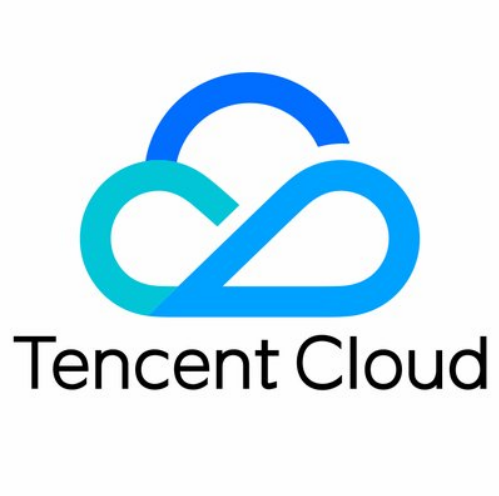
Game Multimedia Engine

Advanced Feature Development Guide

Product Documentation



Contents

Advanced Feature Development Guide
Server-Side Recording
Full Recording

Last updated：2024-01-18 14:10:27

This document describes how to quickly integrate the **GME server-side recording** feature through full recording.

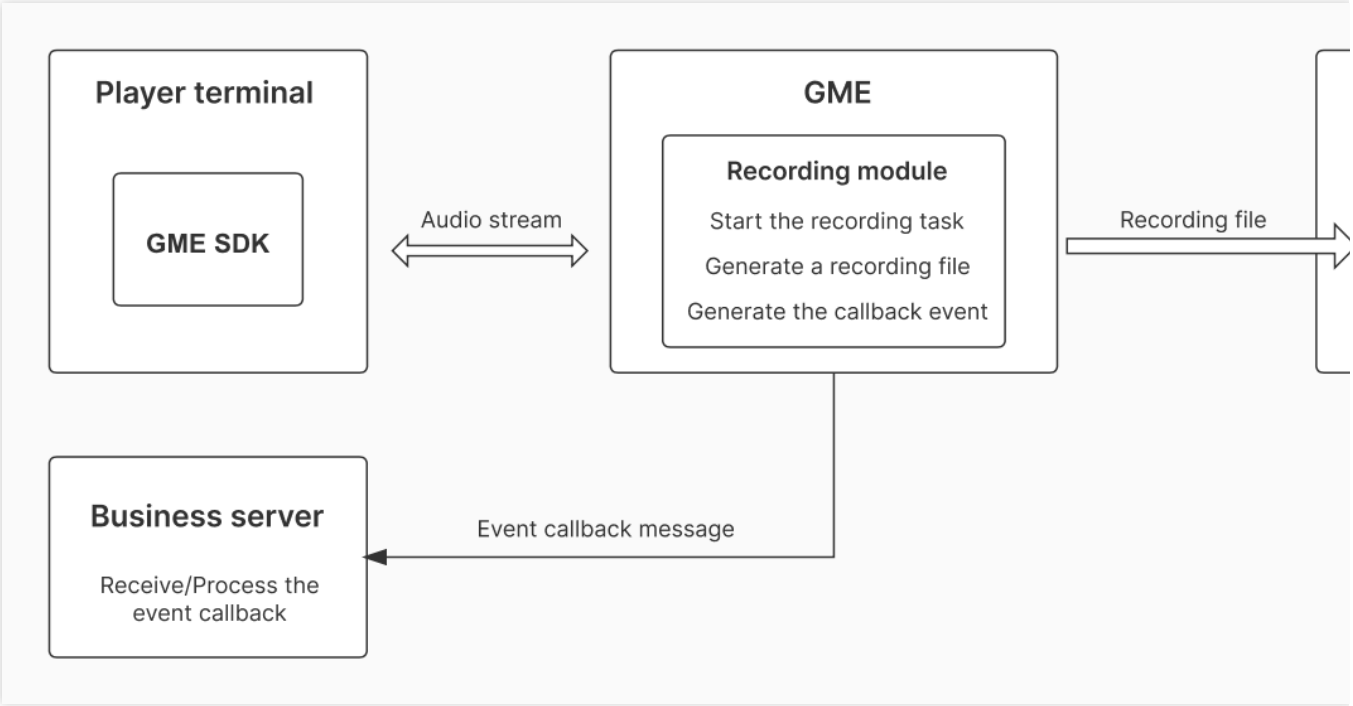
Use Cases

GME provides **server-side recording** capabilities for voice chat audio streams to help you implement various scenarios, including content retention, management, and reproduction. In full recording mode, you can record all voice chat rooms in the application. In custom recording mode, you can record the specified room. In both recording modes, you can record the mixed stream by room or single stream by user. Recording files will be stored in **COS** under your account.
This document describes how to integrate **full recording**. To enable custom recording for your application, see Custom Recording.
Note:
Using GME's server-side recording feature will incur recording service fees. Billing for this feature at Tencent Cloud International will officially start on April 1, 2023. Billing details will be announced in advance in Purchase Guide. Recording files will be stored in **COS** under your account, and **COS bills** will be generated based on your specific usage information such as storage volume, duration, and access frequency. For billing details, see Billing Overview.

Prerequisites

You have activated the voice chat service as instructed in Activating Services.
You have activated the server-side recording service. Currently, this feature is only available to allowed users. To try it out, contact us to add you to the allowlist.
You have integrated the GME SDK, including core APIs and voice chat APIs. For more information, see Quick Integration of Native SDK, Quick Integration of SDK for Unity, and Quick Integration of SDK for Unreal Engine.

Service Architecture



Feature Overview

1. Recording scope

After full recording is enabled, all voice chat rooms will be recorded. You can specify only mixed-stream recording, only single-stream recording, or both.

2. Recording mechanism

Recording task start

A recording task will start when the first user enters the room.

Recording task ending

The recording task will end when the last user exits the room.

Recording task segmentation

If a single audio file lasts two hours or longer, it will be segmented automatically.
If a user turns off the mic, the user's single-stream recording will be segmented automatically. After a user turns on the mic, a new segment will be generated.
If an ongoing recording task is interrupted due to an exception, a new segment will be generated after the task restarts automatically.

Recording task event notification

A recording task event will be notified to the configured callback URL through the callback mechanism. If such an event occurs, you will receive a callback notification such as **Recording started**, **Recording stopped**, or **The recording file has been uploaded**.
For the specific callback information, see Recording Callback Description.

3. Storage location

With GME's server-side recording, recording files will be stored in the specified bucket in **COS** under your account. You need to specify the region of the bucket. For the available regions, see Regions and Access Endpoints.

4. Recording file format

.mp3

5. Recording file naming convention

User single-stream recording files: bizid_roomid_userid/${task start time}_${id}_audio.mp3
Room mixed-stream recording file: bizid_roomid/${taskid} _${task start time} _${id}_audio.mp3
bizid: GME application ID, which can be obtained in the GME console.
roomid: Voice chat room ID, which is defined and passed in to the GME SDK when you use the voice chat service.
userid: Player ID, which is defined and passed in to the GME SDK when you use the voice chat service.
taskid: Recording task ID, which is generated by the GME recording service. Each recording task has a unique task ID.
id: Serial number of a recording task segment, which starts from `0` .

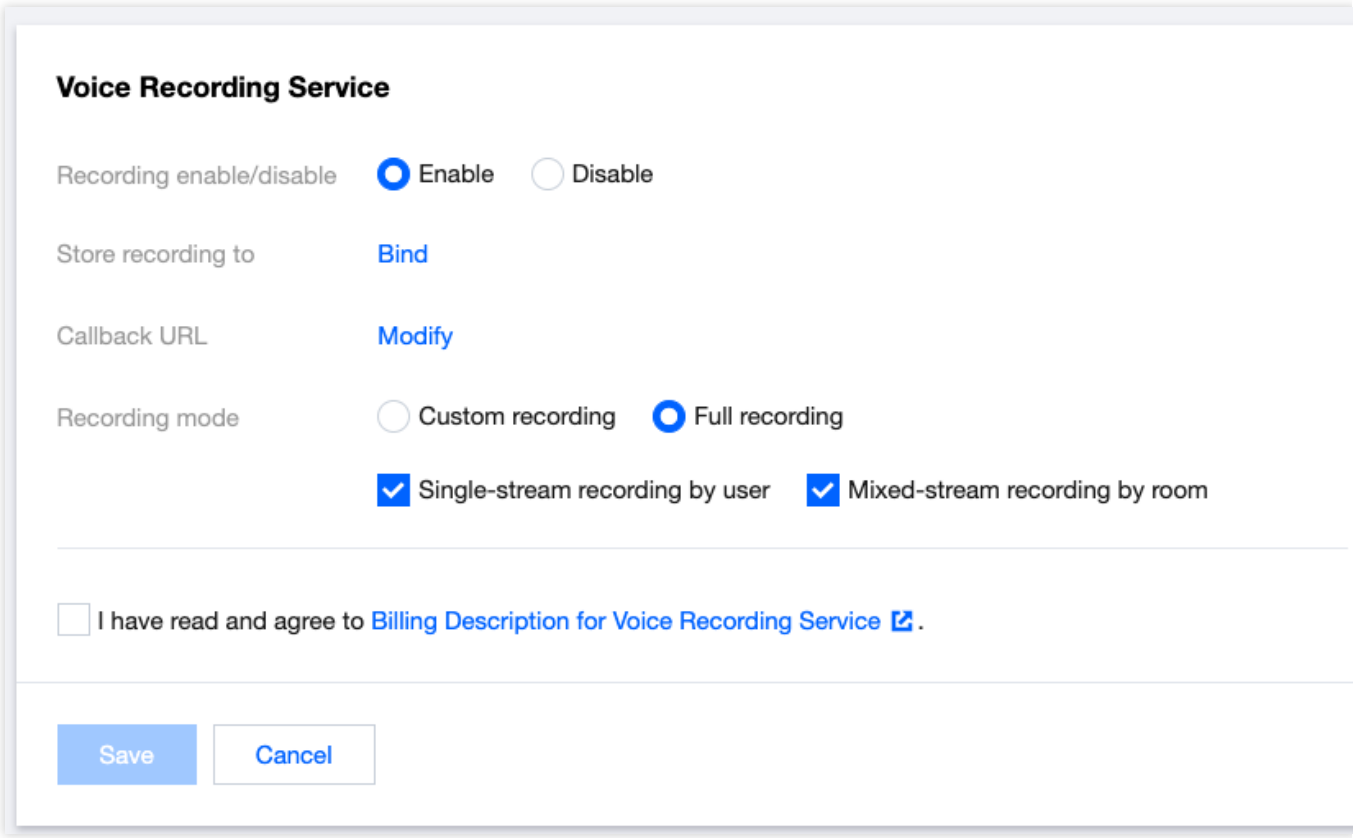
Integration directions

1. Configure the recording service in the consolebusiness side
2. Receive the recording task callback (optional)business side
3. View/Manage recording filesbusiness side

Step 1. Configure the recording service in the console

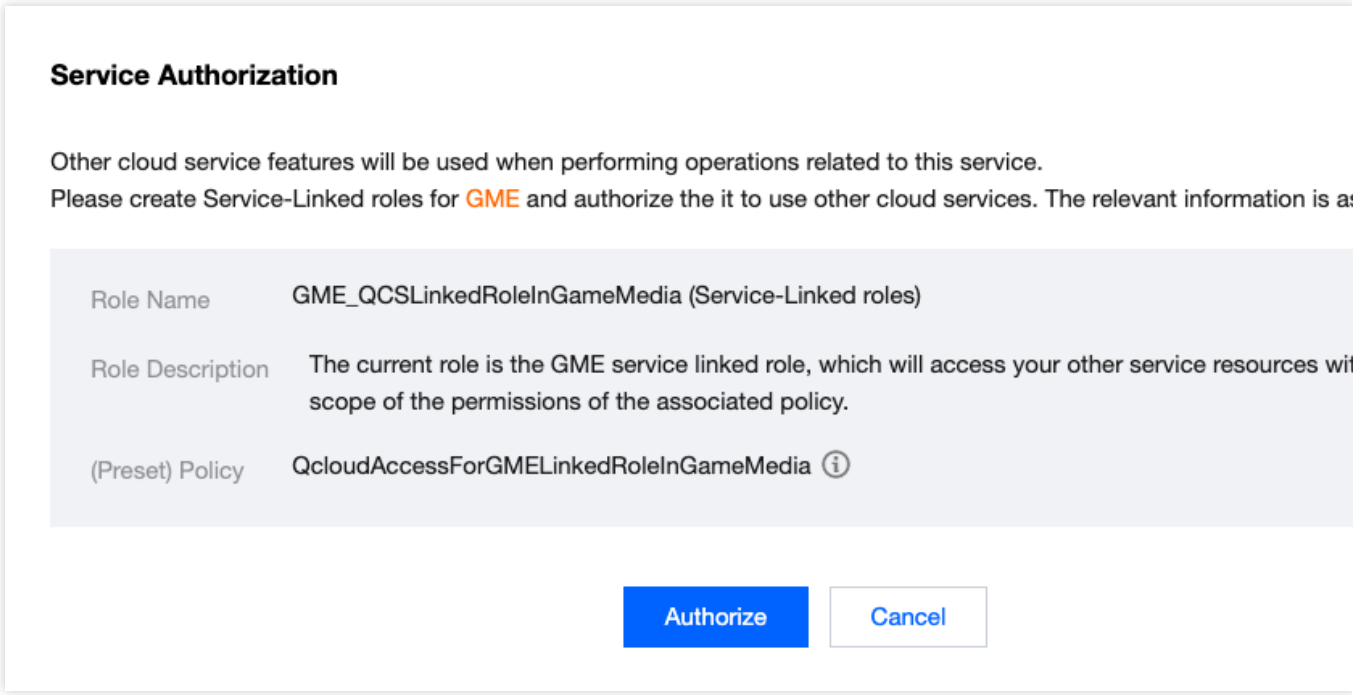
Activating/Deactivating the recording service

Log in to the GME console, click **Service Management** on the left sidebar, locate the target application, click **Set** to enter the **Details** page, and click **Modify** in the **Voice Recording Service** section.



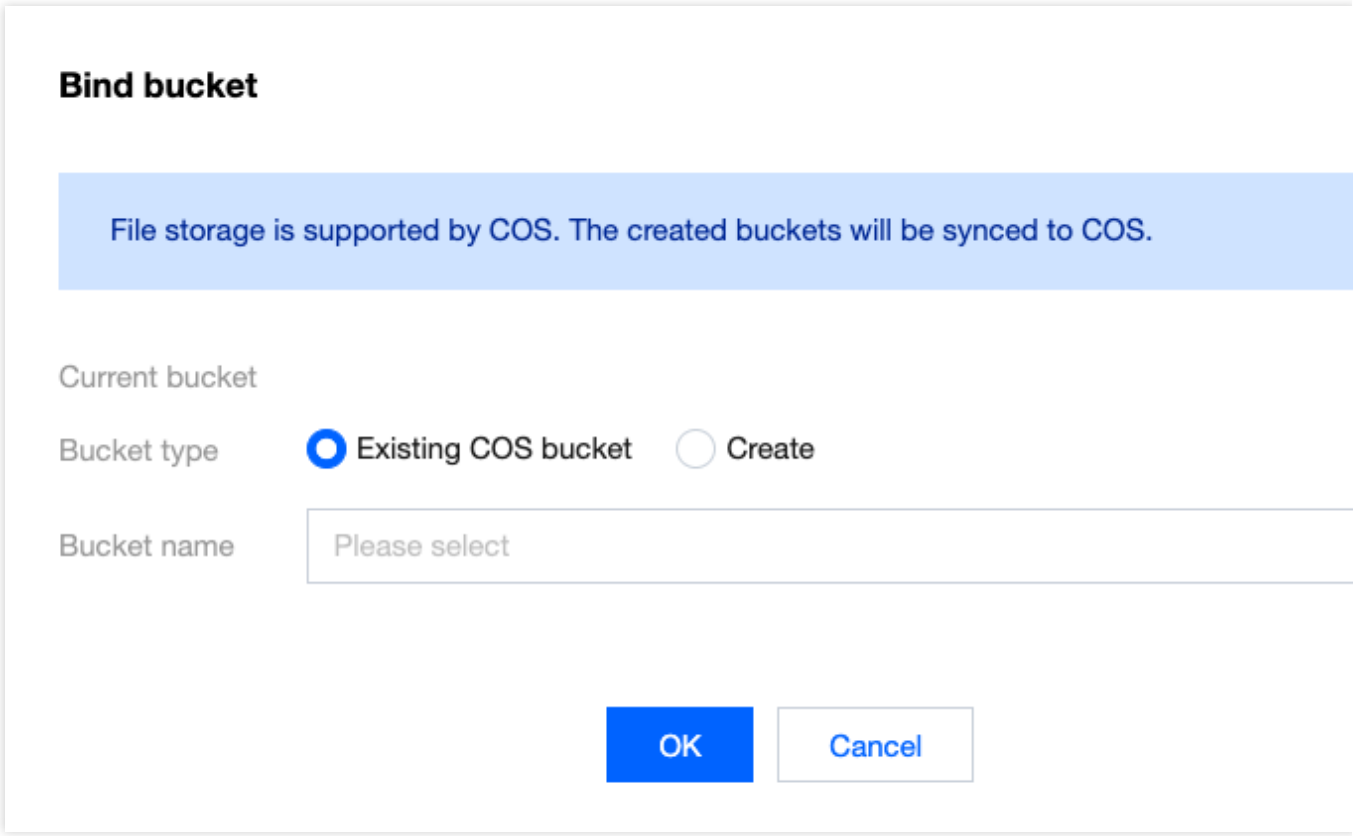
Select **Enable**.

When you activate the recording service for the first time, GME will request access to your **COS service**. You need to grant the access in the pop-up window to activate the server-side recording service.



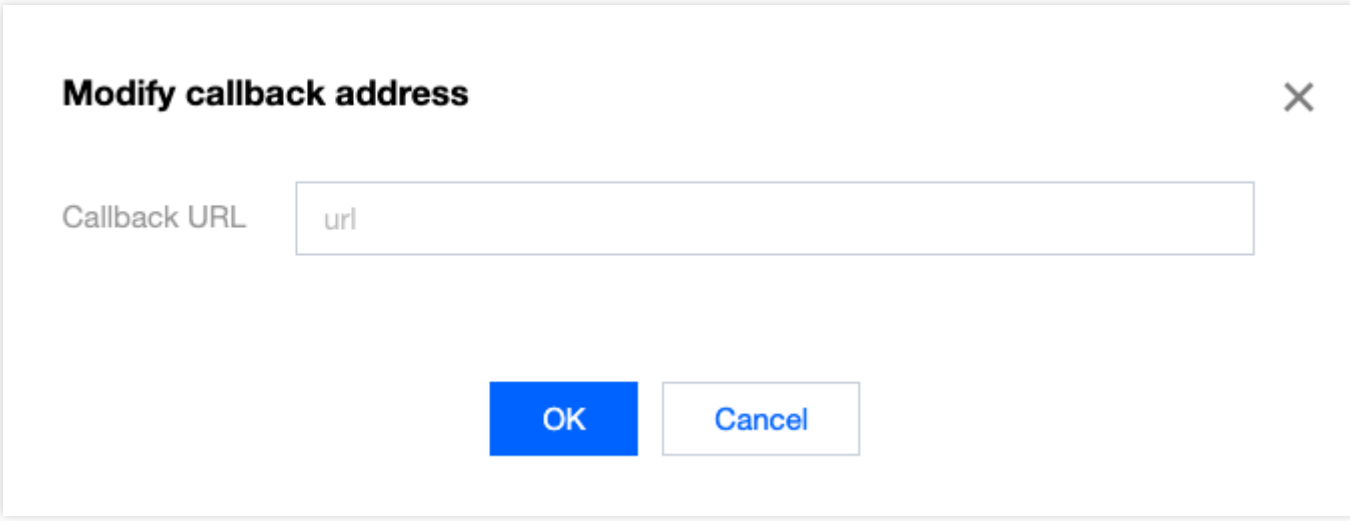
Configuring the recording file storage location

On the **Details** page, click **Modify** in the **Voice Recording Service** section and click **Bind** next to **Recording File Bucket**.
In the **Bind Bucket** pop-up window, you can bind an existing COS bucket (created in the COS console in advance) or create a bucket.



Configuring the recording event callback (optional)

If you want to receive the event callbacks of the recording service, you can configure the callback URL. On the **Details** page, click **Modify** in the **Voice Recording Service** section and click **Modify** next to **Callback URL**. In the pop-up window, enter the URL for receiving callbacks. Currently, event callback messages are pushed only for the recording task completion status.



Configuring the recording scope

You can select **Custom recording** or **Full recording** for **Recording Scope**. Here, you need to select **Full recording** and specify whether to record single streams, mixed streams, or both.
After setting the above configuration items, click **Save**, and the recording service will be activated. If you no longer need the service, select **Disable** for the recording service in the console to avoid incurring additional fees.

Step 2. Receive the recording task callback (optional)

If a callback URL is configured in **step 1**, you can receive recording task event callbacks.

Step 3. View/Manage recording files

An audio file can be generated in several minutes after a recording task ends. You need to log in to the COS console to view and manage recording files.

Custom Recording

Last updated：2024-01-18 14:12:40

This document describes how to integrate the **GME server-side recording** feature through custom recording.

Use Cases

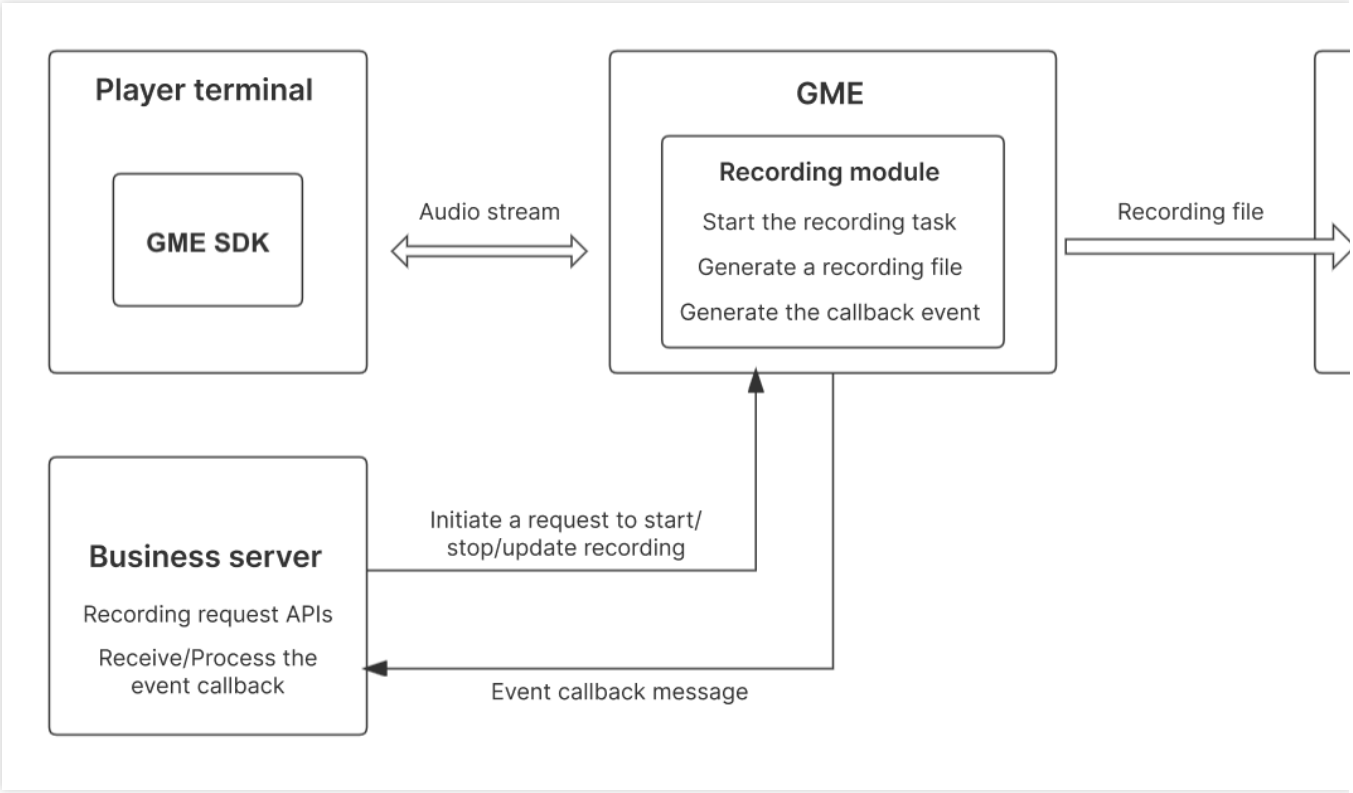
GME provides **server-side recording** capabilities for voice chat audio streams to help you implement various scenarios, including content retention, management, and reproduction. In full recording mode, you can record all voice chat rooms in the application. In custom recording mode, you can record the specified room. In both recording modes, you can record the mixed stream by room or single stream by user. Recording files will be stored in **COS** under your account.
This document describes how to develop and integrate **custom recording**. To enable full recording for your application, see Full Recording.

Note:
Using GME's server-side recording feature will incur recording service fees. Billing for this feature at Tencent Cloud International will officially start on April 1, 2023. For billing details, see Purchase Guide.
Recording files will be stored in **COS** under your account, and **COS bills** will be generated based on your specific usage information such as storage volume, duration, and access frequency. For billing details, see Billing Overview.

Prerequisites

You have activated the voice chat service as instructed in Activating Services.
You have activated the server-side recording service. Currently, this feature is only available to allowed users. To try it out, contact us to add you to the allowlist.
You have integrated the GME SDK, including core APIs and voice chat APIs. For more information, see Quick Integration of Native SDK, Quick Integration of SDK for Unity, and Quick Integration of SDK for Unreal Engine.

Service Architecture



Feature Overview

1. Recording scope

You can specify RoomID to record the mixed stream or a user in a room to record a single stream in the server API. For the ID of the room to be recorded, you can specify players to be recorded in a allowlist and players not to be recorded in a blocklist through server API parameters.

2. Relevant APIs

***StartRecord()*Starts recording**

This API is used to specify single-stream recording or mixed-stream recording, `RoomID` of the room to be recorded, and the subscribed allowlist/blocklist of user IDs.

***StopRecord()*Ends recording**

This API is used to end recording for a `taskid` . If you don't record the `taskid` , you can use `DescribeTaskInfo()` to get the `taskid` of the ongoing recording task in the specified room.

***ModifyRecordInfo()*Updates the recording information**

This API is used to update the recording information of a `taskid` such as recording type and subscribed allowlist/blocklist of user IDs. If you don't record the `taskid` , you can use `DescribeTaskInfo()` to get the `taskid` of the ongoing recording task in the specified room.

DescribeTaskInfo() **Queries the room recording information**

This API is used to query the recording information of the specified room such as `taskid` of the ongoing recording task and subscribed allowlist/blocklist of user IDs.

DescribeRecordInfo() **Queries the recording task information**

This API is used to query the task information of a `taskid` such as recording type, recorded room ID, recorded user ID, and recording file information.

3. Recording mechanism

Recording task start

After the ***StartRecord()*** API is called, the specified room recording task will start.

Recording task ending

After the ***StopRecord()*** API is called, the specified room recording task will stop.

Recording file generation time

Room mixed-stream recording file: After a room recording task starts, if a user is in the room, the mixed-stream audio file will start to be generated immediately; otherwise, file generation will start immediately after the first user enters the room.

User single-stream recording files: After a room recording task starts, if a user within the recording scope enters the room, the single-stream audio file of the user will start to be generated immediately.

Recording task segmentation

If a single audio file lasts two hours or longer, it will be segmented automatically.

If a user turns off the mic, the user's single-stream recording will be segmented automatically. After a user turns on the mic, a new segment will be generated.

If an ongoing recording task is interrupted due to an exception, a new segment will be generated after the task restarts automatically.

Recording task event notification

A recording task event will be notified to the configured callback URL through the callback mechanism. If such an event occurs, you will receive a callback notification such as **Recording started**, **Recording stopped**, or **The recording file has been uploaded**.

For the specific callback information, see Recording Callback Description.

4. Storage location

With GME's server-side recording, recording files will be stored in the specified bucket in **COS** under your account. You need to specify the bucket region. For the specific available regions, see Regions and Access Endpoints.

5. Recording file format

.mp3

6. Recording file naming convention

User single-stream recording files: bizid_roomid_userid/${task start time}_${id}_audio.mp3

Room mixed-stream recording file: bizid_roomid/${taskid}*${task start time}*${id}_audio.mp3

bizid: GME application ID, which can be obtained in the GME console.

roomid: Voice chat room ID, which is defined and passed in to the GME SDK when you use the voice chat service.

userid: Player ID, which is defined and passed in to the GME SDK when you use the voice chat service.

taskid: Recording task ID, which is generated by the GME recording service. Each recording task has a unique task ID.

id: Serial number of a recording task segment, which starts from `0` .

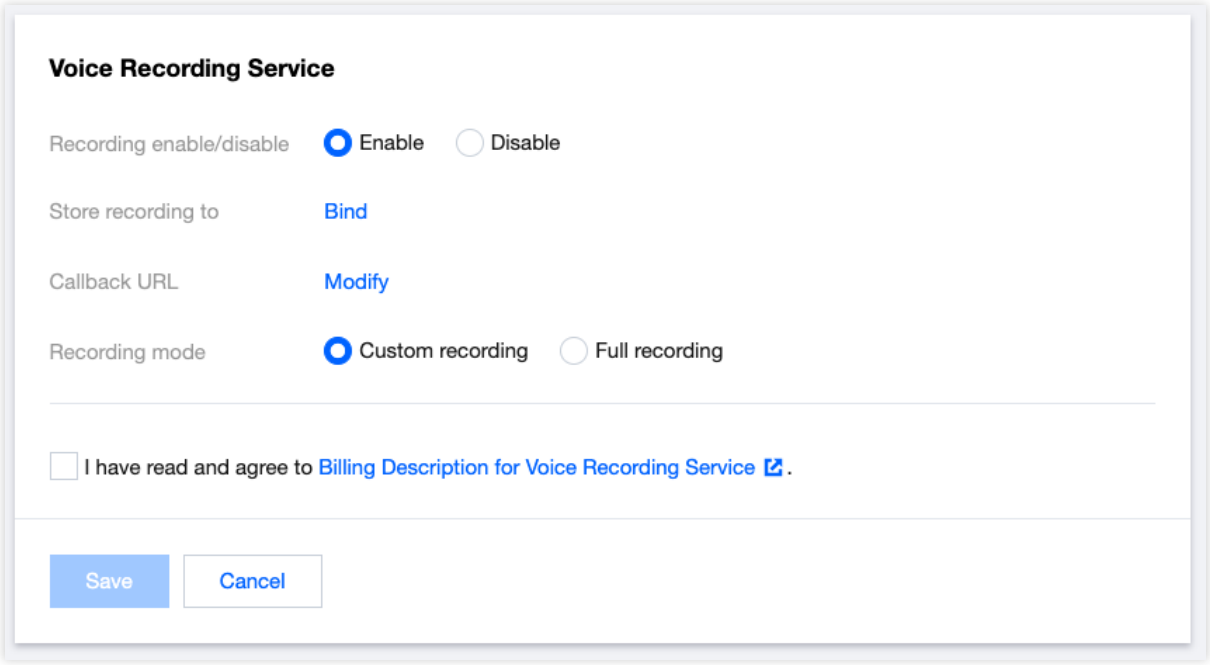
Integration directions

1. Configure the recording service in the consolebusiness side
2. Call the server API to define the recording scopebusiness side
3. Receive the recording task callback (optional)business side
4. View/Manage recording filesbusiness side

Step 1. Configure the recording service in the console

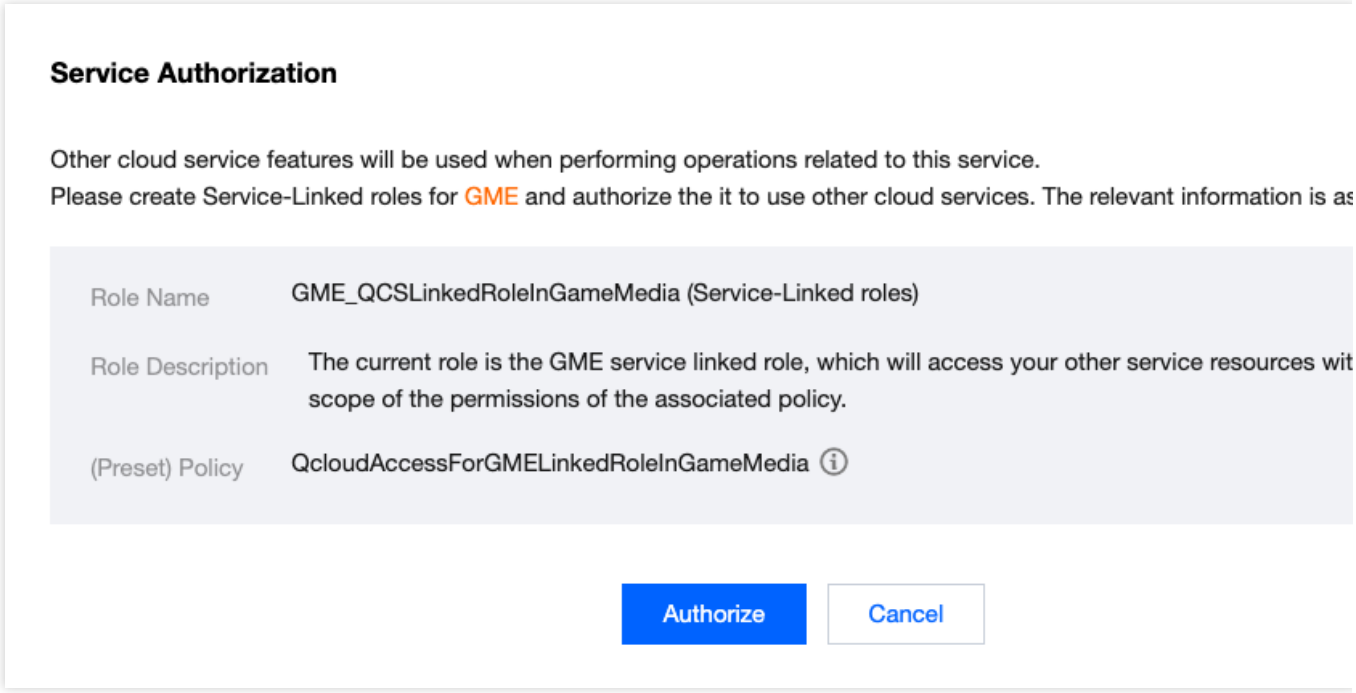
Log in to the GME console, click **Service Management** on the left sidebar, locate the target application, and click **Set** to enter the **Details** page.

Activating/Deactivating the recording service



In the **Voice Recording Service** section, click **Modify** and select **Enable**.

When you activate the recording service for the first time, GME will request access to your **COS service**. You need to grant the access in the pop-up window to activate the server-side recording service.

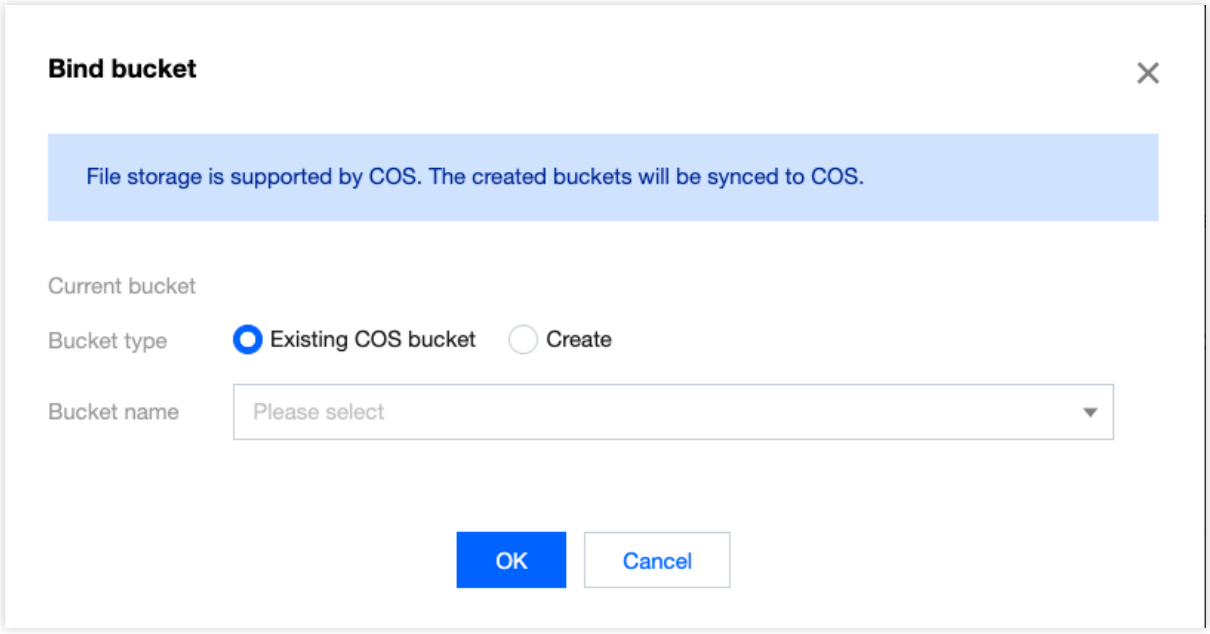


Configuring the recording file storage location

Click **Bind** next to **Recording File Bucket**.

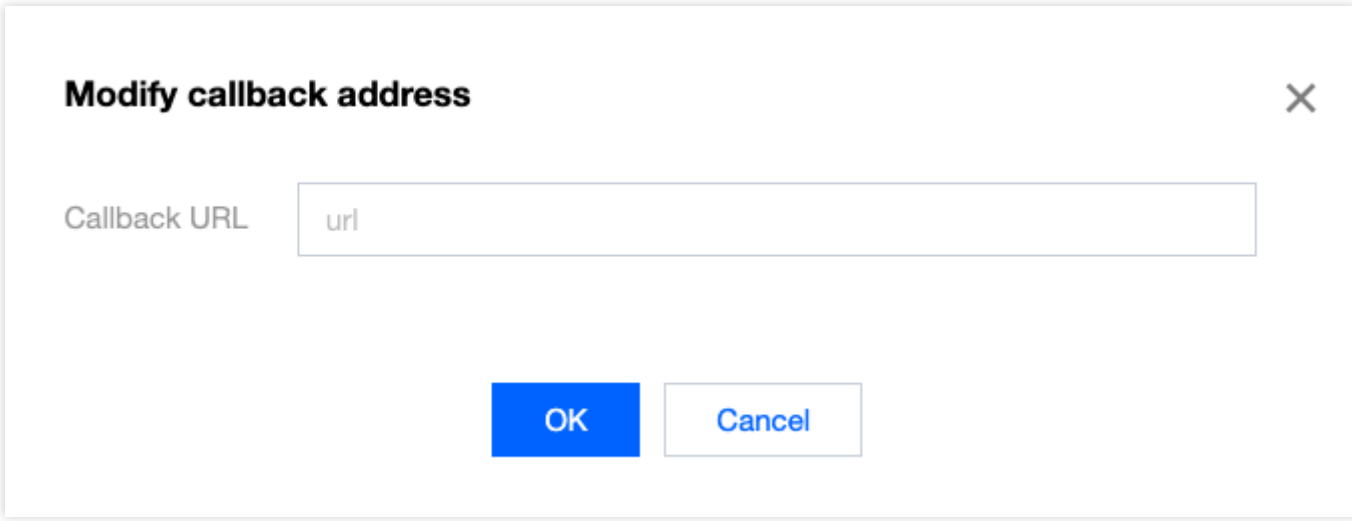
In the **Bind Bucket** pop-up window, you can bind an existing COS bucket (created in the COS console in advance) or

create a bucket.



Configuring the recording event callback (optional)

If you want to receive the event callbacks of the recording service, you can configure the callback URL. On the **Details** page, click **Modify** in the **Voice Recording Service** section and click **Modify** next to **Callback URL**. In the pop-up window, enter the URL for receiving callbacks. Currently, event callback messages are pushed only for the recording task completion status.



Configuring the recording scope

You can select **Custom recording** or **Full recording** for **Recording Scope**. Here, you need to select **Custom recording**; otherwise, the server APIs will fail to be called.
After setting the above configuration items, click **Save**, and the recording service will be activated. If you no longer need the service, select **Disable** for the recording service in the console to avoid incurring additional fees.

Step 2. Receive the recording task callback (optional)

If a callback URL is configured in **step 1**, you can receive recording task event callbacks.

Step 3. Call the server API to define the recording scope

You need to call relevant APIs based on your actual business needs. When designing the call time sequence and process, note that:
(1) You can initiate a request to start recording for an existing `RoomId` only. If the specified `RoomId` doesn't exist, the recording task will fail to be created.
(2) If you don't actively call the API for stopping recording, the recording process will continue as long as there is a user in the room. If all users in the room have left, the original recording task process will continue for 12 hours, during which recording will start automatically when a user enters the room. After 12 hours, recording won't start automatically when a user enters the room.

Step 4. View/Manage recording files

An audio file can be generated in several minutes after a recording task stops. You need to log in to the COS console to view and manage recording files.

Recording Callback Description

Last updated：2024-01-18 14:13:49

Description of the Server-Side Recording Callback

Notes

Subject to the network conditions, the sequence of the notifications received by your server may be different from the sequence of the events that actually occurred.The service offers a retry mechanism, but it is not guaranteed that all messages can arrive.In view of the above, it is not recommended that your core business logic depend on the message notification service.

Network protocol

If the callback URL, i.e., the URL of an HTTP(S) API, is configured in the console, then the POST method should be supported and transferred data should be encoded with UTF-8.

HTTP header parameters

Parameter	Type	Required	Description
Signature	String	Yes	Signature. For more information, see Signature generation.

Signature generation

Signature = HMAC-SH1 (strContent, SecretKey)
strContent: Original signature string, which is the entire JSON content of `body` (the length is subject to `Content-Length`).
body: JSON content called back to the business. The entire content in Sample callback is `body` .
SecretKey: Application permission key, which can be viewed in **Details** in the console.
HMAC-SH1: Signature algorithm.

Callback parameters

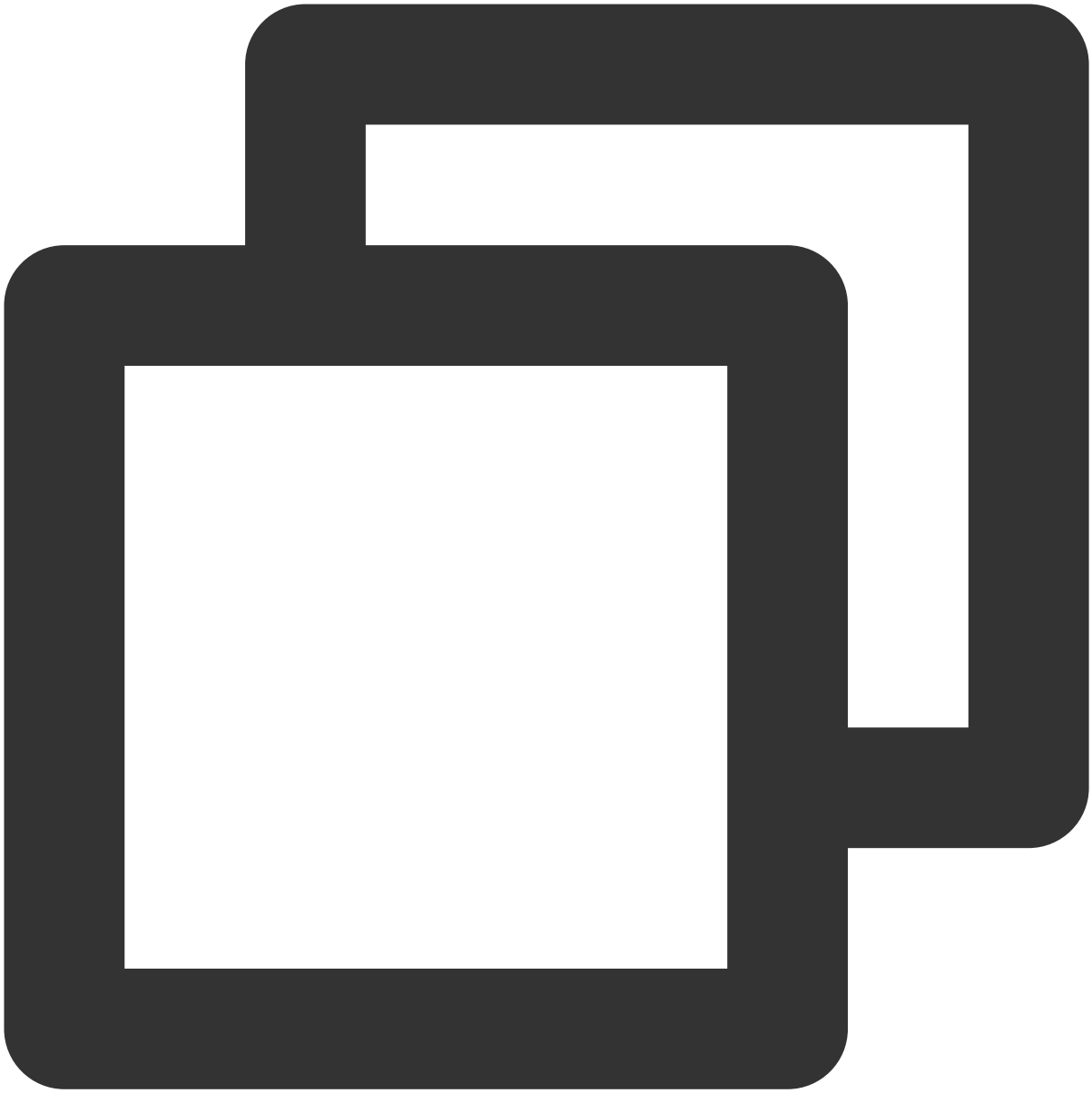
Name	Type	Description
BizID	Integer	Application's `AppID`, which can be viewed in Details in the console.
RoomID	String	Room ID
UserID	String	User ID

RecordMode	Integer	Recording mode. Valid values: 0: Single stream 1: Mixed stream
Timestamp	Integer	Timestamp in seconds when the callback is sent
TaskID	Integer	The task ID assigned by the cloud recording service, which uniquely identifies a recording process and becomes invalid after a recording task ends. When you use the custom recording mode, the task ID can be obtained through the response parameter when recording starts. It needs to be saved by the business as a request parameter for subsequent operations on the recording task.
EventType	Integer	Event type
Detail	EventDetail	Event details, whose format is specified by EventType.

`EventDetail` event details

EventType	Description	Detail
1	Recording started.	`SeqNo` : Segment number of the `Number` type `FileName` : Filename of the `String` type
2	Recording is completed.	`SeqNo` : Segment number of the `Number` type `FileName` : Filename of the `String` type
3	The audio file has been uploaded.	`SeqNo` : Segment number of the `Number` type `FileName` : Filename of the `String` type

Sample callback




```
{
    "BizID":1400000000,
    "RoomID":"100",
    "UserID":"999",
    "TaskID":446946705284000000,
    "RecordMode":1,
    "Timestamp":1675930605,
    "EventType":1,
    "Detail":{
        "SeqNo":0,
        "FileName":"1400000000_100_999/2023-02-09-16-16-45_446946705284000000_audio
```

```
        }
    }
```

Range Voice

Last updated：2023-04-27 17:06:32

Overview

In a voice chat room, a user can talk with other users within a certain distance. This feature calls the GME client API at the business layer to update the sound source position to inform the server of the local player's position. The server forwards the audio streams within the players' ranges after checking the **local coordinates and audio reception range** against **remote coordinates and audio reception range**. By default, 20 audio streams nearest a player are forwarded, and **up to 10,000 players can mic on simultaneously in a room**.

Use Cases

Battle royale games	GME provides the "Team only" and "Everyone" voice modes unique to **battle royale games and mobile survival shooter games**. The voice effect varies by selected mode as follows: 1. If the "Team only" mode is selected, the player can hear only the voice of teammates. 2. If the "Everyone" mode is selected, the player can hear the voice of teammates as well as the voice of enemies within a certain range.
Immersive virtual scenarios	In a virtual scenario like **in-game concert**, when the singer sings in the virtual room, the range voice feature can be used to enable the audience to talk with other players within a certain range while hearing the singer. Up to 10,000 players can turn on their mics simultaneously in a room.

Demo Effects

You can download the free demo to try out the 3D sound effect and range voice features.

Concepts

The range voice feature involves three concepts: **voice mode**, **range for hearing the voice**, and **TeamID**.

Voice mode

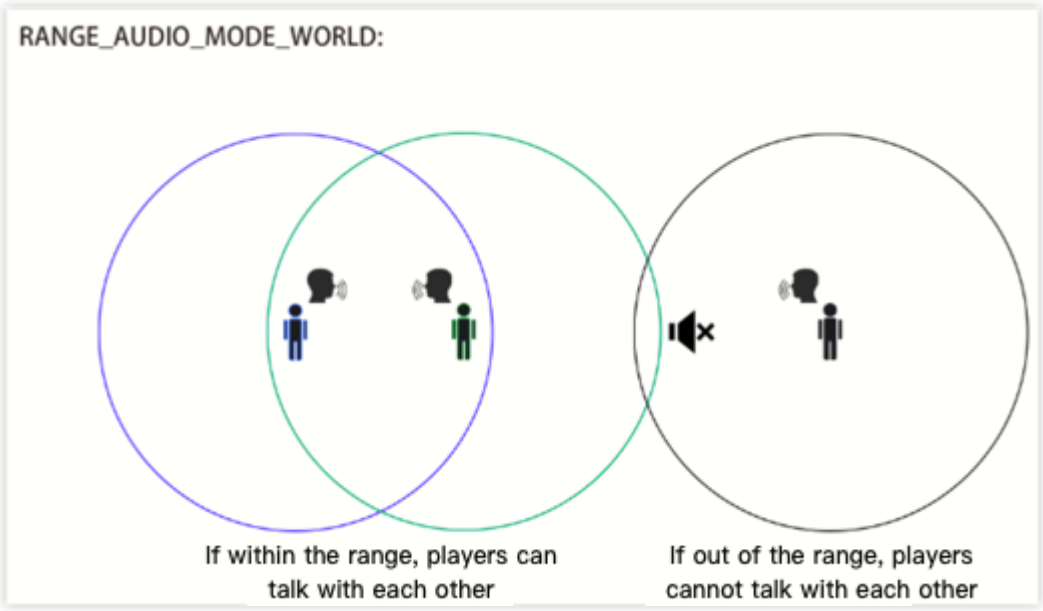
When the user enters a range voice room, the user can choose from two voice modes:

Voice Mode	Parameter	Feature
Everyone	RANGE_AUDIO_MODE_WORLD	In this mode, the player can be heard by other players within a certain range and can talk with them if they are also in this mode. Teammates can hear each other.
Team only	RANGE_AUDIO_MODE_TEAM	Only teammates can hear each other.

Note

Teammates can talk with each other regardless of the distance and voice mode.

Range for hearing the voice



If the voice mode is set to **Everyone** (`RANGE_AUDIO_MODE_WORLD`), the range for hearing the voice will be subject to the `UpdateAudioRecvRange` API.

Suppose players A and B in different teams both set their voice modes to **Everyone** (`RANGE_AUDIO_MODE_WORLD`):

Player Coordinates	Range for Hearing the Voice	Voice Reachability of the Other Player	Voice Reachability of Teammates
A (0,0,0)	10 meters	The player can hear player B, as player B is within 10 meters of the player.	Teammates can talk with each other.

B (0,8,0)	5 meters	The player cannot hear player A, as player A is more than 5 meters away.	Teammates can talk with each other.

Note

For more information on the specific voice reachability of players, see Range Voice.

TeamID

To use the range voice feature, you need to call the `SetRangeAudioTeamID` API to set the team ID (`TeamID`) and then call the `EnterRoom` API to enter the room.

If `TeamID` specified during room entry is not `0` , a user will enter the range voice chat room mode. If a user sets `TeamID` to `1` and the voice mode to `RANGE_AUDIO_MODE_TEAM` when entering a voice chat room, only members whose `TeamID` is `1` can hear the user. If the user sets the voice mode to `RANGE_AUDIO_MODE_WORLD` , players within a certain range as well as members whose `TeamID` is `1` can hear the user.

`TeamID` Value	Voice Mode	Range	Voice Reachability
`TeamID` is not `0` . Suppose `TeamID` is `1` .	RANGE_AUDIO_MODE_TEAM	10 meters	The user can talk with only members whose `TeamID` is `1` .
	RANGE_AUDIO_MODE_WORLD	10 meters	The user can talk with members whose `TeamID` is `1` and members in `RANGE_AUDIO_MODE_WORLD` voice mode within 10 meters in the same room.

If a user enters a voice chat room with `TeamID` set to `0` , the user will enter the range voice host mode, where all members in the room (regardless of their voice mode) can hear the user.

`TeamID` Value	`TeamID` Modification Time	Range	Voice Reachability
TeamID = 0	`TeamID` is not `0` before room entry and is changed to `0` after room entry.	10 meters	All members in the room (regardless of their voice mode) can hear the user. The user can talk with members whose `TeamID` is `0` . The user can hear members in `RANGE_AUDIO_MODE_WORLD` voice mode within 10 meters in the same room.
	`TeamID` is `0` before and during room entry.	10 meters	All members in the room (regardless of their voice mode) can hear the user.

			The user can talk with members whose `TeamID` is `0`. The user cannot hear other members in the room.

Scenarios

Battle royale game: In a battle royale game, every four players are grouped into a team, and a team ID (`TeamID`) needs to be set for them. A battle room can contain 100 players, i.e., 25 teams, so all those teams enter the same voice chat room. In a battle, if a player wants to communicate with a stranger within 10 meters, the player can set the range for hearing the voice to 10 meters and the voice mode to `RANGE_AUDIO_MODE_WORLD` and enable both the mic and speaker. If the player wants to communicate with only teammates, the player can set the voice mode to `RANGE_AUDIO_MODE_TEAM` .

Game concert: If a singer wants to hold a concert in a game but doesn't need to interact with players, players can enter the range voice chat room with their `TeamID` set to `OpenID` , voice mode set to `RANGE_AUDIO_MODE_WORLD` , and range for hearing the voice set to an appropriate value so as to talk with players nearby. The singer can set `TeamID` to `0` before entering the room to be heard by all members in the room without hearing other members.

Host mode: In a game such as virtual board game, the host needs to be heard by all members in the room and hear players within a certain range. In this case, the host can enter the room with `TeamID` set to a non-0 value and change `TeamID` to `0` after room entry, so as to be heard by all members in the room while hearing players within a certain range.

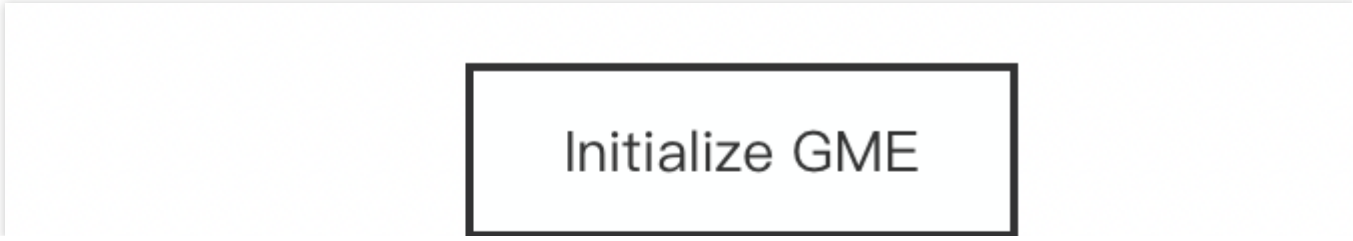
Prerequisites

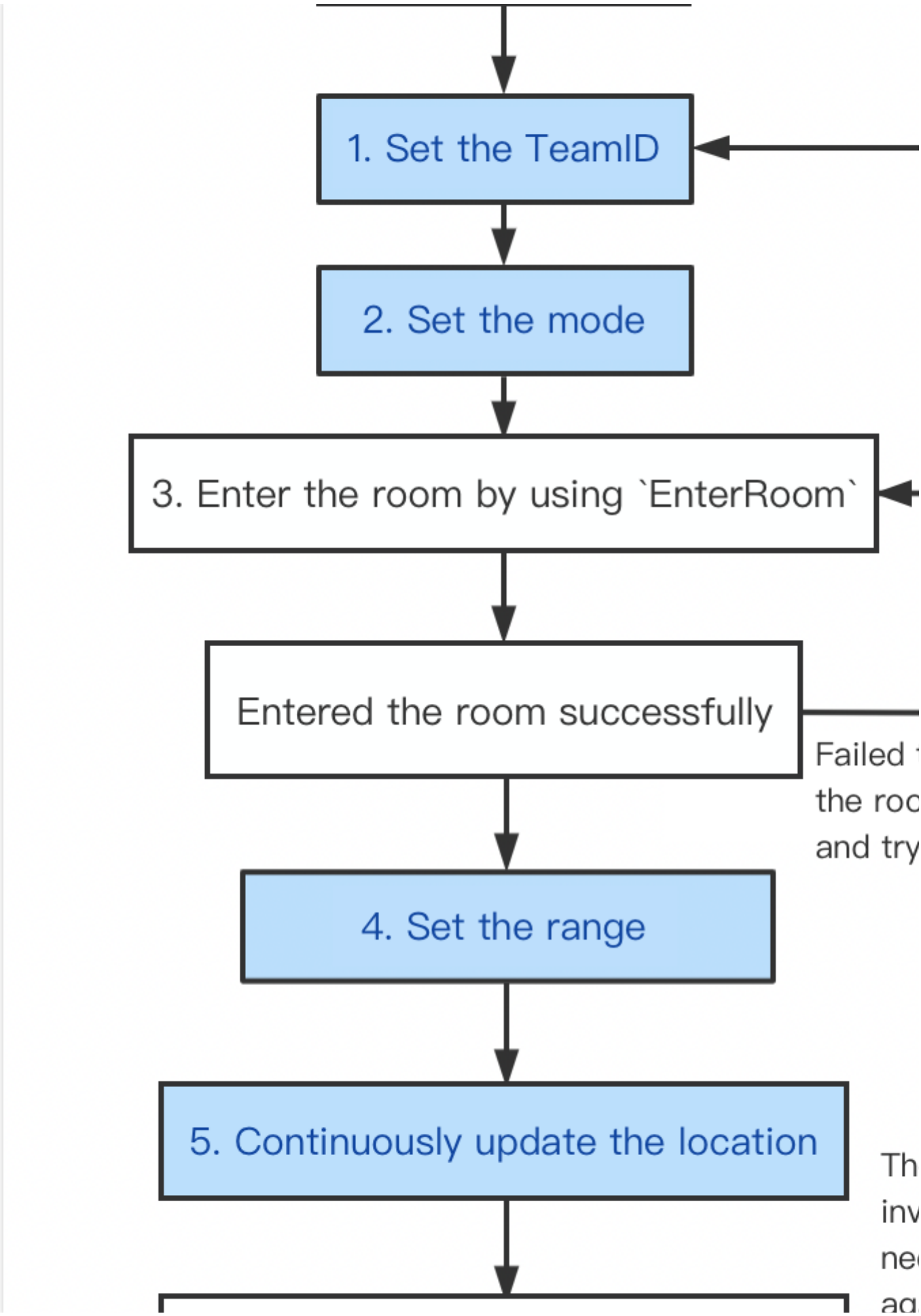
You have activated the voice chat service. For more information, see Activating Services.

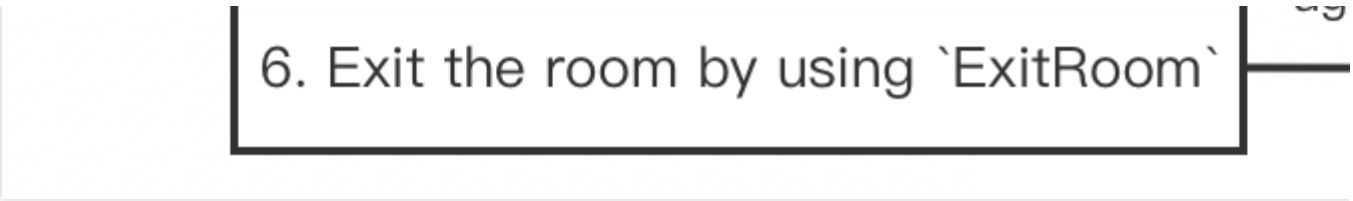
You have integrated the GME SDK, including core APIs and voice chat APIs. For more information, see Quick Integration of Native SDK, Quick Integration of SDK for Unity, and Quick Integration of SDK for Unreal Engine.

You have applied for a voice chat room for over 10,000 users: If more than 1,000 users in a room use range voice, submit a ticket for application.

Directions







Notes

Be sure to follow the flowchart to call the API.

The blue part of the flowchart shows the process of the range voice.

A range voice chat room is different from a team voice chat room in that **you must select the smooth sound quality when entering the range voice chat room.**

Once you enter the room, call `UpdateAudioRecvRange` (once at least) and call `UpdateSelfPosition` once per frame.

1. Set `TeamID`

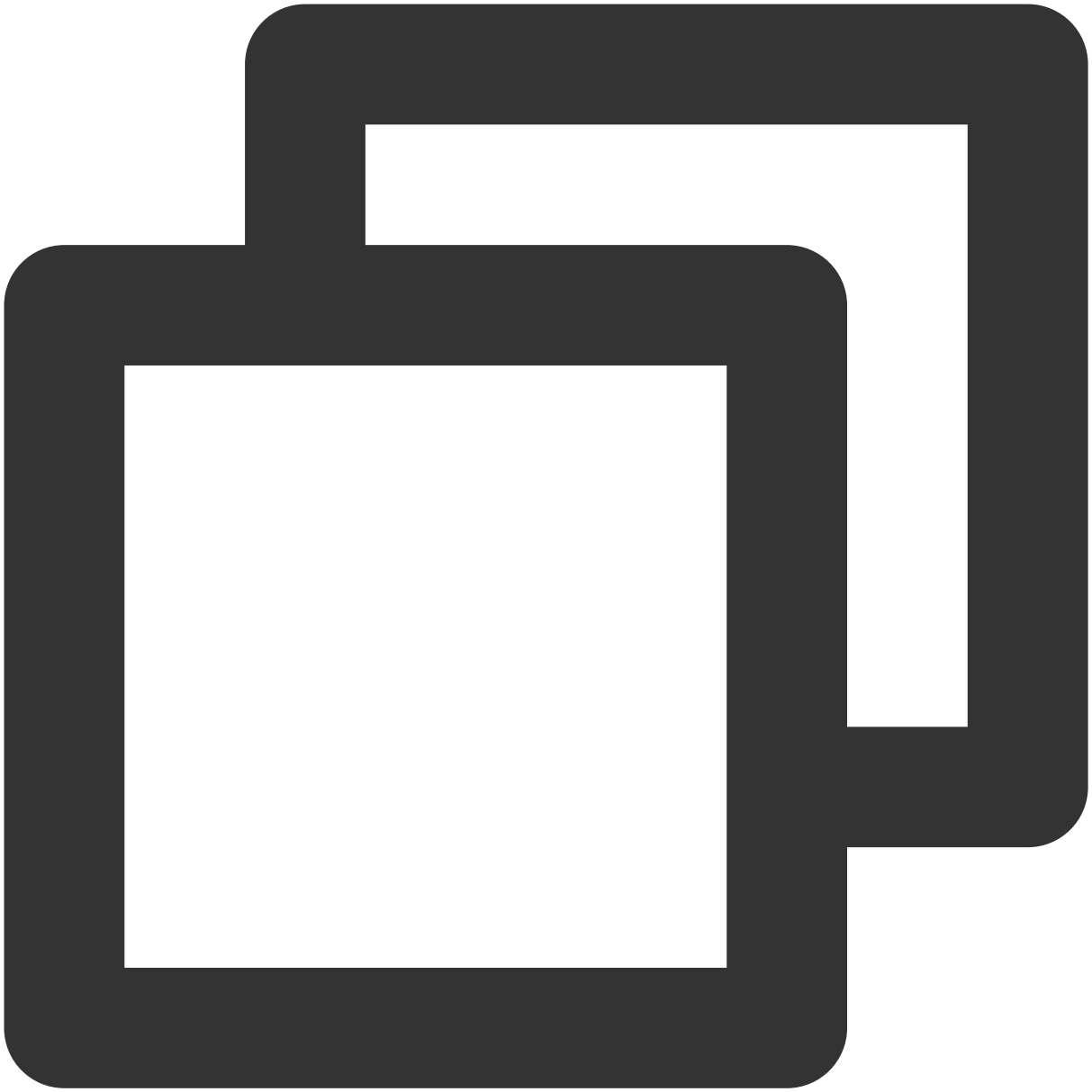
Calling this API to set the team ID before room entry will affect the next room entry.

This API can be called to modify the team ID after room entry. The modification will take effect immediately.

`TeamID` will not be automatically reset to `0` upon room exit. Therefore, once you decide to call this voice mode, do so to set the `TeamID` before each call of `EnterRoom` .

When entering a room after exiting the room, call the API for setting the team ID again after the callback for successful room exit is returned.

Function prototype



```
ITMGContext SetRangeAudioTeamID(int teamID)
```

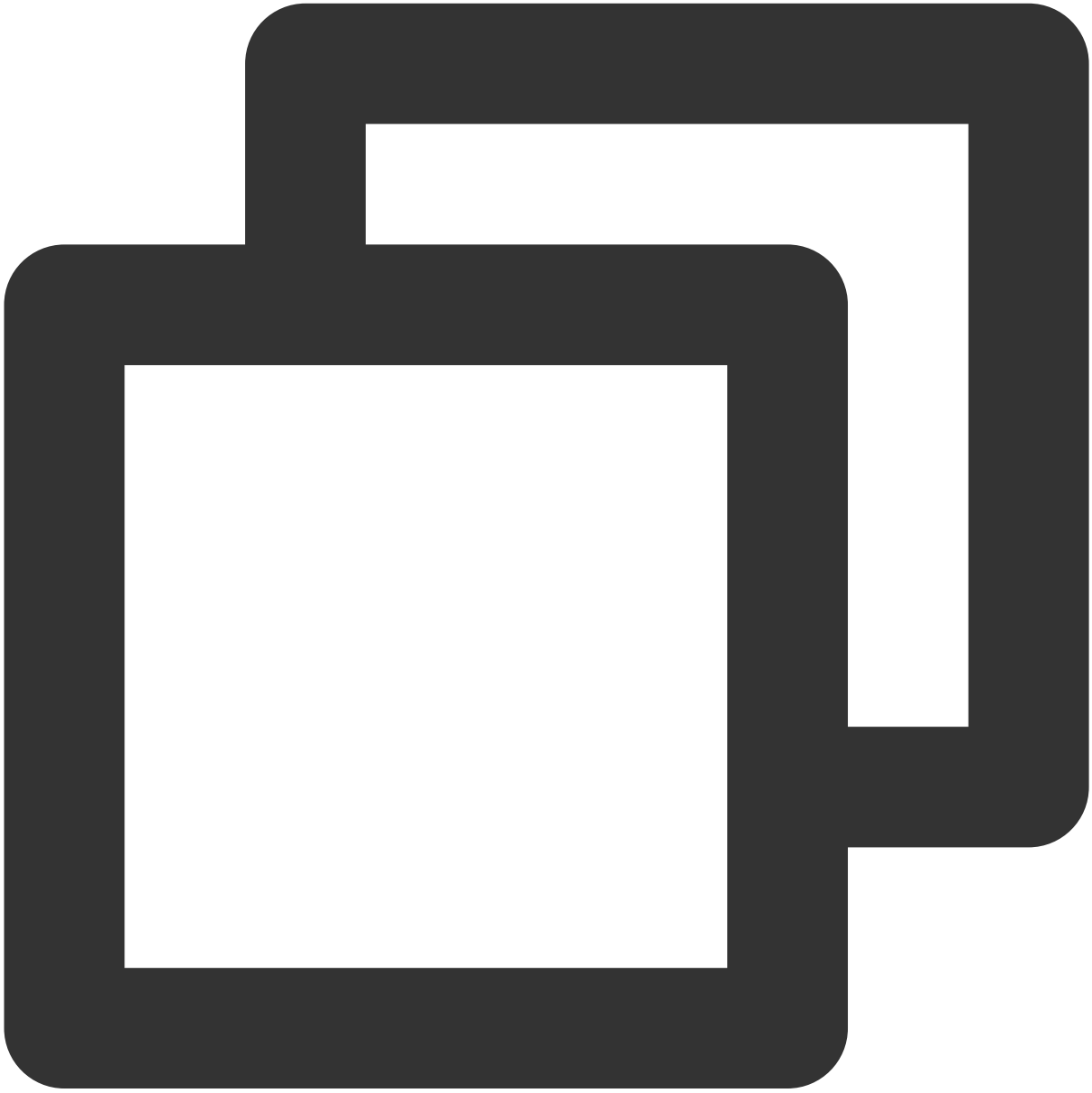
Parameter	Type	Description
teamID	int	Team ID, which is used in range voice mode only. When it is set to `0` (default), the team voice mode is used.

2. Set the voice mode

Calling this API to change the voice mode before room entry will affect the next room entry.

Calling this API to change the voice mode after room entry will directly change the current voice mode.
This parameter will not be automatically reset to `MODE_WORLD` upon room exit. Therefore, once you decide to call this method, do so before each call of `EnterRoom` .

Function prototype



```
ITMGRoom int SetRangeAudioMode(RANGE_AUDIO_MODE rangeAudioMode)
```

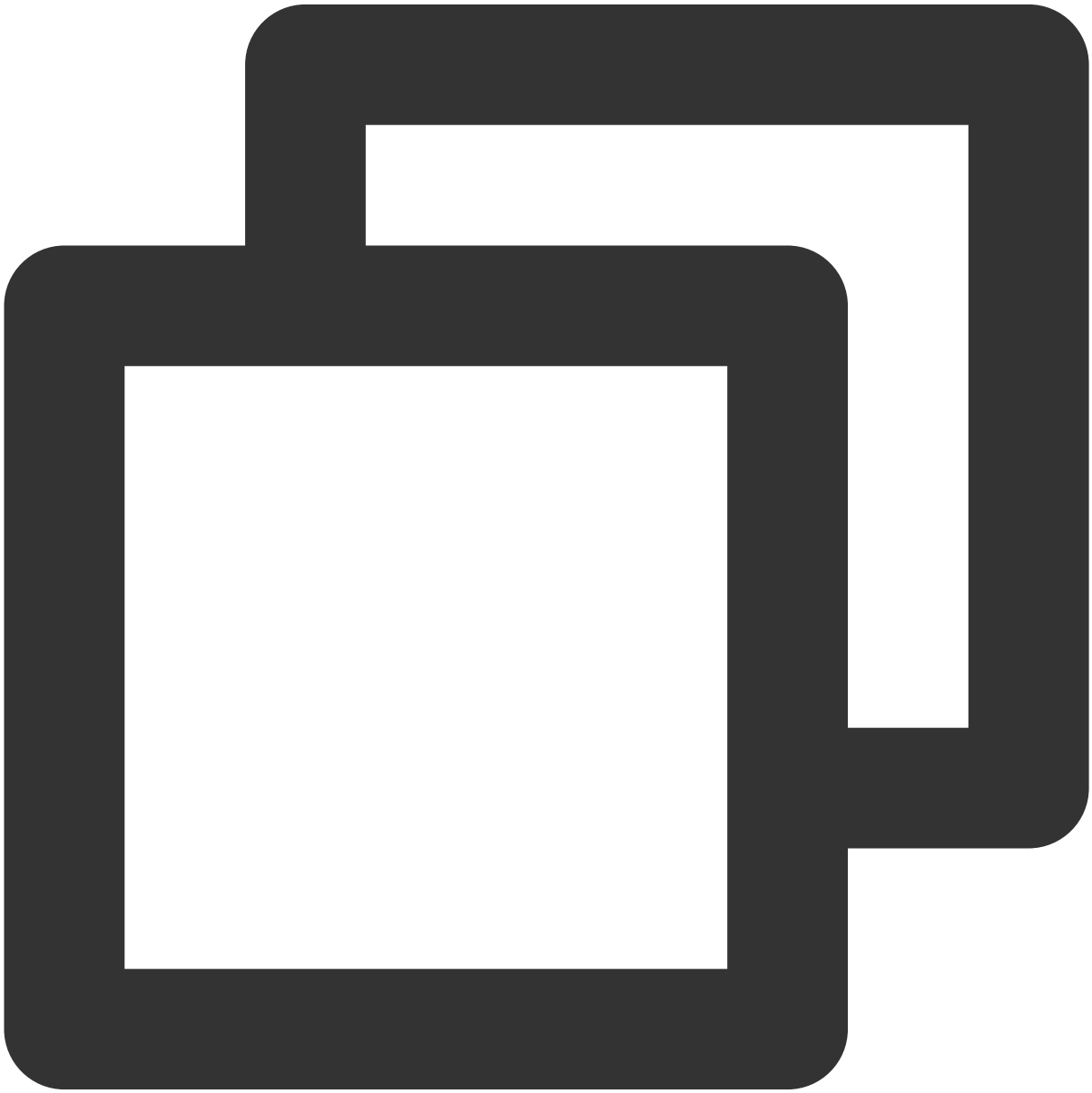
Parameter	Type	Description

rangeAudioMode	int	`0` (MODE_WORLD): everyone; `1` (MODE_TEAM): team only.

3. Enter a voice chat room

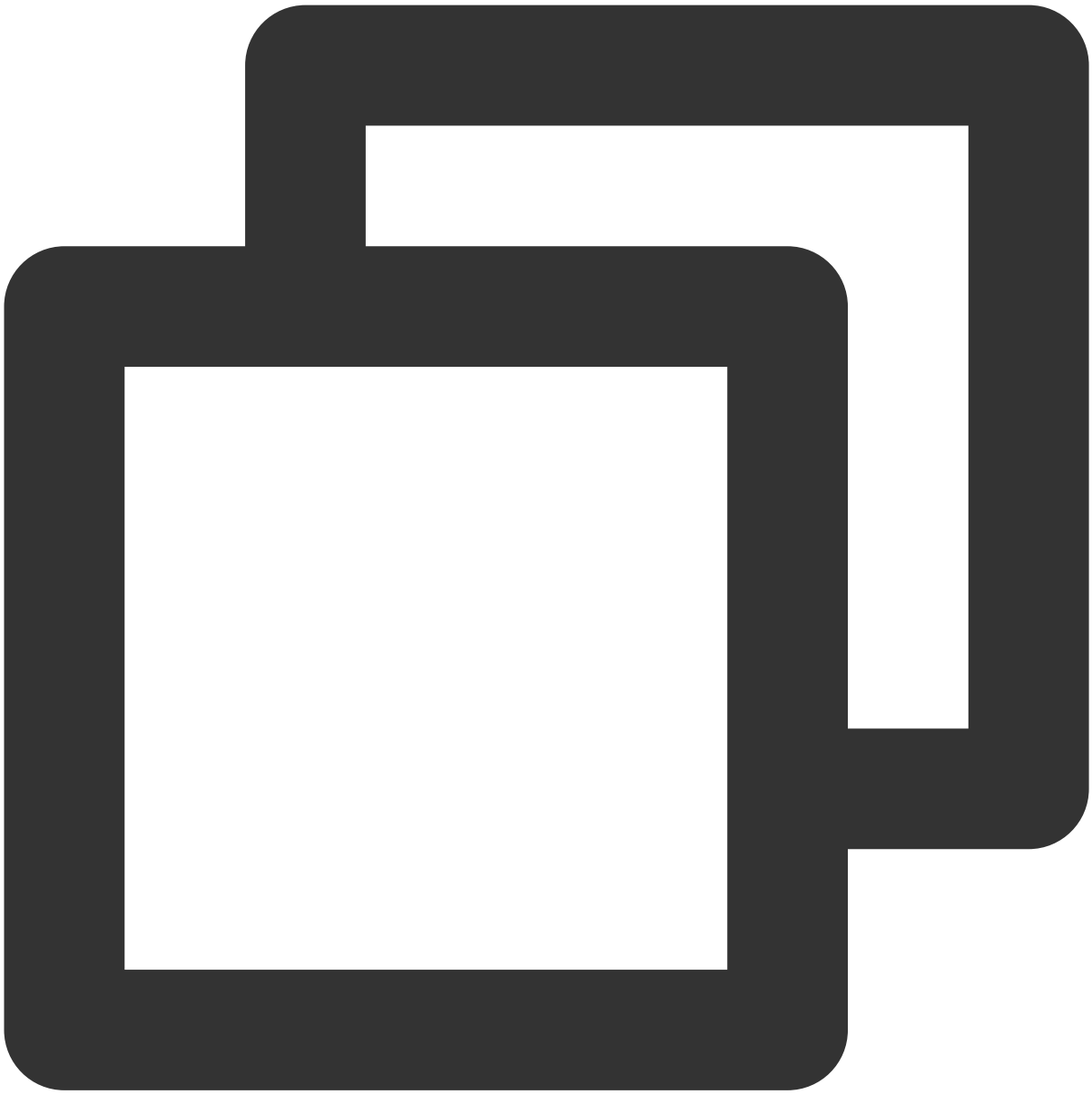
Before entering a voice chat room by calling `EnterRoom`, you need to call the following two APIs: `SetRangeAudioTeamID` and `SetRangeAudioMode`.

Function prototype



```
ITMGContext.GetInstance(this).EnterRoom(roomId,ITMG_ROOM_TYPE_FLUENCY, authBuffer);
```

Choose smooth sound quality when entering the voice chat room, and monitor and process the callback for room entry.



```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ENTER_ROOM == type)
        {
               // Analyze the returned data
            int nErrCode = data.getIntExtra("result" , -1);
            String strErrMsg = data.getStringExtra("error_info");

            if (nErrCode == AVError.AV_OK)
```

```
            {
                //If you receive a success response for room entry, you can proceed
                ScrollView_ShowLog("EnterRoom success");
                Log.i(TAG,"EnterRoom success!");
            }
            else
            {
                //If you fail to enter the room, you need to analyze the returned e
                ScrollView_ShowLog("EnterRoom fail :" + strErrMsg);
                Log.i(TAG,"EnterRoom fail!");
            }
        }
    }
```

Once you enter the room, call `UpdateAudioRecvRange` (once at least), and call `UpdateSelfPosition` once per frame.

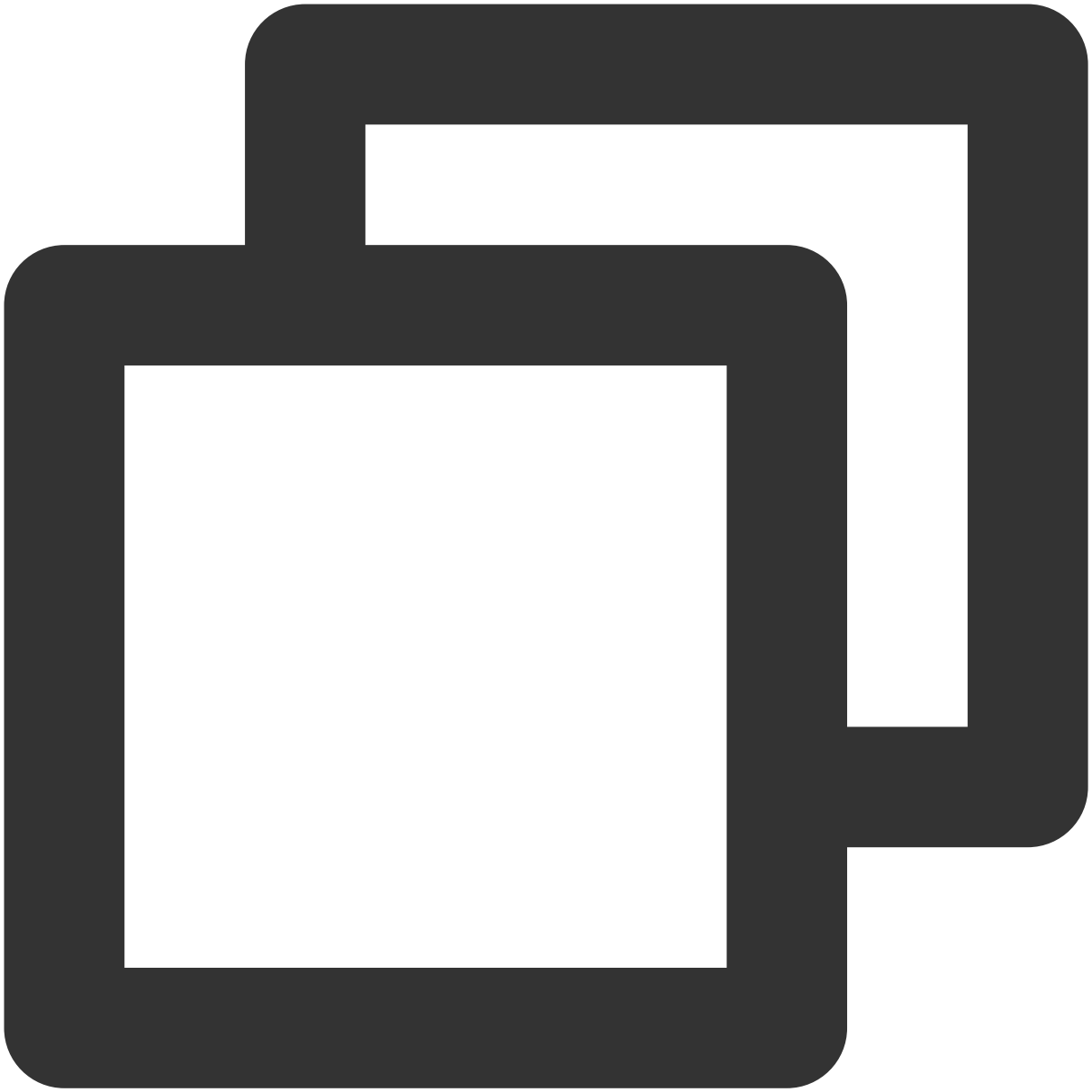
4. Set the voice reception range

This method is used to set the voice reception range (subject to the game engine) and can be called **only after successful room entry**.

This method must be used together with `UpdateSelfPosition` to update the sound source position.

This method needs to be called only once to take effect, and the parameter value can be modified.

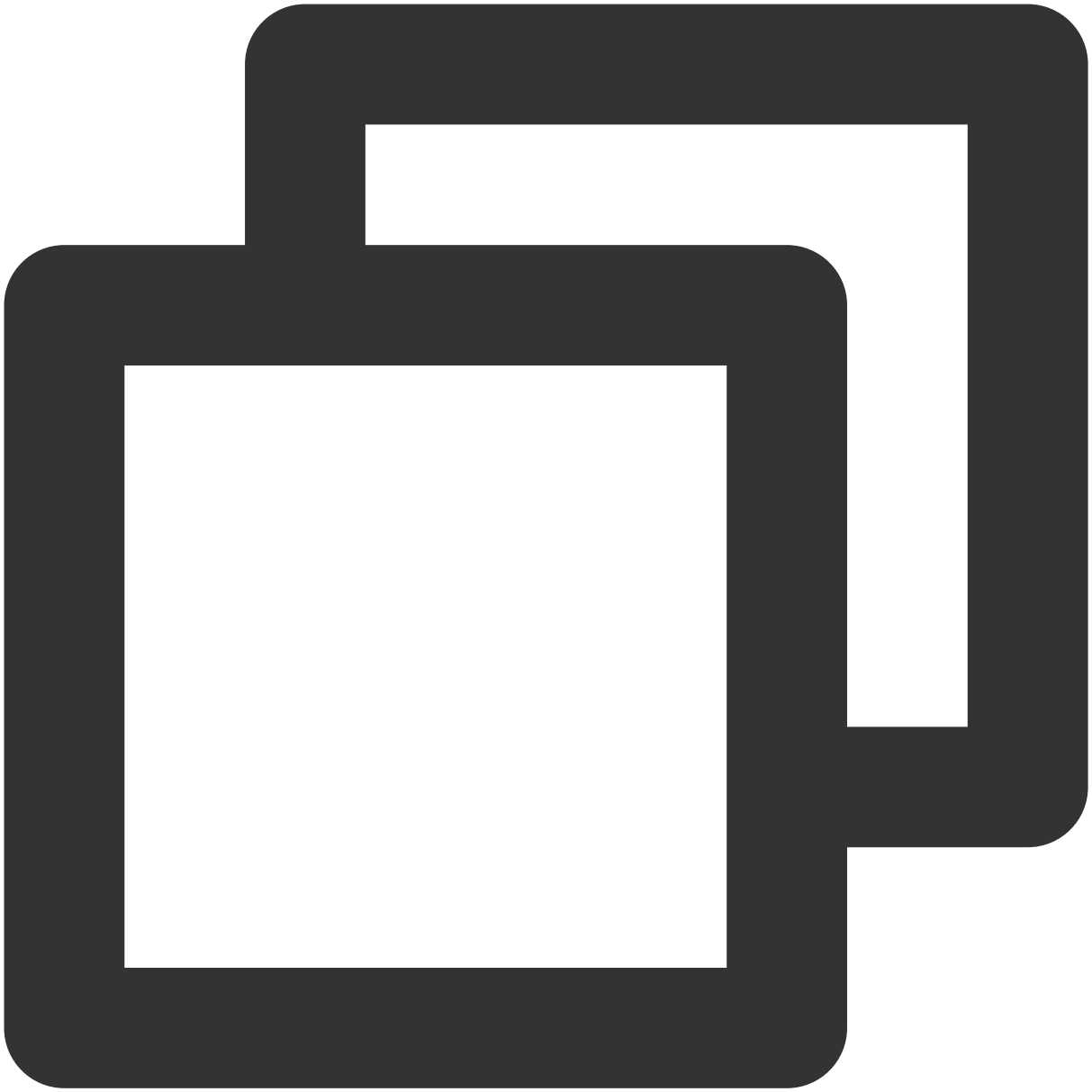
Function prototype



```
ITMGRoom int UpdateAudioRecvRange(int range)
```

Parameter	Type	Description
range	int	Maximum voice reception range in the distance unit used by the game engine.

Sample code



```
ITMGContext.GetInstance().GetRoom().UpdateAudioRecvRange(300);
```

5. Update the sound source position

The purpose of updating the sound source position is to inform the server of the local player's position. The system implements range voice by checking the **local coordinates and voice reception range** against the **remote coordinates and voice reception range**.

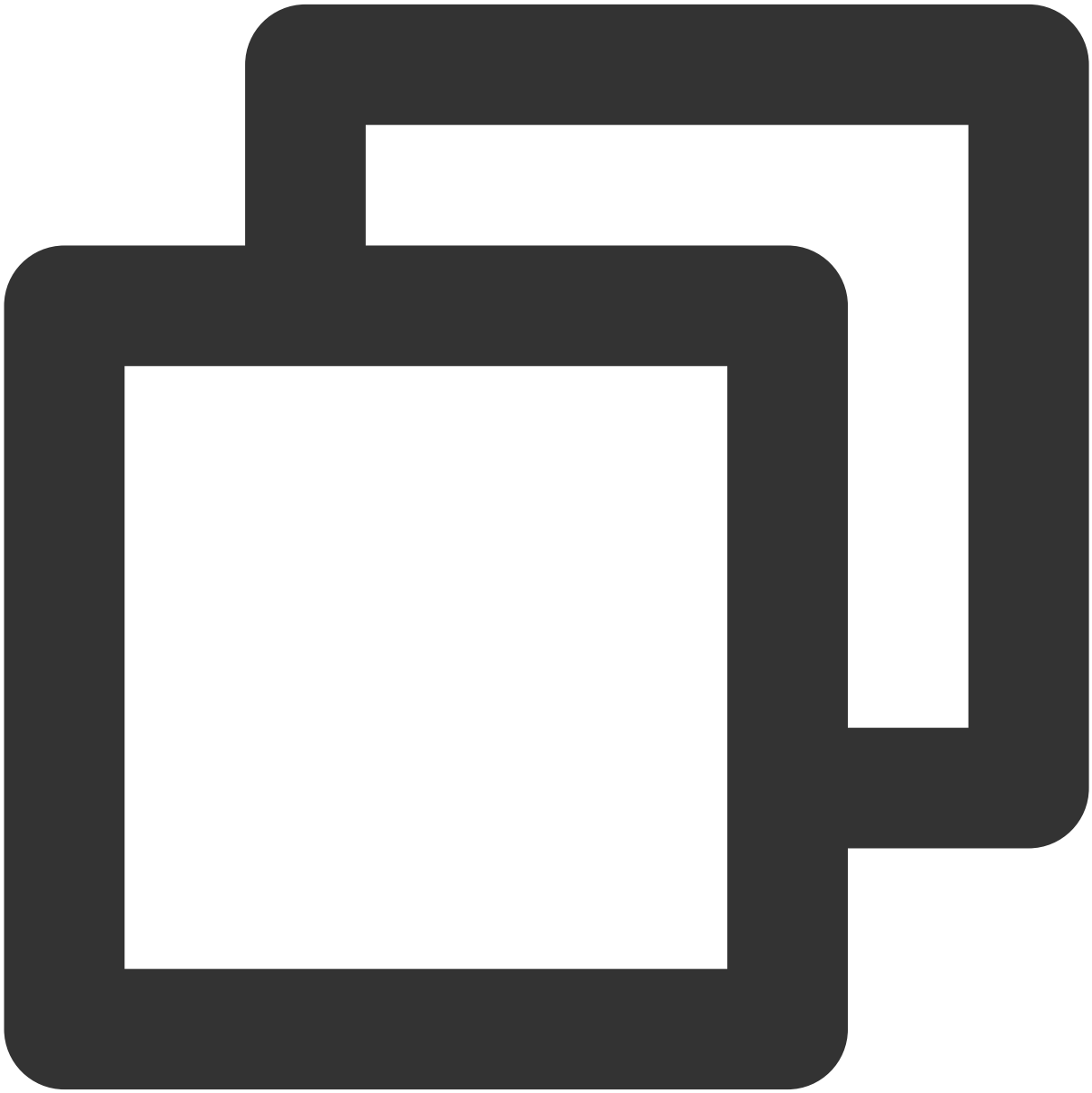
This API is used to update the sound source position information. It can be **called only after successful room entry** and needs to be **called once per frame**. Taking the Unity engine as an example, this API needs to be called in

`Update`.

If range voice is enabled, the sound source position must be updated. Even if range determination is not required, **this API still needs to be called once after room entry**.

If you want to enable 3D sound effect at the same time, set the `axisForward`, `axisRight`, and `axisUp` parameters as specified in "4. Update the sound source position (for the 3D sound effect)" in Range Voice.

Function prototype



```
public abstract int UpdateSelfPosition(int position[3], float axisForward[3], float
```

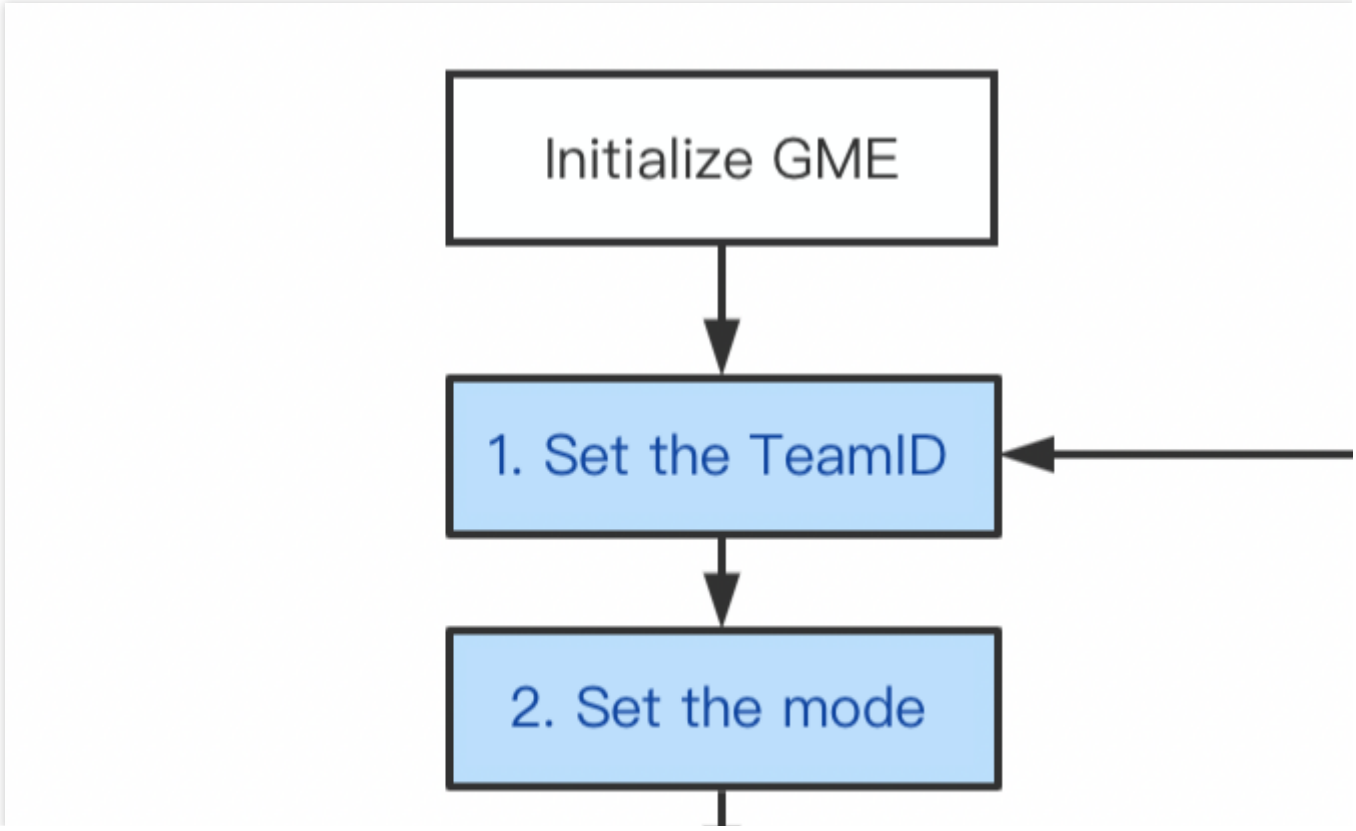
Parameter	Type	Description
position	int[]	Your own position (forward, right, and up) in the world coordinate system
axisForward	float[]	This parameter can be ignored in this product.
axisRight	float[]	This parameter can be ignored in this product.
axisUp	float[]	This parameter can be ignored in this product.

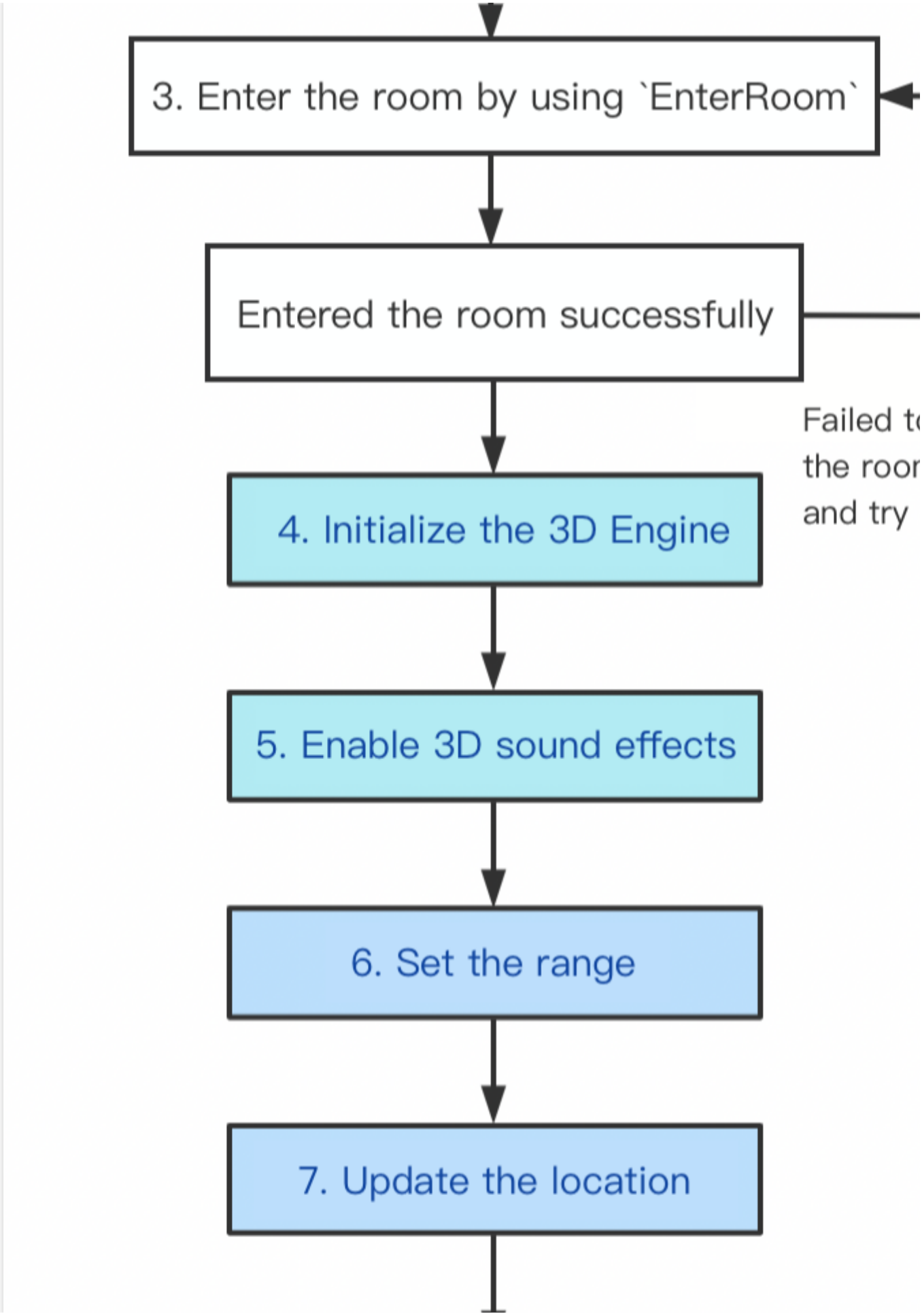
Range voice combined with 3D sound effects

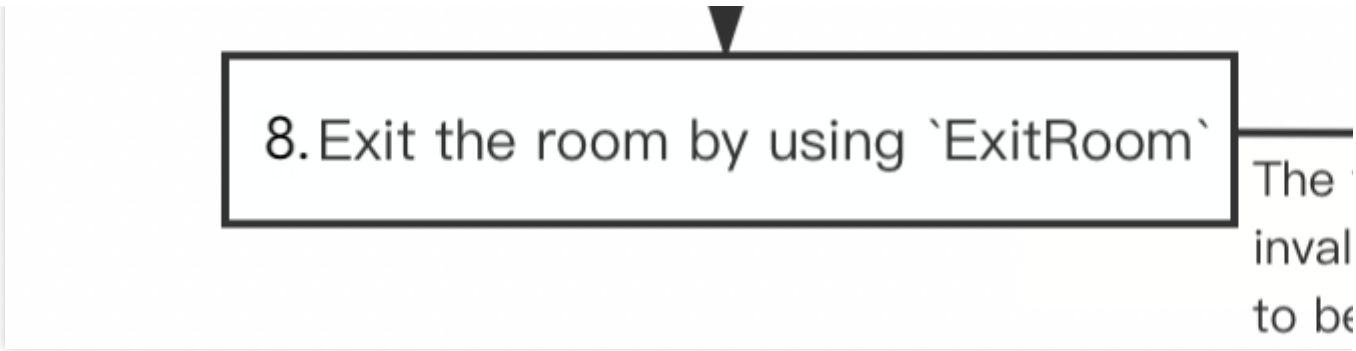
The range voice feature controls the reachability of sound through distance. It is recommended to use with 3D sound effects if you want a more immersive experience.

Directions

If you want to use 3D sound effects while using range voice, please complete the step 1, 2, 3 and then initialize 3D engine and open 3D sound effects.







Note

The blue part of the flowchart shows the process of the 3D sound effect feature.

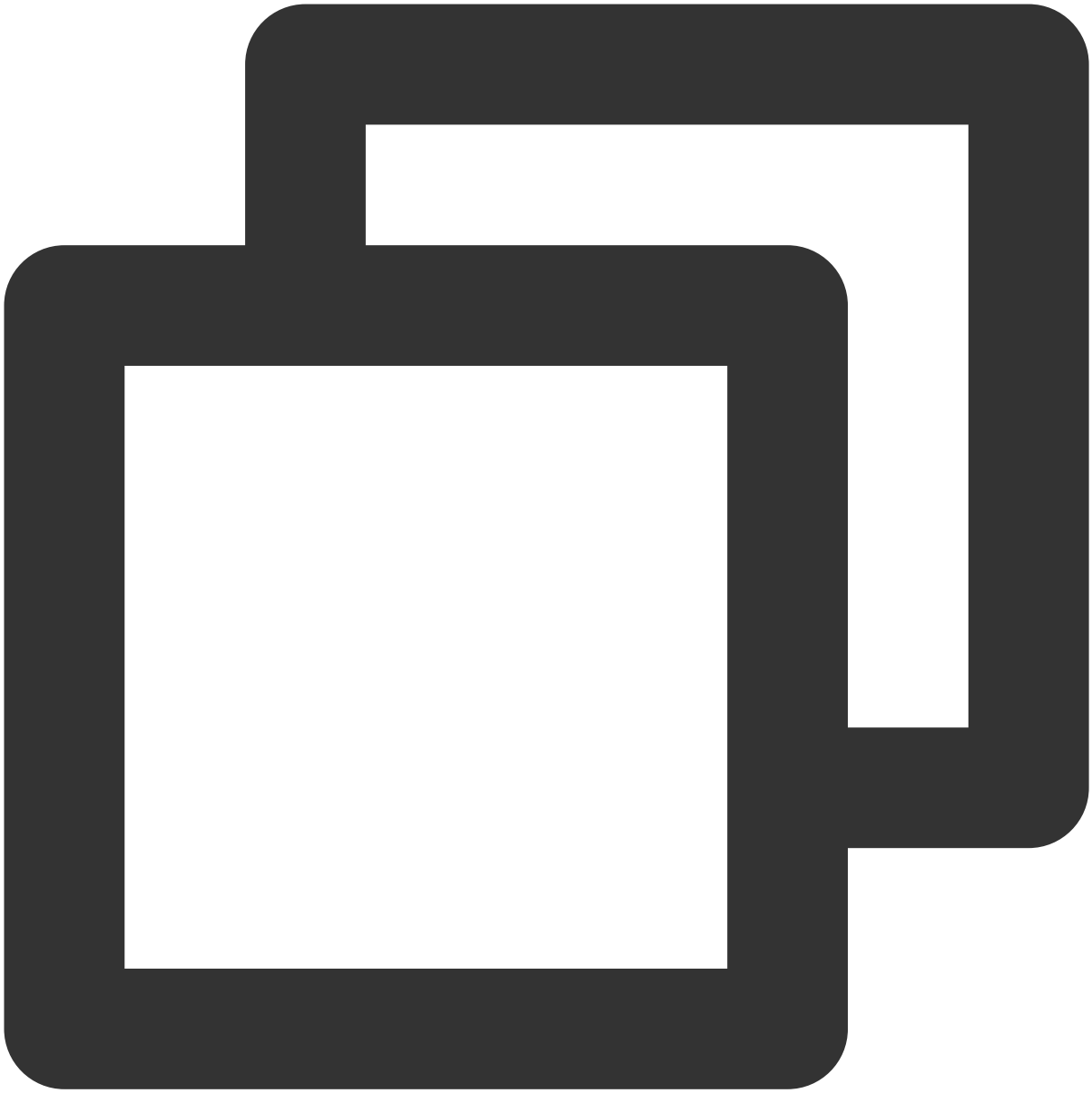
Prerequisites

Perform steps 1, 2, and 3 as instructed in Directions.

4. Initialize the 3D sound effect engine

This function is used to initialize the 3D sound effect engine and needs to be called after room entry. You must call this API before using the 3D sound effect, even if you want to enable only the 3D sound effect reception rather than the 3D sound effect playback.

Function prototype



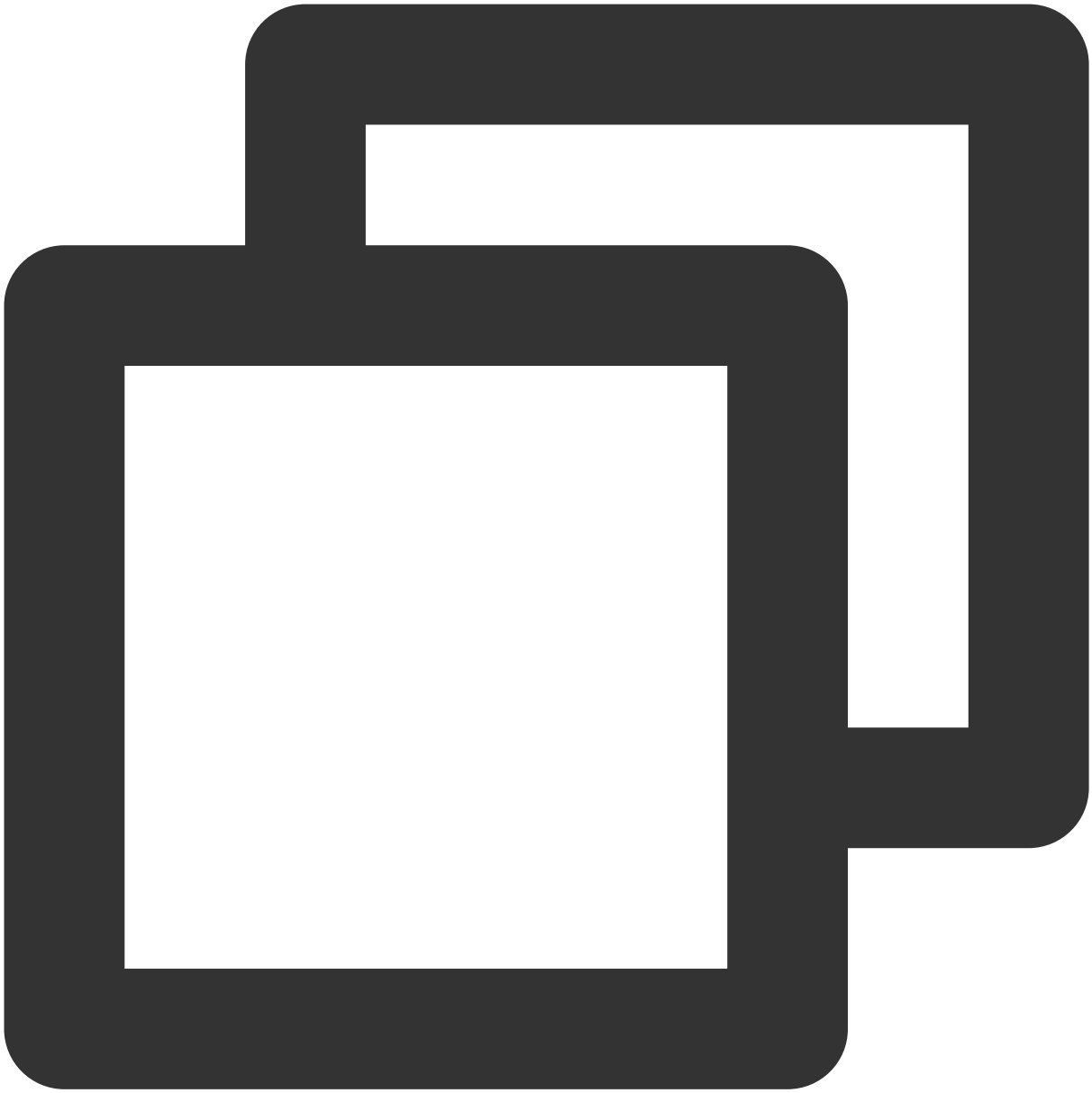
```
public abstract int InitSpatializer(string modelPath)
```

Parameter	Type	Description
modelPath	string	Enter null.

5. Enable/disable the 3D sound effect

This function is used to enable or disable the 3D sound effect. You can hear the 3D sound after enabling it.

Function prototype



```
public abstract int EnableSpatializer(bool enable, bool applyToTeam)
```

Parameter	Type	Description
enable	bool	You can hear the 3D sound effect after enabling it.
applyToTeam	bool	This parameter specifies whether 3D sound effect is enabled within the team and takes effect only when `enable` is `true` .

The `IsEnableSpatializer` API can be used to get the 3D sound effect status.

6. Set the voice reception range (for the 3D sound effect)

This method is used to set the voice reception range (subject to the game engine) and can be called **only after successful room entry**.

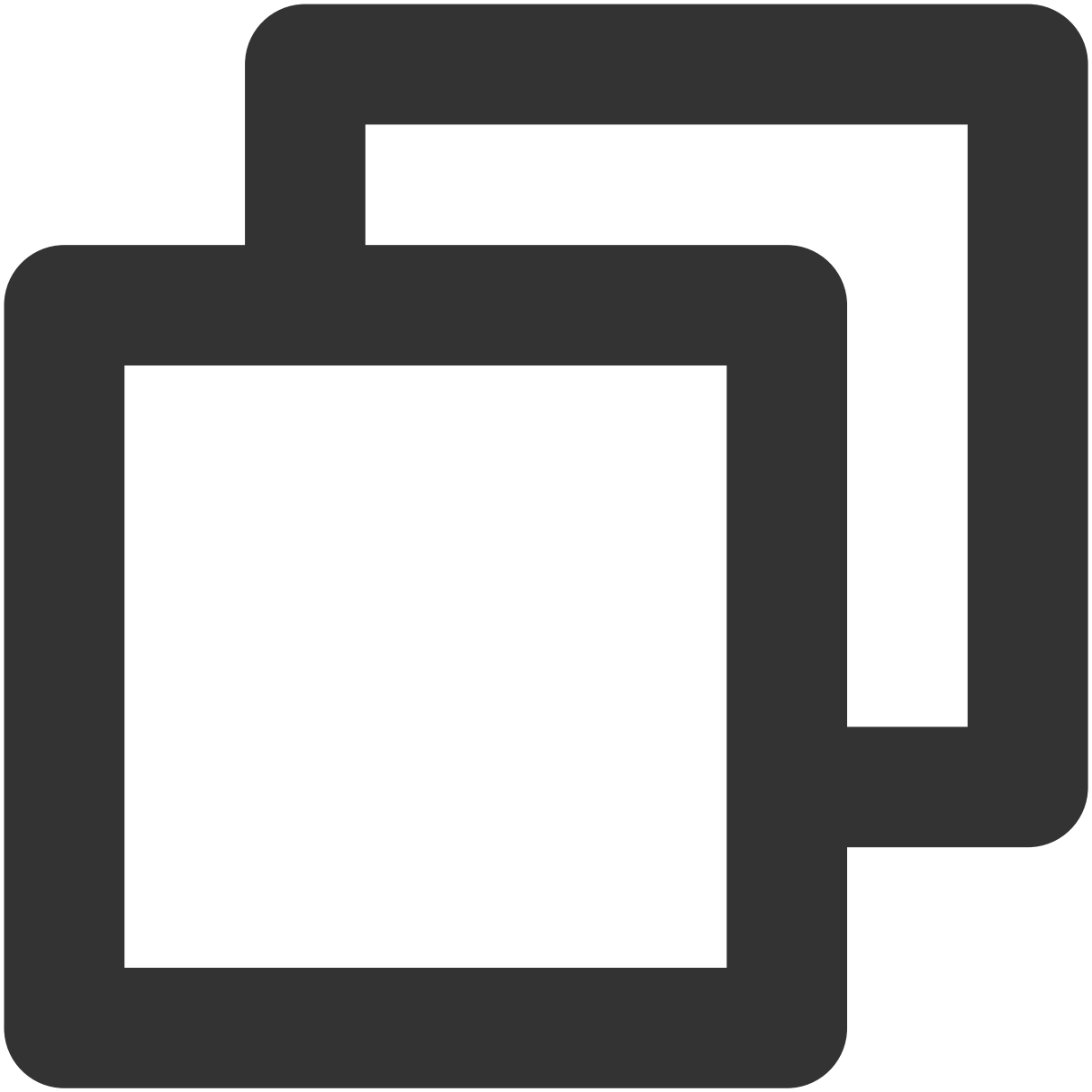
This method must be used together with `UpdateSelfPosition` to update the sound source position.

This method needs to be called only once to take effect.

In the 3D sound effect, the sound will begin to attenuate to almost zero as the distance to the sound source exceeds a specified threshold (range/10).

For more information on the relationship between the distance and sound attenuation, see the appendix.

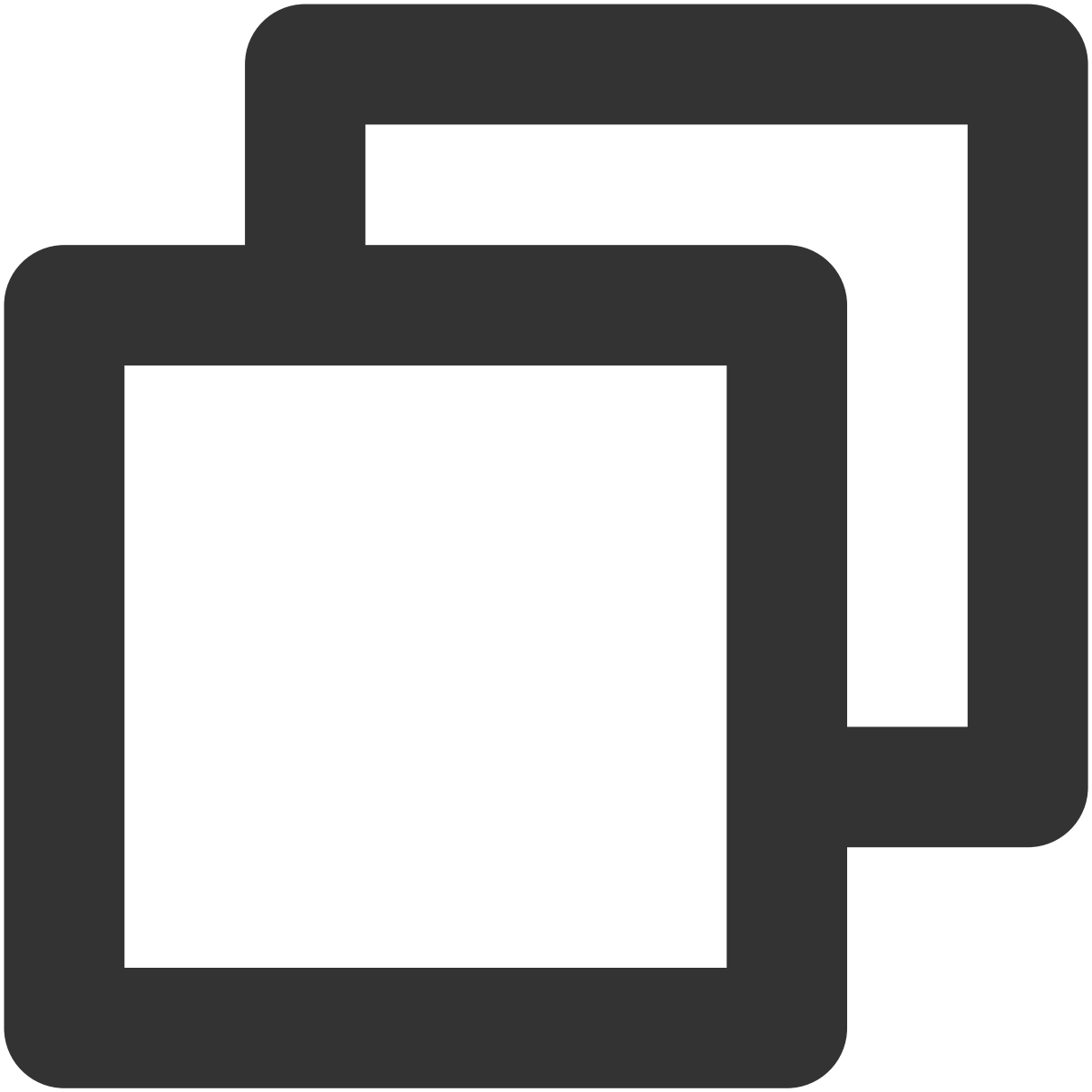
Function prototype



```
ITMGRoom int UpdateAudioRecvRange(int range)
```

Parameter	Type	Description
range	int	Maximum voice reception range in the distance unit used by the game engine.

Sample code



```
ITMGContext.GetInstance().GetRoom().UpdateAudioRecvRange(300);
```

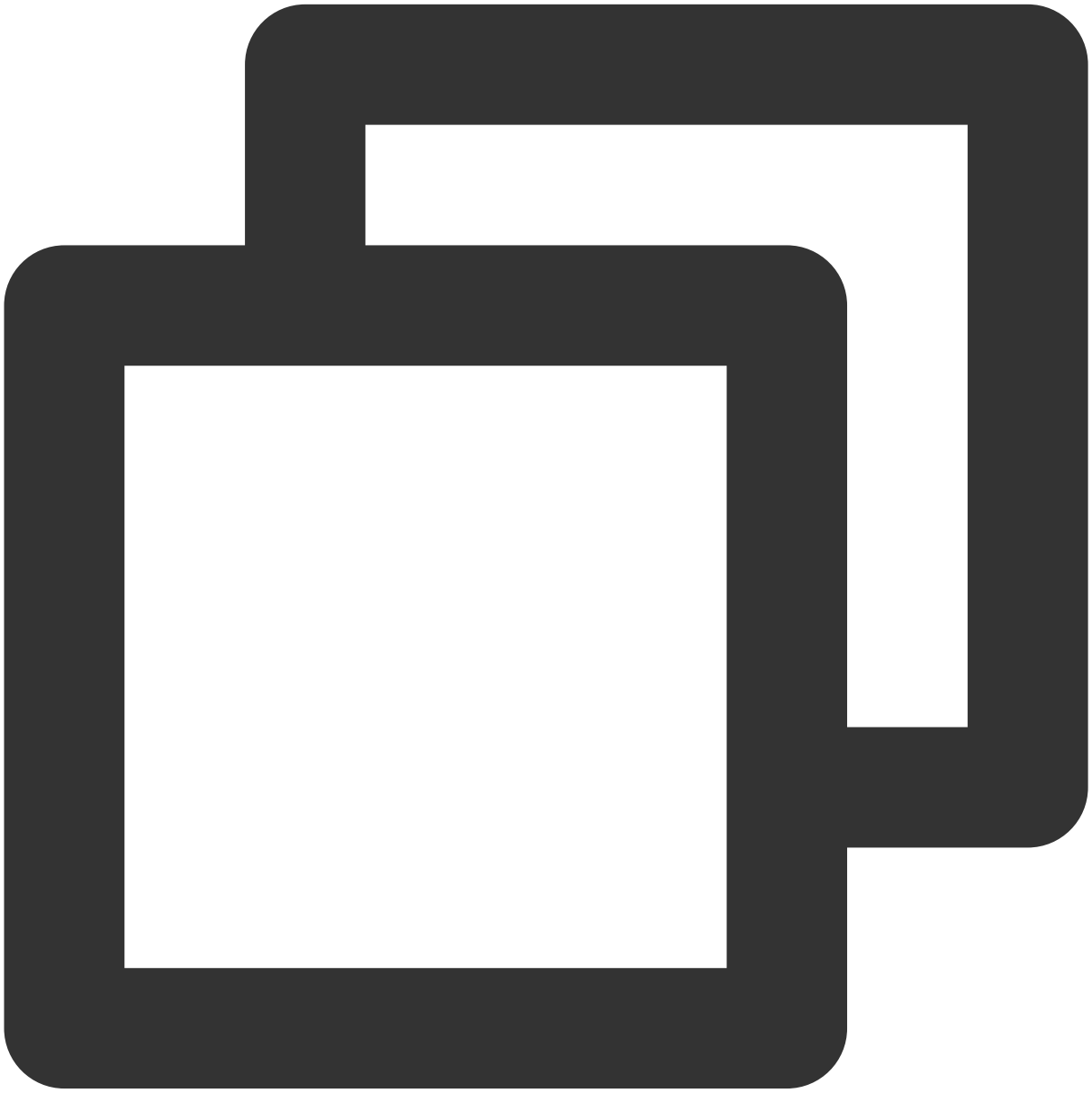
7. Update the sound source position (for the 3D sound effect)

The purpose of updating the sound source position is to inform the server of the local player's position. The system implements range voice by checking the **local coordinates and voice reception range** against the **remote coordinates and voice reception range**.

This API is used to update the sound source position information. It can be **called only after successful room entry** and needs to be **called once per frame**. Taking the Unity engine as an example, this API needs to be called in `Update`.

Directly copy and call the sample code to use this feature.

Function prototype



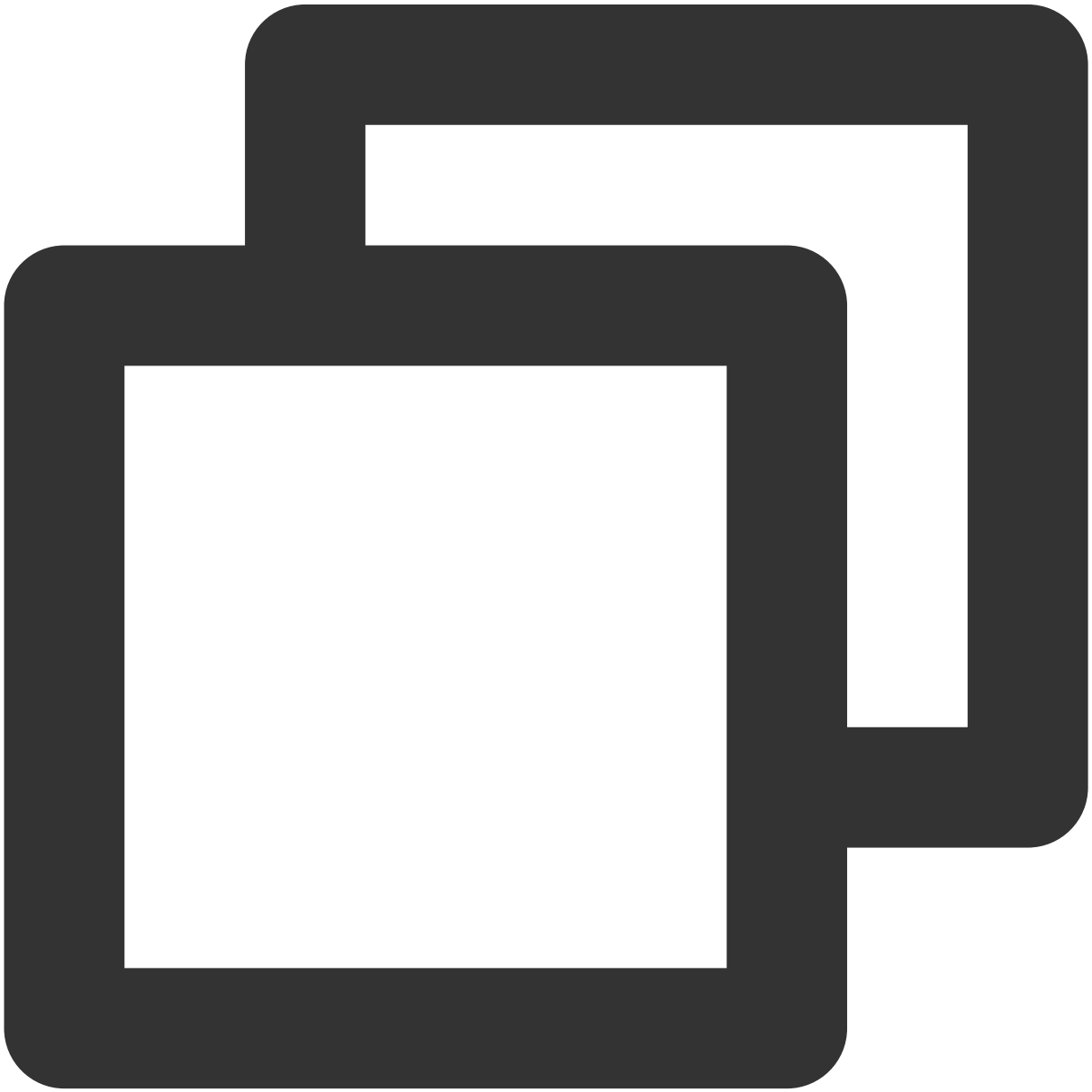
```
public abstract int UpdateSelfPosition(int position[3], float axisForward[3], float
```

Parameter	Type	Description

position	int[]	Local position (forward, right, and up) in the world coordinate system
axisForward	float[]	Forward vector in the local coordinate system
axisRight	float[]	Right vector in the local coordinate system
axisUp	float[]	Up vector in the local coordinate system

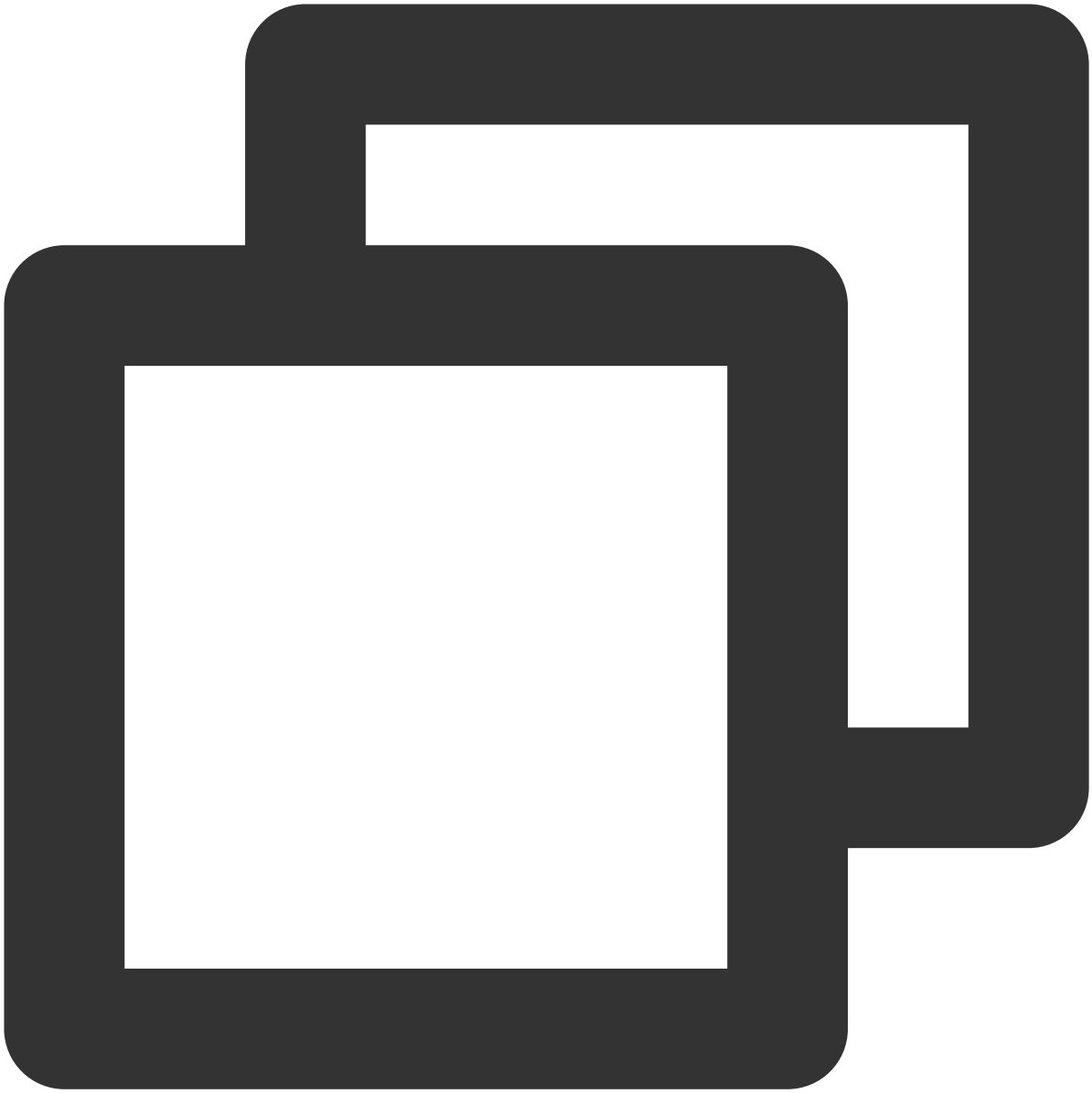
Sample code

Unreal



```
FVector cameraLocation = UGameplayStatics::GetPlayerCameraManager(GetWorld(), 0)->G
FRotator cameraRotation = UGameplayStatics::GetPlayerCameraManager(GetWorld(), 0)->
int position[] = {
    (int)cameraLocation.X,
    (int)cameraLocation.Y,
    (int)cameraLocation.Z };
FMatrix matrix = ((FRotationMatrix)cameraRotation);
float forward[] = {
    matrix.GetColumn(0).X,
    matrix.GetColumn(1).X,
    matrix.GetColumn(2).X };
float right[] = {
    matrix.GetColumn(0).Y,
    matrix.GetColumn(1).Y,
    matrix.GetColumn(2).Y };
float up[] = {
    matrix.GetColumn(0).Z,
    matrix.GetColumn(1).Z,
    matrix.GetColumn(2).Z};
ITMGContextGetInstance()->GetRoom()->UpdateSelfPosition(position, forward, right, u
```

Unity



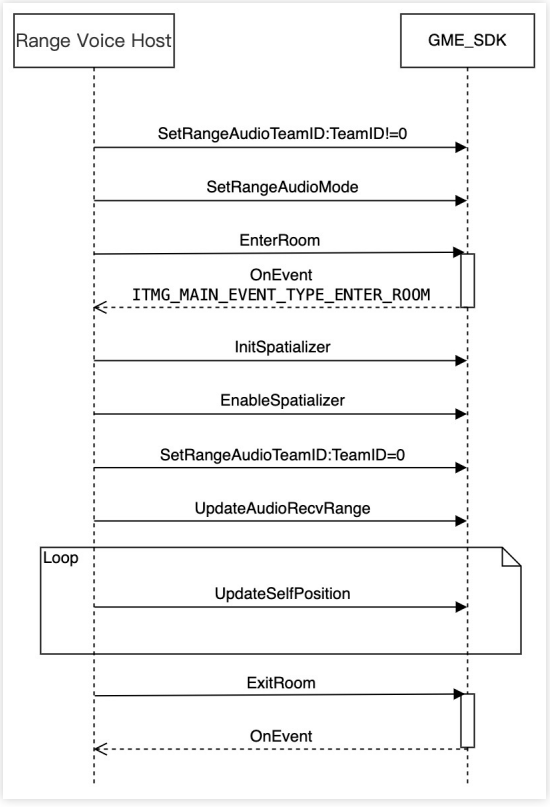
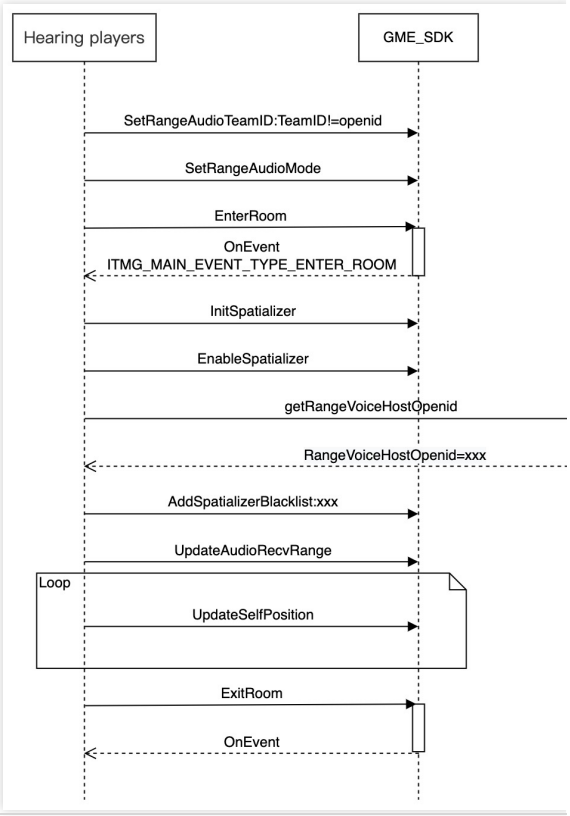
```
Transform selftrans = currentPlayer.gameObject.transform;
Matrix4x4 matrix = Matrix4x4.TRS(Vector3.zero, selftrans.rotation, Vector3.one);
int[] position = new int[3] {
    selftrans.position.z,
    selftrans.position.x,
    selftrans.position.y};
float[] axisForward = new float[3] {
    matrix.m22,
    matrix.m02,
    matrix.m12 };
float[] axisRight = new float[3] {
```

```
    matrix.m20,
    matrix.m00,
    matrix.m10 };
float[] axisUp = new float[3] {
    matrix.m21,
    matrix.m01,
    matrix.m11 };
ITMGContext.GetInstance().GetRoom().UpdateSelfPosition(position, axisForward, axisR
```

Disabling the 3D sound effect for the host

If a player in the scenario uses the range voice host mode, the player's voice needs to be heard by all players in the room. In this case, you need to add the host to the 3D sound effect blocklist of all the other players as instructed in 3D Sound Effect to prevent the host's voice from being attenuated by the 3D sound effect feature.

Below are the API call sequences of different roles:

API Call Sequence of the Host	API Call Sequence of a Listener
	

Appendix

Voice modes

Player voice reachability in different voice modes is as follows:

The table below lists the possible cases of voice reachability for player B in different voice modes if player A selects the "Everyone" mode:

In the Same Team	Within the Range	Voice Mode	Can A and B Hear Each Other
Same team	Yes	MODE_WORLD	Yes
		MODE_TEAM	Yes
	No	MODE_WORLD	Yes
		MODE_TEAM	Yes
Different team	Yes	MODE_WORLD	Yes
		MODE_TEAM	No
	No	MODE_WORLD	No
		MODE_TEAM	No

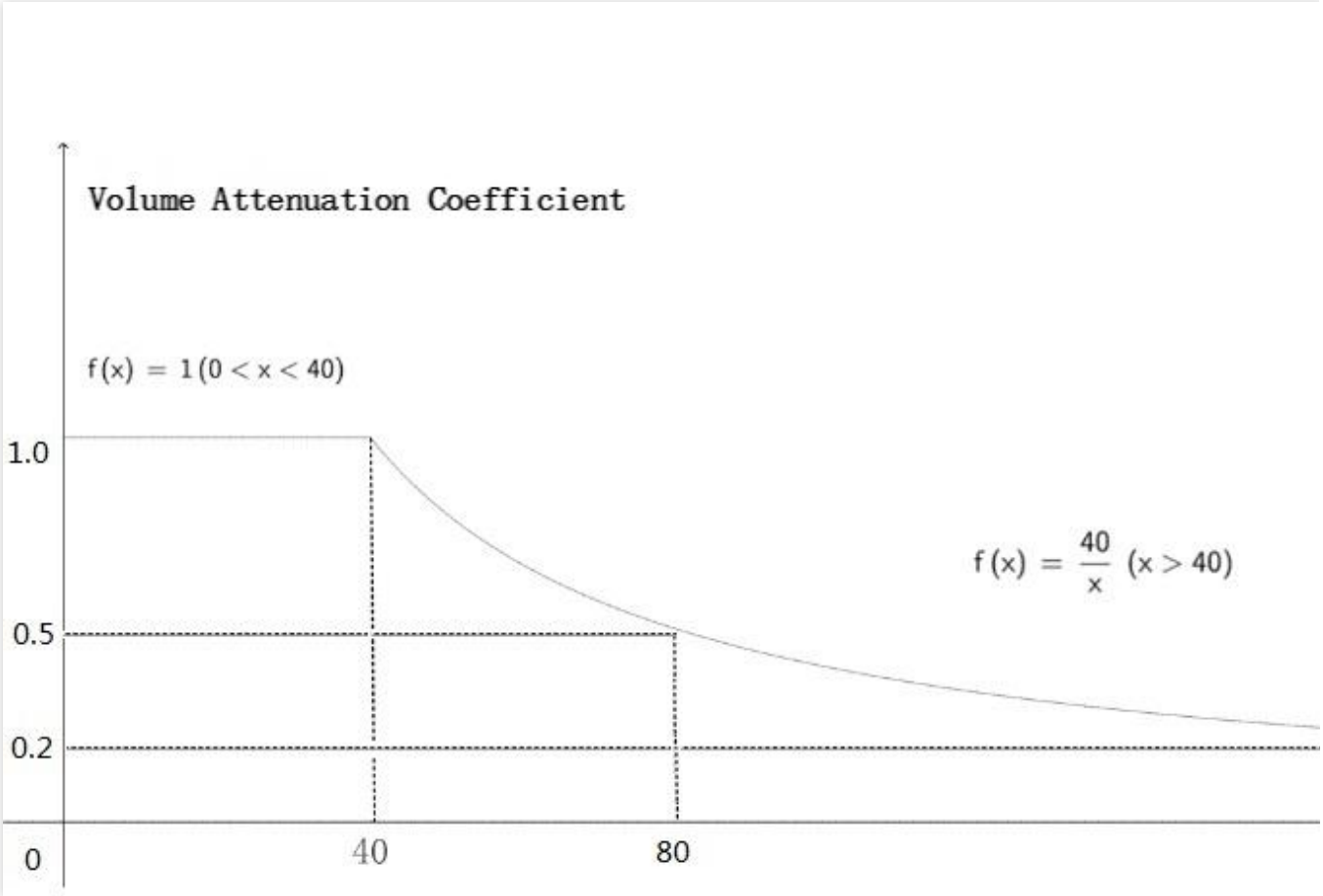
The table below lists the possible cases of voice reachability for player B in different voice modes if player A selects the "Team only" mode:

In the Same Team	Within the Range	Voice Mode	Can A and B Hear Each Other
Same team	Yes	MODE_WORLD	Yes
		MODE_TEAM	Yes
	No	MODE_WORLD	Yes
		MODE_TEAM	Yes
Different team	Yes	MODE_WORLD	No
		MODE_TEAM	No
	No	MODE_WORLD	No
		MODE_TEAM	No

The relationship between distance and sound attenuation

In the 3D sound effect, the sound will begin to attenuate to almost zero as the distance to the sound source exceeds a specified threshold (range/10).

Distance (Unit in the Engine)	Attenuation Coefficient
0 < N < range/10	1.0 (no attenuation)
N ≥ range/10	Range/10/N



3D Sound Effect

Last updated：2023-04-27 17:06:32

This document describes how to integrate with and debug GME APIs for 3D sound effects.

Use Cases

In general voice chat for room entry, the player voice has no 3D sound effects, and players can only have simple interactions with each other. With the 3D sound effect, players can expose their direction and position information while speaking, and their voice can change in real time along with the distance. The 3D sound effect feature delivers players a more real and immersive communication and battle experience in battle royale games.
Click here to download the demo and try out the 3D sound effect.

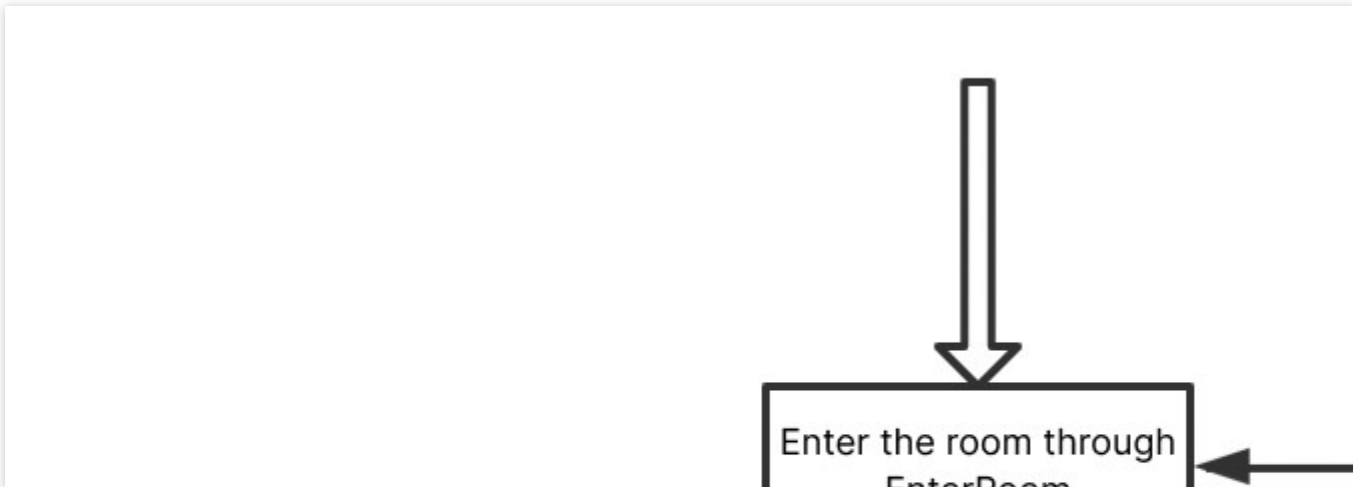
Prerequisites

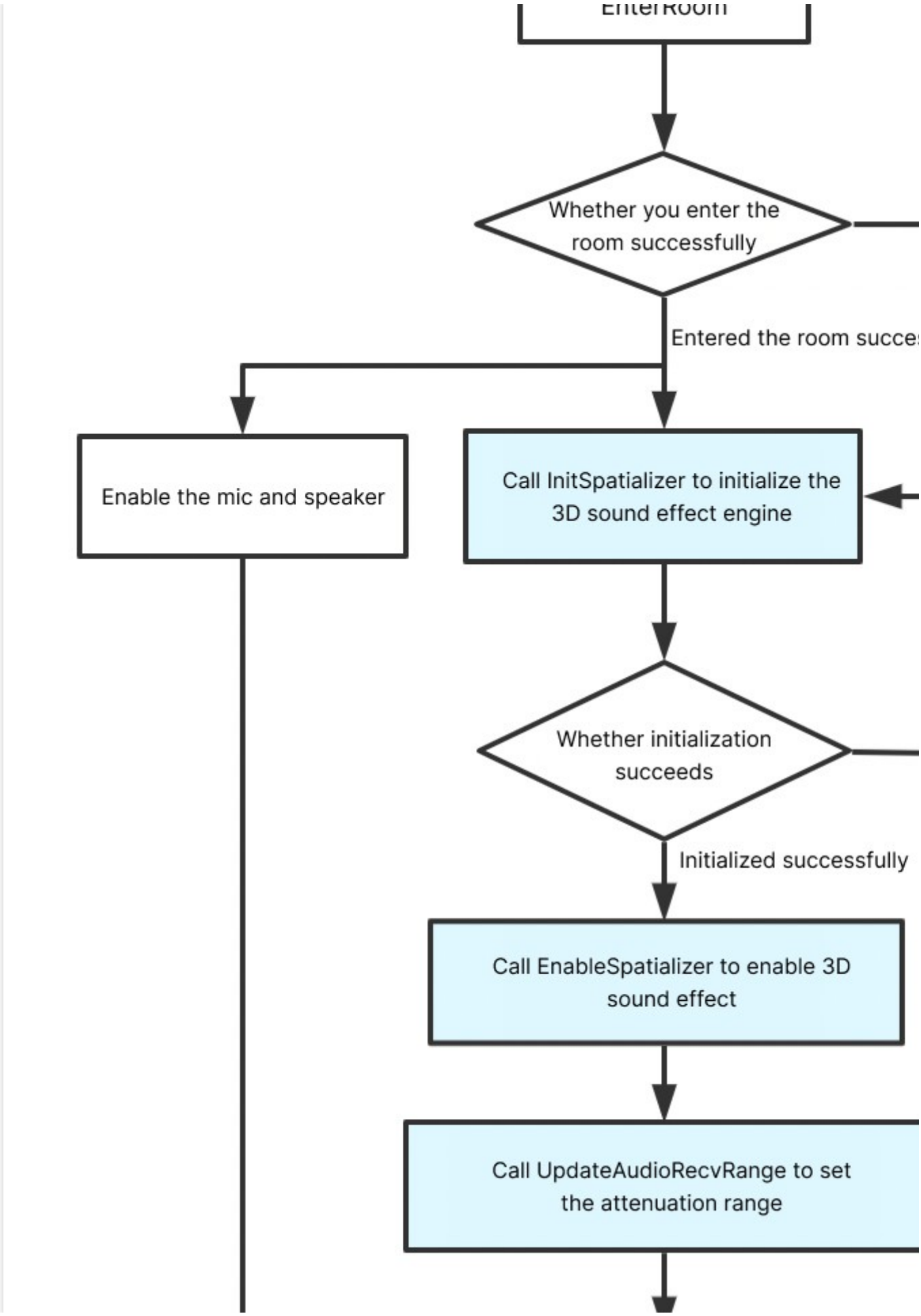
You have activated the voice chat service. For more information, see Activating Services.
You have integrated the GME SDK, including core APIs and voice chat APIs. For more information, see Quick Integration of Native SDK, Quick Integration of SDK for Unity, and Quick Integration of SDK for Unreal Engine.

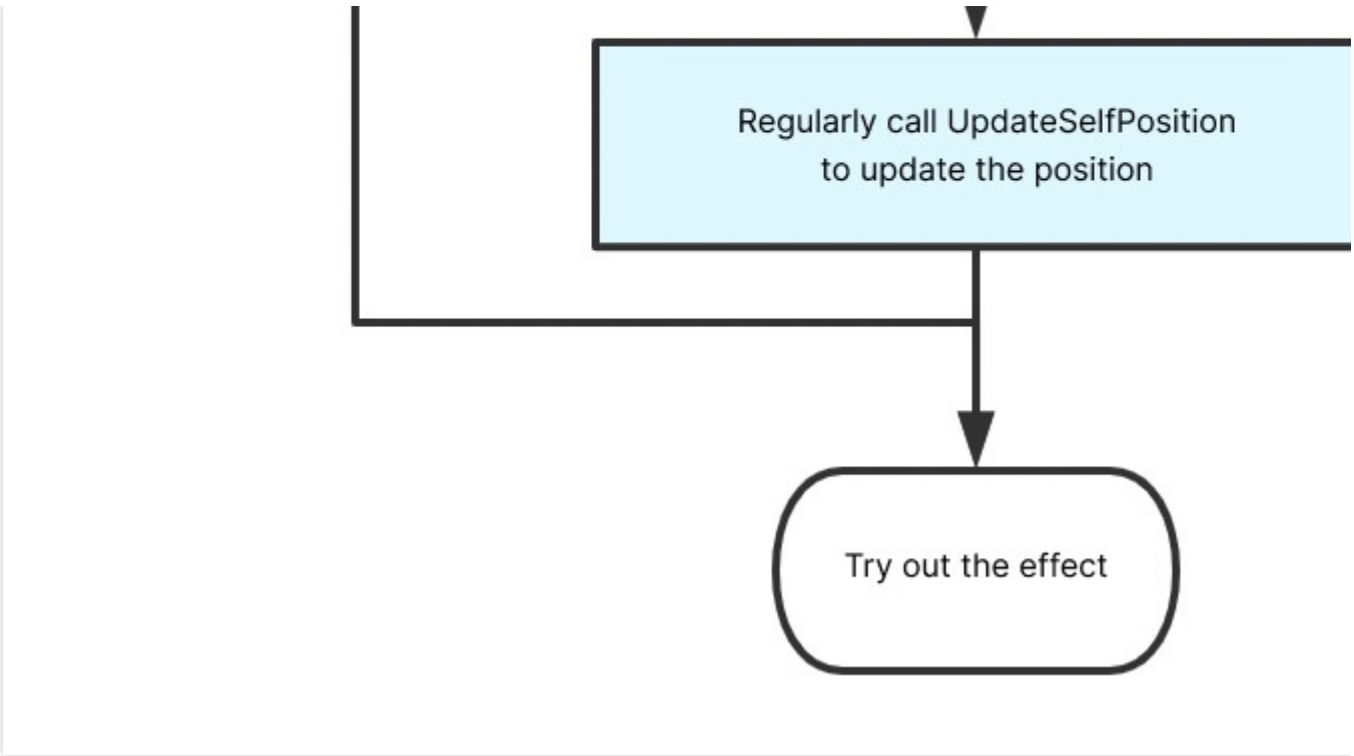
Directions

Implementation flowchart

The 3D sound effect implementation flowchart is as follows. The blue part is new integration steps compared with general voice chat for room entry:



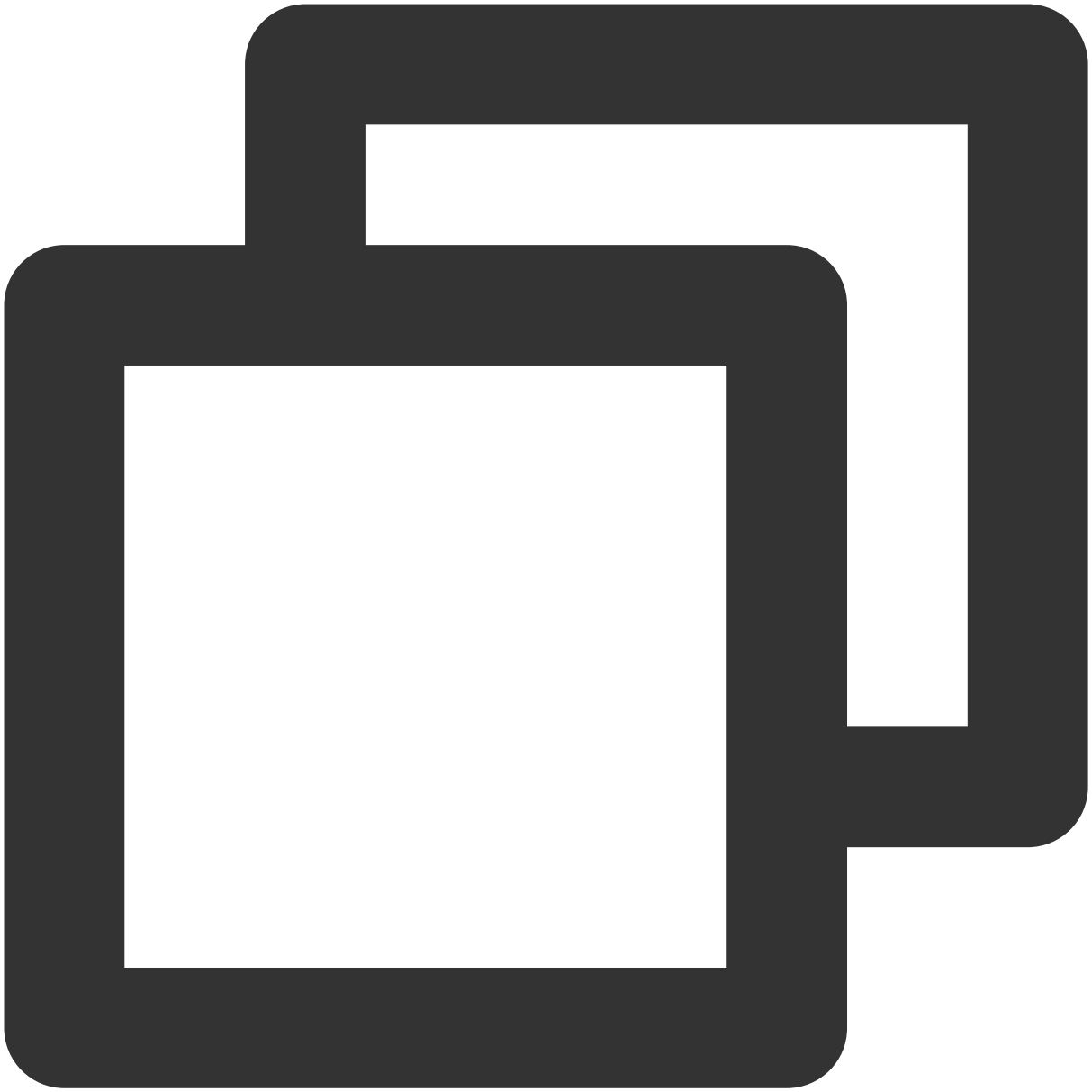




Initializing the 3D sound effect engine

This function is used to initialize the 3D sound effect engine and needs to be called after room entry. You must call this API before using the 3D sound effect, even if you want to enable only the 3D sound effect reception rather than the 3D sound effect playback.

Function prototype



```
public abstract int InitSpatializer(string modelPath)
```

Parameter	Type	Description
modelPath	string	The absolute path of the 3D sound effect resource file

The 3D sound effect resource file needs to be downloaded to your local disk, please select one according to the SDK version:

For SDK versions earlier than 2.8, click here to download (MD5 value: d0b76aa64c46598788c2f35f5a8a8694).

For SDK 2.8–2.9.5, click here to download (MD5 value: 3d4d04b3949e267e34ca809e8a0b9243).

For SDK 2.9.6 or later, as the 3D sound effect resource file is embedded, you can leave `modelPath` empty.

For GME SDK release updates, see Release Notes.

Note on resource path

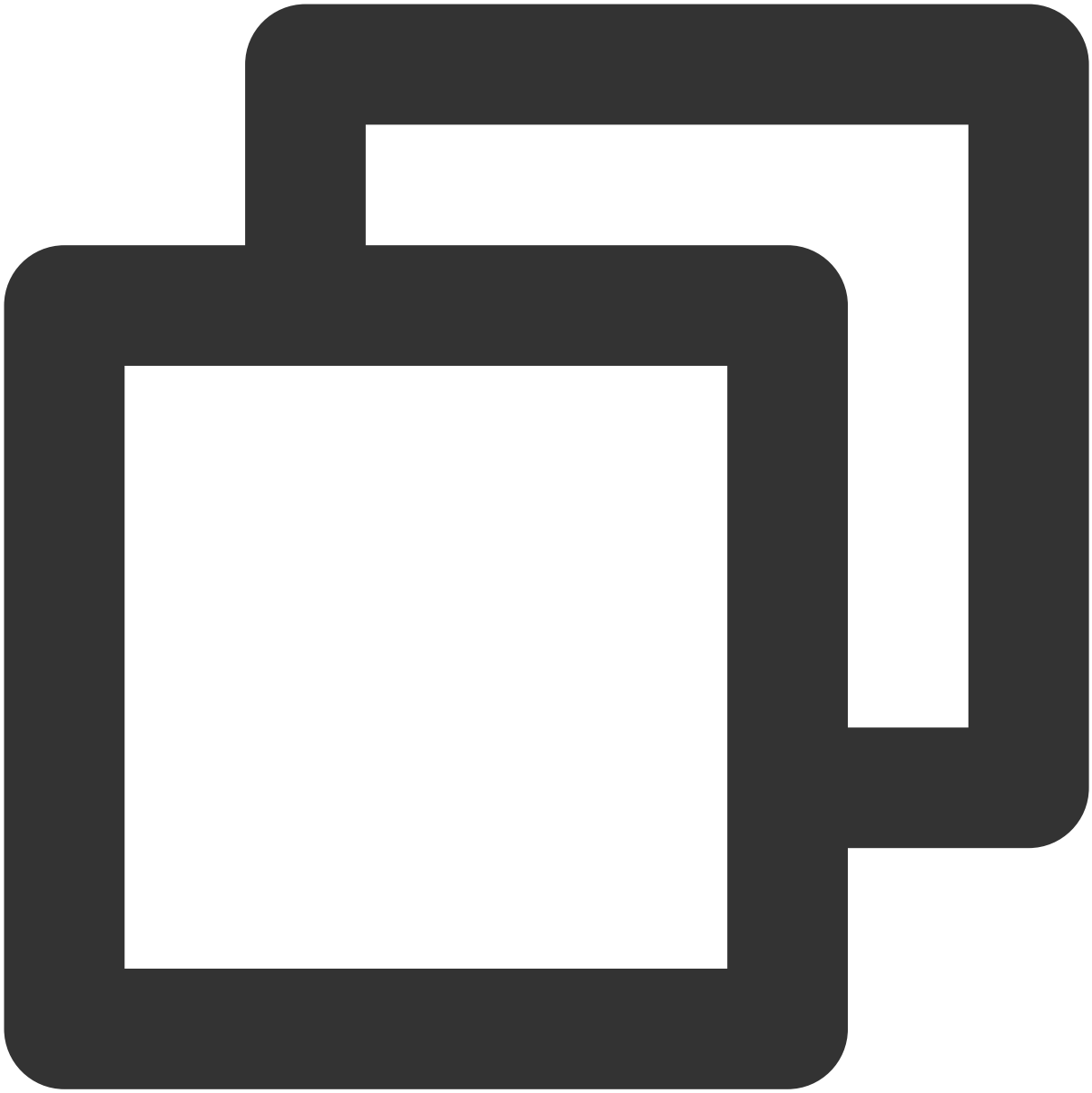
Taking Unity as an example, we recommend that you place the 3D sound effect resource file in the `StreamingAssets` directory of the project and copy it to the corresponding directory of different platforms based on the content of the **copyFileFromAssetsToPersistent** function in `SampleCode` .

Taking Unreal Engine as an example, copy the 3D model file and read the path based on the content of the **CopyAllAssetsToExternal** function in `SampleCode` .

Enabling or disabling the 3D sound effect

This function is used to enable or disable the 3D sound effect. You can hear the 3D sound after enabling it.

Function prototype



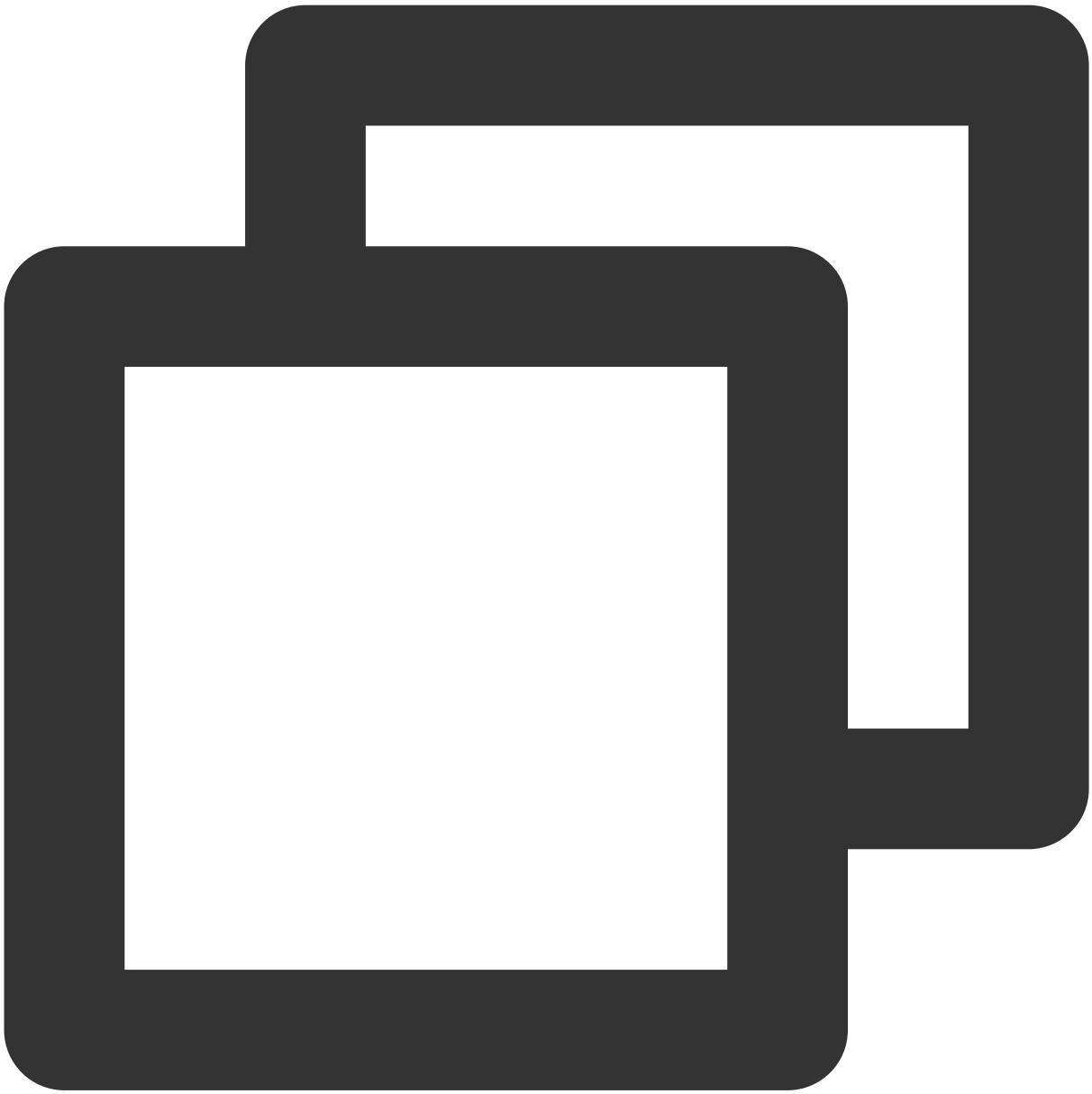
```
public abstract int EnableSpatializer(bool enable, bool applyToTeam)
```

Parameter	Type	Description
enable	bool	You can hear the 3D sound effect after enabling it.
applyToTeam	bool	This parameter specifies whether 3D sound effect is enabled within the team and takes effect only when `enable` is `true` .

Getting the current 3D sound effect status

This function is used to obtain the status of 3D sound effect.

Function prototype



```
public abstract bool IsEnableSpatializer()
```

Returned Value	Description
true	Enabled
false	Disabled

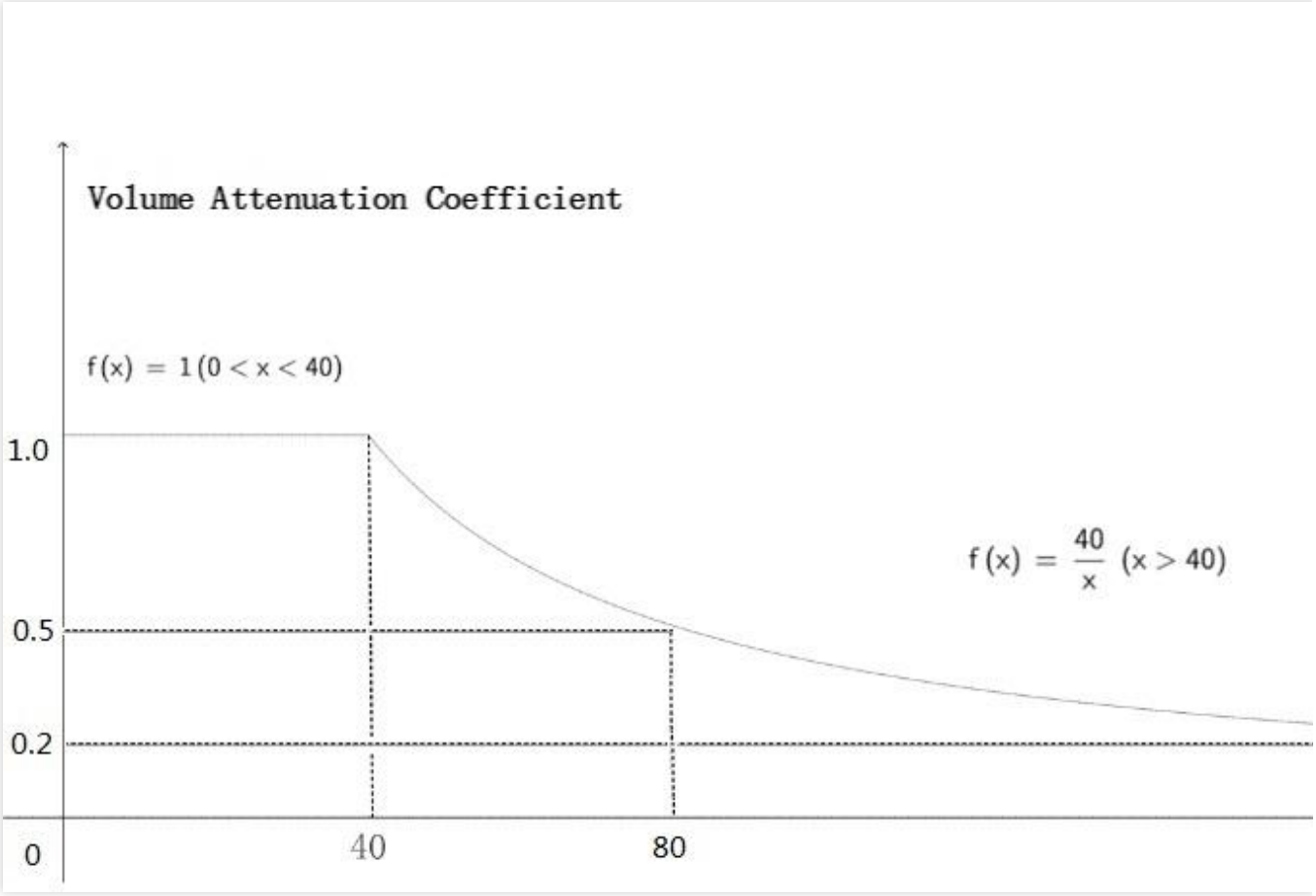
Setting the 3D sound effect attenuation range

The 3D sound effect attenuation range needs to be set. We recommend you set it to `100` .

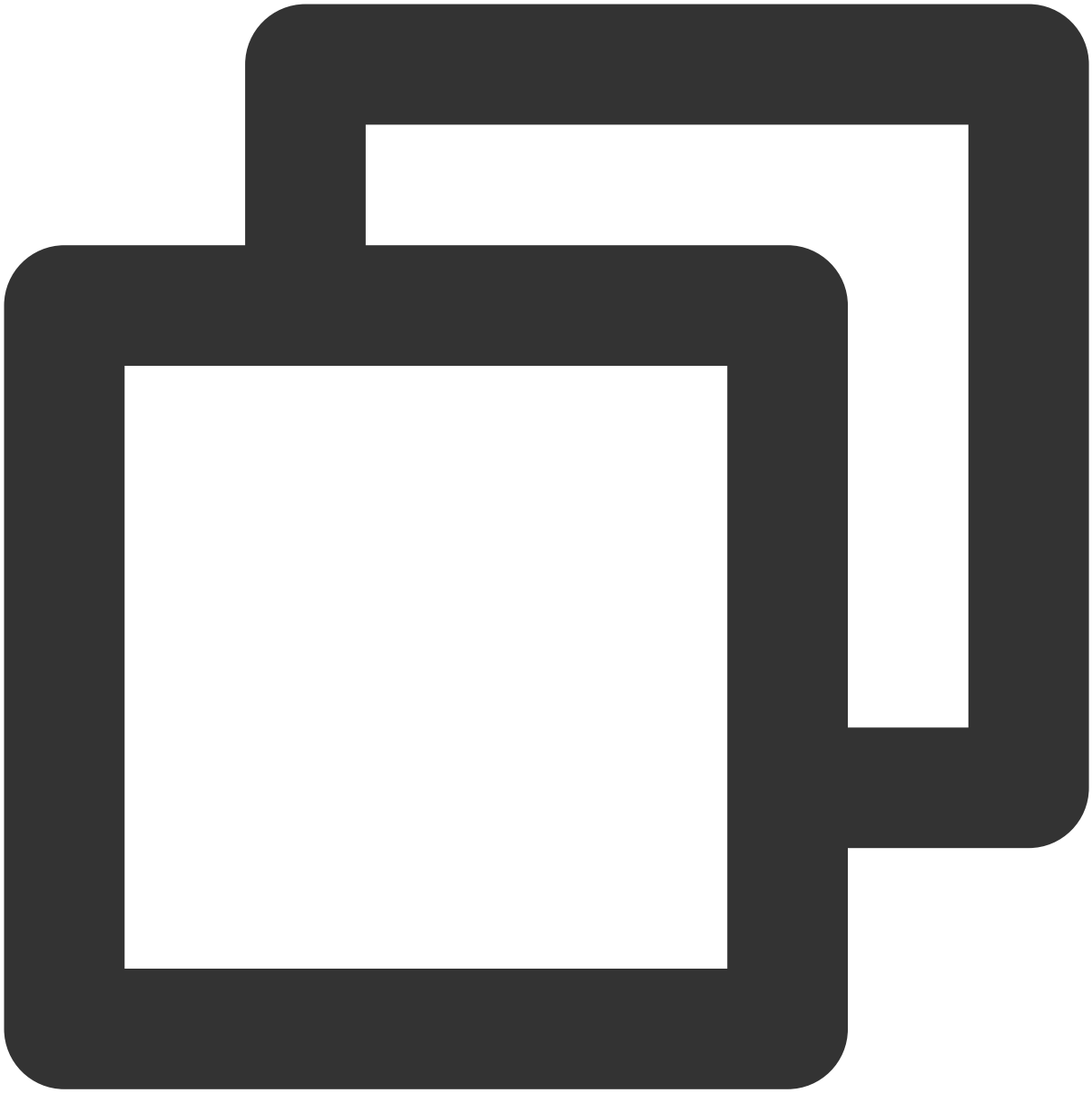
Relationship between distance and sound attenuation

In the 3D sound effect, the sound will begin to attenuate to almost zero as the distance to the sound source exceeds a specified threshold (range/10).

Distance (Unit in the Engine)	Attenuation Coefficient
0 < N < range/10	1.0 (no attenuation)
N ≥ range/10	Range/10/N



Function prototype



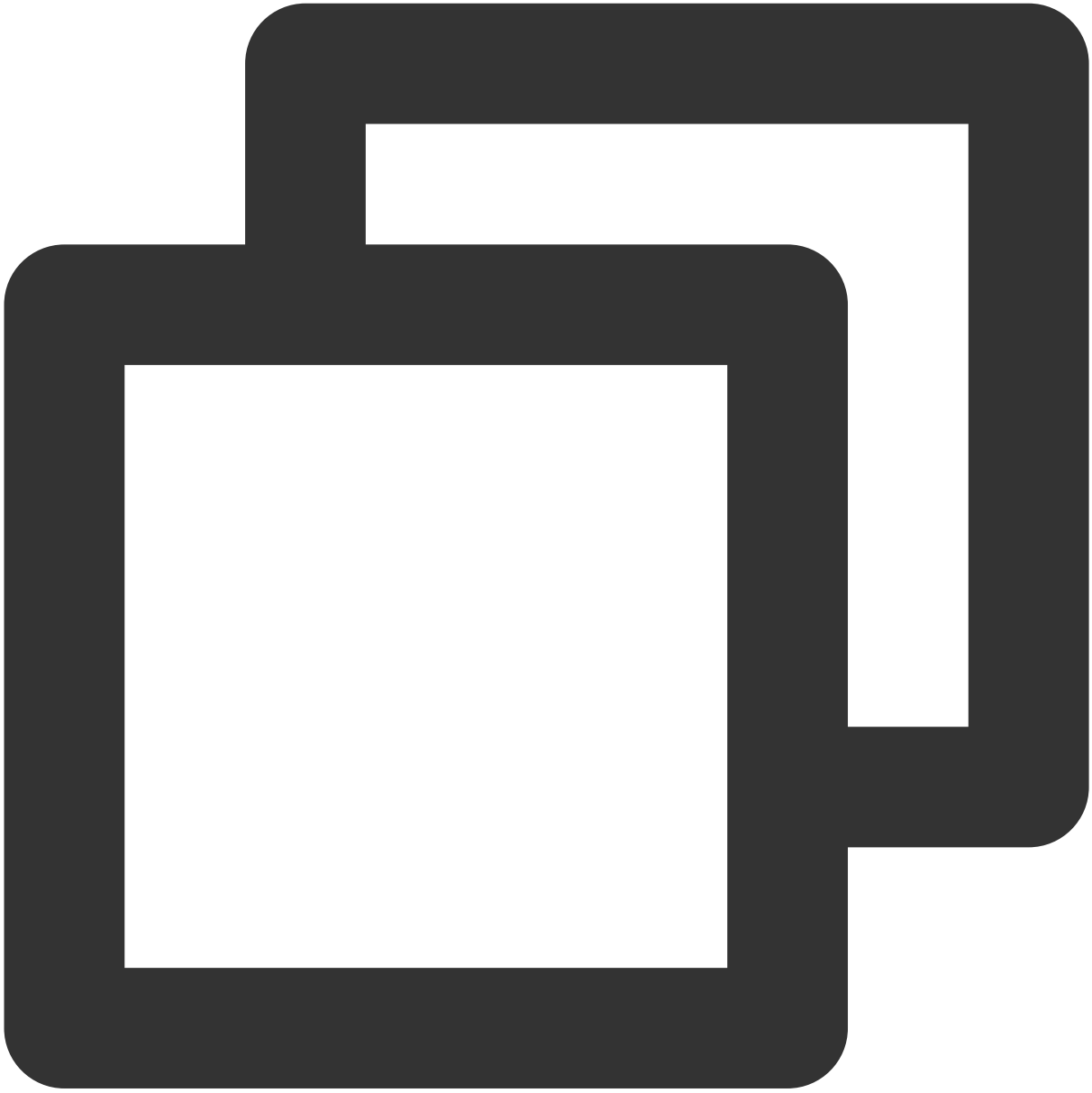
```
public abstract void UpdateAudioRecvRange(int range)
```

Parameter	Type	Description
range	int	The range within which the sound effect can be heard in distance unit used by the game engine. We recommend that you set it to `100` .

Updating the sound source position (including orientation)

This function is used to update the position information of the sound source. It can be called every frame to implement the 3D sound effect.

Function prototype



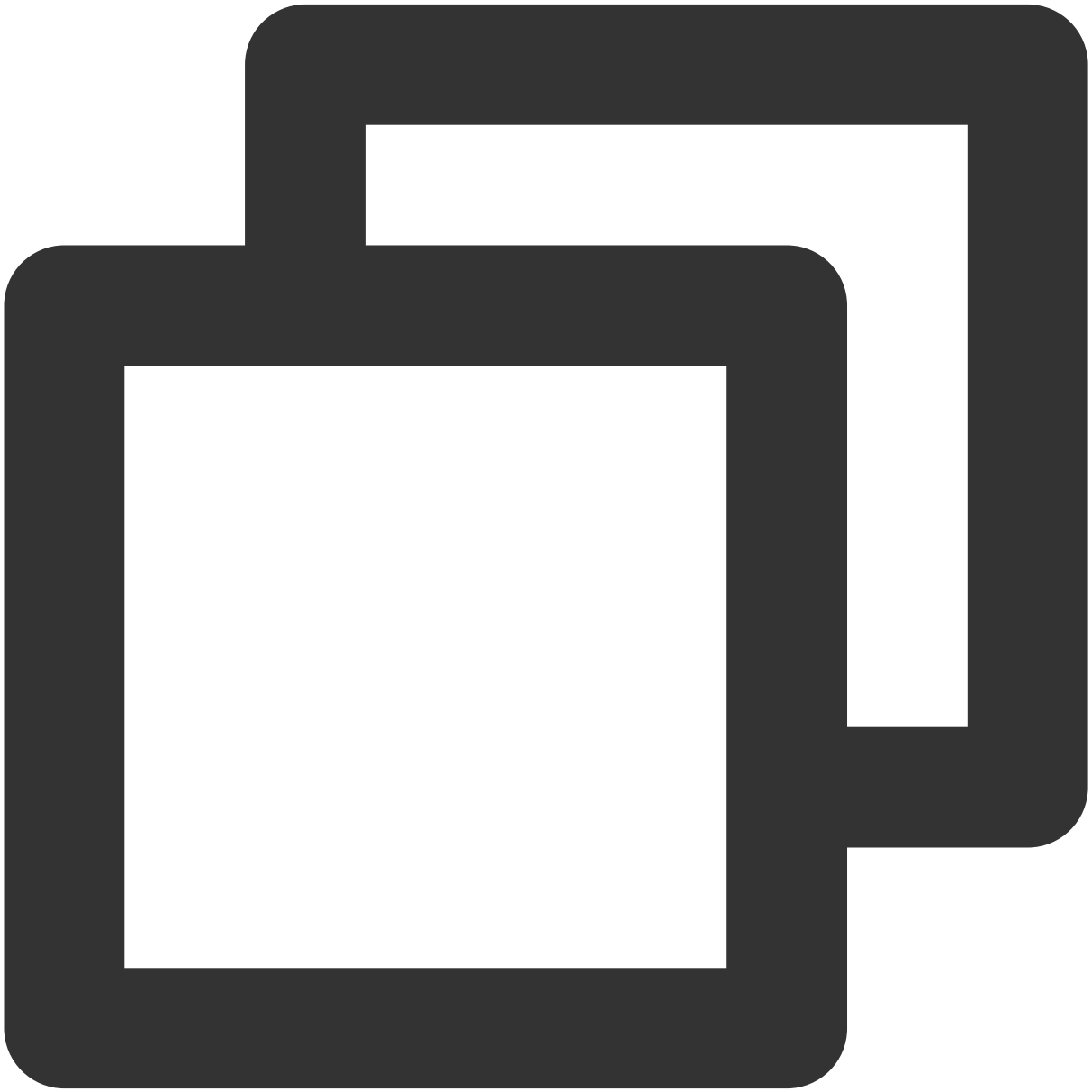
```
public abstract int UpdateSelfPosition(int position[3], float axisForward[3], float
```

Parameter	Type	Description
position	int[]	Local position (forward, right, and up) in the world coordinate system

axisForward	float[]	Forward vector in the local coordinate system
axisRight	float[]	Right vector in the local coordinate system
axisUp	float[]	Up vector in the local coordinate system

Sample code

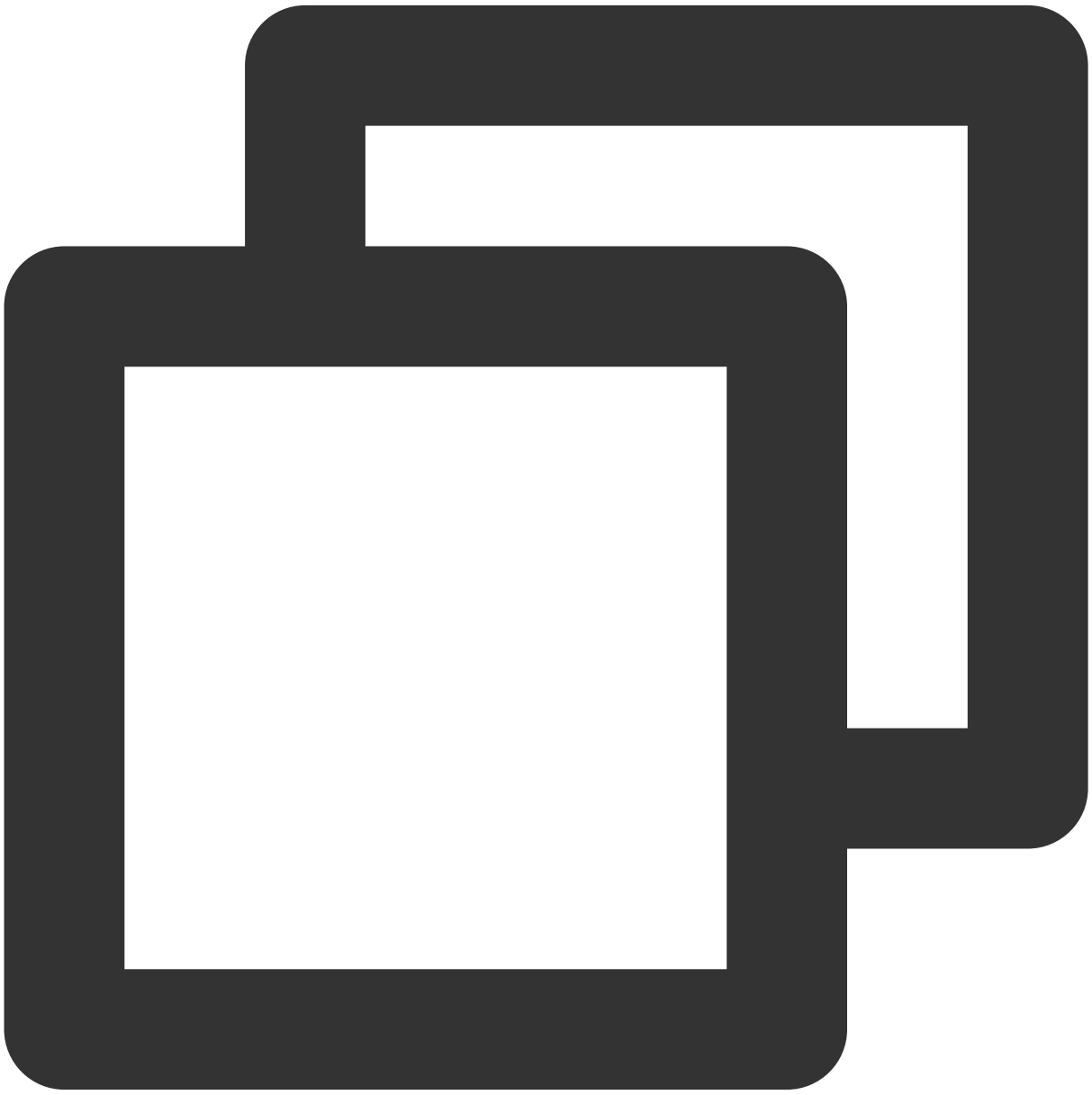
Unreal



```
FVector cameraLocation = UGameplayStatics::GetPlayerCameraManager(GetWorld(), 0)->G
FRotator cameraRotation = UGameplayStatics::GetPlayerCameraManager(GetWorld(), 0)->
```

```
int position[] = { (int)cameraLocation.X,(int)cameraLocation.Y, (int)cameraLocation
FMatrix matrix = ((FRotationMatrix)cameraRotation);
float forward[] = { matrix.GetColumn(0).X,matrix.GetColumn(1).X,matrix.GetColumn(2)
float right[] = { matrix.GetColumn(0).Y,matrix.GetColumn(1).Y,matrix.GetColumn(2).Y
float up[] = { matrix.GetColumn(0).Z,matrix.GetColumn(1).Z,matrix.GetColumn(2).Z};
ITMGContextGetInstance()->GetRoom()->UpdateSelfPosition(position, forward, right, u
```

Unity



```
Transform selftrans = currentPlayer.gameObject.transform;
Matrix4x4 matrix = Matrix4x4.TRS(Vector3.zero, selftrans.rotation, Vector3.one);
int[] position = new int[3] { selftrans.position.z, selftrans.position.x, selftrans
```

```
float[] axisForward = new float[3] { matrix.m22, matrix.m02, matrix.m12 };
float[] axisRight = new float[3] { matrix.m20, matrix.m00, matrix.m10 };
float[] axisUp = new float[3] { matrix.m21, matrix.m01, matrix.m11 };
ITMGContext.GetInstance().GetRoom().UpdateSelfPosition(position, axisForward, axisR
```

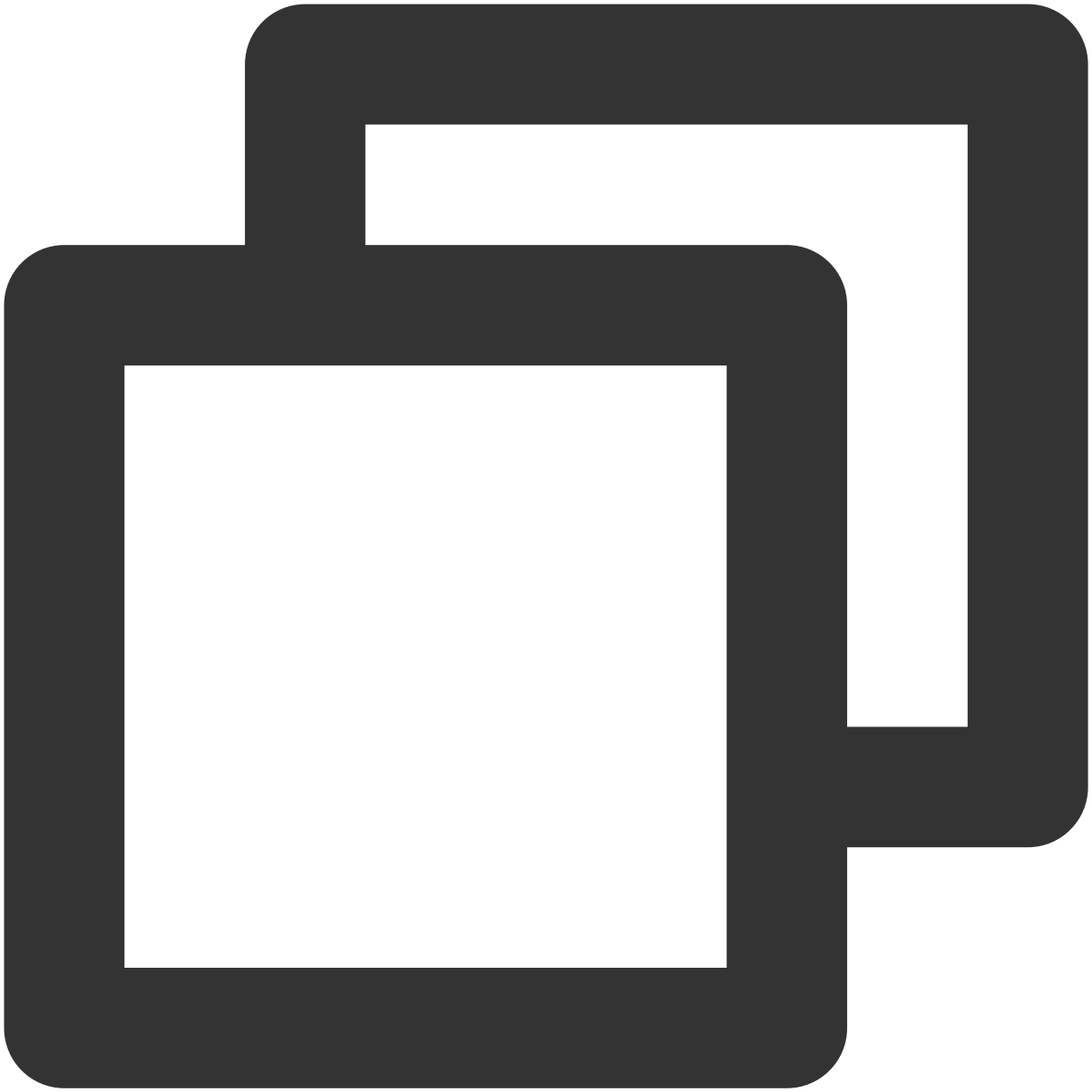
Local position API (for VR scenarios)

This API is suitable for scenarios with high requirements for 3D position changes on VR devices. This feature is an advanced API and can be used only on GME 2.9.2 or later.

In general 3D scenarios, the user only needs to use the `UpdateSelfPosition` function to update the position information and send it to other users over the network. `UpdateOtherPosition` can also pass in the position information of other players locally without using the network, making it suitable for VR game scenarios.

To avoid conflicting with the remotely updated coordinates, if you call `UpdateOtherPosition`, remote coordinates will be discarded, which will affect the next room entry. Therefore, **to update the player coordinates locally, update the coordinates of all players**.

Function prototype



```
public abstract int UpdateOtherPosition(int position[3])
```

Parameter	Type	Description
position	int[]	Position of another player (forward, right, and up) in the world coordinate system

Note

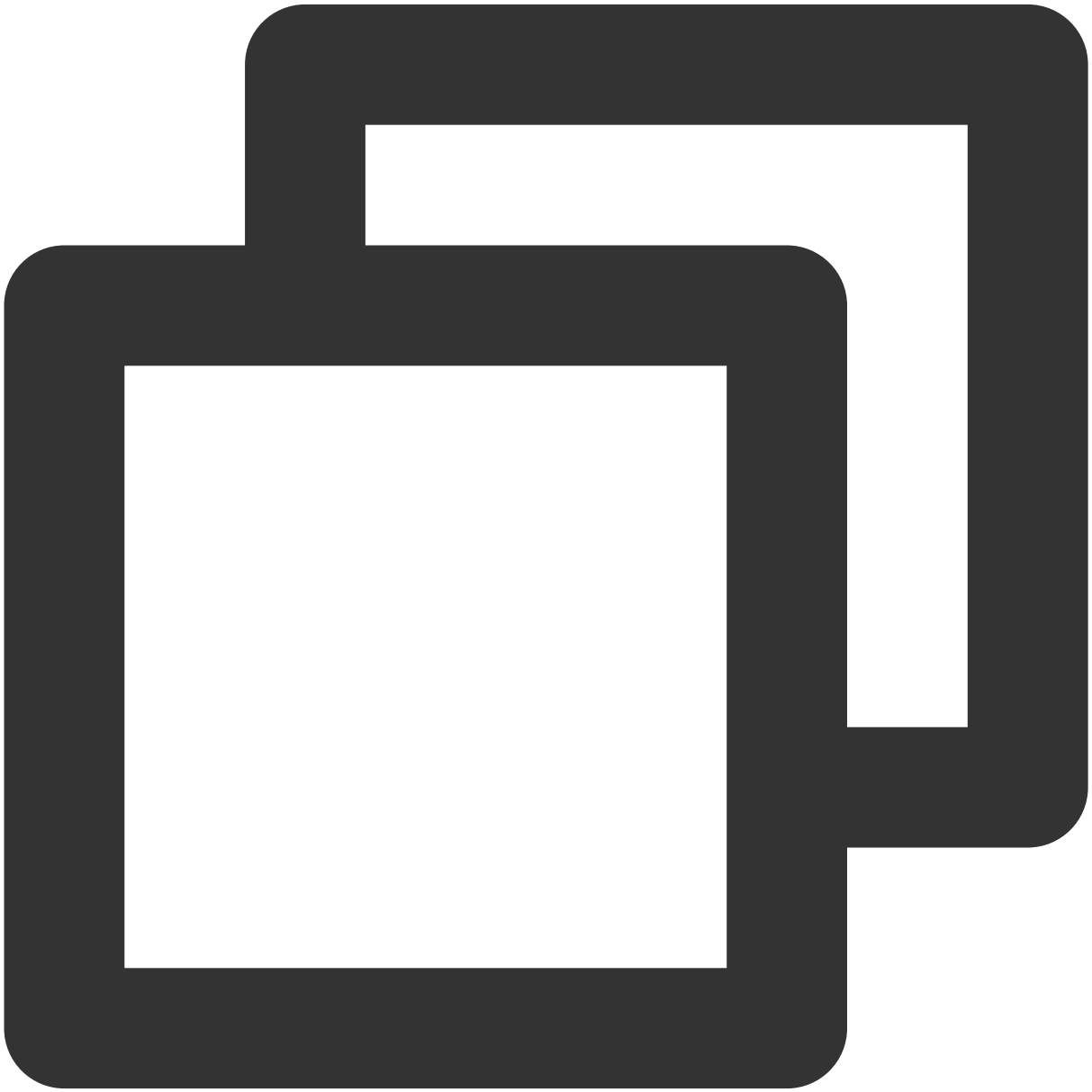
To update player coordinates locally, traverse **all player coordinates** and pass them in through this API.

3D sound effect blocklist API

Note

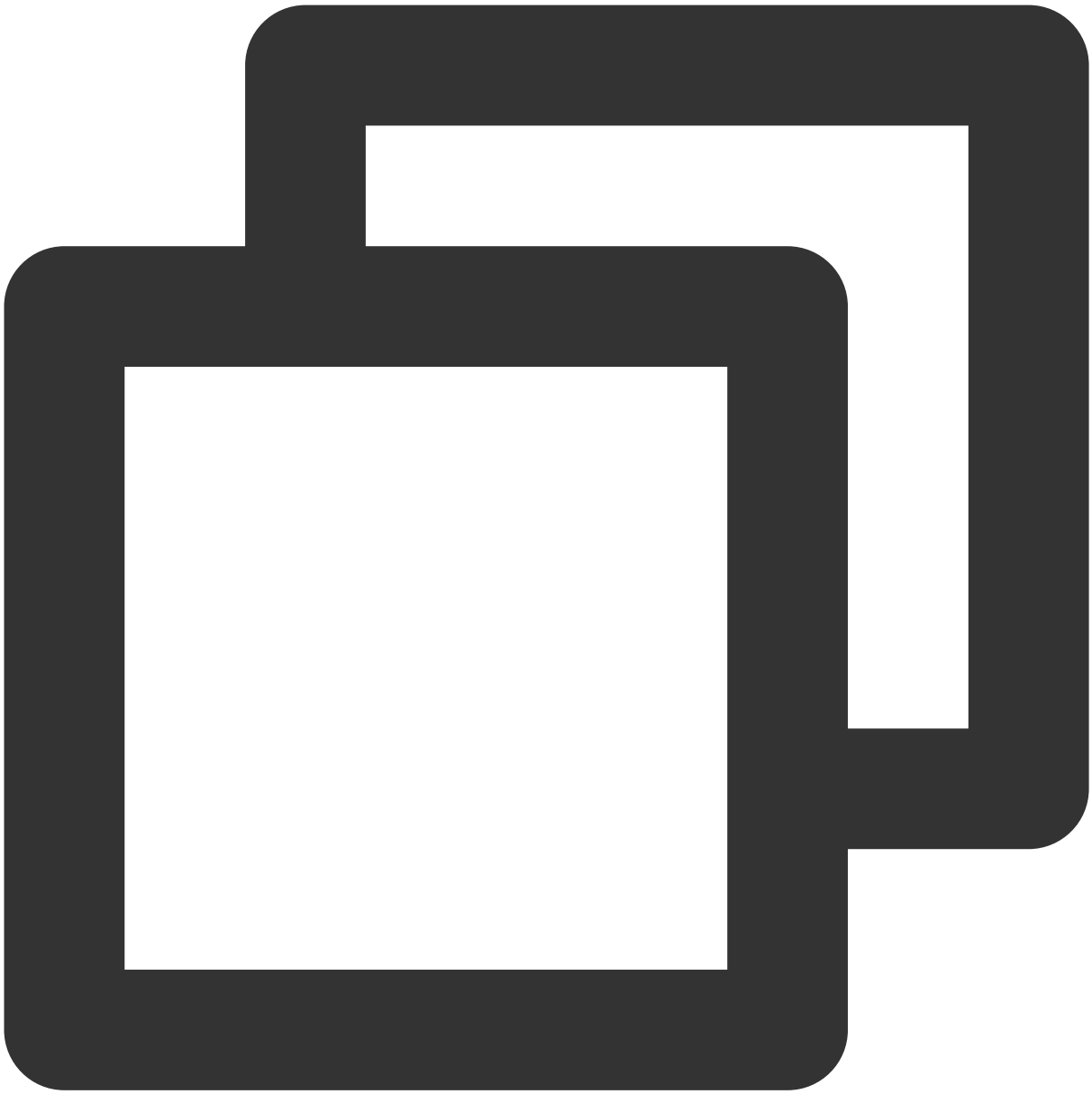
This API takes effect on GME SDK 2.9.3 or later.

Currently, the 3D sound effect feature will take effect for all users in the room after being called. However, in some special scenarios, you may not want to see that someone's voice attenuates because of the 3D sound effect. In this case, you can call this API to add the user to the 3D sound effect blocklist, and voice of the specified `openid` will no longer have the 3D sound effect.



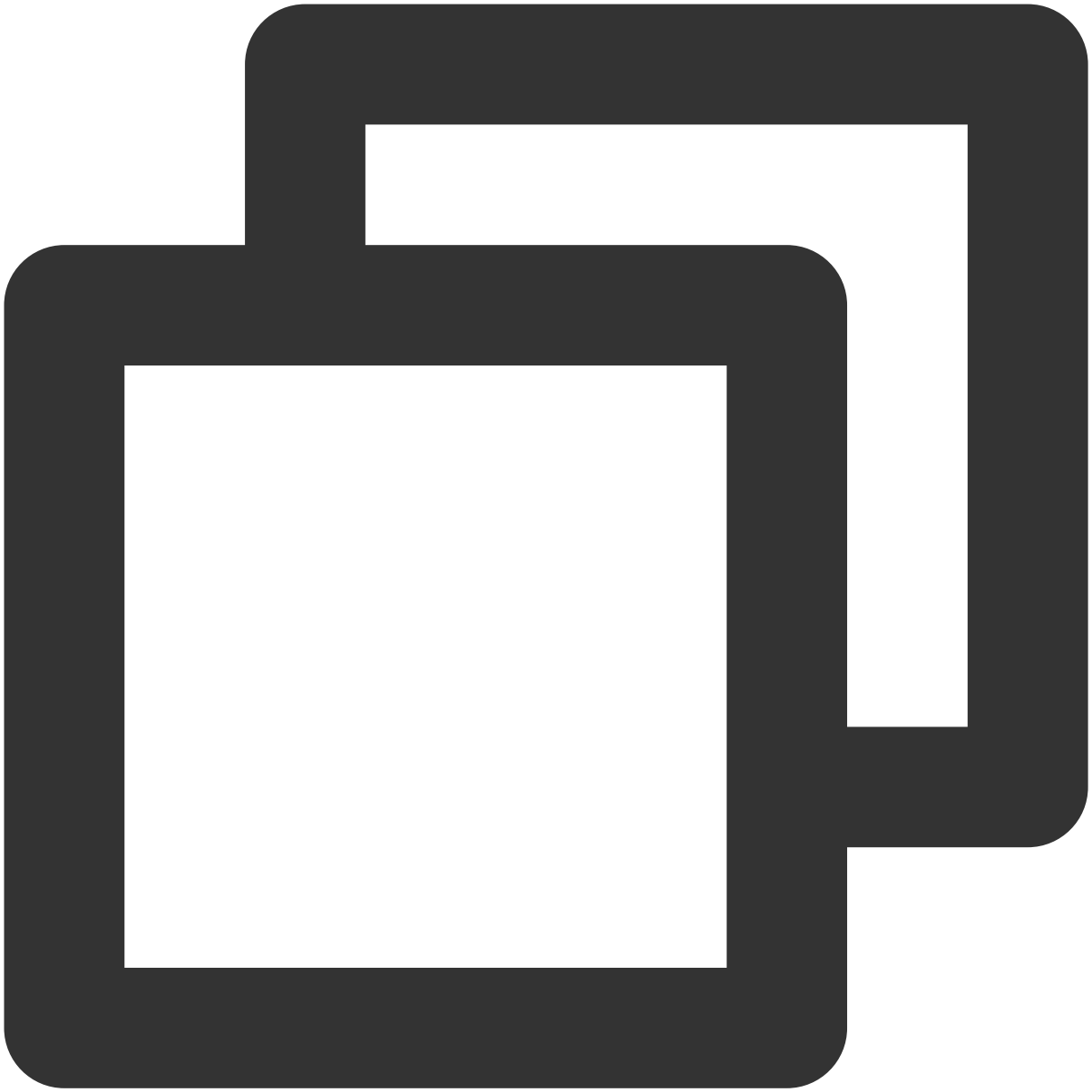
```
virtual int AddSpatializerBlacklist(const char* openId);
```

To remove the `openid` from the blocklist, you need to call the following API:



```
virtual int RemoveSpatializerBlacklist(const char* openId);
```

To clear the blocklist, you need to call the following API:



```
virtual int ClearSpatializerBlacklist();
```

Troubleshooting

If the voice has no 3D sound effects after this feature is connected, you can troubleshoot as follows:

1. Check whether users have successfully entered the room, turned on the mic, and can hear each other.
2. Check whether a stereo headset is used.

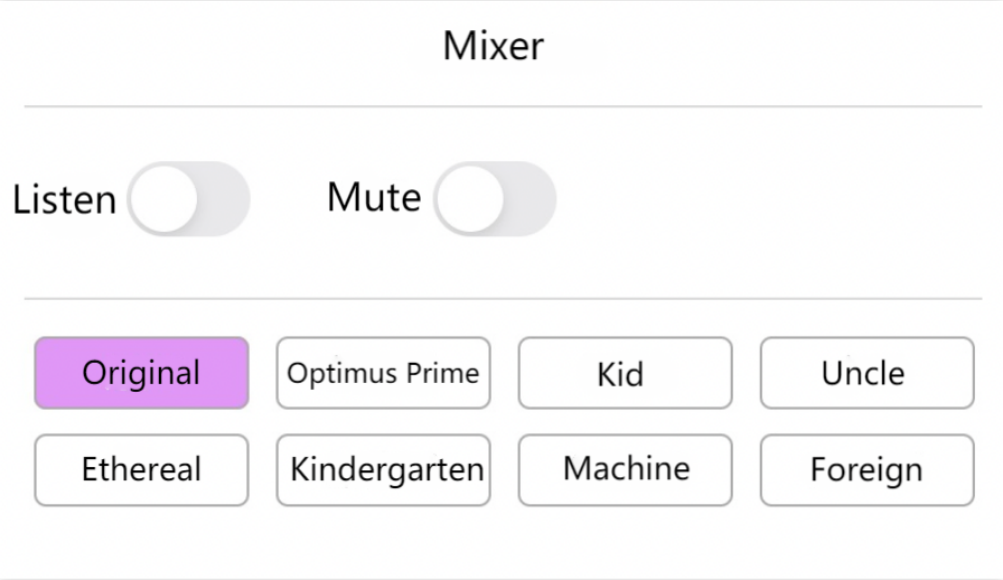
3. Check whether the returned value of `InitSpatializer` is `0` .
4. Check whether the value of `UpdateAudioRecvRange` is too small.
5. Check whether the `UpdateSelfPosition` API is called periodically.
6. Troubleshoot the problem by referring to Error Codes.

Sound Effect and Accompaniment Voice Changing

Last updated：2024-01-18 14:22:54

This document describes how to integrate with and debug GME APIs for voice changing effects.

Overview



Prerequisites

You have activated the voice chat service. For more information, see Activating Services.
You have activated the speech-to-text service. For more information, see Activating Services.
You have integrated the GME SDK, including core APIs and voice chat APIs. For more information, see Quick Integration of Native SDK, Quick Integration of SDK for Unity, and Quick Integration of SDK for Unreal Engine.
You have integrated the `libgmesoundtouch` library file of the GME SDK. Ensure that the project's library files include `libgmesoundtouch`. For more information, see SDK Version Upgrade Guide.

Integrating Real-Time Voice Changing

Voice changing API

After successful room entry and mic enablement, call the `SetVoiceType` API to set the voice changing effect. If the API returns 0, the call is successful, and the local sound heard by users in the room has the voice changing effect. To test the effect, use the in-ear monitoring feature (API: EnableLoopBack).

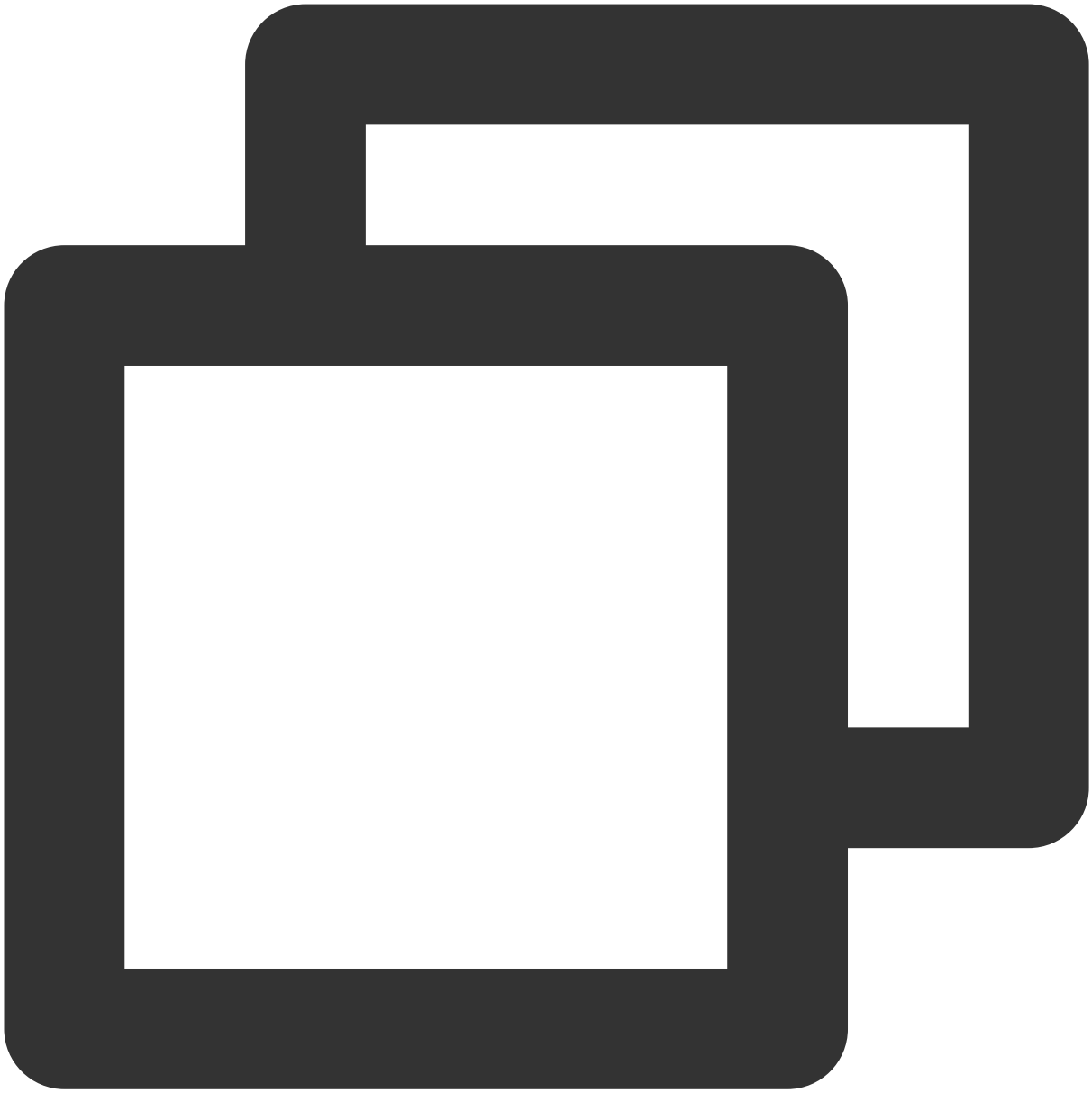
Function prototype

Android

iOS

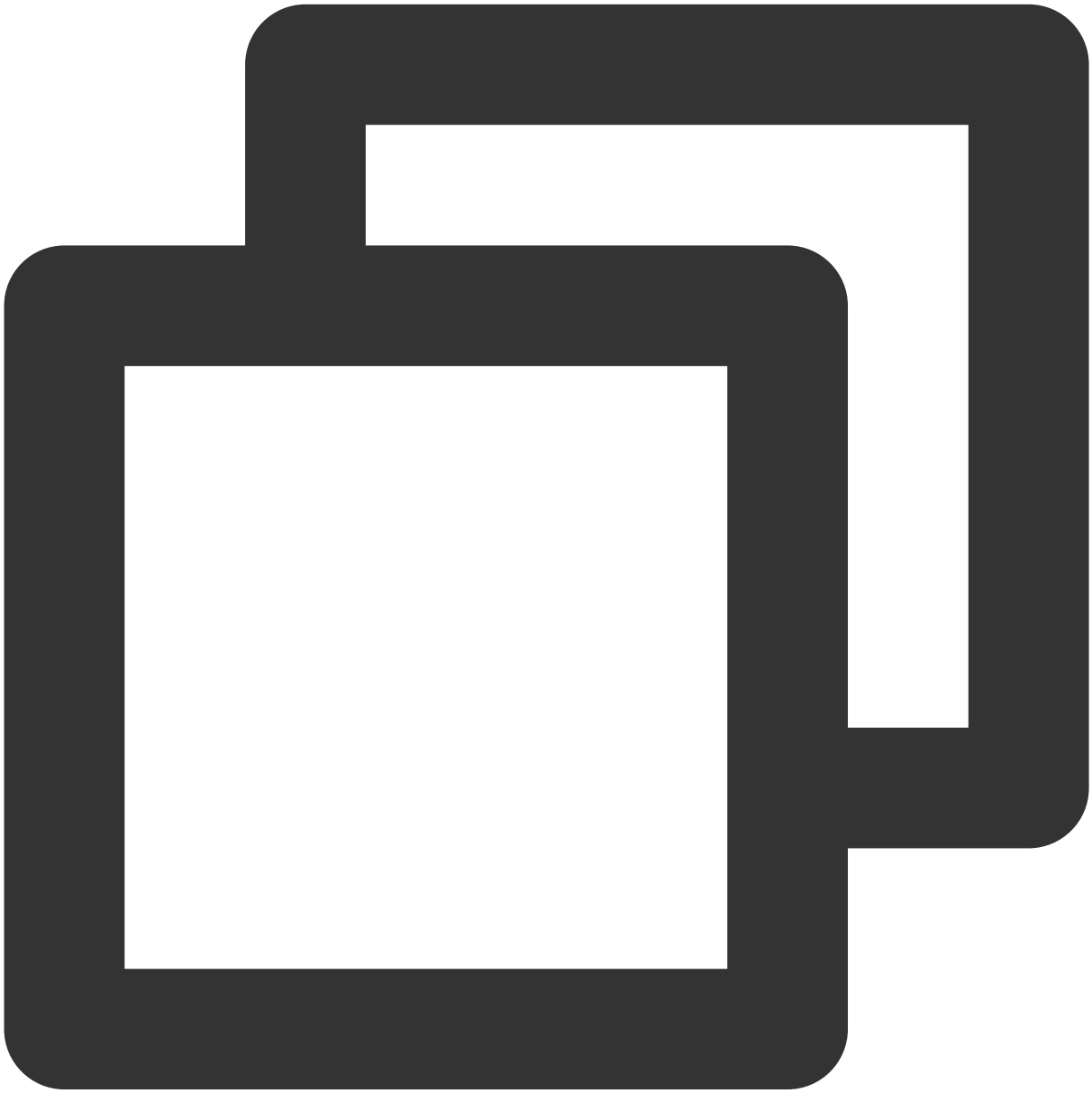
Unity

C++

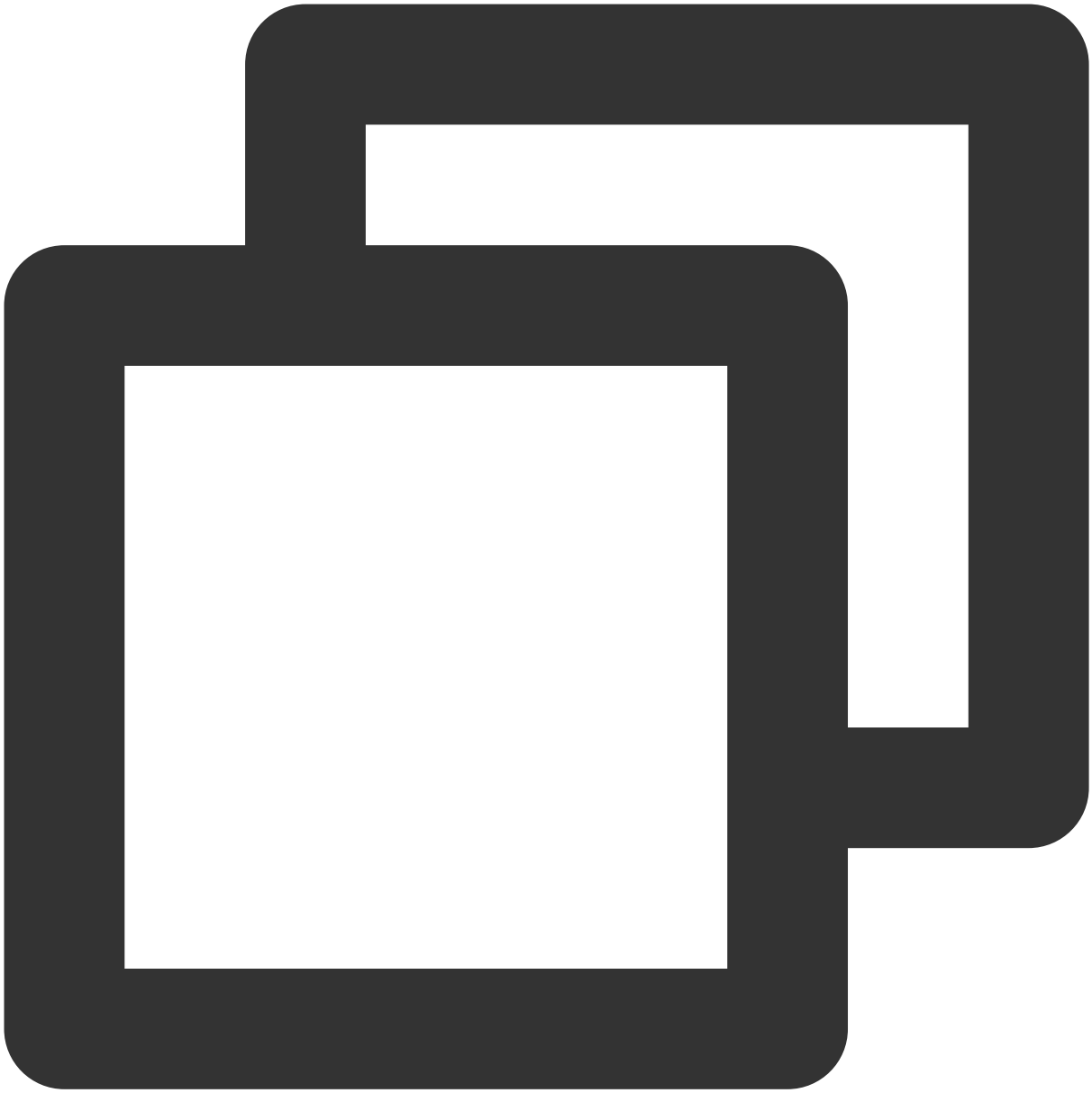


```
public static class ITMG_VoiceType {
    public static final int ITMG_VOICE_TYPE_ORIGINAL_SOUND = 0;
    public static final int ITMG_VOICE_TYPE_LOLITA = 1;
    public static final int ITMG_VOICE_TYPE_UNCLE = 2;
    public static final int ITMG_VOICE_TYPE_INTANGIBLE = 3;
    public static final int ITMG_VOICE_TYPE_DEAD_FATBOY = 4;
    public static final int ITMG_VOICE_TYPE_HEAVY_MENTAL = 5;
    public static final int ITMG_VOICE_TYPE_DIALECT = 6;
    public static final int ITMG_VOICE_TYPE_INFLUENZA = 7;
    public static final int ITMG_VOICE_TYPE_CAGED_ANIMAL = 8;
    public static final int ITMG_VOICE_TYPE_HEAVY_MACHINE = 9;
```

```
        public static final int ITMG_VOICE_TYPE_STRONG_CURRENT = 10;
        public static final int ITMG_VOICE_TYPE_KINDER_GARTEN = 11;
        public static final int ITMG_VOICE_TYPE_HUANG = 12;
    };
    public abstract int SetVoiceType(int type);
```

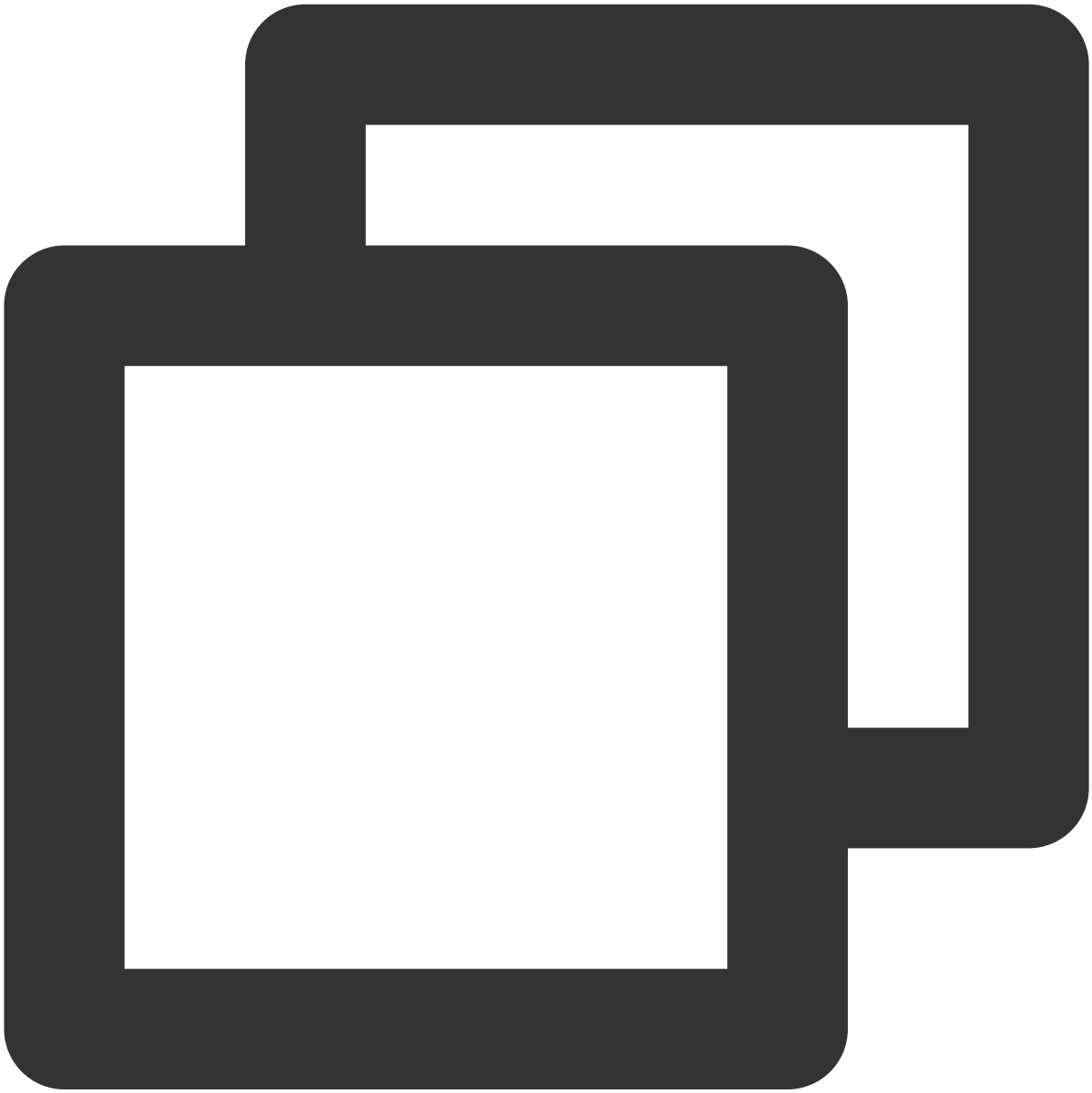


```
-(QAVResult)SetVoiceType:(ITMG_VOICE_TYPE) type
```



```
public abstract class ITMGAudioEffectCtrl{
    public static int VOICE_TYPE_ORIGINAL_SOUND = 0;
    public static int VOICE_TYPE_LOLITA = 1;
    public static int VOICE_TYPE_UNCLE = 2;
    public static int VOICE_TYPE_INTANGIBLE = 3;
    public static int VOICE_TYPE_DEAD_FATBOY = 4;
    public static int VOICE_TYPE_HEAVY_MENTAL = 5;
    public static int VOICE_TYPE_DIALECT = 6;
    public static int VOICE_TYPE_INFLUENZA = 7;
    public static int VOICE_TYPE_CAGED_ANIMAL = 8;
    public static int VOICE_TYPE_HEAVY_MACHINE = 9;
```

```
    public static int VOICE_TYPE_STRONG_CURRENT = 10;
    public static int VOICE_TYPE_KINDER_GARTEN = 11;
    public static int VOICE_TYPE_HUANG = 12;
    public abstract int SetVoiceType(int voiceType);
}
```



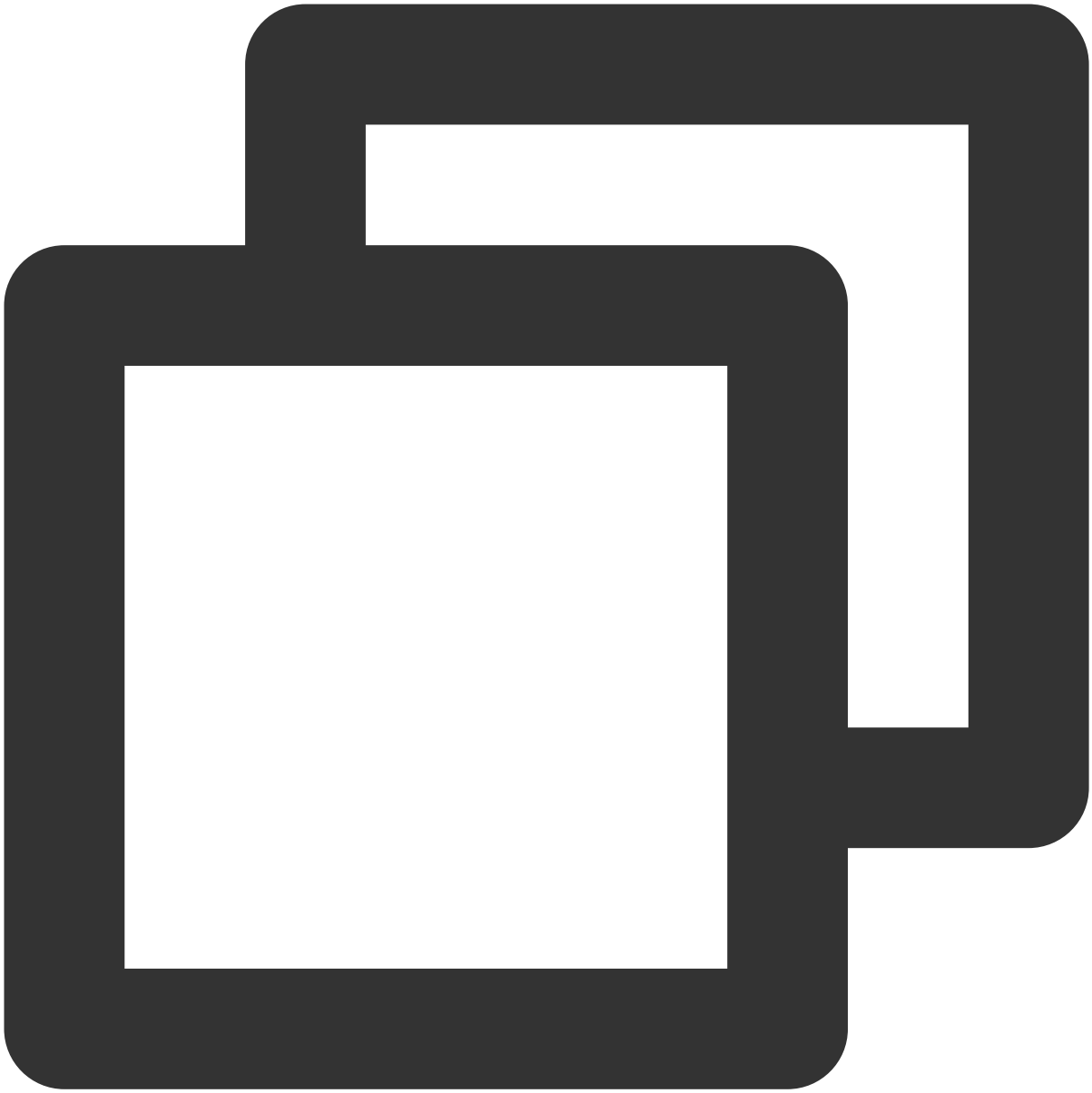
```
class ITMGAudioEffectCtrl {
public:
    virtual ~ITMGAudioEffectCtrl(){};
    virtual int SetVoiceType(ITMG_VOICE_TYPE voiceType) = 0;
```

```
}
```

Parameter	Type	Description
type	int	Indicates the type of local voice changing effect.

Type parameter	Value	Description
ITMG_VOICE_TYPE_ORIGINAL_SOUND	0	Original
ITMG_VOICE_TYPE_LOLITA	1	Lolita
ITMG_VOICE_TYPE_UNCLE	2	Uncle
ITMG_VOICE_TYPE_INTANGIBLE	3	Ethereal
ITMG_VOICE_TYPE_DEAD_FATBOY	4	Fatty
ITMG_VOICE_TYPE_HEAVY_MENTA	5	Heavy metal
ITMG_VOICE_TYPE_DIALECT	6	Foreign
ITMG_VOICE_TYPE_INFLUENZA	7	Catching cold
ITMG_VOICE_TYPE_CAGED_ANIMAL	8	Animal
ITMG_VOICE_TYPE_HEAVY_MACHINE	9	Machine
ITMG_VOICE_TYPE_STRONG_CURRENT	10	Strong current
ITMG_VOICE_TYPE_KINDER_GARTEN	11	Kid
ITMG_VOICE_TYPE_HUANG	12	Urchin

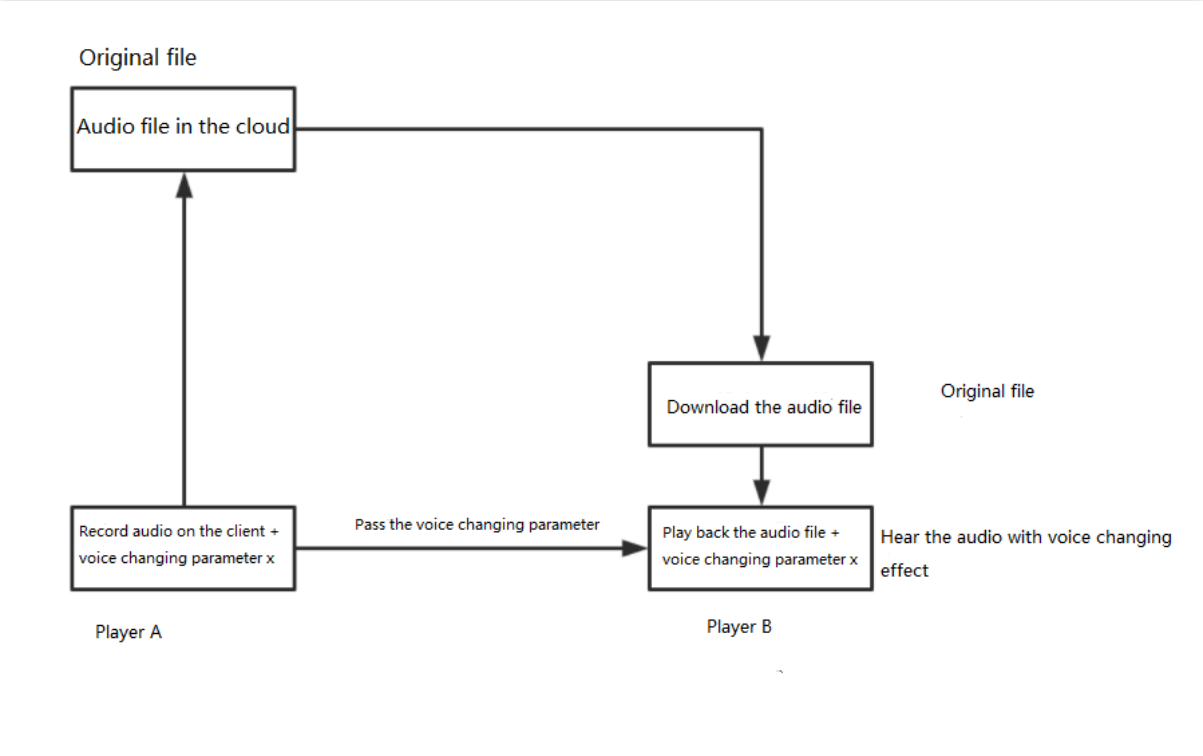
Sample code



```
ITMGContextGetInstance()->GetAudioEffectCtrl()->setVoiceType(0);
```

Integrating Voice Changing for Voice Message

Process



Voice changing doesn't affect the original voice message, as the voice changing effect will be reflected only during playback.

Voice message playback

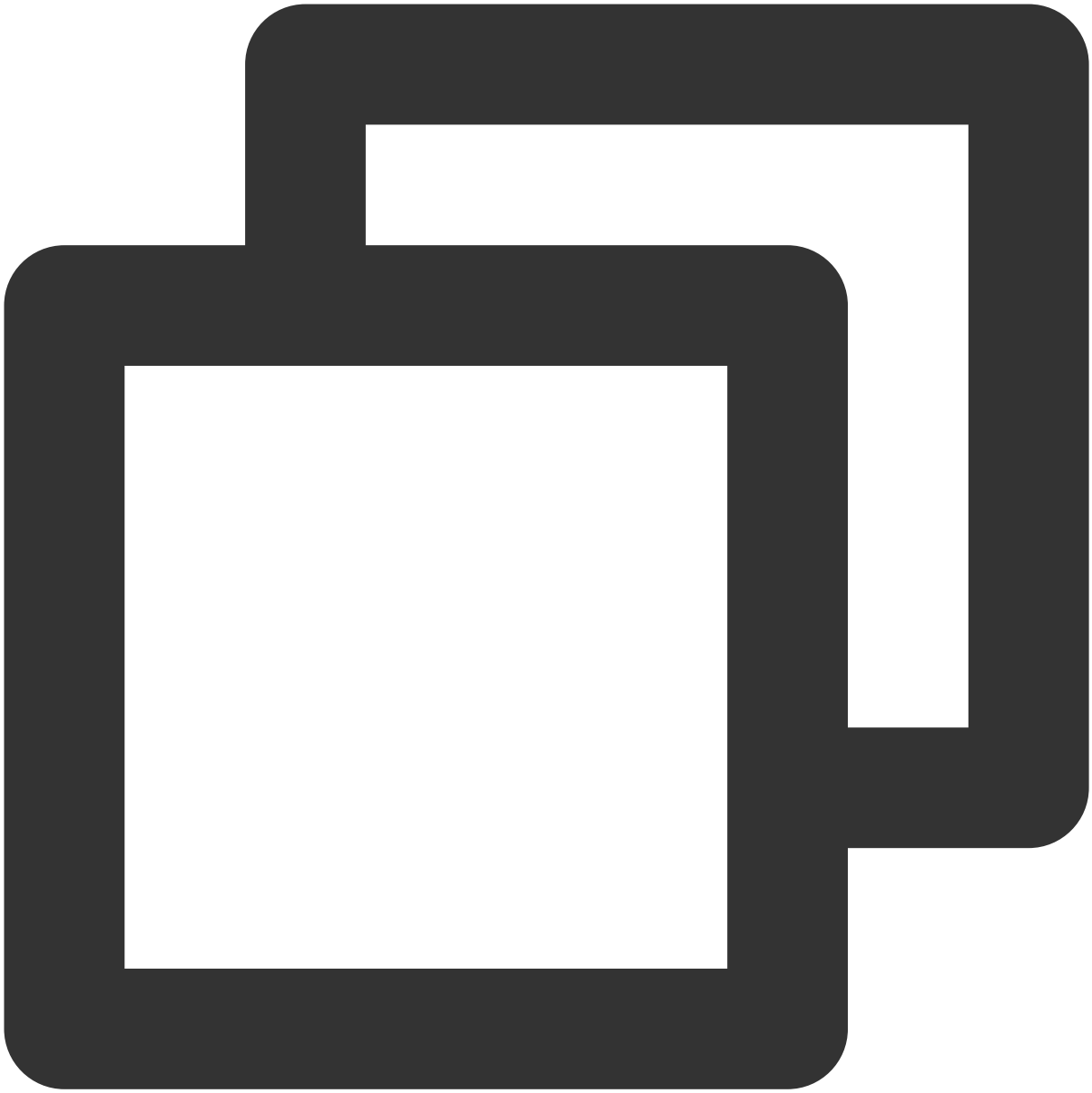
Add voice changing parameters when calling the voice message playback API.

Android

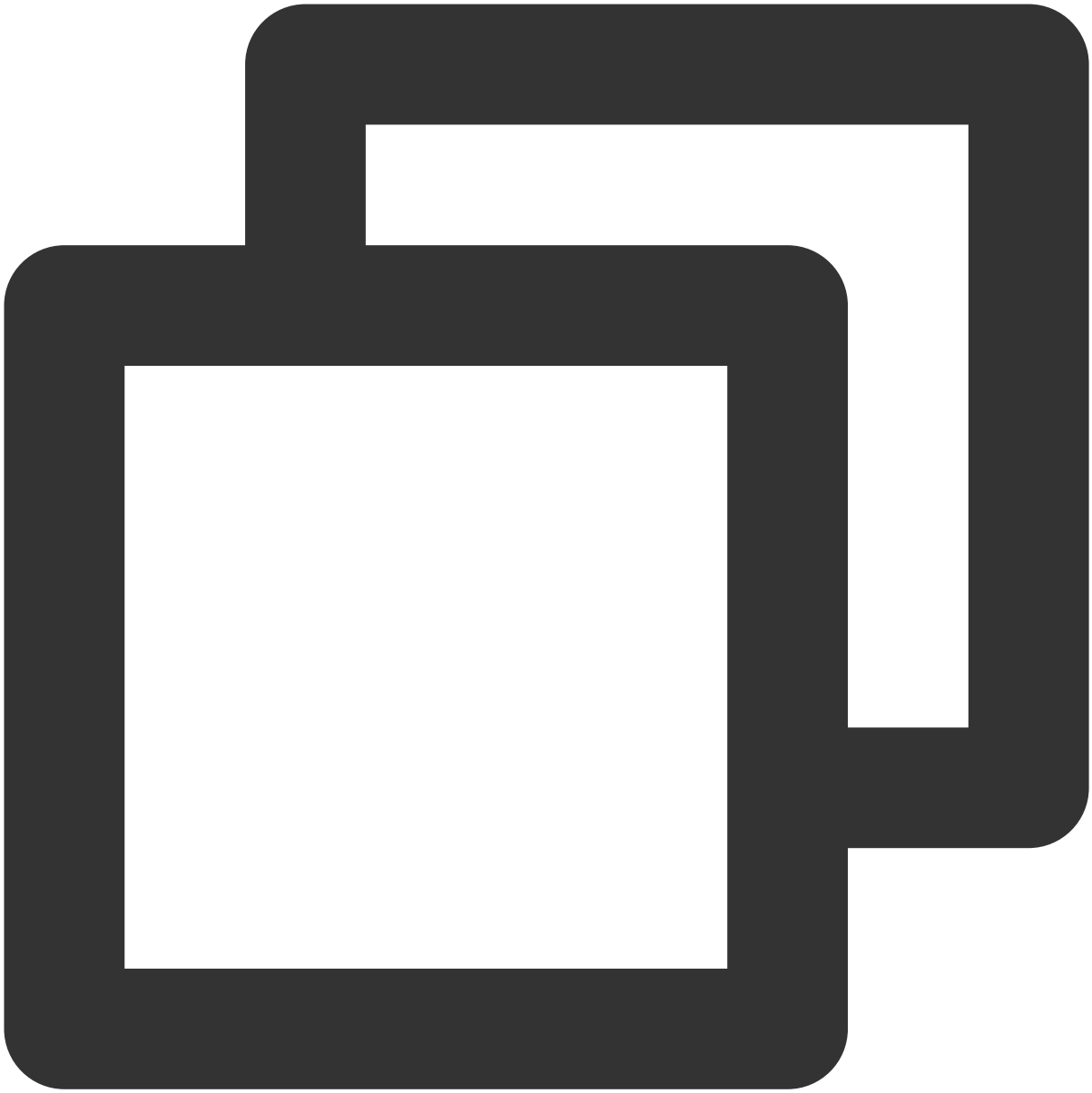
iOS

Unity

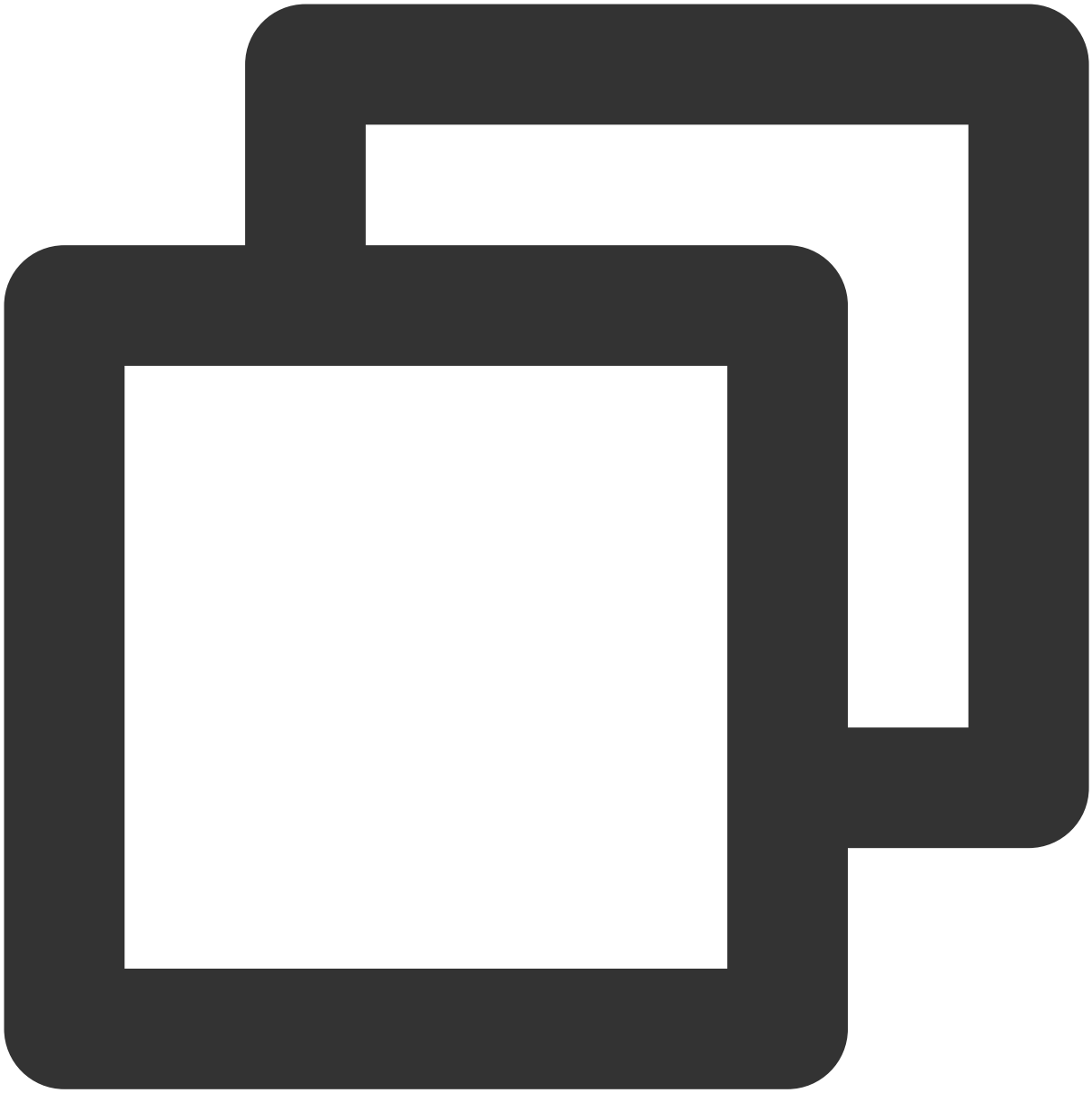
C++



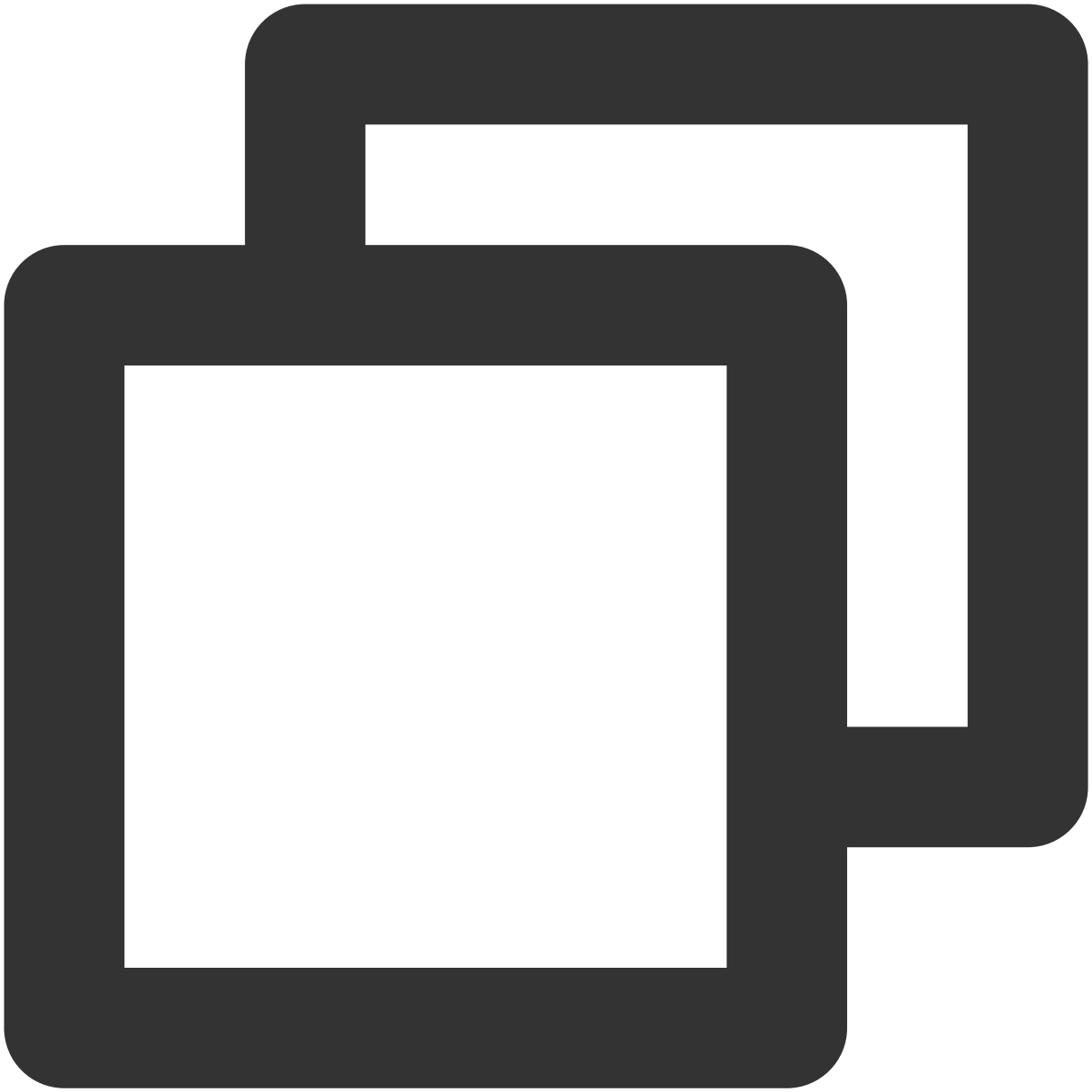
```
public abstract int PlayRecordedFile(String filePath,int voicetype);
```



```
-(int)PlayRecordedFile:(NSString*)filePath VoiceType:(ITMG_VOICE_TYPE) type
```



```
ITMGPTT PlayRecordedFile(string filePath,int voiceType);
```



```
public abstract int PlayRecordedFile(string filePath,int voiceType);
```

Parameter	Type	Description
filePath	string	Local audio file path
voicetype	int	Voice changer type

Error codes

Error Code Value	Cause	Suggested Solution
20485	Playback is not started.	Ensure the existence of the file and the validity of the file path.

Accompaniment in Voice Chat

Last updated：2024-01-18 14:25:17

This document describes the GME APIs for accompaniment in voice chat so that developers can easily integrate and debug them.

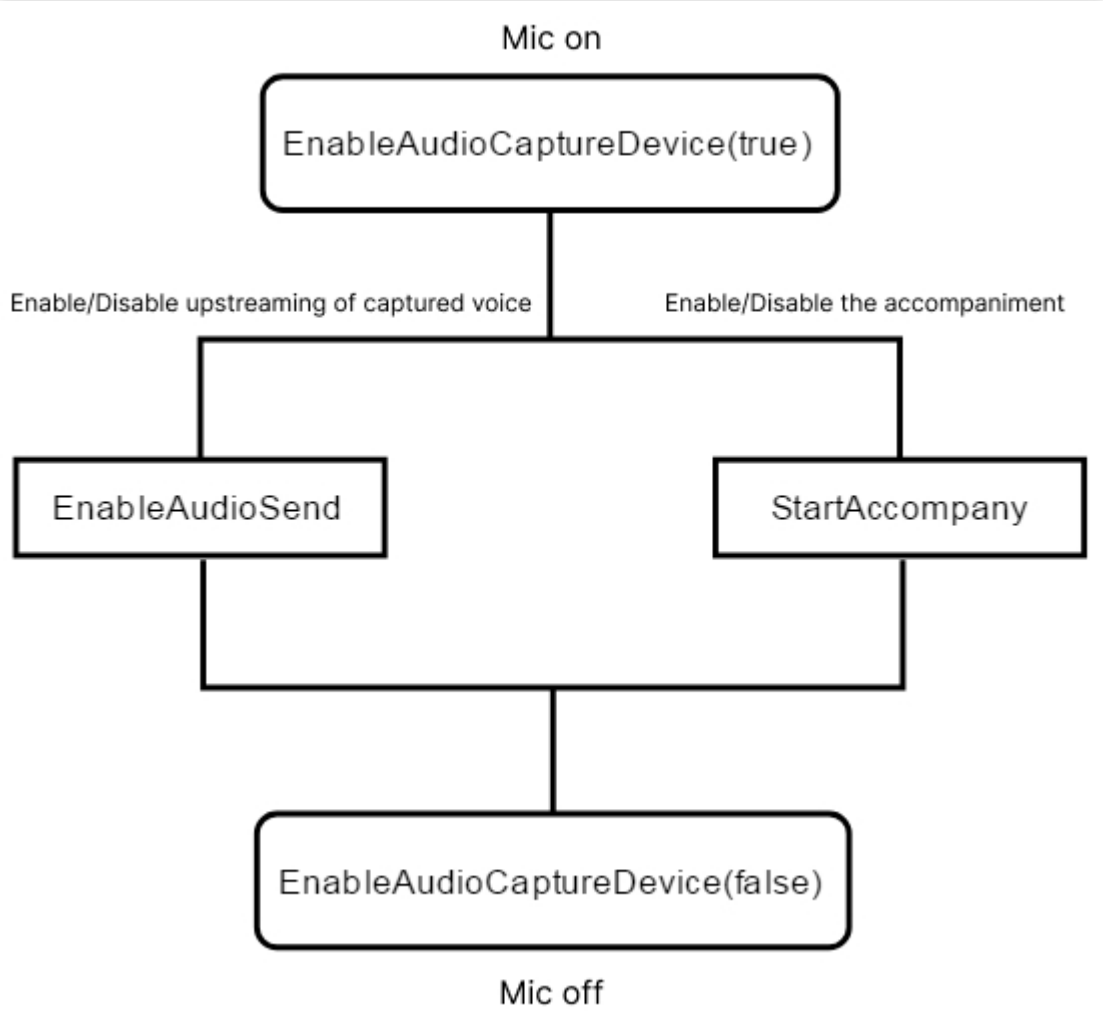
APIs for Accompaniment in Voice Chat

API	Description
StartAccompany	Starts playing back the accompaniment.
StopAccompany	Stops playing back the accompaniment.
IsAccompanyPlayEnd	Indicates whether the accompaniment is over.
PauseAccompany	Pauses playing back the accompaniment.
ResumeAccompany	Resumes playing back the accompaniment.
SetAccompanyVolume	Sets the accompaniment volume.
GetAccompanyVolume	Obtains the accompaniment volume.
SetAccompanyFileCurrentPlayedTimeByMs	Sets the playback progress.

Note:

To use accompaniment in voice chat, you must integrate the GME SDK and enable real-time voice chat.

Flowchart



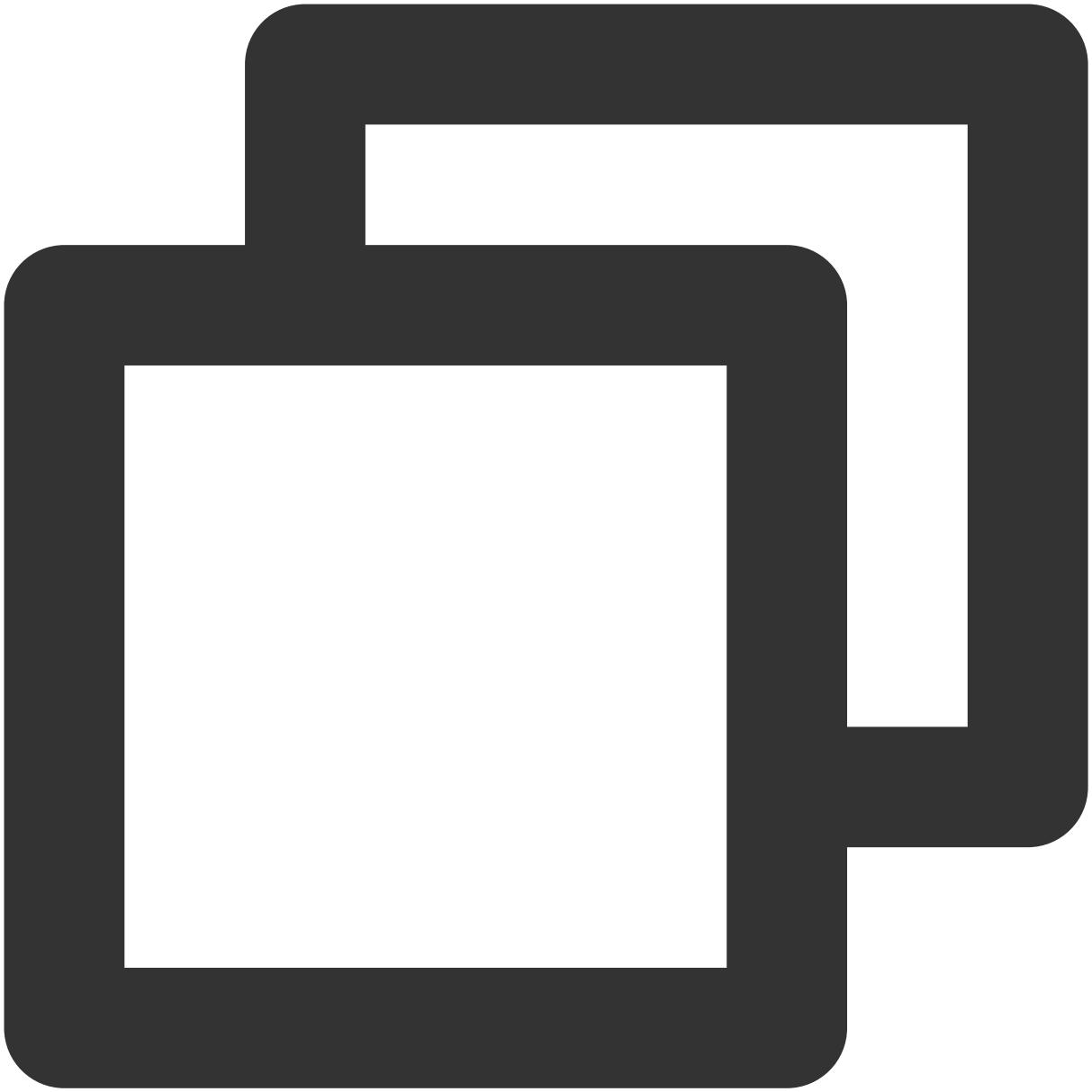
How to use with EnableAudioCaputreDevice

After you enter a voice chat room, call `EnableAudioCaputreDevice` to enable the capturing device, and call `StartAccompany` to play back the accompaniment. To capture human voices via the microphone, you should call `EnableAudioSend` to enable the microphone first.

Starting playing back the accompaniment

This API (StartAccompany) is used to start playing back the accompaniment in M4A, WAV, or MP3 format. Calling this API resets the volume.

Function prototype

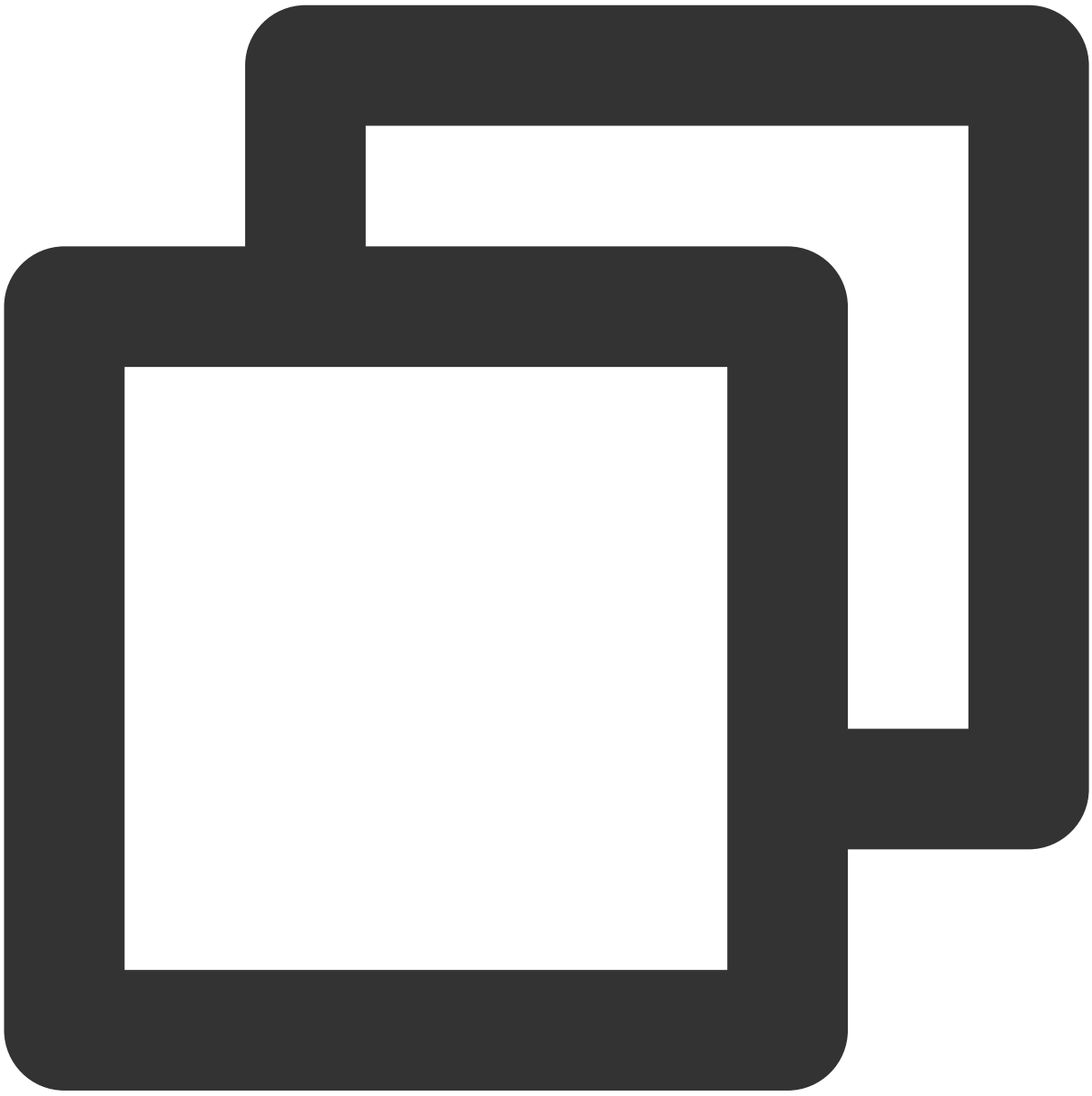


```
ITMGAudioEffectCtrl virtual int StartAccompany(const char* filePath, bool loopBack,
```

Parameter	Type	Description
filePath	char*	Path of the accompaniment file.
loopBack	bool	Indicates whether to output the accompaniment in a mixing mode. This is generally set to `true`, indicating that the audience can also hear the accompaniment.
loopCount	int	The number of loops. `-1` indicates an infinite loop, and `0` indicates not to play back.

msTime	int	Delay time

Sample code

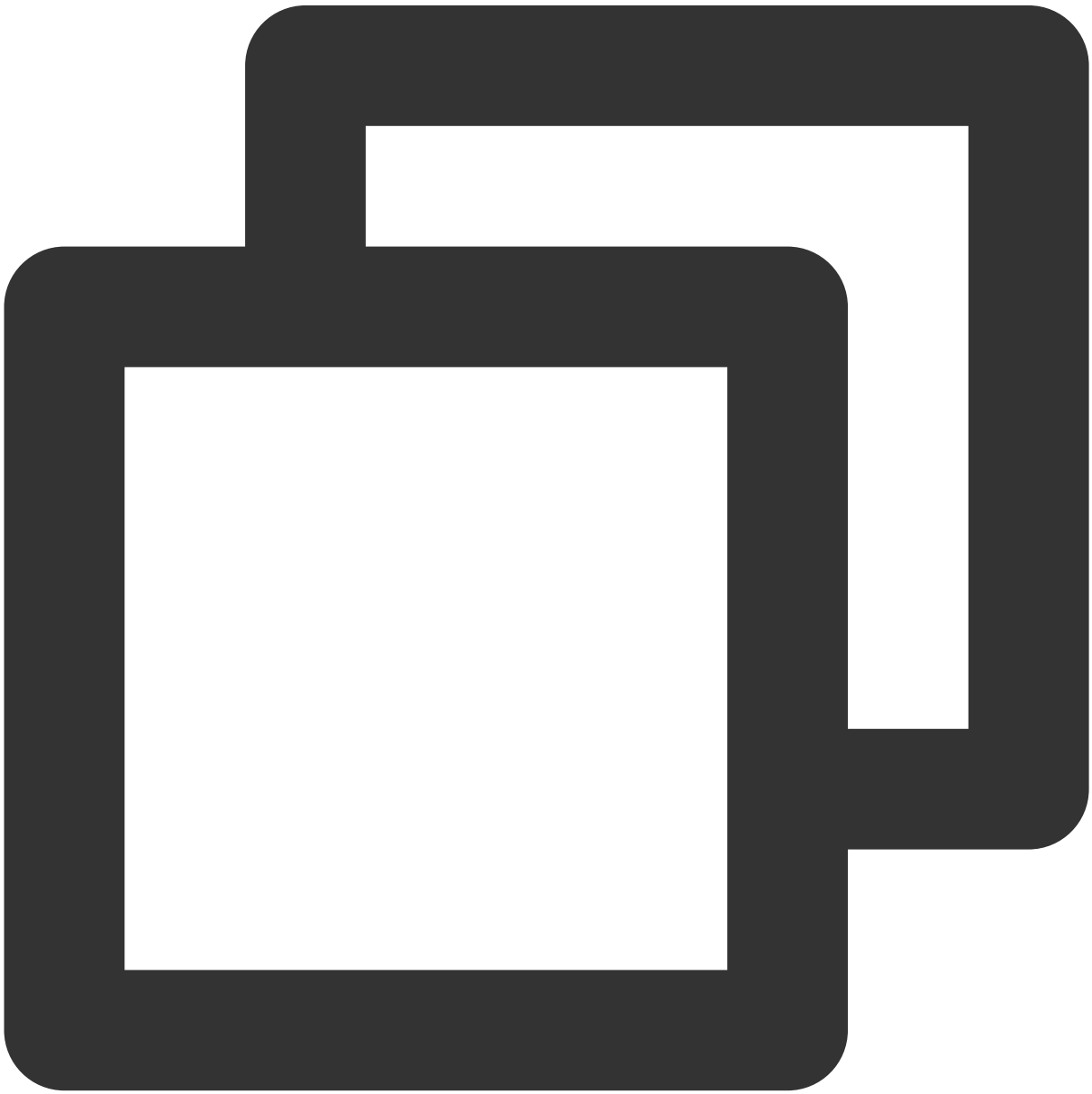


```
// Code for Windows
ITMGContextGetInstance()->GetAudioEffectCtrl()->StartAccompany(filePath,true,-1,0);
// Code for Android
ITMGContext.GetInstance(this).GetAudioEffectCtrl().StartAccompany(filePath,true,loo
// Code for iOS
[[[ITMGContext GetInstance] GetAudioEffectCtrl] StartAccompany:path loopBack:isLoop
```

Starting the playback of the accompaniment being downloaded

Call the `StartAccompanyDownloading` API to start downloading and playing back the accompaniment. The accompaniment download is implemented in the code. When the download is not completed, the file path can be passed as a parameter to `StartAccompanyDownloading`, which can implement download-and-play. `fileSize` is the estimated full file size. When calling this API to pass in a partially downloaded file, first ensure that the file size is at least 10 KB.

Function prototype



```
ITMGAudioEffectCtrl virtual int StartAccompany(const char* filePath, bool loopBack,
```

Note:

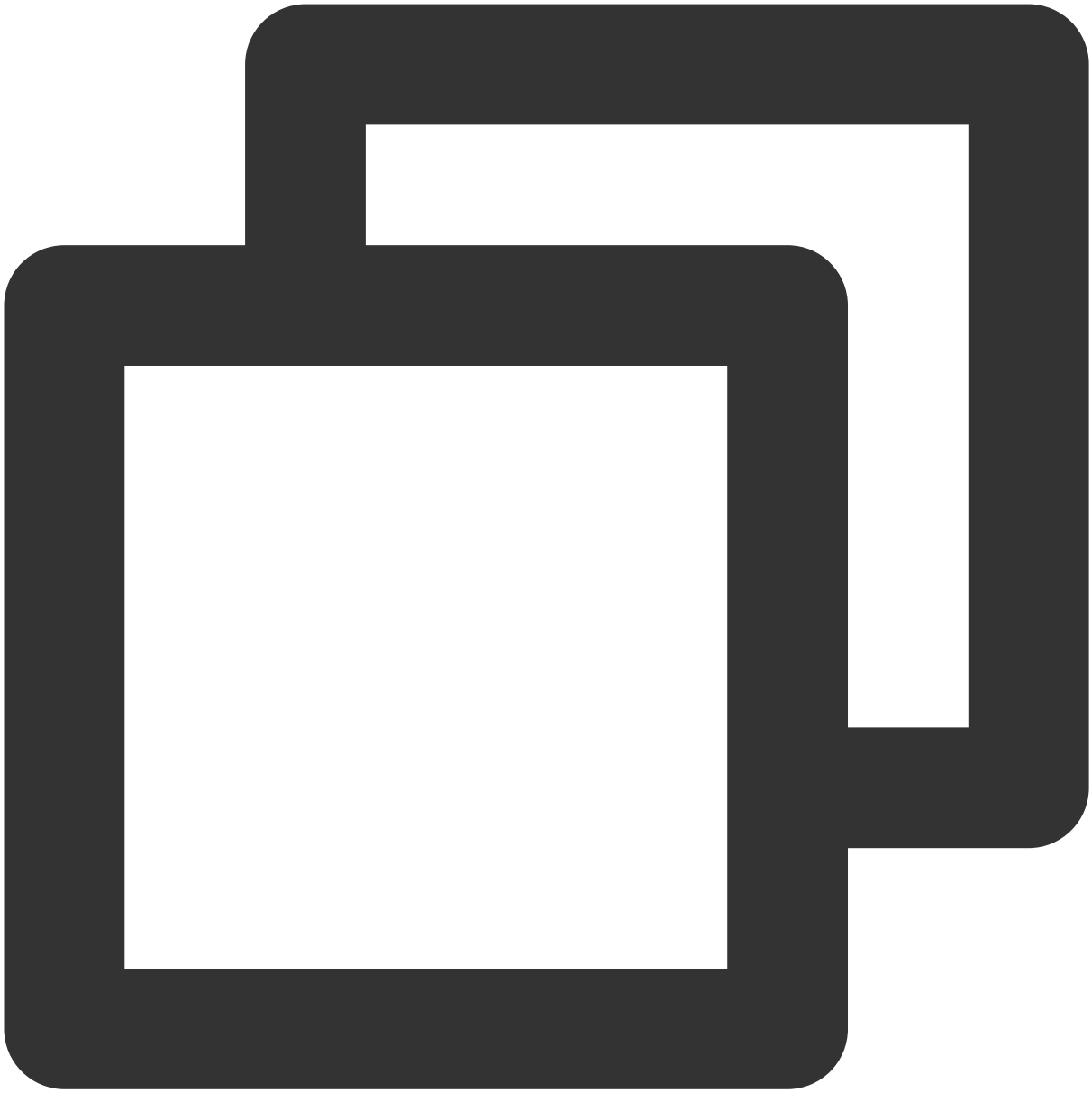
For the iOS client, follow the steps below:

1. To use this feature on iOS, you need to [download the mp3 library](https://picture-1256313114.cos.ap-beijing.myqcloud.com/mp3_codec.zip?_ga =1.162366908.1422691217.1594629603) and import it into your project.
2. Import the downloaded library and add it through “Link Binary With Libraries”.
3. Add the header file “TMGEngine_adv.h” to the same directory as the other SDK header files in your project.

Callback for the accompaniment playback

After the accompaniment is over, call the function `OnEvent` , and the event message ITMG_MAIN_EVENT_TYPE_ACCOMPANY_FINISH will be returned.
The returned parameter `data` includes “result” and “file_path”.

Sample code



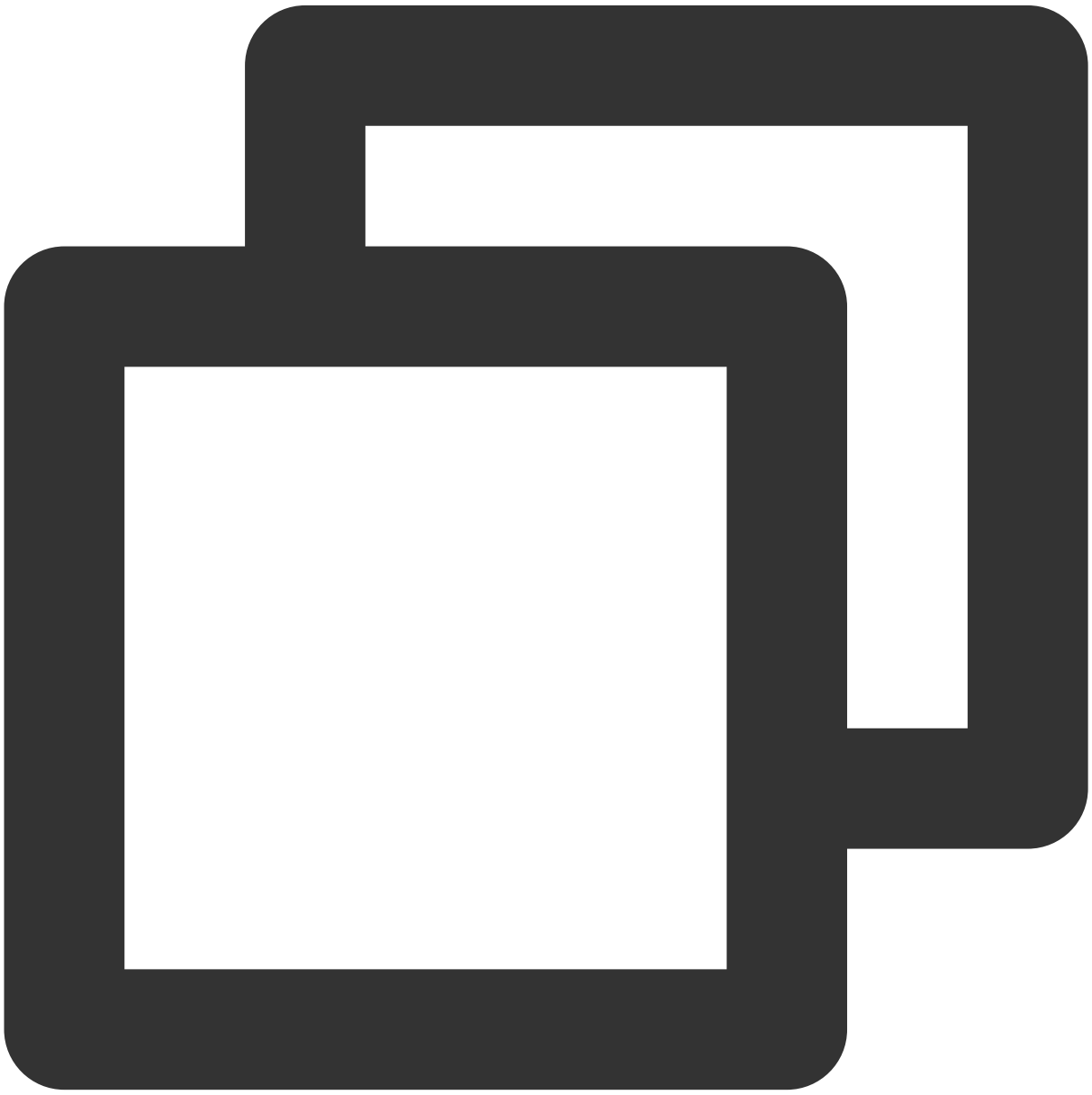
```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
        {
        // Process
        break;
        }
        ...
        case ITMG_MAIN_EVENT_TYPE_ACCOMPANY_FINISH:
        {
        // Process
```

```
            break;
            }
        }
    }
```

Stopping the accompaniment playback

This API (StopAccompany) is used to stop playing back the accompaniment.

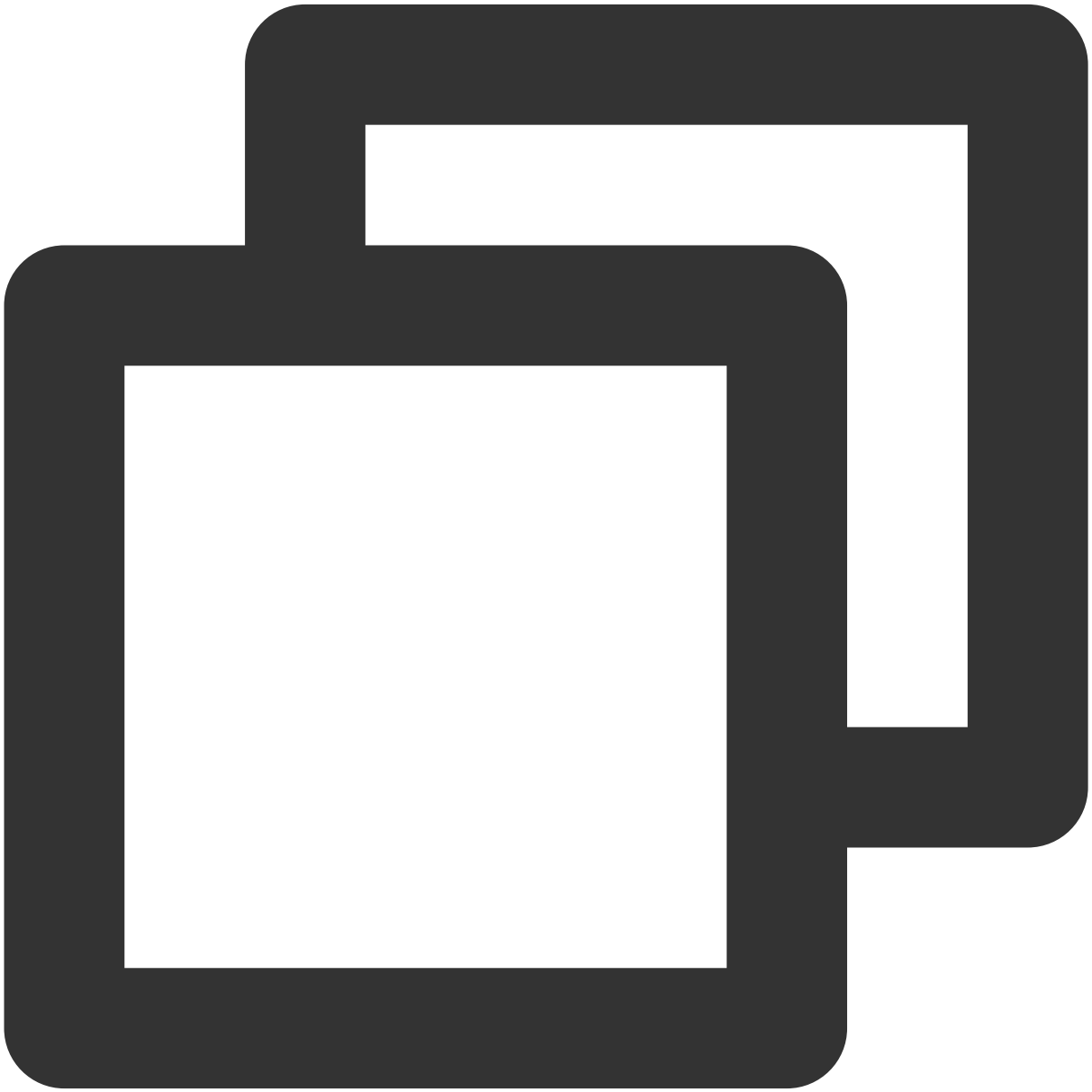
Function prototype



```
ITMGAudioEffectCtrl virtual int StopAccompany(int duckerTime)
```

Parameter	Type	Description
duckerTime	int	Ducking time

Sample code

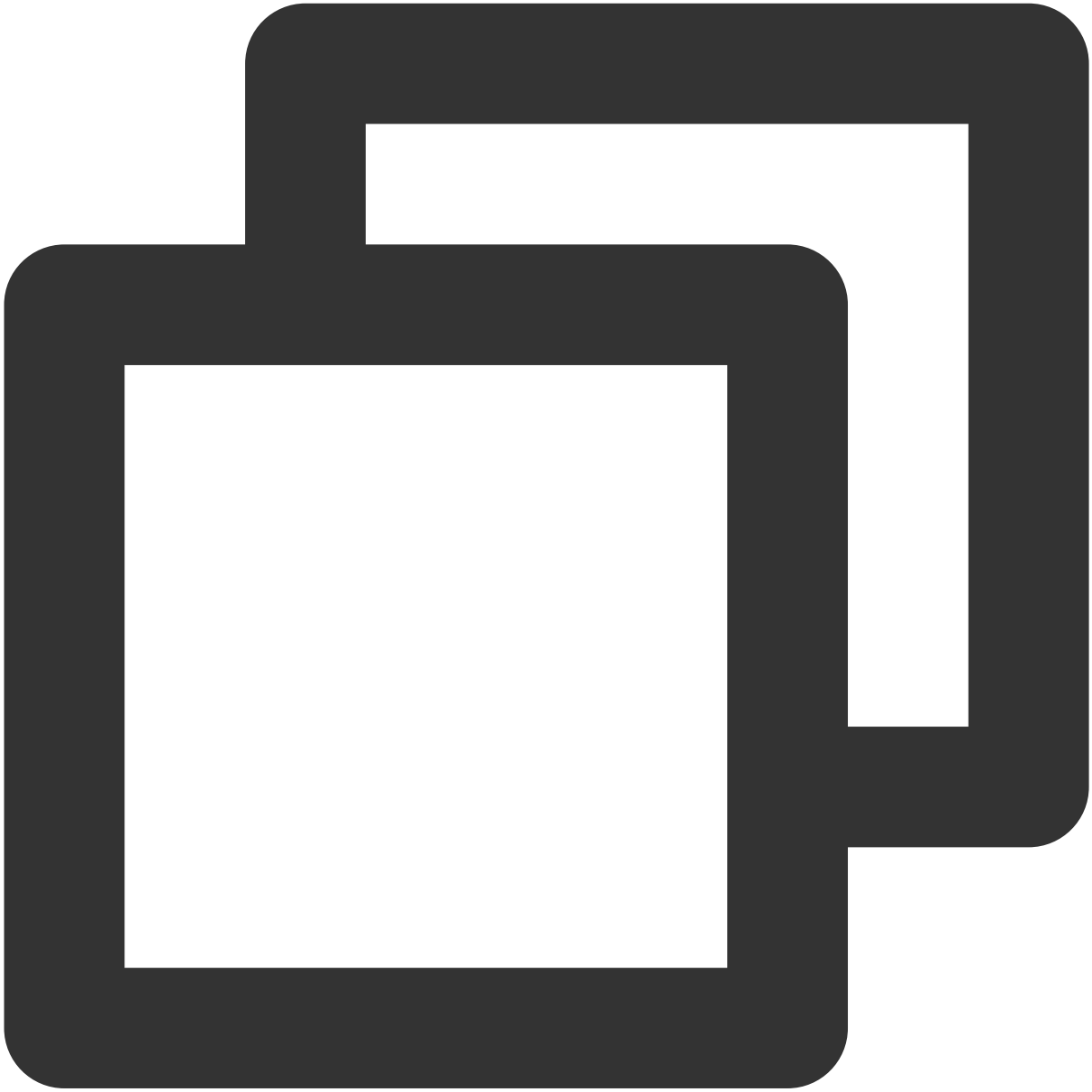


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->StopAccompany(0);
```

Indicating whether the accompaniment is over

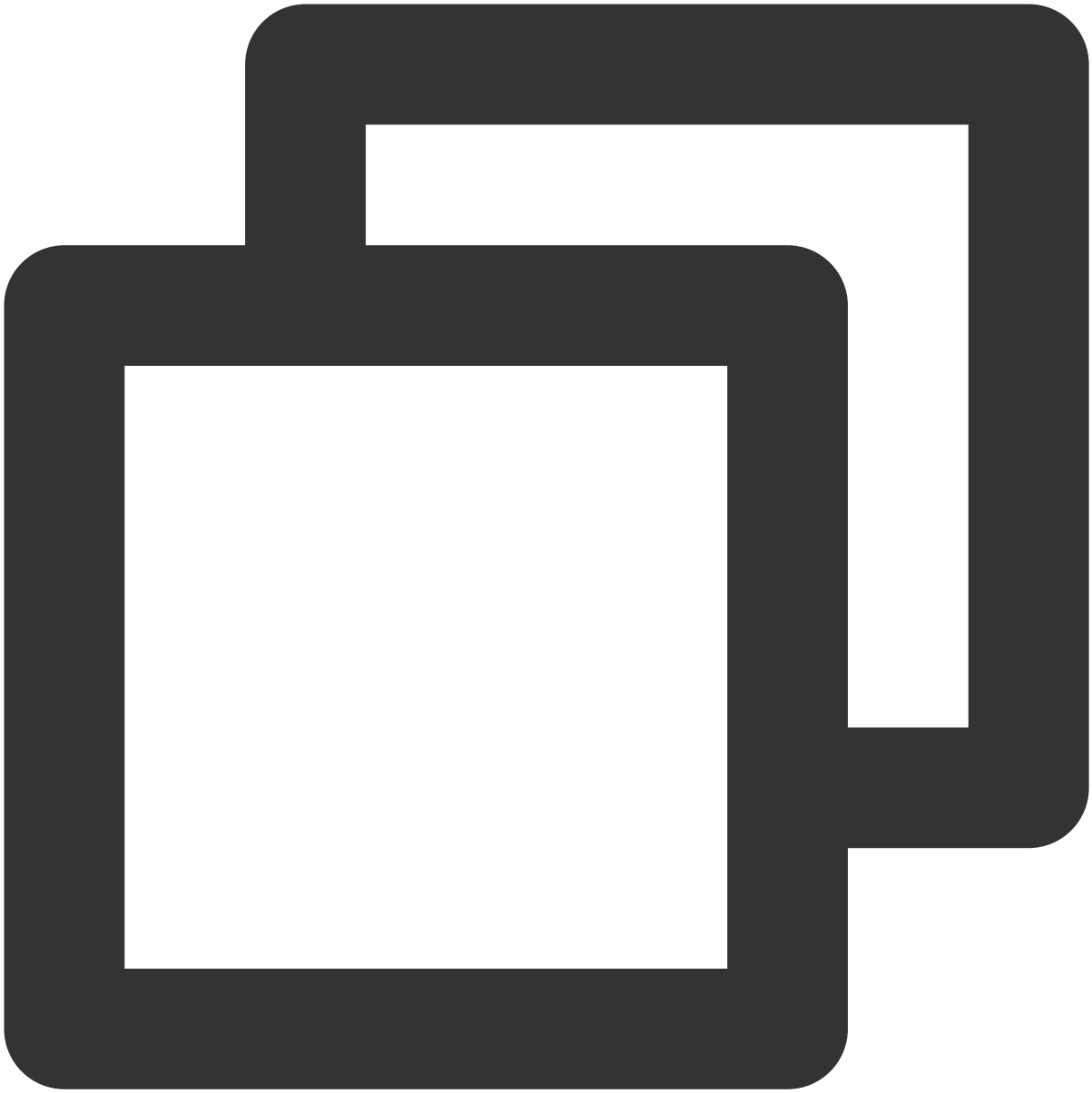
If it is over, `true` is returned. If it is not, `false` is returned.

Function prototype



```
ITMGAudioEffectCtrl virtual bool IsAccompanyPlayEnd()
```

Sample code

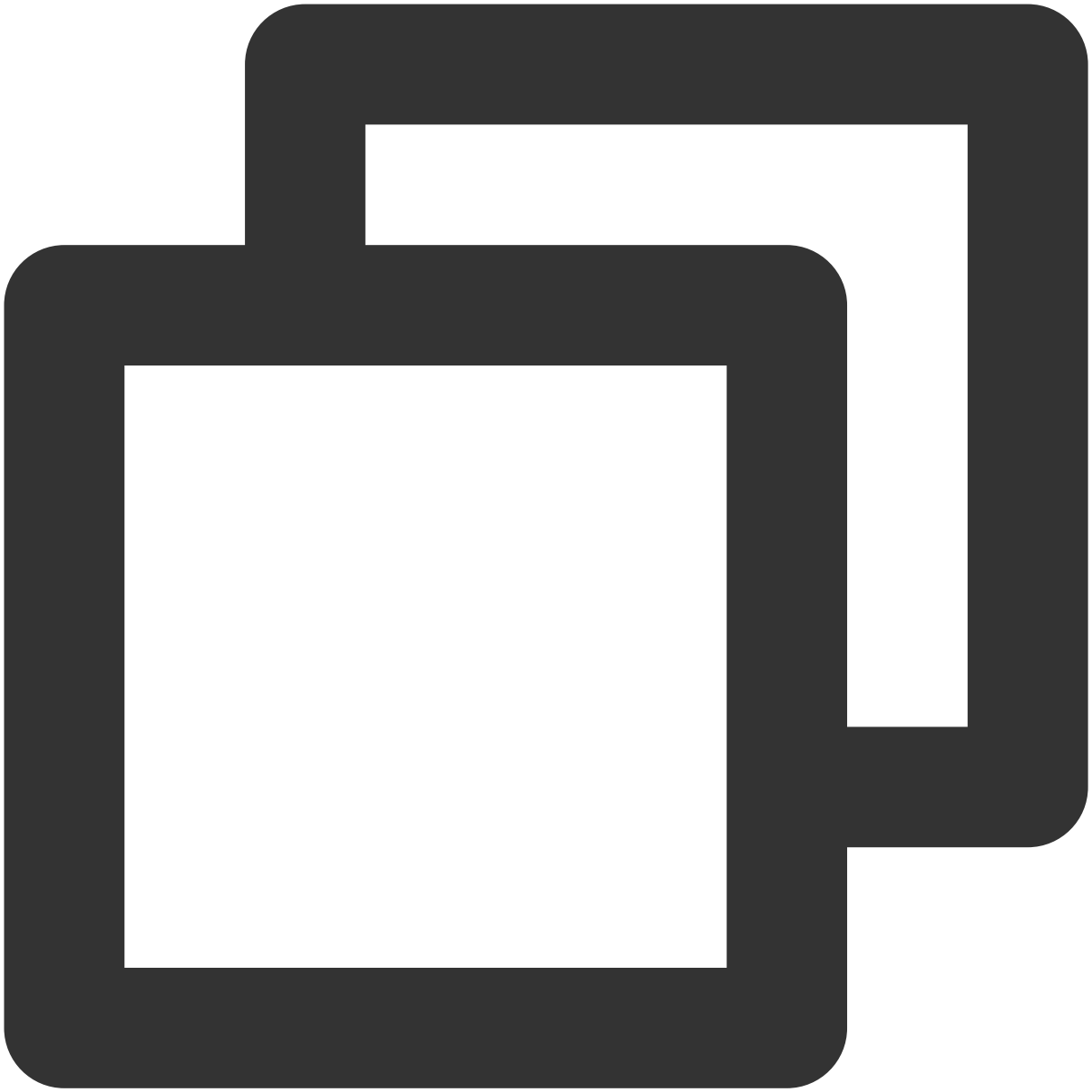


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->IsAccompanyPlayEnd();
```

Pausing the accompaniment playback

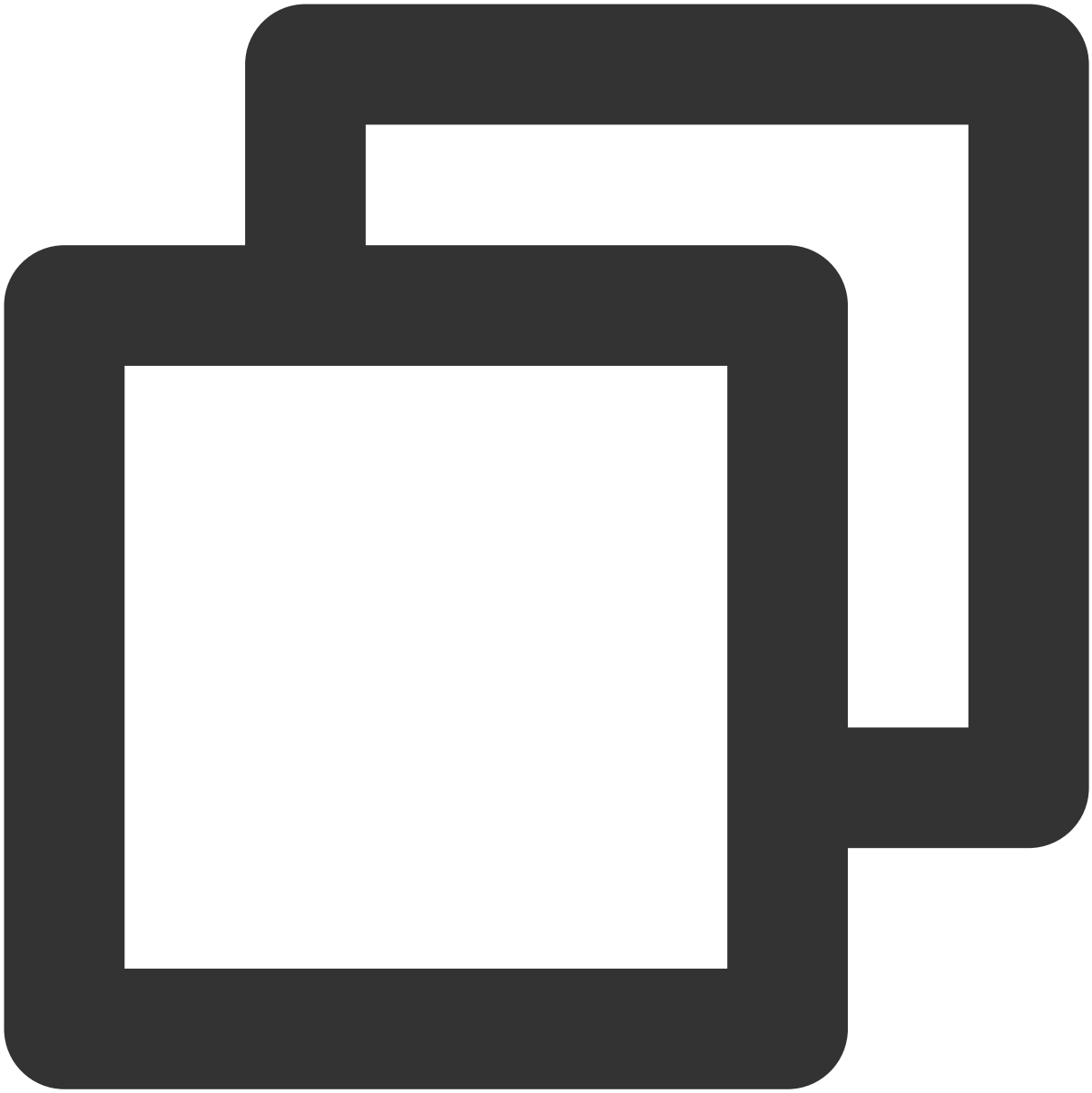
This API (PauseAccompany) is used to pause the accompaniment playback.

Function prototype



```
ITMGAudioEffectCtrl virtual int PauseAccompany()
```

Sample code

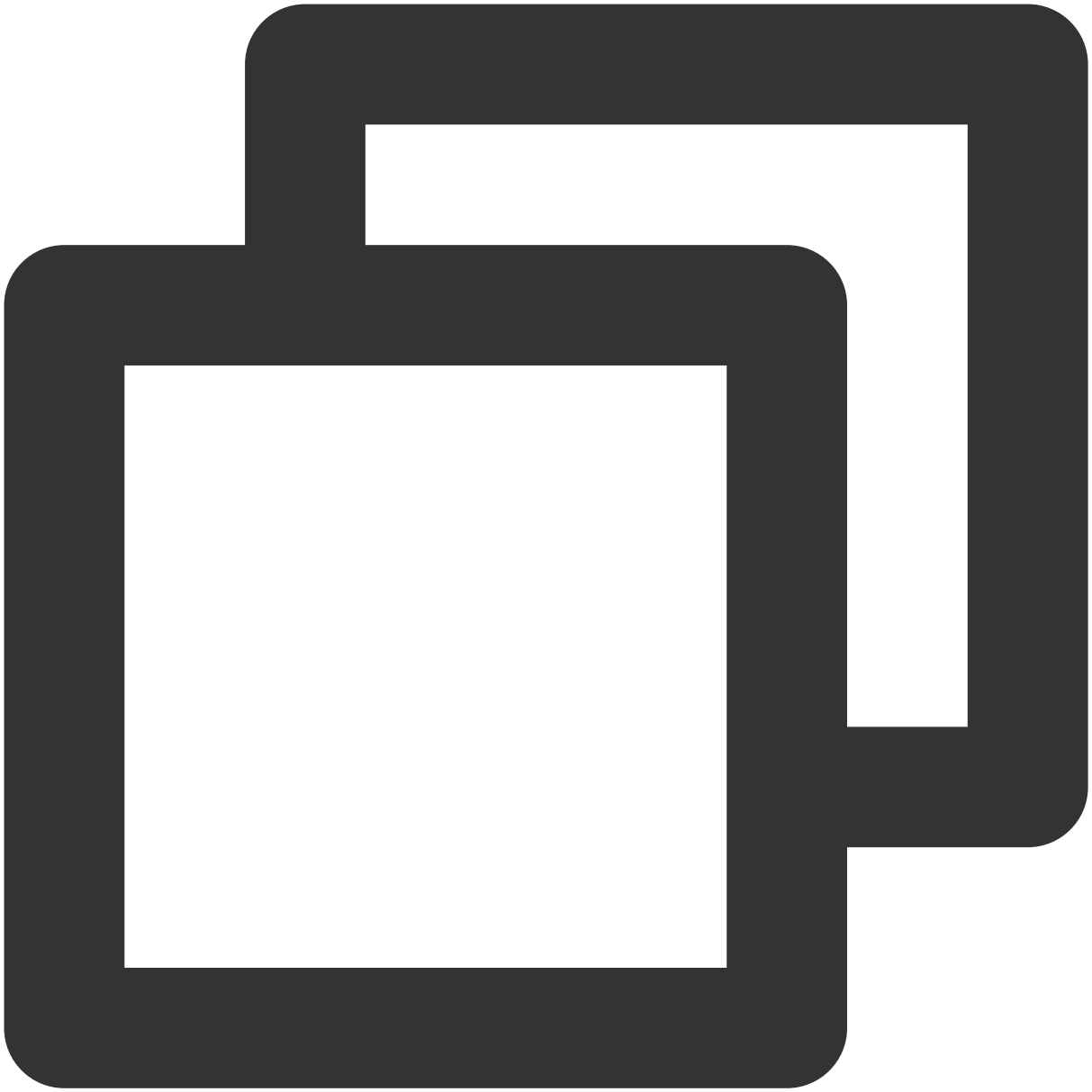


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->PauseAccompany();
```

Resuming the accompaniment playback

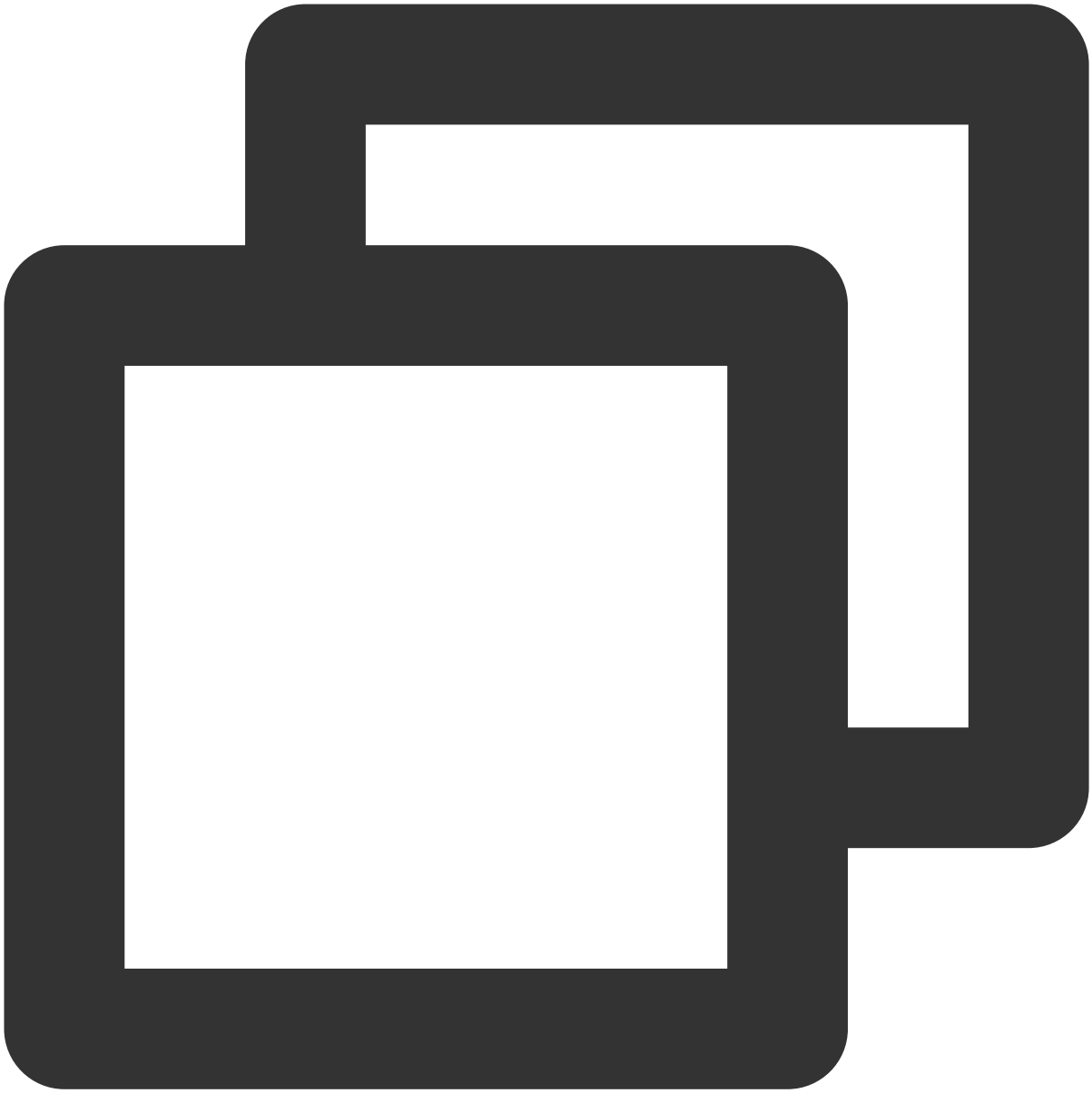
This API (ResumeAccompany) is used to resume the accompaniment playback.

Function prototype



```
ITMGAudioEffectCtrl virtual int ResumeAccompany()
```

Sample code

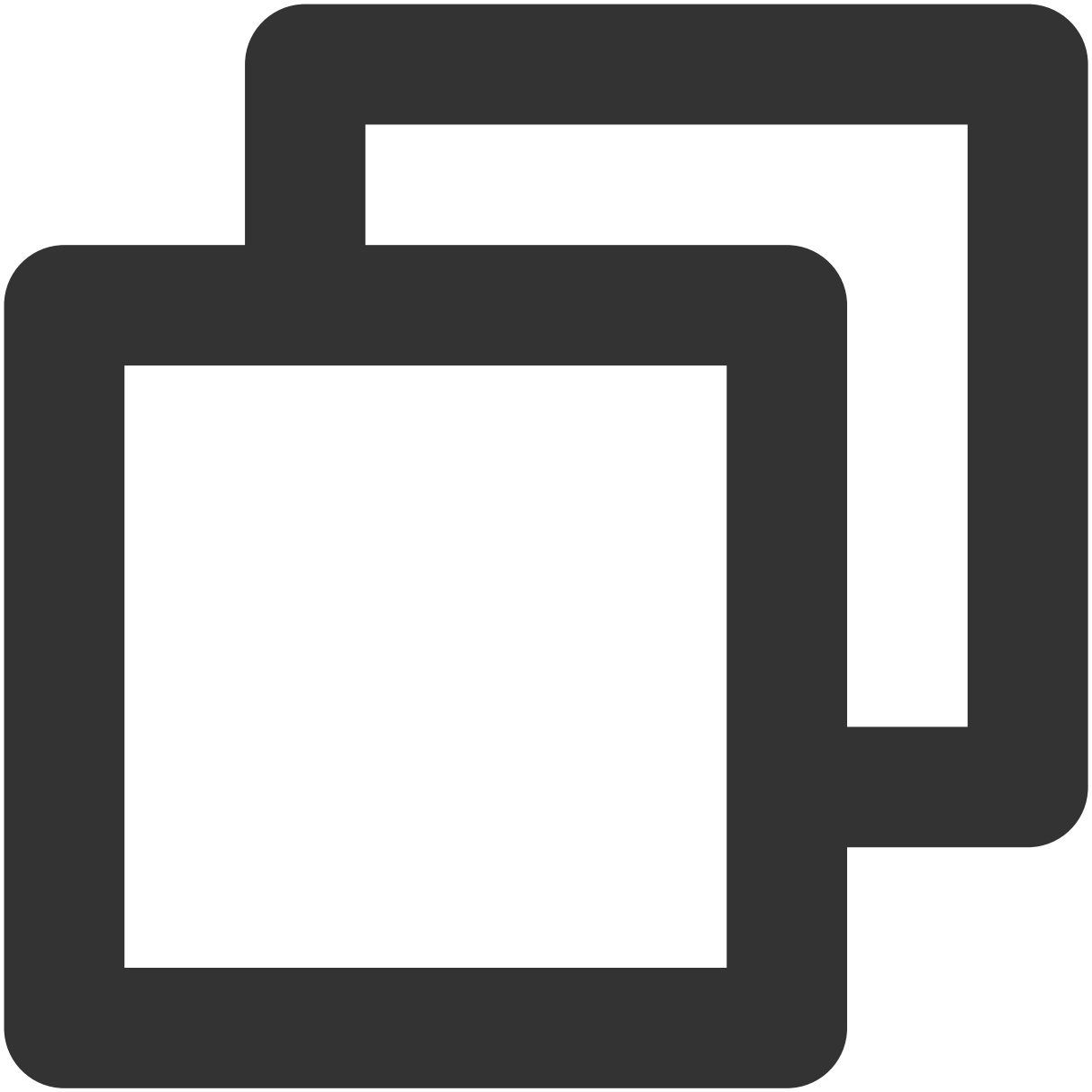


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->ResumeAccompany();
```

Specifying whether the speaker can hear the accompaniment

This API is used to specify whether the speaker can hear the accompaniment.

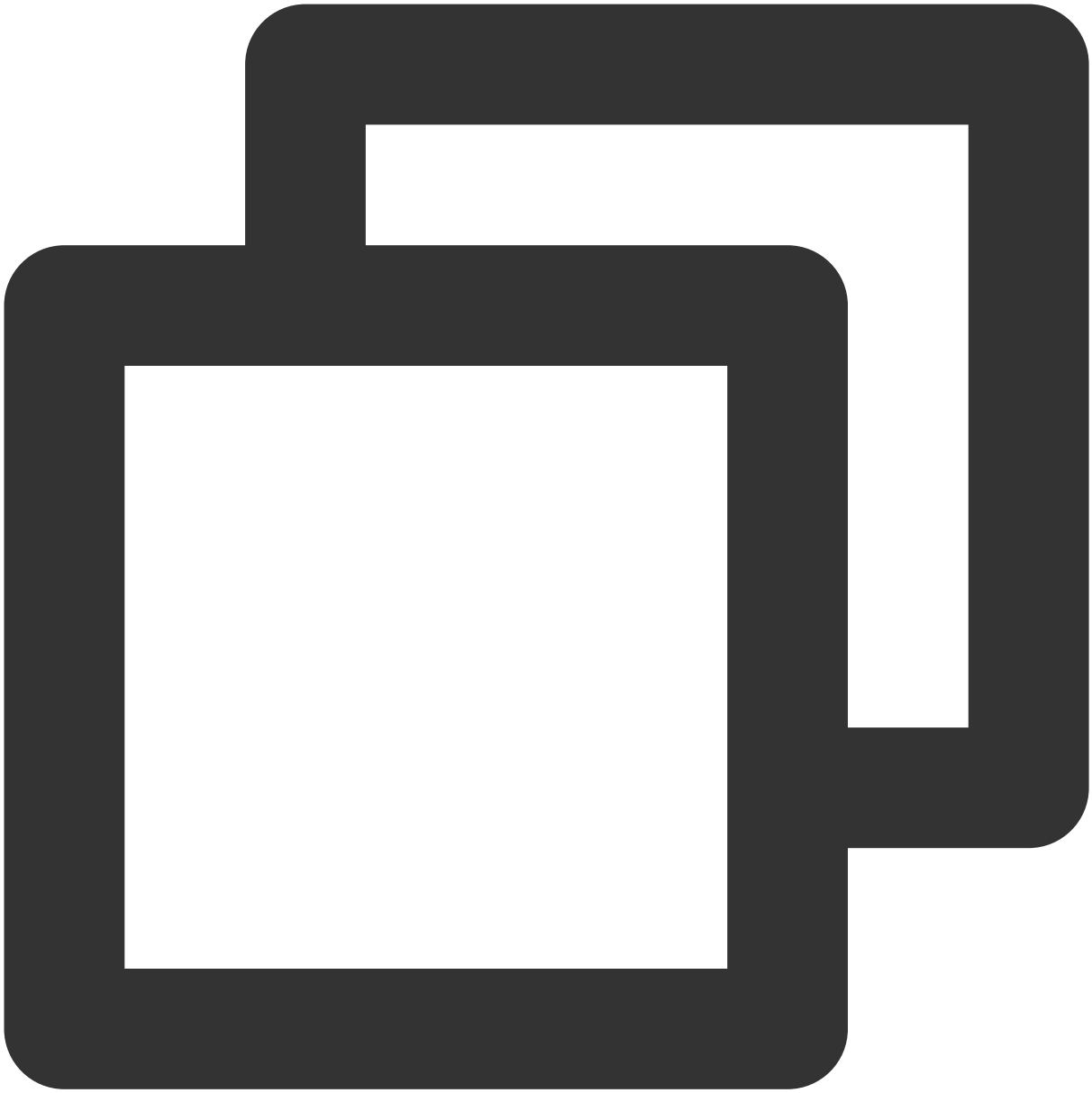
Function prototype



```
ITMGAudioEffectCtrl virtual int EnableAccompanyPlay(bool enable)
```

Parameter	Type	Description
enable	bool	Indicates whether the audience can hear the accompaniment.

Sample code

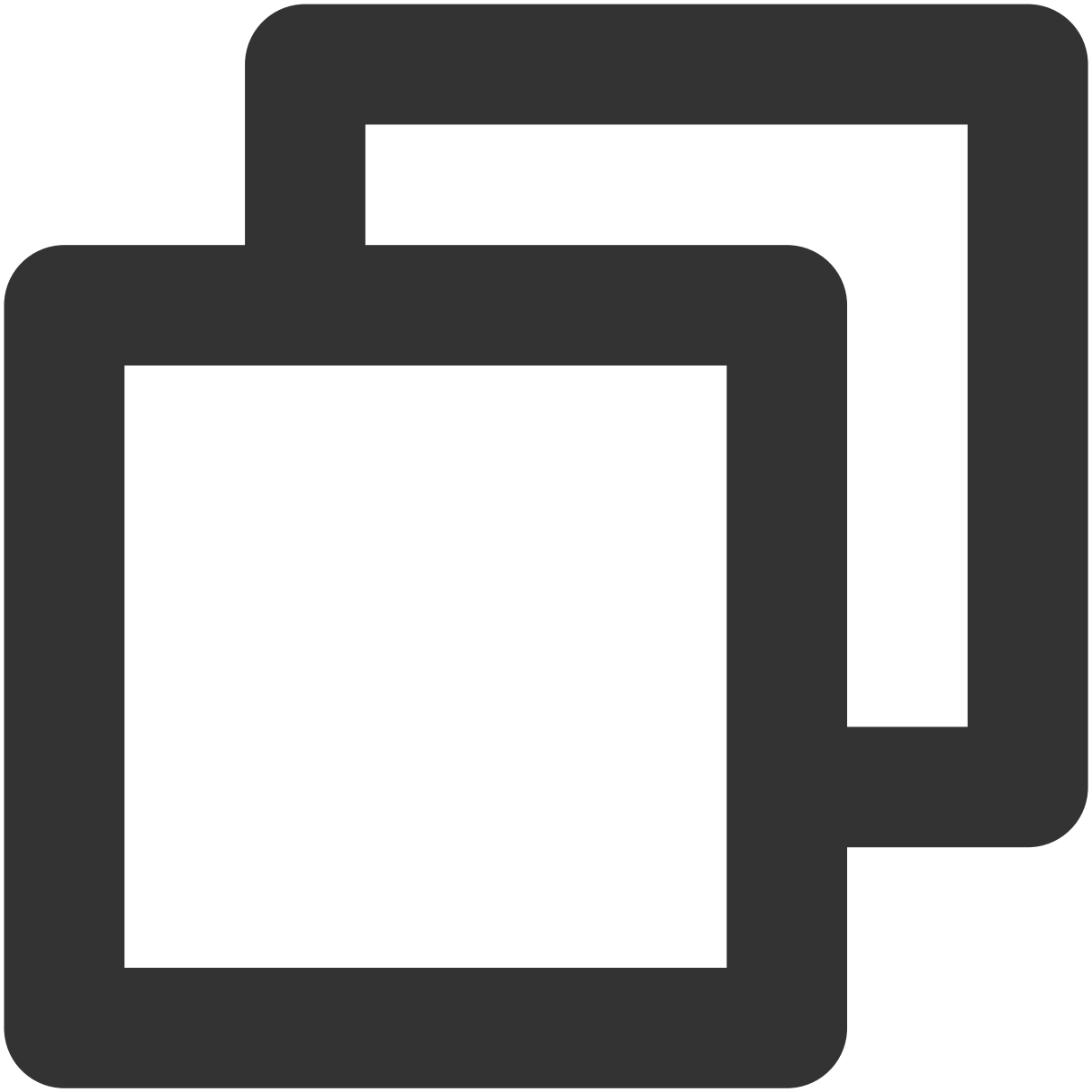


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->EnableAccompanyPlay(false);
```

Specifying whether the audience can hear the accompaniment

This API is used to specify whether the audience can hear the accompaniment.

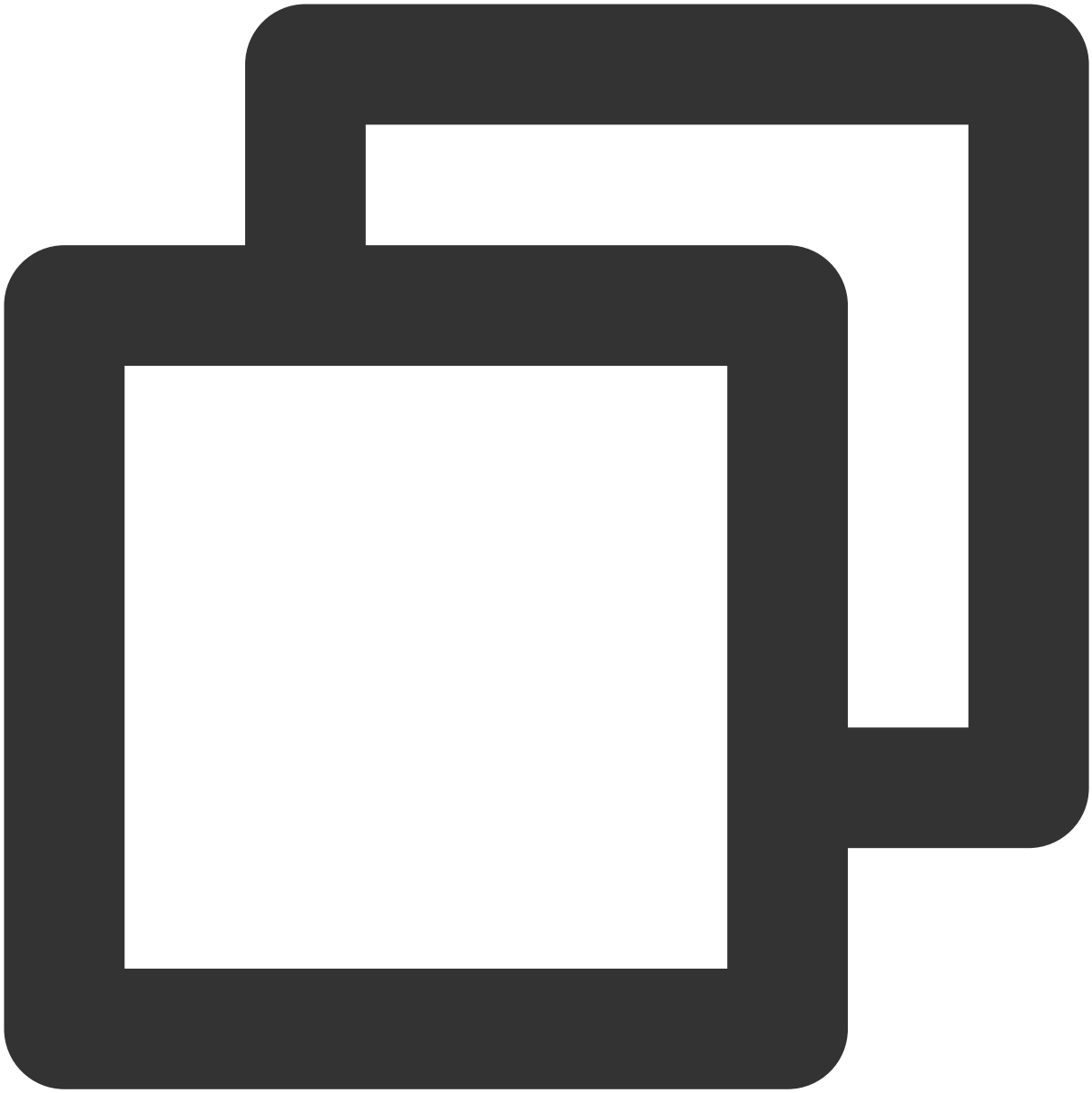
Function prototype



```
ITMGAudioEffectCtrl virtual int EnableAccompanyLoopBack(bool enable)
```

Parameter	Type	Description
enable	bool	Indicates whether the audience can hear the accompaniment.

Sample code

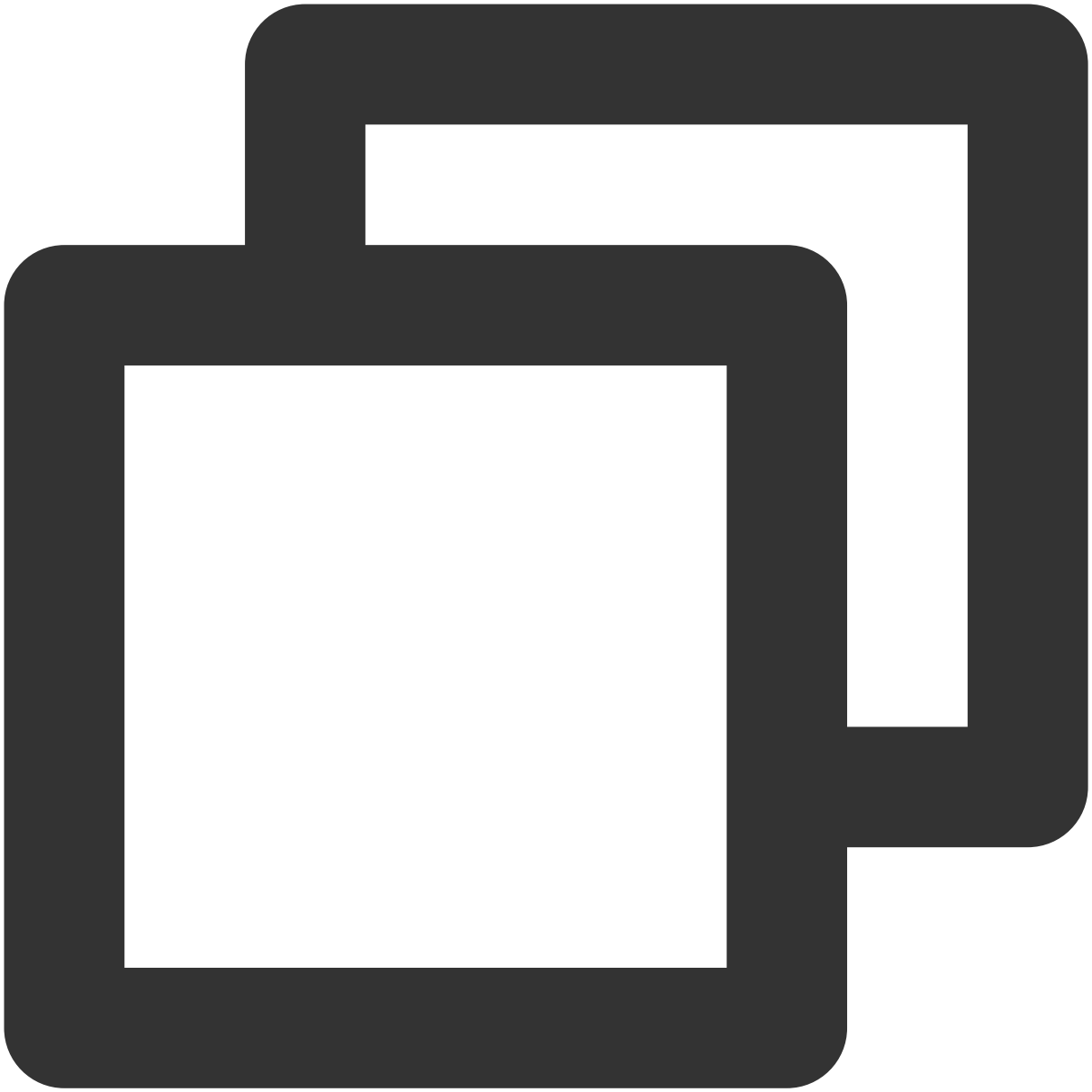


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->EnableAccompanyLoopBack(false);
```

Setting the accompaniment volume

This API (SetAccompanyVolume) is used to set the accompaniment volume. Value range: 0 - 200. The default value is 100. A value greater than 100 means volume up, while a value less than 100 means volume down.

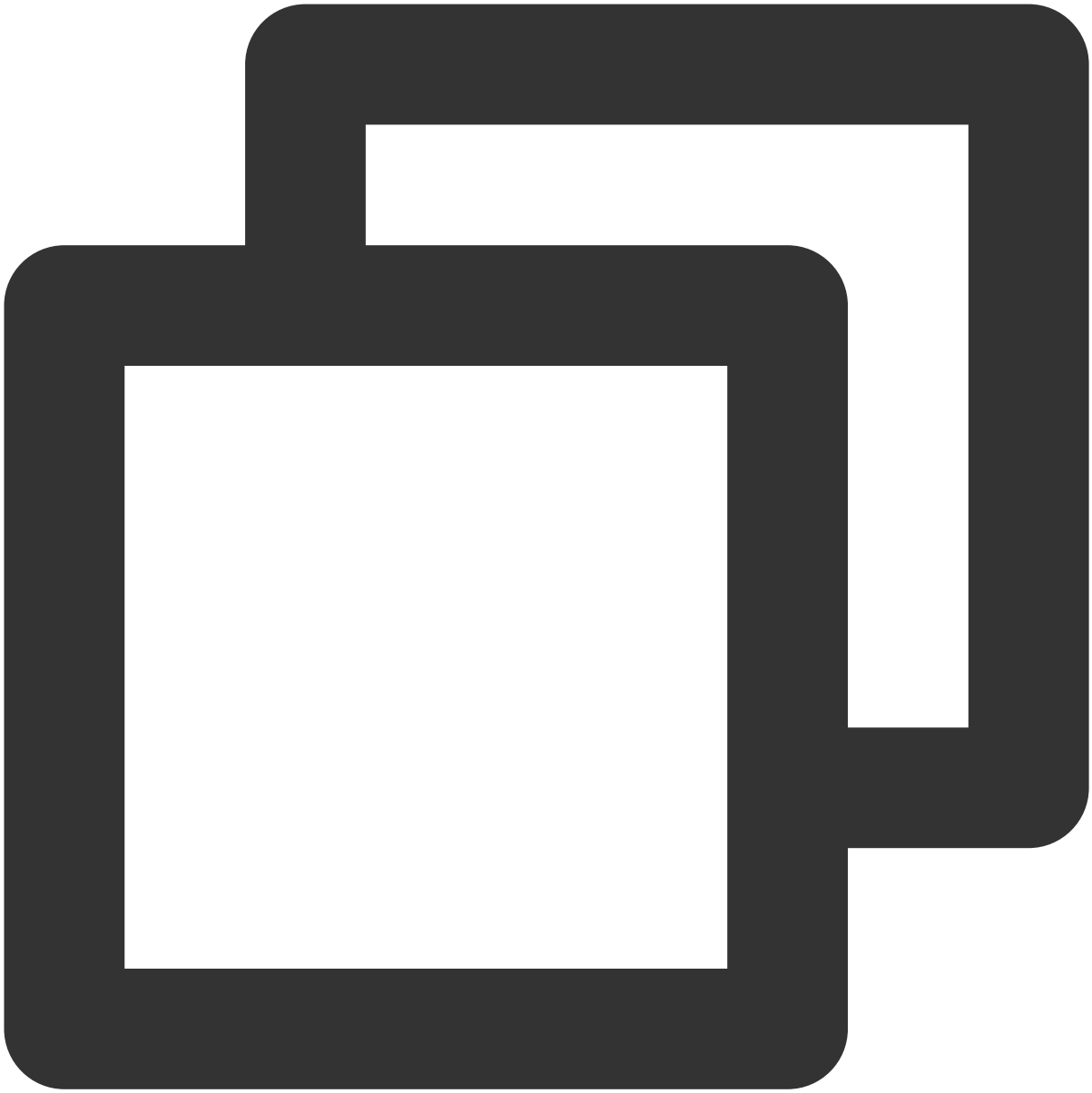
Function prototype



```
ITMGAudioEffectCtrl virtual int SetAccompanyVolume(int vol)
```

Parameter	Type	Description
vol	int	Specifies the volume value.

Sample code

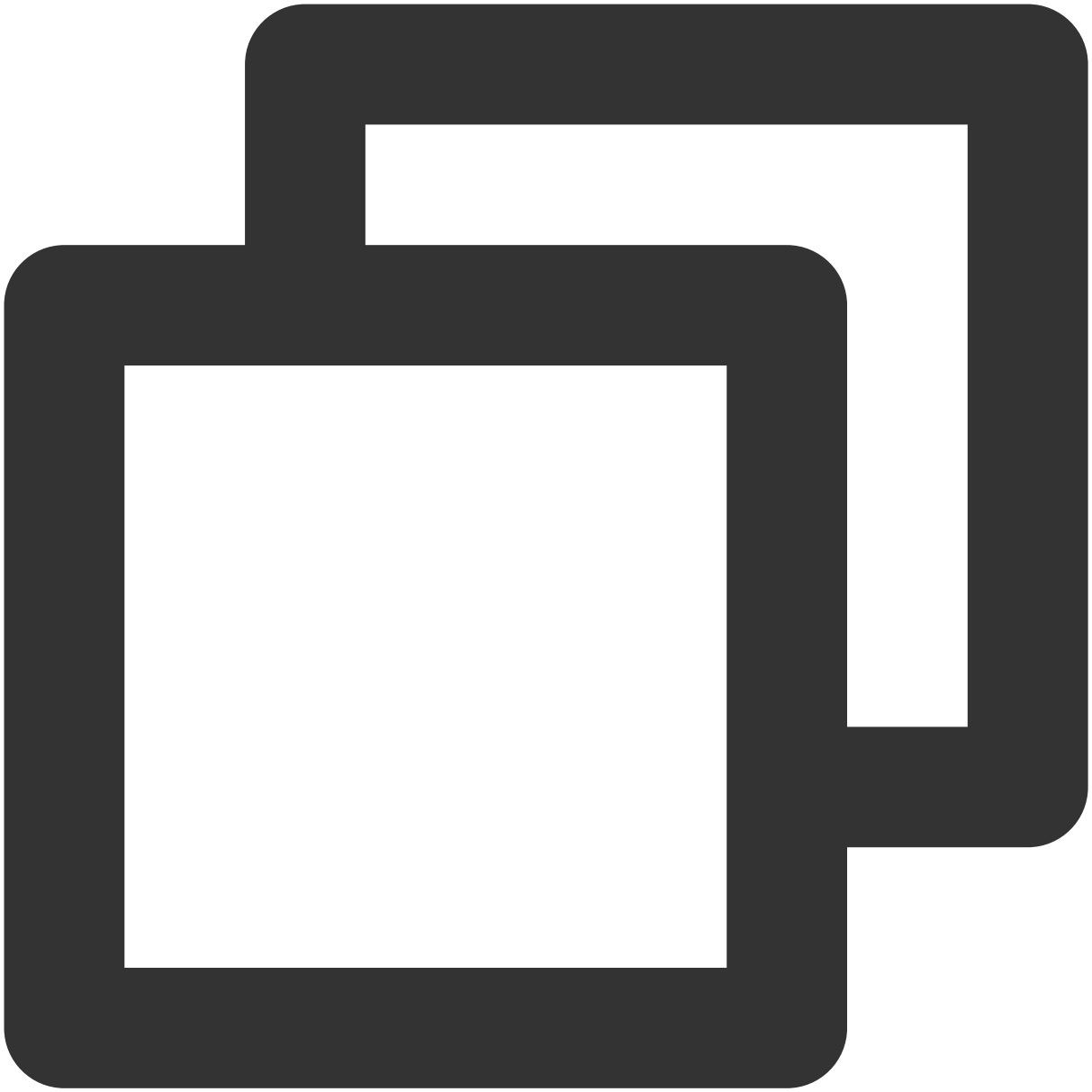


```
int vol=100;
ITMGContextGetInstance()->GetAudioEffectCtrl()->SetAccompanyVolume(vol);
```

Getting the accompaniment volume

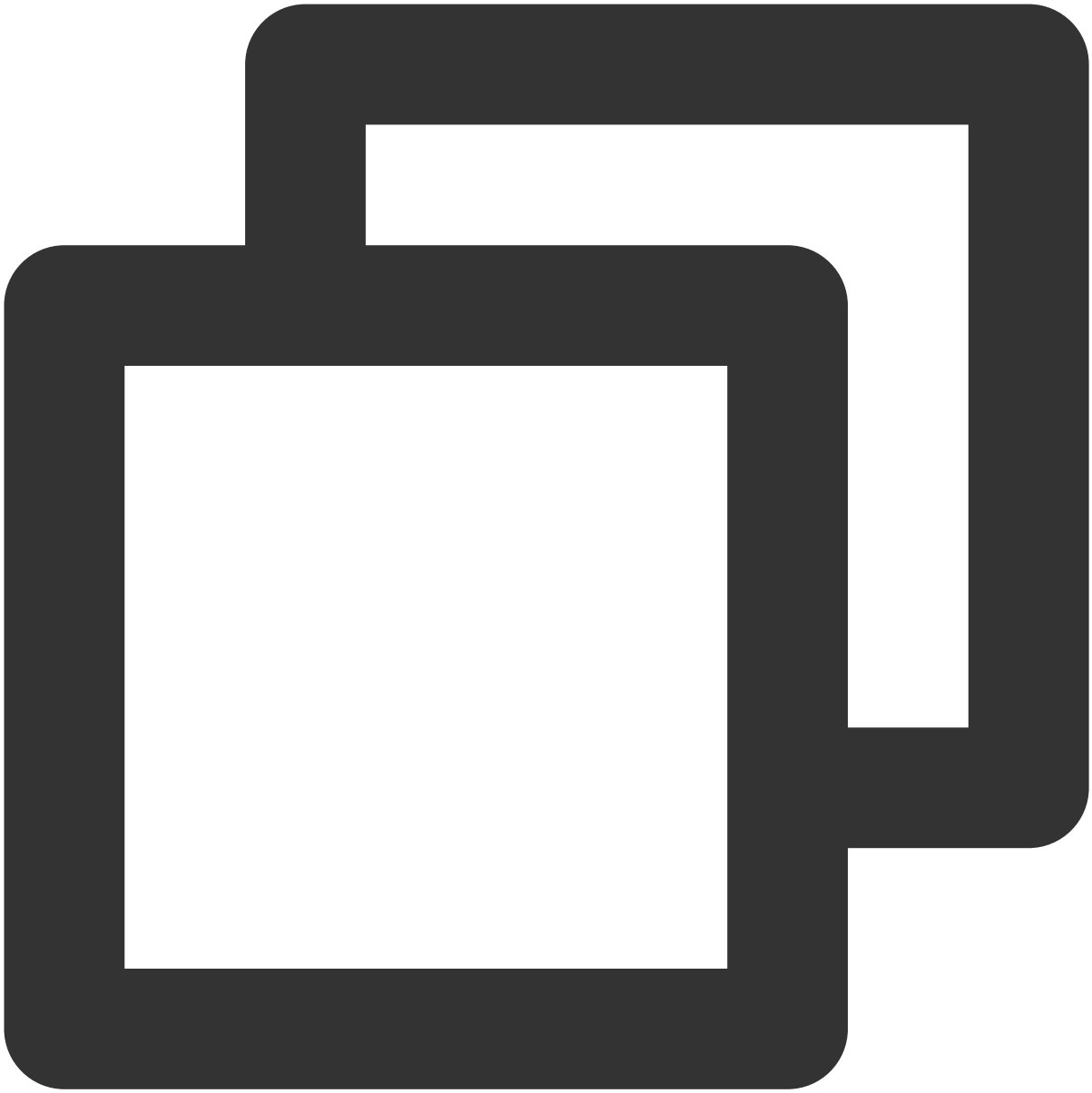
This API (GetAccompanyVolume) is used to obtain the accompaniment volume.

Function prototype



```
ITMGAudioEffectCtrl virtual int GetAccompanyVolume()
```

Sample code

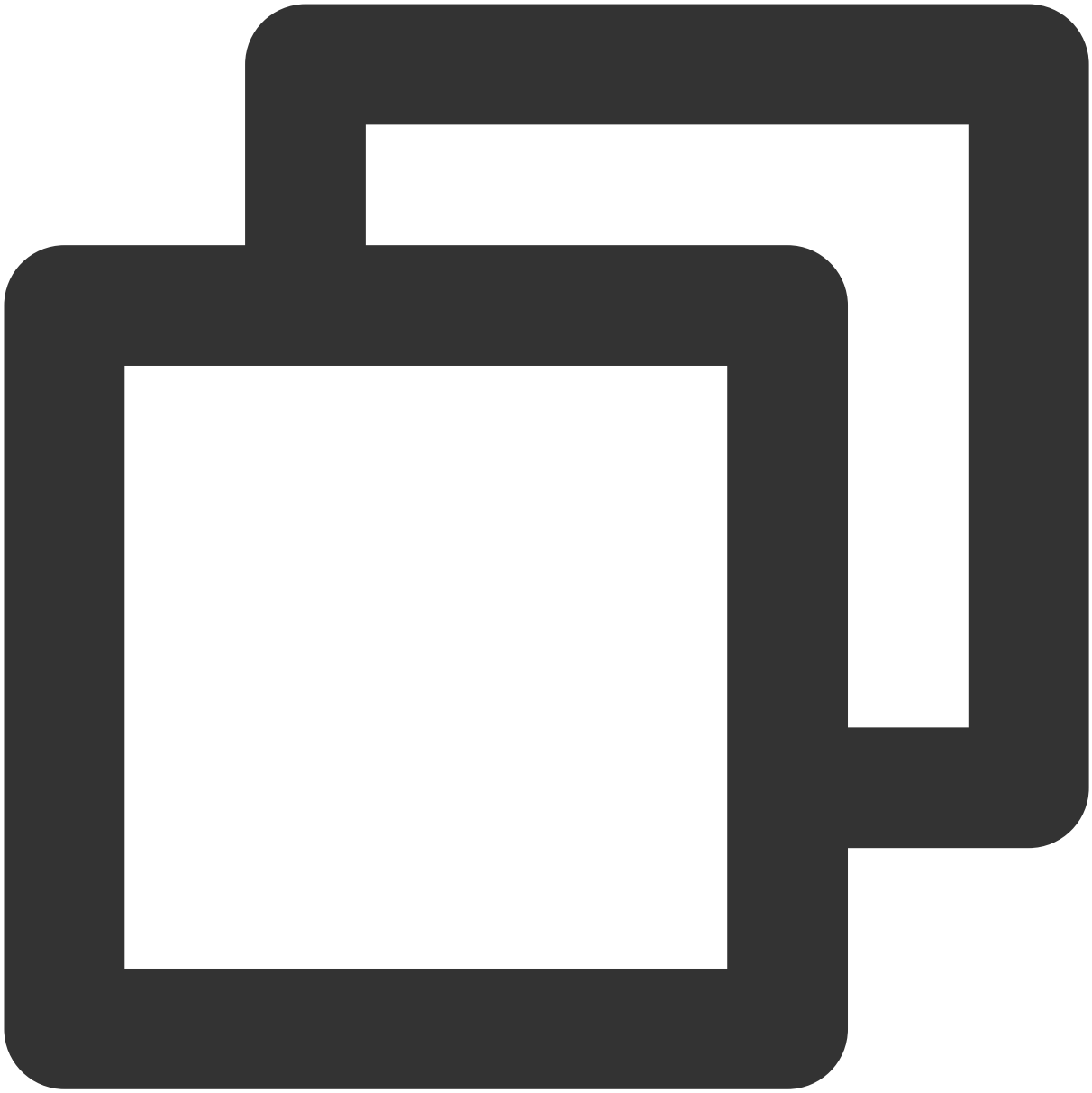


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->GetAccompanyVolume();
```

Getting the accompaniment playback progress

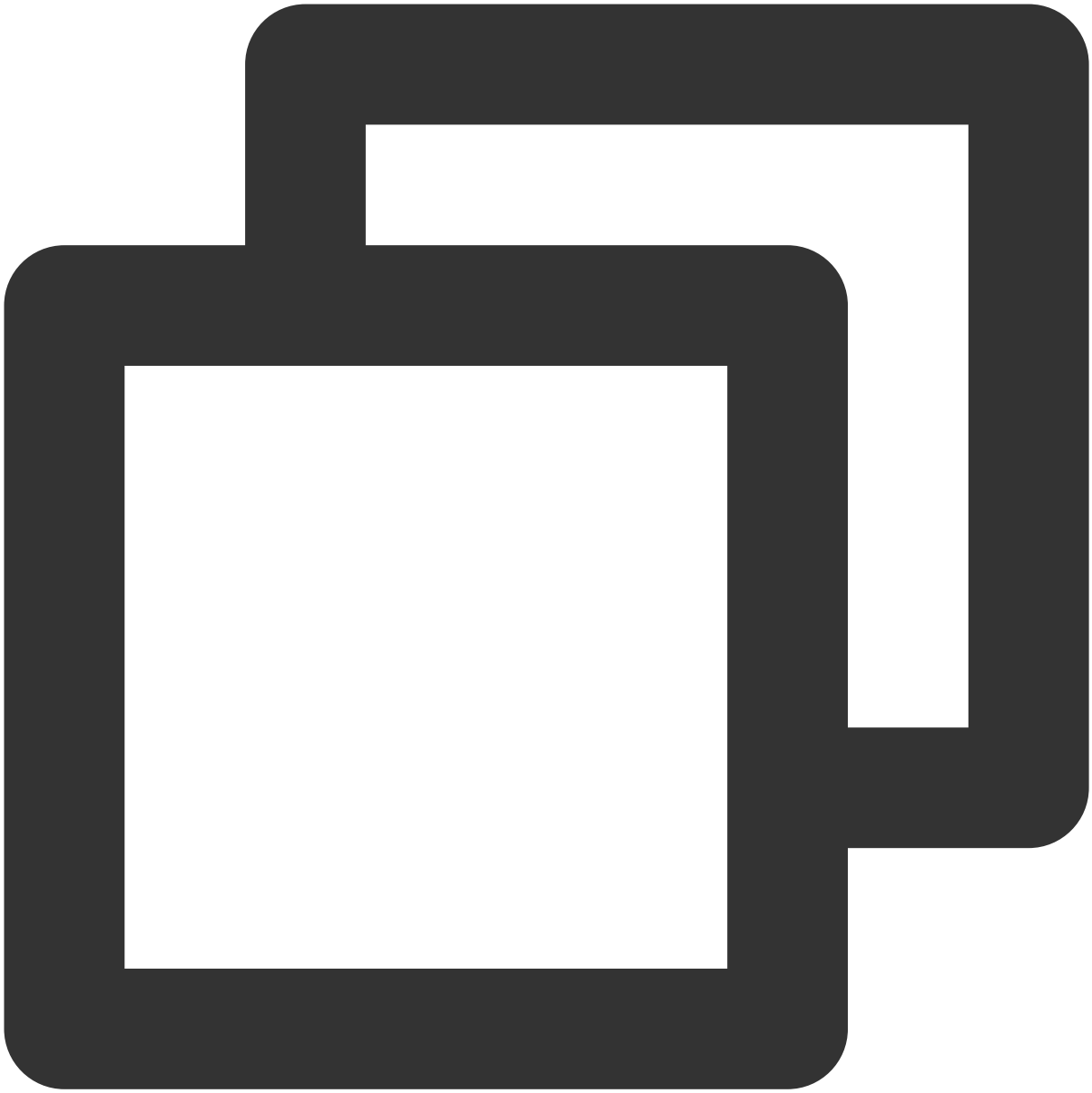
This action requires using both of these two APIs: `GetAccompanyFileTotalTimeByMs` and `GetAccompanyFileCurrentPlayedTimeByMs`. Please note that Current/Total = current loop times, and Current % Total = current loop playback position.

Function prototype



```
ITMGAudioEffectCtrl virtual int GetAccompanyFileTotalTimeByMs()
ITMGAudioEffectCtrl virtual int GetAccompanyFileCurrentPlayedTimeByMs()
```

Sample code

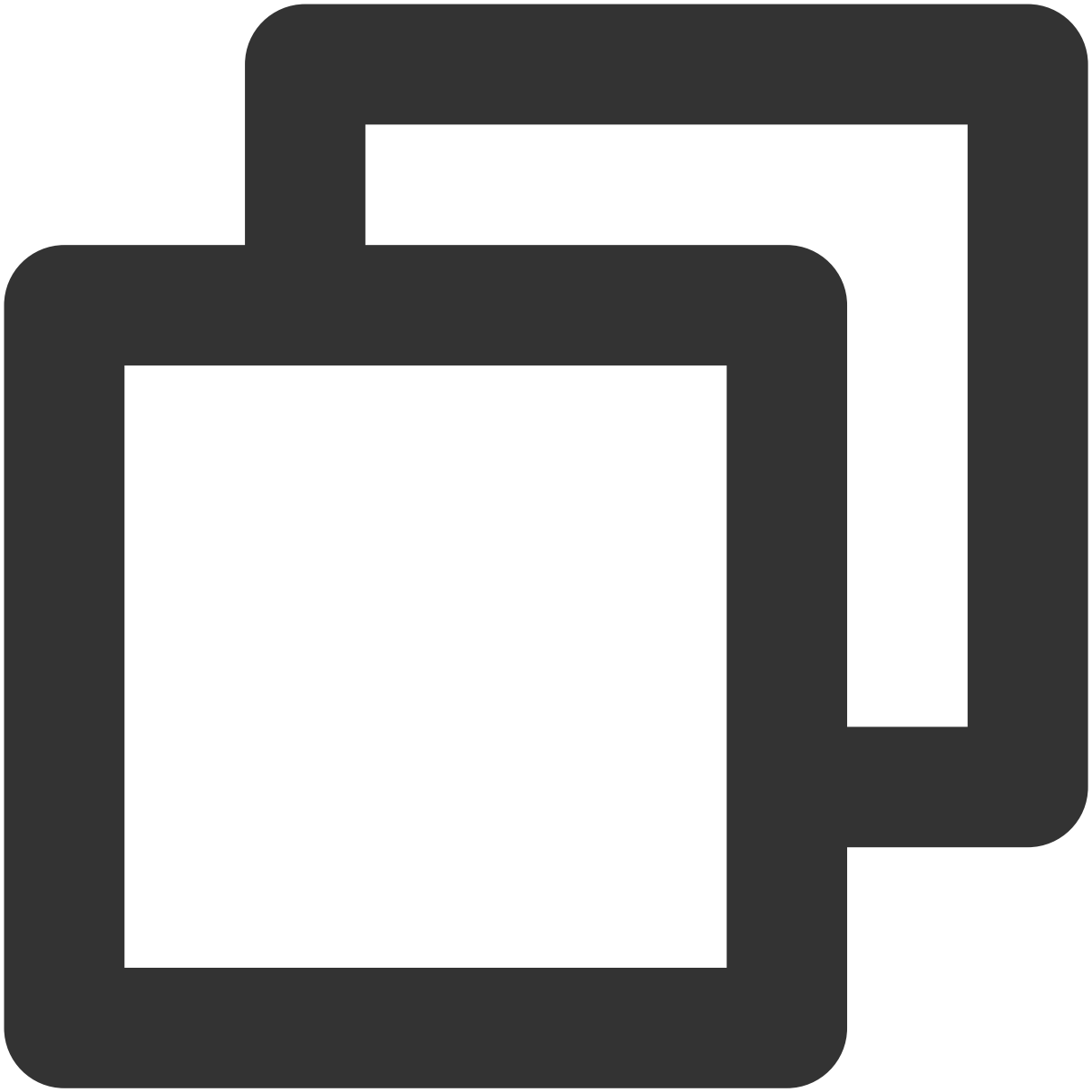


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->GetAccompanyFileTotalTimeByMs();
ITMGContextGetInstance()->GetAudioEffectCtrl()->GetAccompanyFileCurrentPlayedTimeBy
```

Setting the playback progress

This API (SetAccompanyFileCurrentPlayedTimeByMs) is used to set the playback progress.

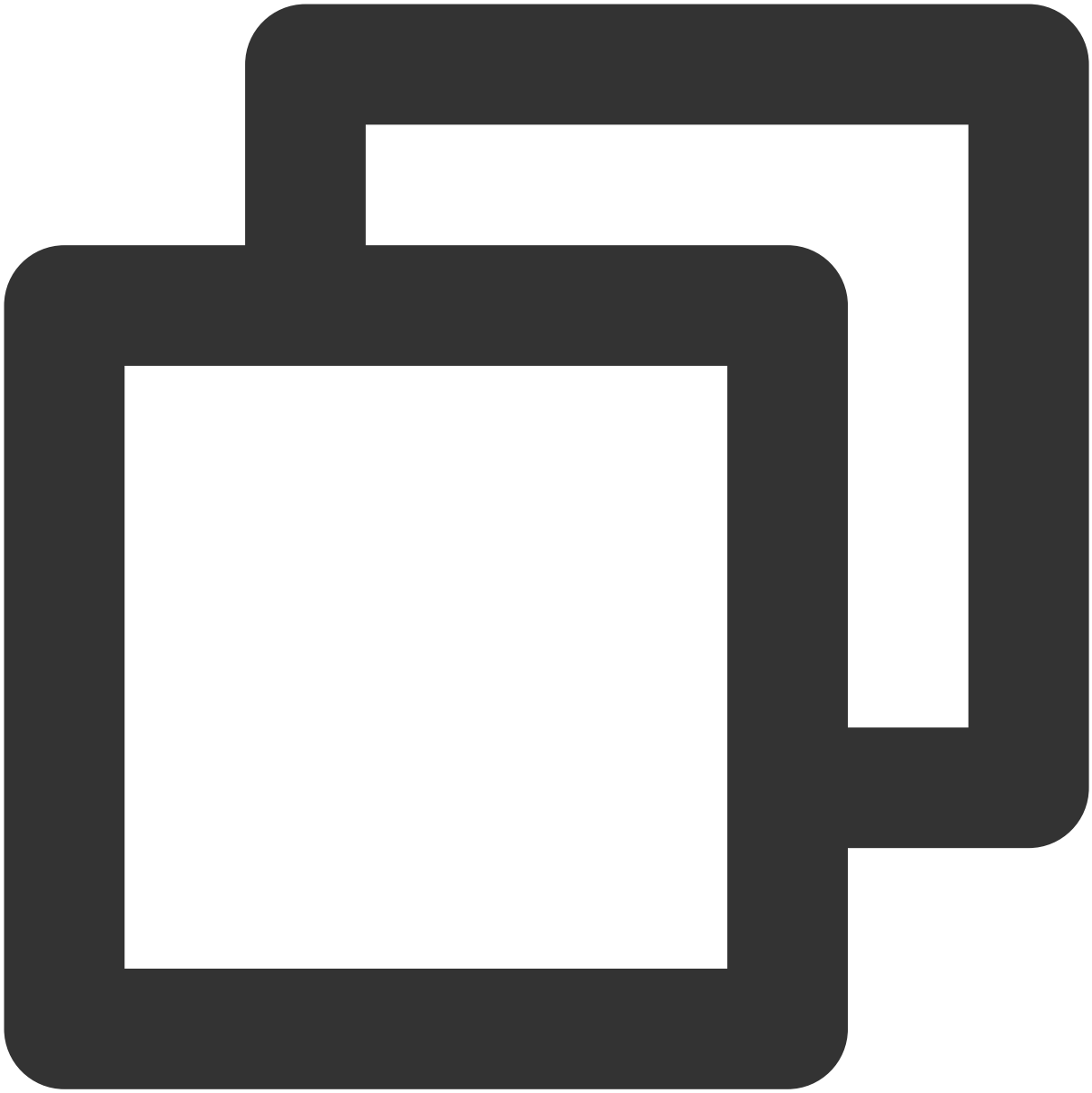
Function prototype



```
ITMGAudioEffectCtrl virtual int SetAccompanyFileCurrentPlayedTimeByMs(unsigned int
```

Parameter	Type	Description
time	int	Specifies the playback progress in milliseconds

Sample code

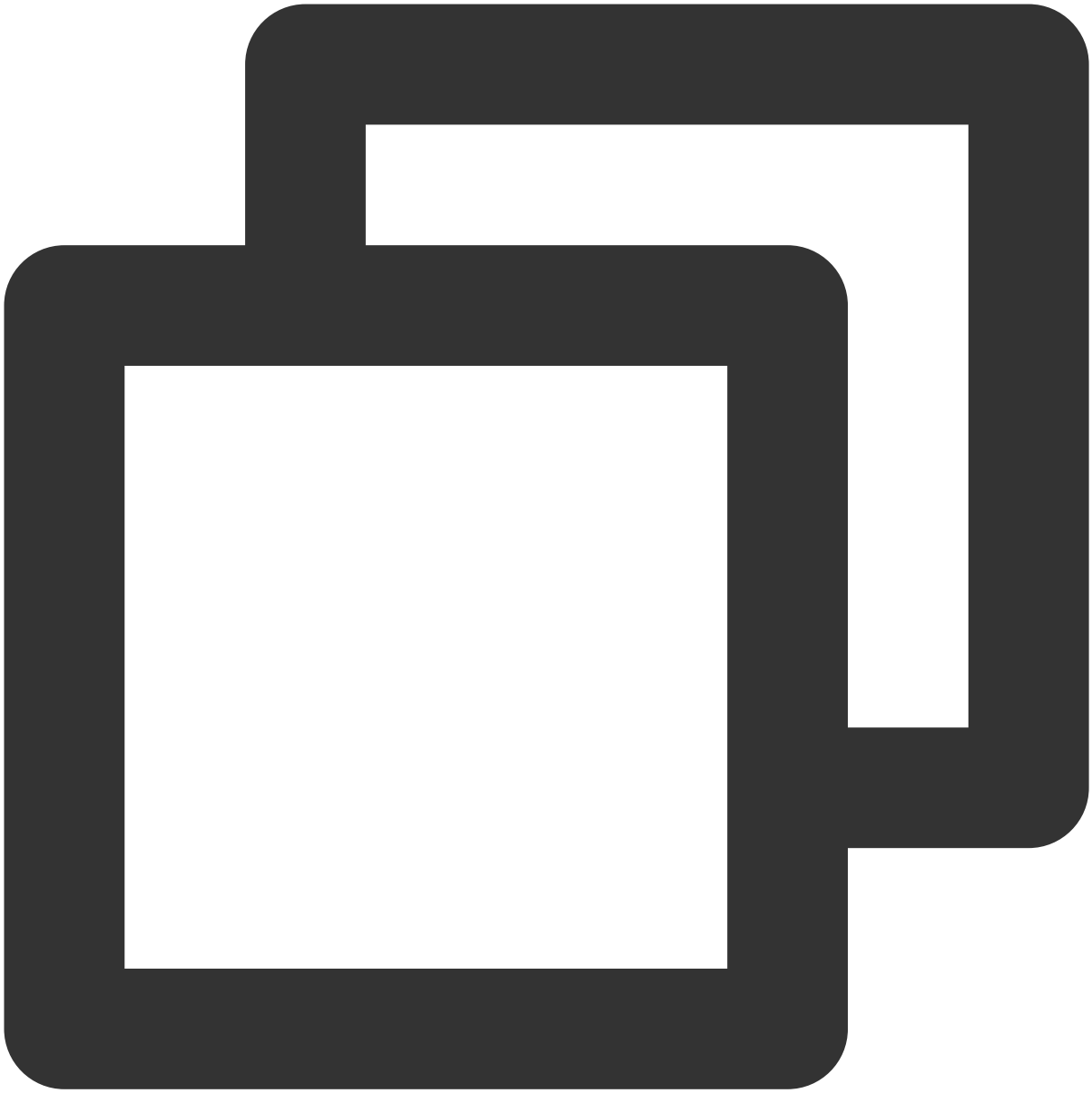


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->SetAccompanyFileCurrentPlayedTimeBy
```

Setting the accompaniment key

This API (SetAccompanyKey) is used to specify the accompaniment key, and should be called before starting the accompaniment playback.

Function prototype



```
ITMGAudioEffectCtrl virtual int SetAccompanyKey(int nKey)
```

Parameter	Type	Description
nKey	int	Key(s) to adjust by. Value range (recommended): -4 to 4, where `0` indicates using the original key.

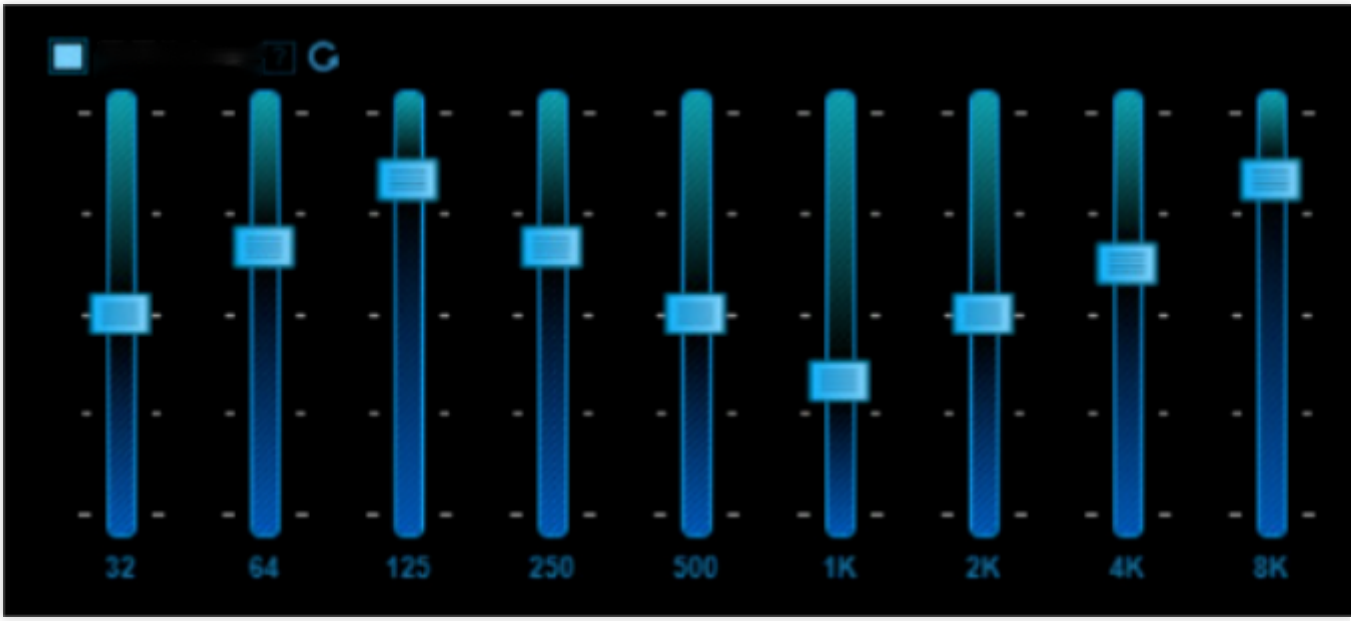
Error Codes

Error Message	Error Code	Description	Solution
QAV_ERR_ACC_OPENFILE_FAILED	4001	Failed to open the file	Checks whether the file or its path exists, and whether you have access to the file.
QAV_ERR_ACC_FILE_FORAMT_NOTSUPPORT	4002	Invalid file format	Checks whether the file format is correct.
QAV_ERR_ACC_DECODER_FAILED	4003	Decoding failure	Checks whether the file format is correct.
QAV_ERR_ACC_BAD_PARAM	4004	Invalid parameter	Checks whether the parameters in the code are correct.
QAV_ERR_ACC_MEMORY_ALLOC_FAILED	4005	Memory allocation failed	System resources have run out. If this error persists, please submit a ticket for assistance.
QAV_ERR_ACC_CREATE_THREAD_FAILED	4006	Failed to create a thread	System resources have run out. If this error persists, please submit a ticket for assistance.
QAV_ERR_ACC_STATE_ILLIGAL	4007	Invalid state	This error occurs if an API is called in a state that does not allow calling.

Real-Time Sound Equalizer

Last updated：2024-01-18 14:27:26

This document describes the GME APIs for equalizer in voice chat so that developers can easily integrate and debug them.



Overview

The GME equalizer feature can adjust the equalizer of the audio stream captured by the GME SDK in real time. This feature can be applied to the online karaoke scenario. After the player starts singing, call the **equalizer** API of the **GME SDK** to adjust the player's real-time audio stream for voice beautification.

Prerequisites

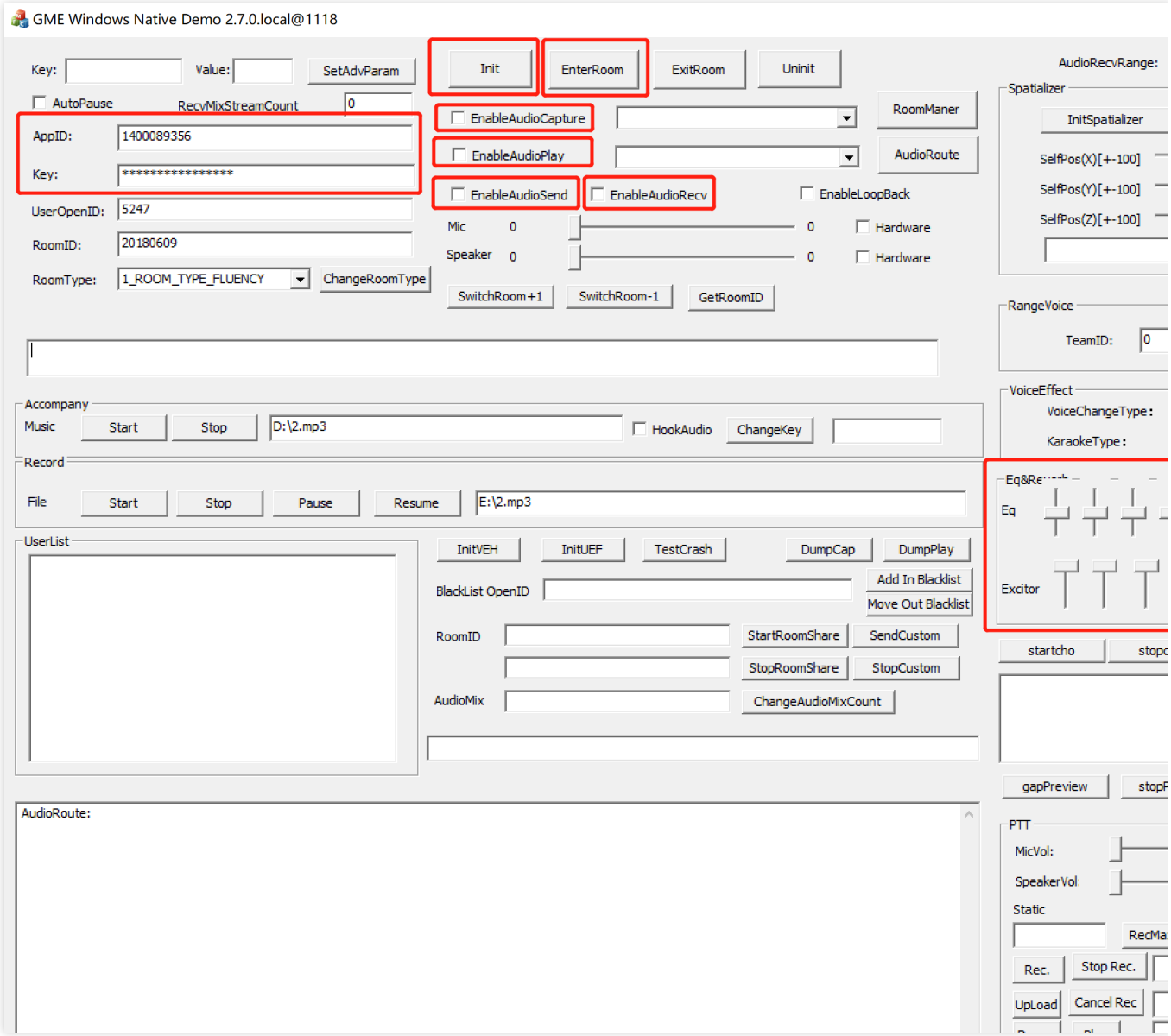
You have activated the voice chat service. For more information, see Activating Services.
You have accessed the GME SDK, including core APIs and voice chat APIs. For more information, see Quick Integration of Native SDK, Quick Integration of SDK for Unity, and Quick Integration of SDK for Unreal Engine.

Demos

Demo Download

Download address >>

This free demo is a Windows executable program.



Configuring parameters

You can enter your GME AppID and Key in the input box.

You can also enter the targeted room ID and OpenID.

Directions

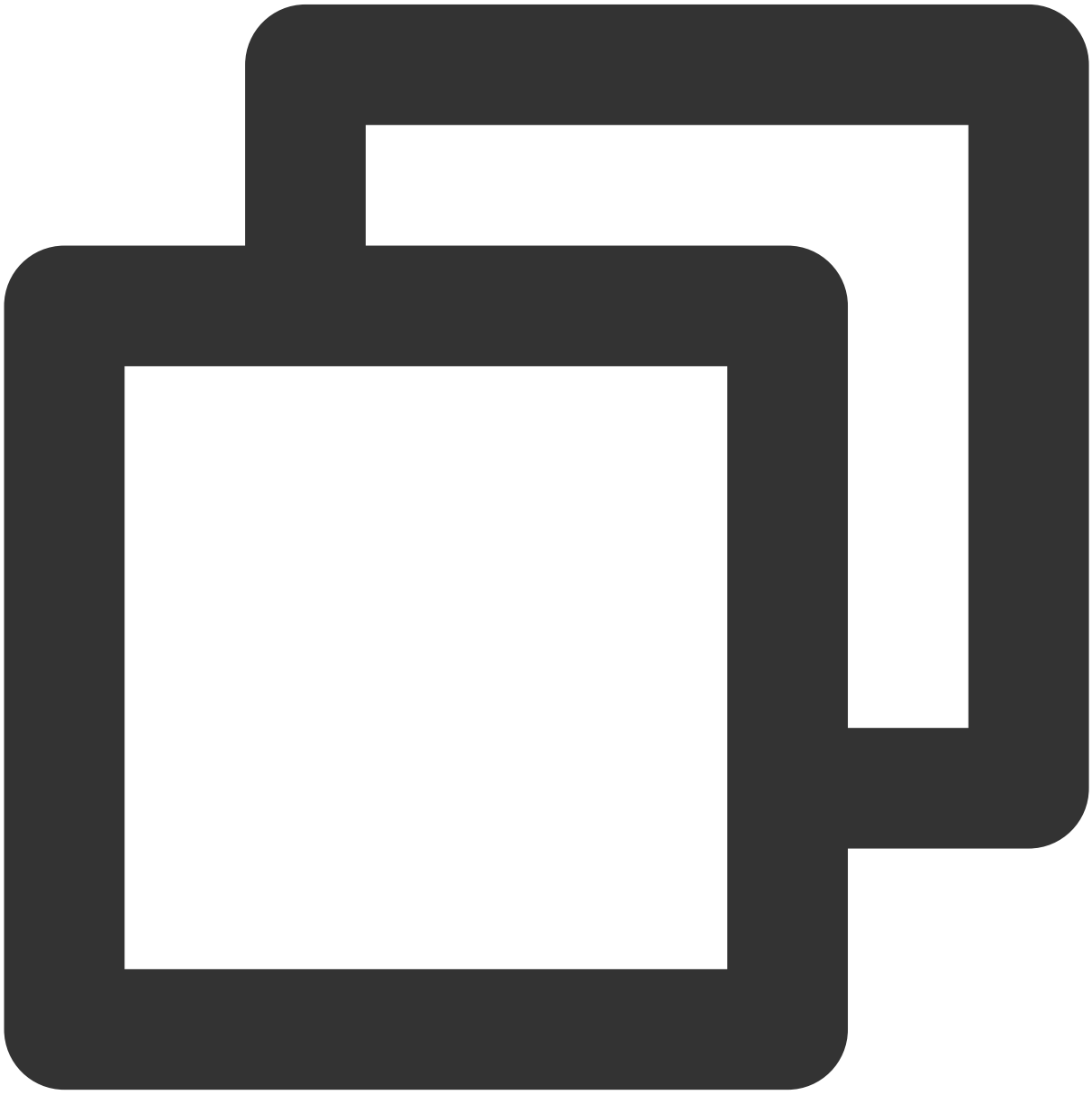
1. Follow the steps to open your mic and speaker: Init > EnterRoom > EnableCapture > EnablePlay > EnableSend > EnableRecv.

2. After successfully entering the room, you can hear your voice by enabling **EnableLoopBack**.
3. Adjust **EQ**

Integrating the Equalizer Feature

Only after successfully entering the room can the API use the equalizer on the sound captured on local.

Function prototype



```
int SetKaraokeType(ITMG_VOICE_TYPE_EQUALIZER* pEqualizer, ITMG_VOICE_TYPE_REVERB* p
```

Parameter	Type	Description
pEqualizer	ITMG_VOICE_TYPE_EQUALIZER	Frequency band gain
pReverb	ITMG_VOICE_TYPE_REVERB	Cover HARMONIC and REVERB

Structure details

The structure member in ITMG_VOICE_TYPE_EQUALIZER is of type float. Value range: [-12,12].

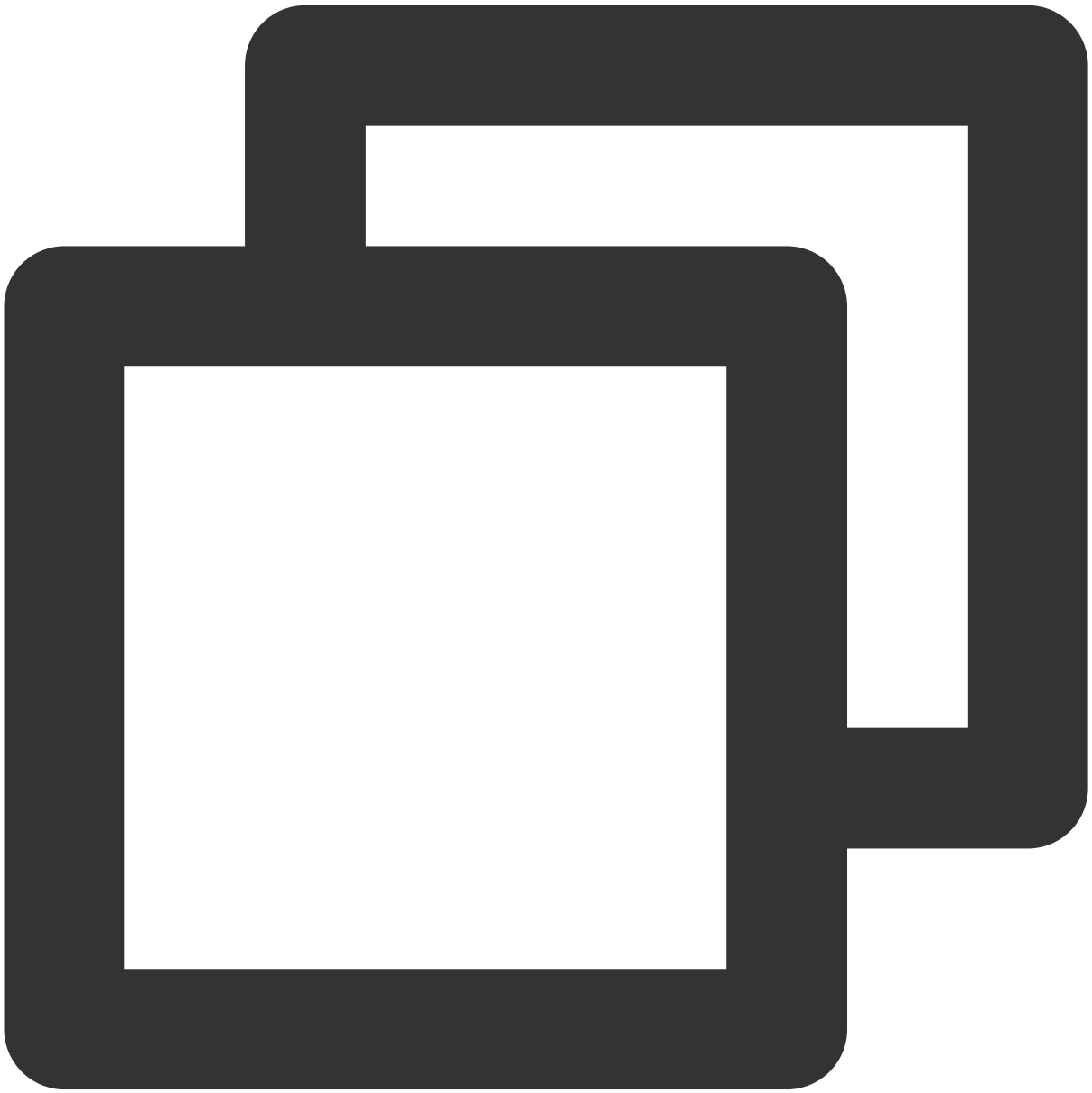
ITMG_VOICE_TYPE_EQUALIZER	Description
EQUALIZER_32HZ	Gain applied on the 32HZ band
EQUALIZER_64HZ	Gain applied on the 64HZ band
EQUALIZER_128HZ	Gain applied on the 128HZ band
EQUALIZER_250HZ	Gain applied on the 250HZ band
EQUALIZER_500HZ	Gain applied on the 500HZ band
EQUALIZER_1KHZ	Gain applied on the 1KHZ band
EQUALIZER_2KHZ	Gain applied on the 2KHZ band
EQUALIZER_4KHZ	Gain applied on the 4KHZ band
EQUALIZER_8KHZ	Gain applied on the 8KHZ band
EQUALIZER_16KHZ	Gain applied on the 16KHZ band
EQUALIZER_MASTER_GAIN	Overall volume

The structure member in ITMG_VOICE_TYPE_REVERB is of type float. Value range: [0,1].

ITMG_VOICE_TYPE_REVERB
HARMONIC_GAIN
HARMONIC_START_FREQUENCY
HARMONIC_BASS_CONTROL
REVERB_SIZE

REVERB_DEPTH
REVERB_GAIN
REVERB_ECHO_DEPTH

Sample code



```
void CTMGSDK_For_AudioDlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBa
{
    if ((CWnd*)pScrollBar == (CWnd*)&m_SliderEQ1 ||
```

```
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ2 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ3 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ4 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ5 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ6 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ7 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ8 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ9 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ10 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ11 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderExGain ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderExStartFrequency ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderExBaseCtrl ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderReverbSize ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderReverbDepth ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderReverbGain ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderReverbEchoDepth
        )
    {
        ITMG_VOICE_TYPE_EQUALIZER equalizer = {
            (m_SliderEQ1.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ2.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ3.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ4.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ5.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ6.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ7.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ8.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ9.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ10.GetPos() - 50) * 24.0f / 100,
            (m_SliderEQ11.GetPos() - 50) * 24.0f / 100
        };

        ITMG_VOICE_TYPE_REVERB reverb = {
            (m_SliderExGain.GetPos())            * 1.0f / 100.0f,
            (m_SliderExStartFrequency.GetPos())  * 1.0f / 100.0f,
            (m_SliderExBaseCtrl.GetPos())        * 1.0f / 100.0f,
            (m_SliderReverbSize.GetPos())        * 1.0f / 100.0f,
            (m_SliderReverbDepth.GetPos())       * 1.0f / 100.0f,
            (m_SliderReverbGain.GetPos())        * 1.0f / 100.0f,
            (m_SliderReverbEchoDepth.GetPos())   * 1.0f / 100.0f
        };

        m_pTmgContext->GetAudioEffectCtrl()->SetKaraokeType(&equalizer, &reverb);

    }
    CDialogEx::OnVScroll(nSBCode, nPos, pScrollBar);
```

```
}
```

Equalizer Instructions

Note:

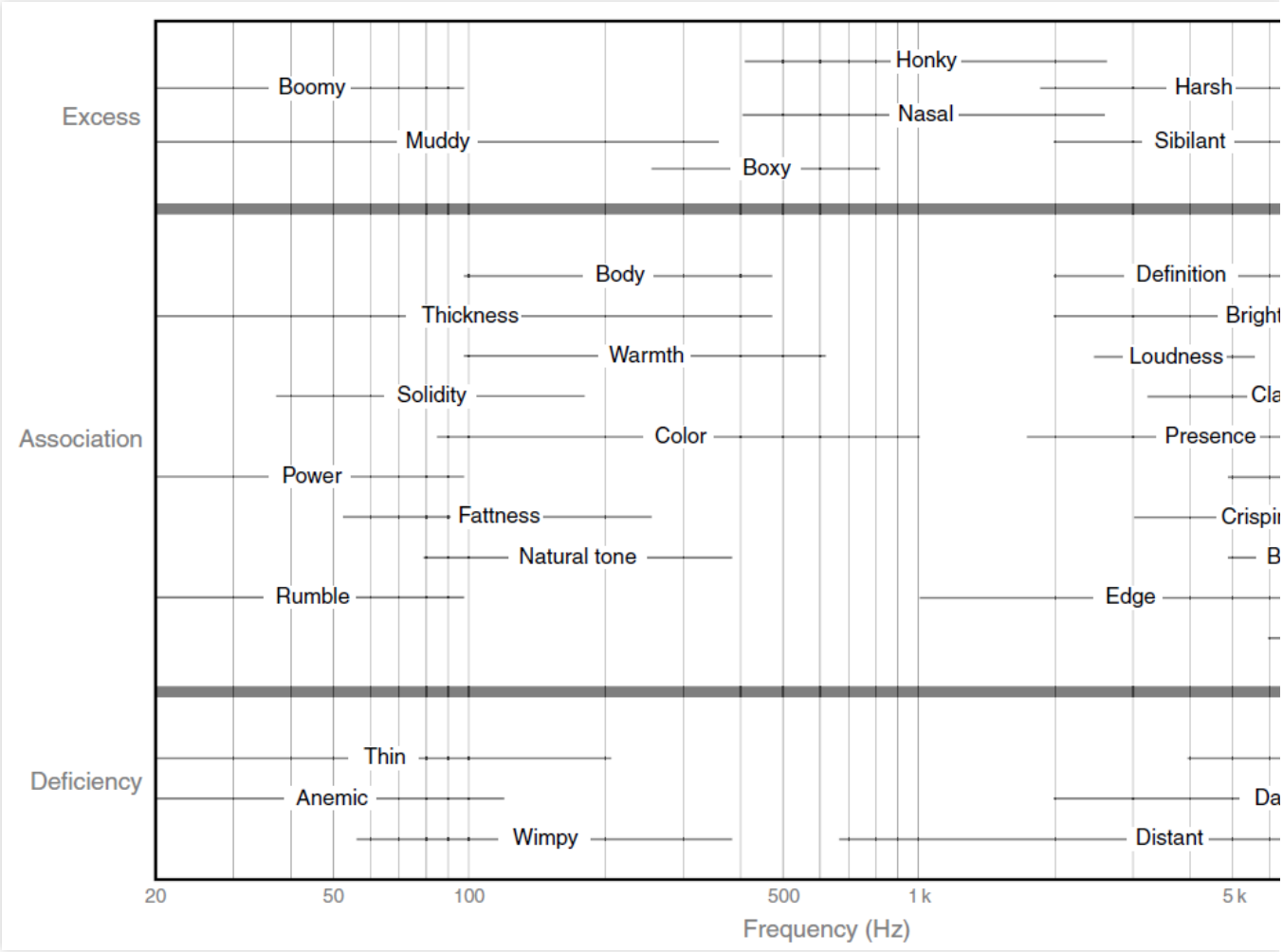
This document is for guidance only.

The sound range of human is roughly between 20 Hz and 20 kHz. The perception of each frequency band presents a logarithmic relationship, so the mixing project often divides the human audible frequency band into 10 octaves, which are divided into the following areas:

Band	Zone	Description
20 Hz – 32 Hz	Infrasound and subwoofer zone	Most of the frequency band in this zone is below the lower limit of the human hearing. Music in the super-large pipe organ and the film of the explosion and thunder sound effects can reach this frequency, while the human voice can not. Generally, VOIP calls are recommended to be tuned to z minimum to remove DC interference and leave the signal energy for other frequency bands.
32 Hz – 64 Hz	Heavy bass zone	It is mainly used to adjust the dive of the drums and bass. The tone is not obvious, which can be reached by part of the baritone. It is recommended to tune down the VoIP call to remove industrial frequency interference and leave the energy for other frequency bands.
64 Hz - 125 Hz	Bass zone	It is the range of the base clock of most orchestral instruments, which also determines the strength of the percussion.
125 Hz - 250 Hz	Bass zone	It is the range of the base clock of the human voice, which determines the perception of the human voice tone. If it is too heavy, it will lead to a slurred sound.
250 Hz—500 Hz	Mid-bass zone	It contains important lower harmonics of the tone in the frequency band, which can be used to adjust the tone and the enhancement of this section makes the tone warm and thick, while excessive enhancement makes it slurred.
500 Hz – 1 kHz	Midrange	It is used to adjust the female voice tone to make it fuller. Note that if it is too high, it will lead to a heavy nasal sound. Cell phones and other mobile playback devices resonance peak in this band, so please don’t make it too high.
1 kHz – 2 kHz	Midrange	It is a sensitive area of the human ear. The value affects the loudness and sense of presence, so it is recommended to enhance.

2 kHz-4 kHz	Soprano zone	It is the most sensitive area of human ear. The enhancement can increase the loudness of the voice and improve the intelligibility. Note that if it is too high, the sibilant could be heavy.
4 kHz – 8 kHz	Soprano zone	It can show the details of high frequency instruments, for example, cymbals and string instruments. Besides, it can decide details of voice, such as labiodental and fricative. Adjustment is not recommended.
8 kHz - 16 kHz	Super treble and super voice zone	It is the high frequency overtones area of the instrument. The adjustment has little effect on the voice.

For details, see:



Sound Effect and Accompaniment

Last updated：2023-04-27 17:06:32

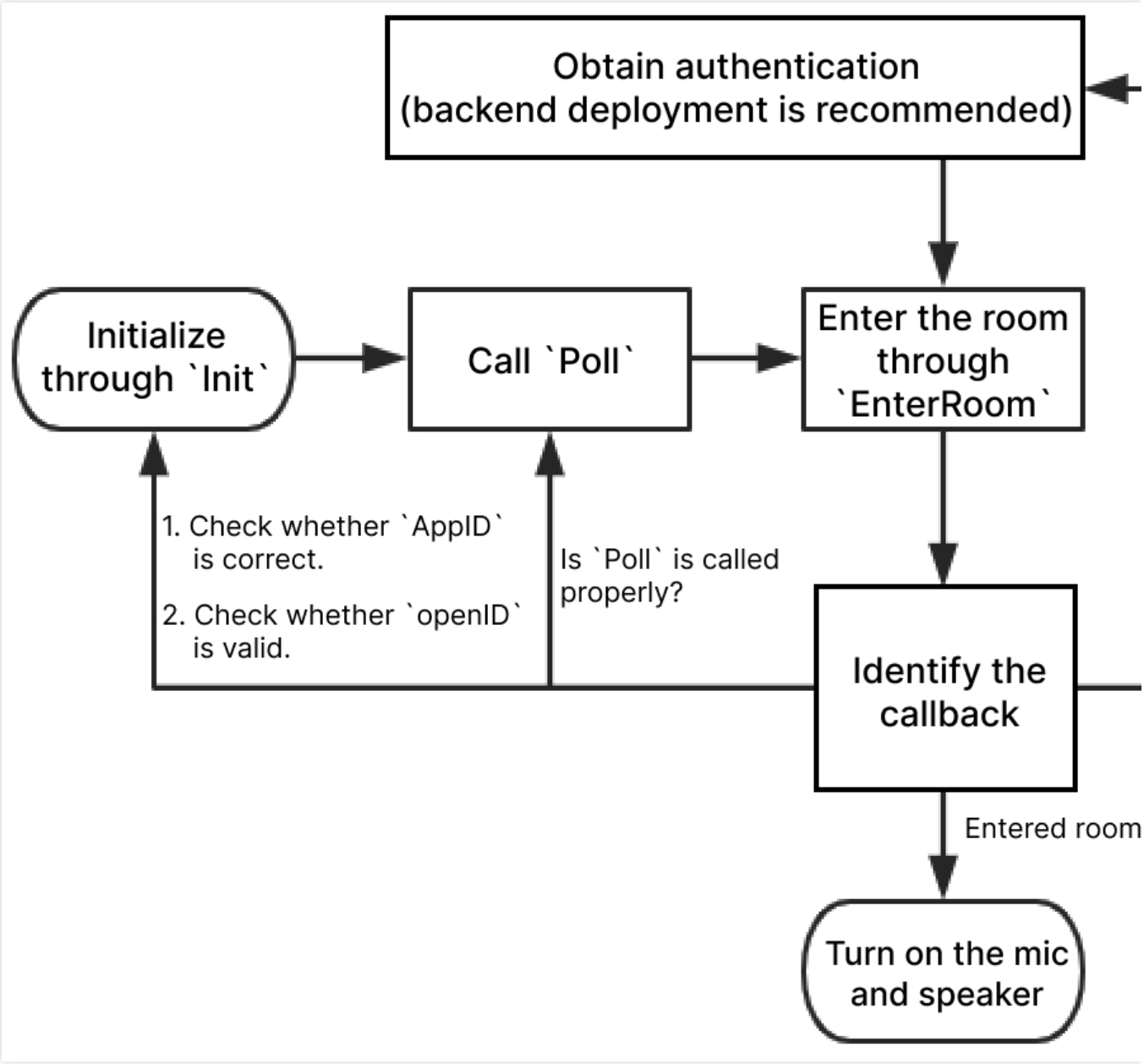
This document describes how to integrate with and debug GME APIs for the karaoke feature in voice chat.

Prerequisites

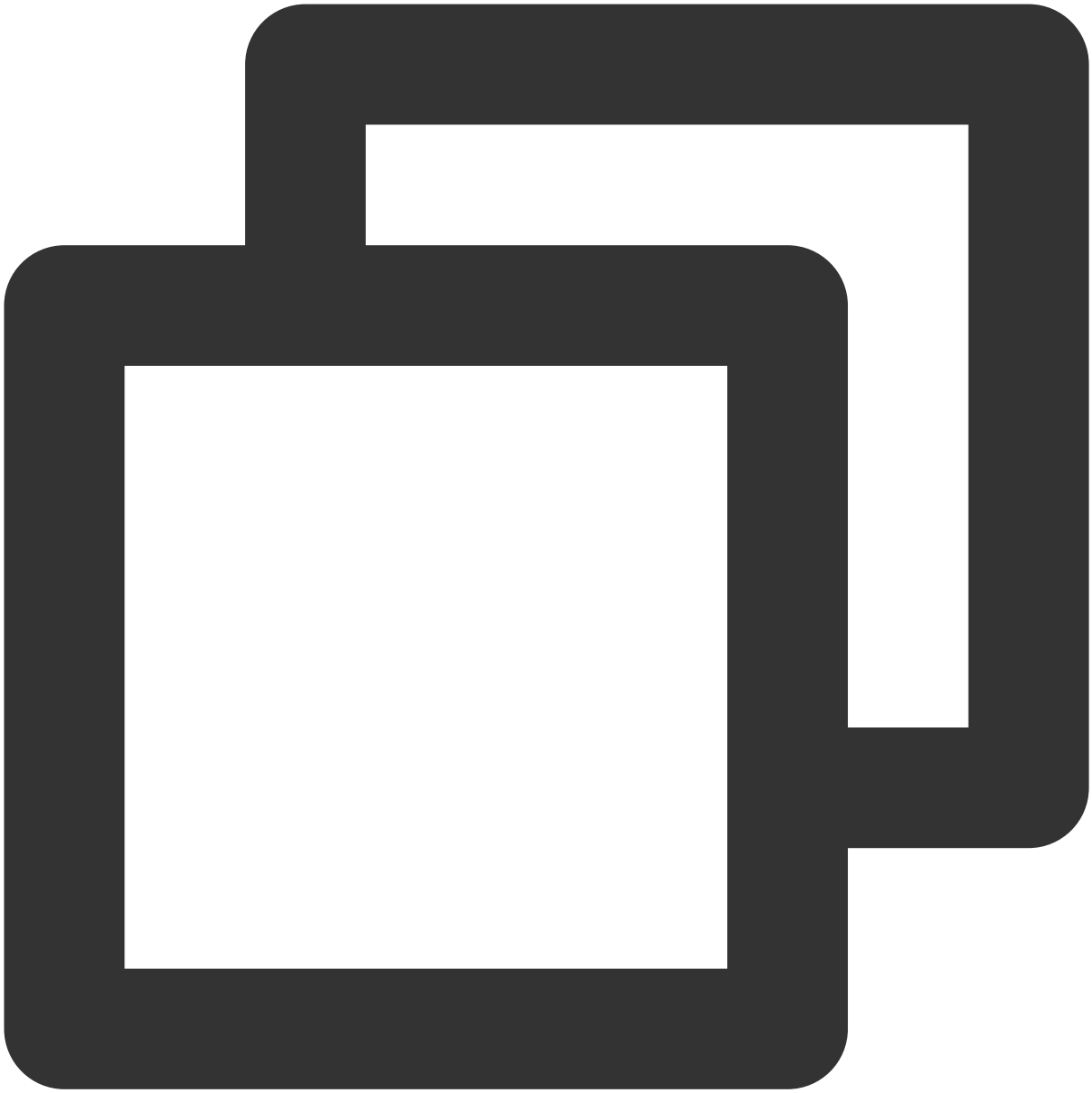
To use the karaoke feature in voice chat, you must integrate the GME SDK and enable voice chat.
The room type parameter needs to be specified during room entry. Set `RoomType` to `2` (recommended) or `3` for room entry.
If an error code is returned, troubleshoot as instructed in Error Codes.

Flowchart

The room entry process is as shown below:



Voice chat room entry APIs:



```
ITMGContext.GetInstance(this).Init(String.valueOf(mAppId), mUserId);// Initialize t
ITMGContext.GetInstance(this).SetTMGDelegate(new MyDelegate());// Set the delegate
EnginePollHelper.createEnginePollHelper();// Call `Poll` periodically to trigger ca

byte[] authbuff = AuthBuffer.getInstance().genAuthBuffer(mAppId, mRoomId, mUserId,m
ITMGContext.GetInstance(this).EnterRoom(mRoomId, 2, authbuff);// Enter a room
```

Note

For more information on the call process and APIs, see Game Multimedia Engine.

Getting Karaoke Feature APIs

1. Download the standard SDK file in SDK Download Guide.
2. Download and import the karaoke feature API file for your platform.

Note

This feature supports the MP3 and OGG formats.

For OGG audio files, click here to download the dynamic OGG library (you don't need to import the library for the SDK for iOS as it is provided in the SDK).

For MP3 audio files, click here to download the dynamic MP3 library (you need to import it only for the SDK for iOS).

Configuration for Android

APIs for Android are provided in the standard JAR package, so you don't need to download additional API files.

Configuration for iOS

1. To use the karaoke feature on iOS, you need to download the dynamic MP3 library and import it to your project.
2. Import the downloaded library and add it in **Link Binary With Libraries**.
3. Add the `TMGEngine_adv.h` header file to the same directory as the other SDK header files in your project.

Configuration for Windows

To use the karaoke feature on Windows, download header files `tmg_sdk_adv.h` and `tmg_type_adv.h` and import them to the project.

Configuration for the Unity engine

To use the karaoke feature in a game with the Unity engine, download and copy the header files `TMGEngine_Adv.cs` and `ITMGEngine_Adv.cs` in the `Unity` folder to the project to import them.

To export the project for iOS, import the dynamic MP3 library as described above.

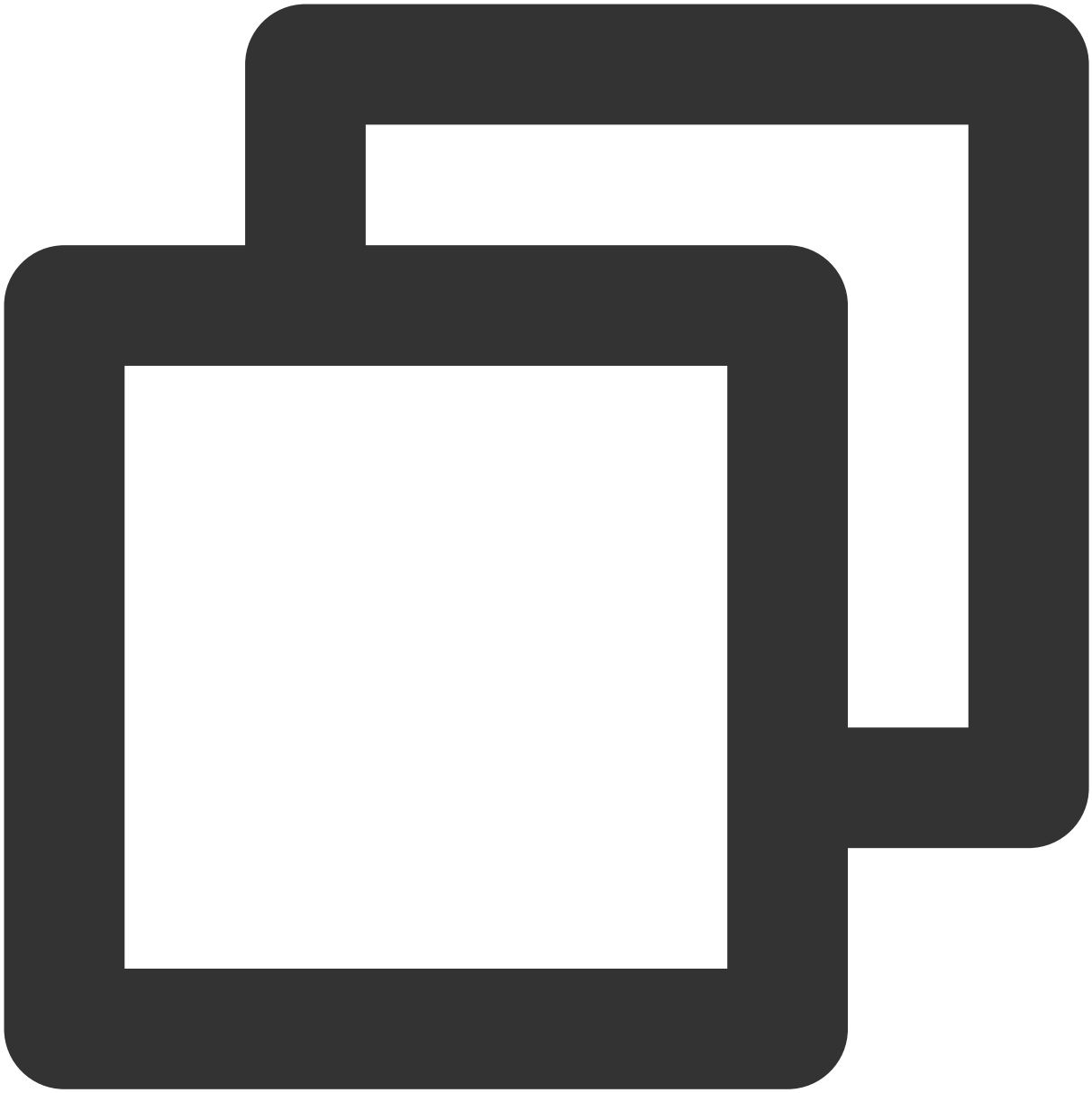
Recording APIs

Starting recording

The `StartRecord` API is used to start recording. There is a callback function for recording completion, and you need to listen for `ITMG_MAIN_EVENT_TYPE_RECORD_COMPLETED` .

Make sure that the mic is turned on during recording (both the device and upstreaming need to be enabled) and the file path is accessible, as the SDK doesn't create folders actively.

Function prototype

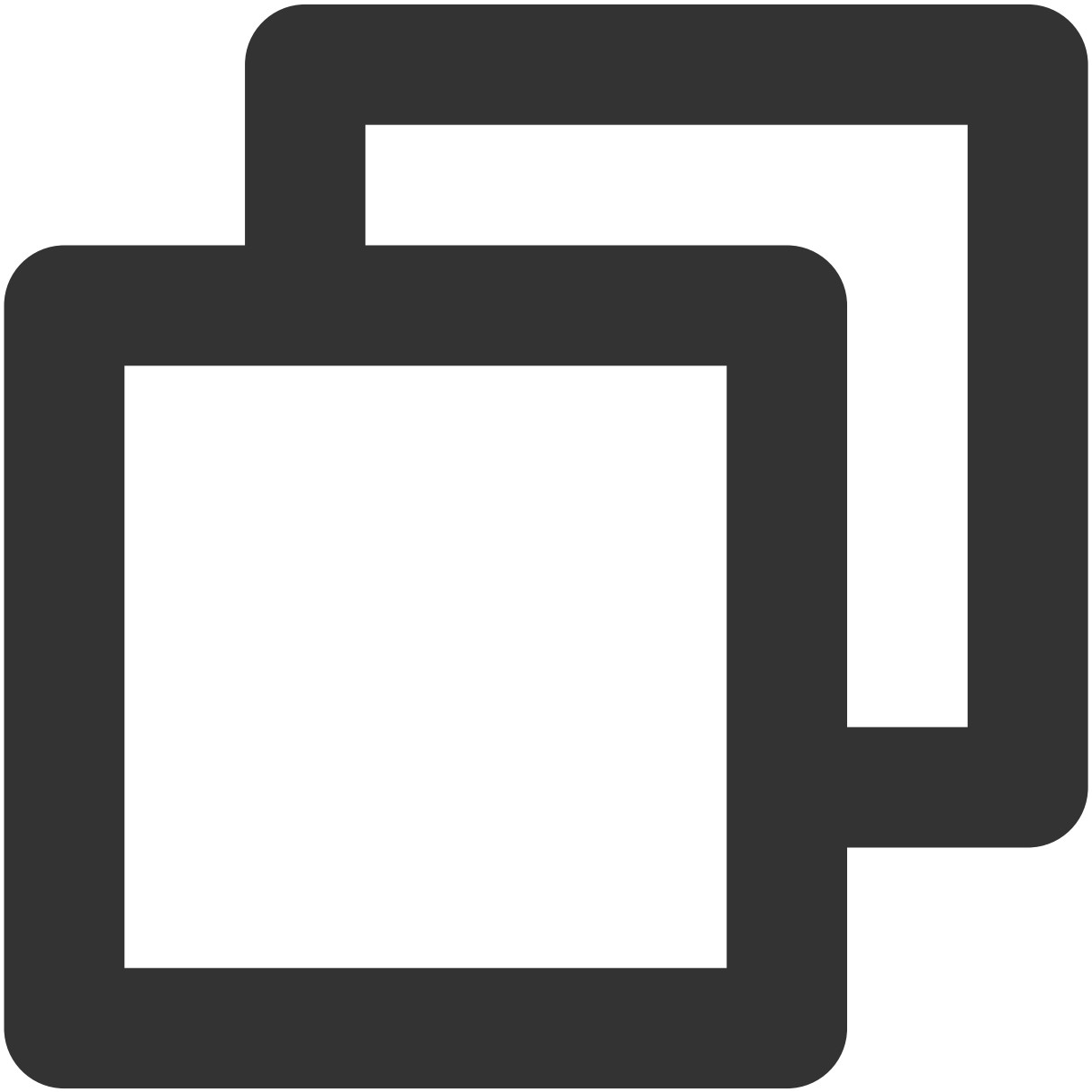


```
int StartRecord(int type, String dstFile, String accMixFile, String accPlayFile)
```

Parameter	Type	Description
type	int	Set this parameter to `ITMG_AUDIO_RECORDING_KTV` in a karaoke scenario or `ITMG_AUDIO_RECORDING_SELF` for an MP3 recording file.
dstFile	String	Target file path for saving the recorded music
accMixFile	String	Accompaniment without the original vocals, which can be mixed with voice to generate a music file.

accPlayFile	String	Music file to be played back, which is the same file as `accMixFile` in normal cases. However, if the user is unfamiliar with the song, this parameter can be set to the path of the music file with the original vocals. In this case, the file with the original vocals will be played back, but the accompaniment without the original vocals will be mixed.

Sample code



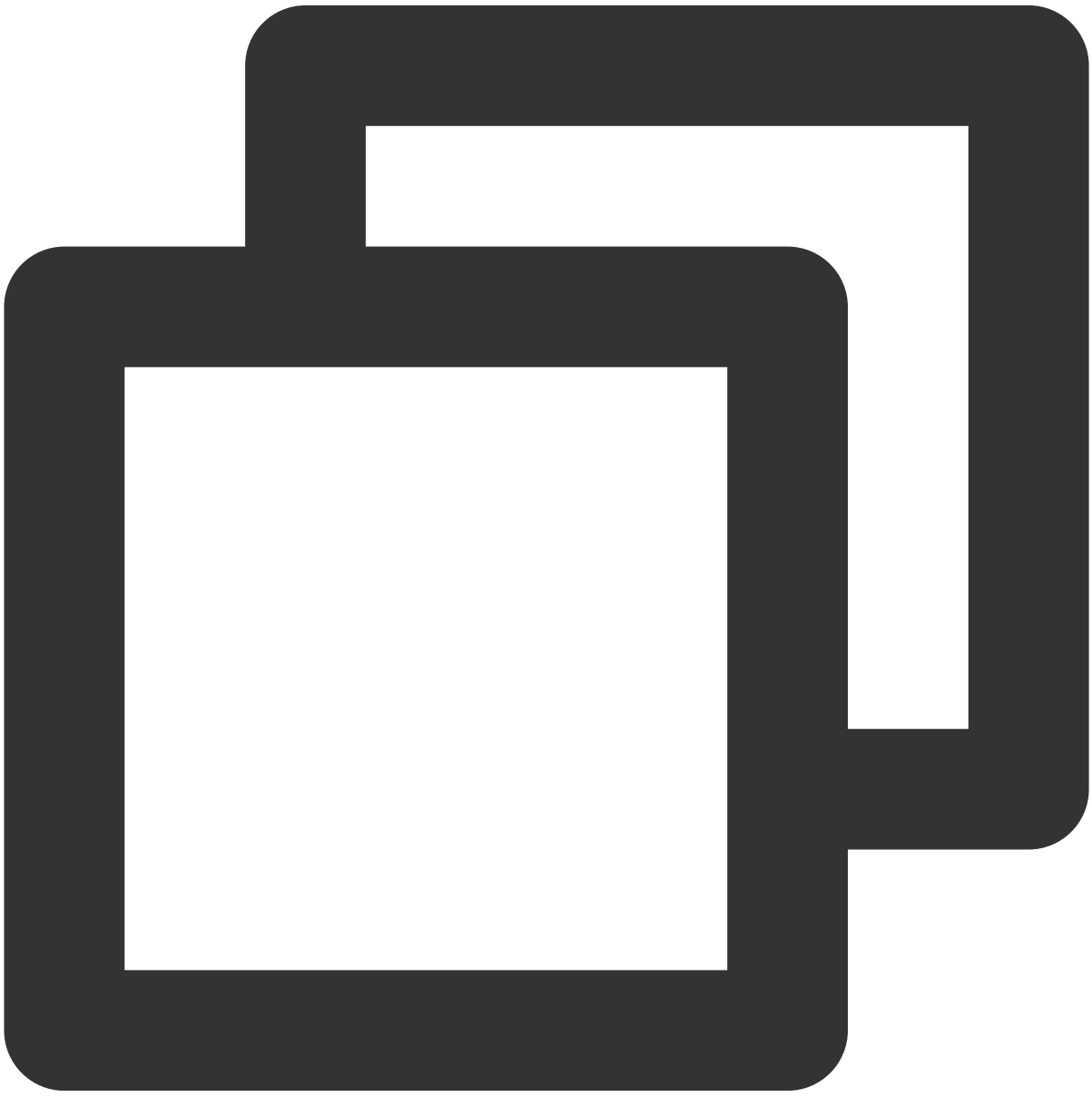
```
//Android
```

```
ITMGAudioRecordCtrl.GetInstance().StartRecord(ITMGAudioRecordCtrl.ITMG_AUDIO_RECORD
// iOS
#import "GMESDK/TMGEngine_adv.h"
[[ITMGAudioRecordCtrl GetInstance]StartPreview]
```

Stopping recording

The `StopRecord` API is used to stop recording.

Function prototype

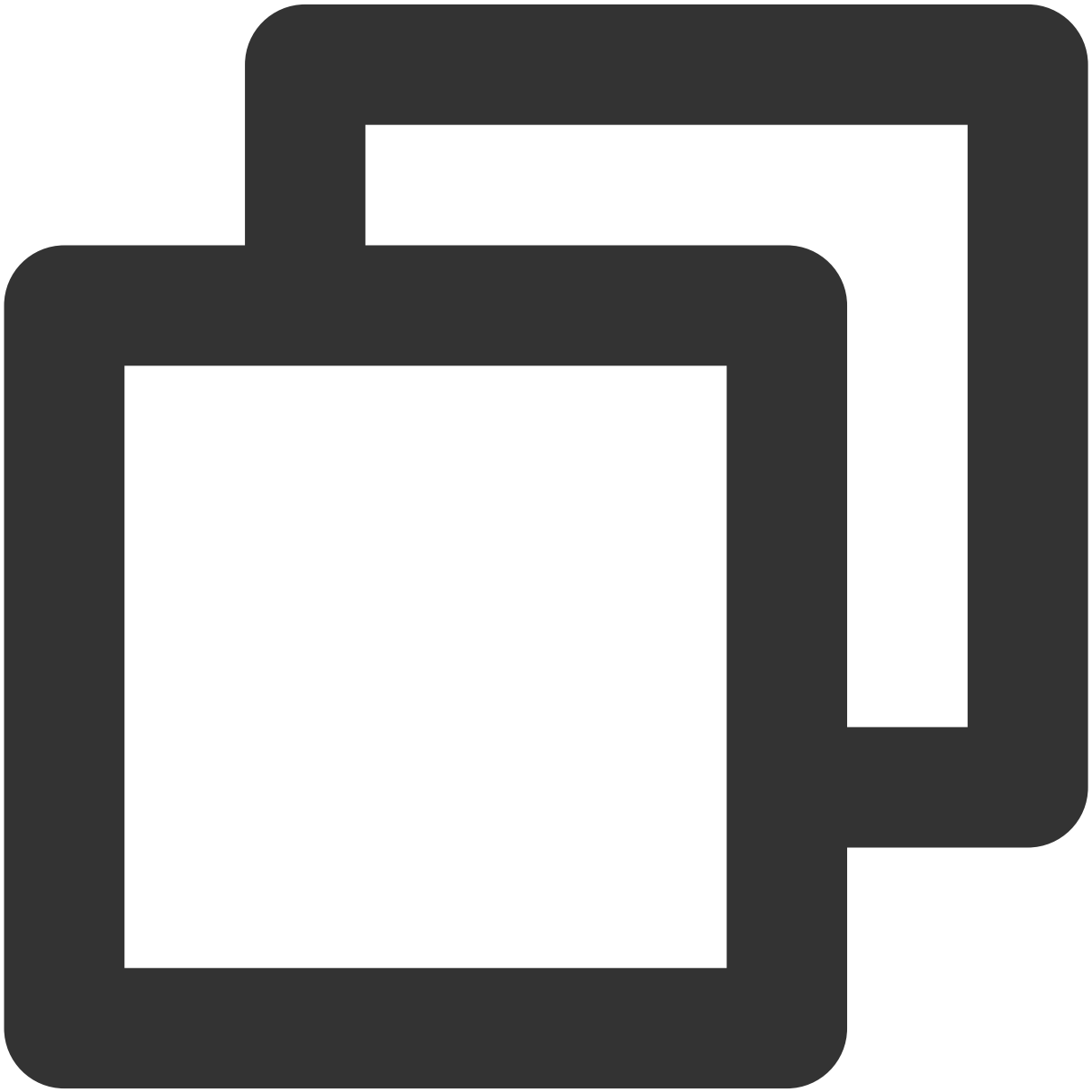


```
int StopRecord()
```

Pausing recording

The `PauseRecord` API is used to pause recording.

Function prototype

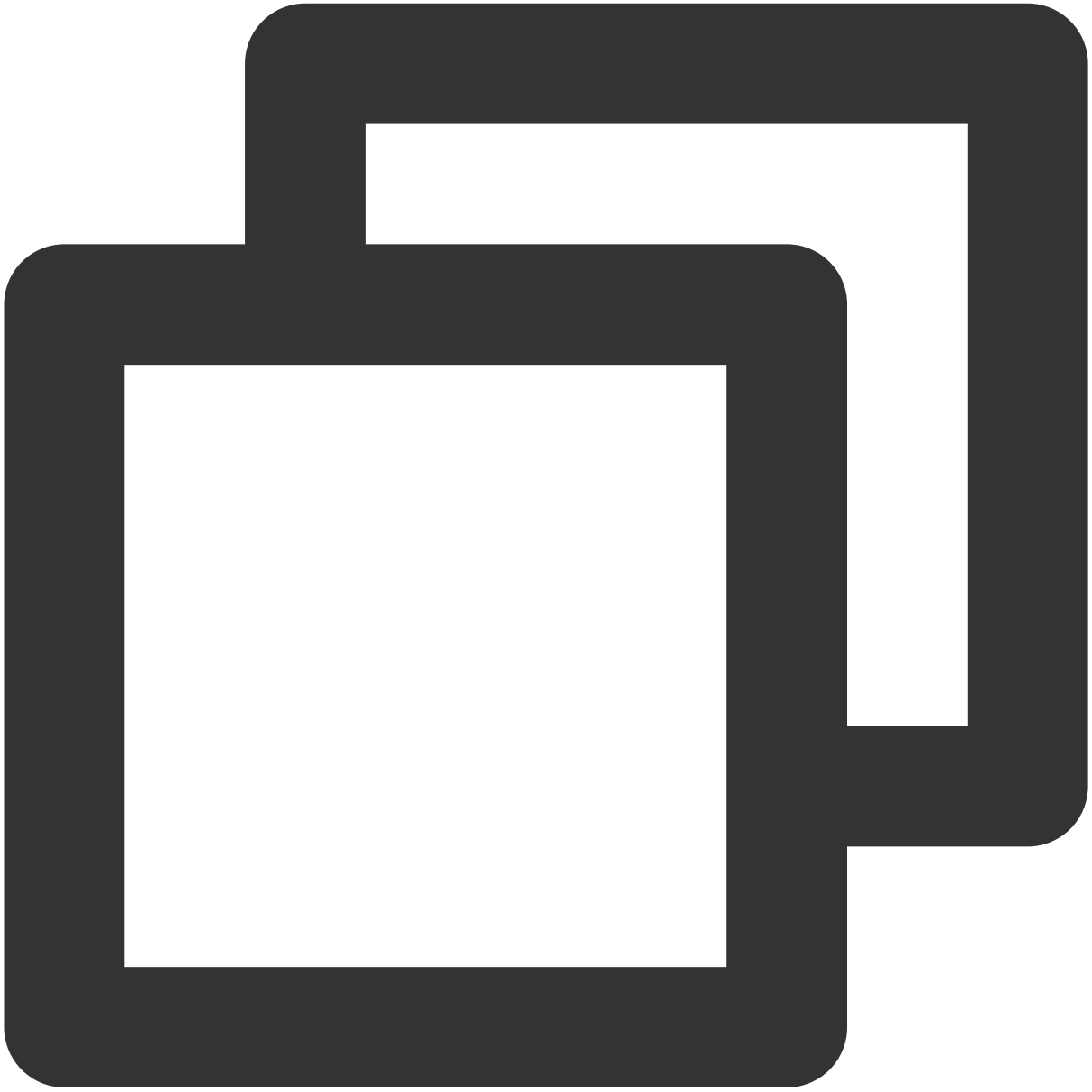


```
int PauseRecord()
```

Resuming recording

The `ResumeRecord` API is used to resume recording.

Function prototype



```
int ResumeRecord()
```

Recording callback

`ITMG_MAIN_EVENT_TYPE_RECORD_COMPLETED` is the callback for recording completion, which will be triggered when the accompaniment playback ends or `StopRecord` is called.

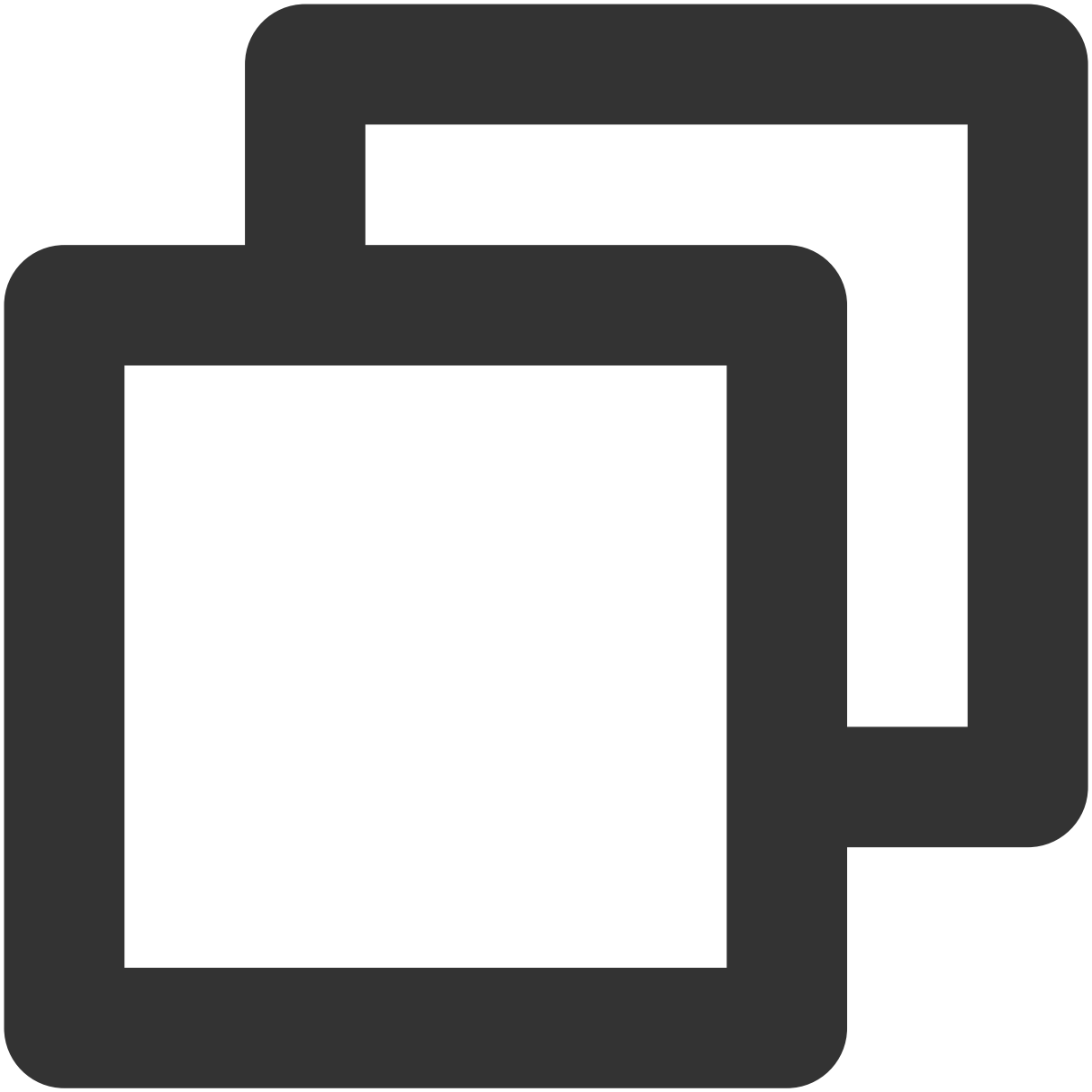
Callback parameters

Parameter	Type	Description
result	int	Recording result. `0` indicates success. For other error codes, troubleshoot as instructed in Error Codes.
filepath	String	Target file path, which is the same as the `dstFile` parameter passed in through `StartRecord`.
duration	String	Recording file duration in ms.

Setting the file to be played back

When the `StartRecord` API is called for recording, the music file to be played back will be set. This API can be used to set another file for playback. Generally, it is used to switch between the song with vocals and accompaniment without vocals.

Function prototype



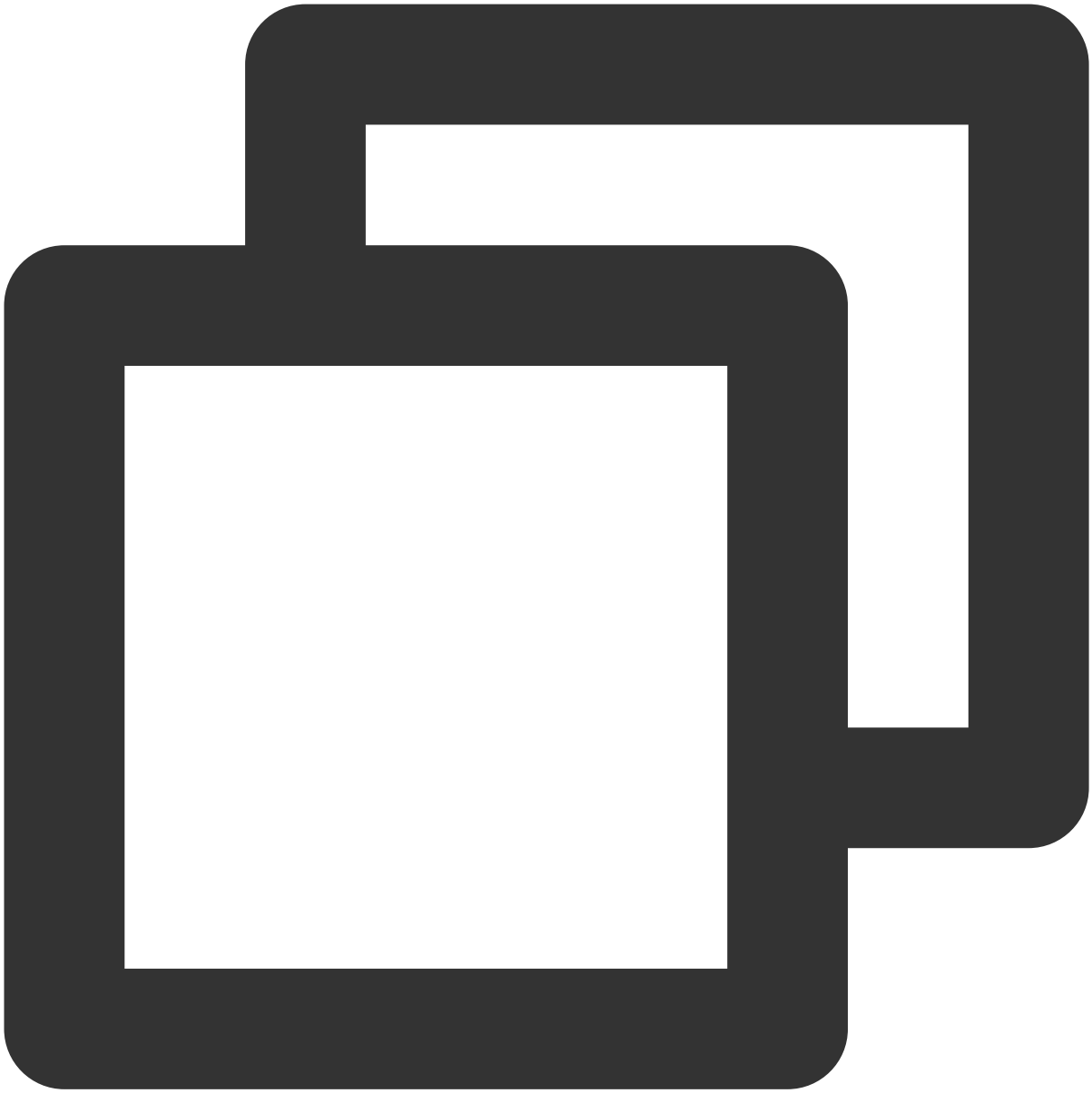
```
int SetAccompanyFile(String accPlayFile)
```

Parameter	Type	Description
accPlayFile	String	Music file to be played back

Getting the accompaniment duration

This API is used to get the duration of the accompaniment file `accMixFile` in ms.

Function prototype

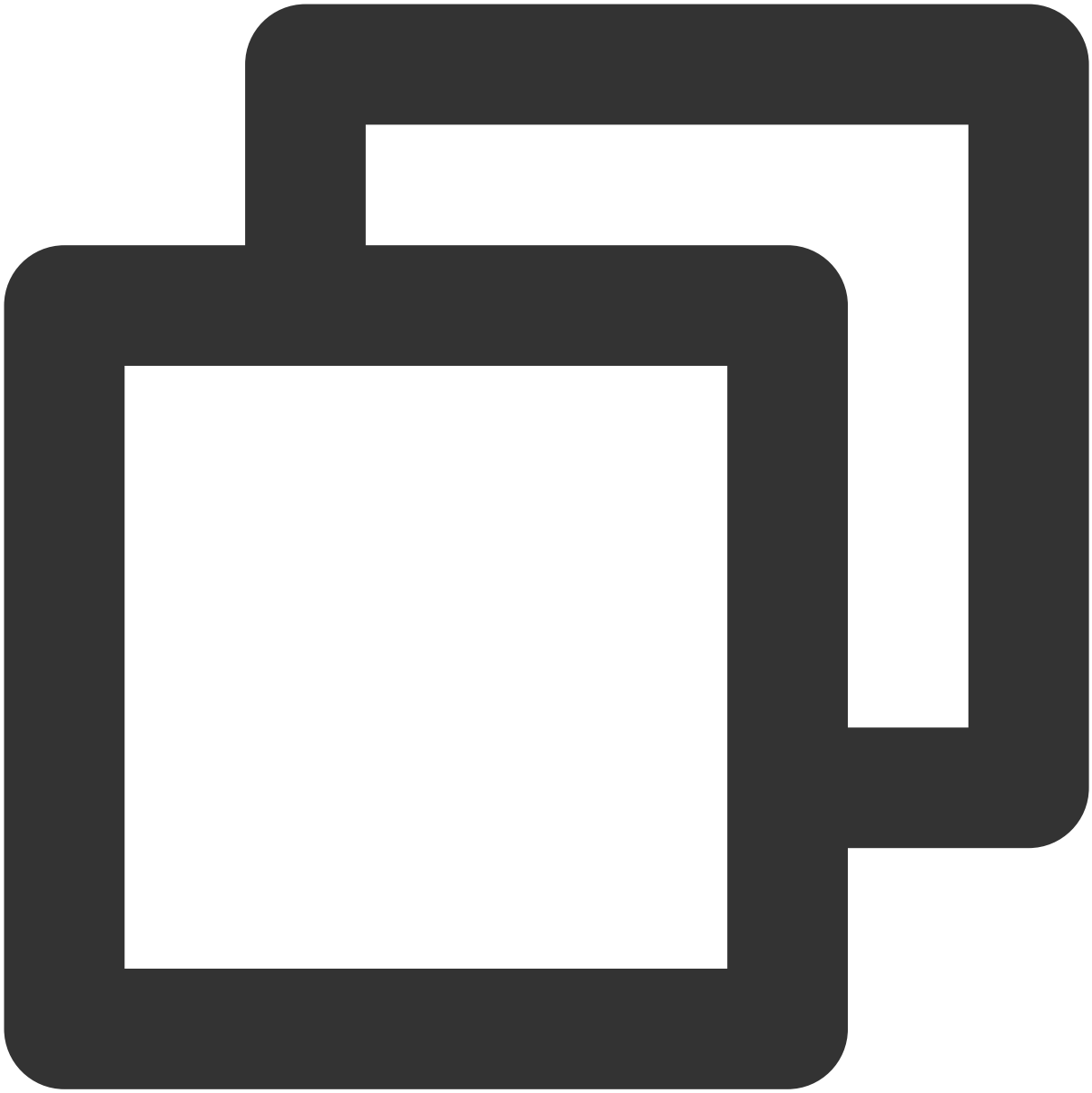


```
int GetAccompanyTotalTimeByMs()
```

Getting the current recording duration

This API is used to get the current recording duration in ms.

Function prototype

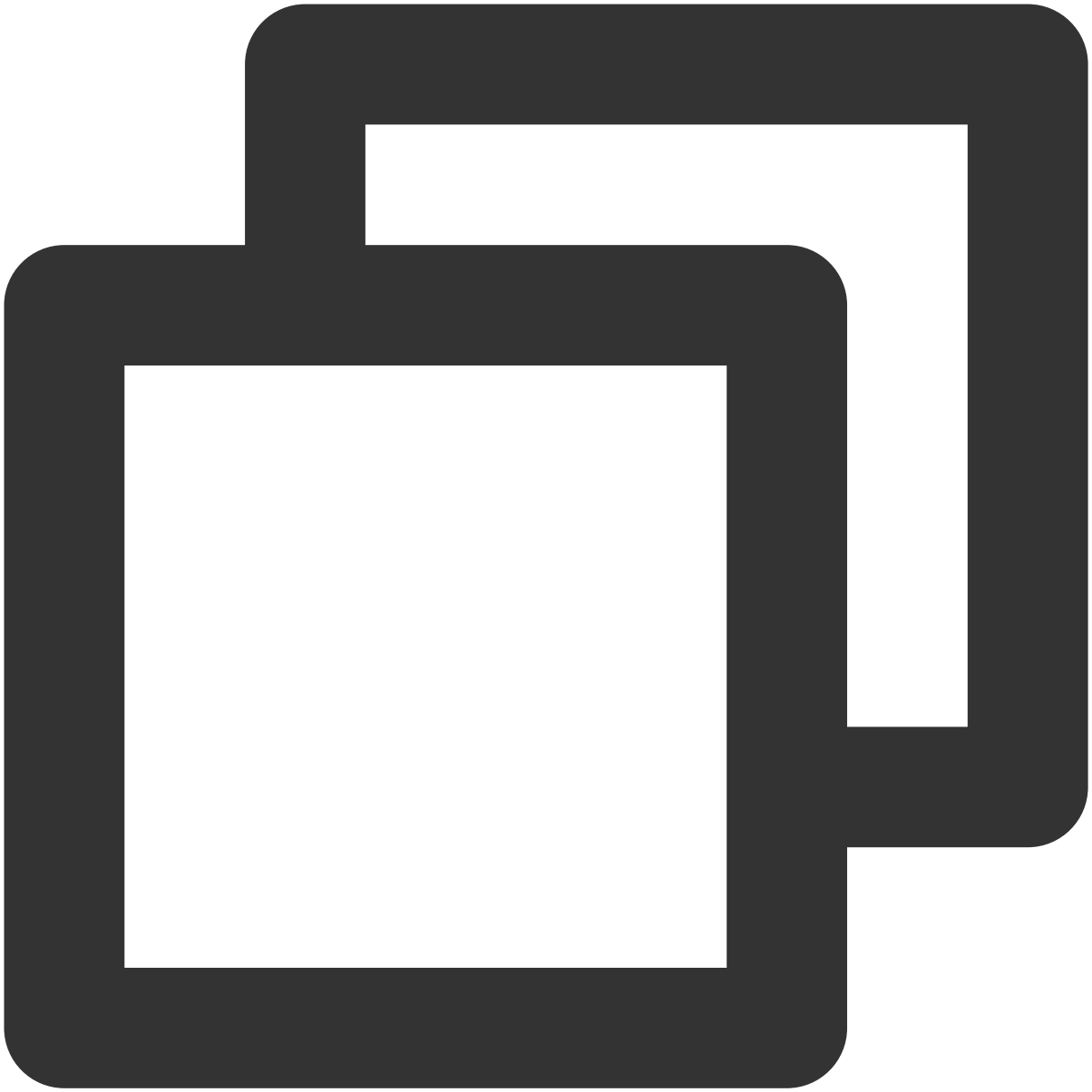


```
int GetRecordTimeByMs()
```

Seeking for recording

This API is used to seek to the specified time for recording. If the parameter value is earlier than the current time, the repeated part will be recorded again. If the parameter value is later than the current time, the unrecorded part will be filled with silence.

Function prototype



```
int SetRecordTimeByMs(int timeMs)
```

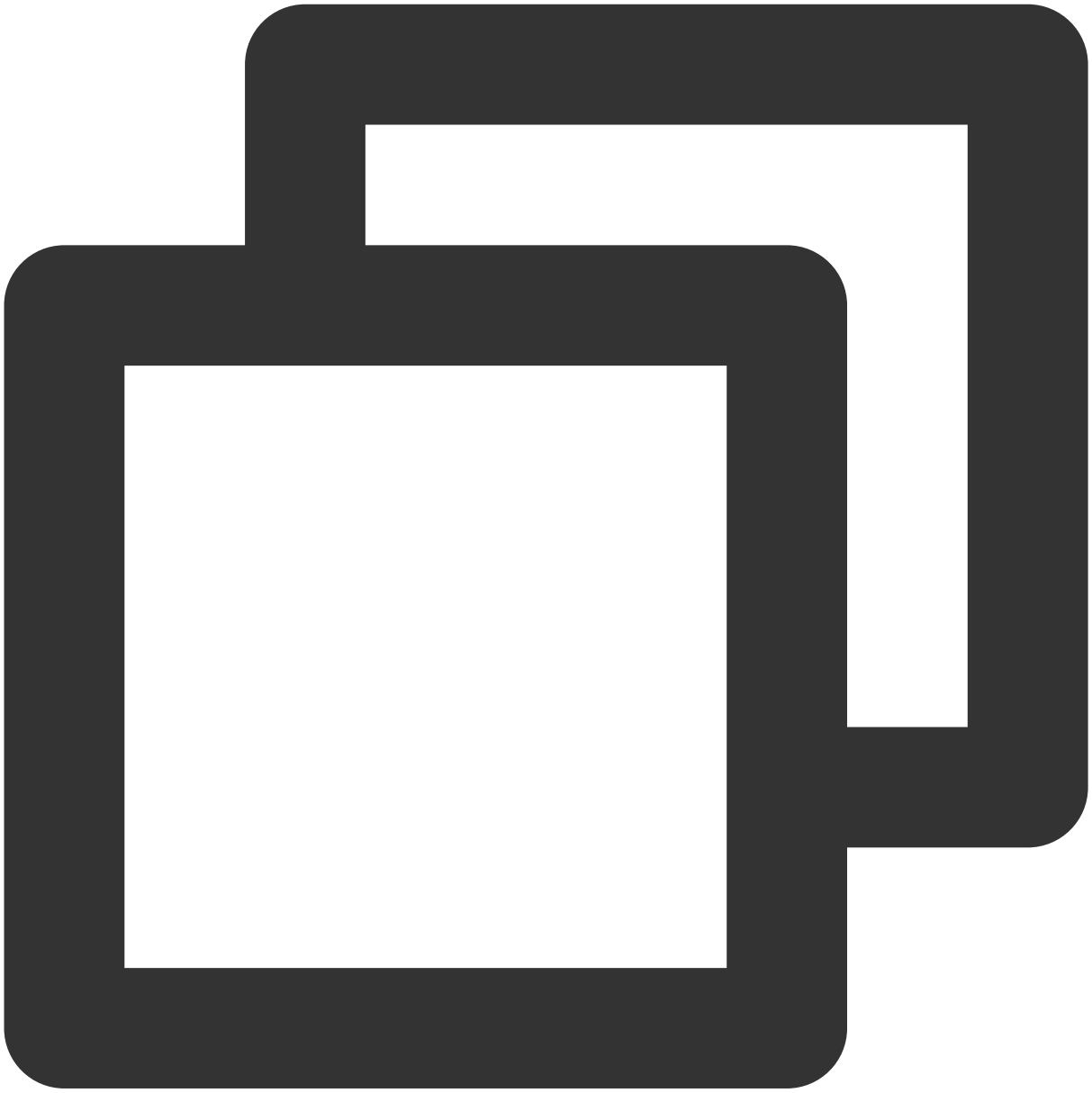
Parameter	Type	Description
timeMs	int	The time to be sought to in ms

Karaoke File Preview

Getting the recording file duration

This API is used to get the recording file duration.

Function prototype

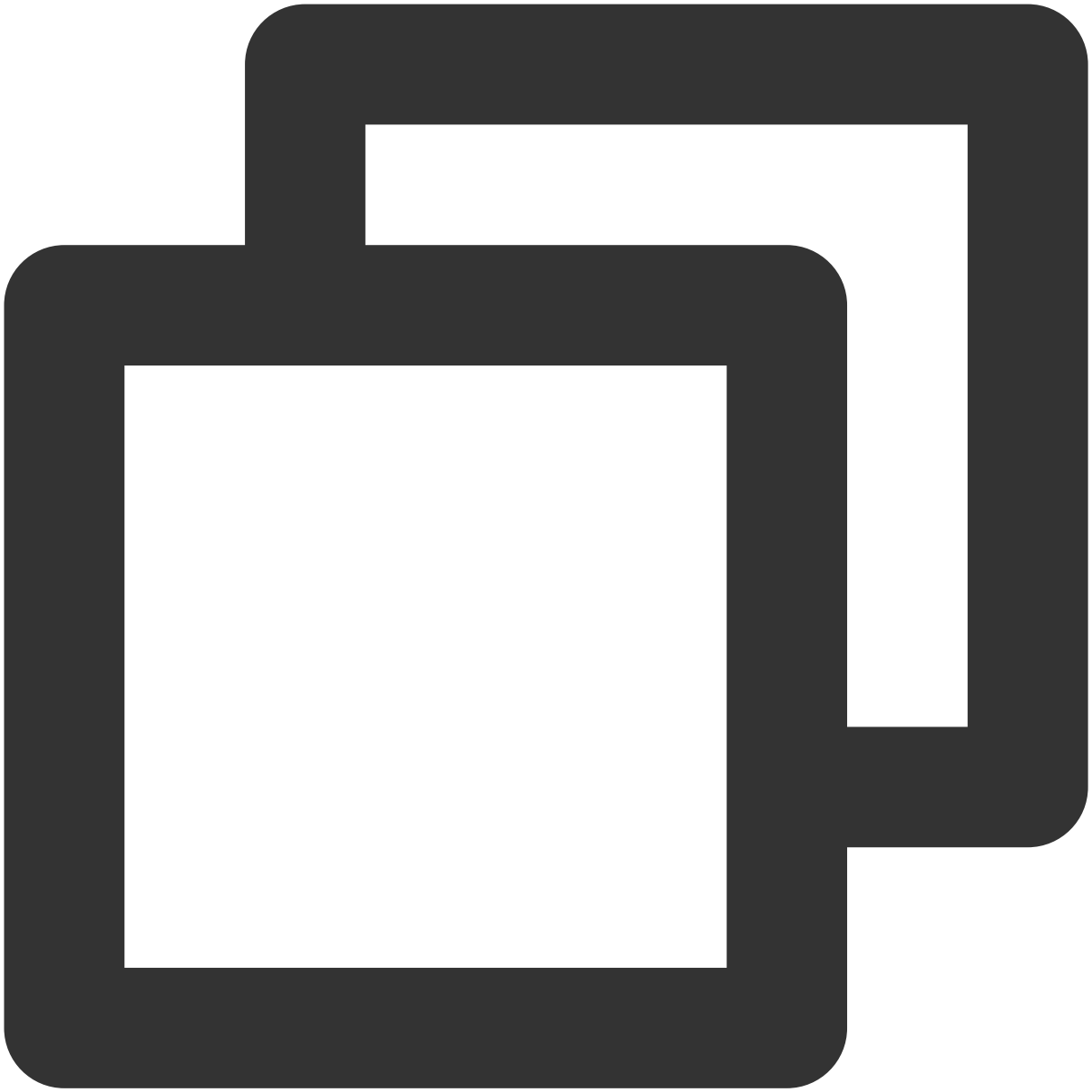


```
int GetRecordFileDurationByMs()
```

Starting recording file preview

This API is used to start recording file preview.

Function prototype

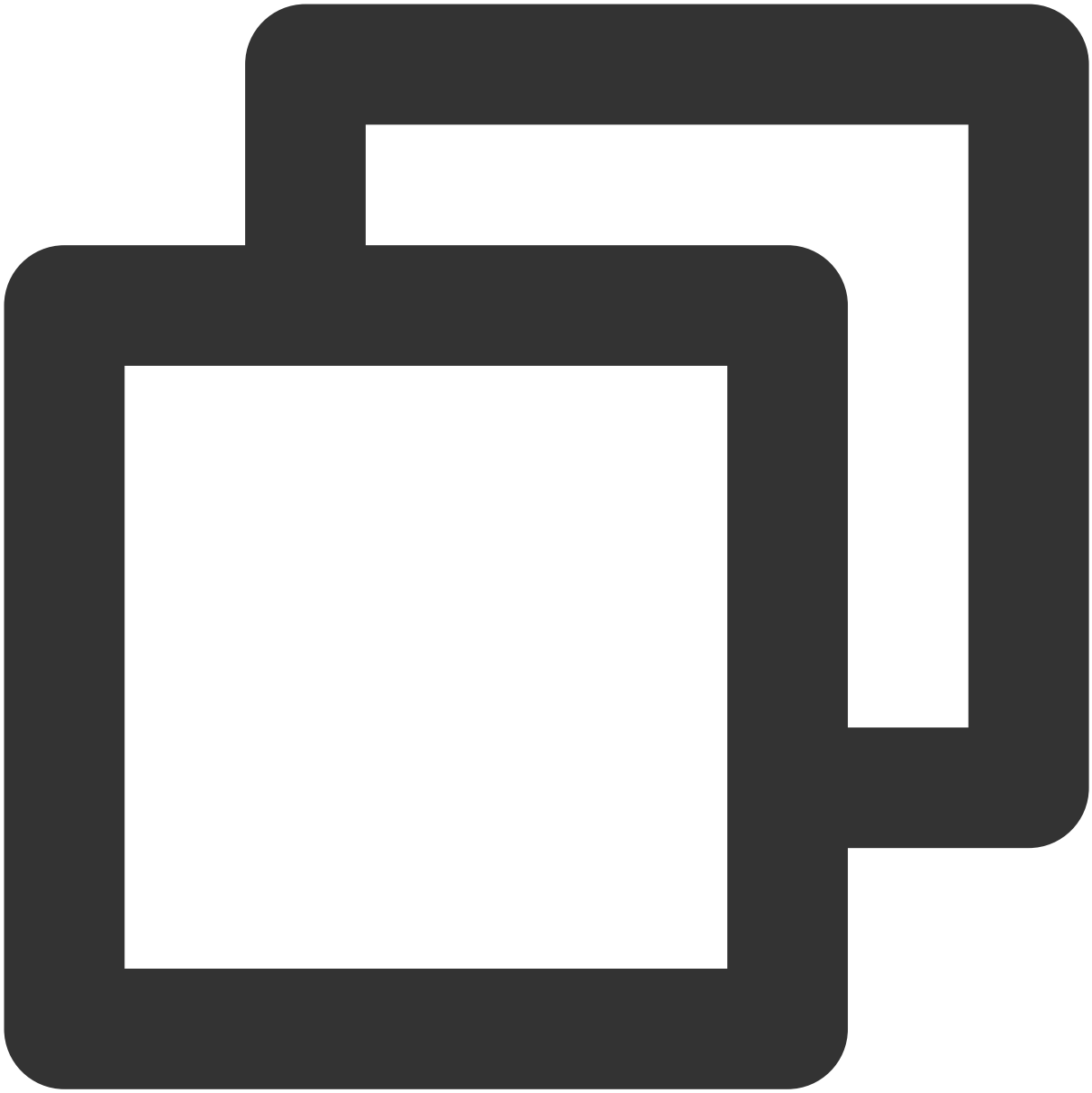


```
int StartPreview()
```

Stopping recording file preview

This API is used to stop recording file preview.

Function prototype

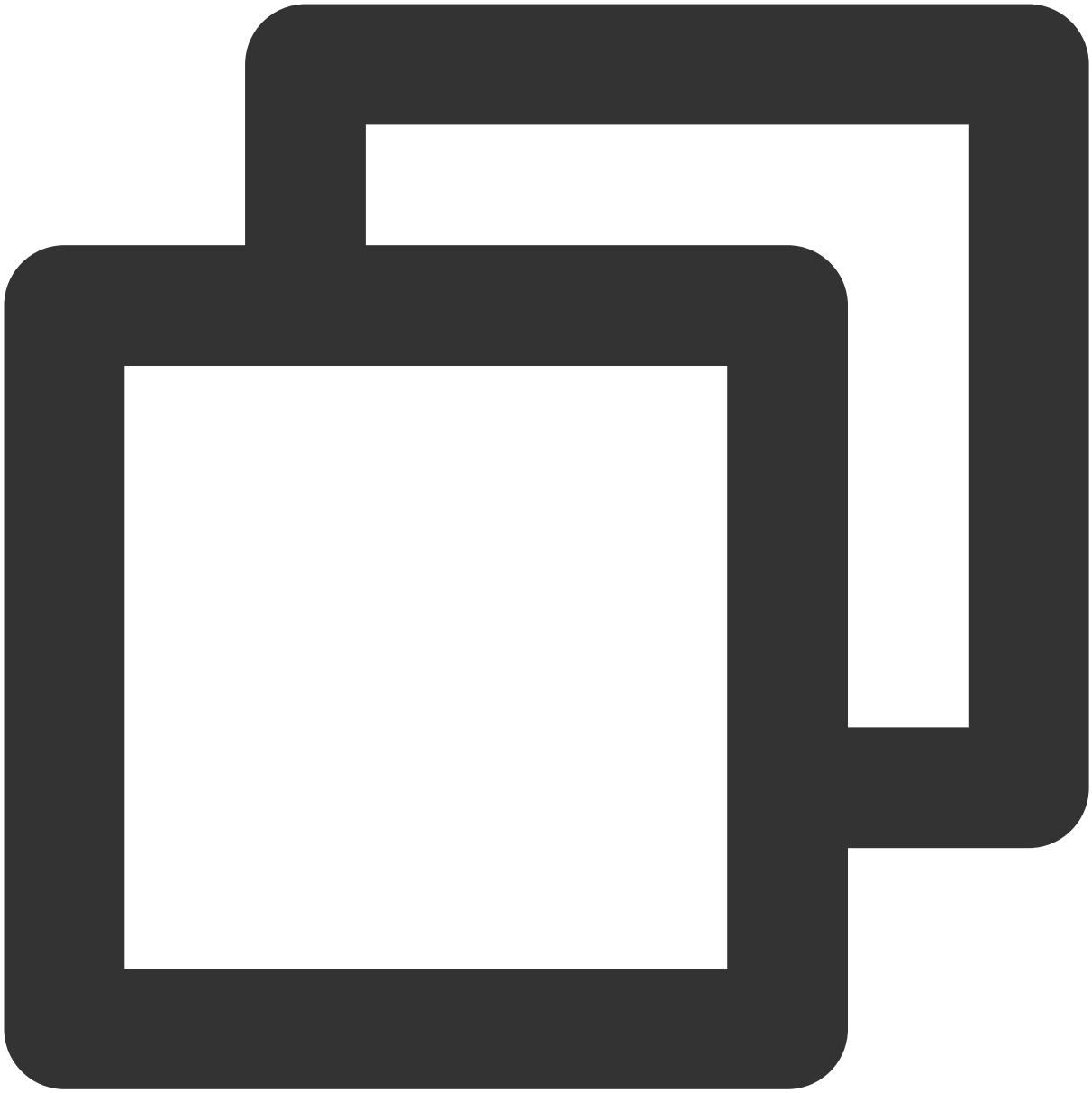


```
int StopPreview()
```

Pausing recording file preview

This API is used to pause recording file preview.

Function prototype

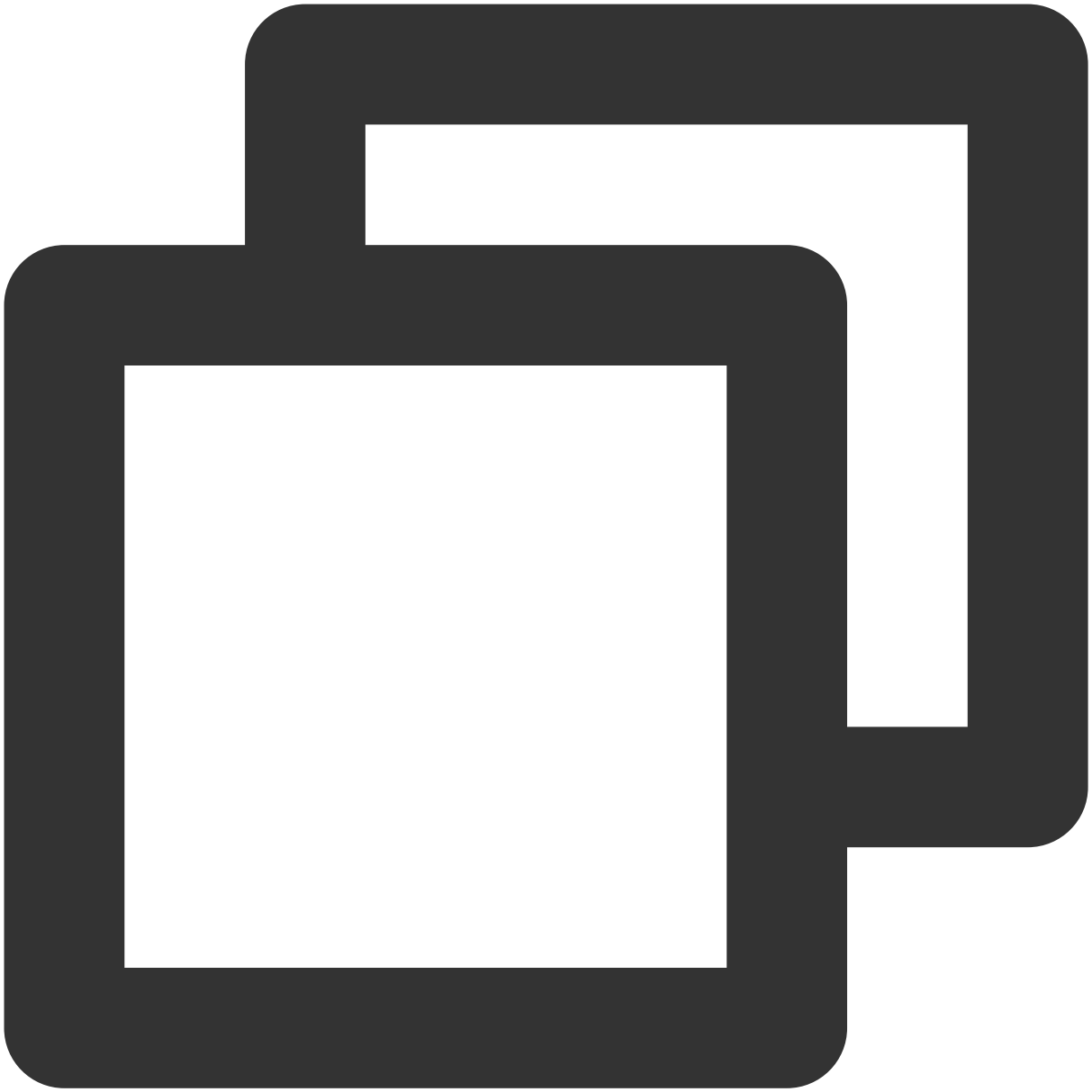


```
int PausePreview()
```

Resuming recording file preview

This API is used to resume recording file preview.

Function prototype

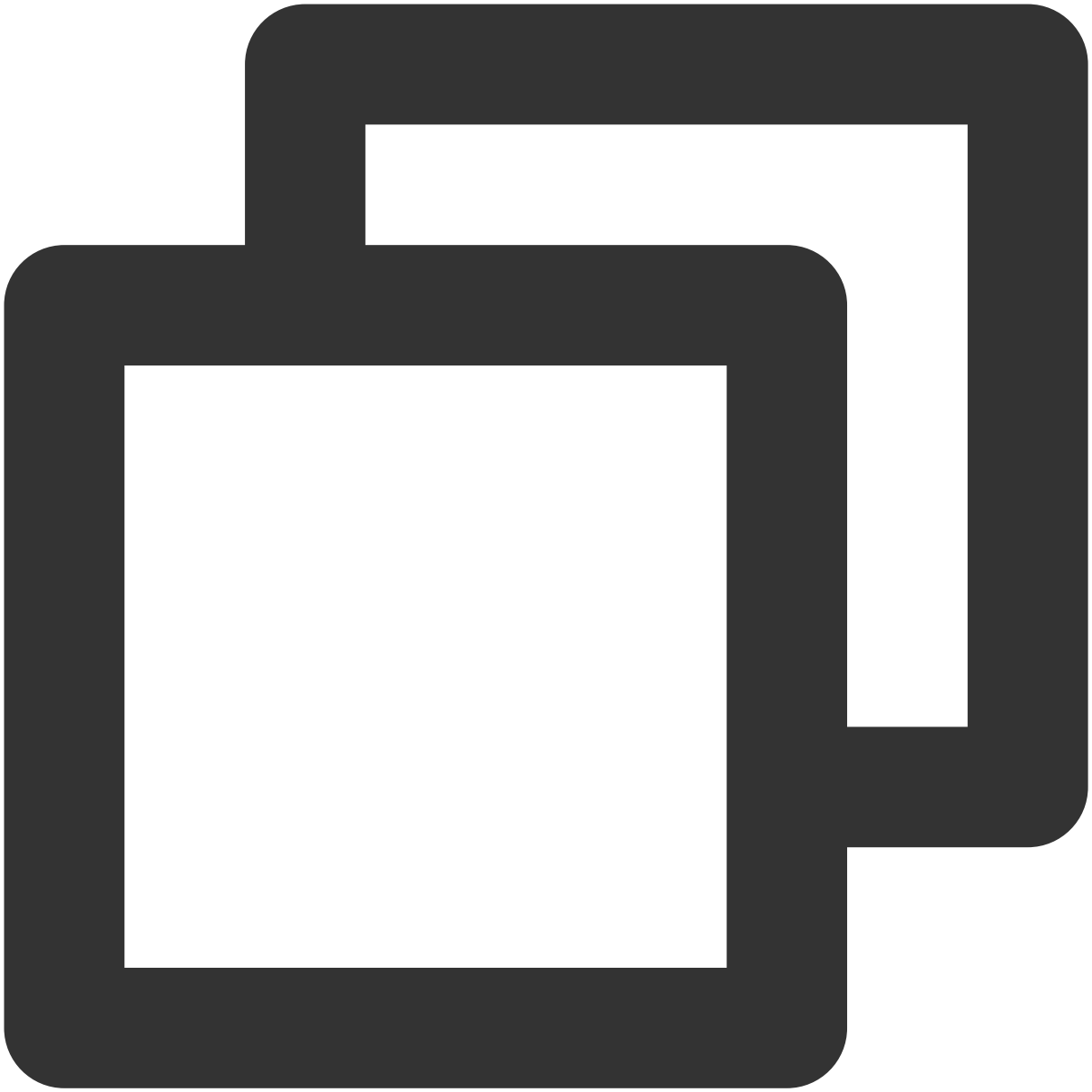


```
int ResumePreview()
```

Setting the current preview time point

This API is used to set the current preview time point.

Function prototype



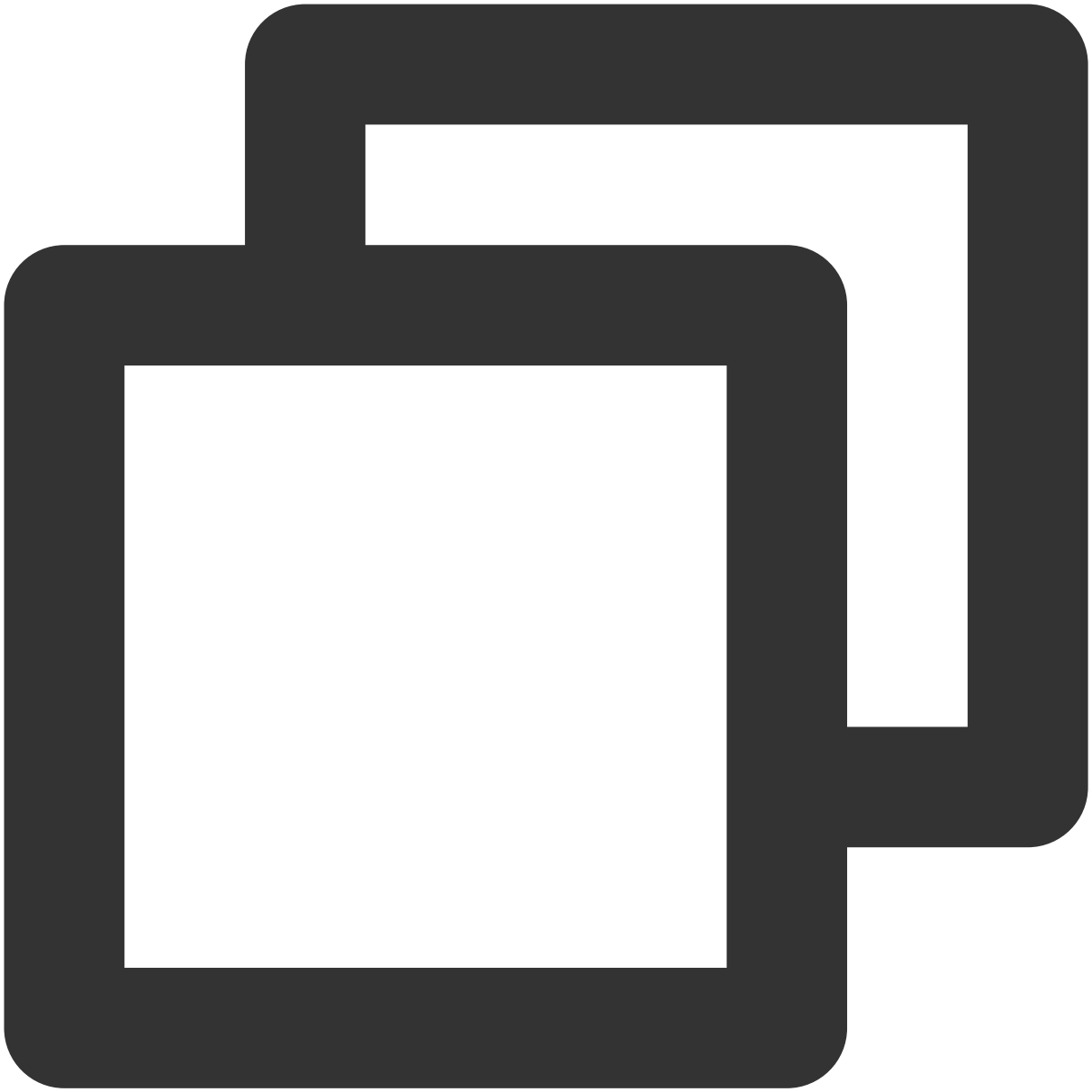
```
int SetPreviewTimeByMs(int time)
```

Parameter	Type	Description
time	int	Preview file time point in ms

Getting the current preview time point

This API is used to get the current preview time point.

Function prototype



```
int GetPreviewTimeByMs()
```

Playback preview callback

`ITMG_MAIN_EVENT_TYPE_RECORD_PREVIEW_COMPLETED` is the callback for preview completion, which will be triggered when the preview file playback ends or the `StopPreview` API is called.

Callback parameters

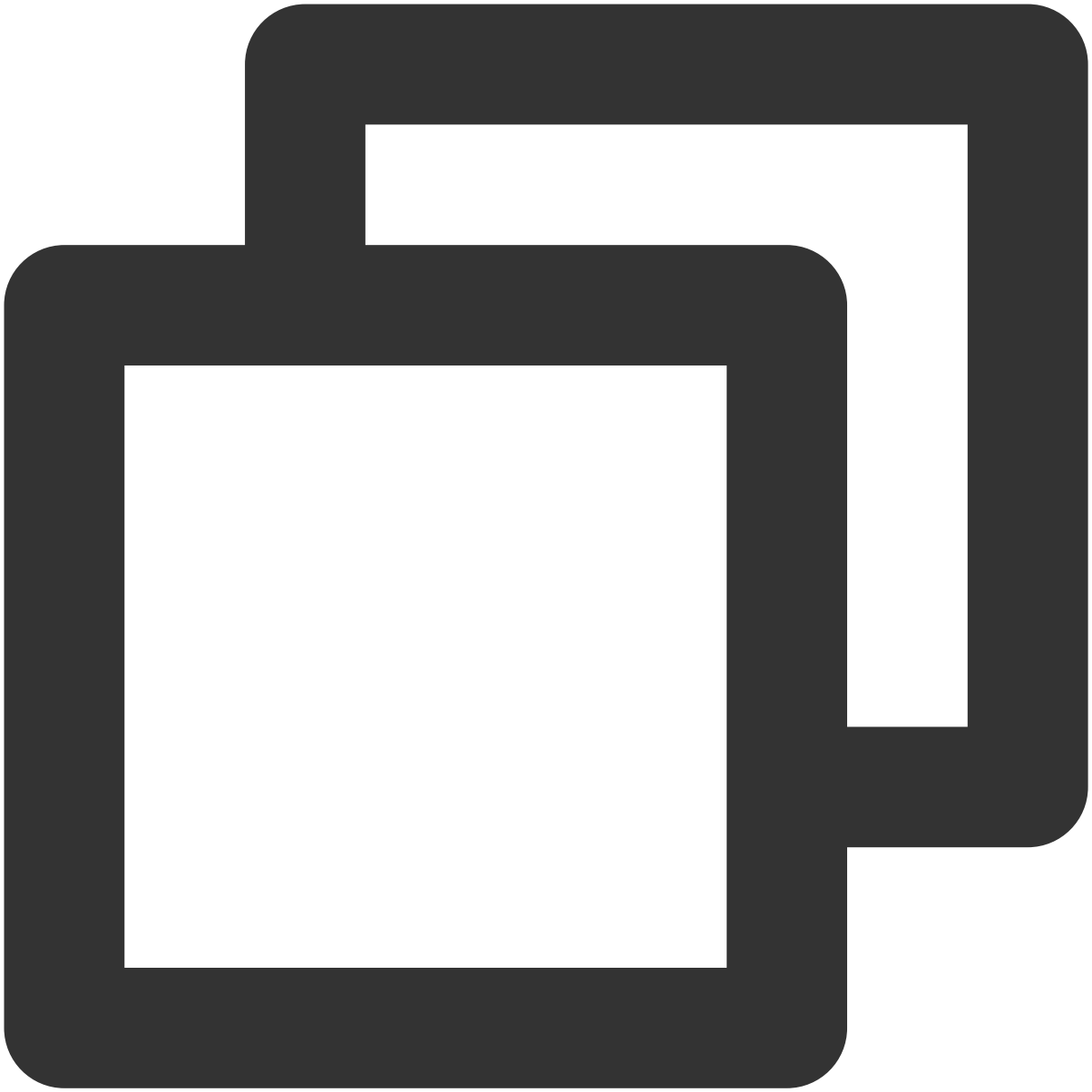
Parameter	Type	Description
result	int	Playback result. `0` indicates success.

File Mix APIs

Mixing files

This API is used to mix the recorded voice and accompaniment into a file.

Function prototype

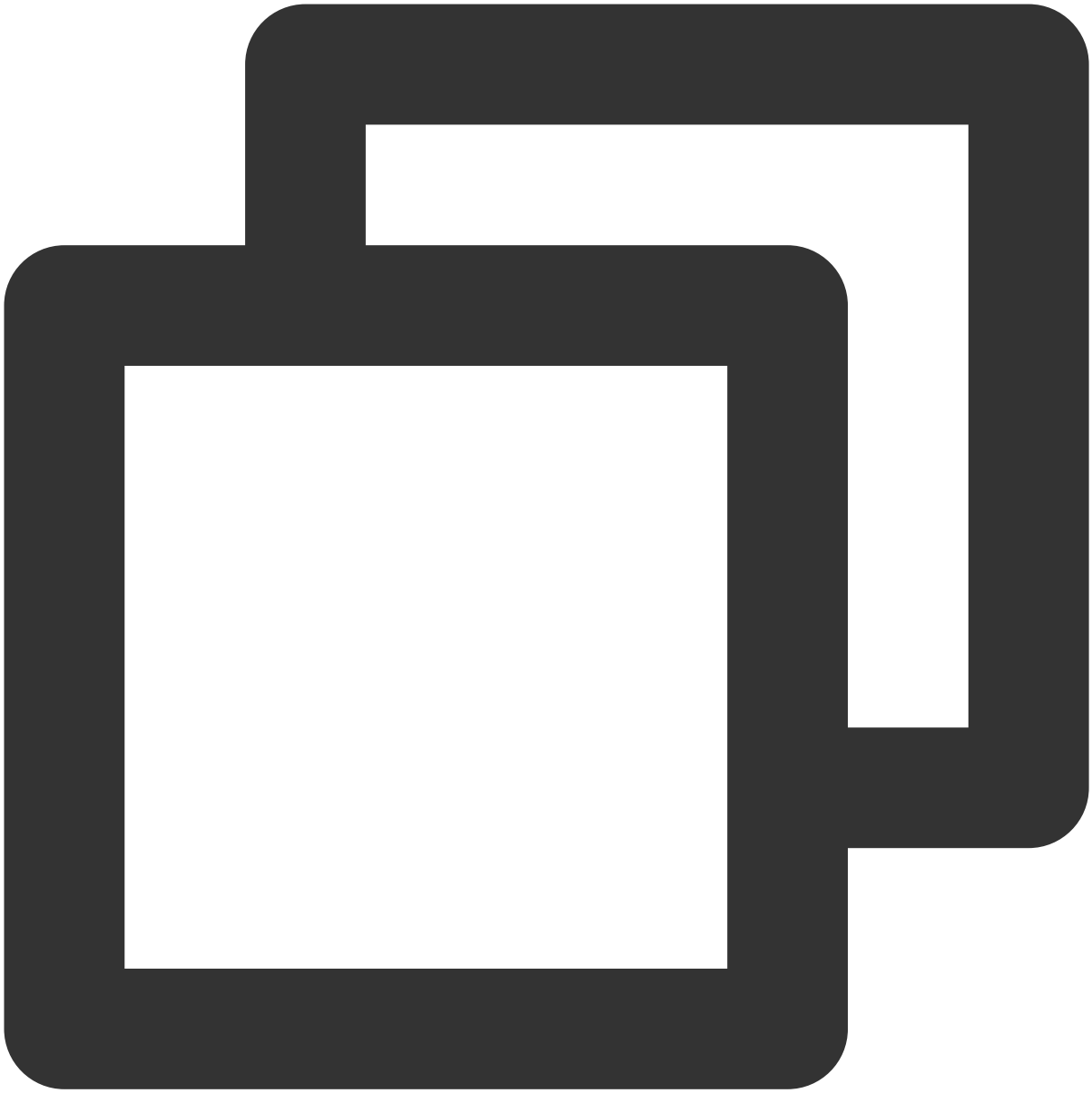


```
int MixRecordFile();
```

Canceling mix

This API is used to cancel the mix operation.

Function prototype



```
int CancelMixRecordFile();
```

File mix callback

`ITMG_MAIN_EVENT_TYPE_RECORD_MIX_COMPLETED` is the callback for mix completion.

Callback parameters

Parameter	Type	Description

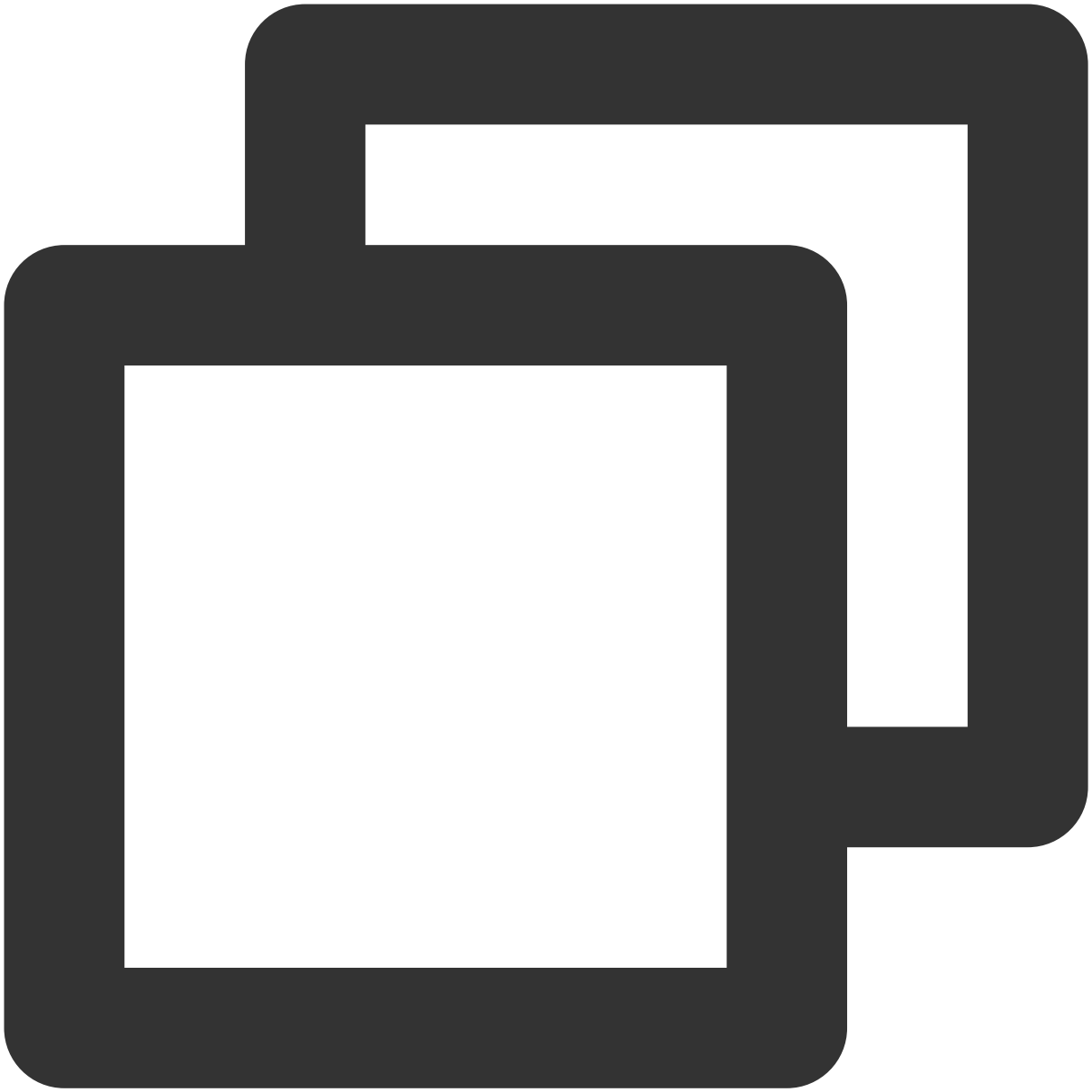
result	int	Mix result. `0` indicates success.
filepath	String	Target file path, which is the same as the `dstFile` parameter passed in through `StartRecord` .
duration	String	Recording file duration in ms.

Advanced Settings

Setting the accompaniment volume level ratio

This API is used to set both the ratios of the voice and accompaniment volume levels after recording.

Function prototype



```
int SetMixWieghts(float mic, float acc)
```

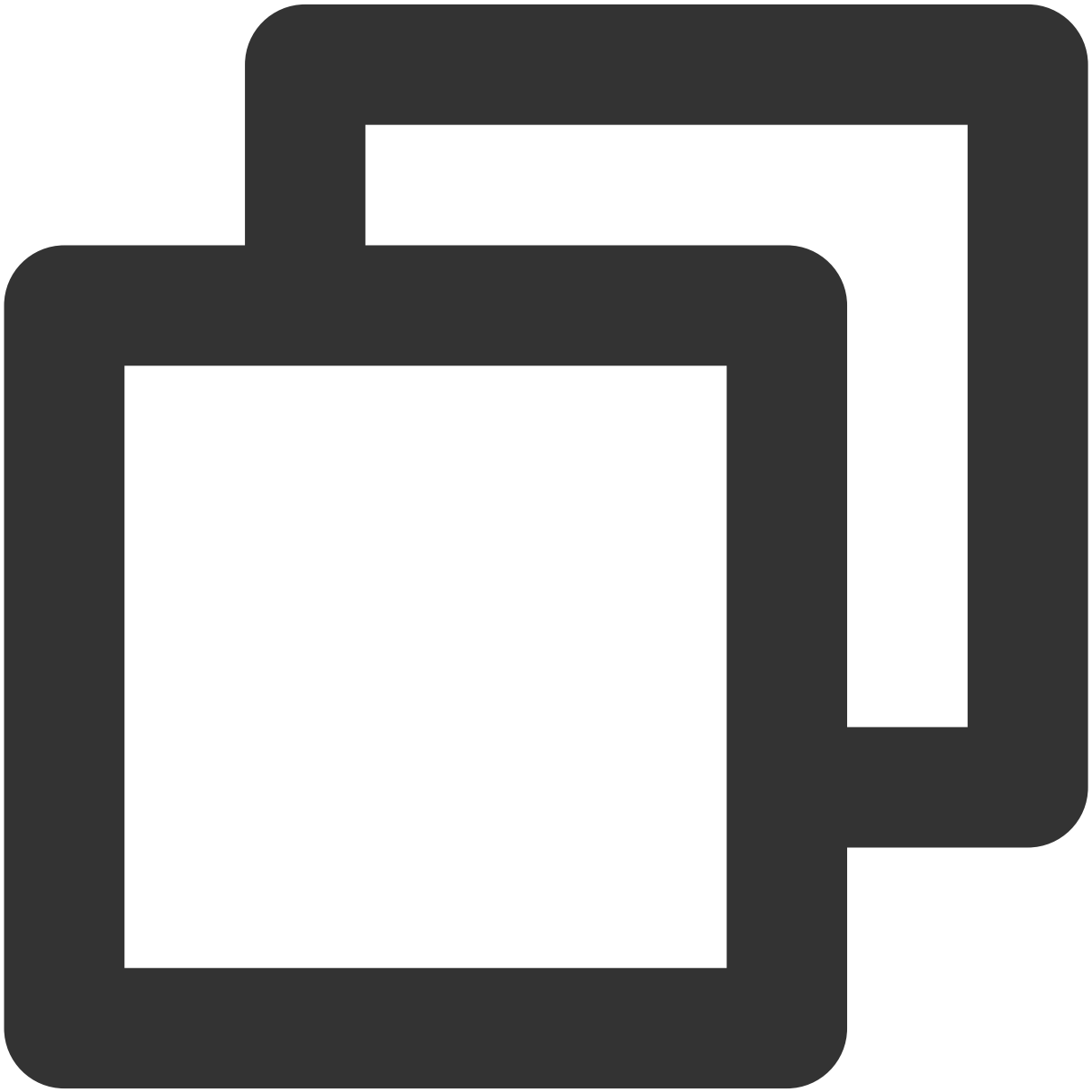
Parameter	Type	Description
mic	float	Ratio of the voice volume level. Value range: 0–2. `1.0` , a value less than `1.0` , and a value greater than `1.0` indicate to keep, decrease, and increase the original volume level respectively.
acc	float	Ratio of the accompaniment volume level. Value range: 0–2. `1.0` , a value less than `1.0` , and a value greater than `1.0` indicate to keep, decrease, and increase the

		original volume level respectively.

Setting the offset

This API is used to set the offset of the voice against the accompaniment. Generally, it is used to align the voice with beats after recording.

Function prototype



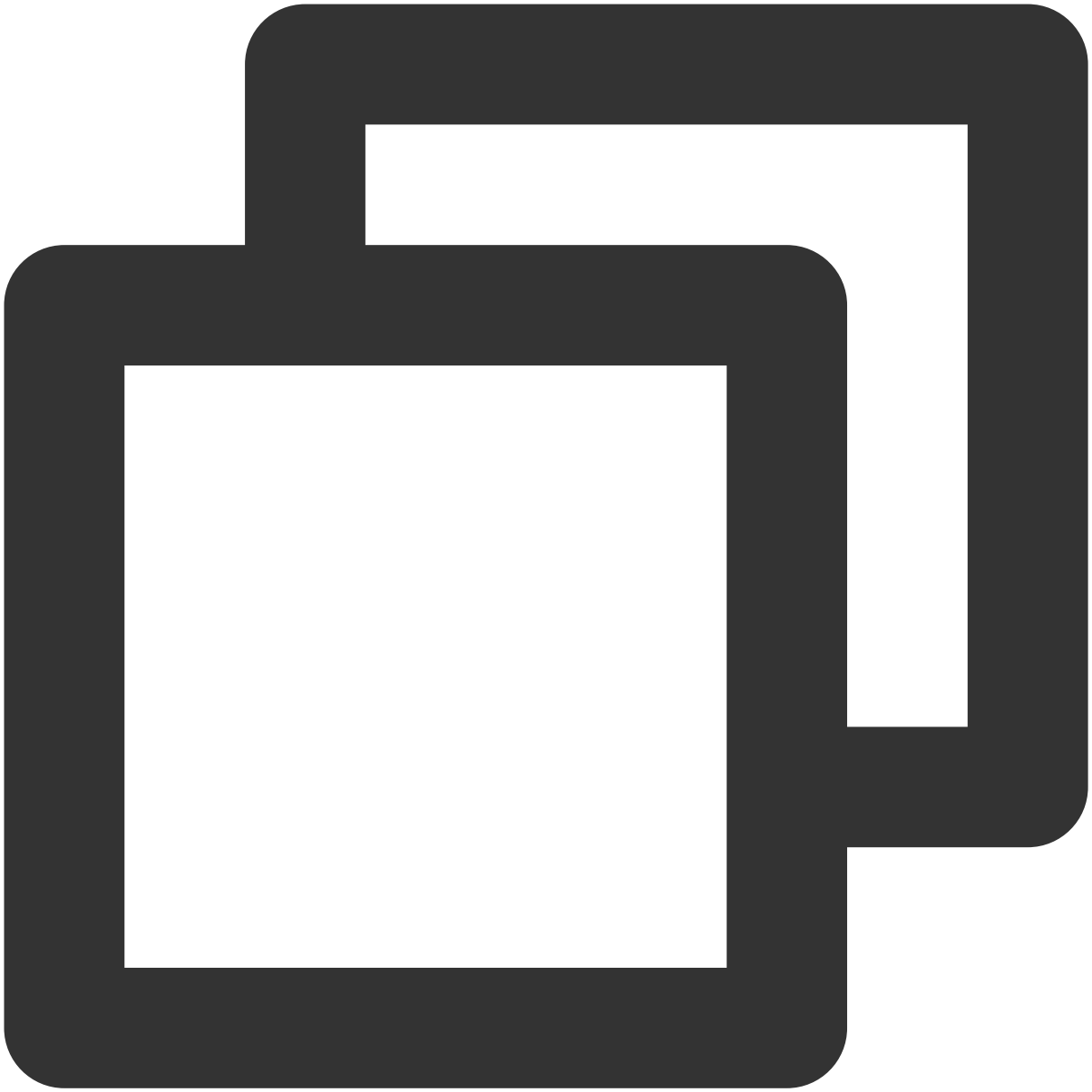
```
int AdjustAudioTimeByMs(int time)
```

Parameter	Type	Description
time	int	Offset of the voice against the accompaniment in ms. If the value is greater than `0`, the voice will be moved backward; if the value is less than `0`, the voice will be moved forward.

Setting the sound effect

This API is used to set the sound effect during or after recording. For specific karaoke sound effects, see Sound Effect in Voice Chat.

Function prototype



```
int SetRecordKaraokeType(int type)
```

Parameter	Type	Description
type	int	This type is the same as the karaoke sound effect type in voice chat. For more information, see Sound Effect in Voice Chat.

Network Audio Stream Forwarding Routing

Last updated：2023-04-27 17:28:55

This document provides a detailed description of using custom audio forwarding routing that makes it easy for developers to debug and integrate the APIs for Game Multimedia Engine (GME).

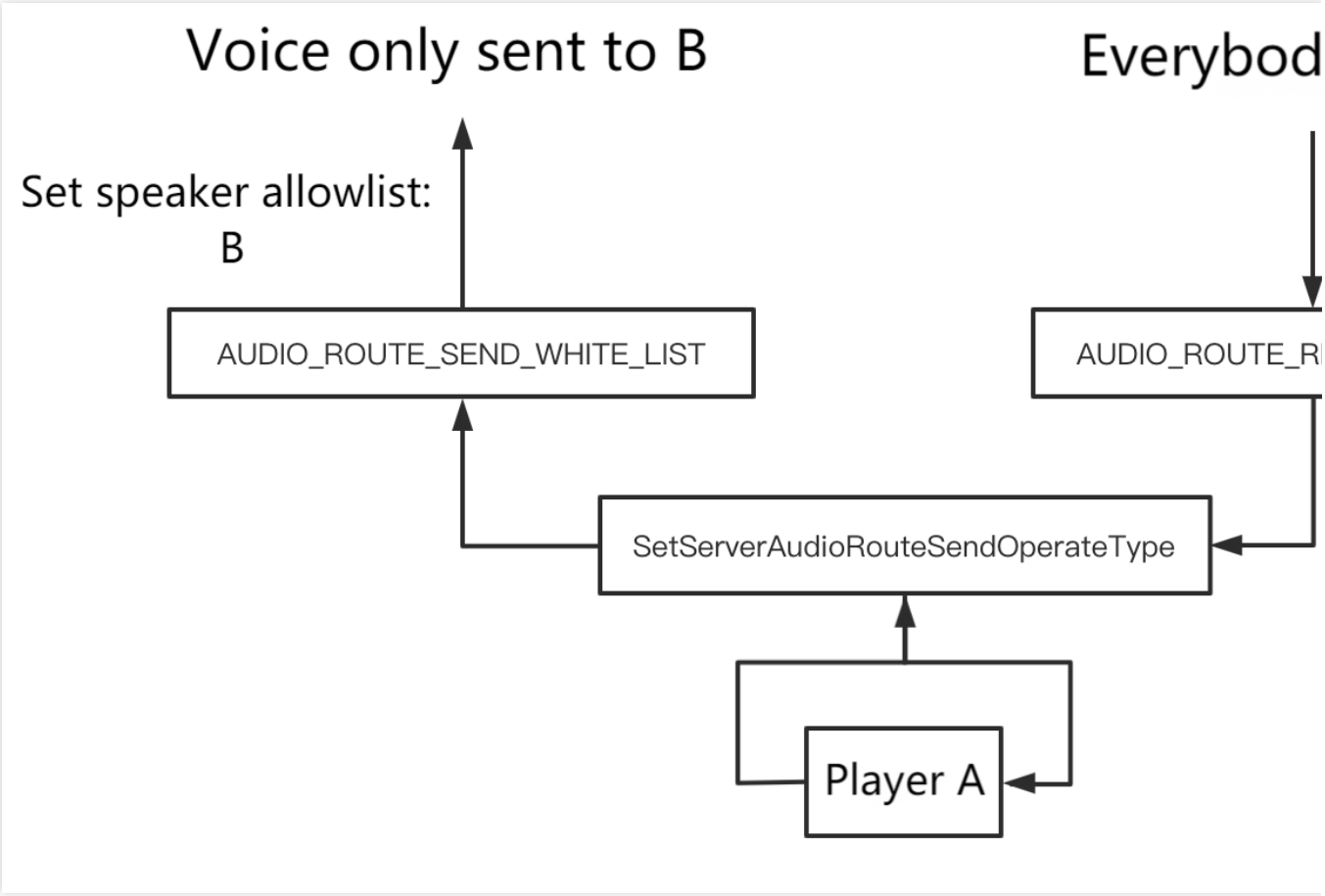
Use Cases

Scenario description: after two friends team up to form a small team, they match three strangers to form a large team. They want to hear the voice of all members in the large team but only talk to each other in the small team.

This can be achieved by the custom audio routing feature. Here, all five users enter the same voice chat room. Then, through the audio routing API settings, the players can be set to hear the voice in the two-player team or the entire room, as well as to be heard by the two-player team or the entire room.

Audio rule distance: SetServerAudioRouteSendOperateType(AUDIO_ROUTE_SEND_WHITE_LIST,"two-player team list",ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE,"two-player team list");

In this way, audio will be only sent to players in the list and will be only received from the three-player team.



Prerequisites

You have activated the voice chat service. For more information, see Activating Services.
You have integrated the GME SDK, including core APIs and voice chat APIs. For more information, see Quick Integration of Native SDK, Quick Integration of SDK for Unity, and Quick Integration of SDK for Unreal Engine.
You have successfully entered the voice room by using GME's voice chat feature and turned on the mic (EnableMic) and speaker (EnableSpeaker).

Accessing Audio Forwarding Routing

Setting audio forwarding rules

This API is used to set audio forwarding rules, and it is called in the callback of successful room entry. After being called, this API will take effect for this room entry and will be invalid after room exit.
Notes

The mute feature `AddBlackList` takes effect locally and has a higher priority than custom audio routing. For example, if A sets to only hear B's voice through `SetServerAudioRouteSendOperateType` but calls `AddBlackList` to mute B, then A will not hear B's voice.

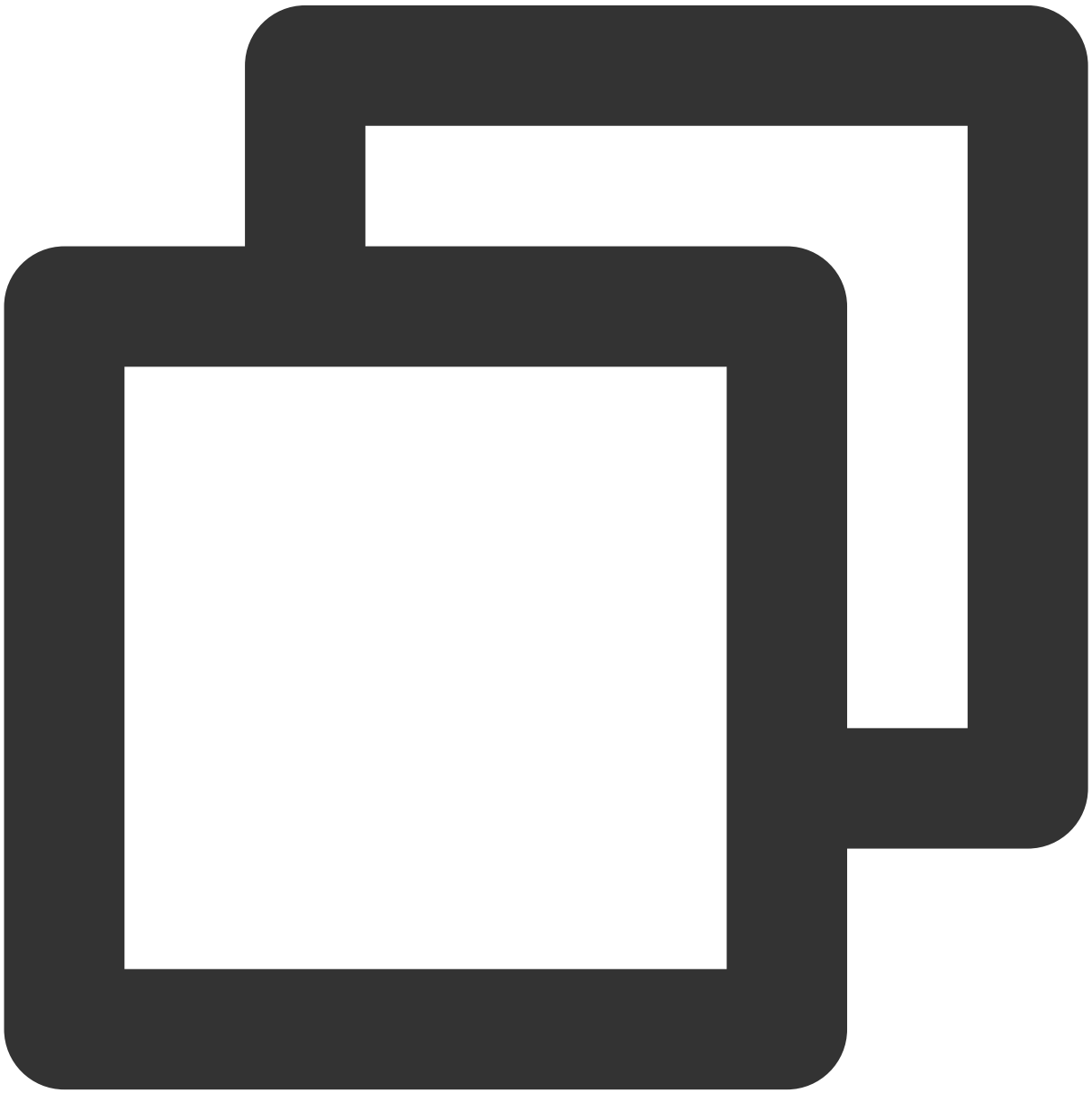
API prototype

Unity

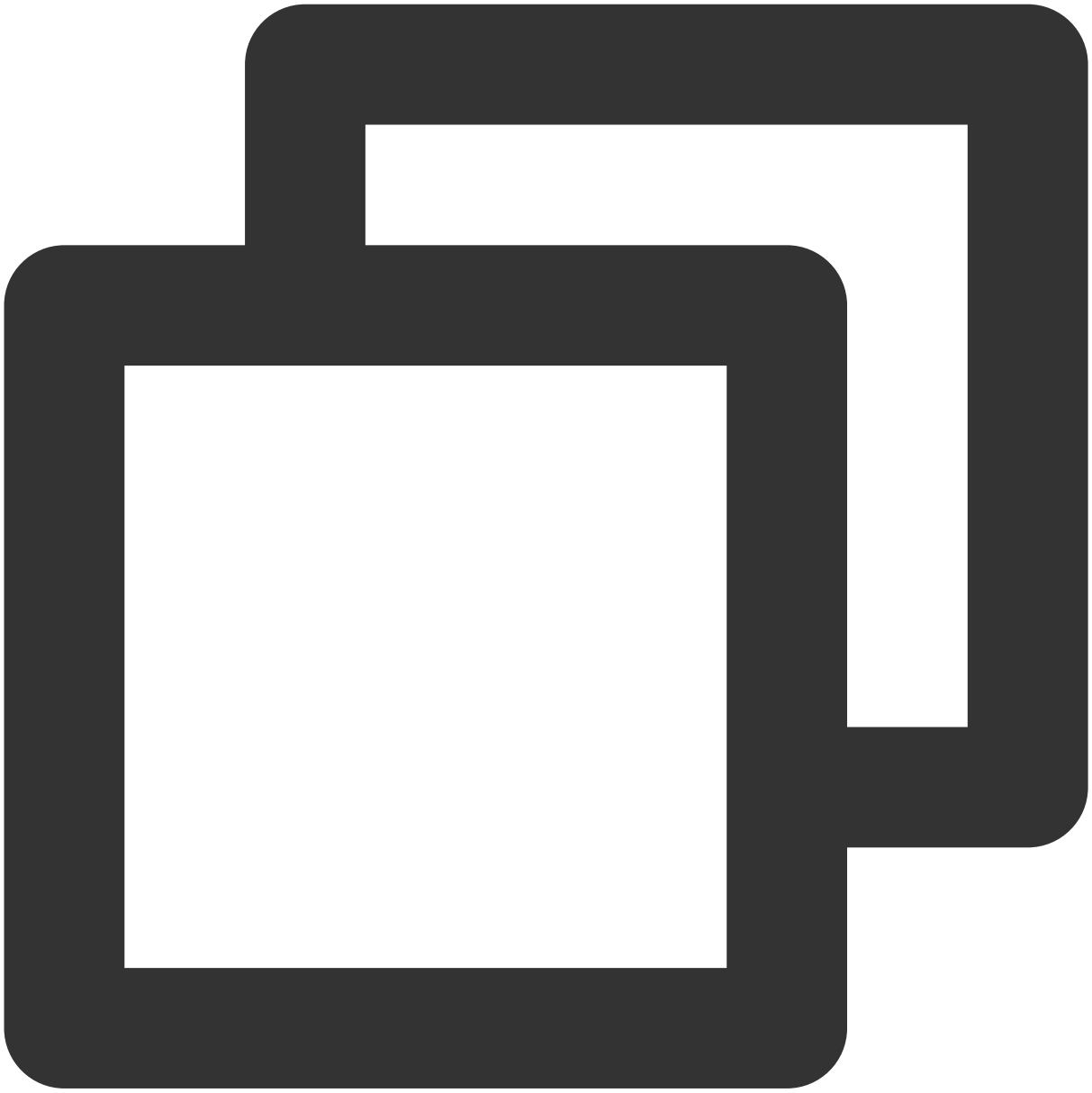
C++

Android

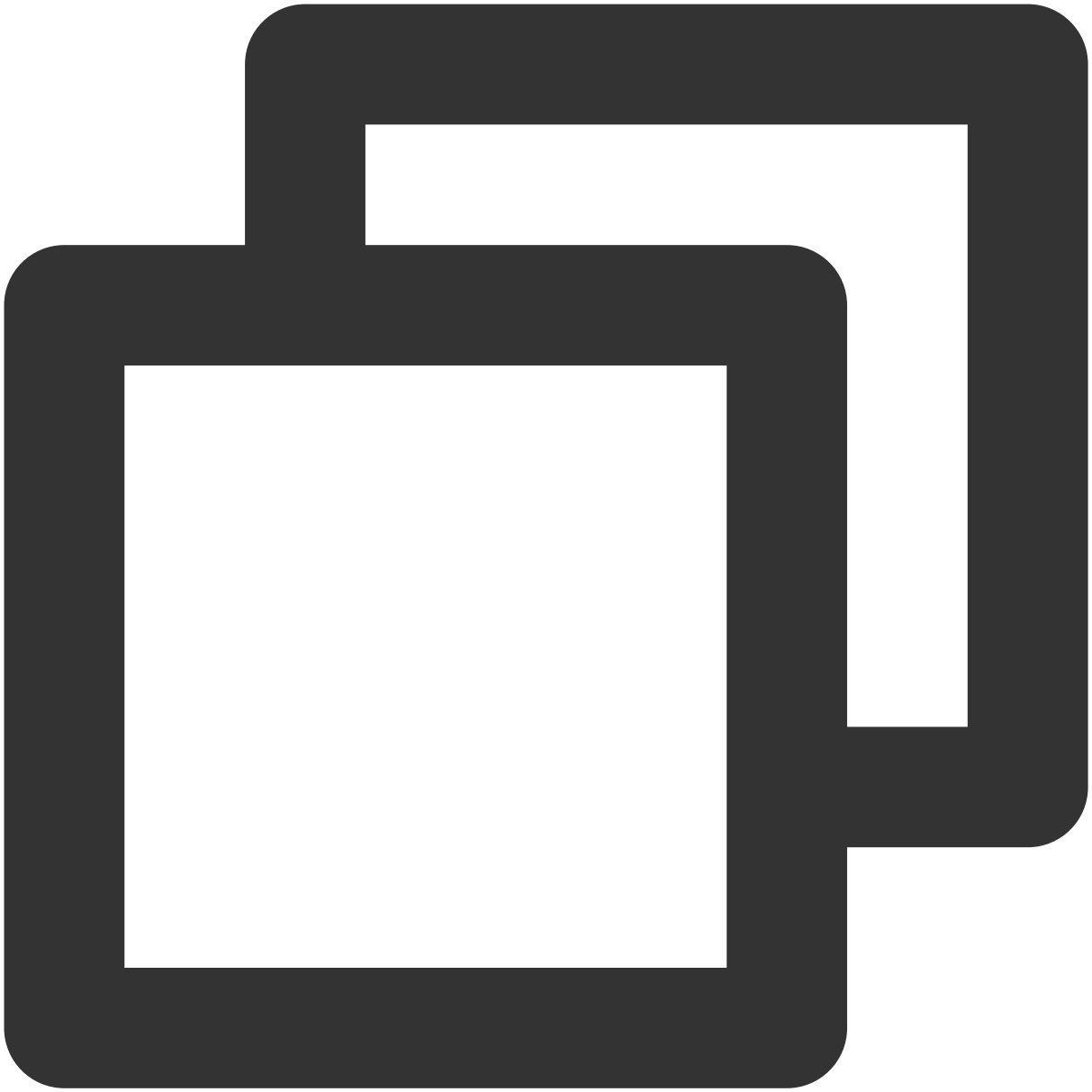
iOS



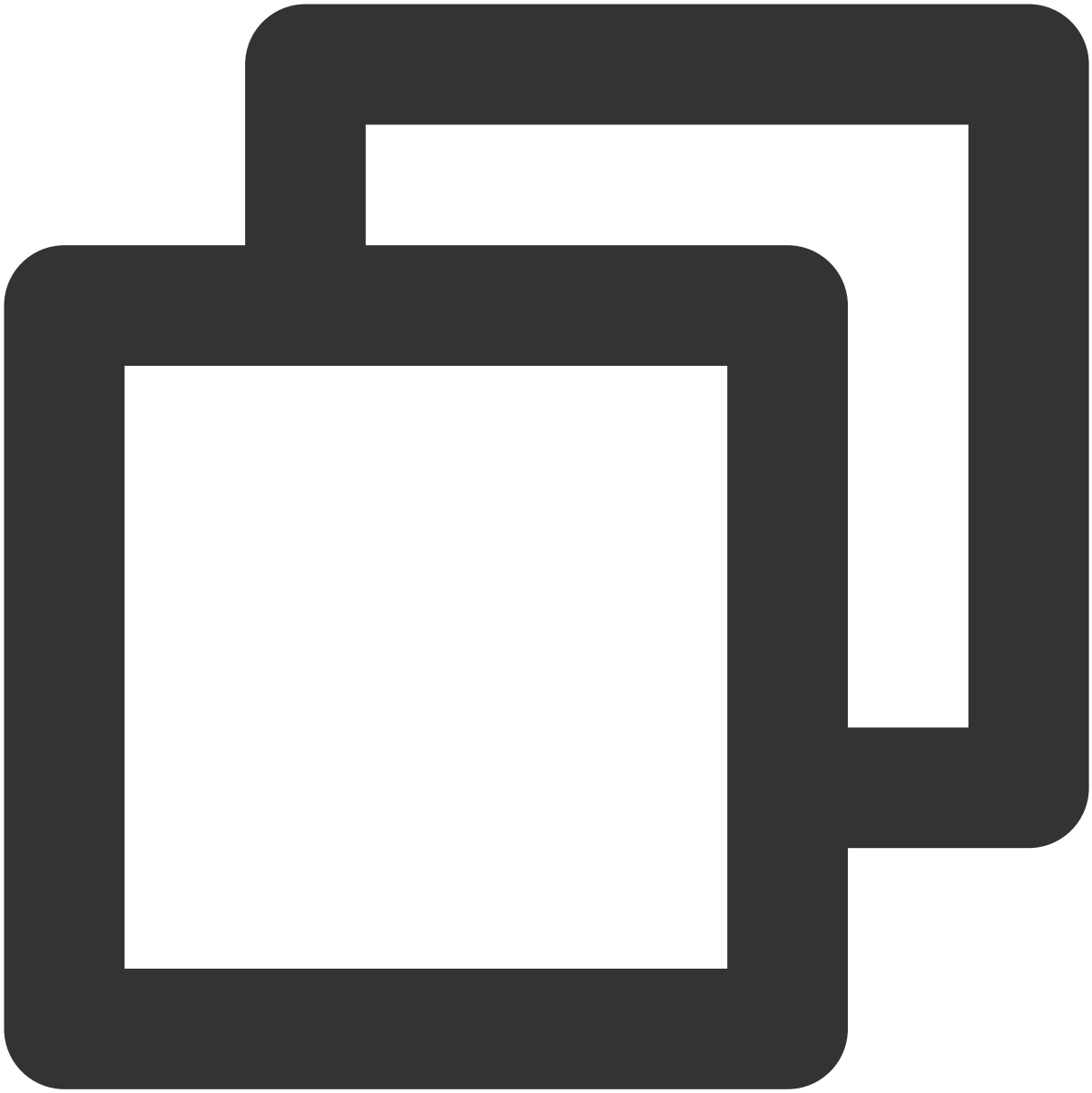
```
public abstract class ITMGRoom{
    public abstract int SetServerAudioRouteSendOperateType(ITMG_SERVER_AUDIO_ROUTE_
}
```



```
virtual int SetServerAudioRoute(ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE SendType, const c
```



```
public abstract int SetServerAudioRoute(ITMGContext.ITMG_SERVER_AUDIO_ROUTE_SEND_TY
```



```
-(int)SetServerAudioRouteSendOperateType:(ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE) Sendty
```

Type description

ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE

Set audio sending rules. There will be different sending effects for the different sending rules.

Receiving Type	Effect
AUDIO_ROUTE_NOT_SEND_TO_ANYONE	The local audio is sent upstream to the backend, but the backend does not forward to anyone, which is equivalent to

	mute the local. At this time, the parameter OpenIDForSend is invalid. You just need to enter null.
AUDIO_ROUTE_SEND_TO_ALL	The local audio is sent upstream to everyone. At this time, the parameter OpenIDForSend is invalid. You just need to enter null.
AUDIO_ROUTE_SEND_BLACK_LIST	The local audio is sent upstream and will not be forwarded to people in the blocklist, which is provided by the parameter OpenIDForSend.
AUDIO_ROUTE_SEND_WHITE_LIST	The local audio is sent upstream and is forwarded to people in the allowlist, which is provided by the parameter OpenIDForSend.

Note

If the passed type is AUDIO_ROUTE_NOT_SEND_TO_ANYONE or AUDIO_ROUTE_SEND_TO_ALL, then the parameter OpenIDForSend does not take effect. You just need to enter null.

If the passed type is AUDIO_ROUTE_SEND_BLACK_LIST, the parameter OpenIDForSend will be the blocklist. Up to 10 people are supported.

If the passed type is AUDIO_ROUTE_SEND_WHITE_LIST, the parameter OpenIDForSend will be the allowlist. Up to 10 people are supported.

ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE

Set audio receiving rules. There will be different receiving effects for the different receiving rules.

Receiving Type	Effect
AUDIO_ROUTE_NOT_RECV_FROM_ANYONE	The local does not receive any audio, which is equivalent to disable the speaker in the room. At this time, the parameter OpenIDForSend is invalid. You just need to enter null.
AUDIO_ROUTE_RECV_FROM_ALL	The local receives everyone's audio. At this time, the parameter OpenIDForSend is invalid. You just need to enter null.
AUDIO_ROUTE_RECV_BLACK_LIST	The local does not receive the audio of people in the blocklist. The blocklist is provided by the parameter OpenIDForSend.
AUDIO_ROUTE_RECV_WHITE_LIST	The local will only receive the audio of people in the allowlist. The allowlist is provided by the parameter OpenIDForSend.

Note

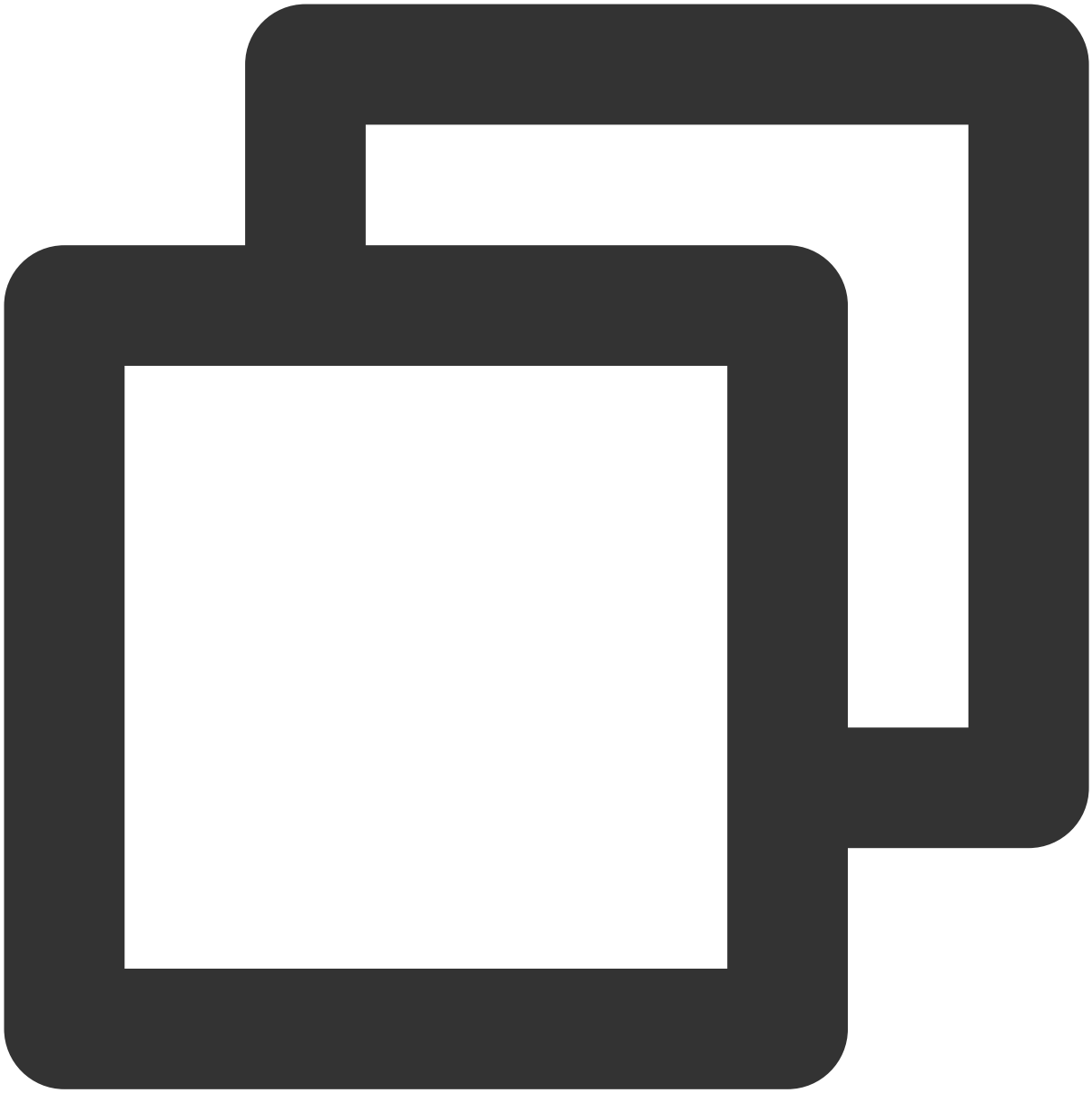
If the passed type is AUDIO_ROUTE_NOT_RECV_FROM_ANYONE or AUDIO_ROUTE_RECV_FROM_ALL, the parameter OpenIDForSend does not take effect.
If the passed type is AUDIO_ROUTE_RECV_BLACK_LIST, the parameter OpenIDForSend will be the blocklist. Up to 10 people are supported.
If the passed type is AUDIO_ROUTE_RECV_WHITE_LIST, the parameter OpenIDForSend will be the allowlist. Up to 10 people are supported.

Returned values

A returned value of QAV_OK indicates the call is successful.
If the callback returns 1004, it means the parameter is incorrect. Check the parameter.
If the callback returns 1001, it means the operation is repeated.
If the callback returns 1201, it means the room does not exist. Check whether the room number is correct.
If the callback returns 10001 and 1005, call the API again.
For more explanations of the returned result, see Error Codes.

Sample code

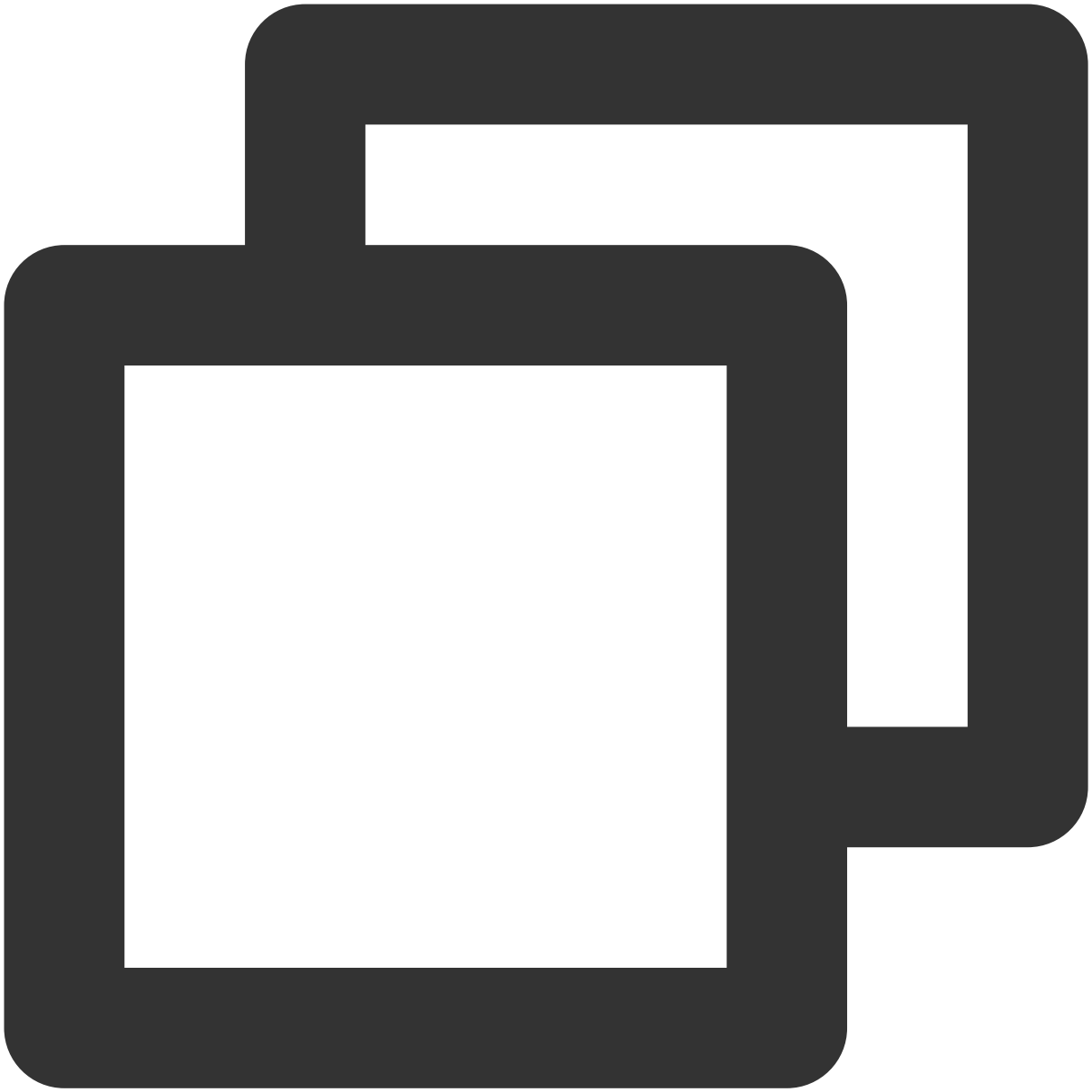
Executed statements



```
@synthesize _sendListArray;
@synthesize _recvListArray;

int ret =  [[[ITMGContext GetInstance] GetRoom] SetServerAudioRouteSendOperateType:
if (ret != QAV_OK) {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Failed to update audi
    [alert show];
}
```

Callback



```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSString* log =[NSString stringWithFormat:@"OnEvent:%d,data:%@", (int)eventType
    switch (eventType) {
                case ITMG_MAIN_EVENT_TYPE_SERVER_AUDIO_ROUTE_EVENT:{
            {
                UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Update au
                              [alert show];
            }
        }
        default:
            break;
```

```
        }
    }
```

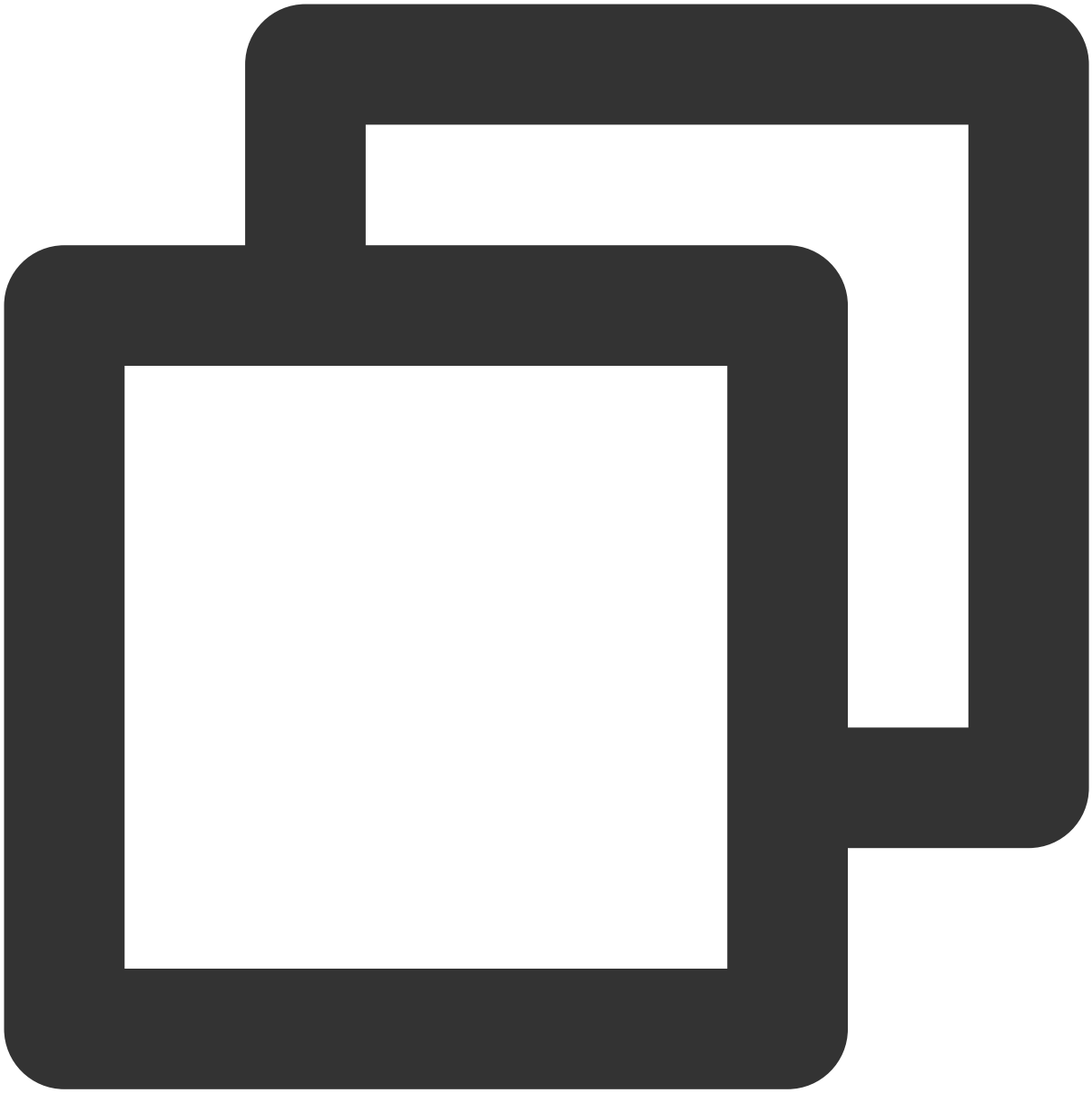
Obtaining audio forwarding rules

This API is used to obtain audio forwarding rules. After being called, the API returns the rule, and the passed array parameter will return the openId of the corresponding rule.

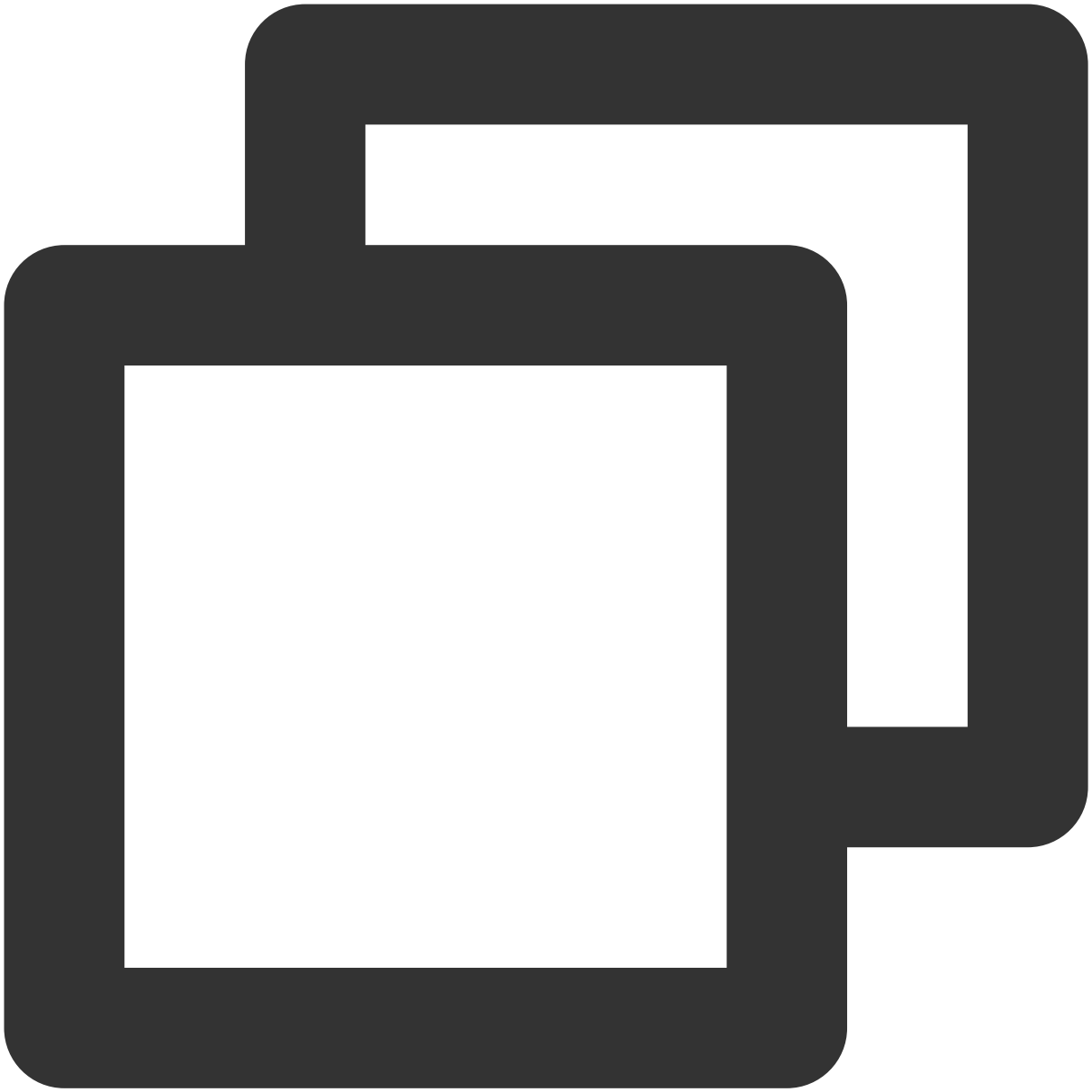
API prototype

Unity

iOS



```
public abstract ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE GetCurrentSendAudioRoute(List<str
public abstract ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE GetCurrentRecvAudioRoute(List<str
```



```
-(ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE)GetCurrentSendAudioRoute:(NSMutableArray *) Ope
-(ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE)GetCurrentRecvAudioRoute:(NSMutableArray *) Ope
```

Returning rules

ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE

Receiving Type	Effect
AUDIO_ROUTE_NOT_SEND_TO_ANYONE	The local audio is sent upstream to the backend, but the backend does not forward to anyone, which is equivalent to

	mute the local.
AUDIO_ROUTE_SEND_TO_ALL	The local audio is sent upstream to everyone.
AUDIO_ROUTE_SEND_BLACK_LIST	The local audio is sent upstream and will not be forwarded to people in the blocklist.
AUDIO_ROUTE_SEND_WHITE_LIST	The local audio is sent upstream and is forwarded to people in the allowlist.
AUDIO_ROUTE_RECV_INQUIRE_ERROR	An error occurred during obtaining. Check whether the local has entered the room and whether the SDK has been initialized.

ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE

Receiving Type	Effect
AUDIO_ROUTE_NOT_RECV_FROM_ANYONE	The local does not receive any audio, which is equivalent to disable the speaker in the room.
AUDIO_ROUTE_RECV_FROM_ALL	The local receives everyone's audio.
AUDIO_ROUTE_RECV_BLACK_LIST	The local does not receive the audio of people in the blocklist.
AUDIO_ROUTE_RECV_WHITE_LIST	The local will only receive the audio of people in the allowlist.
AUDIO_ROUTE_RECV_INQUIRE_ERROR	An error occurred during obtaining. Check whether the local has entered the room and whether the SDK has been initialized.

Notes

Do not use `AUDIO_ROUTE_RECV_INQUIRE_ERROR` in the `SetServerAudioRouteSendOperateType` API.

Custom Message Channel

Last updated：2023-04-27 17:30:57

This document provides a detailed description of using custom audio package attached with message that makes it easy for developers to debug and integrate the APIs for Game Multimedia Engine (GME).

Use Cases

The feature of custom audio package attached with message allows you to attach custom messages in the GME audio package as a signaling broadcast to users in the same room.

Prerequisites

You have activated the voice chat service. For more information, see Activating Services.
You have integrated the GME SDK, including core APIs and voice chat APIs. For more information, see Quick Integration of Native SDK, Quick Integration of SDK for Unity, and Quick Integration of SDK for Unreal Engine.

Use Limits

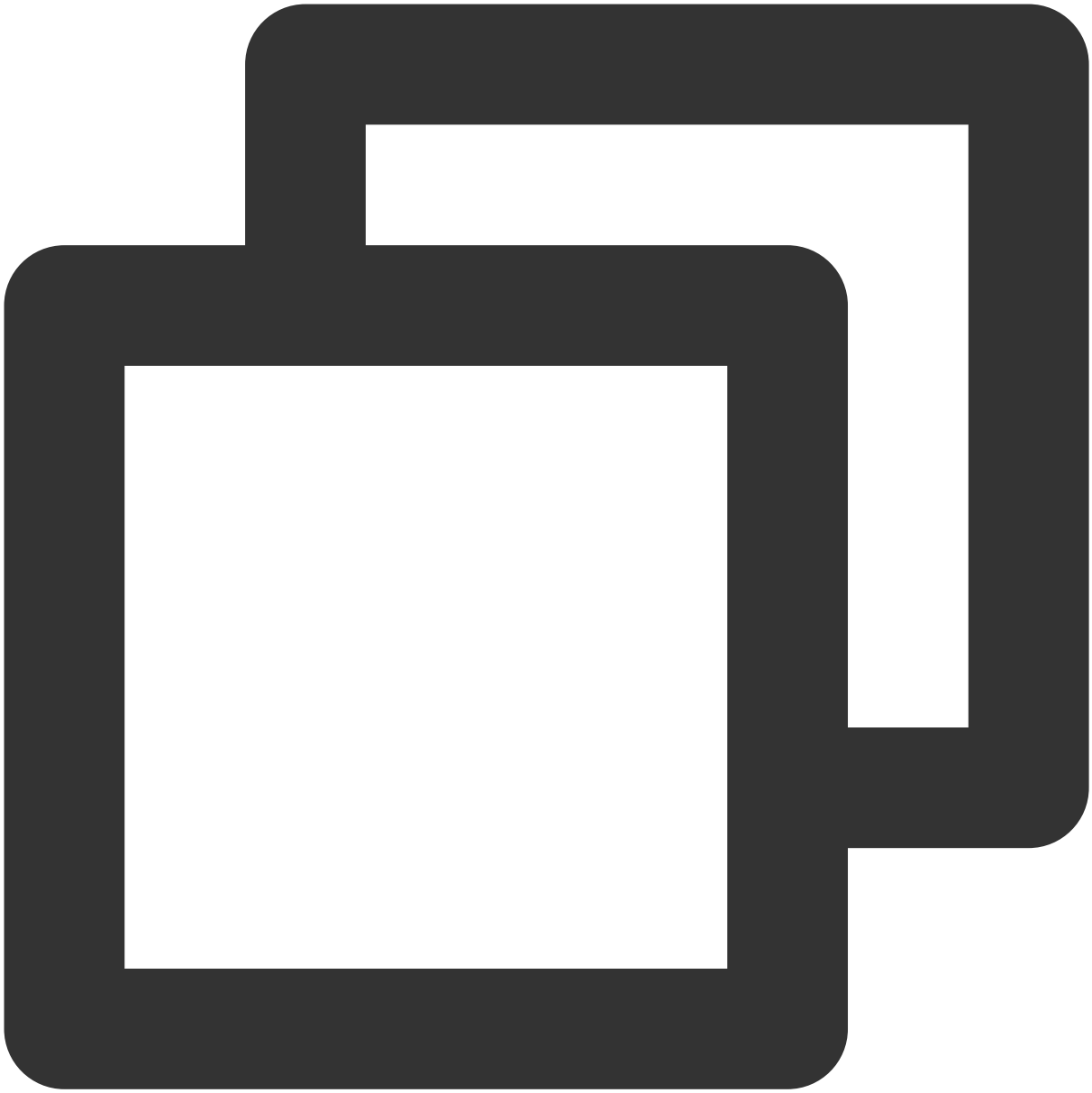
Calling this API must meet two requirements: The room type should be **Standard** or **HD** (`ITMG_ROOM_TYPE_STANDARD` or `ITMG_ROOM_TYPE_HIGHQUALITY`); the mic of the sender and the speaker of the receiver are both on.

Integrating the Custom Message Feature

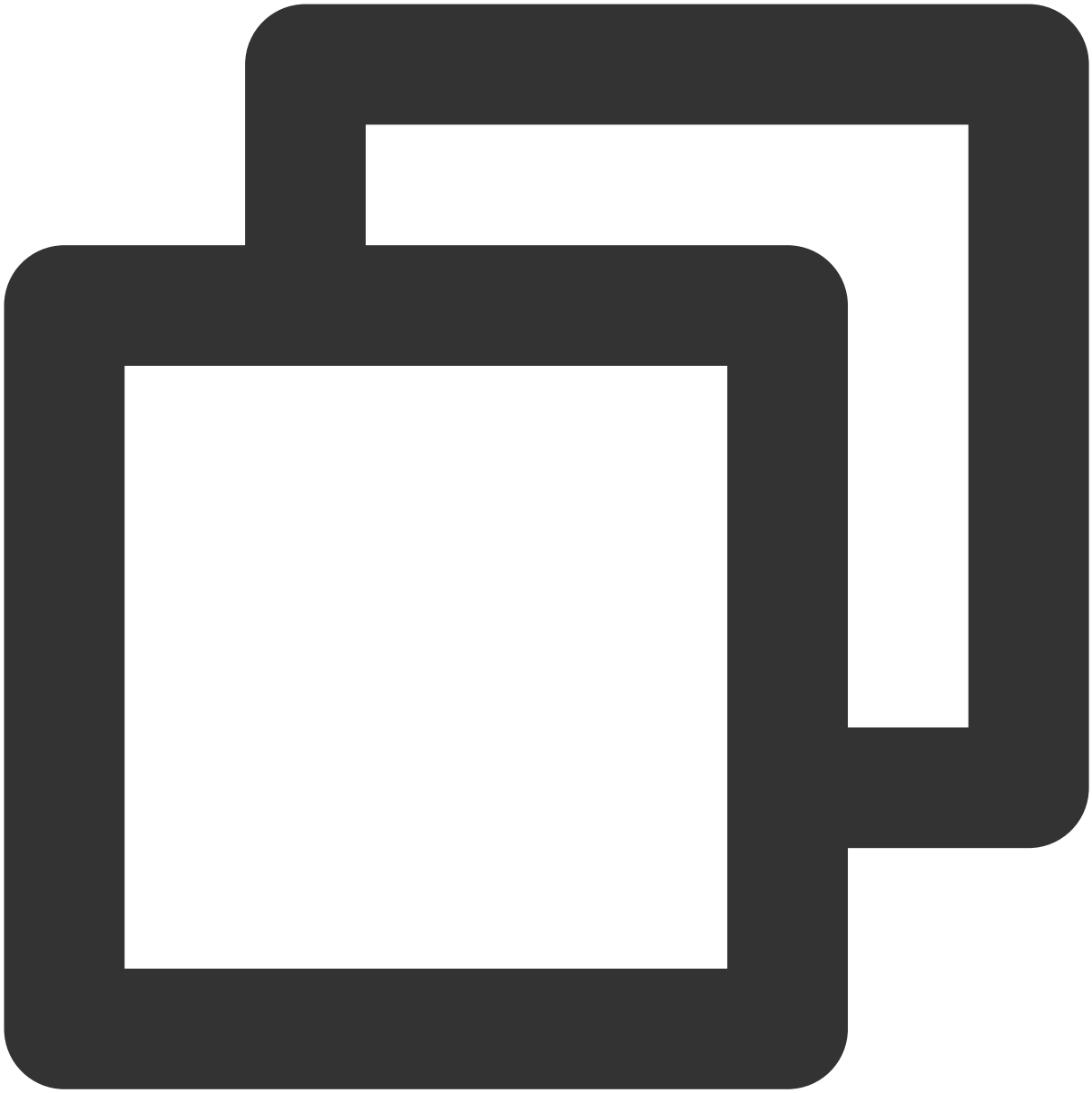
Sending custom messages

API prototype

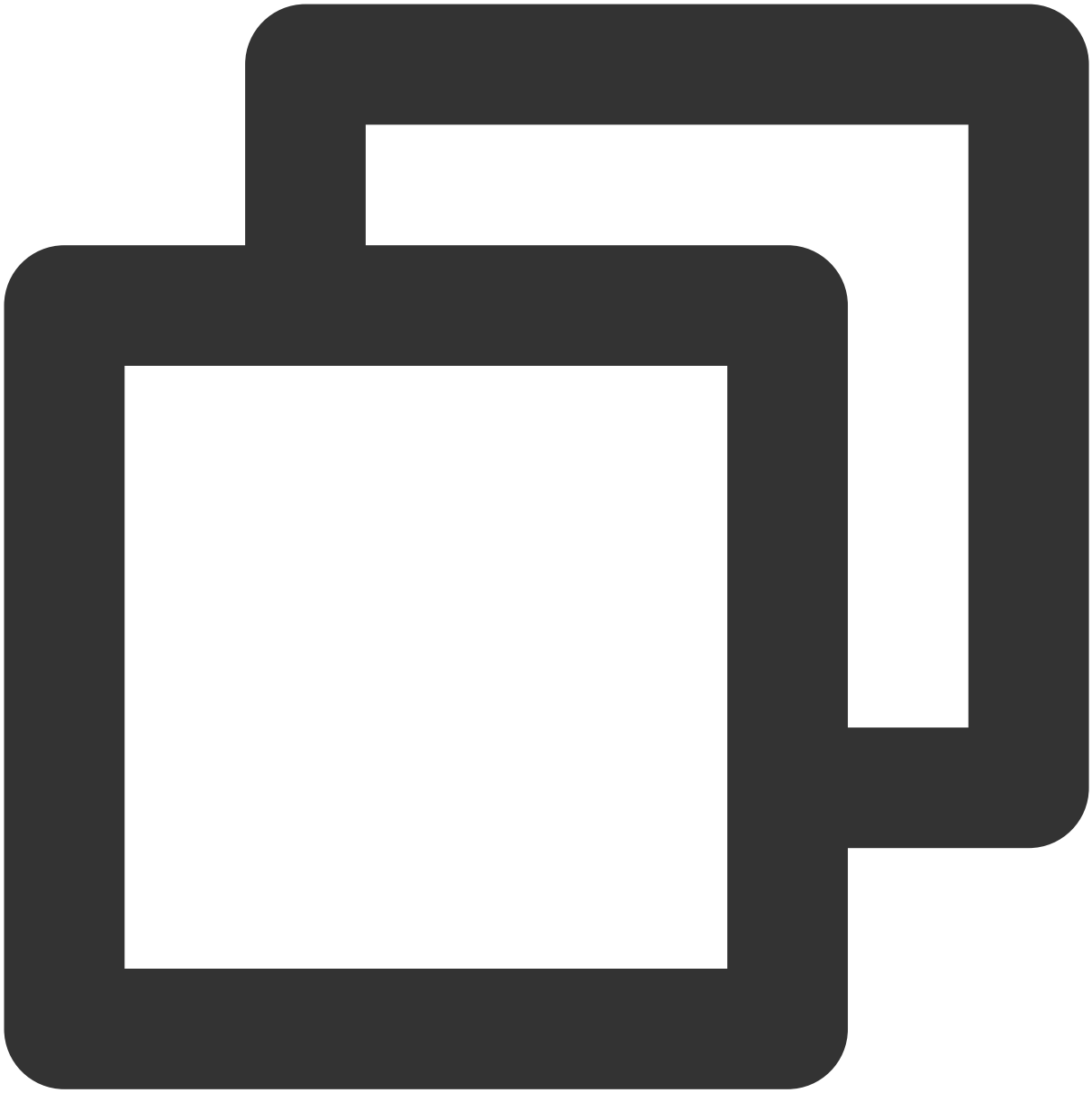
iOS
Android
Unity



```
-(int) SendCustomData:(NSData *)data repeatCout:(int)reaptCout;
```



```
public abstract int SendCustomData(byte[] data,int repeatCout);
```



```
public abstract int SendCustomData(byte[] customdata,int repeatCout);
```

Field description

Parameter	Type	Description
data	NSData * or byte[]	Message to be delivered
reaptCout	int	Number of repeat times. `-1` indicates to send the message repeatedly for

		unlimited times.

Returned values

A returned value of QAV_OK indicates the call is successful.

Callback returning 1004 indicates parameter error, please check parameter. Returning 1201 indicates non-existent room, please check room number.

For more error codes, see Error Codes.

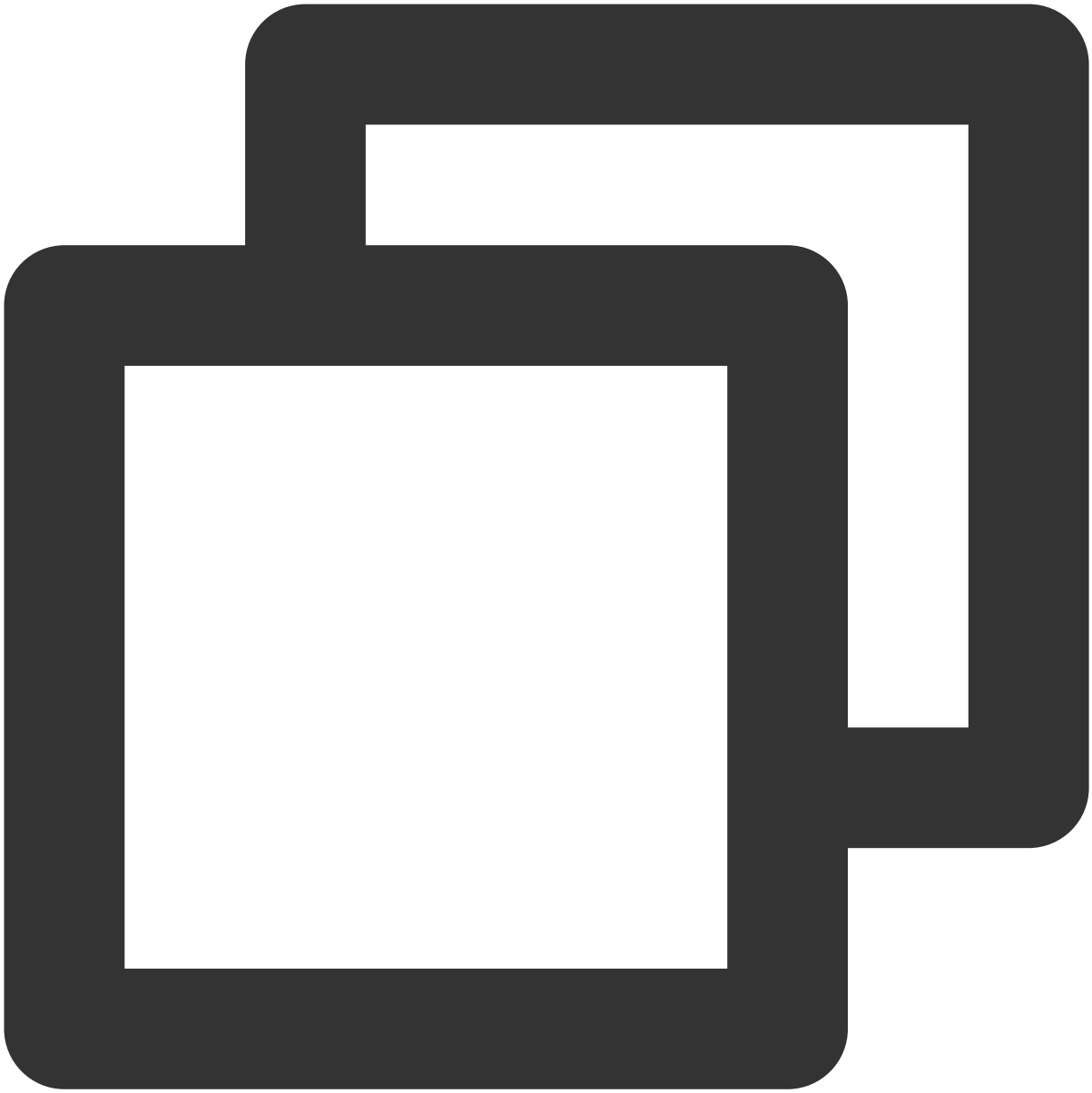
Sample code

Executed statements

iOS

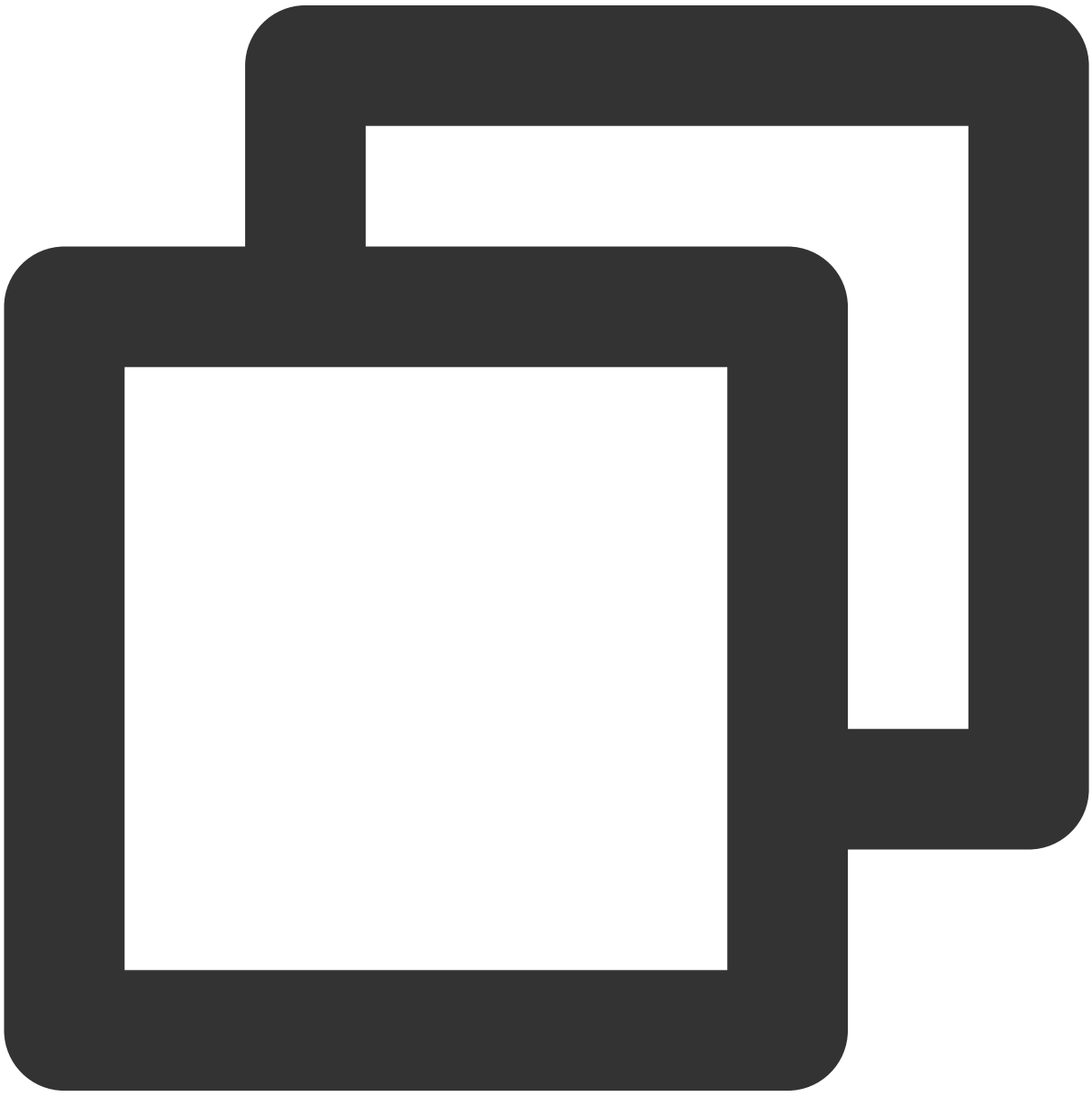
Android

Unity



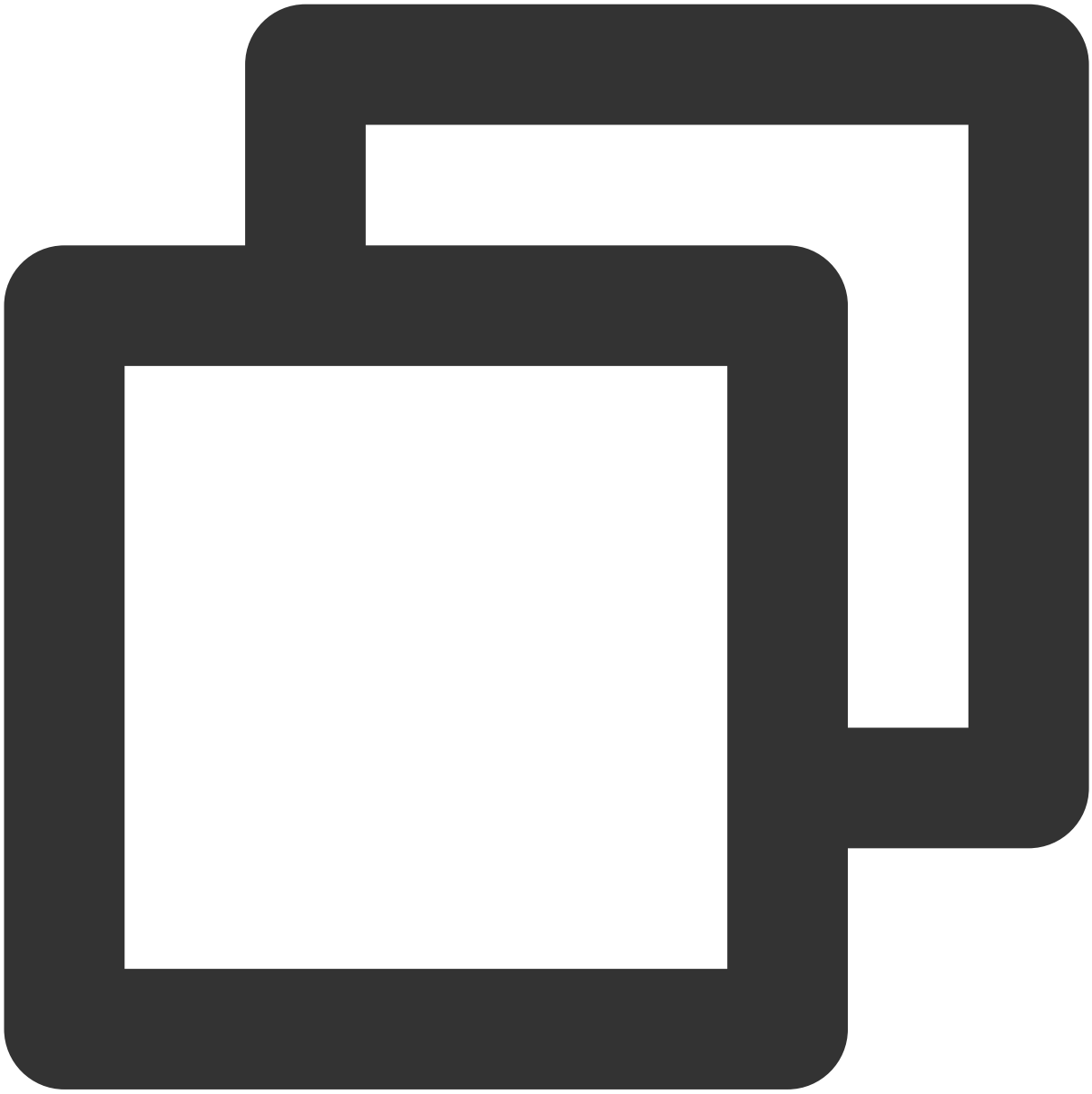
```
-(IBAction)SendCustData:(UIButton*)sender {
    int ret = 0;
    NSString *typeString;
    switch (sender.tag) {
        case 1:
            ret = [[[ITMGContext GetInstance] GetRoom] SendCustomData:[NSData dataW
            typeString = @"sendCustData";
            break;
          case 2:
            ret = [[[ITMGContext GetInstance] GetRoom] StopSendCustomData];
            typeString = @"recvCustData";
```

```
            break;
        default:
            break;
    }
    if(ret != 0){
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"set fail" message
        [alert show];
    }
}
```



```
String strData = mEditData.getText().toString();
```

```
String repeatCount = mEditRepeatCount.getText().toString();
int nRet = ITMGContext.GetInstance(getActivity()).GetRoom().SendCustomData(strData.
```



```
InputField SendCustom_Count_InputField = transform.Find("inroomPanel/imPanel/SendCu
InputField SendCustom_Data_InputField = transform.Find("inroomPanel/imPanel/SendCus

transform.Find("inroomPanel/imPanel/SendCustom_Btn").GetComponent<Button>().onClick
       {
           string data = SendCustom_Data_InputField.text;
           string str_count = SendCustom_Count_InputField.text;
```

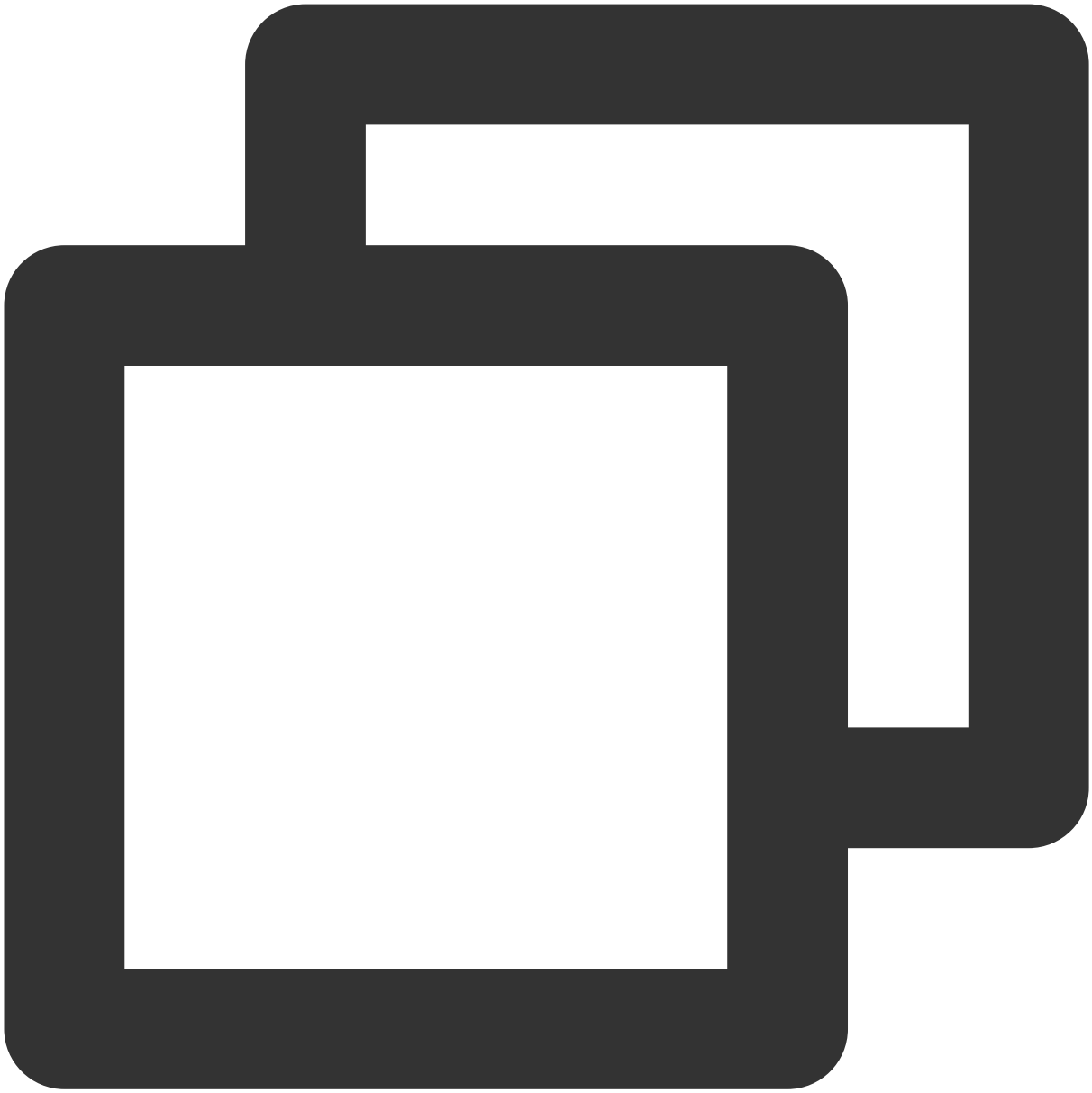
```
            int count = 0;
            if (int.TryParse(str_count, out count)) {
                Debug.Log(data+ count.ToString());
                byte[] byteData = Encoding.Default.GetBytes(data);
               int ret =  ITMGContext.GetInstance().GetRoom().SendCustomData(byteDat
               if(ret != 0 ) {
                  ShowWarnning(string.Format("send customdata failed err:{0}",ret));
               }
            }
        });
}
```

Callback

iOS

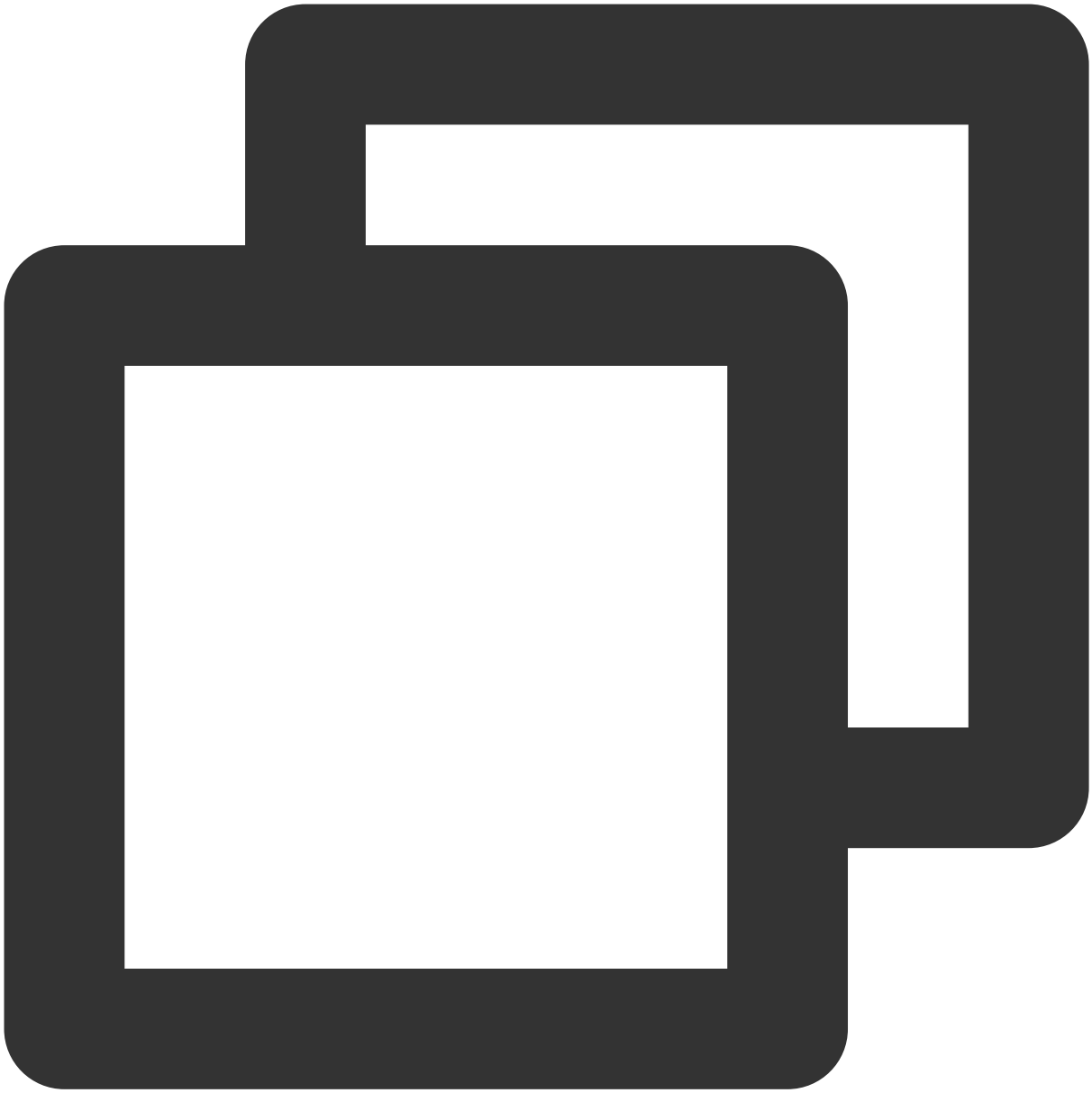
Android

Unity

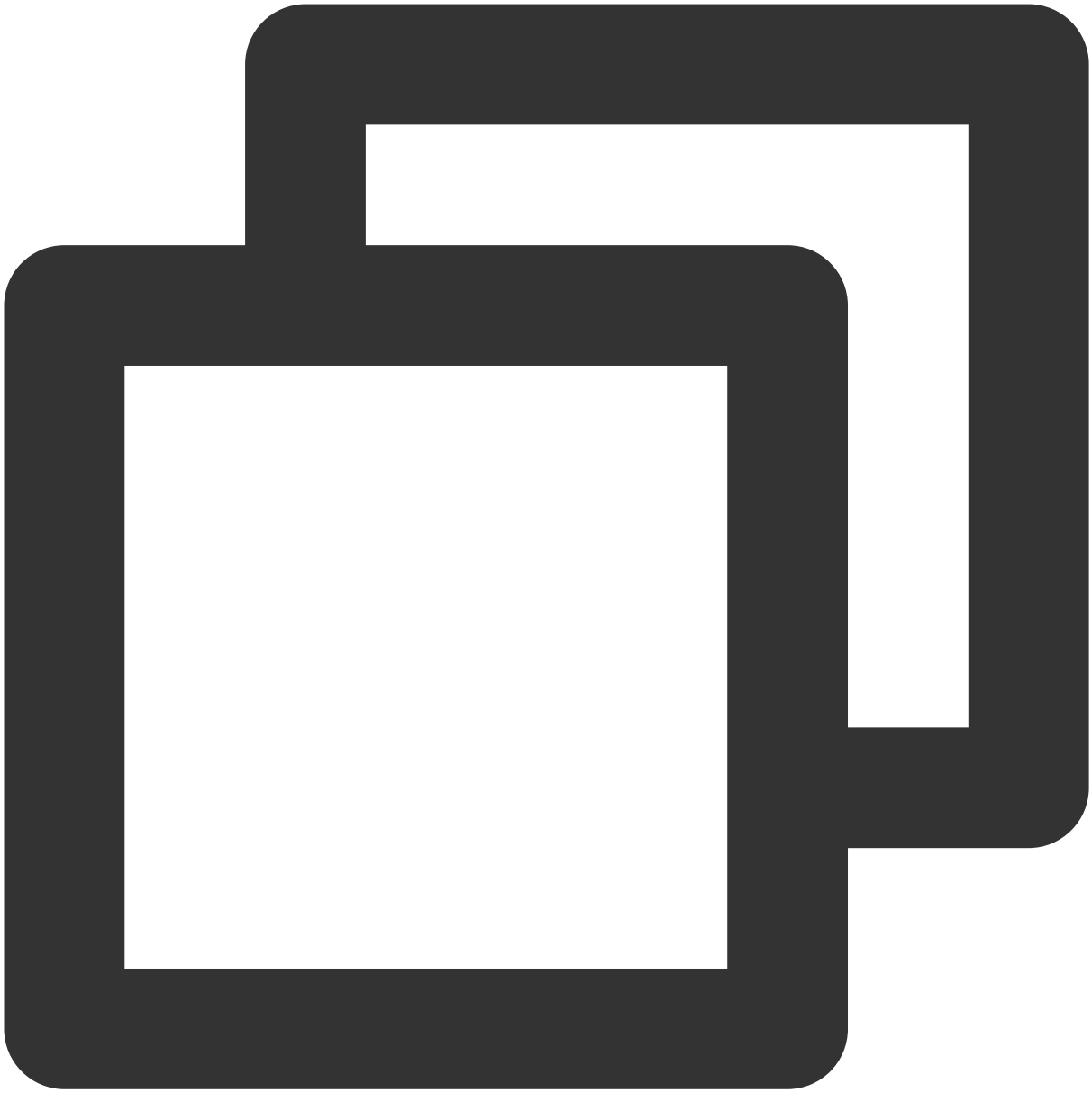


```
-(IBAction)SendCustData:(UIButton*)sender {
    int ret = ret = [[[ITMGContext GetInstance] GetRoom] SendCustomData:[NSData dat

    if(ret != 0){
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"set fail" message
        [alert show];
    }
}
```



```
if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CUSTOMDATA_UPDATE == type
    int subtype  =  data.getIntExtra("sub_event",-1);
    if (subtype == 0) {
        String content =  data.getStringExtra("content");
        String sender = data.getStringExtra("senderid");
        Toast.makeText(getActivity(), String.format("recv content =%s, from:%s", co
            sender), Toast.LENGTH_SHORT).show();
}
```



```
void OnEvent(int eventType,int subEventType,string data)
{
    Debug.Log (data);
    switch (eventType) {
     case (int)ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CUSTOMDATA_UPDATE:
      {
         if(subEventType == (int)ITMG_CUSTOMDATA_SUB_EVENT.ITMG_CUSTOMDATA_AV_SUB_E
             _customData = JsonUtility.FromJson<CustomDataInfo>(data);
           ShowWarnning(string.Format("recve customdata {0}  from {1}",_customData.
         }
      }
```

```
        break;
    }
```

Stopping sending custom messages

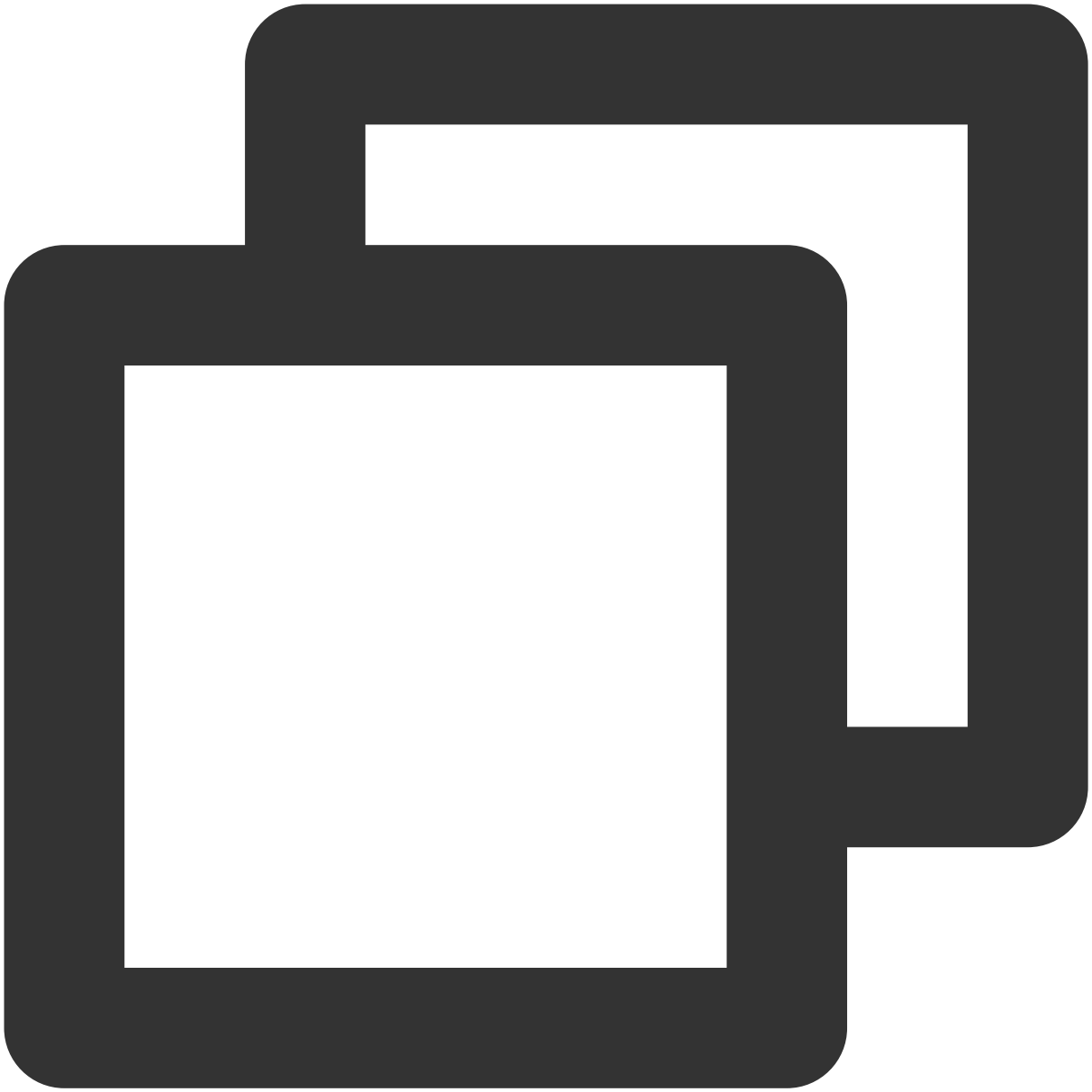
Call this API to stop sending custom messages

API prototype

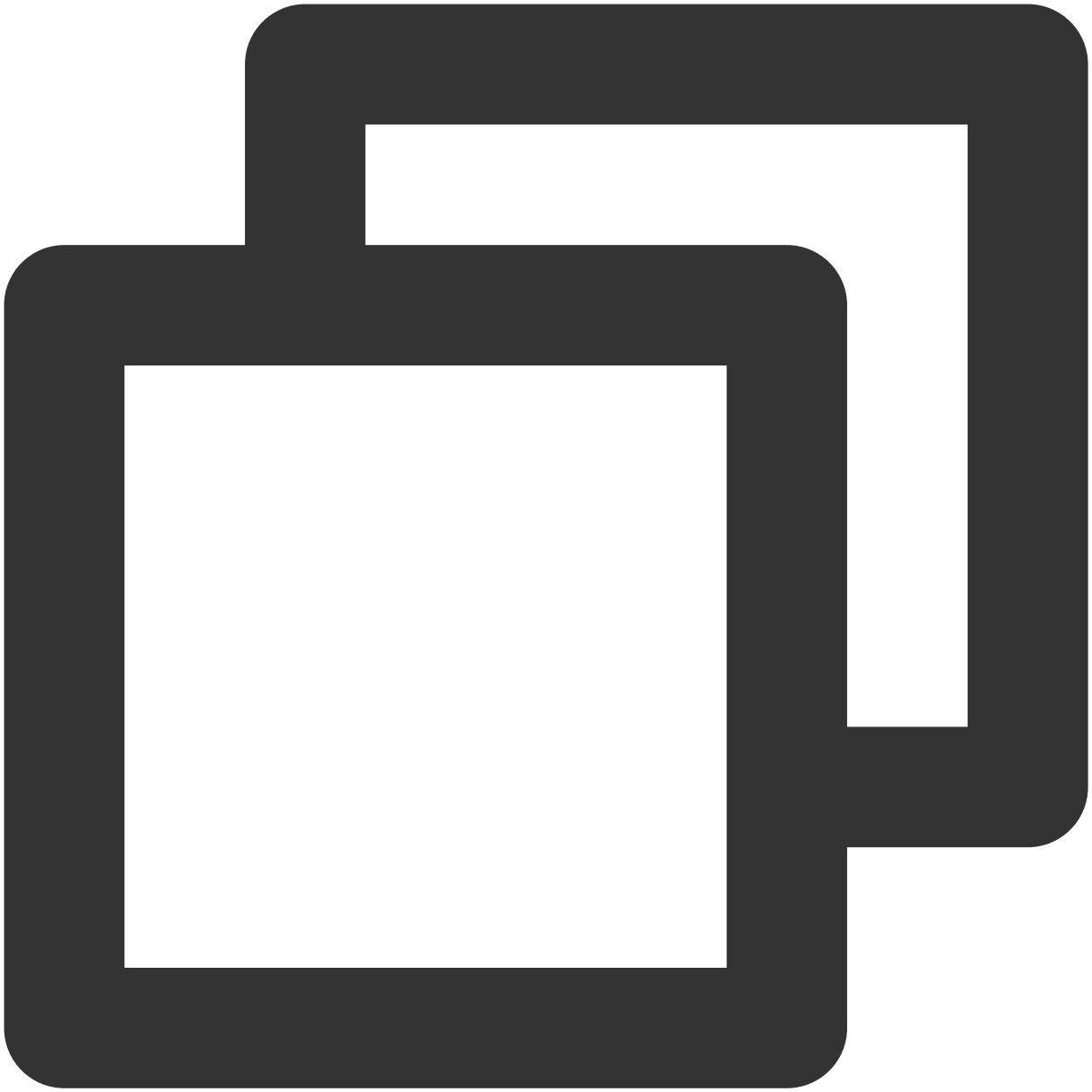
iOS

Android

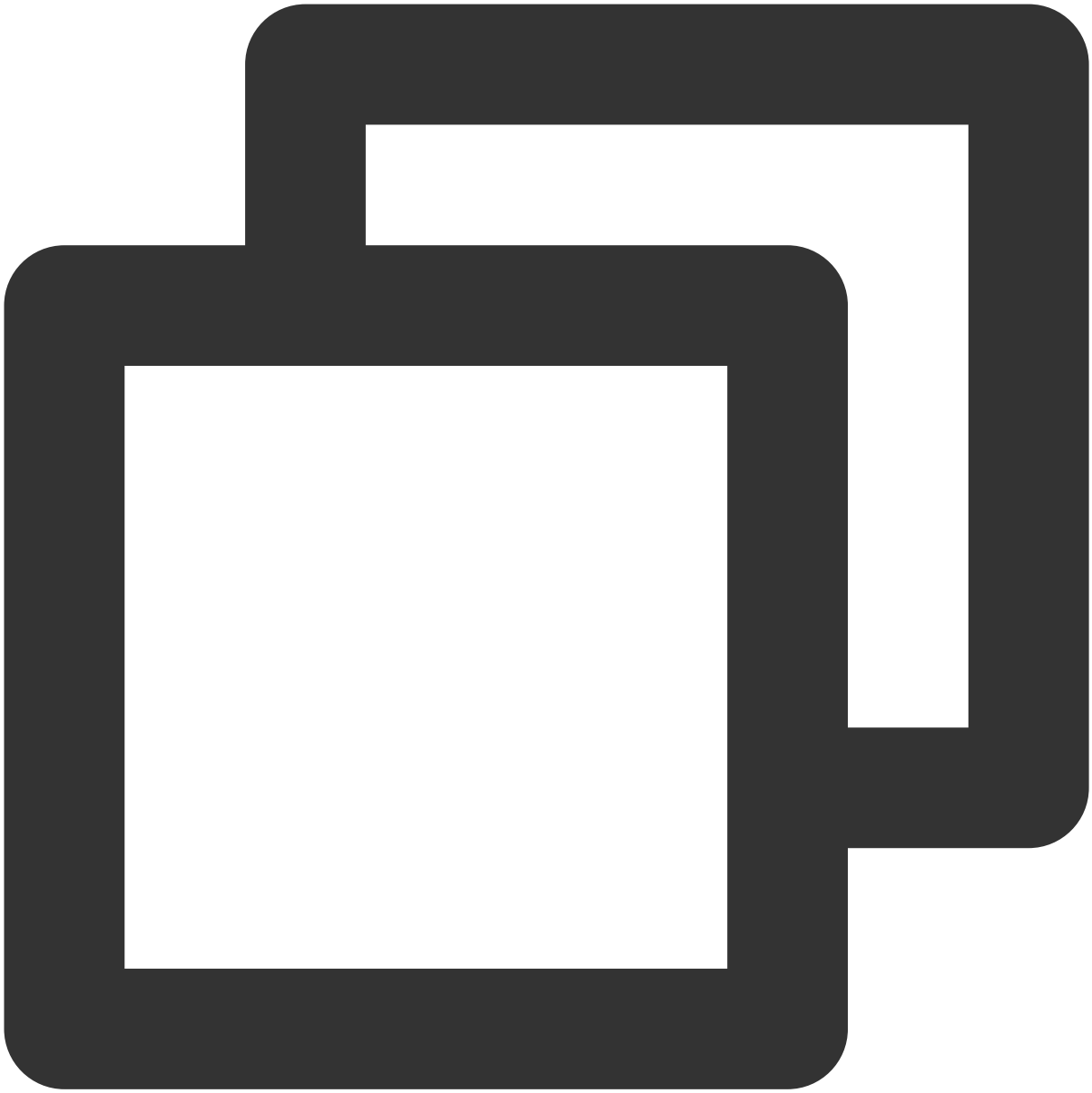
Unity



```
-(int) StopSendCustomData;
```



```
public abstract int StopSendCustomData();
```



```
public abstract int StopSendCustomData();
```

Returned values

If `StopSendCustomData` returns error code 1003, this API has been called and its operation is being performed by the SDK, so there is no need to call it again.

How to deal with the restrictions of corporate firewall

Last updated：2024-01-18 14:32:16

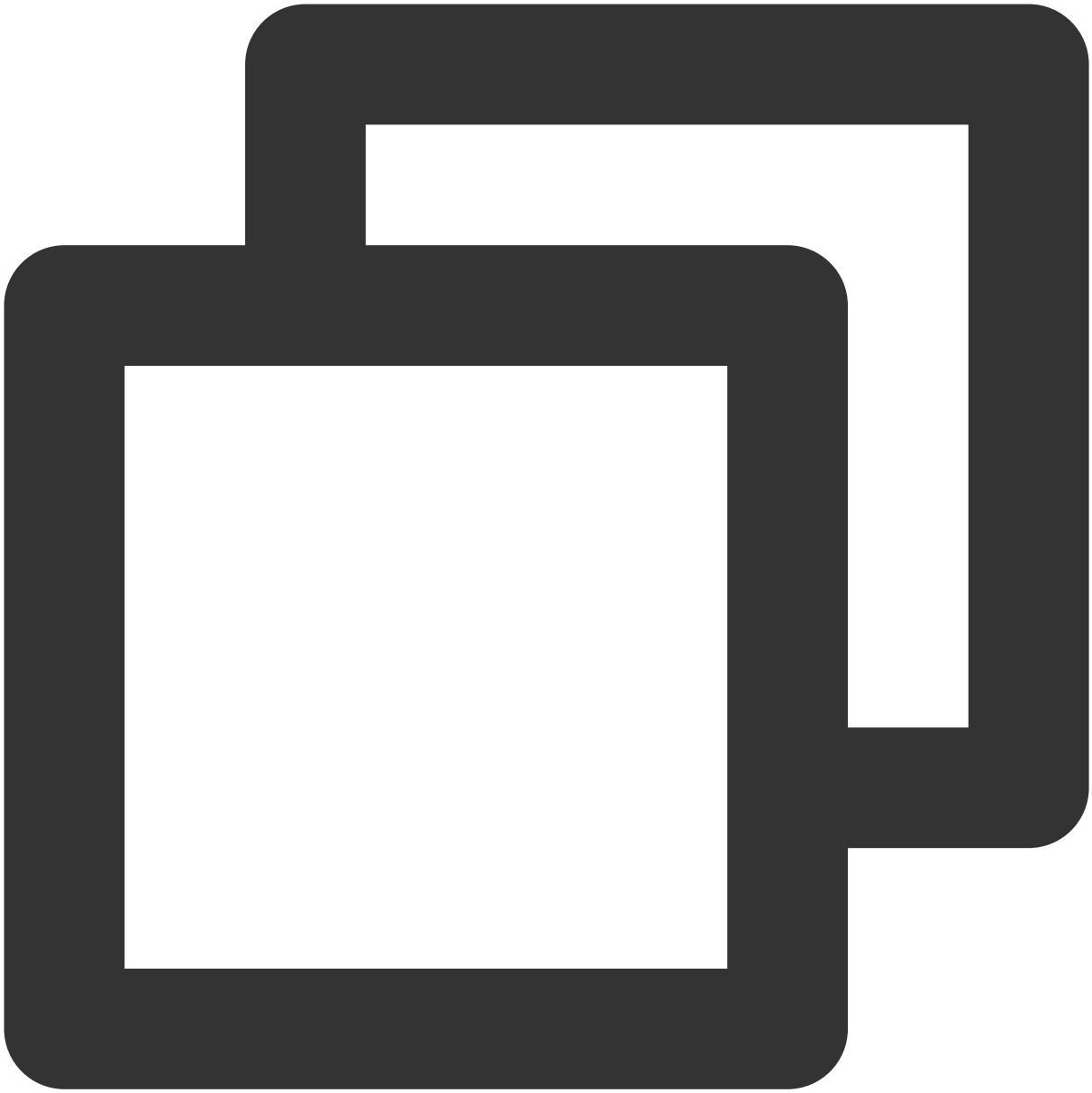
If your organization has restrictions placed on public network access, you need to configure the firewall allowlist accordingly before you can get access. The following describes the relevant rules:

Client Native SDK (v2.2 or Above)

Firewall port:

Port Type	Allowed Item
TCP port	443
UDP port	8000

Domain name allowlist:



```
tcloud.tim.qq.com
gmeconf.qcloud.com
yun.tim.qq.com
gmeosconf.qcloud.com
sg.global.gme.qcloud.com
```

Note:

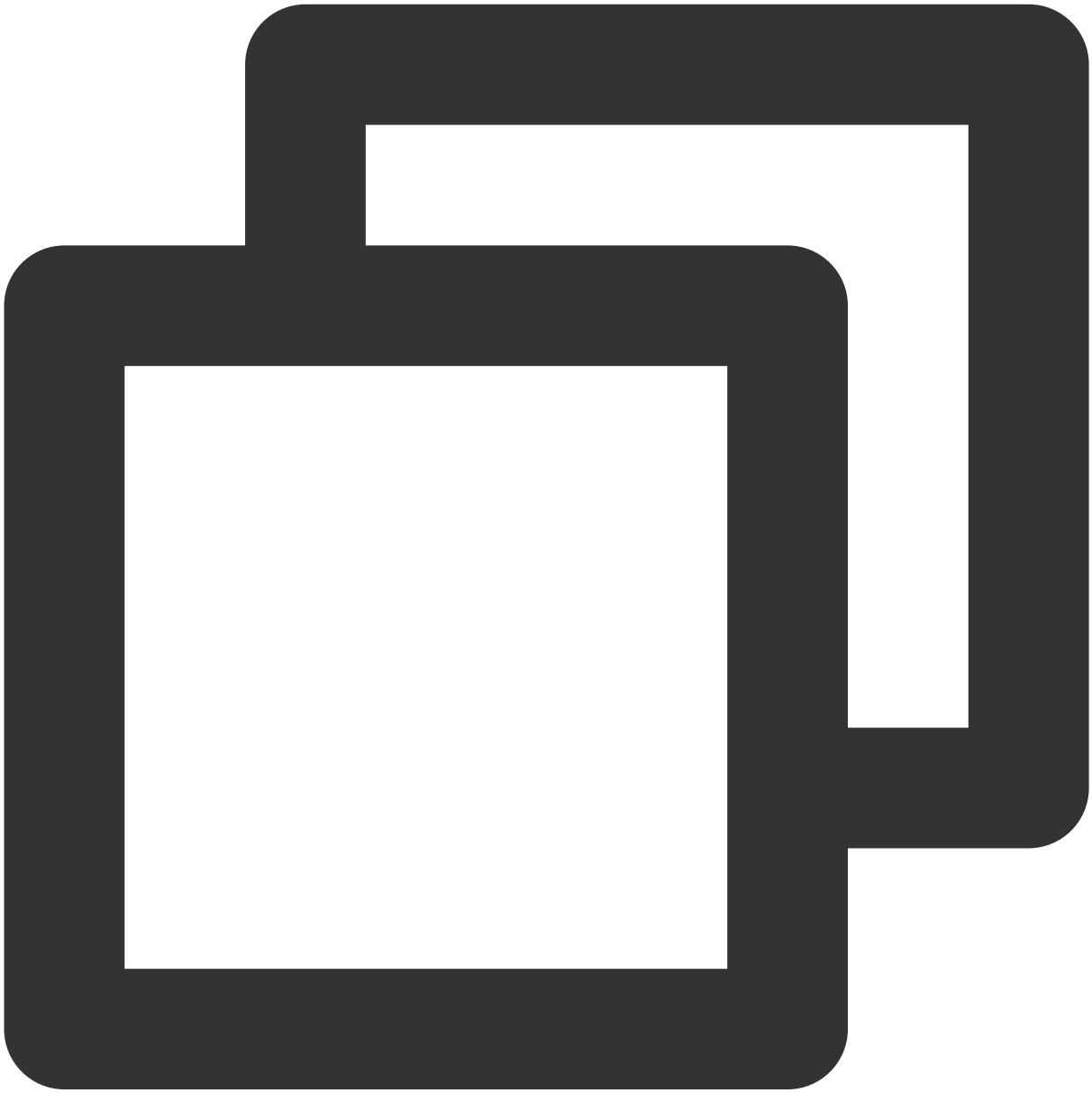
Tencent Cloud server IP addresses are dynamically updated, so we cannot provide you with a list of fixed IPs.

To use the GME SDK on Windows XP, you need to add the following items to the firewall allowlist:

Firewall port:

Port Type	Allowed Item
TCP port	15000

Domain name allowlist:



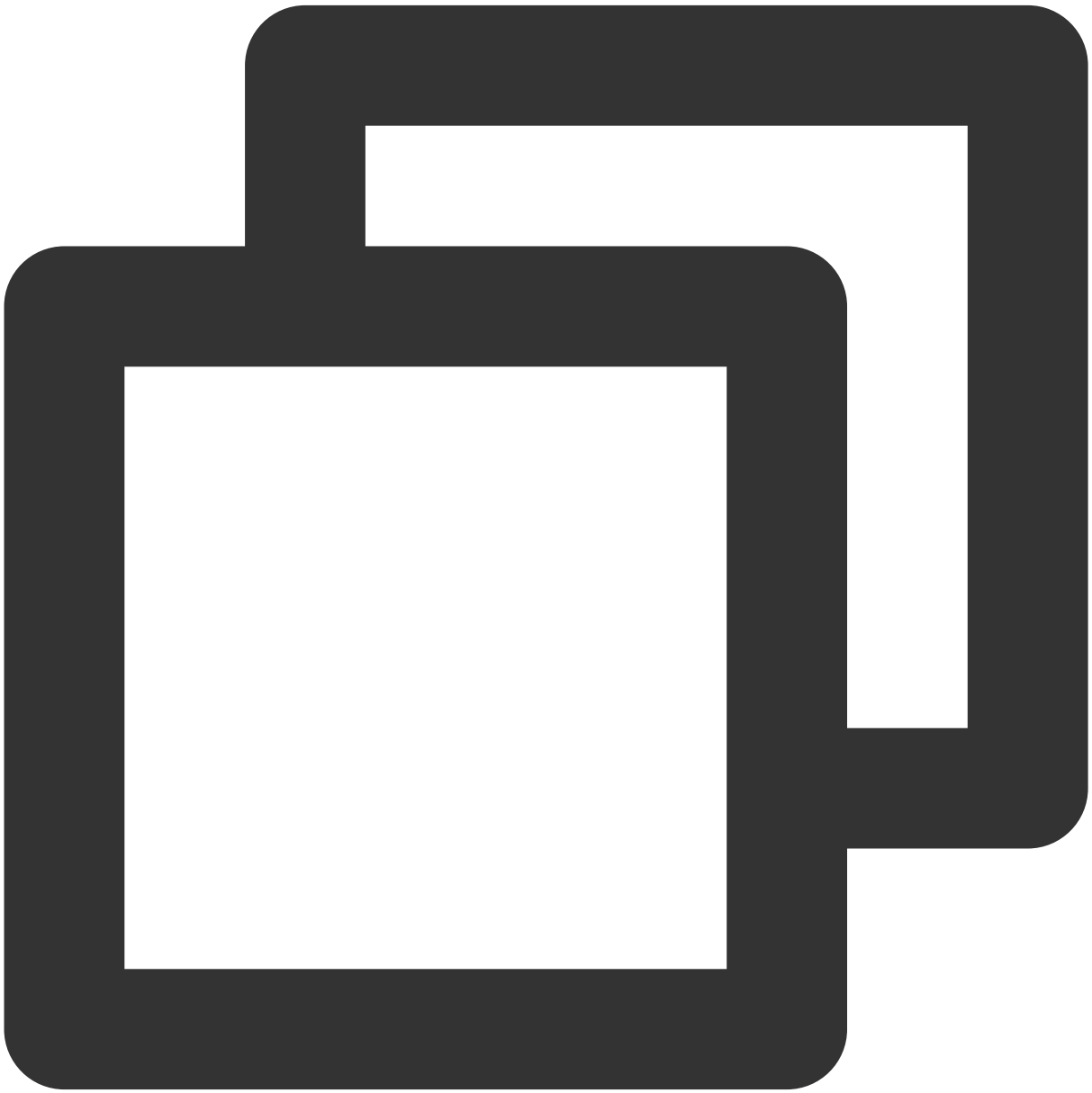
```
cloud.tim.qq.com
openmsf.3g.qq.com
```

Using the SDK for HTML5

Firewall port:

Port Type	Allowed Item
TCP port	443, 8687
UDP port	8000, 8800, 843, 443

Domain name allowlist:



```
qcloud.rtc.qq.com
```

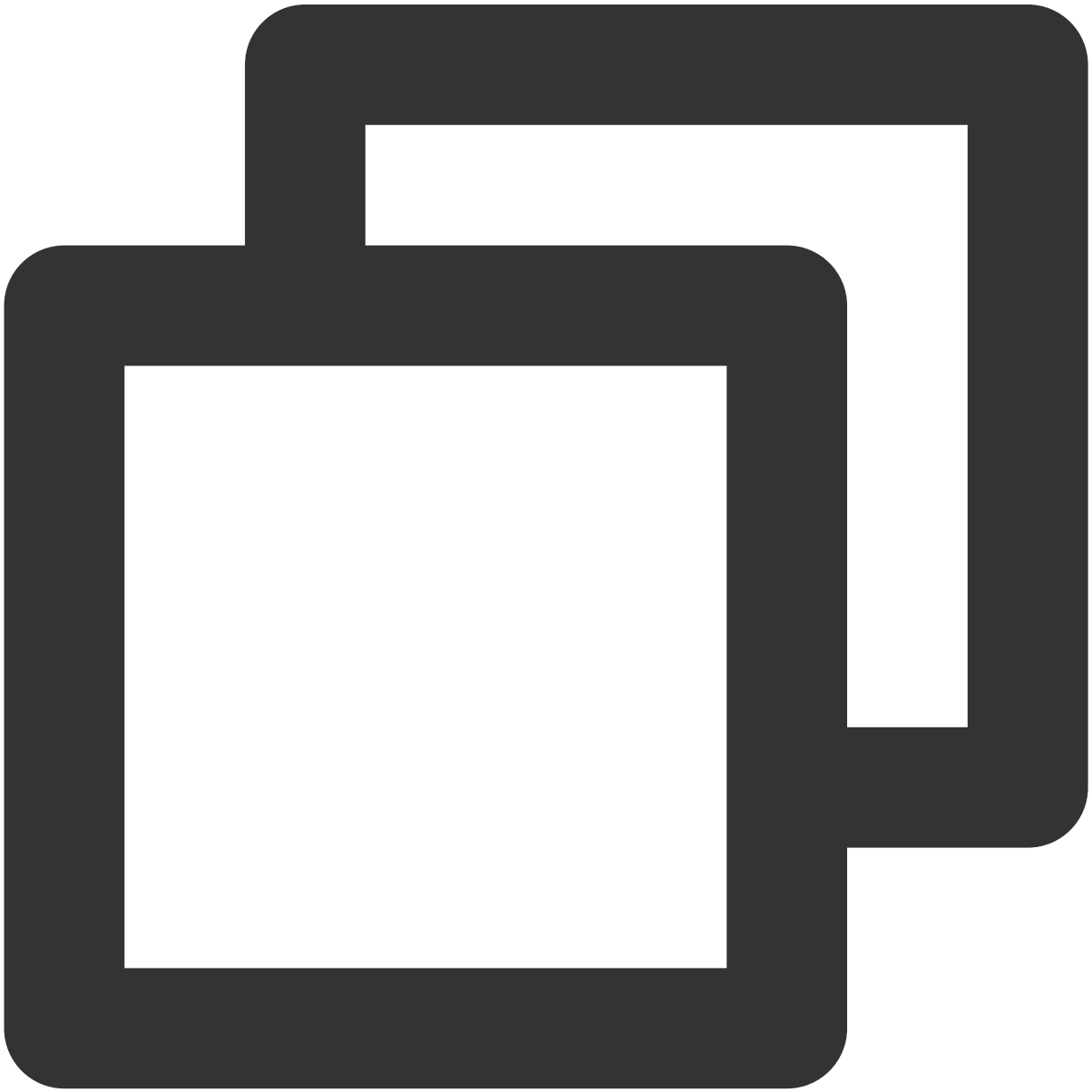
```
rtc.qcloud.qq.com
```

Using Voice Messaging and Speech-to-Text Conversion Service

Firewall port:

Port Type	Allowed Item
TCP port	80, 443

Domain name allowlist:



```
gmespeech.qcloud.com
yun.tim.qq.com
gmeconf.qcloud.com
```

Language Parameter Reference List

Last updated：2023-04-04 15:33:25

This document describes the language parameters of GME's speech-to-text, text translation, and text-to-speech services.

Speech-to-text, text translation, and text-to-speech services

Currently, the speech-to-text, text translation, and text-to-speech services support mainstream languages. The text translation API is `TranslateText` , and the text-to-speech API is `TextToSpeech` .

Note

If your application needs to use the text translation or text-to-speech feature, submit a ticket for application.

Language	Parameter	Description
普通话 （中国大陆）	cmn-Hans-CN	Chinese, Mandarin (Simplified, Chinese mainland)
國語 （中国台灣）	cmn-Hant-TW	Chinese, Mandarin (Traditional, Taiwan (China))
廣東話 （中国香港）	yue-Hant-HK	Chinese, Cantonese (Traditional, Hong Kong (China))
普通話 （中国香港）	cmn-Hans-HK	Chinese, Mandarin (Simplified, Hong Kong (China))
Afrikaans (Suid-Afrika)	af-ZA	Afrikaans (South Africa)
አማርኛ (ኢትዮጵያ)	am-ET	Amharic (Ethiopia)
Հայ (Հայաստան)	hy-AM	Armenian (Armenia)
Azərbaycan (Azərbaycan)	az-AZ	Azerbaijani (Azerbaijan)
Bahasa Indonesia (Indonesia)	id-ID	Indonesian (Indonesia)
Bahasa Melayu (Malaysia)	ms-MY	Malay (Malaysia)
বাংলা (বাংলাদেশ)	bn-BD	Bengali (Bangladesh)
বাংলা (ভারত)	bn-IN	Bengali (India)
Català (Espanya)	ca-ES	Catalan (Spain)
Čeština (Česká republika)	cs-CZ	Czech (Czech Republic)

Dansk (Danmark)	da-DK	Danish (Denmark)
Deutsch (Deutschland)	de-DE	German (Germany)
English (Australia)	en-AU	English (Australia)
English (Canada)	en-CA	English (Canada)
English (Ghana)	en-GH	English (Ghana)
English (Great Britain)	en-GB	English (United Kingdom)
English (India)	en-IN	English (India)
English (Ireland)	en-IE	English (Ireland)
English (Kenya)	en-KE	English (Kenya)
English (New Zealand)	en-NZ	English (New Zealand)
English (Nigeria)	en-NG	English (Nigeria)
English (Philippines)	en-PH	English (Philippines)
English (South Africa)	en-ZA	English (South Africa)
English (Tanzania)	en-TZ	English (Tanzania)
English (United States)	en-US	English (United States)
Español (Argentina)	es-AR	Spanish (Argentina)
Español (Bolivia)	es-BO	Spanish (Bolivia)
Español (Chile)	es-CL	Spanish (Chile)
Español (Colombia)	es-CO	Spanish (Colombia)
Español (Costa Rica)	es-CR	Spanish (Costa Rica)
Español (Ecuador)	es-EC	Spanish (Ecuador)
Español (El Salvador)	es-SV	Spanish (El Salvador)
Español (España)	es-ES	Spanish (Spain)
Español (Estados Unidos)	es-US	Spanish (United States)
Español (Guatemala)	es-GT	Spanish (Guatemala)

Español (Honduras)	es-HN	Spanish (Honduras)
Español (México)	es-MX	Spanish (Mexico)
Español (Nicaragua)	es-NI	Spanish (Nicaragua)
Español (Panamá)	es-PA	Spanish (Panama)
Español (Paraguay)	es-PY	Spanish (Paraguay)
Español (Perú)	es-PE	Spanish (Peru)
Español (Puerto Rico)	es-PR	Spanish (Puerto Rico)
Español (República Dominicana)	es-DO	Spanish (Dominican Republic)
Español (Uruguay)	es-UY	Spanish (Uruguay)
Español (Venezuela)	es-VE	Spanish (Venezuela)
Euskara (Espainia)	eu-ES	Basque (Spain)
Filipino (Pilipinas)	fil-PH	Filipino (Philippines)
Français (Canada)	fr-CA	French (Canada)
Français (France)	fr-FR	French (France)
Galego (España)	gl-ES	Galician (Spain)
ქართული (საქართველო)	ka-GE	Georgian (Georgia)
ગુજરાતી (ભારત)	gu-IN	Gujarati (India)
Hrvatski (Hrvatska)	hr-HR	Croatian (Croatia)
IsiZulu (Ningizimu Afrika)	zu-ZA	Zulu (South Africa)
Íslenska (Ísland)	is-IS	Icelandic (Iceland)
Italiano (Italia)	it-IT	Italian (Italy)
Jawa (Indonesia)	jv-ID	Javanese (Indonesia)
( )	kn-IN	Kannada (India)
( )	km-KH	Khmer (Cambodia)
( )	lo-LA	Lao (Laos)

Latviešu (latviešu)	lv-LV	Latvian (Latvia)
Lietuvių (Lietuva)	lt-LT	Lithuanian (Lithuania)
Magyar (Magyarország)	hu-HU	Hungarian (Hungary)
മലയാളം (ഇന്ത്യ)	ml-IN	Malayalam (India)
मराठी (भारत)	mr-IN	Marathi (India)
Nederlands (Nederland)	nl-NL	Dutch (Netherlands)
नेपाली (नेपाल)	ne-NP	Nepali (Nepal)
Norsk bokmål (Norge)	nb-NO	Norwegian Bokmål (Norway)
Polski (Polska)	pl-PL	Polish (Poland)
Português (Brasil)	pt-BR	Portuguese (Brazil)
Português (Portugal)	pt-PT	Portuguese (Portugal)
Română (România)	ro-RO	Romanian (Romania)
සිංහල (ශ්‍රී ලංකාව)	si-LK	Sinhalese (Sri Lanka)
Slovenčina (Slovensko)	sk-SK	Slovak (Slovakia)
Slovenščina (Slovenija)	sl-SI	Slovenian (Slovenia)
Urang (Indonesia)	su-ID	Sundanese (Indonesia)
Swahili (Tanzania)	sw-TZ	Swahili (Tanzania)
Swahili (Kenya)	sw-KE	Swahili (Kenya)
Suomi (Suomi)	fi-FI	Finnish (Finland)
Svenska (Sverige)	sv-SE	Swedish (Sweden)
தமிழ் (இந்தியா)	ta-IN	Tamil (India)
தமிழ் (சிங்கப்பூர்)	ta-SG	Tamil (Singapore)
தமிழ் (இலங்கை)	ta-LK	Tamil (Sri Lanka)
தமிழ் (மலேசியா)	ta-MY	Tamil (Malaysia)
( )	te-IN	Telugu (India)

Tiếng Việt (Việt Nam)	vi-VN	Vietnamese (Vietnam)
Türkçe (Türkiye)	tr-TR	Turkish (Turkey)
(اردو) پاکستان	ur-PK	Urdu (Pakistan)
(اردو) بھارت	ur-IN	Urdu (India)
Ελληνικά (Ελλάδα)	el-GR	Greek (Greece)
Български (България)	bg-BG	Bulgarian (Bulgaria)
Русский (Россия)	ru-RU	Russian (Russia)
Српски (Србија)	sr-RS	Serbian (Serbia)
Українська (Україна)	uk-UA	Ukrainian (Ukraine)
(עברית) ישראל	he-IL	Hebrew (Israel)
(العربية) إسرائيل	ar-IL	Arabic (Israel)
(العربية) الأردن	ar-JO	Arabic (Jordan)
(العربية) الإمارات	ar-AE	Arabic (United Arab Emirates)
(العربية) البحرين	ar-BH	Arabic (Bahrain)
(العربية) الجزائر	ar-DZ	Arabic (Algeria)
(العربية) السعودية	ar-SA	Arabic (Saudi Arabia)
(العربية) العراق	ar-IQ	Arabic (Iraq)
(العربية) الكويت	ar-KW	Arabic (Kuwait)
(العربية) المغرب	ar-MA	Arabic (Morocco)
(العربية) تونس	ar-TN	Arabic (Tunisia)
(العربية) عُمان	ar-OM	Arabic (Oman)
(العربية) فلسطين	ar-PS	Arabic (Palestine)
(العربية) قطر	ar-QA	Arabic (Qatar)
(العربية) لبنان	ar-LB	Arabic (Lebanon)
(العربية) مصر	ar-EG	Arabic (Egypt)

(ایران)فارسی	fa-IR	Persian (Iran)
हिन्दी (भारत)	hi-IN	Hindi (India)
ไทย (ประเทศไทย)	th-TH	Thai (Thailand)
한국어 (대한민국)	ko-KR	Korean (South Korea)

Integrating GME Chat Room Management

Last updated：2023-04-27 17:14:53

Note: Currently, GME 3.x does not support room management.
This document describes use cases of the room management feature and how to quickly connect to the feature.

Overview

You can use client room management APIs to manage room members and member mic-on/off status easily.

Use Cases

For example, the host in Werewolf can use `EnableMic` to control other players to enable their mic to speak. If a player is dead and doesn't need to listen to room members or speak with a mic, the host use the `ForbidUserOperation` API to forbid the player from manipulating devices.

Prerequisites

You have activated the voice chat service as instructed in Activating Services.
You have integrated the GME SDK, including core APIs and voice chat APIs. For more information, see Quick Integration of Native SDK, Quick Integration of SDK for Unity, and Quick Integration of SDK for Unreal Engine.
Note
This feature is not supported by the SDK for HTML5.

Integration

Class name: ITMGRoomManager

GME chat room management can be used only when there are one or more members in a room, and only to modify the status of a room member.
All the callbacks for API responses are processed through `ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR`. For callback details, see Processing Callbacks.

API list

Type	API
Capturing control APIs	EnableMic, EnableAudioCaptureDevice, and EnableAudioSend
Playback control APIs	EnableSpeaker, EnableAudioPlayDevice, and EnableAudioRecv
Device status acquisition APIs	GetMicState and GetSpeakerState
Sensitive APIs	ForbidUserOperation

Capture Management APIs

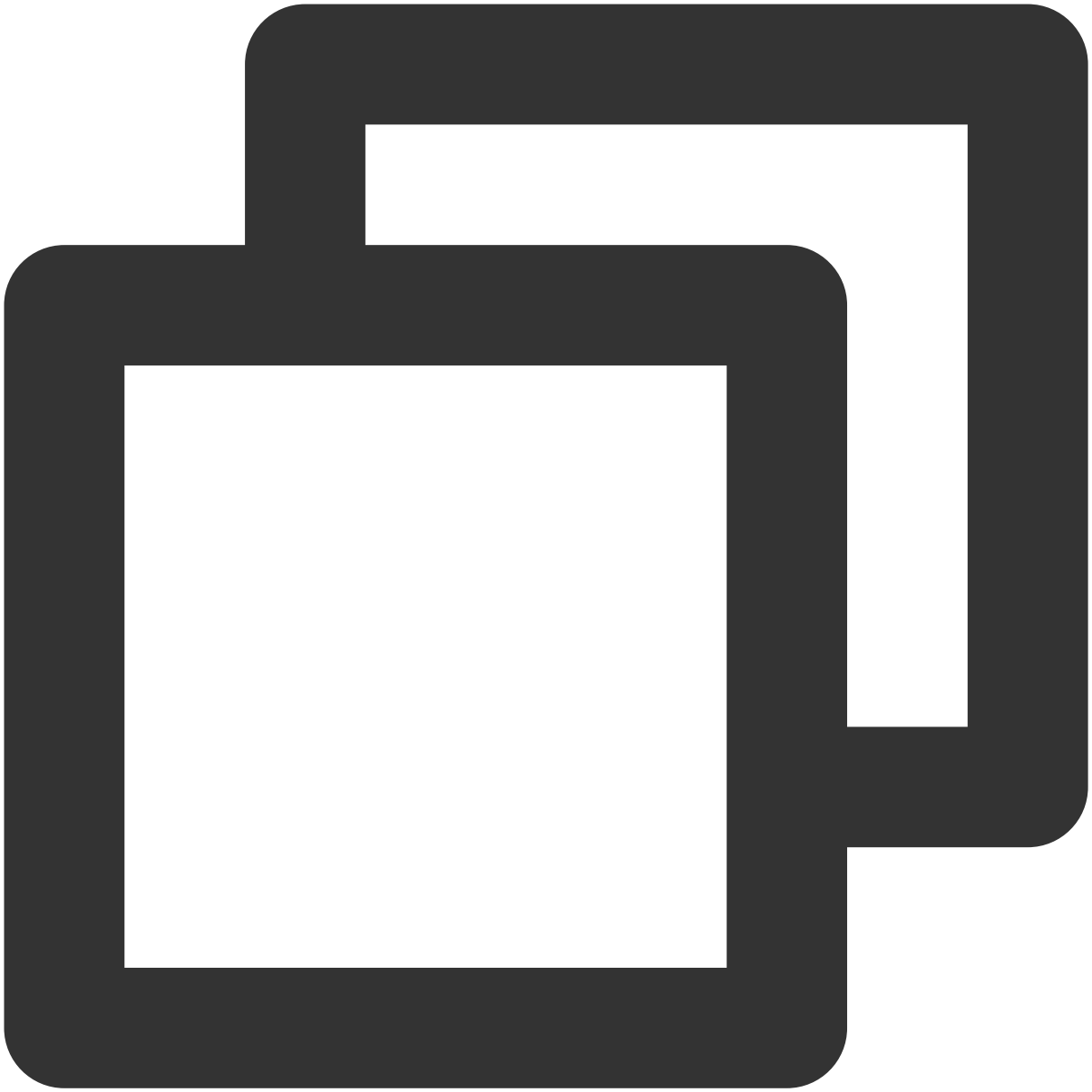
The capturing management APIs are used to manage the **mic**, **audio upstreaming**, and **capturing device**. Mic management is equivalent to audio upstreaming and capturing device management.

Capturing management

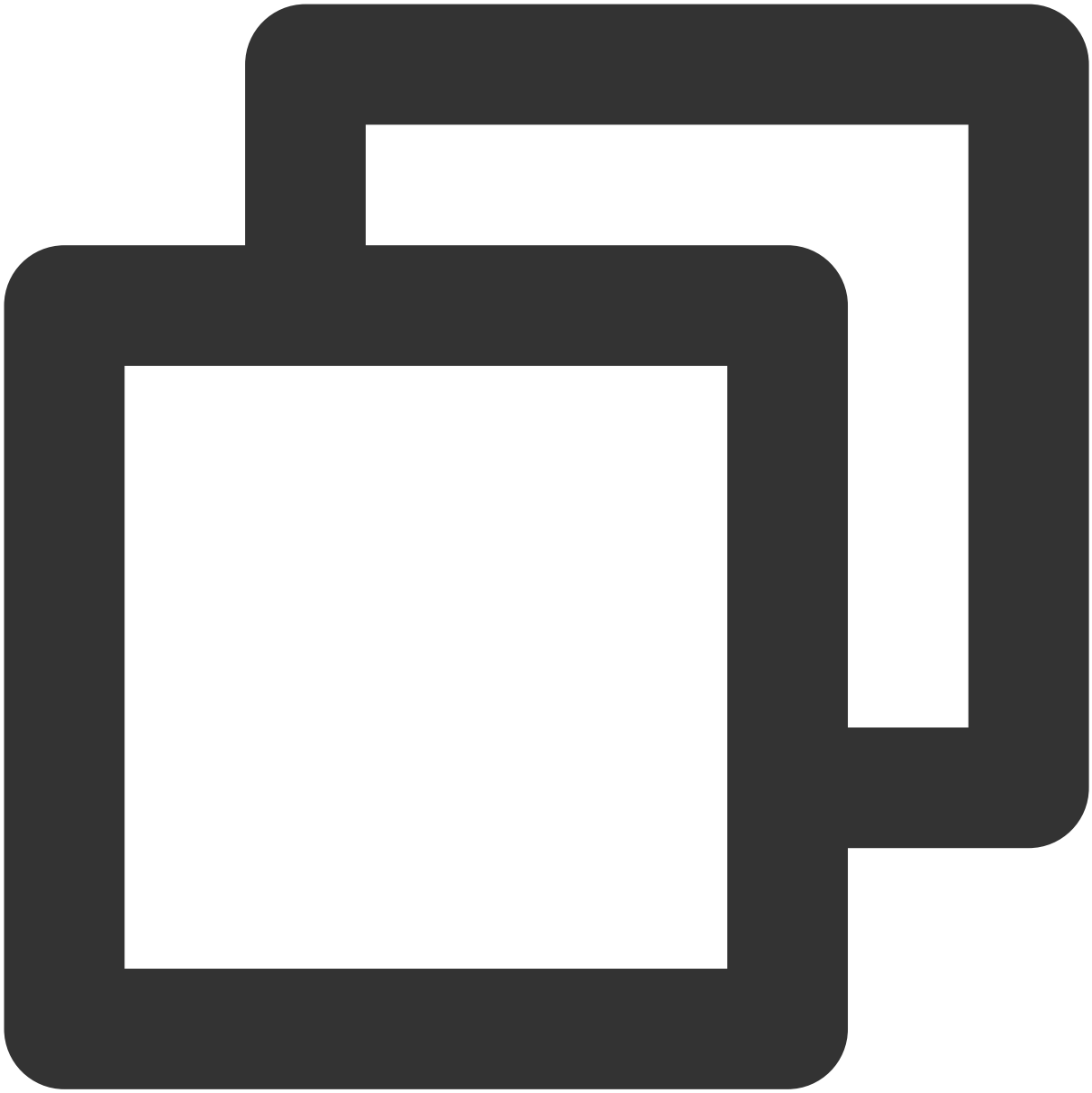
This API (EnableMic) is used to enable/disable the microphone for a user in a chat room.
Calling `EnableMic` is equivalent to calling both `EnableAudioSend` and `EnableAudioCaptureDevice`.

Function prototype

Android
iOS



```
public abstract int EnableMic(boolean isEnabled,String receiverID);
```



```
-(QAVResult)EnableMic:(BOOL)enable Receiver:(NSString *)receiverID;
```

Parameter	Type	Description
enable	BOOL	`YES` : Enable the mic for a user; `NO` : Disable the mic for a user
receiverID	NSString*	Specifies the `OpenId` of the user

Callback

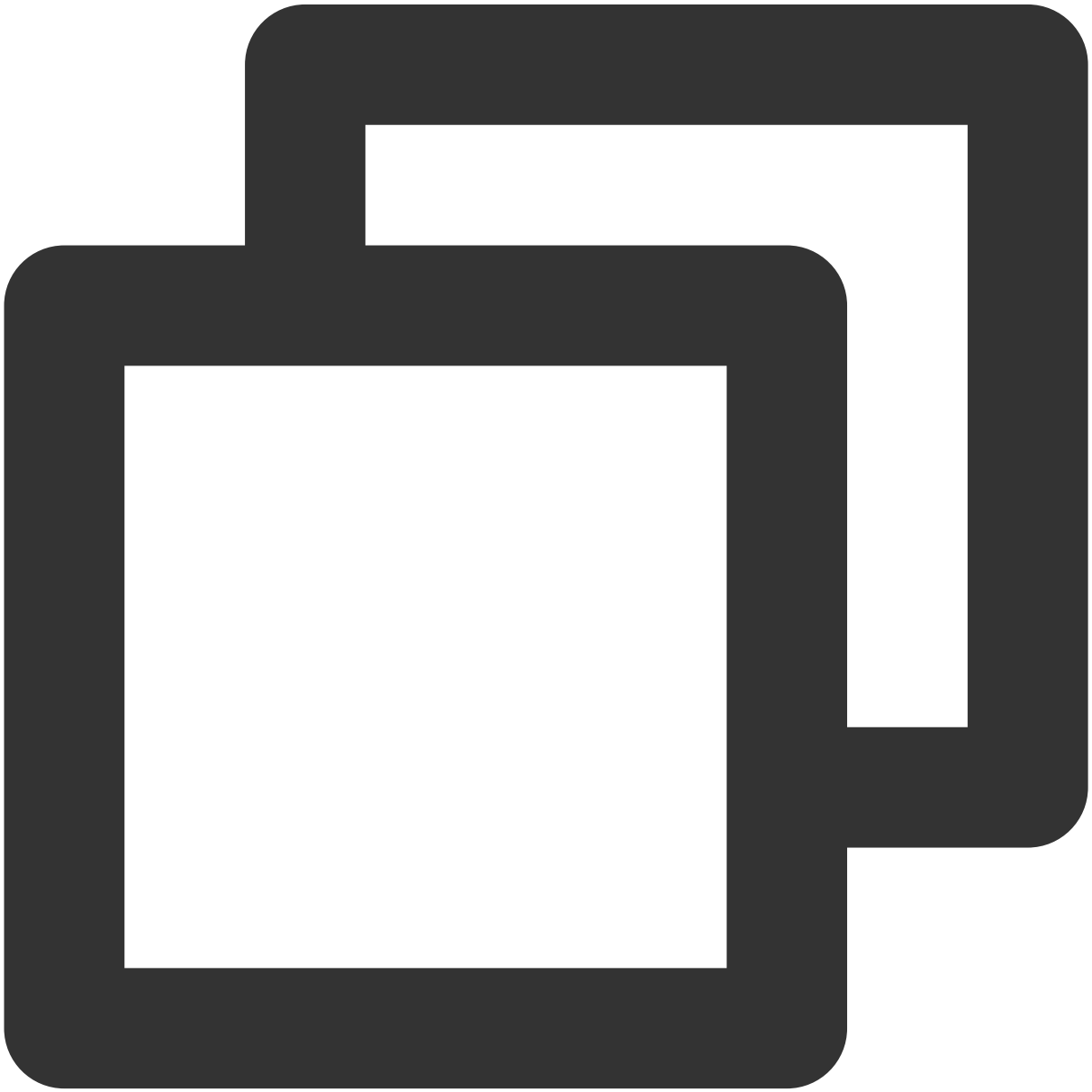
The callback parameter is ITMG_ROOM_MANAGEMENT_MIC_OP.

Audio stream sending management

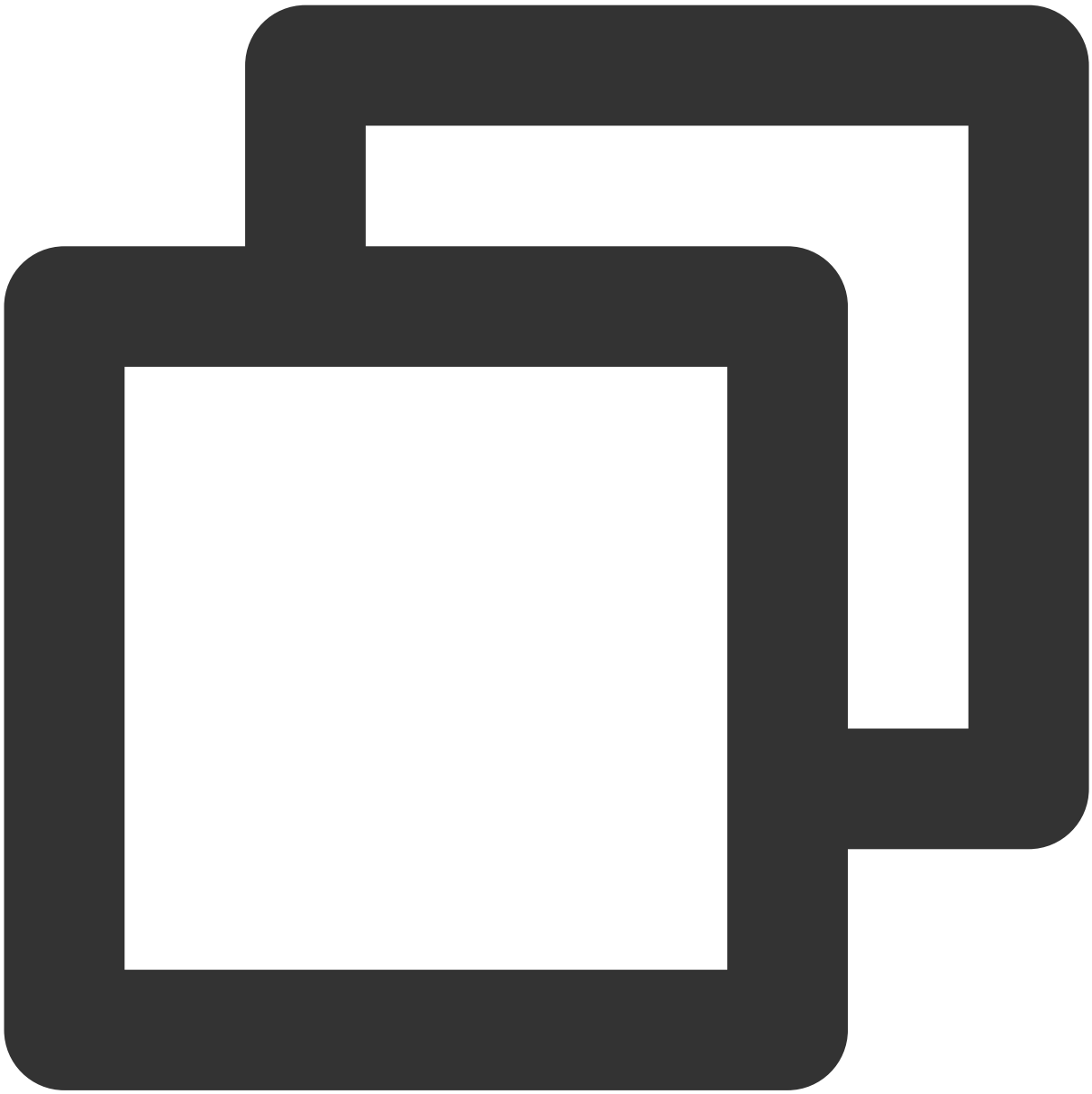
This API (EnableAudioSend) is used to enable/disable audio upstreaming for a user in a chat room but will not affect the microphone capture.

Function prototype

Android
iOS



```
public abstract int EnableAudioSend(boolean isEnabled,String receiverID);
```



```
-(QAVResult)EnableAudioSend:(BOOL)enable Receiver:(NSString *)receiverID;
```

Parameter	Type	Description
enable	BOOL	`YES` : Enable upstreaming for a user; `NO` : Disable upstreaming for a user
receiverID	NSString*	Specifies the `OpenId` of the user

Callback

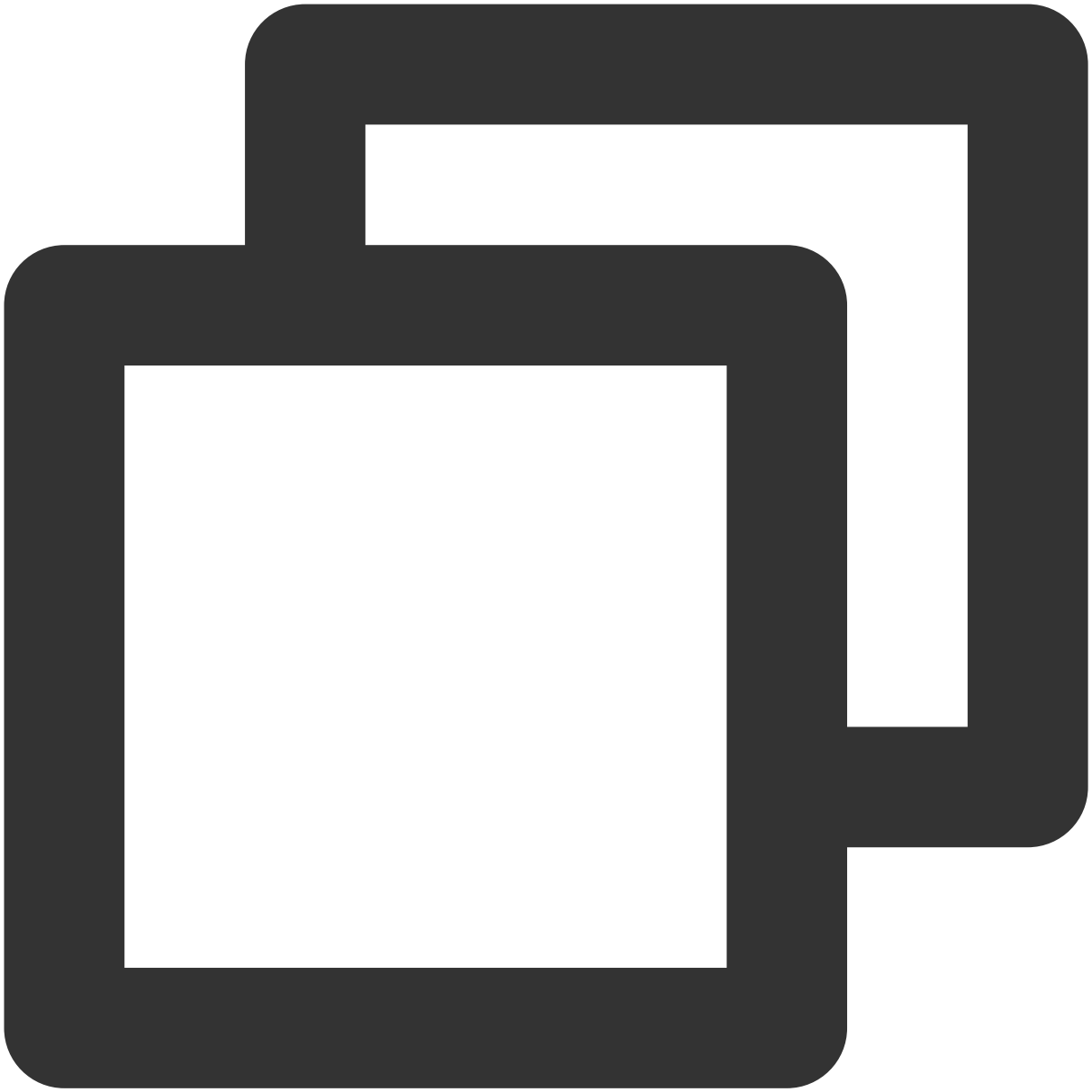
The callback parameter is ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP.

Audio capturing device management

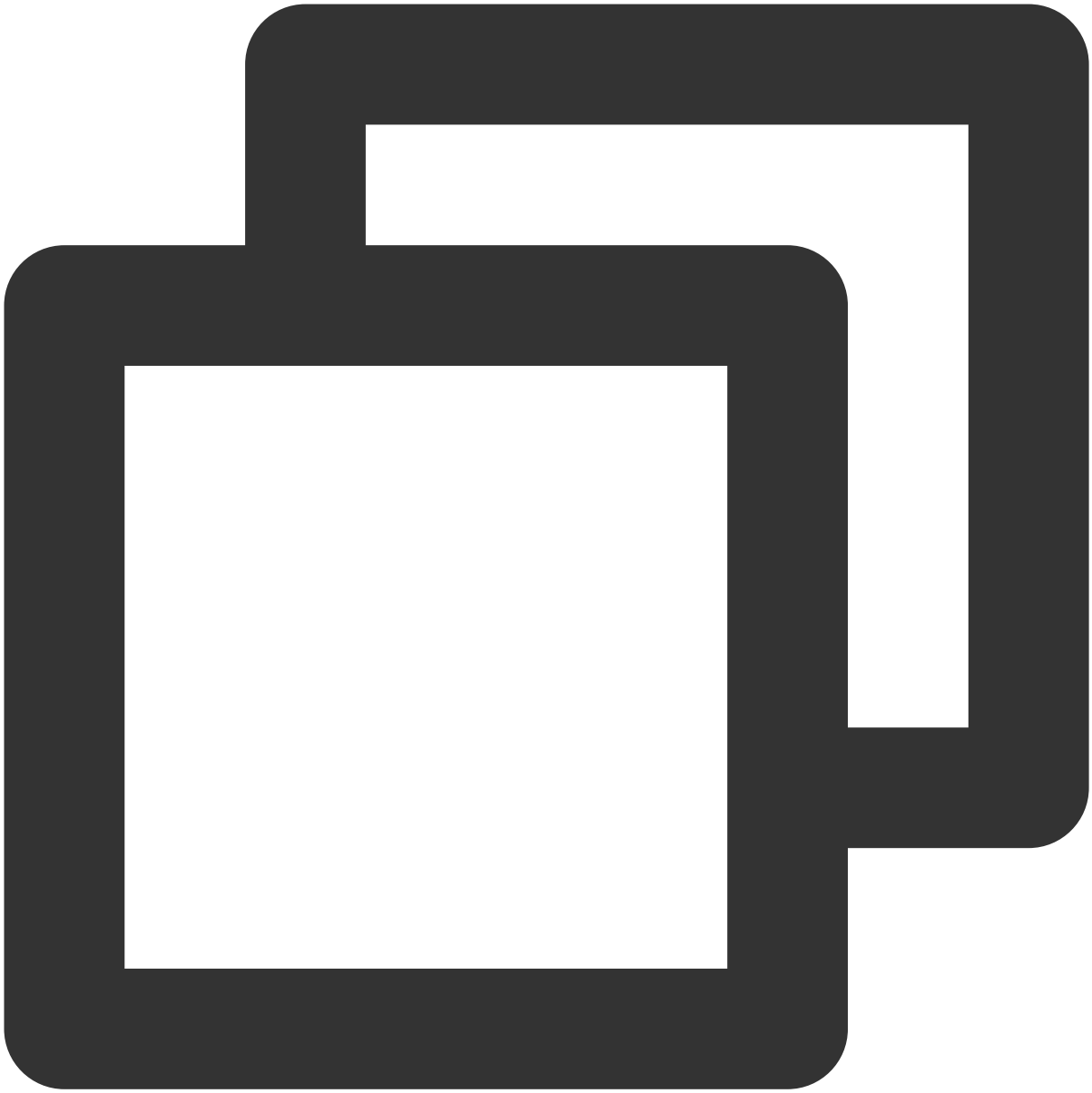
This API (EnableAudioCaptureDevice) is used to enable/disable the audio capture device for a user in a chat room but will not affect audio upstreaming.

Function prototype

Android
iOS



```
public abstract int EnableAudioCaptureDevice(boolean isEnabled,String receiverID);
```



```
-(QAVResult)EnableAudioCaptureDevice:(BOOL)enabled Receiver:(NSString *)receiverID;
```

Parameter	Type	Description
enable	BOOL	`YES` : Enable the audio capturing device for a user; `NO` : Disable the audio capturing device for a user
receiverID	NSString*	Specifies the `OpenId` of the user

Callback

The callback parameter is ITMG_ROOM_MANAGEMENT_CAPTURE_OP.

Playback Management APIs

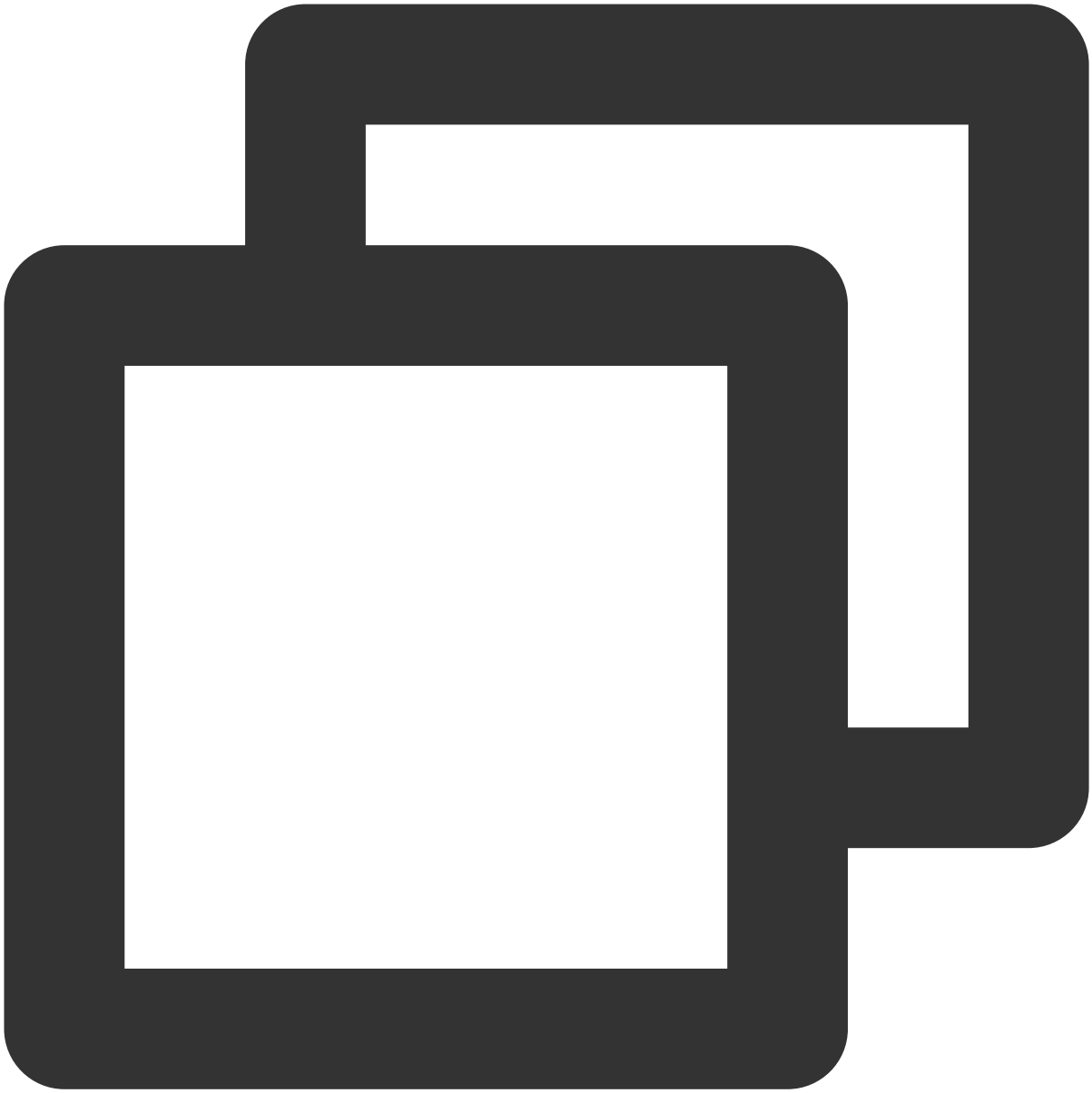
The playback management APIs are used to manage the **speaker**, **audio downstreaming**, and **playback device**. Speaker management is equivalent to audio downstreaming and playback device management.

Playback management

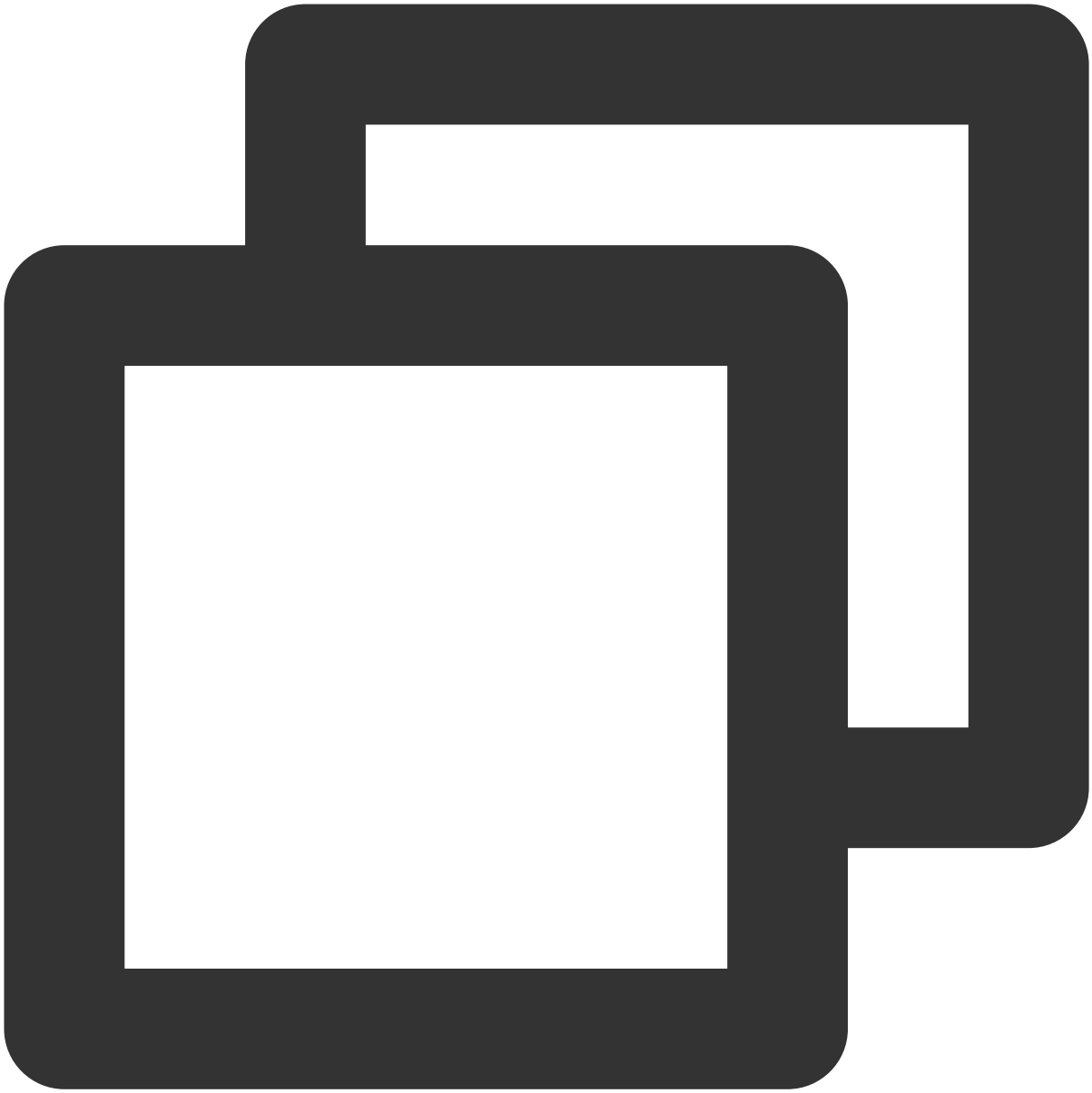
This API (EnableSpeaker) is used to enable/disable the speaker for a user to hear the sound in a chat room. Calling `EnableSpeaker` is equivalent to calling both `EnableAudioRecv` and `EnableAudioPlayDevice`.

Function prototype

Android
iOS



```
public abstract int EnableSpeaker(boolean isEnabled,String receiverID);
```



```
-(QAVResult)EnableSpeaker:(BOOL)enable Receiver:(NSString *)receiverID;
```

Parameter	Type	Description
enable	BOOL	`YES` : Enable the speaker for a user; `NO` : Disable the speaker for a user
receiverID	NSString*	Specifies the `OpenId` of the user

Callback

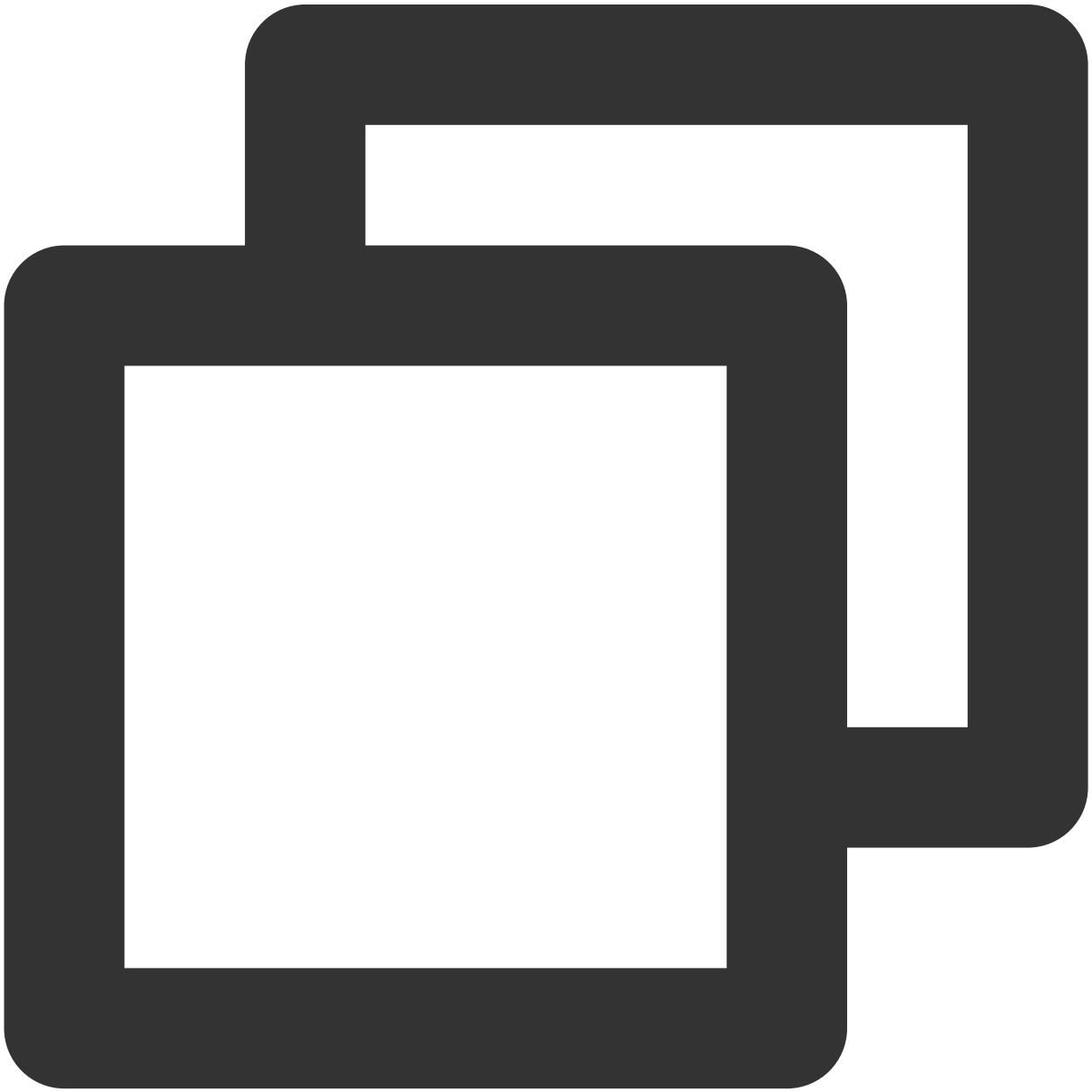
The callback parameter is ITMG_ROOM_MANAGEMENT_SPEAKER_OP.

Audio stream receiving management

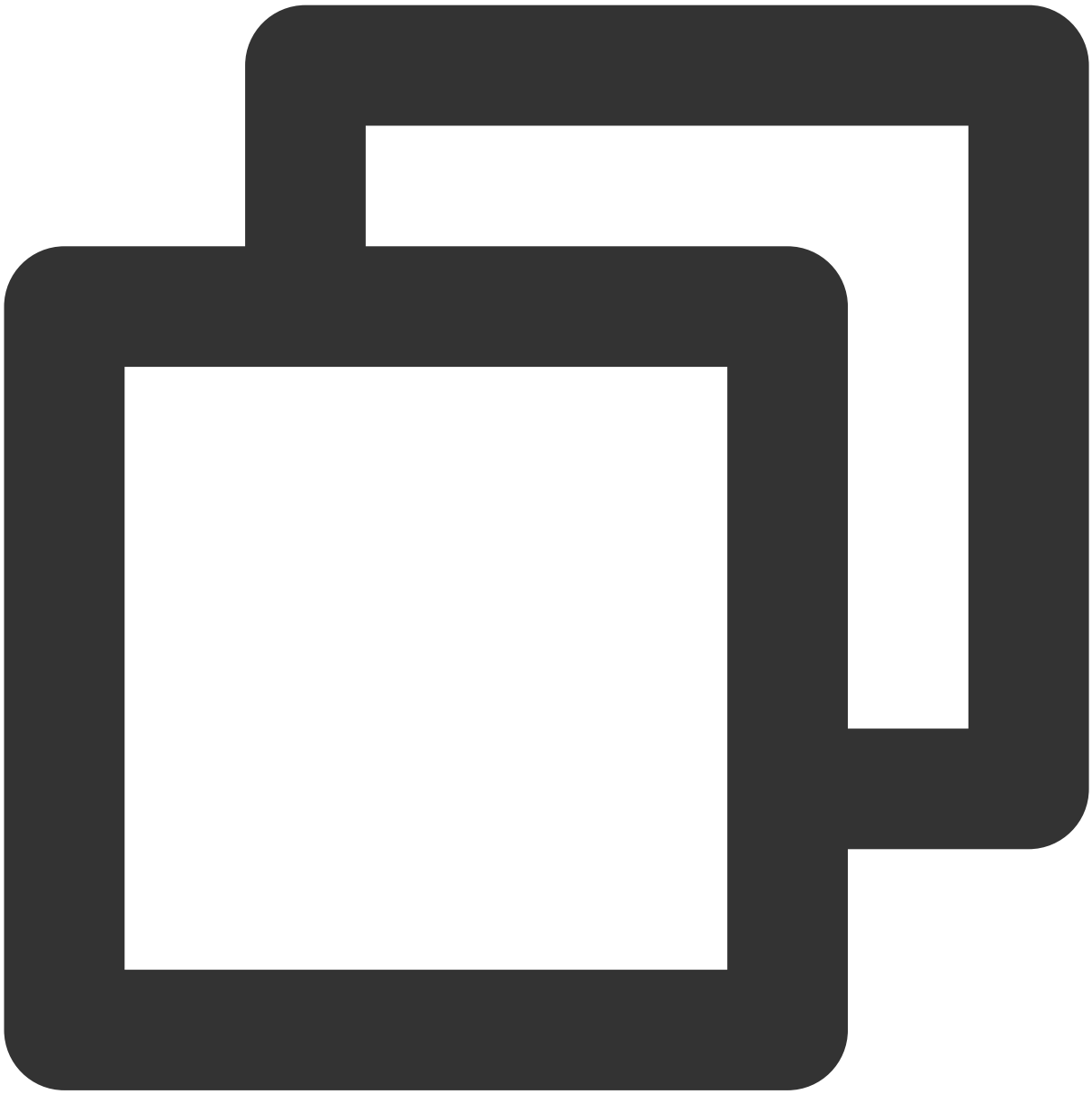
This API (EnableAudioRecv) is used to enable/disable audio downstreaming for a user but will not affect the playback device.

Function prototype

Android
iOS



```
public abstract int EnableAudioRecv(boolean isEnabled,String receiverID);
```



```
-(QAVResult)EnableAudioRecv:(BOOL)enabled Receiver:(NSString *)receiverID;
```

Parameter	Type	Description
enable	BOOL	`YES` : Enable audio downstreaming for a user; `NO` : Disable audio downstreaming for a user
receiverID	NSString*	Specifies the `OpenId` of the user

Callback

The callback parameter is ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP.

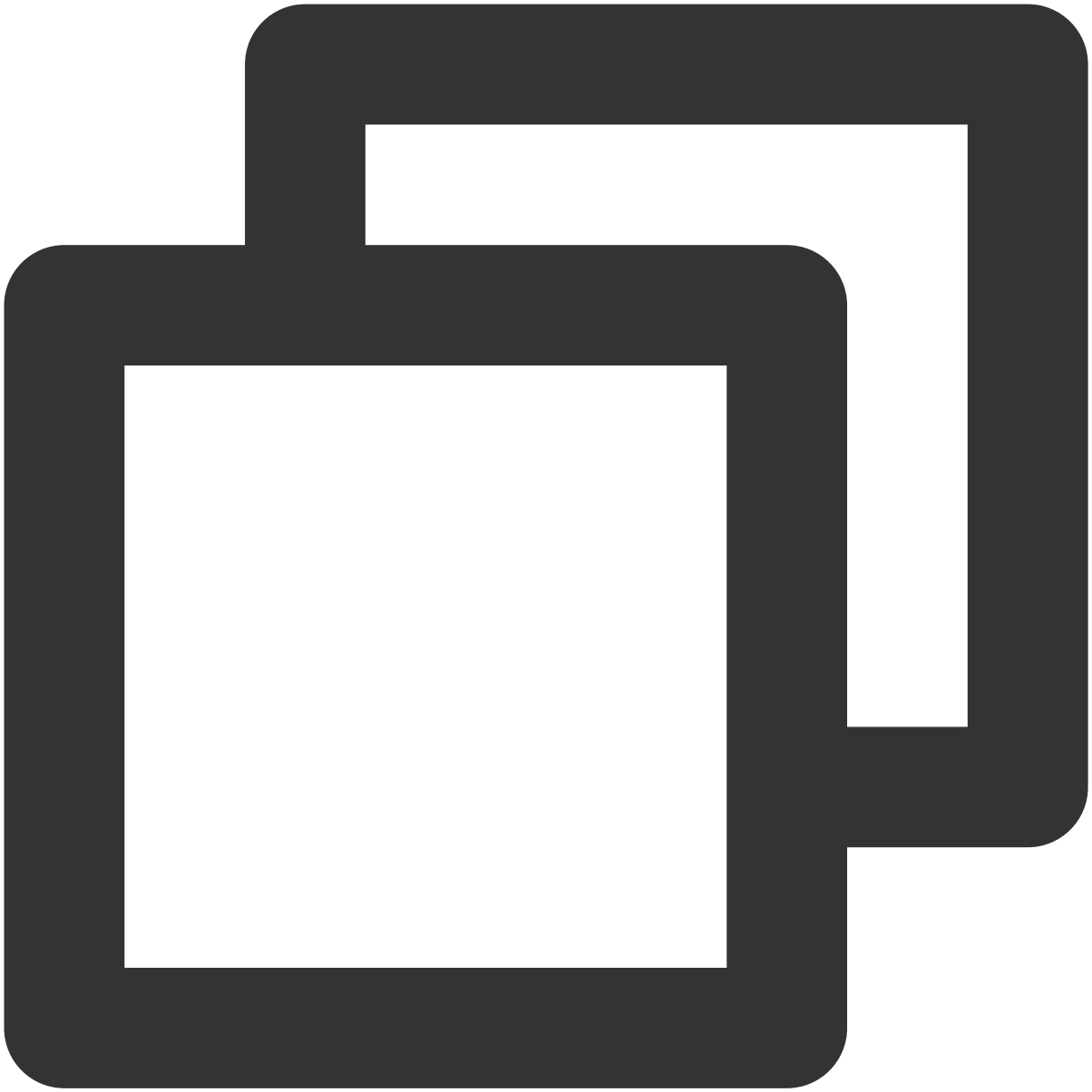
Audio playback device management

This API (EnableAudioPlayDevice) is used to enable/disable the audio playback device for a user but will not affect audio downstreaming.

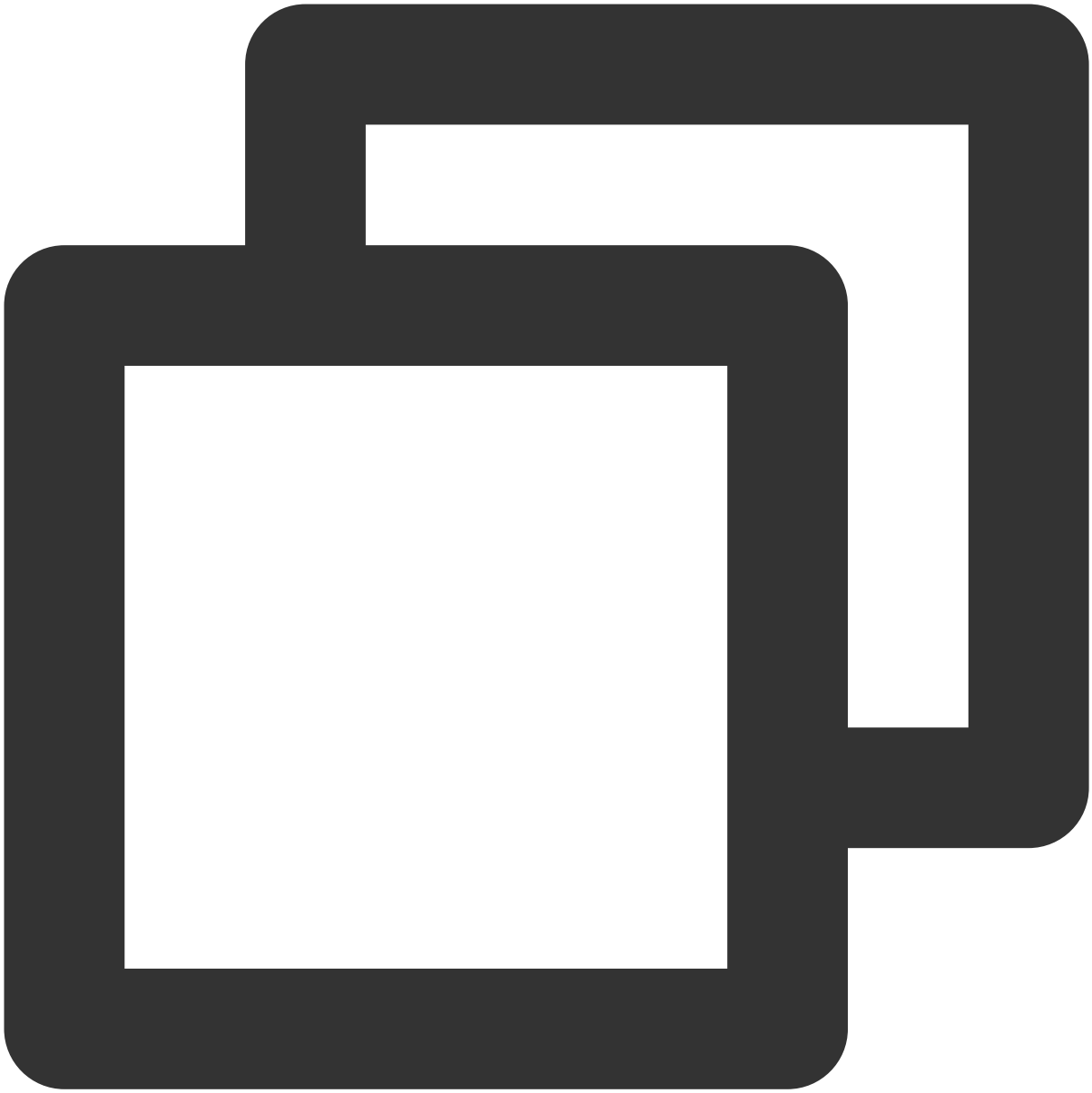
Function prototype

Android

iOS



```
public abstract int EnableAudioPlayDevice(boolean isEnabled,String receiverID);
```



```
-(QAVResult)EnableAudioPlayDevice:(BOOL)enabled Receiver:(NSString *)receiverID;
```

Parameter	Type	Description
enable	BOOL	`YES` : Enable the audio playback device for a user; `NO` : Disable the audio playback device for a user
receiverID	NSString*	Specifies the `OpenId` of the user

Callback

The callback parameter is ITMG_ROOM_MANAGEMENT_PLAY_OP.

APIs for Obtaining the User Status

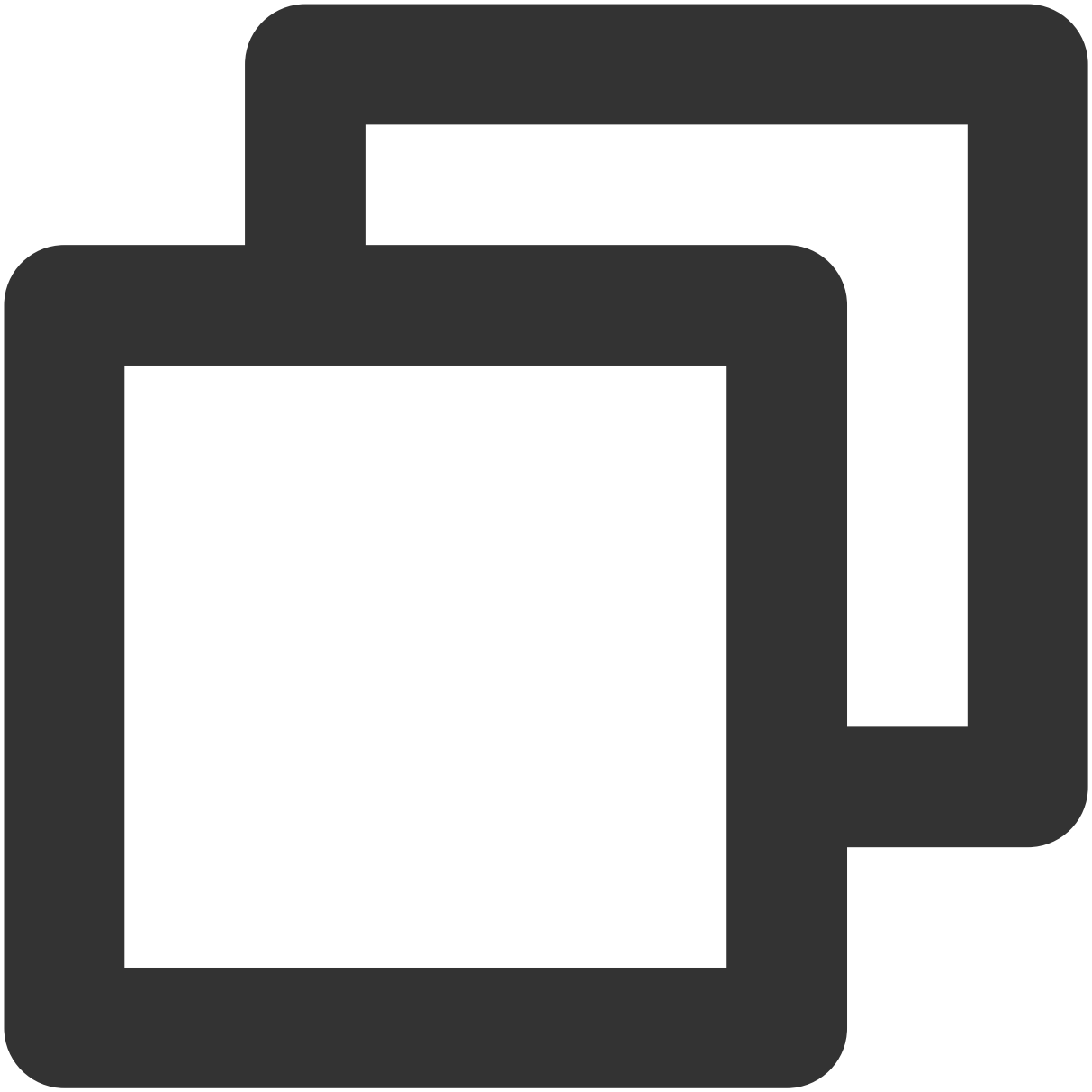
Getting the capturing status of a user

This API is used to obtain the microphone status for a user in a chat room.

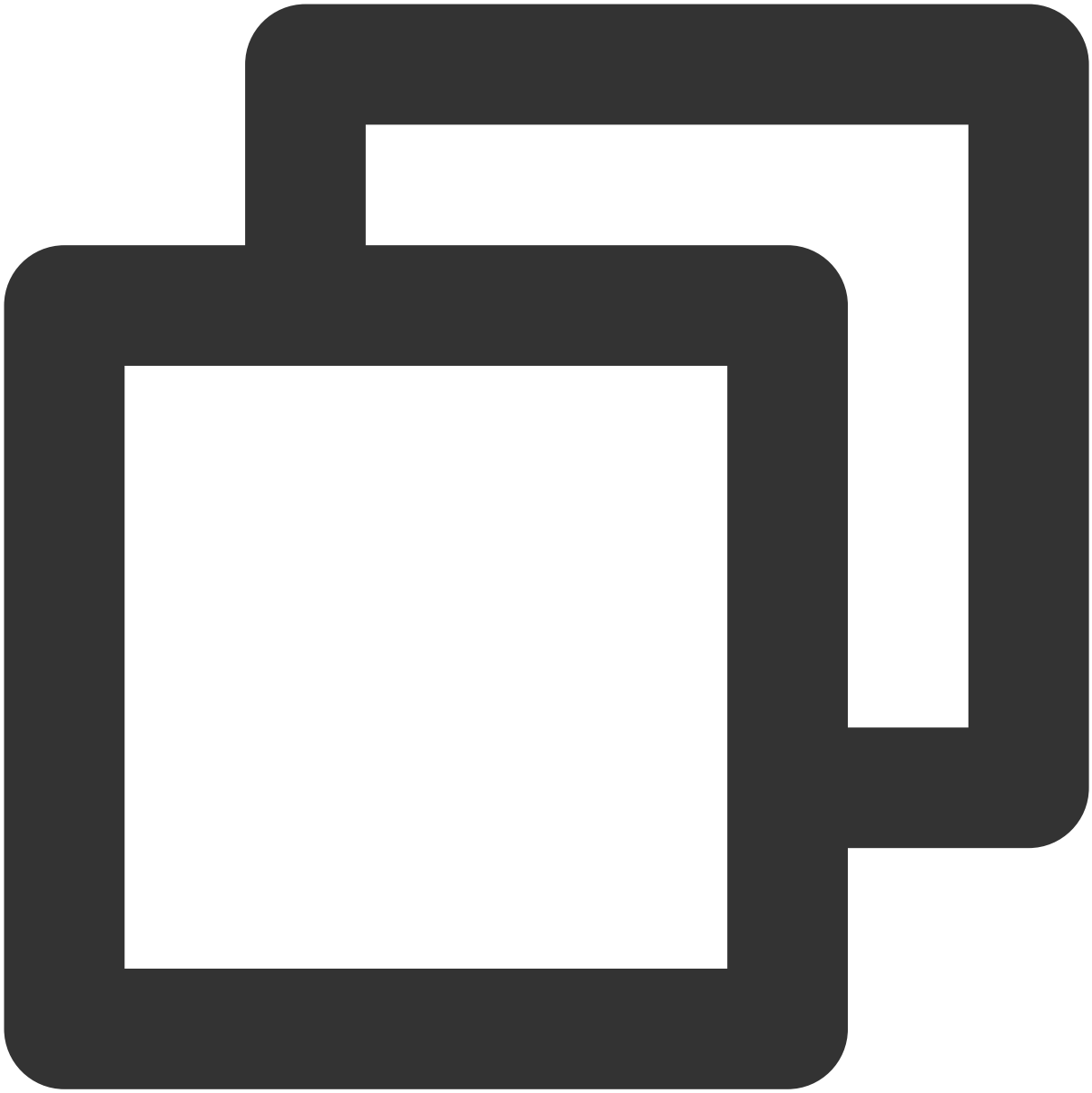
Function prototype

Android

iOS



```
public abstract int GetMicState(String receiverID);
```



```
-(QAVResult)GetMicState:(NSString *)receiverID;
```

Parameter	Type	Description
receiverID	NSString*	Specifies the `OpenId` of the user

Callback

The callback parameter is ITMG_ROOM_MANAGEMENT_GET_MIC_STATE.

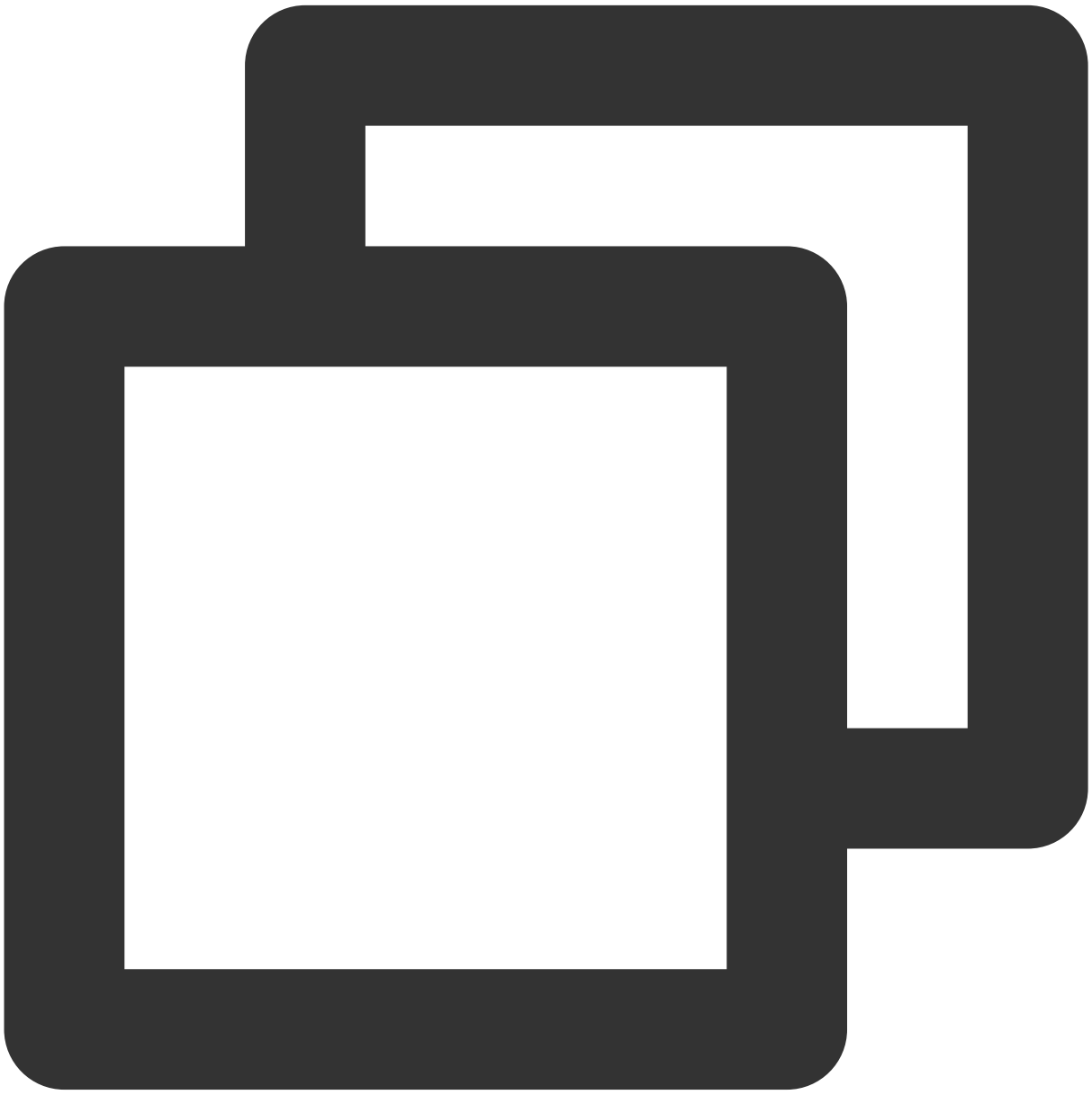
Getting the playback status of a user

This API is used to obtain the speaker status for a user in a chat room.

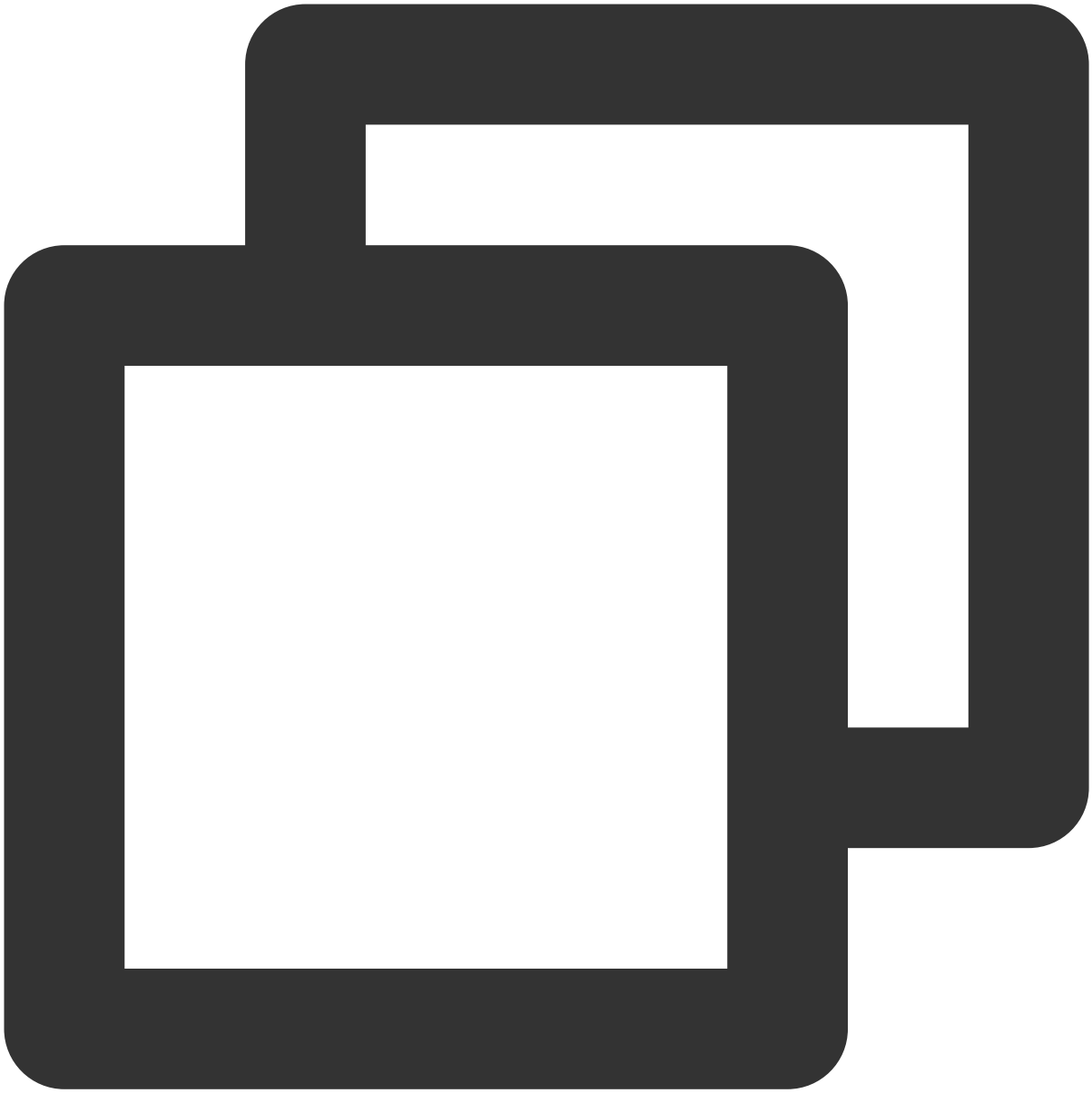
Function prototype

Android

iOS



```
public abstract int GetSpeakerState(String receiverID);
```



```
-(QAVResult)GetSpeakerState:(NSString *)receiverID;
```

Callback

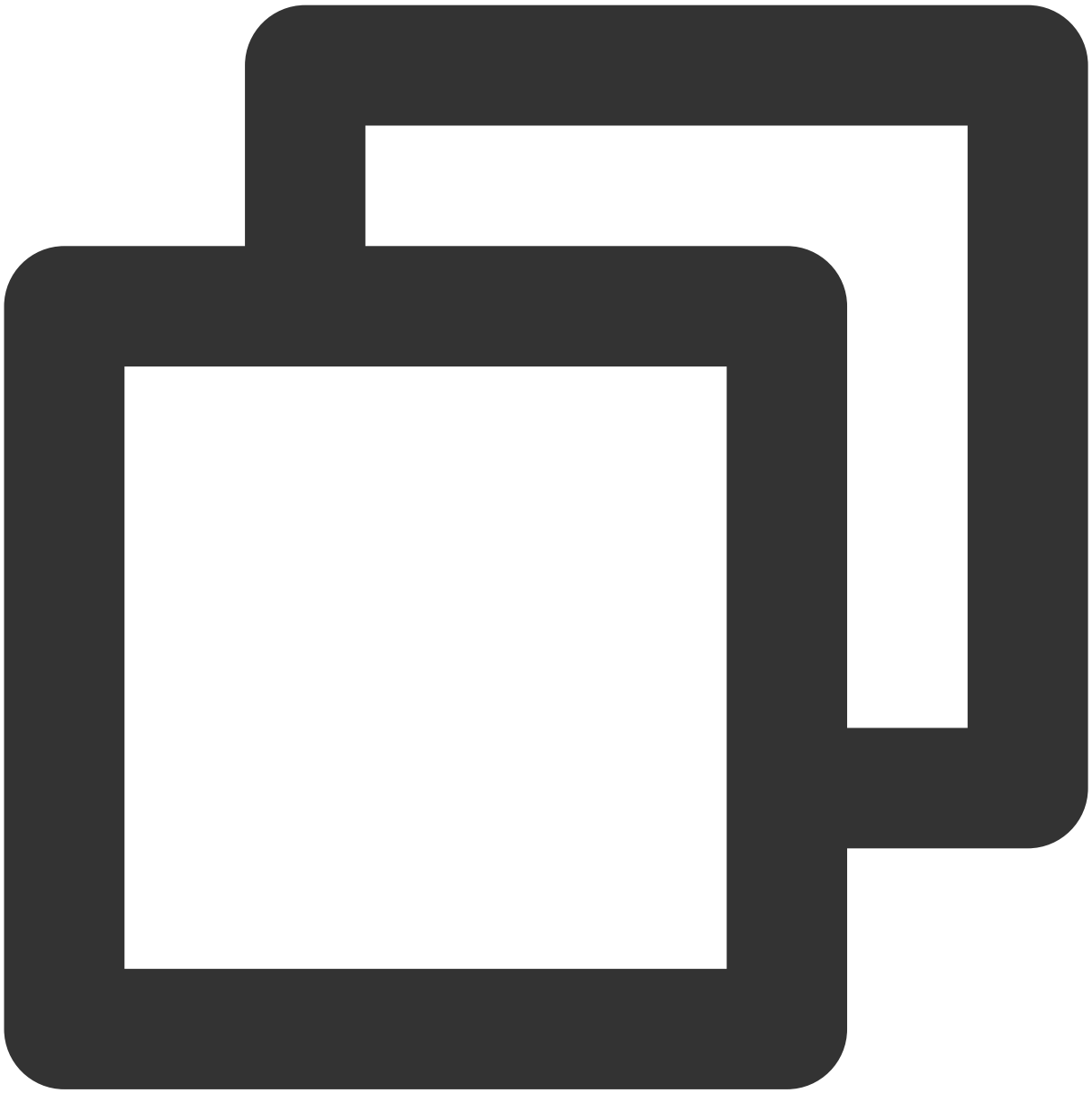
The callback parameter is ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE.

Disabling capturing and playback for a user

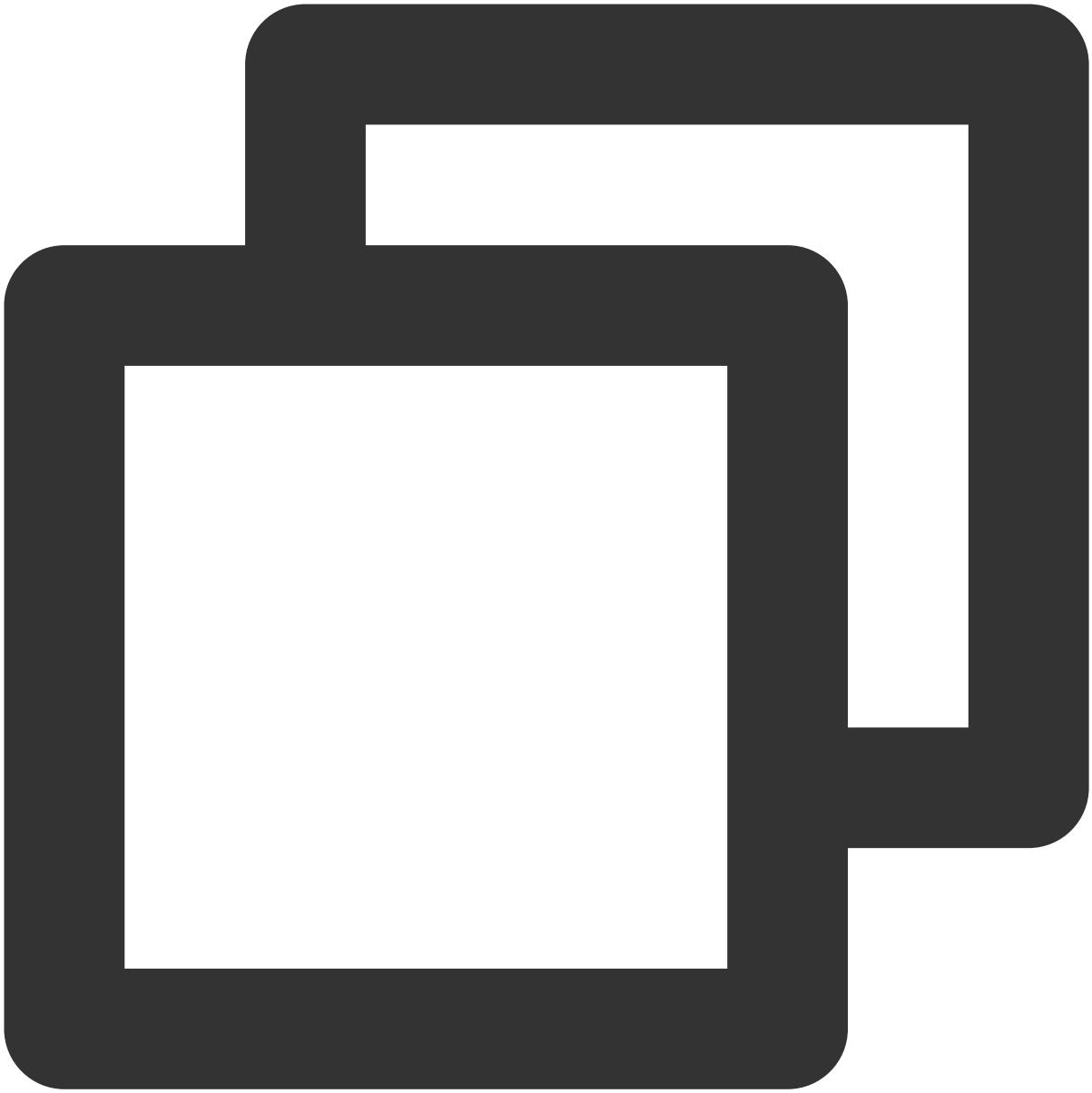
Each user that enters a chat room can use their microphone and speaker by default. Calling this API will have the microphone and speaker disabled for a user until he or she exits the room.

Android

iOS



```
public  abstract int ForbidUserOperation(boolean isEnabled,String receiverID);
```



```
-(QAVResult)ForbidUserOperation:(BOOL)enable Receiver:(NSString *)receiverID;
```

Parameter	Type	Description
enable	BOOL	`YES` : Disable the devices for a user; `NO` : Enable the devices for a user
receiverID	NSString*	Specifies the `OpenId` of the user

Callback

The callback parameter is ITMG_ROOM_MANAGERMENT_FOBIN_OP.

Processing Callbacks

Like all the other GME callbacks, the chat room management callbacks are processed using `OnEvent` with an event named `ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR`. This event returns a parameter structure as shown below:

Callback parameters

Parameter	Type	Description
SenderID	NSString	ID of the event sender. If it's the same as your own `OpenId`, the sender is your client.
ReceiverID	NSString	ID of the event receiver. If it's the same as your own `OpenId`, the receiver is your client.
OperateType	NSNumber	Event Type
Result	NSNumber	Event result. `0` indicates success.
OperateValue	NSNumber	Command details

OperateType

Value	Event Type	Description
0	ITMG_ROOM_MANAGEMENT_CAPTURE_OP	The capturing device was controlled.
1	ITMG_ROOM_MANAGEMENT_PLAY_OP	The playback device was controlled.
2	ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP	Audio upstreaming was controlled.
3	ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP	Audio downstreaming was controlled.
4	ITMG_ROOM_MANAGEMENT_MIC_OP	The mic was controlled.
5	ITMG_ROOM_MANAGEMENT_PLAY_OP	The speaker was controlled.
6	ITMG_ROOM_MANAGEMENT_GET_MIC_STATE	The mic status was obtained.
7	ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE	The speaker status was obtained.
8	ITMG_ROOM_MANAGERMENT_FOBIN_OP	The mic and speaker were disabled.

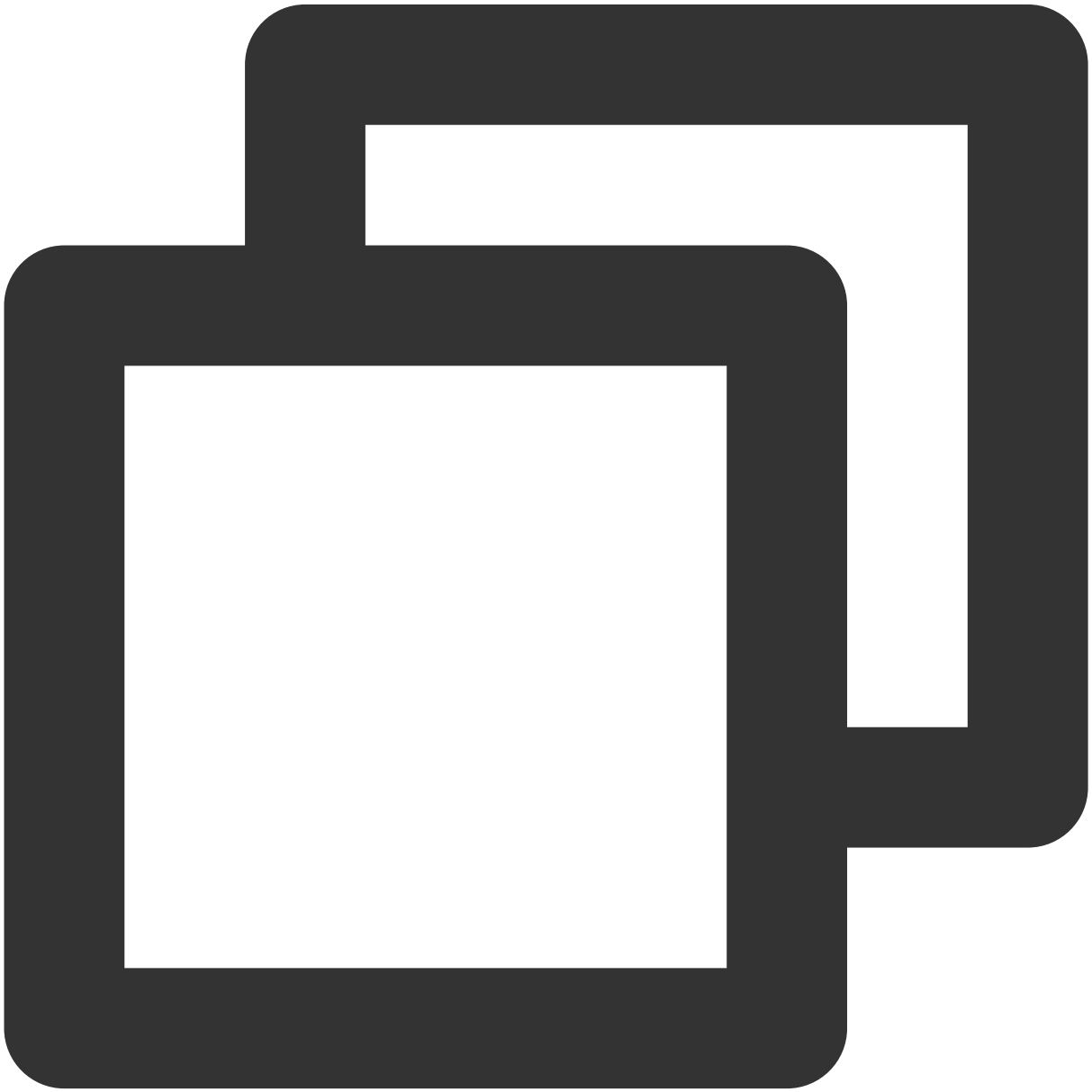
OperateValue

Member	Description
boolValue	`0` : Disable; `1` : Enable.

Sample code

Android

iOS



```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR

            ArrayList<String> operatorArr = new ArrayList<String>();
            operatorArr.add("Capture");
            operatorArr.add("Play back");
            operatorArr.add("Upstream");
            operatorArr.add("Downstream");
            operatorArr.add("Capture and upstream");
            operatorArr.add("Play back and downstream");
            operatorArr.add("Mic status");
            operatorArr.add("Speaker status");
            operatorArr.add("Disable the mic/speaker");

            String SenderID =  data.getStringExtra("SenderID");
            String ReceiverID = data.getStringExtra("ReceiverID");
            int OperateType = data.getIntExtra("OperateType",-1000);

            int Result =data.getIntExtra("Result",-1000);
            boolean OperateValue = data.getBooleanExtra("OperateValue",false);
            if (OperateType == -1000 ||Result == -1000) {
                return;
            }
            if (SenderID.equals(identifier)) {
                if (OperateType == ITMGContext.ITMG_ROOM_MANAGEMENT_GET_MIC_STATE |
                    Toast.makeText(getActivity(), String.format("Sent ID %s the res
                } else  {
                    Toast.makeText(getActivity(), String.format("Sent ID %s the res
                }

            } else if (ReceiverID.equals(identifier)||ReceiverID.equals("ALL")) {
                if (Result == 0) {
                    switch (OperateType) {
                        case ITMGContext.ITMG_ROOM_MANAGEMENT_CAPTURE_OP:
                        {
                            if (!OperateValue) {
                                mSwitchCapture.setChecked(OperateValue);
                            } else  {
                                AlertDialog.Builder dialog = new AlertDialog.Builde
                                dialog.setTitle("Whether to enable device capturing
                                dialog.setMessage("");
                                dialog.setCancelable(false);
                                dialog.setPositiveButton("On", new DialogInterface.
                                    // Set the click event of the **OK** button
                                    @Override
                                    public void onClick(DialogInterface dialog, int
```

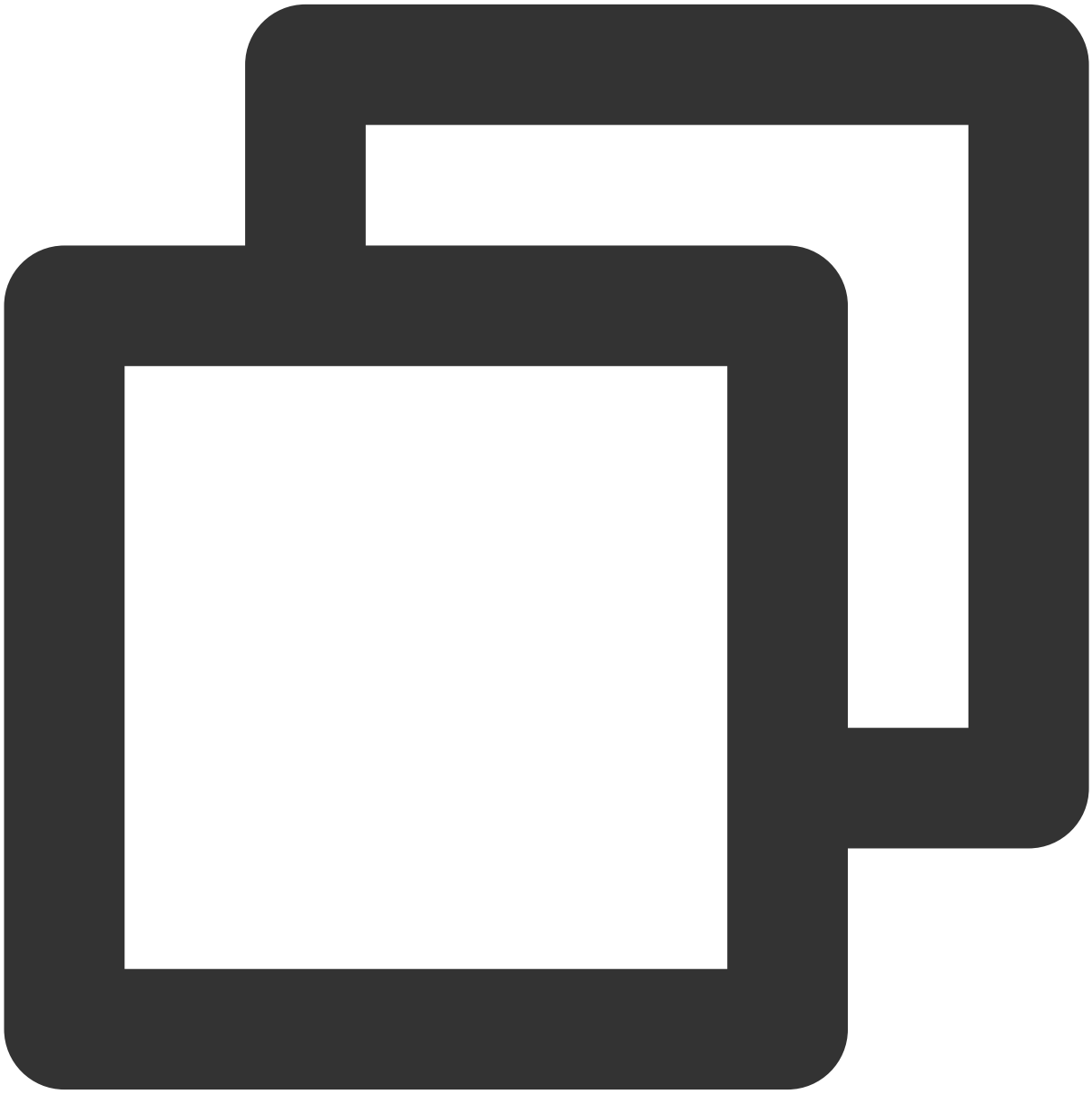
```
                        mSwitchCapture.setChecked(true);
                        ITMGContext.GetInstance(getActivity()).GetA
                    }
                });
                dialog.setNegativeButton("Off", new DialogInterface
                    // Set the click event of the **Cancel** button
                    @Override
                    public void onClick(DialogInterface dialog, int
                    }
                });
                dialog.show();
            }

        }
            break;
        case ITMGContext.ITMG_ROOM_MANAGEMENT_PLAY_OP:
        {
            mSwitchPlayDevice.setChecked(OperateValue);
        }
            break;
        case ITMGContext.ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP:
        {
            if (!OperateValue) {
                mSwitchSend.setChecked(OperateValue);
            } else  {
                AlertDialog.Builder dialog = new AlertDialog.Builde
                dialog.setTitle("Whether to enable upstreaming");
                dialog.setMessage("");
                dialog.setCancelable(false);
                dialog.setPositiveButton("On", new DialogInterface.
                    // Set the click event of the **OK** button
                    @Override
                    public void onClick(DialogInterface dialog, int
                        mSwitchSend.setChecked(true);
                        ITMGContext.GetInstance(getActivity()).GetA
                    }
                });
                dialog.setNegativeButton("Off", new DialogInterface
                    // Set the click event of the **Cancel** button
                    @Override
                    public void onClick(DialogInterface dialog, int
                    }
                });
                dialog.show();
            }
        }
            break;
```

```
                    case ITMGContext.ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP:
                     {
                         mSwitchRecv.setChecked(OperateValue);
                    }
                        break;
                    case ITMGContext.ITMG_ROOM_MANAGEMENT_MIC_OP:
                     {
                         if (!OperateValue) {
                             mSwitchCapture.setChecked(OperateValue);
                             mSwitchSend.setChecked(OperateValue);
                         }  else  {
                             AlertDialog.Builder dialog = new AlertDialog.Build
                             dialog.setTitle("Whether to enable capturing and u
                             dialog.setMessage("");
                             dialog.setCancelable(false);
                             dialog.setPositiveButton("On", new DialogInterface
                                 // Set the click event of the **OK** button
                                 @Override
                                 public void onClick(DialogInterface dialog, in
                                     mSwitchCapture.setChecked(true);
                                     mSwitchSend.setChecked(true);
                                     ITMGContext.GetInstance(getActivity()).Get
                                 }
                             });
                             dialog.setNegativeButton("Off", new DialogInterfac
                                 // Set the click event of the **Cancel** butto
                                 @Override
                                 public void onClick(DialogInterface dialog, in
                                 }
                             });
                             dialog.show();
                         }
                       }
                        break;
                    case ITMGContext.ITMG_ROOM_MANAGEMENT_SPEAKER_OP:
                    {
                        mSwitchPlayDevice.setChecked(OperateValue);
                        mSwitchRecv.setChecked(OperateValue);
                    }
                        break;

                }
            }
            if (OperateType == ITMGContext.ITMG_ROOM_MANAGEMENT_GET_MIC_STATE |
            {
                Toast.makeText(getActivity(), String.format("Received from the
            }
```

```
                else if (OperateType == ITMGContext.ITMG_ROOM_MANAGEMENT_SPEAKER_OP
                    Toast.makeText(getActivity(), String.format("Received from ID %
                } else if (OperateValue == false) {
                    Toast.makeText(getActivity(), String.format("Received from ID %
                }
            }
    }
```



```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSString* log =[NSString stringWithFormat:@"OnEvent:%d,data:%@", (int)eventType
    [self showLog:log];
```

```
    NSLog(@"====%@====",log);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR:
        {
            NSArray *operatorArr = @[@"capture",@"play back",@"upstream",@"downstre
            // _openId
            NSString *SenderID = [data objectForKey:@"SenderID"];
            NSString *ReceiverID = [data objectForKey:@"ReceiverID"];
            NSNumber *OperateType = [data objectForKey:@"OperateType"];
            NSNumber *Result = [data objectForKey:@"Result"];
            NSNumber *OperateValue = [data objectForKey:@"OperateValue"];

            // Request sent
            if ([SenderID isEqualToString:_openId]) {
                if (OperateType.intValue == ITMG_ROOM_MANAGEMENT_GET_MIC_STATE || O
                          NSString *alterString = [NSString stringWithFormat:@"%@ r
                                                   UIAlertView *alert = [[UIAlertView all
                                                   [alert show];
                }
                else
                {
                    NSString *alterString = [NSString stringWithFormat:@"%@%@ reque
                                UIAlertView *alert = [[UIAlertView alloc] initWithTi
                                [alert show];
                }


            }
            else if([ReceiverID isEqualToString:_openId] ){ // Request received
                if (Result.intValue == 0) {
                    switch (OperateType.intValue) {
                        case ITMG_ROOM_MANAGEMENT_CAPTURE_OP:{
                            [_micSwitch setOn:OperateValue.boolValue animated:true]
                        }
                            break;
                        case ITMG_ROOM_MANAGEMENT_PLAY_OP:{
                            [_speakerSwitch setOn:OperateValue.boolValue animated:t
                            }
                            break;
                        case ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP:{
                            [_sendSwitch setOn:OperateValue.boolValue animated:true
                        }
                            break;
                        case ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP:{
                            [_recvSwitch setOn:OperateValue.boolValue animated:true
                        }
                            break;
```

```
                        case ITMG_ROOM_MANAGEMENT_MIC_OP:{
                        [_micSwitch setOn:OperateValue.boolValue animated:true];
                        [_sendSwitch setOn:OperateValue.boolValue animated:true];
                        }
                            break;
                        case ITMG_ROOM_MANAGEMENT_SPEAKER_OP:{
                        [_speakerSwitch setOn:OperateValue.boolValue animated:true]
                        [_recvSwitch setOn:OperateValue.boolValue animated:true];

                        }
                            break;
                        default:
                            break;
                    }

                if (OperateType.intValue == ITMG_ROOM_MANAGEMENT_GET_MIC_STATE || O
                        NSString *alterString = [NSString stringWithFormat:@"%@ req
                                UIAlertView *alert = [[UIAlertView alloc] initWithT
                                [alert show];
                    }
                else{
                        NSString *alterString = [NSString stringWithFormat:@"%@%@ r
                              UIAlertView *alert = [[UIAlertView alloc] initWithTit
                              [alert show];
                    }
                }
            }
        }
        break;
}
```