

Tencent Real-Time Communication Solution 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

Solution

Real-Time Chorus

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

Solution

Real-Time Chorus

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](#).

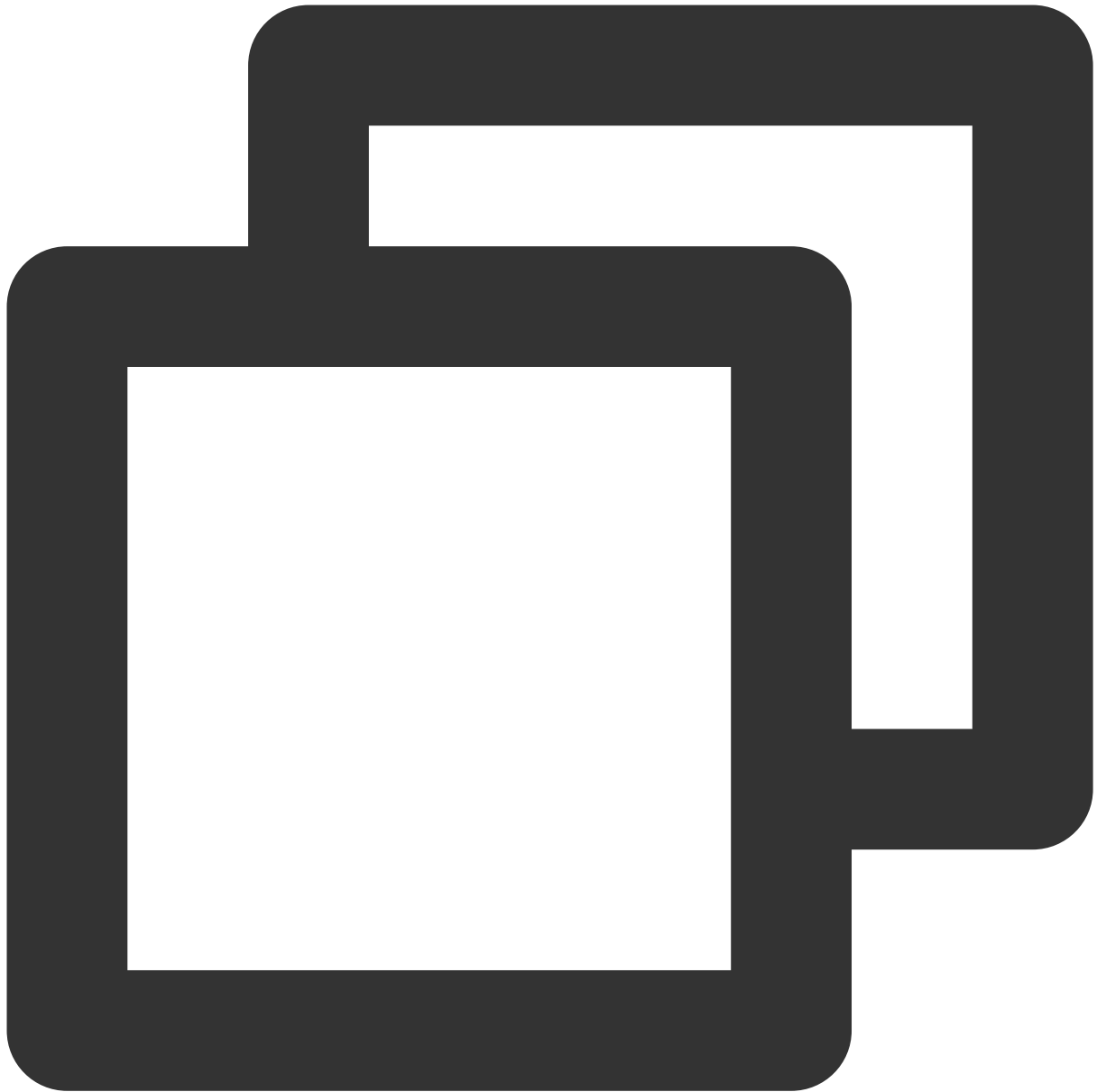


Integration

Step 1. Download and import the `TUIKaraoke` component

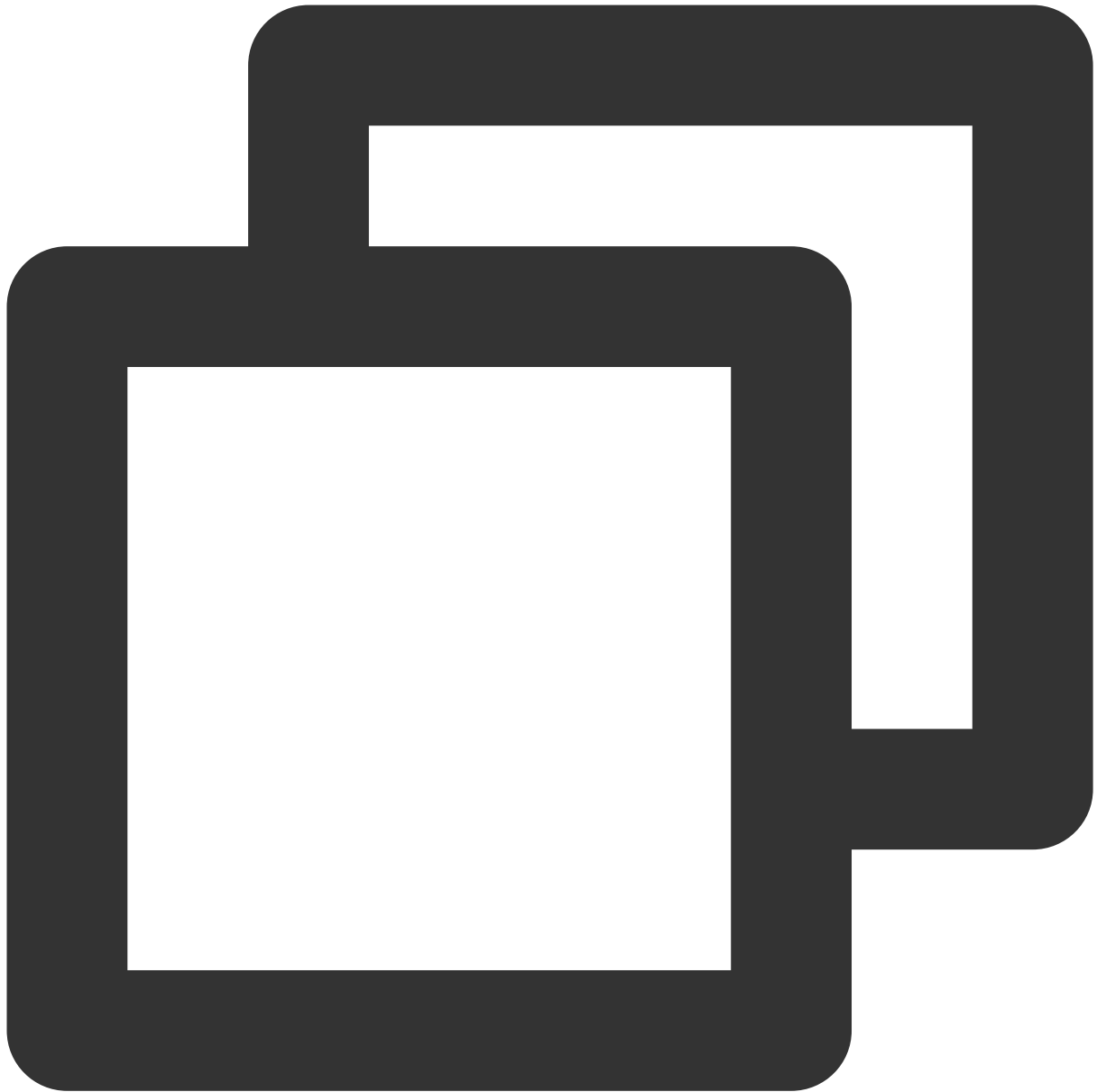
Go to [GitHub](#), clone or download the code, copy the `Source`, `Resources`, and `TXAppBasic` folders and the `TUIKaraoke.podspec` file in the `ios` directory to your project, and complete the following import operations:

Add the following import commands to your `Podfile`:



```
pod 'TUIKaraoke', :path => "./", :subspecs => ["TRTC"]
pod 'TXLiteAVSDK_TRTC'
pod 'TXAppBasic', :path => "TXAppBasic/"
```

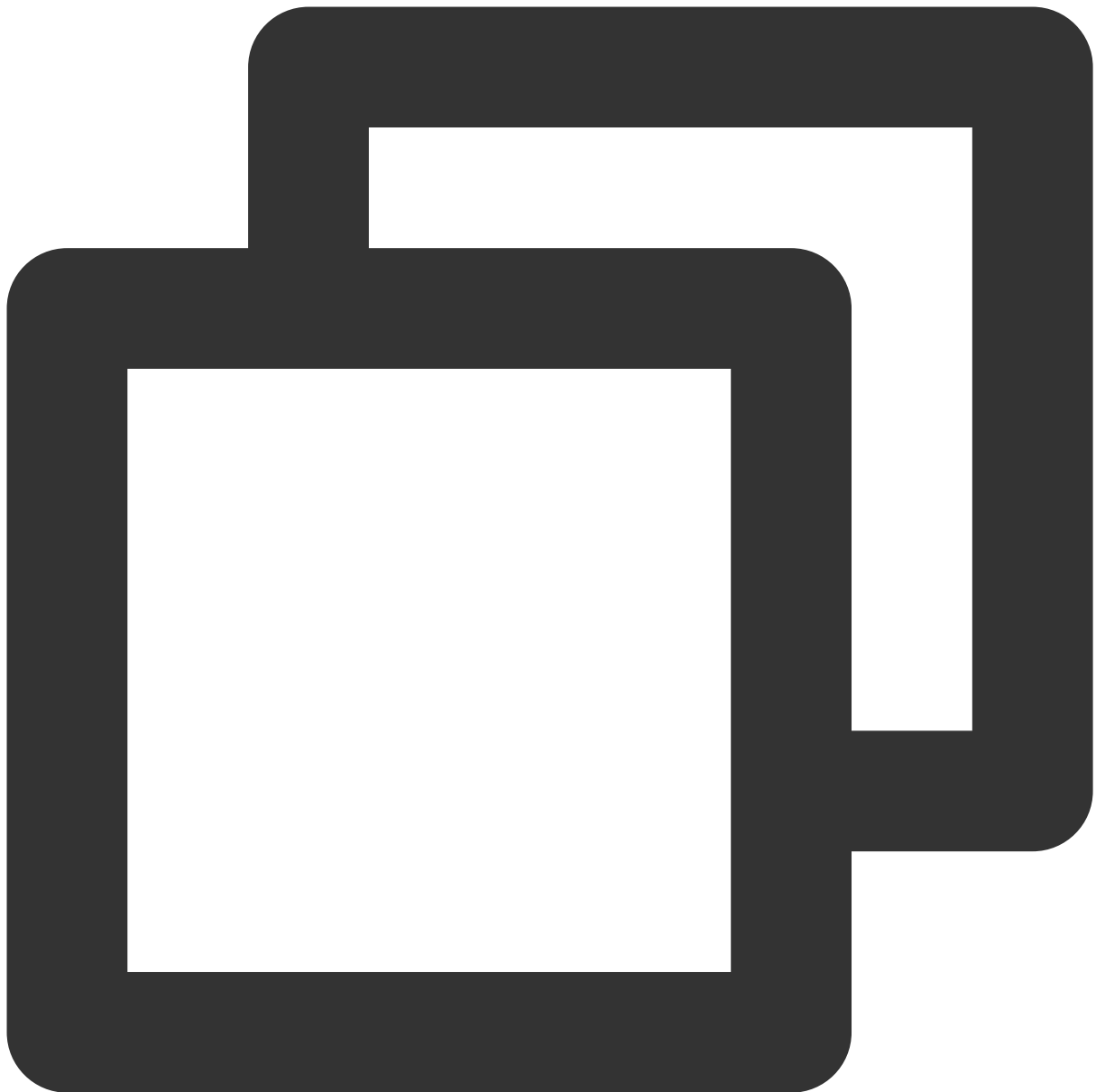
Open Terminal and run the following installation command in the directory of the `Podfile` :



```
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` .

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), 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 three tabs: Application Info, Function Configuration, and Callback Configuration. Under the Application Info tab, there are two instructional steps. Step 1 is 'download SDK + auxiliary demo source' with links for various platforms. Step 2 is 'obtain the secret key to issue UserSig', which includes a warning about the key's sensitivity and a text input field for the 'Secret Key (Key)'. The key field is blurred and highlighted with a red box. Below the field is a '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: 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`.

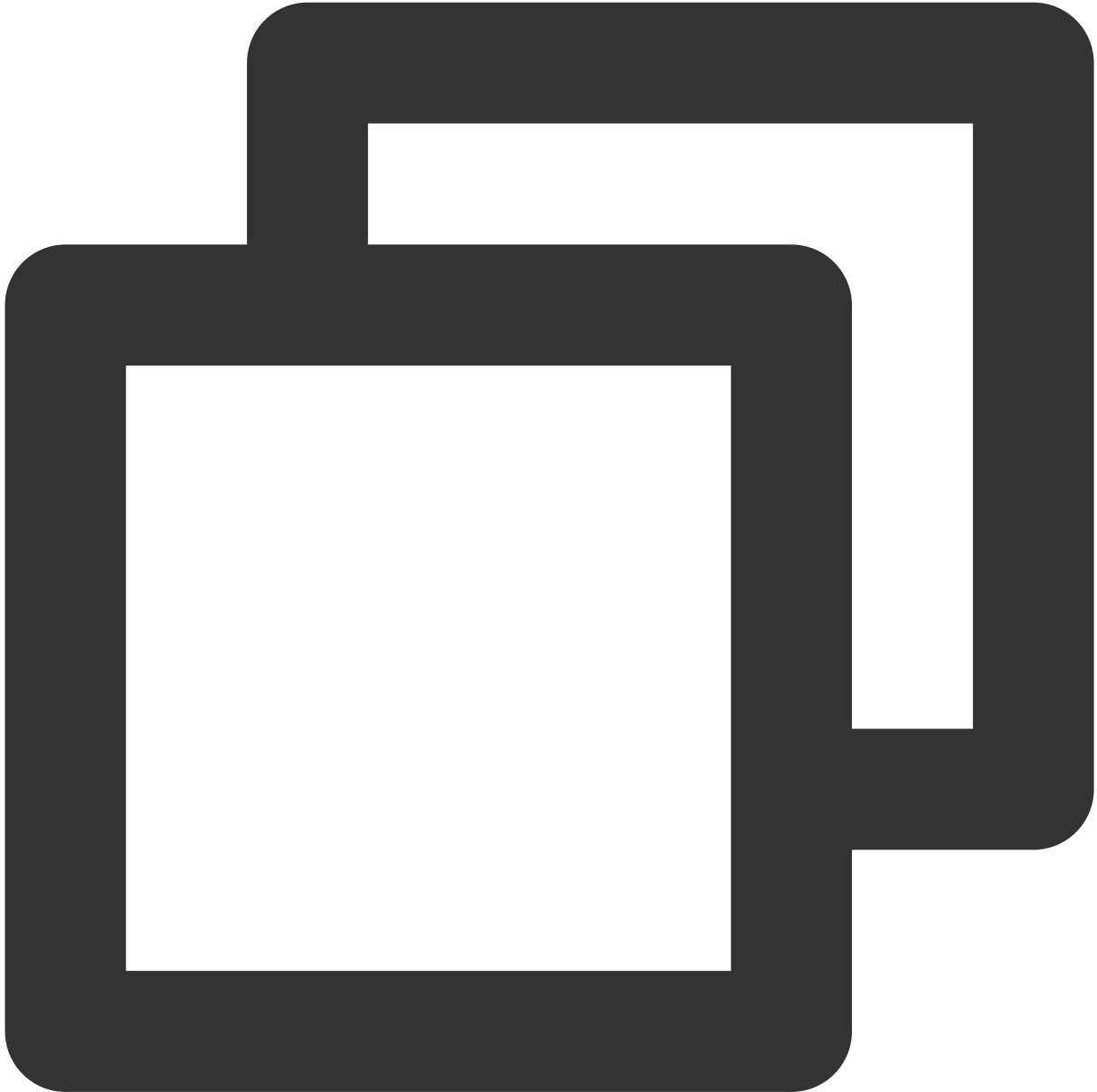


```
int roomId = "Room ID";
let param = RoomParam.init()
param.roomName = "Room name";
param.needRequest = false; // Whether permission is required for listeners to speak
param.seatCount = 8;        // Number of seats in the room. Set it to `8`.
param.coverUrl = "URL of room cover image";

karaokeRoom.createRoom(roomID: Int32(roomInfo.roomID), roomParam: param) { [weak self]
    guard let `self` = self else { return }
    if code == 0 {
        // Room created successfully
    }
}
```

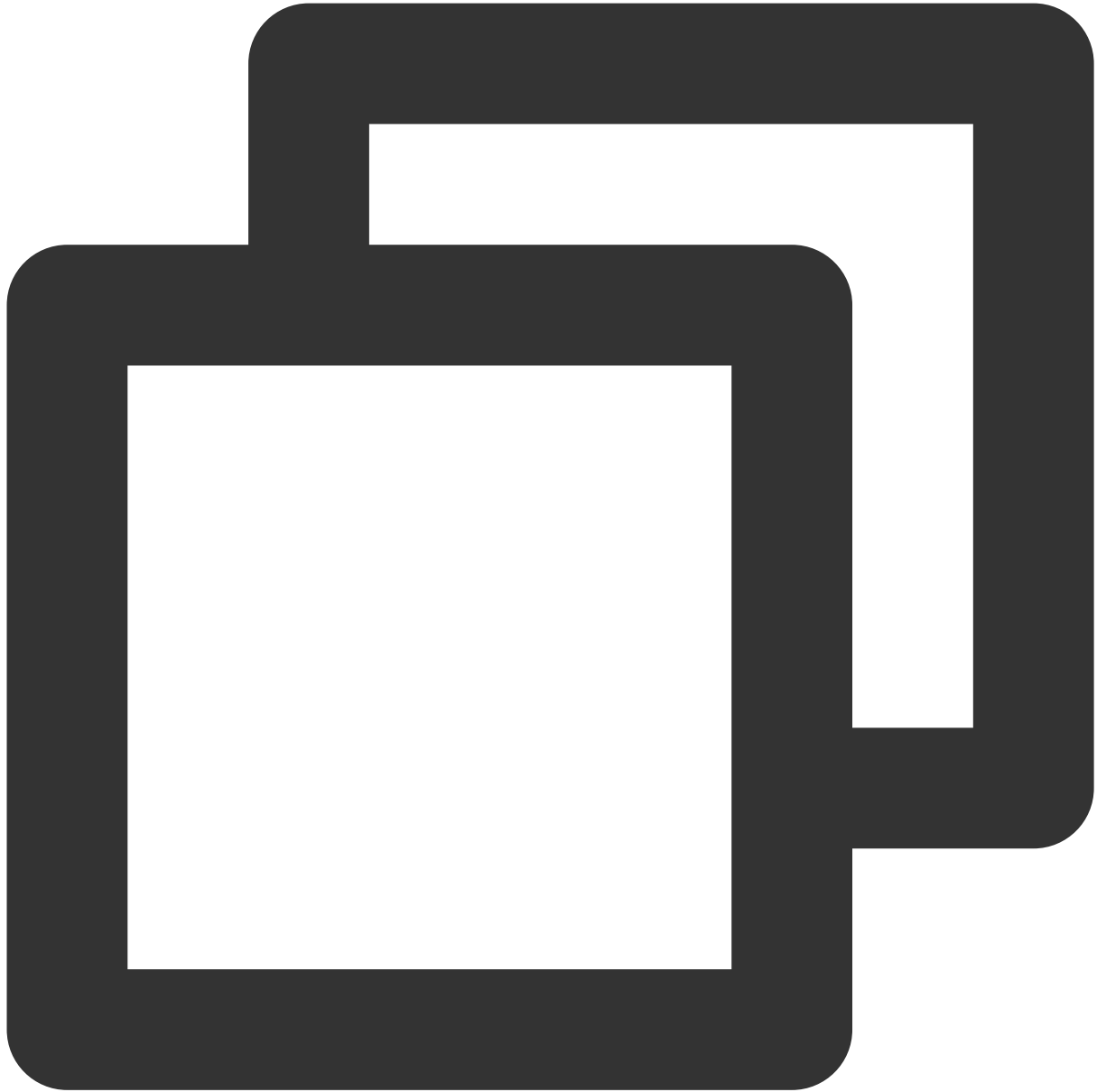
```
}  
}
```

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
```

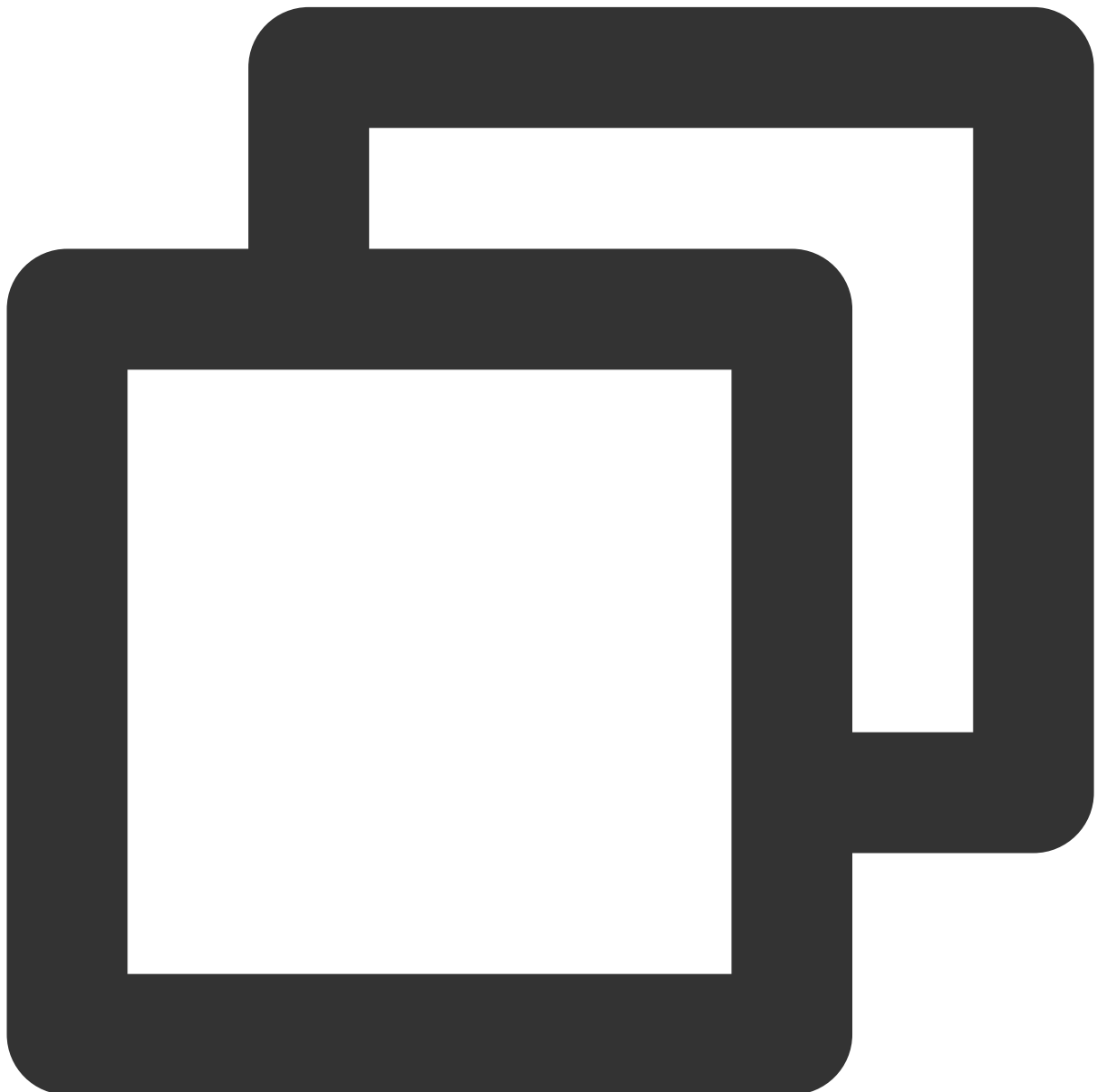
```
func onSeatListChange(seatInfoList: [SeatInfo]) {  
}
```

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: muscicLocalPath, accompan
```

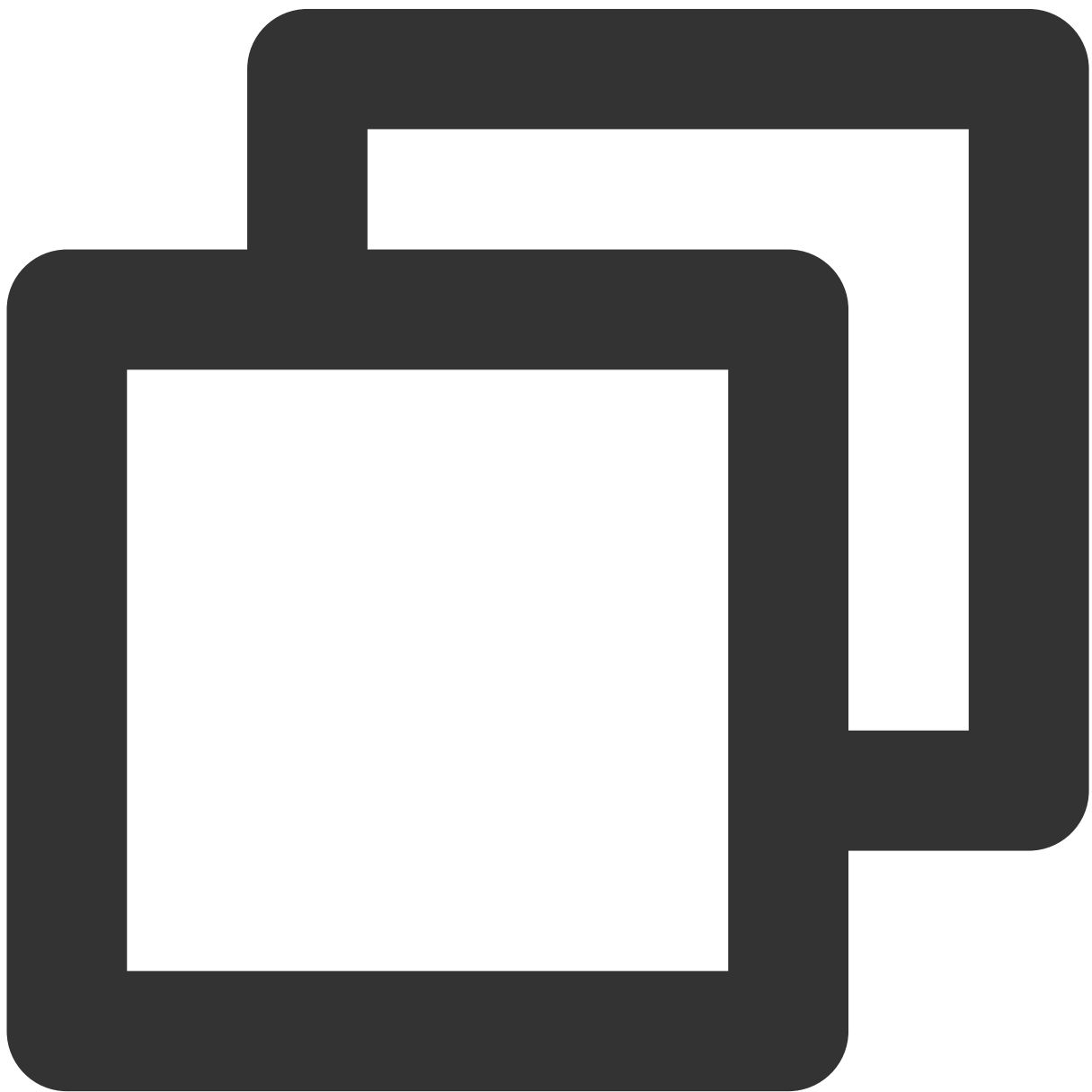
```
// Stop the music
karaokeRoom.stopPlayMusic();
```

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.

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:

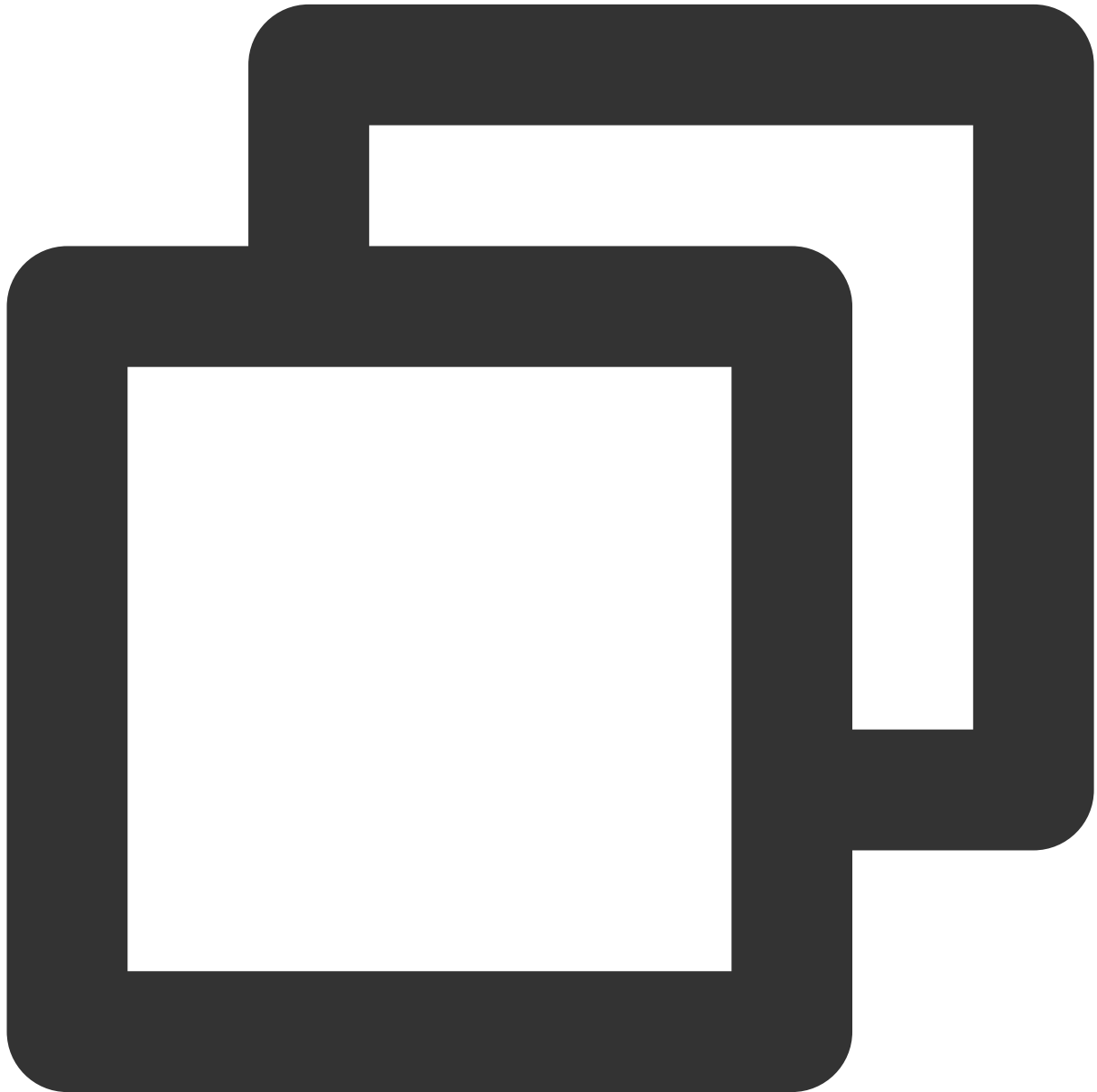


Component Integration

Step 1. Download and import the TUIKaraoke component

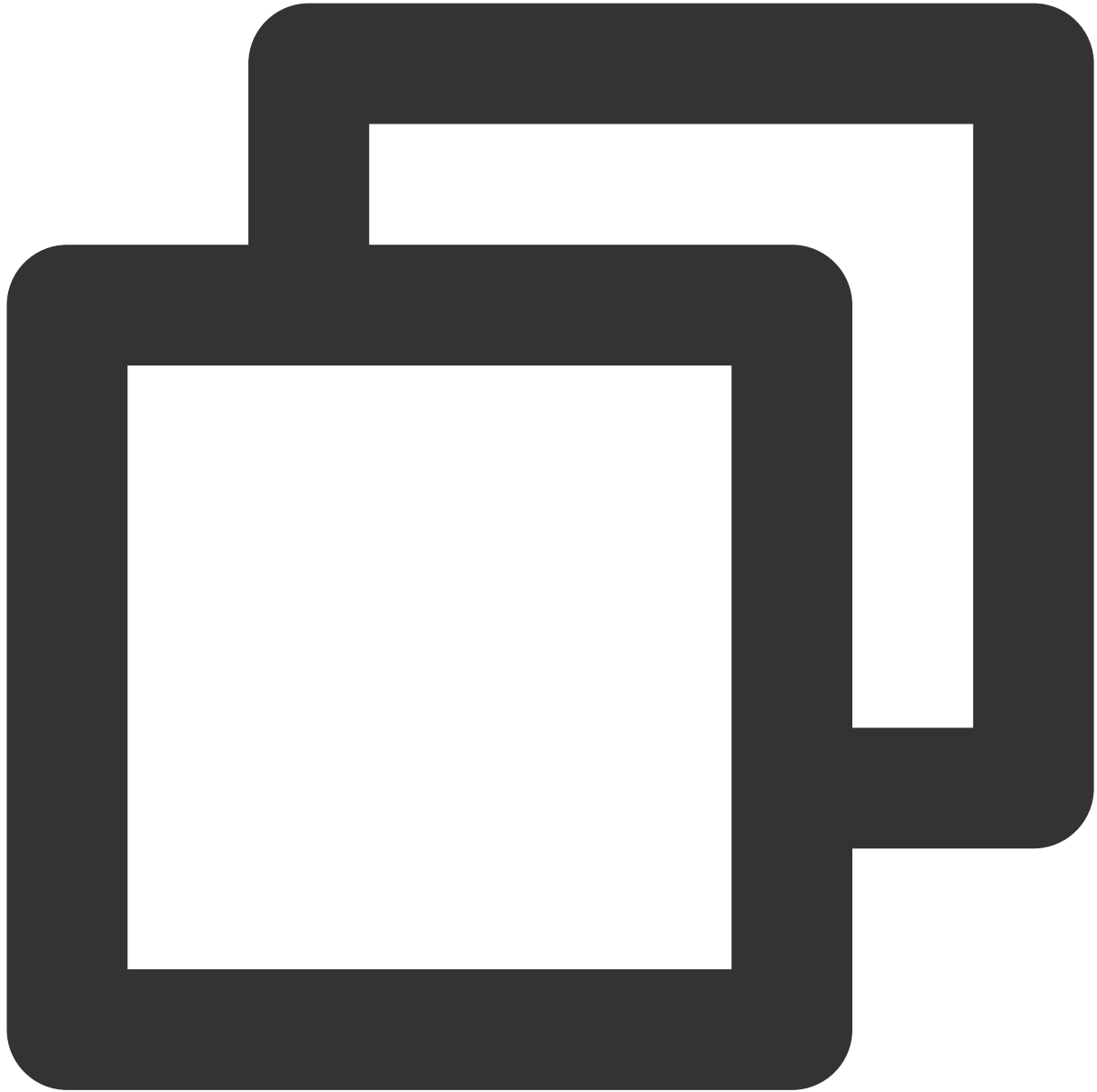
Go to [GitHub](#), clone or download the code, copy the `Source` and `Debug` directories in the `Android` directory to your project, and complete the following import operations:

Complete import in `setting.gradle` as shown below:



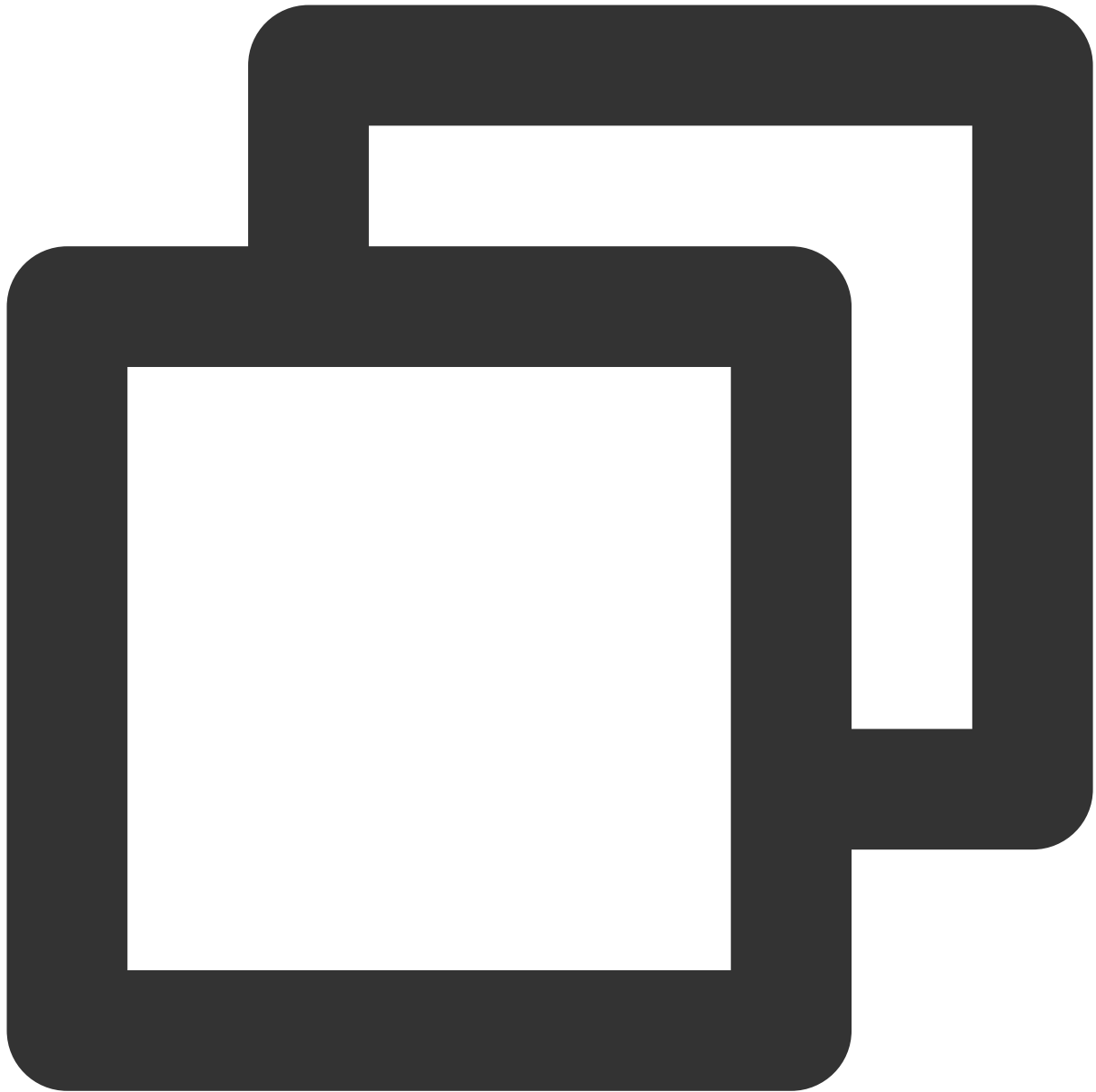
```
include ':Source'  
include ':Debug'
```

Add dependencies on `TUIKaraoke` to the `build.gradle` file in `app` :



```
api project(':Source')
```

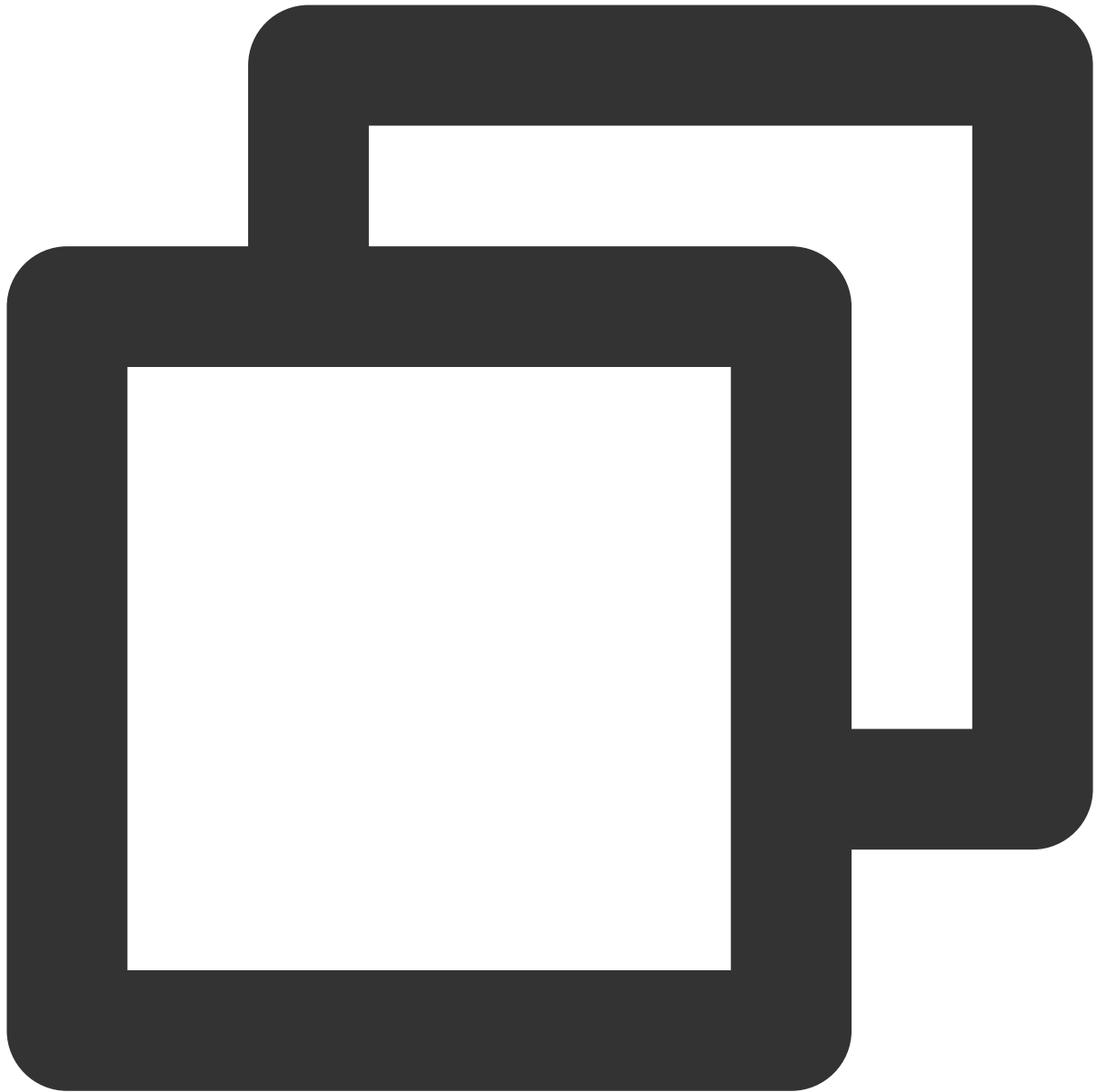
Add dependencies on `TRTC SDK` and `IM SDK` to the `build.gradle` file in the root directory:



```
ext {  
    liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTC1:latest.release"  
    imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"  
}
```

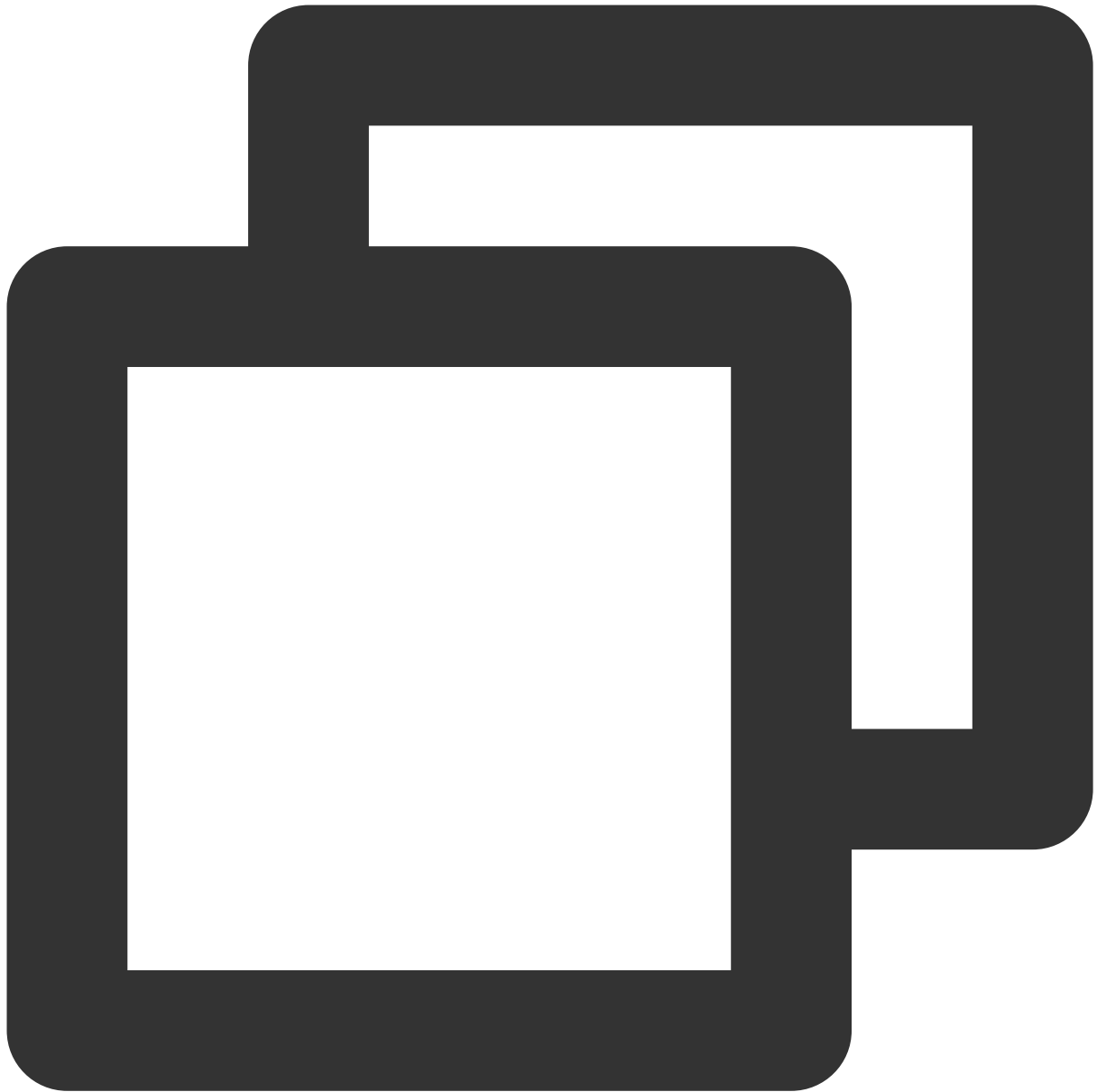
Step 2. Configure permission requests and obfuscation rules

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" /> //
```

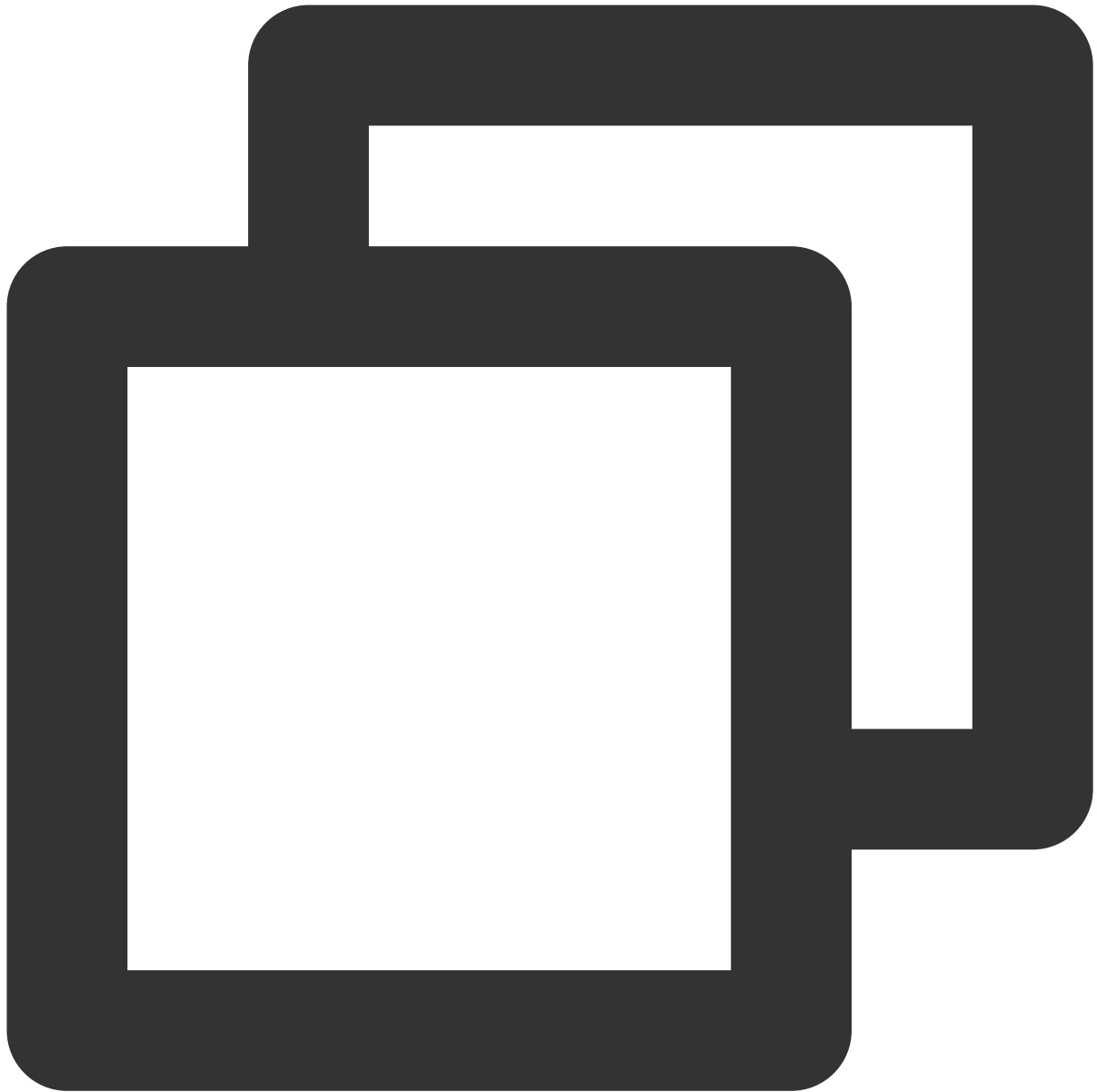
In the `proguard-rules.pro` file, add the SDK classes to the "do not obfuscate" list.



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

Step 3. Initialize and log in to the component

For more information on relevant APIs, see [TUIKaraoke](#).

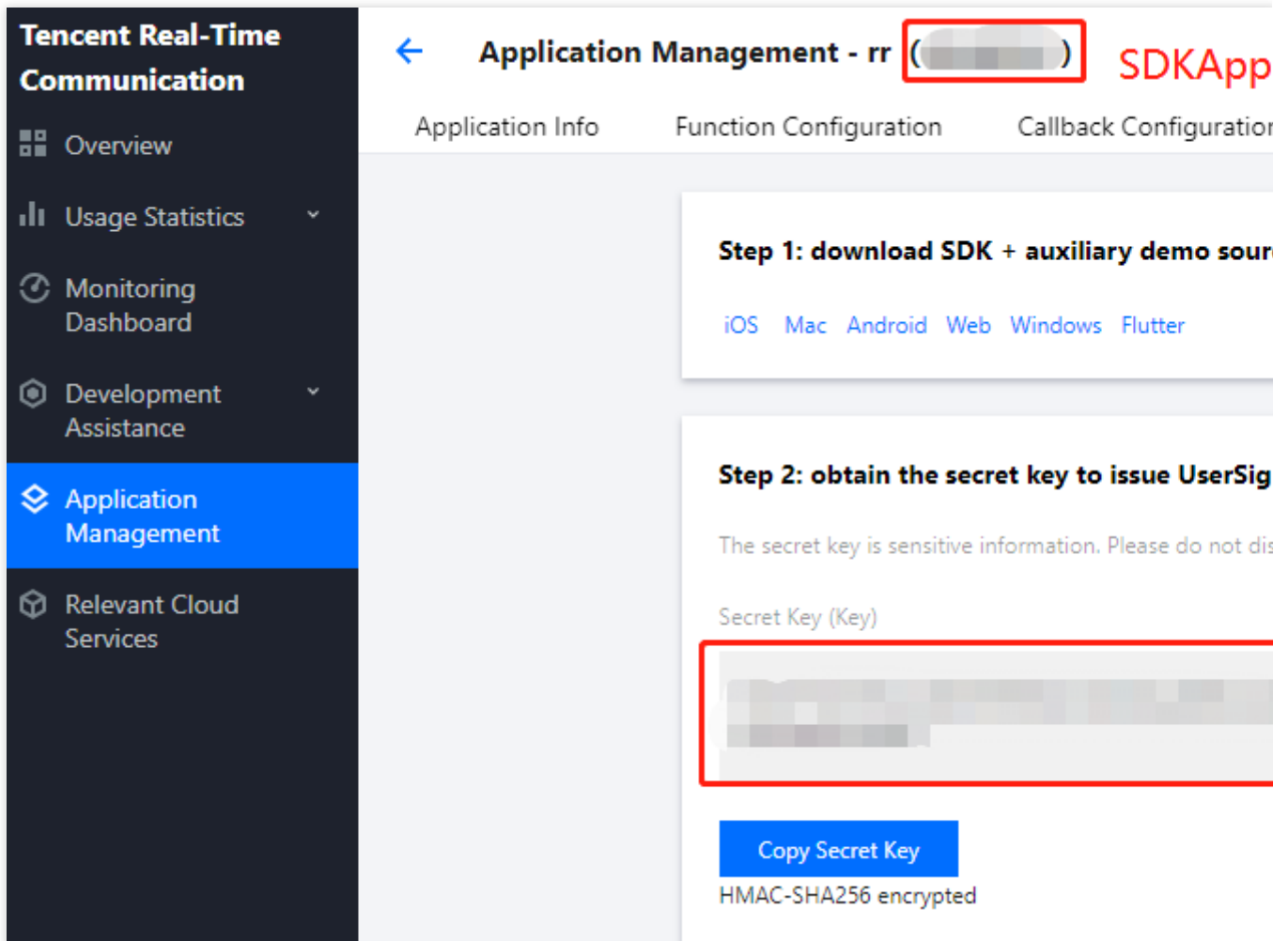


```
// 1. Initialize
TRTCKaraokeRoom mTRTCKaraokeRoom = TRTCKaraokeRoom.sharedInstance(this);
mTRTCKaraokeRoom.setDelegate(this);
// 2. Log in
mTRTCKaraokeRoom.login(SDKAppID, UserID, UserSig, new TRTCKaraokeRoomCallback.Act
    @Override
    public void onCallback(int code, String msg) {
        if (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:



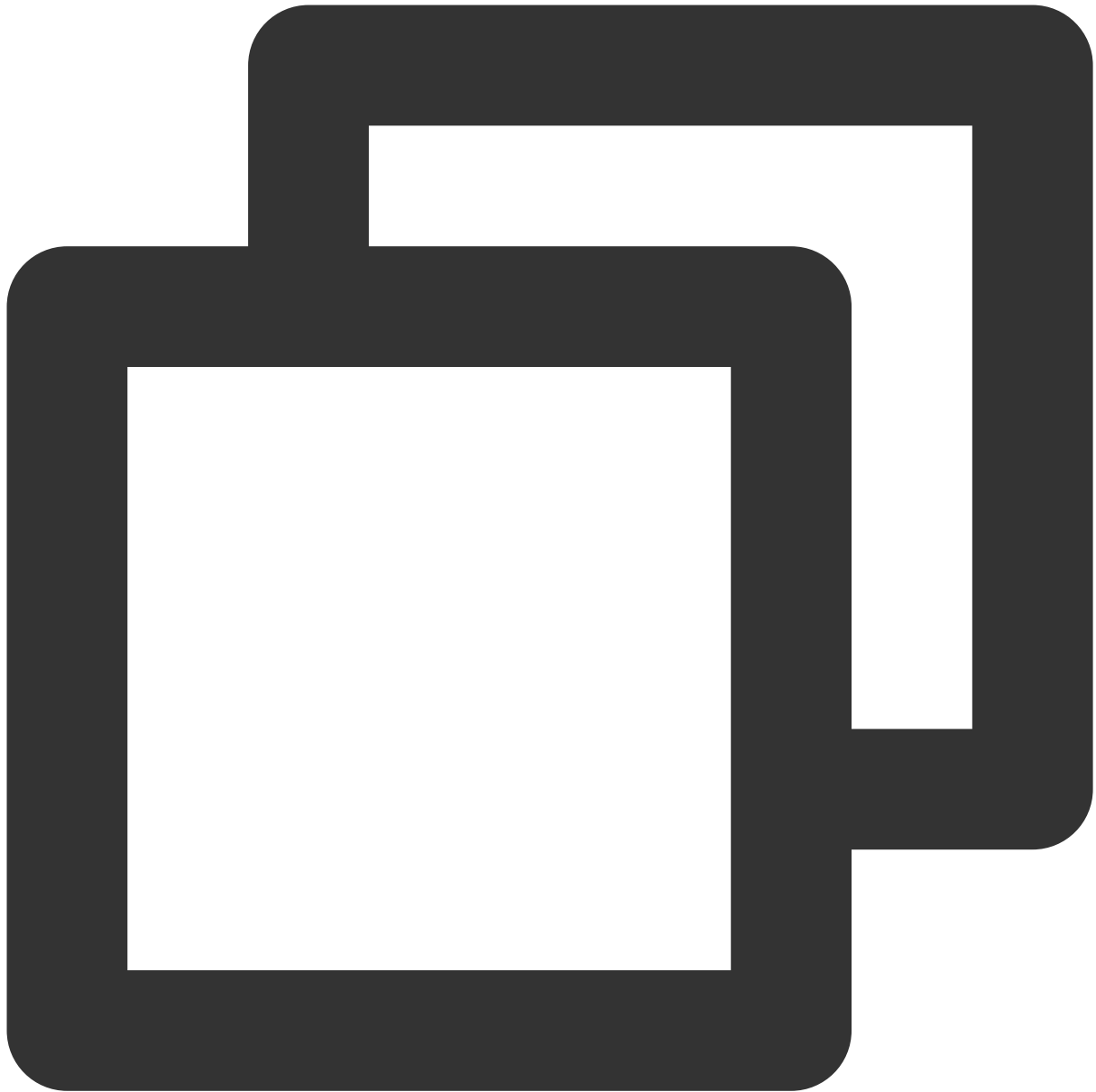
Secretkey: TRTC application key, which corresponds to `SDKAppID`. On the [Application Management](#) page in the TRTC console, the `SecretKey` is as shown below:

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

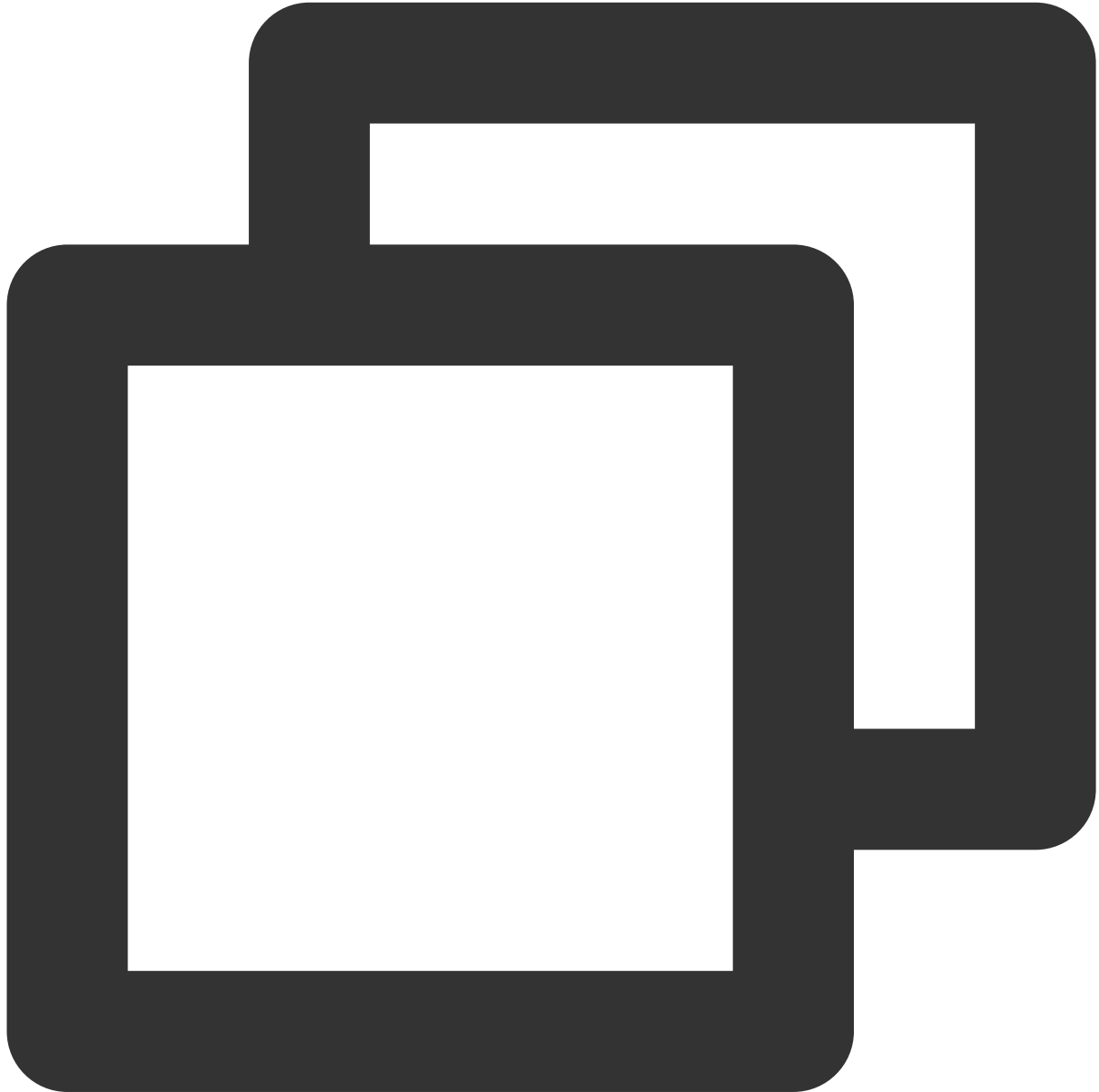
1. The anchor creates a room through [TUIKaraoke.createRoom](#).



```
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
    }
}
```

```
}  
}  
});
```

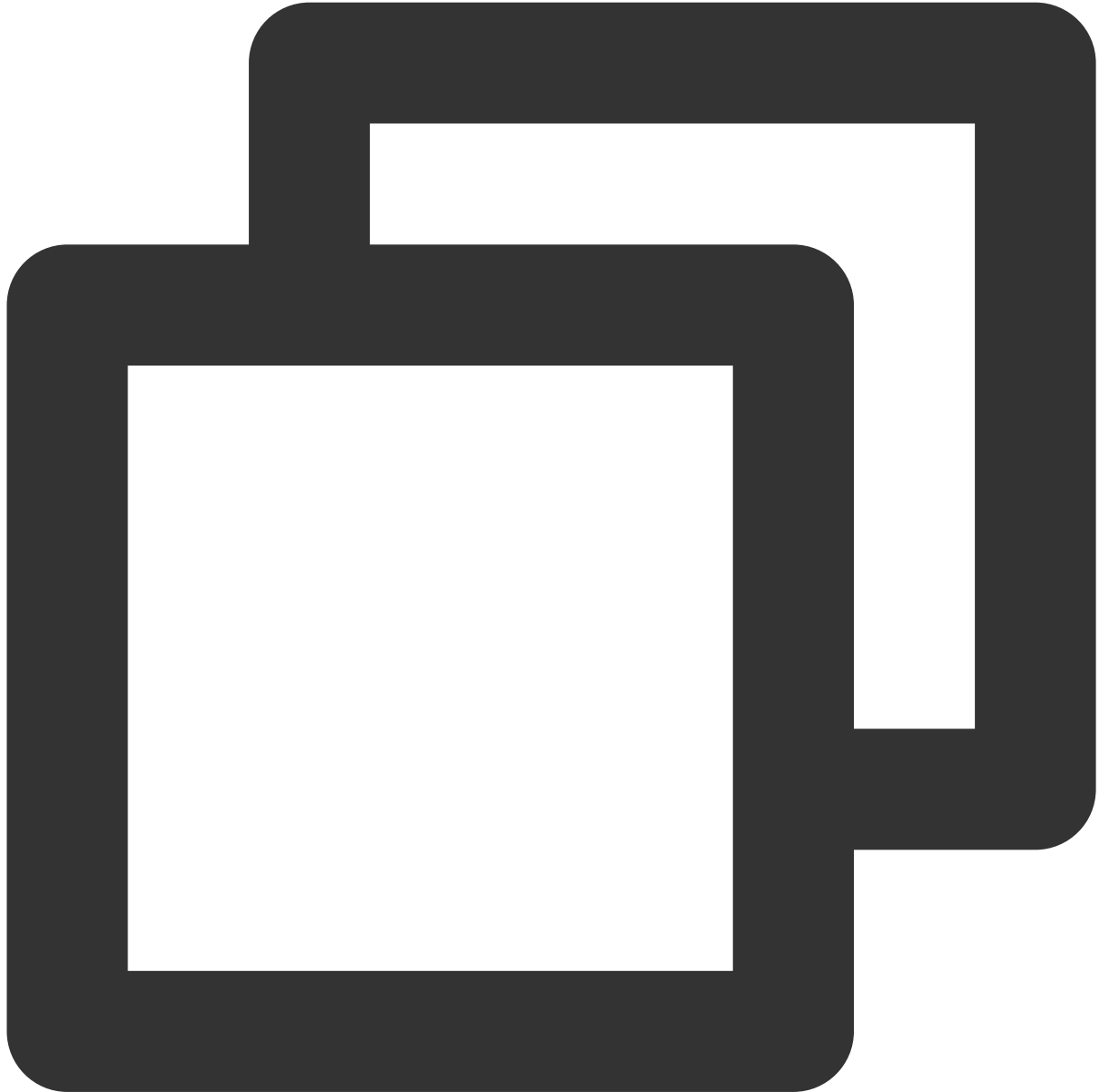
2. A listener enters the room through [TUIKaraoke.enterRoom](#).



```
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
```

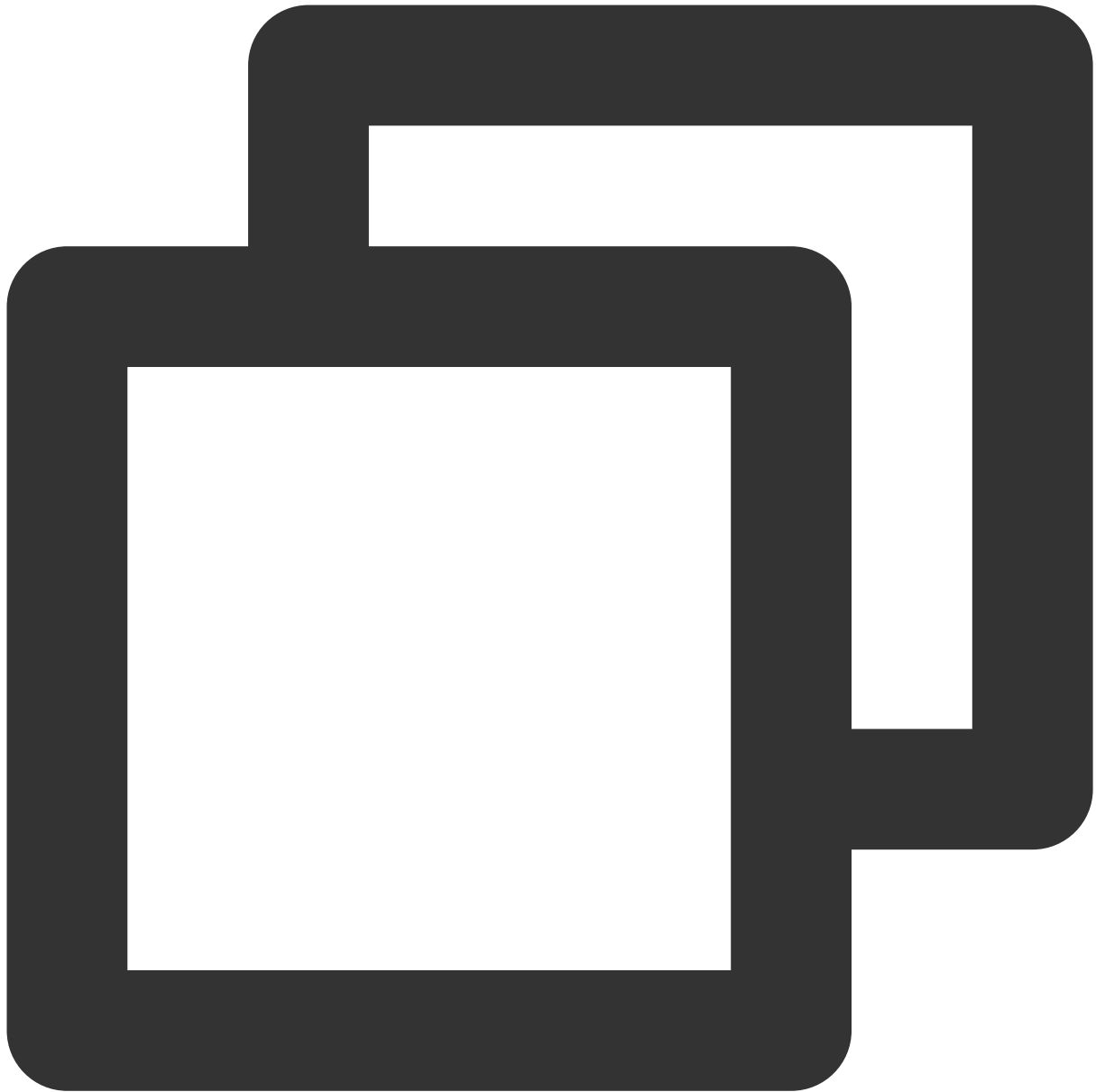
```
    }  
  }  
});  
// 2. The listener receives the `onSeatListChange` callback and refreshes the seat  
@Override  
public void onSeatListChange(final List<TRTCKaraokeRoomDef.SeatInfo> seatInfoList)  
{
```

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](#).

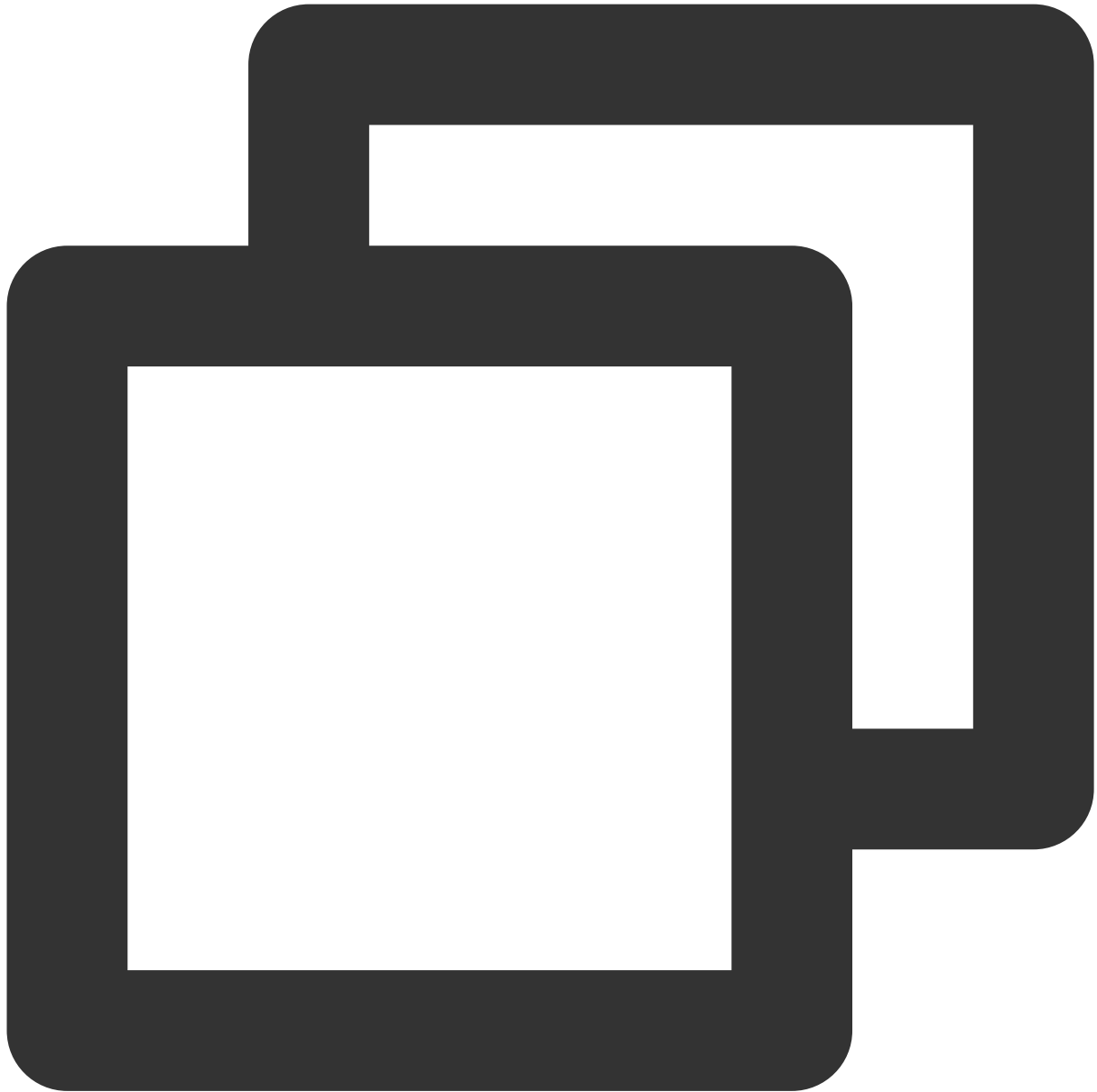


```
// Play back the music
mTRTCKaraokeRoom.startPlayMusic(musicID,url);
// Stop the music
mTRTCKaraokeRoom.stopPlayMusic();
```

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.

Solution Overview (TUIKaraoke)

Implementation Steps

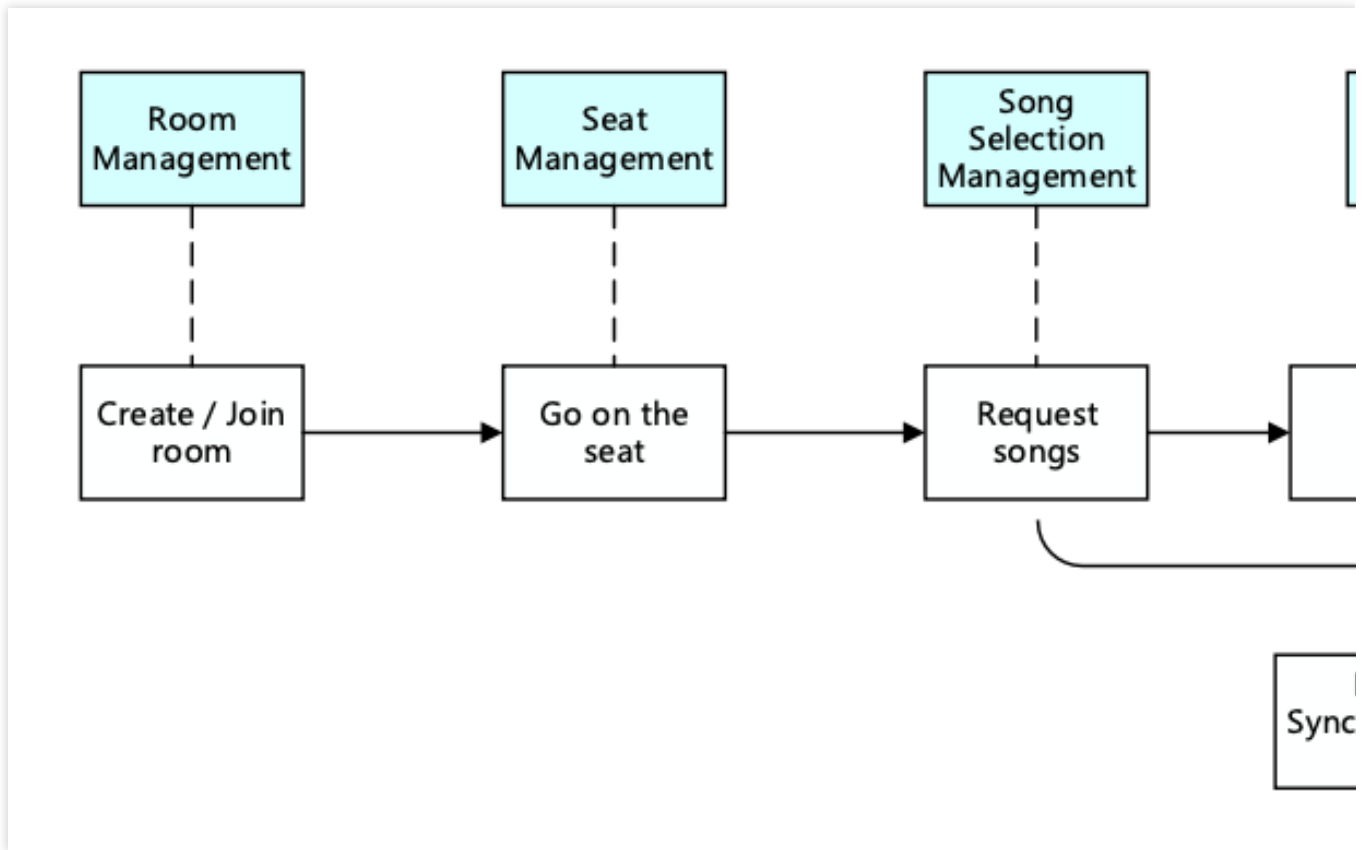
Last updated : 2023-09-26 16:38:31

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.

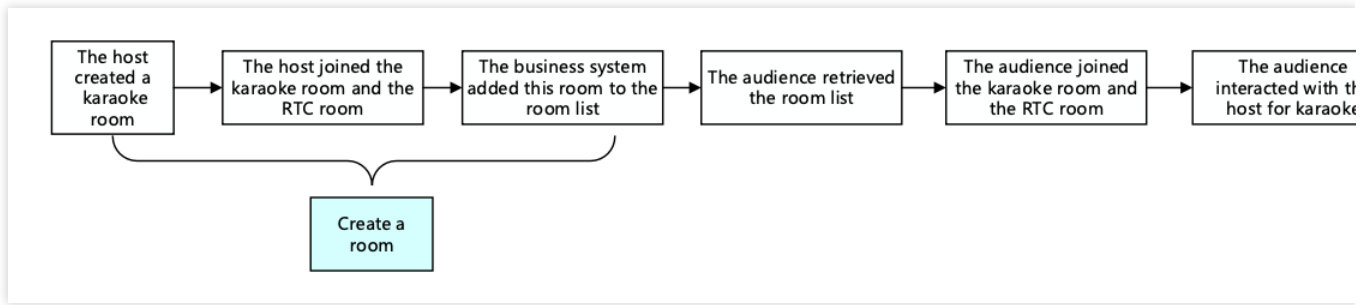


Room Management

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.

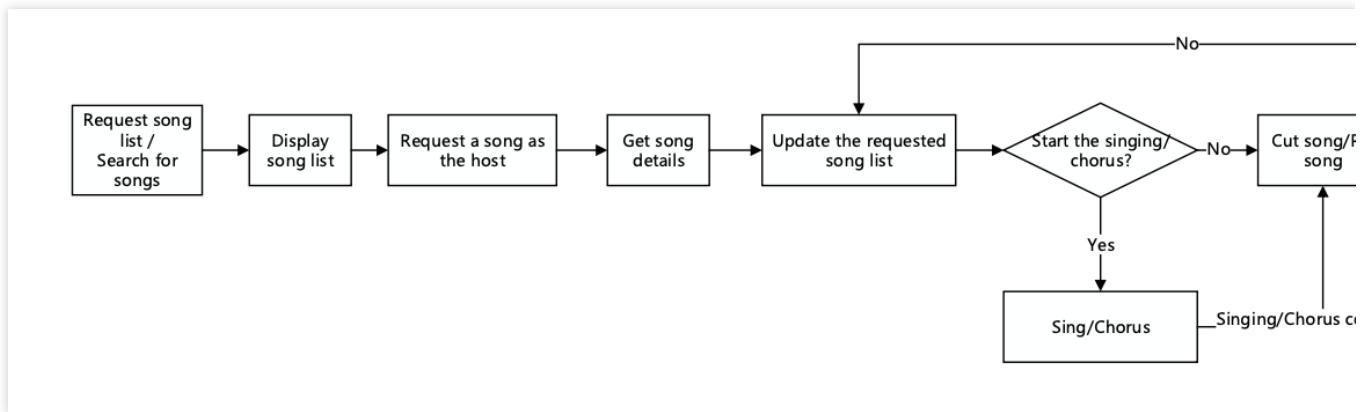
Song selection management

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

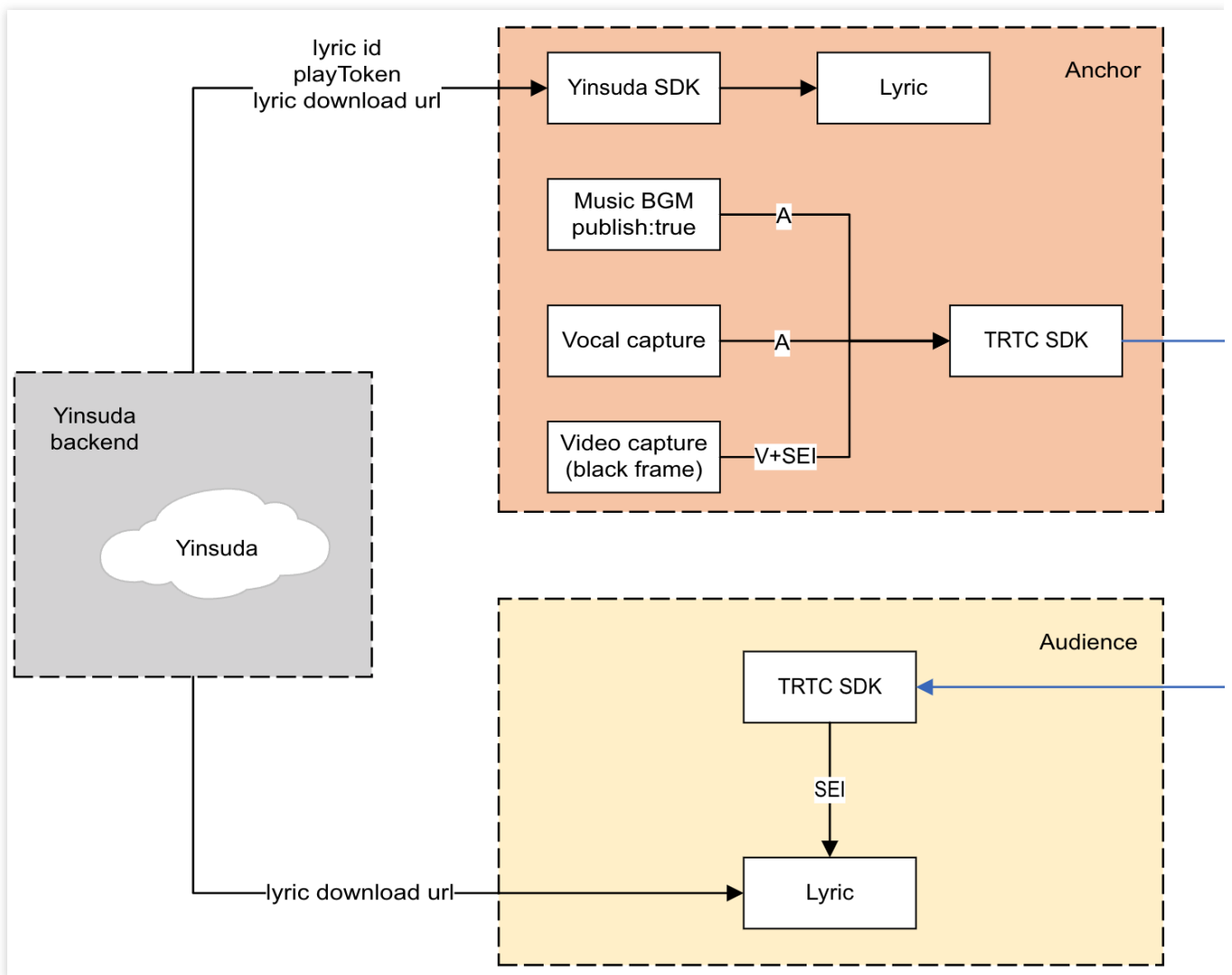
The karaoke system mainly includes the following functions: singing gameplay, start/stop/song cutting, vocal volume adjustment, reverb/sound effects, and lyric synchronization. Below, we will introduce the implementation process of the karaoke management module in detail through two typical karaoke gameplay: 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

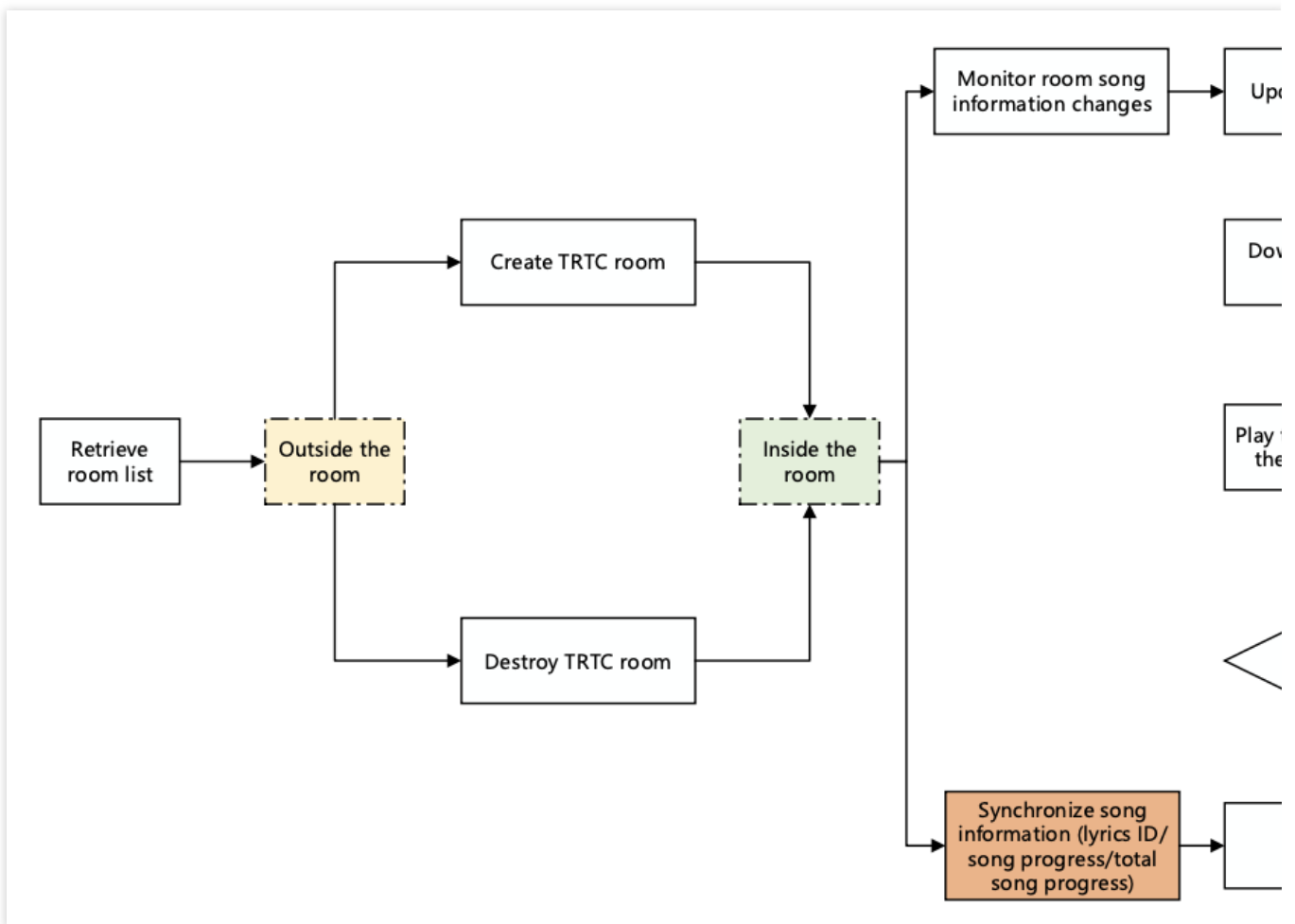
In the singing scenario, different roles have different implementation processes, which can be divided into two roles: singer and audience.

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 Upstream audio and video (no video upstream black frame) Play BGM Send SEI information (send lyric information) 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 Downstream audio and video streams 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.

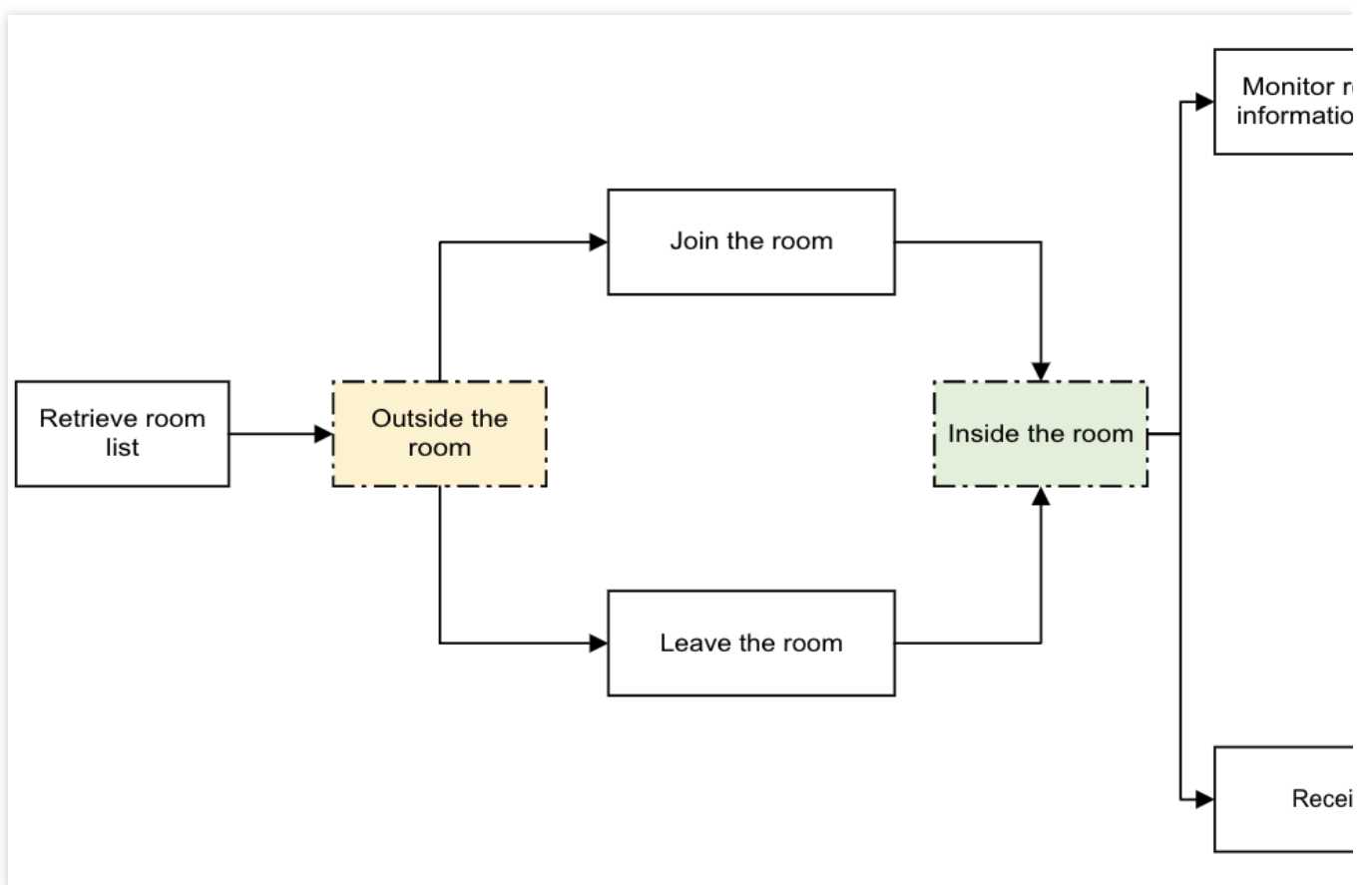
If the singer does not bring up the video upstream, they need to enable video upstream.

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.

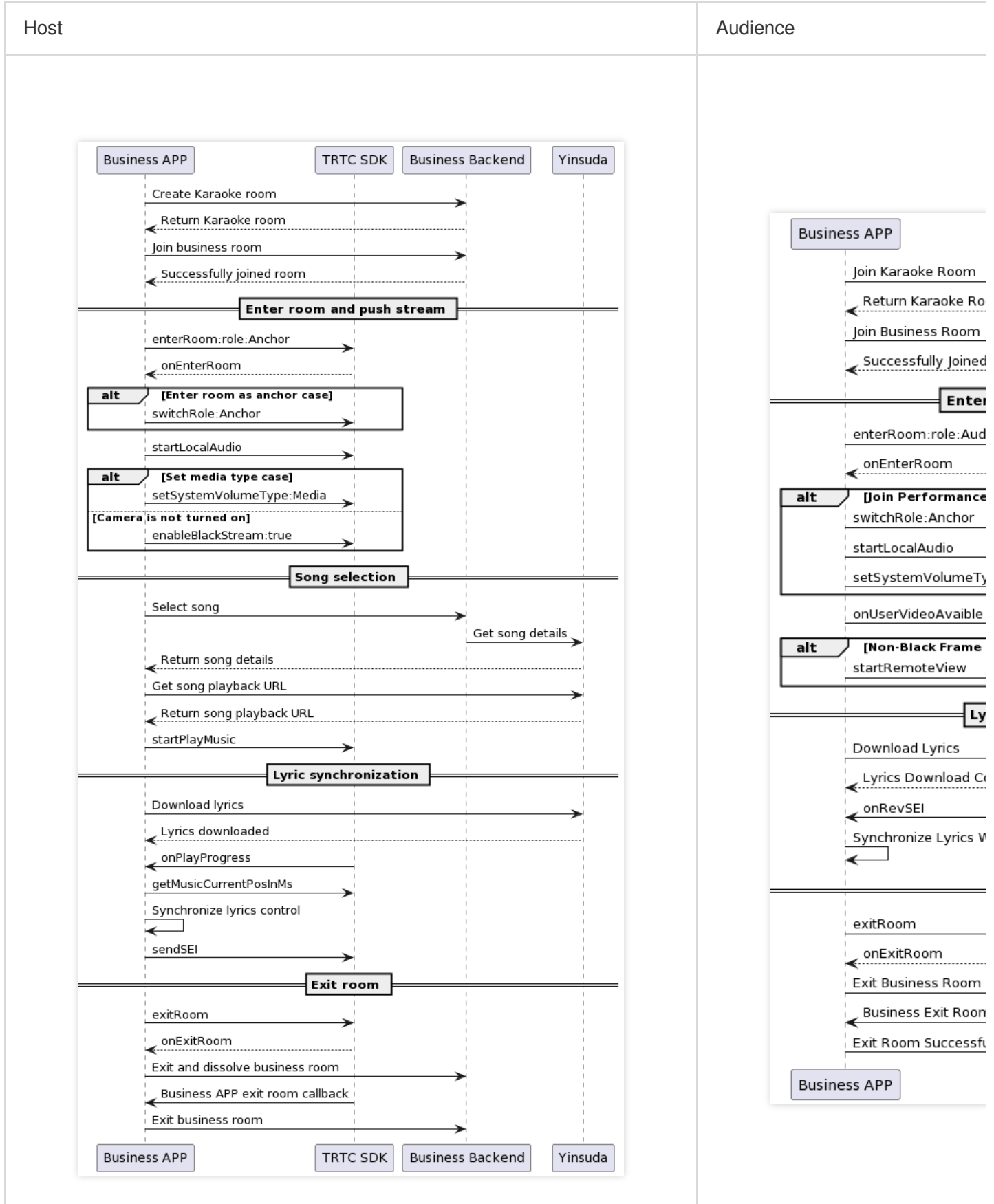
Pull the stream of the singer.

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 :

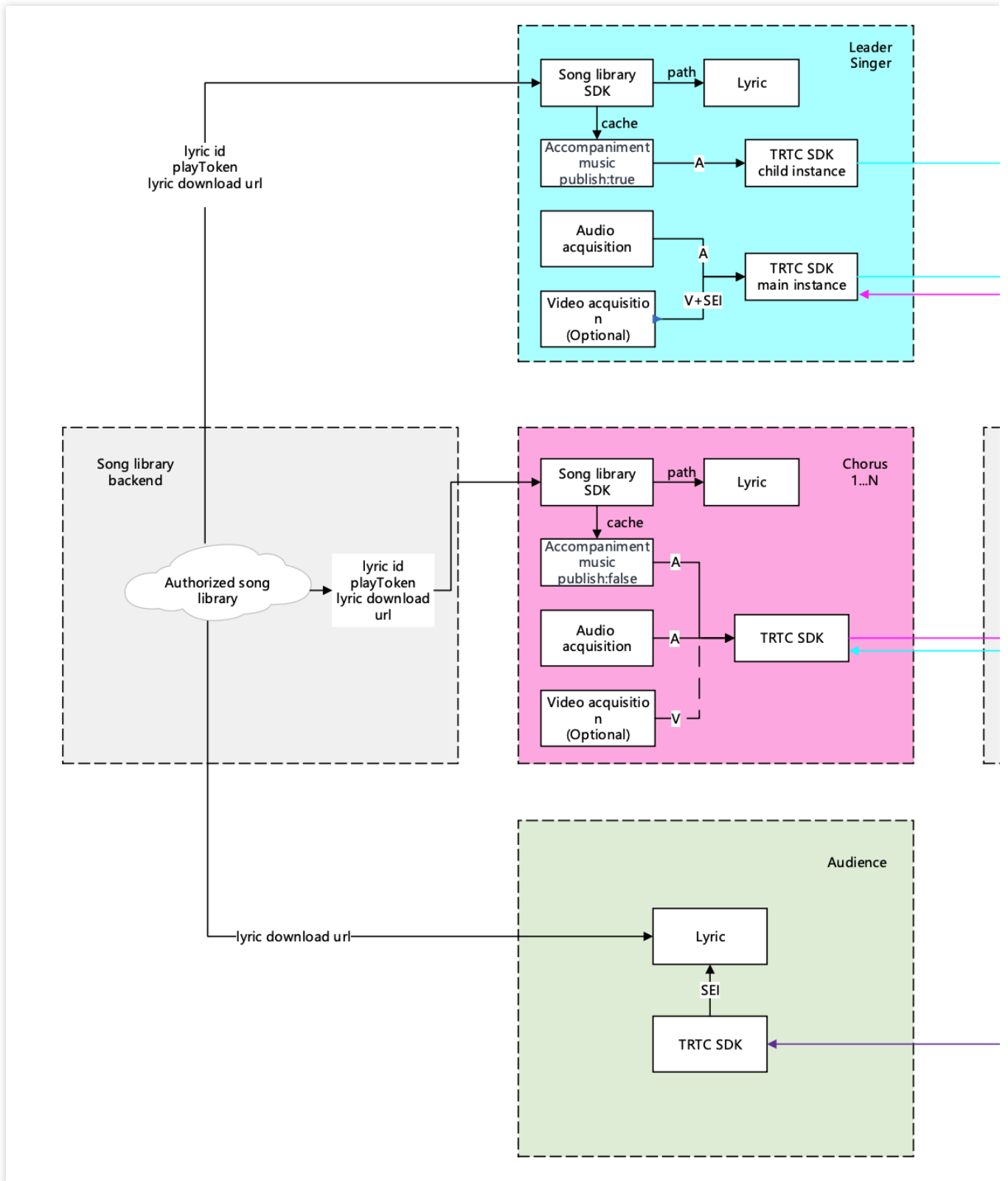
Given the technical threshold required for the above implementation solution, TRTC provides an open-source audio and video UI component called [TUIKaraoke](#) on its official website. By integrating the TUIKaraoke component into your project, you can add online karaoke scenes to your application with 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.

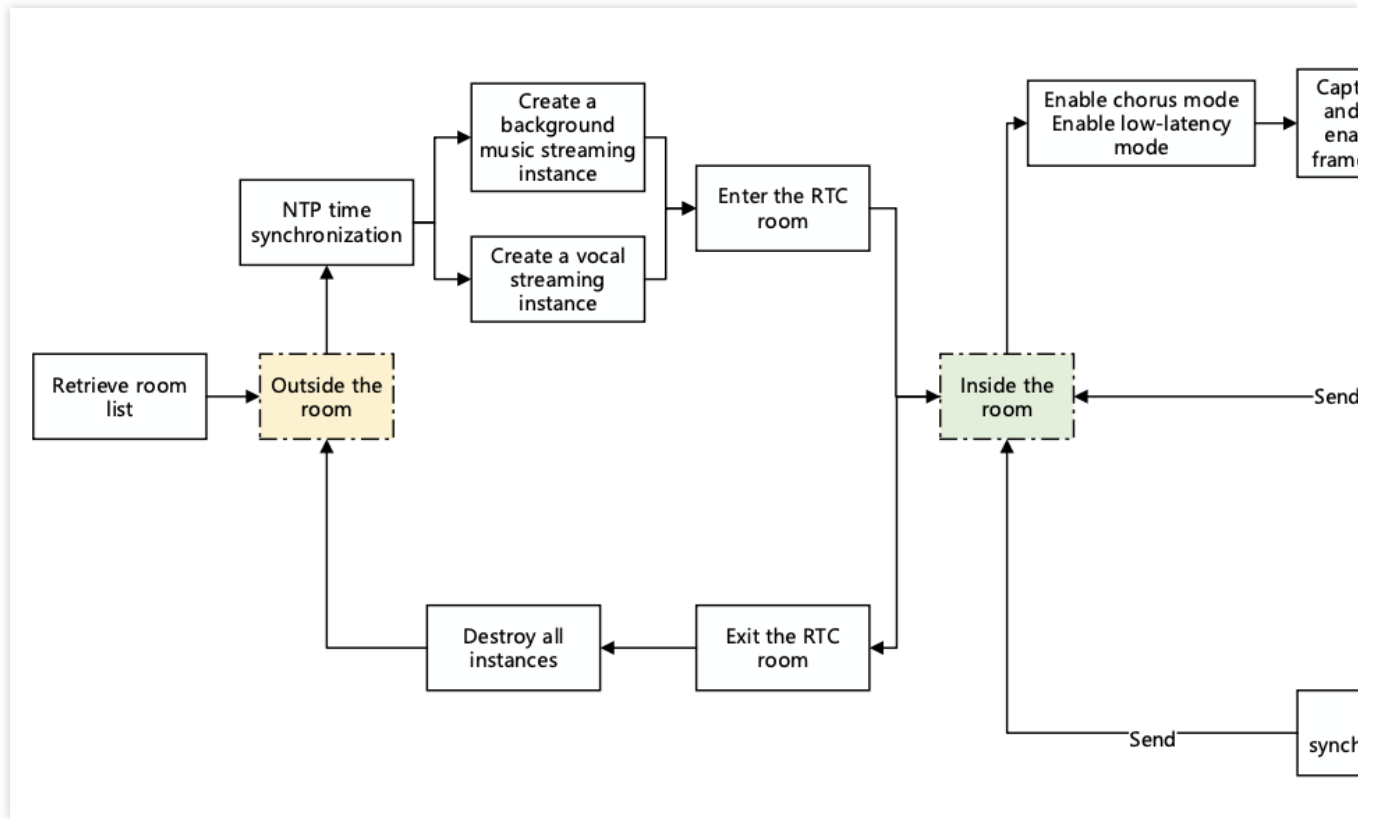
(2) Specific Implementation

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 Upstream music and vocals Song selection and initiating chorus Pushing back mixed stream 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 Upstream vocals Play music locally 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 Downstream mixed stream Receive SEI messages 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.

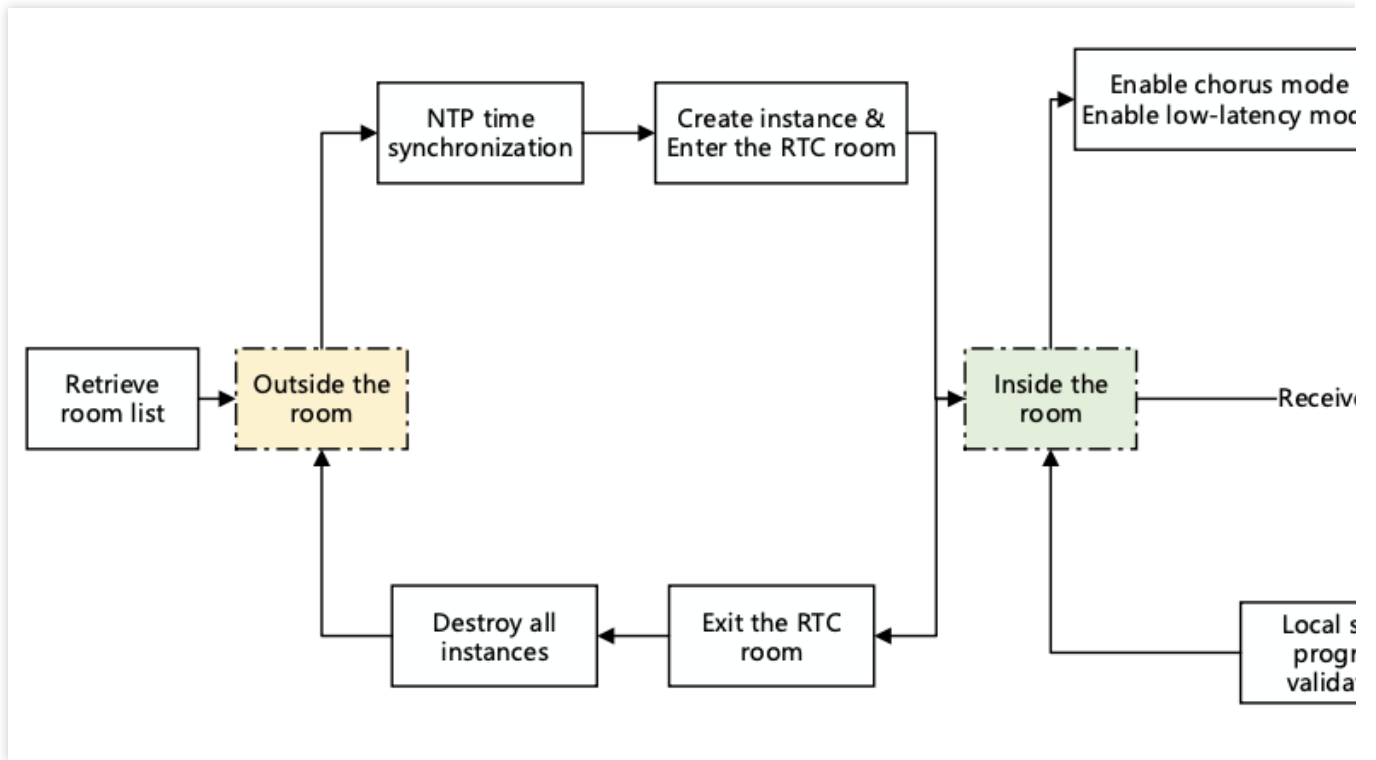
After pushing the stream, the lead singer is responsible for initiating the mixed stream push task.

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】



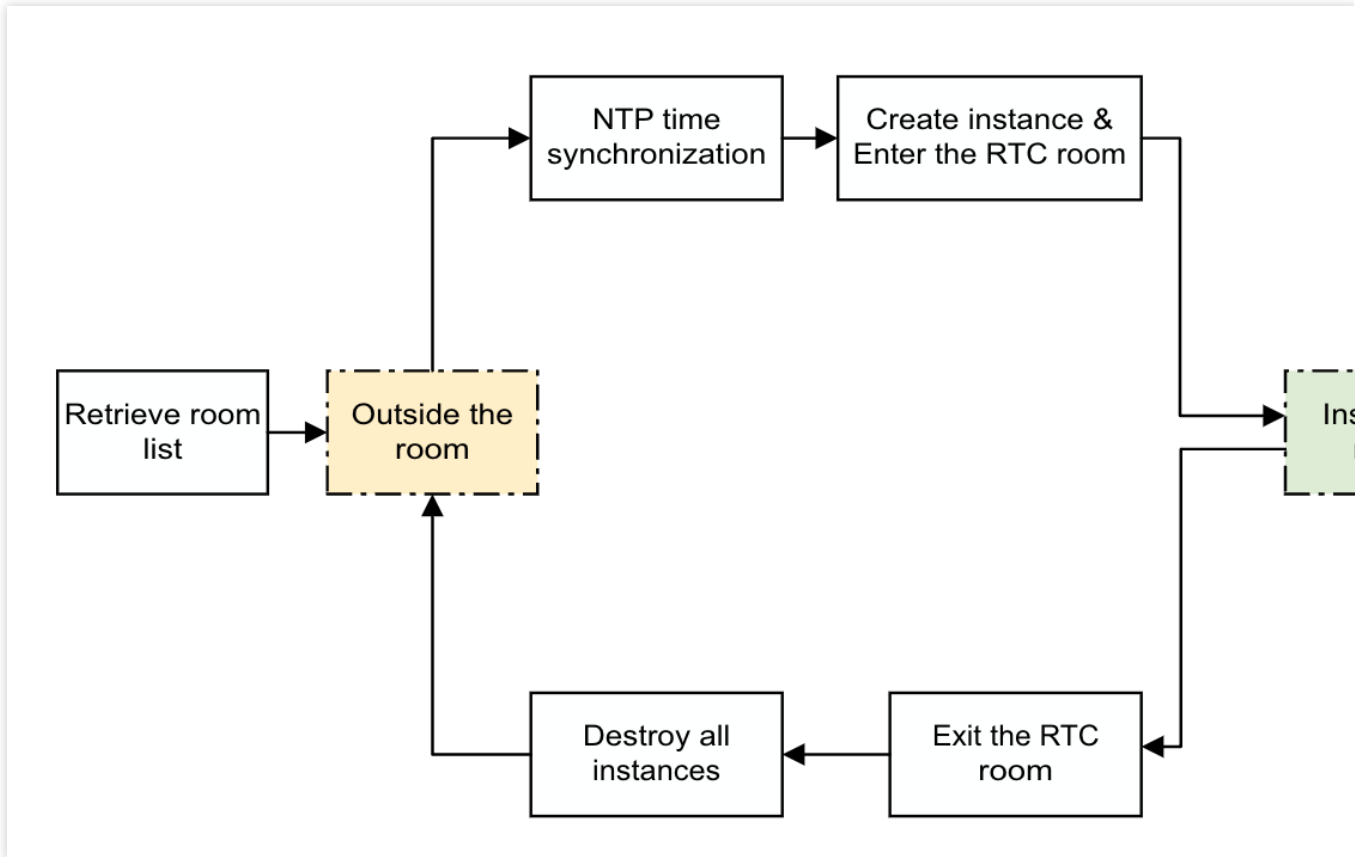
The chorus pushes one vocal stream and pulls the vocal stream of the user on the seat.

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.

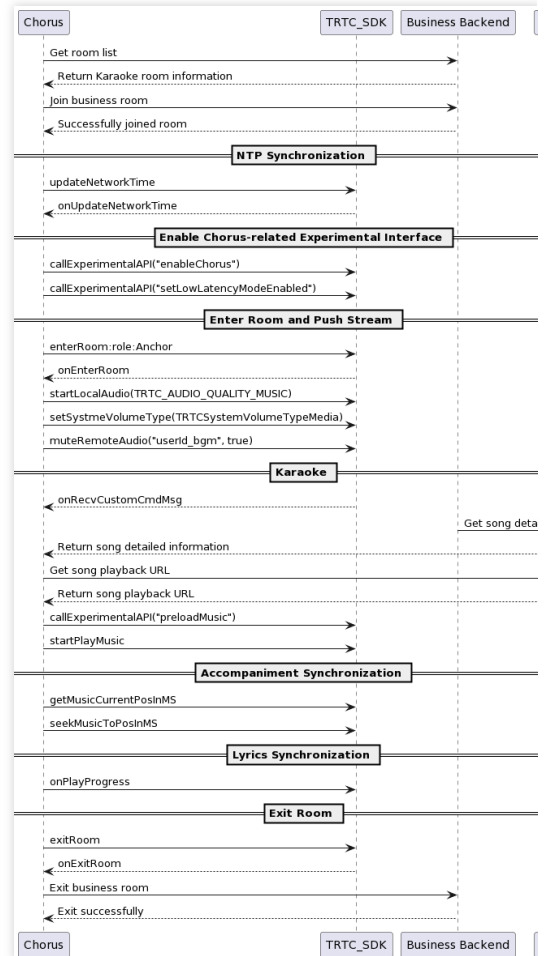
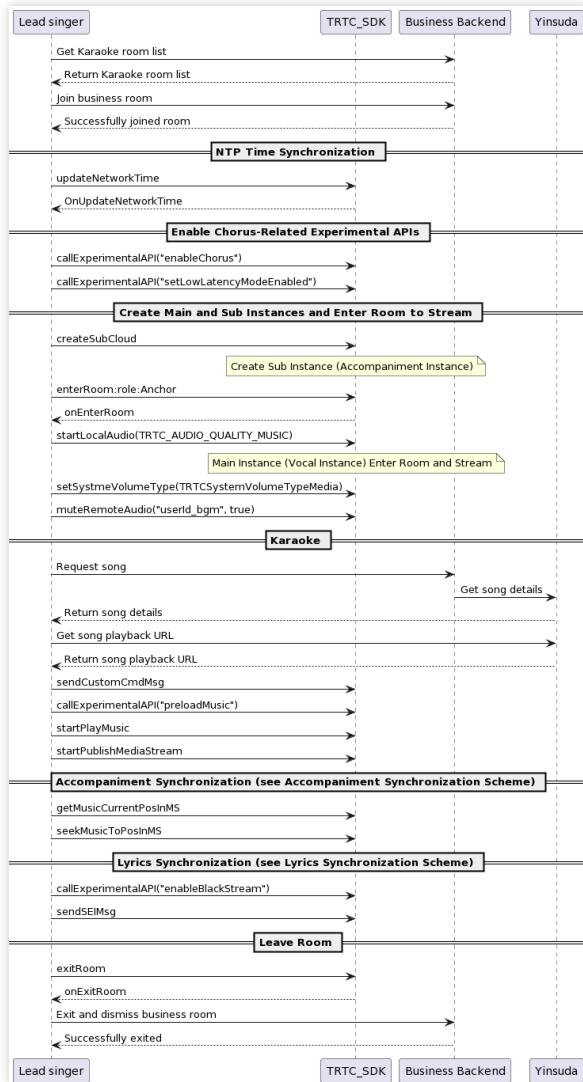
Parse the song progress information in the SEI of the mixed stream for lyric synchronization.

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

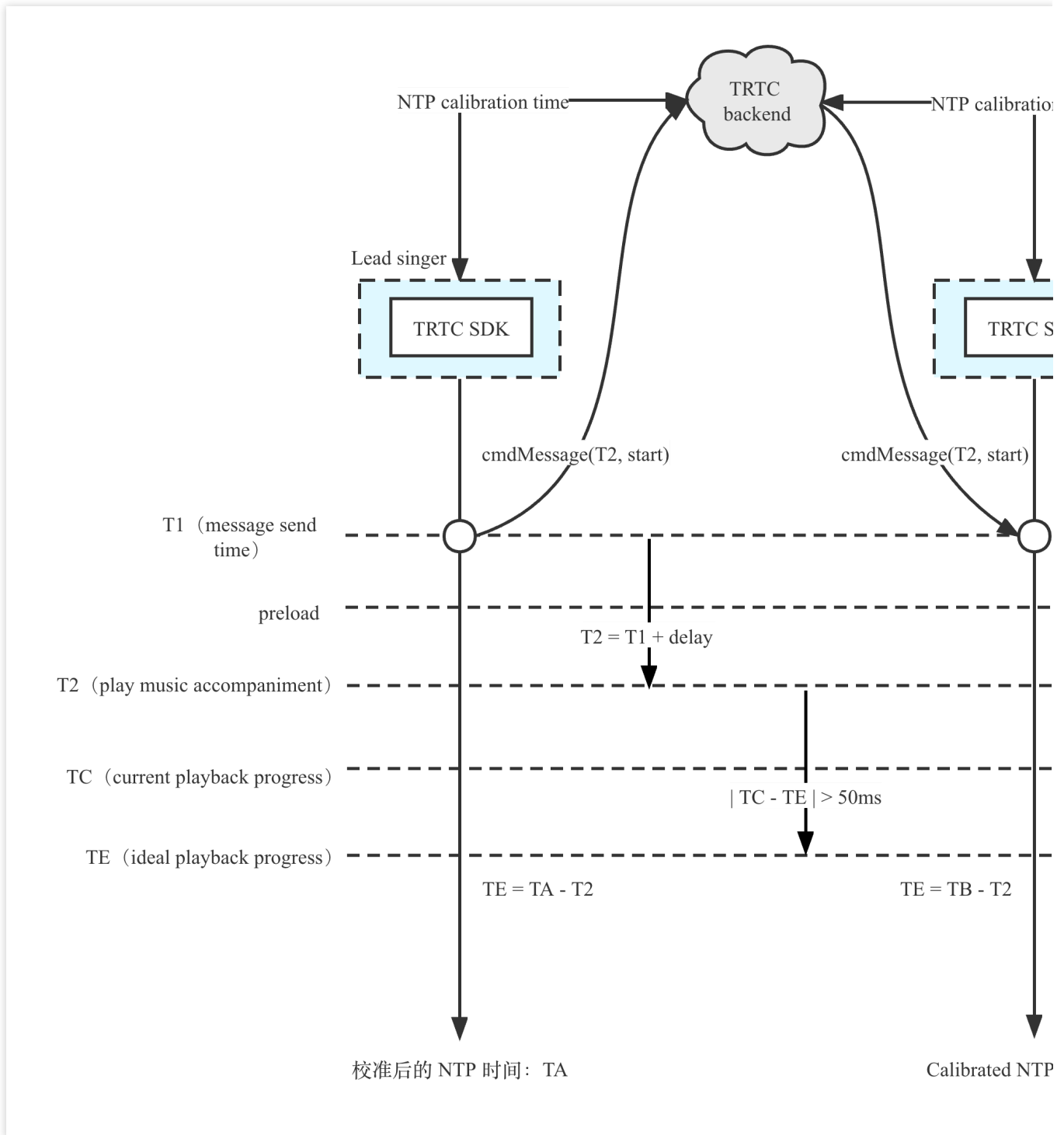
iOS

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:

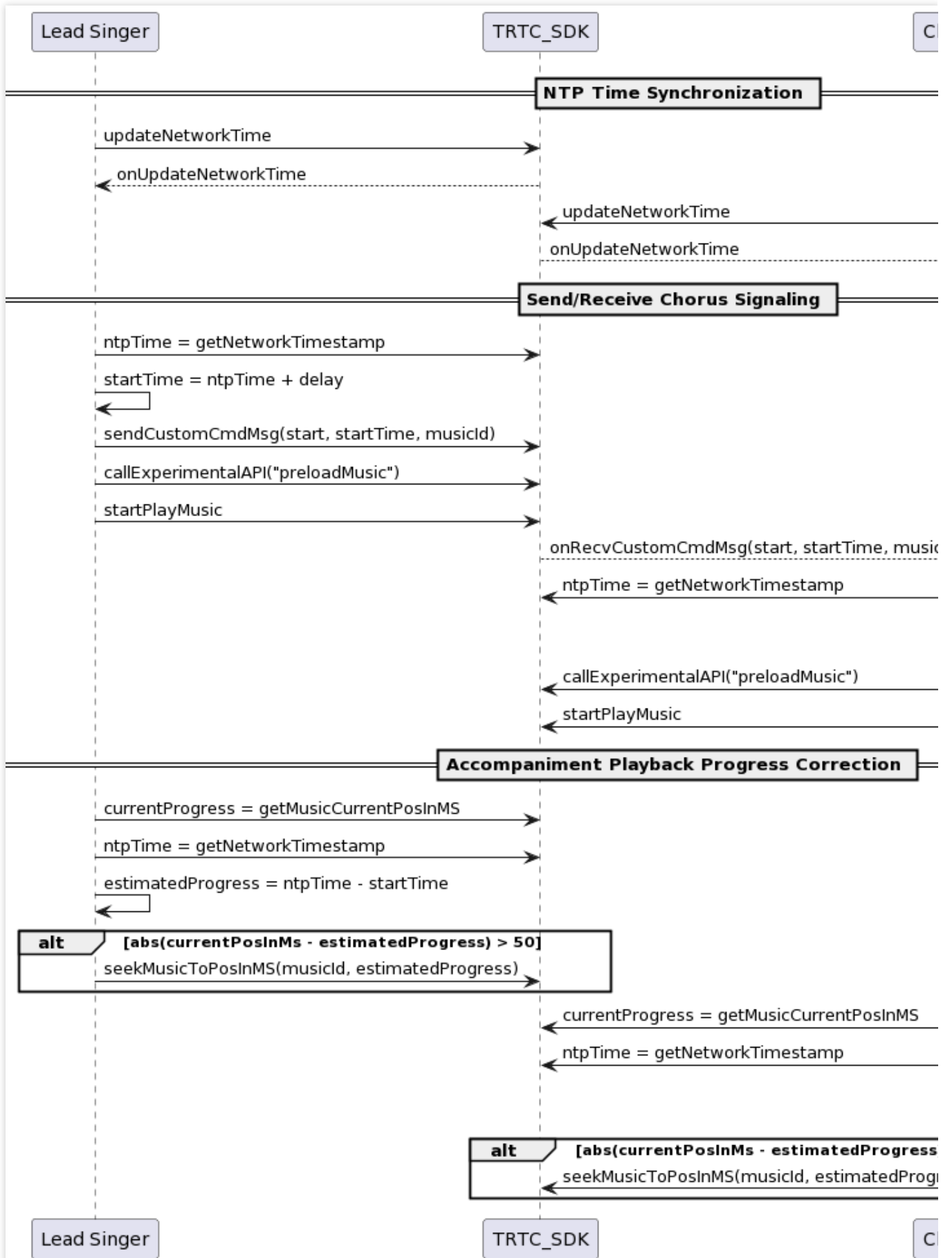
1. Each end performs NTP time calibration, updates and obtains the latest NTP time T to the TRTC cloud.
2. The main singer end sends a chorus signal (custom message) to agree on the start time T2 of the chorus.
3. Preload the song locally based on T2 and play it at a scheduled time.
4. Other chorus users 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.

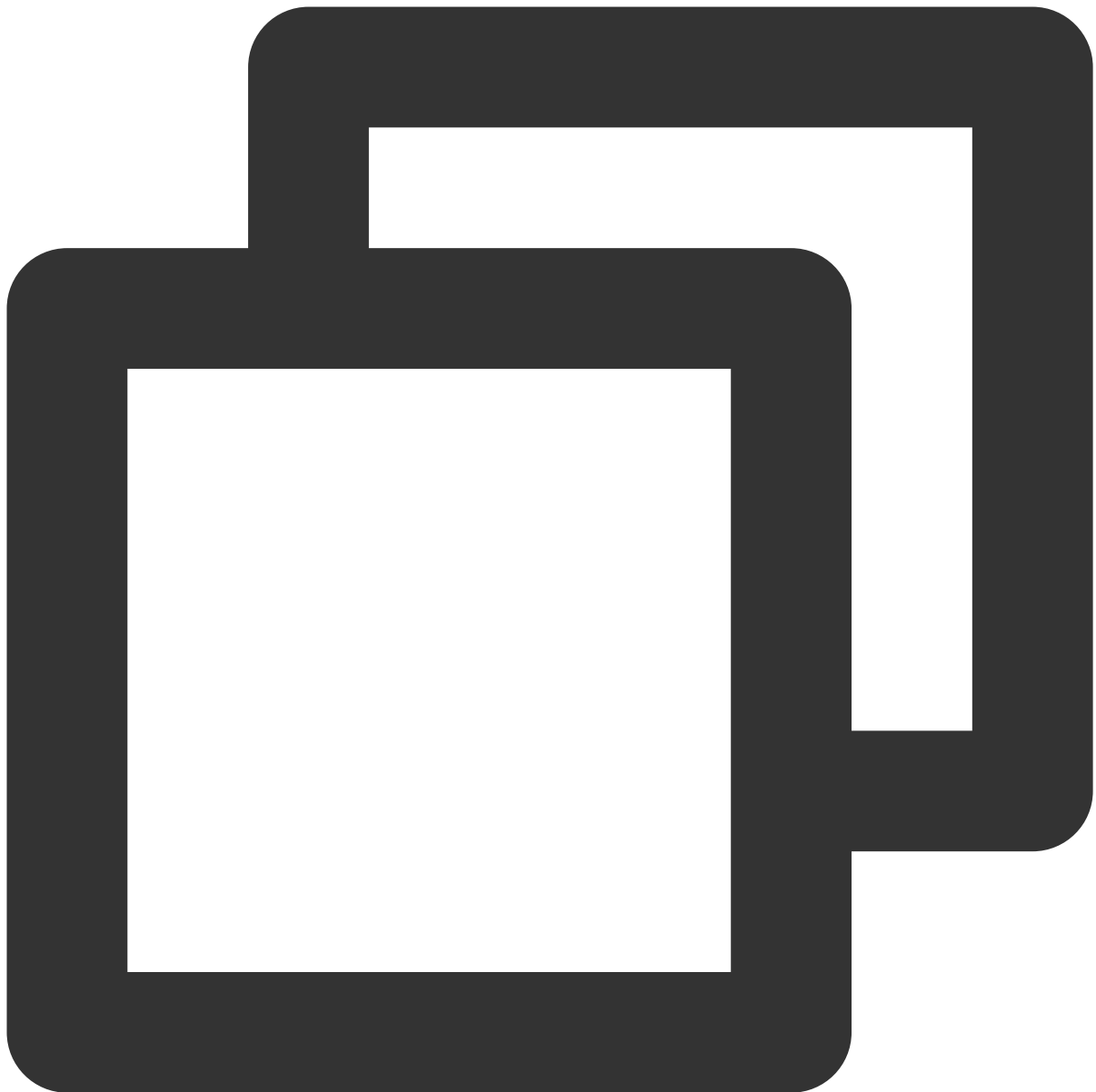
Timing diagram



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



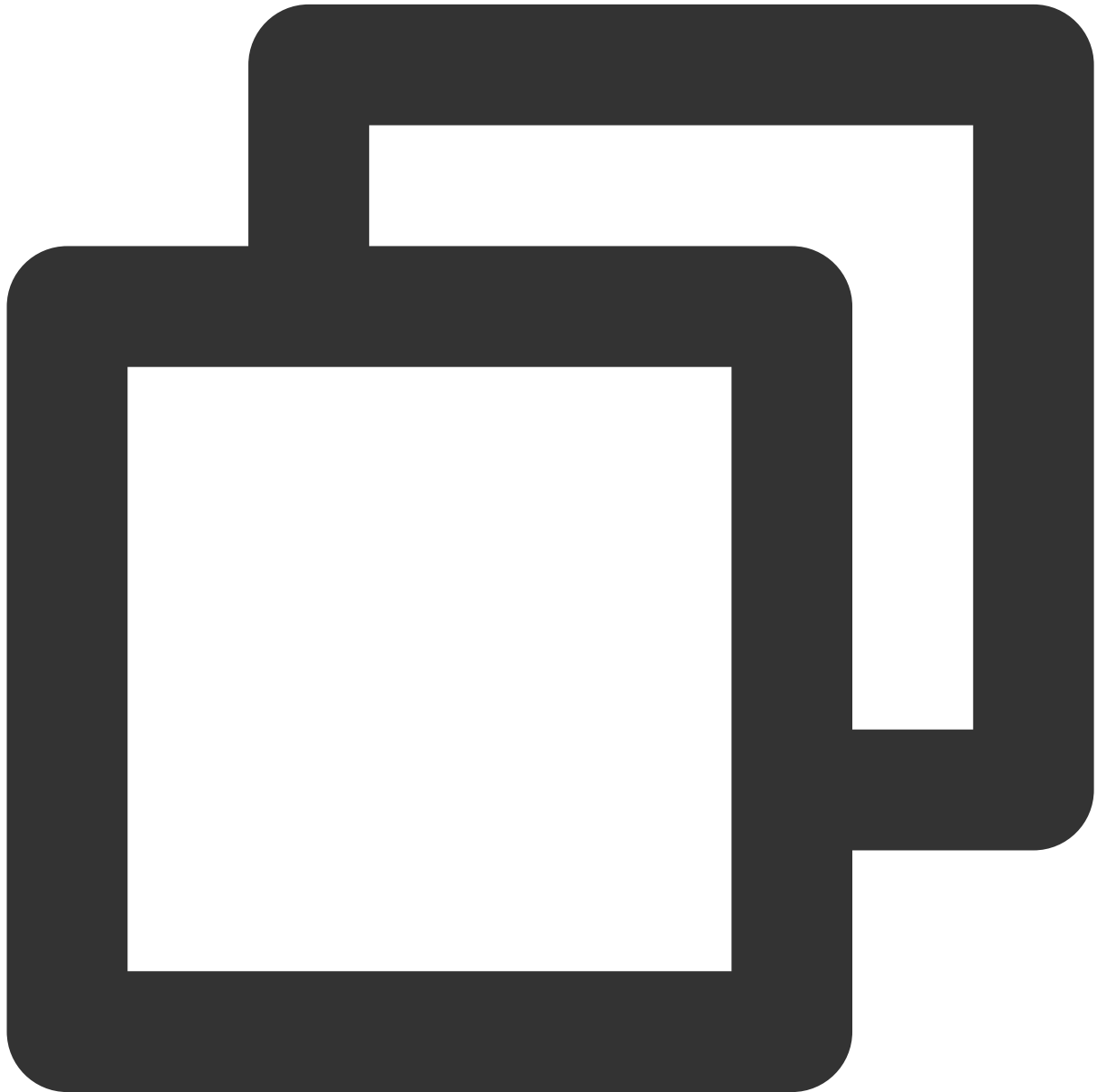
```
// Call the NTP time synchronization interface when entering the room.
[TXLiveBase updateNetworkTime];

// In the TXLiveBaseDelegate callback, determine whether the time synchronization i
- (void)onUpdateNetworkTime:(int)errCode message:(NSString *)errMsg {
    // errCode 0: 0: Time synchronization is successful and the deviation is within
    // 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",
```

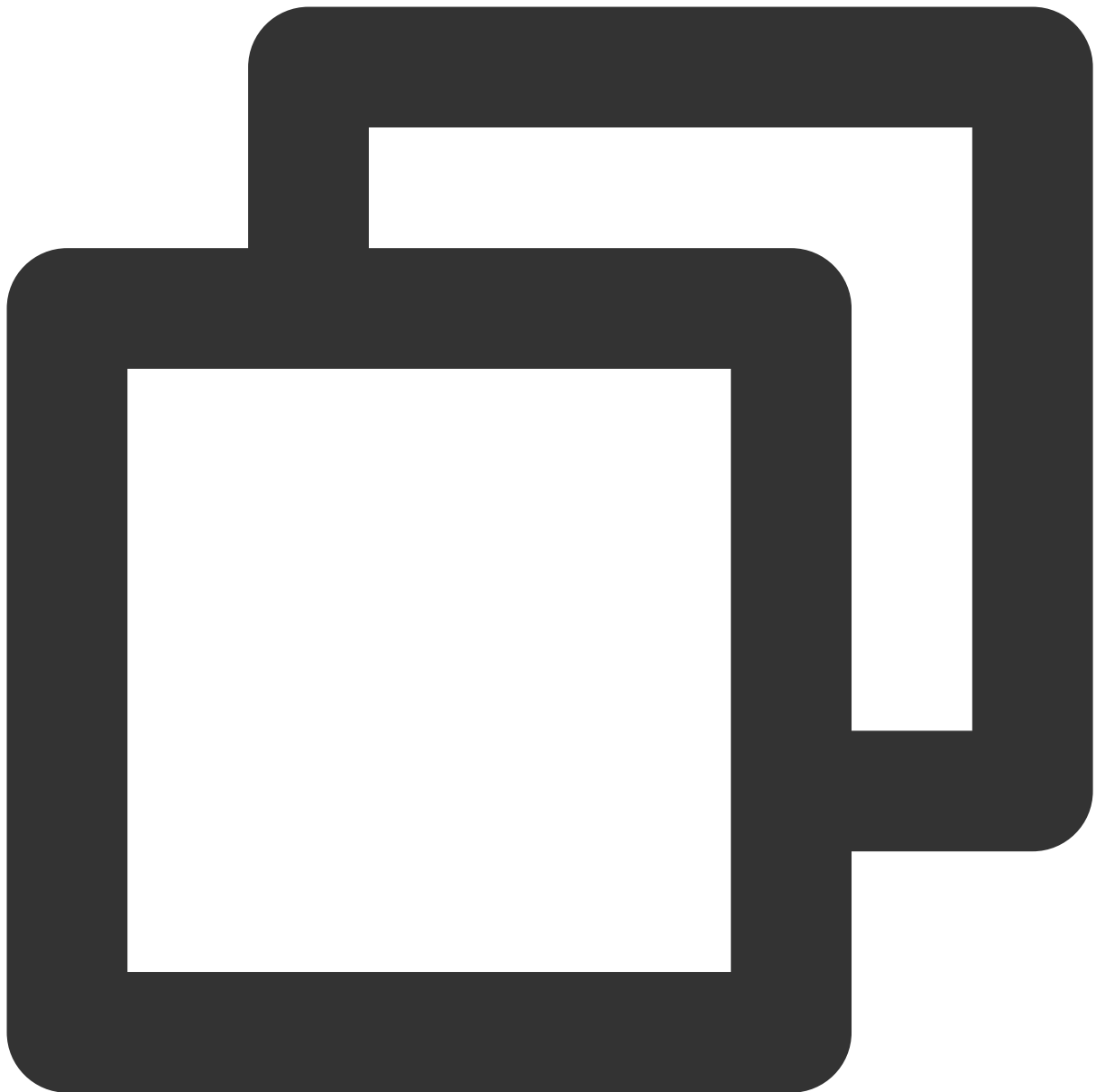
```
        // Scheduled time for a tutti.
        @"startPlayMusicTS": @(startTs),
        @"musicId": @"musicId",
        @"musicDuration": @(musicDuration),
    };
    NSString *jsonString = [self jsonStringFrom:json];
    [trtcCloud sendCustomMessage:jsonString reliable:NO];
```

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:(UIIn  
  
    NSString *msg = [[NSString alloc] initWithData:message encoding:NSUTF8StringEnc  
    NSError *error;  
    NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[msg dataUsingEncoding:  
                                                                    options:NSUTF8StringEncoding  
                                                                    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];

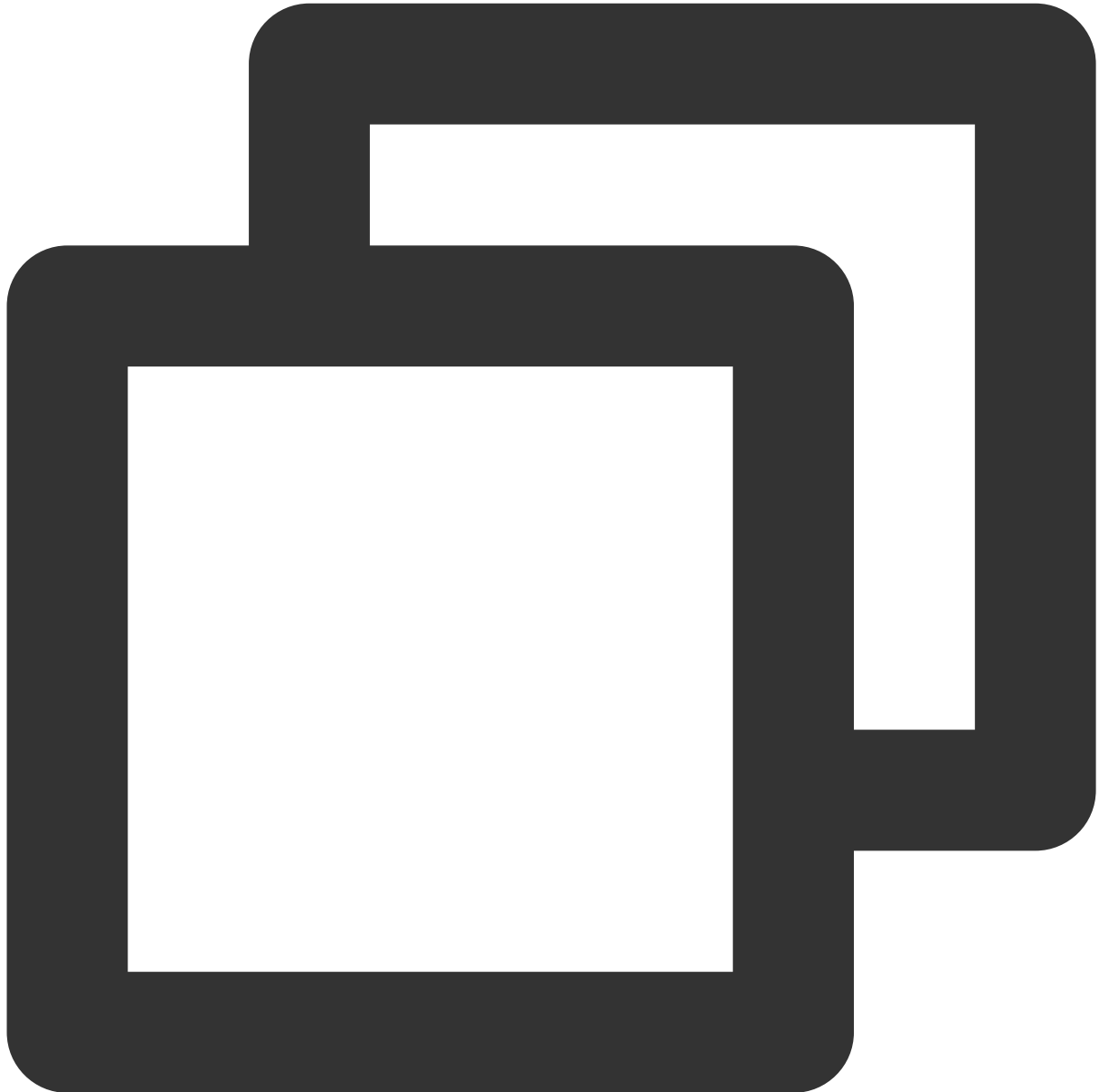
    // play music
    TXAudioMusicParam *param = [[TXAudioMusicParam alloc] init];
    param.ID = musicId;
    param.path = url;
    param.loopCount = 0;
    param.publish = NO;

    [[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  
}
```

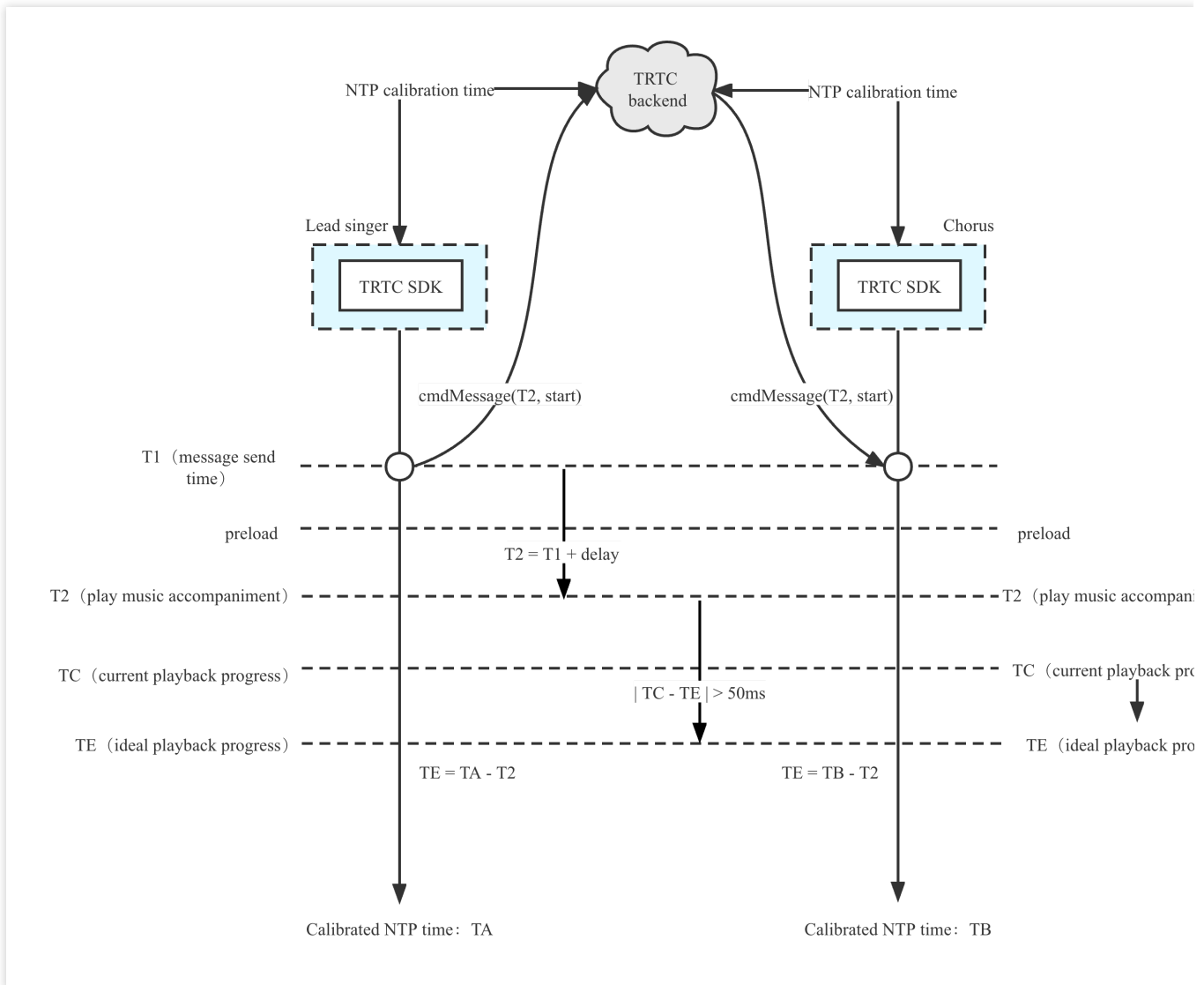
Android

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.



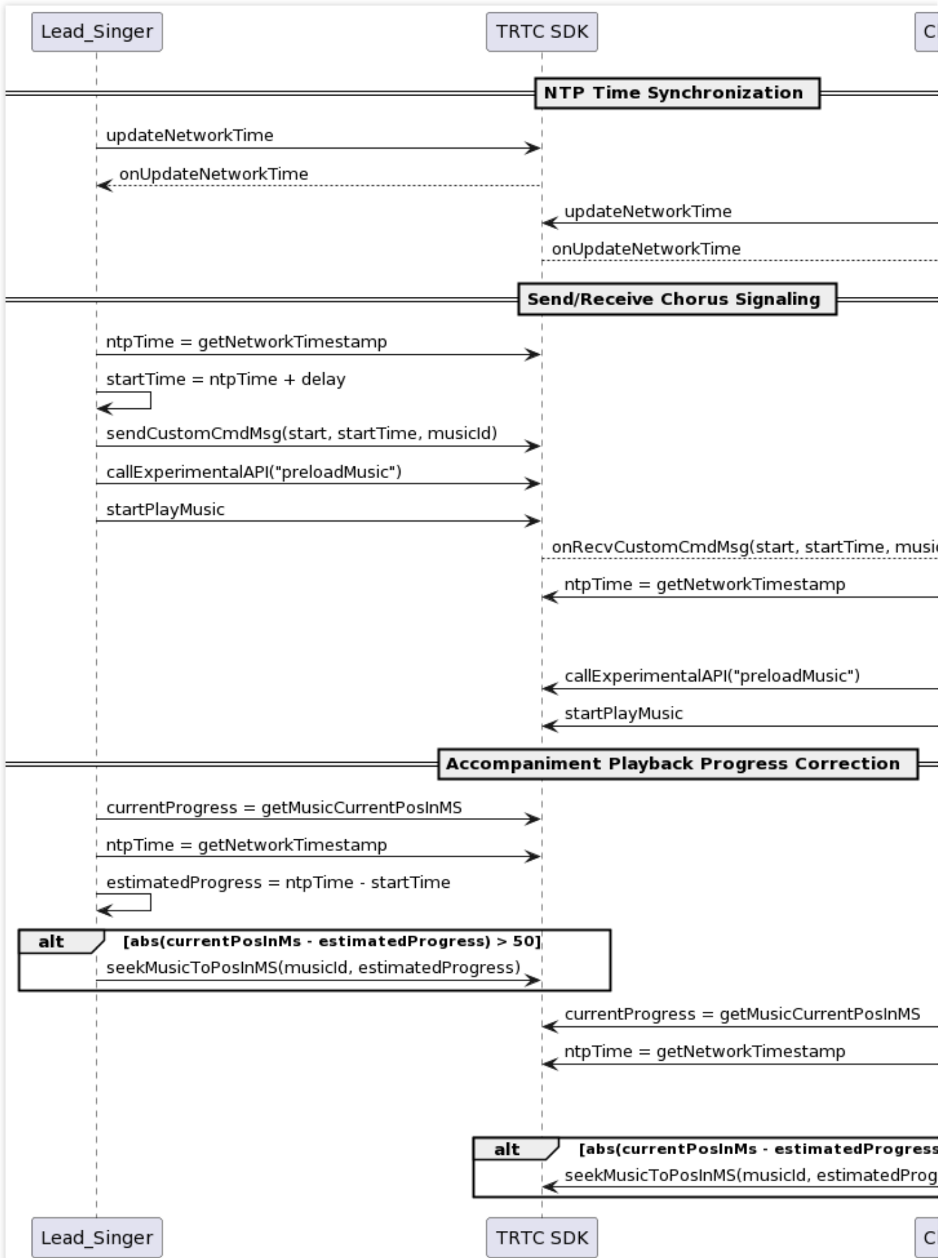
The specific process is as follows:

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.

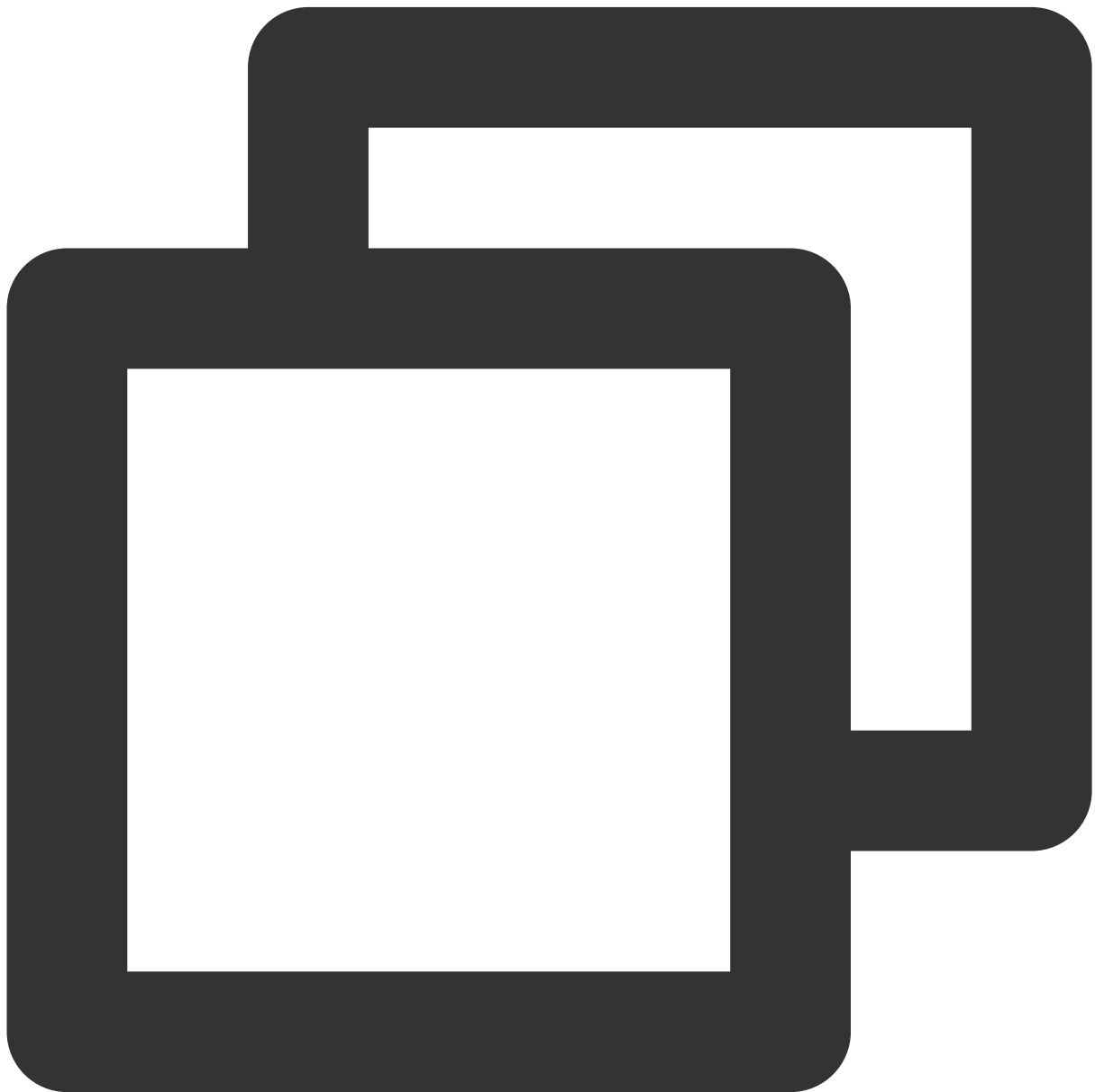
Timing diagram



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

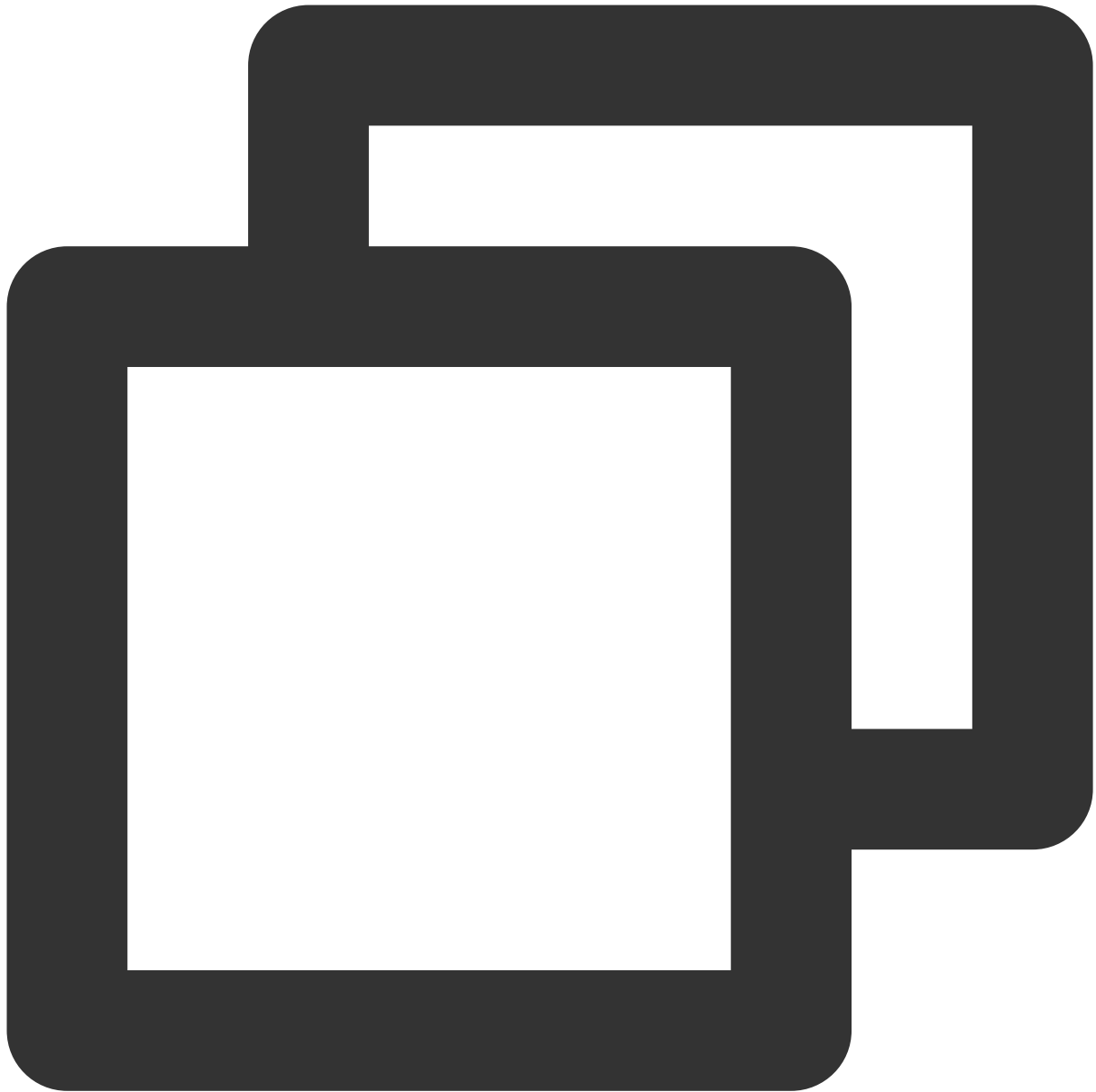
1. NTP calibration service



```
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



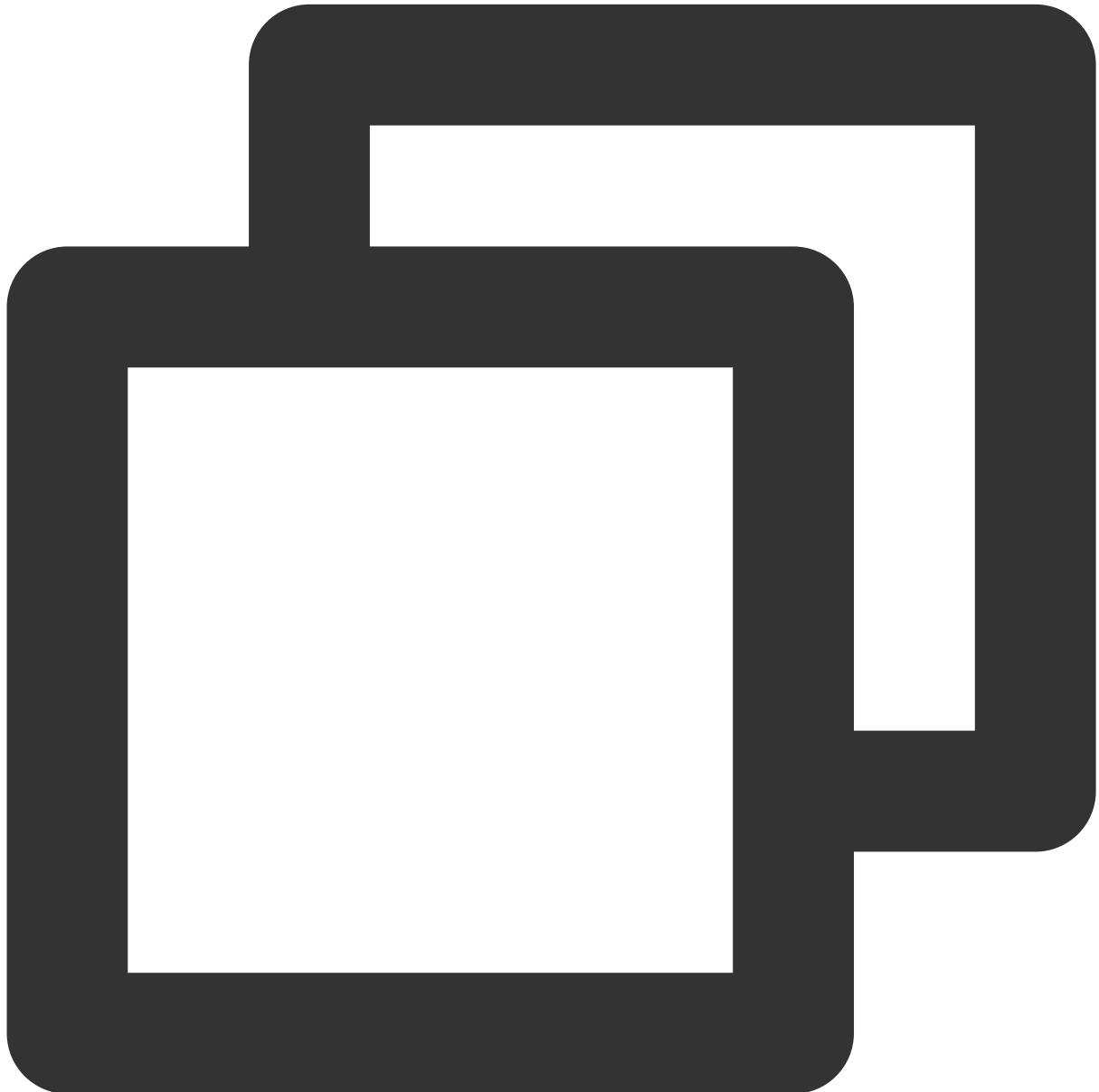
```
JSONObject jsonObject = new JSONObject();
jsonObject.put("cmd", "startChorus");
// Agree on a time for the chorus.
jsonObject.put("startPlayMusicTS", startTs);
jsonObject.put("musicId", "musicId");
String body = jsonObject.toString();
mTRTCCloud.sendCustomCmdMsg(0, body.getBytes(), false, false);
```

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



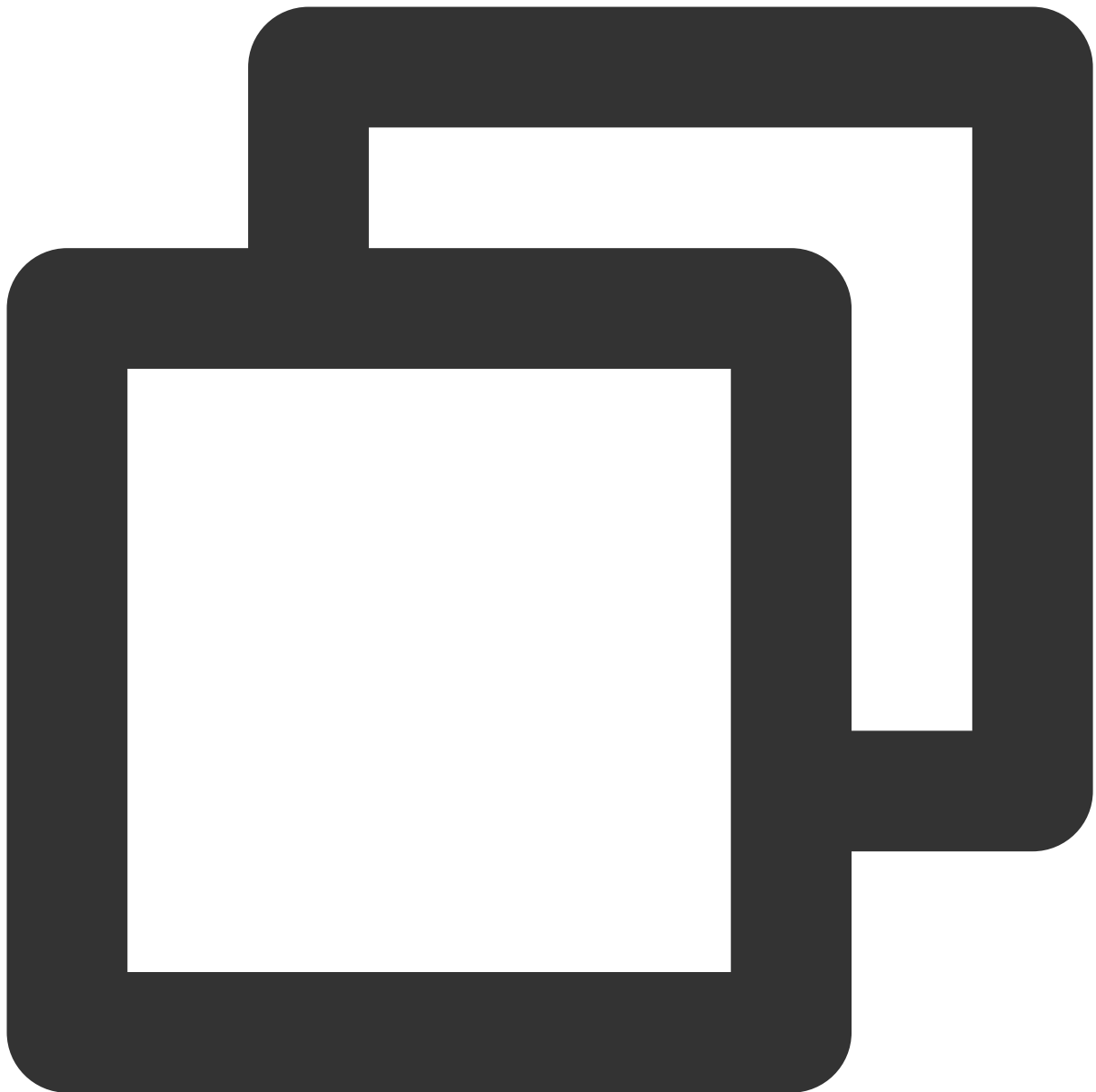
```
public void onRecvCustomCmdMsg(String userId, int cmdID, int seq, byte[] message) {  
    JSONObject json = new JSONObject(new String(message, "UTF-8"));  
    String cmd = json.getString("cmd");  
}
```

```
// 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(musi
// 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) estimatedPr
}
```


Lyric Synchronization

iOS

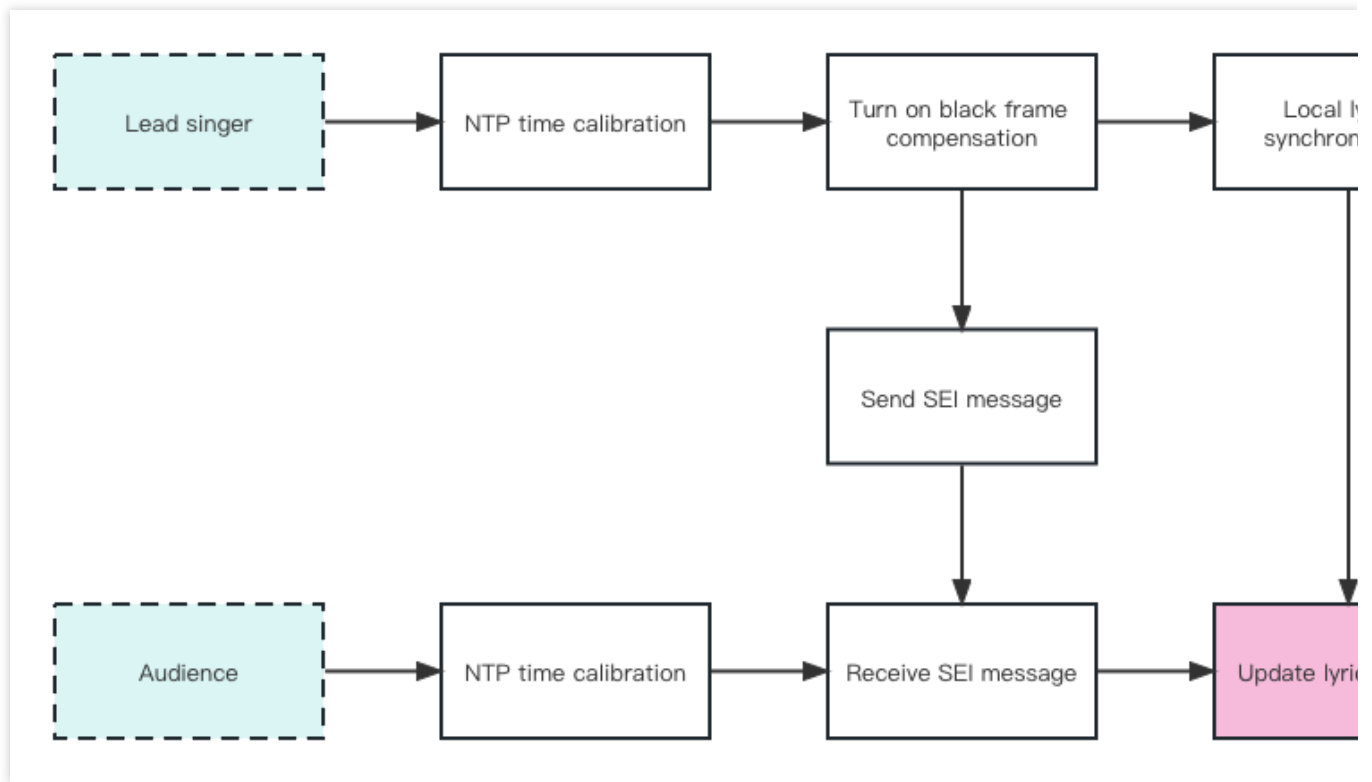
Last updated : 2023-09-27 15:11:25

1.1 Implementation process

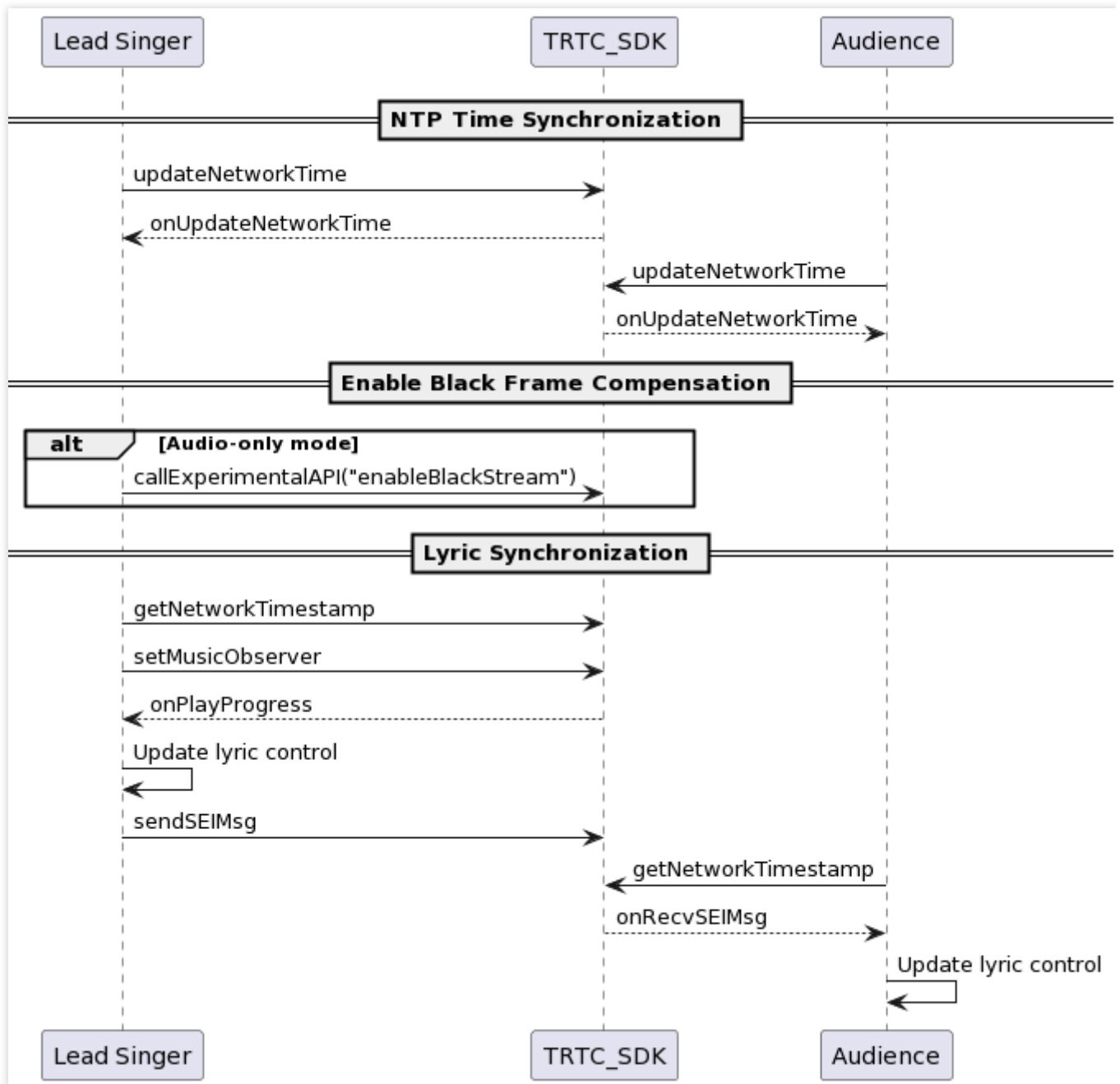
In the lyrics synchronization solution, the actions of the three different roles are as follows:

Main Singer	Chorus	Audience
NTP time calibration Enable black frame insertion Send SEI messages Local lyrics synchronization Update lyrics control	NTP time calibration Local lyrics synchronization Update lyrics control	NTP time calibration Receive SEI messages 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.



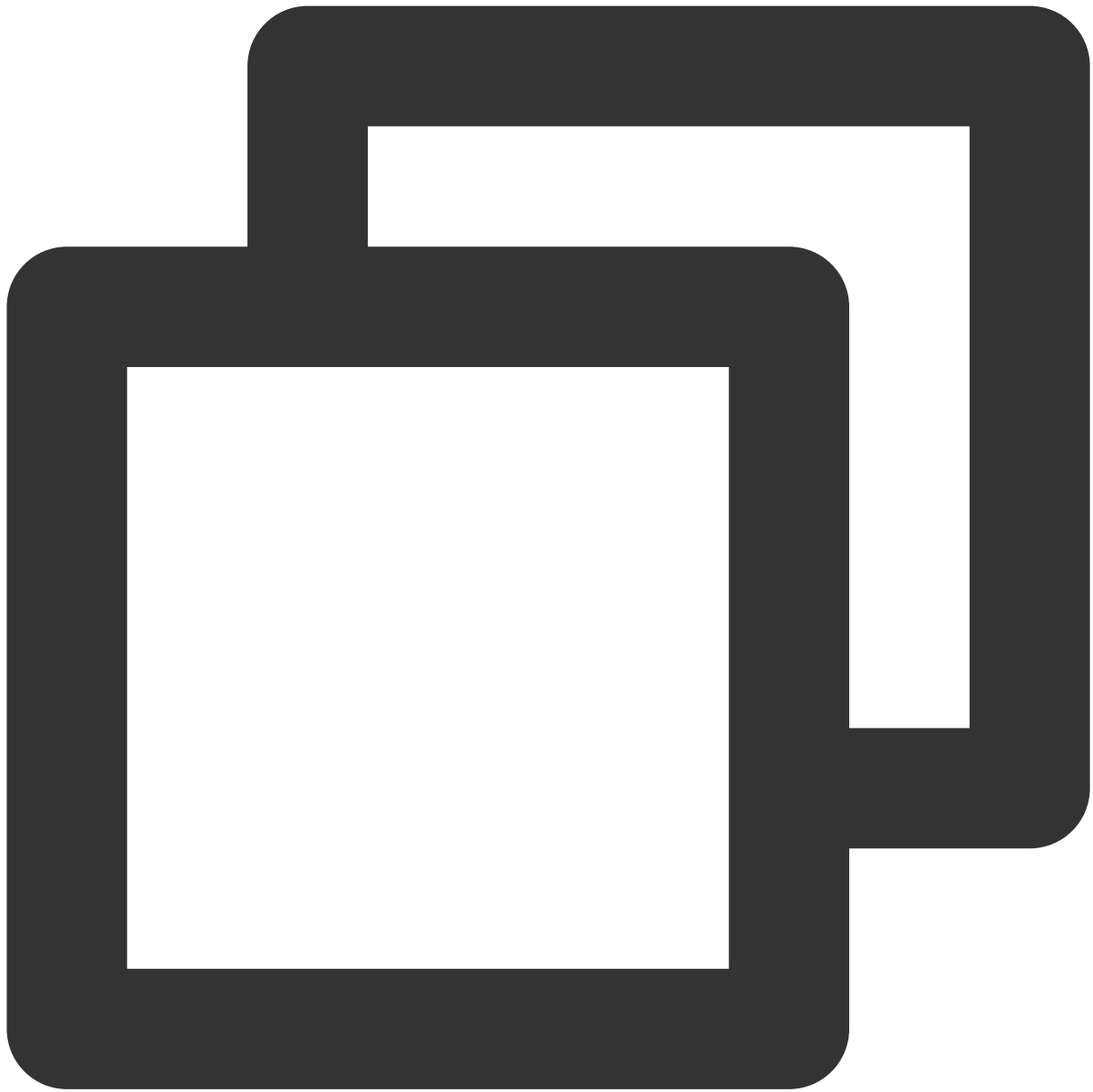
Timing diagram



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:

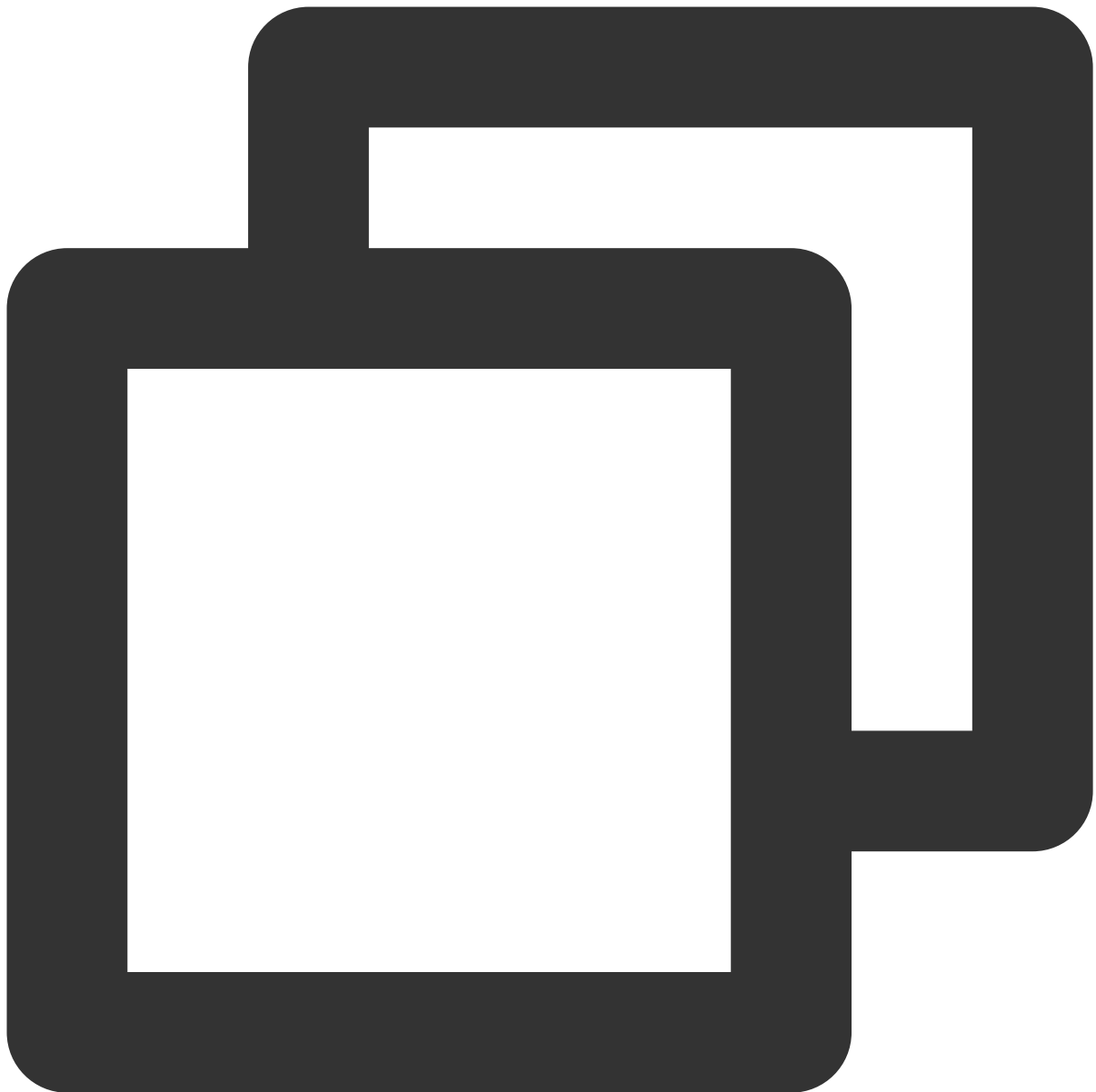
The experimental interface `enableBlackStream` needs to be called after entering the room;

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:
```

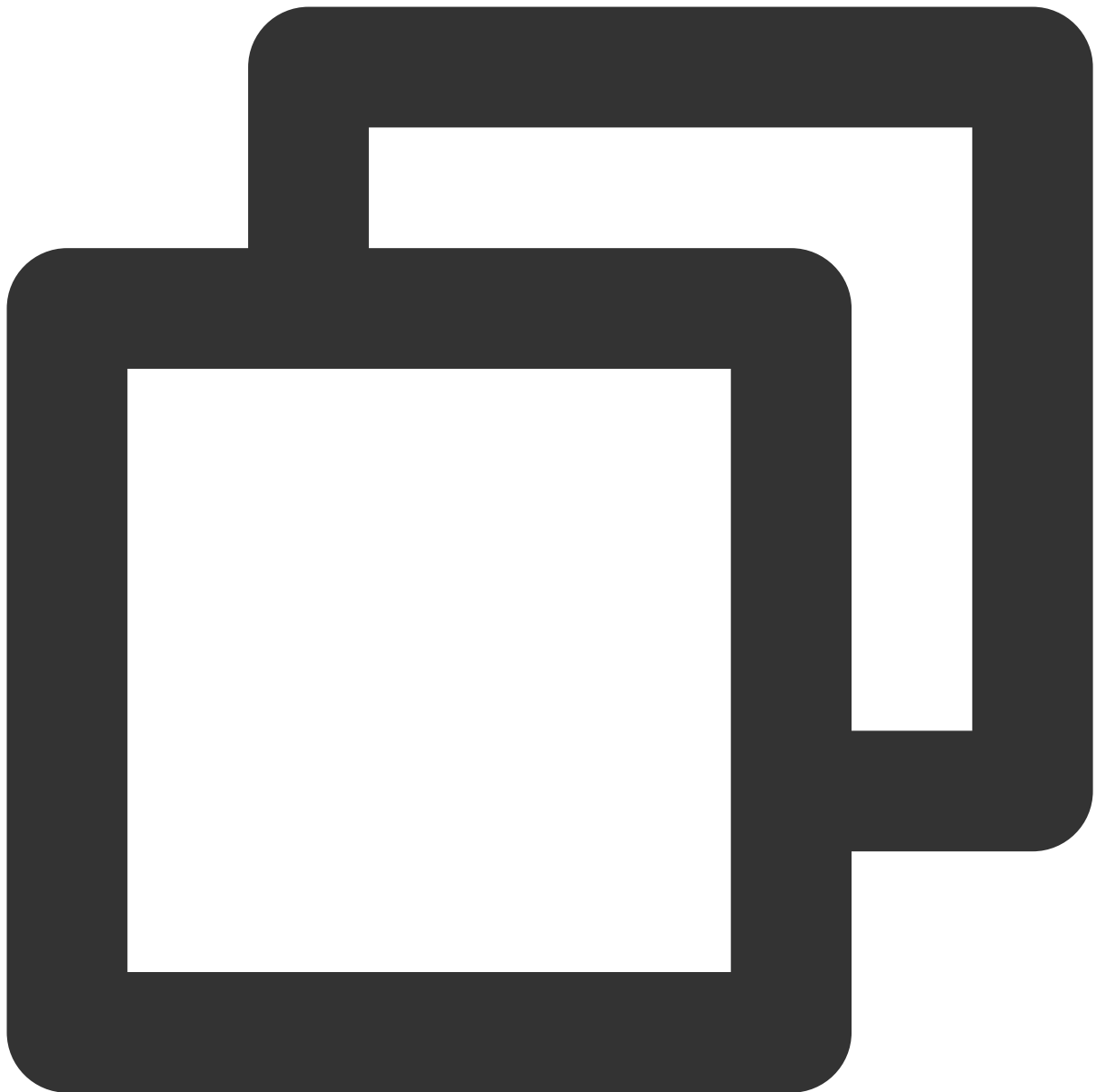
```
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8S  
[trtcCloud sendSEIMsg:[jsonString dataUsingEncoding:NSUTF8StringEncoding] repea  
};
```

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

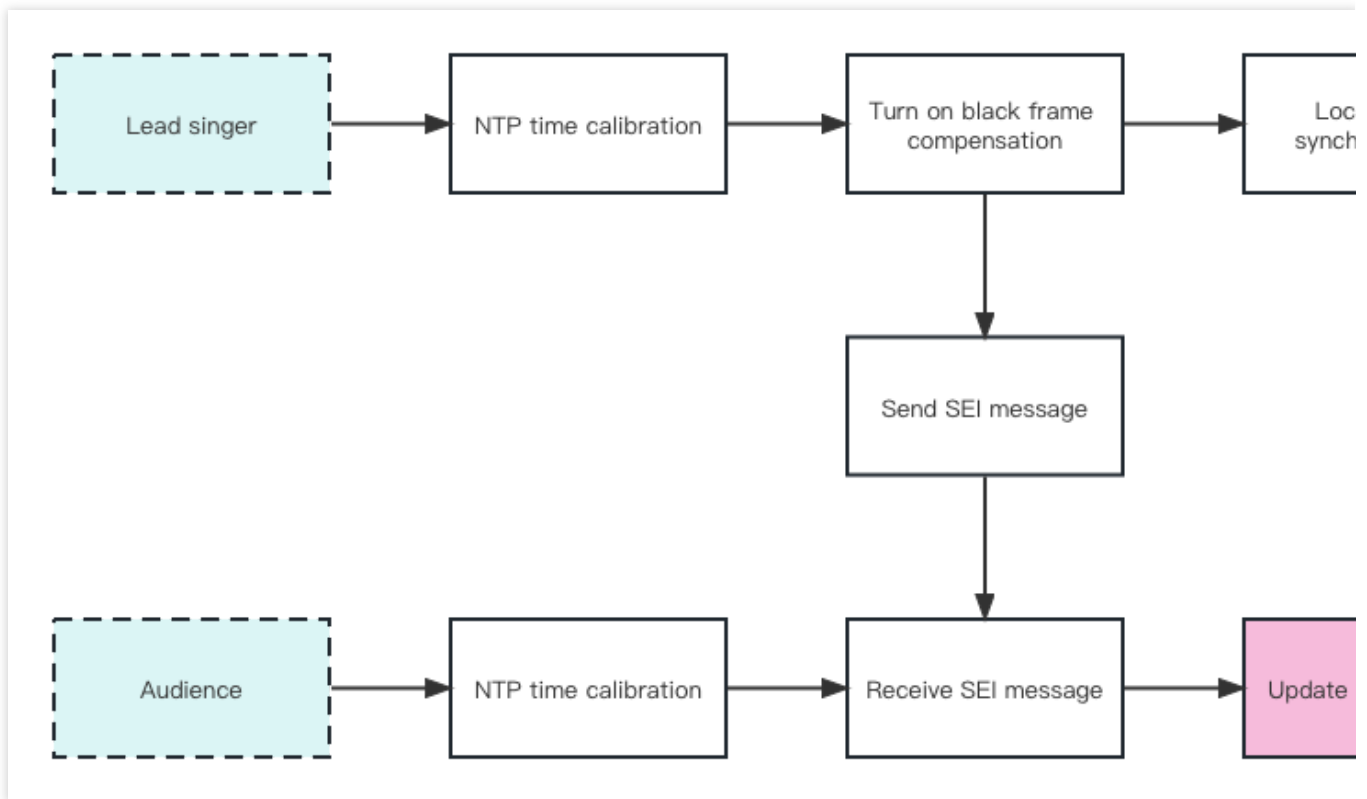
Last updated : 2023-12-28 21:32:43

Implementation process

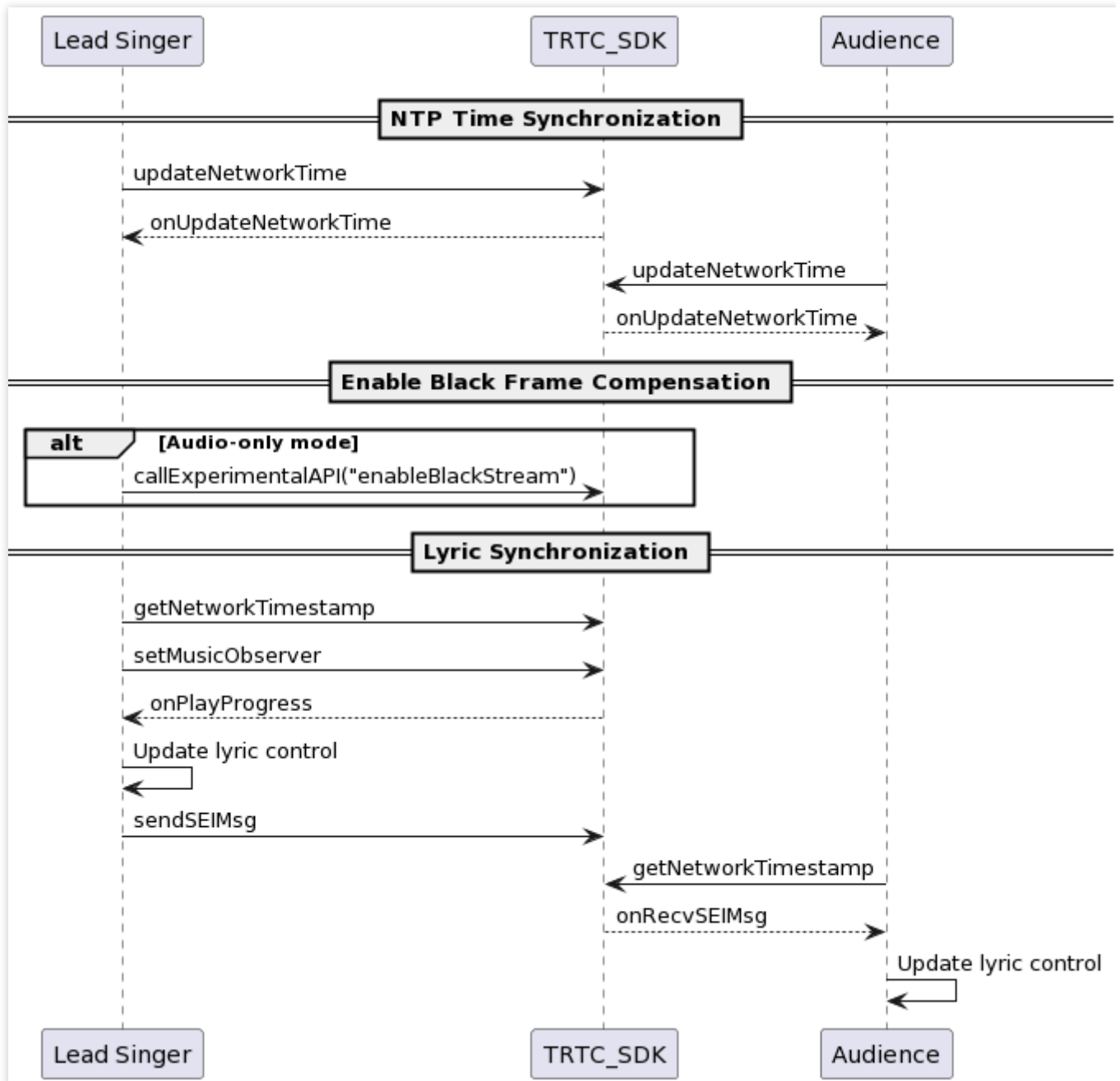
In the lyrics synchronization scheme, the actions of three different roles are as follows:

Lead Singer	Chorus	Audience
NTP time calibration Turn on black frame compensation Send SEI message Local lyrics synchronization Update lyrics widget	NTP time calibration Local lyrics synchronization Update lyrics widget	NTP time calibration Receive SEI message 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.



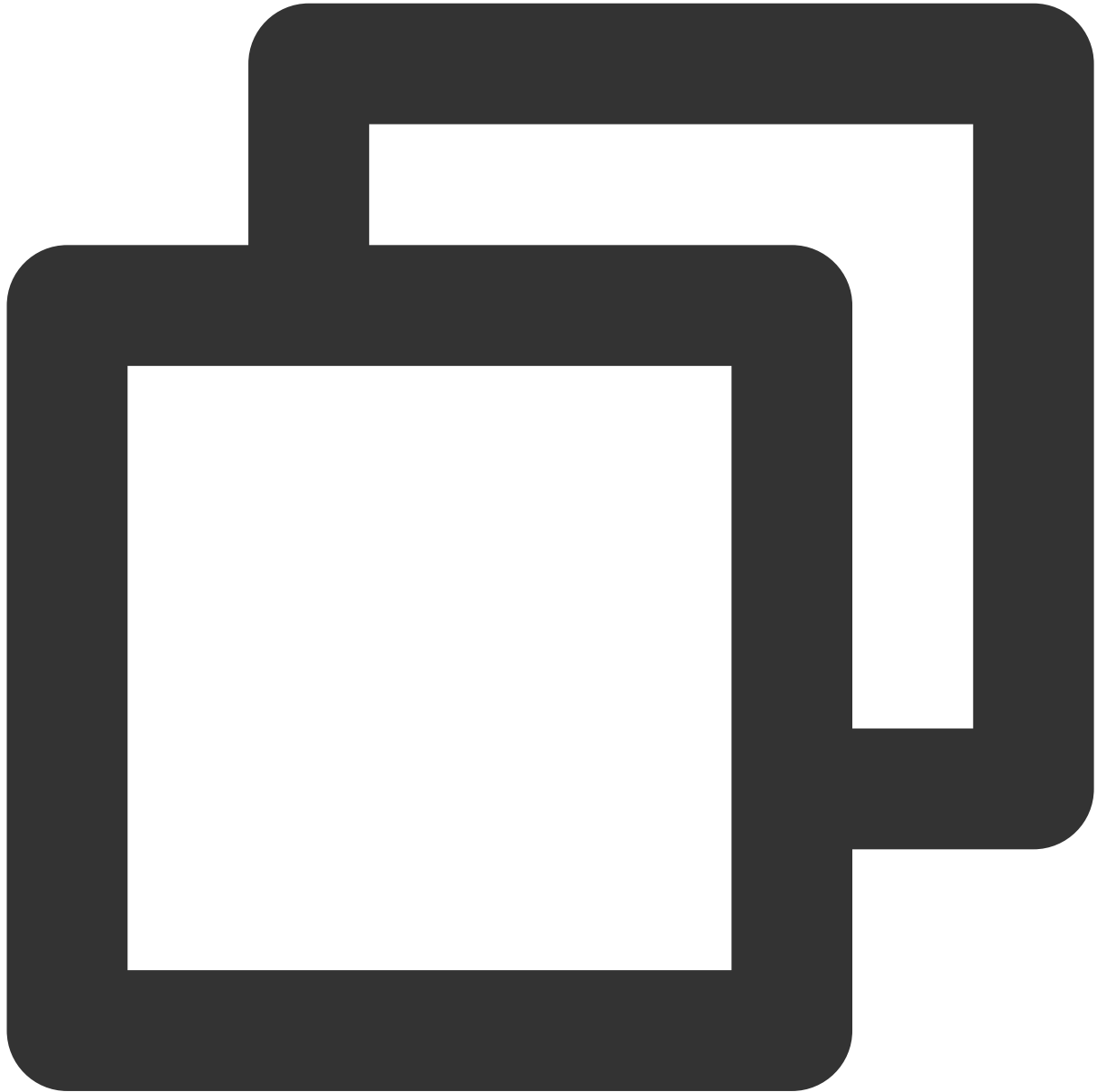
Timing diagram



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



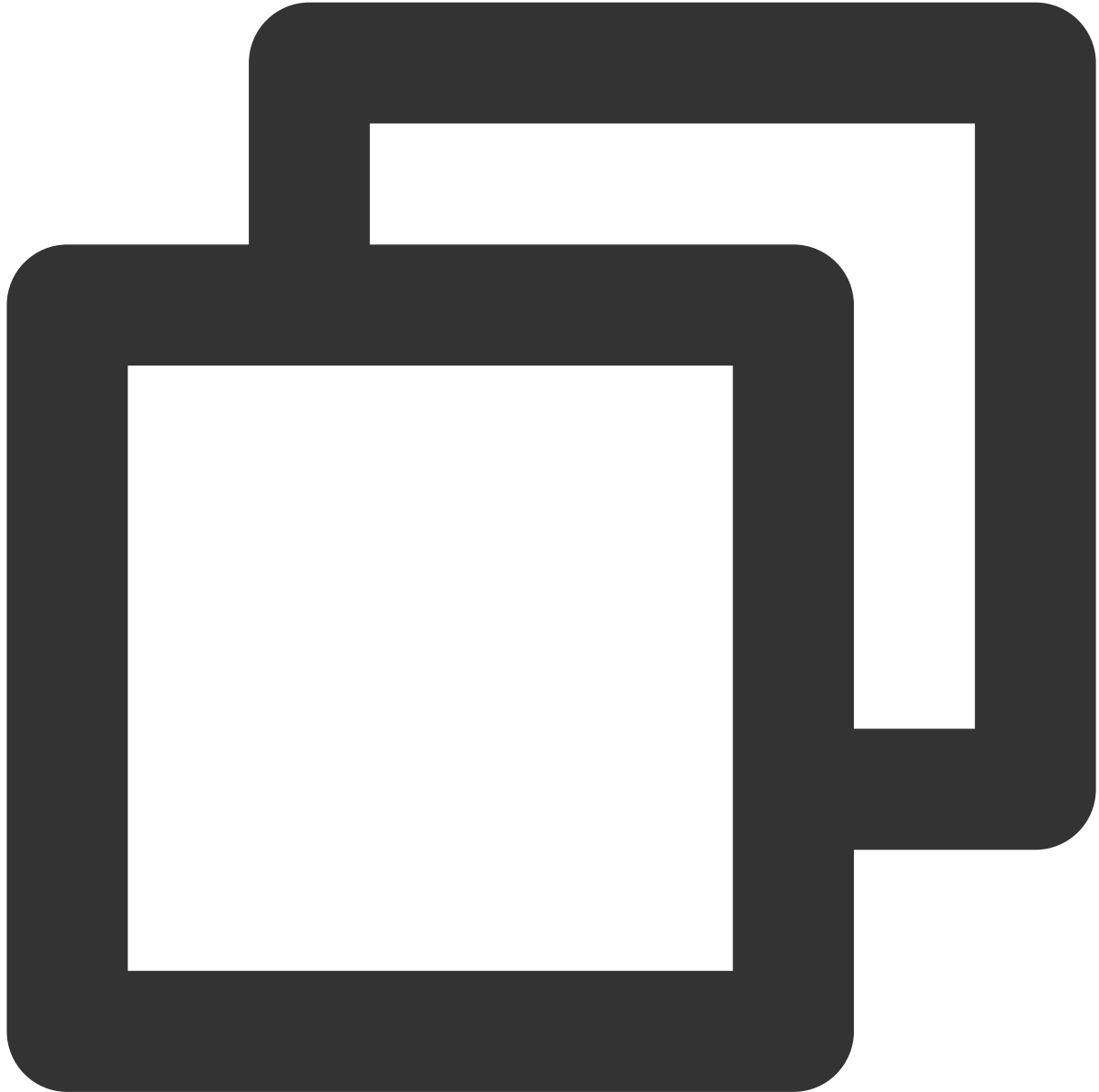
```
// In pure audio mode, the main instance (vocal instance) needs to turn on black fr  
mTRTCCloud.callExperimentalAPI("{\"api\":\"enableBlackStream\",\"params\": {\
```

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);
```

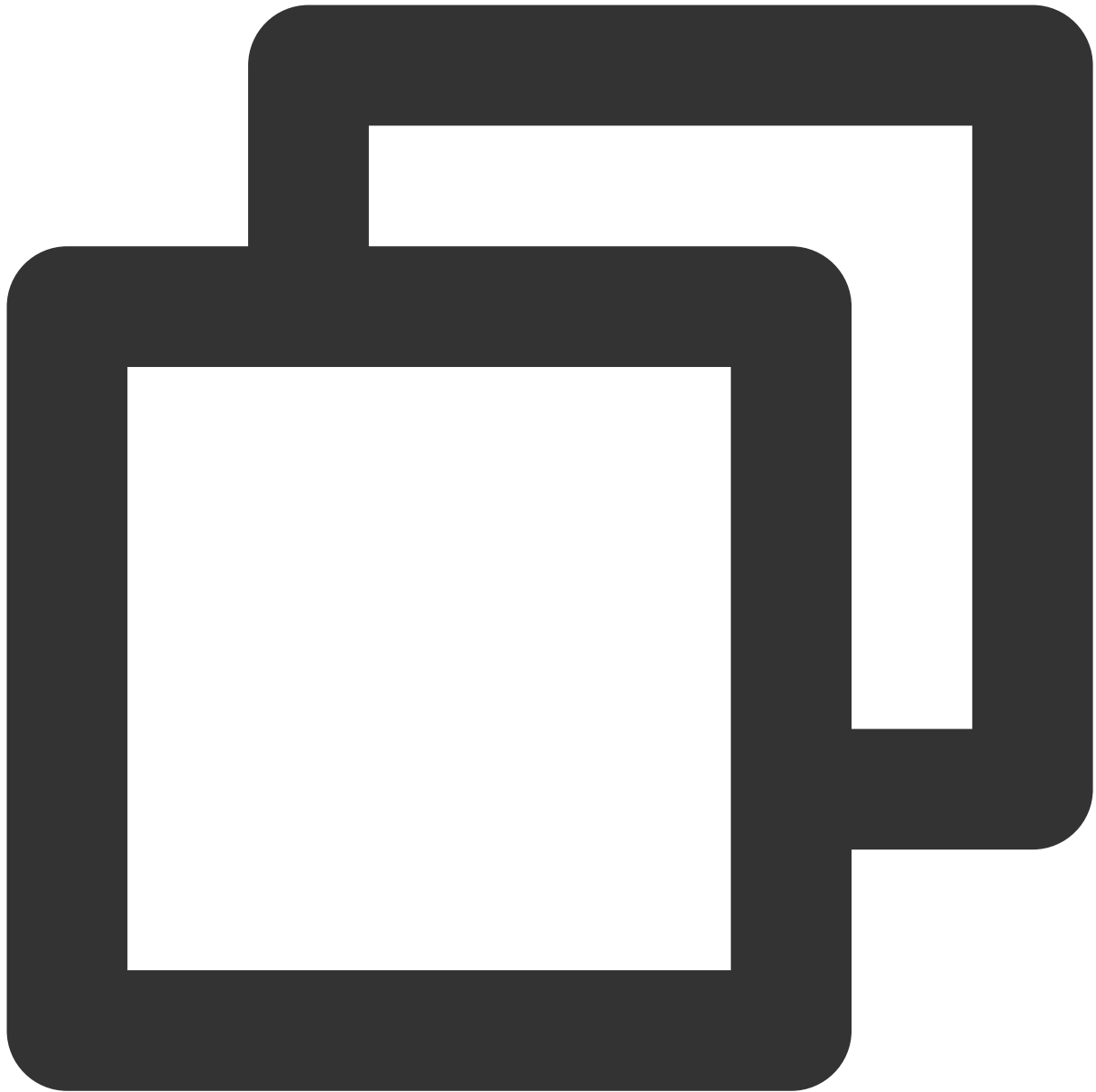
```
        jsonObject.put("ntpTime", ntpTime);
        jsonObject.put("musicId", musicId);
        jsonObject.put("duration", duration);
        jsonObject.toString().getBytes();
        mTRTCCloud.sendSEIMsg(jsonObject.toString().getBytes(), 1);
    }
}
```

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 synchron
    ...
}
```

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];
```

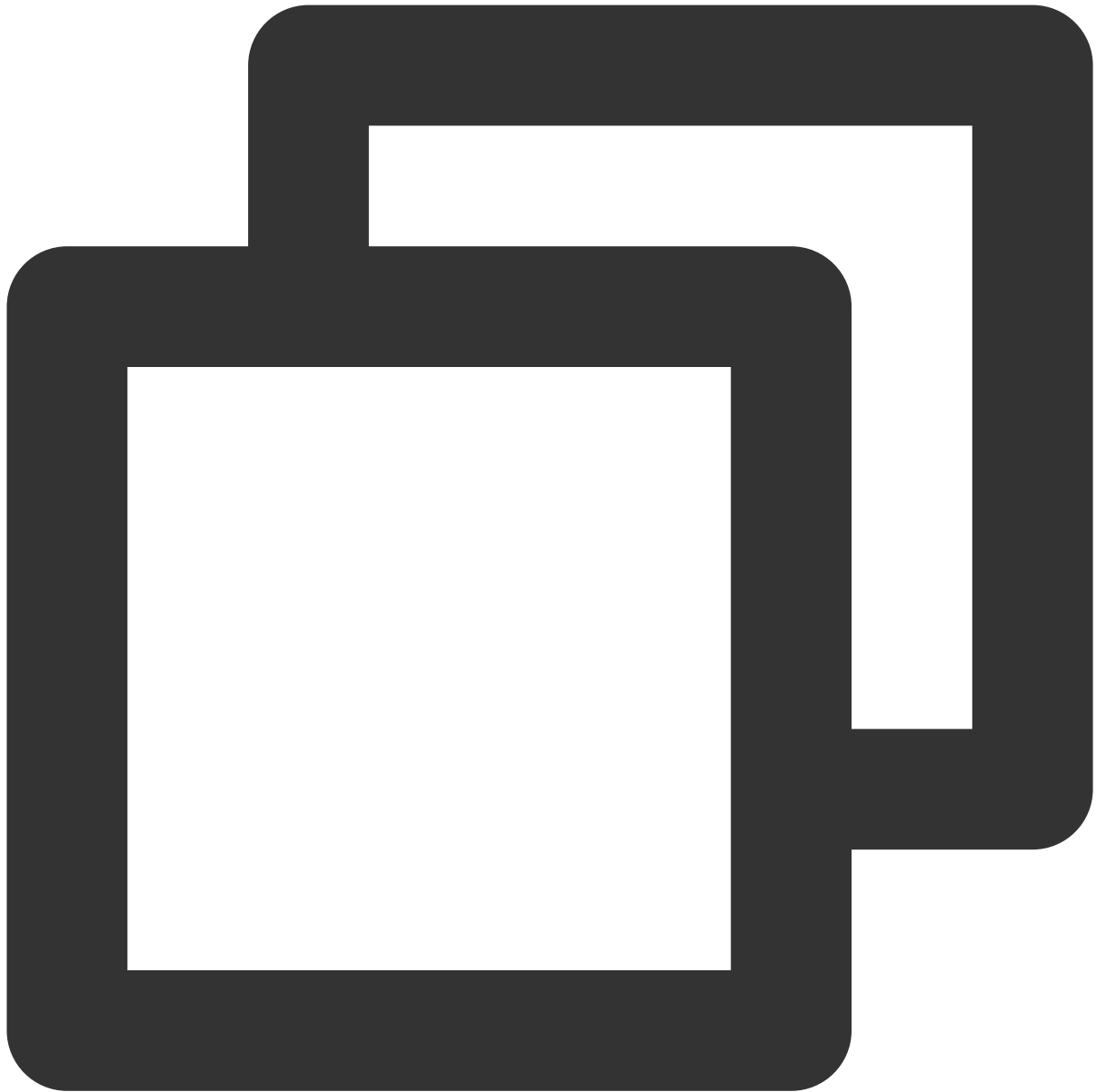
Note :

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
```

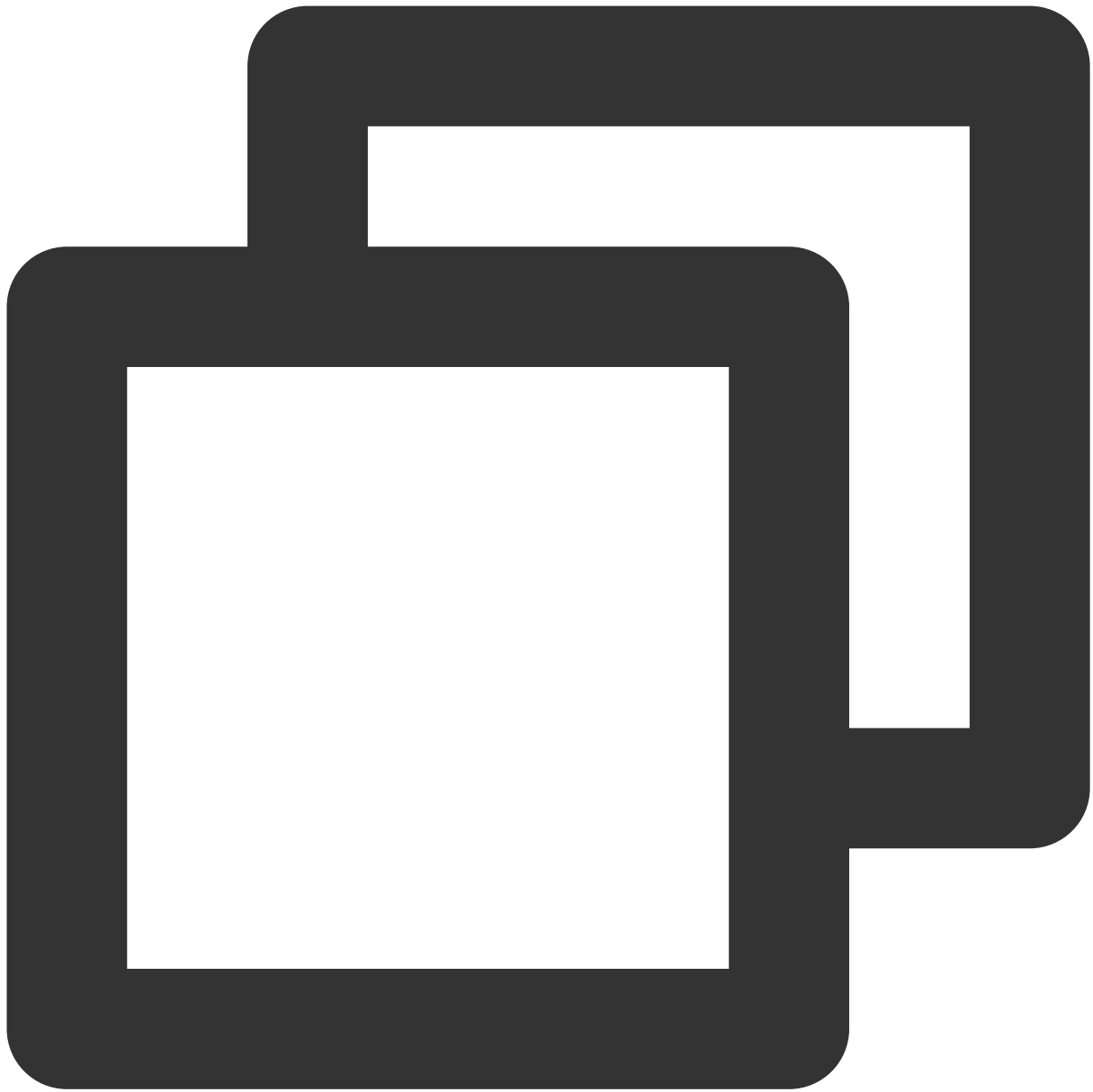
Note:

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\\")
```

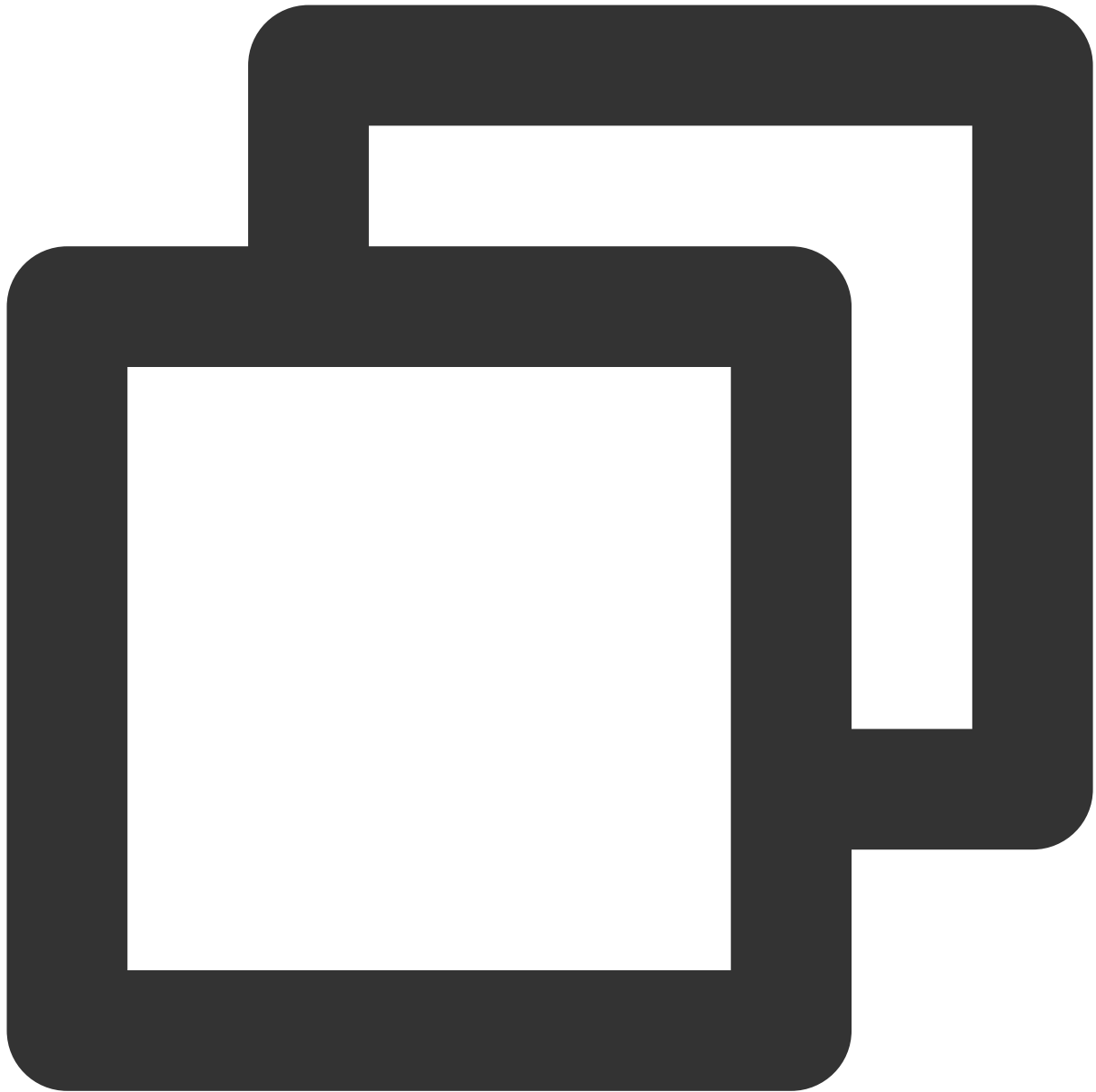

Mixing Stream Solution

iOS

Last updated : 2023-09-26 16:53:38

Specific code implementation

1. Create main and sub-instances

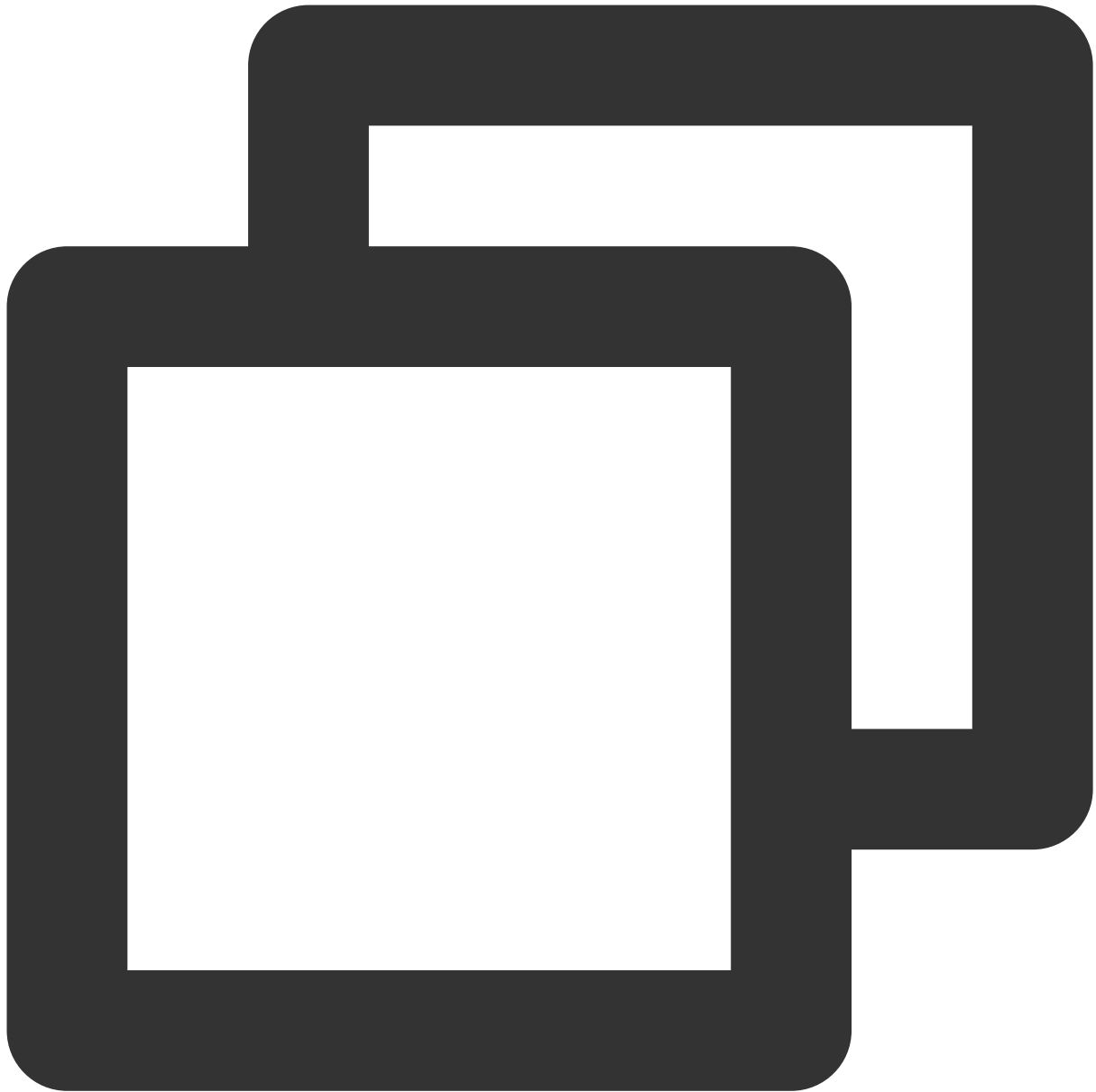


```
// Create TRTCcloud main instance (vocal instance)
TRTCcloud *trtcCloud = [TRTCcloud sharedInstance];
// Create TRTCcloud sub-instance (accompaniment instance)
TRTCcloud *subCloud = [trtcCloud createSubCloud];
```

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:@"%s", 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 :

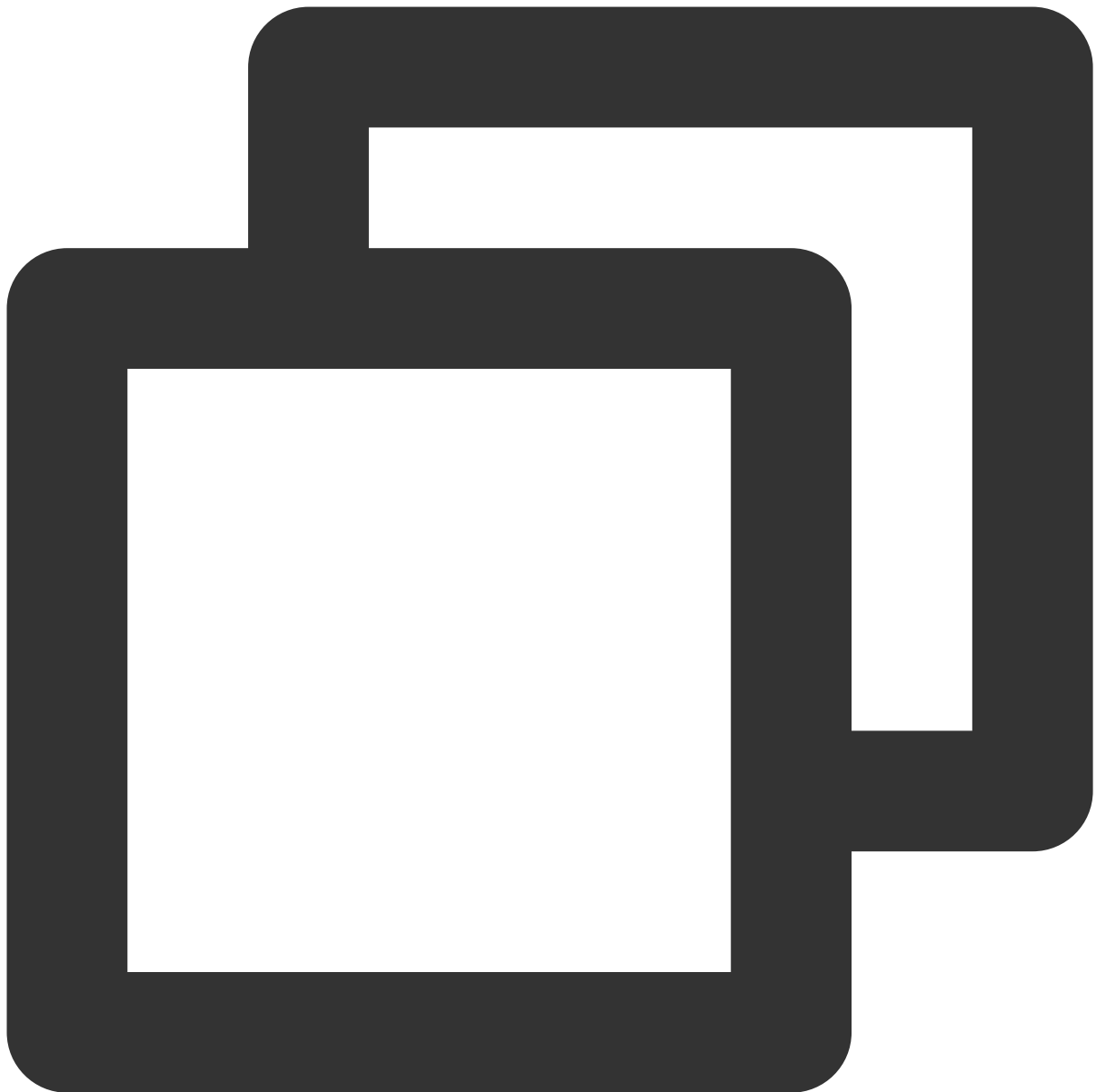
Pay attention to distinguish between the userId of the main instance and the sub-instance, ensuring that they are not duplicated and easy to identify;

Accompaniment instance background music parameter musicParam settings:

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.

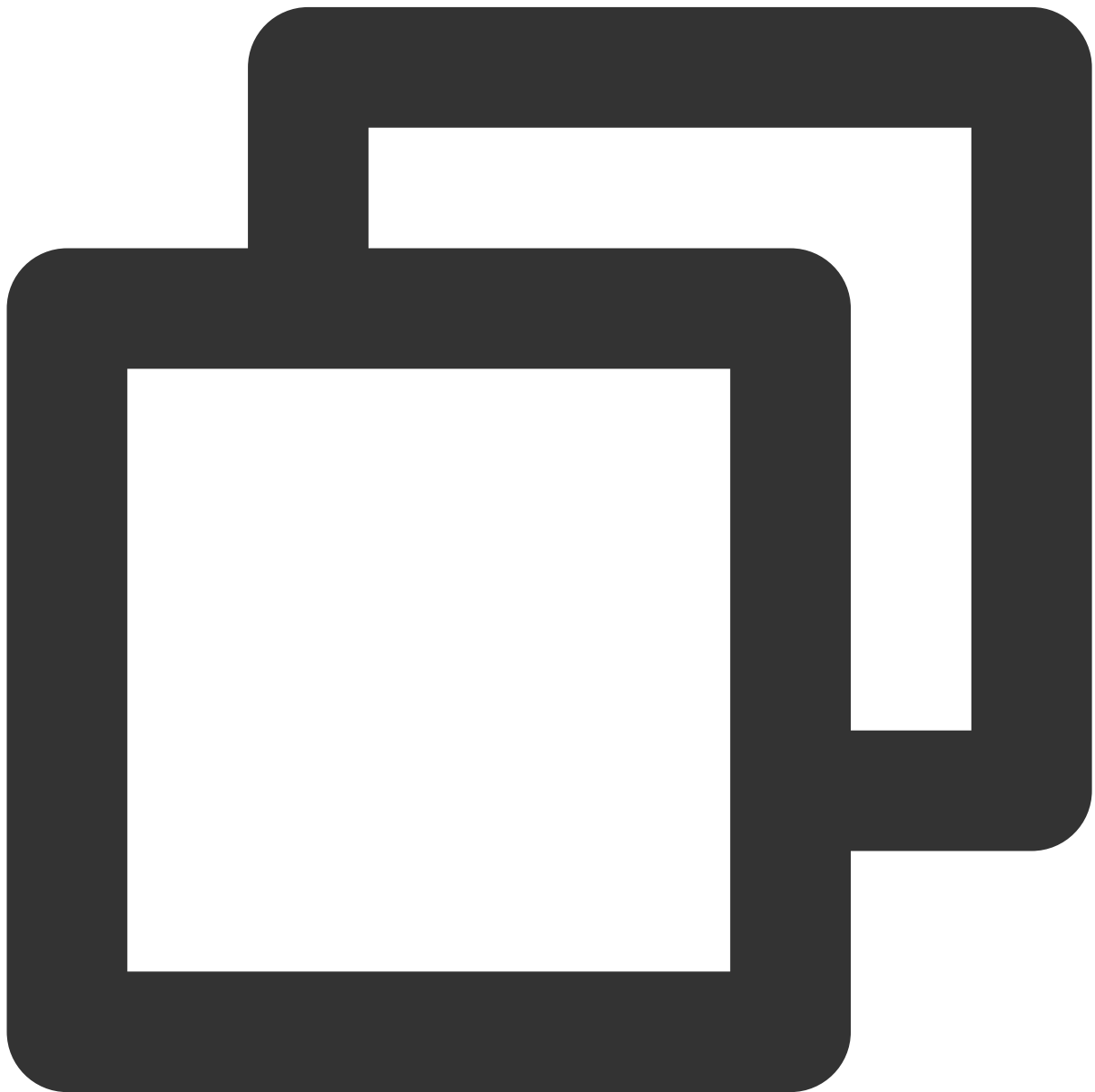
The mixed stream transcoding and pushing method `startPublishMediaStream` used here adopts a brand new backend architecture. The old version of the application needs to provide the `SdkAppId` to apply for an upgrade before it can be used.

Android

Last updated : 2023-09-26 16:54:10

Specific code implementation

1. Create main and sub-instances

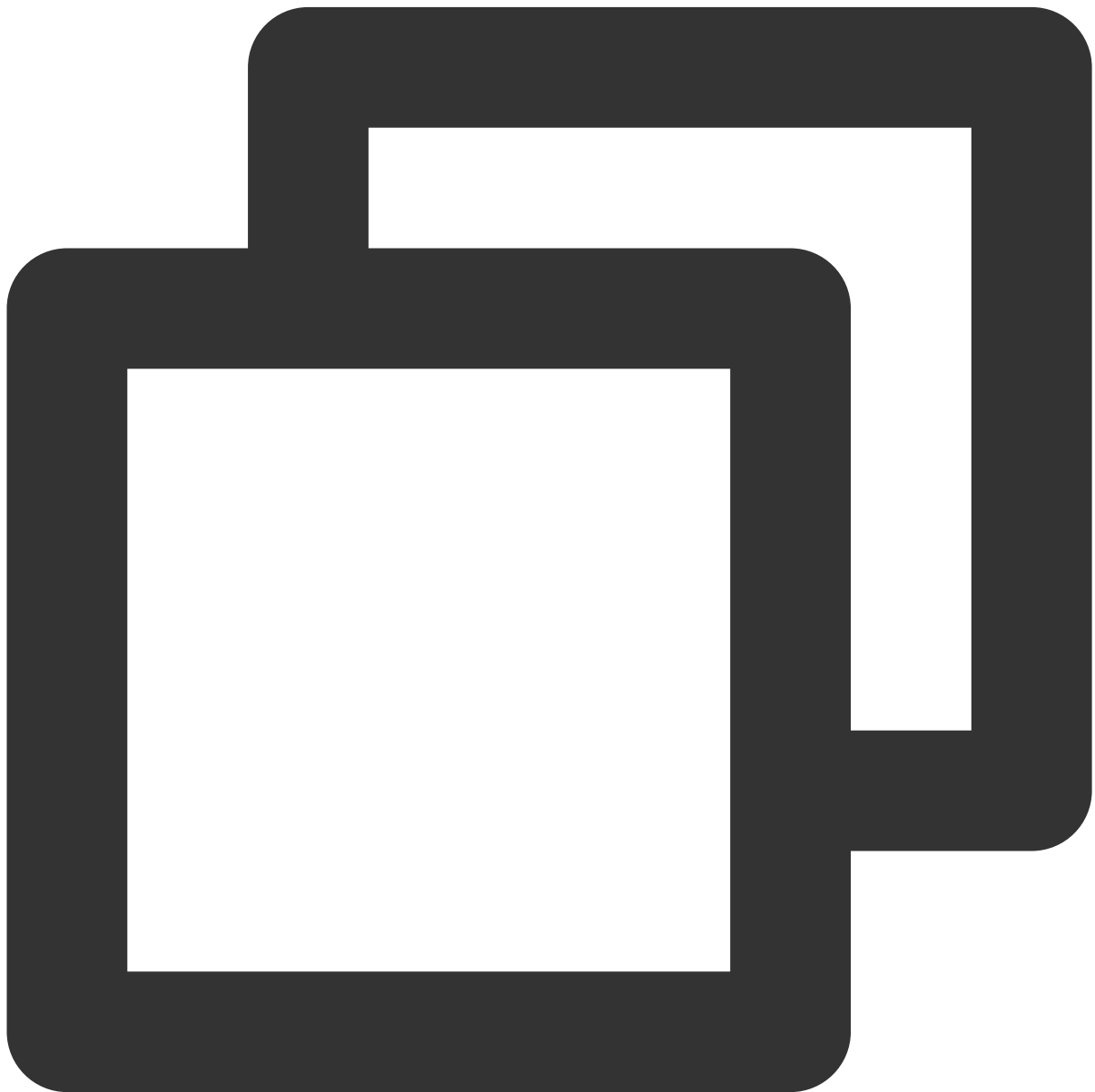


```
// Create TRTCcloud main instance (vocal instance)
```

```
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.TRTCParams params = new TRTCCloudDef.TRTCParams();
```

```
params.sdkAppId = sdkAppId;
params.userId = mUserId;
params.userSig = userSig;
params.role = TRTCCloudDef.TRTCRoleAnchor;
params.roomId = mRoomId;
mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
// Turn on audio uplink and set audio quality
mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
// Set media type
mTRTCCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);
// 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.TRTCStreamEncoderParam();
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
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
setDelegate	Sets event callbacks.
delegateQueue	Sets the thread where event callbacks are.

login	Logs in.
logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
destroyRoom	Terminates a room (called by room owner).
enterRoom	Enters a room (called by listener).
exitRoom	Exits a room (called by listener).
getRoomInfoList	Gets room list details.
getUserInfoList	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
startPlayMusic	Starts music.
stopPlayMusic	Stops music.
pausePlayMusic	Pauses music.
resumePlayMusic	Resumes music.

Seat management APIs

API	Description
enterSeat	Becomes a speaker (called by room owner or listener).
leaveSeat	Becomes a listener (called by speaker).
pickSeat	Places a user in a seat (called by room owner).
kickSeat	Removes a speaker (called by room owner).

muteSeat	Mutes/Unmutes a seat (called by room owner).
closeSeat	Blocks/Unblocks a seat (called by room owner).

Local audio APIs

API	Description
startMicrophone	Starts mic capturing.
stopMicrophone	Stops mic capturing.
setAudioQuality	Sets audio quality.
muteLocalAudio	Mutes/Unmutes local audio.
setSpeaker	Sets whether to play sound from the device's speaker or receiver.
setAudioCaptureVolume	Sets mic capturing volume.
setAudioPlayoutVolume	Sets playback volume.
setVoiceEarMonitorEnable	Enables/Disables in-ear monitoring.

Remote audio APIs

API	Description
muteRemoteAudio	Mutes/Unmutes a specified member.
muteAllRemoteAudio	Mutes/Unmutes all members.

Background music and audio effect APIs

API	Description
getAudioEffectManager	Gets the background music and audio effect management object TXAudioEffectManager .

Message sending APIs

API	Description
sendRoomTextMsg	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.

onAnchorEnterSeat	A user became a speaker or was made a speaker by the room owner.
onAnchorLeaveSeat	A user became a listener or was made a listener by the room owner.
onSeatMute	The room owner muted a seat.
onUserMicrophoneMute	Whether a user's mic is muted
onSeatClose	The room owner blocked a seat.

Callback APIs for room entry/exit by listener

API	Description
onAudienceEnter	A listener entered the room.
onAudienceExit	A listener exited the room.

Message event callback APIs

API	Description
onRecvRoomTextMsg	A text chat message was received.
onRecvRoomCustomMsg	A custom message was received.

Signaling event callback APIs

API	Description
onReceiveNewInvitation	Receipt of an invitation.
onInviteeAccepted	Invitation accepted by invitee.
onInviteeRejected	Invitation declined by invitee.
onInvitationCancelled	Invitation canceled by inviter.

Song event callback APIs

API	Description
onMusicProgressUpdate	Music playback progress.
onMusicPrepareToPlay	Music playback is ready.

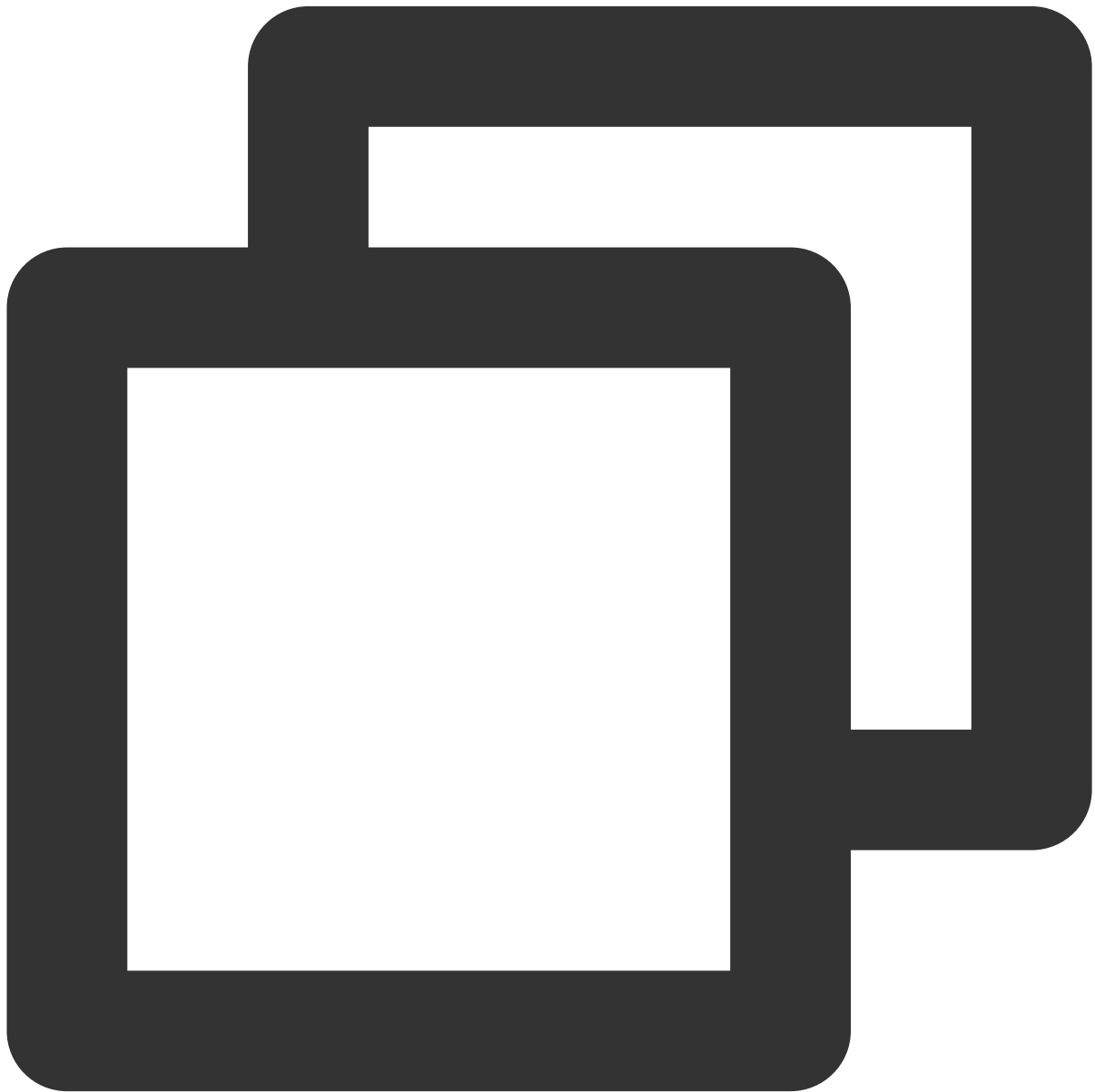
`onMusicCompletePlaying`

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

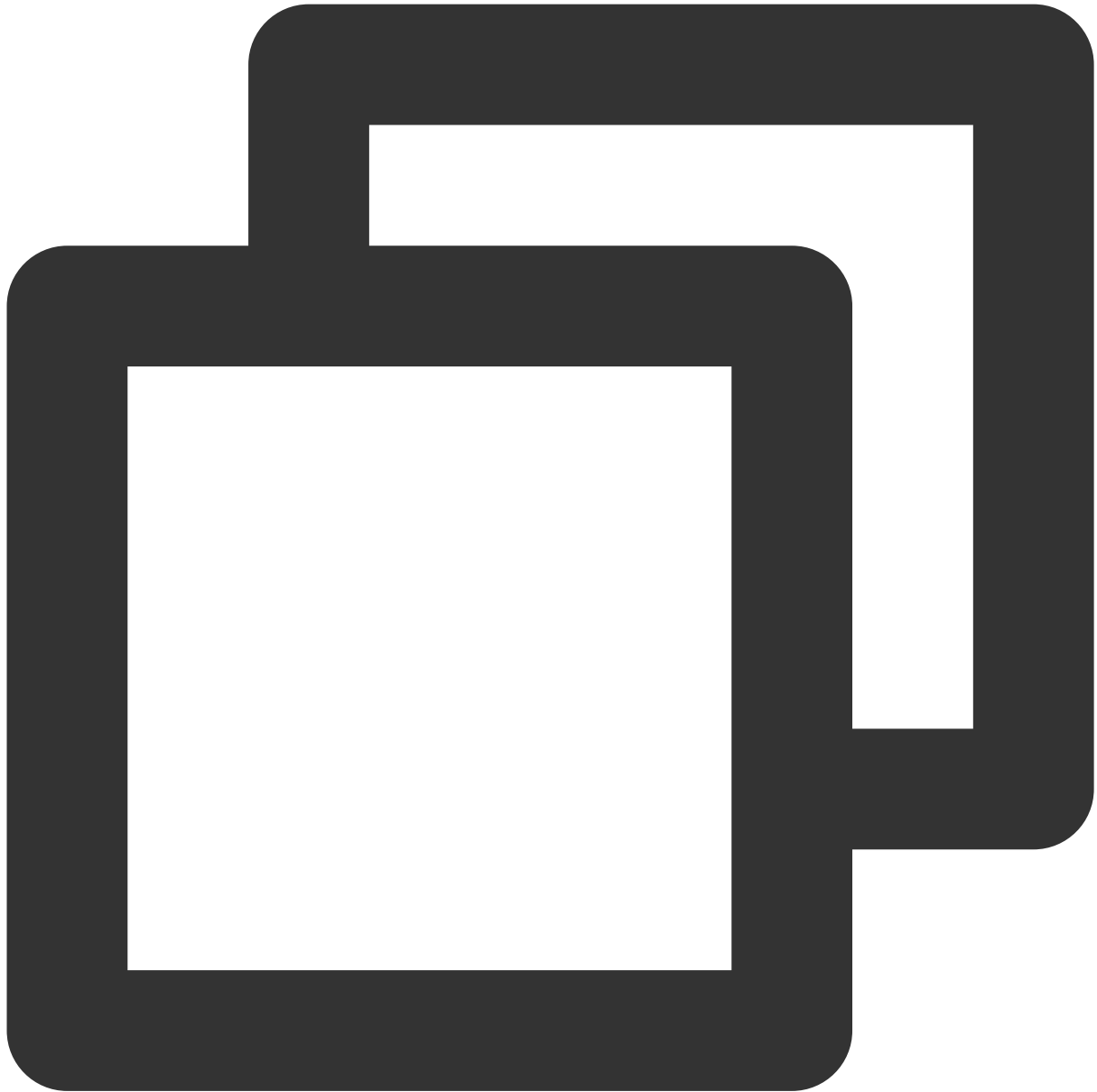
After the instance is terminated, the externally cached `TRTCKaraokeRoom` instance can no longer be used. You need to call [sharedInstance](#) again to get a new instance.



```
/**
 * 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 `T
 *
 * - parameter delegate Callback API
 * - note: Callbacks in `TRTCKaraokeRoom` are sent to you in the main queue by defau
 */
```



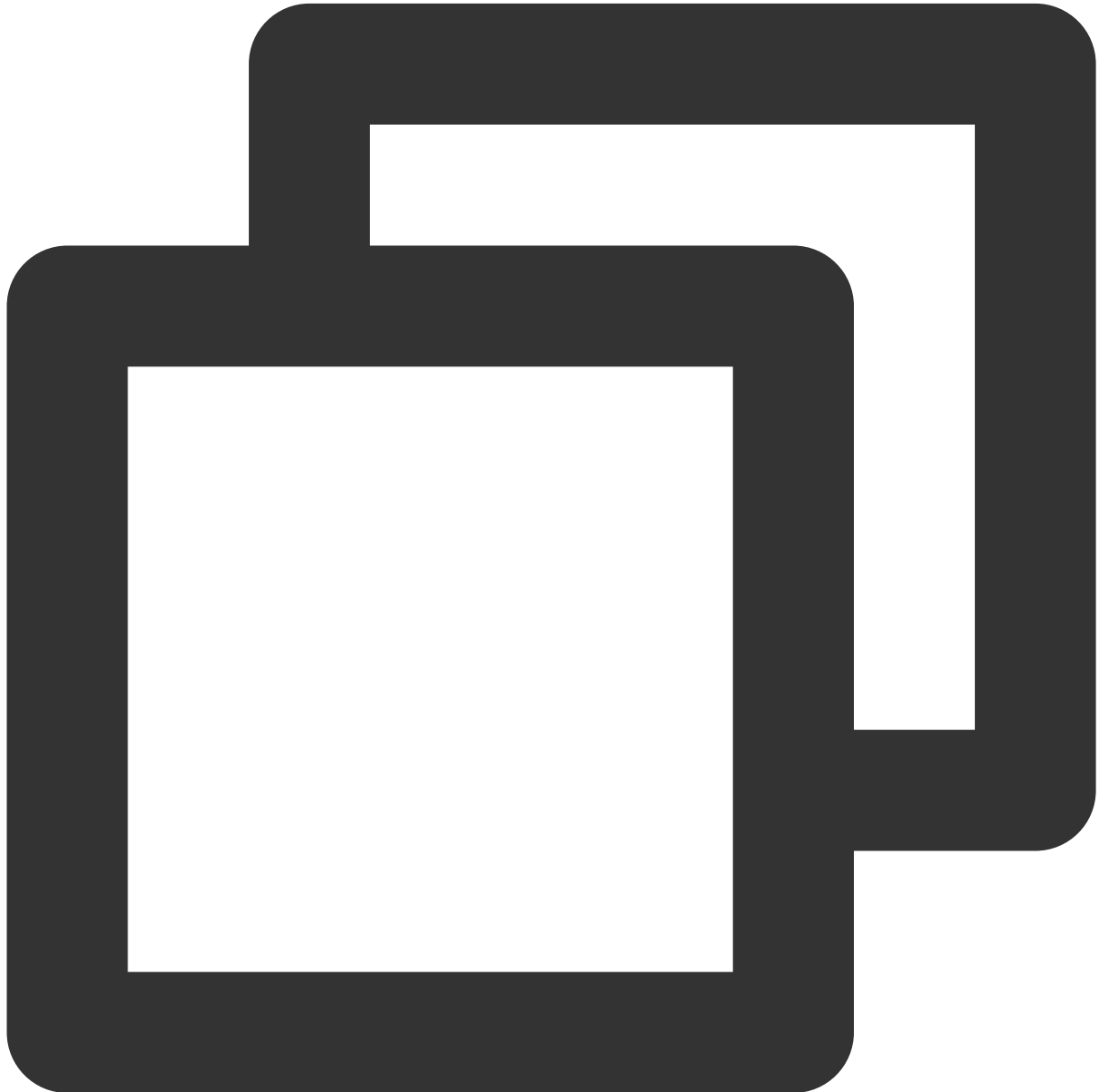
```
- (void)setDelegate:(id<TRTCKaraokeRoomDelegate>)delegate NS_SWIFT_NAME(setDelegate)
```

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:callback:))
```

The parameters are described below:

Parameter	Type	Description
sdkAppId	int	You can view the <code>SDKAppID</code> via Application Management > Application

		Info 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 FAQs > UserSig .
callback	ActionCallback	The callback for login. The code is <code>0</code> if login succeeds.

logout

Log out



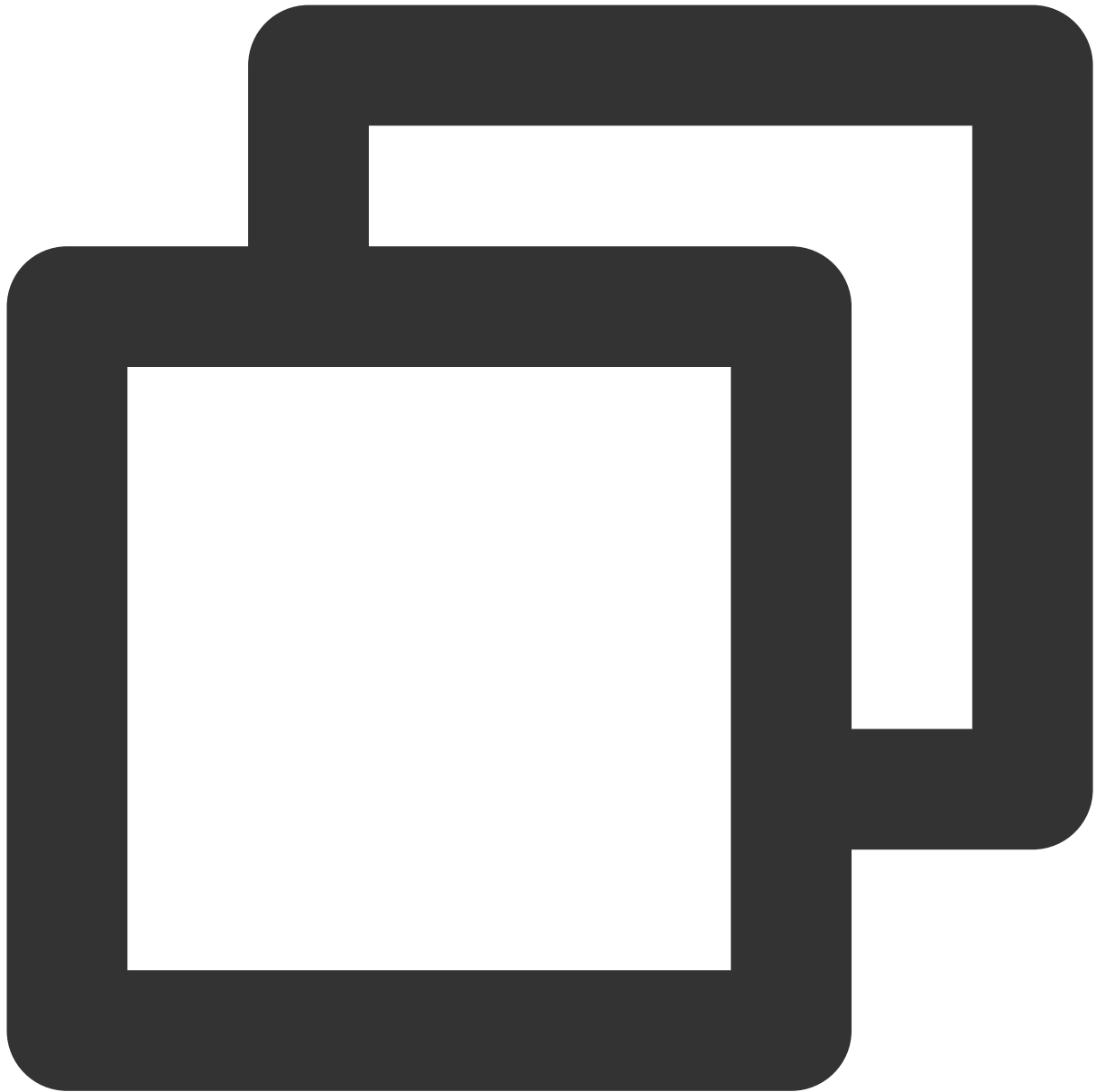
```
- (void)logout:(ActionCallback _Nullable)callback NS_SWIFT_NAME(logout(callback:));
```

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:

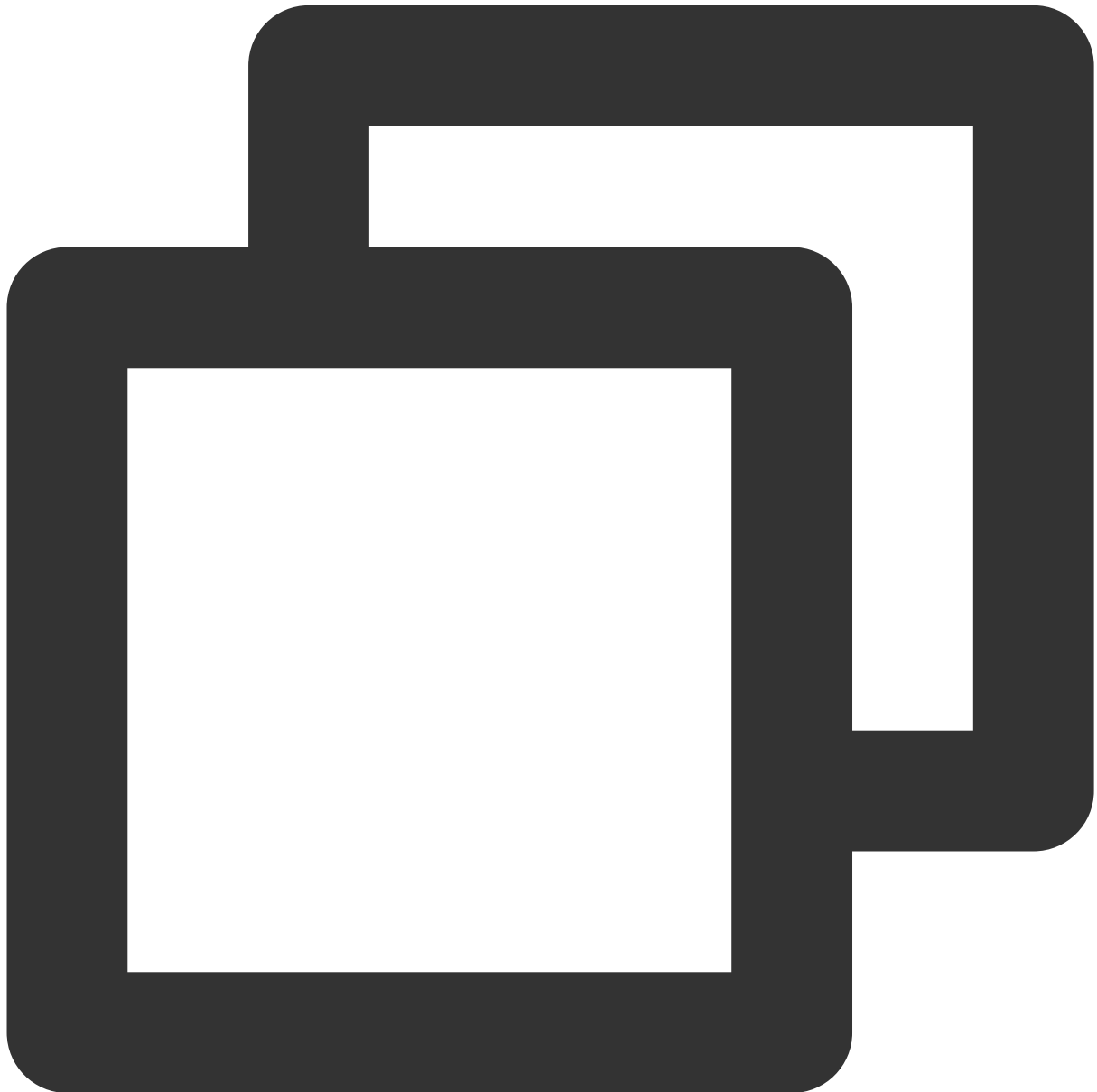
Parameter	Type	Description
userName	String	The username.
avatar	String	The address of the profile photo.
callback	ActionCallback	The callback for profile configuration. The code is 0 if the operation

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 `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).



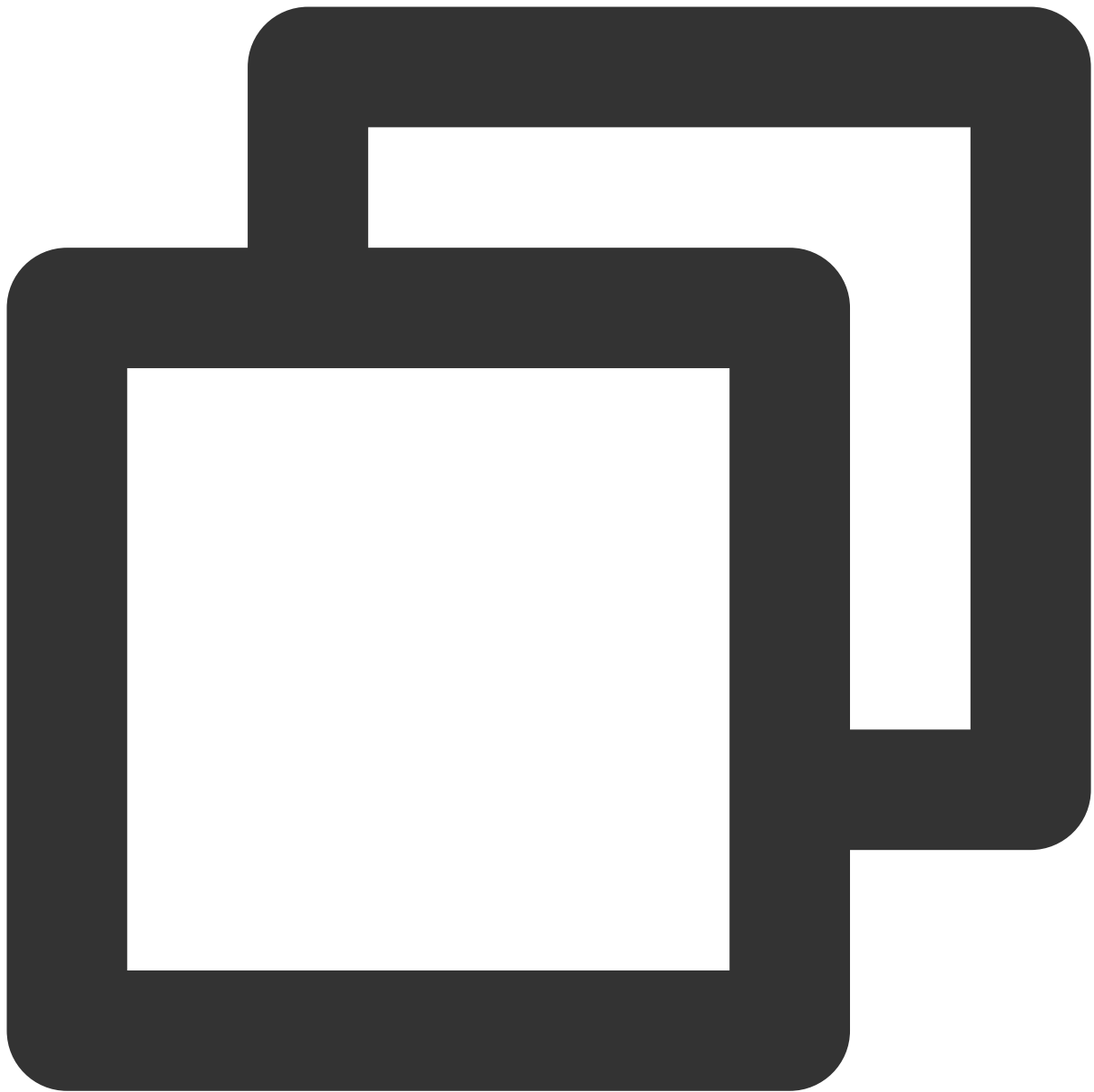
```
- (void)destroyRoom:(ActionCallback _Nullable)callback NS_SWIFT_NAME(destroyRoom(ca
```

The parameters are described below:

Parameter	Type	Description
callback	ActionCallback	The callback for room termination. The code is <code>0</code> 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:

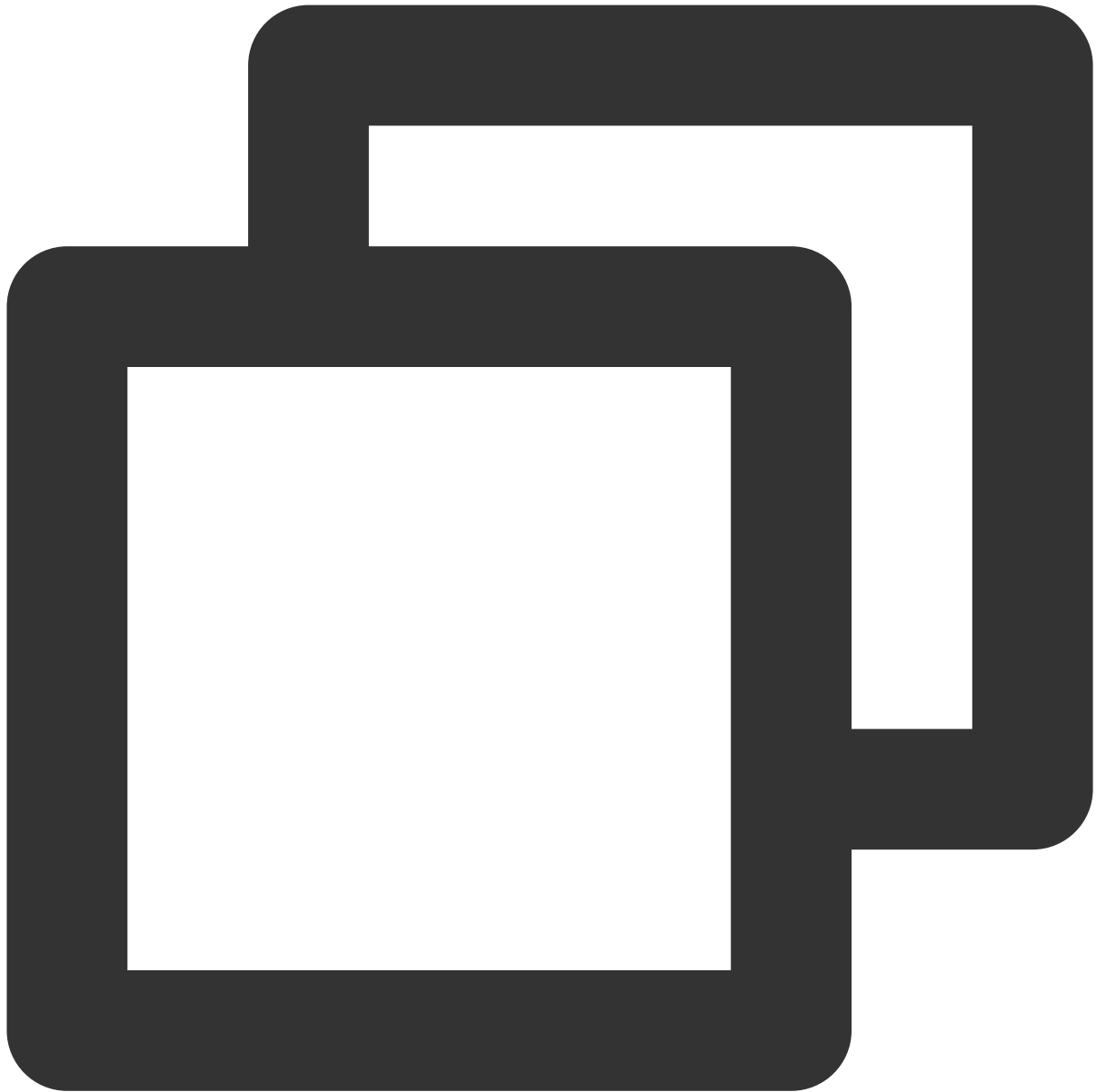
Parameter	Type	Description
roomId	int	The room ID.
callback	ActionCallback	The callback for room entry. The code is 0 if the operation succeeds.

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:

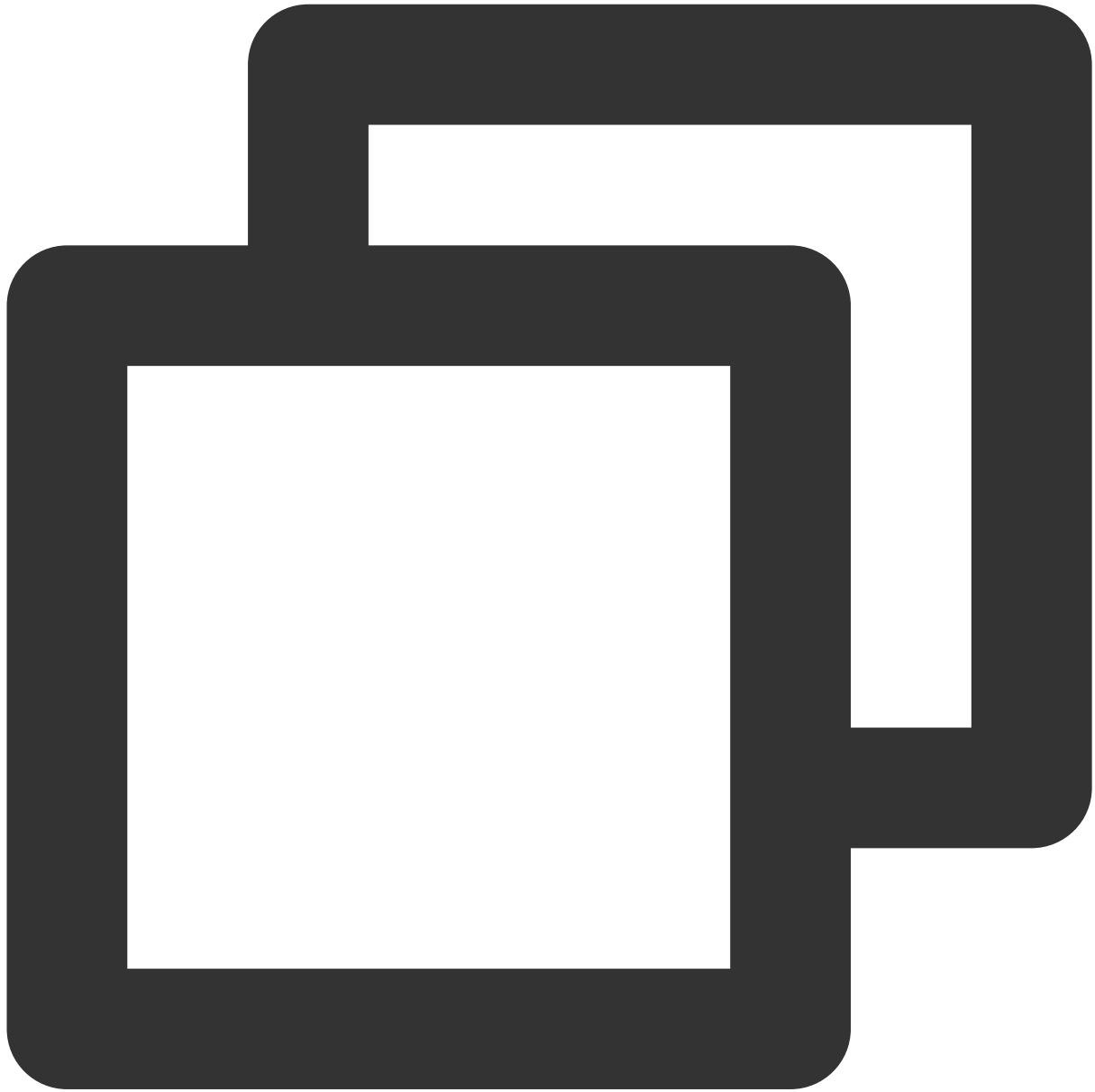
Parameter	Type	Description
callback	ActionCallback	The callback for room exit. The code is 0 if the operation succeeds.

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
```

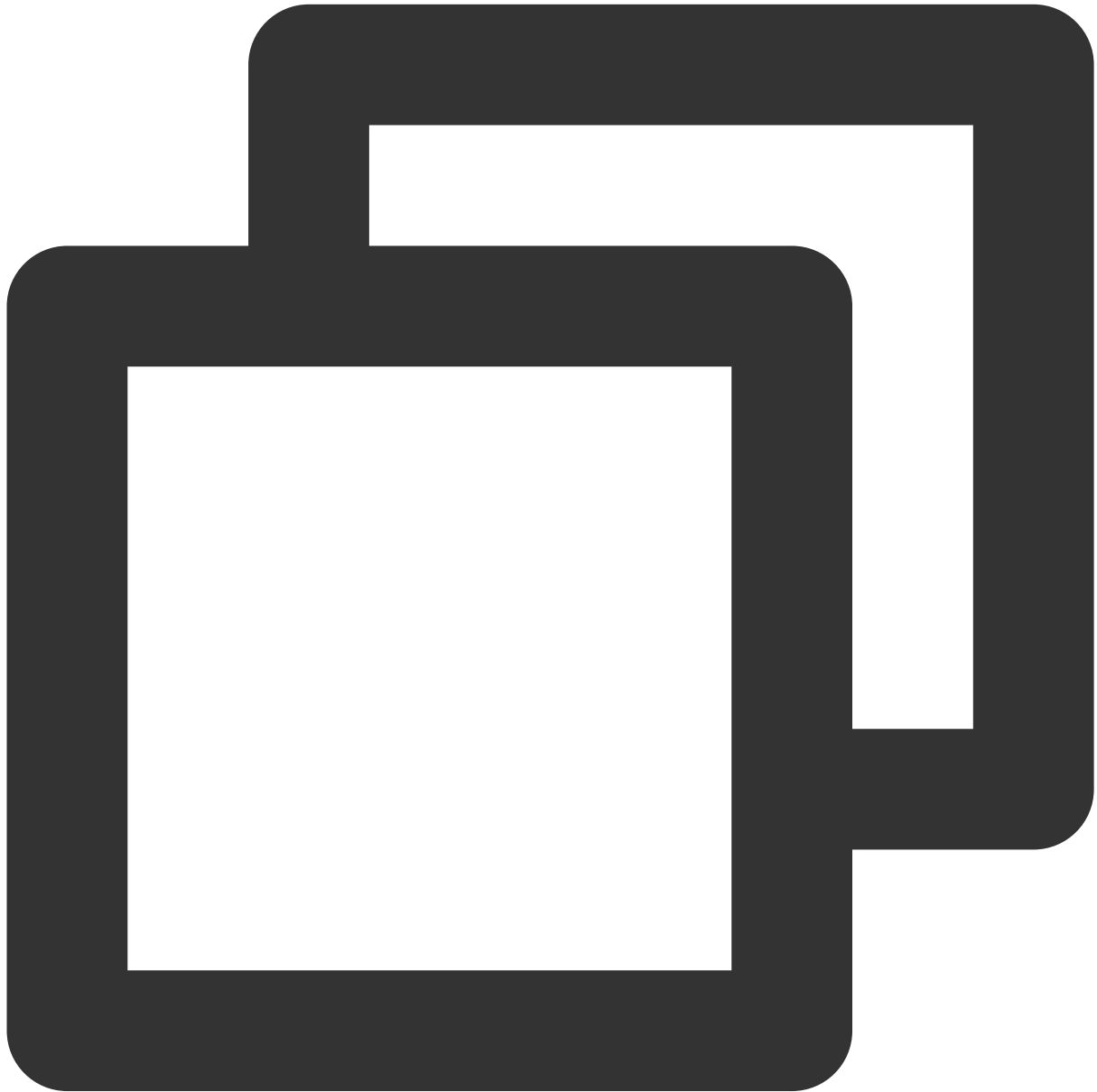
The parameters are described below:

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

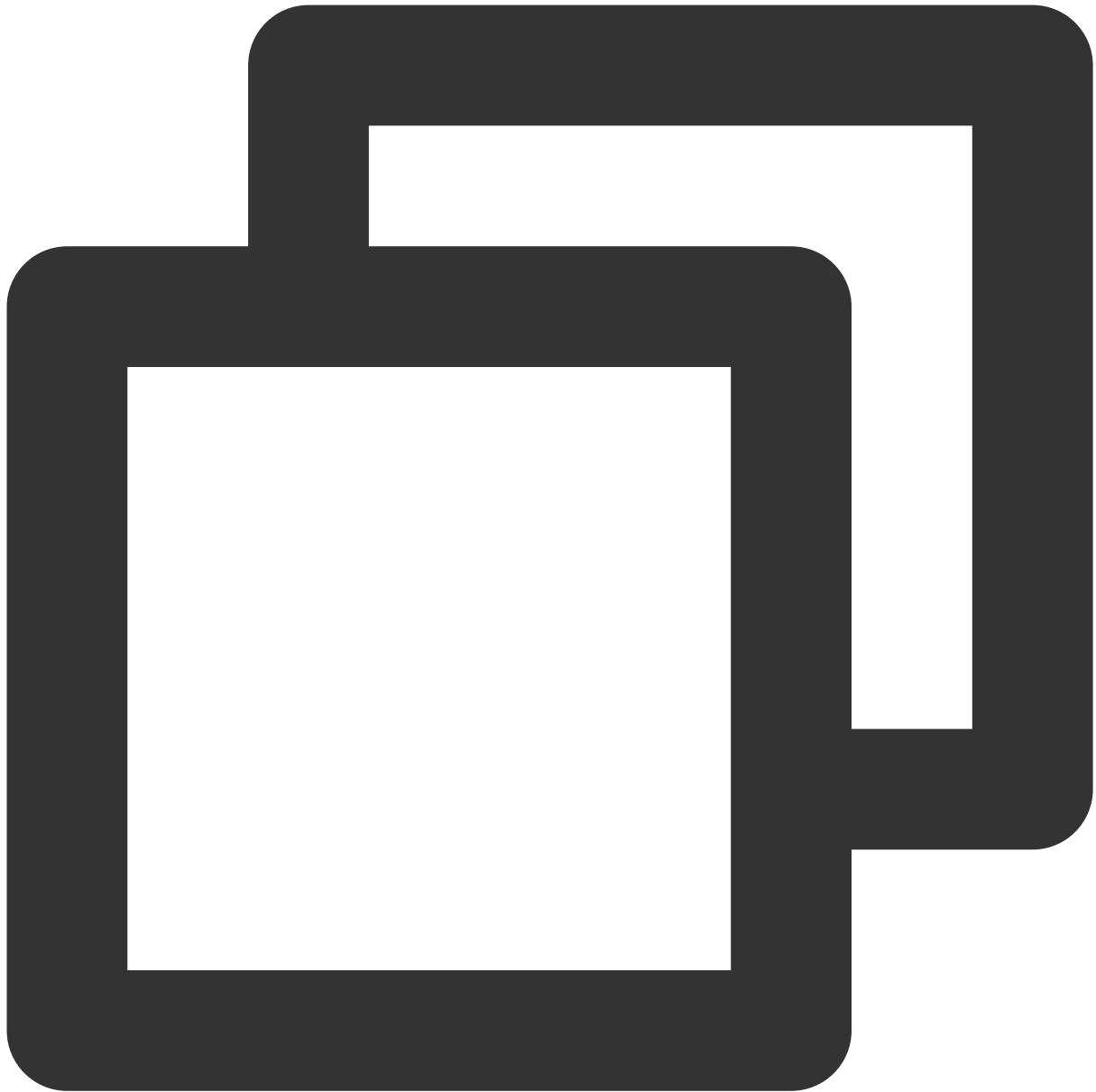
This API is used to play music (called after becoming a speaker).

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.

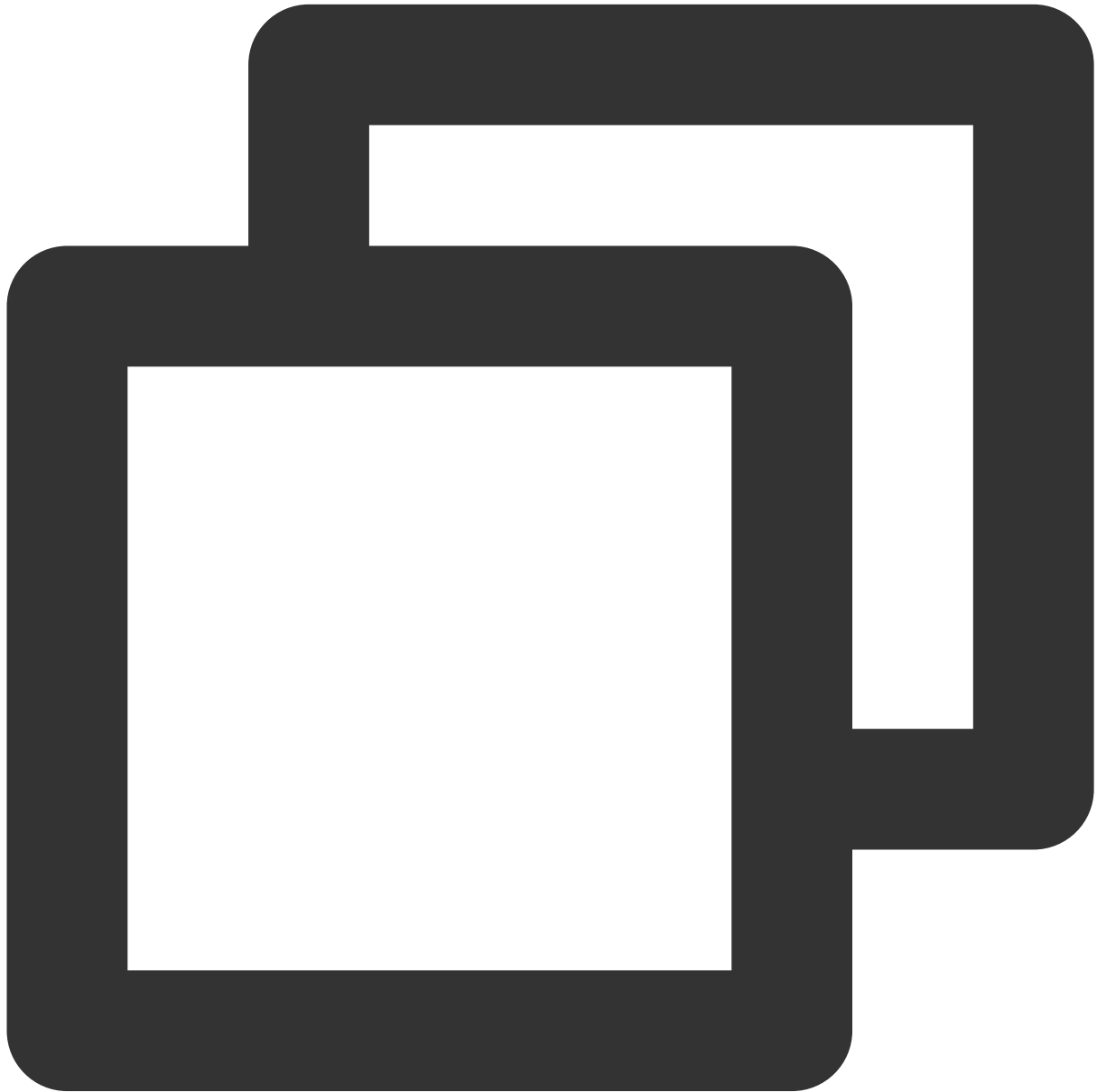
After this API is called, the song being played will stop.

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());
```

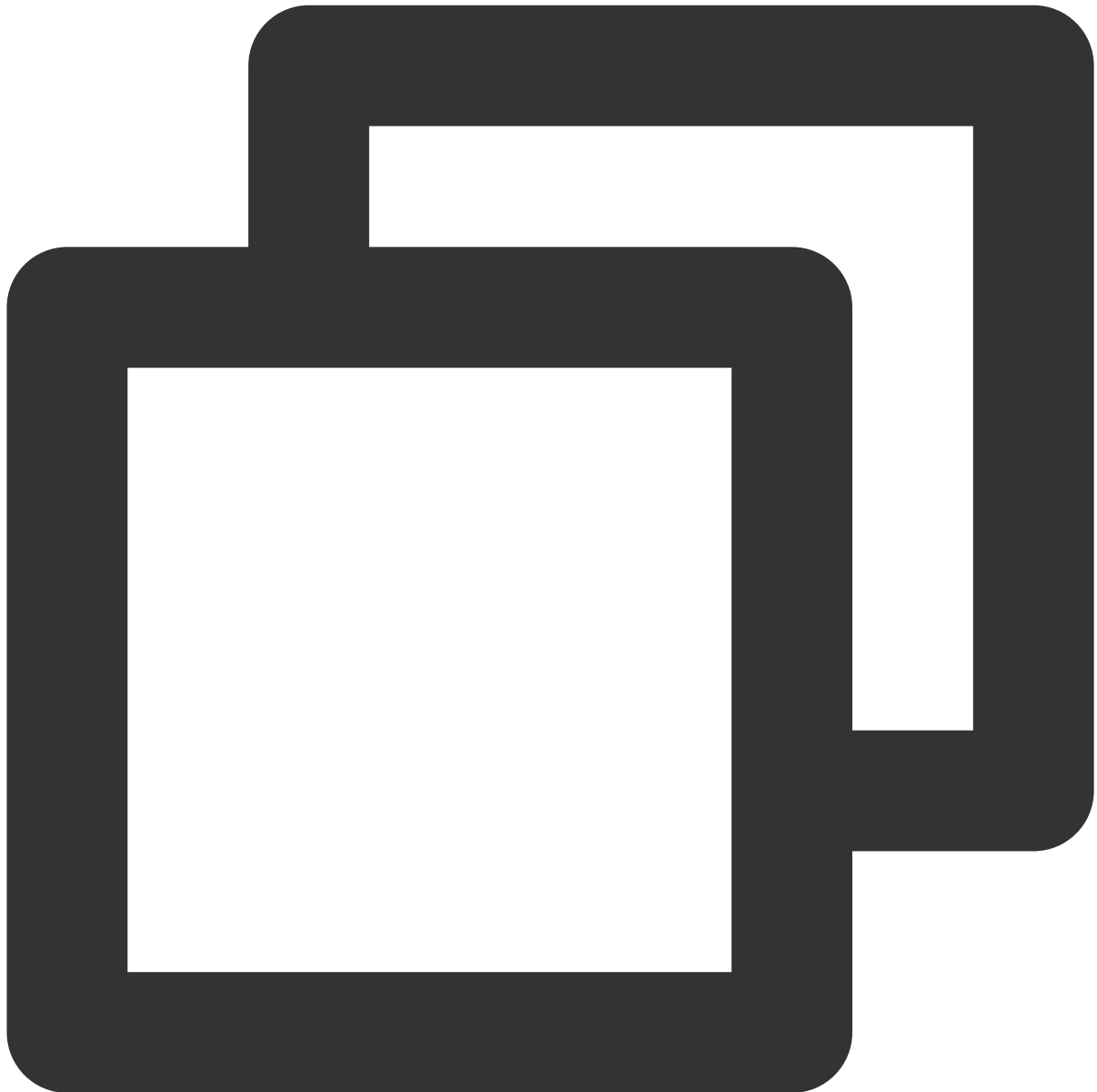
pausePlayMusic

This API is used to pause music (called during music playback).

Note

The `onMusicProgressUpdate` notification will be paused.

No `onMusicCompletePlaying` notification will be received.



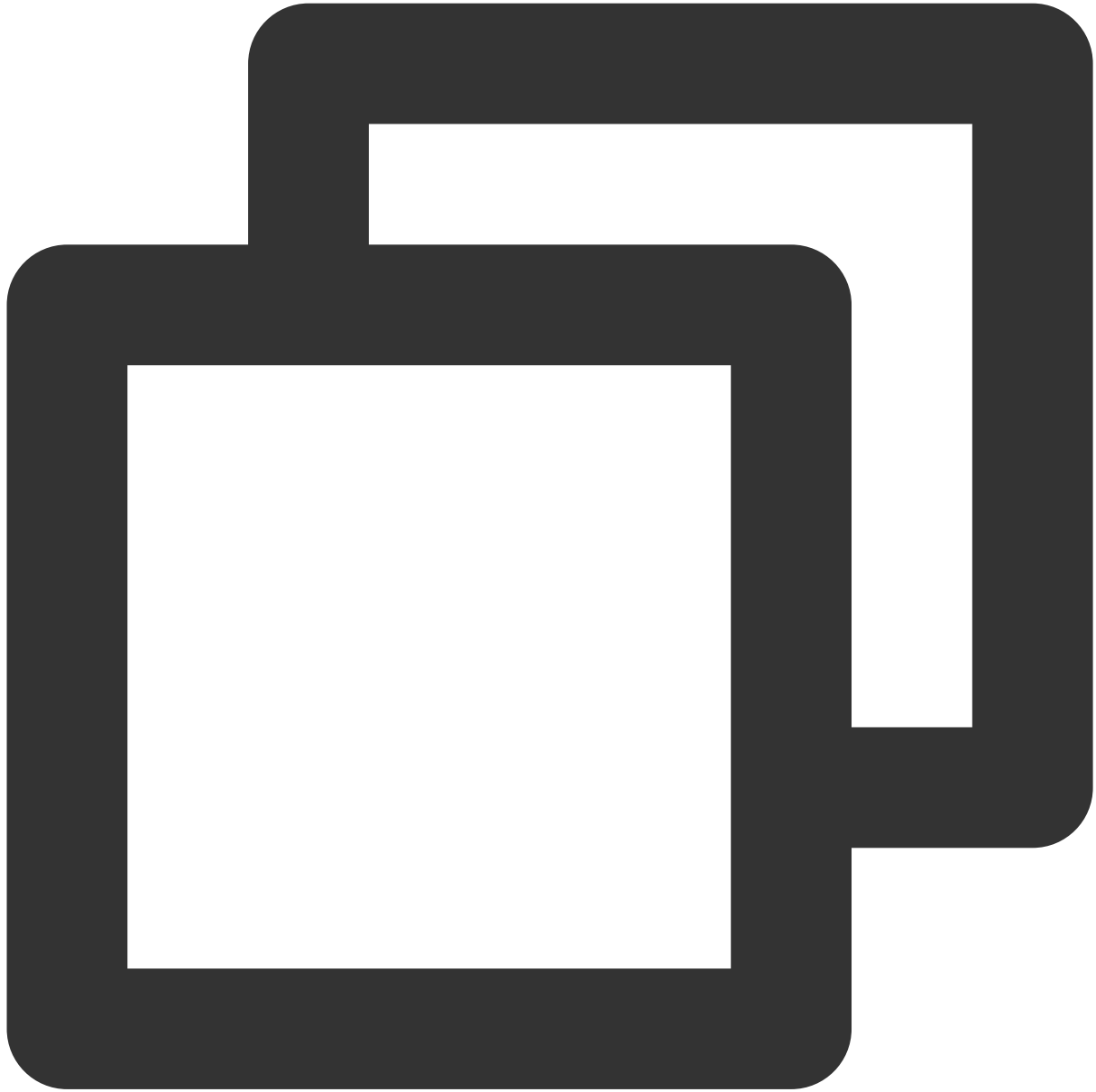
```
- (void)pausePlayMusic NS_SWIFT_NAME(pausePlayMusic());
```

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());
```

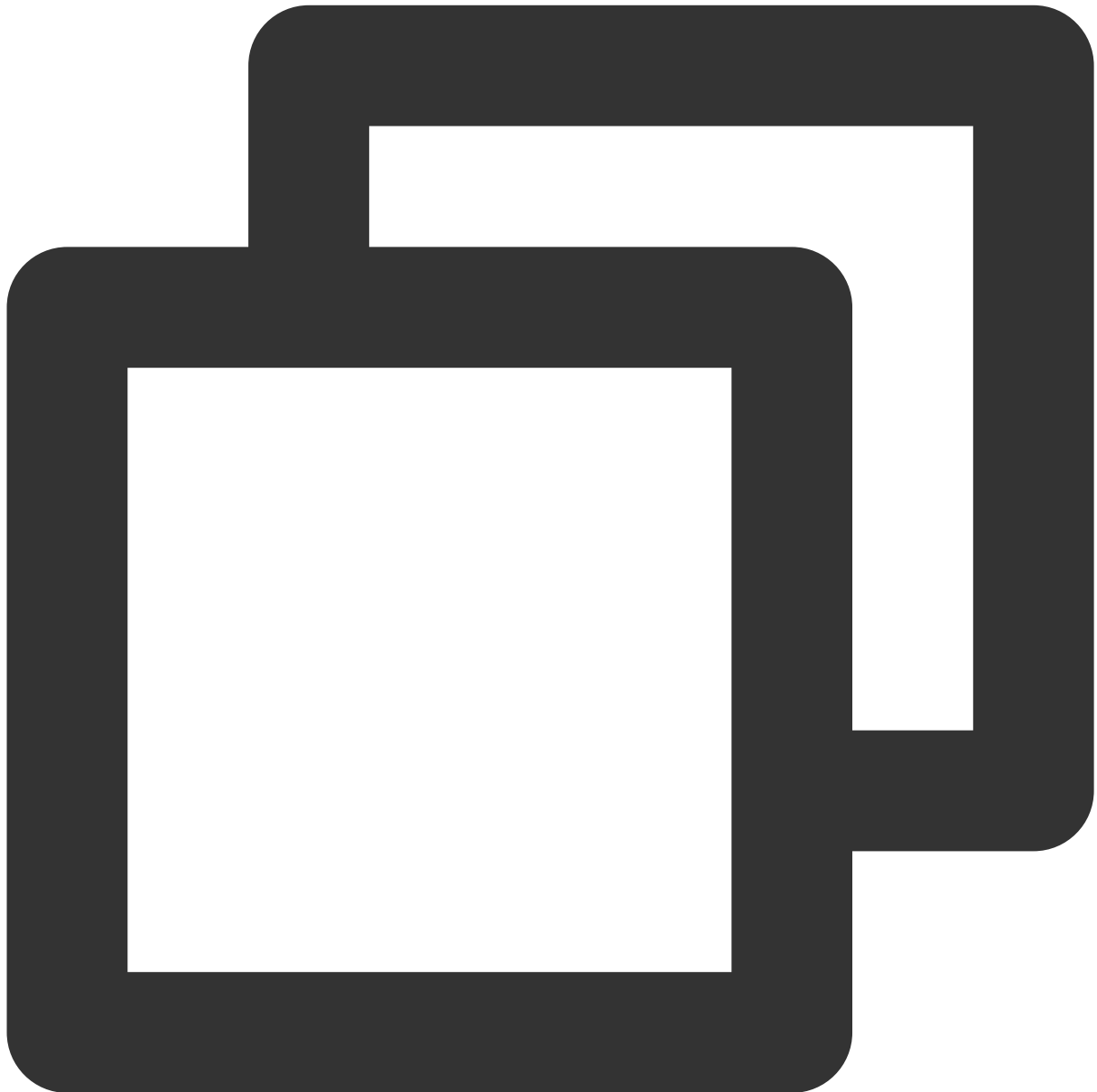
Seat Management APIs

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.

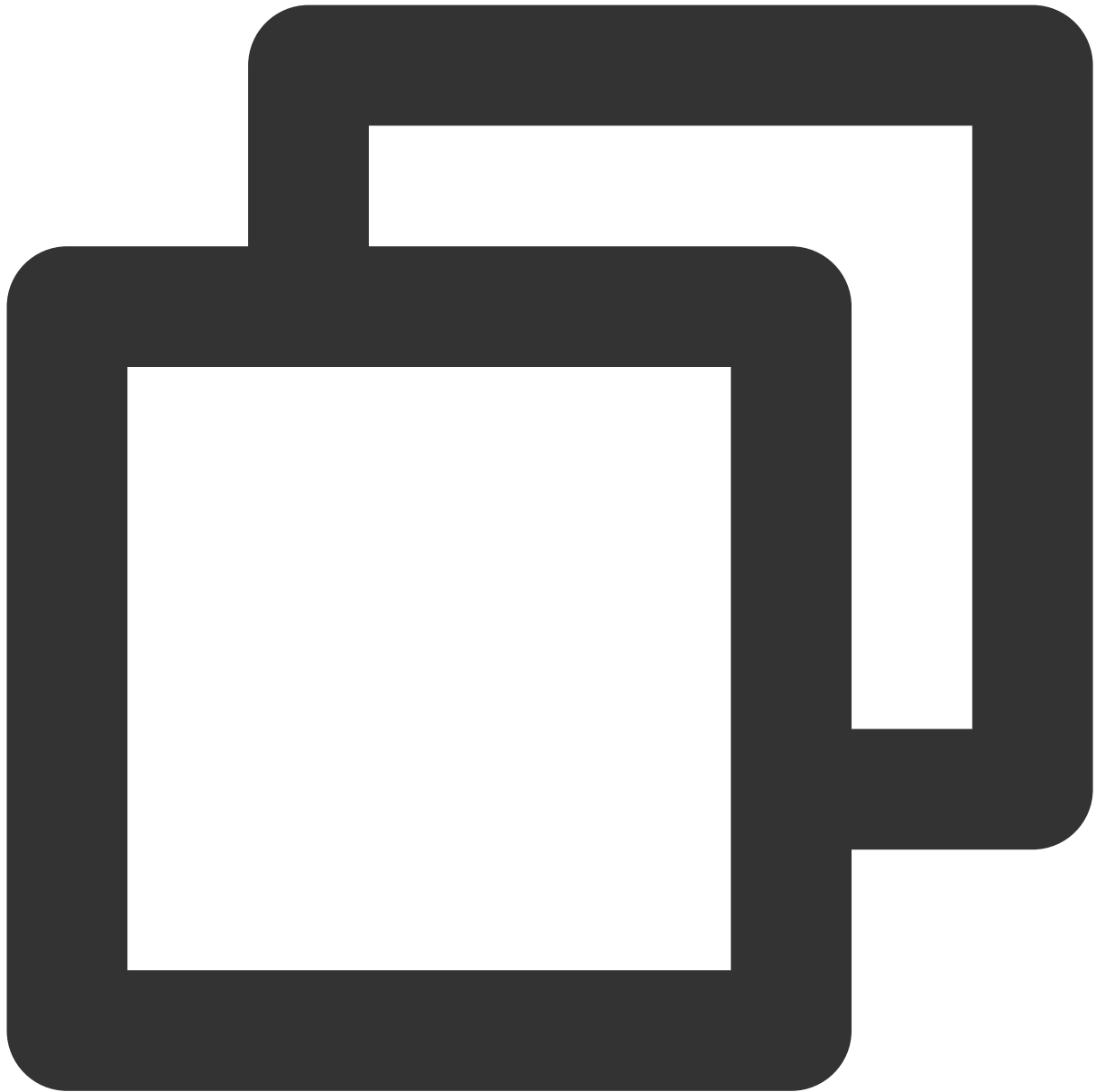
Calling this API will immediately modify the seat list. In cases where listeners need the room owner's permission to take a seat, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept`, call this API.

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(callba
```

The parameters are described below:

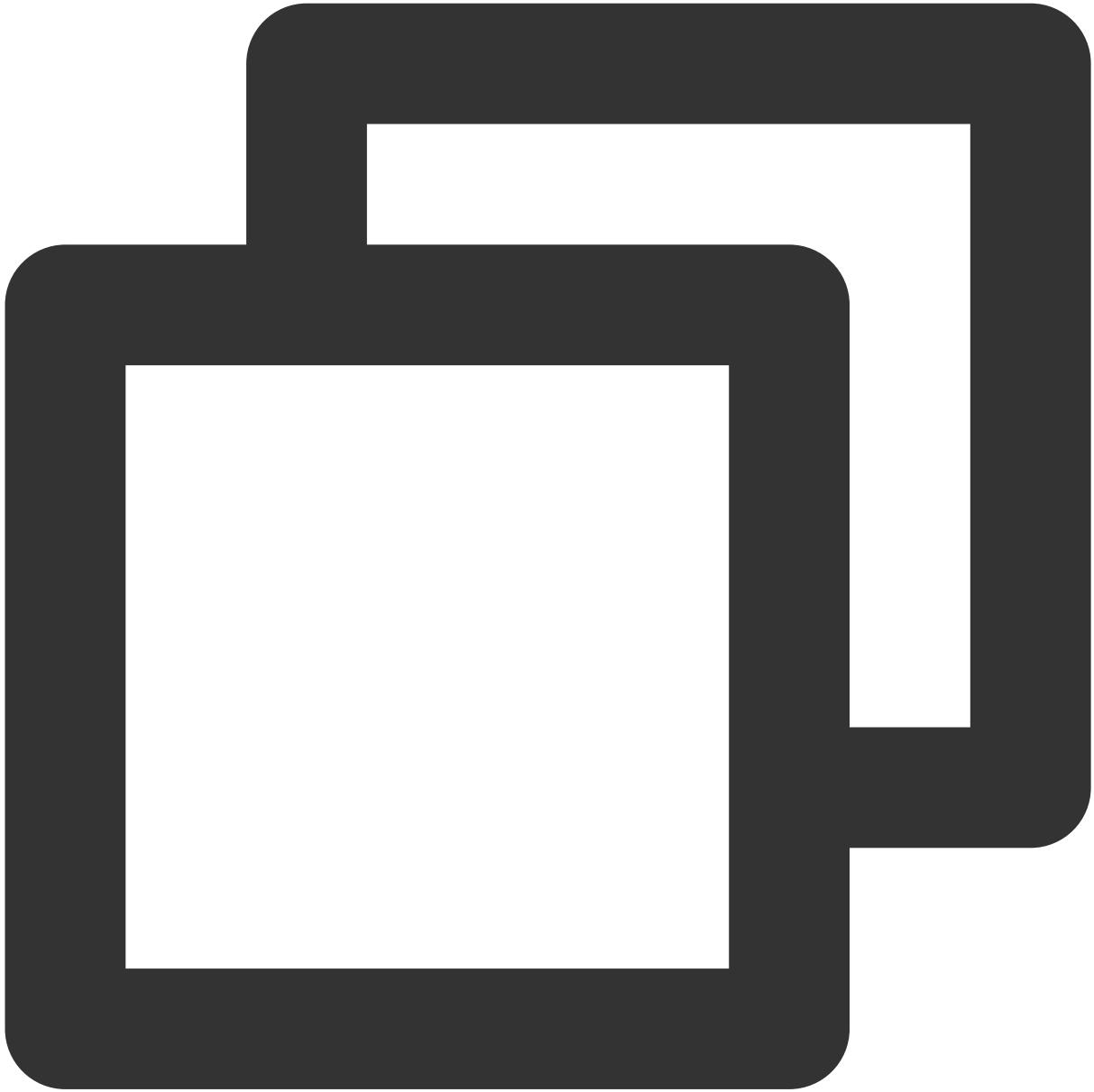
Parameter	Type	Description
callback	ActionCallback	The callback for the operation.

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.

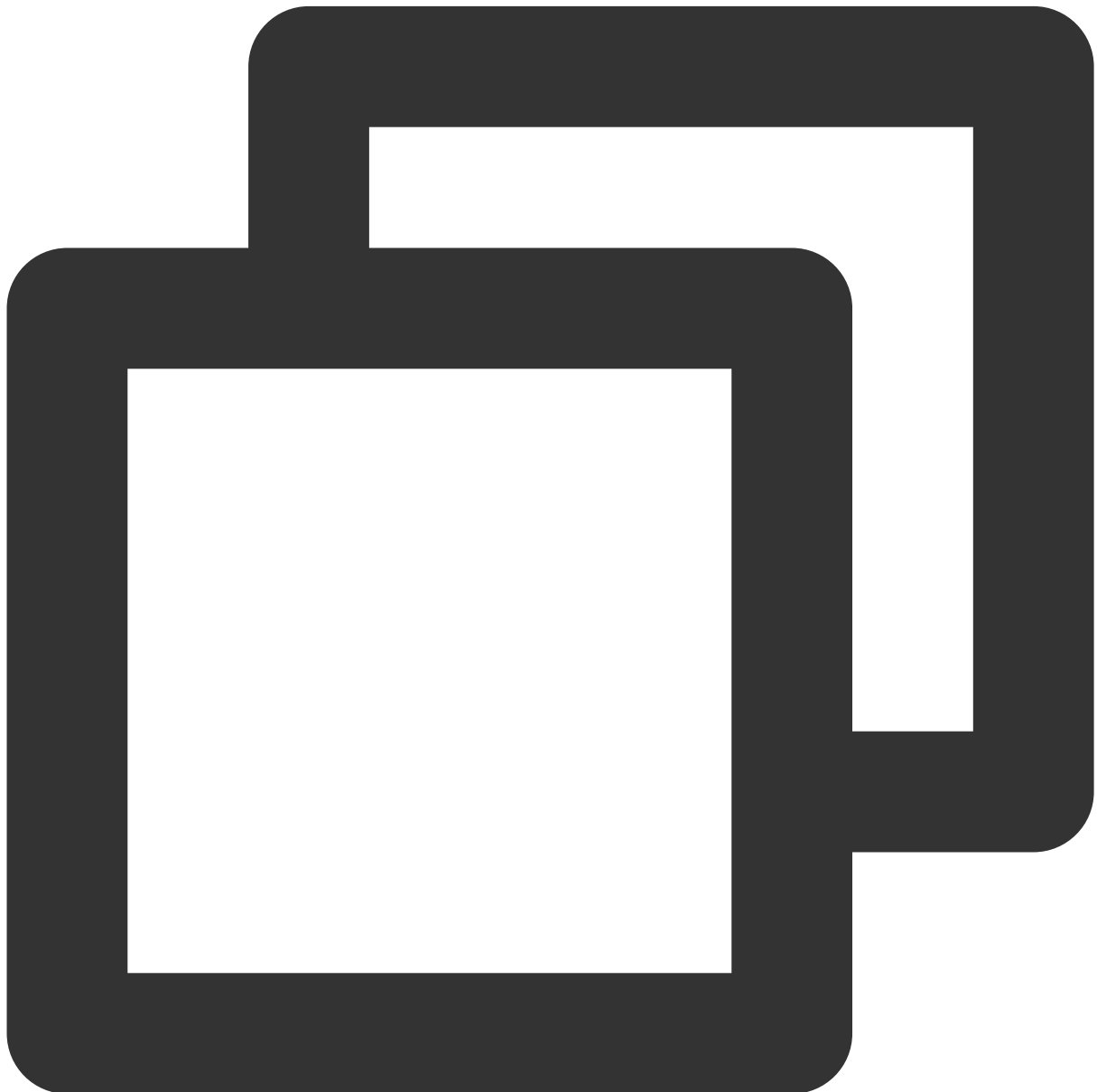
Calling this API will immediately modify the seat list.

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.

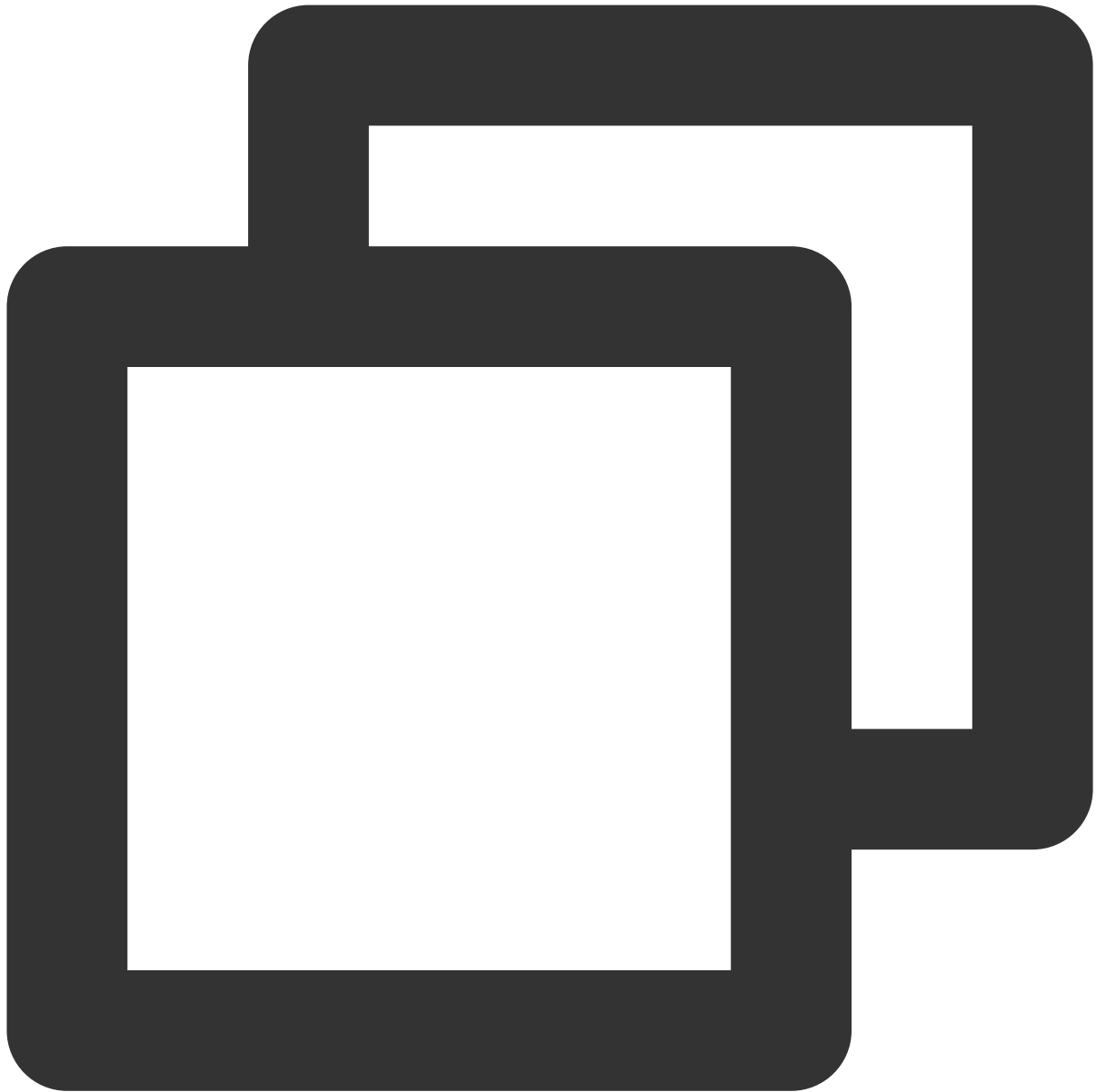
Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will call `muteAudio` to mute/unmute his or her audio.

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:

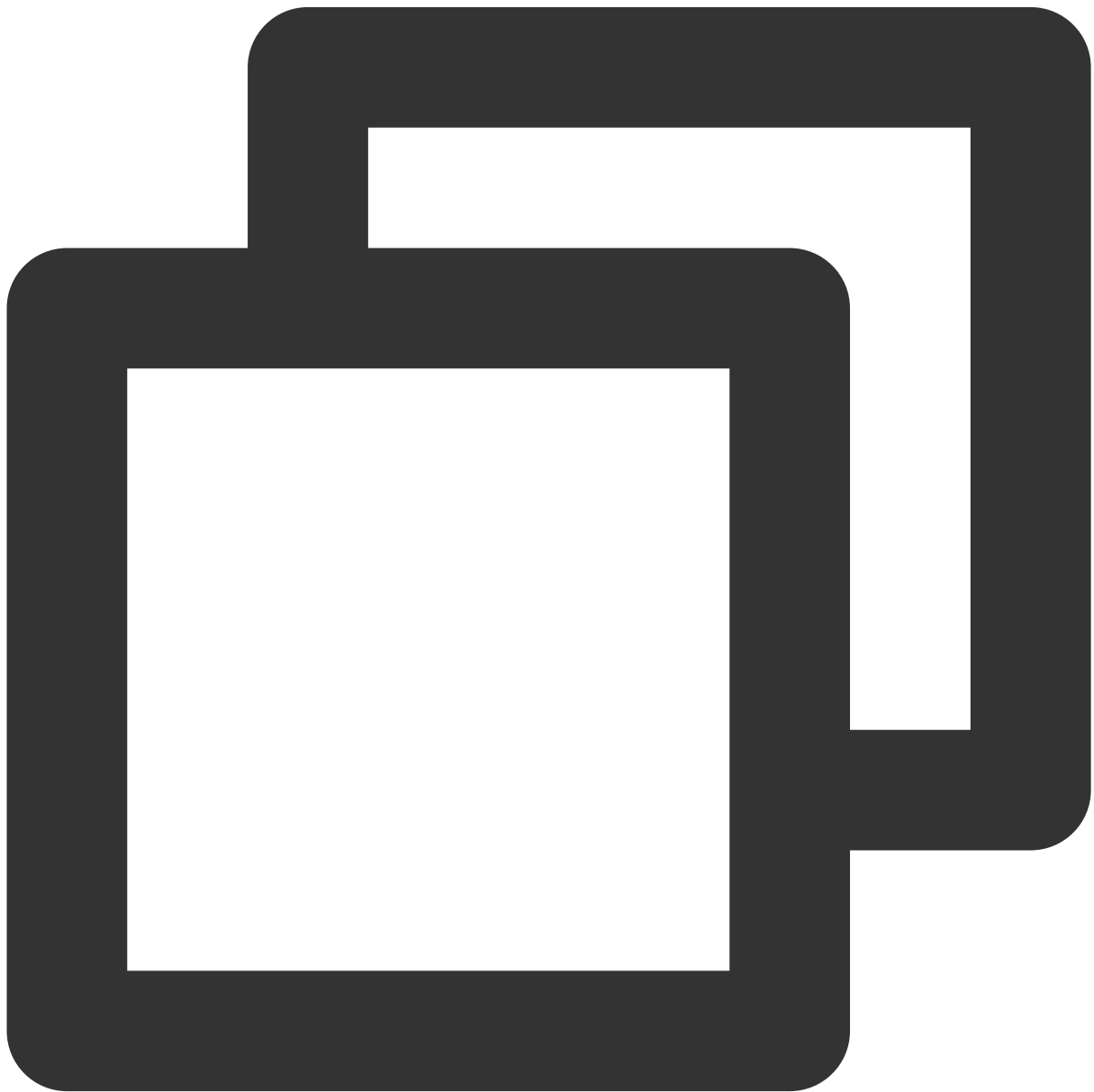
Parameter	Type	Description
seatIndex	int	The number of the seat to block/unblock.
isClose	boolean	<code>true</code> : Block; <code>false</code> : Unblock
callback	ActionCallback	The callback for the operation.

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

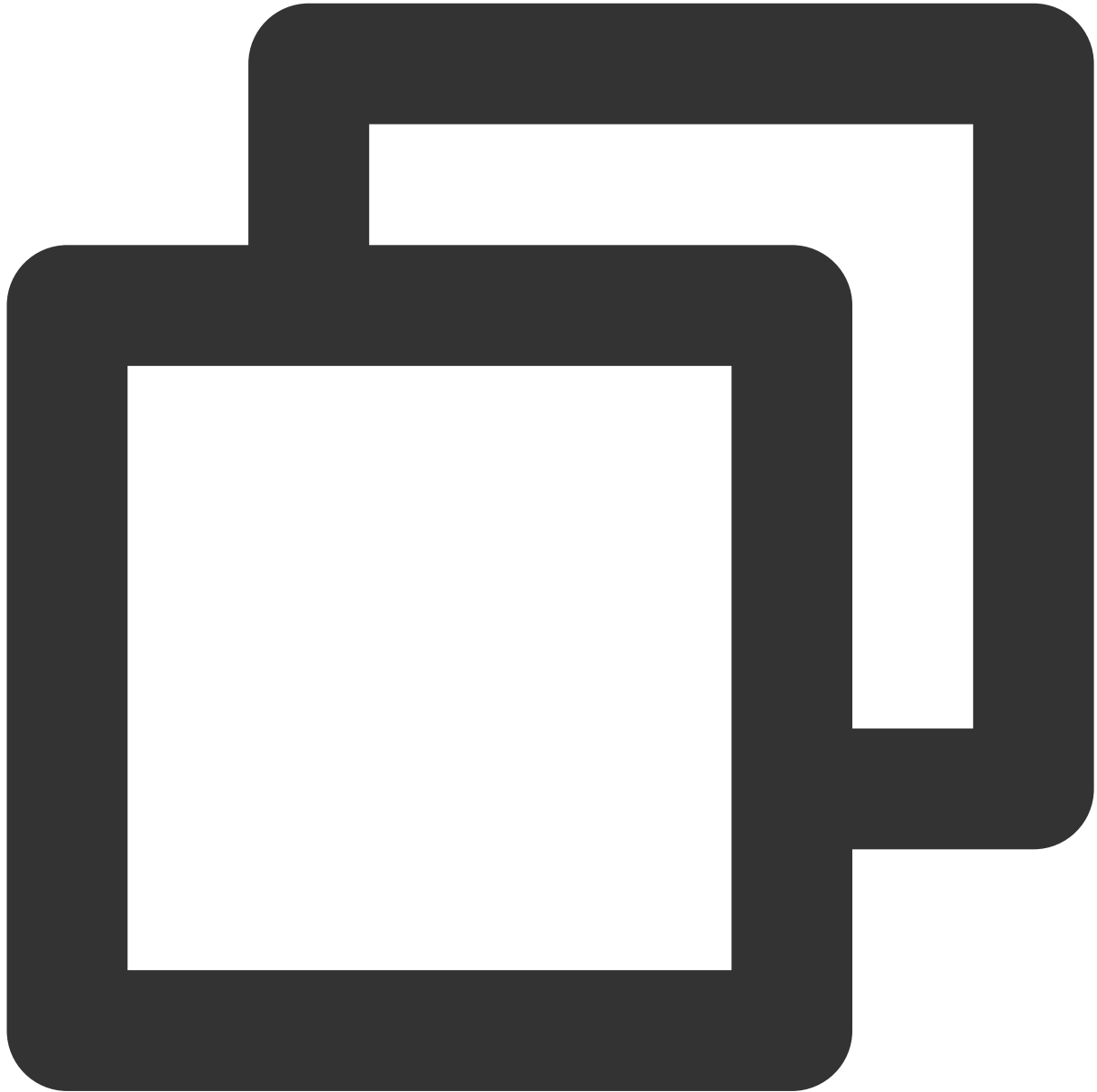
This API is used to start mic capturing.



```
- (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(quality:))
```

The parameters are described below:

Parameter	Type	Description
quality	int	The audio quality. For more information, see setAudioQuality() .

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 muteLocalAudio() .

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.



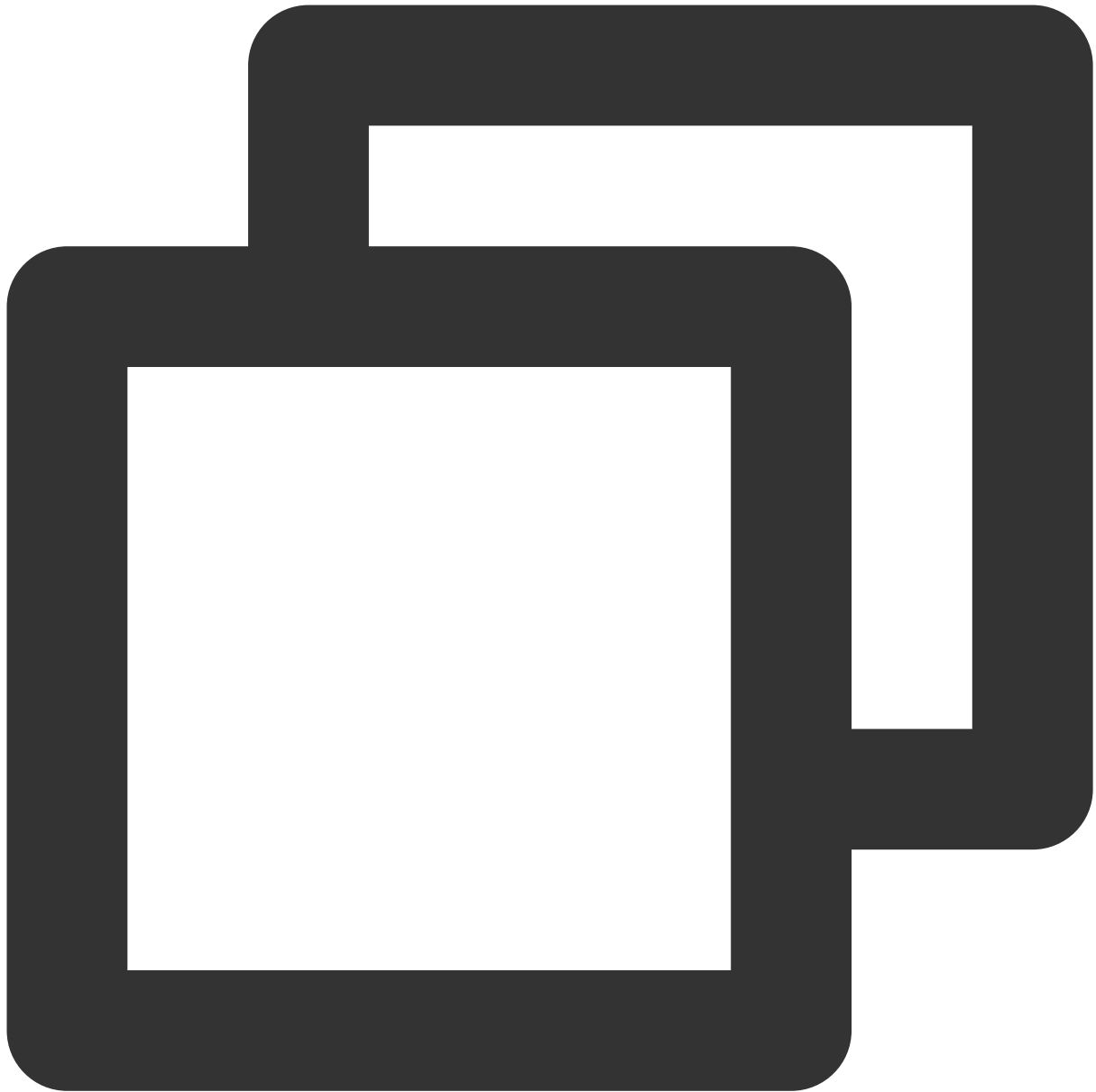
```
- (void)setAudioCaptureVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioCaptureVolume)
```

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.



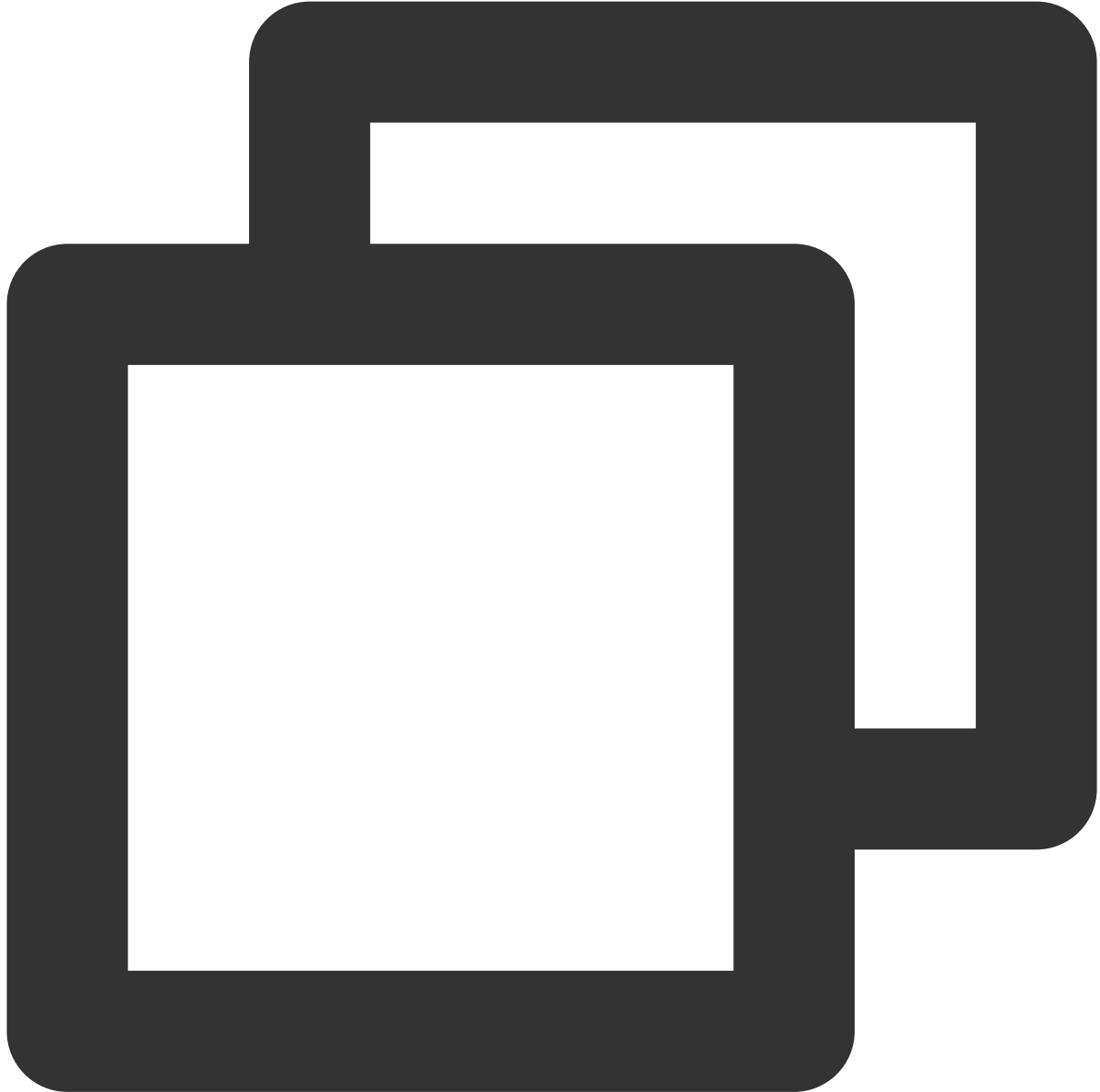
```
- (void)muteRemoteAudio:(NSString *)userId mute:(BOOL)mute NS_SWIFT_NAME(muteRemote
```

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.



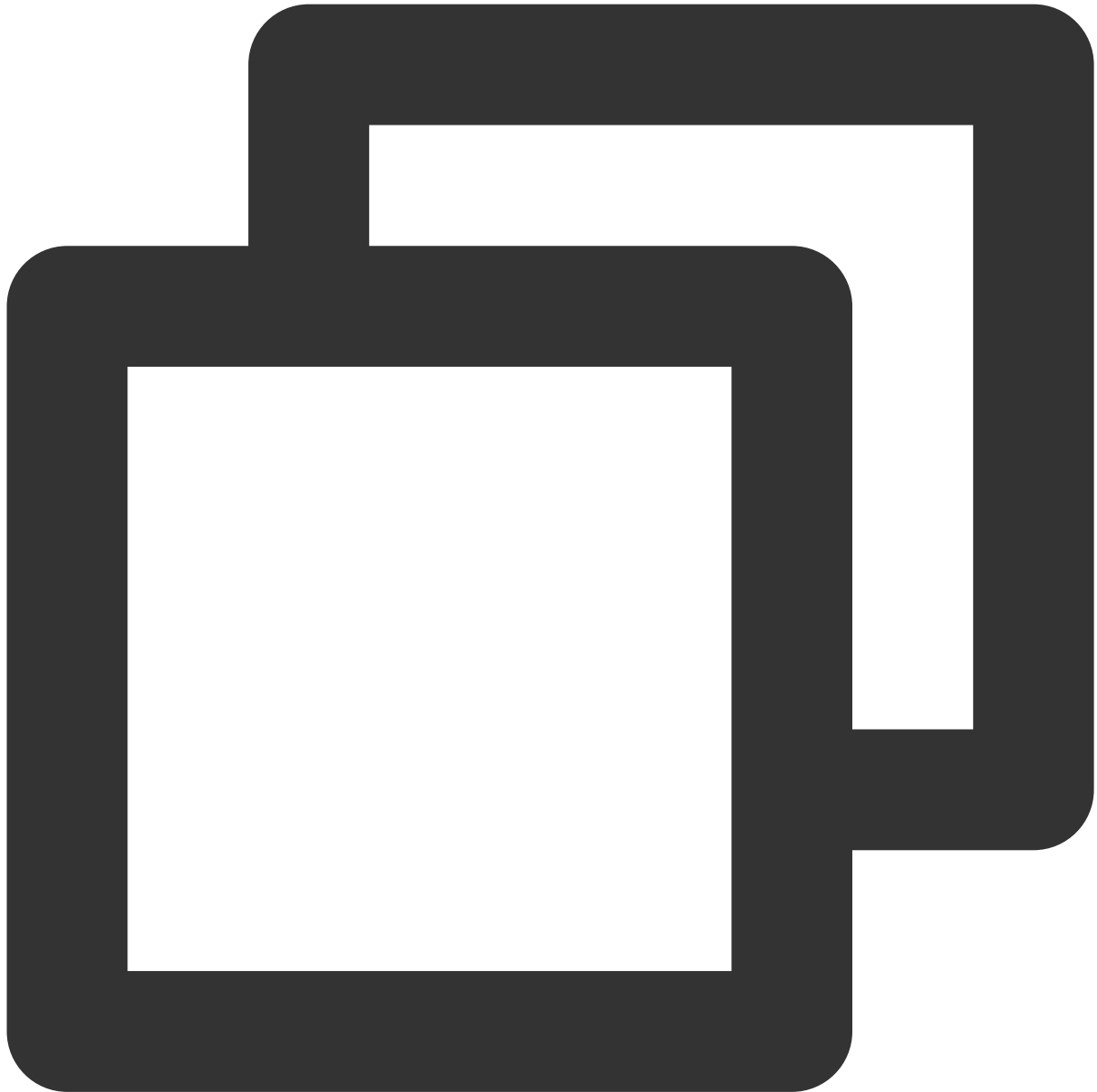
```
- (void)muteAllRemoteAudio:(BOOL)isMute NS_SWIFT_NAME(muteAllRemoteAudio(isMute:));
```

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))
```

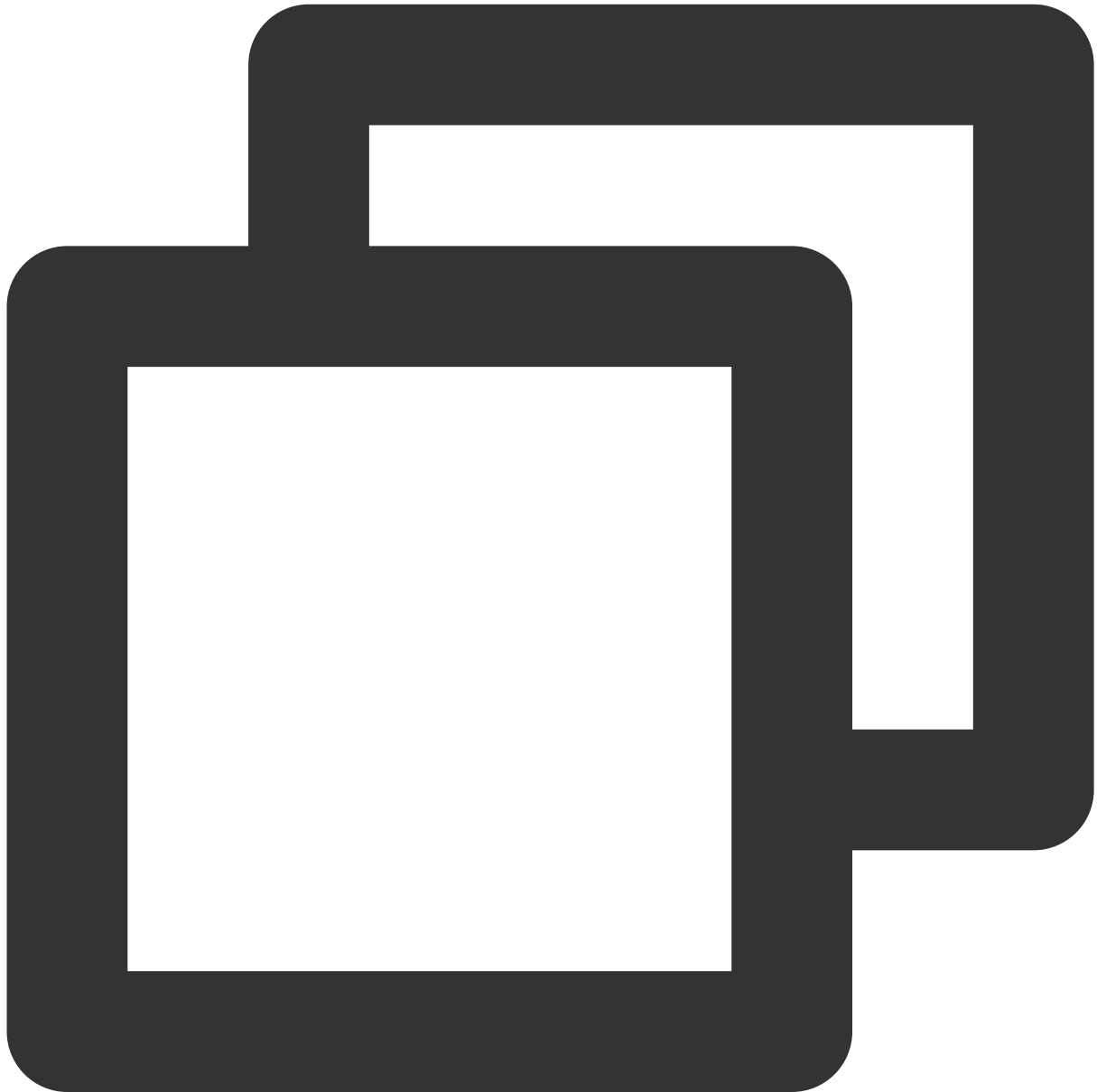
The parameters are described below:

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](#).

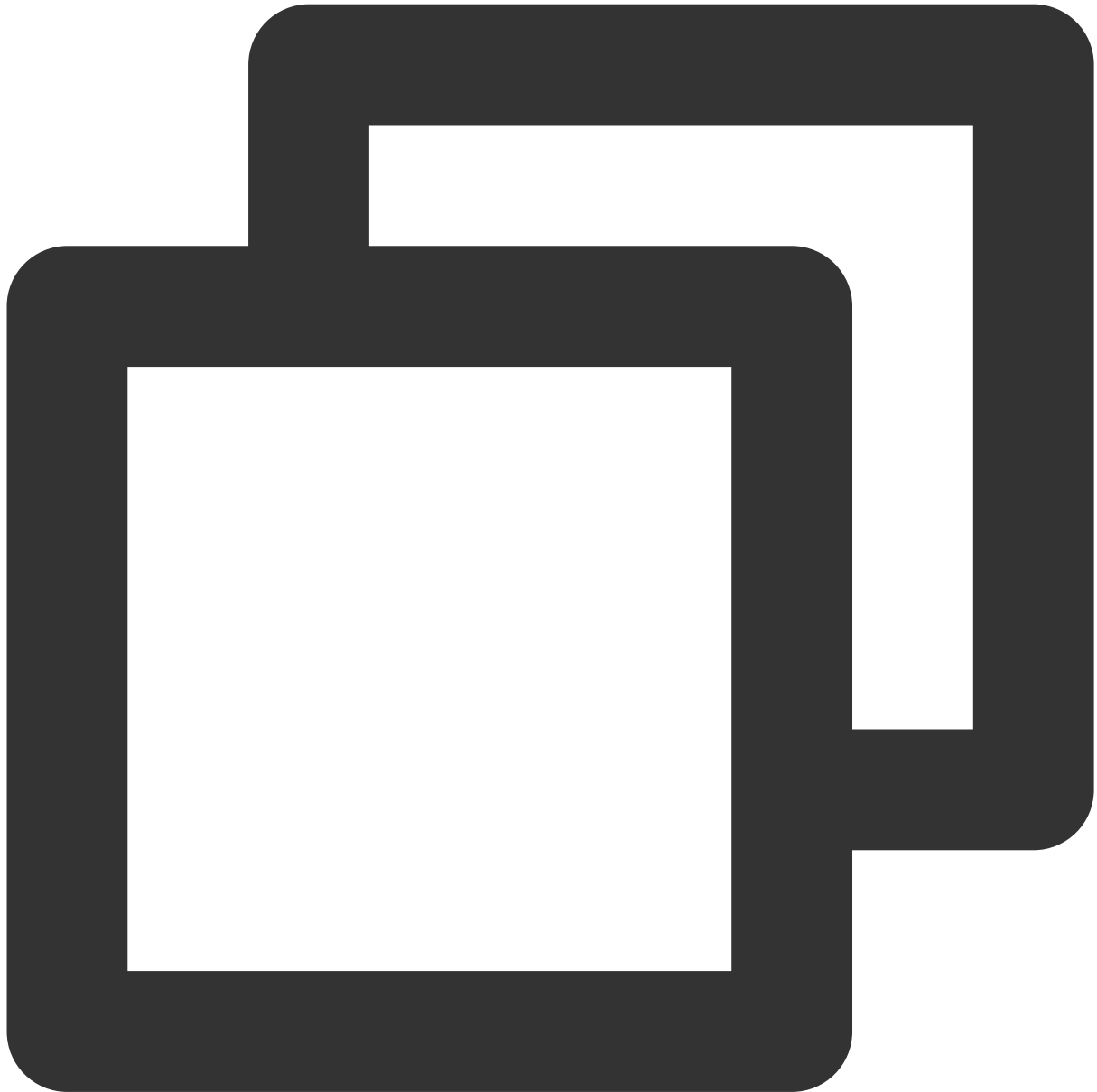


```
- (TXAudioEffectManager * _Nullable)getAudioEffectManager;
```

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.



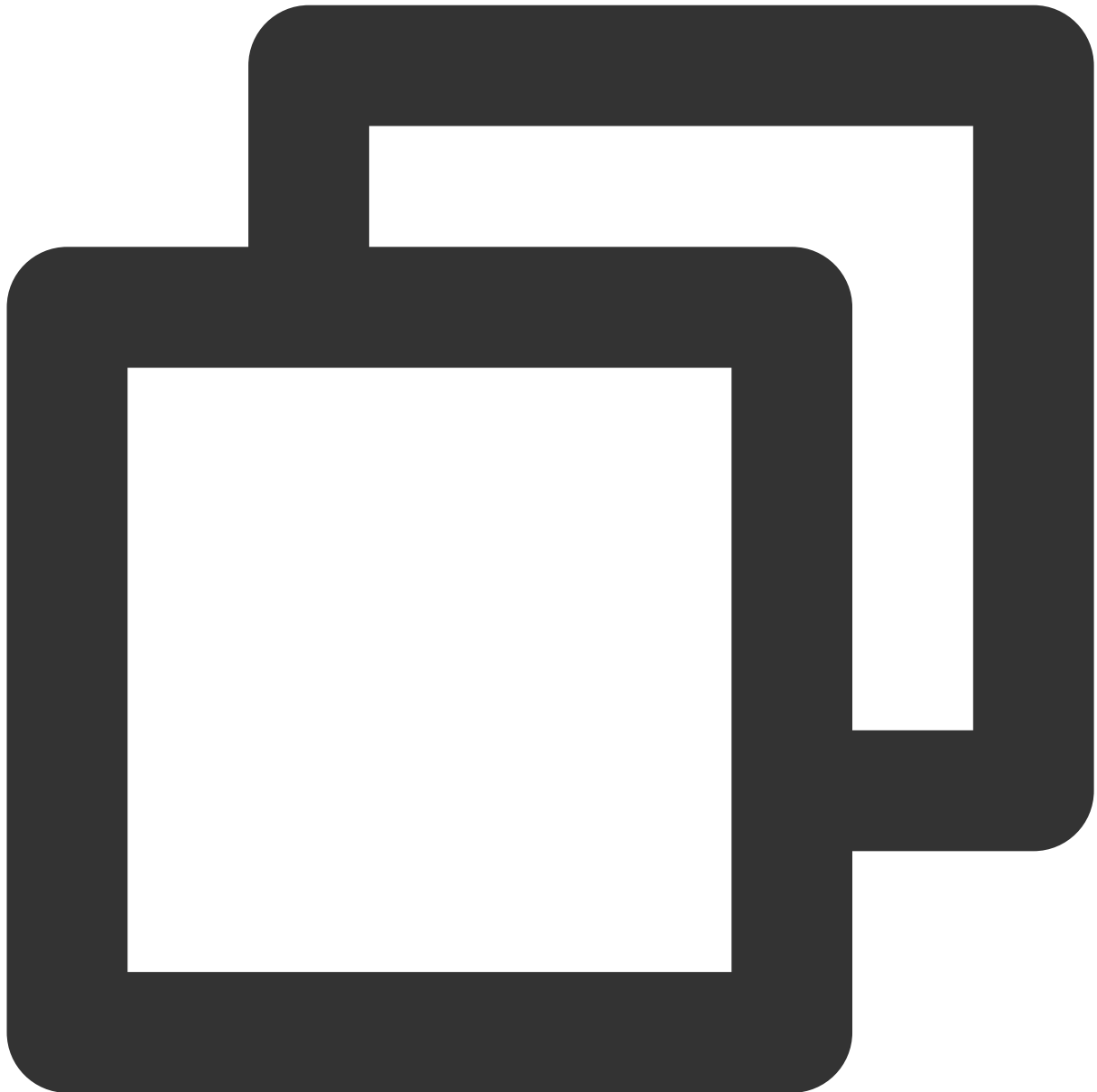
```
- (void)sendRoomTextMsg:(NSString *)message callback:(ActionCallback _Nullable)call
```

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

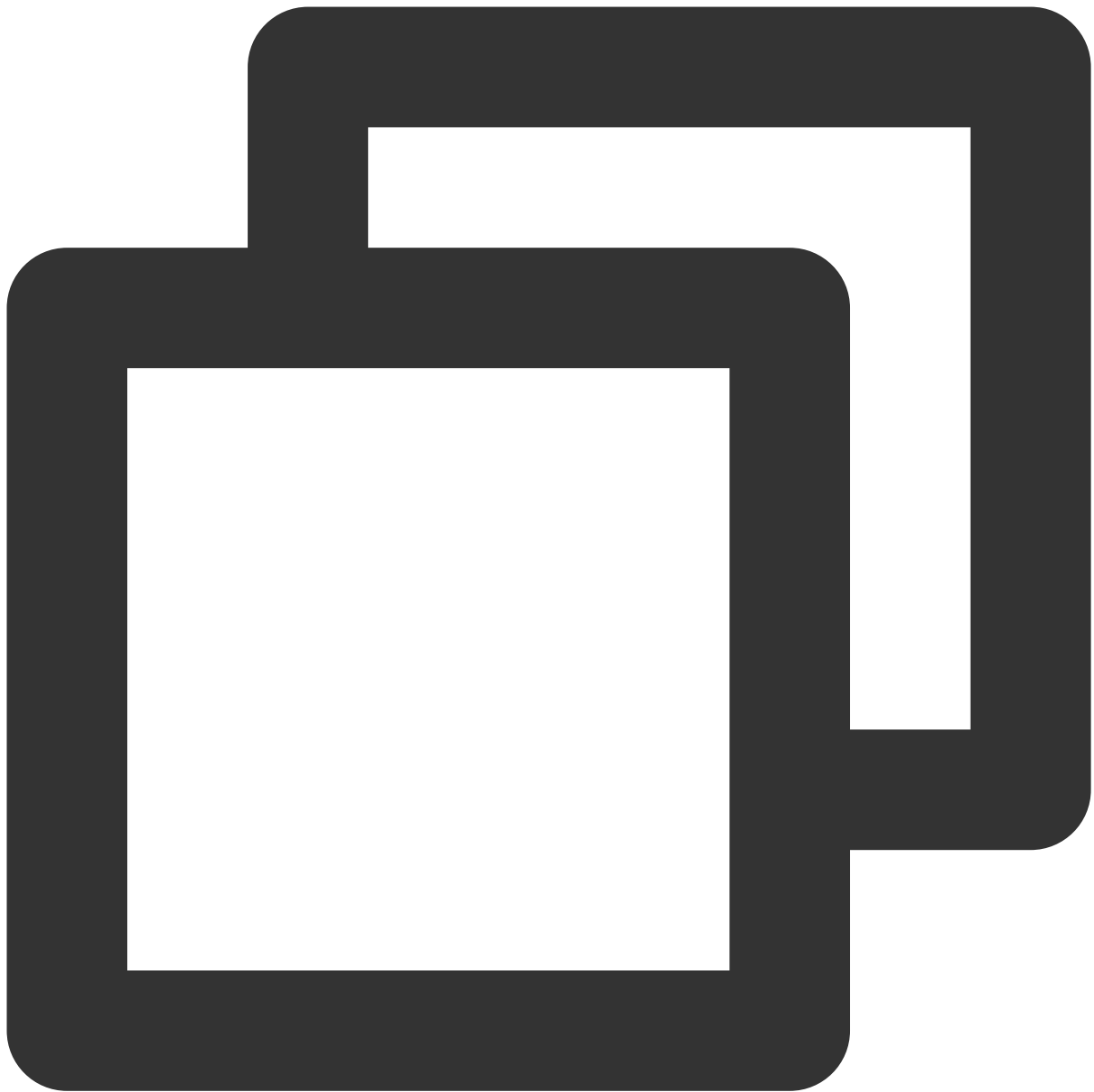
userId	String	The user ID of the invitee.
content	String	The content of the invitation.
callback	ActionCallback	The callback for the operation.

Response parameters:

Parameter	Type	Description
inviteId	String	The invitation ID.

acceptInvitation

This API is used to accept an invitation.



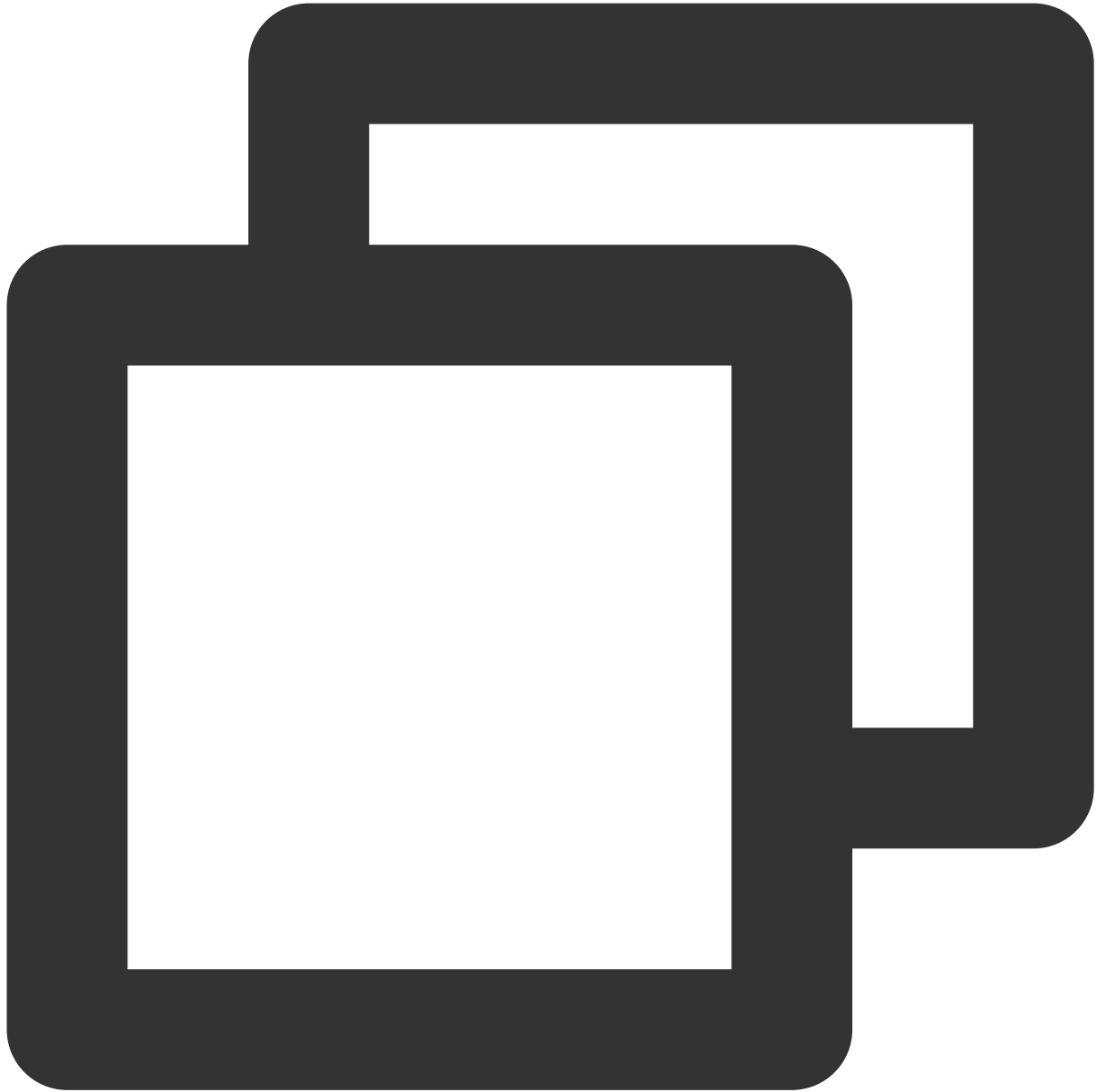
```
- (void)acceptInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)
```

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.



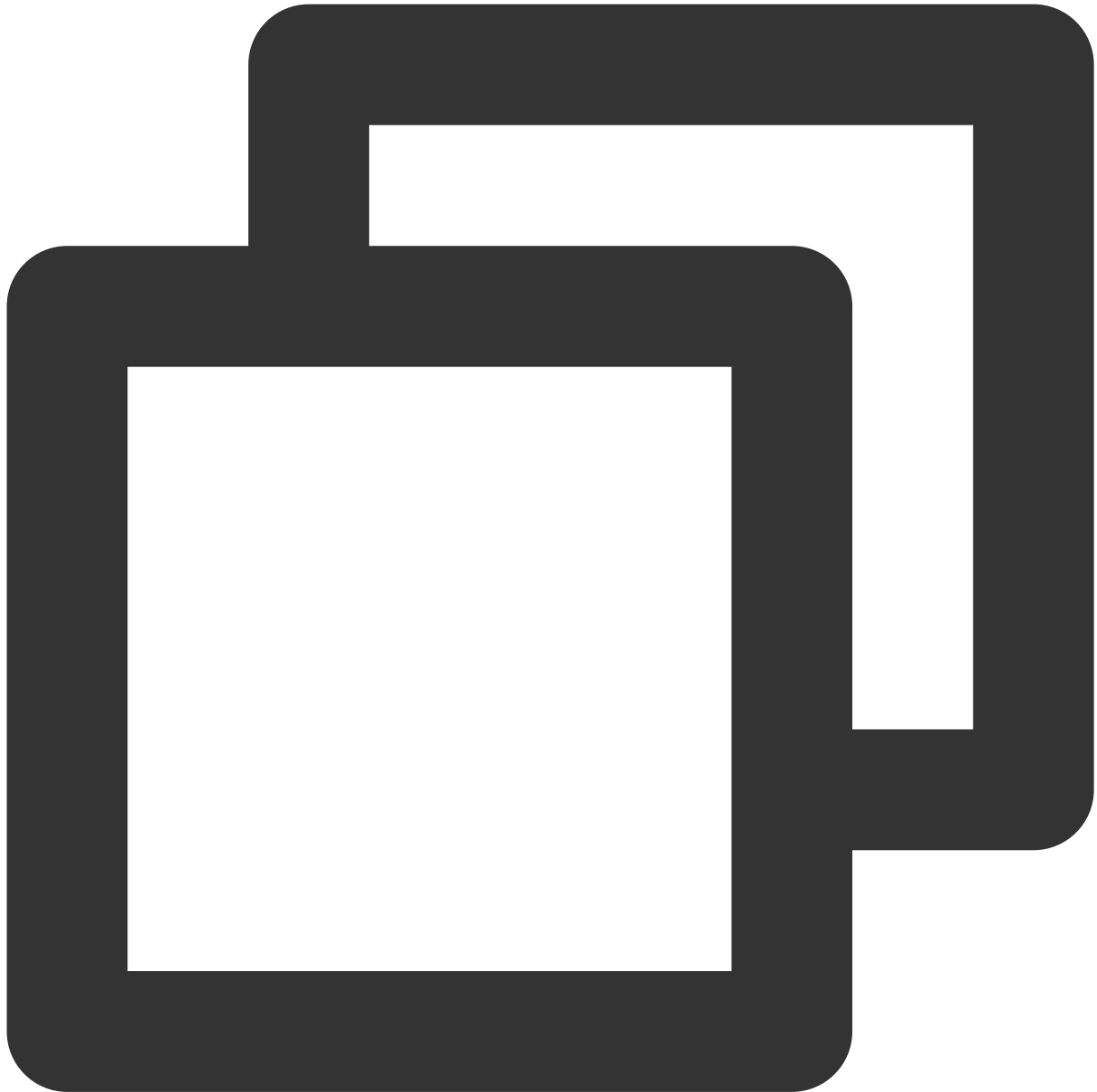
```
- (void)rejectInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)
```

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.



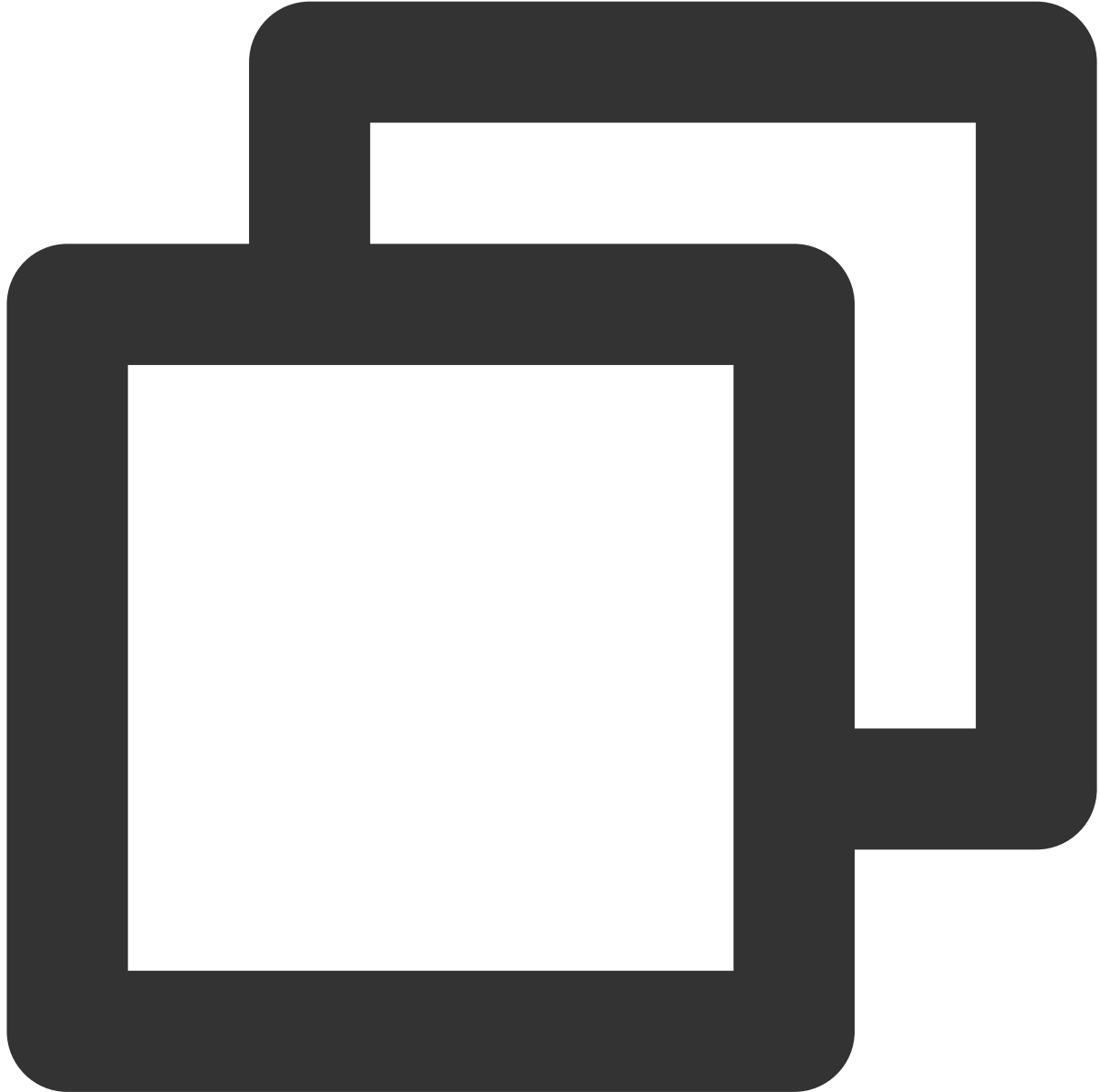
```
- (void)onError:(int)code  
    message:(NSString*)message  
NS_SWIFT_NAME(onError(code:message:));
```

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:));
```

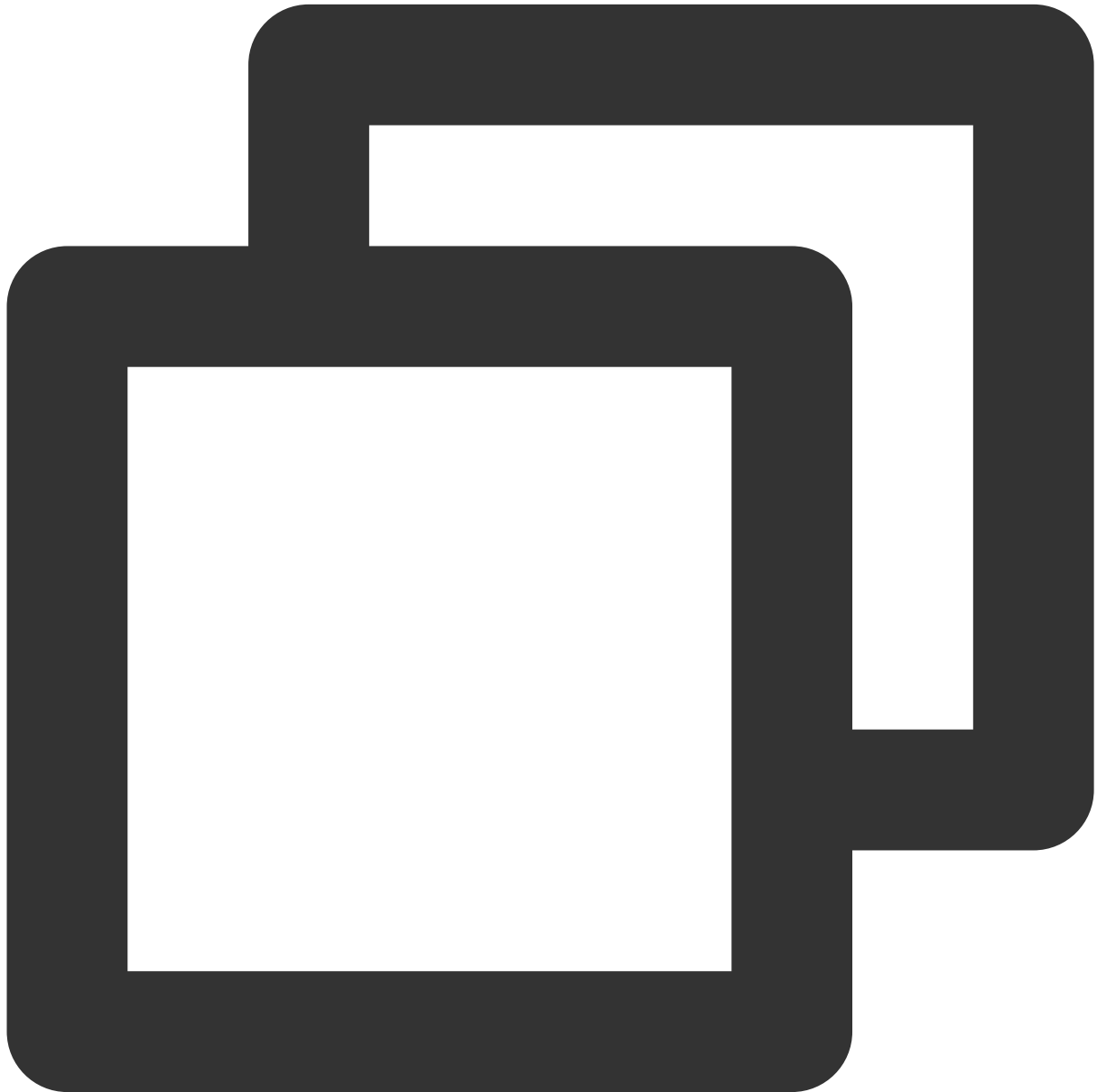
The parameters are described below:

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.



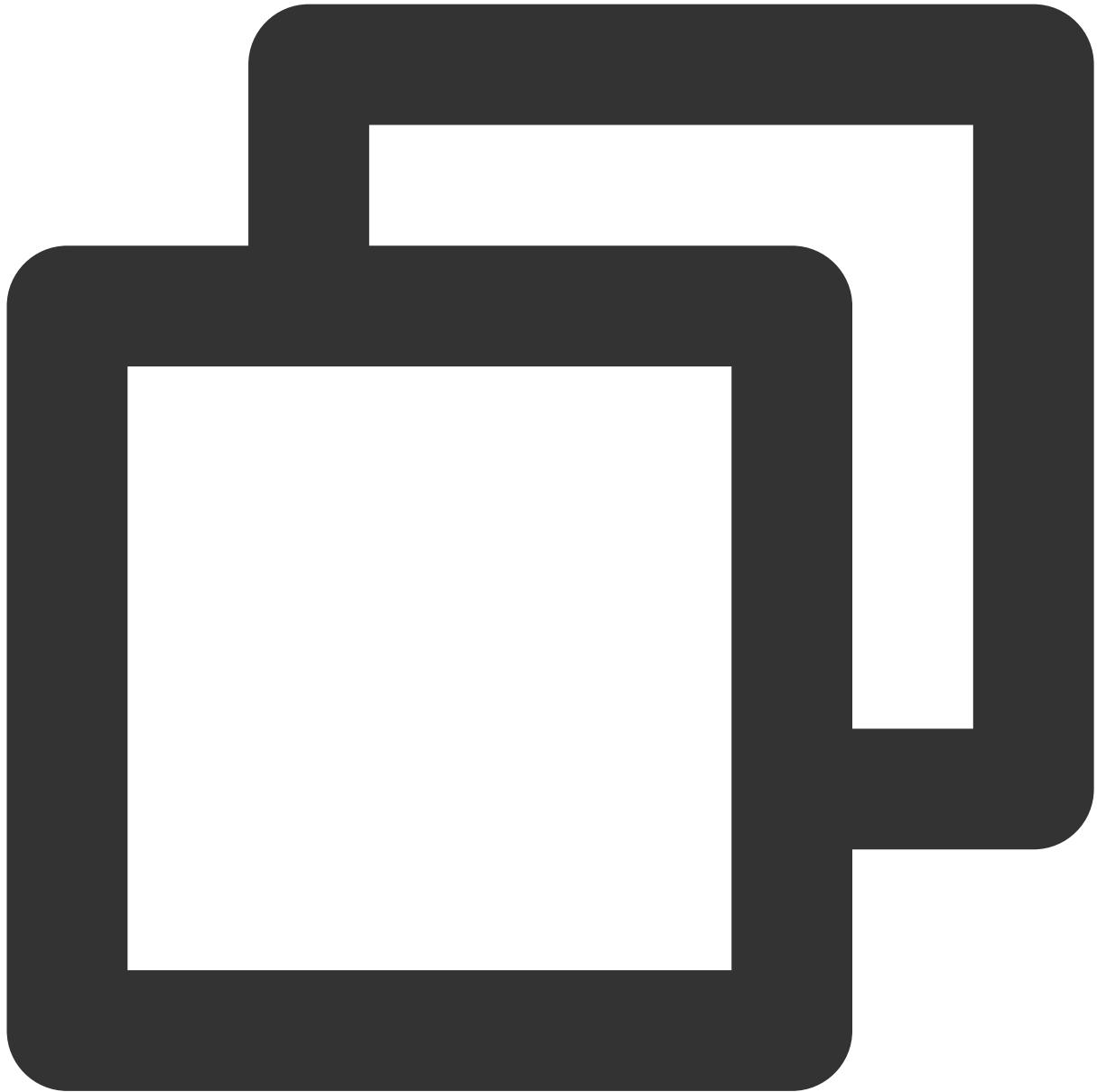
```
- (void)onRoomDestroy:(NSString *)message  
NS_SWIFT_NAME(onRoomDestroy(message:));
```

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.



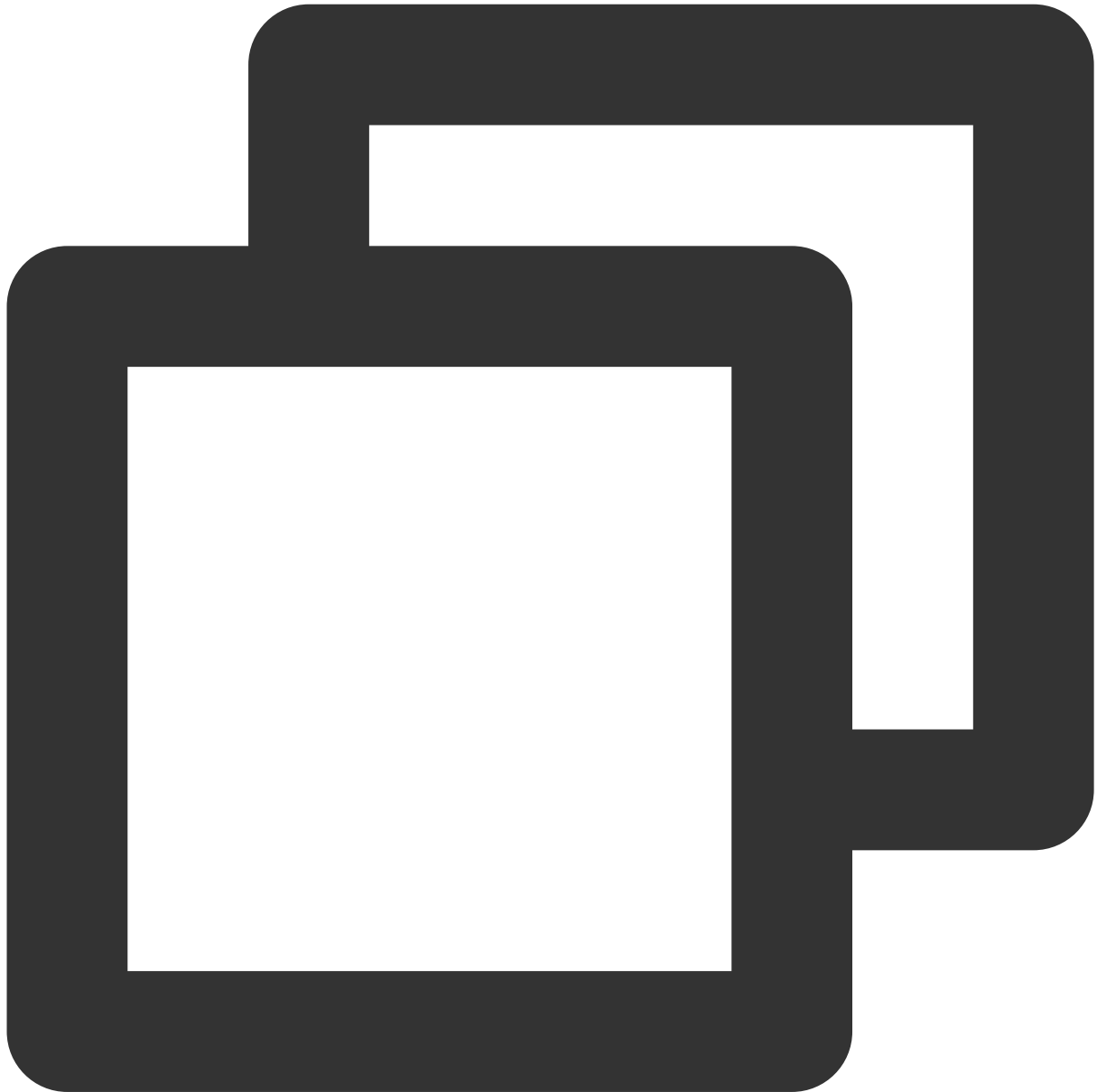
```
- (void)onRoomInfoChange:(KaraokeInfo *)roomInfo  
NS_SWIFT_NAME(onRoomInfoChange(roomInfo:));
```

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:));
```

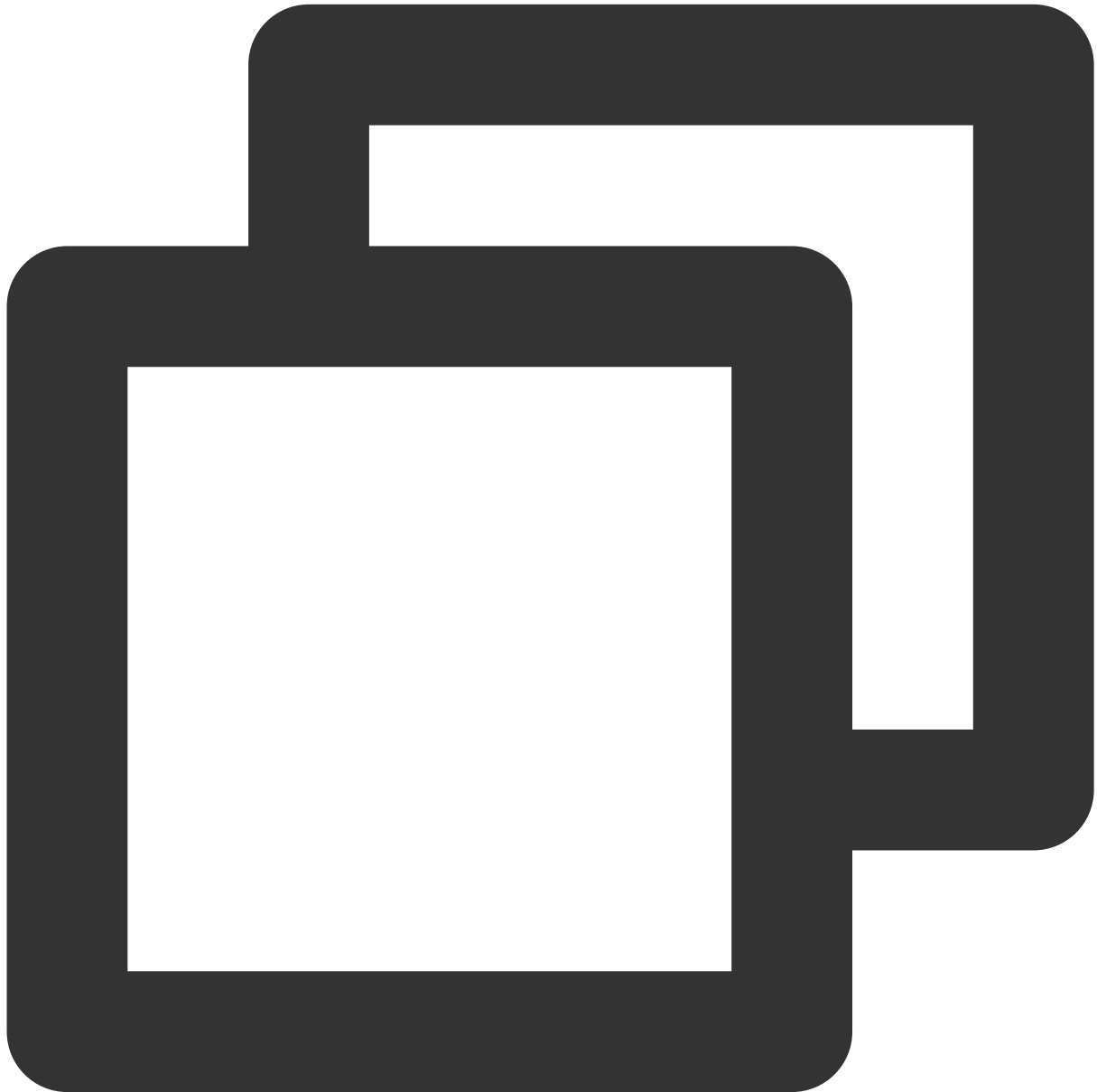
The parameters are described below:

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:));
```

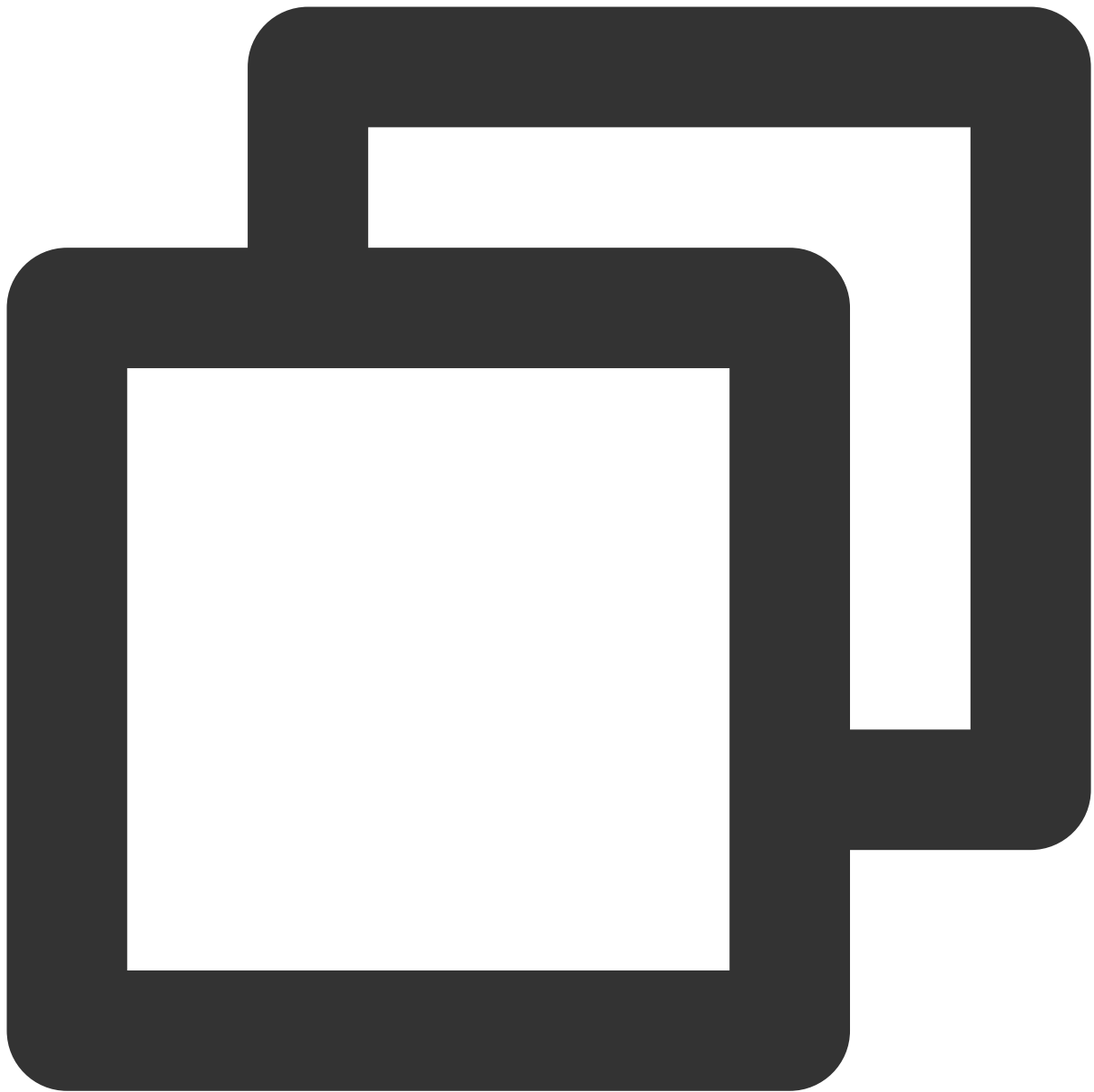

The parameters are described below:

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.



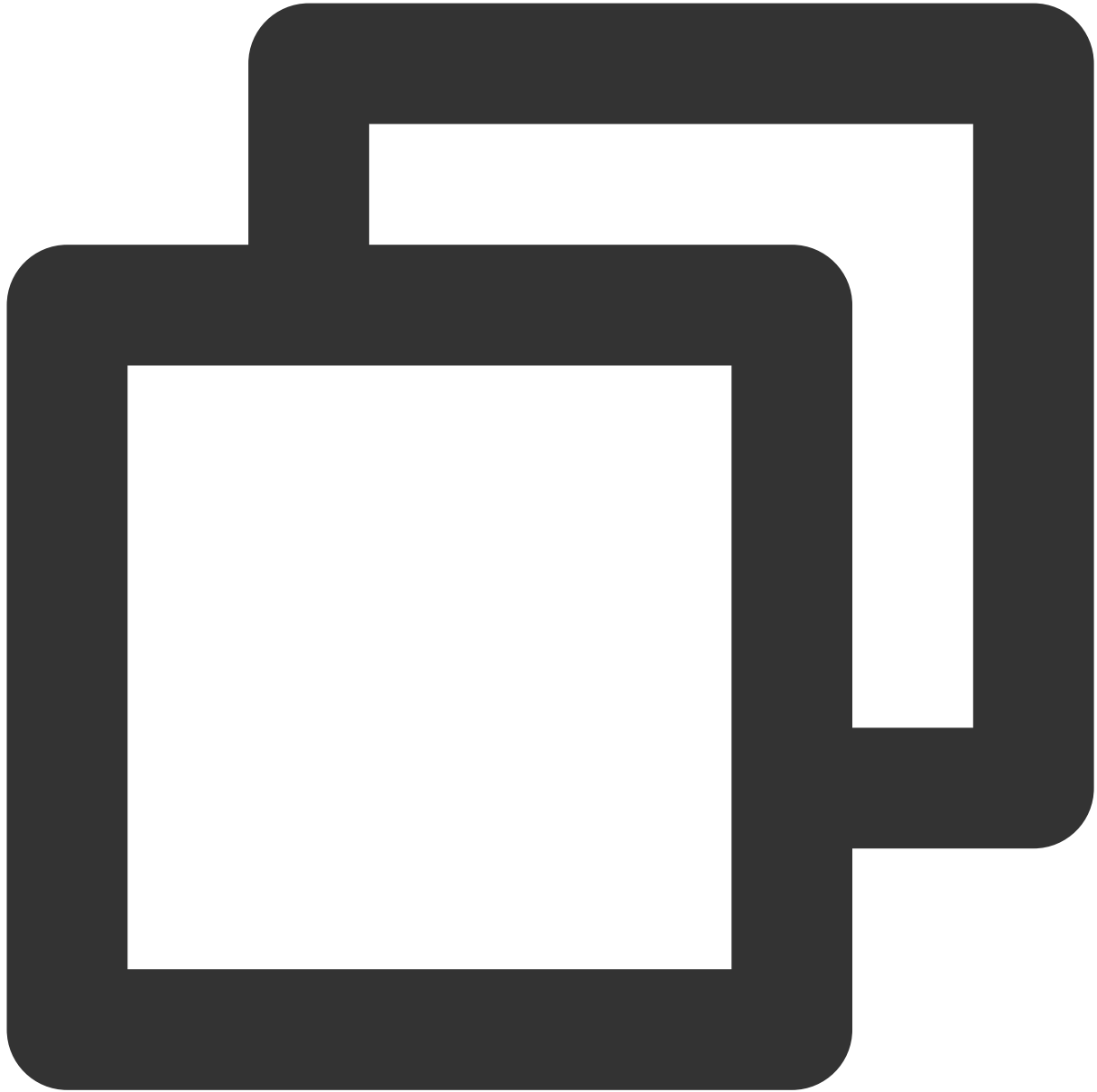
```
- (void)onSeatInfoChange:(NSArray<KaraokeSeatInfo *> *)seatInfoList  
NS_SWIFT_NAME(onSeatListChange(seatInfoList:));
```

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

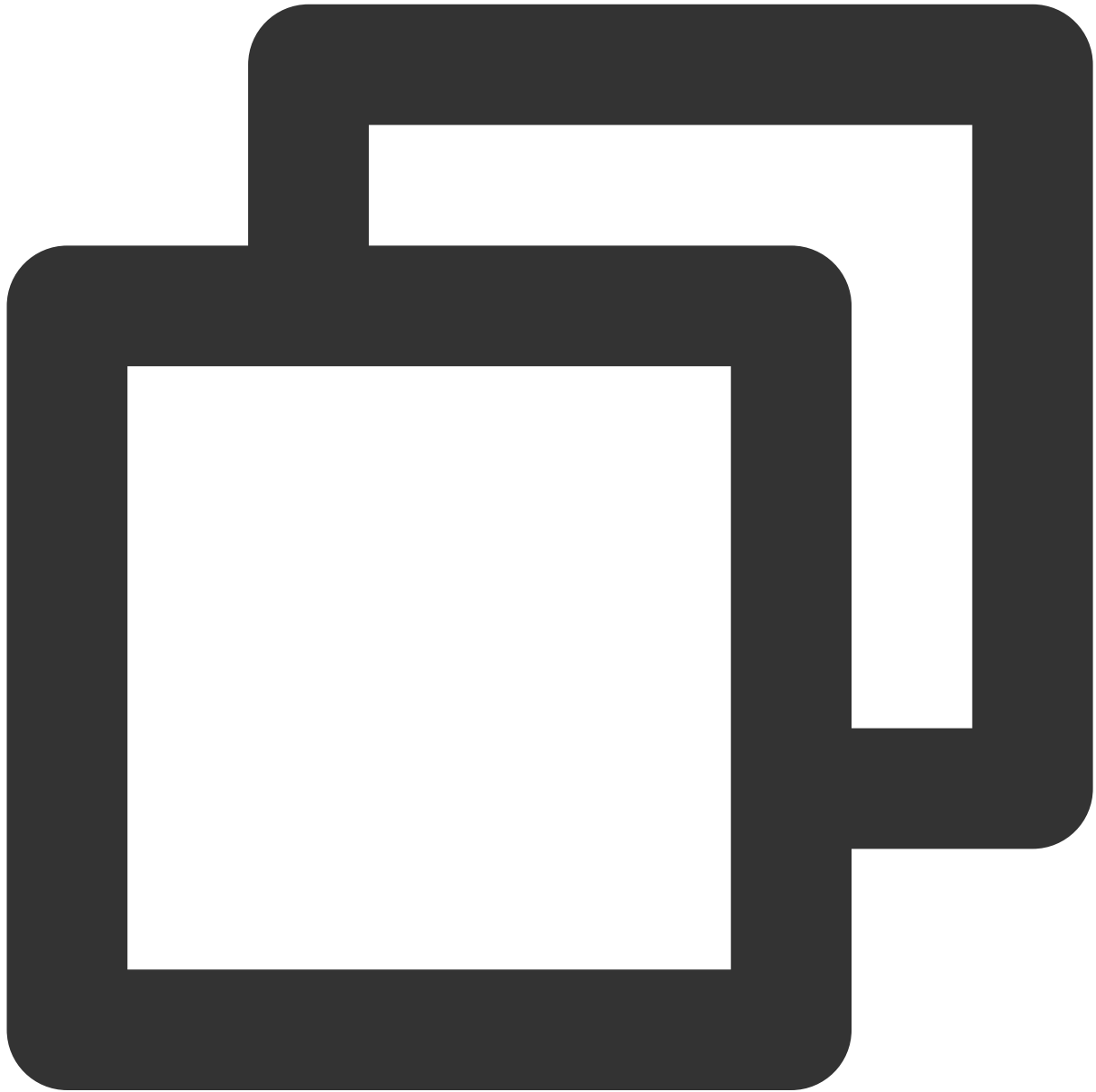
user

UserInfo

Details of the user who took the seat

onAnchorLeaveSeat

A speaker became a listener or was moved to listeners by the room owner.



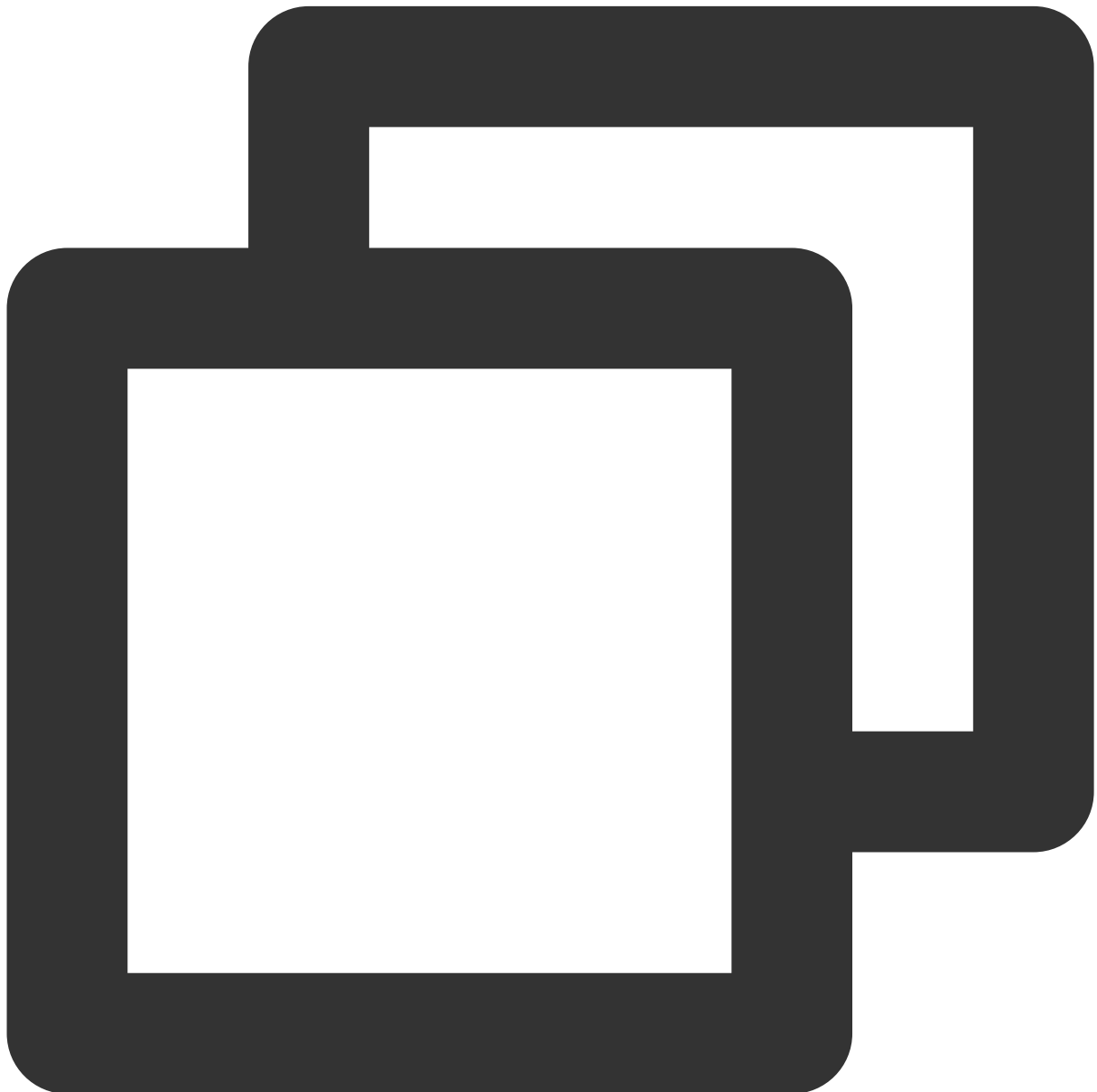
```
- (void)onAnchorLeaveSeat:(NSInteger) index
    user:(KaraokeUserInfo *)user
NS_SWIFT_NAME(onAnchorLeaveSeat(index:user:));
```

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
```

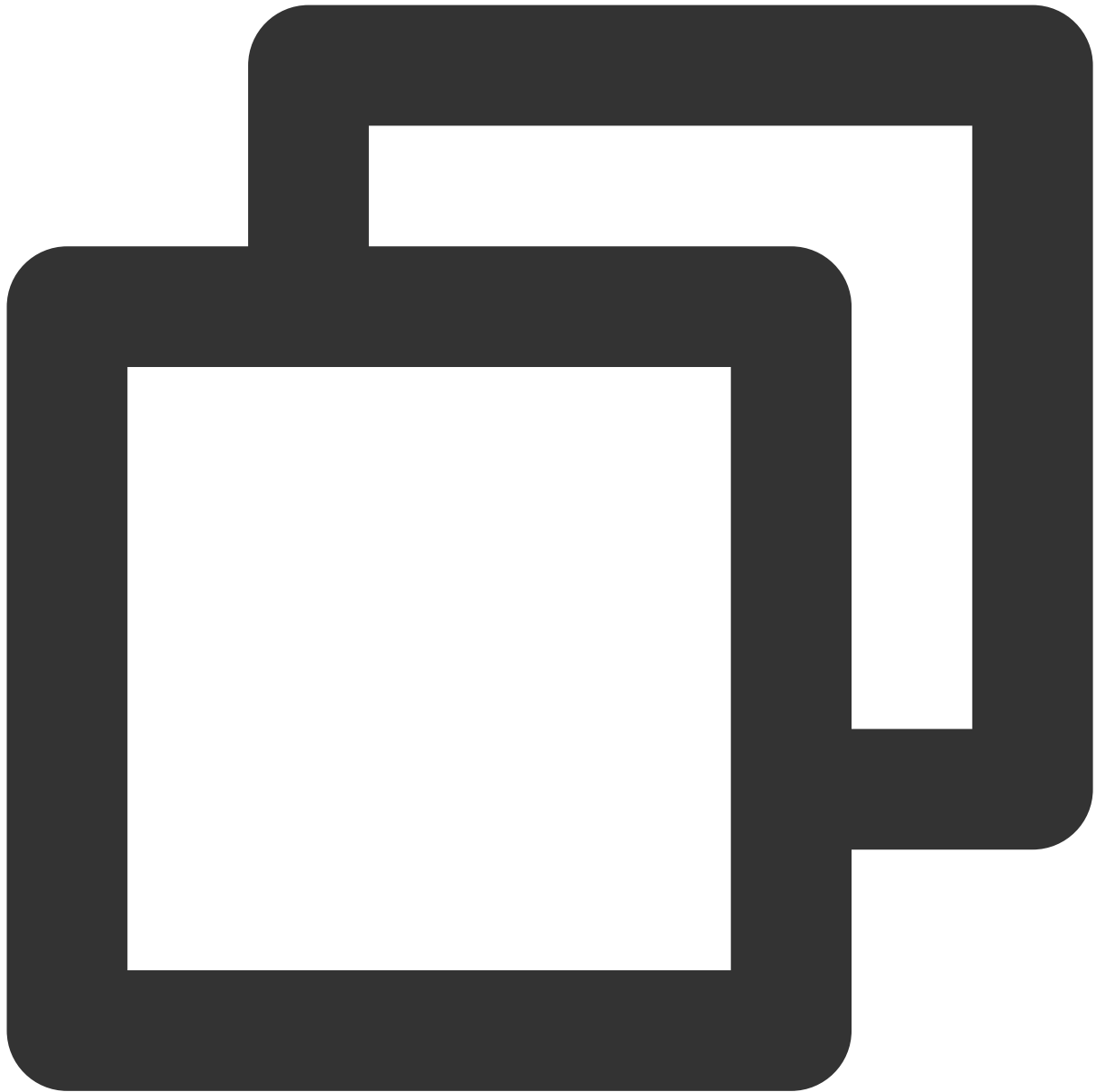
```
isMute: (BOOL) isMute
NS_SWIFT_NAME (onSeatMute (index:isMute:));
```

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:));
```

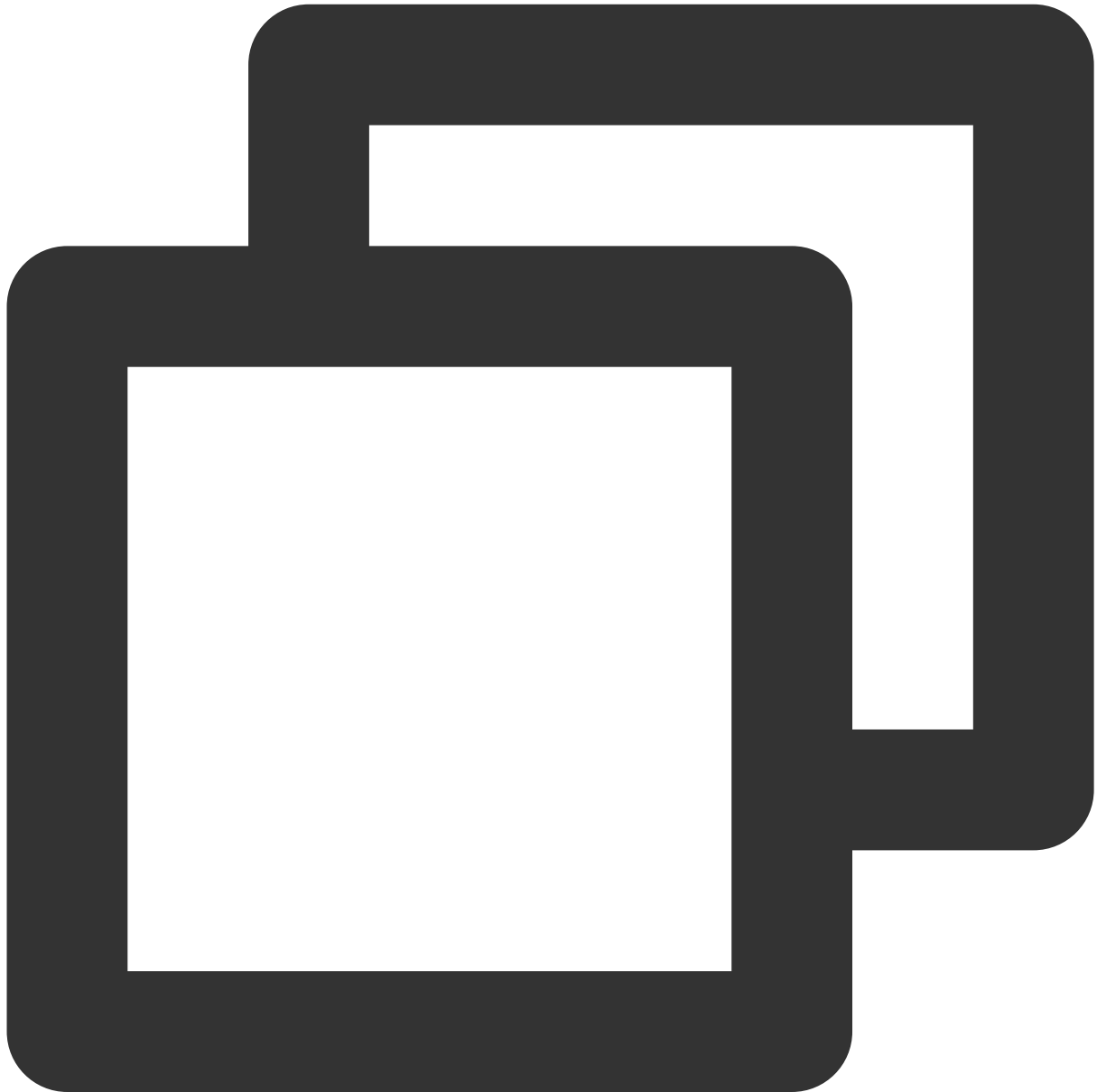
The parameters are described below:

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:));
```

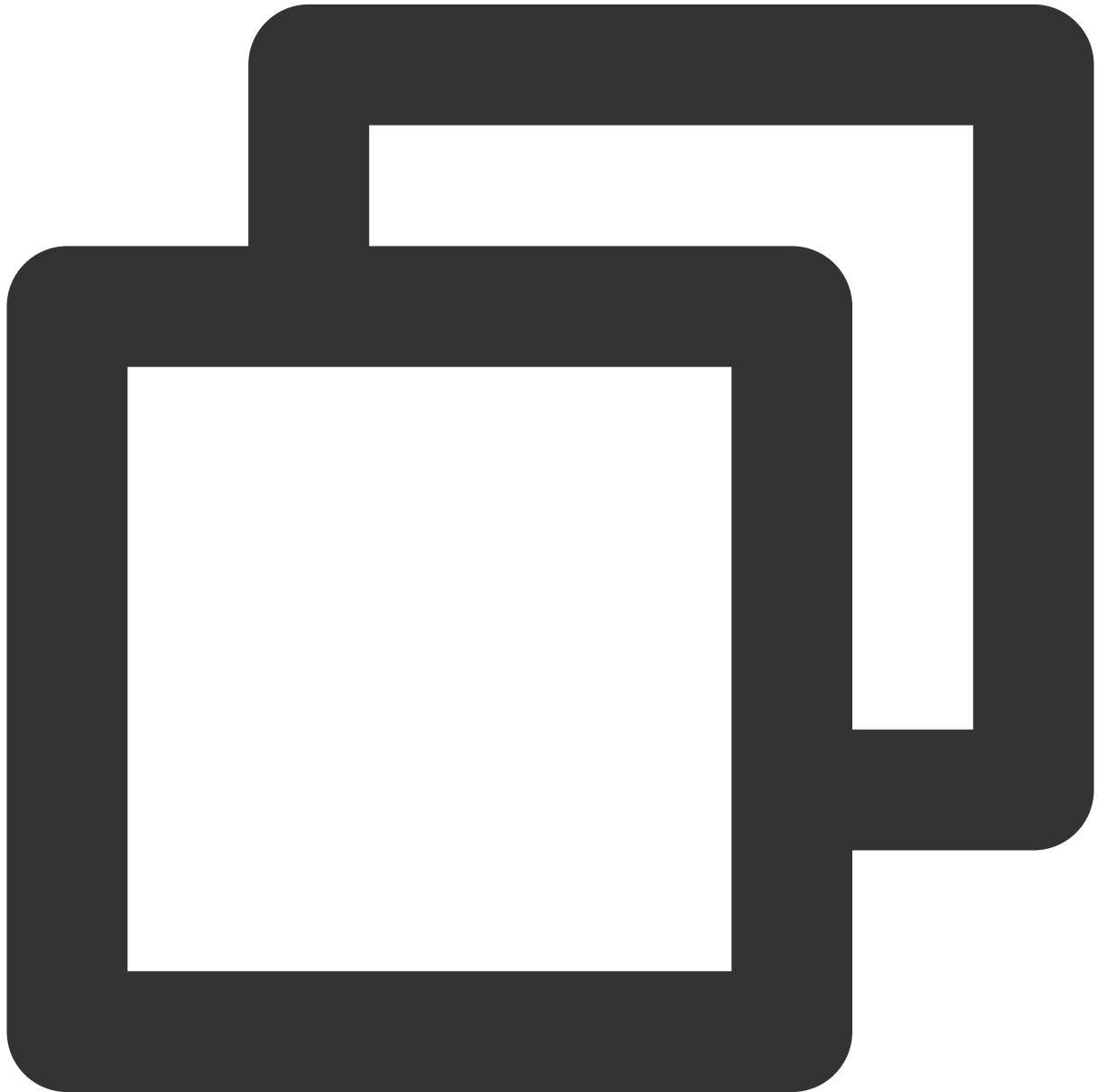
The parameters are described below:

--	--	--

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.



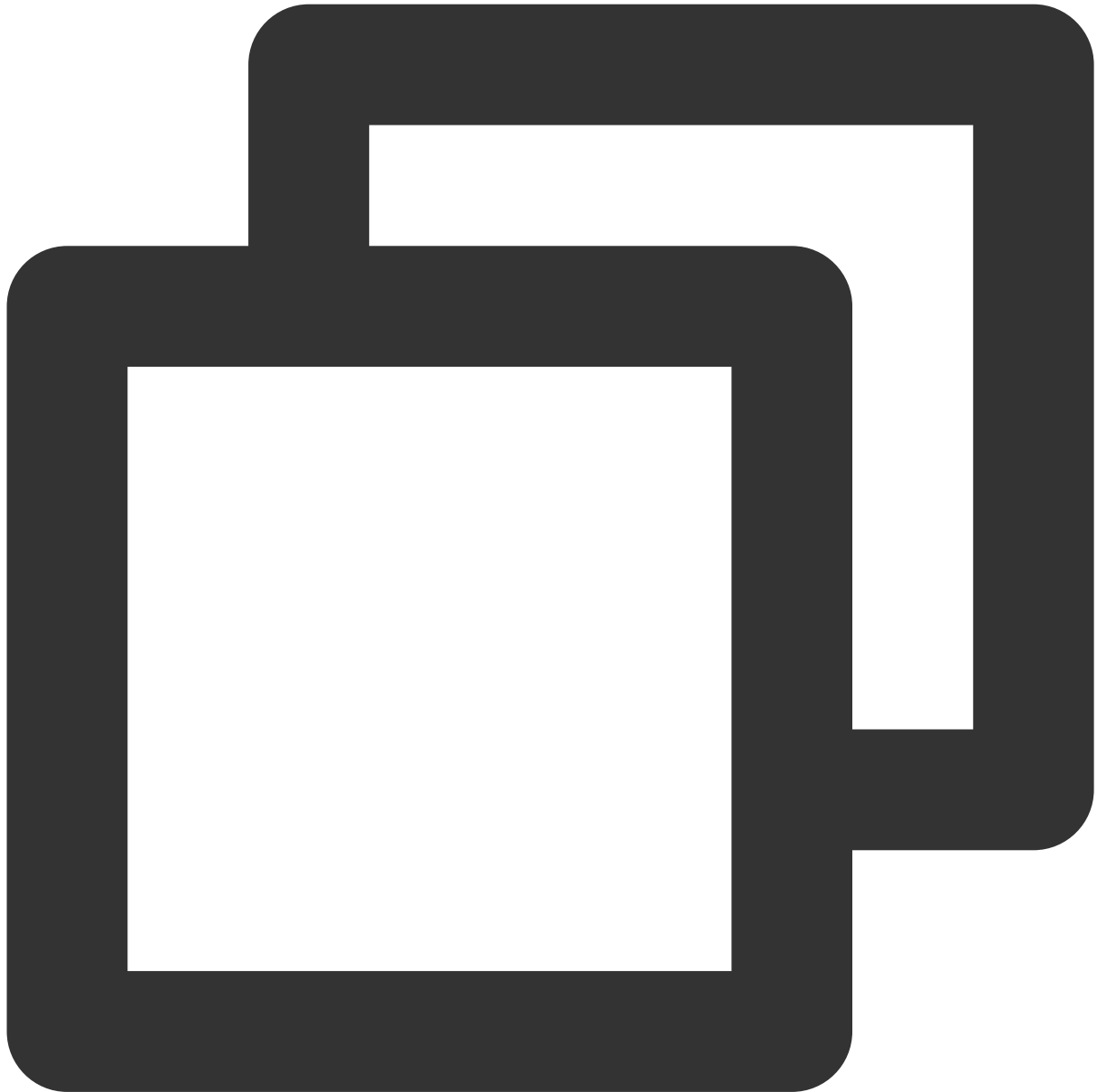
```
- (void)onRecvRoomTextMsg:(NSString *)message
    userInfo:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomTextMsg(message:userInfo:));
```

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:));
```

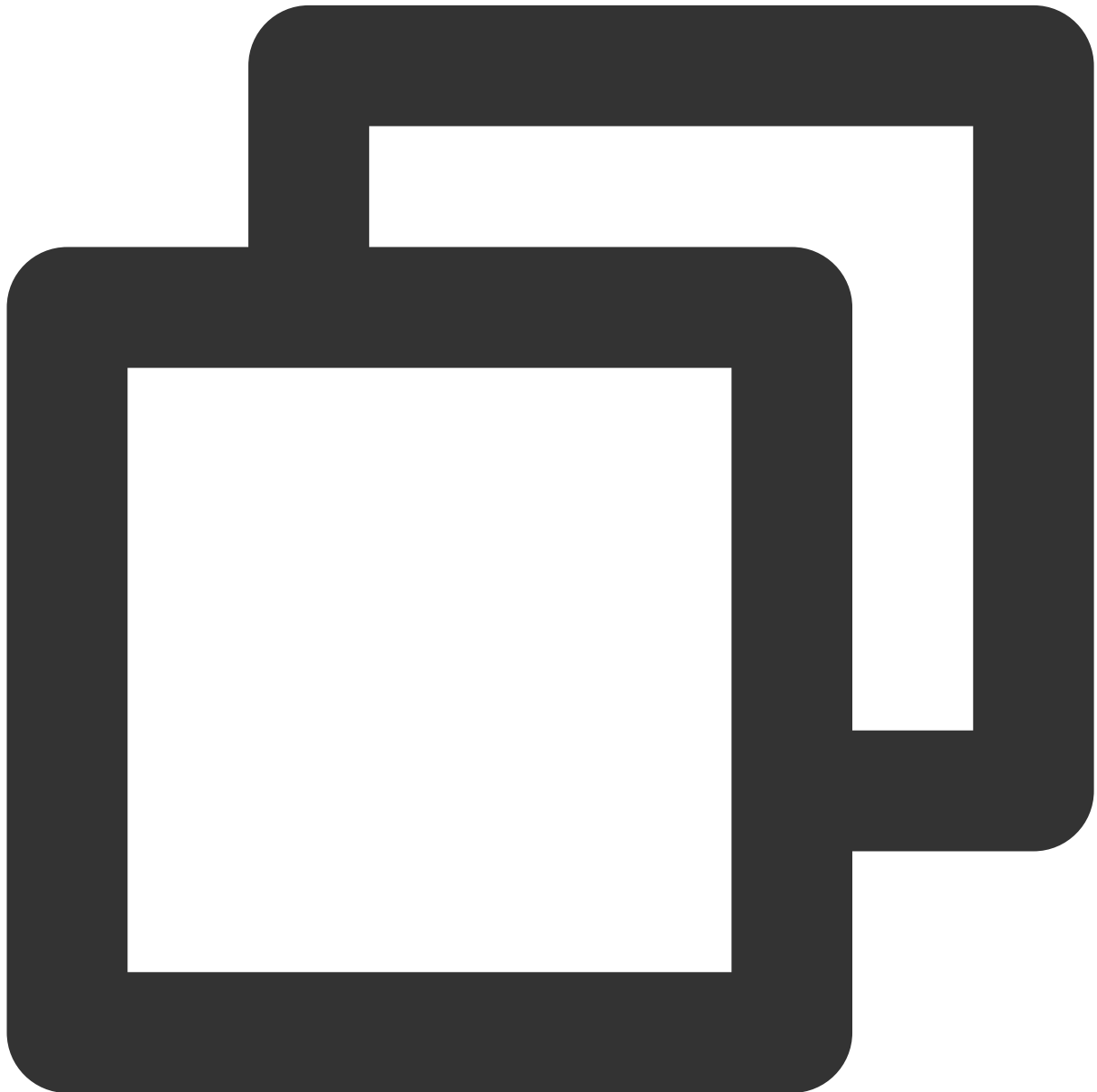
The parameters are described below:

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
```

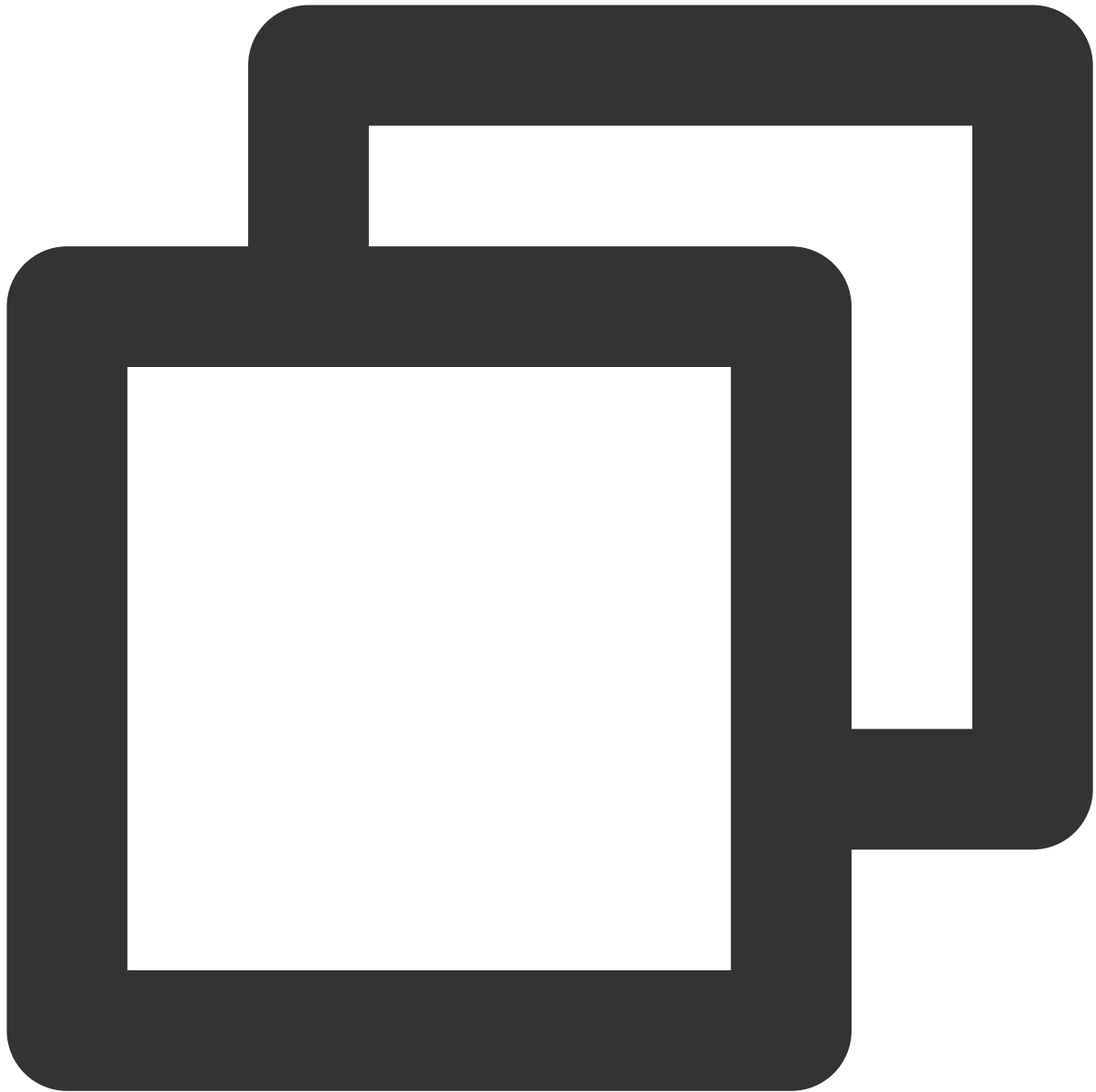
```
invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeAccepted(identifier:invitee:));
```

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



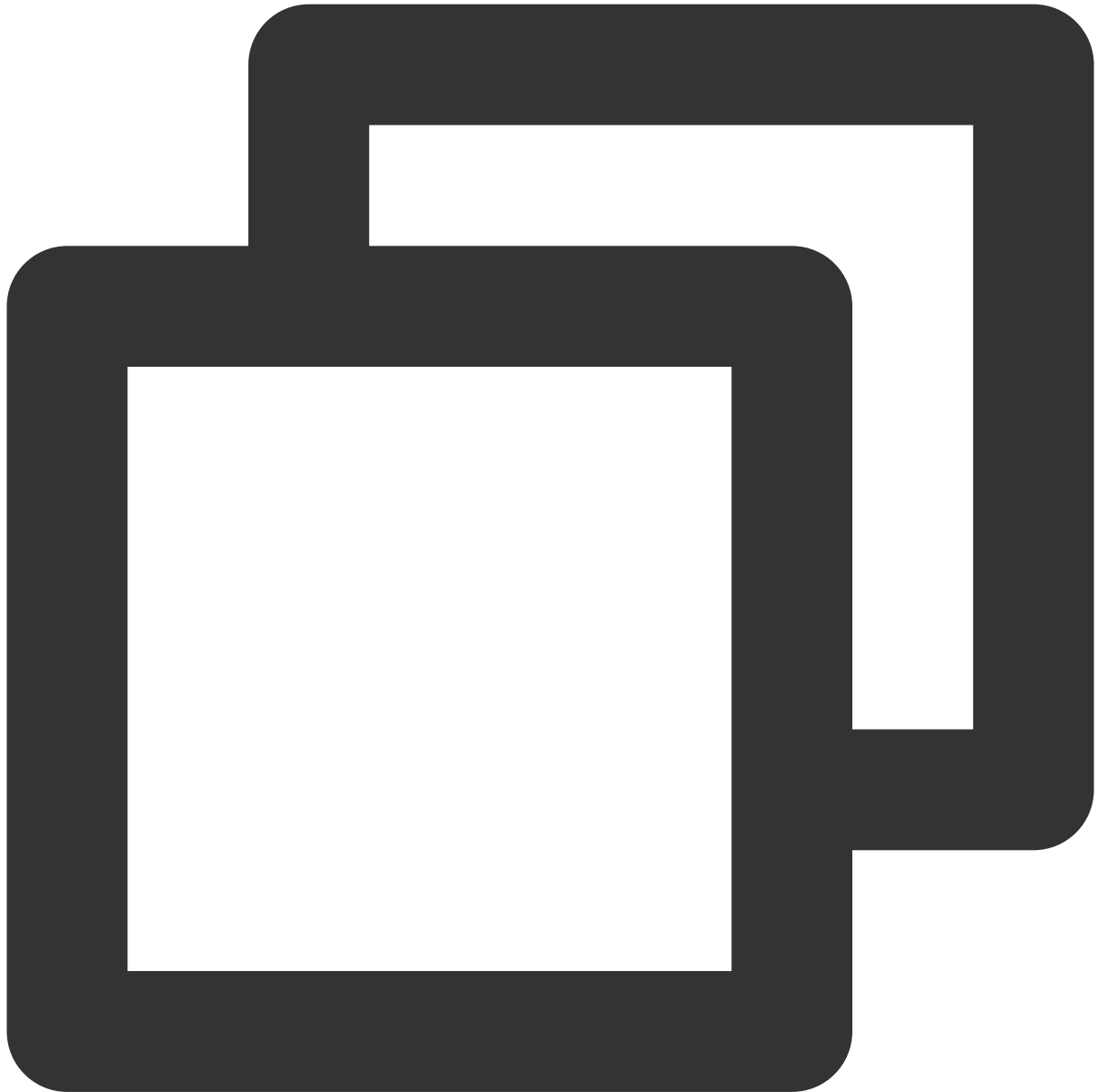
```
- (void)onInviteeRejected:(NSString *)identifier
    invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeRejected(identifier:invitee:));
```

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

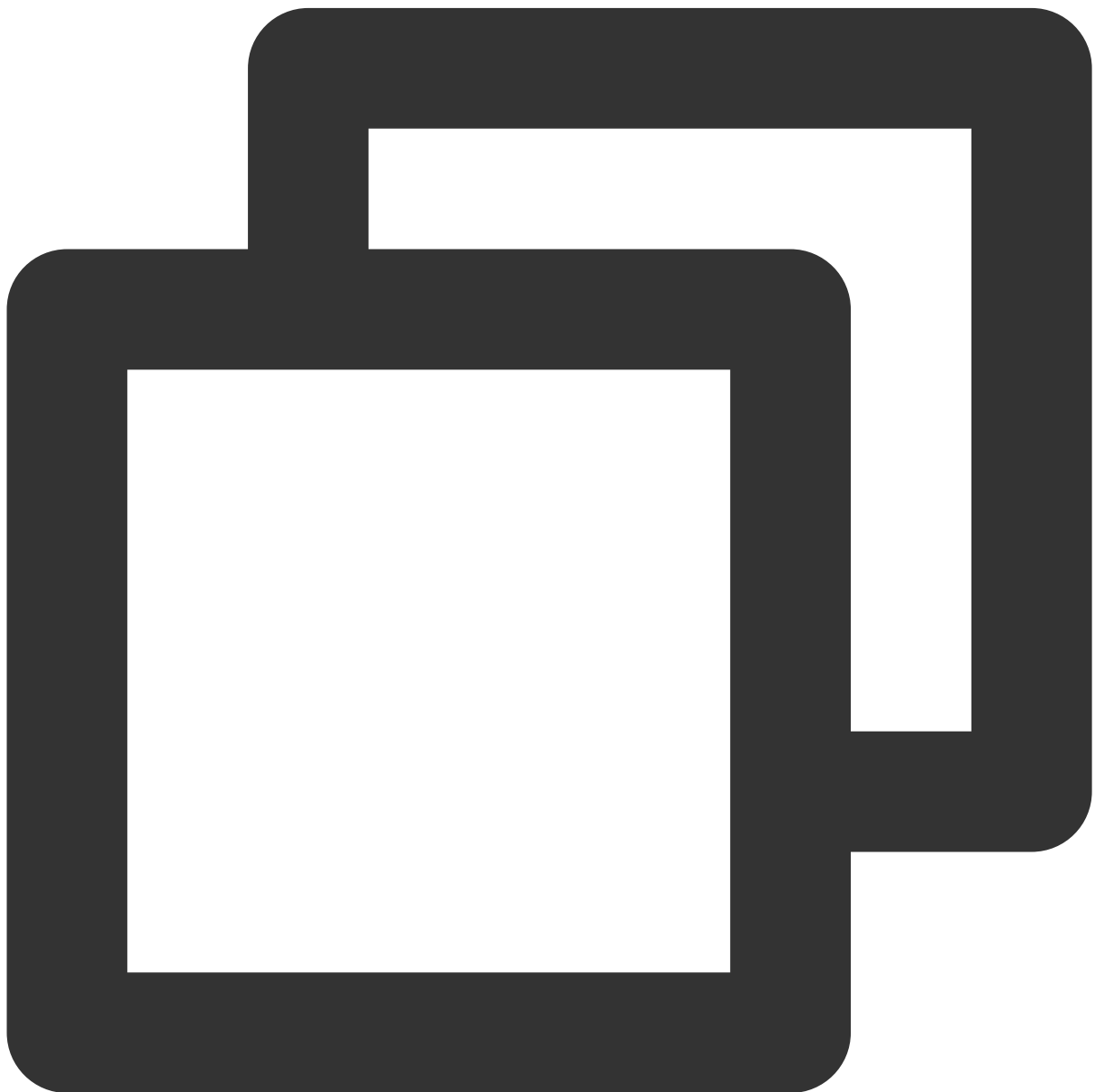
String

The user ID of the invitee.

Music Playback Status Callback APIs

onMusicPrepareToPlay

Music playback is ready.



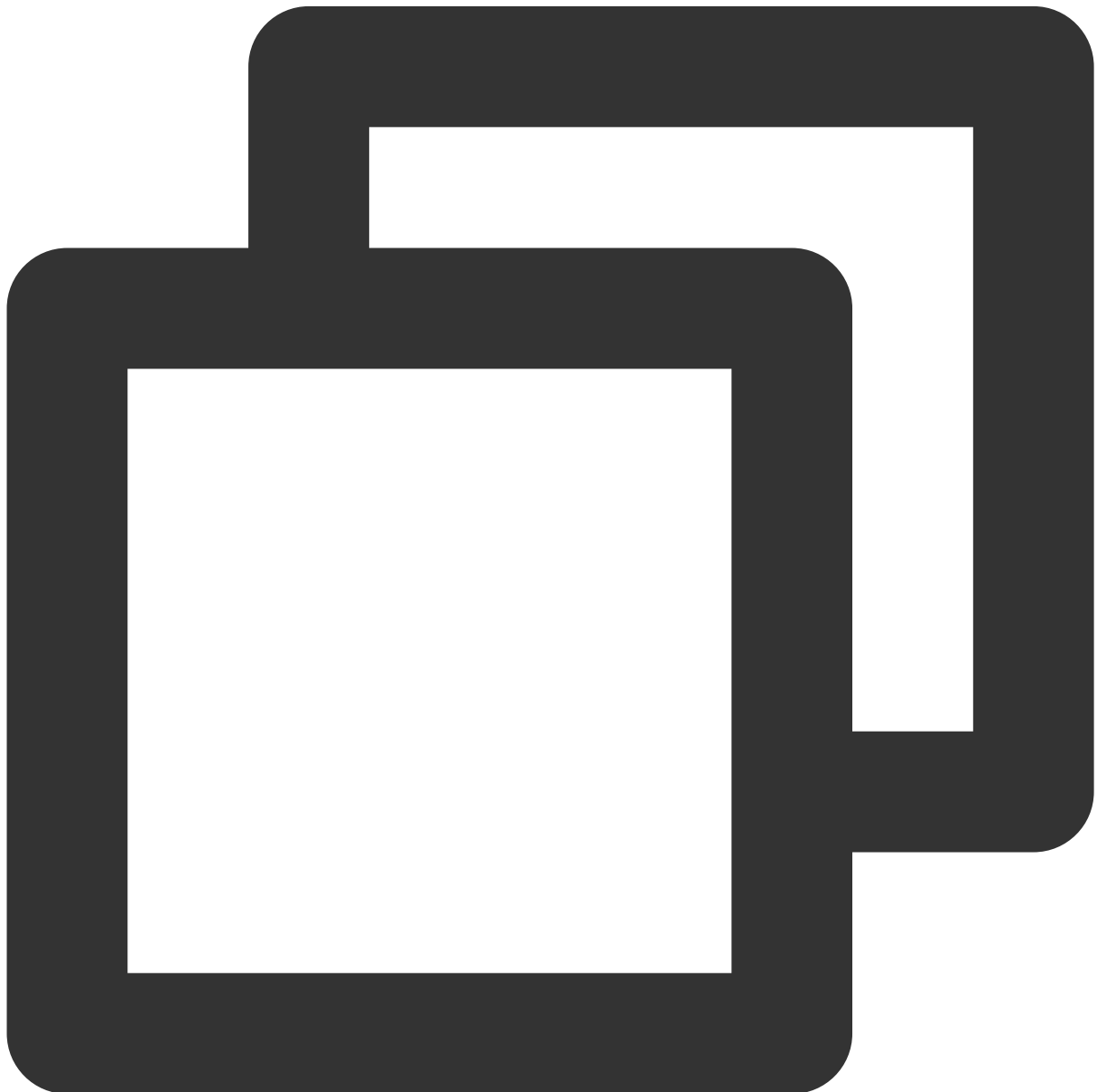
```
- (void)onMusicPrepareToPlay:(int32_t)musicID  
NS_SWIFT_NAME(onMusicPrepareToPlay(musicID:));
```

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
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
setDelegate	Sets event callbacks.
setDelegateHandler	Sets the thread where event callbacks are.
login	Logs in.

logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
destroyRoom	Terminates a room (called by room owner).
enterRoom	Enters a room (called by listener).
exitRoom	Exits a room (called by listener).
getRoomInfoList	Gets room list details.
getUserInfoList	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
startPlayMusic	Starts music.
stopPlayMusic	Stops music.
pausePlayMusic	Pauses music.
resumePlayMusic	Resumes music.

Seat management APIs

API	Description
enterSeat	Becomes a speaker (called by room owner or listener).
leaveSeat	Becomes a listener (called by speaker).
pickSeat	Places a user in a seat (called by room owner).
kickSeat	Removes a speaker (called by room owner).
muteSeat	Mutes/Unmutes a seat (called by room owner).

closeSeat	Blocks/Unblocks a seat (called by room owner).
---------------------------	--

Local audio APIs

API	Description
startMicrophone	Starts mic capturing.
stopMicrophone	Stops mic capturing.
setAudioQuality	Sets audio quality.
muteLocalAudio	Mutes/Unmutes local audio.
setSpeaker	Sets whether to use the device speaker or receiver to play audio.
setAudioCaptureVolume	Sets mic capturing volume.
setAudioPlaybackVolume	Sets playback volume.
setVoiceEarMonitorEnable	Enables/Disables in-ear monitoring.

Remote audio APIs

API	Description
muteRemoteAudio	Mutes/Unmutes a specified member.
muteAllRemoteAudio	Mutes/Unmutes all members.

Background music and audio effect APIs

API	Description
getAudioEffectManager	Gets the background music and audio effect management object TXAudioEffectManager .

Message sending APIs

API	Description
sendRoomTextMsg	Broadcasts a text chat message in a room. This API is generally used for on-screen comments.
sendRoomCustomMsg	Sends a custom text message.

Invitation signaling APIs

API	Description
sendInvitation	Sends an invitation.
acceptInvitation	Accepts an invitation.
rejectInvitation	Declines an invitation.
cancellInvitation	Cancels an invitation.

TRTCKaraokeRoomDelegate API Overview

Common event callbacks

API	Description
onError	Callback for error.
onWarning	Callback for warning.
onDebugLog	Callback of log.

Room event callback APIs

API	Description
onRoomDestroy	The room was terminated.
onRoomInfoChange	The room information changed.
onUserVolumeUpdate	The user volume.

Seat list change callback APIs

API	Description
onSeatListChange	All seat changes.
onAnchorEnterSeat	A user became a speaker or was made a speaker by the room owner.
onAnchorLeaveSeat	A user became a listener or was made a listener by the room owner.

onSeatMute	The room owner muted a seat.
onUserMicrophoneMute	Whether a user's mic is muted.
onSeatClose	The room owner blocked a seat.

Callback APIs for room entry/exit by listener

API	Description
onAudienceEnter	A listener entered the room.
onAudienceExit	A listener exited the room.

Message event callback APIs

API	Description
onRecvRoomTextMsg	A text chat message was received.
onRecvRoomCustomMsg	A custom message was received.

Signaling Event Callback APIs

API	Description
onReceiveNewInvitation	Receipt of an invitation.
onInviteeAccepted	Invitation accepted by invitee.
onInviteeRejected	Invitation declined by invitee.
onInvitationCancelled	The inviter canceled the invitation.

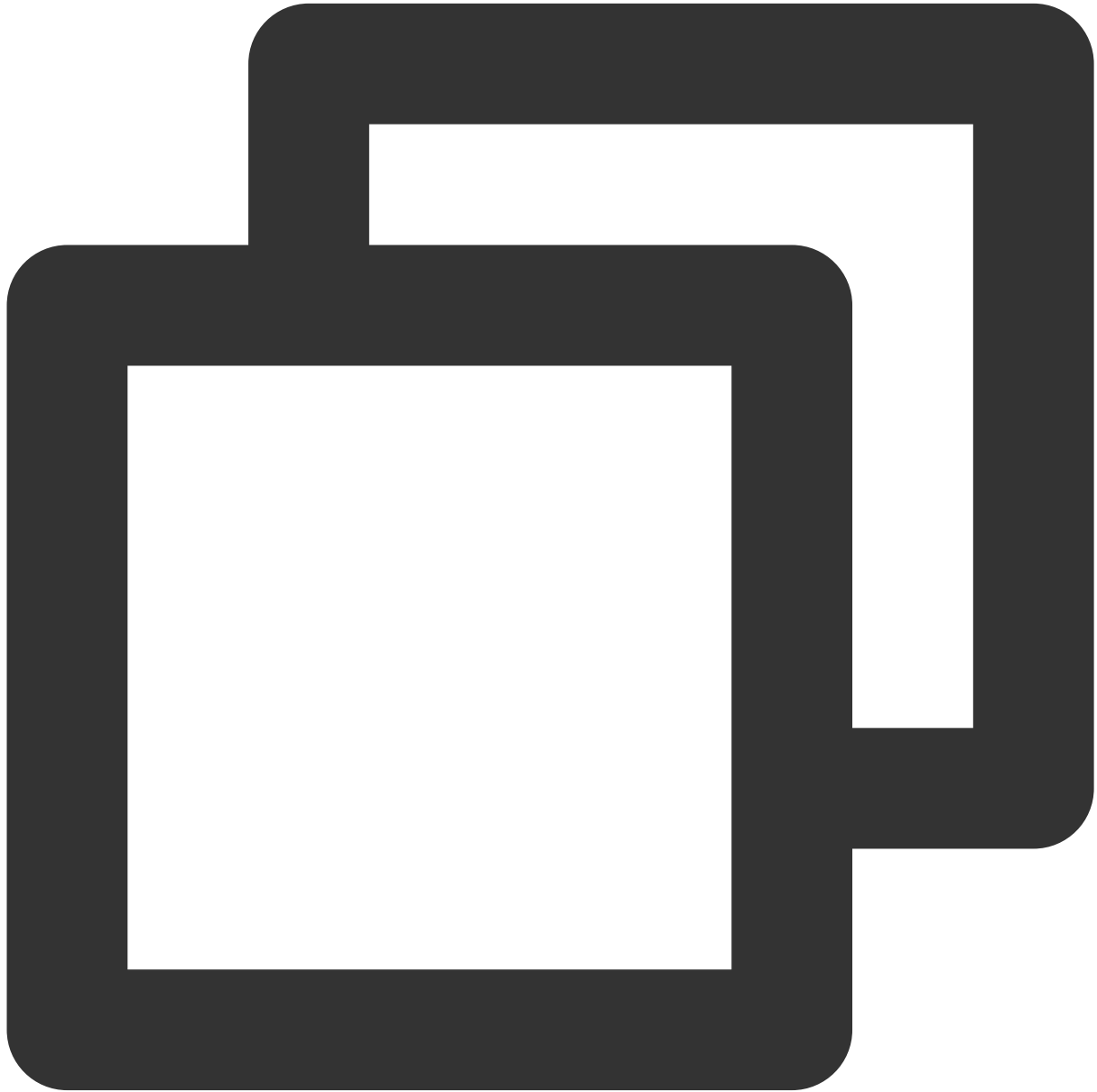
Song event callback APIs

API	Description
onMusicProgressUpdate	Music playback progress.
onMusicPrepareToPlay	Music playback is ready.
onMusicCompletePlaying	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

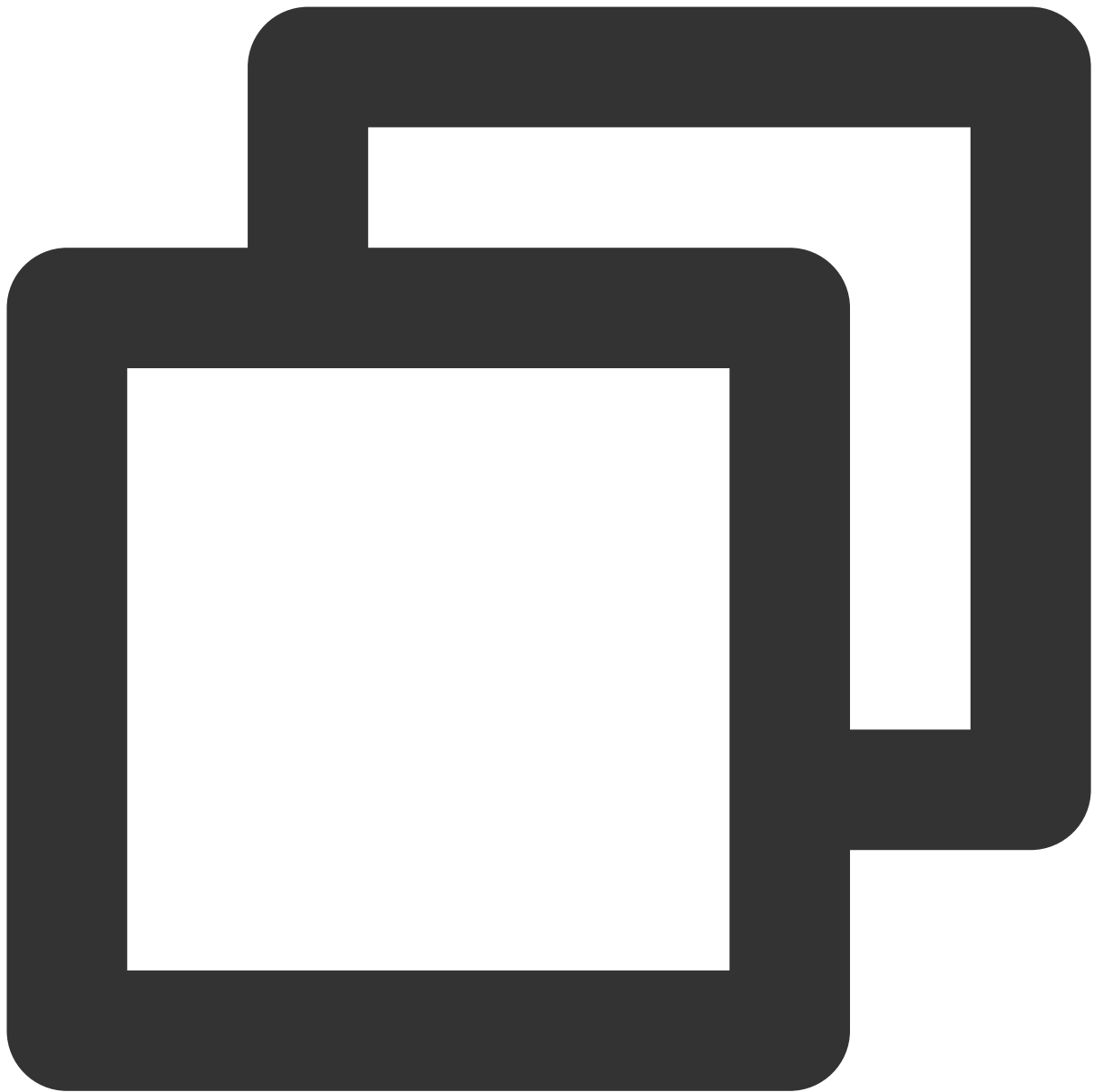
After the instance is terminated, the externally cached `TRTCKaraokeRoom` instance can no longer be used. You need to call `sharedInstance` again to get a new instance.



```
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.



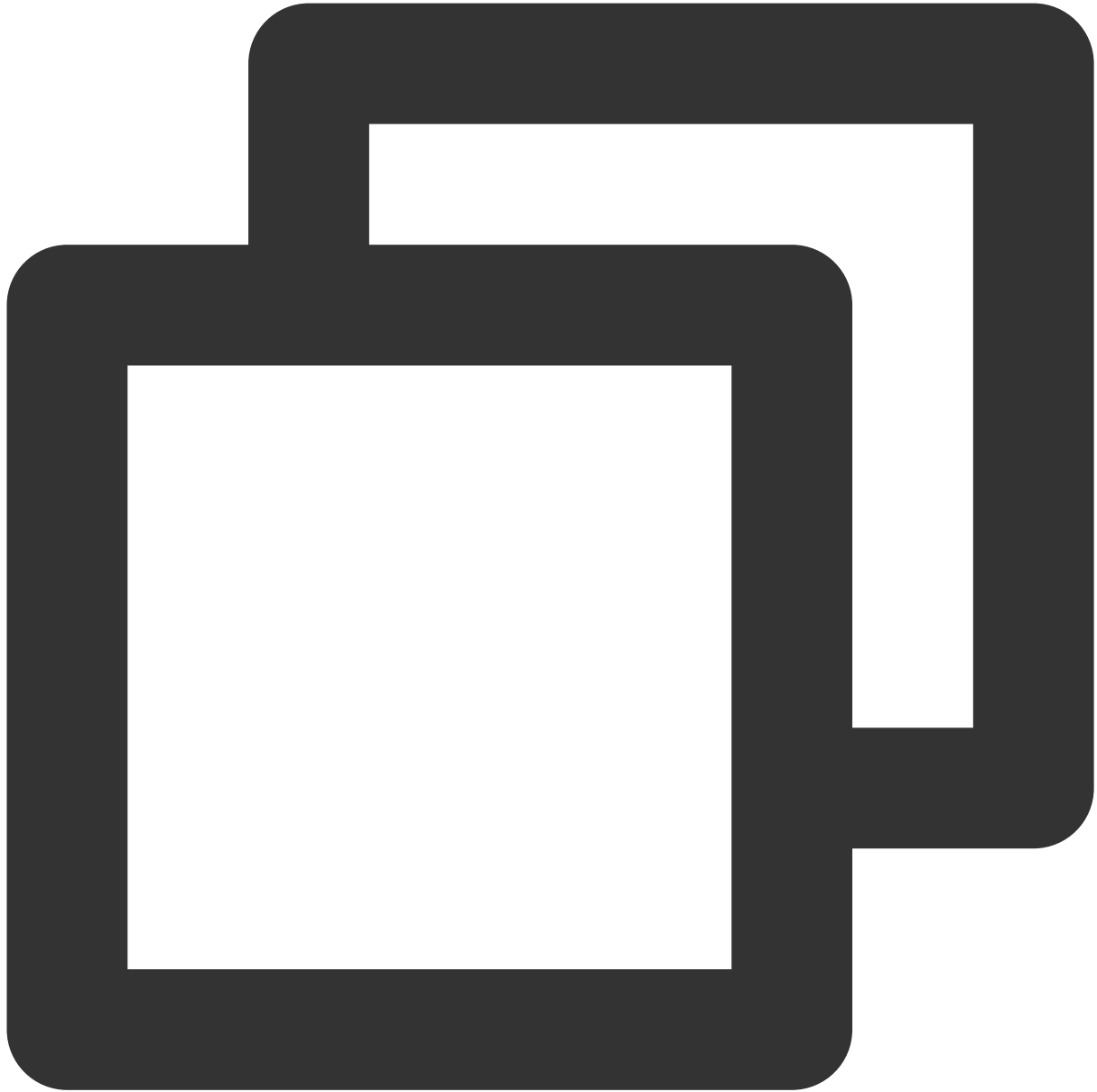
```
public abstract void setDelegateHandler(Handler handler);
```

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);
```

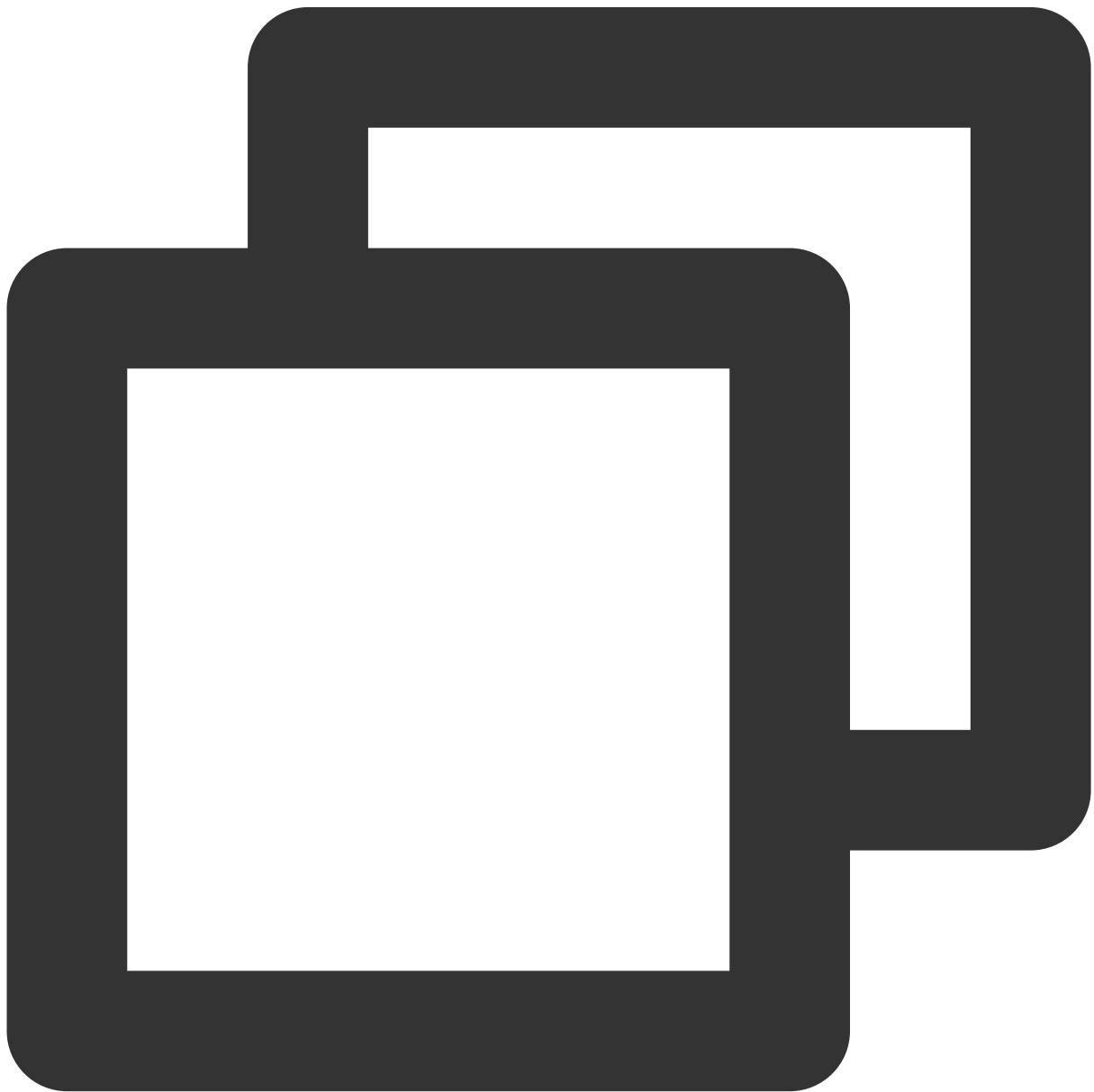
The parameters are described below:

Parameter	Type	Description
sdkAppId	int	You can view the <code>SDKAppID</code> via Application Management > Application Info 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 FAQs > UserSig .
callback	ActionCallback	The callback for login. The code is <code>0</code> if login succeeds.

logout

Log out



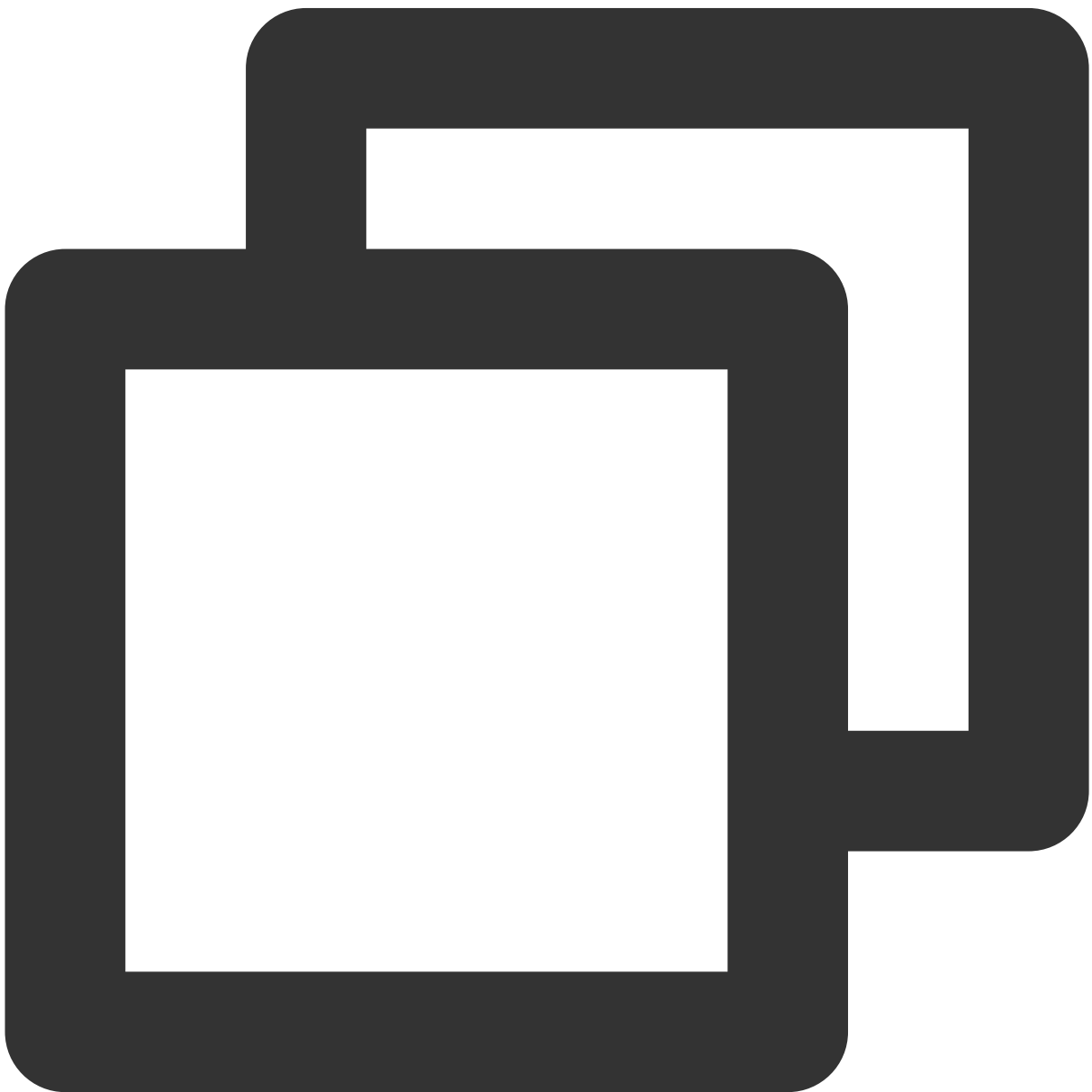
```
public abstract void logout(TRTCKaraokeRoomCallback.ActionCallback callback);
```

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 `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).



```
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, TRICKaraokeRoomCallback.ActionCallback c
```

The parameters are described below:

Parameter	Type	Description
roomId	int	The room ID.
callback	ActionCallback	The callback for room entry. The code is <code>0</code> 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:

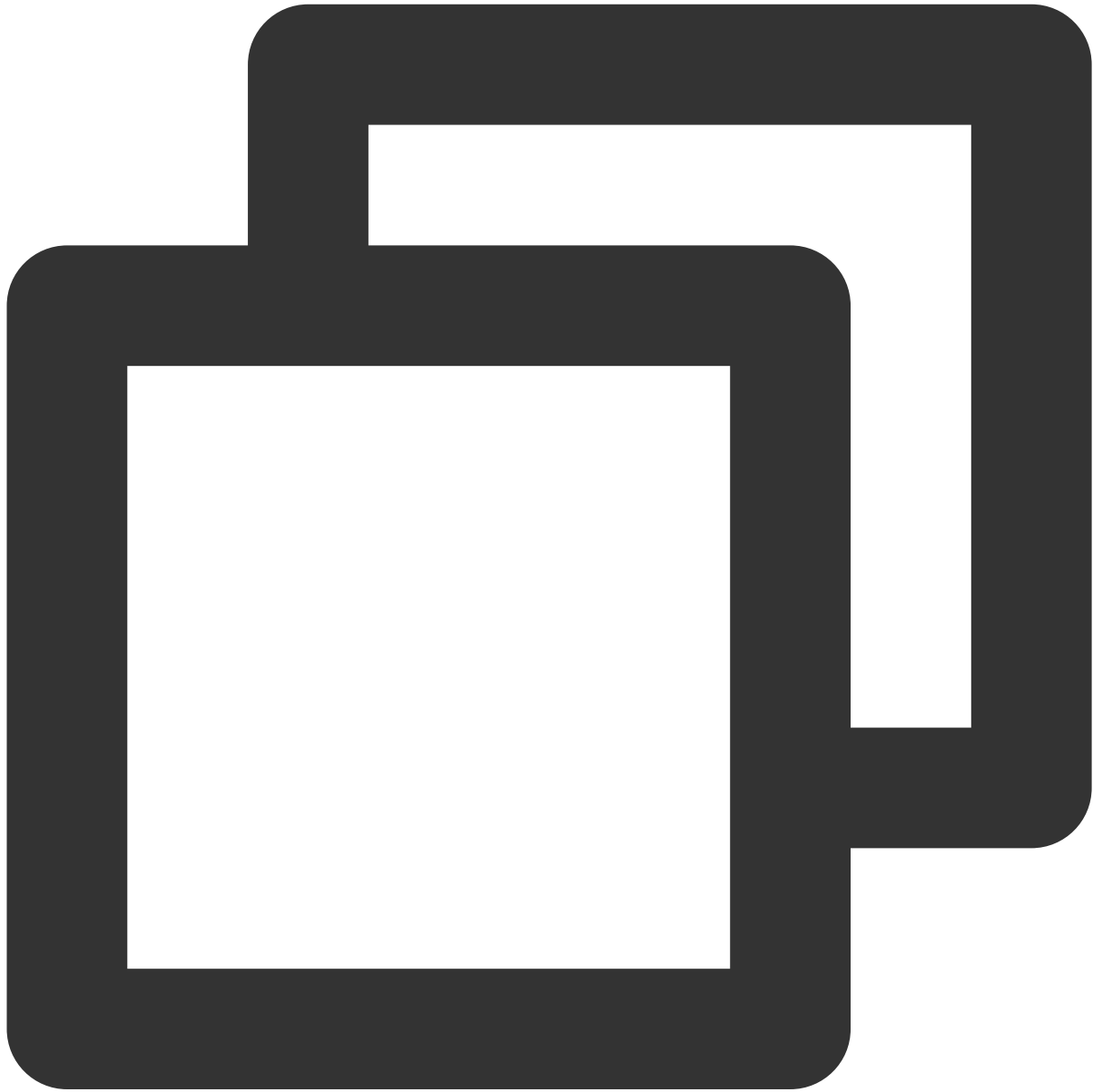
Parameter	Type	Description
callback	ActionCallback	The callback for room exit. The code is 0 if the operation succeeds.

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
```

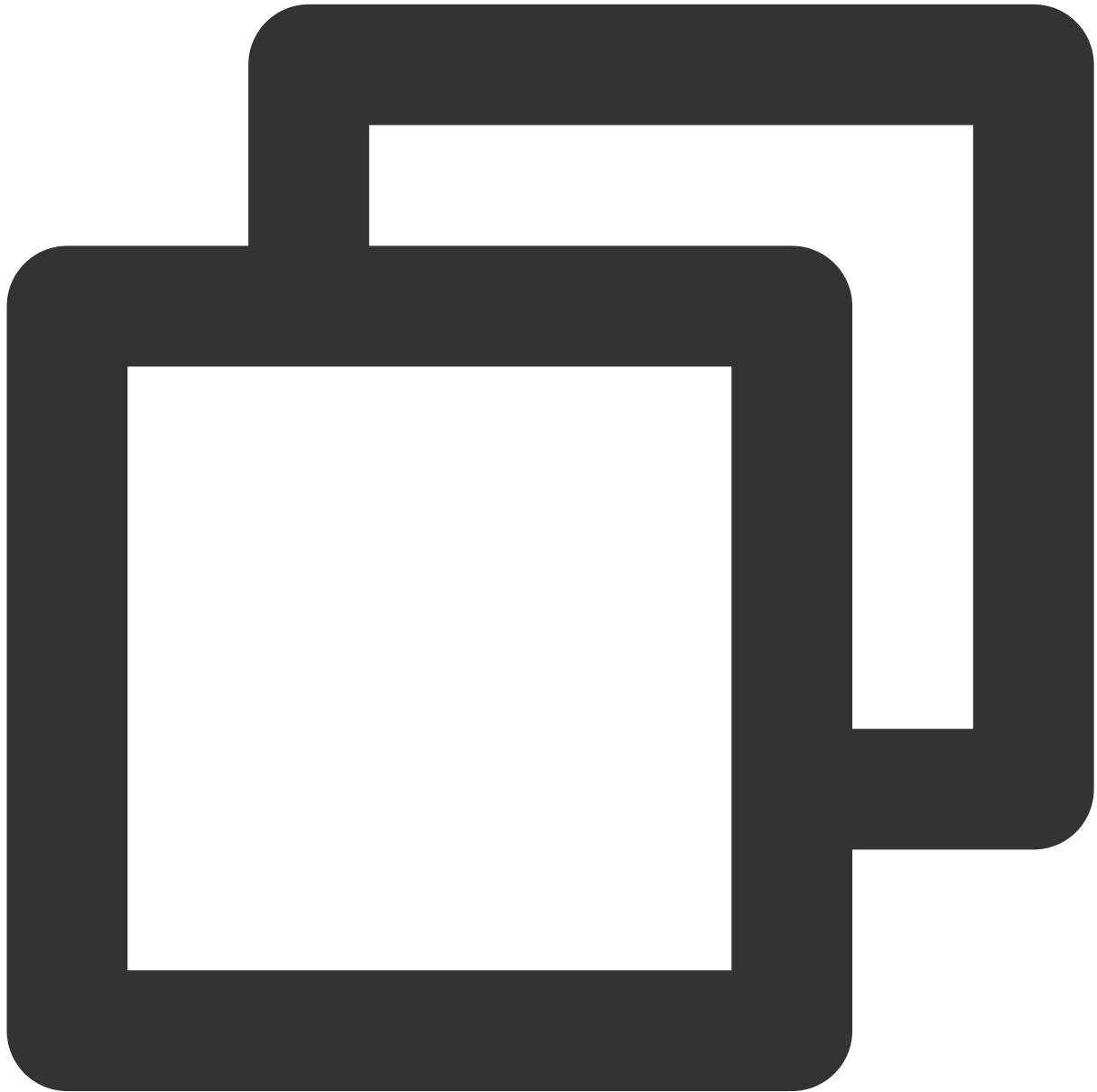
The parameters are described below:

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

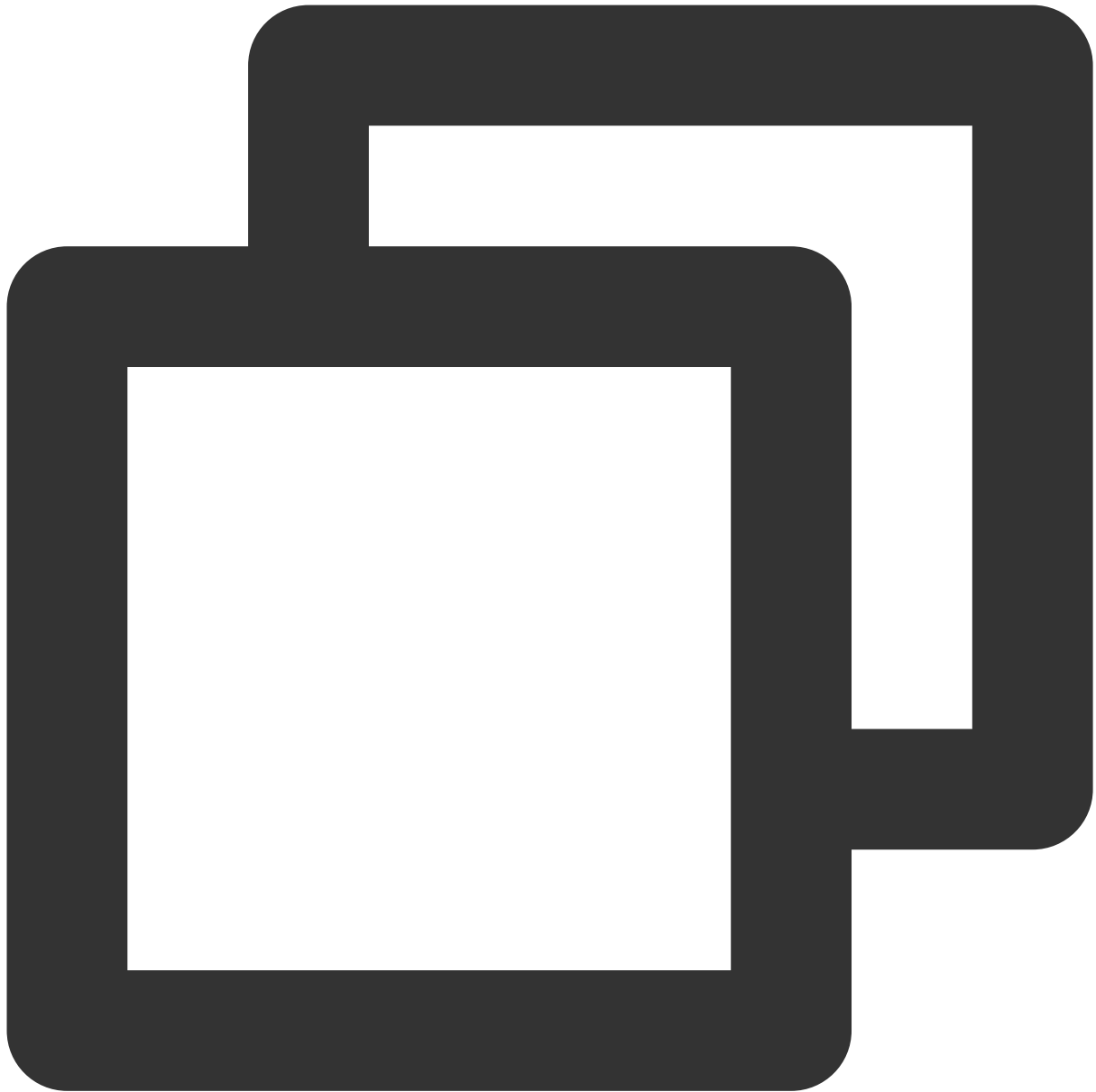
This API is used to play music (called after becoming a speaker).

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.

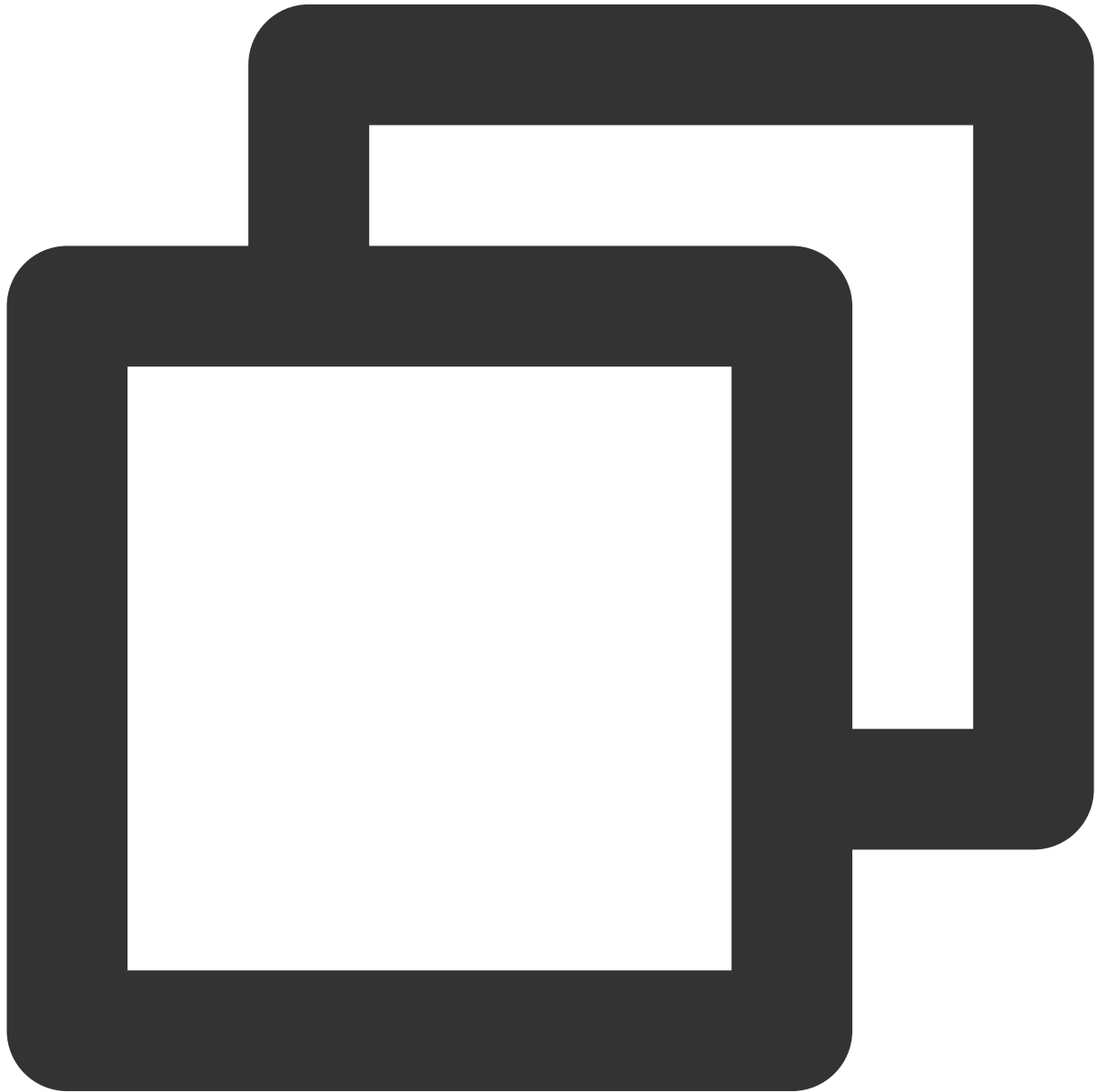
After this API is called, the song that is currently playing will stop.

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();
```

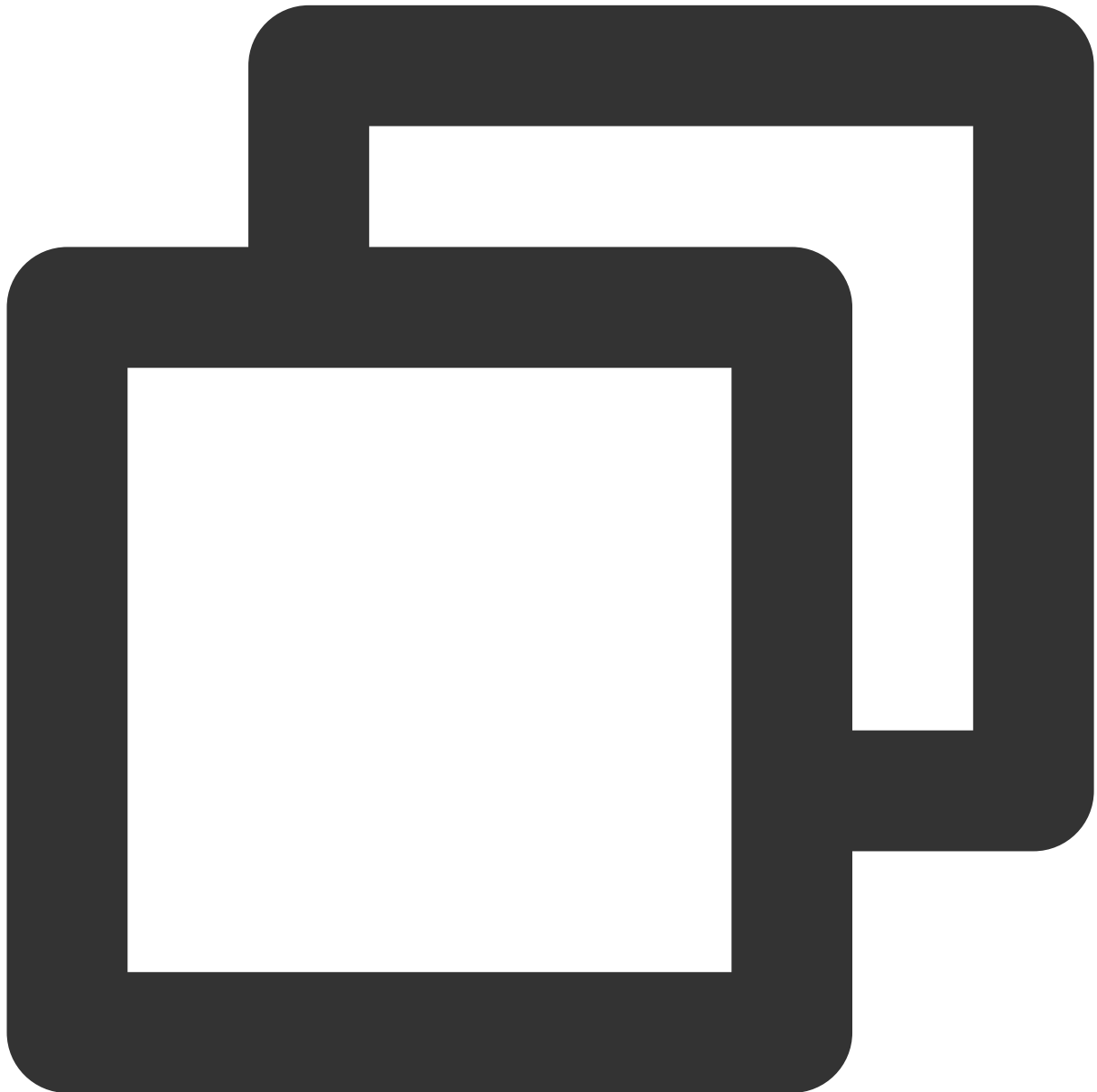
pausePlayMusic

This API is used to pause music (called during music playback).

Note

The `onMusicProgressUpdate` notification will be paused.

No `onMusicCompletePlaying` notification will be received.



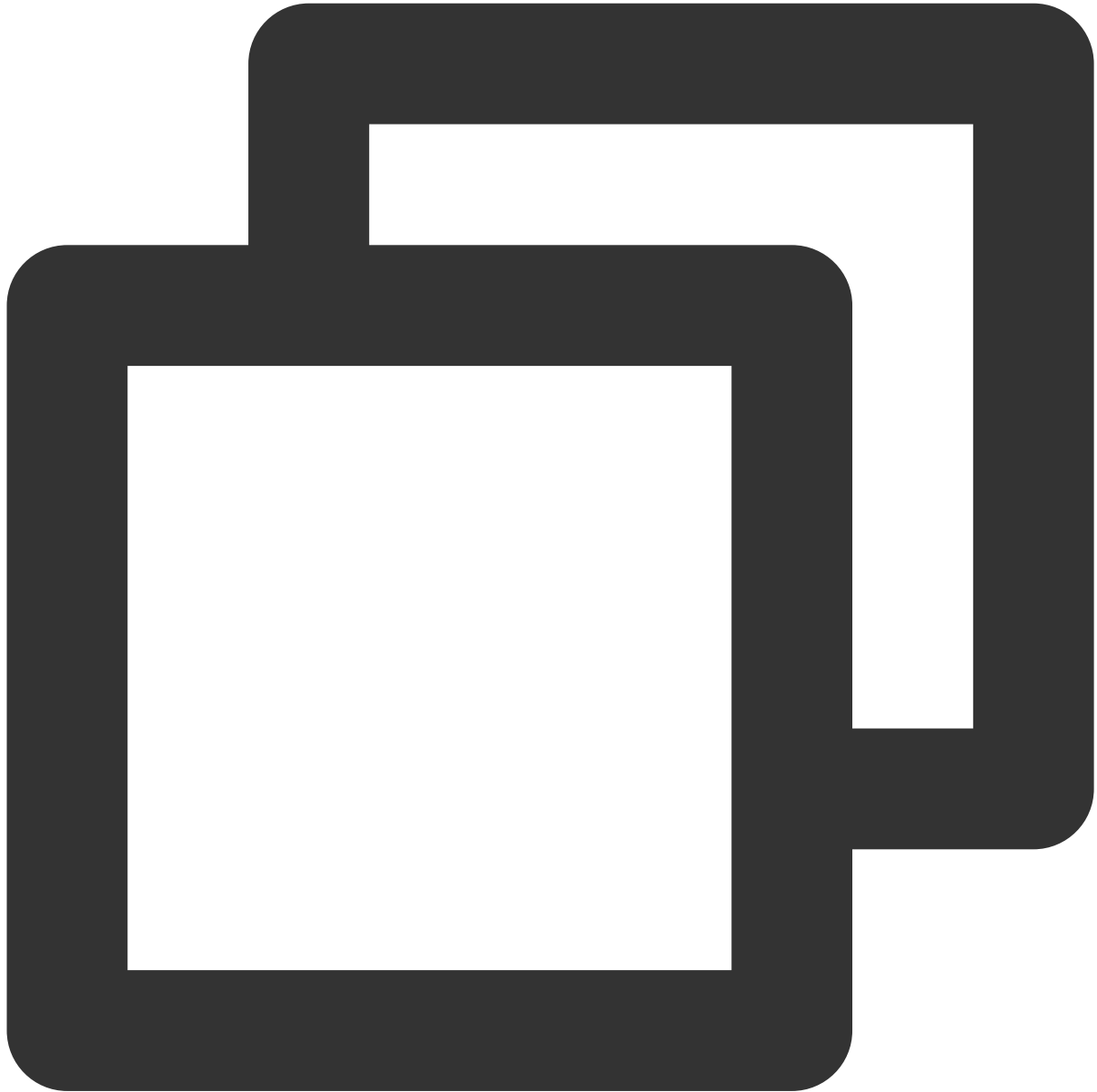
```
public abstract void pausePlayMusic();
```

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();
```

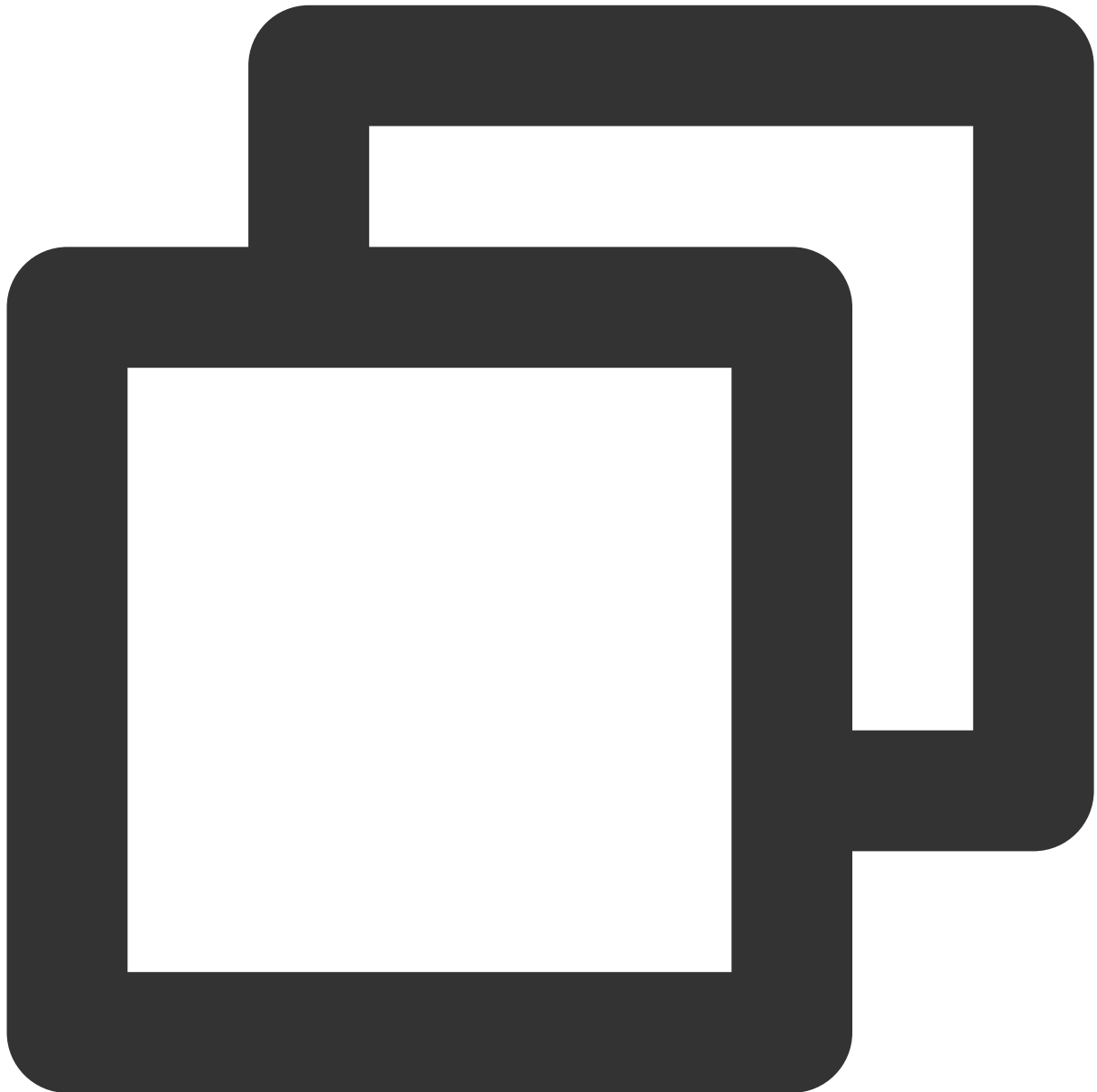
Seat Management APIs

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:

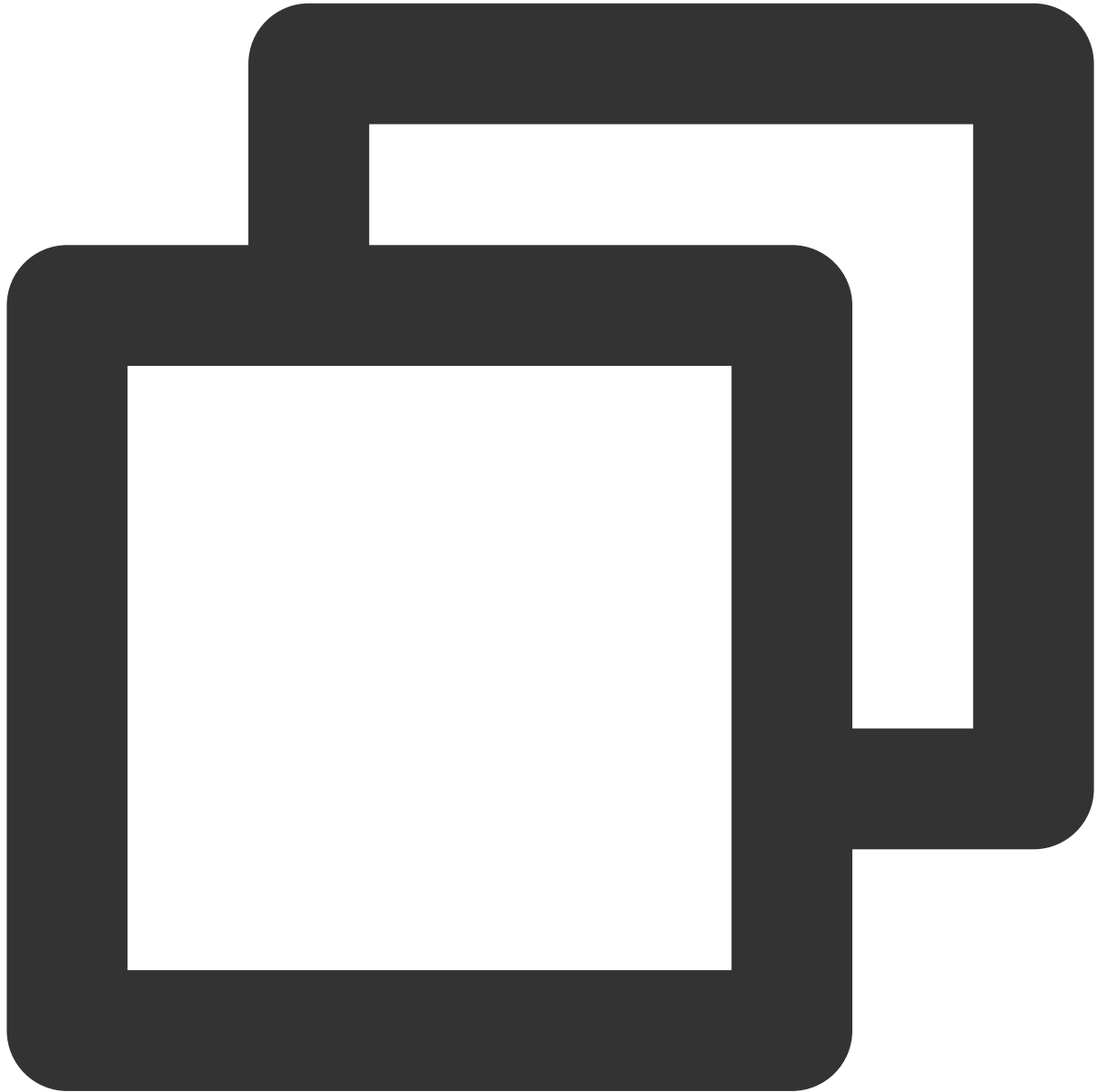
Parameter	Type	Description
callback	ActionCallback	The callback for the operation.

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.

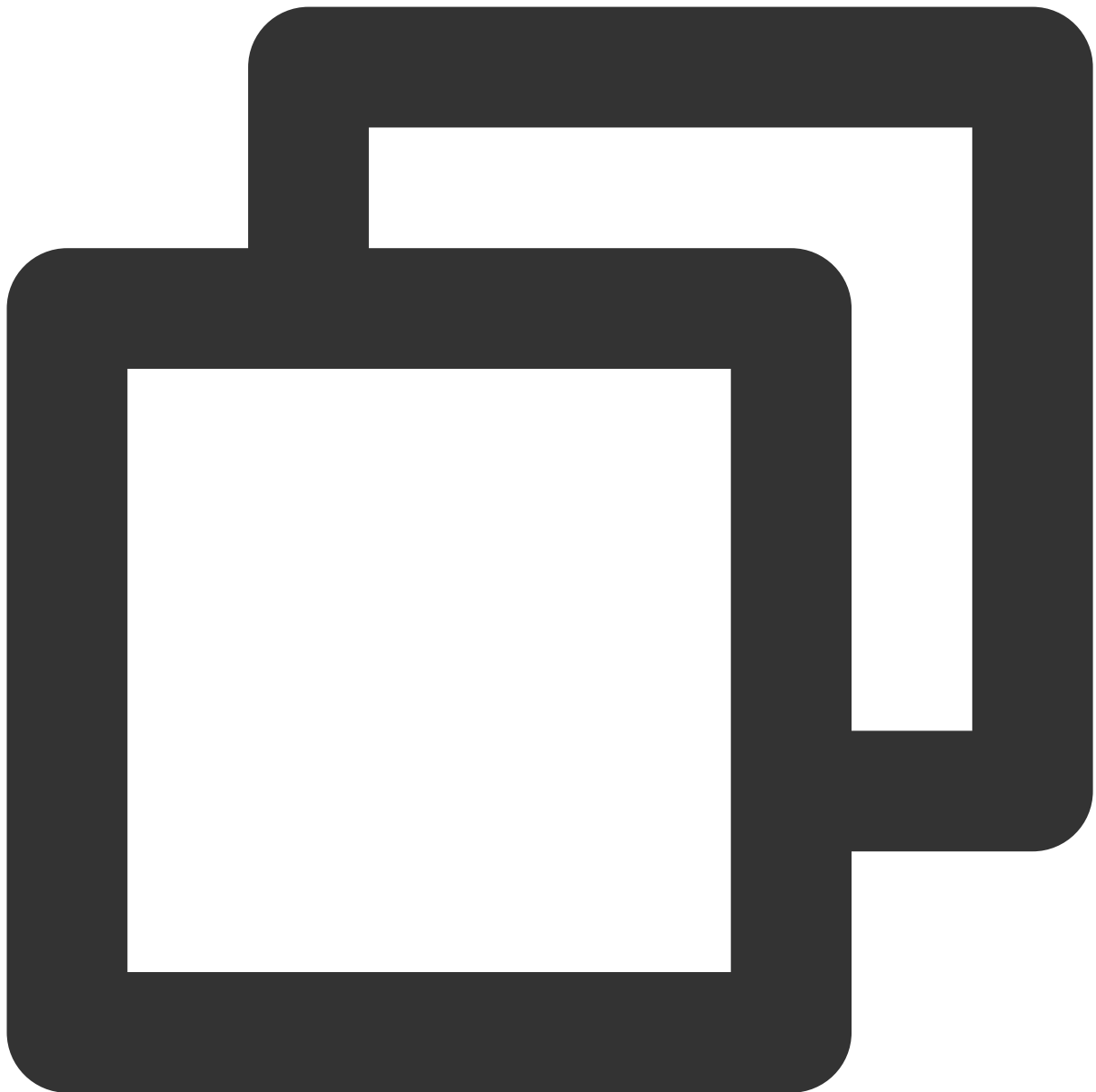
userId	String	The user ID.
callback	ActionCallback	The callback for the operation.

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, TRICKaraokeRoomCallback.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.

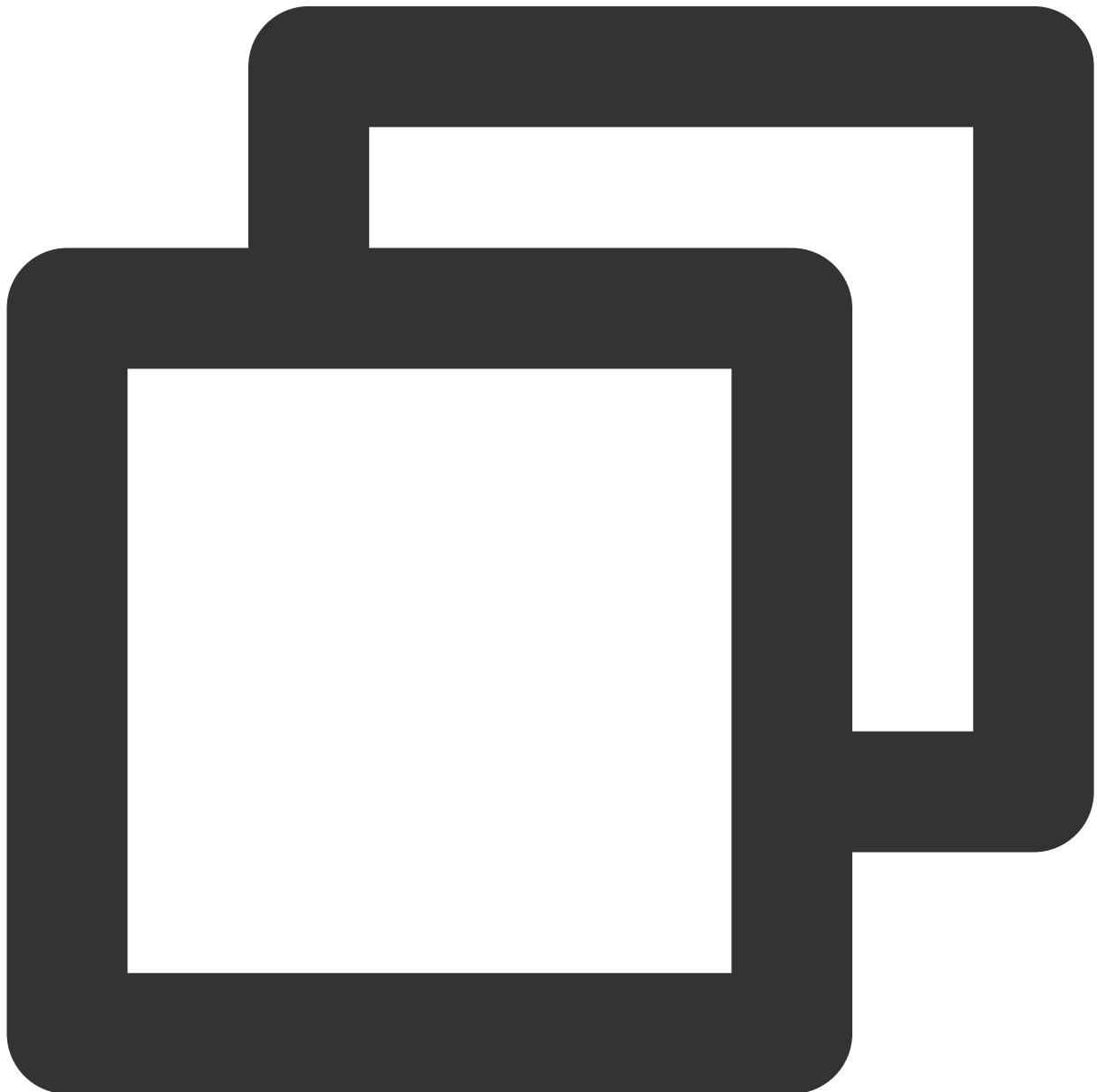
Calling this API will immediately modify the seat list.

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.

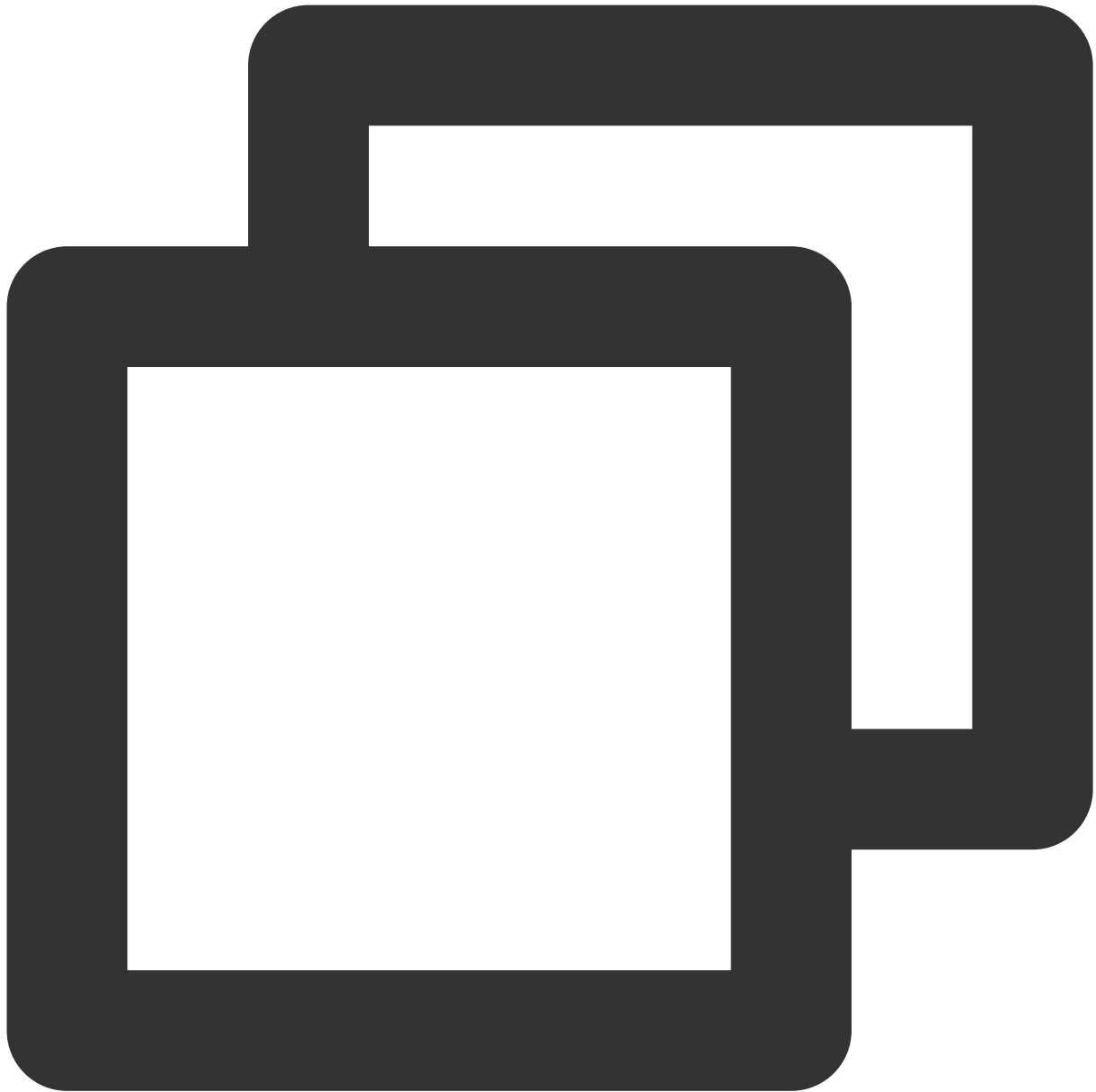
Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will call `muteAudio` to mute/unmute his or her audio.

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, TRICKaraokeRoomCallb
```

The parameters are described below:

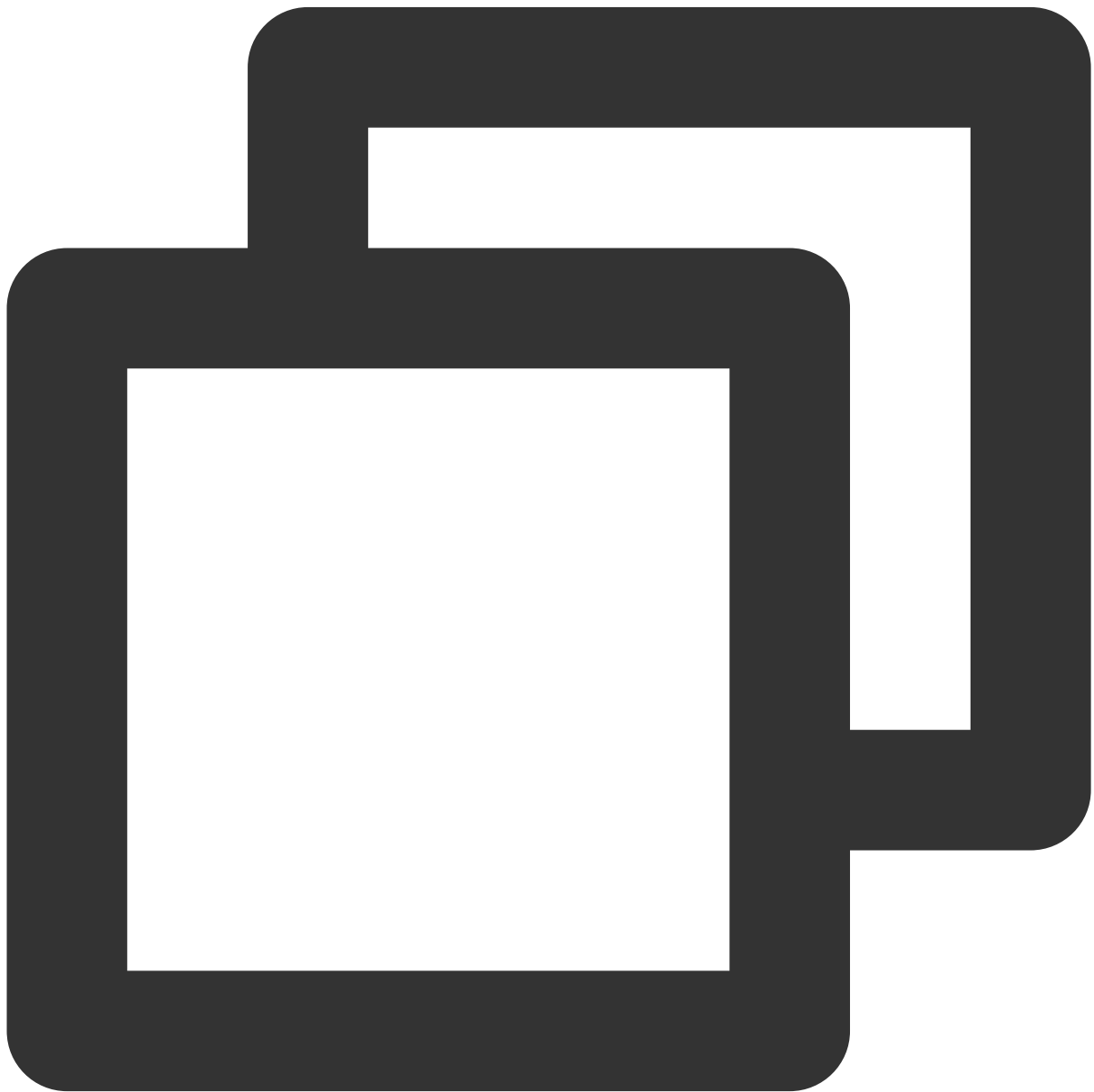
Parameter	Type	Description
seatIndex	int	The number of the seat to block/unblock.
isClose	boolean	<code>true</code> : Block; <code>false</code> : Unblock
callback	ActionCallback	The callback for the operation.

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

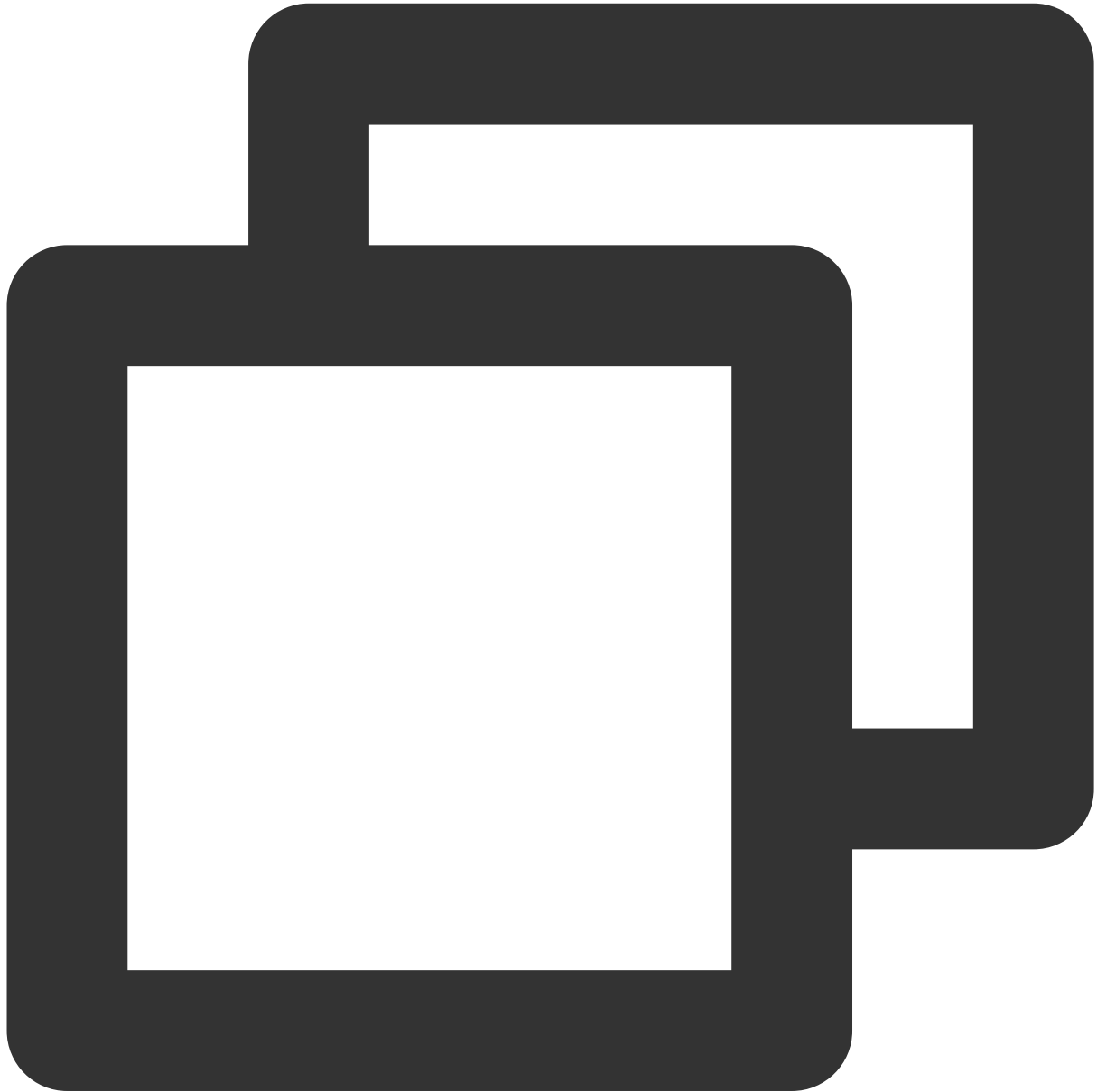
This API is used to start mic capturing.



```
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 setAudioQuality() .

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 muteLocalAudio() .

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	<code>true</code> : Speaker; <code>false</code> : Receiver

setAudioCaptureVolume

This API is used to set the mic capturing volume.



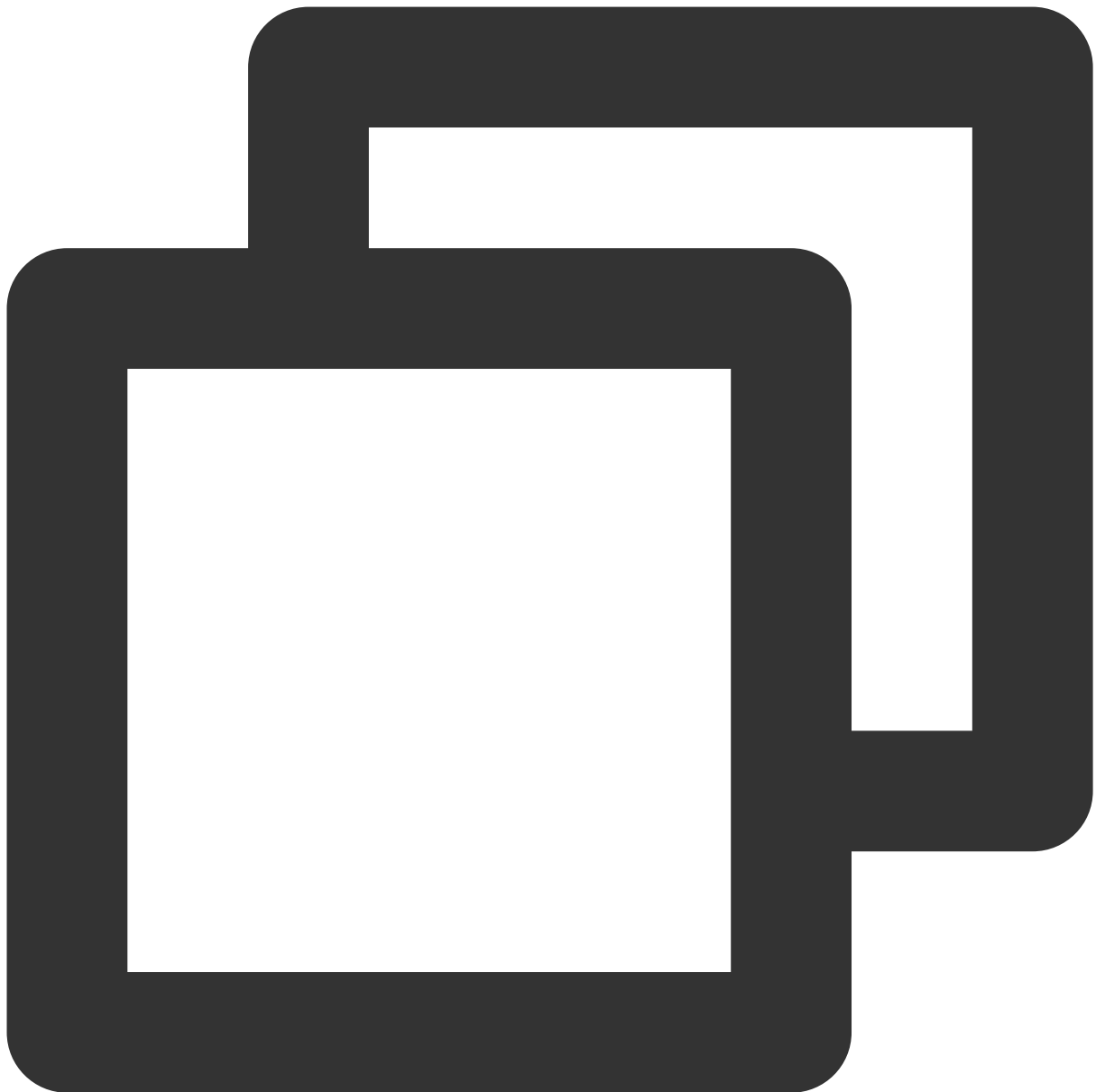
```
public abstract void setAudioCaptureVolume(int volume);
```

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.



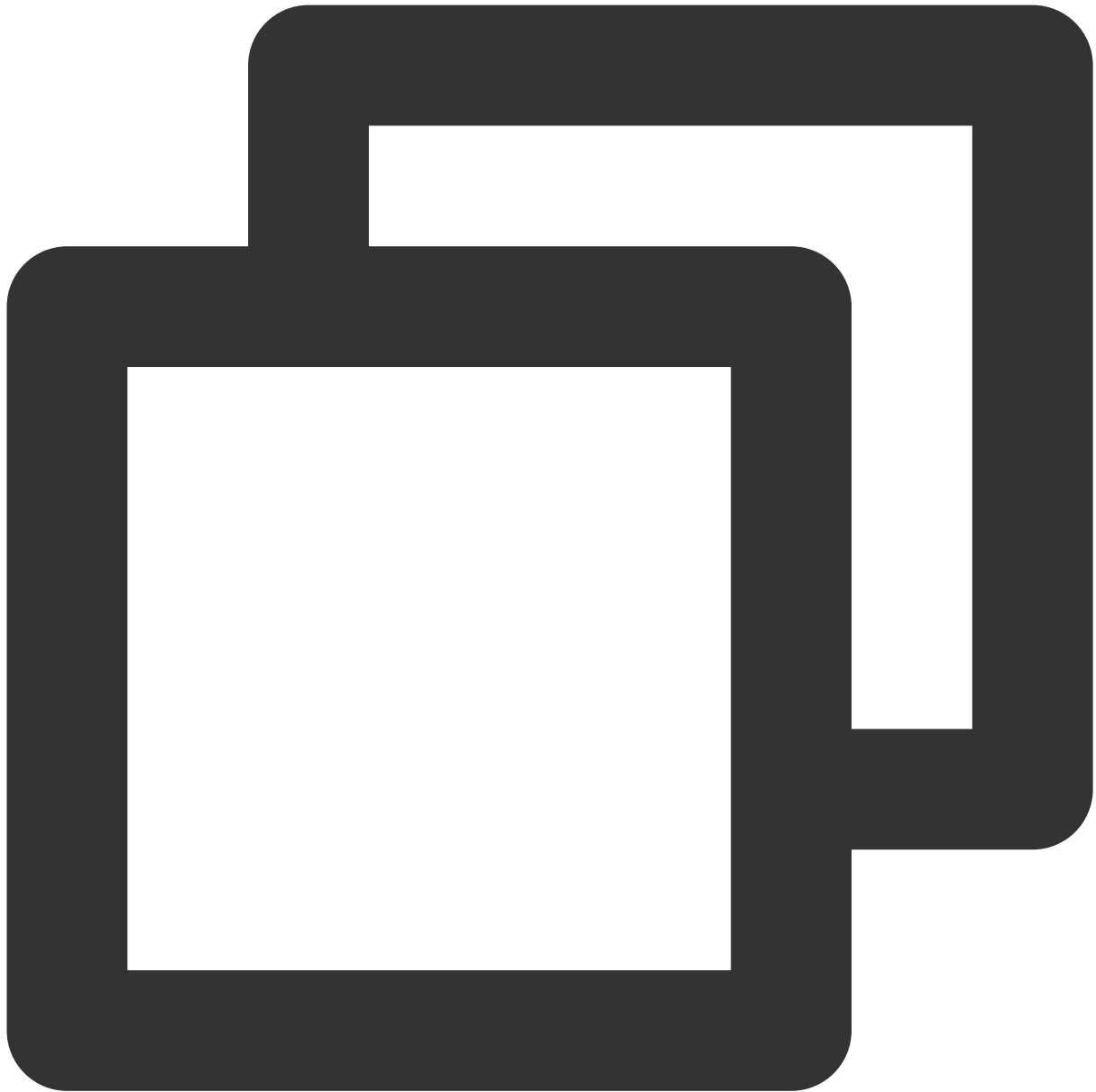
```
public abstract void setAudioPlayoutVolume(int volume);
```

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.



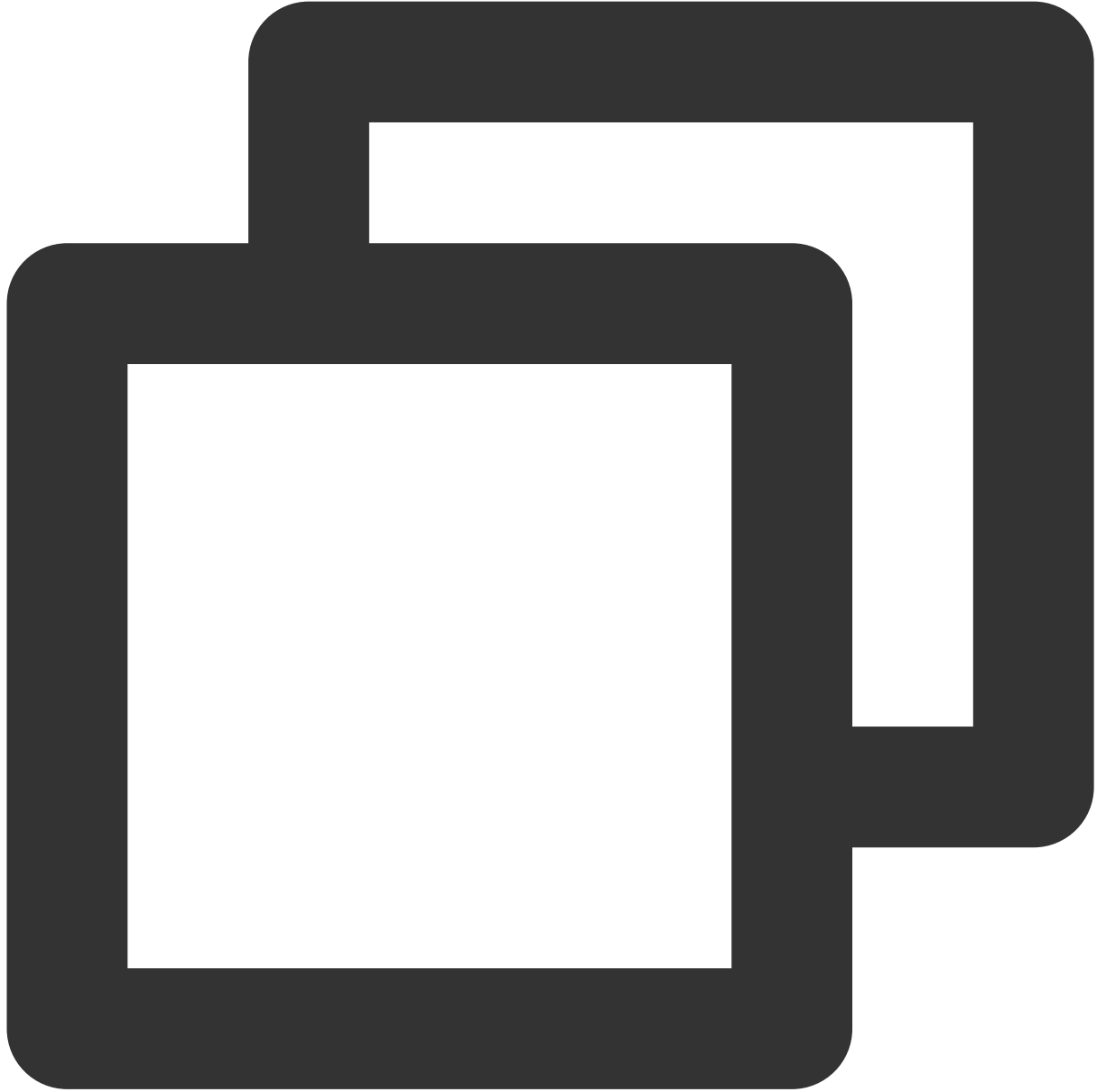
```
public abstract void muteRemoteAudio(String userId, boolean mute);
```

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.



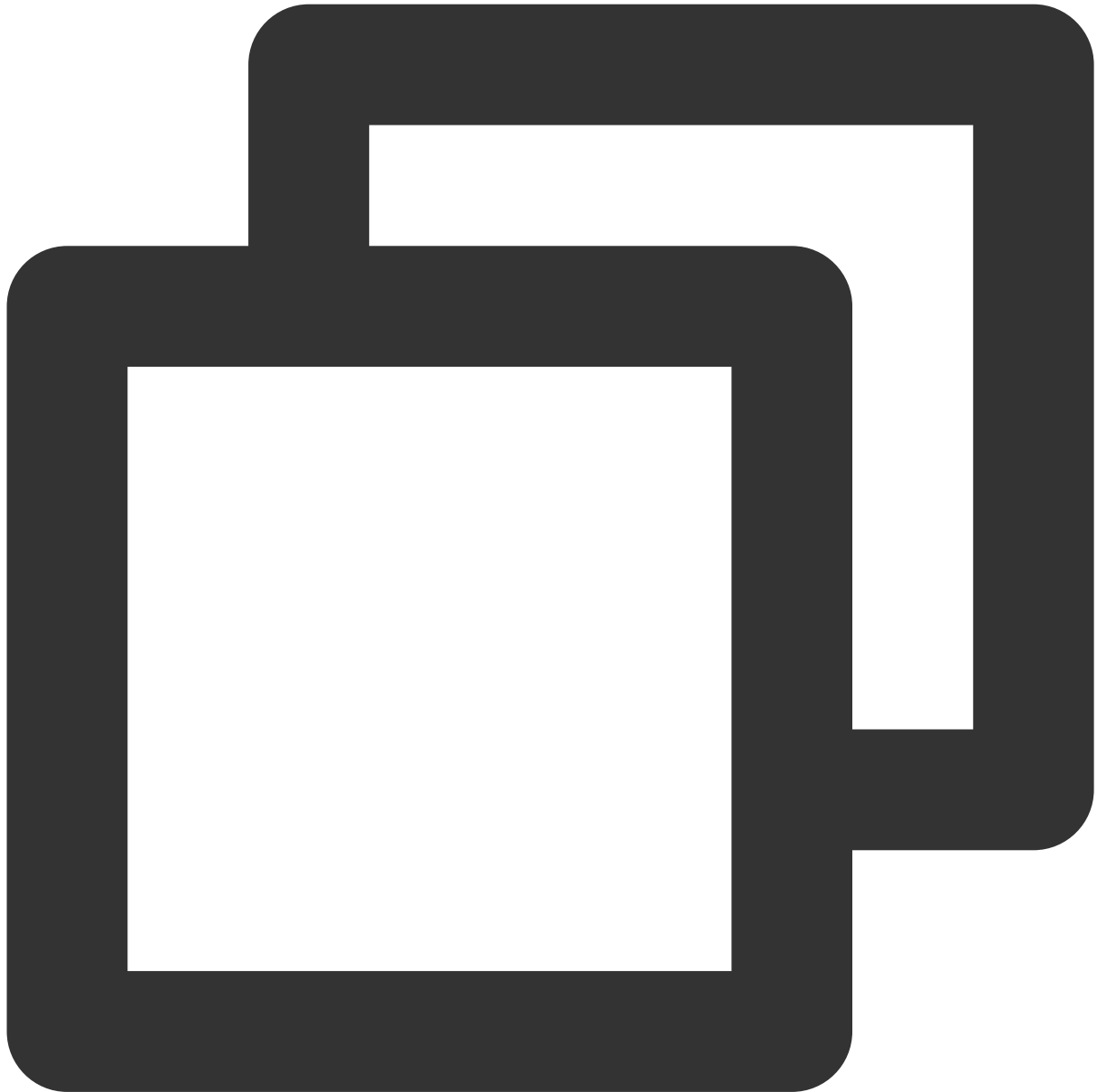
```
public abstract void muteAllRemoteAudio(boolean mute);
```

The parameters are described below:

Parameter	Type	Description
mute	boolean	<code>true</code> : Mute; <code>false</code> : Unmute

setVoiceEarMonitorEnable

This API is used to enable/disable in-ear monitoring.



```
public abstract void setVoiceEarMonitorEnable(boolean enable);
```

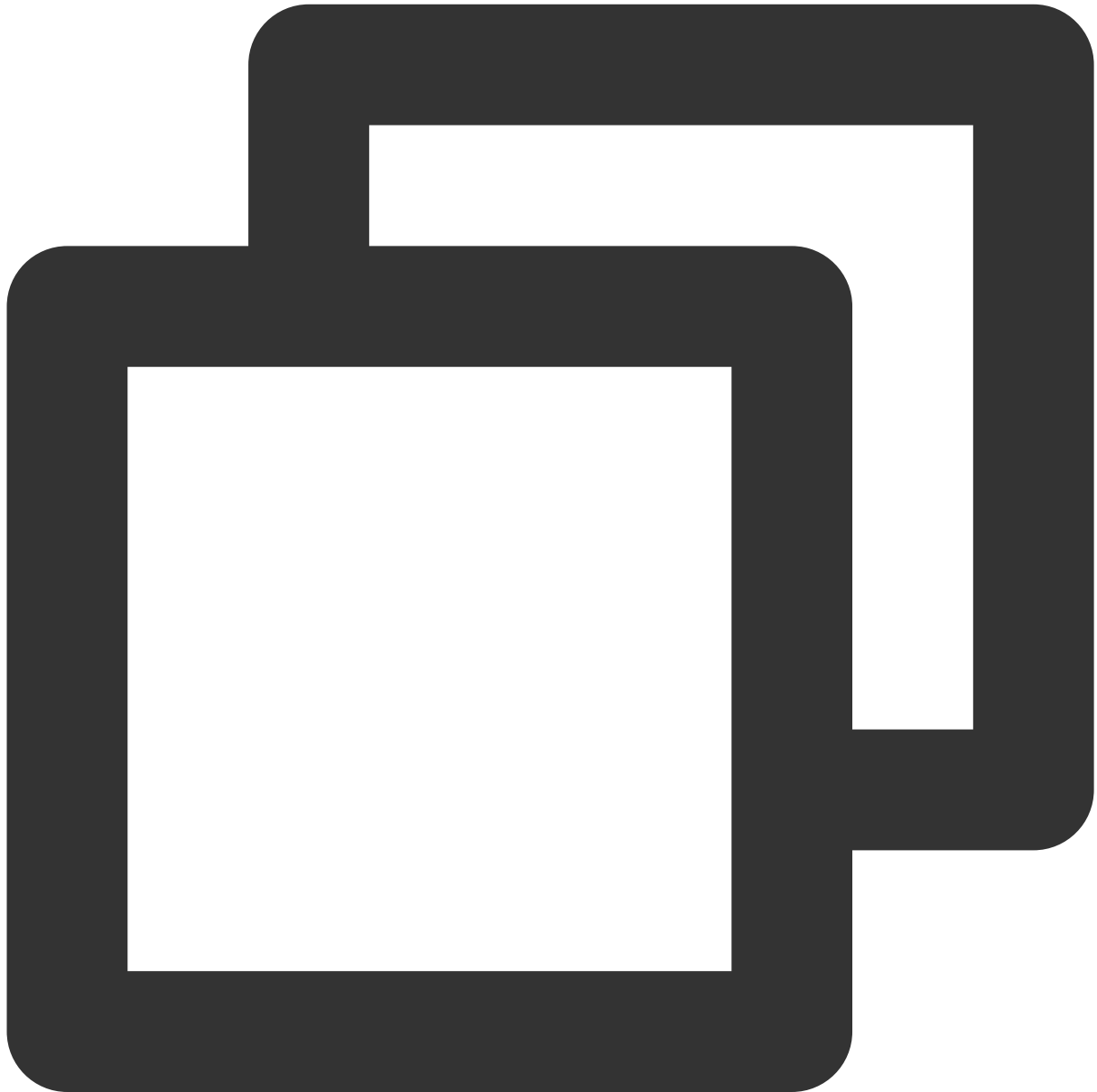
The parameters are described below:

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](#).

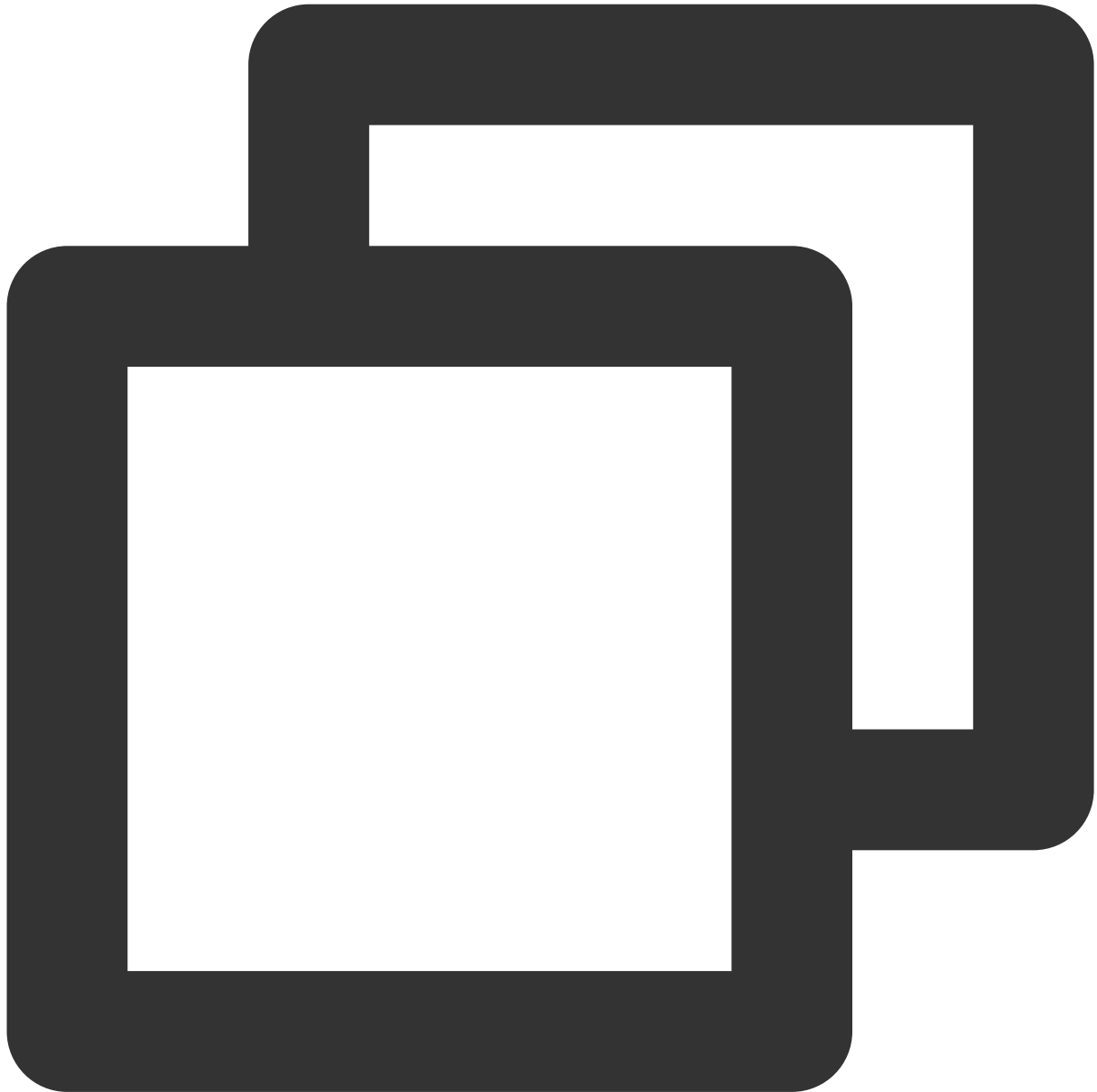


```
public abstract TXAudioEffectManager getAudioEffectManager();
```

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.



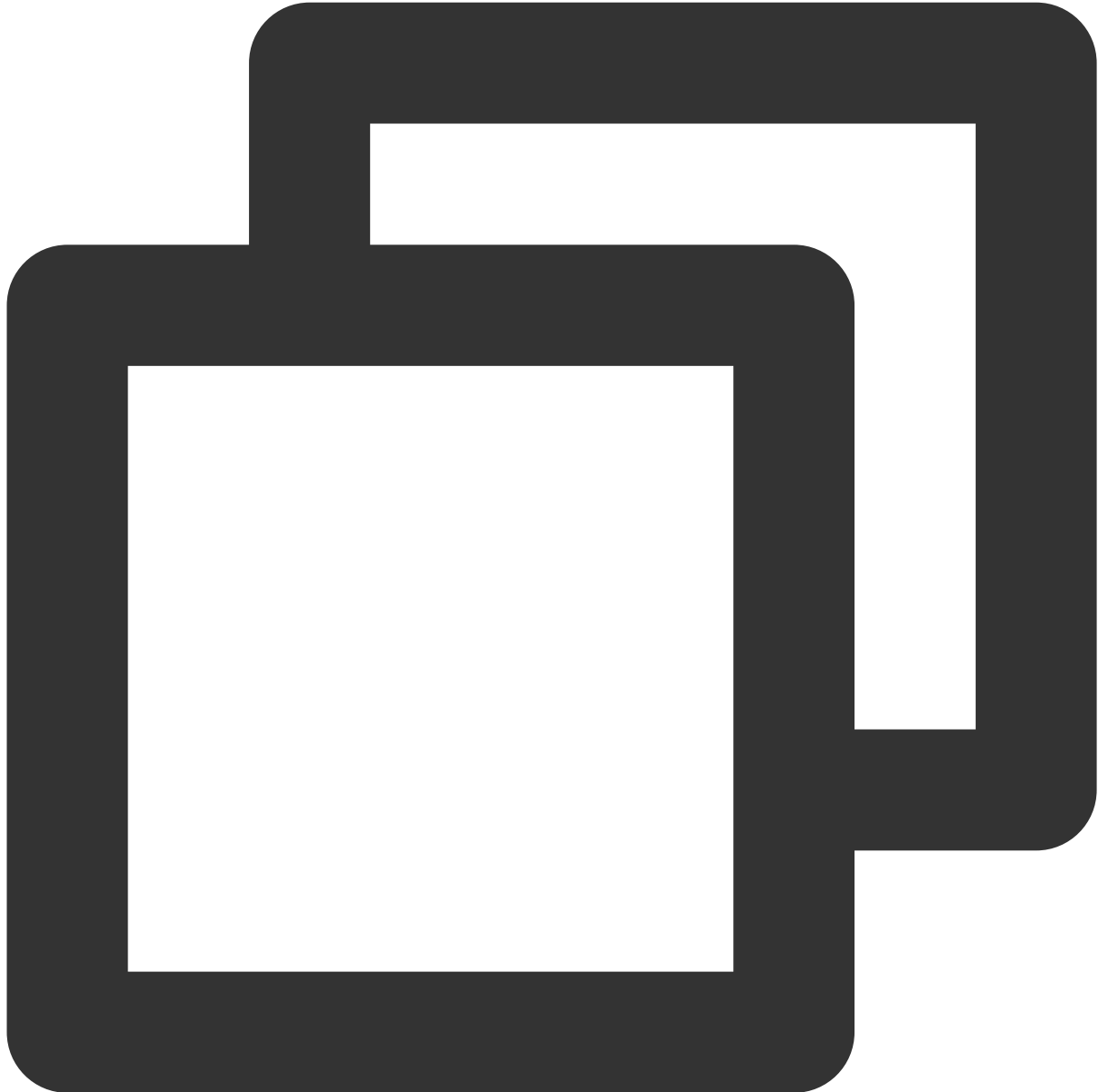
```
public abstract void sendRoomTextMsg(String message, TRTCKaraokeRoomCallback.Action
```

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.

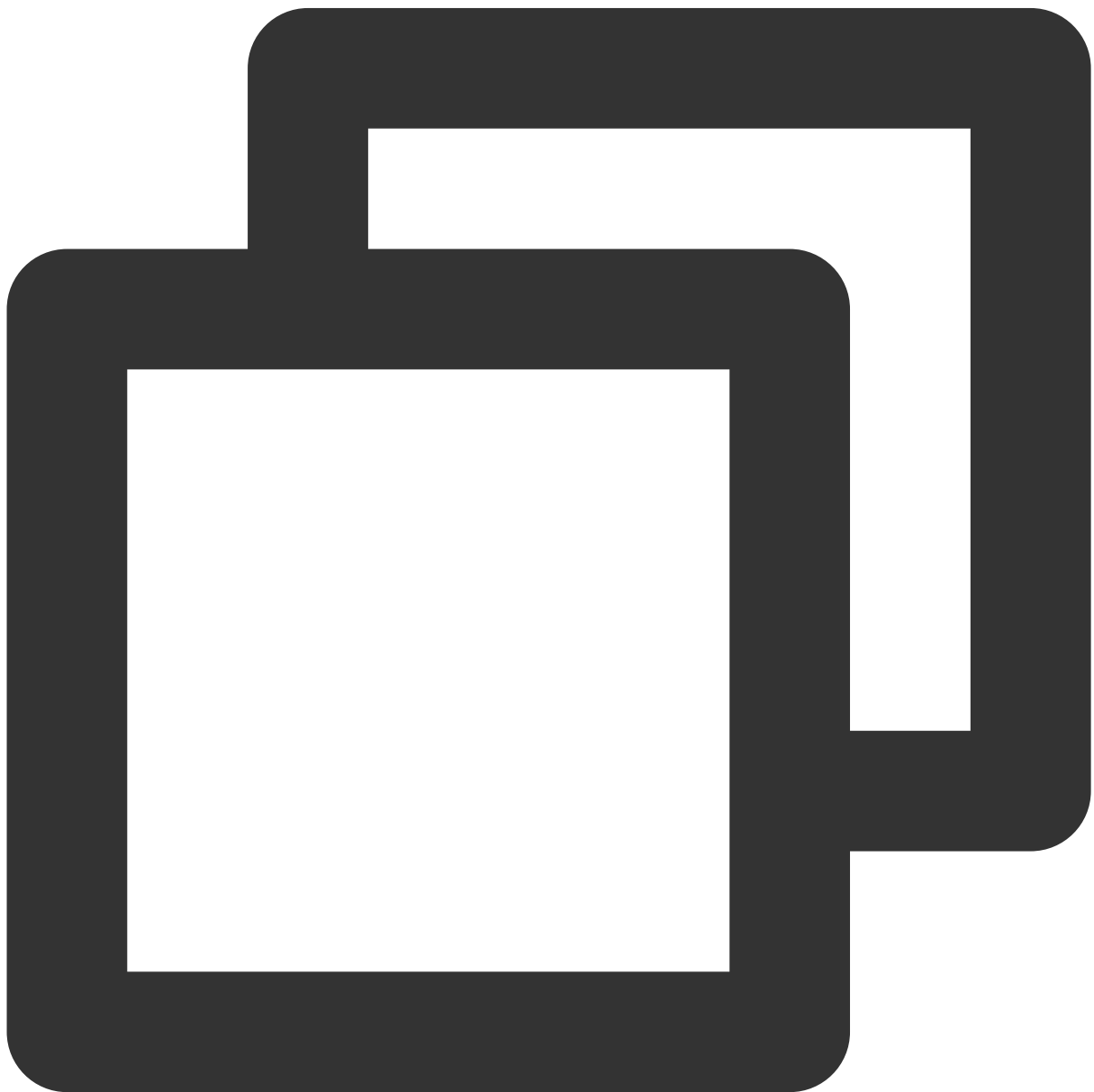
callback	ActionCallback	The callback for the operation.
----------	----------------	---------------------------------

Response parameters:

Parameter	Type	Description
inviteId	String	The invitation ID.

acceptInvitation

This API is used to accept an invitation.



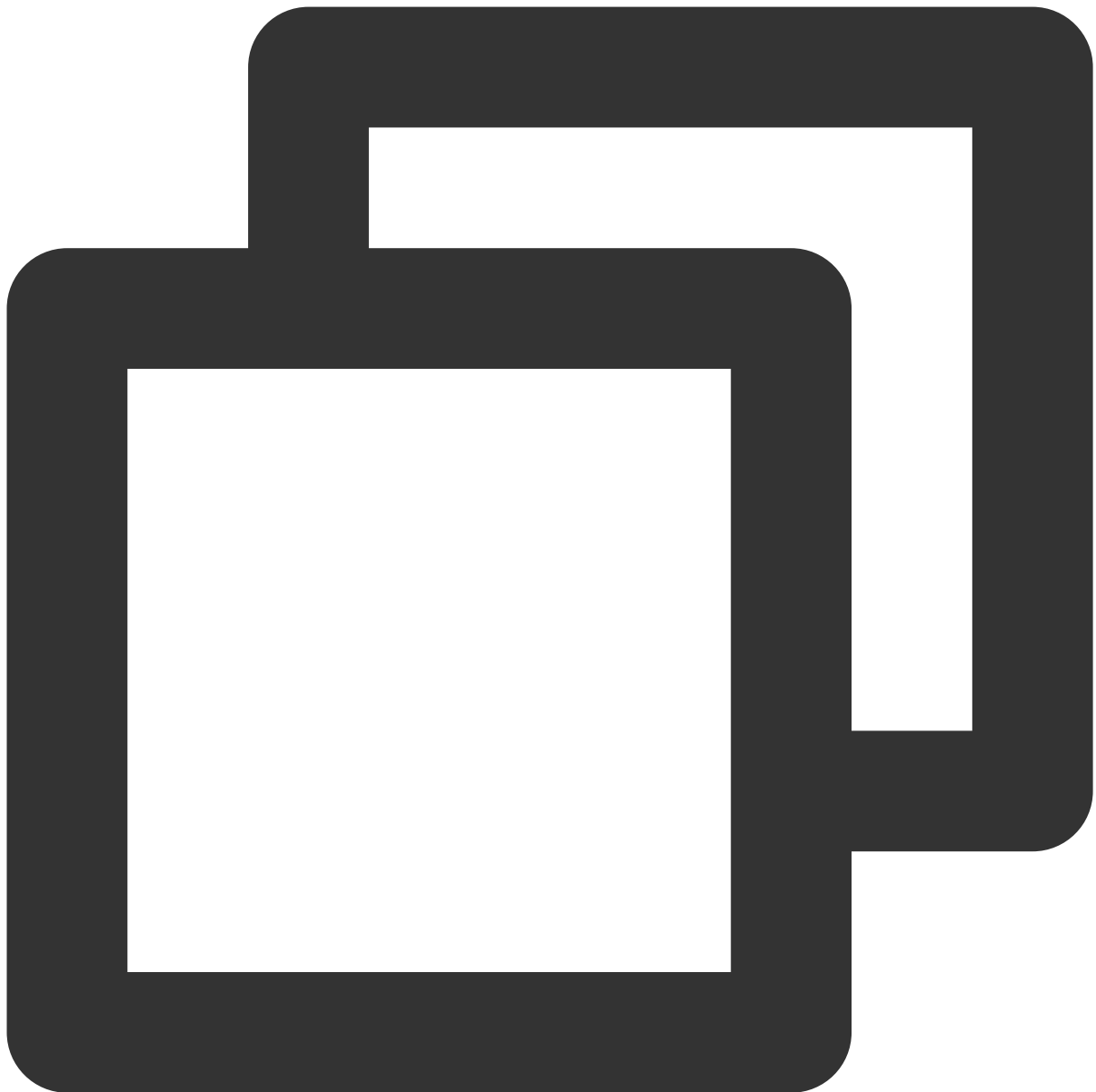
```
public abstract void acceptInvitation(String id, TRTCKaraokeRoomCallback.ActionCall
```

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.



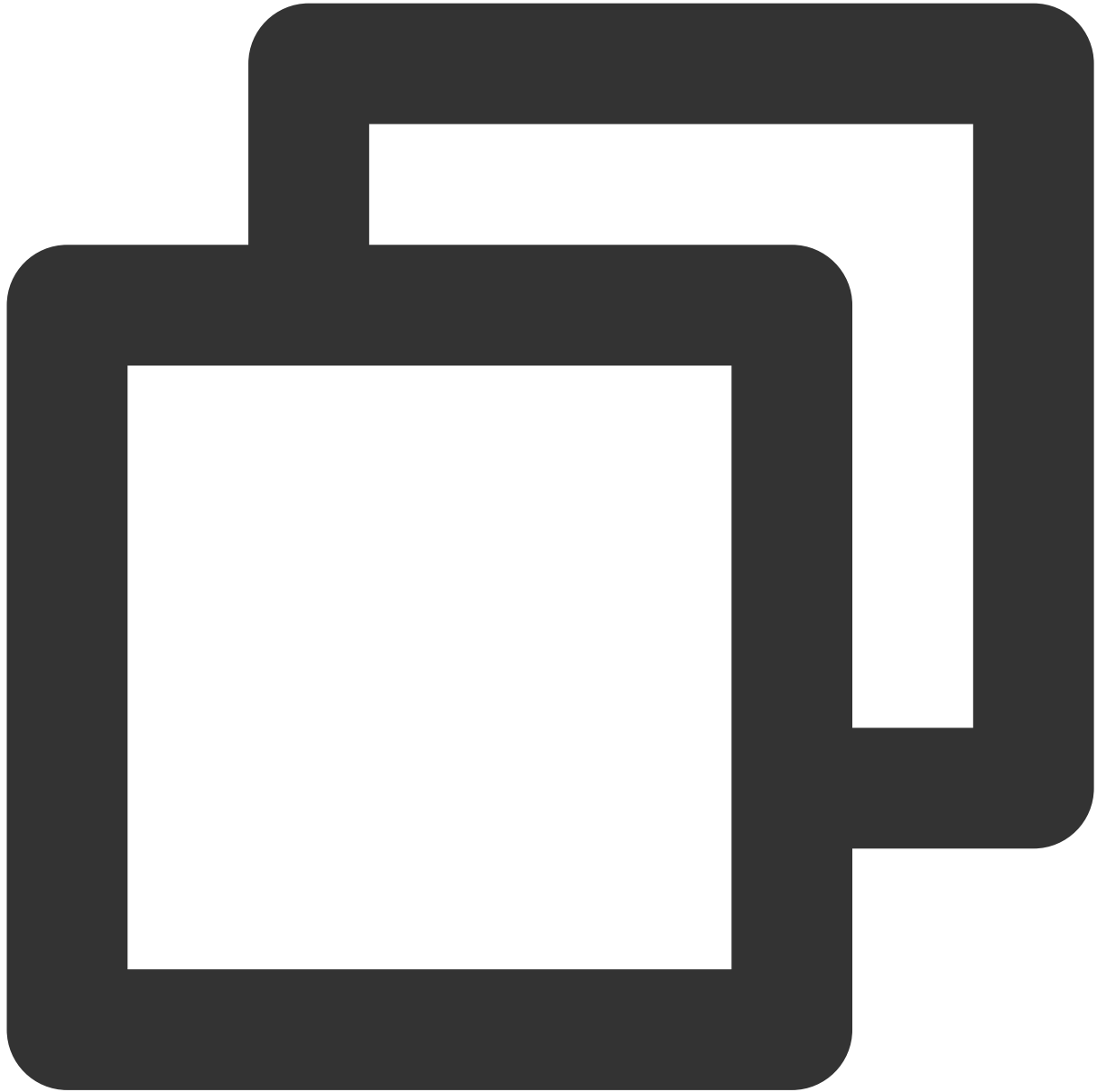
```
public abstract void rejectInvitation(String id, TRTCKaraokeRoomCallback.ActionCall
```

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.



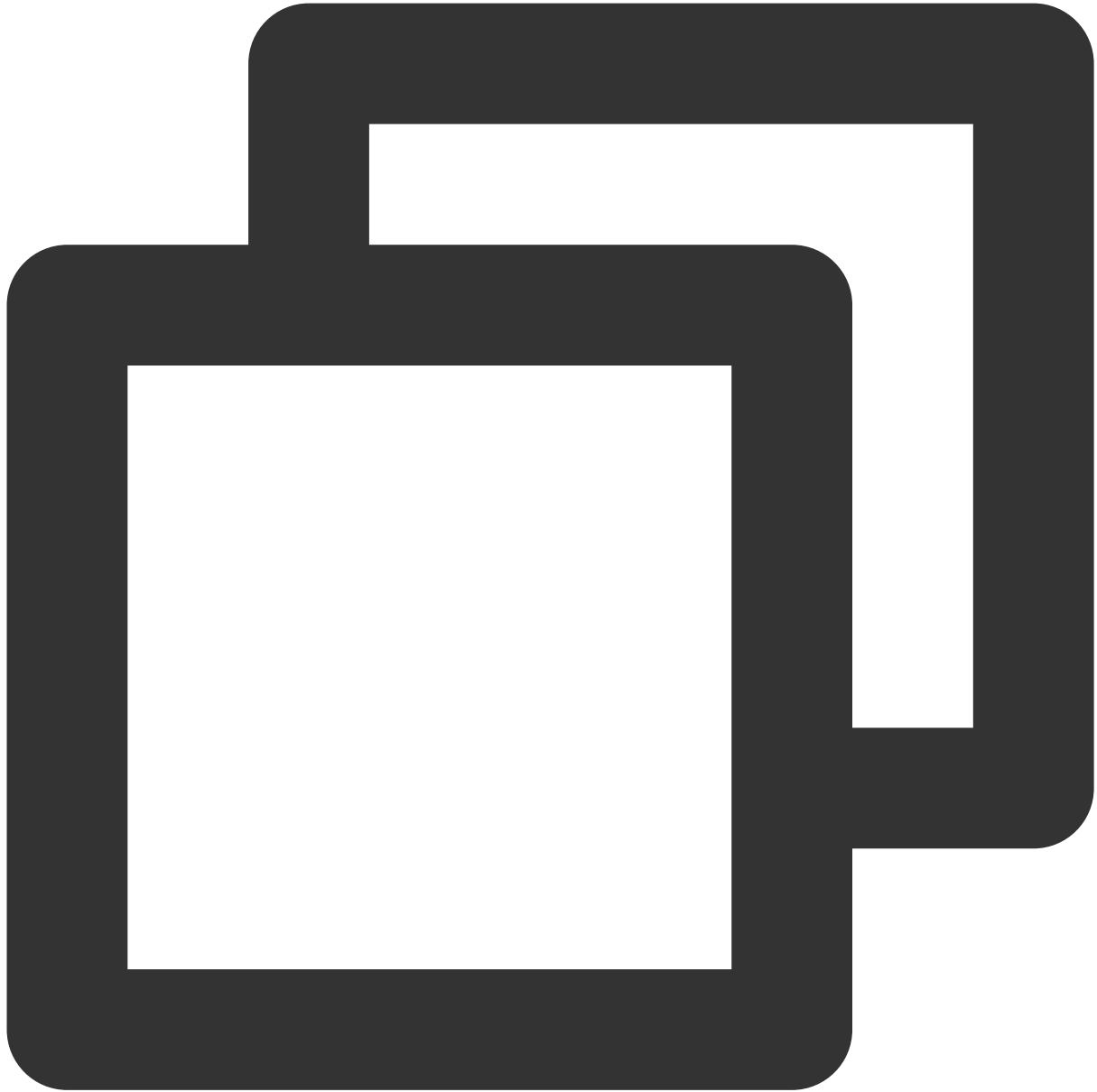
```
void onError(int code, String message);
```

The parameters are described below:

Parameter	Type	Description
code	int	The error code.
message	String	The error message.

onWarning

Callback for warning.



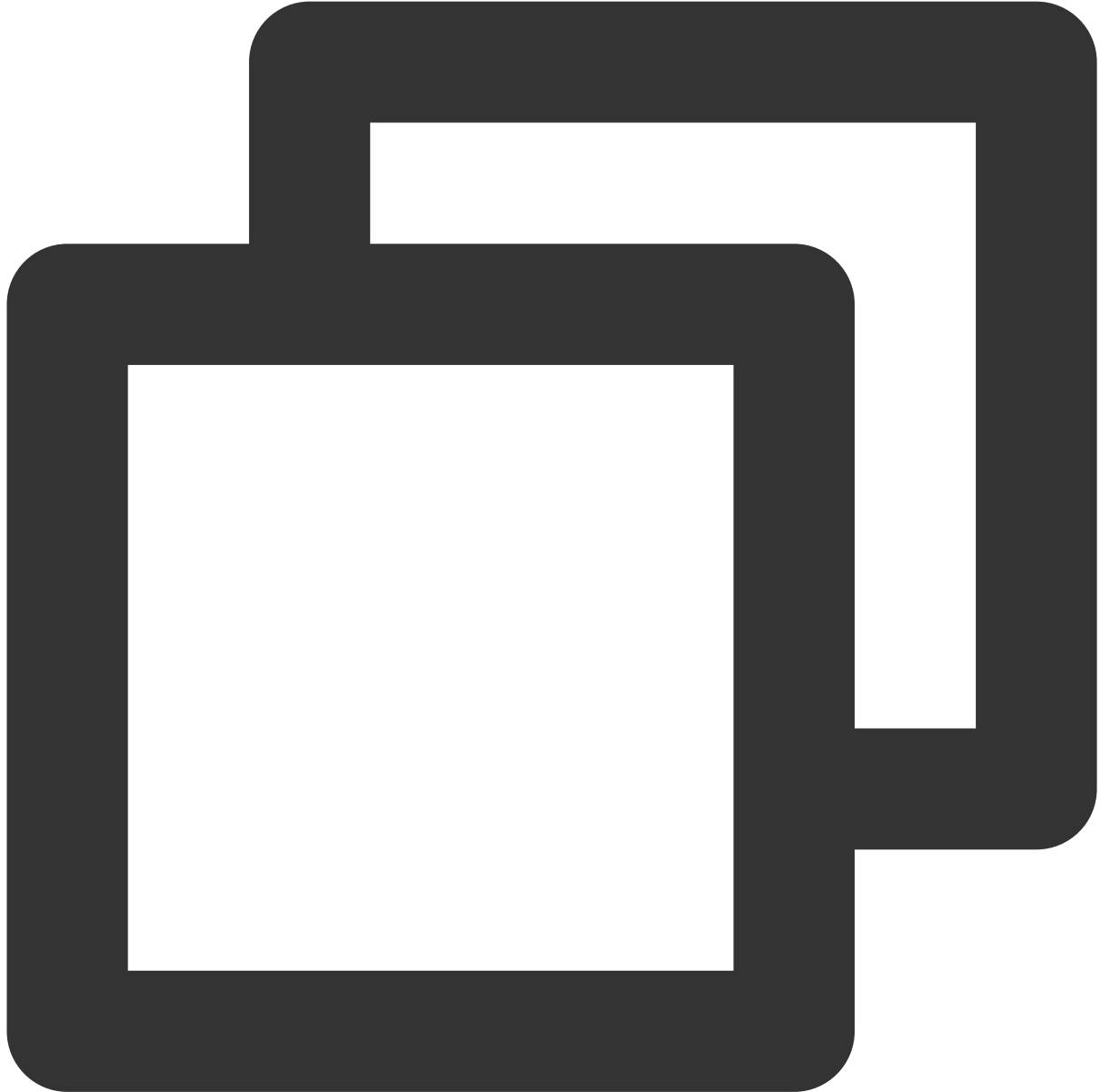
```
void onWarning(int code, String message);
```

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);
```

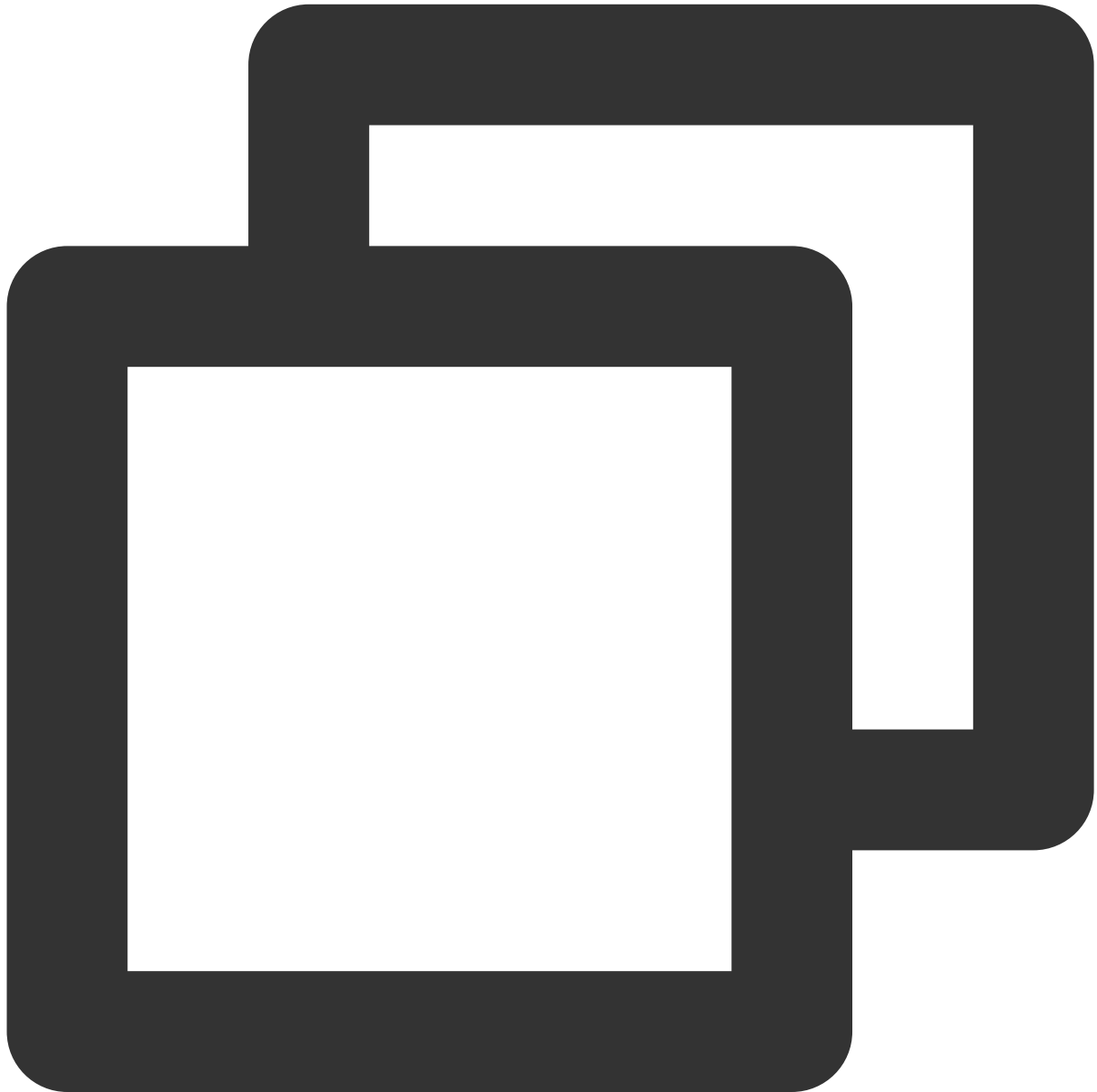
The parameters are described below:

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);
```

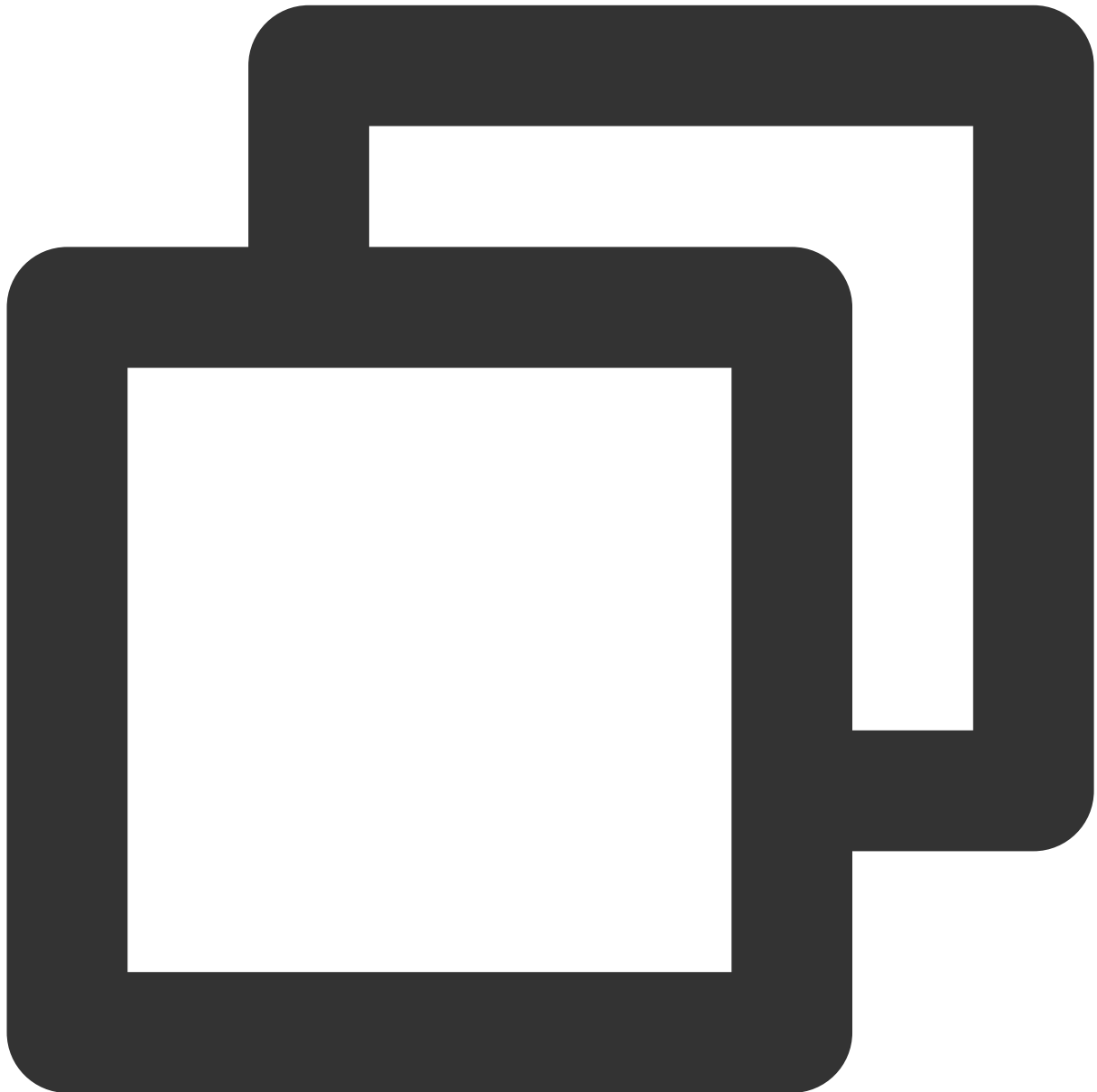
The parameters are described below:

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);
```

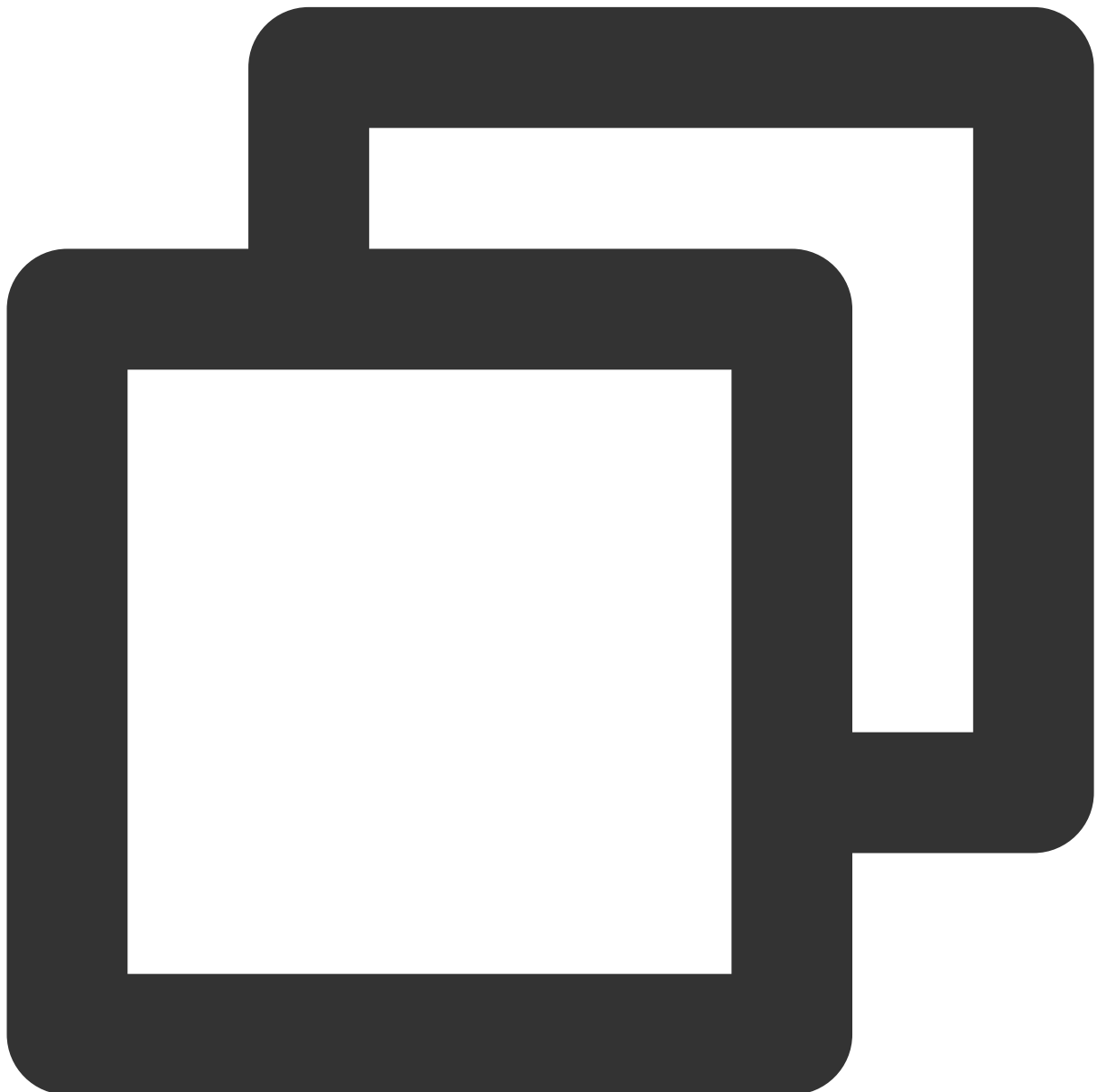
The parameters are described below:

--	--	--

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.



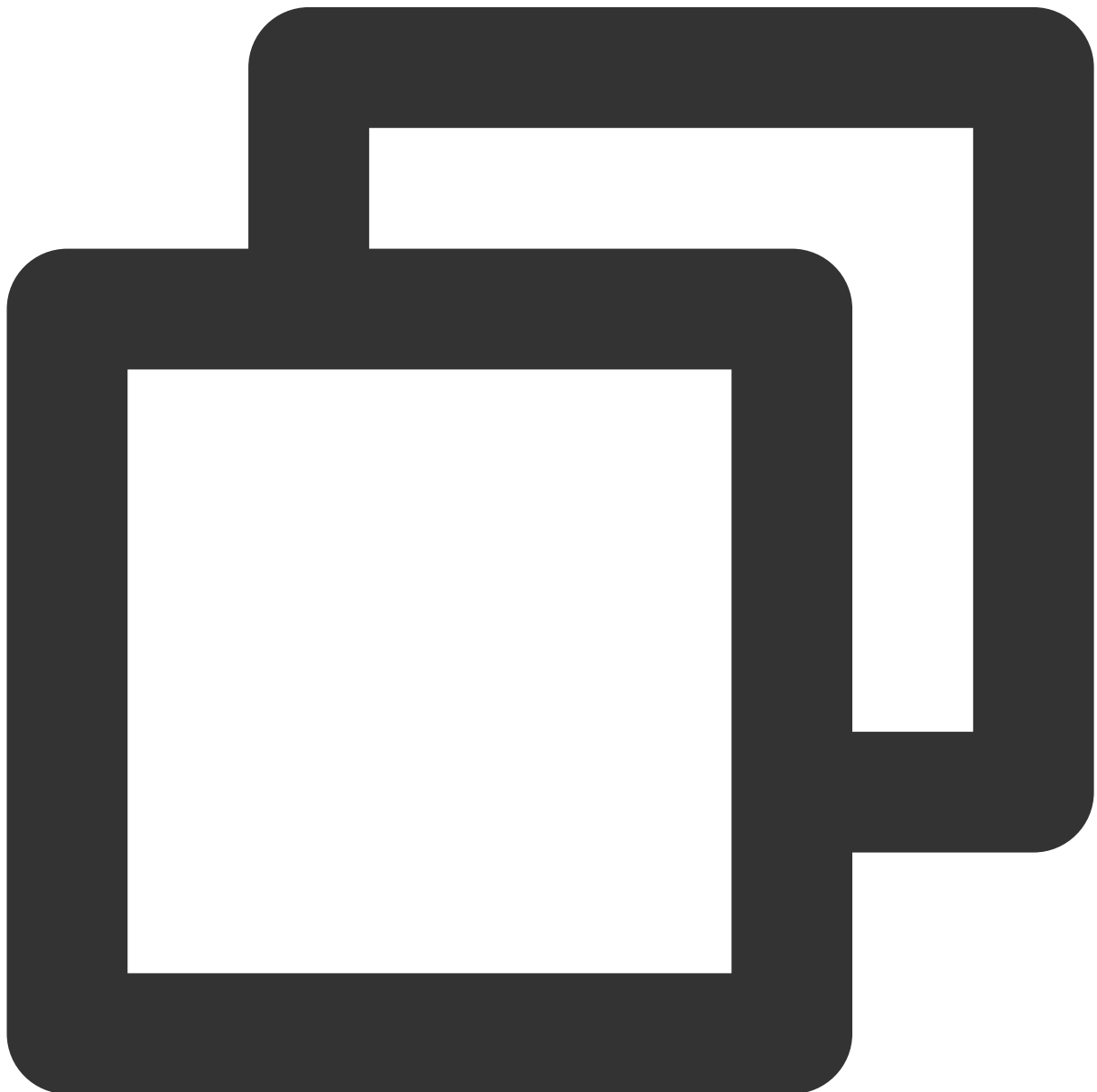
```
void onUserMicrophoneMute(String userId, boolean mute);
```

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
```

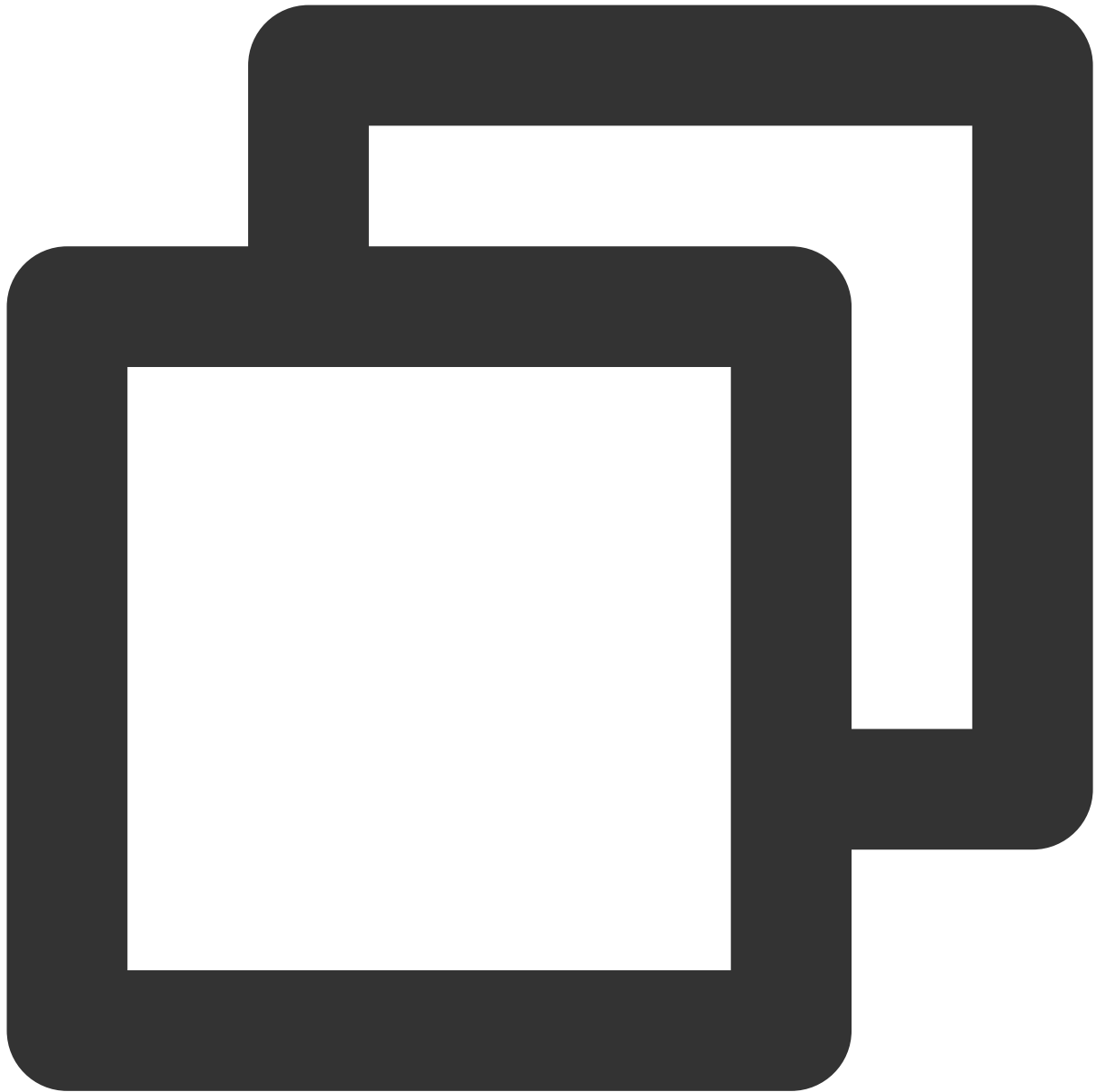
The parameters are described below:

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.



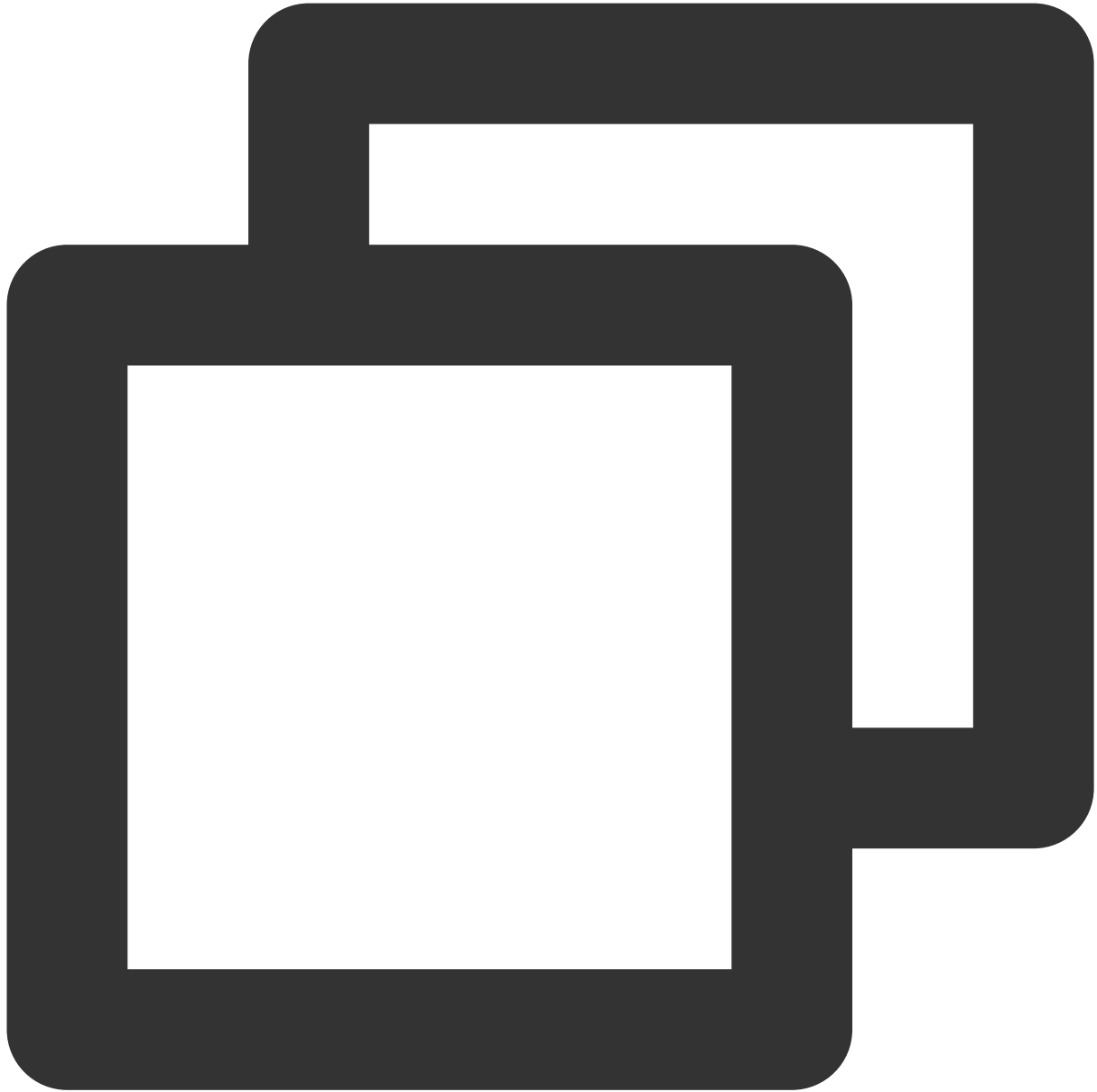
```
void onAnchorEnterSeat(int index, TRTCKaraokeRoomDef.UserInfo user);
```

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.



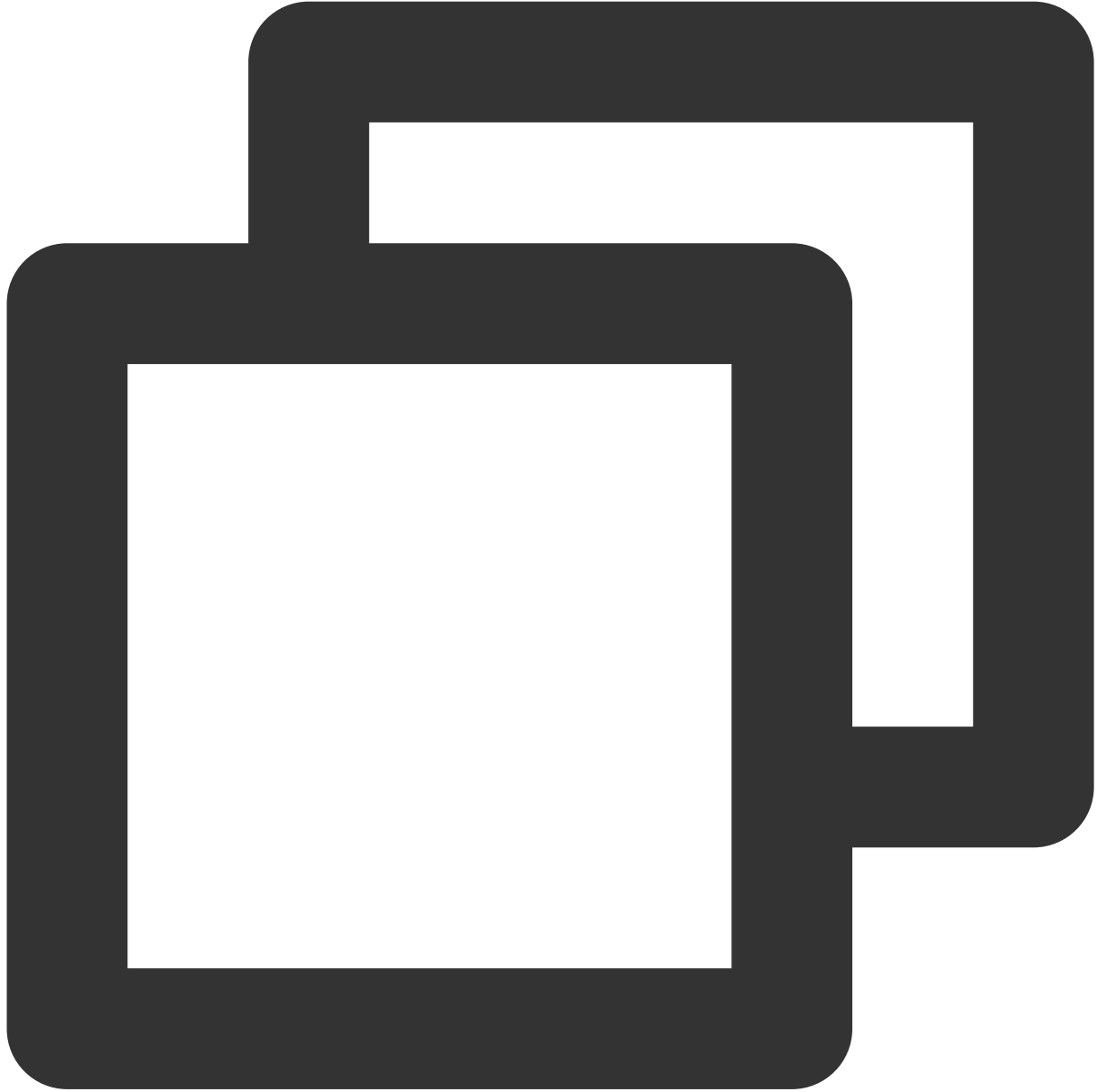
```
void onAnchorLeaveSeat(int index, TRTCKaraokeRoomDef.UserInfo user);
```

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);
```

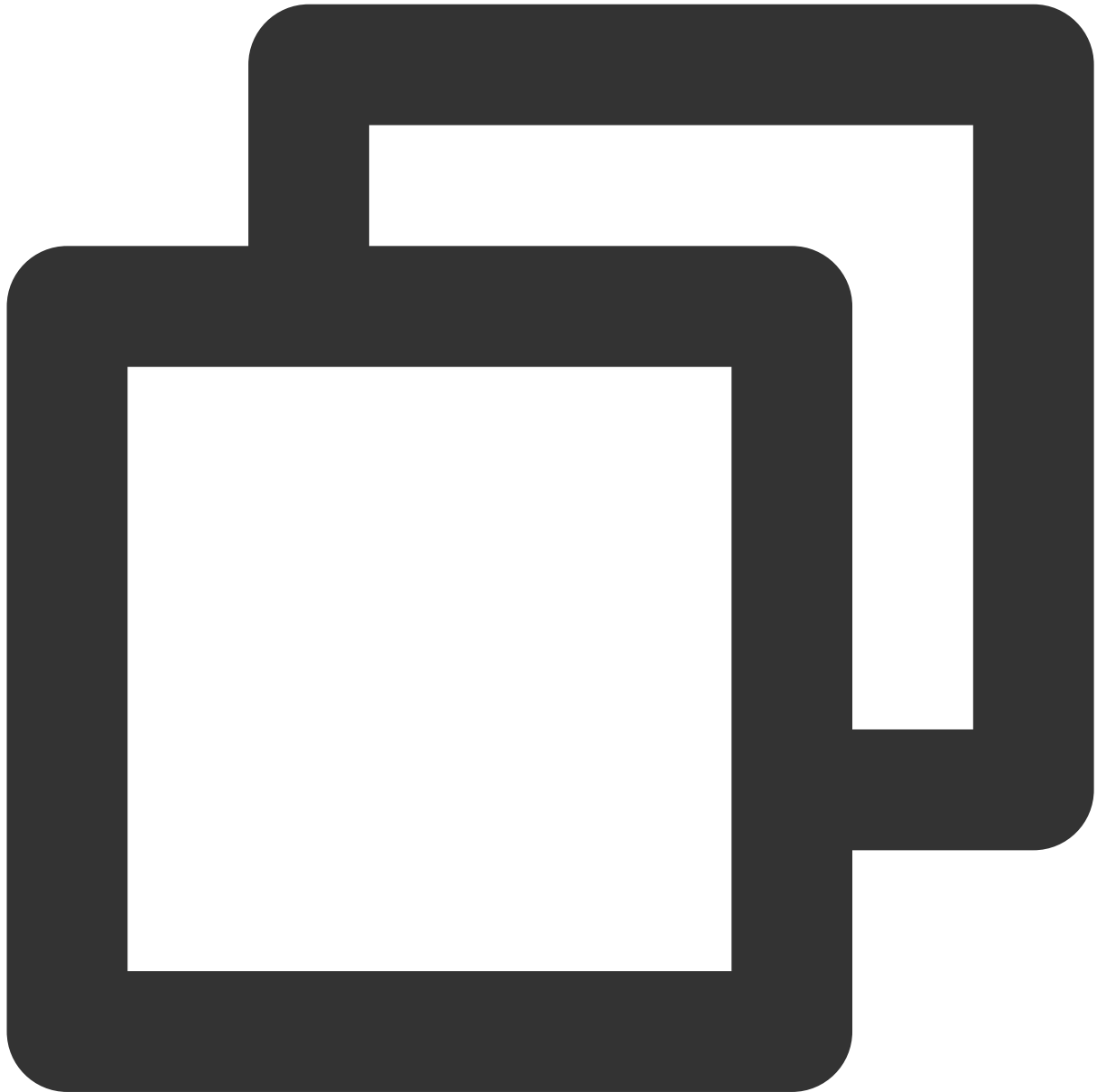
The parameters are described below:

Parameter	Type	Description
index	int	The seat muted/unmuted.
isMute	boolean	

```
true : Muted; false : Unmuted
```

onSeatClose

The room owner blocked/unblocked a seat.



```
void onSeatClose(int index, boolean isClose);
```

The parameters are described below:

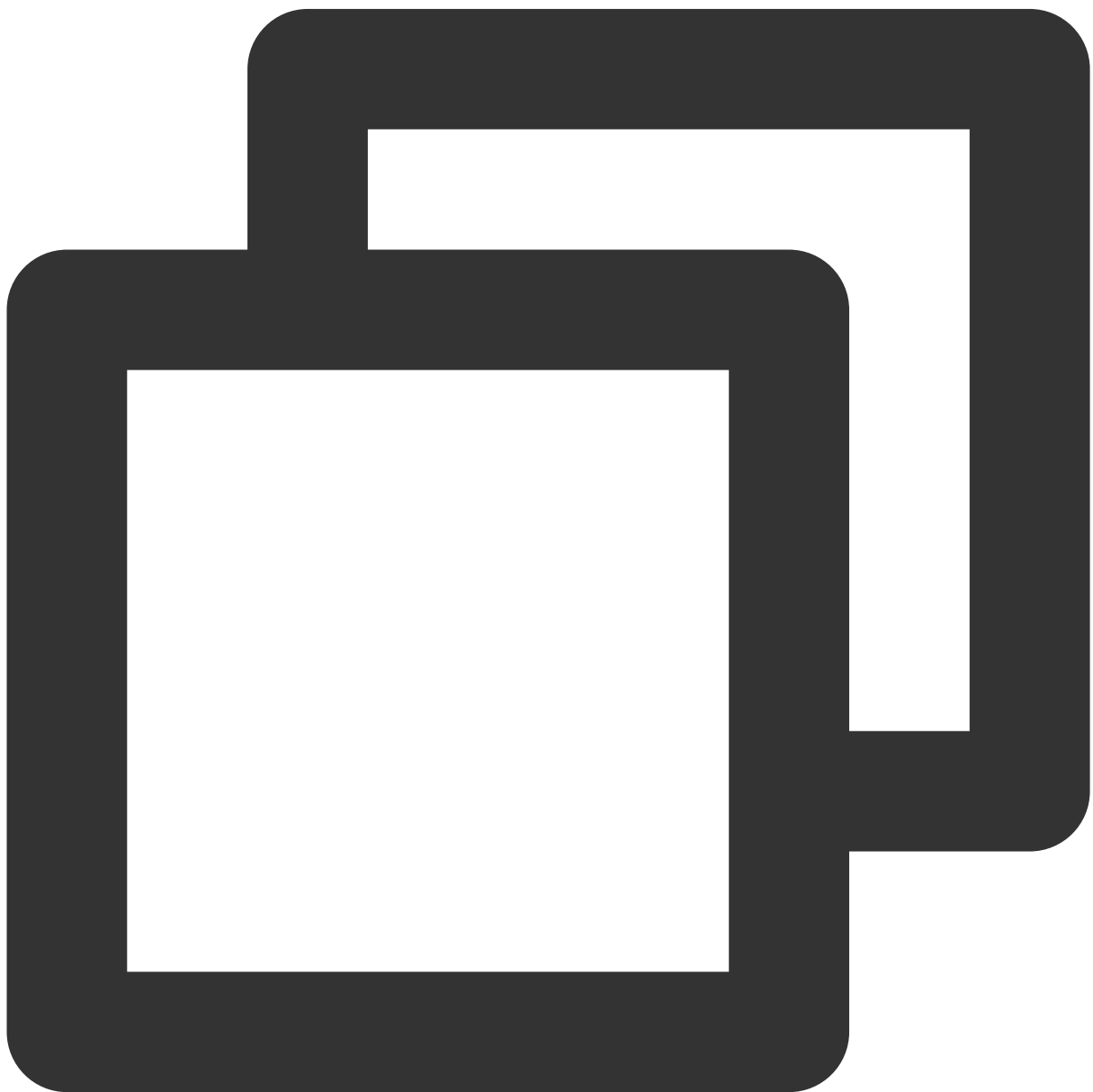
Parameter	Type	Description

index	int	The seat blocked/unblocked.
isClose	boolean	<code>true</code> : Blocked; <code>false</code> : Unblocked

Callback APIs for Room Entry/Exit by Listener

onAudienceEnter

A listener entered the room.



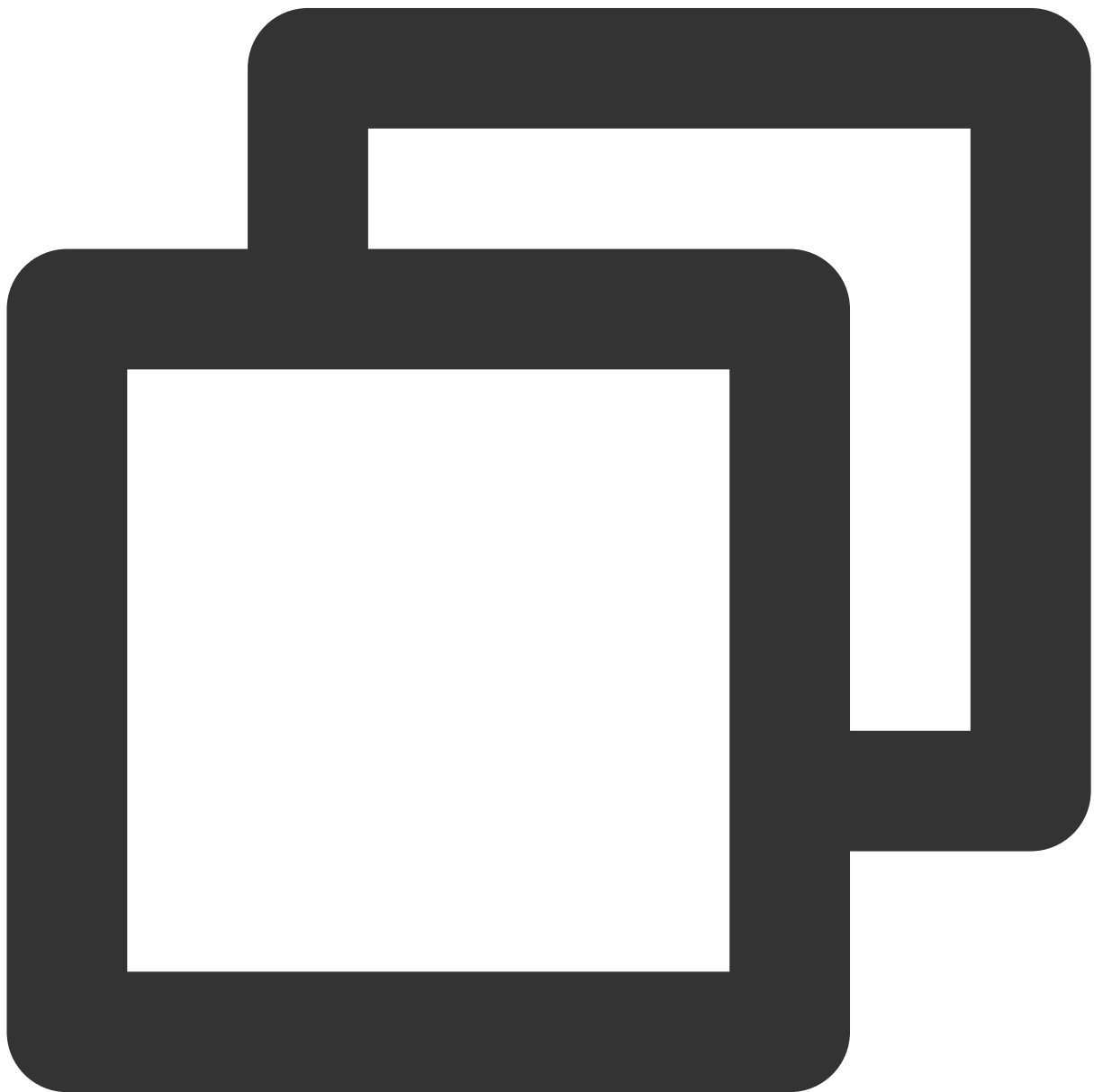
```
void onAudienceEnter (TRTCKaraokeRoomDef.UserInfo userInfo);
```

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);
```

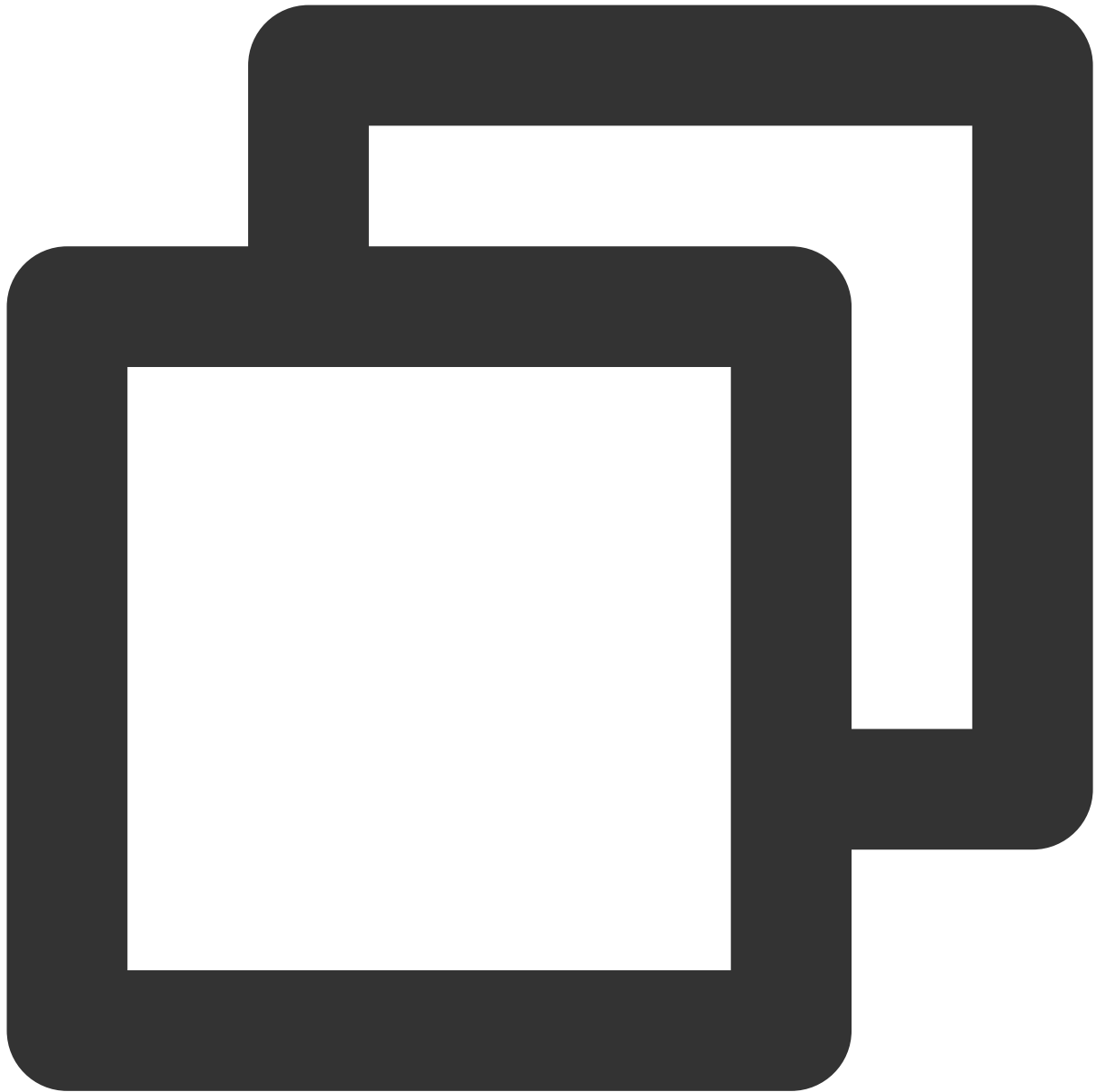
The parameters are described below:

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.



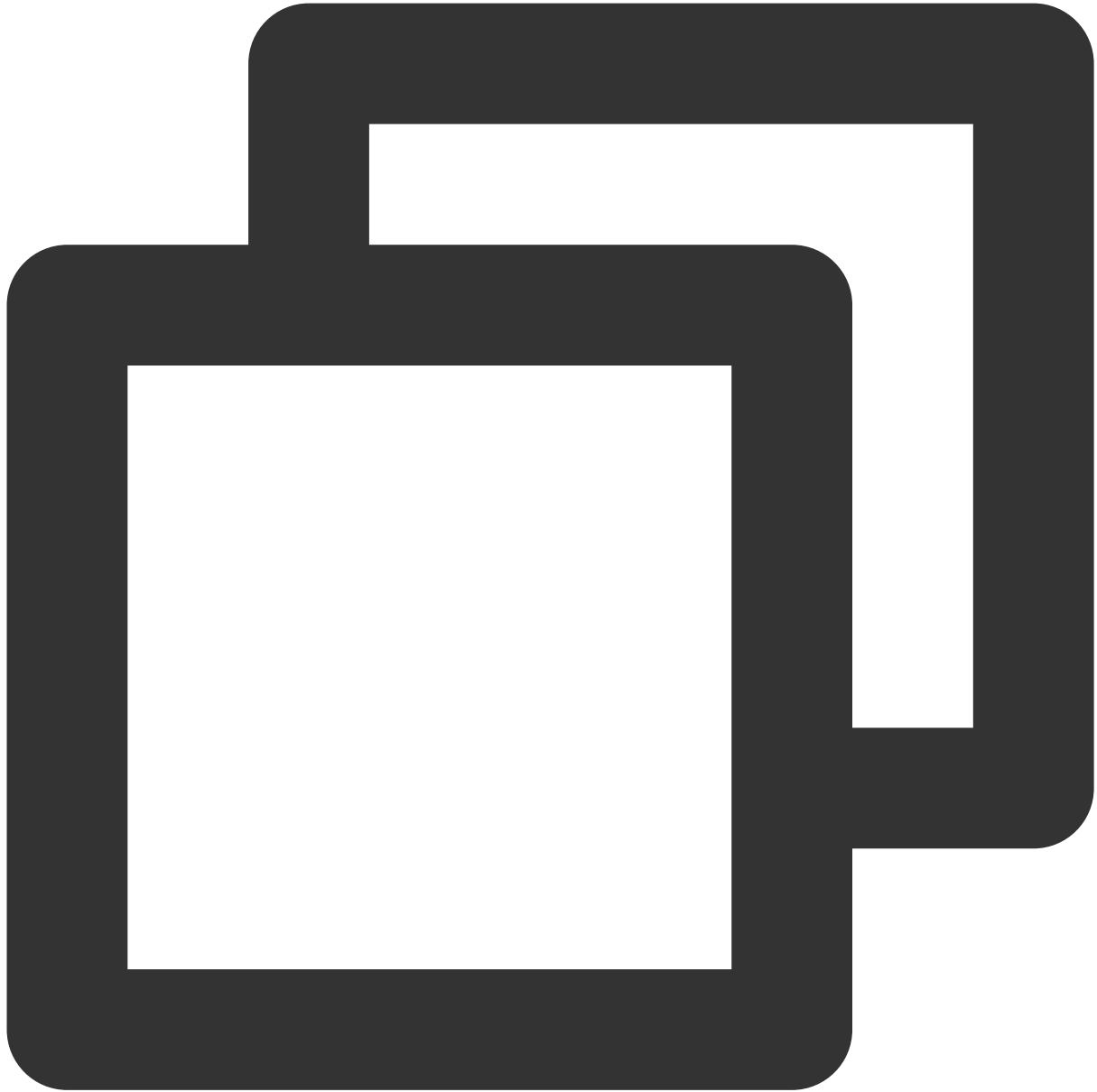
```
void onRecvRoomTextMsg(String message, TRTCKaraokeRoomDef.UserInfo userInfo);
```

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

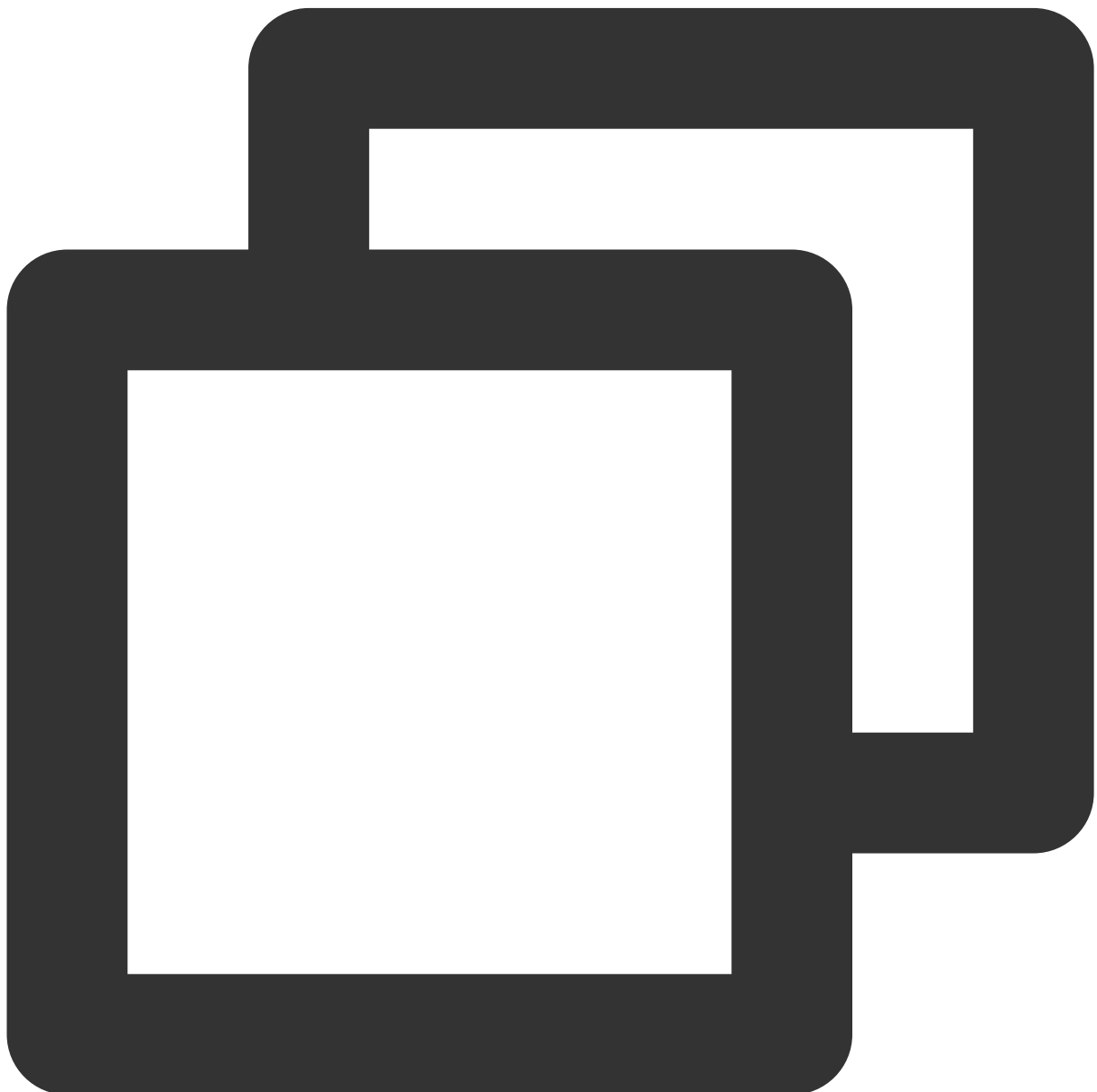
UserInfo

Information of the sender.

Invitation Signaling Callback APIs

onReceiveNewInvitation

An invitation was received.



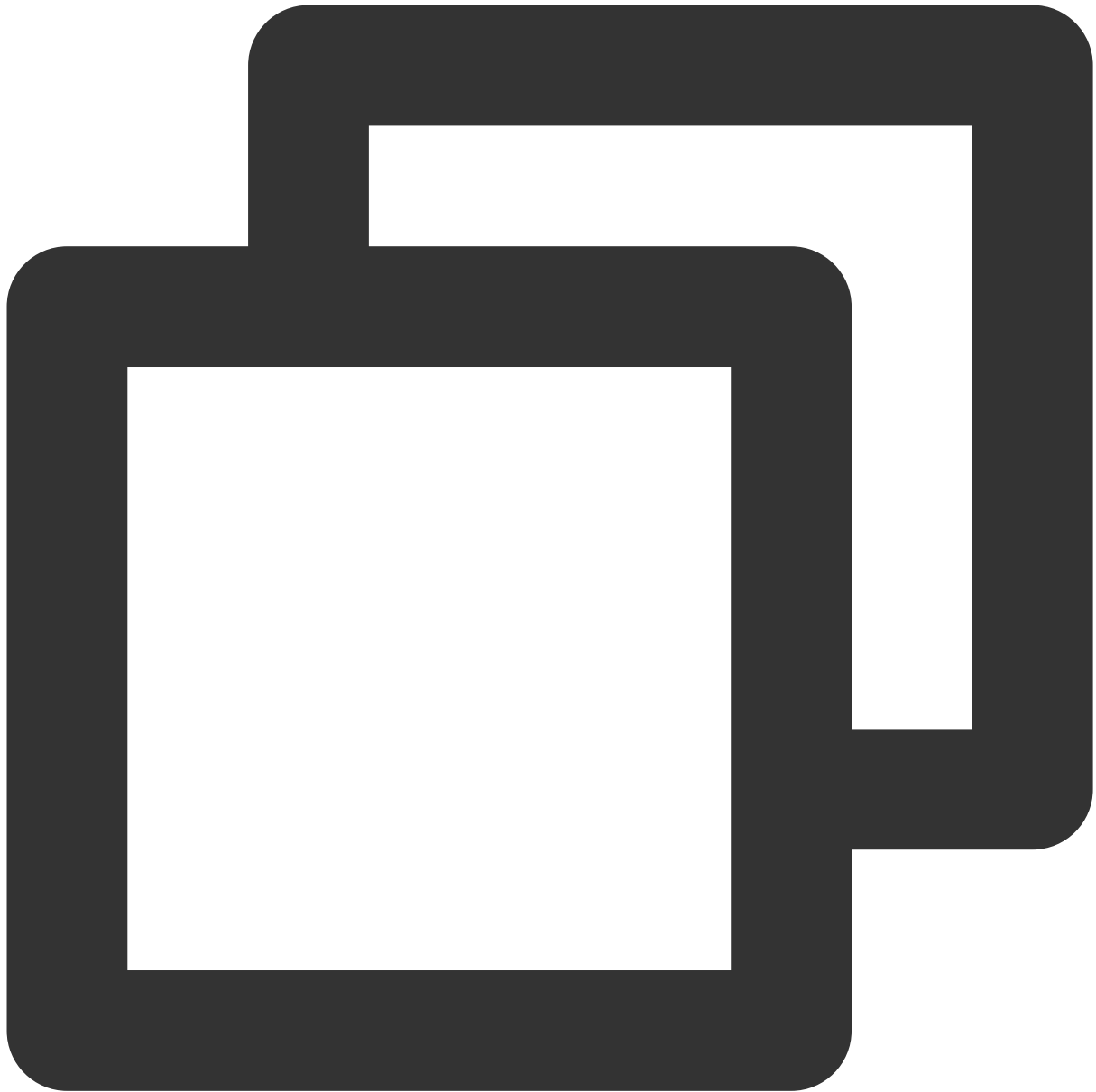
```
void onReceiveNewInvitation(String id, String inviter, String cmd, String content);
```

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.



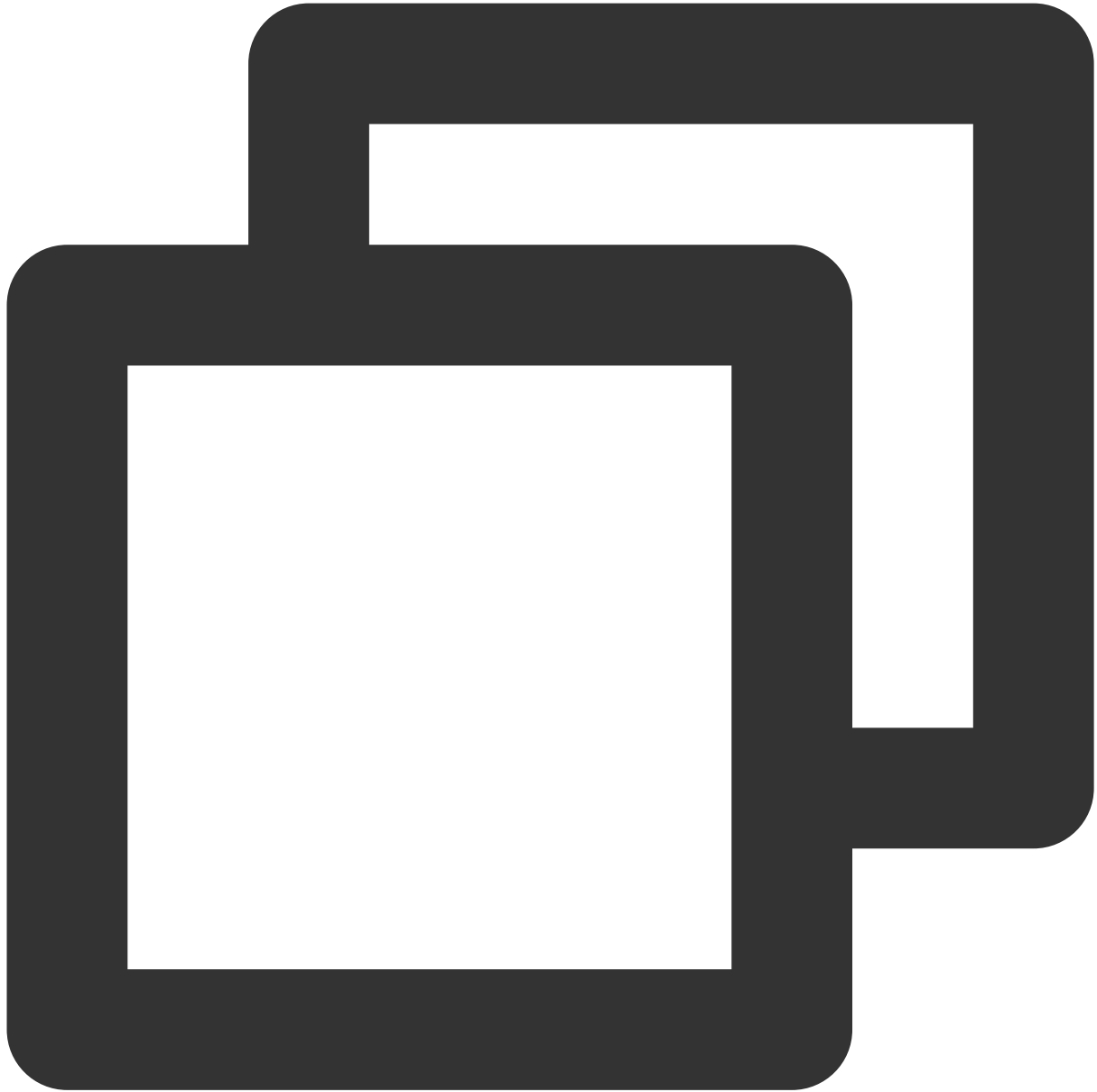
```
void onInviteeAccepted(String id, String invitee);
```

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.



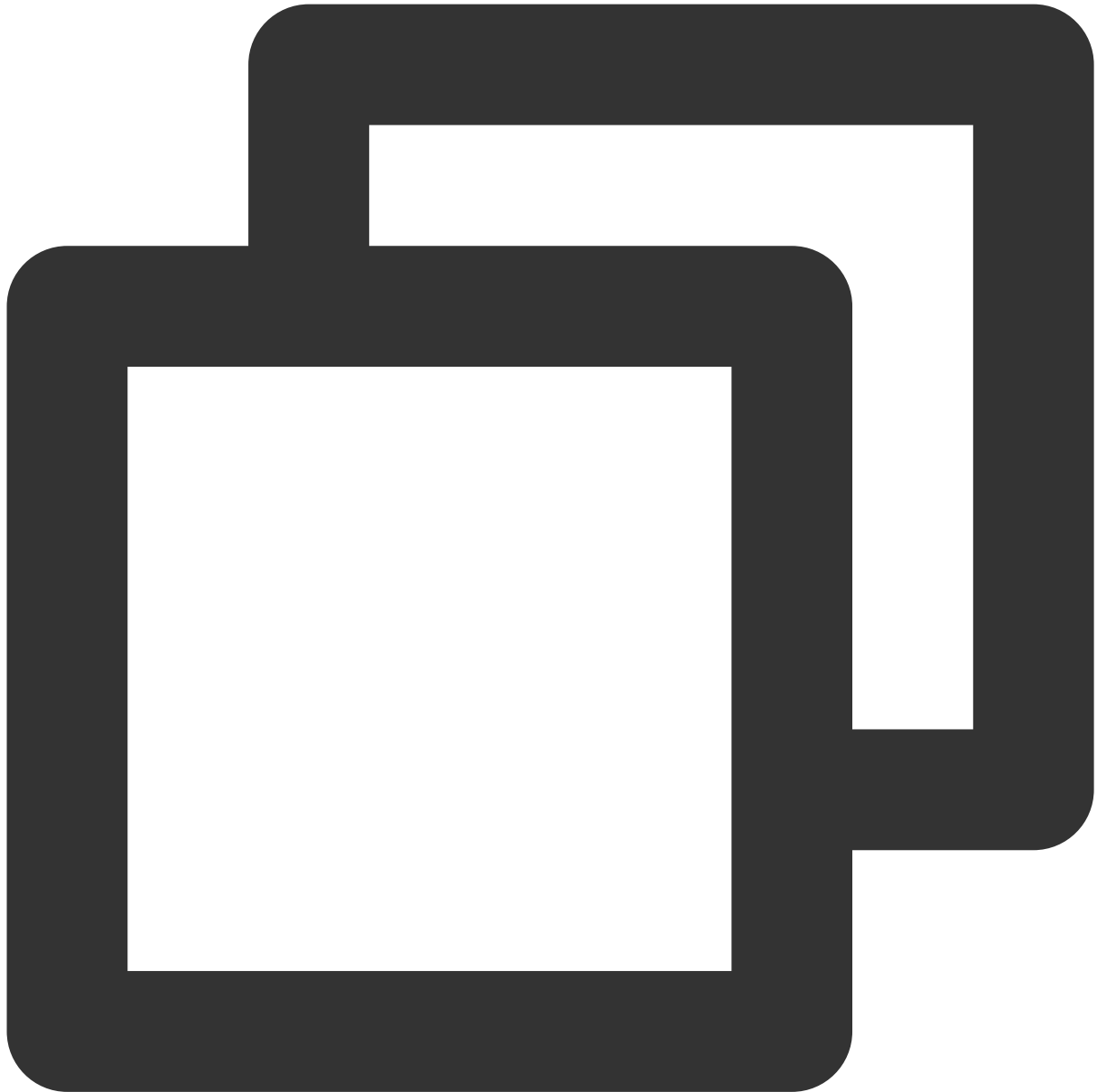
```
void onInviteeRejected(String id, String invitee);
```

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);
```

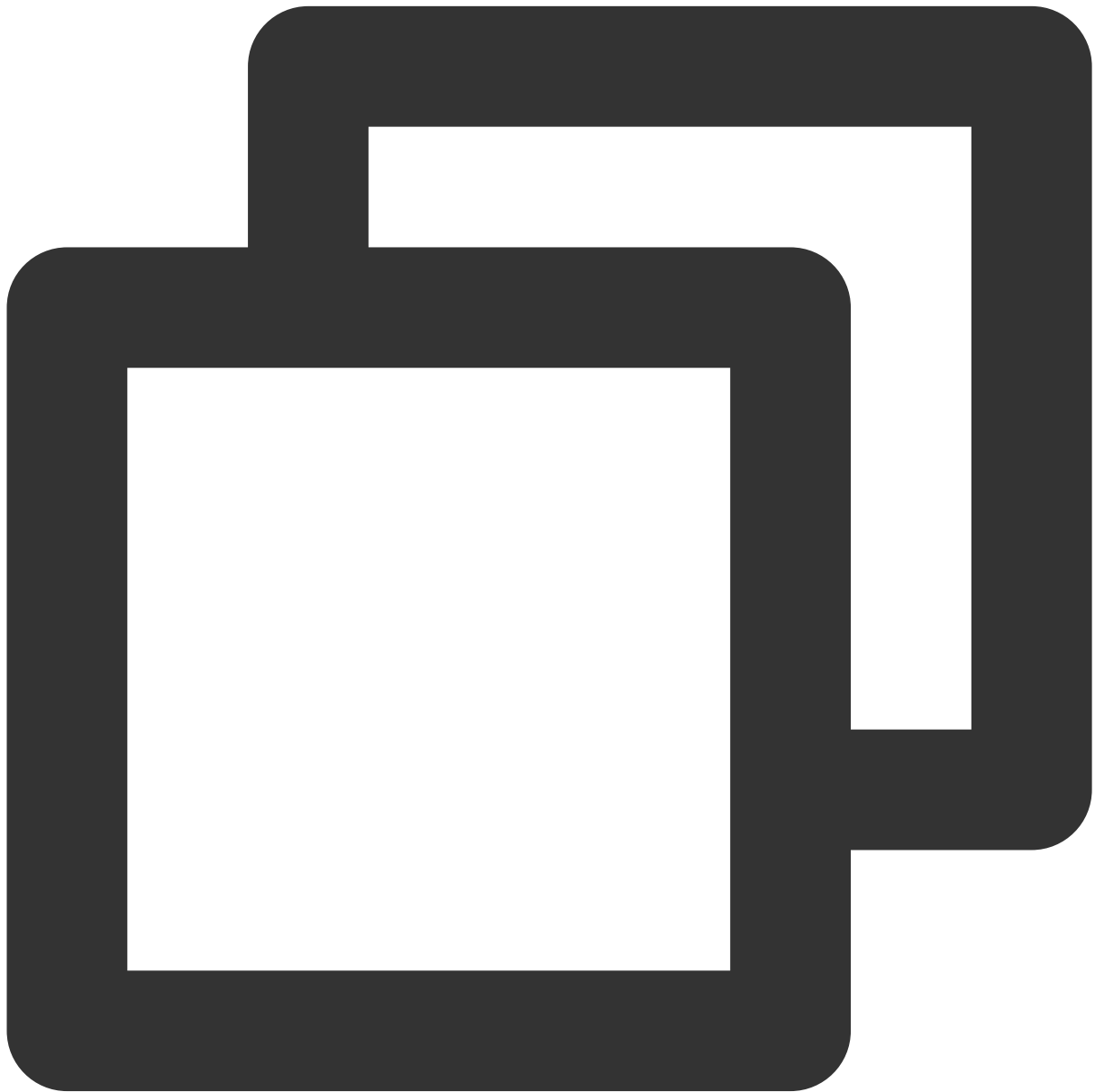
The parameters are described below:

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.



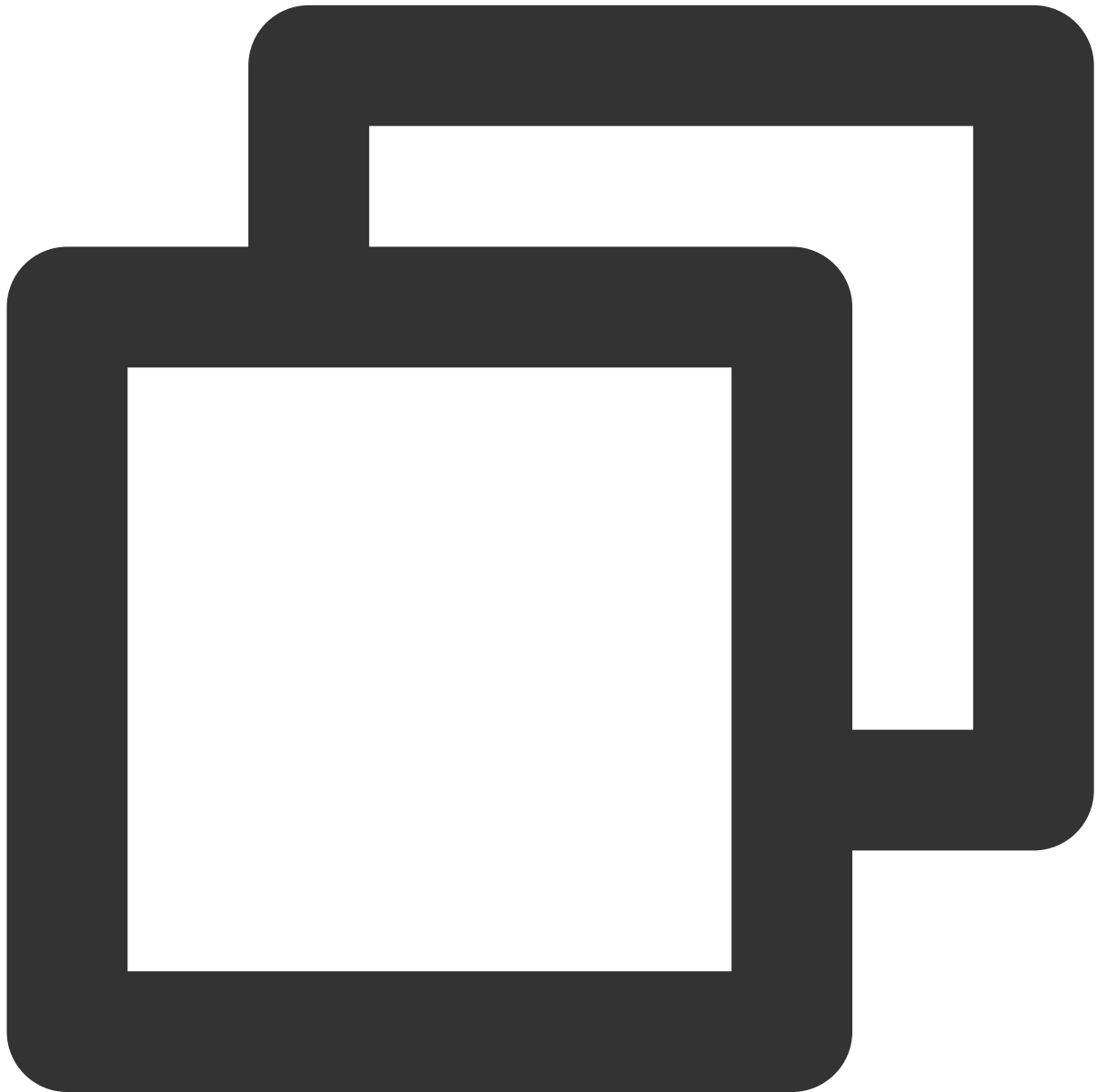
```
void onMusicPrepareToPlay(int musicID);
```

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);
```

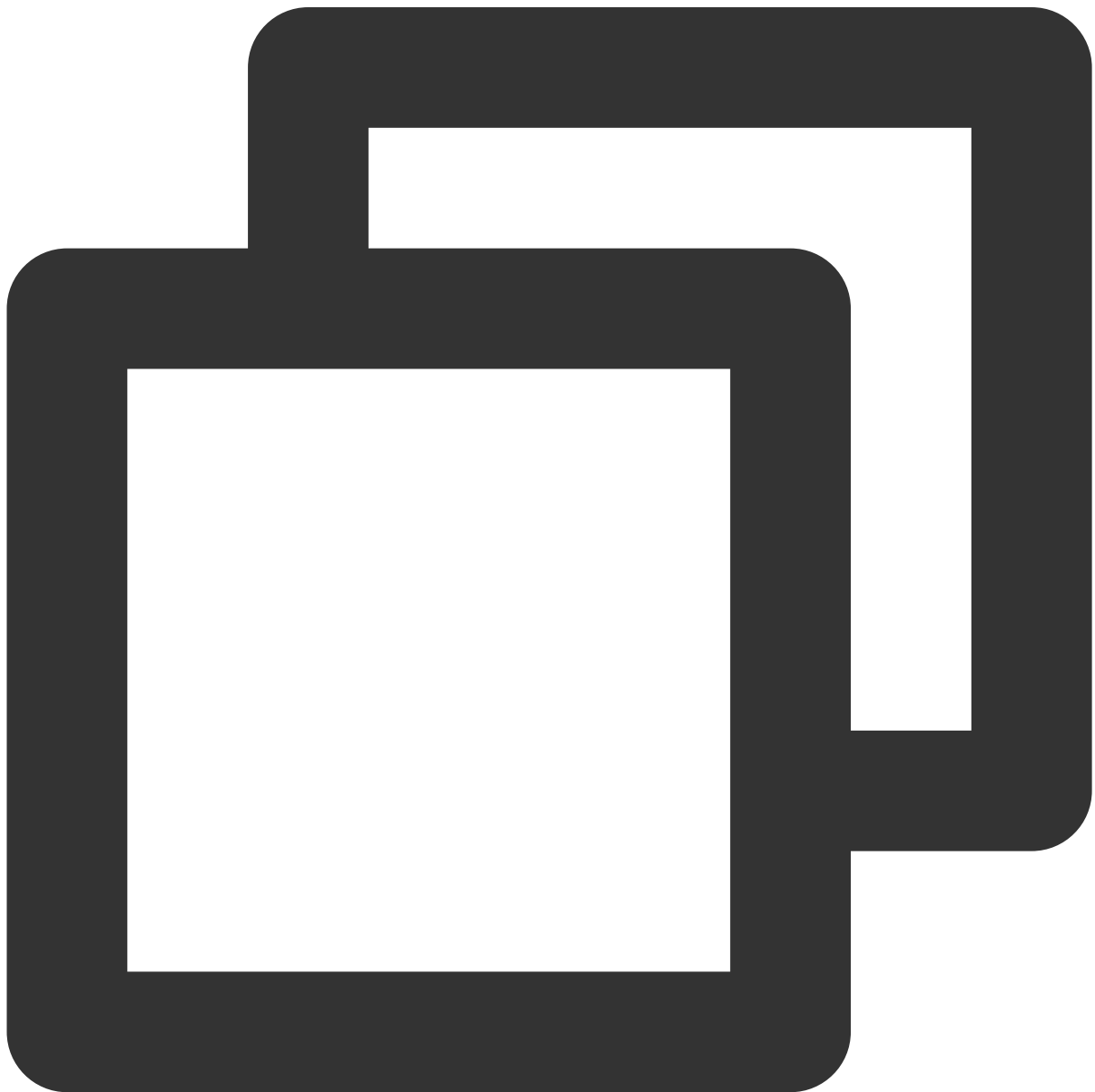
The parameters are described below:

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

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.

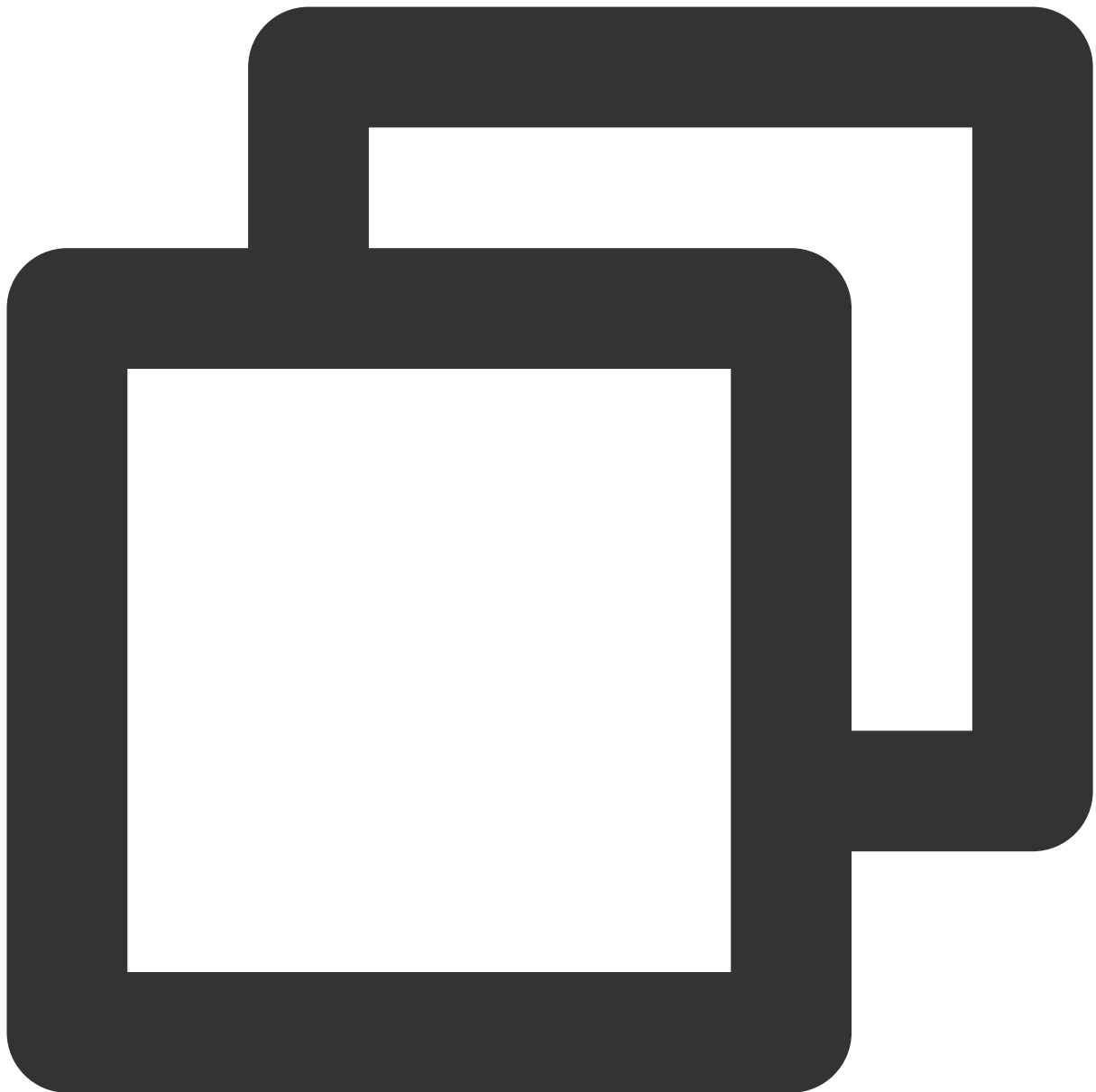
FAQs

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];
```

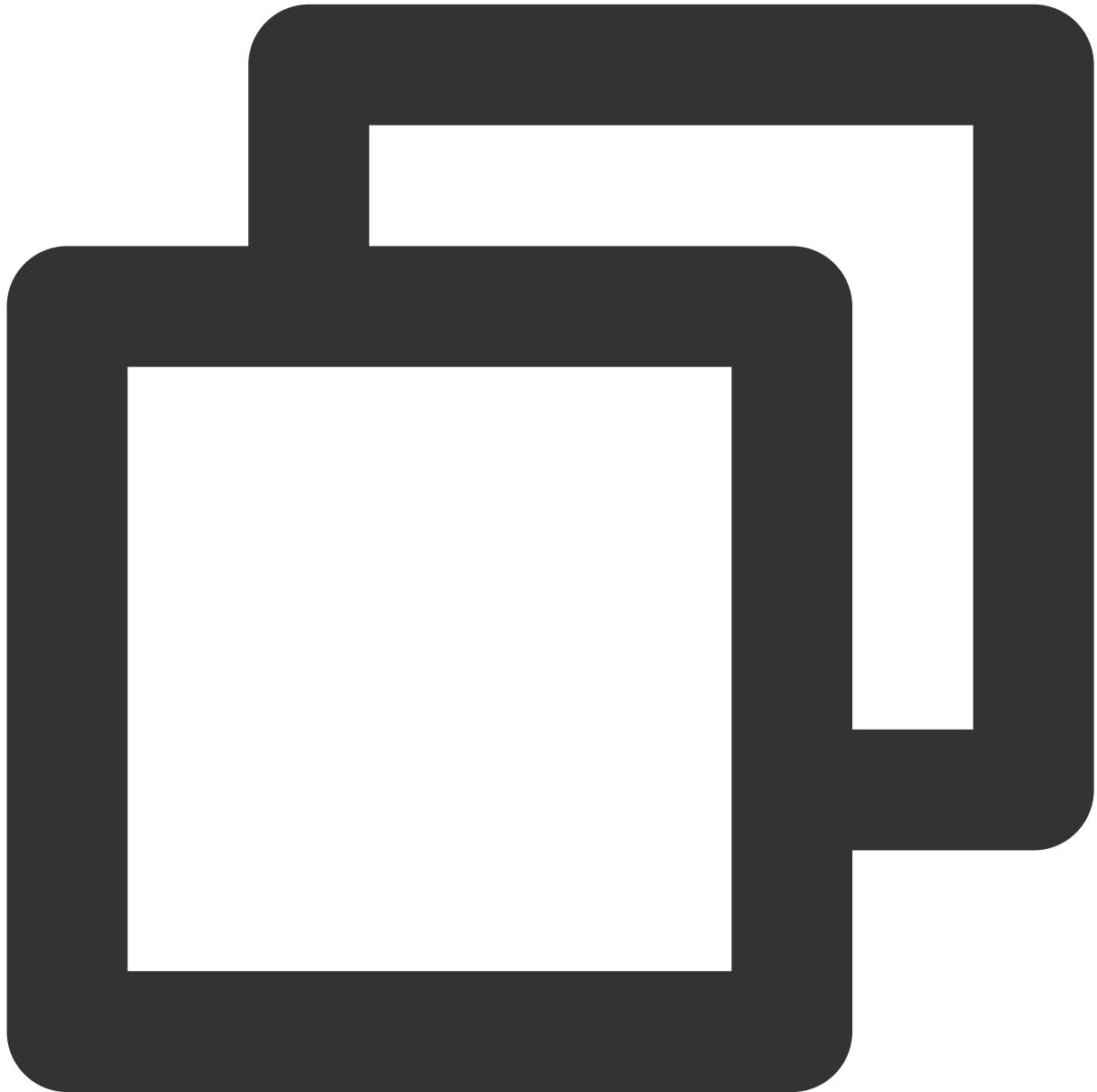
What if there is no effect after enabling the ear monitoring function?

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
NSMutableDictionary *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

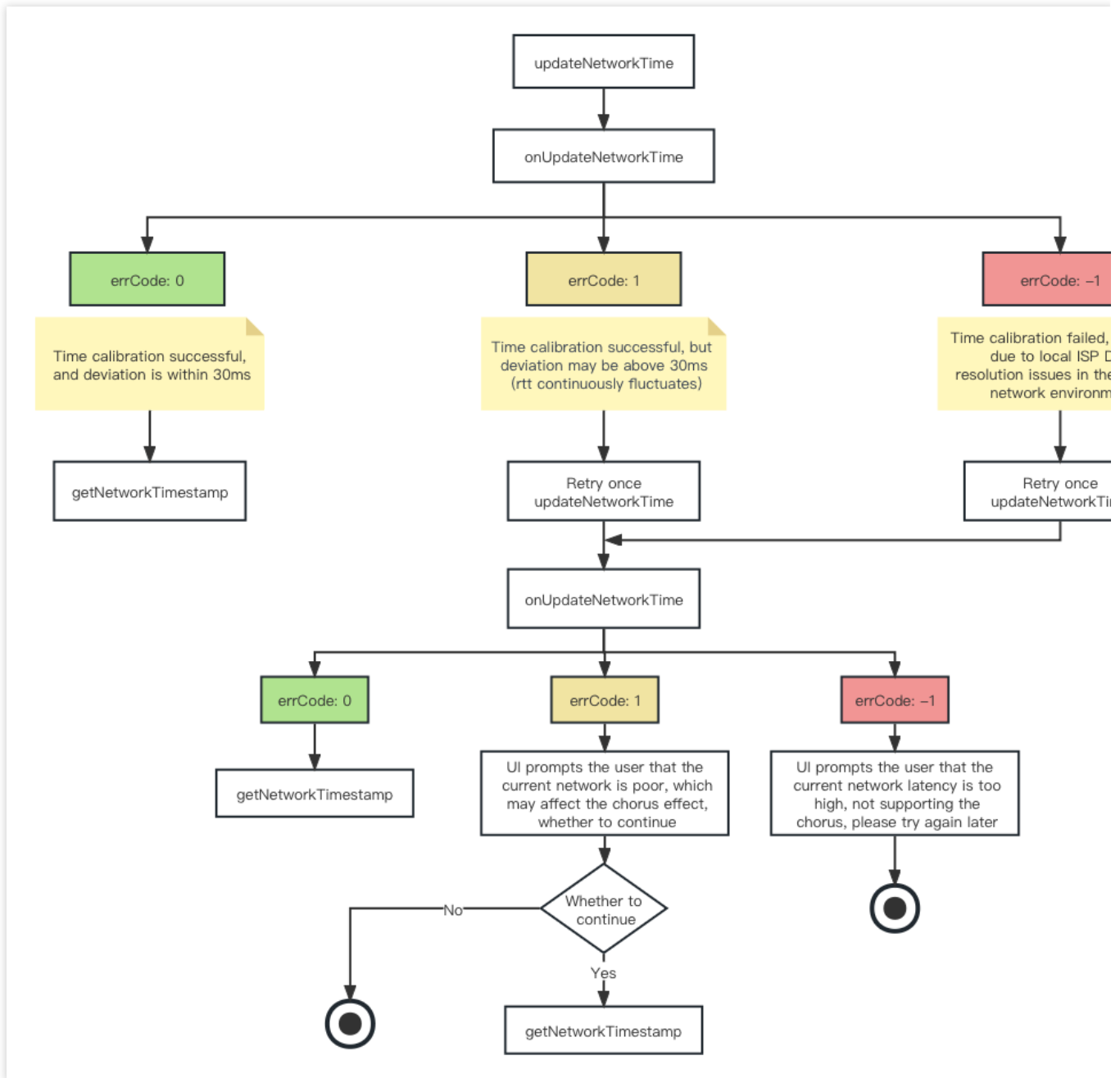
Reminder: “NTP time sync finished, but result maybe inaccurate” ?

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.

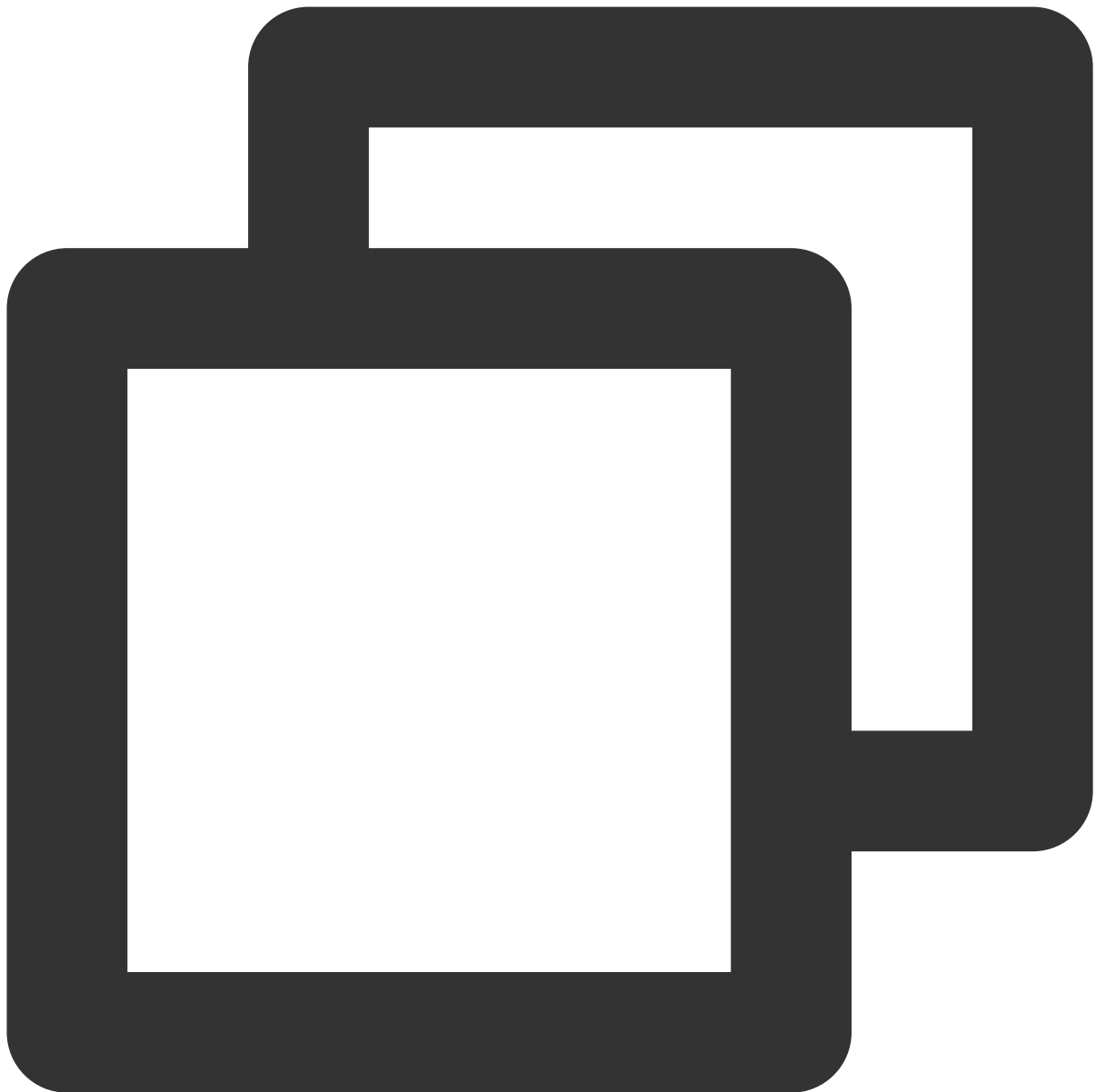
NTP service retry processing logic?



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:



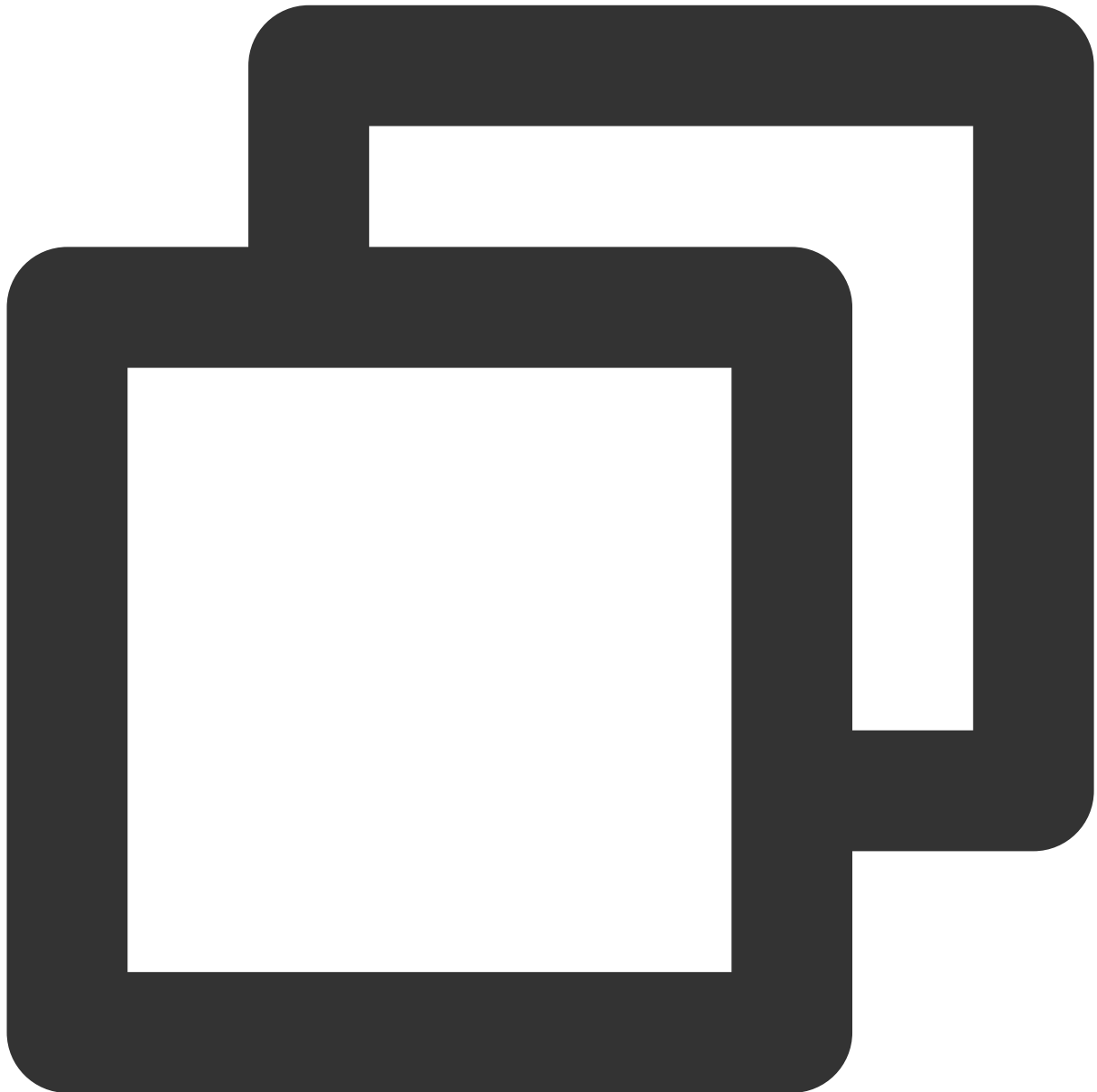
```
TRTCSpeedTestParams *speedTestParams = [[TRTCSpeedTestParams alloc] init];
speedTestParams.sdkAppId = SDK_APP_ID;
speedTestParams.userId = userId;
speedTestParams.userSig = userSig;
// If the actual bandwidth is higher than the expected value, the test result is th
// if the actual bandwidth is lower 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 latency:%@
    [mTextTestResult addObject:[NSString stringWithFormat:@"downlink bandwidth:
    [mTextTestResult addObject:[NSString stringWithFormat:@"uplink bandwidth:%@
    [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

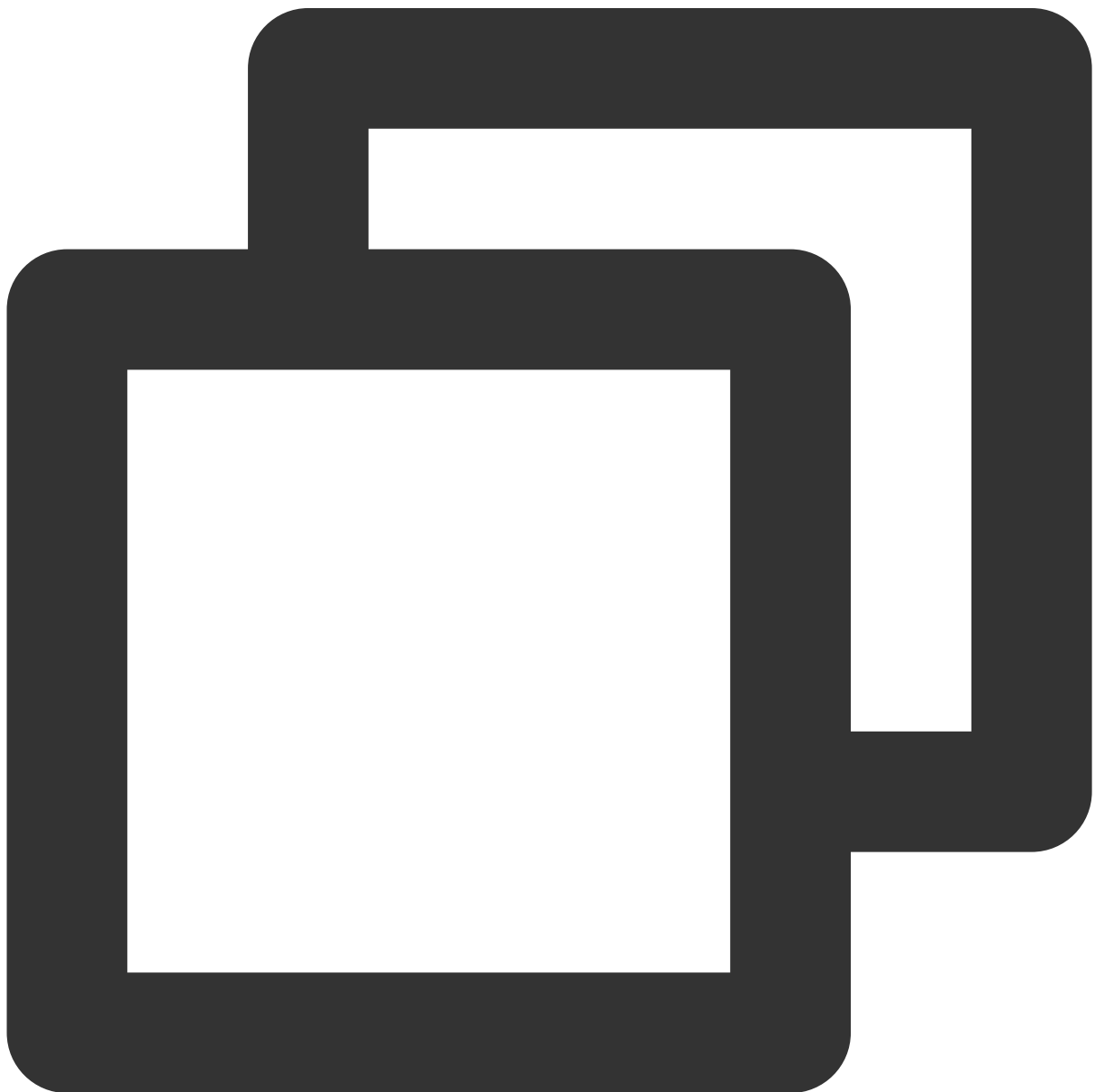
For the specific implementation process and code implementation of the above key actions, please refer to the documentation on [song synchronization](#), [lyrics synchronization](#), [vocal synchronization](#).

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)
```

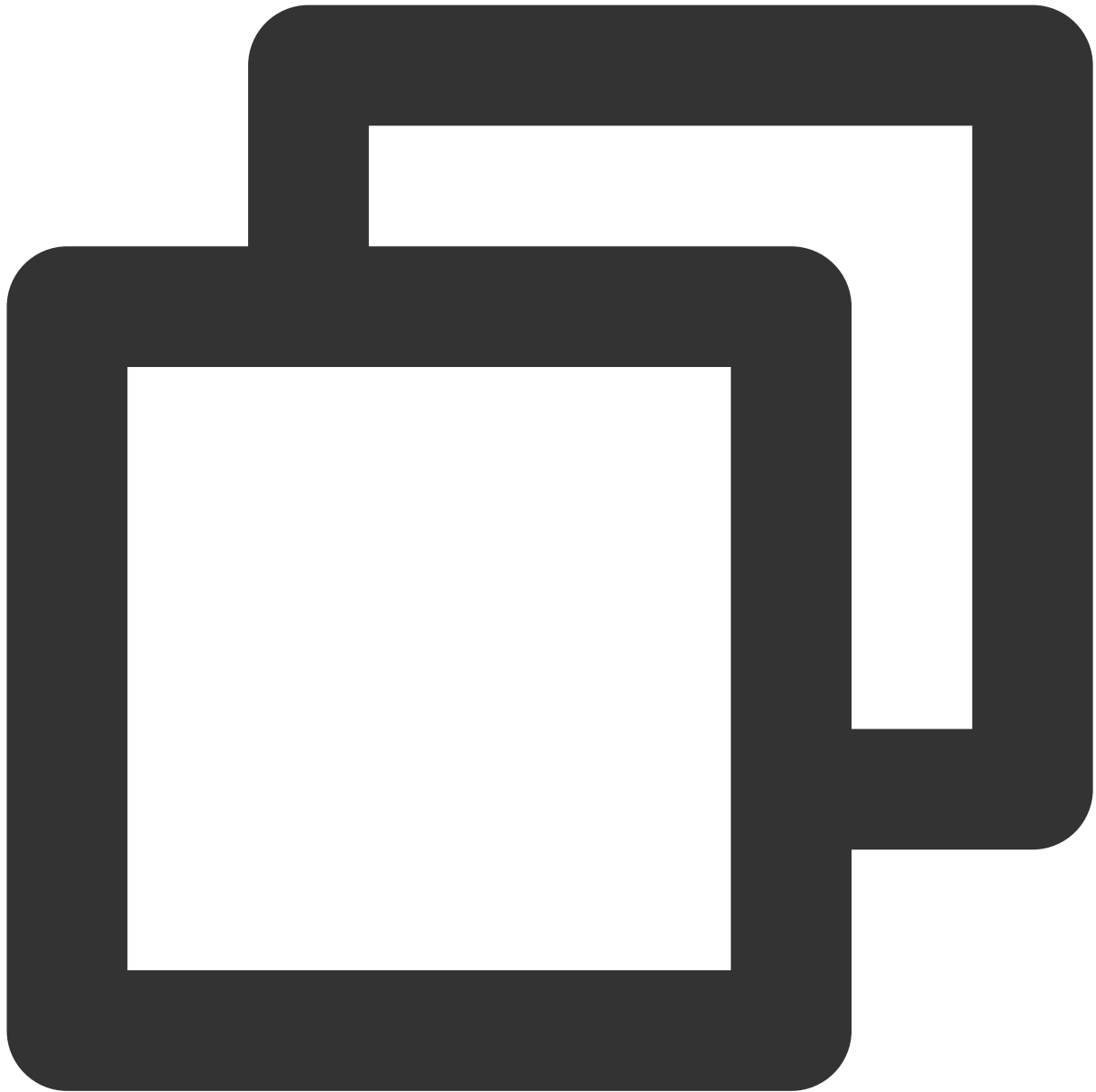
What if there is no effect after enabling the ear monitoring function?

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

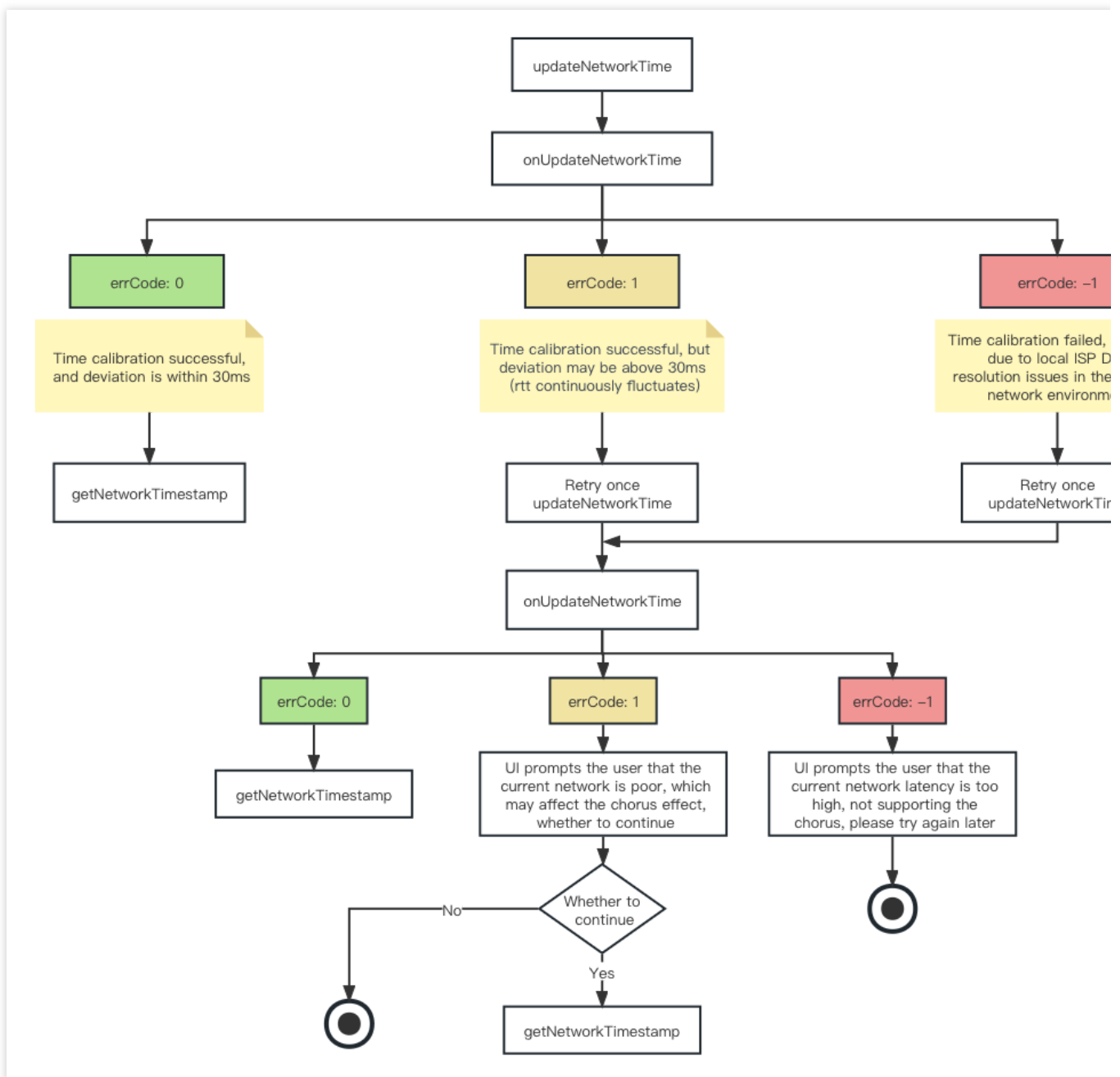
Reminder: “NTP time sync finished, but result maybe inaccurate” ?

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.

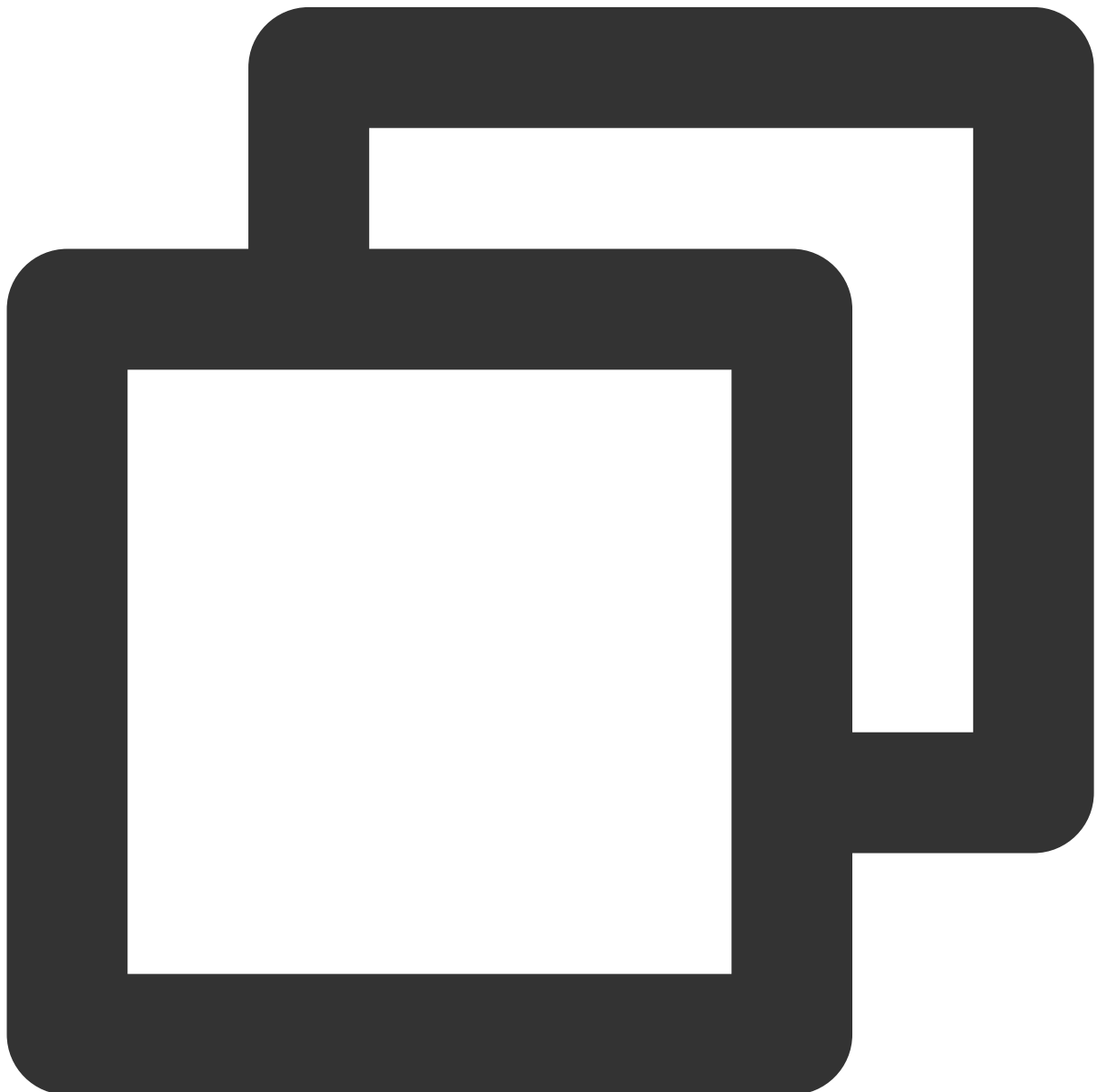
NTP service retry processing logic?



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;
```

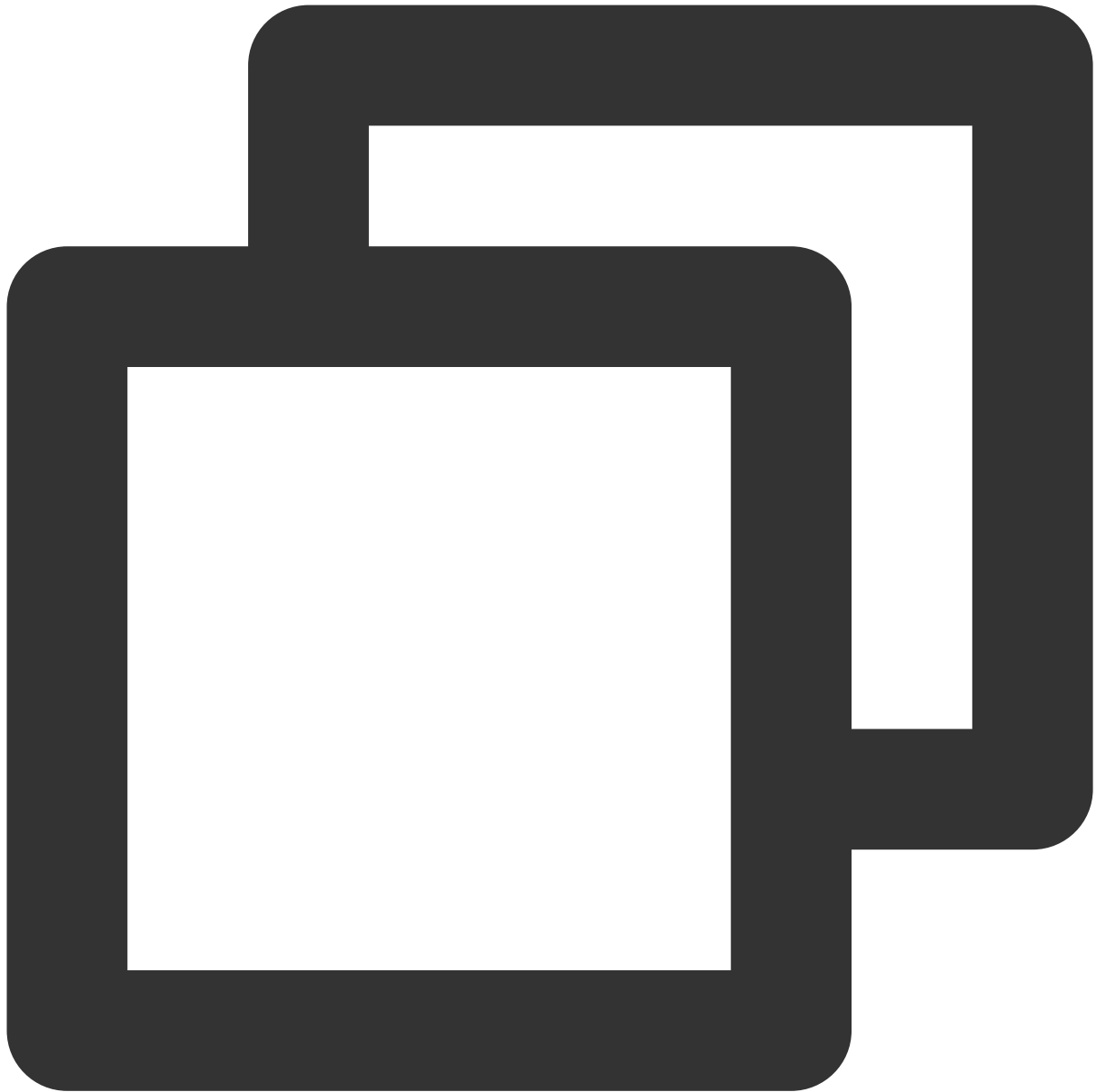
```
speedTestParams.userId = userId;
speedTestParams.userSig = userSig;
// If the actual bandwidth is higher than the expected value, the test result is th
// if the actual bandwidth is lower than the expected value, the test result is the
speedTestParams.expectedDownBandwidth = 3000; // Expected downstream 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](#).