaDevice</a>	Required	-	Media Device Type (Camera/Mic/Screen Sharing)
timeout	number	Required	-	Timeout Time. If timeout is set to 0, there is no timeout time
requestCallback	Function	Optional	Empty Function	Request Callback, used to notify the initiator of the request being accepted/rejected/canceled/timeout/error

Returns : *Promise<string> requestId*

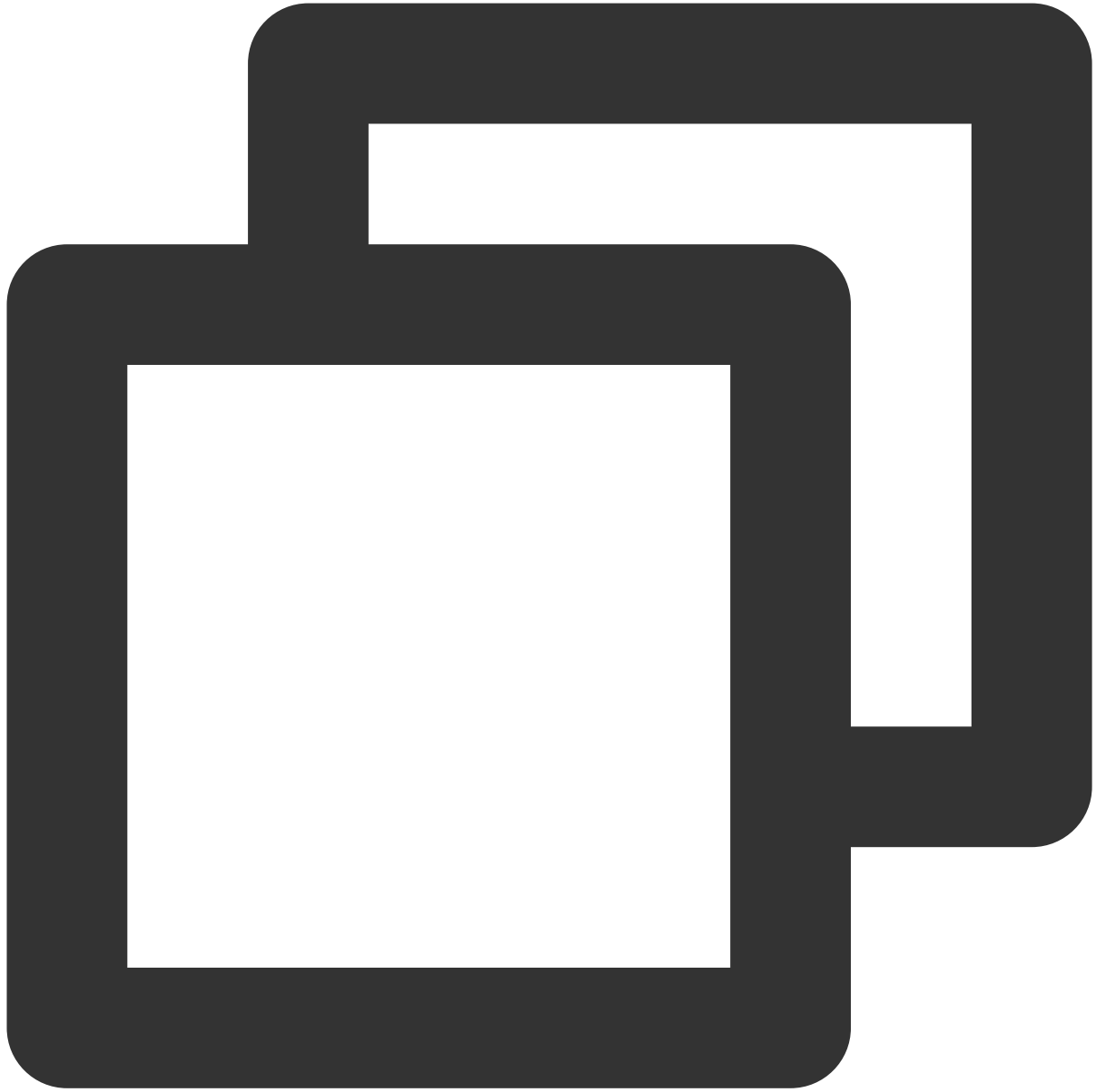
This interface returns requestId, users can use this requestId to call cancelRequest interface to cancel the request.

#### Description:

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number;

### applyToAdminToOpenLocalDevice

Participant applies to the host to open the device.



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.applyToAdminToOpenLocalDevice({
  device: TUIMediaDevice.kMicrophone //The requested device is a mic
  timeout: 0,
  requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
    switch (requestCallbackType) {
      case TUIRequestCallbackType.kRequestAccepted:
        // Request Accepted
        break;
    }
  }
});
```

```
    case TUIRequestCallbackType.kRequestRejected:
        // Request Rejected
        break;
    case TUIRequestCallbackType.kRequestCancelled:
        // Request Canceled
        break;
    case TUIRequestCallbackType.kRequestTimeout:
        // Request Timeout
        break;
    case TUIRequestCallbackType.kRequestError:
        // Request Error
        break;
    default:
        break;
}
},
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
device	<a href="#">TUIMediaDevice</a>	Required	-	Media Device Type (Camera/Mic/Screen Sharing)
timeout	number	Required	-	Timeout Time. If timeout is set to 0, there is no timeout time
requestCallback	Function	Optional	Empty Function	Request Callback, used to notify the initiator of the request being accepted/rejected/canceled/timeout/error

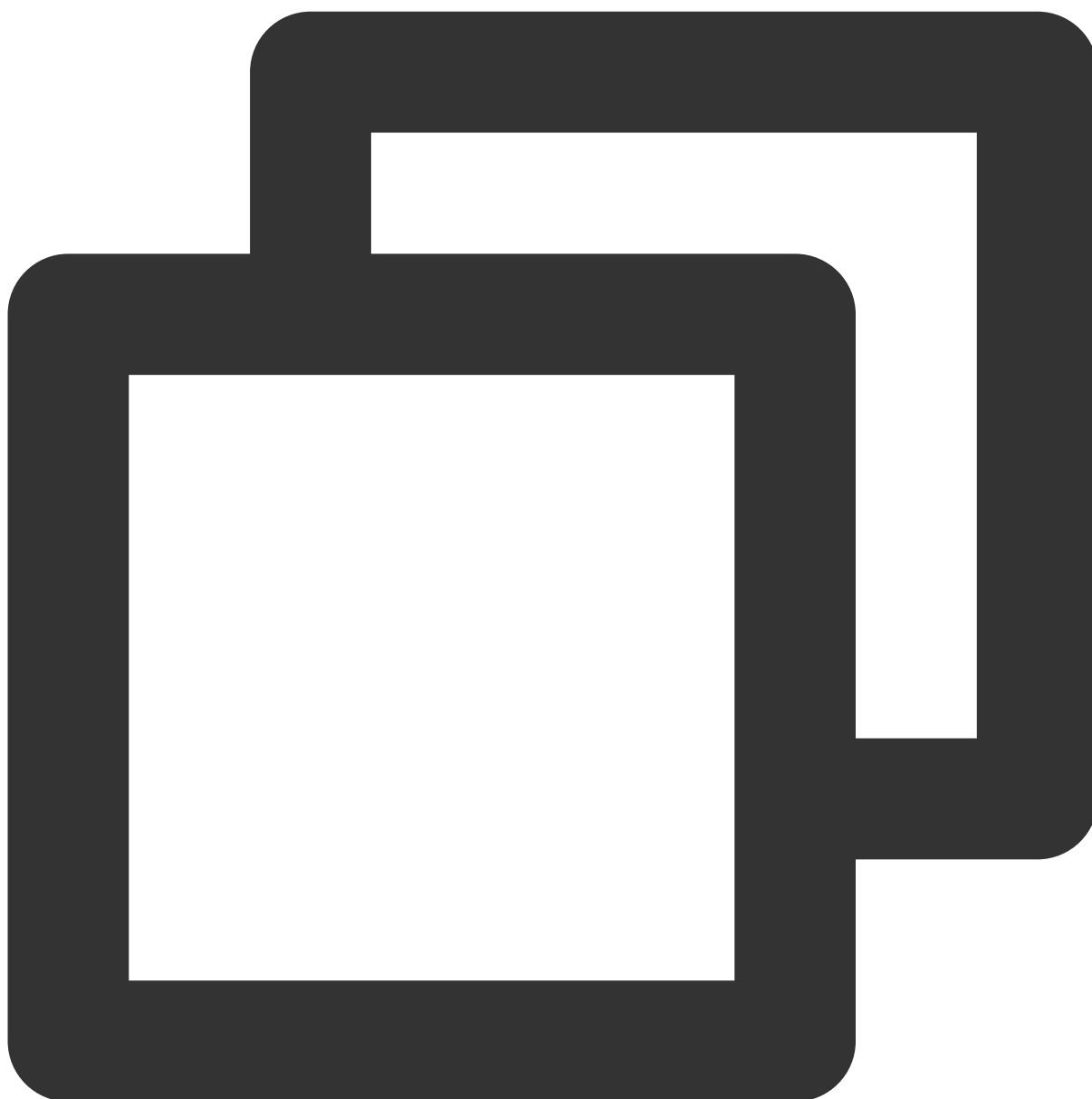
Returns : *Promise<string> requestId*

This interface returns requestId, users can use this requestId to call cancelRequest interface to cancel the request.

## closeRemoteDeviceByAdmin

Close Remote User Media Device.





```
const roomEngine = new TUIRoomEngine();
await roomEngine.closeRemoteDeviceByAdmin({
  userId: 'user_1234',
  device: TUIMediaDevice.kMicrophone, //Close mic
});
```

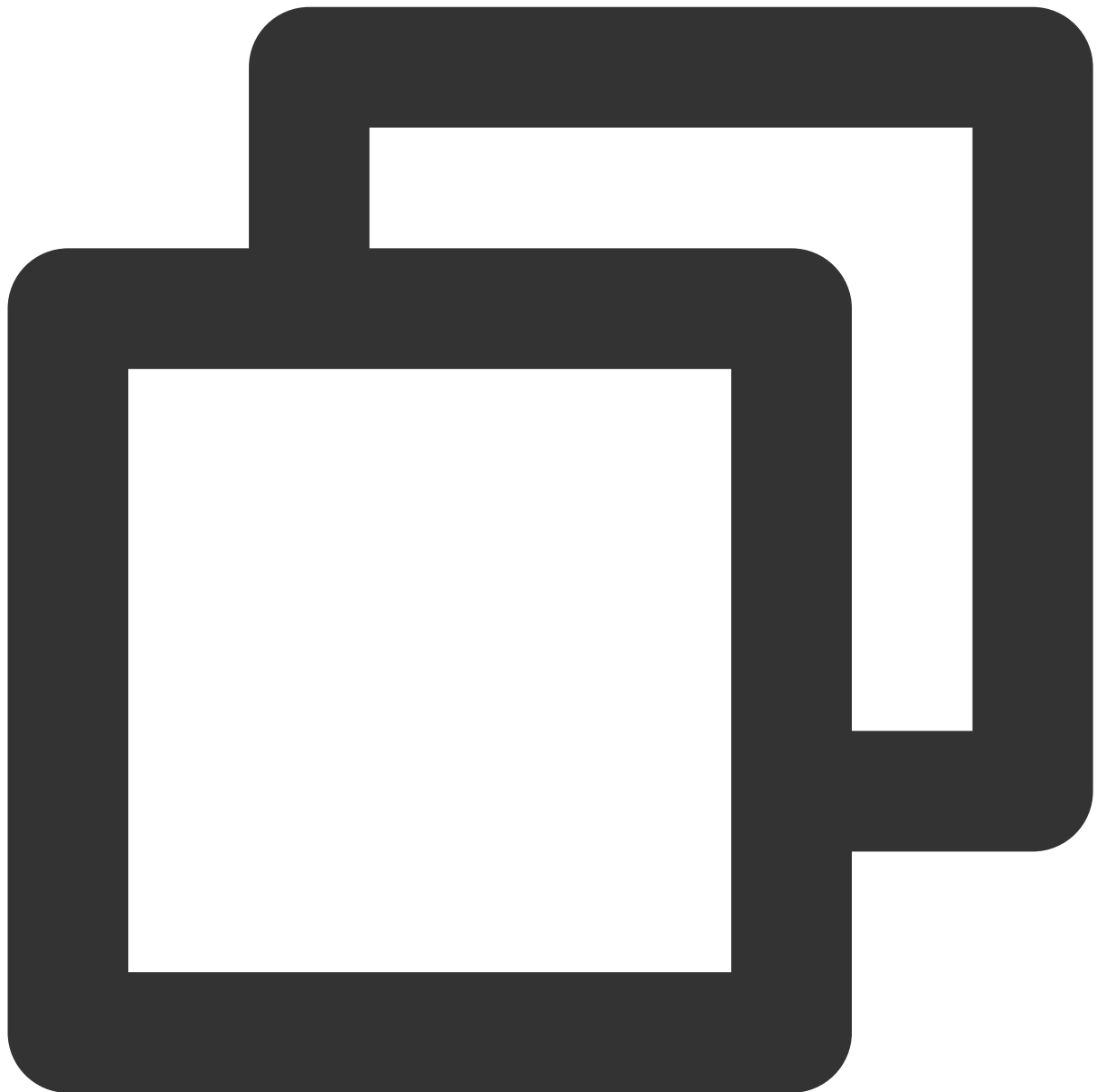
Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID

device	<a href="#">TUIMediaDevice</a>	Required	-	Media Device Type (Camera/Mic/Screen Sharing)
--------	--------------------------------	----------	---	--

## cancelRequest

Cancel Already Sent Request.



```
const roomEngine = new TUIRoomEngine();
await roomEngine.cancelRequest({
  requestId: '',    // Please use Actual requestId
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
requestId	string	Required	-	Request ID

**Returns** : *Promise<string>* requestId

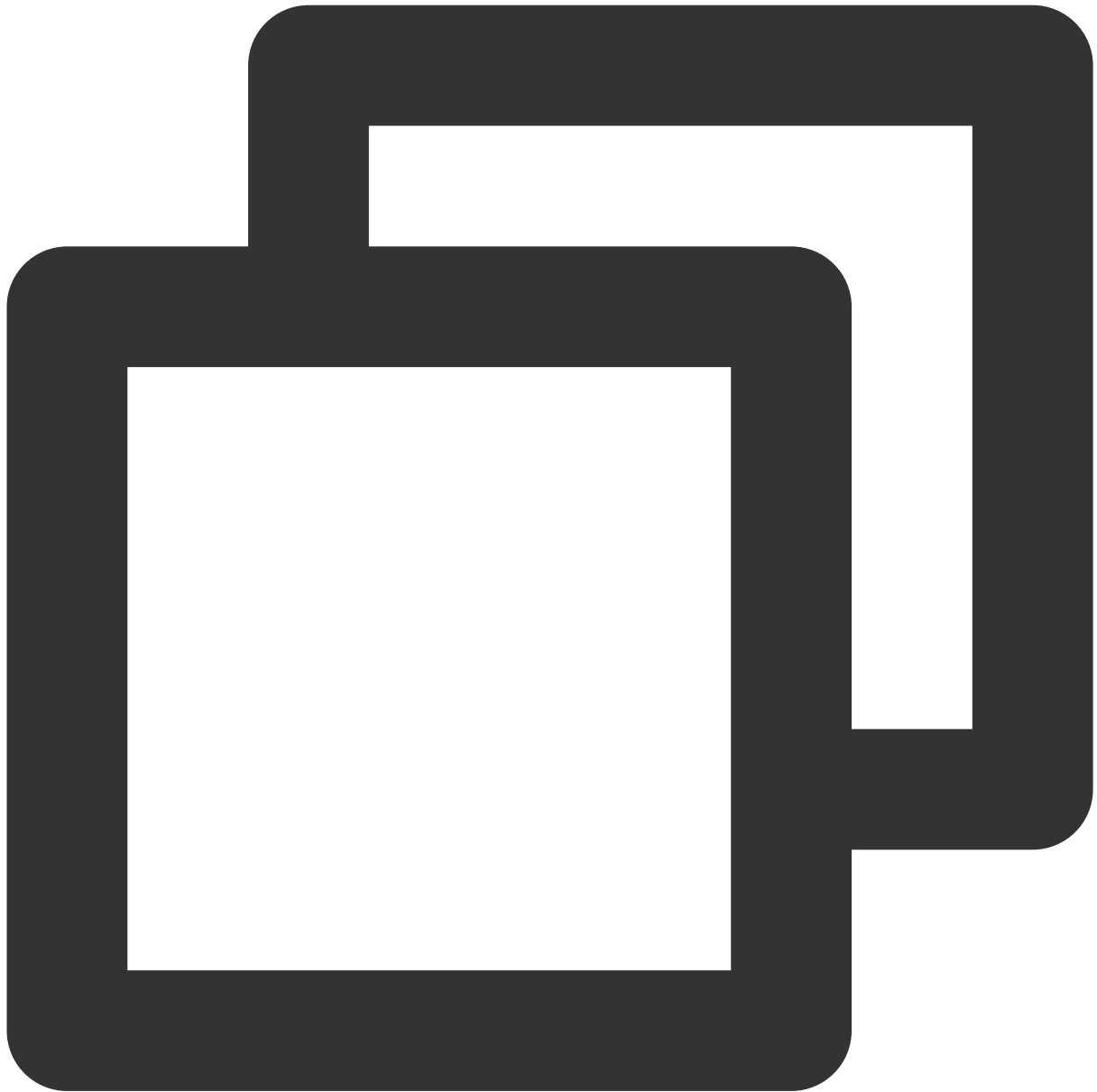
This interface returns requestId, users can use this requestId to call cancelRequest interface to cancel the request

**Note:**

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number;

## responseRemoteRequest

Reply to Remote User's Request.



```
const roomEngine = new TUIRoomEngine();
// Agree to Remote User's Request
await roomEngine.responseRemoteRequest({
  requestId: '',    // Please use Actual requestId
  agree: true,
});
// Reject Remote User's Request
await roomEngine.responseRemoteRequest({
  requestId: '',    // Please use Actual requestId
  agree: false,
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
requestId	string	Required	-	Request ID
agree	boolean	Required	-	Whether to Agree

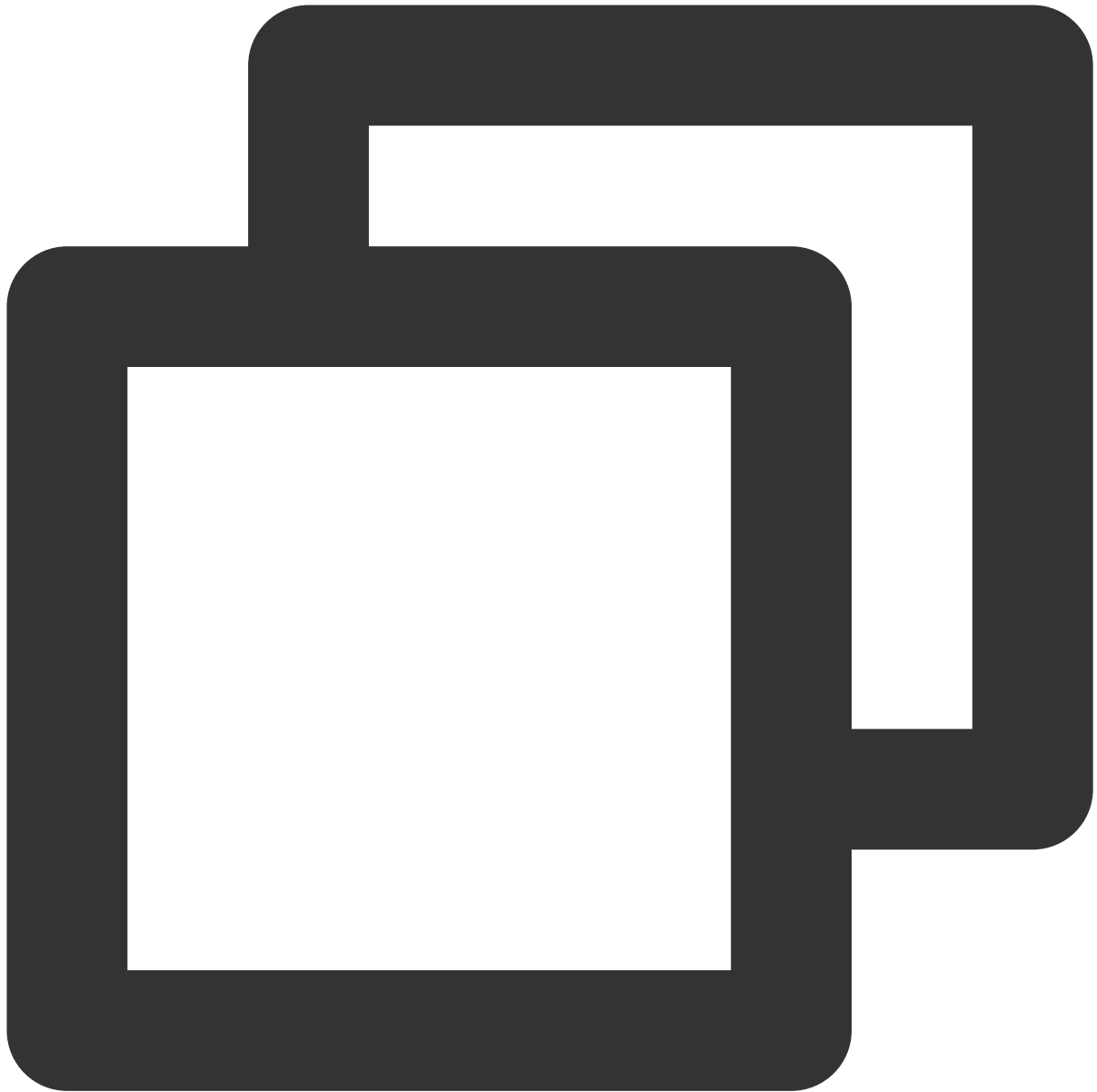
**Returns** : *Promise<void>*

**Note** :

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number;

### **disableDeviceForAllUserByAdmin**

Prohibit/Allow All Users to Open Media Device (This Interface is invalid for Room Owner and Administrator).



```
// Example 1: Prohibit All Users to Open Mic
await roomEngine.disableDeviceForAllUserByAdmin({
  device: TUIMediaDevice.kMicrophone,
  isDisable: true,
})
// Example 2: Allow All Users to Open Mic
await roomEngine.disableDeviceForAllUserByAdmin({
  device: TUIMediaDevice.kMicrophone,
  isDisable: false,
})
```

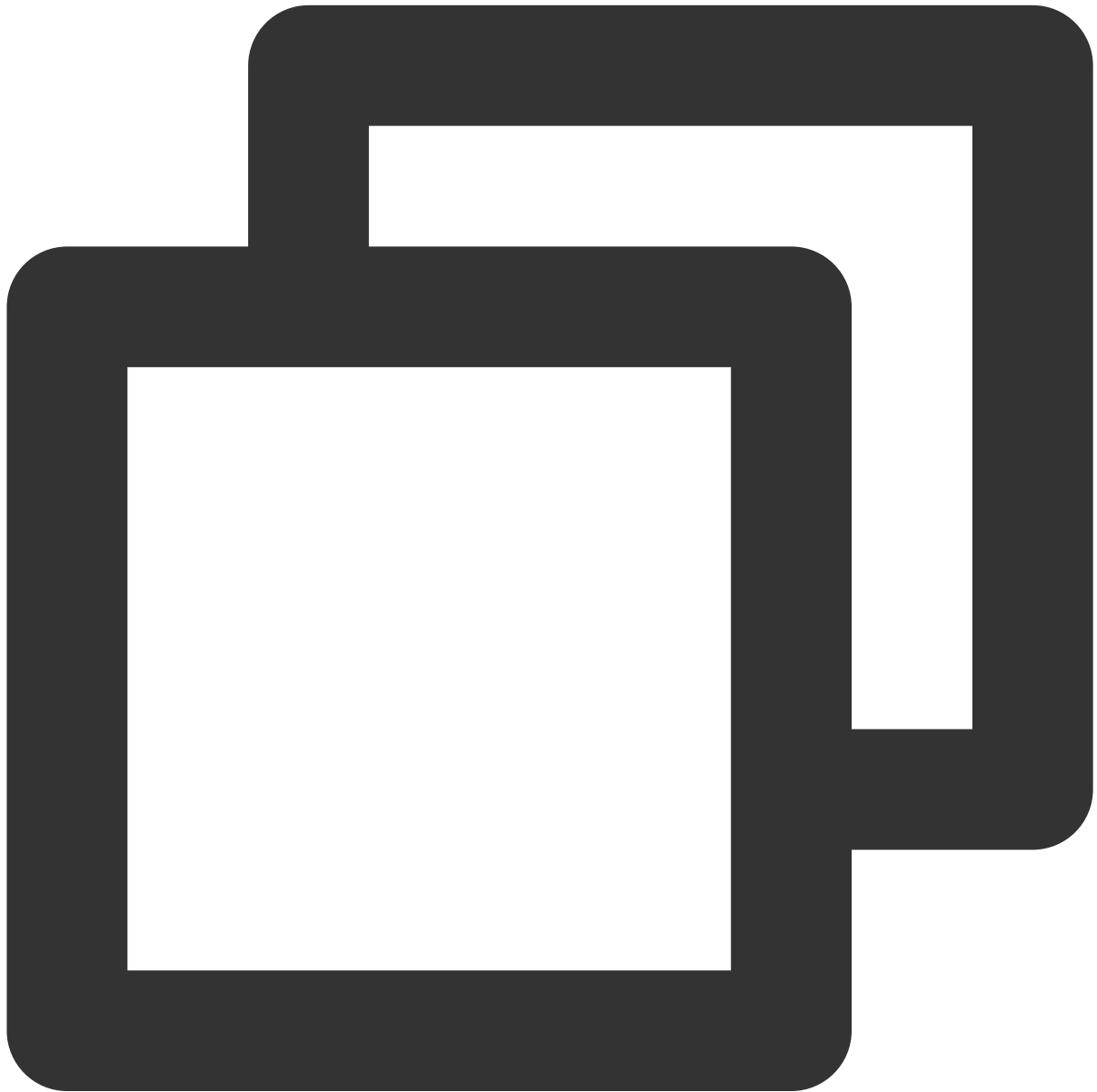
Parameter:

Parameter	Type	Description	Default Value	Meaning
device	<a href="#">TUIMediaDevice</a>	Required	-	Disabled Media Device Type (Camera/Mic/Screen Sharing)
isDisable	boolean	Required	-	Whether it is Prohibited

Returns : *Promise<void>*

### **disableSendingMessageForAllUser**

Whether All Users are Allowed to Send Messages (This Interface is invalid for Room Owner and Administrator).



```
await roomEngine.disableSendingMessageForAllUser({
  isDisable: true,
});
```

Parameter:

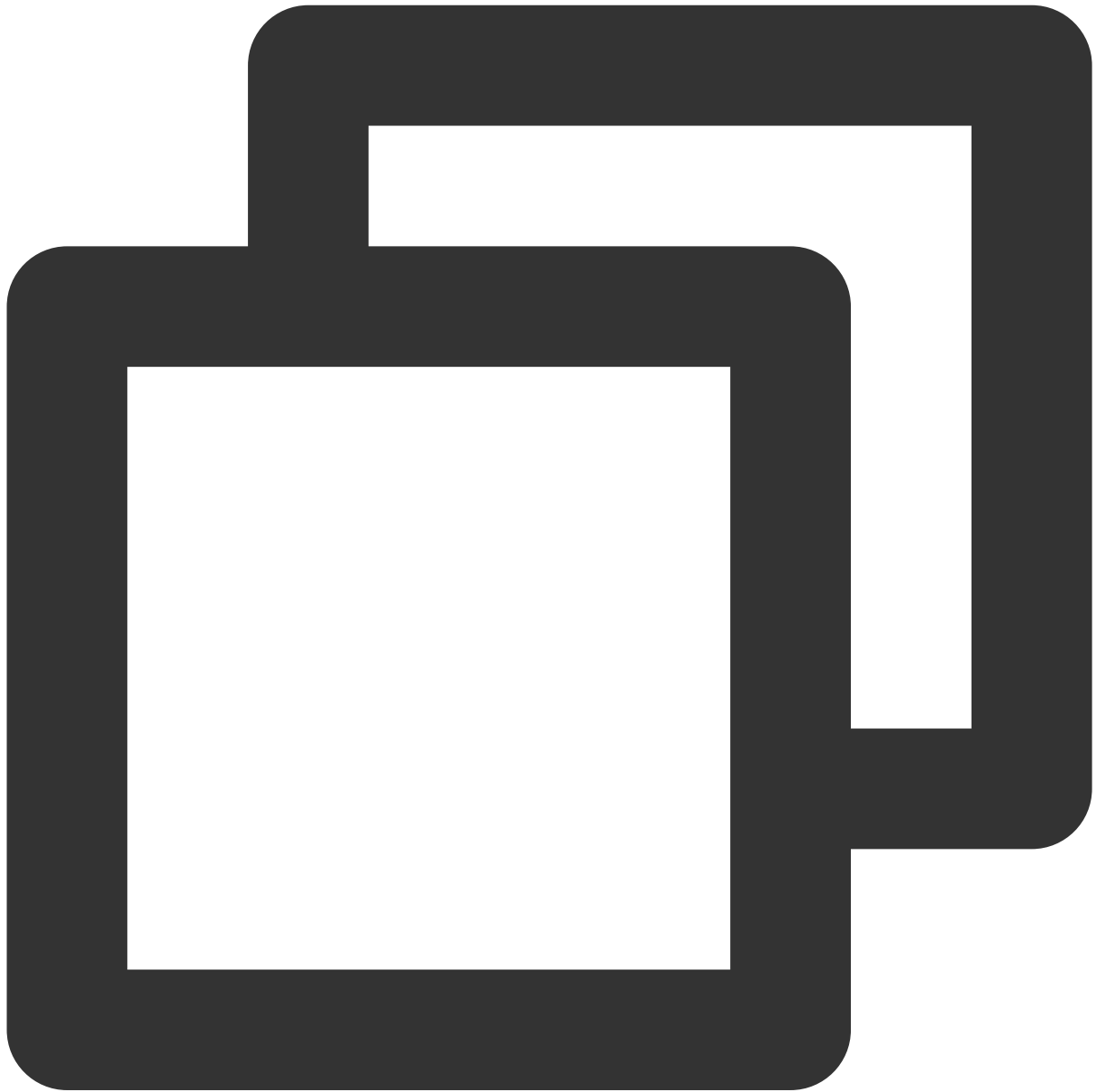
Parameter	Type	Description	Default Value	Meaning
isDisable	boolean	Required	-	Whether it is Disabled



Returns : *Promise<void>*

### **disableSendingMessageByAdmin**

Whether Specific User is Allowed to Send Messages.



```
await roomEngine.disableSendingMessageByAdmin({
  userId: 'user_1234',
  isDisable: true,
});
```

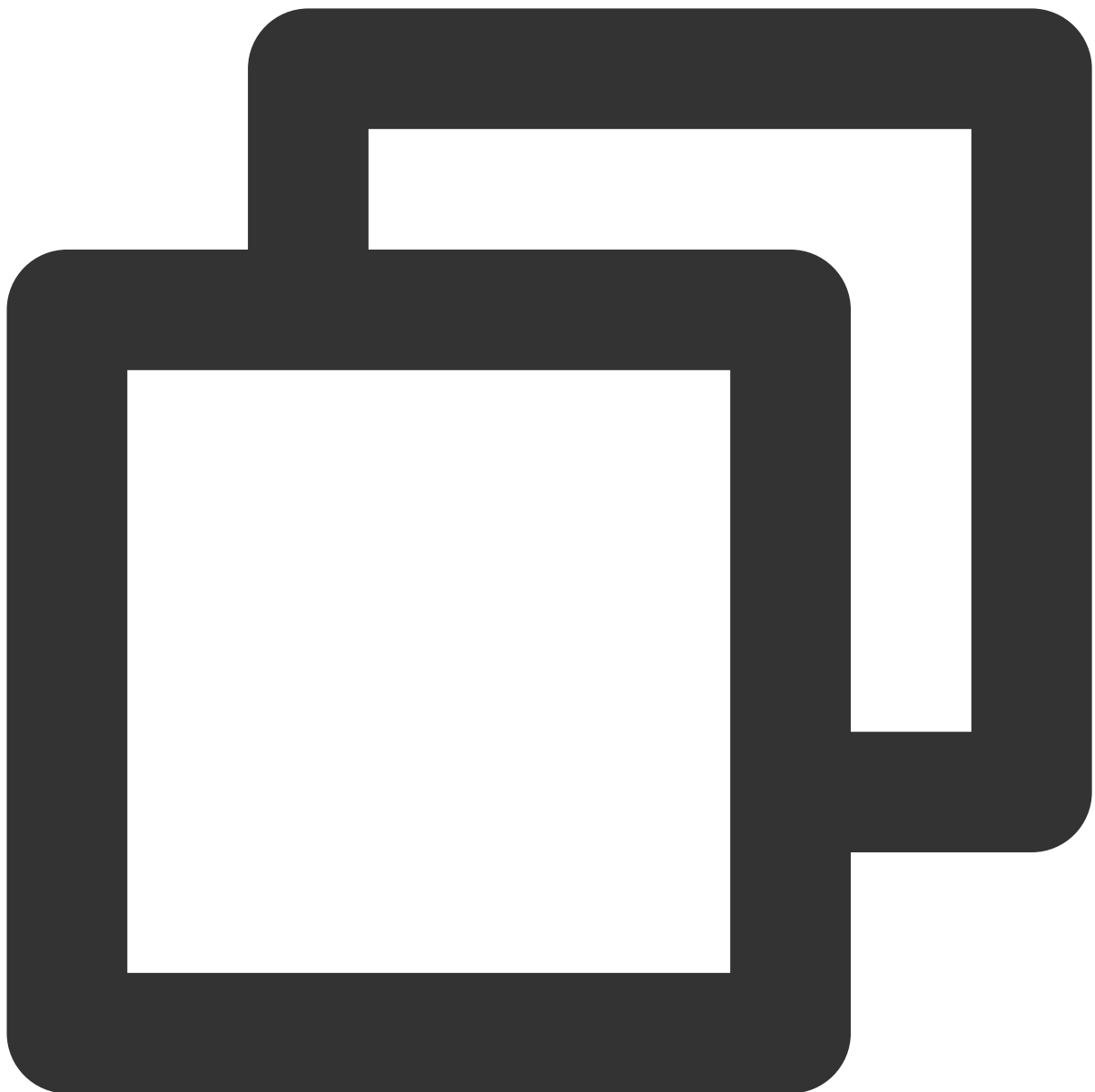
Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID
isDisable	boolean	Required	-	Whether it is Disabled

Returns : *Promise<void>*

### changeUserRole

Change User's Role (Only the Host can call this Interface).



```
const roomEngine = new TUIRoomEngine();

// Transfer the Room to User user_1234
await roomEngine.changeUserRole({
  userId: 'user_1234',
  role: TUIRole.kRoomOwner,
});

// Set user_1234 as Room Administrator
await roomEngine.changeUserRole({
  userId: 'user_1234',
  userRole: TUIRole.kAdministrator,
});
```

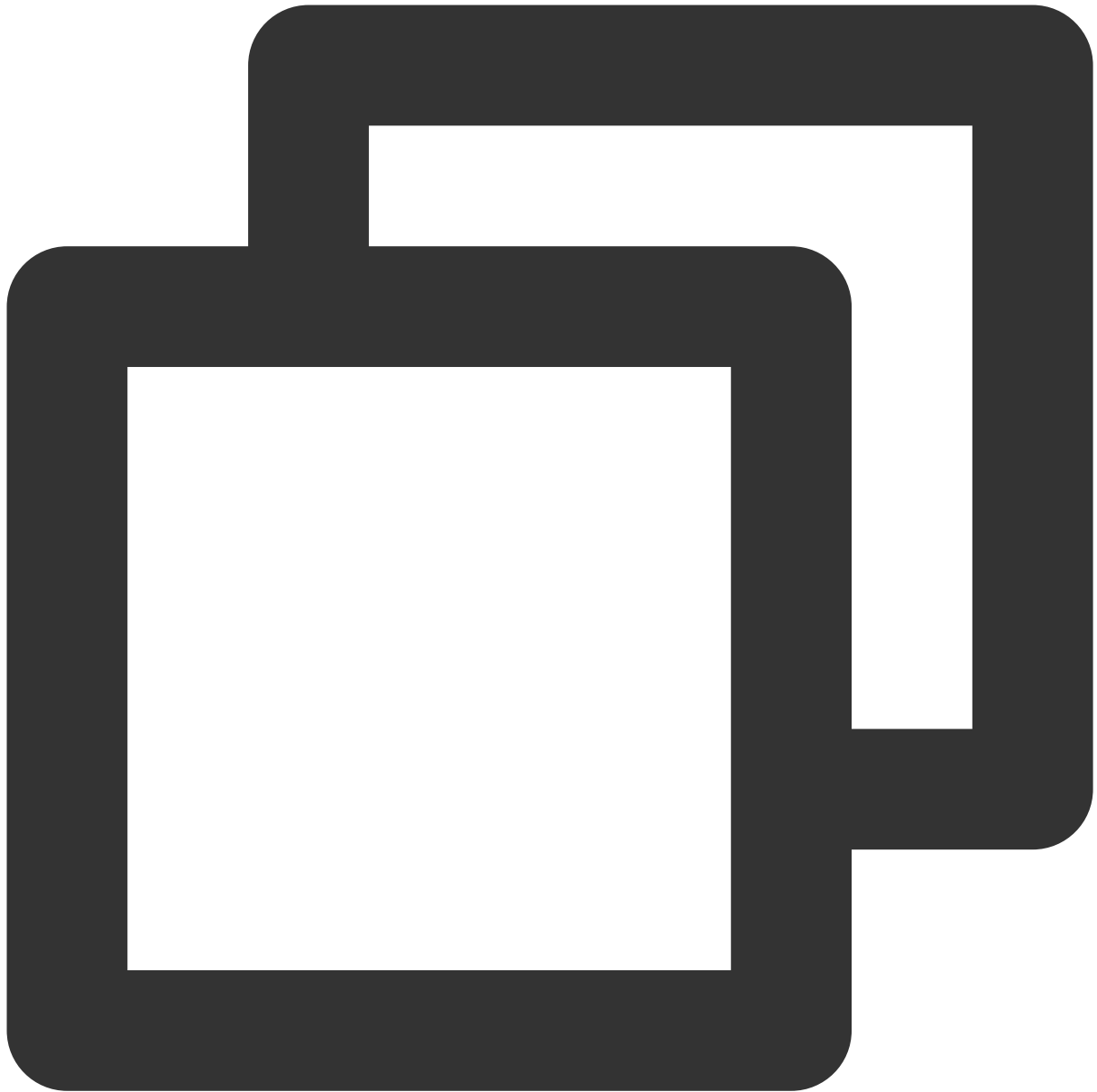
Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID
userRole	<a href="#">TUIRole</a>	Required	-	User Role Host TUIRole.kRoomOwner Administrator TUIRole.kAdministrator General Member TUIRole.kGeneralUser

Returns : *Promise<void>*

## kickRemoteUserOutOfRoom

Kick Out User from Room (Only Host and Administrator can call this Interface).



```
const roomEngine = new TUIRoomEngine();
await roomEngine.kickRemoteUserOutOfRoom({
  userId: 'user_1234',
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
userId	string	Required	-	User ID

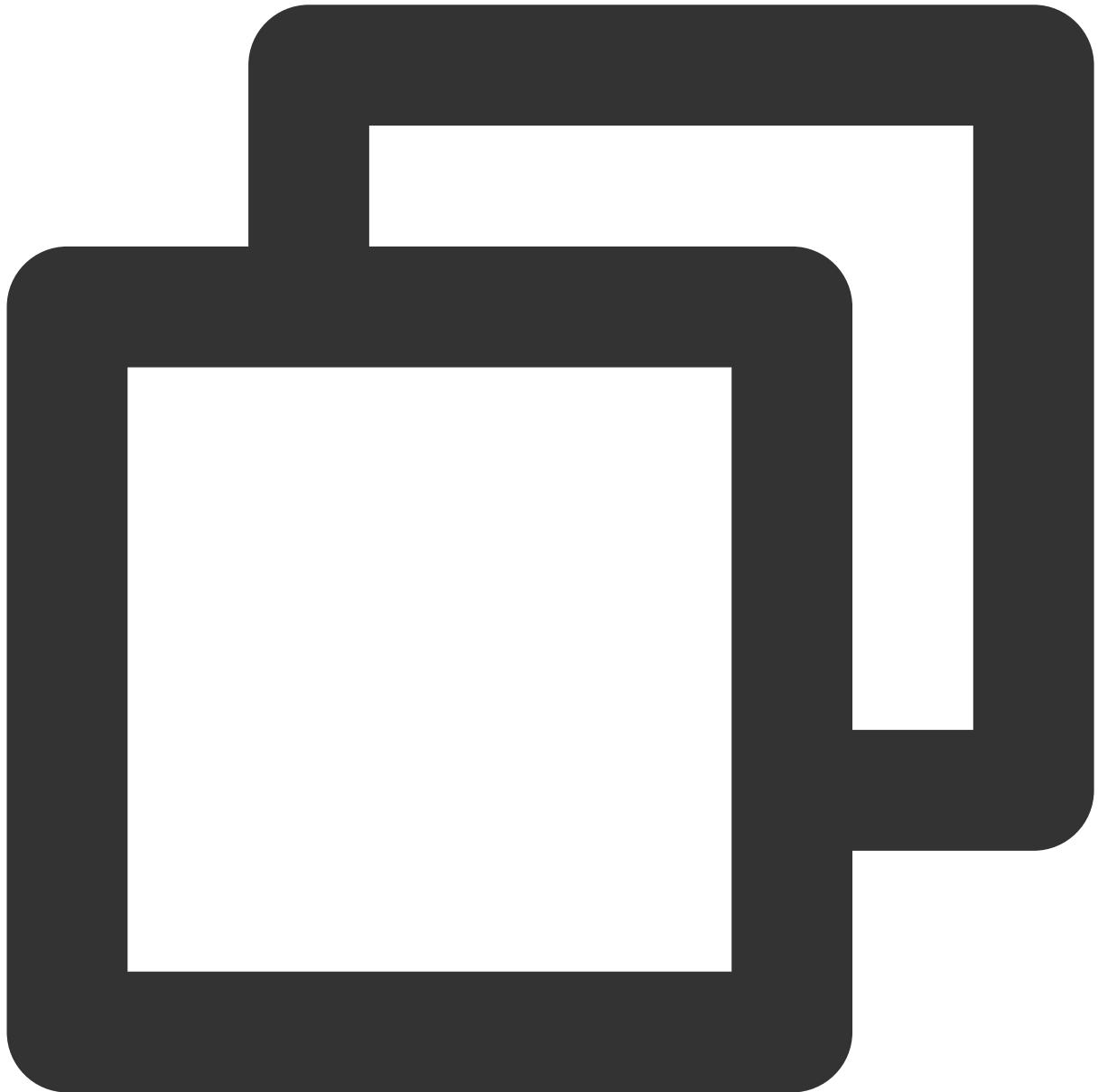
Returns : *Promise<void>*

## setMaxSeatCount

Set Room Seat Maximum Value

When roomType is TUIRoomType.kConference (Education and Conference Scene), maxSeatCount value is not limited;

When roomType is TUIRoomType.kLivingRoom (Live Scene), maxSeatCount is limited to 16;



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.createRoom({ roomId: '12345' });
```

```
await roomEngine.setMaxSeatCount({ maxSeatCount: 16 })
```

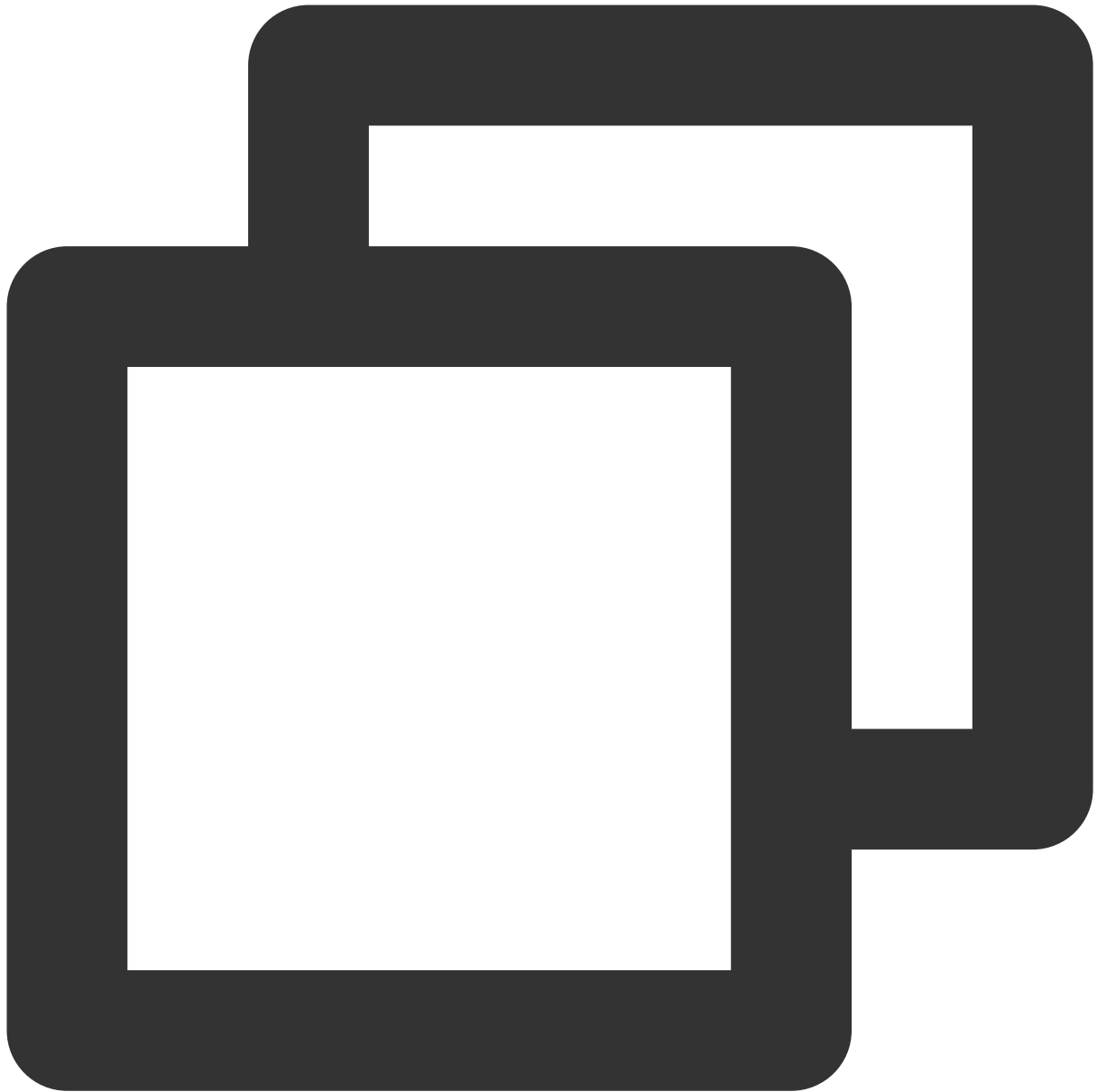
Parameter:

Parameter	Type	Description	Default Value	Meaning
maxSeatCount	number	Required	-	Set Room Seat Maximum Value

Returns : Promise<void>

## getSeatList

Get Seat List



```
const roomEngine = new TUIRoomEngine();  
const seatList = await roomEngine.getSeatList();
```

**Returns** : *Promise*<[TUISeatInfo](#)[]> seatList

seatList for the Current Room's All Seat List

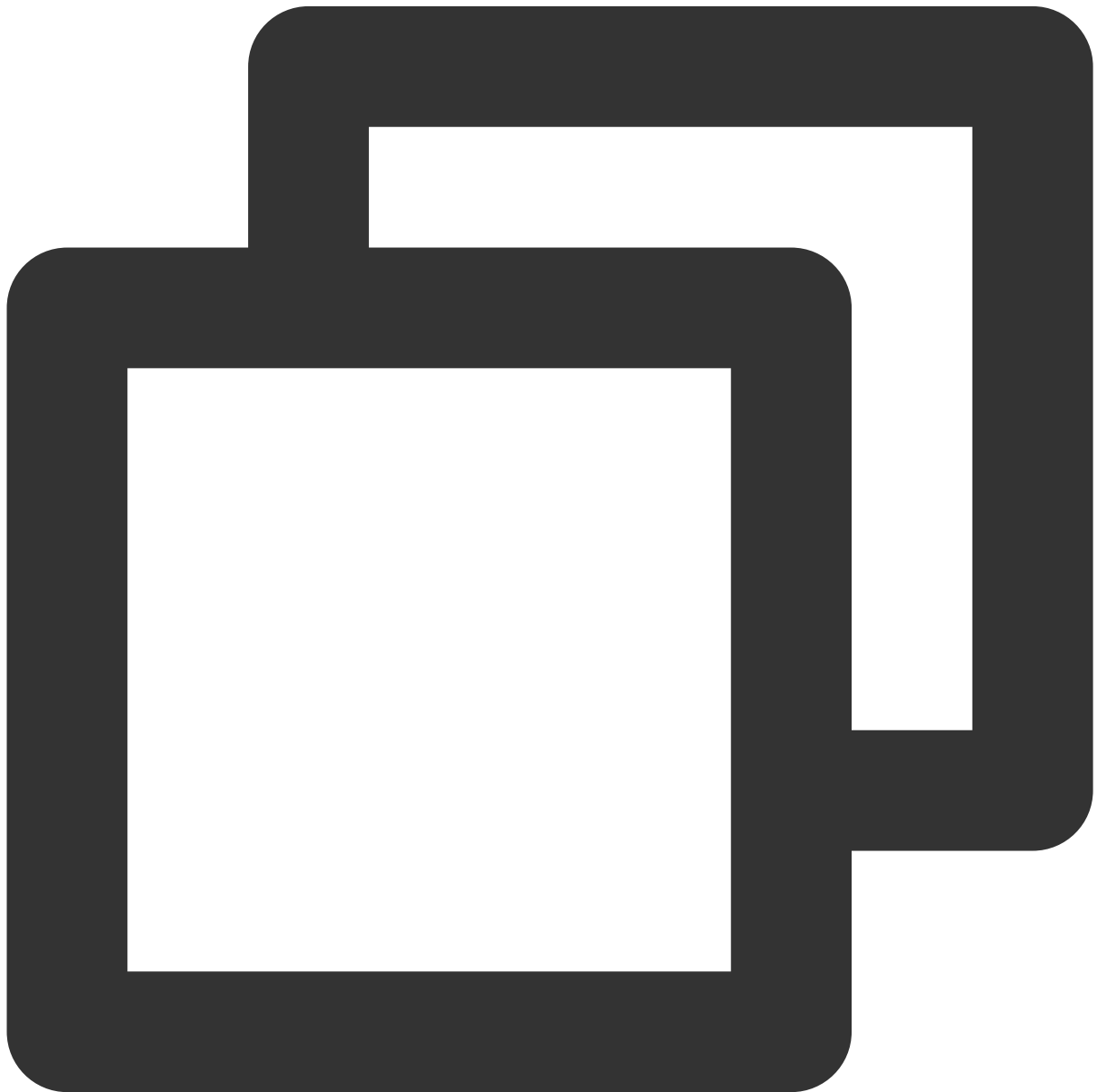
### takeSeat

Mic-off Users can call takeSeat to become Mic-on Users, only Mic-on Users can Publish Local stream.

When `roomInfo.roomType` is `TUIRoomType.kConference` and `roomInfo.speechMode` is `TUISpeechMode.kSpeakAfterTakingSeat`, General Users need to Wait for the Host/Administrator's Agreement to become Mic-on Users after calling `takeSeat` method.

When `roomInfo.roomType` is `TUIRoomType.kLivingRoom` and `roomInfo.speechMode` is `TUISpeechMode.kFreeToSpeak`, General Users become Mic-on Users after calling `takeSeat` method Successfully. Host & Administrator become Mic-on Users after calling `takeSeat` Successfully.

Changes of Mic-on Users are Notified to All Users through `TUIRoomEvents.onSeatListChanged`.



```
const roomEngine = new TUIRoomEngine();  
// Scenario 1: Host/Administrator Go Live
```



```
// Scenario 2: When roomInfo.roomType is TUIRoomType.kConference
// and roomInfo.speechMode is TUISpeechMode.kSpeakAfterTakingSeat, General Users Go
await roomEngine.takeSeat({
  seatIndex: -1,
  timeout: 0,
});

// Scenario 3: When roomInfo.enableSeatControl is true, General Users Go Live
const requestId = await roomEngine.instance?.takeSeat({
  seatIndex: -1,
  timeout: 0,
  requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
    switch (requestCallbackType) {
      case TUIRequestCallbackType.kRequestAccepted:
        // Request Accepted
        break;
      case TUIRequestCallbackType.kRequestRejected:
        // Request Rejected
        break;
      case TUIRequestCallbackType.kRequestCancelled:
        // Request Canceled
        break;
      case TUIRequestCallbackType.kRequestTimeout:
        // Request Timeout
        break;
      case TUIRequestCallbackType.kRequestError:
        // Request Error
        break;
      default:
        break;
    }
  },
});
```

Parameter:

Parameter	Type	Description	Default Value	Meaning
seatIndex	number	Required	-	Seat index, set to -1 when there is No Serial Number
timeout	number	Required	-	Timeout. If timeout is set to 0, there is no Timeout
requestCallback	Function	Optional	Empty Function	Callback, used to Notify Initiator of Request being Accepted/Rejected/Cancelled/Timeout/Error

**Returns** : Promise<string> requestId

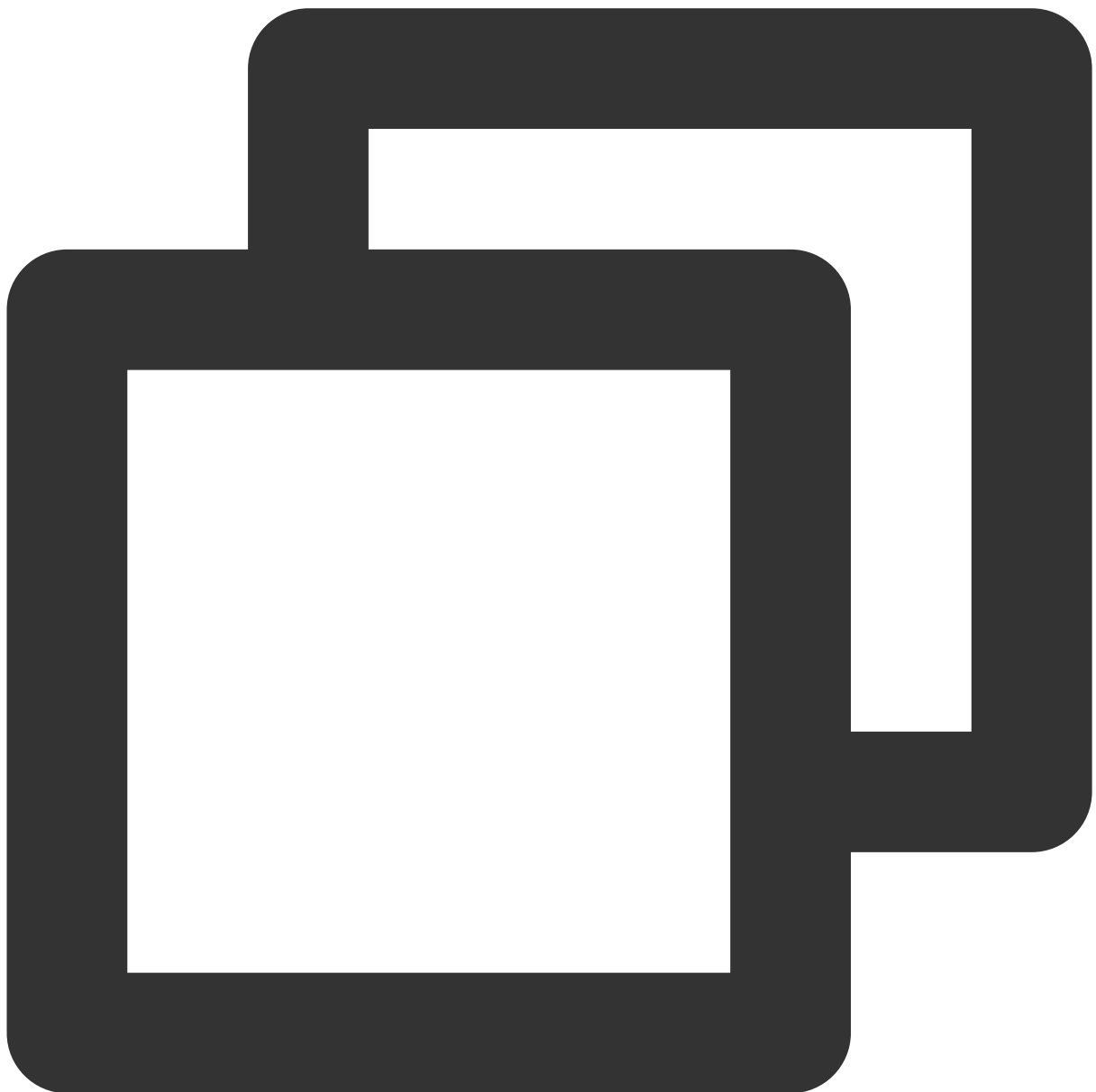
When roomInfo.enableSeatControl is true, General Users call this Interface to return requestId, General Users can use this requestId to call cancelRequest Interface to cancel Go Live Request.

**Note:**

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number;

## leaveSeat

Release Seat.

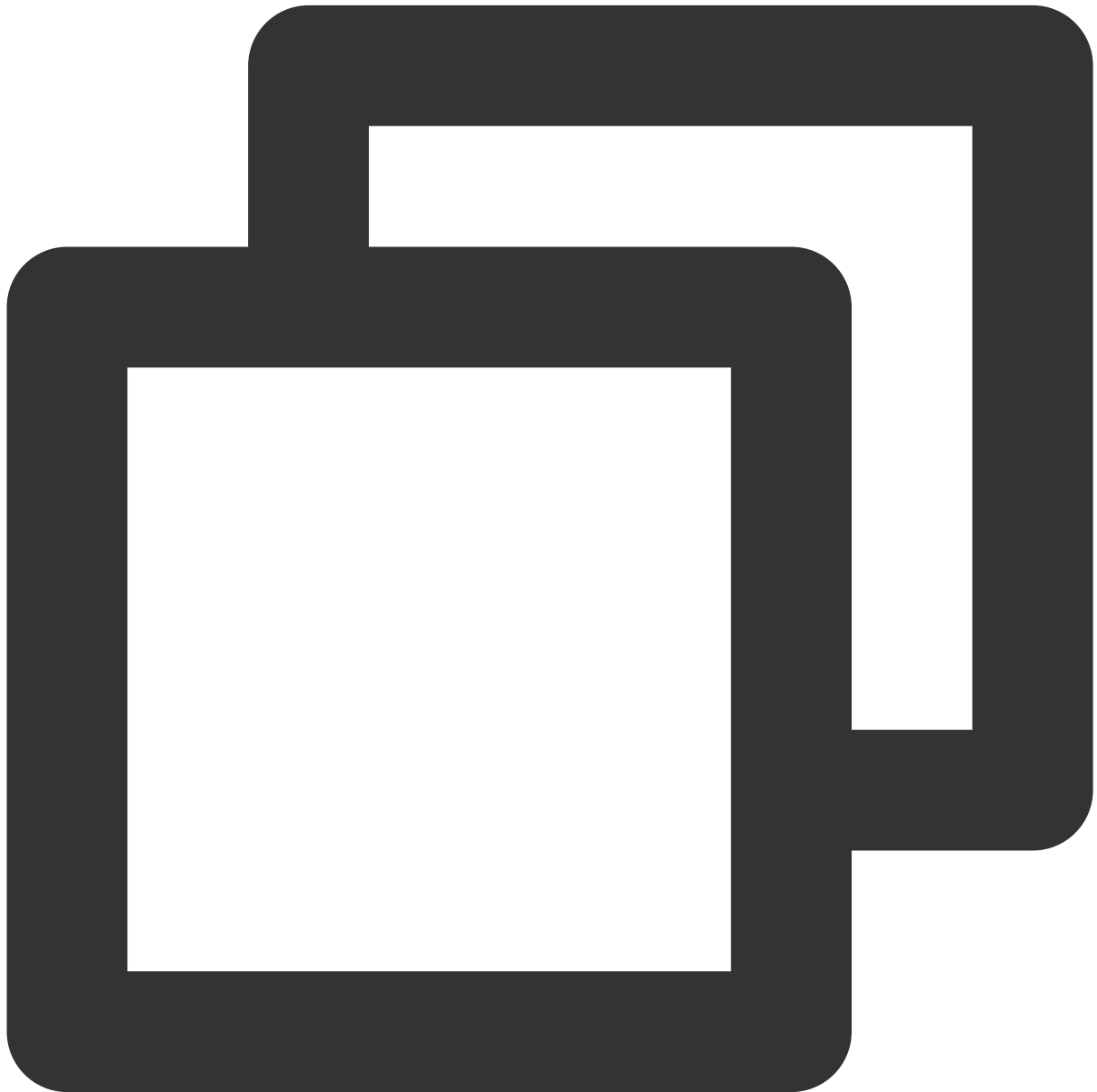


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.leaveSeat();
```

**Returns** : *Promise<void>*

## takeUserOnSeatByAdmin

Invite Others to Go Live.



```
const roomEngine = new TUIRoomEngine();  
const requestId = roomEngine.takeUserOnSeatByAdmin({  
  seatIndex: 0,
```

```

    userId: 'user_1234',
    timeout: 0,
    requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
    switch (requestCallbackType) {
      case TUIRequestCallbackType.kRequestAccepted:
        // Request Accepted
        break;
      case TUIRequestCallbackType.kRequestRejected:
        // Request Rejected
        break;
      case TUIRequestCallbackType.kRequestCancelled:
        // Request Canceled
        break;
      case TUIRequestCallbackType.kRequestTimeout:
        // Request Timeout
        break;
      case TUIRequestCallbackType.kRequestError:
        // Request Error
        break;
      default:
        break;
    }
  },
});

```

**Parameter:**

Parameter	Type	Description	Default Value	Meaning
seatIndex	number	Required	-	Seat index
userId	string	Required	-	User ID
timeout	number	Required	-	Timeout. If timeout is set to 0, there is no Timeout
requestCallback	Function	Optional	Empty Function	Callback, used to Notify Initiator of Request being Accepted/Rejected/Cancelled/Timeout/Error

**Returns :** *Promise<string> requestId*

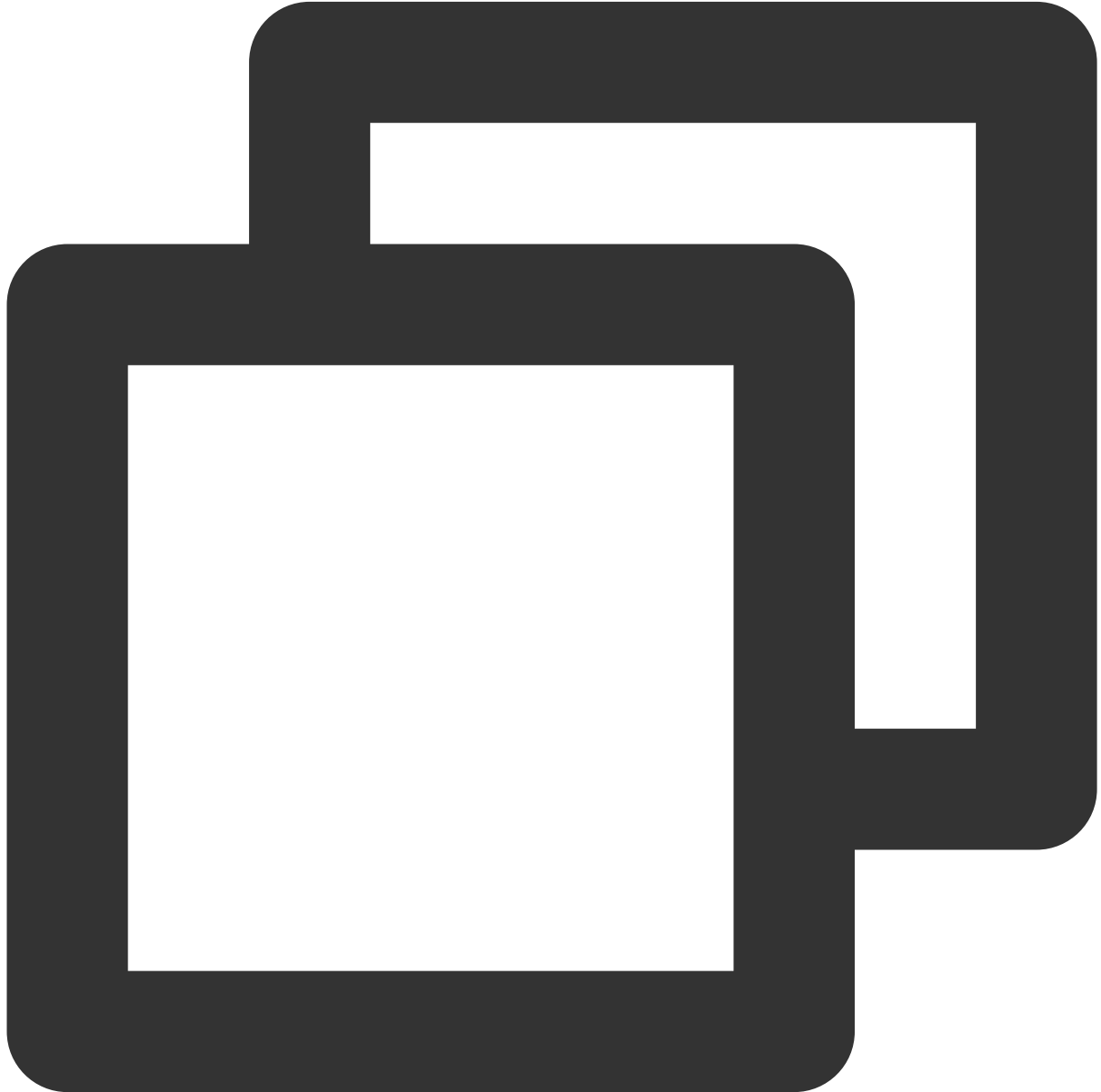
This Interface returns requestId, Users can use this requestId to call cancelRequest Interface to cancel Request.

**Note:**

In v1.0.2 and above, the requestId returned by this interface is of type string; in v1.0.0 and v1.0.1, the requestId returned by this interface is of type number;

## kickUserOffSeatByAdmin

Ask others to get off the mic



```
const roomEngine = new TUIRoomEngine();
const requestId = await roomEngine.kickUserOffSeatByAdmin({
  seatIndex: 0,
  userId: 'user_1234',
});
```

Parameter:

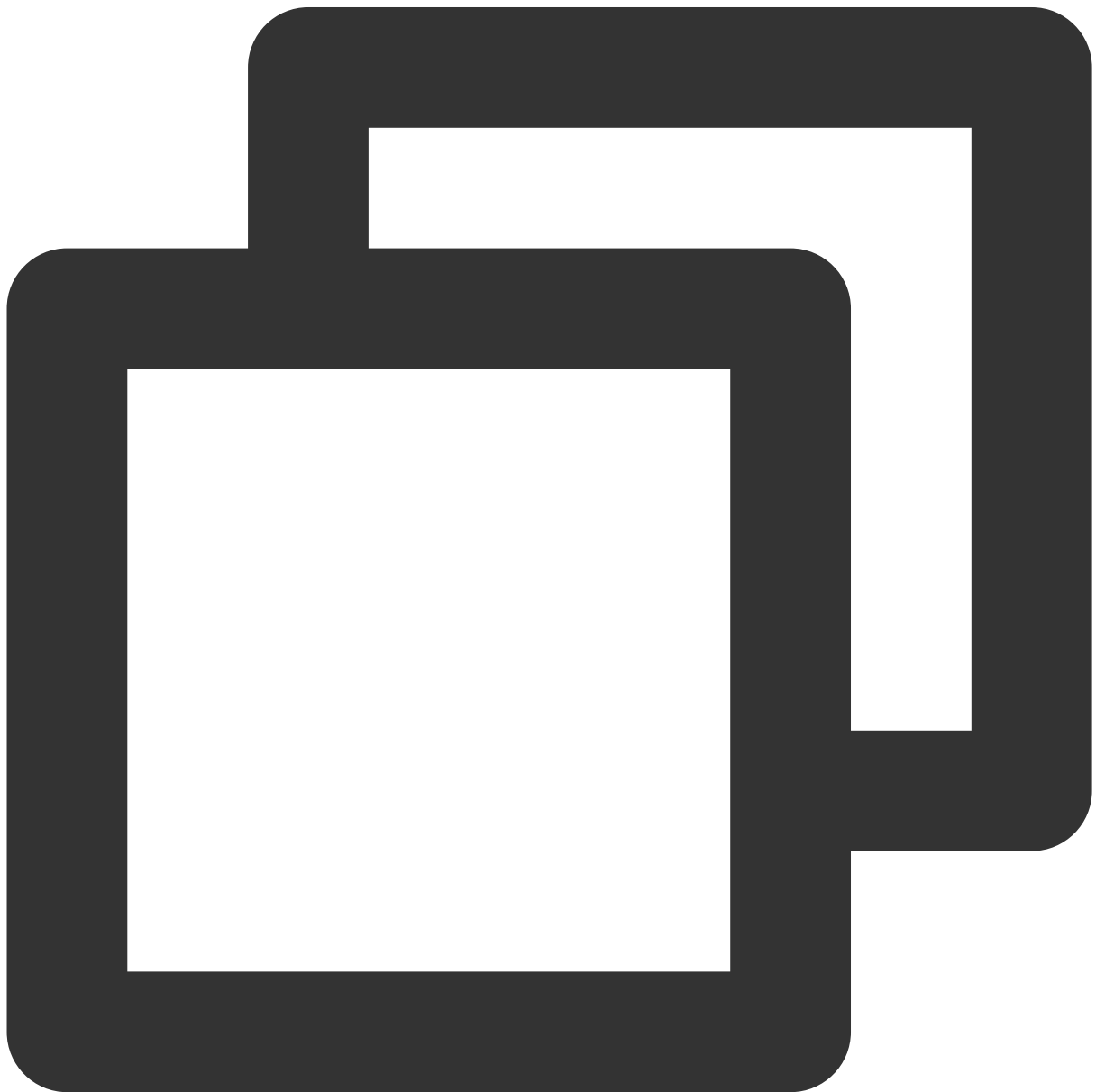
--	--	--	--	--

Parameter	Type	Description	Default Value	Meaning
seatIndex	number	Required	-	Seat index
userId	string	Required	-	User ID

**Returns** : *Promise<void>*

### lockSeatByAdmin

Lock a certain mic position status (Only the room host and administrator can call this method).



```
const roomEngine = new TUIRoomEngine();
await roomEngine.lockSeatByAdmin({
  seatIndex: 0,
  lockParams: {
    lockSeat: true,
    lockVideo: true,
    lockAudio: true,
  },
});
```

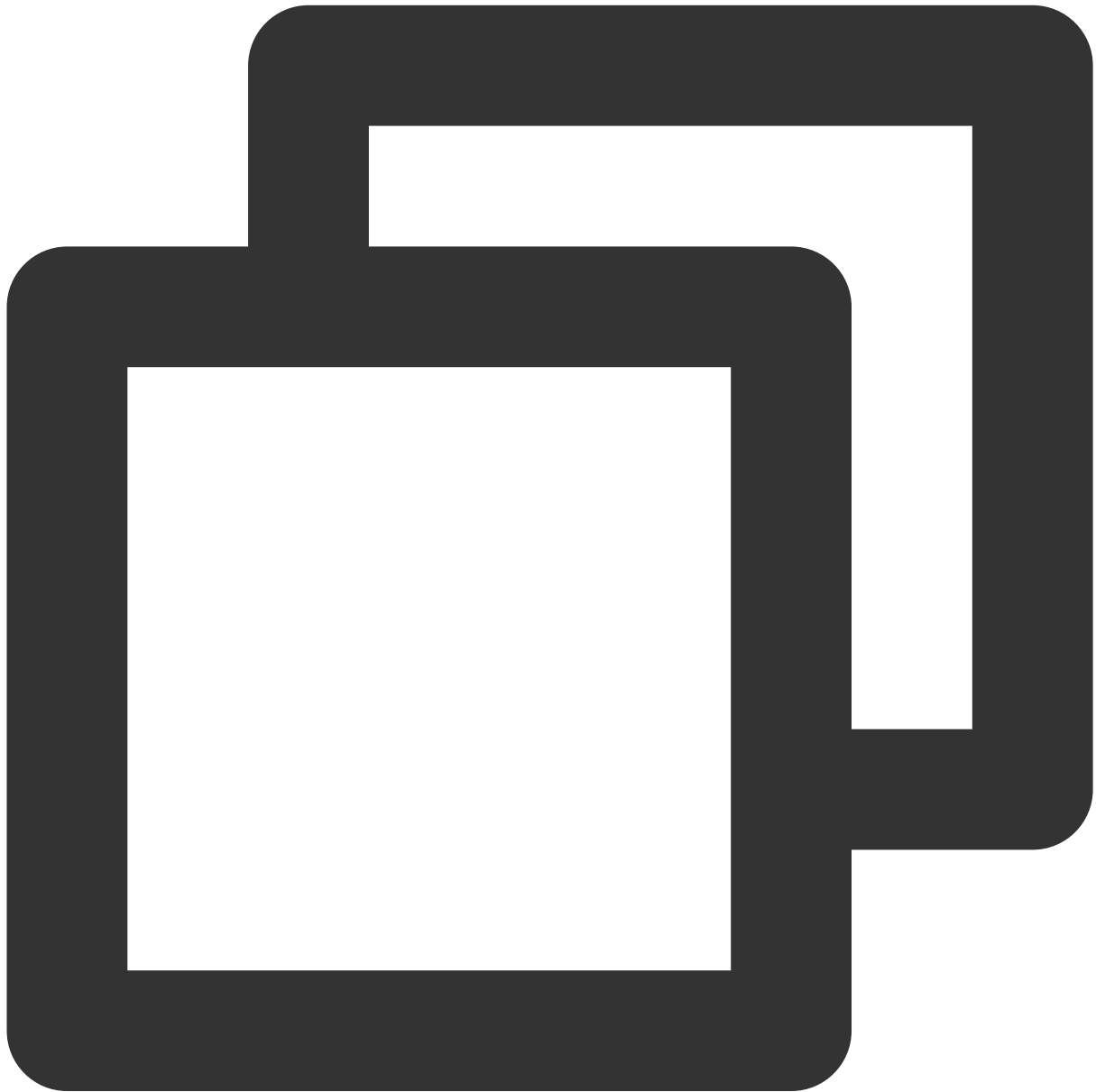
Parameter:

Parameter	Type	Description	Default Value	Meaning
seatIndex	number	Required	-	Mic position index
lockParams	<a href="#">TUISeatLockParams</a>	Required	-	Lock mic parameter

Returns : *Promise<void>*

## startScreenSharingElectron

Start Screen Sharing Electron。



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.startScreenSharingElectron('targetId');
```

Parameter:

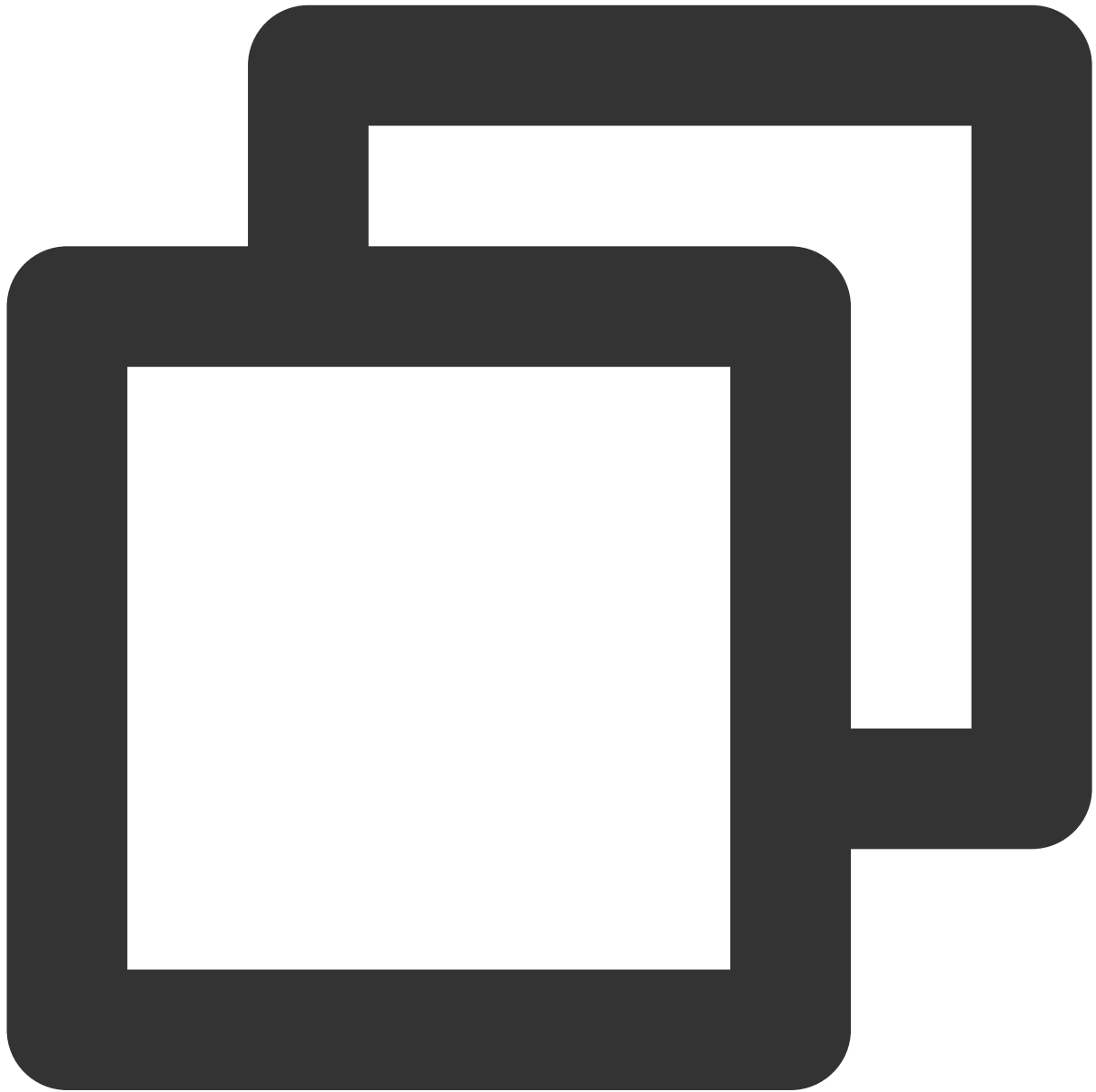
Parameter	Type	Description	Default Value	Meaning
targetId	string	Optional	-	Sharing Window ID, can be obtained from <code>getScreenSharingTarget</code>



**Returns** : *Promise<void>*

## stopScreenSharingElectron

Stop Screen Sharing Electron.

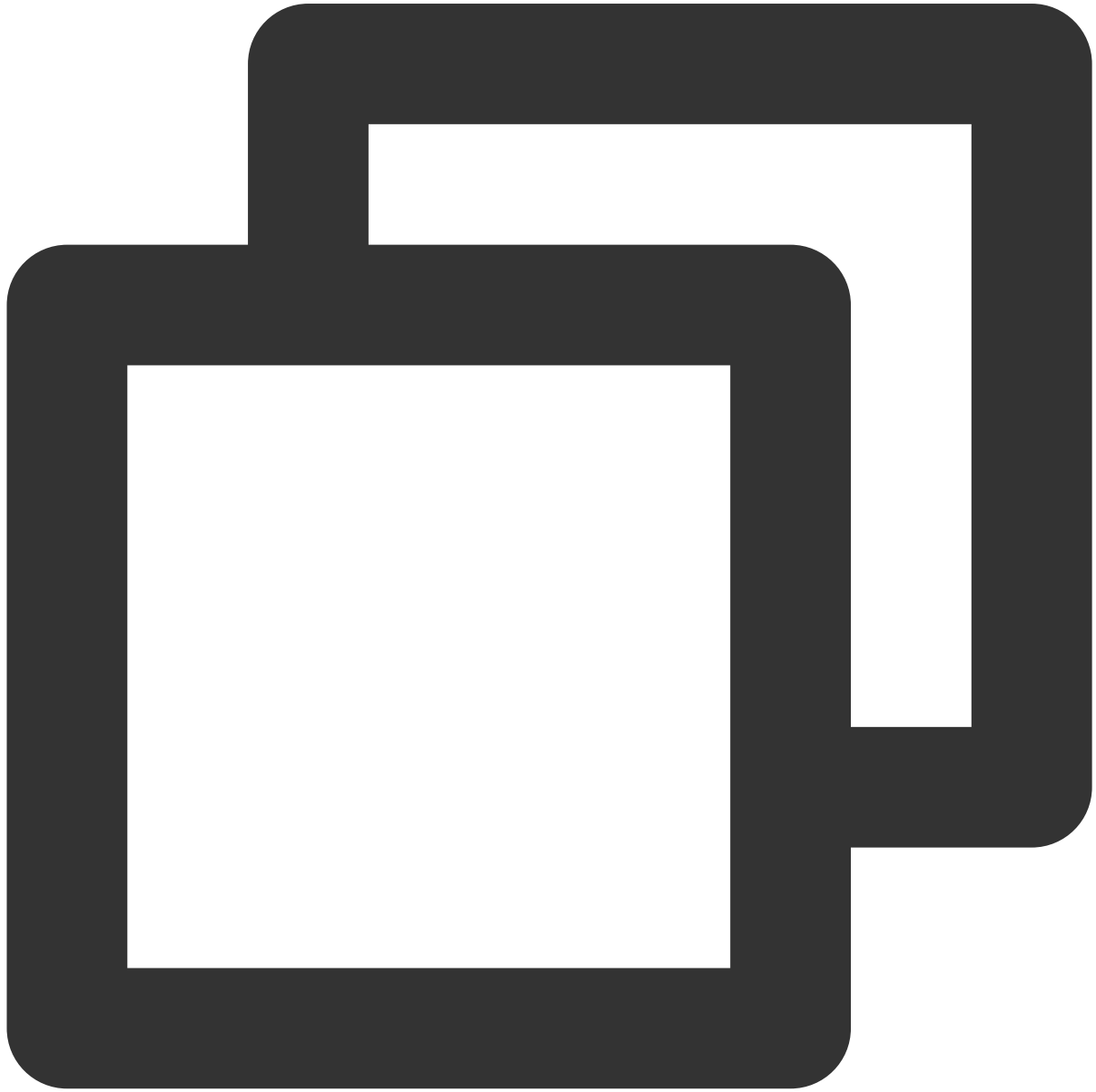


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.stopScreenSharingElectron();
```

**Returns** : *Promise<void>*

## getScreenSharingTarget

Get Screen Sharing list Electron。

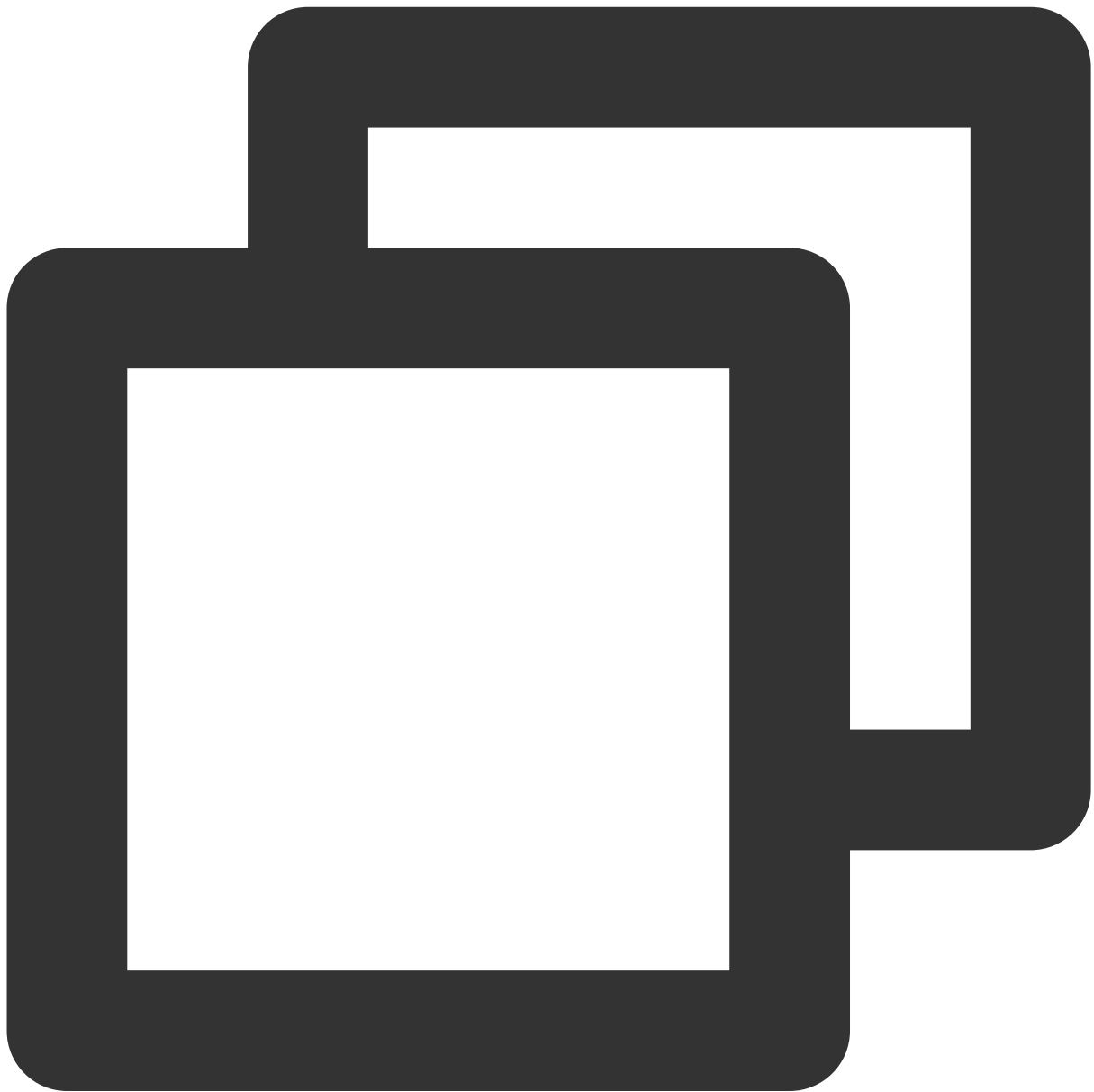


```
const roomEngine = new TUIRoomEngine();  
const screenList = await roomEngine.getScreenSharingTarget();
```

**Returns** : *Promise<void>*

### **selectScreenSharingTarget**

Switch Screen Sharing Window Electron。



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.selectScreenSharingTarget('targetId');
```

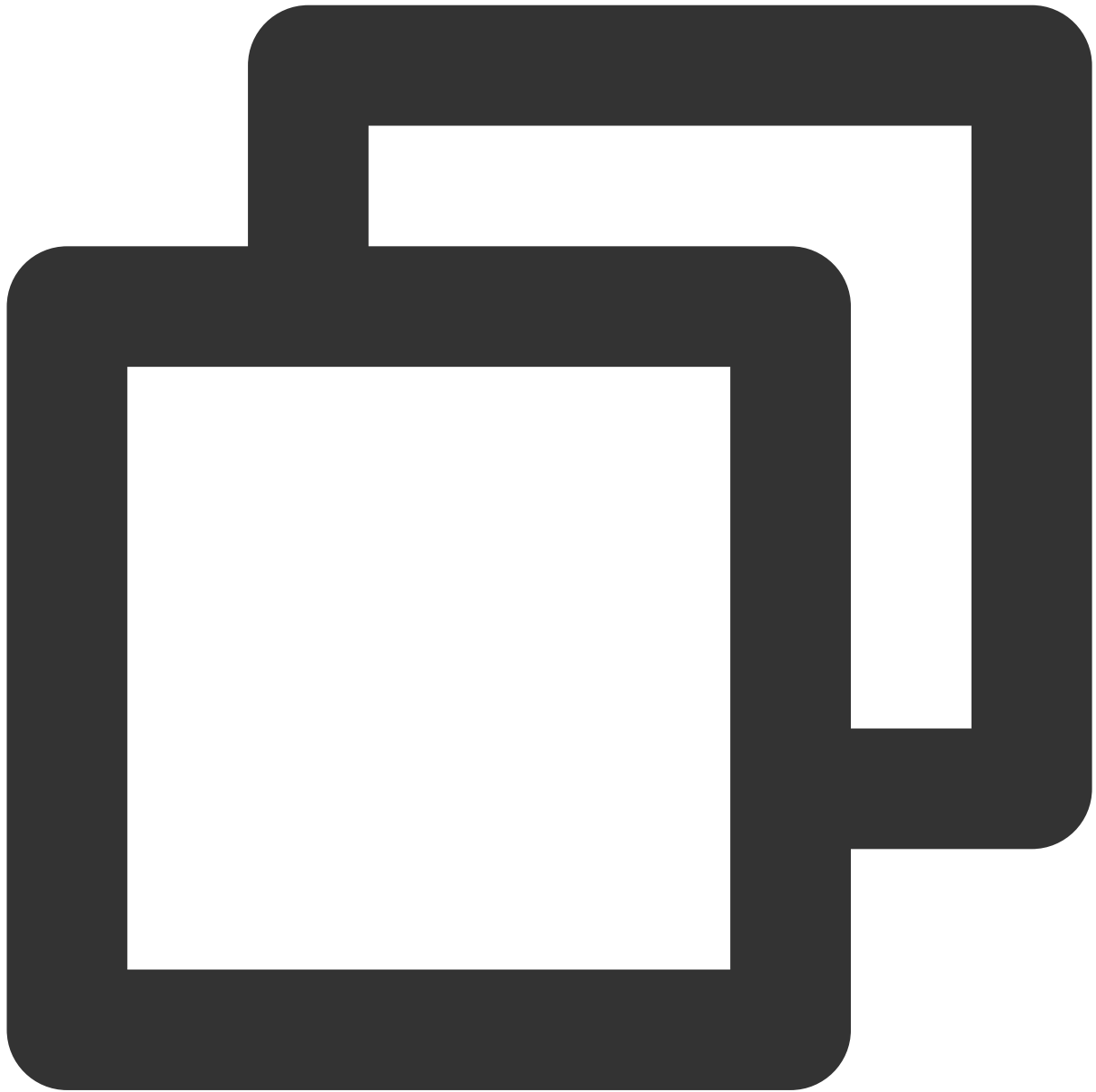
Parameter:

Parameter	Type	Description	Default Value	Meaning
targetId	string	Required	-	Sharing Window ID, can be obtained from <code>getScreenSharingTarget</code>

**Returns** : *Promise<void>*

## **sendTextMessage**

Send text message.



```
const roomEngine = new TUIRoomEngine();
await roomEngine.sendTextMessage({
  messageText: 'hello, everyone',
});
```

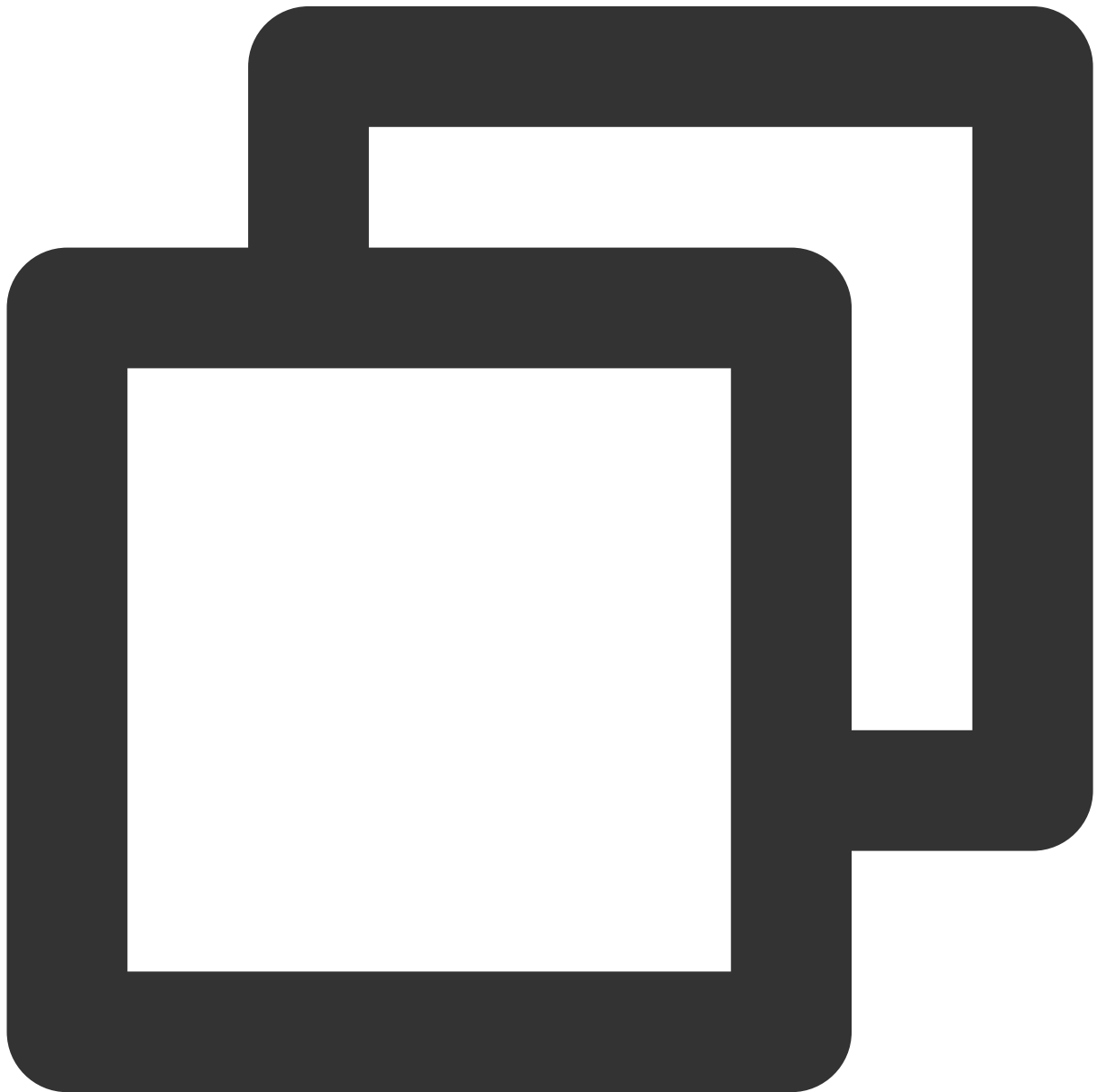
**Parameter:**

Parameter	Type	Description	Default Value	Meaning
messageText	string	Required	-	Text message content

**Returns** : *Promise<void>*

## sendCustomMessage

Send custom message.



```
const roomEngine = new TUIRoomEngine();
```

```
await roomEngine.sendCustomMessage({
  messageText: '{ data:', description: ''}',
});
```

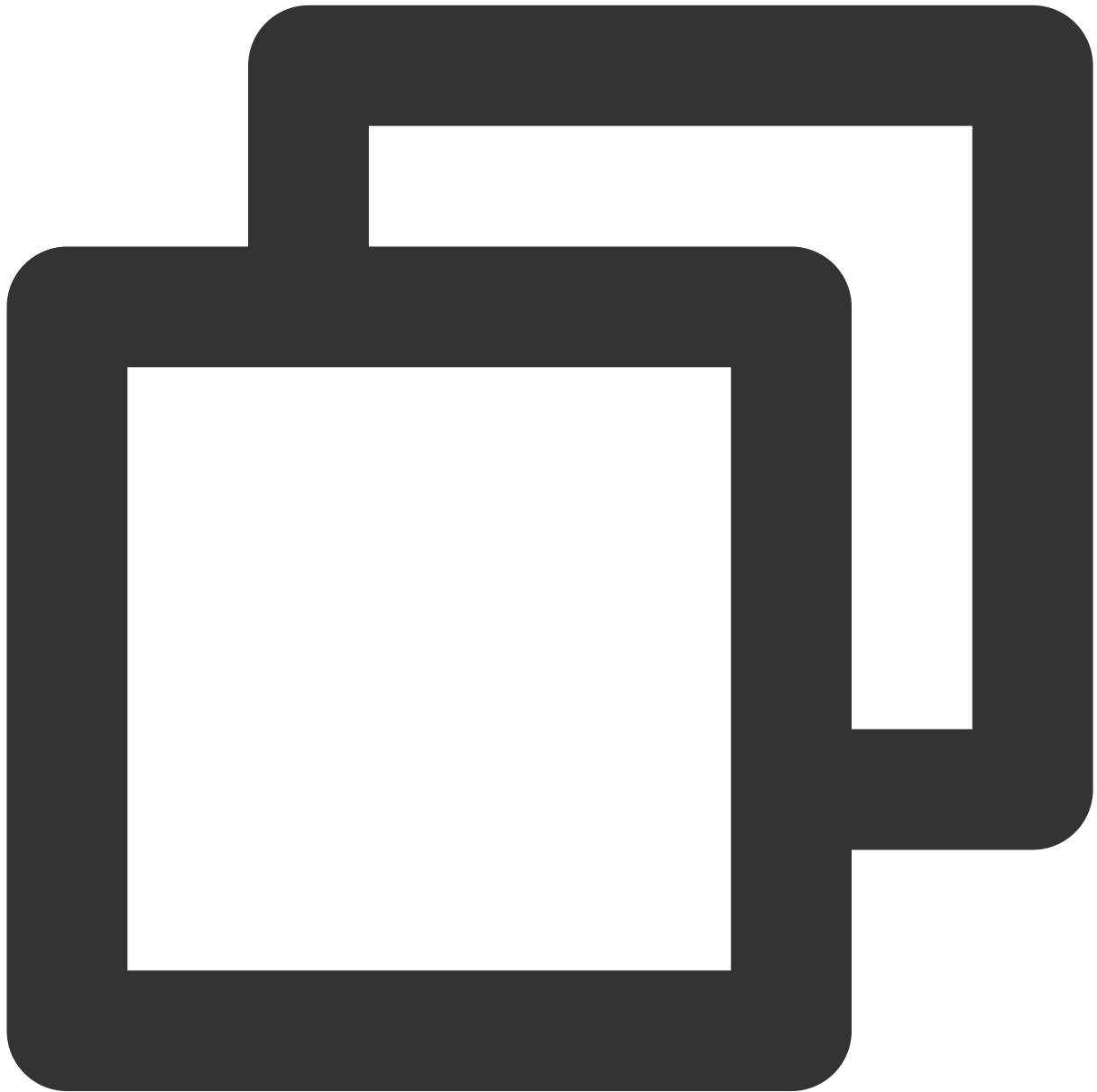
Parameter:

Parameter	Type	Description	Default Value	Meaning
messageText	string	Required	-	Custom message content

**Returns** : *Promise<void>*

## getCameraDevicesList

Get camera device list.

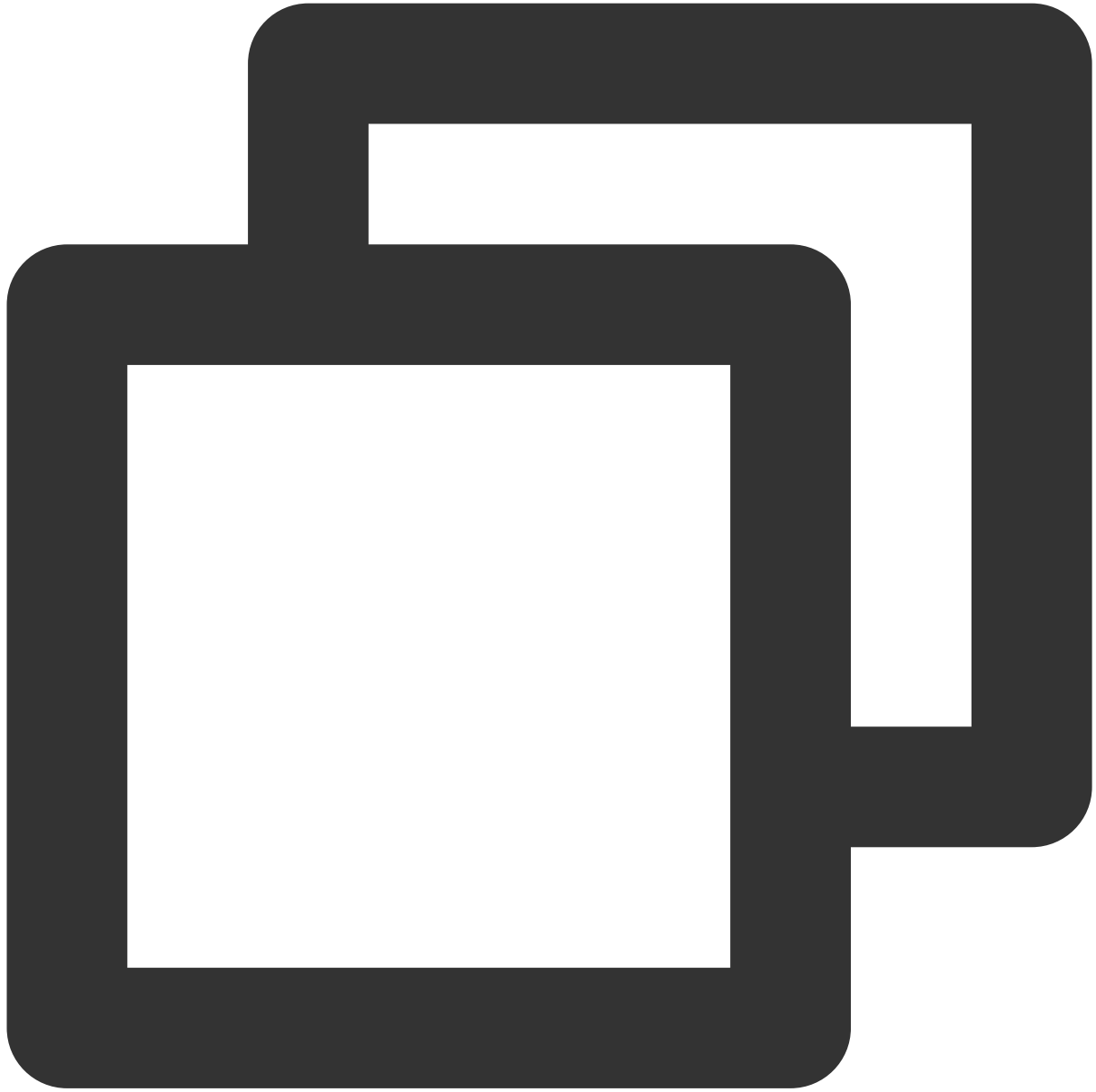


```
const roomEngine = new TUIRoomEngine();
const cameraList = await roomEngine.getCameraDevicesList();
for (i = 0; i < cameraList.length; i++) {
  var camera = cameraList[i];
  console.info("camera deviceName: " + camera.deviceName + " deviceId:" + camera.
}
```

**Returns:** *Promise*<[TRTCDeviceInfo\[\]](#)> cameraList

## getMicDevicesList

Get mic device list.



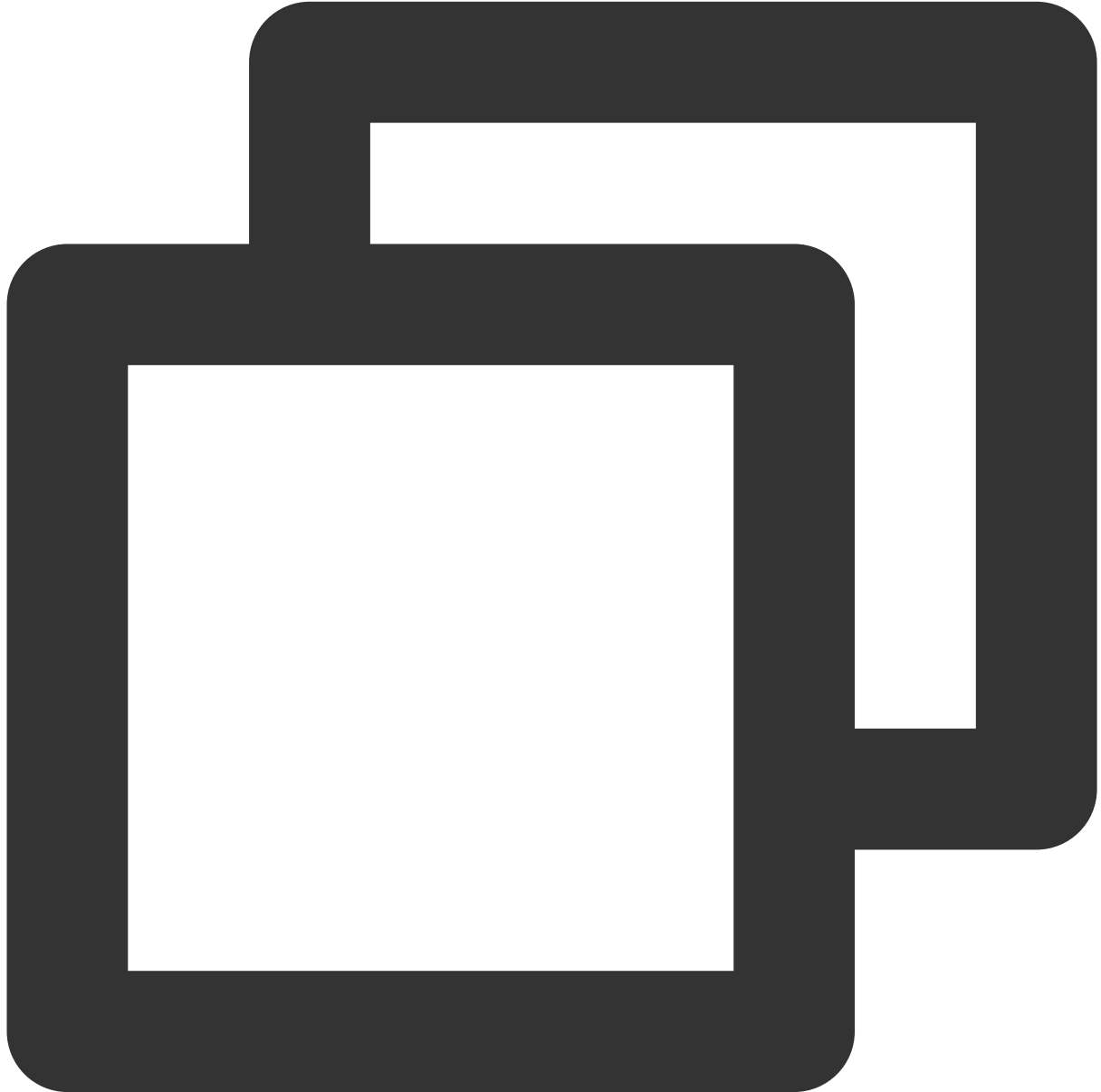
```
const roomEngine = new TUIRoomEngine();
const micList = await roomEngine.getMicDevicesList();
for (i = 0; i < micList.length; i++) {
  var mic = micList[i];
  console.info("mic deviceName: " + mic.deviceName + " deviceId:" + mic.deviceId)
}
```

**Returns:** *Promise*<[TRTCDeviceInfo\[\]](#)> micList



## getSpeakerDevicesList

Get speaker device list.

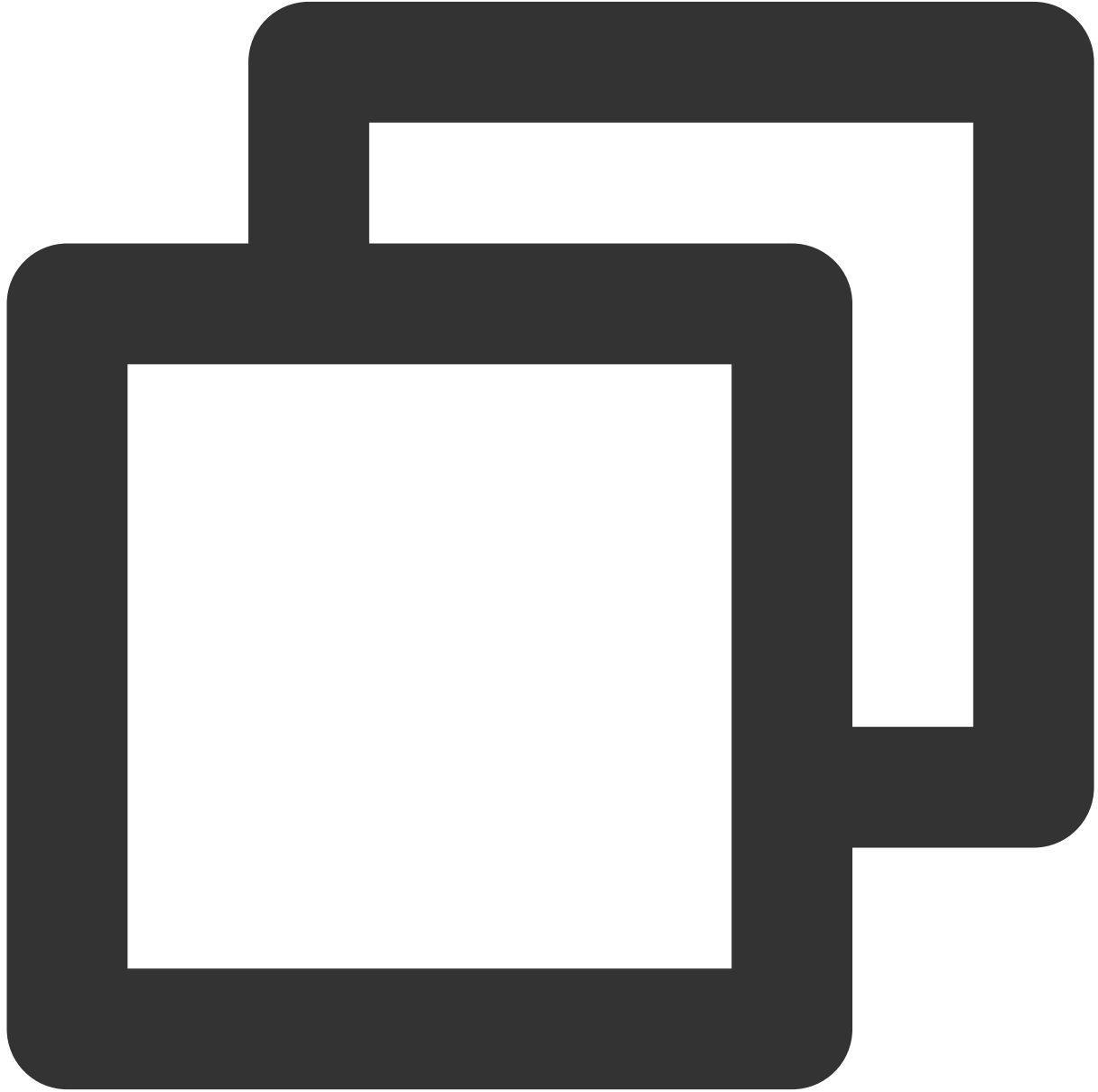


```
const roomEngine = new TUIRoomEngine();
const speakerList = await roomEngine.getSpeakerDevicesList();
for (i = 0; i < speakerList.length; i++) {
  var speaker = speakerList[i];
  console.info("speaker deviceName: " + speaker.deviceName + " deviceId:" + speaker.deviceId);
}
```

**Returns:** *Promise*<[TRTCDeviceInfo\[\]](#)> speakerList

## setCurrentCameraDevice

Set the camera device to be used.



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.setCurrentCameraDevice({ deviceId: '' });
```

Parameter:

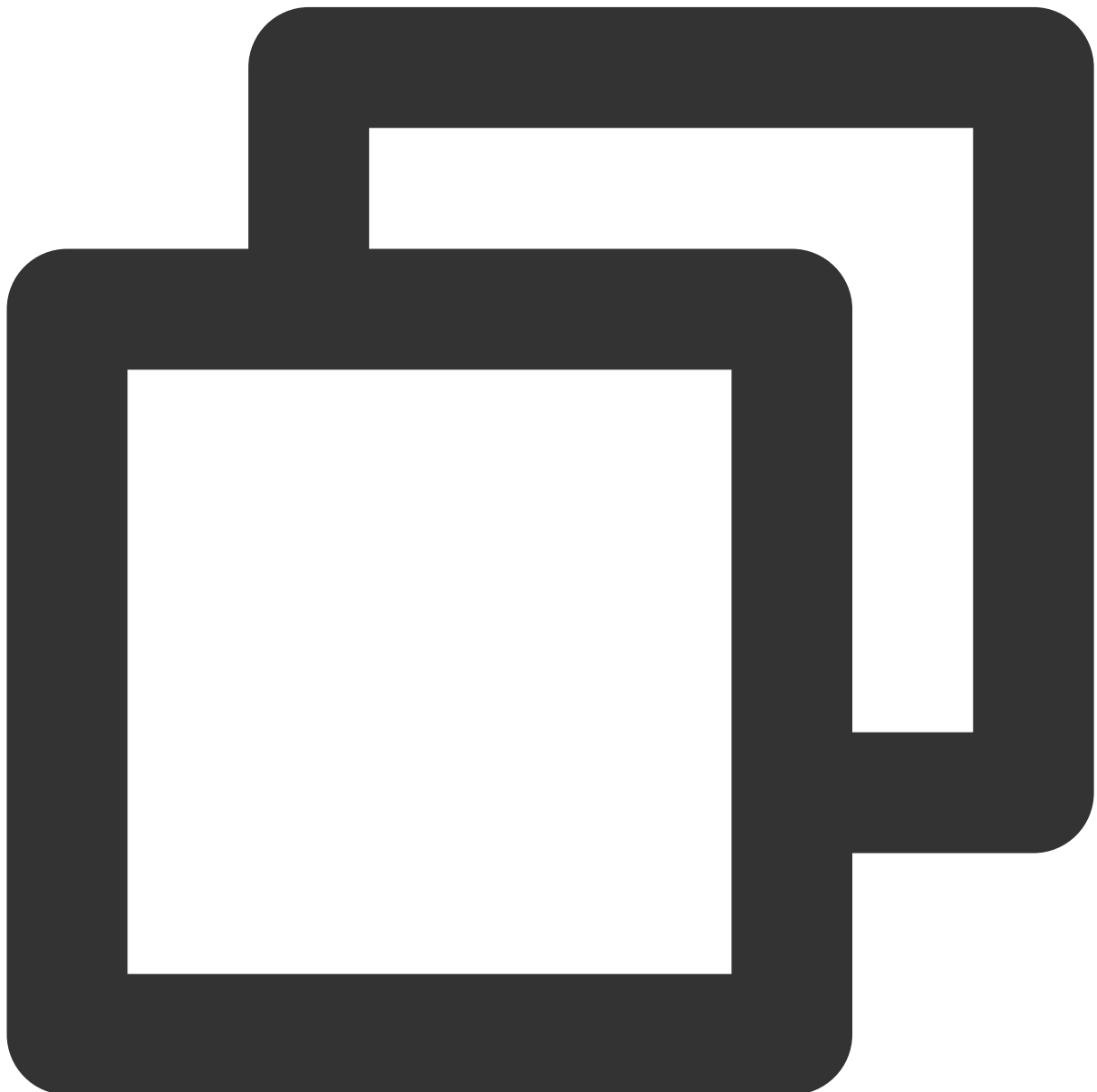
Parameter	Type	Description	Default Value	Meaning

deviceId	string	Required	-	Device ID obtained from getCameraDevicesList
----------	--------	----------	---	---

**Returns :** *void*

### **setCurrentMicDevice**

Set the mic device to be used.



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.setCurrentMicDevice({ deviceId: '' });
```

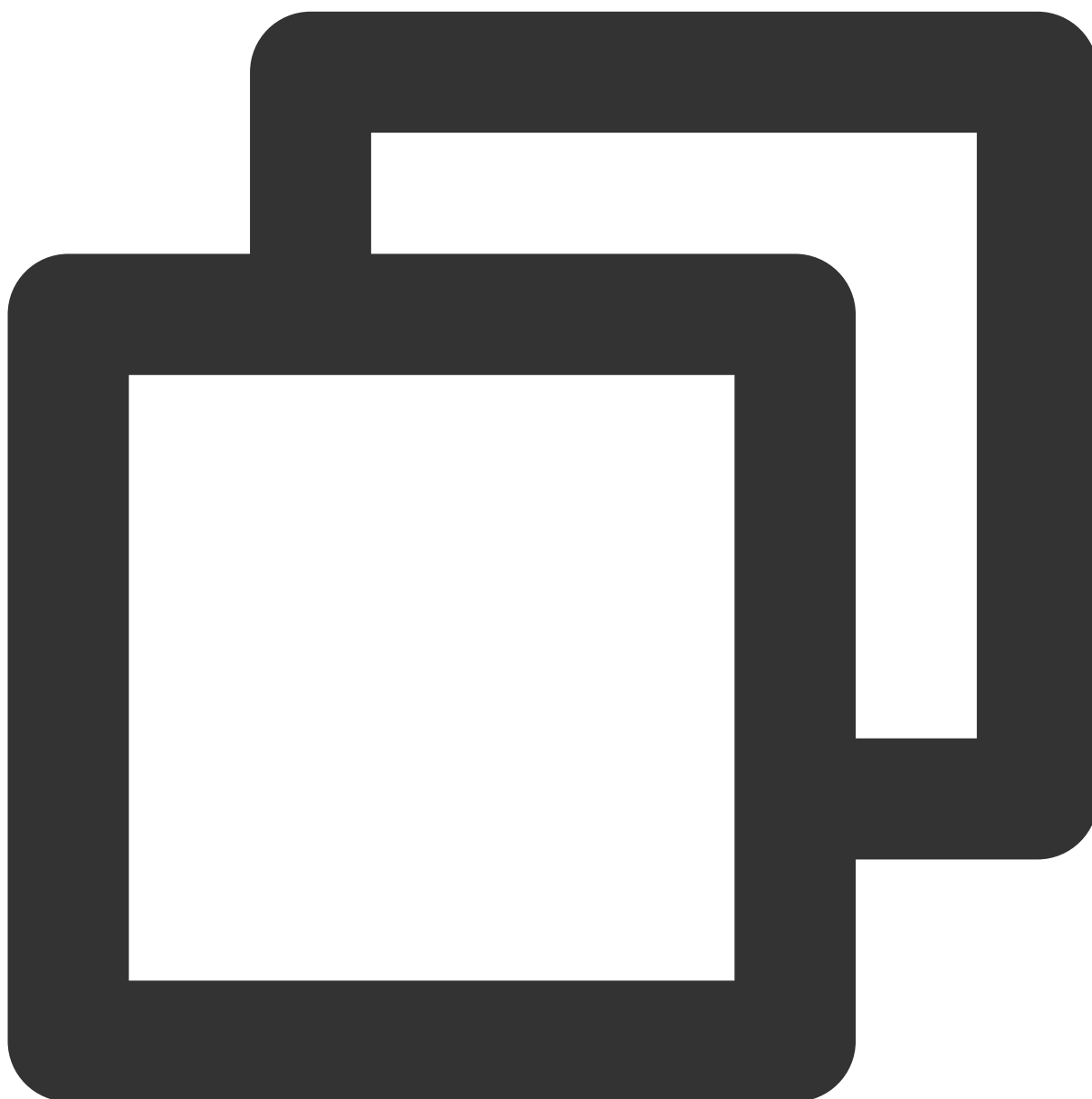
Parameter:

Parameter	Type	Description	Default Value	Meaning
deviceId	string	Required	-	Device ID obtained from getSpeakerDevicesList

**Returns :** *void*

### **setCurrentSpeakerDevice**

Get the currently used camera device.



```
const roomEngine = new TUIRoomEngine();  
await roomEngine.setCurrentSpeakerDevice({ deviceId: '' });
```

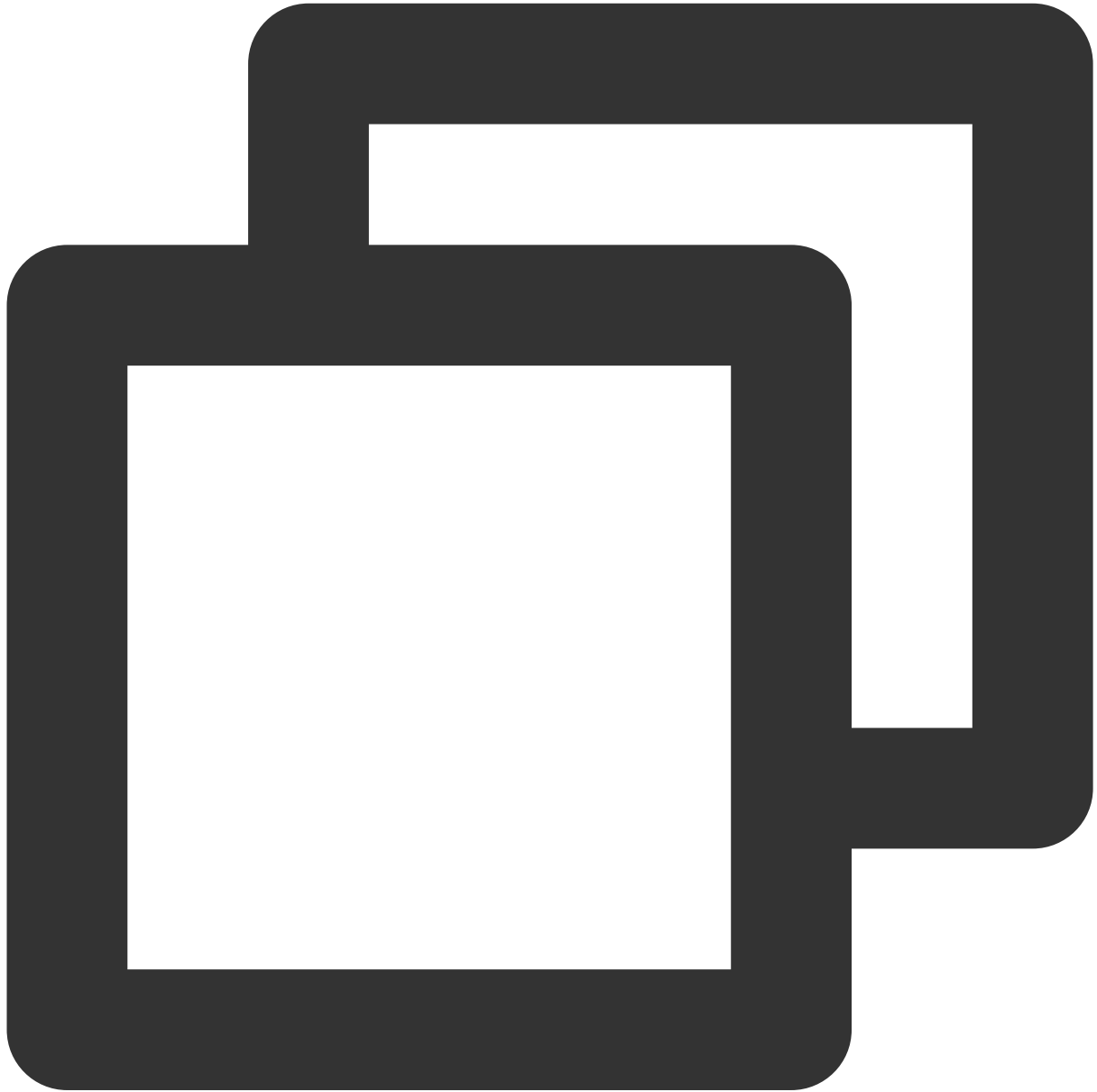
Parameter:

Parameter	Type	Description	Default Value	Meaning
deviceId	string	Required	-	Device ID obtained from getSpeakerDevicesList

**Returns :** *void*

### **getCurrentCameraDevice**

Get the currently used camera device.



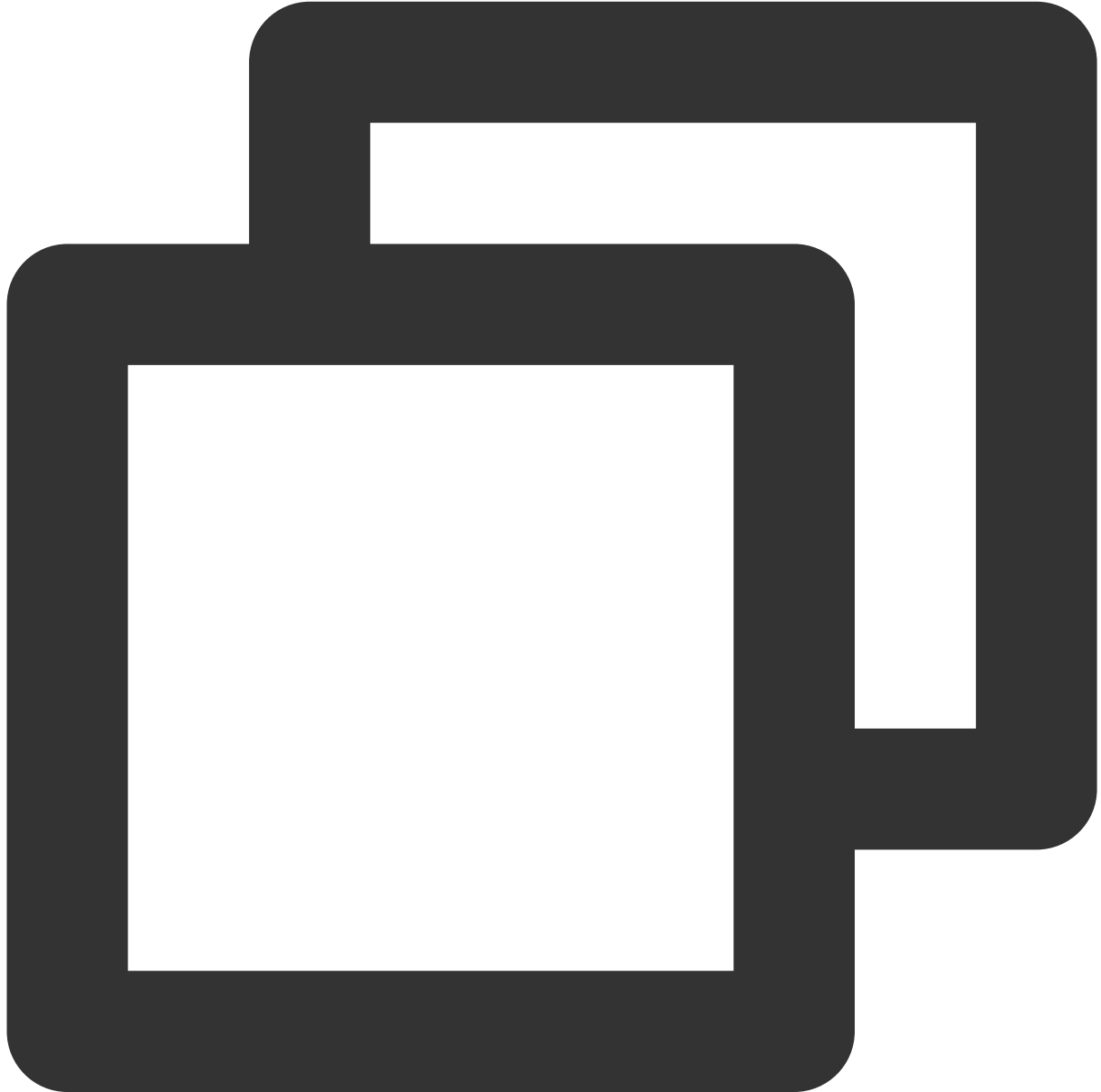
```
const roomEngine = new TUIRoomEngine();  
const currentCameraDevice = roomEngine.getCurrentCameraDevice();
```

**Returns :** [TRTCDeviceInfo](#)

Device information, can get device ID and device name

## getCurrentMicDevice

Get the currently used mic device.



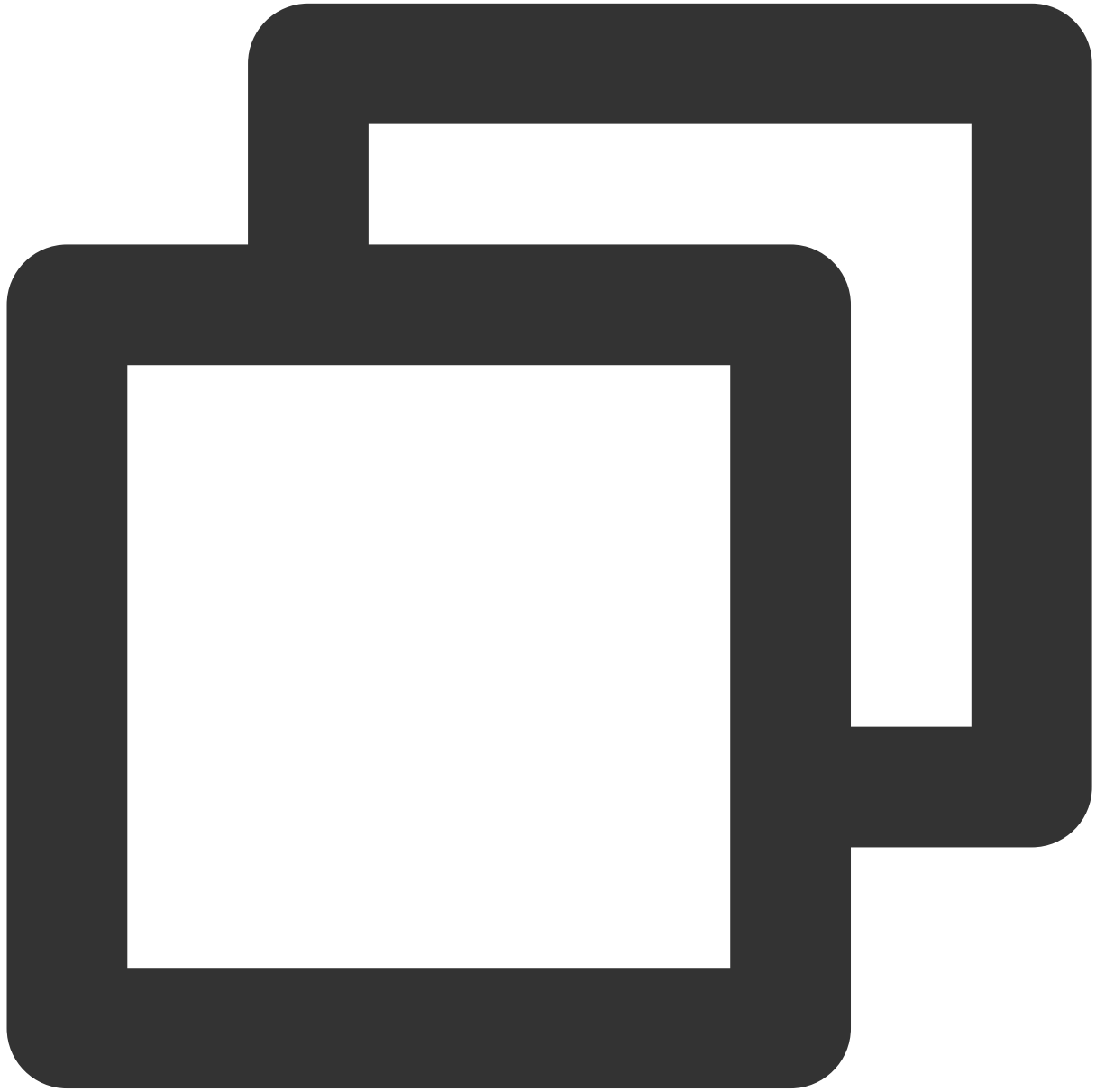
```
const roomEngine = new TUIRoomEngine();  
const currentMicDevice = roomEngine.getCurrentMicDevice();
```

**Returns :** [TRTCDeviceInfo](#)

Device information, can get device ID and device name

## getCurrentSpeakerDevice

Get the currently used speaker device.



```
const roomEngine = new TUIRoomEngine();  
const currentSpeakerDevice = roomEngine.getCurrentSpeakerDevice();
```

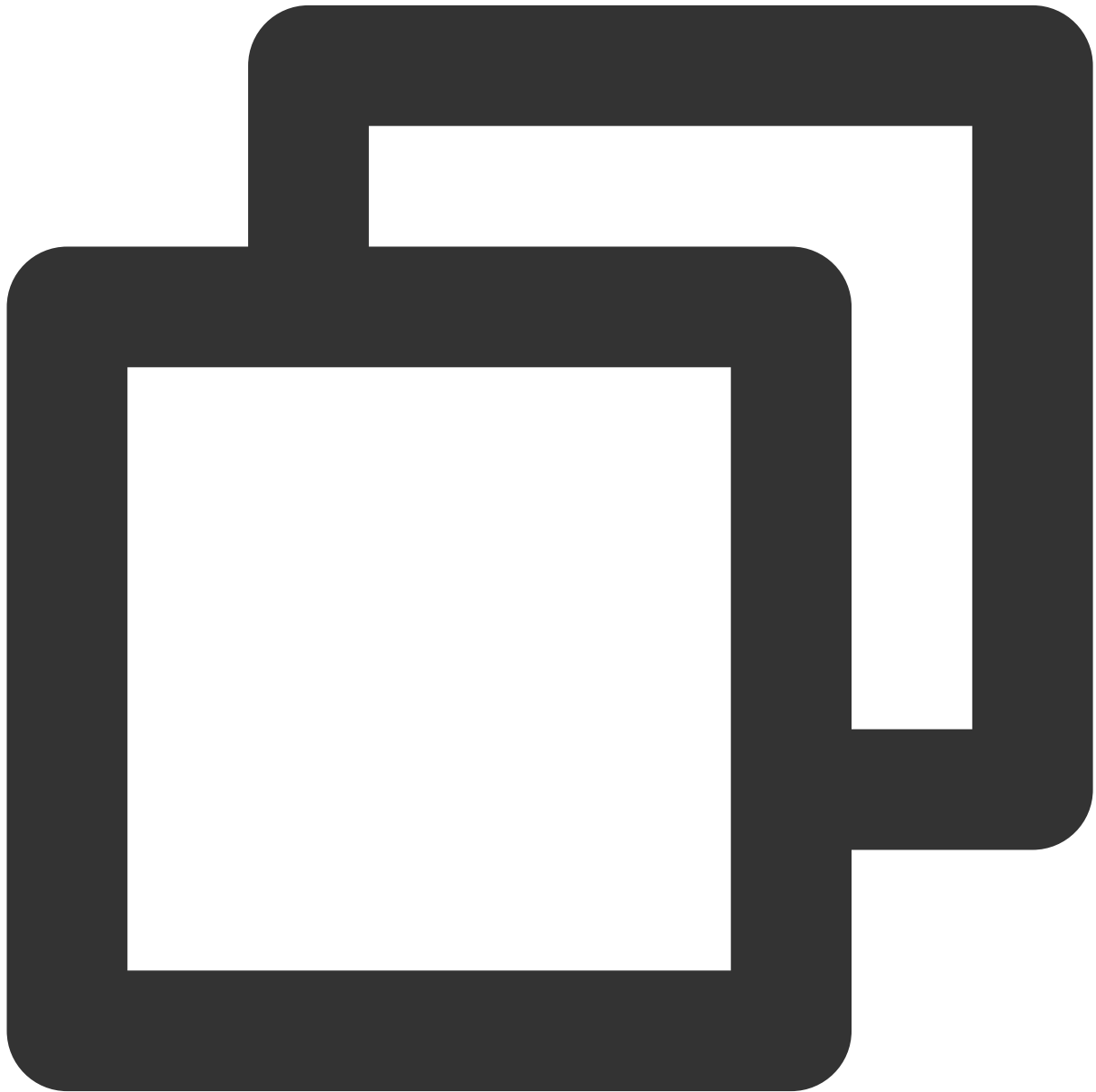
**Returns :** [TRTCDeviceInfo](#)

Device information, can get device ID and device name

### **startCameraDeviceTest**

Start camera test.





```
const roomEngine = new TUIRoomEngine();  
await roomEngine.startCameraDeviceTest({ view: 'test-preview' });
```

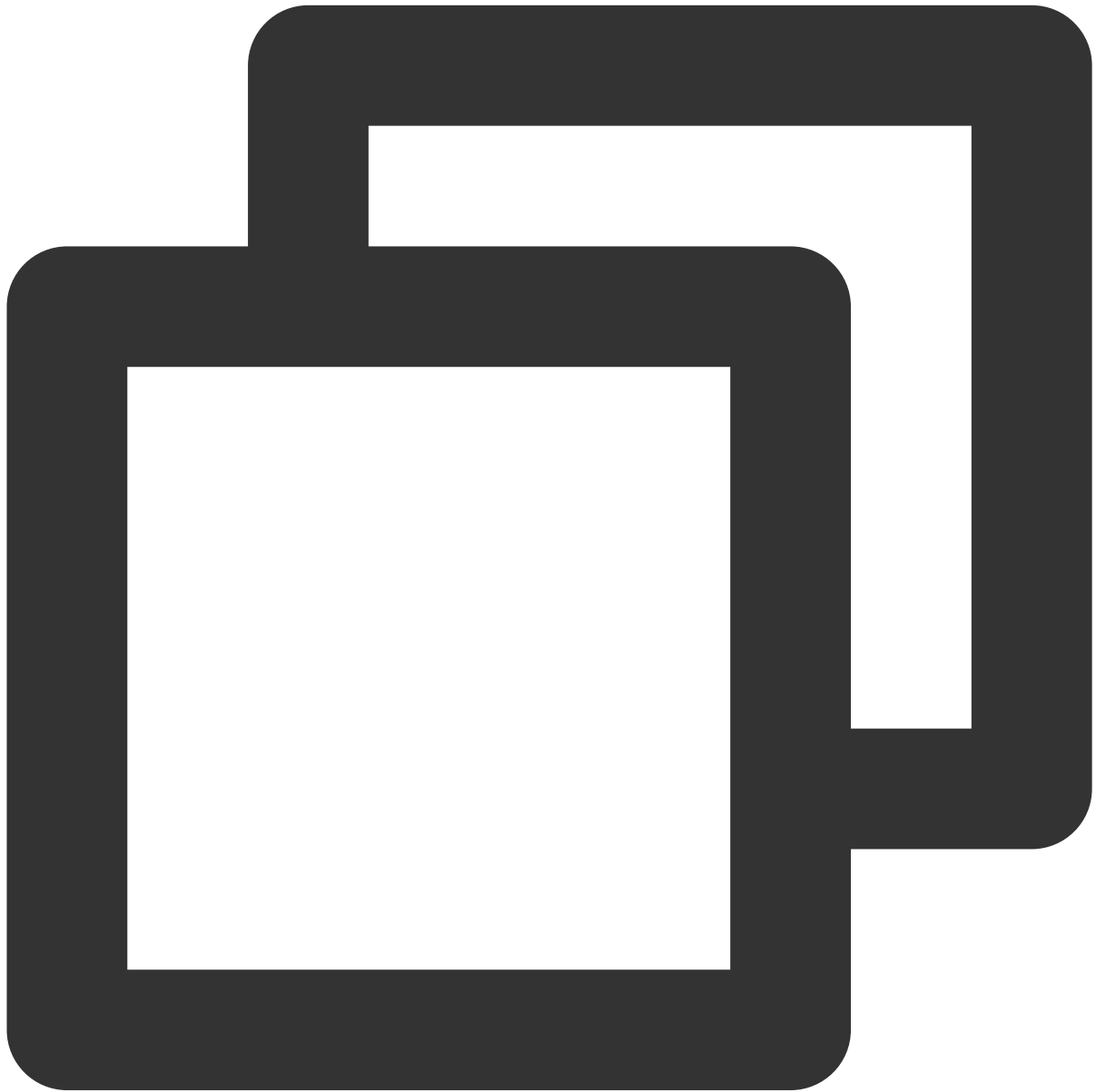
Parameter:

Parameter	Type	Description	Default Value	Meaning
view	string	Required	-	Display the video area of the camera test, the input view is the Id of the div element carrying the preview screen

**Returns** : *Promise<void>*

## **stopCameraDeviceTest**

Stop camera test.

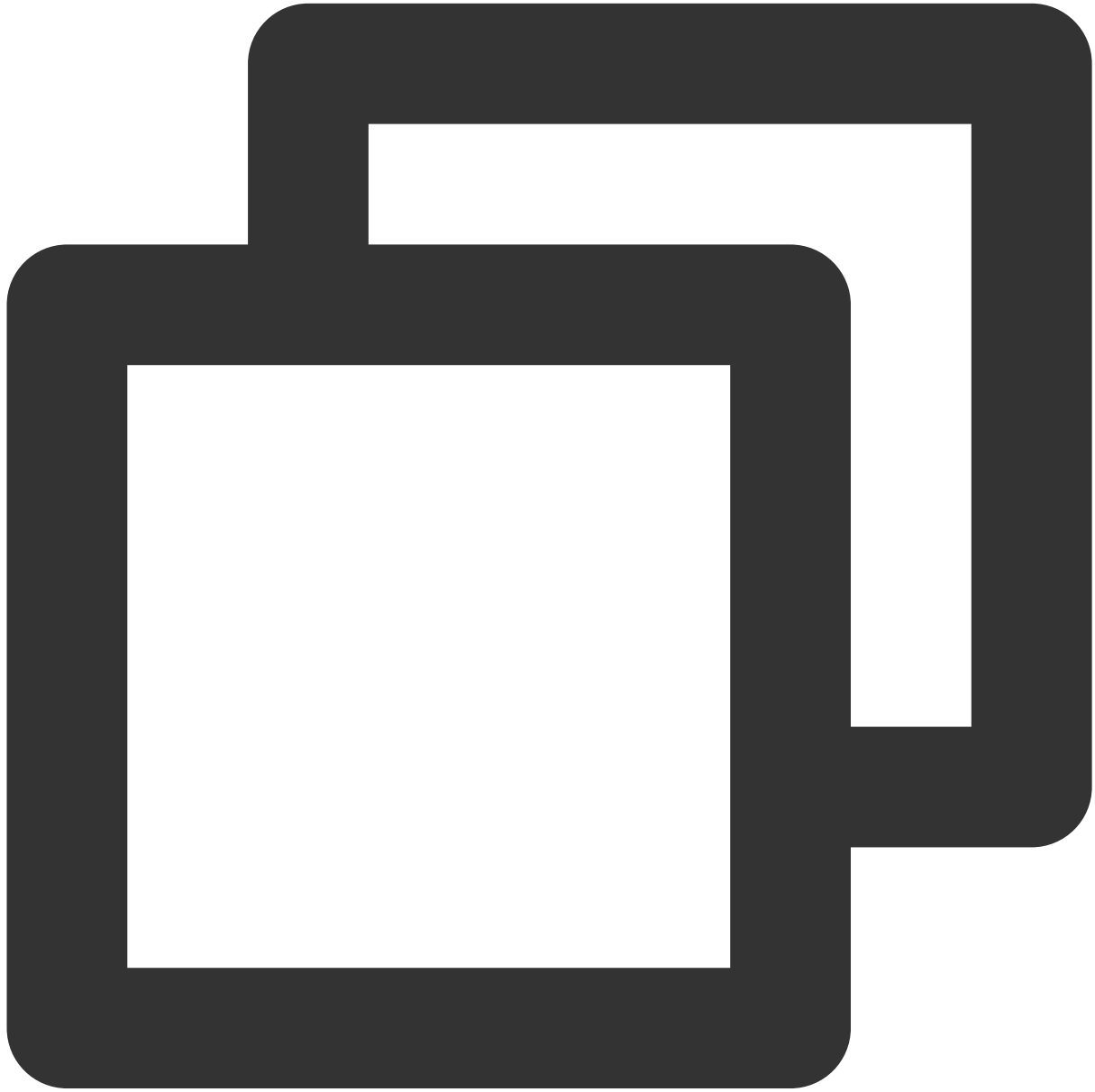


```
const roomEngine = new TUIRoomEngine();  
await roomEngine.stopCameraDeviceTest();
```

**Returns** : *Promise<void>*

**on**

Listen to roomEngine events.



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(event, func);
```

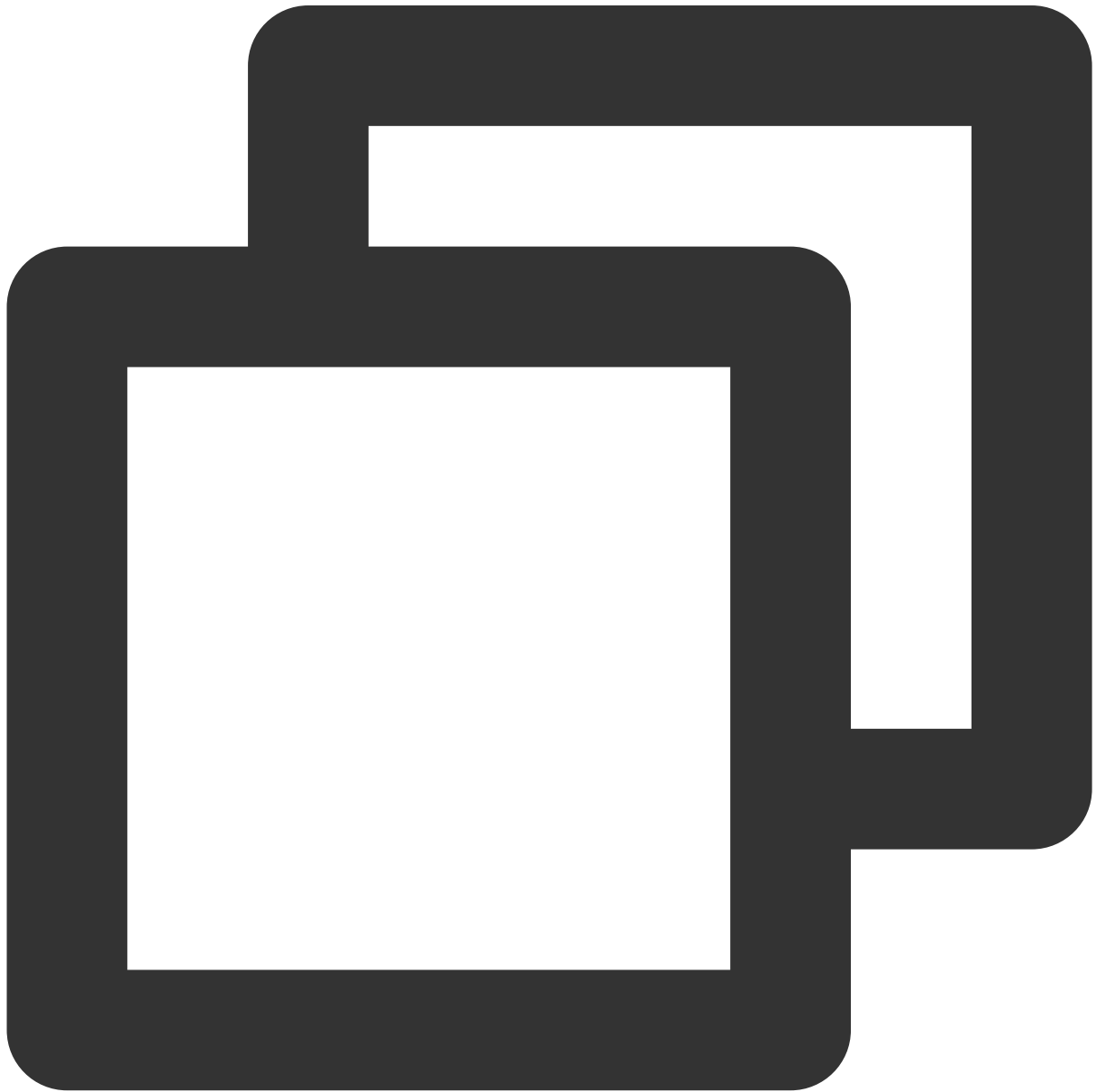
Parameter:

Parameter	Type	Description	Default Value	Meaning
event	<a href="#">TUIRoomEvents</a>	Required	-	TUIRoomEngine event list
func	Function	Required	-	Event callback function

**Returns :** *void*

## **off**

Cancel listening to roomEngine events



```
const roomEngine = new TUIRoomEngine();  
roomEngine.off(event, func);
```

Parameter:

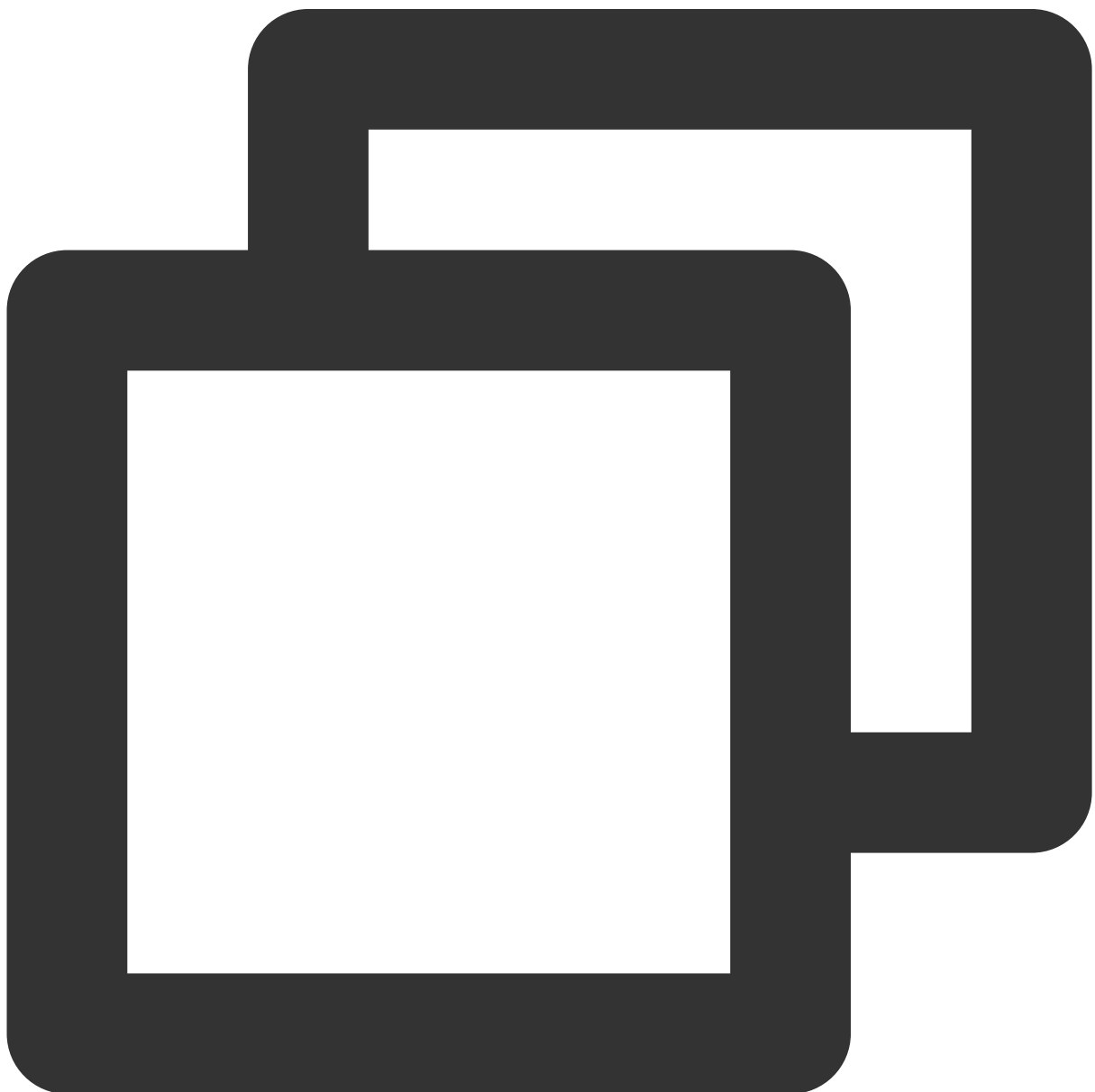
Parameter	Type	Description	Default Value	Meaning
-----------	------	-------------	---------------	---------

event	<a href="#">TUIRoomEvents</a>	Required	-	TUIRoomEngine event list
func	Function	Required	-	Event callback function

**Returns :** *void*

## getTRTCCloud

Get trtcCloud instance, for web-side trtcCloud capabilities, please check: [TRTCCloud API documentation](#).



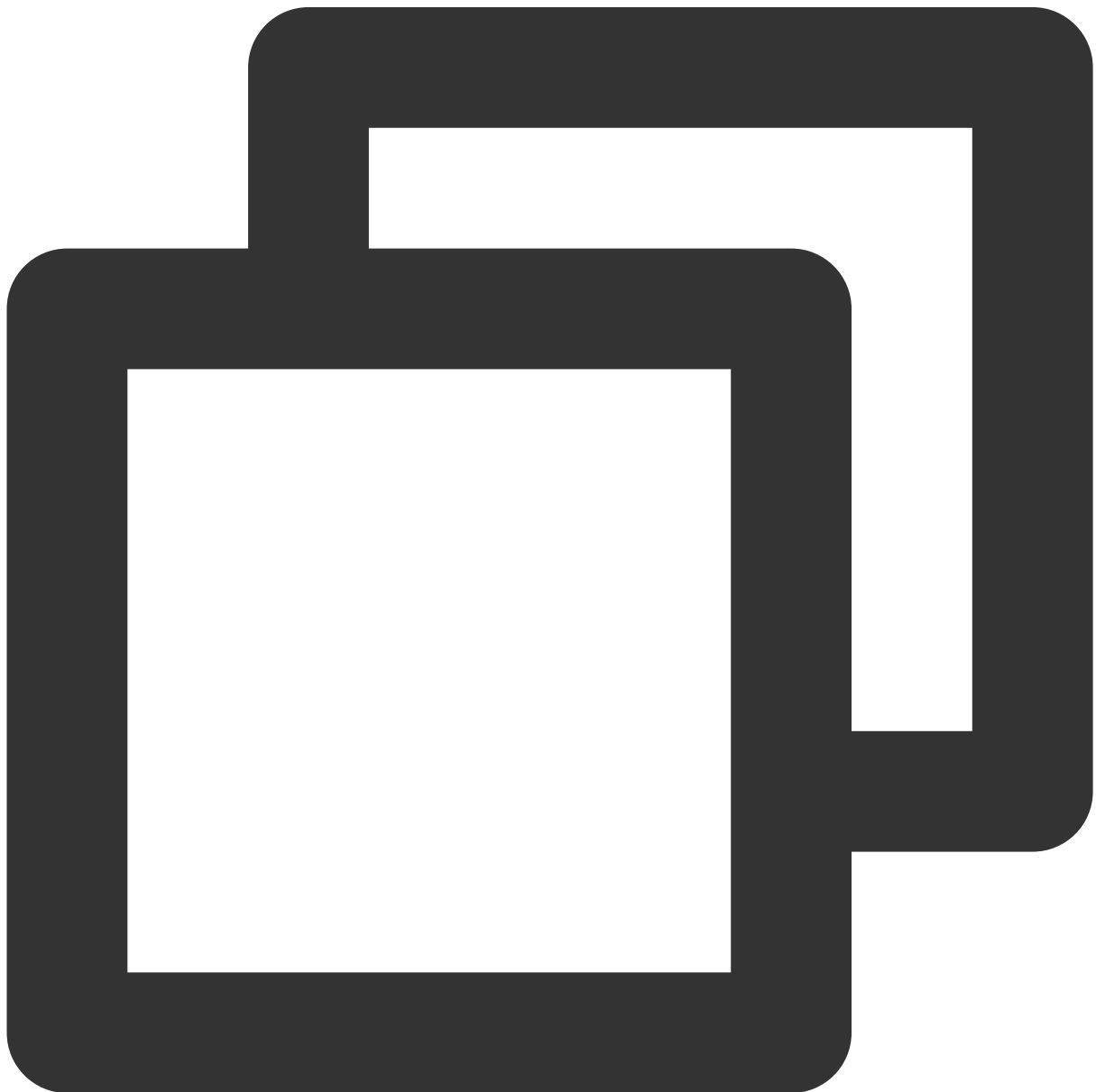
```
const roomEngine = new TUIRoomEngine();
```

```
const trtcCloud = roomEngine.getTRTCCloud();
```

**Returns :** [TRTCCloud](#)

## getTIM

Get tim instance, for web-side tim capabilities, please check: [IM API documentation](#)



```
const roomEngine = new TUIRoomEngine();  
const trtcCloud = roomEngine.getTIM();
```

**Returns :** *TIM*



# TUIRoomEvent

Last updated : 2023-11-22 11:26:33

## TUIRoomEvent API Introduction

TUIRoomEvent API is the Event Interface for Multiplayer Component.

### Event list

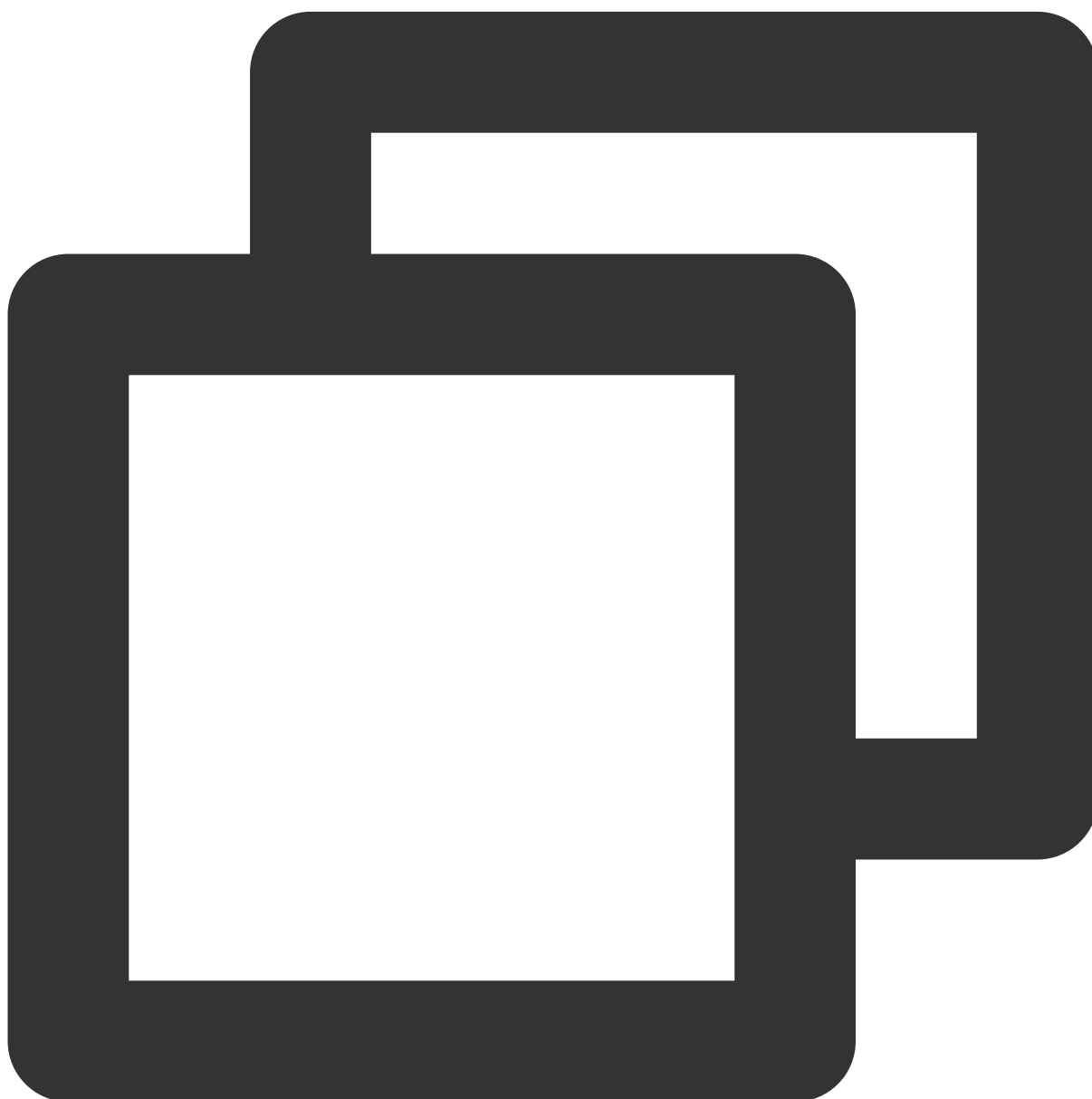
EVENT	Description
<a href="#">TUIRoomEvents.onError</a>	Error Event
<a href="#">TUIRoomEvents.onKickedOutOfRoom</a>	Kicked out of room event
<a href="#">TUIRoomEvents.onKickedOffLine</a>	Current user Kicked off line event
<a href="#">TUIRoomEvents.onUserSigExpired</a>	UserSig Expiration Event
<a href="#">TUIRoomEvents.onRoomDismissed</a>	Host Destroy Room Event
<a href="#">TUIRoomEvents.onRoomNameChanged</a>	Room Name Change Event
<a href="#">TUIRoomEvents.onRoomSpeechModeChanged</a>	Room Speech Mode Change Event
<a href="#">TUIRoomEvents.onAllUserCameraDisableChanged</a>	All members Camera Usage Permission Change Event
<a href="#">TUIRoomEvents.onAllUserMicrophoneDisableChanged</a>	All members Mic Usage Permission Change Event
<a href="#">TUIRoomEvents.onSendMessageForAllUserDisableChanged</a>	All members Send Message Status Change Event
<a href="#">TUIRoomEvents.onRoomMaxSeatCountChanged</a>	Room Maximum Seat Count Change Event
<a href="#">TUIRoomEvents.onRemoteUserEnterRoom</a>	Remote User Entered Room Event
<a href="#">TUIRoomEvents.onRemoteUserLeaveRoom</a>	Remote User Leave Room Event
<a href="#">TUIRoomEvents.onUserRoleChanged</a>	User Role Change Event
<a href="#">TUIRoomEvents.onUserVideoStateChanged</a>	User Video Status Change Event



<a href="#">TUIRoomEvents.onUserAudioStateChanged</a>	User Audio Status Change Event
<a href="#">TUIRoomEvents.onSendMessageForUserDisableChanged</a>	User Send Message Status Event
<a href="#">TUIRoomEvents.onUserVoiceVolumeChanged</a>	User Volume Change Event
<a href="#">TUIRoomEvents.onUserNetworkQualityChanged</a>	User Network Quality Change Event
<a href="#">TUIRoomEvents.onSeatListChanged</a>	Seat List Change Event
<a href="#">TUIRoomEvents.onKickedOffSeat</a>	User Kicked off Seat Event
<a href="#">TUIRoomEvents.onRequestReceived</a>	Request Received Event
<a href="#">TUIRoomEvents.onRequestCancelled</a>	Request Cancelled Event
<a href="#">TUIRoomEvents.onReceiveTextMessage</a>	Receive Text Message Event
<a href="#">TUIRoomEvents.onReceiveCustomMessage</a>	Receive Custom Message Event
<a href="#">TUIRoomEvents.onDeviceChange</a>	Device Change Event
<a href="#">TUIRoomEvents.onUserScreenCaptureStopped</a>	Screen Sharing Stopped Event When the user uses the built-in browser stop sharing button to end screen sharing, the user will receive the 'onUserScreenCaptureStopped' event to modify the screen sharing status.

## onError

Error Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onError, (error) => {
  console.log('TUIRoomError error', error);
})
```

The parameters are shown in the table below:

Parameter	Type	Meaning
code	number	Error Code

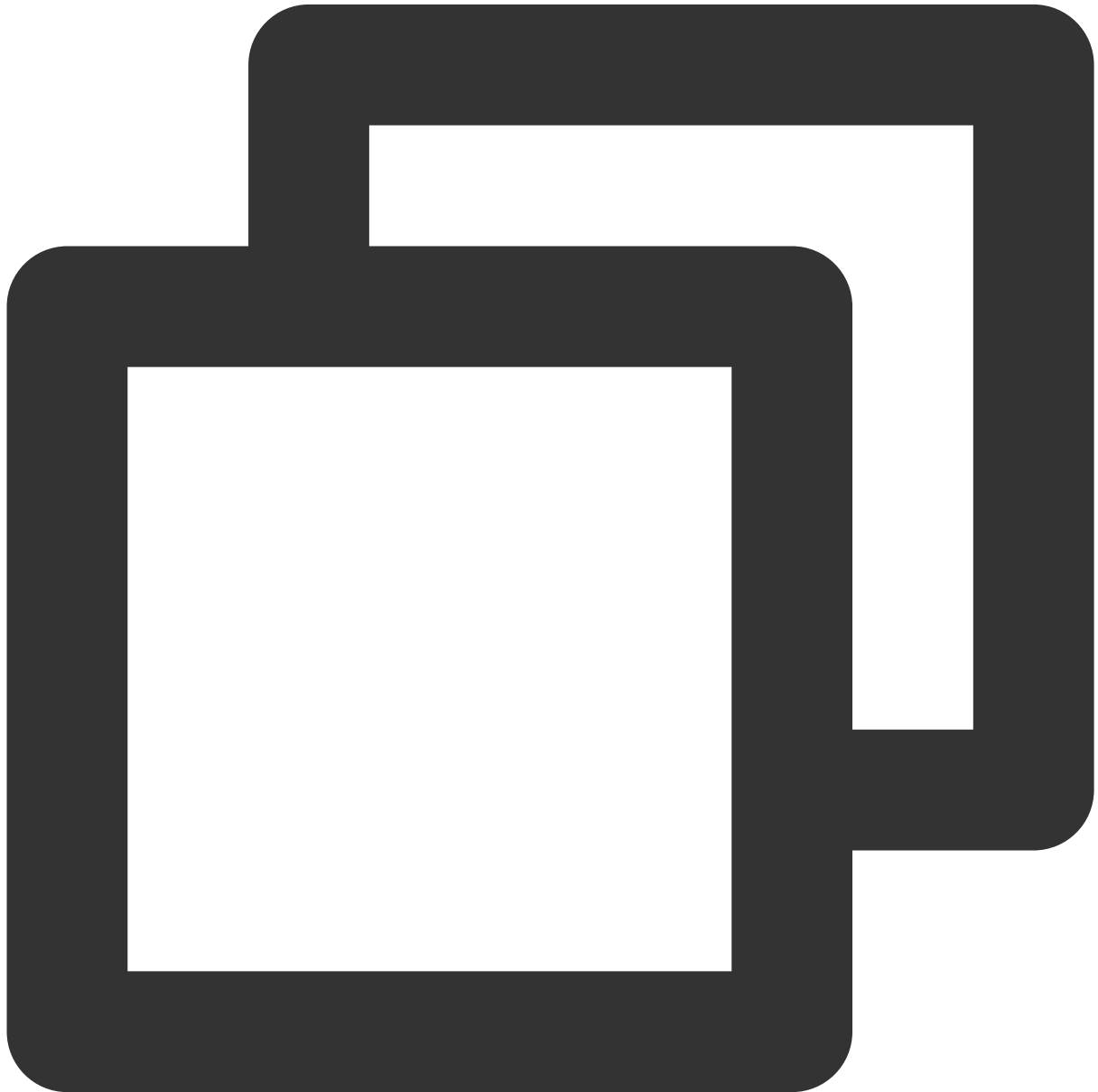
message

string

Error Information

## onKickedOutOfRoom

Kicked out of room event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onKickedOutOfRoom, ({ roomId, message }) => {
  console.log('roomEngine.onKickedOutOfRoom', roomId, message);
});
```

```
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
message	string	Kicked out of room information

## onKickedOffLine

Current user Kicked off line event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onKickedOffLine, ({ message }) => {
  console.log('roomEngine.onKickedOffLine', message);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID

message

string

User logged in on other end information

## onUserSigExpired

UserSig Expiration Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserSigExpired, () => {
  console.log('roomEngine.onUserSigExpired');
});
```

```
});
```

## onRoomDismissed

Host Destroy Room Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomDismissed, ({ roomId }) => {
  console.log('roomEngine.onRoomDismissed', roomId);
});
```

```
});
```

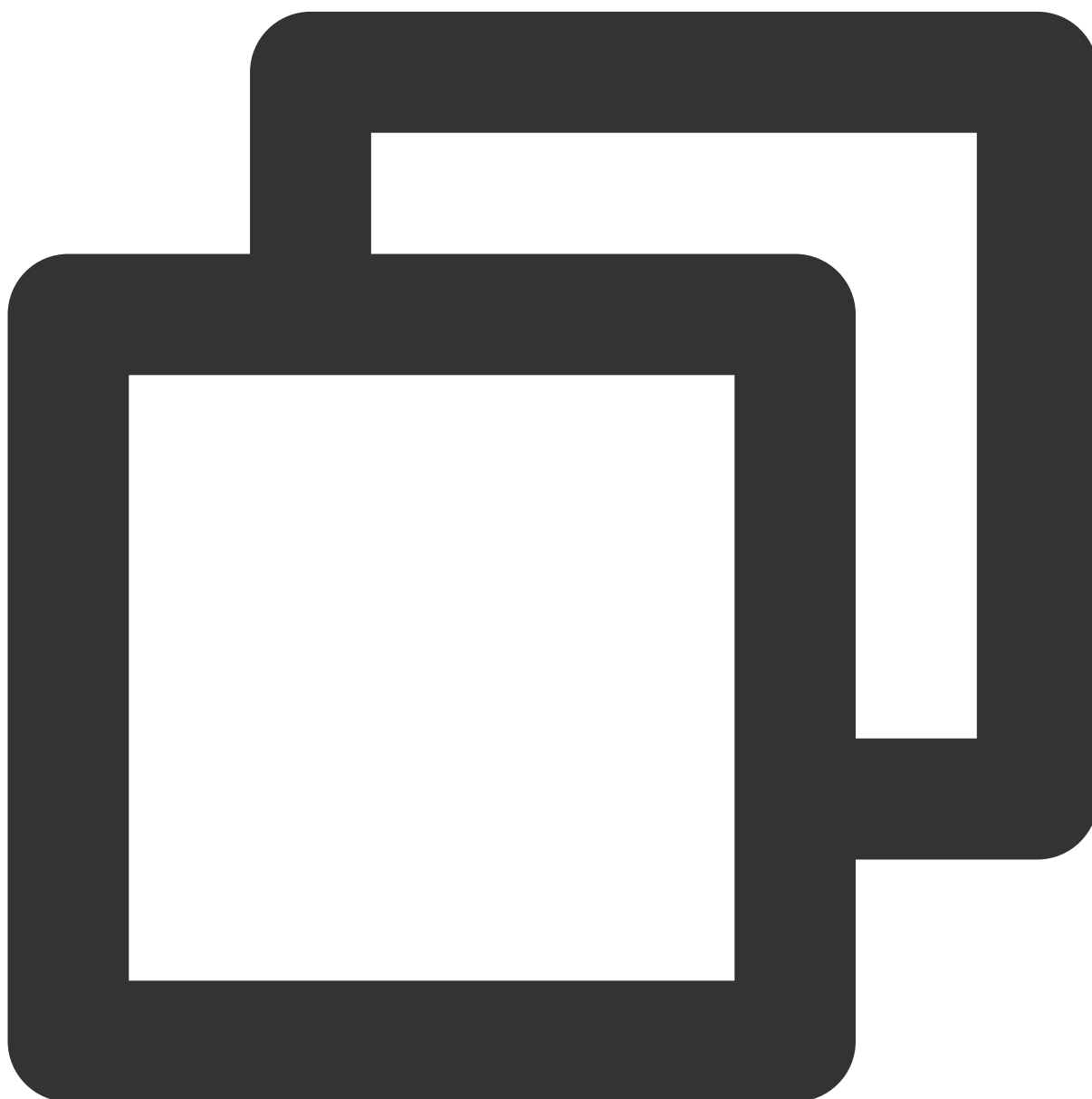
The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID

## onRoomNameChanged

Room ID Modification Event





```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomNameChanged, ({ roomId, roomName }) => {
  console.log('roomEngine.onRoomNameChanged', roomId, roomName);
});
```

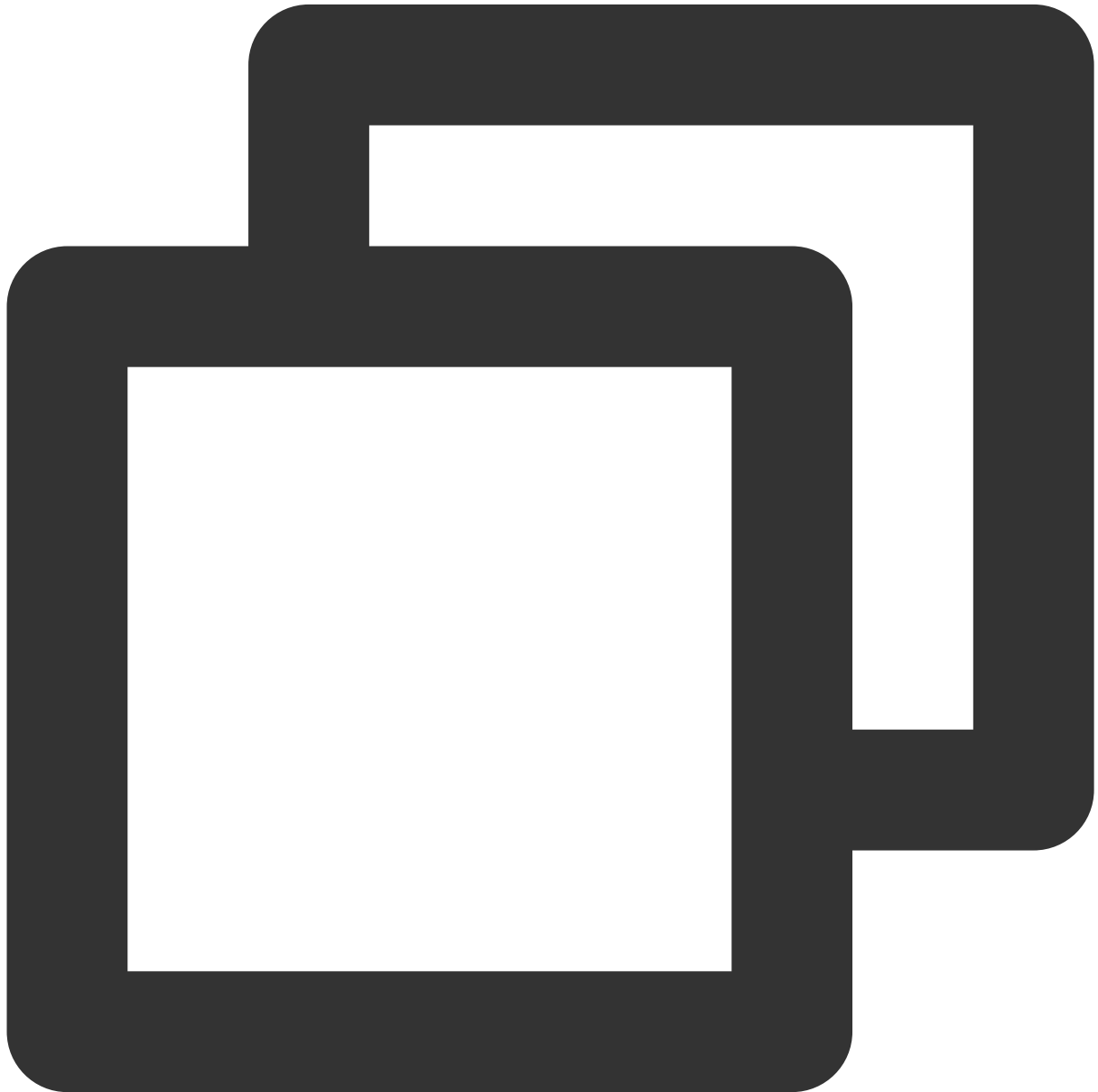
The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID

roomName	string	Room Name
----------	--------	-----------

## onRoomSpeechModeChanged

Room Name Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomSpeechModeChanged, ({ roomId, speechMode }) => {
  console.log('roomEngine.onRoomSpeechModeChanged', roomId, speechMode);
});
```

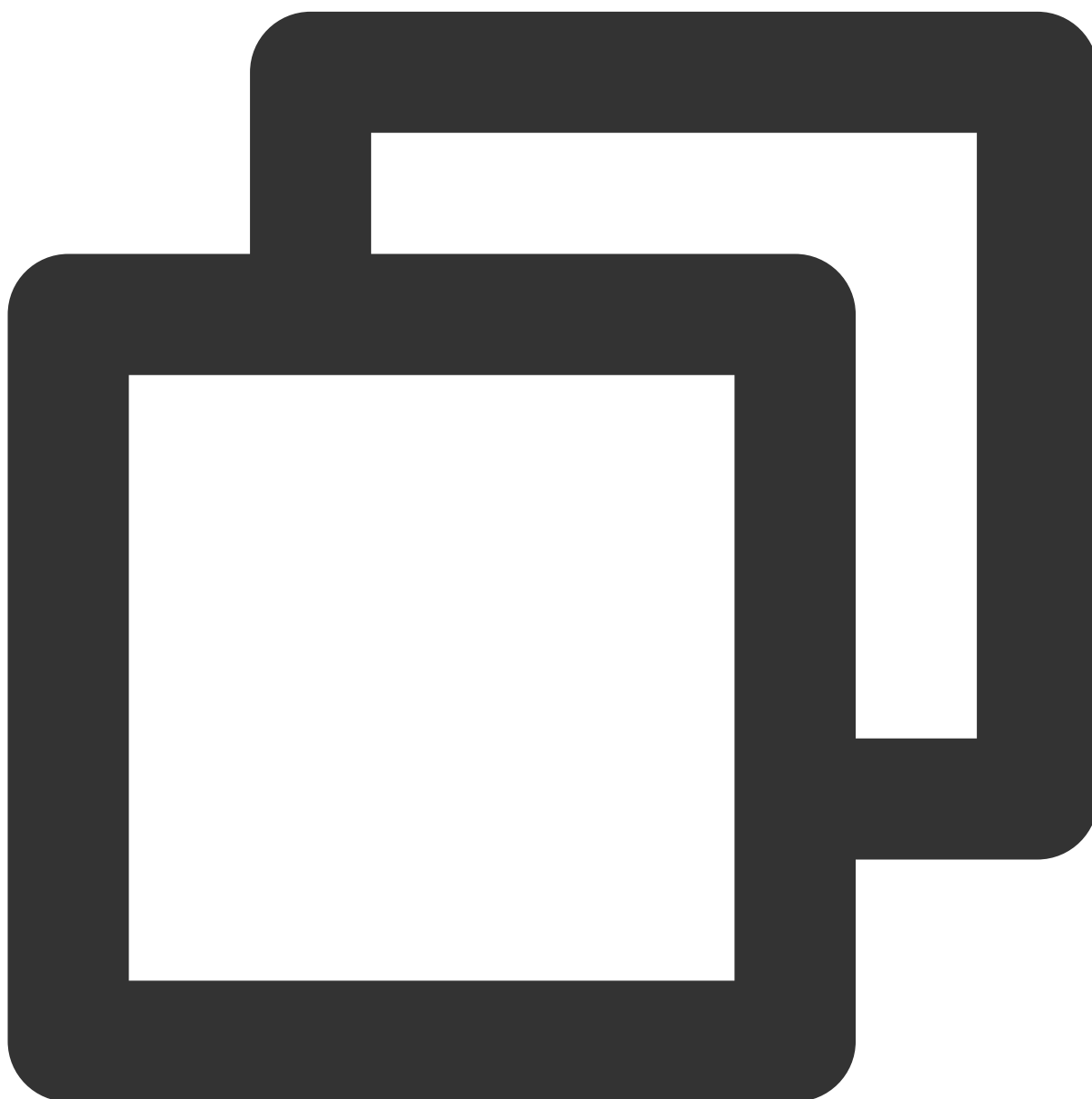
```
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
speechMode	<a href="#">TUISpeechMode</a>	Speech Mode

## onAllUserCameraDisableChanged

All members Camera Usage Permission Change Event



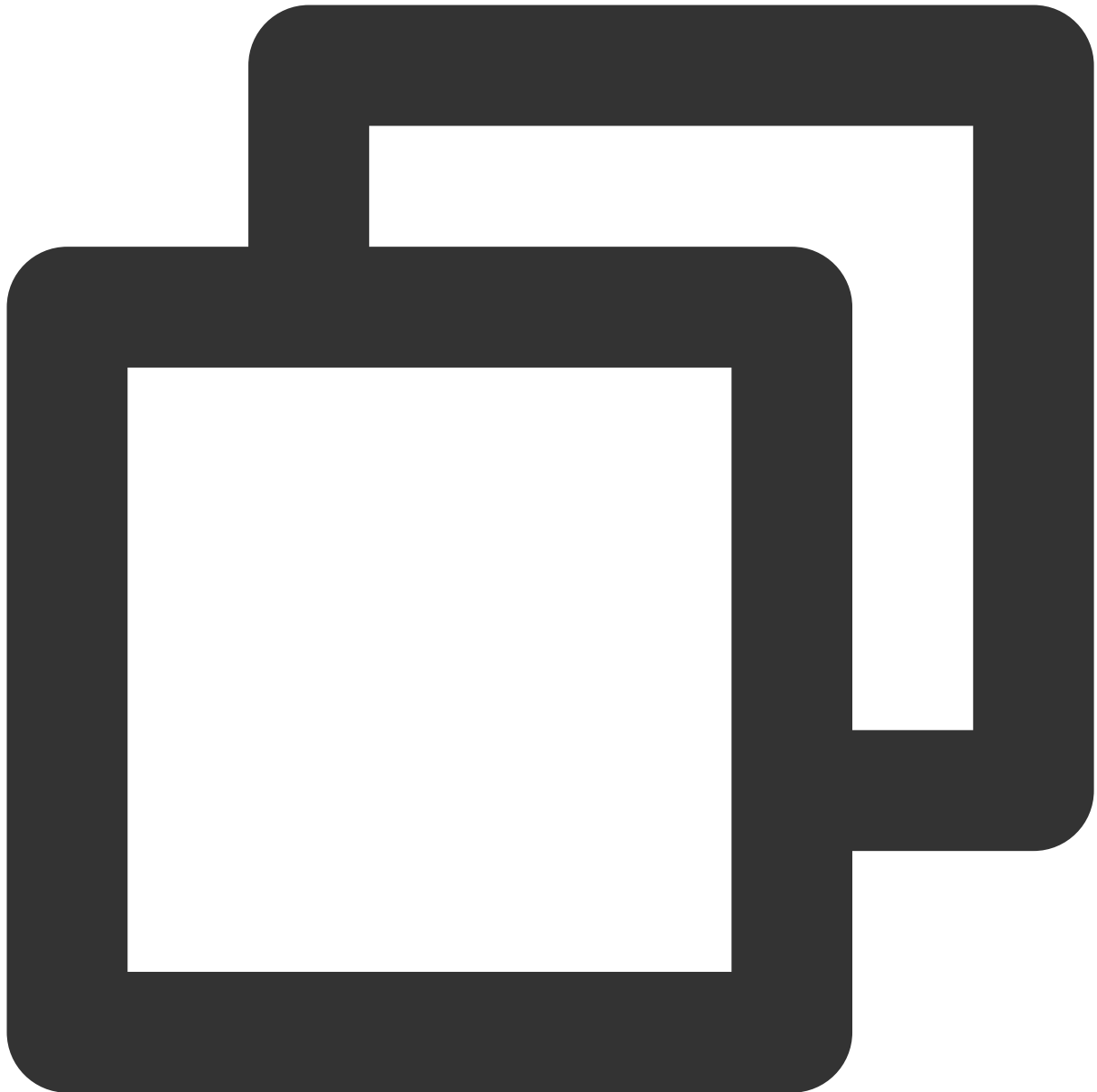
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onAllUserCameraDisableChanged, ({ isDisable }) => {
  console.log('roomEngine.onAllUserCameraDisableChanged', isDisable);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
isDisable	boolean	Allow Camera Usage

## onAllUserMicrophoneDisableChanged

All members Mic Usage Permission Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onAllUserMicrophoneDisableChanged, ({ isDisable }) => {
  console.log('roomEngine.onAllUserMicrophoneDisableChanged', isDisable);
});
```

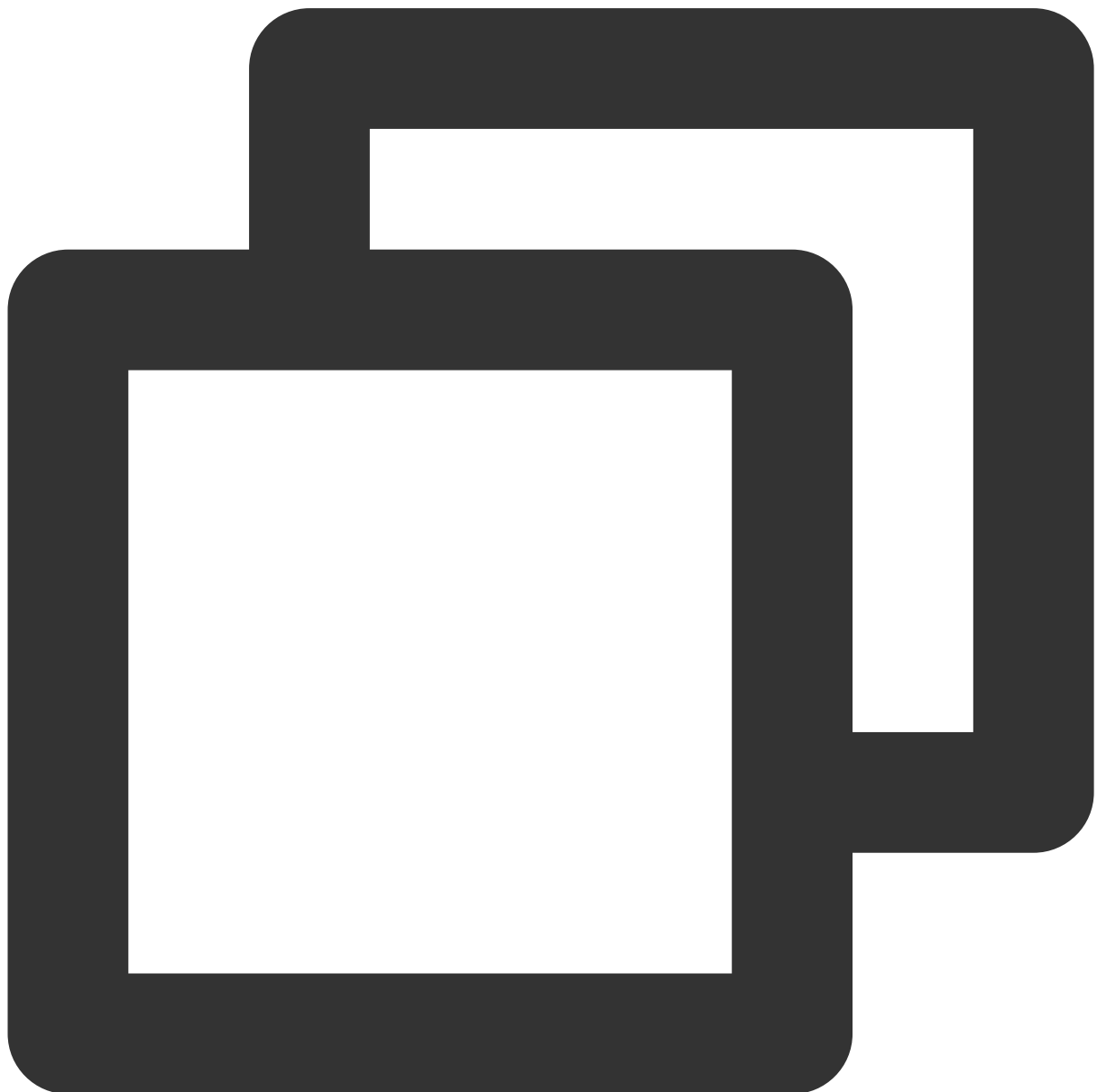
The parameters are shown in the table below:

--	--	--

Parameter	Type	Meaning
isDisable	boolean	Allow Mic Usage

## onSendMessageForAllUserDisableChanged

All members Send Message Permission Change Event



```
const roomEngine = new TUIRoomEngine();
```

```
roomEngine.on(TUIRoomEvents.onSendMessageForAllUserDisableChanged, ({ isDisable })  
    console.log('roomEngine.onSendMessageForAllUserDisableChanged', isDisable);  
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
isDisable	boolean	Allow Sending Text Message

## onRoomMaxSeatCountChanged

Room Maximum Seat Count Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomMaxSeatCountChanged, ({ maxSeatNumber }) => {
  console.log('roomEngine.onRoomMaxSeatCountChanged', maxSeatNumber);
});
```

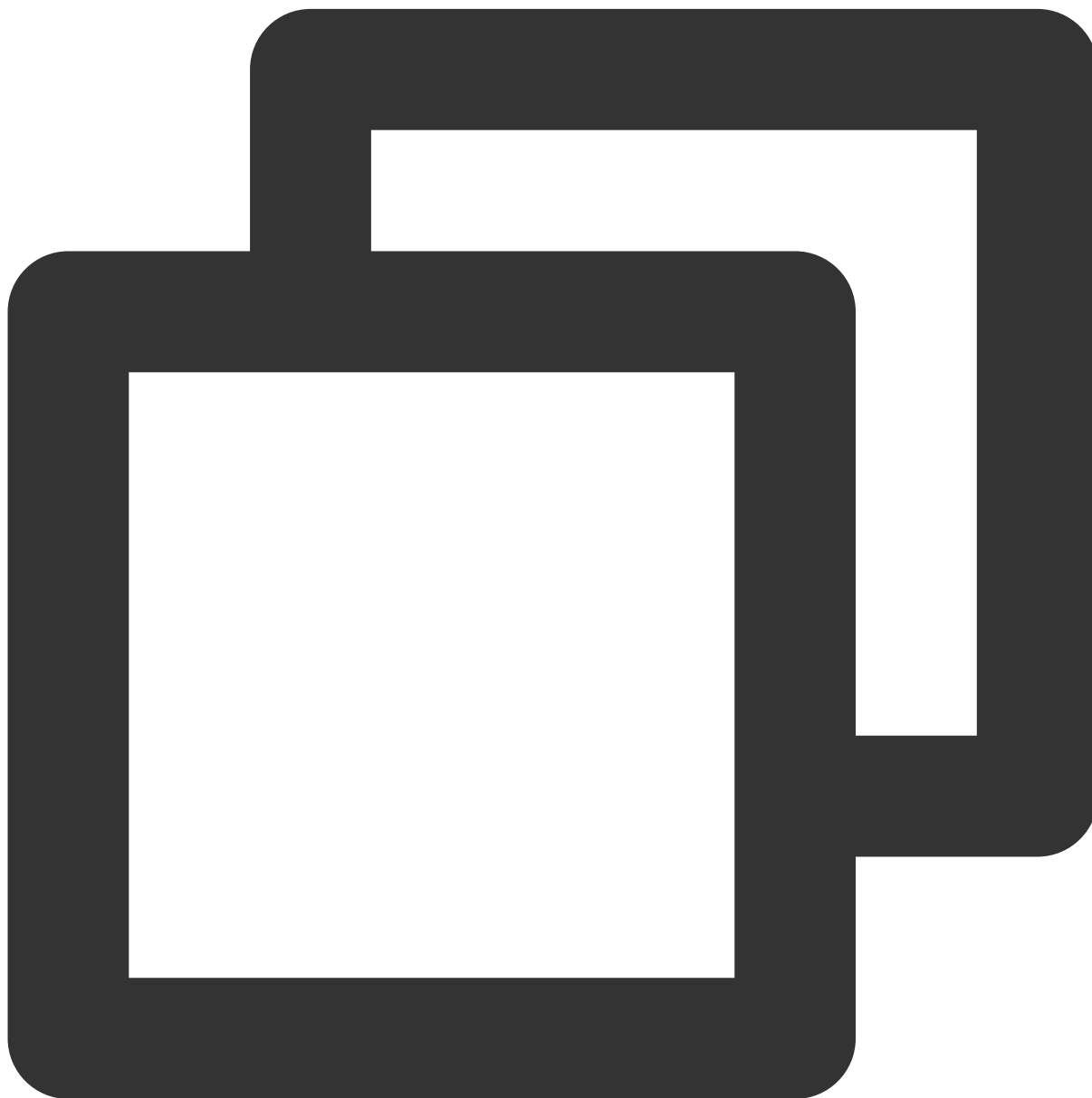
The parameters are shown in the table below:

Parameter	Type	Meaning
maxSeatNumber	number	Maximum Seat Count



## onRemoteUserEnterRoom

Remote User Entered Room Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRemoteUserEnterRoom, ({ roomId, userInfo }) => {
  console.log('roomEngine.onRemoteUserEnterRoom', roomId, userInfo);
});
```

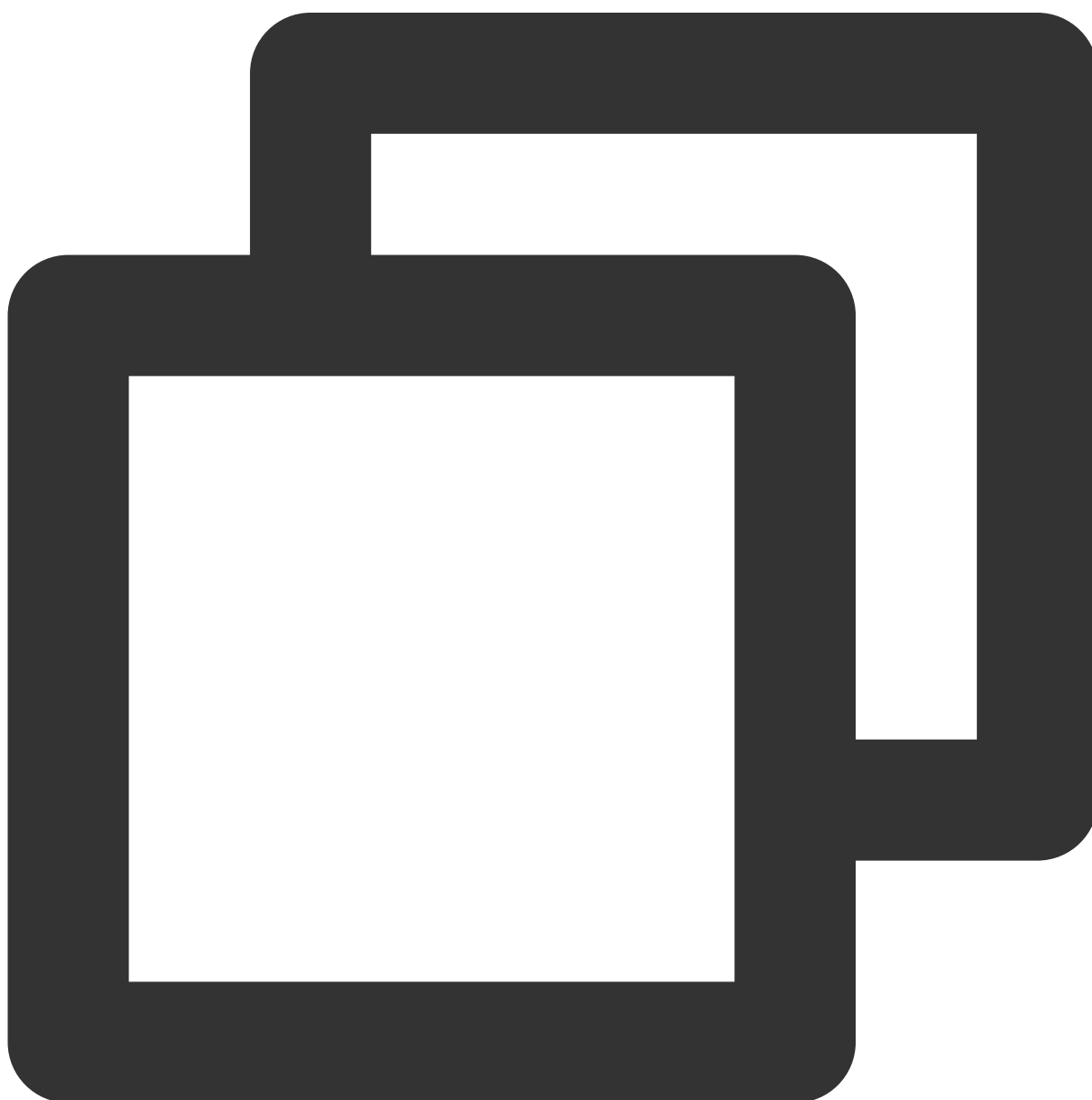
The parameters are shown in the table below:

--	--	--

Parameter	Type	Meaning
roomId	string	Room ID
userInfo	<a href="#">TUIUserInfo</a>	User Information

## onRemoteUserLeaveRoom

Remote User Leave Room Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRemoteUserLeaveRoom, ({ roomId, userInfo }) => {
  console.log('roomEngine.onRemoteUserLeaveRoom', roomId, userInfo);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room ID
userInfo	<a href="#">TUIUserInfo</a>	User Information

## onUserRoleChanged

User Role Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserRoleChanged, ({ userId, userRole }) => {
  console.log('roomEngine.onUserRoleChanged', userId, userRole);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
userId	string	User Id

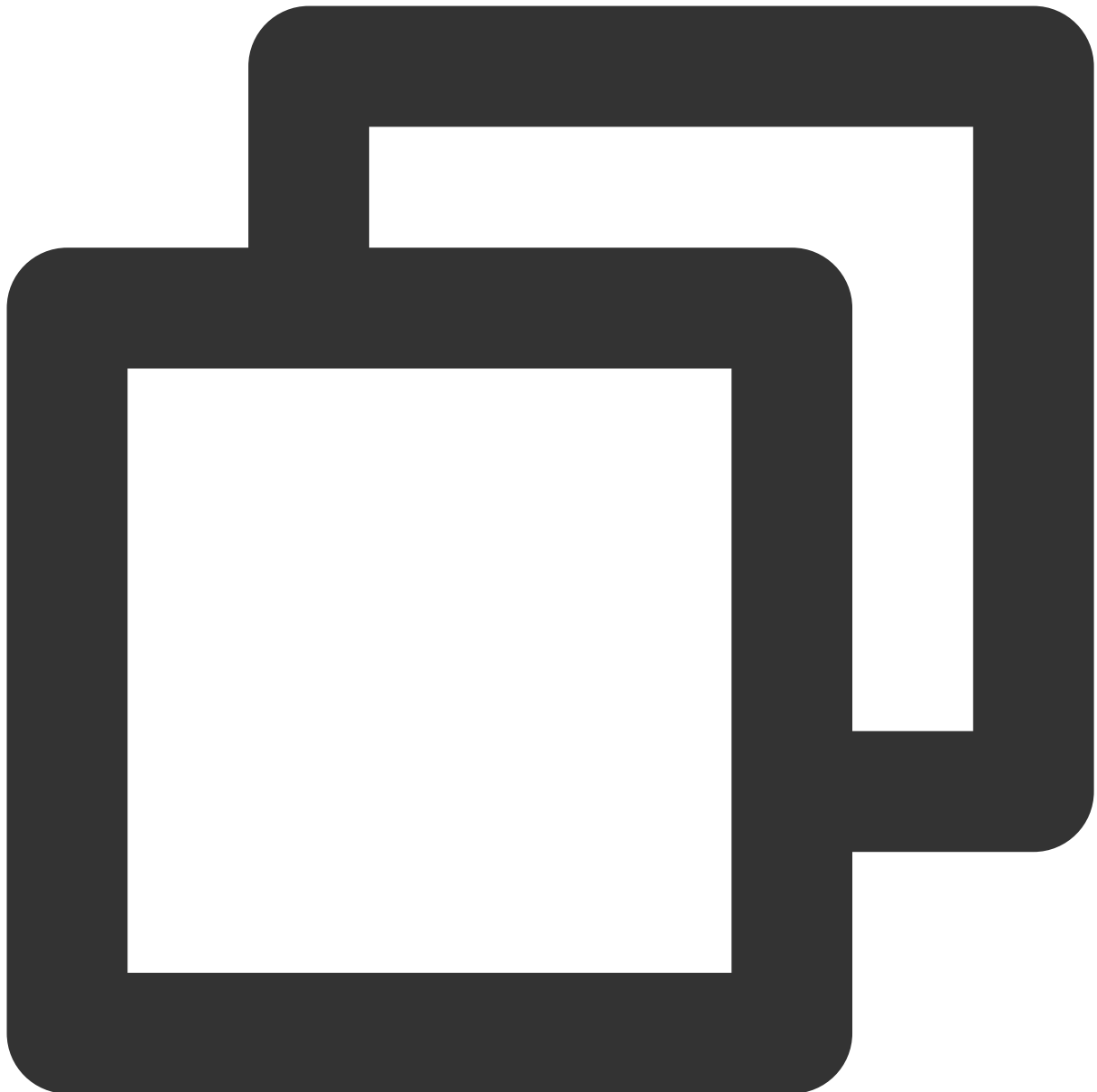
userRole

TUIRole

User Changed Role

## onUserVideoStateChanged

User Video Status Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserVideoStateChanged, ({ userId, streamType, hasVideo, ...data }) => {
  console.log('roomEngine.onUserVideoStateChanged', userId, streamType, hasVideo, ...data);
});
```

```
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
userId	string	User Id
streamType	<a href="#">TUIVideoStreamType</a>	User Stream Type
hasVideo	boolean	Has Video streams
reason	<a href="#">TUIChangeReason</a>	Change Reason, Self-operation/Host-operation

## onUserAudioStateChanged

User Audio Status Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserAudioStateChanged, ({ userId, hasAudio, reason })
  console.log('roomEngine.onUserAudioStateChanged', userId, hasAudio, reason);
});
```

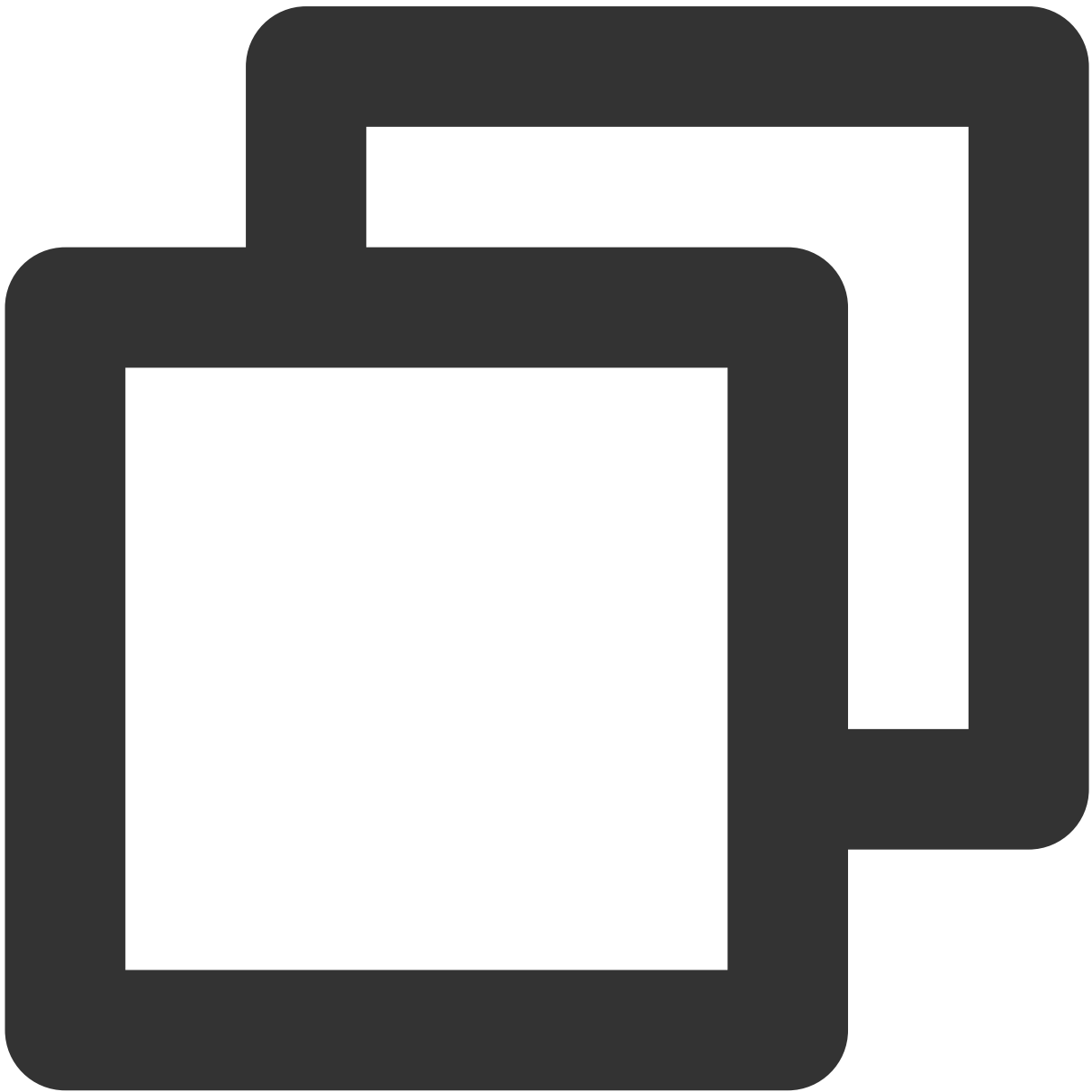
The parameters are shown in the table below:

Parameter	Type	Meaning
userId	string	User Id

hasVideo	boolean	Has Audio streams
reason	<a href="#">TUIChangeReason</a>	Change Reason, Self-operation/Host-operation

## onSendMessageForUserDisableChanged

Member Camera Usage Permission Change Event





```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onSendMessageForAllUserDisableChanged, ({ isDisable })
  console.log('roomEngine.onSendMessageForAllUserDisableChanged', isDisable);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
isDisable	boolean	Allow Sending Text Message

## onUserVoiceVolumeChanged

User Volume Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserVoiceVolumeChanged, ({ userVolumeList }) => {
  userVolumeList.forEach(userVolume => {
    console.log('roomEngine.onUserVoiceVolumeChanged', userVolume.userId, userVolume);
  })
});
```

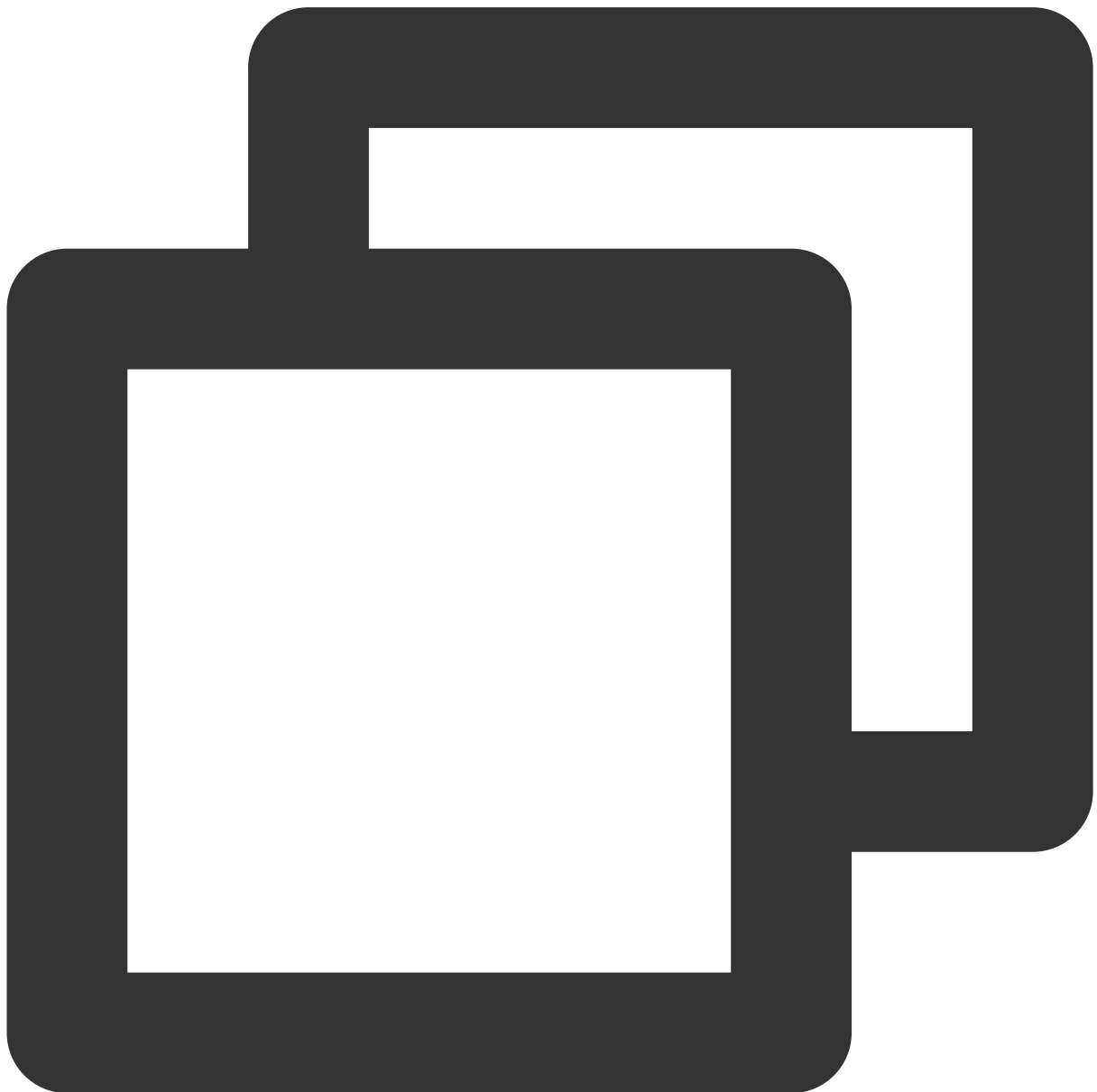
The parameters are shown in the table below:

Parameter	Type	Meaning

userVolumes	Array.<TRTCVolumeInfo>	Volume of all users in the room, including userId and volume information, volume range is 1-100
-------------	------------------------	---

## onUserNetworkQualityChanged

User Network Quality Change Event



```
const roomEngine = new TUIRoomEngine();
```

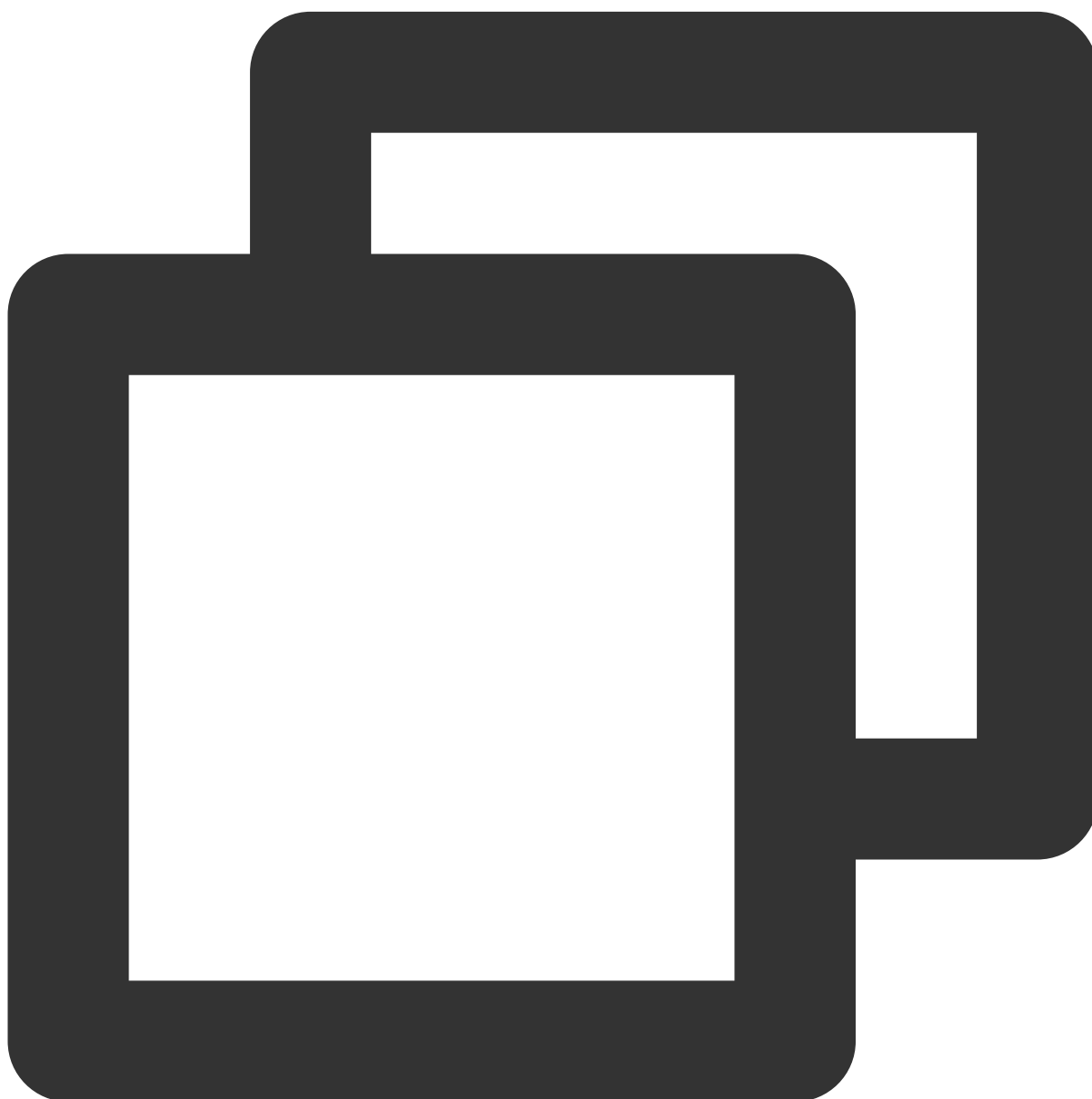
```
roomEngine.on(TUIRoomEvents.onUserNetworkQualityChanged, ({ userNetworkList }) => {
  userNetworkList.forEach(userNetwork => {
    console.log('roomEngine.onUserNetworkQualityChanged', userNetwork.userId, userN
  })
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
networkMap	<a href="#">TUINetworkQuality</a>	Traverse Network Quality Level

## onSeatListChanged

Seat List Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onSeatListChanged, ({ seatList, seatedList, leftList })
  console.log('roomEngine.onSeatListChanged',seatList, seatedList, leftList);
});
```

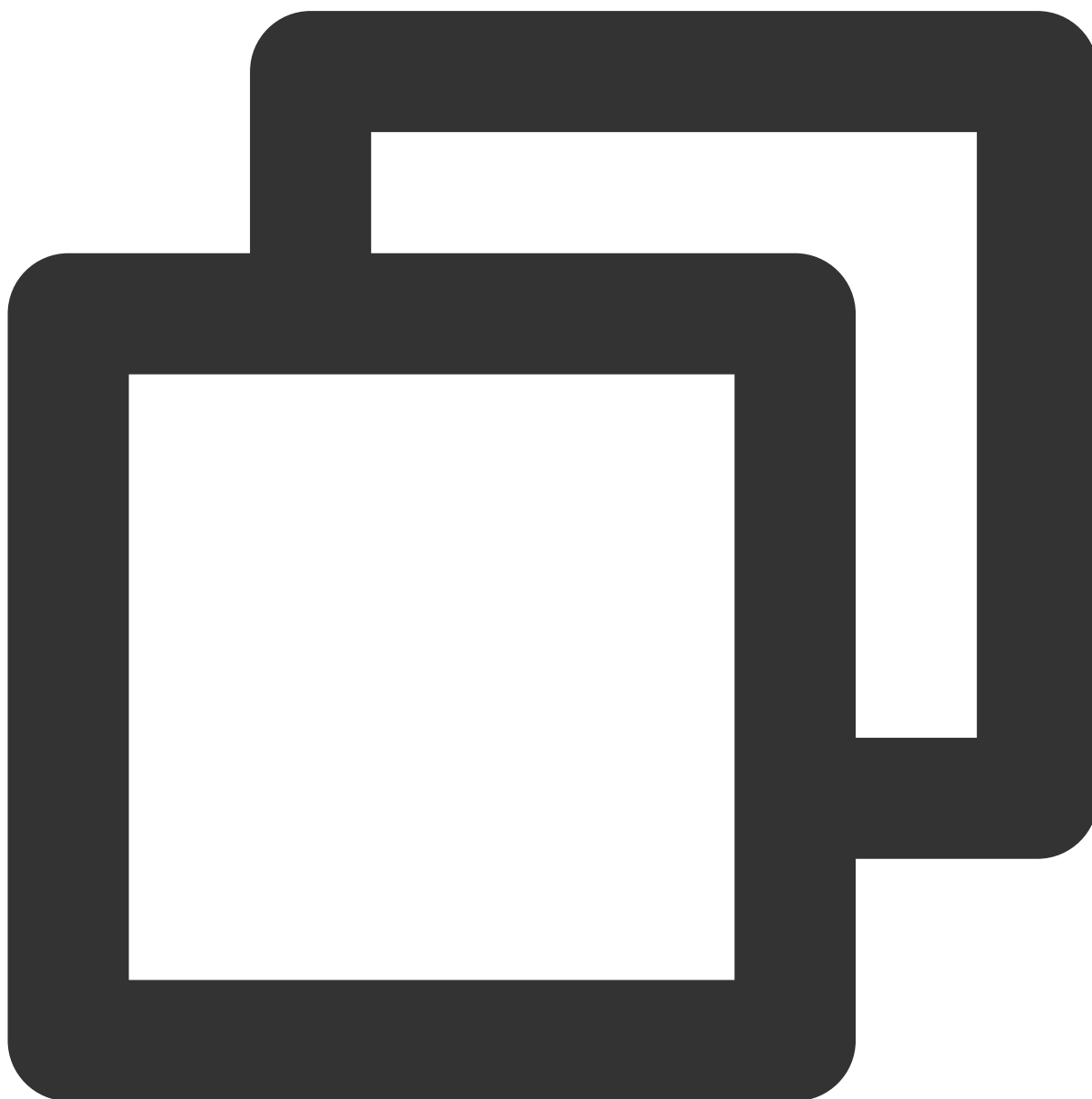
The parameters are shown in the table below:

Parameter	Type	Meaning
seatList	Array.< <a href="#">TUISeatInfo</a> >	Seat List

seatedList	Array.<TUISeatInfo>	New Seat Information
leftList	Array.<TUISeatInfo>	Left Seat Information

## onKickedOffSeat

Seat List Change Event



```
const roomEngine = new TUIRoomEngine();
```

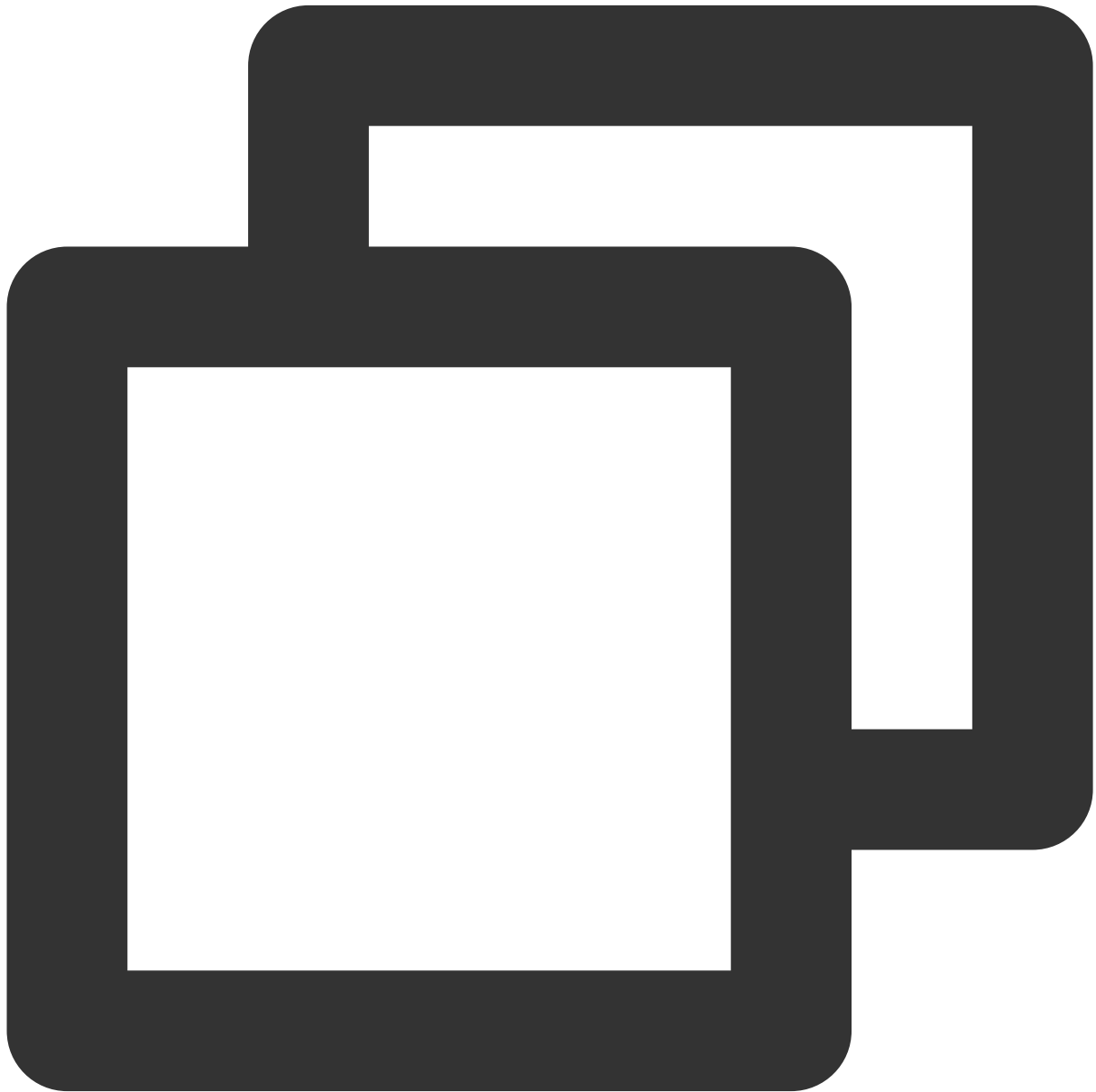
```
roomEngine.on(TUIRoomEvents.onKickedOffSeat, ({ userId }) => {  
    console.log('roomEngine.onKickedOffSeat', userId);  
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
userId	String	Kicked off Seat User Id

## onRequestReceived

Error Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRequestReceived, ({ request }) => {
  console.log('roomEngine.onRequestReceived', request);
});
```

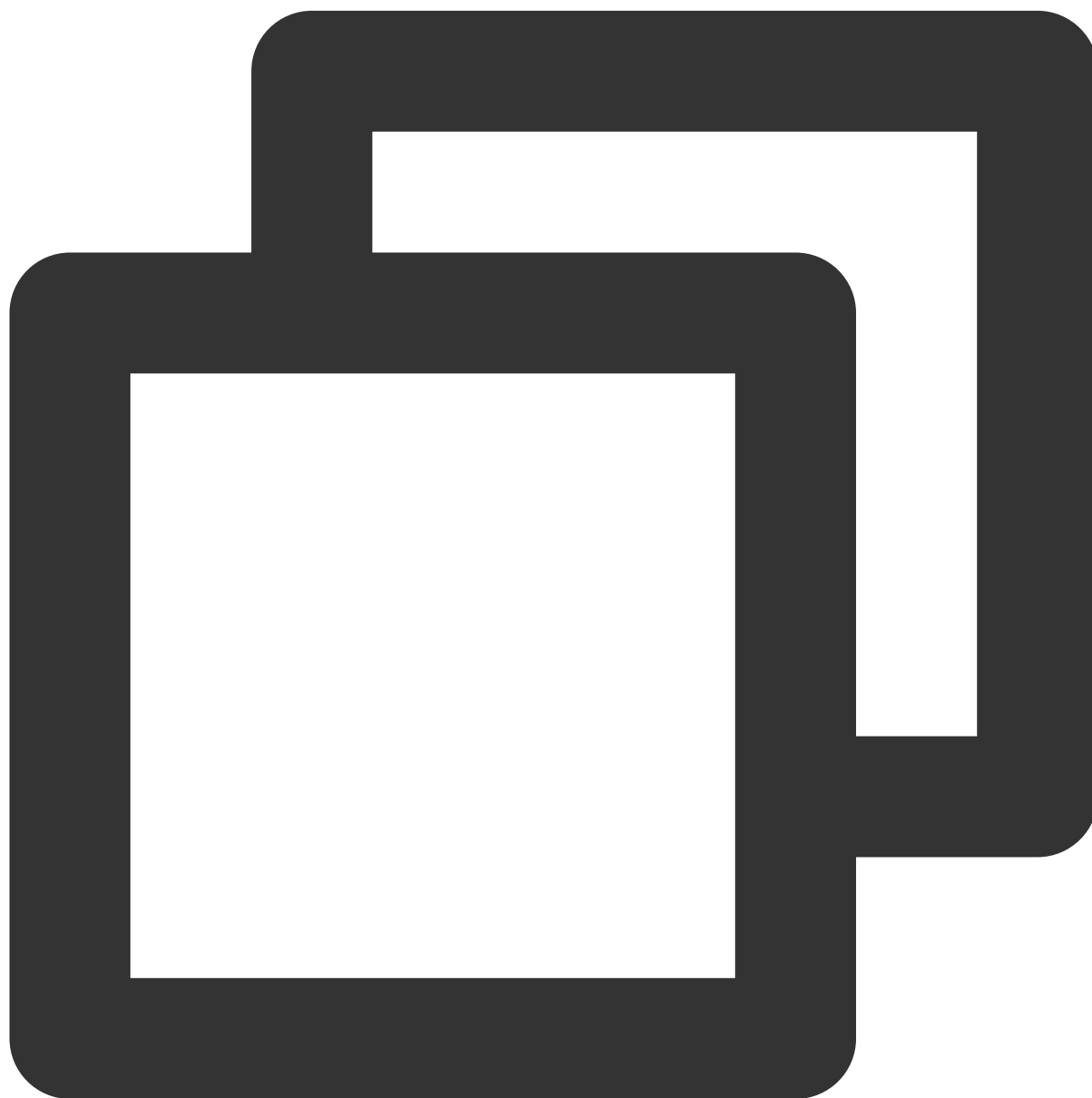
The parameters are shown in the table below:

Parameter	Type	Meaning
request	<a href="#">TUIRequest</a>	Request Received



# onRequestCancelled

Request Cancelled Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRequestCancelled, ({ requestId, userId }) => {
  console.log('roomEngine.onRequestCancelled', requestId, userId);
});
```

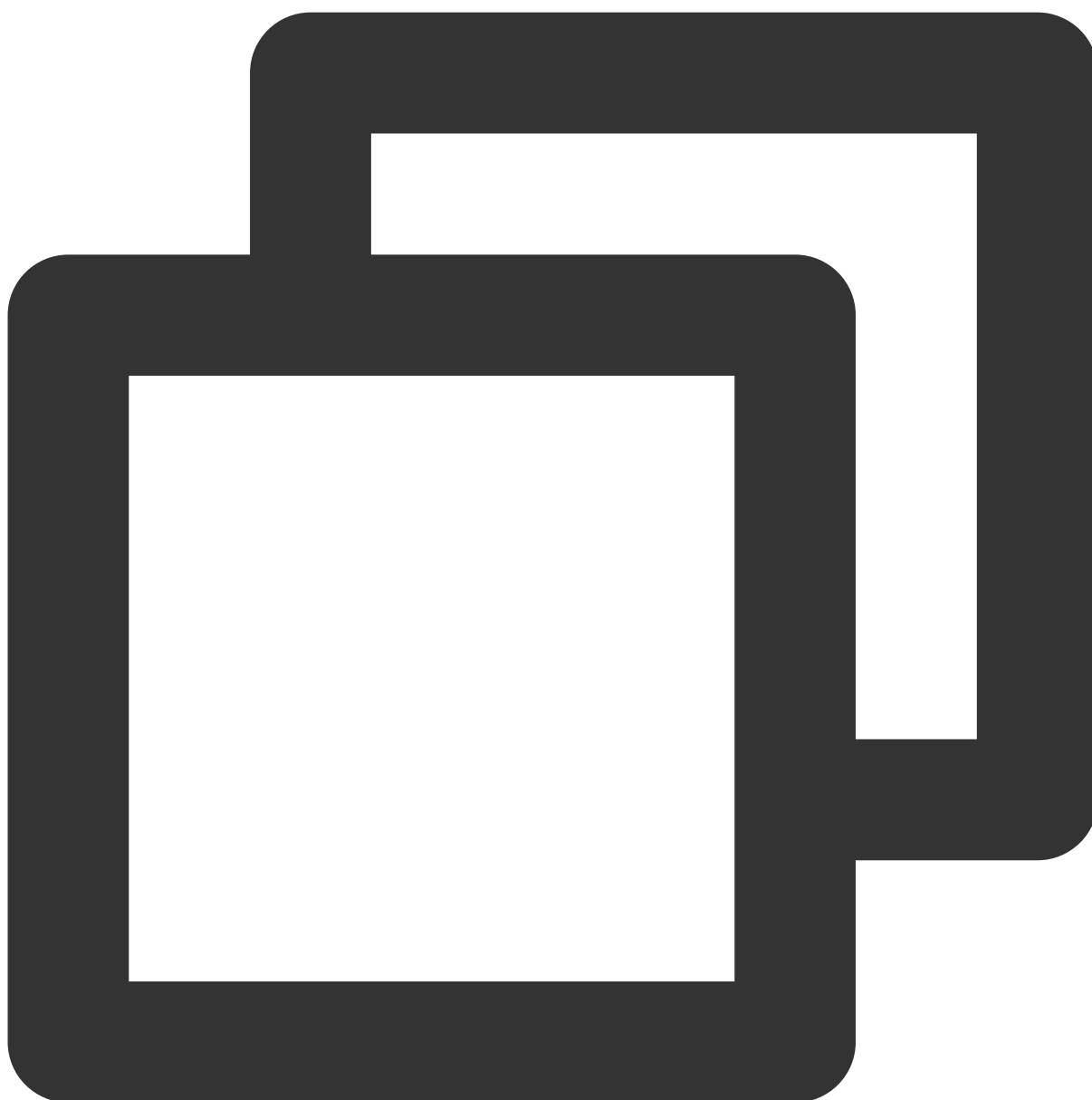
The parameters are shown in the table below:

--	--	--

Parameter	Type	Meaning
requestId	string	Request Id
userId	string	User Id of Cancelled Request

## onReceiveTextMessage

Receive Text Message Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onReceiveTextMessage, ({ roomId, message }) => {
  console.log('roomEngine.onReceiveTextMessage', roomId, message);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room Id
message	<a href="#">TUIMessage</a>	Receive Text Message

## onReceiveCustomMessage

Receive Custom Message Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onReceiveCustomMessage, ({ roomId, message }) => {
  console.log('roomEngine.onReceiveCustomMessage', roomId, message);
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
roomId	string	Room Id

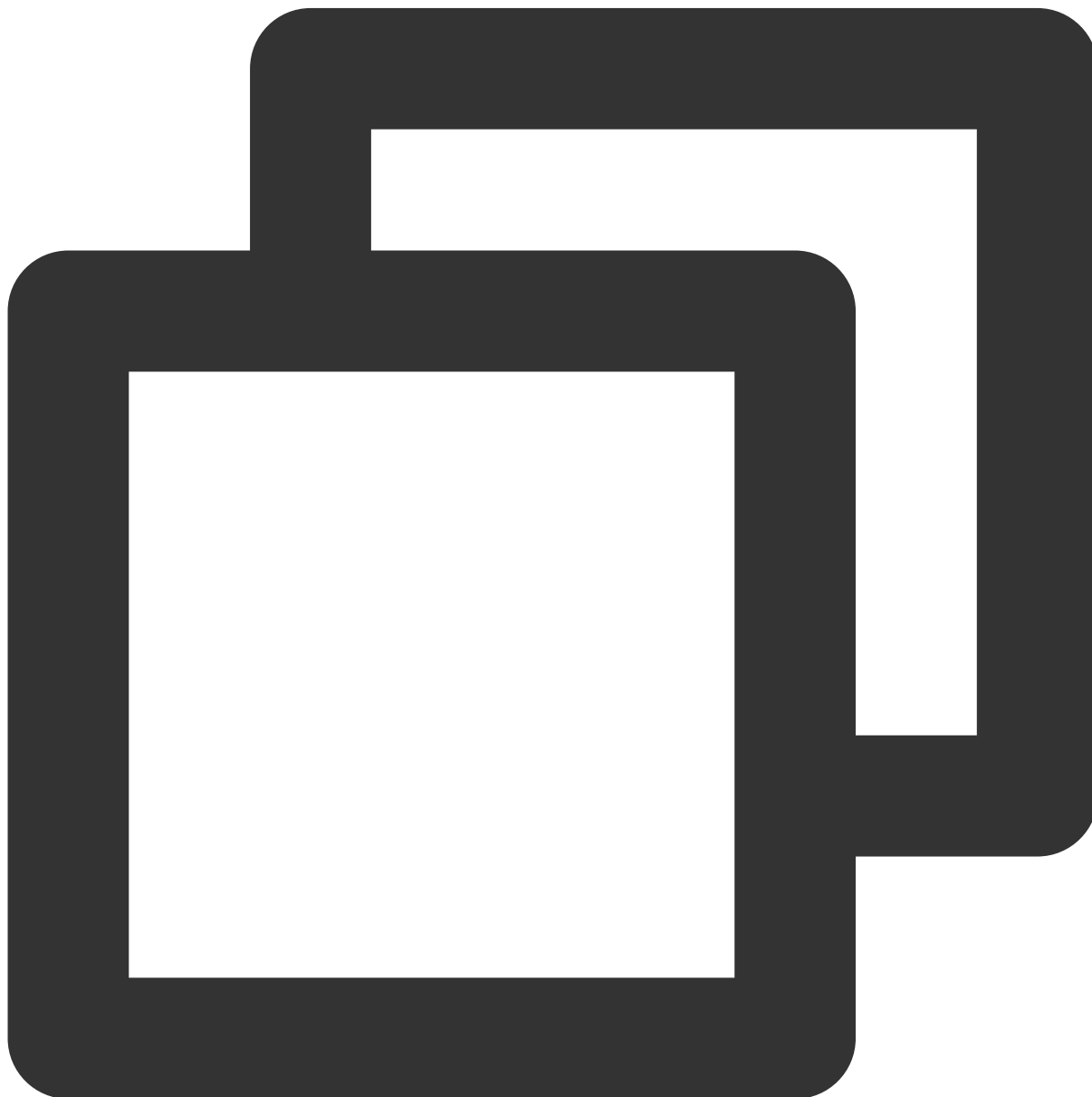
message

TUIMessage

Receive Text Message

## onDeviceChange

Device Change Event



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onDeviceChange, ({ deviceId, type, state }) => {
  console.log('roomEngine.onReceiveCustomMessage', deviceId, type, state);
});
```

```
});
```

The parameters are shown in the table below:

Parameter	Type	Meaning
deviceId	string	Device Id
type	<a href="#">TRTCDeviceType</a>	Device Type
state	<a href="#">TRTCDeviceState</a>	Device Change Status

## onUserScreenCaptureStopped

Screen Sharing Stopped Event, when the user uses the built-in browser stop sharing button to end screen sharing, the user will receive the ' `onUserScreenCaptureStopped` ' event to modify the screen sharing status.



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserScreenCaptureStopped, () => {
  console.log('roomEngine.onReceiveCustomMessage', deviceId, type, state);
});
```

# TUIRoomEngine Defines

Last updated : 2023-11-16 14:33:35

Introduction to Key Type Definition of TUIRoomEngine Electron side.

## Enumeration Value

### TUIRole

User Role, TUIRoomEngine provides three user roles, which are Host, Administrator, and Regular User.

Field	Type	Description
kRoomOwner	number	Host Role
kAdministrator	number	Administrator Role
kGeneralUser	number	Regular User Role

### TUIVideoQuality

Video Resolution

Field	Type	Description
kVideoQuality_360p	number	Low Quality, Resolution is 640 * 360
kVideoQuality_540p	number	SD, Resolution is 960 * 540
kVideoQuality_720p	number	HD, Resolution is 1280 * 720
kVideoQuality_1080p	number	Ultra HD, Resolution is 1920 * 1080

### TUIAudioProfile

Audio Resolution

Field	Type	Description
kAudioProfileSpeech	number	Voice Mode
kAudioProfileDefault	number	Standard Mode (Default Mode)
kAudioProfileMusic	number	Music Mode



## TUIVideoStreamType

### Streams Type

Field	Type	Description
kCameraStream	number	Camera Streams
kScreenStream	number	Screen Sharing Streams
kCameraStreamLow	number	Low Resolution Camera Streams

## TUINetworkQuality

### Network Status

Field	Type	Description
kQualityUnknown	number	Network Condition Unknown
kQualityExcellent	number	Network Condition Excellent
kQualityGood	number	Network Condition Good
kQualityPoor	number	Network Condition Fair
kQualityBad	number	Network Condition Poor
kQualityVeryBad	number	Network Condition Very Poor
kQualityDown	number	Network Connection Disconnected

## TUIRoomType

### Room Type

Field	Type	Description
kGroup	number	Group Type Room, suitable for conference and educational scene, the microphone position in this room is unordered and has no quantity limit
kOpen	number	Open Type Room, suitable for live streaming scene, the microphone position in this room is ordered and has a quantity limit

## TUISpeechMode

### Speech Type

Field	Type	Description
kFreeToSpeak	number	Free Speech Mode
kApplyToSpeak	number	Hand-raising Speech Mode
kSpeakAfterTakingSeat	number	Speak After Sitting (Grab Microphone Position)

## TUICaptureSourceType

Screen Sharing Type

Field	Type	Description
kWindow	number	Sharing Target is a specific Windows or Mac window todo (only for electron)
kScreen	number	Sharing Target is the entire Windows desktop or Mac desktop

## TUIChangeReason

Change Reason (Audio and Video Status Change Operation Reason: Self-initiated modification or modified by room owner/administrator)

Field	Type	Description
kChangedBySelf	number	Self-operation
kChangedByAdmin	number	Room Owner or Administrator Operation

## TUIMediaDevice

Field	Type	Description
kMicrophone	number	Mic
kCamera	number	Camera
kScreen	number	Screen Sharing

## TUIRequestAction

Room Type

Field	Type	Description

kInvalidAction	number	Invalid Operation
kRequestToOpenRemoteCamera	number	Request Remote Camera On
kRequestToOpenRemoteMicrophone	number	Request Remote Mic On
kRequestToConnectOtherRoom	number	Request Remote Cross-room Streaming, web side temporarily unsupported
kRequestToTakeSeat	number	Request Go Live
kRequestRemoteUserOnSeat	number	Request Remote Go Live

## TUIRequestCallbackType

Request Type

Field	Type	Description
kRequestAccepted	number	Peer Accepted
kRequestRejected	number	Peer Rejected
kRequestCancelled	number	Request Canceled
kRequestTimeout	number	Request Timeout
kRequestError	number	Request Error

## Type Definition

### TUILoginUserInfo

Current Logged-in User Information

Field	Type	Description
userId	string	Login User's ID
userName	string	Login User's Name
avatarUrl	string	Login User's Avatar

### TUIRoomInfo

Room data, user can use `roomEngine.getRoomInfo` to get room data.

--	--	--

Field	Type	Description
roomId	string	Room Number, String Type Room Number
roomType	<a href="#">TUIRoomType</a>	Room Type
owner	string	Host's userId
name	string	Room ID
createTime	string	Creation time
roomMemberCount	number	Current total number of people in the room
maxSeatCount	number	Maximum number of microphone positions in the room
enableVideo	boolean	Allow users to join and turn on Audio
enableAudio	boolean	Allow users to join and turn on Video
enableMessage	boolean	Allow users to join and send messages
enableSeatControl	boolean	Enable microphone position control

## TUIUserInfo

### User Information

Field	Type	Description
userId	string	User Id
userName	string	User Name
avatarUrl	string	User Avatar
userRole	<a href="#">TUIRole</a>	User Role
hasAudioStream	boolean	Whether there are Audio streams
hasVideoStream	boolean	Whether there are Video streams
hasScreenStream	boolean	Whether there is Screen Sharing stream

## TUIMessage

### Message Information

Field	Type	Description

messageId	string	Message Id
message	string	Message
timestamp	number	Timestamp information, accurate to seconds
userId	string	User Id
userName	string	User name
avatarUrl	string	User Avatar

## TUIRequest

### Request Information

Field	Type	Description
requestAction	<a href="#">TUIRequestAction</a>	Request Type
timestamp	number	Request Initiation Time
requestId	string	Request Id v1.0.2 and above versions requestId type is string; v1.0.0 and v1.0.1 versions requestId type is number;
userId	string	Initiate Request's User Id
content	string	Other Content

## TUIRequestCallback

### Request Callback Information

Field	Type	Description
requestCallbackType	<a href="#">TUIRequestCallbackType</a>	Request Callback Type, Accept/Reject/Cancel/Timeout/Error
requestId	string	Request Id v1.0.2 and above versions requestId type is string; v1.0.0 and v1.0.1 versions requestId type is number;
userId	string	User Id
code	number	Request Response Code

message	string	Request Status Supplemental Description
---------	--------	---

## TUISeatInfo

### Microphone Position Information

Field	Type	Description
index	number	Microphone Position Number
userId	string	Microphone Position Correspondence's User Id
locked	boolean	Whether the current microphone position is locked
videoMuted	boolean	Whether the current microphone position prohibits Video
audioMuted	boolean	Whether the current microphone position prohibits Audio

## TUISeatLockParams

### Microphone Lock Status

Field	Type	Description
lockSeat	boolean	Lock Microphone Position
lockVideo	boolean	Lock Microphone Position Video
lockAudio	boolean	Lock Microphone Position Audio

## TUINetwork

### Network Information

Field	Type	Description
userId	string	User ID
quality	TUINetworkQuality	Network Quality
upLoss	number	Upstream Packet Loss Rate, Unit (%) The smaller the value, the better, currently only local users have this information
downLoss	number	Downstream Packet Loss Rate, Unit (%) The smaller the value, the better, currently only local users have

		this information
delay	number	Network Latency, Unit ms, currently only local users have this information

# Flutter

## API Overview

Last updated : 2023-11-21 15:20:12

### TUIRoomEngine API List

TUIRoomEngine API is the Audio/Video call Component's No UI Interface, you can use this set of API to customize packaging according to your business needs.

TUIRoomEngine

#### TUIRoomEngine Core Methods

API	Description
<a href="#">createInstance</a>	Create TUIRoomEngine Instance
<a href="#">destroyInstance</a>	Destroy TUIRoomEngine Instance
<a href="#">login</a>	Login interface, you need to initialize user information before entering the room and perform a series of operations.
<a href="#">logout</a>	Logout interface, there will be actively leave room operation, destroy resources
<a href="#">setSelfInfo</a>	Set local user name and avatar
<a href="#">setLoginUserInfo</a>	Set login user information
<a href="#">getSelfInfo</a>	Get local user basic information
<a href="#">addObserver</a>	Set event callback
<a href="#">removeObserver</a>	Remove event callback

#### Room Related Active Interface

API	Description
<a href="#">createRoom</a>	Create room
<a href="#">destroyRoom</a>	Close the room



<a href="#">enterRoom</a>	Entered room
<a href="#">exitRoom</a>	Leave room
<a href="#">connectOtherRoom</a>	Connect to other room
<a href="#">disconnectOtherRoom</a>	Disconnect from other room
<a href="#">fetchRoomInfo</a>	Get room data
<a href="#">updateRoomNameByAdmin</a>	Update room name
<a href="#">updateRoomSpeechModeByAdmin</a>	Set room management mode (only administrator or group owner can call)

## Local User View Rendering, Video Management

API	Description
<a href="#">setLocalVideoView</a>	Set the view control for local user video rendering
<a href="#">openLocalCamera</a>	Open local camera
<a href="#">closeLocalCamera</a>	Close local camera
<a href="#">updateVideoQuality</a>	Update local video codec quality settings
<a href="#">updateVideoQualityEx</a>	Set the encoding parameters of video encoder
<a href="#">setVideoResolutionMode</a>	Set the resolution mode of video encoder
<a href="#">enableGravitySensor</a>	Enable the gravity sensor
<a href="#">startPushLocalVideo</a>	Start pushing local video
<a href="#">stopPushLocalVideo</a>	Stop pushing local video
<a href="#">startScreenSharing</a>	Start screen sharing
<a href="#">stopScreenSharing</a>	Stop screen sharing

## Local User Audio Management

API	Description
<a href="#">openLocalMicrophone</a>	Open local microphone
<a href="#">closeLocalMicrophone</a>	Close local microphone

<a href="#">updateAudioQuality</a>	Update local audio codec quality settings
<a href="#">muteLocalAudio</a>	Mute local audio
<a href="#">unMuteLocalAudio</a>	UnMute local audio

## Remote User View Rendering, Video Management

API	Description
<a href="#">setRemoteVideoView</a>	Set the view control for remote user video rendering
<a href="#">startPlayRemoteVideo</a>	Start playing remote user video
<a href="#">stopPlayRemoteVideo</a>	Stop playing remote user video
<a href="#">muteRemoteAudioStream</a>	Mute remote user

## Room User Information

API	Description
<a href="#">getUserList</a>	Get the member list in the room
<a href="#">getUserInfo</a>	Get member information

## Room User Management

API	Description
<a href="#">changeUserRole</a>	Modify user role (only administrator or group owner can call)
<a href="#">kickRemoteUserOutOfRoom</a>	Kick Remote User out of the Room (Only Administrator or Group Owner can call)
<a href="#">addCategoryTagForUsers</a>	Add category tags to users (Only Administrator or Group Owner can call)
<a href="#">removeCategoryTagForUsers</a>	Remove category tags to users (Only Administrator or Group Owner can call)
<a href="#">getUserListByTag</a>	Get user information in the room based on tags

## Speech Management in Room

--	--

API	Description
<a href="#">disableDeviceForAllUserByAdmin</a>	Control the permission status of whether all users in the current room can open Audio and Video streams capturing devices, such as: Prohibit all from turning on the mic, Prohibit all from turning on the Camera, Prohibit all from turning on Screen Sharing (currently only available in meeting scenes, and only administrators or group owners can invoke).
<a href="#">openRemoteDeviceByAdmin</a>	Request Remote User to Open Media Device (Only Administrator or Group Owner can call)
<a href="#">closeRemoteDeviceByAdmin</a>	Close Remote User's Media Device (Only Administrator or Group Owner can call)
<a href="#">applyToAdminToOpenLocalDevice</a>	Request to Open Local Media Device (Available for Ordinary Users)

## Microphone Seat Management in Room

API	Description
<a href="#">setMaxSeatCount</a>	Set Maximum Number of Microphone Seats (Only supported when entering the room and creating the room)
<a href="#">getSeatList</a>	Get Microphone Seat List
<a href="#">lockSeatByAdmin</a>	Lock Microphone Seat (Including Position Lock, Audio State Lock, Video State Lock)
<a href="#">takeSeat</a>	Apply to Go Live (Not Required in Free Speech Mode)
<a href="#">leaveSeat</a>	Apply to leave the live (Not Required in Free Speech Mode)
<a href="#">takeUserOnSeatByAdmin</a>	Host/Administrator invites user to go live
<a href="#">kickUserOffSeatByAdmin</a>	Host/Administrator kicks user off the microphone seat

## Signaling Management

API	Description
<a href="#">cancelRequest</a>	Cancel Request
<a href="#">responseRemoteRequest</a>	Reply to Request

## Send Message

API	Description
<a href="#">sendMessage</a>	Send Text Message
<a href="#">sendCustomMessage</a>	Send Custom Message
<a href="#">disableSendingMessageByAdmin</a>	Disable Remote User's Text Message Sending Ability (Only Administrator or Group Owner can call)
<a href="#">disableSendingMessageForAllUser</a>	Disable All Users' Text Message Sending Ability (Only Administrator or Group Owner can call) Advanced Feature: Get TRTC Instance

## Advanced Features

API	Description
<a href="#">switchCamera</a>	Switch front/rear camera
<a href="#">setBeautyLevel</a>	Set beauty level
<a href="#">setWhitenessLevel</a>	Set whiteness level

## Debugging related

API	Description
<a href="#">callExperimentalAPI</a>	Call experimental api

# TUIRoomObserver Callback Event

TUIRoomObserver is the Callback Event class corresponding to TUIRoomEngine. You can monitor the Callback Events you need through this Callback.

TUIRoomObserver

# TUIRoomObserver

## Error Callback

--	--

API	Description
<a href="#">onError</a>	Error Callback Event

### Login Status Event Callback

API	Description
<a href="#">onKickedOffLine</a>	User Kicked Offline Event
<a href="#">onUserSigExpired</a>	User Credential Timeout Event

### Room Event Callback

API	Description
<a href="#">onRoomNameChanged</a>	Room Name Change Event
<a href="#">onAllUserMicrophoneDisableChanged</a>	All Users' Microphones Disabled in Room Event
<a href="#">onAllUserCameraDisableChanged</a>	All Users' Cameras Disabled in Room Event
<a href="#">onSendMessageForAllUserDisableChanged</a>	All Users' Text Message Sending Disabled in Room Event
<a href="#">onKickedOutOfRoom</a>	Room Dismissed Event
<a href="#">onRoomDismissed</a>	Kicked Out of Room Event
<a href="#">onRoomSpeechModeChanged</a>	Room Microphone Control Mode Change

### Room User Event Callback

API	Description
<a href="#">onRemoteUserEnterRoom</a>	Remote User Entering Room Event
<a href="#">onRemoteUserLeaveRoom</a>	Remote User Leaving Room Event
<a href="#">onUserRoleChanged</a>	User Role Change Event
<a href="#">onUserVideoStateChanged</a>	User Video State Change Event
<a href="#">onUserAudioStateChanged</a>	User Audio State Change Event
<a href="#">onUserVoiceVolumeChanged</a>	User Volume Change Event
<a href="#">onSendMessageForUserDisableChanged</a>	User Text Message Sending Ability Change Event

<a href="#">onUserNetworkQualityChanged</a>	User Network Status Change Event
<a href="#">onUserScreenCaptureStopped</a>	Screen Sharing End Event

### Room Microphone Seat Event Callback

API	Description
<a href="#">onRoomMaxSeatCountChanged</a>	Room Maximum Microphone Seat Number Change Event (Only effective in conference type rooms)
<a href="#">onSeatListChanged</a>	Microphone Seat List Change Event
<a href="#">onKickedOffSeat</a>	Received User Kicked Off Microphone Event

### Request Signaling Event Callback

API	Description
<a href="#">onRequestReceived</a>	Received Request Message Event
<a href="#">onRequestCancelled</a>	Received Request Cancellation Event

### Room Message Event Callback

API	Description
<a href="#">onReceiveTextMessage</a>	Received Normal Text Message Event
<a href="#">onReceiveCustomMessage</a>	Received Custom Message Event

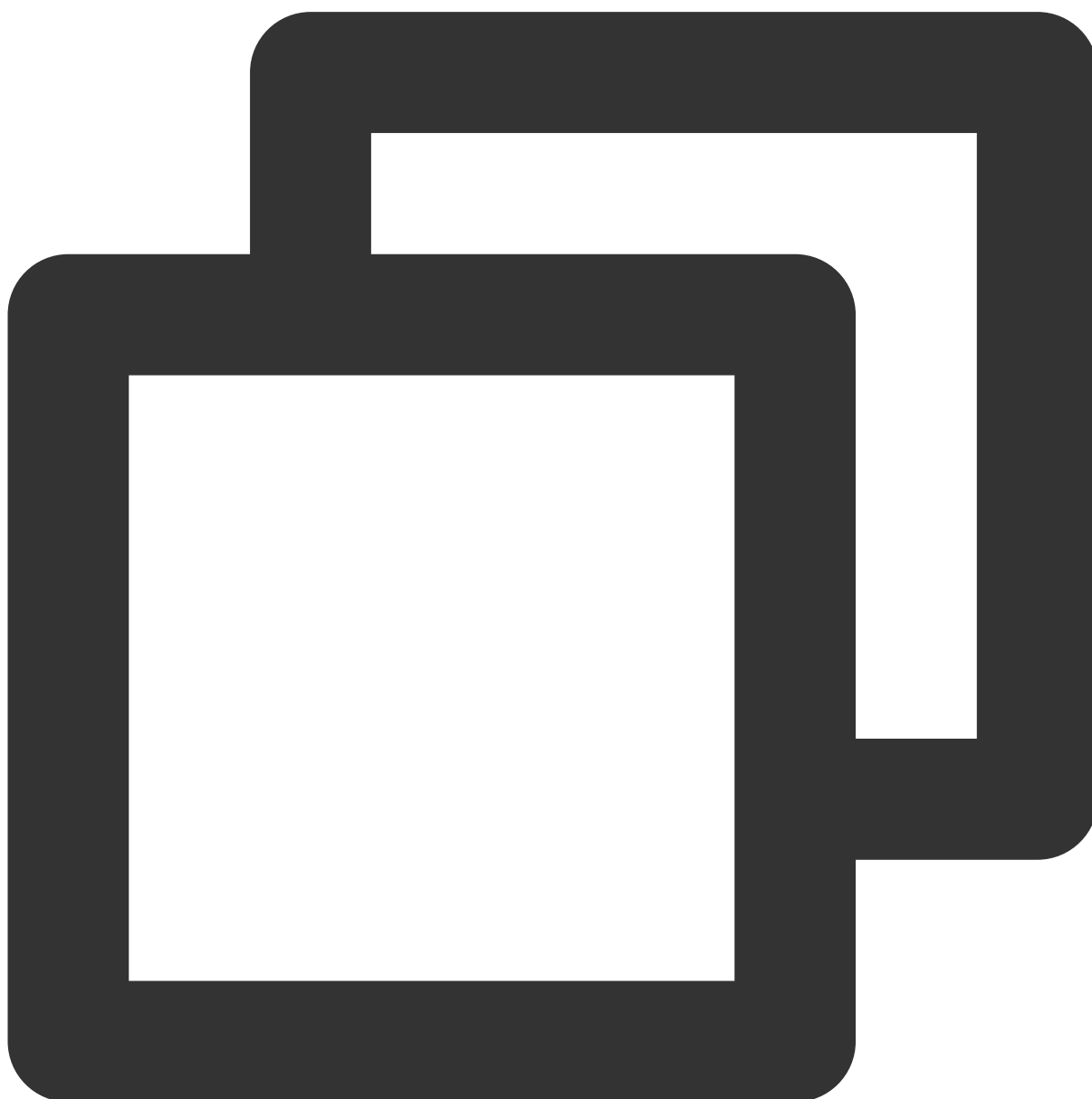
# TUIRoomObserver

Last updated : 2023-11-14 16:34:53

## TUIRoomEngine Event Callback

### **onError**

Error Event.

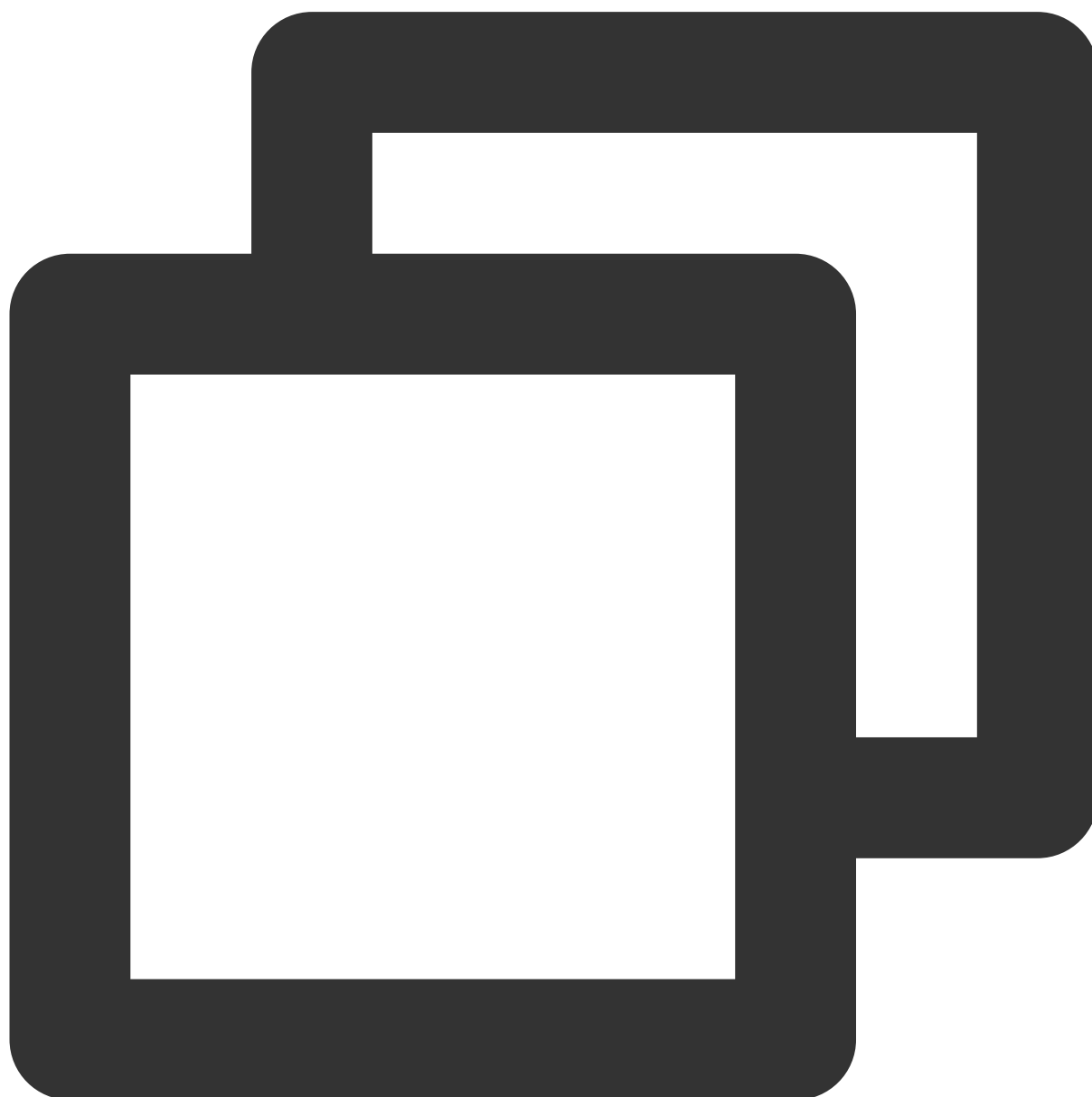


```
OnError onError = (TUIError errorCode, String message) {}
```

Parameter	Type	Description
errorCode	<a href="#">TUIError</a>	Error Code
message	String	Error Message

## onKickedOffLine

Other terminals login and get kicked off event.



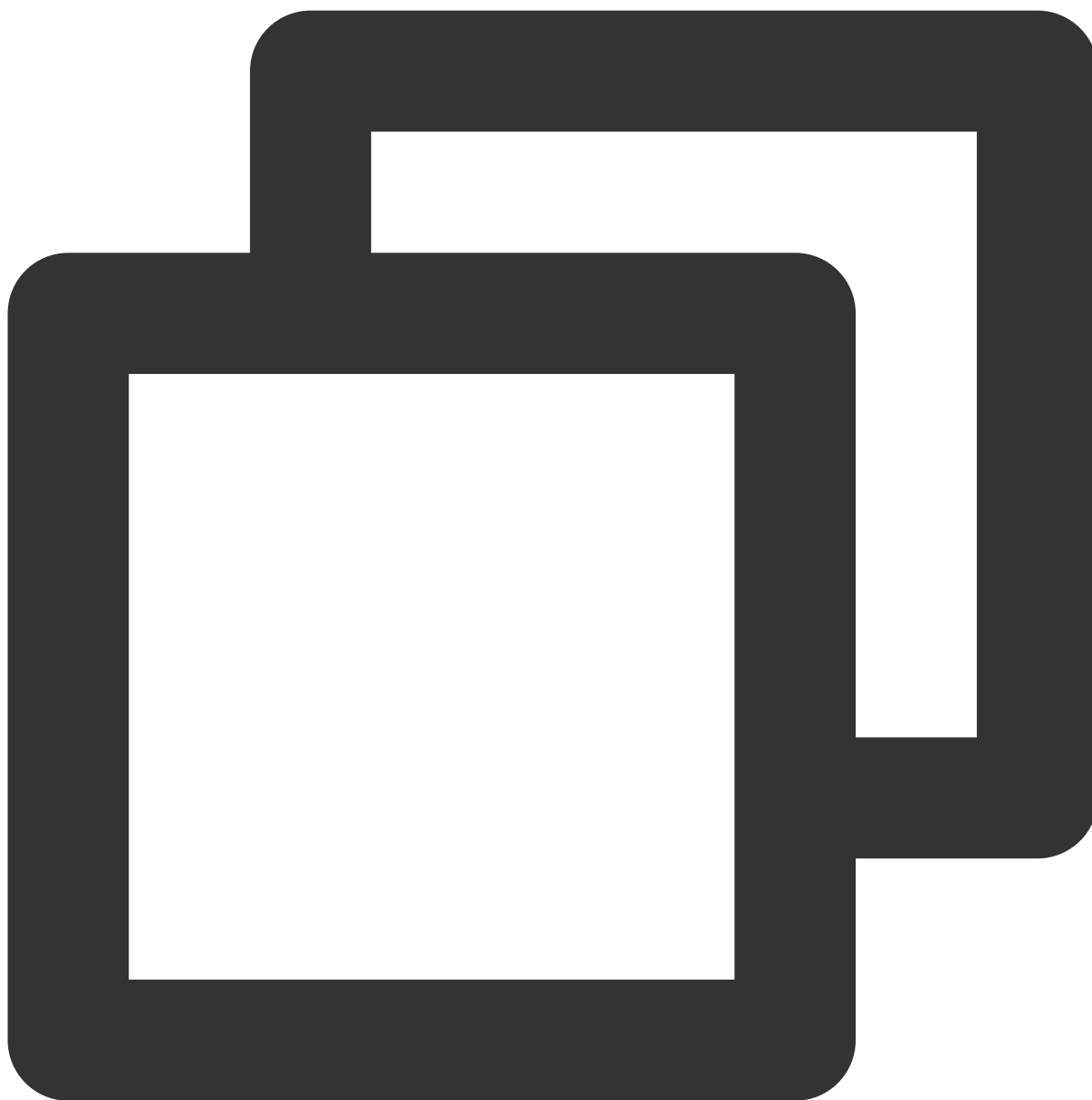


```
OnKickedOffLine onKickedOffLine = (String message) {}
```

Parameter	Type	Description
message	String	Kicked out description

## onUserSigExpired

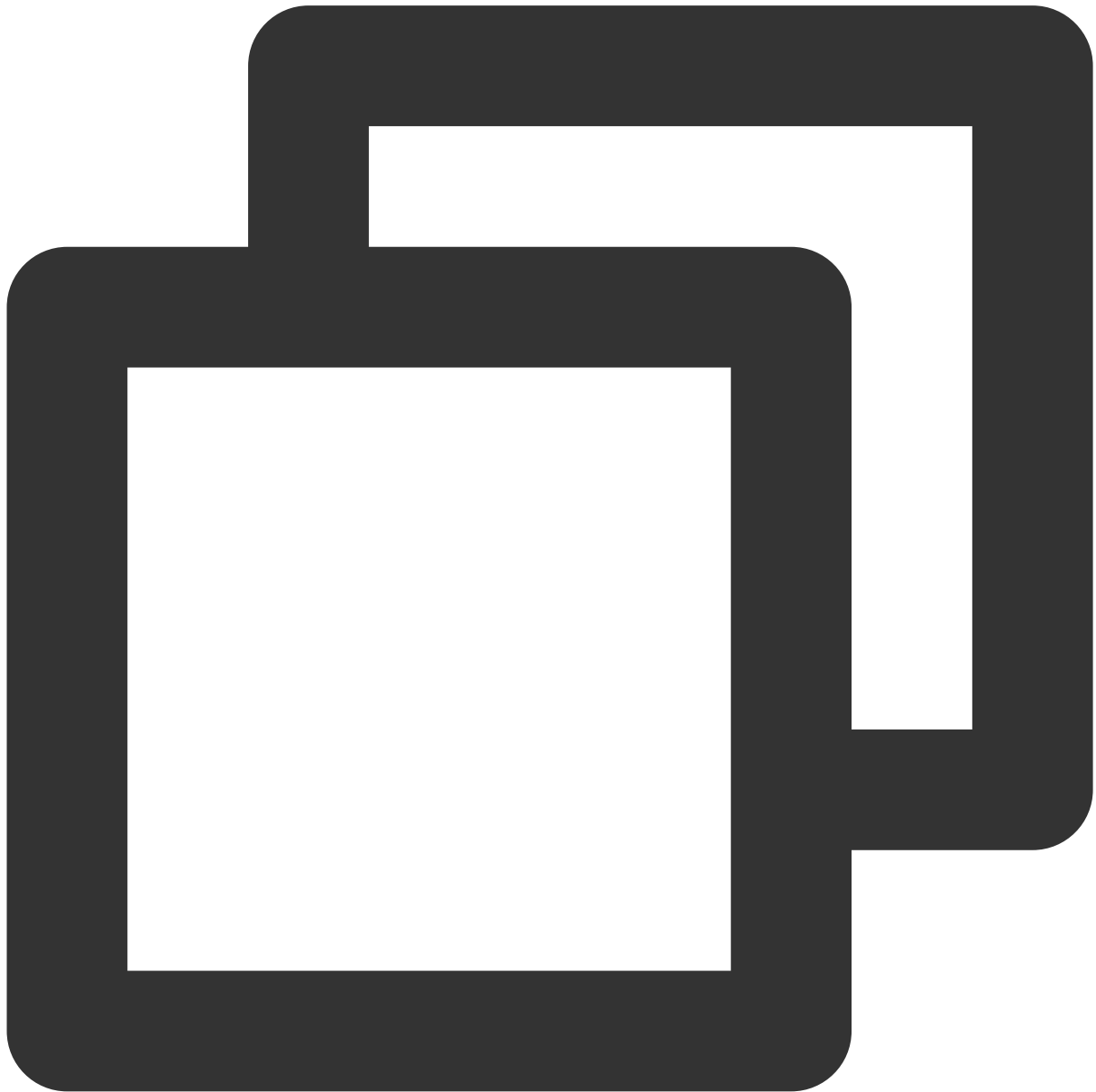
User credential timeout event.



```
OnUserSigExpired onUserSigExpired = () {}
```

## onRoomNameChanged

Room name change event.



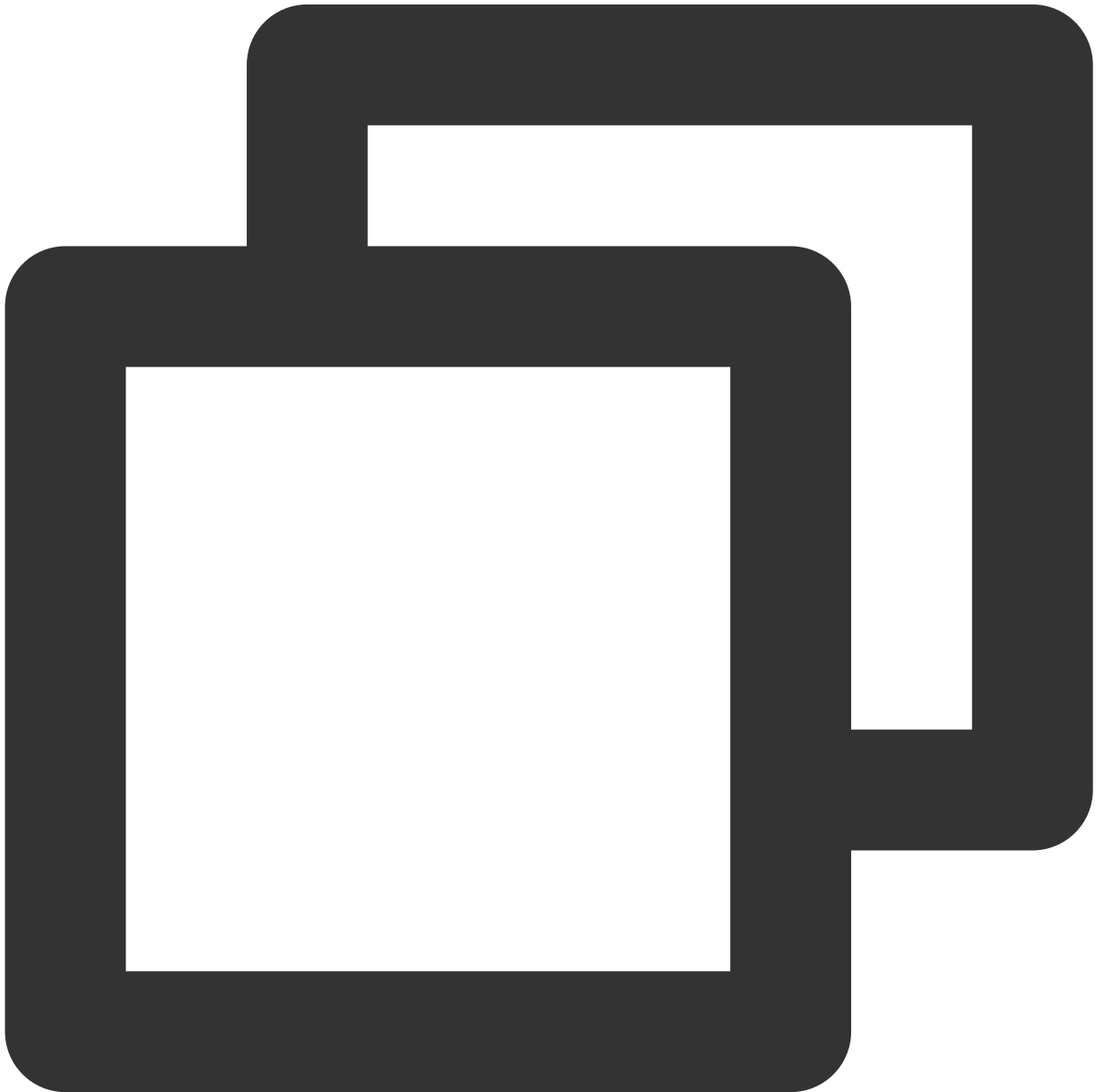
```
OnRoomNameChanged onRoomNameChanged = (String roomId, String roomName) {}
```

Parameter	Type	Description

roomId	String	Room ID
roomName	String	Room Name

**onAllUserMicrophoneDisableChanged**

Inside the room, all users' mic is disabled event.



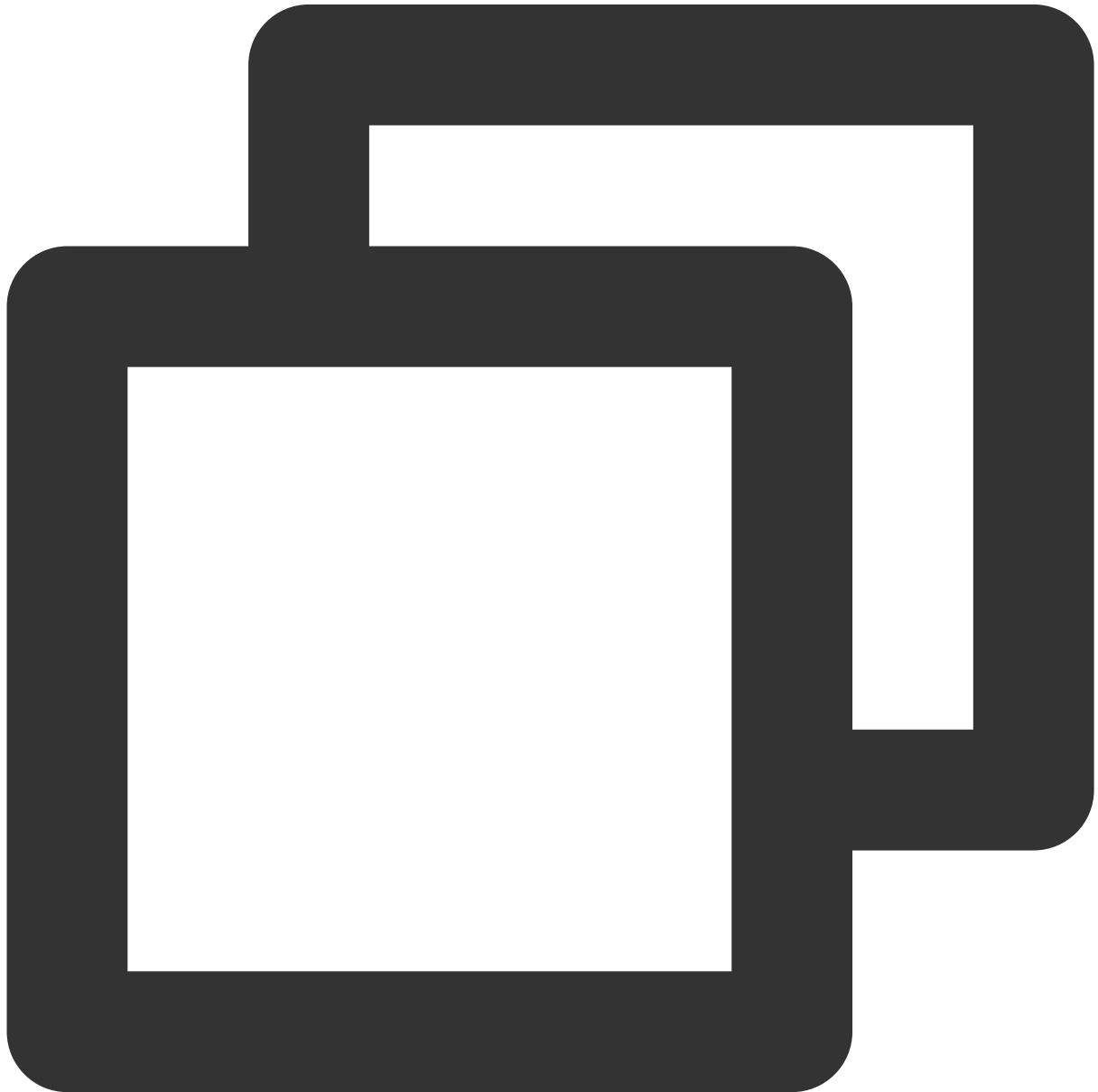
OnAllUserMicrophoneDisableChanged onAllUserMicrophoneDisableChanged = (String roomId)

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

roomId	String	Room ID
isDisable	bool	Whether it is disabled

### onAllUserCameraDisableChanged

All users' Camera in the Room are disabled event.



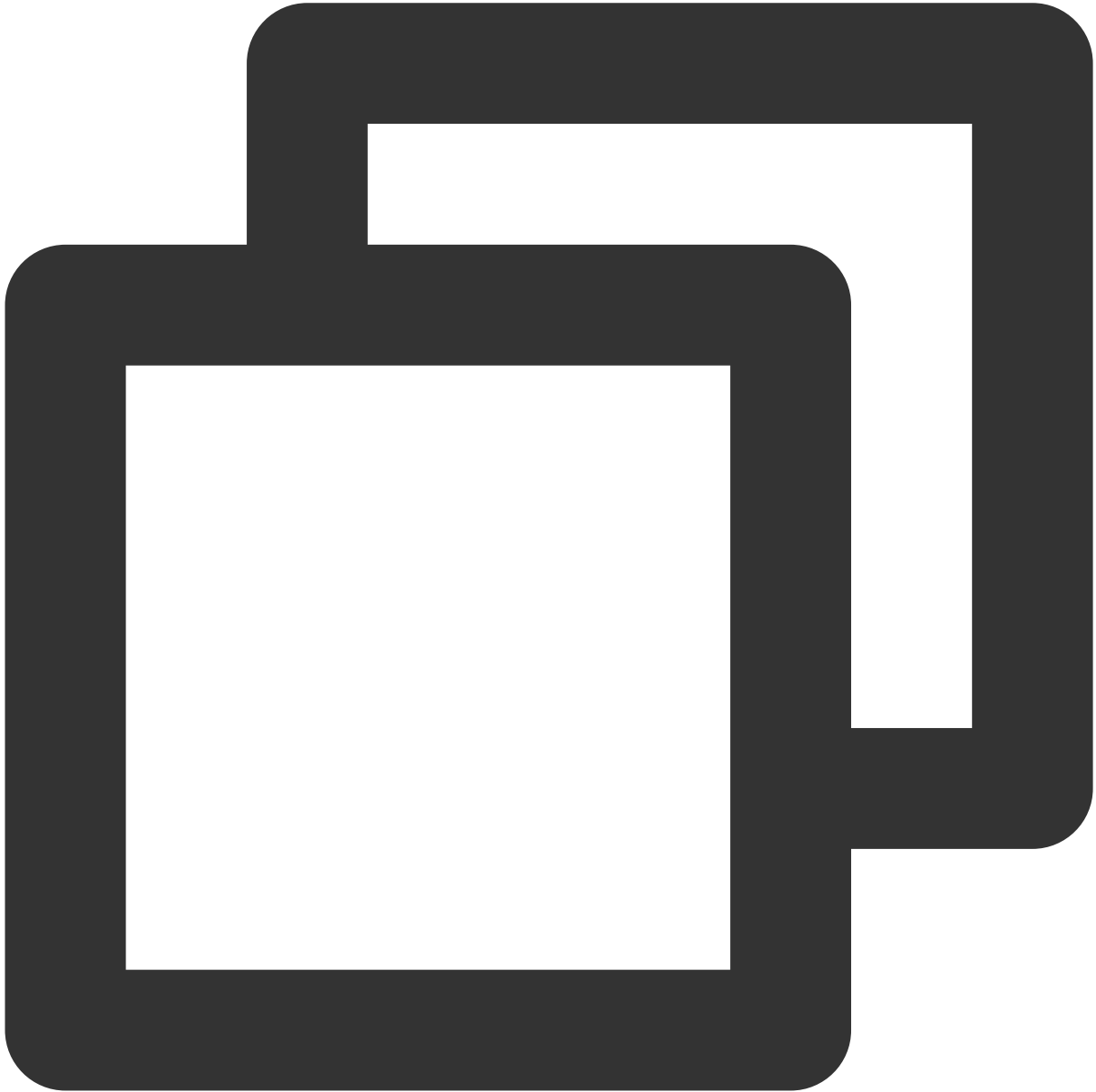
```
OnAllUserCameraDisableChanged onAllUserCameraDisableChanged = (String roomId, bool
```

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

roomId	String	Room ID
isDisable	bool	Whether it is disabled

onSendMessageForAllUserDisableChanged

Inside the room, all users' text message sending is disabled event.

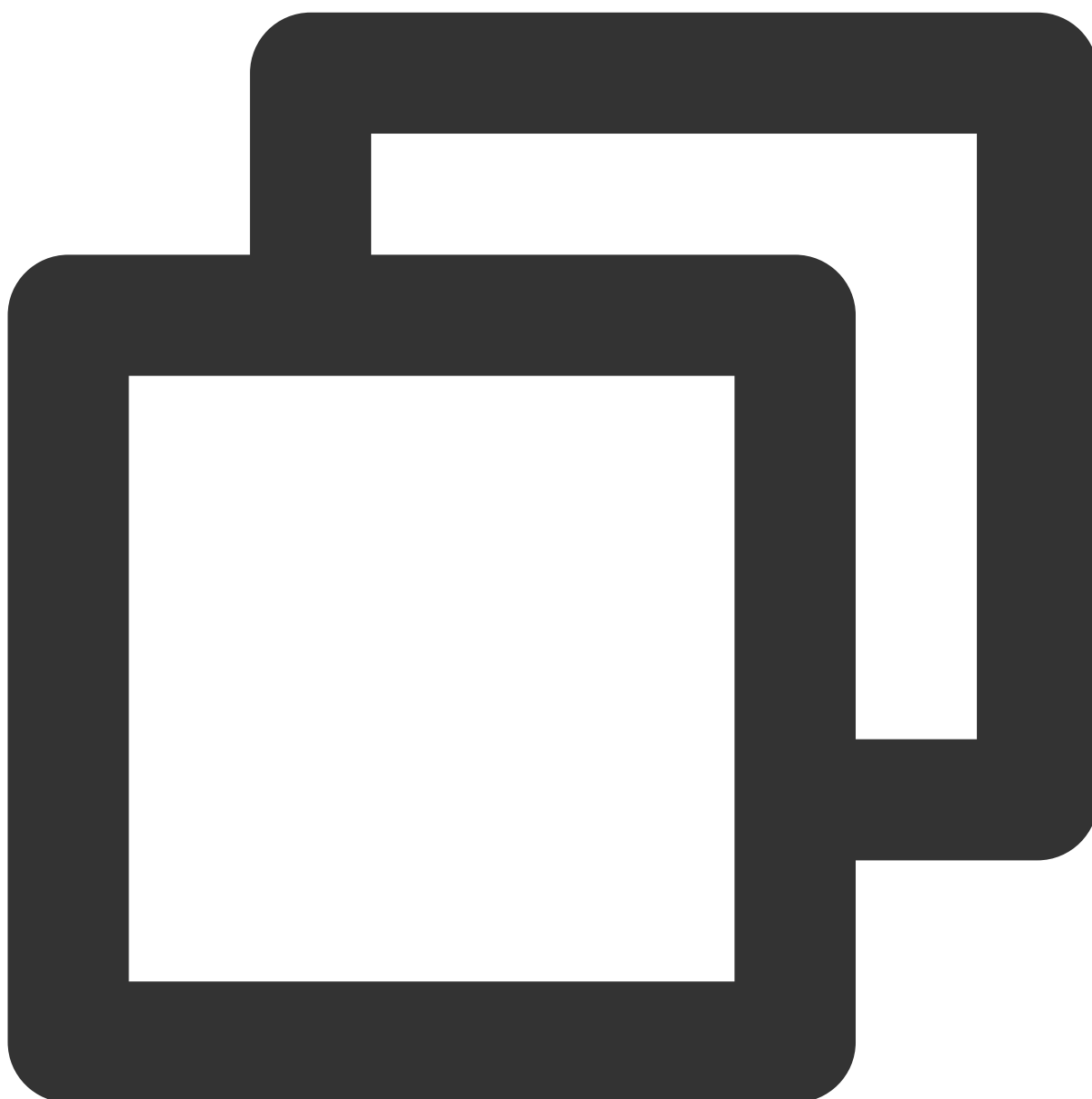


OnSendMessageForAllUserDisableChanged onSendMessageForAllUserDisableChanged = (Stri		

Parameter	Type	Description
roomId	String	Room ID
isDisable	bool	Whether it is disabled

## onRoomDismissed

Room dissolution event.

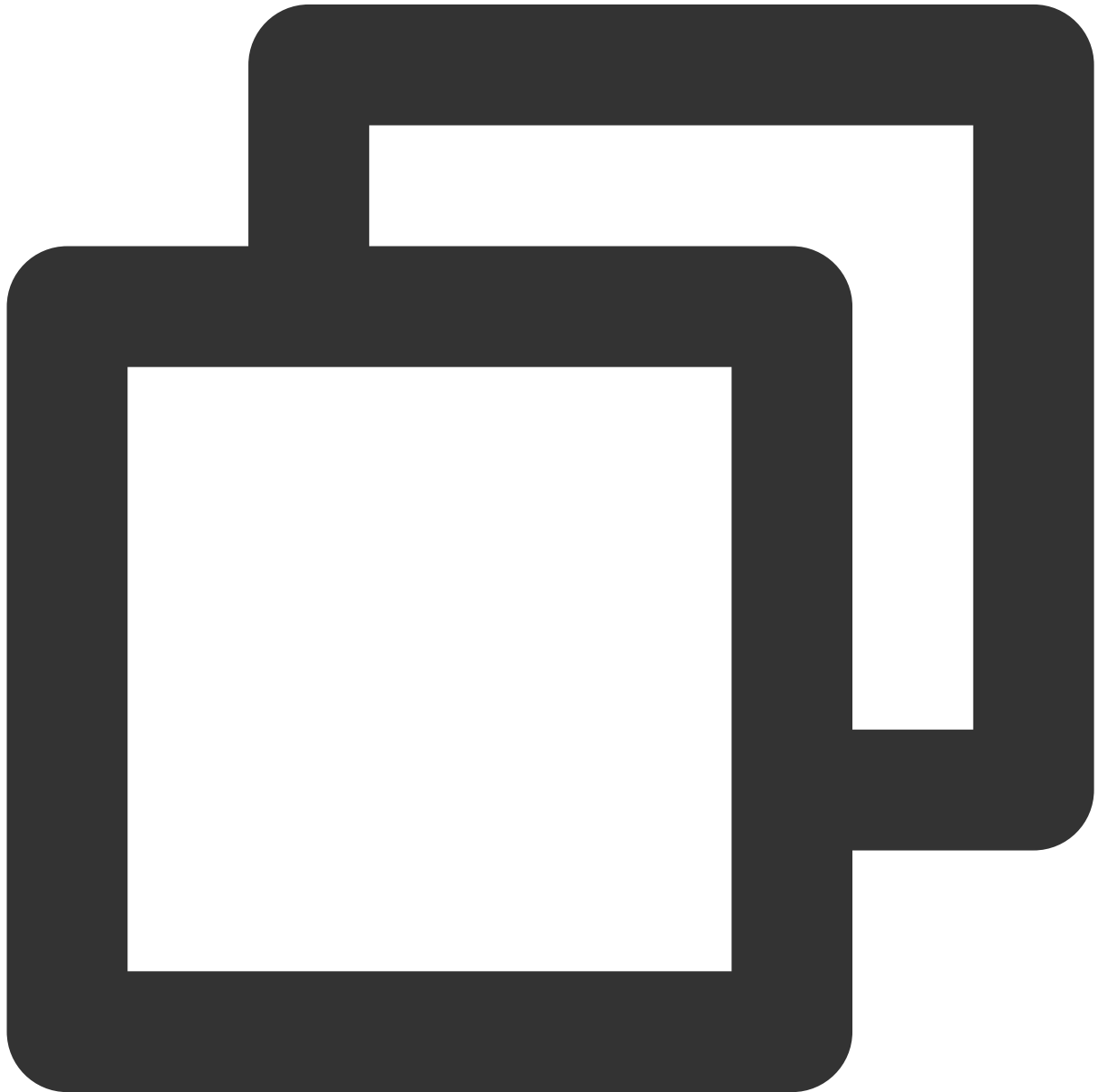


```
OnRoomDismissed onRoomDismissed = (String roomId) {}
```

Parameter	Type	Description
roomId	String	Room ID

## onKickedOutOfRoom

Kick out of the room event



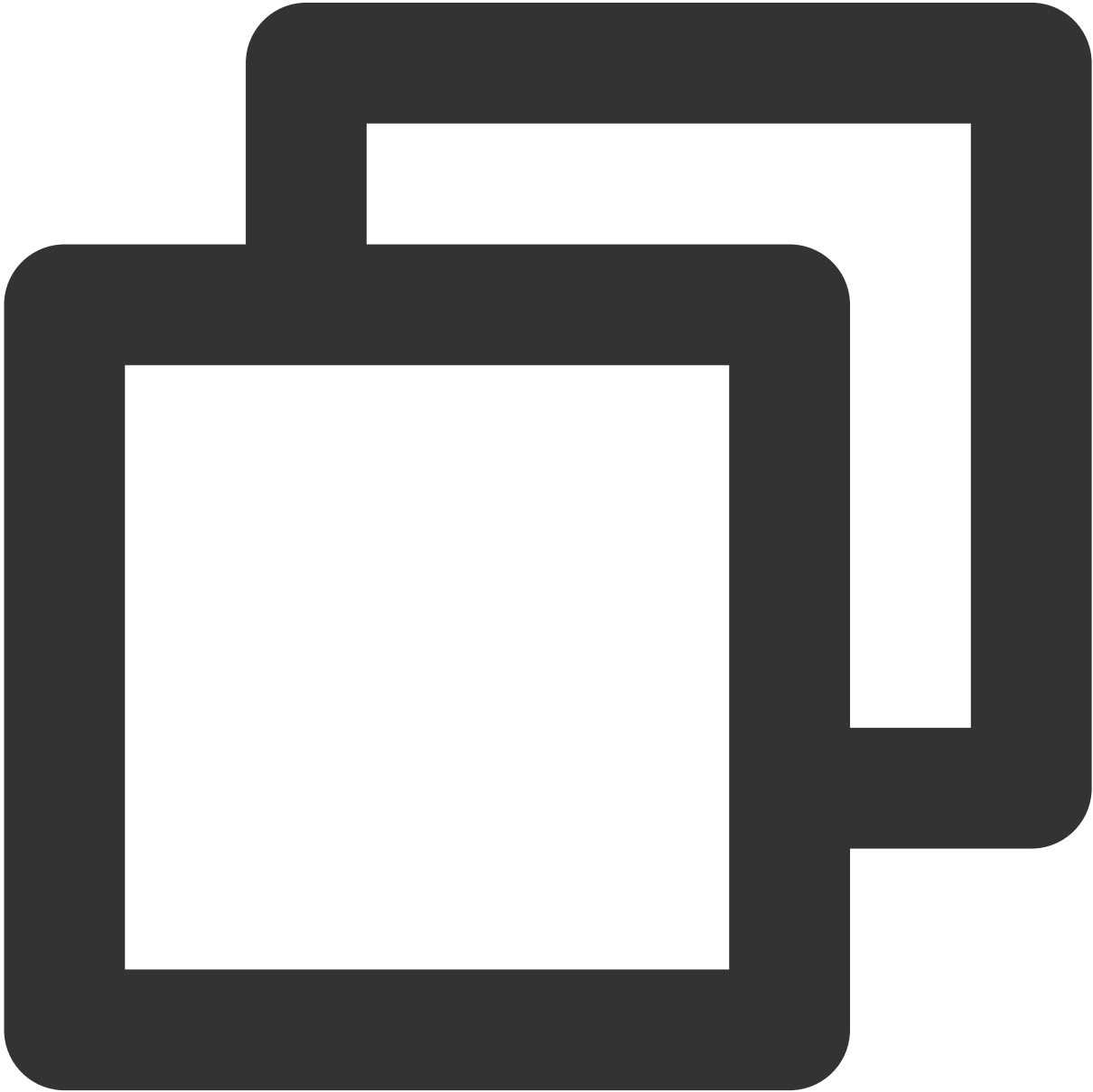
```
OnKickedOutOfRoom onKickedOutOfRoom = (String roomId, String message) {}
```

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

roomId	String	Room ID
message	String	Description of being kicked out

**onRoomSpeechModeChanged**

Mic control mode changes in the room.



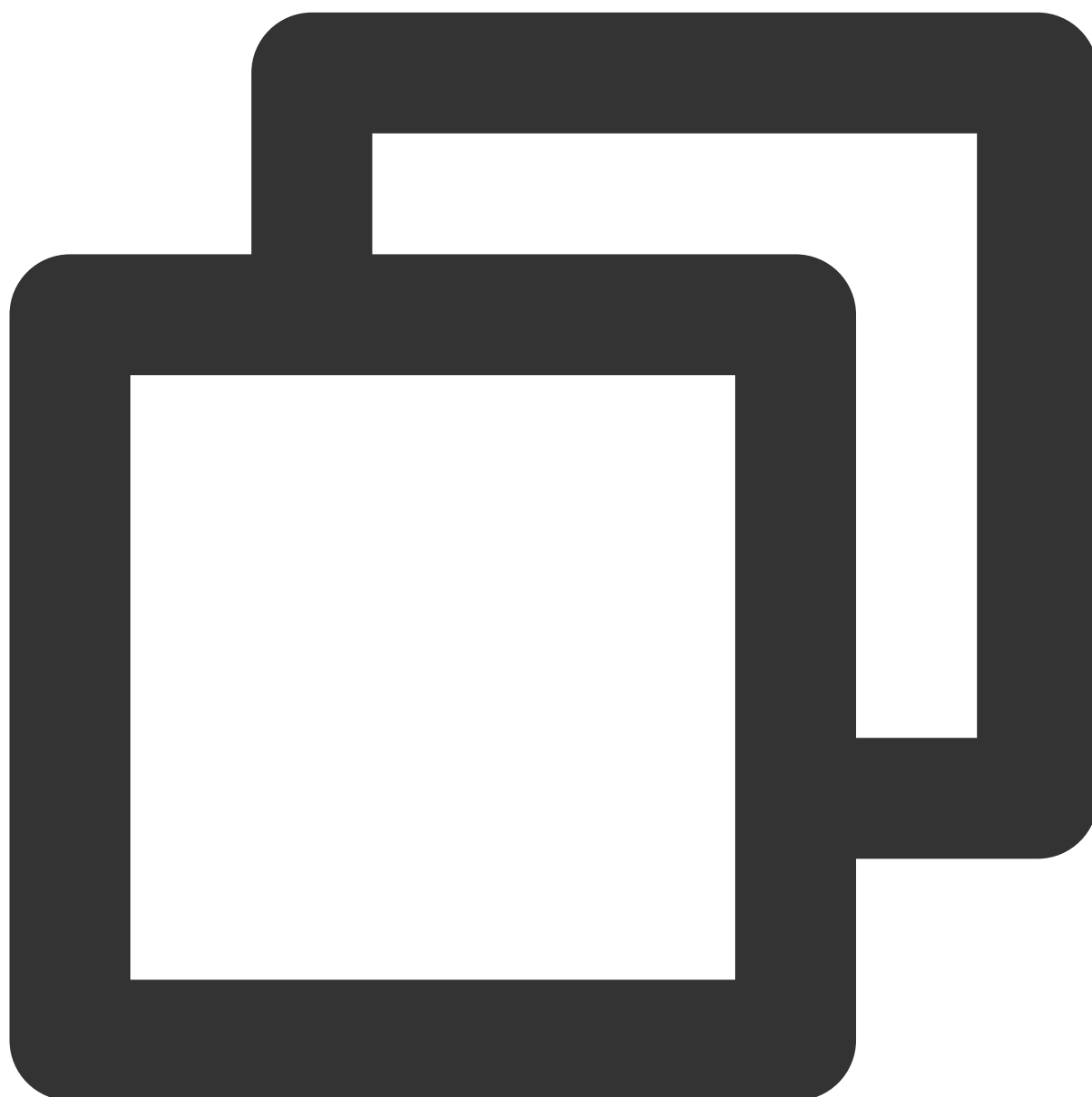
OnRoomSpeechModeChanged onRoomSpeechModeChanged = (String roomId, TUISpeechMode spe		



Parameter	Type	Description
roomId	String	Room ID
speechMode	<a href="#">TUISpeechMode</a>	Mic control mode

## onRemoteUserEnterRoom

Remote user enters the room event.

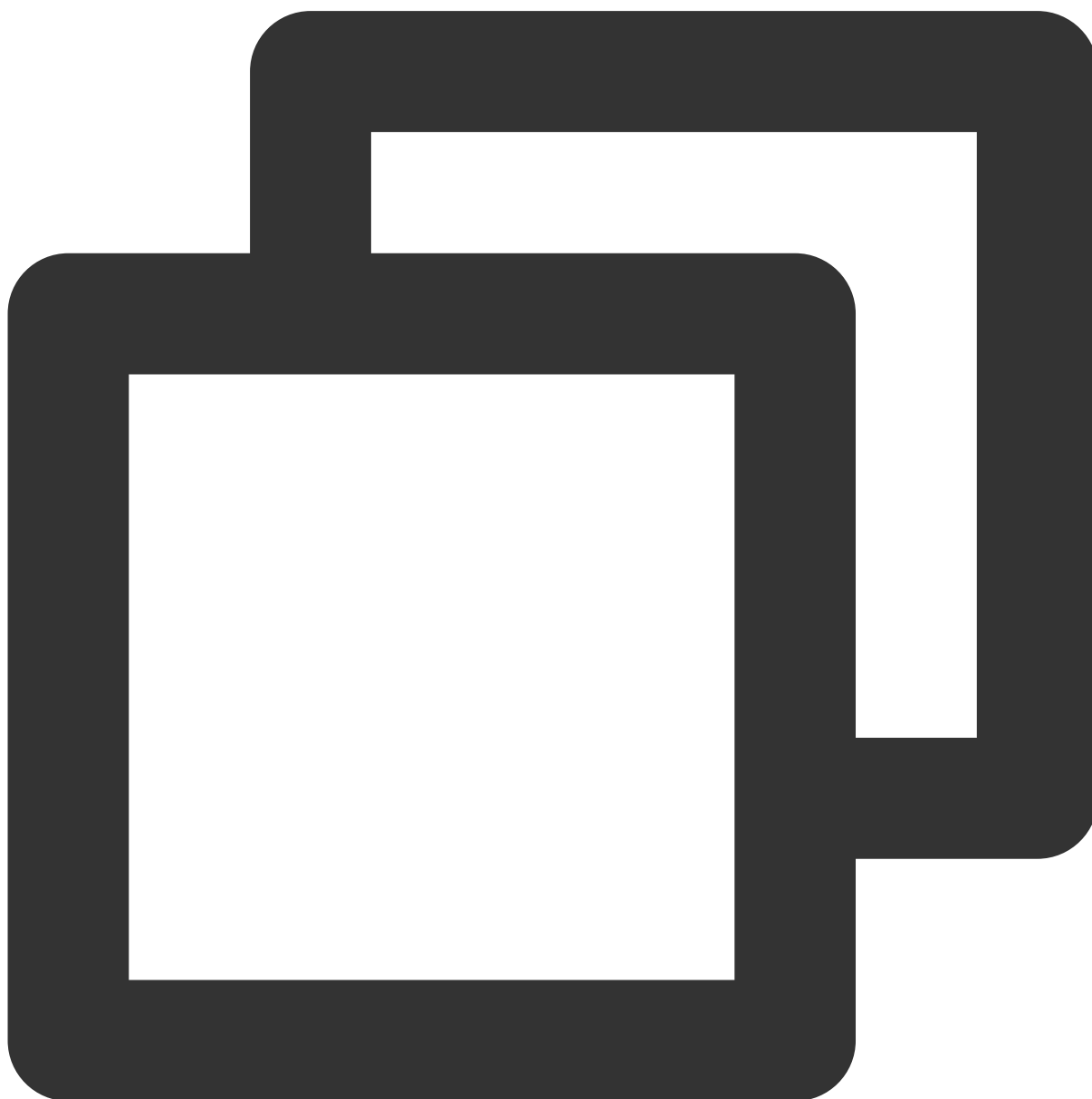


```
OnRemoteUserEnterRoom onRemoteUserEnterRoom = (String roomId, TUIUserInfo userInfo)
```

Parameter	Type	Description
roomId	String	Room ID
userInfo	<a href="#">TUIUserInfo</a>	User information

## onRemoteUserLeaveRoom

Remote user leaves the room event.

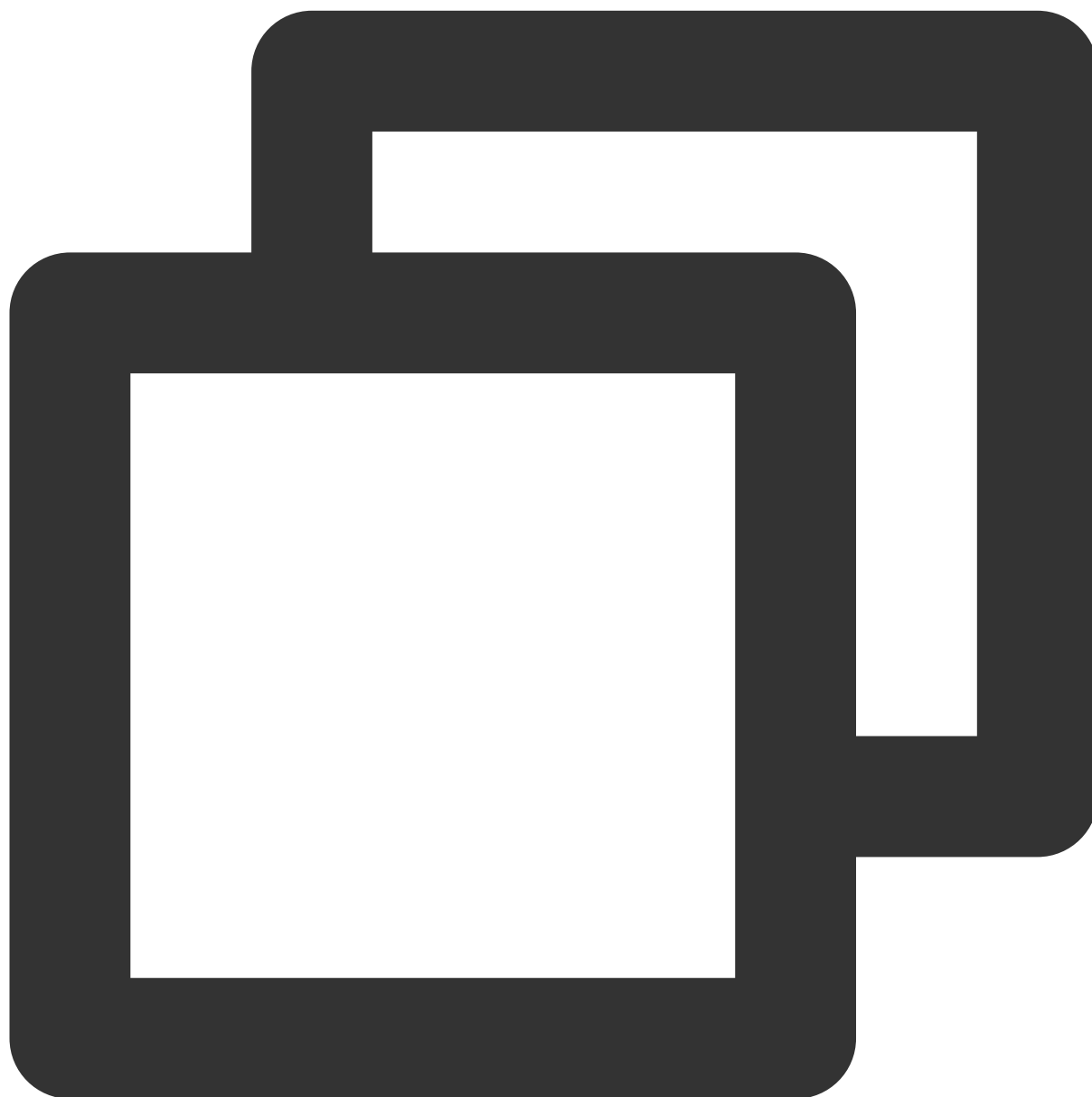


```
OnRemoteUserLeaveRoom onRemoteUserLeaveRoom = (String roomId, TUIUserInfo userInfo)
```

Parameter	Type	Description
roomId	String	Room ID
userInfo	<a href="#">TUIUserInfo</a>	User information

## onUserRoleChanged

User role changes event.

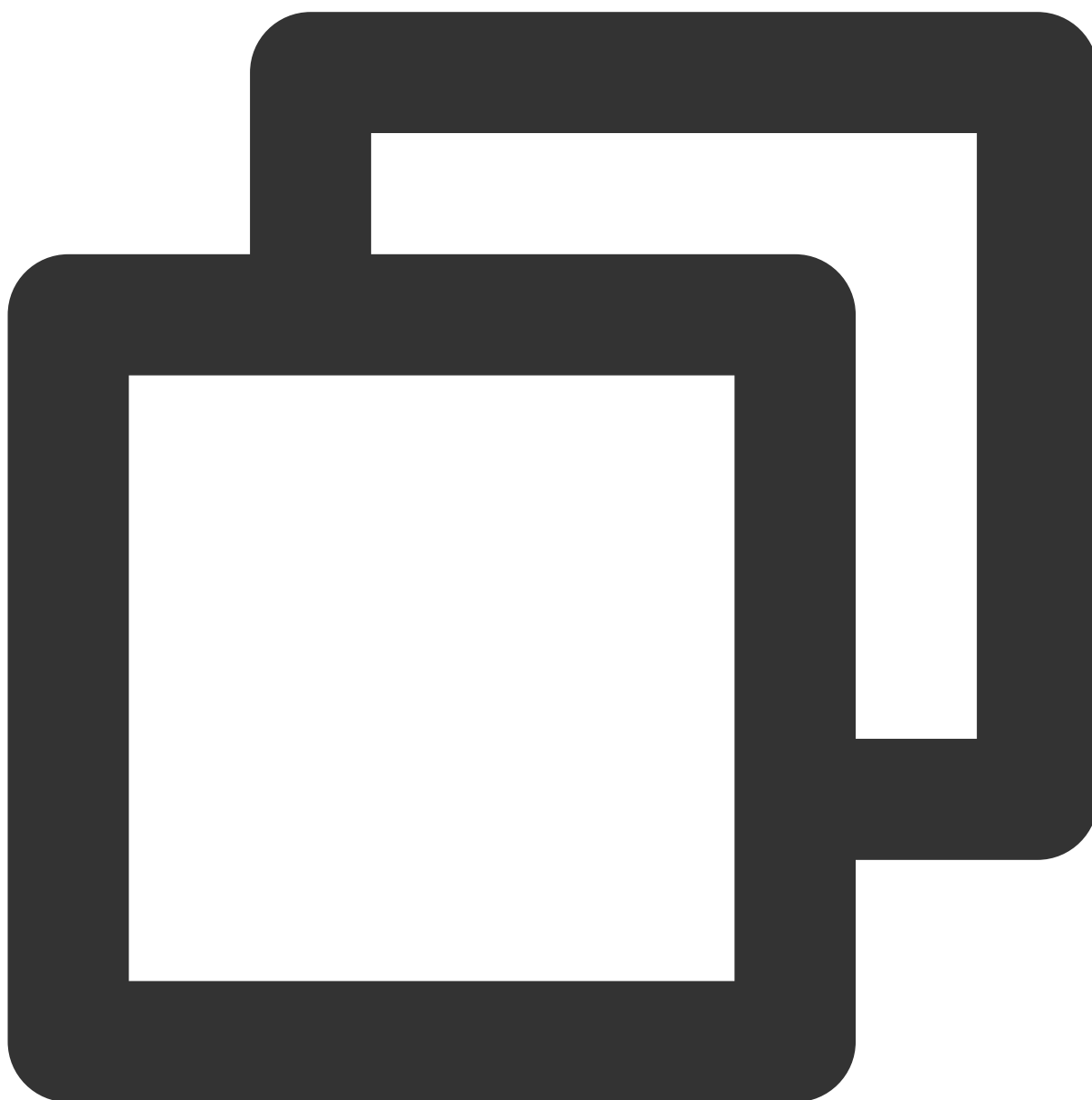


```
OnUserRoleChanged onUserRoleChanged = (String userId, TUIRole role) {}
```

Parameter	Type	Description
userId	String	User ID
role	<a href="#">TUIRole</a>	User Role

## onUserVideoStateChanged

User Video status changes event.



```
OnUserVideoStateChanged onUserVideoStateChanged = (String userId, TUIVideoStreamTyp
```

Parameter	Type	Description
userId	String	User ID
streamType	<a href="#">TUIVideoStreamType</a>	Streams type
hasVideo	bool	Whether there are streams
reason	<a href="#">TUIChangeReason</a>	Reason for streams change

## onUserAudioStateChanged

User Audio status changes event.

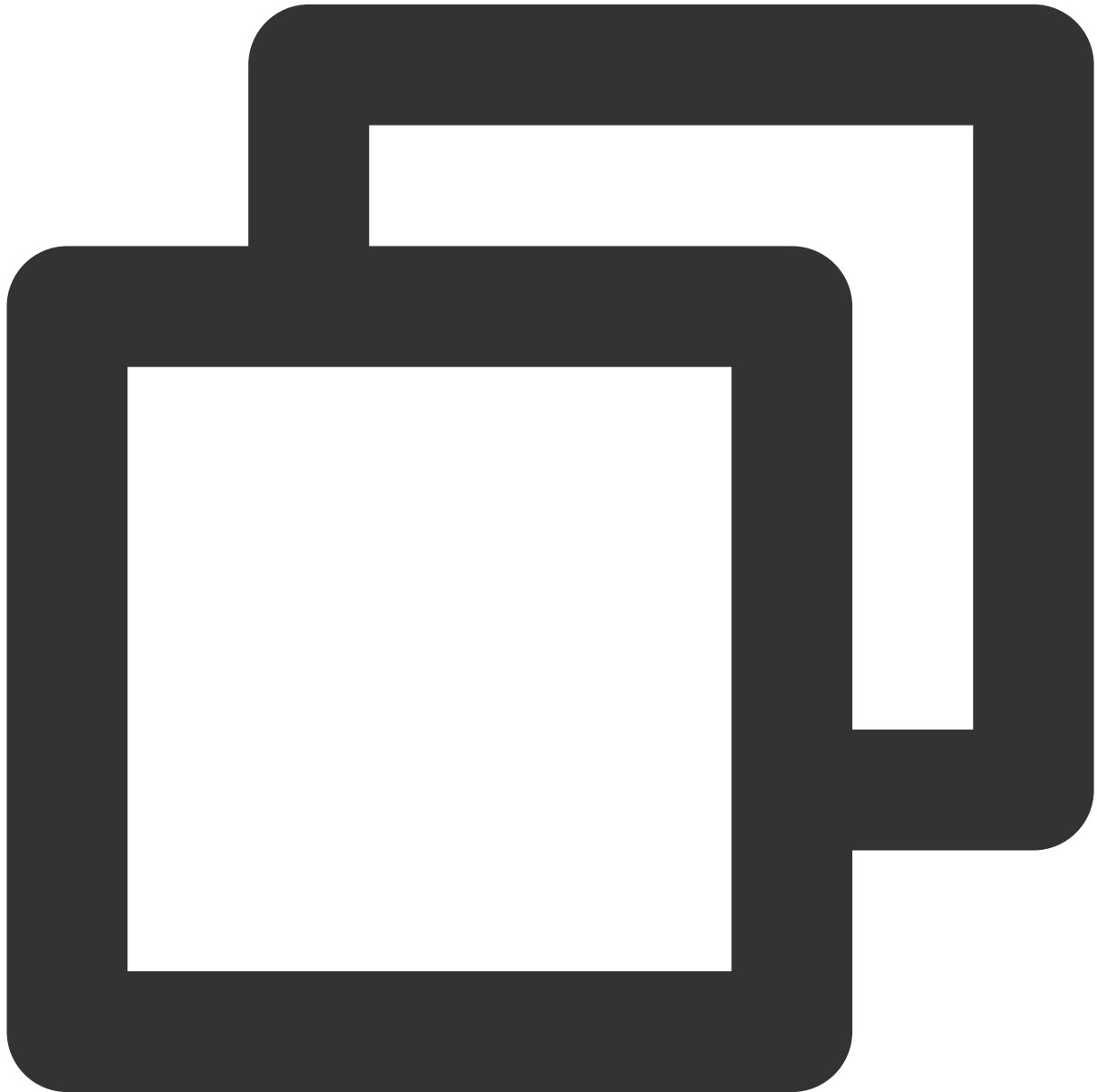


```
OnUserAudioStateChanged onUserAudioStateChanged = (String userId, bool hasAudio, TU
```

Parameter	Type	Description
userId	String	User ID
hasAudio	bool	Whether there are Audio streams
reason	<a href="#">TUIChangeReason</a>	Reason for Audio streams change

## onUserVoiceVolumeChanged

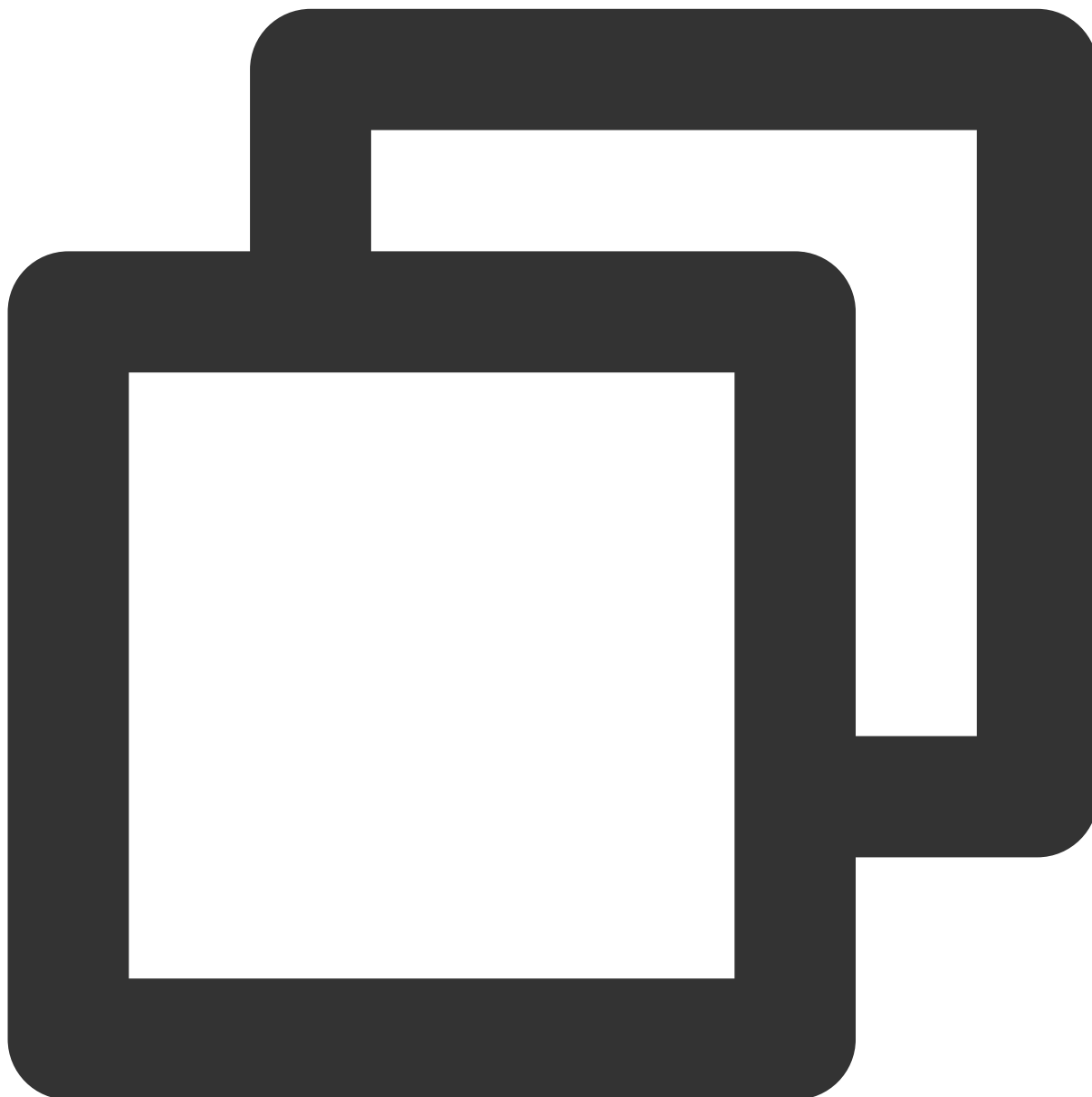
User volume change event.



```
OnUserVoiceVolumeChanged onUserVoiceVolumeChanged = (Map<String, int> volumeMap) {}
```

Parameter	Type	Description
volumeMap	Map	User Volume Map key: userId value: Used for carrying the volume size of all speaking users, Value range 0 - 100

User text message sending ability changes event.



```
OnSendMessageForUserDisableChanged onSendMessageForUserDisableChanged = (String rootElement) {
```

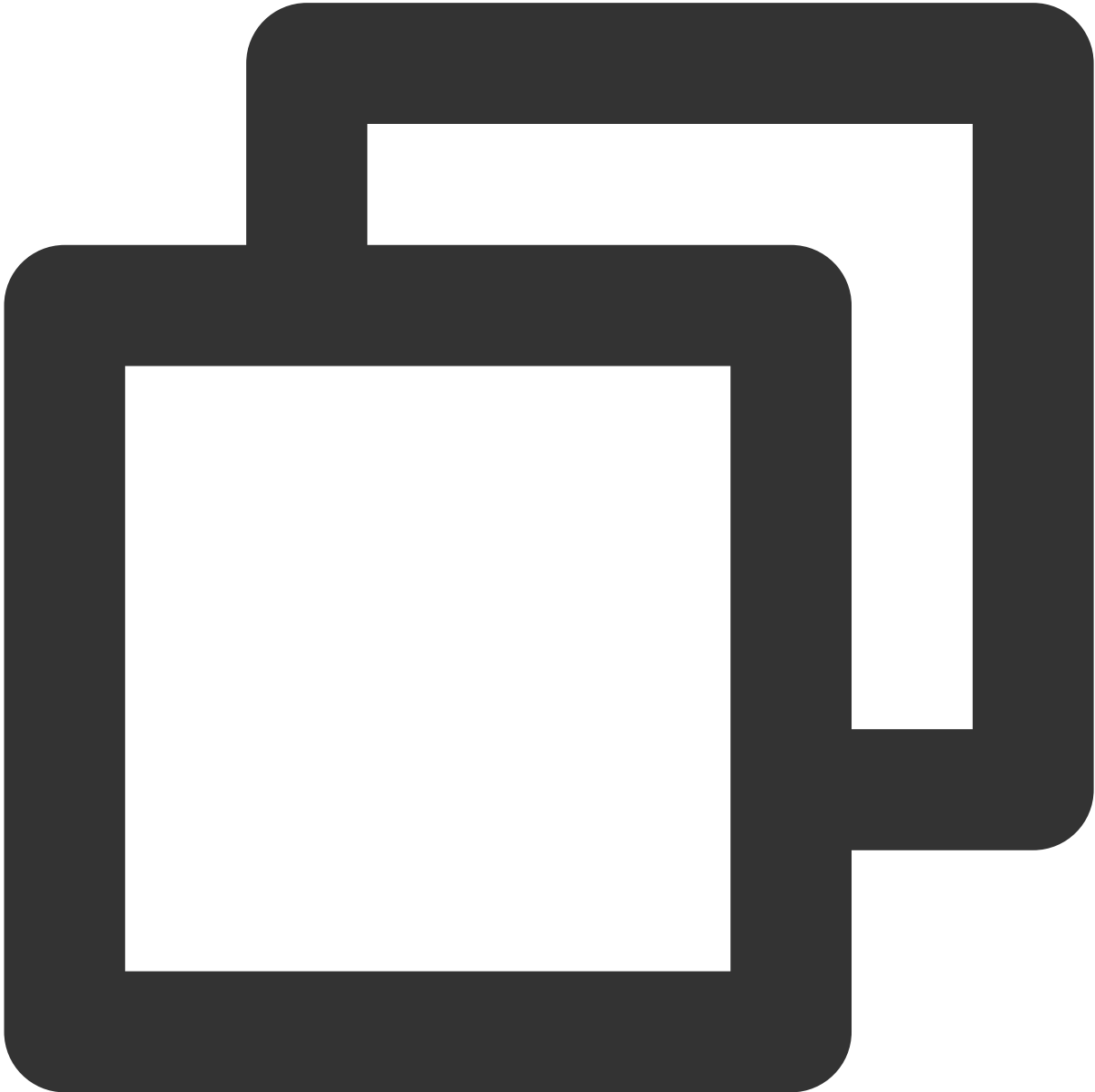
Parameter	Type	Description
roomId	String	Room ID



userId	String	User ID
isDisable	bool	Whether it is prohibited to send text messages.

onUserNetworkQualityChanged

User network status change event.



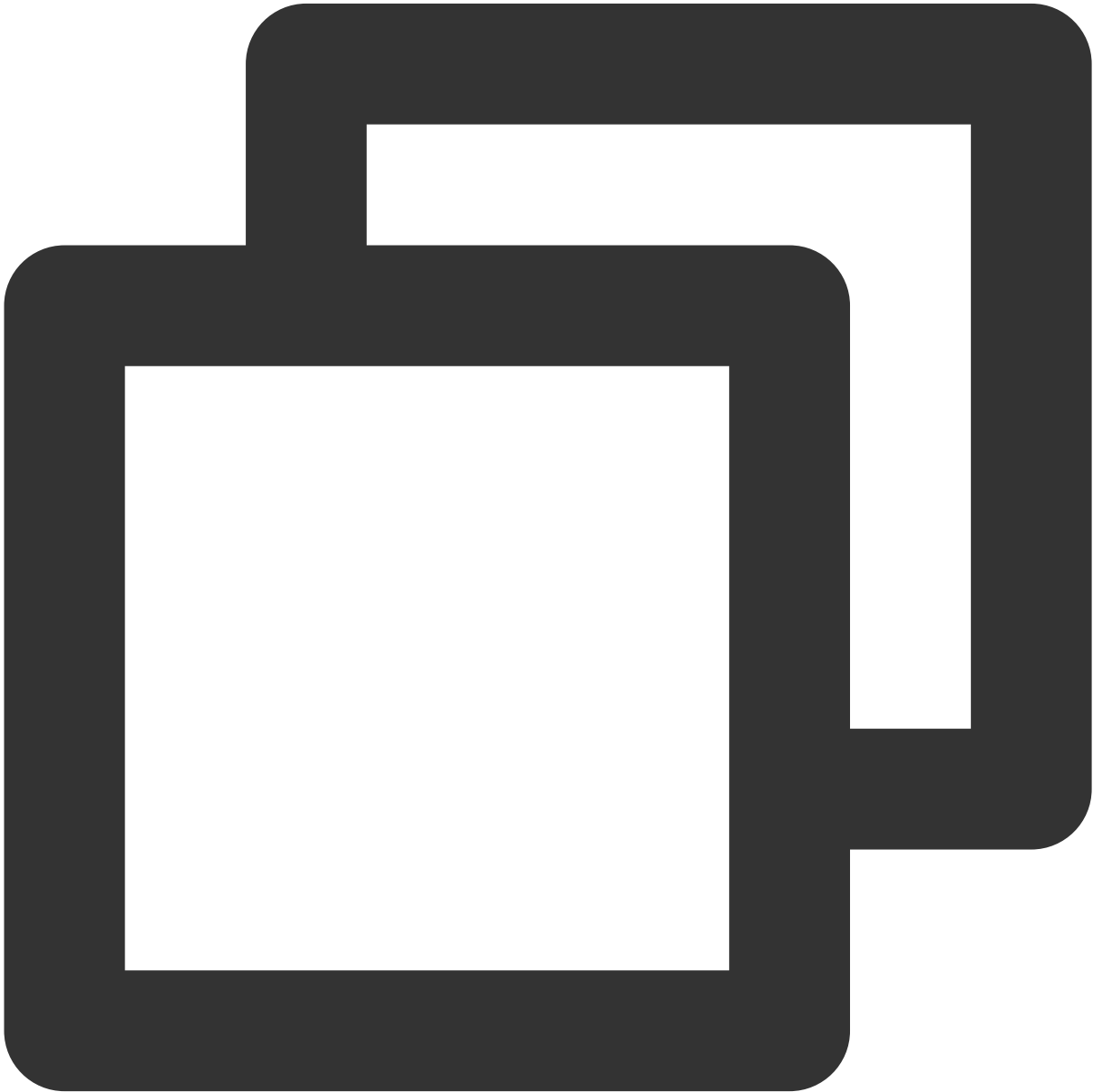
OnUserNetworkQualityChanged onUserNetworkQualityChanged = (Map<String, TUINetwork>

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

networkMap	Map	User Network Status Map key: userId value: Network Condition
------------	-----	--

**onUserScreenCaptureStopped**

Screen Sharing stopped Callback event.



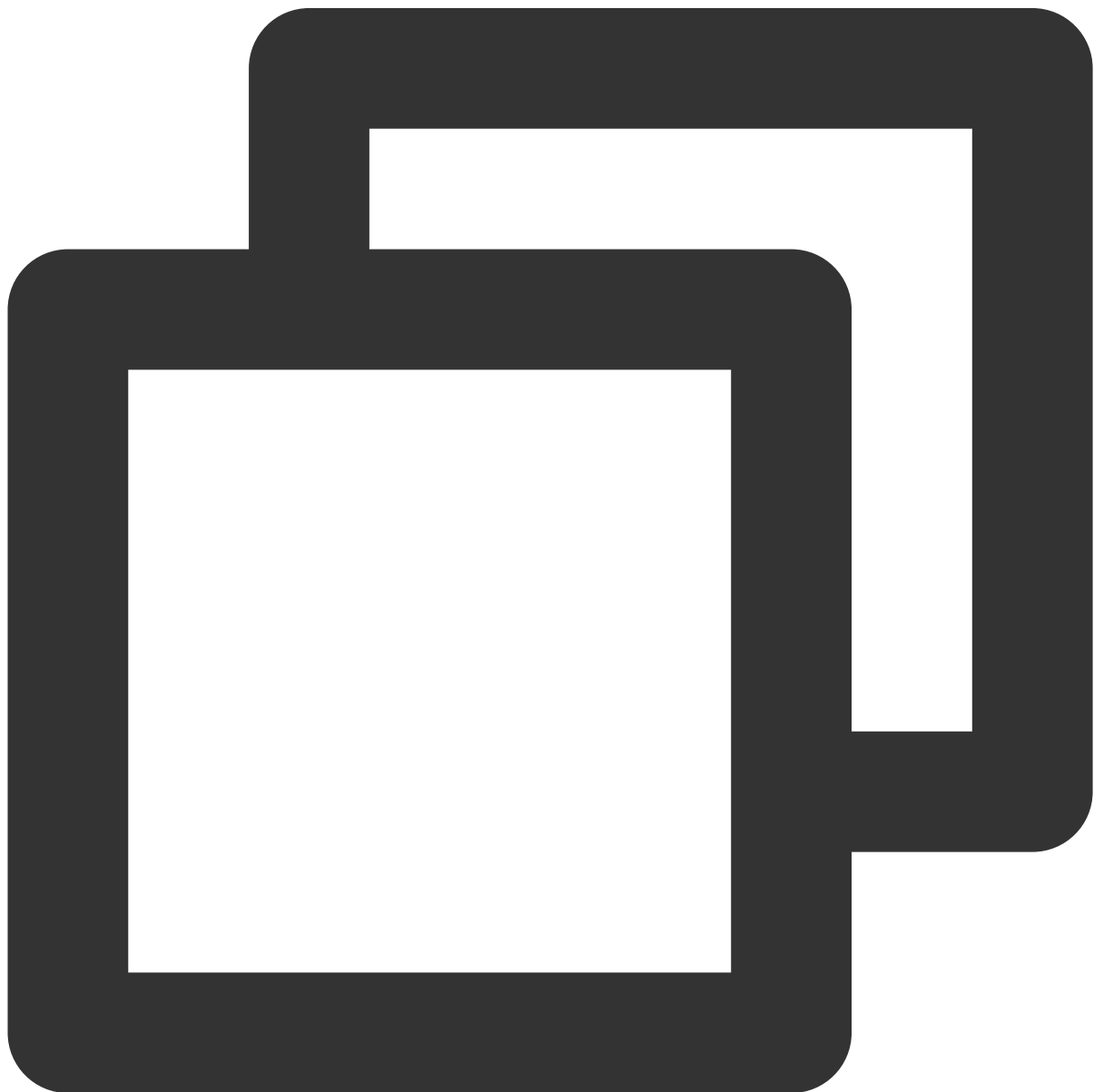
```
OnUserScreenCaptureStopped onUserScreenCaptureStopped = (int reason) {}
```

--	--	--

Parameter	Type	Description
reason	int	Stop reason: 0: User actively stops 1: Screen window closing causes the stop 2: Screen Sharing display screen status change (such as interface being unplugged, Projection mode change, etc.)

### onRoomMaxSeatCountChanged

Maximum number of mic slots changes event in the room (only effective in meeting type rooms).

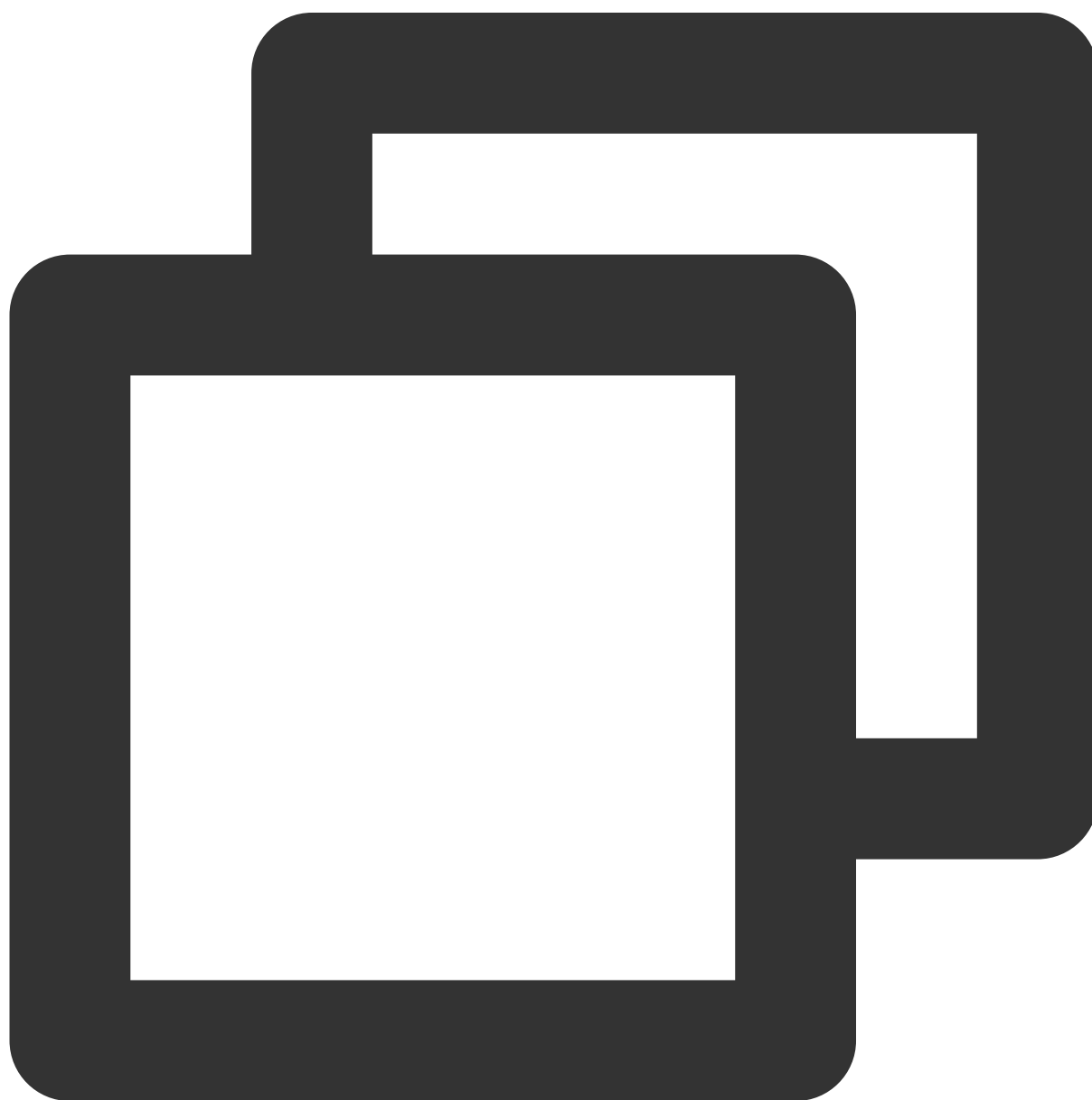


```
OnRoomMaxSeatCountChanged onRoomMaxSeatCountChanged = (String roomId, int maxSeatCo
```

Parameter	Type	Description
roomId	String	Room ID
maxSeatCount	int	Maximum number of mic slots in the room

## onSeatListChanged

Mic slot list changes event.

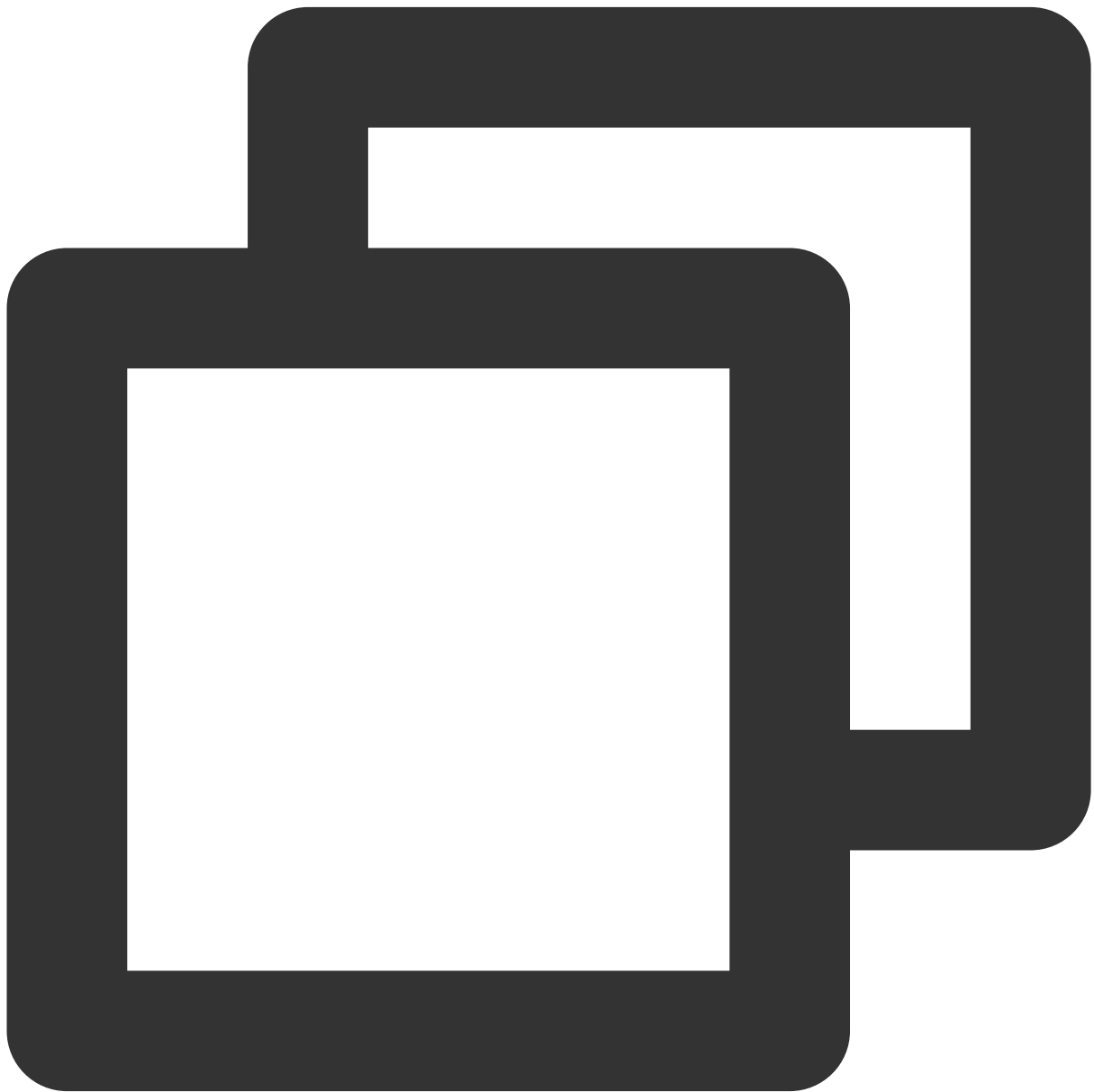


```
OnSeatListChanged onSeatListChanged = (List<TUISeatInfo> seatList, List<TUISeatInfo>
```

Parameter	Type	Description
seatList	List< <a href="#">TUISeatInfo</a> >	The latest user list on the mic, including newly on mic users
seatedList	List< <a href="#">TUISeatInfo</a> >	Newly on mic user list
leftList	List< <a href="#">TUISeatInfo</a> >	Newly off mic user list

## onKickedOffSeat

Received the event of user being kicked off mic.



```
OnKickedOffSeat onKickedOffSeat = (String userId) {}
```

Parameter	Type	Description
userId	String	Operate Kick-out of the (Host/Administrator) User ID

## onRequestReceived

Received request message event.



```
OnRequestReceived onRequestReceived = (TUIRequest request) {}
```

Parameter	Type	Description
request	<a href="#">TUIRequest</a>	Request content

### onRequestCancelled

Received request cancellation event.



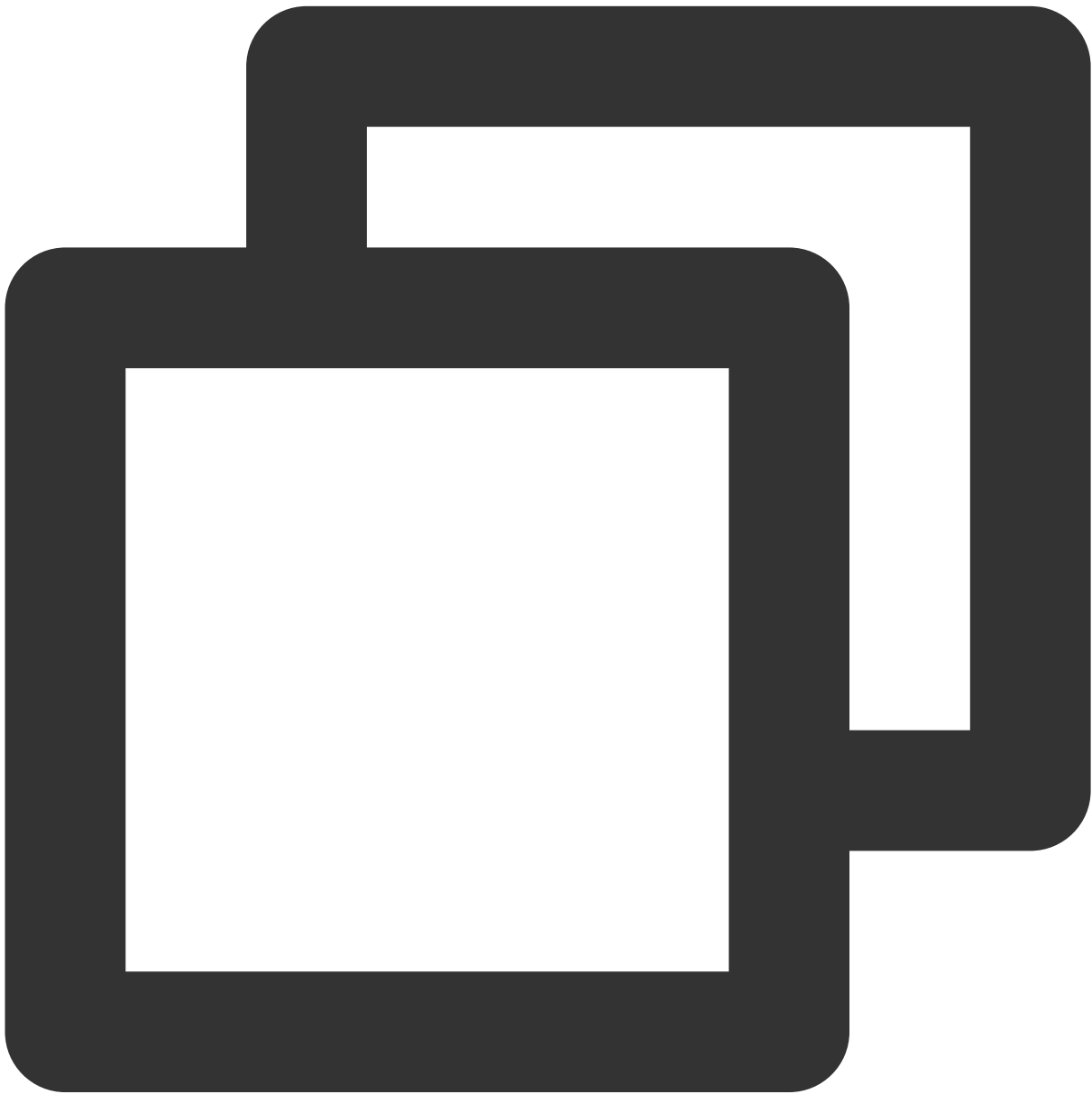
```
OnRequestCancelled onRequestCancelled = (String requestId, String userId) {}
```

Parameter	Type	Description
requestId	String	Request ID
userId	String	Cancel signaling user ID

## onReceiveTextMessage



Received ordinary text message event.

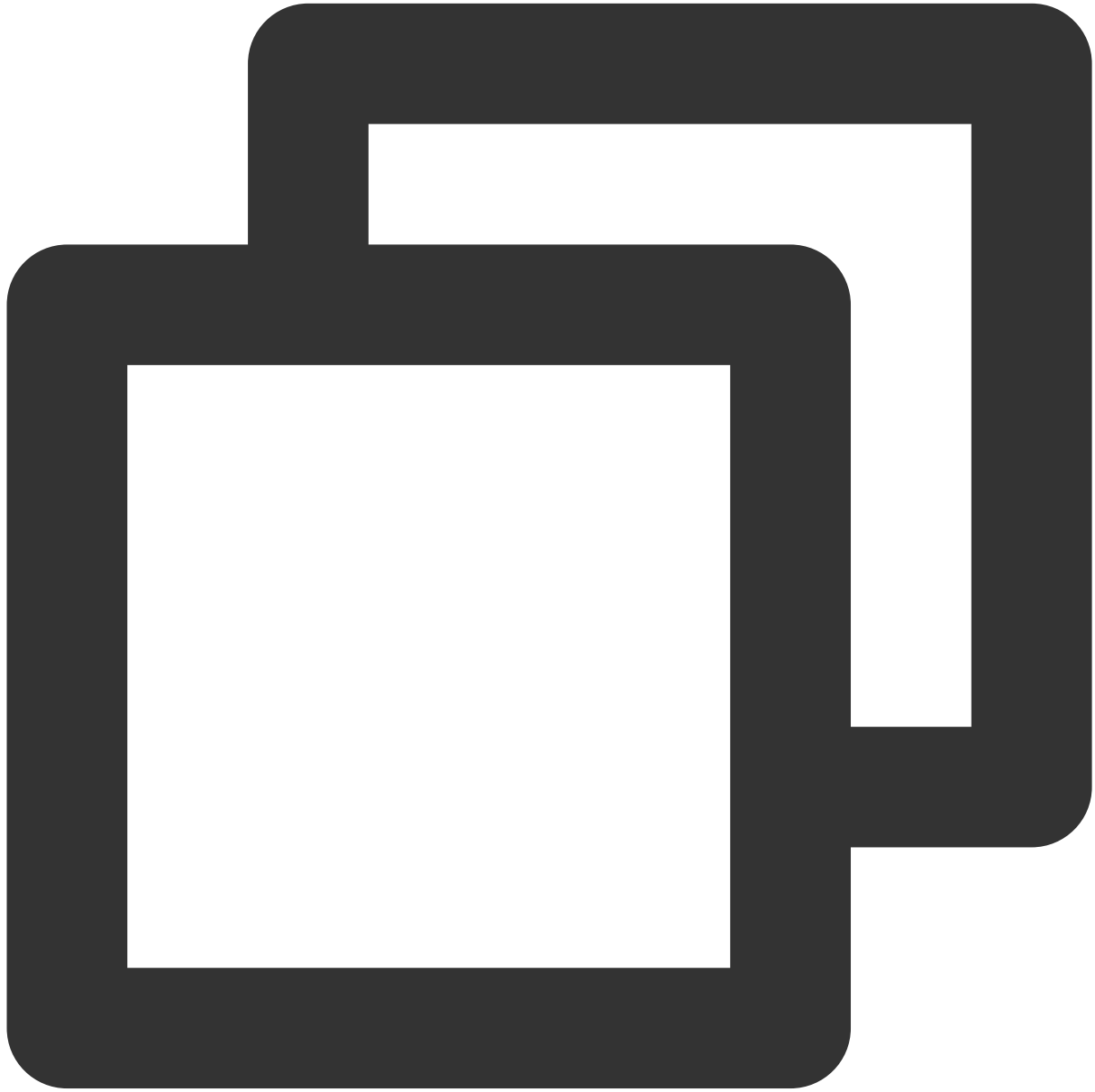


OnReceiveTextMessage onReceiveTextMessage = (String roomId, TUIMessage message) {}

Parameter	Type	Description
roomId	String	Room ID
message	TUIMessage	Message content

onReceiveCustomMessage

Received custom message event.



```
OnReceiveCustomMessage onReceiveCustomMessage = (String roomId, TUIMessage message)
```

Parameter	Type	Description
roomId	String	Room ID
message	<a href="#">TUIMessage</a>	Message content



# TUIRoomEngine

Last updated : 2023-11-21 15:00:32

## TUIRoomEngine API Introduction

TUIRoomEngine API is a No UI Interface for Conference Component, you can use this API to custom encapsulate according to your business needs.

### **createInstance**

Create TUIRoomEngine Instance.



```
static TUIRoomEngine createInstance()
```

**return:**TUIRoomEngine Instance

### **destroyInstance**

Destroy TUIRoomEngine Instance.



```
void destroyInstance()
```

## login

Login interface, you need to initialize user information before entering the room and perform a series of operations.



```
static Future<TUIActionCallback> login(int sdkAppId,  
                                       String userId,  
                                       String userSig)
```

**Parameters:**

Parameter	Type	Meaning
sdkAppId	int	Get sdkAppId information from Application Info
userId	String	User ID

userSig	String	UserSig
---------	--------	---------

## logout

Logout interface, there will be actively leave the room operation, destroy resources.

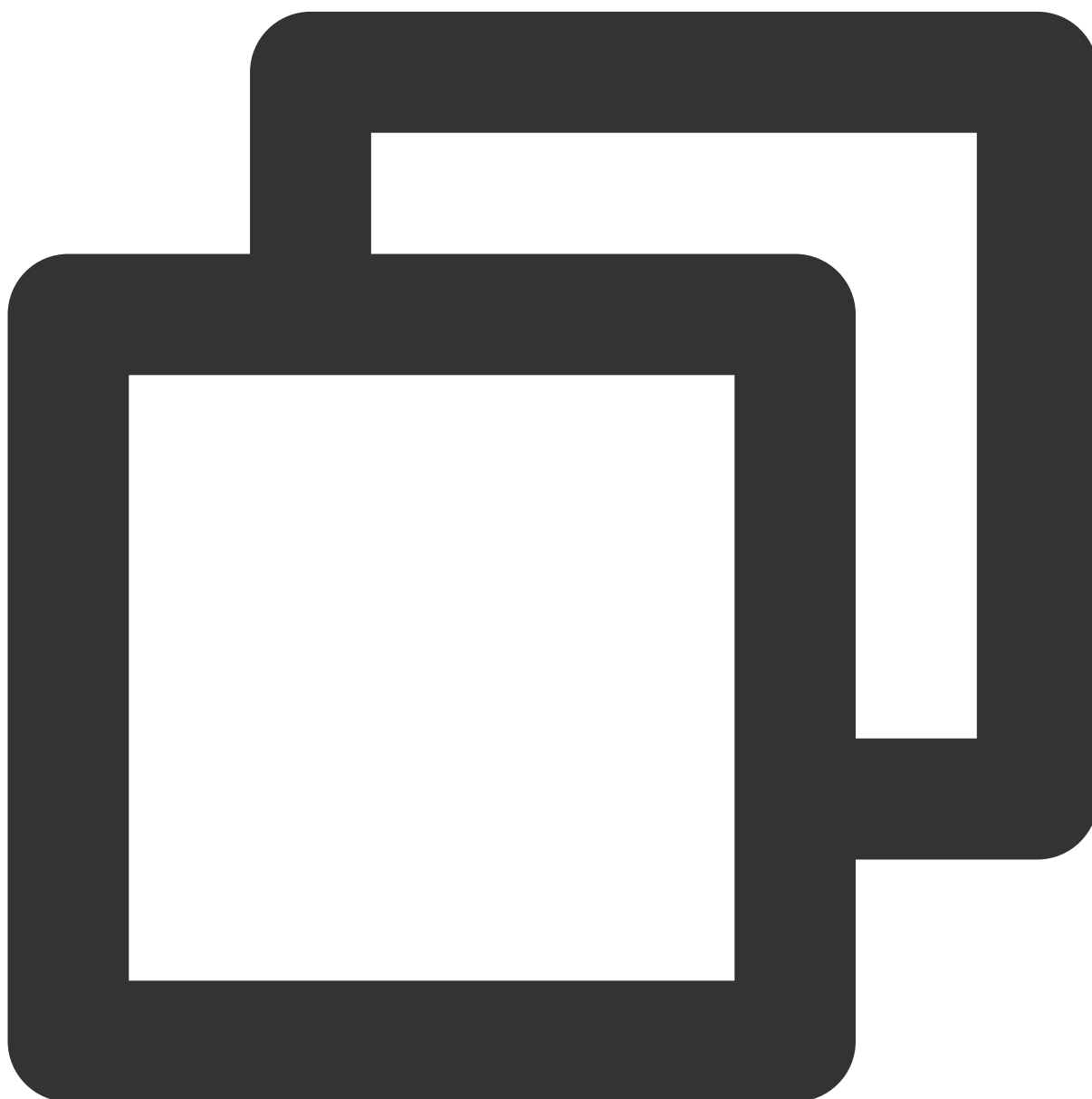


```
static Future<TUIActionCallback> logout ()
```

## setSelfInfo

Set local user name and avatar.





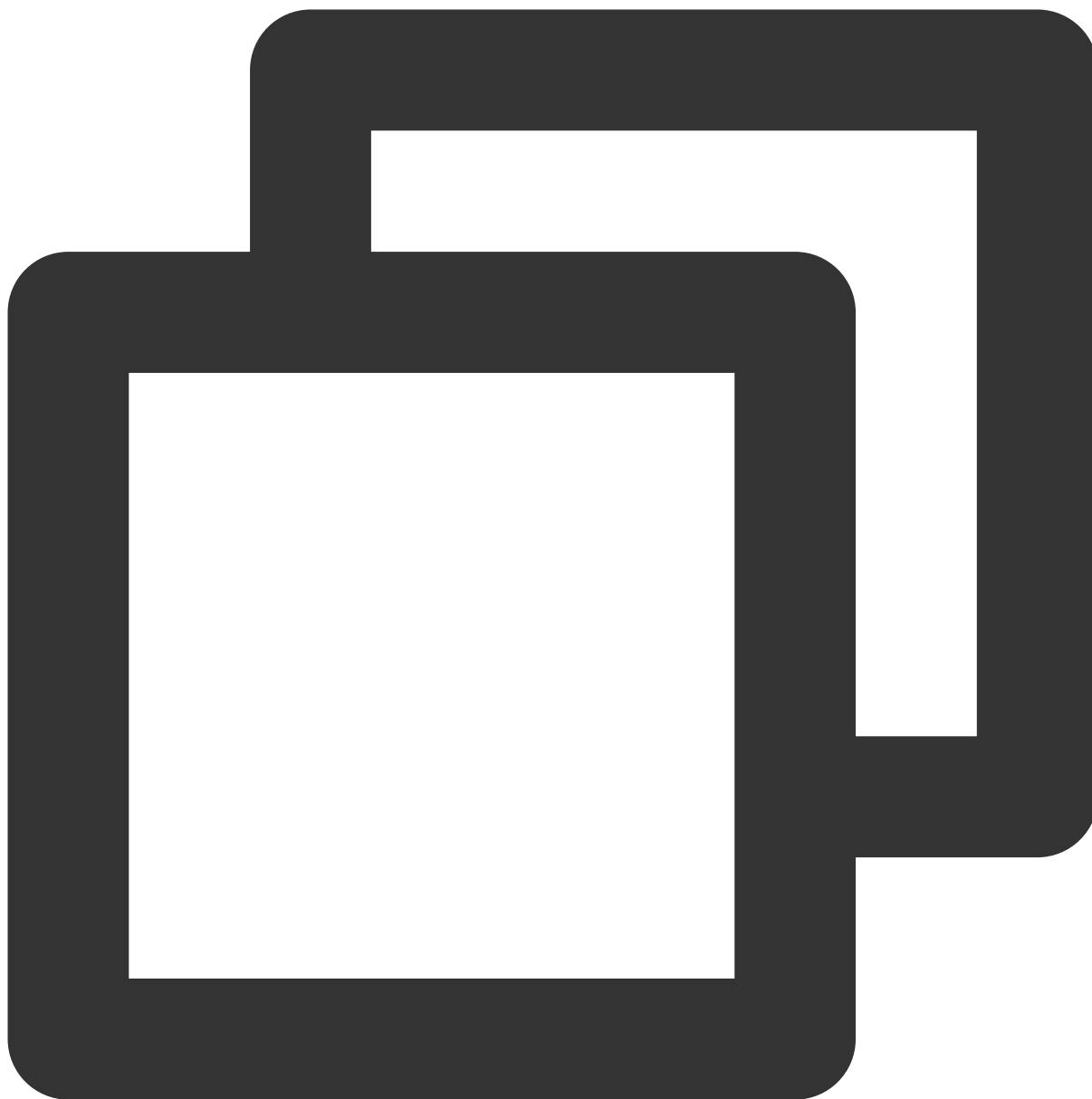
```
static Future<TUIActionCallback> setSelfInfo(String userName, String avatarURL)
```

**Parameters:**

Parameter	Type	Meaning
userName	String	User Name
avatarUrl	String	User avatar URL address

**setLoginUserInfo**

Set login user information.



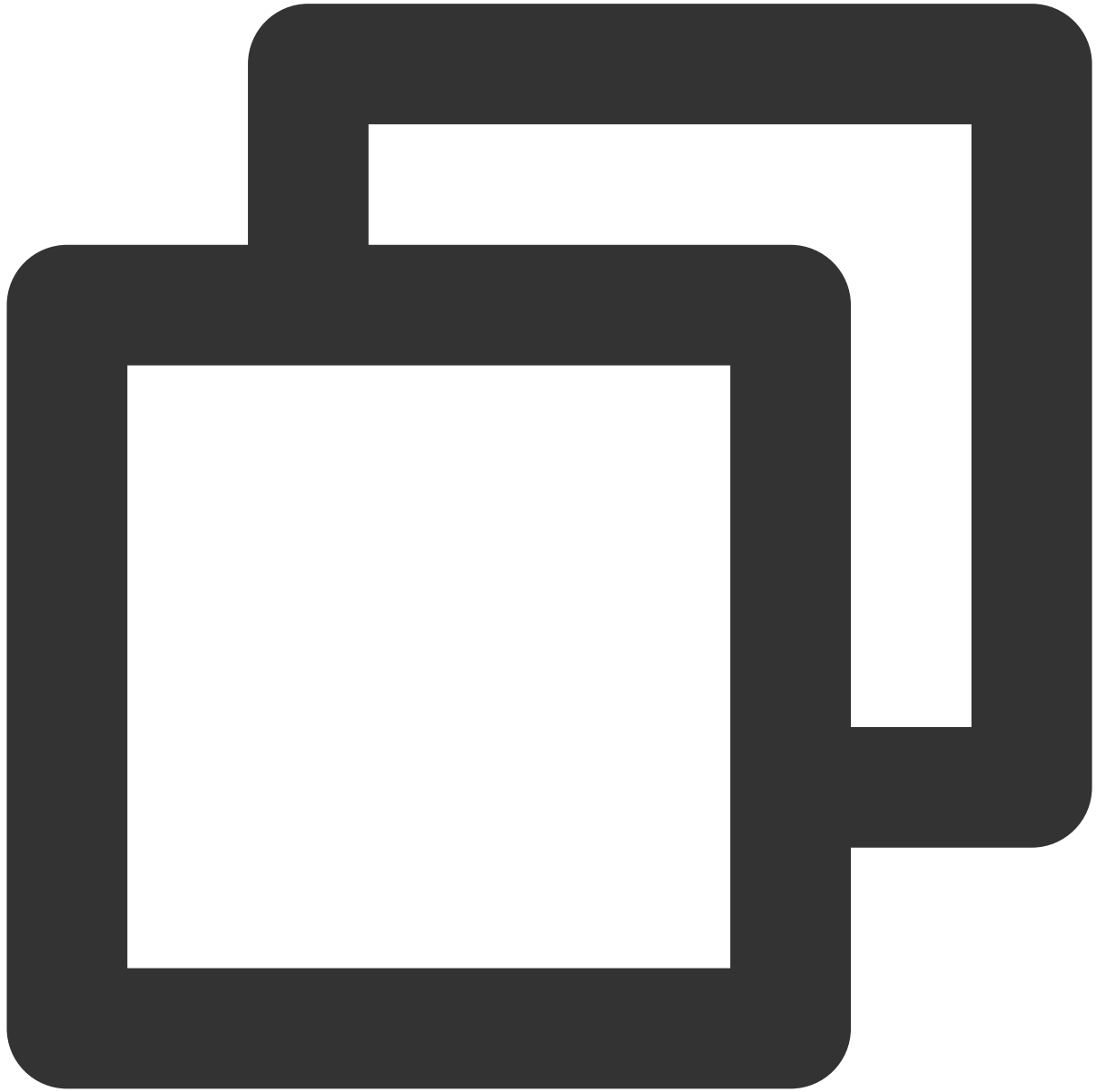
```
static Future<TUIActionCallback> setLoginUserInfo(TUILoginUserInfo userInfo)
```

**Parameters:**

Parameter	Type	Meaning
userInfo	TUILoginUserInfo	User information

**getSelfInfo**

Get the basic information of the local user login.

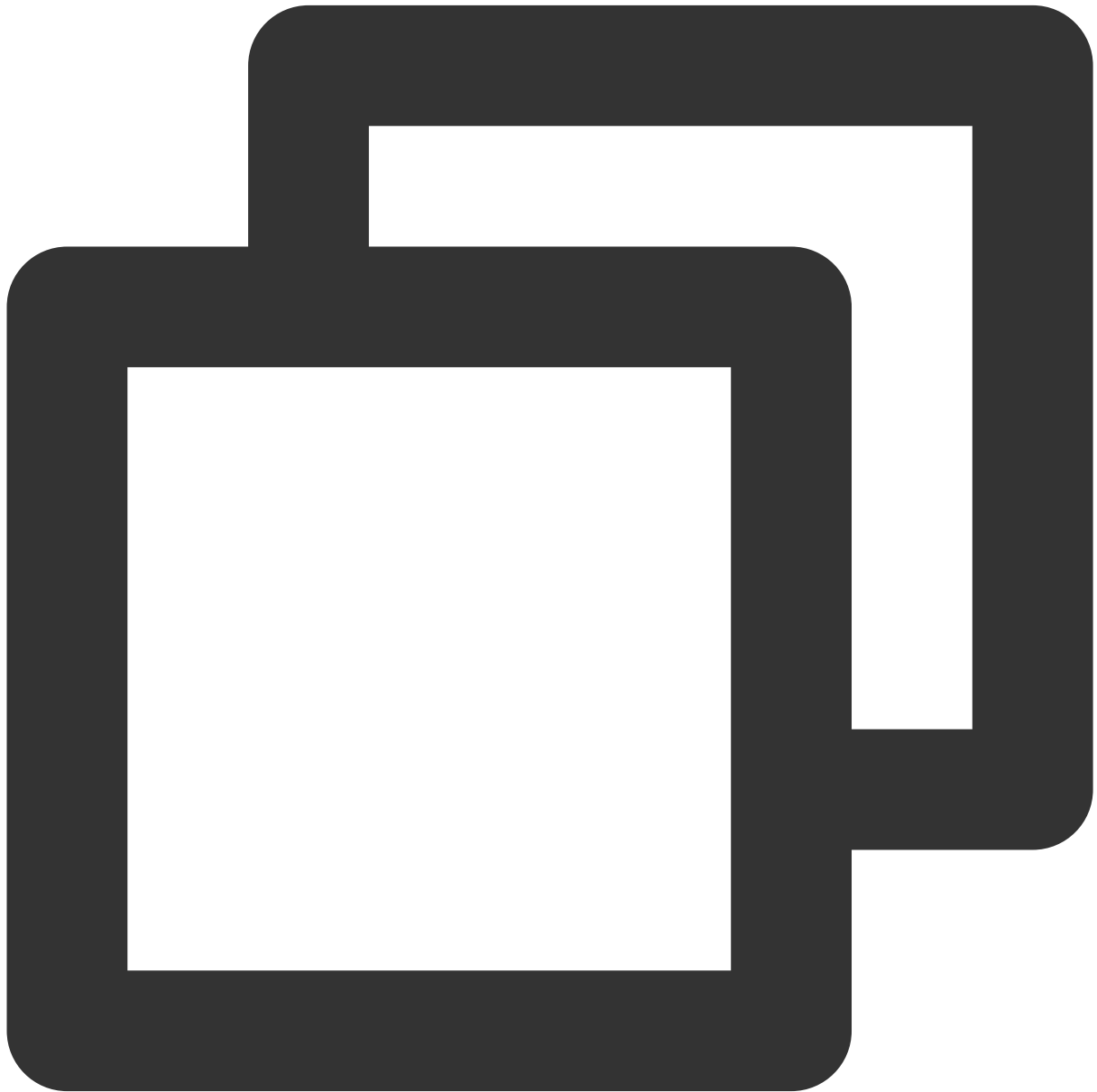


```
static TUILoginUserInfo getSelfInfo()
```

**return:**the basic information of the local user login

### **addObserver**

Add TUIRoomEngine Event Callback.



```
void addObserver(TUIRoomObserver observer)
```

**Parameters:**

Parameter	Type	Meaning
observer	TUIRoomObserver	TUIRoomEngine Event Callback

**removeObserver**

Remove TUIRoomEngine Event Callback.



```
void removeObserver(TUIRoomObserver observer)
```

Parameter	Type	Meaning
observer	TUIRoomObserver	TUIRoomEngine Event Callback

## createRoom

The host creates a room, and the user who calls createRoom is the owner of the room. When creating a room, you can set the Room ID, room name, and whether the room allows users to join, enable video and audio, send messages, and

other functions.



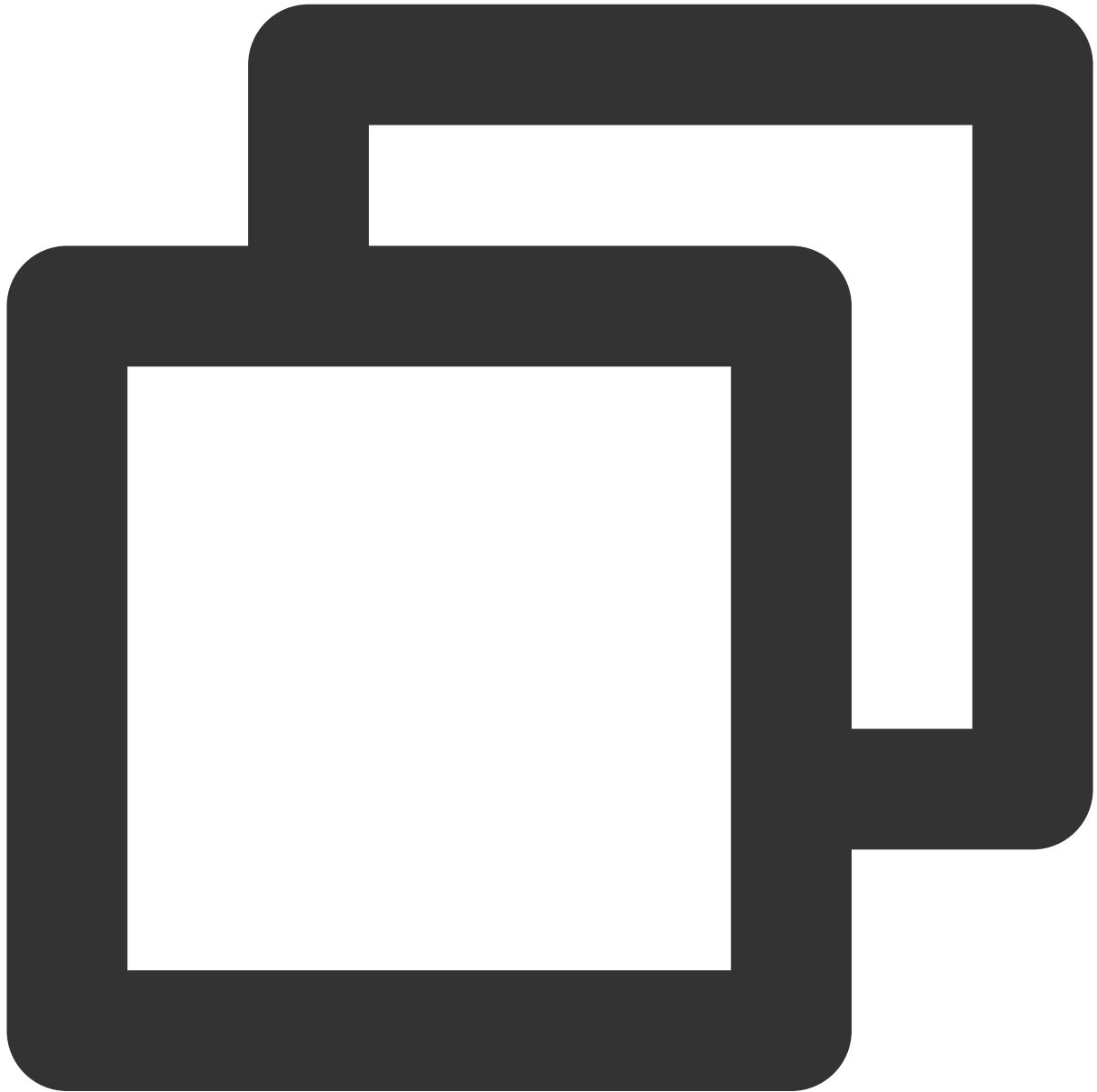
```
Future<TUIActionCallback> createRoom(TUIRoomInfo roomInfo)
```

#### Parameters:

Parameter	Type	Meaning
roomInfo	<a href="#">TUIRoomInfo</a>	Room data

#### destroyRoom

Destroy Room Interface, the room owner must initiate the destruction of the room. After the room is destroyed, it is unavailable for entry.



```
Future<TUIActionCallback> destroyRoom()
```

## enterRoom

Entered room.



```
Future<TUIValueCallBack<TUIRoomInfo>> enterRoom(String roomId)
```

**Parameters:**

Parameter	Type	Meaning
roomId	String	Room ID

**exitRoom**

Exit Room Interface, after the user executes Enter Room, they can leave the room through Leave Room.





```
Future<TUIActionCallback> exitRoom(bool syncWaiting)
```

**Parameters:**

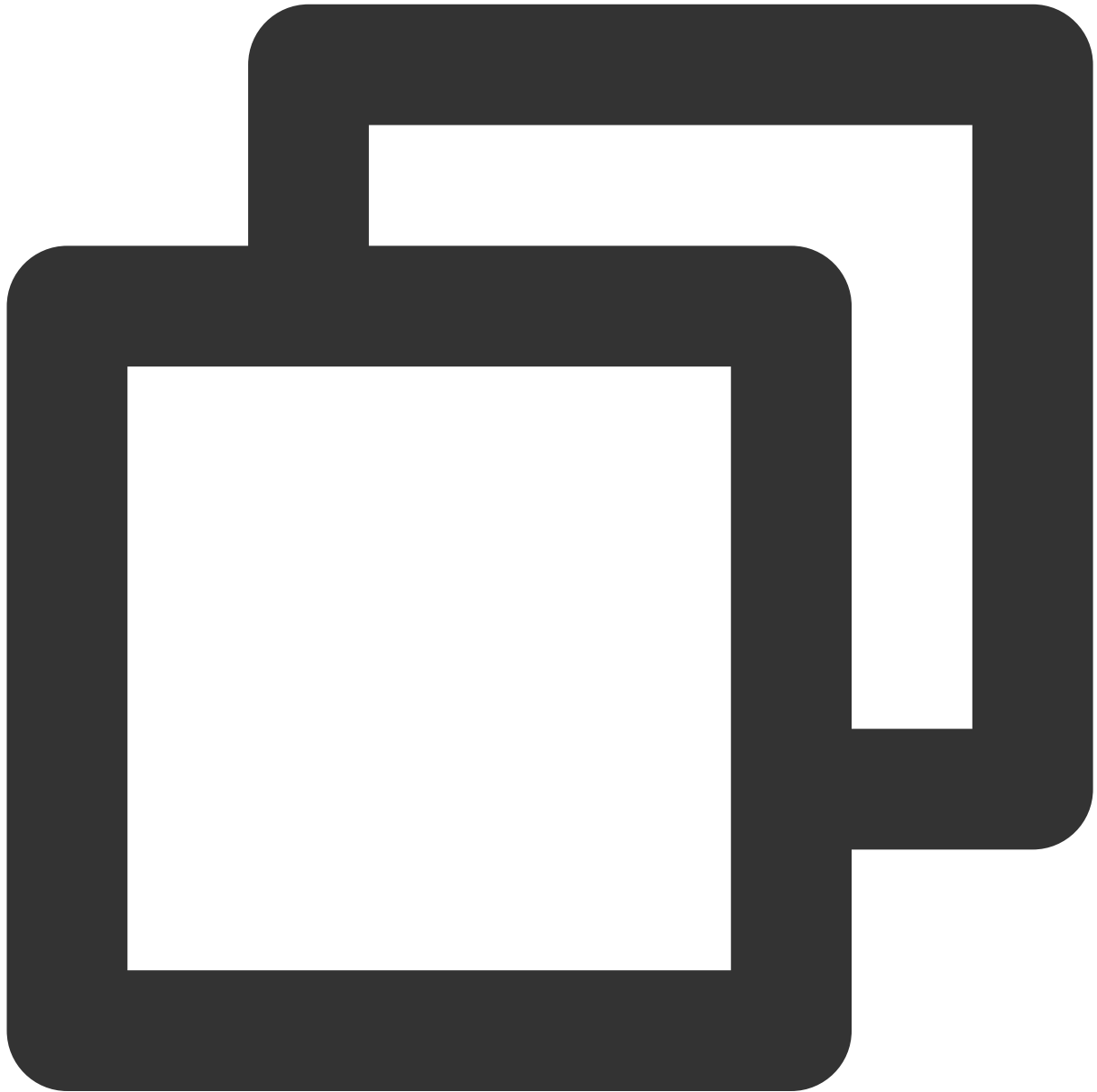
Parameter	Type	Meaning
syncWaiting	bool	Whether to synchronize leaving the room

**connectOtherRoom**

Connect to other rooms.

**Description:**

Used for live streaming scenario to apply for cross-room streaming



```
TUIRequest connectOtherRoom(String roomId,  
                             String userId,  
                             int timeout,  
                             TUIRequestCallback? requestCallback)
```

**Parameters:**

Parameter	Type	Meaning

roomId	String	Room ID
userId	String	User ID
timeout	int	Time
callback	TUIRequestCallback	Connect to other rooms Callback

**Return :** Request body

### **disconnectOtherRoom**

Disconnect from other rooms

**Description:**

Used for live streaming scenario to disconnect cross-room streaming



```
Future<TUIActionCallback> disconnectOtherRoom()
```

### **fetchRoomInfo**

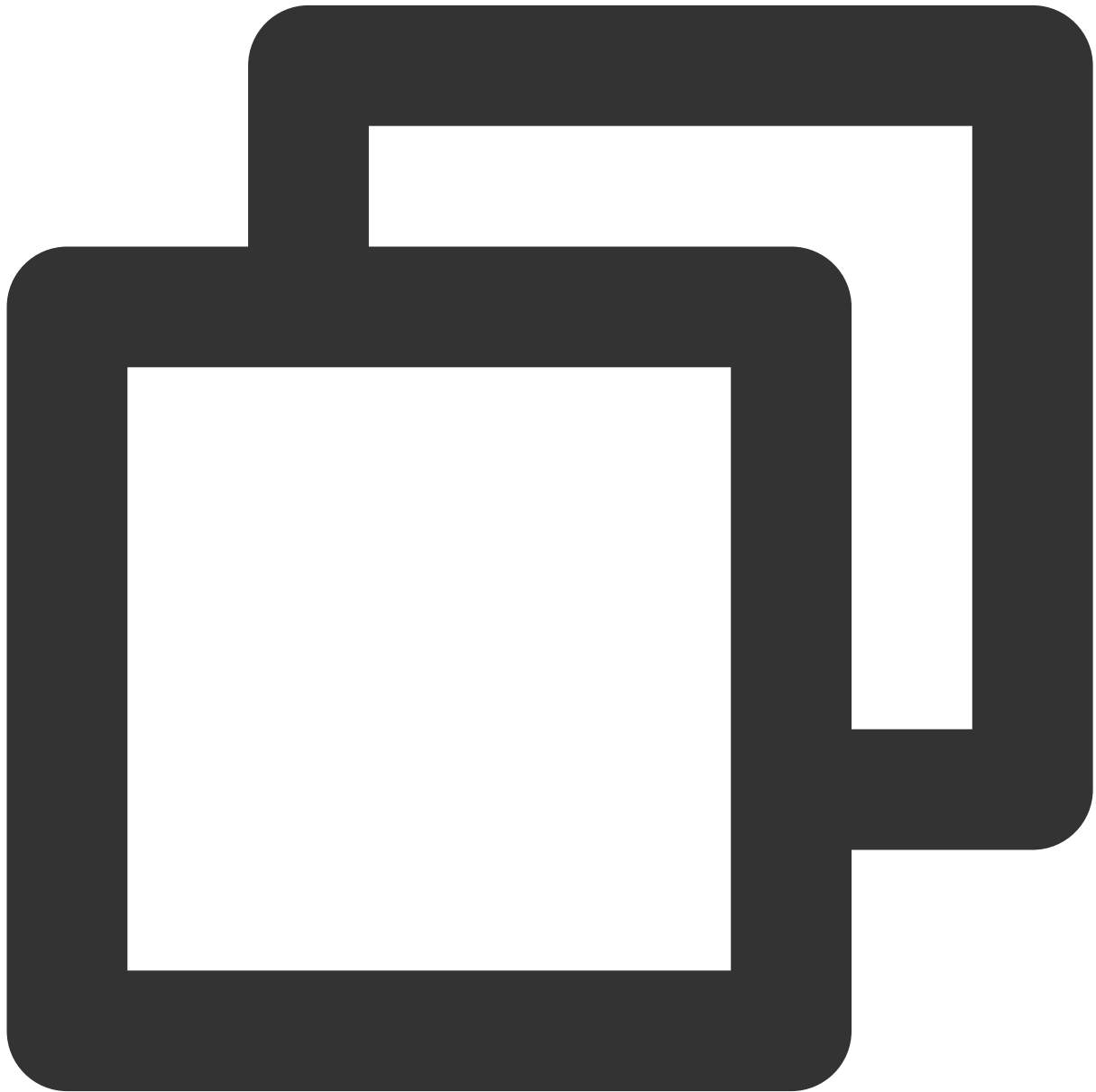
Get Room data.



```
Future<TUIValueCallBack<TUIRoomInfo>> fetchRoomInfo()
```

### **updateRoomNameByAdmin**

Update Room ID.



```
Future<TUIActionCallback> updateRoomNameByAdmin(String roomName)
```

**Parameters:**

Parameter	Type	Meaning
roomName	String	Room ID

**updateRoomSpeechModeByAdmin**

Set Management mode (only Administrator or Group owner can call).



```
Future<TUIActionCallback> updateRoomSpeechModeByAdmin(TUISpeechMode mode)
```

Parameter	Type	Meaning
mode	<a href="#">TUISpeechMode</a>	Management mode

## setLocalVideoView

Set Local user Video Rendering View control.



```
void setLocalVideoView(TUIVideoStreamType streamType,  
                       int viewId)
```

**Parameters:**

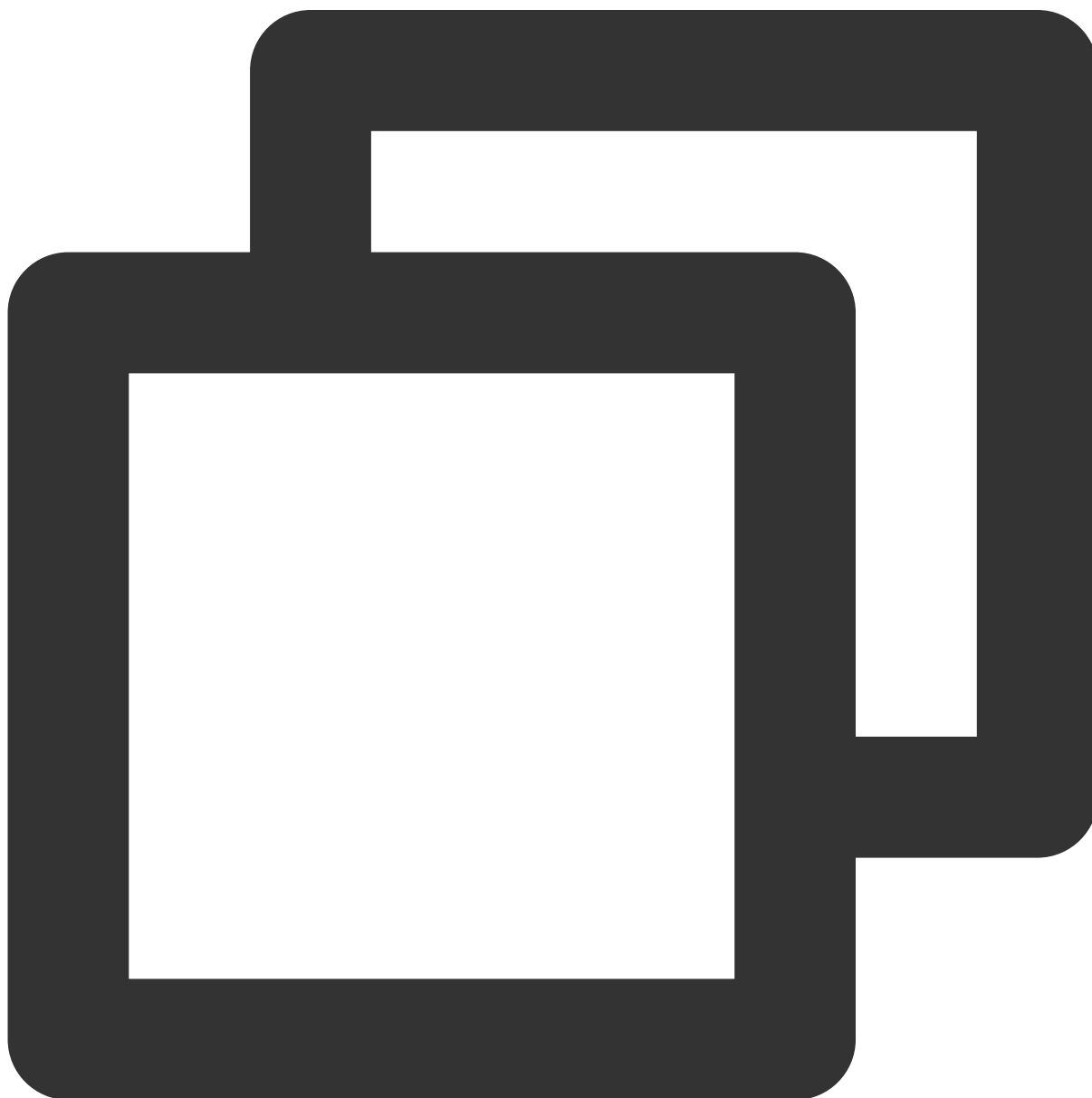
Parameter	Type	Meaning
streamType	<a href="#">TUIVideoStreamType</a>	Local streams type
viewId	int	The int64 type value of the pointer to the view to be rendered, through this viewId, can be converted to the



corresponding native platform view, and the video screen will be rendered on this view.

## openLocalCamera

Open Local Camera, Start Video Capturing.



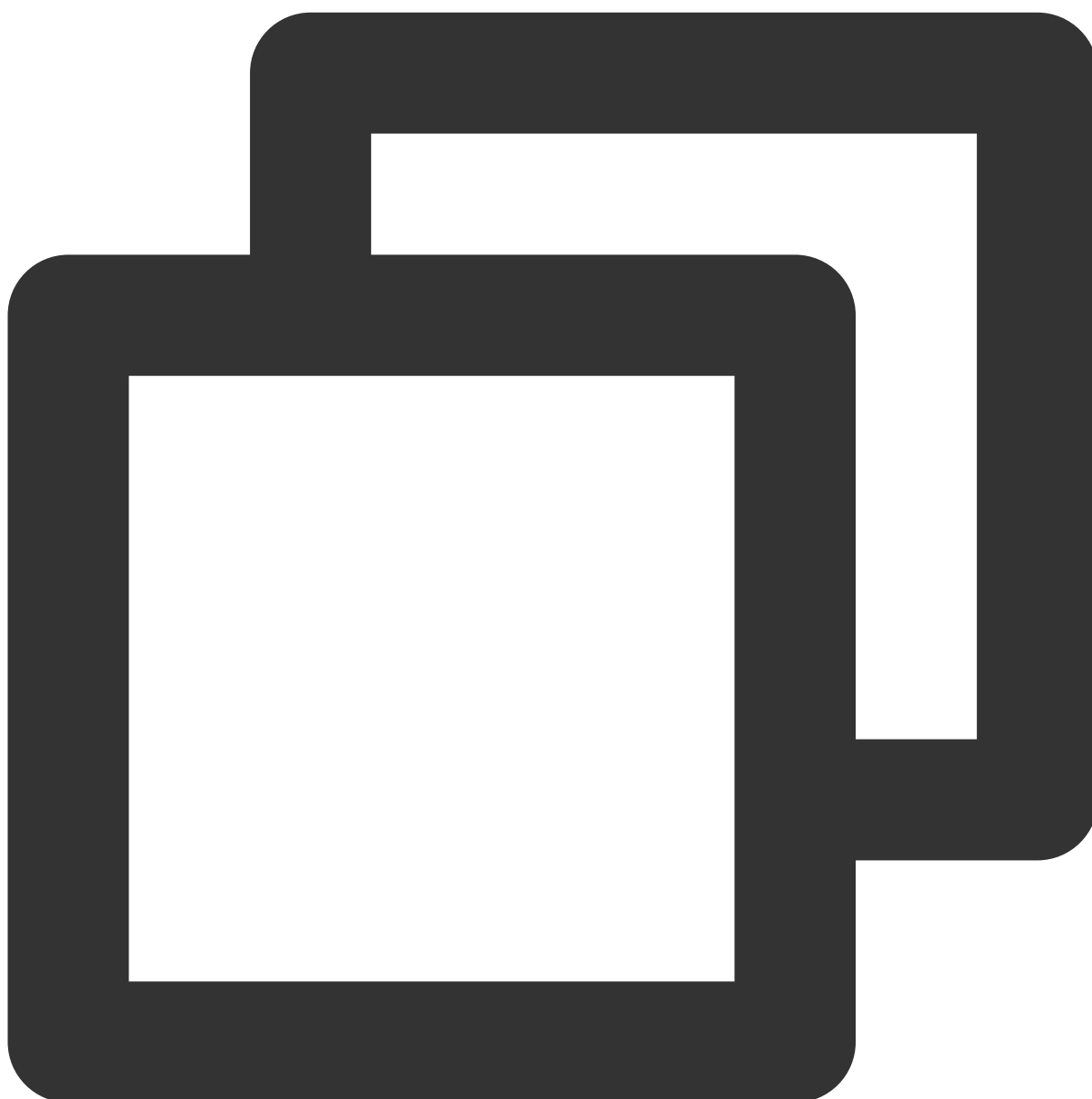
```
Future<TUIActionCallback> openLocalCamera(bool isFront,  
                                           TUIVideoQuality quality)
```

### Parameters:

Parameter	Type	Meaning
isFront	bool	Whether to use Front Camera
quality	<a href="#">TUIVideoQuality</a>	Video Quality

## closeLocalCamera

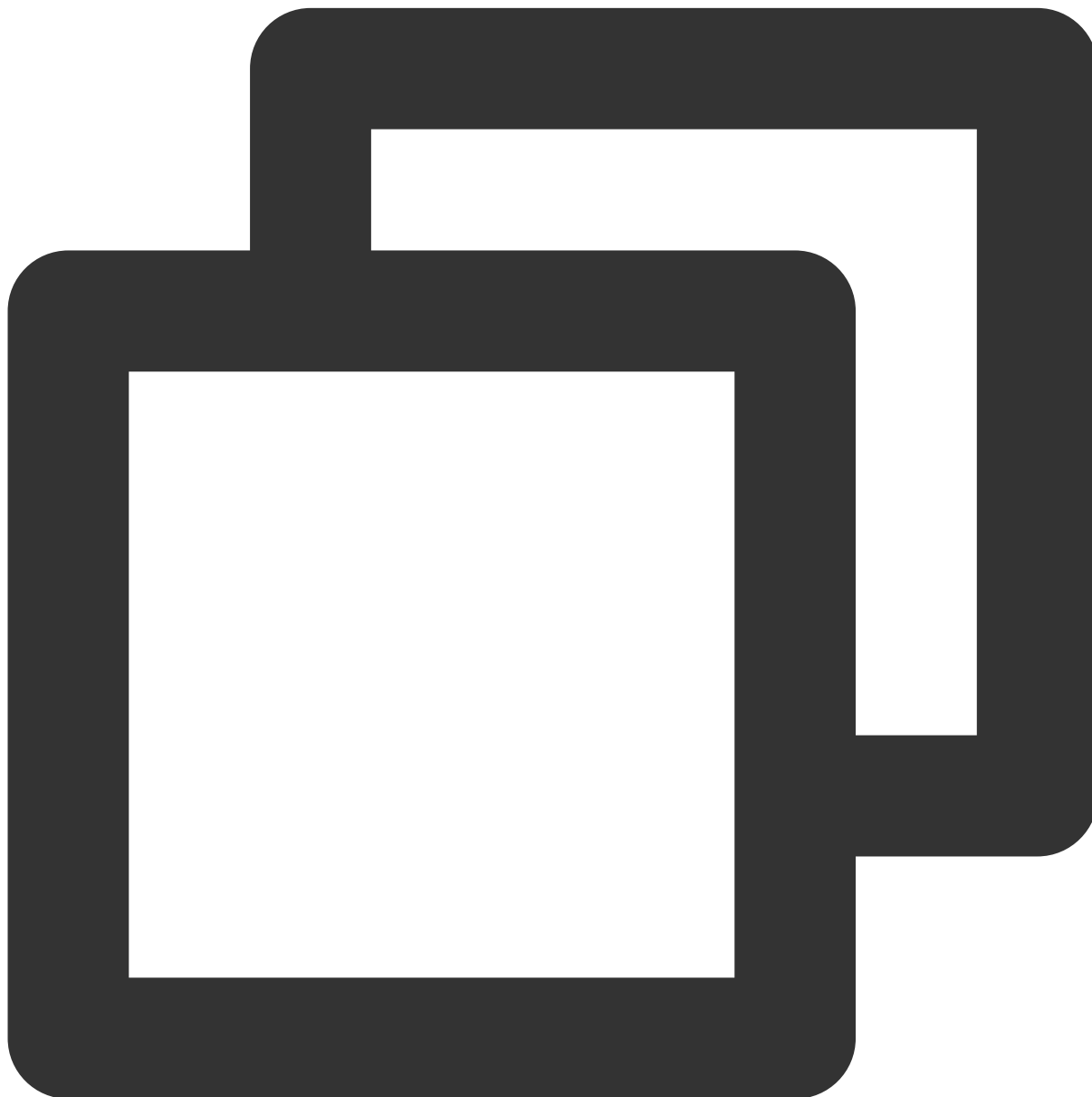
Close Local Camera.



```
void closeLocalCamera()
```

## updateVideoQuality

Set Local Video Parameter.



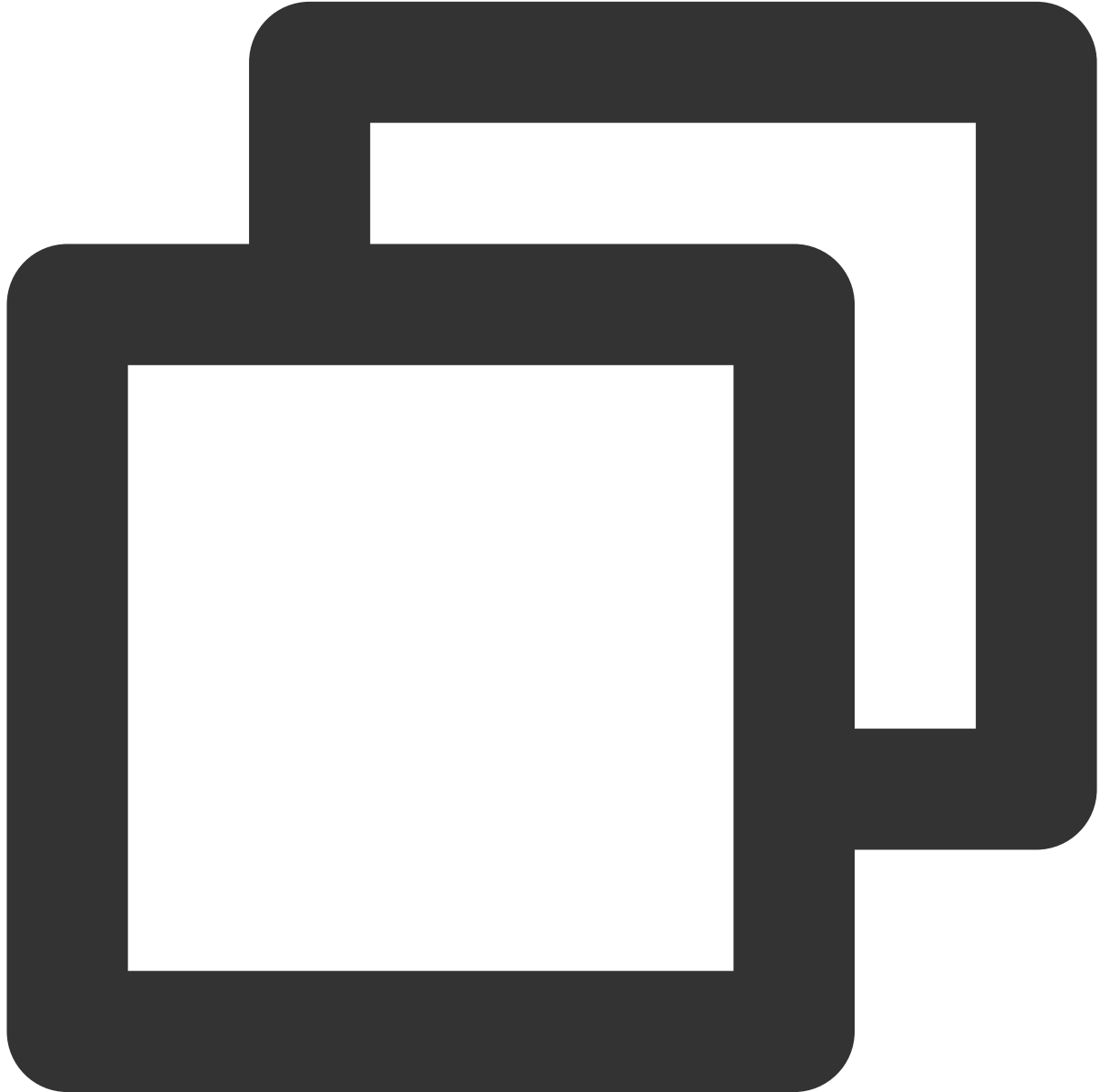
```
void updateVideoQuality(TUIVideoQuality quality)
```

### Parameters:

Parameter	Type	Meaning
quality	<a href="#">TUIVideoQuality</a>	Video Quality

## updateVideoQualityEx

Set the encoding parameters of video encoder.

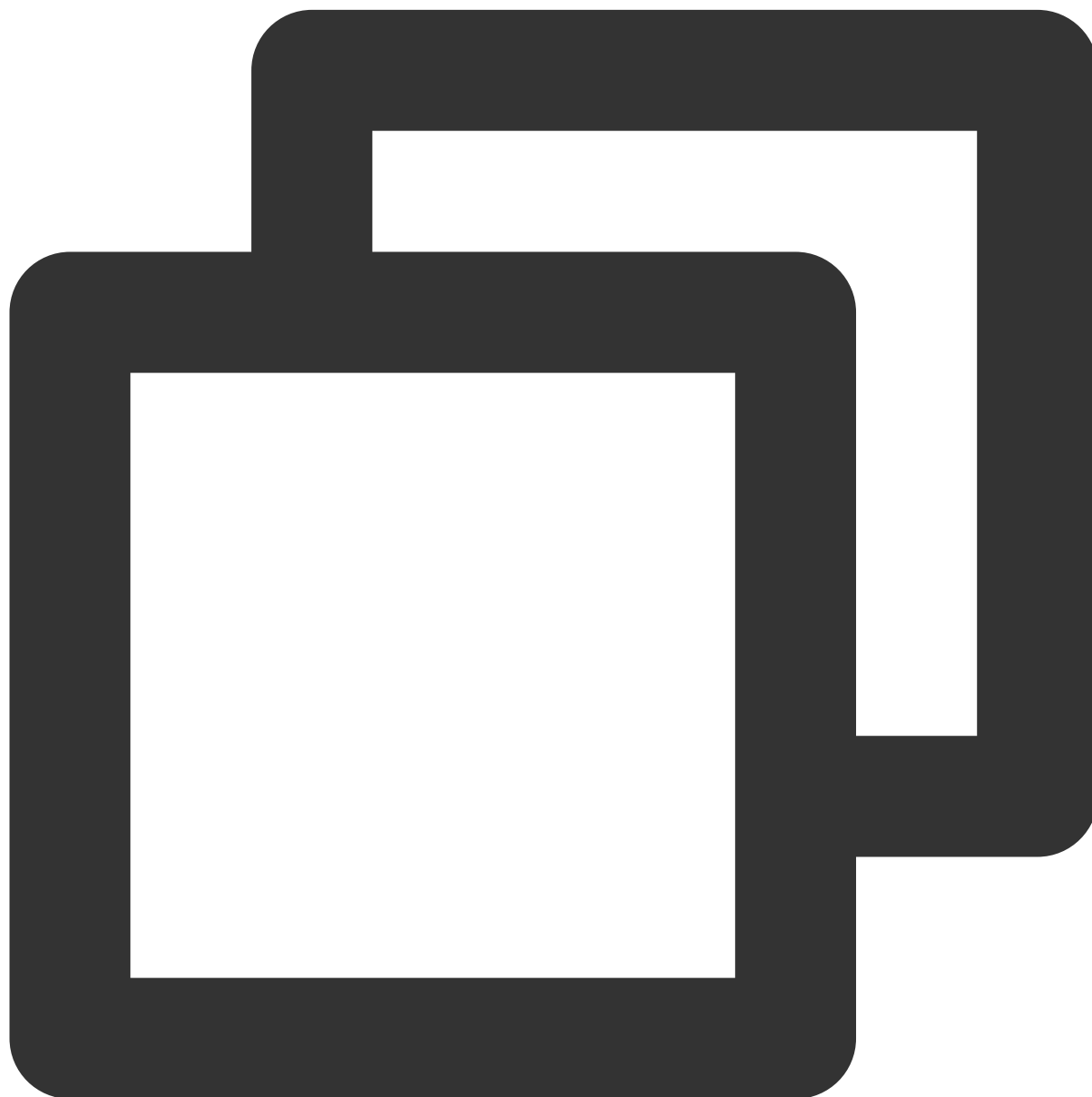


```
void updateVideoQualityEx(  
    TUIVideoStreamType streamType, TUIRoomVideoEncoderParams params);
```

Parameter	Type	Meaning
streamType	TUIVideoStreamType	Video Stream type
params	TUIRoomVideoEncoderParams	Encoding parameters of video encoder

## setVideoResolutionMode

Set the resolution mode of video encoder



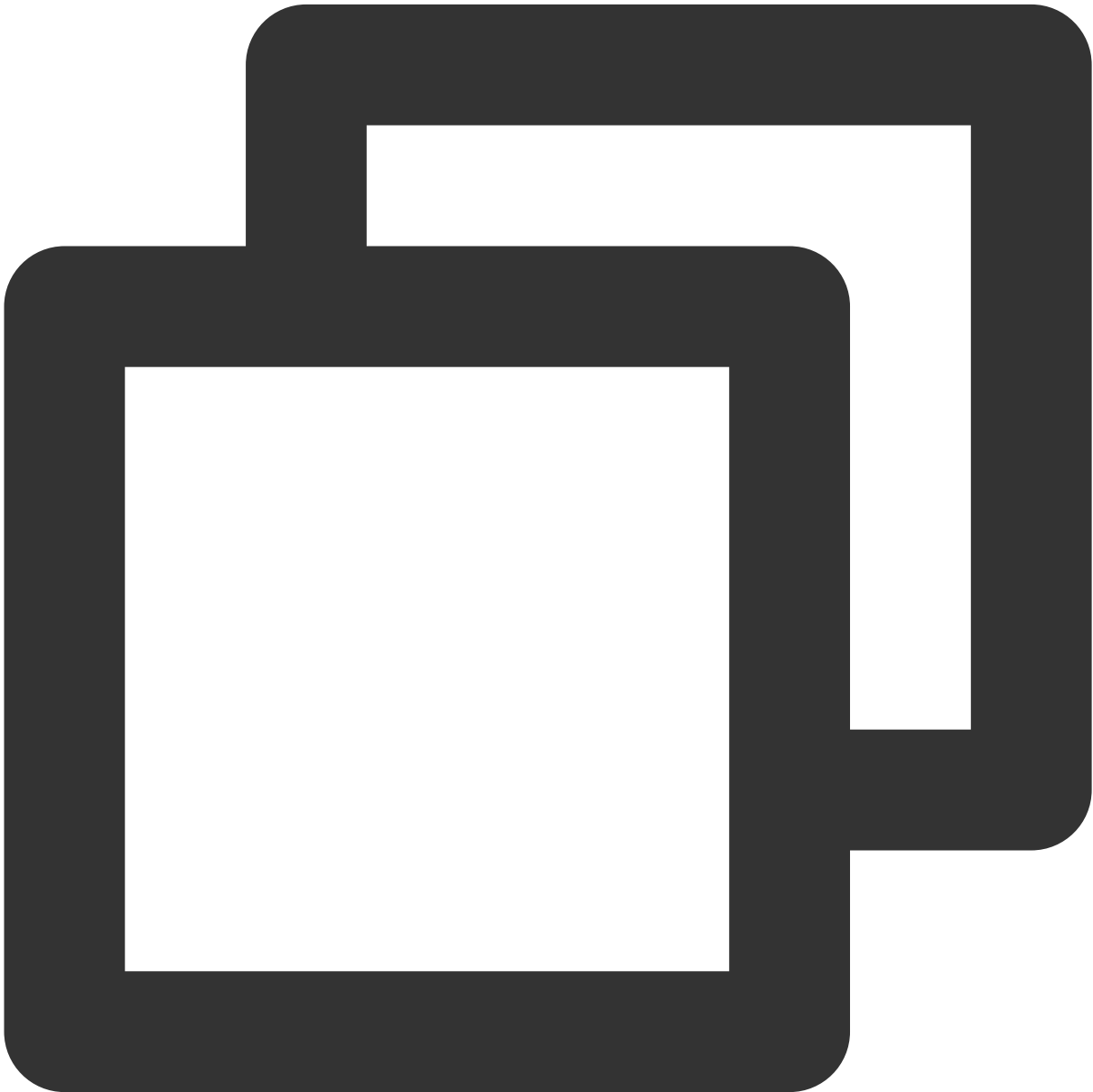
```
void setVideoResolutionMode(  
    TUIVideoStreamType streamType, TUIResolutionMode resolutionMode);
```

Parameter	Type	Meaning
streamType	TUIVideoStreamType	Video Stream type

resolutionMode	TUIResolutionMode	Video resolution mode
----------------	-------------------	-----------------------

enableGravitySensor

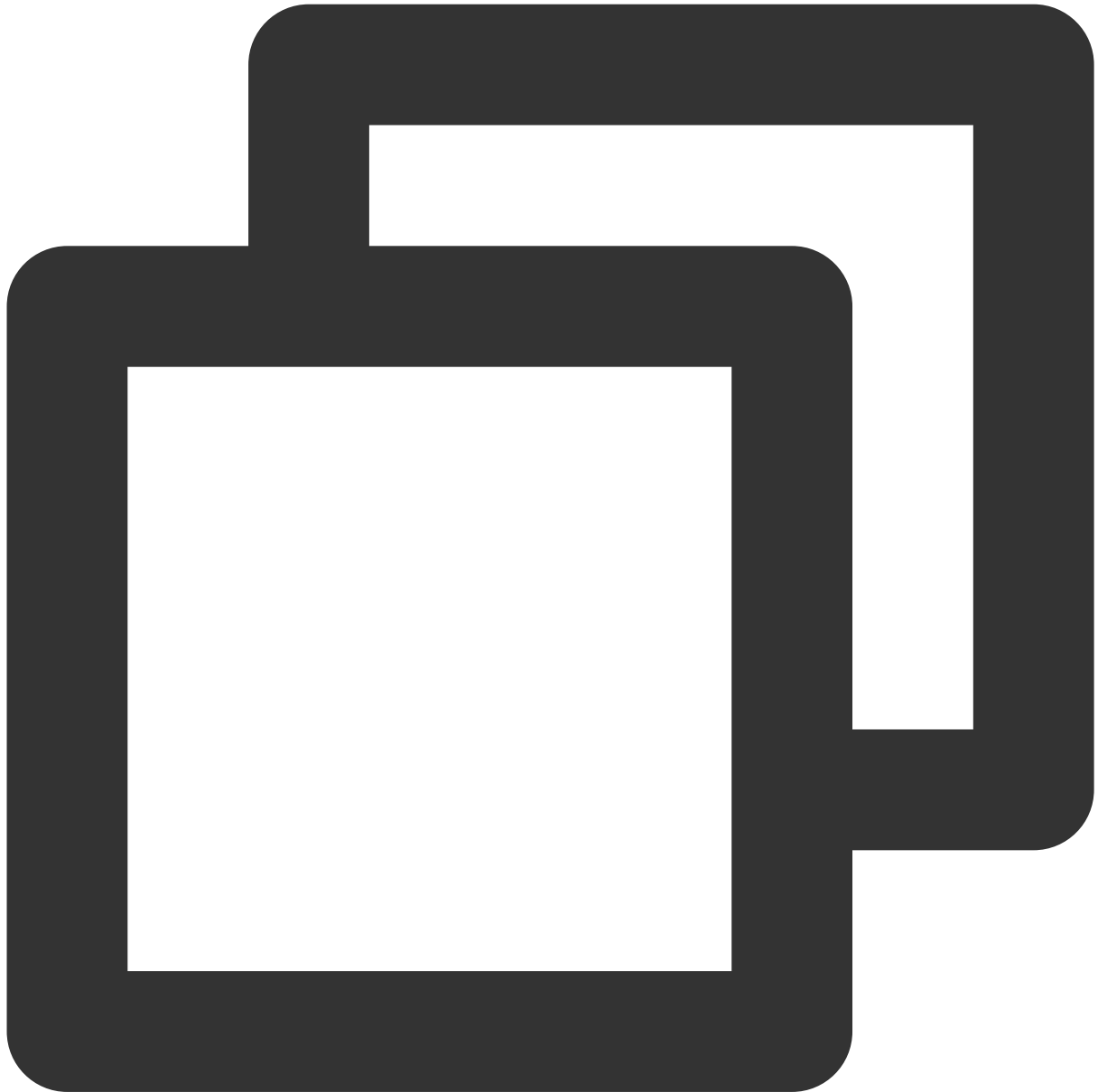
Enable the gravity sensor



<pre>void enableGravitySensor(bool enable);</pre>		
Parameter	Type	Meaning
enable	bool	Whether to enable

## **startPushLocalVideo**

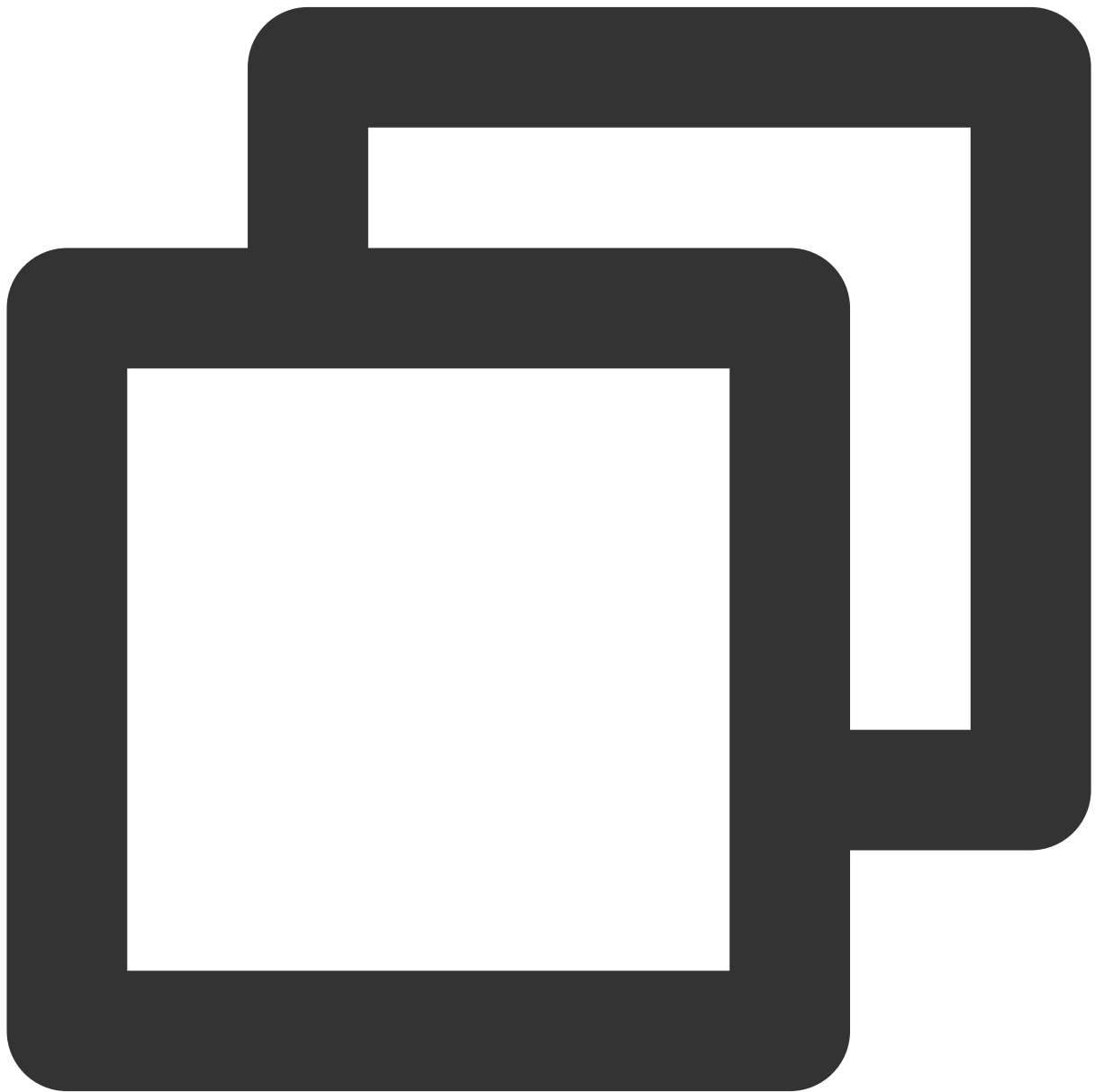
Start pushing Local Video streams to Remote.



```
void startPushLocalVideo()
```

## **stopPushLocalVideo**

Stop pushing Local Video streams to Remote.



```
void stopPushLocalVideo()
```

## startScreenSharing

Start screen sharing

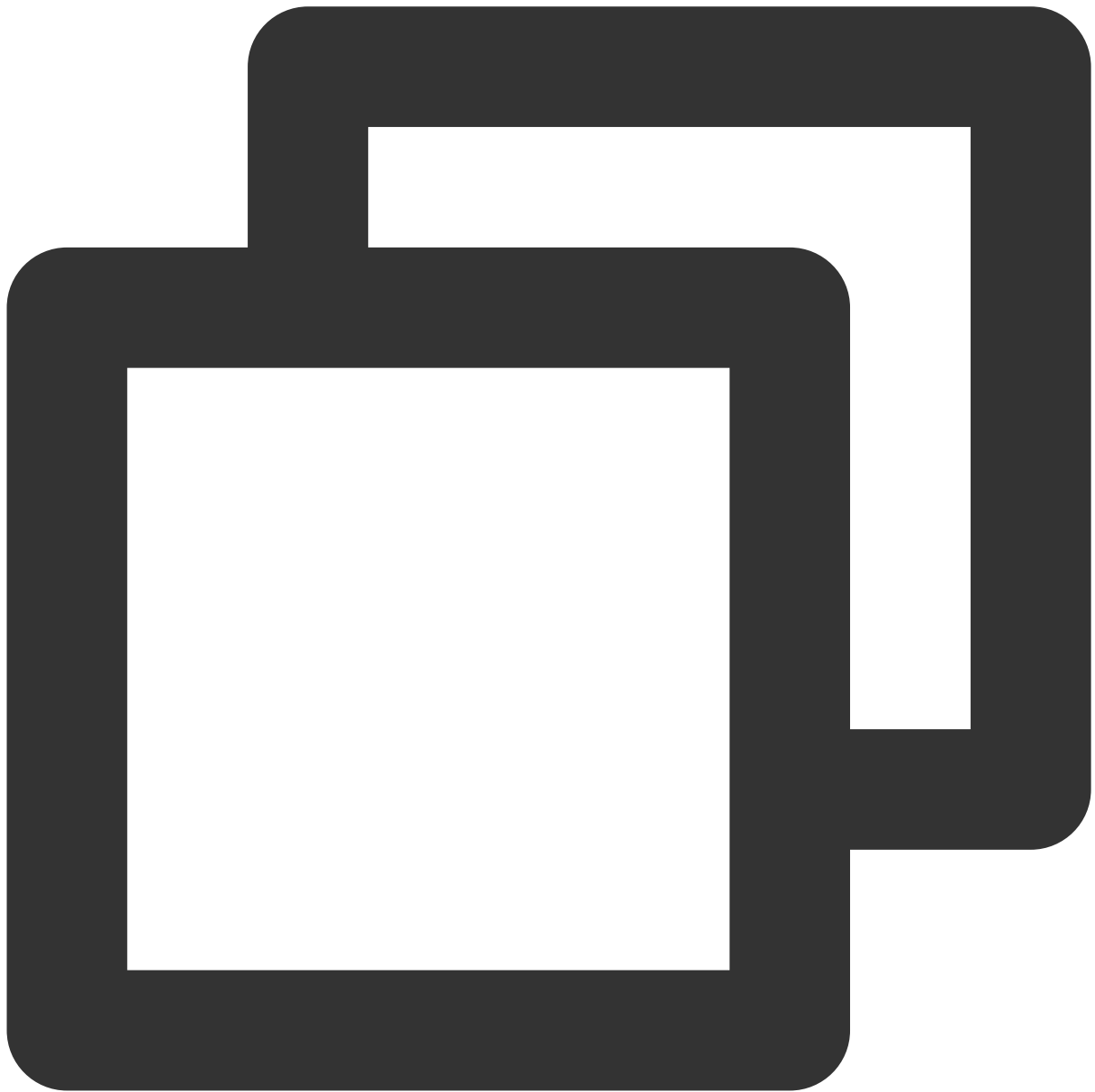




```
Future<void> startScreenSharing({String appGroup = ''})
```

## **stopScreenSharing**

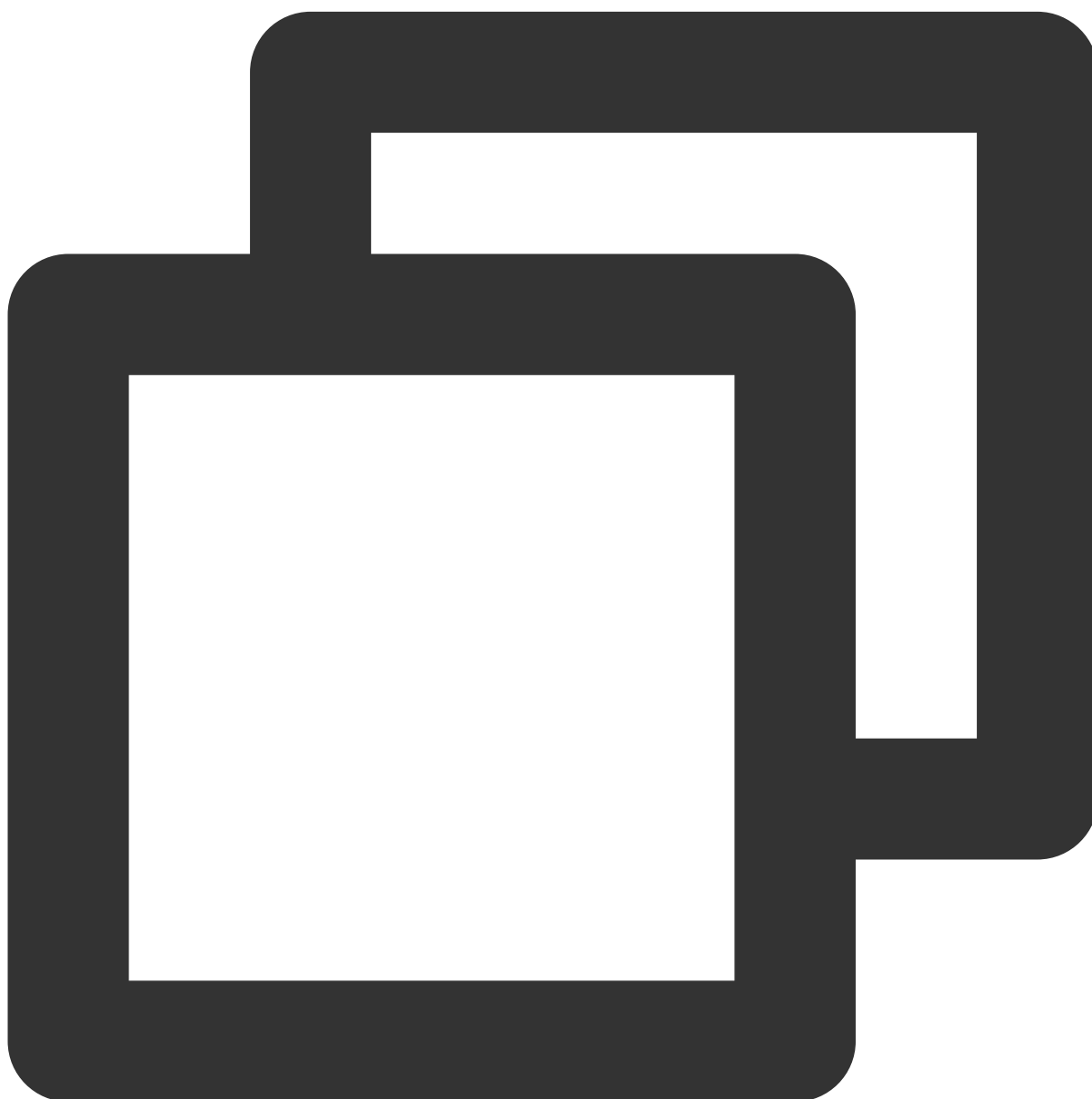
Stop screen sharing



```
Future<void> stopScreenSharing()
```

## openLocalMicrophone

Open Local mic.



```
Future<TUIActionCallback> openLocalMicrophone(TUIAudioQuality quality)
```

Parameter	Type	Meaning
quality	<a href="#">TUIAudioQuality</a>	Audio Quality

## closeLocalMicrophone

Close Local mic.



```
void closeLocalMicrophone()
```

### **updateAudioQuality**

Update Local Audio Codec Quality setting.



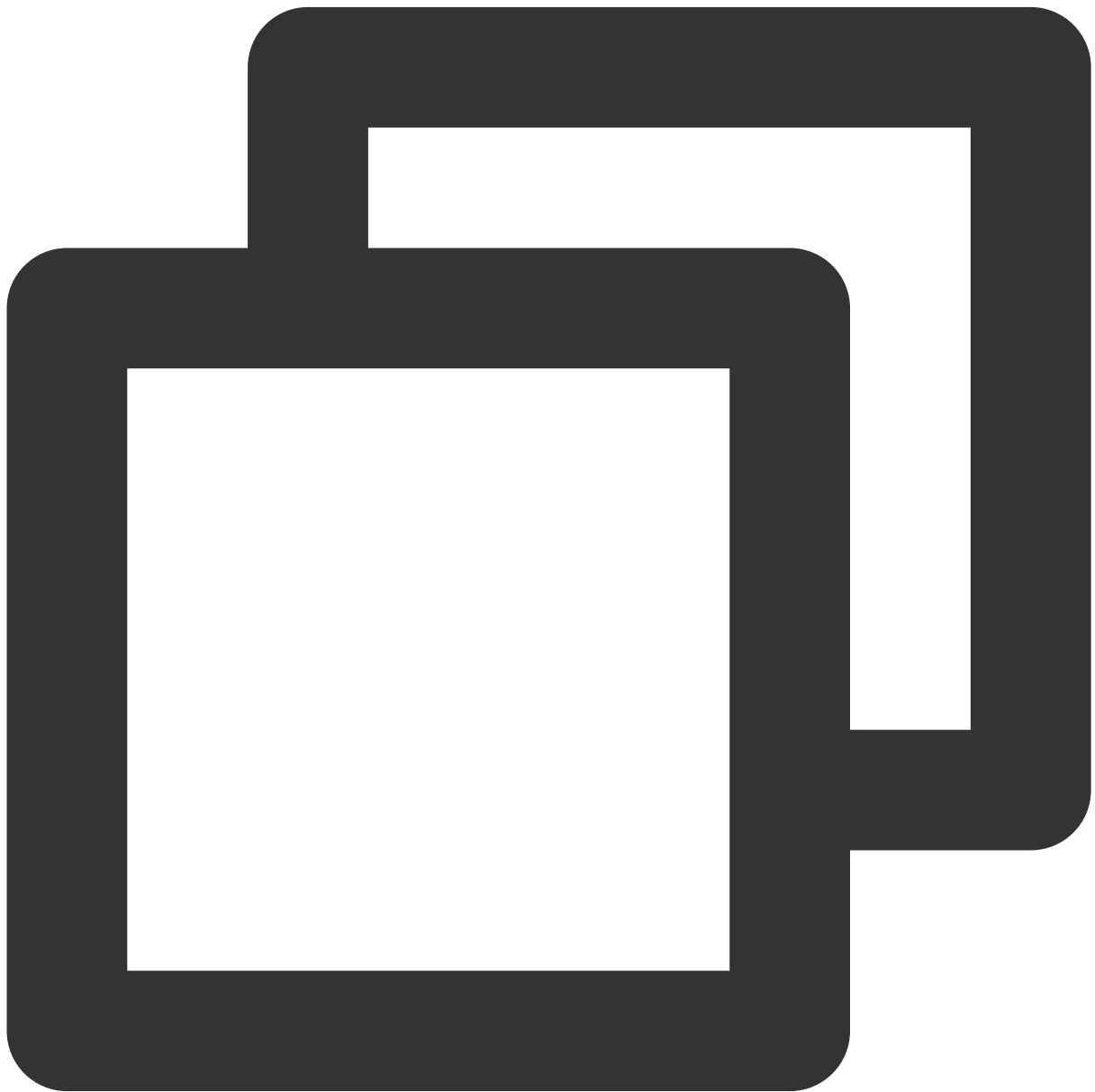
```
void updateAudioQuality(TUIAudioQuality quality)
```

**Parameters:**

Parameter	Type	Meaning
quality	<a href="#">TUIAudioQuality</a>	Audio Quality

**muteLocalAudio**

Mute local audio



```
Future<TUIActionCallback> muteLocalAudio()
```

## **unMuteLocalAudio**

UnMute local audio



```
Future<TUIActionCallback> unMuteLocalAudio()
```

### **setRemoteVideoView**

Set Remote user Video Rendering View control.



```
void setRemoteVideoView(String userId,
                        TUIVideoStreamType streamType,
                        int viewId)
```

**Parameters:**

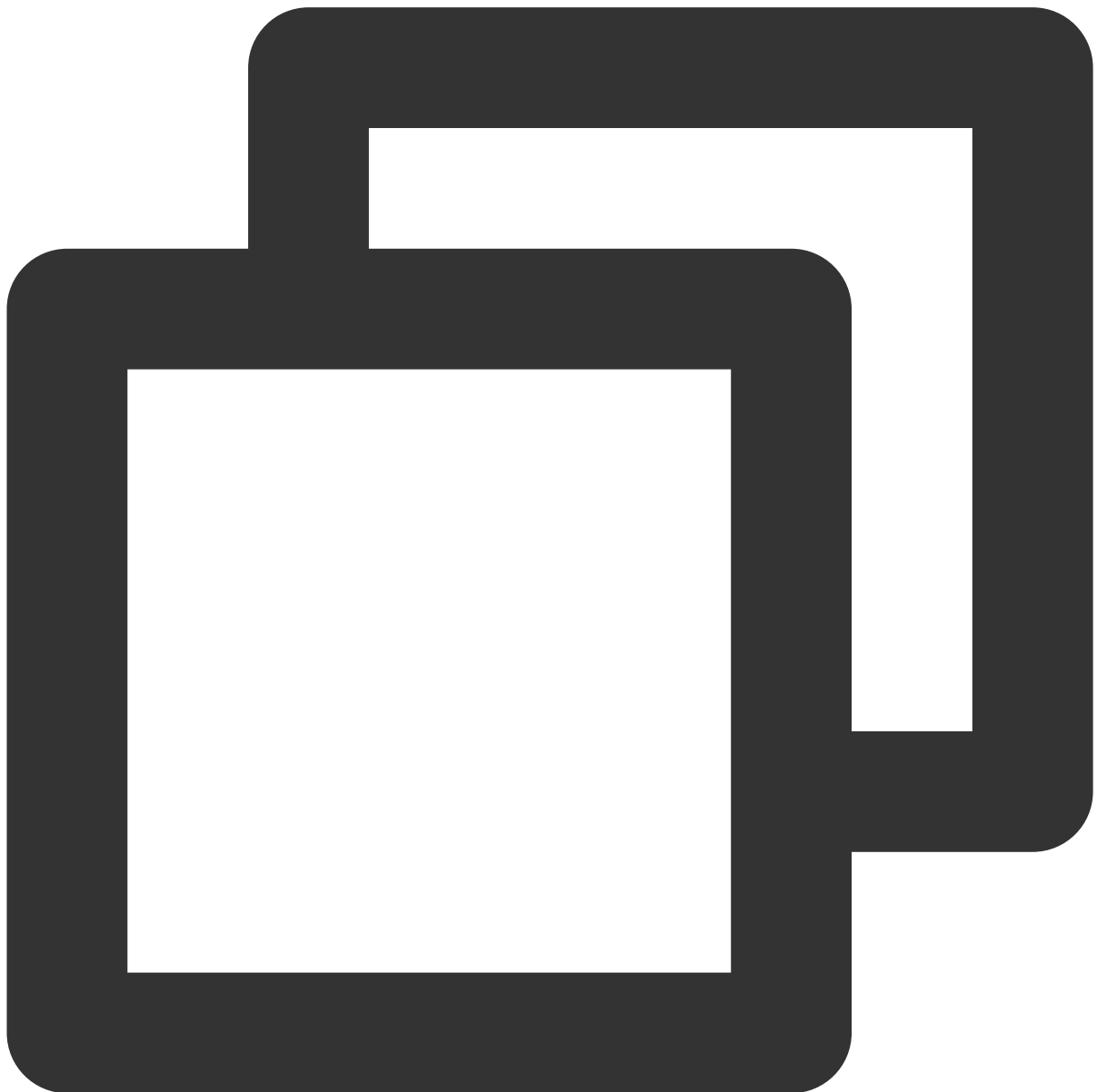
Parameter	Type	Meaning
userId	String	User ID
streamType	<a href="#">TUIVideoStreamType</a>	User streams type



viewId	int	The int64 type value of the pointer to the view to be rendered, through this viewId, can be converted to the corresponding native platform view, and the video screen will be rendered on this view.
--------	-----	--

## startPlayRemoteVideo

Start Playback Remote user Video streams.



```
void startPlayRemoteVideo(String userId,  
                           TUIVideoStreamType streamType,
```

```
TUIPlayCallback? callback)
```

**Parameters:**

Parameter	Type	Meaning
userId	String	User ID
streamType	<a href="#">TUIVideoStreamType</a>	User streams type
callback	TUIPlayCallback?	Playback Operation result Callback

**stopPlayRemoteVideo**

Stop Playback Remote user Video streams.



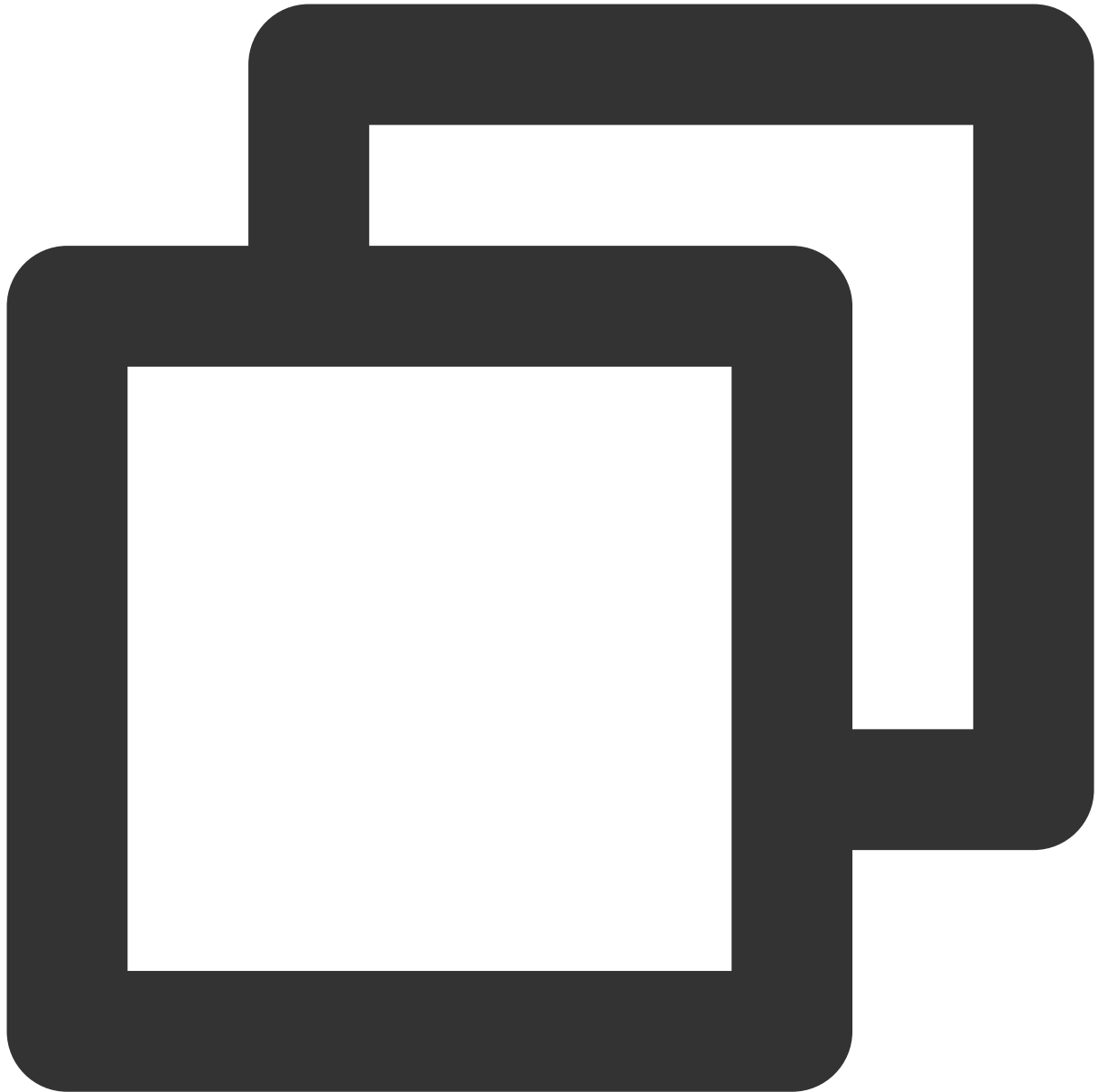
```
void stopPlayRemoteVideo(String userId,  
                          TUIVideoStreamType streamType)
```

**Parameters:**

Parameter	Type	Meaning
userId	String	User ID
streamType	<a href="#">TUIVideoStreamType</a>	User streams type

## **muteRemoteAudioStream**

Mute Remote user.



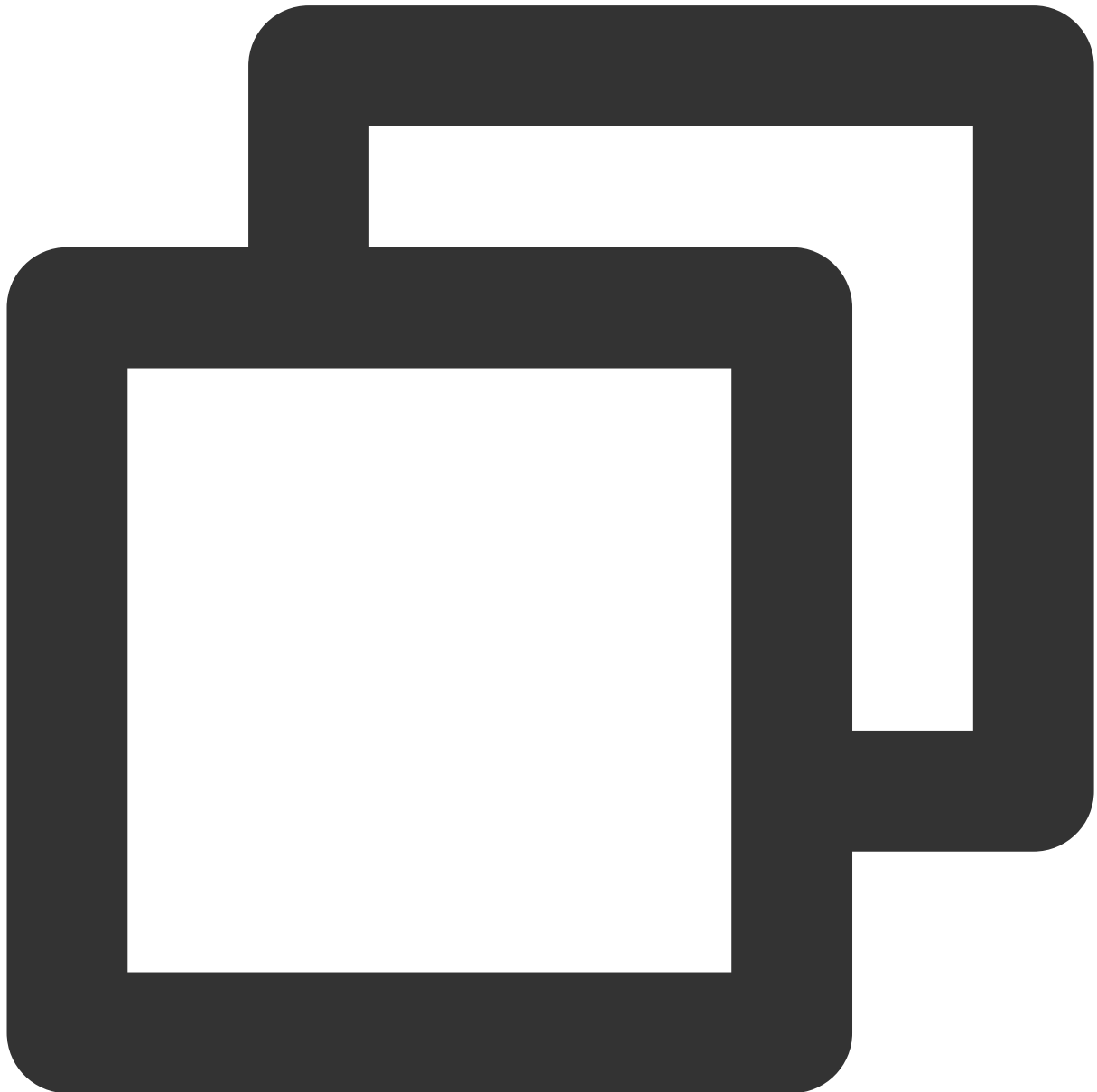
```
void muteRemoteAudioStream(String userId, boolean isMute);
```

### **Parameters:**

Parameter	Type	Meaning
userId	String	User ID
isMute	bool	Whether to mute

## getUserList

Get current User list in the room, Note that the maximum number of User list fetched by this Interface is 100.



```
Future<TUIValueCallBack<TUIUserListResult>> getUserList(int nextSequence)
```

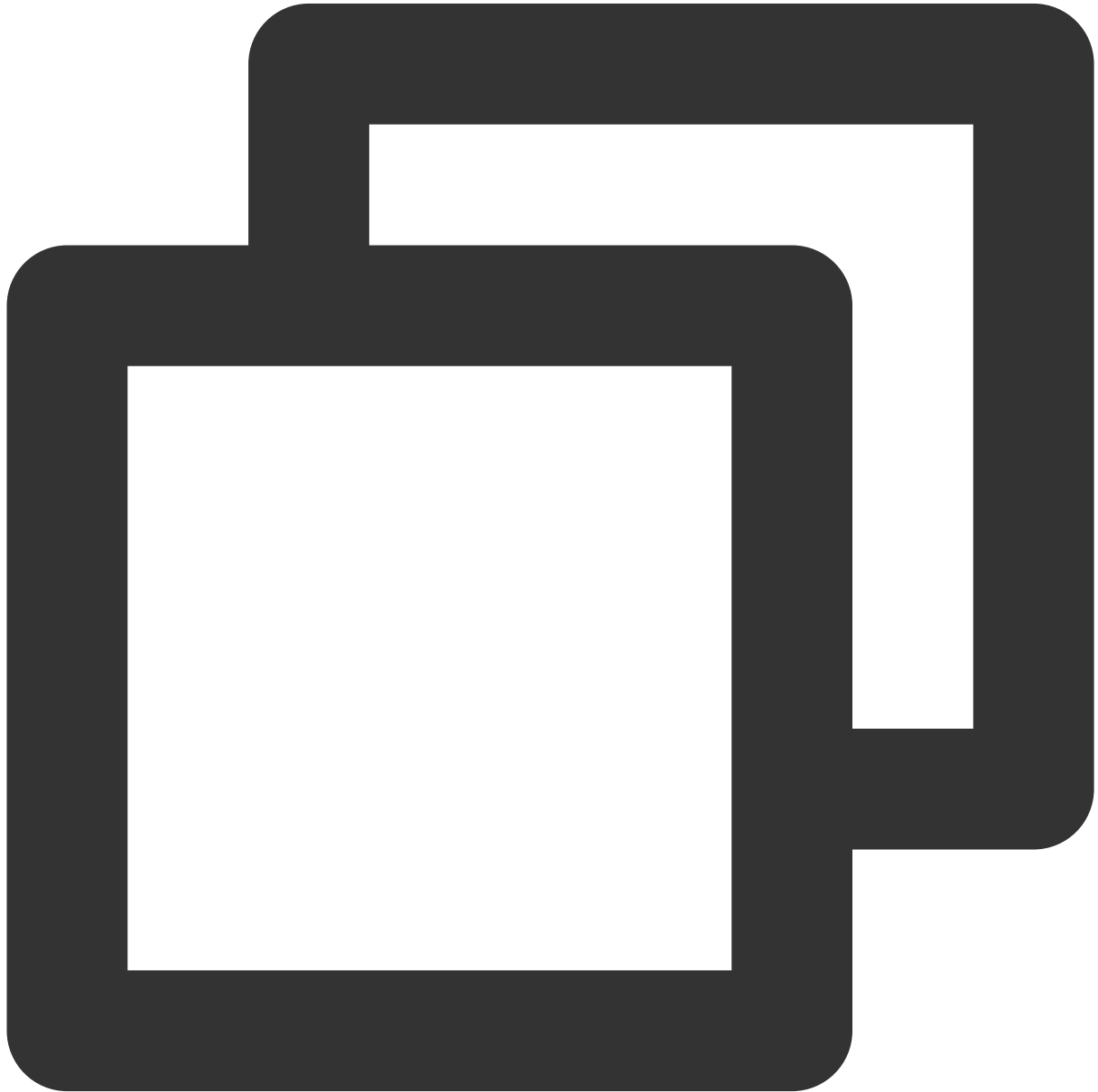
### Parameters:

Parameter	Type	Meaning
nextSequence	int	Pagination Fetch Flag, fill in 0 for the first Fetch, if the

nextSeq in the Callback is not 0, you need to do  
Pagination, pass in the nextSeq to Fetch again until the  
nextSeq in the Callback is 0

## getUserInfo

Get User [Learn more](#).



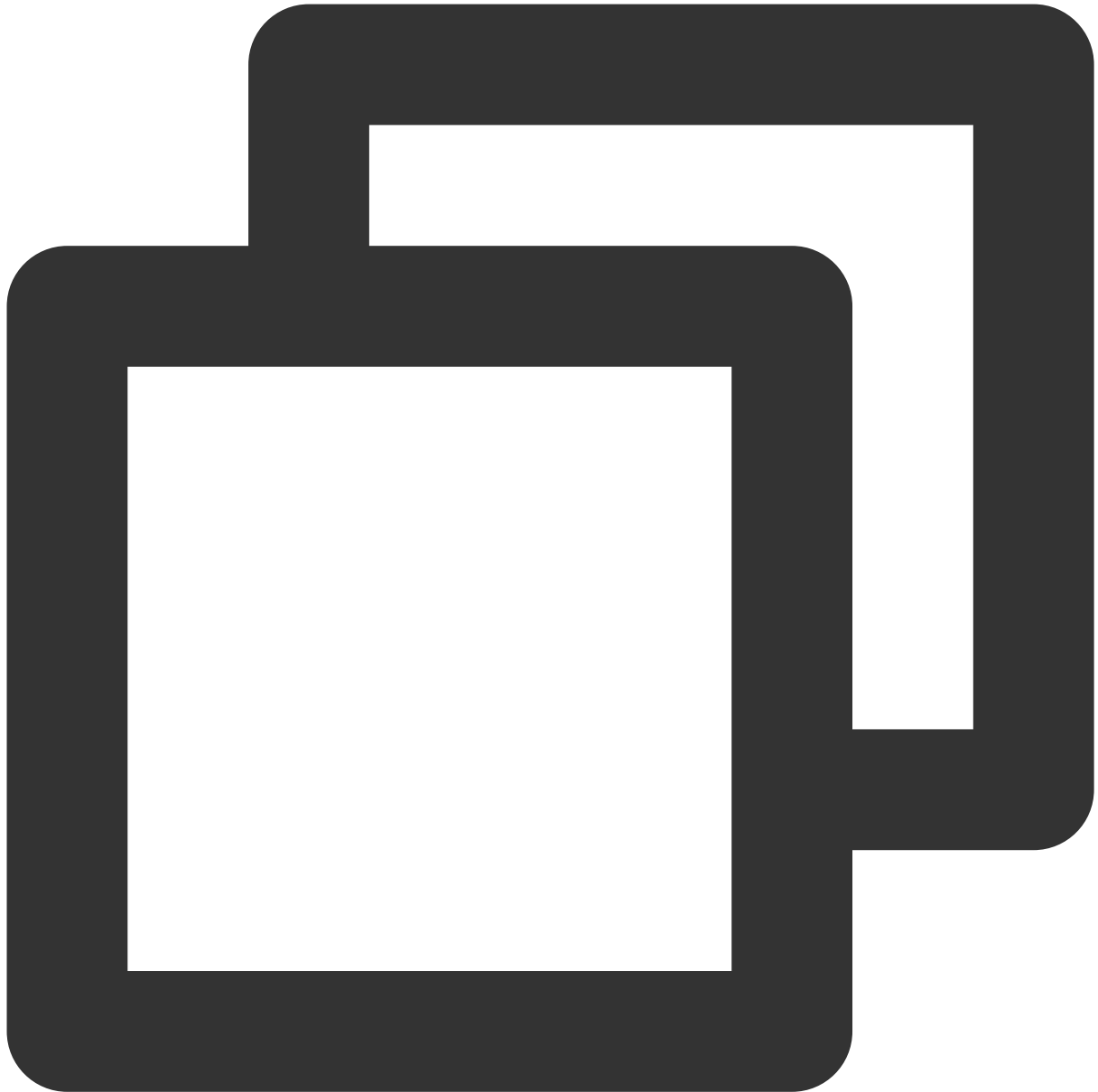
```
Future<TUIValueCallBack<TUIUserInfo>> getUserInfo(String userId)
```

### Parameters:

Parameter	Type	Meaning
userId	String	Get Learn more of the user by userId

## changeUserRole

Change user Role, only Administrator or Group owner can call.



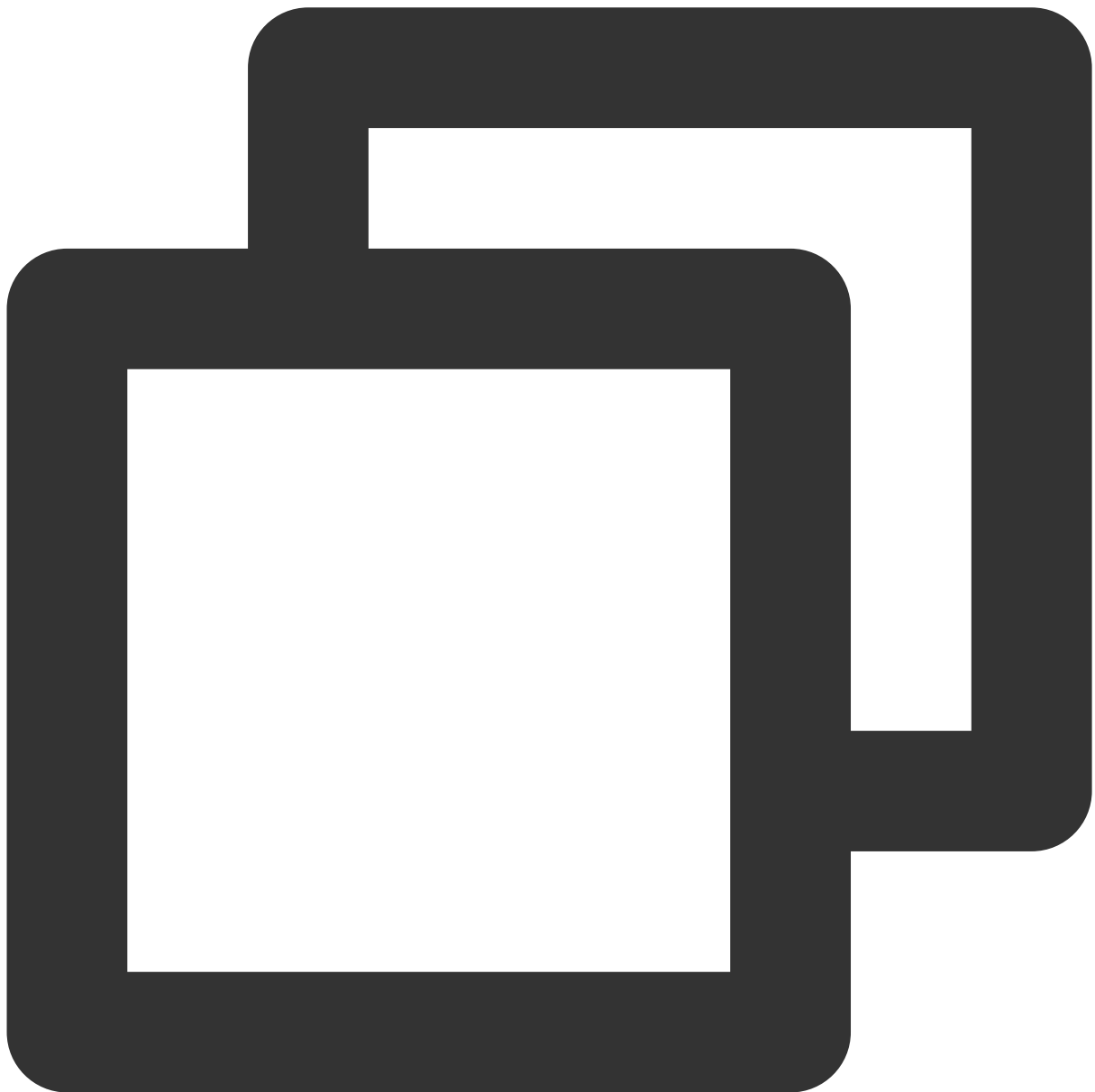
```
Future<TUIActionCallback> changeUserRole(String userId,  
                                         TUIRole role)
```

**Parameters:**

Parameter	Type	Meaning
userId	String	User ID
role	<a href="#">TUIRole</a>	User Role

**kickRemoteUserOutOfRoom**

Kick user out of the room, only Administrator or Group owner can call.





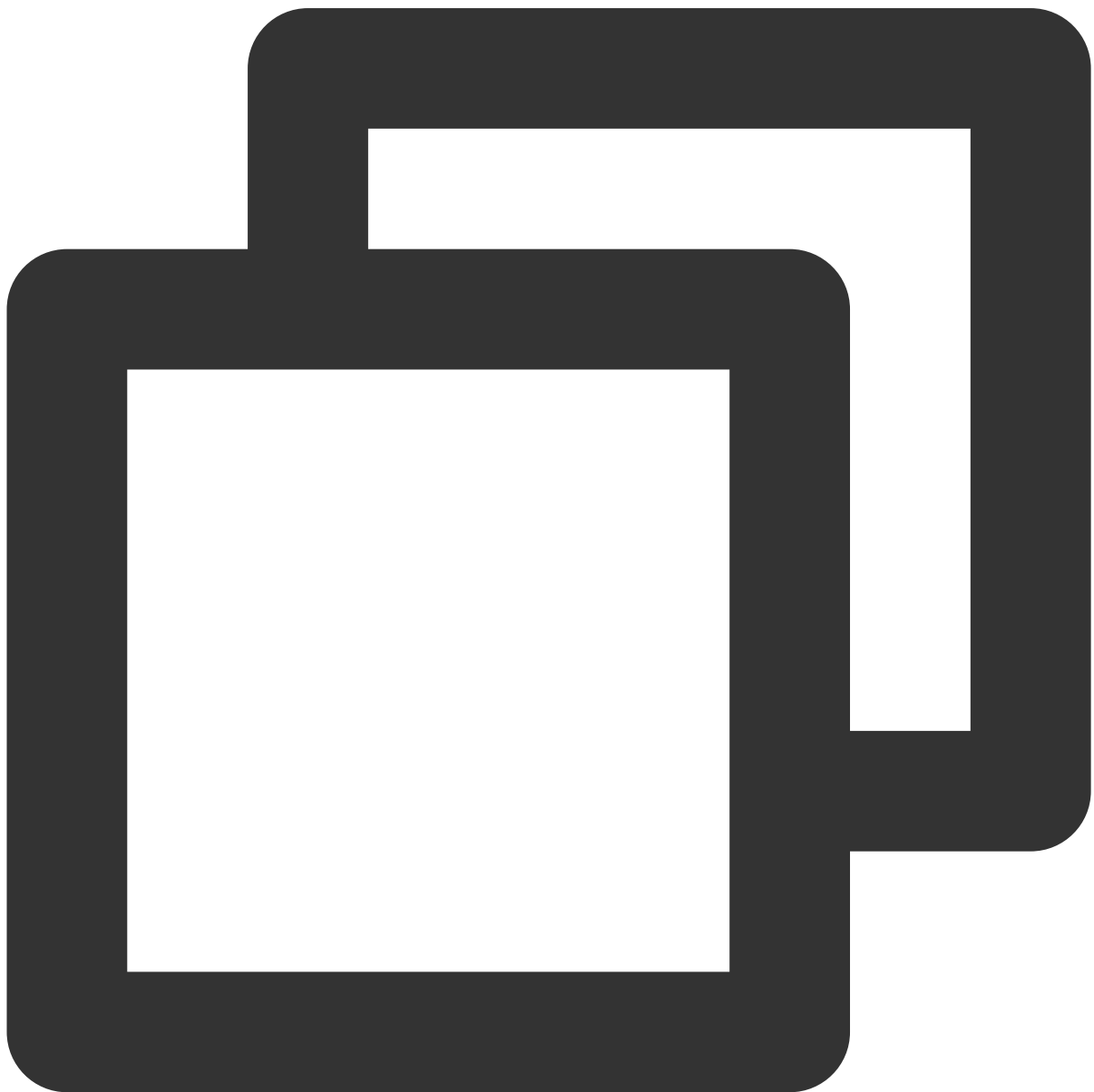
```
Future<TUIActionCallback> kickRemoteUserOutOfRoom(String userId)
```

**Parameters:**

Parameter	Type	Meaning
userId	String	User ID

**addCategoryTagForUsers**

Add category tags to users (Only Administrator or Group Owner can call)



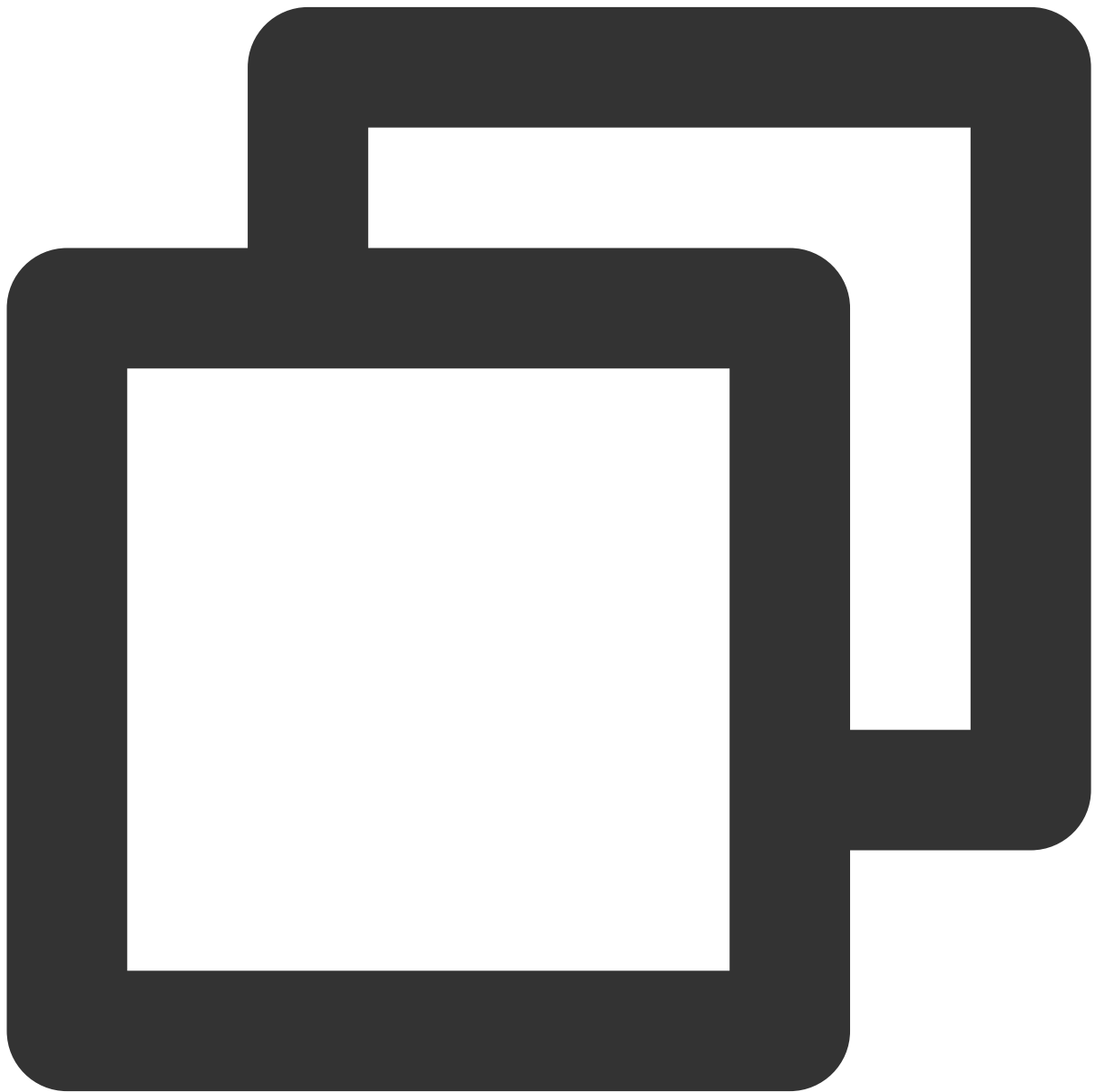
```
Future<TUIActionCallback> addCategoryTagForUsers(int tag, List<String> userList);
```

**Parameters :**

Parameter	Type	Meaning
tag	int	Tag type. Number type, greater than or equal to 1000, you can customize
userList	List<String>	User list

**removeCategoryTagForUsers**

Remove category tags to users (Only Administrator or Group Owner can call)



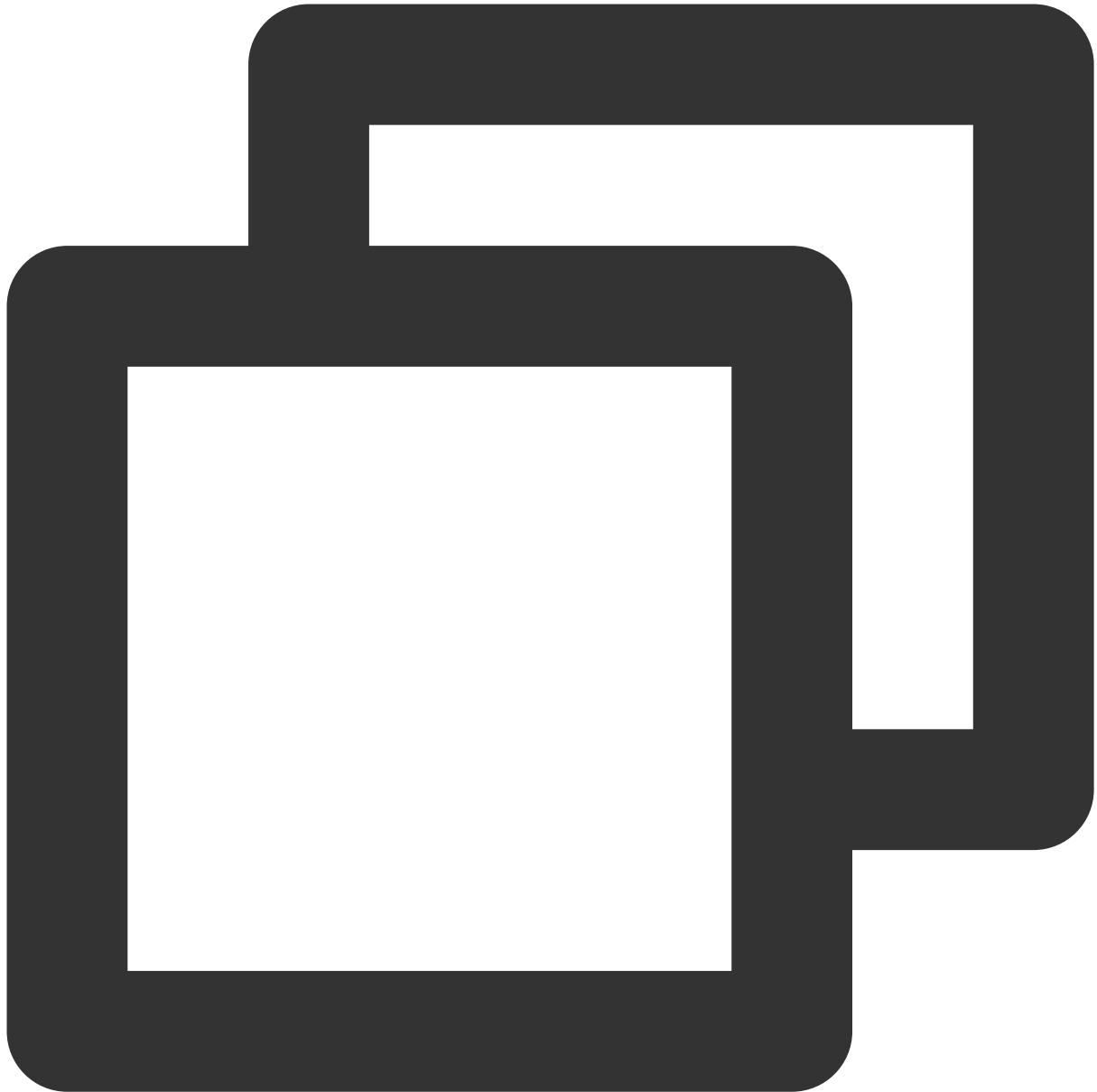
```
Future<TUIActionCallback> removeCategoryTagForUsers(int tag, List<String> userList)
```

**Parameters :**

Parameter	Type	Meaning
tag	int	Tag type. Number type, greater than or equal to 1000, you can customize
userList	List<String>	User list

## getUserListByTag

Get user information in the room based on tags



```
Future<TUIValueCallBack<TUIUserListResult>> getUserListByTag(int tag, int nextSeque
```

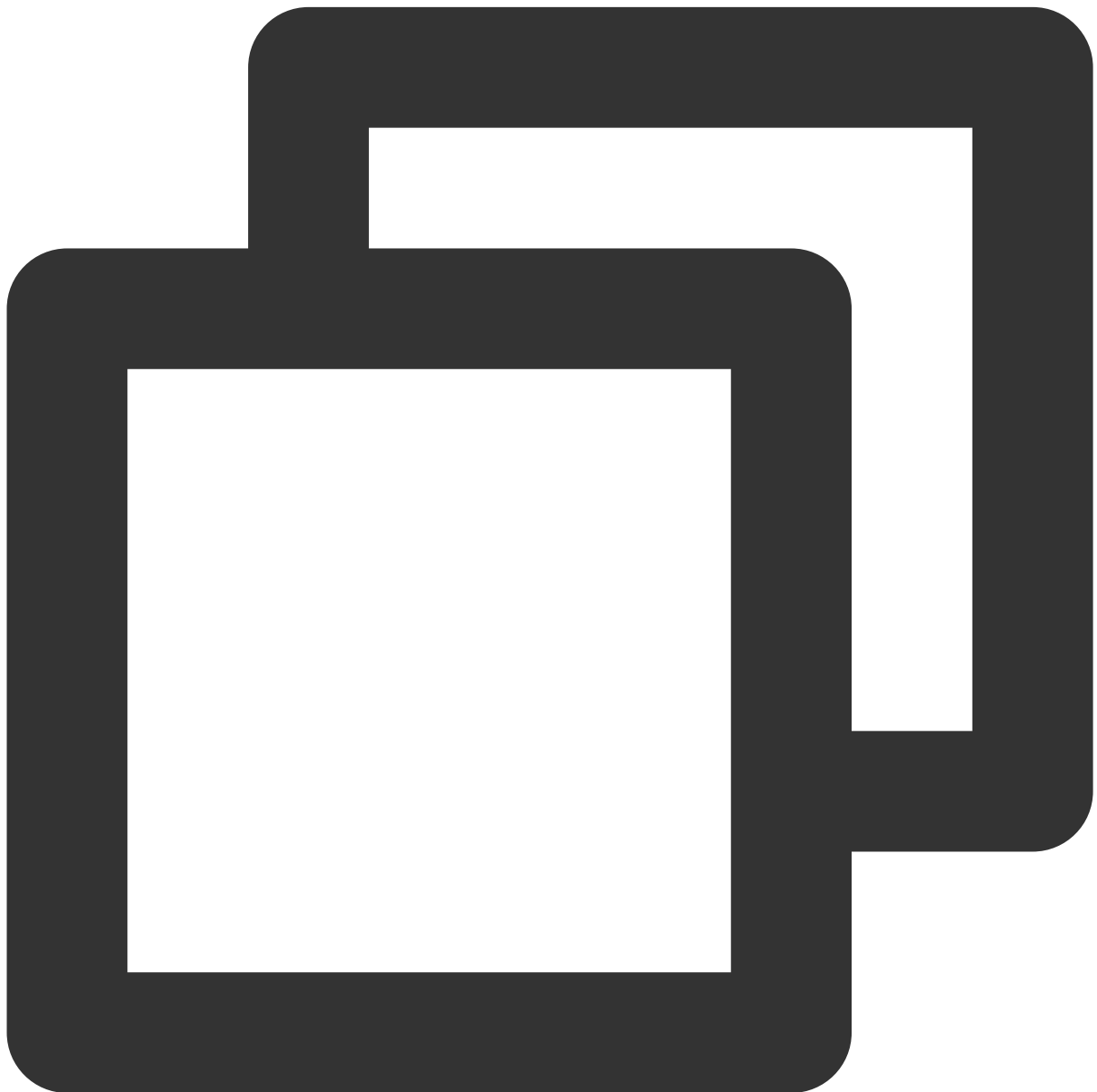
### Parameters :

Parameter	Type	Meaning
tag	int	Tag type. Number type, greater than or equal to 1000, you can customize

nextSequence	int	Pagination Fetch Flag, fill in 0 for the first Fetch, if the nextSeq in the Callback is not 0, you need to do Pagination, pass in the nextSeq to Fetch again until the nextSeq in the Callback is 0
--------------	-----	---

### **disableDeviceForAllUserByAdmin**

Manage Media device for all users, only Administrator or Group owner can call.

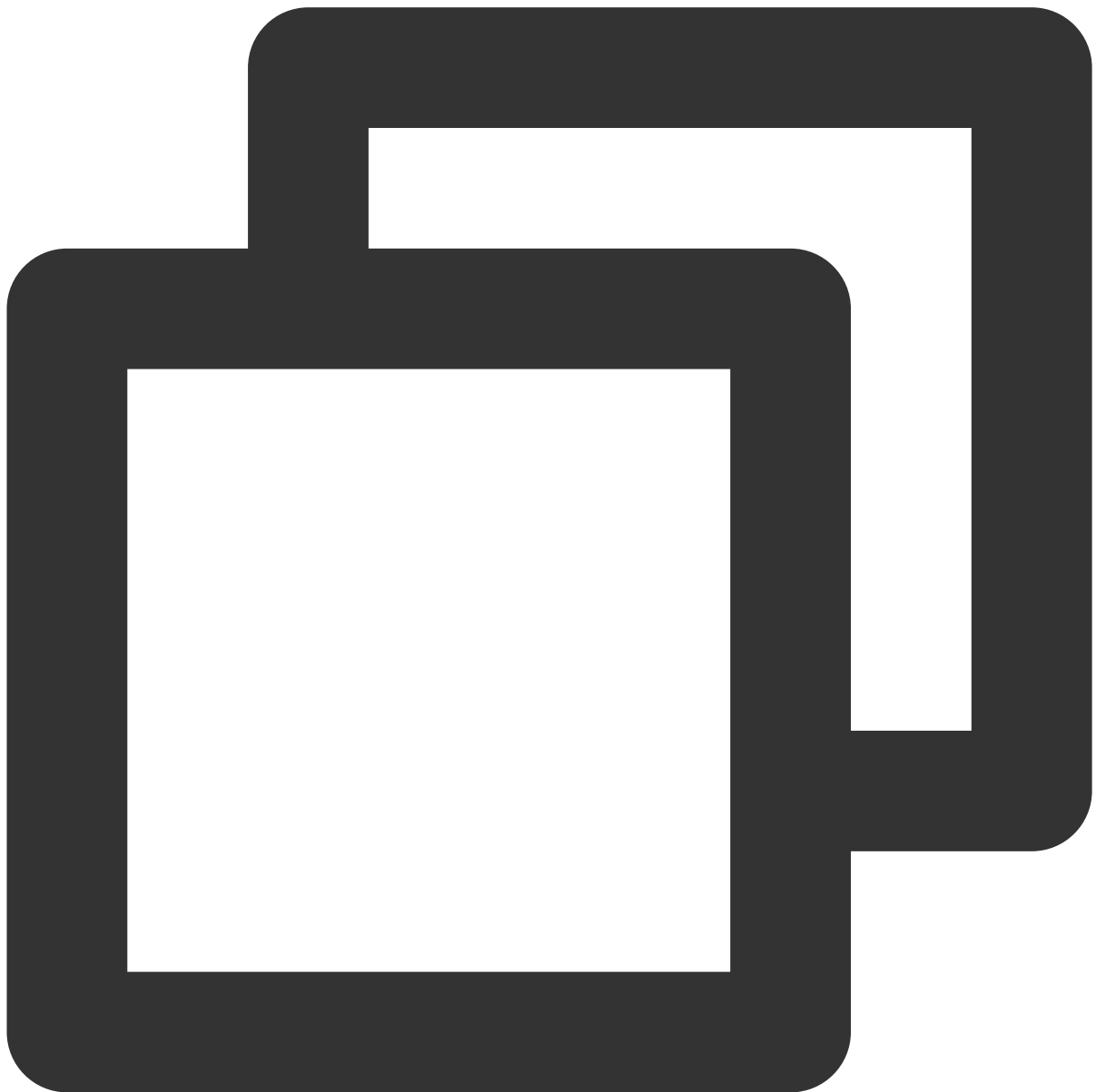


```
Future<TUIActionCallback> disableDeviceForAllUserByAdmin(TUIMediaDevice device,  
                                                         bool isDisable)
```

Parameter	Type	Meaning
device	<a href="#">TUIMediaDevice</a>	Device
isDisable	bool	Whether to Disable

### openRemoteDeviceByAdmin

Request Remote user to open Media device, only Administrator or Group owner can call.



```
TUIRequest openRemoteDeviceByAdmin(String userId,
```

```
TUIMediaDevice device,  
int timeout,  
TUIRequestCallback? requestCallback)
```

Parameter	Type	Meaning
userId	String	User ID
device	<a href="#">TUIMediaDevice</a>	Device
timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout detection and will not trigger timeout Callback
requestCallback	TUIRequestCallback?	Operation result Callback

### closeRemoteDeviceByAdmin

Close Remote user Media device, only Administrator or Group owner can call.



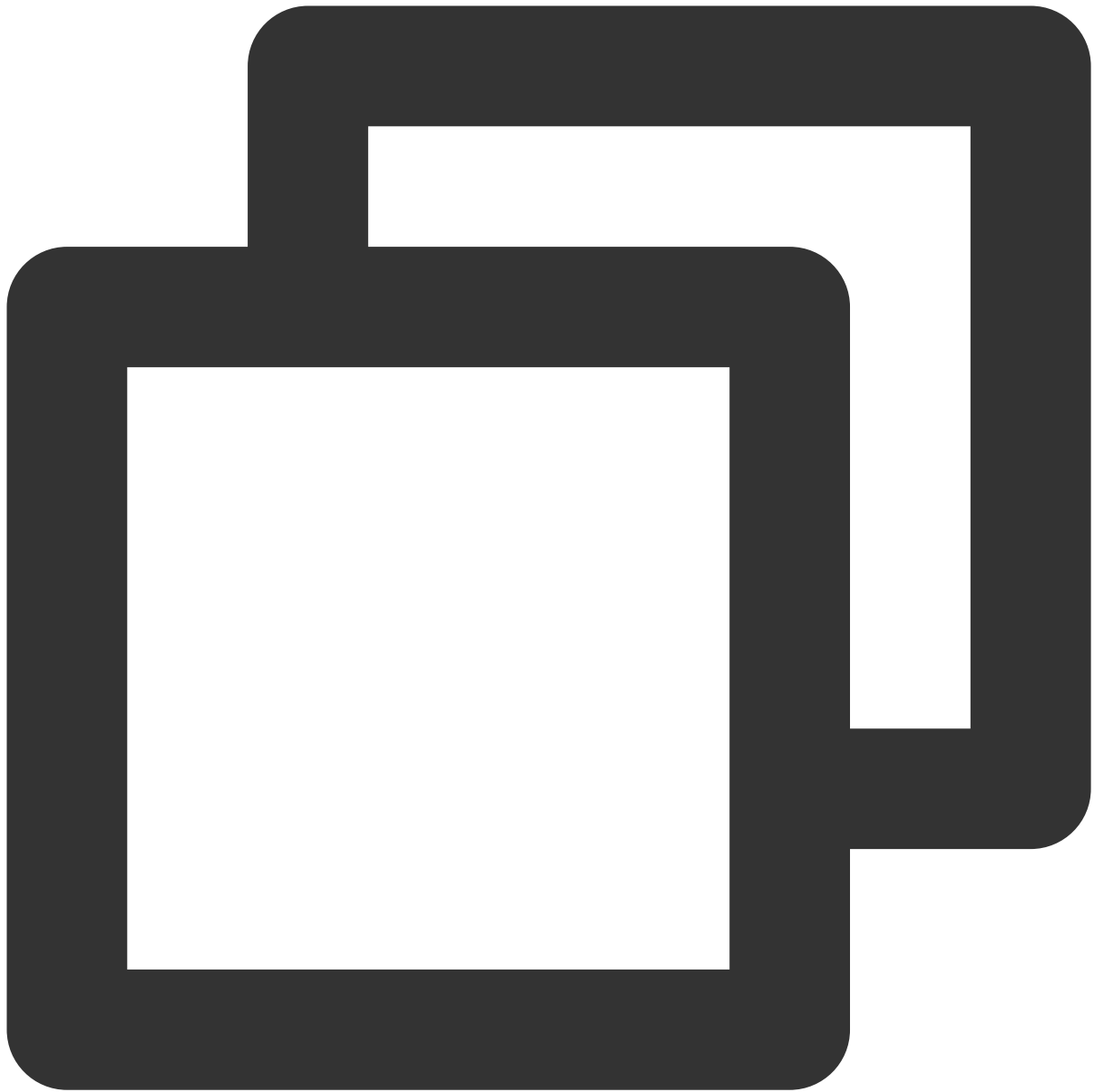
```
Future<TUIActionCallback> closeRemoteDeviceByAdmin(String userId,  
                                                    TUIMediaDevice device)
```

Parameter	Type	Meaning
userId	String	User ID
device	<a href="#">TUIMediaDevice</a>	Device

### applyToAdminToOpenLocalDevice



Lock all users' Media device management.



```
TUIRequest applyToAdminToOpenLocalDevice(TUIMediaDevice device,  
                                           int timeout,  
                                           TUIRequestCallback? requestCallback)
```

Parameter	Type	Meaning
device	<a href="#">TUIMediaDevice</a>	Device
timeout	int	Timeout Duration, Unit in Seconds. If Set to 0, SDK Will Not

		Perform Timeout Detection, Nor Will It Trigger Timeout Callback
callback	TUIRequestCallback	Operation Result Callback

## setMaxSeatCount

Set Maximum number of seats, only supported when entering the room and creating the room

When roomType is RoomType.CONFERENCE (Education and Conference scene), maxSeatCount value is not limited;

When roomType is RoomType.LIVE\_ROOM (Live broadcast scene), maxSeatCount is limited to 16;



```
Future<TUIActionCallback> setMaxSeatCount(int maxSeatCount)
```

Parameter	Type	Meaning
maxSeatCount	int	Maximum number of seats

### lockSeatByAdmin

Lock seat (including position lock, audio status lock, video status lock).

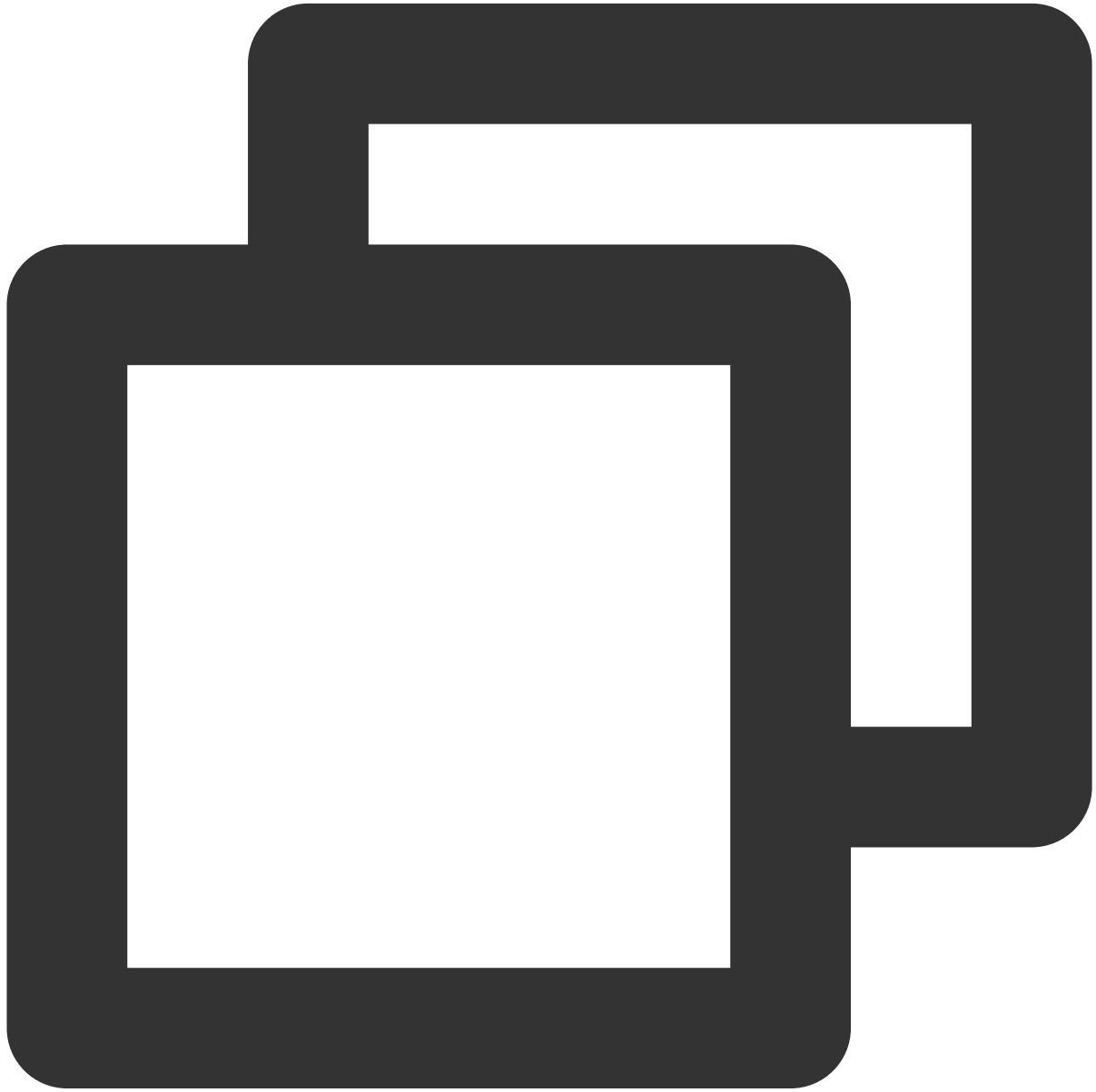


```
Future<TUIActionCallback> lockSeatByAdmin(int seatIndex,  
                                           TUISeatLockParams lockParams)
```

Parameter	Type	Meaning
seatIndex	int	Seat number
lockParams	<a href="#">TUISeatLockParams</a>	Lock microphone parameter

## getSeatList

Get seat list.



```
Future<TUIValueCallBack<List<TUISeatInfo>>> getSeatList ()
```

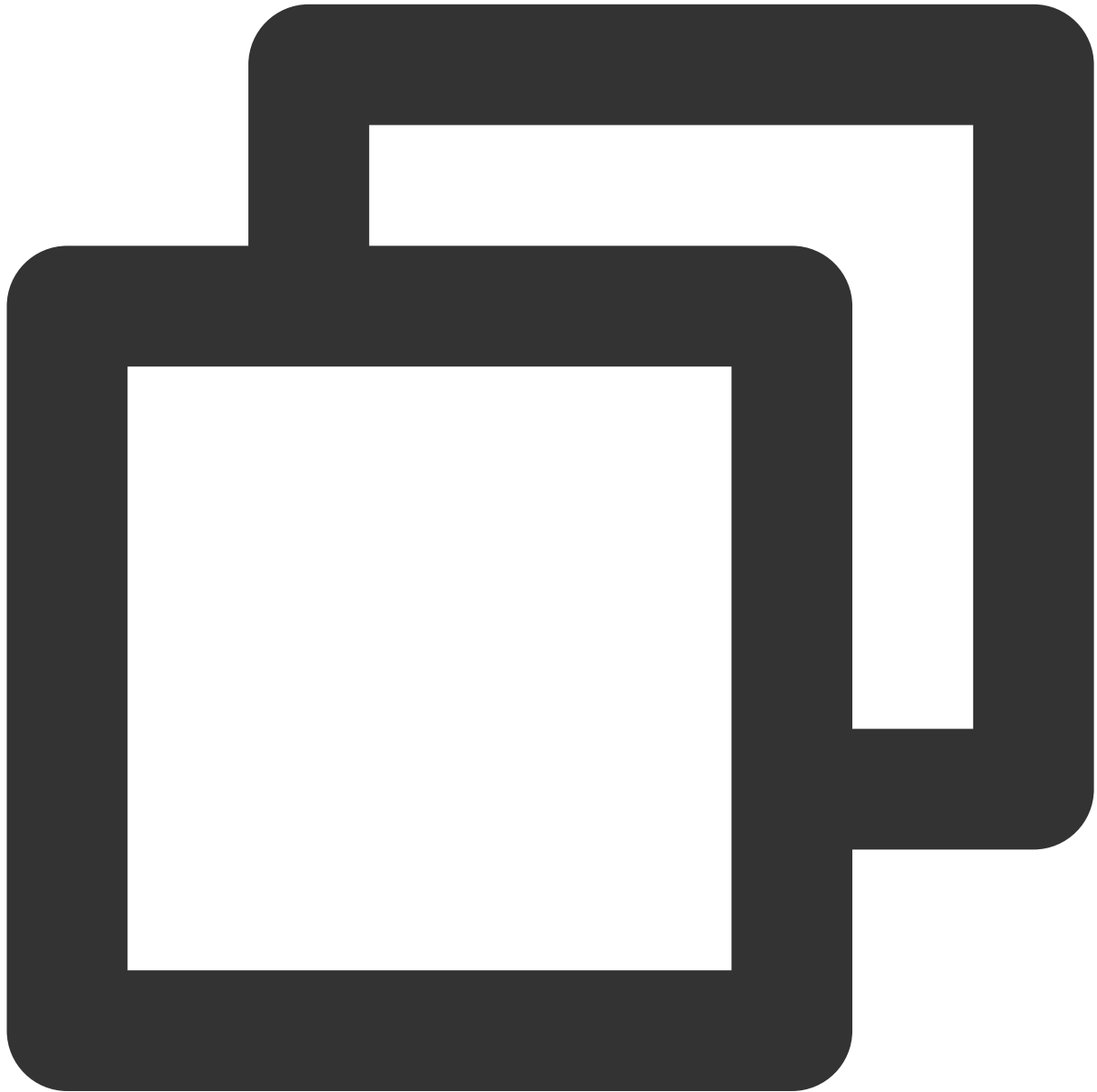
## takeSeat

Local on microphone.

### Explanation:

Conference Scene: [applyToSpeak](#) Mode needs to apply to the host or administrator to allow Go Live, other modes do not support Go Live.

Live Broadcast Scene: [freeToSpeak](#) Mode can freely Go Live, and speak after Go Live; [applySpeakAfterTakingSeat](#) Mode needs to apply to the host or administrator to allow Go Live; other modes do not support Go Live.



```
TUIRequest takeSeat(int seatIndex,  
                    int timeout,  
                    TUIRequestCallback? requestCallback)
```

**Parameters:**

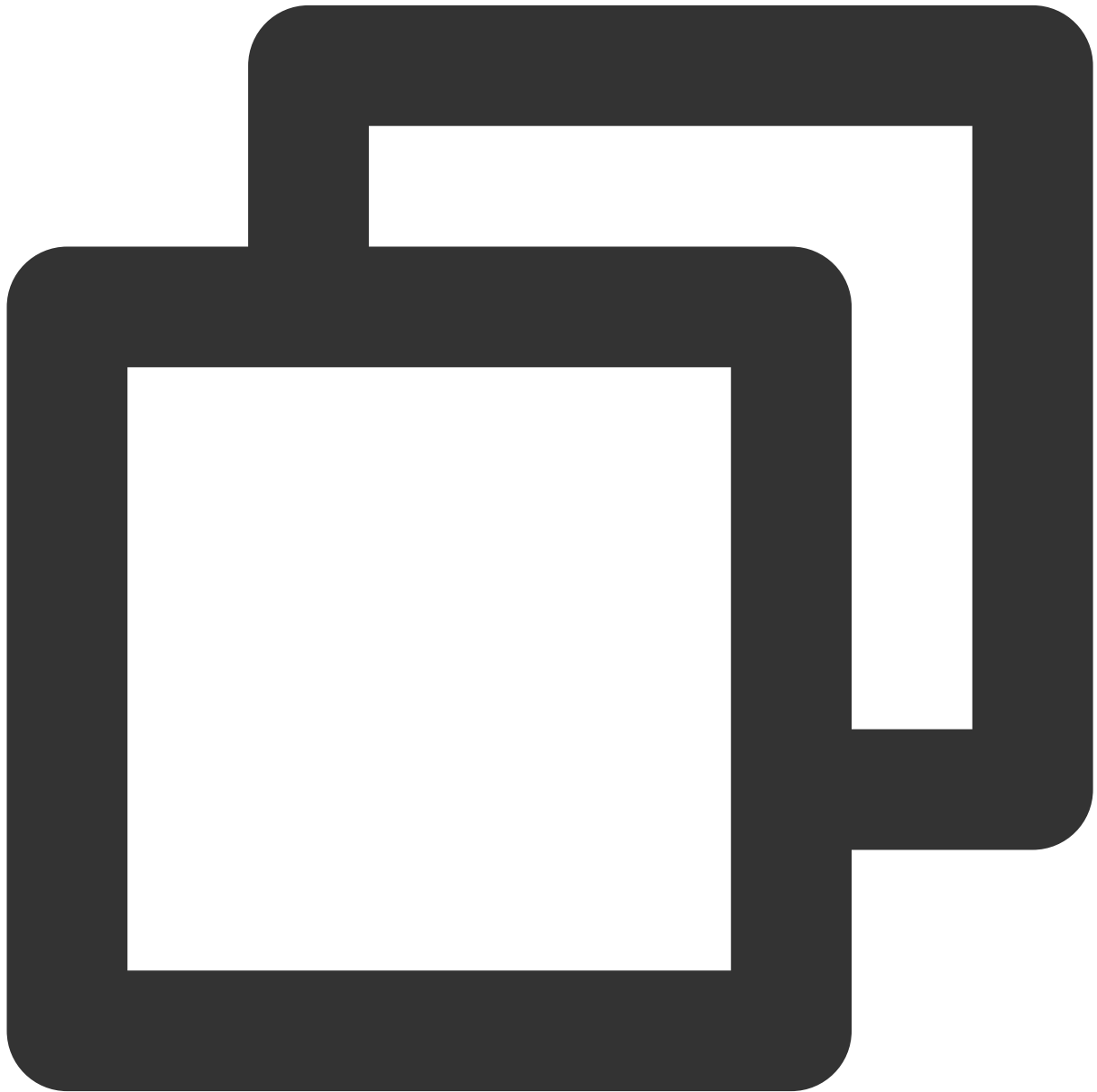
Parameter	Type	Meaning
seatIndex	int	Seat number

timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout detection and will not trigger timeout Callback
requestCallback	TUIRequestCallback?	Call interface Callback, used to notify the request Callback status

**Return:** Request body

## leaveSeat

Local off microphone.



```
Future<TUIActionCallback> leaveSeat ()
```

### **takeUserOnSeatByAdmin**

Host/Administrator Invite User on microphone.





```
TUIRequest takeUserOnSeatByAdmin(int seatIndex,  
                                  String userId,  
                                  int timeout,  
                                  TUIRequestCallback? requestCallback)
```

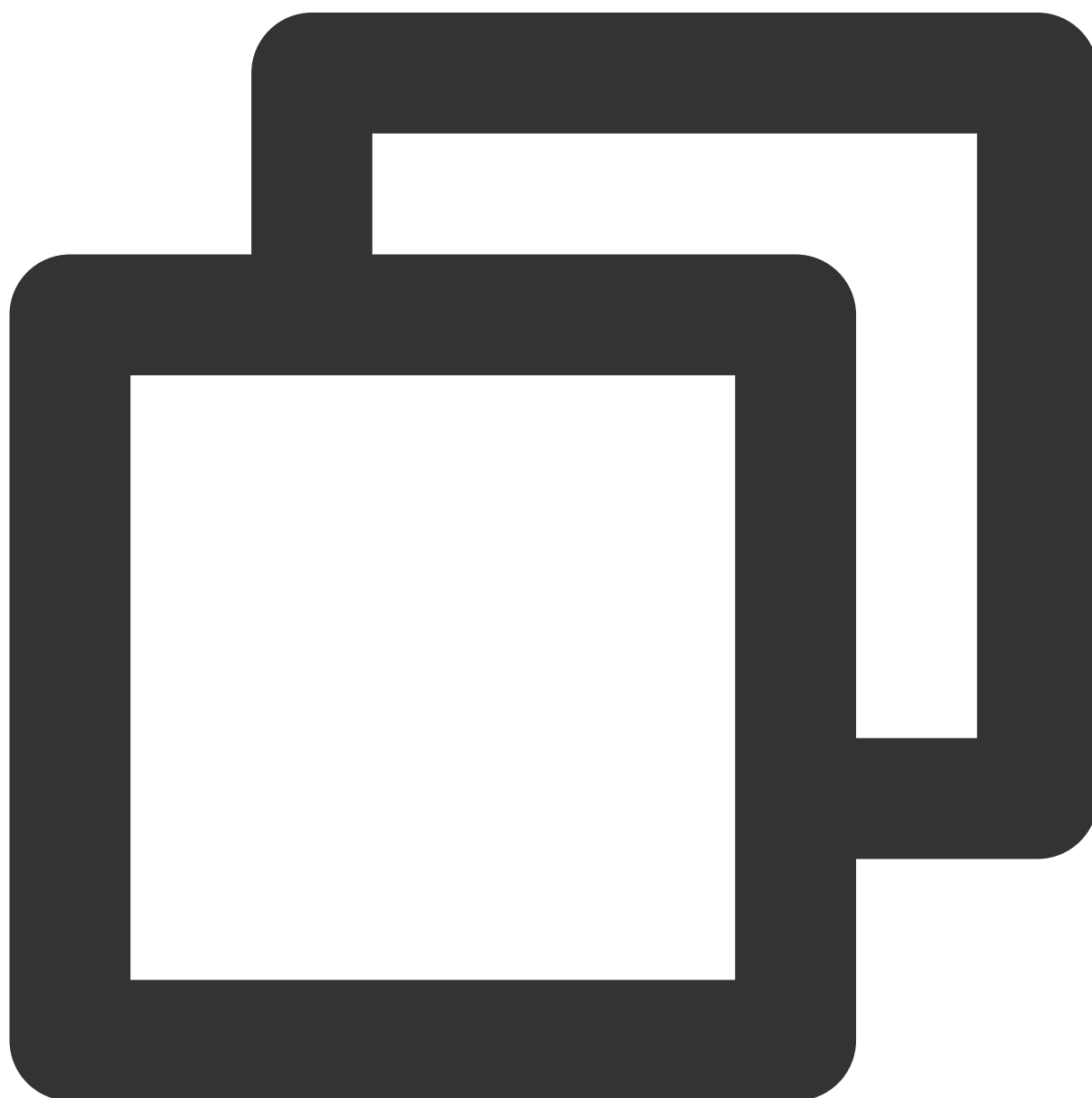
Parameter	Type	Meaning
seatIndex	int	Seat number
userId	String	User ID

timeout	int	Timeout in seconds, if set to 0, SDK will not do timeout detection and will not trigger timeout Callback
requestCallback	TUIRequestCallback?	Call interface Callback, used to notify the request Callback status

**Return:** Request body

### kickUserOffSeatByAdmin

Host/Administrator Take User off microphone.

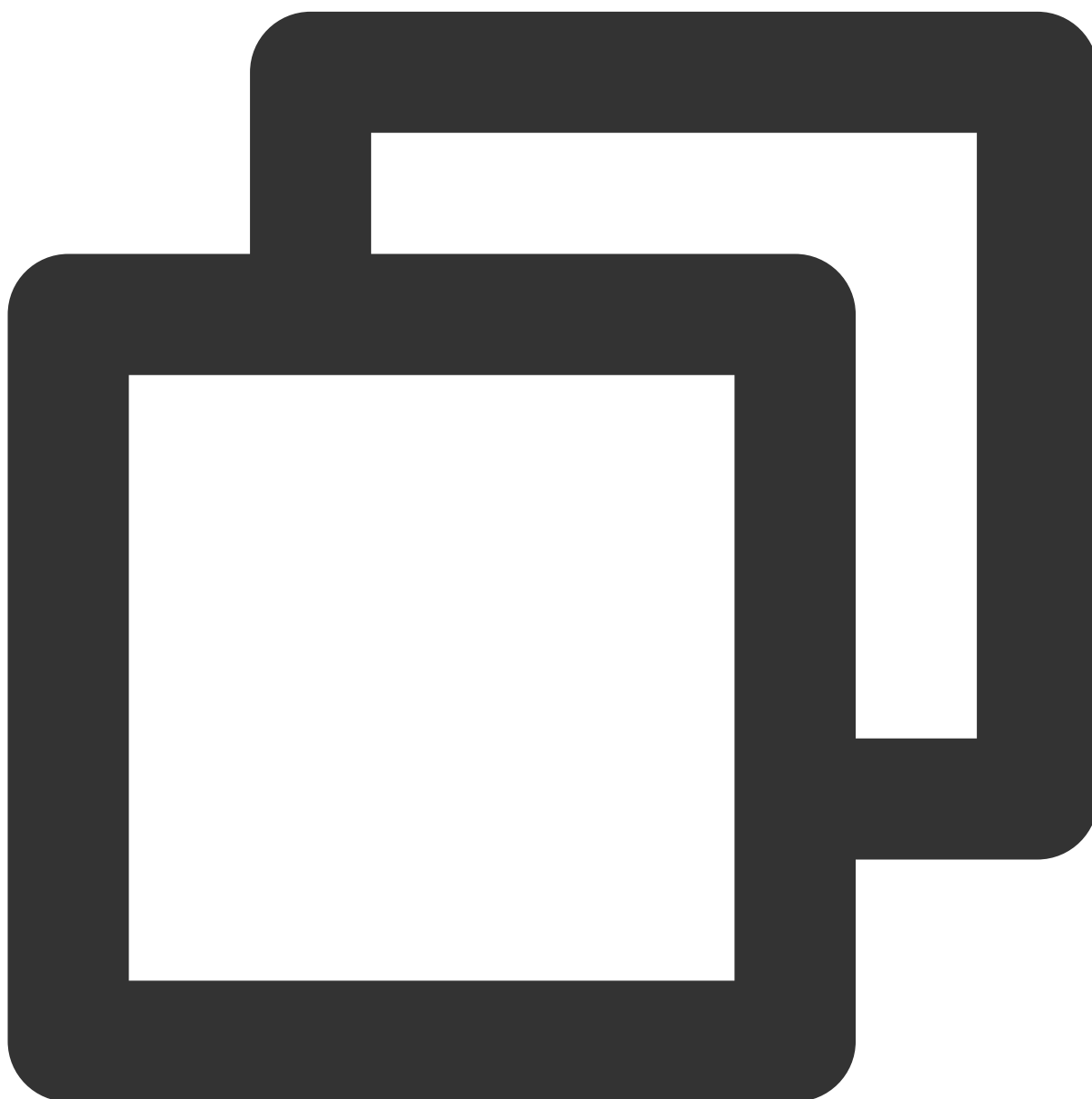


```
Future<TUIActionCallback> kickUserOffSeatByAdmin(int seatIndex,  
                                                String userId)
```

Parameter	Type	Meaning
seatIndex	int	Seat number
userId	String	User ID

## sendTextMessage

Send Text message.



```
Future<TUIActionCallback> sendTextMessage(String message)
```

**Parameters:**

Parameter	Type	Meaning
message	String	Text message Content

**sendCustomMessage**

Send Custom message



```
Future<TUIActionCallback> sendCustomMessage(String message)
```

**Parameters:**

Parameter	Type	Meaning
message	String	Custom message Content

**disableSendingMessageByAdmin**

Disable Remote user's Text message sending Ability (only Administrator or Group owner can call).

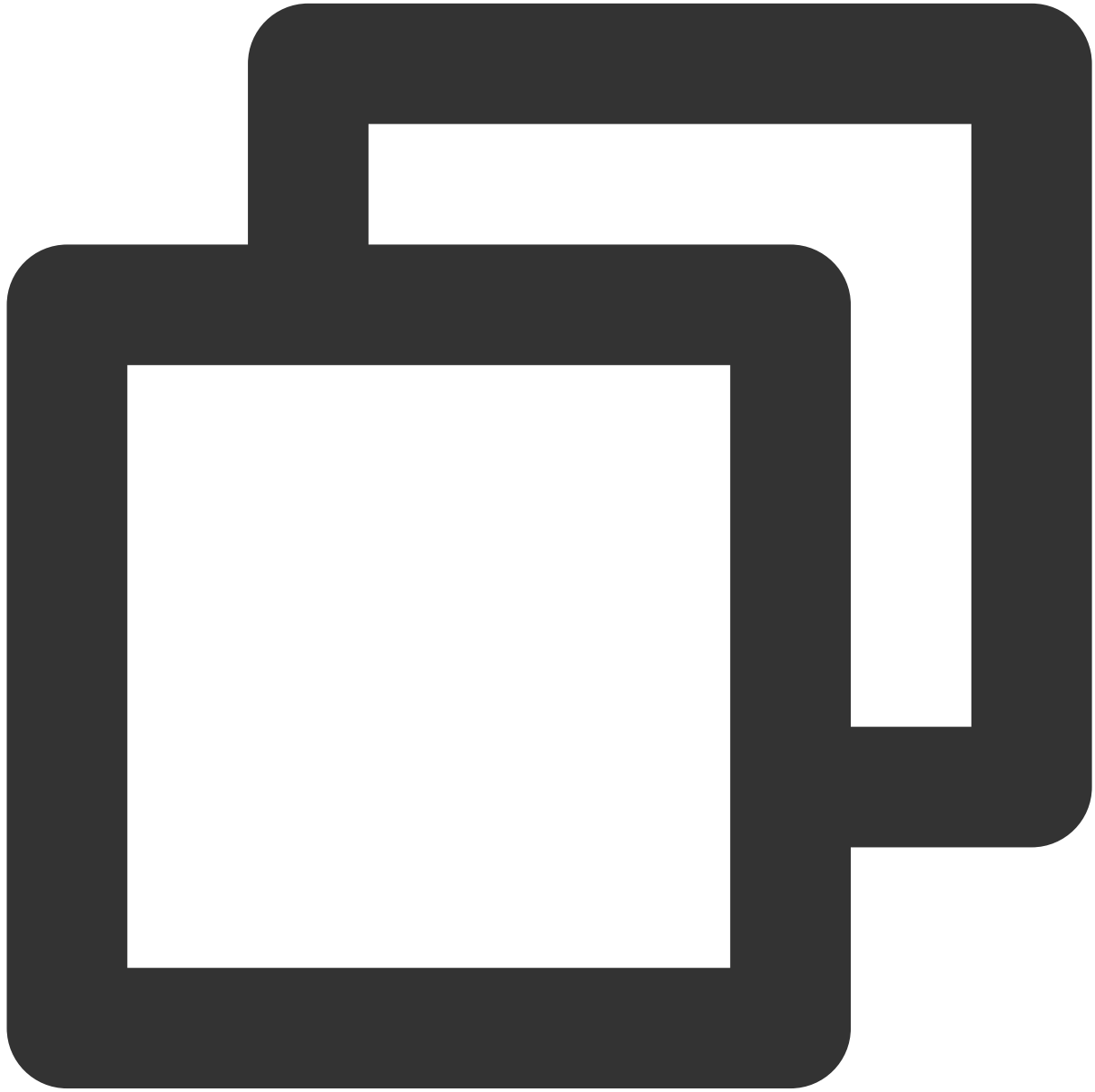


```
Future<TUIActionCallback> disableSendingMessageByAdmin(String userId,  
                                                         bool isDisable)
```

Parameter	Type	Meaning
userId	String	User ID
isDisable	bool	Whether to Disable

### **disableSendingMessageForAllUser**

Disable all users' Text message sending Ability (only Administrator or Group owner can call).



```
Future<TUIActionCallback> disableSendingMessageForAllUser(bool isDisable)
```

Parameter	Type	Meaning
isDisable	bool	Whether to Disable

### cancelRequest

Cancel sent Request.



```
Future<TUIActionCallback> cancelRequest (String requestId)
```

**Parameters:**

Parameter	Type	Meaning
requestId	String	Request ID

**responseRemoteRequest**

Reply to Remote user's Request.





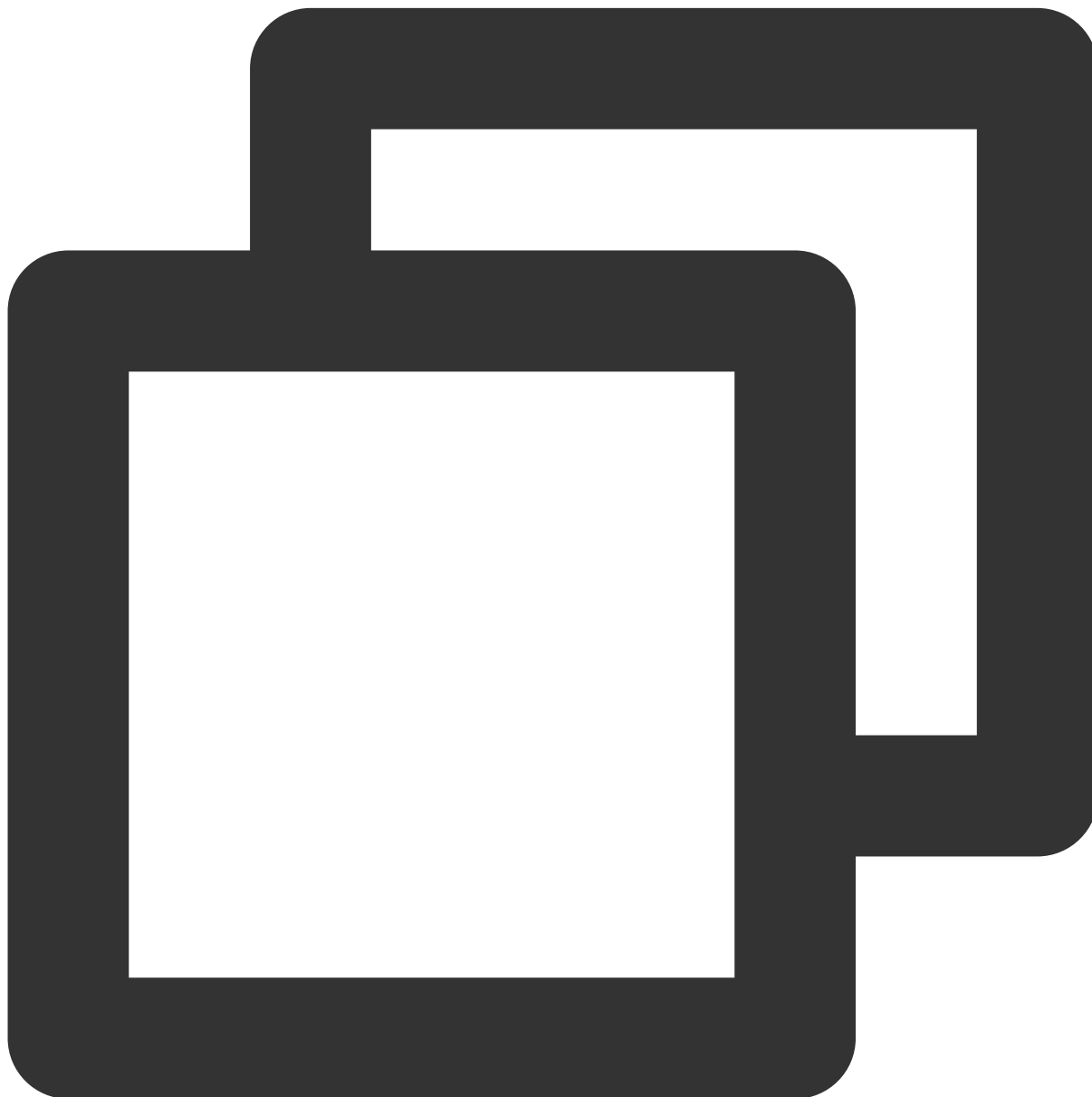
```
Future<TUIActionCallback> responseRemoteRequest(String requestId,  
                                                bool agree)
```

**Parameters:**

Parameter	Type	Meaning
requestId	String	Request ID
agree	bool	Whether to agree

## switchCamera

Switch front/rear camera

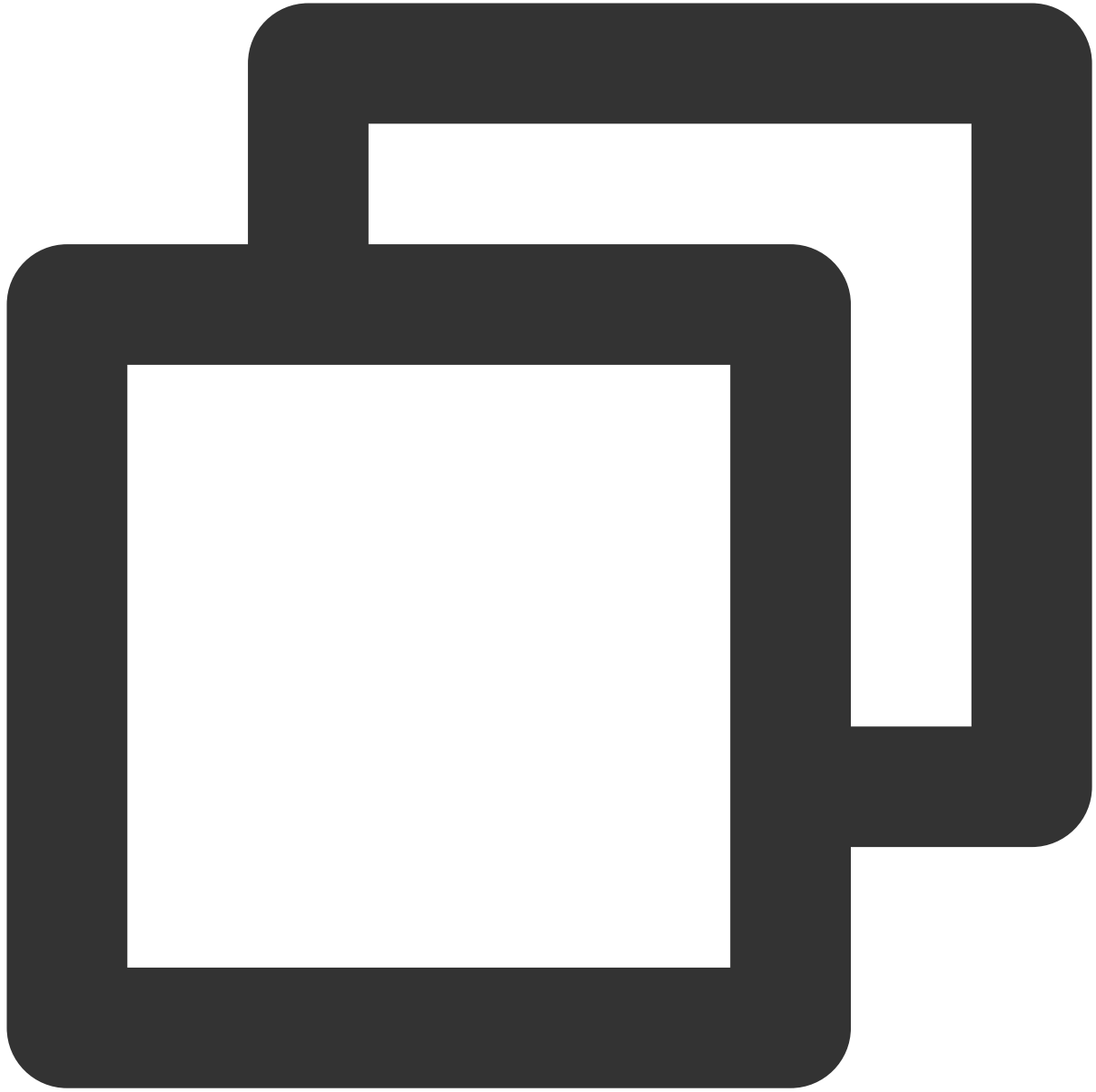


```
Future<int?> switchCamera(bool isFrontCamera);
```

Parameter	Type	Meaning
isFrontCamera	bool	Is front camera

## setBeautyLevel

Set beauty level

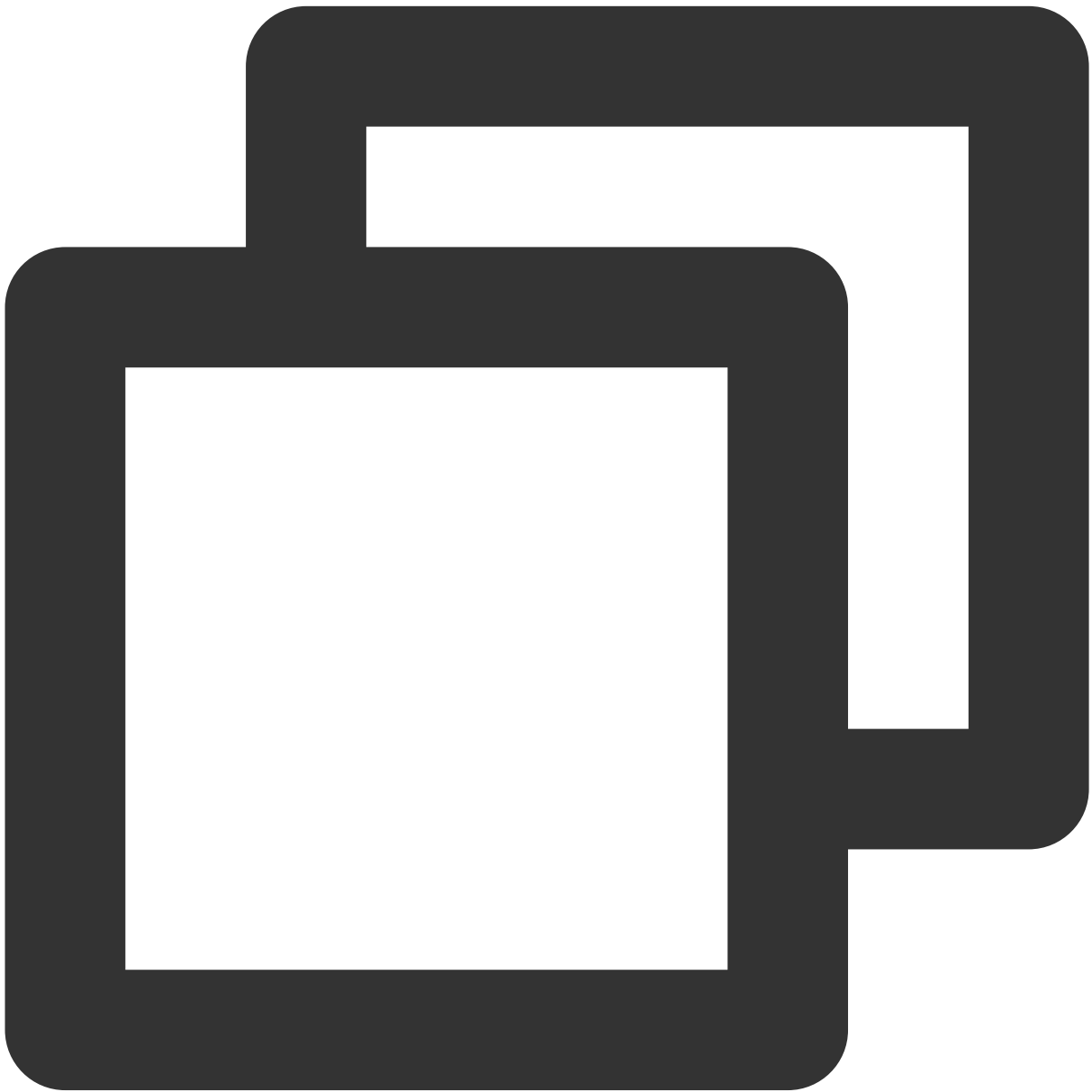


```
void setBeautyLevel(int beautyStyle, int beautyLevel);
```

Parameter	Type	Meaning
beautyStyle	int	beauty style
beautyLevel	int	beauty level

**setWhitenessLevel**

Set whiteness level



```
void setWhitenessLevel(int whitenessLevel);
```

Parameter	Type	Meaning
whitenessLevel	int	whiteness level

**callExperimentalAPI**

Call experimental api



```
void callExperimentalAPI(String jsonStr);
```

Parameter	Type	Meaning
jsonStr	String	Api infomation

# Type Definition

Last updated : 2023-11-21 15:17:59

## Enumeration Definition

### TUIRoomDefine

Type	Description
<a href="#">TUIRoomType</a>	Room Type
<a href="#">TUISpeechMode</a>	Mic Control Mode
<a href="#">TUIMediaDevice</a>	Room Media Device Type
<a href="#">TUIRole</a>	Room Role Type
<a href="#">TUIVideoQuality</a>	Video Quality
<a href="#">TUIAudioQuality</a>	Audio Quality
<a href="#">TUIVideoStreamType</a>	Video Stream Type
<a href="#">TUIChangeReason</a>	Change Reason (User audio and video status change operation reason: self-modification or modified by room owner/administrator)
<a href="#">TUICaptureSourceType</a>	Screen Sharing Capture Source Type
<a href="#">TUIRequestAction</a>	Request Type
<a href="#">TUIResolutionMode</a>	Video resolution mode (landscape or portrait)

### TUICommonDefine

Type	Description
<a href="#">TUIError</a>	Error Code
<a href="#">TUINetworkQuality</a>	Network Quality

## Common Structure

### TUIRoomDefine

Type	Description

<a href="#">TUIRoomInfo</a>	Room data
<a href="#">TUILoginUserInfo</a>	User Login Information
<a href="#">TUIUserInfo</a>	Room User Information
<a href="#">TUISeatInfo</a>	Room Seat Information
<a href="#">TUISeatLockParams</a>	Lock Seat Operation Parameters
<a href="#">TUIUserVoiceVolume</a>	Room User Volume
<a href="#">TUIRequest</a>	Signaling Request
<a href="#">TUIActionCallback</a>	User Operation Callback
<a href="#">TUIPlayCallback</a>	Video Playback Callback
<a href="#">TUIRequestCallback</a>	User Request Callback
<a href="#">TUIUserListResult</a>	User List Information
<a href="#">TUIUserVoiceVolume</a>	User Volume Information
<a href="#">TUIValueCallBack&lt;T&gt;</a>	With Return Value (T) Callback
<a href="#">TUIRoomVideoEncoderParams</a>	Video encoder params

### TUICommonDefine

Type	Description
<a href="#">TUINetwork</a>	Network Quality Information
<a href="#">TUIMessage</a>	Message
<a href="#">TUIImageBuffer</a>	Image Info

### TUIRoomType

Room Type

Enumeration	Value	Description
conference	1	Conference Type Room, suitable for conference and education scenarios, this room can enable free speech, apply for speech, go live and other modes.
livingRoom	2	Live Type Room, suitable for live broadcast scenarios, this room can

enable free speech, mic control mode, and the seats in this room are numbered.

## TUISpeechMode

### Mic Control Mode

Enumeration	Value	Description
freeToSpeak	1	Free speech mode
applyToSpeak	2	Apply to speak mode. (Only effective in conference type room)
applySpeakAfterTakingSeat	3	Go Live mode

### Explanation:

The relationship between Room Type, Mic Control Mode, and Going Live (takeSeat)

Room Type	Mic Control Mode		
	freeToSpeak	applyToSpeak	applySpeakAfterTakingSeat
conference	Not Supported	Not Supported	Need to Apply to the Host/Admin (takeSeat), Can Speak with Mic/Camera After Approval
livingRoom	Can Freely Go Live	Not Supported	Need to Apply to the Host/Admin (takeSeat), Can Speak with Mic/Camera After Approval

## TUIMediaDevice

### Room Media Device Type

Enumeration	Value	Description
microphone	1	Mic
camera	2	Camera
screen	3	Screen Sharing



## TUIRole

### Room Role Types

Enumeration	Value	Description
roomOwner	0	Room Owner. Generally refers to the creator of the room, the highest authority holder in the room
administrator	1	Room Administrator
generalUser	2	General Member in the room

## TUIVideoQuality

### Video Quality

Enumeration	Value	Description
videoQuality_360P	1	Low Definition 360P
videoQuality_540P	2	Standard Definition 540P
videoQuality_720P	3	High Definition 720P
videoQuality_1080P	4	Ultra Definition 1080P

## TUIAudioQuality

### Audio Quality

Enumeration	Value	Description
audioProfileSpeech	0	Vocal Mode
audioProfileDefault	1	Default Mode
audioProfileMusic	2	Music Mode

## TUIVideoStreamType

### Video Stream Types

Enumeration	Value	Description
cameraStream	0	HD Camera Video Stream
screenStream	1	Screen Sharing Video Stream

cameraStreamLow	2	Low Definition Camera Video Stream
-----------------	---	------------------------------------

## TUIChangeReason

Change Reason (User audio and video status change operation reason: self-initiated modification or modified by room owner/administrator)

Enumeration	Value	Description
changedBySelf	0	Self-operation
changedByAdmin	1	Room Owner or Administrator operation

## TUICaptureSourceType

Screen Sharing Capture Source Type

Enumeration	Value	Description
unknown	-1	Undefined
window	0	Window
screen	1	Screen

## TUIRequestAction

Request Type

Enumeration	Value	Description
invalidAction	0	Invalid request
requestToOpenRemoteCamera	1	Request remote user to open the camera
requestToOpenRemoteMicrophone	2	Request remote user to open the microphone
requestToConnectOtherRoom	3	Request to connect to another room
requestToTakeSeat	4	Request to go live
requestRemoteUserOnSeat	5	Request remote user to go live
applyToAdminToOpenLocalCamera	6	Request to the administrator to open the local camera
applyToAdminToOpenLocalMicrophone	7	Request to the administrator to open the local microphone

## TUIResolutionMode

Video resolution mode (landscape or portrait)

Enumeration	Value	Description
landscape	0	landscape
portrait	1	portrait

## TUIError

Error Code

Enumeration	Value	Description
success	0	Operation Successful
errFailed	-1	Temporarily Unclassified General Error
errFreqLimit	-2	Request Limited, Please Try Again Later
errRepeatOperation	-3	Repeated operation, please check whether your interface call is repeated
errSDKAppIDNotFound	-1000	SDKAppID Not Found, Please Confirm Application Info in <a href="#">TRTC Console</a>
errInvalidParameter	-1001	Illegal Input Parameters When Calling API
errSdkNotInitialized	-1002	SDK Not Initialized
errPermissionDenied	-1003	No Operation Permission
errRequirePayment	-1004	Need to Open Extra Package for This Feature
errCameraStartFailed	-1100	Failed to Open Camera
errCameraNotAuthorized	-1101	Camera Not Authorized
errCameraOccupy	-1102	Camera Occupied
errCameraDeviceEmpty	-1103	No Camera Device Currently
errMicrophoneStartFailed	-1104	Microphone Opening Failed
errMicrophoneNotAuthorized	-1105	Microphone Not Authorized
errMicrophoneOccupy	-1106	Microphone Occupied

errMicrophoneDeviceEmpty	-1107	No Microphone Device Currently
errGetScreenSharingTargetFailed	-1108	Failed to Get Screen Sharing Target
errStartScreenSharingFailed	-1109	Failed to Start Screen Sharing
errRoomIdNotExist	-2100	Room Does Not Exist When Entering, or Maybe Closed
errOperationInvalidBeforeEnterRoom	-2101	Need to Enter the Room Before Using This Feature
errExitNotSupportedForRoomOwner	-2102	Room Owner Does Not Support Exit Operation, Conference Room Type: You can transfer the room owner first, then exit the room. LivingRoom Room Type: The room owner can only close the room
errOperationNotSupportedInCurrentRoomType	-2103	This Operation is Not Supported in the Current Room Type
errOperationNotSupportedInCurrentSpeechMode	-2104	This Operation is Not Supported in the Current Speech Mode
errRoomIdInvalid	-2105	Illegal Room ID Creation, Custom ID Must Be Printable ASCII Characters (0x20-0x7e), Up to 48 Bytes
errRoomIdOccupied	-2106	Room ID Already in Use, Please Choose Another Room ID
errRoomNameInvalid	-2107	Illegal Room Name, The Name Must Be Up to 30 Bytes, The Character Encoding Must Be UTF-8, If It Contains Chinese
errAlreadyInOtherRoom	-2108	The Current User is Already in Another Room, You Need to Exit the Room First to Join a New Room. A single roomEngine instance only supports the user entering one room. If you want to enter a different room, please exit the room first or use a new roomEngine instance
errUserNotExist	-2200	User Does Not Exist
errUserNotEntered	-2201	User is Not in the Current Room
errUserNeedOwnerPermission	-2300	Room Owner Permission Required for

		Operation
errUserNeedAdminPermission	-2301	Room Owner or Administrator Permission Required for Operation
errRequestNoPermission	-2310	No Permission for Signaling Request, For Example, Canceling an Invitation Not Initiated by Yourself
errRequestIdInvalid	-2311	Invalid Signaling Request ID or Already Processed
errMaxSeatCountLimit	-2340	Maximum Seat Exceeds Package Quantity Limit
errAlreadyInSeat	-2341	Current User is Already on the Seat
errSeatOccupied	-2342	The Current Seat is Already Occupied
errSeatLocked	-2343	The Current Seat is Locked
errSeatIndexNotExist	-2344	Seat Number Does Not Exist
errUserNotInSeat	-2345	Current User is Not on the Mic
errAllSeatOccupied	-2346	The Number of People on the Mic is Full
errOpenMicrophoneNeedSeatUnlock	-2360	The Current Seat Audio is Locked
errOpenMicrophoneNeedPermissionFromAdmin	-2361	Need to Apply to the Room Owner or Administrator to Open the Microphone
errOpenCameraNeedSeatUnlock	-2370	The Current Seat Video is Locked, The Room Owner Needs to Unlock the Seat Before Opening the Camera
errOpenCameraNeedPermissionFromAdmin	-2371	Need to Apply to the Room Owner or Administrator to Open the Camera
errSendMessageDisabledForAll	-2380	The Current Room Has Enabled Mute for All
errSendMessageDisabledForCurrent	-2381	In the Current Room, You Have Been Muted

## TUINetworkQuality

### Network Quality

Enumeration	Value	Description

qualityUnknown	0	Undefined
qualityExcellent	1	The Current Network is Very Good
qualityGood	2	The Current Network is Good
qualityPoor	3	The Current Network is Average
qualityBad	4	The Current Network is Poor
qualityVeryBad	5	The Current Network is Very Poor
qualityDown	6	The Current Network Does Not Meet the Minimum Requirements of TRTC

## TUIRoomInfo

Room data

Field	Type	Description
roomId	String	Room ID
roomType	<a href="#">TUIRoomType</a>	Room Type
ownerId	String	Host ID, Default is Room Creator (Read Only)
name	String	Room Name, Default is Room ID
speechMode	<a href="#">TUISpeechMode</a>	Room Speech Mode
createTime	int	Room Creation Time (Read Only)
memberCount	int	Number of Members in the Room (Read Only)
maxSeatCount	int	Maximum Seat Quantity (Only Supported When Entering the Room and Creating the Room)
isCameraDisableForAllUser	bool	Whether to Prohibit Opening the Camera (Optional Parameter When Creating a Room). Default Value: false
isMicrophoneDisableForAllUser	bool	Whether to Prohibit Opening the Microphone (Optional Parameter When Creating a Room). Default Value: false
isMessageDisableForAllUser	bool	Whether to Prohibit Sending Messages (Optional Parameter When Creating a Room). Default Value: false

enableCDNStreaming	bool	Whether to Enable CDN Live Streaming (Optional Parameter When Creating a Room, for Live Room Use). Default Value: false
cdnStreamDomain	String	Live Streaming Push Domain (Optional Parameter When Creating a Room, for Live Room Use). Default Value: Empty

## TUILoginUserInfo

### User Login Info

Field	Type	Meaning
userId	String	User ID
userName	String	User Name
avatarUrl	String	User Avatar URL
customInfo	Map<String, String>	Custom Information

## TUIUserInfo

### User Information in the Room

Field	Type	Description
userId	String	User ID
userName	String	User Name
avatarUrl	String	User Avatar URL
userRole	<a href="#">TUIRole</a>	User Role Type
hasAudioStream	bool	Whether There is an Audio Stream. Default Value: false
hasVideoStream	bool	Whether There is a Video Stream. Default Value: false
hasScreenStream	bool	Whether There is a Screen Sharing Stream. Default Value: false

## TUISeatInfo

### Seat Information in the Room

Field	Type	Description
-------	------	-------------

index	int	Seat Number
userId	String	User ID
isLocked	bool	Whether the Seat is Locked. Default Value: false
isVideoLocked	bool	Whether the Seat is Prohibited from Opening the Camera. Default Value: false
isAudioLocked	bool	Whether the Seat is Prohibited from Opening the Microphone. Default Value: false

## TUISeatLockParams

Lock Seat Operation Parameters

Field	Type	Meaning
lockSeat	bool	Lock Seat. Default Value: false
lockVideo	bool	Lock Seat Camera. Default Value: false
lockAudio	bool	Lock Seat Microphone. Default Value: false

## TUIUserVoiceVolume

User Volume in the Room

Field	Type	Description
userId	String	User ID
volume	int	Volume. Used to Carry the Volume of All Speaking Users, Value Range 0 - 100

## TUIRequest

Signaling Request

Field	Type	Description
requestId	String	Request ID
requestAction	<a href="#">TUIRequestAction</a>	Request Type
userId	String	User ID
content	String	Signaling Content



timestamp	int	Timestamp
-----------	-----	-----------

**TUIActionCallback**

Field	Type	Description
code	<a href="#">TUIError</a>	Error Code
message	String?	Error Information

**TUIPlayCallback**

Field	Type	Description
onPlaying	(String userId) {}	Playing Callback
onLoading	(String userId) {}	Loading Callback
onPlayError	(String userId, TUIError code, String message) {}	Playback Error Callback

**TUIRequestCallback**

Field	Type	Description
onAccepted	(String requestId, String userId) {}	Request Accepted Callback
onRejected	(String requestId, String userId, String message) {}	Request Rejected Callback
onCancelled	(String requestId, String userId) {}	Request Cancelled Callback
onTimeout	(String requestId, String userId) {}	Request Timeout Callback
onError	(String requestId, String userId, TUIError error, String message) {}	Request Error Callback

**TUIUserListResult**

Field	Type	Description
nextSequence	int	Pagination Fetch Flag, if the returned nextSequence is not zero, you need to use the returned nextSequence to fetch again until it returns 0

userInfoList	List<TUIUserInfo>	The user list returned by this call
--------------	-------------------	-------------------------------------

## TUIUserVoiceVolume

Field	Type	Description
userId	String	User ID
volume	int	User Volume Size

## TUIValueCallBack<T>

Field	Type	Description
code	<a href="#">TUIError</a>	Error Code
message	String?	Error Message
data	T?	Return Data, example: if T is TUIUserInfo, then the data field type of TUIValueCallBack<TUIUserInfo> is TUIUserInfo

## TUIRoomVideoEncoderParams

Video encoder params

Field	Type	Description
videoResolution	<a href="#">TUIVideoQuality</a>	Video resolution
resolutionMode	<a href="#">TUIResolutionMode</a>	Video resolution mode
fps	int	Video capturing frame rate
bitrate	int	Target video bitrate

## TUINetwork

Network Quality Information

Field	Type	Description
userId	String	User ID
quality	<a href="#">TUINetworkQuality</a>	Network Quality
upLoss	int	Packet Loss Rate for Upstream

downLoss	int	Packet Loss Rate for Downstream
delay	int	Network Delay

## TUIMessage

### Message

Field	Type	Description
messageId	String	Message ID
message	String	Message Text
timestamp	int	Message Timestamp
userId	String	Message Sender
userName	String	Message Sender Nickname
avatarUrl	String	Message Sender Avatar

## TUIImageBuffer

### Image Info

Field	Type	Description
buffer	String	Image Data Cache Address
length	int	Length
width	int	Width
height	int	Height

# FAQs

## Web

Last updated : 2023-10-30 11:03:31

## Environment-related Issues

### What platforms does TUIRoomKit Web support?

Please refer to [TRTC Web SDK's browser support](#) for TUIRoomEngine Web supported platforms.

For environments not listed above, you can open the [TRTC Capability Test](#) in your current browser to test if it fully supports WebRTC features.

### Why can TUIRoomKit be used normally in local development testing, but not when deployed online?

Considering user safety and privacy issues, browsers restrict web pages to only capture mic and Camera in secure environments (such as https, localhost, file:// protocols). HTTP protocol is insecure, and browsers will prohibit media device capturing under HTTP protocol.

If everything works fine in your local development testing, but you cannot capture Camera and mic after deploying the page, please check if your web page is deployed on HTTP protocol. If so, please use HTTPS to deploy your web page and ensure a qualified HTTPS security certificate.

For more details, please refer to [the URL domain and protocol restrictions description](#).

### Does TUIRoomKit Web support integration with iframe?

Yes, it does. To integrate TUIRoom Web in an iframe, you need to add attributes to the iframe tag to enable related permissions, as shown below.

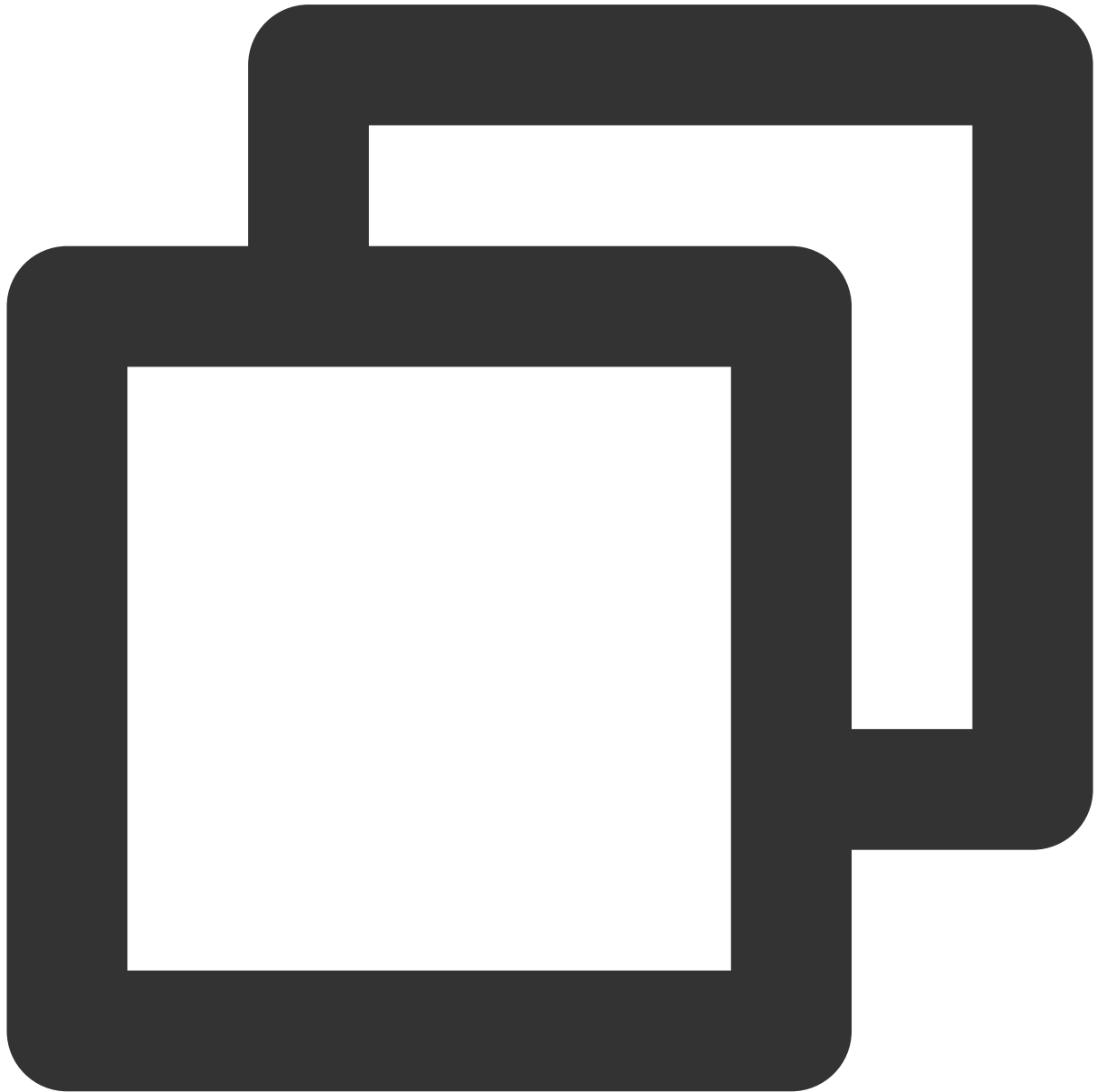


```
// Enable mic, Camera, Screen Sharing, and full-screen permissions
<iframe allow="microphone; camera; display-capture; display; fullscreen;">
```

## Compilation-related Issues

**Webpack 5 importing TUIRoomEngine SDK error: webpack < 5 used to include polyfills for node.js core modules by default. This is no longer the case. Verify if you need this module and configure a polyfill for it.**

The error is caused by the removal of the automatic inclusion of nodejs core module polyfills in webpack5. You can add `configureWebpack` configuration in `vue.config.js` to solve this problem.



```
module.exports = defineConfig({
  // ...
  configureWebpack: (config) => {
    config.resolve.fallback = {
      ...config.resolve.fallback,
      url: false,
      path: false,
      fs: false,
    }
  }
})
```

```
        crypto: false,  
      };  
    }  
  });
```

## Function-related Issues

### Can `tim-js-sdk` and `@tencentcloud/tuiroom-engine-js` be introduced in the project at the same time?

Yes, they can. You can create a `tim` instance object through `TIM` in `tim-js-sdk` and pass it to the `init` interface when initializing `TUIRoomEngine`.



```
await TUIRoomEngine.init({
  sdkAppId: 0,    // Fill in your applied sdkAppId
  userId: '',     // Fill in your business-related userId
  userSig: '',    // Fill in the userSig calculated by the server or locally
  tim,           // Pass in the tim instance
});
```

Also, you can get the tim instance used internally by TUIRoomEngine through the `roomEngine.getTIM` method.



# iOS

Last updated : 2023-10-30 14:10:38

**I need to modify the UI myself. Every time I update the pod, the source code will be refreshed, causing the modifications to be lost. How should I handle this?**

It is suggested to Fork the [TUIRoomKit repository](#) to your personal GitHub account, and then use the [local pod](#) or [git pod path](#) to reference the corresponding code in your project.

For details, please refer to the [official Pod documentation](#):

## Using the files from a folder local to the machine.

If you would like to develop a Pod in tandem with its client project you can use `:path`.

```
pod 'Alamofire', :path => '~/Documents/Alamofire'
```

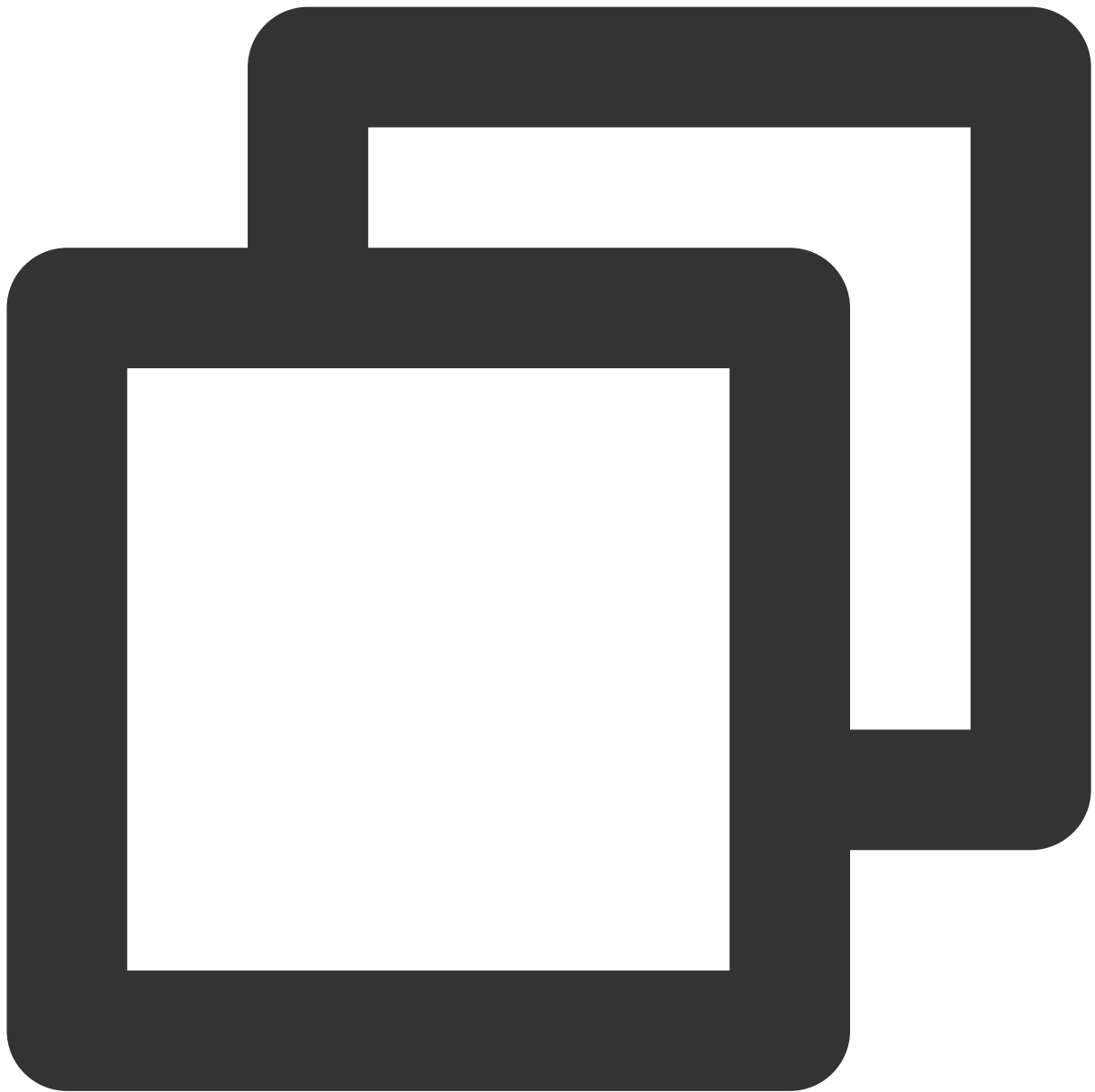
Using this option CocoaPods will assume the given folder to be the root of the Pod and will link the files directly from there in your project. This means that your edits will persist between CocoaPods installations. The referenced folder can be a checkout of your favorite framework or a git submodule of the current repo.

Note that the `podspec` of the Pod file is expected to be in that the designated folder.

## Is there a conflict between TUIRoomKit and the integrated audio and video library?

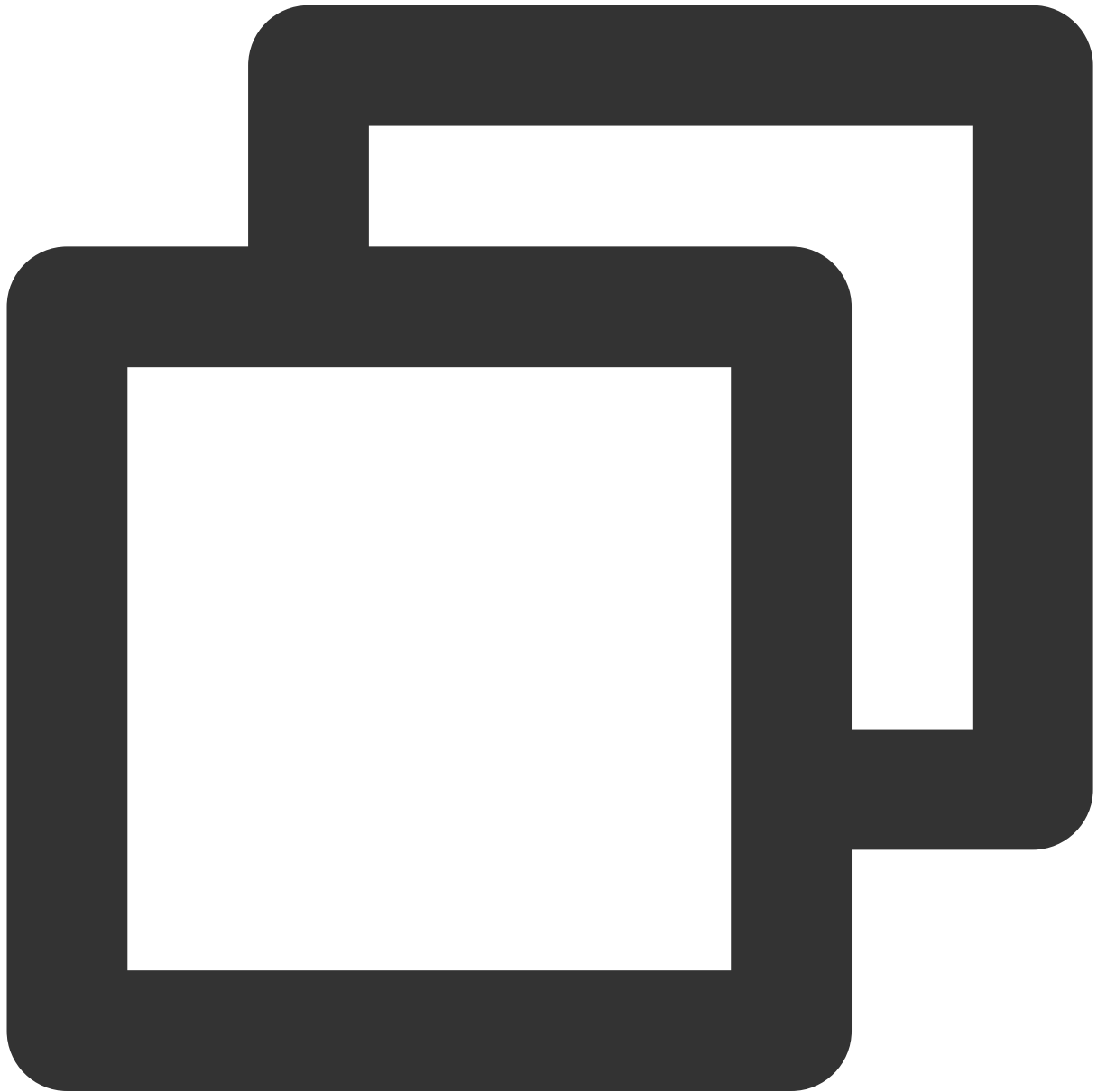
Tencent Cloud's audio and video libraries cannot be integrated at the same time, and there may be symbol conflicts. You can handle it according to the following scenarios.

1. If you are using the `TXLiteAVSDK_TRTC` library, there will be no symbol conflicts. You can directly add dependencies in the Podfile file.



```
pod 'TUIRoomKit'
```

2. If you are using the `TXLiteAVSDK_Professional` library, there will be symbol conflicts. You can add dependencies in the `Podfile` file.



```
pod 'TUIRoomKit/Professional'
```

3. If you are using the `TXLiteAVSDK_Enterprise` library, there will be symbol conflicts. It is recommended to upgrade to `TXLiteAVSDK_Professional` and then use `TUIRoomKit/Professional`.

### How to view TRTC logs?

TRTC logs are compressed and encrypted by default, with the extension `.xlog`. Whether the log is encrypted can be controlled by `setLogCompressEnabled`. The file name containing `C`(compressed) is encrypted and compressed, and the file name containing `R`(raw) is plaintext.

iOS&Mac : Sandbox's `Documents/log`

Android :

Versions 6.7 and earlier : `/sdcard/log/tencent/liteav`

Versions after 6.8 : `/sdcard/Android/data/package name/files/log/tencent/liteav/`

Versions after 8.5 : `/sdcard/Android/data/package name/files/log/liteav/`

Windows :

Versions before 8.8 : `%appdata%/tencent/liteav/log`

Versions 8.8 and later : `%appdata%/liteav/log`

Web : Open the browser Console, or use vConsole to record SDK print information.

**Note:**

To view the .xlog file, you need to download the [decryption tool](#) and run it directly in the Python 2.7 environment with the xlog file in the same directory using `python decode_mars_log_file.py`.

To view the .clog file (new log format after version 9.6), you need to download the [decryption tool](#) and run it directly in the Python 2.7 environment with the clog file in the same directory using `python decompress_clog.py`.

# Android

Last updated : 2023-10-30 11:06:45

## What are the requirements for TUIRoomKit's dependencies on TRTC and IM SDK?

TUIRoomEngine will perform version verification when logging in, prompting you to use the correct TRTC SDK and IM SDK Version.

```
W [W] [05-25/20:11:34.781+8.0] [7993,7993] [global_service.cc:239]Tuikit: warning: im recommend to use 7.2.4
W [W] [05-25/20:11:34.781+8.0] [7993,7993] [global_service.cc:248]Tuikit: warning: trtc recommend to use 11.
```

## Can TUIRoomKit remove the dependency on IM SDK?

No, Real-time Conference SDK requires IM for signaling-related communication, including mic control, bullet screen, and other functions, so it needs to have a dependency on IM SDK.

## Does TUIRoomKit support features like conference scheduling?

It is not supported yet, this feature is in planning, you can follow our update log to get the latest information on the launch status of this feature.

## After integrating TUIRoomKit, how can I quickly modify the user name and user avatar?

You can set it by calling the `setSelfInfo` interface of TUIRoomKit.

# Electron

Last updated : 2023-10-30 11:07:54

## Environment-related issues

### **Is trtc-electron-sdk compatible with the official Electron v12.0.1 version?**

Yes, trtc-electron-sdk is not specifically dependent on Electron's own SDK, so there are no related version dependencies.

# Flutter

Last updated : 2023-11-21 14:55:33

## An error occurs when running on iOS: Error (Xcode): Library not found for -ld64 ?

Xcode 15 changed the compiled linker , and RoomEngine Flutter was adapted in 1.6.0. Old versions of Xcode cannot be compiled, so just use the latest Xcode .

The project integrates tencent\_calls\_uikit and rtc\_room\_engine at the same time. There will be problems such as the two components interfering with each other when joining the room will interrupt the call. How to Resolve ?

This is due to the exclusivity of the RTC service. We recommend that you use two flags, callState and roomState , to record whether you are currently on a call or in a conference. These two flags are used to isolate the two services. , to avoid this problem. For example, if you are currently on a call, entry into the room is prohibited.

# Error Code

Last updated : 2023-12-18 18:13:00

## General Error Code

Error Code	Description
0	Operation Successful
-1	Temporarily Unclassified General Error
-2	Request Rate Limited, Please Try Again Later
-1000	Not Found SDKAppID, Please Confirm Application Info in <a href="#">TRTC Console</a>
-1001	Passing illegal parameters when calling API, check if the parameters are legal
-1002	Not Logged In, Please Call Login API
-1003	Failed to Obtain Permission, Unauthorized Audio/Video Permission, Please Check if Device Permission is Enabled
-1004	This feature requires additional Package, please enable the corresponding Package as needed

## Local User Rendering, Video Management, Audio Management API Callback Error Definition

Error Code	Description
-1100	System Issue, Failed to Open Camera. Check if Camera Device is Normal
-1101	Camera has No System Authorization, Check System Authorization
-1102	Camera is Occupied, Check if Other Process is Using Camera
-1103	No Camera Device Currently, Please Insert Camera Device to Solve the Problem
-1104	System Issue, Failed to Open Mic. Check if Mic Device is Normal
-1105	Mic has No System Authorization, Check System Authorization
-1106	Mic is Occupied



-1107	No Mic Device Currently
-1108	Failed to Obtain Screen Sharing Object, Check Screen Recording Permission
-1109	Failed to Enable Screen Sharing, Check if Someone is Already Screen Sharing in the Room

## Room Management Related API Callback Error Definition

Error Code	Description
-2100	Room Does Not Exist When Entering, May Have Been Closed
-2101	This Feature Can Only Be Used After Entering the Room
-2102	Room Owner Does Not Support Leaving the Room, Conference Room Type: Transfer Room Ownership First, Then Leave the Room. Living Room Type: Room Owner Can Only Close the Room
-2103	This Operation is Not Supported in the Current Room Type
-2104	This Operation is Not Supported in the Current Speaking Mode
-2105	Illegal Custom Room ID, Must Be Printable ASCII Characters (0x20-0x7e), Up to 48 Bytes Long
-2106	Room ID is Already in Use, Please Choose Another Room ID
-2107	Illegal Room Name, Maximum 30 Bytes, Must Be UTF-8 Encoding if Contains Chinese Characters
-2108	User is Already in Another Room, Single RoomEngine Instance Only Supports User Entering One Room, To Enter Different Room, Please Leave the Room or Use New RoomEngine Instance

## Room User Information API Callback Error Definition

Error Code	Description
-2200	User Not Found
-2201	User Not Found in the Room

## Room User Speech Management API Callback Error Definition & Room Mic Seat Management API Callback Error Definition

Error	Description
-------	-------------

Code	
-2300	Room Owner Permission Required for Operation
-2301	Room Owner or Administrator Permission Required for Operation
-2310	No Permission for Signaling Request, e.g. Canceling an Invite Not Initiated by Yourself
-2311	Signaling Request ID is Invalid or Has Been Processed
-2340	Maximum Mic Seat Exceeds Package Quantity Limit
-2341	Current User is Already on Mic Seat
-2342	Mic Seat is Already Occupied
-2343	Mic Seat is Locked
-2344	Mic Seat Serial Number Does Not Exist
-2345	Current User is Not on Mic
-2346	Mic-on Capacity is Full
-2360	Current Mic Seat Audio is Locked
-2361	Need to Apply to Room Owner or Administrator to Open Mic
-2370	Current Mic Seat Video is Locked, Need Room Owner to Unlock Mic Seat Before Opening Camera
-2371	Need to Apply to Room Owner or Administrator to Open Camera
-2380	All Members Muted in the Current Room
-2381	You Have Been Muted in the Current Room

# Interactive Live Audio Streaming

## Integrating TUIVoiceRoom (Android)

Last updated : 2024-01-18 11:18:54

### Overview

**TUIVoiceRoom** is an open-source audio/video UI component. After integrating it into your project, you can make your application support the group audio chat scenario simply by writing a few lines of code. It also supports the [iOS](#) platform. Its basic features are as shown below:

#### Note

All TUIKit components are based on two PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For billing details of Chat, see [Pricing](#).

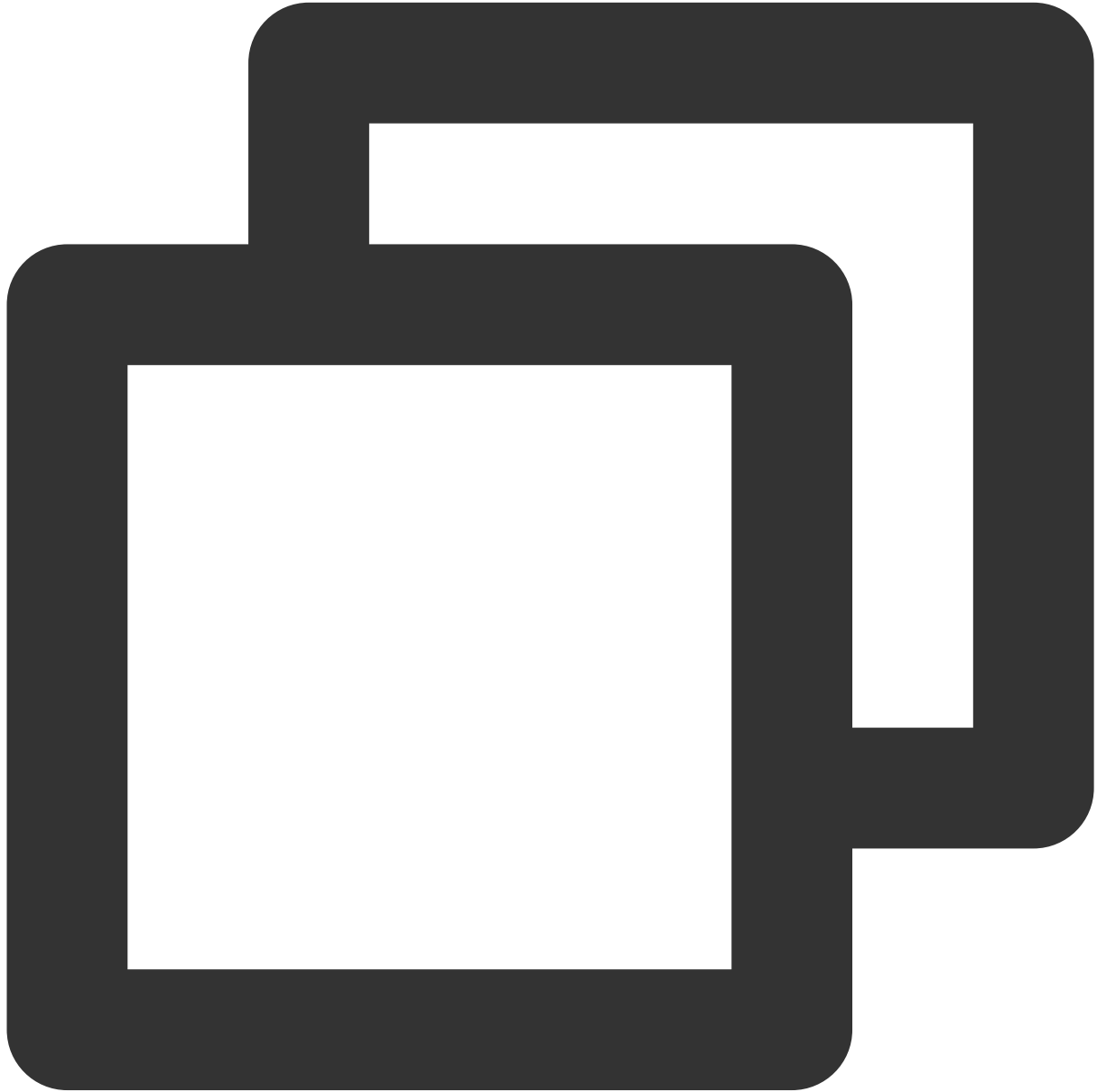


### Integration

## Step 1. Download and import the `TUIVoiceRoom` component

Go to [GitHub](#), clone or download the code, copy the `Android/Source` directory to your project, and complete the following import operations:

Add the code below in `setting.gradle` :



```
include ':Source'
```

Add dependencies on `Source` to the `build.gradle` file in `app` :



```
api project(':Source')
```

Add the TRTC SDK ( `liteavSdk` ) and Chat SDK ( `imSdk` ) dependencies in `build.gradle` in the root directory:



```
ext {  
    liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"  
    imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"  
}
```

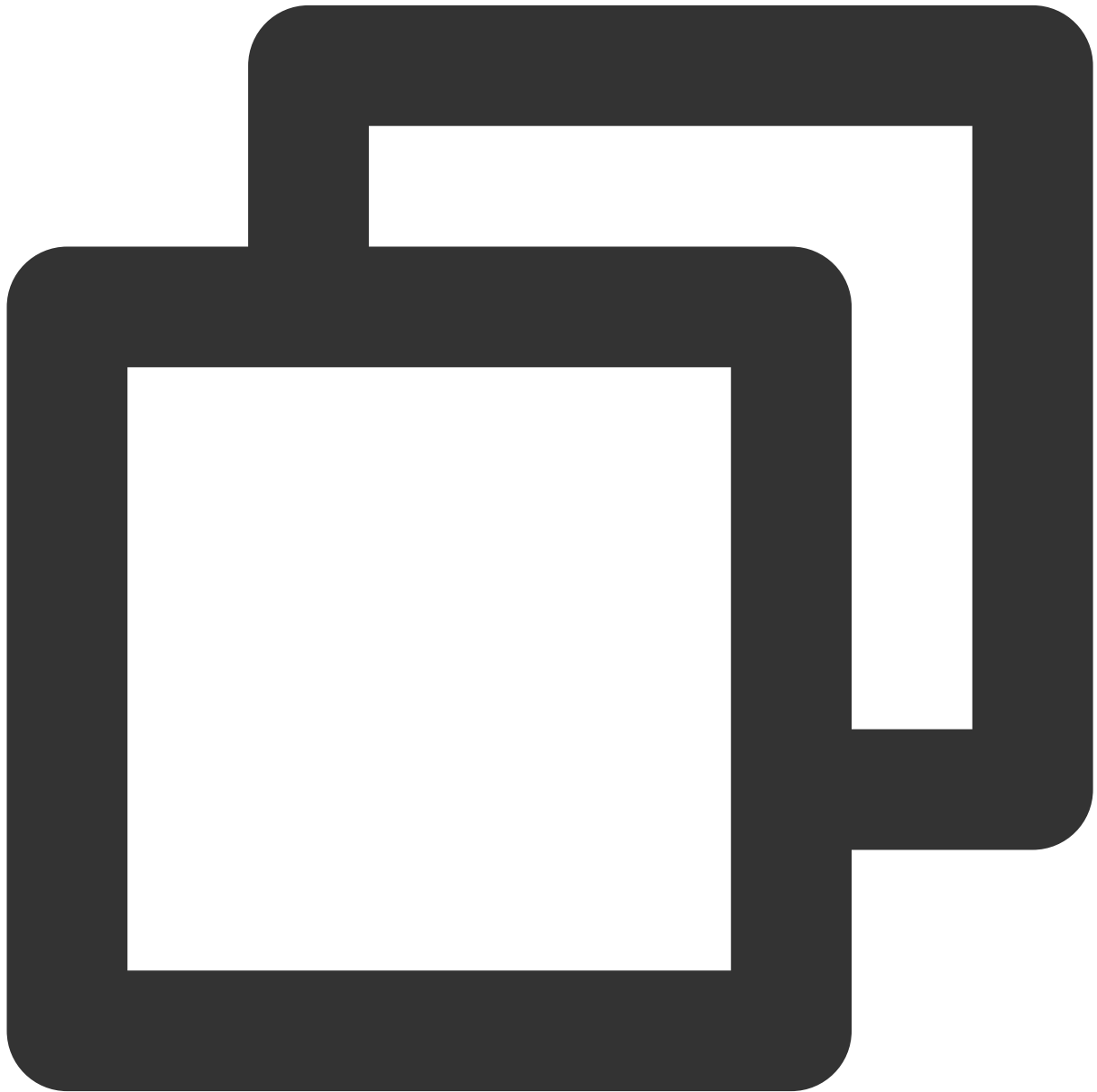
## Step 2. Configure permission requests and obfuscation rules

Configure permission requests for your app in `AndroidManifest.xml`. The SDKs need the following permissions (on Android 6.0 and later, the mic access must be requested at runtime):



```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

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



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

### Step 3. Initialize and log in to the component





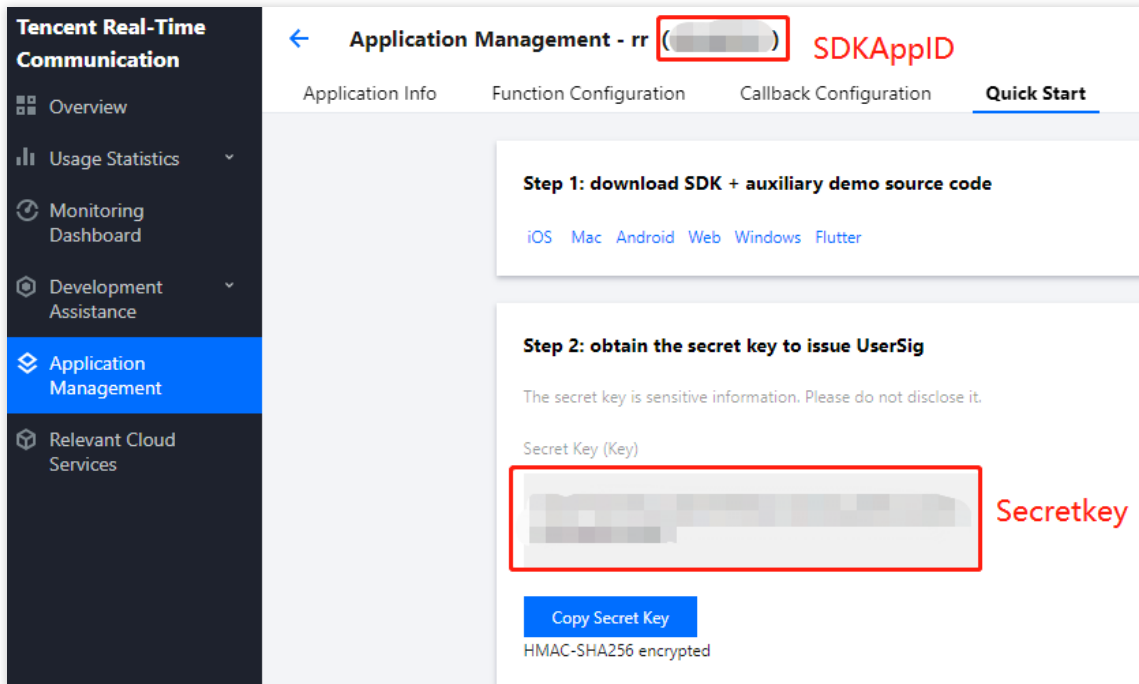
```
// 1. Initialize
TRTCVoiceRoom mTRTCVoiceRoom = TRTCVoiceRoom.sharedInstance(this);
mTRTCVoiceRoom.setDelegate(new TRTCVoiceRoomDelegate() {
});

// 2. Log in
mTRTCVoiceRoom.login(SDKAppID, userId, userSig, new TRTCVoiceRoomCallback.Action
@Override
public void onCallback(int code, String msg) {
    if (code == 0) {
        // Logged in
    }
}
```

```
}  
});
```

### Parameter description:

**SDKAppID:** The **TRTC application ID**. If you haven't activated TRTC, log in to the [TRTC console](#), create a TRTC application, click **Application Info**, and select the **Quick Start** tab to view its `SDKAppID`.



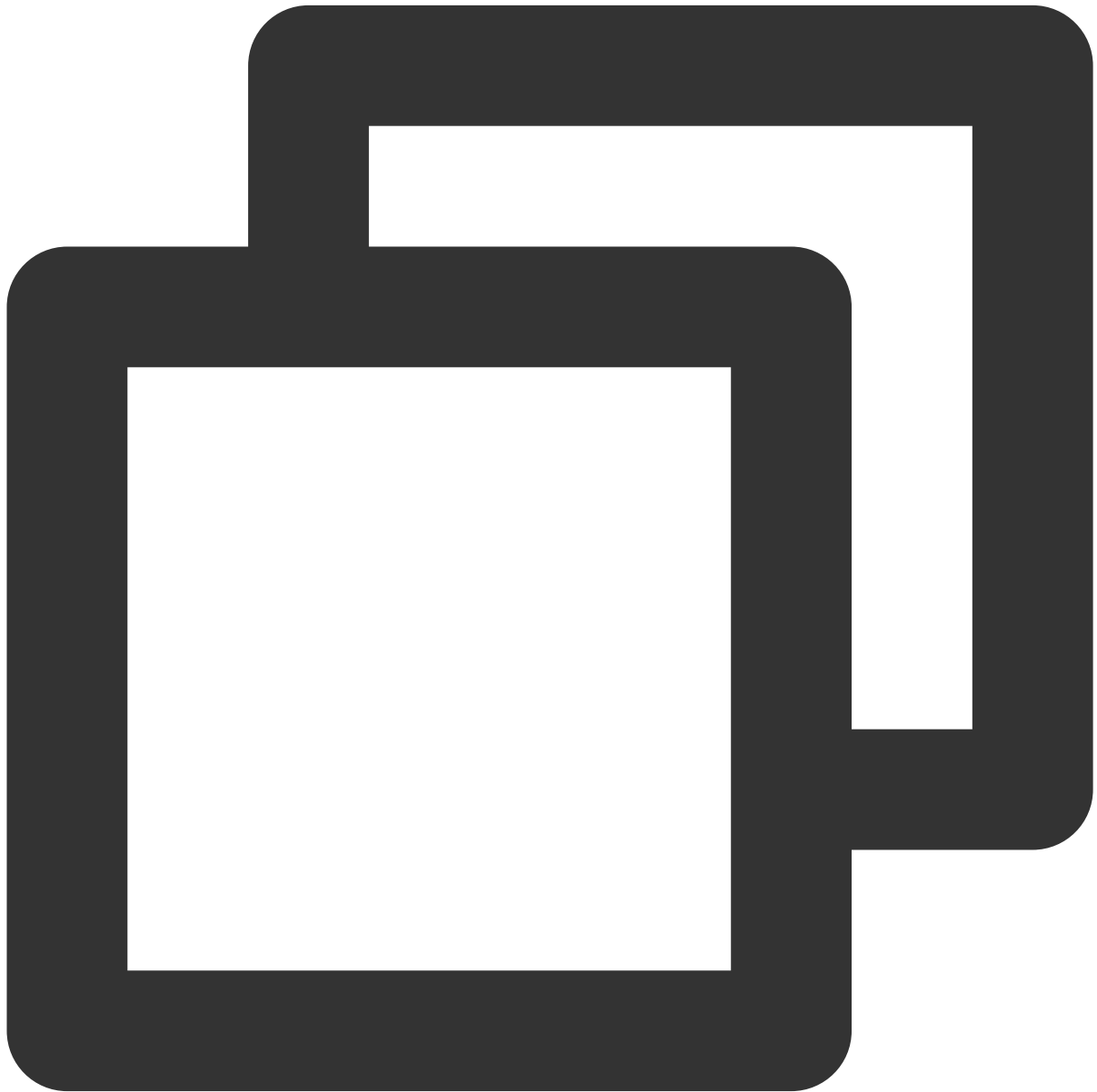
**Secretkey:** The **TRTC application key**. Each secret key corresponds to a `SDKAppID`. You can view your application's secret key on the [Application Management](#) page of the TRTC console.

**userId:** The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (\_). We recommend that you keep it consistent with your user account system.

**userSig:** The security protection signature calculated based on `SDKAppID`, `userId`, and `Secretkey`. You can click [here](#) to directly generate a debugging `userSig` online. For more information, see [UserSig](#).

## Step 4. Implement the audio chat room

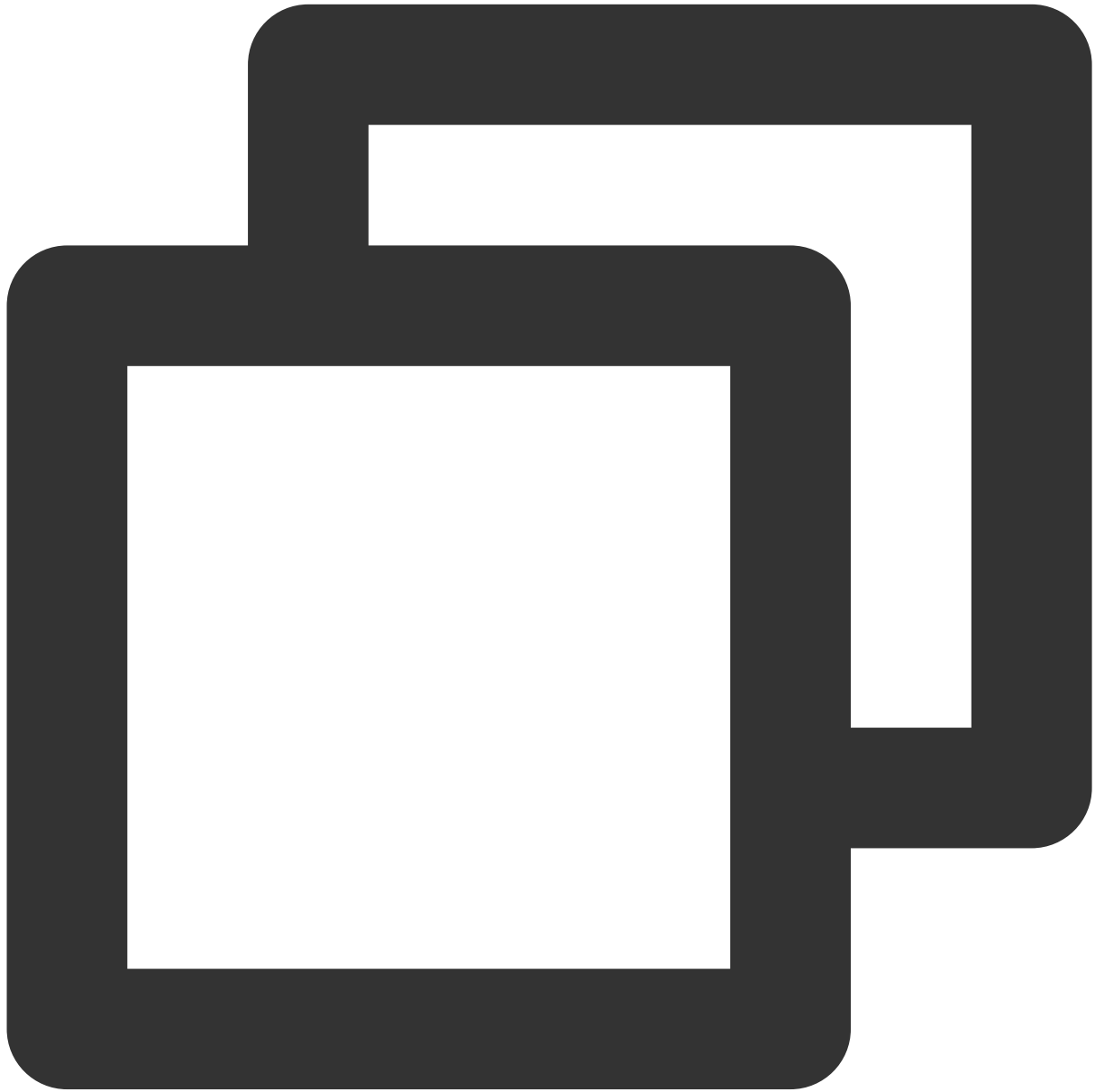
1. The room owner creates an audio chat room through `TRTCVoiceRoom#createRoom`.



```
// 1. The room owner calls an API to create a room
int roomId = 12345; // Room ID
final TRTCVoiceRoomDef.RoomParam roomParam = new TRTCVoiceRoomDef.RoomParam();
roomParam.roomName = "Room name";
roomParam.needRequest = false; // Whether the room owner's permission is required f
roomParam.seatCount = 7; // Number of room seats. In this example, the number is 7.
roomParam.coverUrl = "URL of room cover image";
mTRTCVoiceRoom.createRoom(roomId, roomParam, new TRTCVoiceRoomCallback.ActionCallba
@Override
public void onCallback(int code, String msg) {
    if (code == 0) {
```

```
        // Room created successfully  
    }  
}  
});
```

2. A listener enters the audio chat room through [TRTCVoiceRoom#enterRoom](#).



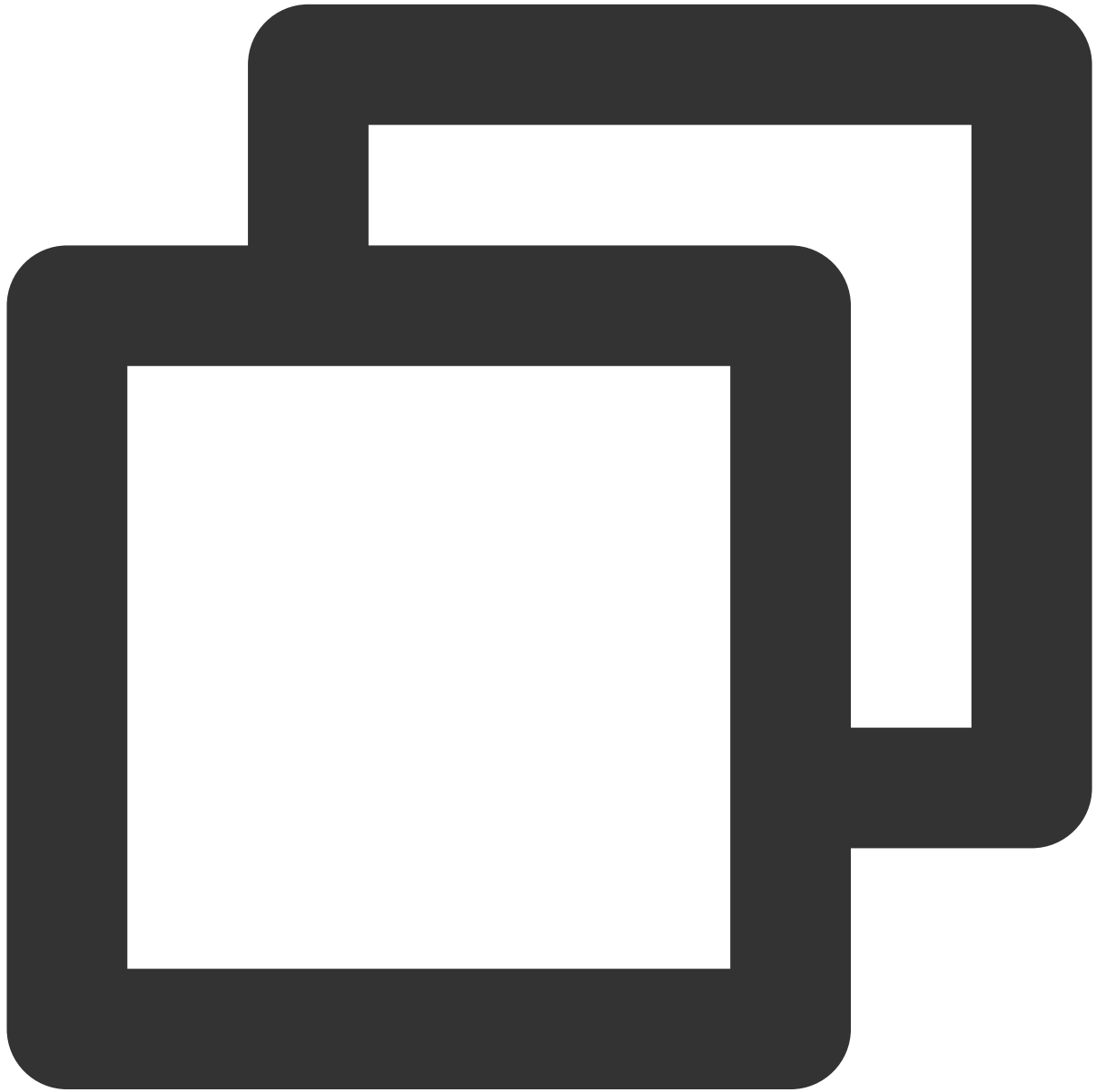
```
// 1. A listener calls an API to enter the room  
mTRTCVoiceRoom.enterRoom(roomId, new TRTCVoiceRoomCallback.ActionCallback() {  
    @Override  
    public void onCallback(int code, String msg) {  
        if (code == 0) {
```

```
        // Entered room successfully
    }

}

});
```

3. A listener mics on through [TRTCVoiceRoom#enterSeat](#).



```
// 1. A listener calls an API to mic on
int seatIndex = 2; // Seat index
mTRTCVoiceRoom.enterSeat(seatIndex, new TRTCVoiceRoomCallback.ActionCallback() {
    @Override
    public void onCallback(int code, String msg) {
```

```
        if (code == 0) {  
            // Operation succeeded  
        }  
    }  
});  
  
// 2. The `onSeatListChange` callback is received, and the seat list is refreshed  
@Override  
public void onSeatListChange(final List<TRTCVoiceRoomDef.SeatInfo> seatInfoList) {  
}
```

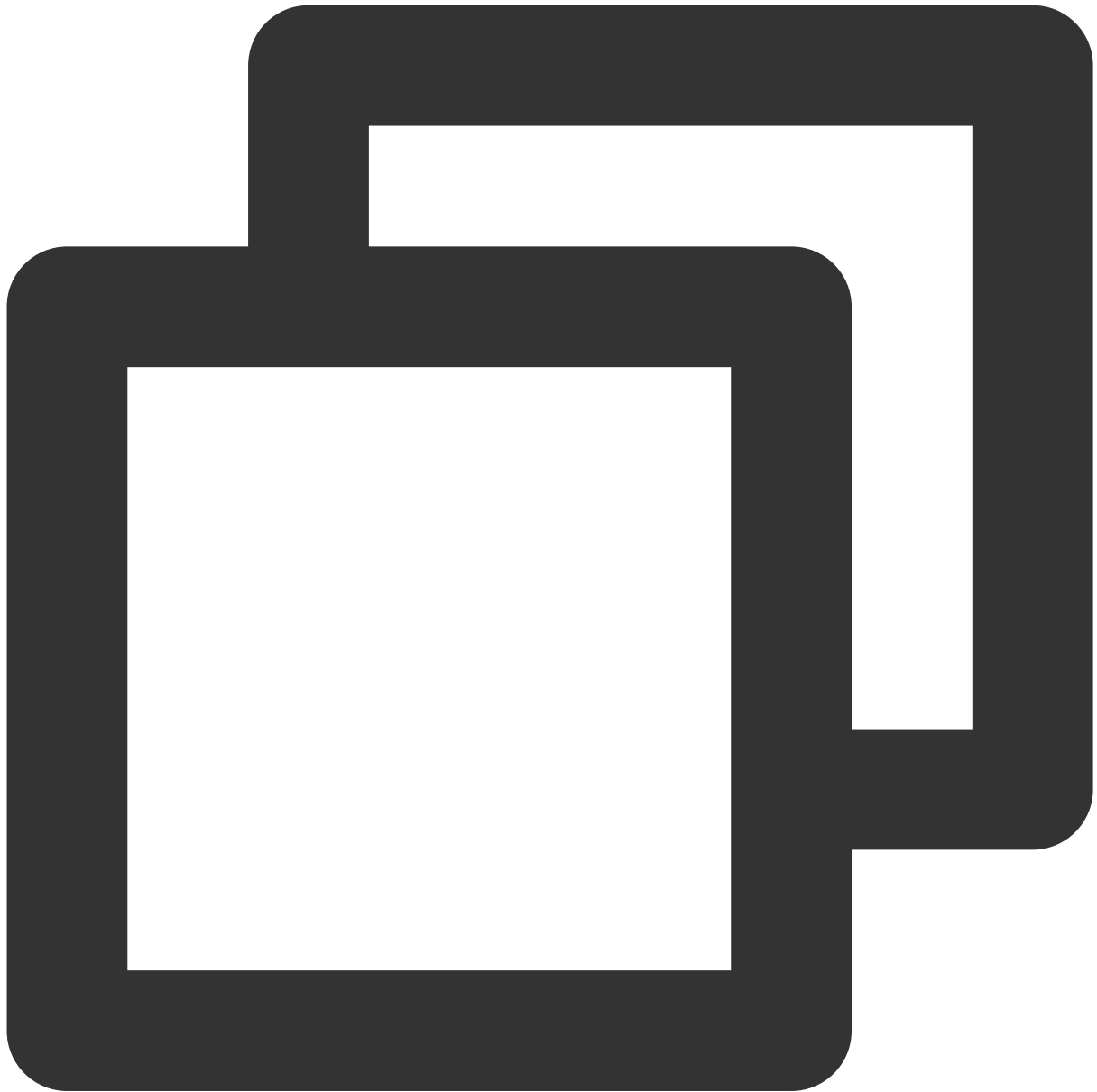
4. The room owner makes a listener a speaker through [TRTCVoiceRoom#pickSeat](#).



```
// 1. The room owner makes a listener a speaker
int seatIndex = 2; // Seat index
String userId = "123"; // The ID of the user who will become a speaker
mTRTCVoiceRoom.pickSeat(1, userId, new TRTCVoiceRoomCallback.ActionCallback() {
    @Override
    public void onCallback(int code, String msg) {
        if (code == 0) {
            // Operation succeeded
        }
    }
});
```

```
// 2. The `onSeatListChange` callback is received, and the seat list is refreshed
@Override
public void onSeatListChange(final List<TRTCVoiceRoomDef.SeatInfo> seatInfoList) {
}
```

5. A listener requests to speak through [TRTCVoiceRoom#sendInvitation](#).



```
// Listener
// 1. A listener calls an API to request to speak
String seatIndex = "1"; // Seat index
String userId = "123"; // User ID
```

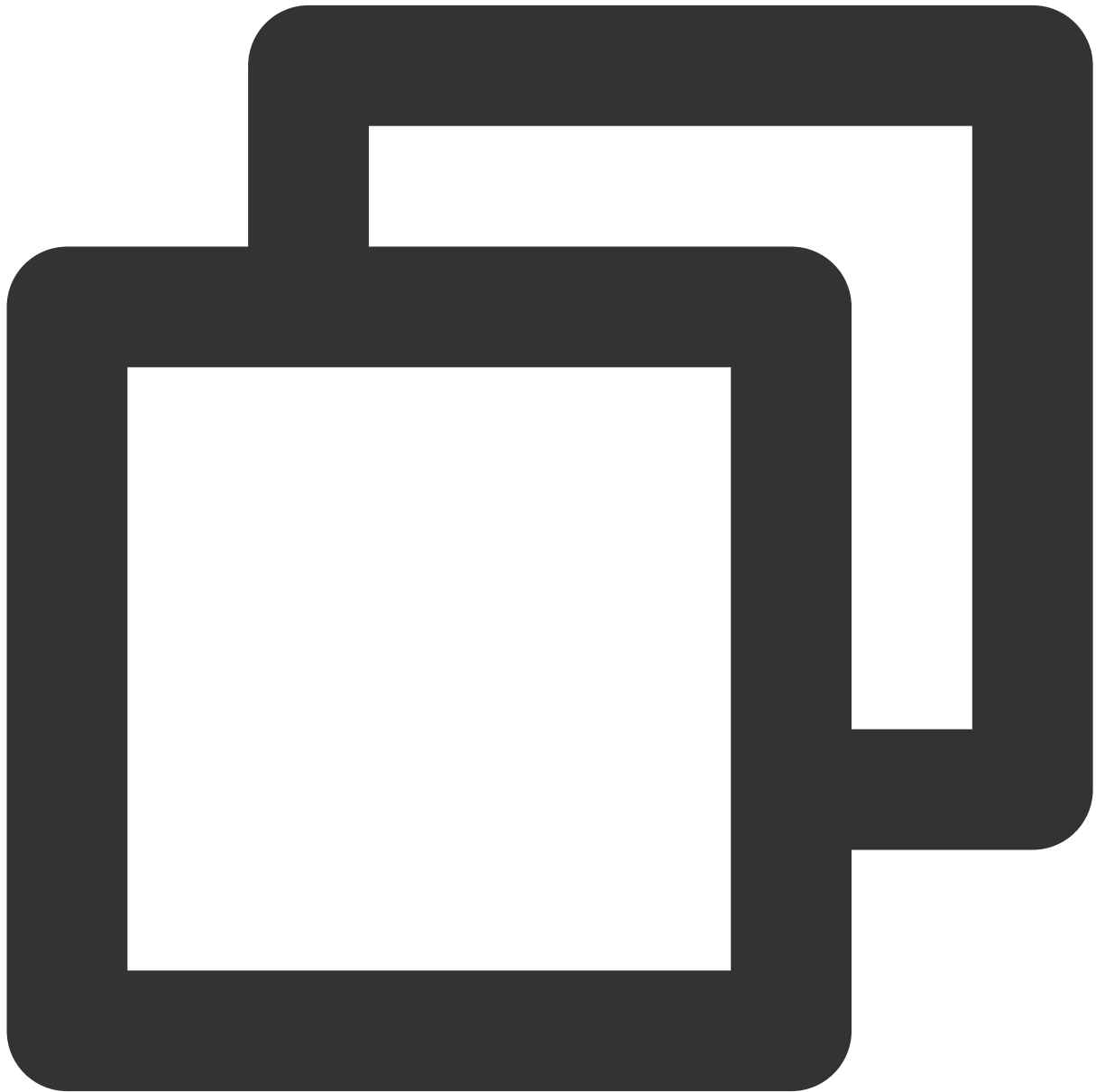


```
String inviteId = mTRTCVoiceRoom.sendInvitation("takeSeat", userId, seatIndex, null);

// 2. Place the user in the seat after the invitation is accepted
@Override
public void onInviteeAccepted(String id, String invitee) {
    if(id.equals(inviteId)) {
        mTRTCVoiceRoom.enterSeat(index, null);
    }
}

// Room owner
// 1. The room owner receives the request
@Override
public void onReceiveNewInvitation(final String id, String inviter, String cmd, final int index) {
    if (cmd.equals("takeSeat")) {
        // 2. The room owner accepts the request
        mTRTCVoiceRoom.acceptInvitation(id, null);
    }
}
```

6. The room owner invites a listener to speak through [TRTCVoiceRoom#sendInvitation](#).



```
// Room owner
// 1. The room owner calls an API to invite a listener to speak
String seatIndex = "1"; // Seat index
String userId = "123"; // User ID
String inviteId = mTRTCVoiceRoom.sendInvitation("pickSeat", userId, seatIndex, null);

// 2. Place the user in the seat after the invitation is accepted
@Override
public void onInviteeAccepted(String id, String invitee) {
    if(id.equals(inviteId)) {
        mTRTCVoiceRoom.pickSeat(index, null);
    }
}
```

```
    }  
}  
  
// Listener  
// 1. The listener receives the invitation  
@Override  
public void onReceiveNewInvitation(final String id, String inviter, String cmd, final  
    if (cmd.equals("pickSeat")) {  
        // 2. The listener accepts the invitation  
        mTRTCVoiceRoom.acceptInvitation(id, null);  
    }  
}
```

7. Implement text chat through [TRTCVoiceRoom#sendRoomTextMsg](#).



```
// Sender: Sends a text chat message
mTRTCVoiceRoom.sendRoomTextMsg("Hello World!", null);
// Receiver: Listens for text chat messages
mTRTCVoiceRoom.setDelegate(new TRTCVoiceRoomDelegate() {
    @Override
    public void onRecvRoomTextMsg(String message, TRTCVoiceRoomDef.UserInfo userInfo)
        Log.d(TAG, "Received a message from" + userInfo.userName + ": " + message);
    }
});
```

#### 8. Implement on-screen commenting through [TRTCVoiceRoom#sendRoomCustomMsg](#).



```
// A sender can customize CMD to distinguish on-screen comments and likes.
// For example, use "CMD_DANMU" to indicate on-screen comments and "CMD_LIKE" to in
mTRTCVoiceRoom.sendRoomCustomMsg("CMD_DANMU", "Hello world", null);
mTRTCVoiceRoom.sendRoomCustomMsg("CMD_LIKE", "", null);
// Receiver: Listens for custom messages
mTRTCVoiceRoom.setDelegate(new TRTCVoiceRoomDelegate() {
    @Override
    public void onRecvRoomCustomMsg(String cmd, String message, TRTCVoiceRoomDef.UserInfo
        if ("CMD_DANMU".equals(cmd)) {
            // An on-screen comment is received
            Log.d(TAG, "Received an on-screen comment from" + userInfo.userName + ": "
```

```
    } else if ("CMD_LIKE".equals(cmd)) {  
        // A like is received  
        Log.d(TAG, userInfo.userName + "liked you.");  
    }  
}  
});
```

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [colleenyu@tencent.com](mailto:colleenyu@tencent.com).

# Integrating TUIVoiceRoom (iOS)

Last updated : 2023-09-25 10:51:43

## Overview

`TUIVoiceRoom` is an open-source audio/video UI component. After integrating it into your project, you can make your application support the group audio chat scenario simply by writing a few lines of code. It also supports the [Android](#) platform. Its basic features are as shown below:

### Note

All TUIKit components are based on two PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For billing details of Chat, see [Pricing](#).



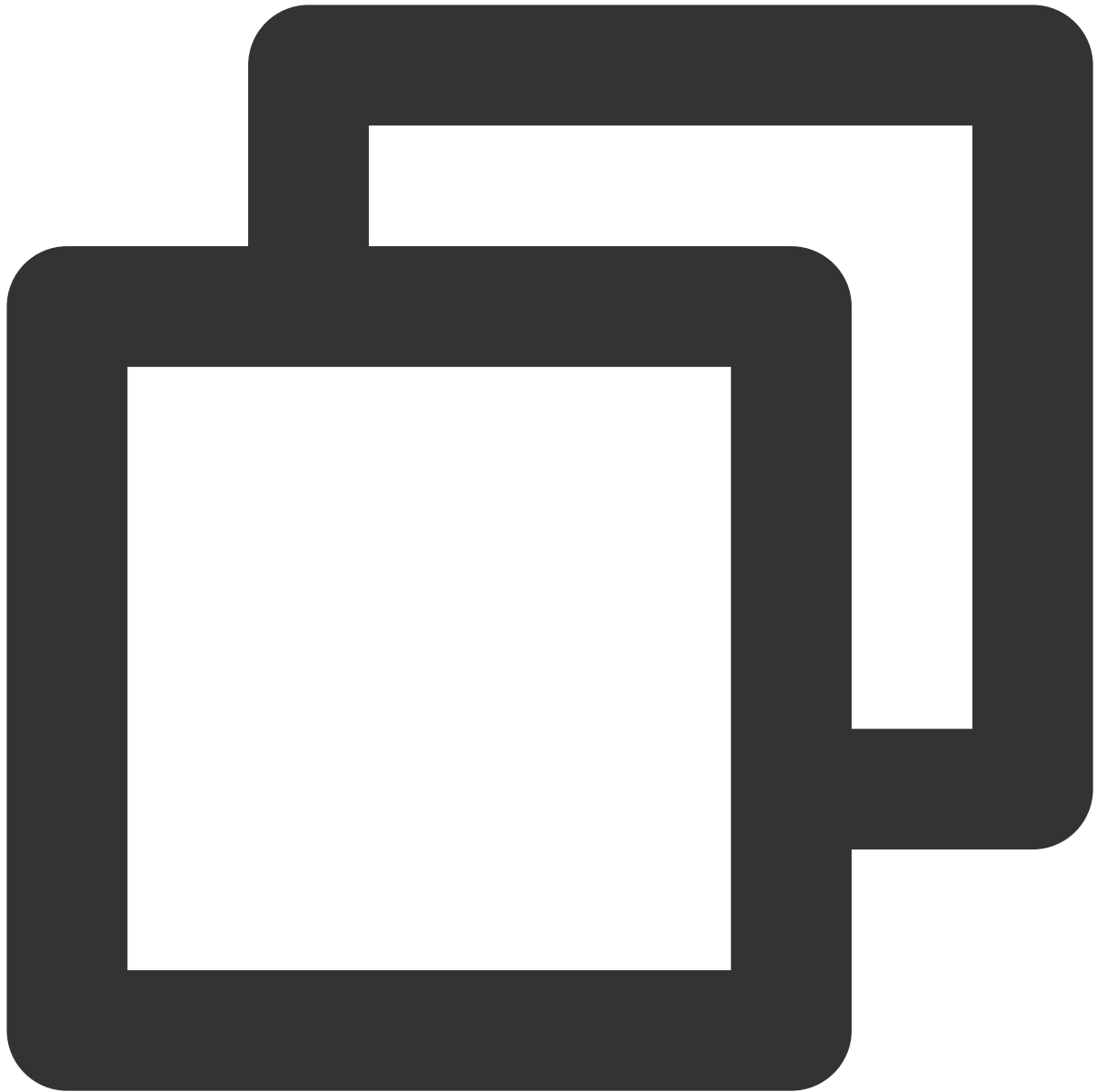
## Integration

### Step 1. Download and import the `TUIVoiceRoom` component

Create the `TUIVoiceRoom` folder at the same level as the `Podfile` in your Xcode project, copy [TXAppBasic](#), [Resources](#), [Source](#), and [TUIVoiceRoom.podspec](#) files from the [iOS](#) directory in the [GitHub repository](#) to the folder, and complete the following import operations:

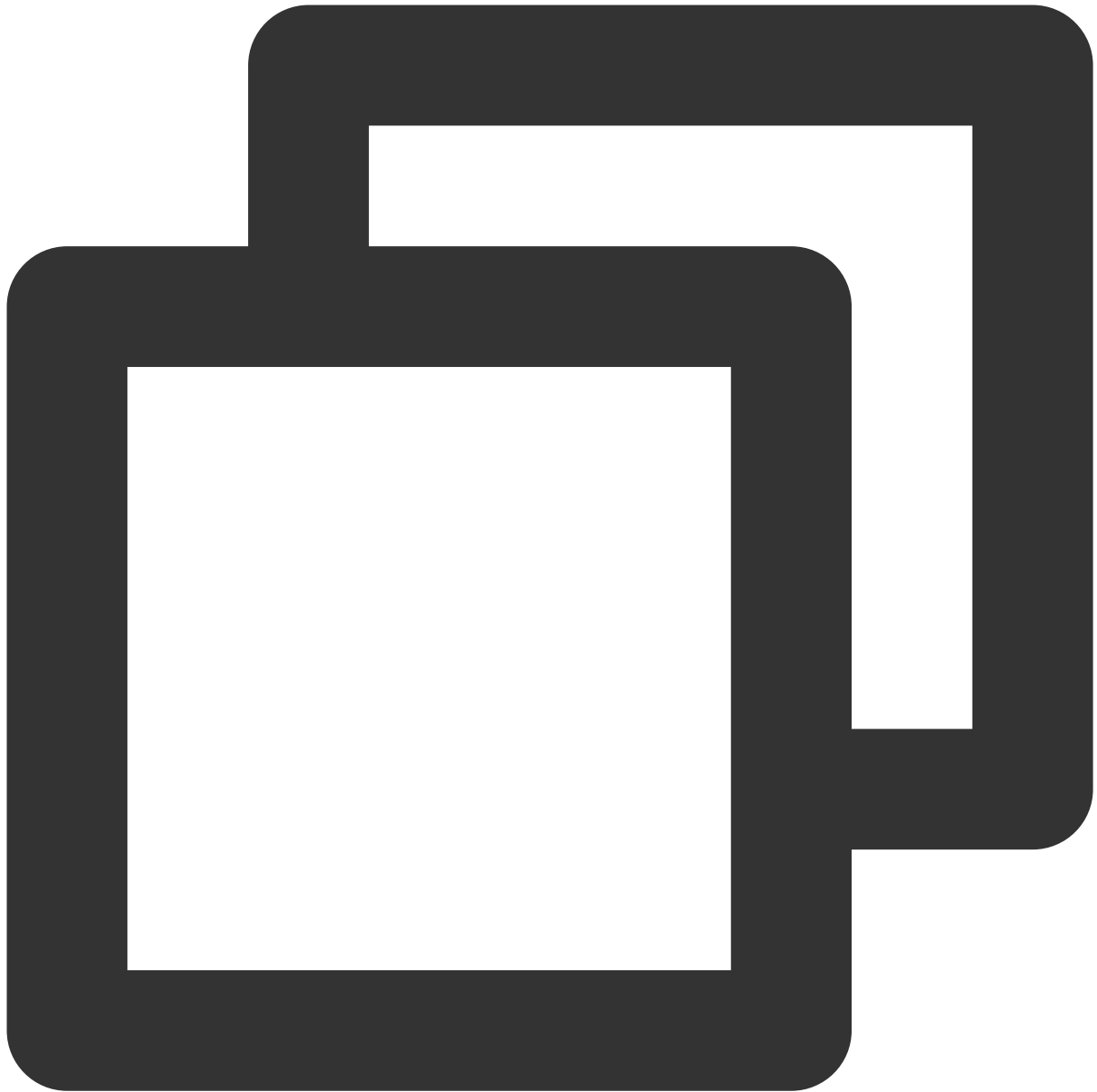
Open the project's `Podfile` and import `TUIVocieRoom.podspec` as follows:





```
# `path` is the path of `TXAppBasic.podspec` relative to the `Podfile`  
pod 'TXAppBasic', :path => "TUIVoiceRoom/TXAppBasic/"  
# `path` is the path of `TUIVoiceRoom.podspec` relative to the `Podfile`  
pod 'TUIVoiceRoom', :path => "TUIVoiceRoom/", :subspecs => ["TRTC"]
```

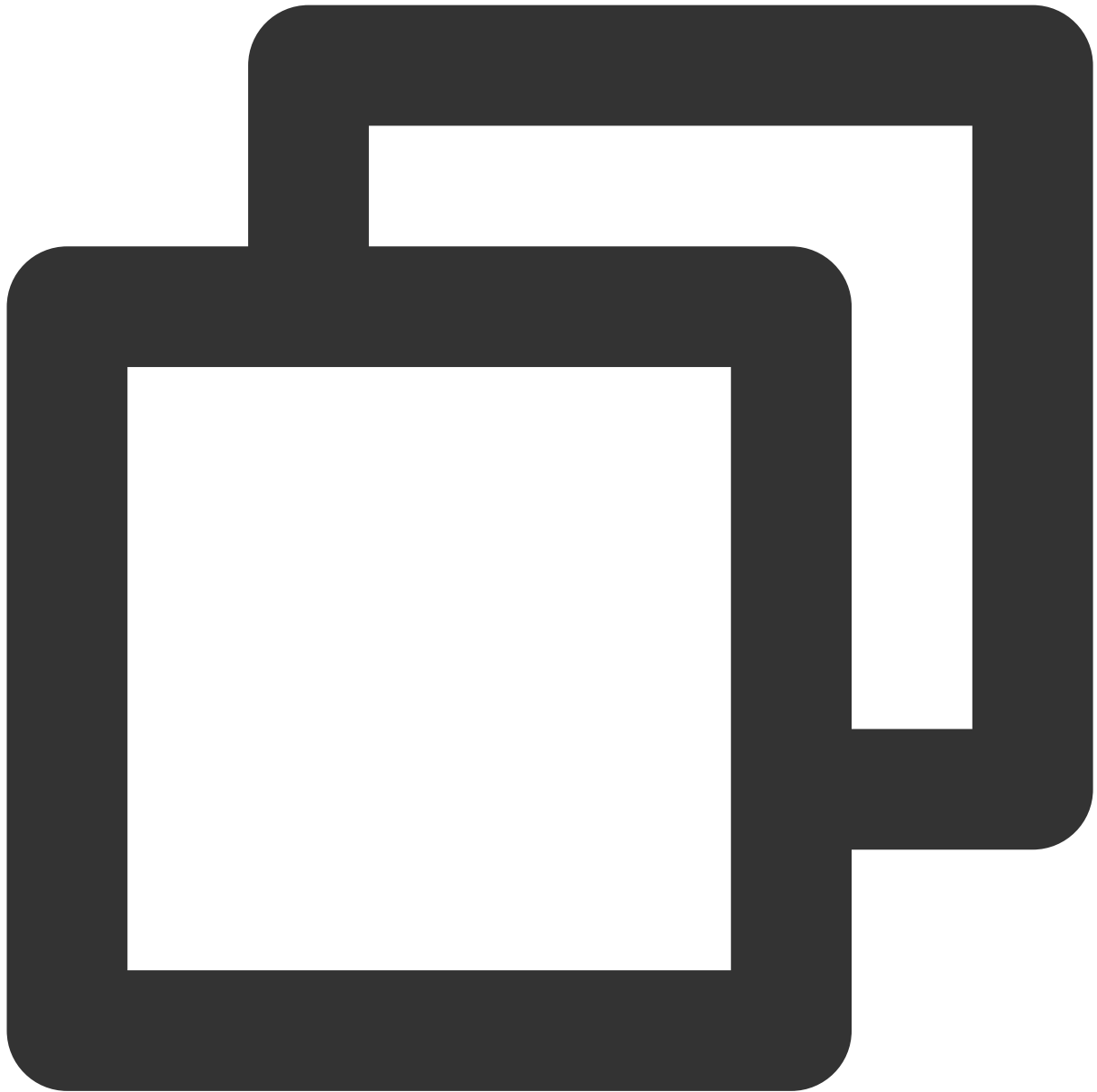
Open Terminal, enter the directory of `Podfile` , and run `pod install` .



```
pod install
```

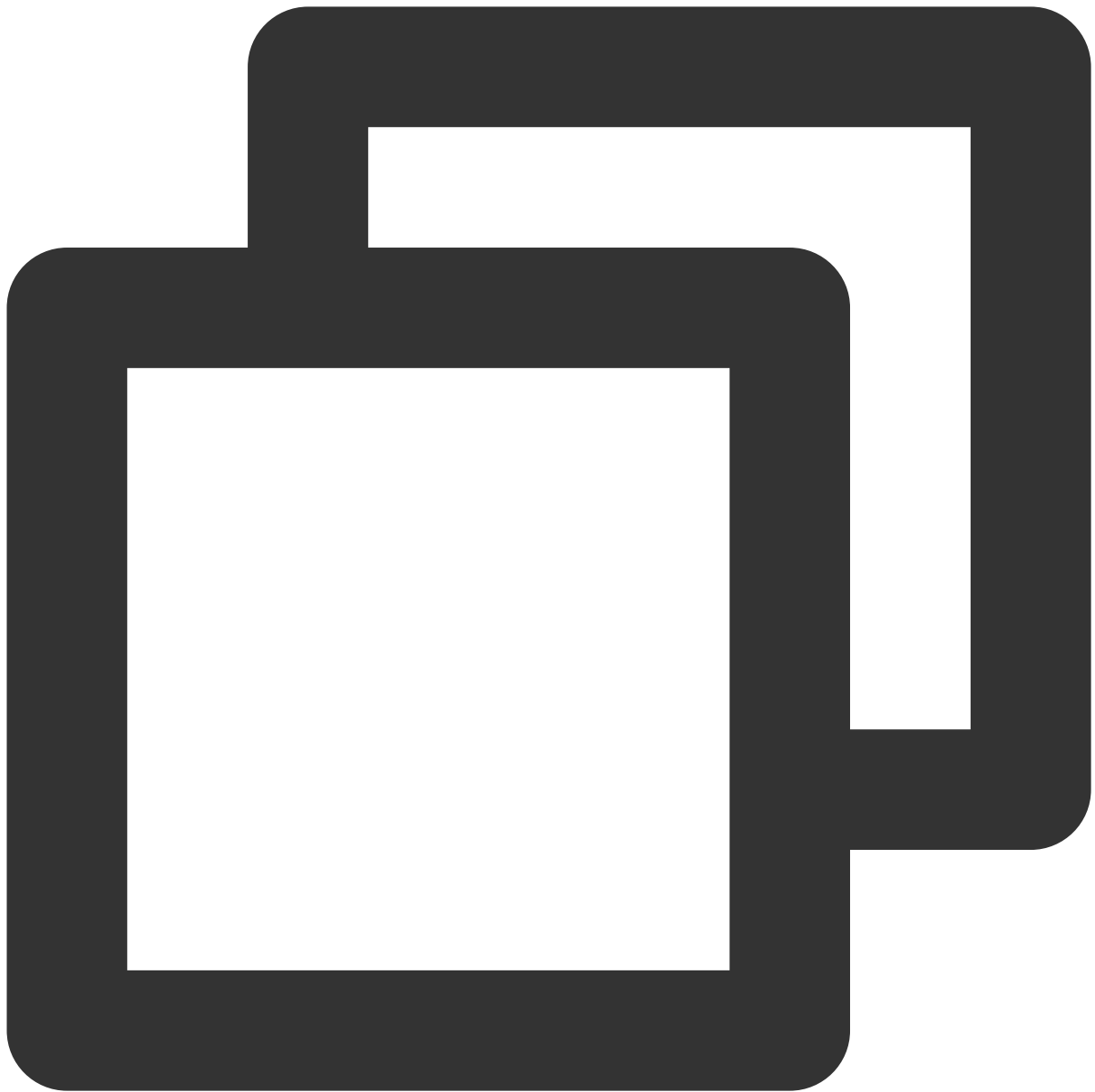
## Step 2. Configure permission requests and obfuscation rules

In `info.plist`, add `Privacy > Microphone Usage Description` to request mic access.



```
<key>NSMicrophoneUsageDescription</key>  
<string>`VoiceRoomApp` needs to access your mic to be able to shoot videos with aud
```

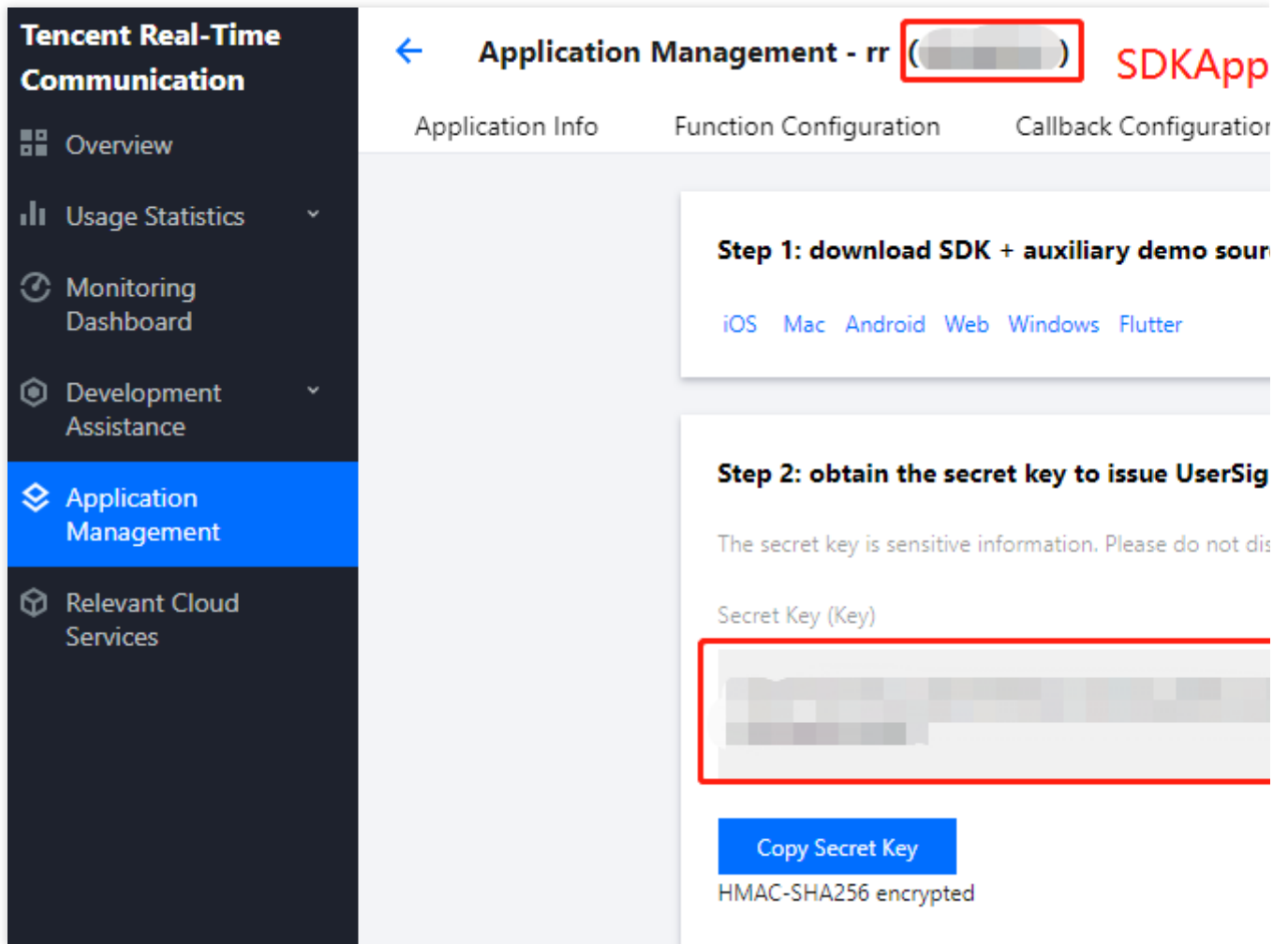
### Step 3. Initialize and log in to the component



```
// Initialize TUIKit
let mTRTCVoiceRoom = TRTCVoiceRoom.shared()
// Log in
mTRTCVoiceRoom.login(sdkAppID: SDKAppID, userId: userId, userSig: userSig) { code,
    if code == 0 {
        // Logged in
    }
}
```

**Parameter description:**

**SDKAppID:** The **TRTC application ID**. If you haven't activated TRTC, log in to the [TRTC console](#), create a TRTC application, click **Application Info**, and select the **Quick Start** tab to view its `SDKAppID` .



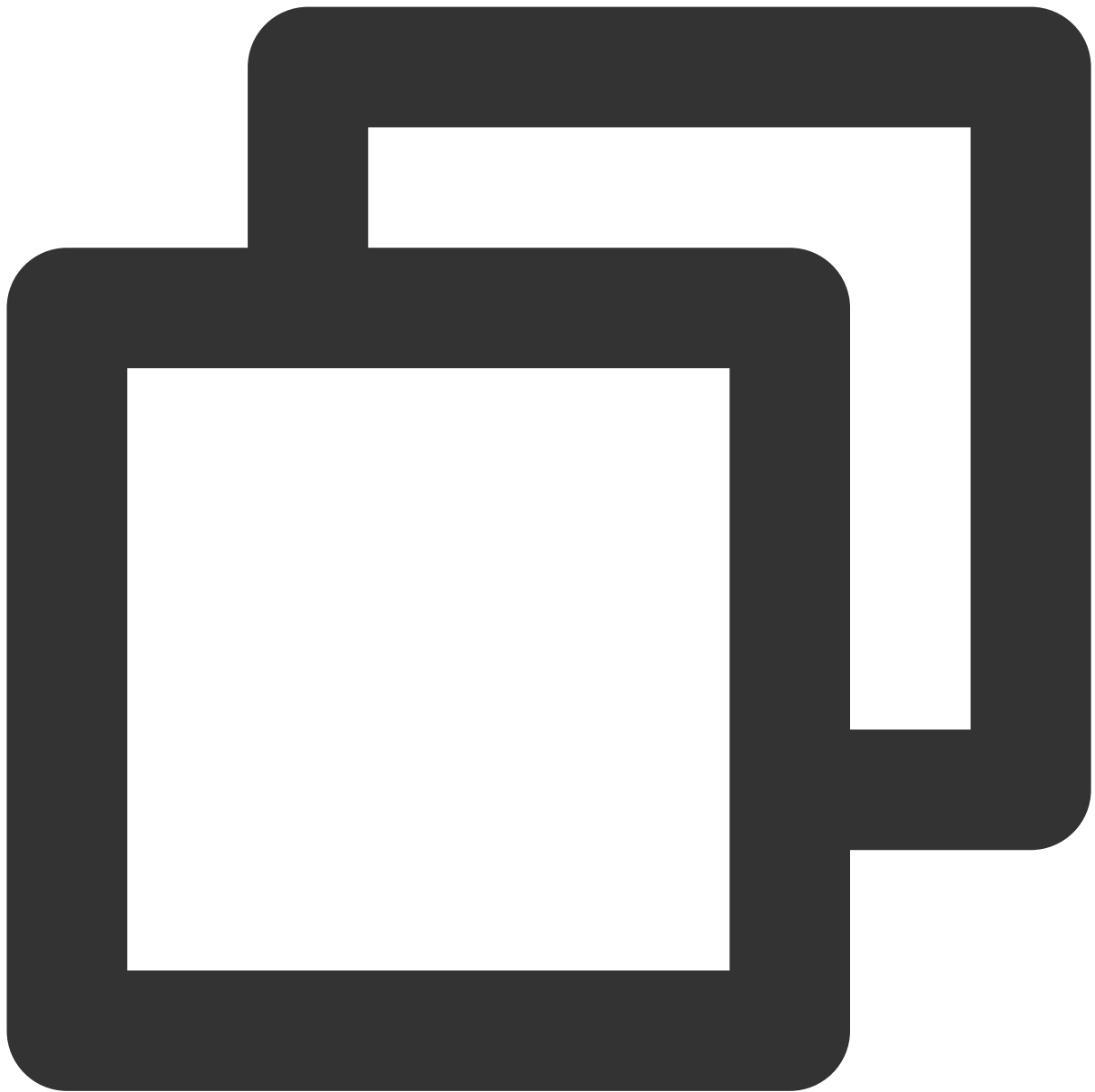
**Secretkey:** The **TRTC application key**. Each secret key corresponds to a `SDKAppID` . You can view your application's secret key on the [Application Management](#) page of the TRTC console.

**userId:** The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (\_). We recommend that you keep it consistent with your user account system.

**UserSig:** The security signature calculated based on `SDKAppID` , `userId` , and `Secretkey` . You can click [here](#) to quickly generate a `UserSig` for testing. For more information, see [UserSig](#).

## Step 4. Implement the audio chat room

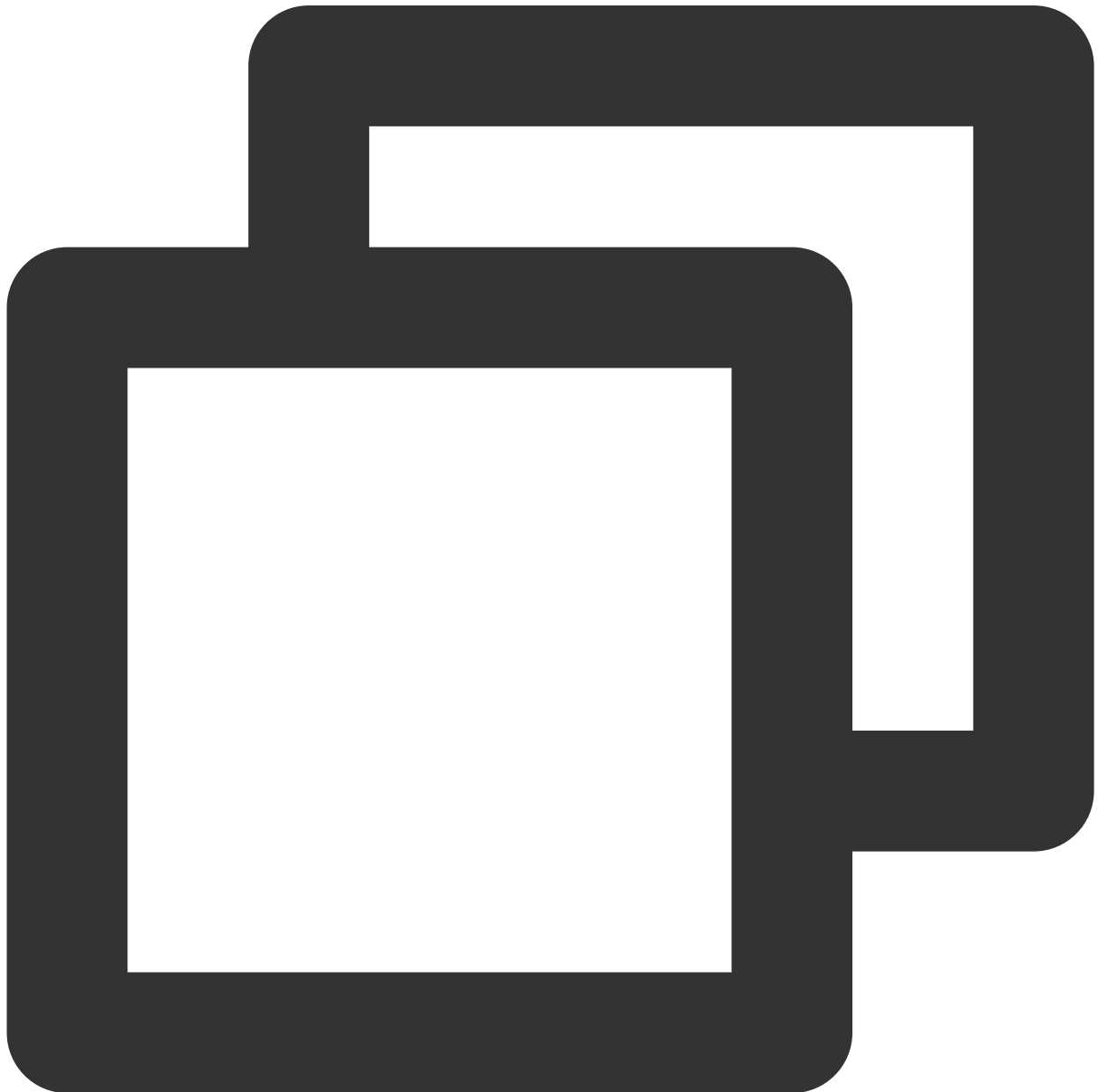
1. The room owner creates an audio chat room through [TRTCVoiceRoom#createRoom](#).



```
// Initialize the audio chat room parameters
let roomParam = VoiceRoomParam()
roomParam.roomName = "Room name"
roomParam.needRequest = false // Whether the room owner's permission is required fo
roomParam.coverUrl = "URL of room cover image"
roomParam.seatCount = 7 // Number of room seats. In this example, the number is 7.
roomParam.seatInfoList = []
// Initialize the seat information
for _ in 0..
```

```
}  
// Create a room  
mTRTCVoiceRoom.createRoom(roomID: yourRoomID, roomParam: roomParam) { (code, message)  
    if code == 0 {  
        // Group created successfully  
    }  
}
```

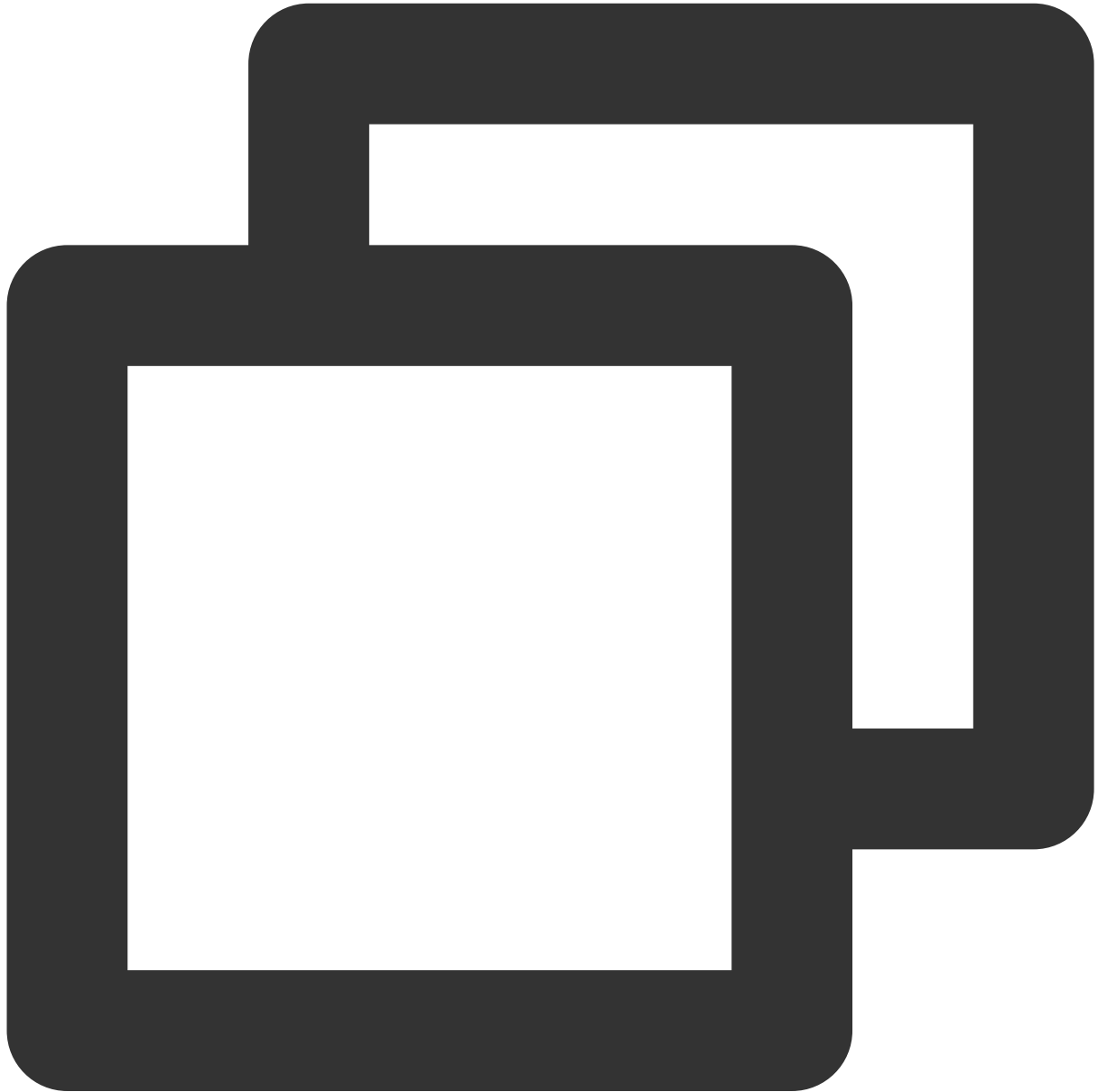
2. A listener enters the audio chat room through [TRTCVoiceRoom#enterRoom](#).



```
// 1. A listener calls an API to enter the room  
mTRTCVoiceRoom.enterRoom(roomID: roomID) { (code, message) in
```

```
// Callback of the room entry result
if code == 0 {
    // Entered room successfully
}
}
```

### 3. A listener mics on through `TRTCVoiceRoom#enterSeat`.



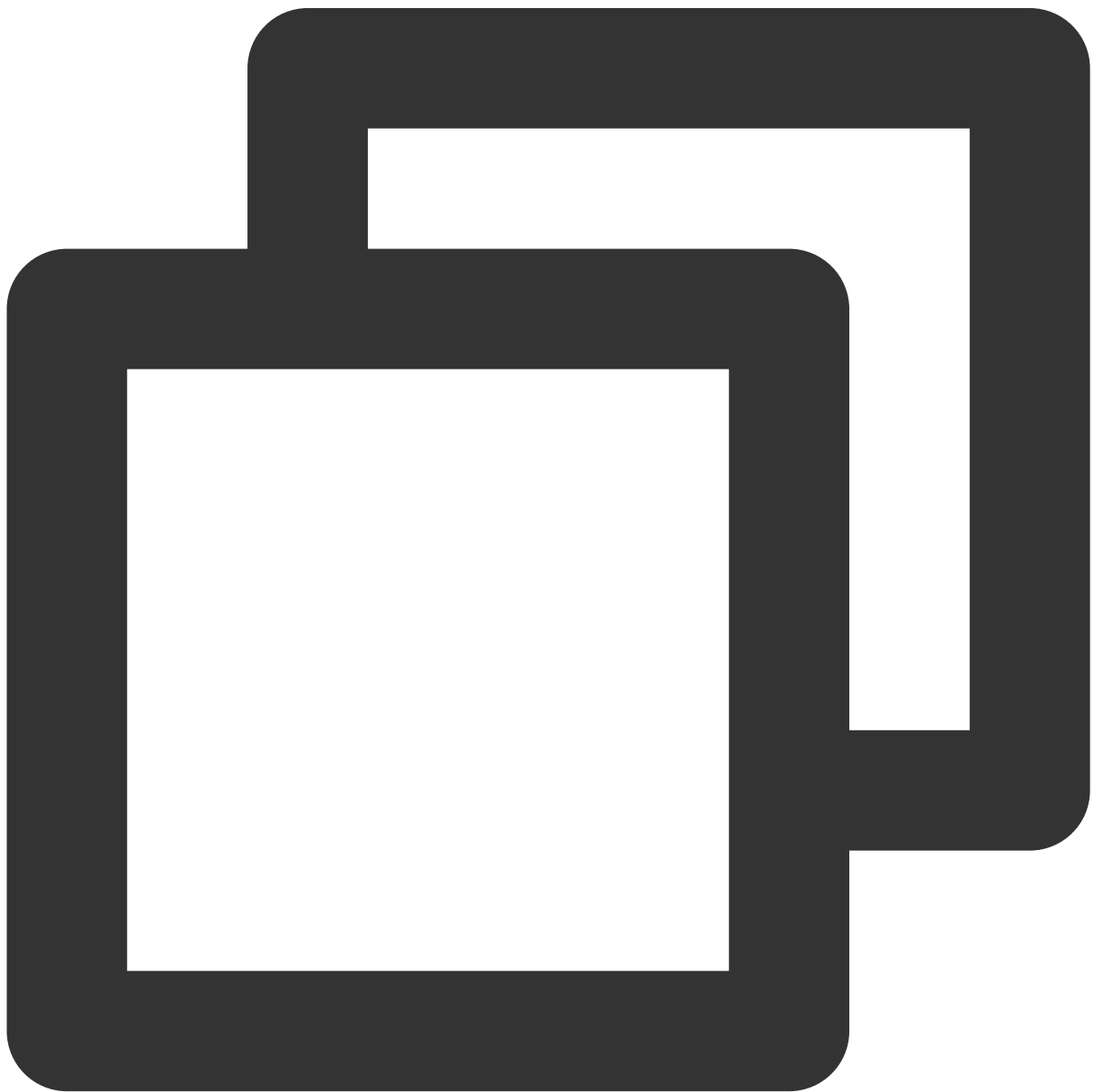
```
// 1. A listener calls an API to mic on
let seatIndex = 2; // Seat index
mTRTCVoiceRoom.enterSeat(seatIndex: 2) { (code, message) in
    if code == 0 {
```



```
// Mic turned on successfully
}
}

// 2. The `onSeatListChange` callback is received, and the seat list is refreshed
@Override
func onSeatListChange(seatInfoList: [VoiceRoomSeatInfo]) {
    // Refreshed seat list
}
```

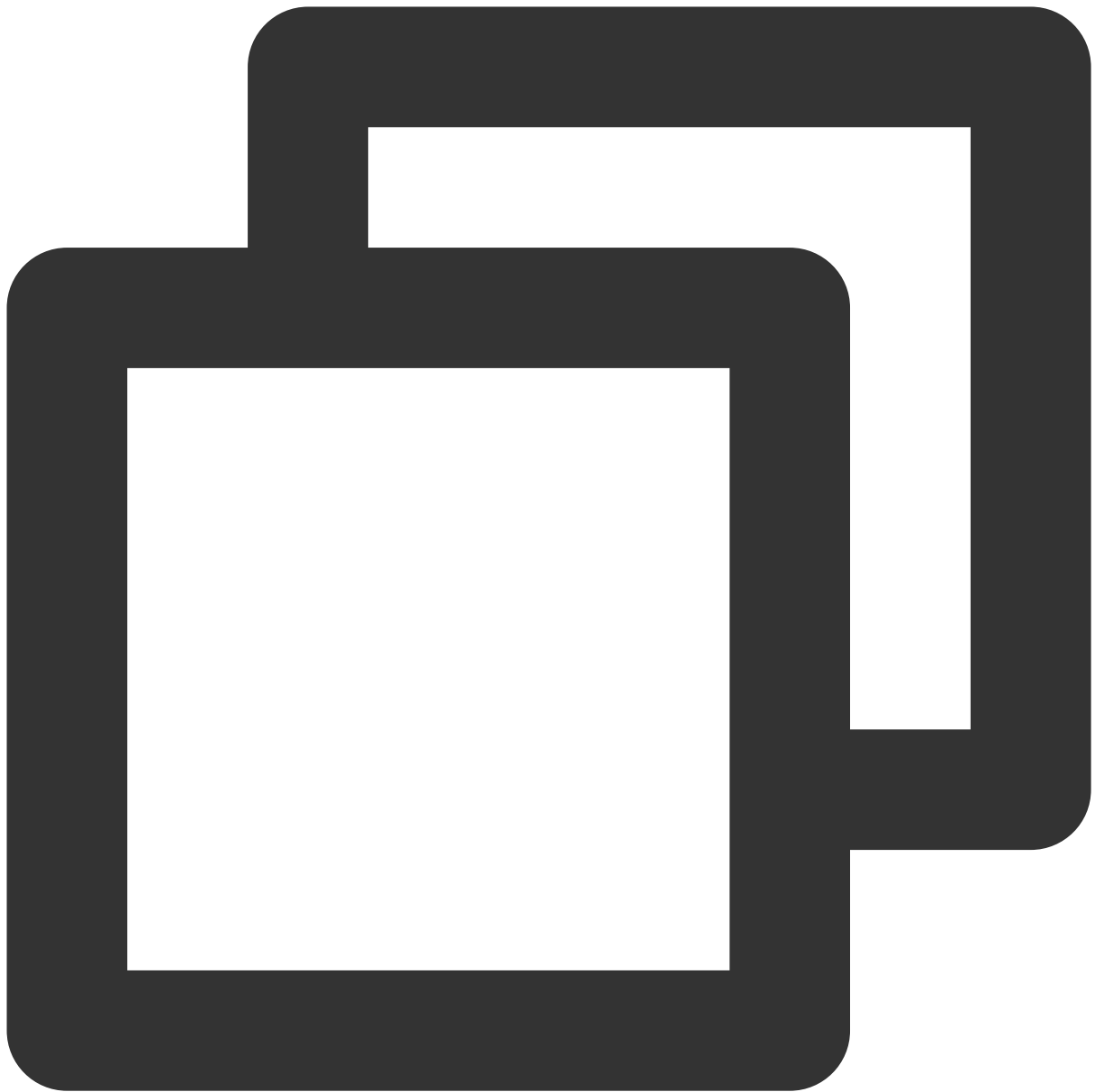
4. The room owner makes a listener speaker through [TRTCVoiceRoom#pickSeat](#).



```
// 1. The room owner makes a listener a speaker
let seatIndex = 2; // Seat index
let userId = "123"; // ID of the user to speak
mTRTCVoiceRoom.pickSeat(seatIndex: 1, userId: "123") { (code, message) in
    if code == 0 {
    }
}

// 2. The `onSeatListChange` callback is received, and the seat list is refreshed
func onSeatListChange(seatInfoList: [VoiceRoomSeatInfo]) {
    // Refreshed seat list
}
```

**5. A listener requests to speak through [TRTCVoiceRoom#sendInvitation](#).**



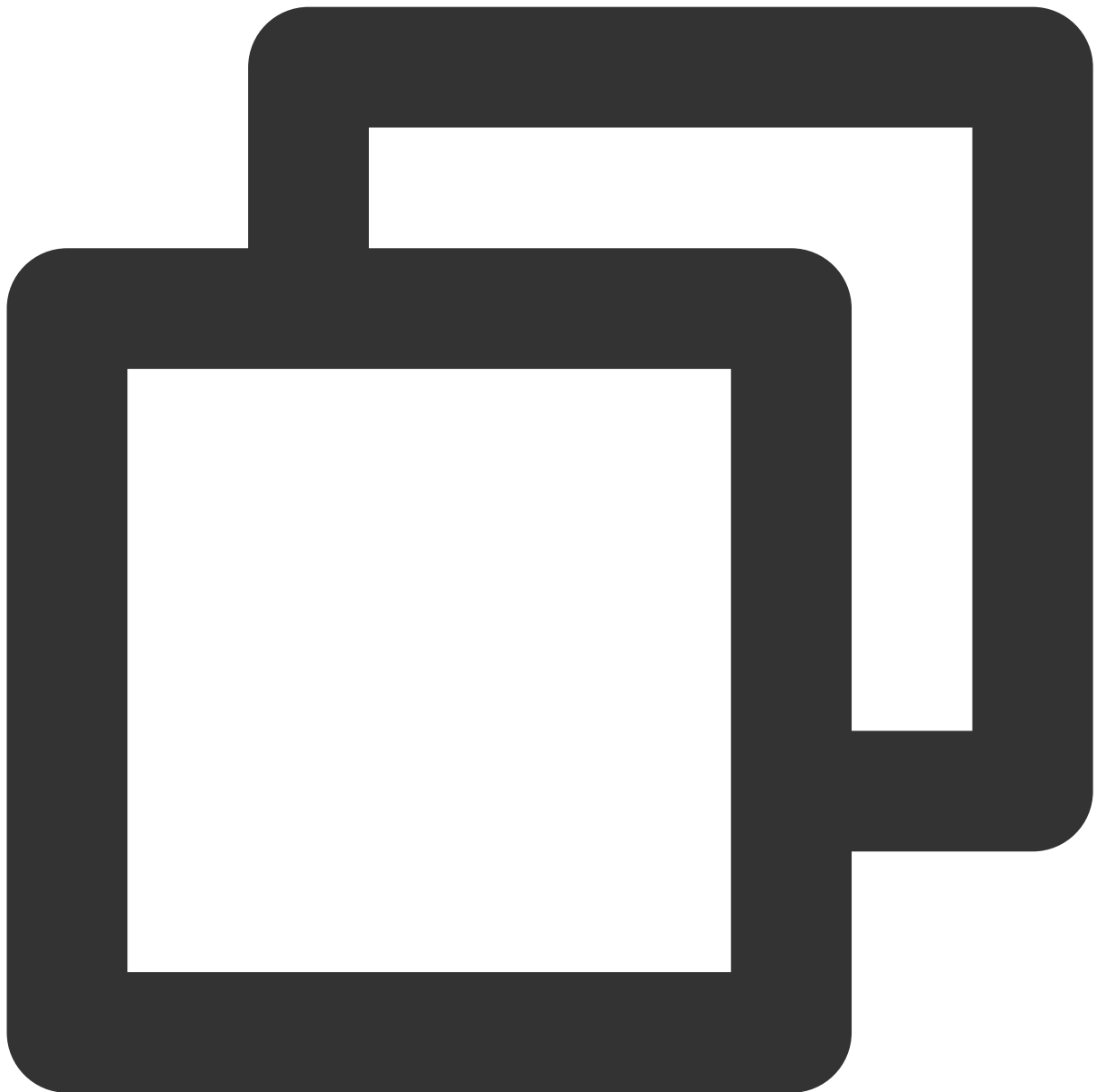
```
// Listener
// 1. A listener calls an API to request to speak
let seatIndex = "1"; // Seat index
let userId = "123"; // User ID
let inviteId = mTRTCVoiceRoom.sendInvitation(cmd: "takeSeat", userId: ownerId,
// Callback of the result
}

// 2. Place the user in the seat after the invitation is accepted
func onInviteeAccepted(identifier: String, invitee: String) {
    if identifier == selfID {
```

```
        self.mTRTCVoiceRoom.enterSeat(seatIndex: ) { (code, message) in
            // Callback of the result
        }
    }
}

// Room owner
// 1. The room owner receives the request
func onReceiveNewInvitation(identifier: String, inviter: String, cmd: String, conte
    if cmd == "takeSeat" {
        // 2. The room owner accepts the request
        self.mTRTCVoiceRoom.acceptInvitation(identifier: identifier, callback: nil)
    }
}
```

6. The room owner invites a listener to speak through [TRTCVoiceRoom#sendInvitation](#).

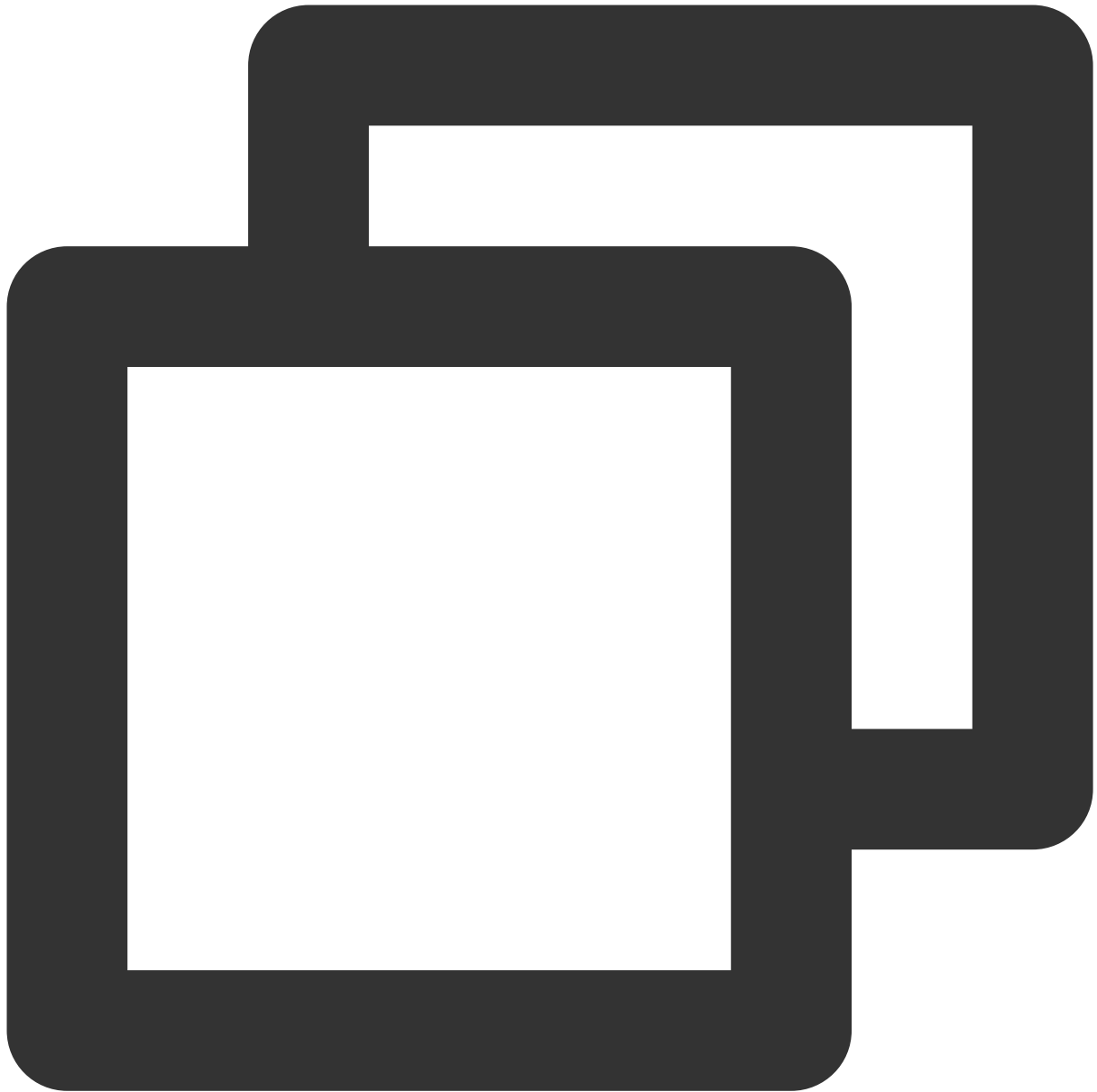


```
// Room owner
// 1. Call `sendInvitation` to invite user `123` to take seat 2
let inviteId = self.mTRTCVoiceRoom.sendInvitation(cmd: "pickSeat", userId: ownerUse
// Callback of the result
}

// 2. Place the user in the seat after the invitation is accepted
func onInviteeAccepted(identifier: String, invitee: String) {
    if identifier == selfID {
        self.mTRTCVoiceRoom.pickSeat(seatIndex: ) { (code, message) in
            // Callback of the result
        }
    }
}
```

```
    }  
  }  
}  
  
// Listener  
// 1. The listener receives the invitation  
func onReceiveNewInvitation(identifier: String, inviter: String, cmd: String, conte  
    if cmd == "pickSeat" {  
        // 2. The listener accepts the invitation  
        self.mTRTCVoiceRoom.acceptInvitation(identifier: identifier, callback: nil)  
    }  
}
```

7. Implement text chat through [TRTCVoiceRoom#sendRoomTextMsg](#).

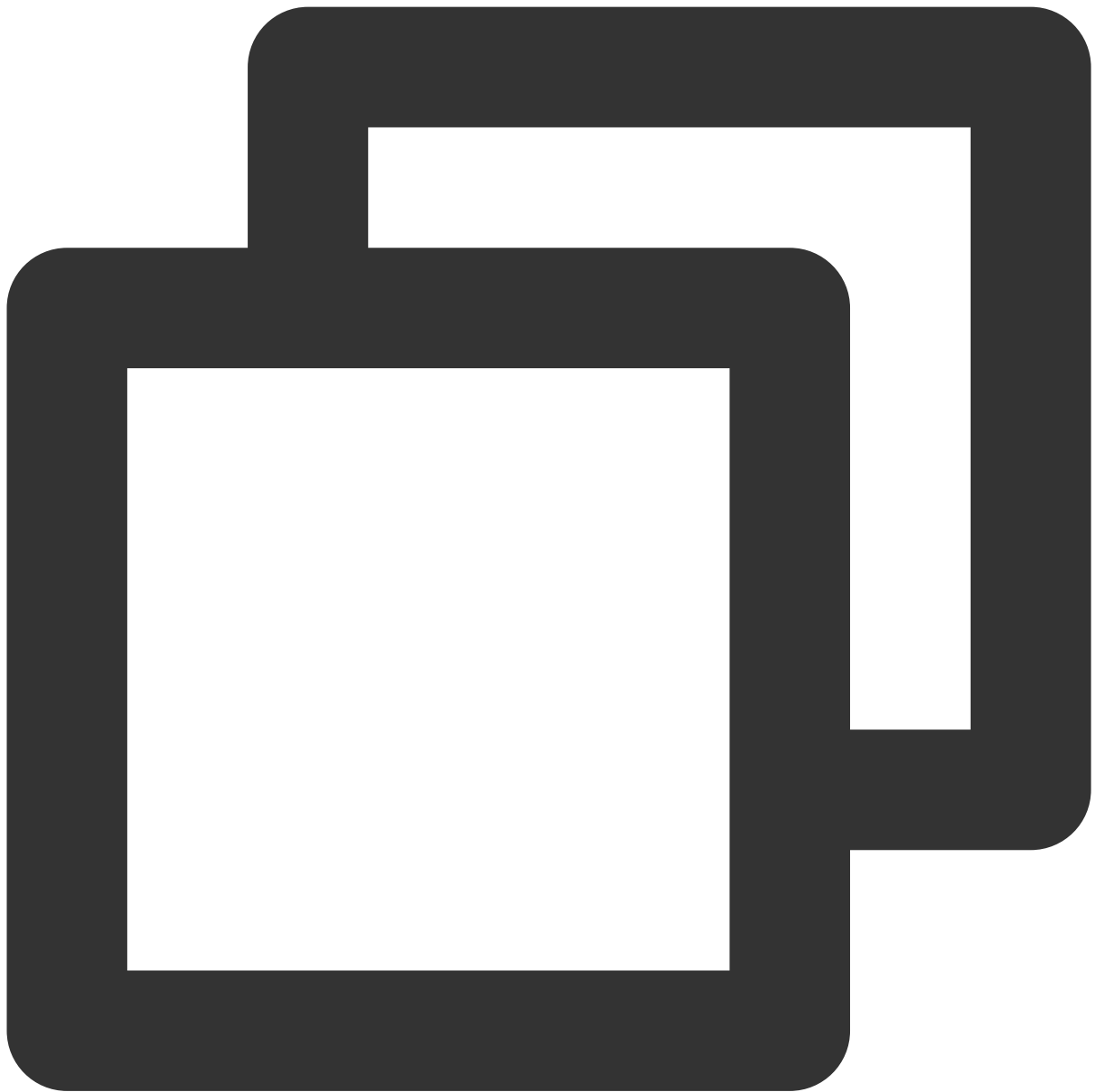


```
// Sender: Sends text chat messages
self.mTRTCVoiceRoom.sendRoomTextMsg(message: message) { (code, message) in

}

// Receiver: Listens for text chat messages
func onRecvRoomTextMsg(message: String, userInfo: VoiceRoomUserInfo) {
    // Handling of the messages received
}
```

#### 8. Implement on-screen commenting through [TRTCVoiceRoom#sendRoomCustomMsg](#).



```
// For example, a sender can customize commands to distinguish on-screen comments a
// For example, use "CMD_DANMU" to indicate on-screen comments and "CMD_LIKE" to in
self.mTRTCVoiceRoom.sendRoomCustomMsg(cmd: "CMD_DANMU", message: "hello world", cal
self.mTRTCVoiceRoom.sendRoomCustomMsg(cmd: "CMD_LIKE", message: "", callback: nil)

// Receiver: Listens for custom messages
func onRecvRoomCustomMsg(cmd: String, message: String, userInfo: VoiceRoomUserInfo)
    if cmd == "CMD_DANMU" {
        // An on-screen comment is received
    }
    if cmd == "CMD_LIKE" {
```



```
// A like is received  
}  
}
```

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [colleenyu@tencent.com](mailto:colleenyu@tencent.com).

# TUIVoiceRoom APIs

## TRTCVoiceRoom (iOS)

Last updated : 2023-09-25 10:52:31

`TRTCVoiceRoom` is based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat. With it, a user can create an audio chat room and become a speaker or enter an audio chat room as a listener. The room owner can invite a listener to speak as well as remove a speaker from the seat. The room owner can also block a seat. Listeners cannot request to take a blocked seat. A listener can request to speak and become a speaker. A speaker can also become a listener. All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

### Note

All TUIKit components are based on two PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For billing details of Chat, see [Pricing](#).

`TRTCVoiceRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, see [Audio Chat Room \(iOS\)](#).

The [TRTC SDK](#) is used as a low-latency audio chat component.

The `AVChatRoom` feature of the [Chat SDK](#) is used to implement chat rooms. The attribute APIs of Chat are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

## TRTCVoiceRoom API Overview

### Basic SDK APIs

API	Description
<a href="#">sharedInstance</a>	Gets a singleton object.
<a href="#">destroySharedInstance</a>	Terminates a singleton object.
<a href="#">setDelegate</a>	Sets event callbacks.
<a href="#">setDelegateHandler</a>	Sets the thread where event callbacks are.
<a href="#">login</a>	Logs in.
<a href="#">logout</a>	Logs out.

<a href="#">setSelfProfile</a>	Sets profile.
--------------------------------	---------------

## Room APIs

API	Description
<a href="#">createRoom</a>	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
<a href="#">destroyRoom</a>	Terminates a room (called by room owner).
<a href="#">enterRoom</a>	Enters a room (called by listener).
<a href="#">exitRoom</a>	Exits a room (called by listener).
<a href="#">getRoomInfoList</a>	Gets room list details.
<a href="#">getUserInfoList</a>	Gets the user information of the specified <code>userId</code> . If the value is <code>nil</code> , the information of all users in the room is obtained.

## Seat management APIs

API	Description
<a href="#">enterSeat</a>	Becomes a speaker (called by room owner or listener).
<a href="#">moveSeat</a>	Changes the seat (called by speaker).
<a href="#">leaveSeat</a>	Becomes a listener (called by speaker).
<a href="#">pickSeat</a>	Places a user in a seat (called by room owner).
<a href="#">kickSeat</a>	Removes a speaker (called by room owner).
<a href="#">muteSeat</a>	Mutes/Unmutes a seat (called by room owner).
<a href="#">closeSeat</a>	Blocks/Unblocks a seat (called by room owner).

## Local audio APIs

API	Description
<a href="#">startMicrophone</a>	Starts mic capturing.
<a href="#">stopMicrophone</a>	Stops mic capturing.

<a href="#">setAudioQuality</a>	Sets audio quality.
<a href="#">muteLocalAudio</a>	Mutes/Unmutes local audio.
<a href="#">setSpeaker</a>	Sets whether to play sound from the device's speaker or receiver.
<a href="#">setAudioCaptureVolume</a>	Sets mic capturing volume.
<a href="#">setAudioPlayOutVolume</a>	Sets playback volume.
<a href="#">setVoiceEarMonitorEnable</a>	Enables/Disables in-ear monitoring.

## Remote audio APIs

API	Description
<a href="#">muteRemoteAudio</a>	Mutes/Unmutes a specified member.
<a href="#">muteAllRemoteAudio</a>	Mutes/Unmutes all members.

## Background music and audio effect APIs

API	Description
<a href="#">getAudioEffectManager</a>	Gets the background music and audio effect management object <a href="#">TXAudioEffectManager</a> .

## Message sending APIs

API	Description
<a href="#">sendRoomTextMsg</a>	Broadcasts a text chat message in a room. This API is generally used for on-screen comments.
<a href="#">sendRoomCustomMsg</a>	Sends a custom text message.

## Invitation signaling APIs

API	Description
<a href="#">sendInvitation</a>	Sends an invitation.
<a href="#">acceptInvitation</a>	Accepts an invitation.
<a href="#">rejectInvitation</a>	Declines an invitation.

[cancelInvitation](#)

Cancels an invitation.

## TRTCVoiceRoomDelegate API Overview

### Common event callbacks

API	Description
<a href="#">onError</a>	Callback for error.
<a href="#">onWarning</a>	Callback for warning.
<a href="#">onDebugLog</a>	Callback of log.

### Room event callback APIs

API	Description
<a href="#">onRoomDestroy</a>	The room was terminated.
<a href="#">onRoomInfoChange</a>	The room information changed.
<a href="#">onUserVolumeUpdate</a>	User volume

### Seat list change callback APIs

API	Description
<a href="#">onSeatListChange</a>	All seat changes
<a href="#">onAnchorEnterSeat</a>	Someone became a speaker or was made a speaker by the room owner.
<a href="#">onAnchorLeaveSeat</a>	Someone became a listener or was made a listener by the room owner.
<a href="#">onSeatMute</a>	The room owner muted a seat.
<a href="#">onUserMicrophoneMute</a>	Whether a user's mic is muted
<a href="#">onSeatClose</a>	The room owner blocked a seat.

### Callback APIs for room entry/exit by listener

--	--

API	Description
<a href="#">onAudienceEnter</a>	A listener entered the room.
<a href="#">onAudienceExit</a>	A listener exited the room.

### Message event callback APIs

API	Description
<a href="#">onRecvRoomTextMsg</a>	A text chat message was received.
<a href="#">onRecvRoomCustomMsg</a>	A custom message was received.

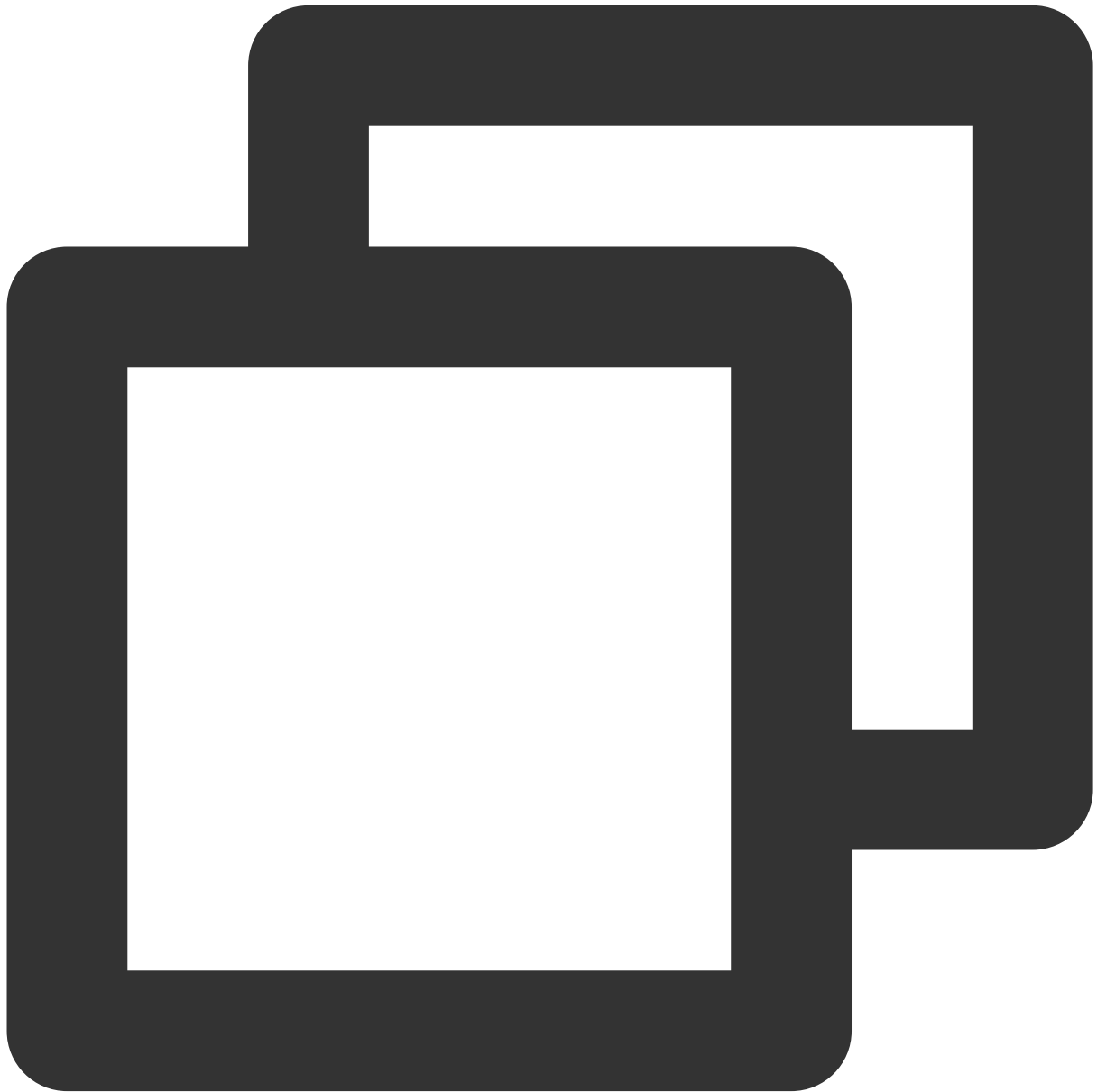
### Signaling event callback APIs

API	Description
<a href="#">onReceiveNewInvitation</a>	An invitation was received.
<a href="#">onInviteeAccepted</a>	The invitee accepted the invitation.
<a href="#">onInviteeRejected</a>	The invitee declined the invitation.
<a href="#">onInvitationCancelled</a>	The inviter canceled the invitation.

## Basic SDK APIs

### sharedInstance

This API is used to get a [TRTCVoiceRoom](#) singleton object.



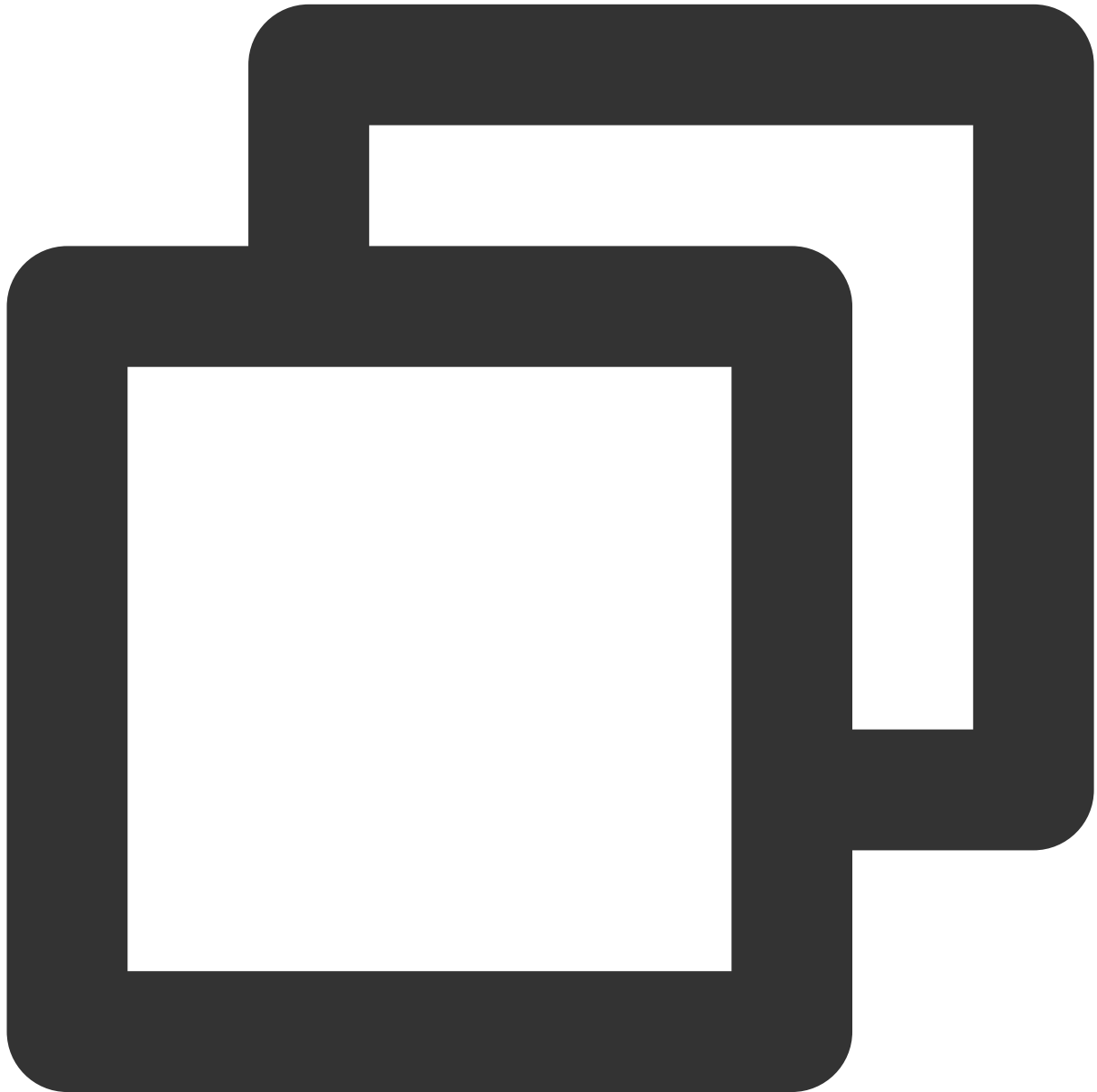
```
/**
 * Get a `TRTCVoiceRoom` singleton object
 *
 * - returns: `TRTCVoiceRoom` instance
 * - note: to terminate a singleton object, call {@link TRTCVoiceRoom#destroySharedI
 */
+ (instancetype) sharedInstance NS_SWIFT_NAME(shared());
```

## destroySharedInstance

This API is used to terminate a [TRTCVoiceRoom](#) singleton object.

#### explain

After the instance is terminated, the externally cached `TRTCVoiceRoom` instance can no longer be used. You need to call [sharedInstance](#) again to get a new instance.

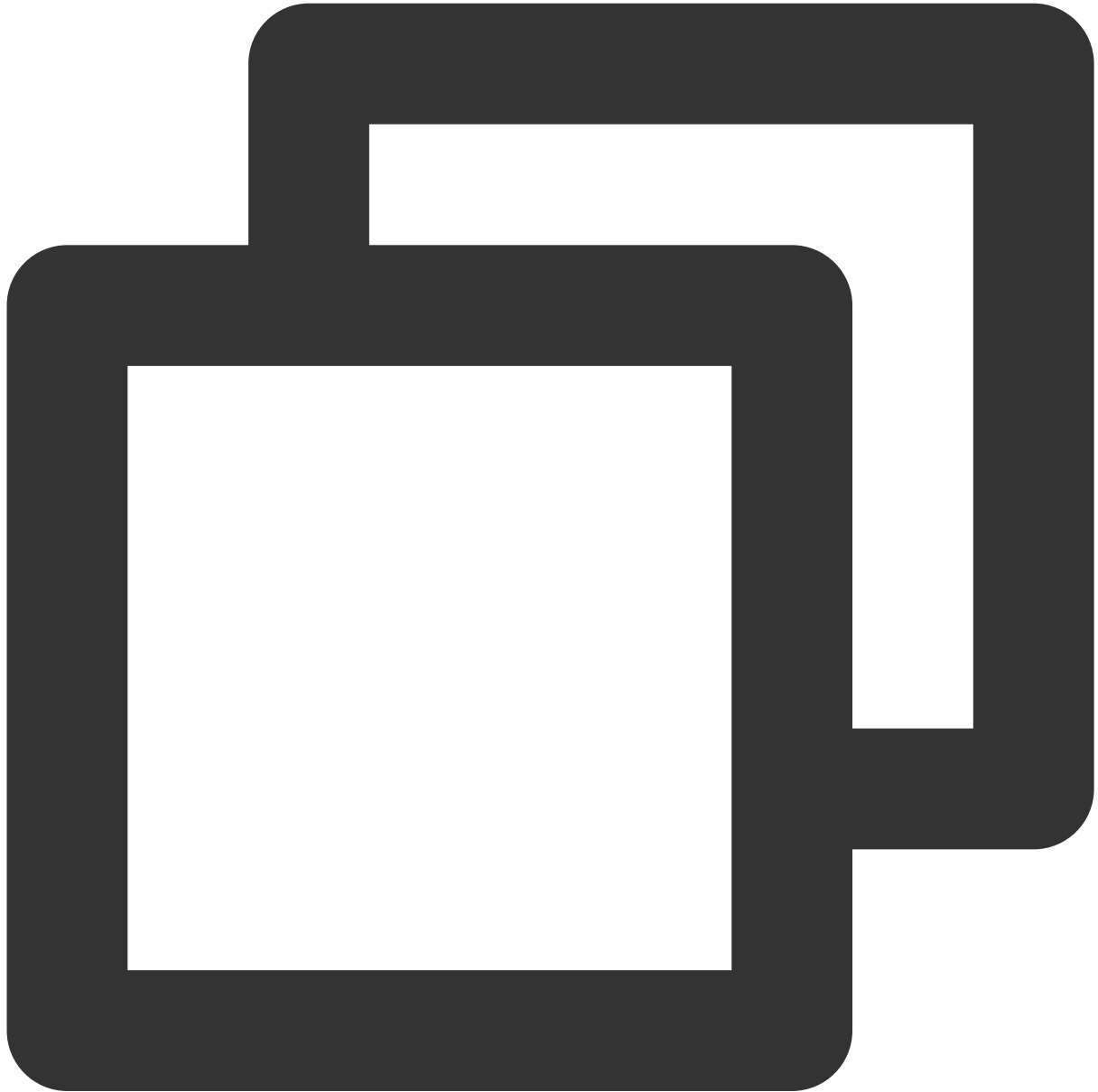


```
/**
 * Terminate a `TRTCVoiceRoom` singleton object
 *
 * - Note: After the instance is terminated, the externally cached `TRTCVoiceRoom` i
 */
+ (void)destroySharedInstance NS_SWIFT_NAME(destroyShared());
```



## setDelegate

This API is used to set the event callback of [TRTCVoiceRoom](#). You can use `TRTCVoiceRoomDelegate` to get different status notifications of [TRTCVoiceRoom](#).



```
/**
 * Set the event callbacks of the component
 *
 * You can use `TRTCVoiceRoomDelegate` to get different status notifications of `TRI
 *
```

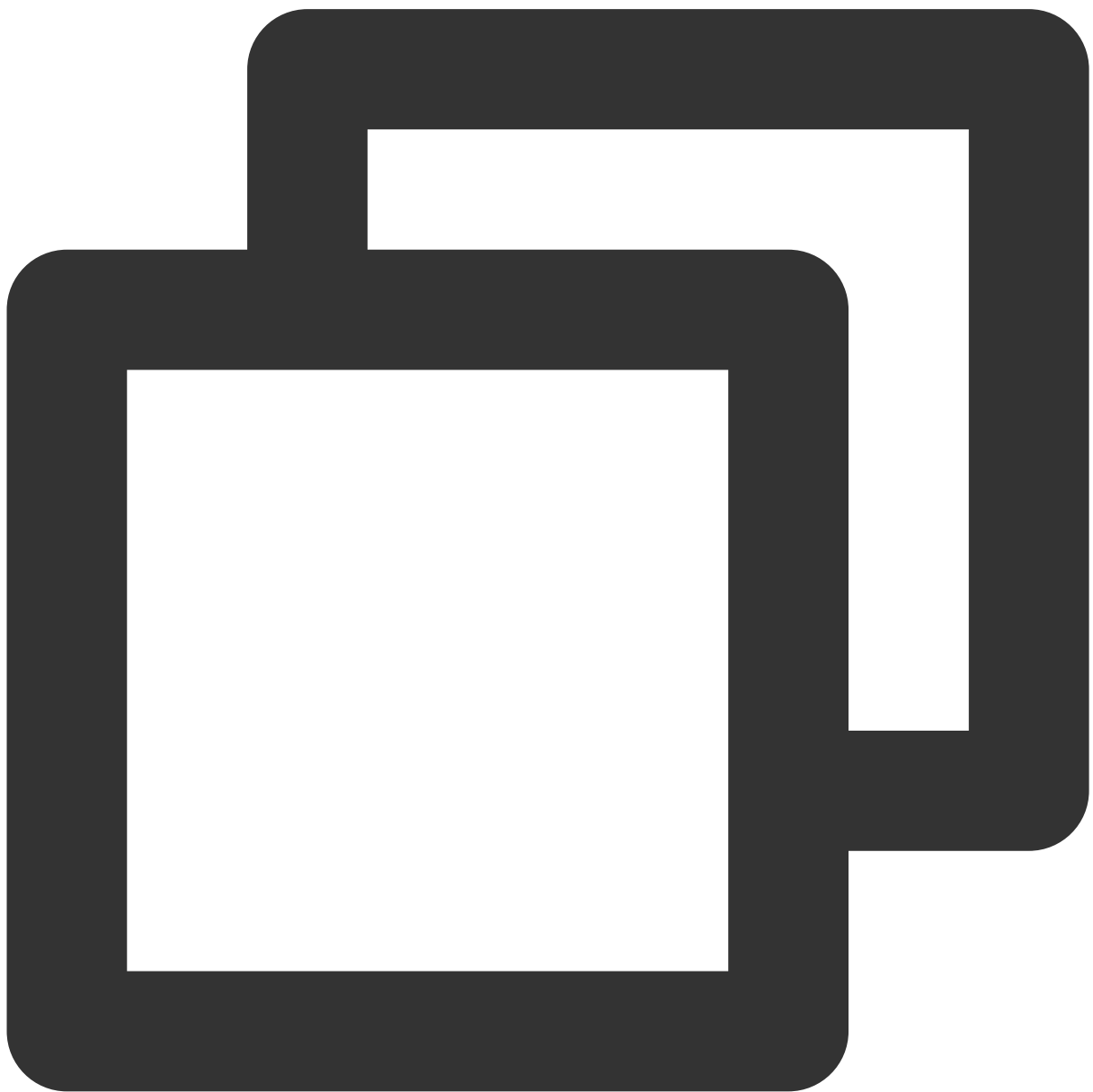
```
* - parameter delegate Callback API
* - Note: Callback events in `TRTCVoiceRoom` are called back to you in the main queue
*/
- (void)setDelegate:(id<TRTCVoiceRoomDelegate>)delegate NS_SWIFT_NAME(setDelegate(d
```

### explain

`setDelegate` is the delegate callback of `TRTCVoiceRoom` .

### setDelegateQueue

This API is used to set the thread queue for event callbacks. The main thread (MainQueue) is used by default.



```
/**
 * Set the queue for event callbacks
 *
 * - parameter queue Queue. Various status callback notifications in `TRTCVoiceRoom`
 */
- (void)setDelegateQueue:(dispatch_queue_t)queue NS_SWIFT_NAME(setDelegateQueue(queue))
```

The parameters are described below:

Parameter	Type	Description
queue	dispatch_queue_t	The status notifications of <code>TRTCVoiceRoom</code> are sent to the thread queue you specify.

## login

Login



```
- (void)login:(int) sdkAppID
    userId:(NSString *)userId
    userSig:(NSString *)userSig
    callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(login(sdkAppID:userId:userSig:callback:))
```

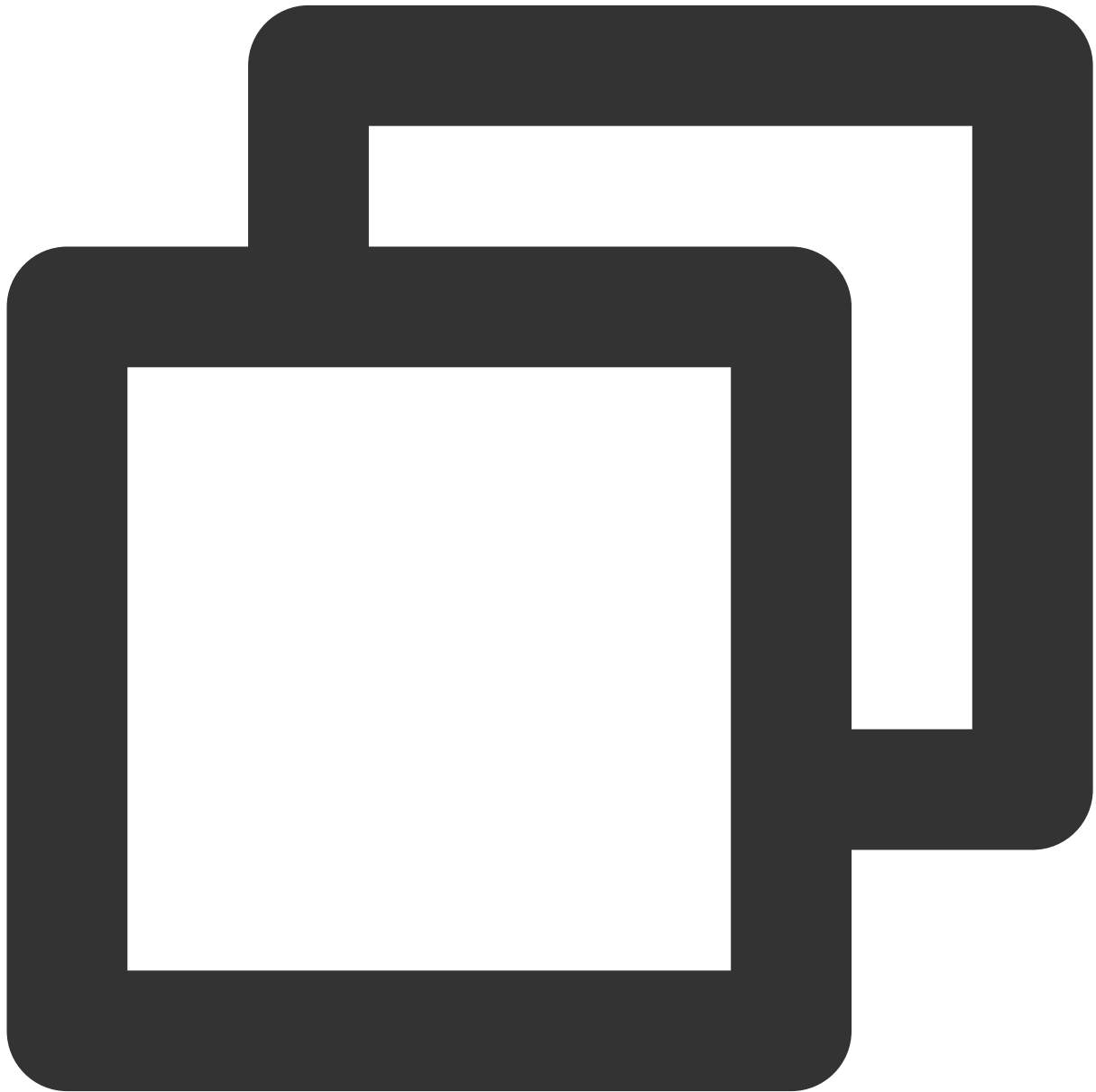
The parameters are described below:

Parameter	Type	Description
sdkAppId	int	You can view <code>SDKAppID</code> in <a href="#">Application Management</a> > <b>Application Info</b> of

		the TRTC console.
userId	NSString	ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_)
userSig	NSString	Tencent Cloud's proprietary security signature. For how to calculate and use it, see <a href="#">FAQs &gt; UserSig</a> .
callback	ActionCallback	Callback for login. The code is 0 if login succeeds.

## logout

Log out



```
- (void)logout:(ActionCallback _Nullable)callback NS_SWIFT_NAME(logout(callback:));
```

The parameters are described below:

Parameter	Type	Description
callback	ActionCallback	Callback for logout. The code is 0 if logout succeeds.

## setSelfProfile

This API is used to set the profile.



```
- (void)setSelfProfile:(NSString *)userName avatarURL:(NSString *)avatarURL callback
```

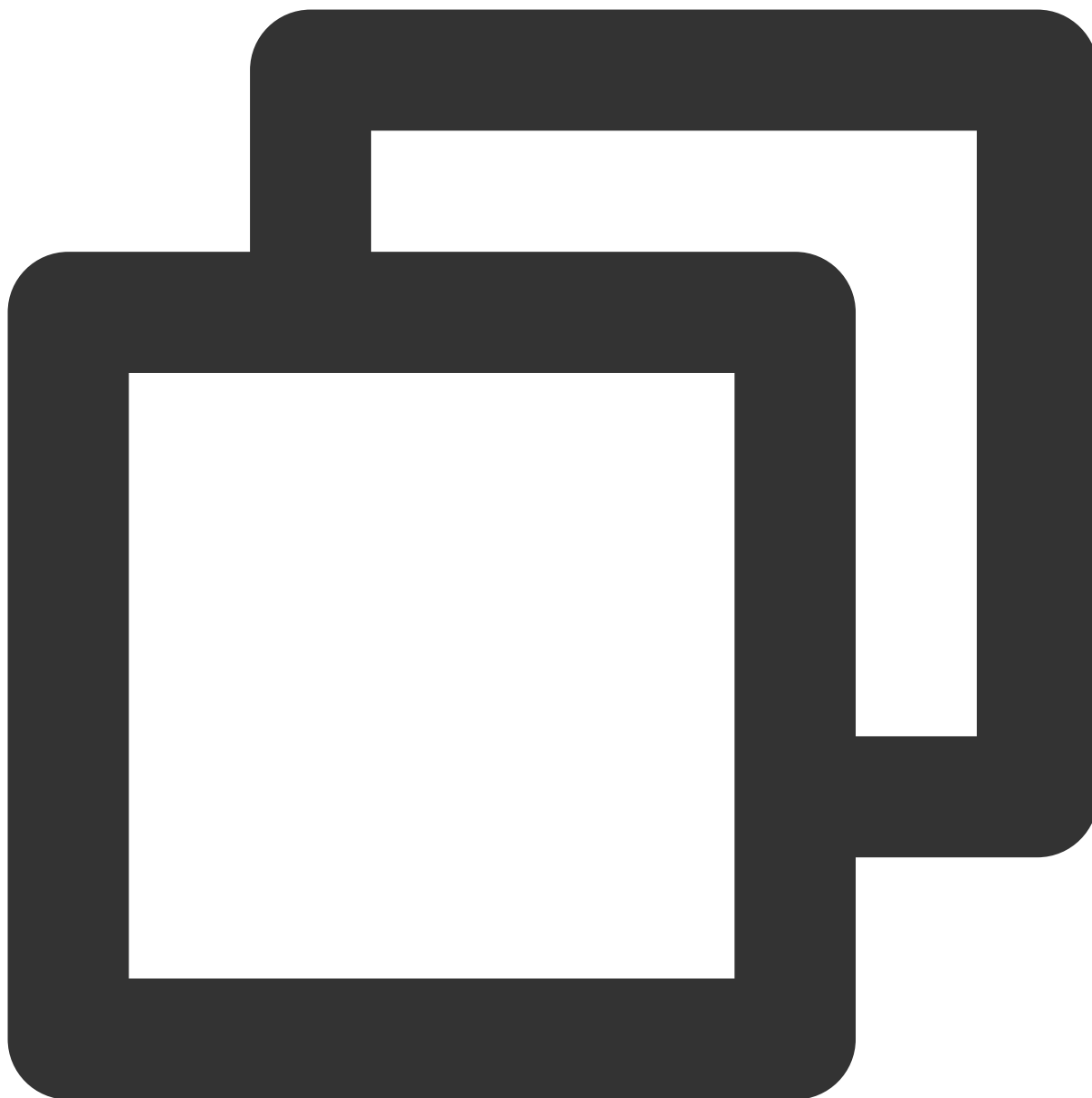
The parameters are described below:

Parameter	Type	Description
userName	NSString	Username
avatarURL	NSString	Profile picture URL
callback	ActionCallback	Callback for profile setting. The code is 0 if the operation succeeds.

## Room APIs

### createRoom

This API is used to create a room (called by room owner).



```
- (void)createRoom:(int)roomId roomParam:(VoiceRoomParam *)roomParam callback:(Acti
```

The parameters are described below:

Parameter	Type	Description



roomId	int	The room ID. You need to assign and manage the IDs in a centralized manner. Multiple <code>roomId</code> values can be aggregated into a karaoke room list. Currently, Tencent Cloud does not provide management services for room lists. Please manage your own room lists.
roomParam	VoiceRoomParam	Room information, such as room name, seat list information, and cover information. To manage seats, you must enter the number of seats in the room.
callback	ActionCallback	Callback for room creation. The code is 0 if the operation succeeds.

The process of creating a karaoke room and becoming a speaker is as follows:

1. A user calls `createRoom` to create an audio chat room, passing in room attributes (e.g. room ID, whether listeners require room owner's consent to speak, number of seats).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user will receive an `onSeatListChange` notification about the change of the seat list, and can update the change to the UI.
4. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

## destroyRoom

This API is used to terminate a room (called by room owner).



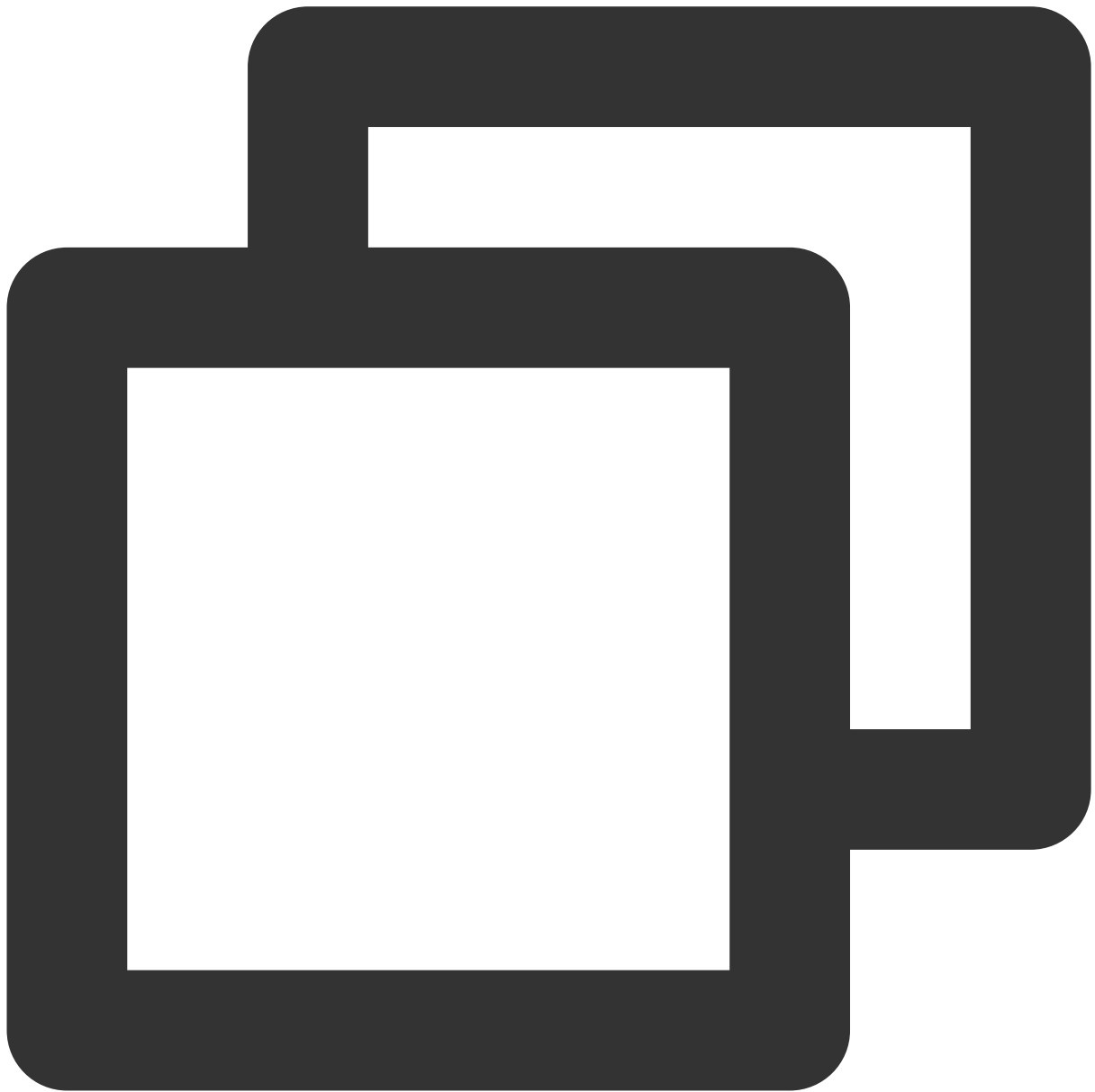
```
- (void)destroyRoom:(ActionCallback _Nullable)callback NS_SWIFT_NAME(destroyRoom(ca
```

The parameters are described below:

Parameter	Type	Description
callback	ActionCallback	Callback for room termination. The code is 0 if the operation succeeds.

## enterRoom

This API is used to enter a room (called by listener).



```
- (void)enterRoom:(NSInteger)roomId callback:(ActionCallback _Nullable)callback NS_
```

The parameters are described below:

Parameter	Type	Description
roomId	NSInteger	The room ID.
callback	ActionCallback	Callback for room entry. The code is 0 if the operation succeeds.

The process of entering a room as a listener is as follows:

1. A user gets the latest audio chat room list from your server. The list may contain the `roomId` and room information of multiple audio chat rooms.
2. The user selects a room, and calls `enterRoom` with the room ID passed in to enter the room.
3. After entering the room, the user receives an `onRoomInfoChange` notification about room attribute change from the component. The attributes can be recorded, and corresponding changes can be made to the UI, including room name, whether room owner's consent is required for listeners to speak, etc.
4. The user will receive an `onSeatListChange` notification about the change of the seat list and can update the change to the UI.
5. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker.

## **exitRoom**

Leave room



```
- (void)exitRoom:(ActionCallback _Nullable)callback NS_SWIFT_NAME(exitRoom(callback
```

The parameters are described below:

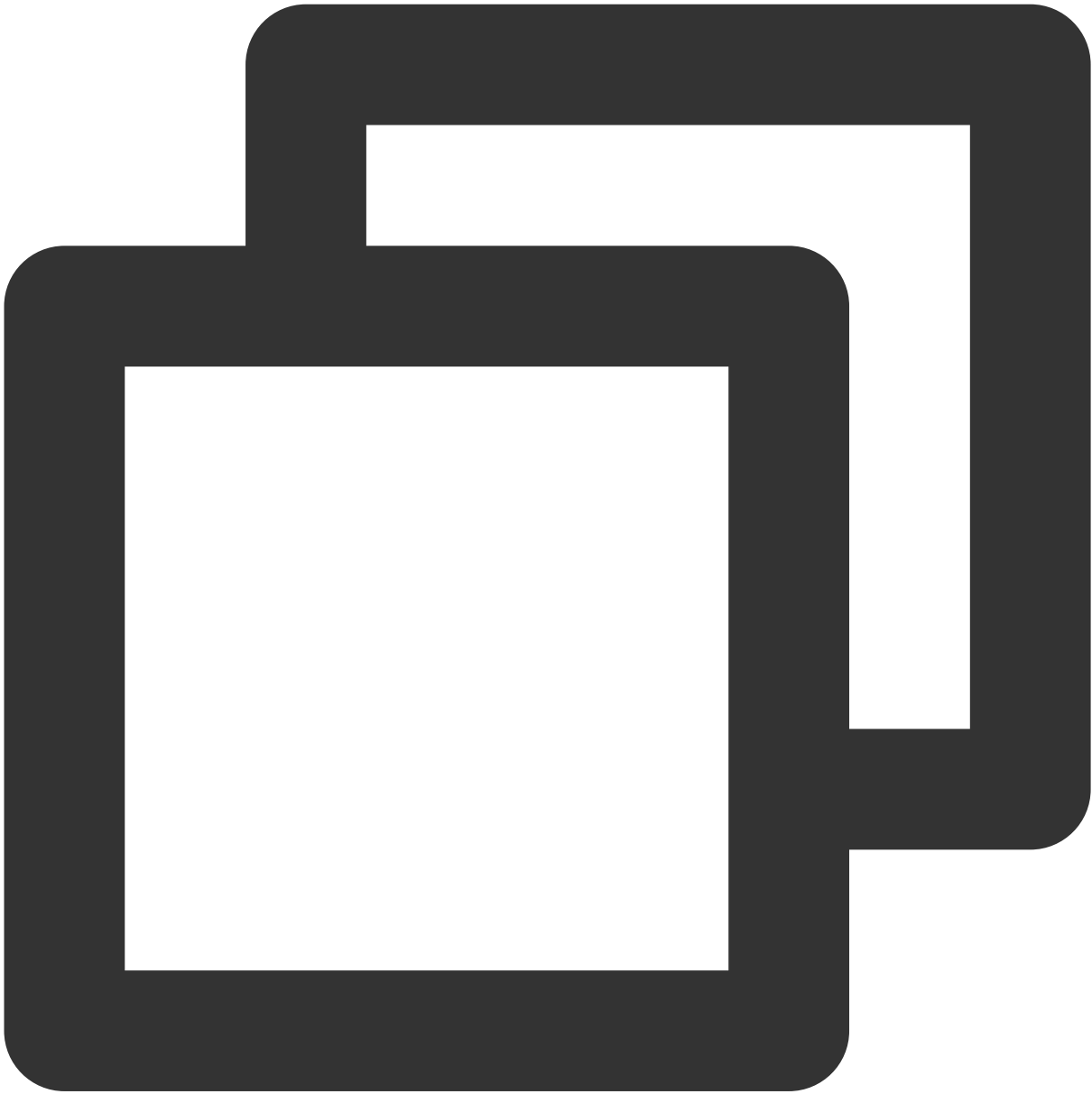
Parameter	Type	Description
callback	ActionCallback	Callback for room exit. The code is 0 if the operation succeeds.

## getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()`.

**explain**

You don't need this API if both the room list and room information are managed on your server.



```
- (void)getRoomInfoList:(NSArray<NSNumber *> *)roomIdList callback:(VoiceRoomInfoCa
```

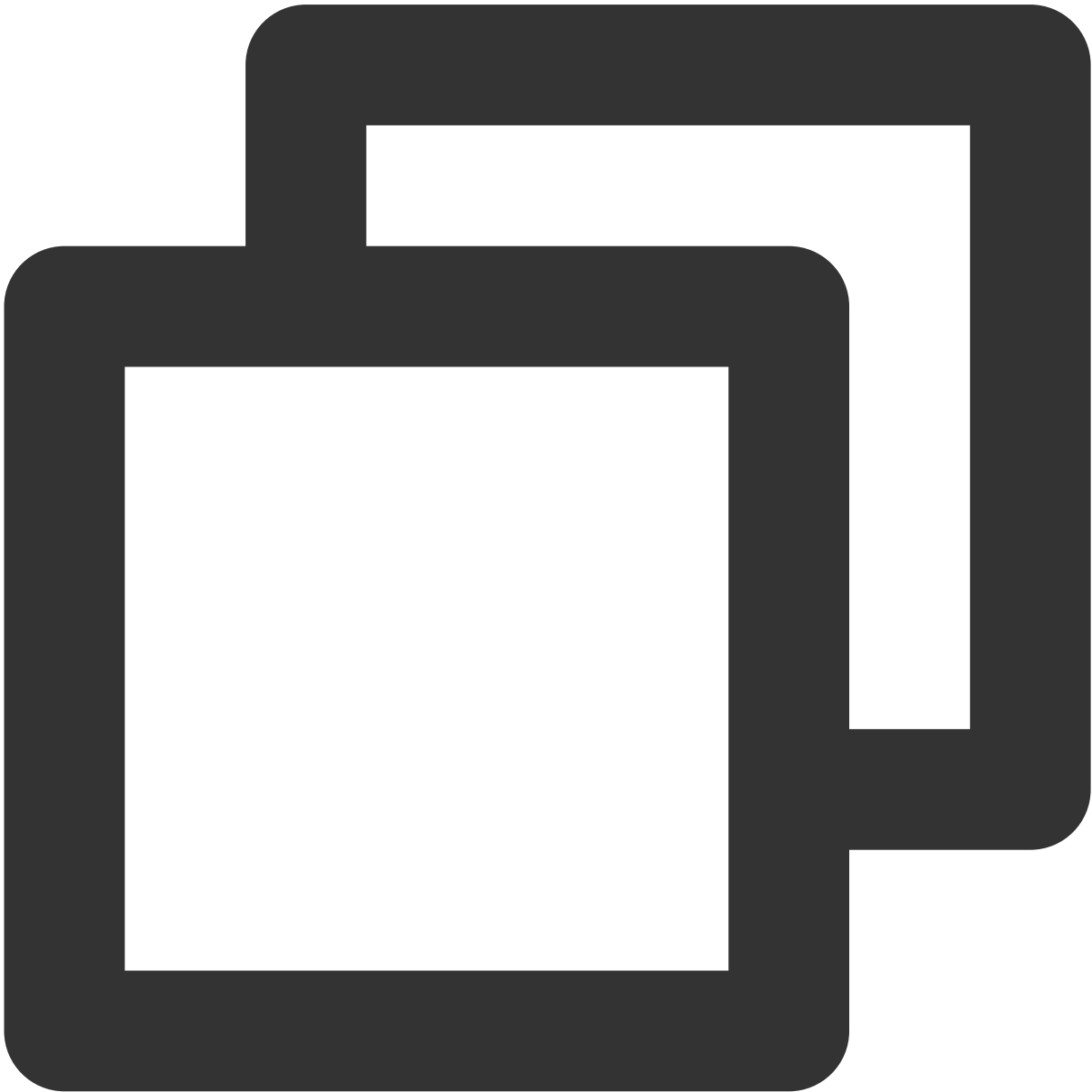
The parameters are described below:

Parameter	Type	Description

roomIdList	NSArray<NSNumber>	Room ID list
callback	RoomInfoCallback	Callback of room details

getUserInfoList

This API is used to get the information of specific users ( `userId` ).



```
- (void)getUserInfoList:(NSArray<NSString *> * _Nullable)userIDList callback:(Voice
```

The parameters are described below:

--	--	--

Parameter	Type	Description
userIdList	NSArray<NSString>	IDs of the users to query. If this parameter is <code>null</code> , the information of all users in the room is queried.
userlistcallback	UserListCallback	Callback of user details

## Seat Management APIs

### enterSeat

This API is used to become a speaker (called by room owner or listener).

#### explain

After a user becomes a speaker, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.





```
- (void)enterSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback
```

The parameters are described below:

Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to take
callback	ActionCallback	Callback for the operation

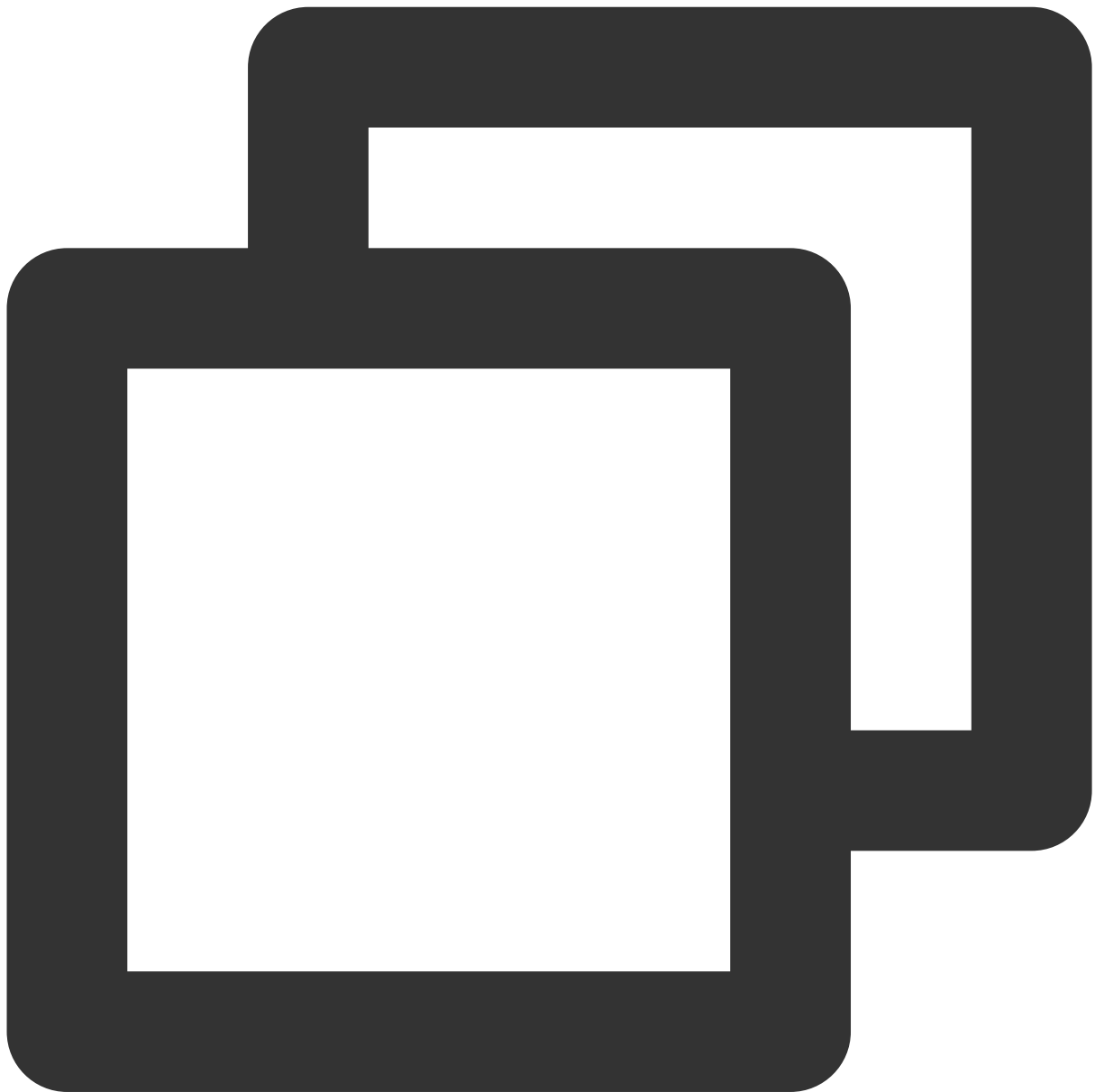
Calling this API will immediately modify the seat list. In cases where listeners need the room owner's consent to take a seat, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call this API.

## moveSeat

This API is used to change one's seat (called by speaker).

### explain

After the seat change, all users in the room will receive the `onSeatListChange`, `onAnchorLeaveSeat`, and `onAnchorEnterSeat` notifications. This API will only change the user's seat number, not the user role.



```
- (NSInteger)moveSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback  
NS_SWIFT_NAME(moveSeat(seatIndex:callback:))
```

The parameters are described below:

Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to change to
callback	ActionCallback	Callback for the operation

Response parameters:

Parameter	Type	Description
code	NSInteger	Result of seat change. <code>0</code> : operation successful; <code>10001</code> : API rate limit exceeded; other values: operation failed

Calling this API will immediately modify the seat list. In cases where listeners need the room owner's consent to take a seat, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call this API.

## leaveSeat

This API is used to become a listener (called by speaker).

### explain

After a speaker becomes a listener, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.



```
- (void)leaveSeat:(ActionCallback _Nullable)callback NS_SWIFT_NAME(leaveSeat(callback))
```

The parameters are described below:

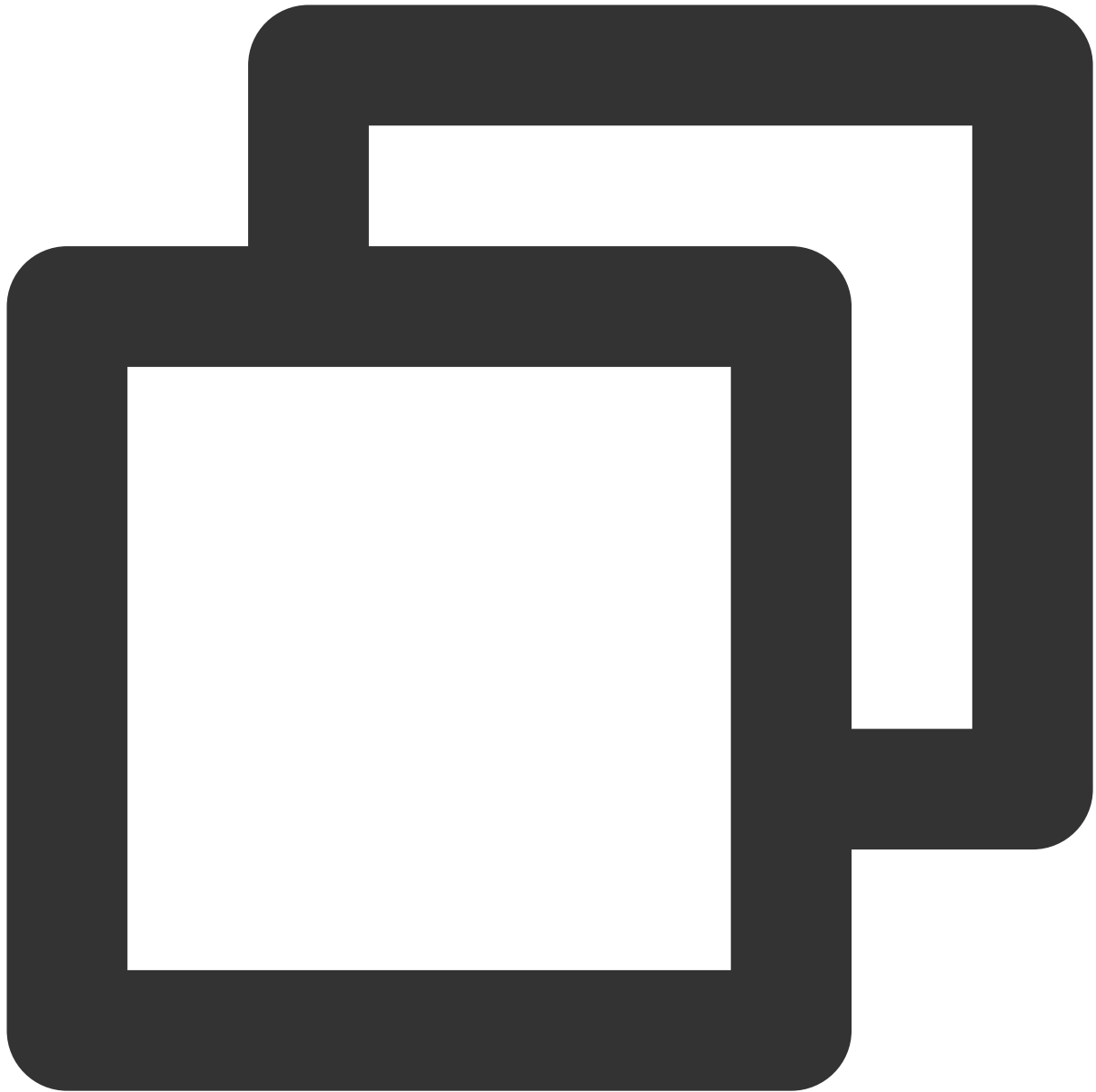
Parameter	Type	Description
callback	ActionCallback	Callback for the operation

## **pickSeat**

This API is used to place a user in a seat (called by room owner).

**explain**

After the room owner places a user in a seat, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.



```
- (void)pickSeat:(NSInteger)seatIndex userId:(NSString *)userId callback:(ActionCal
```

The parameters are described below:

Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to place the user in

userId	NSString	User ID
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. In cases where the room owner needs listeners' consent to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call `pickSeat`.

## kickSeat

This API is used to remove a speaker (called by room owner).

### explain

After a speaker is removed, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.



```
- (void)kickSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback N
```

The parameters are described below:

Parameter	Type	Description
seatIndex	NSInteger	Seat number of the speaker to remove
callback	ActionCallback	Callback for the operation

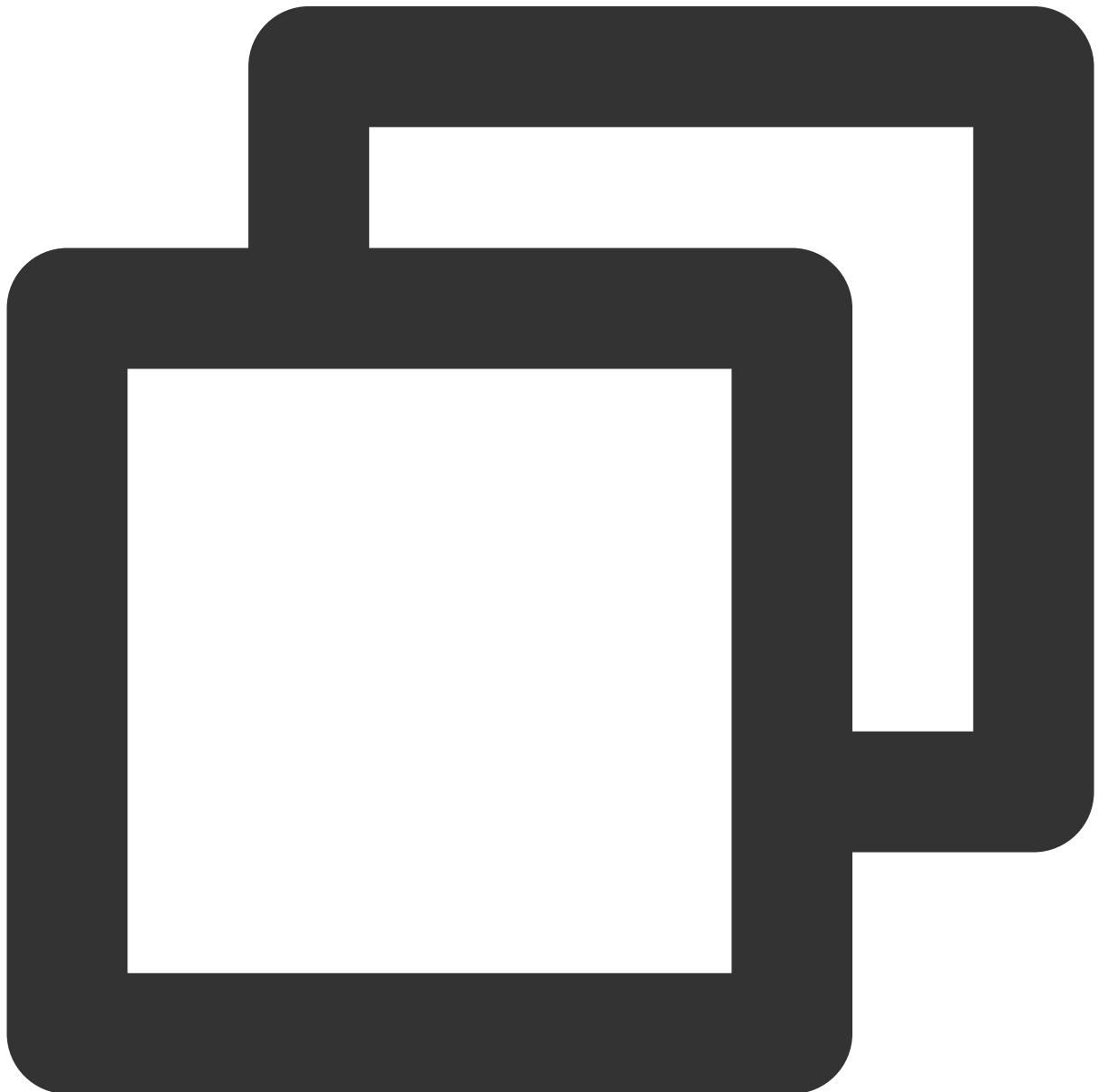
Calling this API will immediately modify the seat list.

## **muteSeat**

This API is used to mute/unmute a seat (called by room owner).

### **explain**

After a seat is muted/unmuted, all members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.



```
- (void)muteSeat:(NSInteger)seatIndex isMute:(BOOL)isMute callback:(ActionCallback
```



The parameters are described below:

Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to mute/unmute
isMute	BOOL	<code>YES</code> : mute the seat; <code>NO</code> : unmute the seat
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will call `muteAudio` to mute/unmute his or her audio.

## closeSeat

This API is used to block/unblock a seat (called by room owner).

### explain

After a seat is blocked/unblocked, all members in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.



```
- (void)closeSeat:(NSInteger)seatIndex isClose:(BOOL)isClose callback:(ActionCallba
```

The parameters are described below:

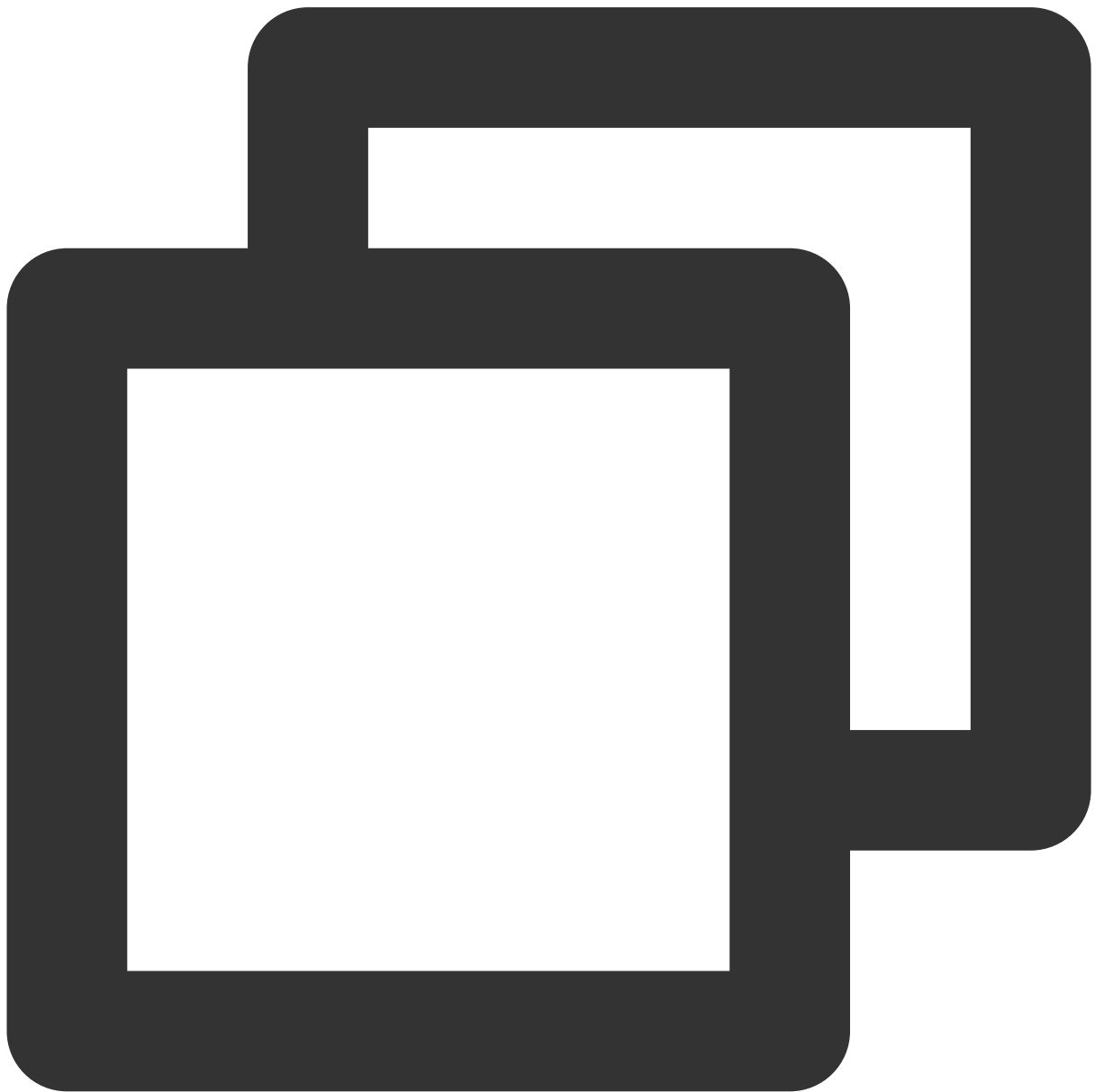
Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to block/unblock
isClose	BOOL	<code>YES</code> : block the seat; <code>NO</code> : unblock the seat
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will leave the seat.

## Local Audio APIs

### **startMicrophone**

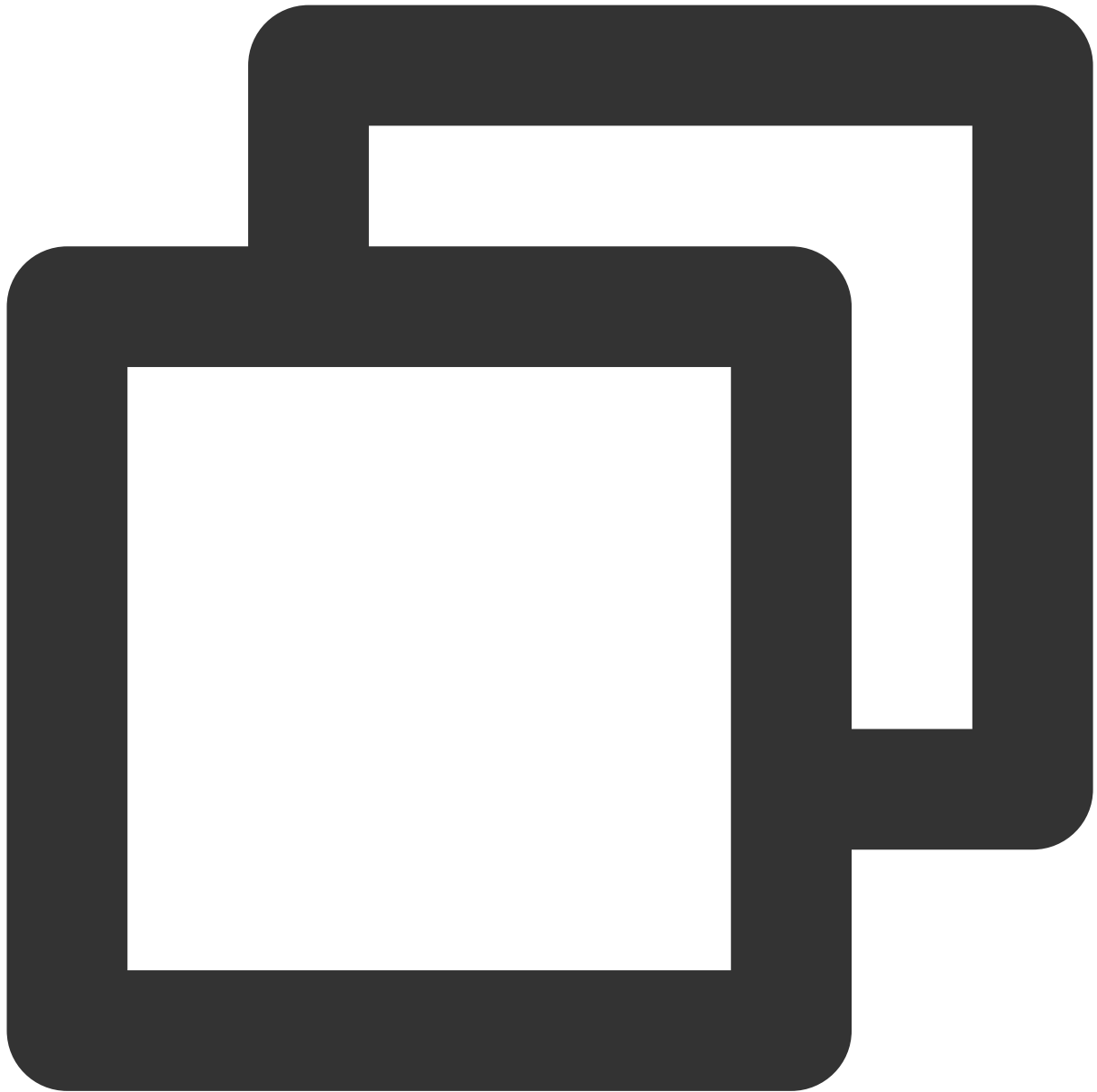
This API is used to start mic capturing.



```
- (void)startMicrophone;
```

## stopMicrophone

This API is used to stop mic capturing.



```
- (void)stopMicrophone;
```

## setAudioQuality

This API is used to set audio quality.



```
- (void)setAudioQuality:(NSInteger)quality NS_SWIFT_NAME(setAudioQuality(quantity:))
```

The parameters are described below:

Parameter	Type	Description
quality	NSInteger	The audio quality. For more information, see <a href="#">setAudioQuality()</a> .

## **muteLocalAudio**

This API is used to mute/unmute local audio.



```
- (void)muteLocalAudio:(BOOL)mute NS_SWIFT_NAME(muteLocalAudio(mute:));
```

The parameters are described below:

Parameter	Type	Description
mute	BOOL	Whether to mute or unmute audio. For more information, see <a href="#">muteLocalAudio()</a> .

## setSpeaker

This API is used to set whether to play sound from the device's speaker or receiver.



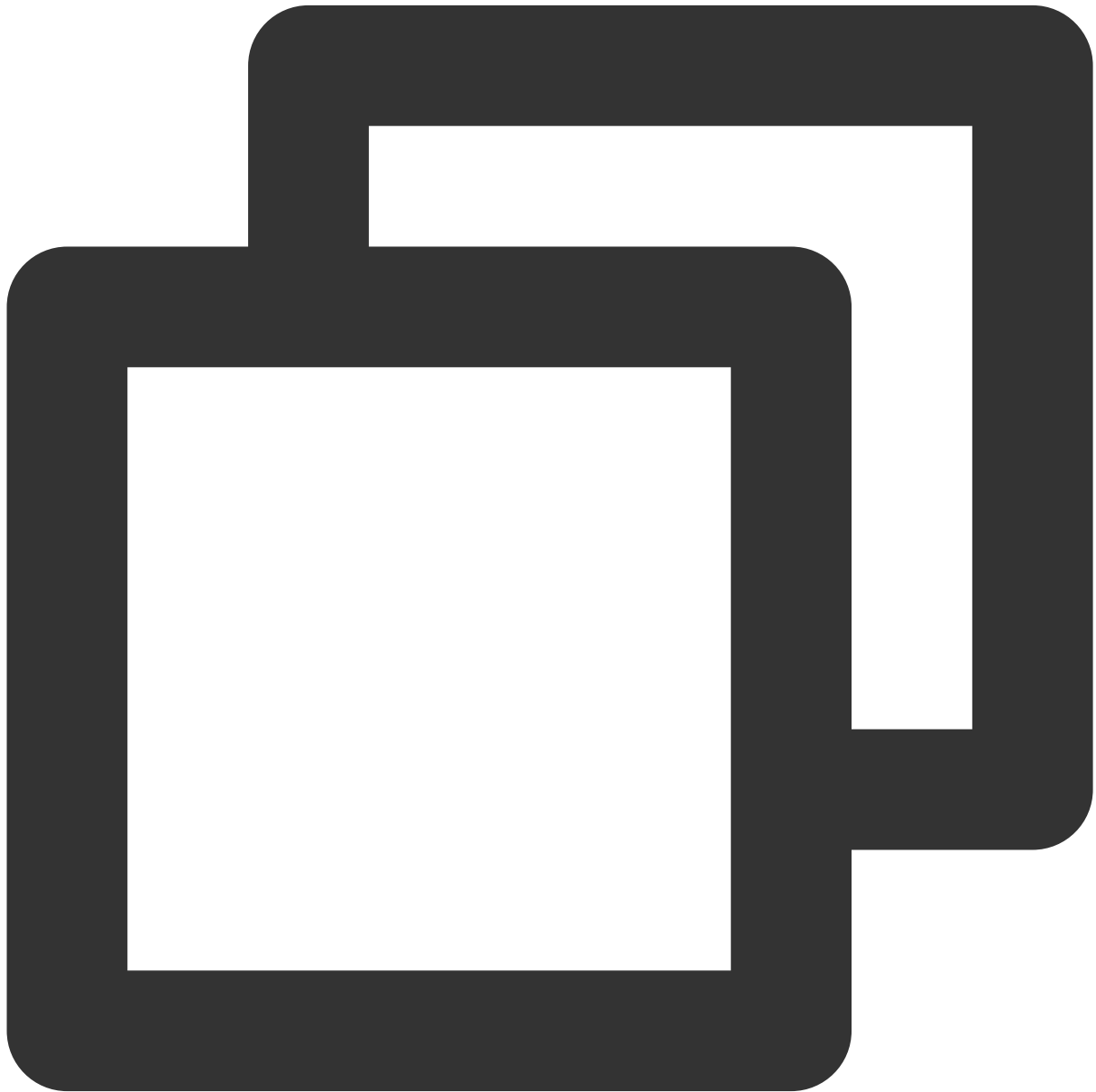
```
- (void)setSpeaker:(BOOL)userSpeaker NS_SWIFT_NAME(setSpeaker(userSpeaker:));
```

The parameters are described below:

Parameter	Type	Description
useSpeaker	BOOL	<code>YES</code> : speaker; <code>NO</code> : receiver

## setAudioCaptureVolume

This API is used to set the mic capturing volume.



```
- (void)setAudioCaptureVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioCaptureVolume)
```

The parameters are described below:

Parameter	Type	Description
volume	NSInteger	Capturing volume. Value range: 0-100. Default value: 100

### setAudioPlayoutVolume

This API is used to set the playback volume.





```
- (void)setAudioPlayoutVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioPlayoutVolume)
```

The parameters are described below:

Parameter	Type	Description
volume	NSInteger	Playback volume. Value range: 0-100. Default value: 100

## **muteRemoteAudio**

This API is used to mute/unmute a specified user.



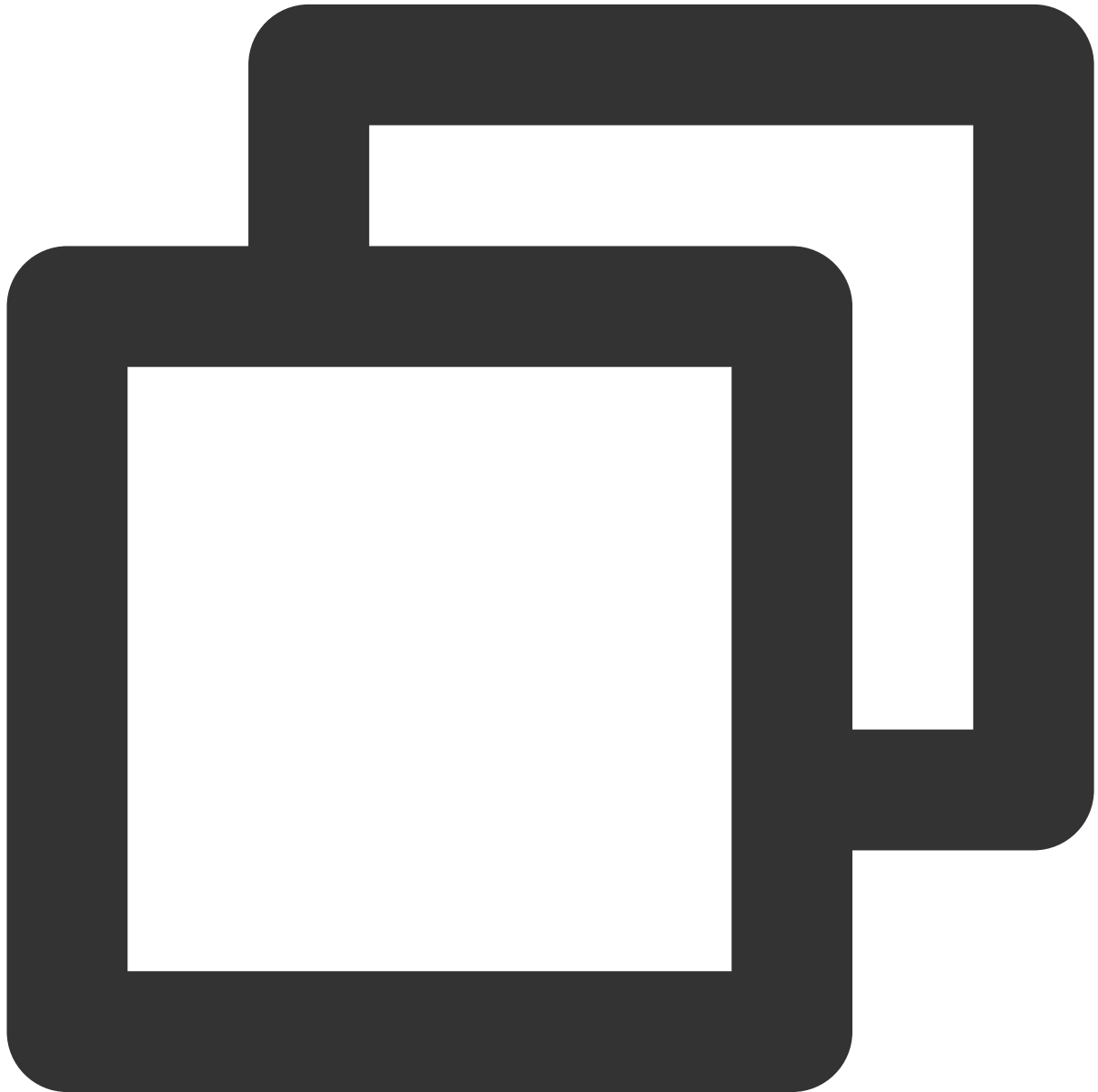
```
- (void)muteRemoteAudio:(NSString *)userId mute:(BOOL)mute NS_SWIFT_NAME(muteRemoteAudio)
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	ID of the user to mute/unmute
mute	BOOL	<code>YES</code> : mute; <code>NO</code> : unmute

## **muteAllRemoteAudio**

This API is used to mute/unmute all users.



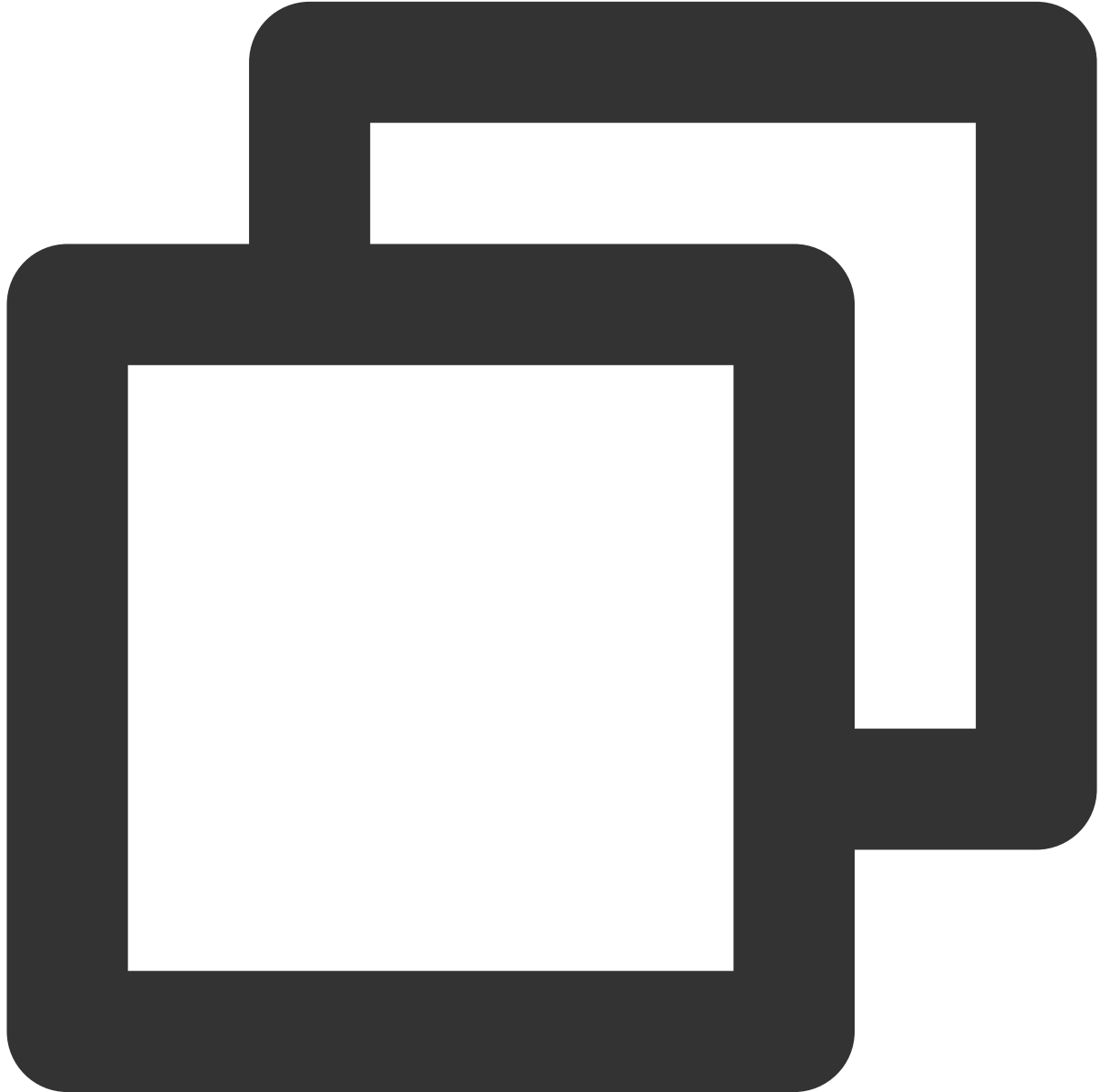
```
- (void)muteAllRemoteAudio:(BOOL)isMute NS_SWIFT_NAME(muteAllRemoteAudio(isMute:));
```

The parameters are described below:

Parameter	Type	Description
mute	BOOL	<code>YES</code> : mute; <code>NO</code> : unmute

## setVoiceEarMonitorEnable

This API is used to enable/disable in-ear monitoring.



```
- (void)setVoiceEarMonitorEnable:(BOOL)enable NS_SWIFT_NAME(setVoiceEarMonitor(enable))
```

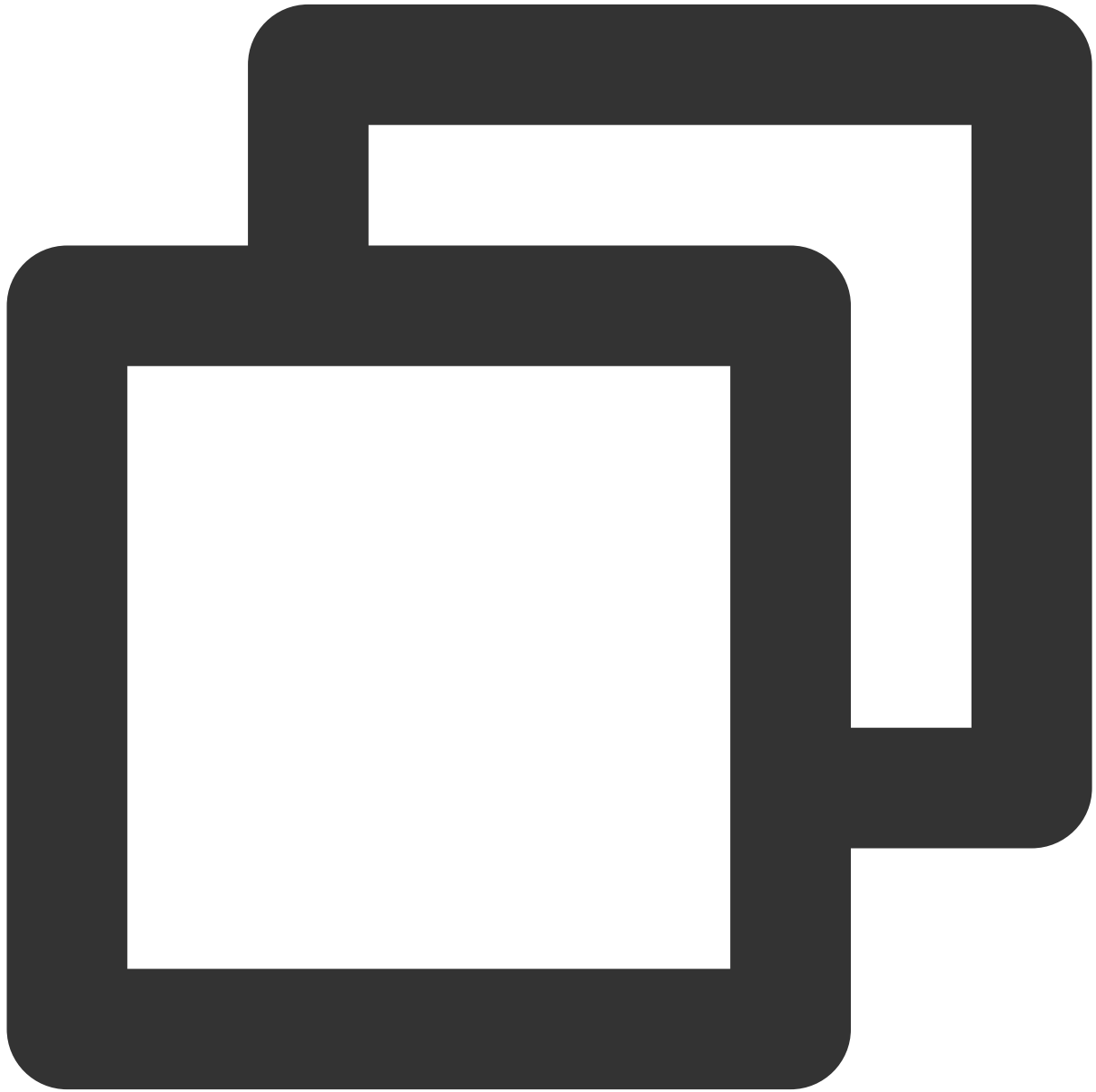
The parameters are described below:

Parameter	Type	Description
enable	BOOL	<code>YES</code> : enable in-ear monitoring; <code>NO</code> : disable in-ear monitoring

## Background Music and Audio Effect APIs

### **getAudioEffectManager**

This API is used to get the background music and audio effect management object [TXAudioEffectManager](#).

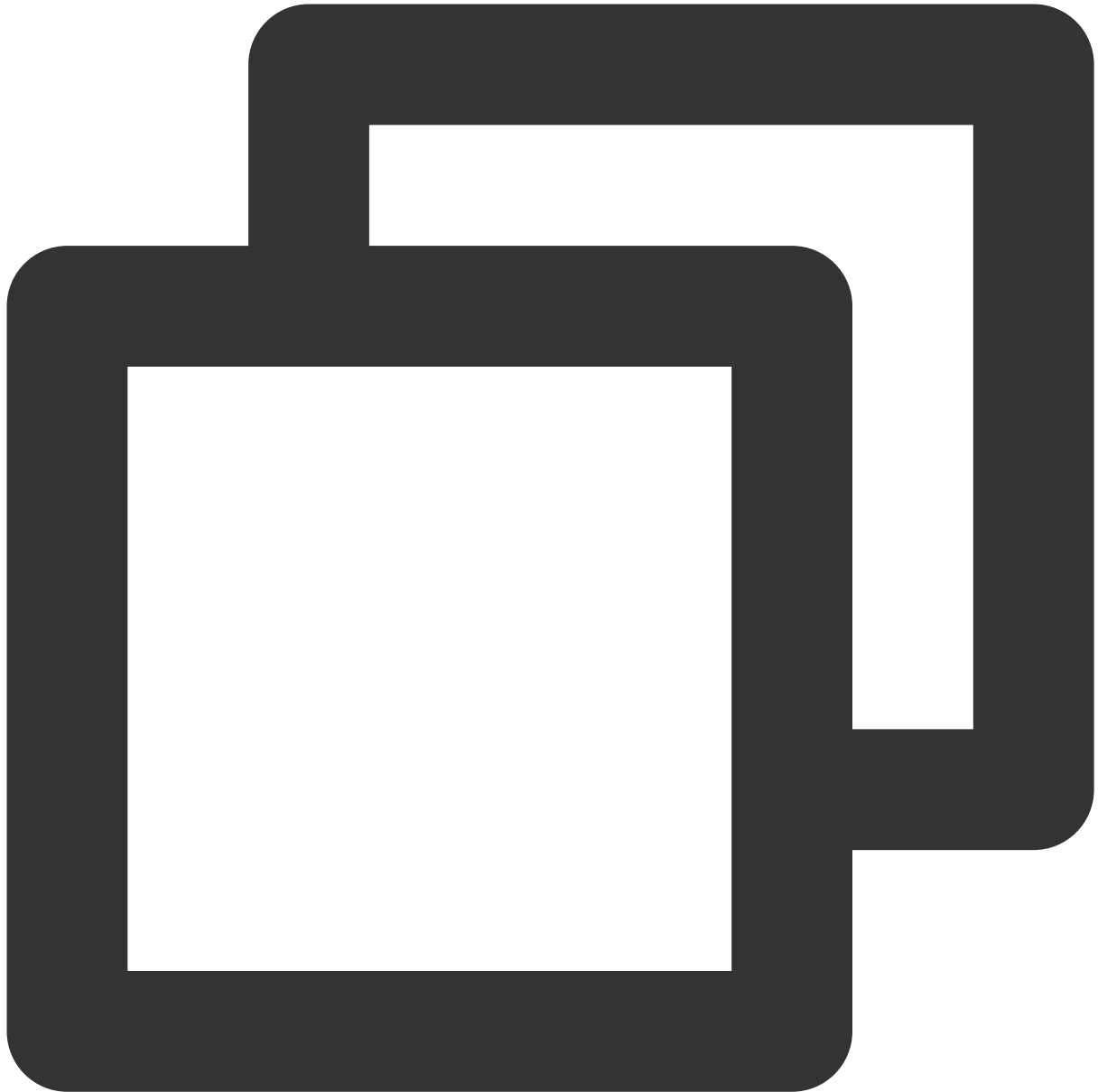


```
- (TXAudioEffectManager * _Nullable)getAudioEffectManager;
```

## Message Sending APIs

## sendRoomTextMsg

This API is used to broadcast a text chat message in a room, which is generally used for on-screen comments.



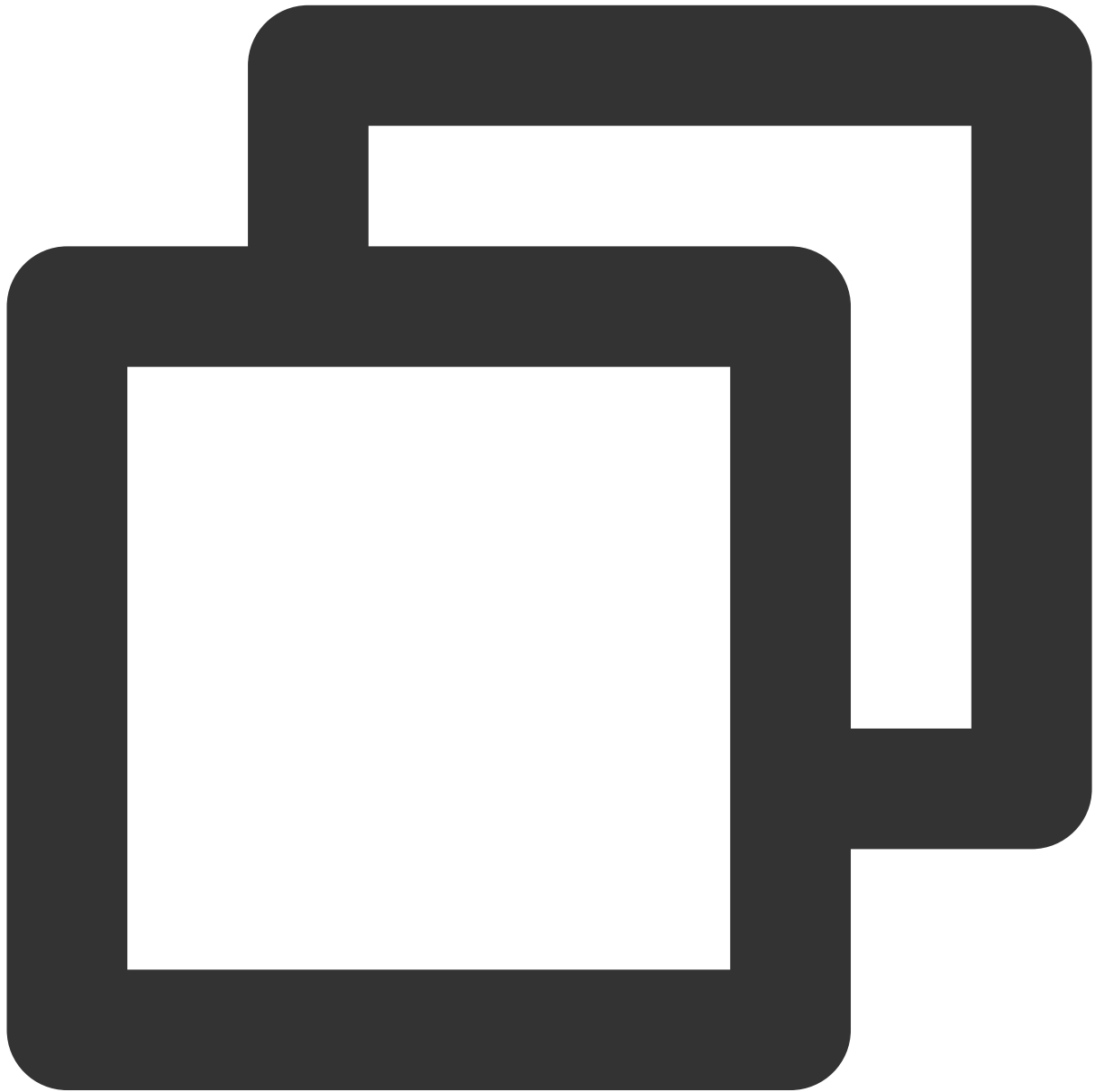
```
- (void)sendRoomTextMsg:(NSString *)message callback:(ActionCallback _Nullable)call
```

The parameters are described below:

Parameter	Type	Description
message	NSString	Text message
callback	ActionCallback	Callback for the operation

## sendRoomCustomMsg

This API is used to send a custom text message.



```
- (void)sendRoomCustomMsg:(NSString *)cmd message:(NSString *)message callback:(Act
```

The parameters are described below:

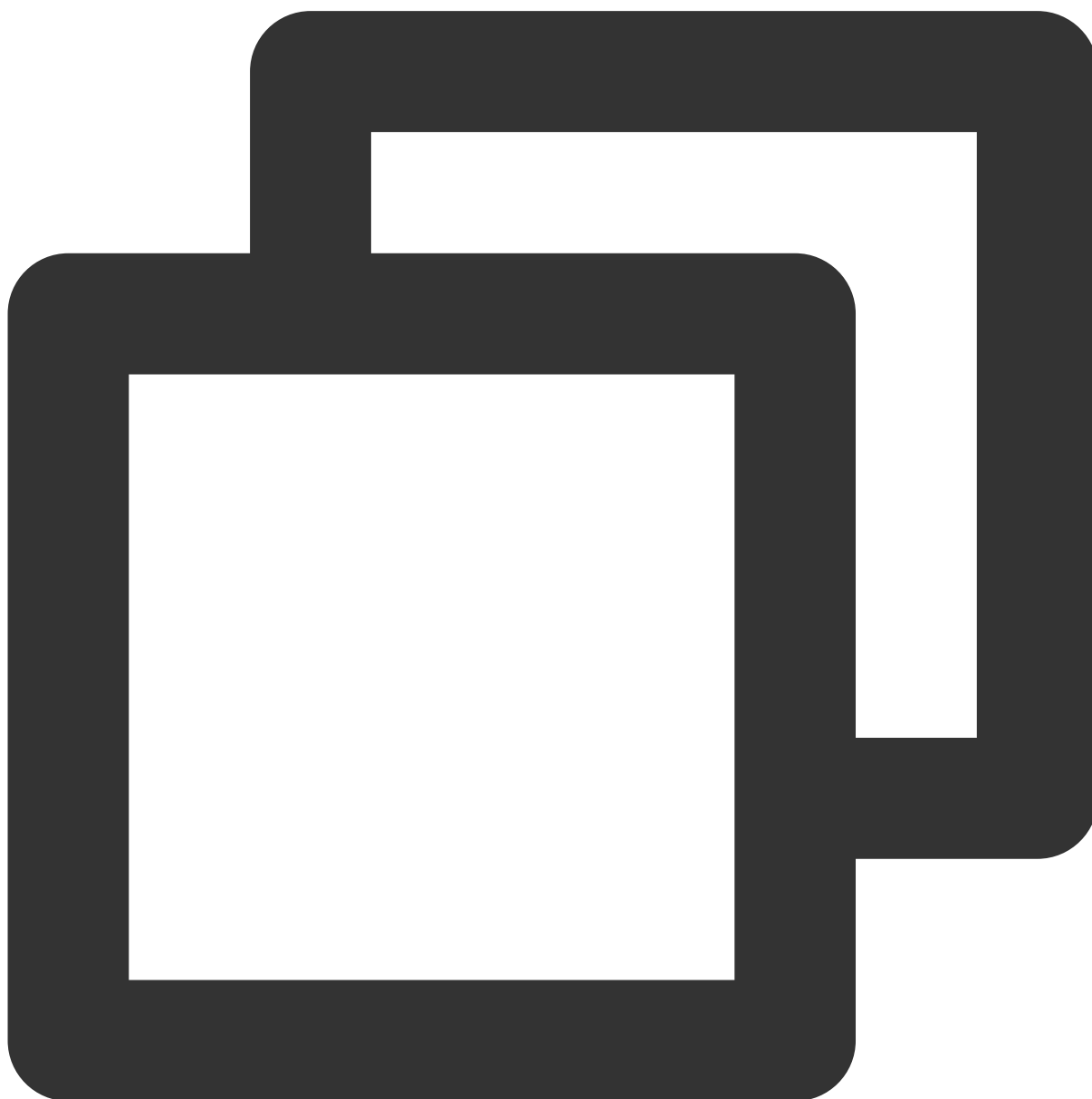
Parameter	Type	Description
cmd	NSString	Custom command word used to distinguish between different message types

message	NSString	Text message
callback	ActionCallback	Callback for the operation

## Invitation Signaling APIs

### **sendInvitation**

This API is used to send an invitation.





```
- (NSString *)sendInvitation:(NSString *)cmd
    userId:(NSString *)userId
    content:(NSString *)content
    callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendI
```

The parameters are described below:

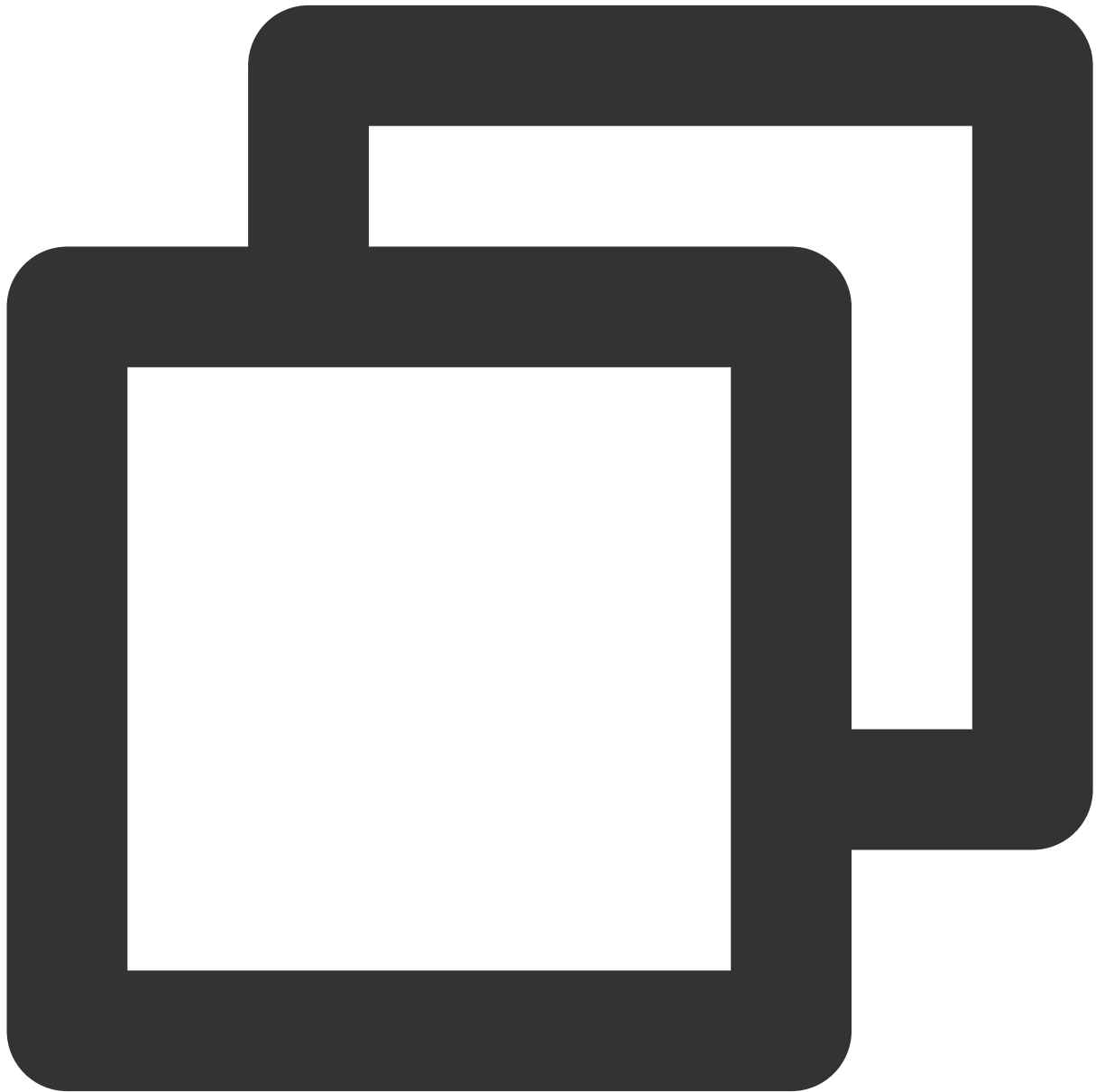
Parameter	Type	Description
cmd	NSString	Custom command of business
userId	NSString	ID of the user to invite
content	NSString	Invitation content
callback	ActionCallback	Callback for the operation

Response parameters:

Parameter	Type	Description
inviteId	NSString	Invitation ID

## acceptInvitation

This API is used to accept an invitation.



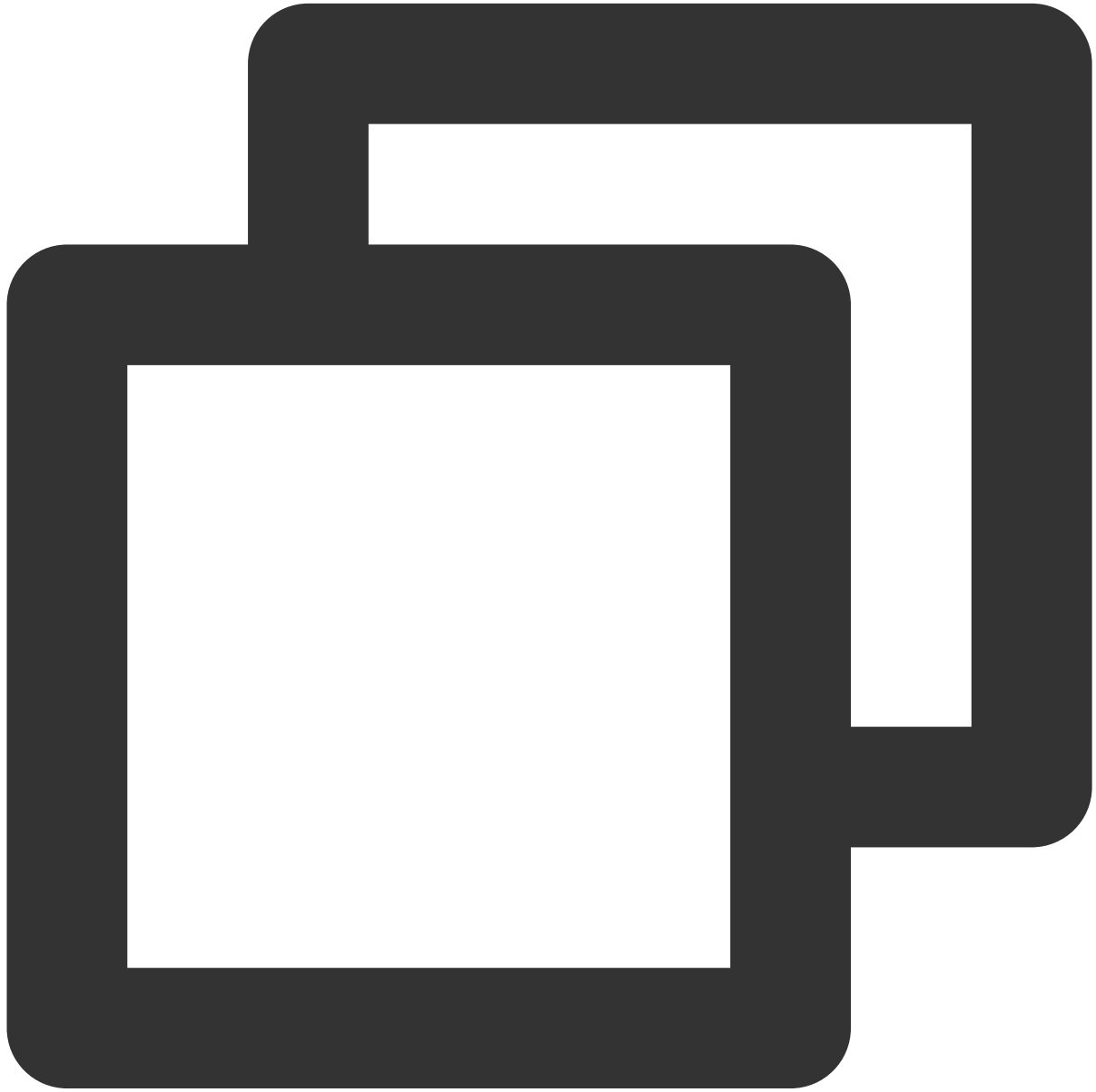
```
- (void)acceptInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)
```

The parameters are described below:

Parameter	Type	Description
id	NSString	Invitation ID
callback	ActionCallback	Callback for the operation

## rejectInvitation

This API is used to decline an invitation.



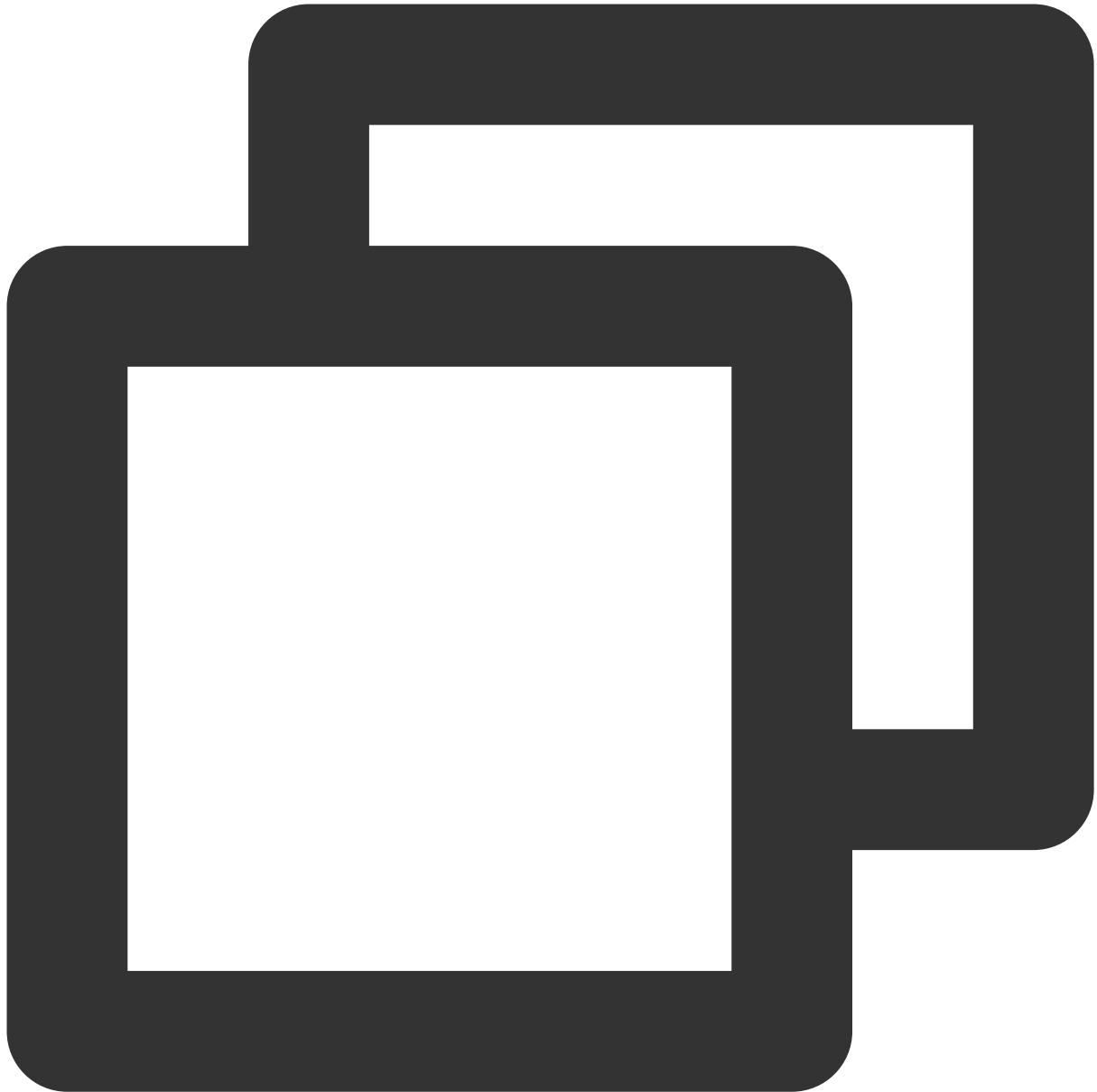
```
- (void)rejectInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)
```

The parameters are described below:

Parameter	Type	Description
id	NSString	Invitation ID
callback	ActionCallback	Callback for the operation

## cancelInvitation

This API is used to cancel an invitation.



```
- (void)cancelInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)
```

The parameters are described below:

Parameter	Type	Description
id	NSString	Invitation ID
callback	ActionCallback	Callback for the operation

## TRTCVoiceRoomDelegate Event Callback APIs

### Common Event Callback APIs

#### **onError**

Callback for error.

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users depending if necessary.



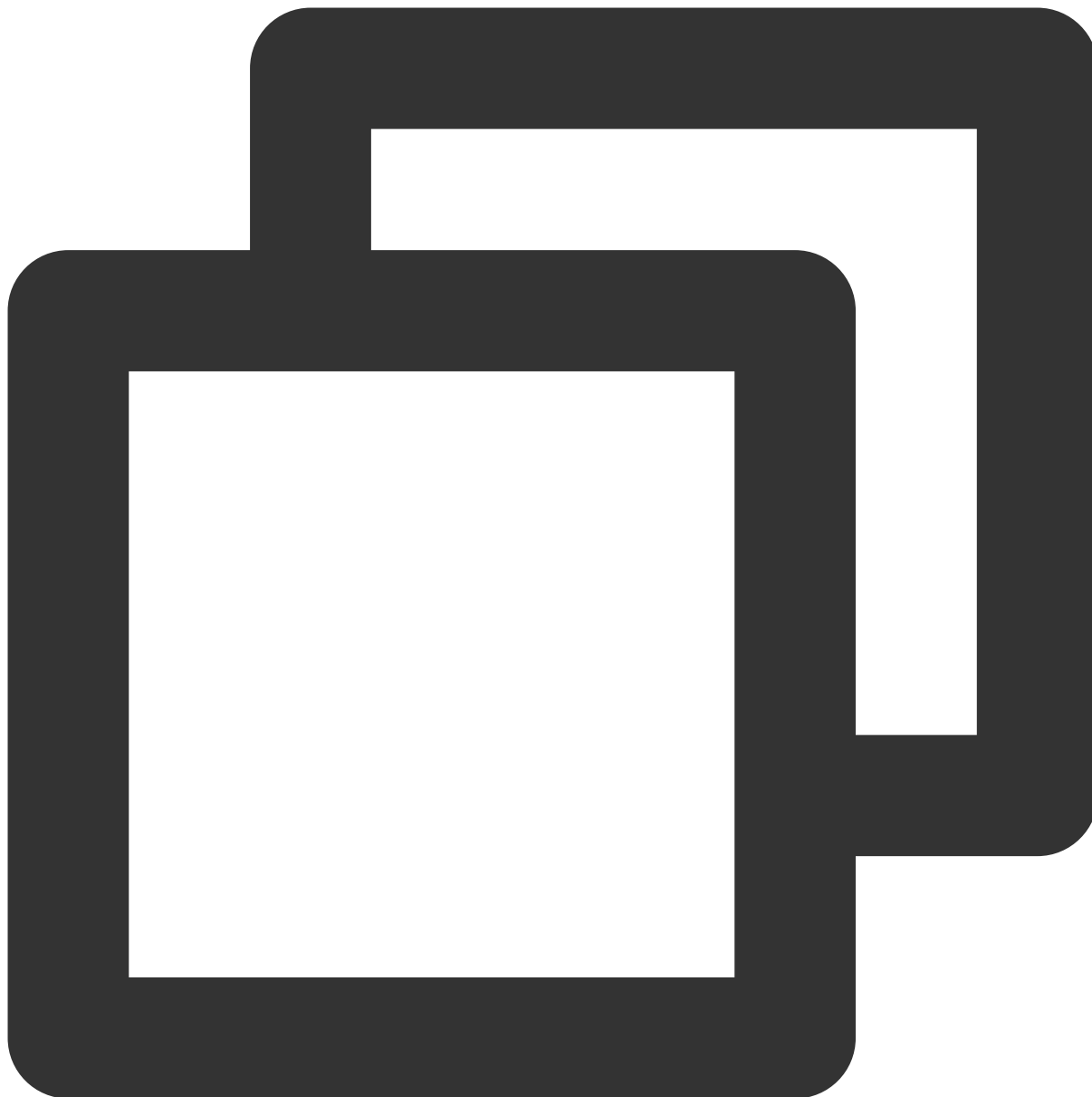
```
- (void)onError:(int)code  
           message:(NSString*)message  
NS_SWIFT_NAME(onError(code:message:));
```

The parameters are described below:

Parameter	Type	Description
code	int	Error code
message	NSString	Error message

## onWarning

Callback for warning.



```
- (void)onWarning:(int)code
    message:(NSString *)message
NS_SWIFT_NAME(onWarning(code:message:));
```

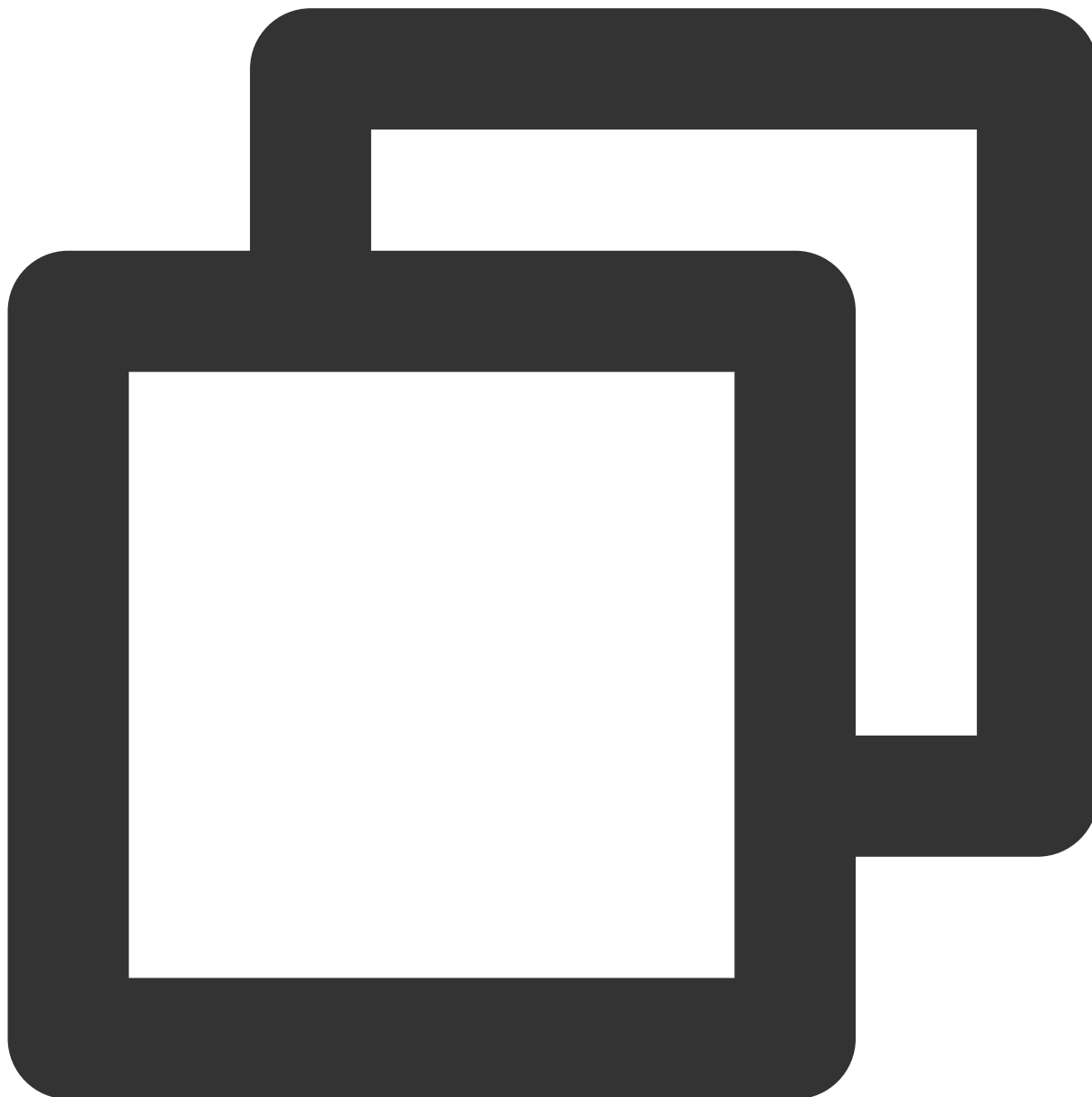
The parameters are described below:

Parameter	Type	Description

code	int	Error code
message	NSString	Warning message

## onDebugLog

Callback for log.



```
- (void)onDebugLog:(NSString *)message  
NS_SWIFT_NAME(onDebugLog(message:)) ;
```



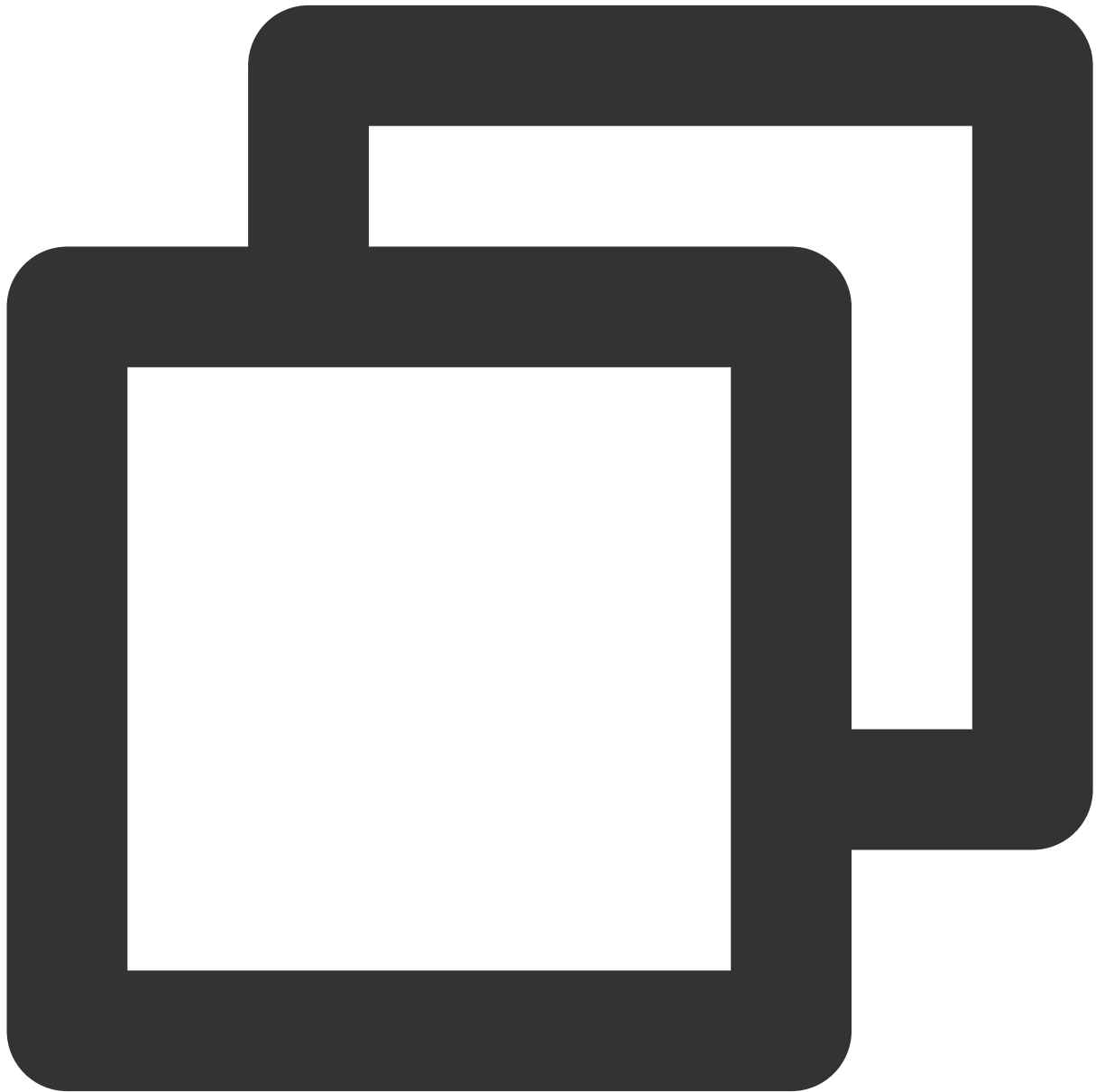
The parameters are described below:

Parameter	Type	Description
message	NSString	Log information

## Room Event Callback APIs

### **onRoomDestroy**

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.



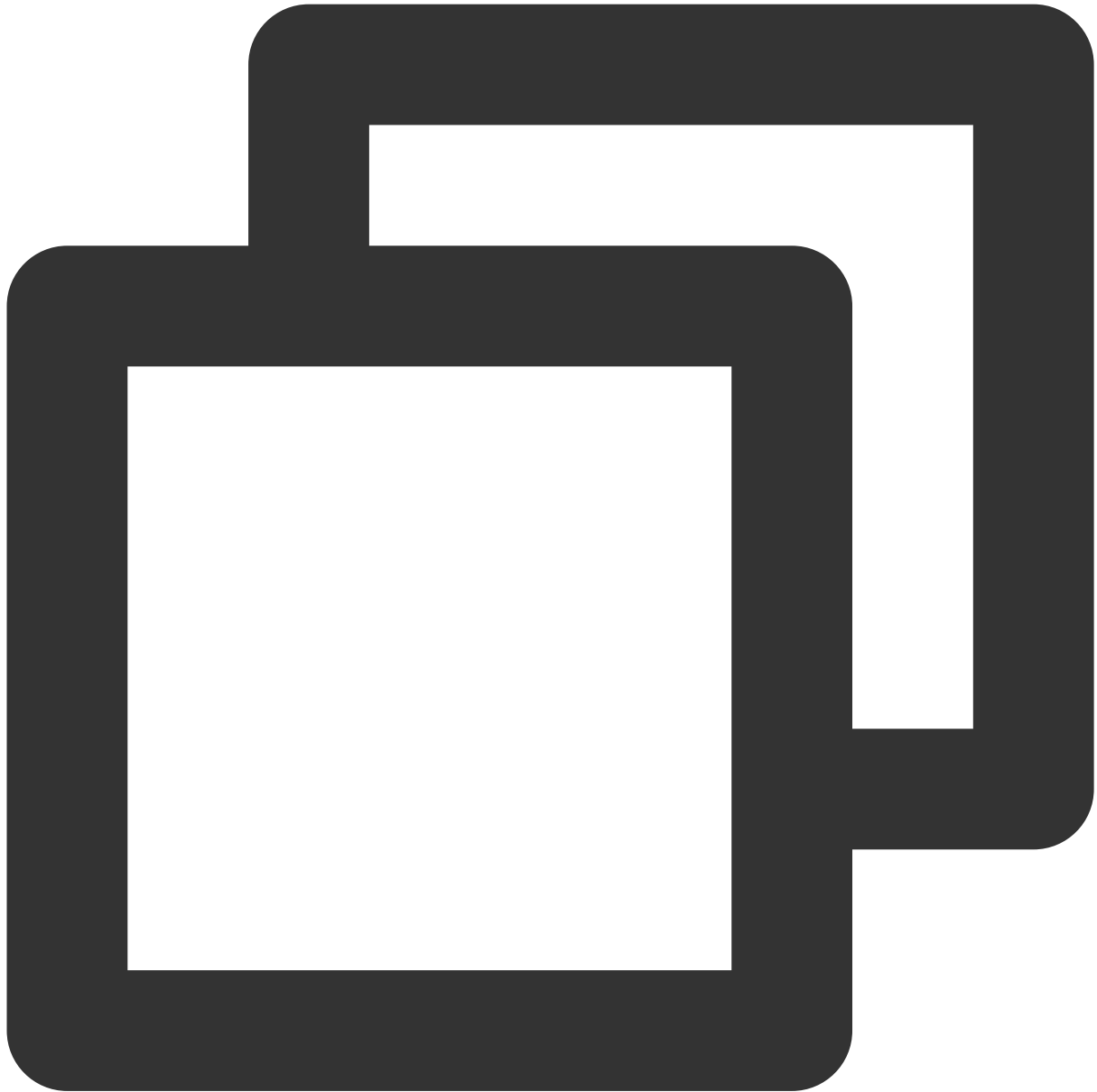
```
- (void)onRoomDestroy:(NSString *)roomId  
NS_SWIFT_NAME(onRoomDestroy(roomId:));
```

The parameters are described below:

Parameter	Type	Description
roomId	NSString	Room ID

## onRoomInfoChange

Callback for change of room information. This callback is sent after successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.



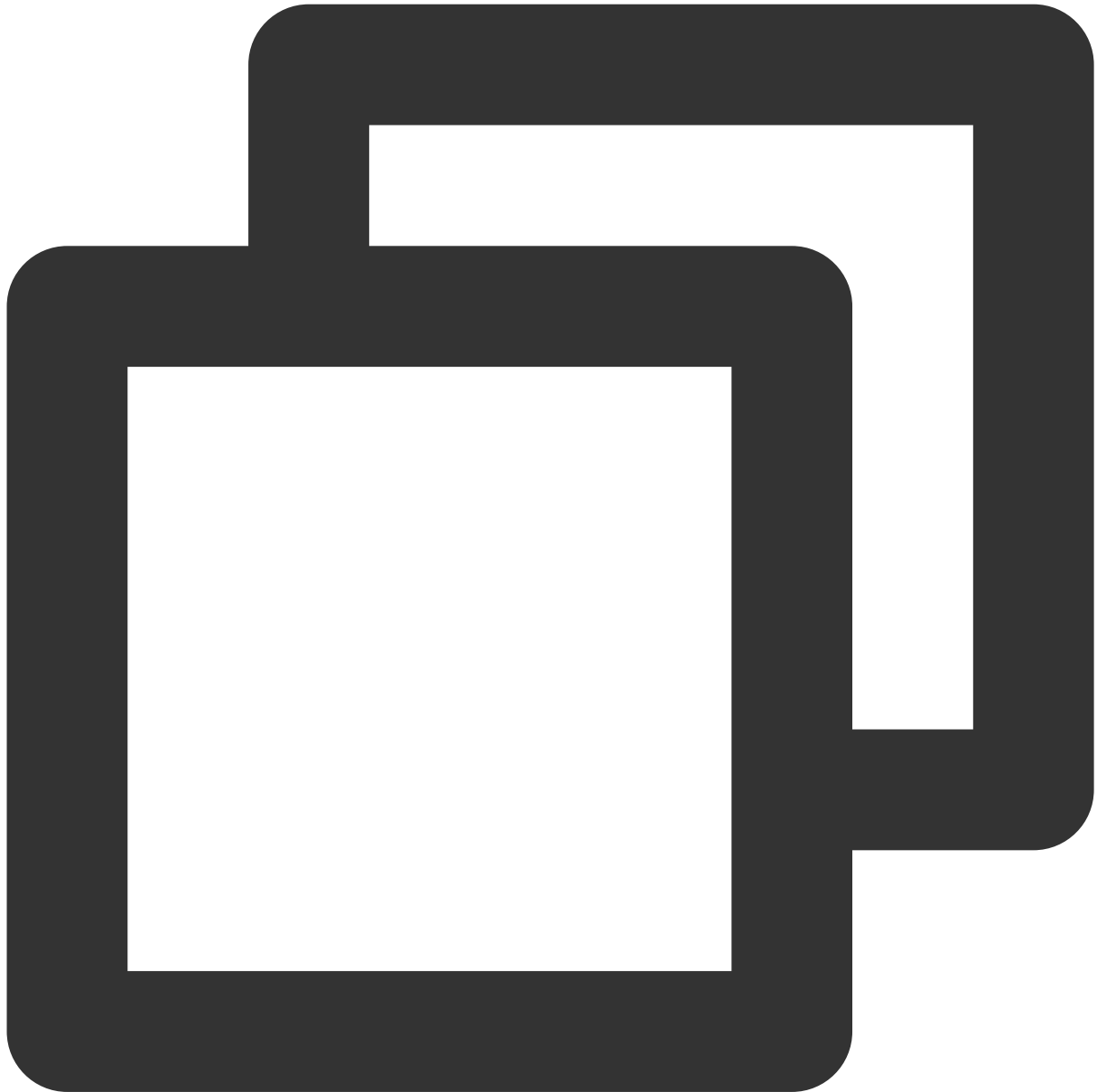
```
- (void)onRoomInfoChange:(VoiceRoomInfo *)roomInfo  
NS_SWIFT_NAME(onRoomInfoChange(roomInfo:));
```

The parameters are described below:

Parameter	Type	Description
roomInfo	VoiceRoomInfo	Room information

## onUserMicrophoneMute

Callback of whether a user's mic is muted. When a user calls `muteLocalAudio`, all members in the room will receive this callback.



```
- (void)onUserMicrophoneMute:(NSString *)userId mute:(BOOL)mute  
NS_SWIFT_NAME(onUserMicrophoneMute(userId:mute:));
```

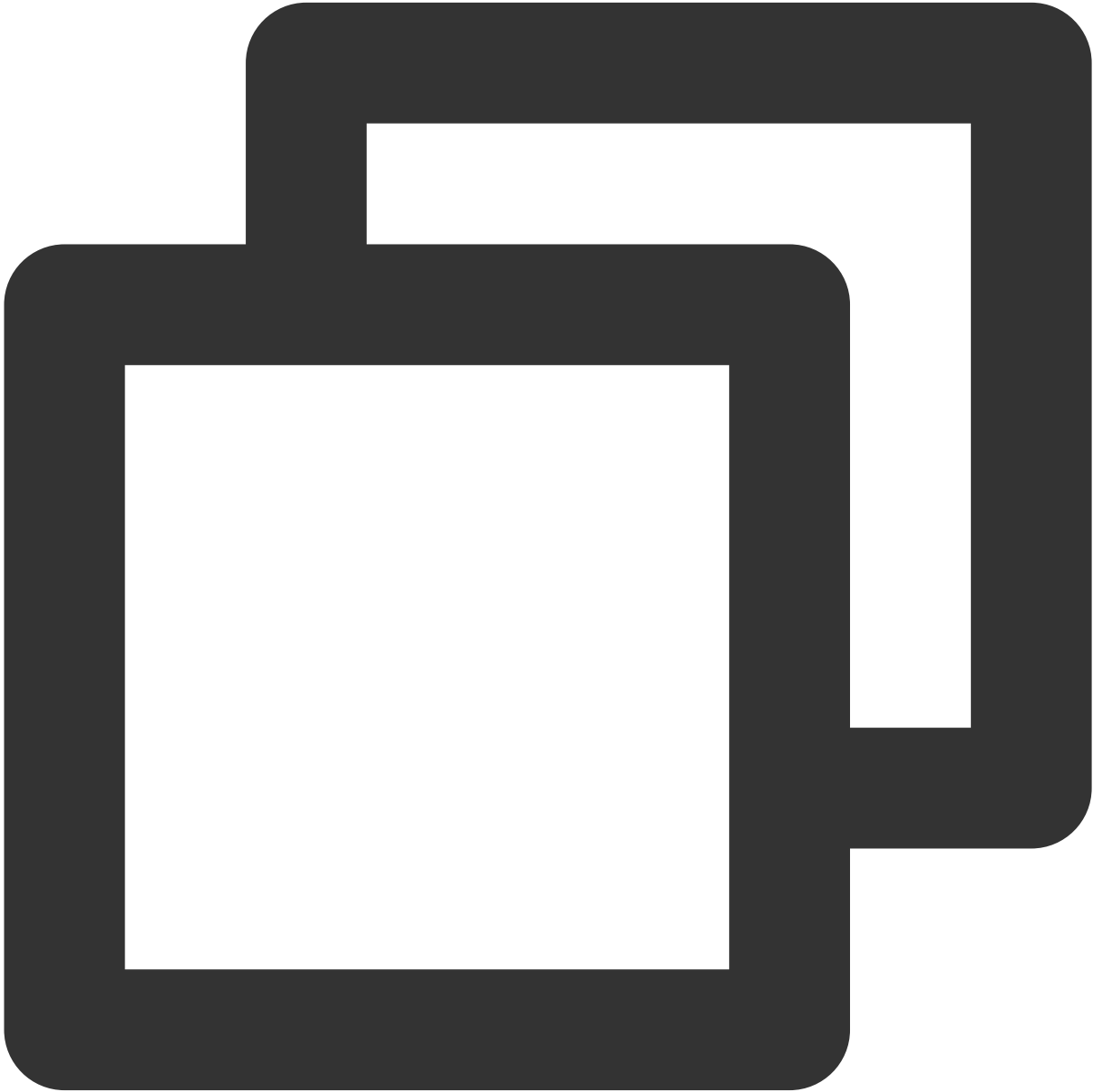
The parameters are described below:

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

userId	NSString	User ID
mute	BOOL	<code>YES</code> : muted; <code>NO</code> : unmuted

**onUserVolumeUpdate**

Notification to all members of the volume after the volume reminder is enabled.



```
- (void)onUserVolumeUpdate:(NSArray<TRTCVolumeInfo *> *)userVolumes totalVolume:(NS
NS_SWIFT_NAME (onUserVolumeUpdate (userVolumes:totalVolume:));
```

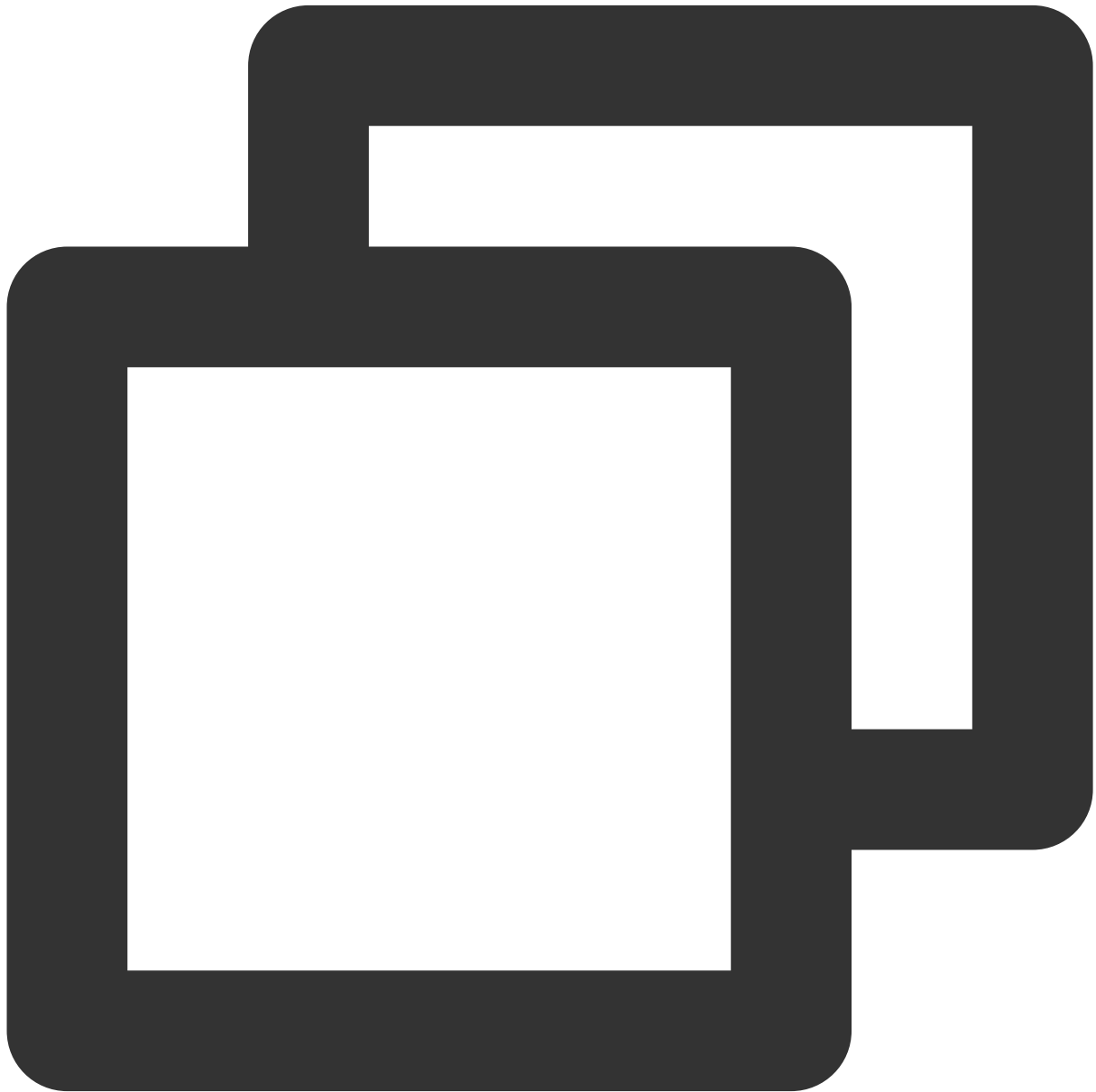
The parameters are described below:

Parameter	Type	Description
userVolumes	NSArray	List of user volumes
totalVolume	NSInteger	Total volume. Value range: 0-100

## Seat Callback APIs

### **onSeatListChange**

Callback for all seat changes.



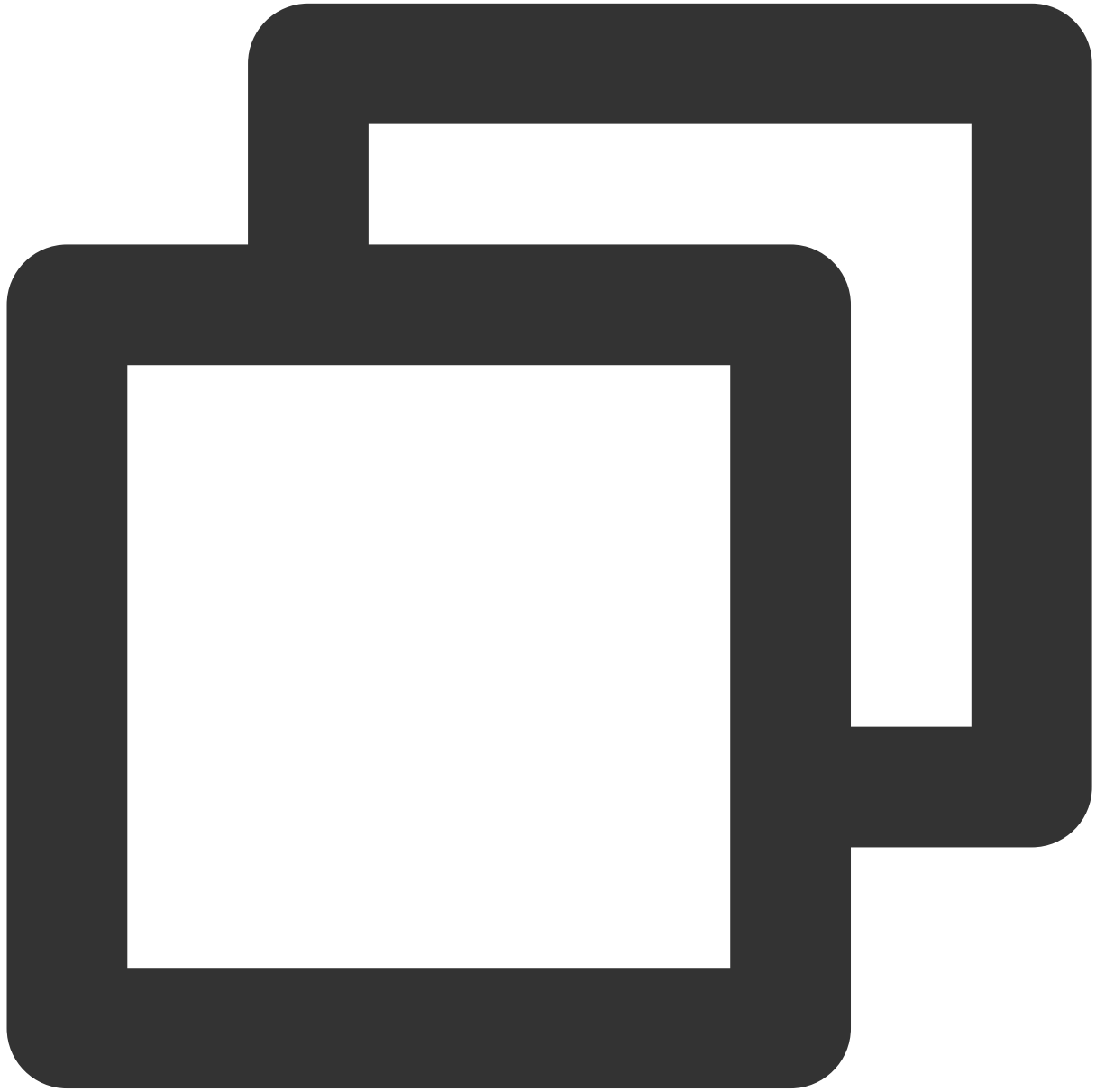
```
- (void)onSeatInfoChange:(NSArray<VoiceRoomSeatInfo *> *)seatInfoList  
NS_SWIFT_NAME(onSeatListChange(seatInfoList:));
```

The parameters are described below:

Parameter	Type	Description
seatInfoList	NSArray<VoiceRoomSeatInfo>	Full seat list

## onAnchorEnterSeat

Someone became a speaker or was made a speaker by the owner.



```
- (void)onAnchorEnterSeat:(NSInteger)index
                        user:(VoiceRoomUserInfo *)user
NS_SWIFT_NAME(onAnchorEnterSeat(index:user:));
```

The parameters are described below:

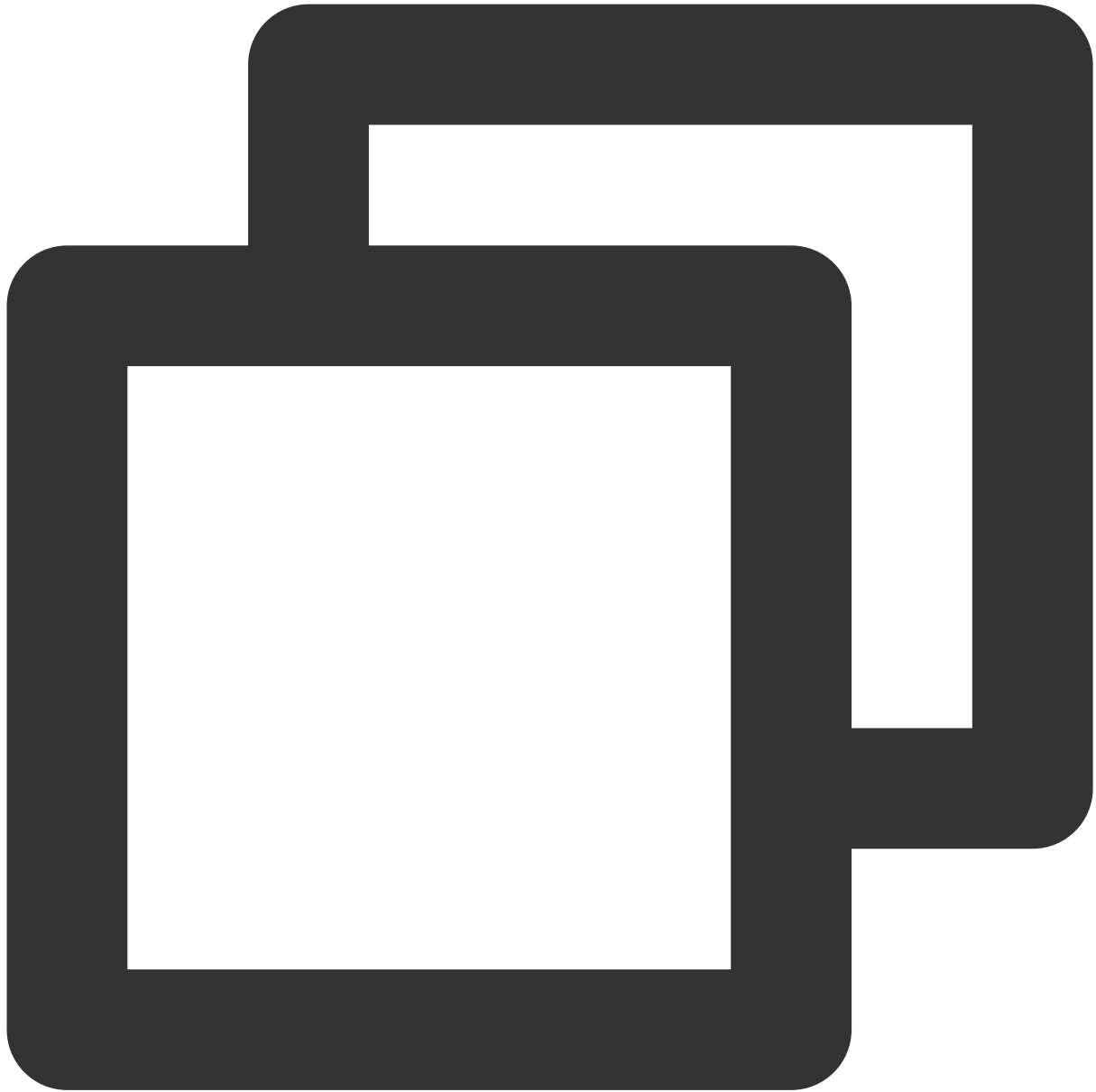
Parameter	Type	Description
index	NSInteger	Number of the seat taken



user	VoiceRoomUserInfo	Information of the user who left the seat
------	-------------------	---

## onAnchorLeaveSeat

A speaker became a listener or was moved to listeners by the room owner.



```
- (void)onAnchorLeaveSeat:(NSInteger)index
    user:(VoiceRoomUserInfo *)user
NS_SWIFT_NAME(onAnchorLeaveSeat(index:user:));
```

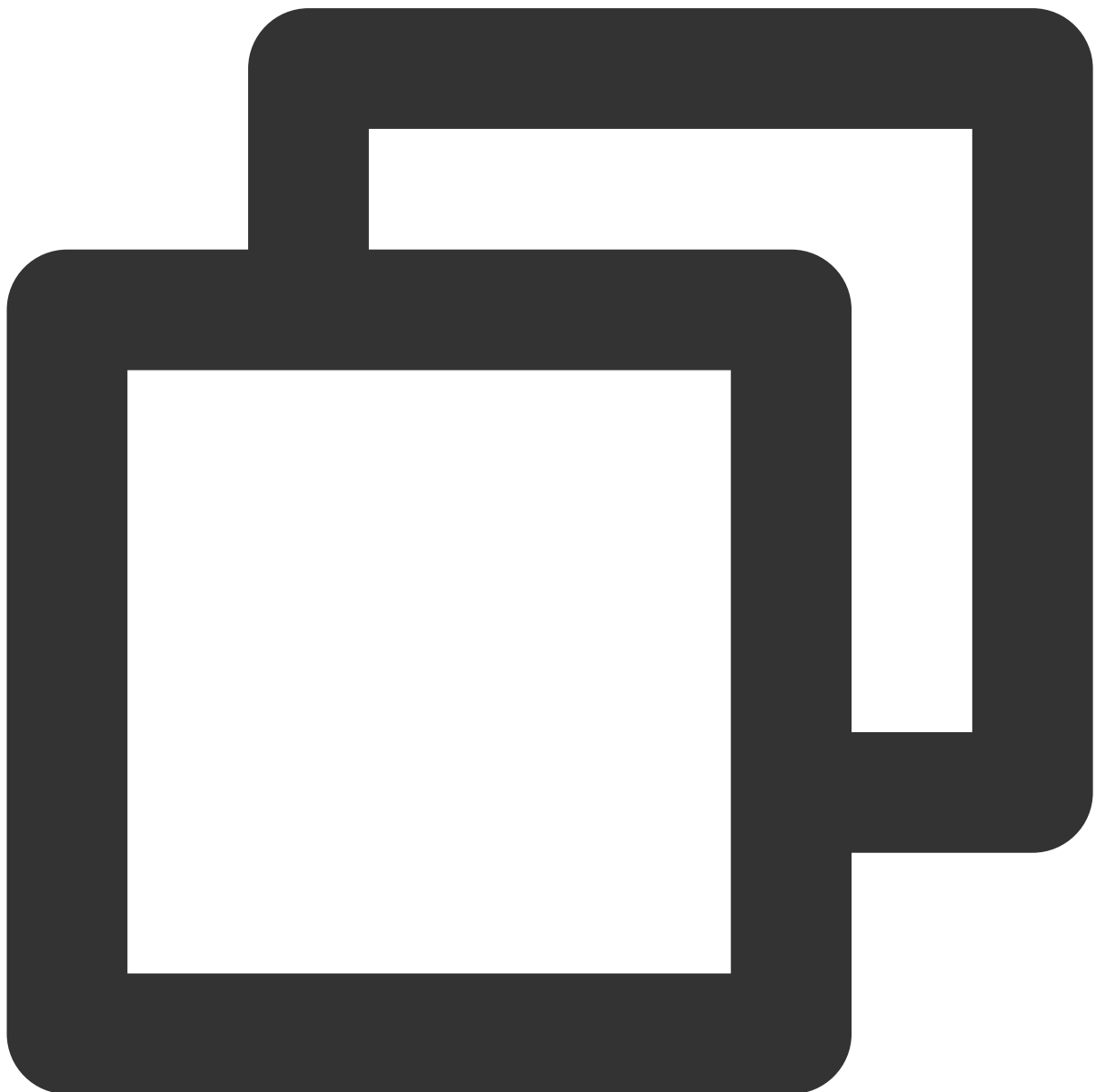
The parameters are described below:

--	--	--

Parameter	Type	Description
index	NSInteger	Number of the seat the user left
user	VoiceRoomUserInfo	Information of the user who left the seat

## onSeatMute

The room owner muted/unmuted a seat.



```
- (void)onSeatMute:(NSInteger) index
```

```
isMute: (BOOL) isMute
NS_SWIFT_NAME (onSeatMute (index: isMute:));
```

The parameters are described below:

Parameter	Type	Description
index	NSInteger	The seat muted/unmuted
isMute	BOOL	YES: The seat was muted; NO: The seat was unmuted.

## onSeatClose

The room owner blocked/unblocked a seat.



```
- (void)onSeatClose:(NSInteger)index  
    isClose:(BOOL)isClose  
NS_SWIFT_NAME(onSeatClose(index:isClose:));
```

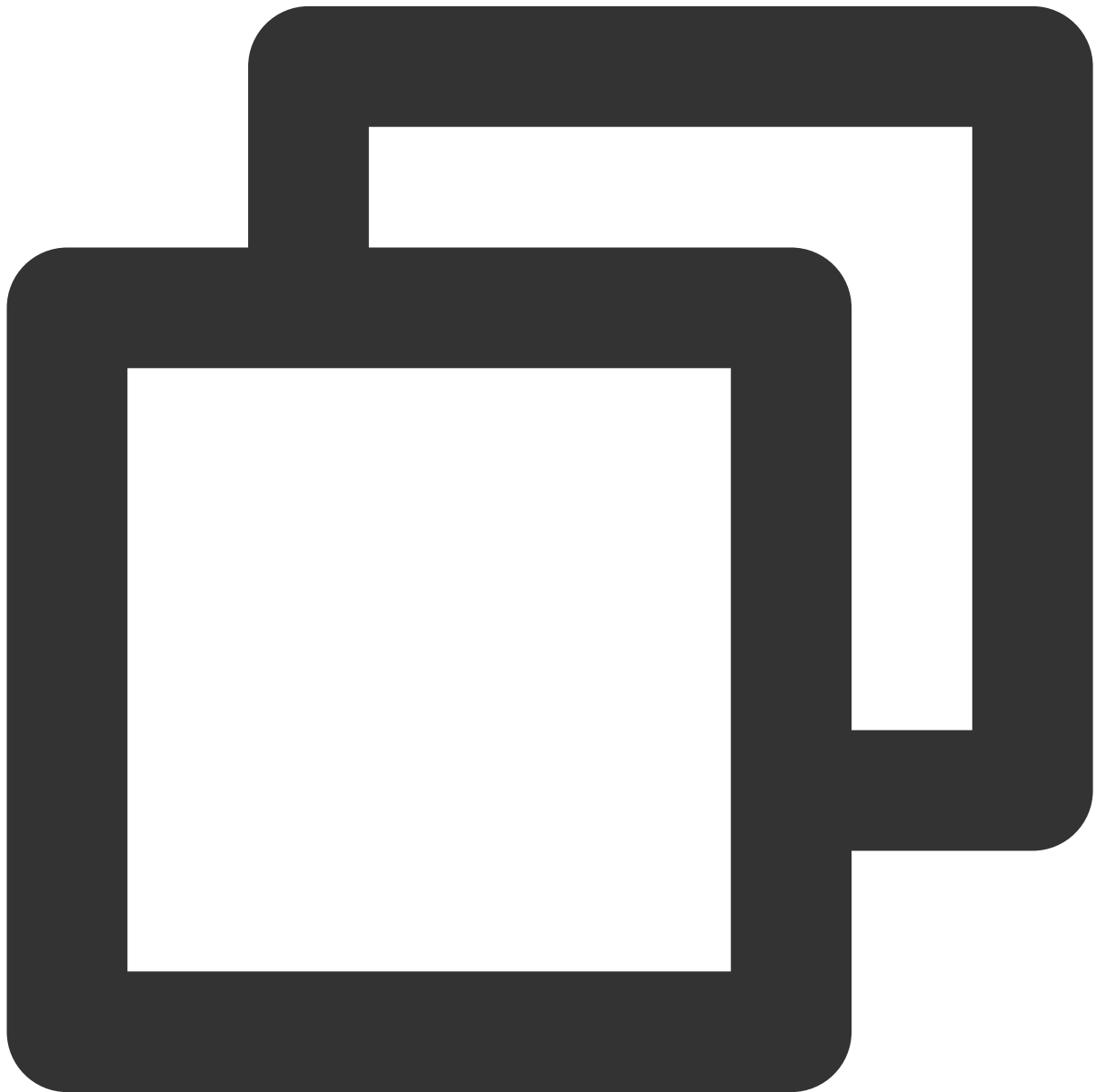
The parameters are described below:

Parameter	Type	Description
index	NSInteger	The seat blocked/unblocked
isClose	BOOL	<code>YES</code> : The seat was blocked; <code>NO</code> : The seat was unblocked.

## Callback APIs for Room Entry/Exit by Listener

### **onAudienceEnter**

A listener entered the room.



```
- (void)onAudienceEnter:(VoiceRoomUserInfo *)userInfo  
NS_SWIFT_NAME(onAudienceEnter(userInfo:));
```

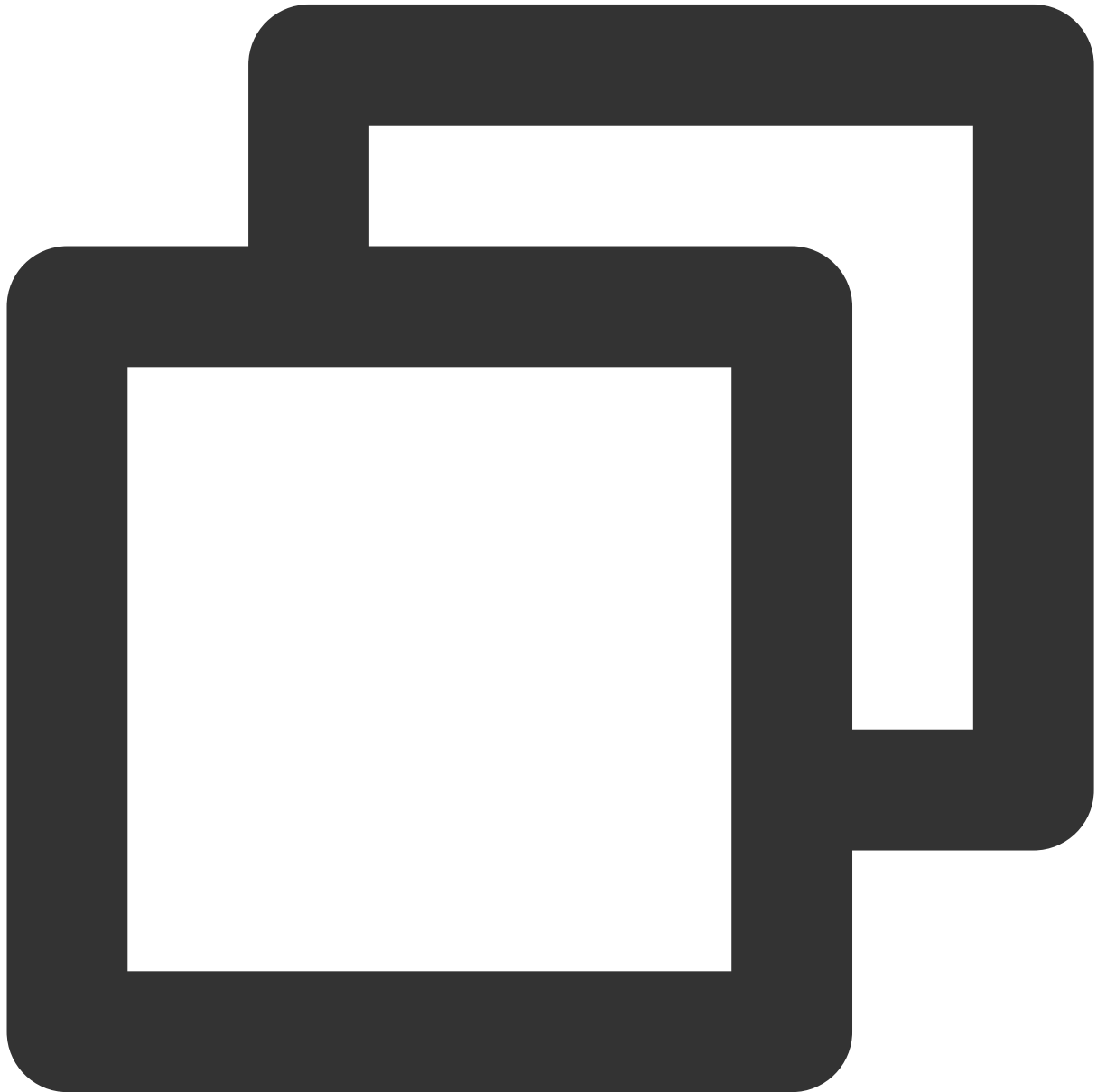
The parameters are described below:

--	--	--

Parameter	Type	Description
userInfo	VoiceRoomUserInfo	Information of the listener who entered

## onAudienceExit

A listener exited the room.



```
- (void)onAudienceExit:(VoiceRoomUserInfo *)userInfo  
NS_SWIFT_NAME(onAudienceExit(userInfo:));
```

The parameters are described below:

Parameter	Type	Description
userInfo	VoiceRoomUserInfo	Information of the user who left

## Message Event Callback APIs

### **onRecvRoomTextMsg**

Callback for receiving a text chat message.



```
- (void)onRecvRoomTextMsg:(NSString *)message
    userInfo:(VoiceRoomUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomTextMsg(message:userInfo:));
```

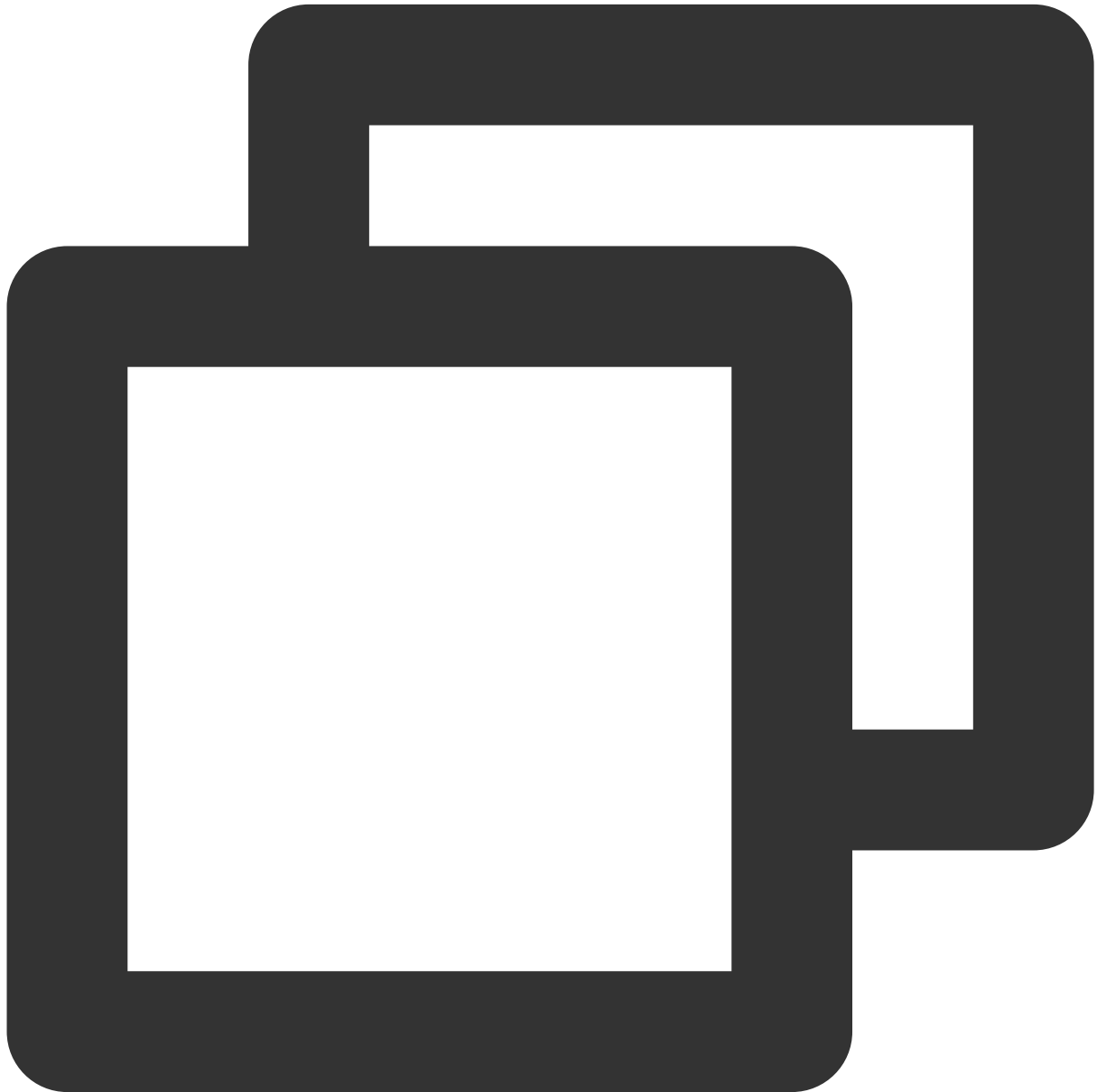
The parameters are described below:

Parameter	Type	Description
message	NSString	Text message
userInfo	VoiceRoomUserInfo	Information of the sender



## onRecvRoomCustomMsg

A custom message was received.



```
- (void)onRecvRoomCustomMsg:(NSString *)command
                        message:(NSString *)message
                userInfo:(VoiceRoomUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomCustomMsg(command:message:userInfo:));
```

The parameters are described below:

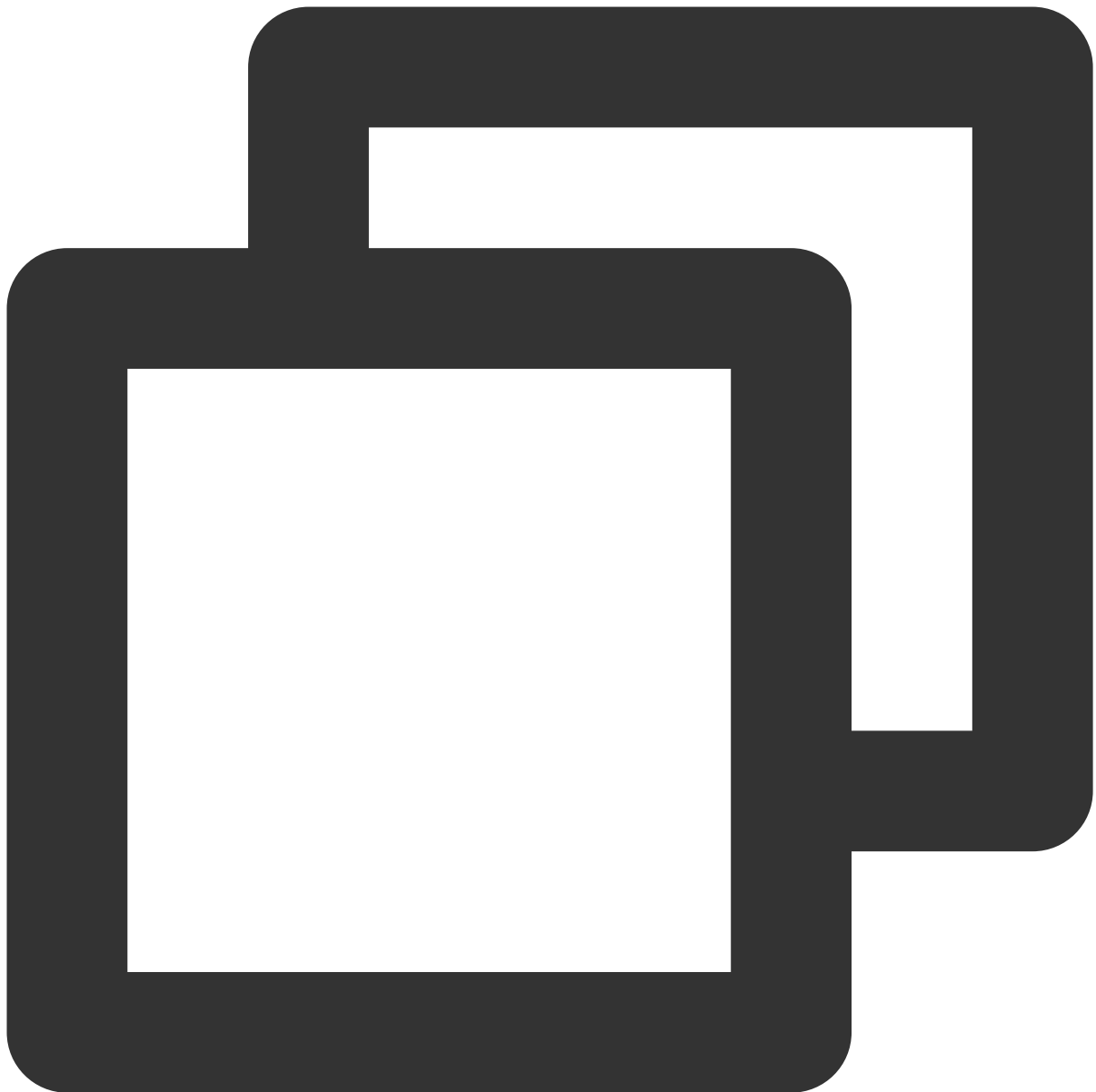
Parameter	Type	Description

command	NSString	Custom command word used to distinguish between different message types
message	NSString	Text message
userInfo	VoiceRoomUserInfo	Information of the sender

## Invitation Signaling Callback APIs

### **onReceiveNewInvitation**

An invitation was received.



```
- (void)onReceiveNewInvitation:(NSString *)identifier
    inviter:(NSString *)inviter
    cmd:(NSString *)cmd
    content:(NSString *)content
NS_SWIFT_NAME (onReceiveNewInvitation(id:inviter:cmd:content:));
```

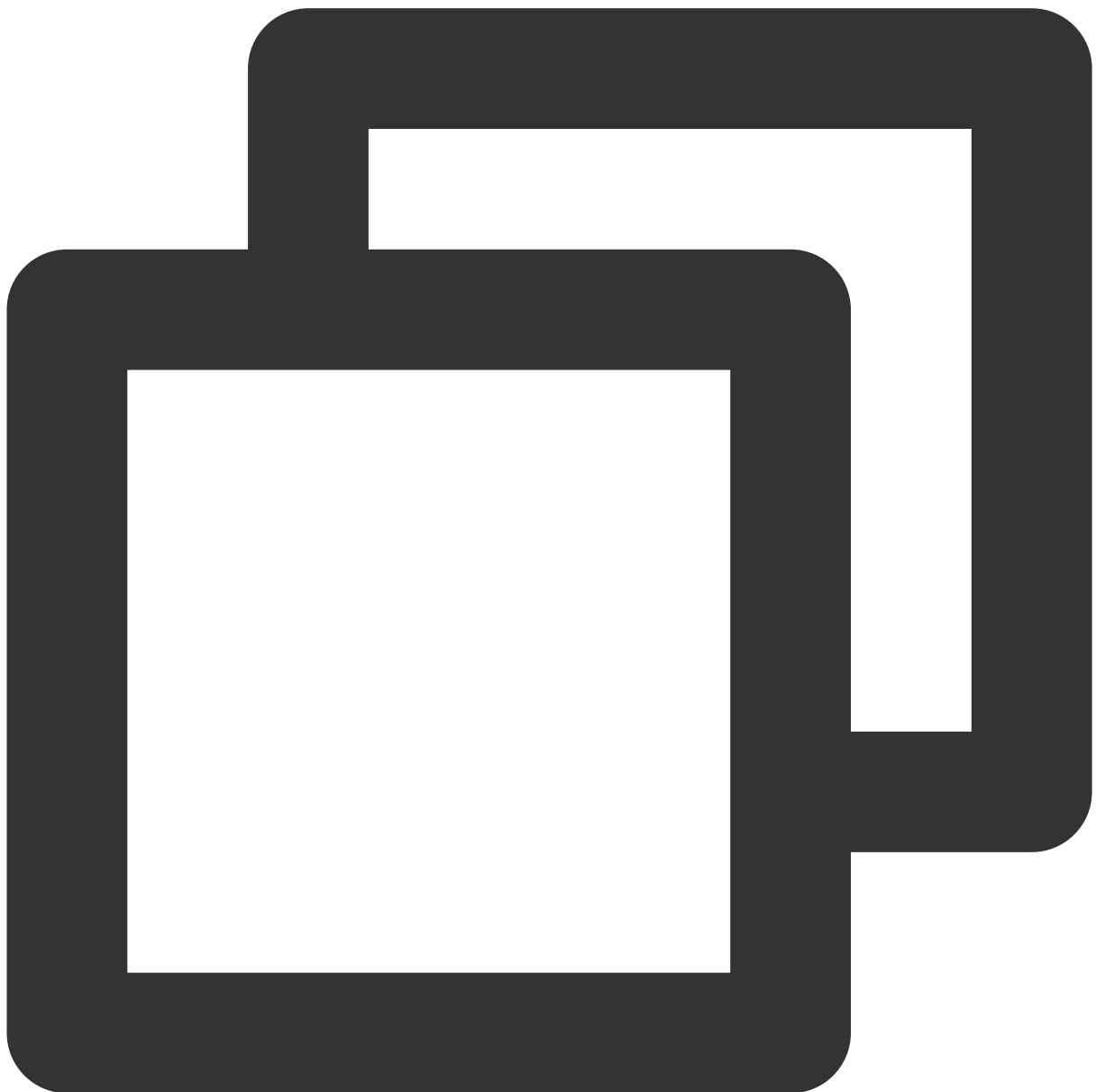
The parameters are described below:

Parameter	Type	Description
id	NSString	Invitation ID

inviter	NSString	Inviter's user ID
cmd	NSString	Custom command word specified by business
content	NSString	Content specified by business

### **onInviteeAccepted**

The invitee accepted the invitation



```
- (void)onInviteeAccepted:(NSString *)identifier
```

```
invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeAccepted(id:invitee:));
```

The parameters are described below:

Parameter	Type	Description
id	NSString	Invitation ID
invitee	NSString	Invitee's user ID

### **onInviteeRejected**

The invitee declined the invitation



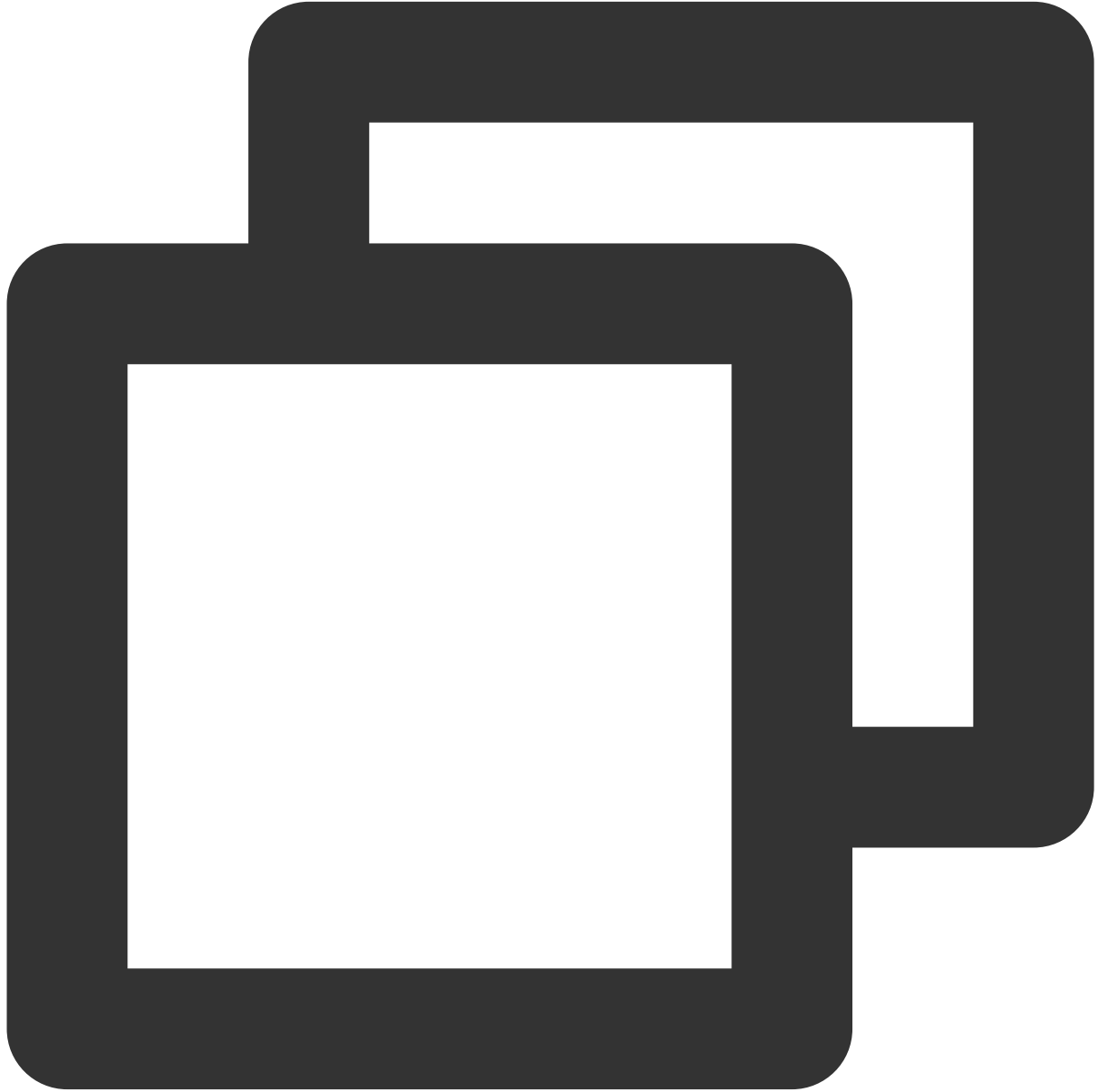
```
- (void)onInviteeRejected:(NSString *)identifier
    invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeRejected(id:invitee:));
```

The parameters are described below:

Parameter	Type	Description
id	NSString	Invitation ID
invitee	NSString	Invitee's user ID

## onInvitationCancelled

The inviter canceled the invitation.



```
- (void)onInvitationCancelled:(NSString *)identifier  
    invitee:(NSString *)invitee NS_SWIFT_NAME(onInvitationCancelled)
```

The parameters are described below:

Parameter	Type	Description
id	NSString	Invitation ID

inviter	NSString	Inviter's user ID
---------	----------	-------------------



# TRTCVoiceRoom (Android)

Last updated : 2023-09-25 10:53:10

`TRTCVoiceRoom` is based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat. With `TRTCVoiceRoom`:

A user can create an audio chat room and become a speaker, or enter an audio chat room as a listener.

The room owner can invite a listener to speak as well as remove a speaker from a seat.

The room owner can also block a seat. A listener cannot request to take a blocked seat to become a speaker.

A listener can request to speak and become a speaker. A speaker can also become a listener.

All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

## Note

All TUIKit components are based on two PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For billing details of Chat, see [Pricing](#).

`TRTCVoiceRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, see [Audio Chat Room \(Android\)](#).

The [TRTC SDK](#) is used as a low-latency audio chat component.

The `AVChatRoom` feature of the [Chat SDK](#) is used to implement chat rooms. The attribute APIs of Chat are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

## TRTCVoiceRoom API Overview

### Basic SDK APIs

API	Description
<a href="#">sharedInstance</a>	Gets a singleton object.
<a href="#">destroySharedInstance</a>	Terminates a singleton object.
<a href="#">setDelegate</a>	Sets event callbacks.
<a href="#">setDelegateHandler</a>	Sets the thread where event callbacks are.
<a href="#">login</a>	Logs in.

<a href="#">logout</a>	Logs out.
<a href="#">setSelfProfile</a>	Sets profile.

## Room APIs

API	Description
<a href="#">createRoom</a>	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
<a href="#">destroyRoom</a>	Terminates a room (called by room owner).
<a href="#">enterRoom</a>	Enters a room (called by listener).
<a href="#">exitRoom</a>	Exits a room (called by listener).
<a href="#">getRoomInfoList</a>	Gets room list details.
<a href="#">getUserInfoList</a>	Gets the user information of the specified <code>userId</code> . If the value is <code>null</code> , the information of all users in the room is obtained.

## Seat management APIs

API	Description
<a href="#">enterSeat</a>	Becomes a speaker (called by room owner or listener).
<a href="#">moveSeat</a>	Changes the seat (called by speaker).
<a href="#">leaveSeat</a>	Becomes a listener (called by speaker).
<a href="#">pickSeat</a>	Places a user in a seat (called by room owner).
<a href="#">kickSeat</a>	Removes a speaker (called by room owner).
<a href="#">muteSeat</a>	Mutes/Unmutes a seat (called by room owner).
<a href="#">closeSeat</a>	Blocks/Unblocks a seat (called by room owner).

## Local audio APIs

API	Description
<a href="#">startMicrophone</a>	Starts mic capturing.
<a href="#">stopMicrophone</a>	Stops mic capturing.

<a href="#">setAudioQuality</a>	Sets audio quality.
<a href="#">muteLocalAudio</a>	Mutes/Unmutes local audio.
<a href="#">setSpeaker</a>	Sets whether to play sound from the device's speaker or receiver.
<a href="#">setAudioCaptureVolume</a>	Sets mic capturing volume.
<a href="#">setAudioPlayOutVolume</a>	Sets playback volume.
<a href="#">setVoiceEarMonitorEnable</a>	Enables/Disables in-ear monitoring.

## Remote audio APIs

API	Description
<a href="#">muteRemoteAudio</a>	Mutes/Unmutes a specified member.
<a href="#">muteAllRemoteAudio</a>	Mutes/Unmutes all members.

## Background music and audio effect APIs

API	Description
<a href="#">getAudioEffectManager</a>	Gets the background music and audio effect management object <a href="#">TXAudioEffectManager</a> .

## Message sending APIs

API	Description
<a href="#">sendRoomTextMsg</a>	Broadcasts a text chat message in a room. This API is generally used for on-screen comments.
<a href="#">sendRoomCustomMsg</a>	Sends a custom text message.

## Invitation signaling APIs

API	Description
<a href="#">sendInvitation</a>	Sends an invitation.
<a href="#">acceptInvitation</a>	Accepts an invitation.
<a href="#">rejectInvitation</a>	Declines an invitation.

[cancelInvitation](#)

Cancels an invitation.

## TRTCVoiceRoomDelegate API Overview

### Common event callbacks

API	Description
<a href="#">onError</a>	Callback for error.
<a href="#">onWarning</a>	Callback for warning.
<a href="#">onDebugLog</a>	Callback of log.

### Room event callback APIs

API	Description
<a href="#">onRoomDestroy</a>	The room was terminated.
<a href="#">onRoomInfoChange</a>	The room information changed.
<a href="#">onUserVolumeUpdate</a>	User volume

### Seat list change callback APIs

API	Description
<a href="#">onSeatListChange</a>	All seat changes
<a href="#">onAnchorEnterSeat</a>	Someone became a speaker or was made a speaker by the room owner.
<a href="#">onAnchorLeaveSeat</a>	Someone became a listener or was moved to listeners by the room owner.
<a href="#">onSeatMute</a>	The room owner muted a seat.
<a href="#">onUserMicrophoneMute</a>	Whether a user's mic is muted
<a href="#">onSeatClose</a>	The room owner blocked a seat.

### Callback APIs for room entry/exit by listener

--	--

API	Description
<a href="#">onAudienceEnter</a>	A listener entered the room.
<a href="#">onAudienceExit</a>	A listener exited the room.

### Message event callback APIs

API	Description
<a href="#">onRecvRoomTextMsg</a>	A text chat message was received.
<a href="#">onRecvRoomCustomMsg</a>	A custom message was received.

## Signaling Event Callback APIs

API	Description
<a href="#">onReceiveNewInvitation</a>	An invitation was received.
<a href="#">onInviteeAccepted</a>	The invitee accepted the invitation.
<a href="#">onInviteeRejected</a>	The invitee declined the invitation.
<a href="#">onInvitationCancelled</a>	The inviter canceled the invitation.

## Basic SDK APIs

### **sharedInstance**

This API is used to get a [TRTCVoiceRoom](#) singleton object.



```
public static synchronized TRTCVoiceRoom sharedInstance(Context context);
```

The parameters are described below:

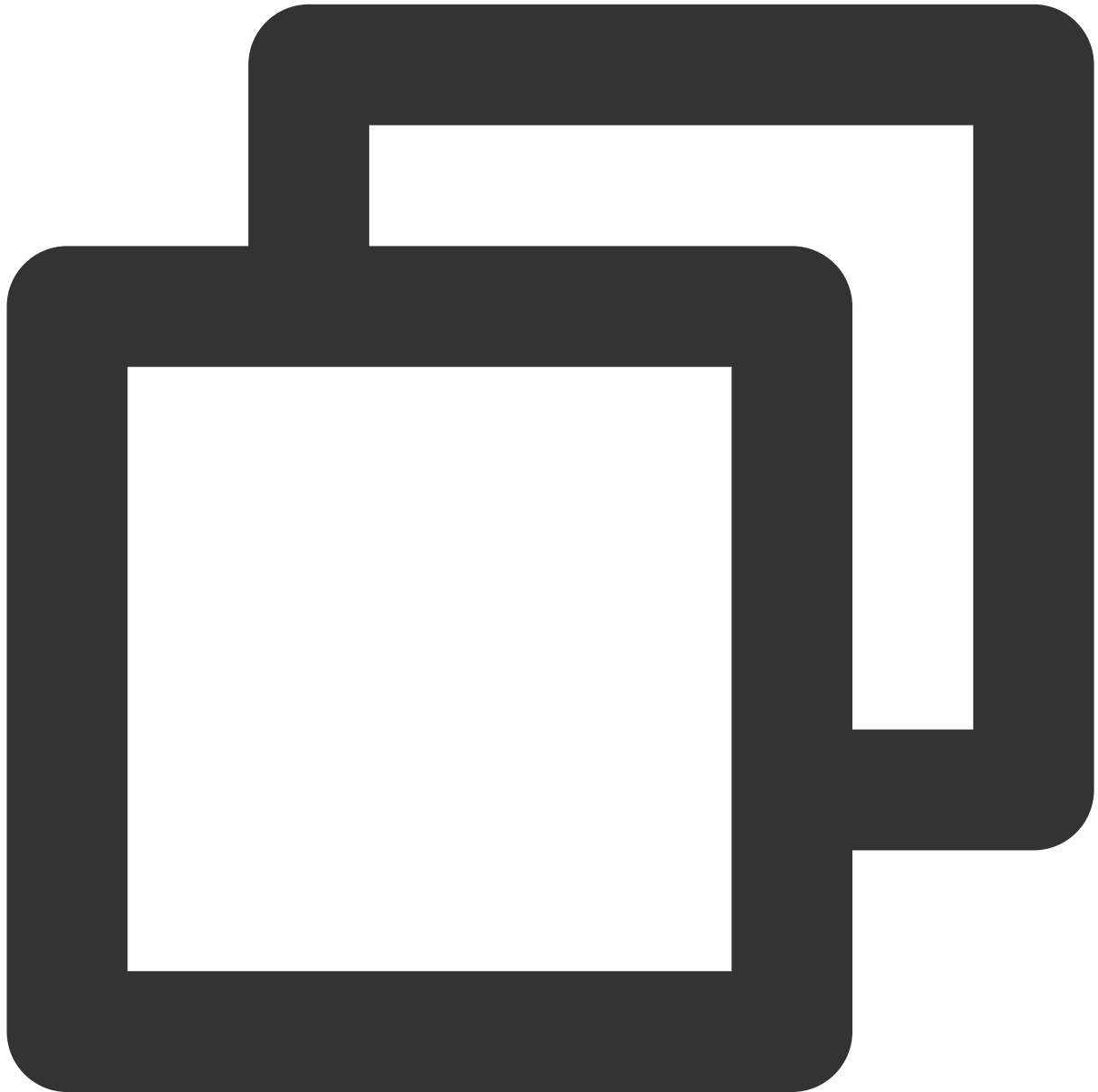
Parameter	Type	Description
context	Context	Android context, which will be converted to <code>ApplicationContext</code> for the calling of system APIs

## **destroySharedInstance**

This API is used to terminate a [TRTCVoiceRoom](#) singleton object.

**Note**

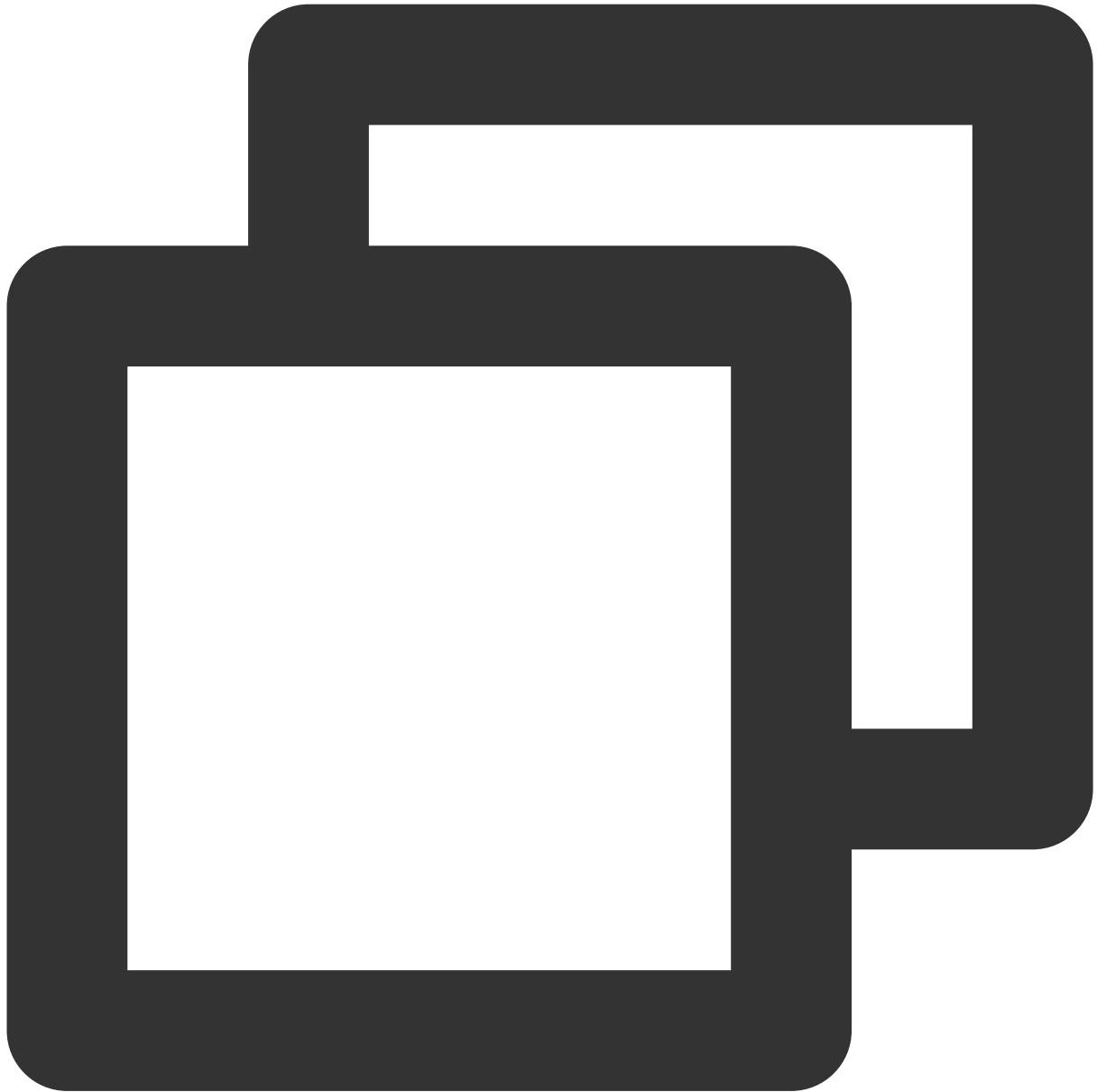
After the instance is terminated, the externally cached `TRTCVoiceRoom` instance can no longer be used. You need to call [sharedInstance](#) again to get a new instance.



```
public static void destroySharedInstance();
```

**setDelegate**

This API is used to set the event callback of [TRTCVoiceRoom](#). You can use `TRTCVoiceRoomDelegate` to get different status notifications of [TRTCVoiceRoom](#).



```
public abstract void setDelegate(TRTCVoiceRoomDelegate delegate);
```

#### Note

`setDelegate` is the delegate callback of `TRTCVoiceRoom` .

#### setDelegateHandler

This API is used to set the thread where event callbacks are.





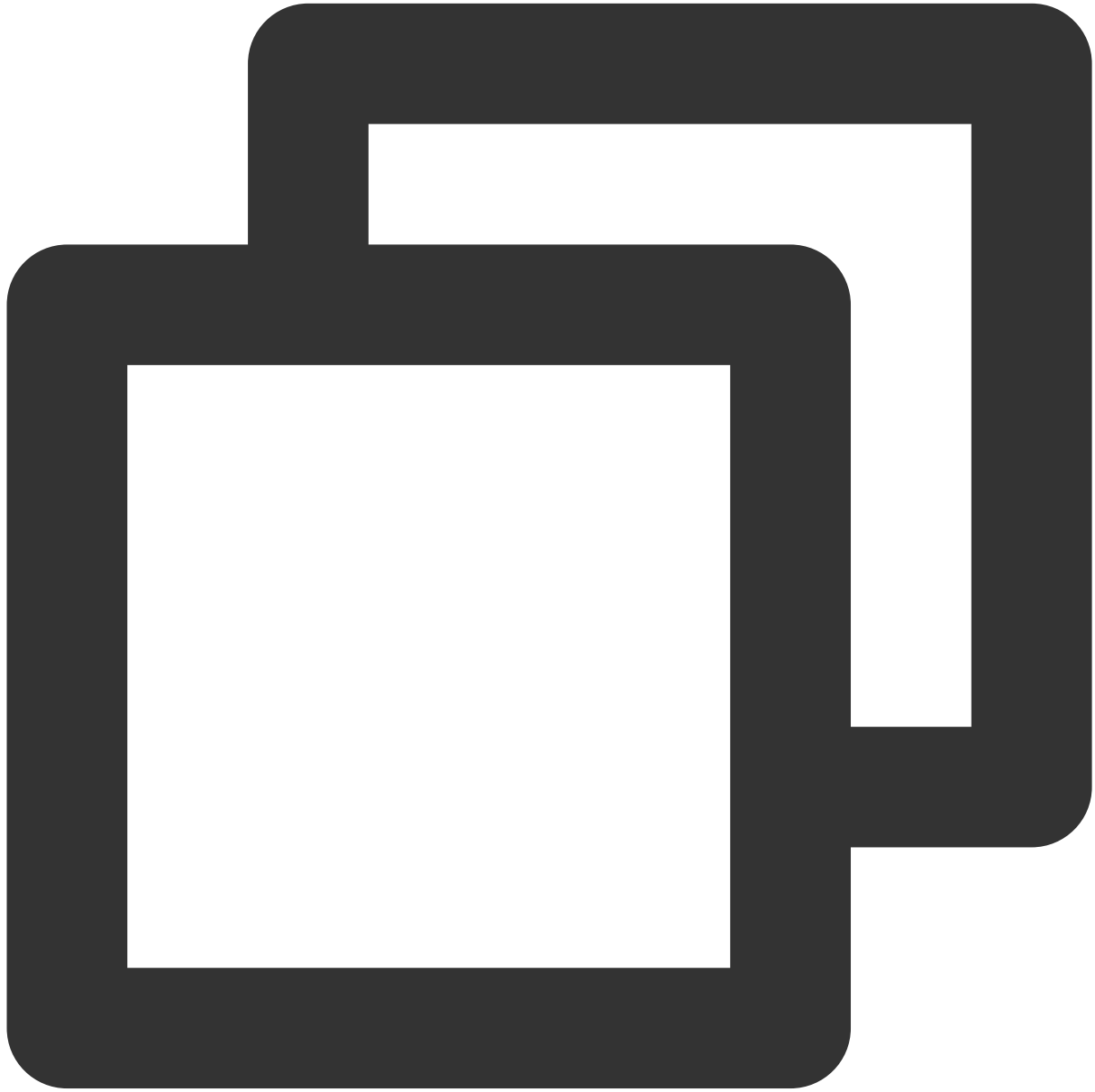
```
public abstract void setDelegateHandler(Handler handler);
```

The parameters are described below:

Parameter	Type	Description
handler	Handler	The status notifications of <code>TRTCVoiceRoom</code> are sent to the handler thread you specify.

## login

Login



```
public abstract void login(int sdkAppId,  
    String userId, String userSig,  
    TRTCVoiceRoomCallback.ActionCallback callback);
```

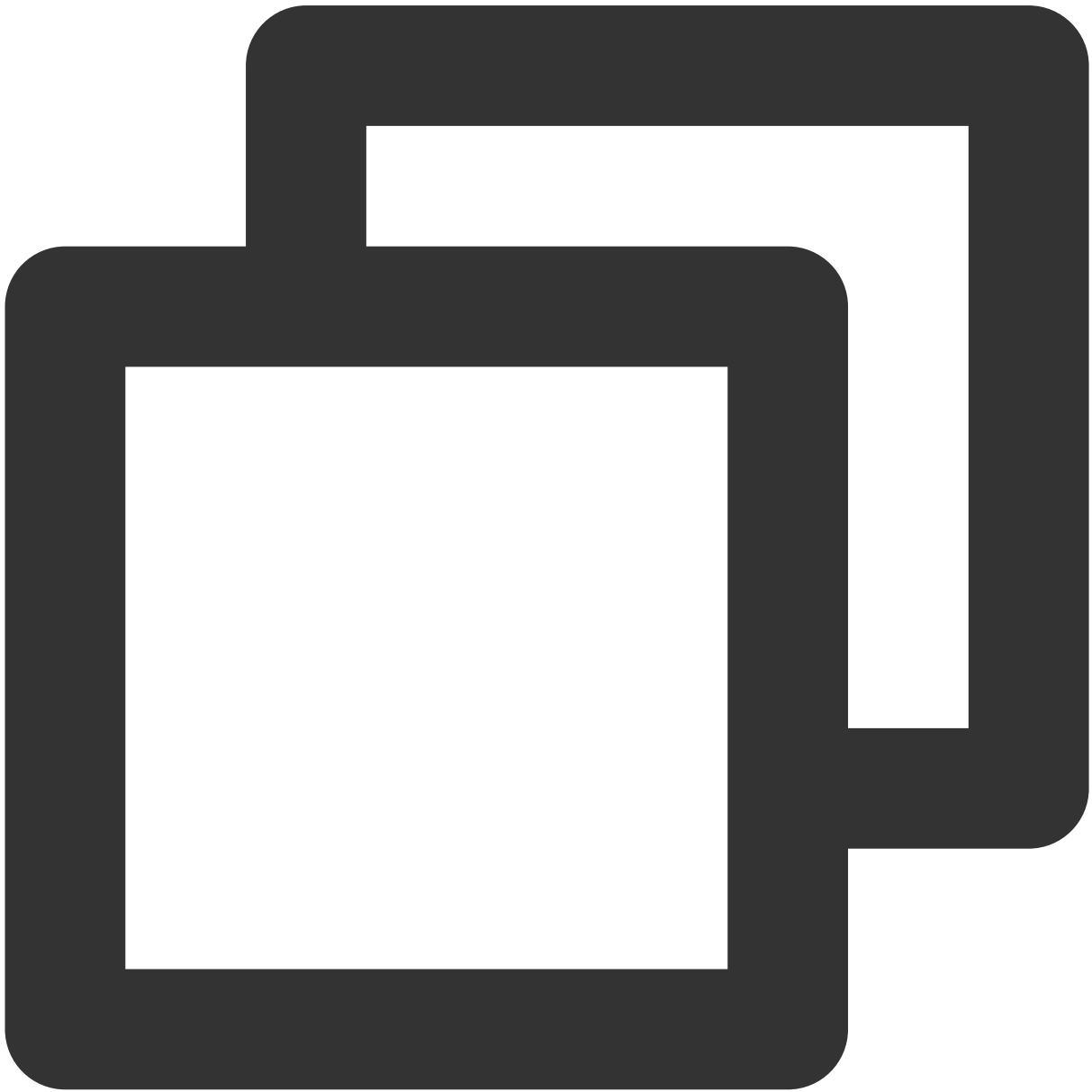
The parameters are described below:

Parameter	Type	Description
sdkAppId	int	You can view <code>SDKAppID</code> in <a href="#">Application Management</a> > <b>Application Info</b> of the TRTC console.

userId	String	The ID of current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_).
userSig	String	Tencent Cloud's proprietary security signature. For how to calculate and use it, see <a href="#">FAQs &gt; UserSig</a> .
callback	ActionCallback	The callback for login. The code is <code>0</code> if login succeeds.

## logout

Log out



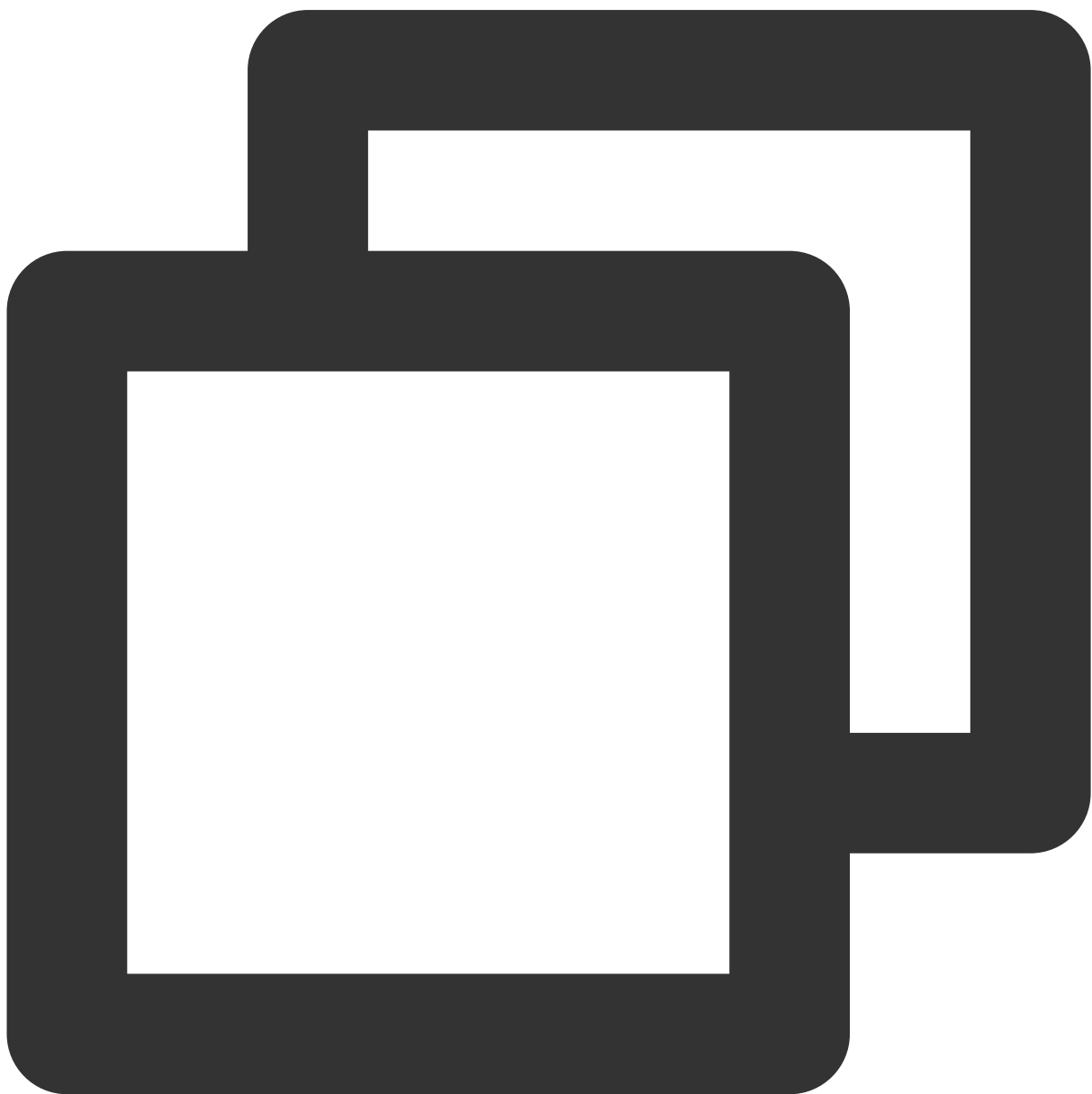
```
public abstract void logout(TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
callback	ActionCallback	The callback for logout. The code is <code>0</code> if logout succeeds.

## setSelfProfile

This API is used to set the profile.



```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCVoiceRoom
```

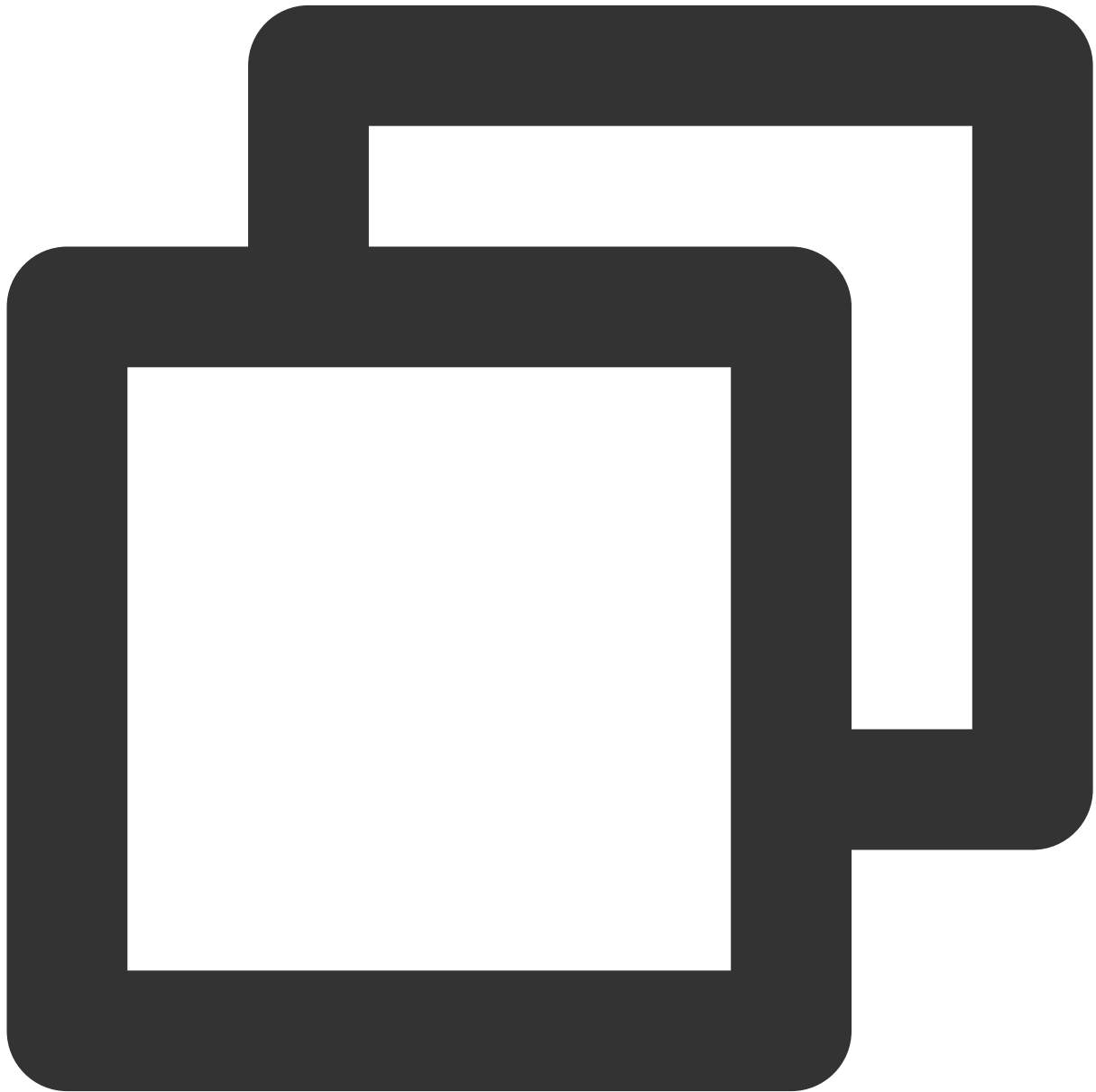
The parameters are described below:

Parameter	Type	Description
userName	String	The username.
avatar	String	The address of the profile photo.
callback	ActionCallback	Callback for profile setting. The code is 0 if the operation succeeds.

## Room APIs

### createRoom

This API is used to create a room (called by room owner).



```
public abstract void createRoom(int roomId, TRTCVoiceRoomDef.RoomParam roomParam, T
```

The parameters are described below:

Parameter	Type	Description
roomId	int	The room ID. You need to assign and manage the IDs in a centralized manner. Multiple <code>roomId</code> values can be aggregated into a karaoke room list. Currently, Tencent Cloud does not provide management services for room lists. Please manage your own room lists.

roomParam	TRTCCreateRoomParam	Room information, such as room name, seat list information, and cover information. To manage seats, you must enter the number of seats in the room.
callback	ActionCallback	Callback for room creation. The code is 0 if the operation succeeds.

The process of creating a karaoke room and becoming a speaker is as follows:

1. A user calls `createRoom` to create an audio chat room, passing in room attributes (e.g. room ID, whether listeners require room owner's consent to speak, number of seats).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
4. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

## **destroyRoom**

This API is used to terminate a room (called by room owner).



```
public abstract void destroyRoom(TRTCVoiceRoomCallback.ActionCallback callback);
```

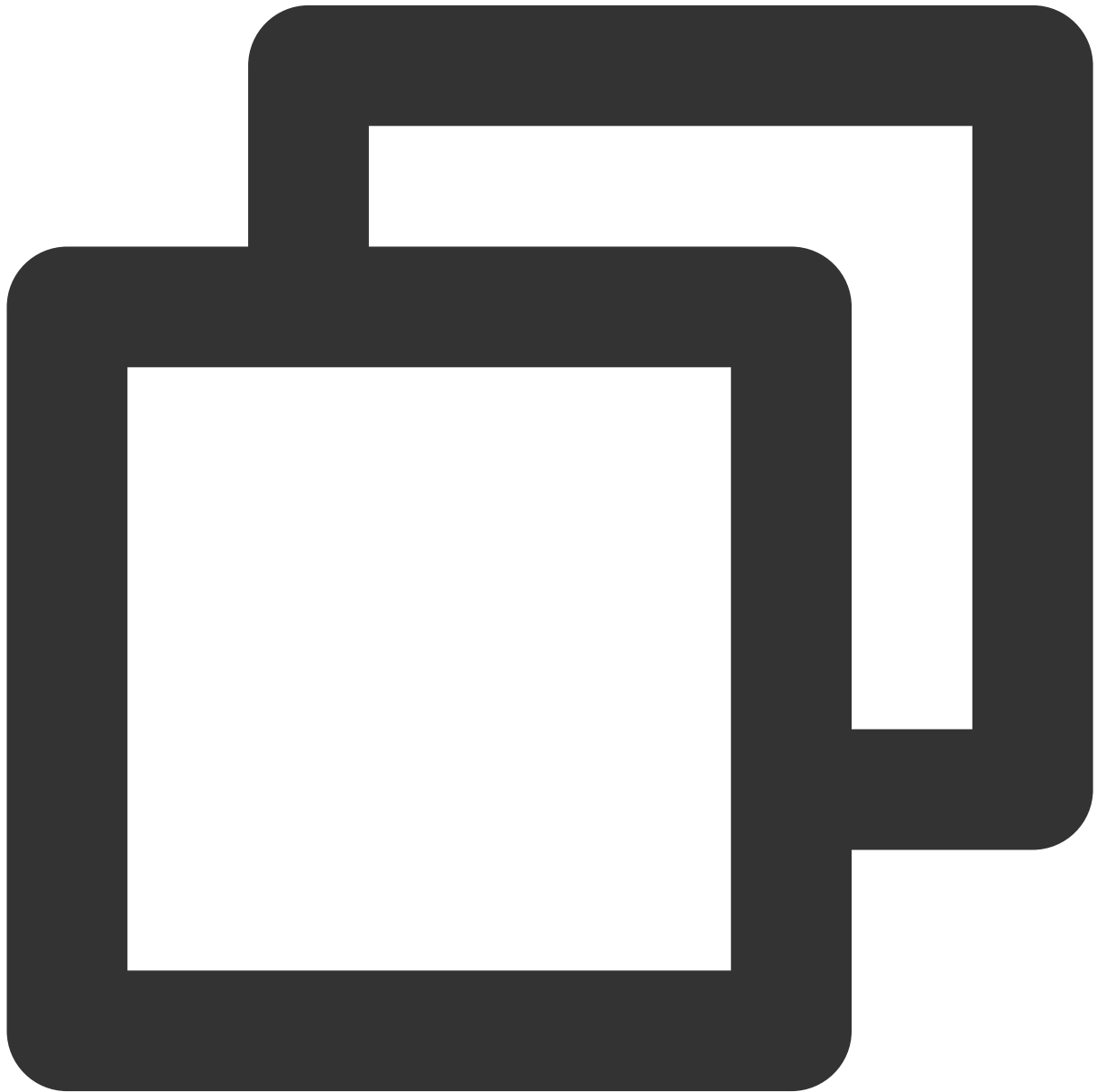
The parameters are described below:

Parameter	Type	Description
callback	ActionCallback	Callback for room termination. The code is 0 if the operation succeeds.

## enterRoom

This API is used to enter a room (called by listener).





```
public abstract void enterRoom(int roomId, TRTCVoiceRoomCallback.ActionCallback cal
```

The parameters are described below:

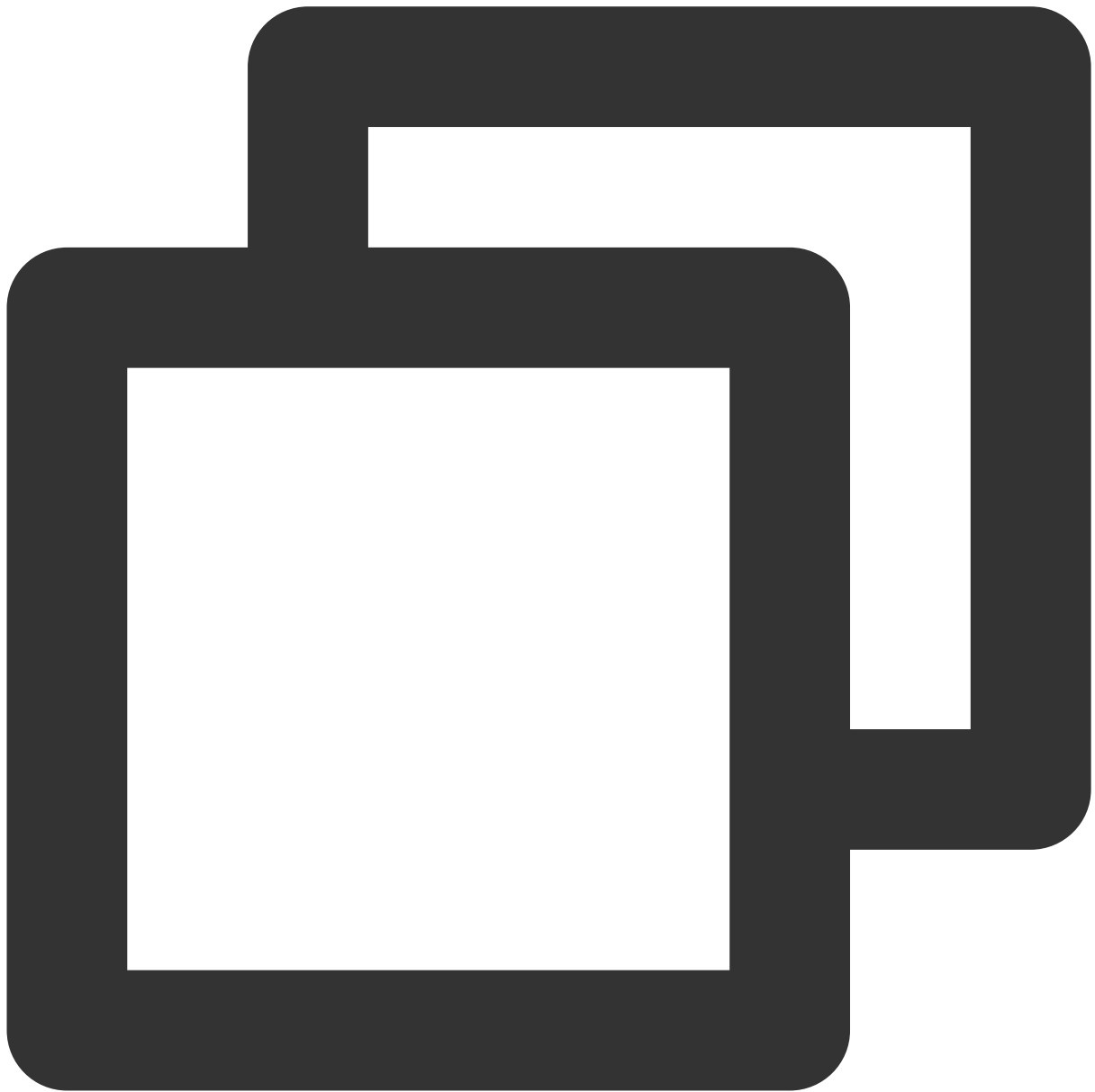
Parameter	Type	Description
roomId	int	The room ID.
callback	ActionCallback	Callback for room entry. The code is 0 if the operation succeeds.

The process of entering a room as a listener is as follows:

1. A user gets the latest audio chat room list from your server. The list may contain the `roomId` and room information of multiple audio chat rooms.
2. The user selects a room, and calls `enterRoom` with the room ID passed in to enter the room.
3. After entering the room, the user receives an `onRoomInfoChange` notification about room attribute change from the component. The attributes can be recorded, and corresponding changes can be made to the UI, including room name, whether room owner's consent is required for listeners to speak, etc.
4. The user will receive an `onSeatListChange` notification about the change of the seat list and can update the change to the UI.
5. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker.

## **exitRoom**

Leave room



```
public abstract void exitRoom(TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are described below:

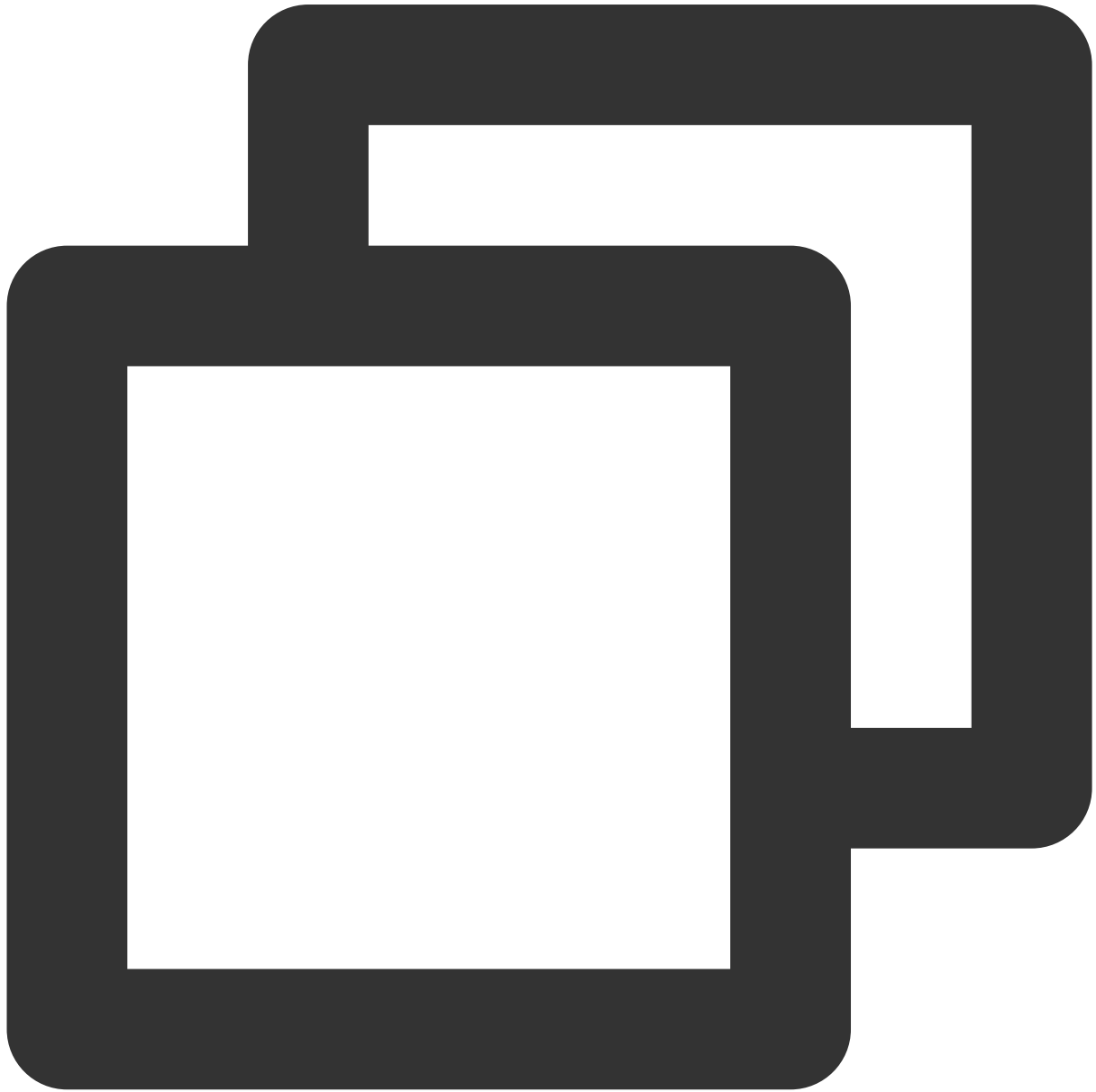
Parameter	Type	Description
callback	ActionCallback	Callback for room exit. The code is 0 if the operation succeeds.

## getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()`.

**Note**

You don't need this API if both the room list and room information are managed on your server.



```
public abstract void getRoomInfoList(List<Integer> roomIdList, TRTCVoiceRoomCallbac
```

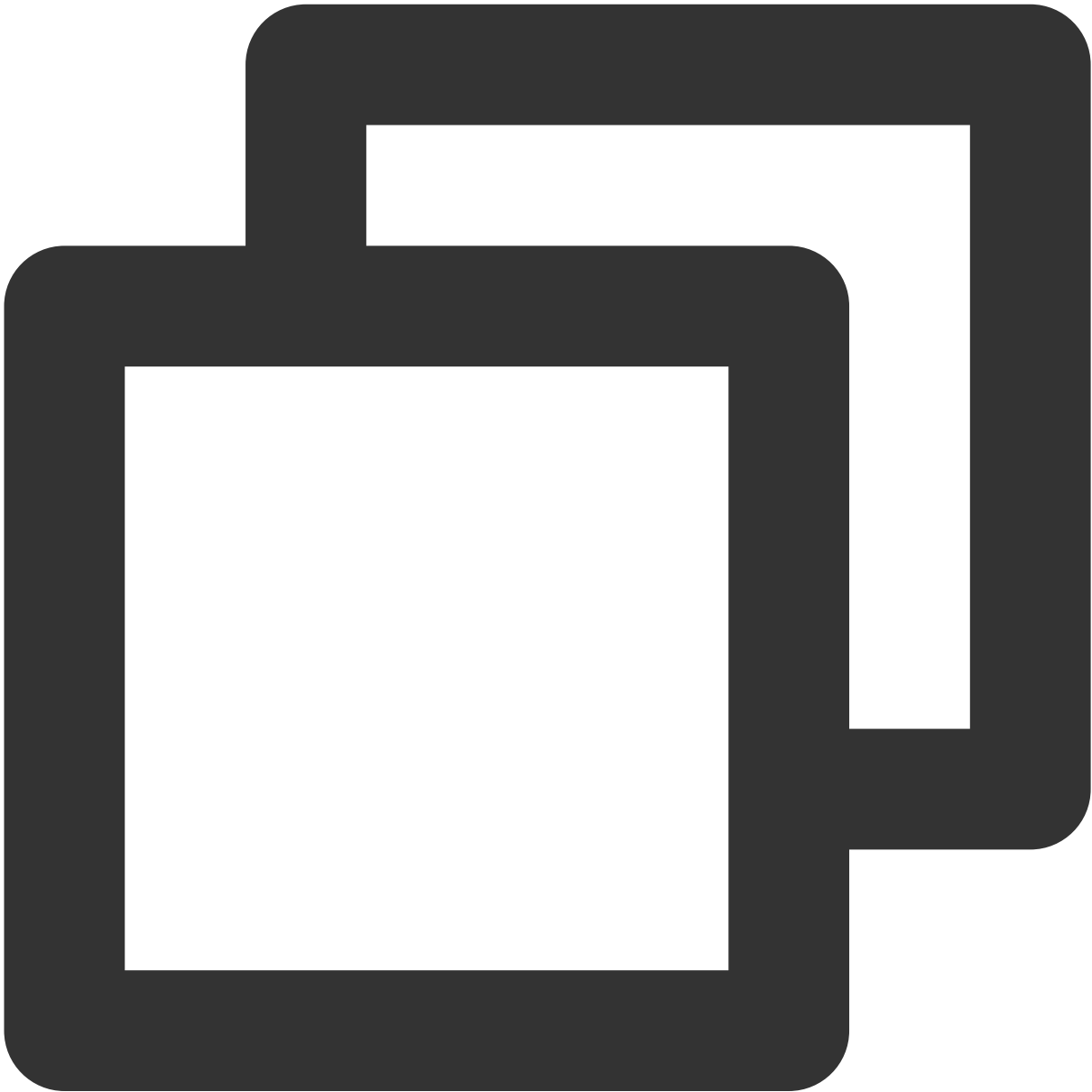
The parameters are described below:

Parameter	Type	Description

roomIdList	List<Integer>	Room ID list
callback	RoomInfoCallback	Callback of room details

getUserInfoList

This API is used to get the user information of a specified `userId` .



```
public abstract void getUserInfoList(List<String> userIdList, TRTCVoiceRoomCallback
```

The parameters are described below:

--	--	--

Parameter	Type	Description
userIdList	List<String>	IDs of the users to query. If this parameter is <code>null</code> , the information of all users in the room is queried.
userlistcallback	UserListCallback	Callback of user details

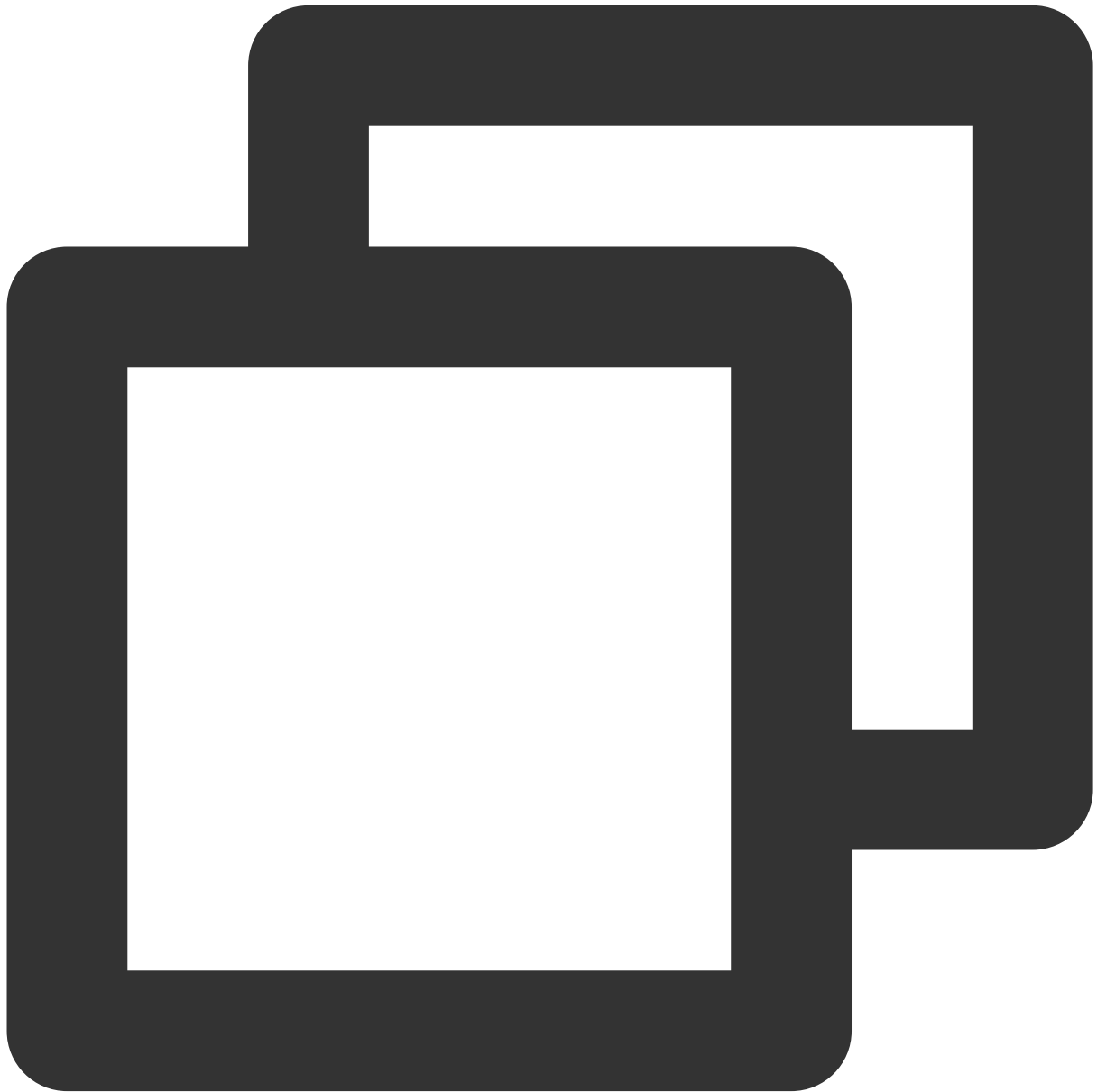
## Seat Management APIs

### enterSeat

This API is used to become a speaker (called by room owner or listener).

#### Note

After a user becomes a speaker, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.



```
public abstract void enterSeat(int seatIndex, TRTCVoiceRoomCallback.ActionCallback
```

The parameters are described below:

Parameter	Type	Description
seatIndex	int	The number of the seat to take
callback	ActionCallback	Callback for the operation

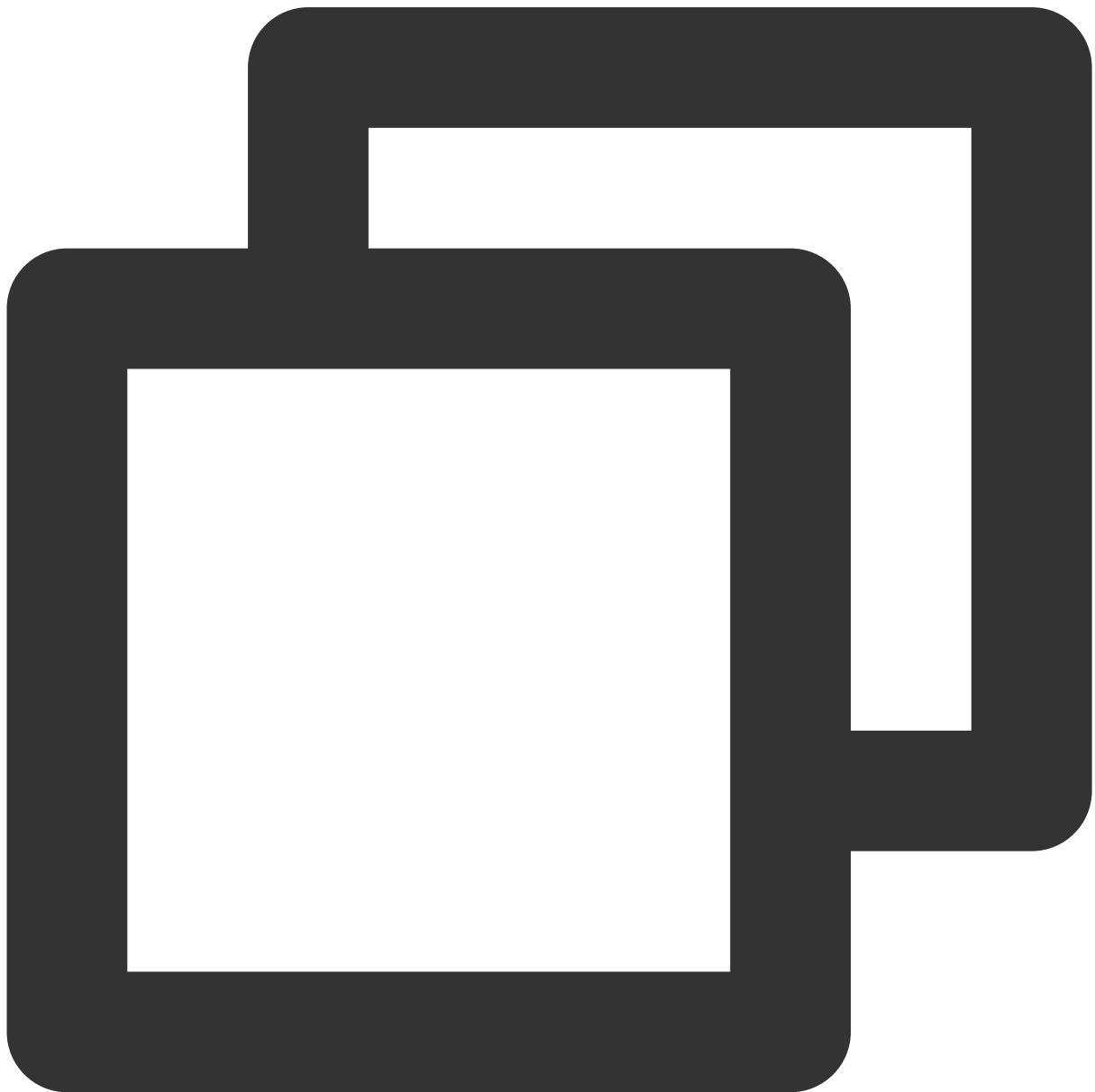
Calling this API will immediately modify the seat list. In cases where listeners need the room owner's consent to take a seat, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call this API.

## moveSeat

This API is used to change one's seat (called by speaker).

### Note

After the seat change, all users in the room will receive the `onSeatListChange`, `onAnchorLeaveSeat`, and `onAnchorEnterSeat` notifications. This API will only change the user's seat number, not the user role.





```
public abstract int moveSeat(int seatIndex, TRTCVoiceRoomCallback.ActionCallback ca
```

The parameters are described below:

Parameter	Type	Description
seatIndex	int	The number of the seat to change to
callback	ActionCallback	Callback for the operation

Response parameters:

Parameter	Type	Description
code	int	Result of seat change. 0 : operation successful; 10001 : API rate limit exceeded; other values: operation failed

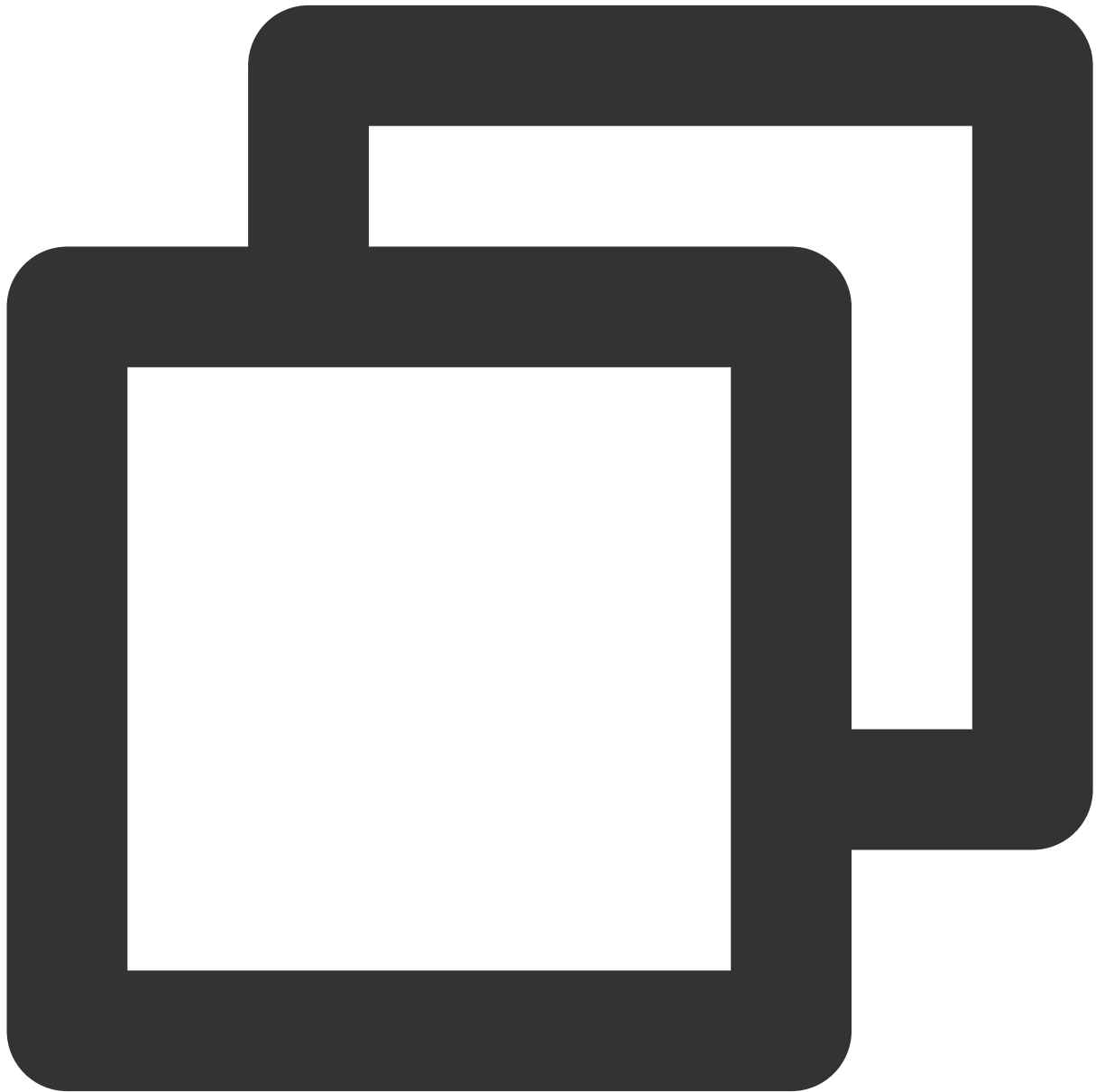
Calling this API will immediately modify the seat list. In cases where listeners need the room owner's consent to take a seat, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call this API.

## leaveSeat

This API is used to become a listener (called by speaker).

### Note

After a speaker becomes a listener, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.



```
public abstract void leaveSeat (TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are described below:

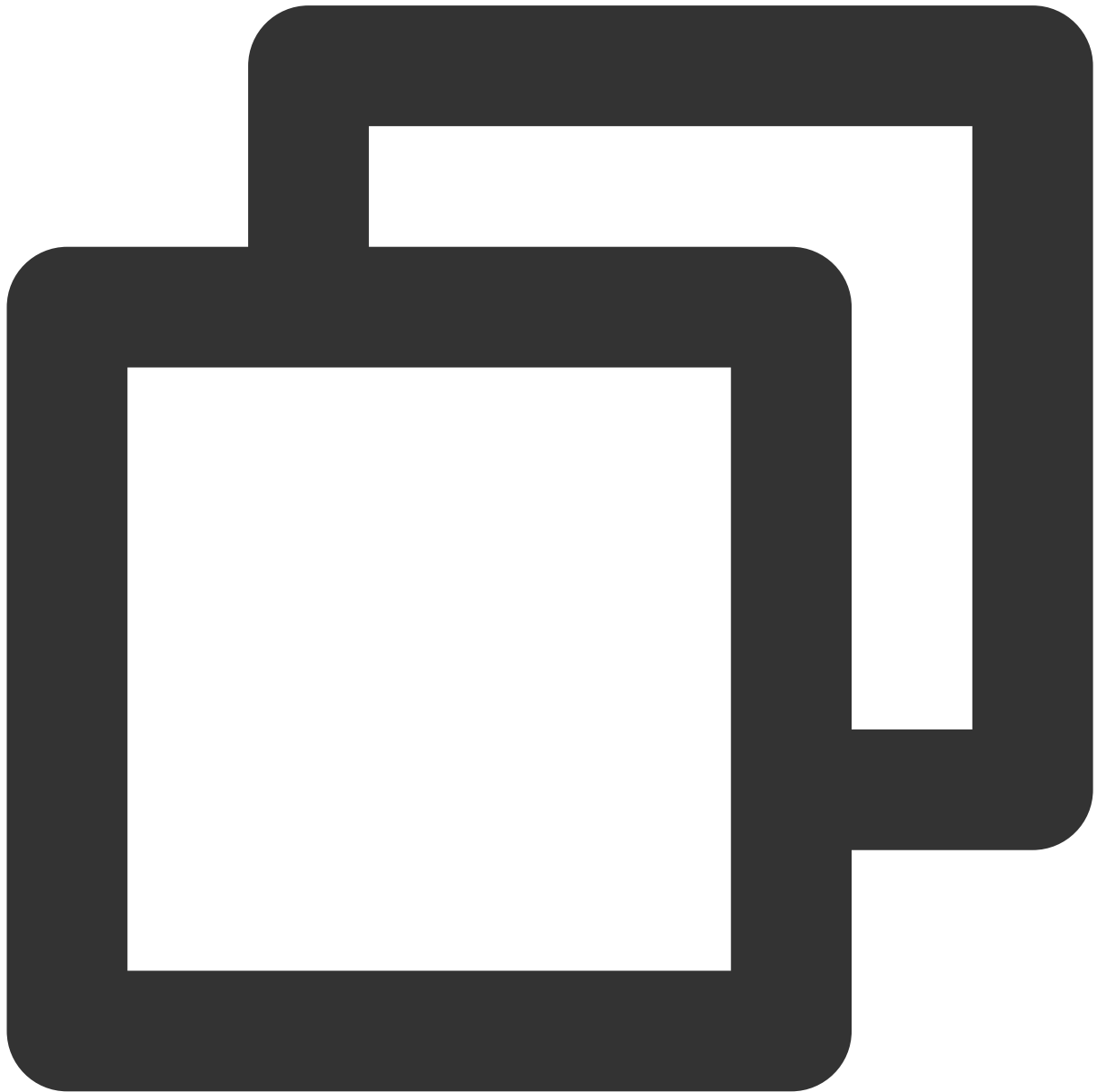
Parameter	Type	Description
callback	ActionCallback	Callback for the operation

## **pickSeat**

This API is used to place a user in a seat (called by room owner).

**Note**

After the room owner places a user in a seat, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.



```
public abstract void pickSeat(int seatIndex, String userId, TRTCVoiceRoomCallback.A
```

The parameters are described below:

Parameter	Type	Description
seatIndex	int	The number of the seat to place the listener in

userId	String	User ID
callback	ActionCallback	Callback for the operation

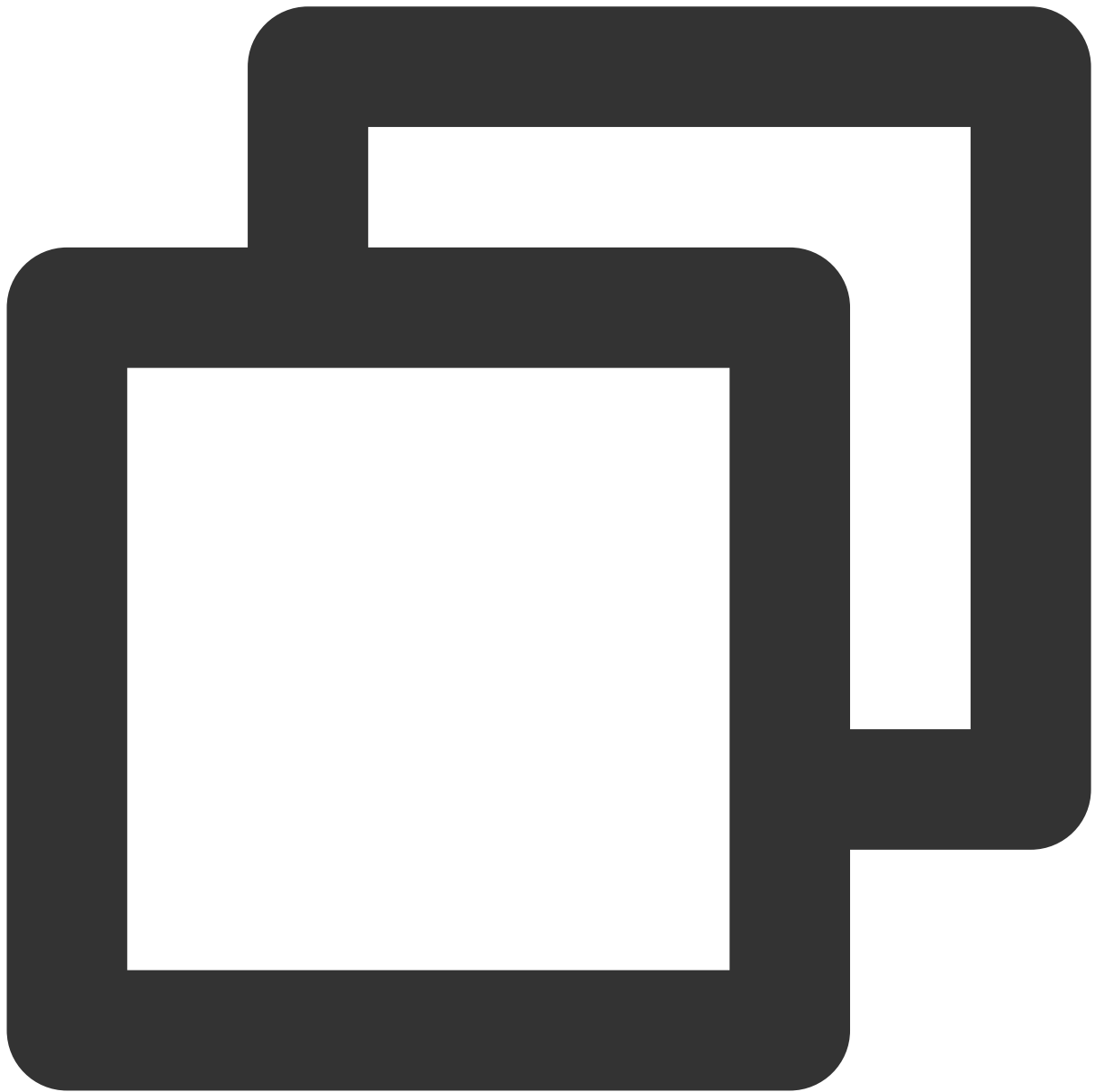
Calling this API will immediately modify the seat list. In cases where the room owner needs listeners' consent to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call `pickSeat`.

## kickSeat

This API is used to remove a speaker (called by room owner).

### Note

After a speaker is removed, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.



```
public abstract void kickSeat(int seatIndex, TRTCVoiceRoomCallback.ActionCallback c
```

The parameters are described below:

Parameter	Type	Description
seatIndex	int	The number of the seat to remove the speaker from
callback	ActionCallback	Callback for the operation

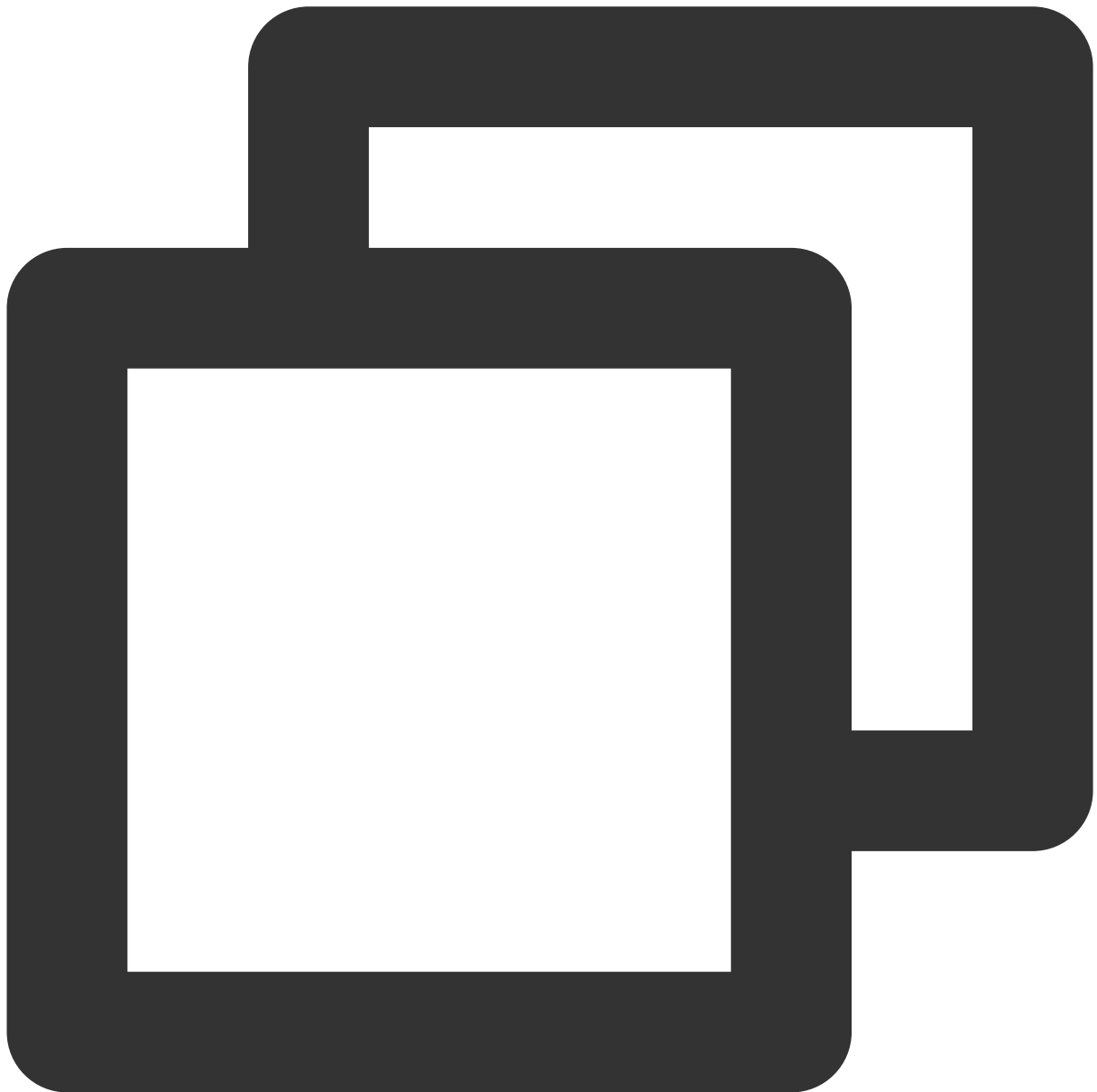
Calling this API will immediately modify the seat list.

## **muteSeat**

This API is used to mute/unmute a seat (called by room owner).

### **Note**

After a seat is muted/unmuted, all members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.



```
public abstract void muteSeat(int seatIndex, boolean isMute, TRTCVoiceRoomCallback.
```

The parameters are described below:

Parameter	Type	Description
seatIndex	int	The number of the seat to block/unblock
isMute	boolean	<code>true</code> : mute; <code>false</code> : unmute
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will call `muteAudio` to mute/unmute his or her audio.

## closeSeat

This API is used to block/unblock a seat (called by room owner).

### Note

After a seat is blocked/unblocked, all members in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.



```
public abstract void closeSeat(int seatIndex, boolean isClose, TRTCVoiceRoomCallbac
```

The parameters are described below:

Parameter	Type	Description
seatIndex	int	The number of the seat to block/unblock
isClose	boolean	<code>true</code> : block; <code>false</code> : unblock
callback	ActionCallback	Callback for the operation

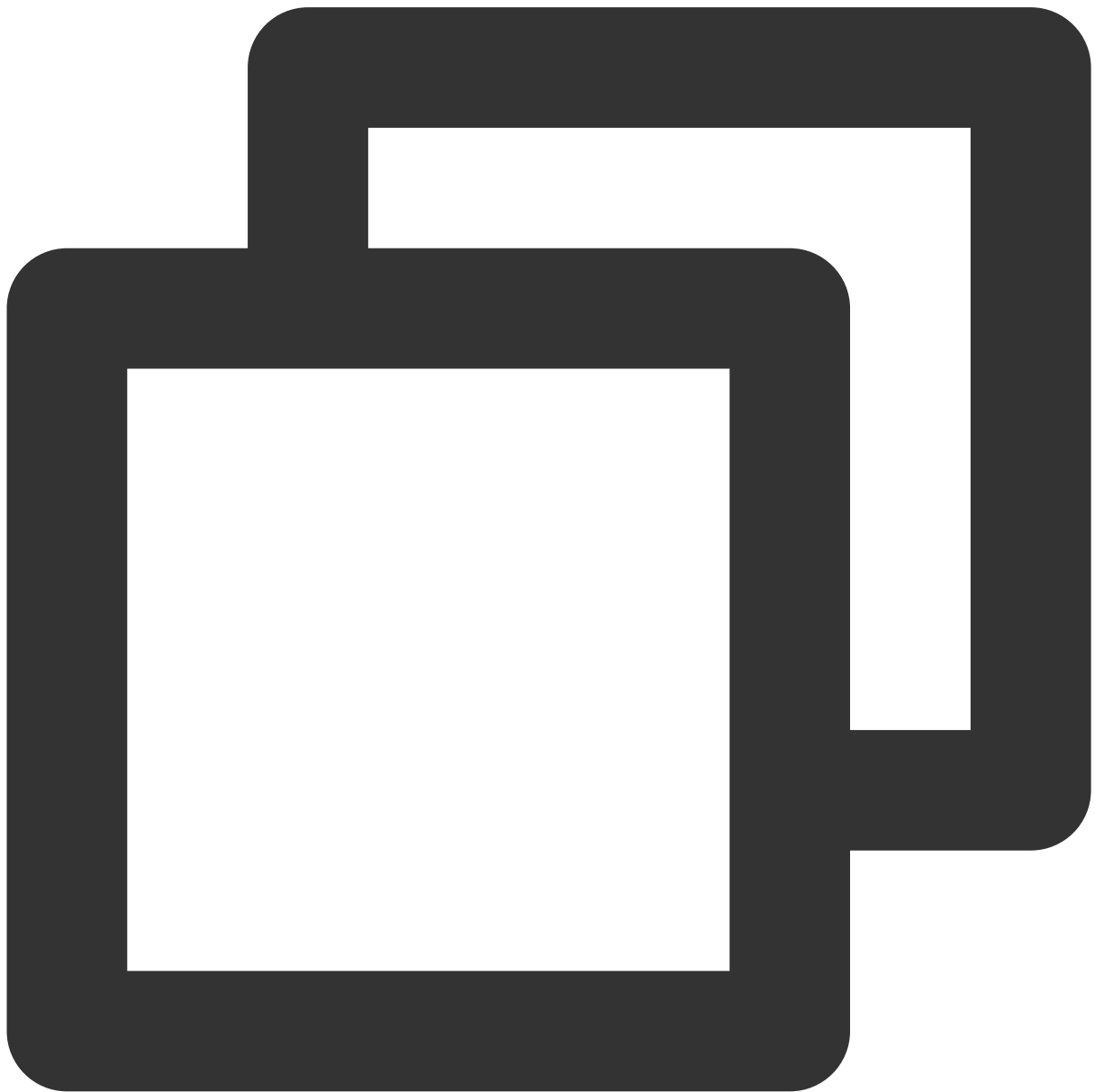


Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will leave the seat.

## Local Audio APIs

### **startMicrophone**

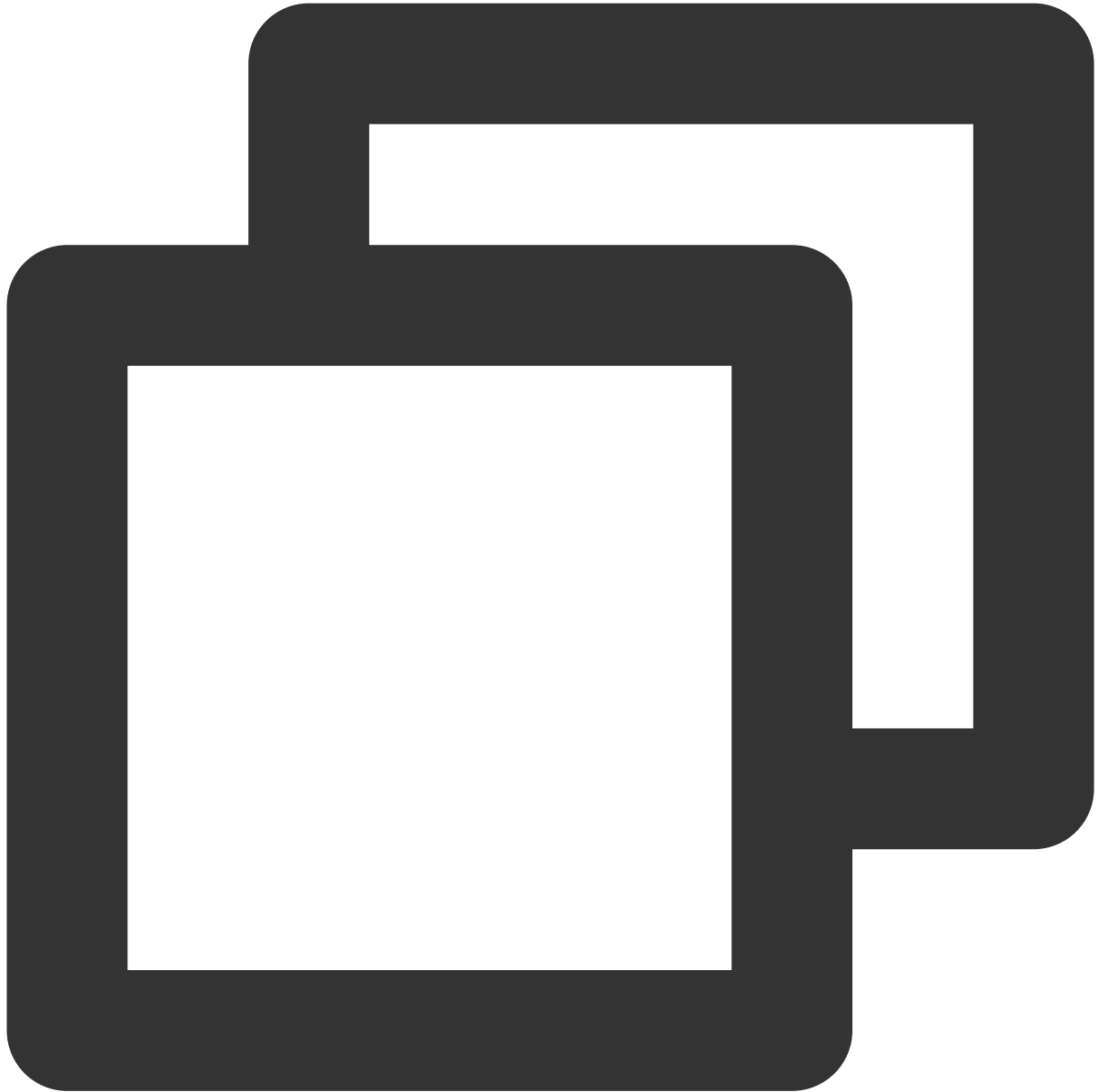
This API is used to start mic capturing.



```
public abstract void startMicrophone();
```

## stopMicrophone

This API is used to stop mic capturing.



```
public abstract void stopMicrophone();
```

## setAudioQuality

This API is used to set audio quality.



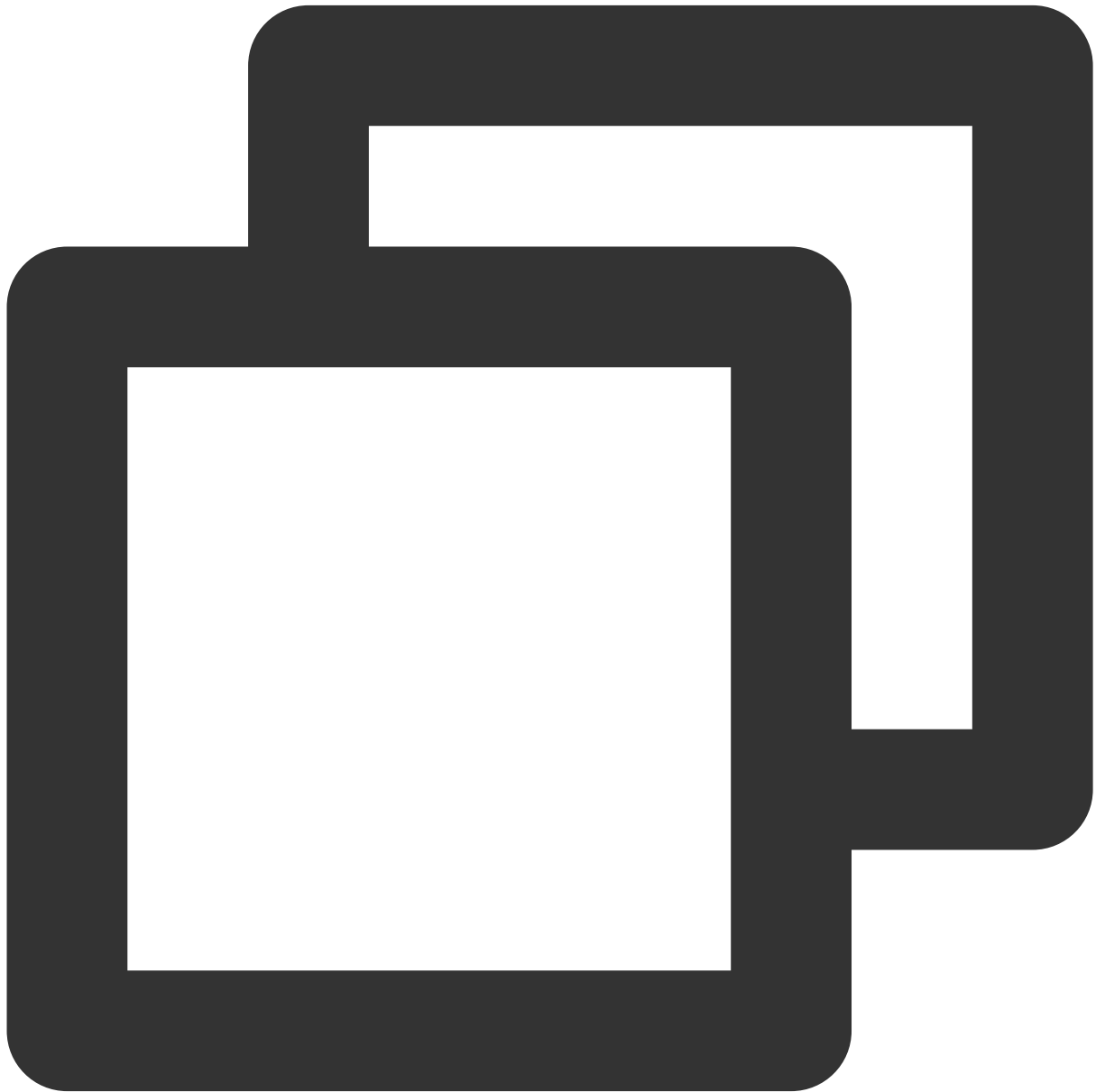
```
public abstract void setAudioQuality(int quality);
```

The parameters are described below:

Parameter	Type	Description
quality	int	The audio quality. For more information, see <a href="#">setAudioQuality()</a> .

## **muteLocalAudio**

This API is used to mute/unmute local audio.



```
public abstract void muteLocalAudio(boolean mute);
```

The parameters are described below:

Parameter	Type	Description
mute	boolean	Whether to mute or unmute audio. For more information, see <a href="#">muteLocalAudio()</a> .

## setSpeaker

This API is used to set whether to play sound from the device's speaker or receiver.



```
public abstract void setSpeaker(boolean useSpeaker);
```

The parameters are described below:

Parameter	Type	Description
useSpeaker	boolean	true : Speaker; false : Receiver

### setAudioCaptureVolume

This API is used to set the mic capturing volume.



```
public abstract void setAudioCaptureVolume(int volume);
```

The parameters are described below:

Parameter	Type	Description
volume	int	Capturing volume. Value range: 0-100 (default: 100)

### **setAudioPlayoutVolume**

This API is used to set the playback volume.



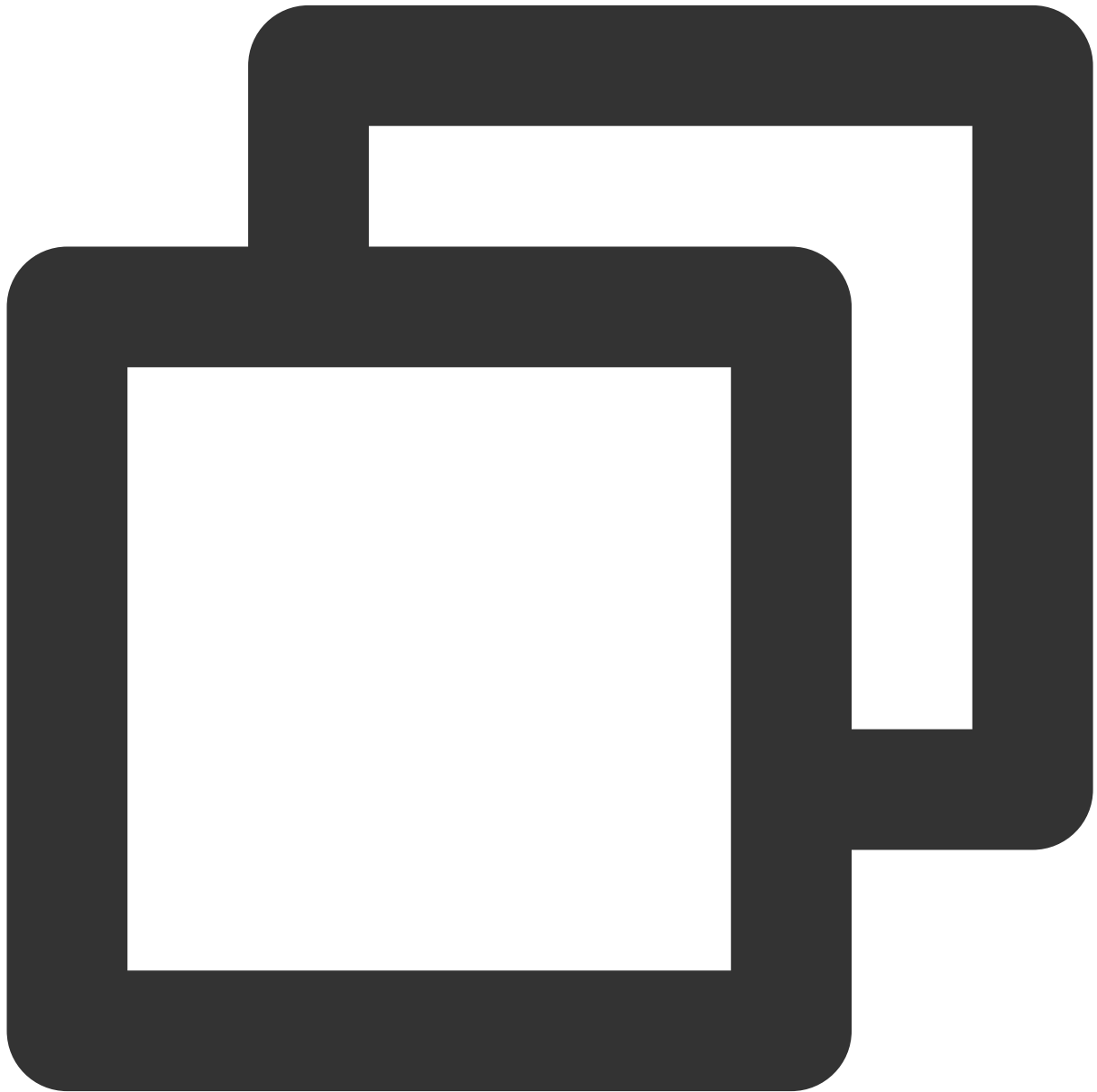
```
public abstract void setAudioPlayoutVolume(int volume);
```

The parameters are described below:

Parameter	Type	Description
volume	int	Playback volume. Value range: 0-100 (default: 100)

## **muteRemoteAudio**

This API is used to mute/unmute a specified user.



```
public abstract void muteRemoteAudio(String userId, boolean mute);
```

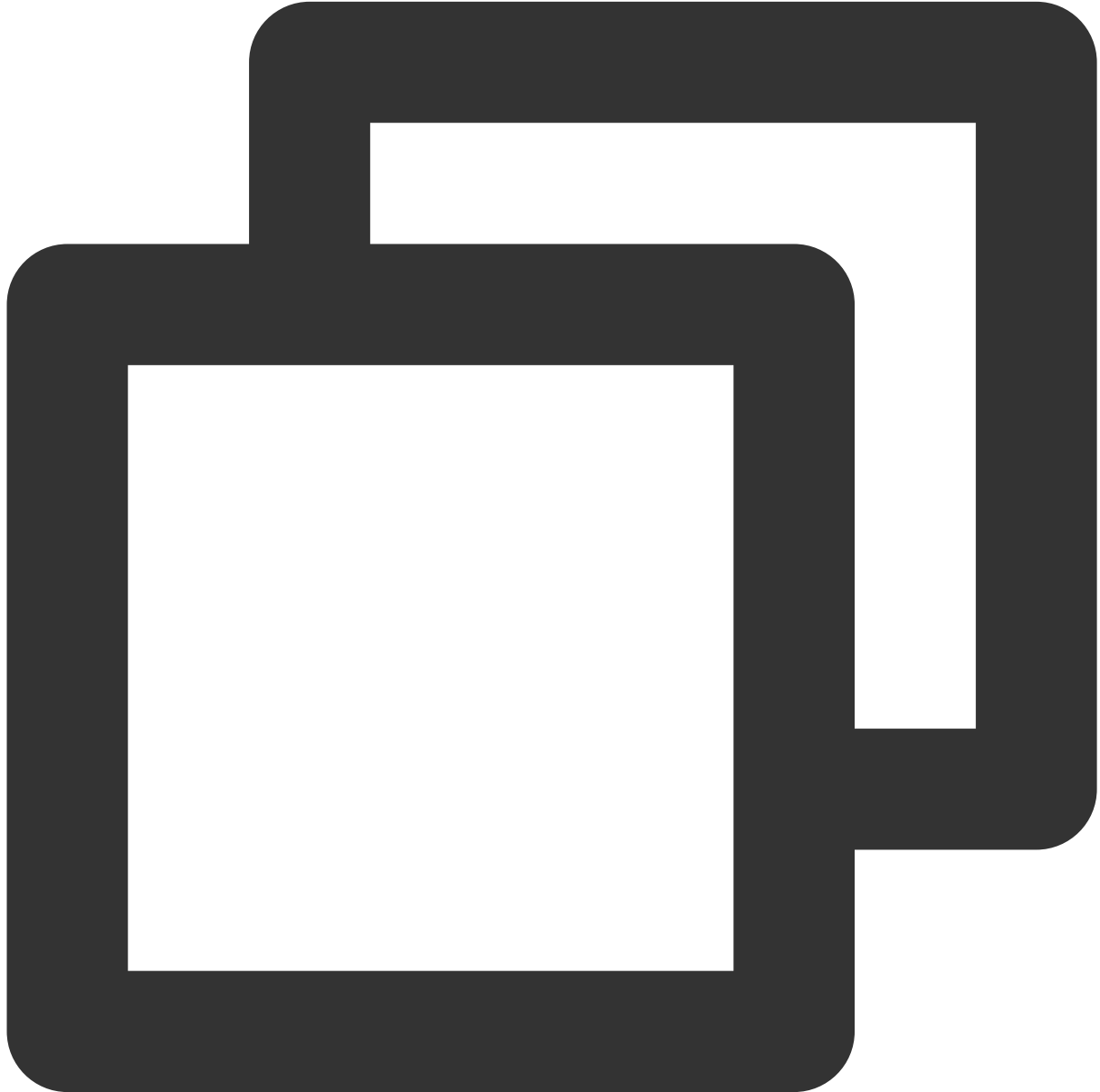
The parameters are described below:

Parameter	Type	Description
userId	String	User ID
mute	boolean	true : Mute; false : Unmute



## **muteAllRemoteAudio**

This API is used to mute/unmute all users.



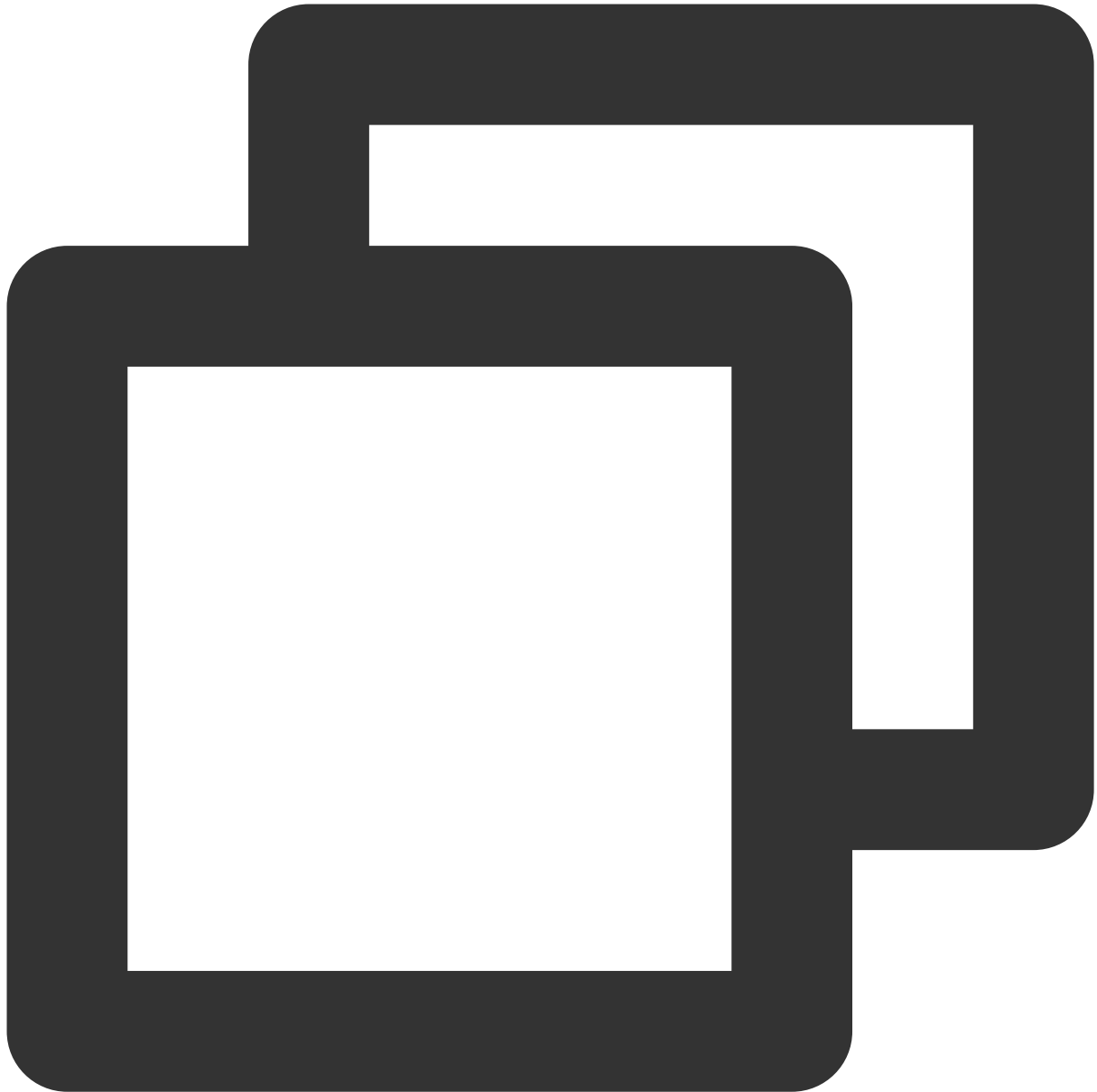
```
public abstract void muteAllRemoteAudio(boolean mute);
```

The parameters are described below:

Parameter	Type	Description
mute	boolean	true : Mute; false : Unmute

## setVoiceEarMonitorEnable

This API is used to enable/disable in-ear monitoring.



```
public abstract void setVoiceEarMonitorEnable(boolean enable);
```

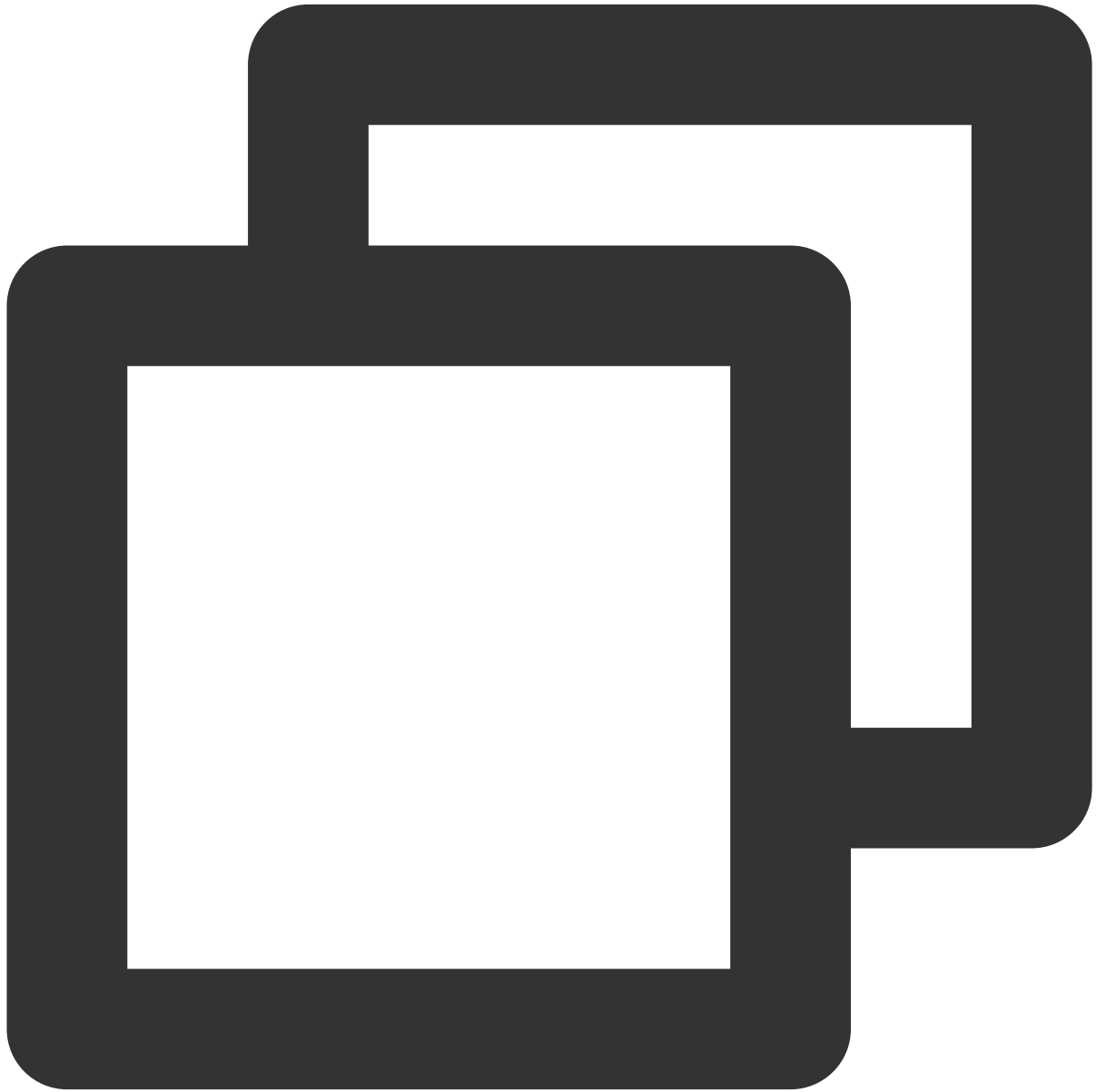
The parameters are described below:

Parameter	Type	Description
enable	boolean	<code>true</code> : Enable; <code>false</code> : Disable

## Background Music and Audio Effect APIs

### **getAudioEffectManager**

This API is used to get the background music and audio effect management object [TXAudioEffectManager](#).

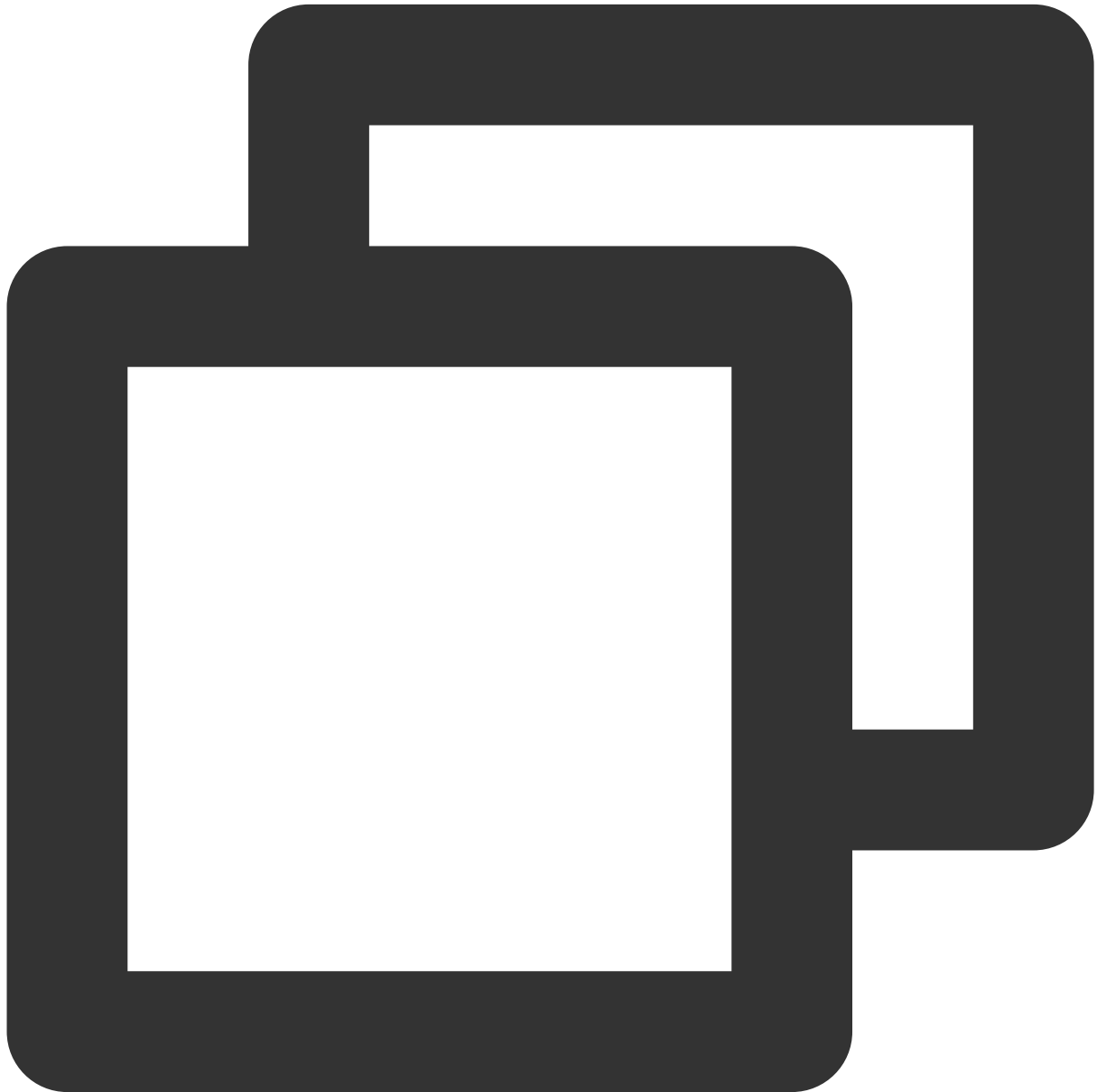


```
public abstract TXAudioEffectManager getAudioEffectManager();
```

## Message Sending APIs

## sendRoomTextMsg

This API is used to broadcast a text chat message in a room, which is generally used for on-screen comments.



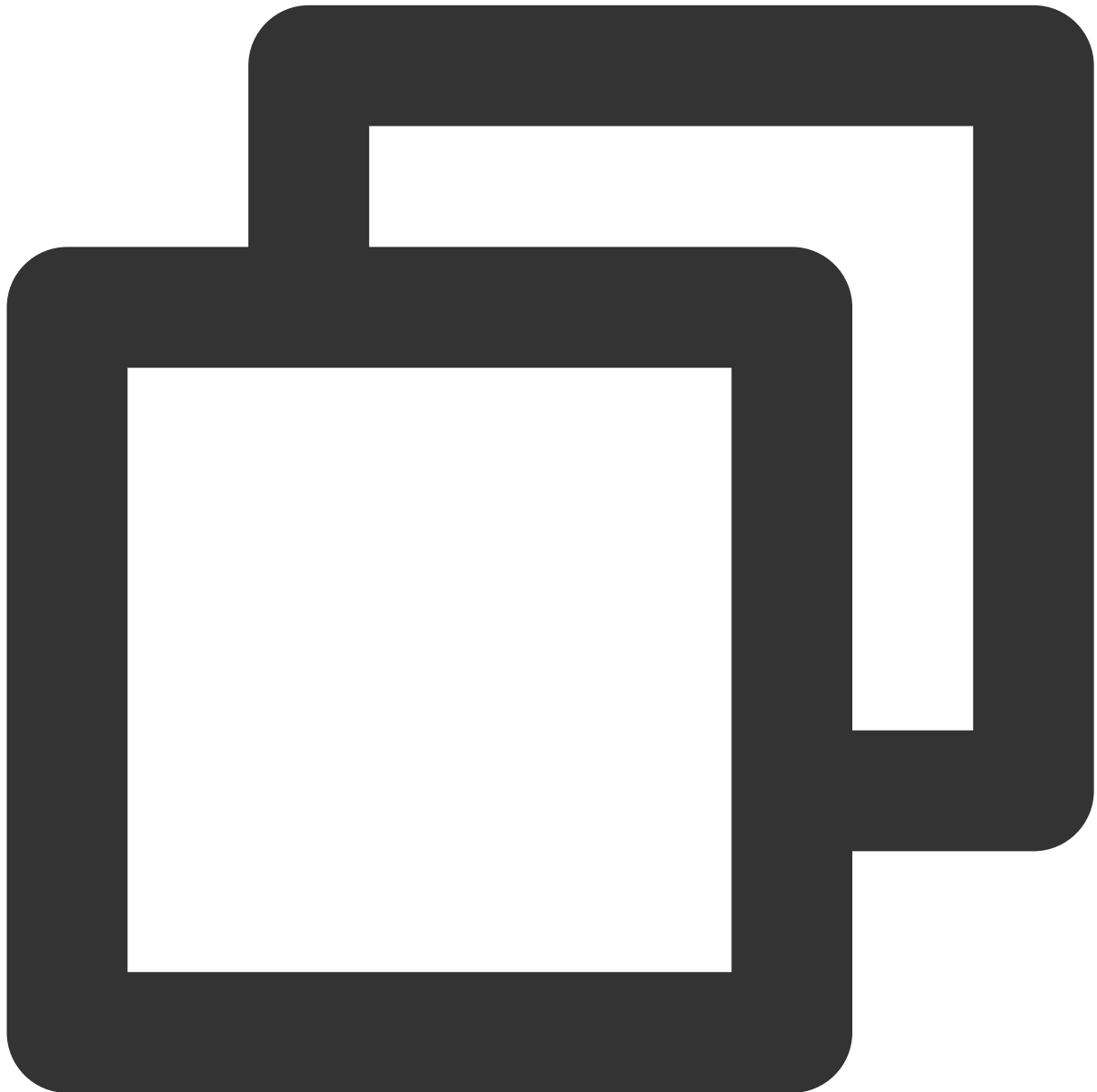
```
public abstract void sendRoomTextMsg(String message, TRTCVoiceRoomCallback.ActionCa
```

The parameters are described below:

Parameter	Type	Description
message	String	Text message
callback	ActionCallback	Callback for the operation

## sendRoomCustomMsg

This API is used to send a custom text message.



```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCVoiceRoomCal
```

The parameters are described below:

Parameter	Type	Description
cmd	String	A custom command word used to distinguish between different message

		types.
message	String	Text message
callback	ActionCallback	Callback for the operation

## Invitation Signaling APIs

### **sendInvitation**

This API is used to send an invitation.



```
public abstract String sendInvitation(String cmd, String userId, String content, TR
```

The parameters are described below:

Parameter	Type	Description
cmd	String	Custom command of business
userId	String	Invitee's user ID
content	String	Invitation content

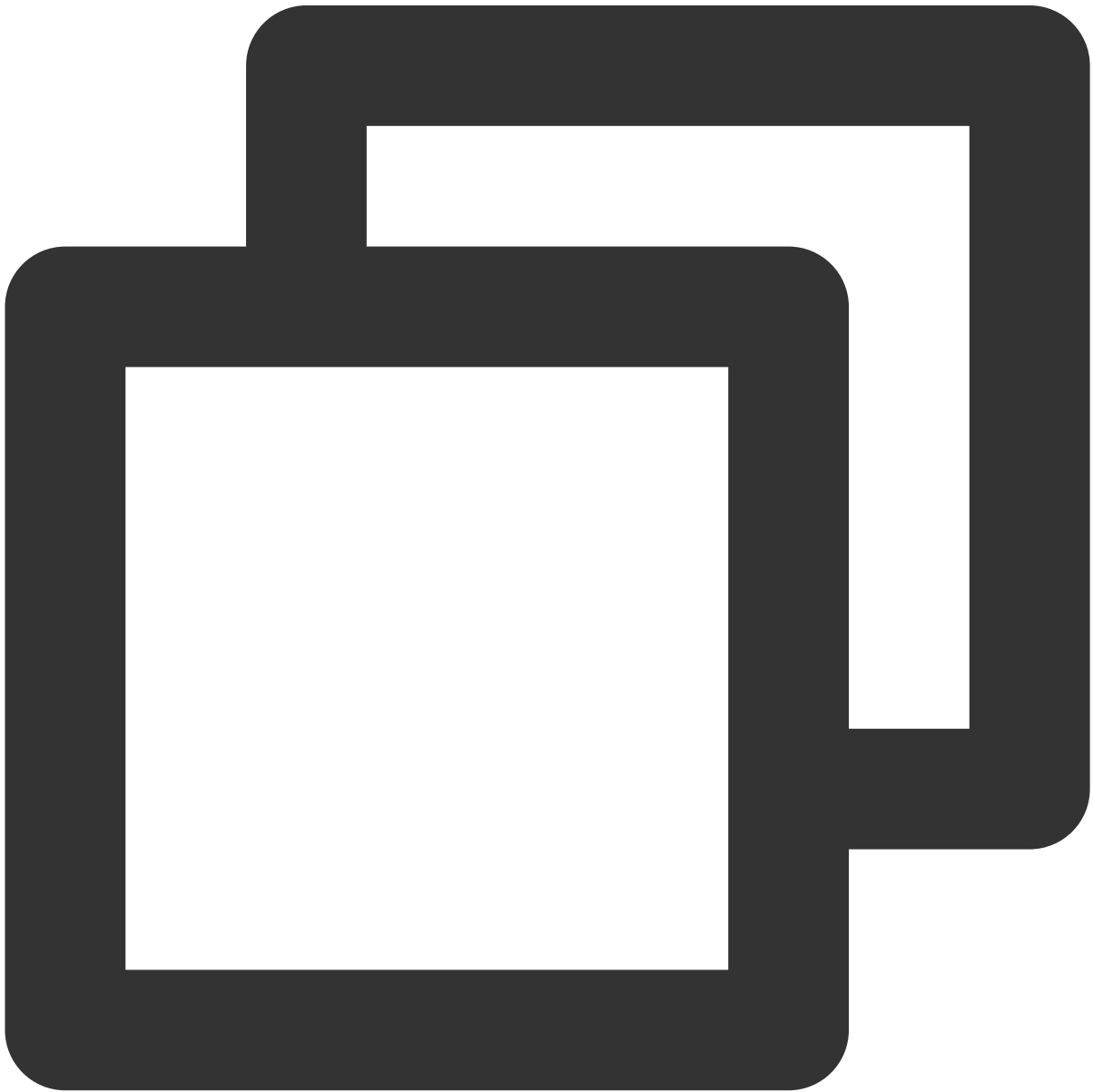
callback	ActionCallback	Callback for the operation
----------	----------------	----------------------------

Response parameters:

Parameter	Type	Description
inviteId	String	Invitation ID

**acceptInvitation**

This API is used to accept an invitation.





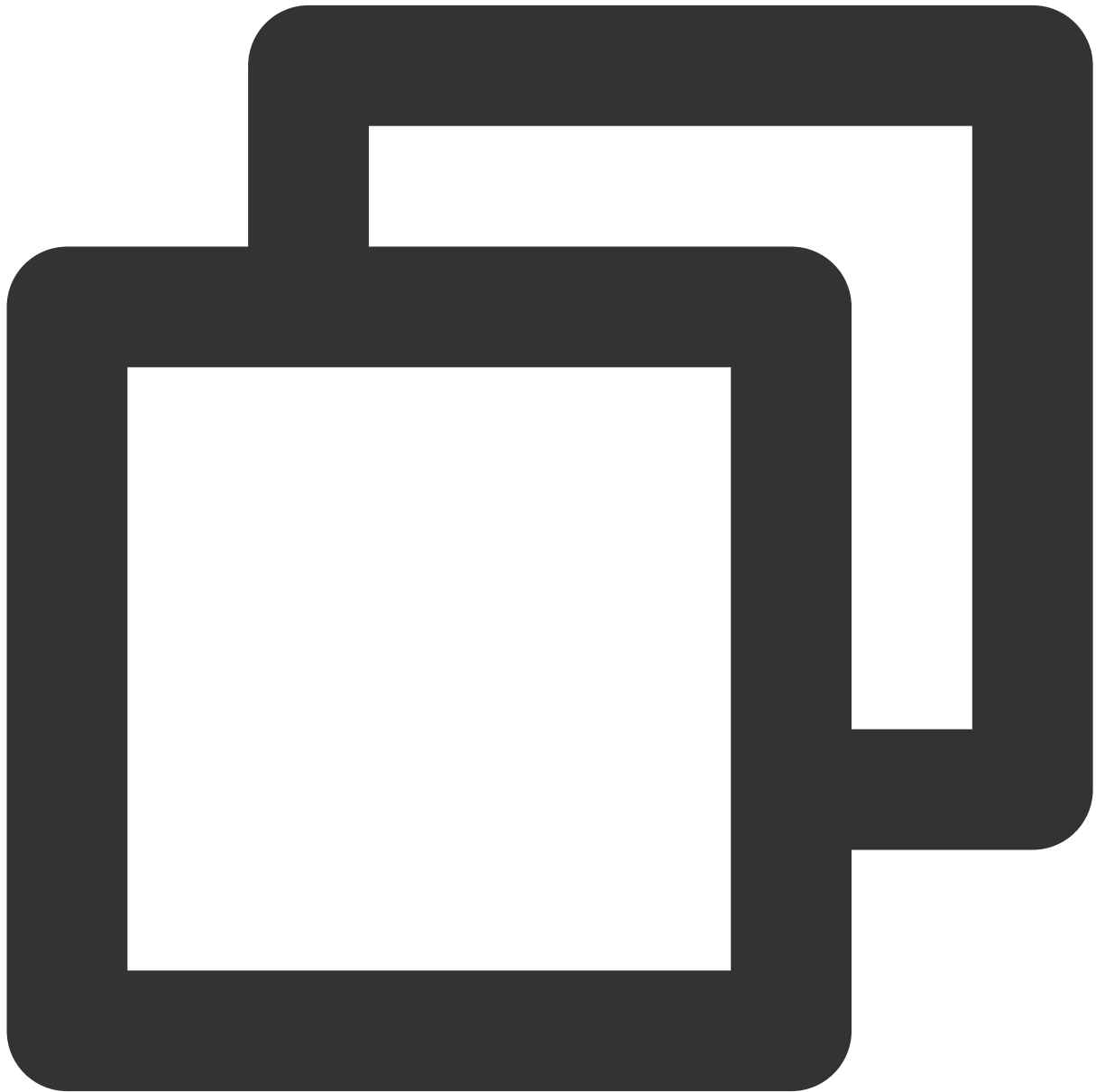
```
public abstract void acceptInvitation(String id, TRTCVoiceRoomCallback.ActionCallba
```

The parameters are described below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

## rejectInvitation

This API is used to decline an invitation.



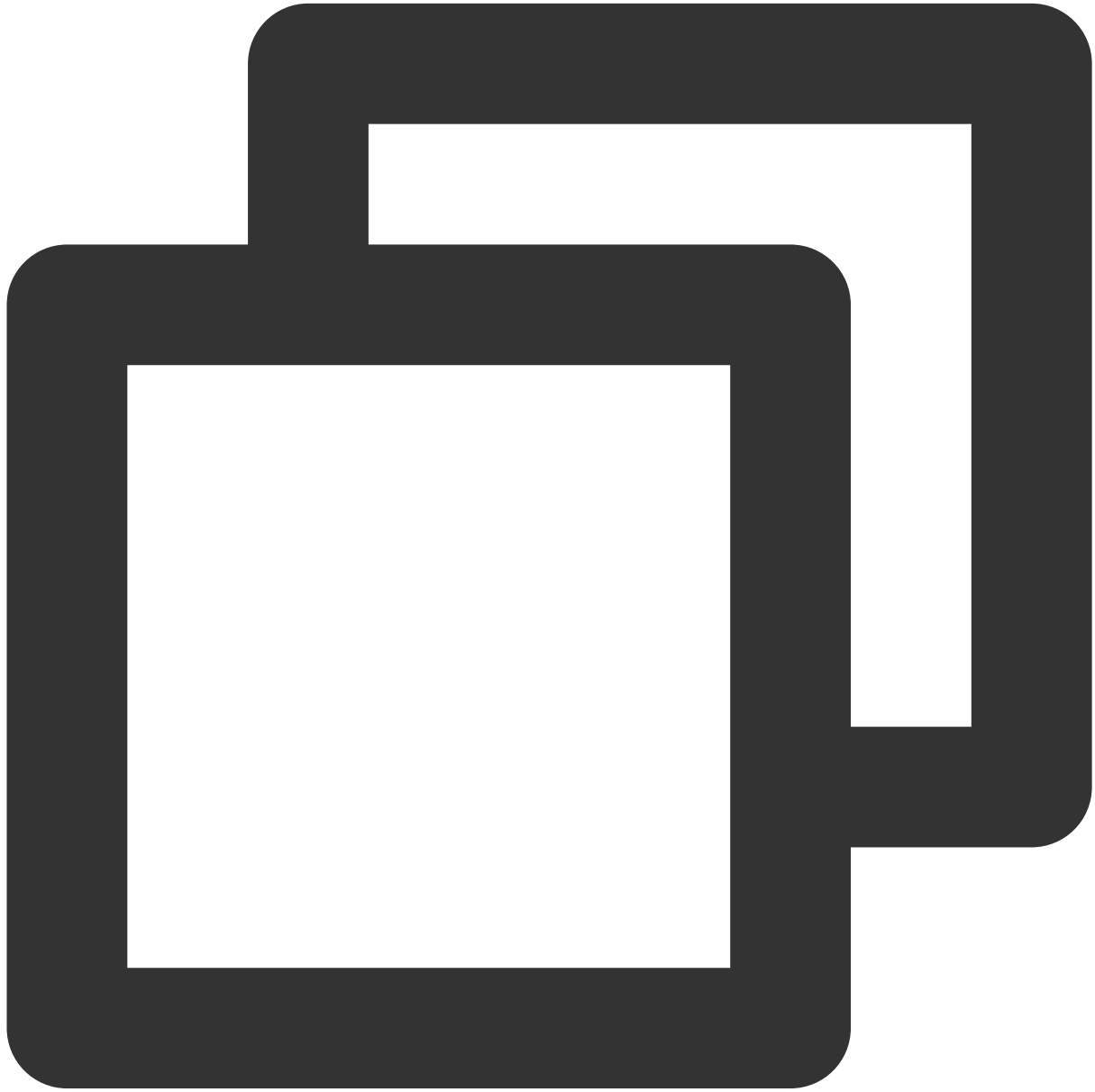
```
public abstract void rejectInvitation(String id, TRTCVoiceRoomCallback.ActionCallba
```

The parameters are described below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

## cancelInvitation

This API is used to cancel an invitation.



```
public abstract void cancelInvitation(String id, TRTCVoiceRoomCallback.ActionCallba
```

The parameters are described below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

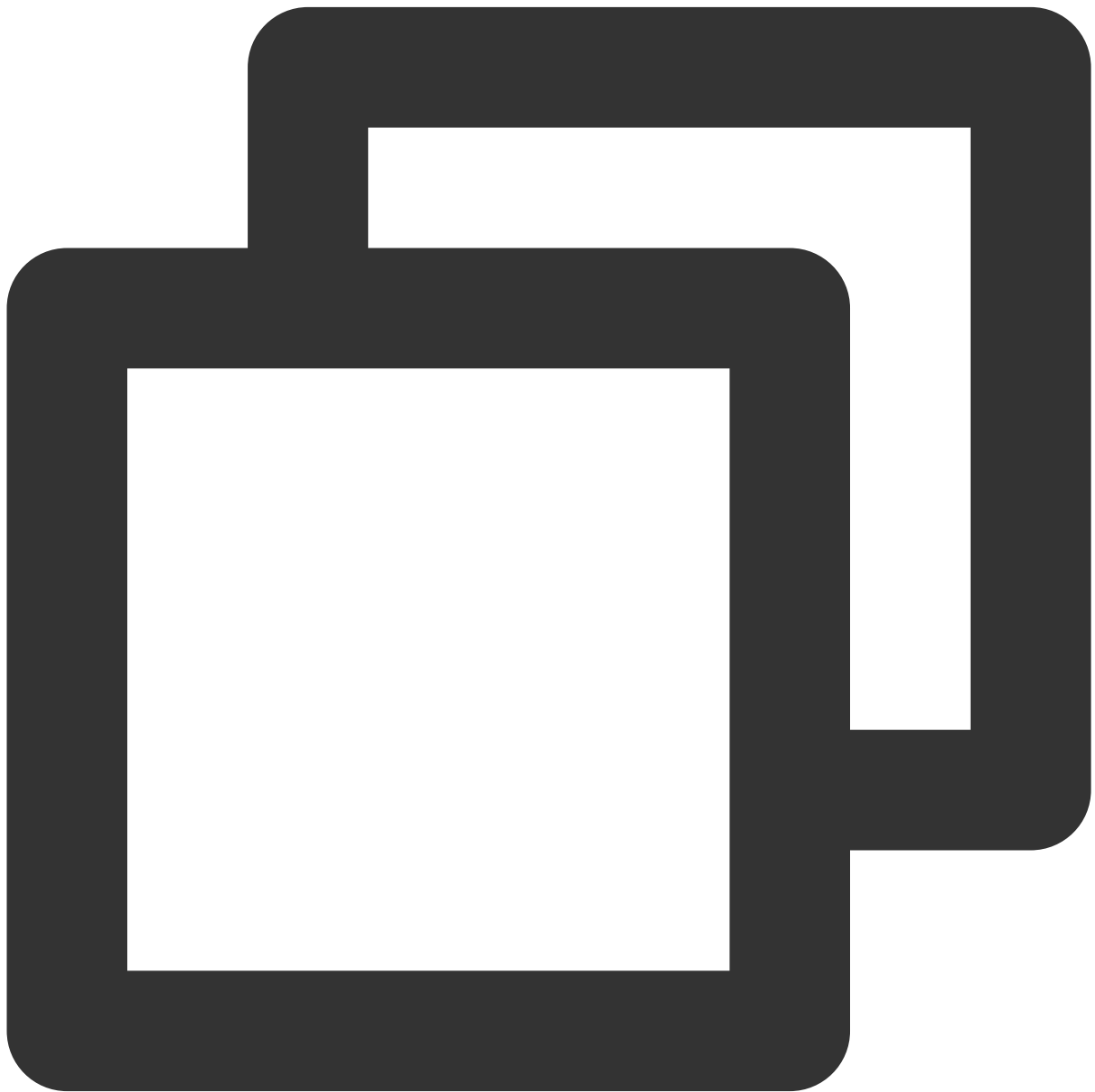
## TRTCVoiceRoomDelegate Event Callback APIs

### Common Event Callback APIs

#### **onError**

Callback for error.

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users depending if necessary.



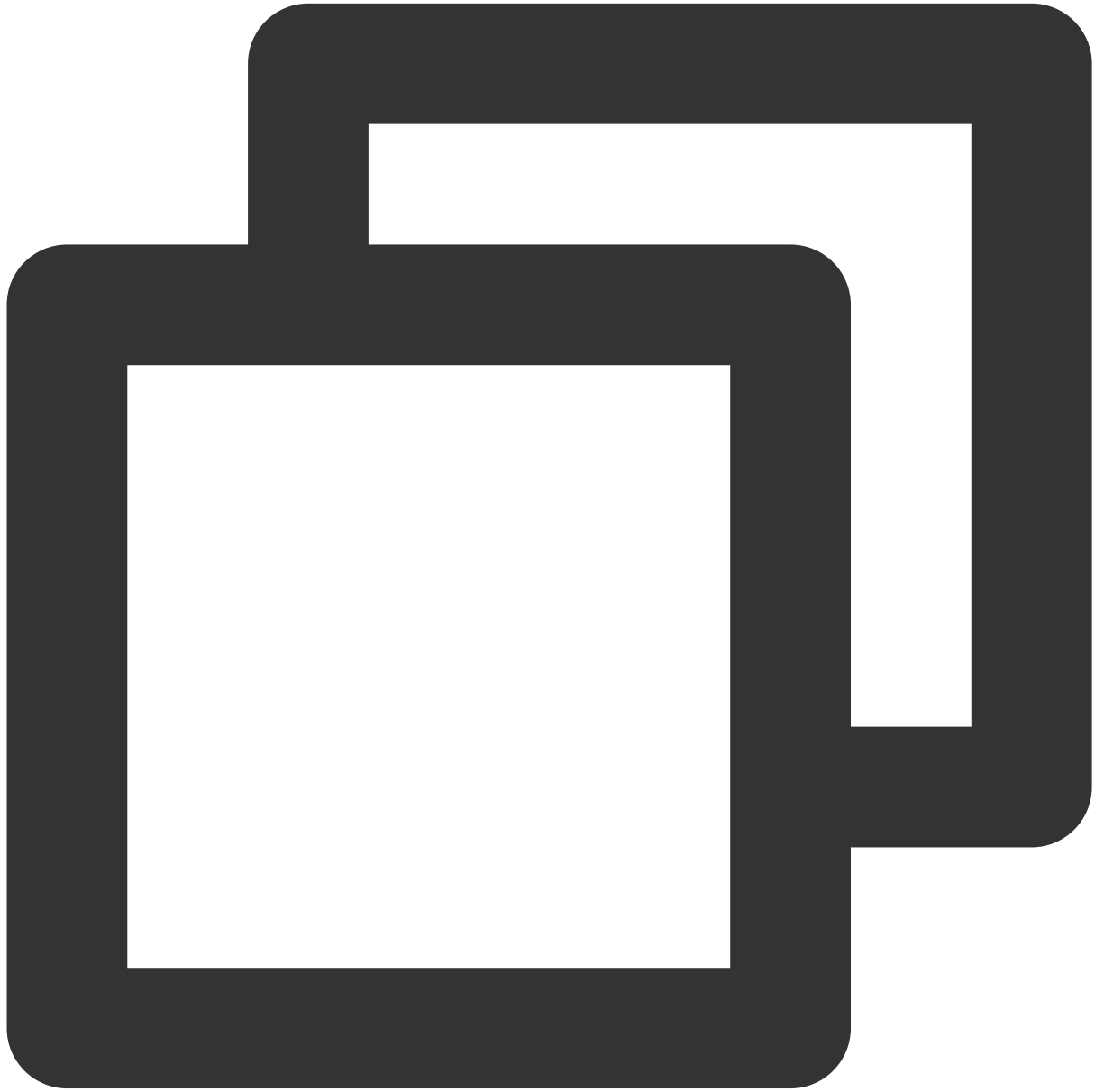
```
void onError(int code, String message);
```

The parameters are described below:

Parameter	Type	Description
code	int	Error code
message	String	Error message

## onWarning

Callback for warning.



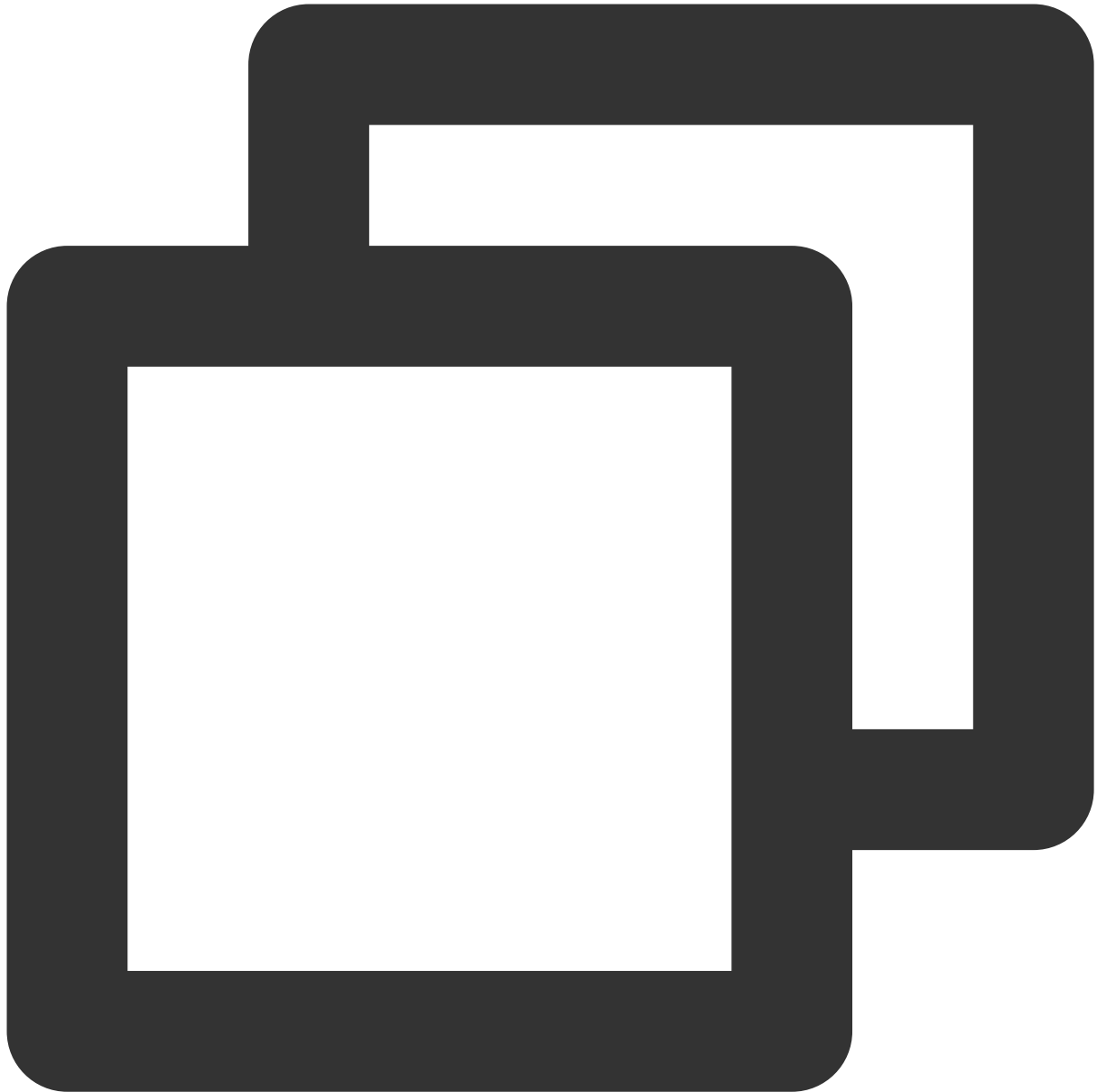
```
void onWarning(int code, String message);
```

The parameters are described below:

Parameter	Type	Description
code	int	Error code
message	String	Warning message

## onDebugLog

Callback for log.



```
void onDebugLog(String message);
```

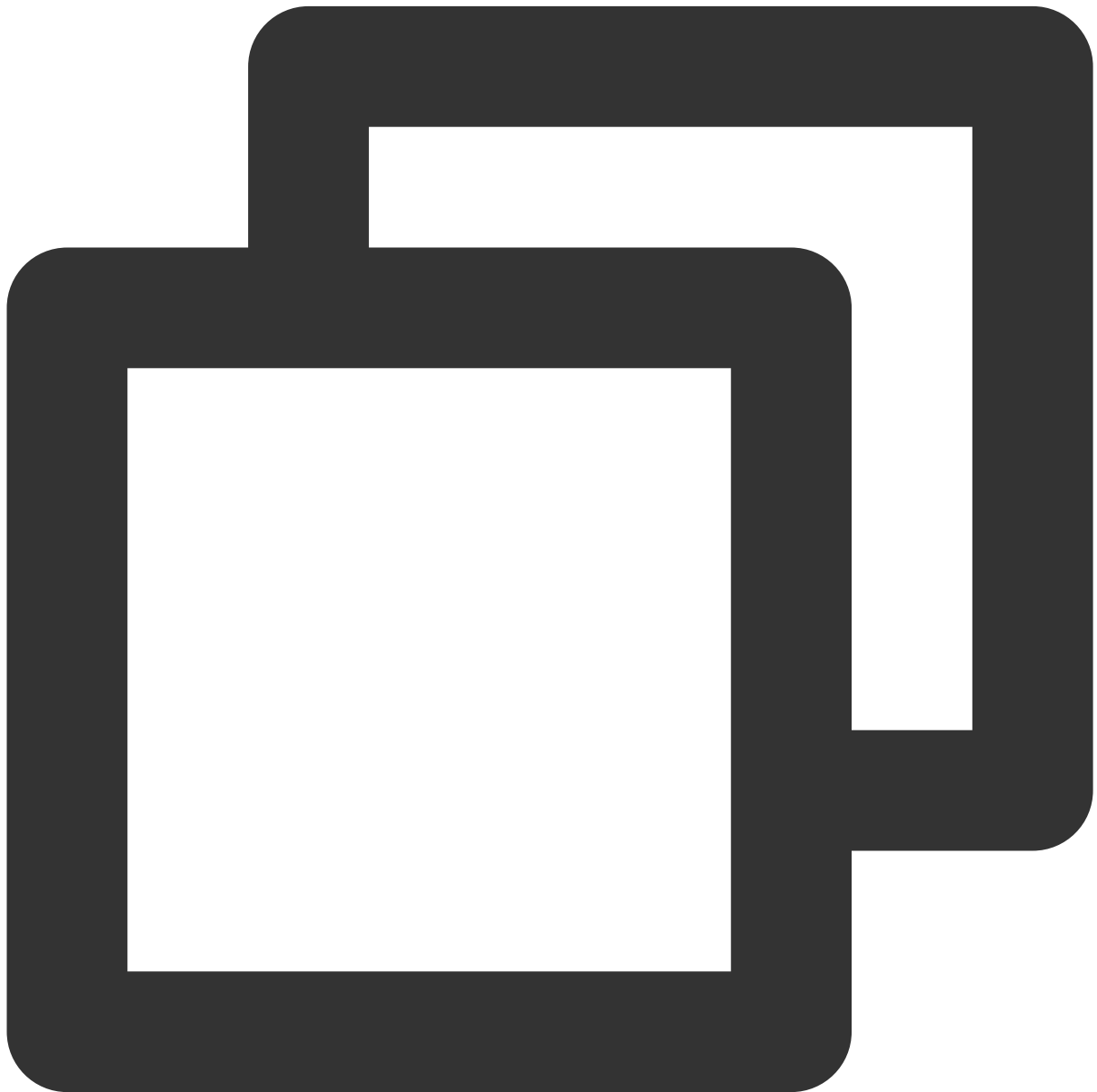
The parameters are described below:

Parameter	Type	Description
message	String	Log information

## Room Event Callback APIs

### onRoomDestroy

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.



```
void onRoomDestroy(String roomId);
```

The parameters are described below:

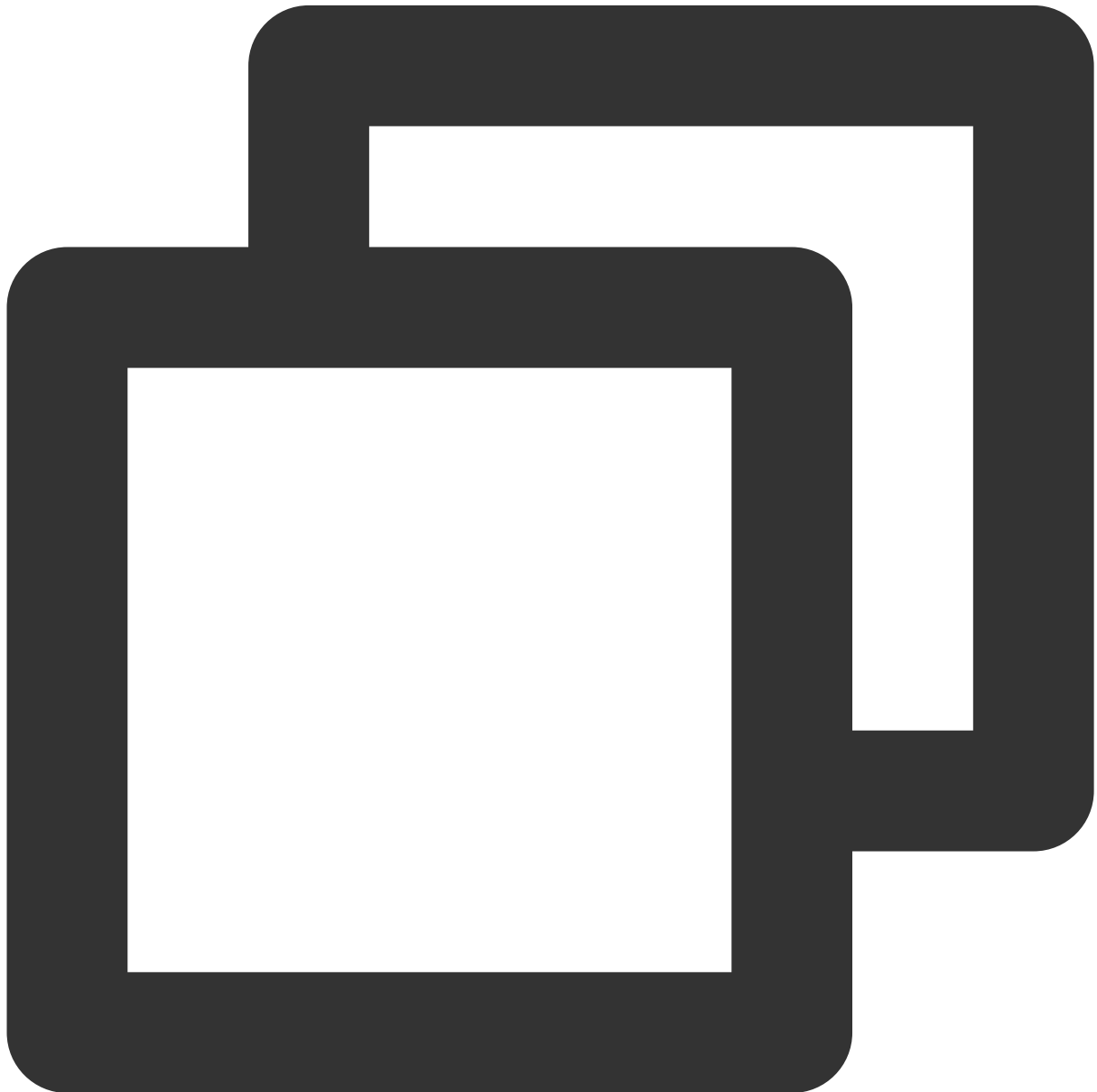
Parameter	Type	Description



roomId	String	Room ID
--------	--------	---------

## onRoomInfoChange

Callback for change of room information. This callback is sent after successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.



```
void onRoomInfoChange (TRTCVoiceRoomDef.RoomInfo roomInfo);
```

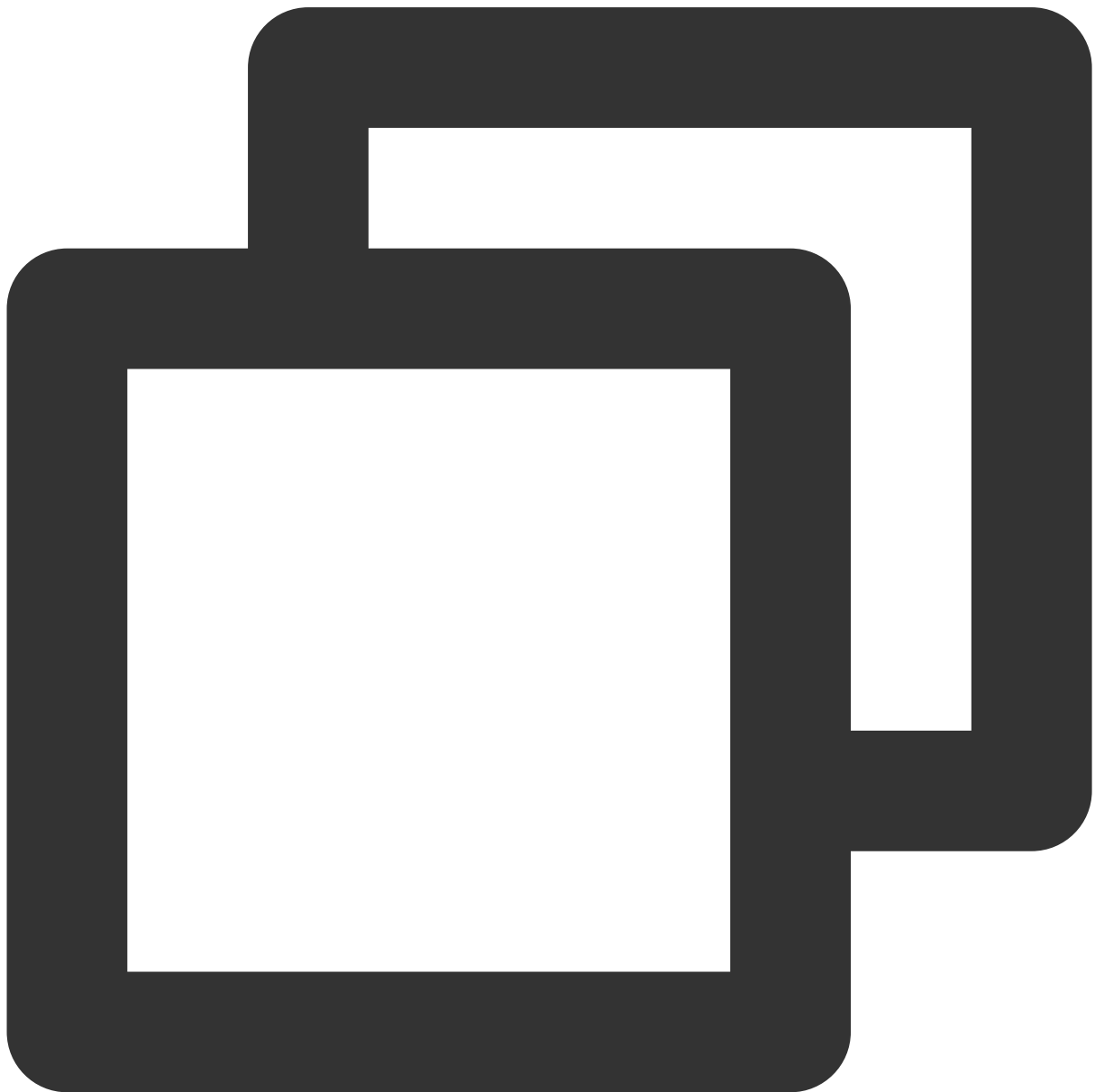
The parameters are described below:

--	--	--

Parameter	Type	Description
roomInfo	RoomInfo	Room information

### onUserMicrophoneMute

Callback of whether a user's mic is muted. When a user calls `muteLocalAudio`, all members in the room will receive this callback.



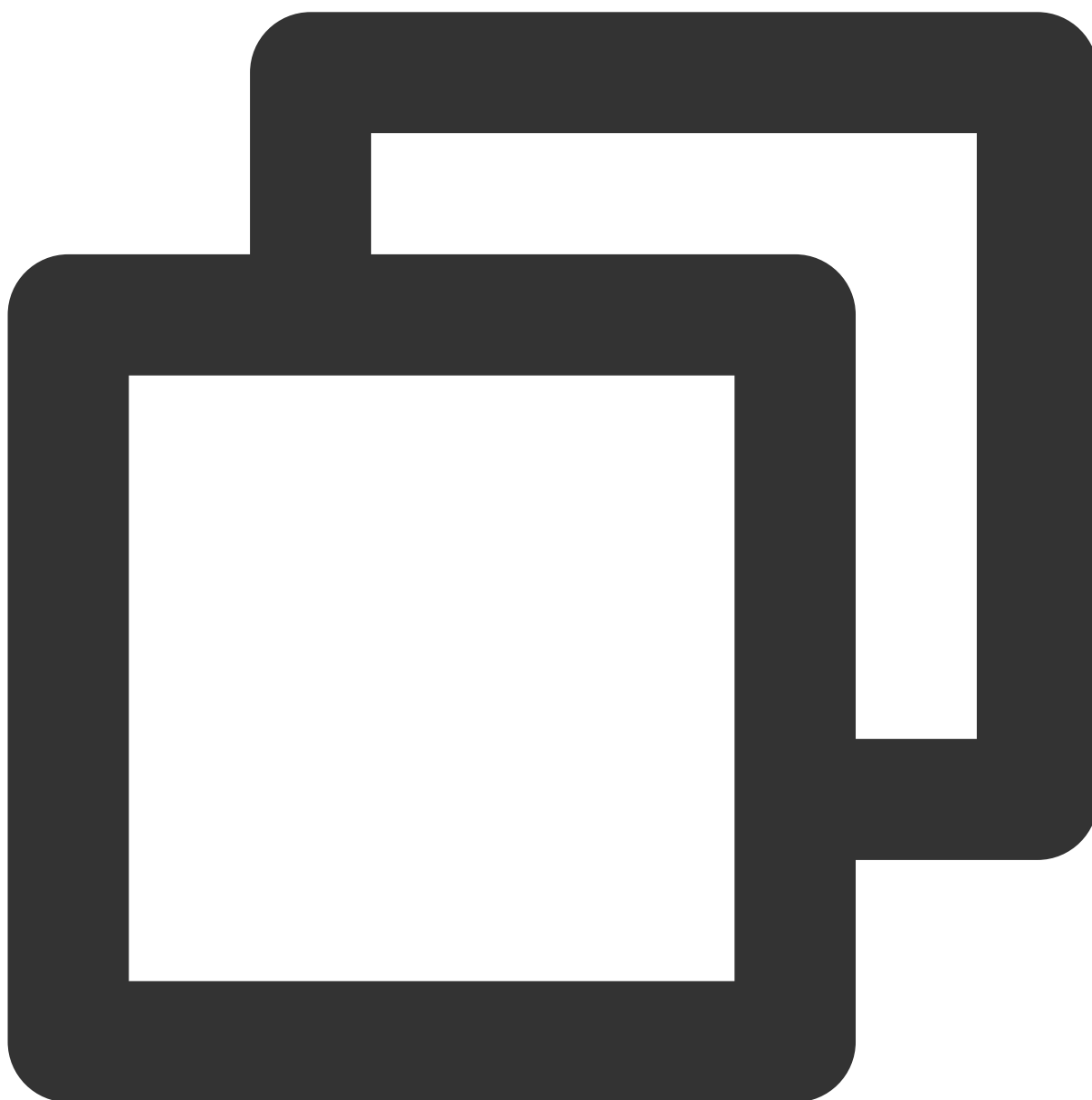
```
void onUserMicrophoneMute(String userId, boolean mute);
```

The parameters are described below:

Parameter	Type	Description
userId	String	User ID
mute	boolean	<code>true</code> : muted; <code>false</code> : unmuted

### onUserVolumeUpdate

Notification to all members of the volume after the volume reminder is enabled.



```
void onUserVolumeUpdate(List<TRTCCloudDef.TRTCVolumeInfo> userVolumes, int totalVol
```

The parameters are described below:

Parameter	Type	Description
userVolumes	ListList<TRTCCloudDef.TRTCVolumeInfo>	List of user IDs
totalVolume	int	Total volume. Value range: 0-100

## Seat Callback APIs

### onSeatListChange

Callback for all seat changes.



```
void onSeatListChange(List<SeatInfo> seatInfoList);
```

The parameters are described below:

Parameter	Type	Description
seatInfoList	List<SeatInfo>	Full seat list

### **onAnchorEnterSeat**

Someone became a speaker or was made a speaker by the owner.



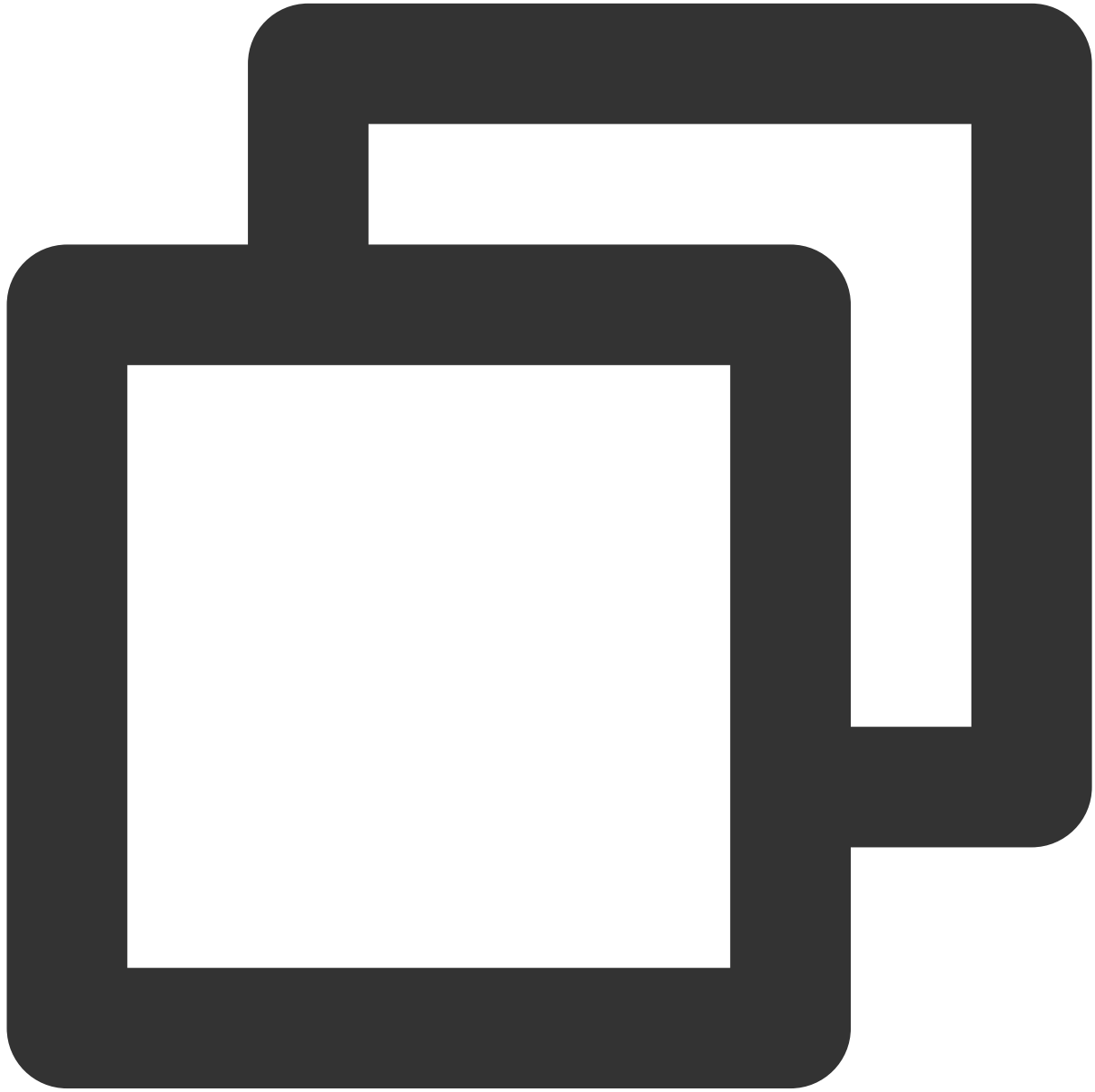
```
void onAnchorEnterSeat(int index, TRTCVoiceRoomDef.UserInfo user);
```

The parameters are described below:

Parameter	Type	Description
index	int	The seat taken
user	UserInfo	Details of the user who took the seat

## onAnchorLeaveSeat

A speaker became a listener or was moved to listeners by the room owner.



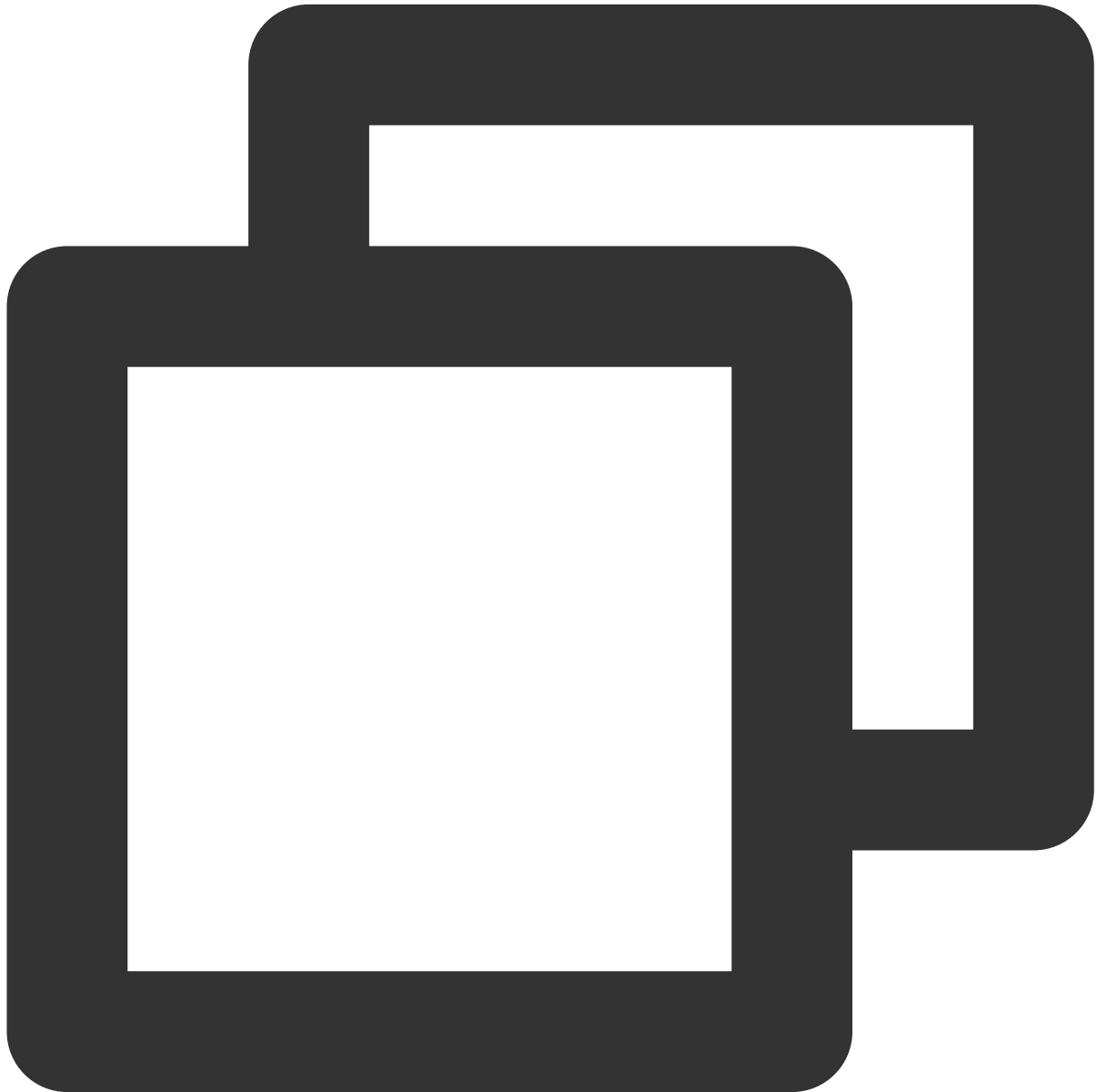
```
void onAnchorLeaveSeat(int index, TRTCVoiceRoomDef.UserInfo user);
```

The parameters are described below:

Parameter	Type	Description
index	int	The seat previously occupied by the speaker
user	UserInfo	Details of the user who took the seat

## onSeatMute

The room owner muted/unmuted a seat.



```
void onSeatMute(int index, boolean isMute);
```

The parameters are described below:

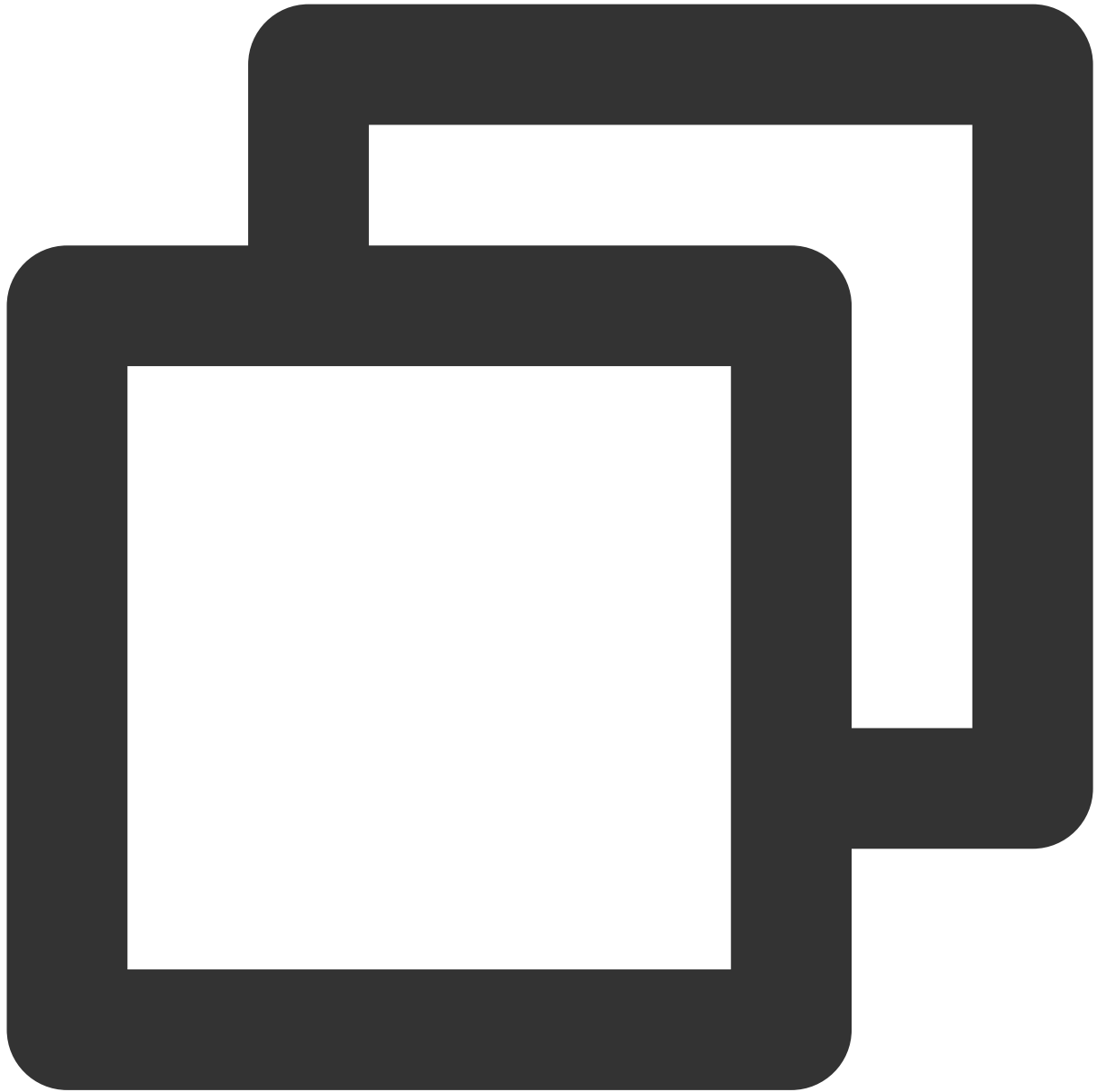
Parameter	Type	Description
index	int	The seat muted/unmuted
isMute	boolean	



```
true : muted; false : unmuted
```

## onSeatClose

The room owner blocked/unblocked a seat.



```
void onSeatClose(int index, boolean isClose);
```

The parameters are described below:

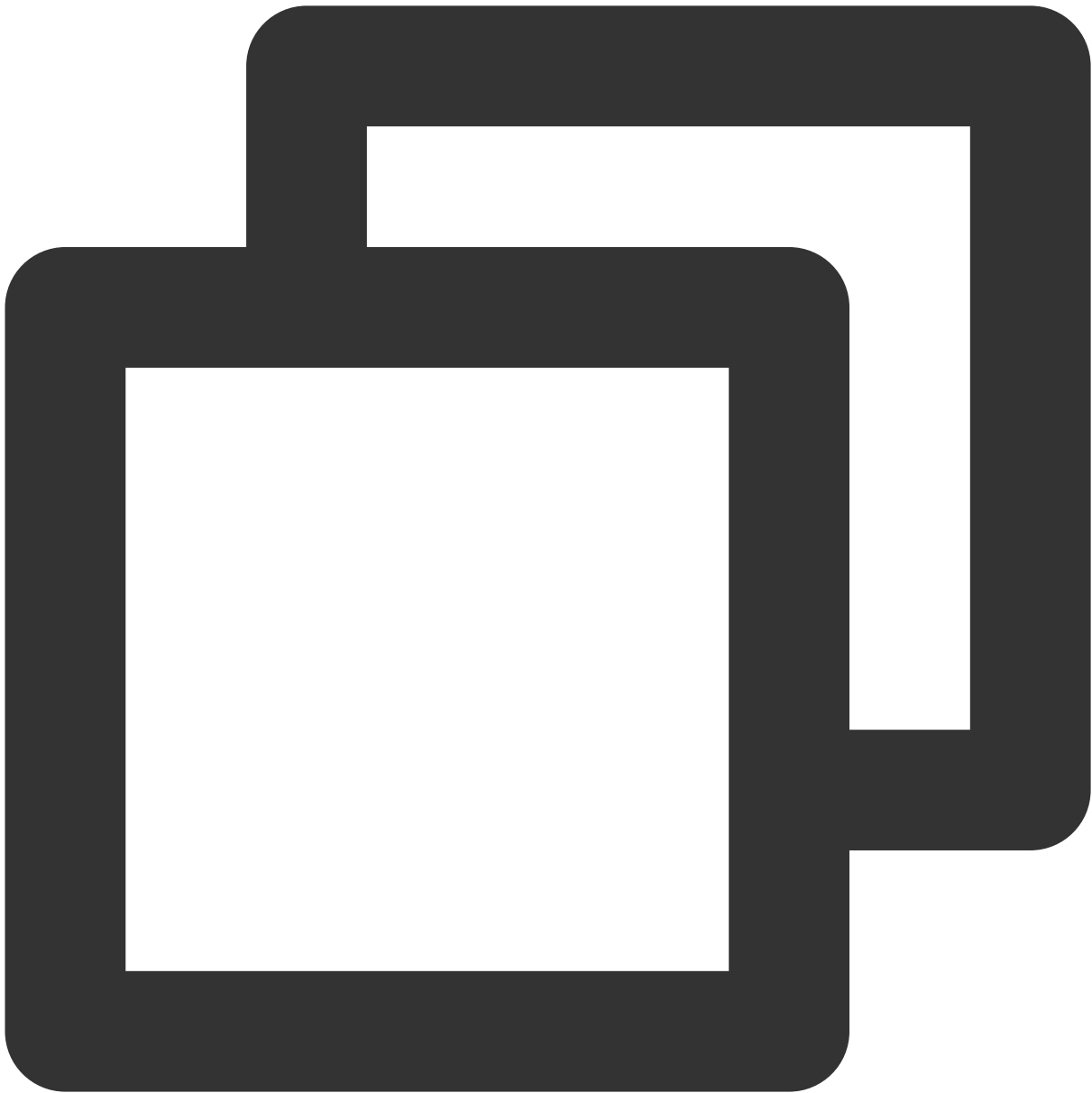
Parameter	Type	Description

index	int	The seat blocked/unblocked
isClose	boolean	true : blocked; false : unblocked

## Callback APIs for Room Entry/Exit by Listener

### onAudienceEnter

A listener entered the room.



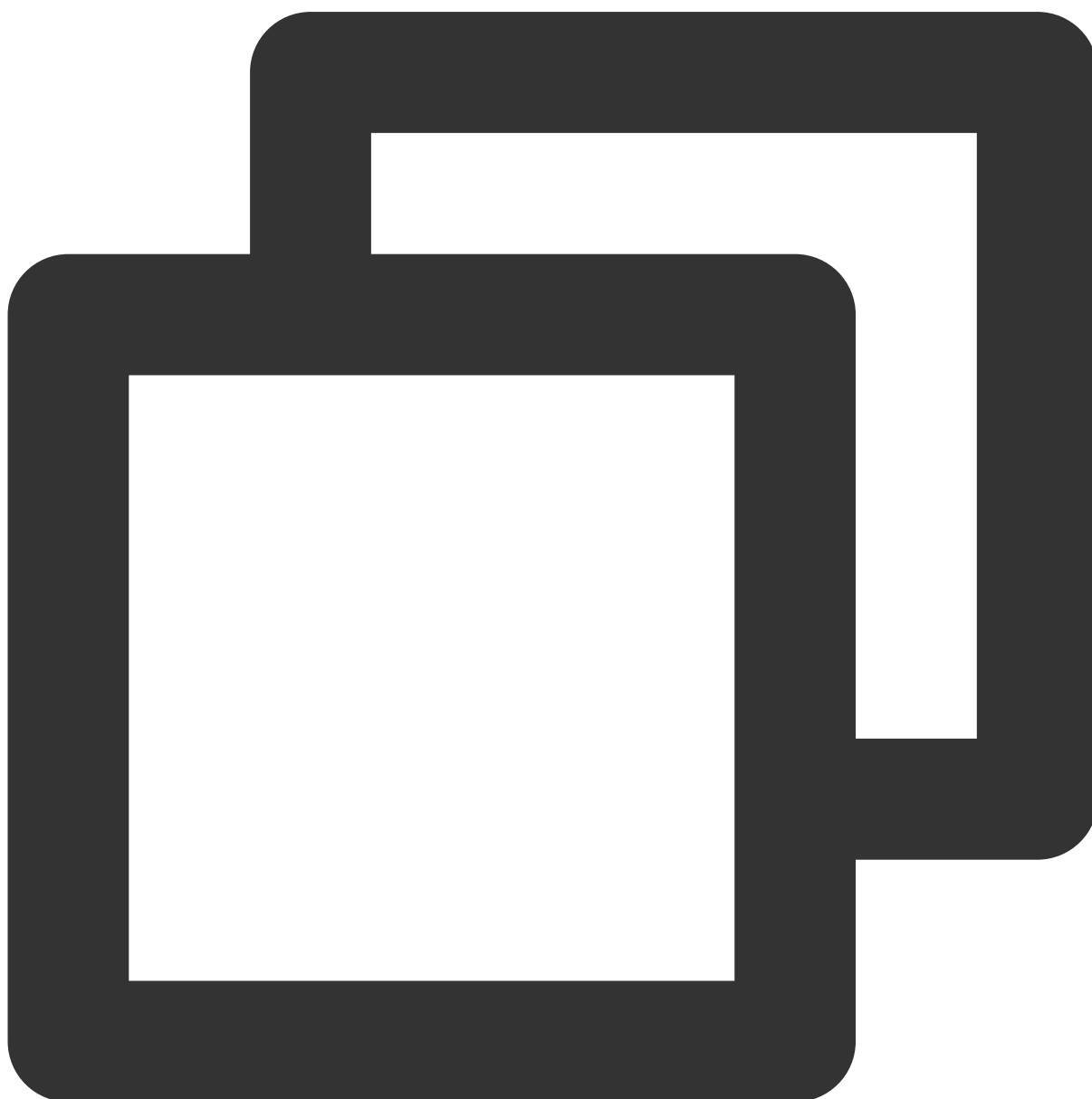
```
void onAudienceEnter (TRTCVoiceRoomDef.UserInfo userInfo);
```

The parameters are described below:

Parameter	Type	Description
userInfo	UserInfo	Information of the listener who entered the room

## onAudienceExit

A listener exited the room.



```
void onAudienceExit (TRTCVoiceRoomDef.UserInfo userInfo);
```

The parameters are described below:

Parameter	Type	Description
userInfo	UserInfo	Information of the listener who exited the room

## Message Event Callback APIs

### **onRecvRoomTextMsg**

Callback for receiving a text chat message.



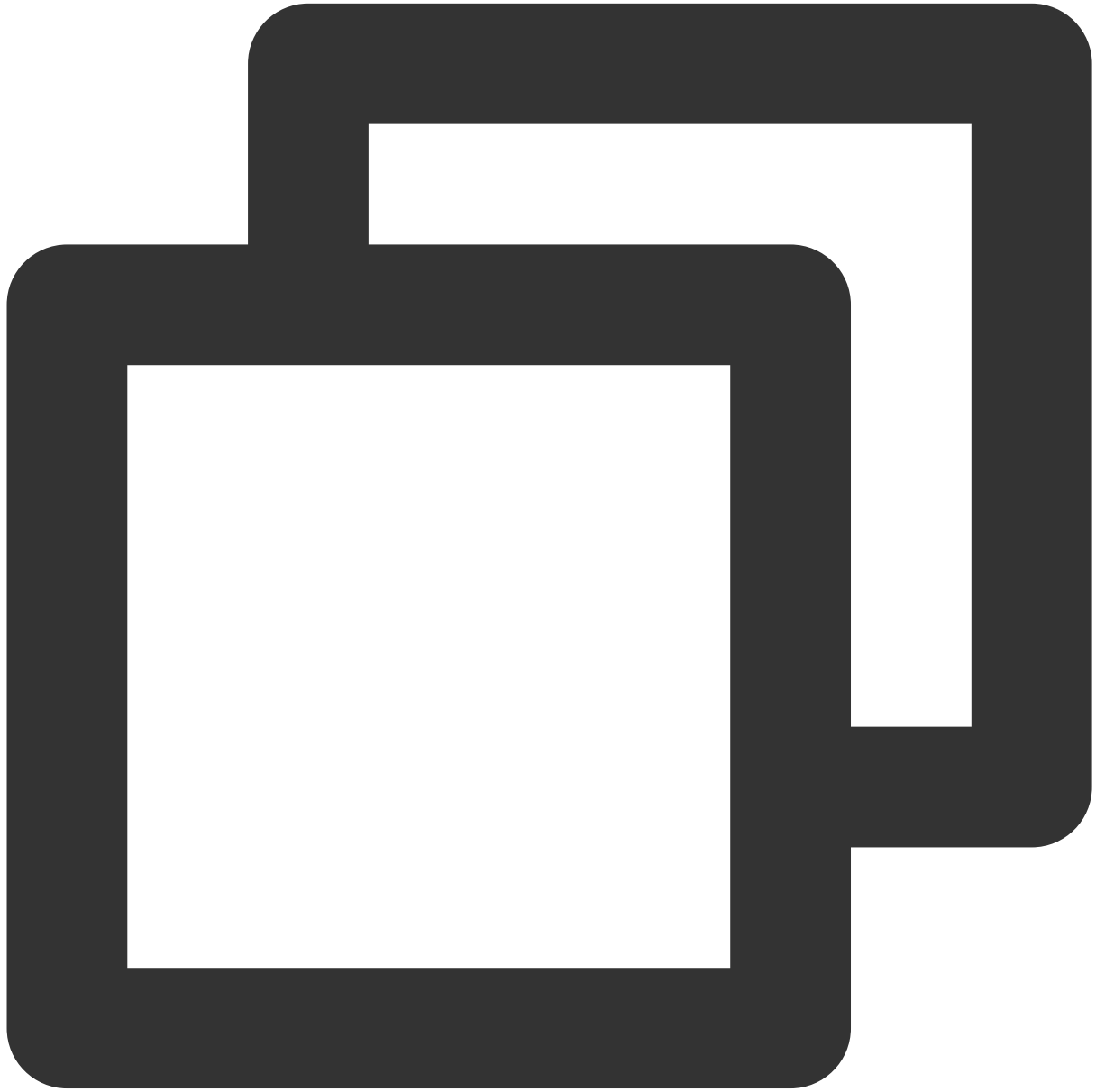
```
void onRecvRoomTextMsg(String message, TRTCVoiceRoomDef.UserInfo userInfo);
```

The parameters are described below:

Parameter	Type	Description
message	String	Text message
userInfo	UserInfo	Information of the sender

### **onRecvRoomCustomMsg**

A custom message was received.



```
void onRecvRoomCustomMsg(String cmd, String message, TRTCVoiceRoomDef.UserInfo user)
```

The parameters are described below:

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message

userInfo

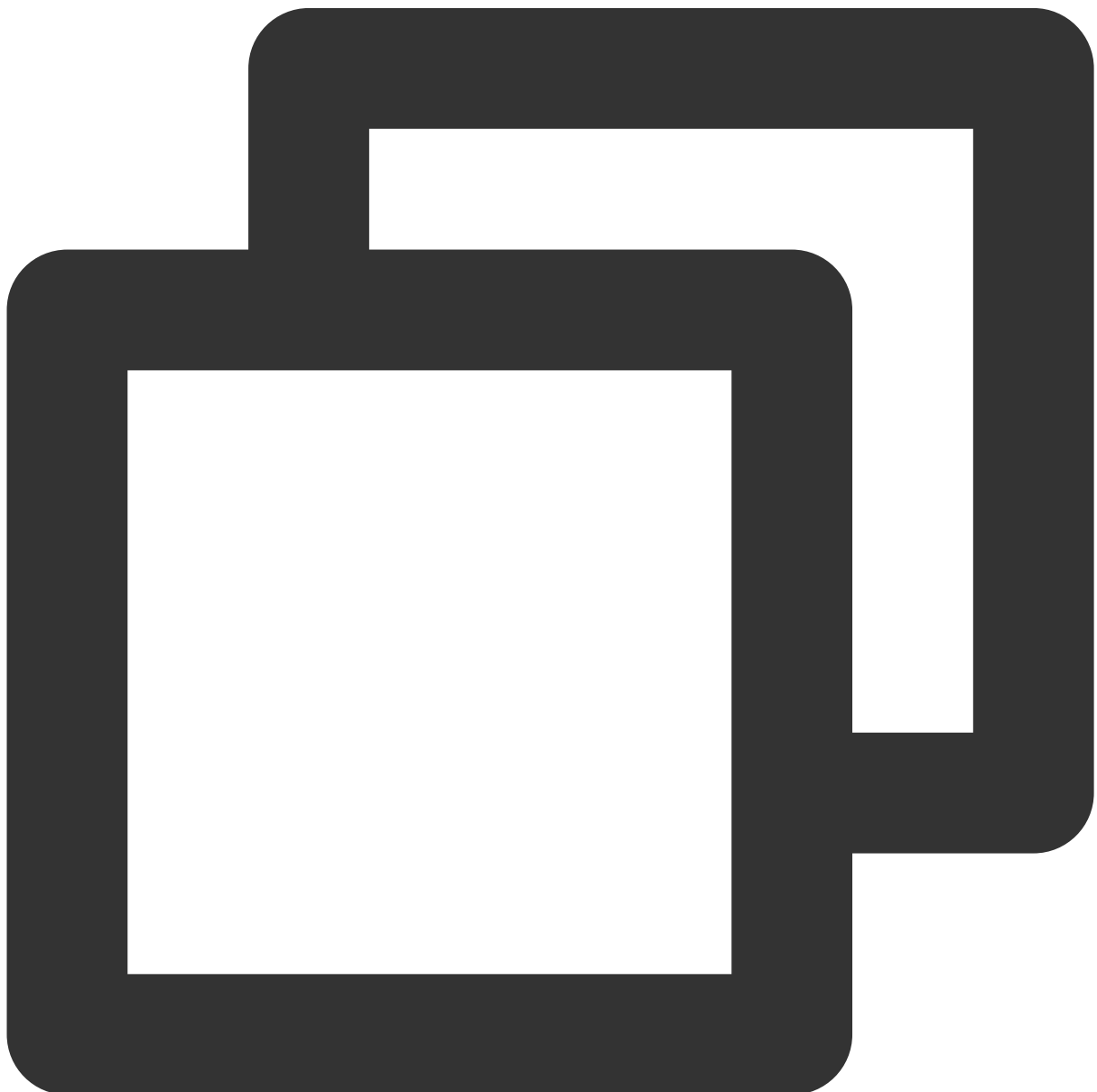
UserInfo

Information of the sender

## Invitation Signaling Callback APIs

### **onReceiveNewInvitation**

An invitation was received.



```
void onReceiveNewInvitation(String id, String inviter, String cmd, String content);
```

The parameters are described below:

Parameter	Type	Description
id	String	Invitation ID
inviter	String	Inviter's user ID
cmd	String	Custom command word specified by business
content	String	Content specified by business

### **onInviteeAccepted**

The invitee accepted the invitation





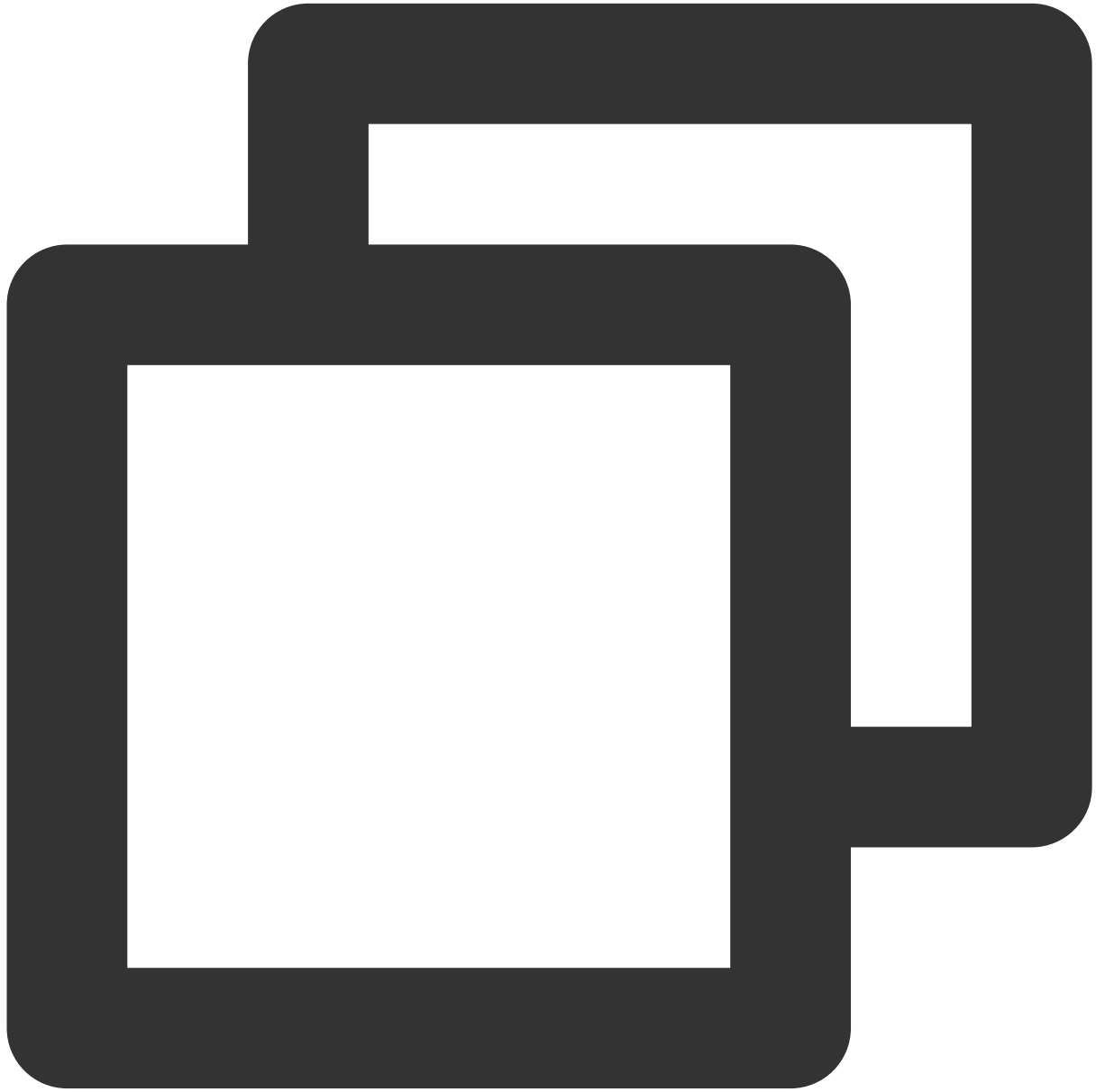
```
void onInviteeAccepted(String id, String invitee);
```

The parameters are described below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee's user ID

### **onInviteeRejected**

The invitee declined the invitation



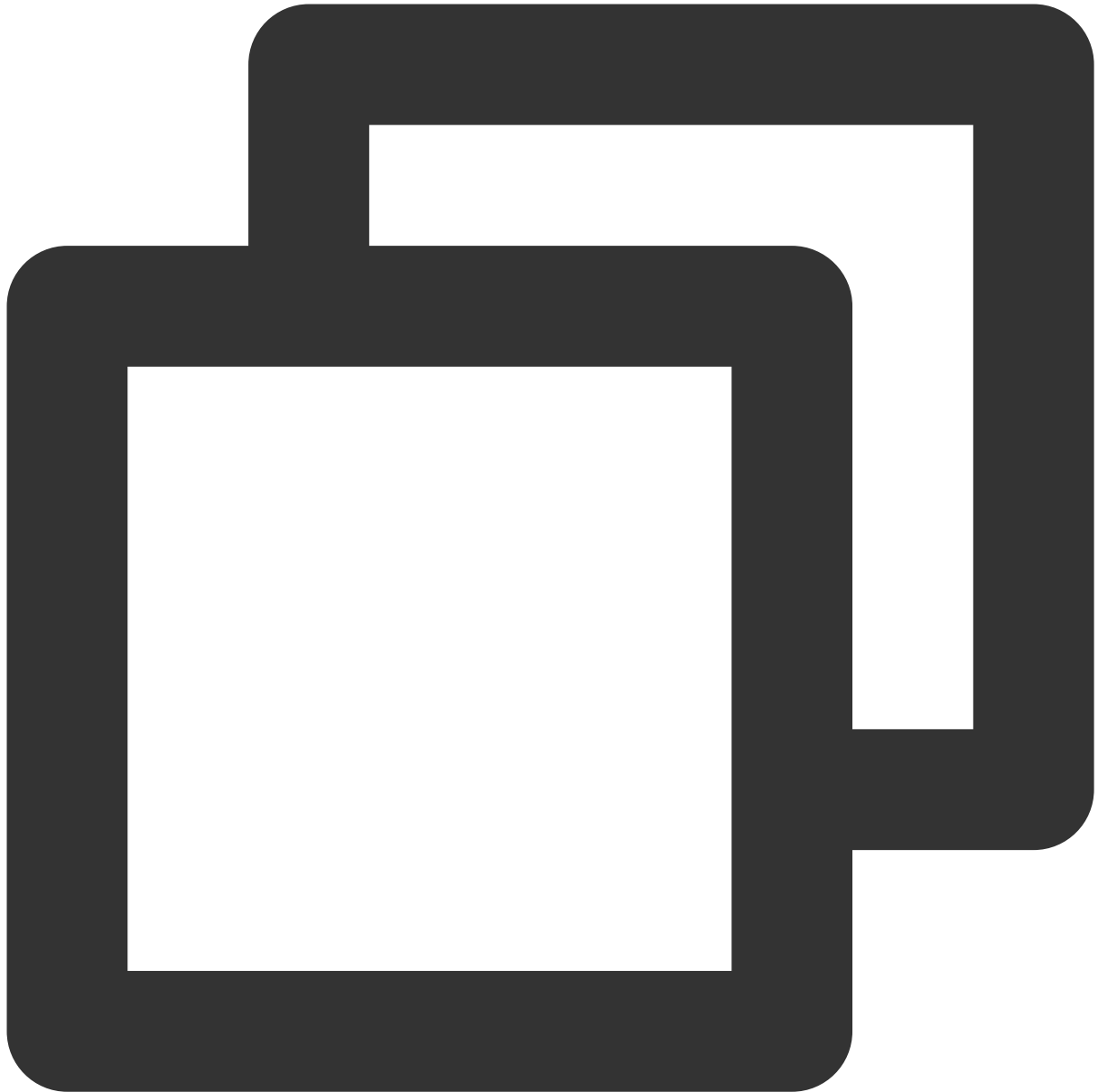
```
void onInviteeRejected(String id, String invitee);
```

The parameters are described below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee's user ID

## onInvitationCancelled

The inviter canceled the invitation.



```
void onInvitationCancelled(String id, String inviter);
```

The parameters are described below:

Parameter	Type	Description
id	String	Invitation ID
inviter	String	Inviter's user ID



# Interactive Video Streaming

## Integrating TUILiveRoom (Android)

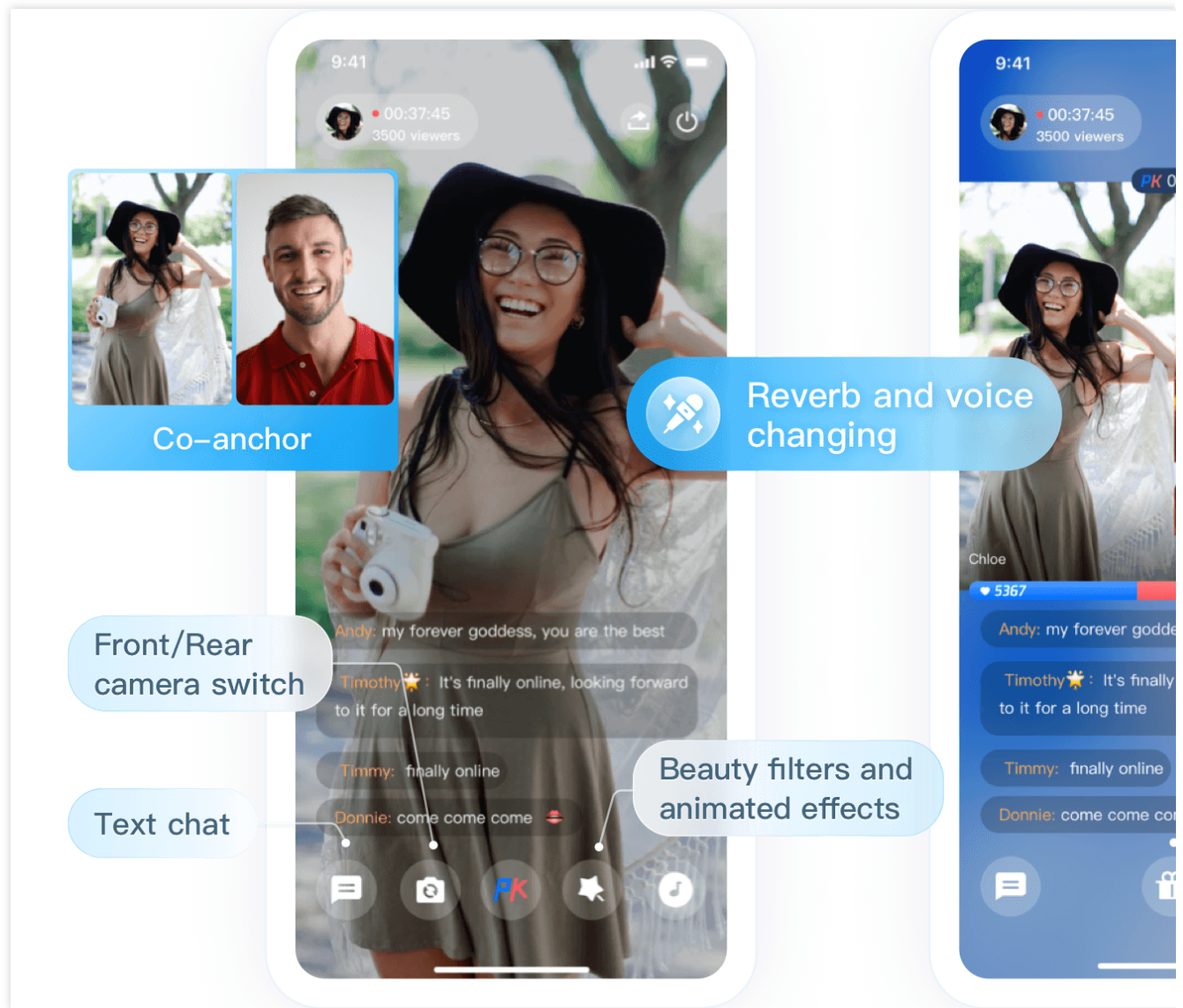
Last updated : 2023-09-21 16:22:21

### Overview

`TUILiveRoom` is an open-source video live streaming scenario UI component. After integrating it into your project, you can enable your application to support interactive video live streaming simply by writing a few lines of code. It provides source code for Android, iOS, and mini program platforms. Its basic features are as shown below:

#### Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see [Pricing](#).

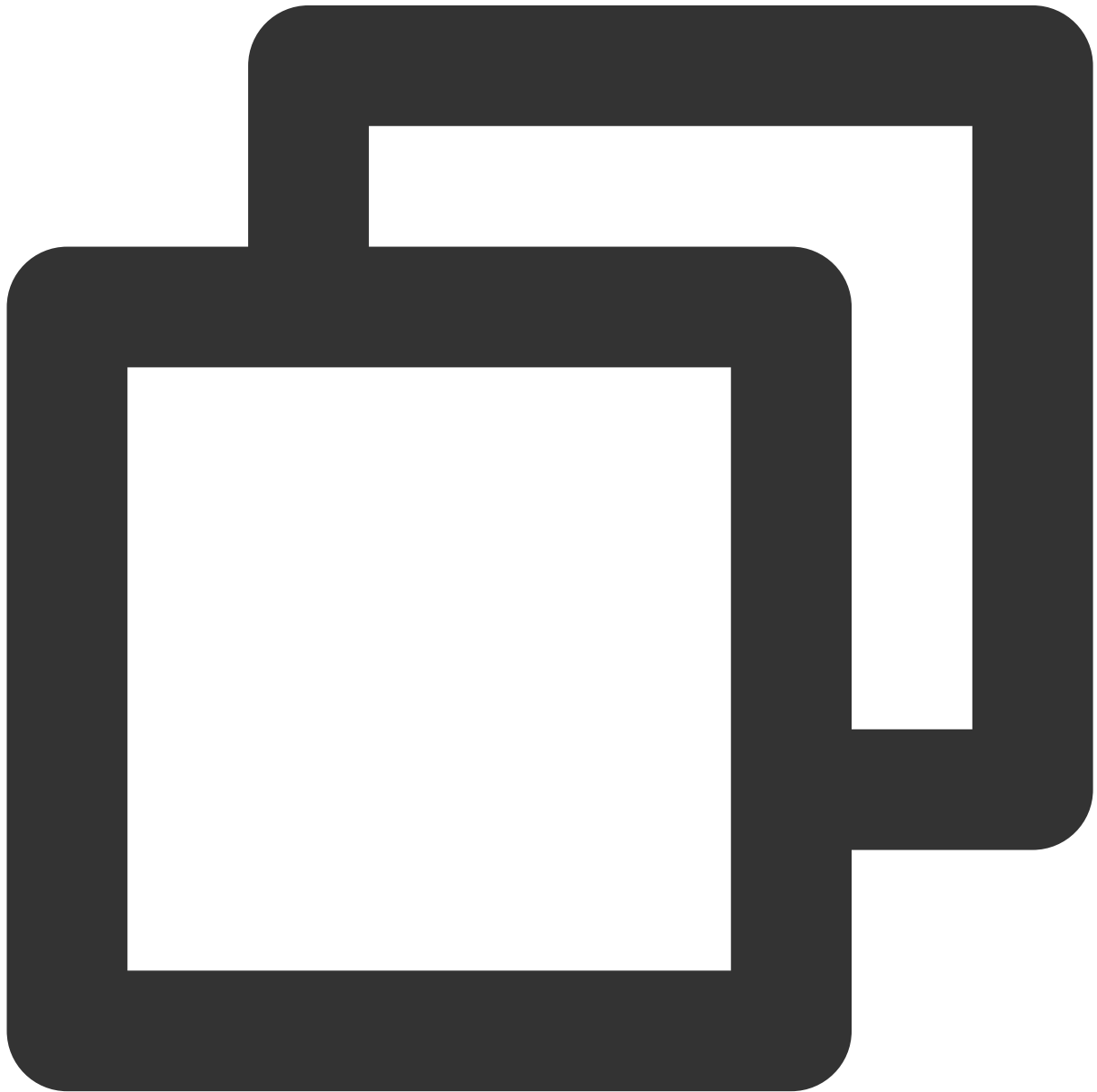


## Integration

### Step 1. Download and import the `TUILiveRoom` component

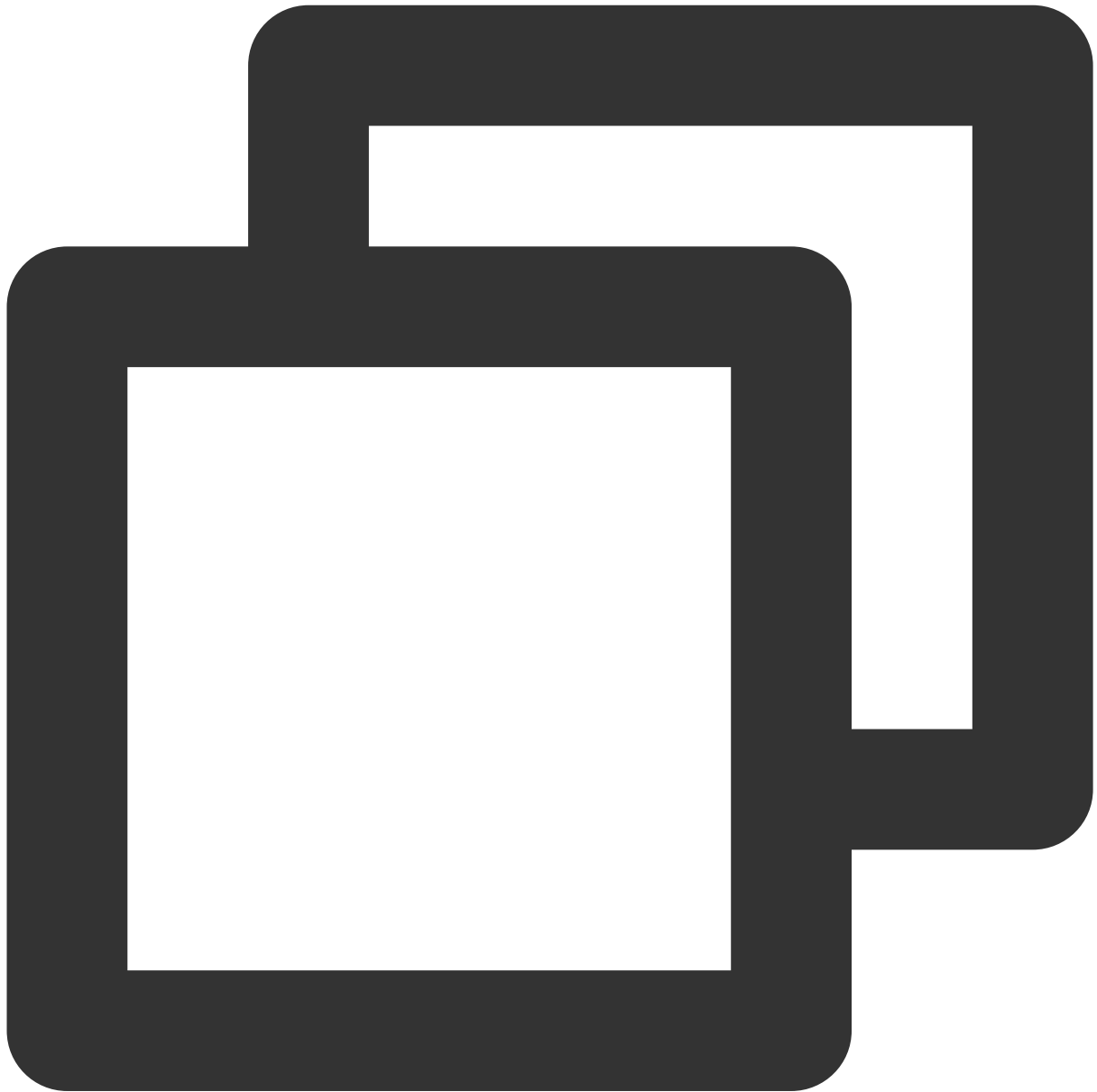
Go to [GitHub](#), clone or download the code, copy the `Android/debug` , `Android/tuiaudioeffect` , `Android/tuibarrage` , `Android/tuibeaauty` , `Android/tuigift` , and `Android/tuiliveroom` directories to your project, and complete the following import operations:

Add the code below in `setting.gradle` :



```
include ':debug'  
include ':tuibeauty'  
include ':tuibarrage'  
include ':tuiaudioeffect'  
include ':tuigift'  
include ':tuiliveroom'
```

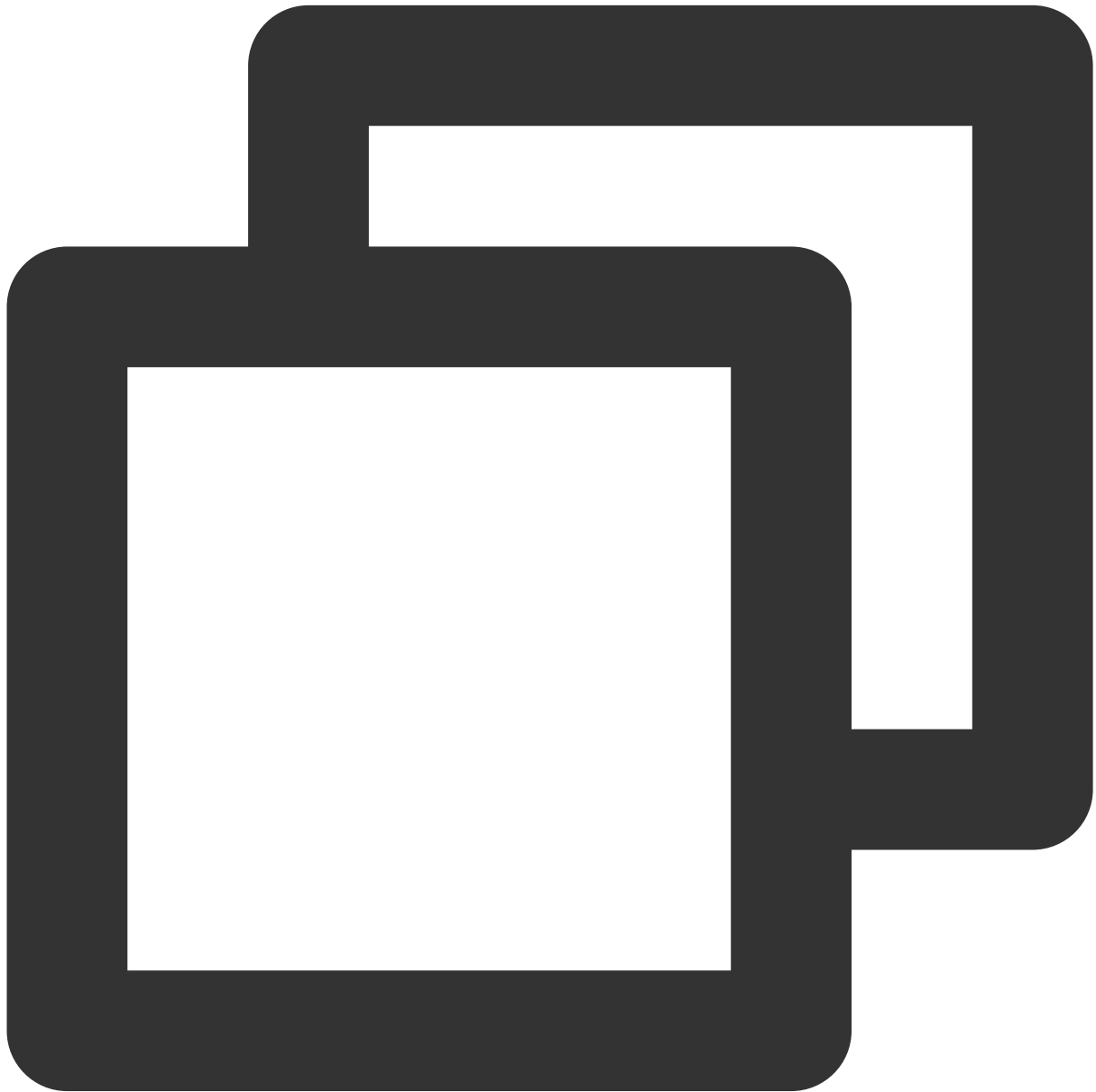
Add dependencies on `tuiliveroom` to the `build.gradle` file in `app` :



```
api project(":tuiliveroom")
```

Add the TRTC SDK ( `liteavSdk` ) and Chat SDK ( `imSdk` ) dependencies in `build.gradle` in the root directory:

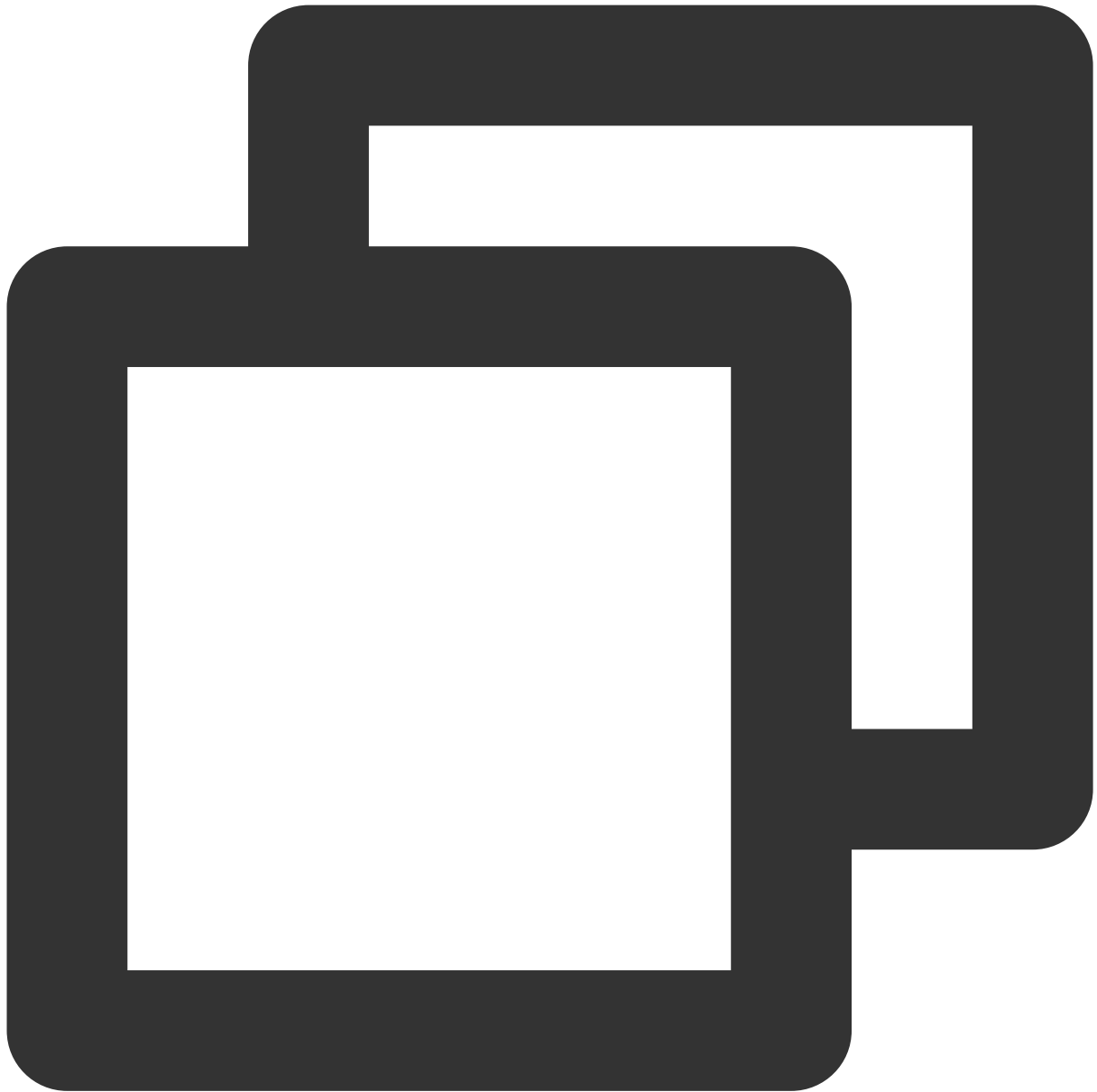




```
ext {  
    liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"  
    imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"  
}
```

## Step 2. Configure permission requests and obfuscation rules

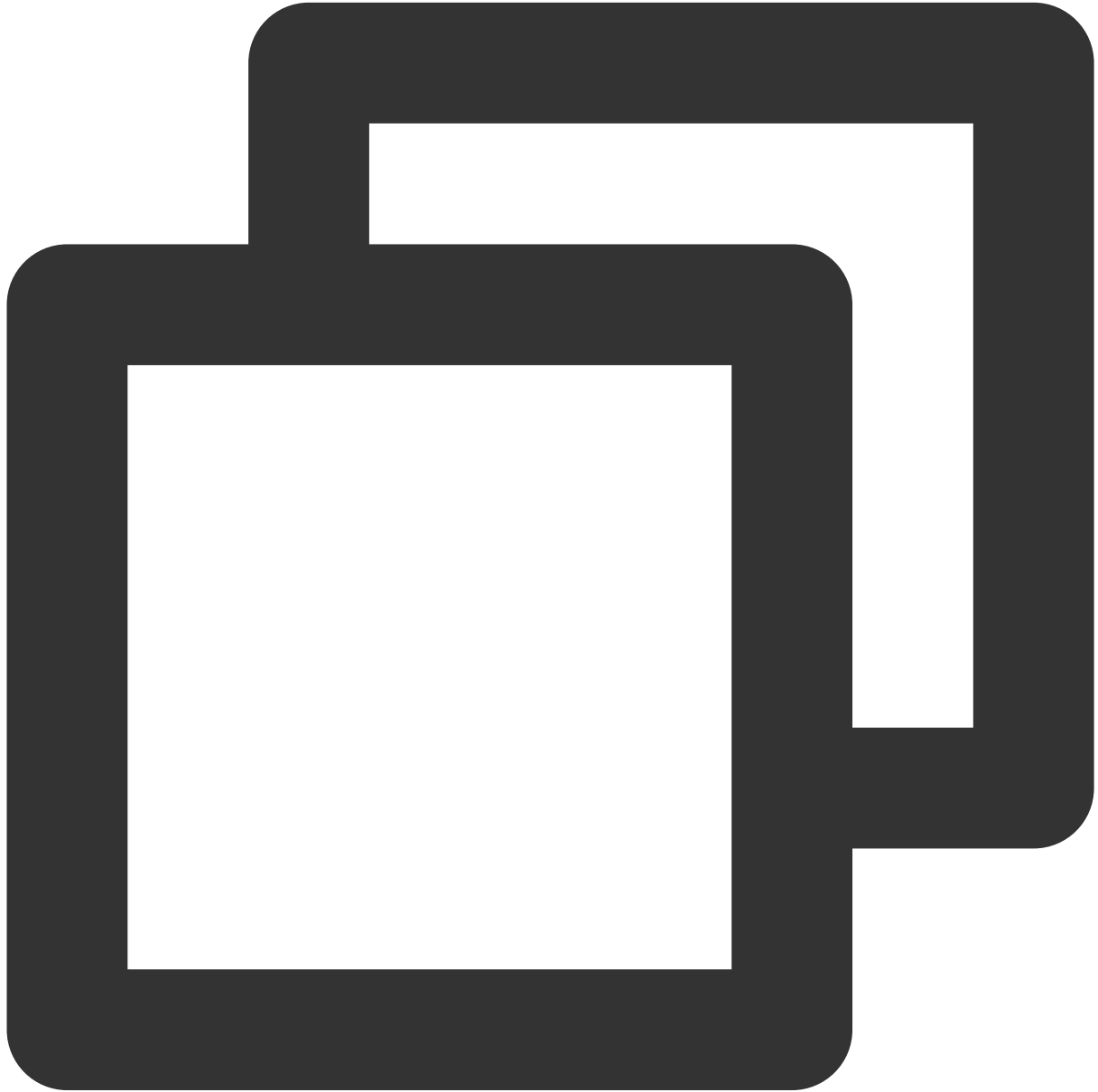
1. Configure permission requests for your application in `AndroidManifest.xml`. The SDKs need the following permissions (on Android 6.0 and later, the camera and storage read permissions must be requested at runtime.)



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

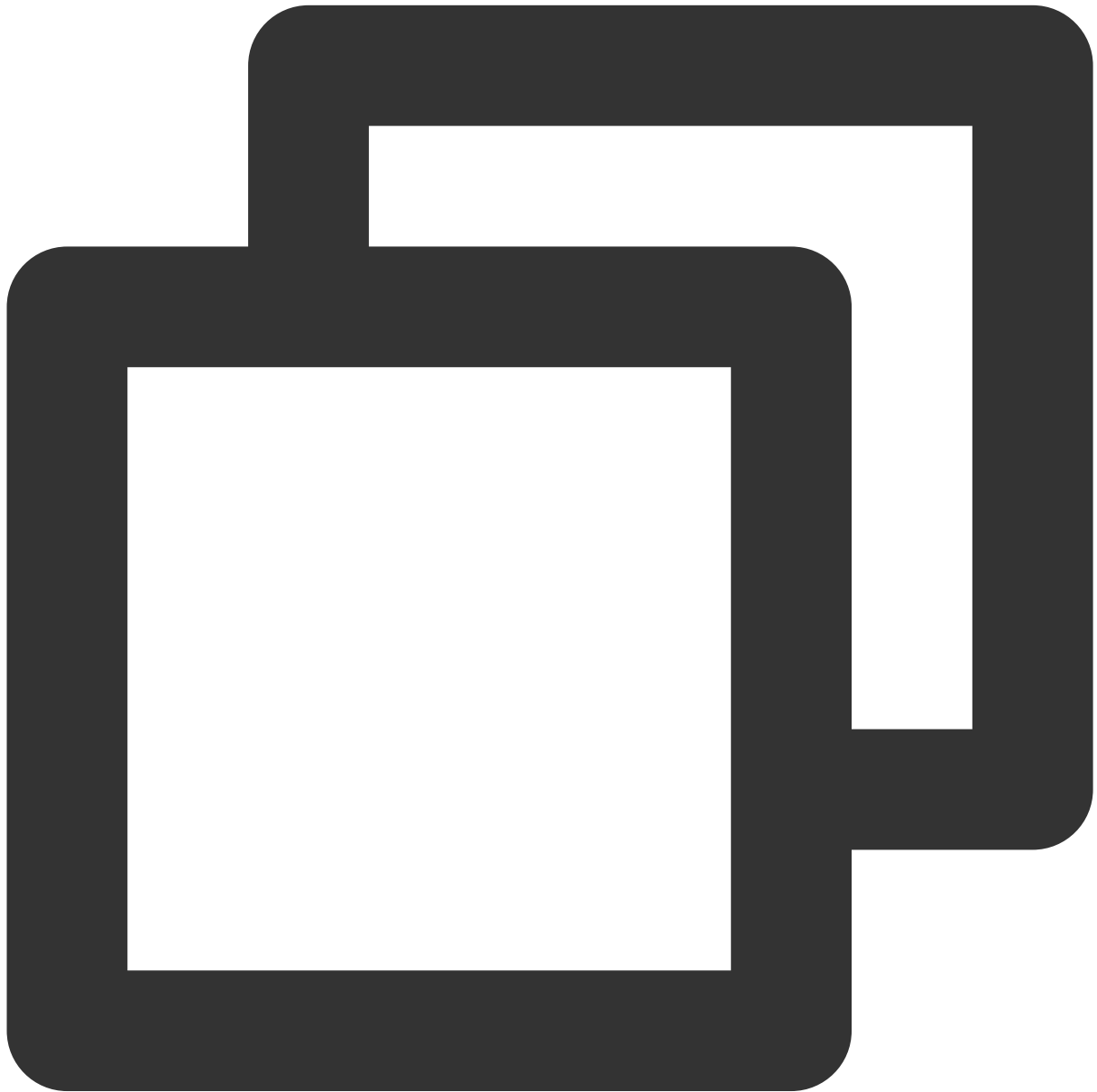
```
<uses-feature android:name="android.hardware.camera.autofocus" />
```

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



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

### Step 3. Initialize and log in to the component



```
// 1. Add a listener and log in
TUILogin.addLoginListener(new TUILoginListener() {
    @Override
    public void onConnecting() {          // Connecting ...
        super.onConnecting();
    }
    @Override
    public void onConnectSuccess() {      // Connected
        super.onConnectSuccess();
    }
    @Override
```

```
public void onConnectFailed(int errorCode, String errorMsg) { // Failed to con
    super.onConnectFailed(errorCode, errorMsg);
}
@Override
public void onKickedOffline() { // Callback for forced logout (for example, du
    super.onKickedOffline();
}
@Override
public void onUserSigExpired() { // Callback for `userSig` expiration
    super.onUserSigExpired();
}
});
TUILogin.login(mContext, "Your SDKAppID", "Your userId", "Your userSig", new TUICal
    @Override
    public void onSuccess() {
    }
    @Override
    public void onError(int errorCode, String errorMsg) {
        Log.d(TAG, "errorCode: " + errorCode + " errorMsg:" + errorMsg);
    }
});

// 2. Initialize the `TUILiveRoom` component
TUILiveRoom mLiveRoom = TUILiveRoom.sharedInstance(mContext);
```

**Parameter description:**

**SDKAppID:** TRTC application ID. If you haven't activated TRTC, log in to the [TRTC console](#), create a TRTC application, click **Application Info**, and select the **Quick Start** tab to view its `SDKAppID` .

**Tencent Real-Time Communication**

Application Management - rr ( ) SDKApp

Application Info Function Configuration Callback Configuration

**Step 1: download SDK + auxiliary demo source code**

iOS Mac Android Web Windows Flutter

**Step 2: obtain the secret key to issue UserSig**

The secret key is sensitive information. Please do not disclose it.

Secret Key (Key)

[Redacted Secret Key]

Copy Secret Key

HMAC-SHA256 encrypted

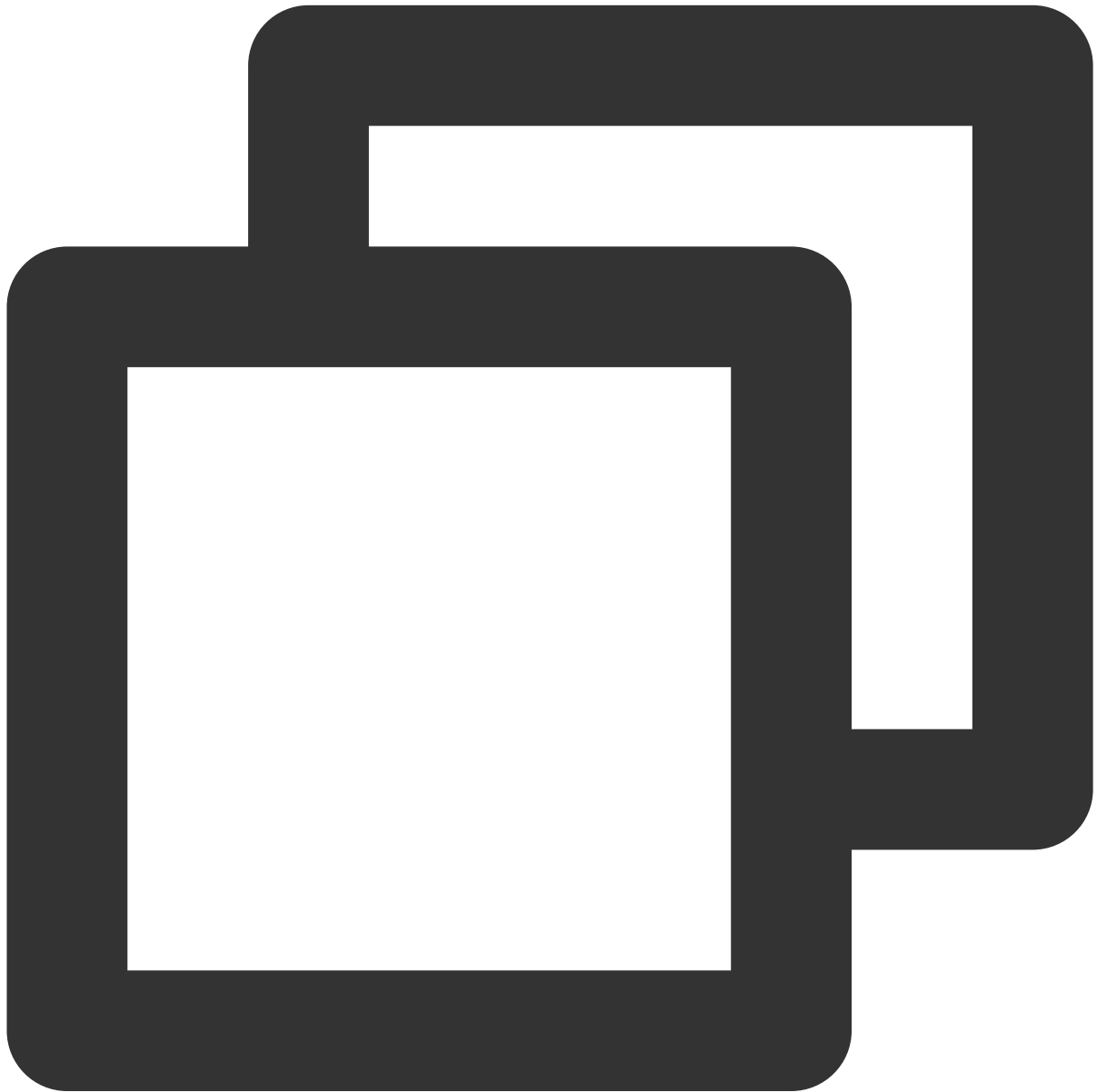
**Secretkey: TRTC application key.** Each secret key corresponds to a `SDKAppID` . You can view your application's secret key on the [Application Management](#) page of the TRTC console.

**userId:** ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (\_). We recommend that you keep it consistent with your user account system.

**UserSig:** Security signature calculated based on `SDKAppID` , `userId` , and `Secretkey` . You can click [here](#) to quickly generate a `UserSig` for testing. For more information, see [UserSig](#).

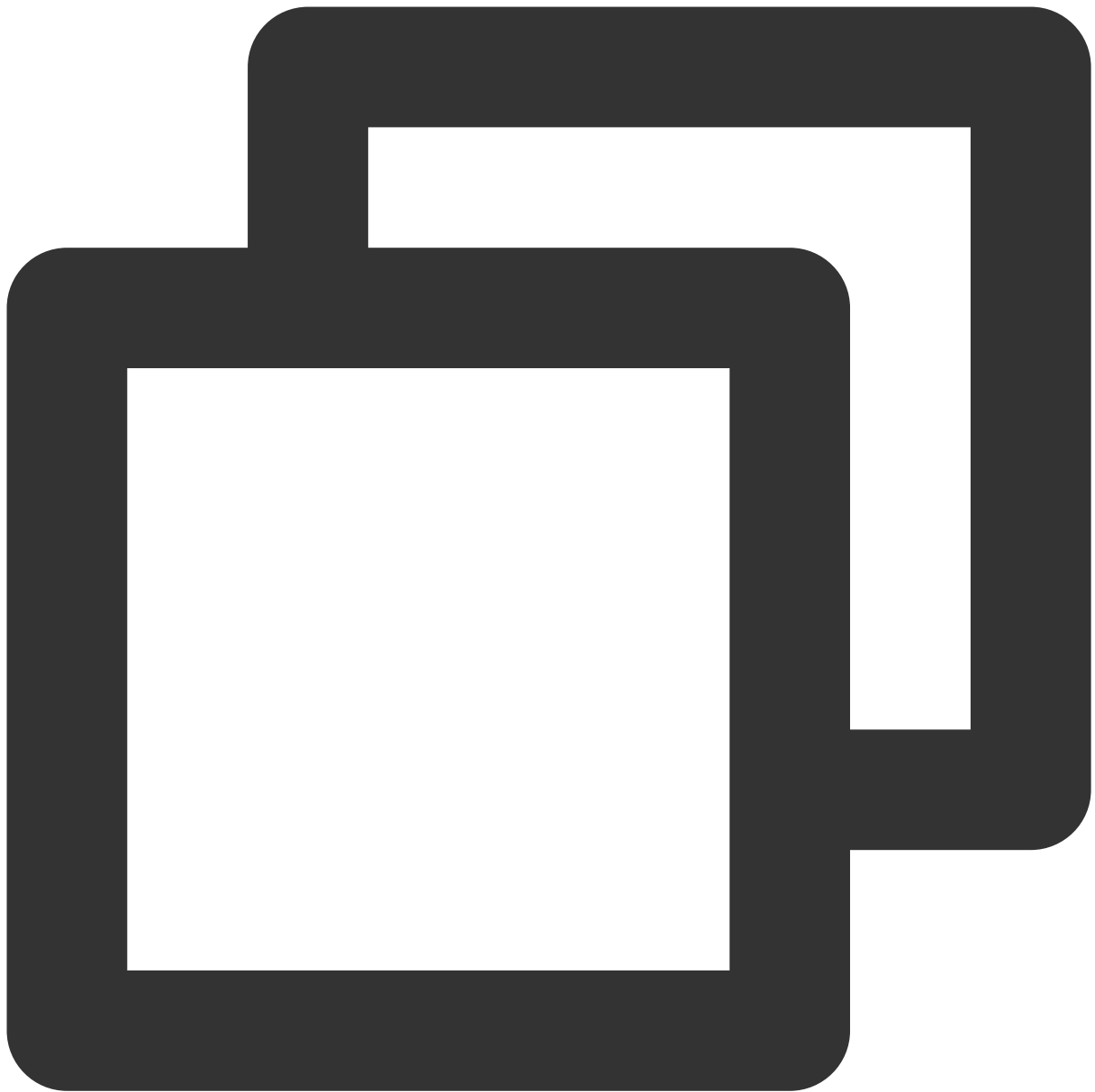
## Step 4. Implement an interactive video live room

### 1. The anchor starts streaming.



```
mLiveRoom.createRoom(int roomId, String roomName, String coverUrl);
```

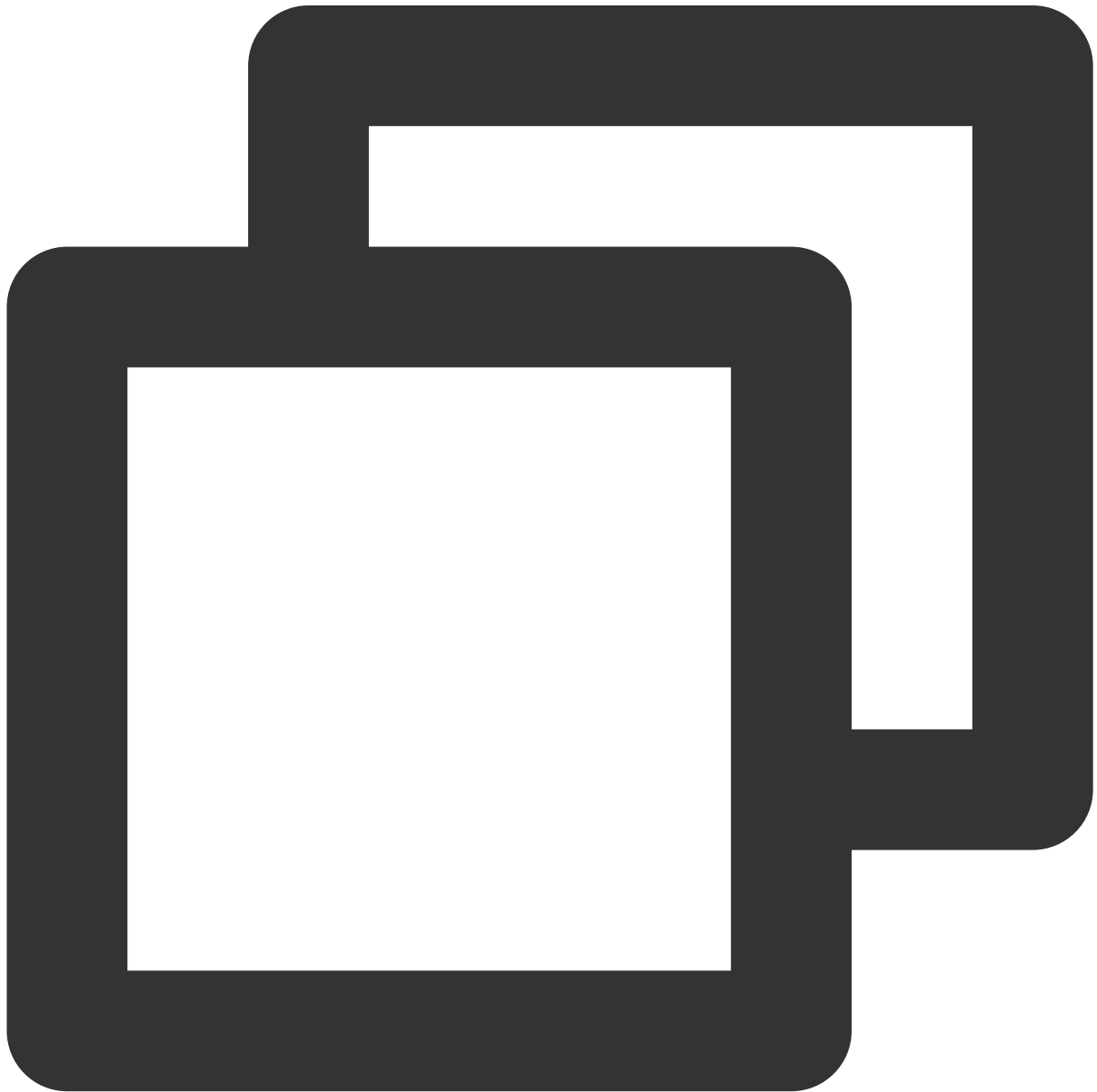
## 2. The audience member watches.



```
mLiveRoom.enterRoom(roomId);
```

3. Audience member and the anchor co-anchor together through [TRTCLiveRoom#requestJoinAnchor](#).





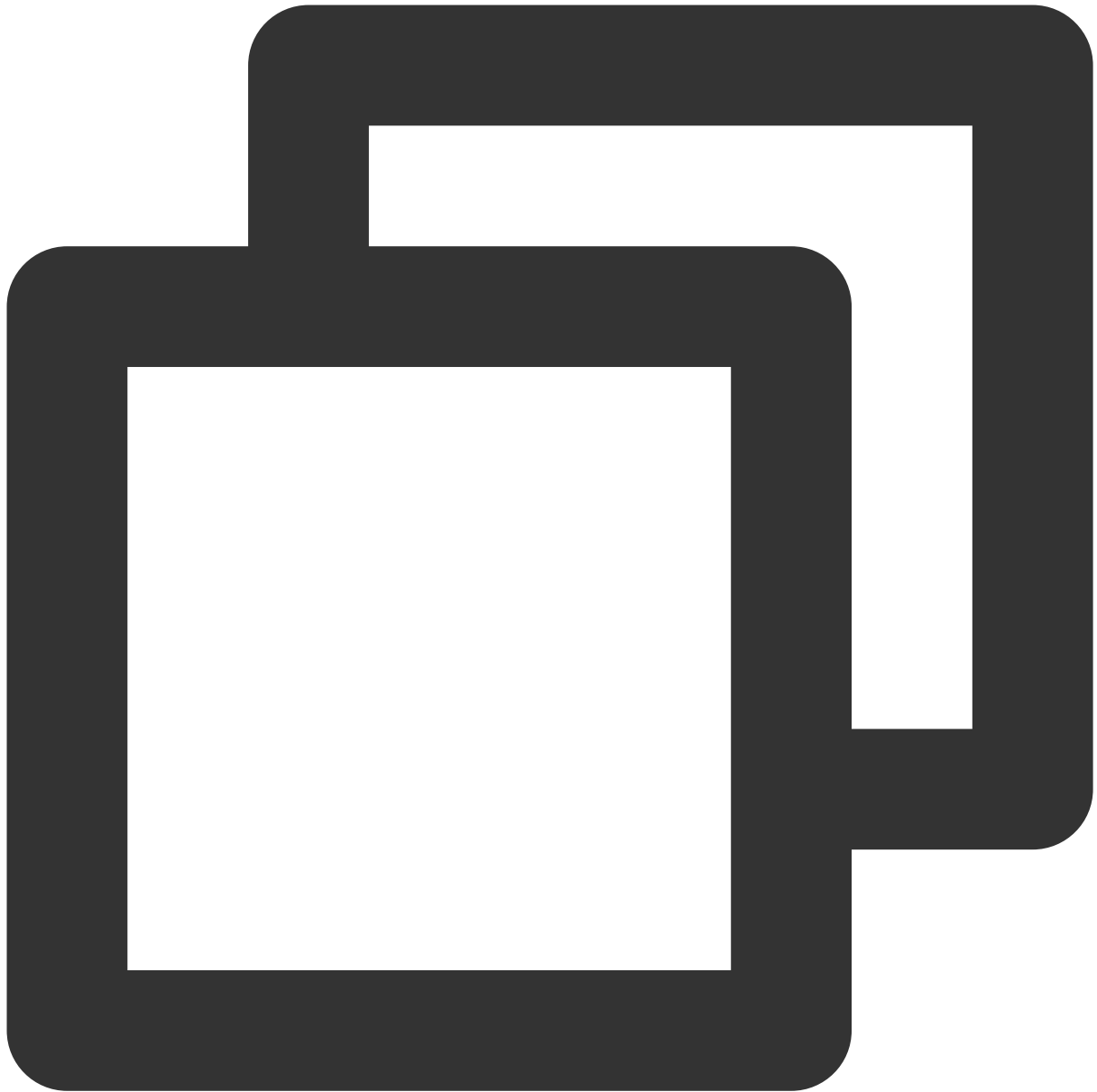
```
// 1. The audience member sends a co-anchoring request
// `LINK_MIC_TIMEOUT` is the timeout period
TRTCLiveRoom mTRTCLiveRoom=TRTCLiveRoom.sharedInstance(mContext);
mTRTCLiveRoom.requestJoinAnchor(mSelfUserId + "requested to co-anchor", LINK_MIC_TI
    new TRTCLiveRoomCallback.ActionCallback() {
        @Override
        public void onCallback(int code, String msg) {
            if (code == 0) {
                // The request is accepted by the anchor
                TXCloudVideoView view = new TXCloudVideoView(context);
                parentView.add(view);
            }
        }
    }
);
```

```
// The audience member turns on the camera and starts pushing streams
mTRTCLiveRoom.startCameraPreview(true, view, null);
mTRTCLiveRoom.startPublish(mSelfUserId + "_stream", null);
    }
}
});

// 2. The anchor receives the co-anchoring request
mTRTCLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
    @Override
    public void onRequestJoinAnchor(final TRTCLiveRoomDef.TRTCLiveUserInfo userInfo,
        String reason, final int timeout) {
        // The anchor accepts the co-anchoring request
        mTRTCLiveRoom.responseJoinAnchor(userInfo.userId, true, "agreed to co-anchor")
    }

    @Override
    public void onAnchorEnter(final String userId) {
        // The anchor receives a notification that the co-anchoring audience member has
        TXCloudVideoView view = new TXCloudVideoView(context);
        parentView.add(view);
        // The anchor plays the audience member's video
        mTRTCLiveRoom.startPlay(userId, view, null);
    }
});
```

4. Anchors from different rooms communicate with each other by calling [TRTCLiveRoom#requestRoomPK](#).



```
// Create room 12345
mLiveRoom.createRoom(12345, "roomA", "Your coverUrl");
// Create room 54321
mLiveRoom.createRoom(54321, "roomB", "Your coverUrl");

// Anchor A:
TRTCLiveRoom mTRTCLiveRoom=TRTCLiveRoom.sharedInstance(mContext);

// 1. Send a cross-room communication request to anchor B
mTRTCLiveRoom.requestRoomPK(54321, "B",
    new TRTCLiveRoomCallback.ActionCallback() {
```

```
@Override
public void onCallback(int code, String msg) {
    // 5. Receive a callback of whether the request is accepted by anchor B
    if (code == 0) {
        // Accepted
    } else {
        // Declined
    }
}

});

mTRTCLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
    @Override
    public void onAnchorEnter(final String userId) {
        // 6. Receive a notification about anchor B's entry
        mTRTCLiveRoom.startPlay(userId, mTXCloudVideoView, null);
    }
});

// Anchor B:
// 2. Receive anchor A's request
mTRTCLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
    @Override
    public void onRequestRoomPK(
        final TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, final int timeout) {
        // 3. Accept anchor A's request
        mTRTCLiveRoom.responseRoomPK(userInfo.userId, true, "");
    }
    @Override
    public void onAnchorEnter(final String userId) {
        // 4. Receive a notification about anchor A's entry and play anchor A's video
        mTRTCLiveRoom.startPlay(userId, mTXCloudVideoView, null);
    }
});
```

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [colleenyu@tencent.com](mailto:colleenyu@tencent.com).

# Integrating TUILiveRoom (iOS)

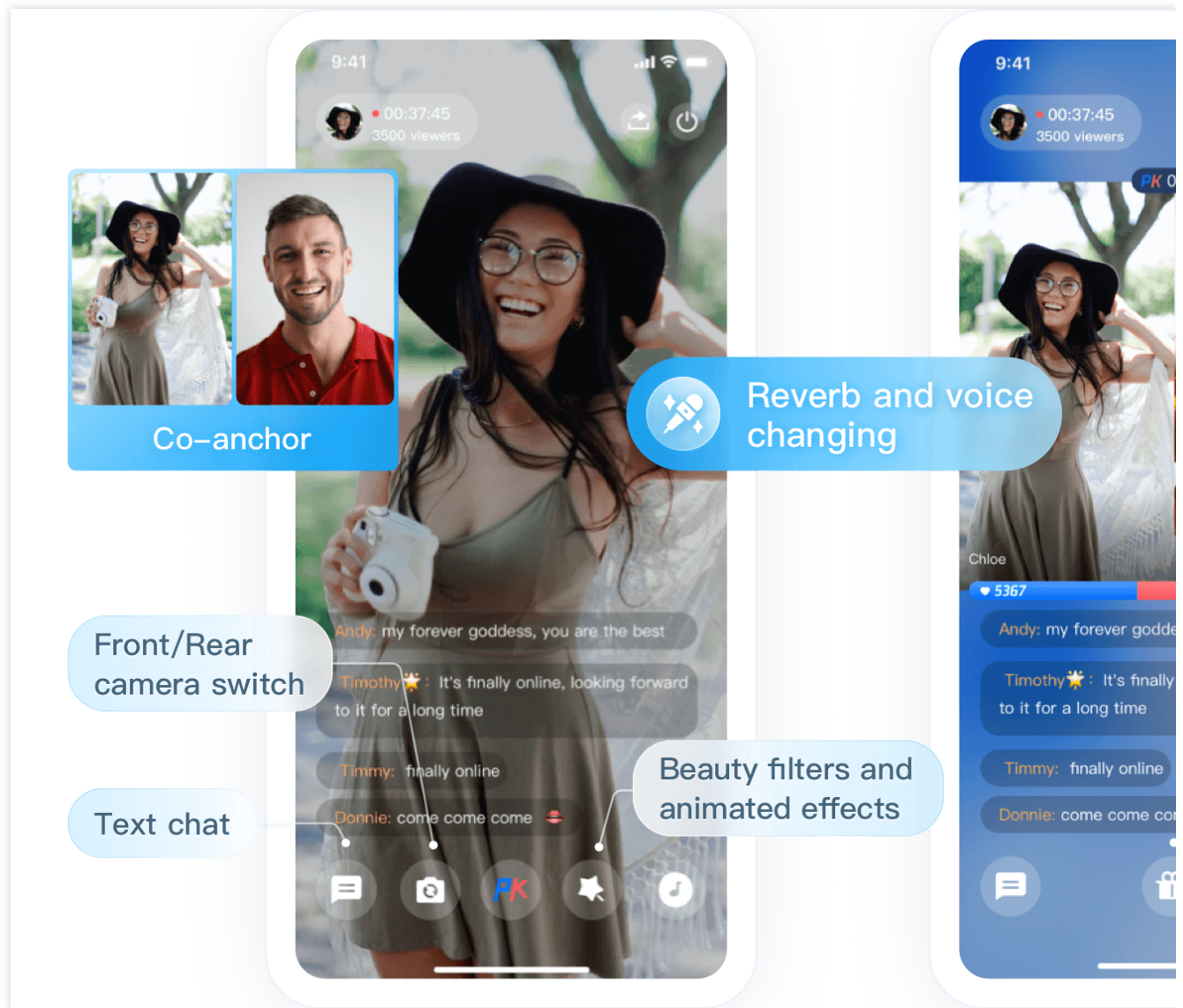
Last updated : 2023-11-20 17:51:26

## Overview

`TUILiveRoom` is an open-source video live streaming scenario UI component. After integrating it into your project, you can enable your application to support interactive video live streaming simply by writing a few lines of code. It provides source code for Android, iOS, and mini program platforms. Its basic features are as shown below:

### Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see [Pricing](#).

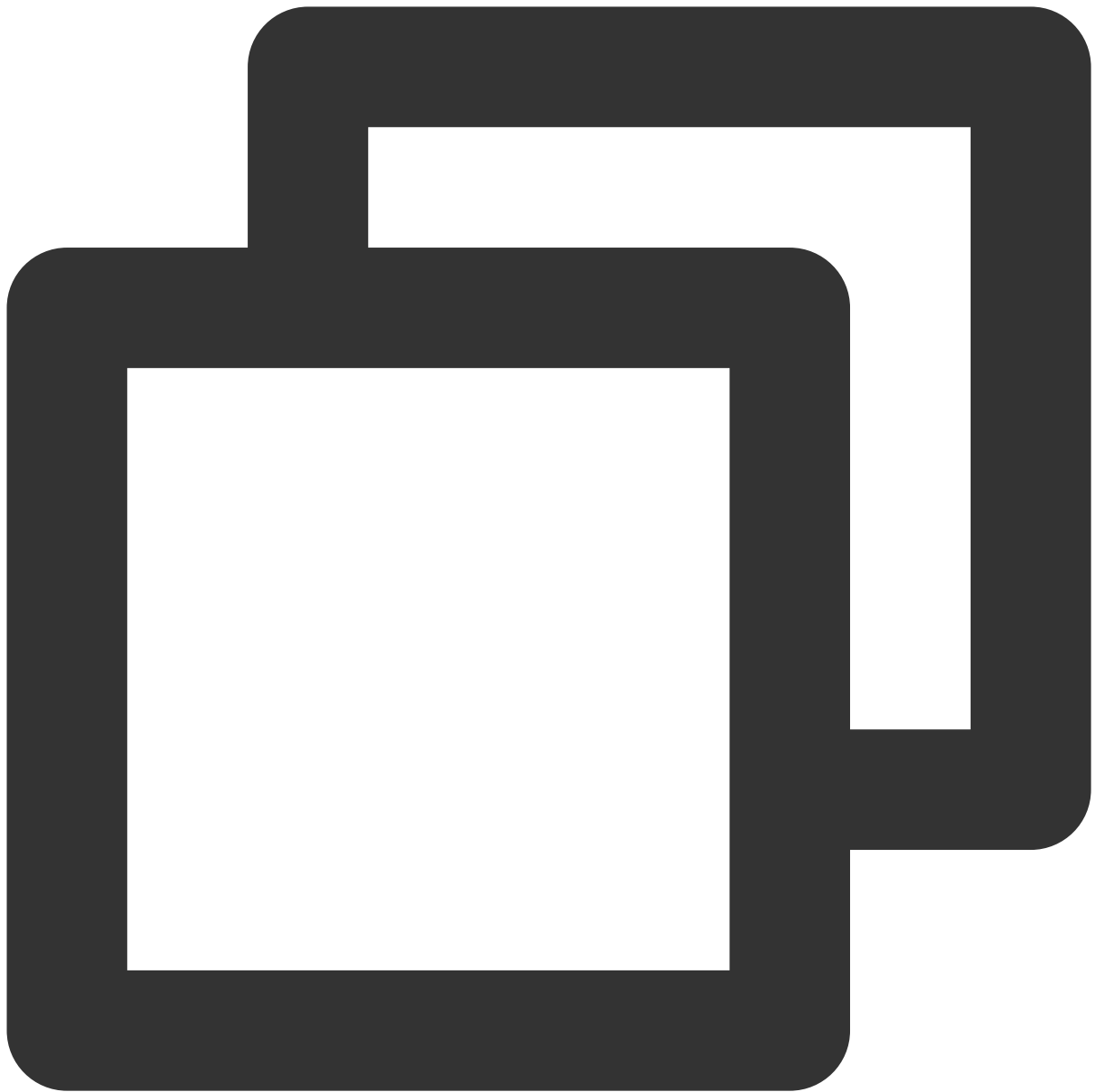


## Integration

### Step 1. Import the `TUILiveRoom` component

To import the component using **CocoaPods**, follow the steps below:

1. Create a `TUILiveRoom` folder in the same directory as `Podfile` in your project.
2. Go to the component's [GitHub page](#), clone or download the code, and copy the `Source`, `Resources`, `TUIBeauty`, `TUIAudioEffect`, `TUIBarrage`, `TUIGift`, and `TUIKitCommon` folders and the `TUILiveRoom.podspec` file in `TUILiveRoom/iOS/` to the `TUILiveRoom` folder in your project.
3. Add the following dependencies to your `Podfile` and run `pod install` to import the component.



```
# :path => "The relative path of `TUILiveRoom.podspec`"
pod 'TUILiveRoom', :path => "./TUILiveRoom/TUILiveRoom.podspec", :subspecs => ["TRT
# :path => "The relative path of `TUIKitCommon.podspec`"
pod 'TUIKitCommon', :path => "./TUILiveRoom/TUIKitCommon/"
# :path => "The relative path of `TUIBeauty.podspec`"
pod 'TUIBeauty', :path => "./TUILiveRoom/TUIBeauty/"
# :path => "The relative path of `TUIAudioEffect.podspec`"
pod 'TUIAudioEffect', :path => "./TUILiveRoom/TUIAudioEffect/"
# :path => "The relative path of `TUIBarrage.podspec`"
pod 'TUIBarrage', :path => "./TUILiveRoom/TUIBarrage/"
# :path => "The relative path of `TUIGift.podspec`"
```

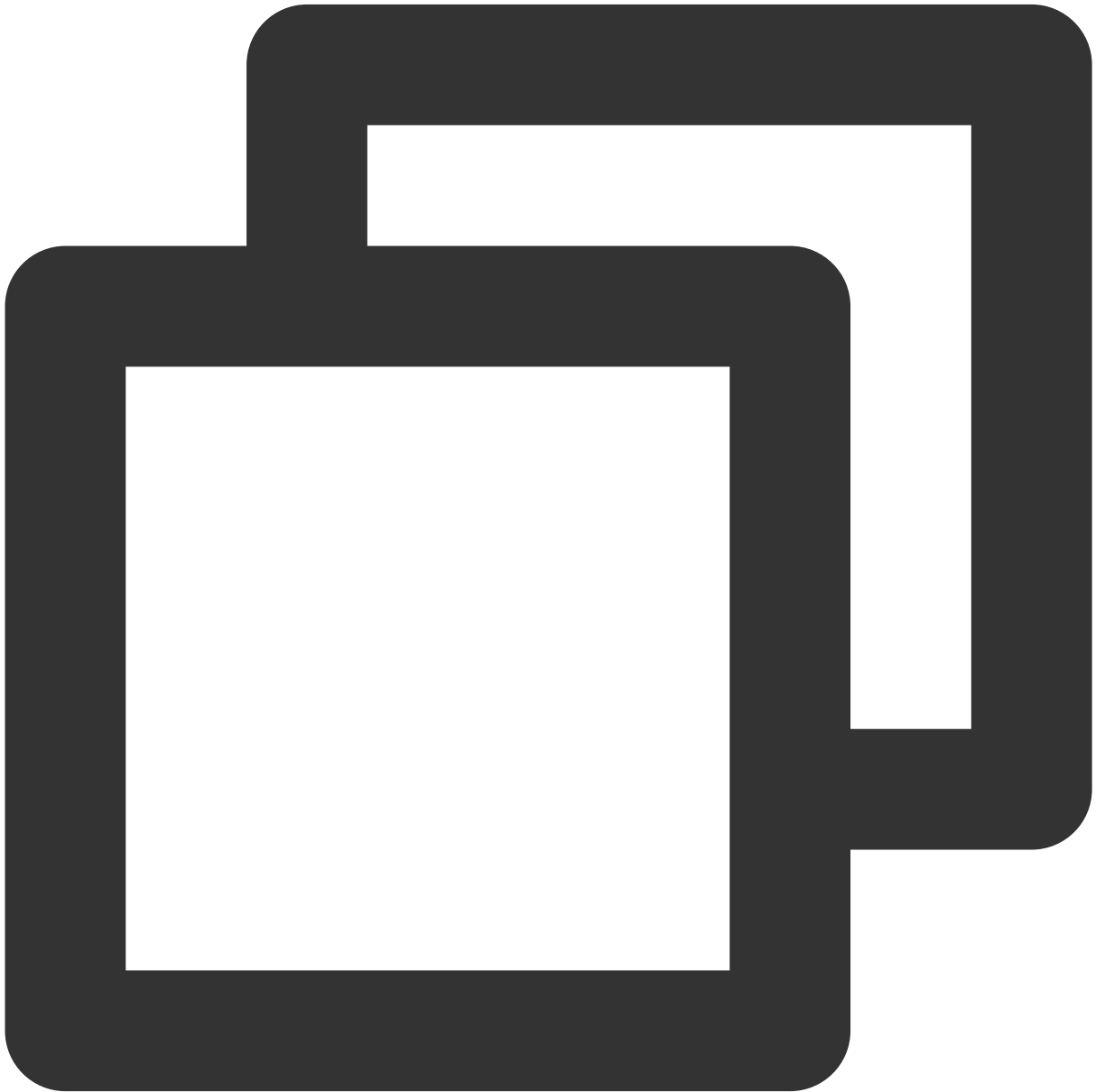
```
pod 'TUIGift', :path => "../TUILiveRoom/TUIGift/"
```

### Note

The `Source` and `Resources` folders and the `TUILiveRoom.podspec` file must be in the same directory. `TUikitCommon.podspec` is in the `TUikitCommon` folder.

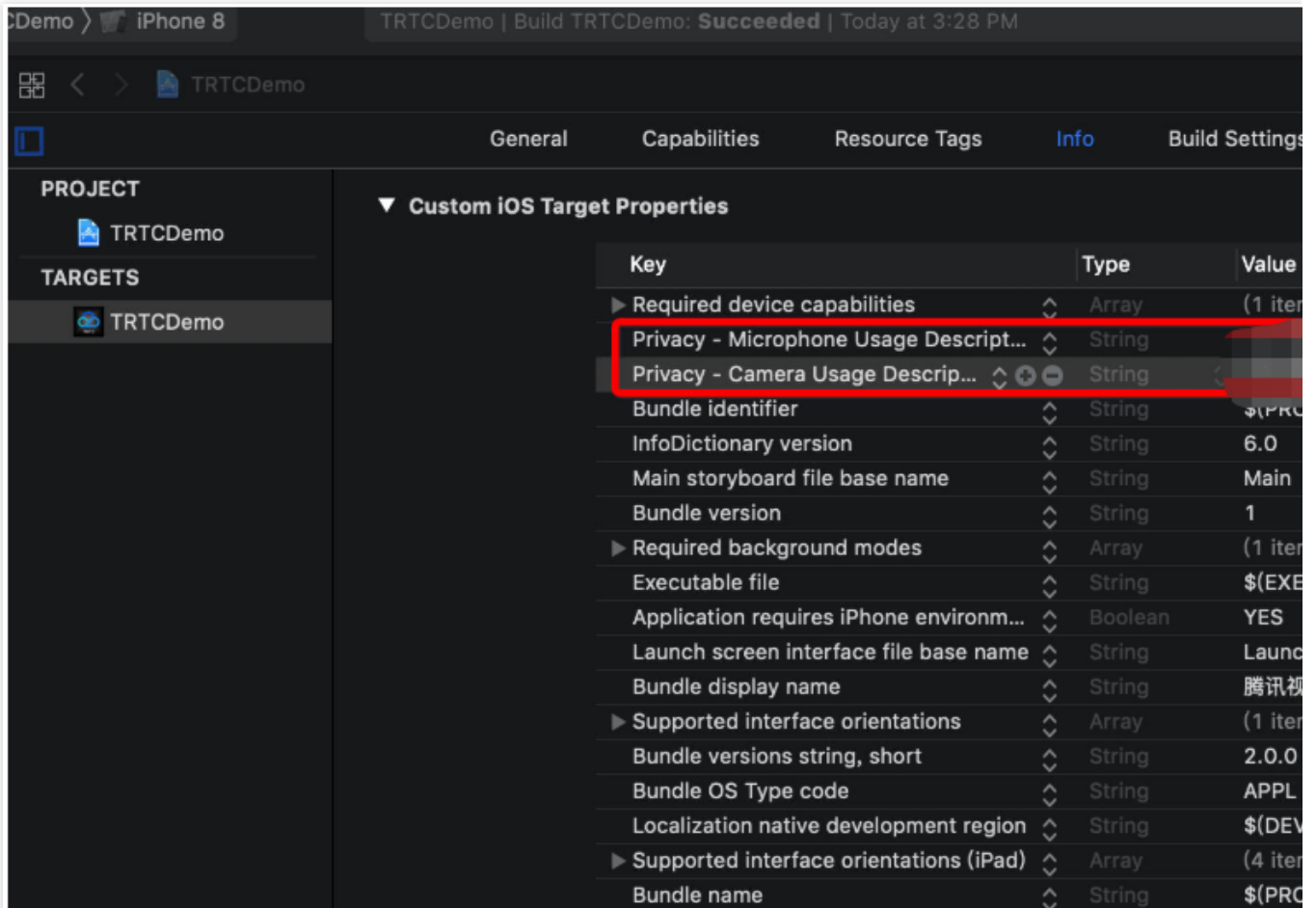
## Step 2. Configure permissions

Your app needs mic and camera permissions to implement audio/video communication. Add the two items below to `Info.plist` of your app. Their content is what users see in the mic and camera access pop-up windows.





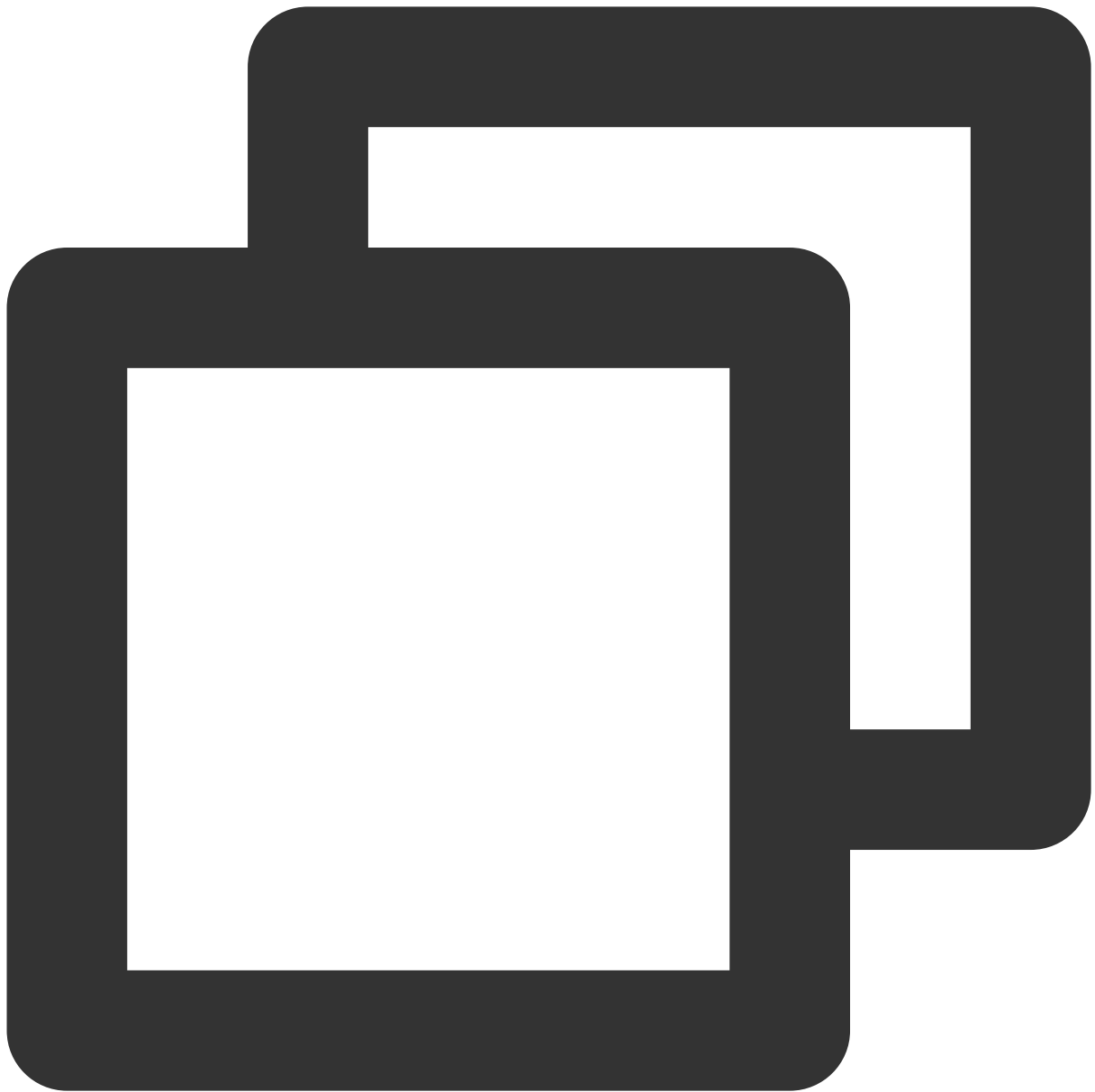
```
<key>NSCameraUsageDescription</key>
<string>RoomApp needs to access your camera to capture video.</string>
<key>NSMicrophoneUsageDescription</key>
<string>RoomApp needs to access your mic to capture audio.</string>
```



### Step 3. Initialize and log in to the component

Objective-C

Swift



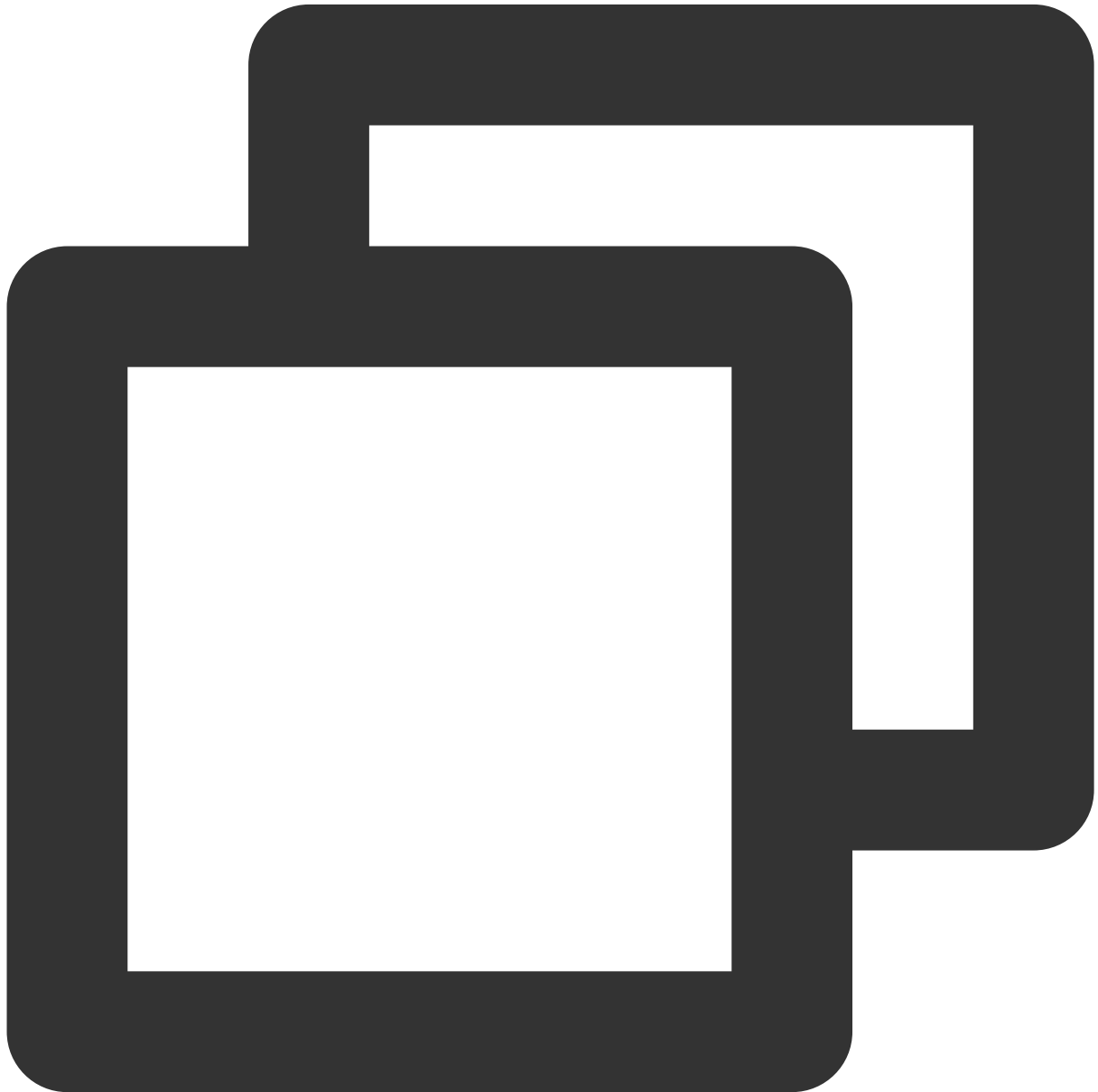
```
@import TUILiveRoom;
#import TUICore;

// 1. Log in to the component
[TUILogin login:@"Your SDKAppID" userID:@"Your UserID" userSig:@"Your UserSig" succ

} fail:^(int code, NSString *msg) {

}];
// 2. Initialize the `TUILiveRoom` instance
TUILiveRoom *mLiveRoom = [TUILiveRoom sharedInstance];
```

...



```
import TUILiveRoom
import TUICore

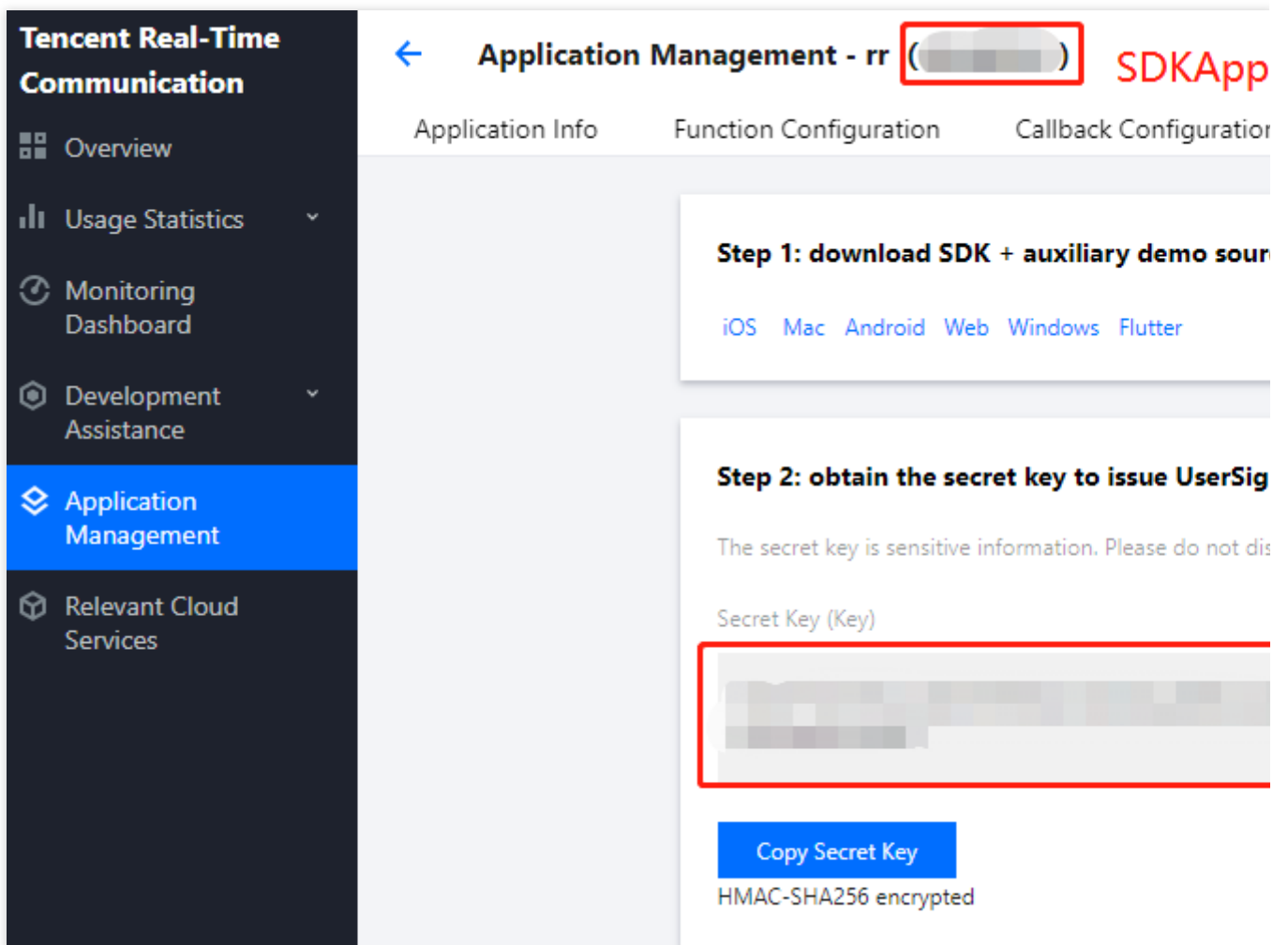
// 1. Log in to the component
TUILogin.login("Your SDKAppID", userID: "Your UserID", userSig: "Your UserSig") {

} fail: { code, msg in
```

```
}  
// 2. Initialize the `TUILiveRoom` instance  
let mLiveRoom = TUILiveRoom.sharedInstance  
```
```

#### Parameter description:

**SDKAppID:** TRTC application ID. If you haven't activated TRTC, log in to the [TRTC console](#), create a TRTC application, click **Application Info**, and select the **Quick Start** tab to view its `SDKAppID`.



The screenshot shows the Tencent Real-Time Communication console interface. On the left is a dark sidebar with navigation options: Overview, Usage Statistics, Monitoring Dashboard, Development Assistance, Application Management (highlighted in blue), and Relevant Cloud Services. The main content area is titled 'Application Management - rr (SDKAppID)' with a red box around the SDKAppID placeholder. Below the title are tabs for 'Application Info', 'Function Configuration', and 'Callback Configuration'. The 'Application Info' tab is active, showing two steps: 'Step 1: download SDK + auxiliary demo source' with links for iOS, Mac, Android, Web, Windows, and Flutter; and 'Step 2: obtain the secret key to issue UserSig'. Under Step 2, there is a warning 'The secret key is sensitive information. Please do not disclose it.' followed by a 'Secret Key (Key)' label and a text input field containing a blurred secret key, which is highlighted with a red box. Below the field is a blue 'Copy Secret Key' button and the text 'HMAC-SHA256 encrypted'.

**Secretkey:** TRTC application key. Each secret key corresponds to an `SDKAppID`. You can view your application's secret key on the [Application Management](#) page of the TRTC console.

**UserId:** Current user ID, which is a custom string that can contain up to 32 bytes of letters and digits (special characters are not supported).

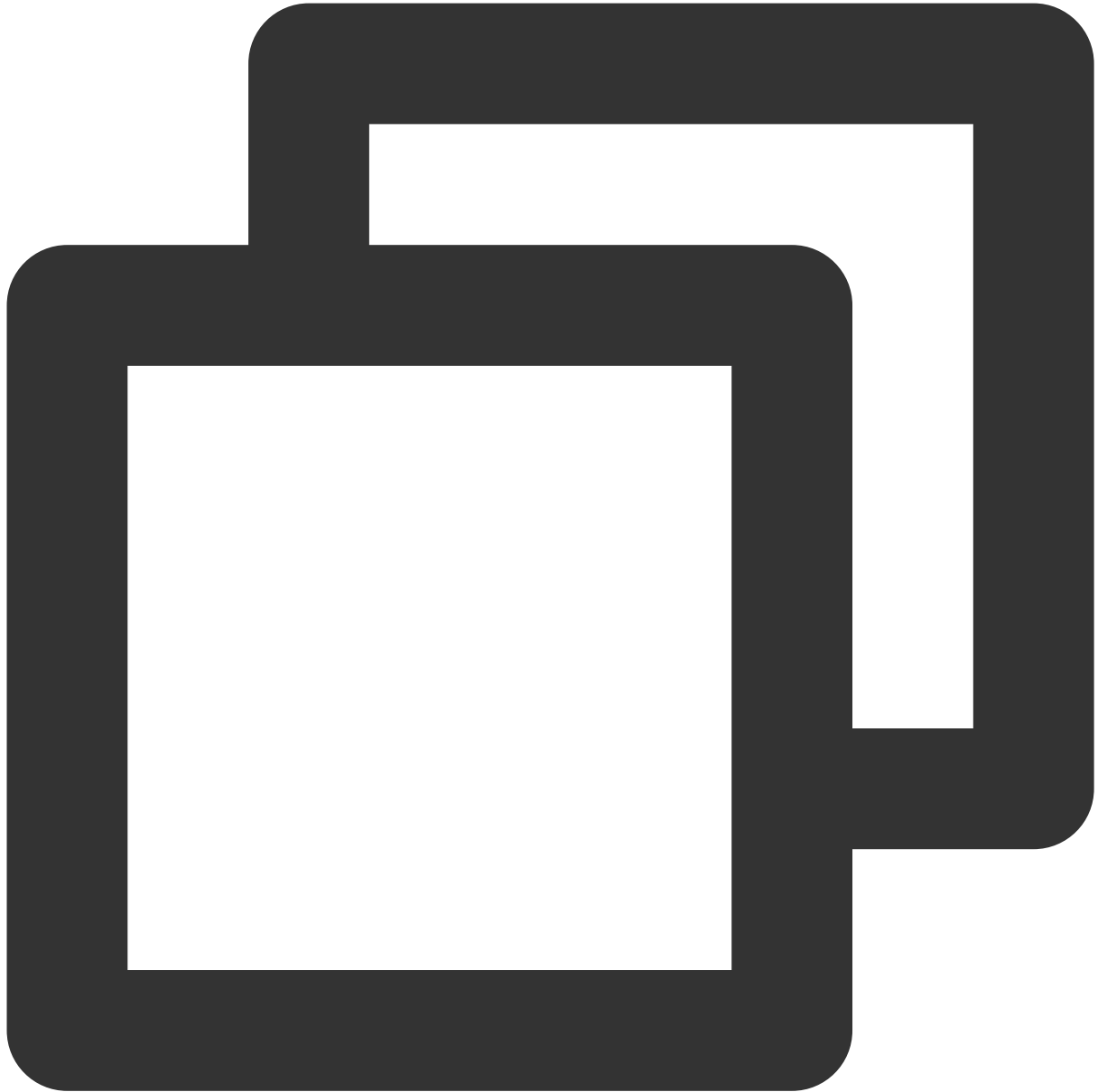
**UserSig:** Security signature calculated based on `SDKAppID`, `userId`, and `Secretkey`. You can click [here](#) to quickly generate a `UserSig` for testing or calculate it on your own by referring to our [TUILiveRoom demo project](#). For more information, see [UserSig](#).

## Step 4. Implement an interactive video live room

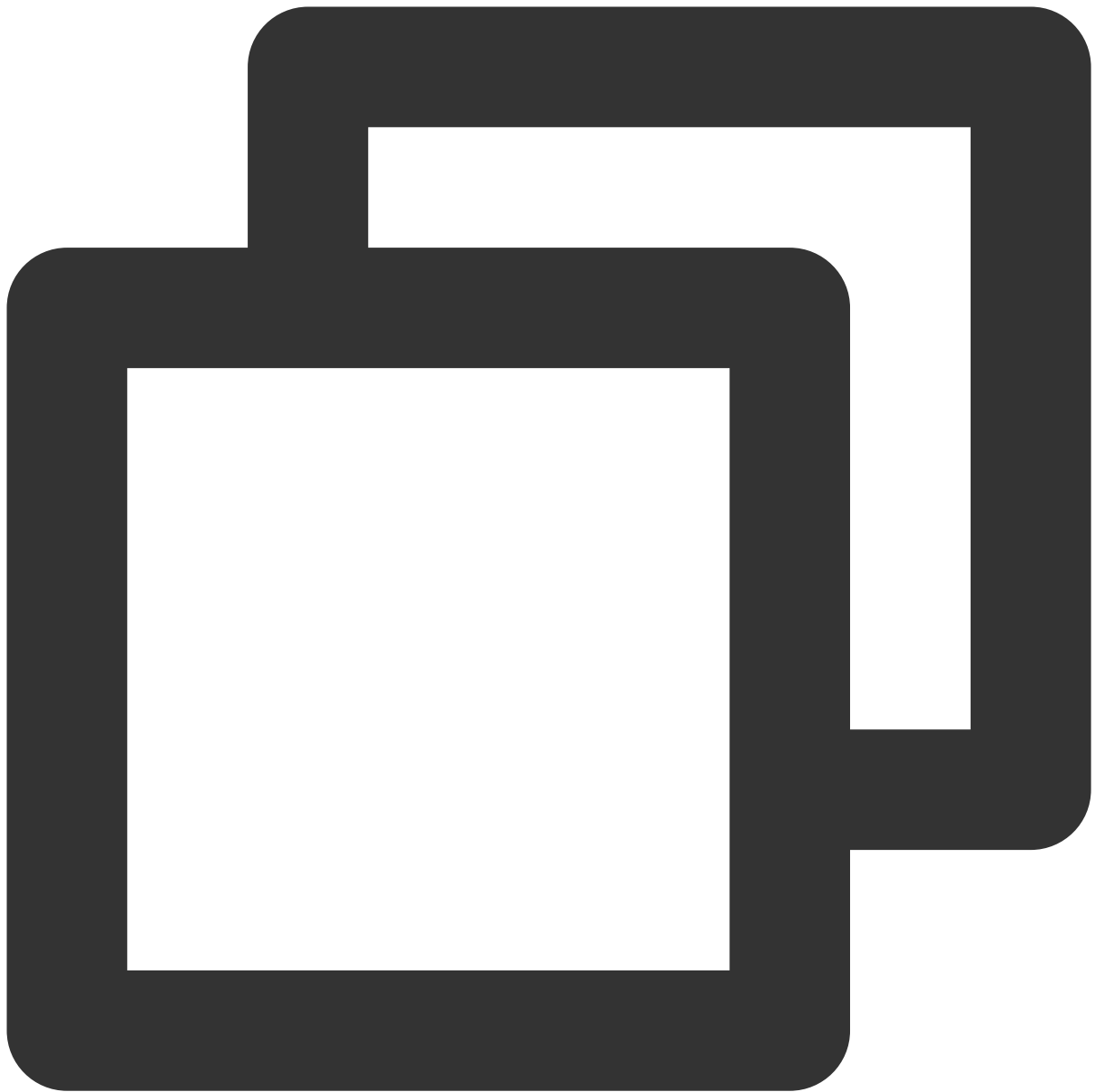
### 1. The anchor starts streaming.

Objective-C

Swift



```
[mLiveRoom createRoomWithRoomId:123 roomName:@"test room" coverUrl:@""];
```

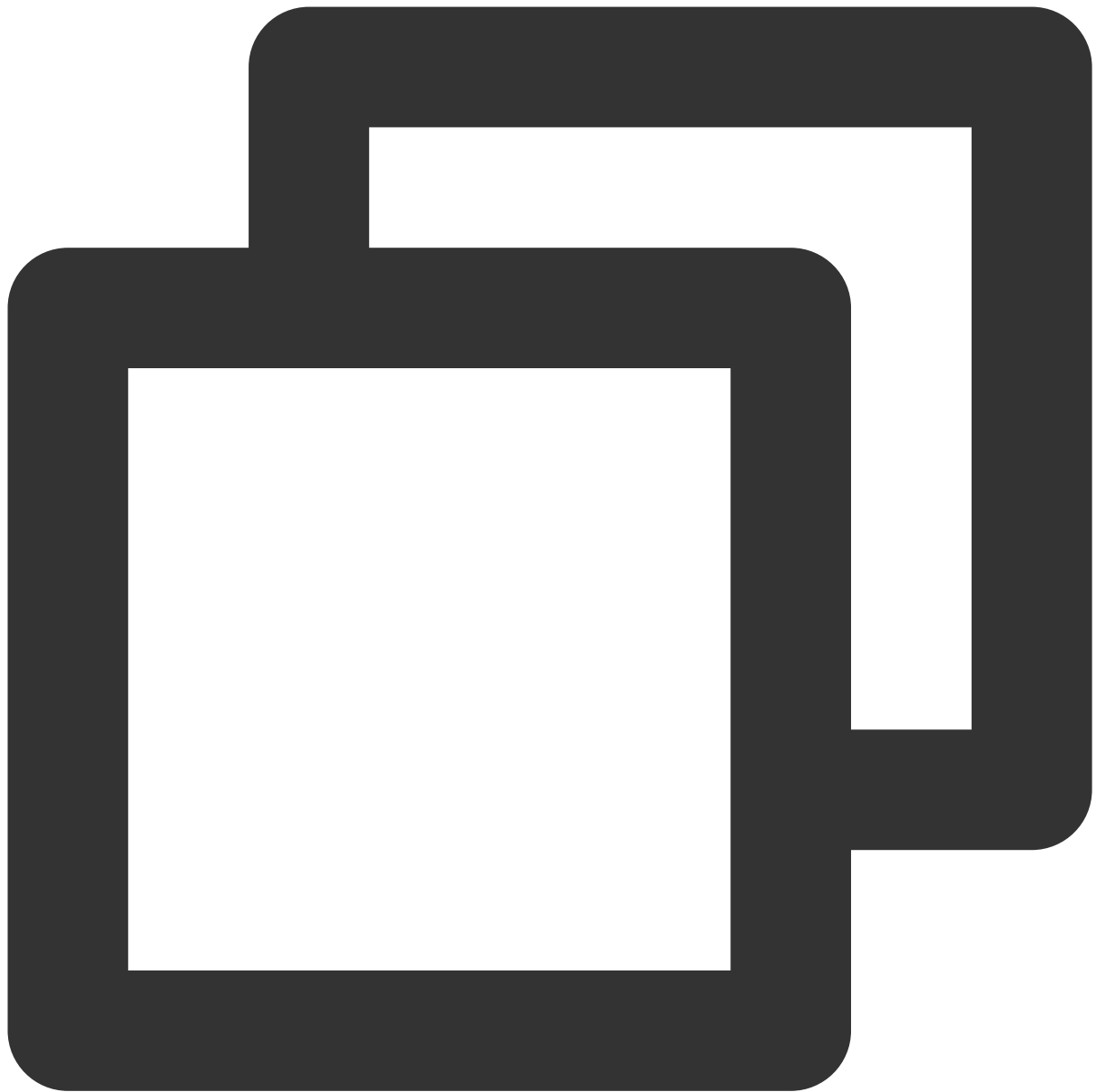


```
mLiveRoom.createRoom(roomId: 123, roomName: "test room", coverUrl:"")
```

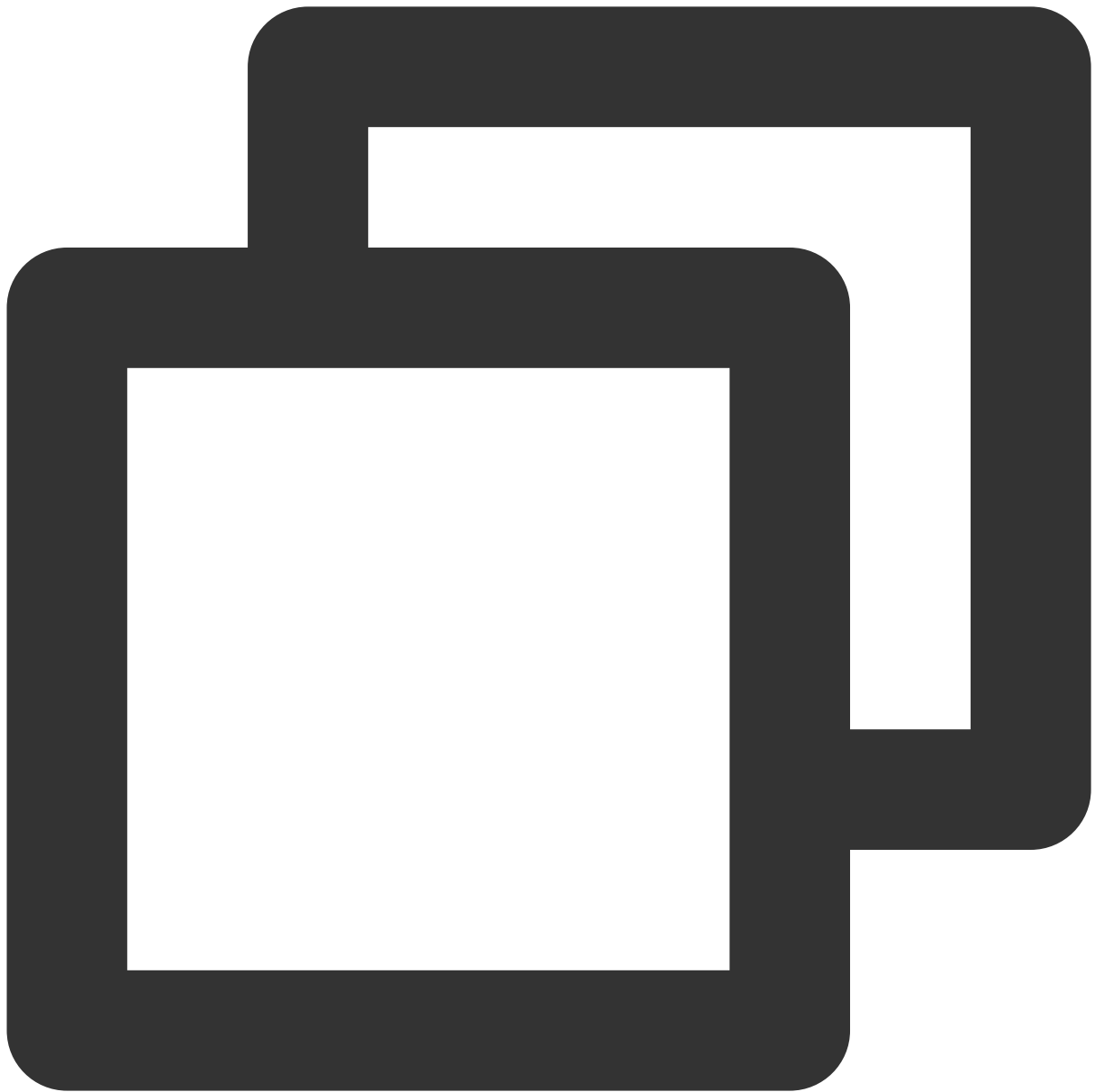
## 2. The audience member watches.

Objective-C

Swift



```
[mLiveRoom enterRoomWithRoomId:123];
```



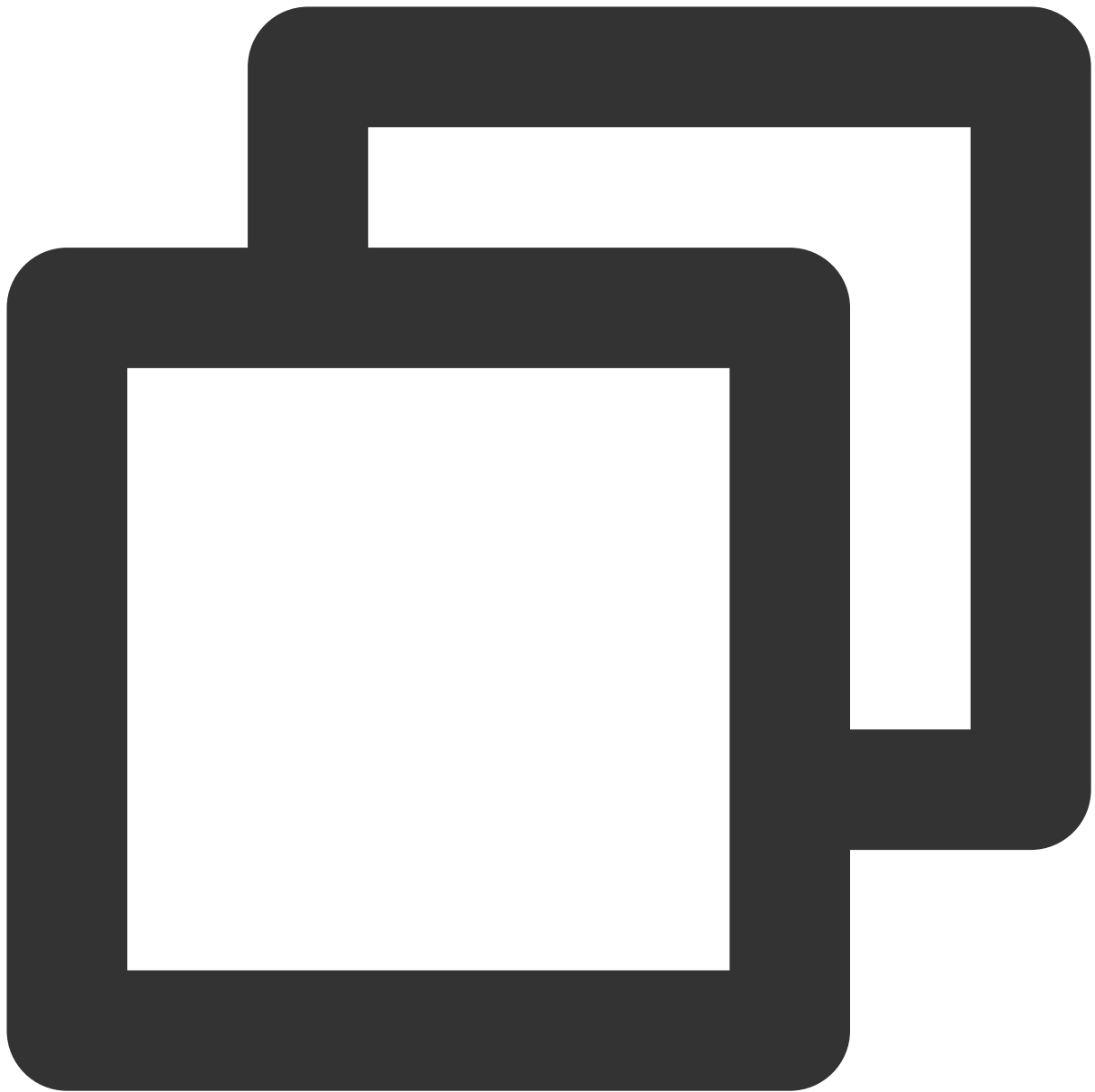
```
mLiveRoom.createRoom(roomId: 123)
```

**3. The audience member and the anchor co-anchor together through [TRTCLiveRoom#requestJoinAnchor](#).**

Objective-C

Swift



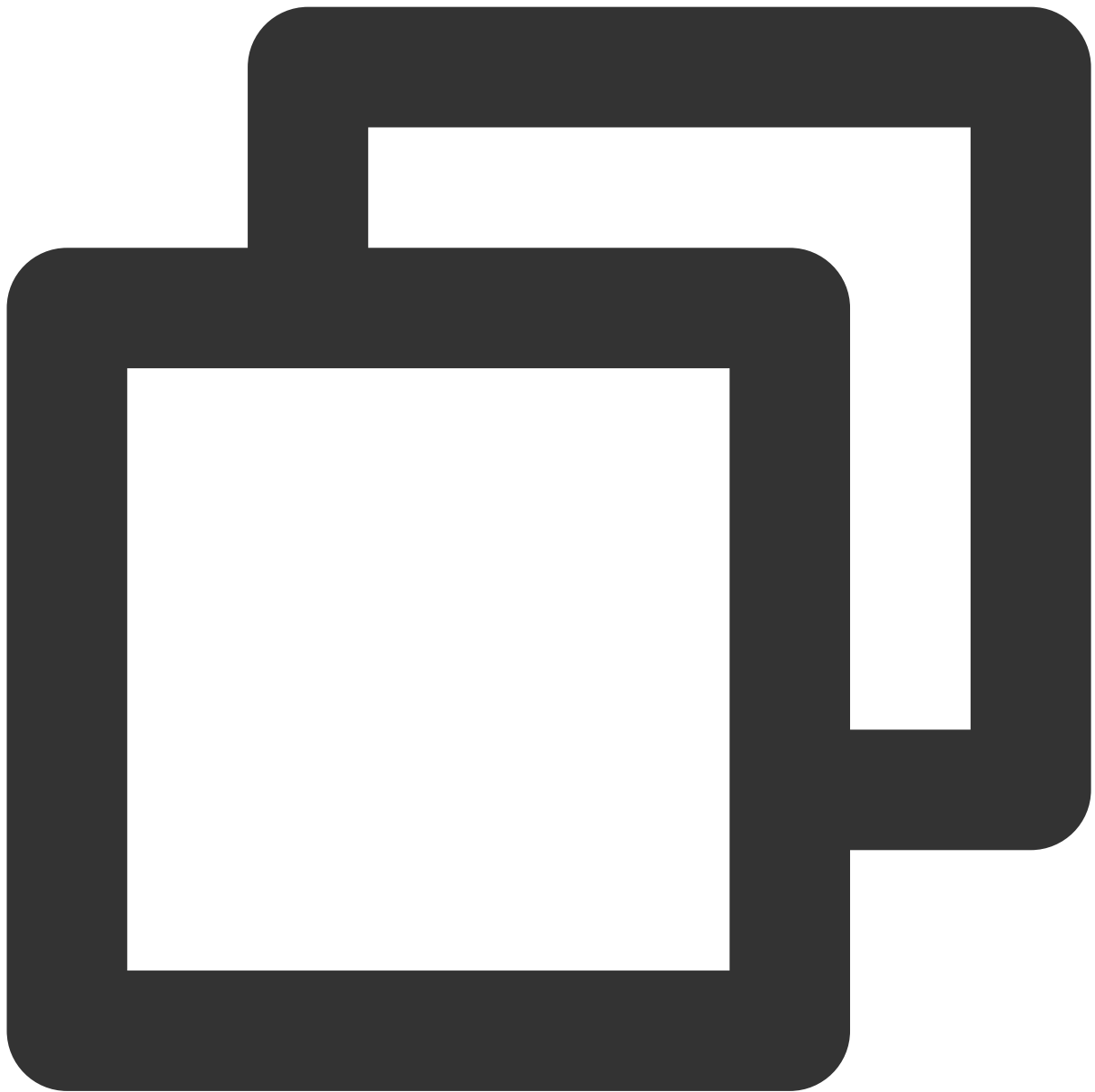


```
// 1. The audience member sends a co-anchoring request
[TRTCLiveRoom sharedInstance].delegate = self;
// @param mSelfUserId String Current user ID
NSString *mSelfUserId = @"1314";
[[TRTCLiveRoom sharedInstance] requestJoinAnchor:[NSString stringWithFormat:@"%@" req
if (agreed) {
    // The request is accepted by the anchor
    UIView *playView = [UIView new];
    [self.view addSubview:playView];
    // The audience member turns on the camera and starts pushing streams
    [[TRTCLiveRoom sharedInstance] startCameraPreviewWithFrontCamera:YES view:play
```

```
        [[TRTCLiveRoom sharedInstance] startPublishWithStreamID:[NSString stringWithFormatFo
    }
}];

// 2. The anchor receives the co-anchoring request
#pragma mark - TRTCLiveRoomDelegate
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onRequestJoinAnchor:(TRTCLiveUser
    // The anchor accepts the co-anchoring request
    [[TRTCLiveRoom sharedInstance] responseJoinAnchor:user.userId agree:YES reason:@
}

- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onAnchorEnter:(NSString *)userID
    // The anchor receives a notification that the co-anchoring audience member has
    UIView *playView = [UIView new];
    [self.view addSubview:playView];
    // The anchor plays the audience member's video
    [[TRTCLiveRoom sharedInstance] startPlayWithUserID:userID view:playView callback
}
```



```
// 1. The audience member sends a co-anchoring request
TRTCLiveRoom.sharedInstance().delegate = self
let mSelfUserId = "1314"
TRTCLiveRoom.sharedInstance().requestJoinAnchor(reason: mSelfUserId + "requested to
guard let self = self else { return }
if agree {
    // The request is accepted by the anchor
    let playView = UIView()
    self.view.addSubview(playView)
    // The audience member turns on the camera and starts pushing streams
    TRTCLiveRoom.sharedInstance().startCameraPreview(frontCamera: true, view: pl
```

```
        TRTCLiveRoom.sharedInstance().startPublish(streamID: mSelfUserId + "_stream"
    }
}

// 2. The anchor receives the co-anchoring request
extension ViewController: TRTCLiveRoomDelegate {

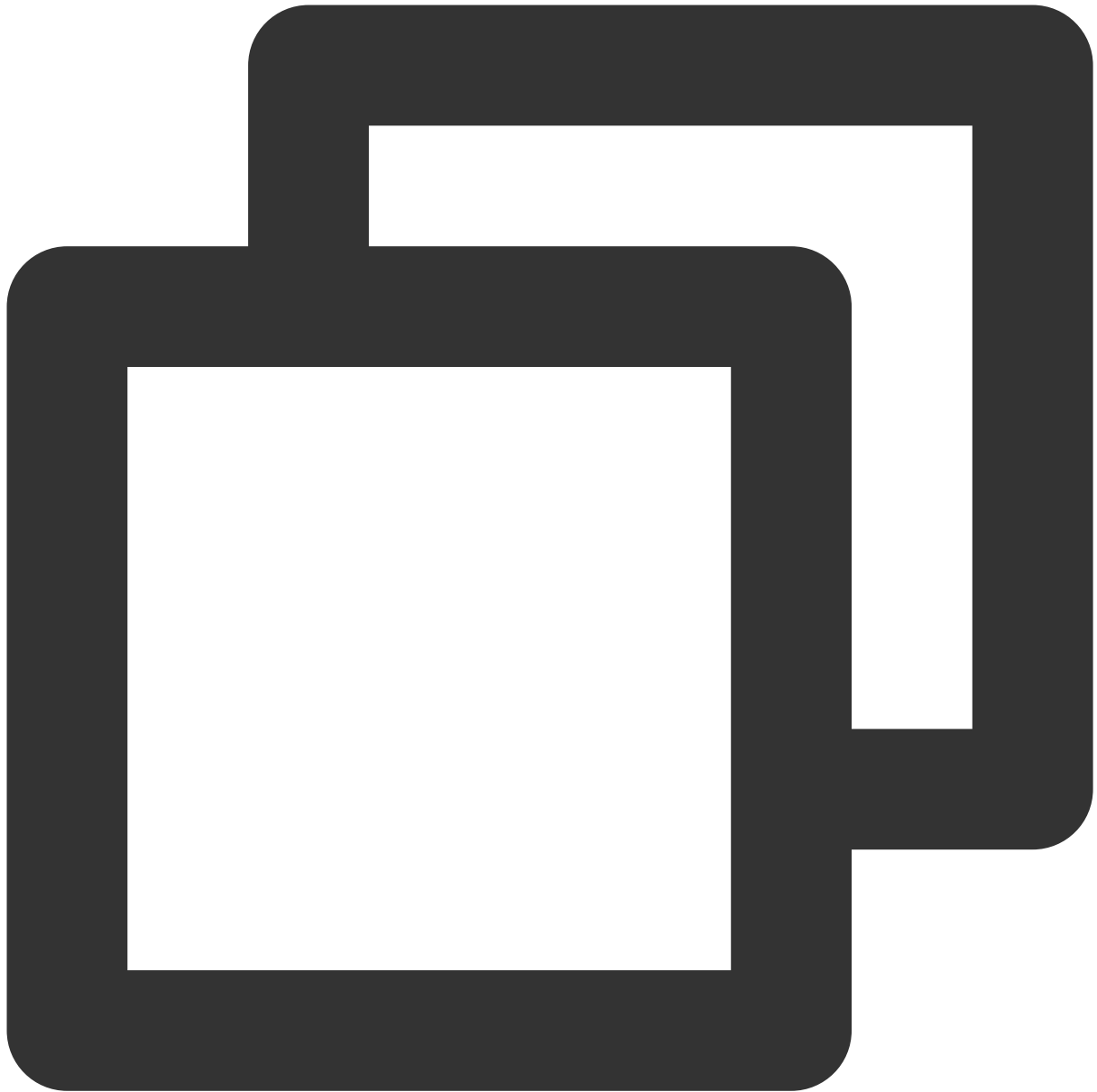
    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRequestJoinAnchor user: TRTCL
        // The anchor accepts the co-anchoring request
        TRTCLiveRoom.sharedInstance().responseRoomPK(userID: user.userId, agree: tru
    }

    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {
        // The anchor receives a notification that the co-anchoring audience member
        let playView = UIView()
        view.addSubview(playView)
        // The anchor plays the audience member's video
        TRTCLiveRoom.sharedInstance().startPlay(userID: userID, view: playView);
    }
}
```

#### 4. Anchors from different rooms communicate with each other by calling [TRTCLiveRoom#requestRoomPK](#).

Objective-C

Swift



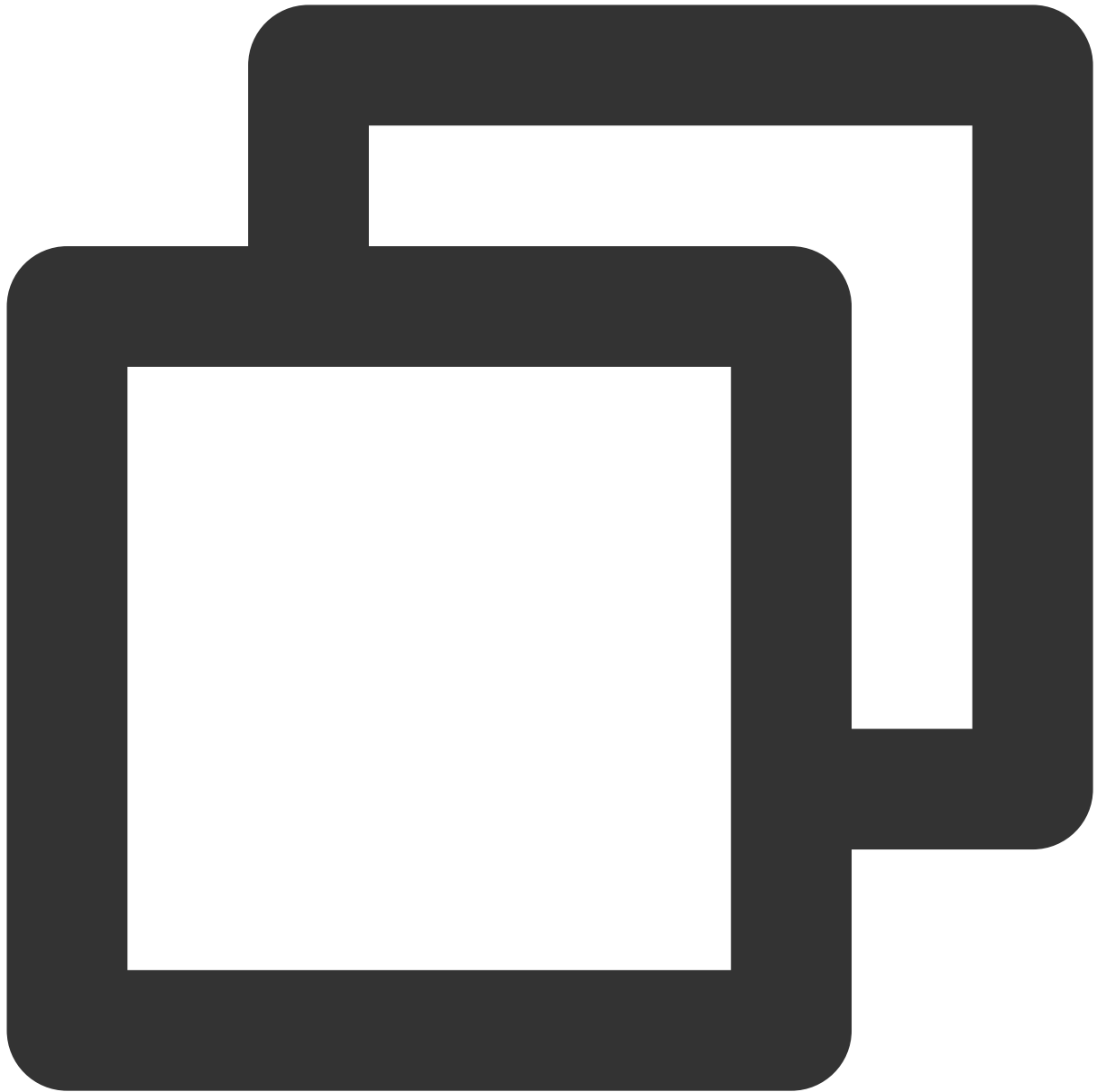
```
// Create room 12345
[[TUILiveRoom sharedInstance] createRoomWithRoomId:12345 roomName:@"roomA" coverUrl
// Create room 54321
[[TUILiveRoom sharedInstance] createRoomWithRoomId:54321 roomName:@"roomB" coverUrl

// Host A
// Send a cross-room communication request to anchor B
[[TRTCLiveRoom sharedInstance] requestRoomPKWithRoomID:543321 userID:@"roomB userId"
    if (agreed) {
        // Anchor B accepts the request
    } else {
```

```
        // Anchor B rejects the request
    }
}];

// Anchor B:
// 2. Receive anchor A's request
#pragma mark - TRTCLiveRoomDelegate
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onRequestRoomPK:(TRTCLiveUserInfo *)userInfo {
    // 3. Accept anchor A's request
    [[TRTCLiveRoom sharedInstance] responseRoomPKWithUserID:user.userId agree:YES reason:nil];
}

- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onAnchorEnter:(NSString *)userID {
    // 4. Receive a notification about anchor A's entry and play anchor A's video
    [[TRTCLiveRoom sharedInstance] startPlayWithUserID:userID view:playAVView callback:nil];
}
```



```
// Create room 12345
TUILiveRoom.sharedInstance.createRoom(roomId: 12345, roomName: "roomA")
// Create room 54321
TUILiveRoom.sharedInstance.createRoom(roomId: 54321, roomName: "roomB")

// Host A
// Send a cross-room communication request to anchor B
TRTCLiveRoom.shareInstance().requestRoomPK(roomID: 543321, userID: "roomB userId",
    guard let self = self else { return }
    if agreed {
        // Anchor B accepts the request
```

```
    } else {  
        // Anchor B rejects the request  
    }  
}  
  
// Anchor B:  
// 2. Receive anchor A's request  
extension ViewController: TRTCLiveRoomDelegate {  
    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRequestRoomPK user: TRTCLiveU  
        // 3. Accept anchor A's request  
        TRTCLiveRoom.sharedInstance().responseRoomPK(userID: user.userId, agree: tru  
    }  
  
    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {  
        // 4. Receive a notification about anchor A's entry and play anchor A's vid  
        TRTCLiveRoom.sharedInstance().startPlay(userID: userID, view: playAView);  
    }  
}
```

## Suggestions and Feedback

If you have any suggestions or feedback, please contact [colleenyu@tencent.com](mailto:colleenyu@tencent.com).



# Integrating TUIPusher and TUIPlayer (Web)

Last updated : 2023-09-25 10:56:13

## Overview

`TUIPusher` and `TUIPlayer` are our web-based open-source components for interactive live streaming (UI included). You can use them together with our basic SDKs such as [TRTC](#) and [Chat](#) to quickly equip your live streaming applications (corporate live streaming, live shopping, vocational training, remote teaching, etc.) with web-based publishing and playback capabilities.

### Note

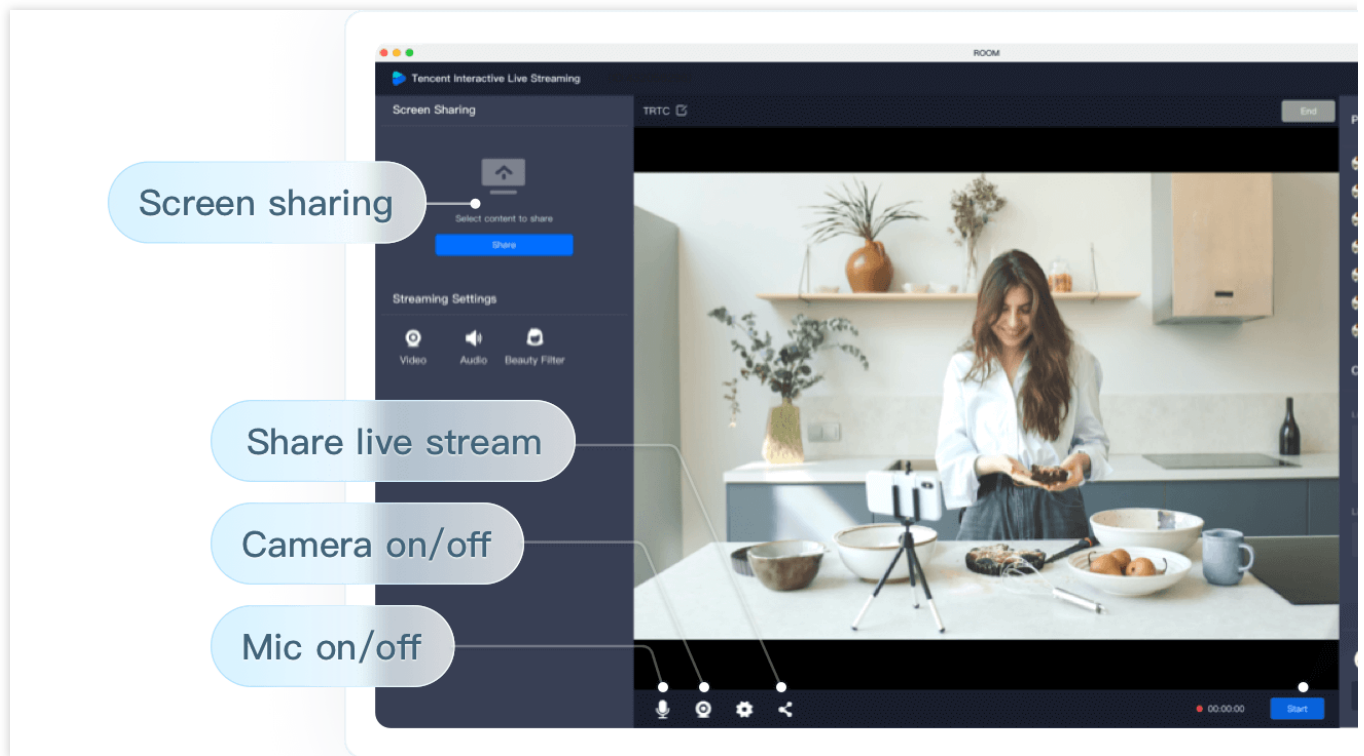
All components of TUIKit use two basic PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports only 100 DAUs) will be activated automatically. For the billing details of Chat, see [Pricing](#).

### Strengths:

A general-purpose live streaming solution with UI that includes common live streaming features such as device selection, beauty filters, publishing, playback, and live chat, helping you quickly bring your services to the market

Easy integration into Tencent Cloud's basic SDKs, including TRTC, Chat, and Superplayer, for excellent flexibility and scalability

Web-based, easy-to-use, and quick updates



## Demos

We provide a [TUIPusher Demo](#) and a [TUIPlayer Demo](#), with user and room management systems, for you to experiment with the features of the components.

### Note

You need to log in with two different accounts to try `TUIPusher` and `TUIPlayer` at the same time.

### `TUIPusher` features

- Capturing and publishing streams from camera and mic
- Customizing video parameters including frame rate, resolution, and bitrate
- Applying beauty filters and setting beauty filter parameters
- Capturing and publishing data from the screen
- Publishing to the TRTC backend and Tencent Cloud's CDNs
- Text chatting with the anchor and other audience members
- Getting the audience list and muting audience members

### `TUIPlayer` features

- Playing the audio/video stream and screen sharing stream at the same time
- Text chatting with the anchor and other audience members

Three playback options: Ultra-low-latency live streaming (300 ms latency), high-speed live streaming (latency within 1,000 ms), and standard live streaming (ultra-high concurrency)

Supports desktop and mobile browsers and landscape mode on mobile devices

#### Note

If your browser does not support WebRTC and can play videos only using standard live streaming protocols, please use a different browser to try WebRTC playback.

## Integration

### Step 1. Create an application

#### Note

`TUIPusher` and `TUIPlayer` are based on TRTC and Chat. Make sure you use the same `SDKAppID` for your TRTC and Chat applications so that they can share your account and authentication information.

You can use the basic content filtering capability of Chat to filter text messages. If you want to customize restricted words, go to the Chat console > **Content Filtering**, and click **Upgrade**.

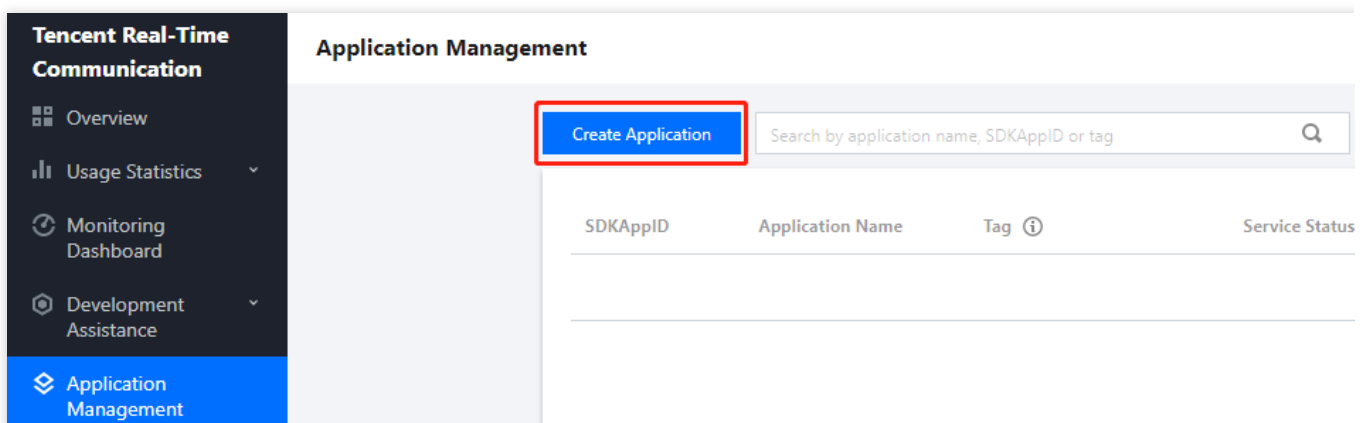
Local `UserSig` calculation is for development and local debugging only and not for official launch. The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to your server for a dynamic `UserSig`. For more information, see [Generating UserSig](#).

Method 1: Via TRTC

Method 2: Via Chat

### Step 1. Create a TRTC application




1. [Sign up for a Tencent Cloud account](#) and activate [TRTC](#) and [IM](#).
2. In the [TRTC console](#), click **Application Management** > **Create Application** to create an application.



## Step 2. Get the TRTC key information

1. In the application list, find the application created and click **Application Info** to view the SDKAppID .

**Application Info** [Edit](#)

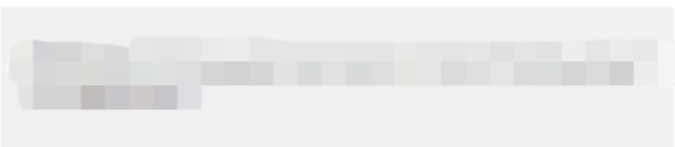
|                  |                                                                                                                                                                                |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Application Name | test_01                                                                                                                                                                        |
| SDKAppID         | 1400616927   |
| SDKSecretKey     | *****                                                                                         |
| Creation Time    | 2021-12-28 11:06:01                                                                                                                                                            |
| Description      | No description. To provide a description, click "Edit".                                                                                                                        |

2. Select the **Quick Start** tab to view the application's secret key.

**Step 2: obtain the secret key to issue UserSig**

The secret key is sensitive information. Please do not disclose it.

Secret Key (Key)



[Copy Secret Key](#)

\* The current mode is "HMAC-SHA256", you can switch to "asymmetric encryption".

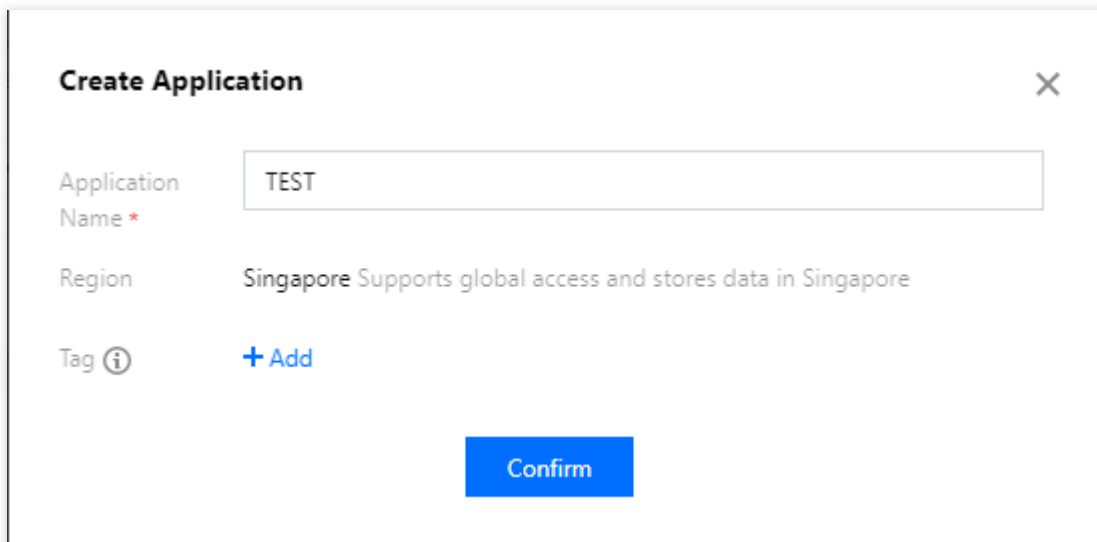
### explain

Accounts creating their first application in the TRTC console will get a 10,000-minute free trial package.

After you create a TRTC application, a Chat application with the same SDKAppID will be created automatically. You can configure package information for the application in the [Chat console](#).

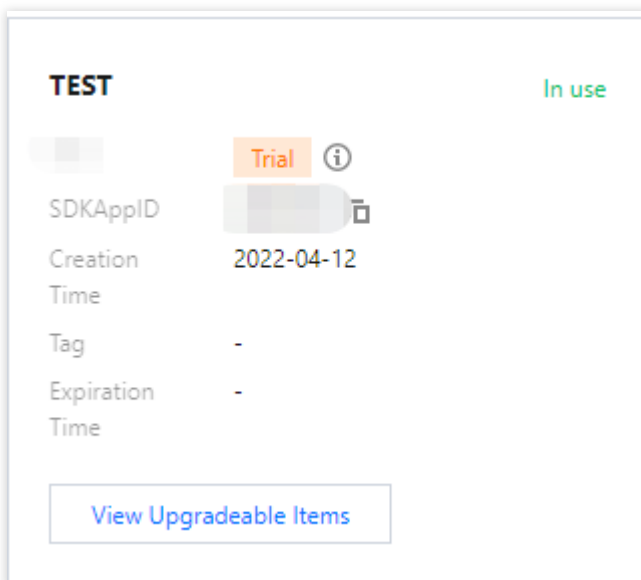
## Step 1. Create a Chat application

1. Log in to the [Chat console](#), and click **Create Application**.



The 'Create Application' dialog box features a title bar with a close button (X). It contains three input fields: 'Application Name' with the value 'TEST', 'Region' with the value 'Singapore Supports global access and stores data in Singapore', and 'Tag' with a '+ Add' button. An information icon (i) is next to the Tag label. A blue 'Confirm' button is at the bottom center.

2. In the pop-up window, enter an application name and click **Confirm**.



The application details pop-up window shows the application name 'TEST' in bold, with a green 'In use' status indicator. Below the name is a 'Trial' label with an information icon (i). The 'SDKAppID' is displayed in a grey box with a copy icon. The 'Creation Time' is '2022-04-12'. The 'Tag' and 'Expiration Time' are both marked with a hyphen (-). A blue button labeled 'View Upgradeable Items' is at the bottom.

3. Go to the [overview page](#) to view the status, edition, `SDKAppID`, creation time, and expiration time of the application created. Note down the `SDKAppID`.

## Step 2. Obtain the key and activate TRTC

1. On the [overview page](#), click the application created to go to the **Basic Configuration** page. In the **Basic Information** section, click **Display key**, and copy and save the key.

**Instant Messaging**

← **Basic Configuration** Current Region: Sir

**Basic Configuration**

**Standard Billing Plan**

Status: In use

Plan: Trial

Expiration Time: -

[Upgrade](#) [More](#)

**App Information**

SDKAppID: [Masked]

Application Name: TEST

Application Type: Game

Application Introduction: -

**Basic Information**

Key: \*\*\*\*\* [Display key](#)  
Key information is sensitive. Keep it confidential

Creation Time: 2022-04-12

Last Modified: 2022-04-12

**Note**

Please store the key information properly to prevent disclosure.

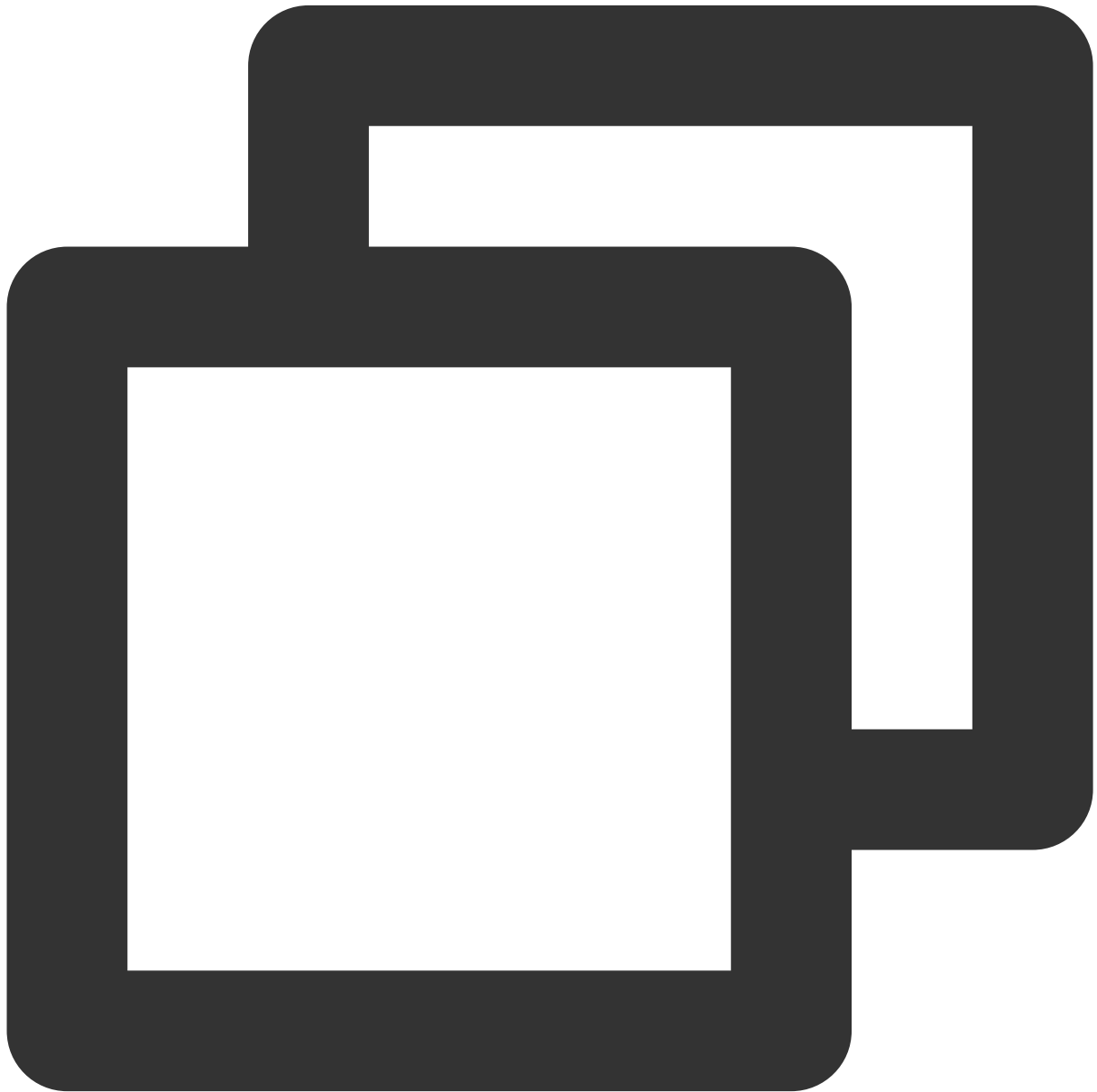
2. On the **Basic Configuration** page, activate TRTC.

**Activate Tencent Real-Time Communication (TRTC)**[Activate](#)

1. You need to activate TRTC to implement features such as voice call, video call, and interactive live streaming in the current IM application.
2. After TRTC is activated, we will create a TRTC application with the same SDKAppID as the current IM application in the [TRTC Console](#) [🔗](#) for you.
3. You must use the same SDKAppID when integrating IM SDK and TRTC SDK. Only in this case the accounts and authentication for both can be reused.

**Step 2. Prepare your project**

1. Download the code for `TUIPusher` and `TUIPlayer` at [GitHub](#).
2. Install dependencies for `TUIPusher` and `TUIPlayer`.



```
cd Web/TUIPusher
npm install
```

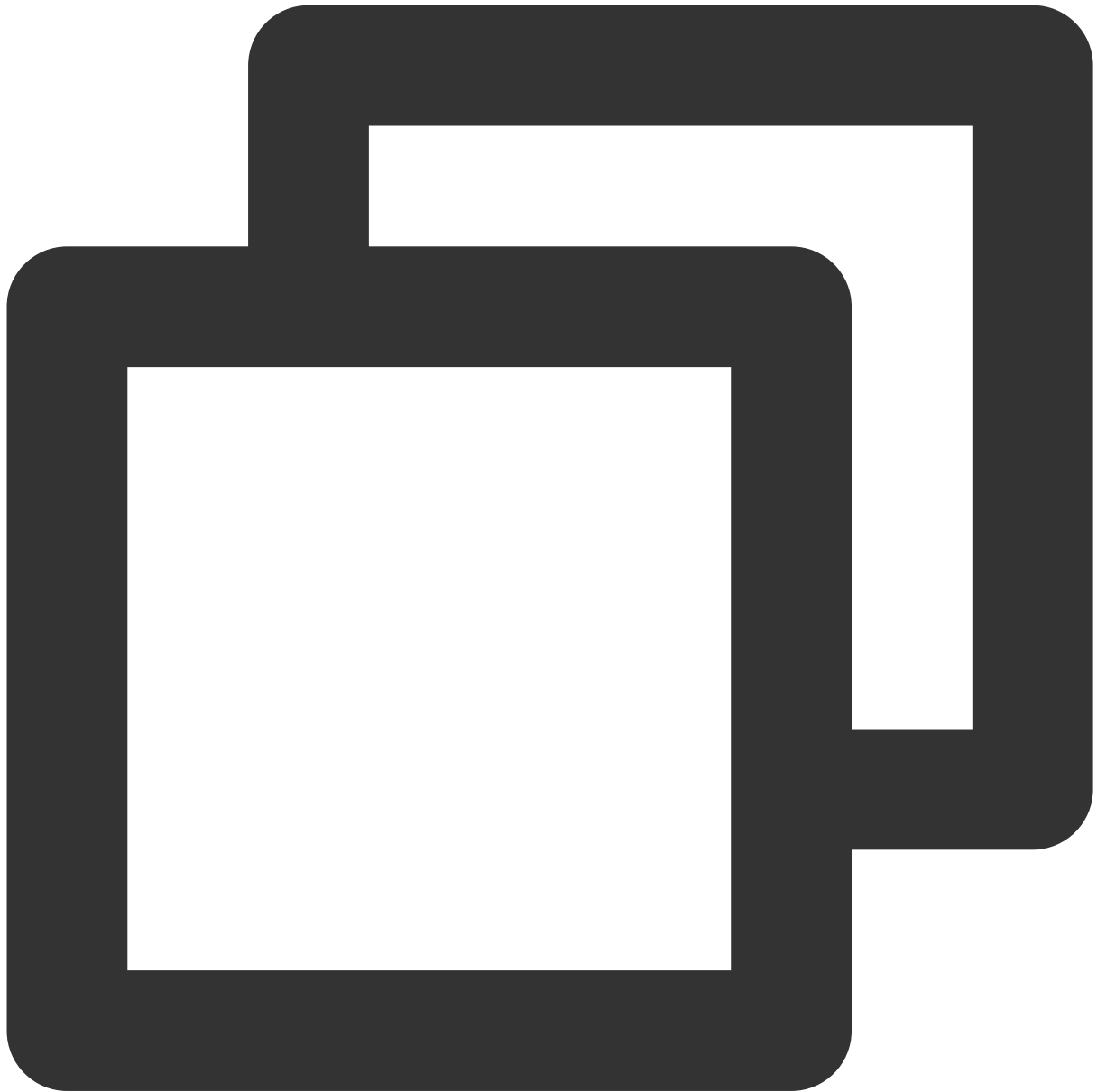
```
cd Web/TUIPlayer
npm install
```

3. Paste `SDKAppID` and the secret key to the specified locations below in the

`TUIPusher/src/config/basic-info-config.js` and `TUIPlayer/src/config/basic-info-config.js` files.

4. Run `TUIPusher` and `TUIPlayer` in a local development environment.





```
cd Web/TUIPusher
npm run serve
```

```
cd Web/TUIPlayer
npm run serve
```

5. You can open `http://localhost:8080` and `http://localhost:8081` to try out the features of `TUIPusher` and `TUIPlayer` .

6. You can modify the room, anchor, and audience information in `TUIPusher/src/config/basic-info-config.js` and `TUIPlayer/src/config/basic-info-config.js` , but **make sure the room and**

anchor information is consistent in the two files.

#### Note

You can now use `TUIPusher` and `TUIPlayer` for ultra-low-latency live streaming. If you want to support high-speed and standard live streaming too, see [Step 3. Enable relay to CDN](#).

Local calculation of `UserSig` is for development and local debugging only and not for official launch. If your

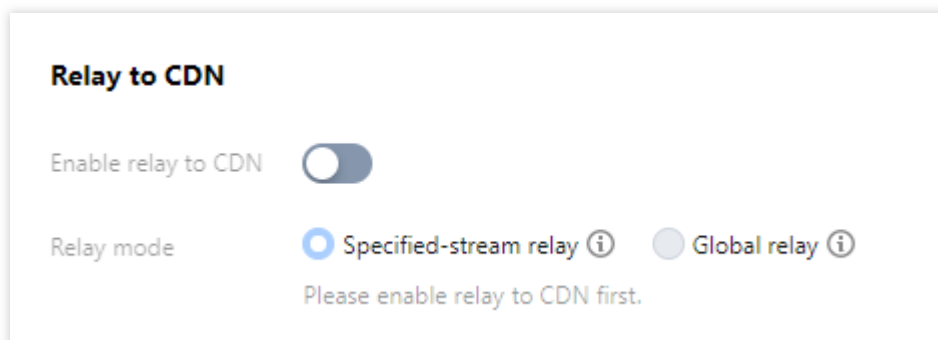
`SECRETKEY` is leaked, attackers will be able to steal your Tencent Cloud traffic.

The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to your server for a dynamic `UserSig`. For more information, see [Calculating UserSig on the server](#).

### Step 3. Enable relay to CDN

Because the high-speed and standard live streaming features of `TUIPusher` and `TUIPlayer` are powered by [CSS](#), you need to enable relay to CDN to use these features.

1. In the [TRTC console](#), enable relay to CDN for your application. You can choose **Specified-stream relay** or **Global relay** based on your needs.



2. On the [Domain Management](#) page, add your playback domain name. For detailed directions, please see [Adding Your Own Domain Names](#).

3. Configure the playback domain name in `TUIPlayer/src/config/basic-info-config.js`.

You can now use all features of `TUIPusher` and `TUIPlayer`, including ultra-low-latency live streaming, high-speed live streaming, and standard live streaming.

### Step 4. Apply in a production environment

To apply `TUIPusher` and `TUIPlayer` to a production environment, in addition to integrating them into your project, you also need to do the following:

Create a user management system to manage user information such as user IDs, usernames, and profile pictures

Create a room management system to manage room information such as room IDs, room names, and anchors

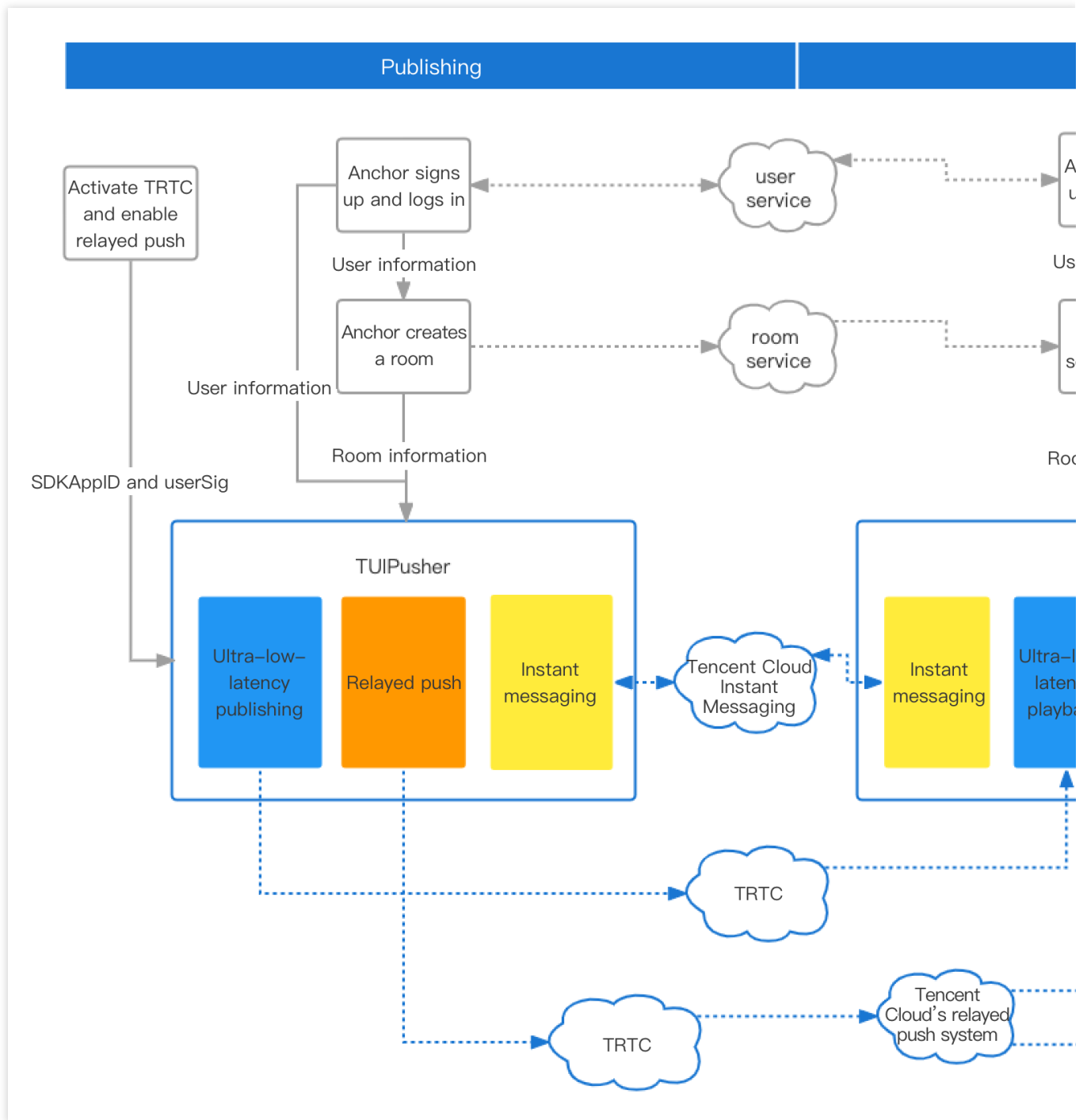
Generate `UserSig` on your server

## Note

In this document, `UserSig` is generated on the client based on the `SDKAppID` and secret key you provide. The secret key may be easily decompiled and reversed, and if your key is disclosed, attackers will be able to steal your traffic. Therefore, **this method is for local debugging only**.

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

Submit account information such as user information, room information, `SDKAppID`, and `UserSig` to `store` of `vuex` for global storage, as shown in `TUIPusher/src/pusher.vue` and `TUIPlayer/src/player.vue`. Then you will be able to use all publishing and playback features of the two components. The diagram below shows the workflow in detail:



## FAQs

### How do I use beauty filters on the web?

See [Enabling Beauty Filters](#).

### How do I implement the screen sharing feature on the web?

See [Screen Sharing](#).

## How do I implement the on-cloud recording feature on the web?

1. For information about how to enable **on-cloud recording**, see [On-Cloud Recording and Playback](#).
2. If you enable **specified user recording**, you can start recording on the web by specifying

`userDefineRecordId` when calling the [TRTC.createClient](#) API.

## How do I publish a stream to CDN on the web?

See [Publishing to CDN](#).

## How do I enable high-speed playback on the web?

Publish streams to CDNs using the TRTC web SDK and play the streams over WebRTC.

# Notes

## Supported platforms

| Operating System | Browser                  | Required Version | TUIPlayer | TUIPusher     | TUIPusher Screen Sharing   |
|------------------|--------------------------|------------------|-----------|---------------|----------------------------|
| macOS            | Safari                   | 11+              | Supported | Supported     | Supported (on Safari 13+)  |
| macOS            | Chrome                   | 56+              | Supported | Supported     | Supported (on Chrome 72+)  |
| macOS            | Firefox                  | 56+              | Supported | Supported     | Supported (on Firefox 66+) |
| macOS            | Edge                     | 80+              | Supported | Supported     | Supported                  |
| macOS            | WeChat built-in browser  | -                | Supported | Not supported | Not supported              |
| macOS            | WeCom built-in browser   | -                | Supported | Not supported | Not supported              |
| Windows          | Chrome                   | 56+              | Supported | Supported     | Supported (on Chrome 72+)  |
| Windows          | QQ Browser (WebKit core) | 10.4+            | Supported | Supported     | Not supported              |

|         |                         |     |                                          |               |                            |
|---------|-------------------------|-----|------------------------------------------|---------------|----------------------------|
| Windows | Firefox                 | 56+ | Supported                                | Supported     | Supported (on Firefox 66+) |
| Windows | Edge                    | 80+ | Supported                                | Supported     | Supported                  |
| Windows | WeChat built-in browser | -   | Supported                                | Not supported | Not supported              |
| Windows | WeCom built-in browser  | -   | Supported                                | Not supported | Not supported              |
| iOS     | WeChat built-in browser | -   | Supported                                | Not supported | Not supported              |
| iOS     | WeCom built-in browser  | -   | Supported                                | Not supported | Not supported              |
| iOS     | Safari                  | -   | Supported                                | Not supported | Not supported              |
| iOS     | Chrome                  | -   | Supported                                | Not supported | Not supported              |
| Android | WeChat built-in browser | -   | Supported                                | Not supported | Not supported              |
| Android | WeCom built-in browser  | -   | Supported                                | Not supported | Not supported              |
| Android | Chrome                  | -   | Supported                                | Not supported | Not supported              |
| Android | QQ Browser              | -   | Supported                                | Not supported | Not supported              |
| Android | Firefox                 | -   | Supported                                | Not supported | Not supported              |
| Android | UC Browser              | -   | Supported (only standard live streaming) | Not supported | Not supported              |

## Domain requirements

For security and privacy reasons, only HTTPS URLs can access all features of `TUIPusher` and `TUIPlayer`. Therefore, please use the HTTPS protocol for the web page of your application in production environments.

### Note

You can use `http://localhost` for local development.

The table below lists the supported domain names and protocols.

| Scenario          | Protocol                               | TUIPlayer | TUIPusher     | TUIPusher<br>Screen<br>Sharing | Remarks     |
|-------------------|----------------------------------------|-----------|---------------|--------------------------------|-------------|
| Production        | HTTPS                                  | Supported | Supported     | Supported                      | Recommended |
| Production        | HTTP                                   | Supported | Not supported | Not supported                  | -           |
| Local development | <code>http://localhost</code>          | Supported | Supported     | Supported                      | Recommended |
| Development       | <code>http://127.0.0.1</code>          | Supported | Supported     | Supported                      | -           |
| Local development | <code>http://[local IP address]</code> | Supported | Not supported | Not supported                  | -           |

## Firewall configuration

Firewall restrictions may cause audio/video calls to fail. To avoid this, add the ports and domains specified in [Firewall Restrictions](#) to the allowlist of your firewall.

## Summary

In future versions, we plan to add support for communication between the web components and TRTC native SDKs (such as the iOS SDK and Android SDK), as well as introduce features such as co-anchoring, advanced filters, custom layout, relaying to multiple platforms, and image/text/music upload.

If you have any requirements or feedback, contact [colleenyu@tencent.com](mailto:colleenyu@tencent.com).

# TUILiveRoom APIs

## TRTCLiveRoom APIs (iOS)

Last updated : 2023-09-25 10:56:51

`TRTCLiveRoom` is based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat. With TRTCLiveRoom:

A user can create a room and start live streaming, or enter a room as an audience member.

The anchor can co-anchor with audience members in the room.

Anchors in two rooms can compete and interact with each other.

All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

### Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see [Pricing](#).

`TRTCLiveRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see [Interactive Live Video Streaming \(iOS\)](#).

The [TRTC SDK](#) is used as a low-latency live streaming component.

The `AVChatRoom` feature of the [Chat SDK](#) is used to implement chat rooms, and Chat messages are used to facilitate the co-anchoring process.

## TRTCLiveRoom API Overview

### Basic SDK APIs

| API                            | Description           |
|--------------------------------|-----------------------|
| <a href="#">delegate</a>       | Sets event callbacks. |
| <a href="#">login</a>          | Logs in.              |
| <a href="#">logout</a>         | Logs out.             |
| <a href="#">setSelfProfile</a> | Sets the profile.     |

### Room APIs

| API | Description |
|-----|-------------|
|-----|-------------|



|                                 |                                                                                                                              |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">createRoom</a>      | Creates a room (called by anchor). If the room does not exist, the system will create the room automatically.                |
| <a href="#">destroyRoom</a>     | Terminates a room (called by anchor).                                                                                        |
| <a href="#">enterRoom</a>       | Enters a room (called by audience).                                                                                          |
| <a href="#">exitRoom</a>        | Exits a room (called by audience).                                                                                           |
| <a href="#">getRoomInfos</a>    | Gets room list details.                                                                                                      |
| <a href="#">getAnchorList</a>   | Gets the anchors and co-anchoring viewers in a room. This API works only if it is called after <code>enterRoom()</code> .    |
| <a href="#">getAudienceList</a> | Gets the information of all audience members in a room. This API works only if it is called after <code>enterRoom()</code> . |

## Stream pushing/pulling APIs

| API                                | Description                                                                                 |
|------------------------------------|---------------------------------------------------------------------------------------------|
| <a href="#">startCameraPreview</a> | Enables preview of the local video.                                                         |
| <a href="#">stopCameraPreview</a>  | Stops local video capturing and preview.                                                    |
| <a href="#">startPublish</a>       | Starts live streaming (pushing streams).                                                    |
| <a href="#">stopPublish</a>        | Stops live streaming (pushing streams).                                                     |
| <a href="#">startPlay</a>          | Plays a remote video. This API can be called in common playback and co-anchoring scenarios. |
| <a href="#">stopPlay</a>           | Stops rendering a remote video.                                                             |

## APIs for anchor-audience co-anchoring

| API                                | Description                                            |
|------------------------------------|--------------------------------------------------------|
| <a href="#">requestJoinAnchor</a>  | Requests co-anchoring (called by audience).            |
| <a href="#">responseJoinAnchor</a> | Responds to a co-anchoring request (called by anchor). |
| <a href="#">kickoutJoinAnchor</a>  | Removes a user from co-anchoring (called by anchor).   |

## APIs for cross-room communication

|  |  |
|--|--|
|  |  |
|--|--|

| API                            | Description                                                        |
|--------------------------------|--------------------------------------------------------------------|
| <a href="#">requestRoomPK</a>  | Sends a cross-room communication request (called by anchor).       |
| <a href="#">responseRoomPK</a> | Responds to a cross-room communication request (called by anchor). |
| <a href="#">quitRoomPK</a>     | Quits cross-room communication.                                    |

## Audio/Video APIs

| API                                | Description                                  |
|------------------------------------|----------------------------------------------|
| <a href="#">switchCamera</a>       | Switches between the front and rear cameras. |
| <a href="#">setMirror</a>          | Sets the mirror mode.                        |
| <a href="#">muteLocalAudio</a>     | Mutes/Unmutes the local user.                |
| <a href="#">muteRemoteAudio</a>    | Mutes/Unmutes a remote user.                 |
| <a href="#">muteAllRemoteAudio</a> | Mutes/Unmutes all remote users.              |

## Background music and audio effect APIs

| API                                   | Description                                                                                         |
|---------------------------------------|-----------------------------------------------------------------------------------------------------|
| <a href="#">getAudioEffectManager</a> | Gets the background music and audio effect management object <a href="#">TXAudioEffectManager</a> . |

## Beauty filter APIs

| API                              | Description                                                                |
|----------------------------------|----------------------------------------------------------------------------|
| <a href="#">getBeautyManager</a> | Gets the beauty filter management object <a href="#">TXBeautyManager</a> . |

## Message sending APIs

| API                               | Description                                                                             |
|-----------------------------------|-----------------------------------------------------------------------------------------|
| <a href="#">sendRoomTextMsg</a>   | Broadcasts a text message in a room. This API is generally used for on-screen comments. |
| <a href="#">sendRoomCustomMsg</a> | Sends a custom text message.                                                            |

## Debugging APIs

| API                               | Description                                                   |
|-----------------------------------|---------------------------------------------------------------|
| <a href="#">showVideoDebugLog</a> | Specifies whether to display debugging information on the UI. |

## TRTCLiveRoomDelegate API Overview

### Common event callbacks

| API                        | Description           |
|----------------------------|-----------------------|
| <a href="#">onError</a>    | Callback for error.   |
| <a href="#">onWarning</a>  | Callback for warning. |
| <a href="#">onDebugLog</a> | Callback of log.      |

### Room event callback APIs

| API                              | Description                   |
|----------------------------------|-------------------------------|
| <a href="#">onRoomDestroy</a>    | The room was terminated.      |
| <a href="#">onRoomInfoChange</a> | The room information changed. |

### Callback APIs for entry/exit of anchors/audience

| API                             | Description                                            |
|---------------------------------|--------------------------------------------------------|
| <a href="#">onAnchorEnter</a>   | There is a new anchor/co-anchoring viewer in the room. |
| <a href="#">onAnchorExit</a>    | An anchor/co-anchoring viewer quit co-anchoring.       |
| <a href="#">onAudienceEnter</a> | An audience member entered the room.                   |
| <a href="#">onAudienceExit</a>  | An audience member left the room.                      |

### Callback APIs for anchor-audience co-anchoring events

| API                                 | Description                          |
|-------------------------------------|--------------------------------------|
| <a href="#">onRequestJoinAnchor</a> | A co-anchoring request was received. |
|                                     |                                      |

[onKickoutJoinAnchor](#)

A user was removed from co-anchoring.

## Callback APIs for cross-room communication events

| API                             | Description                                      |
|---------------------------------|--------------------------------------------------|
| <a href="#">onRequestRoomPK</a> | A cross-room communication request was received. |
| <a href="#">onQuitRoomPK</a>    | Cross-room communication ended.                  |

## Message event callback APIs

| API                                 | Description                    |
|-------------------------------------|--------------------------------|
| <a href="#">onRecvRoomTextMsg</a>   | A text message was received.   |
| <a href="#">onRecvRoomCustomMsg</a> | A custom message was received. |

## Basic SDK APIs

### delegate

This API is used to get callbacks for the events of [TRTCLiveRoom](#). You can use `TRTCLiveRoomDelegate` to get the callbacks.



```
@property(nonatomic, weak) id<TRTCLiveRoomDelegate> delegate;
```

**Note**

`delegate` is the delegate callback of `TRTCLiveRoom` .

**login**

Login



```
/// Log in to the component.  
/// - Parameters:  
///   - sdkAppID: You can view `SDKAppID` in [Application Management] (https://console.cloud.tencent.com/rtc)  
///   - userID: ID of the current user, which is a string that can contain only letters and numbers  
///   - userSig: Tencent Cloud's proprietary security signature. For how to calculate it, see User Signature  
///   - config: global configuration information, which should be initialized during application initialization  
///   - callback: Callback for login. The return code is `0` if login is successful  
/// - Note:  
///   - We recommend setting the validity period of `userSig` to 7 days to avoid cache expiration  
- (void)loginWithSdkAppID:(int)sdkAppID  
  userID:(NSString *)userID
```

```

        userSig:(NSString *)userSig
        config:(TRTCLiveRoomConfig *)config
        callback:(Callback _Nullable)callback
NS_SWIFT_NAME(login(sdkAppID:userID:userSig:config:callback:));

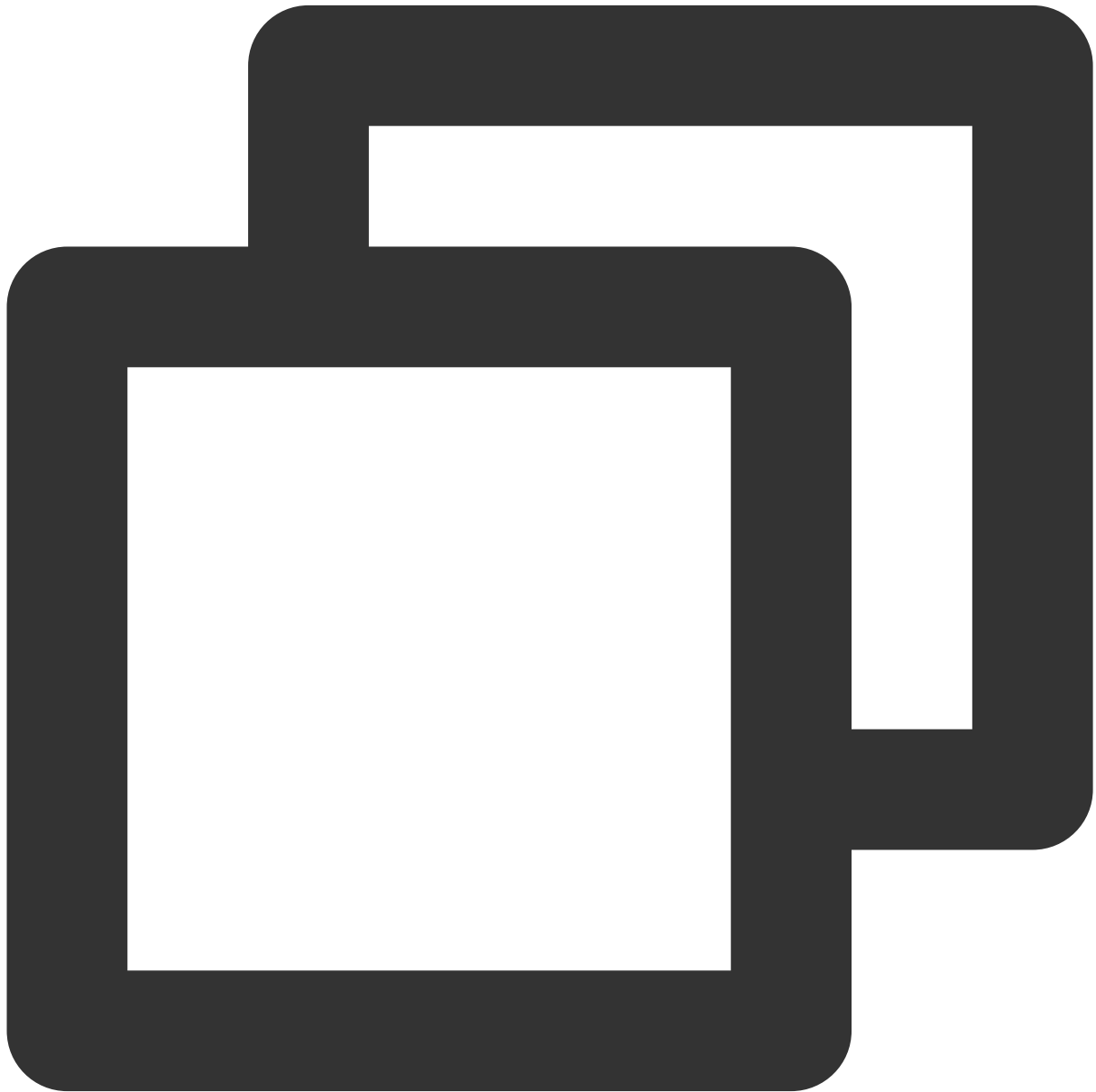
```

The parameters are described below:

| Parameter | Type                                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sdkAppId  | Int                                       | You can view <code>SDKAppID</code> in <a href="#">Application Management</a> > <b>Application Info</b> of the TRTC console.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| userID    | String                                    | The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| userSig   | String                                    | Tencent Cloud's proprietary security signature. For how to calculate and use it, see <a href="#">FAQs &gt; UserSig</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| config    | TRTCLiveRoomConfig                        | Global configuration information, which needs to be initialized during login and cannot be modified afterward.<br><code>useCDNFirst</code> : specifies the way audience watch live streams. <code>true</code> means watching over CDNs, which is cost-efficient but has high latency. <code>false</code> means watching under the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is lower than 1 second.<br><code>CDNPlayDomain</code> : specifies the domain name for CDN live streaming. It takes effect only if <code>useCDNFirst</code> is set to <code>true</code> . You can set it in <a href="#">Domain Management</a> of the CSS console. |
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for login. The code is <code>0</code> if login is successful.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

## logout

Log out



```
// Log out
/// - Parameter callback: Callback for logout. The code is `0` if logout is successful.
- (void)logout:(Callback _Nullable)callback
NS_SWIFT_NAME(logout(_:));
```

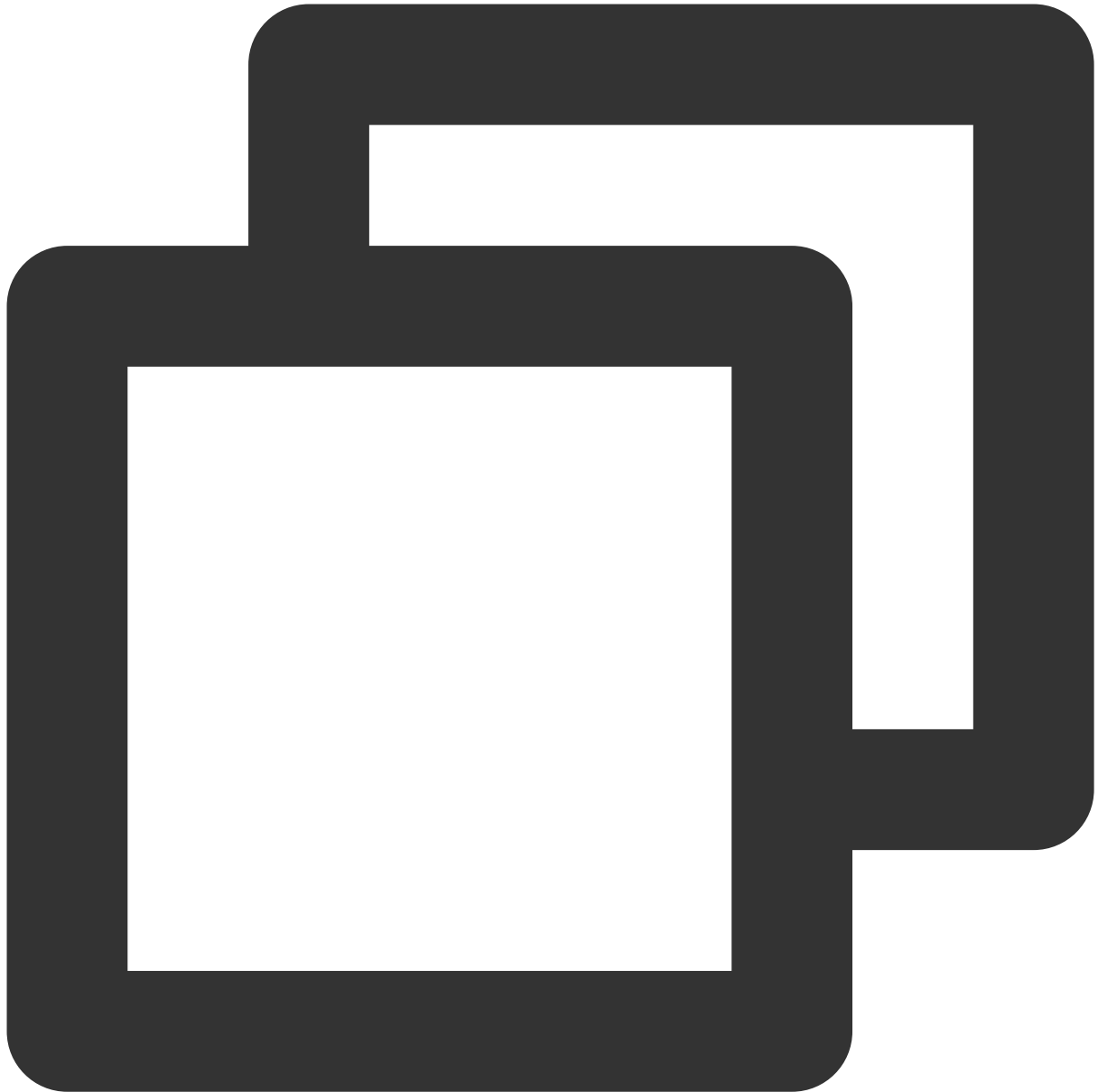
The parameters are described below:

| Parameter | Type                                      | Description                                                              |
|-----------|-------------------------------------------|--------------------------------------------------------------------------|
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for logout. The code is <code>0</code> if logout is successful. |



## setSelfProfile

This API is used to set the profile.



```
/// Set user profiles. The information will be stored in Tencent Cloud Chat.  
/// - Parameters:  
///   - name: username  
///   - avatarURL: profile picture URL  
///   - callback: Callback for setting user profiles. The code is `0` if the operat  
- (void)setSelfProfileWithName:(NSString *)name  
    avatarURL:(NSString * _Nullable)avatarURL  
    callback:(Callback _Nullable)callback
```

```
NS_SWIFT_NAME(setSelfProfile(name:avatarURL:callback:));
```

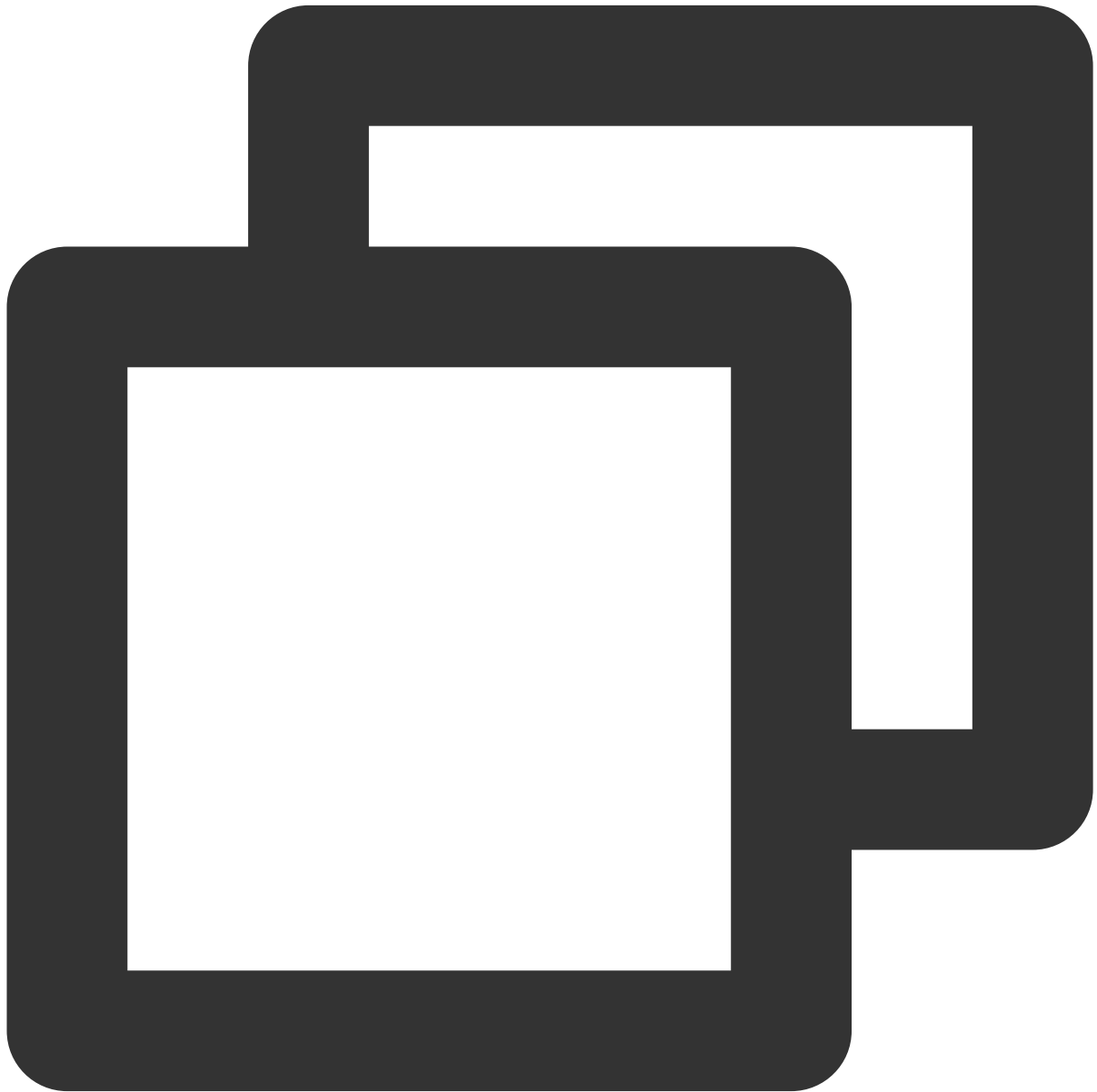
The parameters are described below:

| Parameter | Type                                      | Description                                                                       |
|-----------|-------------------------------------------|-----------------------------------------------------------------------------------|
| name      | String                                    | Username                                                                          |
| avatarURL | String                                    | Profile picture URL                                                               |
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for setting user profiles. The code is 0 if the operation is successful. |

## Room APIs

### createRoom

This API is used to create a room (called by anchor).



```
/// Create a room (called by anchor). If the room does not exist, the system will c
/// The process of creating a room and starting live streaming as an anchor is as f
/// 1. A user calls `startCameraPreview()` to enable camera preview and set beauty
/// 2. The user calls `createRoom()` to create a room, the result of which is retur
/// 3. The user calls `starPublish()` to push streams.
/// - Parameters:
///   - roomId: The room ID. You need to assign and manage the IDs in a centralized
///   - roomParam: room information, such as room name and cover information. If bo
///   - callback: Callback for room entry. The code is `0` if room entry is success
/// - Note:
///   - This API is called by an anchor to start live streaming. An anchor can crea
```

```
- (void)createRoomWithRoomID:(UInt32)roomID
    roomParam:(TRTCCreateRoomParam *)roomParam
    callback:(Callback _Nullable)callback
NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

The parameters are described below:

| Parameter | Type                                      | Description                                                                                                                                                                                                                                                     |
|-----------|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| roomID    | UInt32                                    | The room ID. You need to assign and manage the IDs in a centralized manner. Multiple <code>roomId</code> values can be aggregated into a live room list. Currently, Tencent Cloud does not provide list management services. Please manage your own room lists. |
| roomParam | TRTCCreateRoomParam                       | Room information, such as room name and cover information. If both the room list and room information are managed by yourself, you can ignore this parameter.                                                                                                   |
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for room creation. The code is <code>0</code> if the operation is successful.                                                                                                                                                                          |

The process of creating a room and starting live streaming as an anchor is as follows:

1. A user calls `startCameraPreview()` to enable camera preview and set beauty filters.
2. The user calls `createRoom()` to create a room, the result of which is returned via a callback.
3. The user calls `starPublish()` to push streams.

## destroyRoom

This API is used to terminate a room (called by anchor).



```
/// Terminate a room (called by anchor)
/// After creating a room, an anchor can call this API to terminate it.
/// - parameter callback: callback for room termination. The code is `0` if the operation is successful.
/// - Note:
///   - After creating a room, an anchor can call this API to terminate it
- (void)destroyRoom:(Callback _Nullable)callback
NS_SWIFT_NAME(destroyRoom(callback:));
```

The parameters are described below:

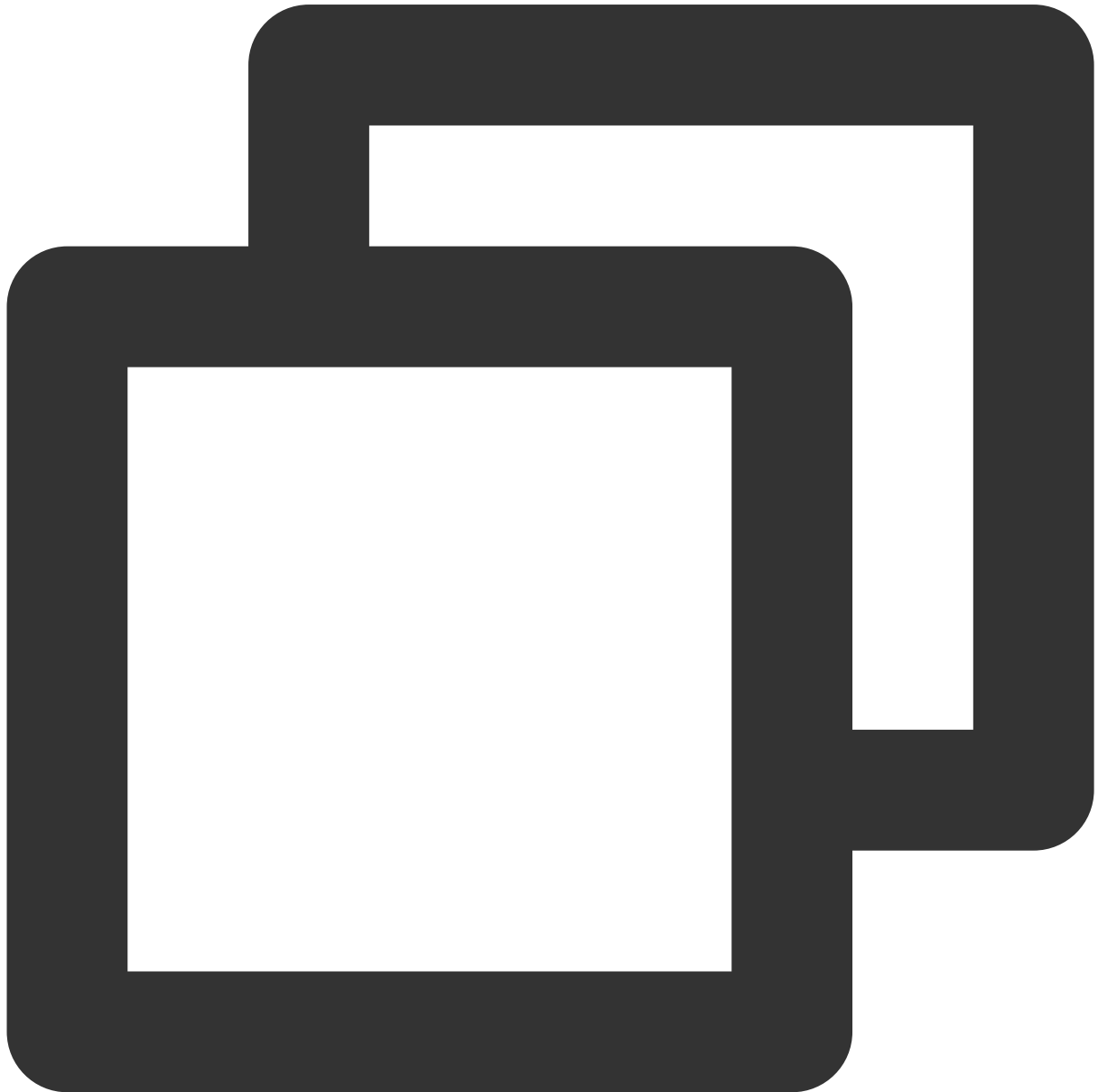
| Parameter | Type | Description |
|-----------|------|-------------|
|-----------|------|-------------|

callback

(\_ code: Int, \_ message:  
String?) -> VoidCallback for room termination. The code is `0` if the  
operation is successful.

## enterRoom

This API is used to enter a room (called by audience).



```
/// Enter a room (called by audience)
/// The process of entering a room and starting playback as an audience member is a
/// 1. A user gets the latest room list from your server. The list may contain the
/// 2. The user selects a room and calls `enterRoom()` to enter the room.
```

```

/// 3. If the room list managed by your server contains the `userID` of the anchor
/// If the room list contains `roomId` only, upon room entry, the user will receive
/// The user can then call `startPlay(userID)`, passing in the `userID` obtained fr
/// - Parameters:
///   - roomId: The room ID.
///   - useCDNFirst: whether to play streams via CDNs whenever possible
///   - cdnDomain: CDN domain name
///   - callback: Callback for room entry. The code is `0` if room entry is success
/// - Note:
///   - This API is called by a user to enter a room.
///   - An anchor cannot use this API to enter a room he or she created, but must u
- (void)enterRoomWithRoomID:(UInt32)roomId
    useCDNFirst:(BOOL)useCDNFirst
    cdnDomain:(NSString * _Nullable)cdnDomain
    callback:(Callback)callback
NS_SWIFT_NAME (enterRoom(roomID:useCDNFirst:cdnDomain:callback:));

```

The parameters are described below:

| Parameter | Type                                         | Description                                                            |
|-----------|----------------------------------------------|------------------------------------------------------------------------|
| roomId    | UInt32                                       | The room ID.                                                           |
| callback  | (_ code: Int, _ message: String?)<br>-> Void | Callback for room entry. The code is 0 if the operation is successful. |

The process of entering a room and starting playback as an audience member is as follows:

1. A user gets the latest room list from your server. The list may contain the `roomId` and other information of multiple rooms.
2. The user selects a room and calls `enterRoom()` to enter the room.
3. The user calls `startPlay(userID)`, passing in the anchor's `userID` to start playback.

If the room list contains anchors' `userID`, the user can call `startPlay(userID)` to start playback.

If the user does not have the anchor's `userID` before room entry, he or she can find it in the

`onAnchorEnter(userID)` callback of `TRTCLiveRoomDelegate`, which is returned after room entry, and can then call `startPlay(userID)` to start playback.

## exitRoom

This API is used to leave a room.



```
/// Leave a room (called by audience)
/// - Parameter callback: callback for room exit. The code is `0` if the operation
/// - Note:
///   - This API is called by an audience member to leave a room.
///   - An anchor cannot use this API to leave a room.

- (void)exitRoom:(Callback _Nullable)callback
NS_SWIFT_NAME(exitRoom(callback:));
```

The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|



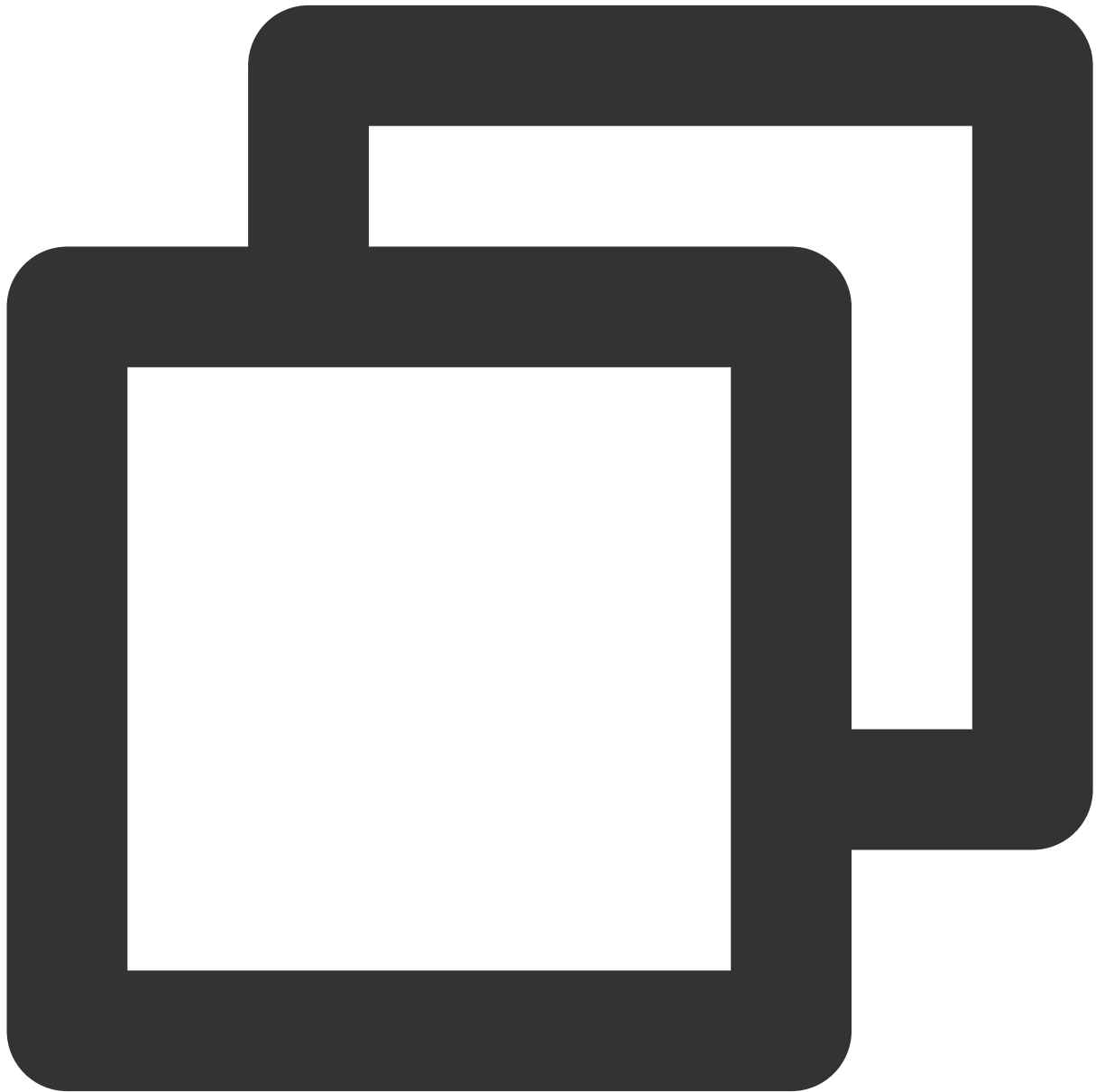
| Parameter | Type                                      | Description                                                           |
|-----------|-------------------------------------------|-----------------------------------------------------------------------|
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for room exit. The code is 0 if the operation is successful. |

## getRoomInfos

This API is used to get room list details, which are set by anchors via `roomInfo` when they call `createRoom()`.

### Note

You don't need this API if both the room list and room information are managed on your server.



```
/// Get room details
/// The information is provided by anchors via the `roomInfo` parameter when they c
/// - Parameter roomIDs: The list of room IDs.
/// - Parameter callback: Callback of room details.
- (void)getRoomInfosWithRoomIDs:(NSArray<NSNumber *> *)roomIDs
    callback:(RoomInfoCallback _Nullable)callback
NS_SWIFT_NAME(getRoomInfos(roomIDs:callback:));
```

The parameters are described below:

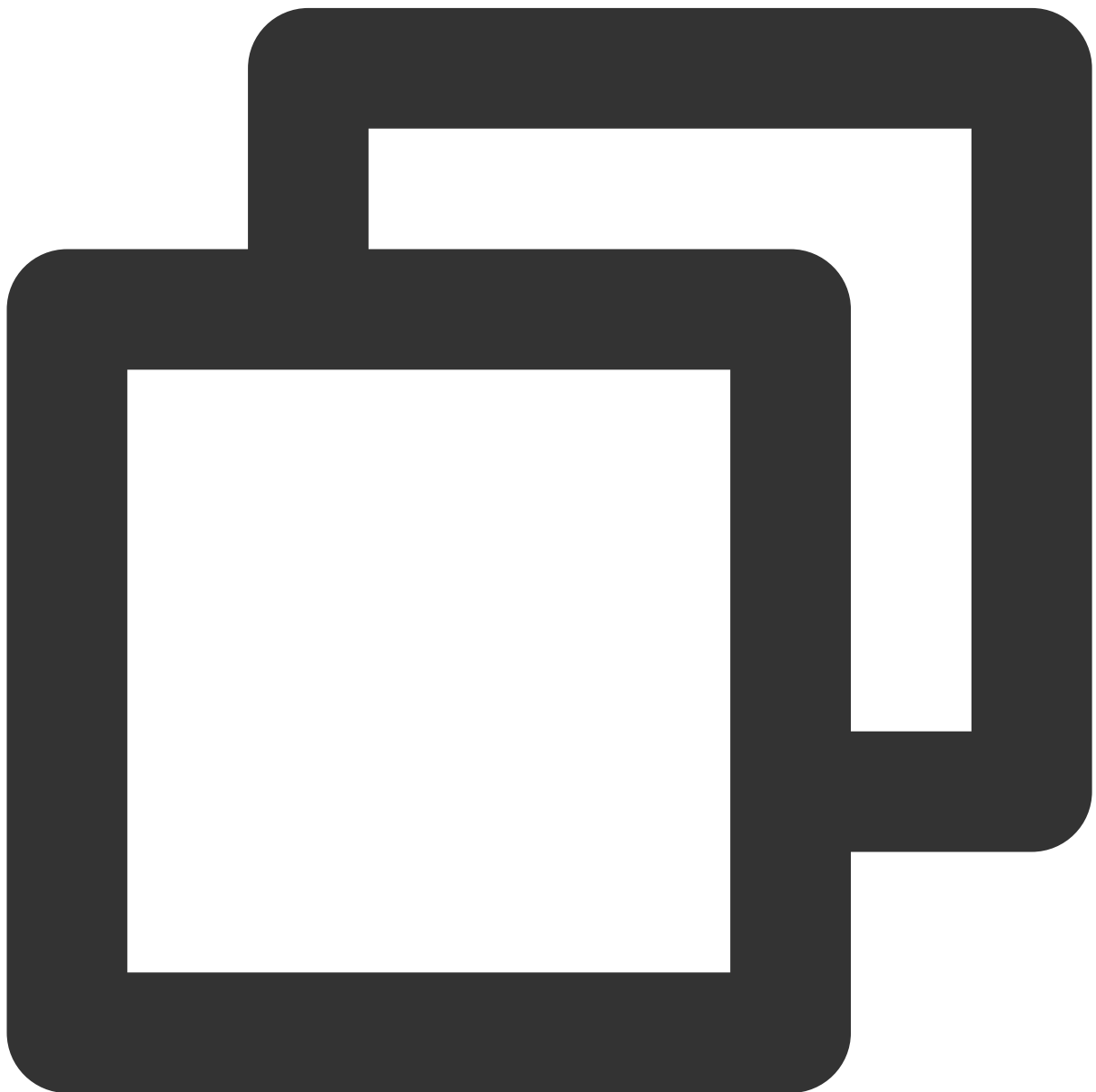
|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter | Type                                                                      | Description              |
|-----------|---------------------------------------------------------------------------|--------------------------|
| roomIDs   | [UInt32]                                                                  | List of room IDs         |
| callback  | (_ code: Int, _ message: String?, _ roomList: [TRTCLiveRoomInfo]) -> Void | Callback of room details |

## getAnchorList

This API is used to get the anchors and co-anchoring viewers in a room. It takes effect only if it is called after

```
enterRoom()
```



```
/// Get the anchors and co-anchoring viewers in a room. This API takes effect only  
/// - Parameter callback: Callback of user details.  
- (void)getAnchorList:(UserListCallback _Nullable)callback  
NS_SWIFT_NAME(getAnchorList(callback:));
```

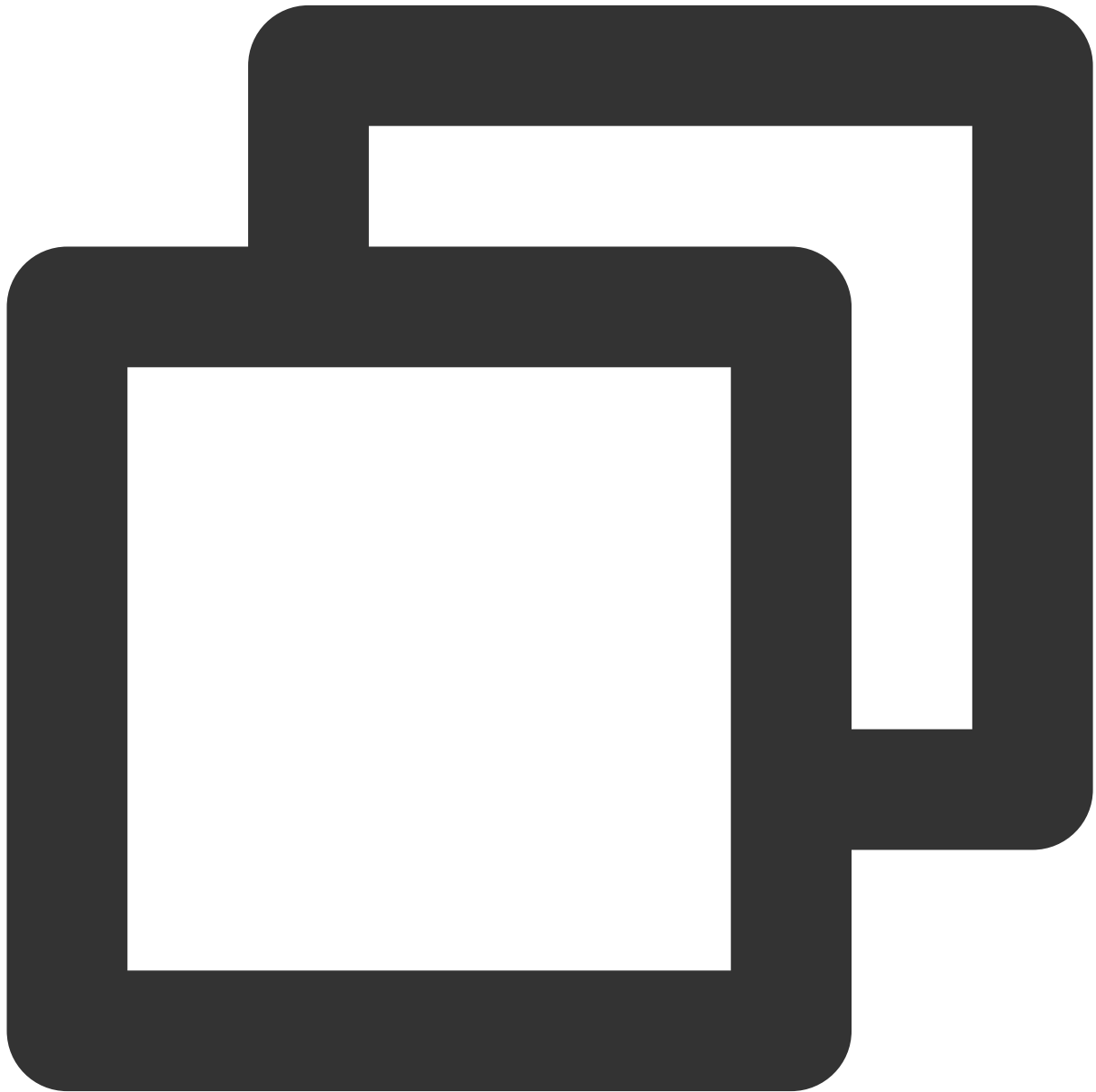
The parameters are described below:

| Parameter | Type                                                                     | Description              |
|-----------|--------------------------------------------------------------------------|--------------------------|
| callback  | (_ code: Int, _ message: String, _ userList: [TRTCLiveUserInfo]) -> Void | Callback of user details |

## getAudienceList

This API is used to get the information of all audience members in a room. It takes effect only if it is called after

```
enterRoom() .
```



```
/// Get the information of all audience members in a room. This API takes effect on  
/// - Parameter callback: Callback of user details.  
- (void)getAudienceList:(UserListCallback _Nullable)callback  
NS_SWIFT_NAME(getAudienceList(callback:));
```

The parameters are described below:

| Parameter | Type                                                                | Description      |
|-----------|---------------------------------------------------------------------|------------------|
| callback  | (_ code: Int, _ message: String, _ userList: [TRTCLiveUserInfo]) -> | Callback of user |

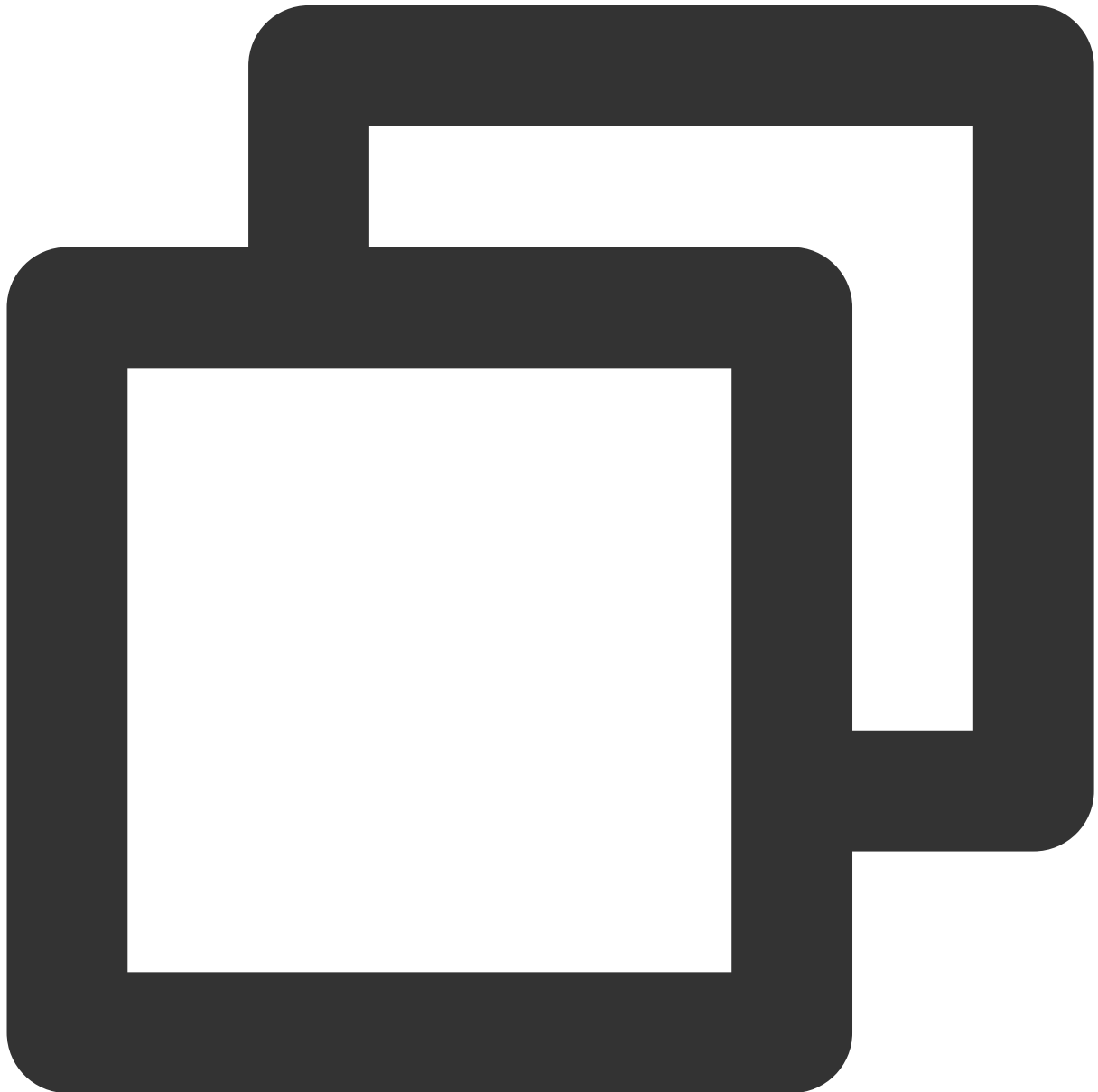
Void

[details](#)

## Stream Pushing/Pulling APIs

### **startCameraPreview**

This API is used to enable local video preview.



```
/// Enable local video preview  
/// - Parameters:
```

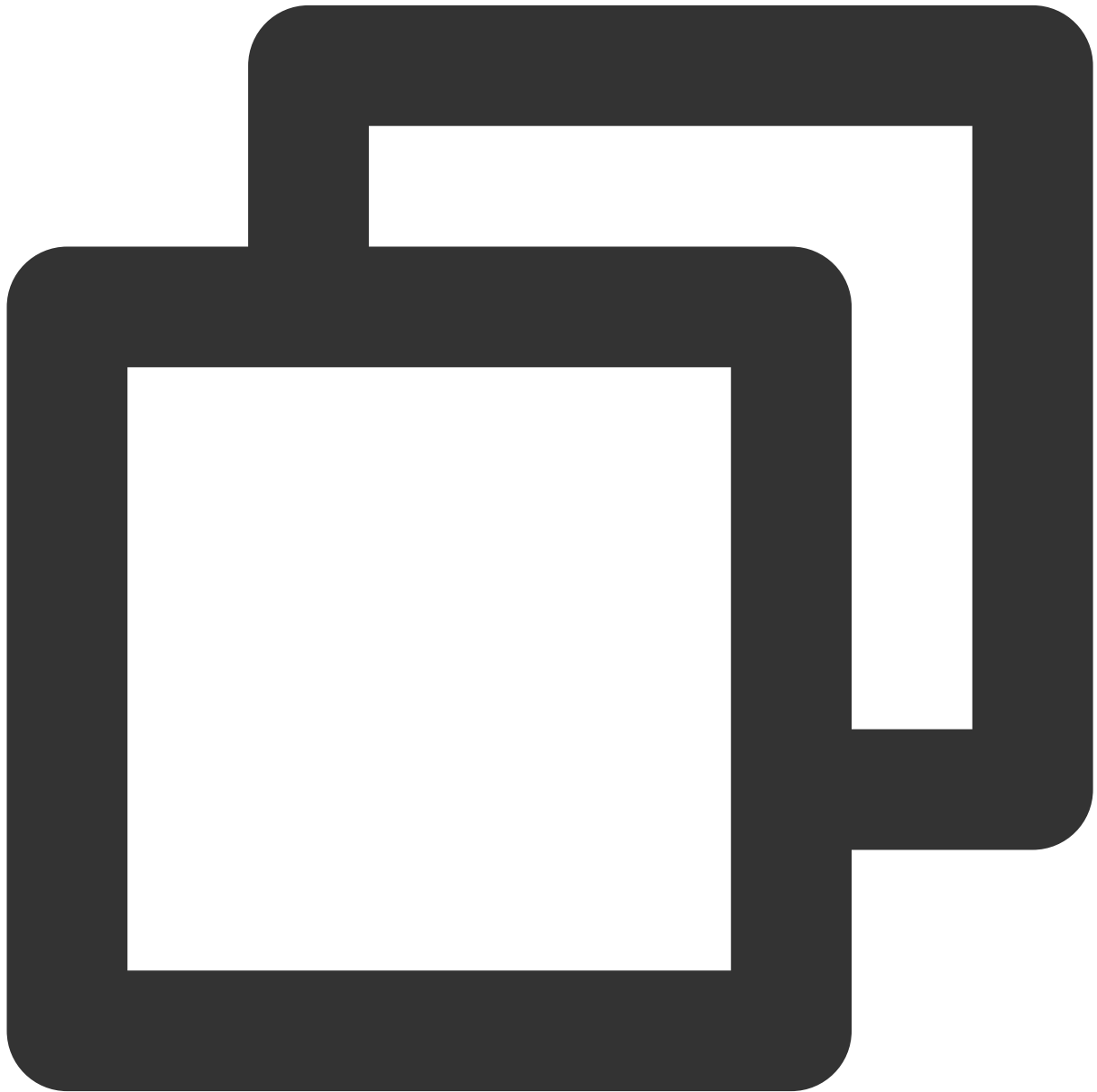
```
/// - frontCamera: `true`: Front camera; `false`: Rear camera
/// - view: The control that loads video images.
/// - callback: Callback for the operation
- (void)startCameraPreviewWithFrontCamera:(BOOL)frontCamera
    view:(UIView *)view
    callback:(Callback _Nullable)callback
NS_SWIFT_NAME(startCameraPreview(frontCamera:view:callback:));
```

The parameters are described below:

| Parameter   | Type                                      | Description                                                        |
|-------------|-------------------------------------------|--------------------------------------------------------------------|
| frontCamera | Bool                                      | <code>true</code> : Front camera; <code>false</code> : Rear camera |
| view        | UIView                                    | The control that loads video images                                |
| callback    | (_ code: Int, _ message: String?) -> Void | Callback for the operation                                         |

## stopCameraPreview

This API is used to stop local video capturing and preview.



```
/// Stop local video capturing and preview  
- (void)stopCameraPreview;
```

### **startPublish**

This API is used to start live streaming (pushing streams), which can be called in the following scenarios:

An anchor starts live streaming.

A viewer starts co-anchoring.





```
/// Start live streaming (push streams). This API can be called in the following scenarios:  
/// 1. An anchor starts live streaming.  
/// 2. An audience member starts co-anchoring.  
/// - Parameters:  
///   - streamID: the `streamID` used to bind live streaming CDNs. You need to set a unique  
///   - callback: Callback for the operation  
- (void)startPublishWithStreamID:(NSString *)streamID  
    callback:(Callback _Nullable)callback  
NS_SWIFT_NAME(startPublish(streamID:callback:));
```

The parameters are described below:

| Parameter | Type                                                     | Description                                                                                                                                                                                 |
|-----------|----------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| streamID  | String                                                   | The <code>streamID</code> used to bind live streaming CDNs. You need to set it to the <code>streamID</code> of the anchor if you want the audience to receive the anchor's stream via CDNs. |
| callback  | ( <code>_ code: Int, _ message: String?</code> ) -> Void | Callback for the operation                                                                                                                                                                  |

## stopPublish

This API is used to stop live streaming (pushing streams), which can be called in the following scenarios:

An anchor ends live streaming.

A viewer ends co-anchoring.



```
/// Stop live streaming (pushing streams). This API can be called in the following  
/// 1. An anchor ends live streaming.  
/// 2. An audience member ends co-anchoring.  
/// - Parameter callback: Callback for the operation.  
- (void)stopPublish:(Callback _Nullable)callback  
NS_SWIFT_NAME(stopPublish(callback:));
```

The parameters are described below:

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

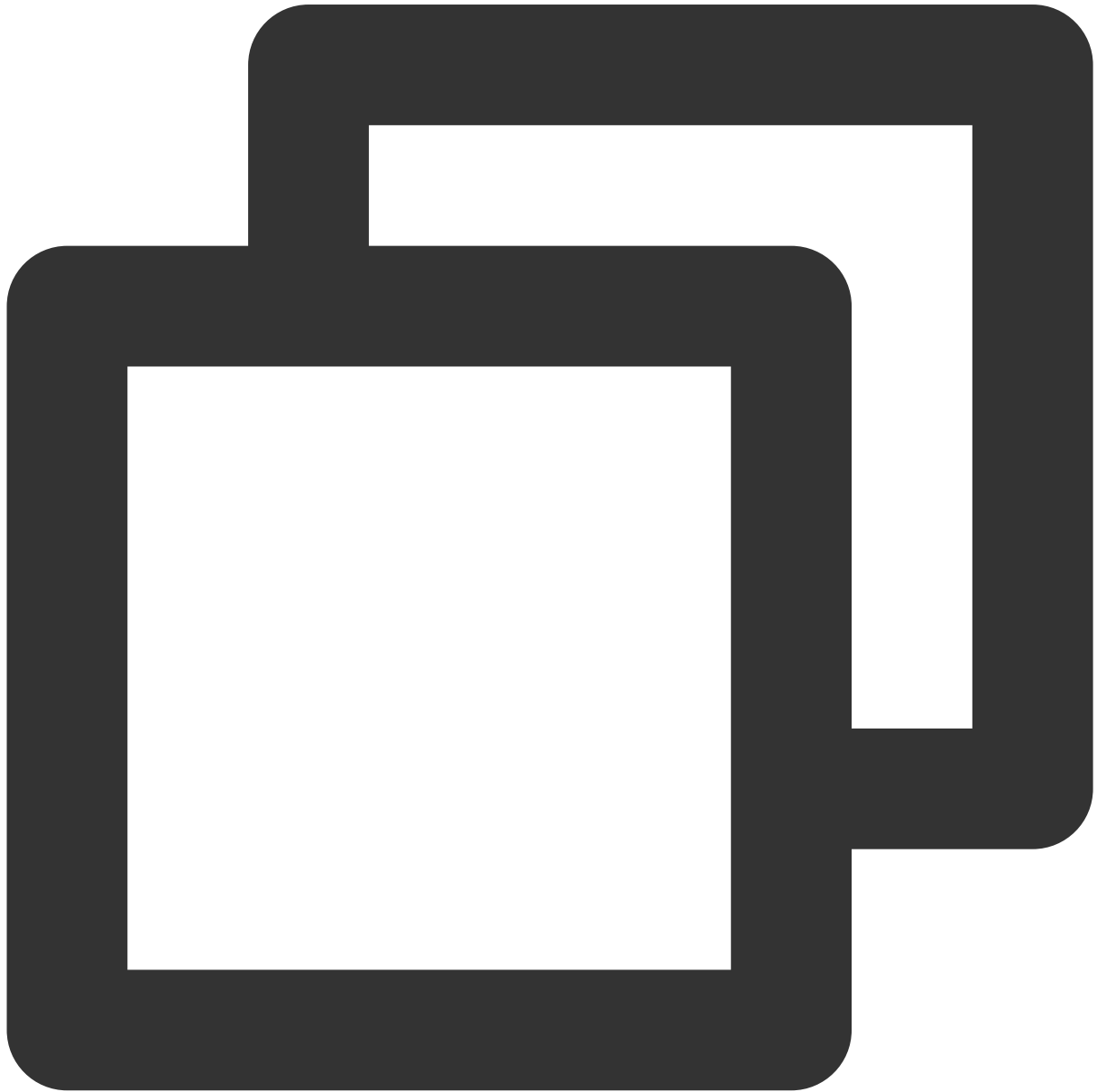
callback

(\_ code: Int, \_ message: String?) -&gt; Void

Callback for the operation

## startPlay

This API is used to play a remote video. It can be called in common playback and co-anchoring scenarios.



```
/// Play a remote video. This API can be called in common playback and co-anchoring
/// Common playback scenario
/// 1. If the room list managed by your server contains the `userID` of the anchor
/// 2. If the room list contains `roomId` only, upon room entry, a user will receive
/// The user can then call `startPlay(userID)`, passing in the `userID` obtained fr
```

```
/// Co-anchoring scenario
/// After co-anchoring starts, the anchor will receive the `onAnchorEnter(userID)`
/// - Parameters:
///   - userID: The ID of the user whose video is to be played.
///   - view: The control that loads video images.
///   - callback: Callback for the operation
- (void)startPlayWithUserID:(NSString *)userID
    view:(UIView *)view
    callback:(Callback _Nullable)callback
NS_SWIFT_NAME(startPlay(userID:view:callback:));
```

The parameters are described below:

| Parameter | Type                                      | Description                                |
|-----------|-------------------------------------------|--------------------------------------------|
| userID    | String                                    | ID of the user whose video is to be played |
| view      | UIView                                    | The control that loads video images        |
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for the operation                 |

### Common playback scenario

If the room list contains anchors' `userID`, after entering a room, a user can call `startPlay(userID)` to play the anchor's video.

If a user does not have the anchor's `userID` before room entry, he or she can find it in the `onAnchorEnter(userID) callback` of `TRTCLiveRoomDelegate`, which is returned after room entry, and can then call `startPlay(userID)` to play the anchor's video.

### Co-anchoring scenario

After co-anchoring starts, the anchor will receive the `onAnchorEnter(userID) callback` from `TRTCLiveRoomDelegate` and can call `startPlay(userID)`, passing in the `userID`` obtained from the callback to play the co-anchoring user's video.

### stopPlay

This API is used to stop rendering a remote video. It needs to be called after the `onAnchorExit()` callback is received.



```
/// Stop rendering a remote video
/// - Parameters:
///   - userID: The ID of the remote user.
///   - callback: Callback for the operation
/// - Note:
///   - Call this API after receiving the `onAnchorExit` callback.
- (void)stopPlayWithUserID:(NSString *)userID
    callback:(Callback _Nullable)callback
NS_SWIFT_NAME(stopPlay(userID:callback:));
```

The parameters are described below:

| Parameter | Type                                      | Description                |
|-----------|-------------------------------------------|----------------------------|
| userID    | String                                    | ID of the remote user      |
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for the operation |

## APIs for Anchor-Audience Co-anchoring

### **requestJoinAnchor**

This API is used to send a co-anchoring request (called by audience).



```
/// Send a co-anchoring request
/// - Parameters:
///   - reason: reason for co-anchoring
///   - responseCallback: callback of the response
/// - Note: After an audience member sends a co-anchoring request, the anchor will
- (void)requestJoinAnchor:(NSString *)reason
    timeout:(double)timeout
    responseCallback:(ResponseCallback _Nullable)responseCallback
NS_SWIFT_NAME(requestJoinAnchor(reason:timeout:responseCallback:));
```

The parameters are described below:



| Parameter        | Type                                        | Description                                 |
|------------------|---------------------------------------------|---------------------------------------------|
| reason           | String                                      | Reason for co-anchoring                     |
| timeout          | long                                        | Timeout period for the co-anchoring request |
| responseCallback | (_ agreed: Bool, _ reason: String?) -> Void | Callback of the anchor's response           |

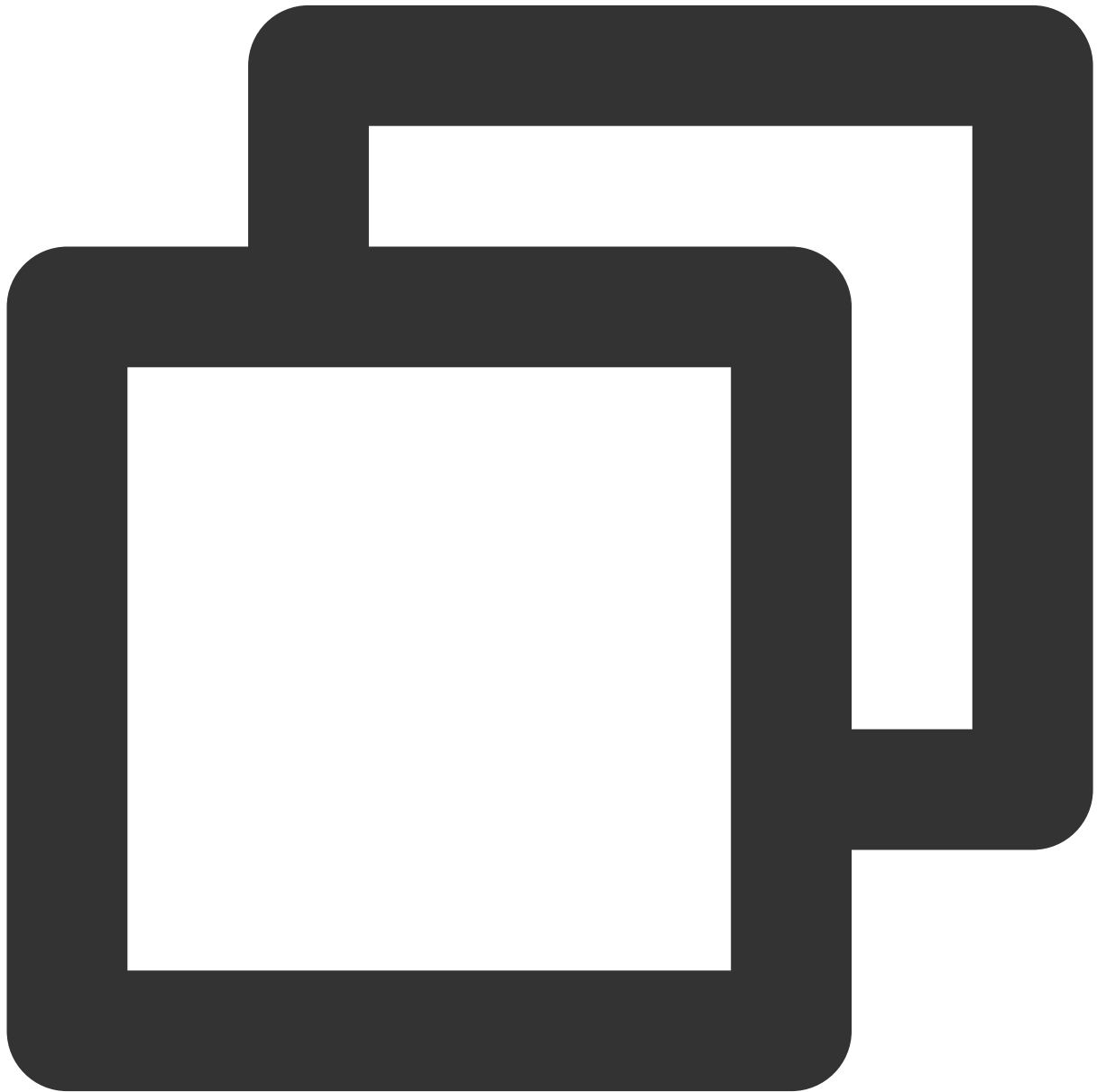
The process of co-anchoring between anchors and audience members is as follows:

1. A **viewer** calls `requestJoinAnchor()` to send a co-anchoring request to the anchor.
2. The **anchor** receives the `onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate`.
3. The **anchor** calls `responseJoinAnchor()` to accept or reject the co-anchoring request.
4. The **viewer** receives the `responseCallback` callback, which carries the anchor's response.
5. If the request is accepted, the **viewer** calls `startCameraPreview()` to enable local camera preview.
6. The **viewer** calls `startPublish()` to push streams.
7. After the viewer starts pushing streams, the **anchor** receives the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate`.
8. The **anchor** calls `startPlay()` to play the co-anchoring viewer's video.
9. If there are other viewers co-anchoring with the anchor in the room, the new co-anchoring **viewer** will receive the `onAnchorEnter()` callback and can call `startPlay()` to play other co-anchoring viewers' video.

## responseJoinAnchor

This API is used to respond to a co-anchoring request (called by anchor) after receiving the

`onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate`.



```
/// Respond to a co-anchoring request
/// - Parameters:
///   - user: user ID of the viewer
///   - agree: `true`: Accept; `false`: Reject
///   - reason: reason for accepting/rejecting the request
/// - Note: After the anchor responds to the request, the viewer will receive the `
- (void)responseJoinAnchor:(NSString *)userID
    agree:(BOOL)agree
    reason:(NSString *)reason
NS_SWIFT_NAME(responseJoinAnchor(userID:agree:reason:));
```

The parameters are described below:

| Parameter | Type    | Description                                             |
|-----------|---------|---------------------------------------------------------|
| userID    | String  | The user ID of the audience member.                     |
| agree     | Bool    | <code>true</code> : accept; <code>false</code> : reject |
| reason    | String? | Reason for accepting/rejecting the request              |

## kickoutJoinAnchor

This API is used to remove a user from co-anchoring (called by anchor). The removed user will receive the

`onKickoutJoinAnchor()` callback of `TRTCLiveRoomDelegate` .



```
/// Remove a user from co-anchoring
/// - Parameters:
///   - userID: ID of the user to remove from co-anchoring
///   - callback: Callback for the operation
/// - Note: After the anchor calls this API to remove a user from co-anchoring, the
- (void)kickoutJoinAnchor:(NSString *)userID
    callback:(Callback _Nullable)callback
NS_SWIFT_NAME(kickoutJoinAnchor(userID:callback:));
```

The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter | Type                                      | Description                                     |
|-----------|-------------------------------------------|-------------------------------------------------|
| userID    | String                                    | The ID of the user to remove from co-anchoring. |
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for the operation                      |

## APIs for Cross-Room Communication

### **requestRoomPK**

This API is used to send a cross-room communication request (called by anchor).



```
/// Request cross-room communication
/// - Parameters:
///   - roomId: room ID of the anchor to invite
///   - userID: user ID of the anchor to invite
///   - responseCallback: callback of the response
/// - Note: After a cross-room communication request is sent, the invited anchor wi
- (void)requestRoomPKWithRoomID:(UInt32)roomId
    userID:(NSString *)userID
    responseCallback:(ResponseCallback _Nullable)responseCallback
NS_SWIFT_NAME(requestRoomPK(roomID:userID:responseCallback:));
```

The parameters are described below:

| Parameter        | Type                                        | Description                     |
|------------------|---------------------------------------------|---------------------------------|
| roomId           | UInt32                                      | room ID of the anchor to invite |
| userID           | String                                      | user ID of the anchor to invite |
| responseCallback | (_ agreed: Bool, _ reason: String?) -> Void | Callback of the response        |

Anchors in different rooms can communicate with each other. The process of starting cross-room communication between anchor A and anchor B is as follows:

1. **Anchor A** calls `requestRoomPK()` to send a cross-room communication to anchor B.
2. **Anchor B** receives the `onRequestRoomPK()` callback of `TRTCLiveRoomDelegate`.
3. **Anchor B** calls `responseRoomPK()` to respond to the cross-room communication request.
4. If **anchor B** accepts the request, he or she would wait for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and call `startPlay()` to play anchor A's video.
5. **Anchor A** receives the `responseCallback` callback, which carries anchor B's response.
6. If the request is accepted, **anchor A** waits for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and calls `startPlay()` to play anchor B's video.

## responseRoomPK

This API is used to respond to a cross-room communication request (called by anchor), after which the request sending anchor will receive the `responseCallback` passed in to `requestRoomPK`.



```
/// Respond to a cross-room communication request
/// Respond to a communication request from another anchor
/// - Parameters:
///   - user: user ID of the request sending anchor
///   - agree: `true`: Accept; `false`: Reject
///   - reason: reason for accepting/rejecting the request
/// - Note: After the anchor responds to the request, the anchor sending the request
- (void)responseRoomPKWithUserID:(NSString *)userID
    agree:(BOOL)agree
    reason:(NSString *)reason
NS_SWIFT_NAME(responseRoomPK(userID:agree:reason:));
```



The parameters are described below:

| Parameter | Type    | Description                                             |
|-----------|---------|---------------------------------------------------------|
| userID    | String  | User ID of the request sending anchor                   |
| agree     | Bool    | <code>true</code> : accept; <code>false</code> : reject |
| reason    | String? | Reason for accepting/rejecting the request              |

## quitRoomPK

This API is used to quit cross-room communication. If either anchor quits cross-room communication, the other anchor will receive the `trtcLiveRoomOnQuitRoomPK()` callback of `TRTCLiveRoomDelegate`.



```
/// End cross-room communication
/// - Parameter callback: Callback for ending cross-room communication
// - Note: If either anchor ends cross-room communication, the other anchor will re
- (void)quitRoomPK:(Callback _Nullable)callback
NS_SWIFT_NAME(quitRoomPK(callback:));
```

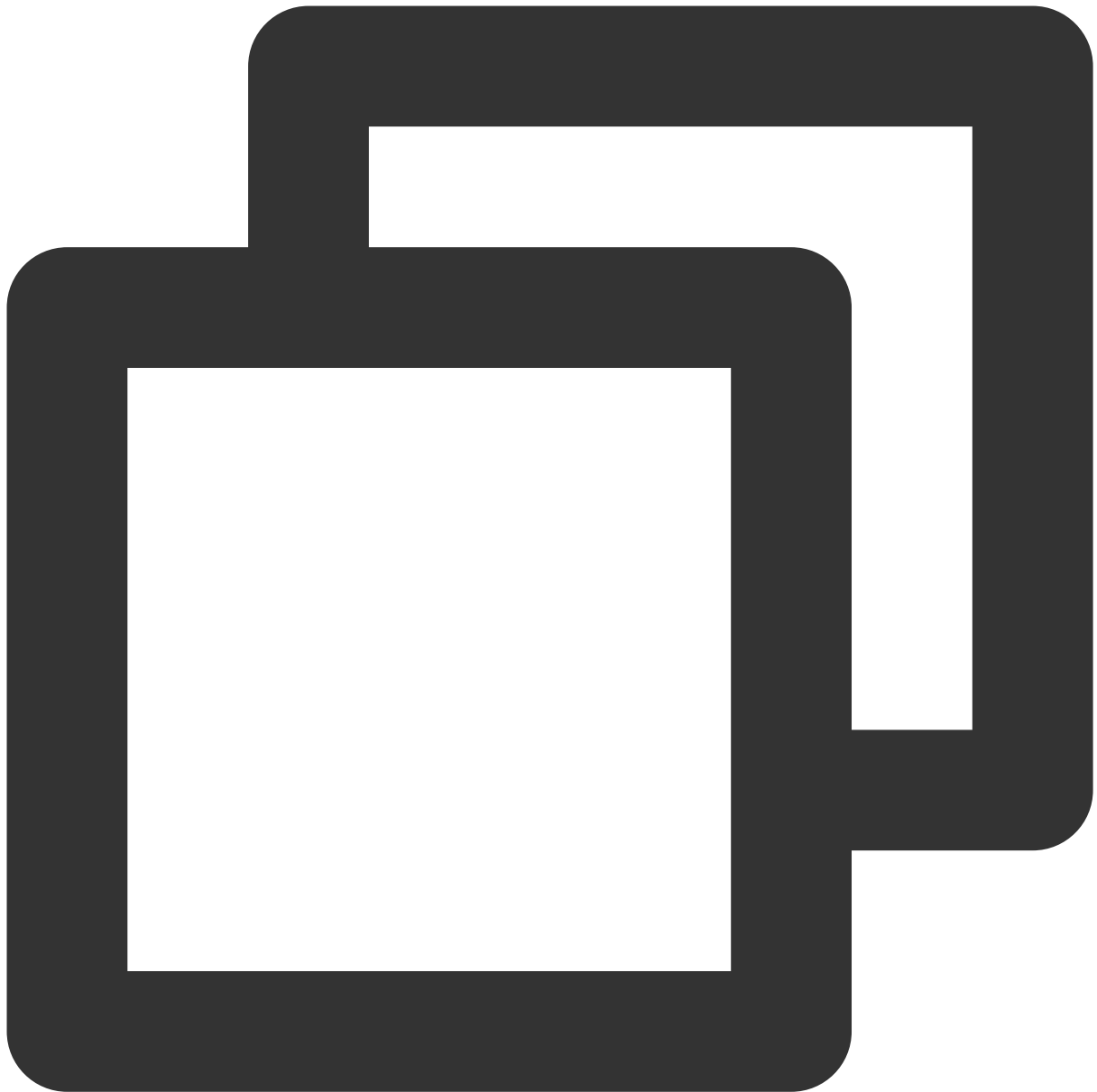
The parameters are described below:

| Parameter | Type                                      | Description                |
|-----------|-------------------------------------------|----------------------------|
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for the operation |

## Audio/Video APIs

### switchCamera

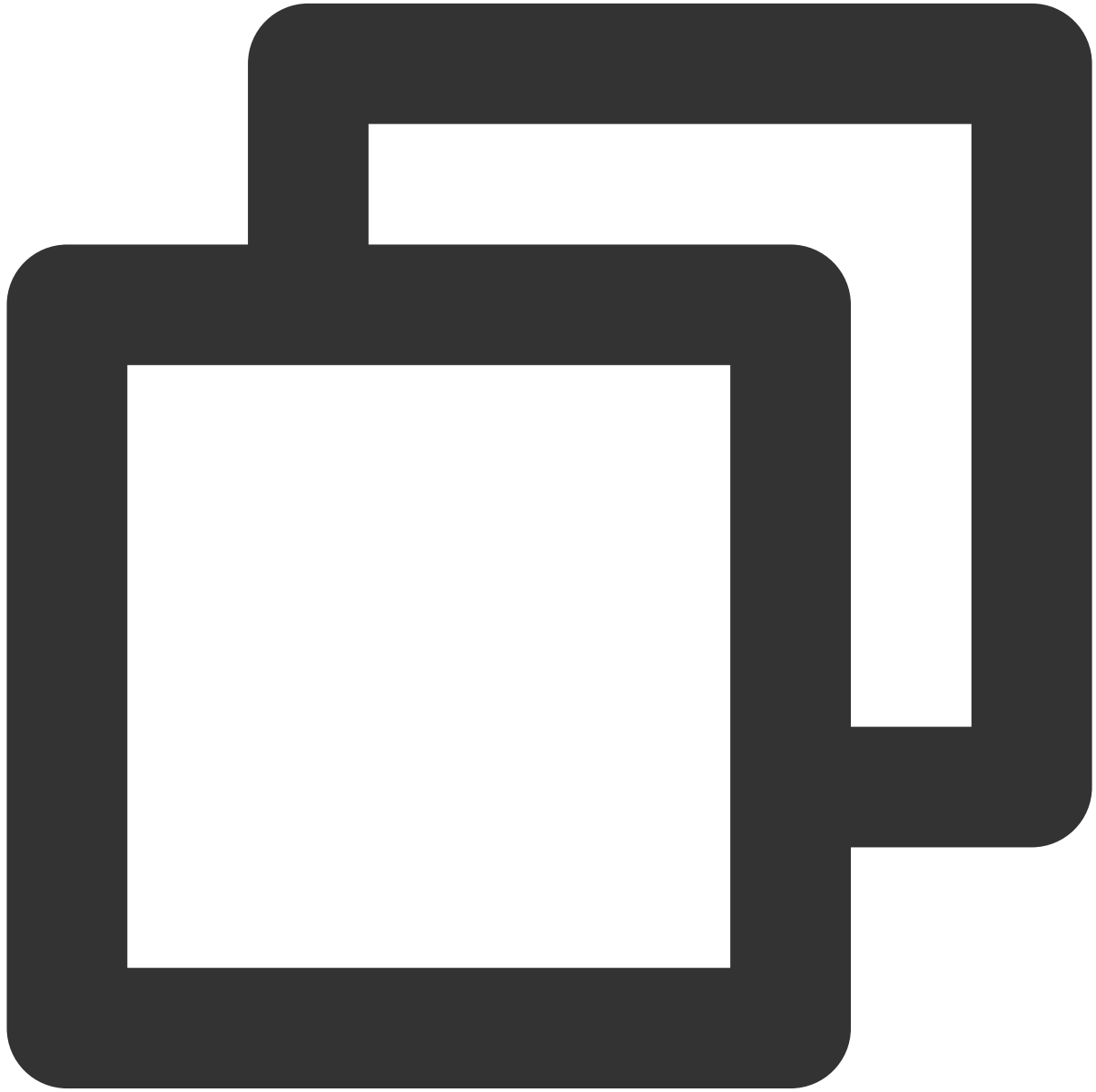
This API is used to switch between the front and rear cameras.



```
/// Switch between the front and rear cameras  
- (void)switchCamera;
```

### setMirror

This API is used to set the mirror mode.



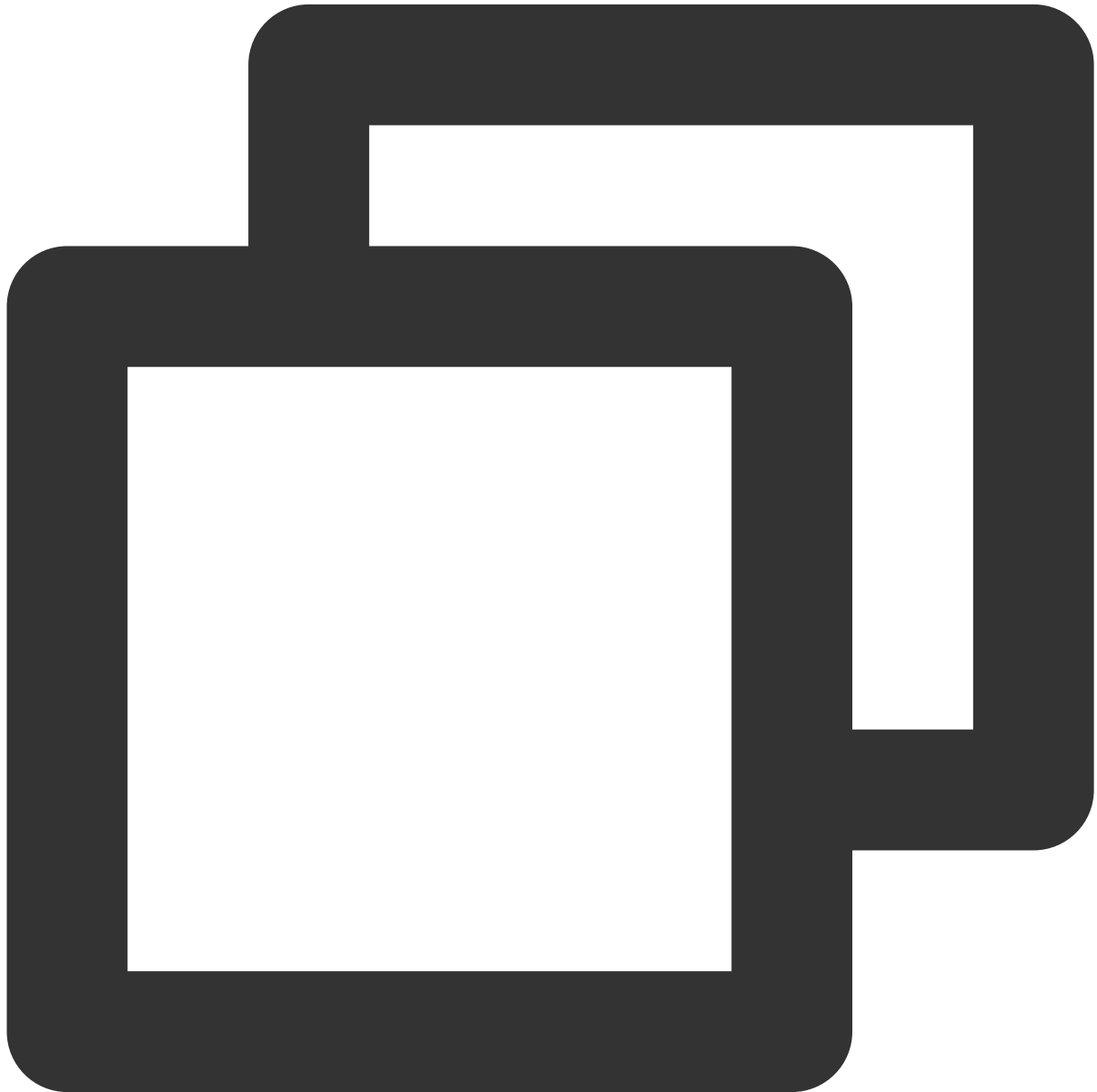
```
/// Specify whether to mirror video
/// - Parameter isMirror: Enable/Disable mirroring
- (void)setMirror:(BOOL)isMirror
NS_SWIFT_NAME(setMirror(isMirror:));
```

The parameters are described below:

| Parameter | Type | Description              |
|-----------|------|--------------------------|
| isMirror  | Bool | Enable/Disable mirroring |

## **muteLocalAudio**

This API is used to mute or unmute the local user.



```
/// Mute or unmute the local user
/// - Parameter isMuted: `true`: Mute; `false`: Unmute
- (void)muteLocalAudio:(BOOL)isMuted
NS_SWIFT_NAME(muteLocalAudio(isMuted));
```

The parameters are described below:

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

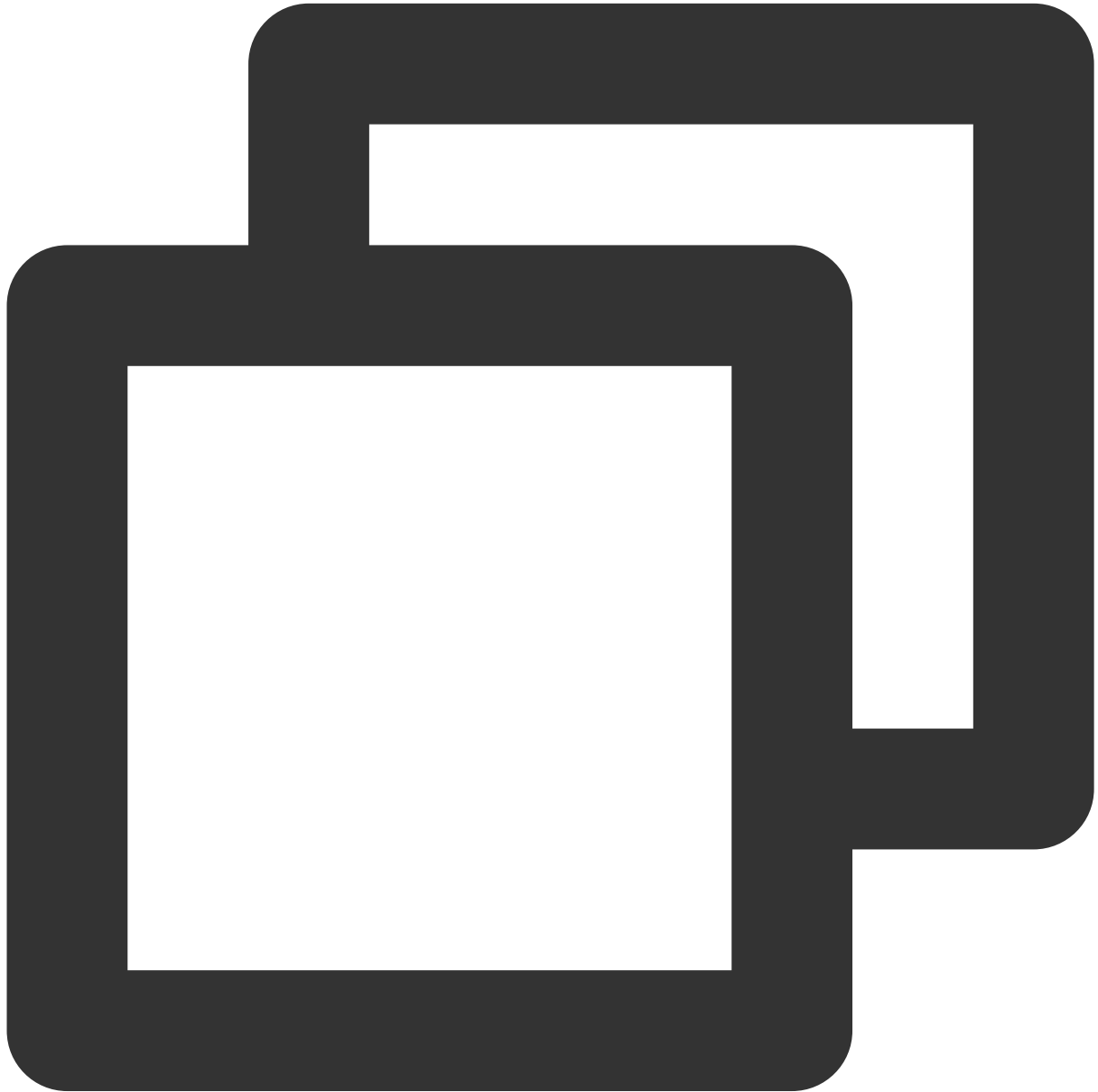
isMuted

Bool

`true` : mute; `false` : unmute

## **muteRemoteAudio**

This API is used to mute or unmute a remote user.



```
/// Mute or unmute a remote user
/// - Parameters:
///   - userID: ID of the remote user
///   - isMuted: `true`: Mute; `false`: Unmute
- (void)muteRemoteAudioWithUserID:(NSString *)userID isMuted:(BOOL)isMuted
```

```
NS_SWIFT_NAME(muteRemoteAudio(userID:isMuted:));
```

The parameters are described below:

| Parameter | Type   | Description                                           |
|-----------|--------|-------------------------------------------------------|
| userID    | String | ID of the remote user                                 |
| isMuted   | Bool   | <code>true</code> : mute; <code>false</code> : unmute |

## **muteAllRemoteAudio**

This API is used to mute or unmute all remote users.



```
/// Mute or unmute all remote users
/// - Parameter isMuted: `true`: Mute; `false`: Unmute
- (void)muteAllRemoteAudio:(BOOL)isMuted
NS_SWIFT_NAME(muteAllRemoteAudio(_));
```

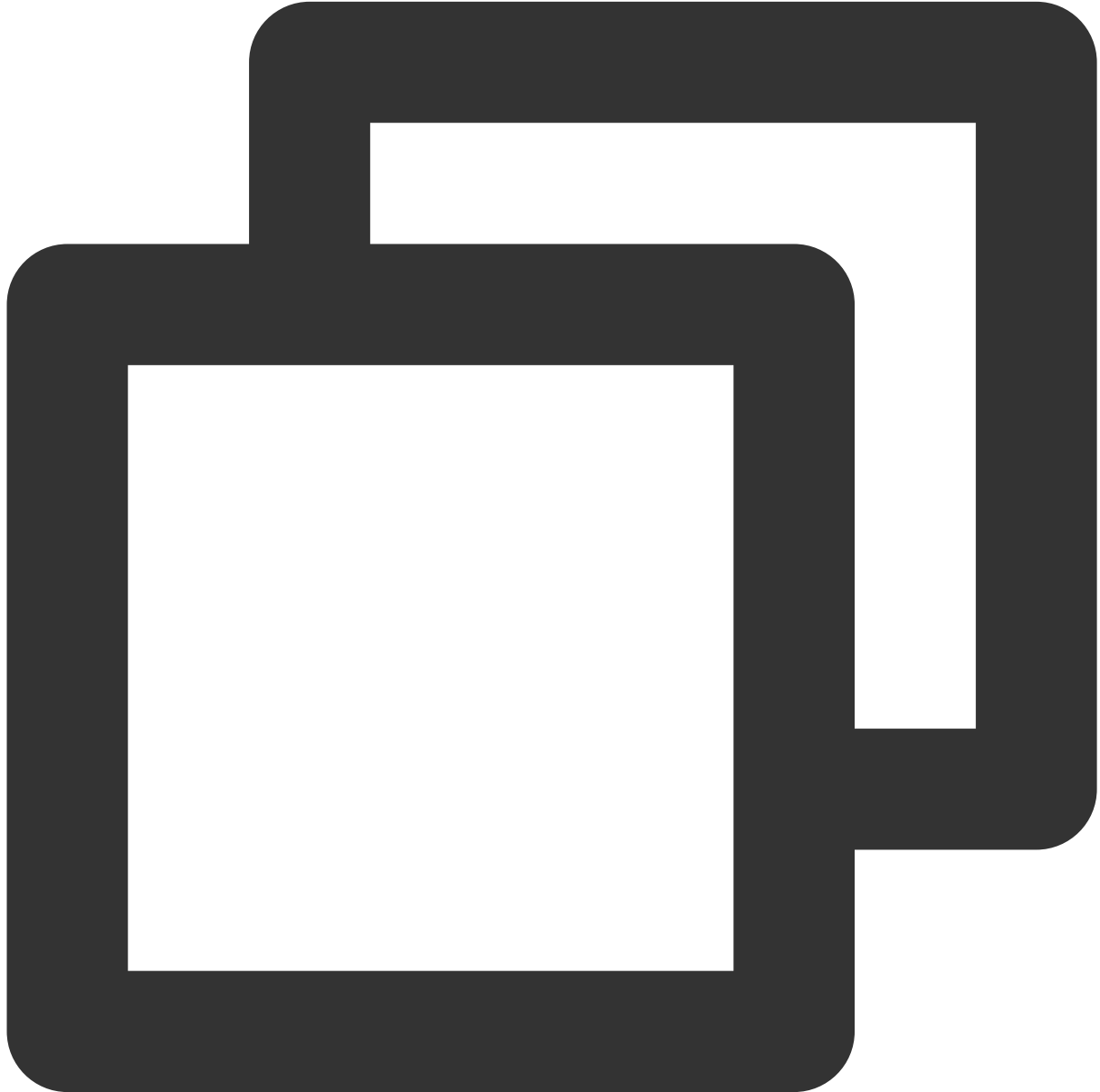
The parameters are described below:

| Parameter | Type | Description                                           |
|-----------|------|-------------------------------------------------------|
| isMuted   | Bool | <code>true</code> : mute; <code>false</code> : unmute |



## setAudioQuality

This API is used to set audio quality.



```
/// Set audio quality. Valid values: `1` (low), `2` (average), `3` (high)
/// - Parameter quality: audio quality
- (void)setAudioQuality:(NSInteger)quality
NS_SWIFT_NAME(setAudioQuality(qquality:));
```

The parameters are described below:

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

quality

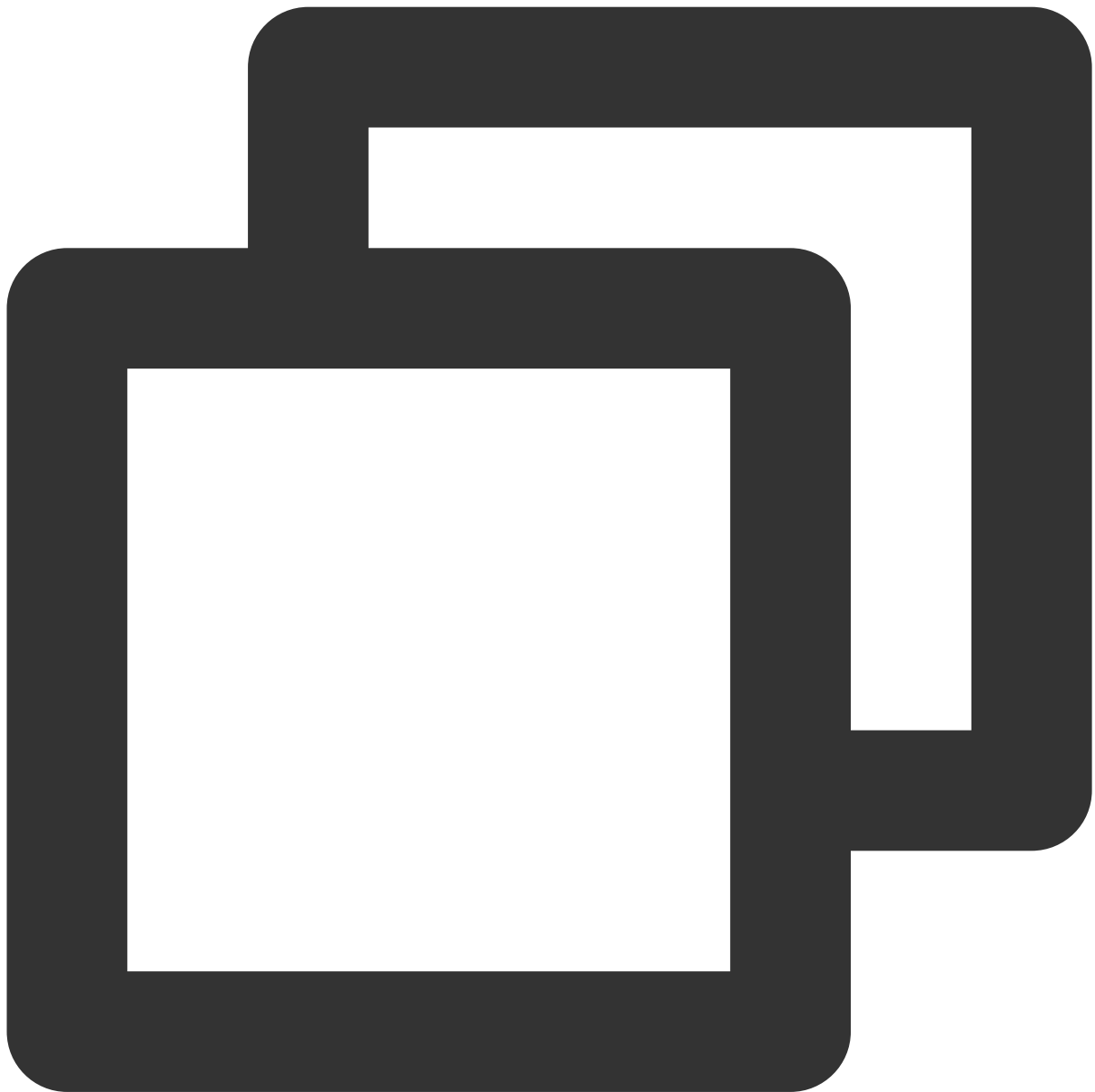
NSInteger

1 : speech; 2 : standard; 3 : music

## Background Music and Audio Effect APIs

### getAudioEffectManager

This API is used to get the background music and audio effect management object [TXAudioEffectManager](#).

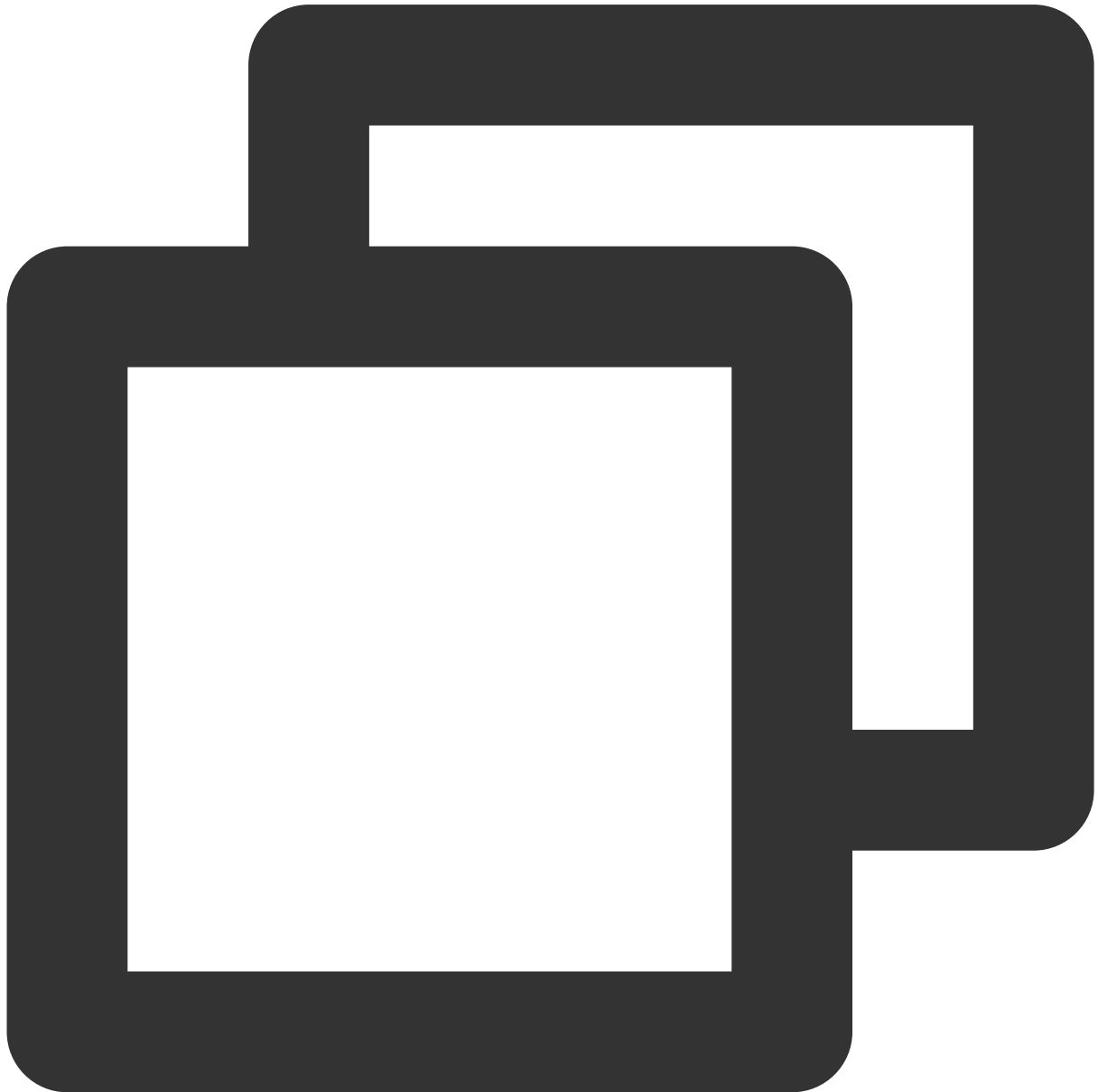


```
/// Get the audio effect management object  
- (TXAudioEffectManager *)getAudioEffectManager;
```

## Beauty Filter APIs

### getBeautyManager

This API is used to get the beauty filter management object [TXBeautyManager](#).



```
/* Get the beauty filter management object TXBeautyManager  
*/
```

```
* You can do the following using TXBeautyManager:  
* - Set the beauty filter style and apply effects including skin brightening, rosy  
* - Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose posi  
* - Apply animated effects such as face widgets (materials).  
* - Add makeup effects.  
* - Recognize gestures.  
*/  
- (TXBeautyManager *)getBeautyManager;
```

You can do the following using `TXBeautyManager` :

Set the beauty filter style and apply effects including skin brightening, rosy skin, eye enlarging, face slimming, chin slimming, chin lengthening/shortening, face shortening, nose narrowing, eye brightening, teeth whitening, eye bag removal, wrinkle removal, and smile line removal.

Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose position, lip thickness, and face shape.

Apply animated effects such as face widgets (materials).

Add makeup effects.

Recognize gestures.

## Message Sending APIs

### **sendRoomTextMsg**

This API is used to broadcast a text chat message in a room, which is generally used for on-screen comments.



```
/// Send a text chat message that can be seen by all users in a room
/// - Parameters:
///   - message: text chat message
///   - callback: Callback for message sending
- (void)sendRoomTextMsg:(NSString *)message callback:(Callback _Nullable)callback
NS_SWIFT_NAME(sendRoomTextMsg(message:callback:));
```

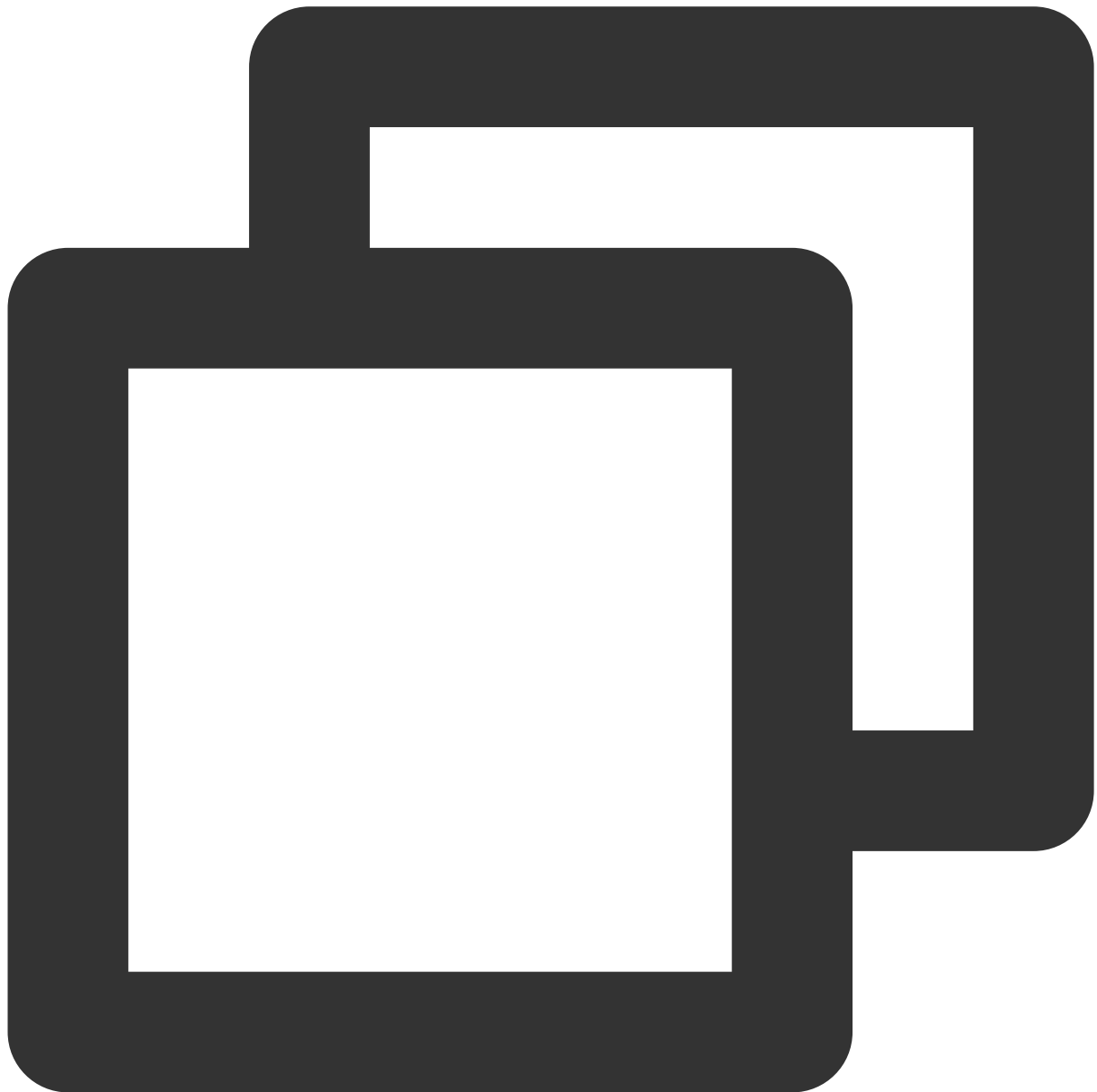
The parameters are described below:

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

|          |                                           |                              |
|----------|-------------------------------------------|------------------------------|
| message  | String                                    | Text message                 |
| callback | (_ code: Int, _ message: String?) -> Void | Callback for message sending |

## sendRoomCustomMsg

This API is used to send a custom text message.



```
/// Send a custom message
/// - Parameters:
///   - command: custom command word used to distinguish between different message
```

```
/// - message: text message
/// - callback: Callback for message sending
- (void)sendRoomCustomMsgWithCommand:(NSString *)command message:(NSString *)message
NS_SWIFT_NAME(sendRoomCustomMsg(command:message:callback:));
```

The parameters are described below:

| Parameter | Type                                      | Description                                                             |
|-----------|-------------------------------------------|-------------------------------------------------------------------------|
| command   | String                                    | Custom command word used to distinguish between different message types |
| message   | String                                    | Custom text message                                                     |
| callback  | (_ code: Int, _ message: String?) -> Void | Callback for message sending                                            |

## Debugging APIs

### showVideoDebugLog

This API is used to specify whether to display debugging information on the UI.



```
/// Specify whether to display debugging information on the UI
/// - Parameter isShow: show/hide debugging information
- (void)showVideoDebugLog:(BOOL)isShow
NS_SWIFT_NAME(showVideoDebugLog(_:));
```

The parameters are described below:

| Parameter | Type | Description                     |
|-----------|------|---------------------------------|
| isShow    | Bool | Show/Hide debugging information |



## TRTCLiveRoomDelegate Event Callback APIs

### Common Event Callback APIs

#### **onError**

Callback for error.

#### **Note**

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
    onError:(NSInteger)code
    message:(NSString *)message
NS_SWIFT_NAME(trtcLiveRoom(_:onError:message:));
```

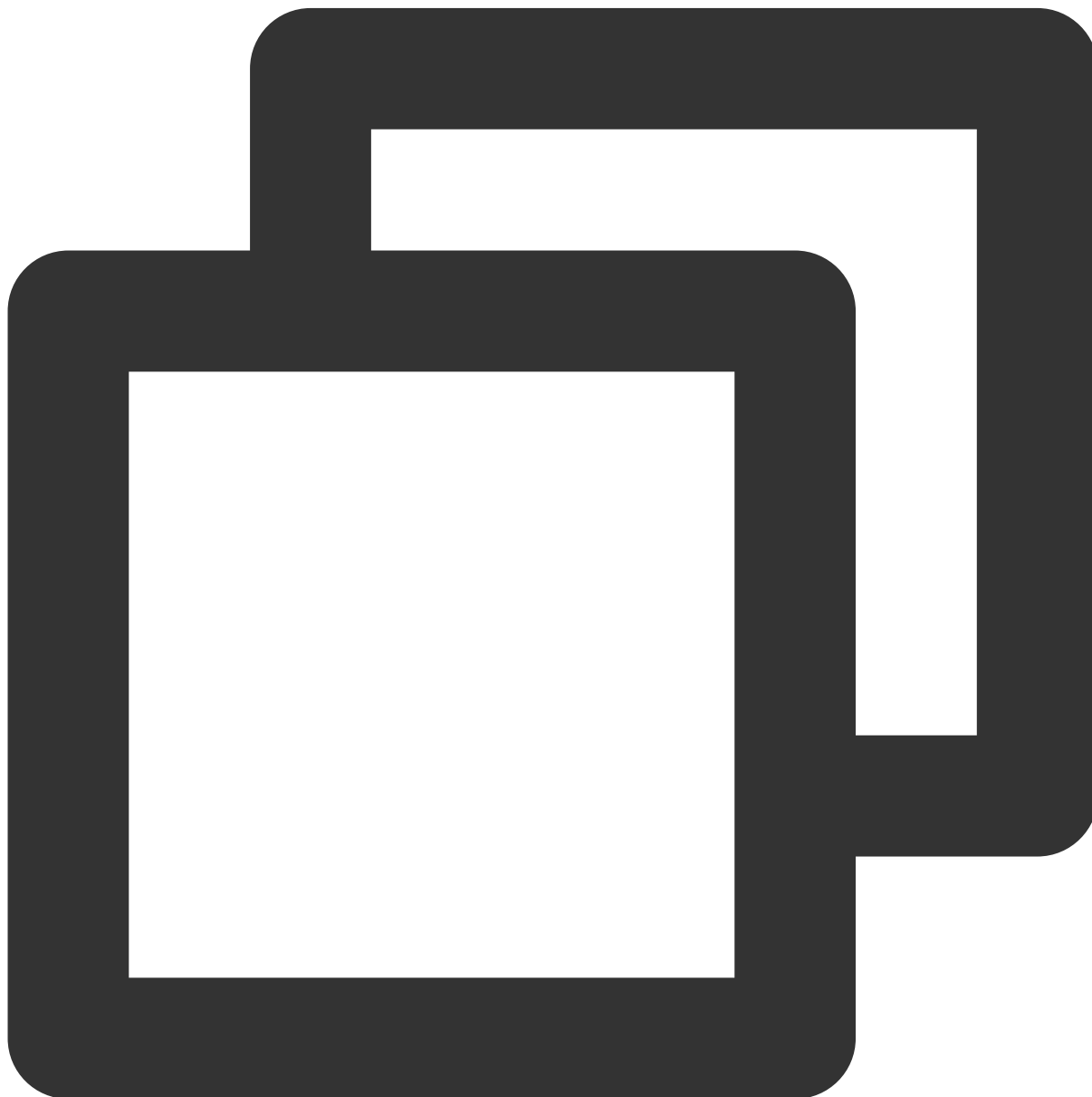
The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
|              |                  |                                                      |

|         |         |               |
|---------|---------|---------------|
| code    | Int     | Error code    |
| message | String? | Error message |

## onWarning

Callback for warning.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
    onWarning:(NSInteger)code
    message:(NSString *)message
```

```
NS_SWIFT_NAME(trtcLiveRoom(_:onWarning:message:));
```

The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| code         | Int              | TRTCWarningCode                                      |
| message      | String?          | Warning message                                      |

## onDebugLog

Callback for log.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
    onDebugLog:(NSString *)log
NS_SWIFT_NAME(trtcLiveRoom(_:onDebugLog:));
```

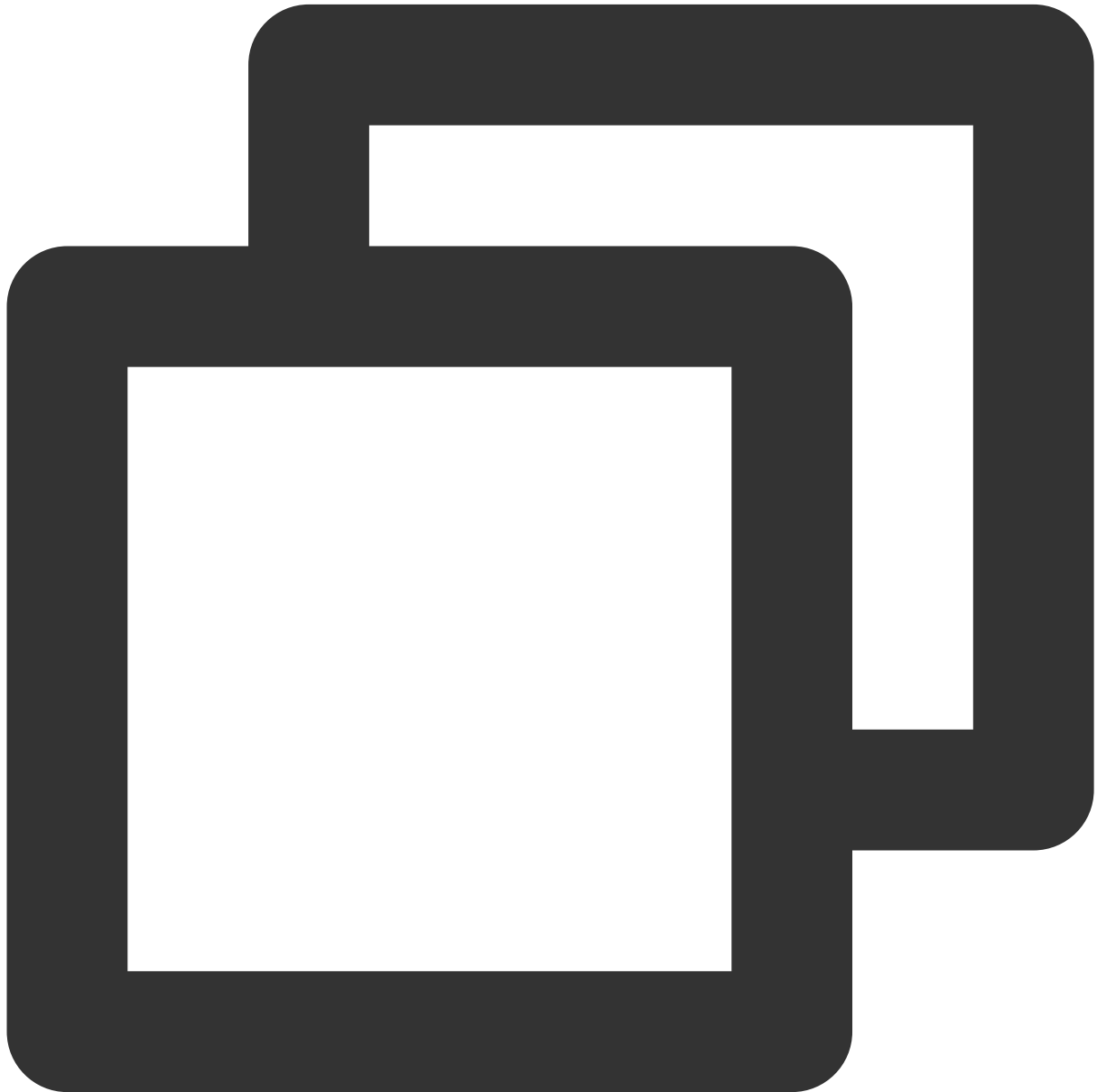
The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| log          | String           | Log information                                      |

## Room Event Callback APIs

### onRoomDestroy

Callback for room termination. All users in a room will receive this callback after the anchor leaves the room.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
    onRoomDestroy:(NSString *)roomId
NS_SWIFT_NAME(trtcLiveRoom(_:onRoomDestroy:));
```

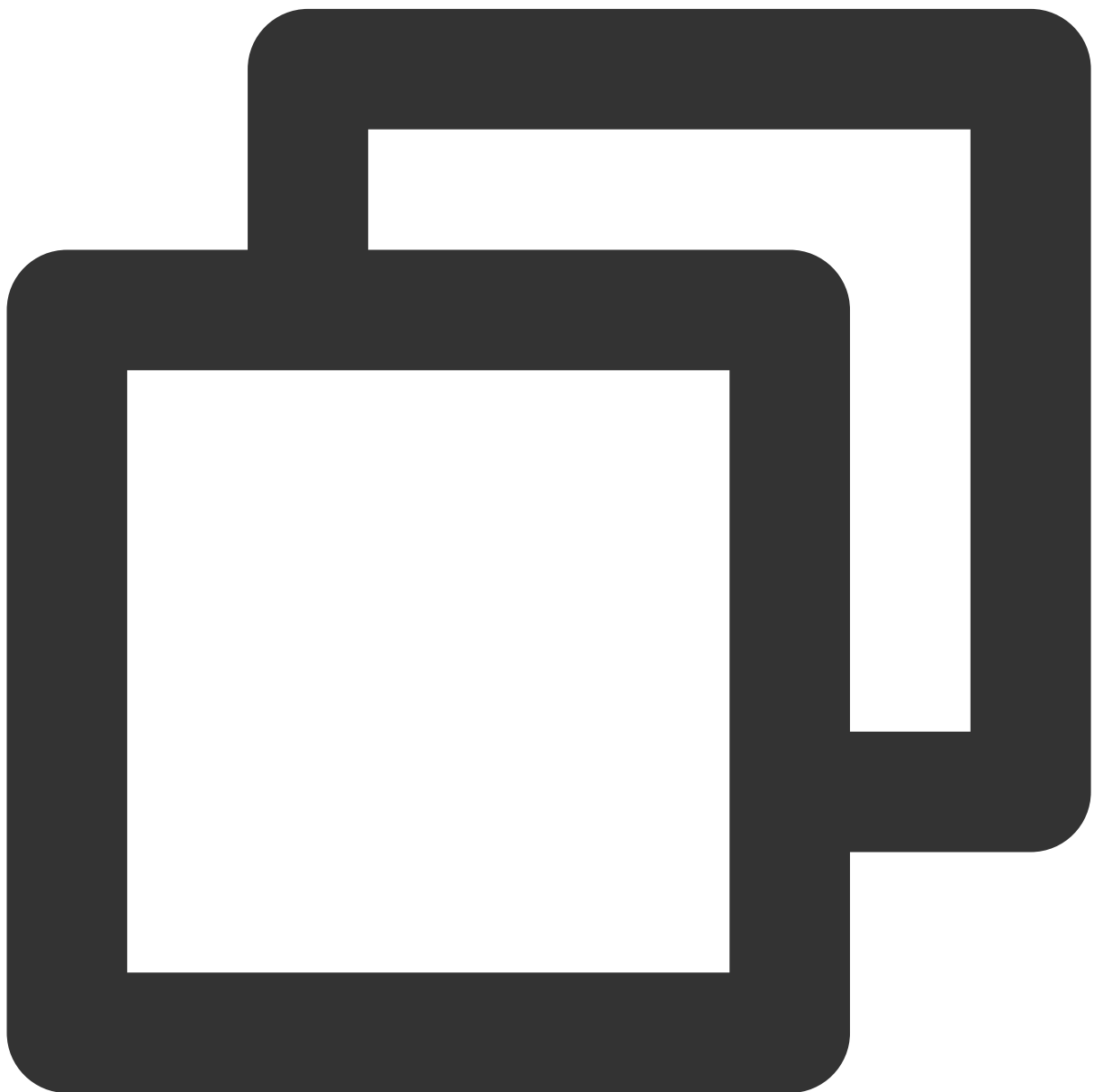
The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| roomId       | String           | room ID                                              |

## onRoomInfoChange

Callback for change of room information. This callback is usually used to notify users of room status change in co-anchoring and cross-room communication scenarios.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
    onRoomInfoChange:(TRTCLiveRoomInfo *)info
NS_SWIFT_NAME(trtcLiveRoom(_:onRoomInfoChange:));
```

The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| info         | TRTCLiveRoomInfo | room information                                     |

## Callback APIs for Entry/Exit of Anchors/Audience

### onAnchorEnter

Callback for a new anchor/co-anchoring viewer. Audience will receive this callback when there is a new anchor/co-anchoring viewer in the room and can call `startPlay()` of `TRTCLiveRoom` to play the video of the anchor/co-anchoring viewer.





```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
    onAnchorEnter:(NSString *)userID
NS_SWIFT_NAME(trtcLiveRoom(_:onAnchorEnter:));
```

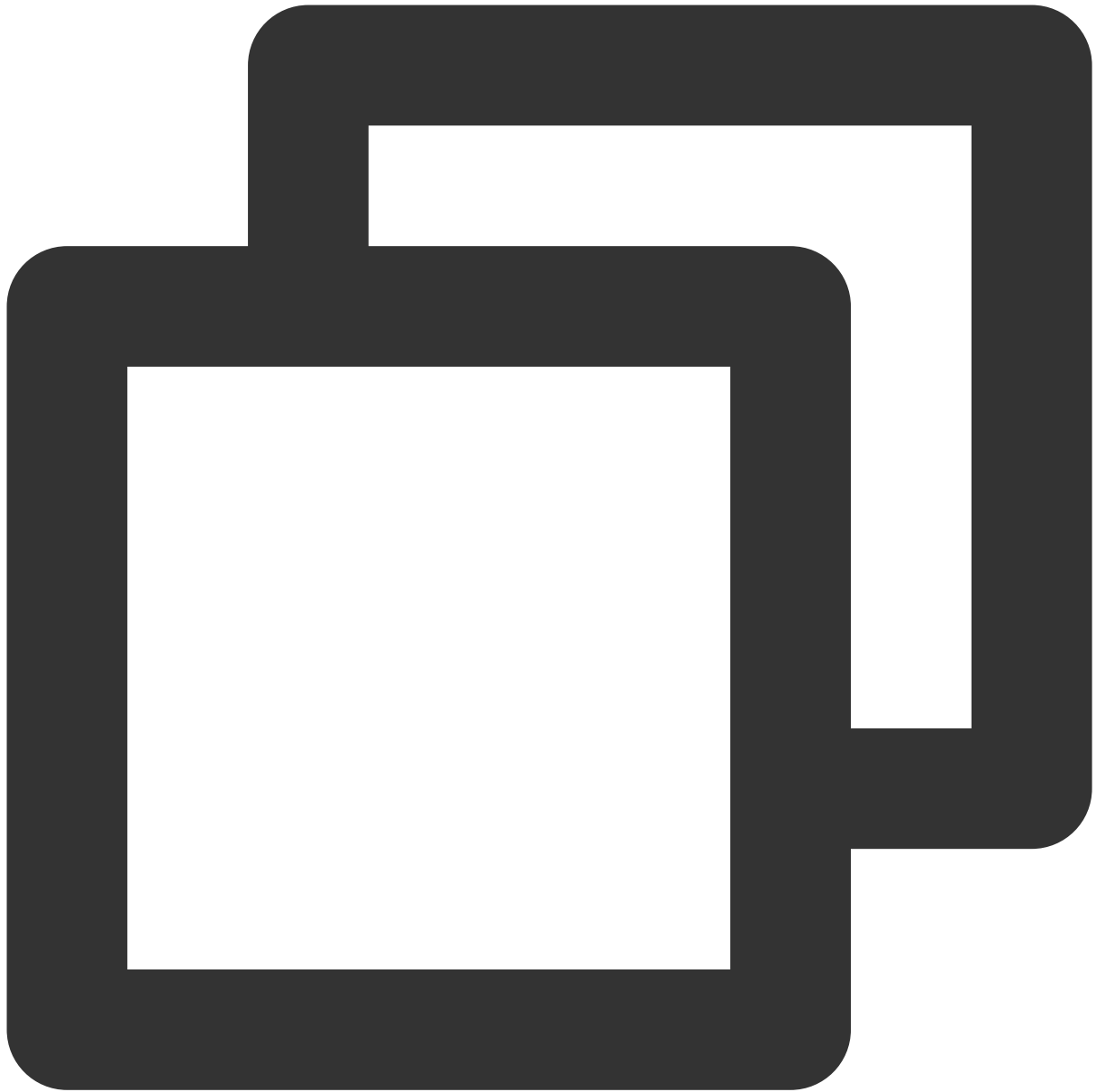
The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| userID       | String           | User ID of the new anchor/co-anchoring viewer        |

## onAnchorExit

Callback for the quitting of an anchor/co-anchoring viewer. The anchors and co-anchoring viewers in a room will receive this callback after an anchor/co-anchoring viewer quits cross-room communication/co-anchoring and can call

`stopPlay()` of `TRTCLiveRoom` to stop playing the video of the anchor/co-anchoring viewer.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
    onAnchorExit:(NSString *)userID
NS_SWIFT_NAME(trtcLiveRoom(_:onAnchorExit:));
```

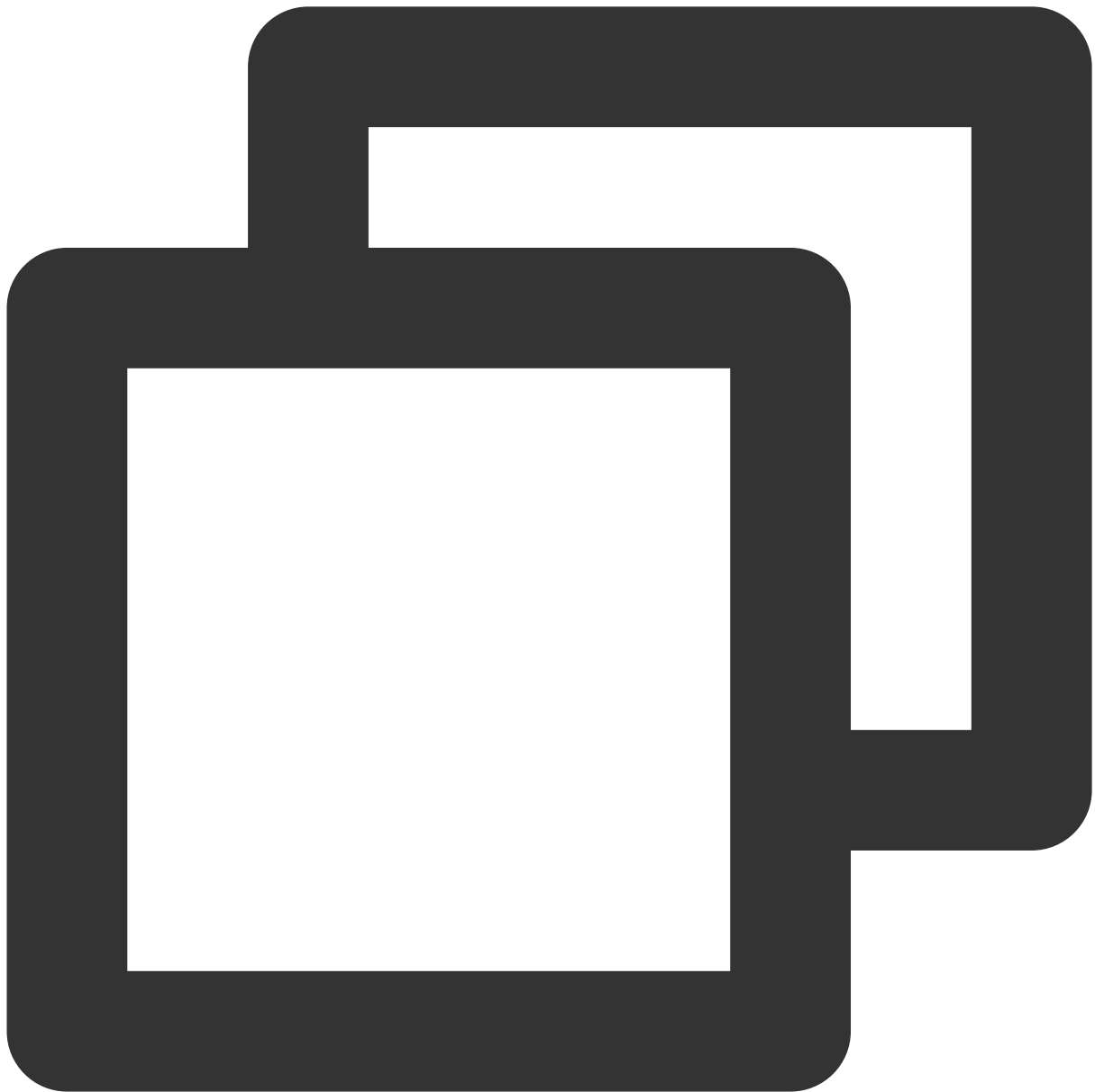
The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| userID       | String           | User ID of the anchor/co-anchoring viewer            |

## onAudienceEnter

Callback for room entry by a viewer.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
```

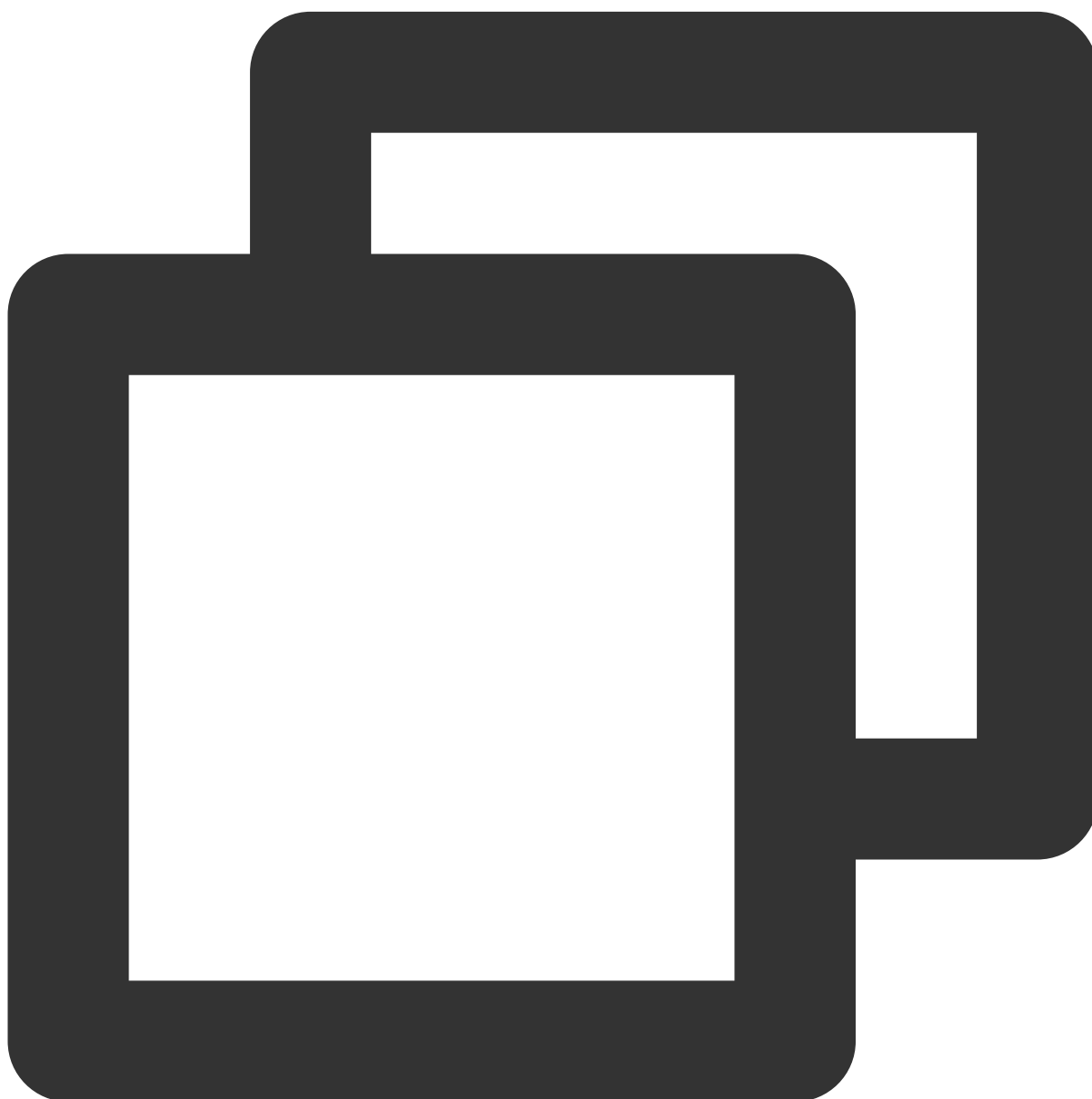
```
onAudienceEnter:(TRTCLiveUserInfo *)user  
NS_SWIFT_NAME(trtcLiveRoom(_:onAudienceEnter:));
```

The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| user         | TRTCLiveUserInfo | Information of the user who entered the room         |

## onAudienceExit

Callback for room exit by a viewer.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
    onAudienceExit:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onAudienceExit:));
```

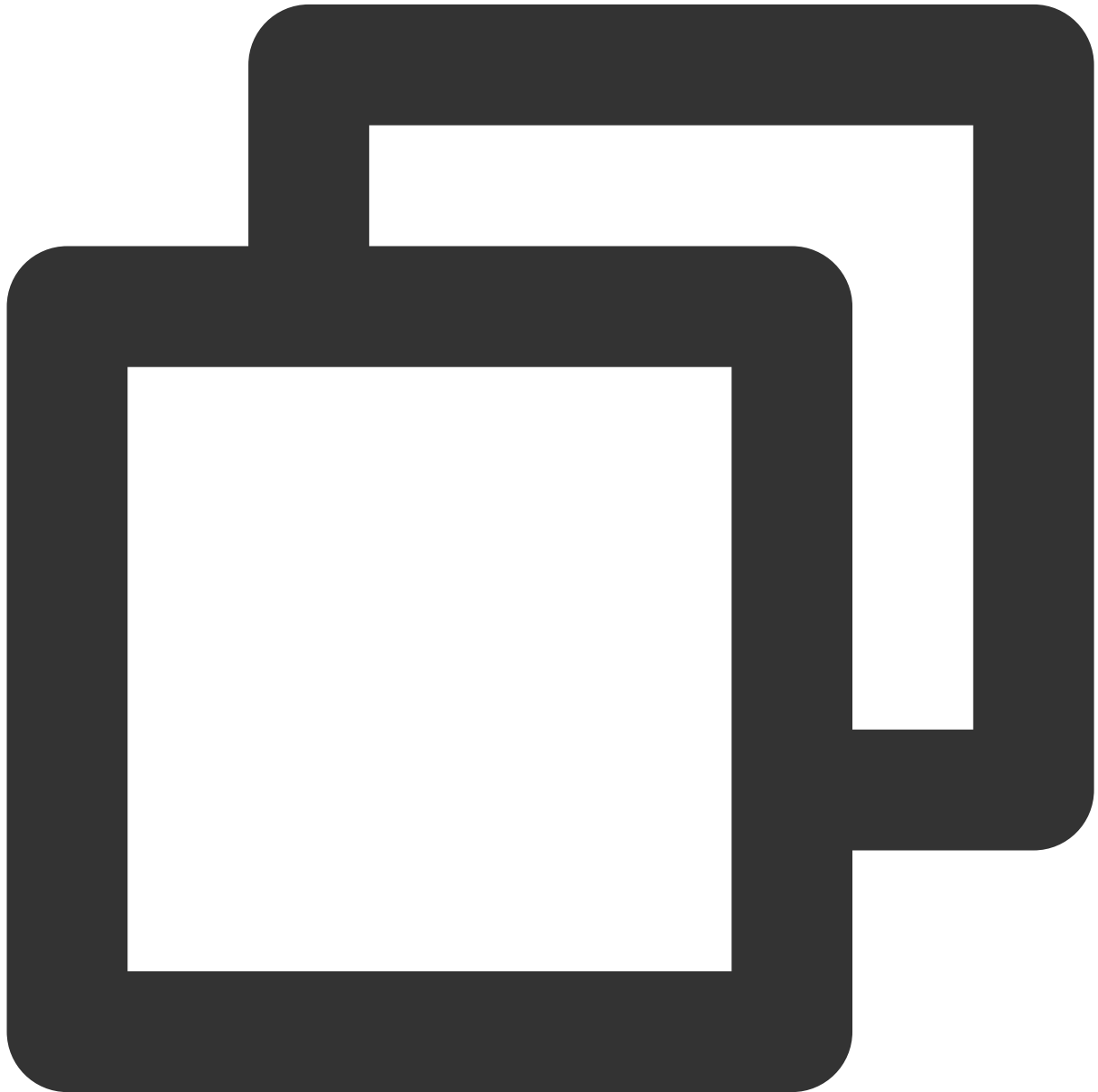
The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| user         | TRTCLiveUserInfo | Information of the user who left the room            |

# Callback APIs Audience-Anchor Co-anchoring Events

## **onRequestJoinAnchor**

Callback for receiving a co-anchoring request from a viewer.



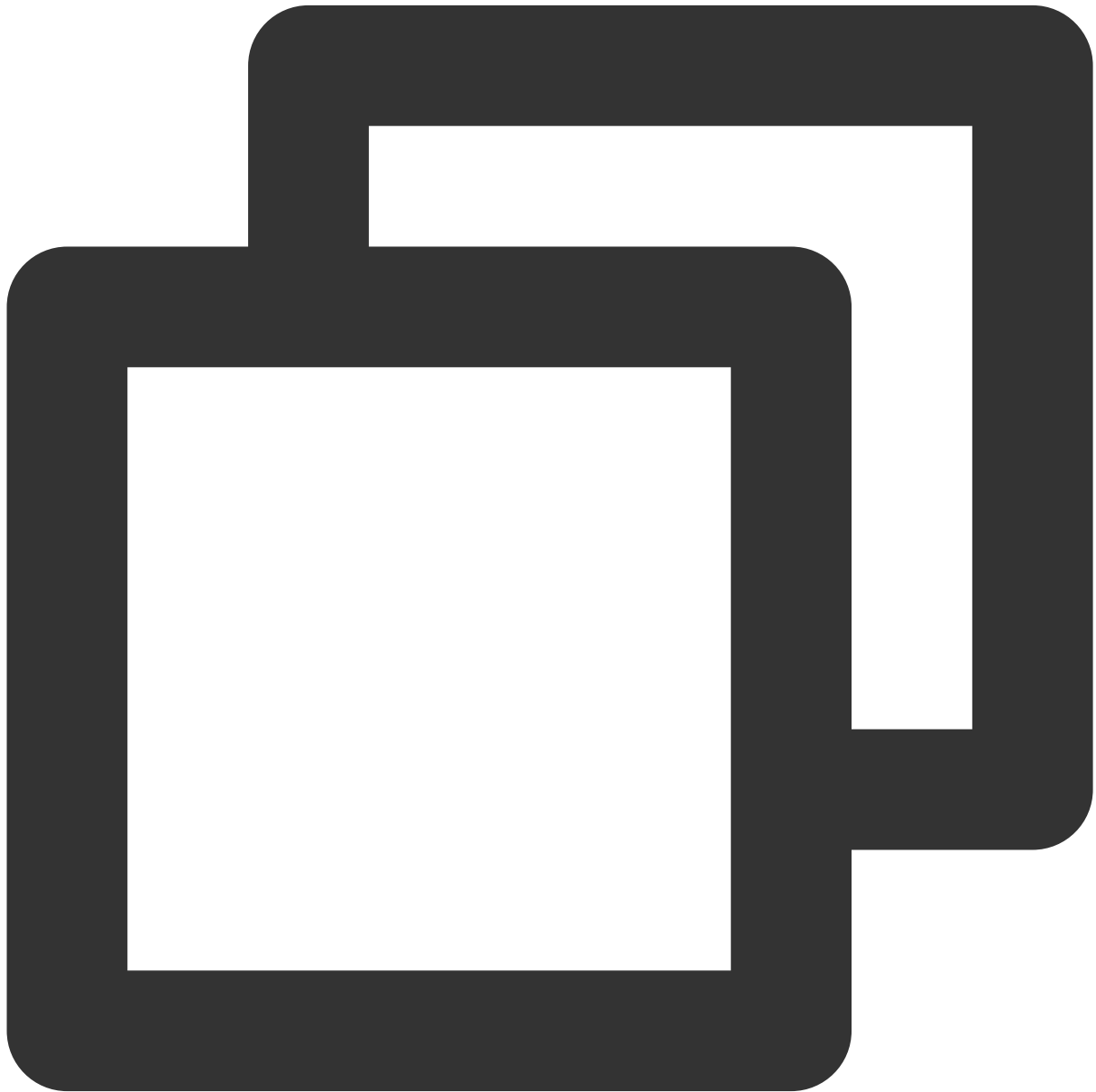
```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRequestJoinAnchor:(TRTCLiveUserInfo *)user
    reason:(NSString * _Nullable)reason
    timeout:(double)timeout
NS_SWIFT_NAME(trtcLiveRoom(_:onRequestJoinAnchor:reason:timeout:));
```

The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| user         | TRTCLiveUserInfo | Information of the user who requested co-anchoring   |
| reason       | String?          | Reason for co-anchoring                              |
| timeout      | Double           | Timeout period for response from the anchor          |

### onKickoutJoinAnchor

Callback for being removed from co-anchoring. After receiving this callback, a co-anchoring viewer needs to call `stopPublish()` of `TRTCLiveRoom` to quit co-anchoring.



```
- (void)trtcLiveRoomOnKickoutJoinAnchor:(TRTCLiveRoom *)liveRoom  
NS_SWIFT_NAME(trtcLiveRoomOnKickoutJoinAnchor(_:));
```

The parameters are described below:

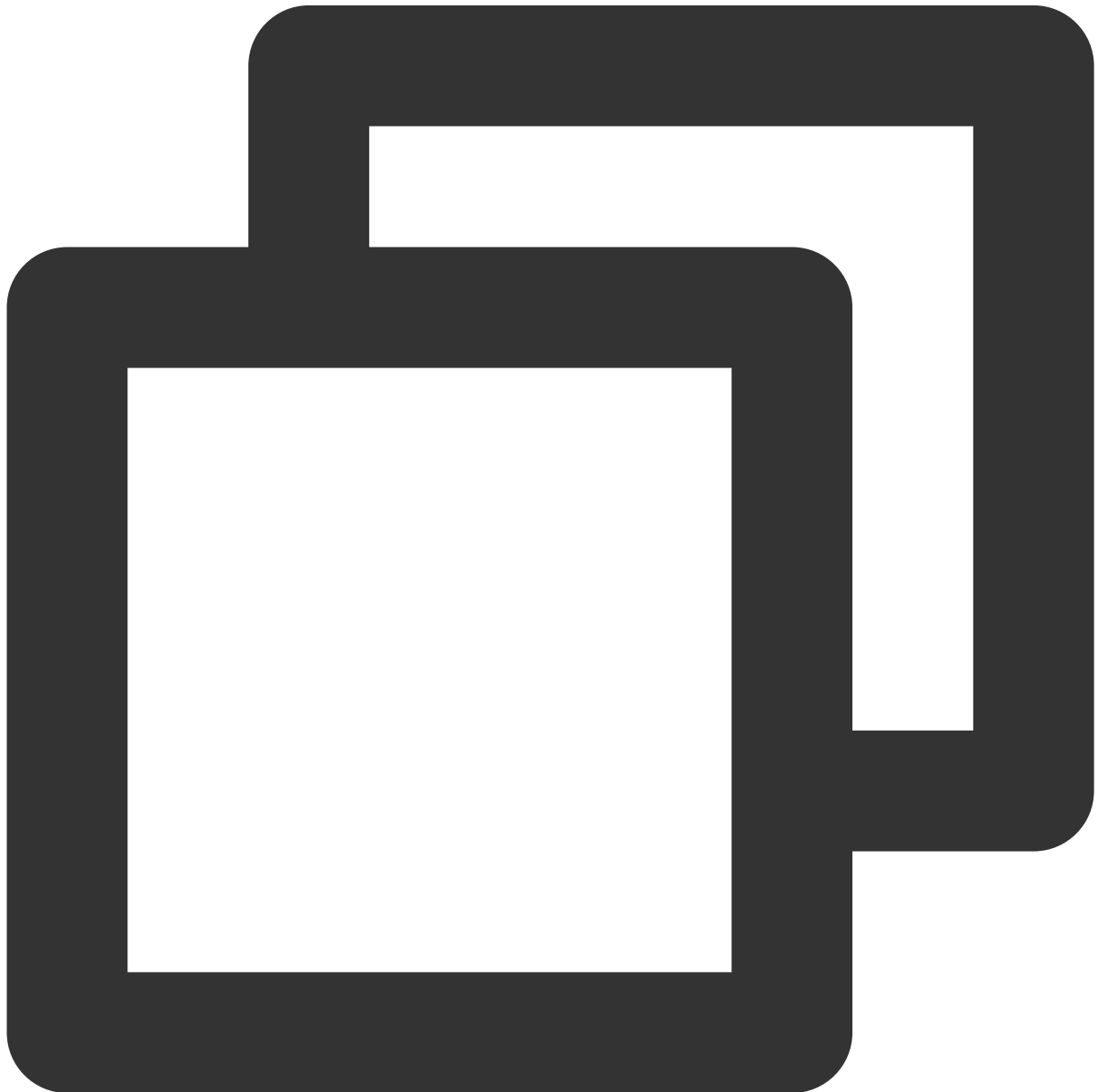
| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |



## Callback APIs for Cross-Room Communication Events

### **onRequestRoomPK**

Callback for receiving a cross-room communication request. If an anchor accepts the request, he or she should wait for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and call `startPlay()` to play the other anchor's video.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom  
  onRequestRoomPK:(TRTCLiveUserInfo *)user
```

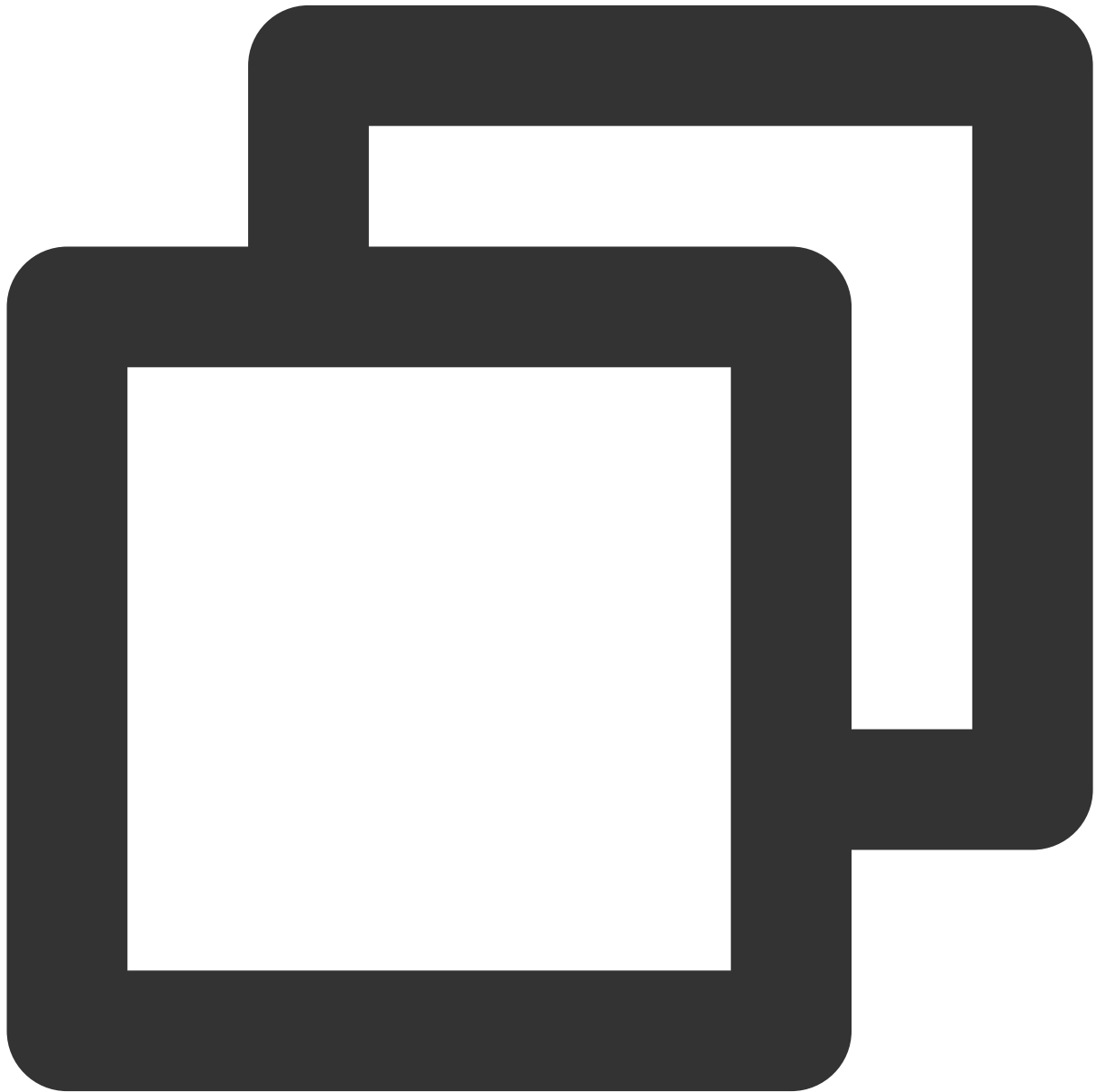
```
timeout:(double)timeout
NS_SWIFT_NAME(trtcLiveRoom(_:onRequestRoomPK:timeout:));
```

The parameters are described below:

| Parameter    | Type             | Description                                            |
|--------------|------------------|--------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance   |
| user         | TRTCLiveUserInfo | Information of the anchor who requested communication. |
| timeout      | Double           | Timeout period for response from the anchor            |

## onQuitRoomPK

Callback for ending cross-room communication.



```
- (void)trtcLiveRoomOnQuitRoomPK:(TRTCLiveRoom *)liveRoom  
NS_SWIFT_NAME(trtcLiveRoomOnQuitRoomPK(_:));
```

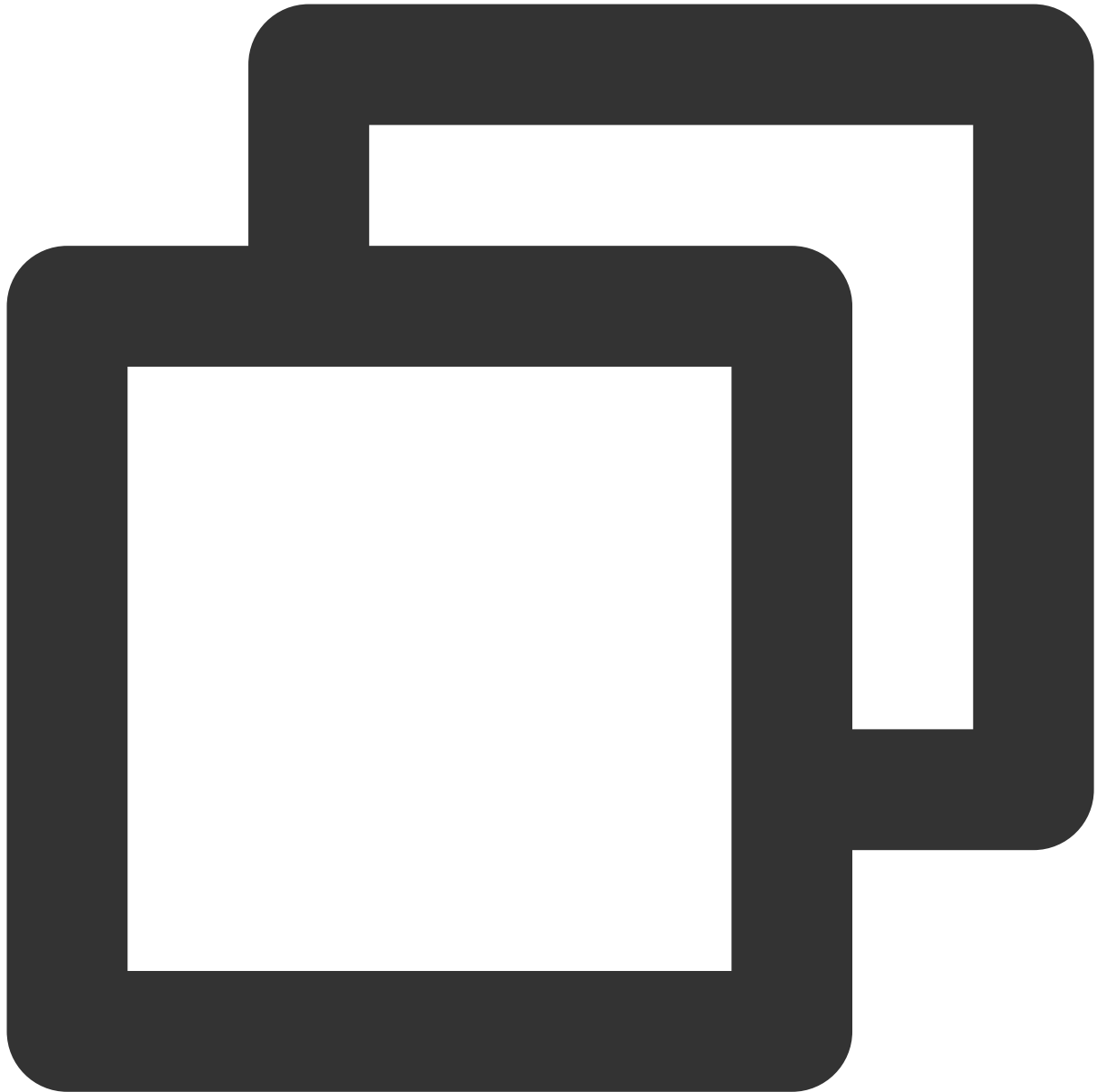
The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |

# Message Event Callback APIs

## onRecvRoomTextMsg

Callback for receiving a text chat message.



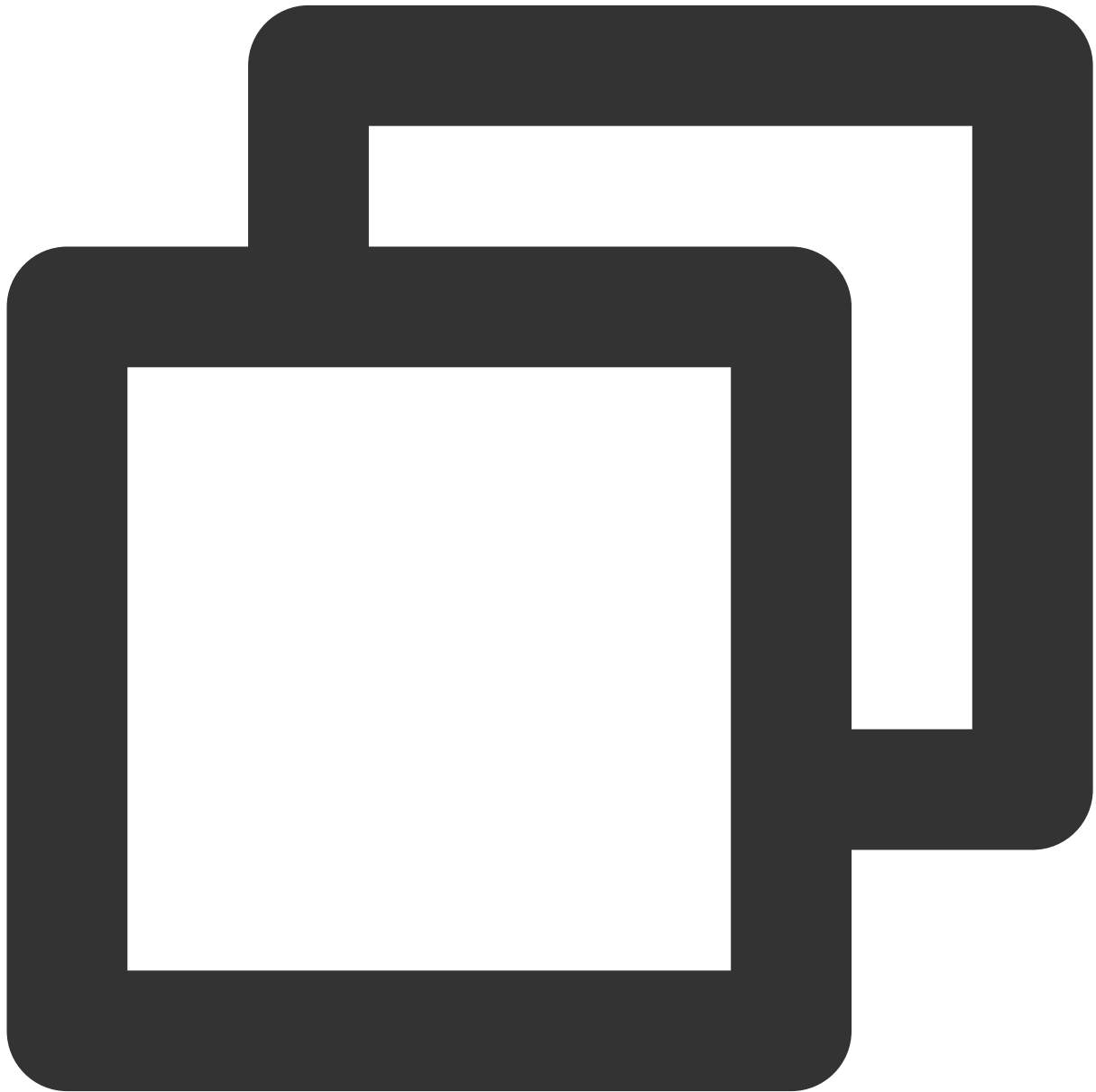
```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom  
  onRecvRoomTextMsg:(NSString *)message  
    fromUser:(TRTCLiveUserInfo *)user  
NS_SWIFT_NAME(trtcLiveRoom(_:onRecvRoomTextMsg:fromUser:));
```

The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |
| message      | String           | Text message                                         |
| user         | TRTCLiveUserInfo | Information of the sender                            |

### **onRecvRoomCustomMsg**

A custom message was received.



```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRecvRoomCustomMsgWithCommand:(NSString *)command
    message:(NSString *)message
    fromUser:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onRecvRoomCustomMsg:message:fromUser:));
```

The parameters are described below:

| Parameter    | Type             | Description                                          |
|--------------|------------------|------------------------------------------------------|
| trtcLiveRoom | TRTCLiveRoomImpl | Current <code>TRTCLiveRoom</code> component instance |

|         |                  |                                                                         |
|---------|------------------|-------------------------------------------------------------------------|
| command | String           | Custom command word used to distinguish between different message types |
| message | String           | Text message                                                            |
| user    | TRTCLiveUserInfo | Information of the sender                                               |

# TRTCLiveRoom APIs (Android)

Last updated : 2024-03-11 10:27:28

`TRTCLiveRoom` includes the following features which are based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat.

A user can create a room and start live streaming, or enter a room as an audience member.

The anchor can co-anchor with audience members in the room.

Anchors in two rooms can compete and interact with each other.

All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

## Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see [Pricing](#).

`TRTCLiveRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see [Interactive Live Video Streaming \(Android\)](#).

The [TRTC SDK](#) is used as a low-latency audio chat component.

The `AVChatRoom` feature of the [Chat SDK](#) is used to implement chat rooms. The attribute APIs of Chat are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

## TRTCLiveRoom API Overview

### Basic SDK APIs

| API                                   | Description                                |
|---------------------------------------|--------------------------------------------|
| <a href="#">sharedInstance</a>        | Gets a singleton object.                   |
| <a href="#">destroySharedInstance</a> | Terminates a singleton object.             |
| <a href="#">setDelegate</a>           | Sets event callbacks.                      |
| <a href="#">setDelegateHandler</a>    | Sets the thread where event callbacks are. |
| <a href="#">login</a>                 | Logs in.                                   |
| <a href="#">logout</a>                | Logs out.                                  |
|                                       |                                            |



[setSelfProfile](#)

Sets the profile.

## Room APIs

| API                             | Description                                                                                                                  |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">createRoom</a>      | Creates a room (called by anchor). If the room does not exist, the system will create the room automatically.                |
| <a href="#">destroyRoom</a>     | Terminates a room (called by anchor).                                                                                        |
| <a href="#">enterRoom</a>       | Enters a room (called by audience).                                                                                          |
| <a href="#">exitRoom</a>        | Exits a room (called by audience).                                                                                           |
| <a href="#">getRoomInfos</a>    | Gets room list details.                                                                                                      |
| <a href="#">getAnchorList</a>   | Gets the anchors in a room. This API works only if it is called after <code>enterRoom()</code> .                             |
| <a href="#">getAudienceList</a> | Gets the information of all audience members in a room. This API works only if it is called after <code>enterRoom()</code> . |

## Stream pushing/pulling APIs

| API                                | Description                                                                                 |
|------------------------------------|---------------------------------------------------------------------------------------------|
| <a href="#">startCameraPreview</a> | Enables preview of the local video.                                                         |
| <a href="#">stopCameraPreview</a>  | Stops local video capturing and preview.                                                    |
| <a href="#">startPublish</a>       | Starts live streaming (pushing streams).                                                    |
| <a href="#">stopPublish</a>        | Stops live streaming (pushing streams).                                                     |
| <a href="#">startPlay</a>          | Plays a remote video. This API can be called in common playback and co-anchoring scenarios. |
| <a href="#">stopPlay</a>           | Stops rendering a remote video.                                                             |

## APIs for anchor-audience co-anchoring

| API                                | Description                                            |
|------------------------------------|--------------------------------------------------------|
| <a href="#">requestJoinAnchor</a>  | Requests co-anchoring (called by audience).            |
| <a href="#">responseJoinAnchor</a> | Responds to a co-anchoring request (called by anchor). |
|                                    |                                                        |

[kickoutJoinAnchor](#)

Removes a user from co-anchoring (called by anchor).

## APIs for cross-room communication

| API                            | Description                                                        |
|--------------------------------|--------------------------------------------------------------------|
| <a href="#">requestRoomPK</a>  | Sends a cross-room communication request (called by anchor).       |
| <a href="#">responseRoomPK</a> | Responds to a cross-room communication request (called by anchor). |
| <a href="#">quitRoomPK</a>     | Quits cross-room communication.                                    |

## Audio/Video APIs

| API                                | Description                                  |
|------------------------------------|----------------------------------------------|
| <a href="#">switchCamera</a>       | Switches between the front and rear cameras. |
| <a href="#">setMirror</a>          | Specifies whether to mirror video.           |
| <a href="#">muteLocalAudio</a>     | Mutes/Unmutes the local user.                |
| <a href="#">muteRemoteAudio</a>    | Mutes/Unmutes a remote user.                 |
| <a href="#">muteAllRemoteAudio</a> | Mutes/Unmutes all remote users.              |

## Background music and audio effect APIs

| API                                   | Description                                                                                         |
|---------------------------------------|-----------------------------------------------------------------------------------------------------|
| <a href="#">getAudioEffectManager</a> | Gets the background music and audio effect management object <a href="#">TXAudioEffectManager</a> . |

## Beauty filter APIs

| API                              | Description                                                                |
|----------------------------------|----------------------------------------------------------------------------|
| <a href="#">getBeautyManager</a> | Gets the beauty filter management object <a href="#">TXBeautyManager</a> . |

## Message sending APIs

| API                             | Description                                                                                  |
|---------------------------------|----------------------------------------------------------------------------------------------|
| <a href="#">sendRoomTextMsg</a> | Broadcasts a text chat message in a room. This API is generally used for on-screen comments. |

|                                   |                              |
|-----------------------------------|------------------------------|
| <a href="#">sendRoomCustomMsg</a> | Sends a custom text message. |
|-----------------------------------|------------------------------|

## Debugging APIs

| API                               | Description                                                   |
|-----------------------------------|---------------------------------------------------------------|
| <a href="#">showVideoDebugLog</a> | Specifies whether to display debugging information on the UI. |

# TRTCLiveRoomDelegate API Overview

## Common event callbacks

| API                        | Description           |
|----------------------------|-----------------------|
| <a href="#">onError</a>    | Callback for error.   |
| <a href="#">onWarning</a>  | Callback for warning. |
| <a href="#">onDebugLog</a> | Callback of log.      |

## Room event callback APIs

| API                              | Description                   |
|----------------------------------|-------------------------------|
| <a href="#">onRoomDestroy</a>    | The room was terminated.      |
| <a href="#">onRoomInfoChange</a> | The room information changed. |

## Callback APIs for entry/exit of anchors/audience

| API                             | Description                          |
|---------------------------------|--------------------------------------|
| <a href="#">onAnchorEnter</a>   | There is a new anchor in the room.   |
| <a href="#">onAnchorExit</a>    | An anchor left the room.             |
| <a href="#">onAudienceEnter</a> | An audience member entered the room. |
| <a href="#">onAudienceExit</a>  | An audience member left the room.    |

## Callback APIs for anchor-audience co-anchoring events

| API                                 | Description                           |
|-------------------------------------|---------------------------------------|
| <a href="#">onRequestJoinAnchor</a> | A co-anchoring request was received.  |
| <a href="#">onKickoutJoinAnchor</a> | A user was removed from co-anchoring. |

### Callback APIs for cross-room communication events

| API                             | Description                                      |
|---------------------------------|--------------------------------------------------|
| <a href="#">onRequestRoomPK</a> | A cross-room communication request was received. |
| <a href="#">onQuitRoomPK</a>    | Cross-room communication ended.                  |

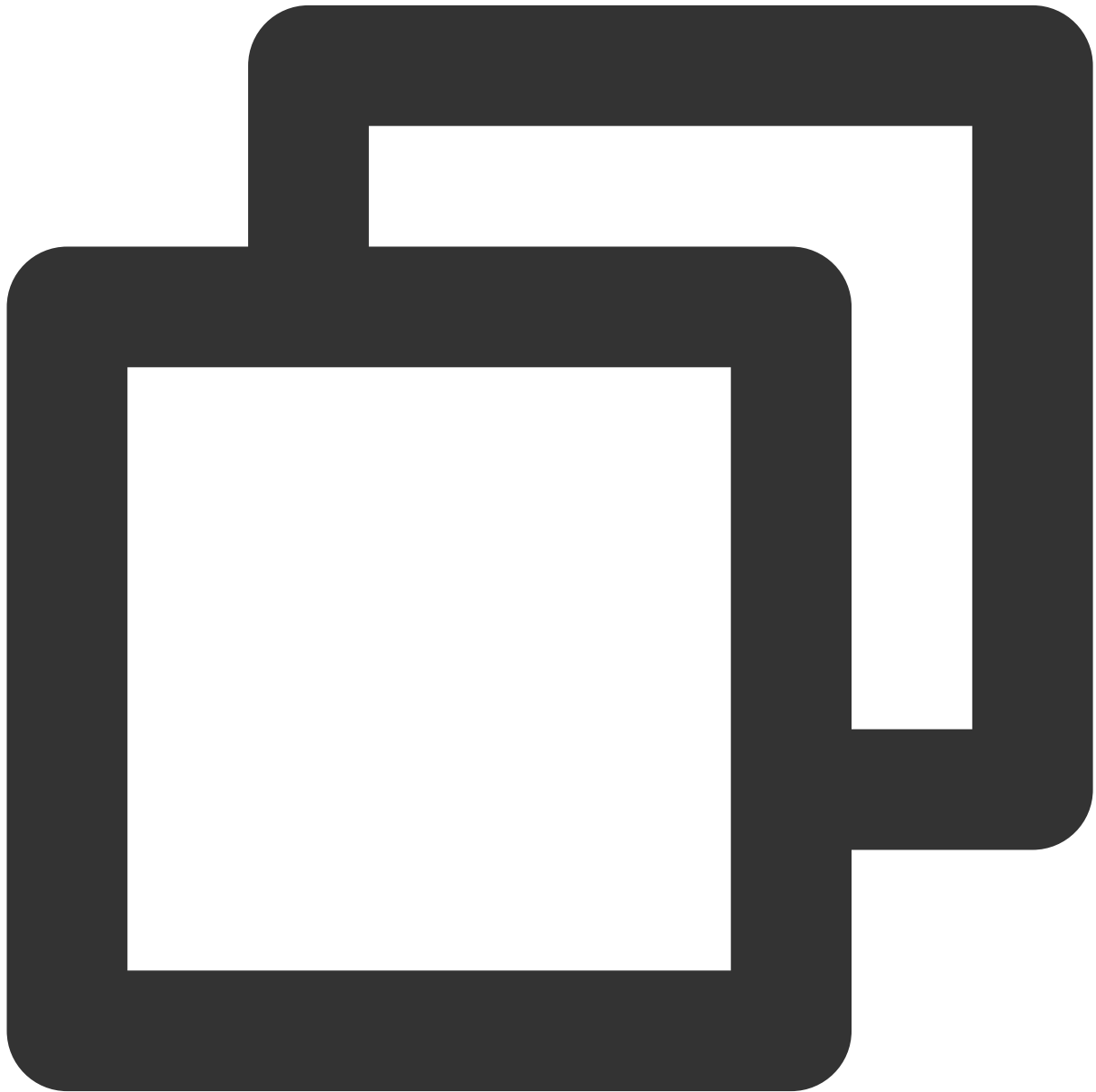
### Message event callback APIs

| API                                 | Description                       |
|-------------------------------------|-----------------------------------|
| <a href="#">onRecvRoomTextMsg</a>   | A text chat message was received. |
| <a href="#">onRecvRoomCustomMsg</a> | A custom message was received.    |

## Basic SDK APIs

### sharedInstance

This API is used to get a [TRTCLiveRoom](#) singleton object.



```
public static synchronized TRTCLiveRoom sharedInstance(Context context);
```

The parameters are described below:

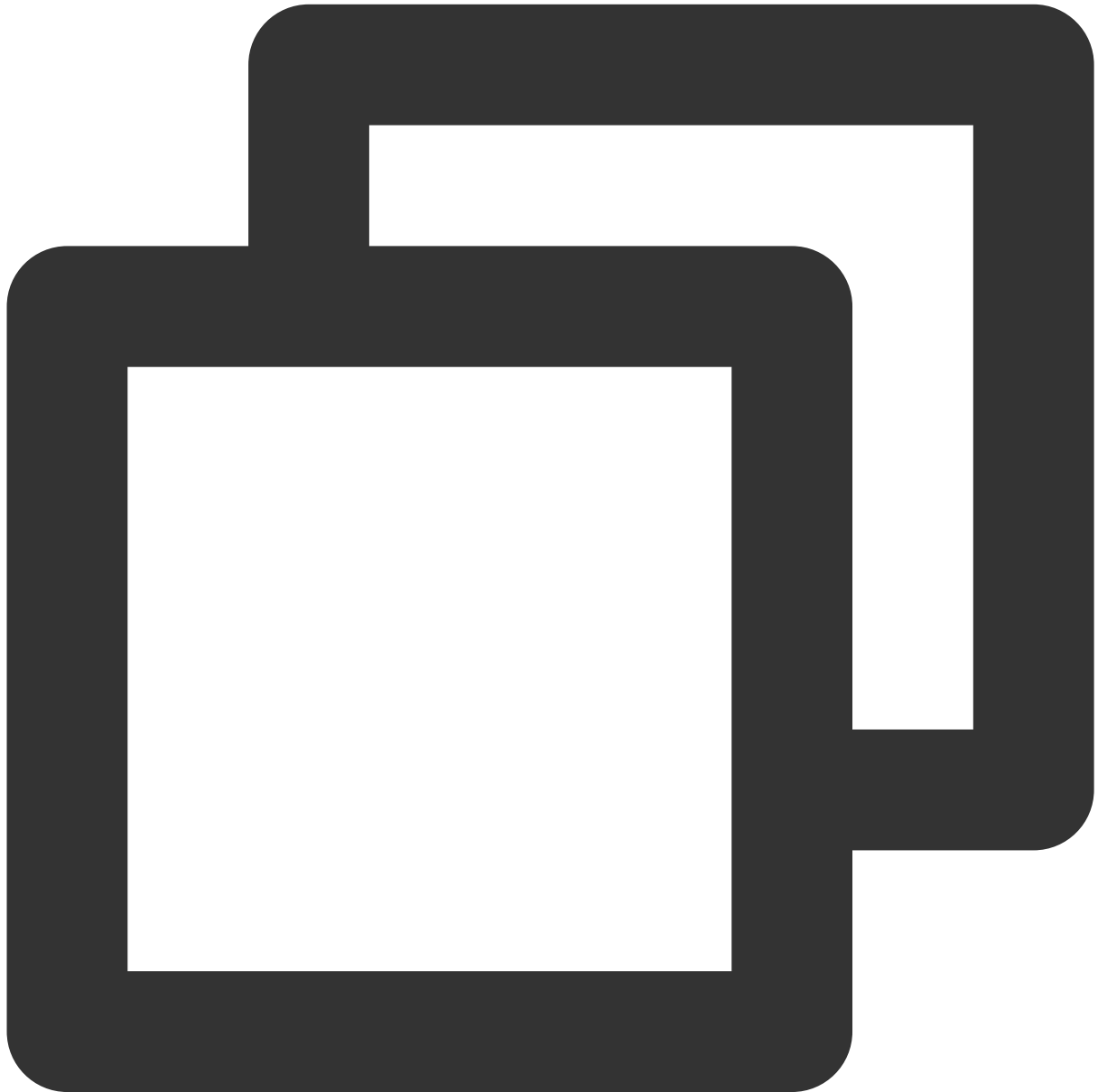
| Parameter | Type    | Description                                                                                                |
|-----------|---------|------------------------------------------------------------------------------------------------------------|
| context   | Context | Android context, which will be converted to <code>ApplicationContext</code> for the calling of system APIs |

## **destroySharedInstance**

This API is used to terminate the [TRTCLiveRoom](#) singleton object.

**Note**

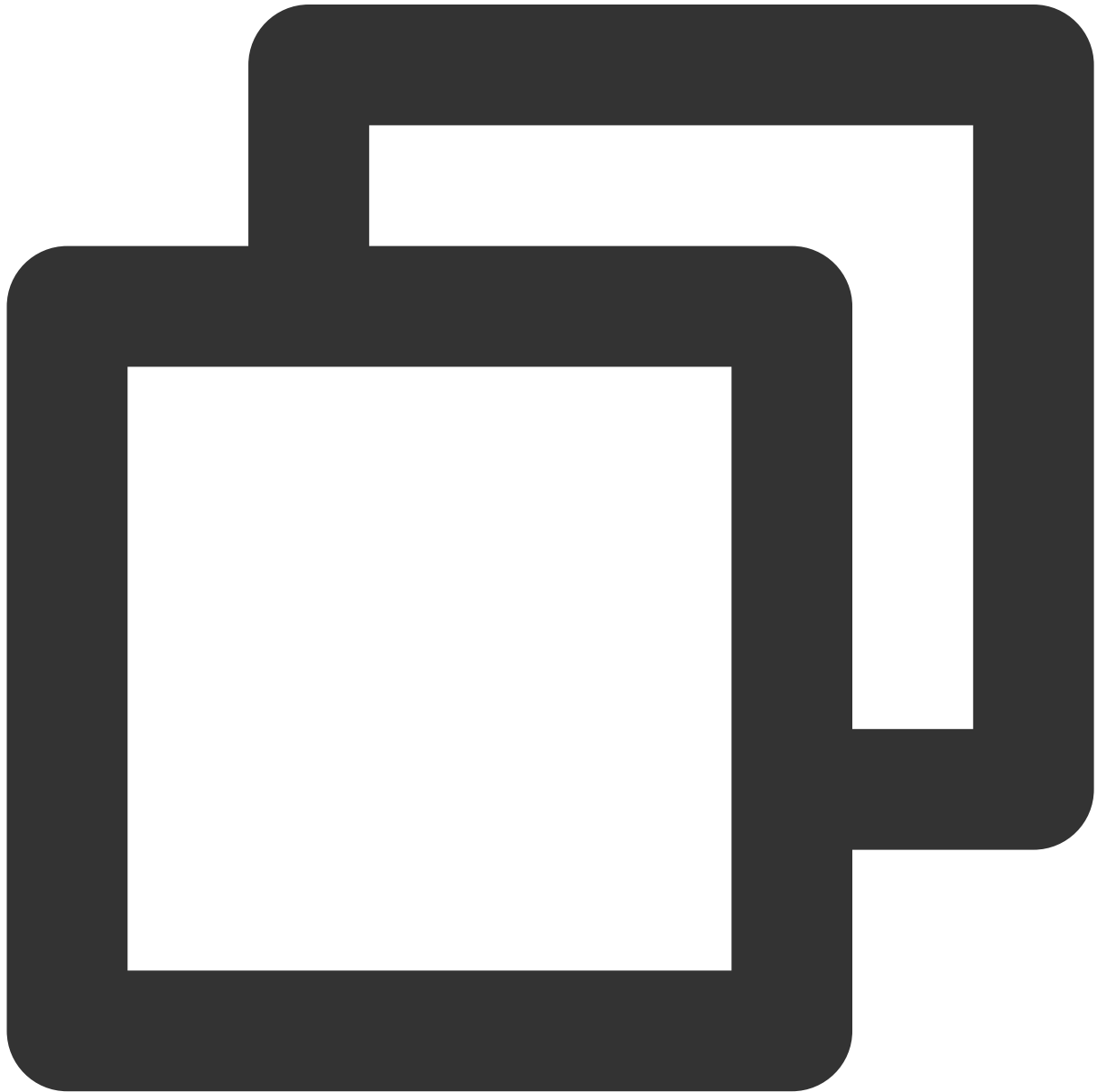
After the instance is terminated, the externally cached `TRTCLiveRoom` instance can no longer be used. You need to call [sharedInstance](#) again to get a new instance.



```
public static void destroySharedInstance();
```

**setDelegate**

This API is used to get callbacks for the events of [TRTCLiveRoom](#). You can use `TRTCLiveRoomDelegate` to get the callbacks.



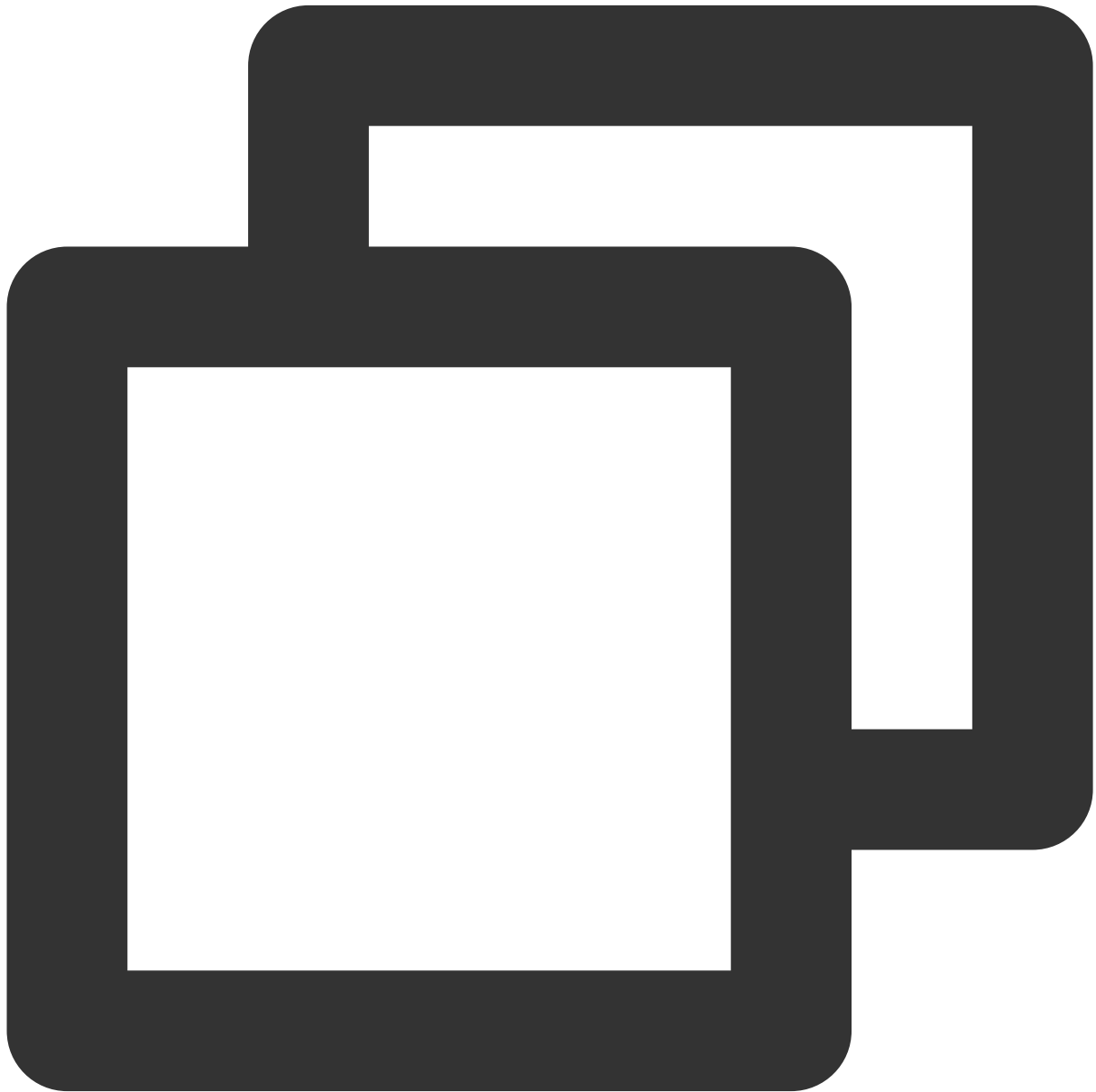
```
public abstract void setDelegate(TRTCLiveRoomDelegate delegate);
```

#### Note

`setDelegate` is the delegate callback of `TRTCLiveRoom`.

#### setDelegateHandler

This API is used to set the thread where event callbacks are.



```
public abstract void setDelegateHandler(Handler handler);
```

The parameters are described below:

| Parameter | Type    | Description                                                                                                                                 |
|-----------|---------|---------------------------------------------------------------------------------------------------------------------------------------------|
| handler   | Handler | Callbacks for the events of <code>TRTCLiveRoom</code> are returned via this handler. Do not use it together with <code>setDelegate</code> . |

## login



Login



```
public abstract void login(int sdkAppId,  
    String userId, String userSig,  
    TRTCLiveRoomDef.TRTCLiveRoomConfig config,  
    TRTCLiveRoomCallback.ActionCallback callback);
```

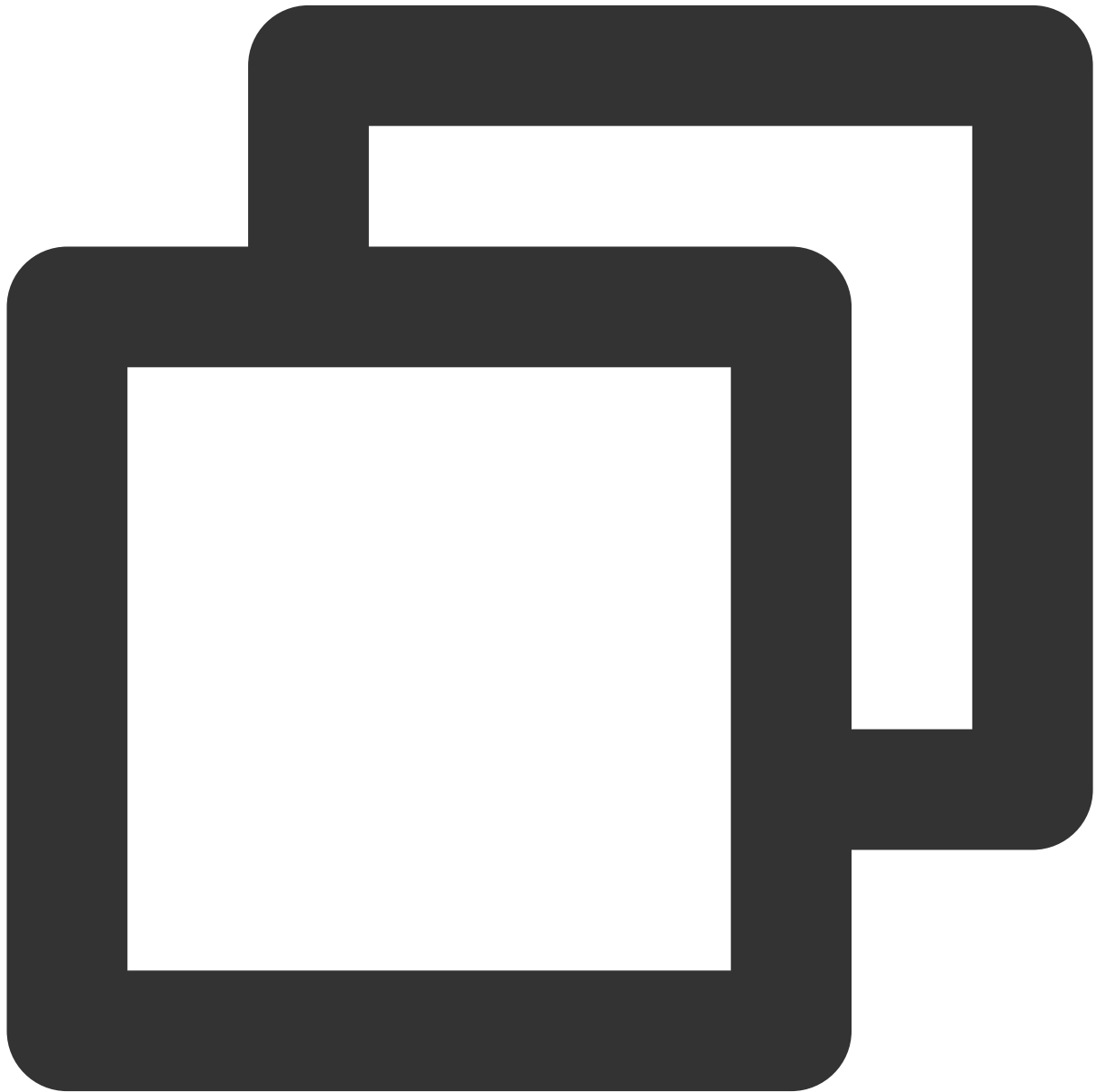
The parameters are described below:

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

|          |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sdkAppId | int                | You can view <code>SDKAppID</code> in <a href="#">Application Management</a> > <b>Application Info</b> of the TRTC console.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| userId   | String             | The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| userSig  | String             | Tencent Cloud's proprietary security signature. For how to calculate and use it, see <a href="#">FAQs &gt; UserSig</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| config   | TRTCLiveRoomConfig | <p>Global configuration information, which needs to be initialized during login and cannot be modified afterward.</p> <p><code>useCDNFirst</code> : Specifies the way the audience watches live streams. <code>true</code> means that the audience watches live streams over CDNs, which is cost-efficient but has high latency. <code>false</code> means that the audience watches live streams in the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is within 1 second.</p> <p><code>CDNPlayDomain</code> : Specifies the domain name for CDN live streaming. It takes effect only if <code>useCDNFirst</code> is set to <code>true</code> . You can set it in <a href="#">Domain Management</a> of the CSS console.</p> |
| callback | ActionCallback     | The callback for login. The code is <code>0</code> if login succeeds.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## logout

Log out



```
public abstract void logout(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are described below:

| Parameter | Type           | Description                                            |
|-----------|----------------|--------------------------------------------------------|
| callback  | ActionCallback | Callback for logout. The code is 0 if logout succeeds. |

## setSelfProfile

This API is used to set the profile.



```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCLiveRoom
```

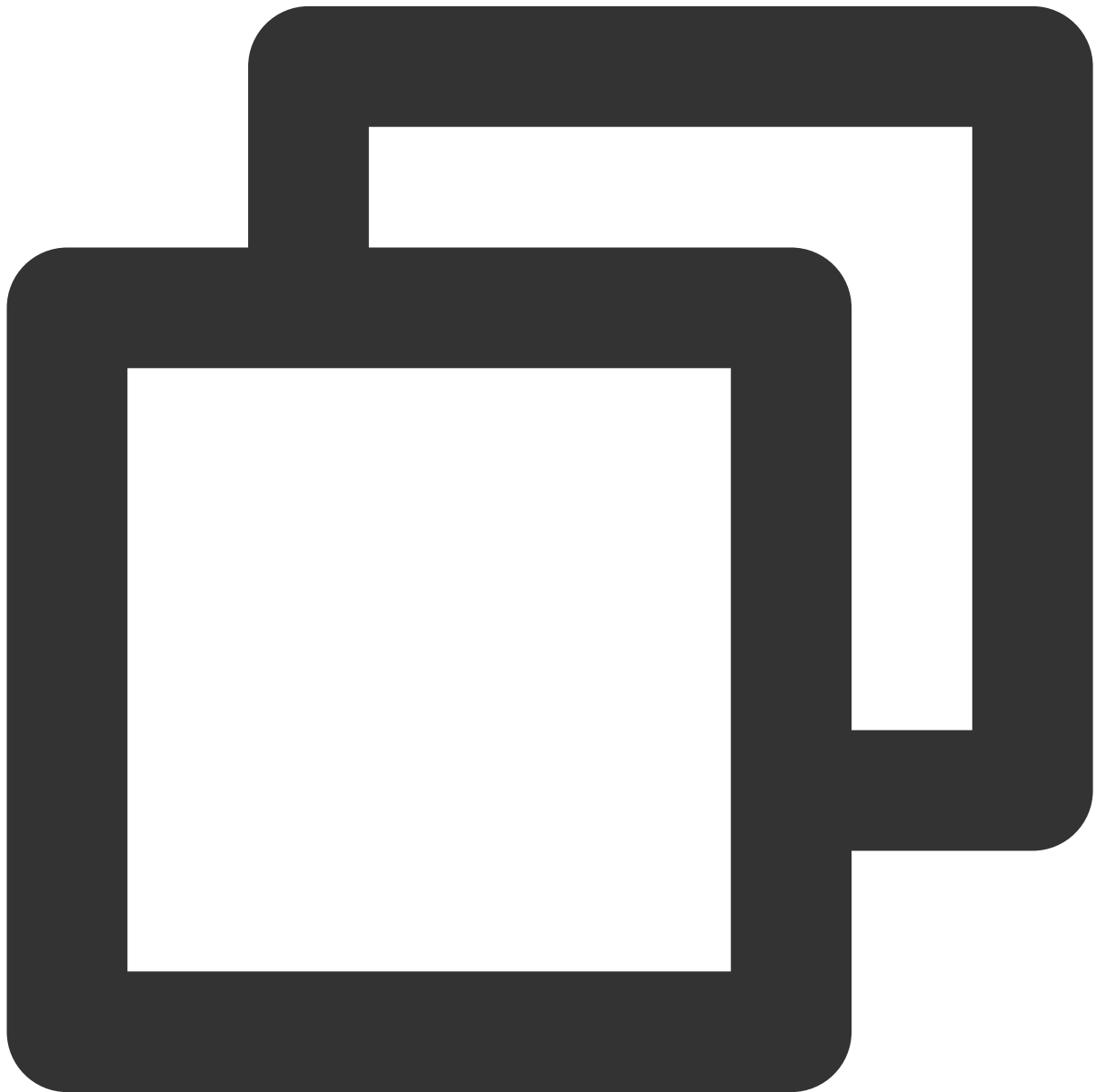
The parameters are described below:

| Parameter | Type           | Description                                                            |
|-----------|----------------|------------------------------------------------------------------------|
| userName  | String         | Username                                                               |
| avatarURL | String         | Profile photo URL                                                      |
| callback  | ActionCallback | Callback for profile setting. The code is 0 if the operation succeeds. |

## Room APIs

### createRoom

This API is used to create a room (called by anchor).



```
public abstract void createRoom(int roomId, TRTCLiveRoomDef.TRTCCreateRoomParam roomDef)
```

The parameters are described below:

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

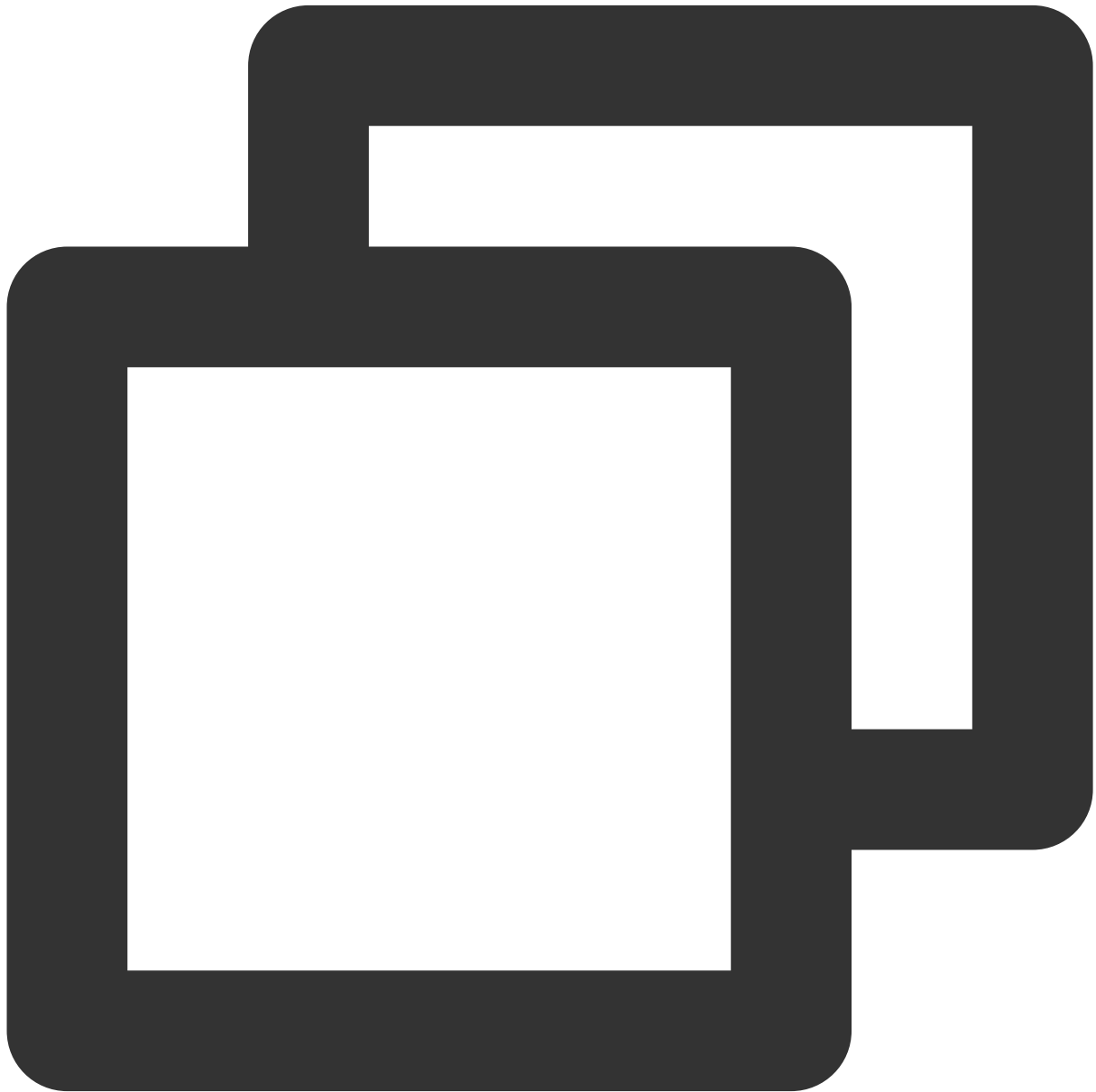
|           |                     |                                                                                                                                                                                                                                                                 |
|-----------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| roomId    | int                 | The room ID. You need to assign and manage the IDs in a centralized manner. Multiple <code>roomId</code> values can be aggregated into a live room list. Currently, Tencent Cloud does not provide list management services. Please manage your own room lists. |
| roomParam | TRTCCreateRoomParam | Room information, such as room name and cover information. If both the room list and room information are managed on your server, you can ignore this parameter.                                                                                                |
| callback  | ActionCallback      | Callback for room creation. The code is 0 if the operation succeeds.                                                                                                                                                                                            |

The process of creating a room and starting live streaming as an anchor is as follows:

1. A user calls `startCameraPreview()` to enable camera preview and set beauty filters.
2. The user calls `createRoom()` to create a room, the result of which is returned via the `ActionCallback` callback.
3. The user calls `starPublish()` to push streams.

## destroyRoom

This API is used to terminate a room (called by anchor).



```
public abstract void destroyRoom(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are described below:

| Parameter | Type           | Description                                                             |
|-----------|----------------|-------------------------------------------------------------------------|
| callback  | ActionCallback | Callback for room termination. The code is 0 if the operation succeeds. |

## enterRoom

This API is used to enter a room (called by audience).



```
public abstract void enterRoom(int roomId, TRTCLiveRoomCallback.ActionCallback call
```

The parameters are described below:

| Parameter | Type           | Description                                                       |
|-----------|----------------|-------------------------------------------------------------------|
| roomId    | int            | The room ID.                                                      |
| callback  | ActionCallback | Callback for room entry. The code is 0 if the operation succeeds. |



The process of entering a room and starting playback as an audience member is as follows:

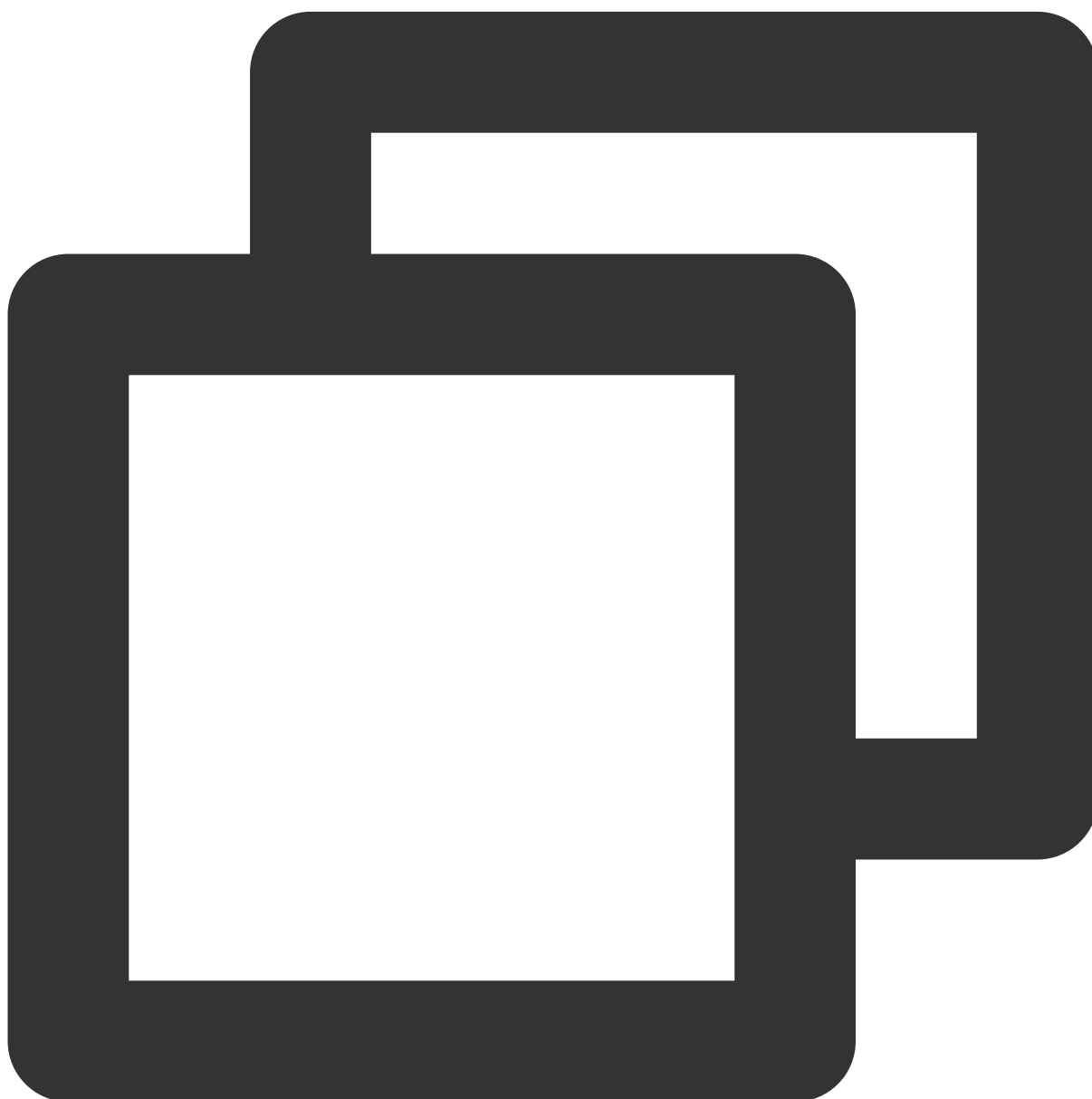
1. A user gets the latest room list from your server. The list may contain the `roomId` and other information of multiple rooms.
2. The user selects a room and calls `enterRoom()` to enter the room.
3. The user calls `startPlay(userId)`, passing in the anchor's `userId` to start playback.

If the room list contains the anchor's `userId`, the user can call `startPlay(userId)` to start playback.

If the user does not have the anchor's `userId` before room entry, he or she can find it in the `onAnchorEnter(userId)` callback of `TRTCLiveRoomDelegate`, which is returned after room entry, and can then call `startPlay(userId)` to start playback.

## exitRoom

This API is used to leave a room.



```
public abstract void exitRoom(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are described below:

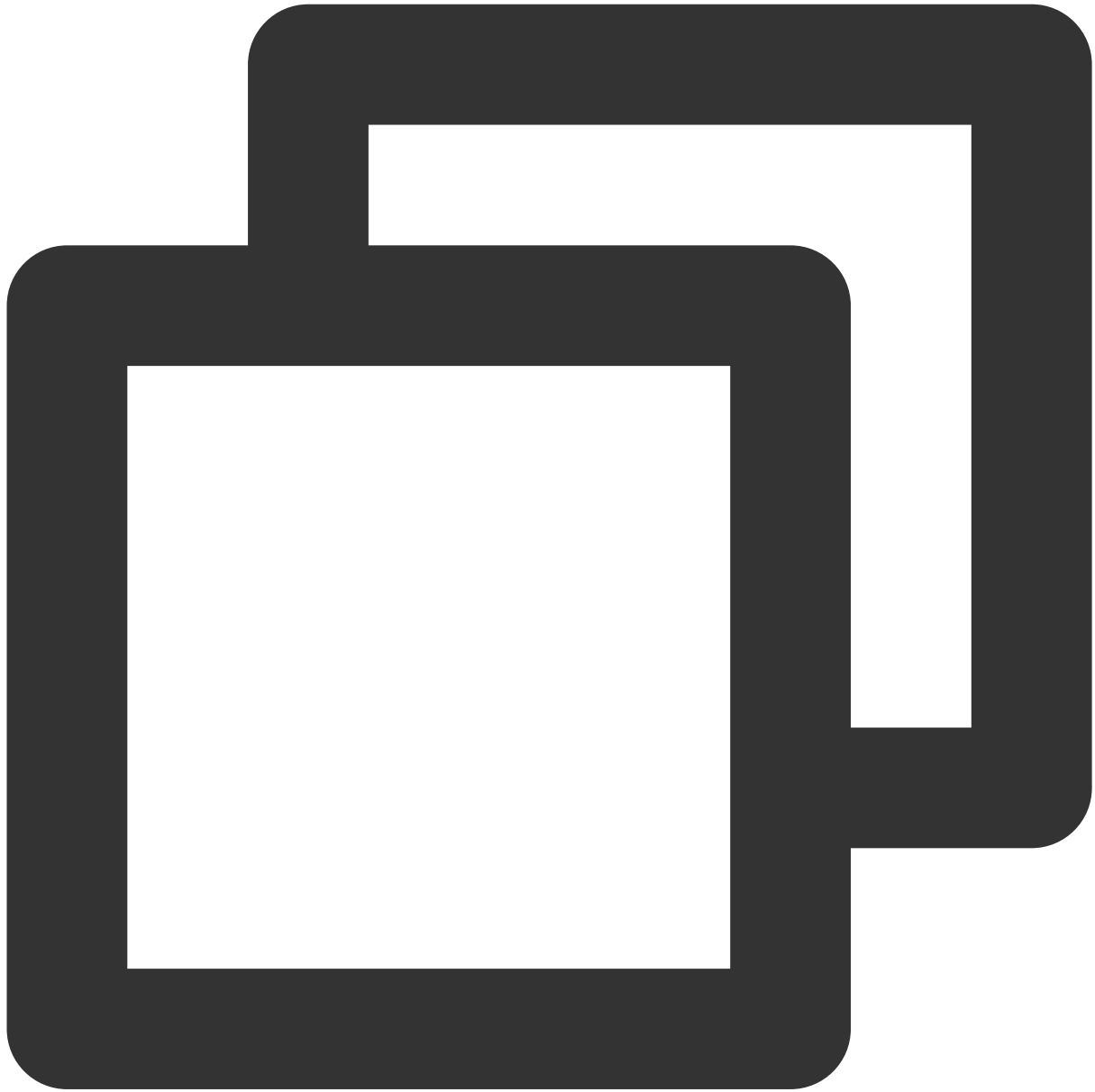
| Parameter | Type           | Description                                                                   |
|-----------|----------------|-------------------------------------------------------------------------------|
| callback  | ActionCallback | Callback for room exit. The code is <code>0</code> if the operation succeeds. |

## getRoomInfos

This API is used to get room list details, which are set by anchors via `roomInfo` when they call `createRoom()`.

**Note**

You don't need this API if both the room list and room information are managed on your server.



```
public abstract void getRoomInfos(List<Integer> roomIdList, TRTCLiveRoomCallback.Ro
```

The parameters are described below:

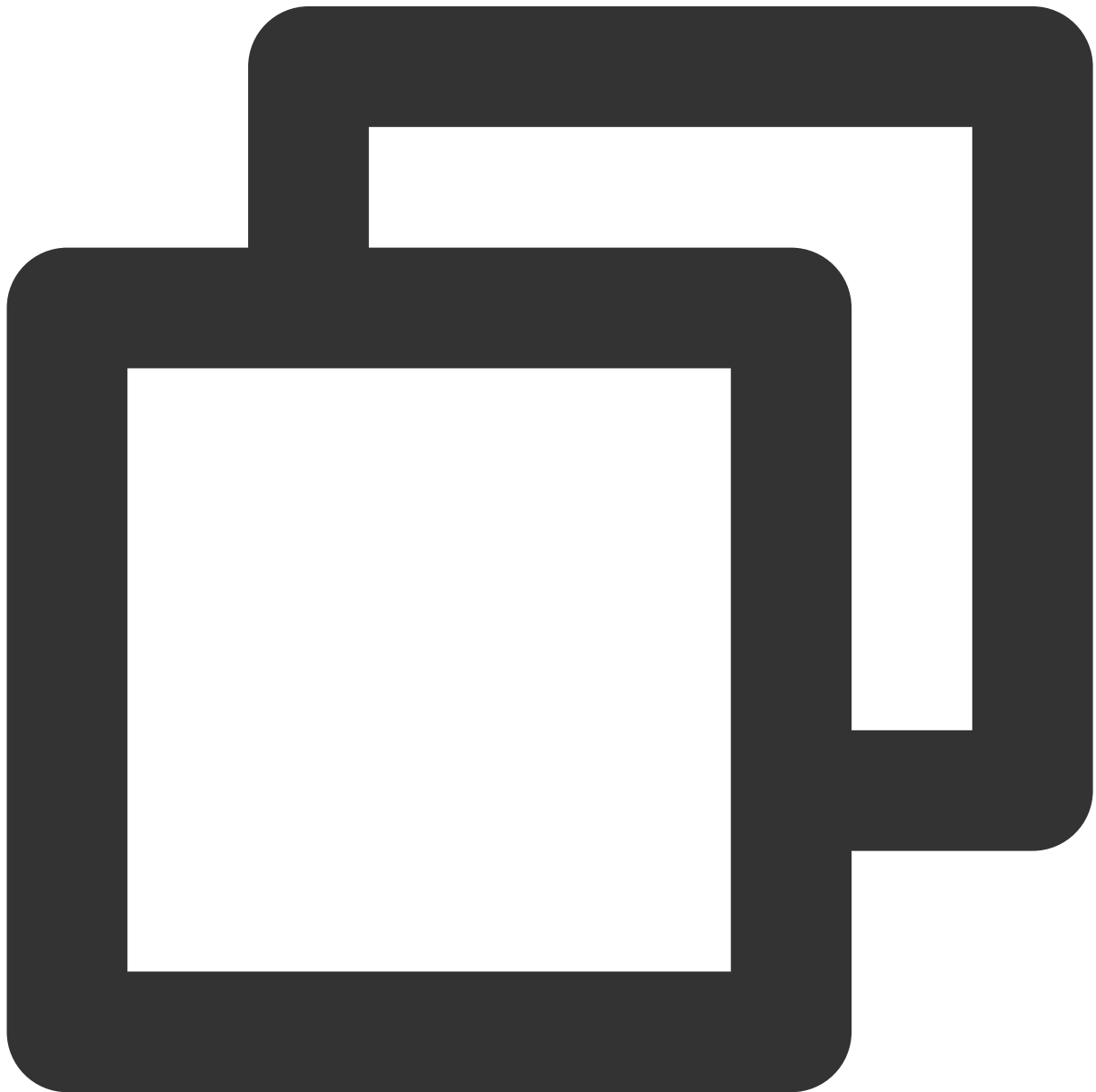
| Parameter | Type | Description |
|-----------|------|-------------|
|           |      |             |

|            |                  |                          |
|------------|------------------|--------------------------|
| roomIdList | List<Integer>    | Room ID list             |
| callback   | RoomInfoCallback | Callback of room details |

## getAnchorList

This API is used to get the anchors and co-anchoring viewers in a room. It takes effect only if it is called after

`enterRoom()` .



```
public abstract void getAnchorList(TRTCLiveRoomCallback.UserListCallback callback);
```

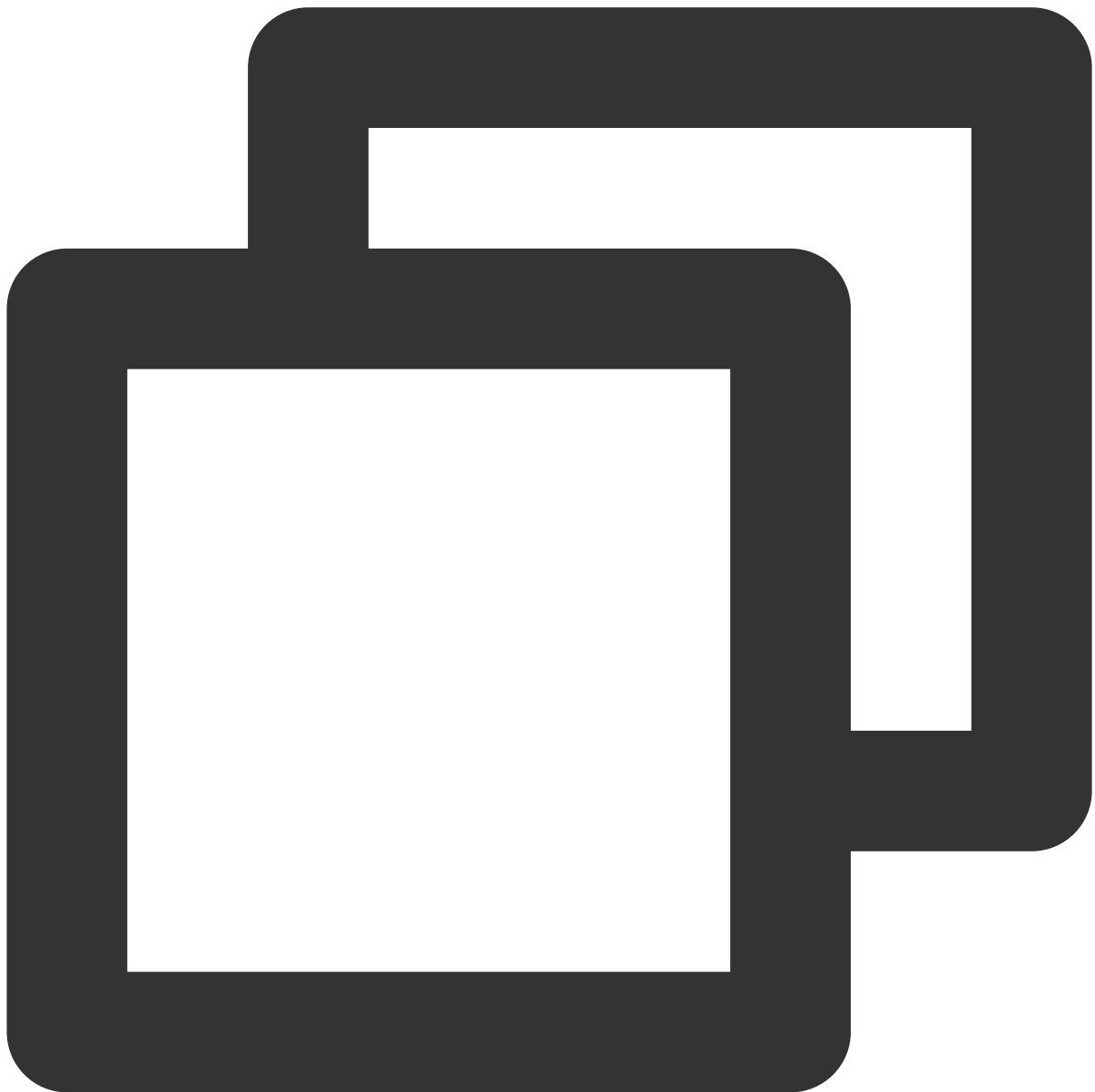
The parameters are described below:

| Parameter | Type             | Description              |
|-----------|------------------|--------------------------|
| callback  | UserListCallback | Callback of user details |

## getAudienceList

This API is used to get the information of all audience members in a room. It takes effect only if it is called after

```
enterRoom() .
```



```
public abstract void getAudienceList (TRTCLiveRoomCallback.UserListCallback callback
```

The parameters are described below:

| Parameter | Type             | Description              |
|-----------|------------------|--------------------------|
| callback  | UserListCallback | Callback of user details |

## Stream Pushing/Pulling APIs

### startCameraPreview

This API is used to enable local video preview.



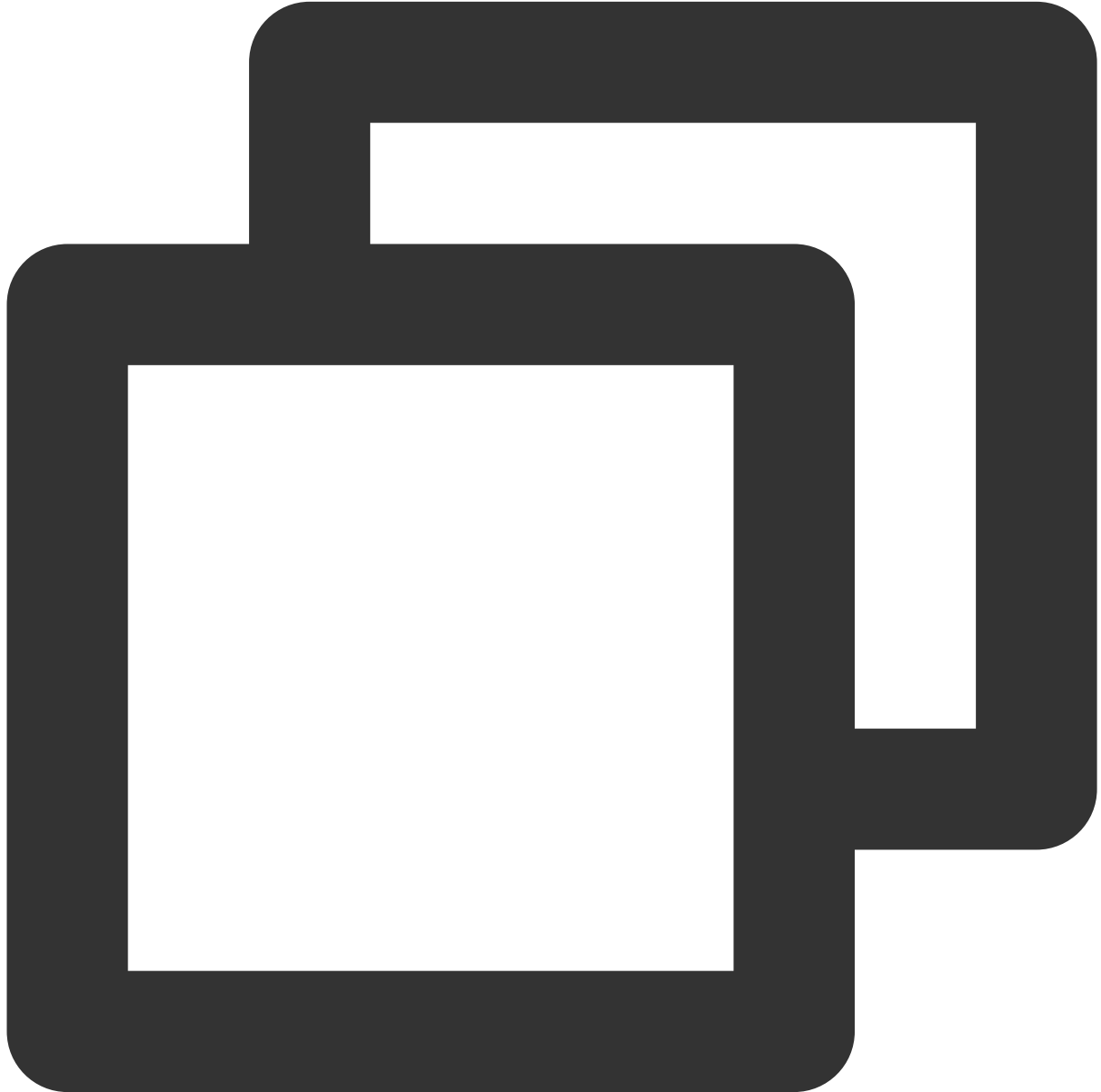
```
public abstract void startCameraPreview(boolean isFront, TXCloudVideoView view, TRT
```

The parameters are described below:

| Parameter | Type             | Description                                                        |
|-----------|------------------|--------------------------------------------------------------------|
| isFront   | boolean          | <code>true</code> : Front camera; <code>false</code> : Rear camera |
| view      | TXCloudVideoView | The control that loads video images                                |
| callback  | ActionCallback   | Callback for the operation                                         |

## stopCameraPreview

This API is used to stop local video capturing and preview.



```
public abstract void stopCameraPreview();
```

## startPublish

This API is used to start live streaming (pushing streams), which can be called in the following scenarios:

An anchor starts live streaming.

An audience member starts co-anchoring.





```
public abstract void startPublish(String streamId, TRTCLiveRoomCallback.ActionCallb
```

The parameters are described below:

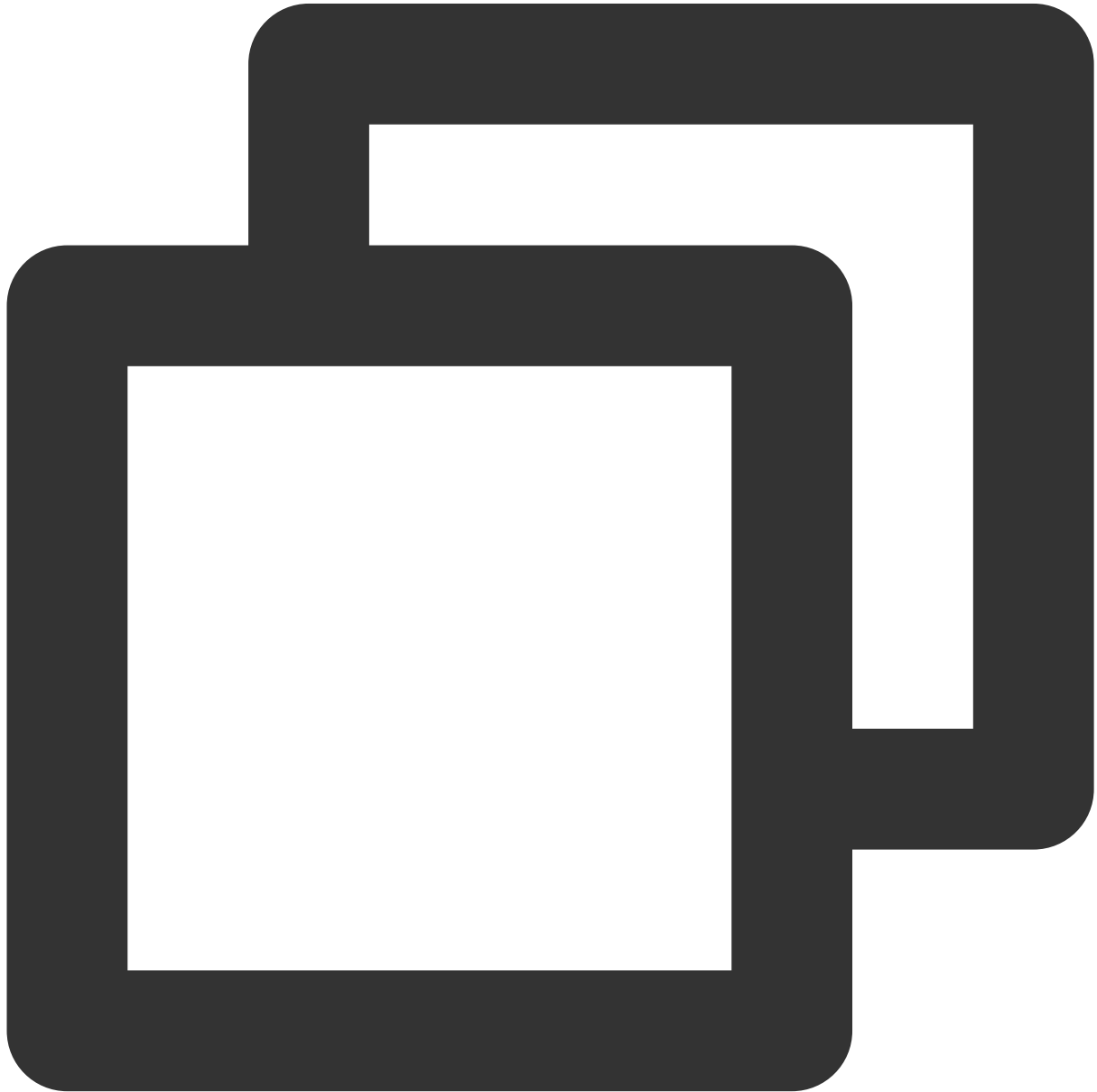
| Parameter | Type           | Description                                                                                                                                                                                         |
|-----------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| streamId  | String         | The <code>streamId</code> used to bind live streaming CDNs. You need to set it to the <code>streamId</code> of the anchor if you want audience to play the anchor's stream via live streaming CDNs. |
| callback  | ActionCallback | Callback for the operation                                                                                                                                                                          |

## stopPublish

This API is used to stop live streaming (pushing streams), which can be called in the following scenarios:

An anchor ends live streaming.

An audience member ends co-anchoring.



```
public abstract void stopPublish(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are described below:

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

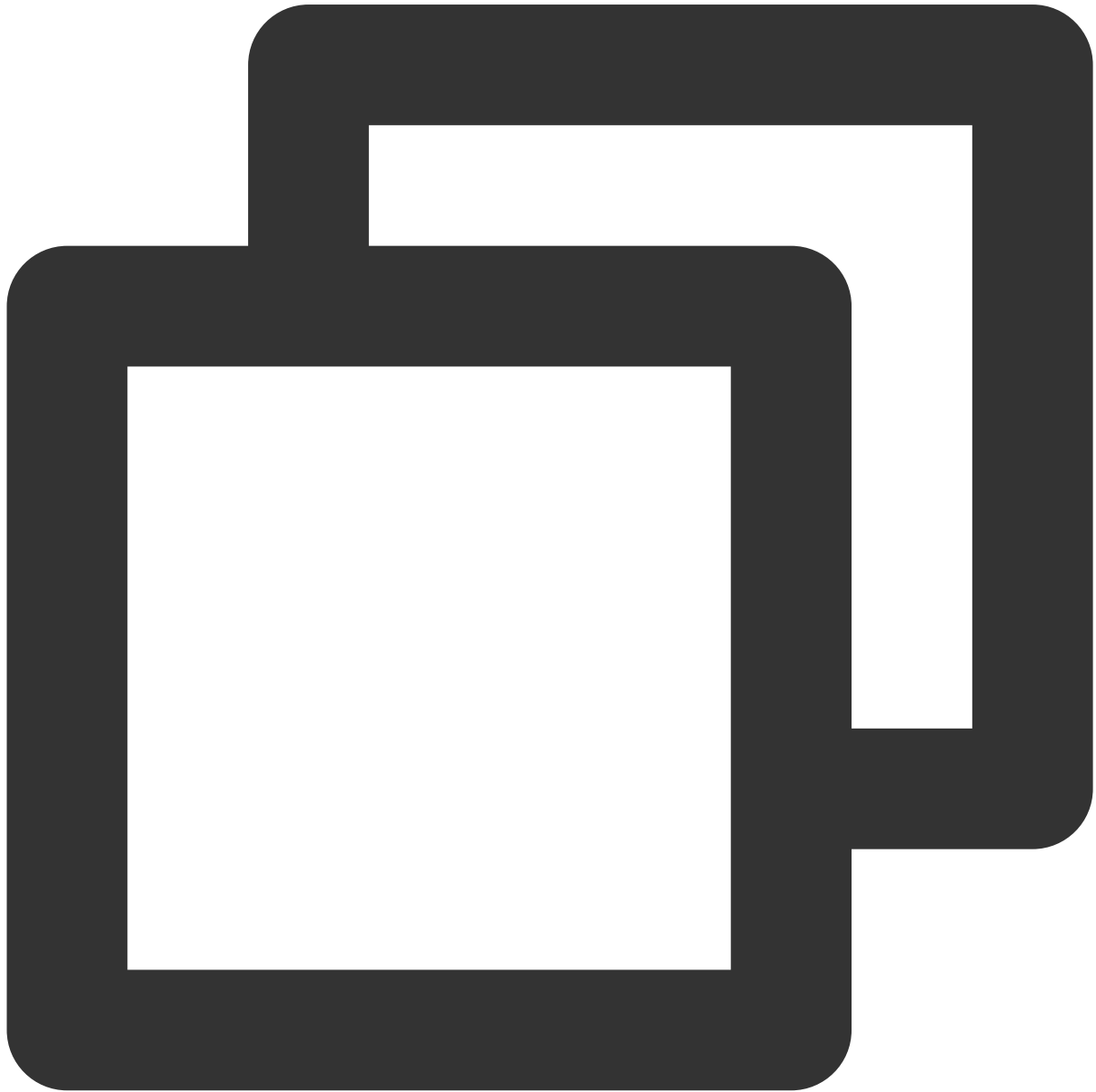
callback

ActionCallback

Callback for the operation

## startPlay

This API is used to play a remote video. It can be called in common playback and co-anchoring scenarios.



```
public abstract void startPlay(String userId, TXCloudVideoView view, TRTCLiveRoomCa
```

The parameters are described below:

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

|          |                  |                                            |
|----------|------------------|--------------------------------------------|
| userId   | String           | ID of the user whose video is to be played |
| view     | TXCloudVideoView | The control that loads video images        |
| callback | ActionCallback   | Callback for the operation                 |

### Common playback scenario

If the room list contains anchors' `userId`, after entering a room, a user can call `startPlay(userId)` to play the anchor's video.

If a user does not have the anchor's `userId` before room entry, he or she can find it in the `onAnchorEnter(userId)` callback of `TRTCLiveRoomDelegate`, which is returned after room entry, and can then call `startPlay(userId)` to play the anchor's video.

### Co-anchoring scenario

After co-anchoring starts, the anchor will receive the `onAnchorEnter(userId)` callback from `TRTCLiveRoomDelegate` and can call `startPlay(userId)`, passing in the `userId` obtained from the callback to play the co-anchoring user's video.

### stopPlay

This API is used to stop rendering a remote video. It needs to be called after the `onAnchorExit()` callback is received.



```
public abstract void stopPlay(String userId, TRTCLiveRoomCallback.ActionCallback ca
```

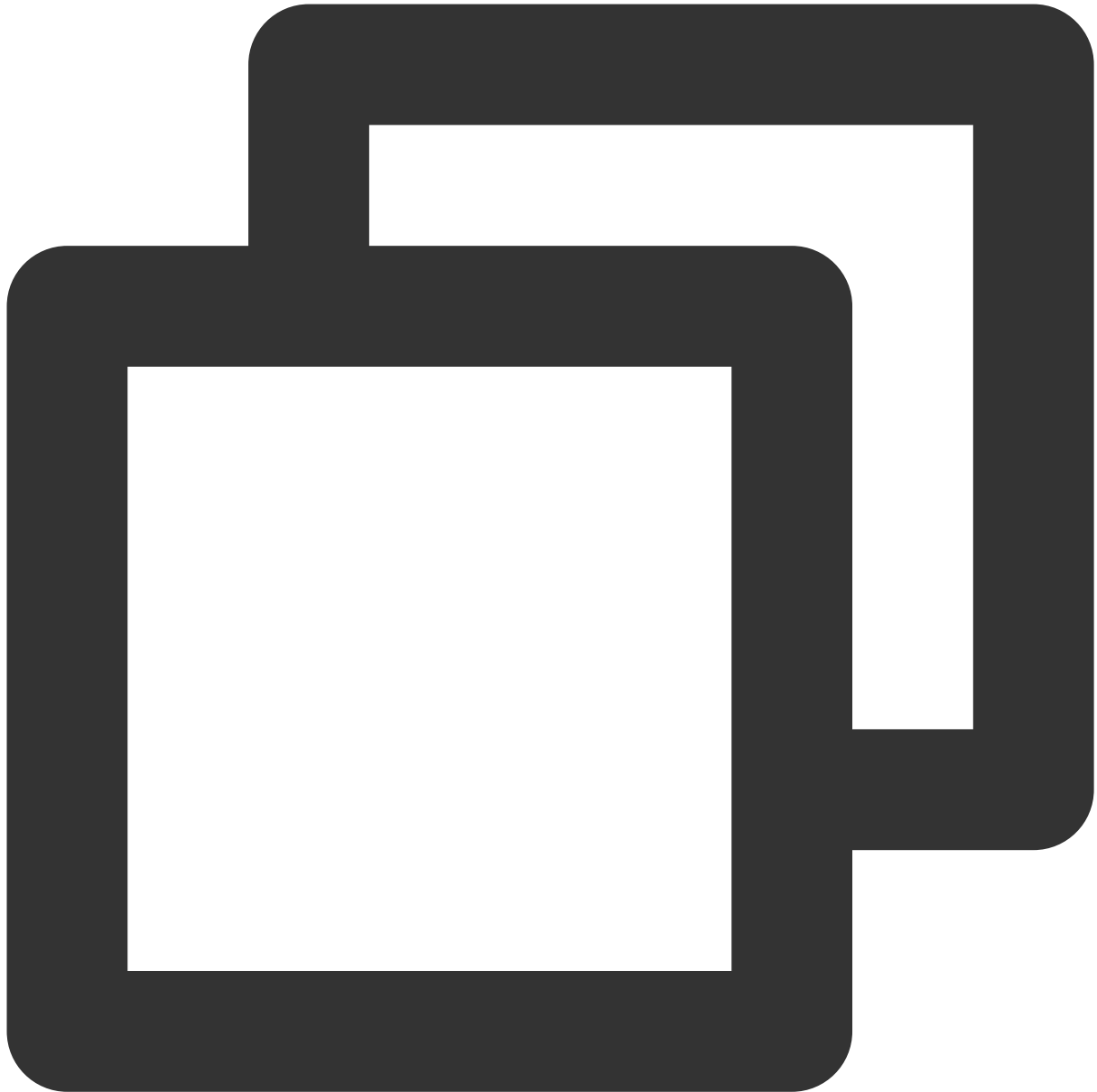
The parameters are described below:

| Parameter | Type           | Description                |
|-----------|----------------|----------------------------|
| userId    | String         | ID of the remote user      |
| callback  | ActionCallback | Callback for the operation |

## APIs for Anchor-Audience Co-anchoring

### **requestJoinAnchor**

This API is used to send a co-anchoring request (called by audience).



```
public abstract void requestJoinAnchor(String reason, int timeout, TRTCLiveRoomCall
```

The parameters are described below:

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

|          |                |                                   |
|----------|----------------|-----------------------------------|
| reason   | String         | Reason for co-anchoring           |
| timeout  | int            | Timeout period                    |
| callback | ActionCallback | Callback of the anchor's response |

The process of co-anchoring between anchors and audience members is as follows:

1. A **viewer** calls `requestJoinAnchor()` to send a co-anchoring request to the anchor.
2. The **anchor** receives the `onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate`.
3. The **anchor** calls `responseJoinAnchor()` to accept or reject the co-anchoring request.
4. The **viewer** receives the `responseCallback` callback, which carries the anchor's response.
5. If the request is accepted, the **viewer** calls `startCameraPreview()` to enable local camera preview.
6. The **viewer** calls `startPublish()` to push streams.
7. After the viewer starts pushing streams, the **anchor** receives the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate`.
8. The **anchor** calls `startPlay()` to play the co-anchoring viewer's video.
9. If there are other viewers co-anchoring with the anchor in the room, the new co-anchoring **viewer** will receive the `onAnchorEnter()` callback and can call `startPlay()` to play other co-anchoring viewers' video.

## responseJoinAnchor

This API is used to respond to a co-anchoring request (called by anchor) after receiving the

`onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate`.



```
public abstract void responseJoinAnchor(String userId, boolean agree, String reason
```

The parameters are described below:

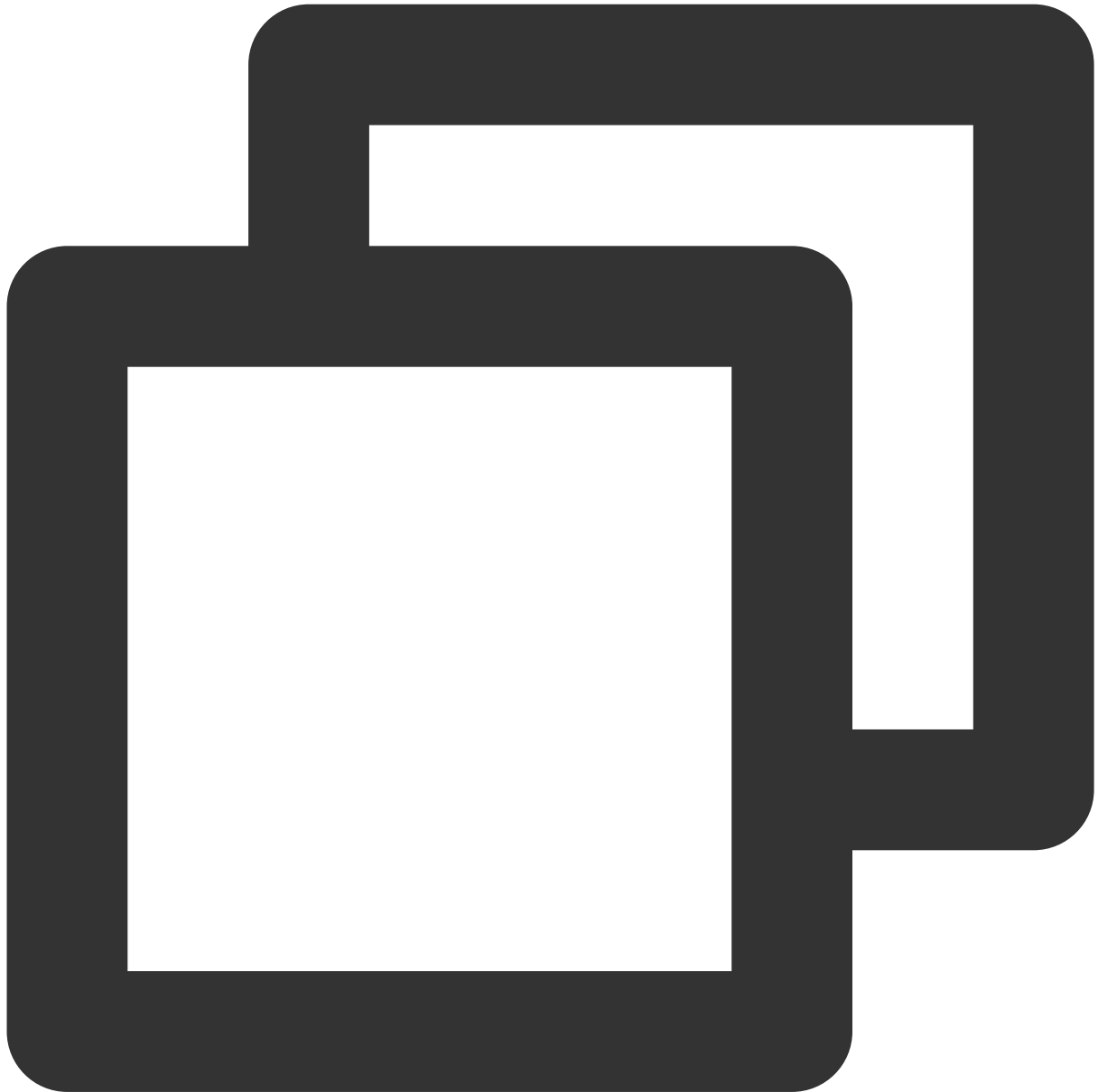
| Parameter | Type    | Description                                             |
|-----------|---------|---------------------------------------------------------|
| userId    | String  | The user ID of the audience member.                     |
| agree     | boolean | <code>true</code> : accept; <code>false</code> : reject |
| reason    | String  | Reason for accepting/rejecting the request              |



## kickoutJoinAnchor

This API is used to remove a user from co-anchoring (called by anchor). The removed user will receive the

`onKickoutJoinAnchor()` callback of `TRTCLiveRoomDelegate`.



```
public abstract void kickoutJoinAnchor(String userId, TRTCLiveRoomCallback.ActionCa
```

The parameters are described below:

| Parameter | Type   | Description                      |
|-----------|--------|----------------------------------|
| userId    | String | The ID of the co-anchoring user. |

callback

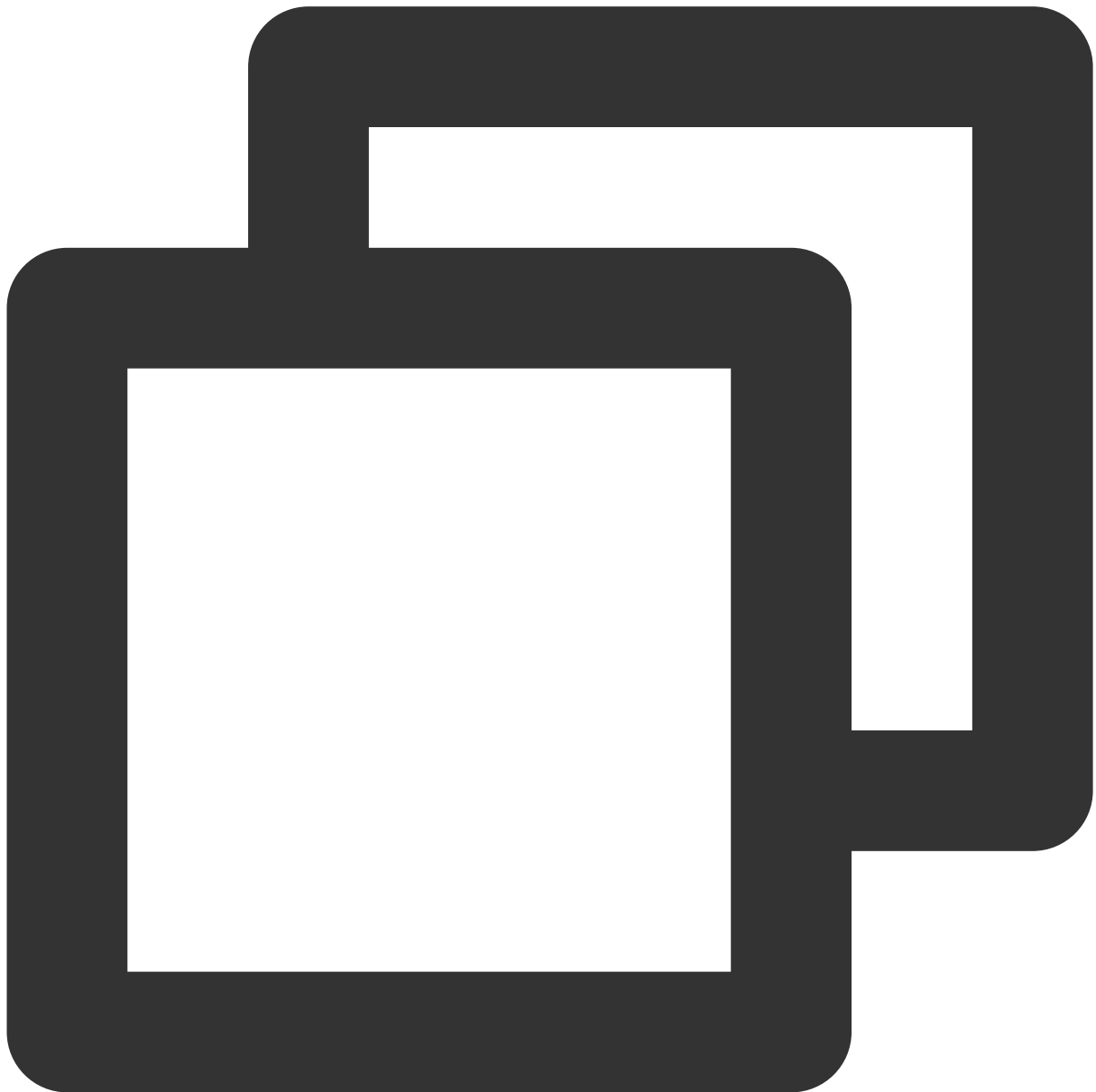
ActionCallback

Callback for the operation

## APIs for Cross-Room Communication

### **requestRoomPK**

This API is used to send a cross-room communication request (called by anchor).



```
public abstract void requestRoomPK(int roomId, String userId, TRTCLiveRoomCallback.
```

The parameters are described below:

| Parameter | Type           | Description                                       |
|-----------|----------------|---------------------------------------------------|
| roomId    | int            | Room ID of the anchor to call                     |
| userId    | String         | User ID of the anchor to call                     |
| callback  | ActionCallback | Callback for requesting cross-room communication. |

Anchors in different rooms can communicate with each other. The process of starting cross-room communication between anchor A and anchor B is as follows:

1. **Anchor A** calls `requestRoomPK()` to send a cross-room communication request to anchor B.
2. **Anchor B** receives the `onRequestRoomPK()` callback of `TRTCLiveRoomDelegate`.
3. **Anchor B** calls `responseRoomPK()` to respond to the cross-room communication request.
4. If **anchor B** accepts the request, he or she would wait for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and call `startPlay()` to play anchor A's video.
5. **Anchor A** receives the `responseCallback` callback, which carries anchor B's response.
6. If the request is accepted, **anchor A** waits for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and calls `startPlay()` to play anchor B's video.

## responseRoomPK

This API is used to respond to a cross-room communication request (called by anchor), after which the request sending anchor will receive the `responseCallback` passed in to `requestRoomPK`.



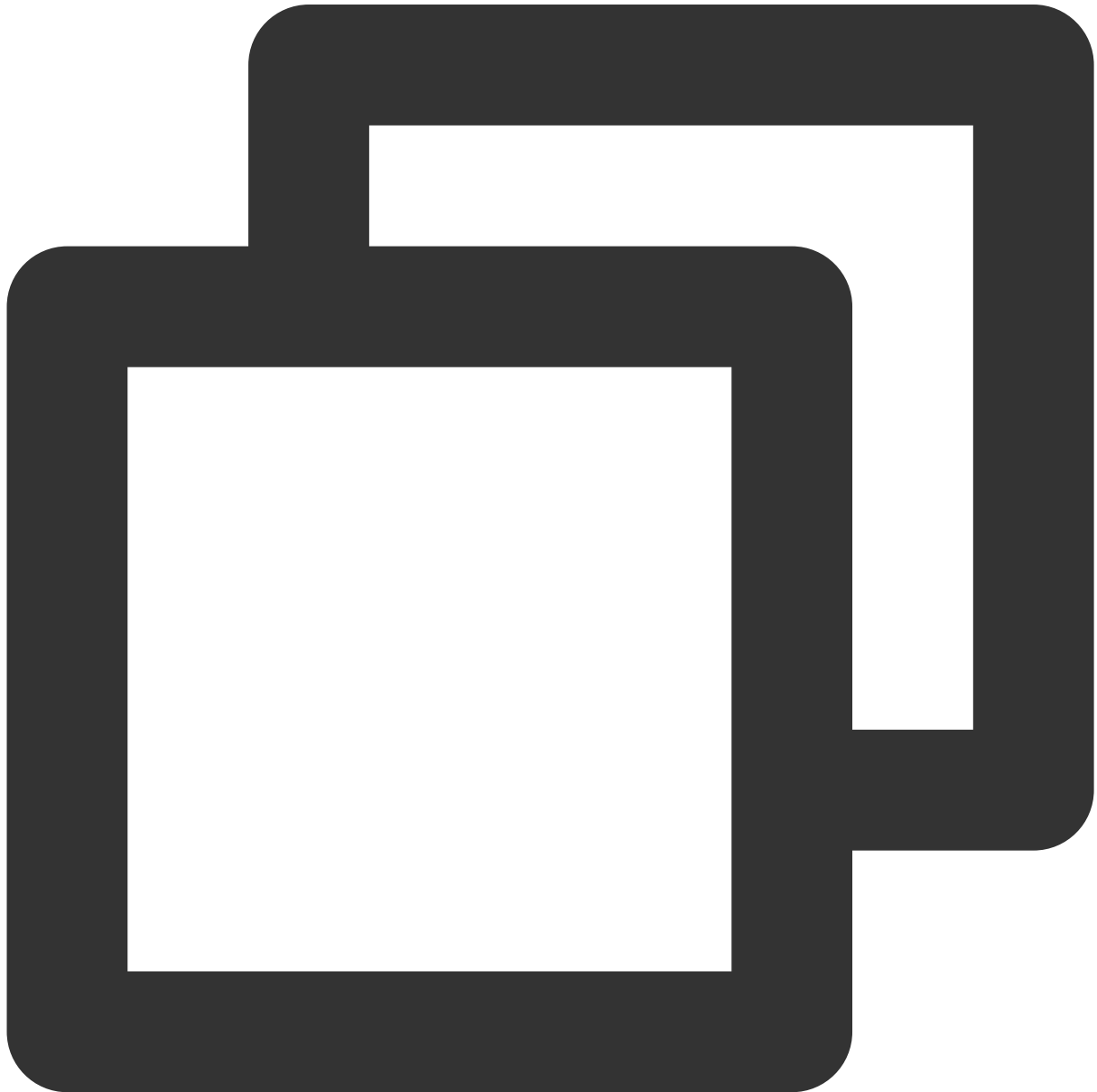
```
public abstract void responseRoomPK(String userId, boolean agree, String reason);
```

The parameters are described below:

| Parameter | Type    | Description                                             |
|-----------|---------|---------------------------------------------------------|
| userId    | String  | User ID of the request sending anchor                   |
| agree     | boolean | <code>true</code> : Accept; <code>false</code> : Reject |
| reason    | String  | Reason for accepting/rejecting the request              |

## quitRoomPK

This API is used to quit cross-room communication. If either anchor quits cross-room communication, the other anchor will receive the `onQuitRoomPk()` callback of `TRTCLiveRoomDelegate`.



```
public abstract void quitRoomPK(TRTCLiveRoomCallback.ActionCallback callback);
```

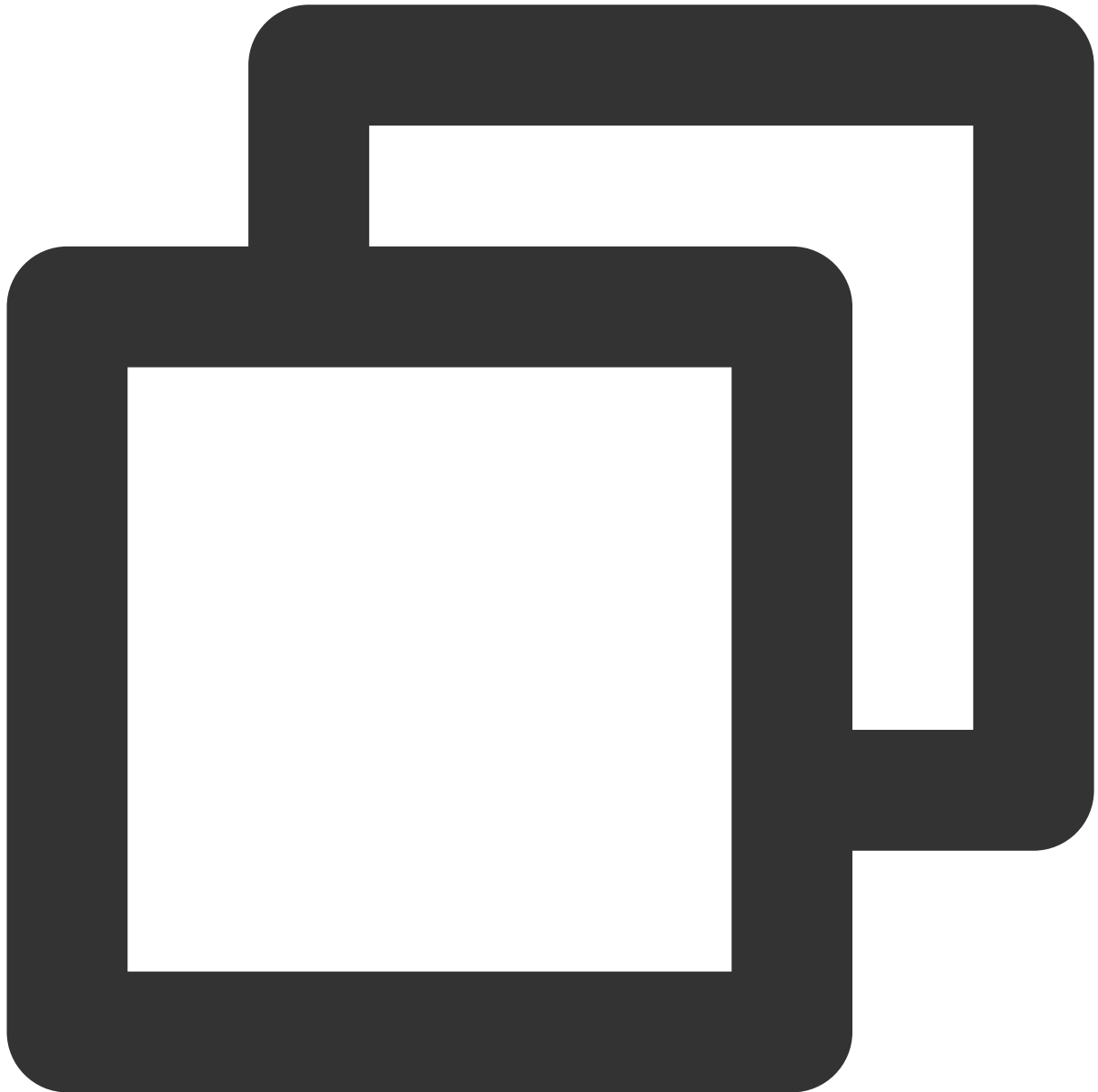
The parameters are described below:

| Parameter | Type           | Description                |
|-----------|----------------|----------------------------|
| callback  | ActionCallback | Callback for the operation |

## Audio/Video APIs

### **switchCamera**

This API is used to switch between the front and rear cameras.



```
public abstract void switchCamera();
```

### **setMirror**

This API is used to set the mirror mode.



```
public abstract void setMirror(boolean isMirror);
```

The parameters are described below:

| Parameter | Type    | Description              |
|-----------|---------|--------------------------|
| isMirror  | boolean | Enable/Disable mirroring |

### **muteLocalAudio**

This API is used to mute or unmute the local user.



```
public abstract void muteLocalAudio(boolean mute);
```

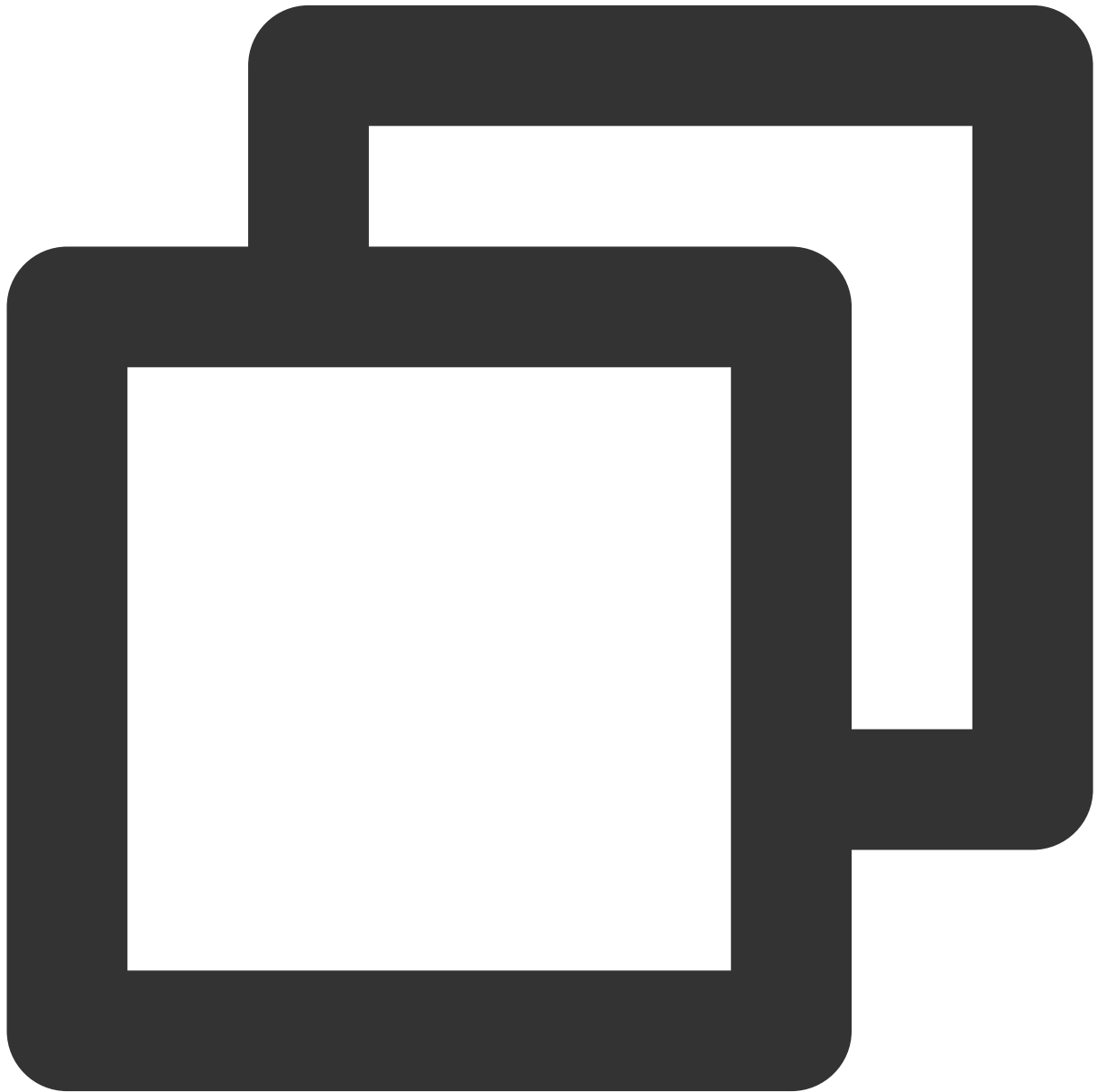
The parameters are described below:

| Parameter | Type    | Description                 |
|-----------|---------|-----------------------------|
| mute      | boolean | true : Mute; false : Unmute |

## **muteRemoteAudio**

This API is used to mute or unmute a remote user.





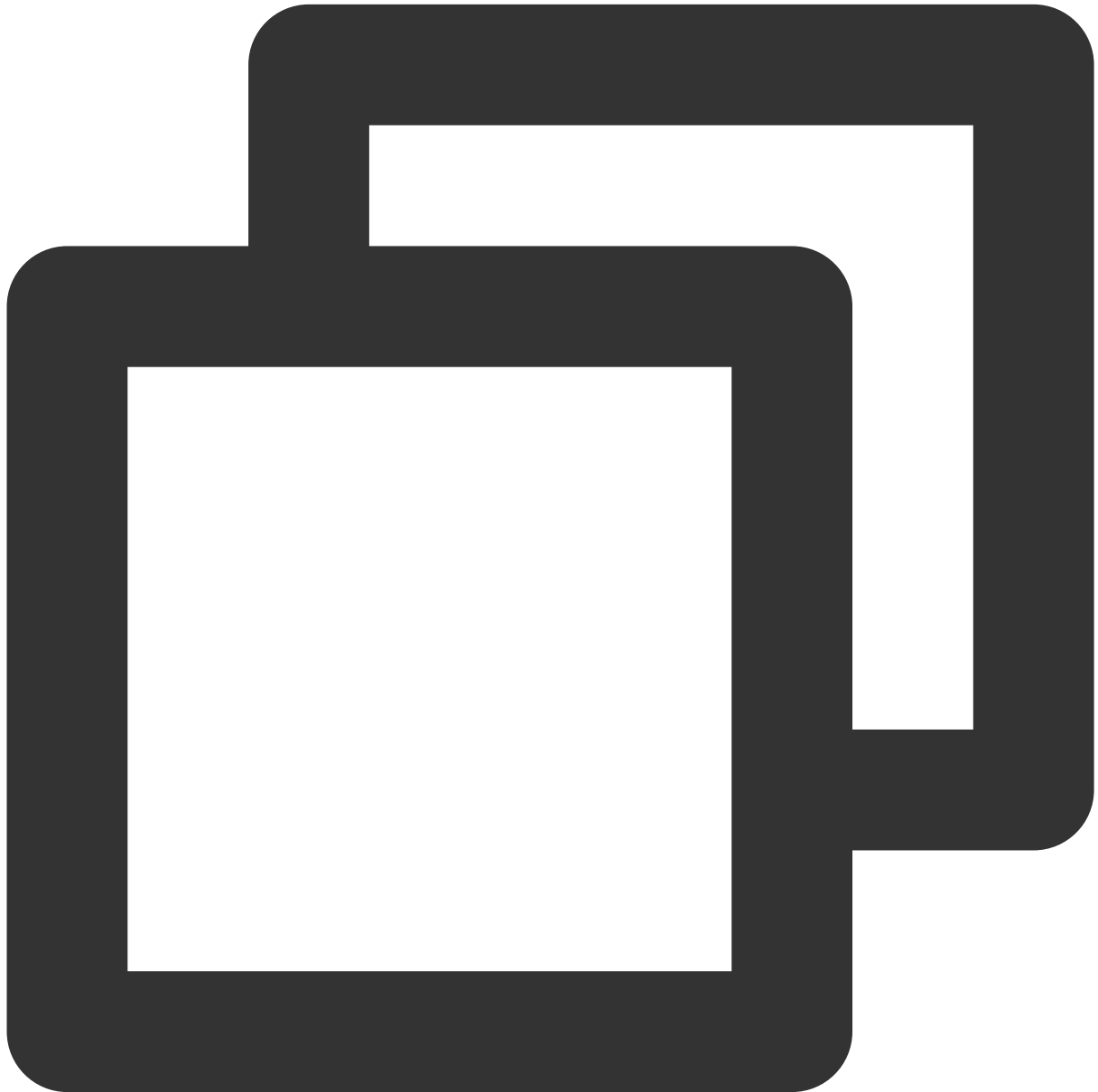
```
public abstract void muteRemoteAudio(String userId, boolean mute);
```

The parameters are described below:

| Parameter | Type    | Description                                           |
|-----------|---------|-------------------------------------------------------|
| userId    | String  | ID of the remote user                                 |
| mute      | boolean | <code>true</code> : Mute; <code>false</code> : Unmute |

## **muteAllRemoteAudio**

This API is used to mute or unmute all remote users.



```
public abstract void muteAllRemoteAudio(boolean mute);
```

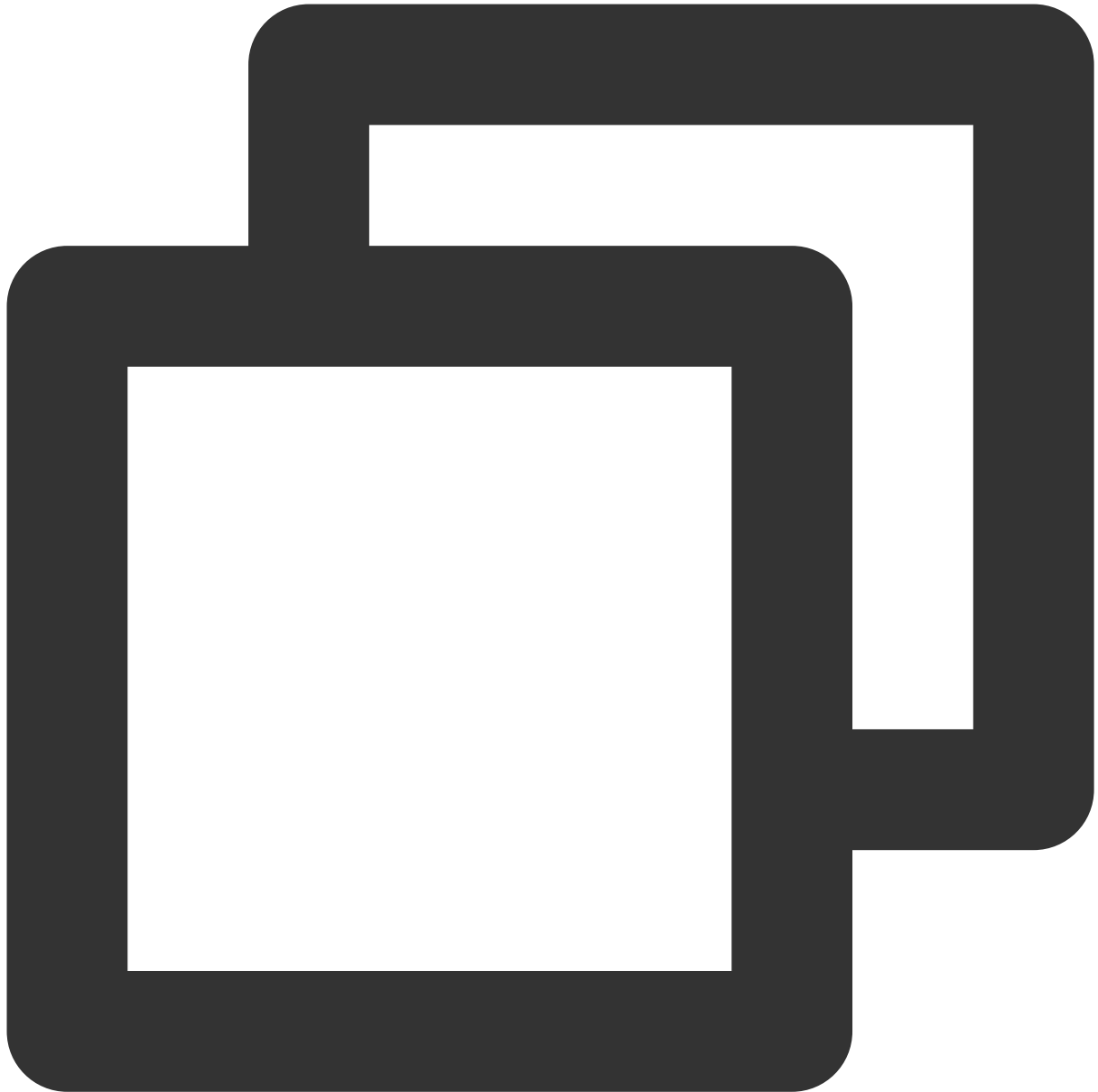
The parameters are described below:

| Parameter | Type    | Description                 |
|-----------|---------|-----------------------------|
| mute      | boolean | true : Mute; false : Unmute |

## Background Music and Audio Effect APIs

### **getAudioEffectManager**

This API is used to get the background music and audio effect management object [TXAudioEffectManager](#).

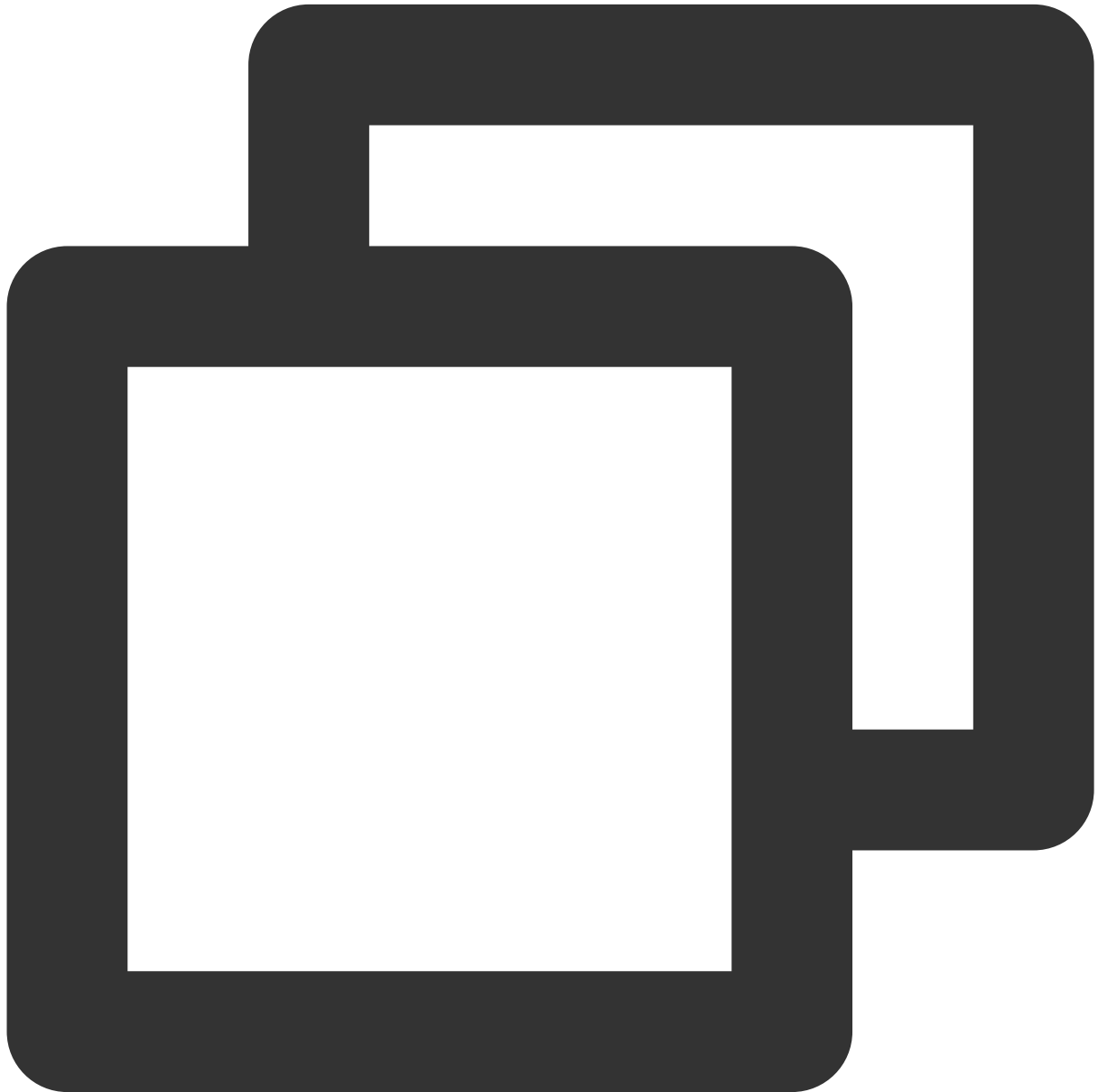


```
public abstract TXAudioEffectManager getAudioEffectManager();
```

## Beauty Filter APIs

## getBeautyManager

This API is used to get the beauty filter management object [TXBeautyManager](#).



```
public abstract TXBeautyManager getBeautyManager();
```

You can do the following using `TXBeautyManager` :

Set the beauty filter style and apply effects including skin brightening, rosy skin, eye enlarging, face slimming, chin slimming, chin lengthening/shortening, face shortening, nose narrowing, eye brightening, teeth whitening, eye bag removal, wrinkle removal, and smile line removal.

Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose position, lip thickness, and face shape.

Apply animated effects such as face widgets (materials).

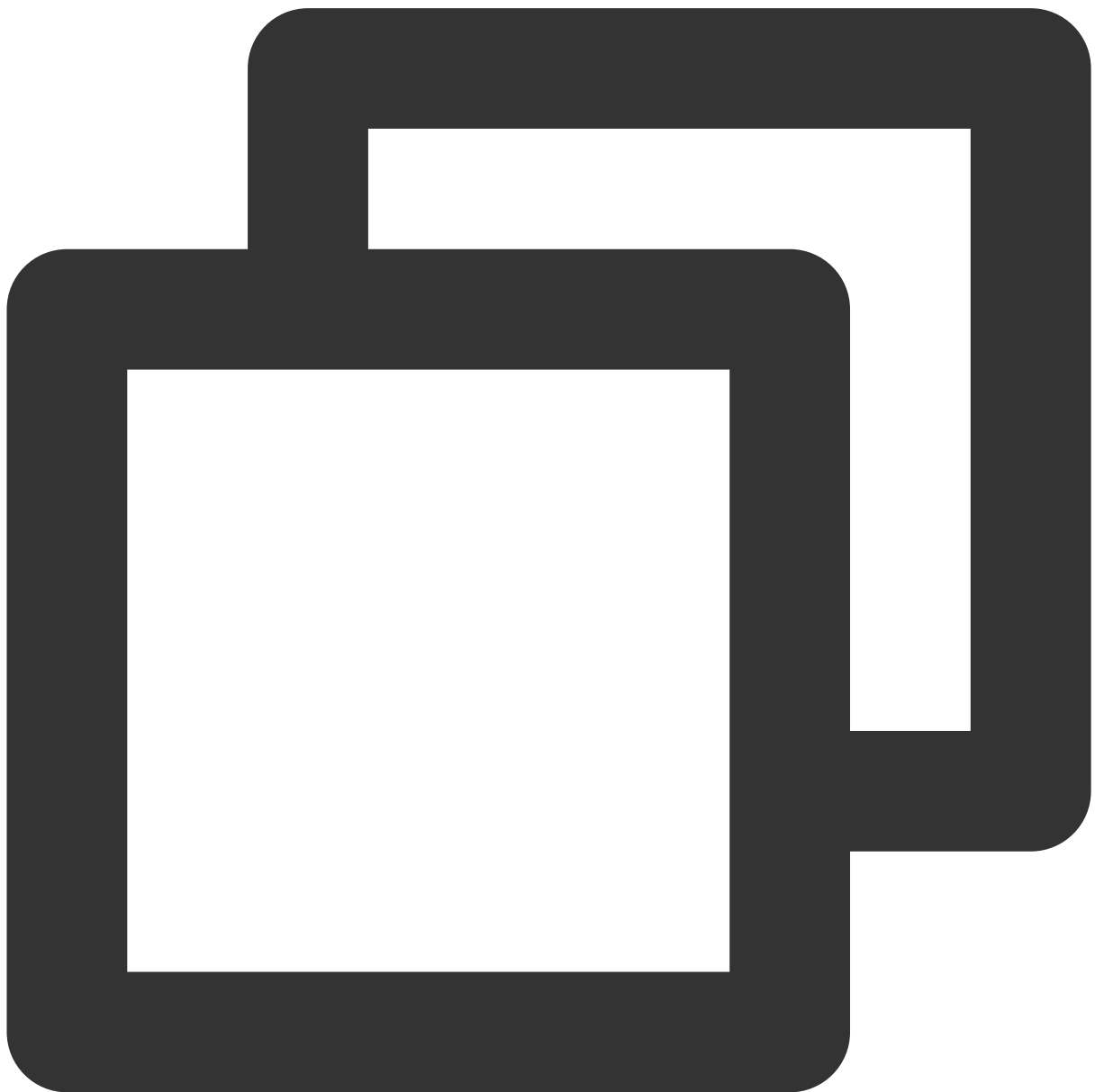
Add makeup effects.

Recognize gestures.

## Message Sending APIs

### **sendRoomTextMsg**

This API is used to broadcast a text chat message in a room, which is generally used for on-screen comments.



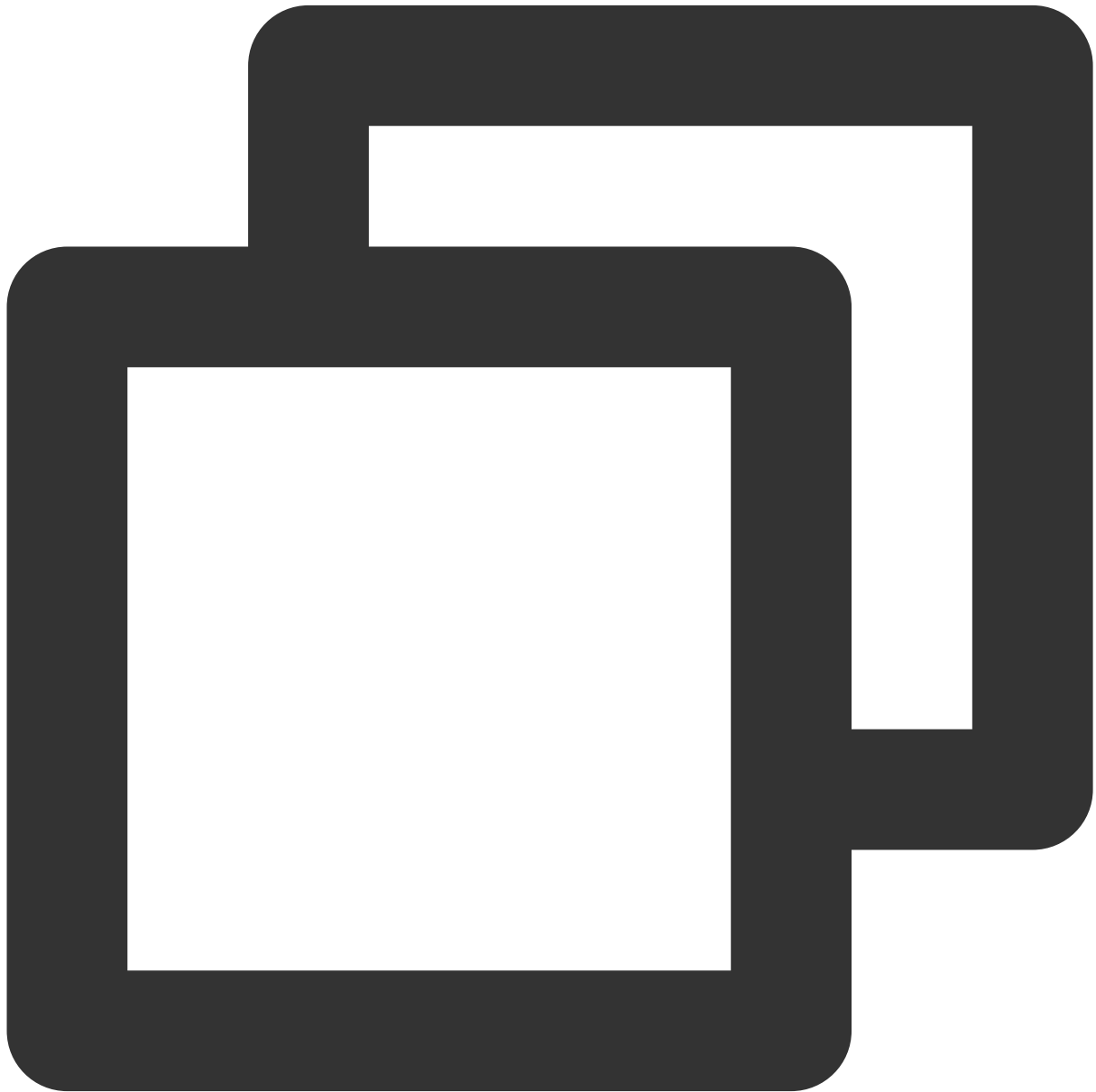
```
public abstract void sendRoomTextMsg(String message, TRTCLiveRoomCallback.ActionCal
```

The parameters are described below:

| Parameter | Type           | Description                |
|-----------|----------------|----------------------------|
| message   | String         | Text message               |
| callback  | ActionCallback | Callback for the operation |

## sendRoomCustomMsg

This API is used to send a custom text message.



```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCLiveRoomCall
```

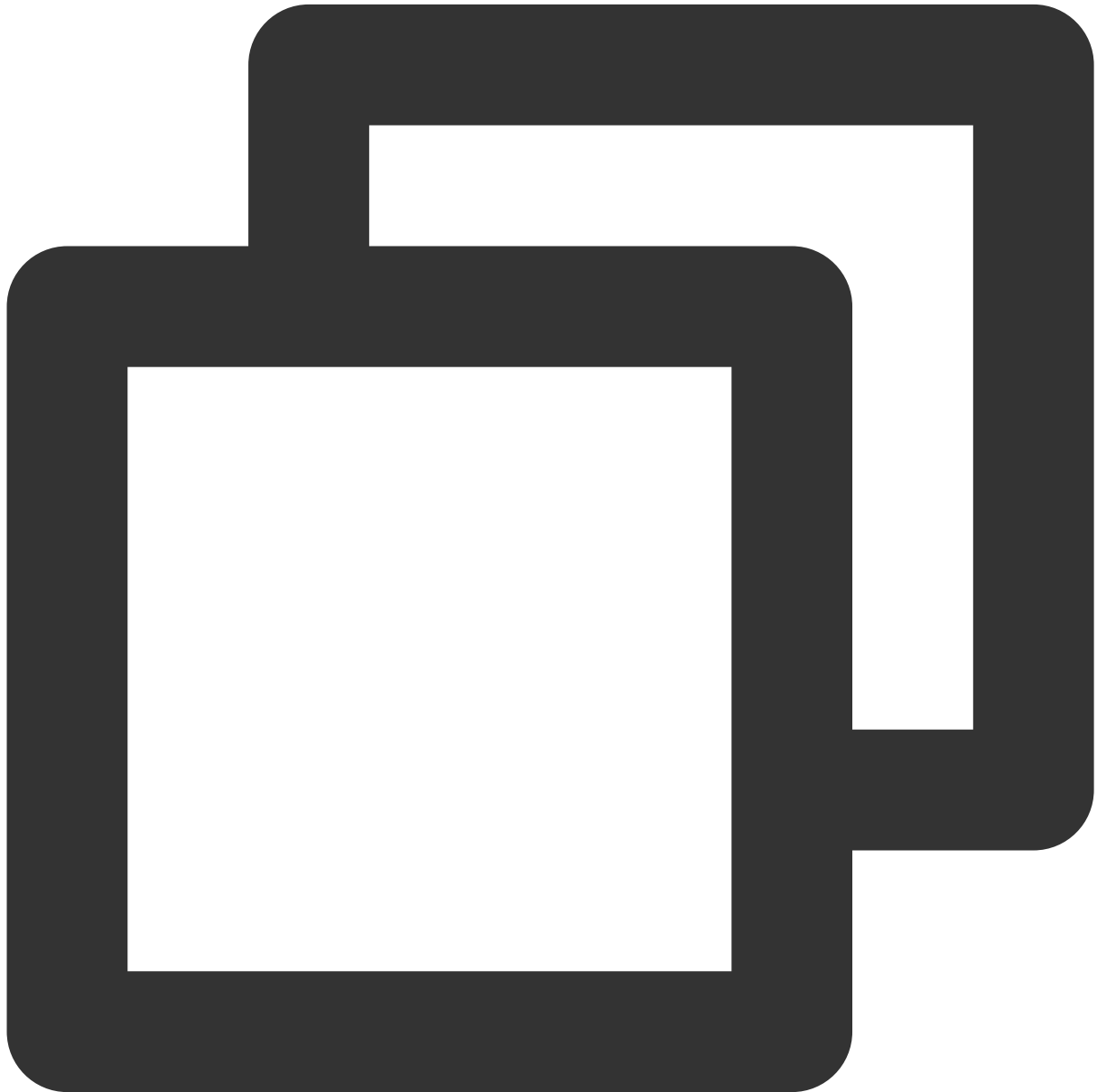
The parameters are described below:

| Parameter | Type           | Description                                                               |
|-----------|----------------|---------------------------------------------------------------------------|
| cmd       | String         | A custom command word used to distinguish between different message types |
| message   | String         | Text message                                                              |
| callback  | ActionCallback | Callback for the operation                                                |

## Debugging APIs

### showVideoDebugLog

This API is used to specify whether to display debugging information on the UI.



```
public abstract void showVideoDebugLog(boolean isShow);
```

The parameters are described below:

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



|        |         |                                 |
|--------|---------|---------------------------------|
| isShow | boolean | Show/Hide debugging information |
|--------|---------|---------------------------------|

## TRTCLiveRoomDelegate Event Callback APIs

### Common Event Callback APIs

#### **onError**

Callback for error.

#### **Note**

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



```
void onError(int code, String message);
```

The parameters are described below:

| Parameter | Type   | Description   |
|-----------|--------|---------------|
| code      | int    | Error code    |
| message   | String | Error message |

## onWarning

Callback for warning.



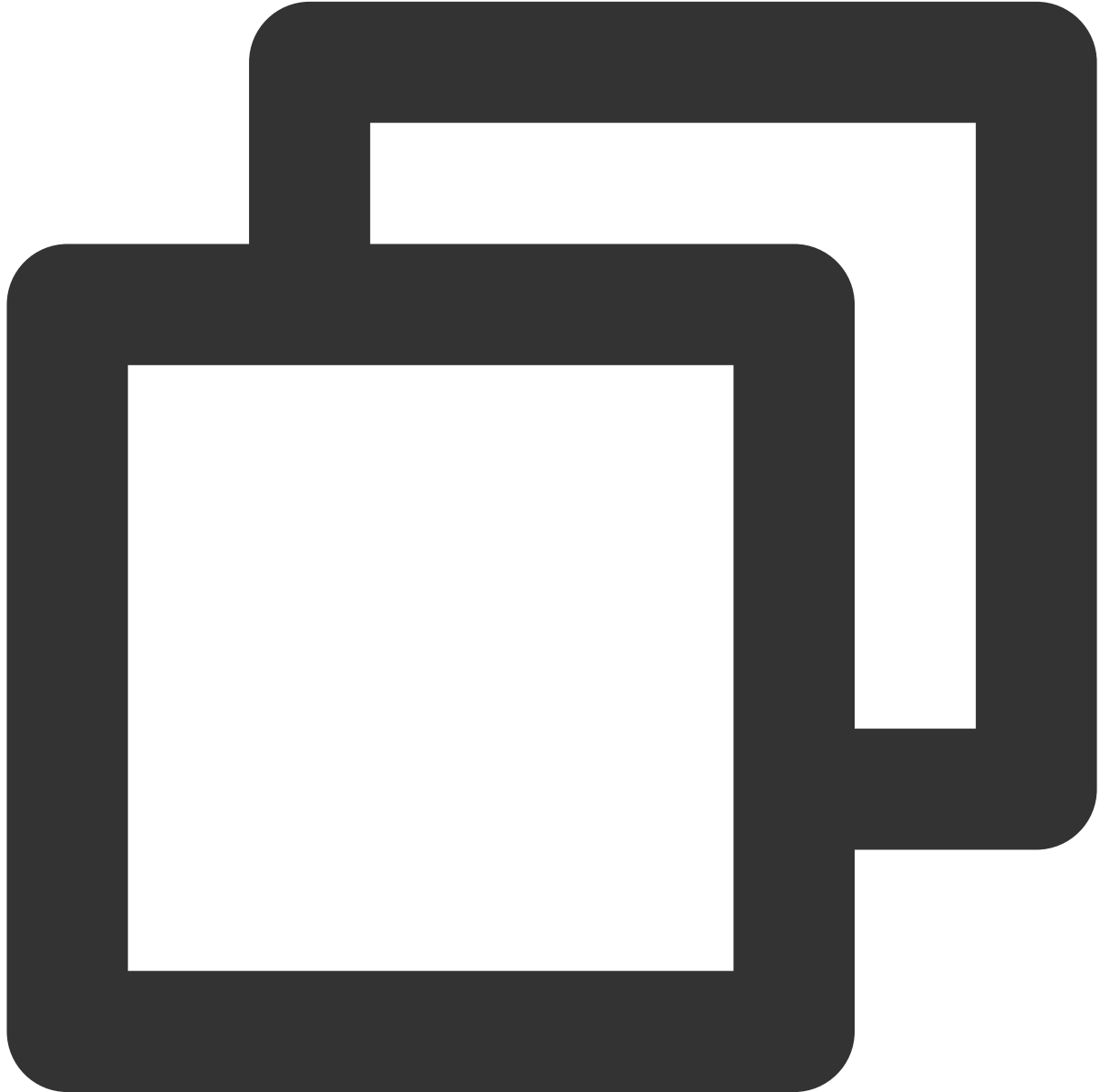
```
void onWarning(int code, String message);
```

The parameters are described below:

| Parameter | Type   | Description     |
|-----------|--------|-----------------|
| code      | int    | Error code      |
| message   | String | Warning message |

## onDebugLog

Callback for log.



```
void onDebugLog(String message);
```

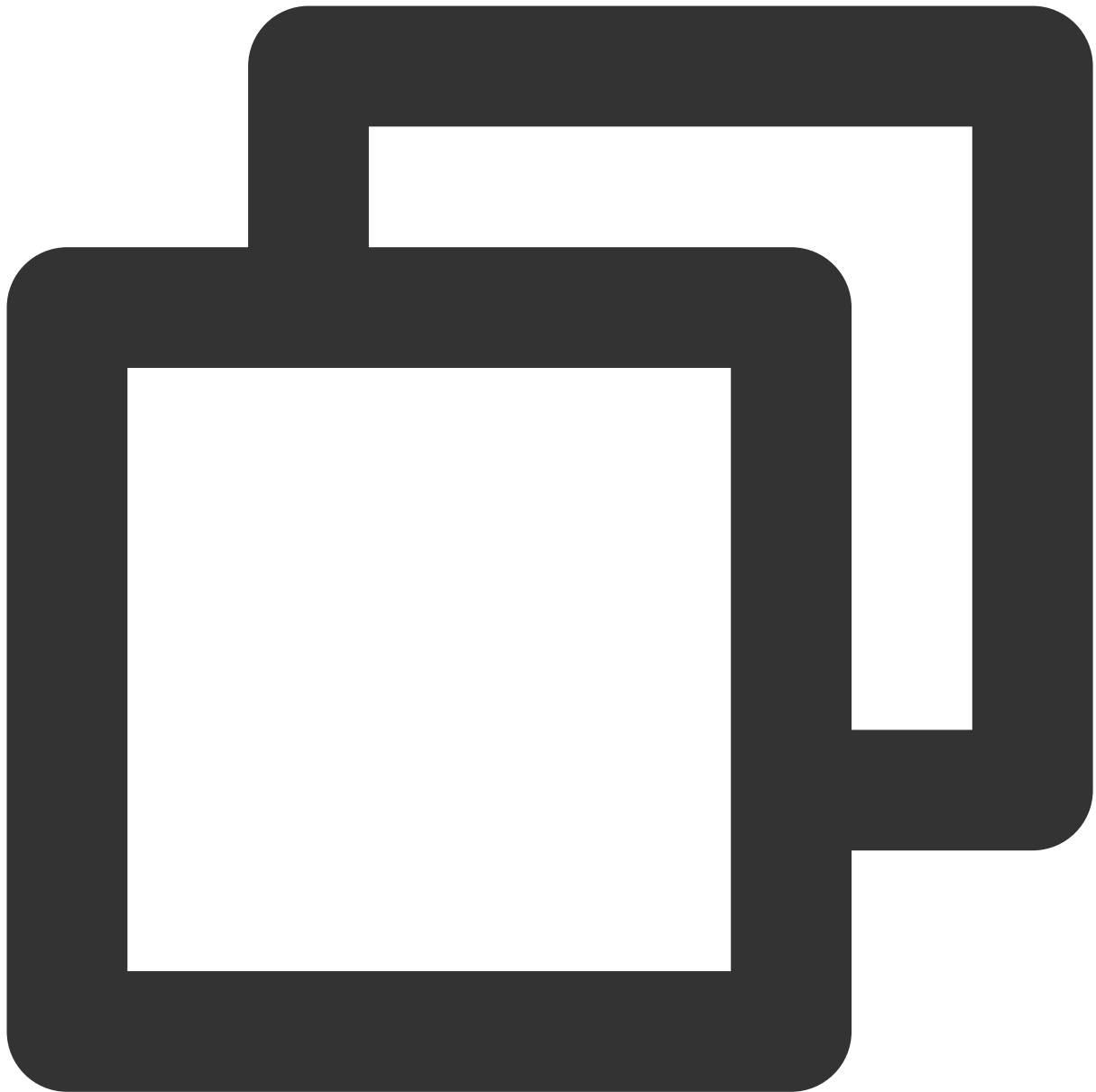
The parameters are described below:

| Parameter | Type   | Description     |
|-----------|--------|-----------------|
| message   | String | Log information |

## Room Event Callback APIs

### onRoomDestroy

Callback for room termination. All users in a room will receive this callback after the anchor leaves the room.



```
void onRoomDestroy(String roomId);
```

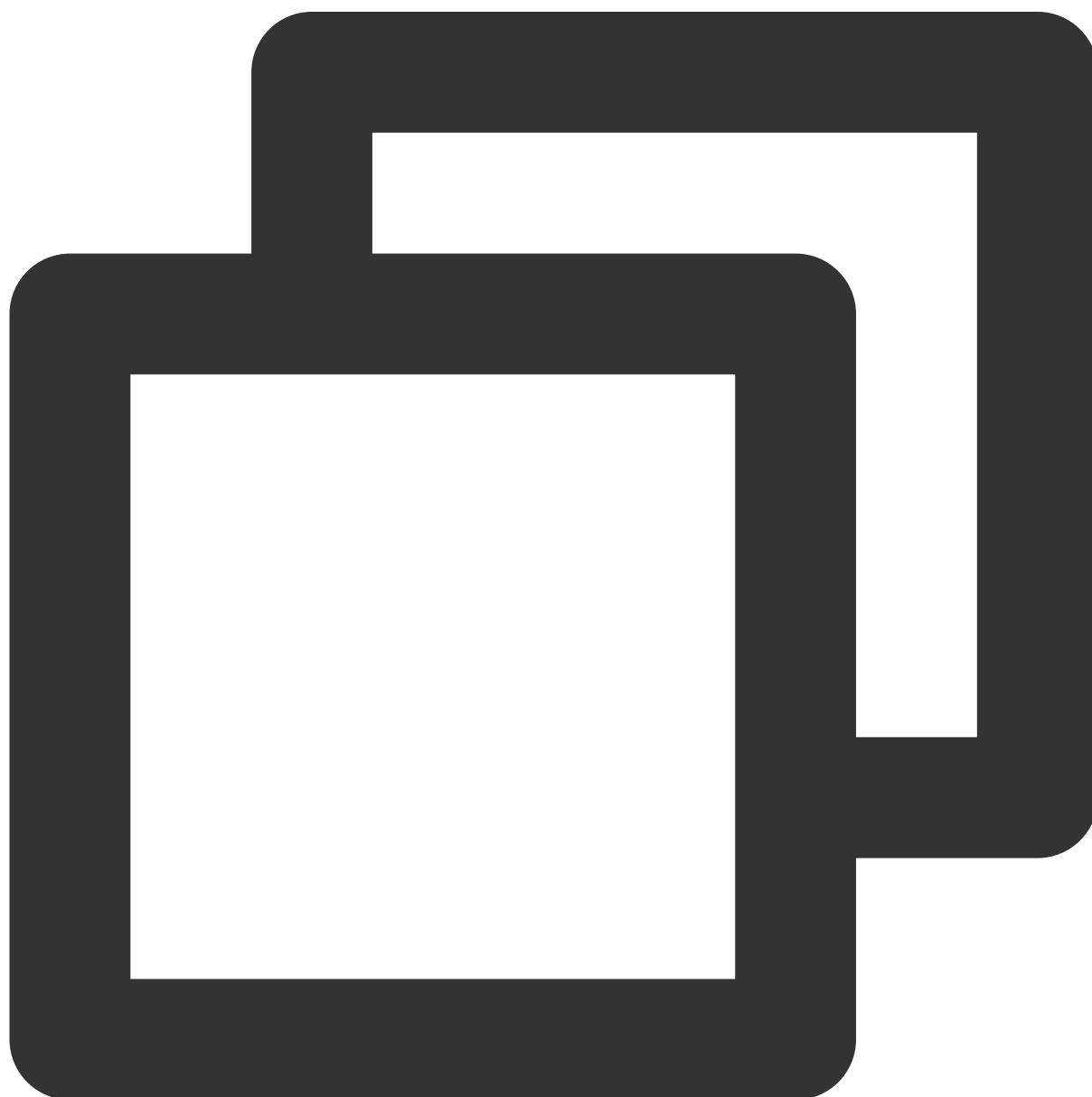
The parameters are described below:

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

|        |        |         |
|--------|--------|---------|
| roomId | String | Room ID |
|--------|--------|---------|

## onRoomInfoChange

Callback for change of room information. This callback is usually used to notify users of room status change in co-anchoring and cross-room communication scenarios.



```
void onRoomInfoChange (TRTCLiveRoomDef.TRTCLiveRoomInfo roomInfo);
```

The parameters are described below:

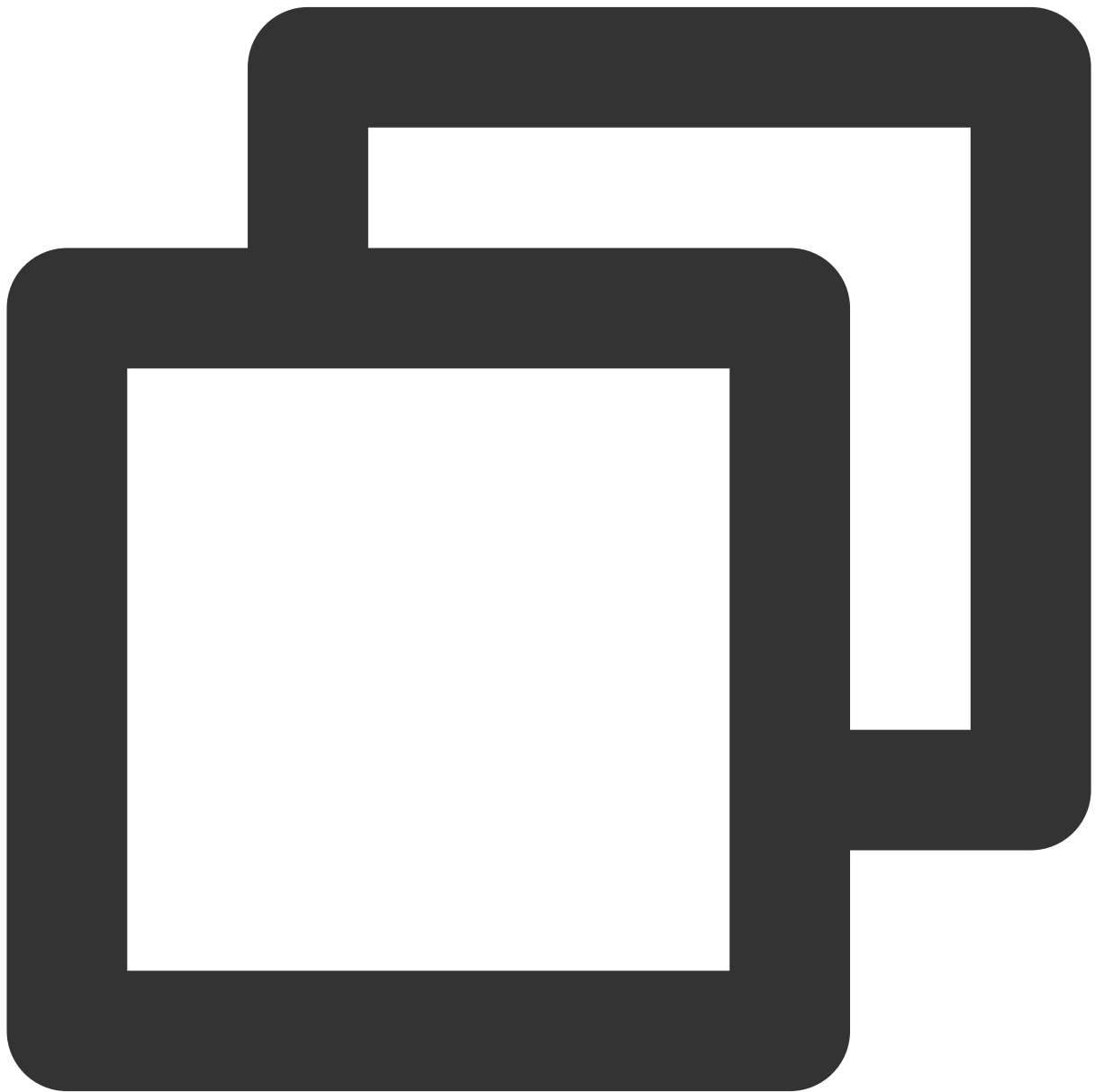
|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter | Type             | Description      |
|-----------|------------------|------------------|
| roomInfo  | TRTCLiveRoomInfo | Room information |

## Callback APIs for Entry/Exit of Anchors/Audience

### onAnchorEnter

Callback for a new anchor/co-anchoring viewer. The audience will receive this callback when there is a new anchor/co-anchoring viewer in the room and can call `startPlay()` of `TRTCLiveRoom` to play the video of the anchor/co-anchoring viewer.



```
void onAnchorEnter(String userId);
```

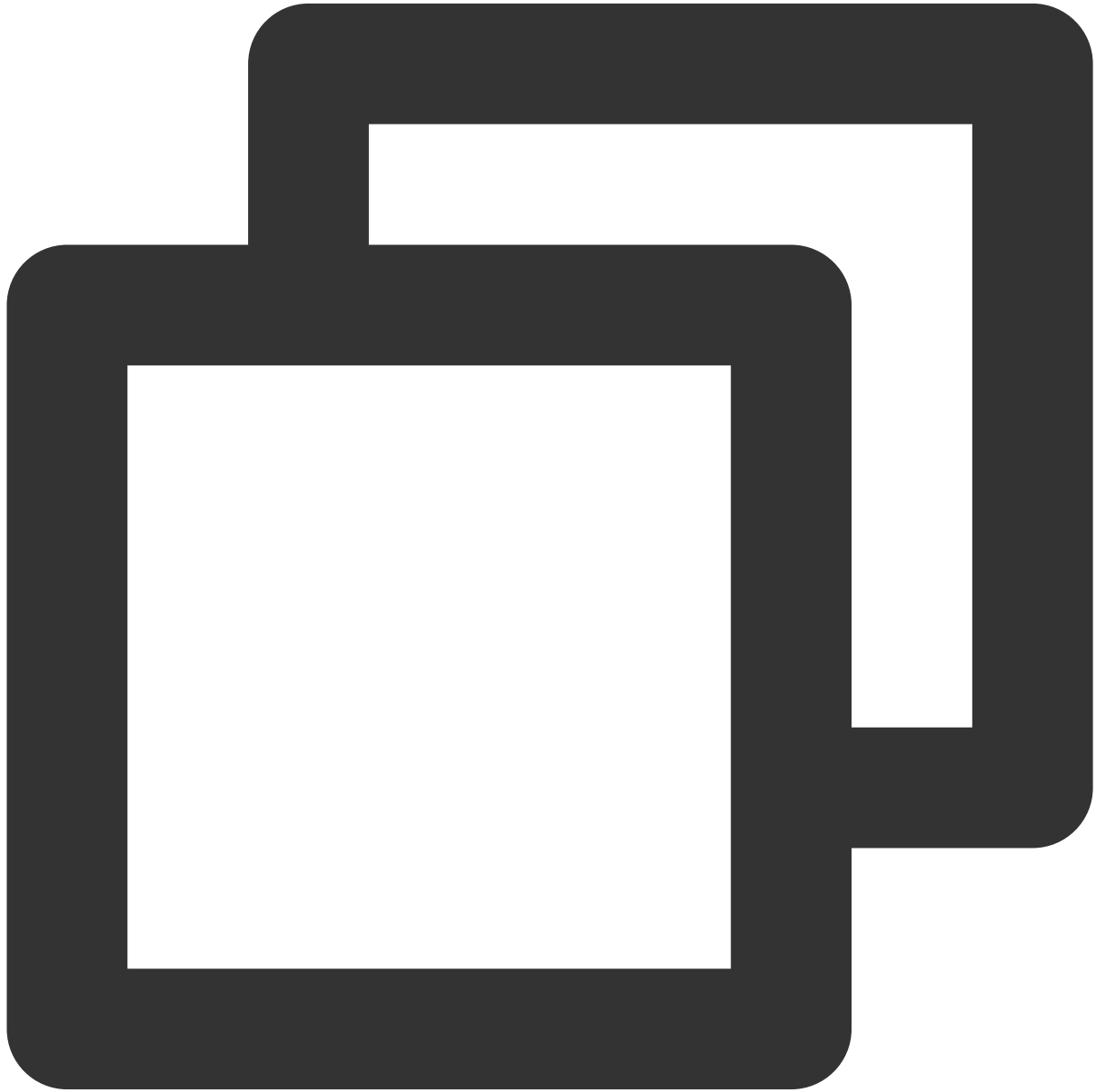
The parameters are described below:

| Parameter | Type   | Description                                   |
|-----------|--------|-----------------------------------------------|
| userId    | String | User ID of the new anchor/co-anchoring viewer |

## onAnchorExit



Callback for the quitting of an anchor/co-anchoring viewer. The anchors and co-anchoring viewers in a room will receive this callback after an anchor/co-anchoring viewer quits cross-room communication/co-anchoring and can call `stopPlay()` of `TRTCLiveRoom` to stop playing the video of the anchor/co-anchoring viewer.



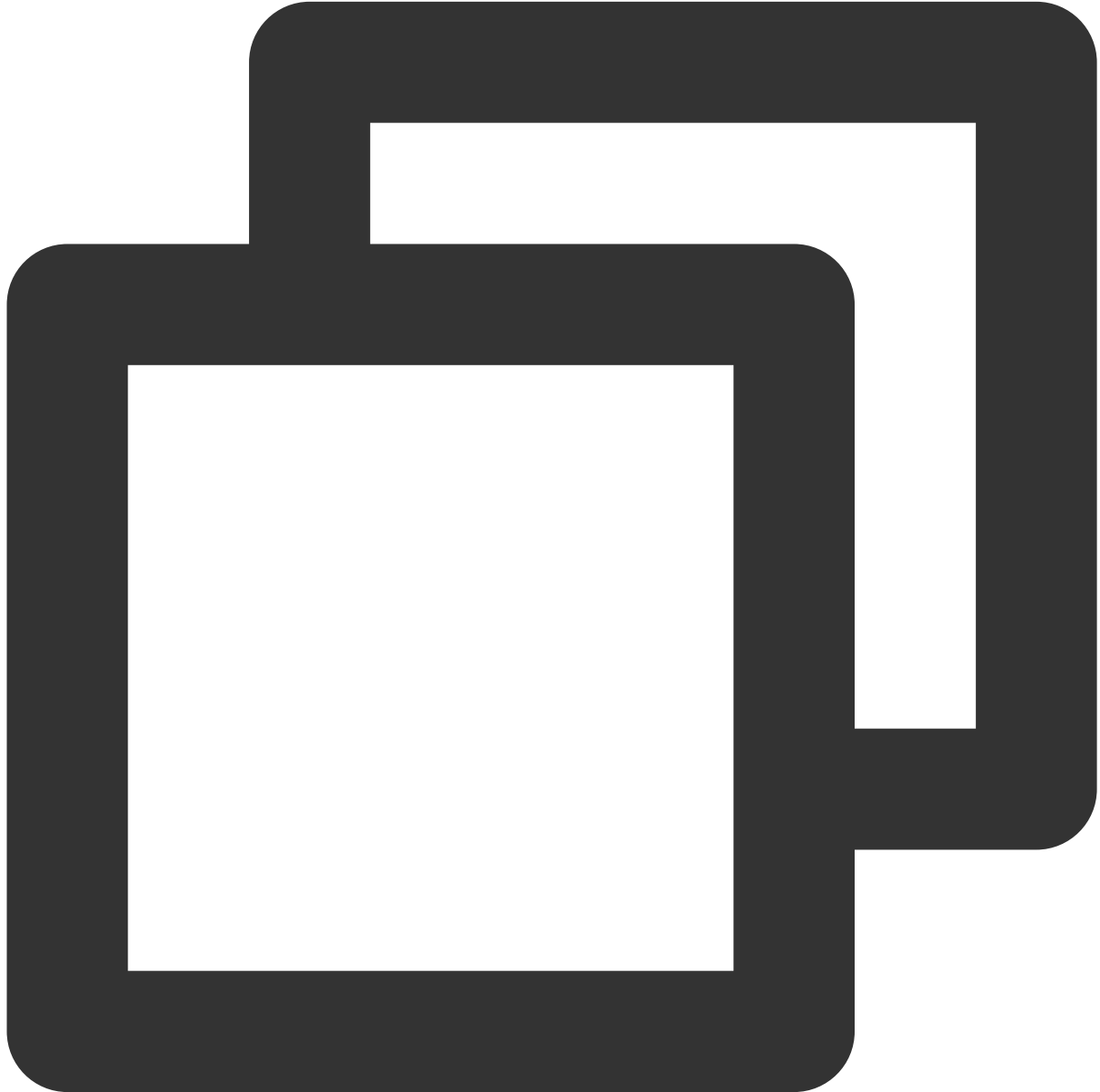
```
void onAnchorExit(String userId);
```

The parameters are described below:

| Parameter | Type   | Description             |
|-----------|--------|-------------------------|
| userId    | String | ID of the user who quit |

## onAudienceEnter

Callback for room entry by a viewer.



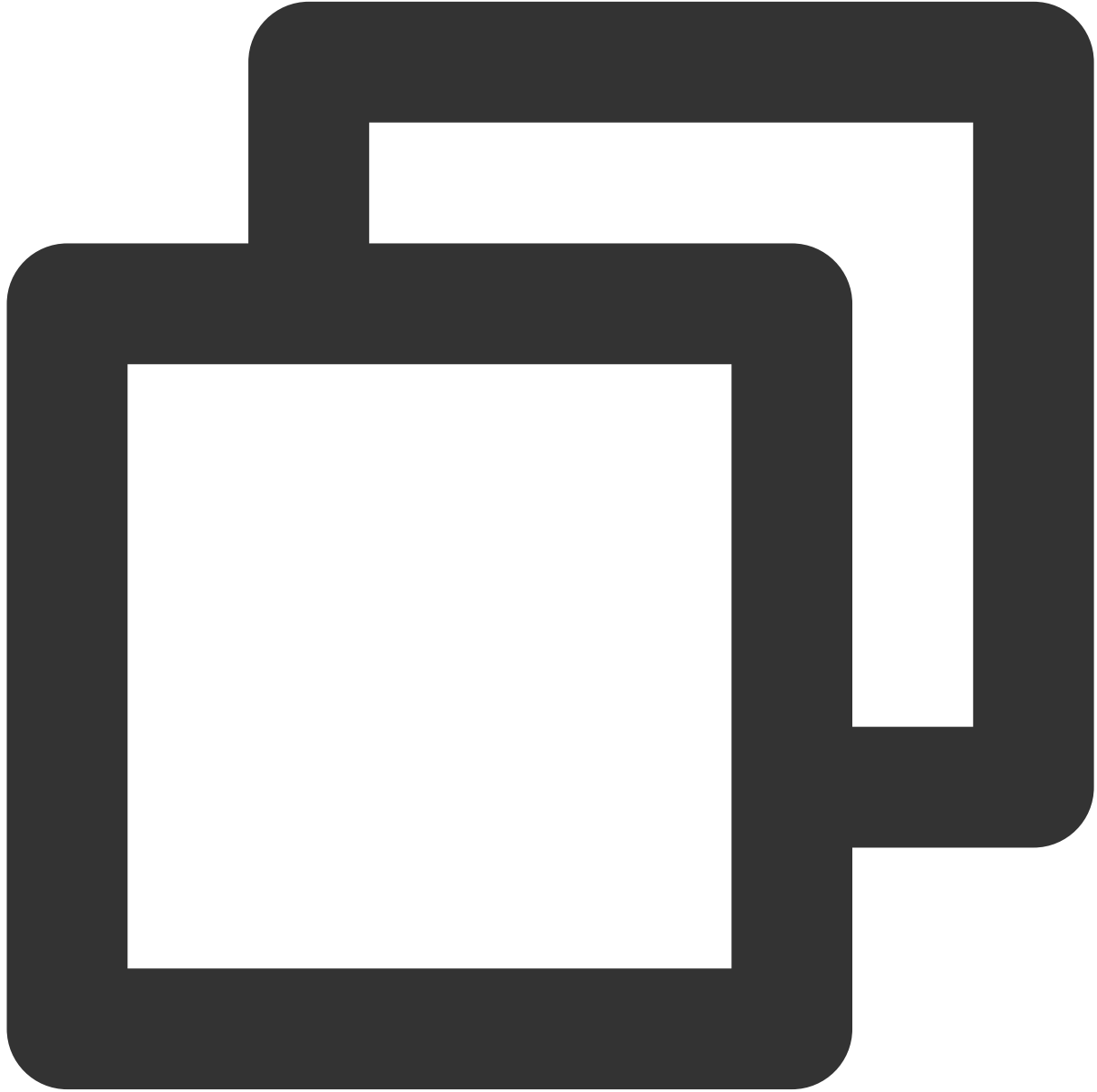
```
void onAudienceEnter (TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

The parameters are described below:

| Parameter | Type             | Description                                    |
|-----------|------------------|------------------------------------------------|
| userInfo  | TRTCLiveUserInfo | Information of the viewer who entered the room |

## onAudienceExit

Callback for room exit by a viewer.



```
void onAudienceExit (TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

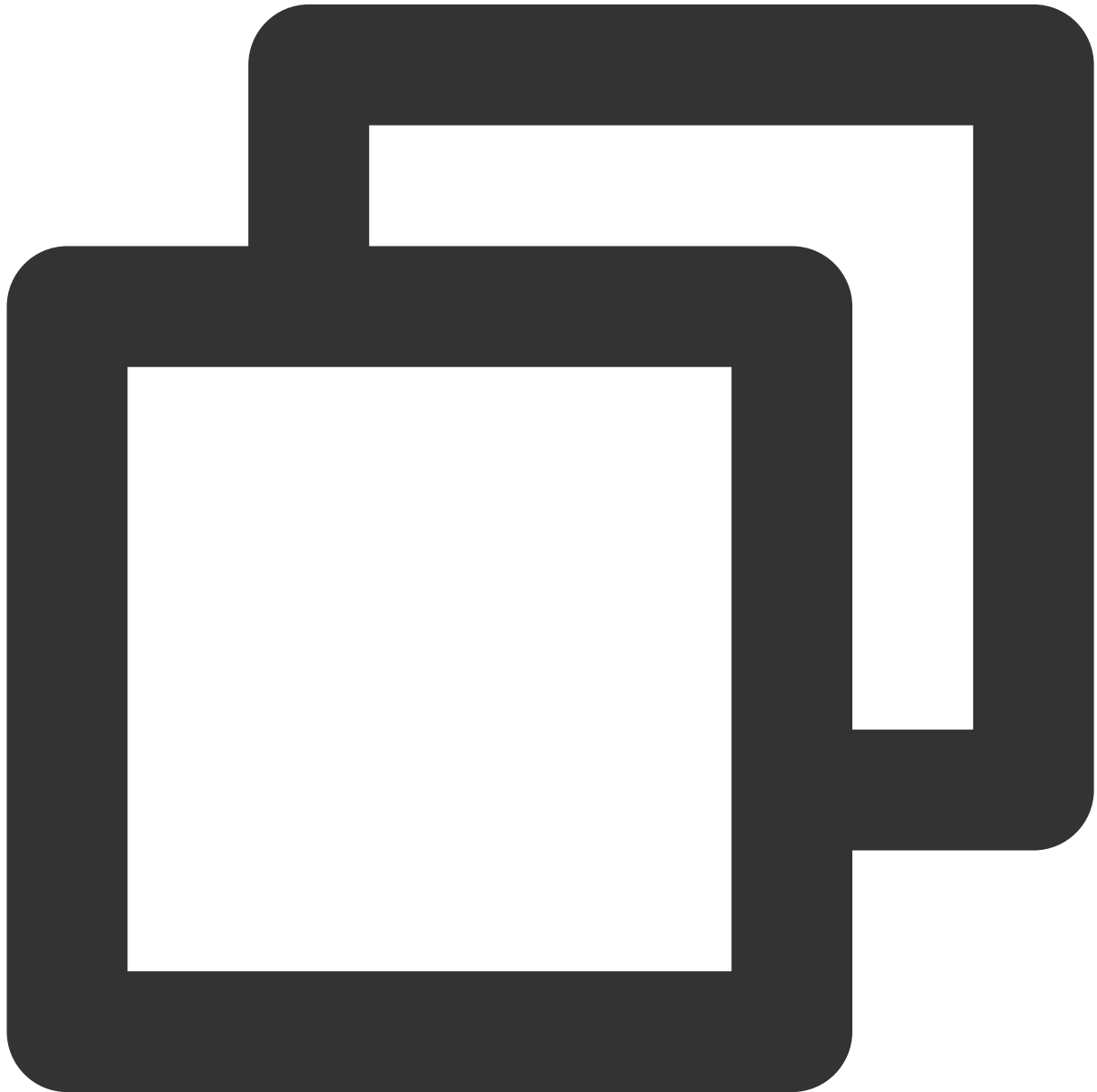
The parameters are described below:

| Parameter | Type             | Description                                 |
|-----------|------------------|---------------------------------------------|
| userInfo  | TRTCLiveUserInfo | Information of the viewer who left the room |

# Callback APIs Audience-Anchor Co-anchoring Events

## onRequestJoinAnchor

Callback for receiving a co-anchoring request from a viewer.



```
void onRequestJoinAnchor(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, String reason,
```

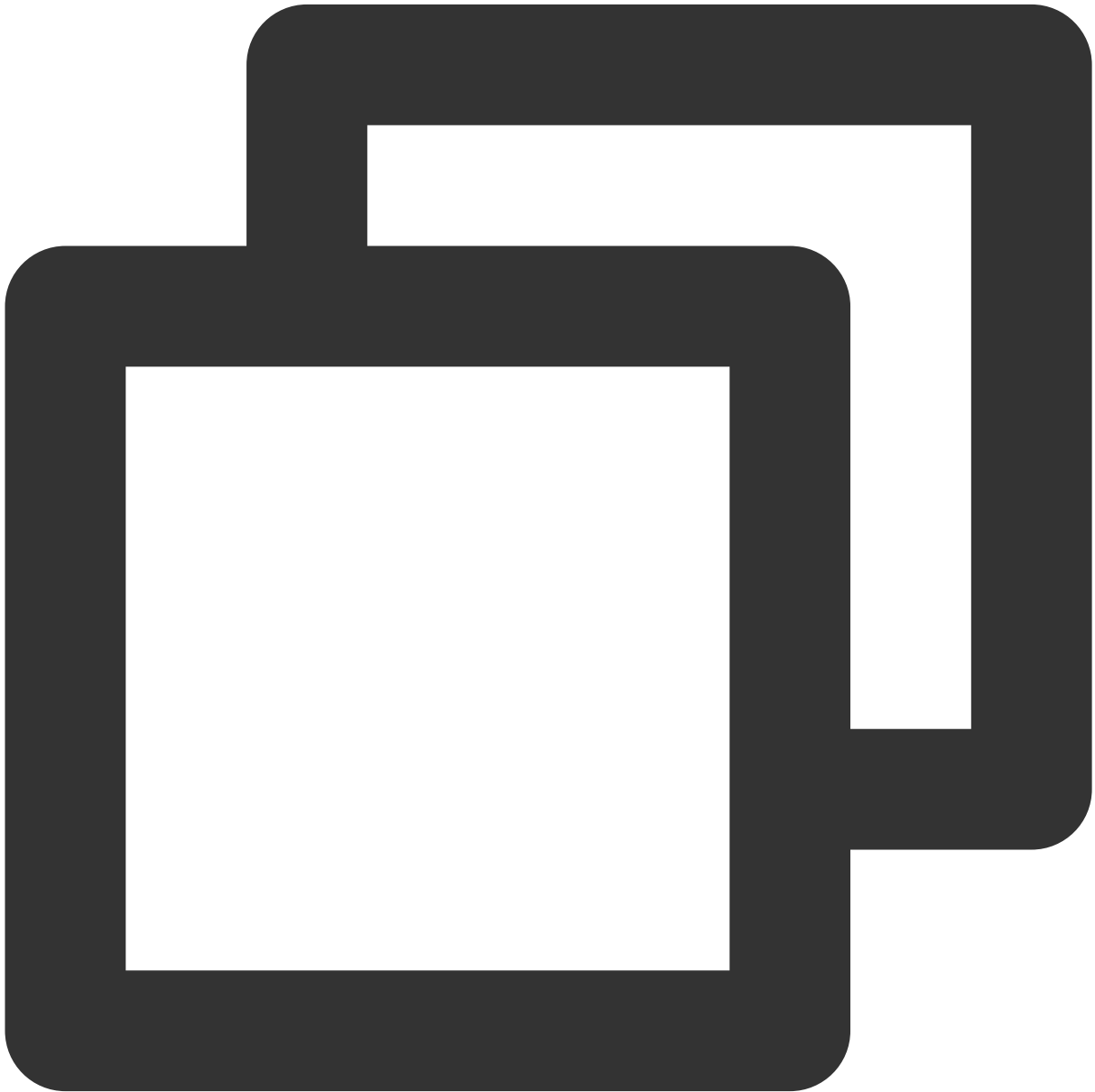
The parameters are described below:

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

|          |                  |                                                                                                                                                   |
|----------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| userInfo | TRTCLiveUserInfo | Information of the viewer who requested co-anchoring                                                                                              |
| reason   | String           | Reason for co-anchoring                                                                                                                           |
| timeout  | int              | Timeout period for response from the anchor. If the anchor does not respond to the request within the period, it will be discarded automatically. |

onKickoutJoinAnchor

Callback for being removed from co-anchoring. After receiving this callback, a co-anchoring viewer needs to call `stopPublish()` of `TRTCLiveRoom` to quit co-anchoring.



```
void onKickoutJoinAnchor();
```

## Callback APIs for Cross-Room Communication Events

### **onRequestRoomPK**

Callback for receiving a cross-room communication request. If an anchor accepts the request, he or she should wait for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and call `startPlay()` to play the other anchor's video.



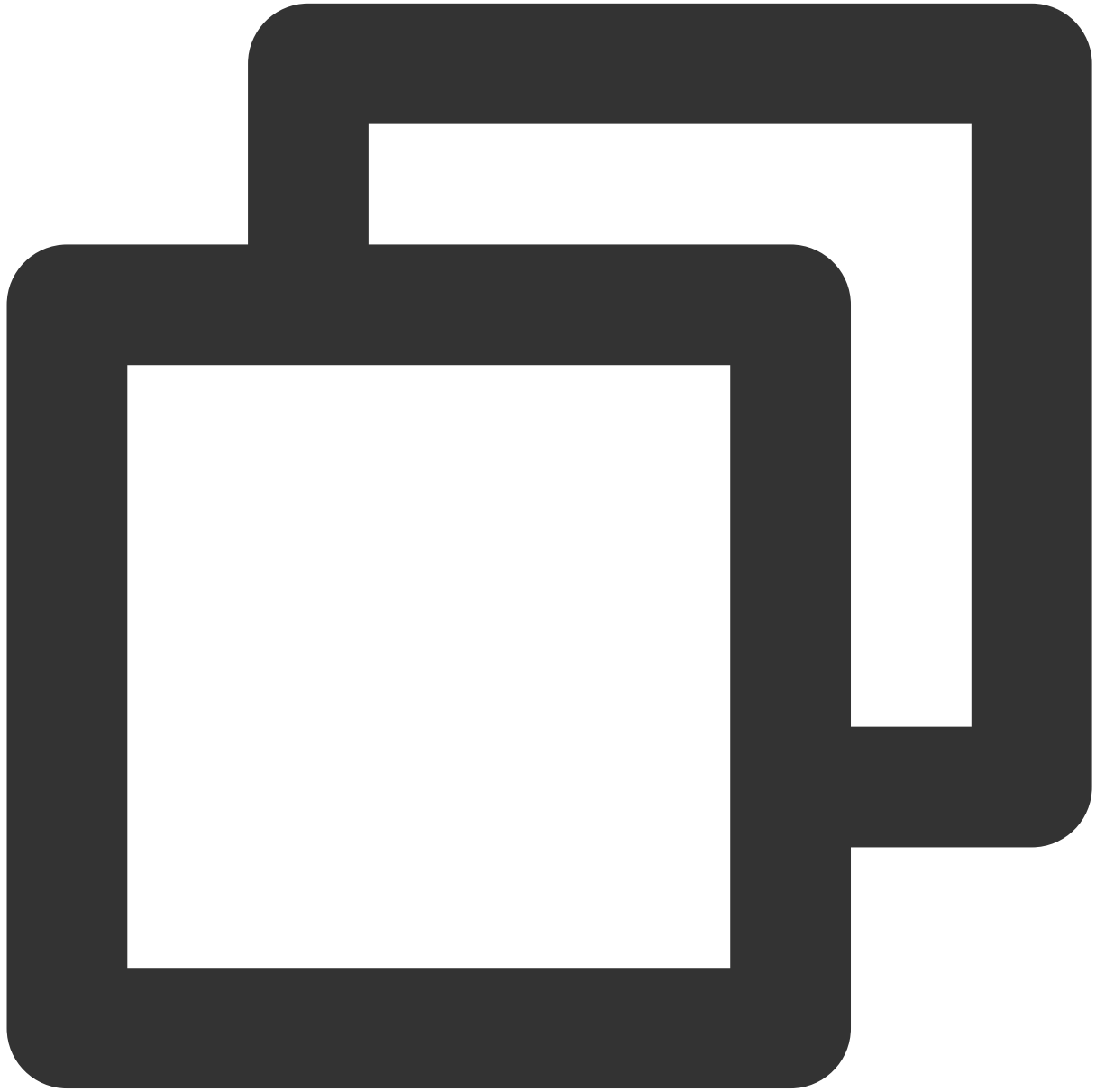
```
void onRequestRoomPK(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, int timeout);
```

The parameters are described below:

| Parameter | Type             | Description                                                       |
|-----------|------------------|-------------------------------------------------------------------|
| userInfo  | TRTCLiveUserInfo | Information of the anchor who requested cross-room communication. |
| timeout   | int              | Timeout period for response from the anchor                       |

### onQuitRoomPK

Callback for ending cross-room communication.



```
void onQuitRoomPK();
```

## Message Event Callback APIs

### **onRecvRoomTextMsg**

Callback for receiving a text chat message.





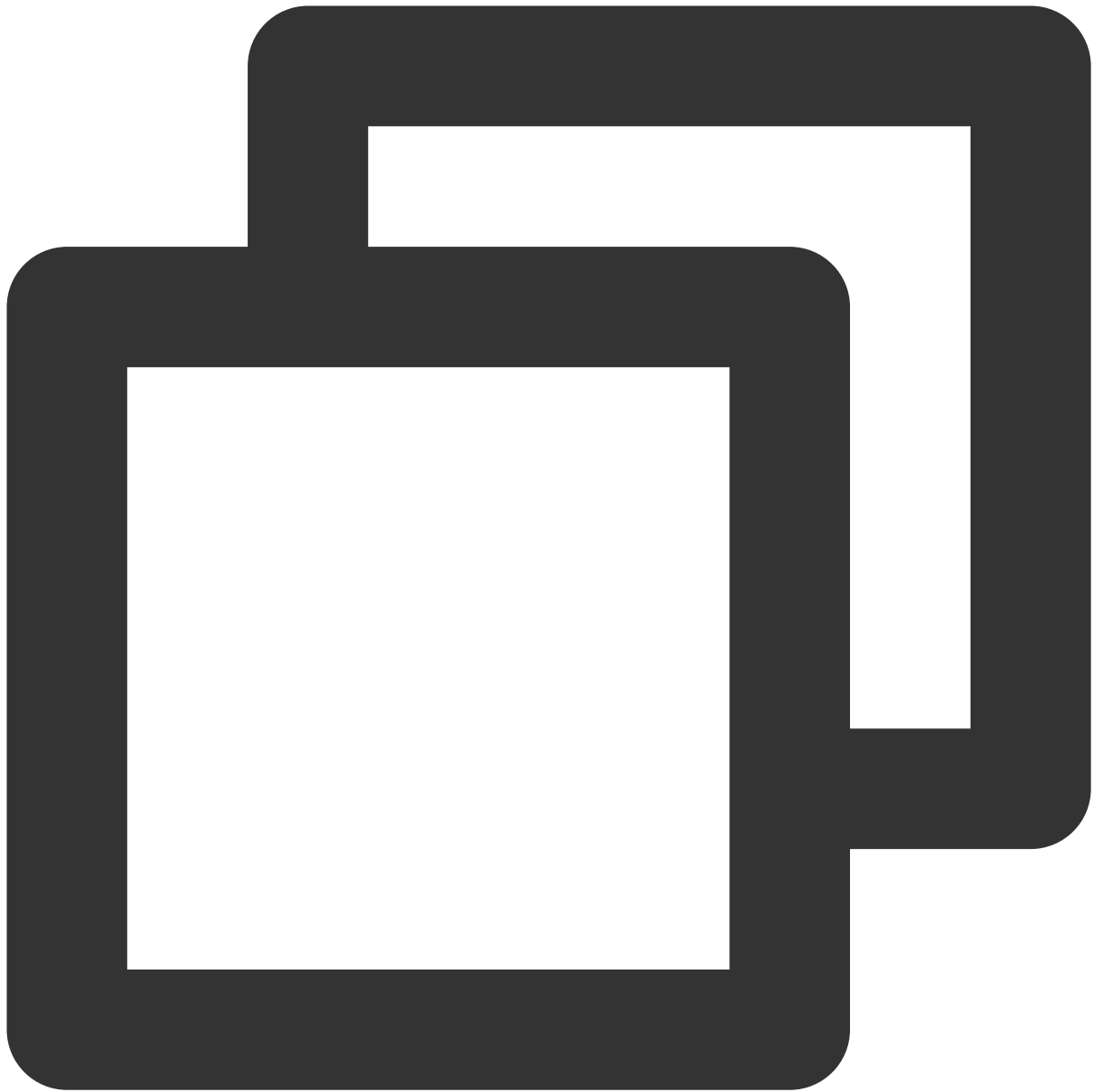
```
void onRecvRoomTextMsg(String message, TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

The parameters are described below:

| Parameter | Type             | Description               |
|-----------|------------------|---------------------------|
| message   | String           | Text message              |
| userInfo  | TRTCLiveUserInfo | Information of the sender |

### **onRecvRoomCustomMsg**

A custom message was received.



```
void onRecvRoomCustomMsg(String cmd, String message, TRTCLiveRoomDef.TRTCLiveUserIn
```

The parameters are described below:

| Parameter | Type   | Description                                                               |
|-----------|--------|---------------------------------------------------------------------------|
| command   | String | A custom command word used to distinguish between different message types |
| message   | String | Text message                                                              |

userInfo

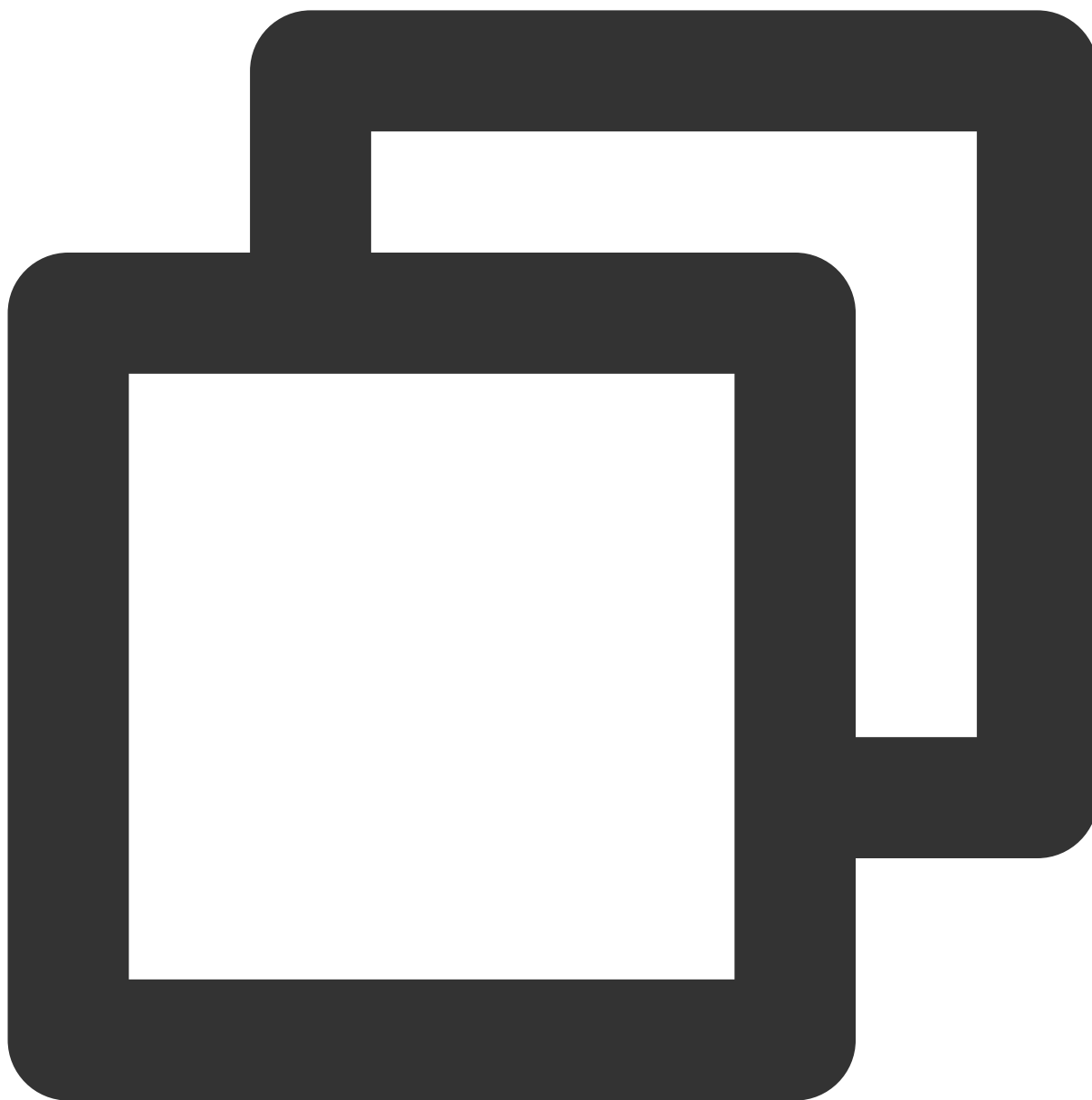
TRTCLiveUserInfo

Information of the sender

## TRTCAudioEffectManager

### playBGM

Background Music



```
void playBGM(String url, int loopTimes, int bgmVol, int micVol, TRTCCloud.BGMNotify
```

The parameters are described below:

| Parameter | Type                | Description                |
|-----------|---------------------|----------------------------|
| url       | String              | Path of the music file     |
| loopTimes | int                 | Loop times                 |
| bgmVol    | int                 | Volume of background music |
| micVol    | int                 | Audio capturing volume     |
| notify    | TRTCCloud.BGMNotify | Playback notification      |

## stopBGM

stop playing background music



```
void stopBGM();
```

### **pauseBGM**

pause background music



```
void pauseBGM();
```

### **resumeBGM**

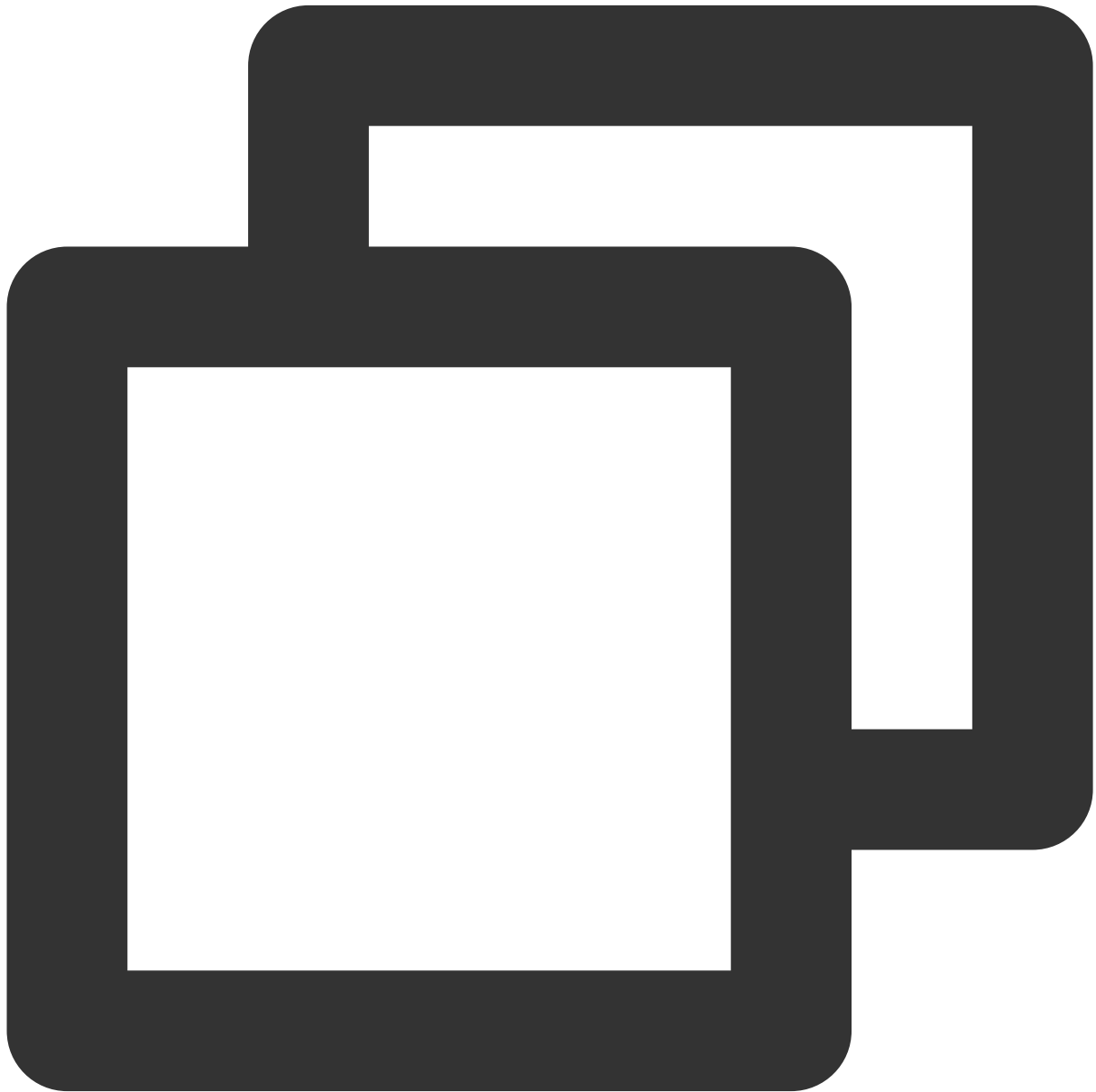
This API is used to resume background music playback.



```
void resumeBGM();
```

### **setBGMVolume**

This API is used to set the playback volume of background music.



```
void setBGMVolume(int volume);
```

The parameters are described below:

| Parameter | Type | Description                                    |
|-----------|------|------------------------------------------------|
| volume    | int  | Volume. Value range: 0-100. Default value: 100 |

## setBGMPosition

This API is used to set the background music playback progress.





```
int setBGMPosition(int position);
```

The parameters are described below:

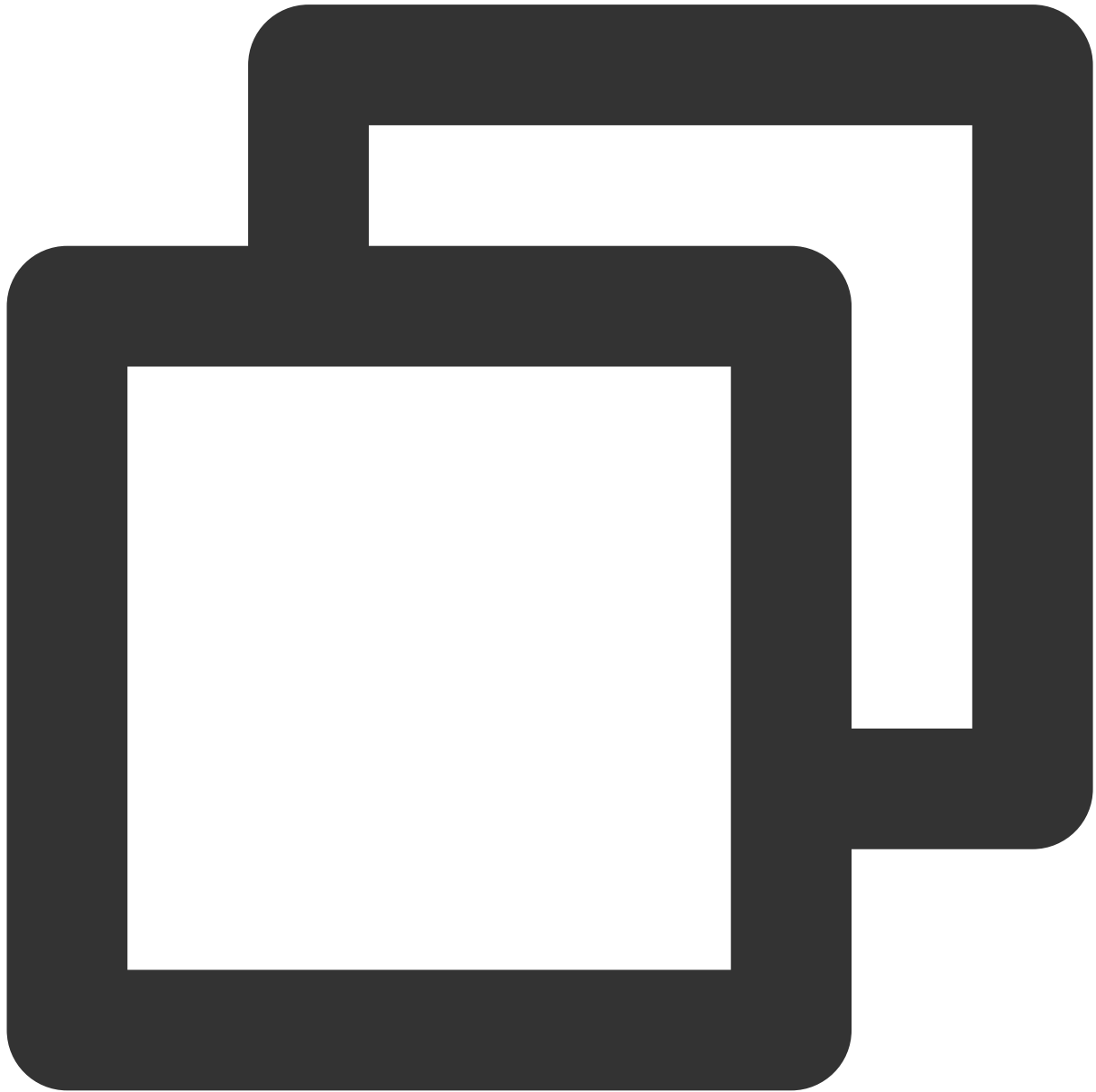
| Parameter | Type | Description                                                |
|-----------|------|------------------------------------------------------------|
| position  | int  | Playback progress of background music in milliseconds (ms) |

#### Return code

0 : successful

## setMicVolume

This API is used to set the mic volume. It can be used to control the volume of the mic when background music is mixed.



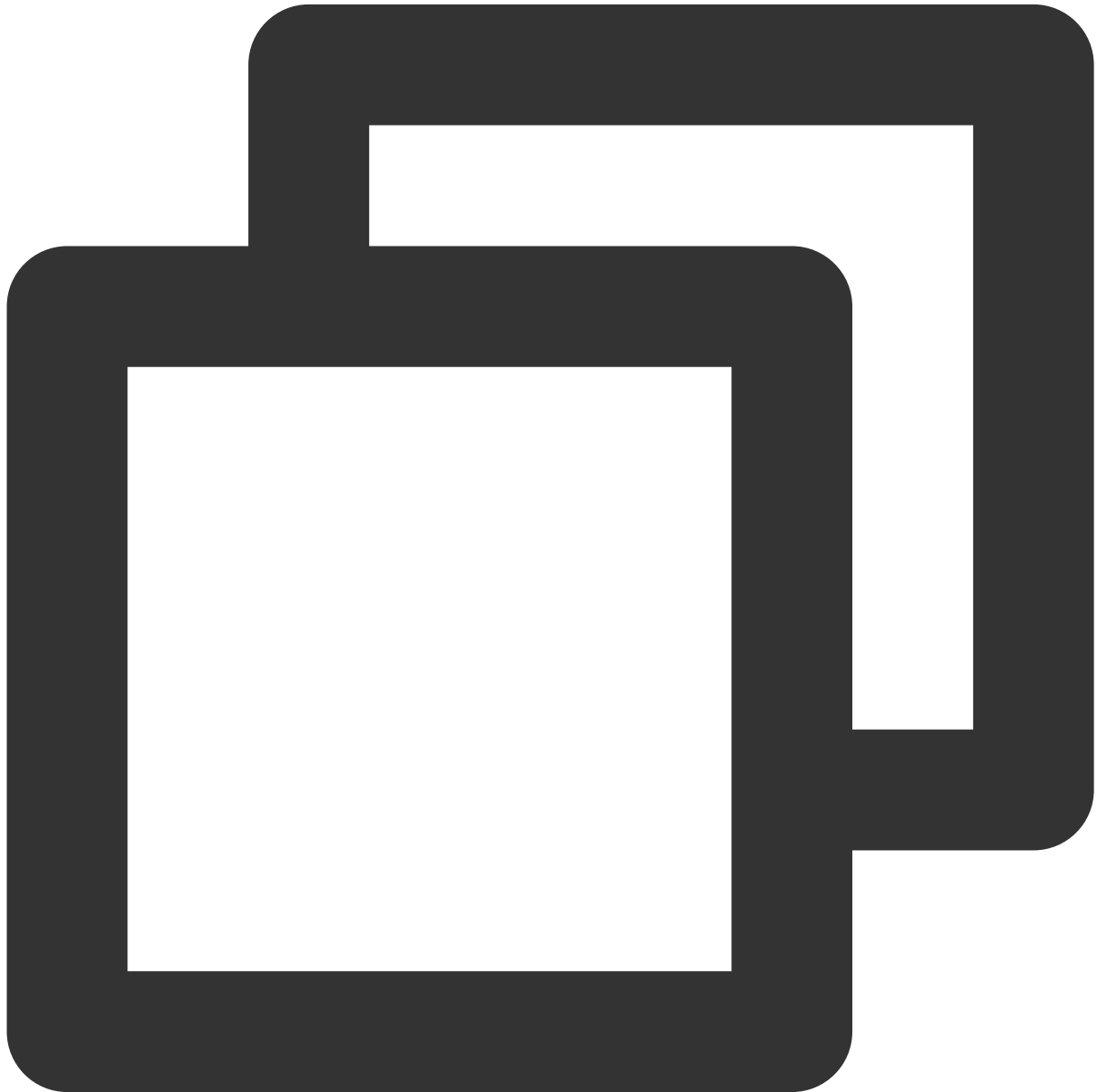
```
void setMicVolume(int volume);
```

The parameters are described below:

| Parameter | Type | Description                                    |
|-----------|------|------------------------------------------------|
| volume    | Int  | Volume. Value range: 0-100. Default value: 100 |

## setReverbType

This API is used to set the reverb effect.



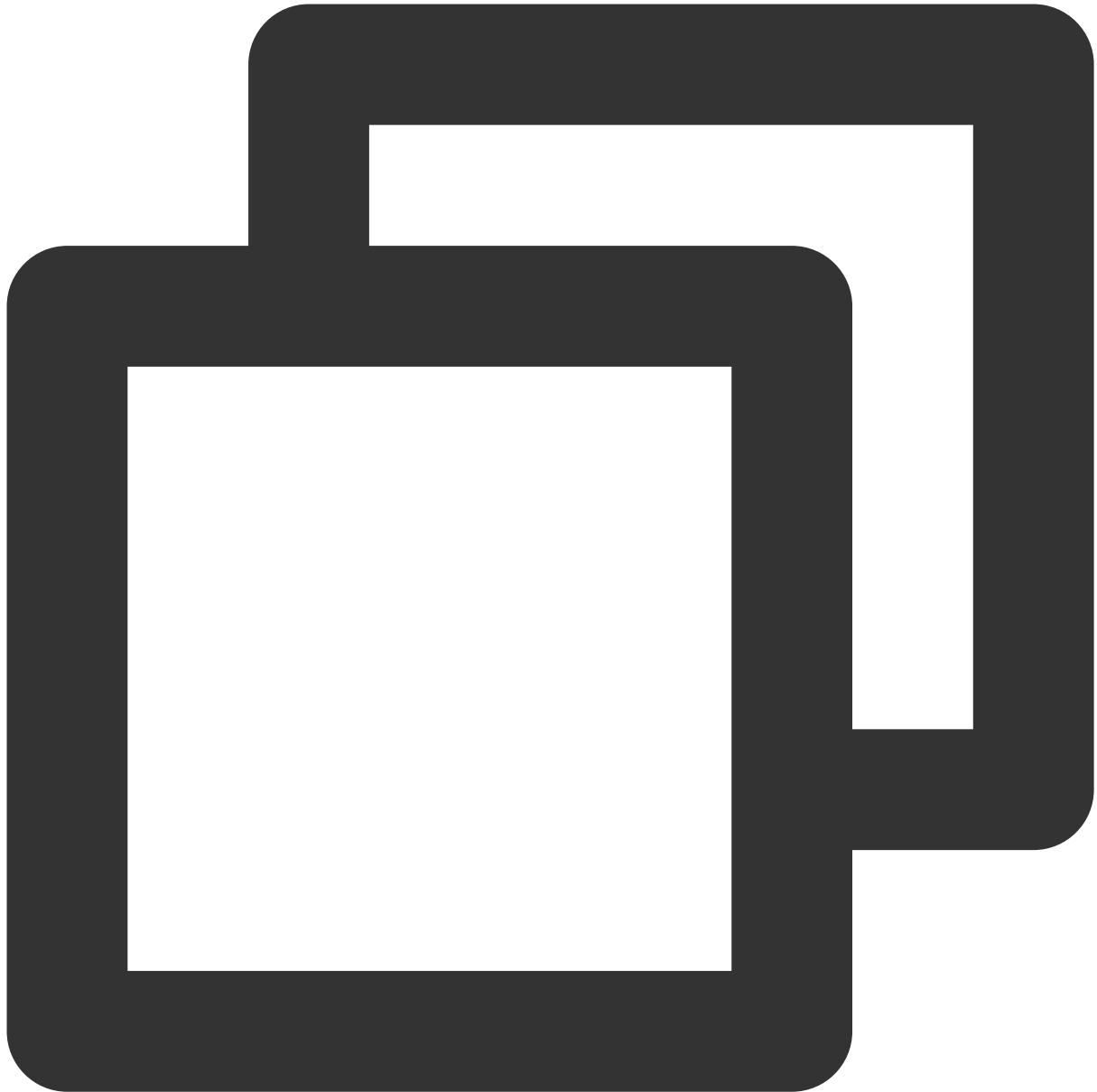
```
void setReverbType(int reverbType);
```

The parameters are described below:

| Parameter  | Type | Description                                                                                                               |
|------------|------|---------------------------------------------------------------------------------------------------------------------------|
| reverbType | int  | Reverb effect. For details, please see the definitions of <a href="#">TRTC_REVERB_TYPE</a> in <code>TRTCCloudDef</code> . |

## setVoiceChangerType

This API is used to set the voice changing effect.



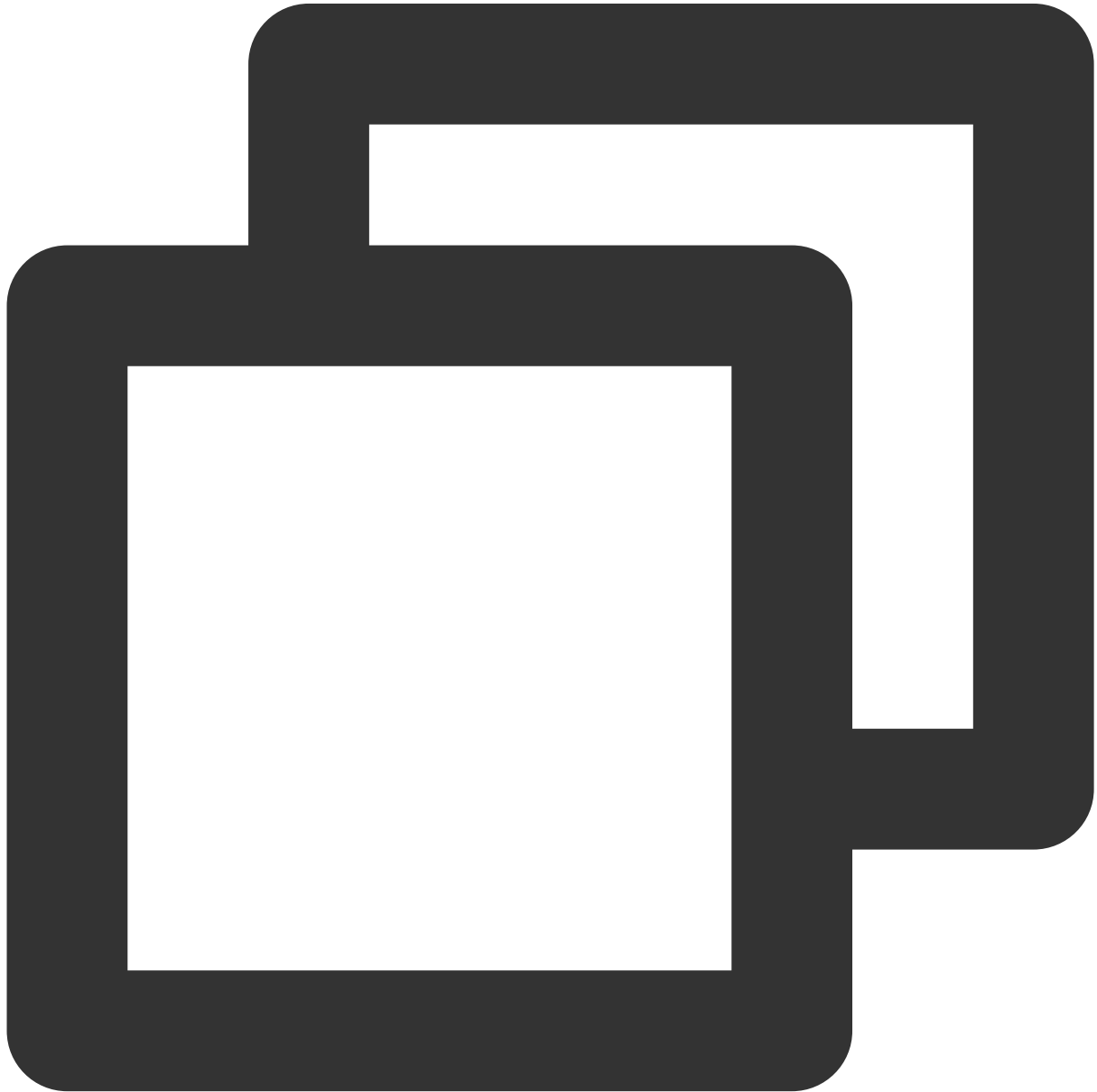
```
void setVoiceChangerType(int type);
```

The parameters are described below:

| Parameter | Type | Description                                                                                                                              |
|-----------|------|------------------------------------------------------------------------------------------------------------------------------------------|
| type      | int  | Voice changing effect. For details, please see the definitions of <a href="#">TRTC_VOICE_CHANGER_TYPE</a> in <code>TRTCCloudDef</code> . |

## playAudioEffect

This API is used to play an audio effect. For each audio effect, you need to assign an ID, which is used to start and stop the playback of an audio effect as well as set its playback volume. Supported formats include AAC, MP3, and M4A.



```
void playAudioEffect(int effectId, String path, int count, boolean publish, int vol
```

The parameters are described below:

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

|          |         |                                                                                                                                             |
|----------|---------|---------------------------------------------------------------------------------------------------------------------------------------------|
| effectId | int     | Audio effect ID                                                                                                                             |
| path     | String  | Audio effect path                                                                                                                           |
| count    | int     | Loop times                                                                                                                                  |
| publish  | boolean | Whether to push the audio effect. <code>true</code> : push the effect to remote users; <code>false</code> : preview the effect locally only |
| volume   | int     | Volume. Value range: 0-100. Default value: <code>100</code>                                                                                 |

## pauseAudioEffect

This API is used to pause playing an audio effect.



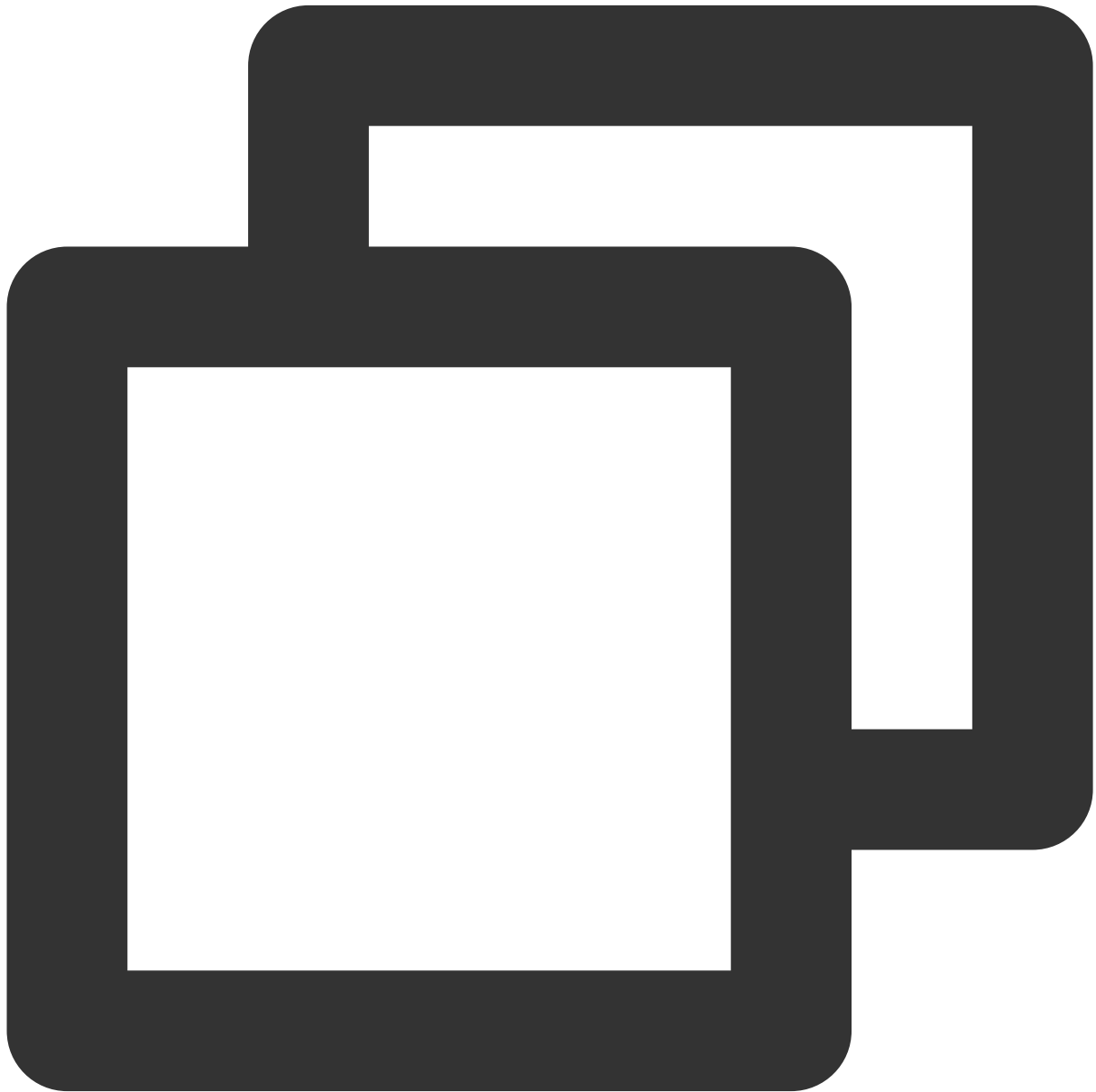
```
void pauseAudioEffect (int effectId);
```

The parameters are described below:

| Parameter | Type | Description     |
|-----------|------|-----------------|
| effectId  | int  | Audio effect ID |

## resumeAudioEffect

This API is used to resume playing an audio effect.



```
void resumeAudioEffect(int effectId);
```

The parameters are described below:

| Parameter | Type | Description     |
|-----------|------|-----------------|
| effectId  | int  | Audio effect ID |

## stopAudioEffect

This API is used to stop playing an audio effect.





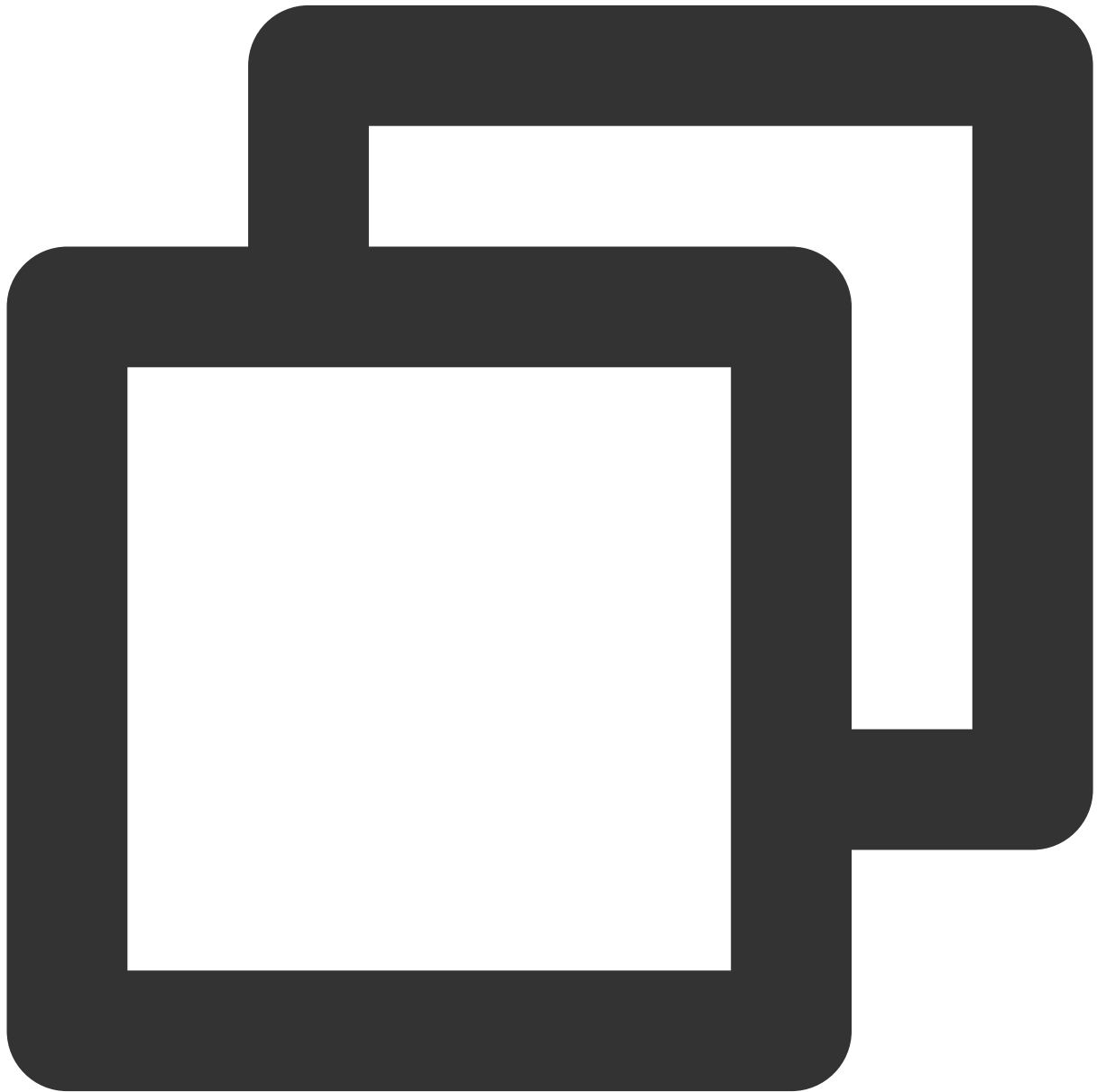
```
void stopAudioEffect(int effectId);
```

The parameters are described below:

| Parameter | Type | Description     |
|-----------|------|-----------------|
| effectId  | int  | Audio effect ID |

### **stopAllAudioEffects**

This API is used to stop playing all audio effects.



```
void stopAllAudioEffects();
```

### **setAudioEffectVolume**

This API is used to set the volume of an audio effect.



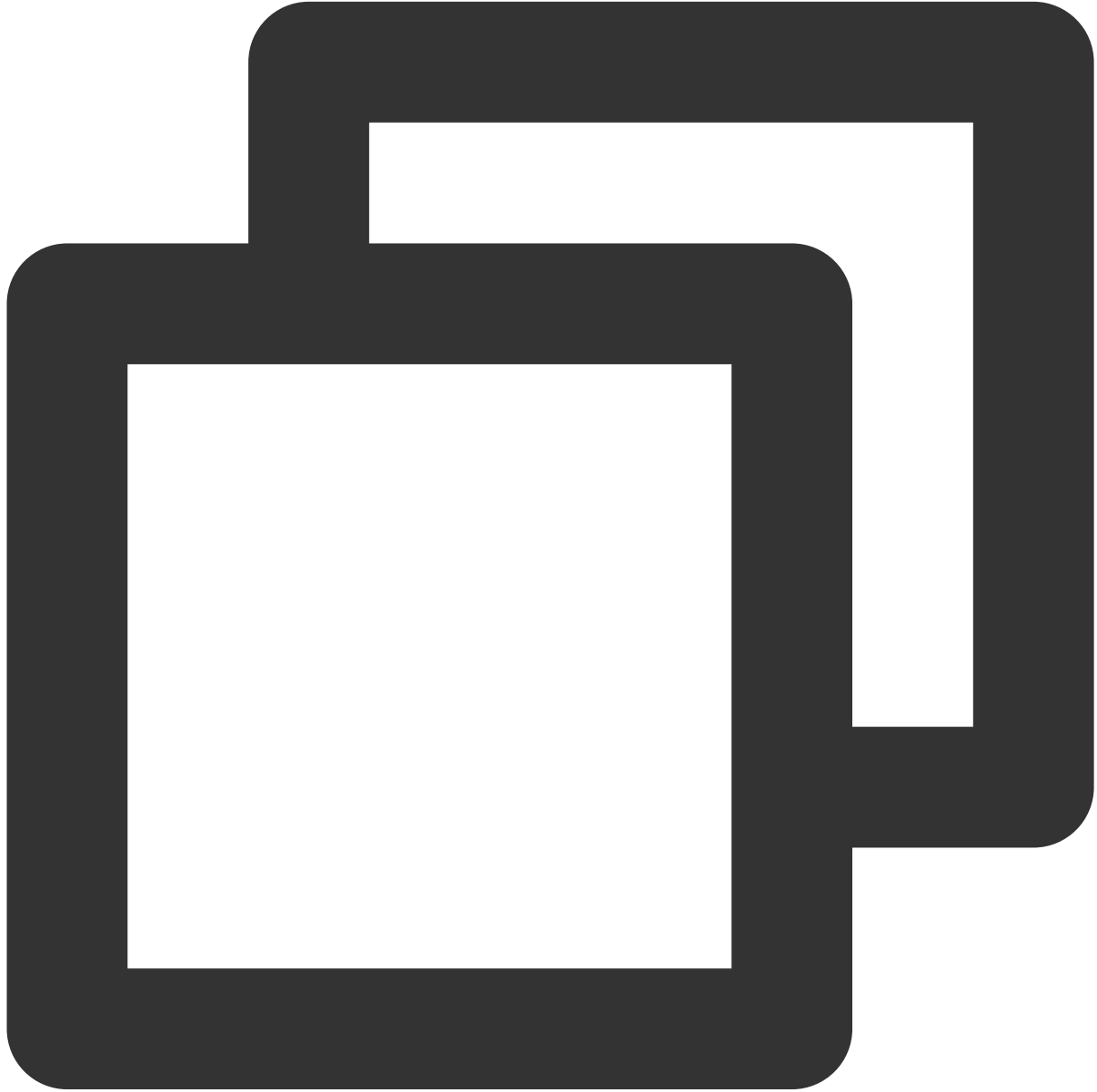
```
void setAudioEffectVolume(int effectId, int volume);
```

The parameters are described below:

| Parameter | Type | Description                                    |
|-----------|------|------------------------------------------------|
| effectId  | int  | Audio effect ID                                |
| volume    | int  | Volume. Value range: 0-100. Default value: 100 |

## setAllAudioEffectsVolume

This API is used to set the volume of all audio effects.



```
void setAllAudioEffectsVolume(int volume);
```

The parameters are described below:

| Parameter | Type | Description                                    |
|-----------|------|------------------------------------------------|
| volume    | int  | Volume. Value range: 0-100. Default value: 100 |



# Online Karaoke

## Quick Integration (TUIKaraoke)

### iOS

Last updated : 2023-09-21 16:22:21

## Overview

TUIKaraoke is an open-source audio/video UI component that you can integrate into your project to bring online karaoke, seat management, gift giving/receiving, text chat, and other TRTC features to your application. TUIKaraoke requires only a few lines of code and also supports the Android platform. Its basic features are shown below:

### Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK Trial Edition (which supports up to 100 DAUs) will also be activated automatically. For Chat billing details, see [Pricing](#).

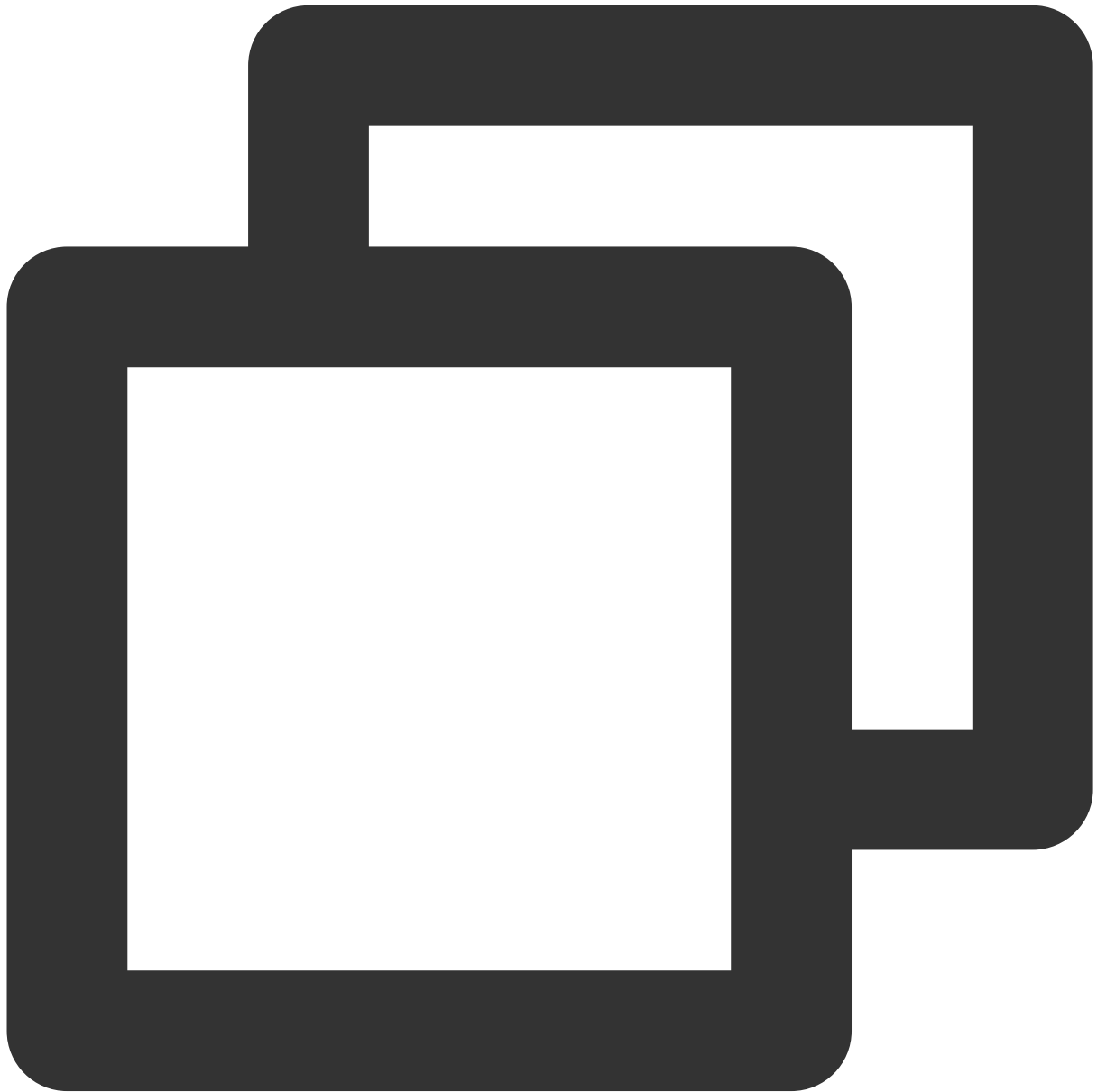


## Integration

### Step 1. Download and import the `TUIKaraoke` component

Go to [GitHub](#), clone or download the code, copy the `Source`, `Resources`, and `TXAppBasic` folders and the `TUIKaraoke.podspec` file in the `ios` directory to your project, and complete the following import operations:

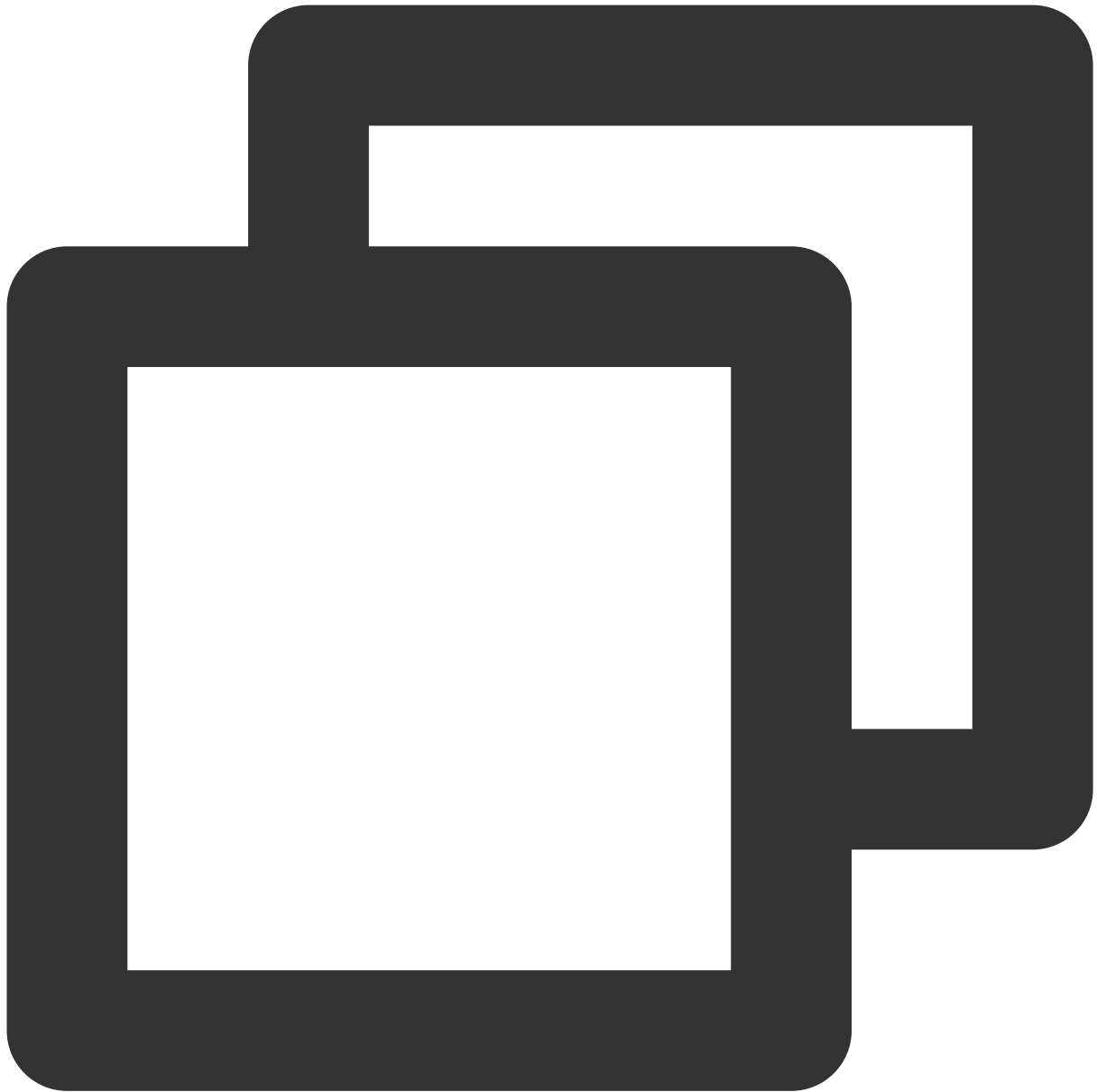
Add the following import commands to your `Podfile`:



```
pod 'TUIKaraoke', :path => "./", :subspecs => ["TRTC"]  
pod 'TXLiteAVSDK_TRTC'  
pod 'TXAppBasic', :path => "TXAppBasic/"
```

Open Terminal and run the following installation command in the directory of the Podfile :

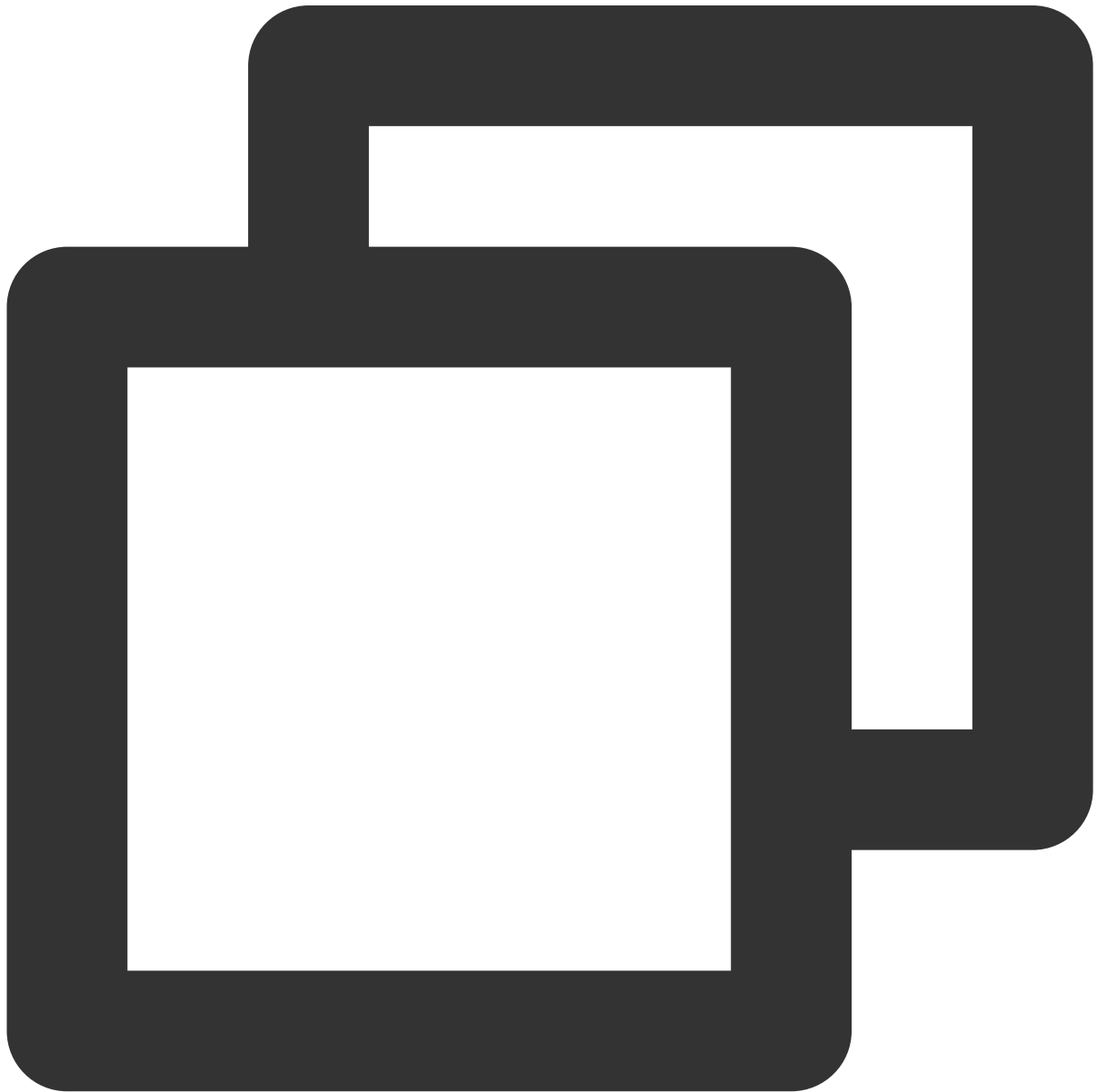




```
pod install
```

## Step 2. Configure permissions

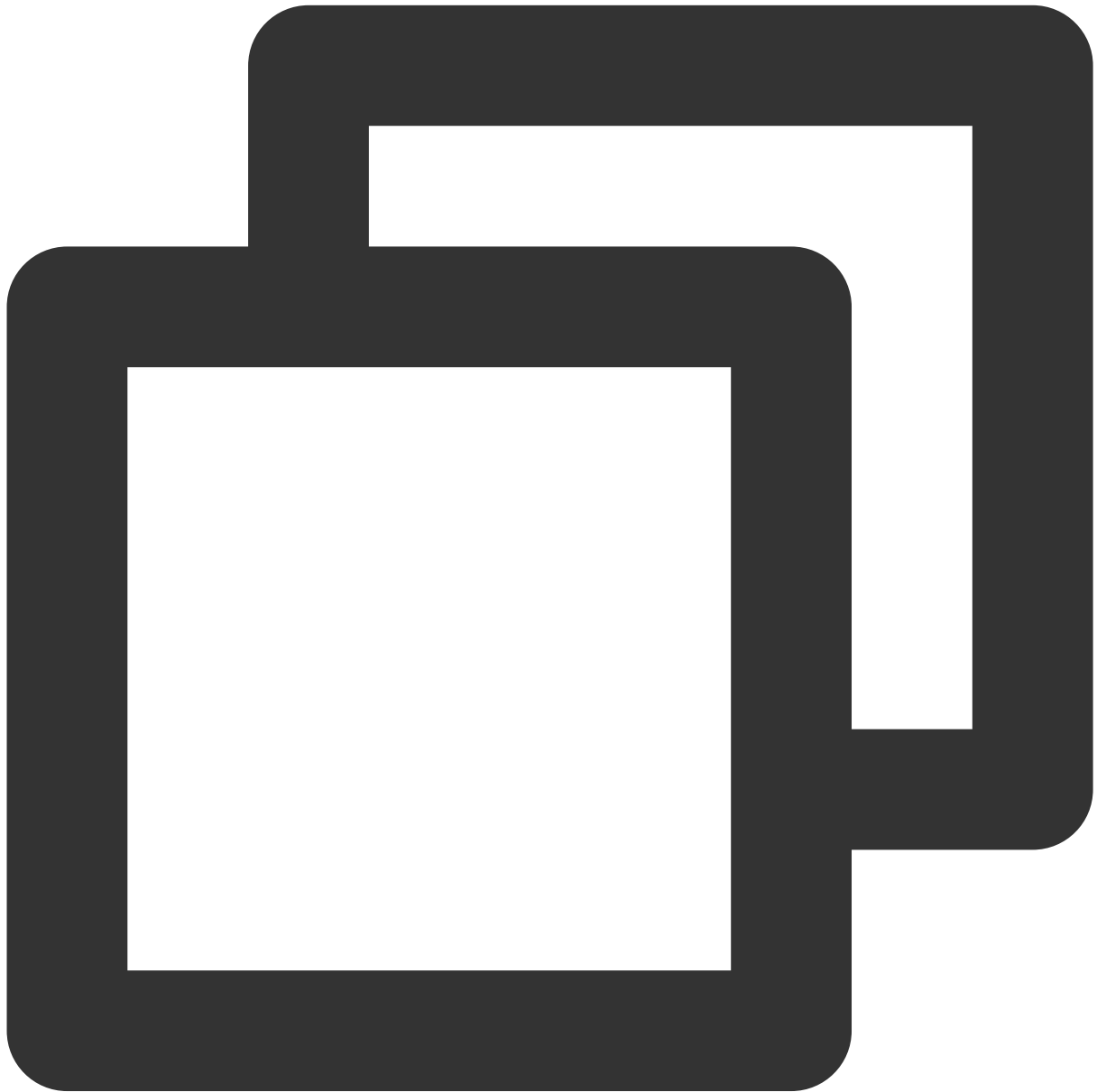
Configure permission requests for your app in the `info.plist` file of your project. The SDKs need the following permissions (on iOS, the mic access must be requested at runtime):



```
<key>NSMicrophoneUsageDescription</key>  
  <string>`TUIKaraoke` needs to access your mic.</string>
```

### Step 3. Initialize and log in to the component

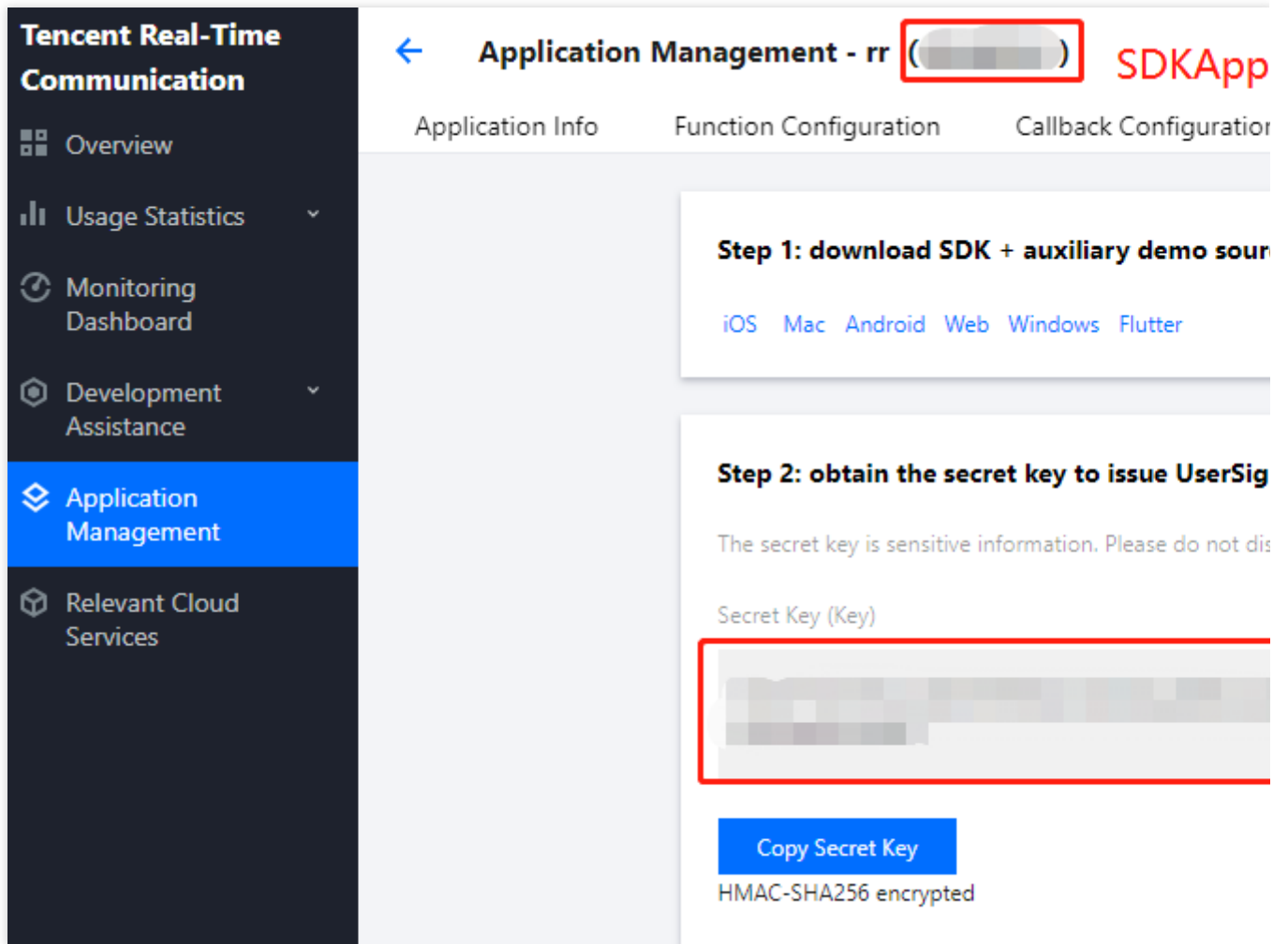
For more information on relevant APIs, see [TRTCKaraoke \(iOS\)](#).



```
// 1. Initialize
let karaokeRoom = TRTCKaraokeRoom.shared()
karaokeRoom.setDelegate(delegate: self)
// 2. Log in
karaokeRoom.login(SDKAppID: Int32(SDKAppID), UserId: UserId, UserSig: ProfileMana
    if code == 0 {
        // Logged in
    }
}
```

**Parameter description:**

**SDKAppID**: TRTC application ID. If you haven't activated TRTC, log in to the [TRTC console](#), create a TRTC application, click **Application Info**, and select the **Quick Start** tab to view its `SDKAppID`.



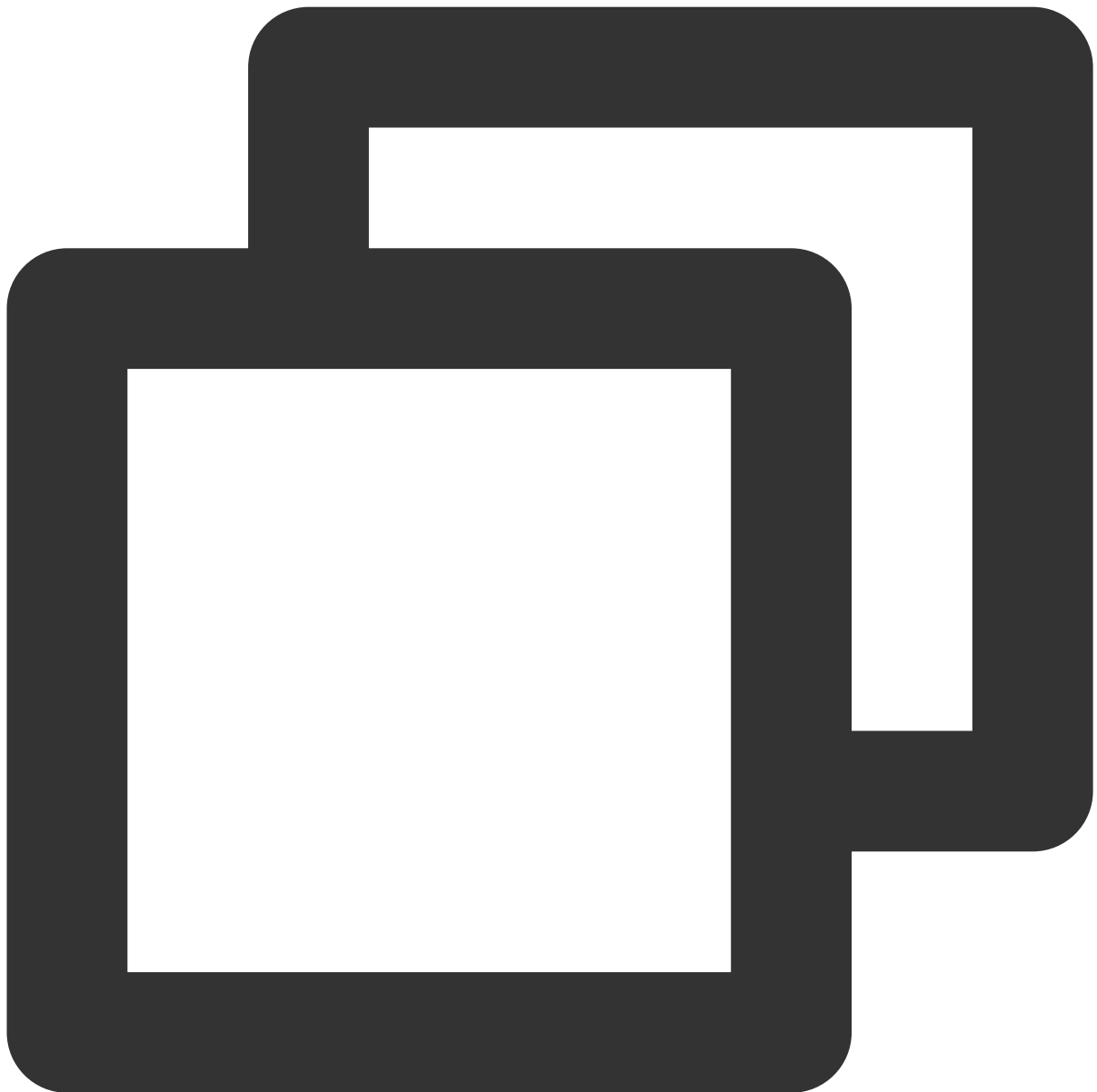
**SecretKey**: TRTC application key. Each secret key corresponds to an `SDKAppID`. You can view your application's secret key on the [Application Management](#) page of the TRTC console.

**userId**: Current user ID, which is a custom string that can contain up to 32 bytes of letters and digits (special characters are not supported).

**userSig**: The security protection signature calculated based on `SDKAppID`, `userId`, and `Secretkey`. You can click [here](#) to directly generate a debugging `userSig` online. For more information, see [UserSig](#).

## Step 4. Implement the online karaoke scenario

1. The room owner creates a room through `TUIKaraoke.createRoom`.

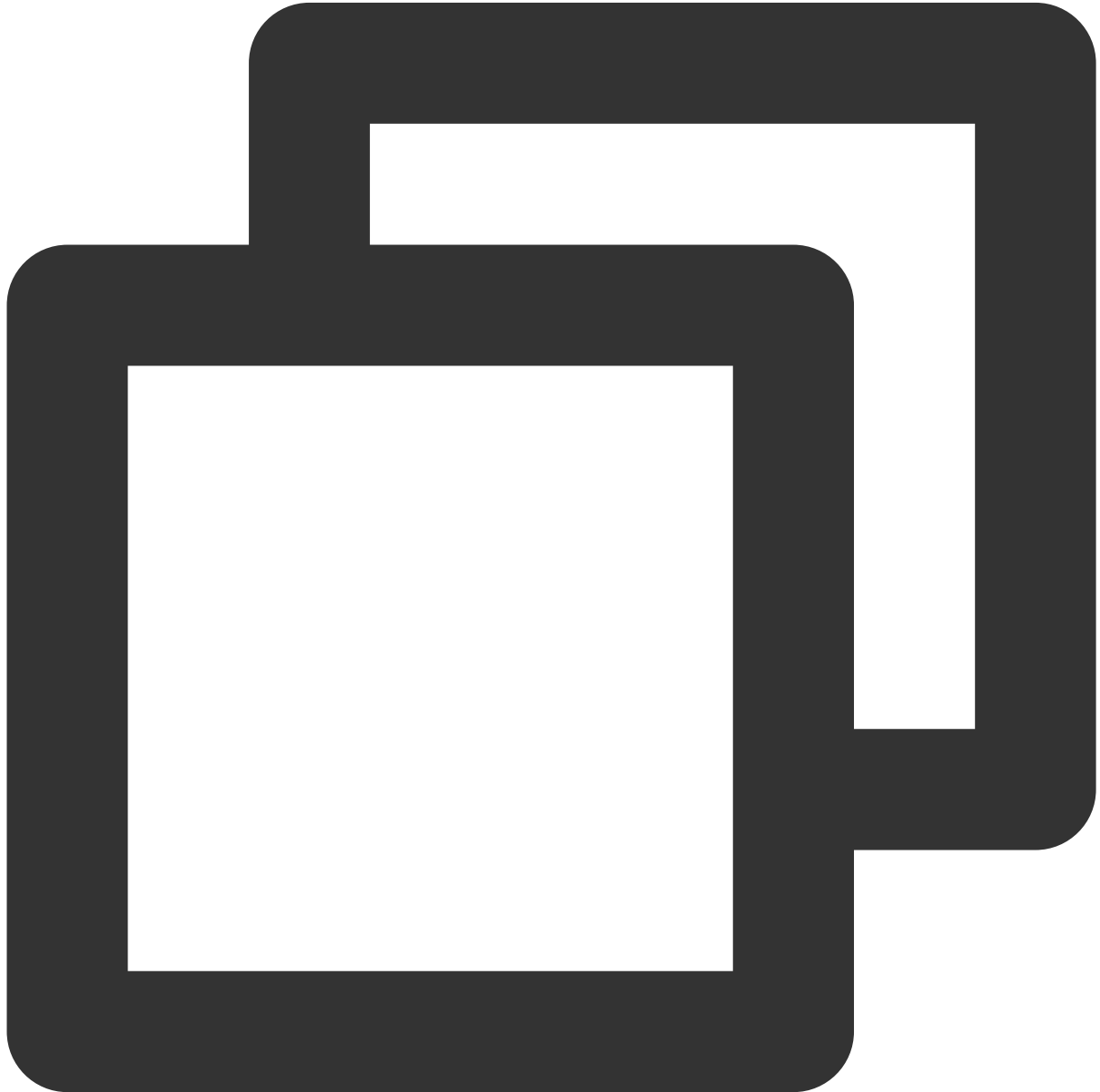


```
int roomId = "Room ID";
let param = RoomParam.init()
param.roomName = "Room name";
param.needRequest = false; // Whether permission is required for listeners to speak
param.seatCount = 8;        // Number of seats in the room. Set it to `8`.
param.coverUrl = "URL of room cover image";

karaokeRoom.createRoom(roomID: Int32(roomInfo.roomID), roomParam: param) { [weak self]
    guard let `self` = self else { return }
    if code == 0 {
        // Room created successfully
    }
}
```

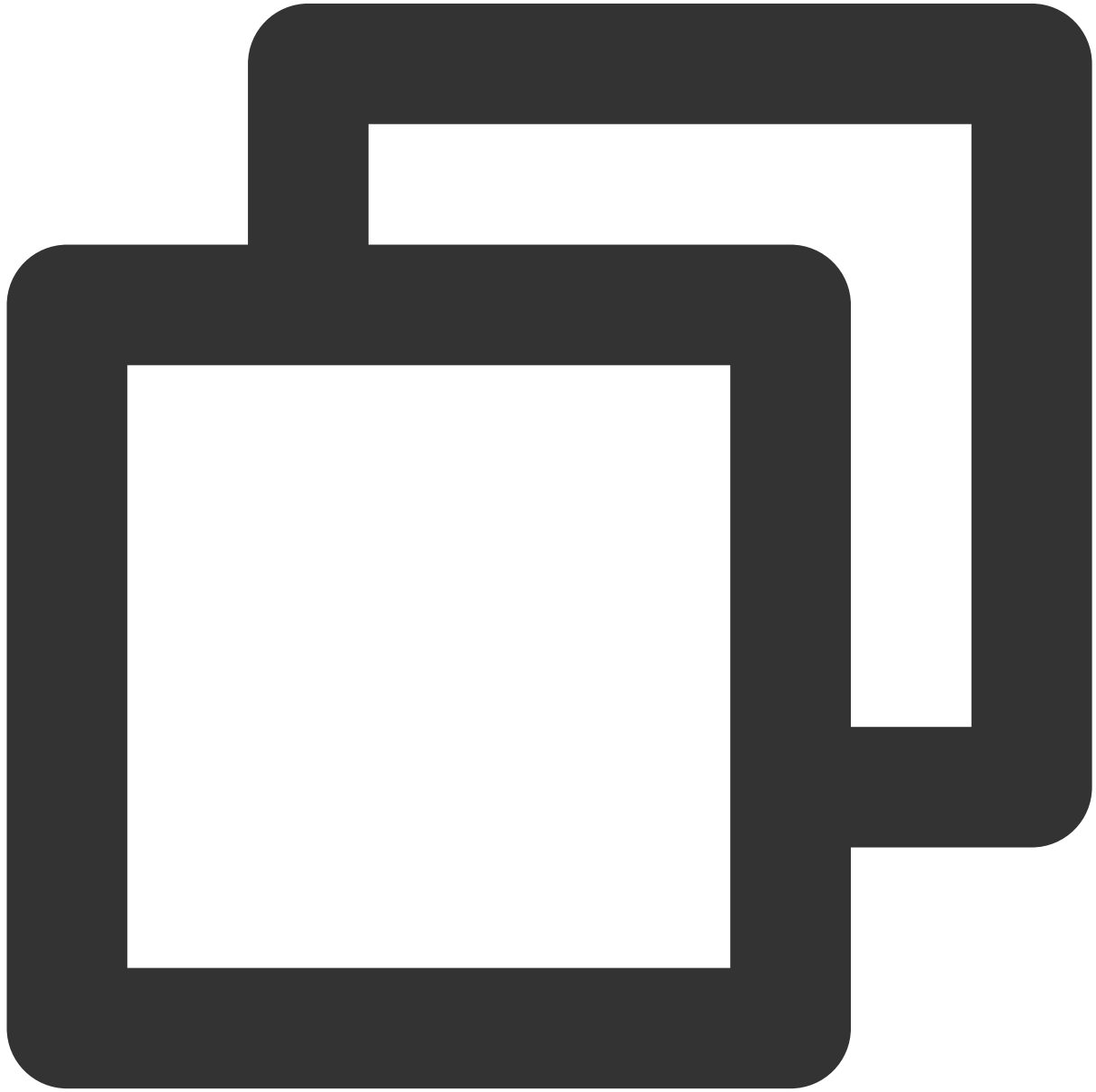
```
}  
}
```

2. A listener enters the room through [TUIKaraoke.enterRoom](#).



```
karaokeRoom.enterRoom(roomID: roomInfo.roomID) { [weak self] (code, message) in  
    guard let `self` = self else { return }  
    if code == 0 {  
        // Entered room successfully  
    }  
}
```

### 3. A listener turns their mic on through [TUIKaraoke.enterSeat](#).



```
// 1. A listener calls an API to mic on
int seatIndex = 1;
karaokeRoom.enterSeat(seatIndex: seatIndex) { [weak self] (code, message) in
    guard let `self` = self else { return }
    if code == 0 {
        // Mic turned on successfully
    }
}
// 2. The listener receives the `onSeatListChange` callback and refreshes the seat
```

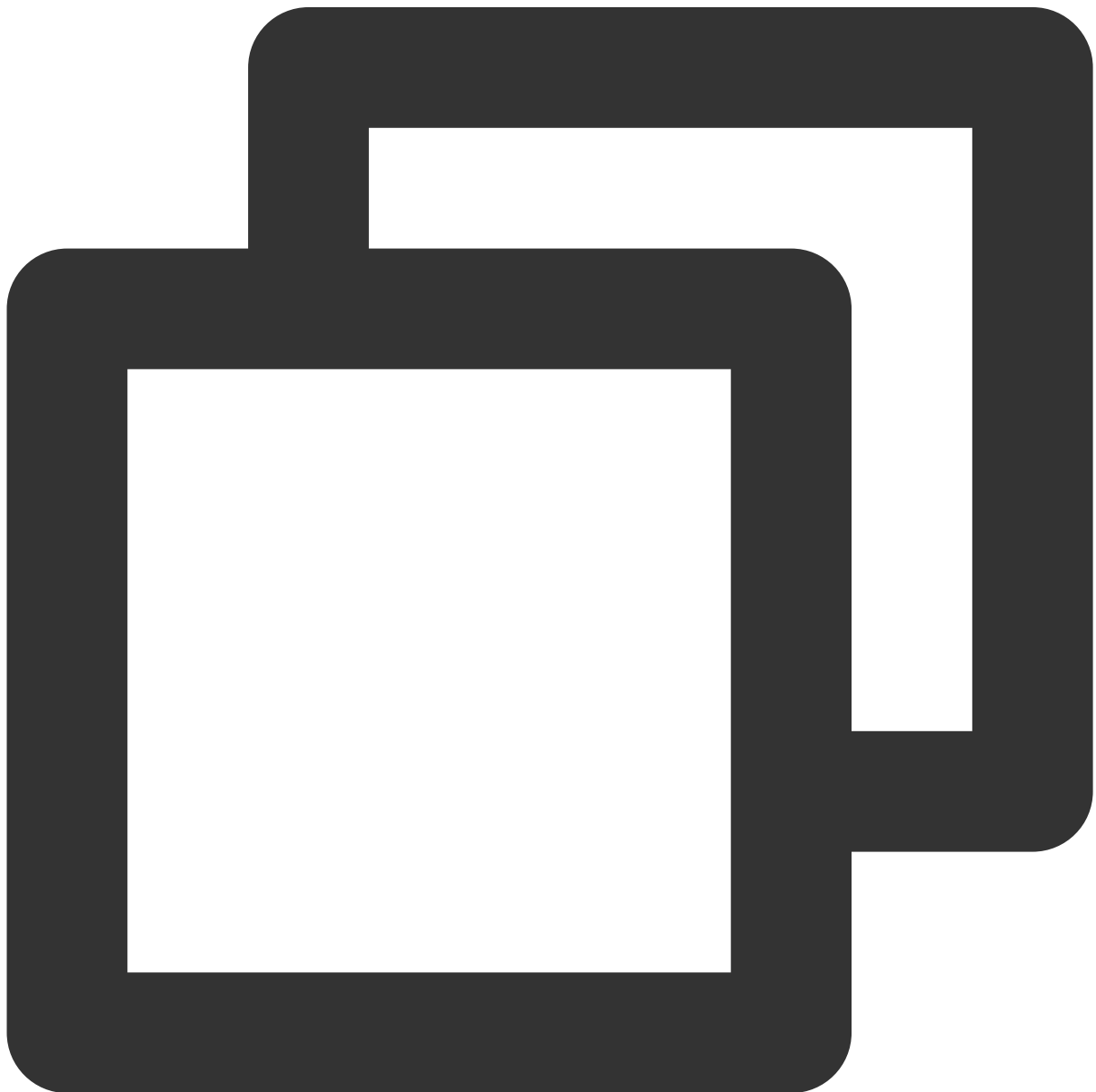
```
func onSeatListChange(seatInfoList: [SeatInfo]) {  
}
```

**Note**

You can implement other seat management operations as instructed in [TRTCKaraoke \(iOS\)](#) or by referring to the [TUIKaraoke demo project](#).

**4. Play back songs and try out the karaoke scenario**

You can get the music ID and URL to play back a song. For more information, see [Music Playback APIs](#).



```
// Play back the music  
karaokeRoom.startPlayMusic(musicID: musicID, originalUrl: musicLocalPath, accompan
```



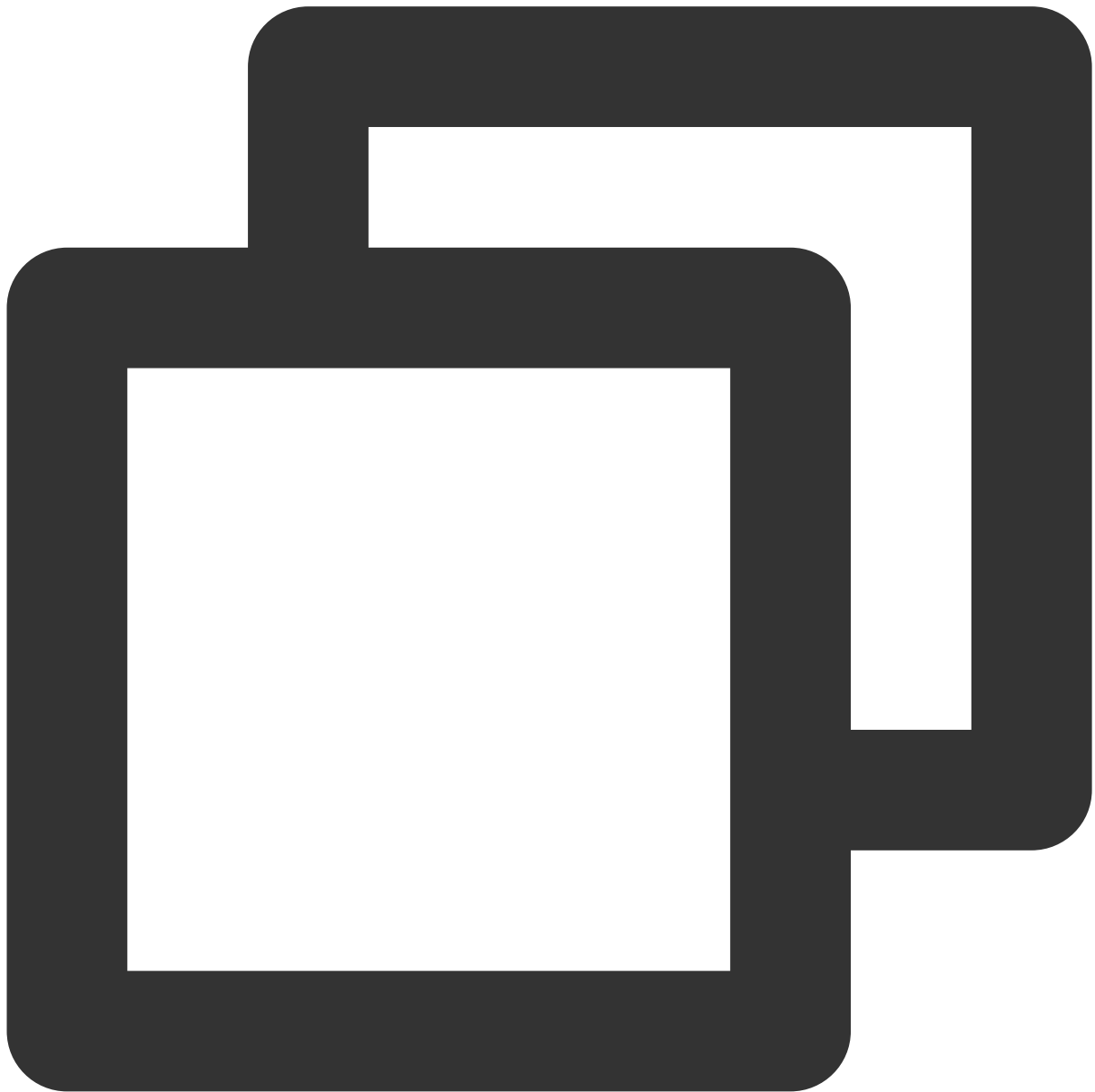
```
// Stop the music  
karaokeRoom.stopPlayMusic();
```

After completing the previous steps, you can implement the basic karaoke features. If your business needs more features such as chat and gift giving, you can integrate the following capabilities:

### Step 5. Add the text chat feature (optional)

If you want implement a text chat feature between speakers and listeners, implement message sending/receiving as follows:

For more information on relevant APIs, see [sendRoomTextMsg](#).

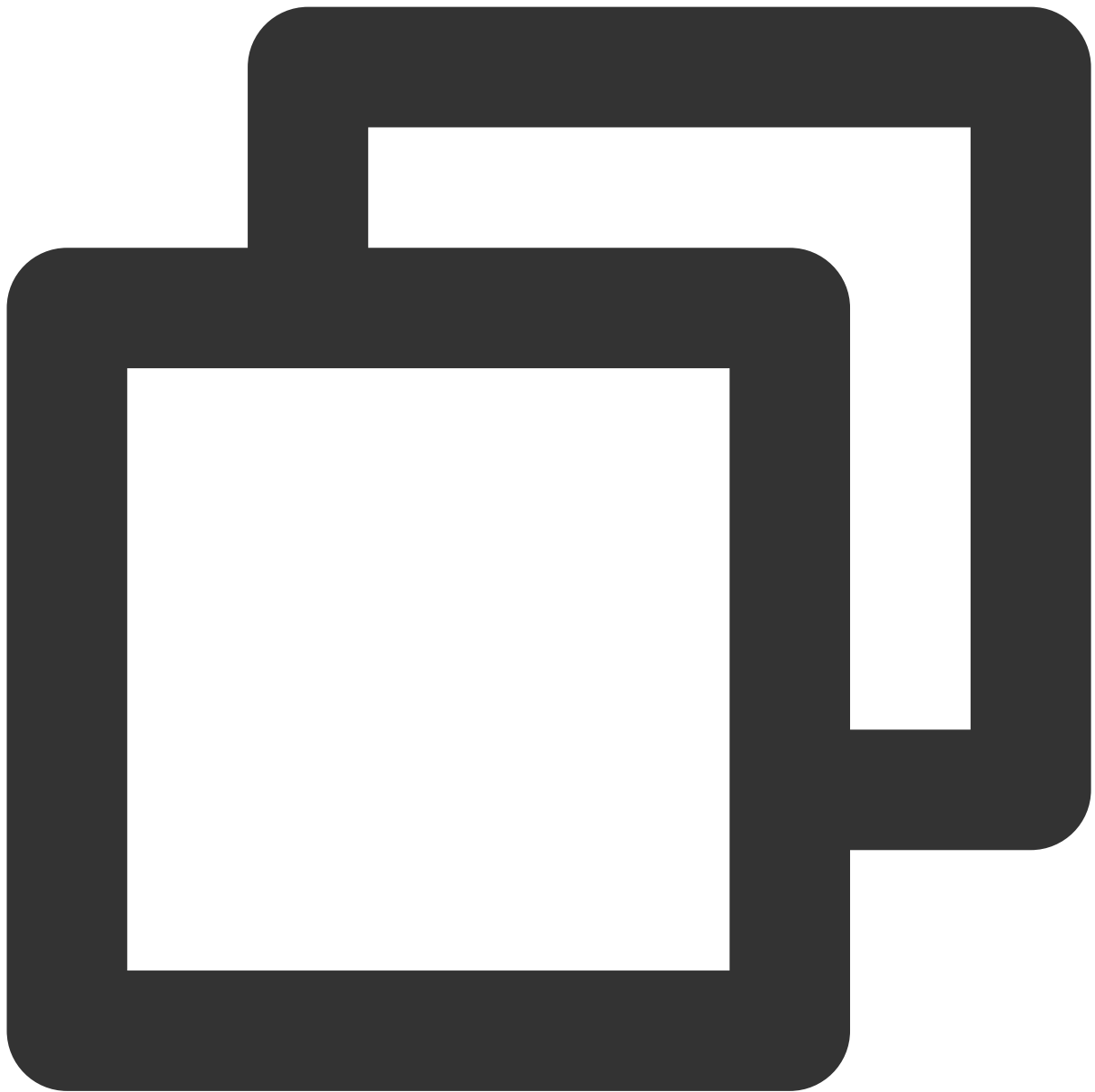


```
// Sender: Sends text chat messages
karaokeRoom.sendRoomTextMsg(message: message) { [weak self] (code, message) in
    if code == 0 {
        // Sent successfully
    }
}

// Receiver: Listens for text chat messages
karaokeRoom.setDelegate(delegate: self)
func onRecvRoomTextMsg(message: String, userInfo: UserInfo) {
    debugPrint("Received a message from" + userInfo.userName + ": " + message)
}
```

## Step 6. Add the gift giving feature (optional)

You can implement gift giving, receiving, and displaying as follows:



```
// Sender: Customize `IMCMD_GIFT` to distinguish between gift messages
karaokeRoom.sendRoomCustomMsg(cmd: kSendGiftCmd, message: message) { code, msg in
    if (code == 0) {
        // Sent successfully
    }
}

// Receiver: Listens for gift messages
karaokeRoom.setDelegate(delegate: self)
func onRecvRoomCustomMsg(cmd: String, message: String, userInfo: UserInfo) {
    if cmd == kSendGiftCmd {
```

```
        debugPrint("Received a gift from" + userInfo.userName + ": " + message)
    }
}
```

## FAQs

**Does the `TUIKaraoke` component support sound effect features such as voice change, tone change, and reverb?**

Yes. For more information, see the [TUIKaraoke demo project](#).

### Note

If you have any suggestions or feedback, please contact [colleenyu@tencent.com](mailto:colleenyu@tencent.com).

# Android

Last updated : 2023-09-25 10:58:37

## Component Overview

TUIKaraoke is an open-source audio/video UI component that you can integrate into your project to bring online karaoke, seat management, gift giving/receiving, text chat, and other TRTC features to your application. TUIKaraoke requires only a few lines of code and also supports the iOS platform. Its basic features are shown below:

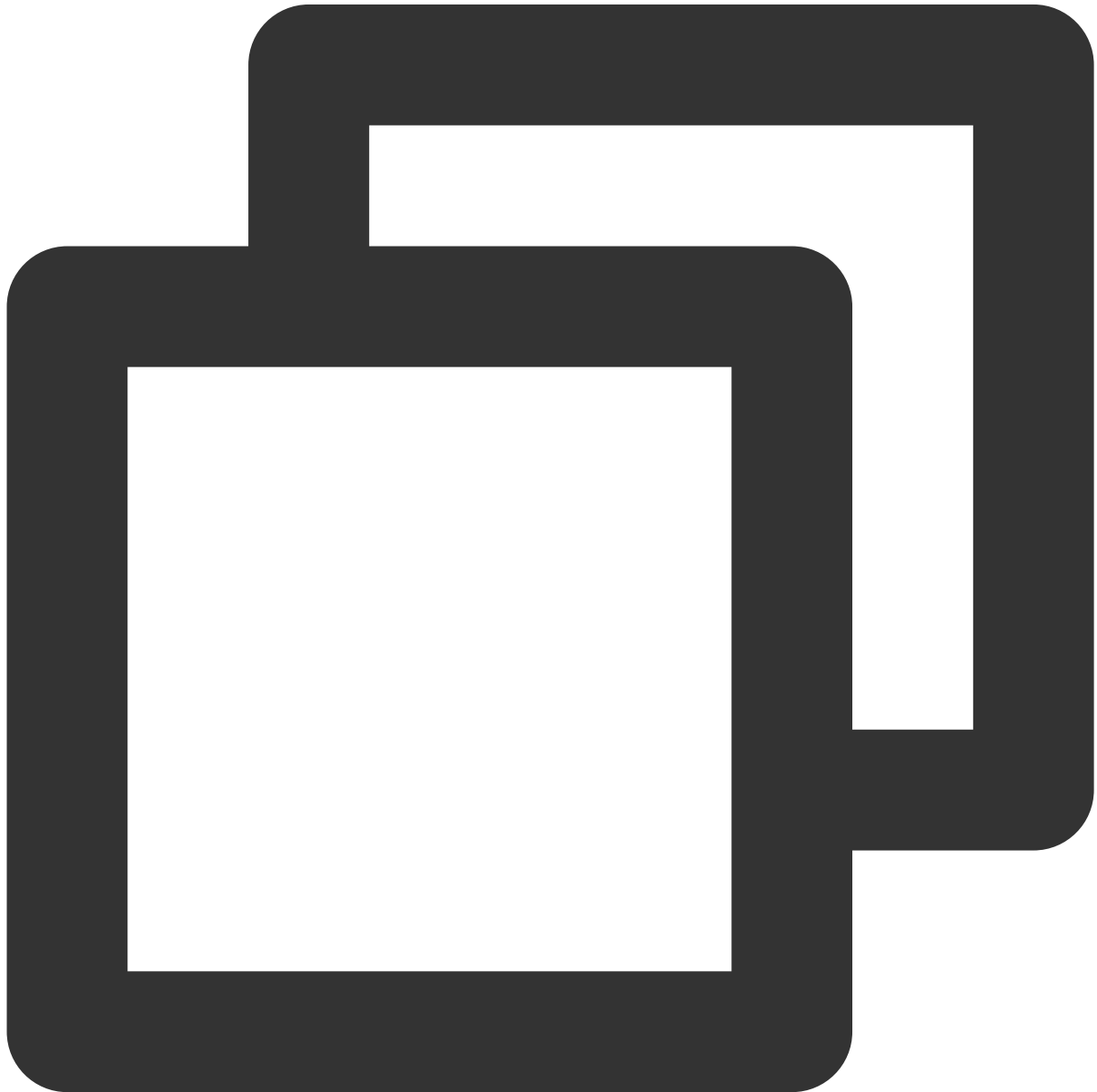


## Component Integration

### Step 1. Download and import the TUIKaraoke component

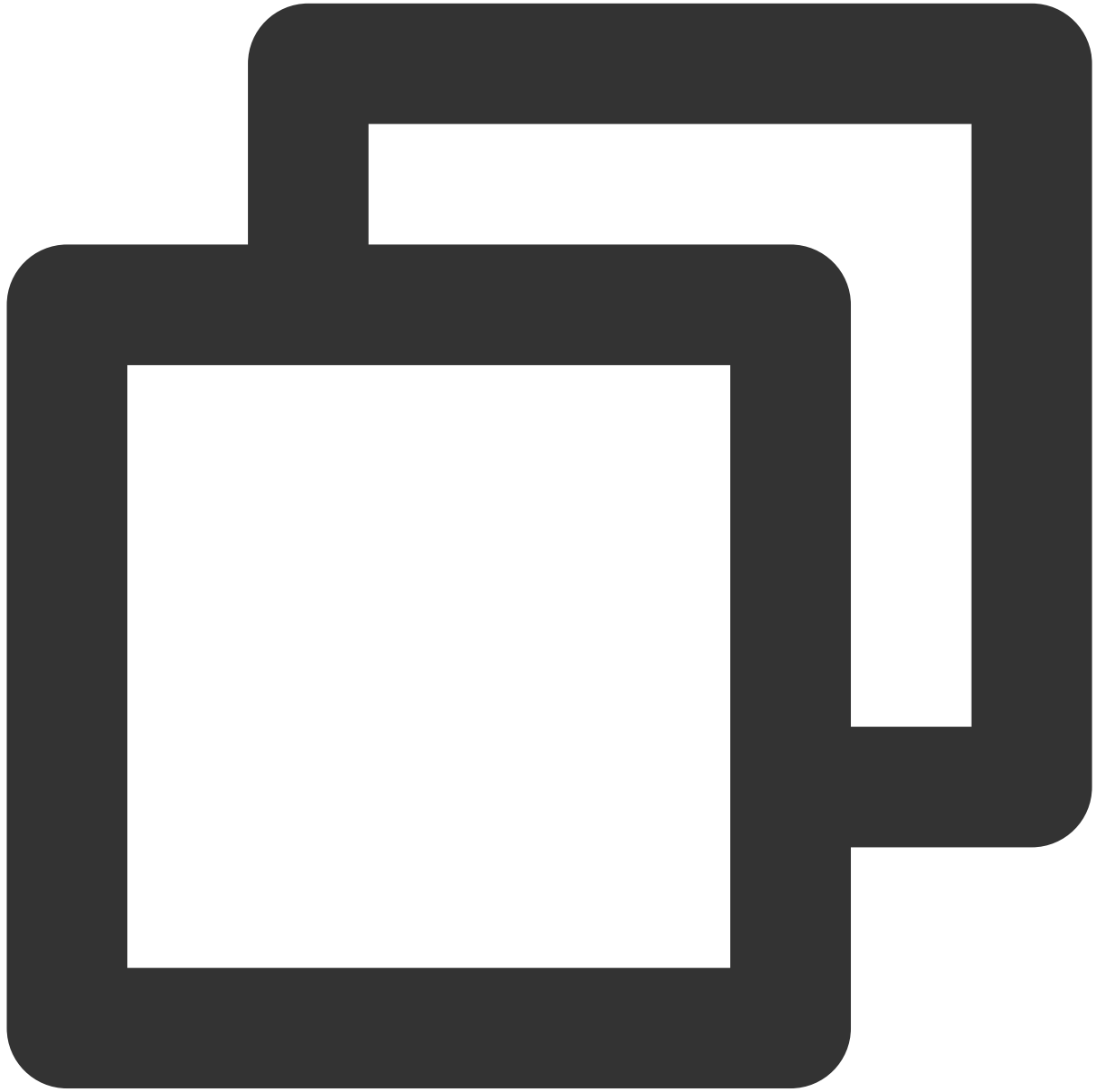
Go to [GitHub](#), clone or download the code, copy the `Source` and `Debug` directories in the `Android` directory to your project, and complete the following import operations:

Complete import in `setting.gradle` as shown below:



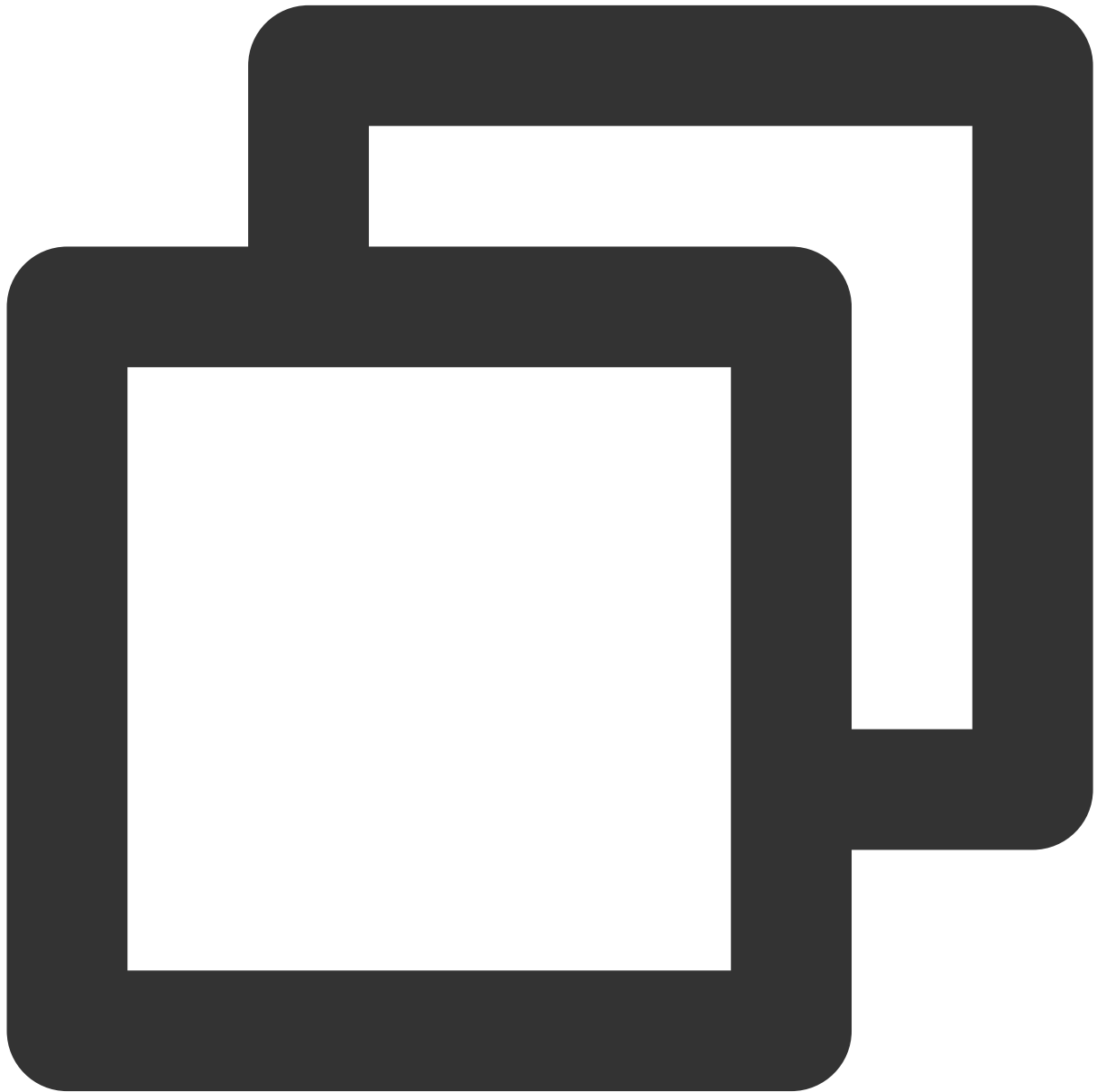
```
include ':Source'  
include ':Debug'
```

Add dependencies on `TUIKaraoke` to the `build.gradle` file in `app` :



```
api project(':Source')
```

Add dependencies on `TRTC SDK` and `IM SDK` to the `build.gradle` file in the root directory:

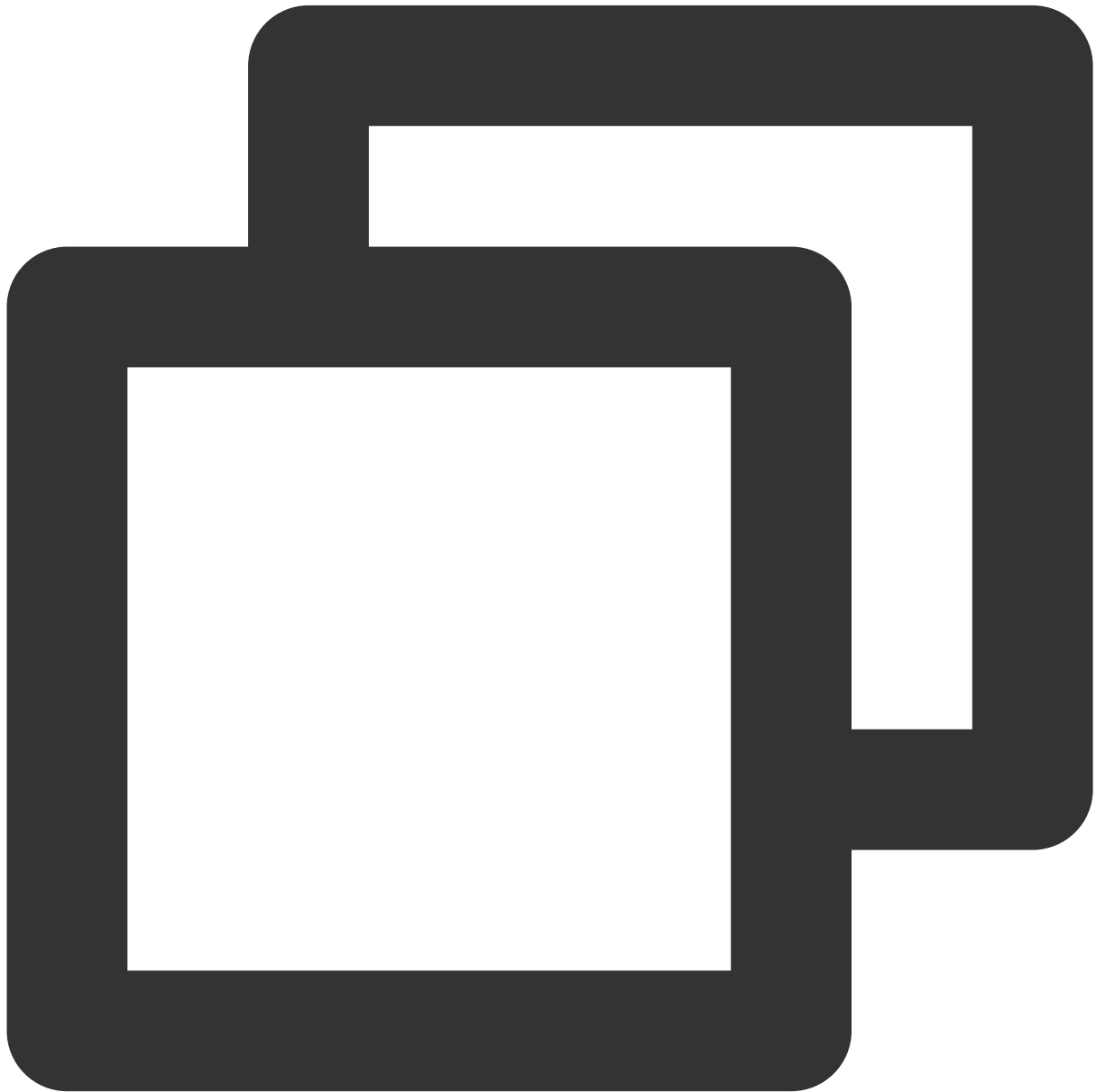


```
ext {  
    liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTC1:latest.release"  
    imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"  
}
```

## Step 2. Configure permission requests and obfuscation rules

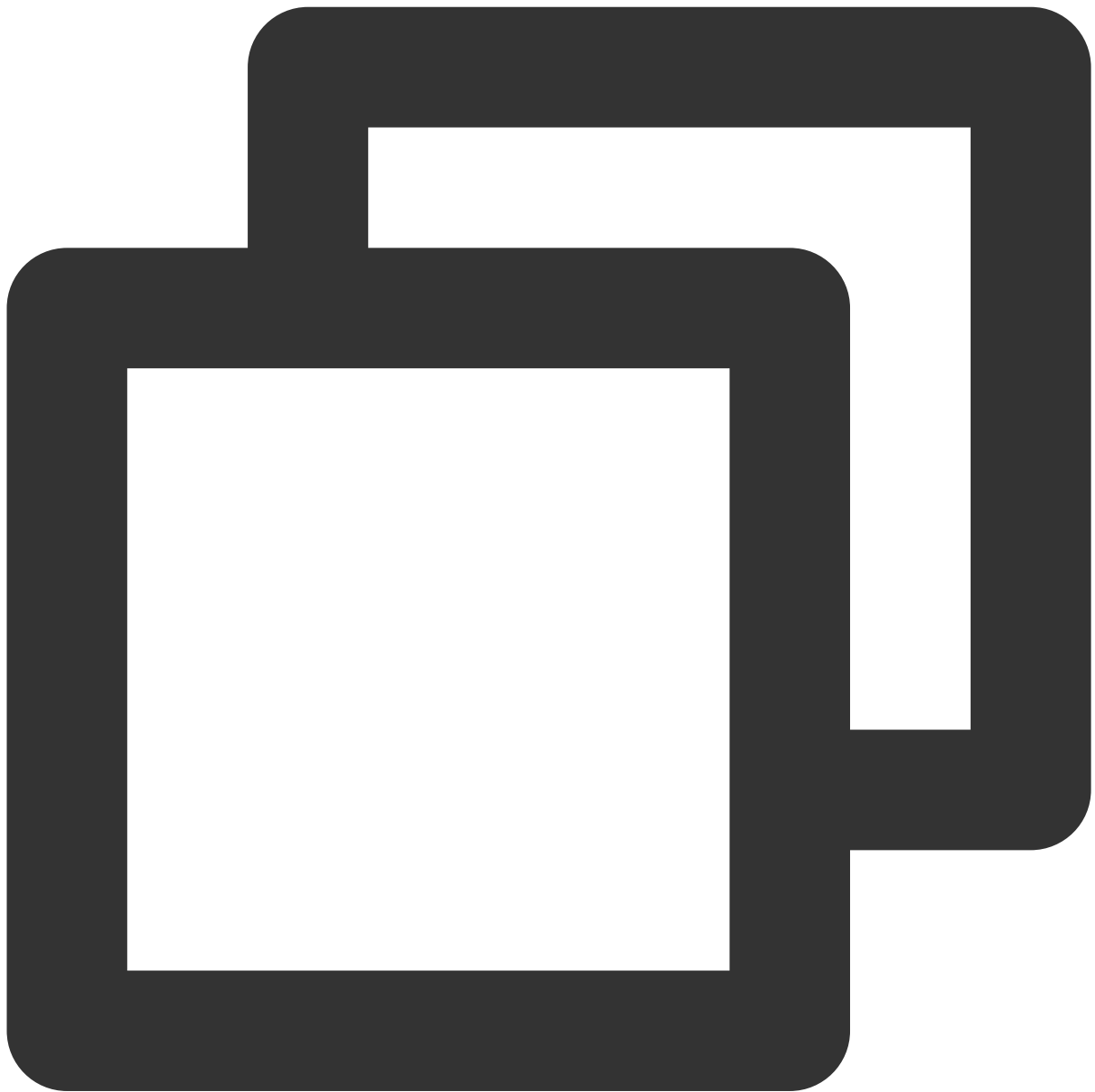
Configure permission requests for your app in `AndroidManifest.xml`. The SDKs need the following permissions (on Android 6.0 and later, the mic access and storage read permission must be requested at runtime):





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

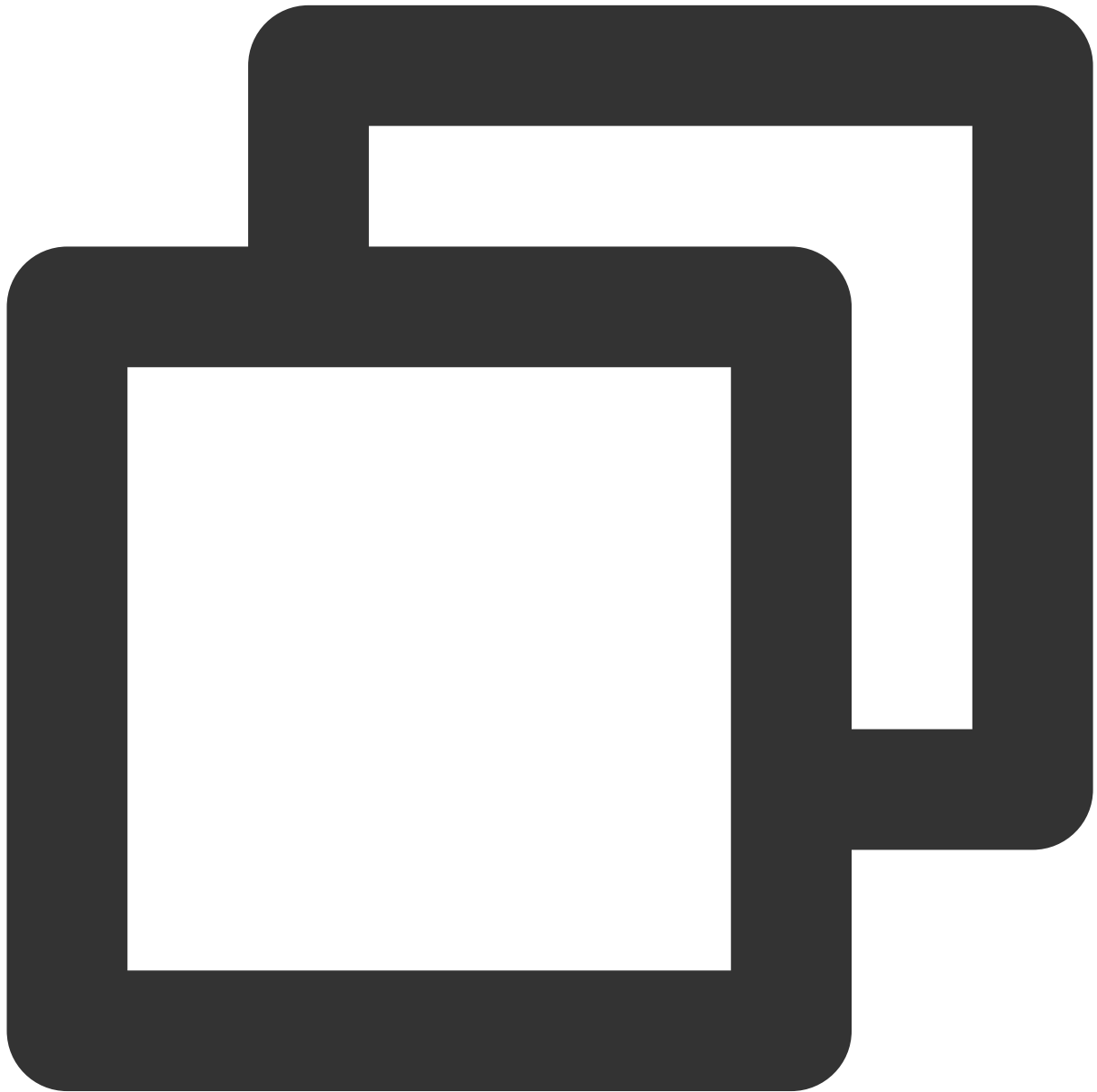
In the `proguard-rules.pro` file, add the SDK classes to the "do not obfuscate" list.



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

### Step 3. Initialize and log in to the component

For more information on relevant APIs, see [TUIKaraoke](#).

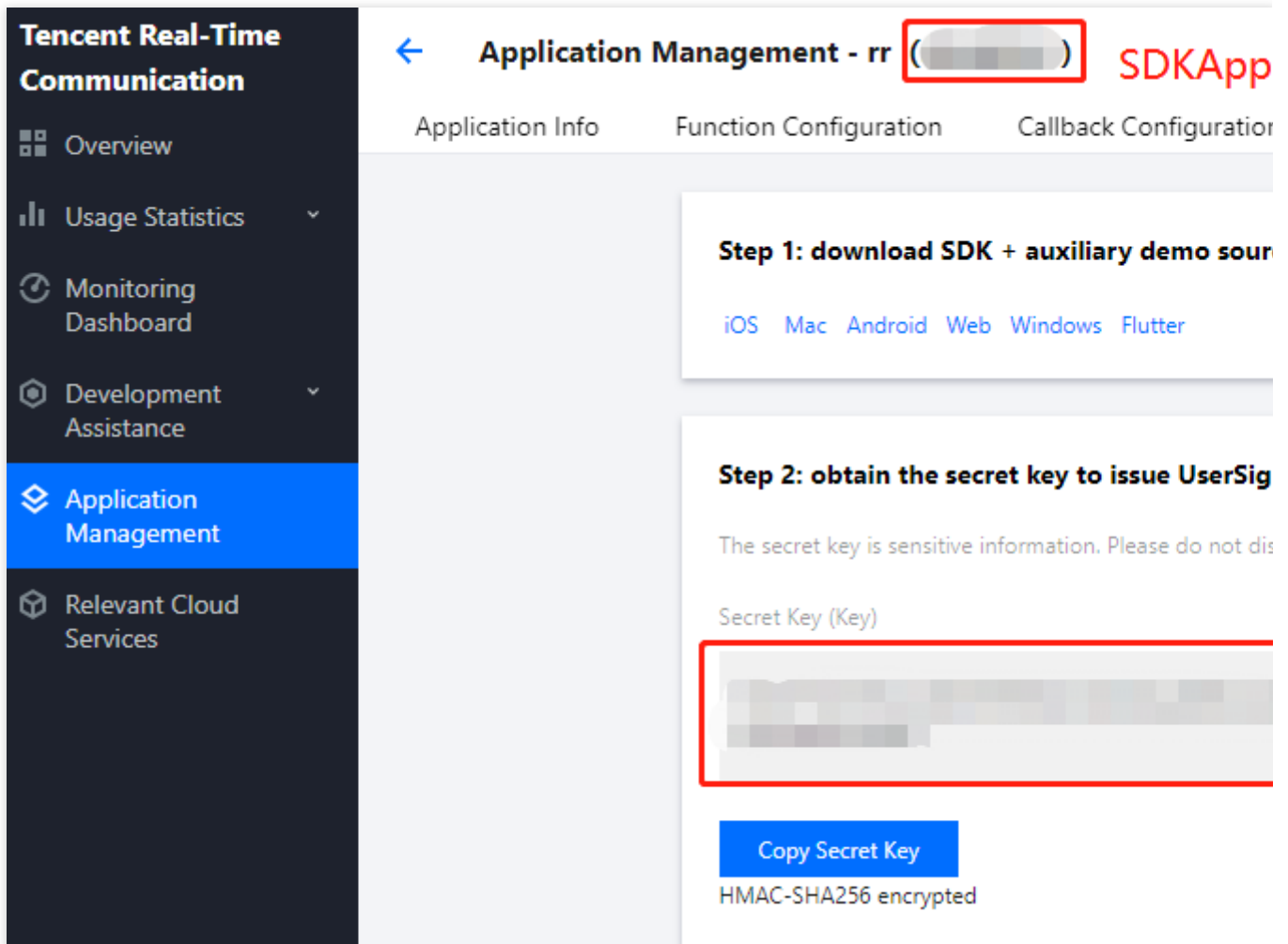


```
// 1. Initialize
TRTCKaraokeRoom mTRTCKaraokeRoom = TRTCKaraokeRoom.sharedInstance(this);
mTRTCKaraokeRoom.setDelegate(this);
// 2. Log in
mTRTCKaraokeRoom.login(SDKAppID, UserID, UserSig, new TRTCKaraokeRoomCallback.Act
    @Override
    public void onCallback(int code, String msg) {
        if (code == 0) {
            // Logged in
        }
    }
}
```

```
});
```

**Parameter description:**

**SDKAppID:** TRTC application ID. If you haven't activated the TRTC service, log in to the [TRTC console](#), create a TRTC application, and click **Application Info**. The `SDKAppID` is as shown below:



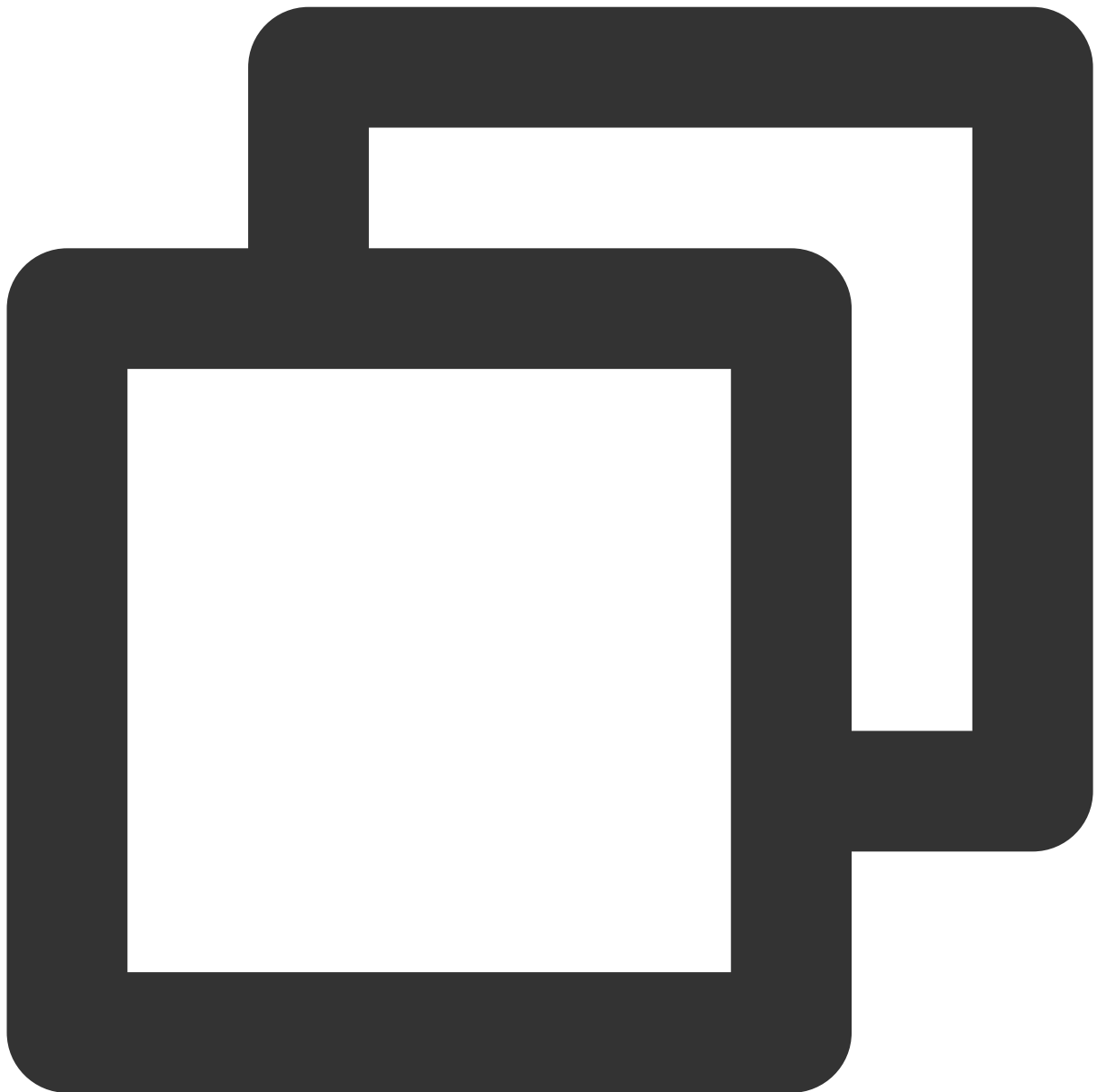
**Secretkey:** TRTC application key, which corresponds to `SDKAppID`. On the [Application Management](#) page in the TRTC console, the `SecretKey` is as shown below:

**userId:** Current user ID, which is a string and can contain up to 32 bytes of letters and digits (special symbols are not supported). You can customize it based on your actual account system.

**userSig:** Security protection signature calculated based on `SDKAppID`, `userId`, and `Secretkey`. You can click [here](#) to directly generate a debugging `userSig` online. For more information, see [UserSig](#).

**Step 4. Implement the online karaoke scenario**

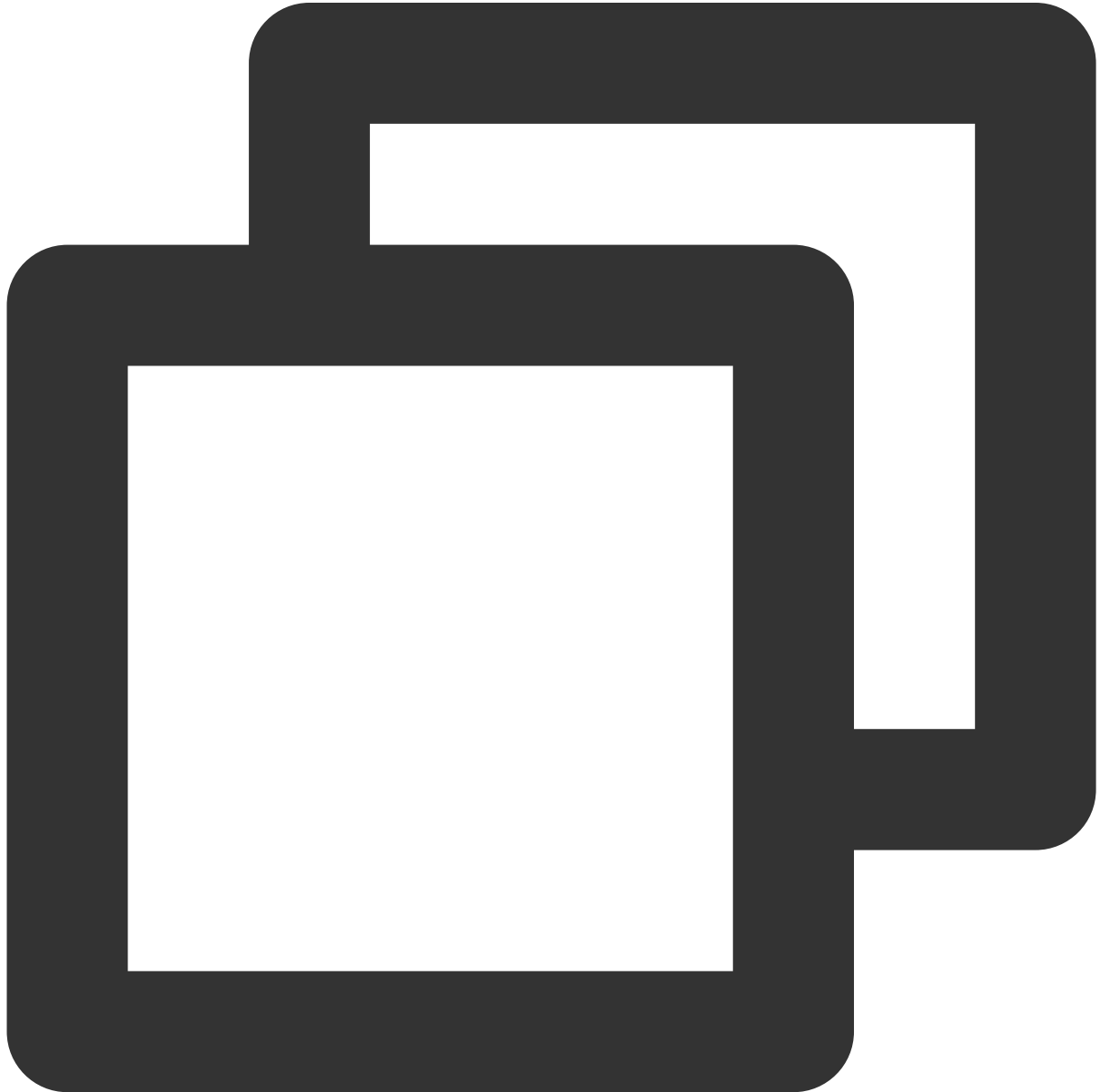
1. The anchor creates a room through [TUIKaraoke.createRoom](#).



```
int roomId = "Room ID";
TRTCKaraokeRoomDef.RoomParam roomParam = new TRTCKaraokeRoomDef.RoomParam();
roomParam.roomName = "Room name";
roomParam.needRequest = false; // Whether your consent is required for listeners to
roomParam.seatCount = 8;       // Number of seats in the room. Set it to `8`
roomParam.coverUrl = "URL of room cover image";
mTRTCKaraokeRoom.createRoom(roomId, roomParam, new TRTCKaraokeRoomCallback.ActionCa
@Override
public void onCallback(int code, String msg) {
    if (code == 0) {
        // Room created successfully
    }
}
```

```
}  
}  
});
```

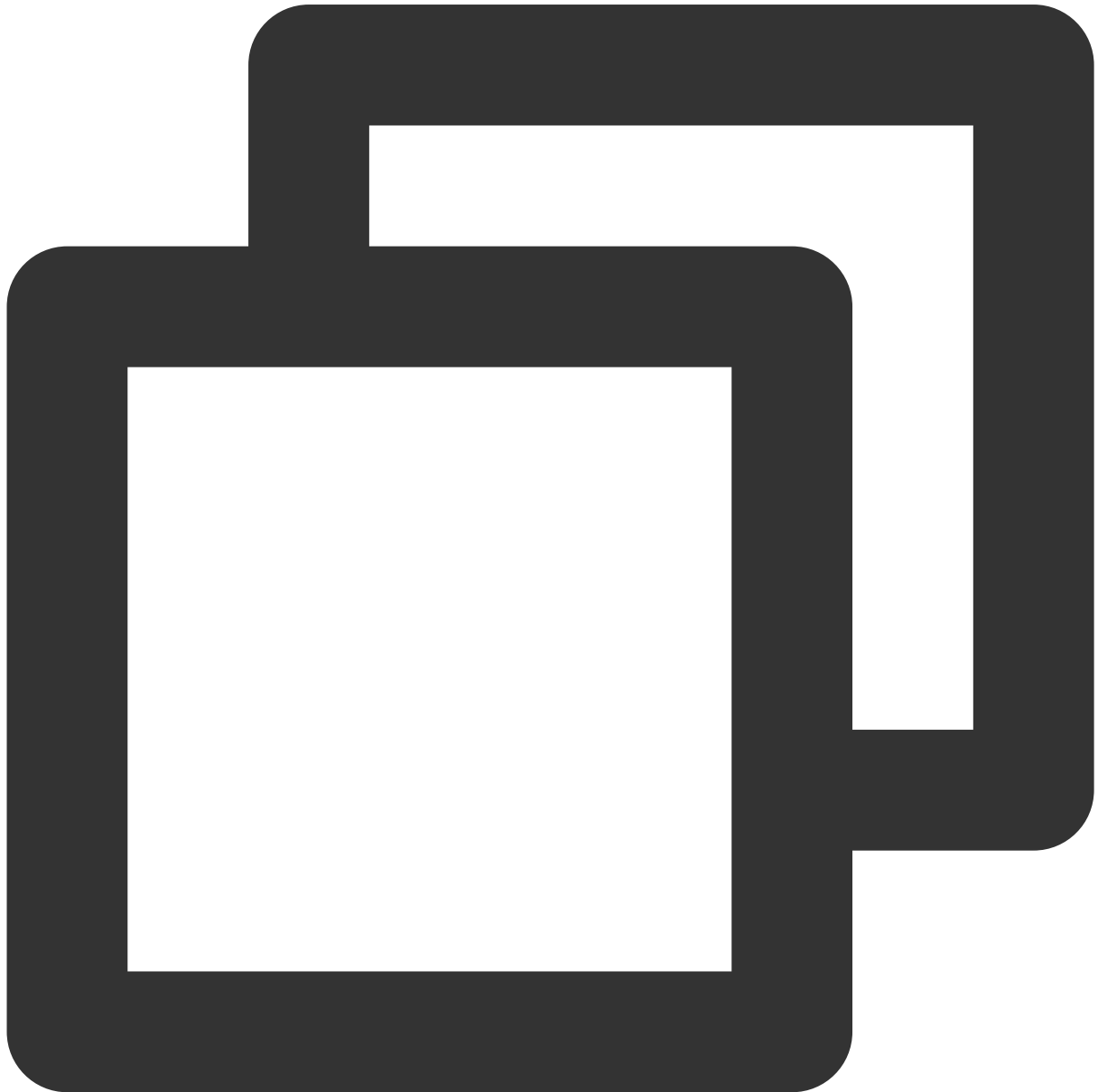
2. A listener enters the room through [TUIKaraoke.enterRoom](#).



```
mTRTCKaraokeRoom.enterRoom(roomId, new TRTCKaraokeRoomCallback.ActionCallback() {  
    @Override  
    public void onCallback(int code, String msg) {  
        if (code == 0) {  
            // Entered room successfully  
        }  
    }  
})
```

```
}  
});
```

3. A listener mics on through [TUIKaraoke.enterSeat](#).



```
// 1. A listener calls an API to mic on  
int seatIndex = 1;  
mTRTCKaraokeRoom.enterSeat(seatIndex, new TRTCKaraokeRoomCallback.ActionCallback()  
    @Override  
    public void onCallback(int code, String msg) {  
        if (code == 0) {  
            // Mic turned on successfully
```

```
    }  
    }  
});  
// 2. The listener receives the `onSeatListChange` callback and refreshes the seat  
@Override  
public void onSeatListChange(final List<TRTCKaraokeRoomDef.SeatInfo> seatInfoList)  
{  
}
```

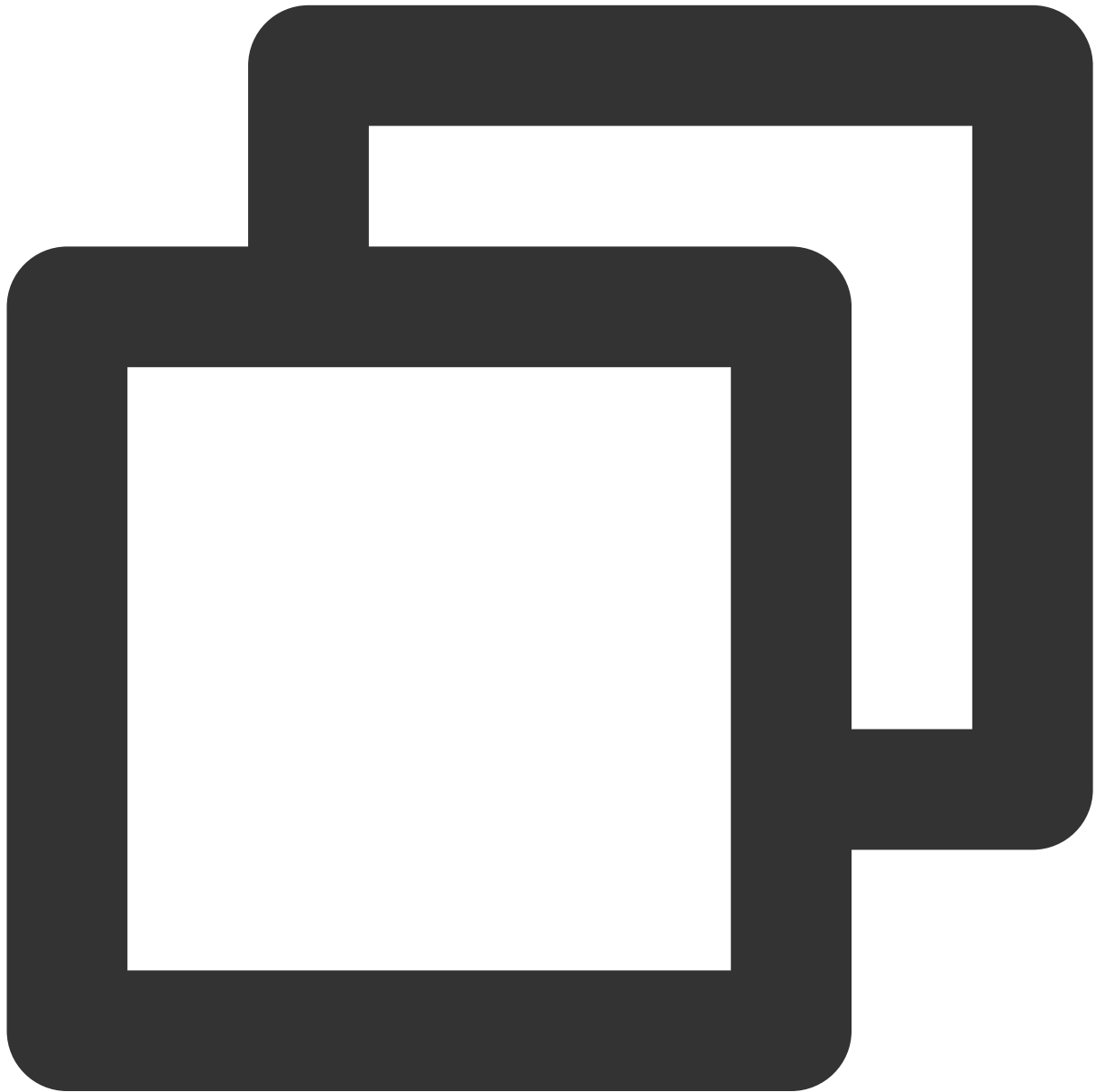
**Note**

You can implement other seat management operations as instructed in [TRTCKaraoke \(Android\)](#) or by referring to the [TUIKaraoke demo project](#).

**4. Play back songs and try out the karaoke scenario**

You can get the music ID and URL to play back a song based on your business. For more information, see [Music Playback APIs](#).



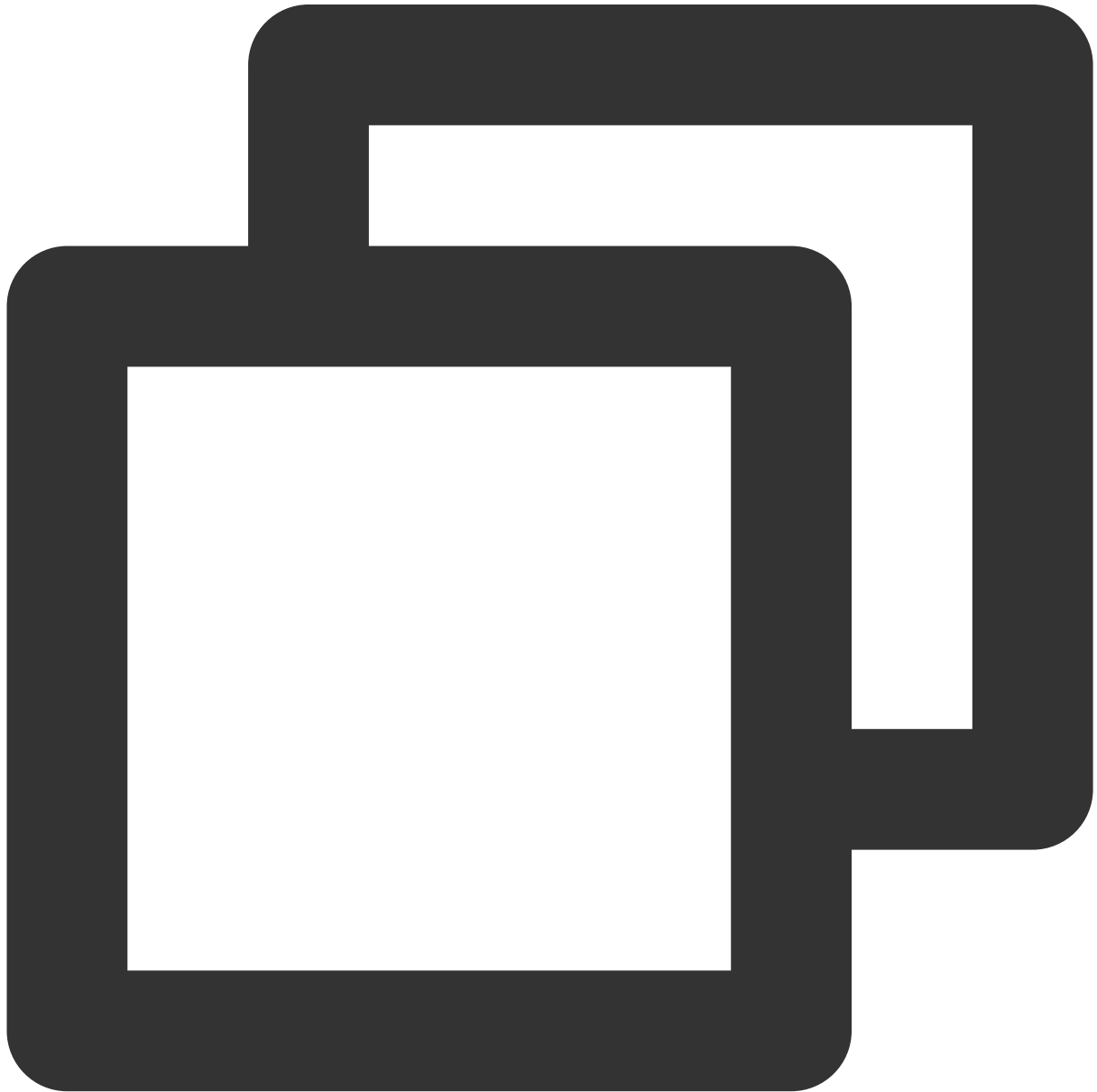


```
// Play back the music
mTRTCKaraokeRoom.startPlayMusic(musicID,url);
// Stop the music
mTRTCKaraokeRoom.stopPlayMusic();
```

After completing the previous steps, you can implement the basic karaoke features. If your business needs more features such as text chat and gift giving, you can integrate the following capabilities:

### Step 5. Add the text chat feature (optional)

If you want the text chat feature between anchors and listeners, implement message sending/receiving as follows:  
For more information on relevant APIs, see [sendRoomTextMsg](#).

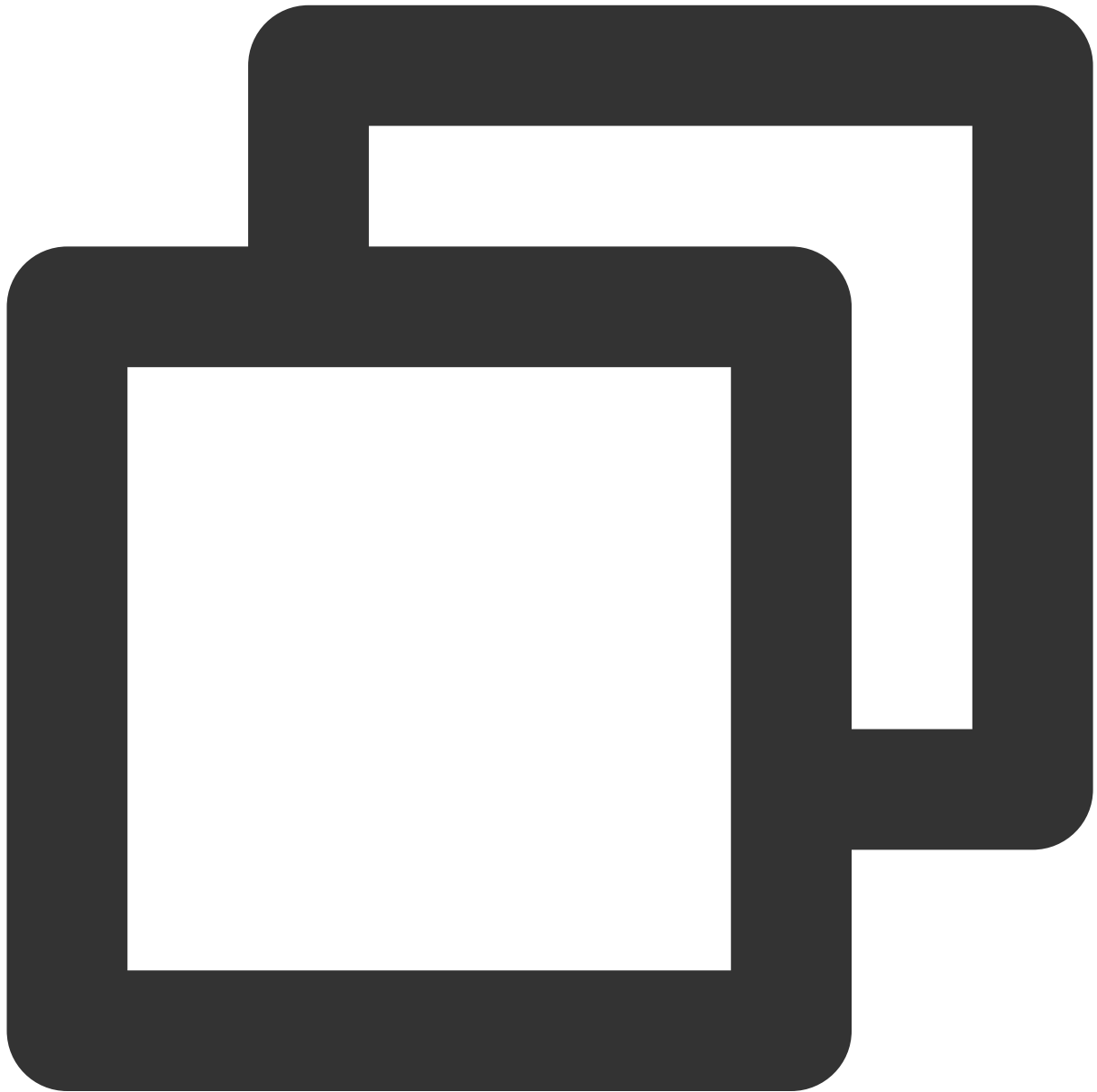


```
// Sender: Sends text messages
mTRTCKaraokeRoom.sendRoomTextMsg("Hello Word!", new TRTCKaraokeRoomCallback.ActionC
@Override
public void onCallback(int code, String msg) {
    if (code == 0) {
        // Sent successfully
    }
}
```

```
});  
// Receiver: Listens for text messages  
mTRTCKaraokeRoom.setDelegate(new TRTCKaraokeRoomDelegate() {  
    @Override  
    public void onRecvRoomTextMsg(String message, TRTCKaraokeRoomDef.UserInfo userInfo)  
        Log.d(TAG, "Received a message from" + userInfo.userName + ": " + message);  
    }  
});
```

## Step 6. Add the gift giving feature (optional)

If you want the gift giving and receiving features, implement gift giving, receiving, and displaying as follows:



```
// Sender: Customize `CMD_GIFT` to distinguish between gift messages
mTRTCKaraokeRoom.sendRoomCustomMsg("CMD_GIFT",date, new TRTCKaraokeRoomCallback.Act
@Override
public void onCallback(int code, String msg) {
    if (code == 0) {
        // Sent successfully
    }
}
});

// Receiver: Listens for gift messages
```

```
mTRTCKaraokeRoom.setDelegate(new TRTCKaraokeRoomDelegate() {  
    @Override  
    public void onRecvRoomCustomMsg(String cmd, String message, TRTCKaraokeRoomDef.  
        if ("CMD_GIFT".equals(cmd)) {  
            // Received a gift message  
            Log.d(TAG, "Received a gift from" + userInfo.userName + ": " + message)  
        }  
    }  
});
```

## FAQs

**Does the `TUIKaraoke` component support sound effect features such as voice change, tone change, and reverb?**

Yes.

**Note**

If you have any suggestions or feedback, please contact [colleenyu@tencent.com](mailto:colleenyu@tencent.com).

# Solution Overview (TUIKaraoke)

## Implementation Steps

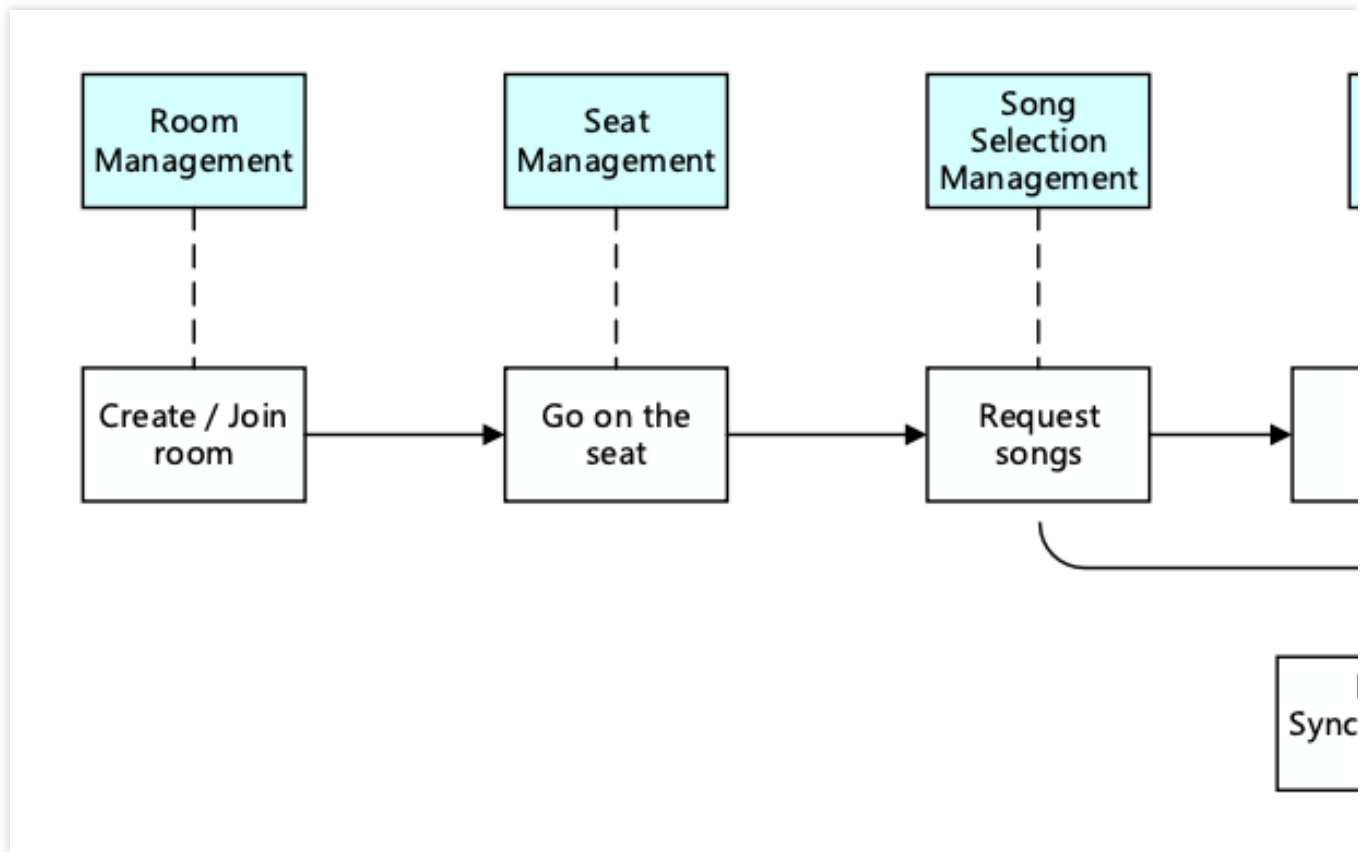
Last updated : 2023-09-26 16:38:31

### Introduction

To implement a complete online Karaoke scenario, multiple functional modules are required, including room management, seat management, song selection management, and Karaoke management. The key actions and features of each functional module are shown in the table below. In the following sections, each functional module will be introduced in detail to provide a complete understanding of the required functions for building a Karaoke room.

| Room Management | Seat Management    | Song Selection Management | Karaoke Management      |
|-----------------|--------------------|---------------------------|-------------------------|
| Room List       | Go on/off the seat | Song List Display         | Karaoke Play Mode       |
| Create Room     | Seat Control       | Search for Songs          | Song Switching          |
| Join Room       | Lock the Seat      | Song Selection            | Vocal Volume Adjustment |
| Leave Room      | Take Seat          | Song Top                  | Reverb/Sound Effects    |
| Destroy Room    | Mute Seat          | Selected Song List        | Lyric Synchronization   |

The room owner creates the Karaoke room, and users can choose to join the room they are interested in. After entering the room, users can go on the seat to participate in the interaction and have voice interaction with the room owner. Of course, users can also choose to go directly on the seat to participate in the chorus. These are two different Karaoke play modes. The overall business process of the online Karaoke scenario is shown in the figure below.

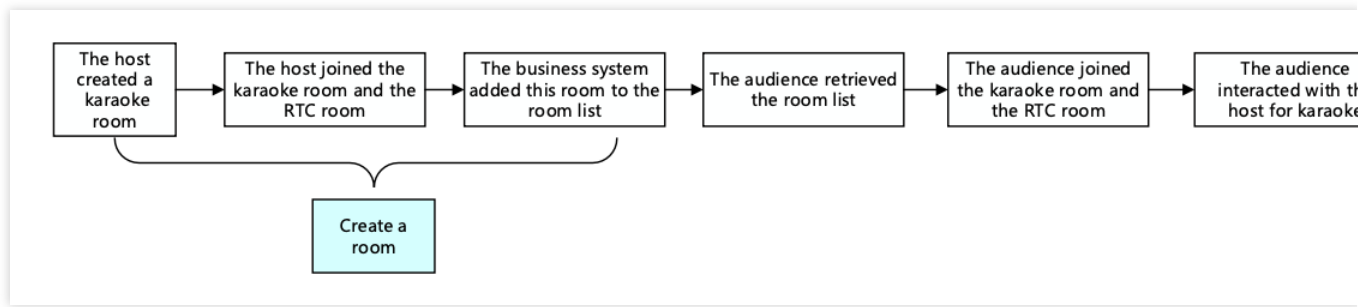


## Room Management

Room management is mainly responsible for maintaining the room list. The main functions include creating a room, joining a room, destroying a room, and leaving a room. Moreover, Karaoke rooms are different from ordinary rooms and require a separate Karaoke room identifier to start related component management, such as song selection management and Karaoke management.

**Create Room:** After logging into the business system, users can create a room. After creating a room, the room list needs to be updated with the new room.

**Destroy Room:** After all users leave the room, the room needs to be destroyed. After destroying the room, the room list needs to be updated with the deletion of the room.

**Note :**

Room management is a necessary module for implementing online karaoke, but it is not the main functional module. The specific implementation can be combined with the business system and TRTC SDK, please refer to the voice chat room scene access solution for details.

## Seat Management

The seats in the karaoke room are generally ordered and limited. Seat management is mainly responsible for defining the number of seats in the room and managing the status of all seats in the current room according to the business scenario. Seat management mainly includes the following functions: going on/off the seat, locking the seat, inviting to go on the seat, and muting the seat.

After entering the room, users can only apply to go on the seat for the seats that are in idle state.

After the host agrees to let the user go on the seat, the seat status needs to be changed to a non-idle state.

After the user stops streaming and goes off the seat, the seat status needs to be reset.

The host has the right to lock the seat, invite to go on the seat, force to go off the seat, and mute the seat.

**Note :**

Seat management is a necessary module for implementing online karaoke, but it is not the main functional module. The specific implementation can be combined with the business system and IM SDK, please refer to the voice chat room scene access solution for details.

## Song selection management

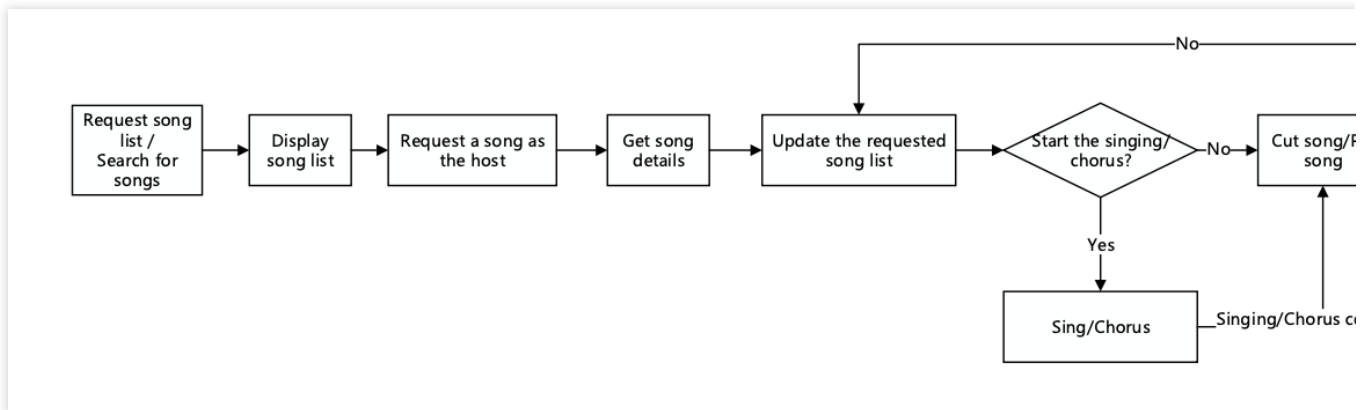
### Basic Introduction

Song selection management is an important part of the online karaoke scene, which mainly includes the following functions: song list display, song search, song selection and queue management, and list of selected songs.

Moreover, each karaoke room needs to maintain a list of selected songs and an automatic queue management function, which requires the business backend to implement. Song list display and song search need to be combined with Yinsuda Authorized Music for Live Streaming to achieve.



## Implementation Process



The entire song selection management mainly involves the business-side app, the business backend, and the Yinsuda backend, each with its own functions:

### Business-side app:

Call the song selection API to report song information.

Call the song cutting API to notify the business backend to update the list of selected songs.

Call the singing confirmation API to notify the business backend.

### Business backend:

Maintain the list of selected songs.

Send notifications to the business-side app to update the current list of selected songs.

### Yinsuda backend:

Provide APIs to obtain the recommended song list and song list details for live interactive music Song List/Song List Details.

Provide an API to obtain the details of live interactive music Get Live Interactive Music Details (playToken, lyric download URL).

Provide an API to search for live interactive music Search Live Interactive Music.

## Karaoke Management

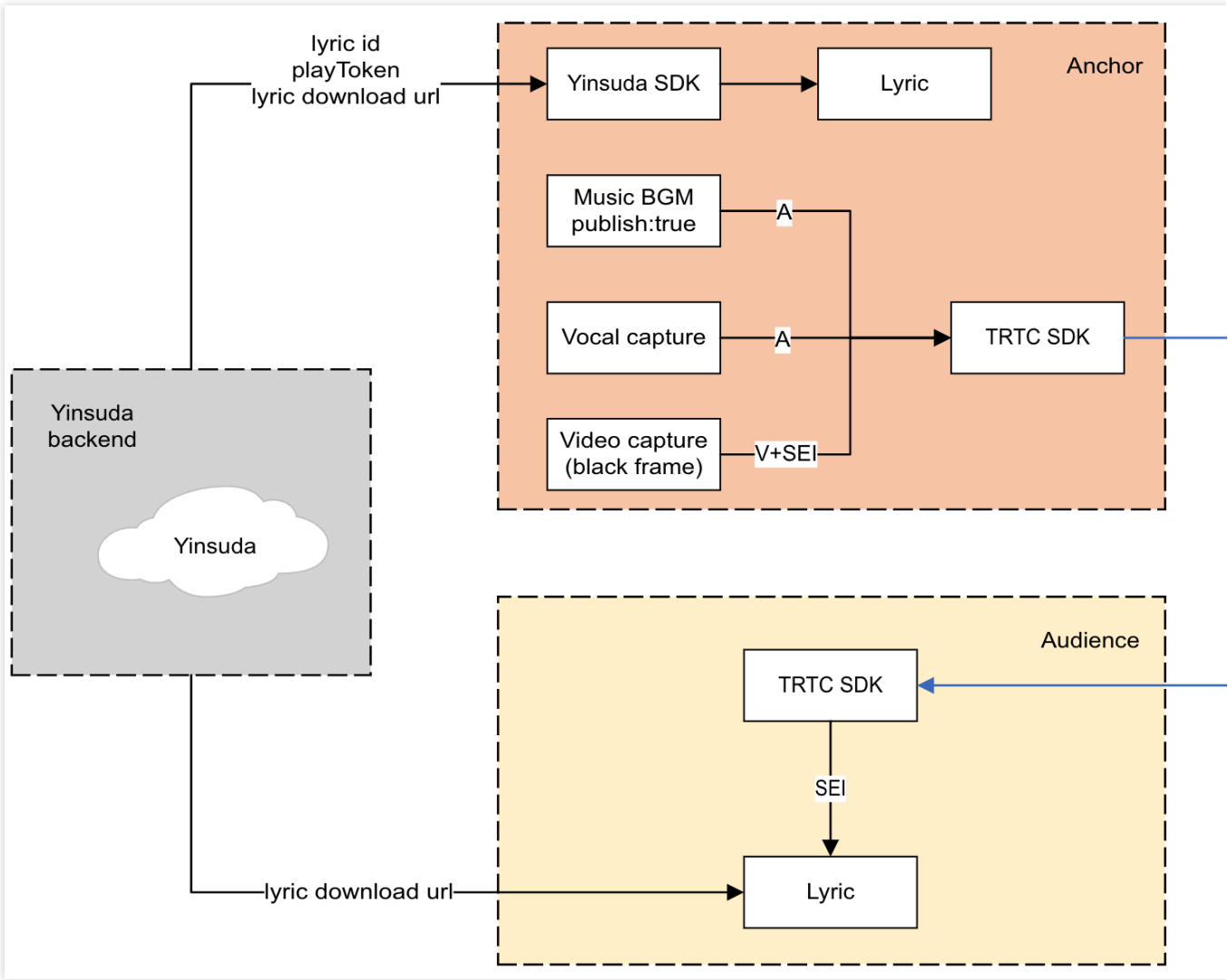
The karaoke system mainly includes the following functions: singing gameplay, start/stop/song cutting, vocal volume adjustment, reverb/sound effects, and lyric synchronization. Below, we will introduce the implementation process of the karaoke management module in detail through two typical karaoke gameplay: solo singing and real-time chorus.

### Solo Singing

This is mainly a multi-user interactive Karaoke scene. After the host goes on the seat, they can select songs for singing. Once the host successfully selects a song, all song selection information will be displayed on the song selection platform. The host can then choose to begin singing.

(1) Solution Architecture

The overall solution architecture mainly utilizes the VOD SDK to achieve song downloading, the VOD backend to obtain the playToken and lyric download address of the song, and the TRTC SDK to implement the singer's voice streaming, song playback, and streaming. The overall solution architecture is as follows:



(2) Specific Implementation

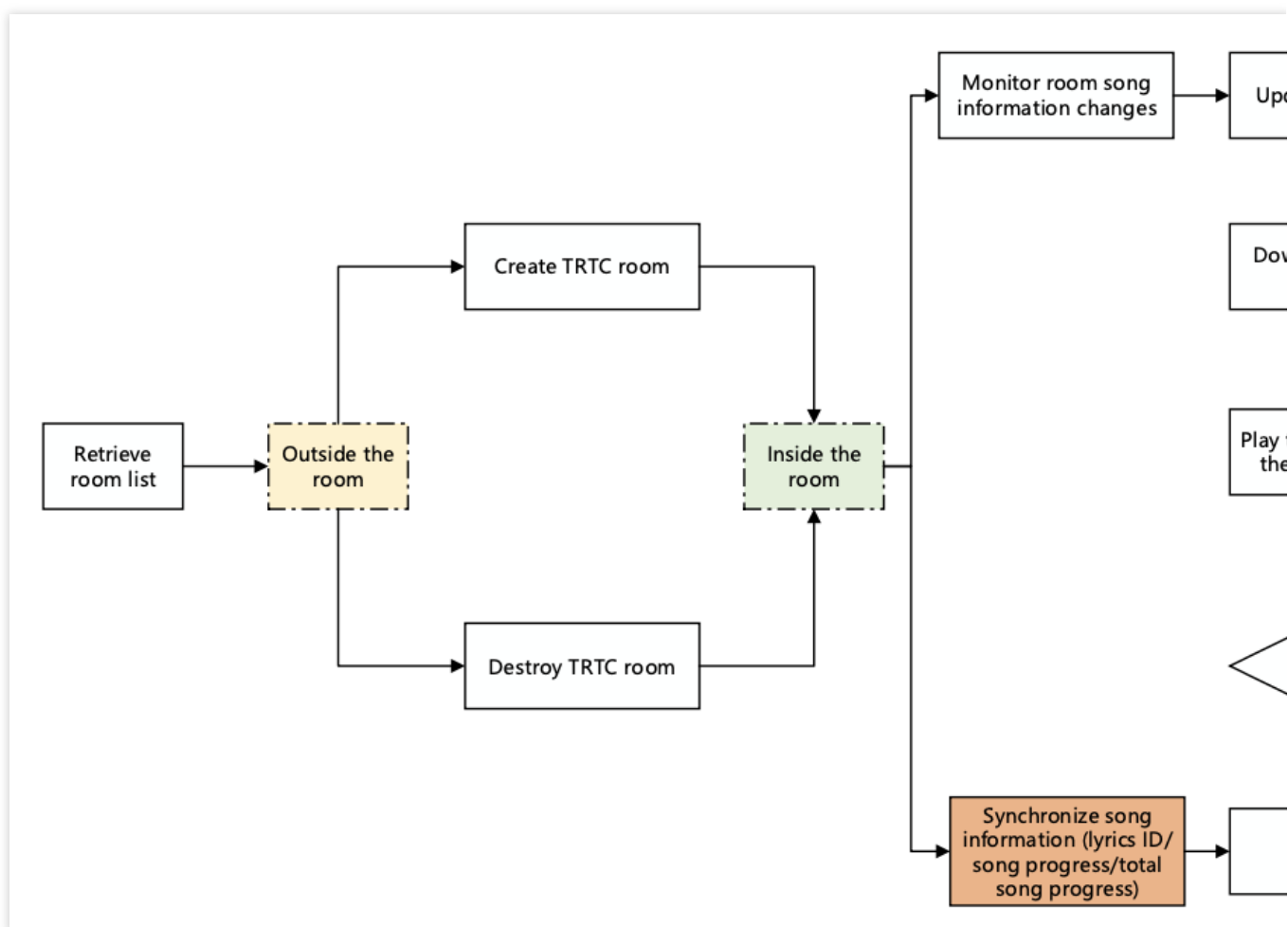
In the singing scenario, different roles have different implementation processes, which can be divided into two roles: singer and audience.

| Role | Description | Differences |
|------|-------------|-------------|
|      |             |             |

|          |                                                                                                                                                                                                                                   |                                                                                                                                                                       |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Singer   | The singer in the Karaoke room is evolved from the host who selects songs and sings after going on the seat. After leaving the room, the room is automatically dissolved and the list of selected songs is automatically cleared. | The role must be a host<br>Upstream audio and video (no video upstream black frame)<br>Play BGM<br>Send SEI information (send lyric information)<br>Song selection    |
| Audience | The audience in the Karaoke room plays the stream of the singer.                                                                                                                                                                  | The role is an audience, but can also become a host by going on the seat<br>Downstream audio and video streams<br>Receive SEI information (receive lyric information) |

The basic implementation processes for different roles are as follows:

【Host】



The host creates and joins a TRTC room, automatically goes on the seat, and becomes a singer after selecting a song.

After selecting a song, the song/lyric is downloaded, and then the song is played through the BGM playback interface.

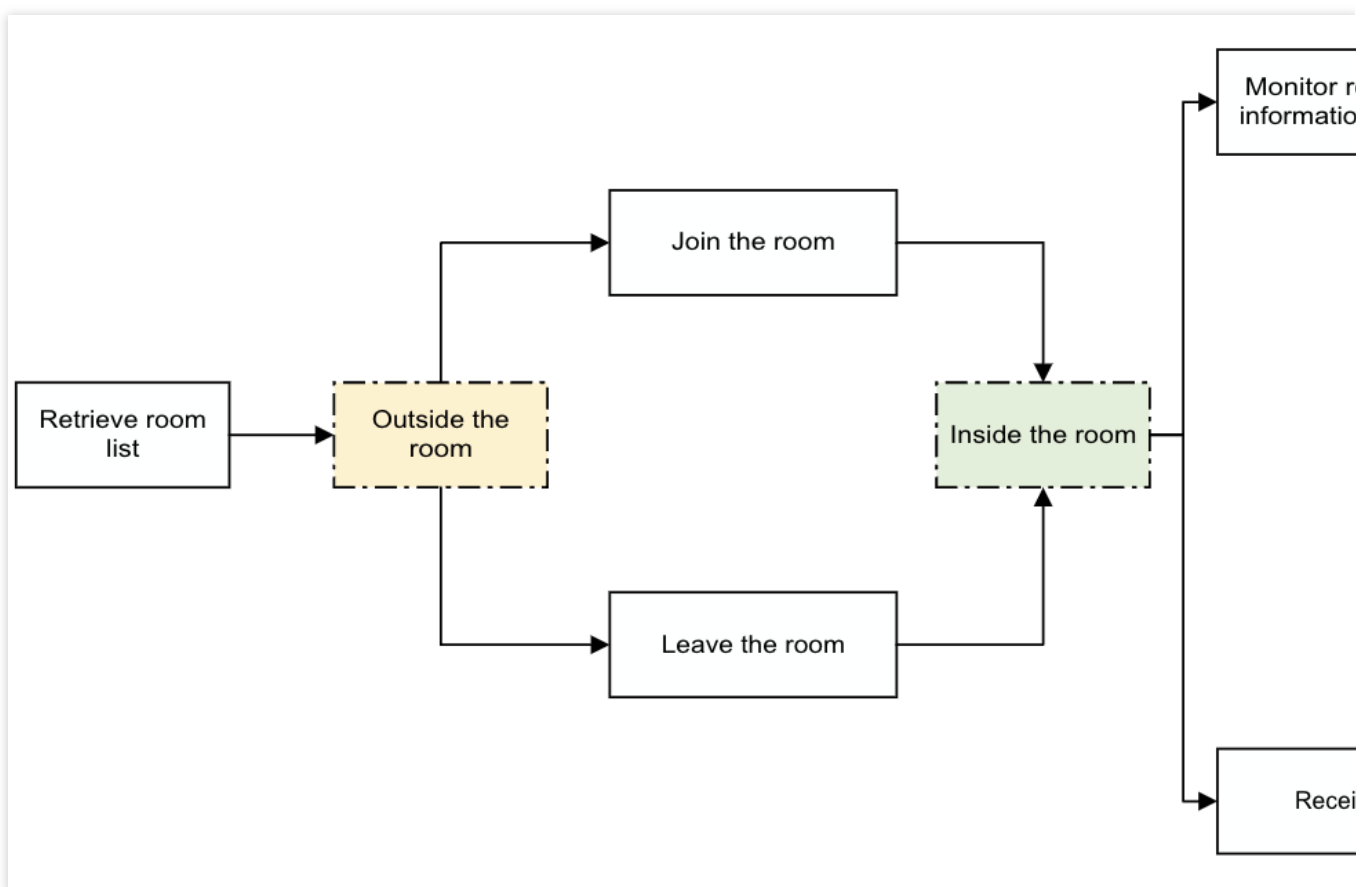
If the singer does not bring up the video upstream, they need to enable video upstream.

Synchronize the lyric progress of everyone through SEI information.

The singer can cut the song at any time during the singing process, and then download and sing the song/lyric again after the download is complete.

After the host leaves the room, the TRTC room will be dissolved.

【Audience】



The audience joins the TRTC room.

Listen for changes in the room's song and load the lyrics.

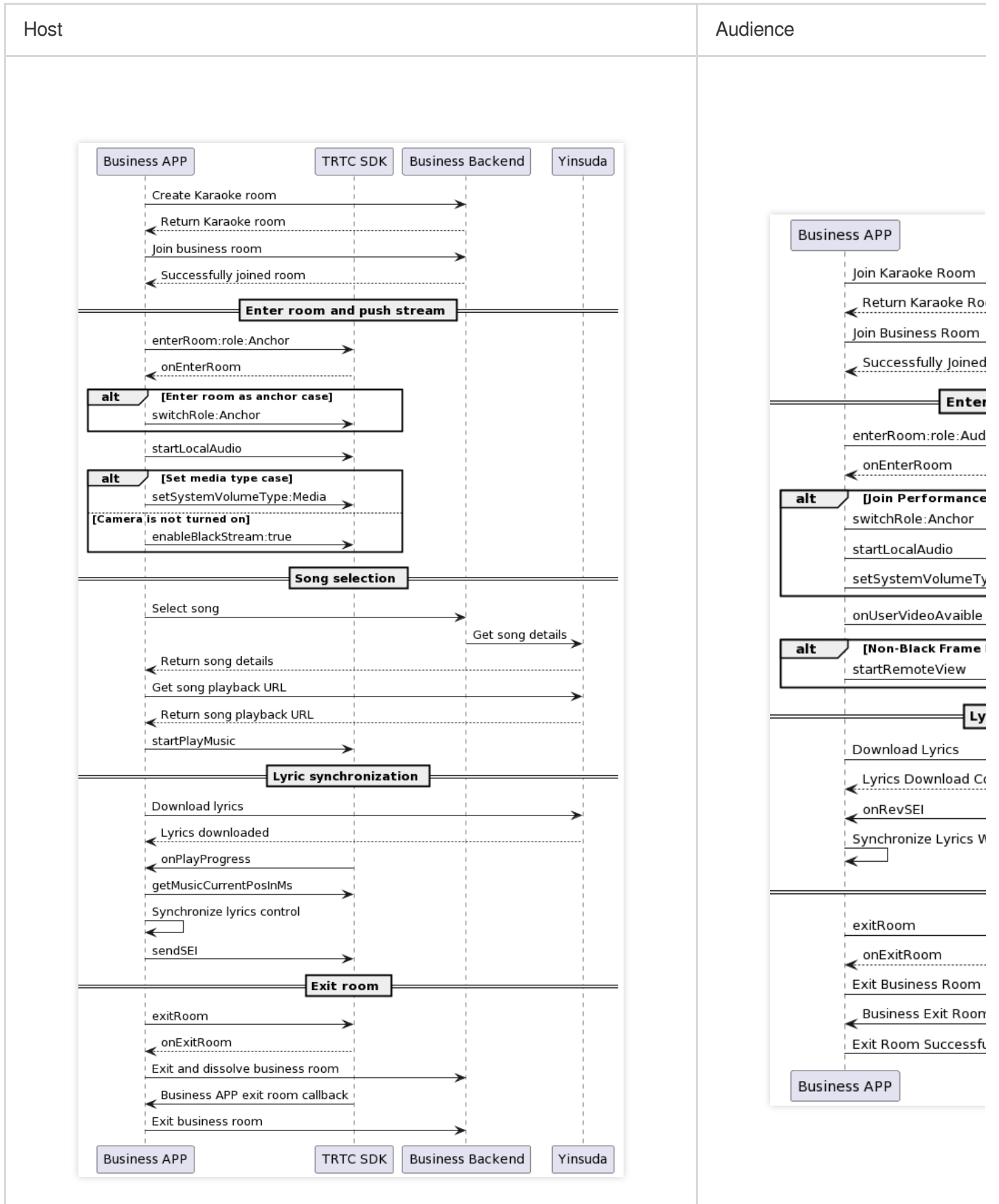
Pull the stream of the singer.

Parse the SEI information sent by the singer and synchronize the lyrics.

The main task is to listen for the SEI information of the song and update the corresponding song control.

### (3) API call sequence

The API calls for different roles are sequenced as follows:



**Note :**

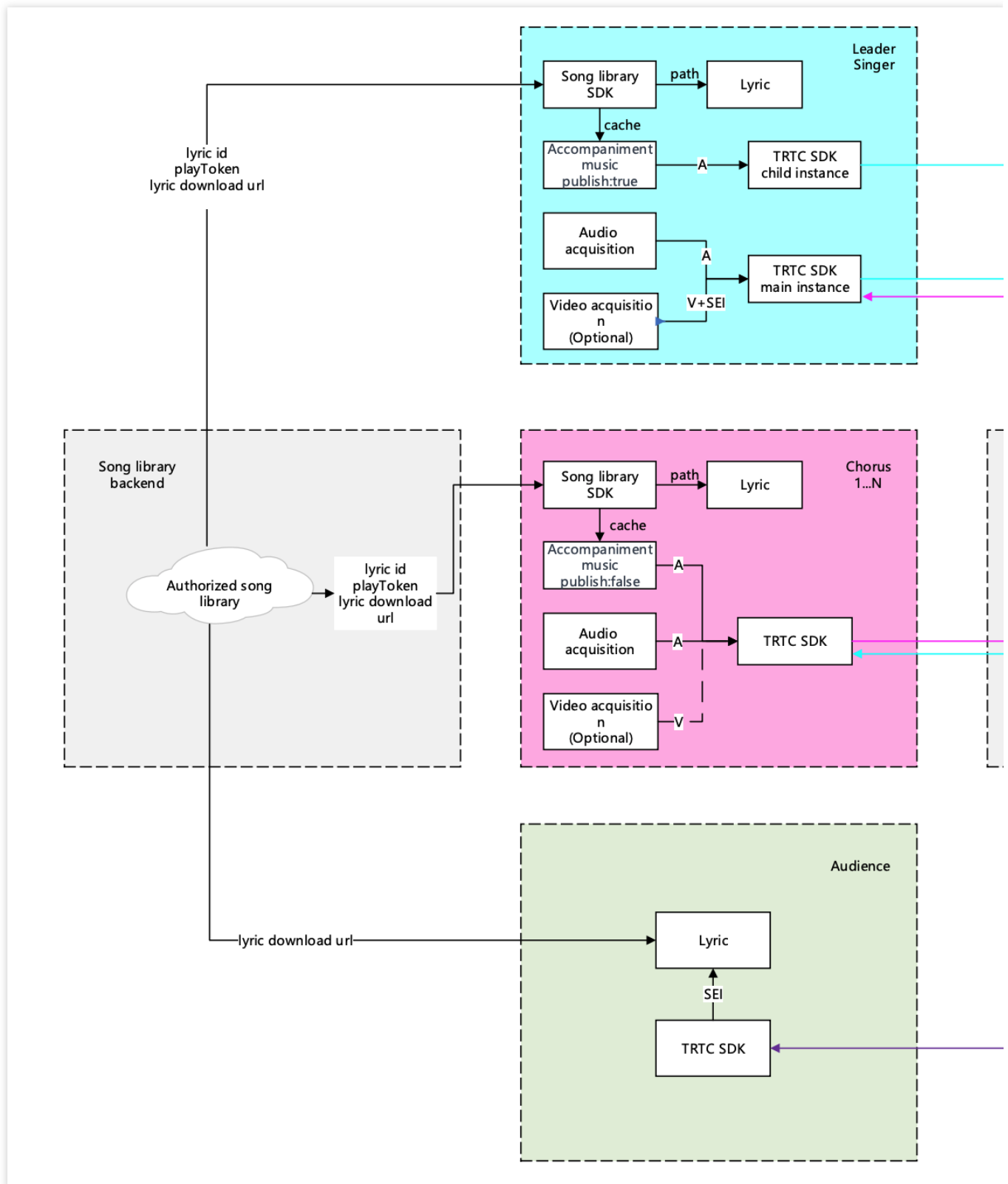
Given the technical threshold required for the above implementation solution, TRTC provides an open-source audio and video UI component called [TUIKaraoke](#) on its official website. By integrating the TUIKaraoke component into your project, you can add online karaoke scenes to your application with just a few lines of code, and experience TRTC's related capabilities in Karaoke scenarios, such as karaoke, seat management, gift giving and receiving, text chat, and more.

**Real-time Chorus**

Real-time chorus refers to playing songs simultaneously on various ends while connected, and then singing together on the seat. In multi-user mode, the singers can hear each other's voices almost without delay, achieving true real-time chorus.

**(1) Solution Architecture**

In terms of media streams, the singers push and pull streams to each other, and one ***lead singer pushes out the music***, while other ***singers play the music locally***, with time synchronization through NTP. In addition, the song and the voices of all singers are mixed and processed into one stream by the mixing robot, and then pushed back to the TRTC room. The audience only needs to pull one stream to hear the synchronized voices from all ends, perfectly achieving the effect of multi-person chorus. The solution architecture for real-time chorus is shown in the following figure.



The advantages of this solution are:

It reduces end-to-end latency.

It provides a solution for users to join the chorus midway.

It accurately synchronizes music, lyrics, and vocals between different ends.

It improves the performance of devices on different ends and the accuracy of local time, and reduces the impact of network environment latency.

**Note :**

Depending on business needs, you can choose a real-time chorus solution for either pure audio or audio and video scenarios. If it is a pure audio scenario, black frames need to be added to send SEI messages for lyric synchronization.

The lead singer needs to use a sub-instance to upstream both the music and vocals at the same time; other singers only need to pull each other's vocal streams and play the music locally; the audience only needs to pull one mixed stream.

The figure shows the RTC viewing solution, where the mixing robot pushes the mixed stream back to the RTC room; in the CDN viewing solution, the mixing robot pushes the mixed stream to the live CDN, and the audience pulls the CDN stream to watch.

**(2) Specific Implementation**

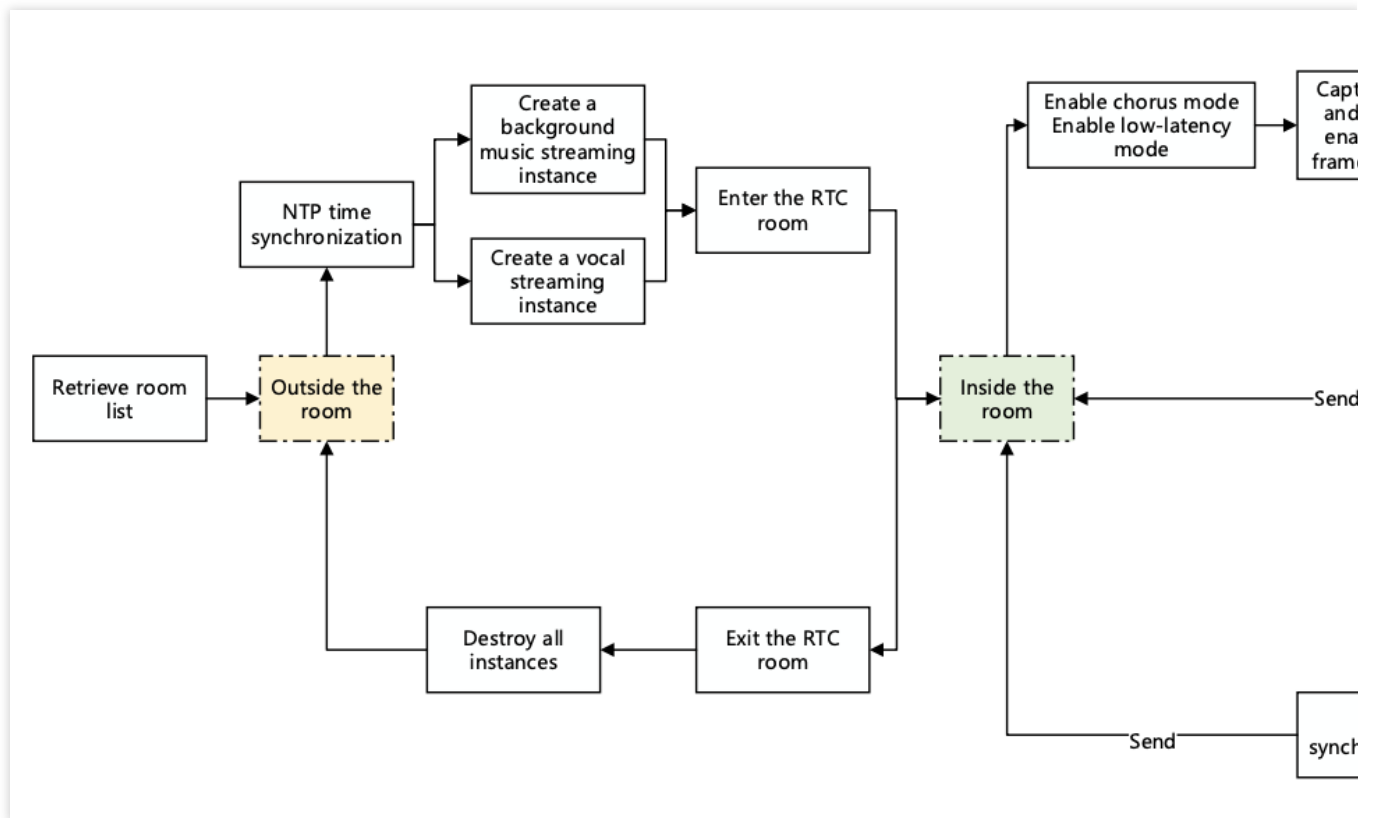
We can divide the users in the online karaoke room into three roles: lead singer, chorus, and audience, as shown in the table below.

| Role        | Description                                                                                                                     | Differences                                                                                                                                          |
|-------------|---------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lead Singer | The lead singer is responsible for selecting songs, sending chorus signals, and sending SEI messages.                           | The role must be an Anchor<br>Upstream music and vocals<br>Song selection and initiating chorus<br>Pushing back mixed stream<br>Sending SEI messages |
| Chorus      | The chorus can receive and process chorus signals, and participate in the chorus on the seat.                                   | The role must be an Anchor<br>Upstream vocals<br>Play music locally<br>Receive chorus signals                                                        |
| Audience    | After entering the karaoke room, the audience can pull the stream from the seat and also participate in the chorus on the seat. | The role must be an Audience<br>Downstream mixed stream<br>Receive SEI messages<br>Apply to become an Anchor to go on the seat                       |

The basic implementation processes for different roles are shown in the following figure:

【Lead Singer】





The lead singer needs to select a song and send chorus signals.

The lead singer creates a sub-instance to push vocals and music, and pulls the vocals of other singers.

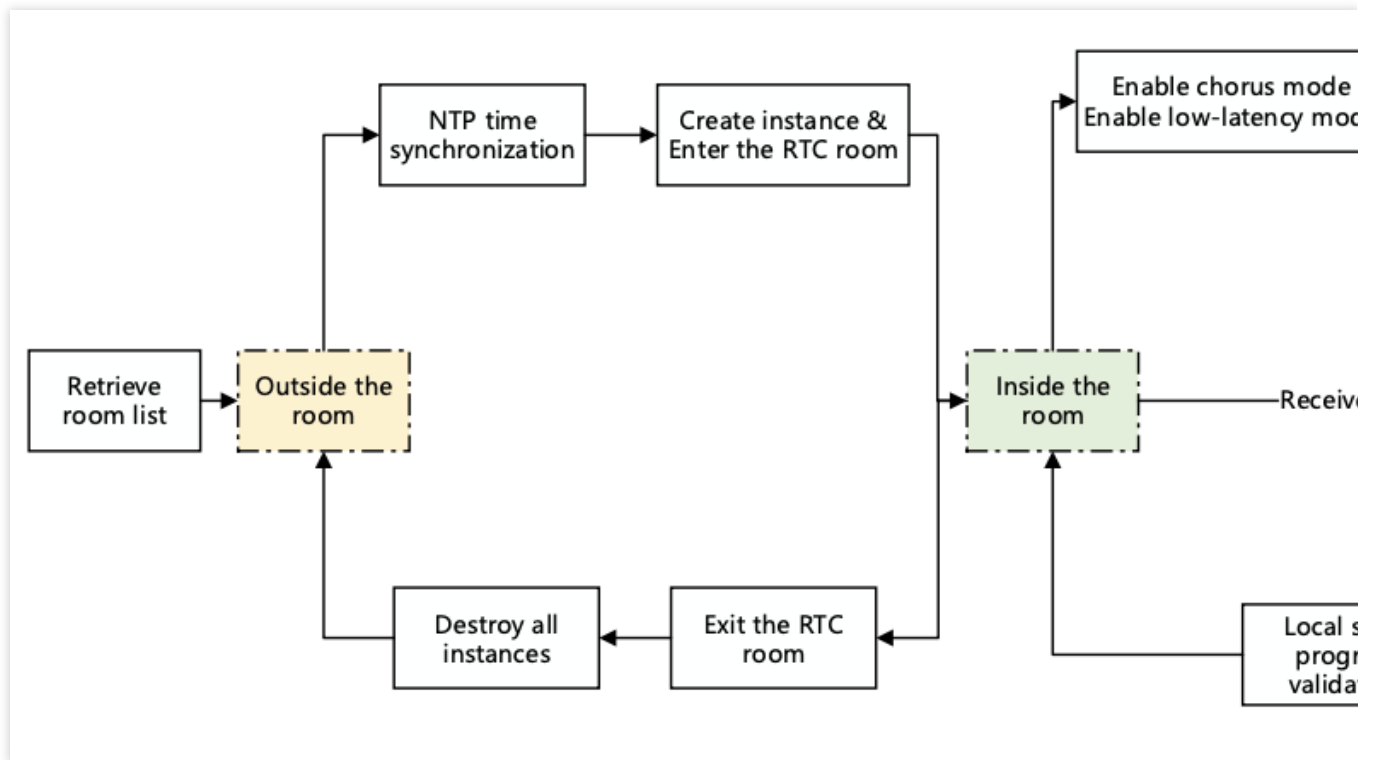
After pushing the stream, the lead singer is responsible for initiating the mixed stream push task.

After starting the performance, play the music and synchronize the lyrics through the playback progress callback.

SEI messages need to be sent to synchronize the song progress on the audience end.

All singers need to calibrate the local song playback progress according to NTP.

【Chorus】



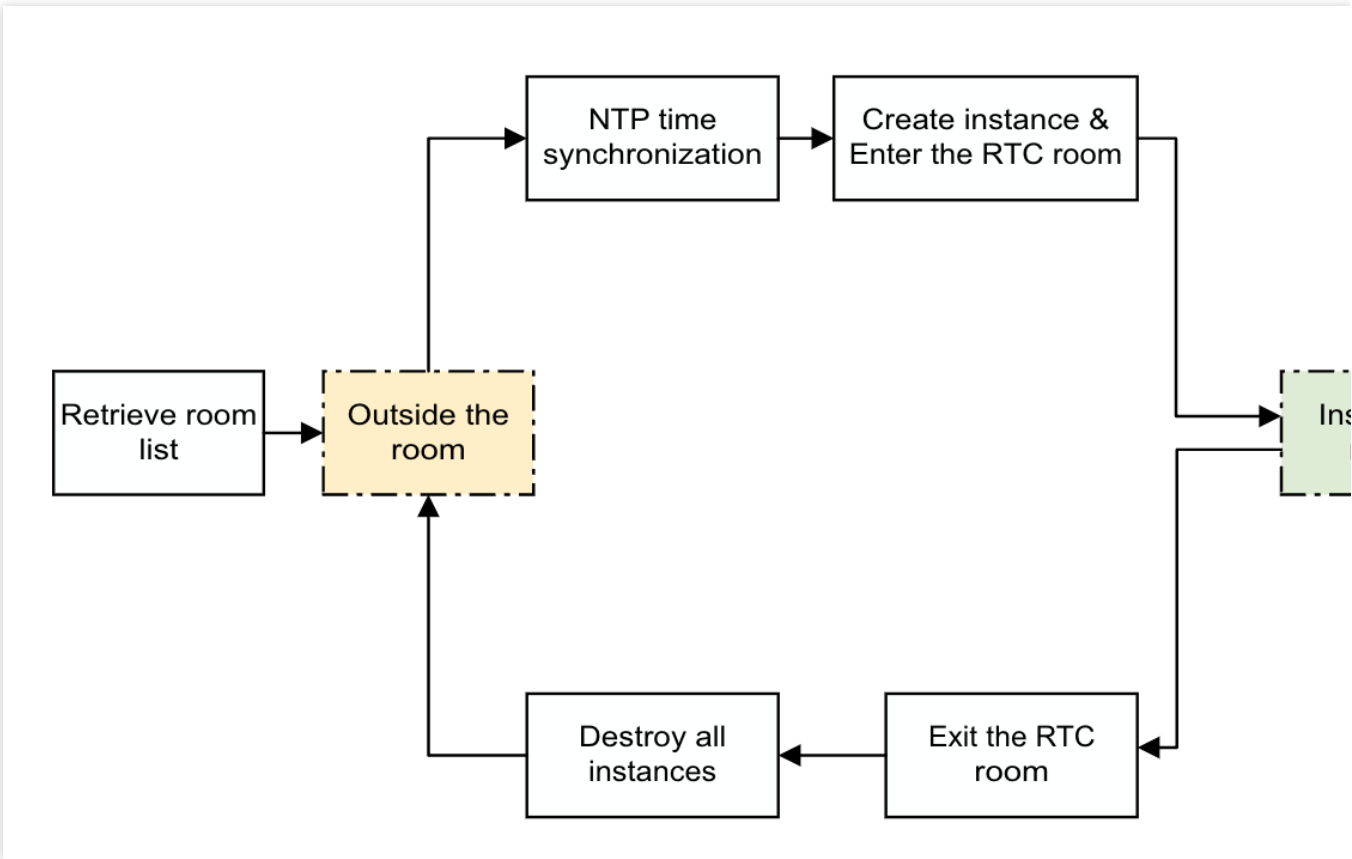
The chorus pushes one vocal stream and pulls the vocal stream of the user on the seat.

The chorus needs to listen for and receive chorus signals, and pre-load music resources.

After starting the performance, play the music locally, and the chorus synchronizes the lyrics through the playback progress callback.

All singers need to calibrate the local song playback progress according to NTP.

【Audience】



Pull the mixed stream to listen to the chorus.

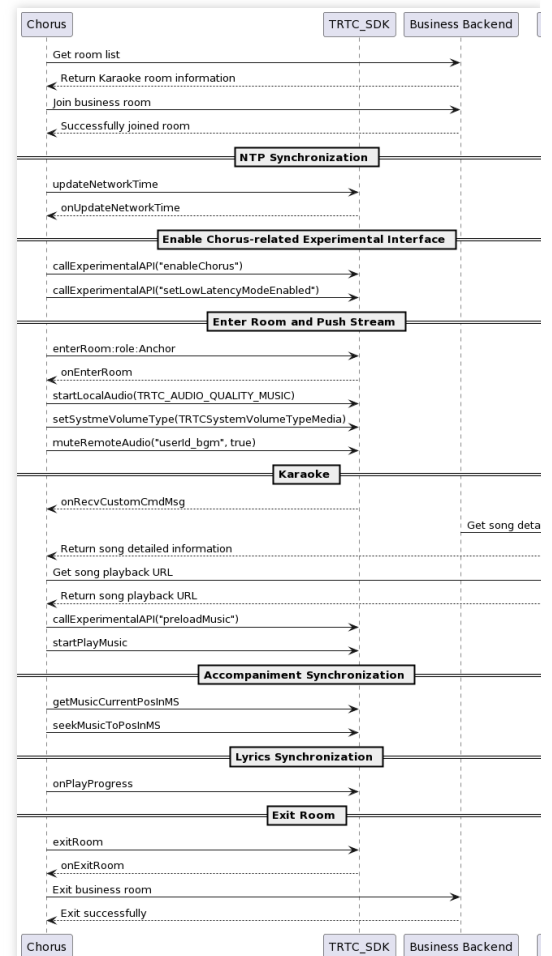
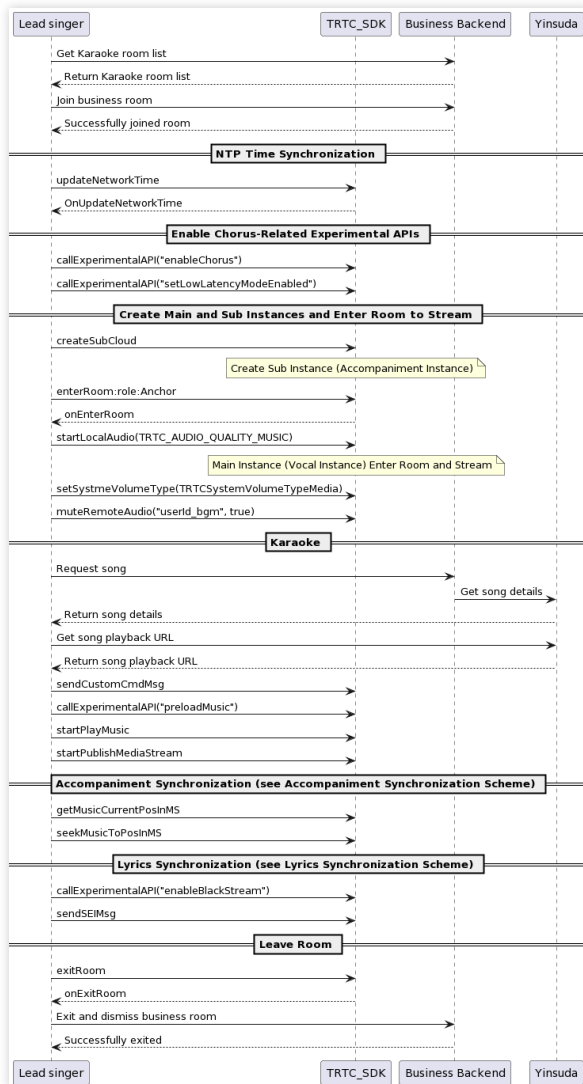
Parse the song progress information in the SEI of the mixed stream for lyric synchronization.

After going on the seat, stop pulling the mixed stream, switch to pulling the vocal stream on the seat, and start the chorus mode.

(3) API call sequence

The sequence of API calls for different roles is as follows:

| Lead singer API sequence | Chorus API sequence |
|--------------------------|---------------------|
|                          |                     |



## Note :

Considering the technical expertise required for the above implementation, TRTC's official website provides an open-source audio and video UI component called TUIKaraoke, which can be integrated into your project. With just a few lines of code, you can add real-time karaoke scenes to your application and experience TRTC's related capabilities for KTV scenarios, such as singing, seat management, gift exchange, text chat, and more.

# Song Synchronization

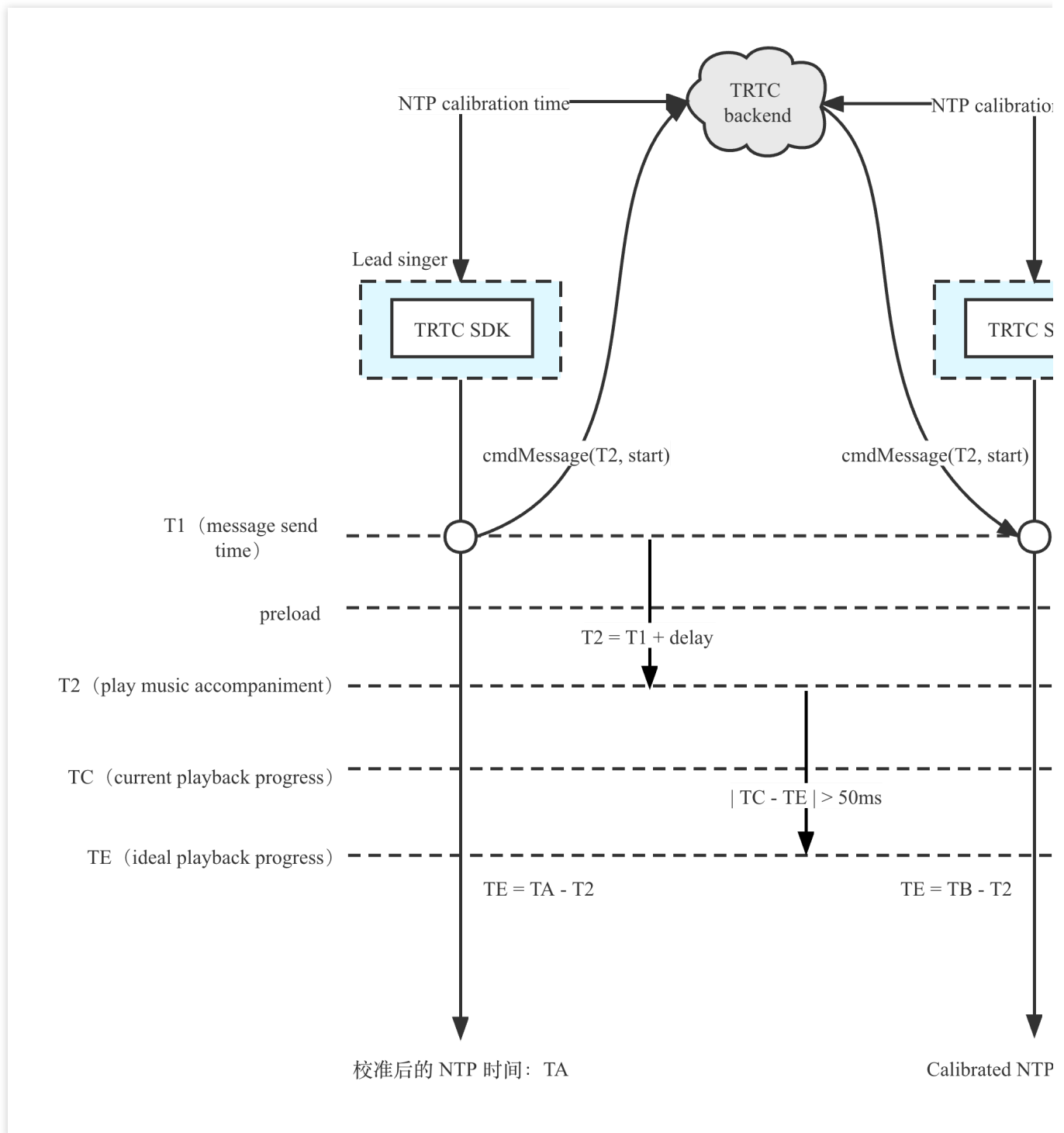
## iOS

Last updated : 2023-09-27 14:44:28

Real-time synchronization of song progress is required in the real-time solution to avoid increasing end-to-end delay due to song errors after the start of the performance. Synchronizing the song requires using NTP time. The local clocks of different devices are not consistent and there is a certain error, so Tencent Cloud's self-developed NTP service needs to be introduced. At the same time, users who join the chorus midway also need to synchronize the song progress, and only after synchronizing the progress can they participate in the chorus.

## Implementation process

The method of synchronizing songs is that the main singer user agrees to start playing the song at a future point in time (such as N seconds after delay), and other users participate in the chorus. The time of each end is based on NTP time, which will start synchronizing after the TRTC SDK is initialized.



The specific process is as follows:

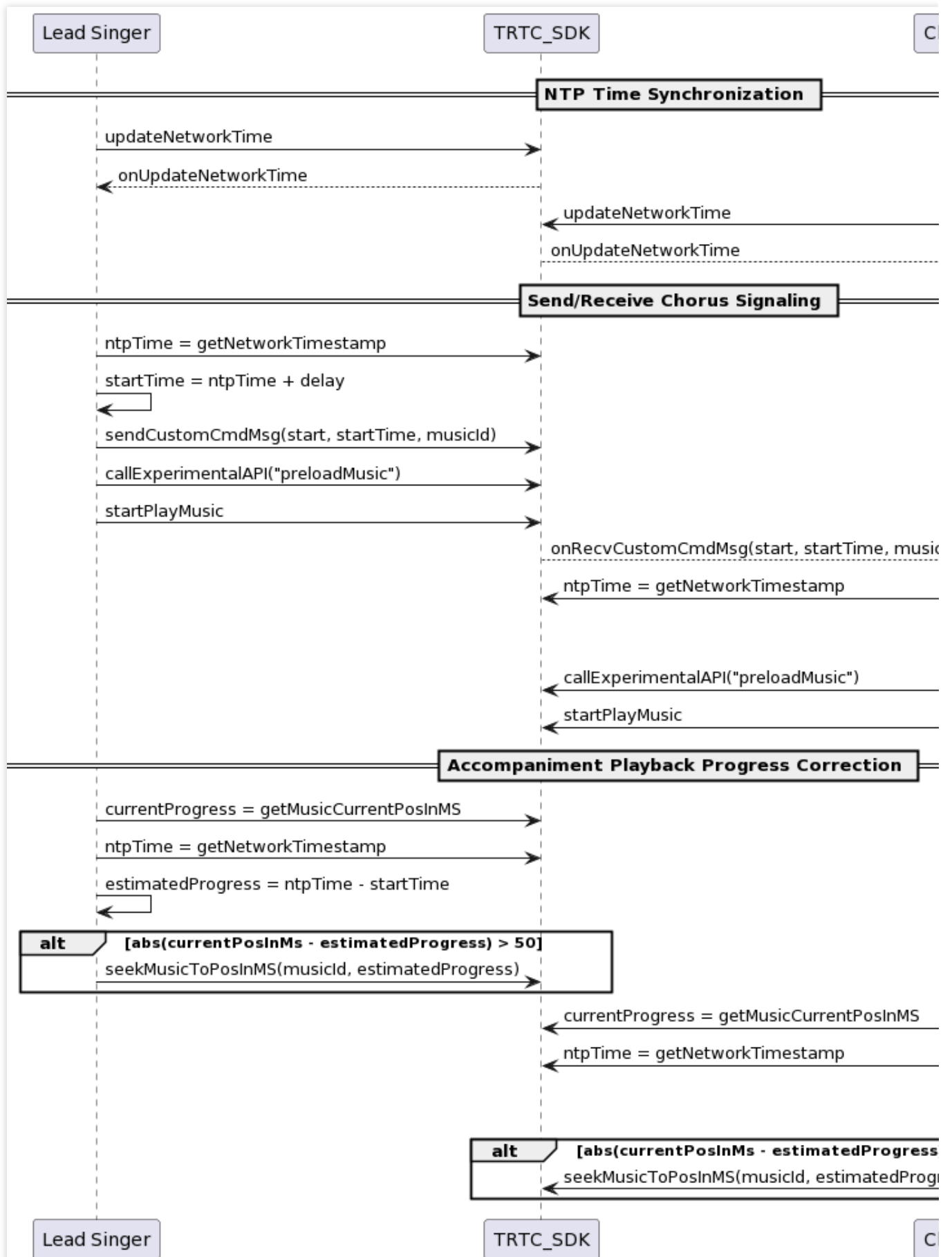
1. Each end performs NTP time calibration, updates and obtains the latest NTP time T to the TRTC cloud.
2. The main singer end sends a chorus signal (custom message) to agree on the start time T2 of the chorus.
3. Preload the song locally based on T2 and play it at a scheduled time.
4. Other chorus users execute step 3 after receiving the chorus signal.

5. During the process, the local song playback progress is verified, and seek calibration is performed when the difference between TE and TC exceeds 50ms.

**Note :**

The 50ms error here is a typical value, which can be adjusted appropriately according to the business tolerance. It is recommended to fluctuate around 50ms.

## Timing diagram

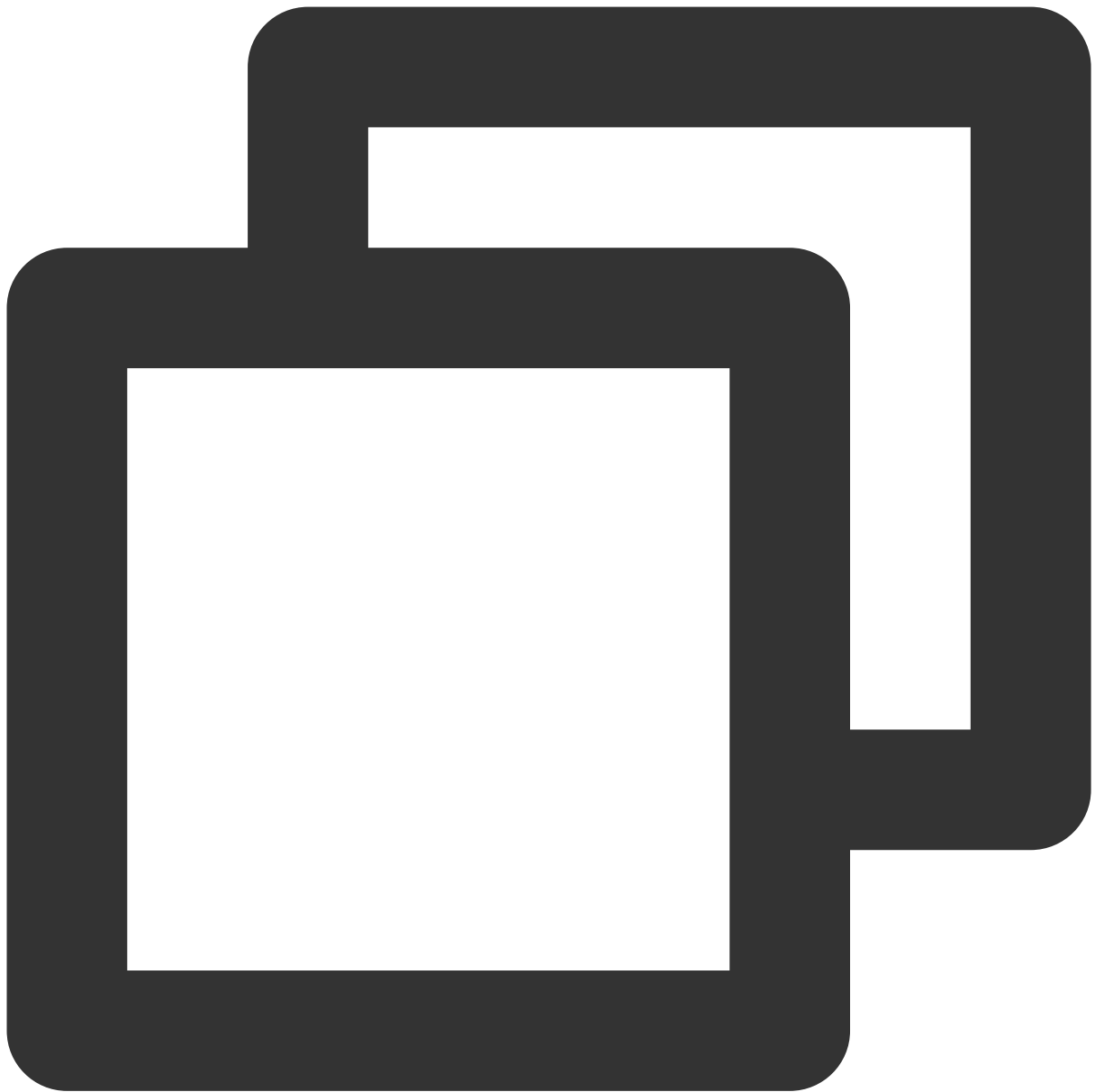




The song synchronization timing can mainly be divided into three parts: NTP time calibration, sending and receiving chorus signals, and correcting the song playback progress. The following will provide specific code implementation for these three parts.

## Key code implementation

### 1. NTP calibration time service

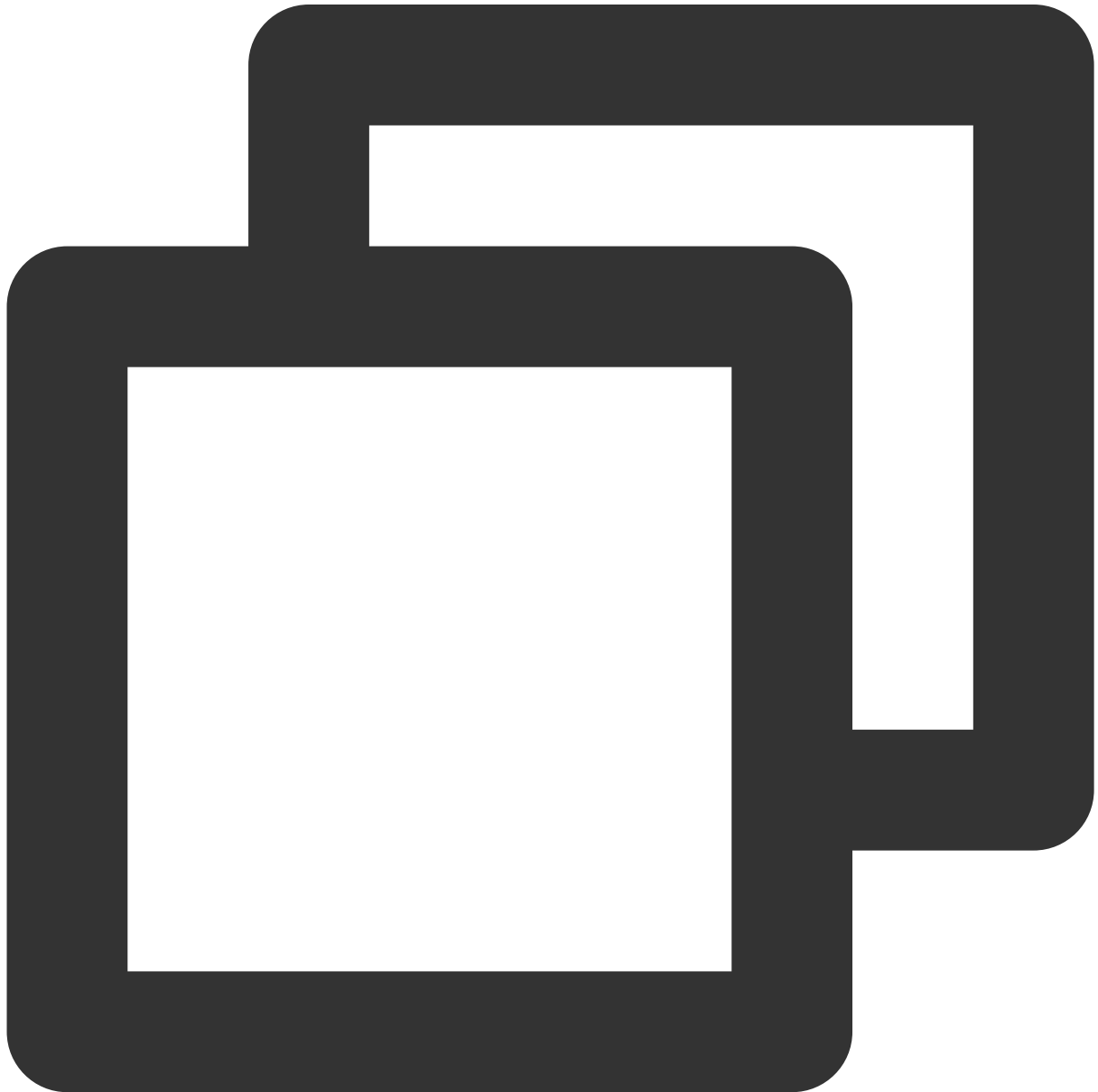


```
// Call the NTP time synchronization interface when entering the room.
[TXLiveBase updateNetworkTime];

// In the TXLiveBaseDelegate callback, determine whether the time synchronization is successful.
- (void)onUpdateNetworkTime:(int)errCode message:(NSString *)errMsg {
    // errCode 0: 0: Time synchronization is successful and the deviation is within 30ms;
    // 1: Time synchronization is successful, but the deviation may be over 30ms;
    // -1: Time synchronization failed.
    if (errCode == 0) {
        // Call TXLiveBase's getNetworkTimestamp to obtain the NTP timestamp.
        NSInteger ntpTime = [TXLiveBase getNetworkTimestamp];
    }
}
```

```
    } else {  
        // Call updateNetworkTime again to initiate a time synchronization.  
        [TXLiveBase updateNetworkTime];  
    }  
}
```

## 2. Sending chorus signals on the main singer end



```
NSDictionary *json = @{  
    @"cmd": @"startChorus",
```

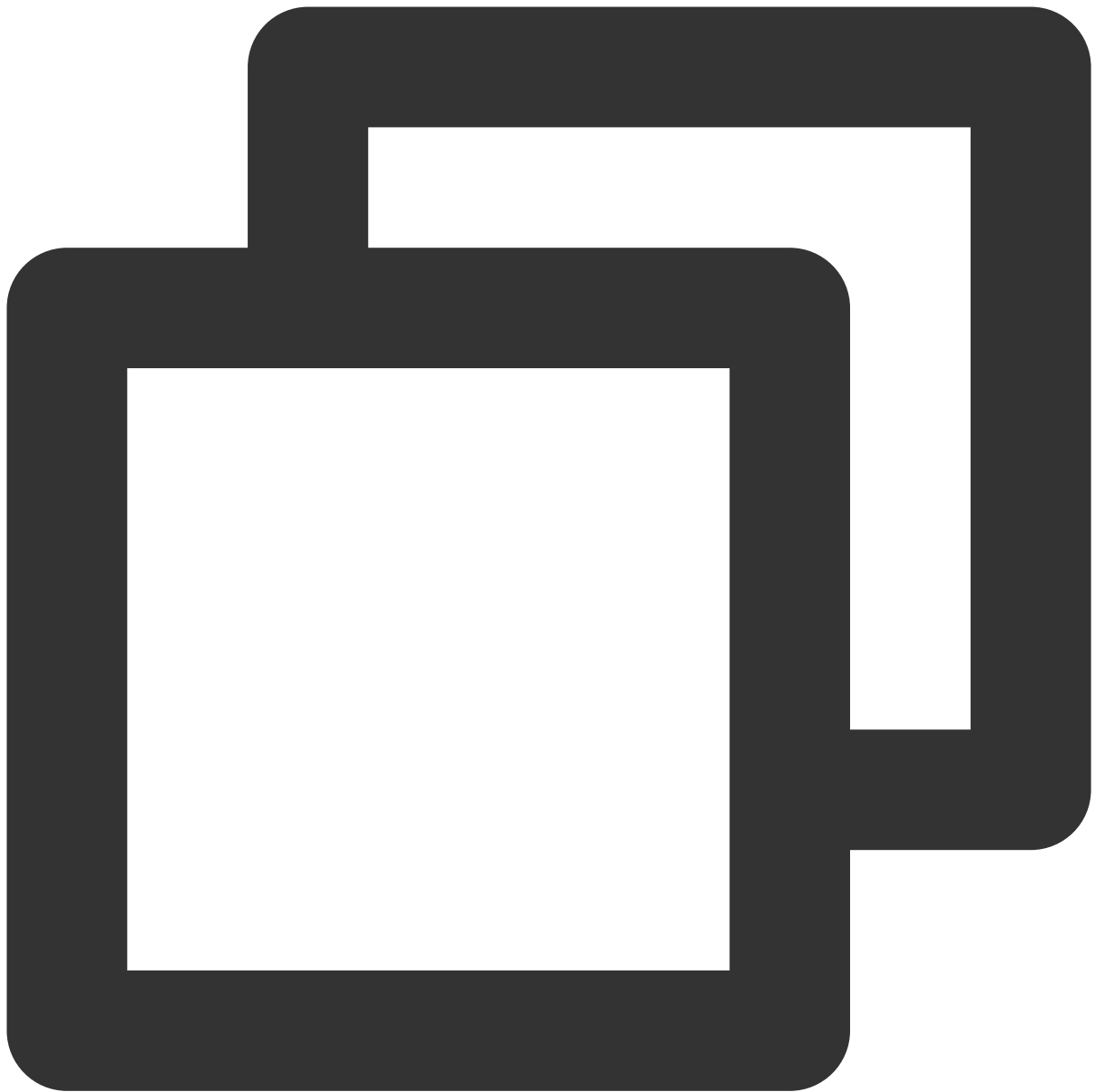
```
        // Scheduled time for a tutti.
        @"startPlayMusicTS": @(startTs),
        @"musicId": @"musicId",
        @"musicDuration": @(musicDuration),
    };
    NSString *jsonString = [self jsonStringFrom:json];
    [trtcCloud sendCustomMessage:jsonString reliable:NO];
```

**Note :**

It is recommended that the main singer send chorus signal messages to the room at a fixed time frequency in a loop, so that chorus users can join in midway;

The reason for **not using SEI messages to send chorus signals** is that the SEI information will be inserted into the video frame, causing a lot of invalid information to be carried in the video stream pulled by the audience side.

### 3. Receiving chorus signals on the chorus end



```
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:(NSInteger)cmdId seq:(UInt32)seq {  
    NSString *msg = [[NSString alloc] initWithData:message encoding:NSUTF8StringEncoding];  
    NSError *error;  
    NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[msg dataUsingEncoding:NSUTF8StringEncoding] options:NSJSONReadingMutableContainers error:&error];  
  
    NSObject *cmdObj = [json objectForKey:@"cmd"];  
  
    NSInteger musicDuration = [[json objectForKey:@"musicDuration"] integerValue];  
}
```

```

NSString *cmd = (NSString *)cmdObj;

// tutti command
if ([cmd isEqualToString:@"startChorus"]) {
    // tutti start time
    NSObject *startPlayMusicTsObj = [json objectForKey:@"startPlayMusicTS"];
    NSString *musicId = [json objectForKey:@"musicId"];

    NSInteger startPlayMusicTs = ((NSNumber *)startPlayMusicTsObj).longLongValue;
    // The difference between the scheduled tutti time and the current time.
    NSInteger startDelayMS = labs(startPlayMusicTs - [TXLiveBase getNetworkTime]);
    // Start preloading, and jump the song progress according to the difference
    // between the scheduled duet time and the current NTP time.
    NSDictionary *jsonDict = @{
        @"api": @"preloadMusic",
        @"params": @{
            @"musicId": @(musicId),
            @"path": path,
            @"startTimeMS": @(startDelayMS),
        },
    };
    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict options:0];
    NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEncoding];
    [subCloud callExperimentalAPI:jsonString];

    // play music
    TXAudioMusicParam *param = [[TXAudioMusicParam alloc] init];
    param.ID = musicId;
    param.path = url;
    param.loopCount = 0;
    param.publish = NO;

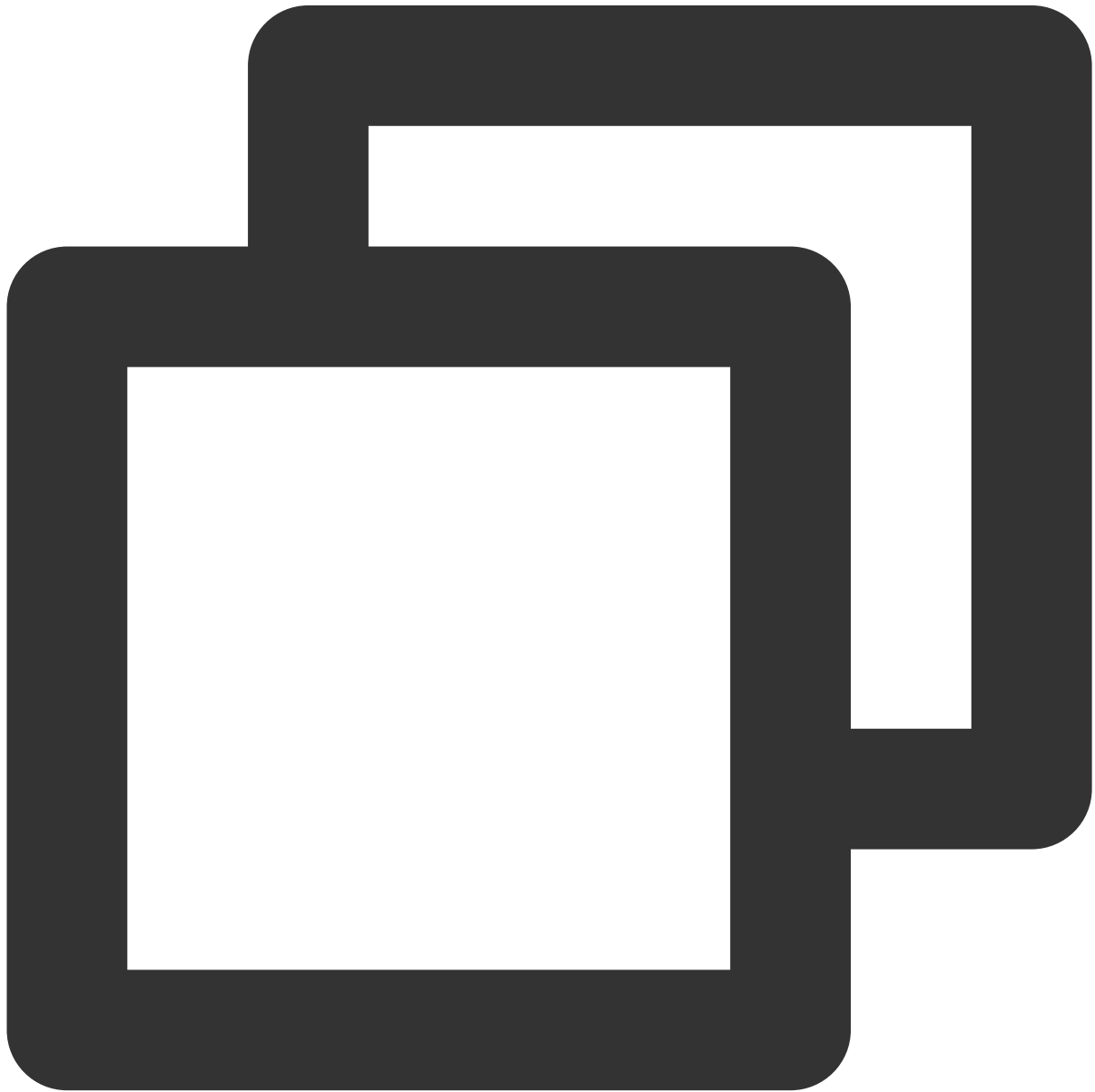
    [[subCloud getAudioEffectManager] startPlayMusic:param onStart:^(NSInteger progressMs, NSInteger durationMs) {
        // star play callback
    } onProgress:^(NSInteger progressMs, NSInteger durationMs) {
        // lyric progress callback
    } onComplete:^(NSInteger errCode) {
        // play completely callback
    }];
}
}

```

**Note :**

After the chorus end receives the first startChorus signal, the status should be changed from "not singing" to "singing", and no longer respond to the startChorus signal to avoid restarting the BGM playback.

#### 4. Song playback progress correction



```
self.startPlayChorusMusicTs; // The originally scheduled tutti time.
// Current playback progress
NSInteger currentProgress = [[self audioEffecManager] getMusicCurrentPosInMS:self.c
// The ideal playback time progress of the current song.
NSInteger estimatedProgress = [TXLiveBase getNetworkTimestamp] - self.startPlayChor
if (estimatedProgress >= 0 && labs(currentProgress - estimatedProgress) > 50) {
```

```
// When the playback progress exceeds 50ms, make adjustments.  
[[subCloud getAudioEffectManager] seekMusicToPosInMS:self.currentPlayMusicID pt  
}
```



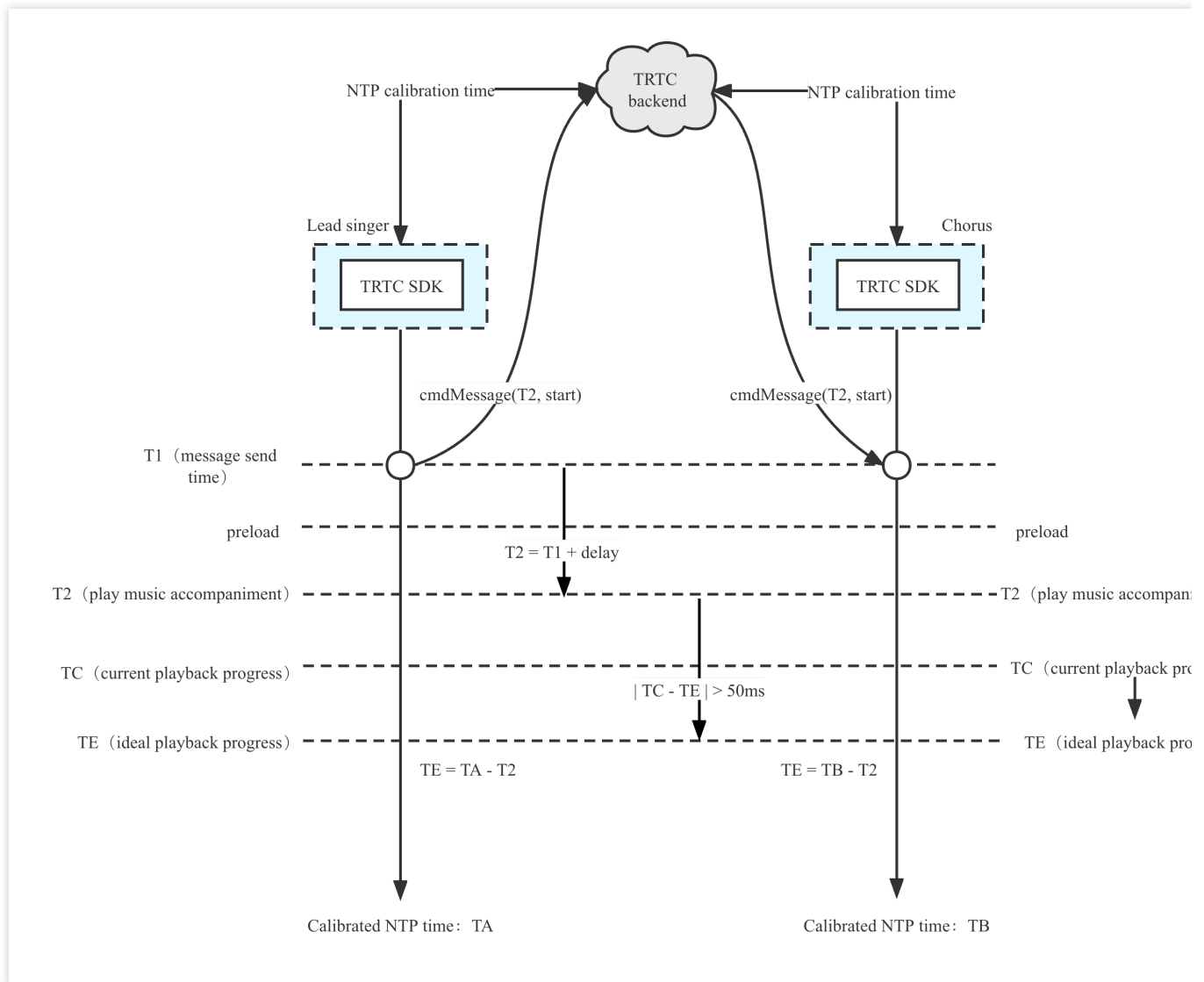
# Android

Last updated : 2023-09-26 16:45:13

Real-time synchronization of song progress is required in the real-time solution to avoid increasing end-to-end delay due to song errors after the start of the performance. Synchronizing the song requires using NTP time, as the local clocks of different devices are not consistent and have some error. Therefore, Tencent Cloud's self-developed NTP service needs to be introduced. In addition, users who join the chorus midway also need to synchronize the song progress before they can participate in the chorus.

## Implementation process

The practice of song synchronization is as follows: The lead singer user agrees to start playing the song at a certain point in the future (e.g., after a delay of N seconds), and other users participate in the chorus. The time of each end is based on NTP time, which will be synchronized after TRTC SDK initialization.



The specific process is as follows:

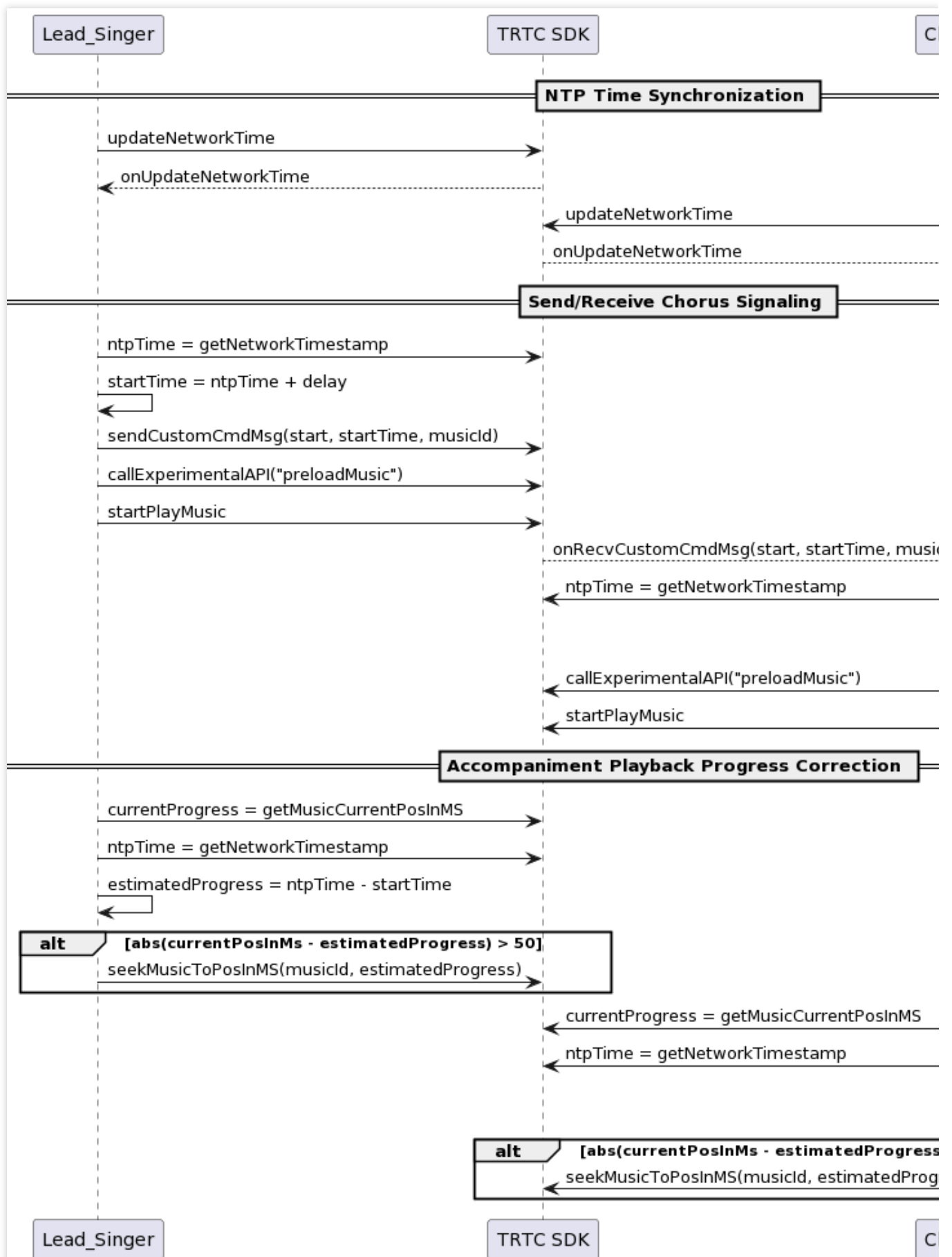
1. Each end performs NTP calibration, updates and obtains the latest NTP time T from the TRTC cloud.
2. The lead singer sends a chorus signaling (custom message), agreeing on the start time T2 for the chorus.
3. The local end preloads the song according to T2 and plays it on schedule.
4. Other chorus users perform step 3 after receiving the chorus signaling.
5. During the process, the local song playback progress is checked, and when the difference between TE and TC exceeds 50ms, seek calibration is performed.

**Note:**

The 50ms error mentioned here is a typical value, and can be adjusted according to the tolerance of the business. It is recommended to fluctuate around 50ms.

## Timing diagram

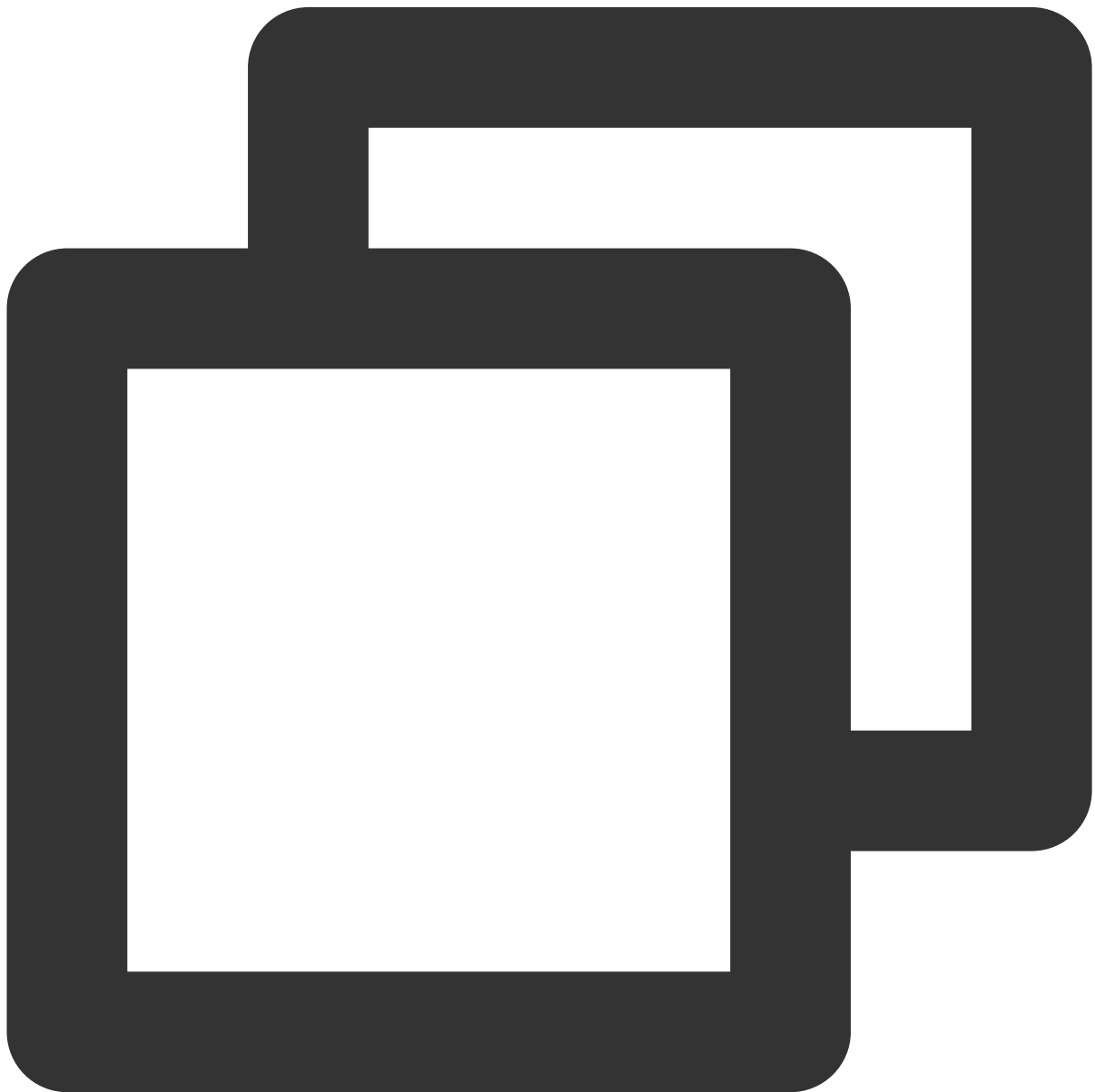




The song synchronization timing can be mainly divided into three parts: NTP calibration, sending and receiving chorus signaling, and song playback progress correction. The specific code implementation for these three parts will be provided below.

## Key code implementation

### 1. NTP calibration service

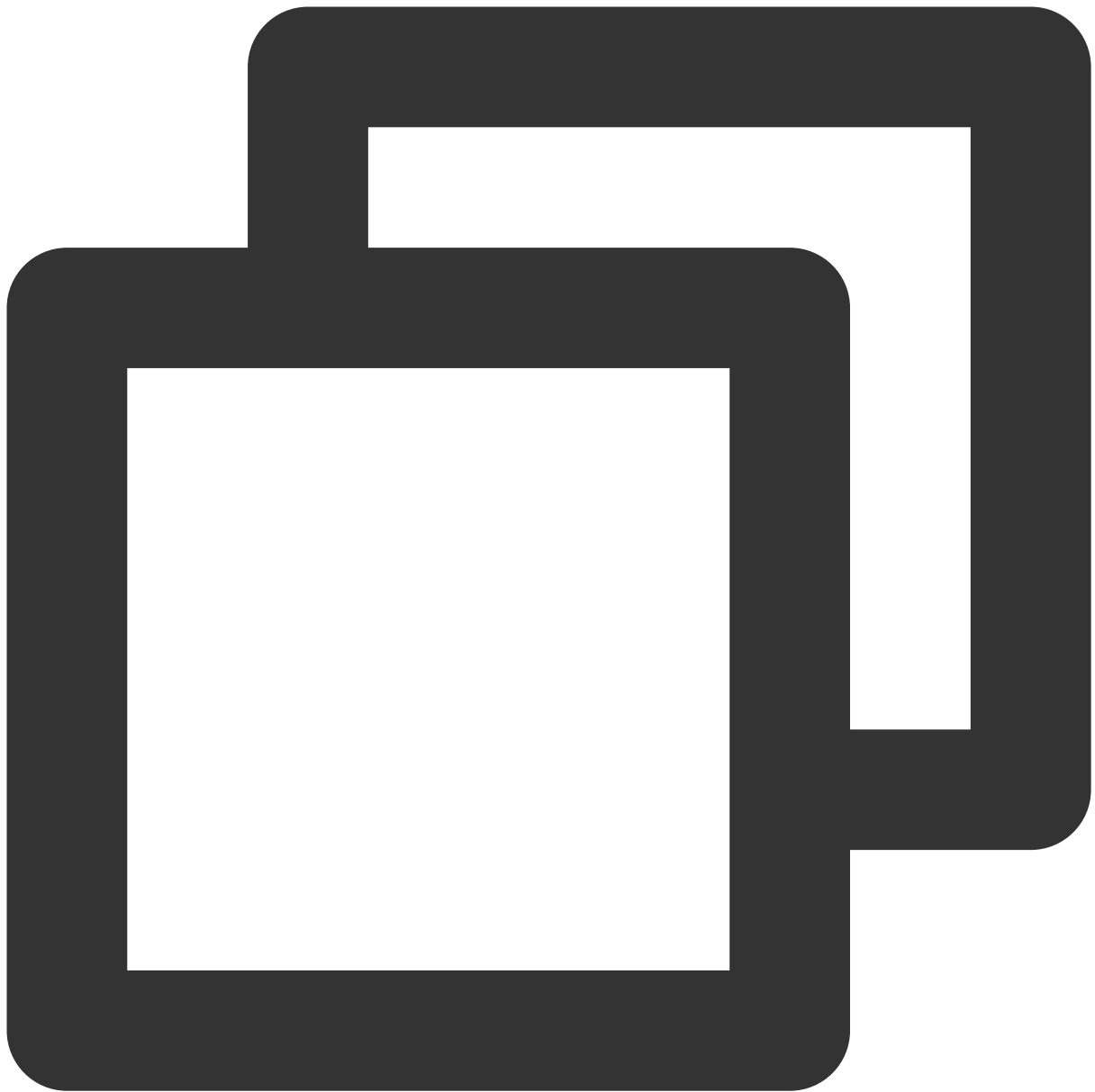


```
TXLiveBase.setListener(new TXLiveBaseListener() {
```

```
@Override
public void onUpdateNetworkTime(int errCode, String errMsg) {
    super.onUpdateNetworkTime(errCode, errMsg);
    // errCode 0: Calibration is successful and the deviation is within 30ms;
    //          1: Calibration is successful, but the deviation may be more than
    //          -1: Calibration failed.
    if (errCode == 0) {
        // Call getNetworkTimestamp of TXLivebase to get the NTP timestamp.
        long ntpTime = TXLiveBase.getNetworkTimestamp();
    } else {
        // Call updateNetworkTime again to start a calibration.
        TXLiveBase.updateNetworkTime();
    }
}

});
TXLiveBase.updateNetworkTime();
```

## 2. Lead singer sends chorus signaling



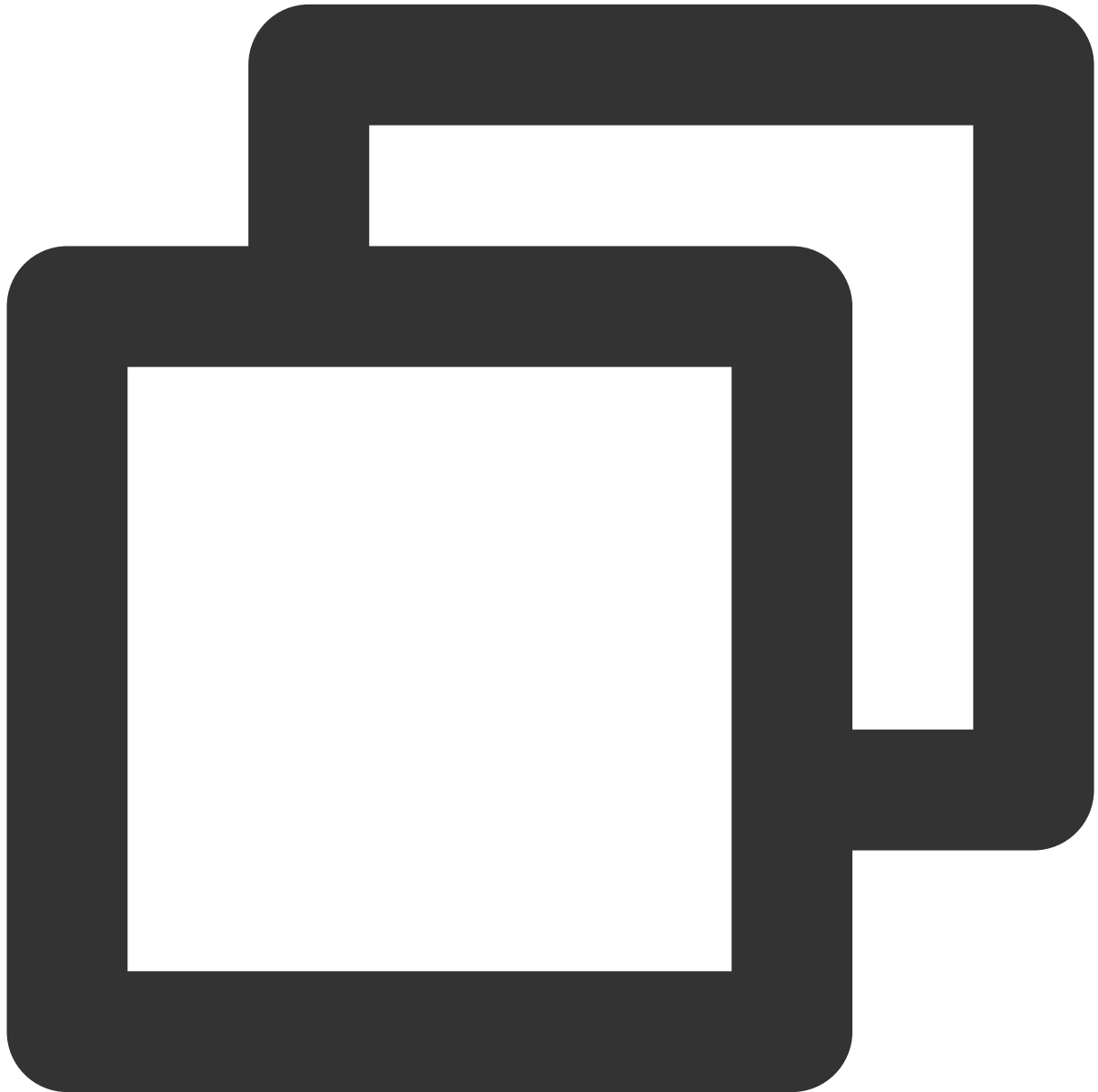
```
JSONObject jsonObject = new JSONObject();
jsonObject.put("cmd", "startChorus");
// Agree on a time for the chorus.
jsonObject.put("startPlayMusicTS", startTs);
jsonObject.put("musicId", "musicId");
String body = jsonObject.toString();
mTRTCCloud.sendCustomCmdMsg(0, body.getBytes(), false, false);
```

**Note:**

It is recommended that the lead singer sends chorus signaling messages to the room at a fixed time interval, so that the chorus users can join the chorus midway.

Reason for **not using SEI messages to send chorus signaling**: SEI information will be inserted into the video frame, causing the video stream pulled by the audience side to carry a lot of invalid information.

### 3. Chorus end receives chorus signaling



```
public void onRecvCustomCmdMsg(String userId, int cmdID, int seq, byte[] message) {  
    JSONObject json = new JSONObject(new String(message, "UTF-8"));  
    String cmd = json.getString("cmd");  
}
```

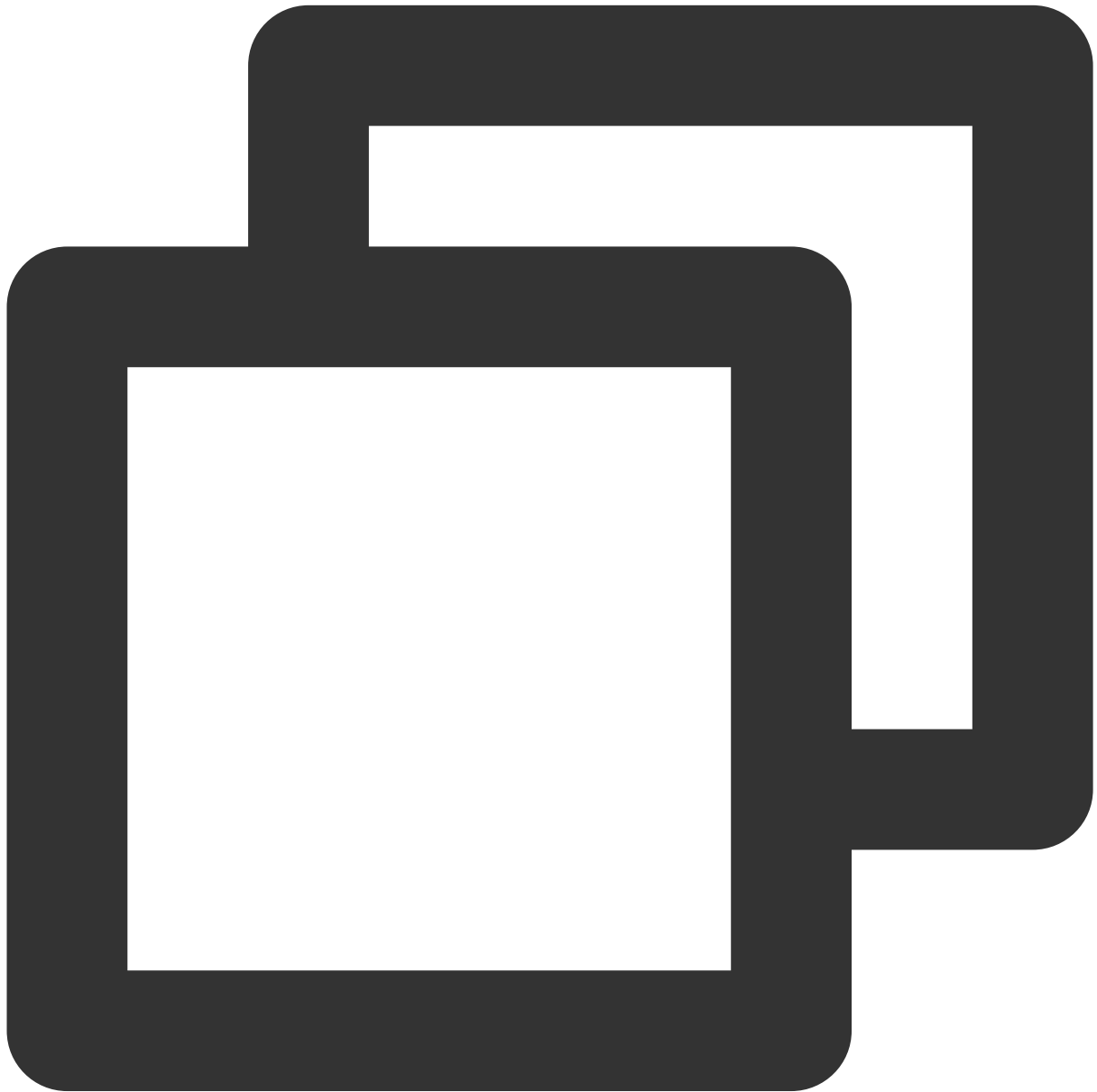


```
// Chorus command
if (cmd.equals("startChorus")) {
// Chorus start time
long startPlayMusicTs = json.getLong("startPlayMusicTS");
int musicId = json.getInt("musicId");
// The difference between the agreed chorus time and the current time
long delayMs = Math.abs(startPlayMusicTs - getNtpTime());
// Start preloading, and jump the song progress according to the agreed chorus
mTRTCCloud.callExperimentalAPI("{\"api\":\"preloadMusic\",\"params\":{\""
// Play the song
TXAudioEffectManager.AudioMusicParam param = new TXAudioEffectManager.AudioMus
param.publish = false;
mTRTCCloud.getAudioEffectManager().startPlayMusic(param);
}
```

**Note:**

After the chorus end receives the first startChorus signaling, the status should change from "not in chorus" to "in chorus", and no longer respond to startChorus signaling to avoid restarting BGM playback.

#### 4. Song playback progress correction



```
long mStartPlayMusicTs = "The initially agreed chorus time";
long currentProgress = subCloud.getAudioEffectManager().getMusicCurrentPosInMS(musicID);
// The ideal playback progress of the current song
long estimatedProgress = getNtpTime() - mStartPlayMusicTs;
// When the playback progress exceeds 50ms, make corrections
if (estimatedProgress >= 0 && Math.abs(currentProgress - estimatedProgress) > 50)
    subCloud.getAudioEffectManager().seekMusicToPosInMS(mMusicID, (int) estimatedProgress);
}
```

# Lyric Synchronization

## iOS

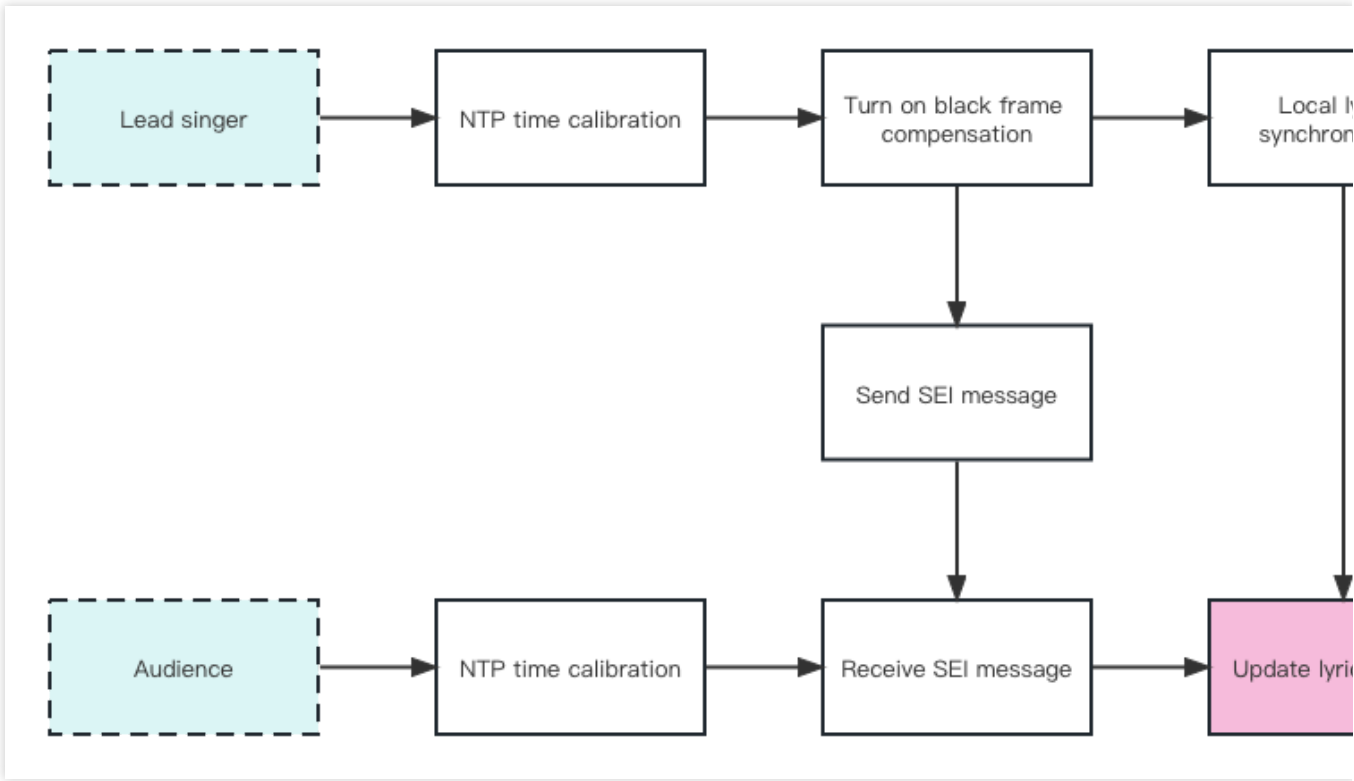
Last updated : 2023-09-27 15:11:25

### 1.1 Implementation process

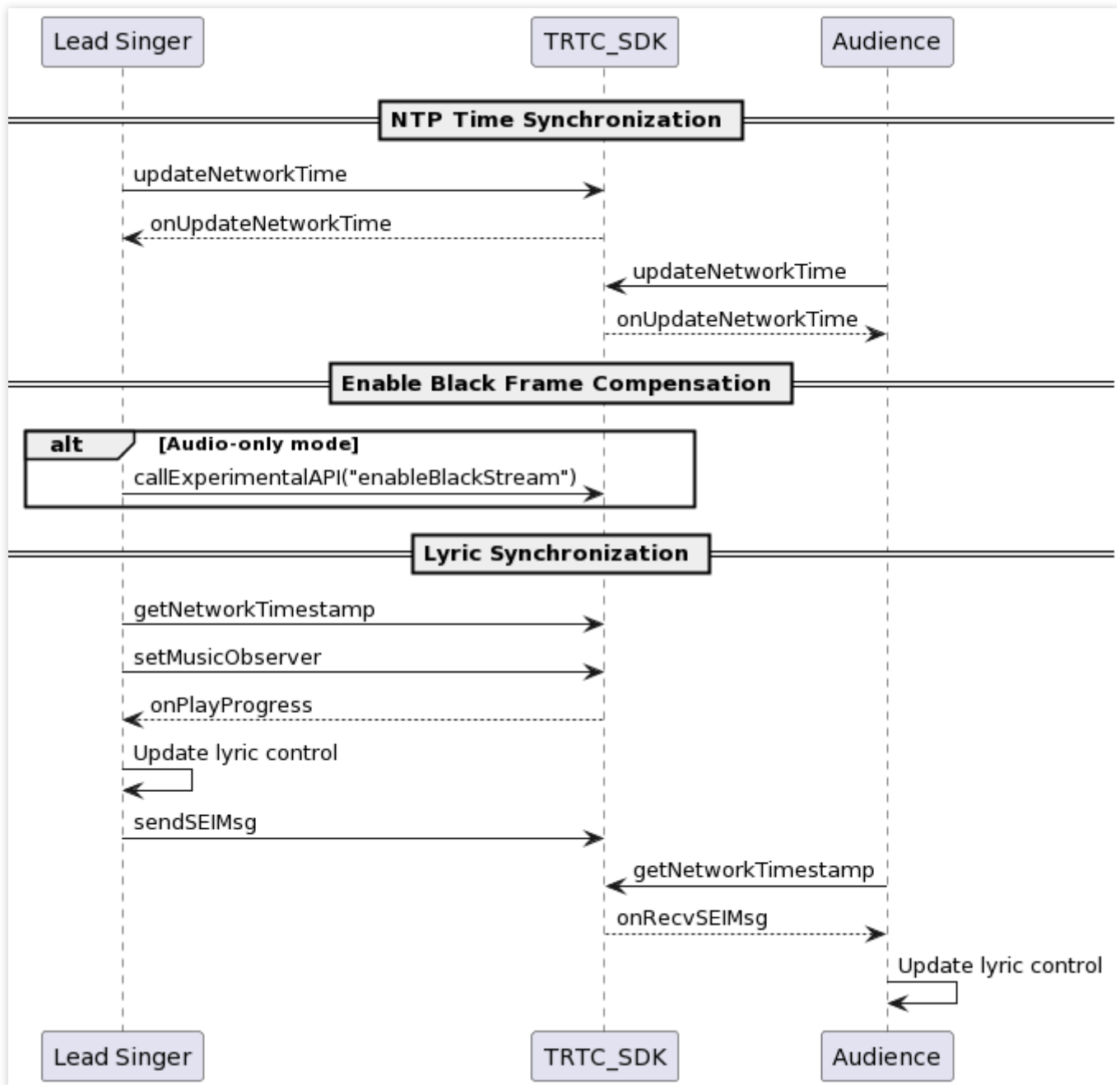
In the lyrics synchronization solution, the actions of the three different roles are as follows:

| Main Singer                                                                                                                        | Chorus                                                                        | Audience                                                              |
|------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| NTP time calibration<br>Enable black frame insertion<br>Send SEI messages<br>Local lyrics synchronization<br>Update lyrics control | NTP time calibration<br>Local lyrics synchronization<br>Update lyrics control | NTP time calibration<br>Receive SEI messages<br>Update lyrics control |

Among them, the main singer and chorus update the lyrics progress locally based on the synchronized song playback progress; the audience end needs to receive SEI messages containing the latest lyrics progress sent by the main singer end to update the local lyrics progress.



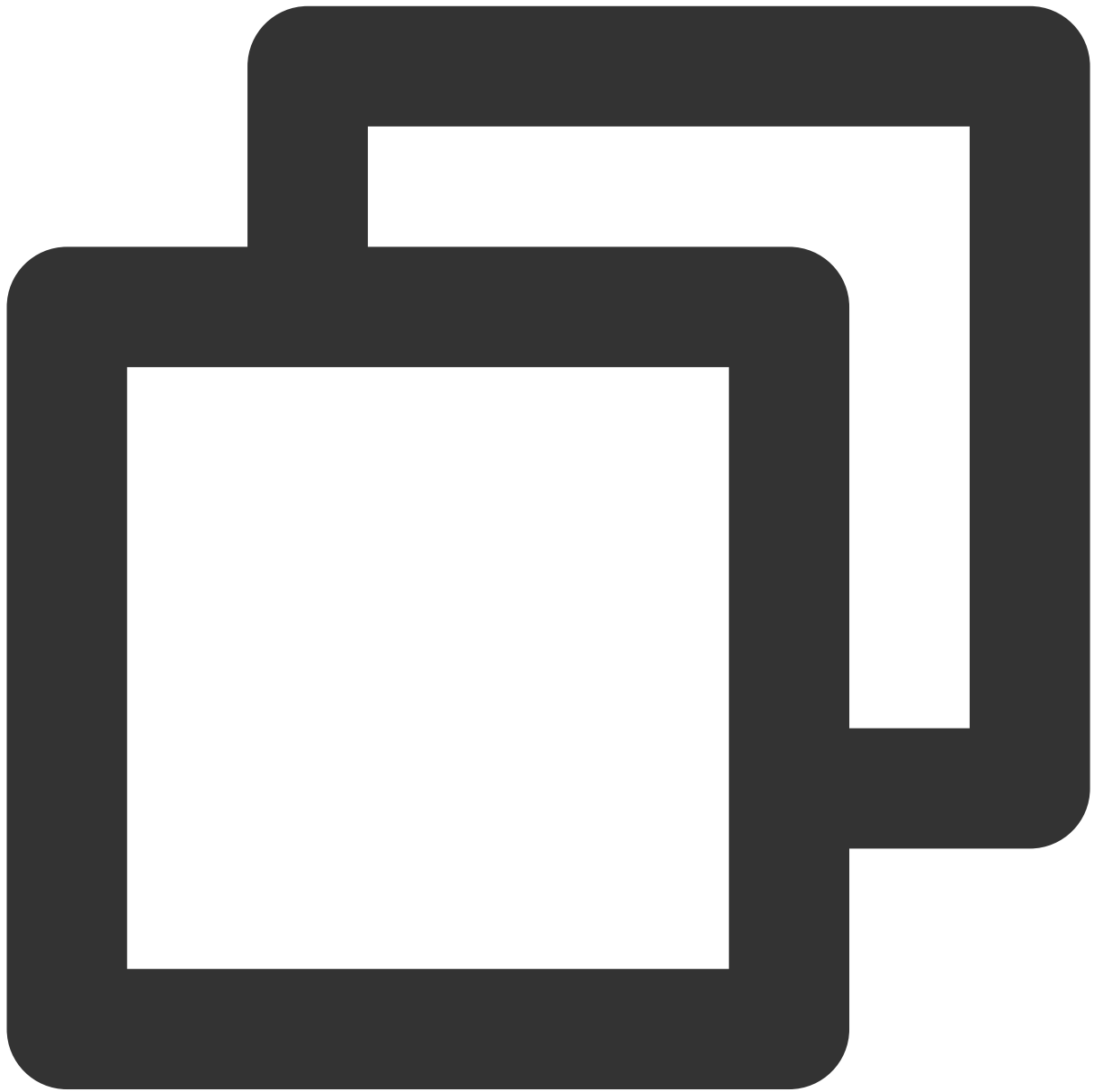
Timing diagram



The synchronization of lyrics timing can mainly be divided into three parts: NTP time synchronization, enabling black frame compensation, and local and remote lyrics synchronization. The code implementation of NTP time synchronization has been provided in the [Song Synchronization](#) document. The following will provide specific code implementation for the latter two parts.

## Key code implementation

## 1. Enable Black Frame Insertion



```
// In pure audio mode, the main instance (vocal instance)
// needs to enable black frame padding to carry SEI messages.
NSDictionary *jsonDic = @{
    @"api": @"enableBlackStream",
    @"params":
        @{
            @"enable": @(1)
        }
};
```

```
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr  
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin  
[trtcCloud callExperimentalAPI:jsonString];
```

**Note:**

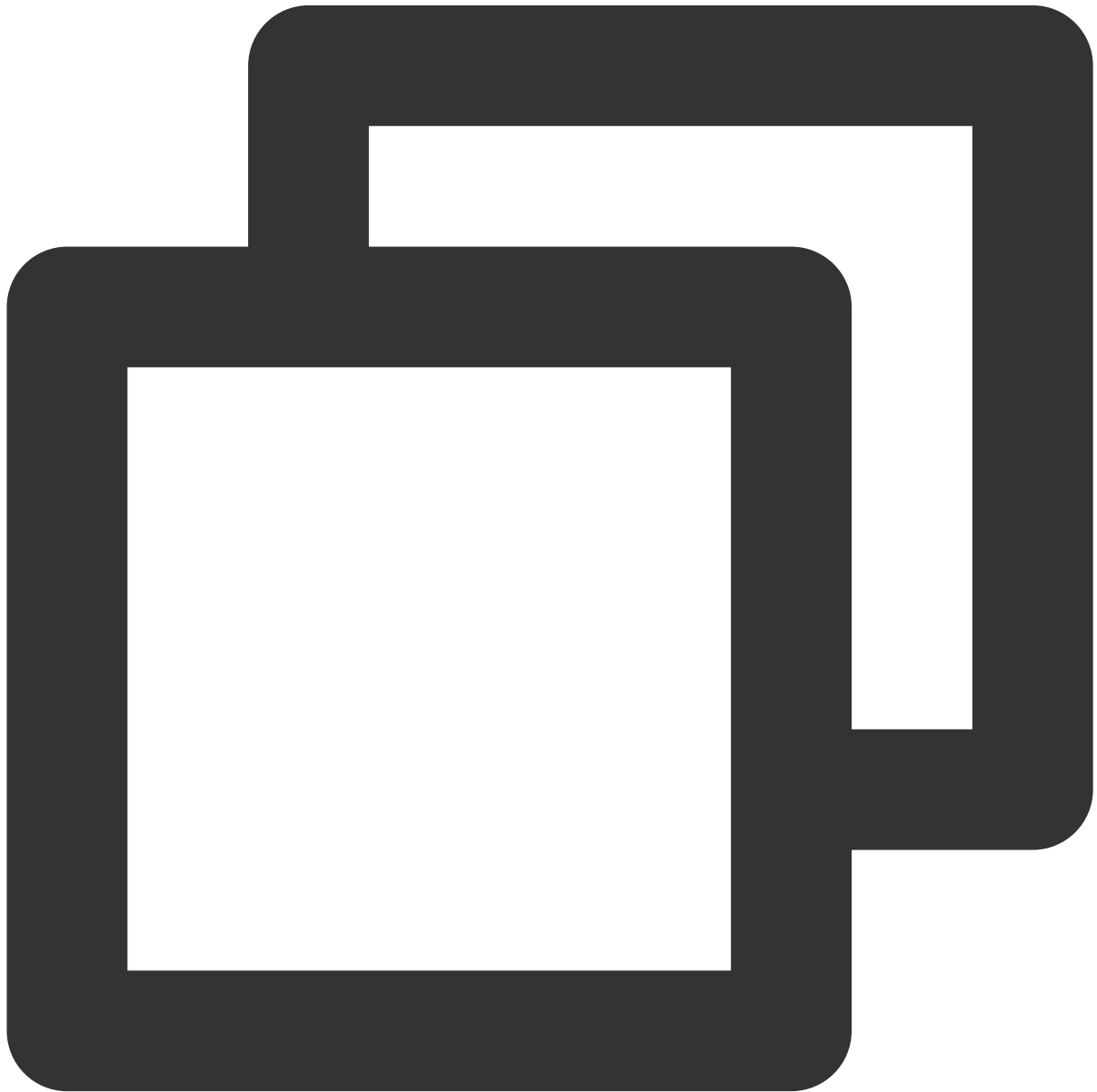
The experimental interface `enableBlackStream` needs to be called after entering the room;

On Android, the value type of the `enable` parameter is Boolean, and on iOS it is Integer;

The receiving end needs to call `startRemoteView(userId, null)` after receiving

`onUserVideoAvailable(userId, true)` .

## 2. Sending Song Progress through SEI Message



```
TXAudioMusicProgressBlock progressBlock = ^(NSInteger progressMs, NSInteger durationMs) {
    // current ntp time
    NSInteger ntpTime = [TXLiveBase getNetworkTimestamp];
    // Notify the song progress, users will scroll the lyrics here.
    NSDictionary *progressMsg = @{
        @"bgmProgressTime": @(progressMs),
        @"ntpTime": @(ntpTime),
        @"musicId": @(musicId),
        @"duration": @(durationMs),
    };
    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:progressMsg options:0 error:nil];
};
```



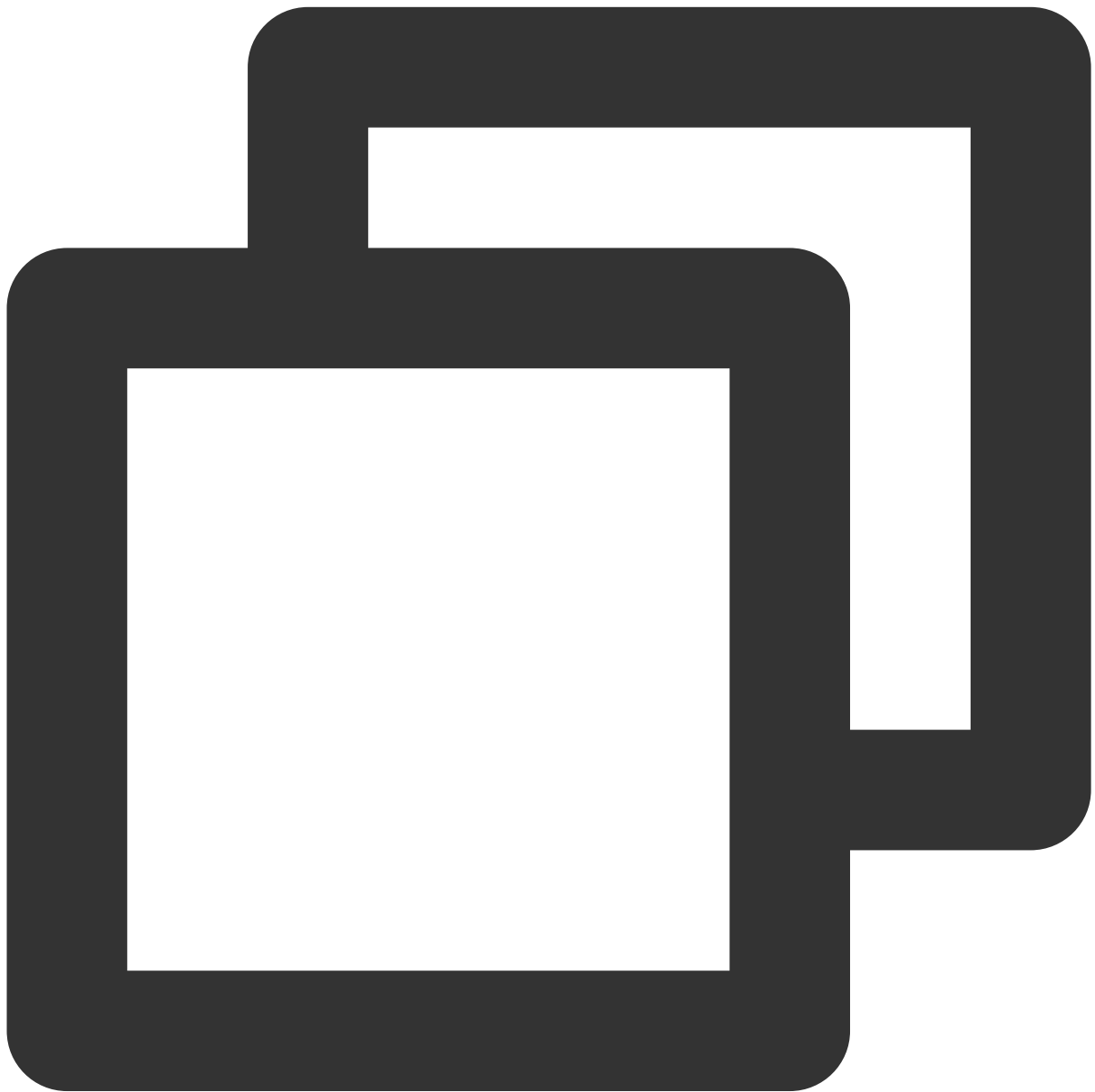
```
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8S  
[trtcCloud sendSEIMsg:[jsonString dataUsingEncoding:NSUTF8StringEncoding] repea  
};
```

**Note:**

The frequency at which the lead singer sends SEI messages is determined by the frequency of background music playback event callbacks, which is usually 200ms;

The reason for **not directly using CMD messages to send song progress** is that the signaling transmitted through the SEI channel can be transmitted with the video frame to the live CDN, which has better compatibility for viewers who pull the CDN stream.

### 3. Synchronization of Local and Remote Lyrics



```
// local lyrics synchronization
TXAudioMusicProgressBlock progressBlock = ^(NSInteger progressMs, NSInteger duration)
{
    ...
    // TODO Update the logic of the lyrics control.
    // Determine whether it is necessary to seek the lyrics control
    // based on the latest progress and the error of the local lyrics progress.
    ...
};

// remote lyrics synchronization.
- (void)onRecvSEIMsg:(NSString *)userId message:(NSData *)message {
```

```
NSError *err = nil;
NSDictionary *dic = [NSJSONSerialization JSONObjectWithData:message options:NSJ
if (err || ![dic isKindOfClass:[NSDictionary class]]) {
    // Parsing error.
    return;
}
NSInteger bgmProgressTime = [[dic objectForKey:@"bgmProgressTime"] integerValue]
NSInteger ntpTime = [[dic objectForKey:@"ntpTime"] integerValue];
int32_t musicId = [[dic objectForKey:@"musicId"] intValue];
NSInteger duration = [[dic objectForKey:@"duration"] integerValue];
...
// TODO Update the logic of the lyrics control.
// Determine whether it is necessary to seek the lyrics control
// based on the received latest progress and the error of the local lyrics prog
...
}
```

### Note

If reusing the TUIKaraoke component's lyric control, please refer to the code logic in the [TUIKaraoke TRTCLyricView](#) section to synchronize the progress of the lyric control.

# Android

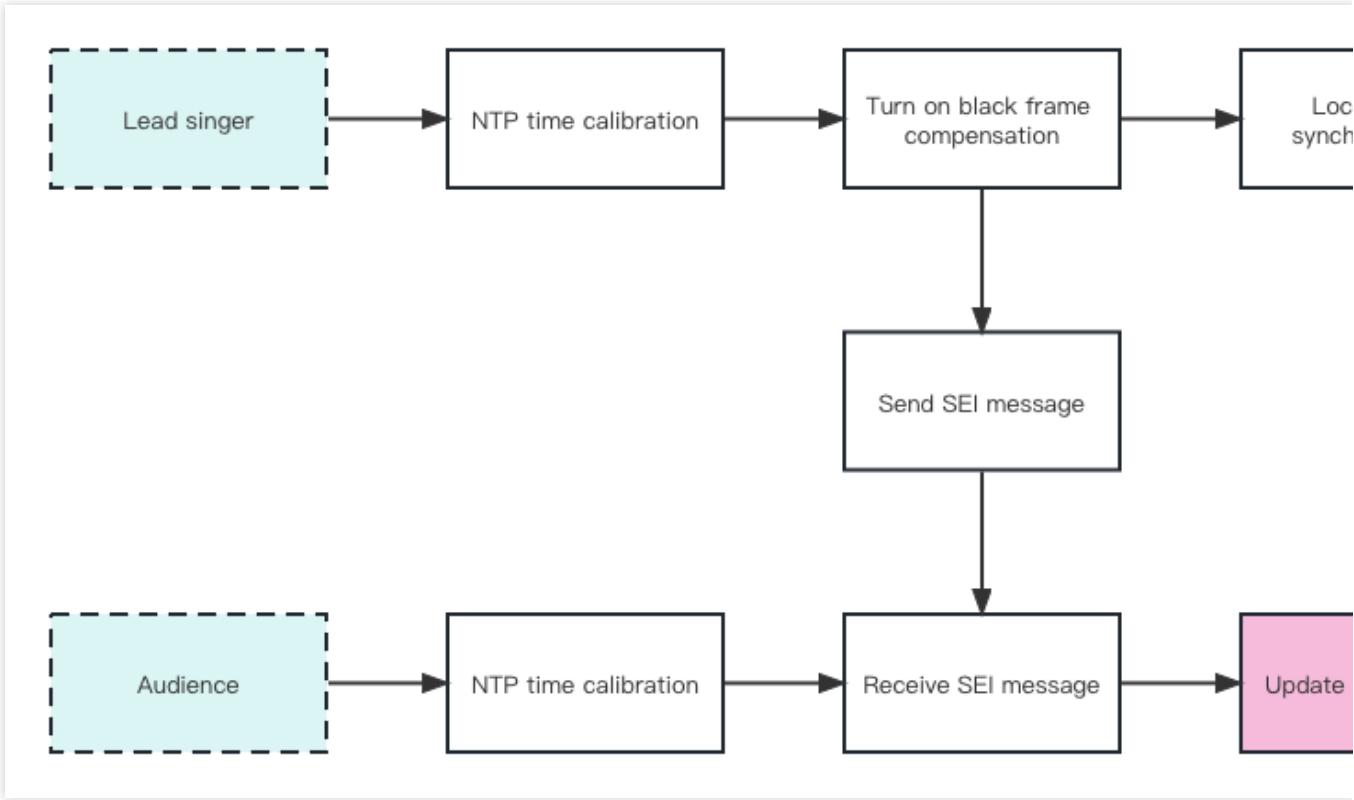
Last updated : 2023-12-28 21:32:43

## Implementation process

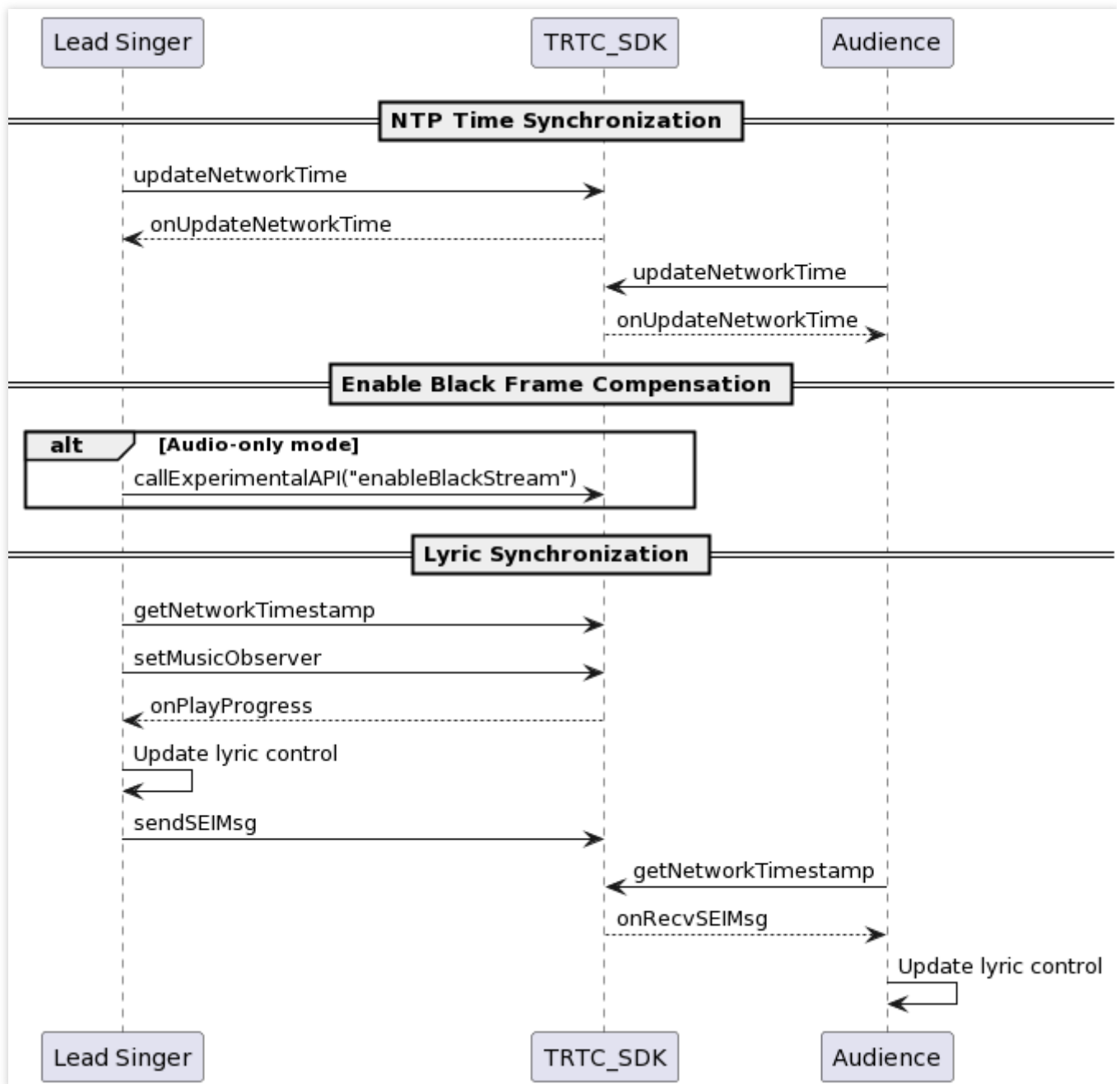
In the lyrics synchronization scheme, the actions of three different roles are as follows:

| Lead Singer                                                                                                                          | Chorus                                                                       | Audience                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|---------------------------------------------------------------------|
| NTP time calibration<br>Turn on black frame compensation<br>Send SEI message<br>Local lyrics synchronization<br>Update lyrics widget | NTP time calibration<br>Local lyrics synchronization<br>Update lyrics widget | NTP time calibration<br>Receive SEI message<br>Update lyrics widget |

The lead singer and chorus update the lyrics progress locally according to the synchronized song playback progress; the audience needs to receive the SEI message sent by the lead singer, which contains the latest lyrics progress, to update the local lyrics progress.



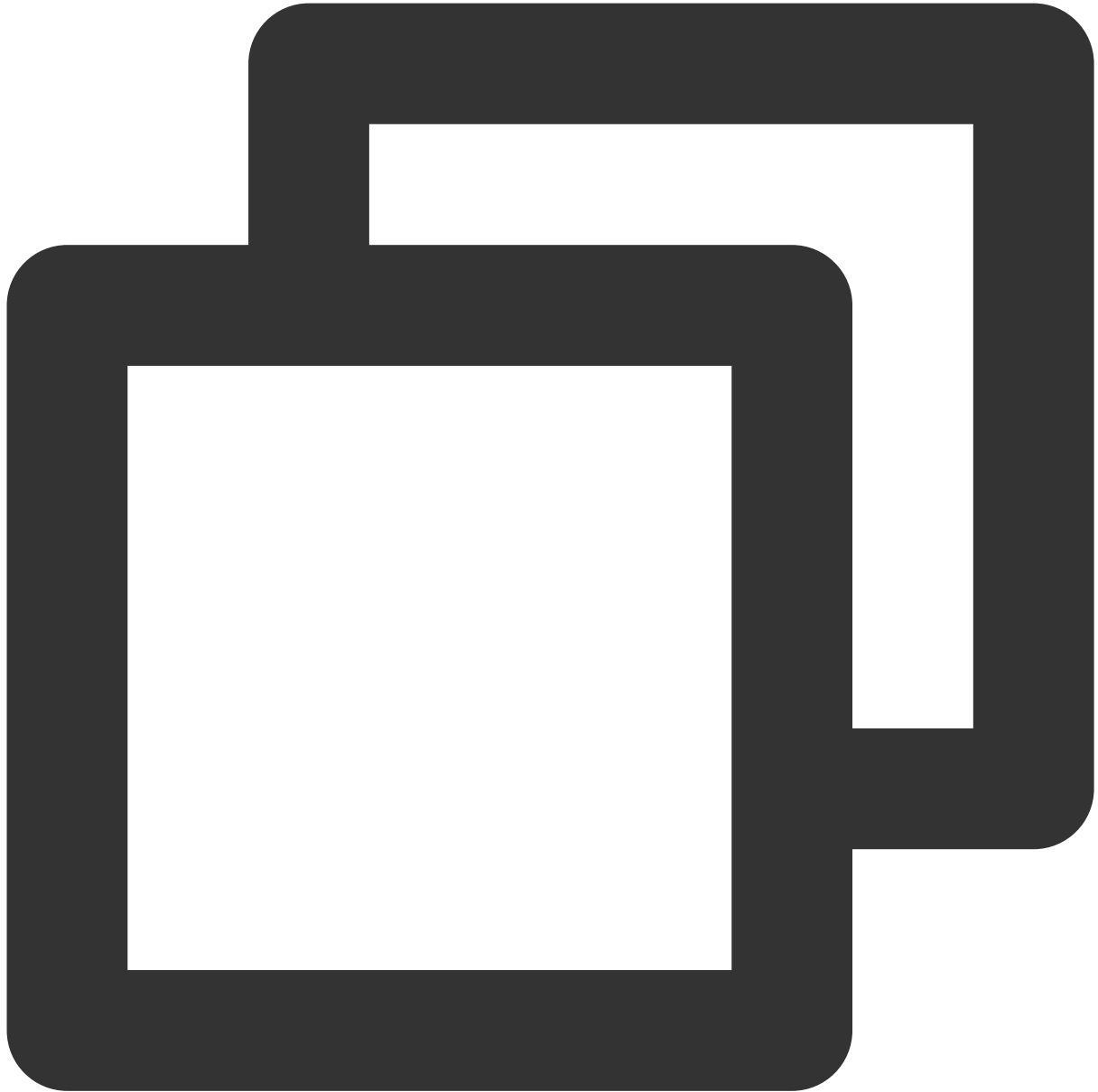
## Timing diagram



The lyrics synchronization timing can be mainly divided into three parts: NTP time calibration, turning on black frame compensation, and local and remote lyrics synchronization. The code implementation of NTP time calibration has been given in the [song synchronization](#), and the specific code implementation for the latter two parts will be provided below.

# Key code implementation

## 1. Turn on black frame compensation



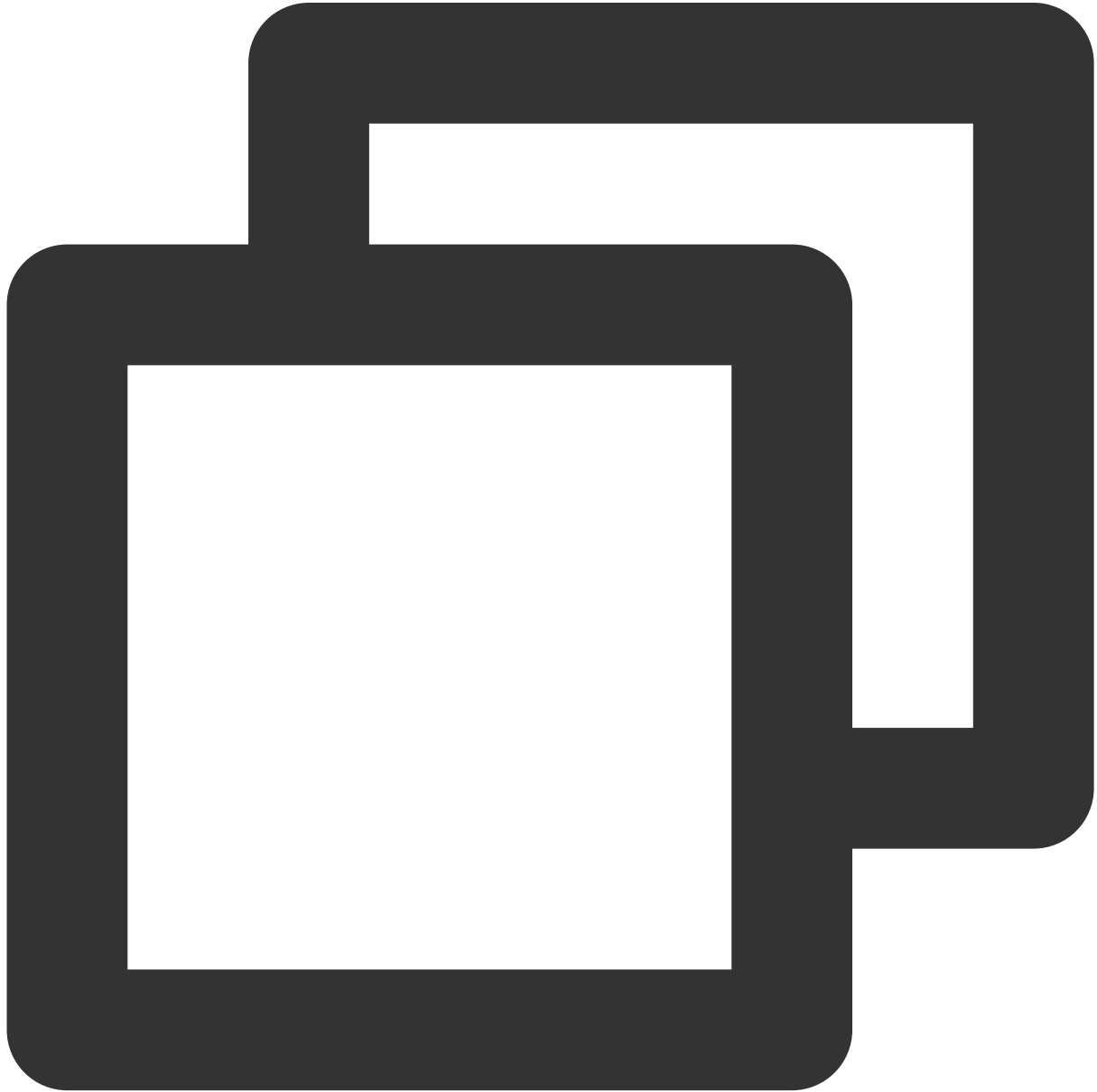
```
// In pure audio mode, the main instance (vocal instance) needs to turn on black fr  
mTRTCCloud.callExperimentalAPI("{\"api\":\"enableBlackStream\",\"params\": {\\"
```

**Note:**

The experimental interface `enableBlackStream` needs to be called after entering the room.  
On Android, the value type of the enable parameter is boolean, and on iOS, it is integer.

The receiver needs to call `startRemoteView(userId, null)` when `onUserVideoAvailable(userId, true)` is received.

## 2. Send song progress through SEI message



```
mAudioEffectManager.setMusicObserver(mCurPlayMusicId, new TXAudioEffectManager.TXMu
@Override
public void onPlayProgress(int id, long curPtsMS, long durationMS) {
    JSONObject jsonObject = new JSONObject();
    // Current NTP time
    long ntpTime = TXLiveBase.getNetworkTimestamp();
    jsonObject.put("bgmProgressTime", curTime);
}
```

```
        jsonObject.put("ntpTime", ntpTime);
        jsonObject.put("musicId", musicId);
        jsonObject.put("duration", duration);
        jsonObject.toString().getBytes();
        mTRTCCloud.sendSEIMsg(jsonObject.toString().getBytes(), 1);
    }
}
```

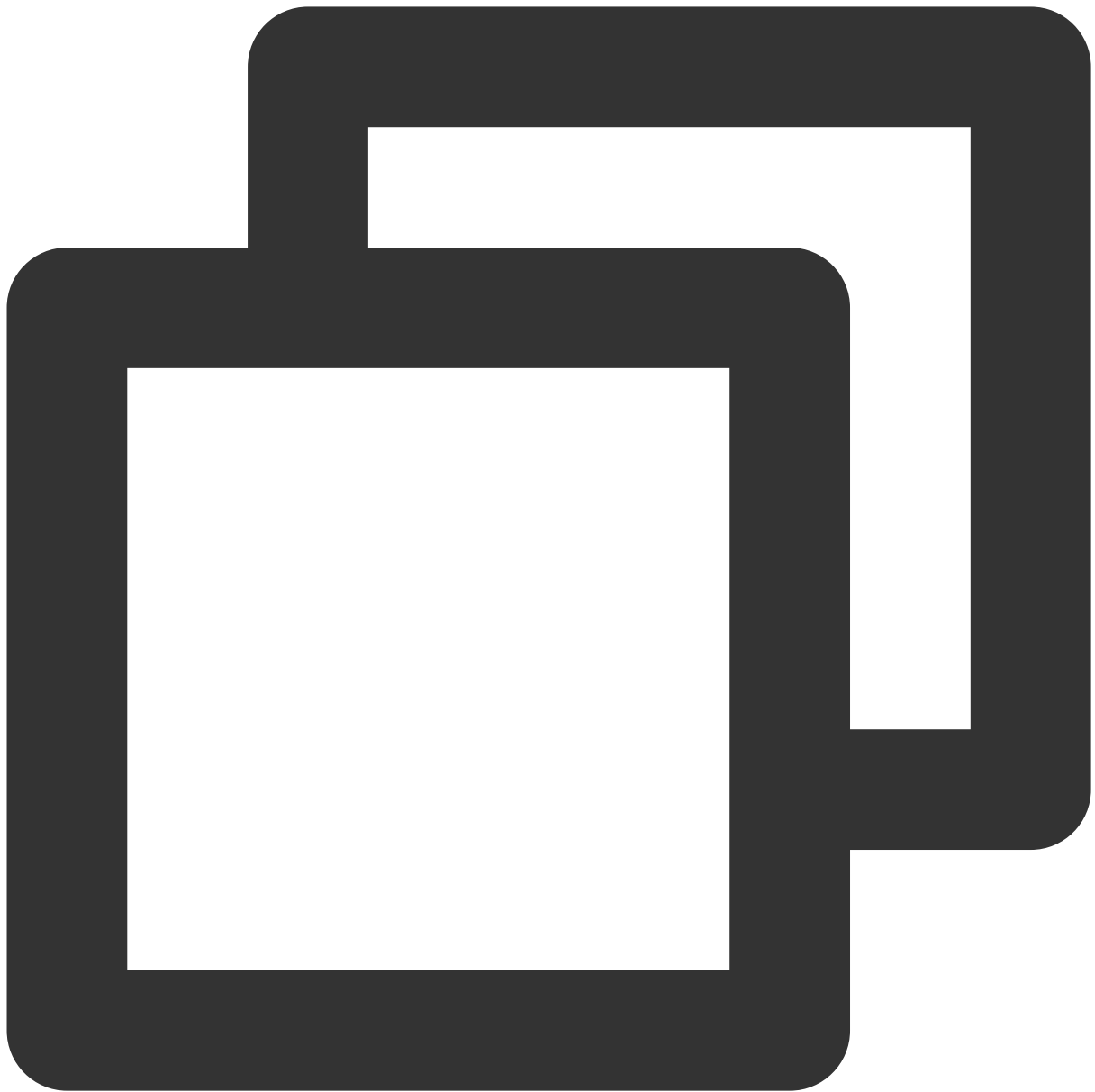
**Note:**

The frequency of the lead singer sending SEI messages is determined by the frequency of background music playback event callbacks, usually 200ms;

The reason for **not directly using CMD messages to send song progress**: The signaling transmitted by the SEI channel can be transmitted to the live CDN along with the video frames, providing better compatibility for the audience pulling the CDN stream.

### 3. Local and remote lyrics synchronization





```
// Local lyrics synchronization
mAudioEffectManager.setMusicObserver(mCurPlayMusicId, new TXAudioEffectManager.TXMu
    @Override
    public void onPlayProgress(int id, long curPtsMS, long durationMS) {
        ...
        // TODO Update lyrics widget logic:
        // Determine whether to seek the lyrics widget based on the latest progress
        ...
    }
}
```

```
// Remote lyrics synchronization
@Override
public void onRecvSEIMsg(String userId, byte[] data) {
    String result = new String(data);
    JSONObject jsonObject = new JSONObject(result);
    long bgmProgressTime = jsonObject.getLong("bgmProgressTime");
    long ntpTime = jsonObject.getLong("ntpTime");
    String musicId = jsonObject.getString("musicId");
    long duration = jsonObject.getLong("duration");
    ...
    // TODO Update lyrics widget logic:
    // If you reuse the TUIKaraoke component's lyrics widget,
    //please refer to the code logic of the TUIKaraoke LyricsView section to synchr
    ...
}
```

**Note:**

If you reuse the TUIKaraoke component's lyrics widget, please refer to the code logic of the [TUIKaraoke LyricsView](#) section to synchronize the lyrics widget progress.

# Vocal Synchronization

## iOS

Last updated : 2023-09-26 16:52:53

## Introduction to Synchronization of Vocals and Songs

Due to the existence of certain gaps between the jitter buffer for local voice collection, the jitter buffer for song playback mixing, and the sound reaching the singer's ears, when the singer sings completely facing the lyrics and BGM playback, remote audiences will feel that there is a certain delay between the playback of BGM, vocals, and lyrics. The chorus solution in the TRTC SDK uses low-latency AAudio collection internally. You only need to enable chorus mode and low-latency mode after entering the room.

## Specific Code Implementation

### Enable Chorus Mode



```
// Enable tutti mode for the main instance (vocal instance) by reducing the buffer
// and enabling audio redundancy protection.
NSDictionary *jsonDic = @{
    @"api": @"enableChorus",
    @"params": @{
        @"enable": @(YES),
        @"audioSource": @(0)
    }
};

NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
```

```
[trtcCloud callExperimentalAPI:jsonString];
// Enable tutti mode for the sub-instance (accompaniment instance) by reducing the
// and enabling audio redundancy protection.
NSDictionary *jsonDic = @{
    @"api": @"enableChorus",
    @"params": @{
        @"enable": @(YES),
        @"audioSource": @(1)
    }
};

NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[subCloud callExperimentalAPI:jsonString];
```

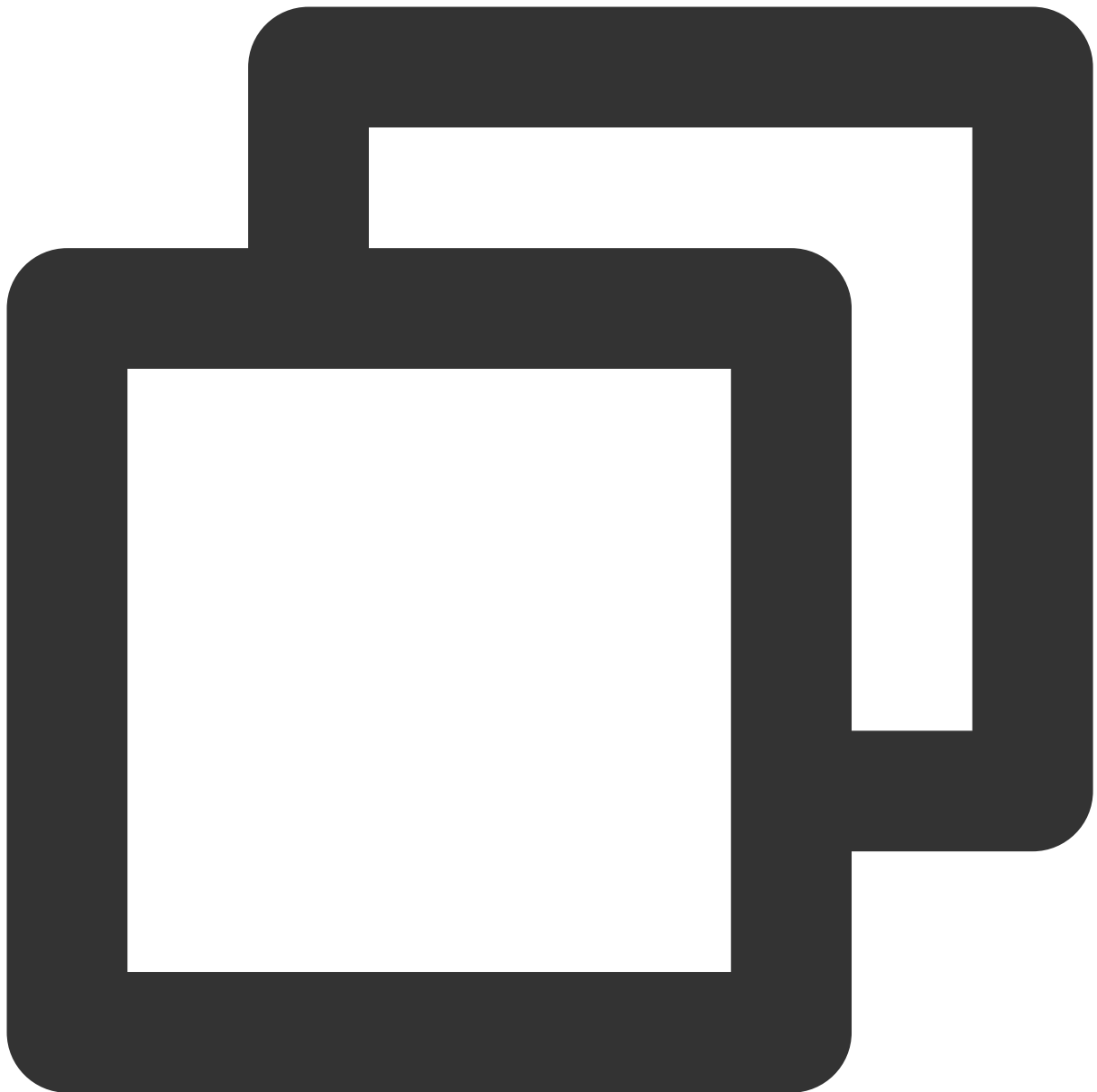
**Note :**

The parameter settings for enabling chorus mode through the experimental interface `enableChorus` are as follows:

audioSource: 0 (vocals)

audioSource: 1 (accompaniment)

**Enable Low-Latency Mode (High-Performance Audio AAudio)**



```
// Enable high-performance audio AAudio for the main instance (vocal instance).
NSDictionary *jsonDic = @{
    @"api": @"setLowLatencyModeEnabled",
    @"params": @{
        @"enable": @(1)
    }
};

NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[trtcCloud callExperimentalAPI:jsonString];
// Enable high-performance audio AAudio for the sub-instance (accompaniment instanc
```

```
NSDictionary *jsonDic = @{
    @"api": @"setLowLatencyModeEnabled",
    @"params": @{
        @"enable": @(1)
    }
};

NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[subCloud callExperimentalAPI:jsonString];
```

# Android

Last updated : 2023-09-26 16:53:14

## Introduction to vocal and song synchronization

Due to the jitter buffer of local vocal collection, the jitter buffer of song playback mixing, and the certain GAP between sound playback to the human ear and singing, when the singer sings along with the lyrics and BGM, the remote audience feels that there is a certain delay in the BGM playback, vocals, and lyrics. The chorus scheme uses low-latency AAudio collection inside the TRTC SDK. Specifically, you only need to enable the chorus mode and low-latency mode after entering the room.

## Specific code implementation

### Enable chorus mode





```
// The main instance (vocal instance) enables chorus mode (reducing buffer interval  
mTRTCCloud.callExperimentalAPI("{\\"api\\":\\"enableChorus\\",\\"params\\":{\\"enab  
// The sub-instance (accompaniment instance) enables chorus mode (reducing buffer i  
subCloud.callExperimentalAPI("{\\"api\\":\\"enableChorus\\",\\"params\\":{\\"enable
```

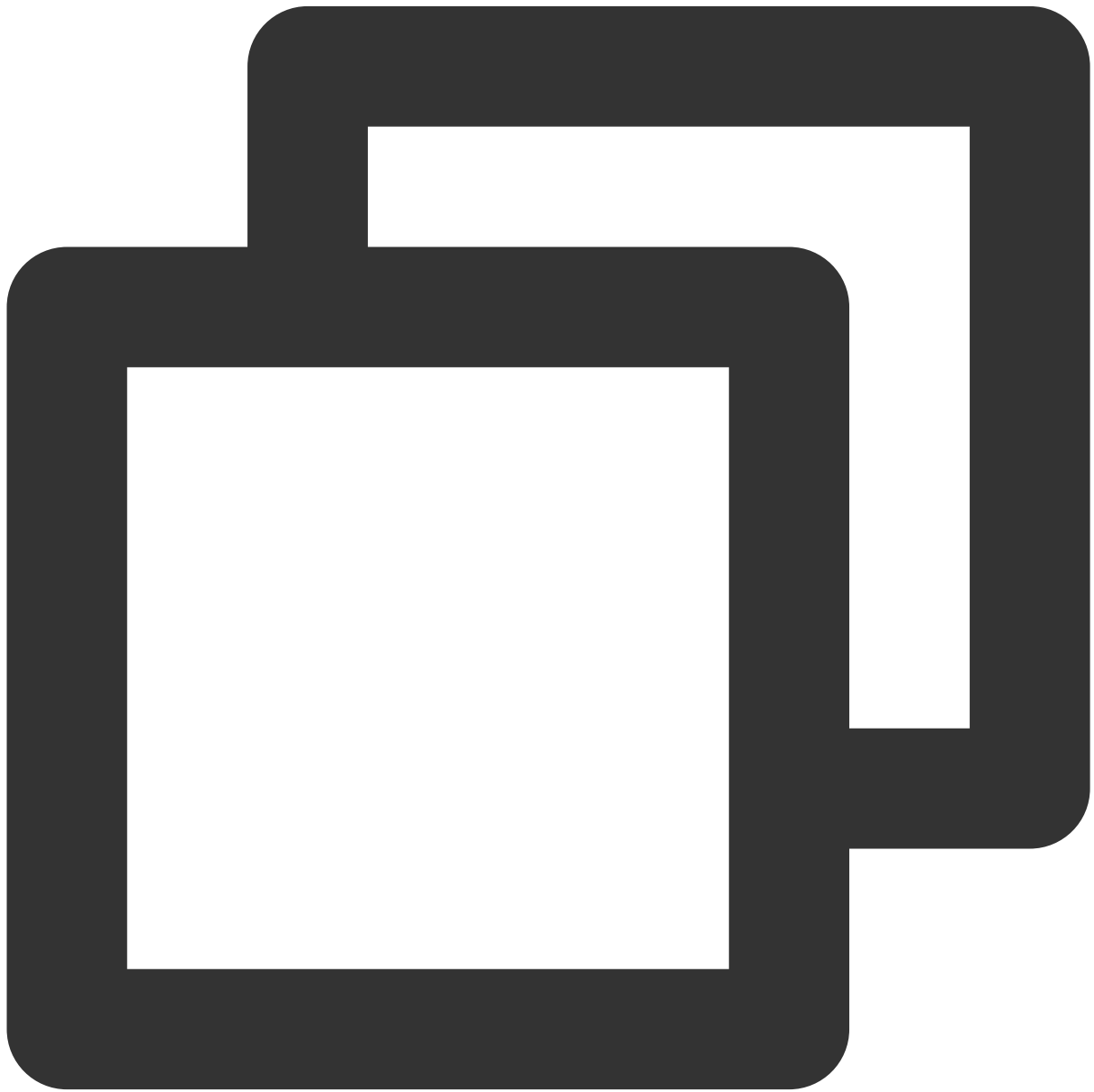
**Note:**

Parameter settings for the experimental interface enableChorus to enable chorus mode:

audioSource : 0(vocals).

audioSource : 1 (accompaniment).

## Enable low-latency mode (high-performance audio AAudio)



```
// The main instance (vocal instance) enables high-performance audio AAudio
mTRTCCloud.callExperimentalAPI("{\\\"api\\\":\\\"setLowLatencyModeEnabled\\\",\\\"params\\\"}");
// The sub-instance (accompaniment instance) enables high-performance audio AAudio
subCloud.callExperimentalAPI("{\\\"api\\\":\\\"setLowLatencyModeEnabled\\\",\\\"params\\\"}");
```

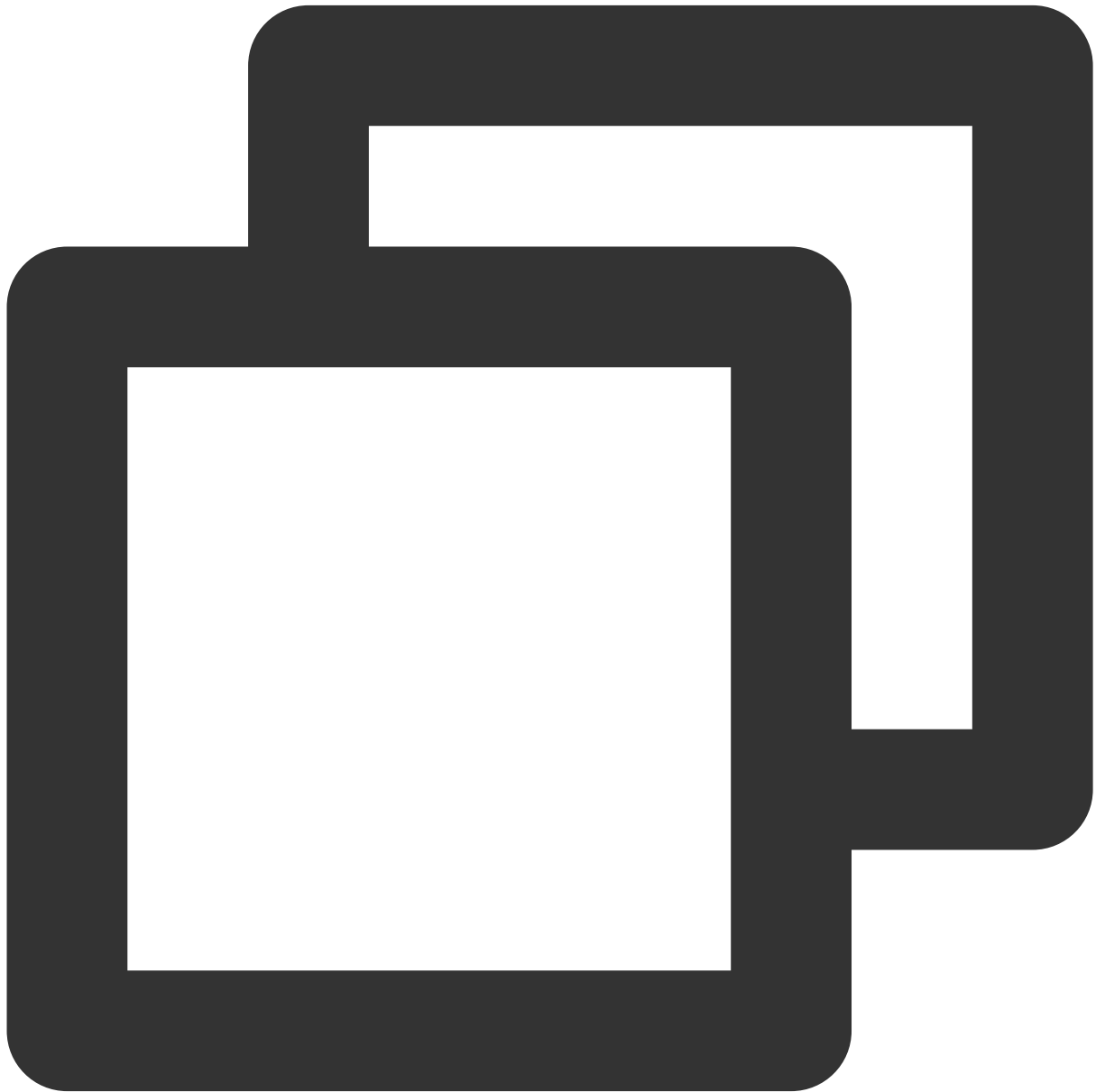
# Mixing Stream Solution

## iOS

Last updated : 2023-09-26 16:53:38

### Specific code implementation

#### 1. Create main and sub-instances



```
// Create TRTCCloud main instance (vocal instance)
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
// Create TRTCCloud sub-instance (accompaniment instance)
TRTCCloud *subCloud = [trtcCloud createSubCloud];
```

**Note :**

In the real-time chorus scheme, the lead singer needs to create the main instance-vocal instance and the sub-instance-accompaniment instance separately for uploading vocals and accompaniment music.

**2. Vocal instance enters the room and pushes the stream**



```
TRTCParams *params = [[TRTCParams alloc] init];
params.sdkAppId = sdkAppId;
params.userId = userId;
params.userSig = userSign;
params.role = TRTCRoleAnchor;
params.roomId = roomIdIntValue;
[trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
// Turn on audio uplink and set audio quality
[trtcCloud startLocalAudio:TRTCAudioQualityMusic];
// Set media type
[trtcCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
```

```
// Mute remote accompaniment music  
[trtcCloud muteRemoteAudio:remoteAudioId mute:YES];
```

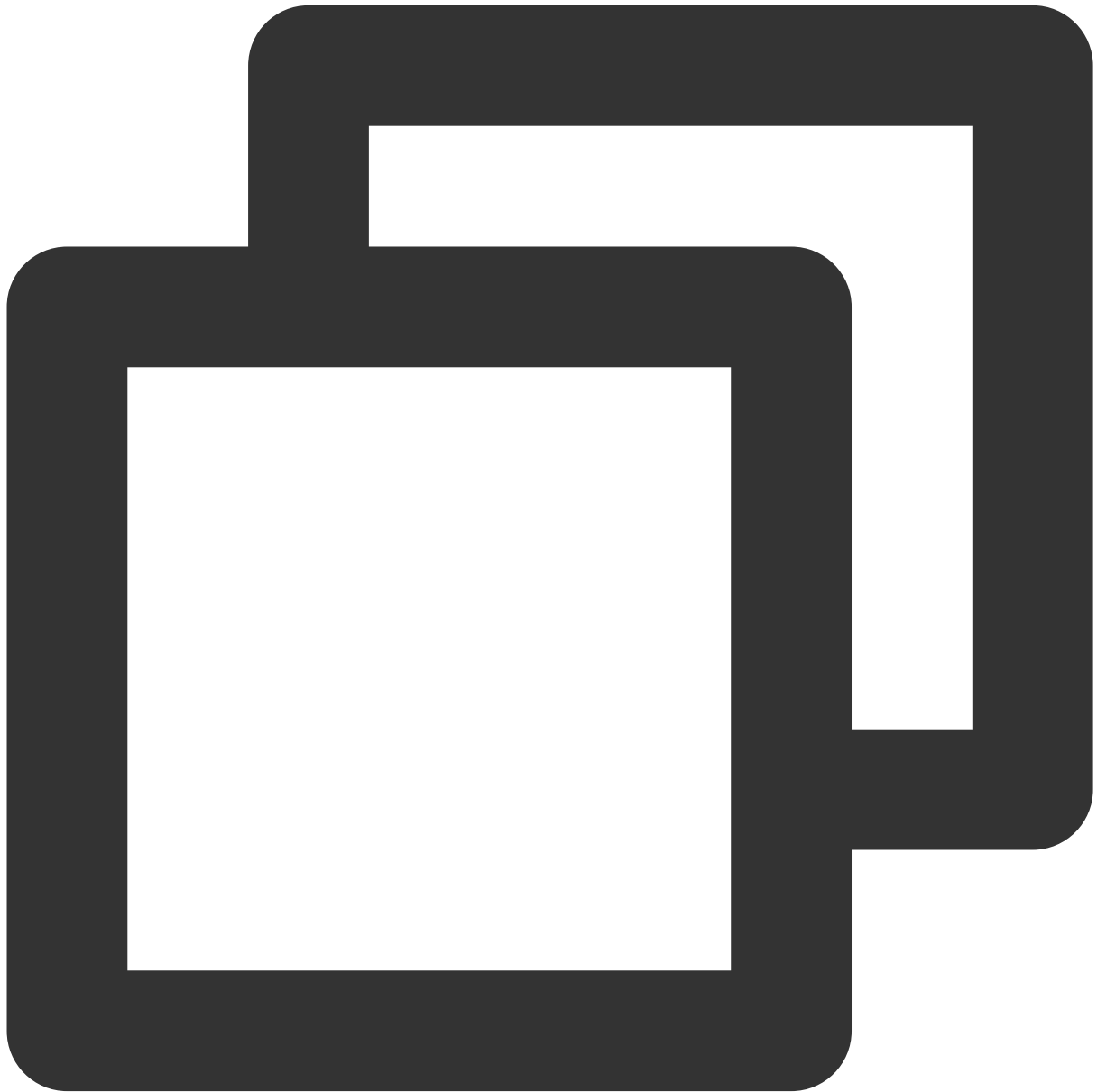
**Note :**

In pure RTC audio scenarios, it is recommended to use VOICE\_CHATROOM for entering the room.

If there is a need for video or CDN forwarding, the room entry scenario must use LIVE, as VOICE\_CHATROOM will add pure audio parameters during forwarding, causing SEI messages to fail to pass through.

The lead singer/chorus needs to muteRemoteAudio(true) to unsubscribe from the audio stream uploaded by the accompaniment instance, otherwise, the local and remote accompaniment music will be played repeatedly.

**3. Accompaniment example: Joining room and pushing stream.**



```
TRTCParams *bgmParams = [[TRTCParams alloc] init];
bgmParams.sdkAppId = sdkAppId;
bgmParams.userId = [NSString stringWithFormat:@"%%%%",userId,@"_bgm"];
bgmParams.userSig = bgmUserSign;
bgmParams.role = TRTCRoleAnchor;
bgmParams.roomId = roomIdIntValue;
[subCloud enterRoom:bgmParams appScene:TRTCAppSceneLIVE];
// Set media type
[subCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
// Enable preloading
NSDictionary *jsonDict = @{
```

```

        @"api": @"preloadMusic",
        @"params": @{
            @"musicId": @(self.currentPlayMusicID),
            @"path": path,
            @"startTimeMS": @(startMs),
        }

    };

    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict options:0 error:
    NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
    [subCloud callExperimentalAPI:jsonString];
    // Play accompaniment music and push the stream (play at the agreed time)
    TXAudioMusicParam *musicParam = [[TXAudioMusicParam alloc] init];
    musicParam.ID = musicID;
    musicParam.path = url;
    musicParam.loopCount = 0;
    musicParam.publish = YES;
    // Send accompaniment music to the remote end
    param.publish = YES;
    [[subCloud getAudioEffectManager] startPlayMusic:musicParam onStart:startBlock onPr

```

**Note :**

Pay attention to distinguish between the userId of the main instance and the sub-instance, ensuring that they are not duplicated and easy to identify;

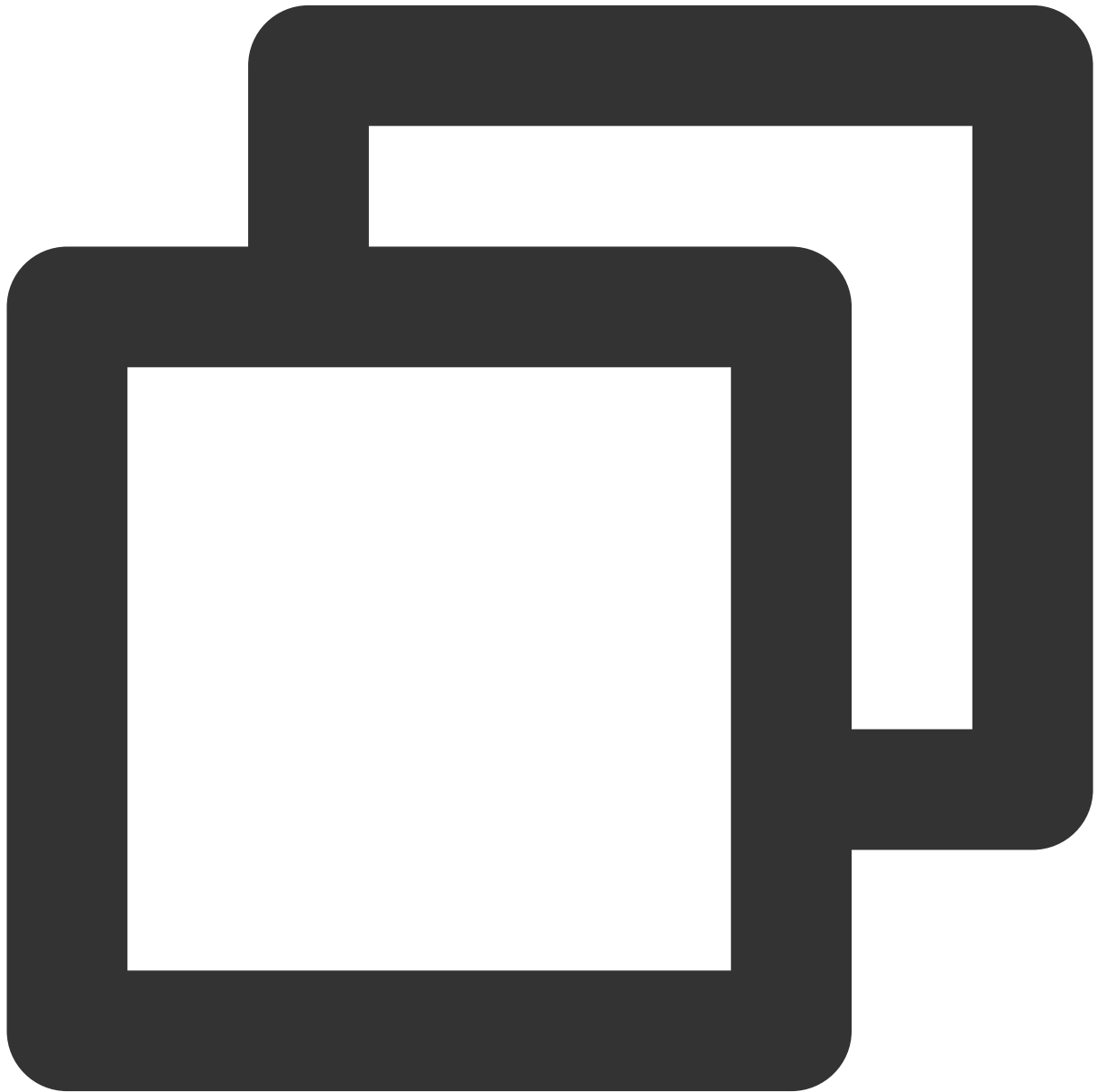
Accompaniment instance background music parameter musicParam seettings:

publish : YES (while the music is playing locally, remote users can also hear the music)

publish : NO (default value, the music can only be heard locally, remote users cannot hear it)

**4. Initiating mixed stream transcoding and pushing back.**





```
// Create a TRTCPublishTarget object
TRTCPublishTarget *publishTarget = [[TRTCPublishTarget alloc] init];
// Push back to the room after mixing, if publishing to CDN, fill in TRTCPublishMix
publishTarget.mode = TRTCPublishMixStreamToRoom;
// The userid of the mixing robot, which cannot be duplicated with other users' use
publishTarget.mixStreamIdentity = [NSString stringWithFormat:@"%s%@",userId,@"_mix"]

// Set the encoding parameters of the transcoded audio stream
TRTCStreamEncoderParam *streamEncoderParam = [[TRTCStreamEncoderParam alloc] init];
streamEncoderParam.videoEncodedFPS = 15;
streamEncoderParam.videoEncodedGOP = 3;
```

```
streamEncoderParam.videoEncodedKbps = 30;
streamEncoderParam.audioEncodedSampleRate = 48000;
streamEncoderParam.audioEncodedChannelNum = 2;
streamEncoderParam.audioEncodedKbps = 64;
streamEncoderParam.audioEncodedCodecType = 2;

// Set audio mixing parameters
TRTCStreamMixingConfig *streamMixingConfig = [[TRTCStreamMixingConfig alloc] init];
// Support filling in empty values, which will automatically mix the audio of all h
streamMixingConfig.audioMixUserList = @[];

// Initiate mixed stream transcoding and pushing request
[trtcCloud startPublishMediaStream:publishTarget encoderParam:streamEncoderParam mi
```

**Note:**

It is recommended to prioritize the lead singer to initiate mixed stream transcoding and pushing through the mixing robot to the backend, mixing the accompaniment music and all vocal streams and pushing them back to the TRTC room, or pushing them to the live CDN.

In automatic subscription mode, the hosts participating in the mixed stream transcoding will pull each other's single stream by default and not receive the mixed stream pushed back to the room; the audience will automatically pull the mixed stream pushed back to the room and no longer receive the single stream.

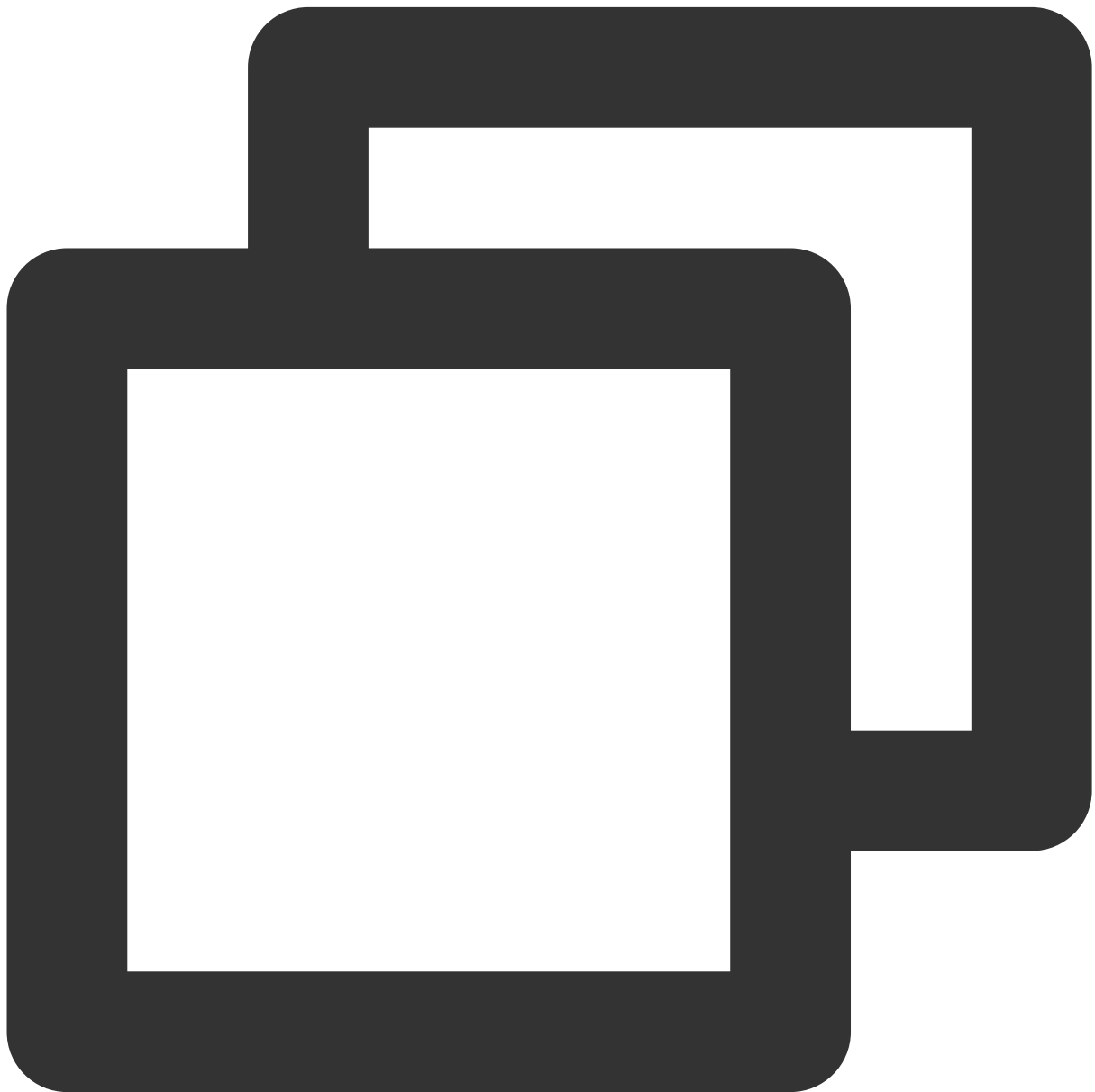
The mixed stream transcoding and pushing method `startPublishMediaStream` used here adopts a brand new backend architecture. The old version of the application needs to provide the `SdkAppId` to apply for an upgrade before it can be used.

# Android

Last updated : 2023-09-26 16:54:10

## Specific code implementation

### 1. Create main and sub-instances

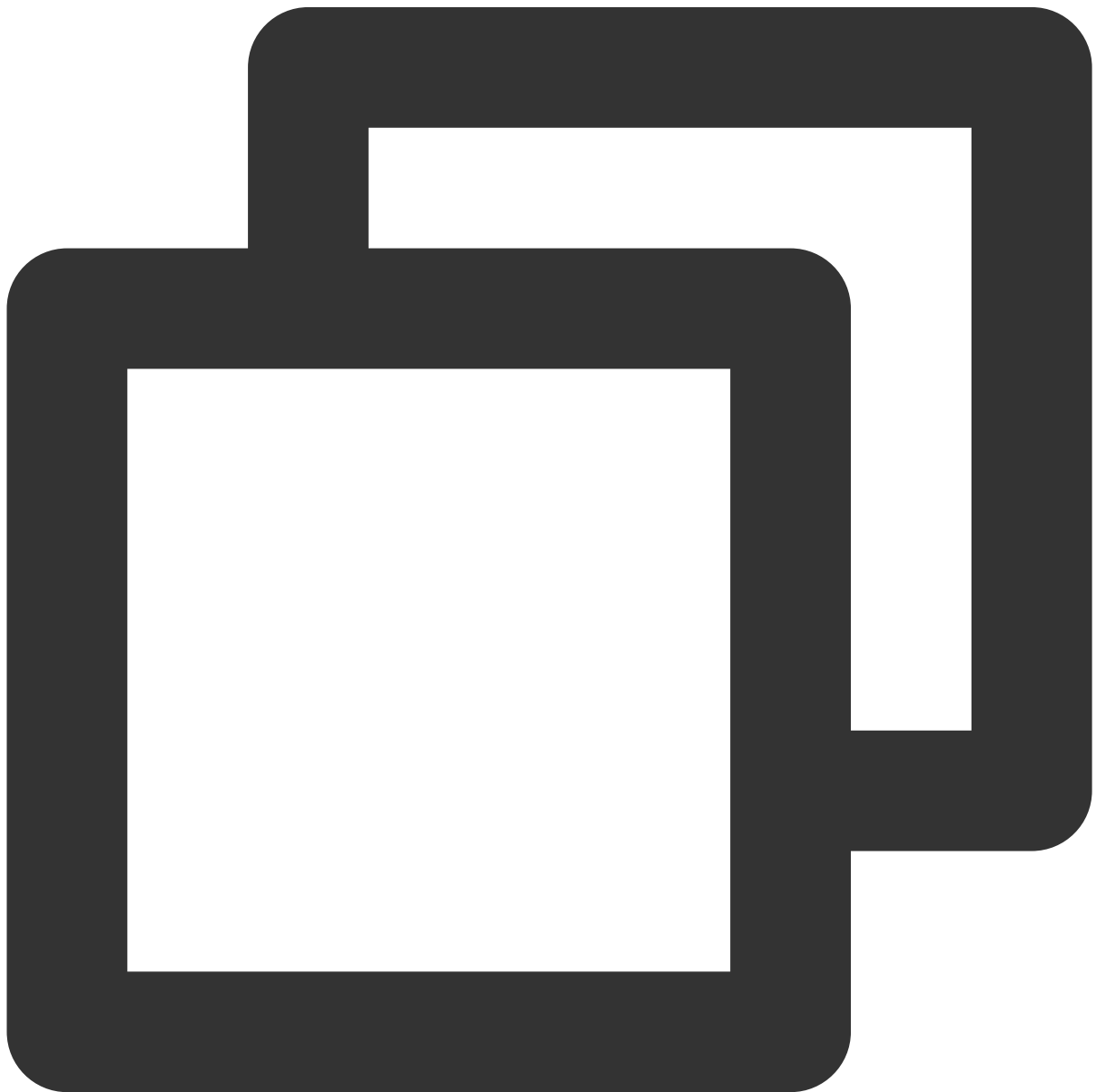


```
// Create TRTCCloud main instance (vocal instance)
```

```
TRTCCloud mTRTCCloud = TRTCCloud.sharedInstance(getApplicationContext());  
// Create TRTCCloud sub-instance (accompaniment instance)  
TRTCCloud subCloud = mTRTCCloud.createSubCloud();
```

**Note:**

In the real-time chorus scheme, the lead singer needs to create the main instance-vocal instance and the sub-instance-accompaniment instance separately for uploading vocals and accompaniment music.

**2. Vocal instance enters the room and pushes the stream**

```
TRTCCloudDef.TRTCPParams params = new TRTCCloudDef.TRTCPParams();
```

```
params.sdkAppId = sdkAppId;
params.userId = mUserId;
params.userSig = userSig;
params.role = TRTCCloudDef.ORTCRoleAnchor;
params.roomId = mRoomId;
mTRTCCloud.enterRoom(params, TRTCCloudDef.ORTC_APP_SCENE_LIVE);
// Turn on audio uplink and set audio quality
mTRTCCloud.startLocalAudio(TRTCCloudDef.ORTC_AUDIO_QUALITY_MUSIC);
// Set media type
mTRTCCloud.setSystemVolumeType(TRTCCloudDef.ORTCSystemVolumeTypeMedia);
// Mute remote accompaniment music
mTRTCCloud.muteRemoteAudio(mUserId + "_bgm", true);
```

**Notice :**

In pure RTC audio scenarios, it is recommended to use VOICE\_CHATROOM for entering the room.

If there is a need for video or CDN forwarding, the room entry scenario must use LIVE, as VOICE\_CHATROOM will add pure audio parameters during forwarding, causing SEI messages to fail to pass through.

The lead singer/chorus needs to muteRemoteAudio(true) to unsubscribe from the audio stream uploaded by the accompaniment instance, otherwise, the local and remote accompaniment music will be played repeatedly.

### 3. Accompaniment instance enters the room and pushes the stream



```
TRTCCloudDef.TRTCParams bgmParams = new TRTCCloudDef.TRTCParams();
bgmParams.sdkAppId = sdkAppId;
bgmParams.userId = mUserId + "_bgm";
bgmParams.userSig = userSig;
bgmParams.role = TRTCCloudDef.TRTCRoleAnchor;
bgmParams.roomId = mRoomId;
subCloud.enterRoom(bgmParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
// Set media type
subCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);

// Enable preloading
```

```
subCloud.callExperimentalAPI("{\"api\":\"preloadMusic\",\"params\":{\"music\n// Play accompaniment music and push the stream (play at the agreed time)\nTXAudioEffectManager.AudioMusicParam param = new TXAudioEffectManager.AudioMusicPar\n// Send accompaniment music to the remote end\nparam.publish = true;\nsubCloud.getAudioEffectManager().startPlayMusic(param);
```

**Note:**

Pay attention to distinguish between the `userId` of the main instance and the sub-instance, ensuring that they are not duplicated and easy to identify;

Accompaniment instance background music parameter `AudioMusicParam` settings:

`publish` : `true` (while the music is playing locally, remote users can also hear the music)

`publish` : `false` (default value, the music can only be heard locally, remote users cannot hear it)

#### 4. Initiate mixed stream transcoding and pushing



```
// Create a TRTCPublishTarget object
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
// Push back to the room after mixing, if publishing to CDN, fill in TRTC_PublishMi
target.mode = TRTCCloudDef.TRTC_PublishMixStream_ToRoom;
target.mixStreamIdentity.intRoomId = Integer.parseInt(mRoomId);
// The userid of the mixing robot, which cannot be duplicated with other users' use
target.mixStreamIdentity.userId = mUserId + "_mix";

// Set the encoding parameters of the transcoded audio stream
TRTCCloudDef.TRTCStreamEncoderParam trtcStreamEncoderParam = new TRTCCloudDef.TRTCS
trtcStreamEncoderParam.audioEncodedChannelNum = 2;
```



```
trtcStreamEncoderParam.audioEncodedKbps = 64;
trtcStreamEncoderParam.audioEncodedCodecType = 2;
trtcStreamEncoderParam.audioEncodedSampleRate = 48000;

// Set audio mixing parameters
TRTCCloudDef.TRTCStreamMixingConfig trtcStreamMixingConfig = new TRTCCloudDef.TRTCS
// Support filling in empty values, which will automatically mix the audio of all h
trtcStreamMixingConfig.audioMixUserList = null;

// Initiate mixed stream transcoding and pushing request
mTRTCCloud.startPublishMediaStream(target, trtcStreamEncoderParam, trtcStreamMixing
```

**Note :**

It is recommended to prioritize the lead singer to initiate mixed stream transcoding and pushing through the mixing robot to the backend, mixing the accompaniment music and all vocal streams and pushing them back to the TRTC room, or pushing them to the live CDN.

In automatic subscription mode, the hosts participating in the mixed stream transcoding will pull each other's single stream by default and not receive the mixed stream pushed back to the room; the audience will automatically pull the mixed stream pushed back to the room and no longer receive the single stream.

The mixed stream transcoding and pushing method `startPublishMediaStream` used here adopts a brand new backend architecture. The old version of the application needs to provide the `SdkAppId` to apply for an upgrade before it can be used.

# TUIKaraoke APIs

## TRTCKaraoke (iOS)

Last updated : 2023-09-25 10:59:08

`TRTCKaraokeRoom` is based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat. With TRTCKaraoke:

A user can create a karaoke room and become a speaker, or enter a karaoke room as a listener.

The room owner can manage song requests as well as remove a speaker from a seat.

The room owner can also block a seat. Listeners cannot request to take a blocked seat.

A listener can become a speaker to request songs and sing. A speaker can also become a listener.

All users can send gifts and text as well as custom messages. Custom messages can be used to send on-screen comments and give likes.

### Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see [Pricing](#).

`TRTCKaraokeRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, see [Karaoke \(iOS\)](#).

The [TRTC SDK](#) is used as a low-latency audio chat component.

The `AVChatRoom` feature of the [Chat SDK](#) is used to implement chat rooms. The attribute APIs of IM are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

## `TRTCKaraokeRoom` API Overview

### Basic SDK APIs

| API                                   | Description                                |
|---------------------------------------|--------------------------------------------|
| <a href="#">sharedInstance</a>        | Gets a singleton object.                   |
| <a href="#">destroySharedInstance</a> | Terminates a singleton object.             |
| <a href="#">setDelegate</a>           | Sets event callbacks.                      |
| <a href="#">delegateQueue</a>         | Sets the thread where event callbacks are. |
|                                       |                                            |

|                                |               |
|--------------------------------|---------------|
| <a href="#">login</a>          | Logs in.      |
| <a href="#">logout</a>         | Logs out.     |
| <a href="#">setSelfProfile</a> | Sets profile. |

## Room APIs

| API                             | Description                                                                                                                                               |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">createRoom</a>      | Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.                                           |
| <a href="#">destroyRoom</a>     | Terminates a room (called by room owner).                                                                                                                 |
| <a href="#">enterRoom</a>       | Enters a room (called by listener).                                                                                                                       |
| <a href="#">exitRoom</a>        | Exits a room (called by listener).                                                                                                                        |
| <a href="#">getRoomInfoList</a> | Gets room list details.                                                                                                                                   |
| <a href="#">getUserInfoList</a> | Gets the user information of the specified <code>userId</code> . If the value is <code>nil</code> , the information of all users in the room is obtained. |

## Music playback APIs

| API                             | Description    |
|---------------------------------|----------------|
| <a href="#">startPlayMusic</a>  | Starts music.  |
| <a href="#">stopPlayMusic</a>   | Stops music.   |
| <a href="#">pausePlayMusic</a>  | Pauses music.  |
| <a href="#">resumePlayMusic</a> | Resumes music. |

## Seat management APIs

| API                       | Description                                           |
|---------------------------|-------------------------------------------------------|
| <a href="#">enterSeat</a> | Becomes a speaker (called by room owner or listener). |
| <a href="#">leaveSeat</a> | Becomes a listener (called by speaker).               |
| <a href="#">pickSeat</a>  | Places a user in a seat (called by room owner).       |
| <a href="#">kickSeat</a>  | Removes a speaker (called by room owner).             |

|                           |                                                |
|---------------------------|------------------------------------------------|
| <a href="#">muteSeat</a>  | Mutes/Unmutes a seat (called by room owner).   |
| <a href="#">closeSeat</a> | Blocks/Unblocks a seat (called by room owner). |

## Local audio APIs

| API                                      | Description                                                       |
|------------------------------------------|-------------------------------------------------------------------|
| <a href="#">startMicrophone</a>          | Starts mic capturing.                                             |
| <a href="#">stopMicrophone</a>           | Stops mic capturing.                                              |
| <a href="#">setAudioQuality</a>          | Sets audio quality.                                               |
| <a href="#">muteLocalAudio</a>           | Mutes/Unmutes local audio.                                        |
| <a href="#">setSpeaker</a>               | Sets whether to play sound from the device's speaker or receiver. |
| <a href="#">setAudioCaptureVolume</a>    | Sets mic capturing volume.                                        |
| <a href="#">setAudioPlayoutVolume</a>    | Sets playback volume.                                             |
| <a href="#">setVoiceEarMonitorEnable</a> | Enables/Disables in-ear monitoring.                               |

## Remote audio APIs

| API                                | Description                       |
|------------------------------------|-----------------------------------|
| <a href="#">muteRemoteAudio</a>    | Mutes/Unmutes a specified member. |
| <a href="#">muteAllRemoteAudio</a> | Mutes/Unmutes all members.        |

## Background music and audio effect APIs

| API                                   | Description                                                                                         |
|---------------------------------------|-----------------------------------------------------------------------------------------------------|
| <a href="#">getAudioEffectManager</a> | Gets the background music and audio effect management object <a href="#">TXAudioEffectManager</a> . |

## Message sending APIs

| API                             | Description                                                                                  |
|---------------------------------|----------------------------------------------------------------------------------------------|
| <a href="#">sendRoomTextMsg</a> | Broadcasts a text chat message in a room. This API is generally used for on-screen comments. |

|                                |                                   |
|--------------------------------|-----------------------------------|
| <code>sendRoomCustomMsg</code> | Sends a custom text chat message. |
|--------------------------------|-----------------------------------|

### Invitation signaling APIs

| API                           | Description             |
|-------------------------------|-------------------------|
| <code>sendInvitation</code>   | Sends an invitation.    |
| <code>acceptInvitation</code> | Accepts an invitation.  |
| <code>rejectInvitation</code> | Declines an invitation. |
| <code>cancelInvitation</code> | Cancels an invitation.  |

## TRTCKaraokeRoomDelegate API Overview

### Common event callbacks

| API                     | Description           |
|-------------------------|-----------------------|
| <code>onError</code>    | Callback for error.   |
| <code>onWarning</code>  | Callback for warning. |
| <code>onDebugLog</code> | Callback of log.      |

### Room event callback APIs

| API                             | Description                   |
|---------------------------------|-------------------------------|
| <code>onRoomDestroy</code>      | The room was terminated.      |
| <code>onRoomInfoChange</code>   | The room information changed. |
| <code>onUserVolumeUpdate</code> | User volume                   |

### Seat list change callback APIs

| API                           | Description       |
|-------------------------------|-------------------|
| <code>onSeatListChange</code> | All seat changes. |
|                               |                   |

|                                      |                                                                    |
|--------------------------------------|--------------------------------------------------------------------|
| <a href="#">onAnchorEnterSeat</a>    | A user became a speaker or was made a speaker by the room owner.   |
| <a href="#">onAnchorLeaveSeat</a>    | A user became a listener or was made a listener by the room owner. |
| <a href="#">onSeatMute</a>           | The room owner muted a seat.                                       |
| <a href="#">onUserMicrophoneMute</a> | Whether a user's mic is muted                                      |
| <a href="#">onSeatClose</a>          | The room owner blocked a seat.                                     |

### Callback APIs for room entry/exit by listener

| API                             | Description                  |
|---------------------------------|------------------------------|
| <a href="#">onAudienceEnter</a> | A listener entered the room. |
| <a href="#">onAudienceExit</a>  | A listener exited the room.  |

### Message event callback APIs

| API                                 | Description                       |
|-------------------------------------|-----------------------------------|
| <a href="#">onRecvRoomTextMsg</a>   | A text chat message was received. |
| <a href="#">onRecvRoomCustomMsg</a> | A custom message was received.    |

### Signaling event callback APIs

| API                                    | Description                     |
|----------------------------------------|---------------------------------|
| <a href="#">onReceiveNewInvitation</a> | Receipt of an invitation.       |
| <a href="#">onInviteeAccepted</a>      | Invitation accepted by invitee. |
| <a href="#">onInviteeRejected</a>      | Invitation declined by invitee. |
| <a href="#">onInvitationCancelled</a>  | Invitation canceled by inviter. |

### Song event callback APIs

| API                                   | Description              |
|---------------------------------------|--------------------------|
| <a href="#">onMusicProgressUpdate</a> | Music playback progress. |
| <a href="#">onMusicPrepareToPlay</a>  | Music playback is ready. |
|                                       |                          |

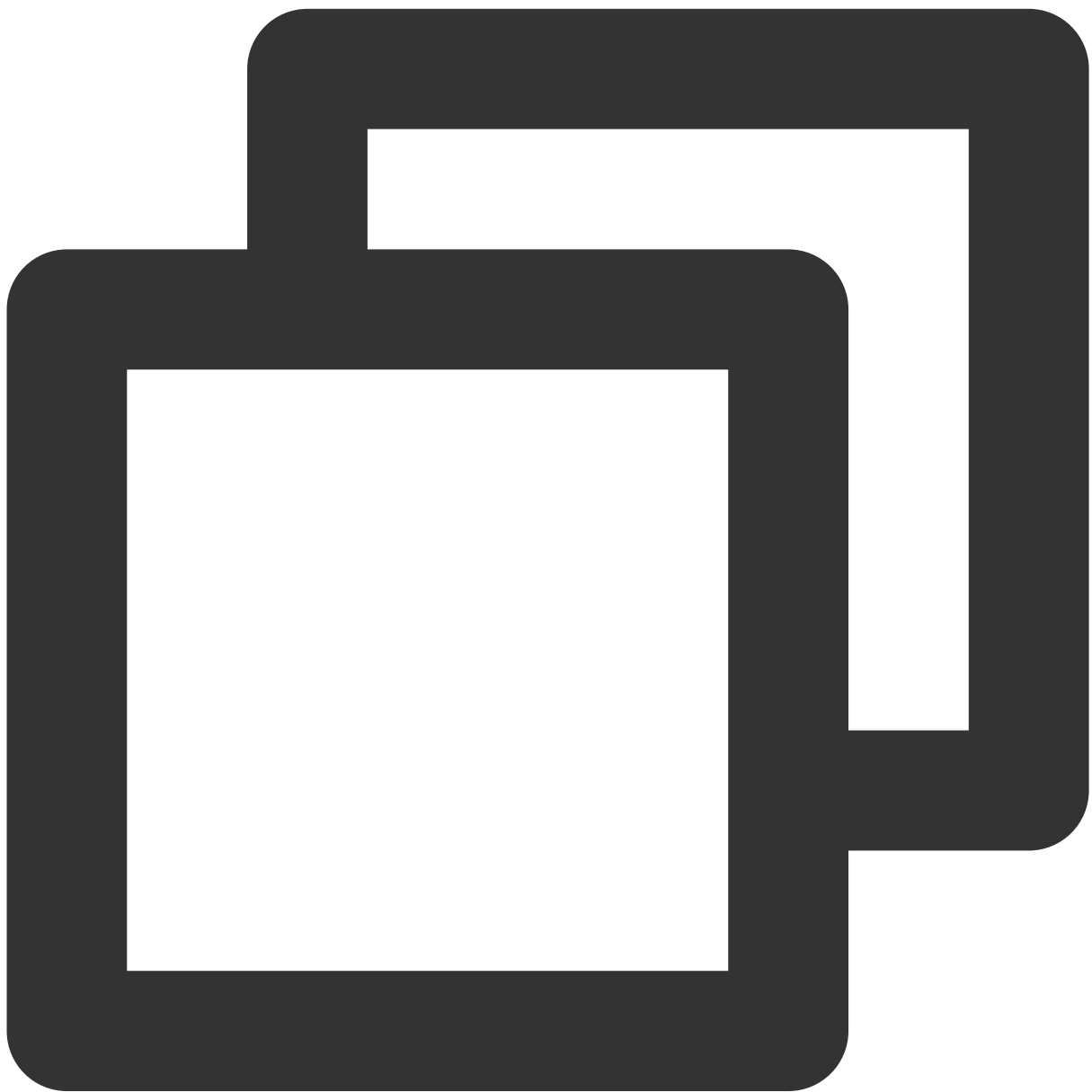
`onMusicCompletePlaying`

Music playback was completed.

## Basic SDK APIs

### **sharedInstance**

This API is used to get a [TRTCKaraokeRoom](#) singleton object.



```
/**
 * Get a `TRTCKaraokeRoom` singleton object
 *
 * - returns: `TRTCKaraokeRoom` instance
 * - note: To terminate a singleton object, call {@link TRTCKaraokeRoom#destroyShare
 */
+ (instancetype) sharedInstance NS_SWIFT_NAME(shared());
```

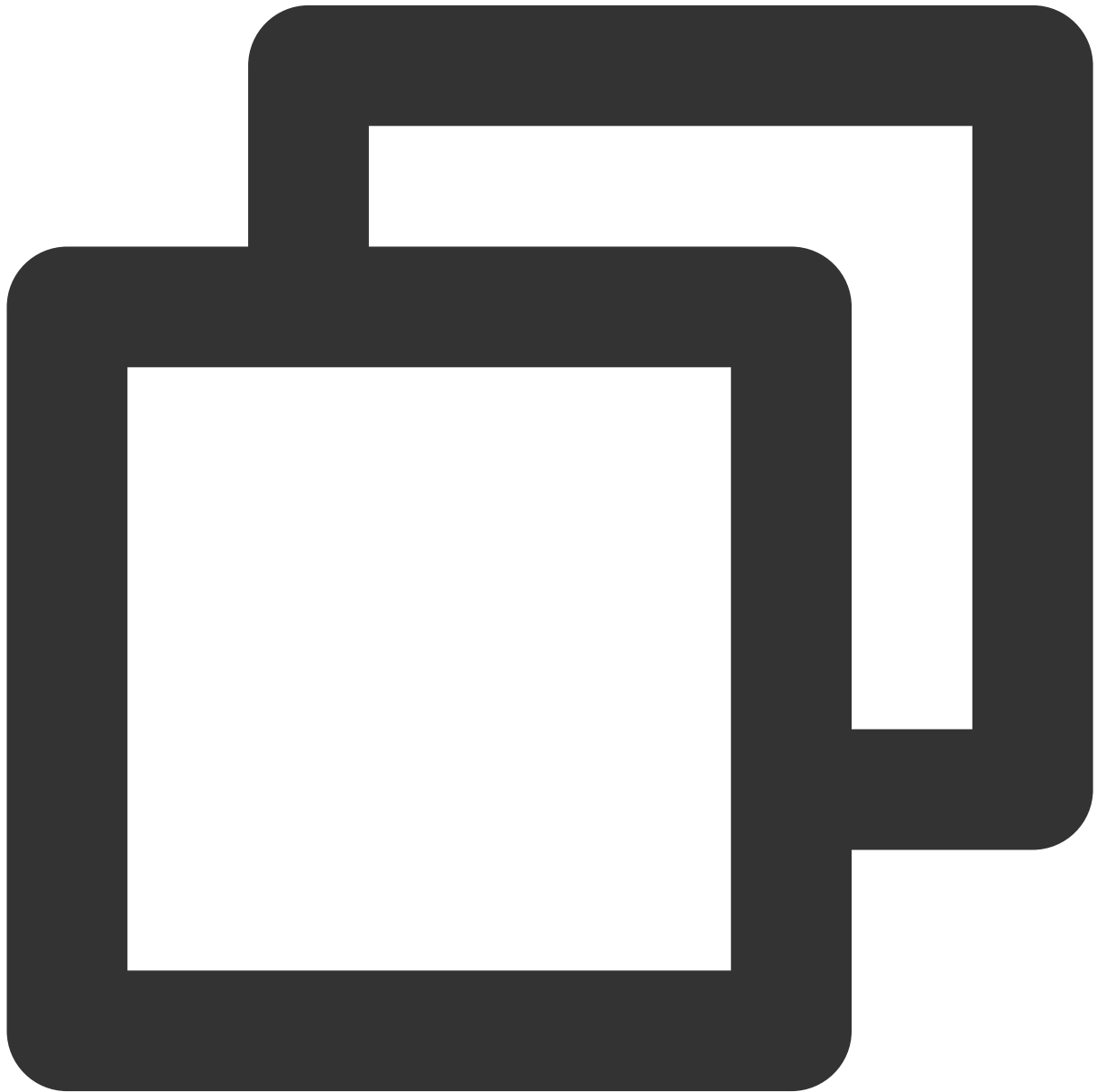
## destroySharedInstance

This API is used to terminate a [TRTCKaraokeRoom](#) singleton object.

### Note

After the instance is terminated, the externally cached `TRTCKaraokeRoom` instance can no longer be used. You need to call [sharedInstance](#) again to get a new instance.

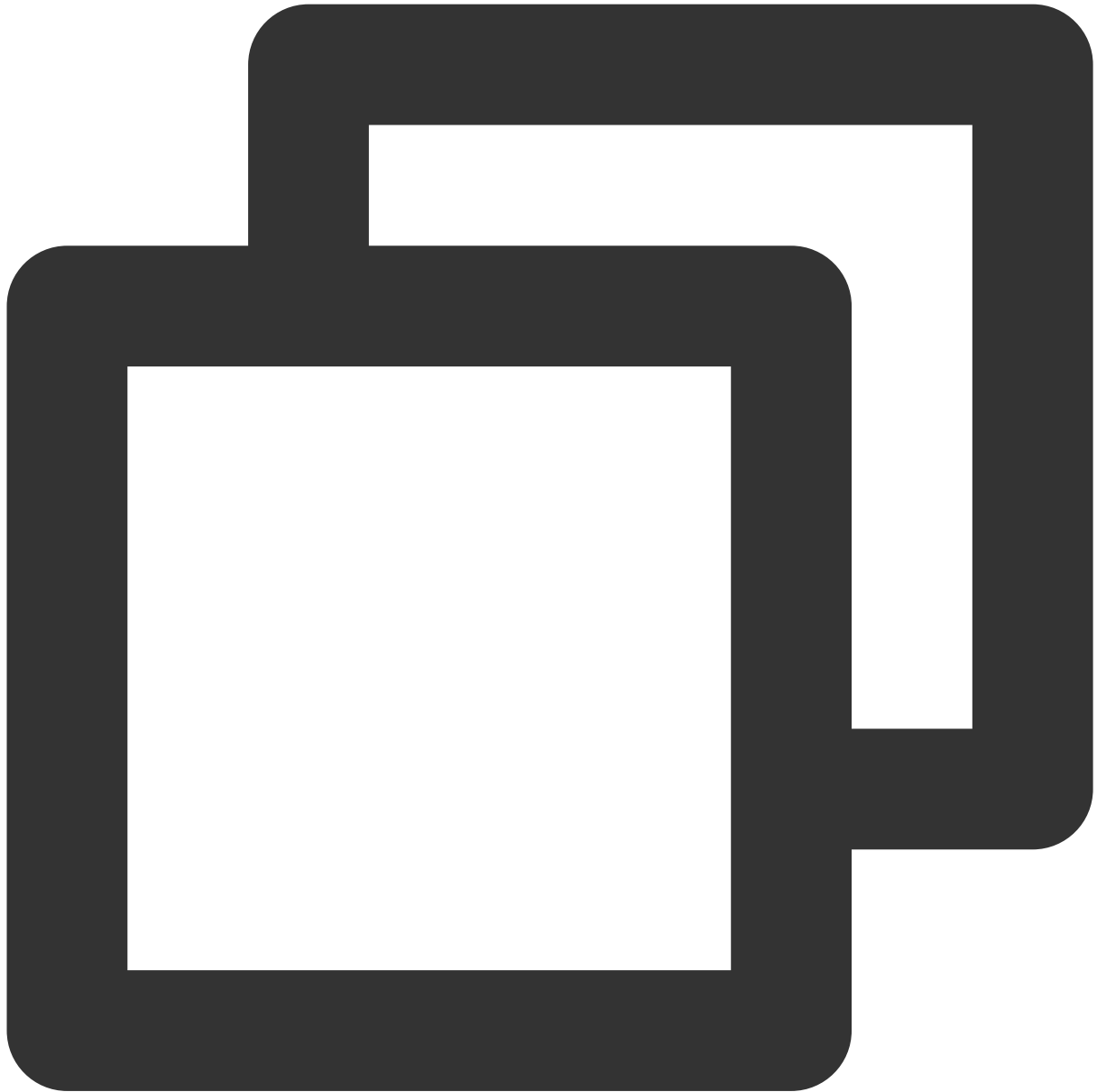




```
/**
 * Terminate the `TRTCKaraokeRoom` singleton object
 *
 * - note: After the instance is terminated, the externally cached `TRTCKaraokeRoom`
 */
+ (void)destroySharedInstance NS_SWIFT_NAME(destroyShared());
```

## setDelegate

This API is used to set the event callbacks of [TRTCKaraokeRoom](#). You can use `TRTCKaraokeRoomDelegate` to get different status notifications of [TRTCKaraokeRoom](#).



```
/**
 * Set the event callbacks of the component
 *
 * You can use `TRTCKaraokeRoomDelegate` to get different status notifications of `TRTCKaraokeRoom`
 *
 * - parameter delegate Callback API
 * - note: Callbacks in `TRTCKaraokeRoom` are sent to you in the main queue by default
 */
```

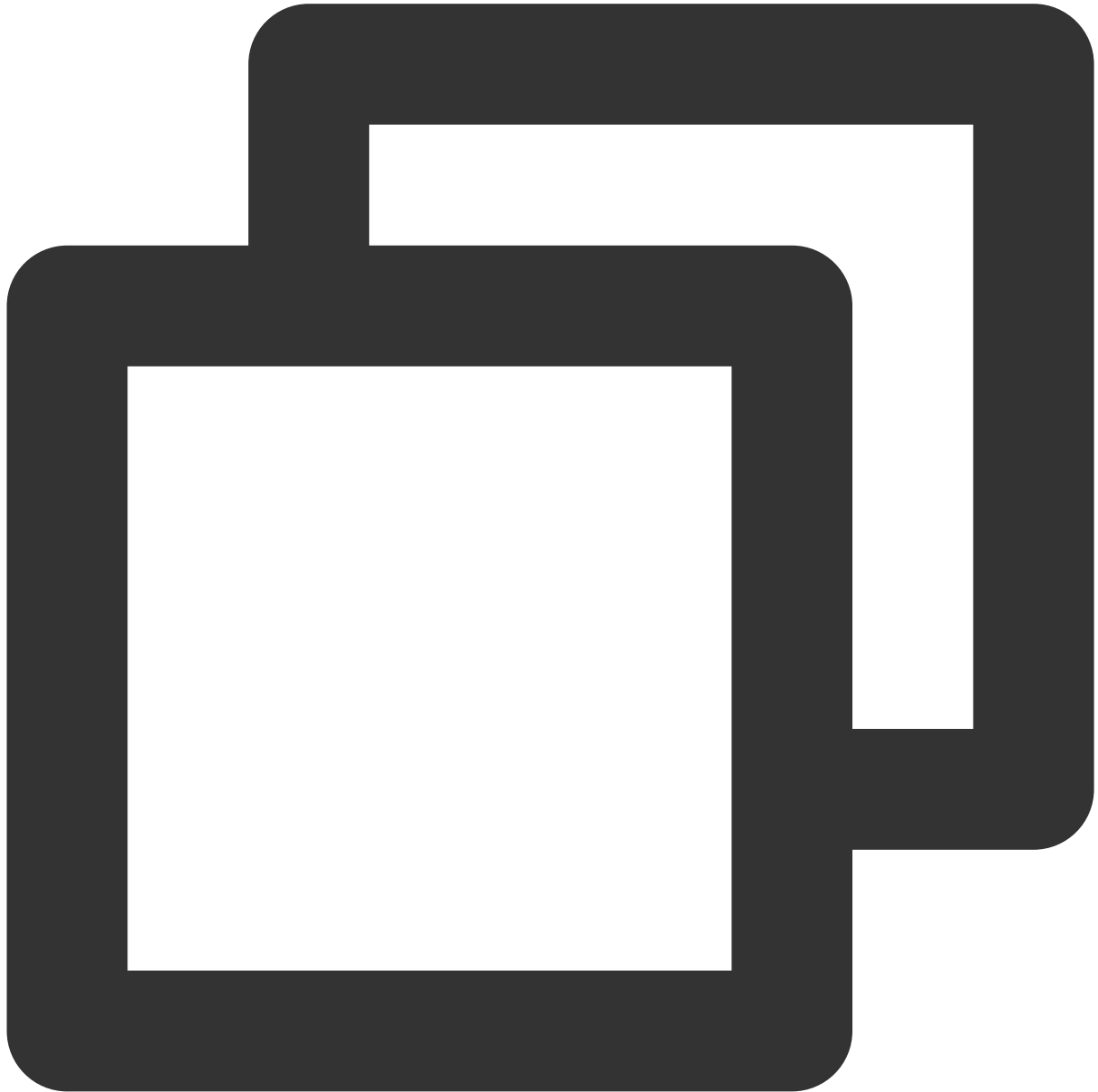
```
- (void)setDelegate:(id<TRTCKaraokeRoomDelegate>)delegate NS_SWIFT_NAME(setDelegate)
```

**Note**

`setDelegate` is the delegate callback of `TRTCKaraokeRoom` .

**setDelegateQueue**

This API is used to set the thread queue for event callbacks. The main thread (MainQueue) is used by default.



```
/**  
 * Set the queue for event callbacks  
 */
```

```
* - parameter queue. The status notifications of `TRTCKaraokeRoom` will be sent to
*/
- (void)setDelegateQueue:(dispatch_queue_t)queue NS_SWIFT_NAME(setDelegateQueue(queue))
```

The parameters are described below:

| Parameter | Type             | Description                                                                                        |
|-----------|------------------|----------------------------------------------------------------------------------------------------|
| queue     | dispatch_queue_t | The status notifications of <code>TRTCKaraokeRoom</code> are sent to the thread queue you specify. |

## login

Login



```
- (void)login:(int) sdkAppID
    userId:(NSString *)userId
    userSig:(NSString *)userSig
    callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(login(sdkAppID:userId:
    userSig:userSig:callback:))
```

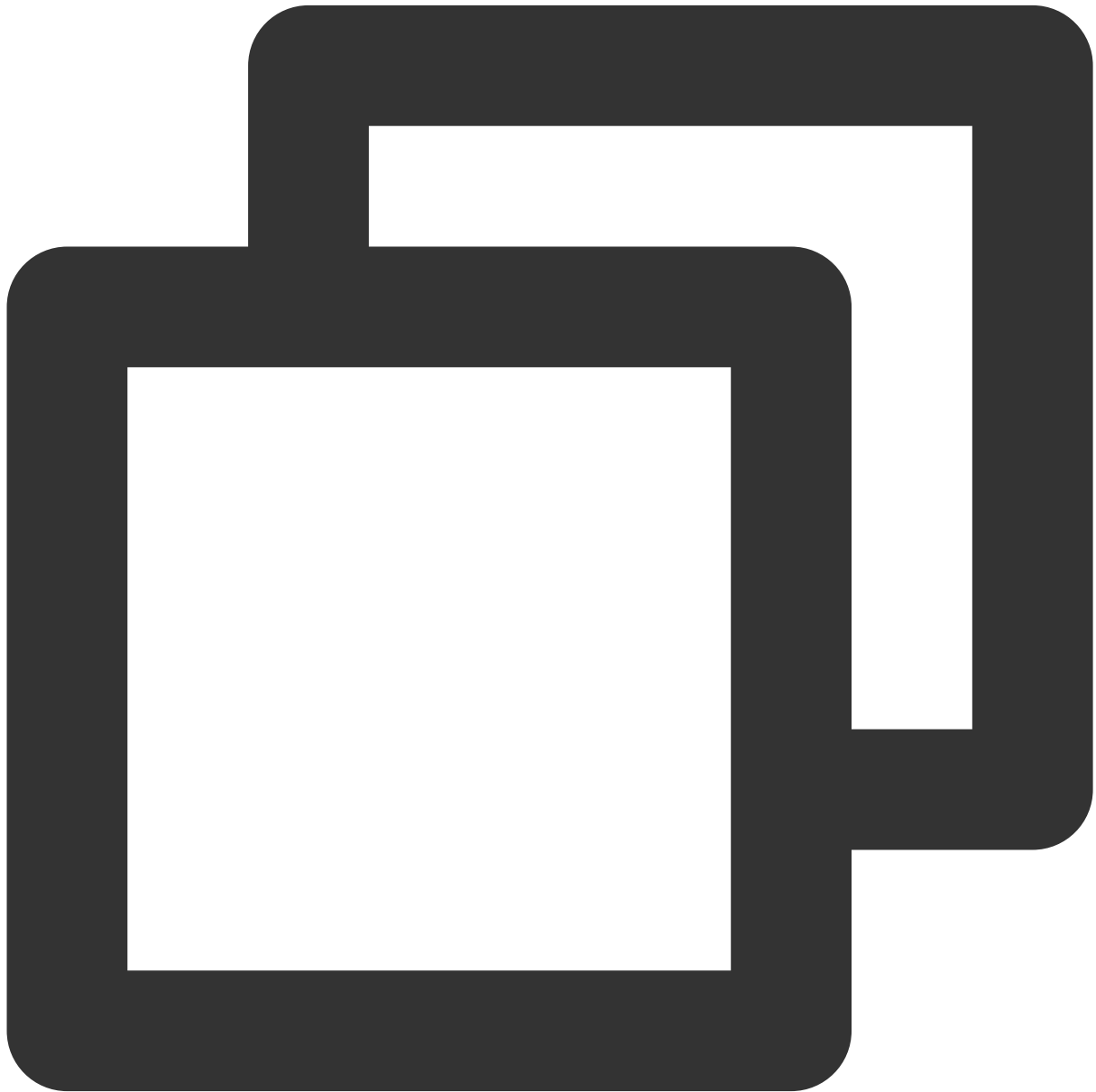
The parameters are described below:

| Parameter | Type | Description                                                                                            |
|-----------|------|--------------------------------------------------------------------------------------------------------|
| sdkAppId  | int  | You can view the <code>SDKAppID</code> via <a href="#">Application Management</a> > <b>Application</b> |

|          |                | <b>Info</b> in the TRTC console.                                                                                                           |
|----------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| userId   | String         | The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). |
| userSig  | String         | Tencent Cloud's proprietary security signature. For how to calculate and use it, see <a href="#">FAQs &gt; UserSig</a> .                   |
| callback | ActionCallback | The callback for login. The code is <code>0</code> if login succeeds.                                                                      |

## logout

Log out



```
- (void)logout:(ActionCallback _Nullable)callback NS_SWIFT_NAME(logout(callback:));
```

The parameters are described below:

| Parameter | Type           | Description                                                |
|-----------|----------------|------------------------------------------------------------|
| callback  | ActionCallback | The callback for logout. The code is 0 if logout succeeds. |

## setSelfProfile

This API is used to set the profile.



```
- (void)setSelfProfile:(NSString *)userName avatarURL:(NSString *)avatarURL callback
```

The parameters are described below:

| Parameter | Type           | Description                                                            |
|-----------|----------------|------------------------------------------------------------------------|
| userName  | String         | The username.                                                          |
| avatar    | String         | The address of the profile photo.                                      |
| callback  | ActionCallback | The callback for profile configuration. The code is 0 if the operation |

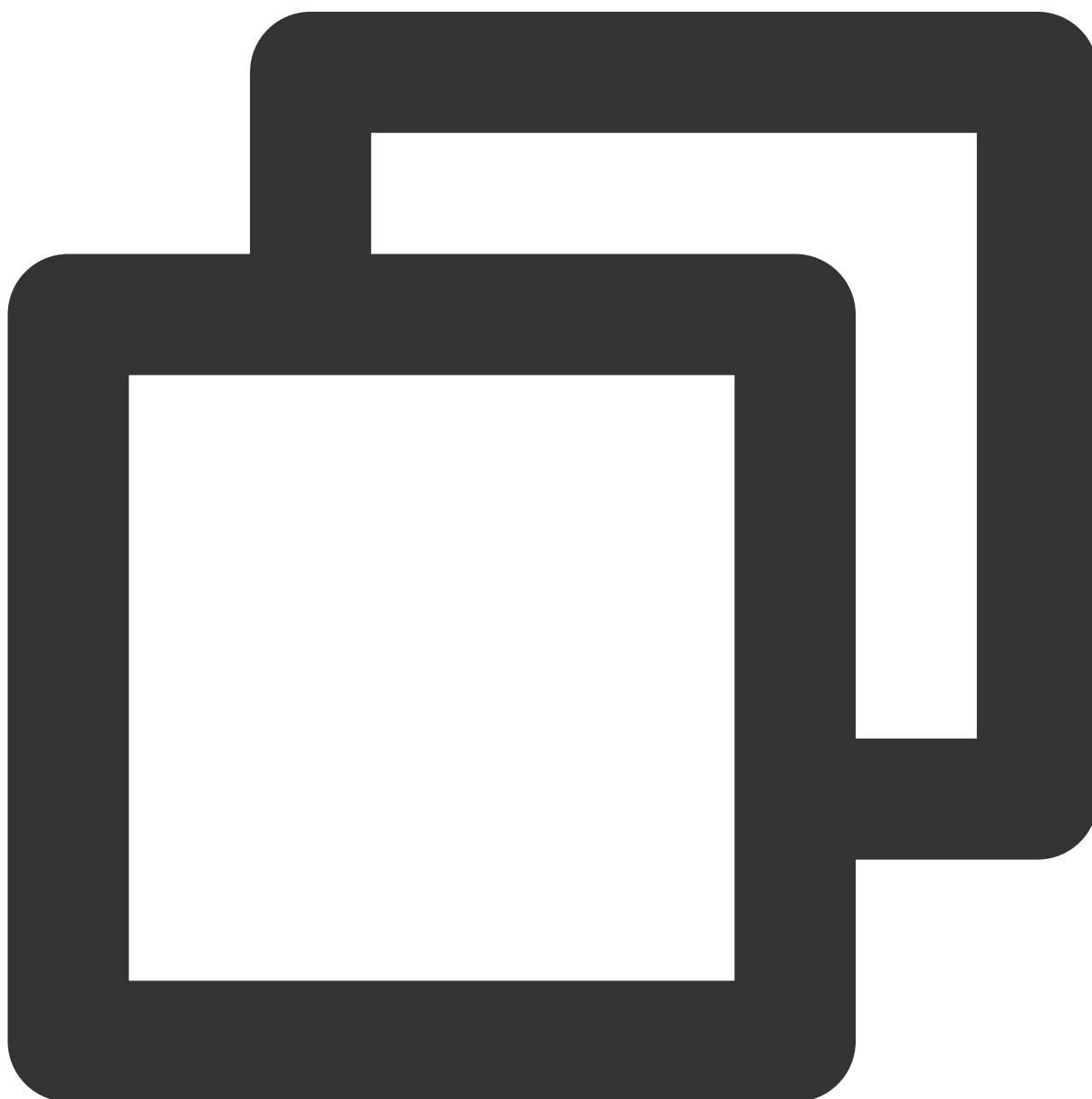


|  |  |           |
|--|--|-----------|
|  |  | succeeds. |
|--|--|-----------|

## Room APIs

### **createRoom**

This API is used to create a room (called by room owner).



```
- (void)createRoom:(int)roomId roomParam:(RoomParam *)roomParam callback:(ActionCal
```

The parameters are described below:

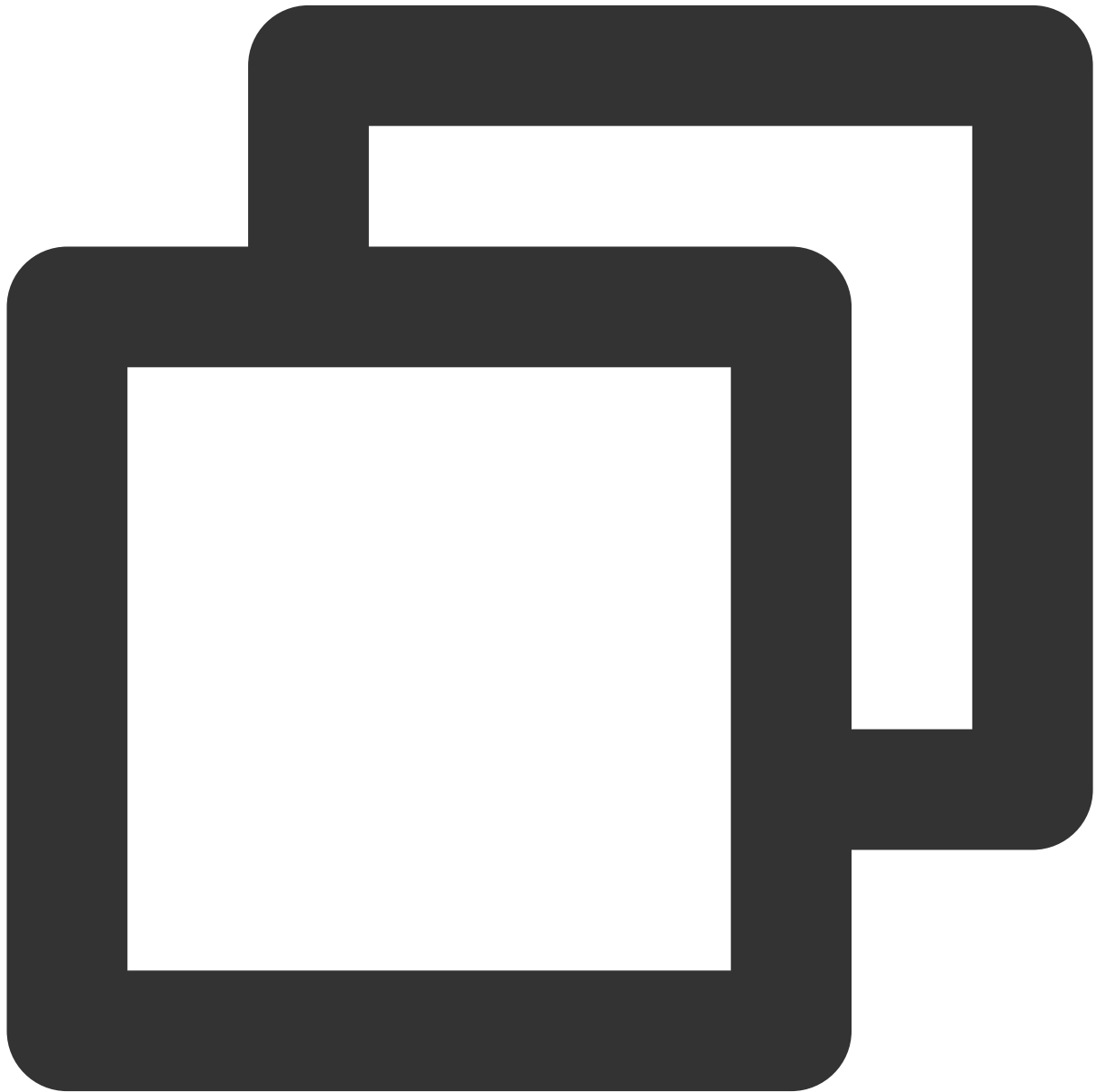
| Parameter | Type                | Description                                                                                                                                                                                                                                                           |
|-----------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| roomId    | int                 | The room ID. You need to assign and manage room IDs in a centralized manner. Multiple <code>roomId</code> values can be aggregated into a room list. Currently, Tencent Cloud does not provide management services for room lists. Please manage your own room lists. |
| roomParam | TRTCCreateRoomParam | Room information, such as room name, seat list information, and cover information. To manage seats, you must enter the number of seats in the room.                                                                                                                   |
| callback  | ActionCallback      | The callback for room creation. The code is 0 if the operation succeeds.                                                                                                                                                                                              |

The process of creating a karaoke room and becoming a speaker is as follows:

1. A user calls `createRoom` to create a karaoke room, passing in room attributes (e.g., room ID, whether listeners need room owner's permission to speak, number of seats).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user will receive an `onSeatListChangest` notification about the change of the seat list, and can update the change to the UI.
4. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

## destroyRoom

This API is used to terminate a room (called by room owner).



```
- (void)destroyRoom:(ActionCallback _Nullable)callback NS_SWIFT_NAME(destroyRoom(callback))
```

The parameters are described below:

| Parameter | Type           | Description                                                                 |
|-----------|----------------|-----------------------------------------------------------------------------|
| callback  | ActionCallback | The callback for room termination. The code is 0 if the operation succeeds. |

## enterRoom

This API is used to enter a room (called by listener).



```
- (void)enterRoom:(NSInteger)roomId callback:(ActionCallback _Nullable)callback NS_
```

The parameters are described below:

| Parameter | Type           | Description                                                           |
|-----------|----------------|-----------------------------------------------------------------------|
| roomId    | int            | The room ID.                                                          |
| callback  | ActionCallback | The callback for room entry. The code is 0 if the operation succeeds. |

The process of entering a room as a listener is as follows:

1. A user gets the latest karaoke room list from your server. The list may contain the `roomId` and room information of multiple karaoke rooms.
2. The user selects a room, and enters the room by calling `enterRoom` with the room ID passed in.
3. After entering the room, the user receives an `onRoomInfoChange` notification about room attribute change from the component. The attributes can be recorded, and corresponding changes can be made to the UI, including room name, whether room owner's permission is required for listeners to speak, etc.
4. The user will receive an `onSeatListChange` notification about the change of the seat list and can update the change to the UI.
5. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker.

## **exitRoom**

Leave room



```
- (void)exitRoom:(ActionCallback _Nullable)callback NS_SWIFT_NAME(exitRoom(callback
```

The parameters are described below:

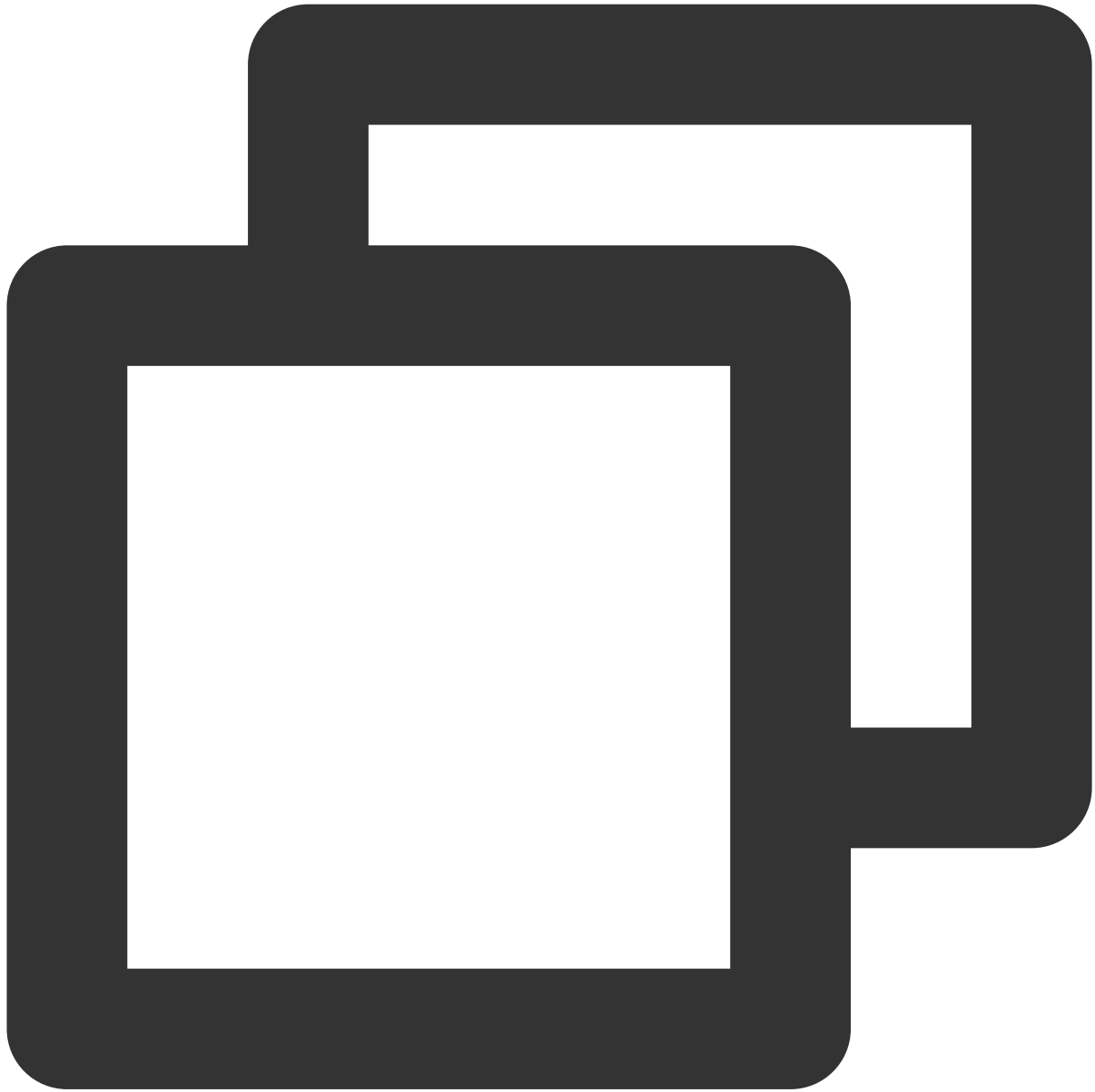
| Parameter | Type           | Description                                                          |
|-----------|----------------|----------------------------------------------------------------------|
| callback  | ActionCallback | The callback for room exit. The code is 0 if the operation succeeds. |

## getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()`.

**Note**

You don't need this API if both the room list and room information are managed on your server.



```
- (void)getRoomInfoList:(NSArray<NSNumber *> *)roomIdList callback:(KaraokeInfoCall
```

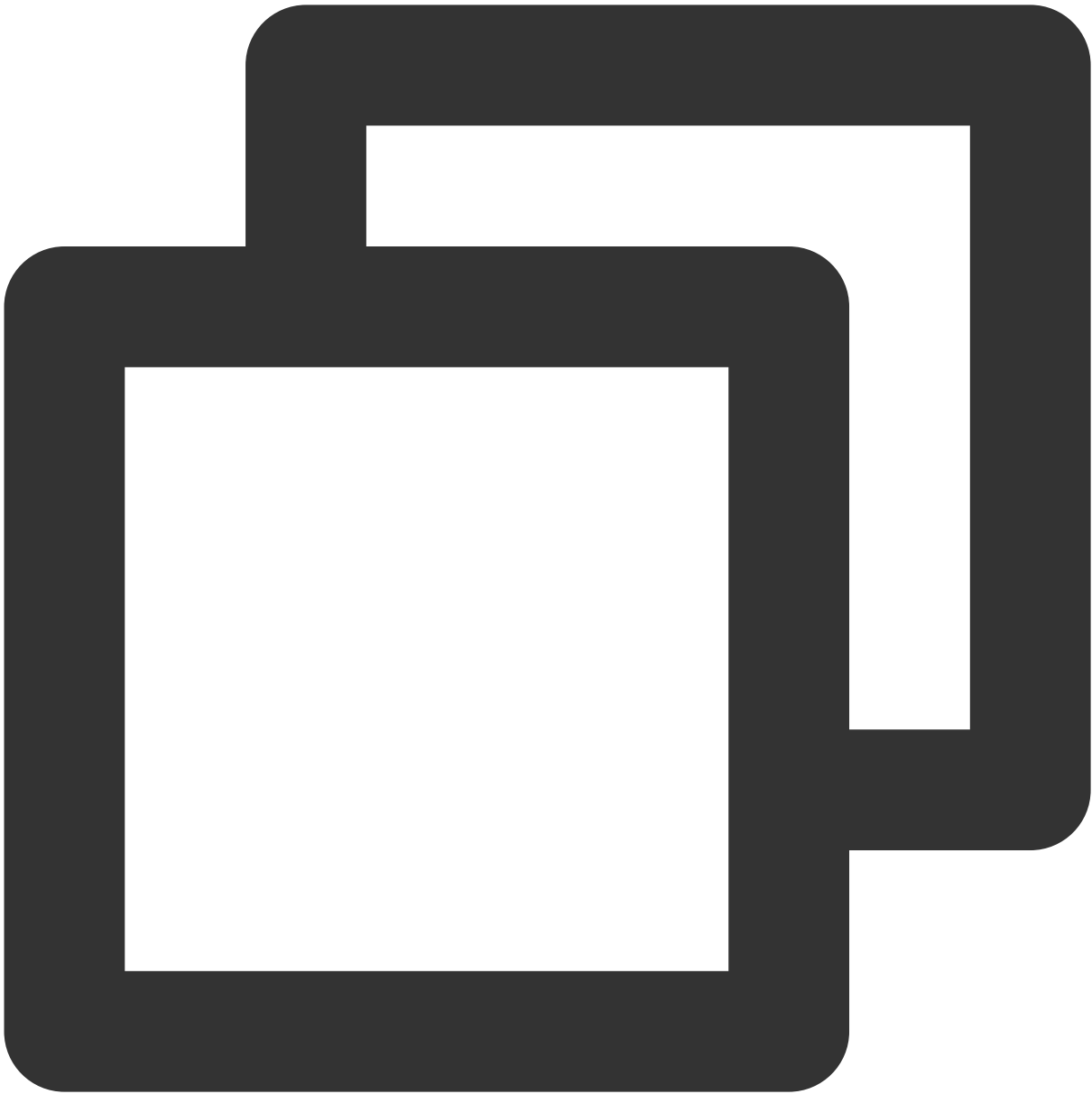
The parameters are described below:

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

|            |                  |                               |
|------------|------------------|-------------------------------|
| roomIdList | List<Integer>    | The room ID list.             |
| callback   | RoomInfoCallback | The callback of room details. |

### getUserInfoList

This API is used to get the information of specific users ( `userId` ).



```
- (void)getUserInfoList:(NSArray<NSString *> * _Nullable)userIdList callback:(Karao
```

The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|



| Parameter        | Type             | Description                                                                                                          |
|------------------|------------------|----------------------------------------------------------------------------------------------------------------------|
| userIdList       | List<String>     | The user IDs to query. If this parameter is <code>null</code> , the information of all users in the room is queried. |
| userlistcallback | UserListCallback | The callback of user details.                                                                                        |

## Music Playback APIs

### startPlayMusic

This API is used to play music (called after becoming a speaker).

#### Note

After music playback starts, you will receive an `onMusicPrepareToPlay` notification.

During music playback, all members in the room will continuously receive an `onMusicProgressUpdate` notification.

After music playback stops, you will receive an `onMusicCompletePlaying` notification.



```
- (void)startPlayMusic:(int32_t)musicID originalUrl:(NSString *)originalUrl accompa
```

The parameters are described below:

| Parameter    | Type    | Description                                  |
|--------------|---------|----------------------------------------------|
| musicID      | int32_t | The music ID.                                |
| originalUrl  | String  | The absolute path of the vocal track.        |
| accompanyUrl | String  | The absolute path of the instrumental track. |

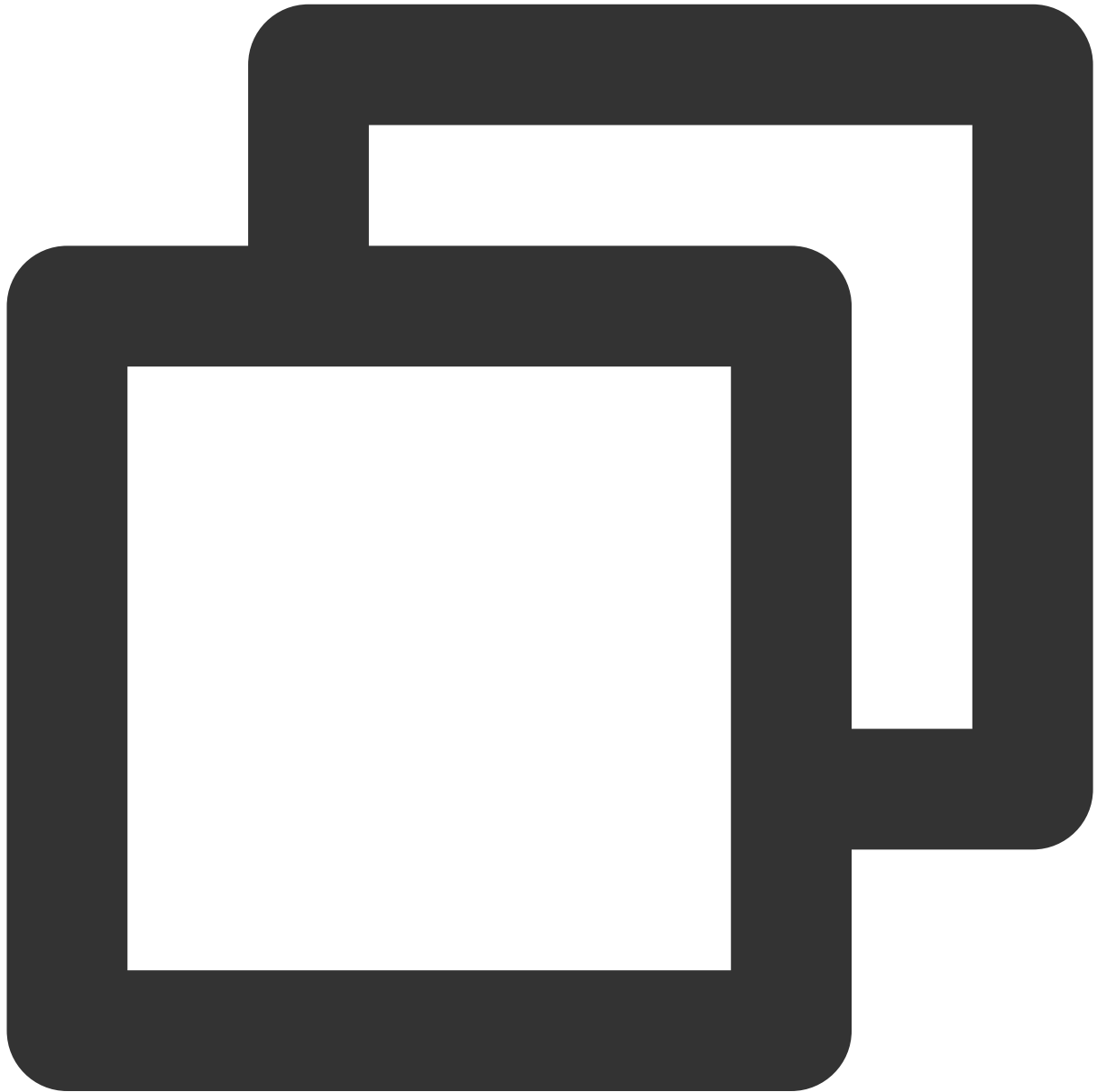
After this API is called, the song being played will stop.

## stopPlayMusic

This API is used to stop music (called during music playback).

### Note

After music playback stops, you will receive an `onMusicCompletePlaying` notification.



```
- (void)stopPlayMusic NS_SWIFT_NAME(stopPlayMusic());
```

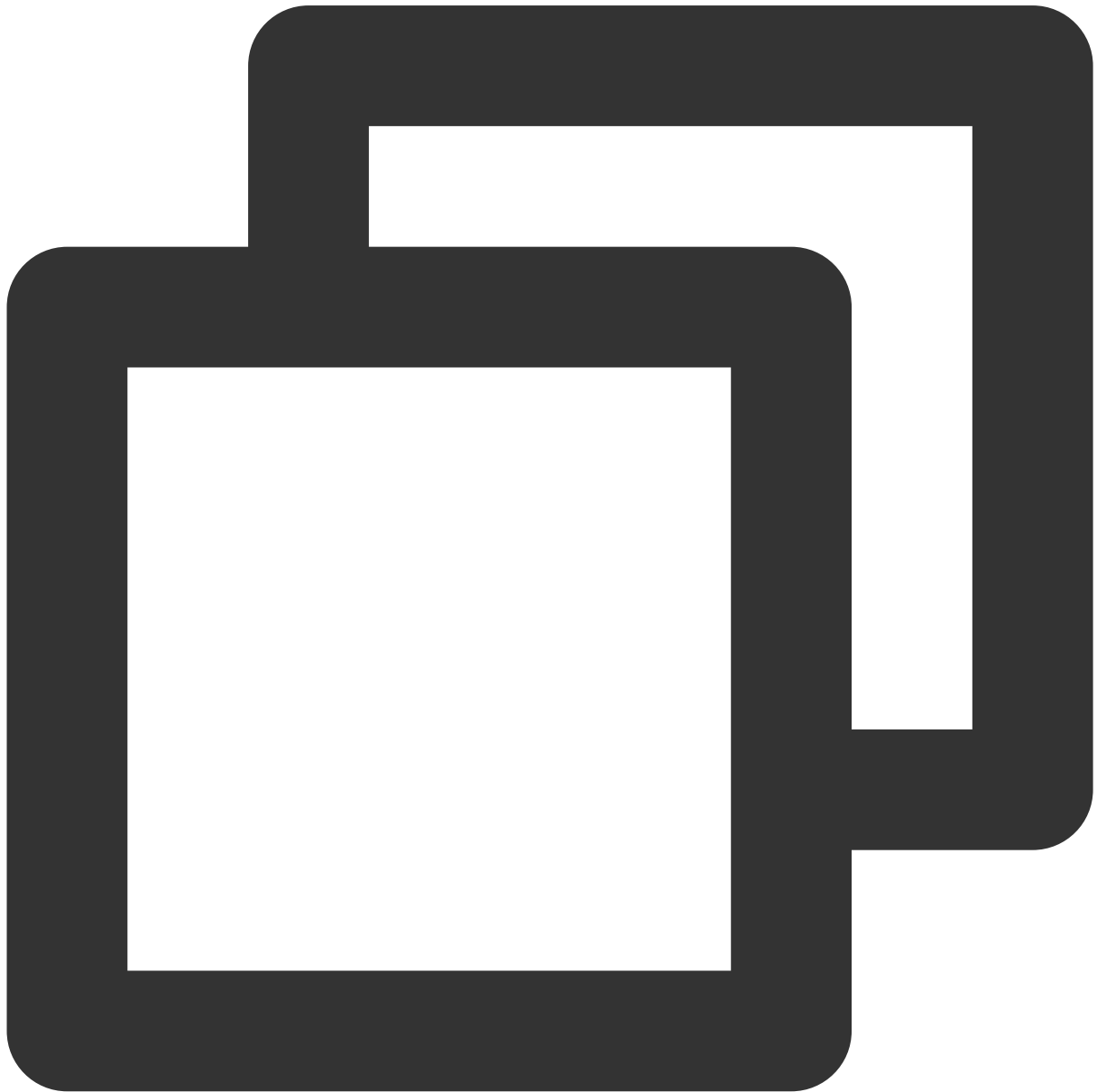
## pausePlayMusic

This API is used to pause music (called during music playback).

### Note

The `onMusicProgressUpdate` notification will be paused.

No `onMusicCompletePlaying` notification will be received.



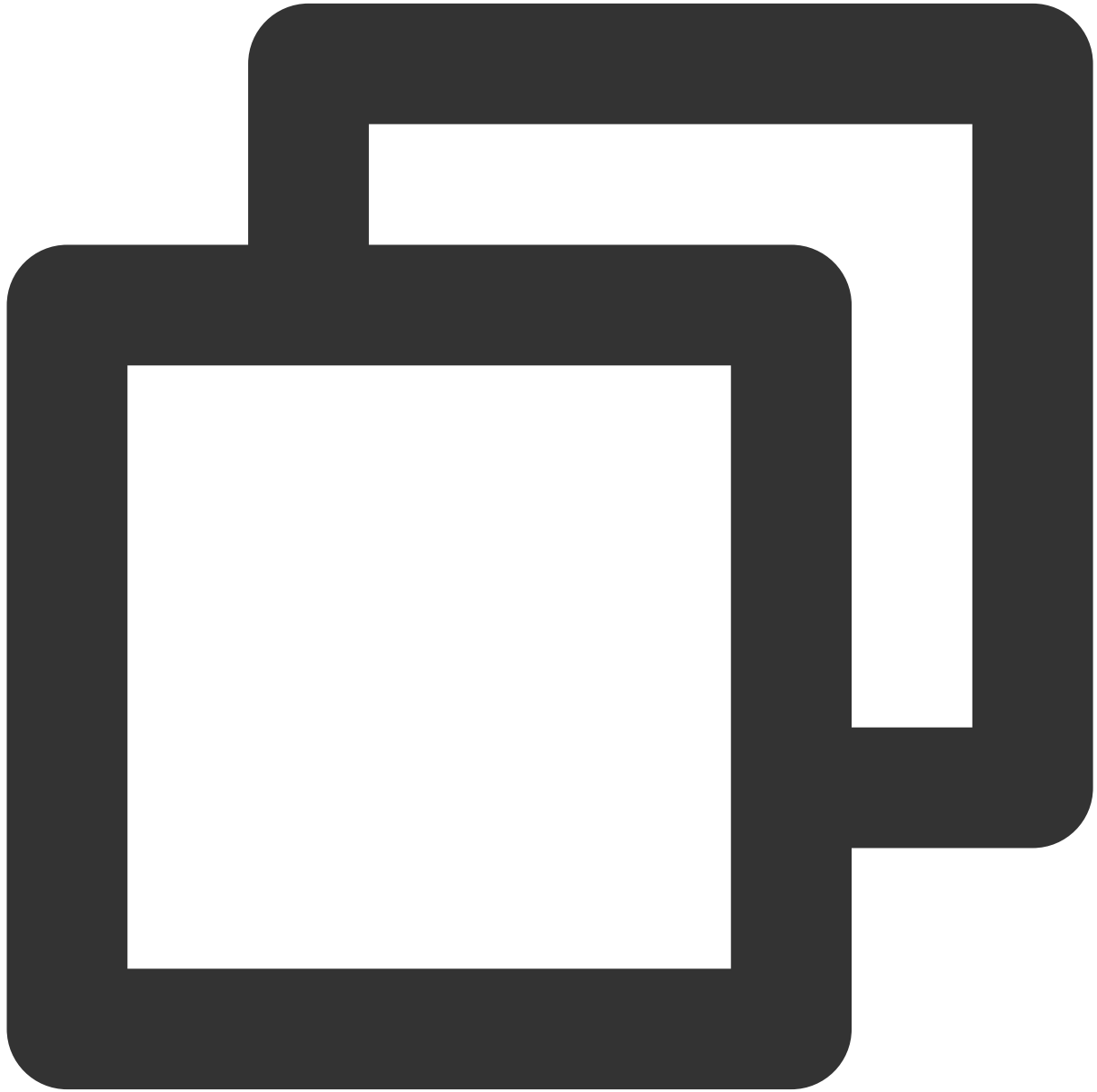
```
- (void)pausePlayMusic NS_SWIFT_NAME(pausePlayMusic());
```

## resumePlayMusic

This API is used to resume music (called after music playback is paused).

**Note**

No `onMusicPrepareToPlay` notification will be received.



```
- (void)resumePlayMusic NS_SWIFT_NAME(resumePlayMusic());
```

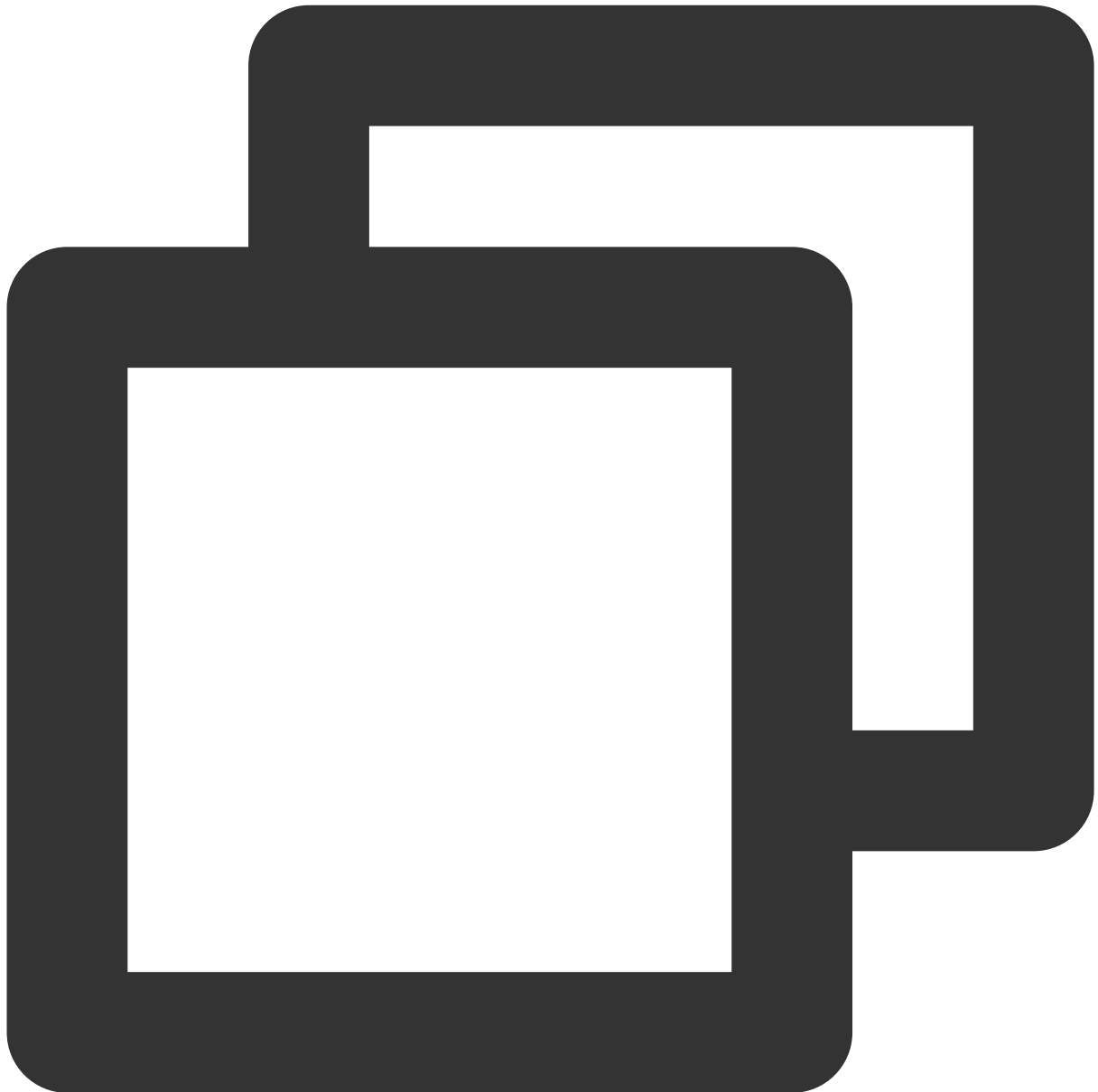
## Seat Management APIs

## enterSeat

This API is used to become a speaker (called by room owner or listener).

### Note

After a user becomes a speaker, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.



```
- (void)enterSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback
```

The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter | Type           | Description                         |
|-----------|----------------|-------------------------------------|
| seatIndex | int            | The number of the seat to be taken. |
| callback  | ActionCallback | The callback for the operation.     |

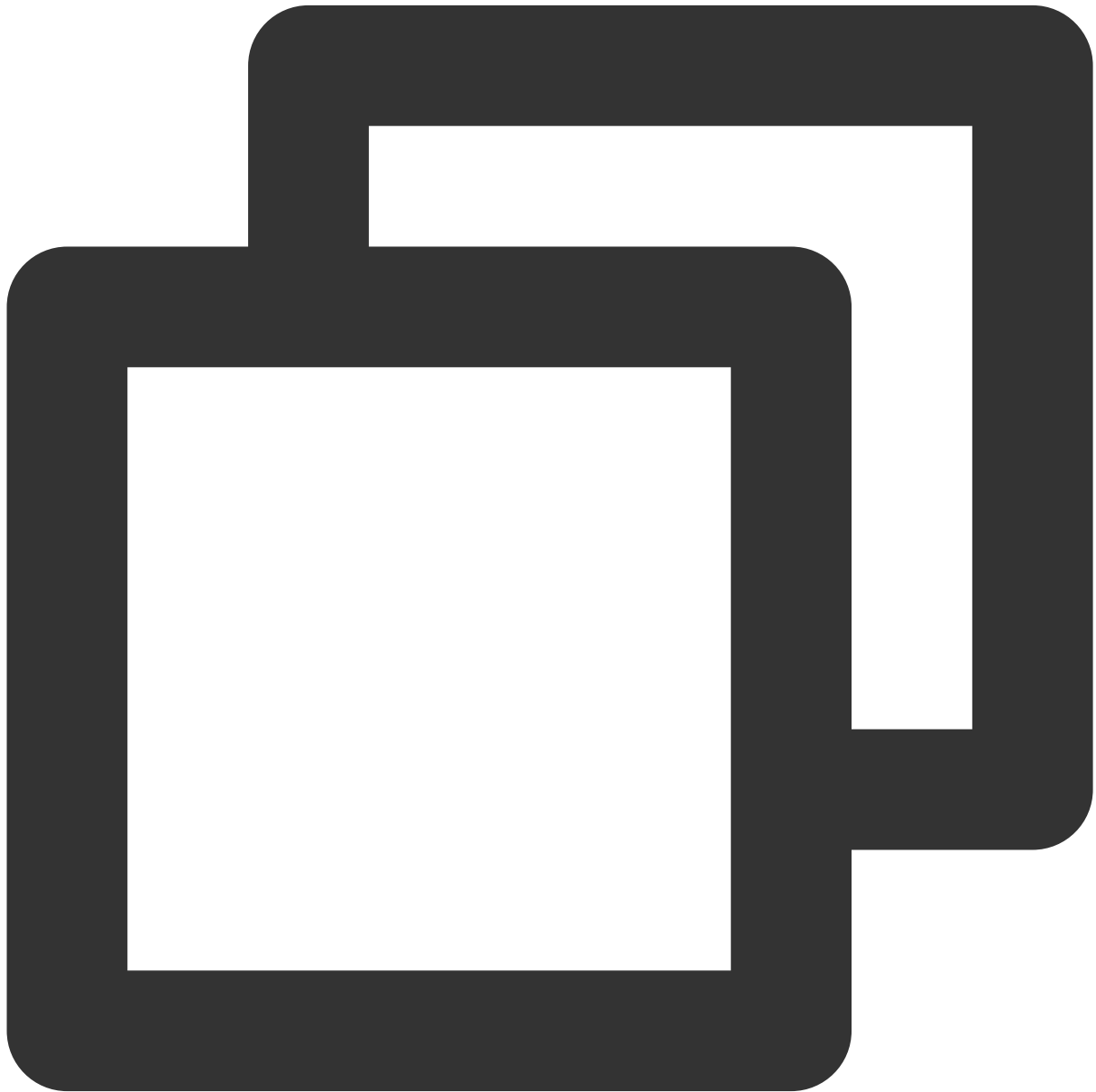
Calling this API will immediately modify the seat list. In cases where listeners need the room owner's permission to take a seat, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call this API.

## leaveSeat

This API is used to become a listener (called by speaker).

### Note

After a speaker becomes a listener, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.



```
- (void)leaveSeat:(ActionCallback _Nullable)callback NS_SWIFT_NAME(leaveSeat(callback))
```

The parameters are described below:

| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| callback  | ActionCallback | The callback for the operation. |

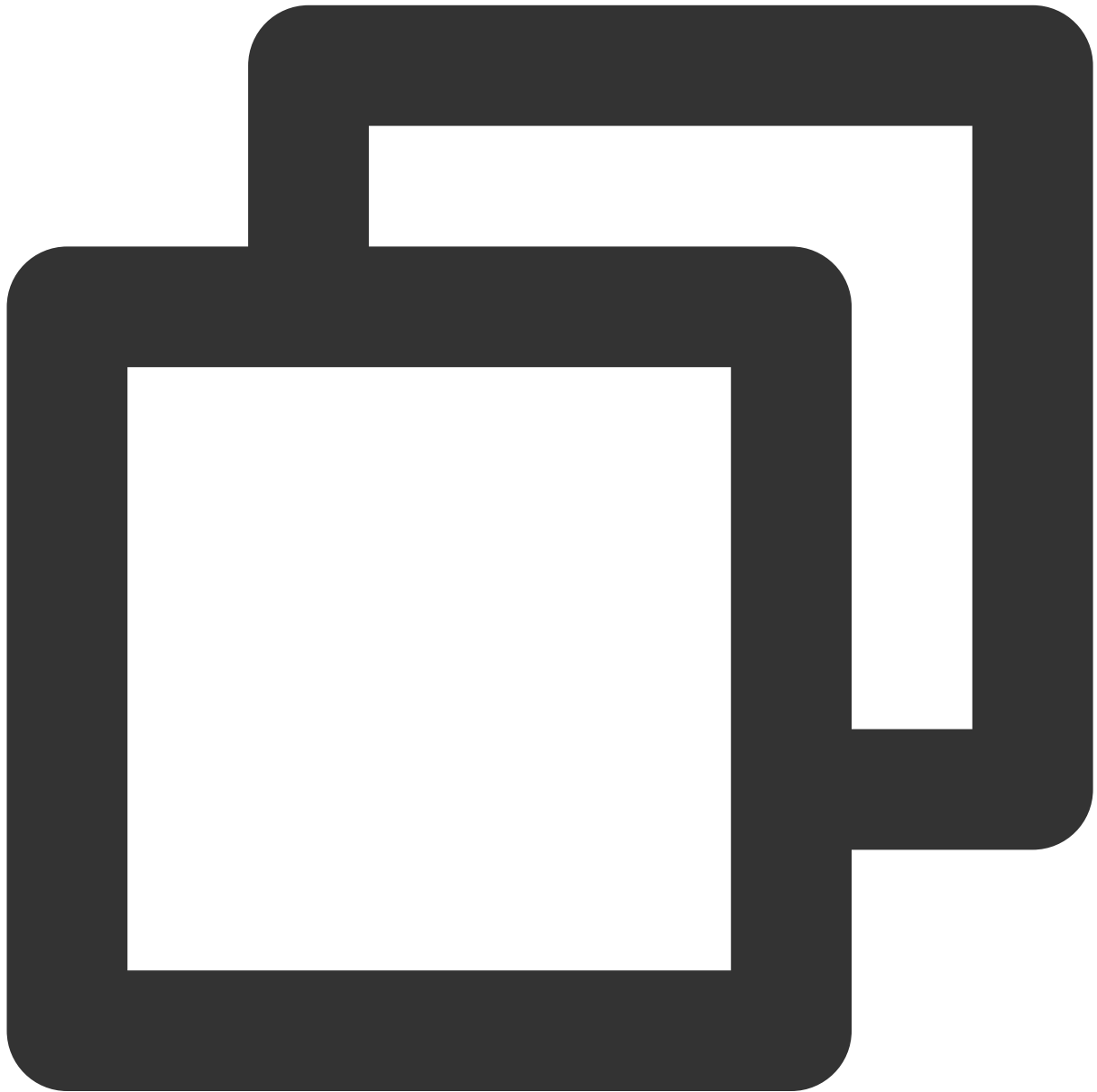
## **pickSeat**

This API is used to place a user in a seat (called by room owner).



**Note**

After the room owner makes someone a speaker, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.



```
- (void)pickSeat:(NSInteger)seatIndex userId:(NSString *)userId callback:(ActionCal
```

The parameters are described below:

| Parameter | Type | Description                                      |
|-----------|------|--------------------------------------------------|
| seatIndex | int  | The number of the seat to place the listener in. |

|          |                |                                 |
|----------|----------------|---------------------------------|
| userId   | String         | The User ID.                    |
| callback | ActionCallback | The callback for the operation. |

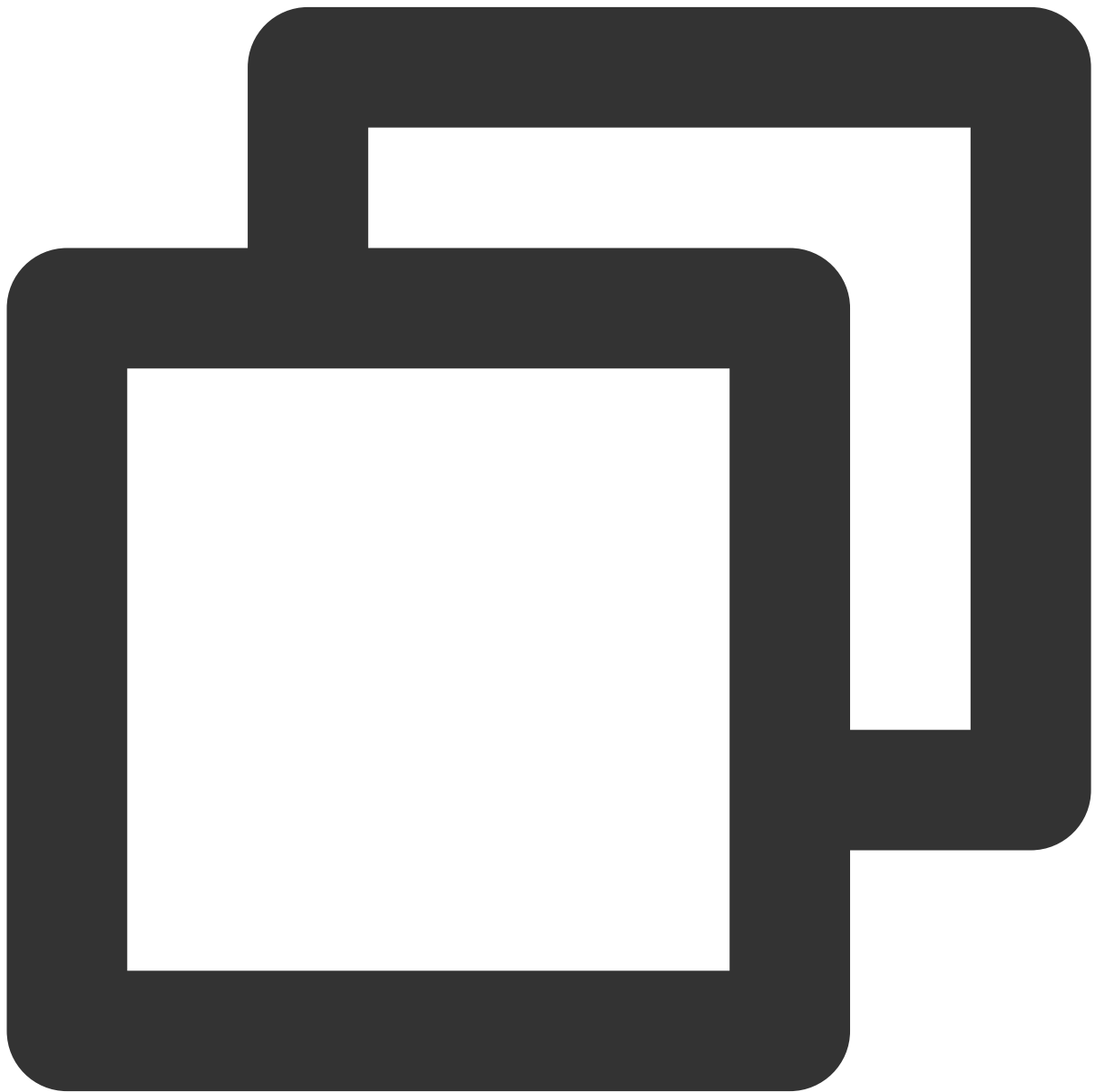
Calling this API will immediately modify the seat list. In cases where the room owner needs listeners' permission to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call `pickSeat`.

## kickSeat

This API is used to remove a speaker (called by room owner).

### Note

After a speaker is removed from a seat, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.



```
- (void)kickSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback N
```

The parameters are described below:

| Parameter | Type           | Description                                        |
|-----------|----------------|----------------------------------------------------|
| seatIndex | int            | The number of the seat to remove the speaker from. |
| callback  | ActionCallback | The callback for the operation.                    |

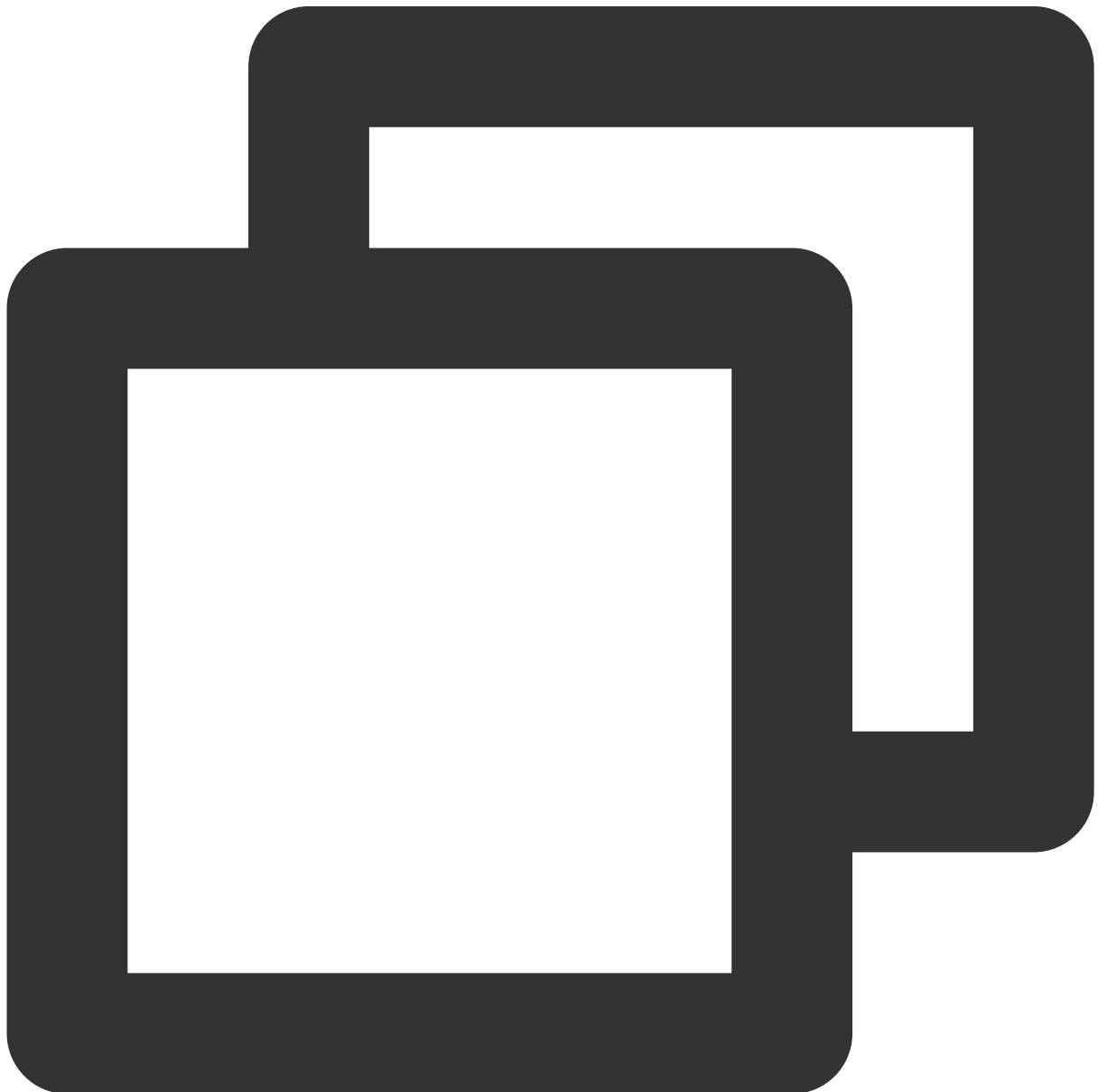
Calling this API will immediately modify the seat list.

## **muteSeat**

This API is used to mute/unmute a seat (called by room owner).

### **Note**

After a seat is muted/unmuted, all members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.



```
- (void)muteSeat:(NSInteger)seatIndex isMute:(BOOL)isMute callback:(ActionCallback
```

The parameters are described below:

| Parameter | Type           | Description                                           |
|-----------|----------------|-------------------------------------------------------|
| seatIndex | int            | The number of the seat to mute/unmute.                |
| isMute    | boolean        | <code>true</code> : Mute; <code>false</code> : Unmute |
| callback  | ActionCallback | The callback for the operation.                       |

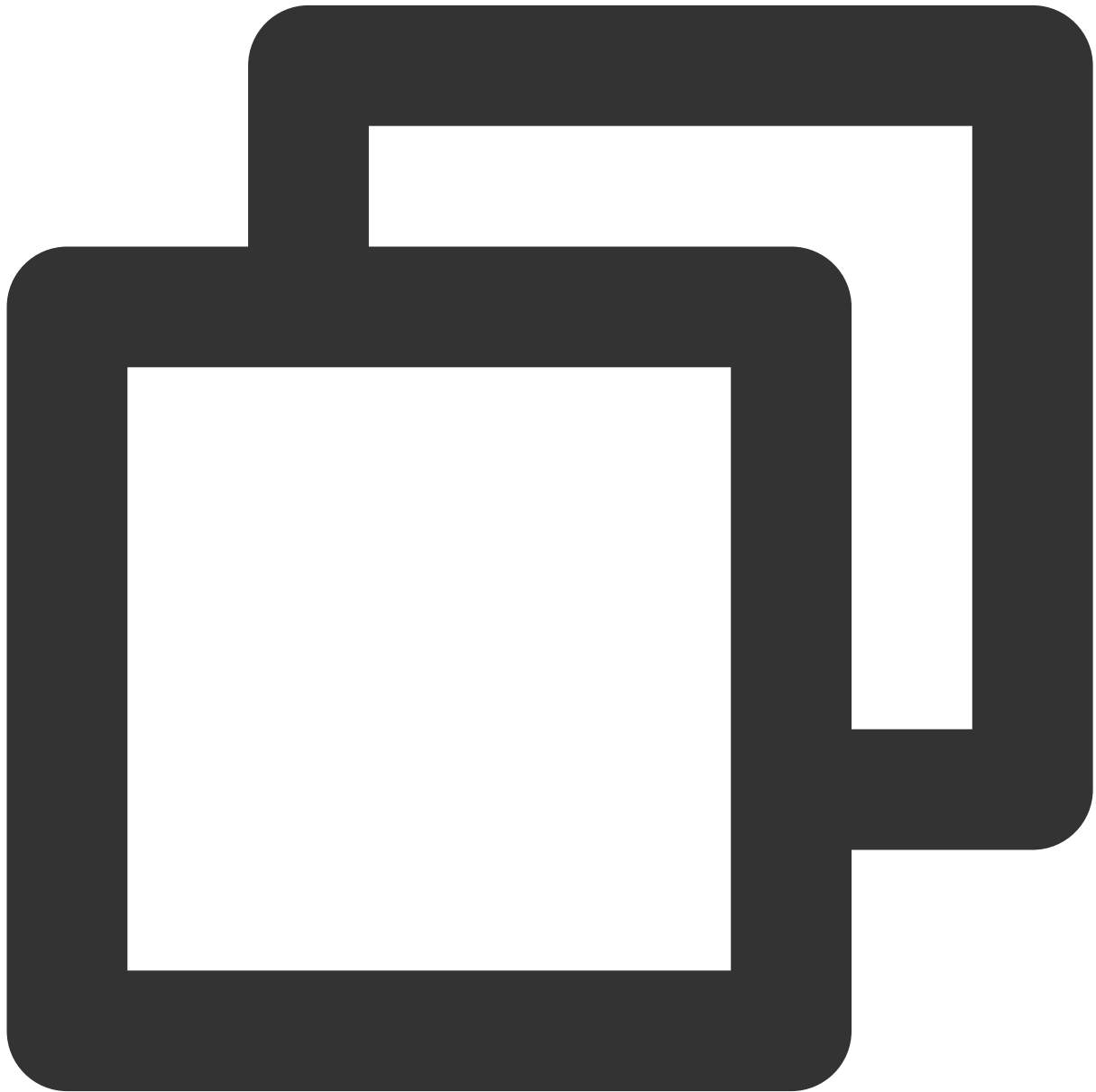
Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will call `muteAudio` to mute/unmute his or her audio.

## closeSeat

This API is used to block/unblock a seat (called by room owner).

### Note

After a seat is blocked/unblocked, all members in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.



```
- (void)closeSeat:(NSInteger)seatIndex isClose:(BOOL)isClose callback:(ActionCallba
```

The parameters are described below:

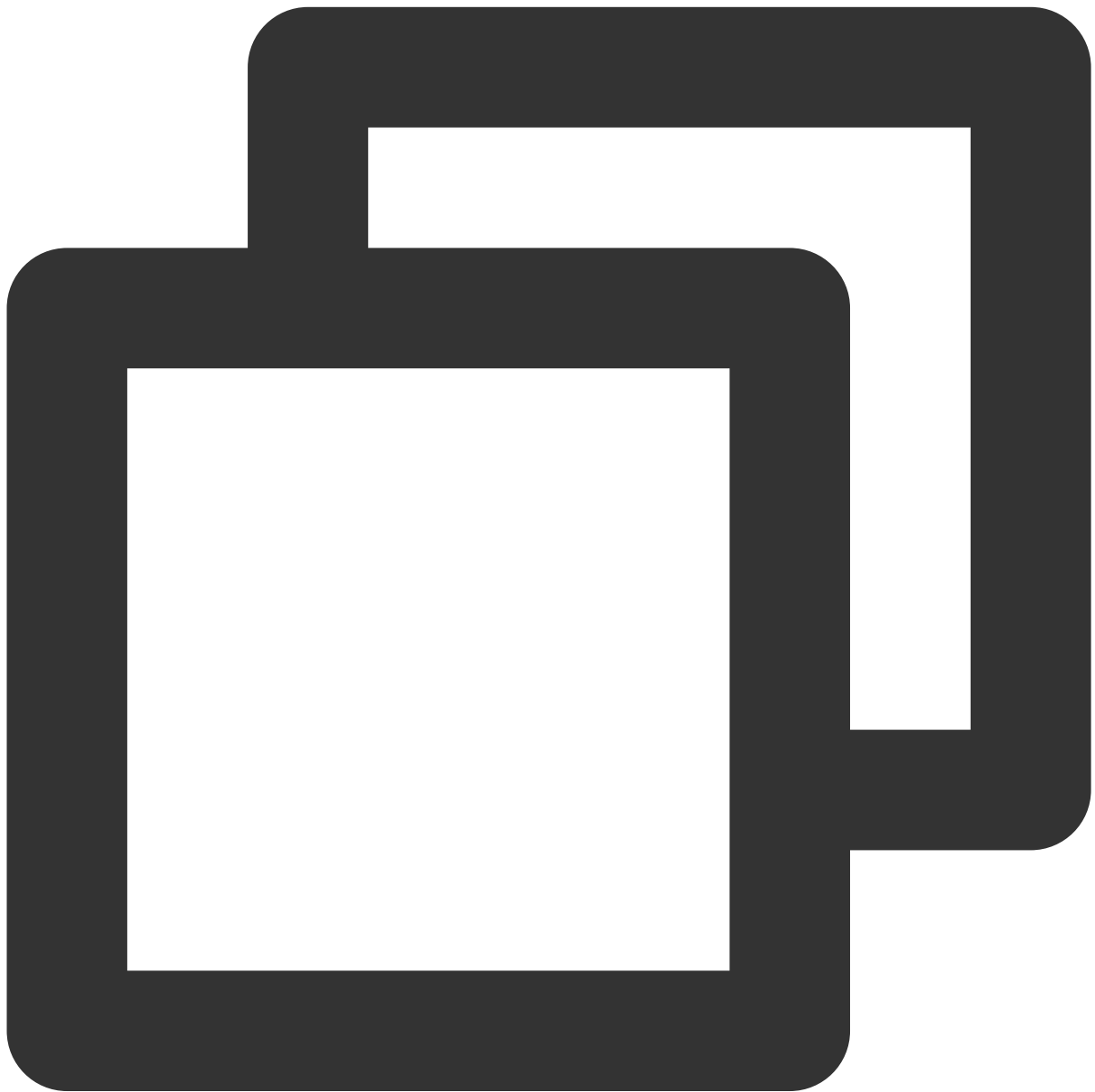
| Parameter | Type           | Description                                             |
|-----------|----------------|---------------------------------------------------------|
| seatIndex | int            | The number of the seat to block/unblock.                |
| isClose   | boolean        | <code>true</code> : Block; <code>false</code> : Unblock |
| callback  | ActionCallback | The callback for the operation.                         |

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will leave the seat.

## Local Audio APIs

### **startMicrophone**

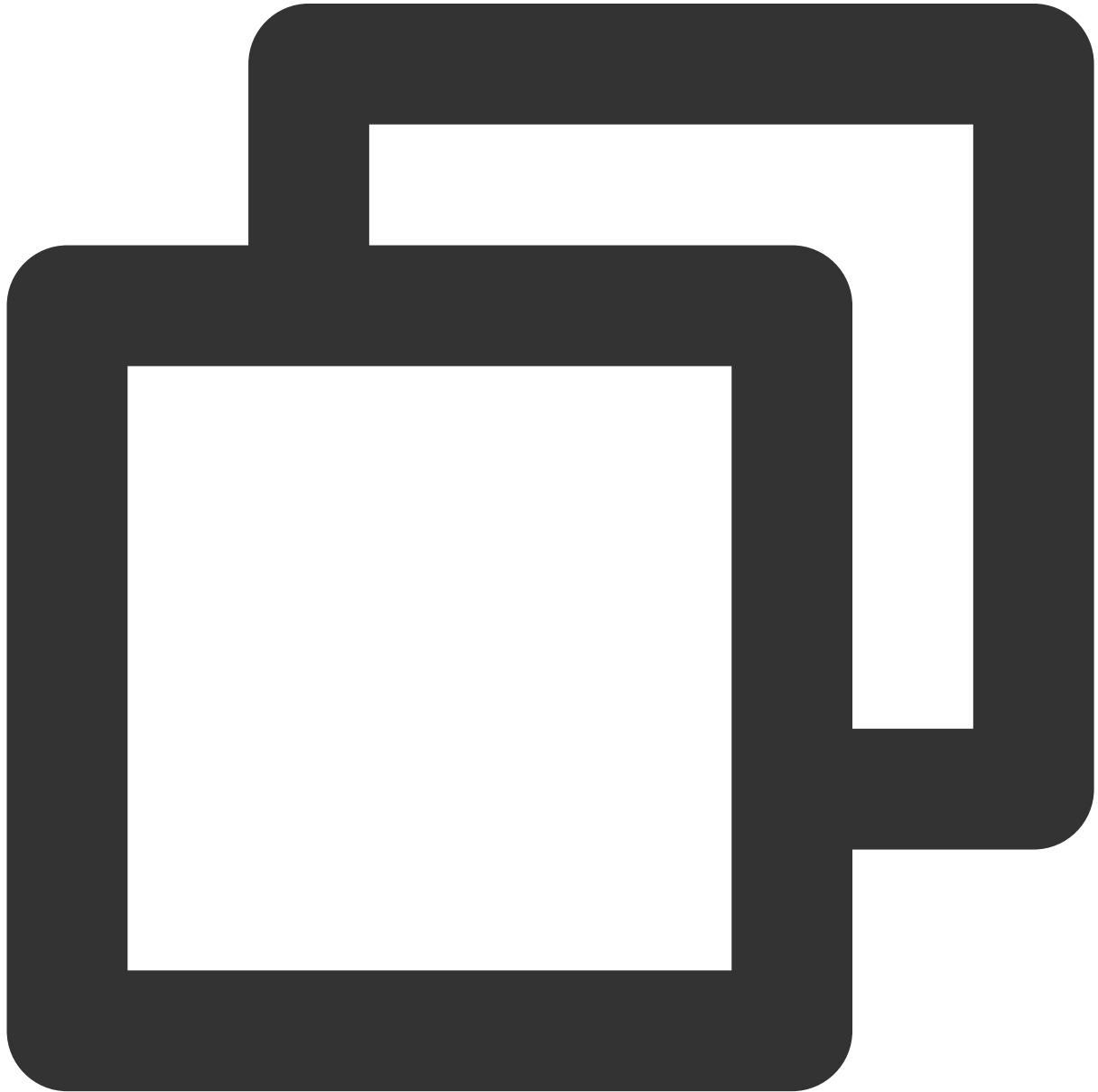
This API is used to start mic capturing.



```
- (void)startMicrophone;
```

## stopMicrophone

This API is used to stop mic capturing.

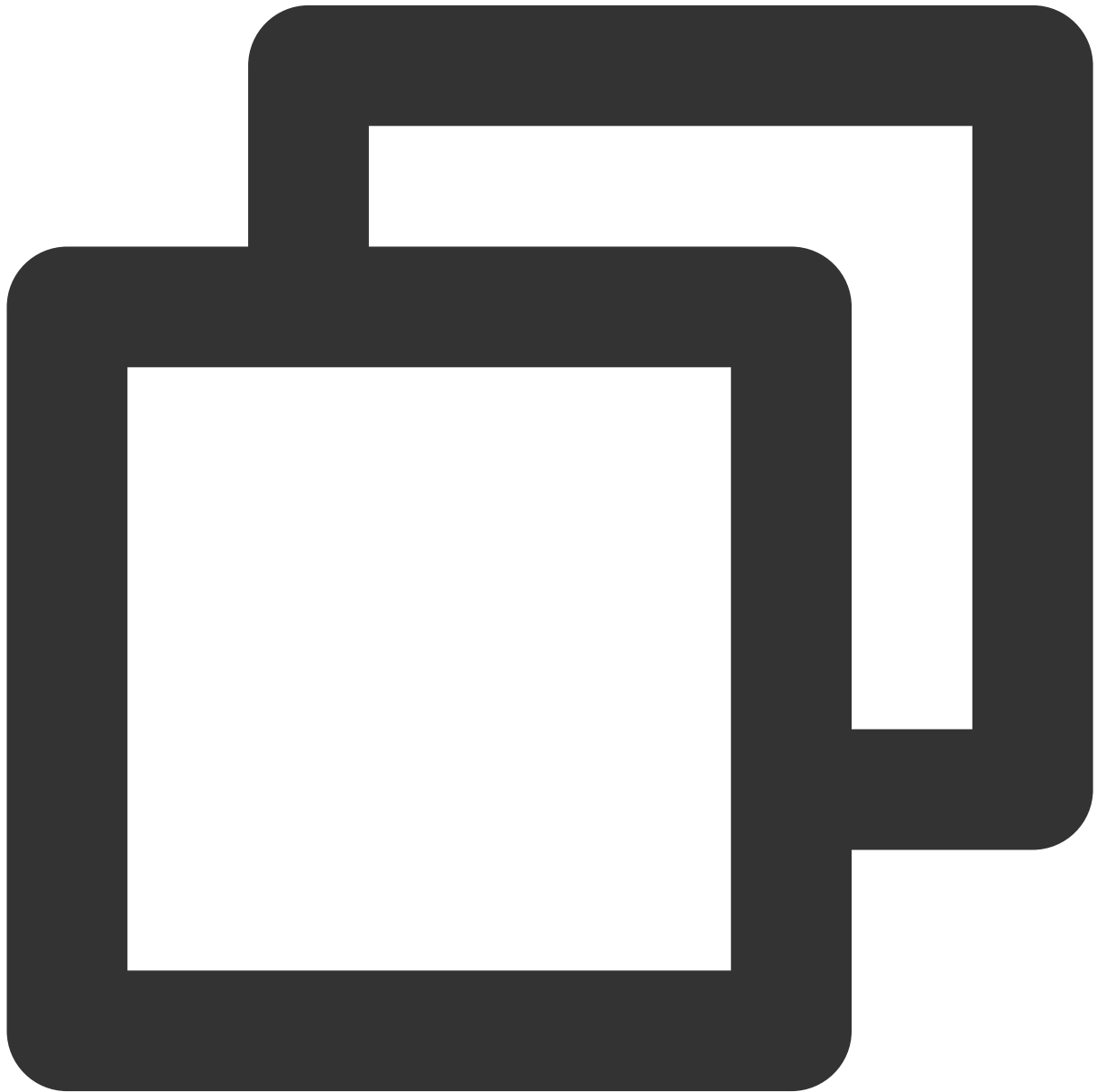


```
- (void)stopMicrophone;
```

## setAudioQuality

This API is used to set audio quality.





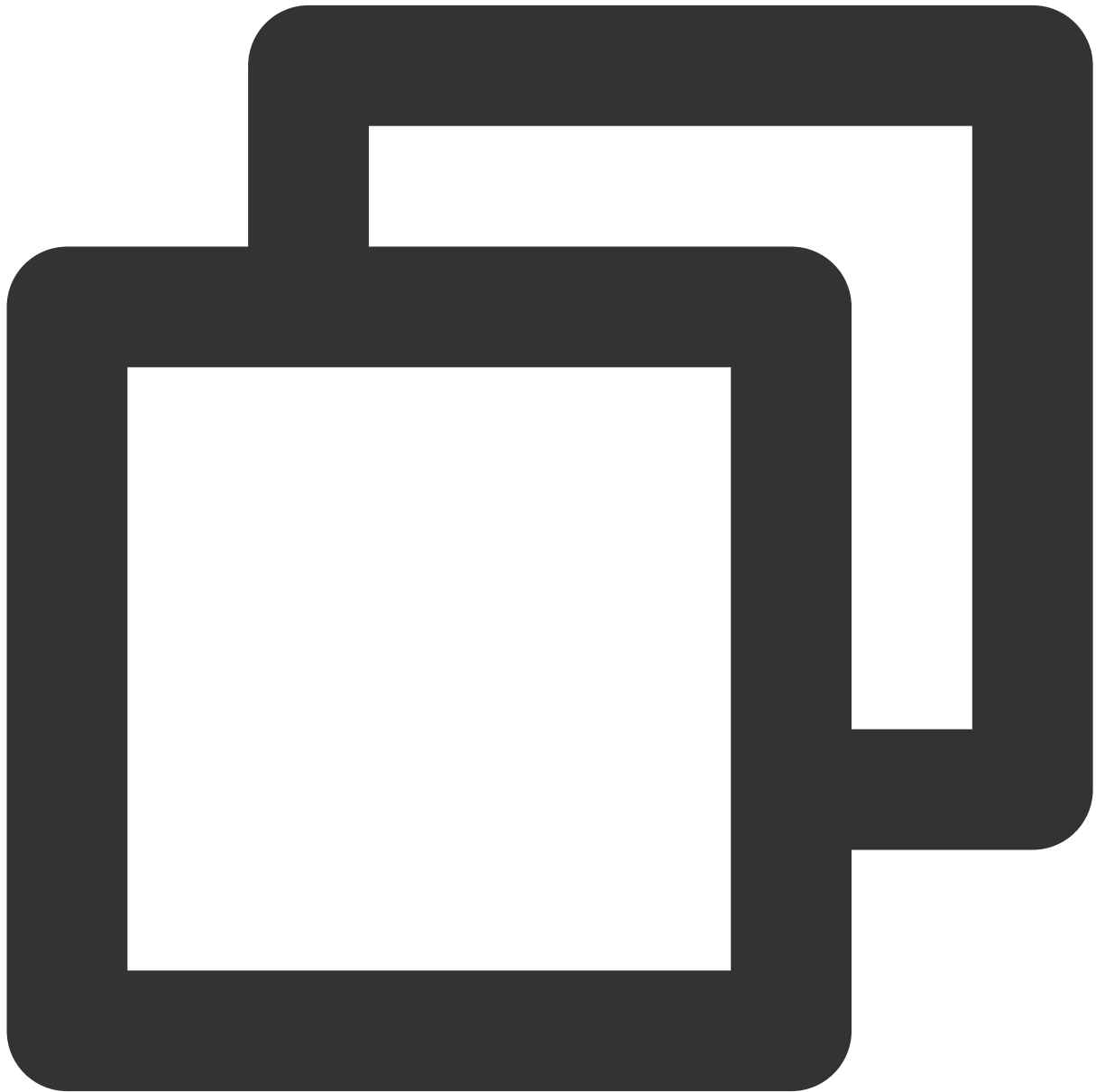
```
- (void)setAudioQuality:(NSInteger)quality NS_SWIFT_NAME(setAudioQuality(quantity:))
```

The parameters are described below:

| Parameter | Type | Description                                                                      |
|-----------|------|----------------------------------------------------------------------------------|
| quality   | int  | The audio quality. For more information, see <a href="#">setAudioQuality()</a> . |

## **muteLocalAudio**

This API is used to mute/unmute local audio.



```
- (void)muteLocalAudio:(BOOL)mute NS_SWIFT_NAME(muteLocalAudio(mute:));
```

The parameters are described below:

| Parameter | Type    | Description                                                                                   |
|-----------|---------|-----------------------------------------------------------------------------------------------|
| mute      | boolean | Whether to mute or unmute audio. For more information, see <a href="#">muteLocalAudio()</a> . |

## setSpeaker

This API is used to set whether to play sound from the device's speaker or receiver.



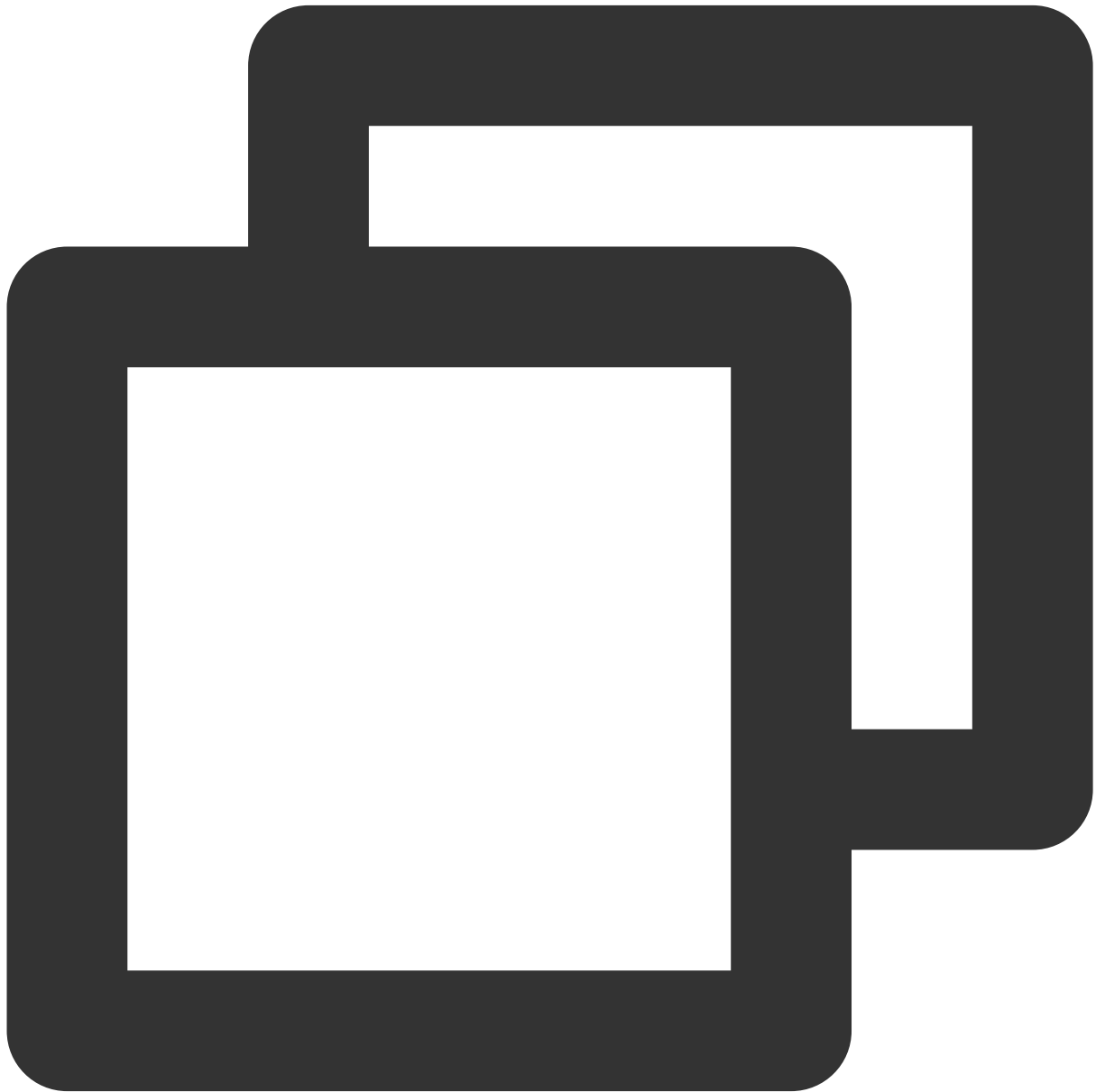
```
- (void)setSpeaker:(BOOL)userSpeaker NS_SWIFT_NAME(setSpeaker(userSpeaker:));
```

The parameters are described below:

| Parameter  | Type    | Description                                                |
|------------|---------|------------------------------------------------------------|
| useSpeaker | boolean | <code>true</code> : Speaker; <code>false</code> : Receiver |

## setAudioCaptureVolume

This API is used to set the mic capturing volume.



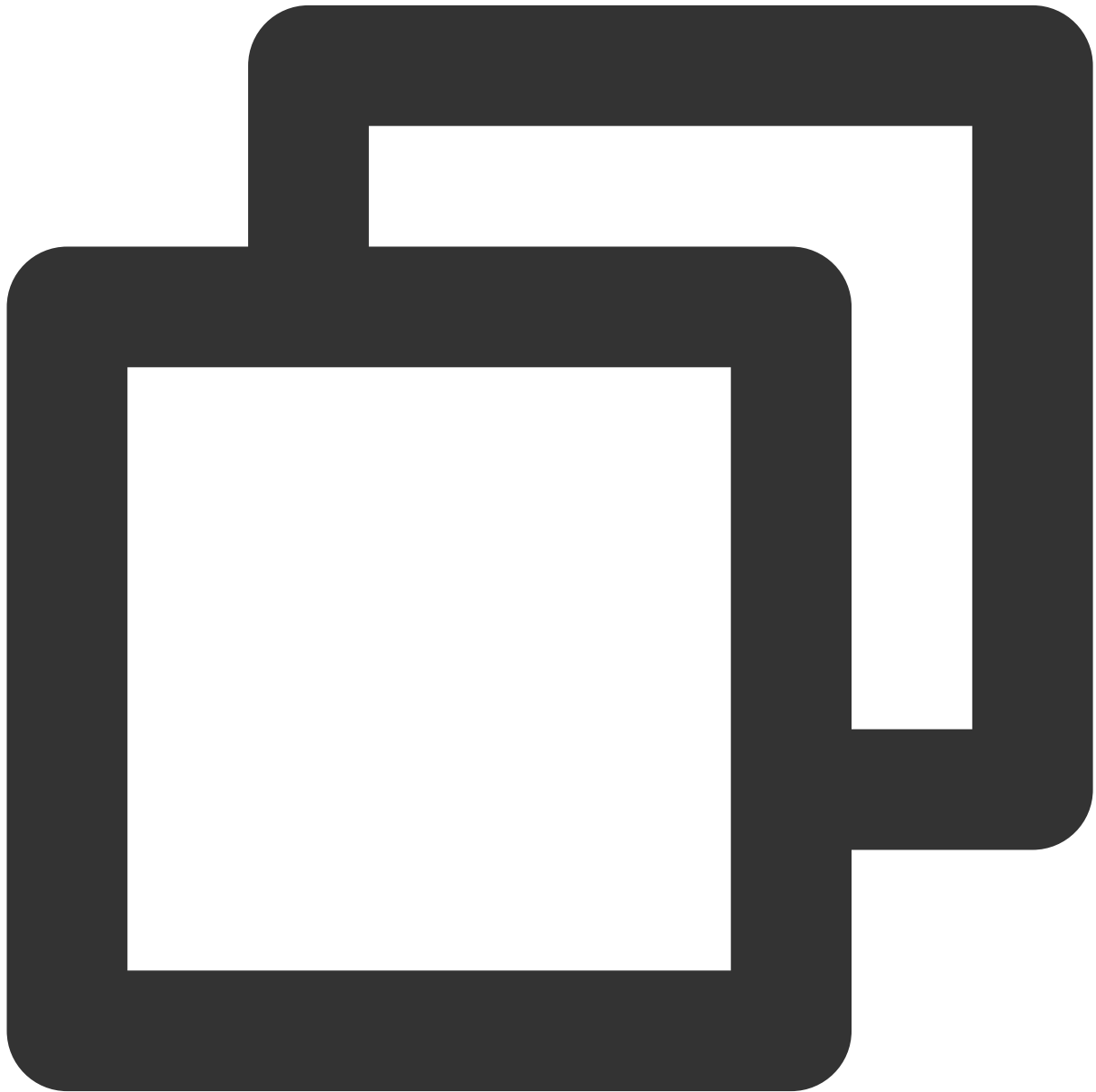
```
- (void)setAudioCaptureVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioCaptureVolume)
```

The parameters are described below:

| Parameter | Type | Description                                             |
|-----------|------|---------------------------------------------------------|
| volume    | int  | The capturing volume. Value range: 0-100 (default: 100) |

## setAudioPlayoutVolume

This API is used to set the playback volume.



```
- (void)setAudioPlayoutVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioPlayoutVolume)
```

The parameters are described below:

| Parameter | Type | Description                                            |
|-----------|------|--------------------------------------------------------|
| volume    | int  | The playback volume. Value range: 0-100 (default: 100) |

## **muteRemoteAudio**

This API is used to mute/unmute a specified user.



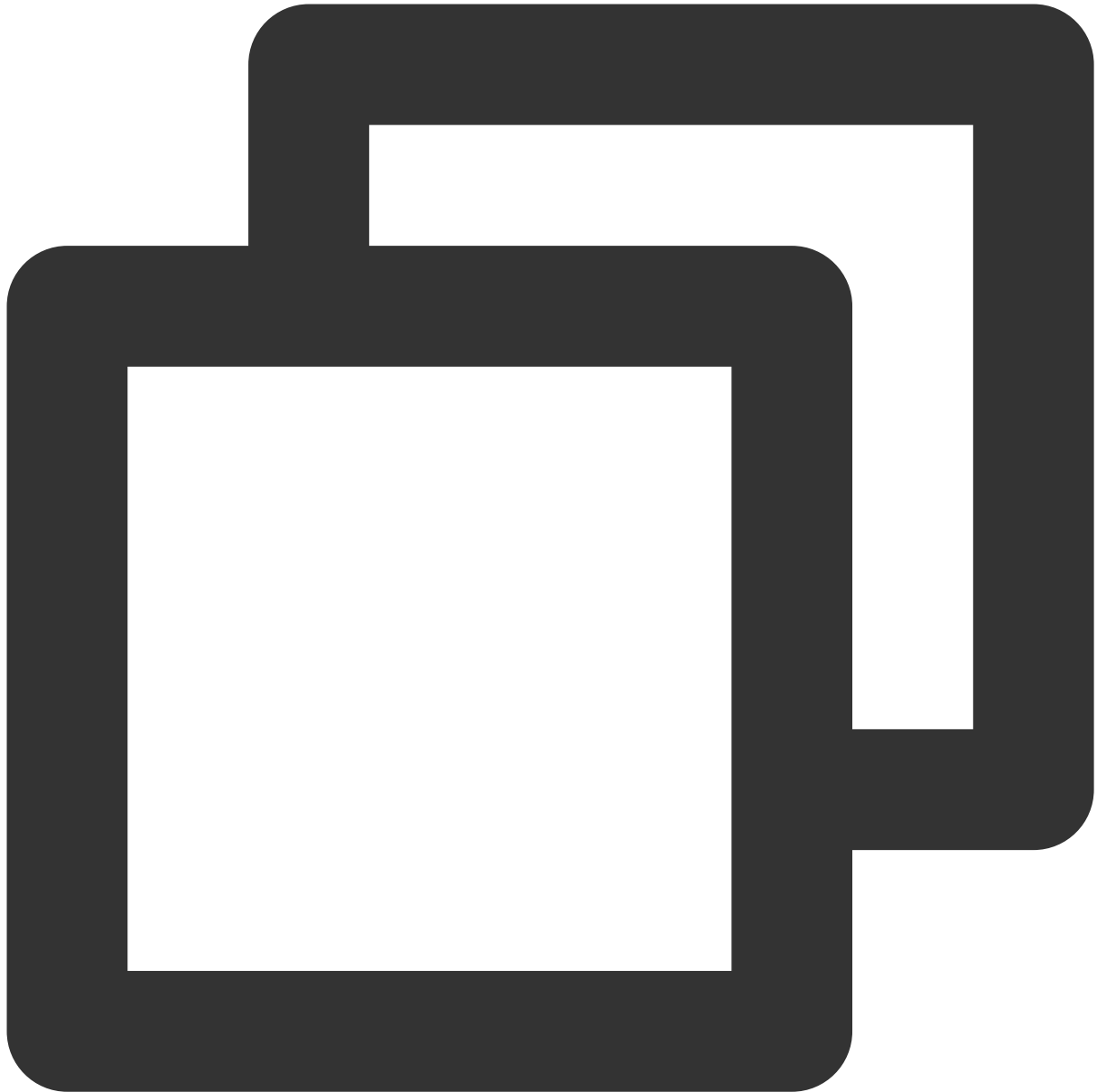
```
- (void)muteRemoteAudio:(NSString *)userId mute:(BOOL)mute NS_SWIFT_NAME(muteRemoteAudio)
```

The parameters are described below:

| Parameter | Type    | Description                                           |
|-----------|---------|-------------------------------------------------------|
| userId    | String  | The user ID.                                          |
| mute      | boolean | <code>true</code> : Mute; <code>false</code> : Unmute |

## **muteAllRemoteAudio**

This API is used to mute/unmute all users.



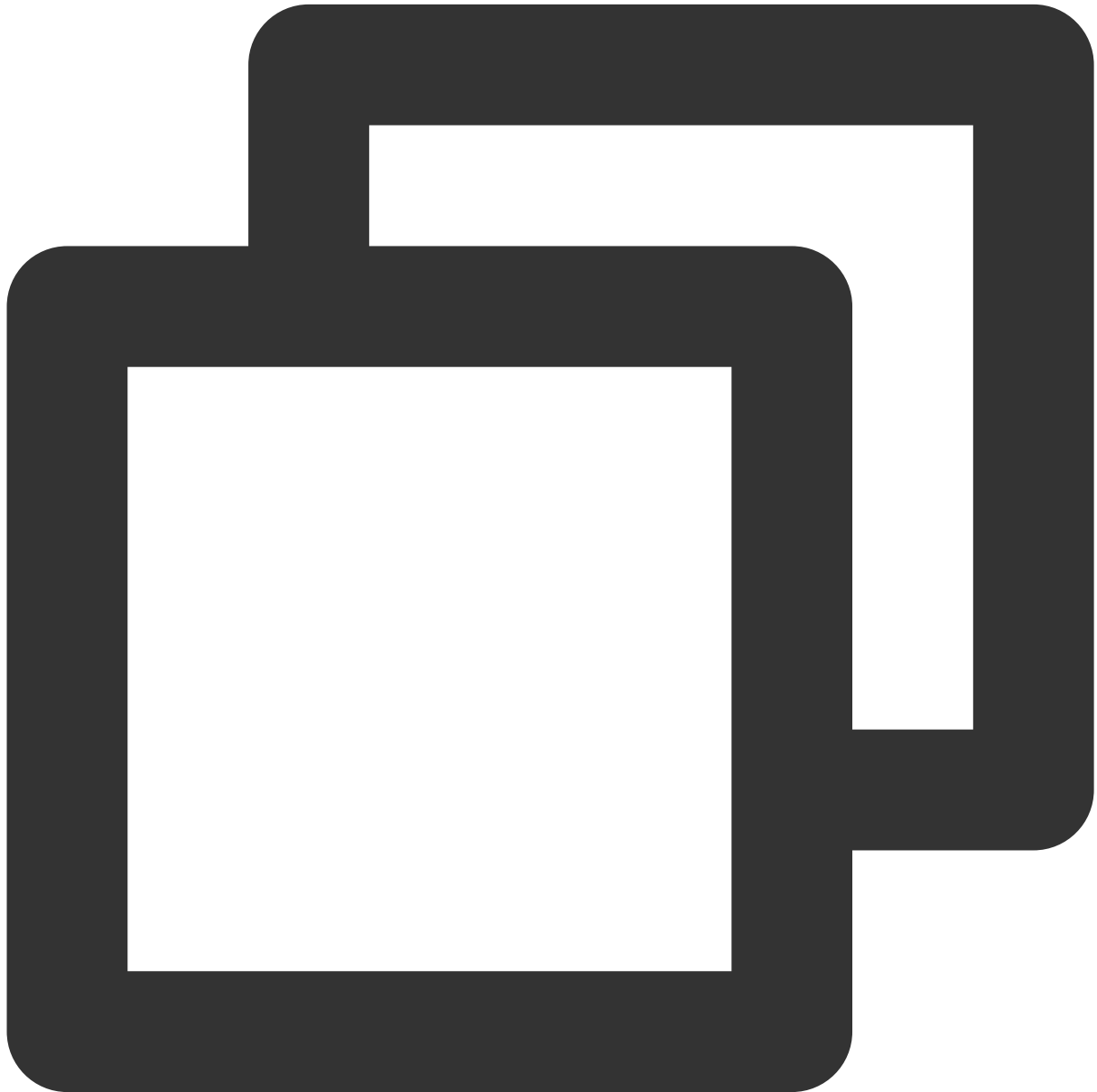
```
- (void)muteAllRemoteAudio:(BOOL)isMute NS_SWIFT_NAME(muteAllRemoteAudio(isMute:));
```

The parameters are described below:

| Parameter | Type    | Description                                           |
|-----------|---------|-------------------------------------------------------|
| isMute    | boolean | <code>true</code> : Mute; <code>false</code> : Unmute |

## setVoiceEarMonitorEnable

This API is used to enable/disable in-ear monitoring.



```
- (void)setVoiceEarMonitorEnable:(BOOL)enable NS_SWIFT_NAME(setVoiceEarMonitor(enable))
```

The parameters are described below:

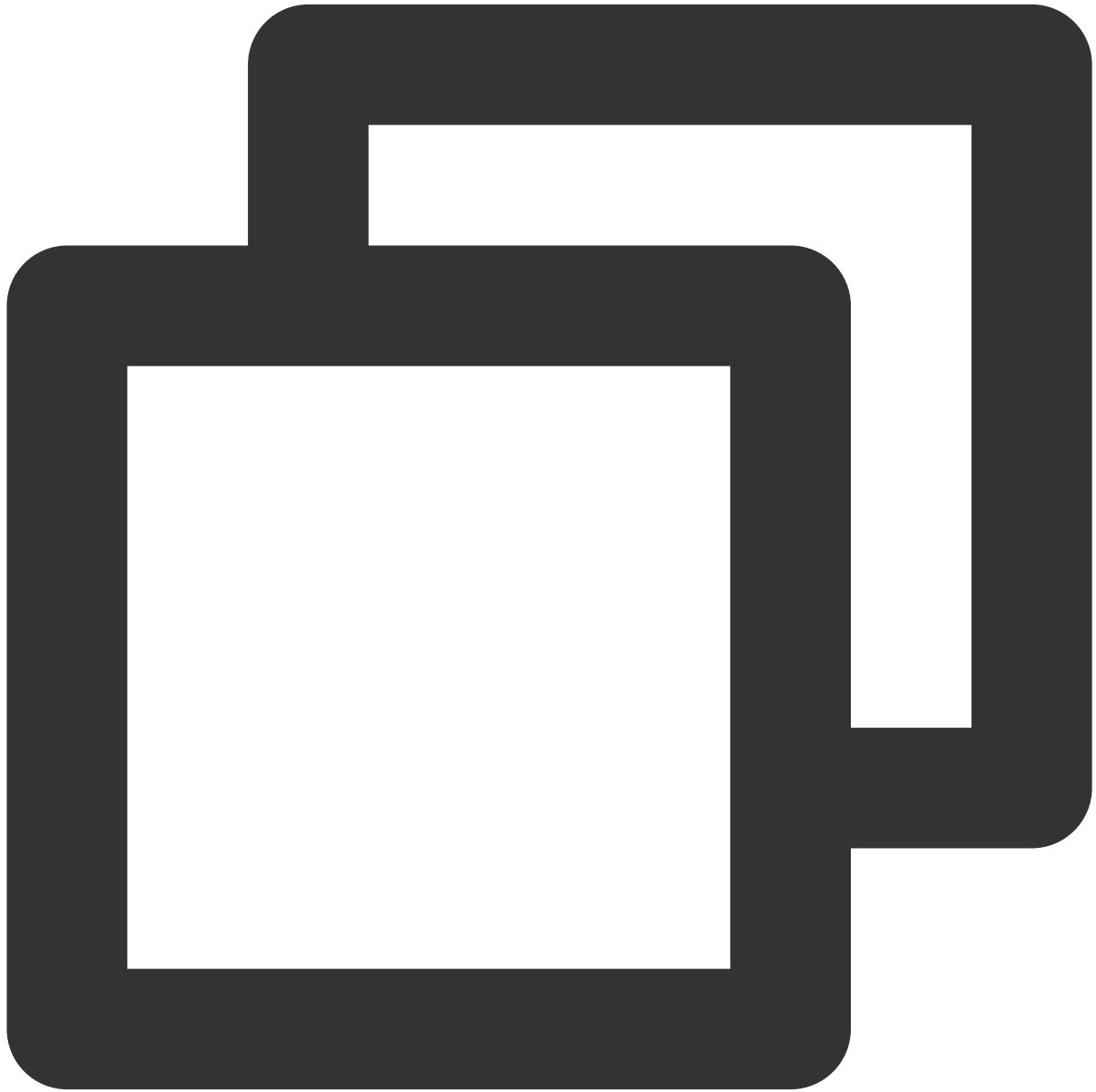
| Parameter | Type    | Description                                              |
|-----------|---------|----------------------------------------------------------|
| enable    | boolean | <code>true</code> : Enable; <code>false</code> : Disable |



## Background Music and Audio Effect APIs

### **getAudioEffectManager**

This API is used to get the background music and audio effect management object [TXAudioEffectManager](#).

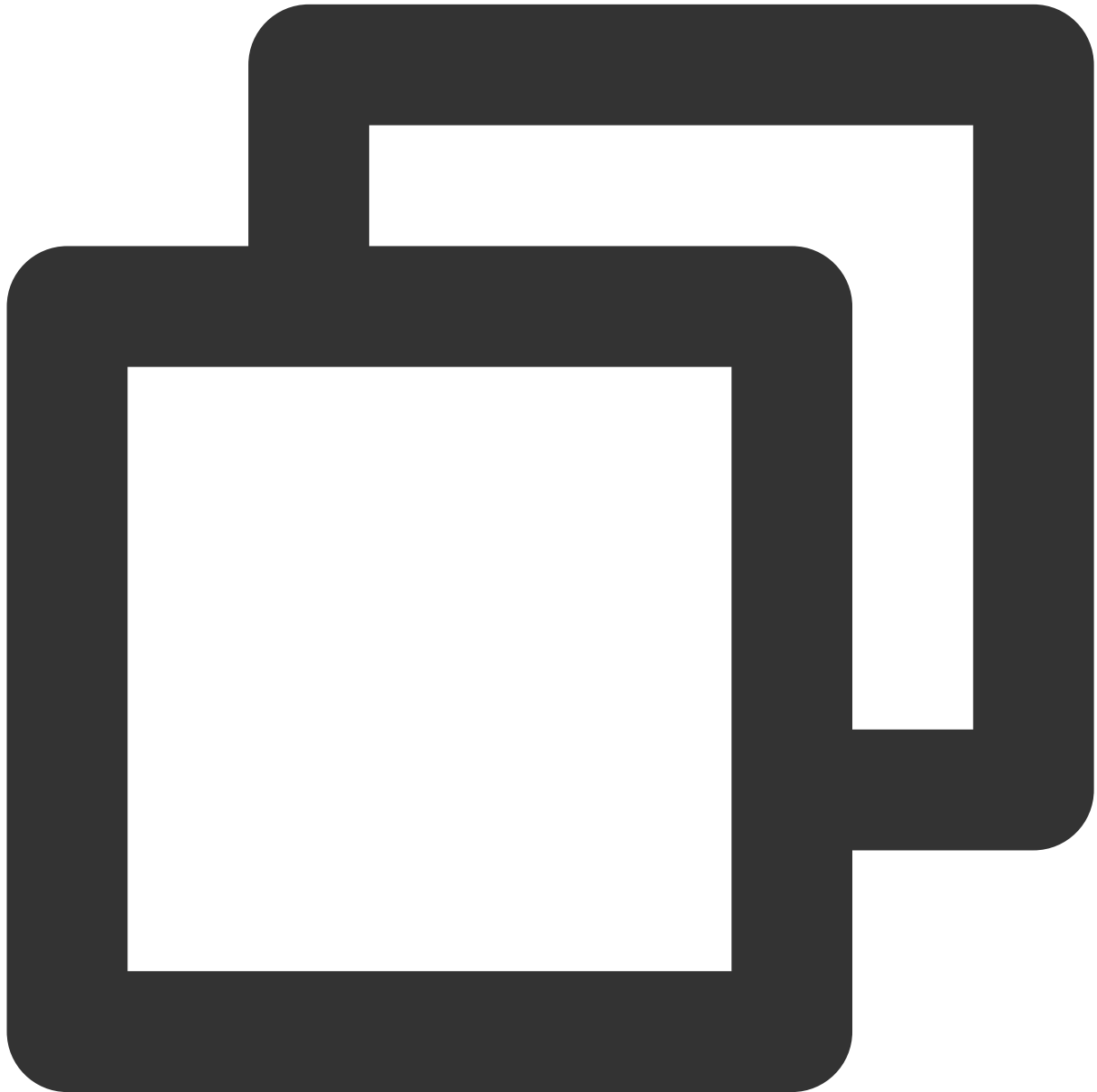


```
- (TXAudioEffectManager * _Nullable)getAudioEffectManager;
```

## Message Sending APIs

## sendRoomTextMsg

This API is used to broadcast a text chat message in a room, which is generally used for on-screen comments.



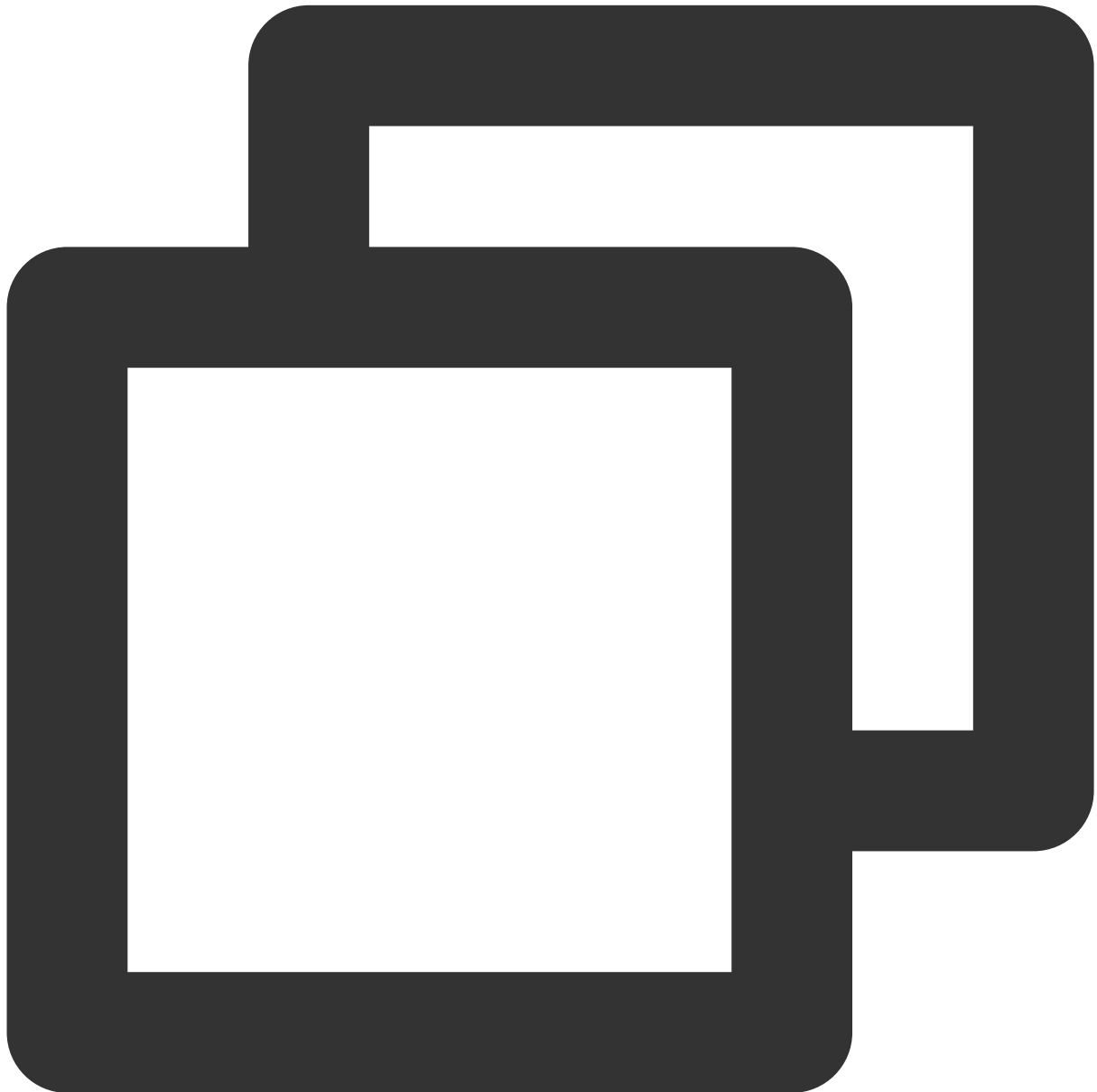
```
- (void)sendRoomTextMsg:(NSString *)message callback:(ActionCallback _Nullable)call
```

The parameters are described below:

| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| message   | String         | A text chat message.            |
| callback  | ActionCallback | The callback for the operation. |

## sendRoomCustomMsg

This API is used to send a custom text chat message.



```
- (void)sendRoomCustomMsg:(NSString *)cmd message:(NSString *)message callback:(Act
```

The parameters are described below:

| Parameter | Type   | Description                                                         |
|-----------|--------|---------------------------------------------------------------------|
| cmd       | String | A custom command word used to distinguish between different message |

|          |                |                                 |
|----------|----------------|---------------------------------|
|          |                | types.                          |
| message  | String         | A text chat message.            |
| callback | ActionCallback | The callback for the operation. |

## Invitation Signaling APIs

### **sendInvitation**

This API is used to send an invitation.



```
- (NSString *)sendInvitation:(NSString *)cmd
    userId:(NSString *)userId
    content:(NSString *)content
    callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendI
```

The parameters are described below:

| Parameter | Type   | Description                |
|-----------|--------|----------------------------|
| cmd       | String | Custom command of business |
|           |        |                            |

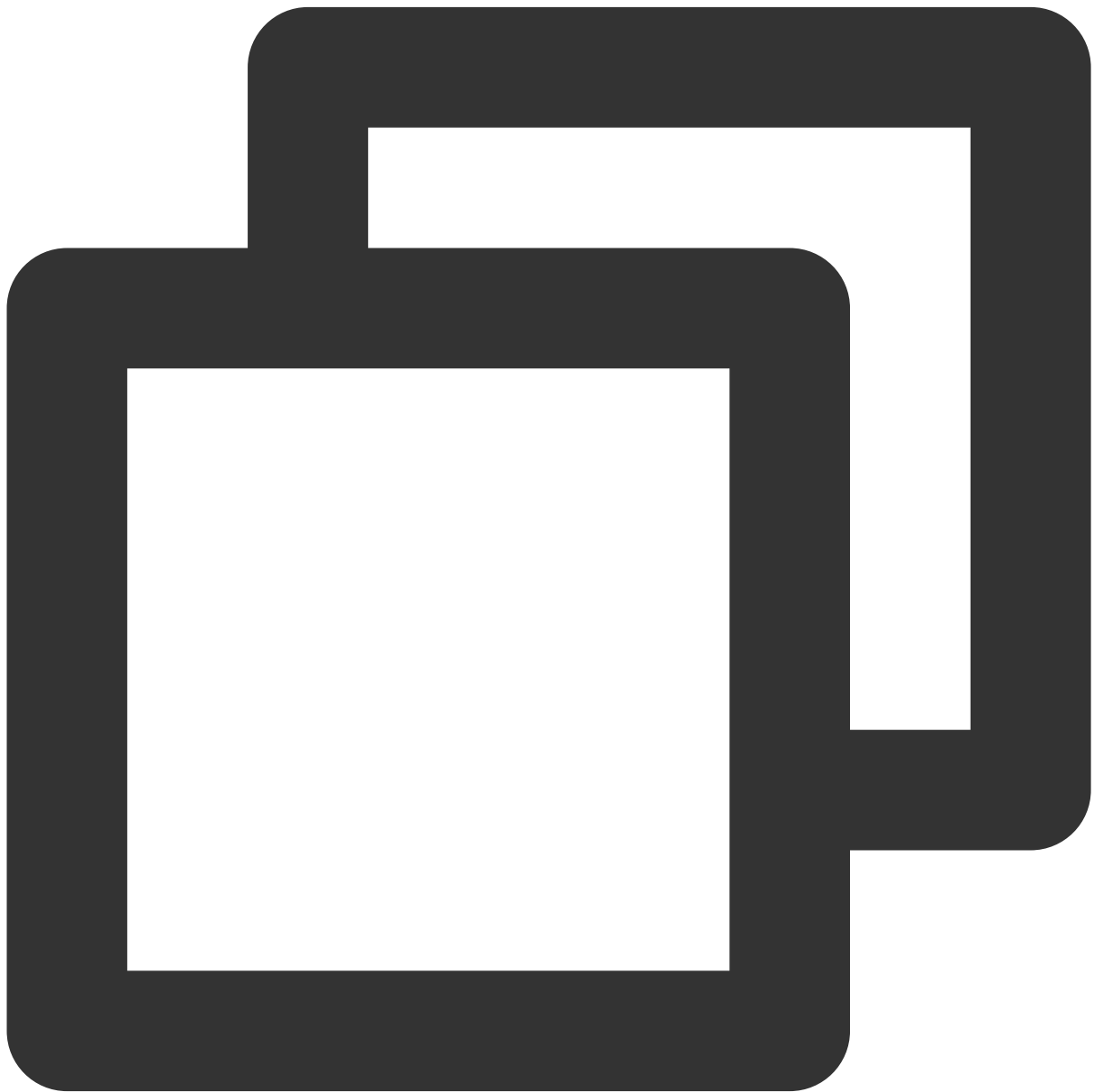
|          |                |                                 |
|----------|----------------|---------------------------------|
| userId   | String         | The user ID of the invitee.     |
| content  | String         | The content of the invitation.  |
| callback | ActionCallback | The callback for the operation. |

Response parameters:

| Parameter | Type   | Description        |
|-----------|--------|--------------------|
| inviteId  | String | The invitation ID. |

## acceptInvitation

This API is used to accept an invitation.



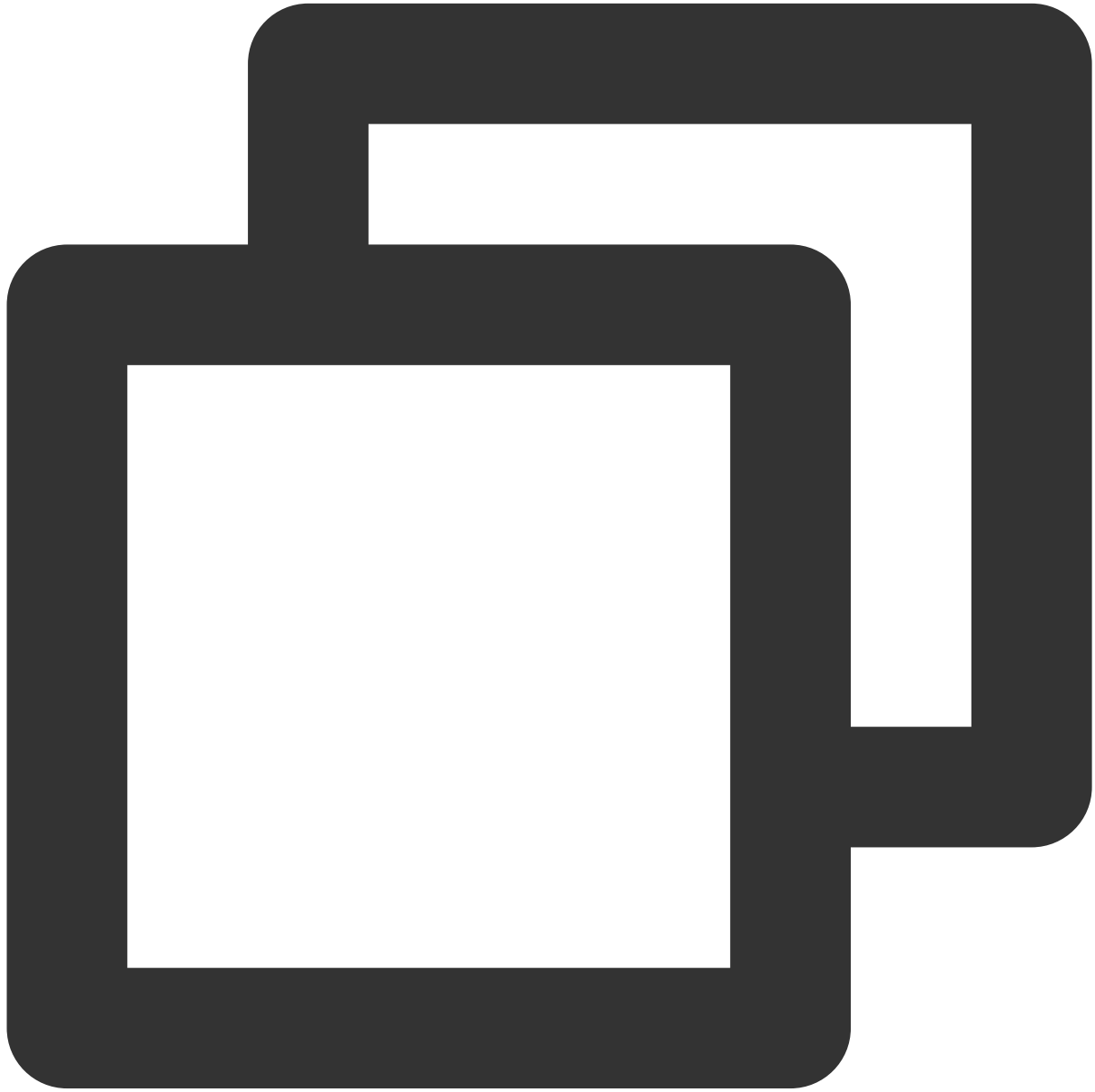
```
- (void)acceptInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)
```

The parameters are described below:

| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| id        | String         | The invitation ID.              |
| callback  | ActionCallback | The callback for the operation. |

## rejectInvitation

This API is used to decline an invitation.



```
- (void)rejectInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)
```

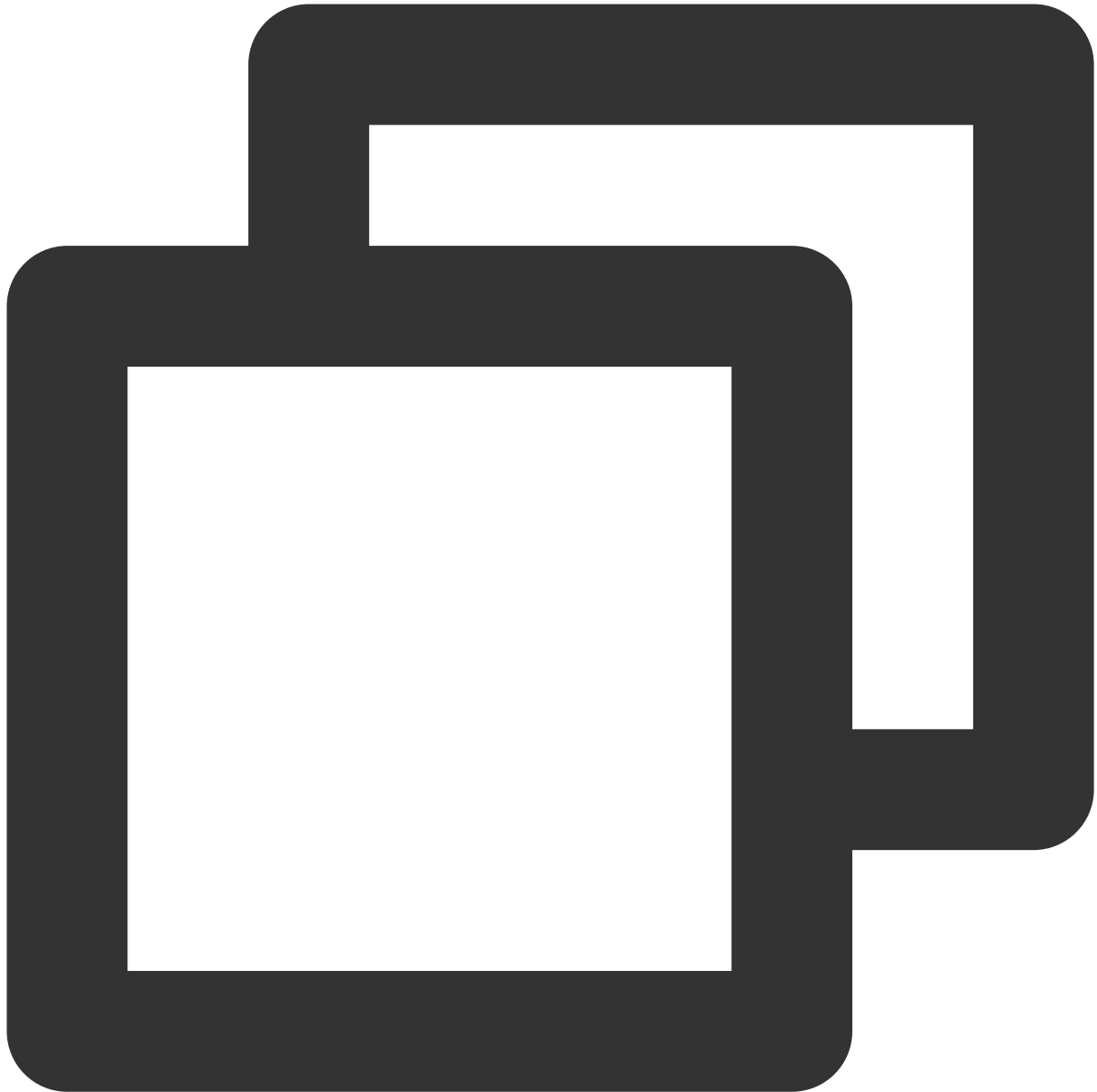
The parameters are described below:

| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| id        | String         | The invitation ID.              |
| callback  | ActionCallback | The callback for the operation. |



## cancelInvitation

This API is used to cancel an invitation.



```
- (void)cancelInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)
```

The parameters are described below:

| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| id        | String         | The invitation ID.              |
| callback  | ActionCallback | The callback for the operation. |

## TRTCKaraokeRoomDelegate Event Callback APIs

### Common Event Callback APIs

#### **onError**

Callback for error.

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users depending if necessary.



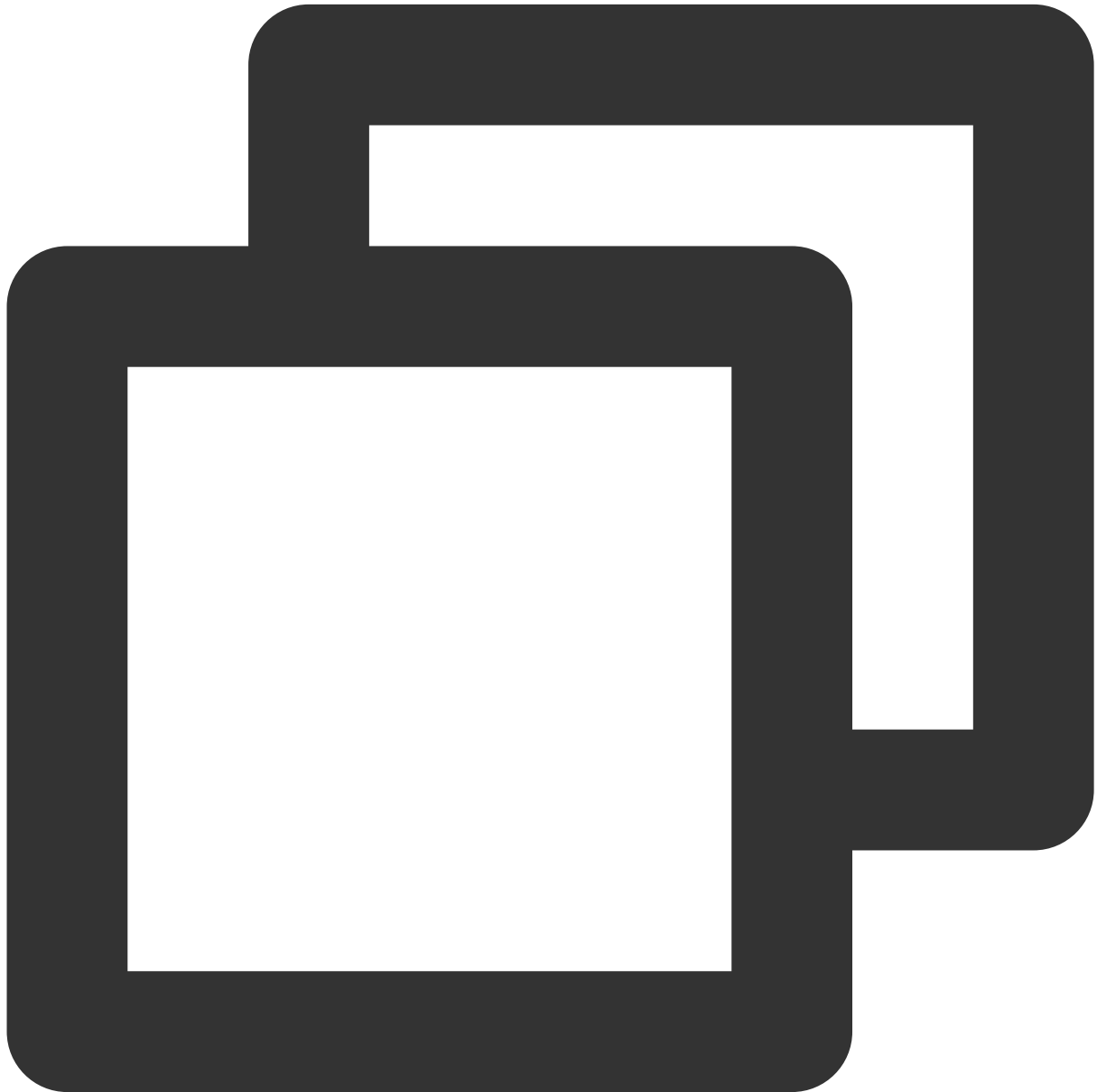
```
- (void)onError:(int)code  
    message:(NSString*)message  
NS_SWIFT_NAME(onError(code:message:));
```

The parameters are described below:

| Parameter | Type   | Description        |
|-----------|--------|--------------------|
| code      | int    | The error code.    |
| message   | String | The error message. |

## onWarning

Callback for warning.



```
- (void)onWarning:(int)code  
                message:(NSString *)message  
NS_SWIFT_NAME(onWarning(code:message:));
```

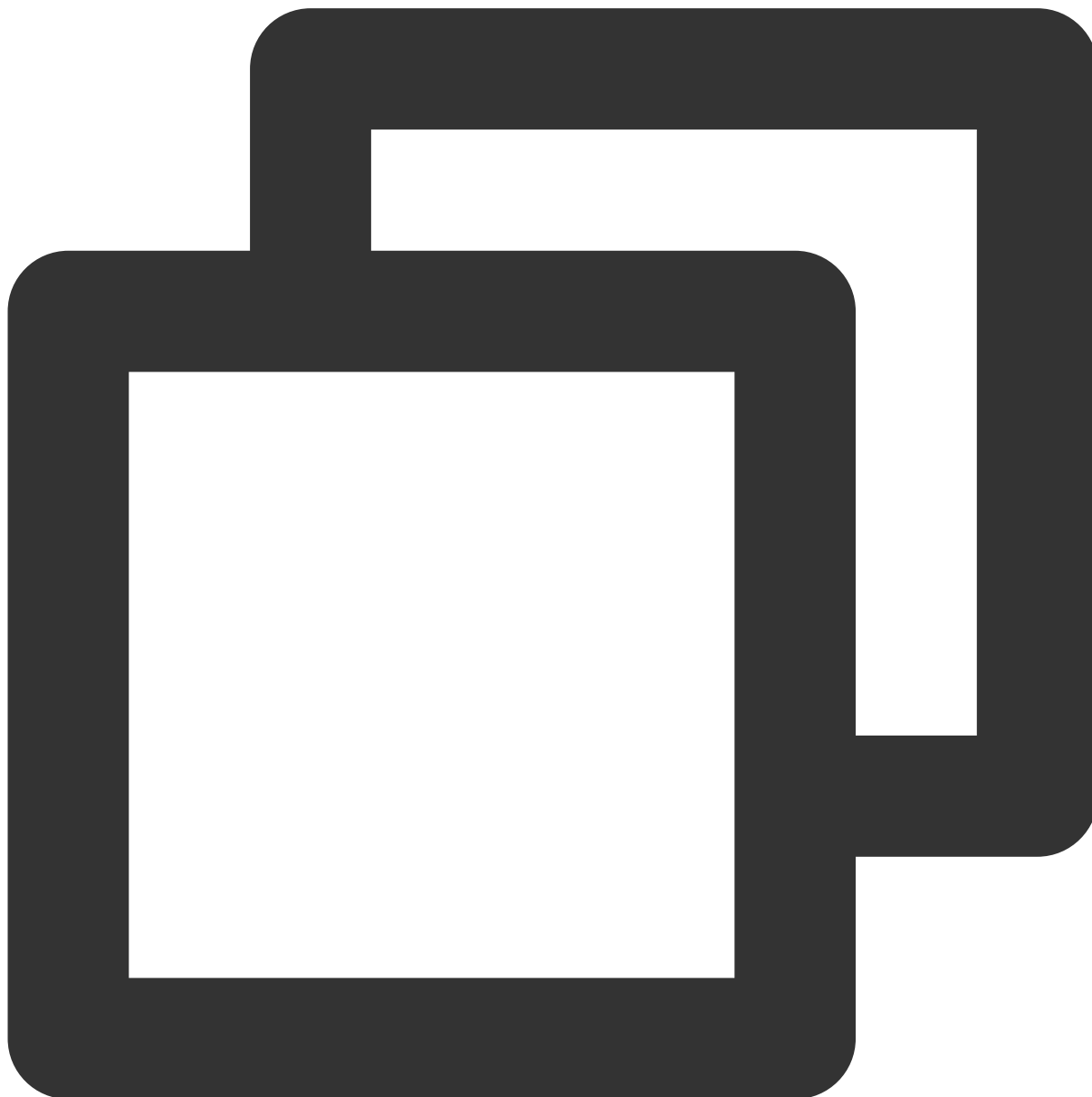
The parameters are described below:

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

|         |        |                 |
|---------|--------|-----------------|
| code    | int    | Error code      |
| message | String | Warning message |

## onDebugLog

Callback for log.



```
- (void)onDebugLog:(NSString *)message  
NS_SWIFT_NAME(onDebugLog(message:));
```

The parameters are described below:

| Parameter | Type   | Description     |
|-----------|--------|-----------------|
| message   | String | Log information |

## Room Event Callback APIs

### **onRoomDestroy**

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.



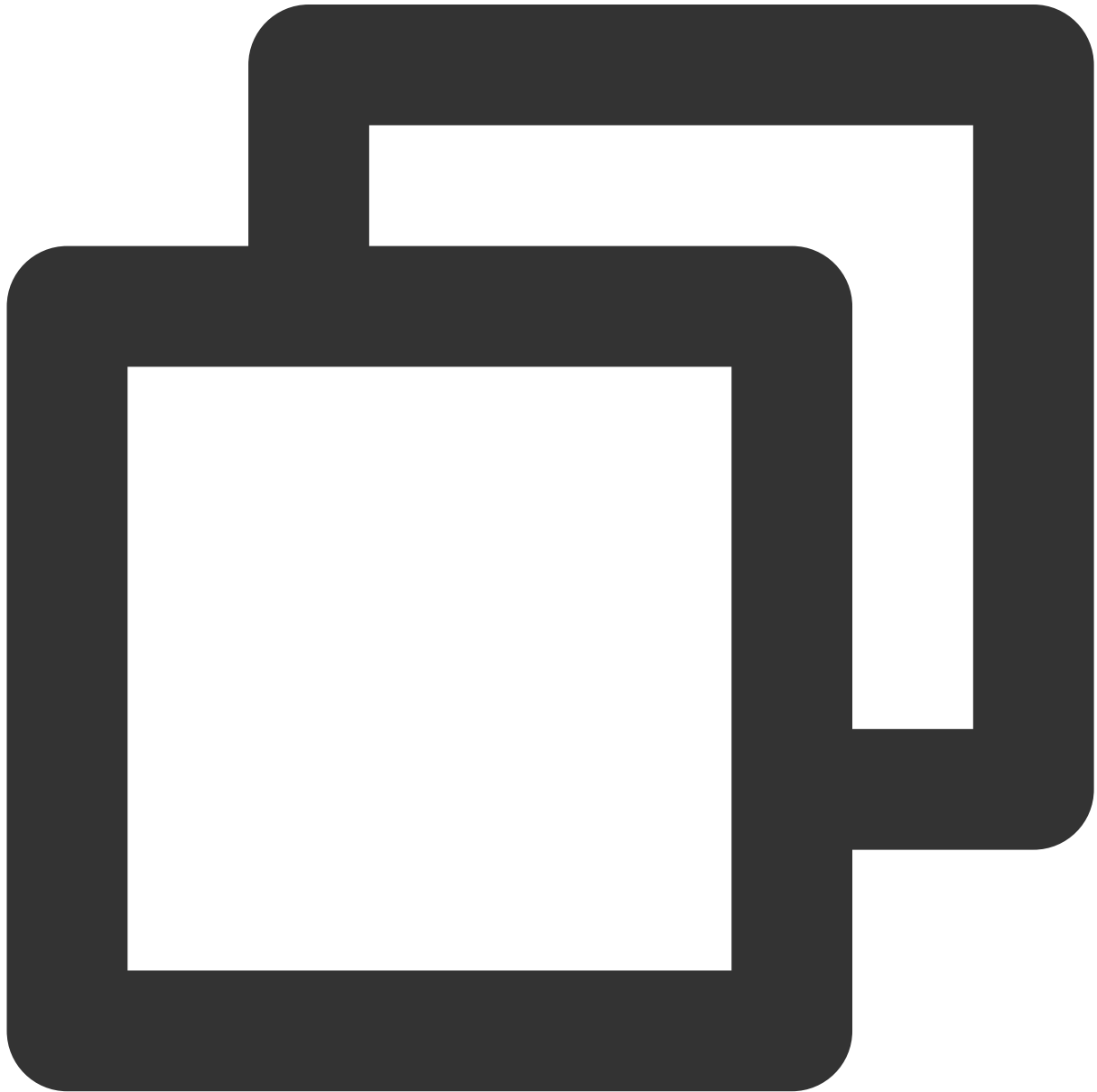
```
- (void)onRoomDestroy:(NSString *)message  
NS_SWIFT_NAME(onRoomDestroy(message:)) ;
```

The parameters are described below:

| Parameter | Type   | Description          |
|-----------|--------|----------------------|
| message   | String | Callback information |

## onRoomInfoChange

Callback for change of room information. This callback is sent after successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.



```
- (void)onRoomInfoChange:(KaraokeInfo *)roomInfo  
NS_SWIFT_NAME(onRoomInfoChange(roomInfo:));
```

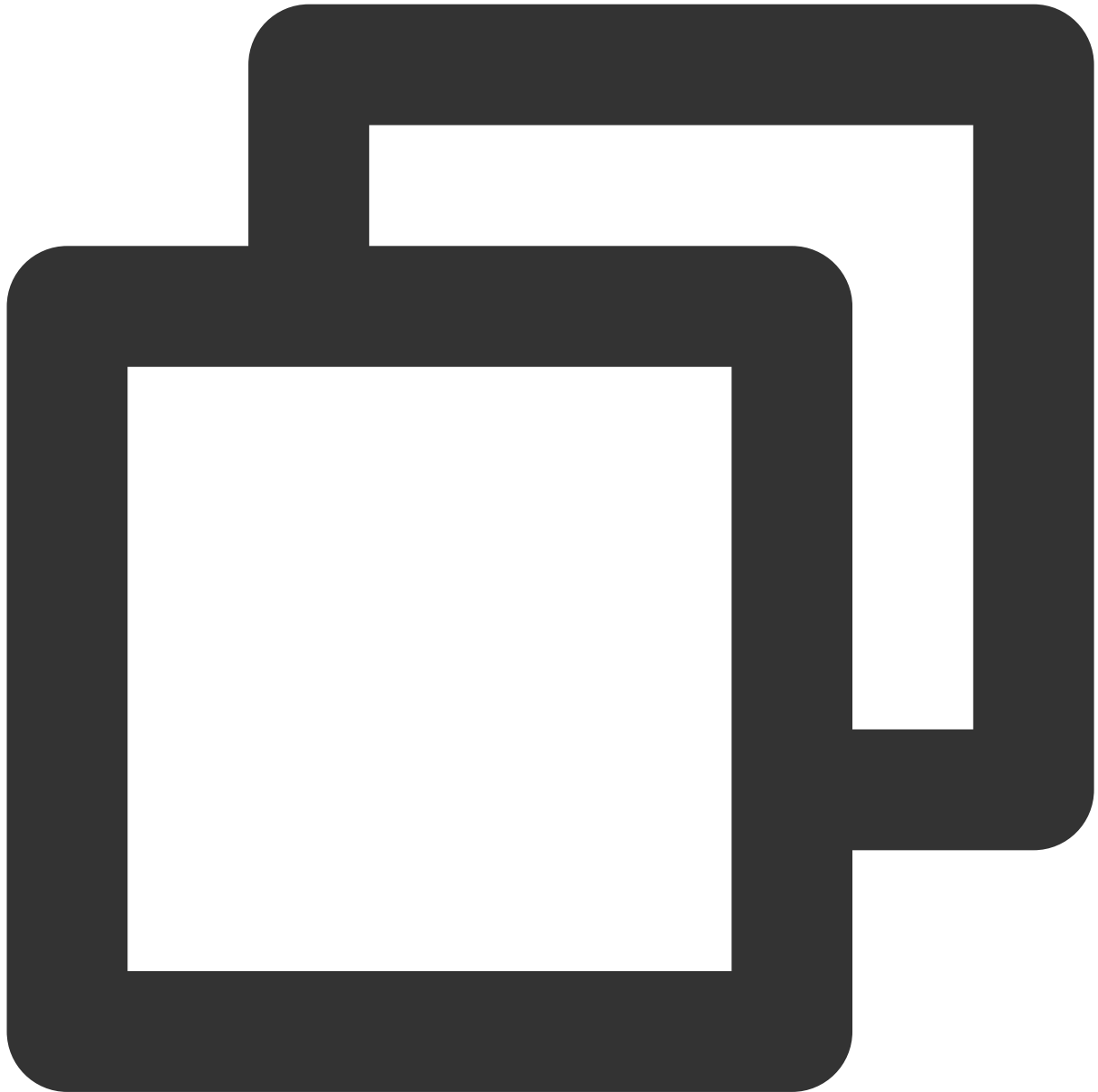
The parameters are described below:

| Parameter | Type     | Description      |
|-----------|----------|------------------|
| roomInfo  | RoomInfo | Room information |



## onUserMicrophoneMute

Callback of whether a user's mic is muted. When a user calls `muteLocalAudio`, all members in the room will receive this callback.



```
- (void)onUserMicrophoneMute:(NSString *)userId mute:(BOOL)mute  
NS_SWIFT_NAME(onUserMicrophoneMute(userId:mute:));
```

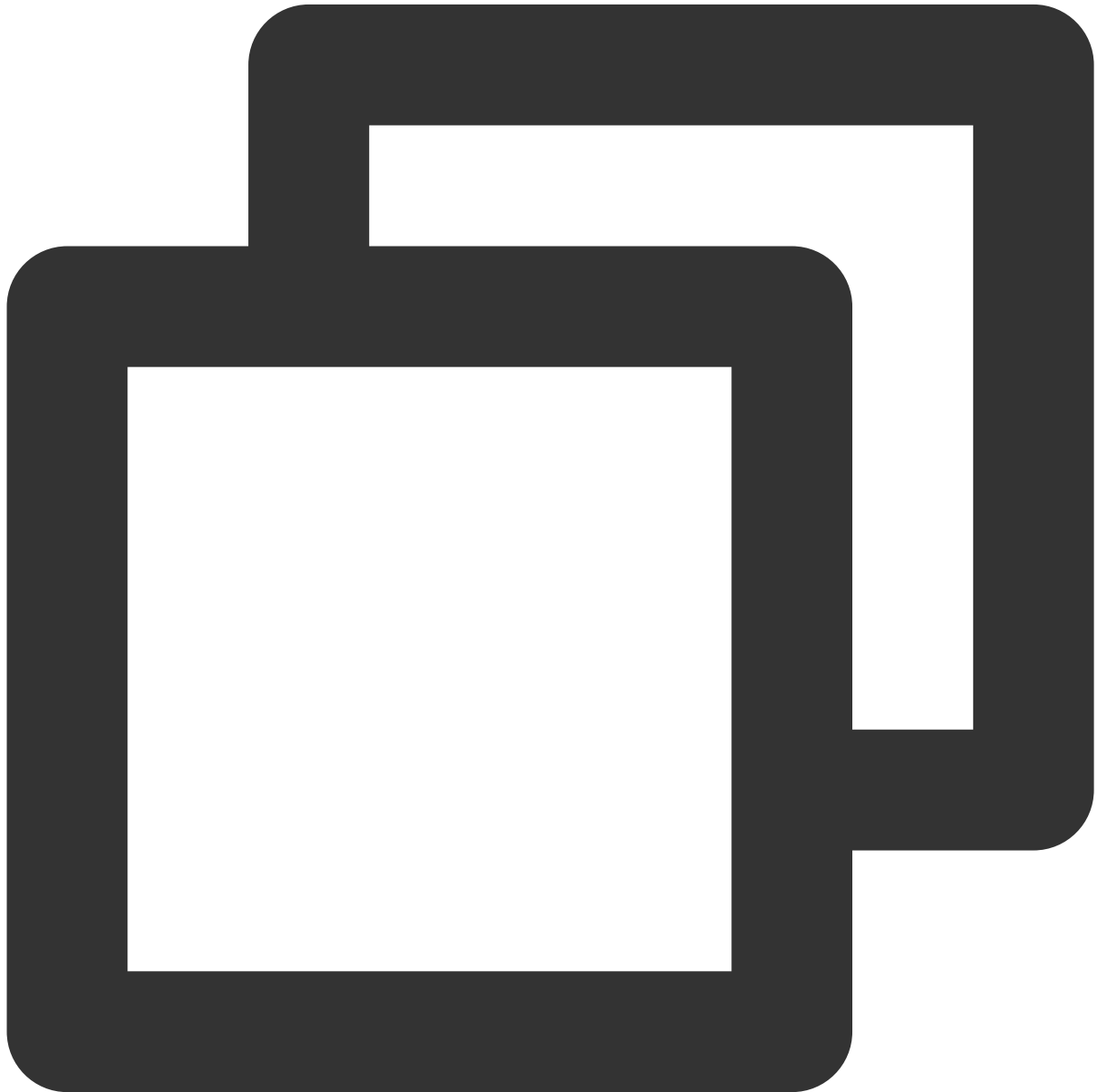
The parameters are described below:

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

|        |         |                            |
|--------|---------|----------------------------|
| userId | String  | The User ID.               |
| mute   | boolean | Volume. Value range: 0-100 |

## onUserVolumeUpdate

Notification to all members of the volume after the volume reminder is enabled.



```
- (void)onUserVolumeUpdate:(NSArray<TRTCVolumeInfo *> *)userVolumes totalVolume:(NS  
NS_SWIFT_NAME(onUserVolumeUpdate(userVolumes:totalVolume:));
```

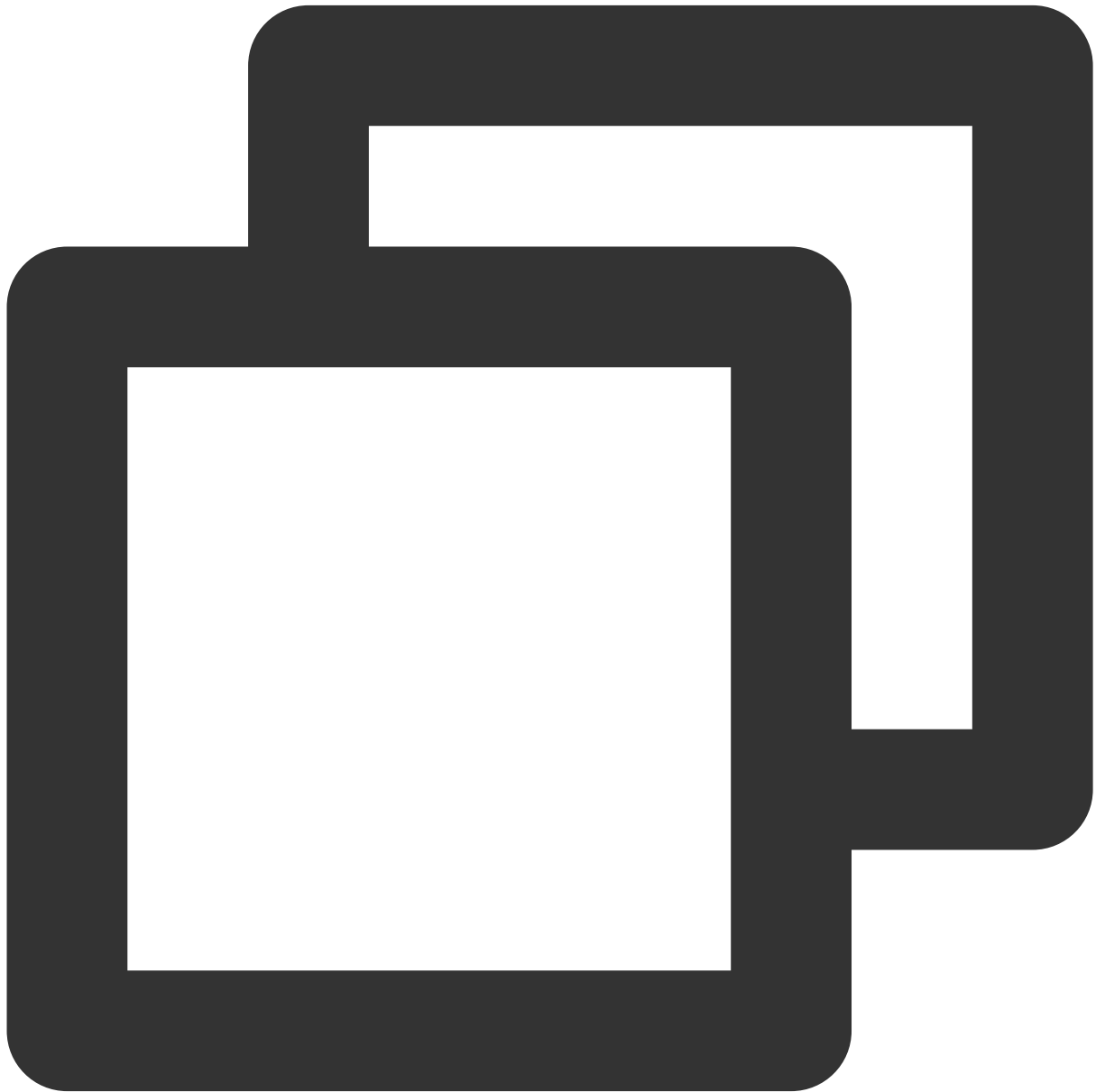
The parameters are described below:

| Parameter   | Type | Description                      |
|-------------|------|----------------------------------|
| userVolumes | List | List of user volumes             |
| totalVolume | int  | Total volume. Value range: 0-100 |

## Seat Callback APIs

### **onSeatListChange**

Callback for all seat changes.



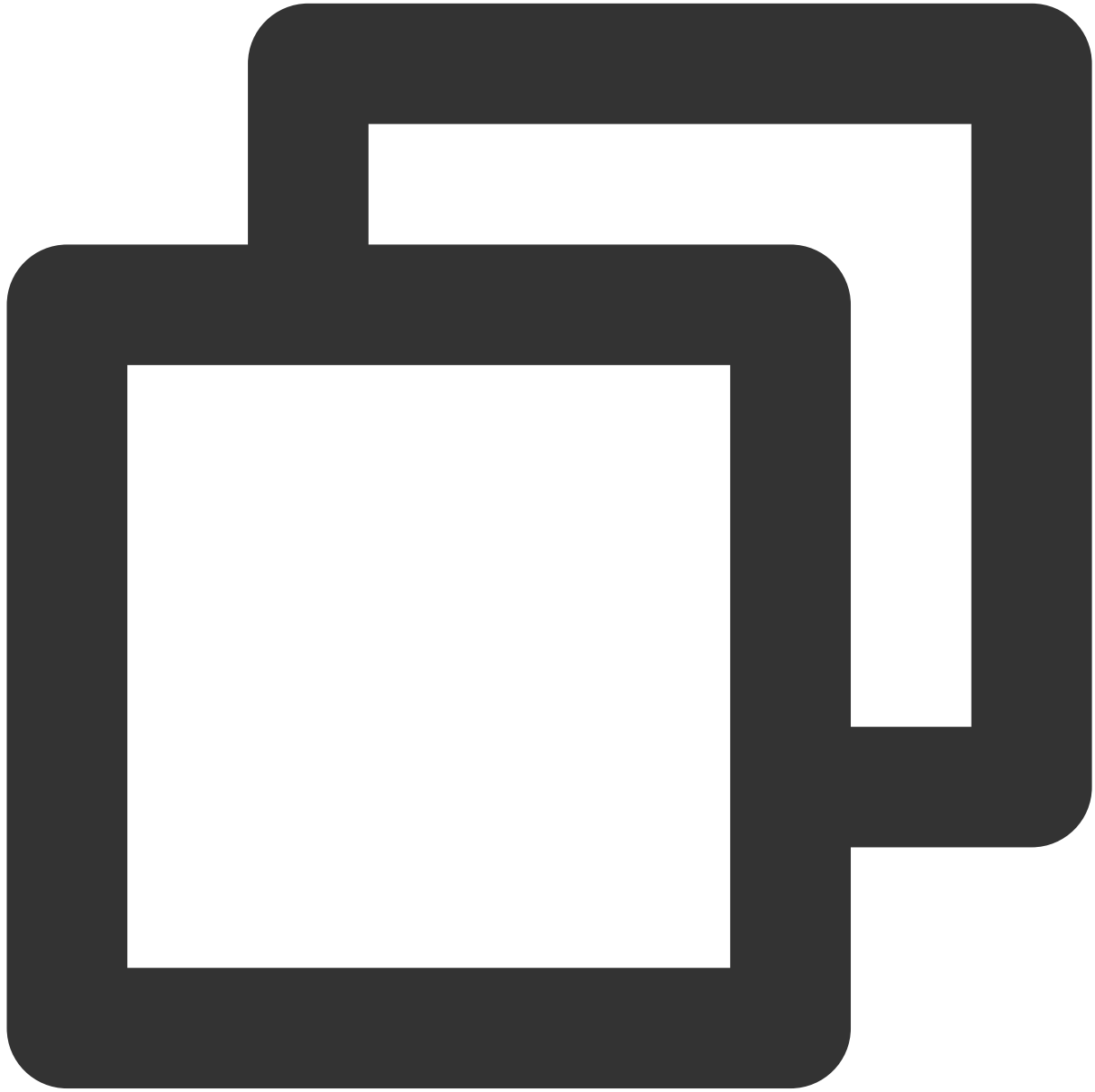
```
- (void)onSeatInfoChange:(NSArray<KaraokeSeatInfo *> *)seatInfoList  
NS_SWIFT_NAME(onSeatListChange(seatInfoList:));
```

The parameters are described below:

| Parameter    | Type           | Description    |
|--------------|----------------|----------------|
| seatInfoList | List<SeatInfo> | Full seat list |

## onAnchorEnterSeat

Someone became a speaker or was made a speaker by the owner.



```
- (void)onAnchorEnterSeat:(NSInteger)index  
                        user:(KaraokeUserInfo *)user  
NS_SWIFT_NAME(onAnchorEnterSeat(index:user:));
```

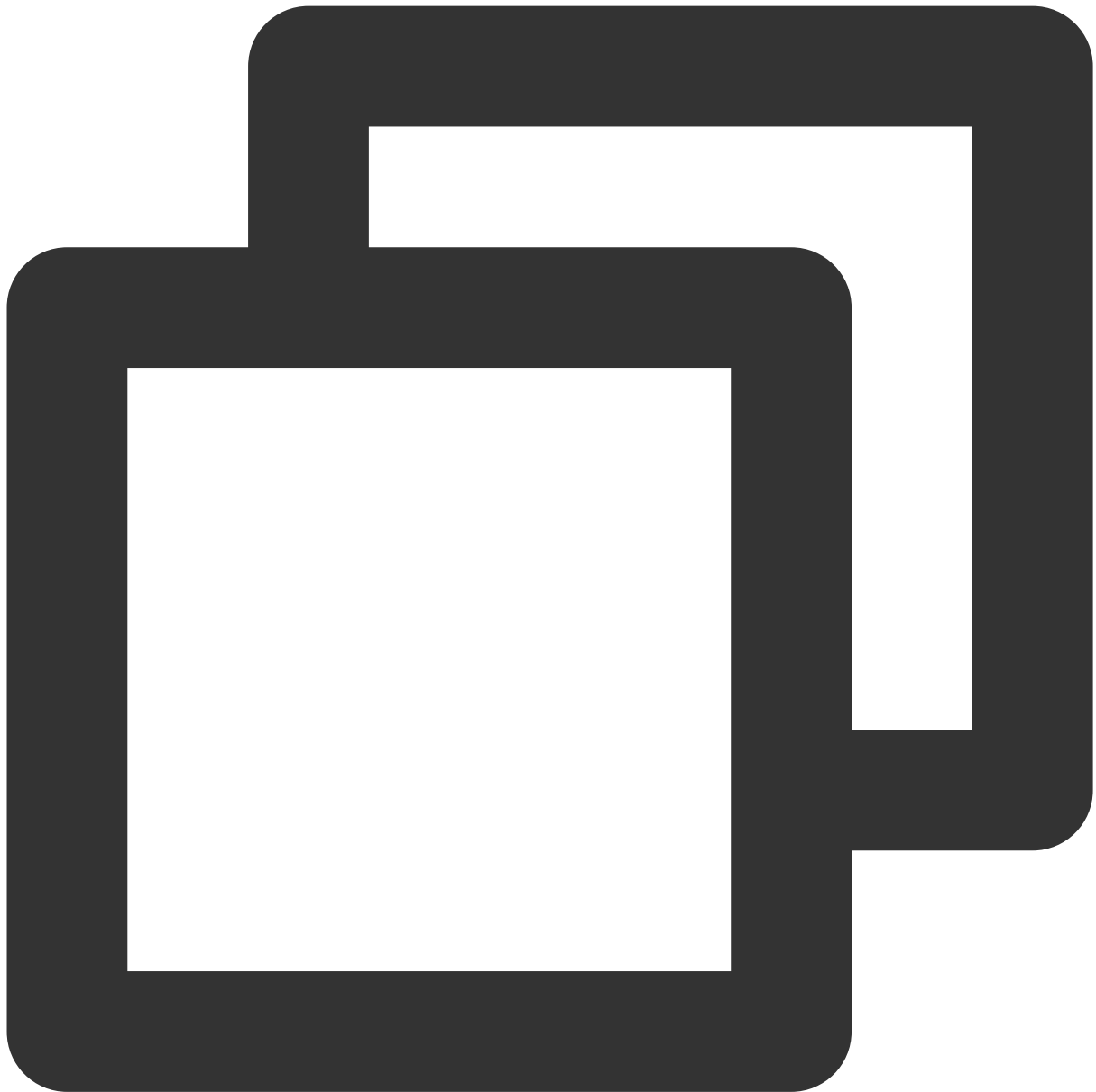
The parameters are described below:

| Parameter | Type | Description    |
|-----------|------|----------------|
| index     | int  | The seat taken |
|           |      |                |

|      |          |                                       |
|------|----------|---------------------------------------|
| user | UserInfo | Details of the user who took the seat |
|------|----------|---------------------------------------|

## onAnchorLeaveSeat

A speaker became a listener or was moved to listeners by the room owner.



```
- (void)onAnchorLeaveSeat:(NSInteger)index
    user:(KaraokeUserInfo *)user
NS_SWIFT_NAME(onAnchorLeaveSeat(index:user:));
```

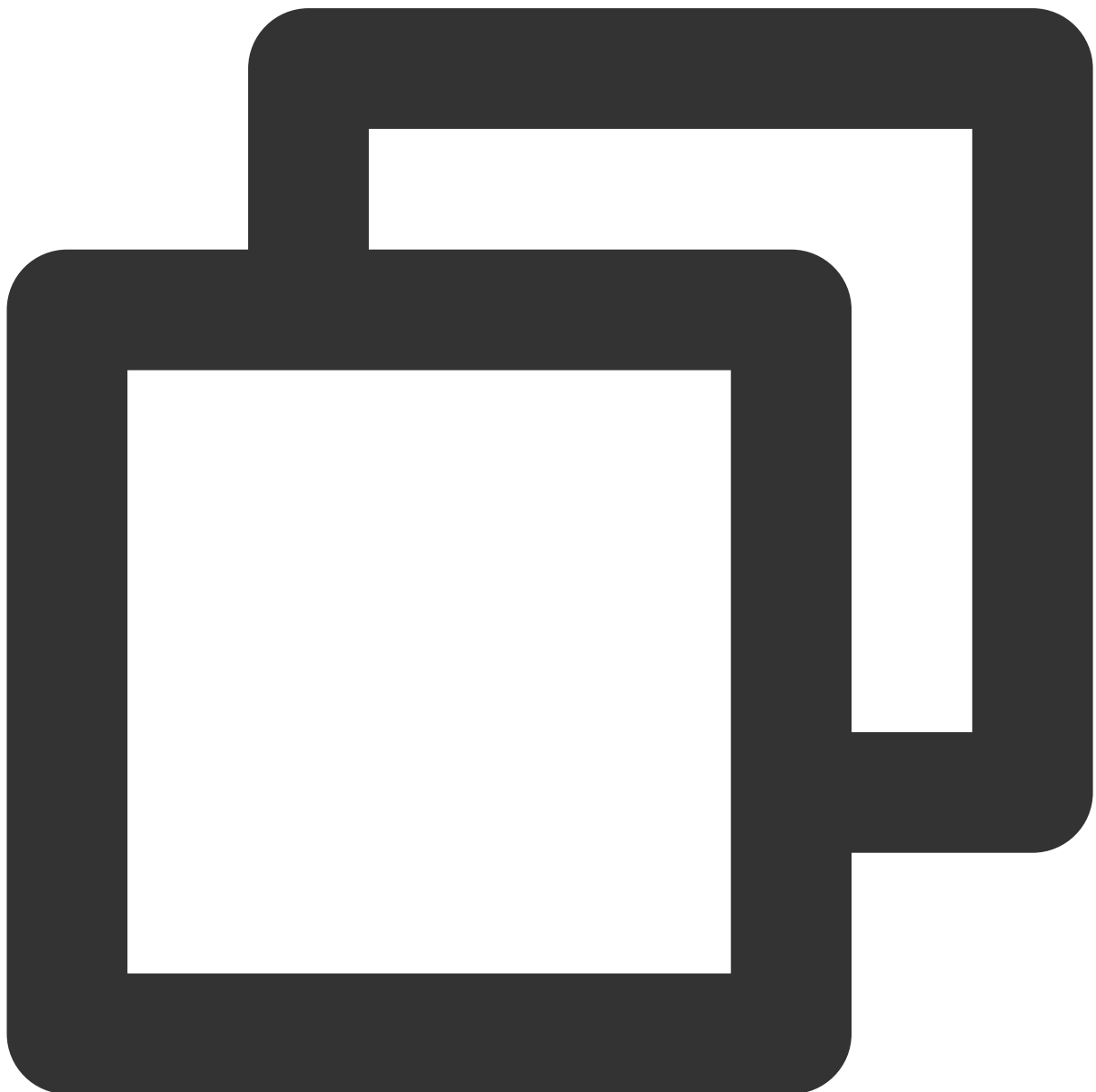
The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter | Type     | Description                                 |
|-----------|----------|---------------------------------------------|
| index     | int      | The seat previously occupied by the speaker |
| user      | UserInfo | Details of the user who took the seat       |

## onSeatMute

The room owner muted/unmuted a seat.



```
- (void)onSeatMute:(NSInteger) index
```

```
isMute: (BOOL) isMute
NS_SWIFT_NAME (onSeatMute (index: isMute:));
```

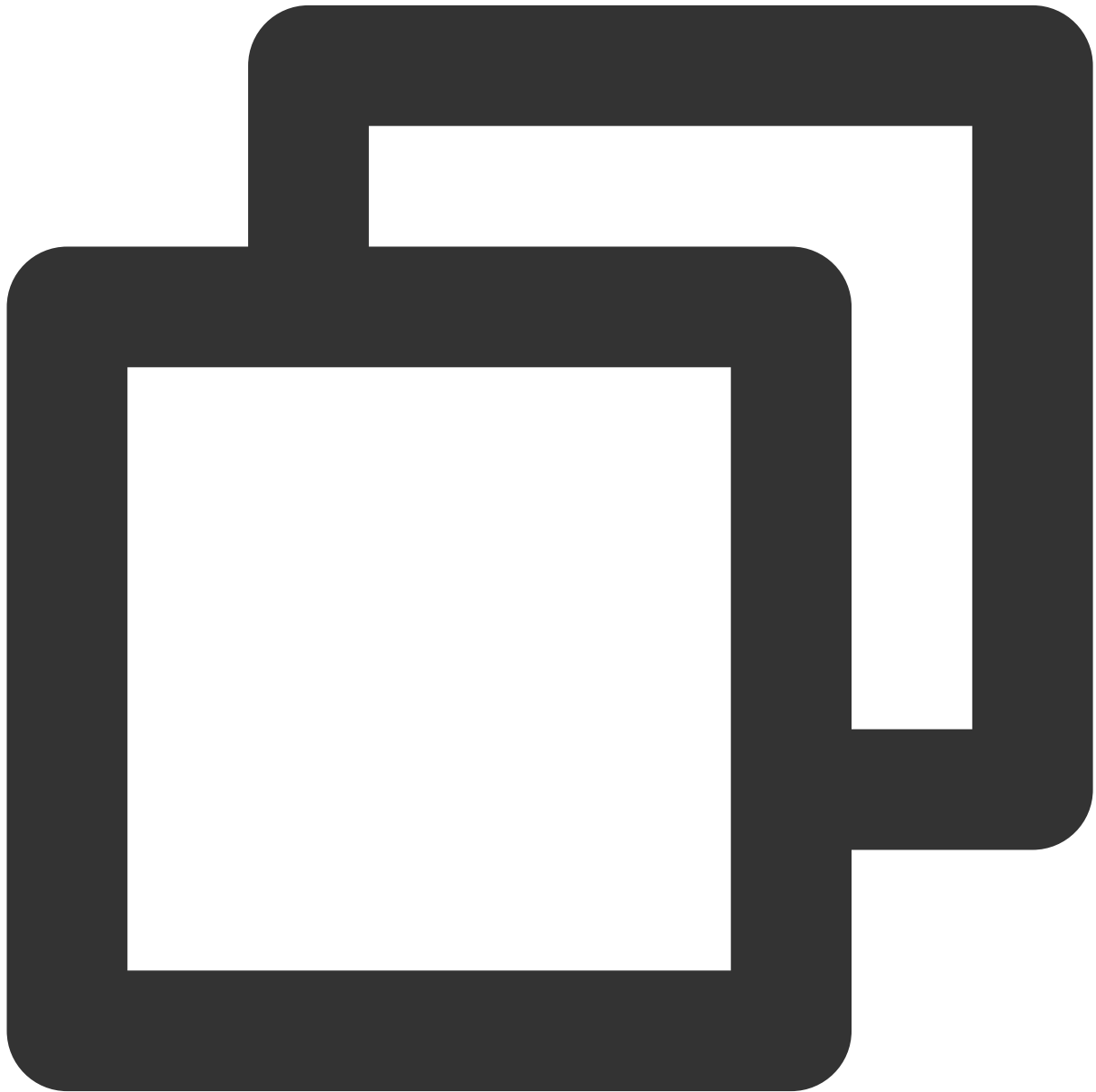
The parameters are described below:

| Parameter | Type    | Description                                             |
|-----------|---------|---------------------------------------------------------|
| index     | int     | The seat muted/unmuted                                  |
| isMute    | boolean | <code>true</code> : Muted; <code>false</code> : Unmuted |

## onSeatClose

The room owner blocked/unblocked a seat.





```
- (void)onSeatClose:(NSInteger)index  
    isClose:(BOOL)isClose  
NS_SWIFT_NAME(onSeatClose(index:isClose:));
```

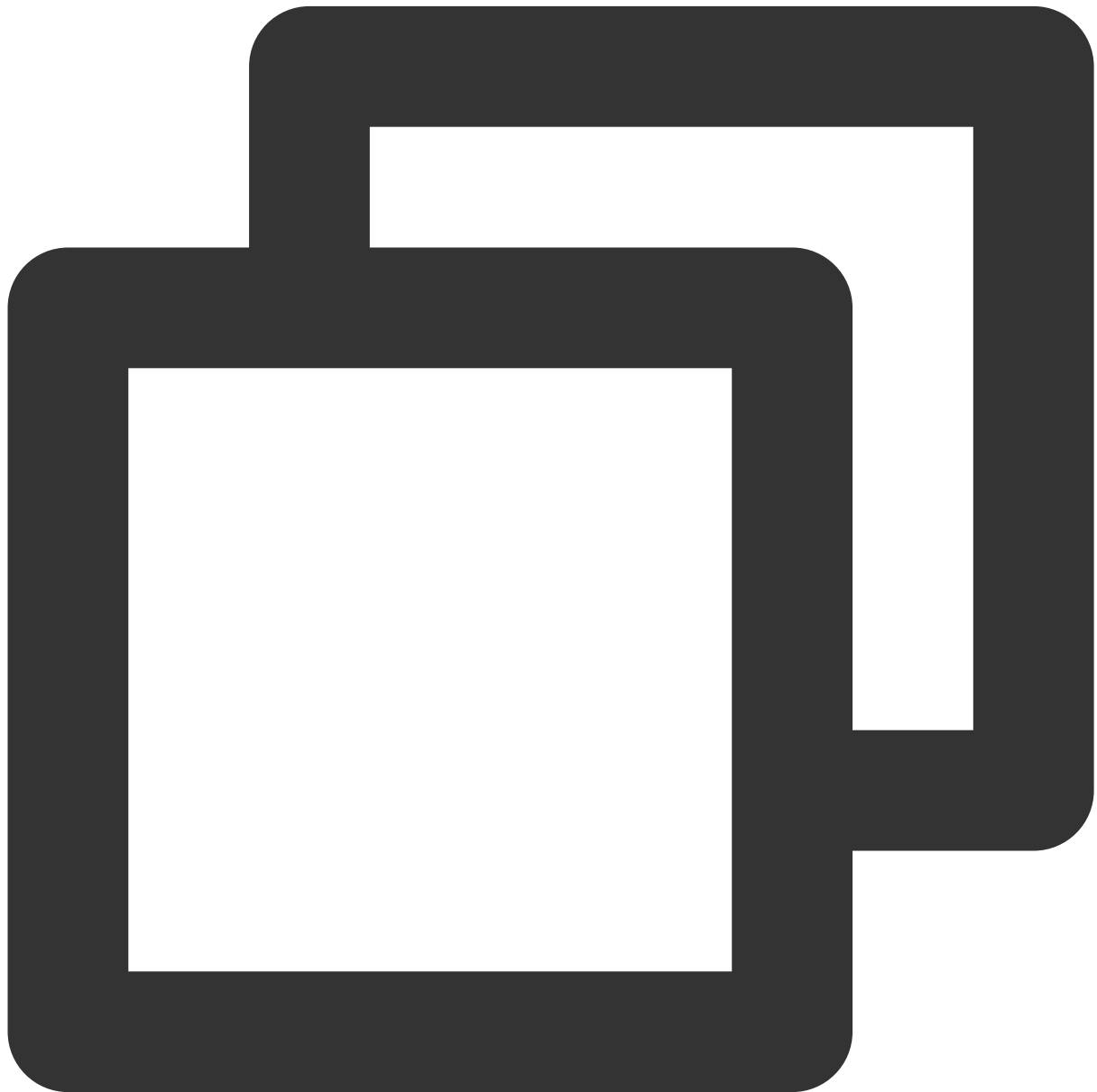
The parameters are described below:

| Parameter | Type    | Description                                                 |
|-----------|---------|-------------------------------------------------------------|
| index     | int     | The seat blocked/unblocked                                  |
| isClose   | boolean | <code>true</code> : Blocked; <code>false</code> : Unblocked |

## Callback APIs for Room Entry/Exit by Listener

### **onAudienceEnter**

A listener entered the room.



```
- (void)onAudienceEnter:(KaraokeUserInfo *)userInfo  
NS_SWIFT_NAME(onAudienceEnter(userInfo:));
```

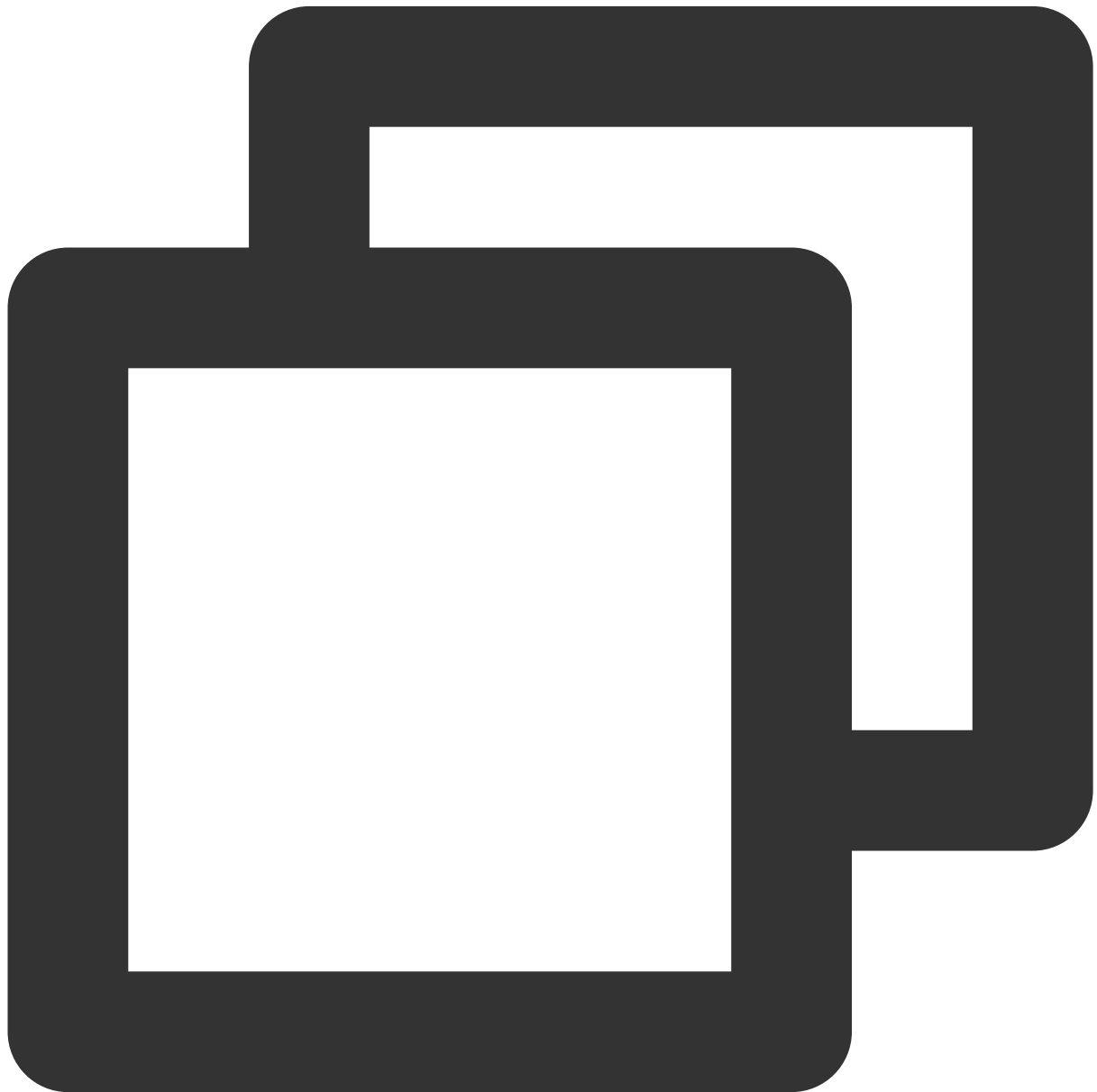
The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter | Type     | Description                                      |
|-----------|----------|--------------------------------------------------|
| userInfo  | UserInfo | Information of the listener who entered the room |

## onAudienceExit

A listener exited the room.



```
- (void)onAudienceExit:(KaraokeUserInfo *)userInfo  
NS_SWIFT_NAME(onAudienceExit(userInfo:));
```

The parameters are described below:

| Parameter | Type     | Description                                     |
|-----------|----------|-------------------------------------------------|
| userInfo  | UserInfo | Information of the listener who exited the room |

## Message Event Callback APIs

### **onRecvRoomTextMsg**

Callback for receiving a text chat message.



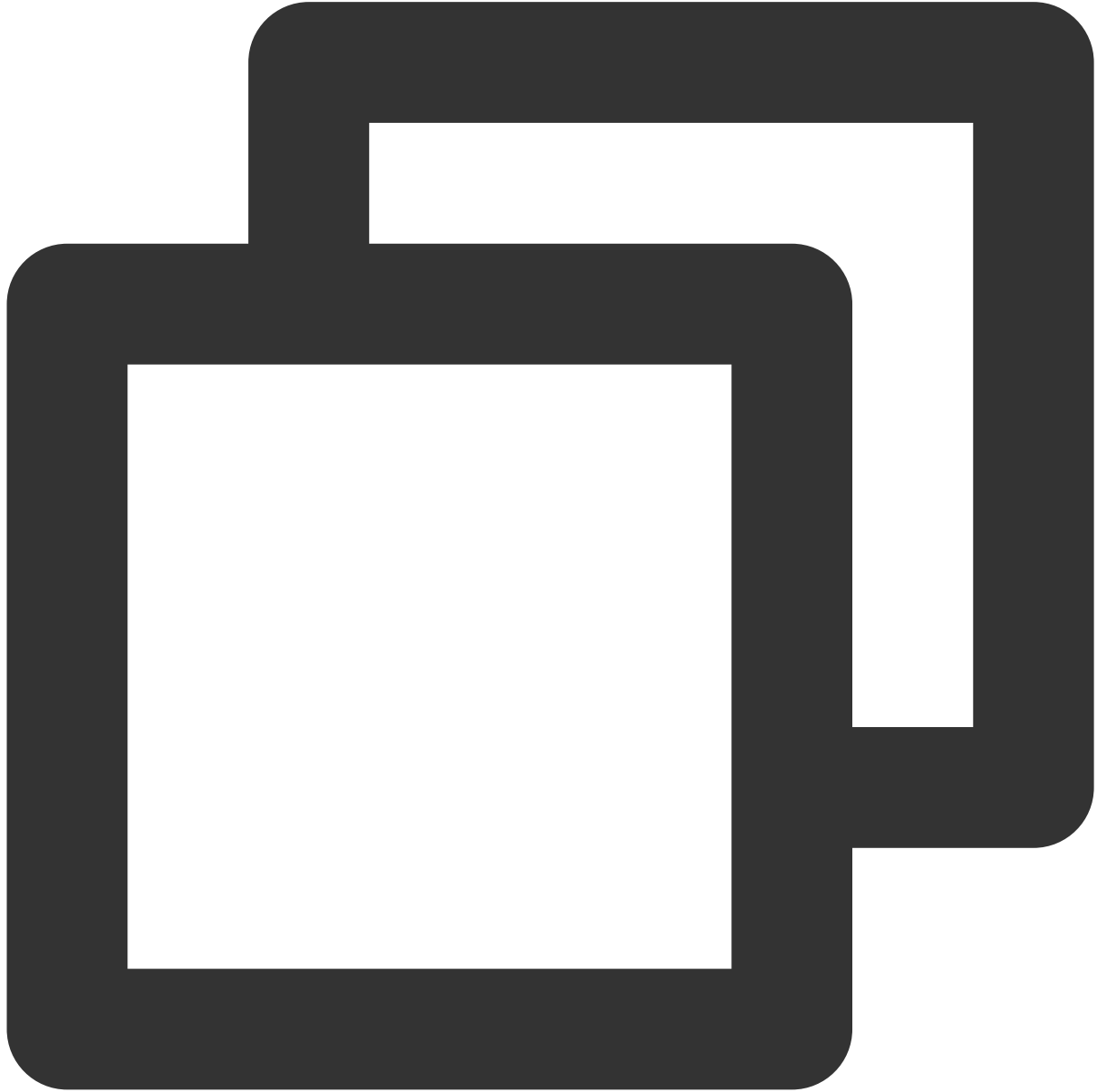
```
- (void)onRecvRoomTextMsg:(NSString *)message
    userInfo:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomTextMsg(message:userInfo:));
```

The parameters are described below:

| Parameter | Type     | Description               |
|-----------|----------|---------------------------|
| message   | String   | A text chat message.      |
| userInfo  | UserInfo | Information of the sender |

## onRecvRoomCustomMsg

A custom message was received.



```
- (void)onRecvRoomCustomMsg:(NSString *)cmd
    message:(NSString *)message
    userInfo:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomCustomMsg(cmd:message:userInfo));
```

The parameters are described below:

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

|          |          |                                                                         |
|----------|----------|-------------------------------------------------------------------------|
| command  | String   | Custom command word used to distinguish between different message types |
| message  | String   | A text chat message.                                                    |
| userInfo | UserInfo | Information of the sender                                               |

## Invitation Signaling Callback APIs

### **onReceiveNewInvitation**

An invitation was received.



```
- (void)onReceiveNewInvitation:(NSString *)identifier
    inviter:(NSString *)inviter
    cmd:(NSString *)cmd
    content:(NSString *)content
NS_SWIFT_NAME (onReceiveNewInvitation(identifier:inviter:cmd:content:));
```

The parameters are described below:

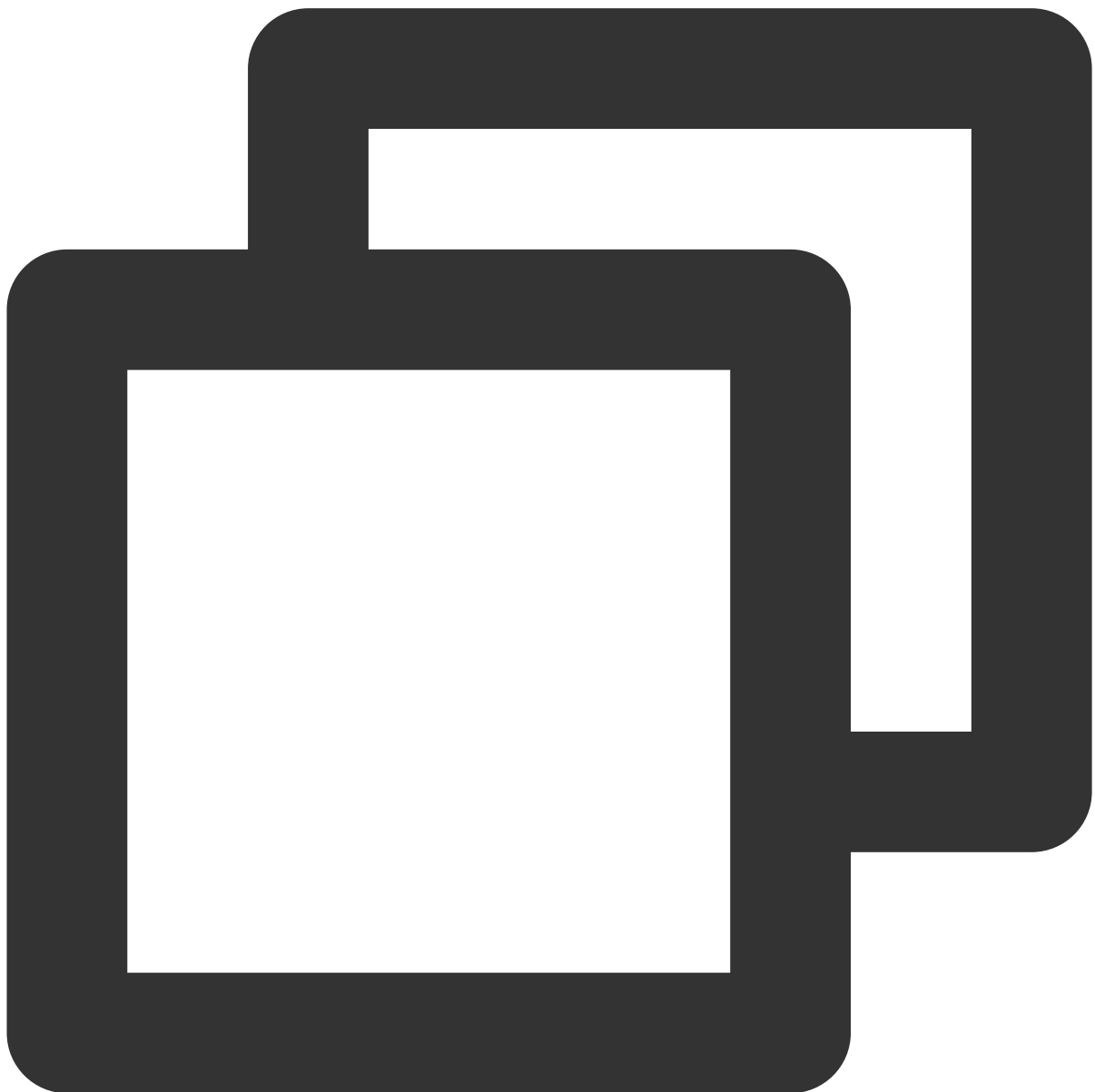
| Parameter | Type   | Description        |
|-----------|--------|--------------------|
| id        | String | The invitation ID. |



|         |          |                                              |
|---------|----------|----------------------------------------------|
| inviter | String   | The user ID of the inviter.                  |
| cmd     | String   | A custom command word specified by business. |
| content | UserInfo | Content specified by business                |

## onInviteeAccepted

The invitee accepted the invitation



```
- (void)onInviteeAccepted:(NSString *)identifier
```

```
invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeAccepted(identifier:invitee:));
```

The parameters are described below:

| Parameter | Type   | Description                 |
|-----------|--------|-----------------------------|
| id        | String | The invitation ID.          |
| invitee   | String | The user ID of the invitee. |

### **onInviteeRejected**

The invitee declined the invitation



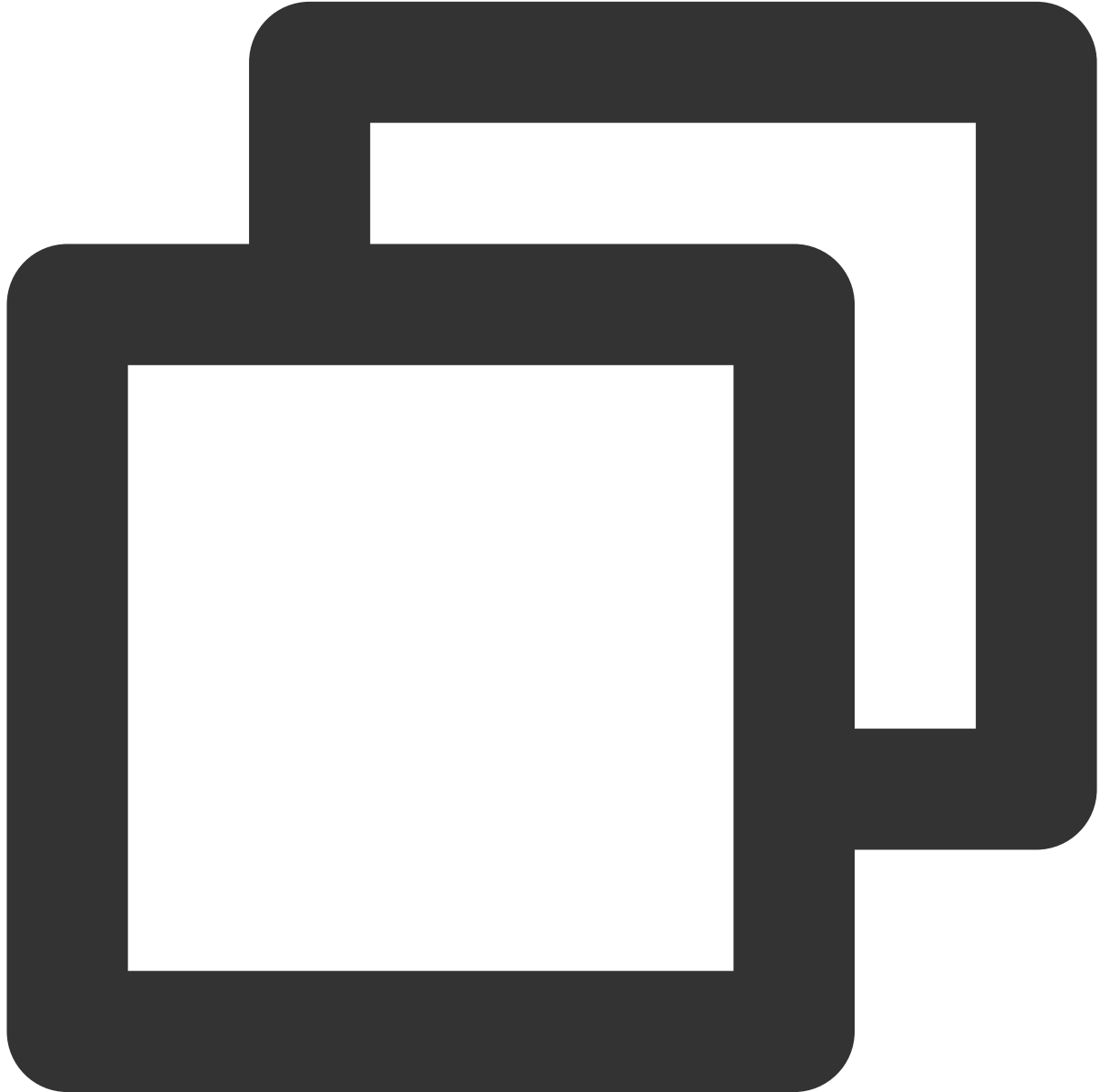
```
- (void)onInviteeRejected:(NSString *)identifier
    invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeRejected(identifier:invitee:));
```

The parameters are described below:

| Parameter | Type   | Description                 |
|-----------|--------|-----------------------------|
| id        | String | The invitation ID.          |
| invitee   | String | The user ID of the invitee. |

## onInvitationCancelled

The inviter canceled the invitation.



```
- (void)onInvitationCancelled:(NSString *)identifier  
    invitee:(NSString *)invitee NS_SWIFT_NAME(onInvitationCancelled)
```

The parameters are described below:

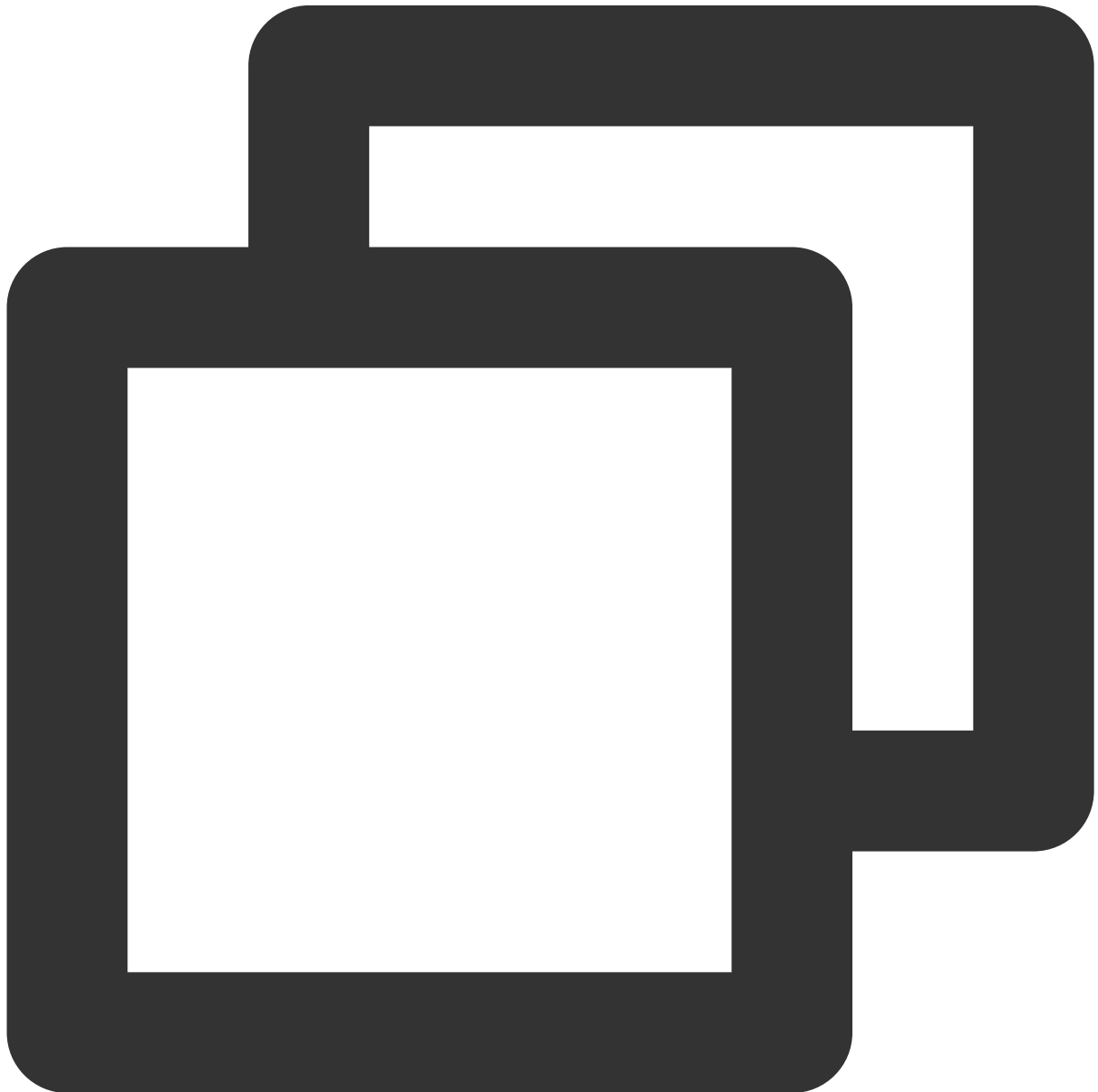
| Parameter | Type   | Description        |
|-----------|--------|--------------------|
| id        | String | The invitation ID. |

|         |        |                             |
|---------|--------|-----------------------------|
| invitee | String | The user ID of the invitee. |
|---------|--------|-----------------------------|

## Music Playback Status Callback APIs

### **onMusicPrepareToPlay**

Music playback is ready.



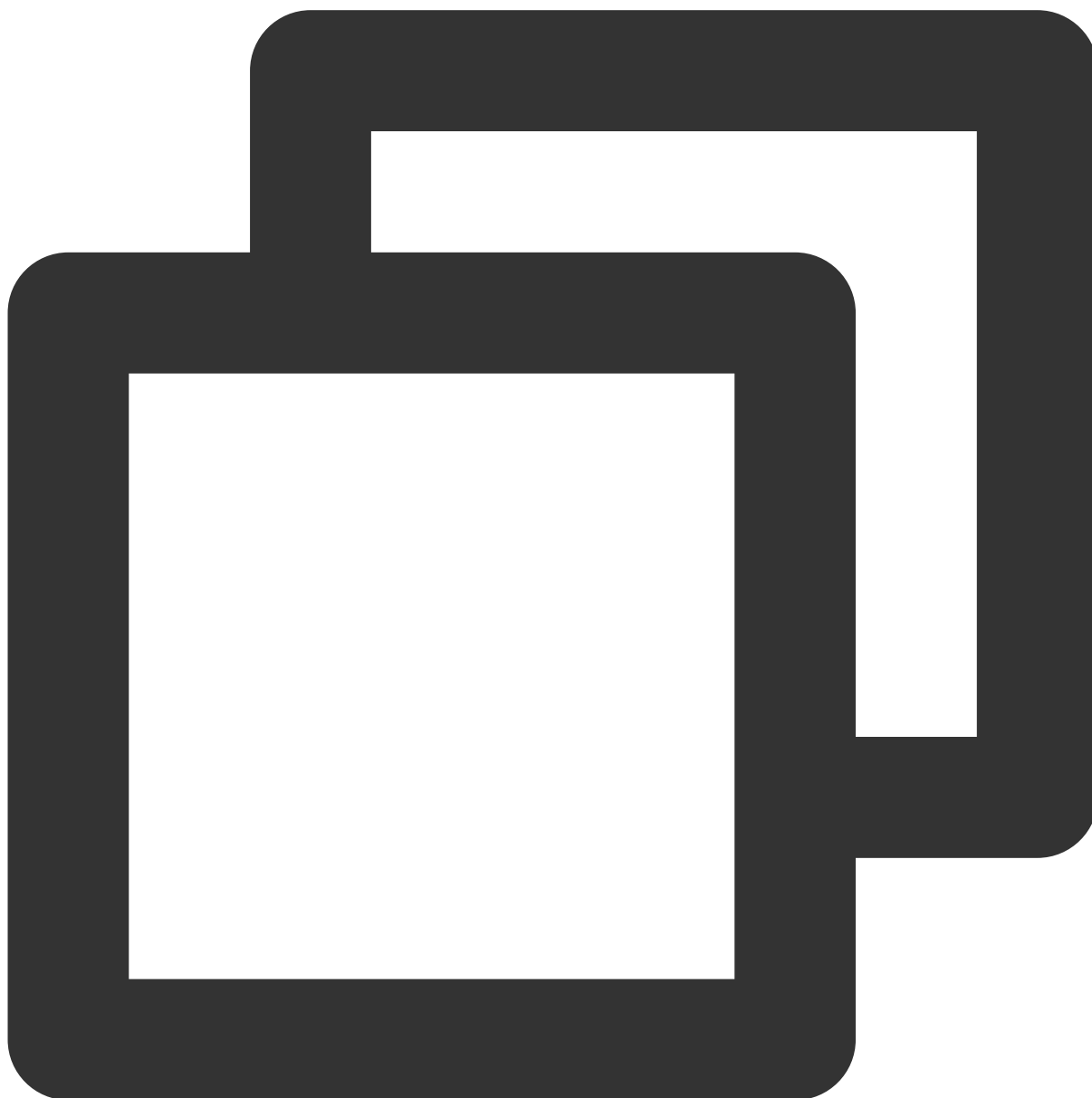
```
- (void)onMusicPrepareToPlay:(int32_t)musicID  
NS_SWIFT_NAME(onMusicPrepareToPlay(musicID));
```

The parameters are described below:

| Parameter | Type    | Description                                 |
|-----------|---------|---------------------------------------------|
| musicID   | int32_t | <code>musicID</code> passed in for playback |

### onMusicProgressUpdate

Music playback progress.



```
- (void)onMusicProgressUpdate:(int32_t)musicID
```

```
progress:(NSInteger)progress total:(NSInteger)total
NS_SWIFT_NAME(onMusicProgressUpdate(musicID:progress:total:));
```

The parameters are described below:

| Parameter | Type      | Description                                 |
|-----------|-----------|---------------------------------------------|
| musicID   | int32_t   | <code>musicID</code> passed in for playback |
| progress  | NSInteger | Current playback progress in ms             |
| total     | NSInteger | Total duration in ms                        |

## onMusicCompletePlaying

Music playback was completed.



```
- (void)onMusicCompletePlaying:(int32_t)musicID  
NS_SWIFT_NAME(onMusicCompletePlaying(musicID:));
```

The parameters are described below:

| Parameter | Type    | Description                                 |
|-----------|---------|---------------------------------------------|
| musicID   | int32_t | <code>musicID</code> passed in for playback |



# TRTCKaraoke (Android)

Last updated : 2023-09-25 10:59:36

`TRTCKaraokeRoom` includes the following features, which are based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat.

A user can create a karaoke room and become a speaker or enter a karaoke room as a listener.

The room owner can manage song requests as well as remove a speaker from a seat.

The room owner can also block a seat. A listener cannot request to take a blocked seat to become a speaker.

A listener can become a speaker to request songs and sing. A speaker can also become a listener.

All users can send gifts as well as custom chat messages. Custom messages can be used to send on-screen comments and give likes.

## Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely [TRTC](#) and [Chat](#). When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see [Pricing](#).

`TRTCKaraokeRoom` is an open-source class that depends on two closed-source Tencent Cloud SDKs. For the specific implementation process, see [Karaoke \(Android\)](#).

The [TRTC SDK](#) is used as a low-latency audio chat component.

The `AVChatRoom` feature of the [Chat SDK](#) is used to implement chat rooms. The attribute APIs of Chat are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

## `TRTCKaraokeRoom` API Overview

### Basic SDK APIs

| API                                   | Description                                |
|---------------------------------------|--------------------------------------------|
| <a href="#">sharedInstance</a>        | Gets a singleton object.                   |
| <a href="#">destroySharedInstance</a> | Terminates a singleton object.             |
| <a href="#">setDelegate</a>           | Sets event callbacks.                      |
| <a href="#">setDelegateHandler</a>    | Sets the thread where event callbacks are. |
| <a href="#">login</a>                 | Logs in.                                   |
|                                       |                                            |

|                                |               |
|--------------------------------|---------------|
| <a href="#">logout</a>         | Logs out.     |
| <a href="#">setSelfProfile</a> | Sets profile. |

## Room APIs

| API                             | Description                                                                                                                                                |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">createRoom</a>      | Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.                                            |
| <a href="#">destroyRoom</a>     | Terminates a room (called by room owner).                                                                                                                  |
| <a href="#">enterRoom</a>       | Enters a room (called by listener).                                                                                                                        |
| <a href="#">exitRoom</a>        | Exits a room (called by listener).                                                                                                                         |
| <a href="#">getRoomInfoList</a> | Gets room list details.                                                                                                                                    |
| <a href="#">getUserInfoList</a> | Gets the user information of the specified <code>userId</code> . If the value is <code>null</code> , the information of all users in the room is obtained. |

## Music playback APIs

| API                             | Description    |
|---------------------------------|----------------|
| <a href="#">startPlayMusic</a>  | Starts music.  |
| <a href="#">stopPlayMusic</a>   | Stops music.   |
| <a href="#">pausePlayMusic</a>  | Pauses music.  |
| <a href="#">resumePlayMusic</a> | Resumes music. |

## Seat management APIs

| API                       | Description                                           |
|---------------------------|-------------------------------------------------------|
| <a href="#">enterSeat</a> | Becomes a speaker (called by room owner or listener). |
| <a href="#">leaveSeat</a> | Becomes a listener (called by speaker).               |
| <a href="#">pickSeat</a>  | Places a user in a seat (called by room owner).       |
| <a href="#">kickSeat</a>  | Removes a speaker (called by room owner).             |
| <a href="#">muteSeat</a>  | Mutes/Unmutes a seat (called by room owner).          |

|                           |                                                |
|---------------------------|------------------------------------------------|
| <a href="#">closeSeat</a> | Blocks/Unblocks a seat (called by room owner). |
|---------------------------|------------------------------------------------|

## Local audio APIs

| API                                      | Description                                                       |
|------------------------------------------|-------------------------------------------------------------------|
| <a href="#">startMicrophone</a>          | Starts mic capturing.                                             |
| <a href="#">stopMicrophone</a>           | Stops mic capturing.                                              |
| <a href="#">setAudioQuality</a>          | Sets audio quality.                                               |
| <a href="#">muteLocalAudio</a>           | Mutes/Unmutes local audio.                                        |
| <a href="#">setSpeaker</a>               | Sets whether to use the device speaker or receiver to play audio. |
| <a href="#">setAudioCaptureVolume</a>    | Sets mic capturing volume.                                        |
| <a href="#">setAudioPlayoutVolume</a>    | Sets playback volume.                                             |
| <a href="#">setVoiceEarMonitorEnable</a> | Enables/Disables in-ear monitoring.                               |

## Remote audio APIs

| API                                | Description                       |
|------------------------------------|-----------------------------------|
| <a href="#">muteRemoteAudio</a>    | Mutes/Unmutes a specified member. |
| <a href="#">muteAllRemoteAudio</a> | Mutes/Unmutes all members.        |

## Background music and audio effect APIs

| API                                   | Description                                                                                         |
|---------------------------------------|-----------------------------------------------------------------------------------------------------|
| <a href="#">getAudioEffectManager</a> | Gets the background music and audio effect management object <a href="#">TXAudioEffectManager</a> . |

## Message sending APIs

| API                               | Description                                                                                  |
|-----------------------------------|----------------------------------------------------------------------------------------------|
| <a href="#">sendRoomTextMsg</a>   | Broadcasts a text chat message in a room. This API is generally used for on-screen comments. |
| <a href="#">sendRoomCustomMsg</a> | Sends a custom text message.                                                                 |

## Invitation signaling APIs

| API                              | Description             |
|----------------------------------|-------------------------|
| <a href="#">sendInvitation</a>   | Sends an invitation.    |
| <a href="#">acceptInvitation</a> | Accepts an invitation.  |
| <a href="#">rejectInvitation</a> | Declines an invitation. |
| <a href="#">cancelInvitation</a> | Cancels an invitation.  |

## TRTCKaraokeRoomDelegate API Overview

### Common event callbacks

| API                        | Description           |
|----------------------------|-----------------------|
| <a href="#">onError</a>    | Callback for error.   |
| <a href="#">onWarning</a>  | Callback for warning. |
| <a href="#">onDebugLog</a> | Callback of log.      |

### Room event callback APIs

| API                                | Description                   |
|------------------------------------|-------------------------------|
| <a href="#">onRoomDestroy</a>      | The room was terminated.      |
| <a href="#">onRoomInfoChange</a>   | The room information changed. |
| <a href="#">onUserVolumeUpdate</a> | The user volume.              |

### Seat list change callback APIs

| API                               | Description                                                        |
|-----------------------------------|--------------------------------------------------------------------|
| <a href="#">onSeatListChange</a>  | All seat changes.                                                  |
| <a href="#">onAnchorEnterSeat</a> | A user became a speaker or was made a speaker by the room owner.   |
| <a href="#">onAnchorLeaveSeat</a> | A user became a listener or was made a listener by the room owner. |

|                                      |                                |
|--------------------------------------|--------------------------------|
| <a href="#">onSeatMute</a>           | The room owner muted a seat.   |
| <a href="#">onUserMicrophoneMute</a> | Whether a user's mic is muted. |
| <a href="#">onSeatClose</a>          | The room owner blocked a seat. |

### Callback APIs for room entry/exit by listener

| API                             | Description                  |
|---------------------------------|------------------------------|
| <a href="#">onAudienceEnter</a> | A listener entered the room. |
| <a href="#">onAudienceExit</a>  | A listener exited the room.  |

### Message event callback APIs

| API                                 | Description                       |
|-------------------------------------|-----------------------------------|
| <a href="#">onRecvRoomTextMsg</a>   | A text chat message was received. |
| <a href="#">onRecvRoomCustomMsg</a> | A custom message was received.    |

## Signaling Event Callback APIs

| API                                    | Description                          |
|----------------------------------------|--------------------------------------|
| <a href="#">onReceiveNewInvitation</a> | Receipt of an invitation.            |
| <a href="#">onInviteeAccepted</a>      | Invitation accepted by invitee.      |
| <a href="#">onInviteeRejected</a>      | Invitation declined by invitee.      |
| <a href="#">onInvitationCancelled</a>  | The inviter canceled the invitation. |

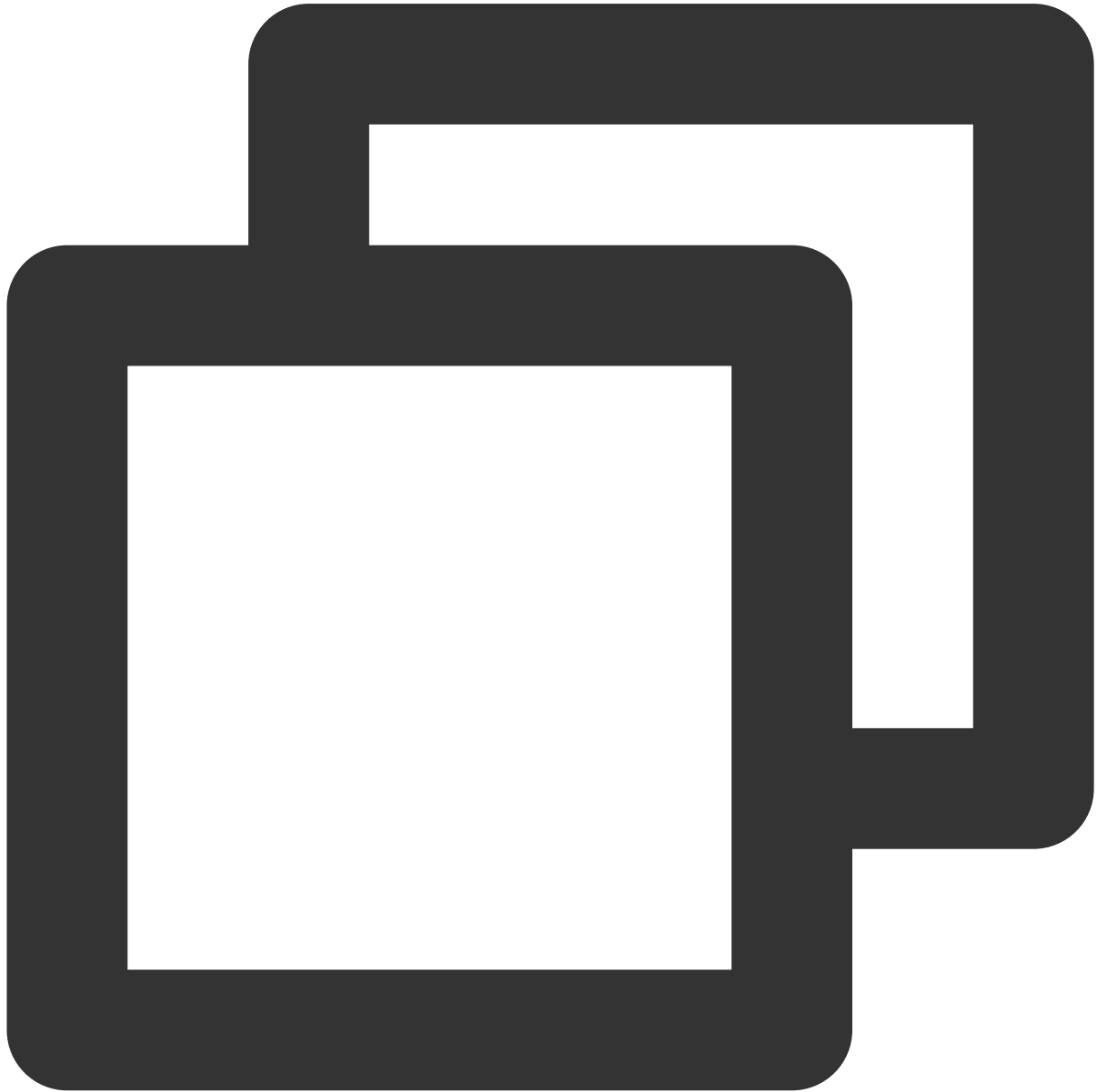
### Song event callback APIs

| API                                    | Description                   |
|----------------------------------------|-------------------------------|
| <a href="#">onMusicProgressUpdate</a>  | Music playback progress.      |
| <a href="#">onMusicPrepareToPlay</a>   | Music playback is ready.      |
| <a href="#">onMusicCompletePlaying</a> | Music playback was completed. |

## Basic SDK APIs

### **sharedInstance**

This API is used to get a [TRTCKaraokeRoom](#) singleton object.



```
public static synchronized TRTCKaraokeRoom sharedInstance(Context context);
```

The parameters are described below:

| Parameter | Type    | Description                                                                                                 |
|-----------|---------|-------------------------------------------------------------------------------------------------------------|
| context   | Context | Android context, which will be converted to <code>ApplicationContext</code> for the calling of system APIs. |

## destroySharedInstance

This API is used to terminate a [TRTCKaraokeRoom](#) singleton object.

### Note

After the instance is terminated, the externally cached `TRTCKaraokeRoom` instance can no longer be used. You need to call [sharedInstance](#) again to get a new instance.

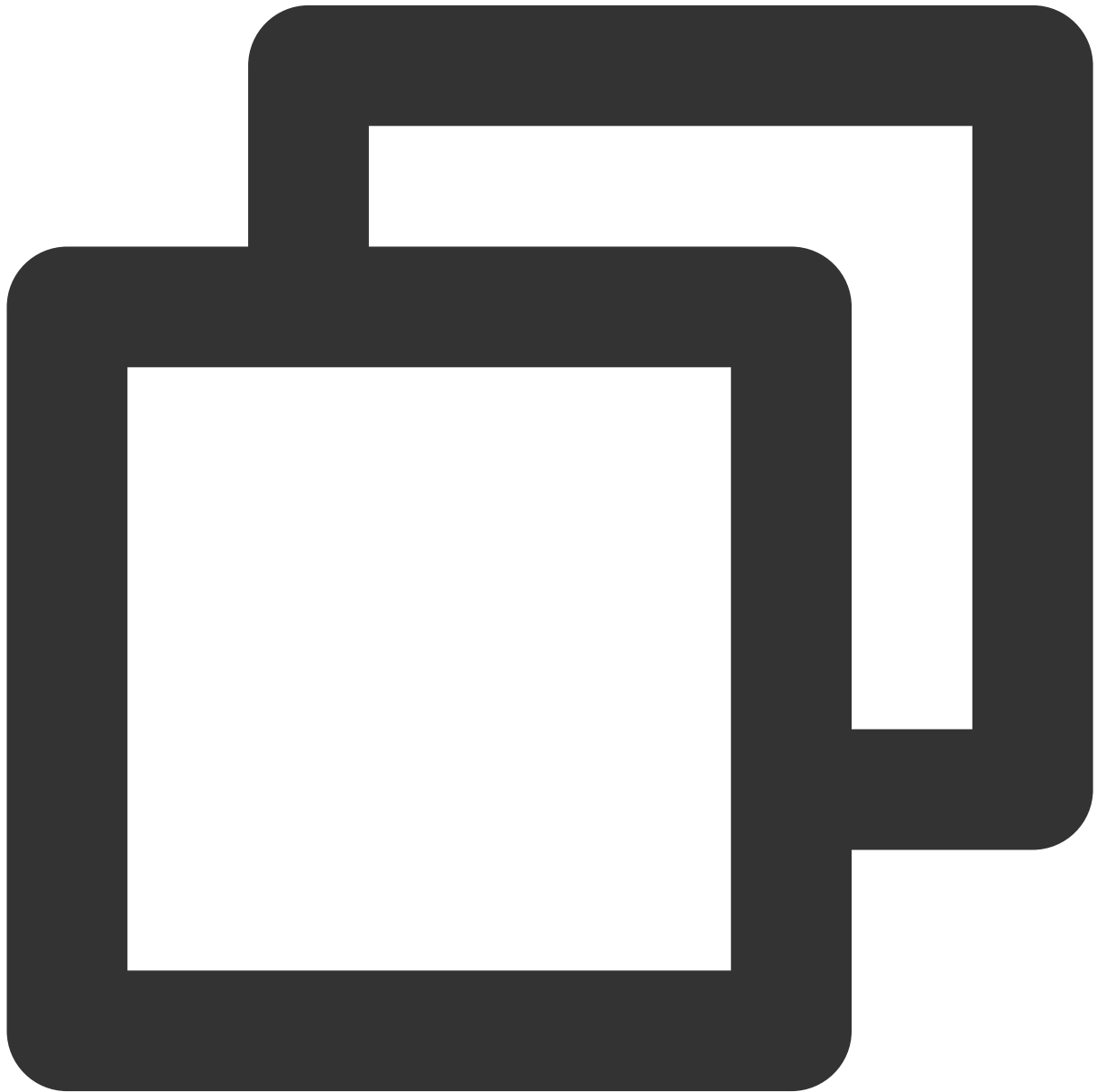


```
public static void destroySharedInstance();
```

### setDelegate

This API is used to set the event callbacks of [TRTCKaraokeRoom](#). You can use `TRTCKaraokeRoomDelegate` to get different status notifications of [TRTCKaraokeRoom](#).





```
public abstract void setDelegate(TRTCKaraokeRoomDelegate delegate);
```

**Note**

`setDelegate` is the delegate callback of `TRTCKaraokeRoom` .

**setDelegateHandler**

This API is used to set the thread where event callbacks are.



```
public abstract void setDelegateHandler(Handler handler);
```

The parameters are described below:

| Parameter | Type    | Description                                                                                          |
|-----------|---------|------------------------------------------------------------------------------------------------------|
| handler   | Handler | The status notifications of <code>TRTCKaraokeRoom</code> are sent to the handler thread you specify. |

## login

Login



```
public abstract void login(int sdkAppId,  
    String userId, String userSig,  
    TRTCKaraokeRoomCallback.ActionCallback callback);
```

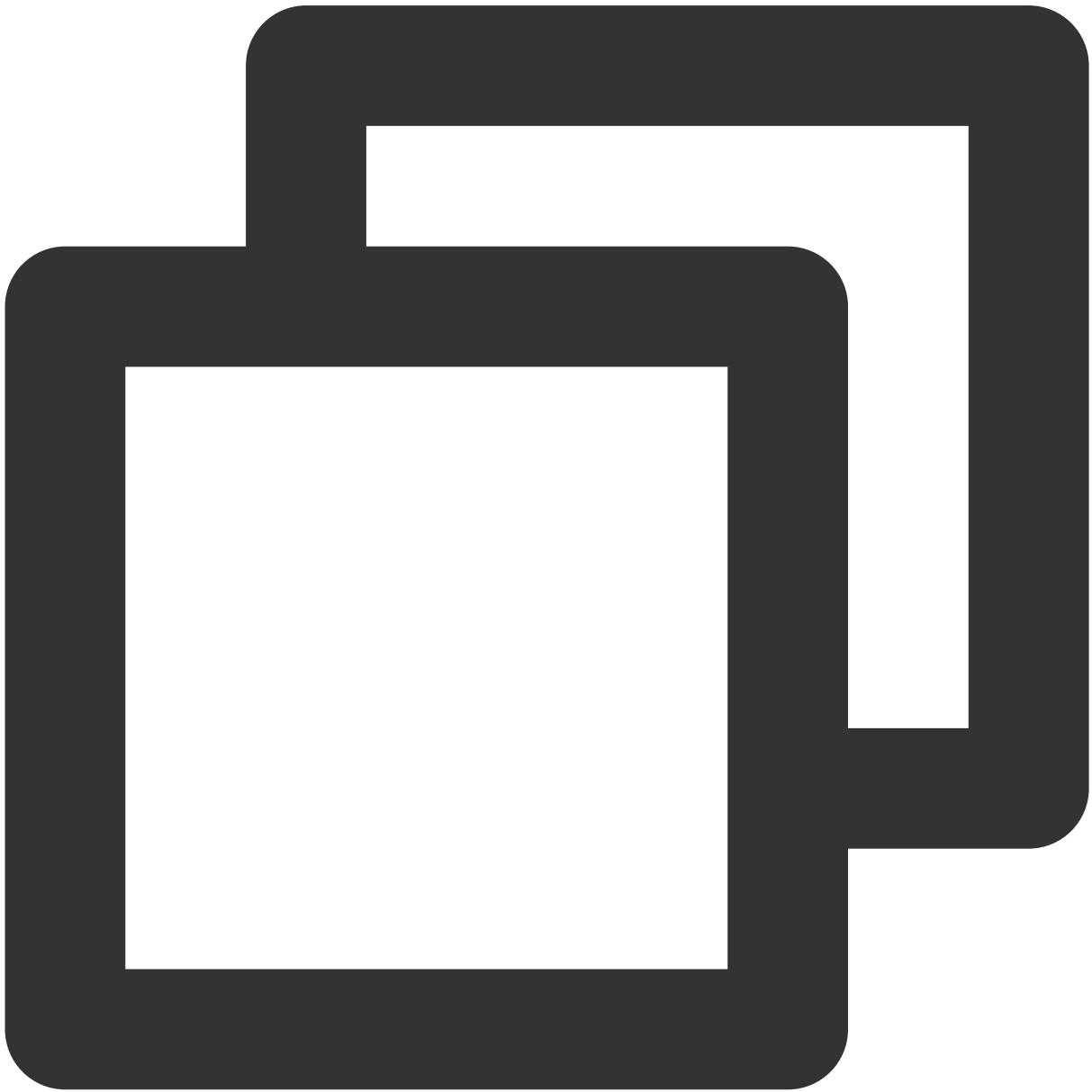
The parameters are described below:

| Parameter | Type | Description                                                                                                                      |
|-----------|------|----------------------------------------------------------------------------------------------------------------------------------|
| sdkAppId  | int  | You can view the <code>SDKAppID</code> via <a href="#">Application Management</a> > <b>Application Info</b> in the TRTC console. |

|          |                |                                                                                                                                            |
|----------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| userId   | String         | The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). |
| userSig  | String         | Tencent Cloud's proprietary security signature. For how to calculate and use it, see <a href="#">FAQs &gt; UserSig</a> .                   |
| callback | ActionCallback | The callback for login. The code is <code>0</code> if login succeeds.                                                                      |

## logout

Log out



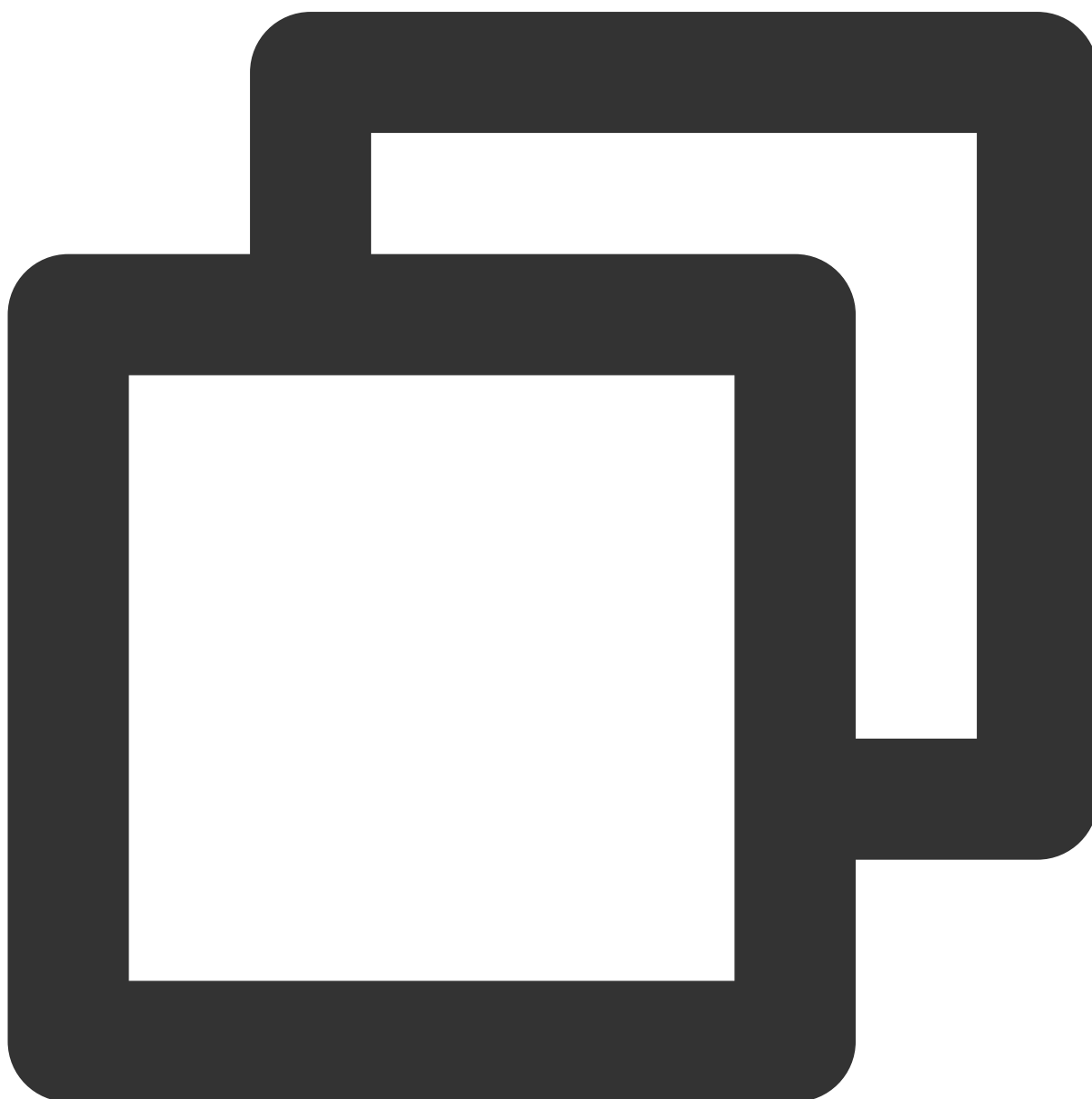
```
public abstract void logout (TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are described below:

| Parameter | Type           | Description                                                |
|-----------|----------------|------------------------------------------------------------|
| callback  | ActionCallback | The callback for logout. The code is 0 if logout succeeds. |

## setSelfProfile

This API is used to set the profile.



```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCKaraokeR
```

The parameters are described below:

| Parameter | Type           | Description                                                                      |
|-----------|----------------|----------------------------------------------------------------------------------|
| userName  | String         | The username.                                                                    |
| avatar    | String         | The address of the profile photo.                                                |
| callback  | ActionCallback | The callback for profile configuration. The code is 0 if the operation succeeds. |

## Room APIs

### createRoom

This API is used to create a room (called by room owner).



```
public abstract void createRoom(int roomId, TRTCKaraokeRoomDef.RoomParam roomParam,
```

The parameters are described below:

| Parameter | Type | Description                                                                                                                                                                                                                                                                           |
|-----------|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| roomId    | int  | The room ID. You need to assign and manage room IDs in a centralized manner. Multiple <code>roomId</code> values can be aggregated into a karaoke room list. Currently, Tencent Cloud does not provide management services for karaoke room lists. Please manage your own room lists. |

|           |                     |                                                                                                                                                     |
|-----------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| roomParam | TRTCCreateRoomParam | Room information, such as room name, seat list information, and cover information. To manage seats, you must enter the number of seats in the room. |
| callback  | ActionCallback      | The callback for room creation. The code is <code>0</code> if the operation succeeds.                                                               |

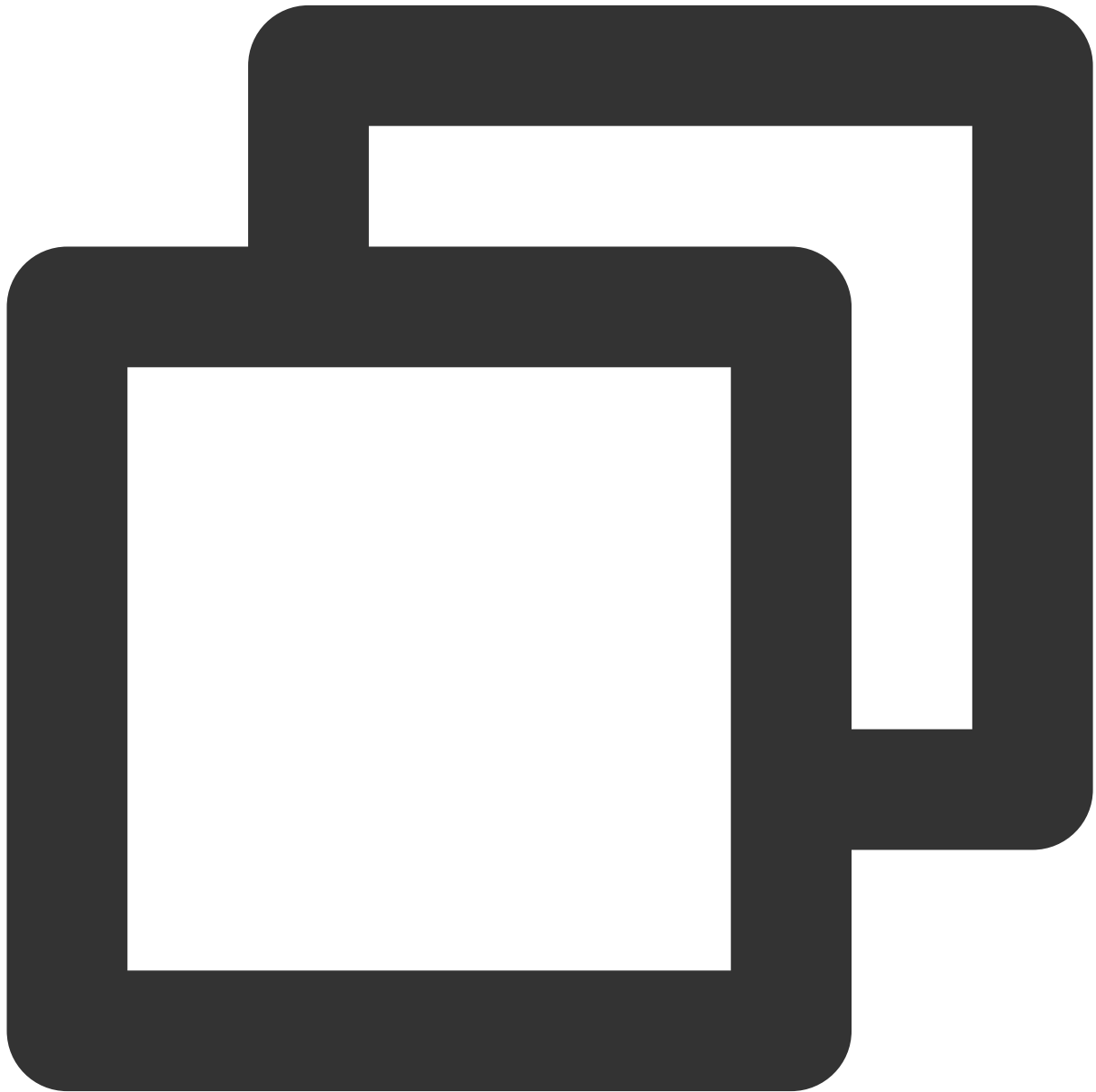
The process of creating a karaoke room and becoming a speaker is as follows:

1. A user calls `createRoom` to create a karaoke room, passing in room attributes (i.e., room ID, whether listeners need room owner's permission to speak, number of seats).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
4. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

## **destroyRoom**

This API is used to terminate a room (called by room owner).





```
public abstract void destroyRoom(TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are described below:

| Parameter | Type           | Description                                                                 |
|-----------|----------------|-----------------------------------------------------------------------------|
| callback  | ActionCallback | The callback for room termination. The code is 0 if the operation succeeds. |

## enterRoom

This API is used to enter a room (called by listener).



```
public abstract void enterRoom(int roomId, TRTCKaraokeRoomCallback.ActionCallback callback)
```

The parameters are described below:

| Parameter | Type           | Description                                                           |
|-----------|----------------|-----------------------------------------------------------------------|
| roomId    | int            | The room ID.                                                          |
| callback  | ActionCallback | The callback for room entry. The code is 0 if the operation succeeds. |

The process of entering a room as a listener is as follows:

1. A user gets the latest karaoke room list from your server. The list may contain the `roomId` and room information of multiple karaoke rooms.
2. The user selects a room, and enters the room by calling `enterRoom` with the room ID passed in.
3. After entering the room, the user receives an `onRoomInfoChange` notification about room attribute change from the component. The attributes can be recorded, and corresponding changes can be made to the UI, including room name, whether room owner's permission is required for listeners to speak, etc.
4. The user will receive an `onSeatListChange` notification about the change of the seat list and can update the change to the UI.
5. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker.

## **exitRoom**

Leave room



```
public abstract void exitRoom(TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are described below:

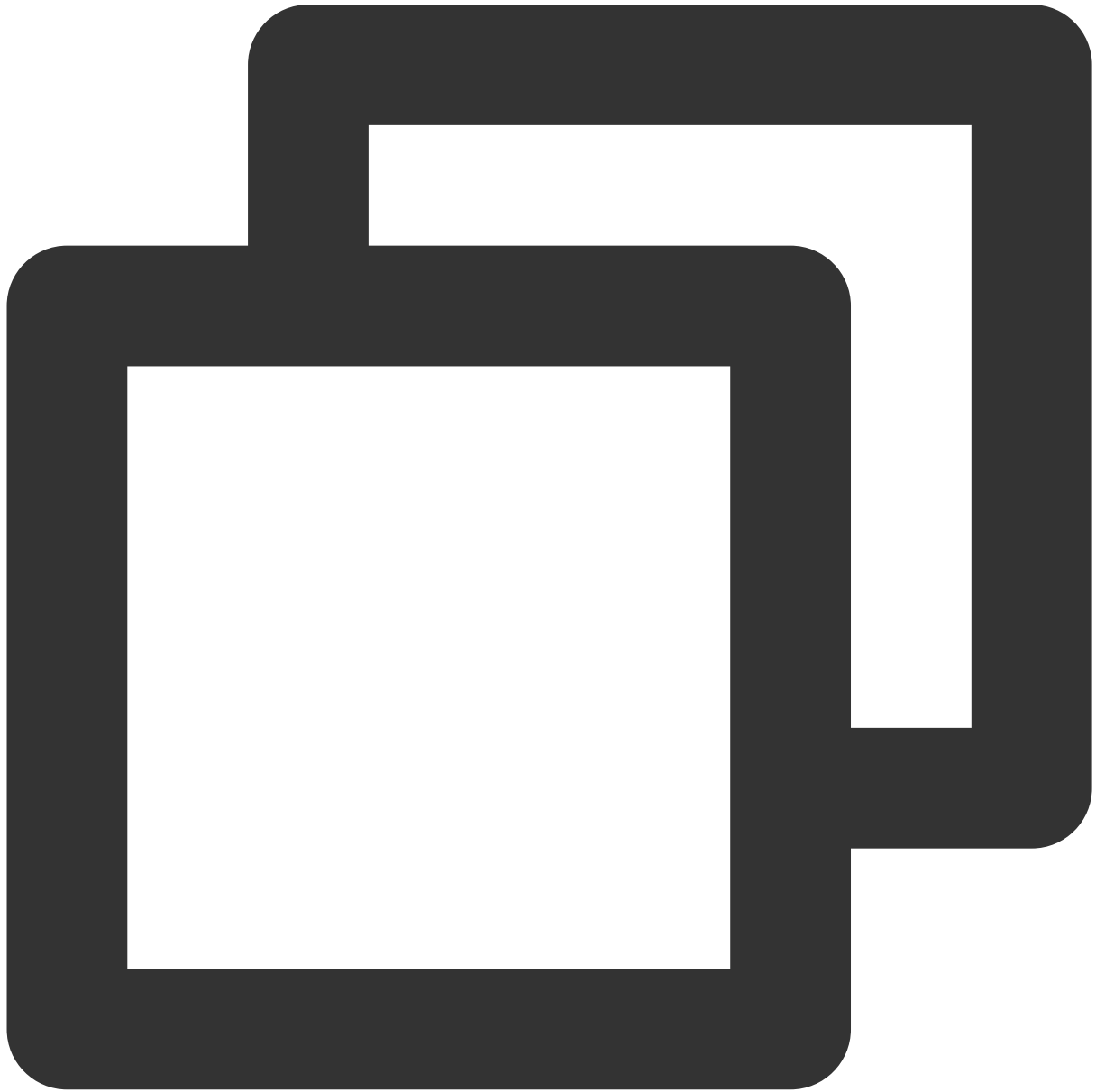
| Parameter | Type           | Description                                                          |
|-----------|----------------|----------------------------------------------------------------------|
| callback  | ActionCallback | The callback for room exit. The code is 0 if the operation succeeds. |

## getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()`.

**Note**

You don't need this API if both the room list and room information are managed on your server.



```
public abstract void getRoomInfoList(List<Integer> roomIdList, TRTCKaraokeRoomCallb
```

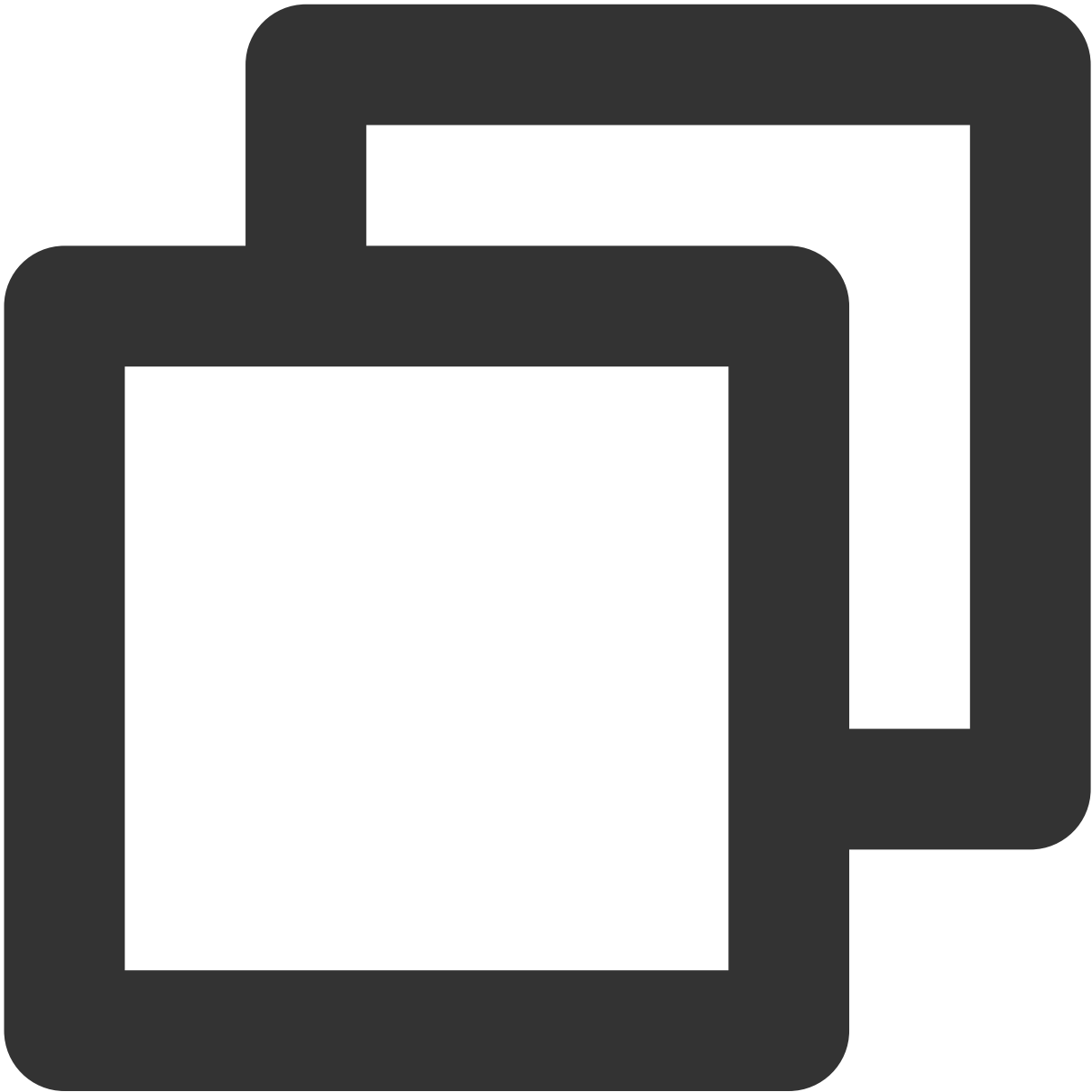
The parameters are described below:

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

|            |                  |                               |
|------------|------------------|-------------------------------|
| roomIdList | List<Integer>    | The list of room IDs.         |
| callback   | RoomInfoCallback | The callback of room details. |

getUserInfoList

This API is used to get the user information of a specified `userId` .



```
public abstract void getUserInfoList(List<String> userIdList, TRTCKaraokeRoomCallba
```

The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter        | Type             | Description                                                                                                                  |
|------------------|------------------|------------------------------------------------------------------------------------------------------------------------------|
| userIdList       | List<String>     | The IDs of the users to query. If this parameter is <code>null</code> , the information of all users in the room is queried. |
| userlistcallback | UserListCallback | The callback of user details.                                                                                                |

## Music Playback APIs

### startPlayMusic

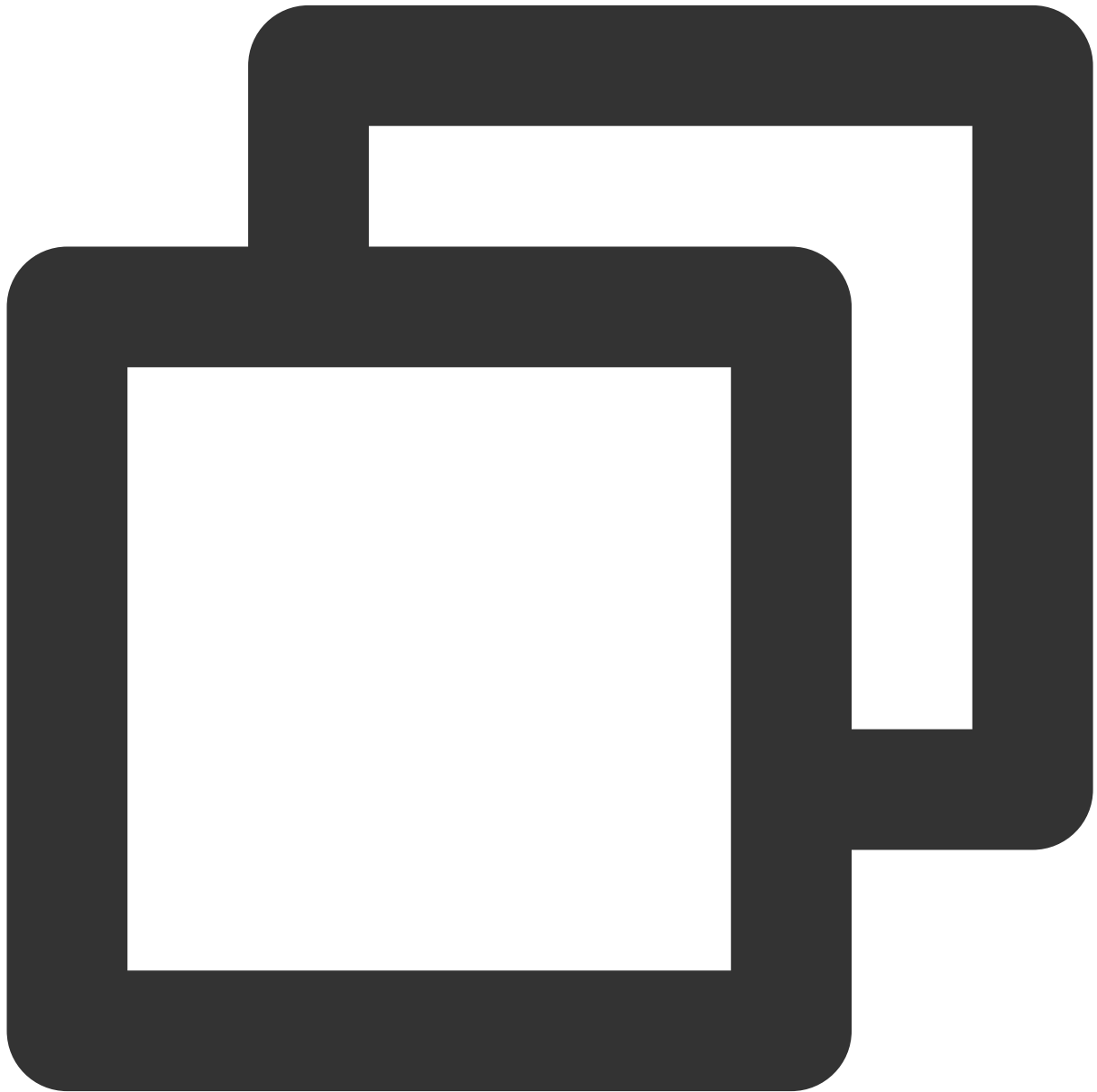
This API is used to play music (called after becoming a speaker).

#### Note

After music playback starts, you will receive an `onMusicPrepareToPlay` notification.

During music playback, all members in the room will continuously receive an `onMusicProgressUpdate` notification.

After music playback stops, you will receive an `onMusicCompletePlaying` notification.



```
public abstract void startPlayMusic(int musicID, String originalUrl, String accompa
```

The parameters are described below:

| Parameter    | Type   | Description                                  |
|--------------|--------|----------------------------------------------|
| musicID      | int    | The music ID.                                |
| originalUrl  | String | The absolute path of the vocal track.        |
| accompanyUrl | String | The absolute path of the instrumental track. |



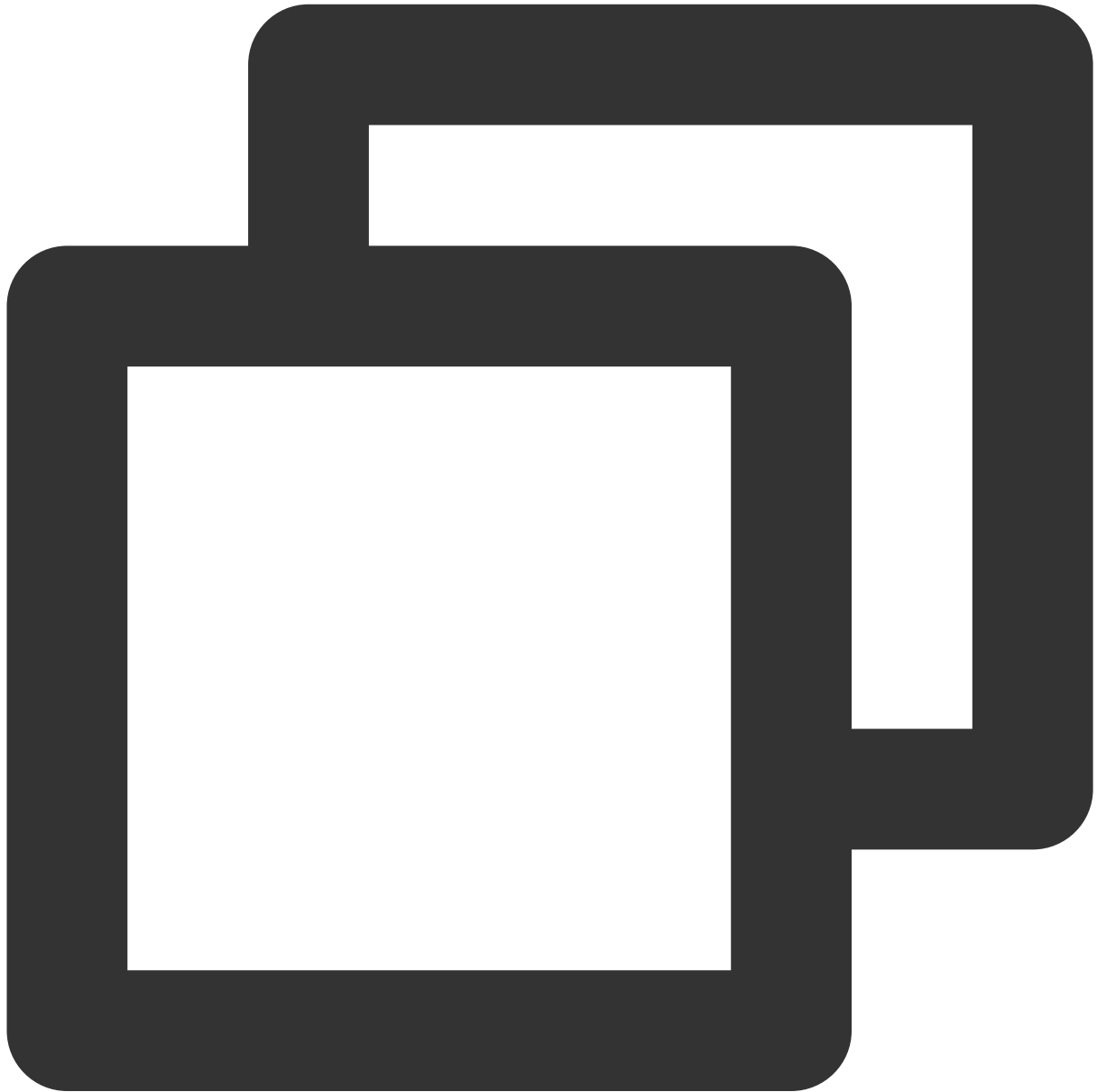
After this API is called, the song that is currently playing will stop.

### **stopPlayMusic**

This API is used to stop music (called during music playback).

#### **Note**

After music playback stops, you will receive an `onMusicCompletePlaying` notification.



```
public abstract void stopPlayMusic();
```

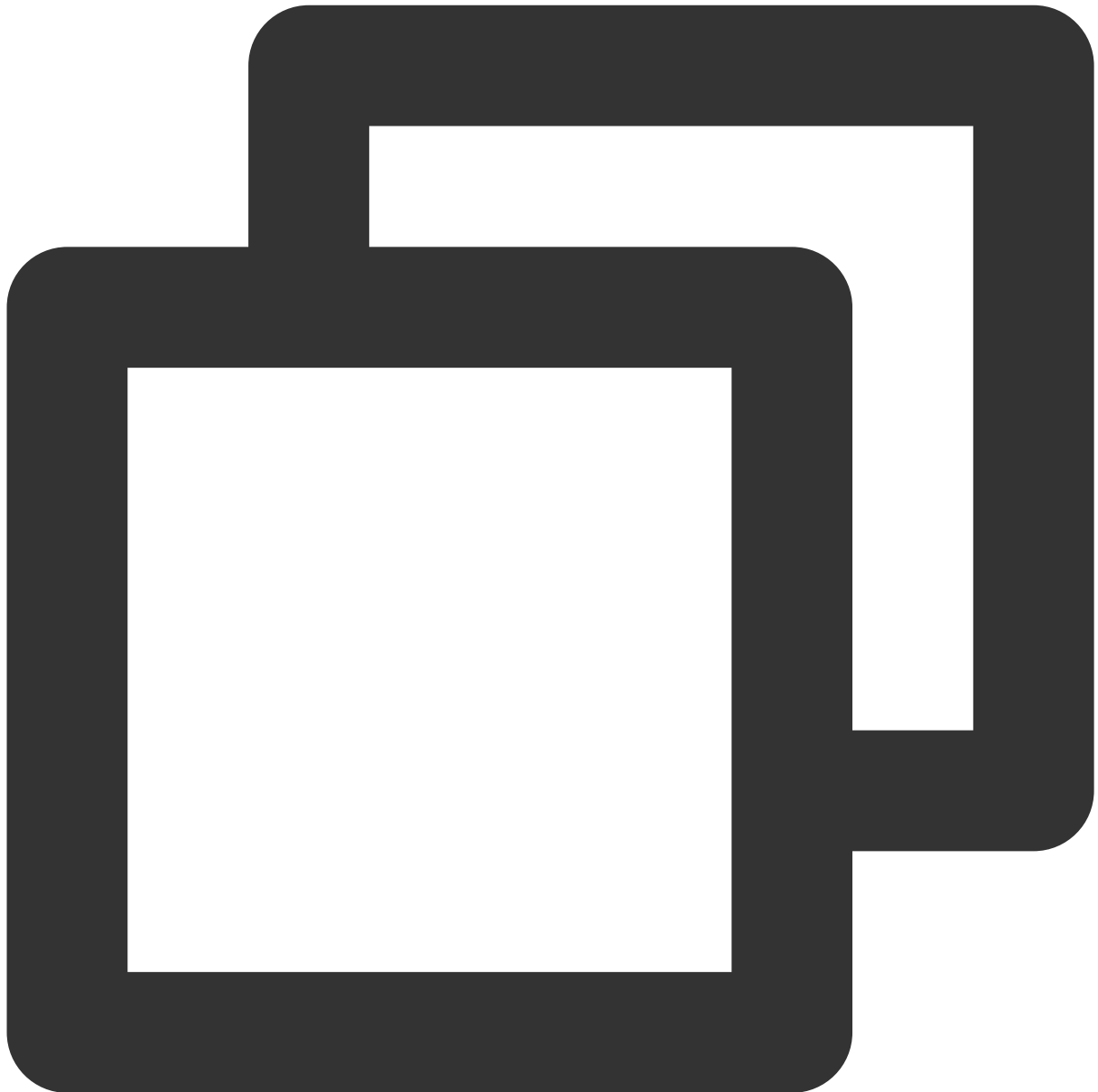
## pausePlayMusic

This API is used to pause music (called during music playback).

### Note

The `onMusicProgressUpdate` notification will be paused.

No `onMusicCompletePlaying` notification will be received.



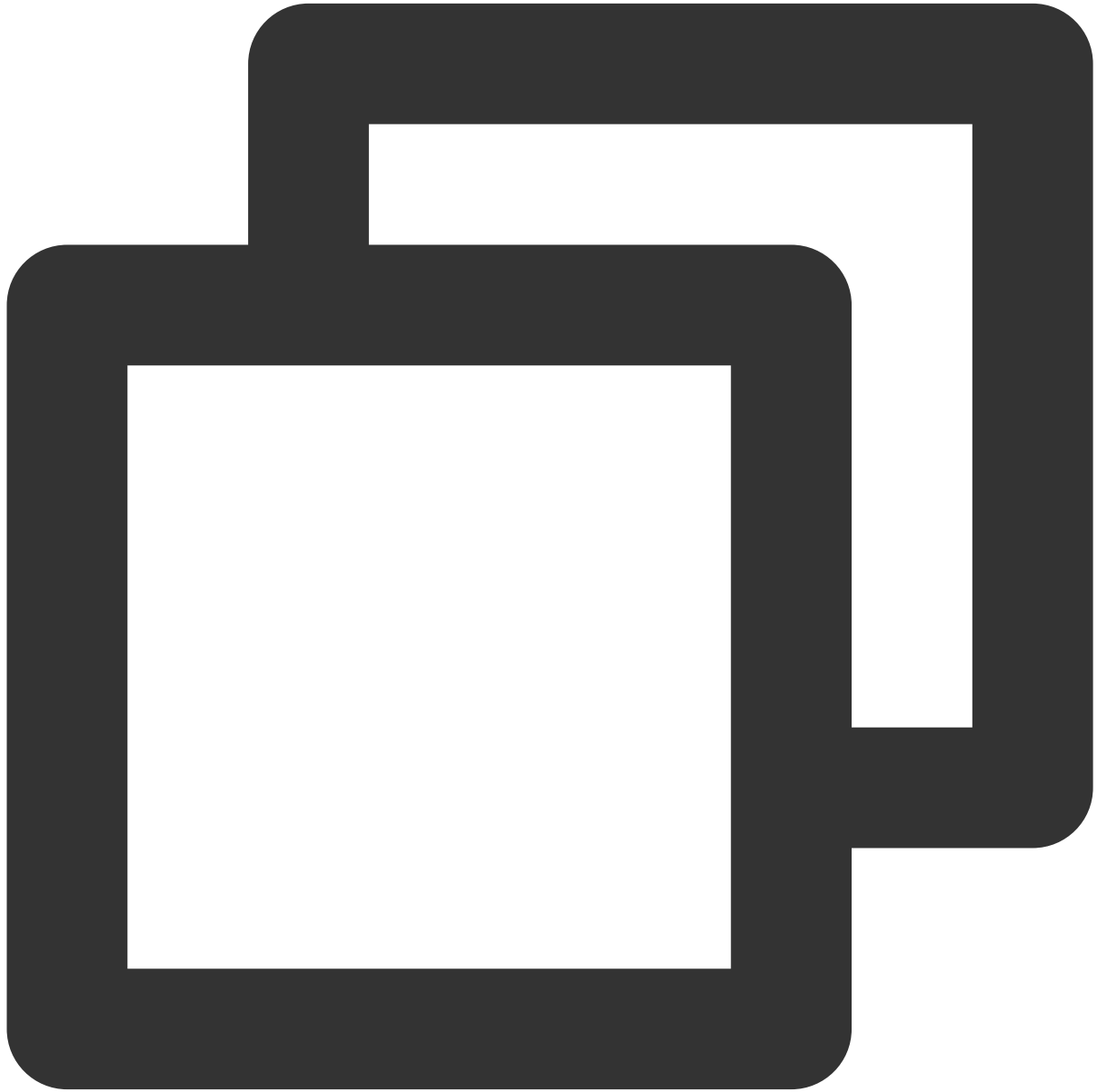
```
public abstract void pausePlayMusic();
```

## resumePlayMusic

This API is used to resume music (called after music playback is paused).

**Note**

No `onMusicPrepareToPlay` notification will be received.



```
public abstract void resumePlayMusic();
```

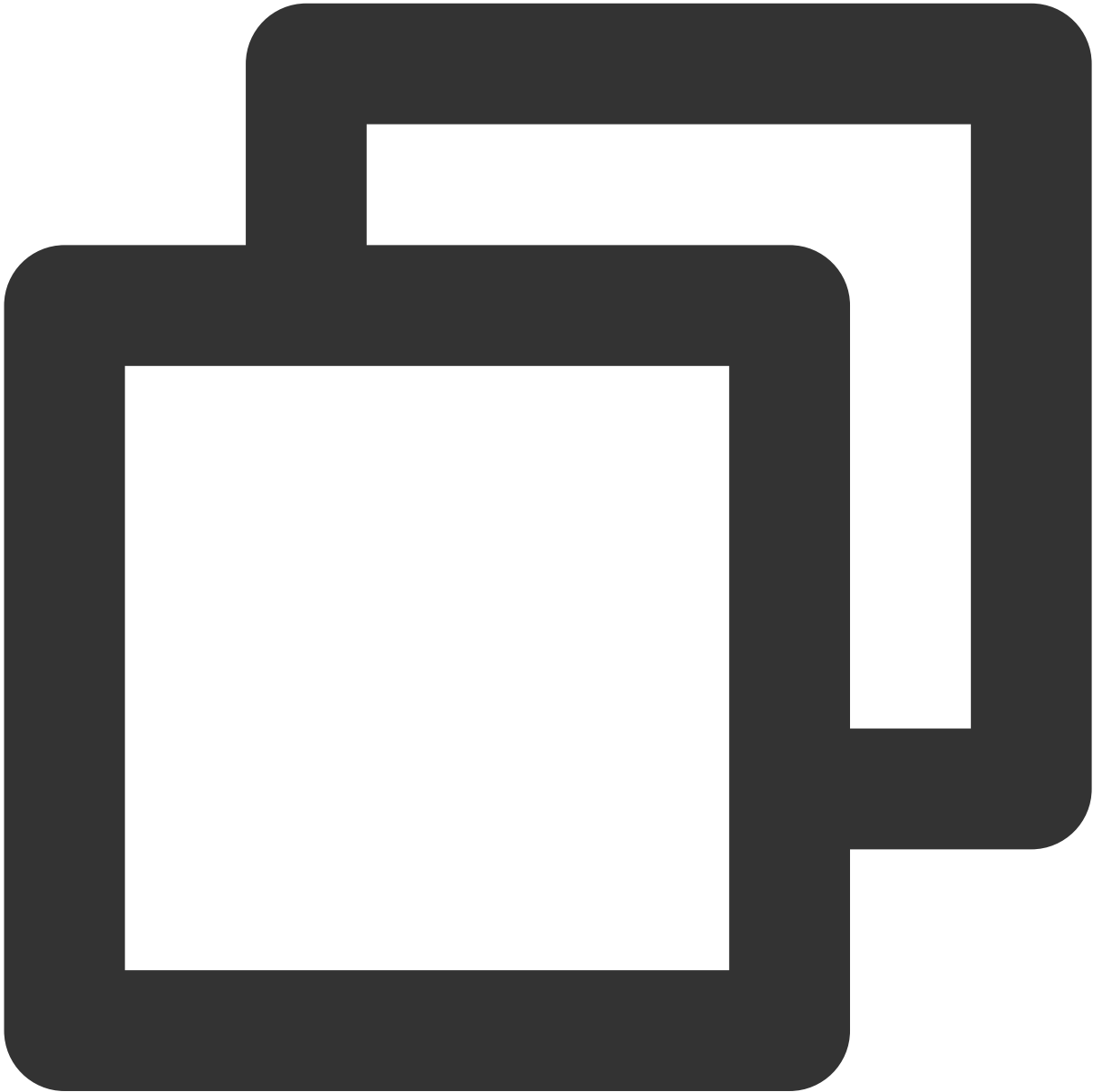
## Seat Management APIs

## enterSeat

This API is used to become a speaker (called by room owner or listener).

### Note

After a user becomes a speaker, all users in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.



```
public abstract void enterSeat(int seatIndex, TRTCKaraokeRoomCallback.ActionCallbac
```

The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter | Type           | Description                         |
|-----------|----------------|-------------------------------------|
| seatIndex | int            | The number of the seat to be taken. |
| callback  | ActionCallback | The callback for the operation.     |

Calling this API will immediately modify the seat list. In cases where listeners need the room owner's permission to take a seat, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call this API.

## leaveSeat

This API is used to become a listener (called by speaker).

### Note

After a speaker becomes a listener, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.



```
public abstract void leaveSeat (TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are described below:

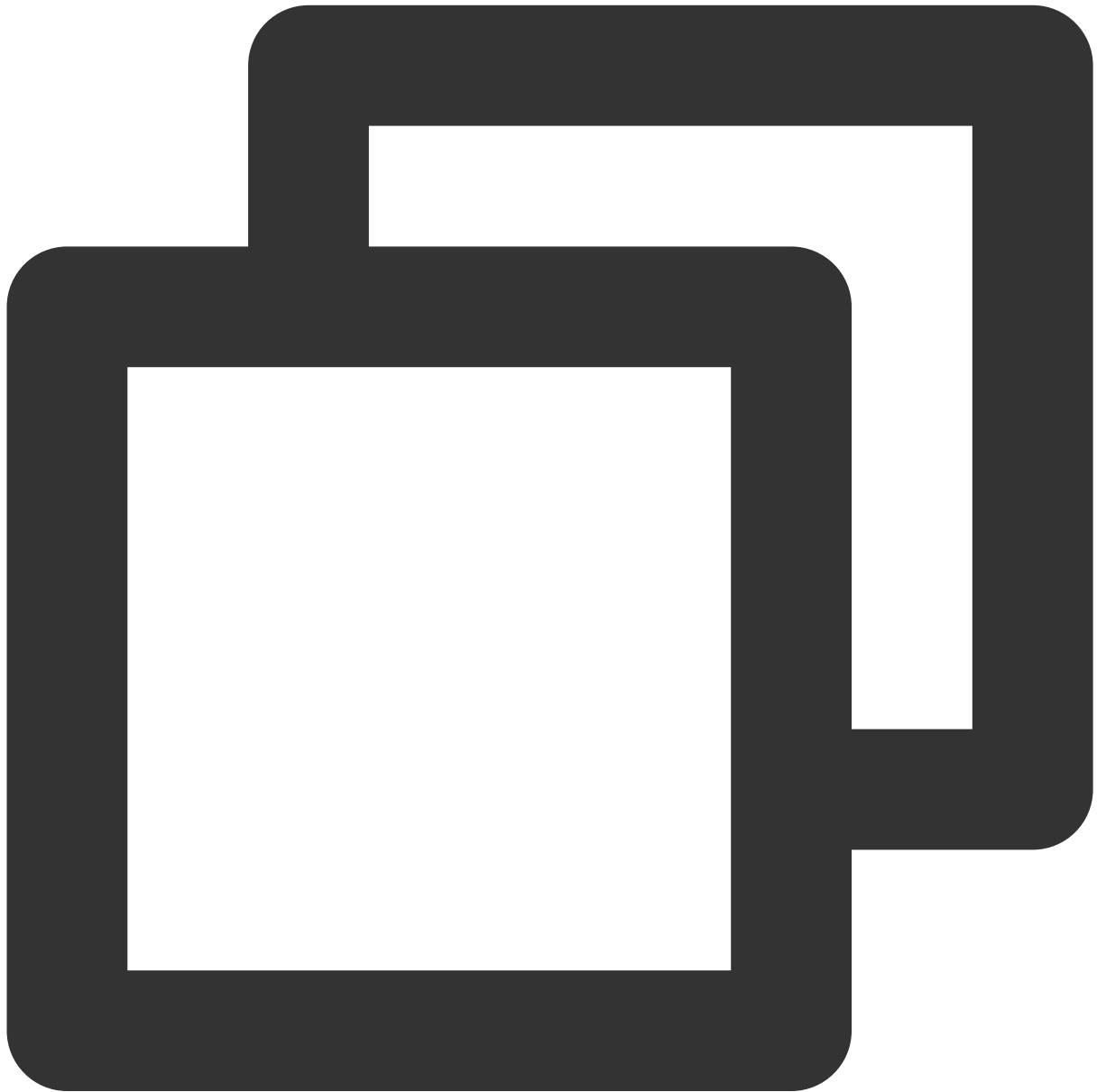
| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| callback  | ActionCallback | The callback for the operation. |

## **pickSeat**

This API is used to place a user in a seat (called by room owner).

**Note**

After the room owner makes someone a speaker, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.



```
public abstract void pickSeat(int seatIndex, String userId, TRTCKaraokeRoomCallback
```

The parameters are described below:

| Parameter | Type | Description                                      |
|-----------|------|--------------------------------------------------|
| seatIndex | int  | The number of the seat to place the listener in. |
|           |      |                                                  |

|          |                |                                 |
|----------|----------------|---------------------------------|
| userId   | String         | The user ID.                    |
| callback | ActionCallback | The callback for the operation. |

Calling this API will immediately modify the seat list. In cases where the room owner needs listeners' permission to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call `pickSeat`.

## kickSeat

This API is used to remove a speaker (called by room owner).

### Note

After a speaker is removed from a seat, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.





```
public abstract void kickSeat(int seatIndex, TRTCKaraokeRoomCallback.ActionCallback
```

The parameters are described below:

| Parameter | Type           | Description                                        |
|-----------|----------------|----------------------------------------------------|
| seatIndex | int            | The number of the seat to remove the speaker from. |
| callback  | ActionCallback | The callback for the operation.                    |

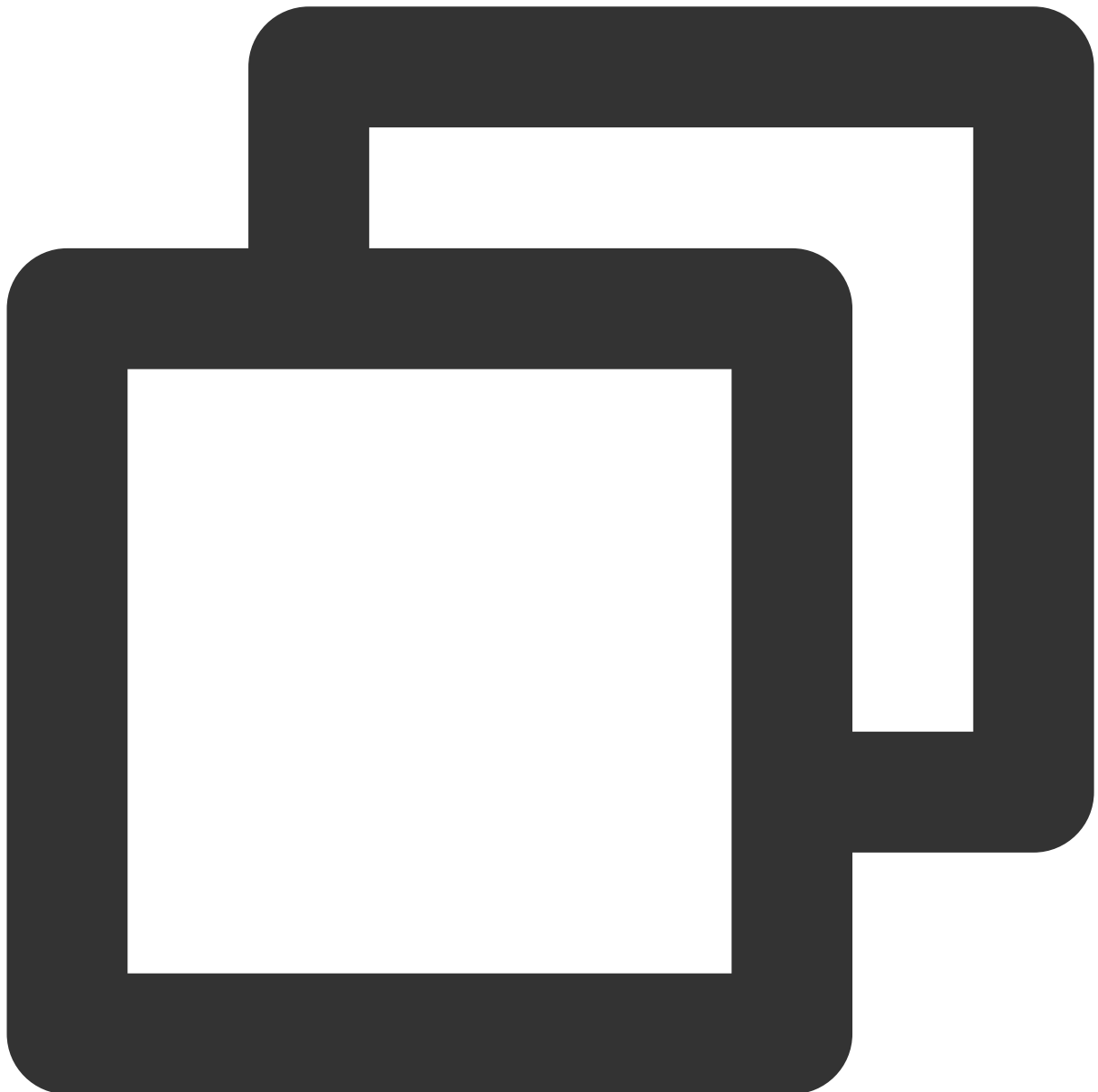
Calling this API will immediately modify the seat list.

## **muteSeat**

This API is used to mute/unmute a seat (called by room owner).

### **Note**

After a seat is muted/unmuted, all members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.



```
public abstract void muteSeat(int seatIndex, boolean isMute, TRTCKaraokeRoomCallbac
```

The parameters are described below:

| Parameter | Type           | Description                                           |
|-----------|----------------|-------------------------------------------------------|
| seatIndex | int            | The number of the seat to mute/unmute.                |
| isMute    | boolean        | <code>true</code> : Mute; <code>false</code> : Unmute |
| callback  | ActionCallback | The callback for the operation.                       |

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will call `muteAudio` to mute/unmute his or her audio.

## closeSeat

This API is used to block/unblock a seat (called by room owner).

### Note

After a seat is blocked/unblocked, all members in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.



```
public abstract void closeSeat(int seatIndex, boolean isClose, TRTCKaraokeRoomCallb
```

The parameters are described below:

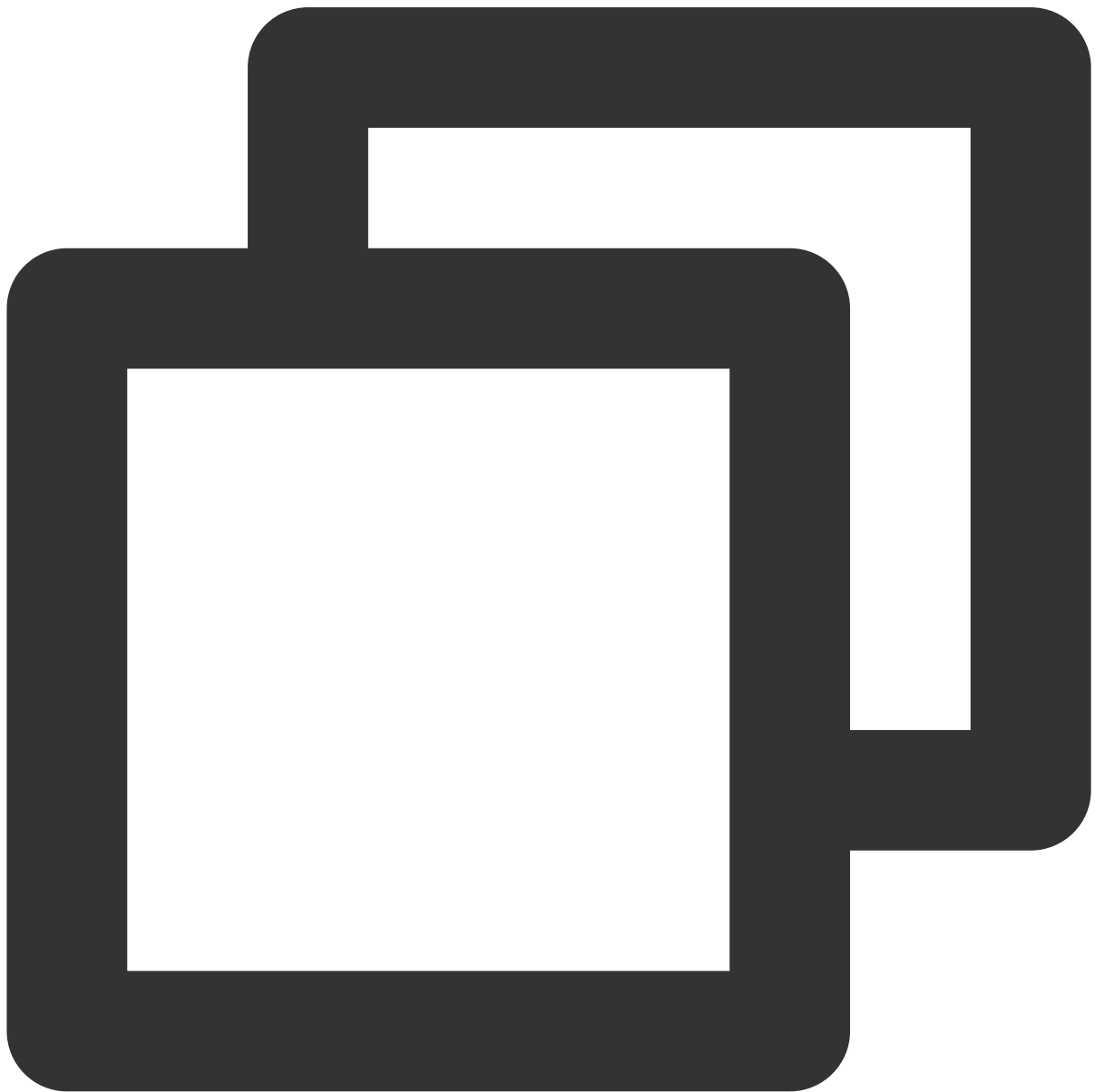
| Parameter | Type           | Description                                             |
|-----------|----------------|---------------------------------------------------------|
| seatIndex | int            | The number of the seat to block/unblock.                |
| isClose   | boolean        | <code>true</code> : Block; <code>false</code> : Unblock |
| callback  | ActionCallback | The callback for the operation.                         |

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will leave the seat.

## Local Audio APIs

### **startMicrophone**

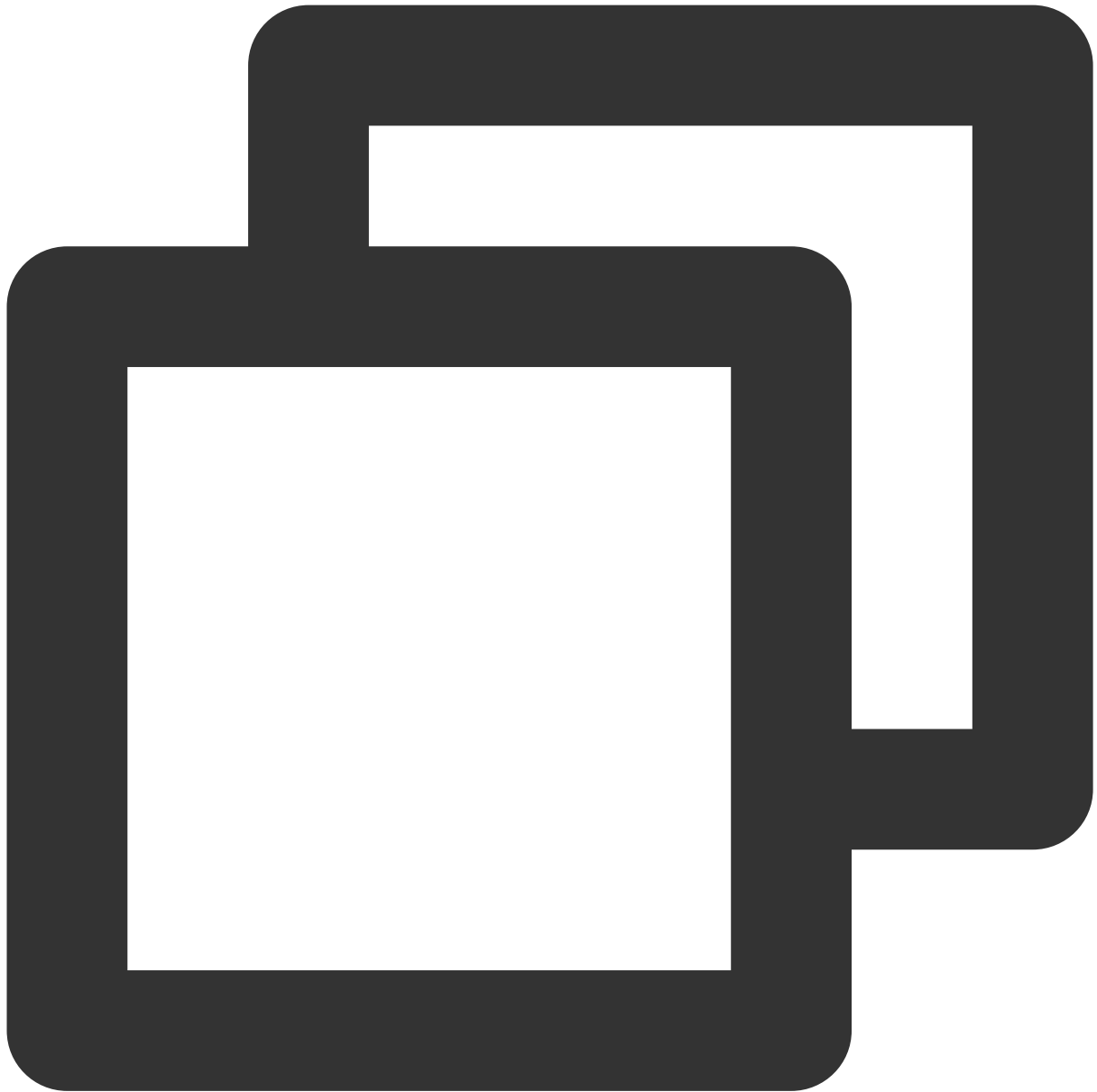
This API is used to start mic capturing.



```
public abstract void startMicrophone();
```

## stopMicrophone

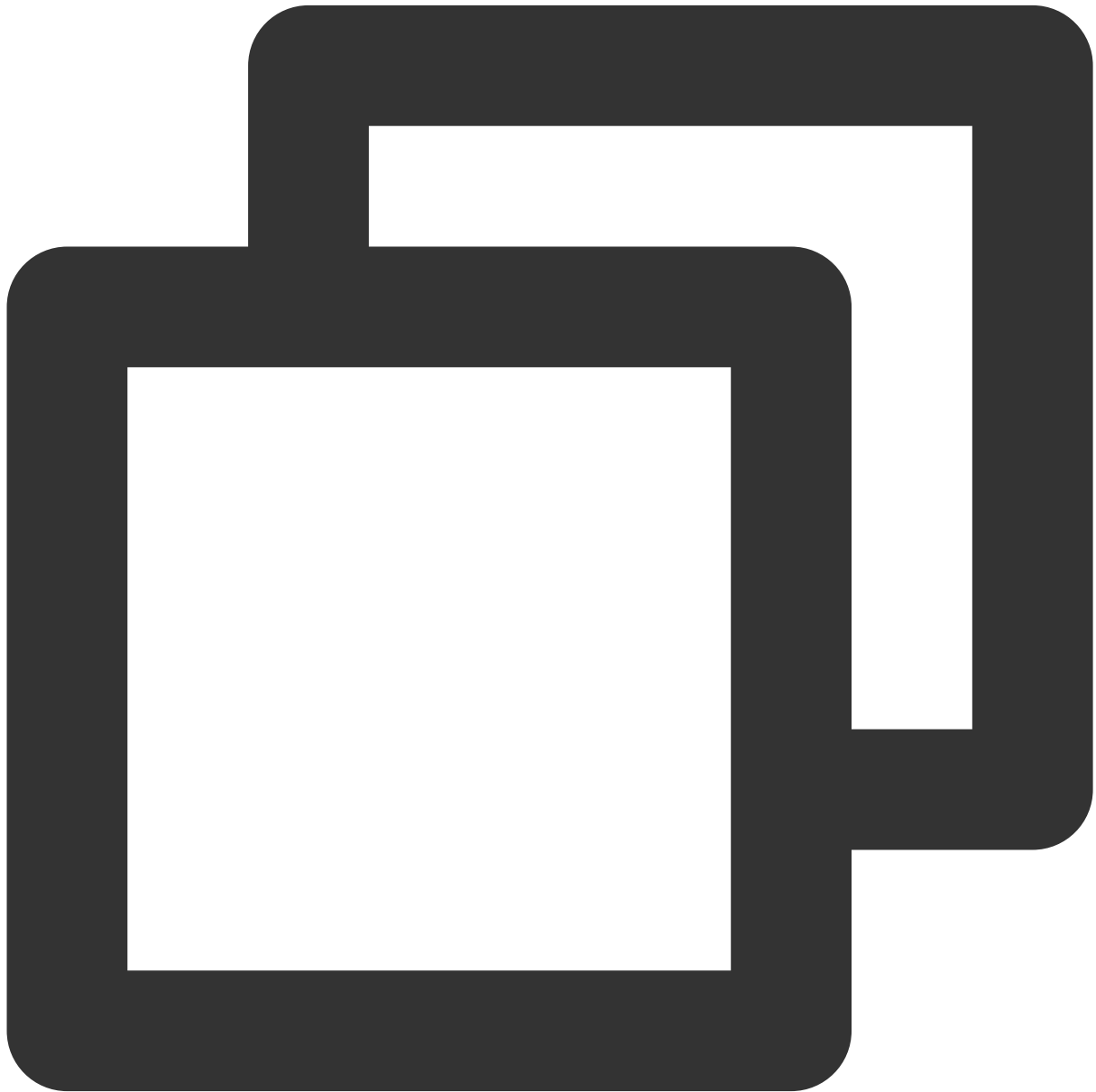
This API is used to stop mic capturing.



```
public abstract void stopMicrophone();
```

## setAudioQuality

This API is used to set audio quality.



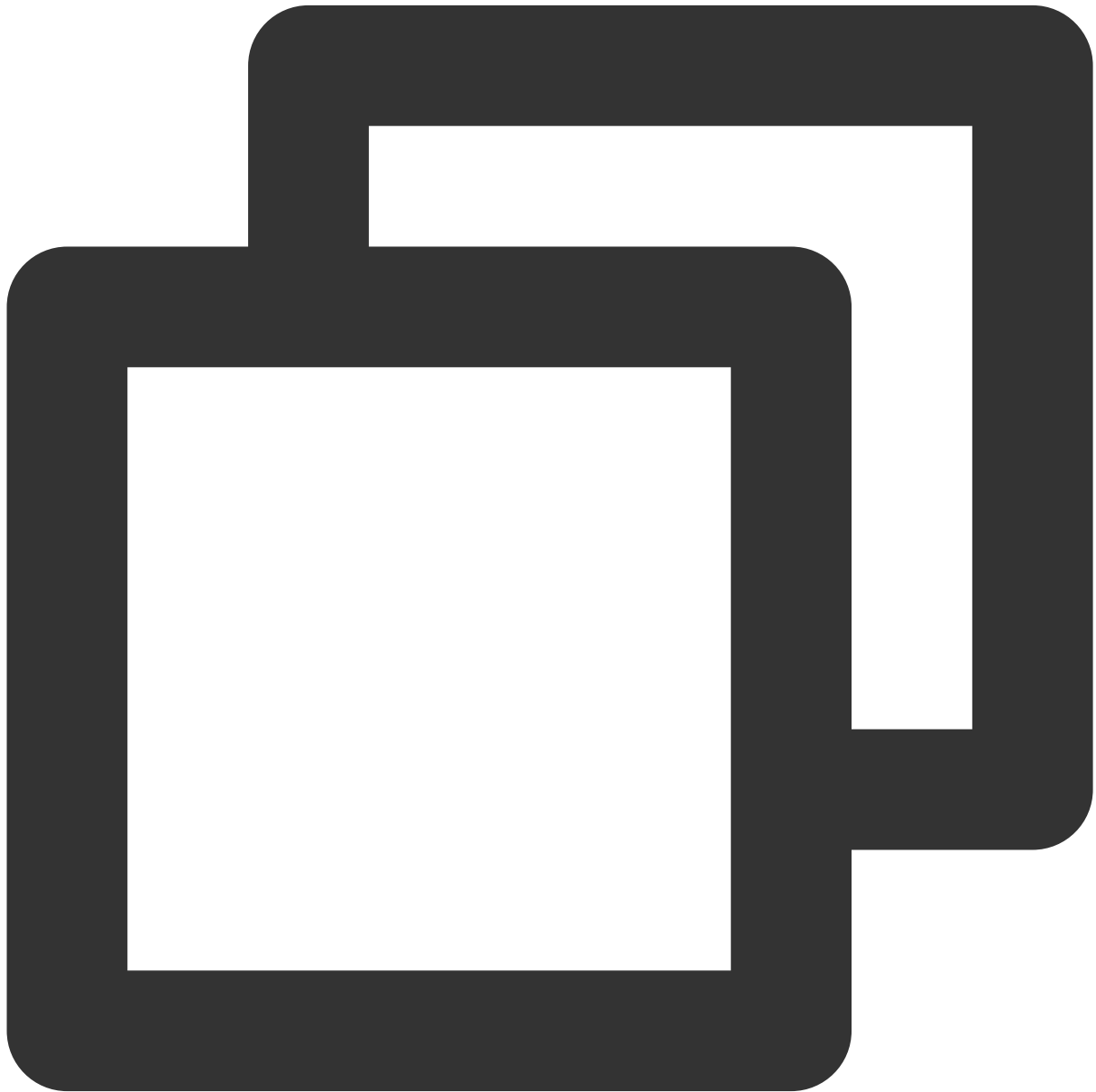
```
public abstract void setAudioQuality(int quality);
```

The parameters are described below:

| Parameter | Type | Description                                                                      |
|-----------|------|----------------------------------------------------------------------------------|
| quality   | int  | The audio quality. For more information, see <a href="#">setAudioQuality()</a> . |

## **muteLocalAudio**

This API is used to mute/unmute local audio.



```
public abstract void muteLocalAudio(boolean mute);
```

The parameters are described below:

| Parameter | Type    | Description                                                                                   |
|-----------|---------|-----------------------------------------------------------------------------------------------|
| mute      | boolean | Whether to mute or unmute audio. For more information, see <a href="#">muteLocalAudio()</a> . |

## setSpeaker

This API is used to set whether to play sound from the device's speaker or receiver.





```
public abstract void setSpeaker(boolean useSpeaker);
```

The parameters are described below:

| Parameter  | Type    | Description                      |
|------------|---------|----------------------------------|
| useSpeaker | boolean | true : Speaker; false : Receiver |

### setAudioCaptureVolume

This API is used to set the mic capturing volume.



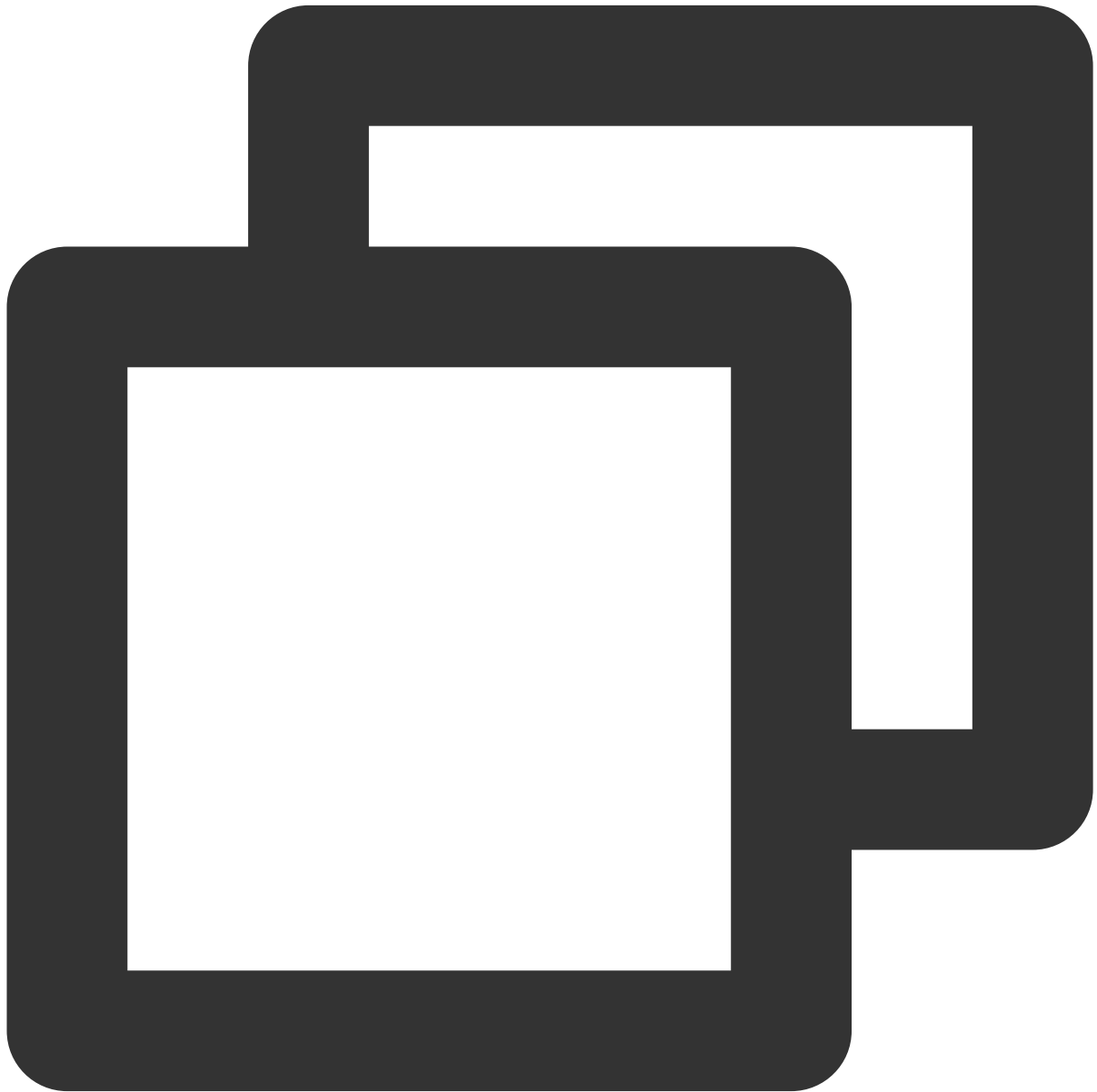
```
public abstract void setAudioCaptureVolume(int volume);
```

The parameters are described below:

| Parameter | Type | Description                                             |
|-----------|------|---------------------------------------------------------|
| volume    | int  | The capturing volume. Value range: 0-100 (default: 100) |

### **setAudioPlayoutVolume**

This API is used to set the playback volume.



```
public abstract void setAudioPlayoutVolume(int volume);
```

The parameters are described below:

| Parameter | Type | Description                                            |
|-----------|------|--------------------------------------------------------|
| volume    | int  | The playback volume. Value range: 0-100 (default: 100) |

## **muteRemoteAudio**

This API is used to mute/unmute a specified user.



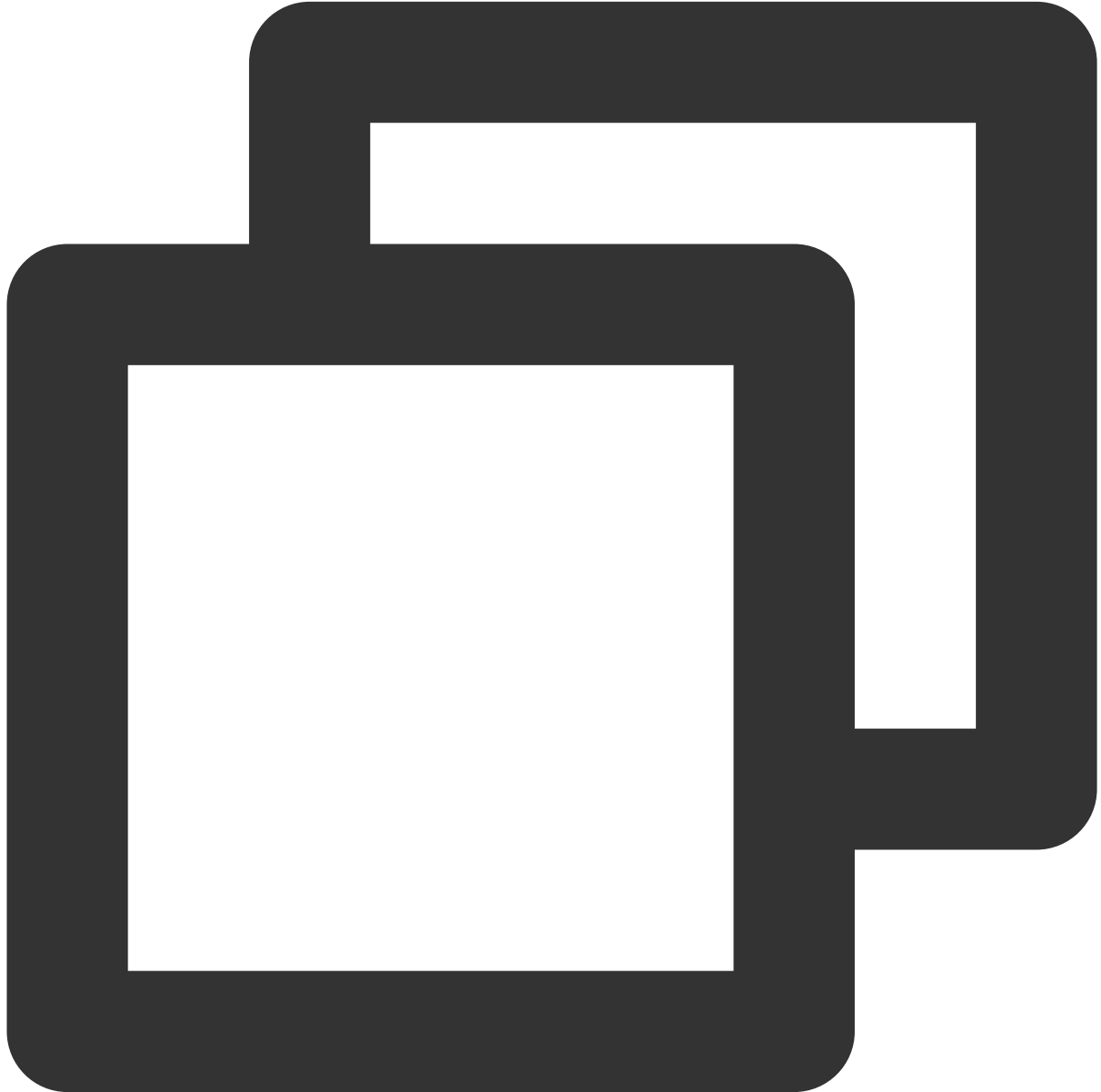
```
public abstract void muteRemoteAudio(String userId, boolean mute);
```

The parameters are described below:

| Parameter | Type    | Description                                           |
|-----------|---------|-------------------------------------------------------|
| userId    | String  | The user ID.                                          |
| mute      | boolean | <code>true</code> : Mute; <code>false</code> : Unmute |

## **muteAllRemoteAudio**

This API is used to mute/unmute all users.



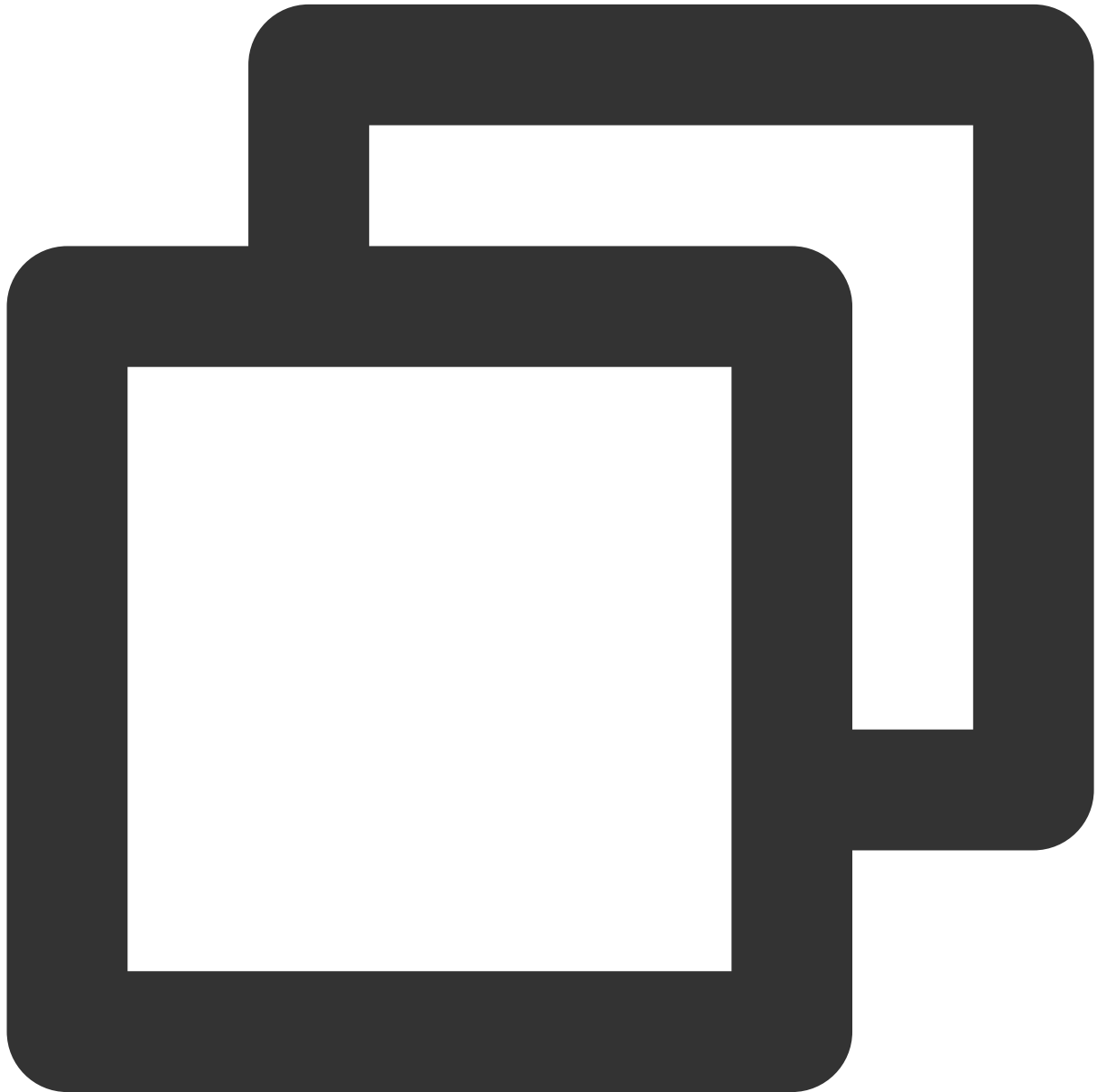
```
public abstract void muteAllRemoteAudio(boolean mute);
```

The parameters are described below:

| Parameter | Type    | Description                 |
|-----------|---------|-----------------------------|
| mute      | boolean | true : Mute; false : Unmute |

## setVoiceEarMonitorEnable

This API is used to enable/disable in-ear monitoring.



```
public abstract void setVoiceEarMonitorEnable(boolean enable);
```

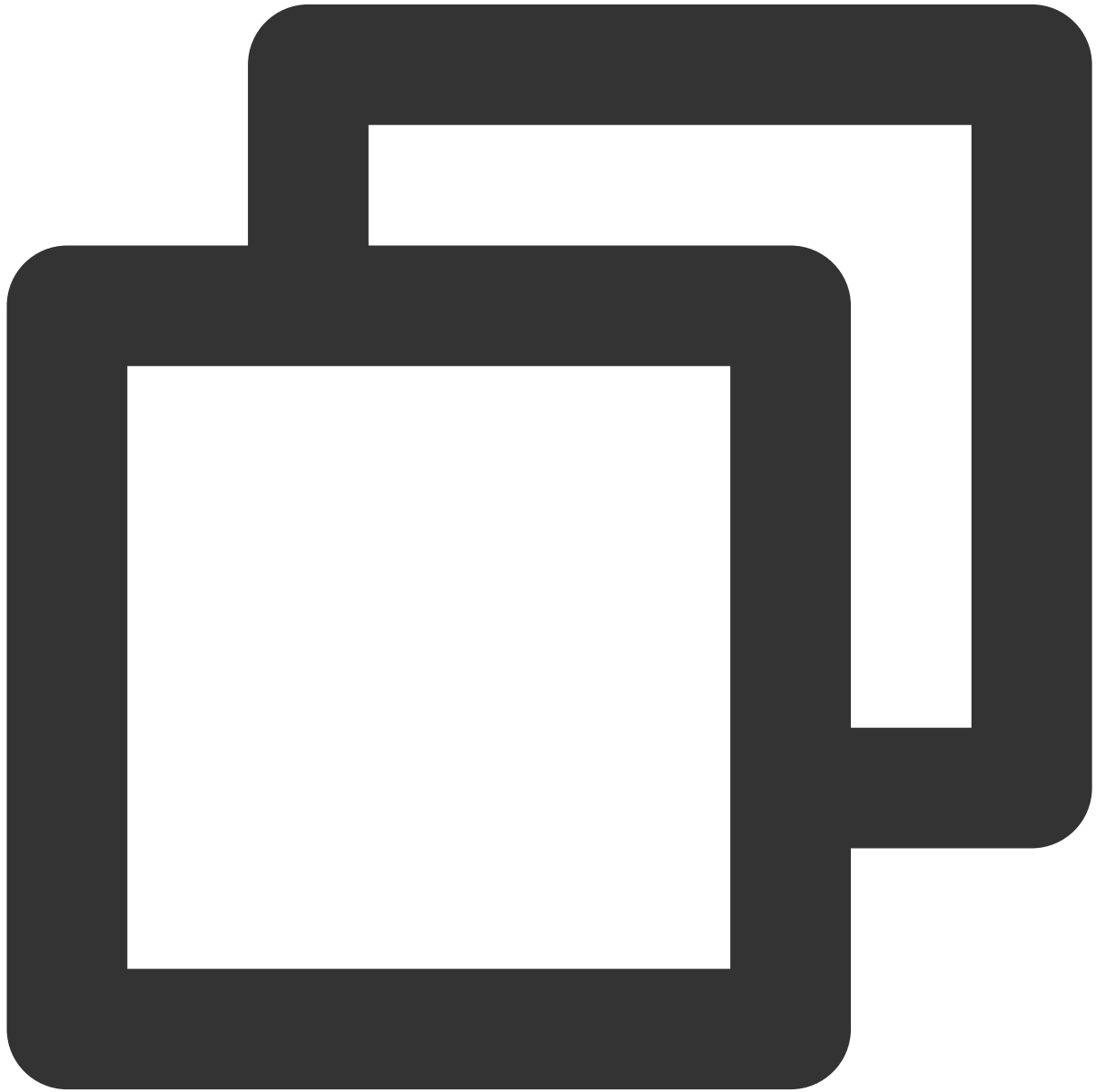
The parameters are described below:

| Parameter | Type    | Description                                              |
|-----------|---------|----------------------------------------------------------|
| enable    | boolean | <code>true</code> : Enable; <code>false</code> : Disable |

## Background Music and Audio Effect APIs

### **getAudioEffectManager**

This API is used to get the background music and audio effect management object [TXAudioEffectManager](#).

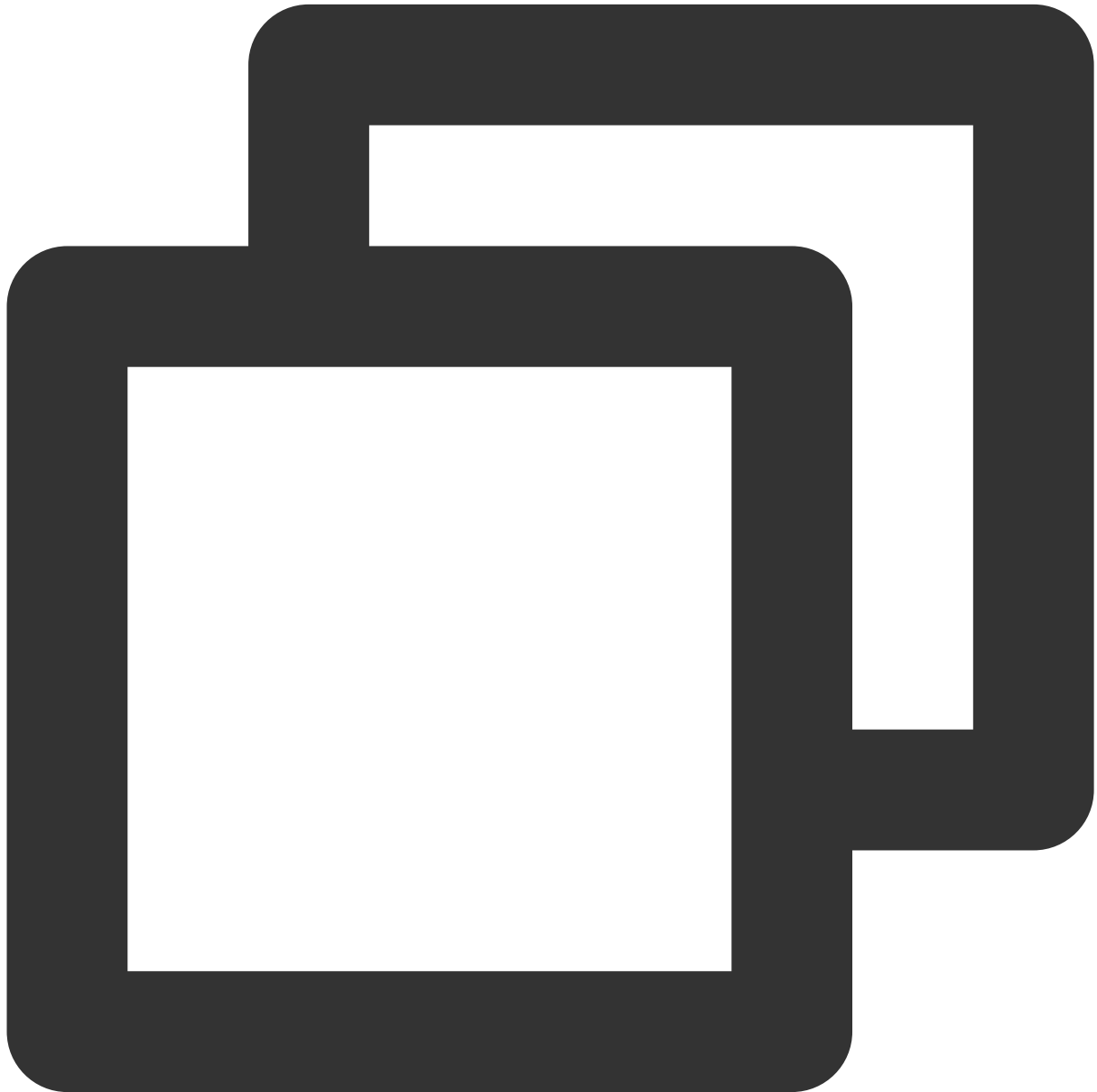


```
public abstract TXAudioEffectManager getAudioEffectManager();
```

## Message Sending APIs

## sendRoomTextMsg

This API is used to broadcast a text chat message in a room, which is generally used for on-screen comments.



```
public abstract void sendRoomTextMsg(String message, TRTCKaraokeRoomCallback.Action
```

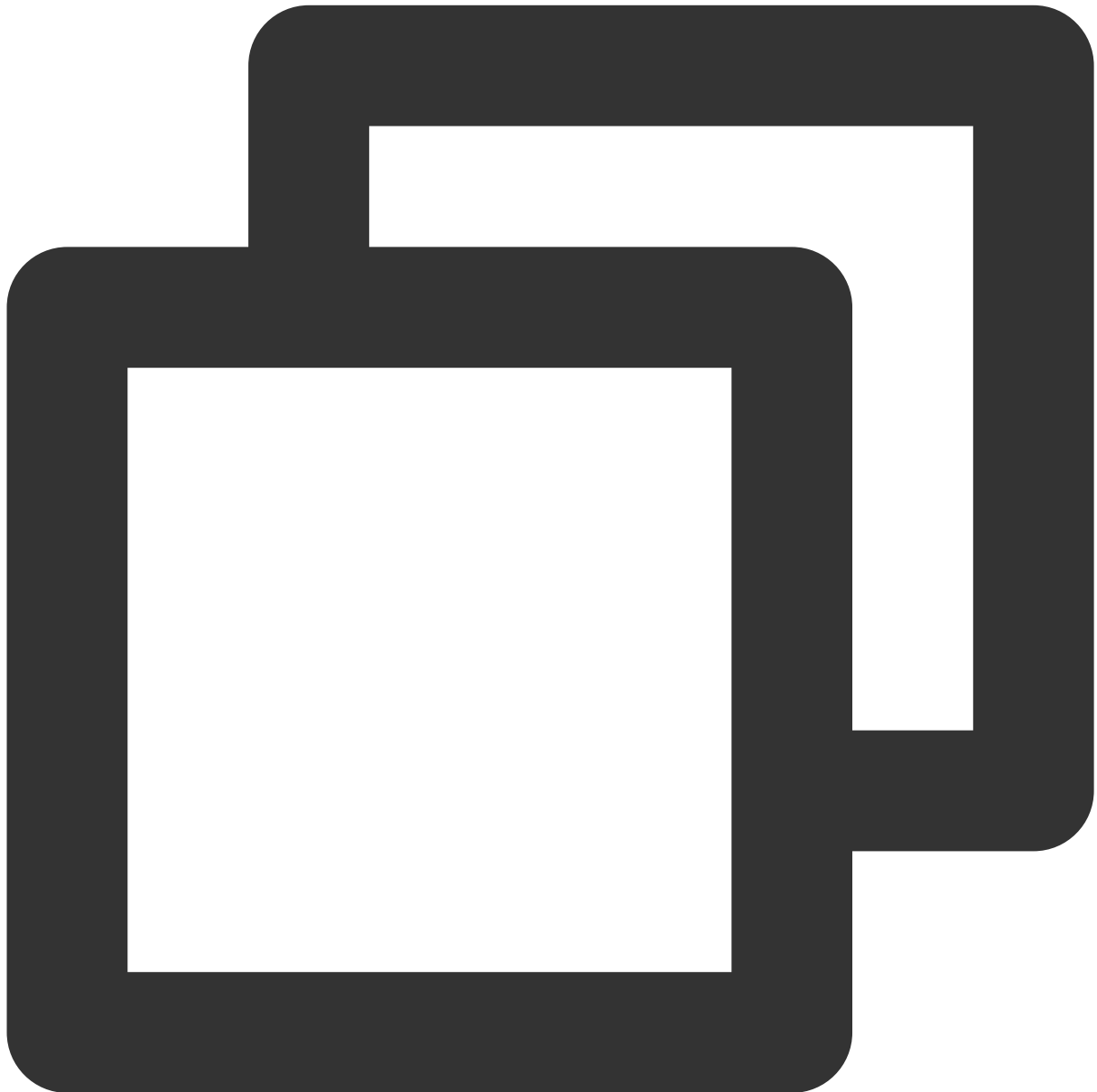
The parameters are described below:

| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| message   | String         | A text chat message.            |
| callback  | ActionCallback | The callback for the operation. |



## sendRoomCustomMsg

This API is used to send a custom text message.



```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCKaraokeRoomC
```

The parameters are described below:

| Parameter | Type   | Description                                                         |
|-----------|--------|---------------------------------------------------------------------|
| cmd       | String | A custom command word used to distinguish between different message |

|          |                |                                 |
|----------|----------------|---------------------------------|
|          |                | types.                          |
| message  | String         | A text chat message.            |
| callback | ActionCallback | The callback for the operation. |

## Invitation Signaling APIs

### **sendInvitation**

This API is used to send an invitation.



```
public abstract String sendInvitation(String cmd, String userId, String content, TR
```

The parameters are described below:

| Parameter | Type   | Description                    |
|-----------|--------|--------------------------------|
| cmd       | String | Custom command of business     |
| userId    | String | The user ID of the invitee.    |
| content   | String | The content of the invitation. |

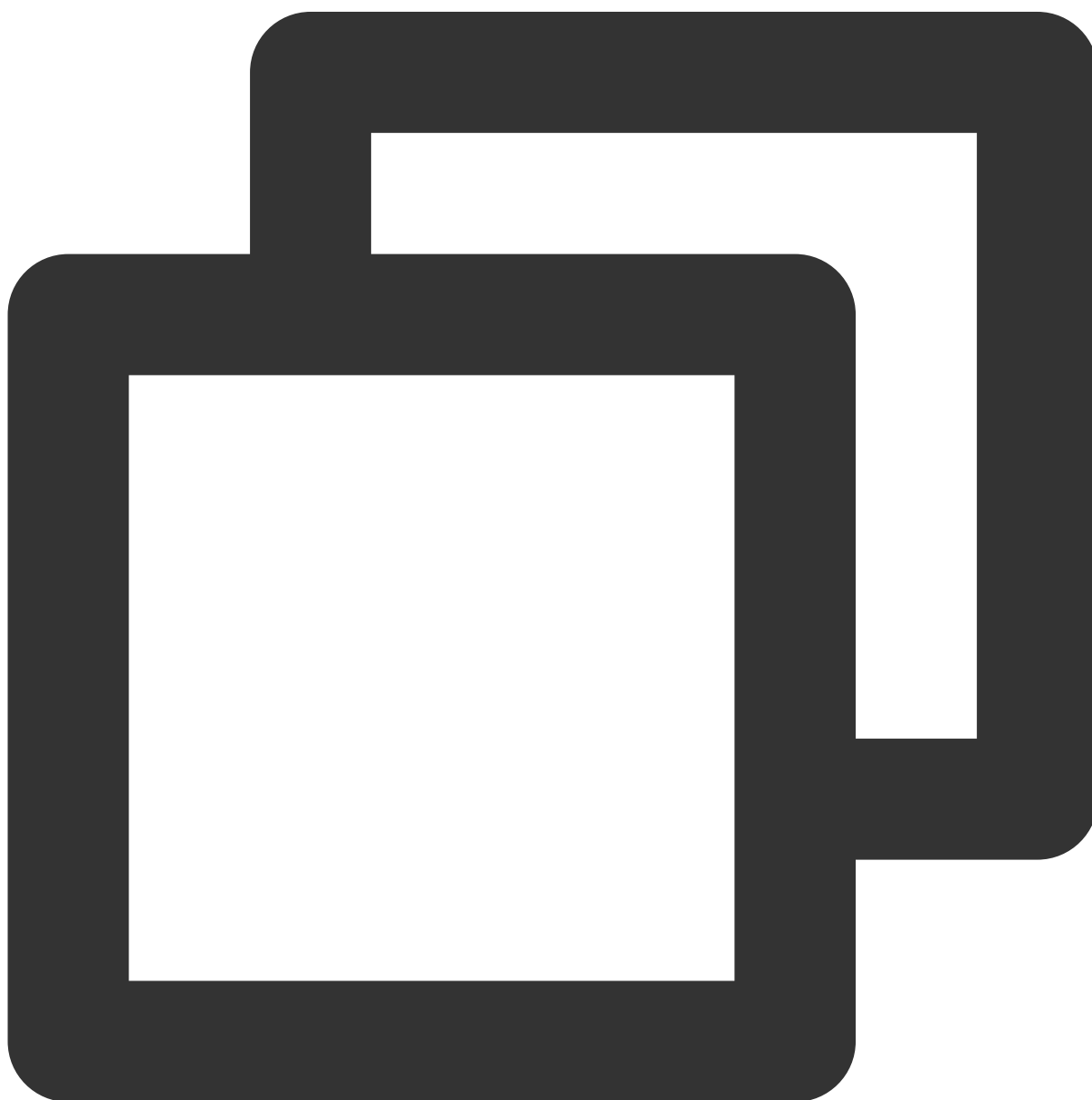
|          |                |                                 |
|----------|----------------|---------------------------------|
| callback | ActionCallback | The callback for the operation. |
|----------|----------------|---------------------------------|

Response parameters:

| Parameter | Type   | Description        |
|-----------|--------|--------------------|
| inviteId  | String | The invitation ID. |

## acceptInvitation

This API is used to accept an invitation.



```
public abstract void acceptInvitation(String id, TRTCKaraokeRoomCallback.ActionCall
```

The parameters are described below:

| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| id        | String         | The invitation ID.              |
| callback  | ActionCallback | The callback for the operation. |

## rejectInvitation

This API is used to decline an invitation.



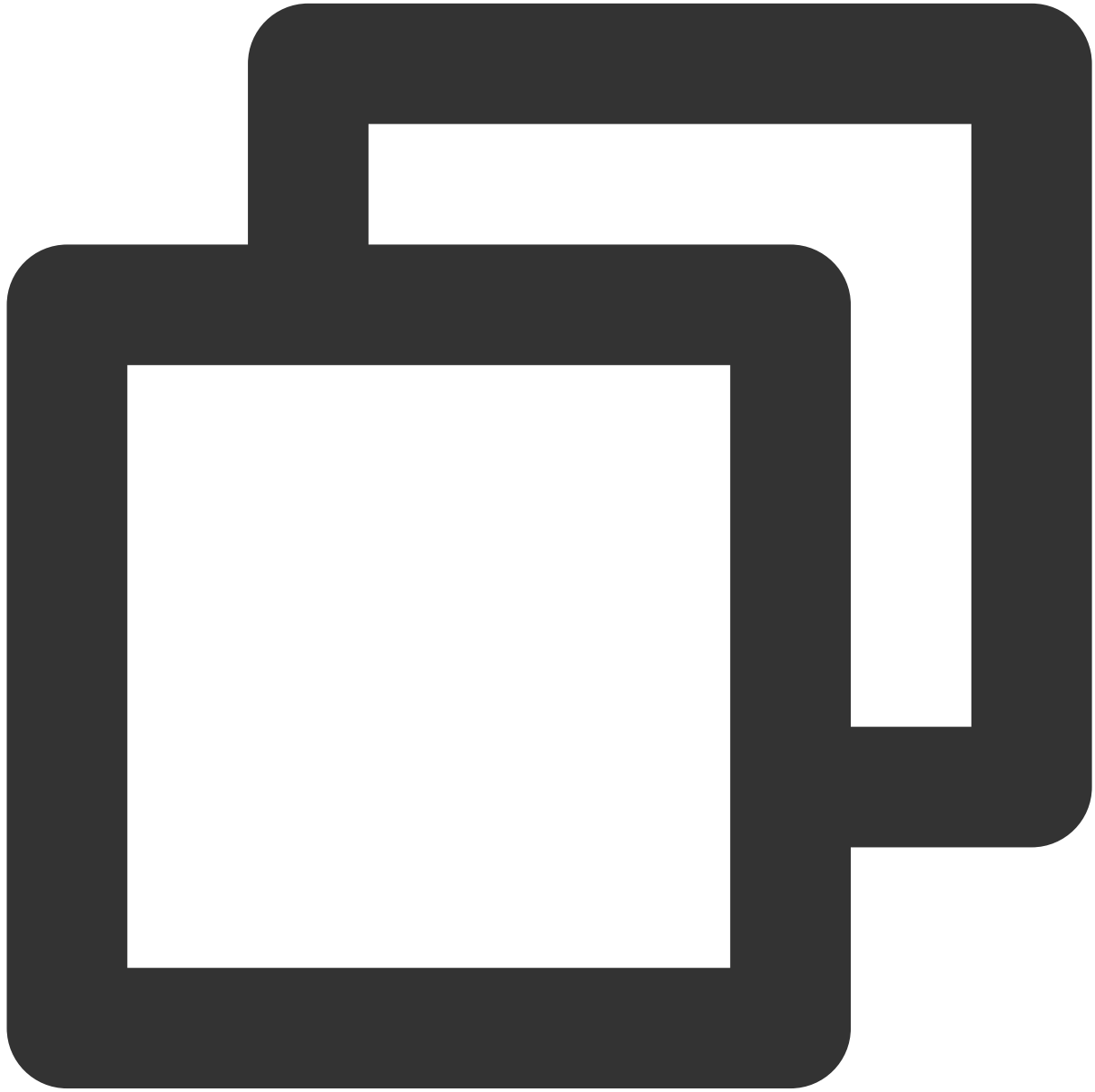
```
public abstract void rejectInvitation(String id, TRTCKaraokeRoomCallback.ActionCall
```

The parameters are described below:

| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| id        | String         | Invitation ID                   |
| callback  | ActionCallback | The callback for the operation. |

## cancelInvitation

This API is used to cancel an invitation.



```
public abstract void cancelInvitation(String id, TRTCKaraokeRoomCallback.ActionCall
```

The parameters are described below:

| Parameter | Type           | Description                     |
|-----------|----------------|---------------------------------|
| id        | String         | The invitation ID.              |
| callback  | ActionCallback | The callback for the operation. |

## TRTCKaraokeRoomDelegate Event Callback APIs

### Common Event Callback APIs

#### **onError**

Callback for error.

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users depending if necessary.





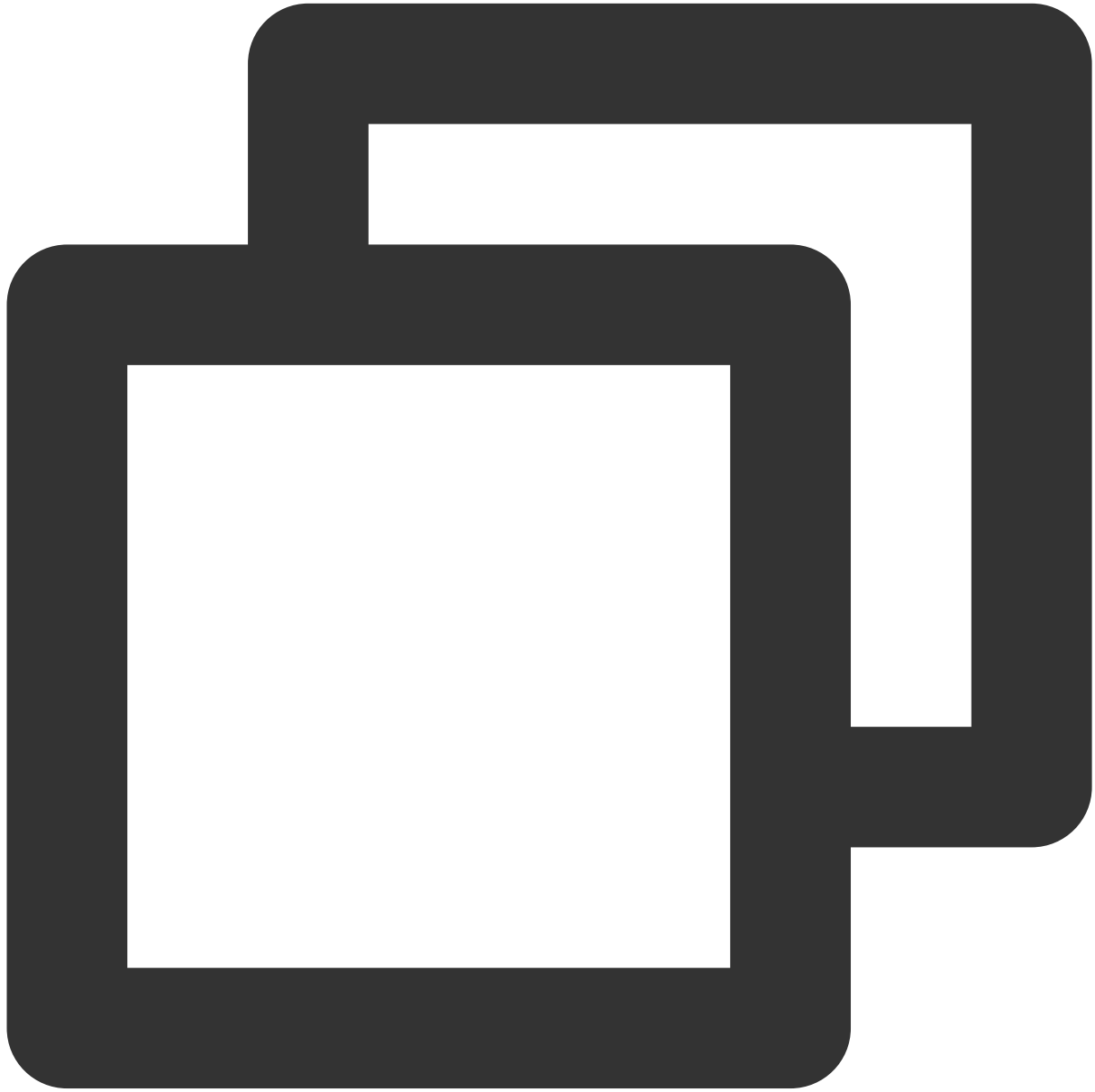
```
void onError(int code, String message);
```

The parameters are described below:

| Parameter | Type   | Description        |
|-----------|--------|--------------------|
| code      | int    | The error code.    |
| message   | String | The error message. |

## onWarning

Callback for warning.



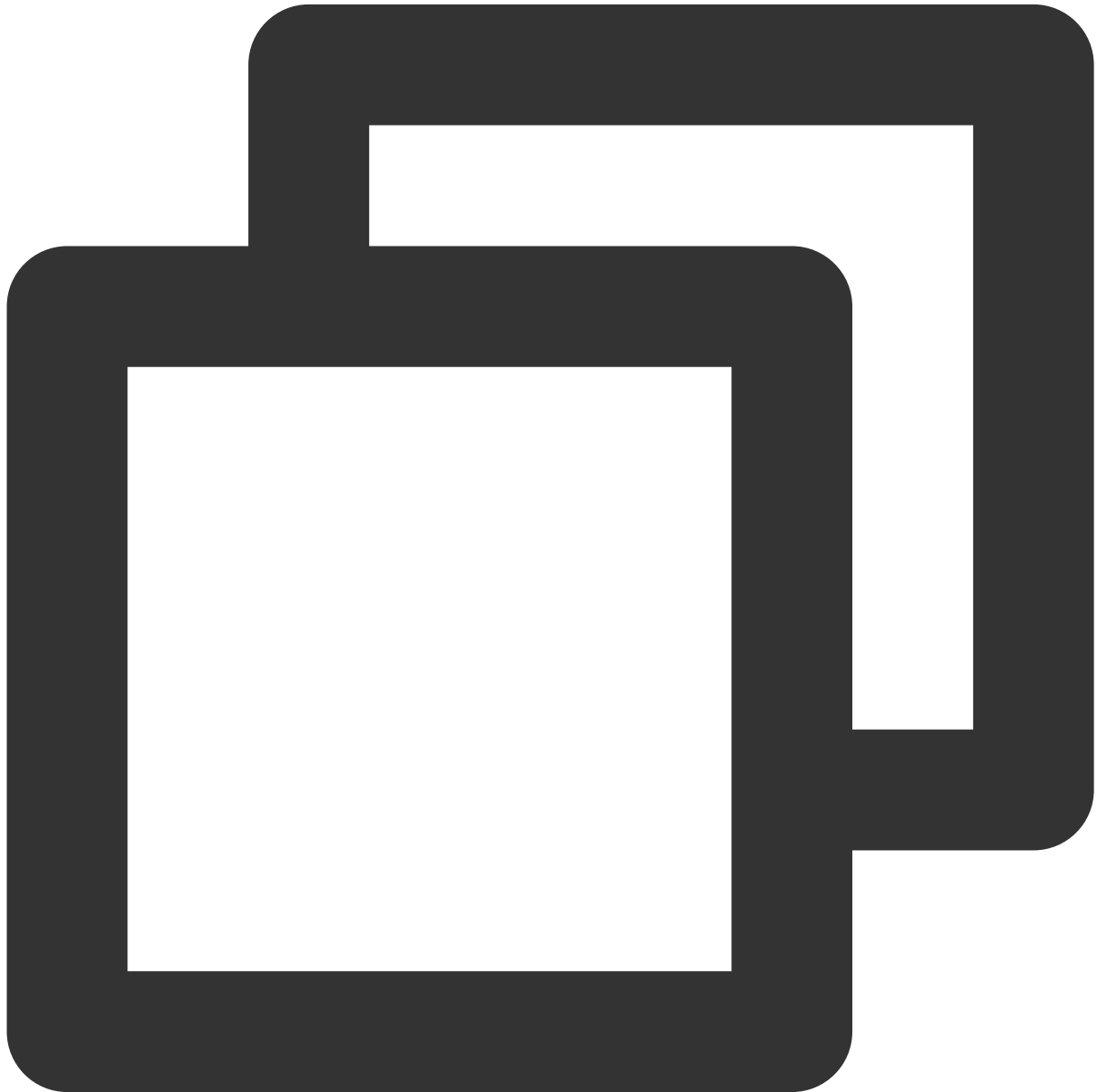
```
void onWarning(int code, String message);
```

The parameters are described below:

| Parameter | Type   | Description          |
|-----------|--------|----------------------|
| code      | int    | The error code.      |
| message   | String | The warning message. |

## onDebugLog

Callback for log.



```
void onDebugLog(String message);
```

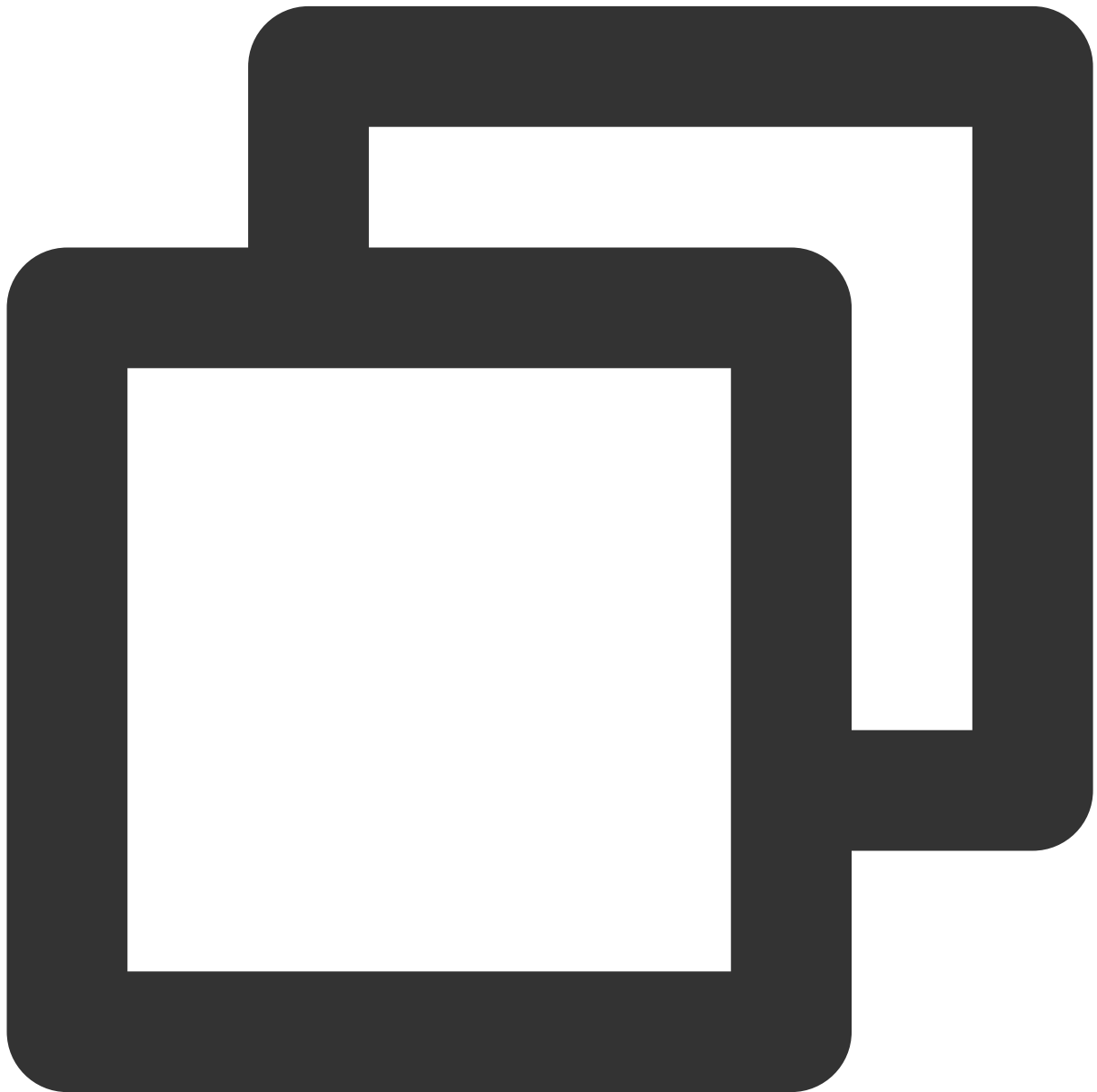
The parameters are described below:

| Parameter | Type   | Description      |
|-----------|--------|------------------|
| message   | String | Log information. |

## Room Event Callback APIs

### onRoomDestroy

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.



```
void onRoomDestroy(String roomId);
```

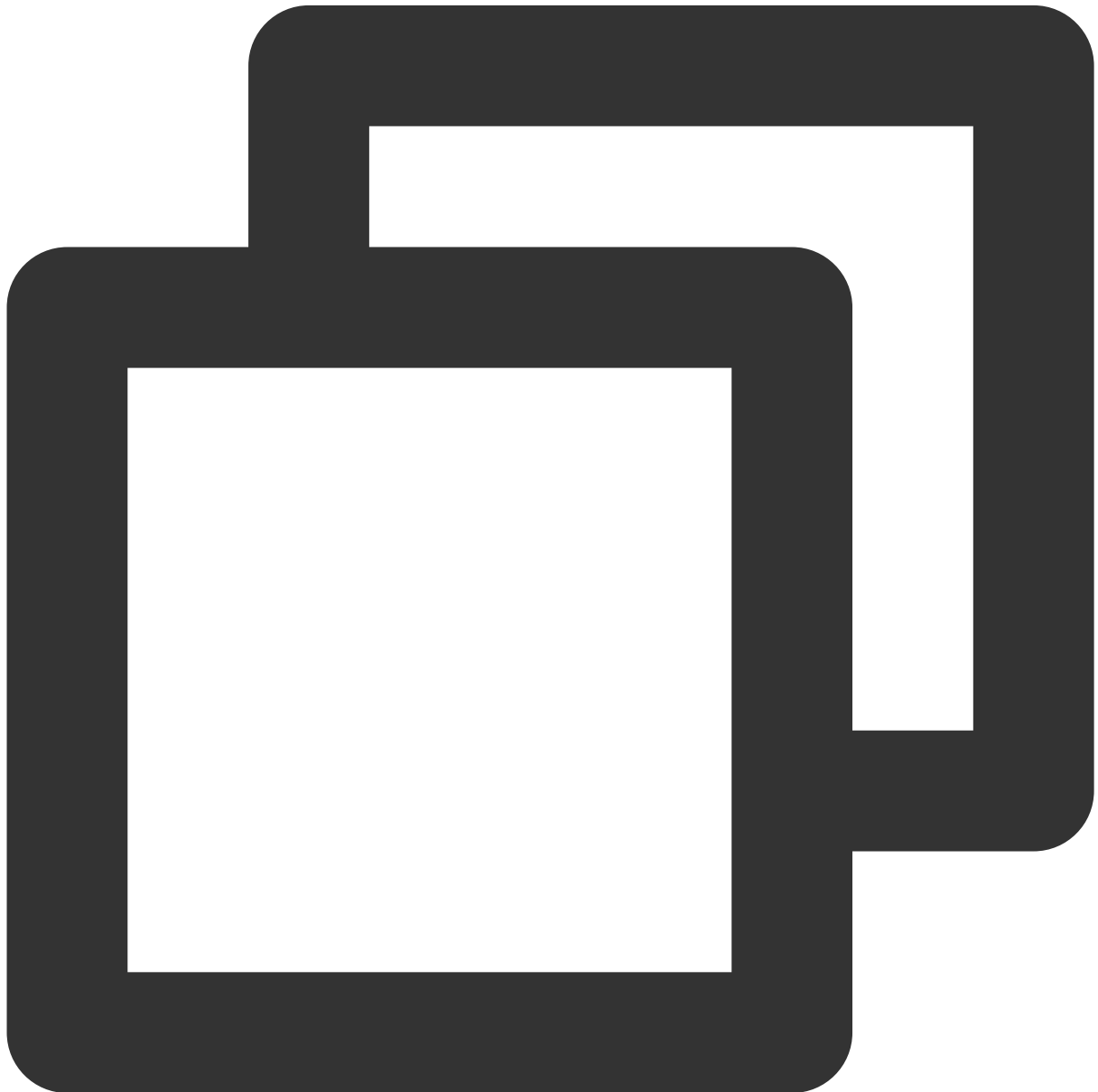
The parameters are described below:

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

|        |        |              |
|--------|--------|--------------|
| roomId | String | The room ID. |
|--------|--------|--------------|

## onRoomInfoChange

Callback for change of room information. This callback is sent after successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.



```
void onRoomInfoChange (TRTCKaraokeRoomDef.RoomInfo roomInfo);
```

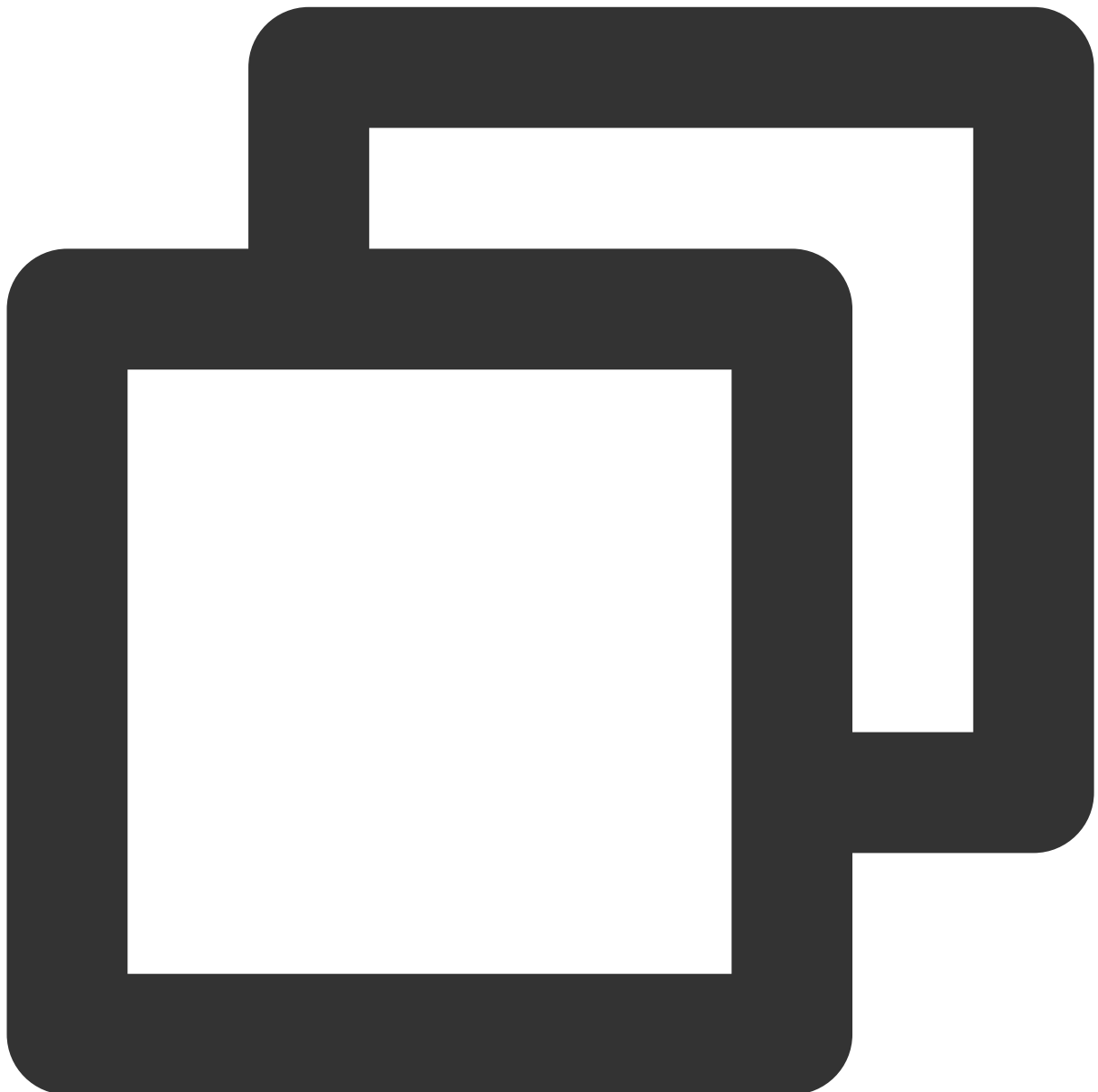
The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter | Type     | Description       |
|-----------|----------|-------------------|
| roomInfo  | RoomInfo | Room information. |

### onUserMicrophoneMute

Callback of whether a user's mic is muted. When a user calls `muteLocalAudio`, all members in the room will receive this callback.



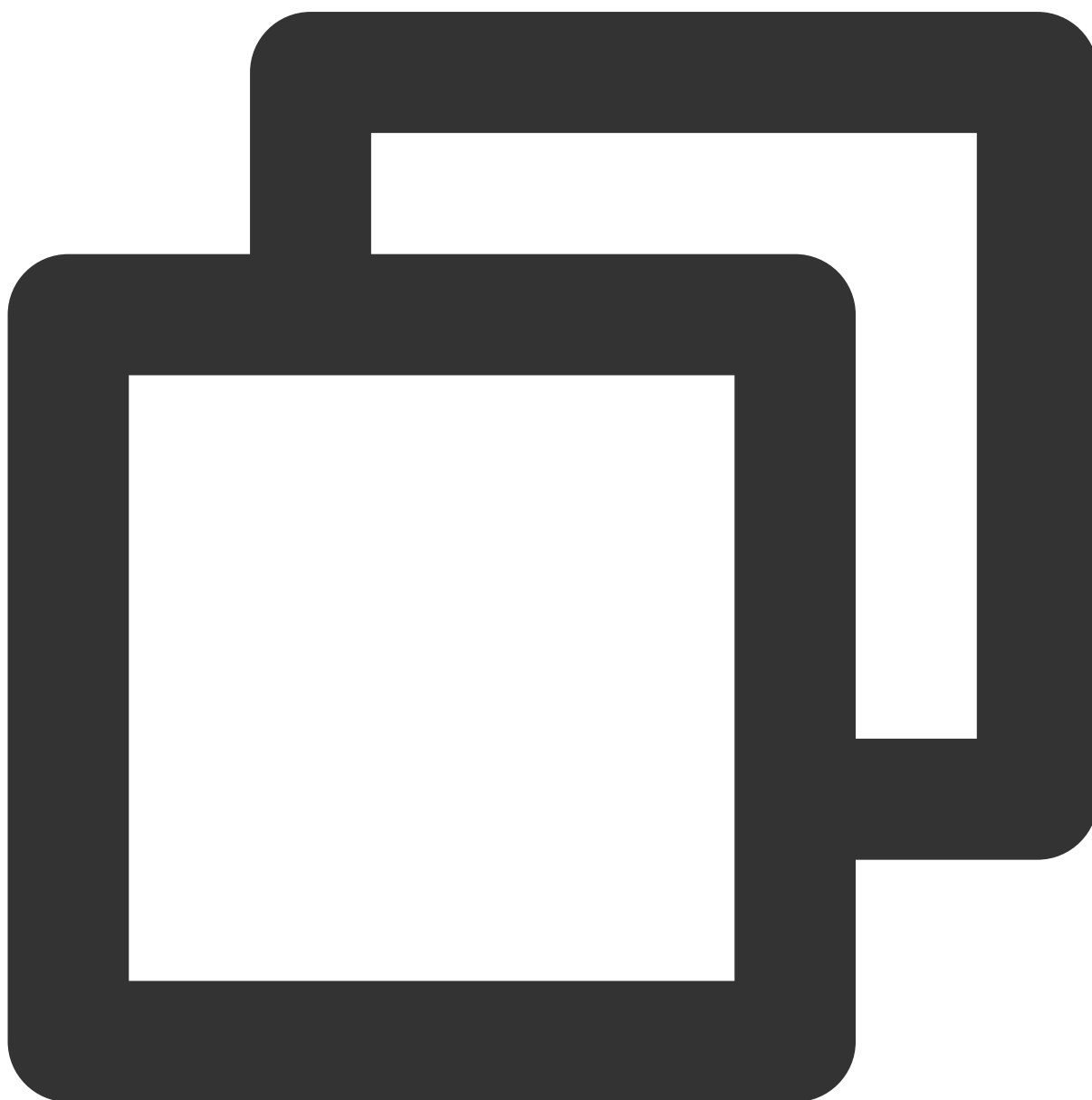
```
void onUserMicrophoneMute(String userId, boolean mute);
```

The parameters are described below:

| Parameter | Type    | Description                          |
|-----------|---------|--------------------------------------|
| userId    | String  | The user ID.                         |
| mute      | boolean | The volume level. Value range: 0-100 |

### **onUserVolumeUpdate**

Notification to all members of the volume after the volume reminder is enabled.



```
void onUserVolumeUpdate(List<TRTCCloudDef.TRTCVolumeInfo> userVolumes, int totalVol
```

The parameters are described below:

| Parameter   | Type | Description                          |
|-------------|------|--------------------------------------|
| userVolumes | List | List of user volumes.                |
| totalVolume | int  | The total volume. Value range: 0-100 |

## Seat Callback APIs

### onSeatListChange

Callback for all seat changes.





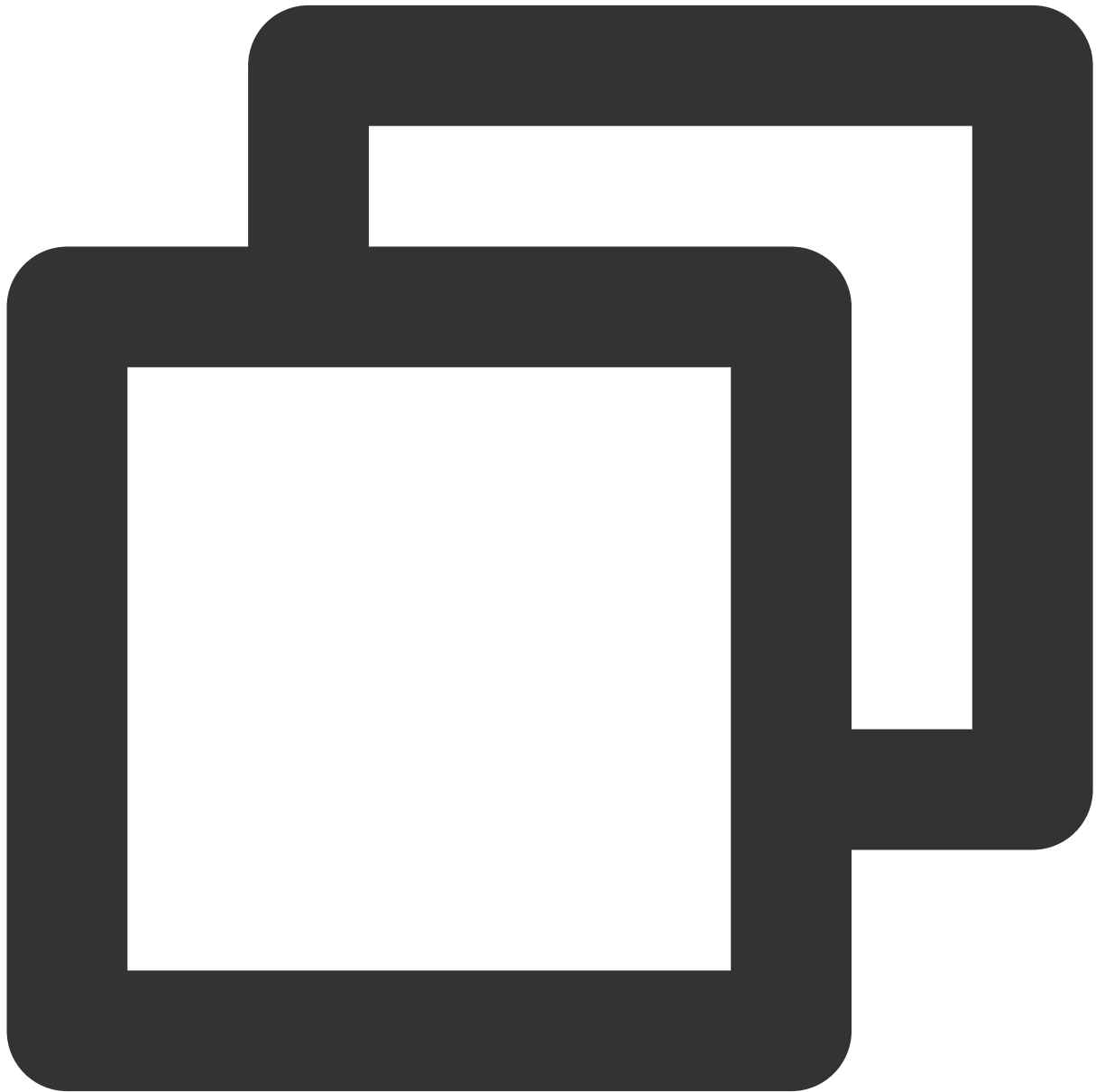
```
void onSeatListChange(List<SeatInfo> seatInfoList);
```

The parameters are described below:

| Parameter    | Type           | Description         |
|--------------|----------------|---------------------|
| seatInfoList | List<SeatInfo> | The full seat list. |

### onAnchorEnterSeat

Someone became a speaker or was made a speaker by the owner.



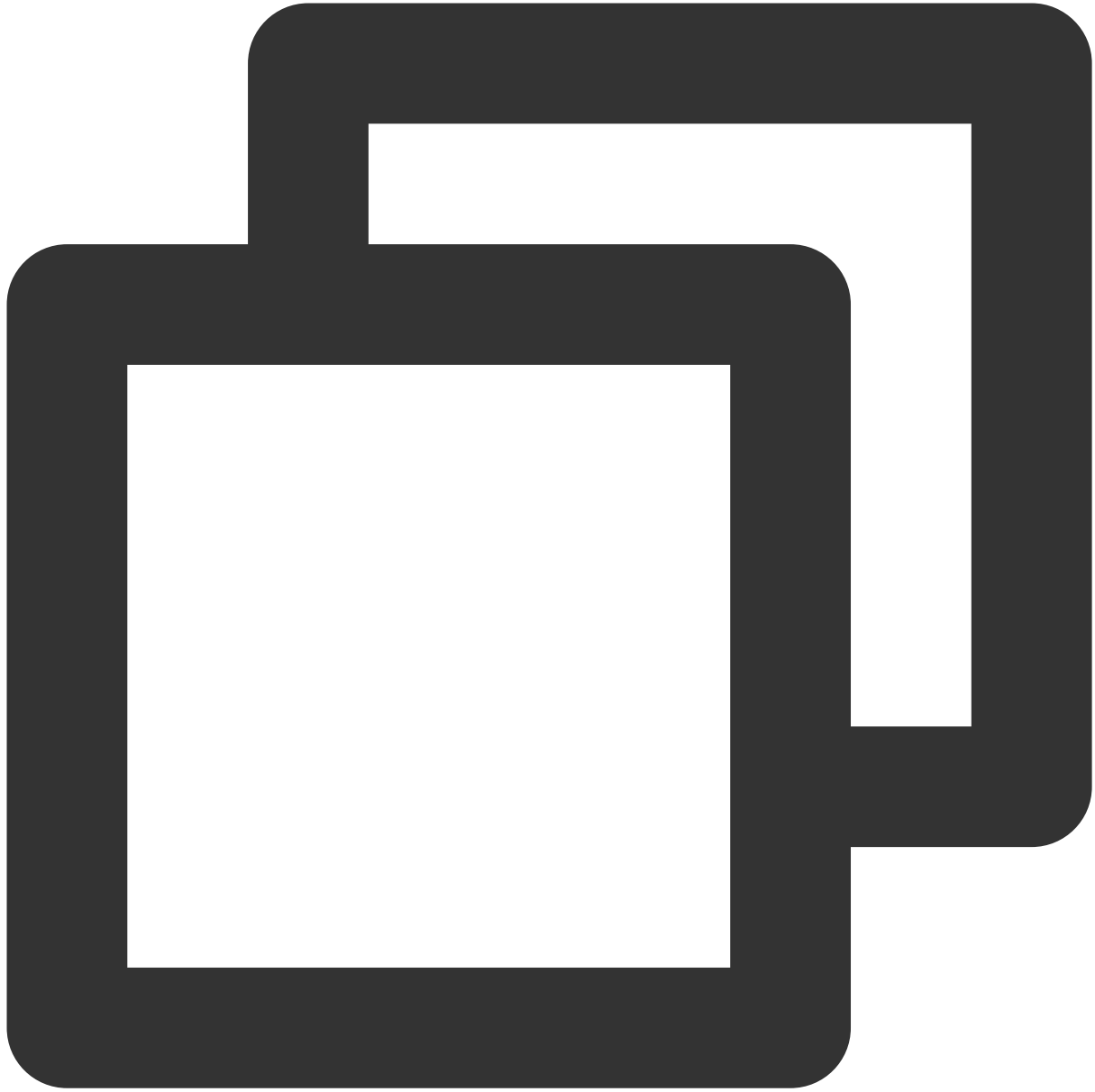
```
void onAnchorEnterSeat(int index, TRTCKaraokeRoomDef.UserInfo user);
```

The parameters are described below:

| Parameter | Type     | Description                                |
|-----------|----------|--------------------------------------------|
| index     | int      | The seat taken.                            |
| user      | UserInfo | The details of the user who took the seat. |

## onAnchorLeaveSeat

A speaker became a listener or was made a listener by the room owner.



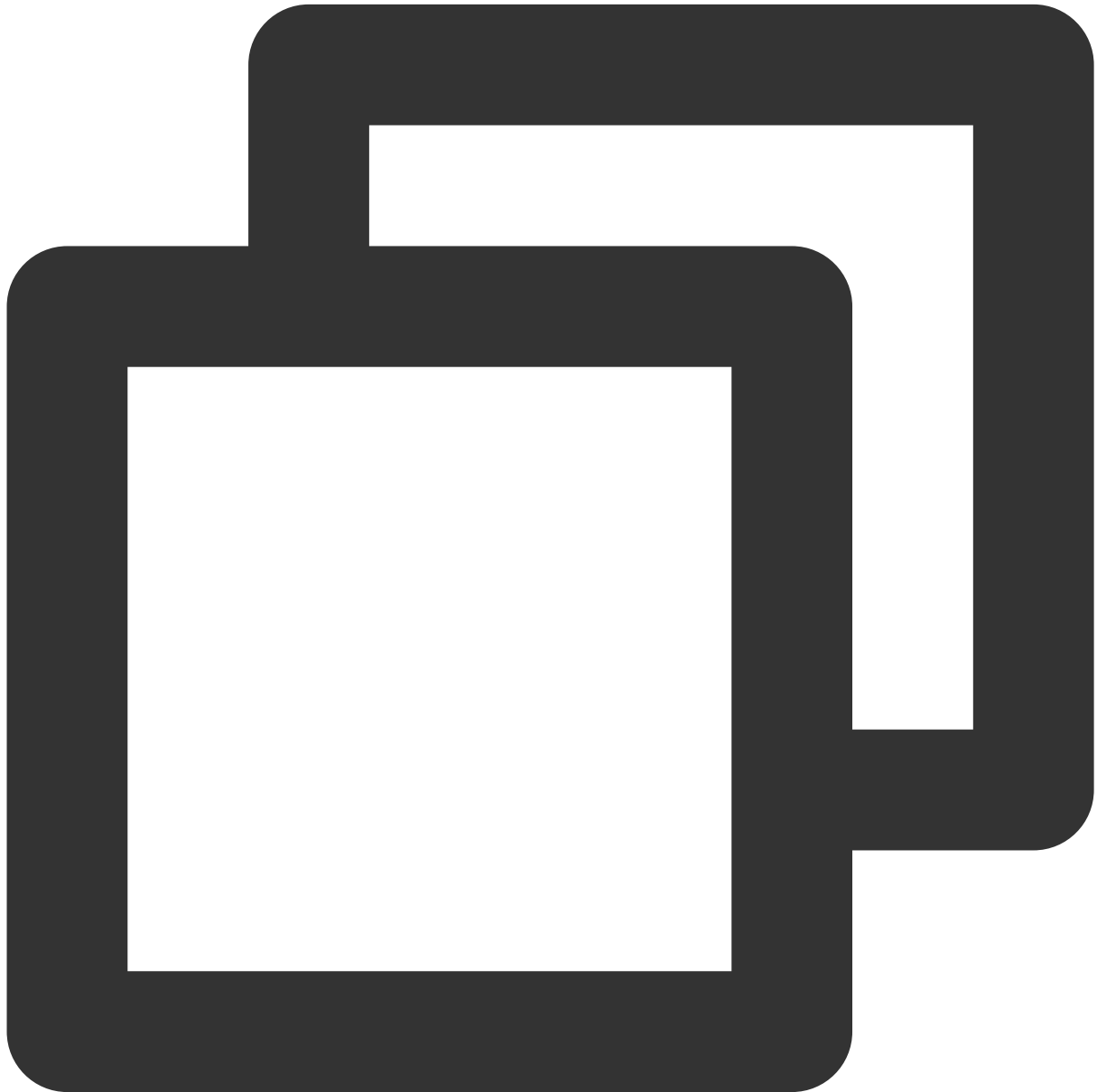
```
void onAnchorLeaveSeat(int index, TRTCKaraokeRoomDef.UserInfo user);
```

The parameters are described below:

| Parameter | Type     | Description                                    |
|-----------|----------|------------------------------------------------|
| index     | int      | The seat previously occupied by the speaker.   |
| user      | UserInfo | The details of the user who became a listener. |

## onSeatMute

The room owner muted/unmuted a seat.



```
void onSeatMute(int index, boolean isMute);
```

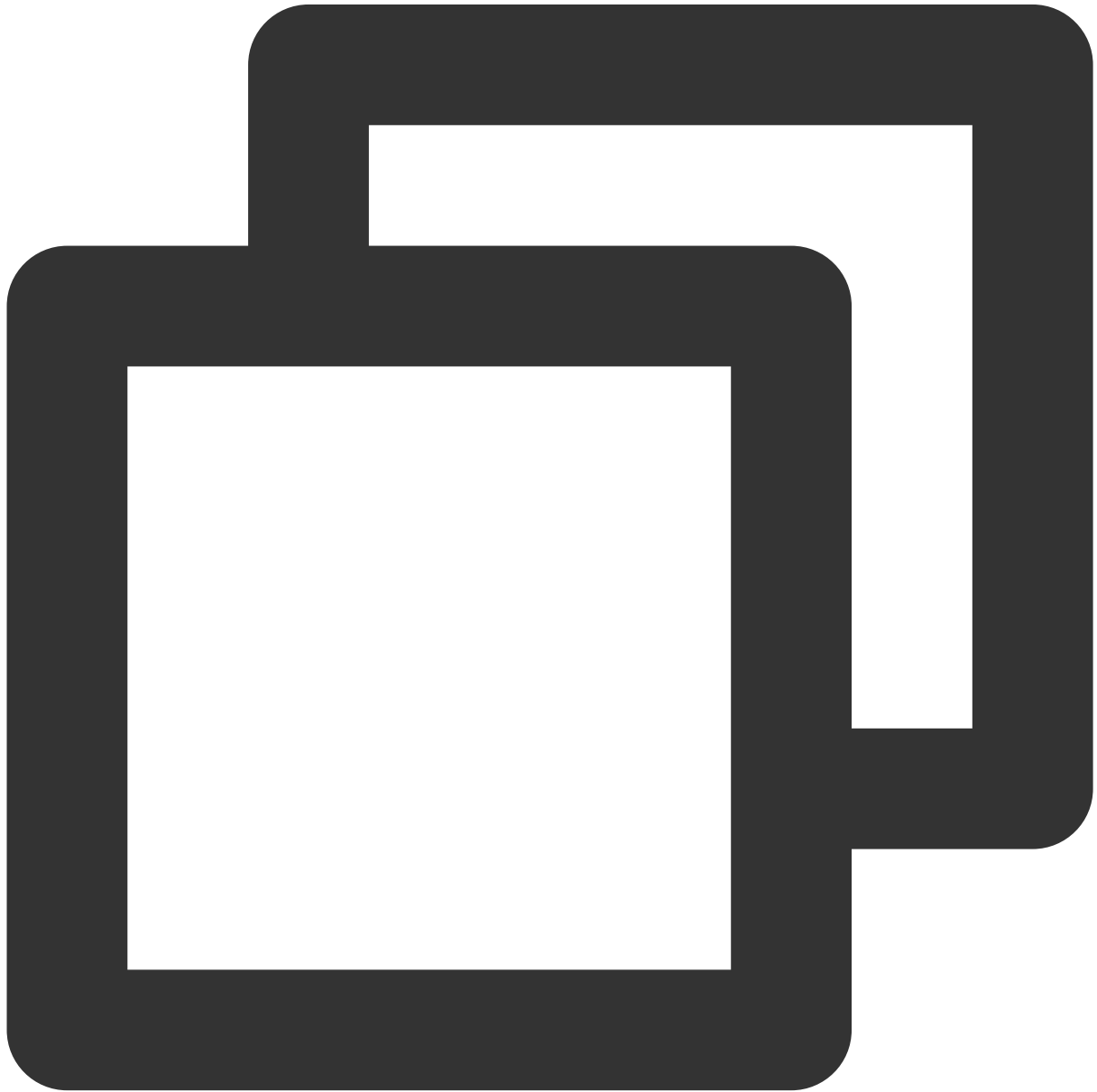
The parameters are described below:

| Parameter | Type    | Description             |
|-----------|---------|-------------------------|
| index     | int     | The seat muted/unmuted. |
| isMute    | boolean |                         |

|  |  |                                                         |
|--|--|---------------------------------------------------------|
|  |  | <code>true</code> : Muted; <code>false</code> : Unmuted |
|--|--|---------------------------------------------------------|

## onSeatClose

The room owner blocked/unblocked a seat.



```
void onSeatClose(int index, boolean isClose);
```

The parameters are described below:

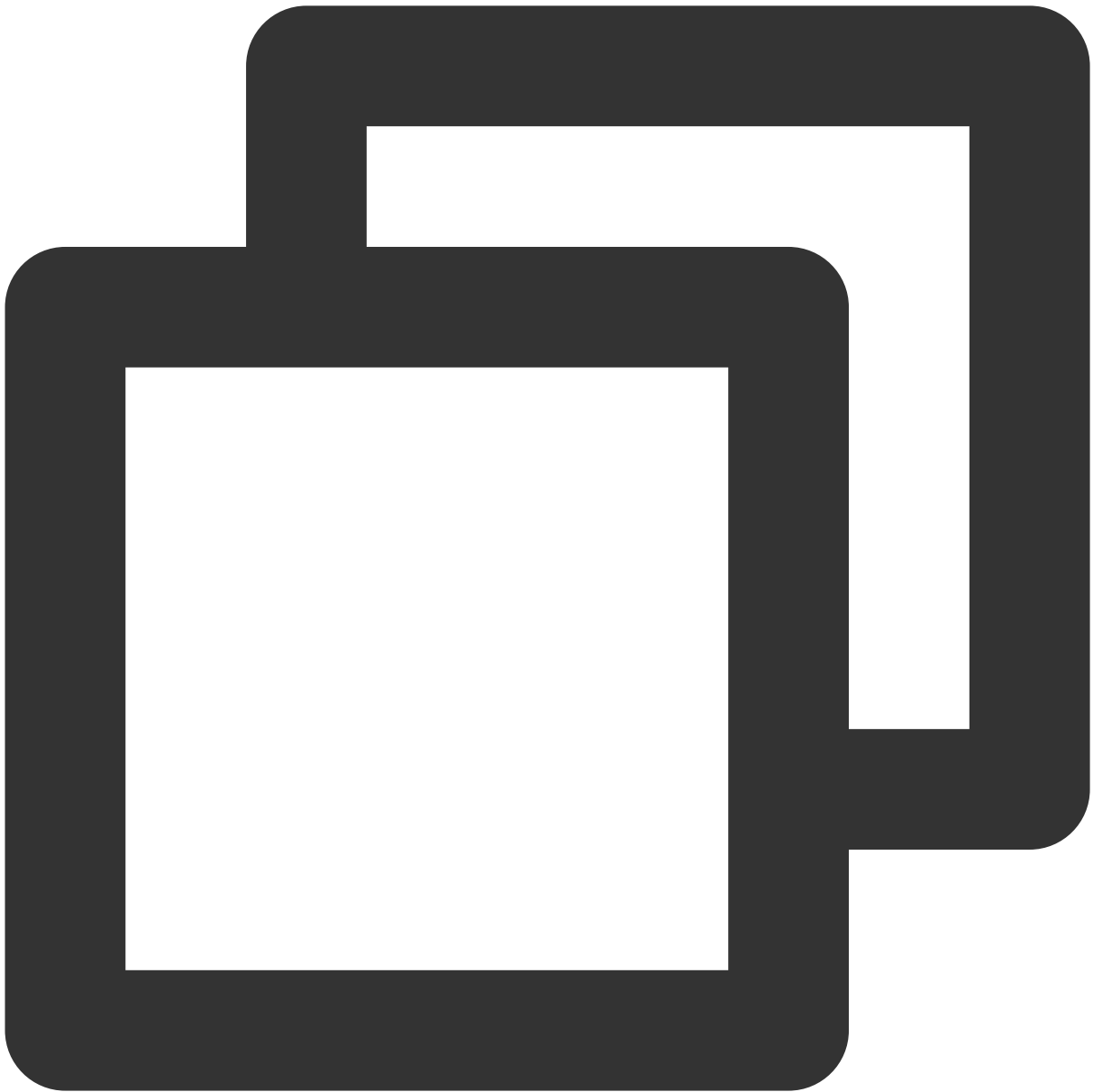
| Parameter | Type | Description |
|-----------|------|-------------|
|           |      |             |

|         |         |                                                             |
|---------|---------|-------------------------------------------------------------|
| index   | int     | The seat blocked/unblocked.                                 |
| isClose | boolean | <code>true</code> : Blocked; <code>false</code> : Unblocked |

## Callback APIs for Room Entry/Exit by Listener

### onAudienceEnter

A listener entered the room.



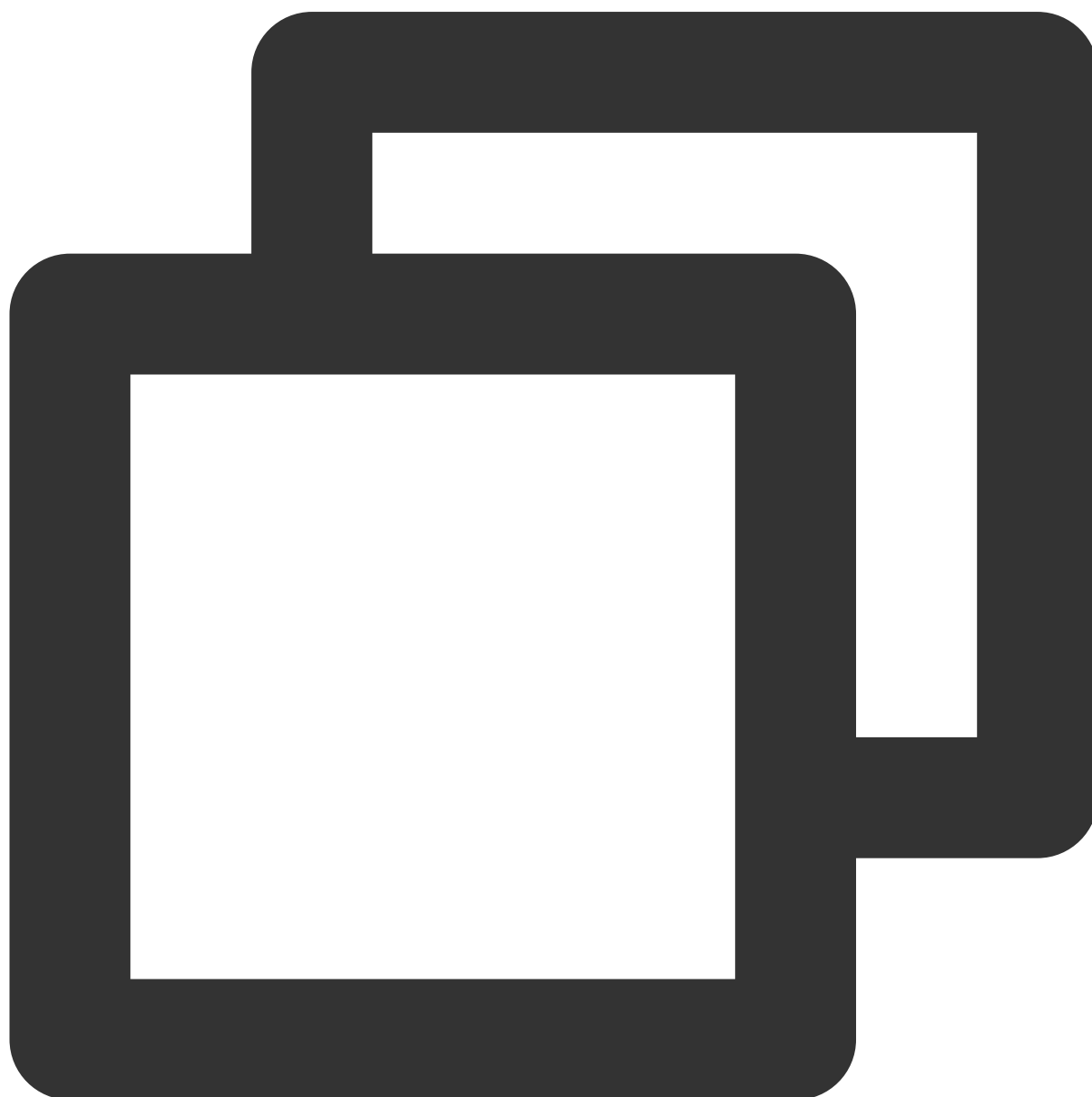
```
void onAudienceEnter (TRTCKaraokeRoomDef.UserInfo userInfo);
```

The parameters are described below:

| Parameter | Type     | Description                                           |
|-----------|----------|-------------------------------------------------------|
| userInfo  | UserInfo | The information of the listener who entered the room. |

## onAudienceExit

A listener exited the room.



```
void onAudienceExit (TRTCKaraokeRoomDef.UserInfo userInfo);
```

The parameters are described below:

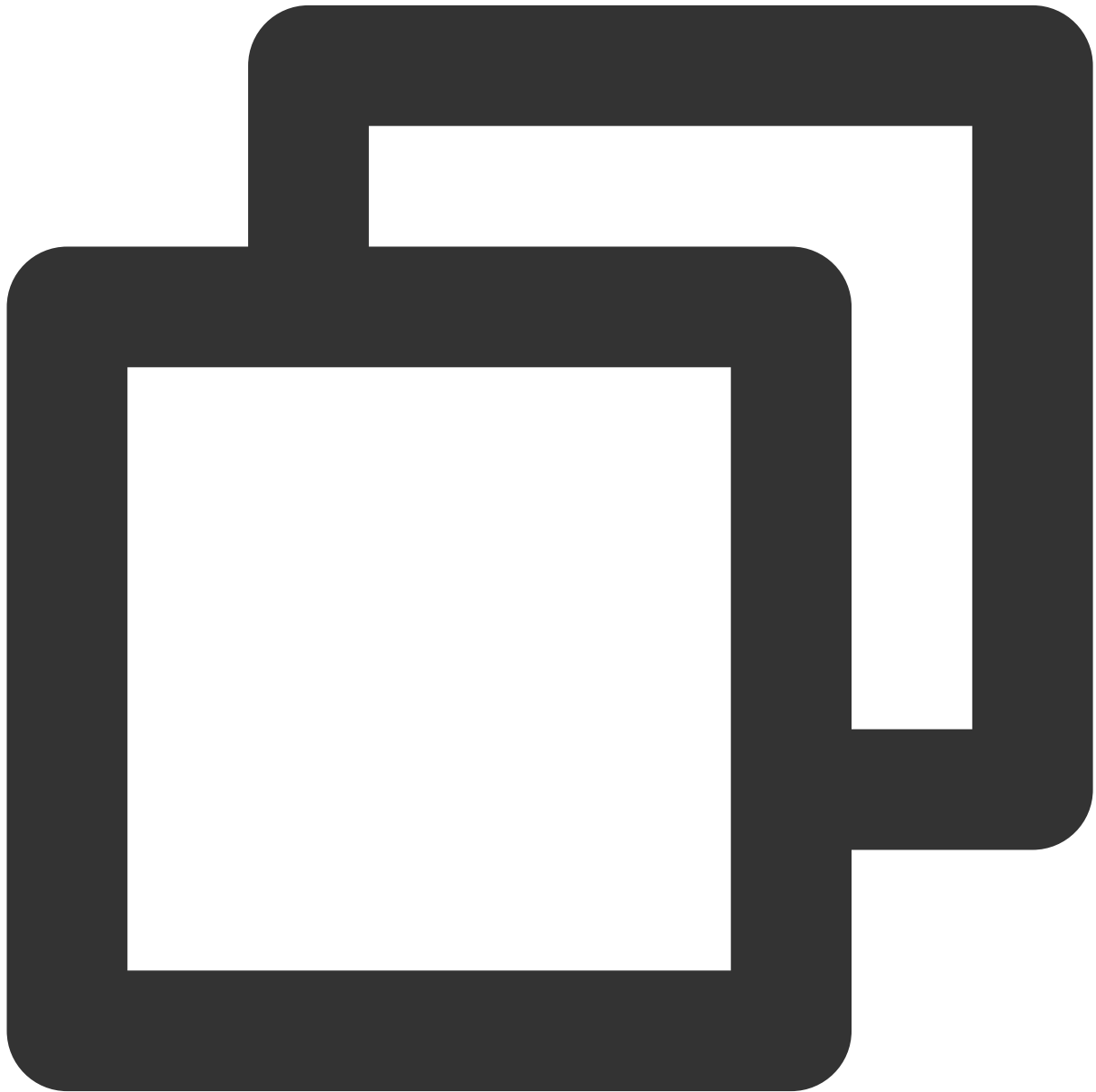
| Parameter | Type     | Description                                          |
|-----------|----------|------------------------------------------------------|
| userInfo  | UserInfo | The information of the listener who exited the room. |

## Message Event Callback APIs

### **onRecvRoomTextMsg**

Callback for receiving a text chat message.





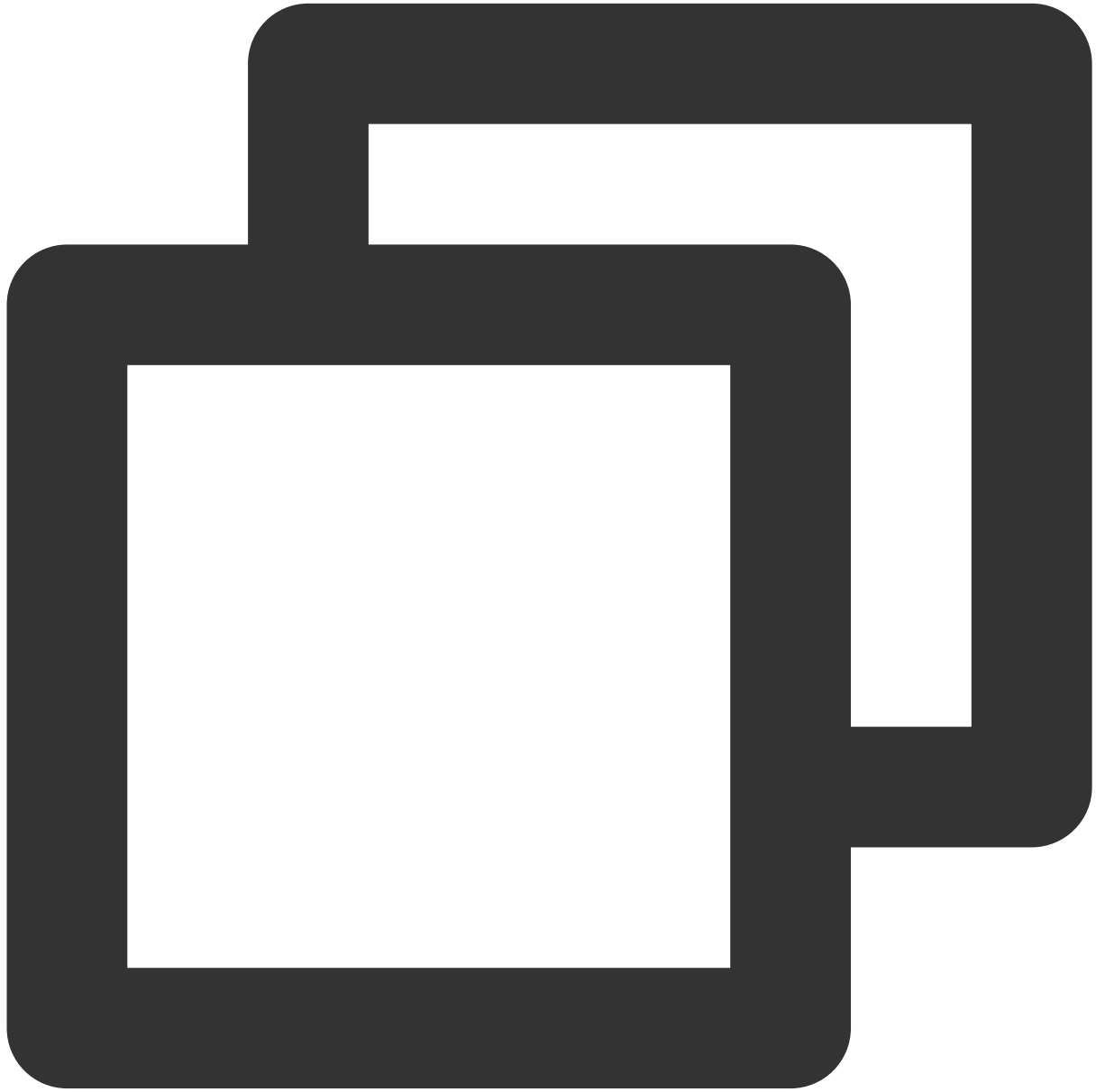
```
void onRecvRoomTextMsg(String message, TRTCKaraokeRoomDef.UserInfo userInfo);
```

The parameters are described below:

| Parameter | Type     | Description                |
|-----------|----------|----------------------------|
| message   | String   | A text chat message.       |
| userInfo  | UserInfo | Information of the sender. |

### **onRecvRoomCustomMsg**

A custom message was received.



```
void onRecvRoomCustomMsg(String cmd, String message, TRTCKaraokeRoomDef.UserInfo us
```

The parameters are described below:

| Parameter | Type   | Description                                                                |
|-----------|--------|----------------------------------------------------------------------------|
| command   | String | A custom command word used to distinguish between different message types. |
| message   | String | A text chat message.                                                       |
|           |        |                                                                            |

userInfo

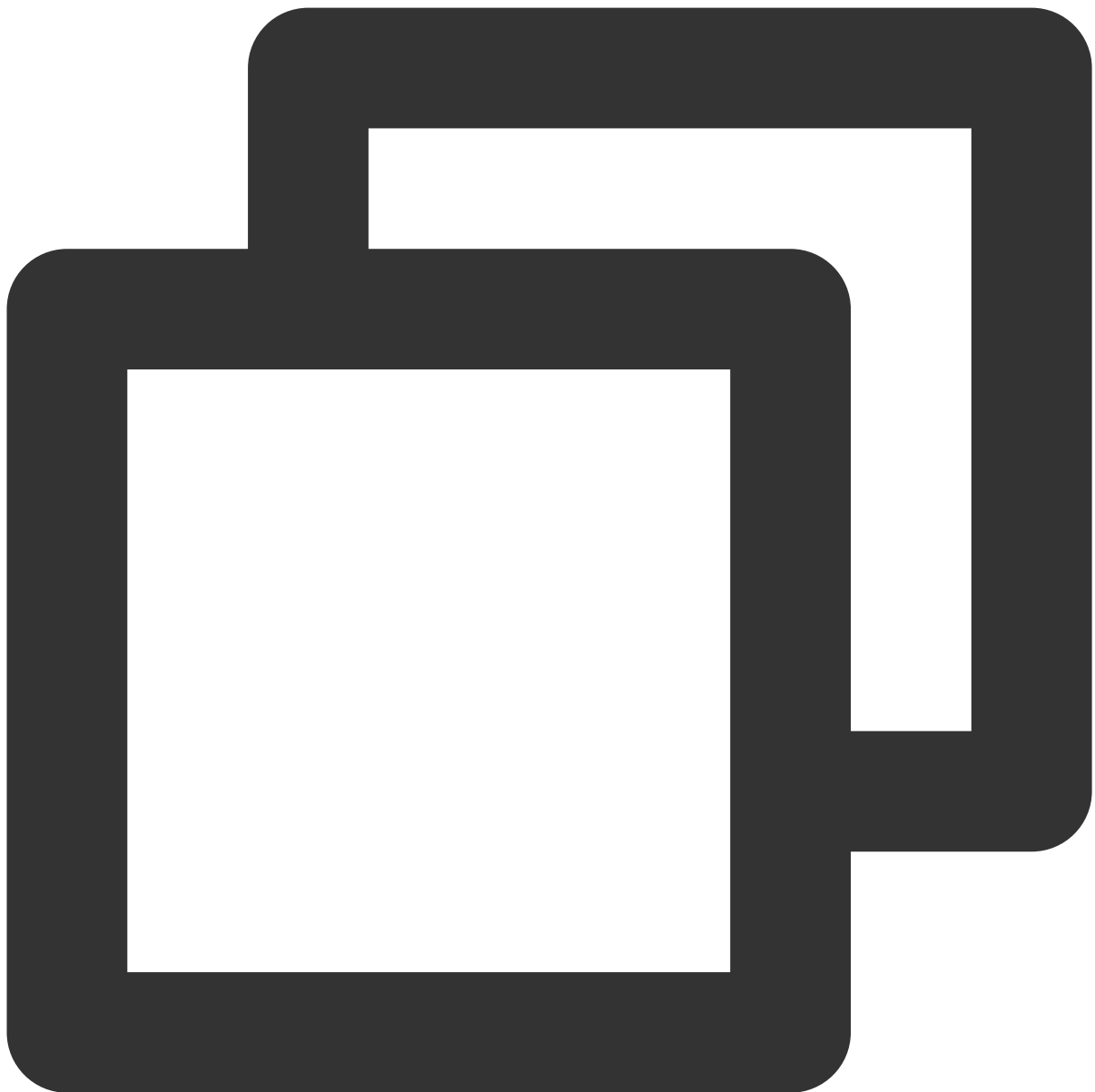
UserInfo

Information of the sender.

## Invitation Signaling Callback APIs

### **onReceiveNewInvitation**

An invitation was received.



```
void onReceiveNewInvitation(String id, String inviter, String cmd, String content);
```

The parameters are described below:

| Parameter | Type   | Description                                  |
|-----------|--------|----------------------------------------------|
| id        | String | The invitation ID.                           |
| inviter   | String | The user ID of the inviter.                  |
| cmd       | String | A custom command word specified by business. |
| content   | String | Content specified by business                |

### **onInviteeAccepted**

The invitee accepted the invitation.



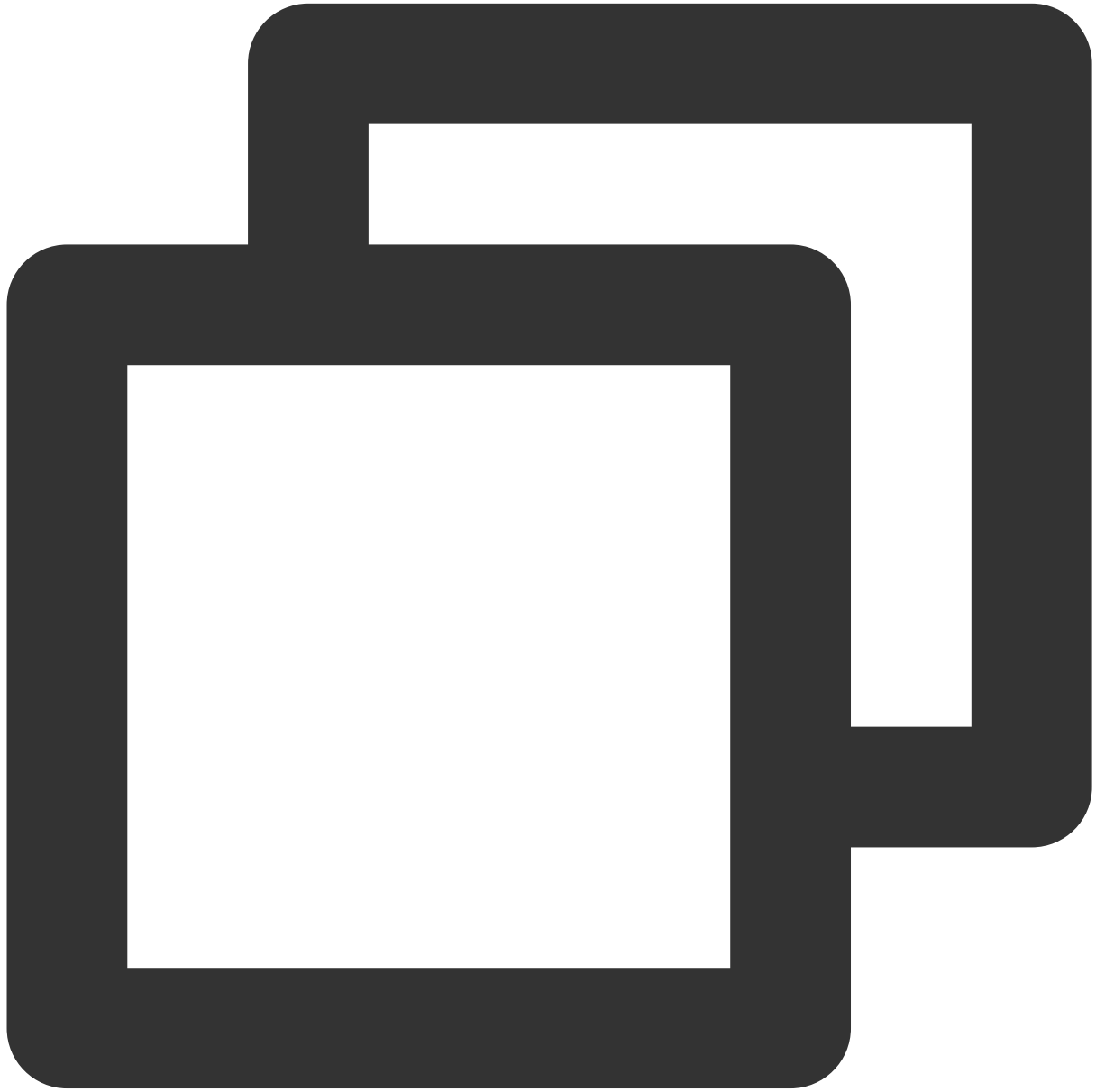
```
void onInviteeAccepted(String id, String invitee);
```

The parameters are described below:

| Parameter | Type   | Description                 |
|-----------|--------|-----------------------------|
| id        | String | The invitation ID.          |
| invitee   | String | The user ID of the invitee. |

### **onInviteeRejected**

The invitee declined the invitation.



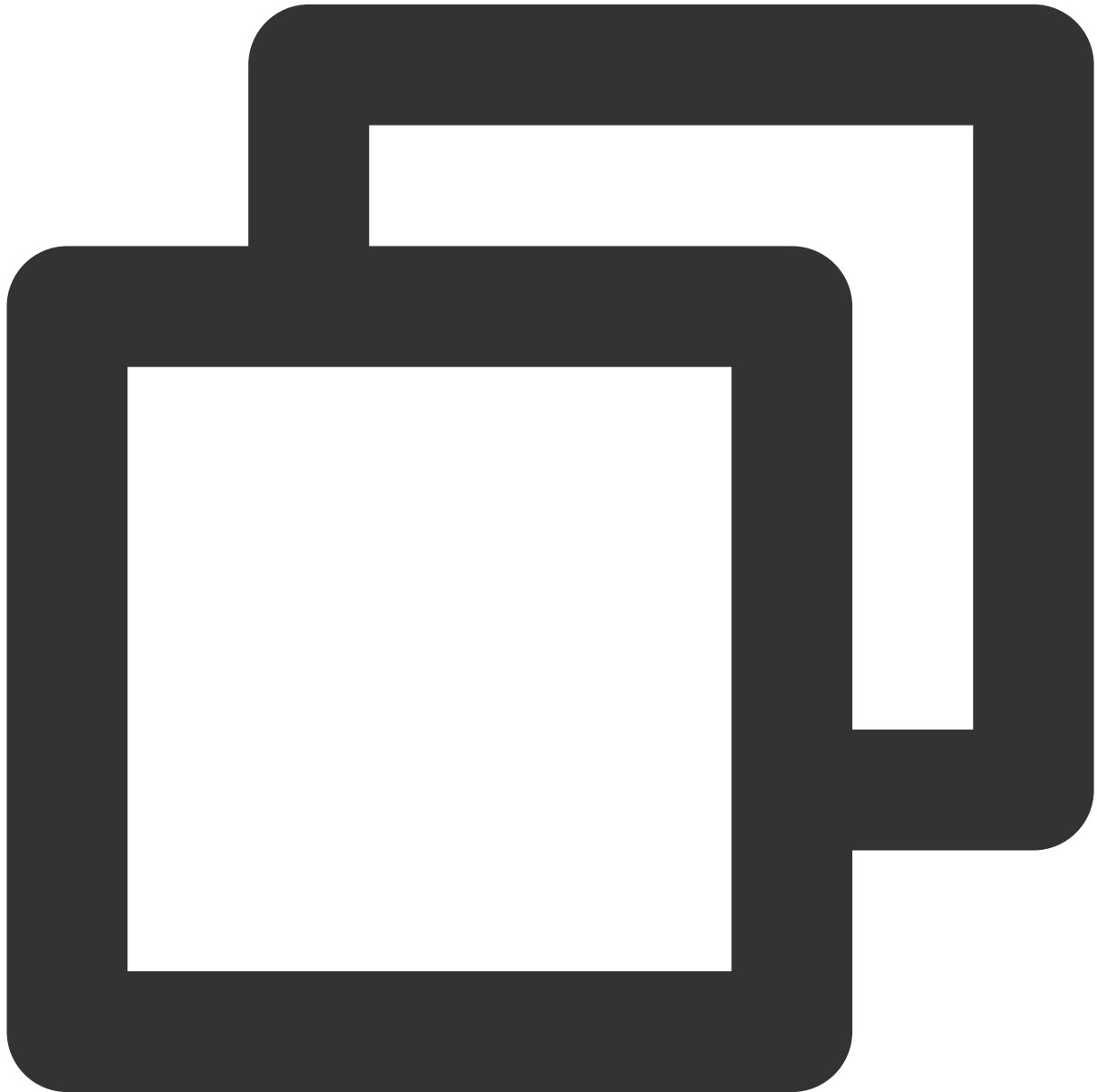
```
void onInviteeRejected(String id, String invitee);
```

The parameters are described below:

| Parameter | Type   | Description                 |
|-----------|--------|-----------------------------|
| id        | String | The invitation ID.          |
| invitee   | String | The user ID of the invitee. |

## onInvitationCancelled

The inviter canceled the invitation.



```
void onInvitationCancelled(String id, String inviter);
```

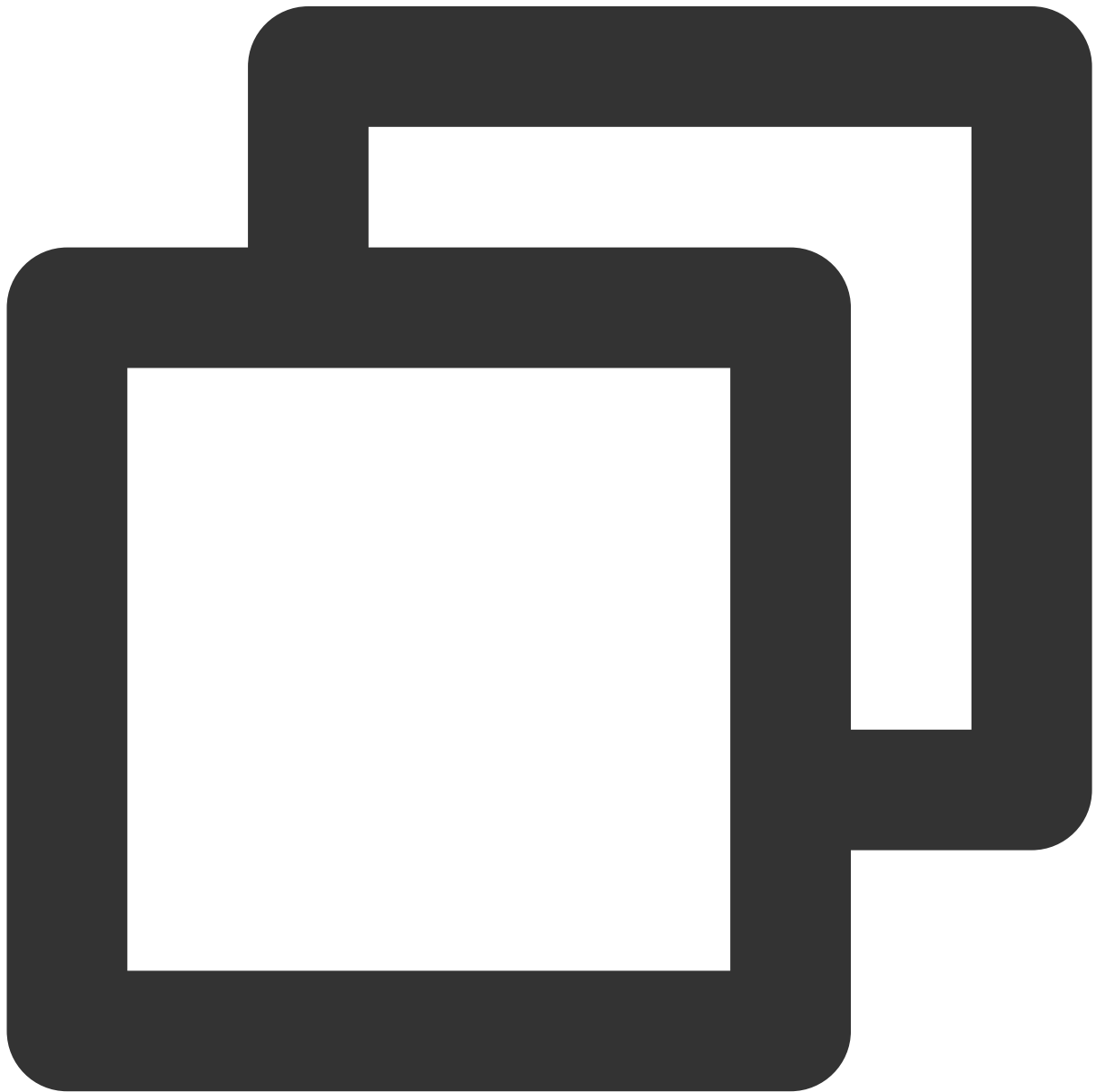
The parameters are described below:

| Parameter | Type   | Description                 |
|-----------|--------|-----------------------------|
| id        | String | The invitation ID.          |
| inviter   | String | The user ID of the inviter. |

## Music Playback Status Callback APIs

### **onMusicPrepareToPlay**

Music playback is ready.



```
void onMusicPrepareToPlay(int musicID);
```

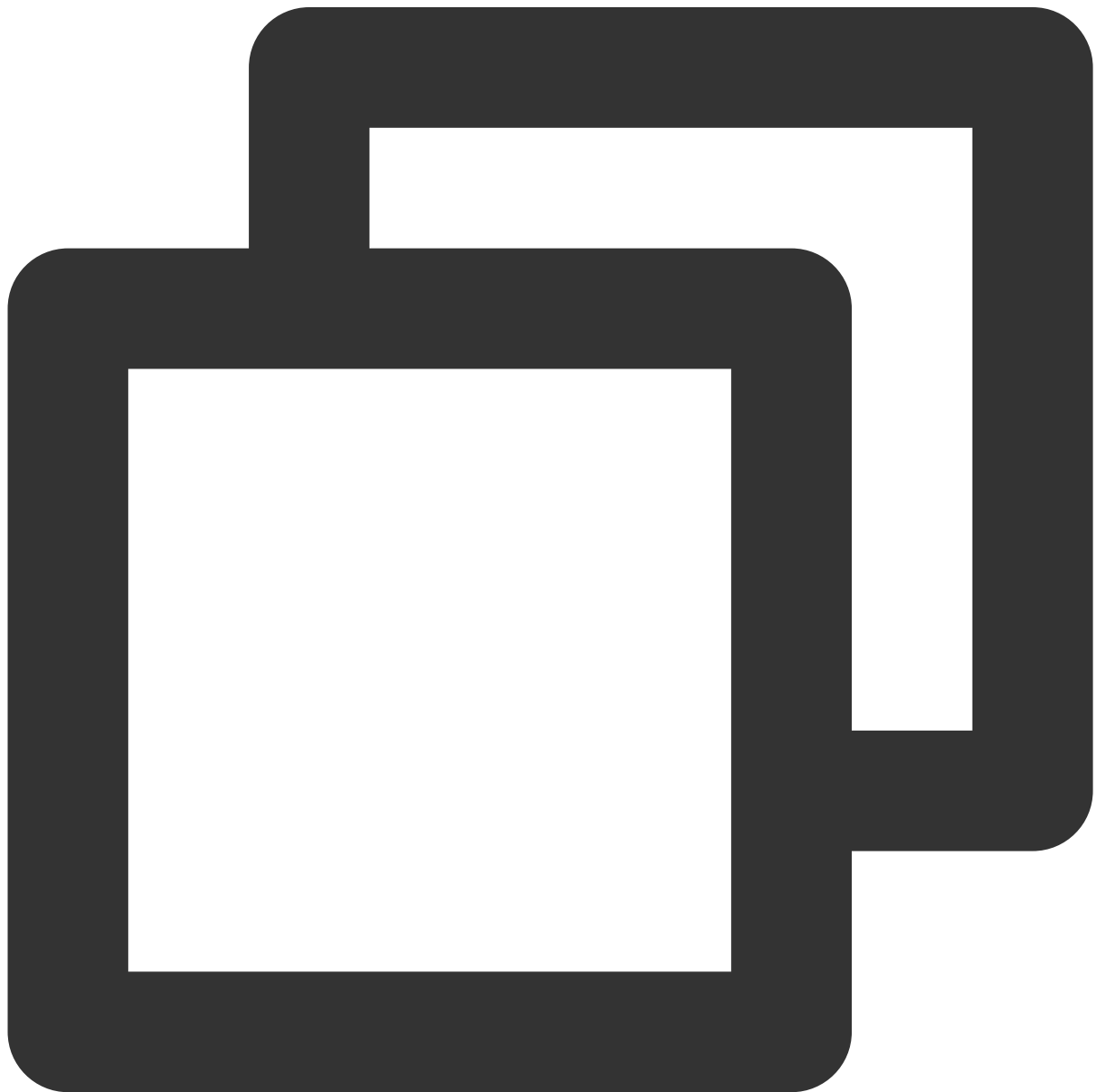
The parameters are described below:



| Parameter | Type | Description                                      |
|-----------|------|--------------------------------------------------|
| musicID   | int  | The <code>musicID</code> passed in for playback. |

## onMusicProgressUpdate

Music playback progress.



```
void onMusicProgressUpdate(int musicID, long progress, long total);
```

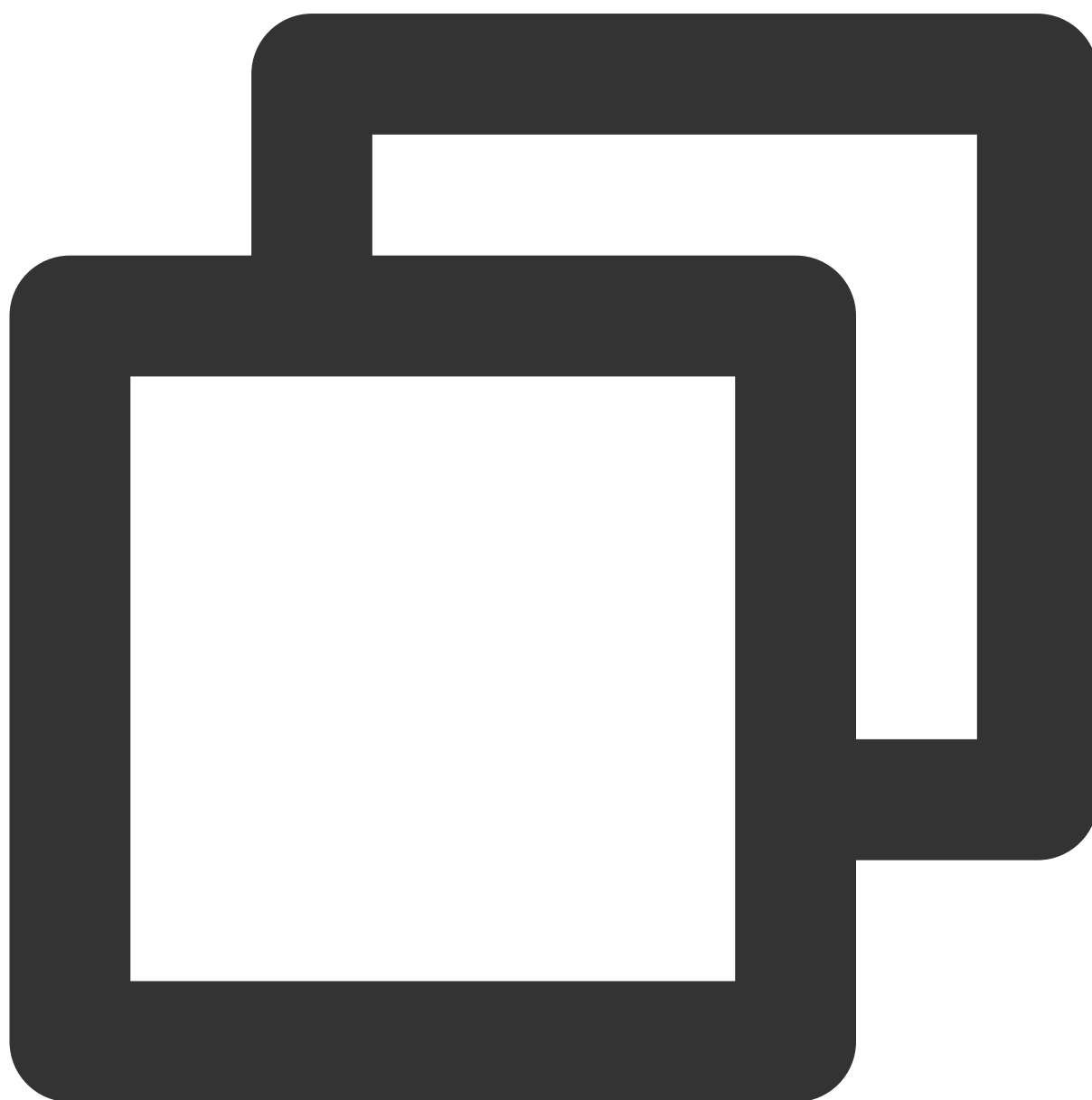
The parameters are described below:

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

| Parameter | Type | Description                                      |
|-----------|------|--------------------------------------------------|
| musicID   | int  | The <code>musicID</code> passed in for playback. |
| progress  | long | The current playback progress in ms.             |
| total     | long | The total duration in ms.                        |

### **onMusicCompletePlaying**

Music playback was completed.



```
void onMusicCompletePlaying(int musicID);
```

The parameters are described below:

| Parameter | Type | Description                                      |
|-----------|------|--------------------------------------------------|
| musicID   | int  | The <code>musicID</code> passed in for playback. |

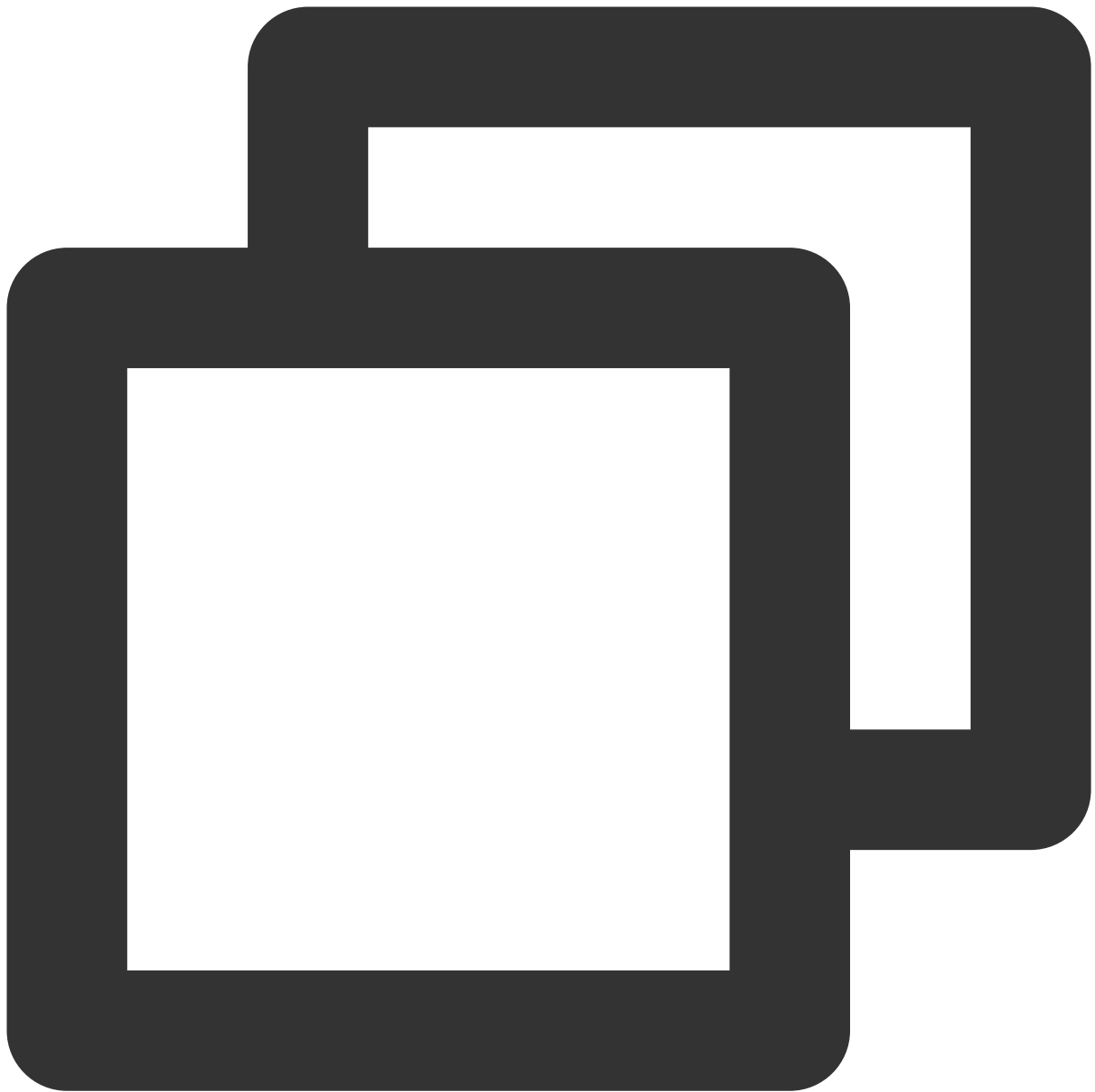
# FAQs

## iOS

Last updated : 2023-09-26 17:01:07

### Ear Monitor-related Issues

**In Karaoke scenarios, ear monitoring is likely to be used. How can I enable the ear monitoring function?**



```
[[trtcCloud getAudioEffectManager] enableVoiceEarMonitor:YES];
```

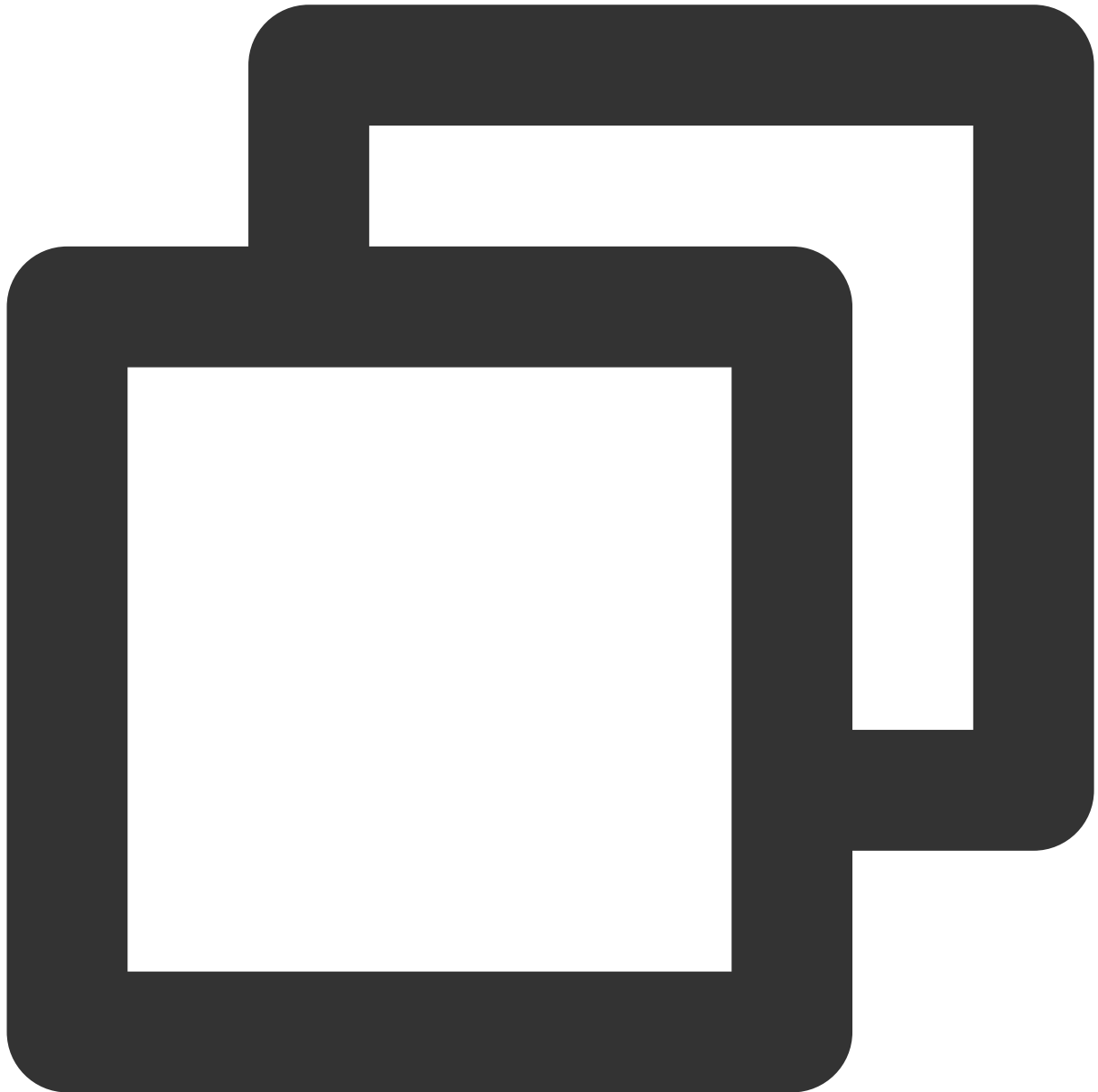
### What if there is no effect after enabling the ear monitoring function?

Due to the high hardware latency of Bluetooth headsets, please try to prompt the host to wear wired headphones on the user interface. At the same time, please note that not all phones can achieve excellent ear monitoring effects after enabling this feature. The TRTC SDK has already blocked this effect on some phones with poor ear monitoring performance.

## Is the ear monitoring latency too high?

Please check if you are using a Bluetooth headset. Due to the high hardware latency of Bluetooth headsets, please try to use wired headphones as much as possible.

In addition, you can try to improve the high latency of ear monitoring by enabling hardware ear monitoring through the experimental interface `setSystemAudioKitEnabled`. Currently, for Huawei and Vivo devices, the SDK uses hardware ear monitoring by default, while other devices use software ear monitoring by default.



```
// Enable hardware ear monitoring
NSDictionary *jsonDic = @{
    @"api": @"setSystemAudioKitEnabled",
```

```
        @"params": @{@"enable": @(1)}  
    };  
    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr  
    NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin  
    [trtcCloud callExperimentalAPI:jsonString];
```

## NTP Time Synchronization Issues

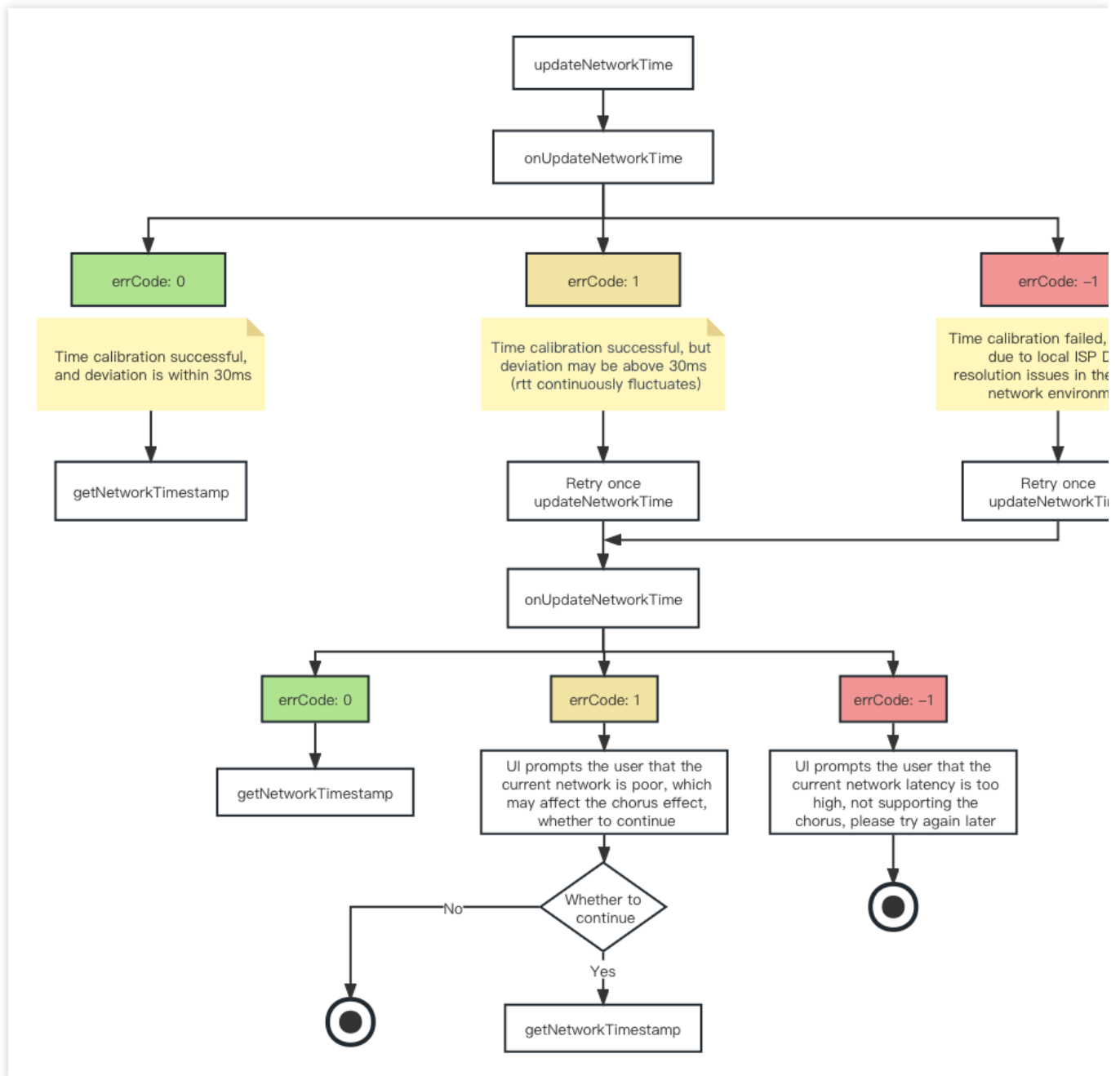
### Reminder: “NTP time sync finished, but result maybe inaccurate” ?

NTP time synchronization is successful, but the deviation may be more than 30ms, reflecting poor client network environment and continuous rtt jitter.

### Reminder: “Error in AddressResolver: No address associated with hostname” ?

NTP time synchronization failure may be due to temporary anomalies in the local carrier DNS resolution under the current network environment. Please try again later.

### NTP service retry processing logic?

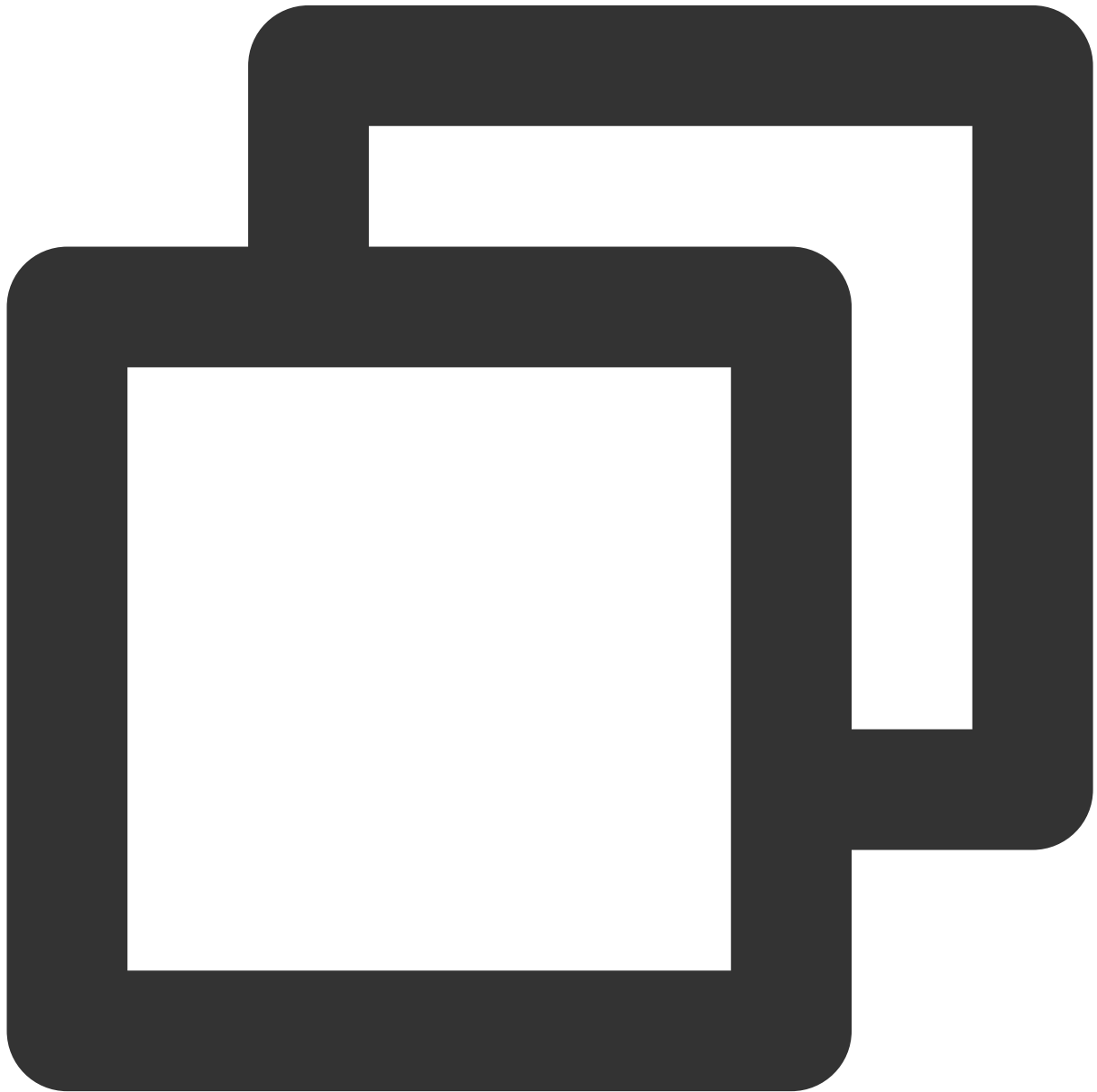


## Network Speed Test Recommendations

Online Karaoke scenarios have high network requirements for users, especially real-time chorus. A high-quality and stable network environment is necessary for a good Karaoke experience. Therefore, it is recommended to perform a network speed test on the user before entering the room, and give a UI layer reminder to users who do not meet the network requirements, prohibiting them from joining the Karaoke room or participating in chorus.

Initiating network speed test with TRTC SDK:





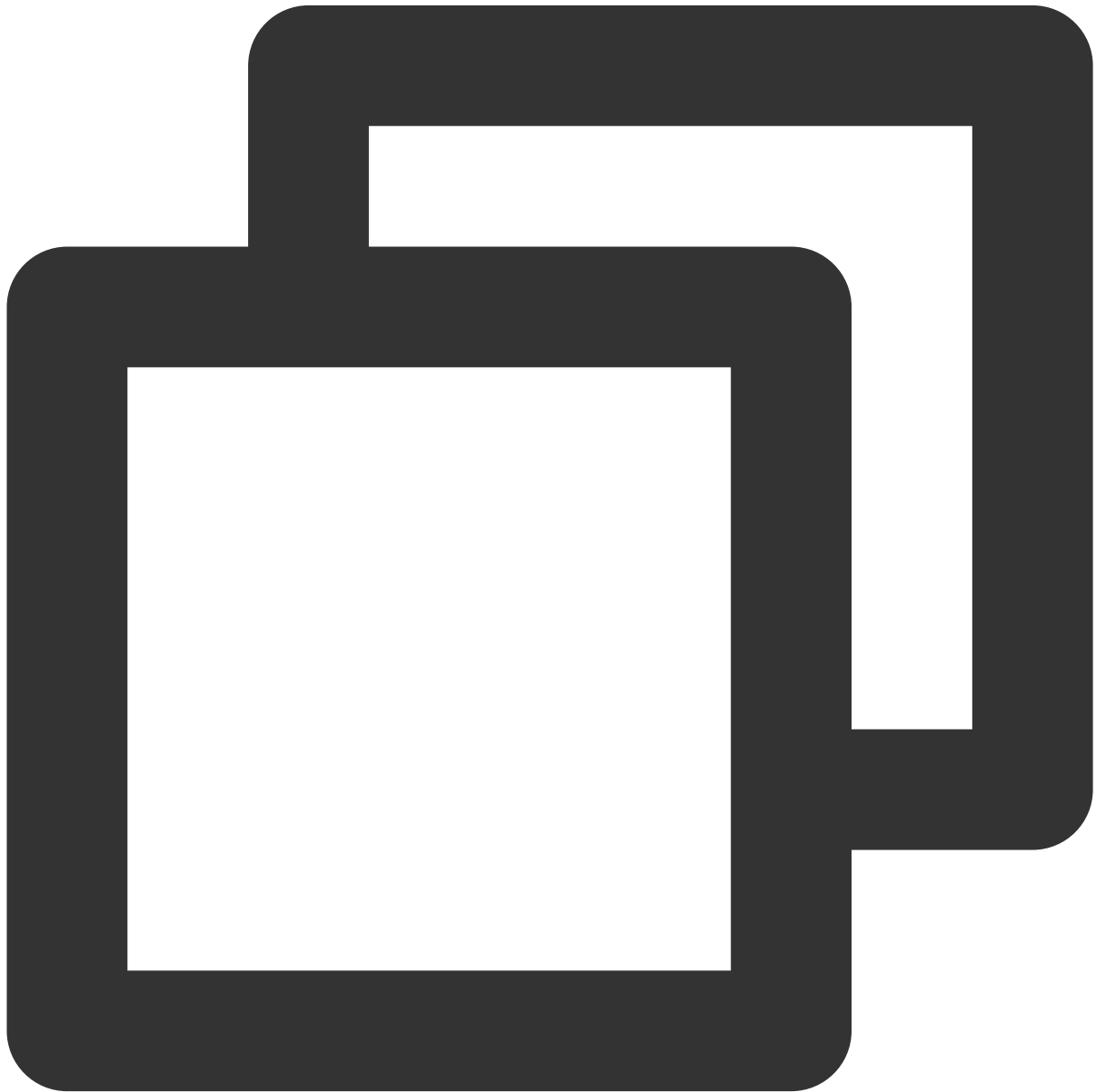
```
TRTCSpeedTestParams *speedTestParams = [[TRTCSpeedTestParams alloc] init];
speedTestParams.sdkAppId = SDK_APP_ID;
speedTestParams.userId = userId;
speedTestParams.userSig = userSig;
// If the actual bandwidth is higher than the expected value, the test result is th
// if the actual bandwidth is lower than the expected value, the test result is the
speedTestParams.expectedDownBandwidth = 3000; // Expected downstream bandwidth, ran
speedTestParams.expectedUpBandwidth = 3000; // Expected upstream bandwidth, ranging
[trtcCloud startSpeedTest:speedTestParams];
```

**Note :**

Expected upstream bandwidth, ranging from 10 to 5000 kbps

Please perform the network speed test before entering the room. Network speed testing in the room will affect the normal audio and video transmission effects, and due to too much interference, the network speed test results will also be inaccurate.

TRTC SDK network speed test result callback:



```
- (void)onSpeedTestResult:(TRTCSpeedTestResult *)result {  
    NSString *tquality = @"Unknown";  
    switch (result.quality) {  
        case TRTCQuality_Unknown:
```

```

        tquality = @"Unknown";
        break;
    case TRTCQuality_Excellent:
        tquality = @"The current network is excellent";
        break;
    case TRTCQuality_Good:
        tquality = @"The current network is good";
        break;
    case TRTCQuality_Poor:
        tquality = @"The current network is poor";
        break;
    case TRTCQuality_Bad:
        tquality = @"The current network is bad";
        break;
    case TRTCQuality_Vbad:
        tquality = @"The current network is very bad";
        break;
    case TRTCQuality_Down:
        tquality = @"The current network does not meet TRTC\\`s minimum request";
        break;
    default:
        break;
}

if (result.success) {
    [mTextTestResult addObject:@"test successfull!\\n"];
    [mTextTestResult addObject:[NSString stringWithFormat:@"IP address:%@ \\n",
    [mTextTestResult addObject:[NSString stringWithFormat:@"uplink packet loss
    [mTextTestResult addObject:[NSString stringWithFormat:@"downlink packet los
    [mTextTestResult addObject:[NSString stringWithFormat:@"network lantency:%i
    [mTextTestResult addObject:[NSString stringWithFormat:@"downlink bandwidth:
    [mTextTestResult addObject:[NSString stringWithFormat:@"uplink bandwidth:%i
    [mTextTestResult addObject:[NSString stringWithFormat:@"downlink bandwidth:
} else {
    [mTextTestResult addObject:@"test successfull!\\n"];
    [mTextTestResult addObject:[NSString stringWithFormat:@"errMsg:%@ \\n", re:
}
}

```

## Joining a Chorus Midway

The real-time chorus solution theoretically has no limit on the number of chorus participants, supporting multiple people to participate in the chorus simultaneously, as well as joining the chorus midway.

The following are the key actions for joining a chorus midway:

NTP time synchronization

Enable chorus experimental interface

Enter the room and start streaming on the microphone

Receive chorus signaling, obtain accompaniment resources and chorus agreed time

Calculate the difference between the agreed time and the current time, preload and seek accompaniment

Start participating in the chorus and synchronize accompaniment progress and lyrics progress in real-time

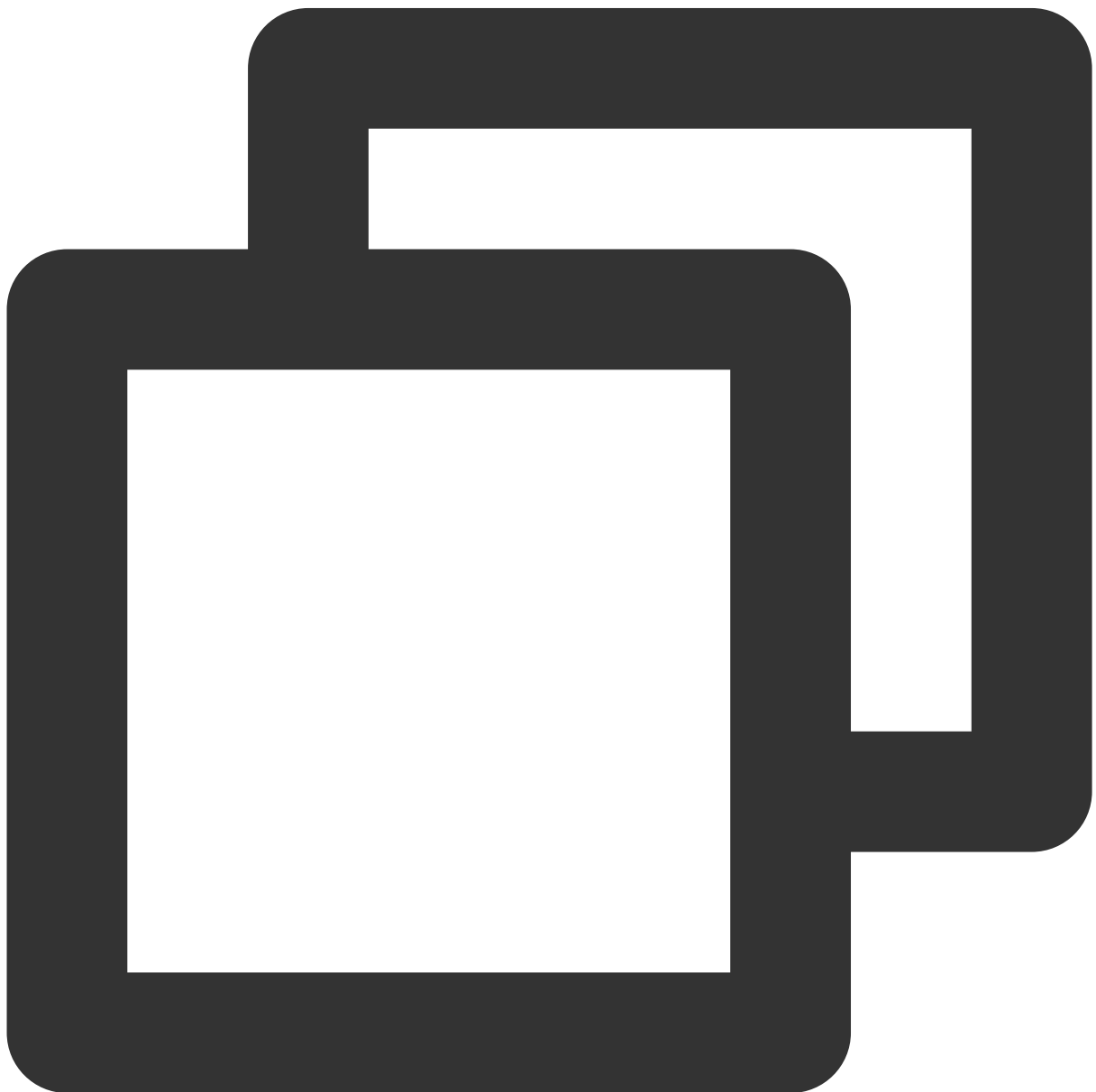
For the specific implementation process and code implementation of the above key actions, please refer to the documentation on [song synchronization](#), [lyrics synchronization](#), [vocal synchronization](#).

# Android

Last updated : 2023-09-27 11:28:11

## Ear Monitor-related Issues

**In Karaoke scenarios, ear monitoring is likely to be used. How can I enable the ear monitoring function?**



```
mTRTCCloud.getAudioEffectManager().enableVoiceEarMonitor(true)
```

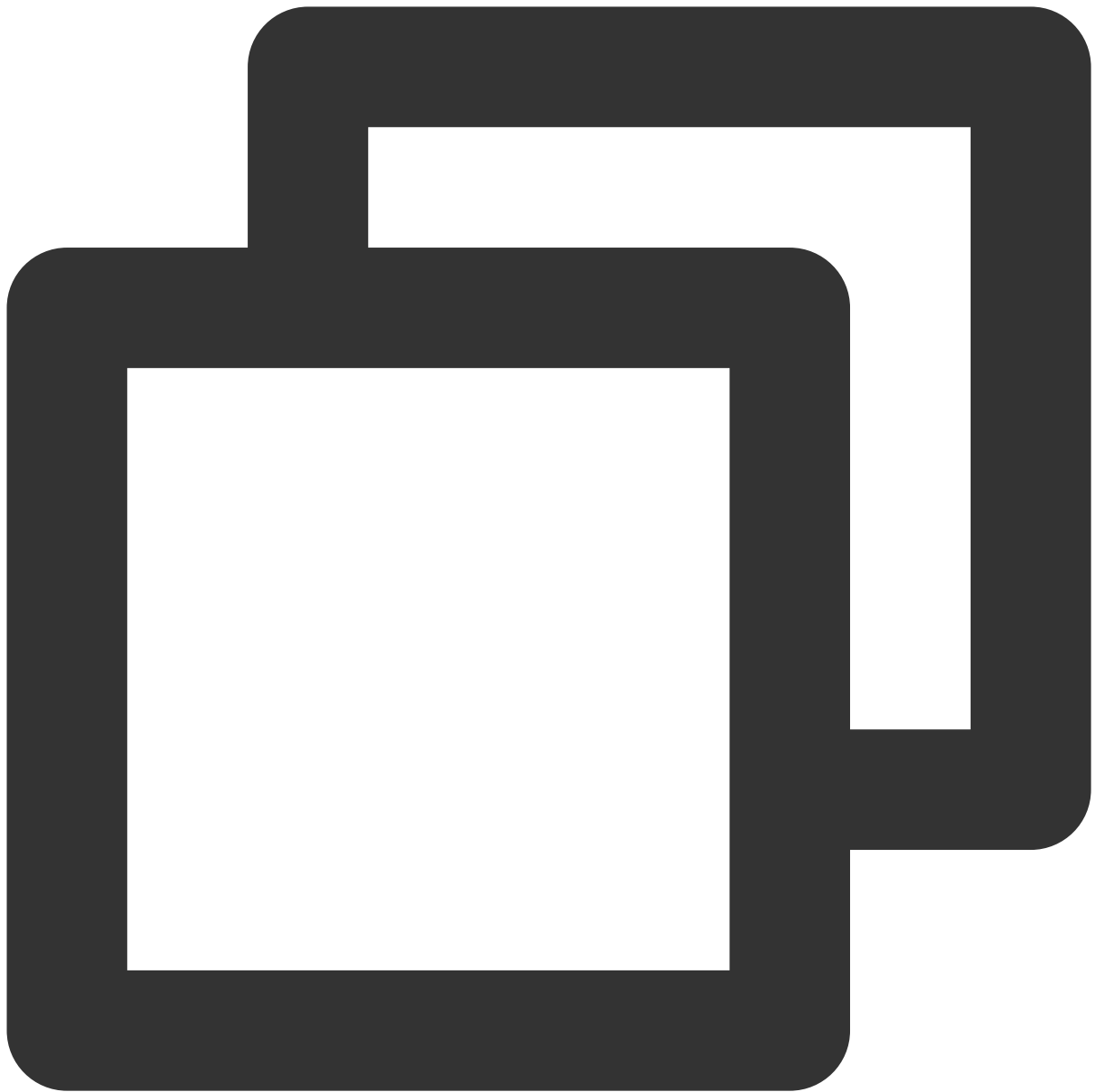
## What if there is no effect after enabling the ear monitoring function?

Due to the high hardware latency of Bluetooth headsets, please try to prompt the host to wear wired headphones on the user interface. At the same time, please note that not all phones can achieve excellent ear monitoring effects after enabling this feature. The TRTC SDK has already blocked this effect on some phones with poor ear monitoring performance.

## Is the ear monitoring latency too high?

Please check if you are using a Bluetooth headset. Due to the high hardware latency of Bluetooth headsets, please try to use wired headphones as much as possible.

In addition, you can try to improve the high latency of ear monitoring by enabling hardware ear monitoring through the experimental interface `setSystemAudioKitEnabled`. Currently, for Huawei and Vivo devices, the SDK uses hardware ear monitoring by default, while other devices use software ear monitoring by default.



```
// Enable hardware ear monitoring  
mTRTCCloud.callExperimentalAPI("{\\"api\\":\\"setSystemAudioKitEnabled\\", \\"param
```

## NTP Time Synchronization Issues

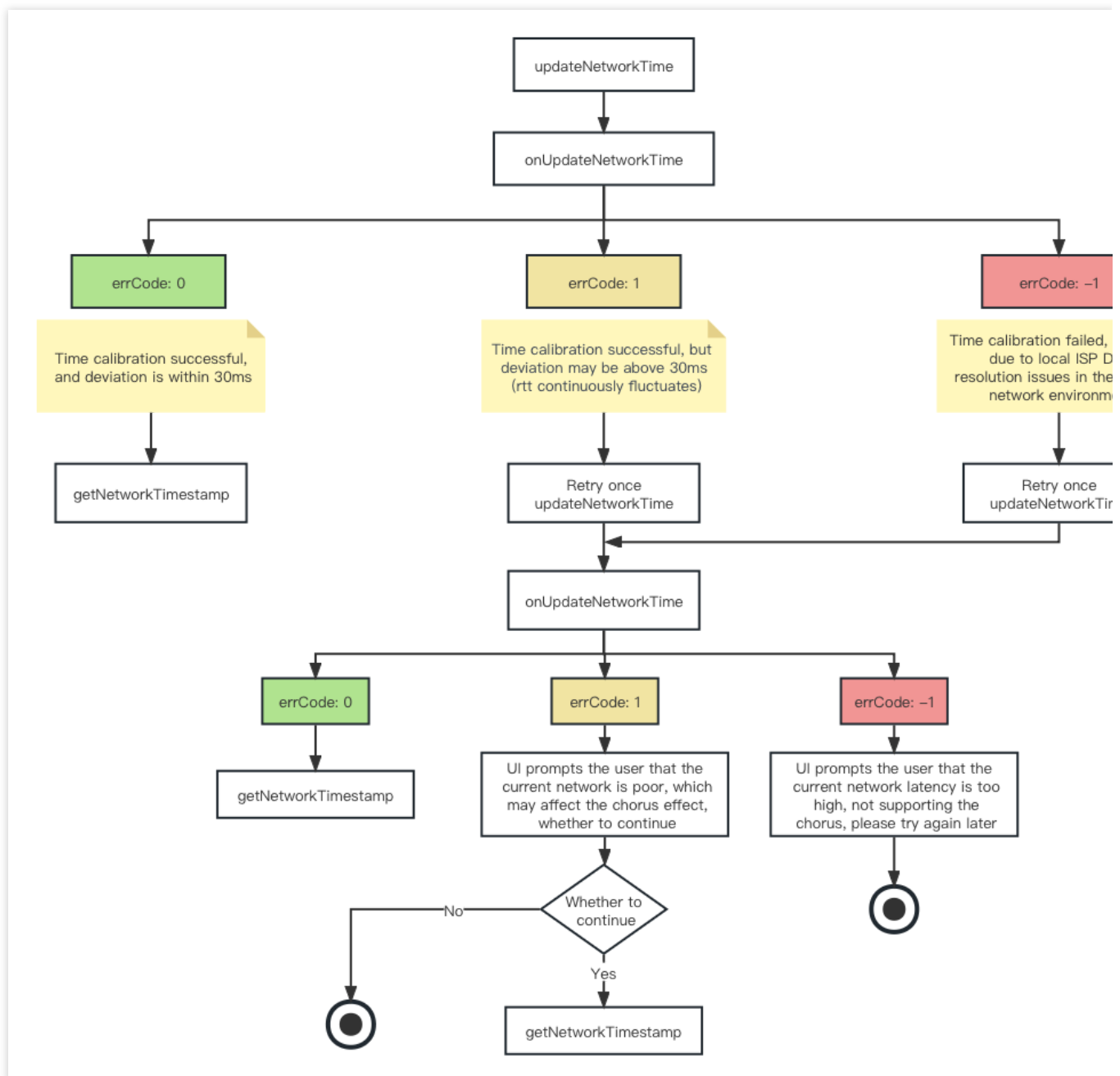
**Reminder: “NTP time sync finished, but result maybe inaccurate” ?**

NTP time synchronization is successful, but the deviation may be more than 30ms, reflecting poor client network environment and continuous rtt jitter.

### Reminder: “Error in AddressResolver: No address associated with hostname” ?

NTP time synchronization failure may be due to temporary anomalies in the local carrier DNS resolution under the current network environment. Please try again later.

### NTP service retry processing logic?

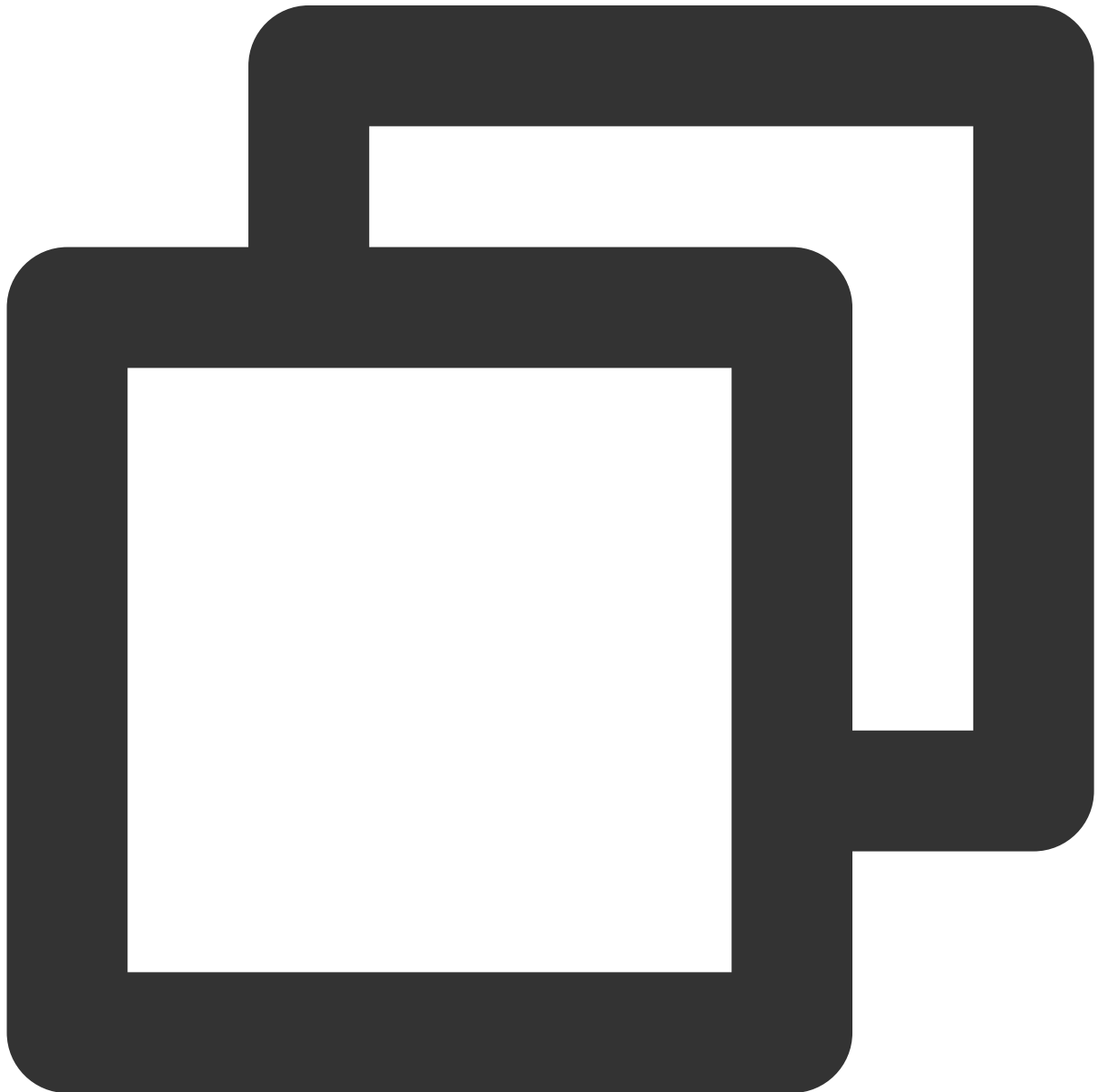




## Network Speed Test Recommendations

Online Karaoke scenarios have high network requirements for users, especially real-time chorus. A high-quality and stable network environment is necessary for a good Karaoke experience. Therefore, it is recommended to perform a network speed test on the user before entering the room, and give a UI layer reminder to users who do not meet the network requirements, prohibiting them from joining the Karaoke room or participating in chorus.

Initiating network speed test with TRTC SDK:



```
TRTCCloudDef.TRTCSpeedTestParams speedTestParams = new TRTCCloudDef.TRTCSpeedTestPa  
speedTestParams.sdkAppId = SDK_APP_ID;
```

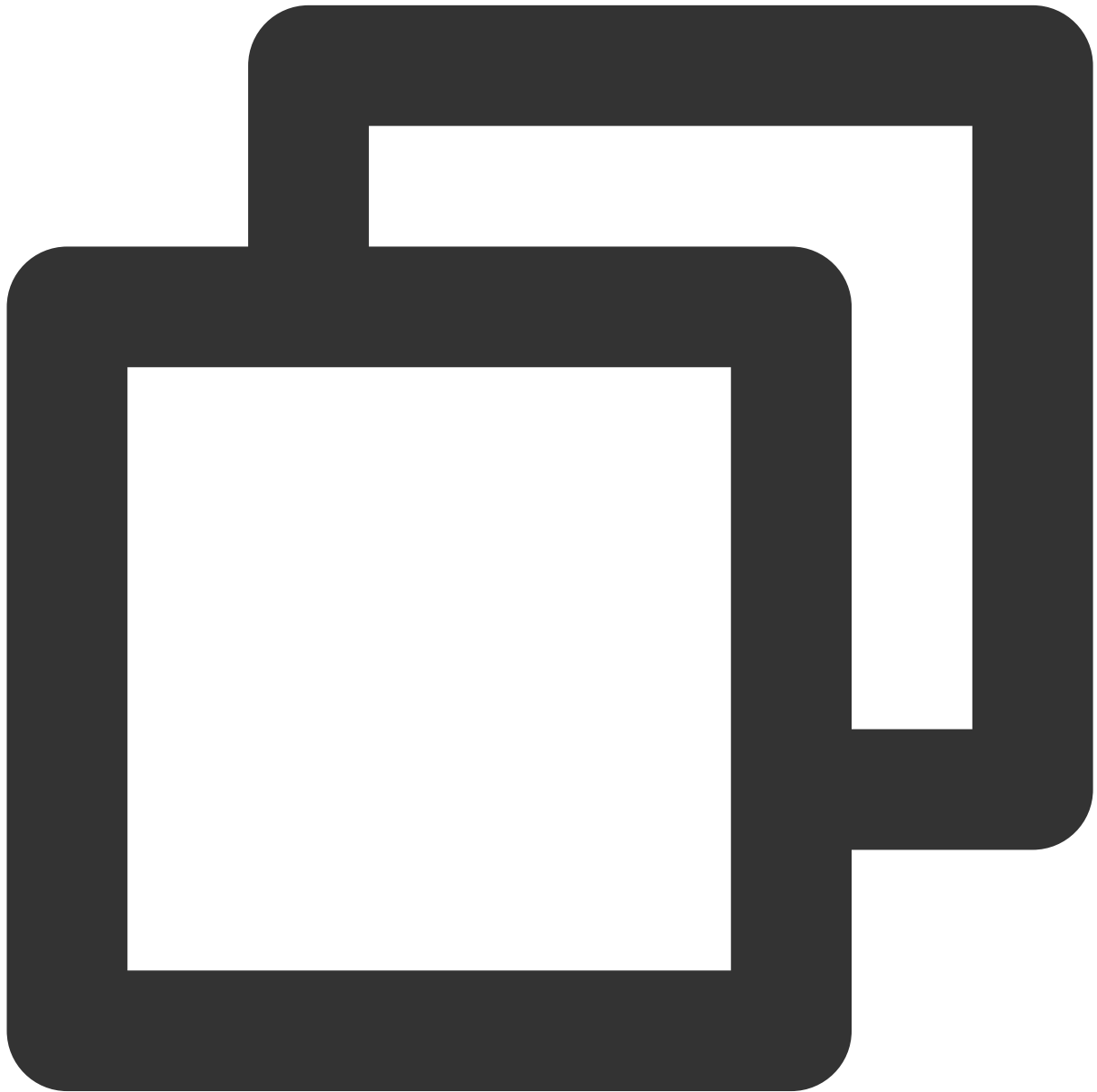
```
speedTestParams.userId = userId;
speedTestParams.userSig = userSig;
// If the actual bandwidth is higher than the expected value, the test result is th
// if the actual bandwidth is lower than the expected value, the test result is the
speedTestParams.expectedDownBandwidth = 3000; // Expected downstream bandwidth, ran
speedTestParams.expectedUpBandwidth = 3000; // Expected upstream bandwidth, ranging
mTRTCCloud.startSpeedTest(speedTestParams);
```

**Note:**

Expected upstream bandwidth, ranging from 10 to 5000 kbps

Please perform the network speed test before entering the room. Network speed testing in the room will affect the normal audio and video transmission effects, and due to too much interference, the network speed test results will also be inaccurate.

TRTC SDK network speed test result callback:



```
@Override
public void onSpeedTestResult(TRTCCloudDef. TRTCSpeedTestResult result) {
    String tquality = "Unknown";
    switch (result.quality) {
        case 0:
            tquality = "Unknown";
            break;
        case 1:
            tquality = "The current network is very good";
            break;
        case 2:
```

```
        tquality = "The current network is good";
        break;
    case 3:
        tquality = "The current network is average";
        break;
    case 4:
        tquality = "The current network is poor";
        break;
    case 5:
        tquality = "The current network is very poor";
        break;
    case 6:
        tquality = "The current network does not meet TRTC's minimum requiremen
        break;
}

if (result.success) {
    mTextTestResult.append("Speed test successful!" + "\\n");
    mTextTestResult.append("IP address: " + result.ip + "\\n");
    mTextTestResult.append("Uplink packet loss rate: " + result.upLostRate + "\
    mTextTestResult.append("Downlink packet loss rate: " + result.downLostRate
    mTextTestResult.append("Network latency: " + result.rtt + "ms\\n");
    mTextTestResult.append("Downlink bandwidth: " + result.availableDownBandwid
    mTextTestResult.append("Uplink bandwidth: " + result.availableUpBandwidth +
    mTextTestResult.append("Network quality: " + tquality + "\\n");
} else {
    mTextTestResult.append("Speed test failed!" + "\\n");
    mTextTestResult.append("Error message: " + result.errMsg + "\\n");
}
}
```

## Joining a Chorus Midway

The real-time chorus solution theoretically has no limit on the number of chorus participants, supporting multiple people to participate in the chorus simultaneously, as well as joining the chorus midway.

The following are the key actions for joining a chorus midway:

NTP time synchronization

Enable chorus experimental interface

Enter the room and start streaming on the microphone

Receive chorus signaling, obtain accompaniment resources and chorus agreed time

Calculate the difference between the agreed time and the current time, preload and seek accompaniment

Start participating in the chorus and synchronize accompaniment progress and lyrics progress in real-time

For the specific implementation process and code implementation of the above key actions, please refer to the documentation on [song synchronization](#), [lyrics synchronization](#), [vocal synchronization](#).