游戏多媒体引擎

进阶功能开发指南

产品文档

腾讯云

文档目录

进阶功能开发指南
服务端录制
全量录制

最近更新时间：2024-01-18 14:12:12

本文帮助您通过全量录制的方式快速接入 **GME 服务端录制**功能。

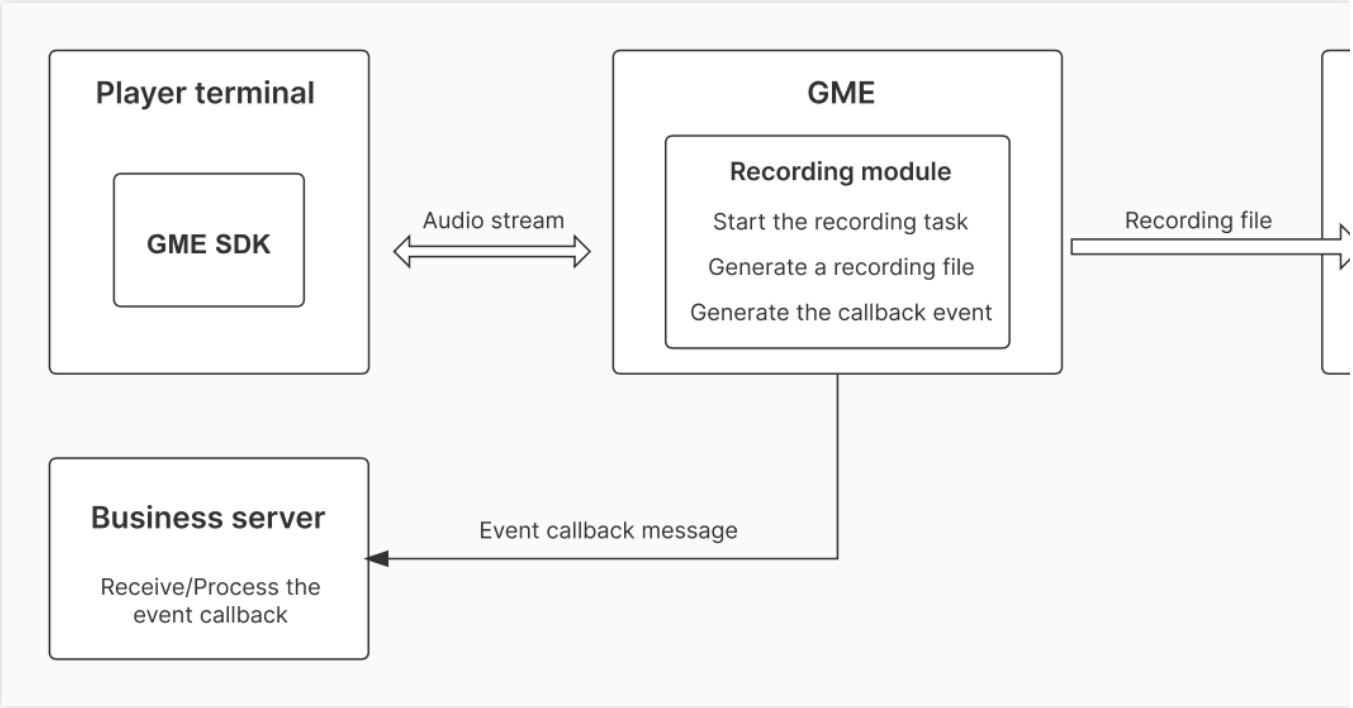
使用场景

GME 对实时语音流提供**服务端录制**能力，帮助开发者实现内容留存、内容管理、内容再生产等场景。全量录制：支持录制应用内全量语音房间按房间维度混流、按用户维度单流；自定义录制：支持录制用户指定房间按房间维度混流、按用户维度单流。录制后的音频文件将存储在您账号下的**对象存储**(COS)服务中。
本文仅针对**全量录制**的开发接入方法进行说明。若您需要对应用开启自定义录制，请参见 开发指南-自定义录制
注意：
使用 GME 服务端录制功能，录制过程将在 GME 产生录制服务费用。GME 国际站录制服务将从2023年4月1日起正式计费，详细计费信息将提前公示在 GME 购买指南。
录制后的文件将存储在您的腾讯云账号下的**对象存储（COS）**服务中，将会根据您的存储量、存储时长、访问频次等具体使用方式产生**对象存储（COS）**账单。详细计费信息请参考COS计费说明。

前提条件

已开通实时语音服务：可参见 服务开通指引。
已开通服务端录制服务：目前服务端录制功能针对白名单用户提供，请联系我们开通白名单。
已接入GME SDK：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入、Unity SDK 快速接入、Unreal SDK 快速接入。

服务架构



功能说明

1. 录制范围

开启全量录制，将对所有实时语音房间进行录制。您可以配置仅录制房间混流，或仅录制用户单流，或同时录制单流和混流。

2. 录制机制

录制任务启动机制

第一个用户进房后触发录制任务启动。

录制任务终止机制

最后一个用户退房后触发录制任务终止。

录制任务分片机制

当单个音频文件时长达到两个小时，将自动进行音频文件分片。
当用户关麦，用户单流录制音频将自动分片，用户开麦后启动新的分片。
若录制中的任务异常中断，任务自动重连后将启动新的分片。

录制任务事件通知机制

录制任务事件通过回调机制通知到您配置的回调地址。当发生这些事件时，您会收到回调通知：**录制开始、录制停止、录制文件上传完成。**
关于回调信息的详情，请参见 录制回调说明。

3. 存储位置

GME 服务端录制，录制完成后的音频文件将存储在您账号下的**对象存储（COS）**服务中的指定存储桶。存储桶地域由您指定，可选的地域列表请参见 对象存储地域说明。

4. 录制文件格式

.mp3

5. 录制文件命名规则

用户单流录制文件：`bizid_roomid_userid/${任务开始时间}_${id}_audio.mp3` 。
房间混流录制文件：`bizid_roomid/${taskid} _${任务开始时间} _${id}_audio.mp3` 。
bizid：GME 应用 ID。可在 GME 控制台 获取。
roomid：语音房间 ID。是您在使用实时语音服务时定义并传入至 GME SDK。
userid：玩家 ID。是您在使用实时语音服务时定义并传入至 GME SDK。
taskid：录制任务 ID，由 GME 录制服务生成。每一个录制任务都具有一个独特的任务 ID。
id：针对一个录制任务的分片的序列号。序列号初始序号为0。

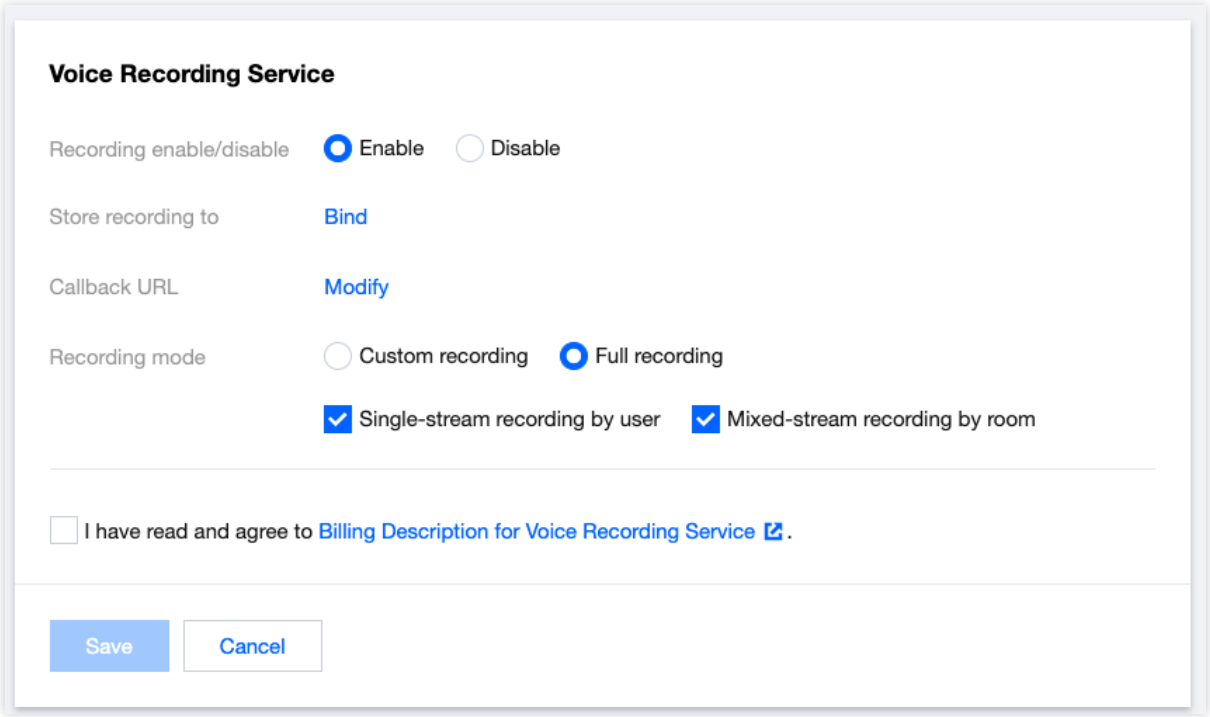
接入步骤

1. 在控制台完成录制服务配置（业务侧）
2. 接收录制任务回调（可选）（业务侧）
3. 查看/管理录制文件（业务侧）

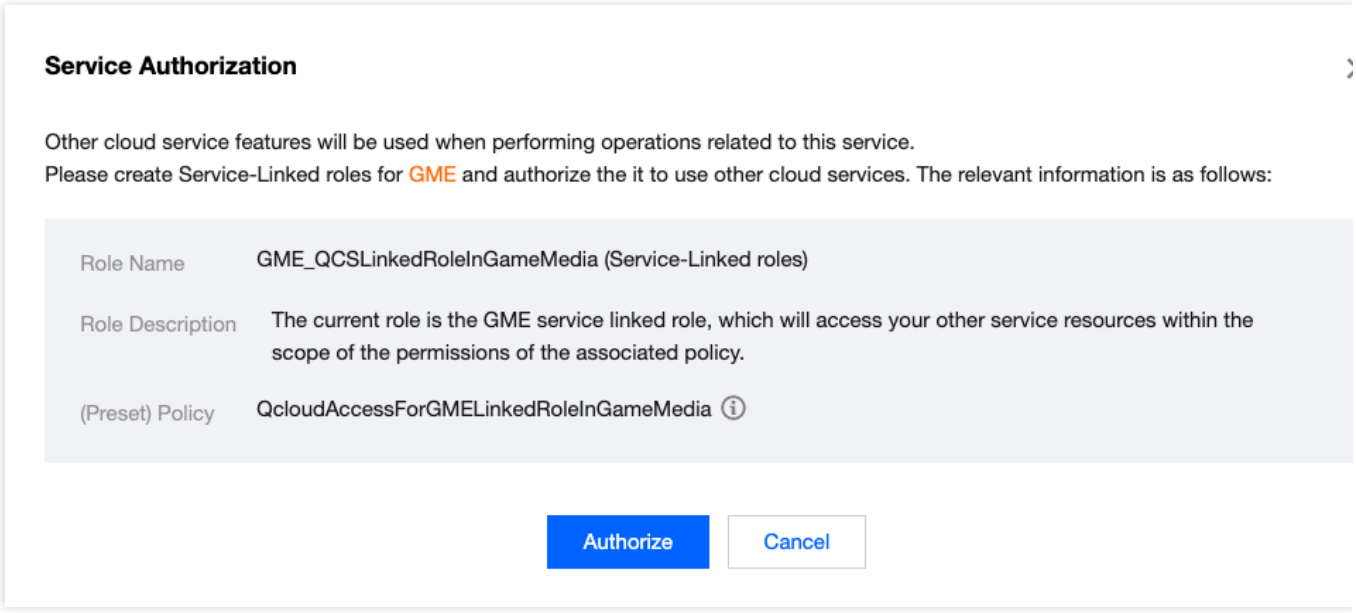
步骤一：在控制台完成录制服务配置

开通/关闭录制服务

登录 GME 控制台，进入**服务管理**菜单，对希望启用录制服务的应用单击**设置**，进入应用详情页。在页面中对**语音录制服务**单击**修改**。

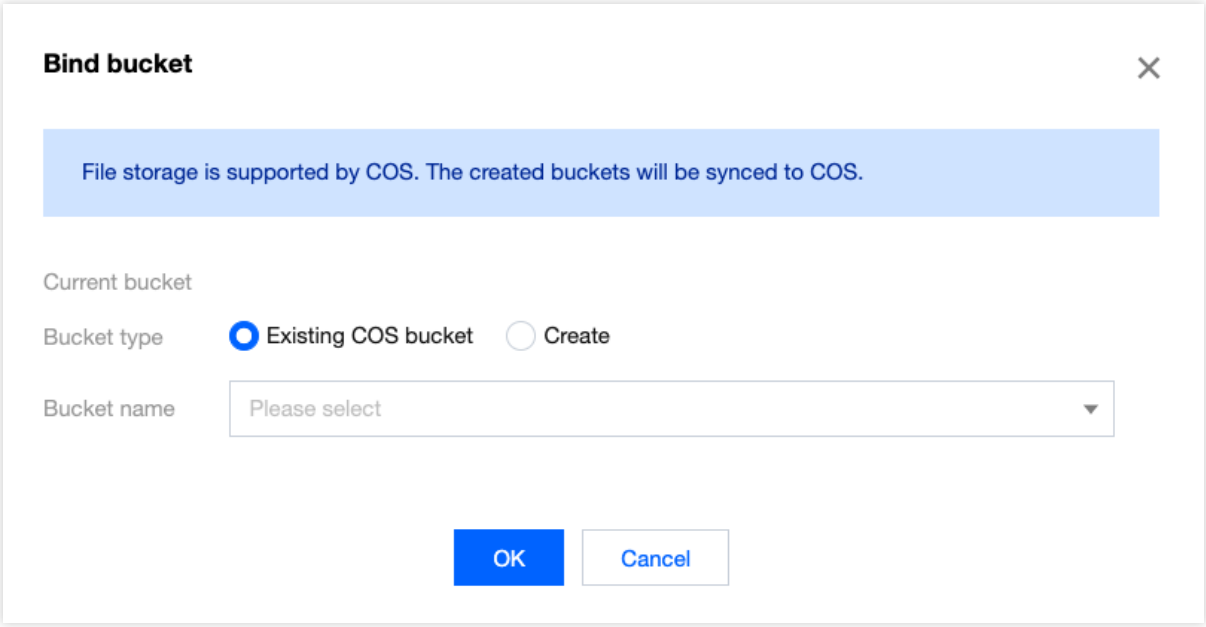


将录制开关置为**开启**。首次开启录制服务时，GME 将会向您申请服务授权，以便于访问您的**对象存储 COS 服务**，您需要在弹窗页面需要同意授权后，才能正常开启服务端录制服务。



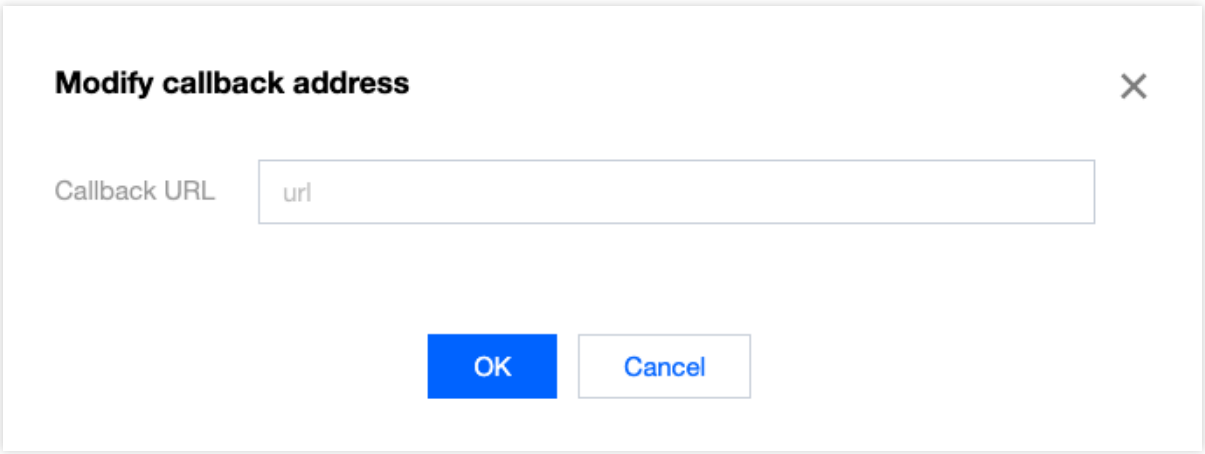
录音文件存储配置

进入应用详情页面，在页面中对**语音录制服务**单击**修改**，在**录制文件存储桶**单击**绑定**。
在**绑定存储桶**弹窗中，您可以绑定一个已有的 COS 存储桶（已有的存储桶需要您在 COS 控制台 先行创建），或新建一个存储桶。



录制事件回调配置（可选）

如果您希望接收录制服务的事件回调，可以配置回调地址。操作路径：进入应用详情页面，在页面中对**语音录制服务**单击**修改**，在**回调地址**单击**修改**，在弹窗中输入接收回调的 url 地址。目前仅针对录制任务完成状态推送事件回调消息。



录制范围配置

录制范围可选自定义录制或全量录制。您此时应选中全量，并在复选框中指定是否录制单流、是否录制混流。

完成上述配置后，单击**保存**，录制服务即可开启。若您不再需要使用录制服务，请及时在控制台将录制服务开关置为**关闭**， 避免产生预期之外的费用。

步骤二：接收录制任务回调（可选）

如果您在**步骤一**中配置了回调地址，则需要接收录制任务事件回调。

步骤三：查看/管理录制文件

一个录制任务结束后的数分钟内即可生成完成的音频文件。您需要登录您的 对象存储 COS 控制台 对录制文件进行查看和管理。

自定义录制

最近更新时间：2024-01-18 14:13:34

本文帮助您通过自定义录制的方式接入 **GME 服务端录制**功能。

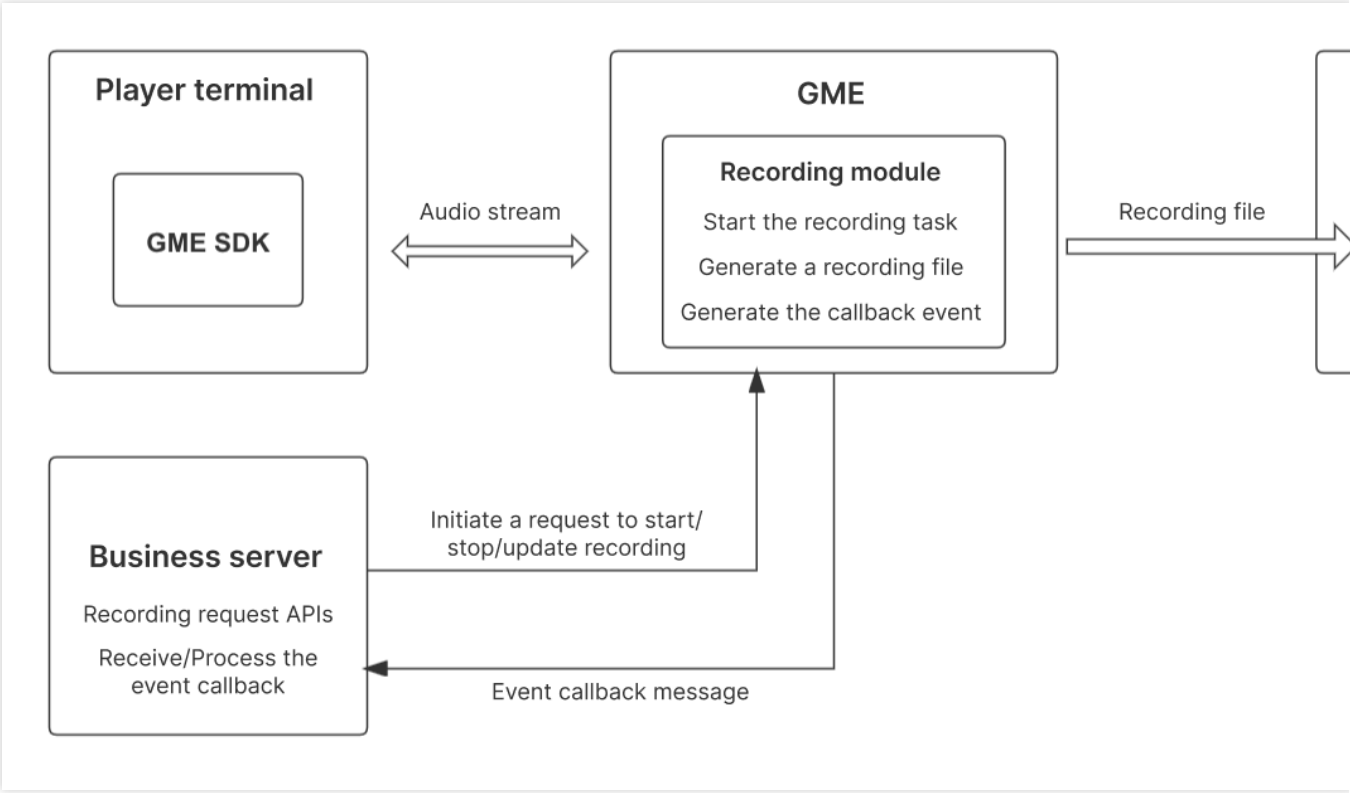
使用场景

GME 对实时语音流提供**服务端录制**能力，帮助开发者实现内容留存、内容管理、内容再生产等场景。全量录制：支持录制应用内全量语音房间按房间维度混流、按用户维度单流。自定义录制：支持录制用户指定房间按房间维度混流、按用户维度单流。录制后的音频文件将存储在您账号下的**对象存储**（COS）服务中。
本文仅针对**自定义录制**的开发接入方法进行说明。若您需要对应用开启全量录制，请参见 开发指南-全量录制。
注意：
使用 GME 服务端录制功能，录制过程将在 GME 产生录制服务费用。GME 国际站录制服务将从2023年4月1日起正式计费，计费详情请参见 GME 购买指南。
录制后的文件将存储在您的腾讯云账号下的**对象存储（COS）**服务中，将会根据您的存储量、存储时长、访问频次等具体使用方式产生**对象存储（COS）**账单。详细计费信息请参见 COS 计费说明。

前提条件

已开通实时语音服务：可参见 服务开通指引。
已开通服务端录制服务：目前服务端录制功能针对白名单用户提供，请联系我们开通白名单。
已接入GME SDK：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入、Unity SDK 快速接入、Unreal SDK 快速接入。

服务架构



功能说明

1. 录制范围

您可通过服务端接口指定需要录制的房间 ID 混流或房间内用户单流。
针对指定录制的房间 ID，您可以通过服务端接口参数指定需要录制的玩家白名单，或无需录制的玩家黑名单。

2. 相关接口

StartRecord() 开始录制
您可在这个接口参数中定义录制单流或混流、指定需要录制的 RoomID、指定订阅的白名单用户 ID 或黑名单用户 ID。
StopRecord() 结束录制
您可以通过这个接口，对某个 taskid 停止录制。若您没有记录 taskid，则需要先通过 DescribeTaskInfo() 获取指定房间正在进行的录制任务 taskid。
ModifyRecordInfo()更新录制信息
您可以通过这个接口，对某个 taskid 更新录制信息。包括更新录制类型、更新订阅的白名单用户 ID 或黑名单用户 ID。若您没有记录 taskid，则需要先通过 DescribeTaskInfo() 获取指定房间正在进行的录制任务 taskid。
DescribeTaskInfo()查询房间录制信息
您可以通过这个接口，查询指定房间的录制信息。包括进行中的录制任务 taskid、订阅的白名单/黑名单用户 ID。
DescribeRecordInfo()查询录制任务信息

您可以通过这个接口，查询某个 taskid 的任务信息。包括录制类型、录制的房间 ID、录制的用户 ID、录制文件信息

3. 录制机制

录制任务启动机制

StartRecord() 接口后，指定的房间录制任务将会启动。

录制任务终止机制

StopRecord() 接口后，指定的房间录制任务将会结束。

录音文件生成时机

房间混流录音文件：房间录制任务启动后，若房间已有用户，则混流音频文件立刻开始生成；若房间内无用户，则第一个用户进房后，混流音频文件立刻开始生成。
用户单流录音文件：房间录制任务启动后，在录制范围内的用户进房，则该用户的单流音频文件立刻开始生成。

录制任务分片机制

当单个音频文件时长达到两个小时，将自动进行音频文件分片。
当用户关麦，用户单流录制音频将自动分片，用户开麦后启动新的分片。
若录制中的任务异常中断，任务自动重连后将启动新的分片。

录制任务事件通知机制

录制任务事件通过回调机制通知到您配置的回调地址。当发生这些事件时，您会收到回调通知：**录制开始、录制停止、录制文件上传完成。**
关于回调信息的详情，请参考 录制回调说明。

4. 存储位置

GME 服务端录制，录制完成后的音频文件将存储在您账号下的**对象存储 COS** 服务中的指定存储桶。存储桶地域由您指定，可选的地域列表请参考 对象存储地域说明。

5. 录制文件格式

.mp3

6. 录制文件命名规则

用户单流录制文件：`bizid_roomid_userid/${任务开始时间}_${id}_audio.mp3` 。
房间混流录制文件：`bizid_roomid/${taskid}` *`${任务开始时间}`* `${id}_audio.mp3` 。
bizid: GME 应用 ID。可在 GME 控制台 获取。
roomid: 语音房间 ID。是您在使用实时语音服务时定义并传入至 GME SDK。
userid: 玩家 ID。是您在使用实时语音服务时定义并传入至 GME SDK。

taskid: 录制任务 ID，由 GME 录制服务生成。每一次成功调用 录制任务都具有一个独特的任务 ID。
id: 针对一个录制任务的分片的序列号。序列号初始序号为0。

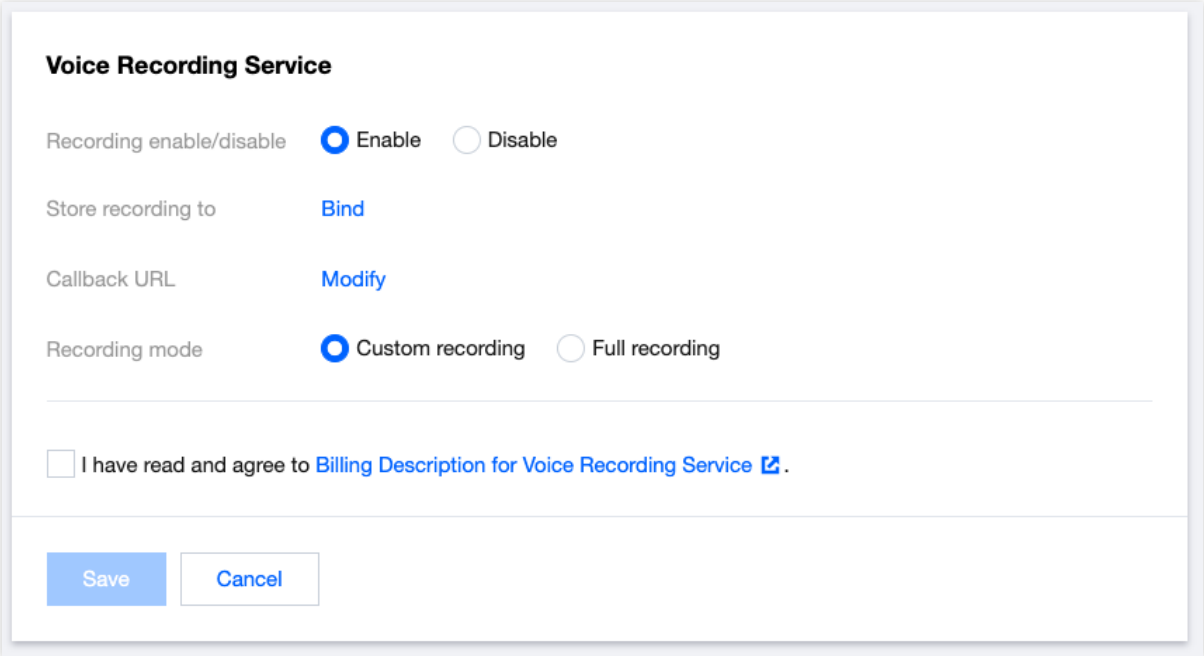
接入步骤

1. 在控制台完成录制服务配置（业务侧）
2. 调用服务端接口，定义录制范围（业务侧）
3. 接收录制任务回调（可选）（业务侧）
4. 查看/管理录制文件（业务侧）

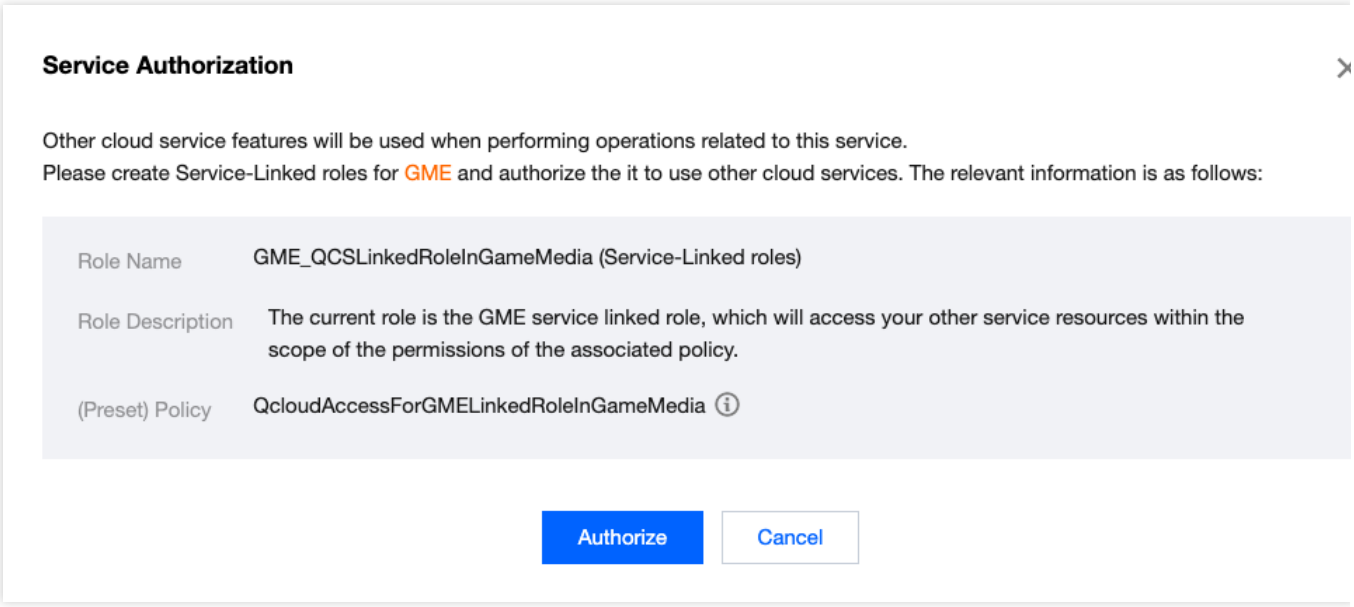
步骤一：在控制台完成录制服务配置

登录 GME 控制台，进入**服务管理**菜单，对希望启用录制服务的应用单击**设置**，进入应用详情页。

开通/关闭录制服务

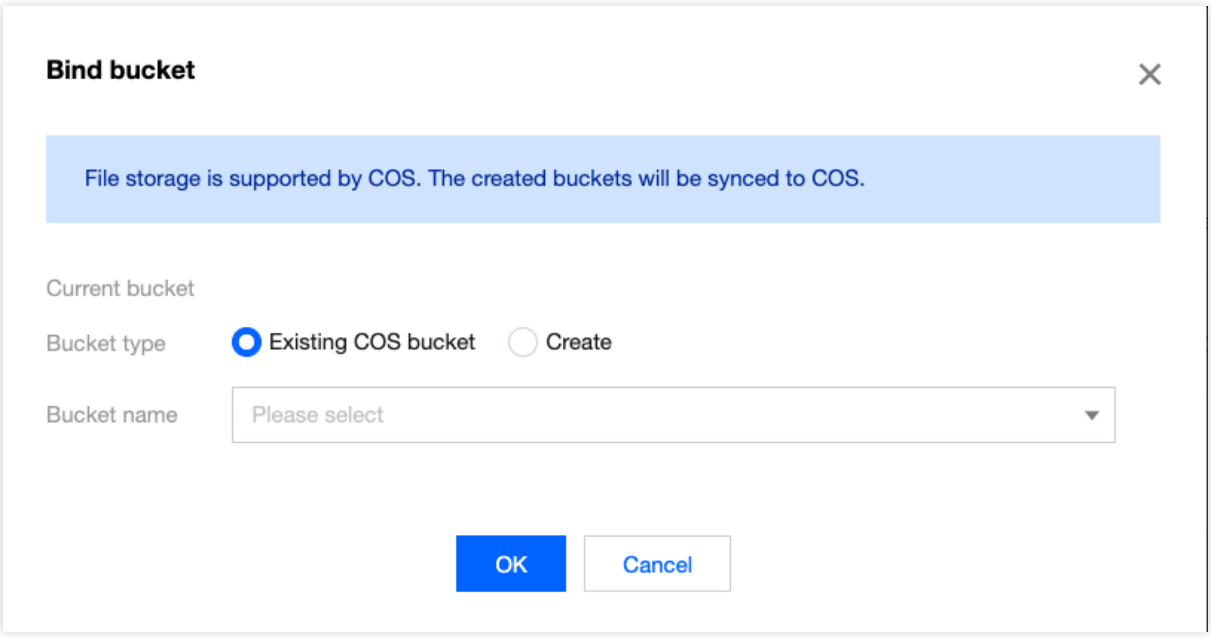


在页面中对**语音录制服务**单击**修改**，将录制开关置为**开启** 。首次开启录制服务时，GME 将会向您申请服务授权，以便于访问您的**对象存储 COS 服务**，您需要在弹窗页面需要同意授权后，才能正常开启服务端录制服务。



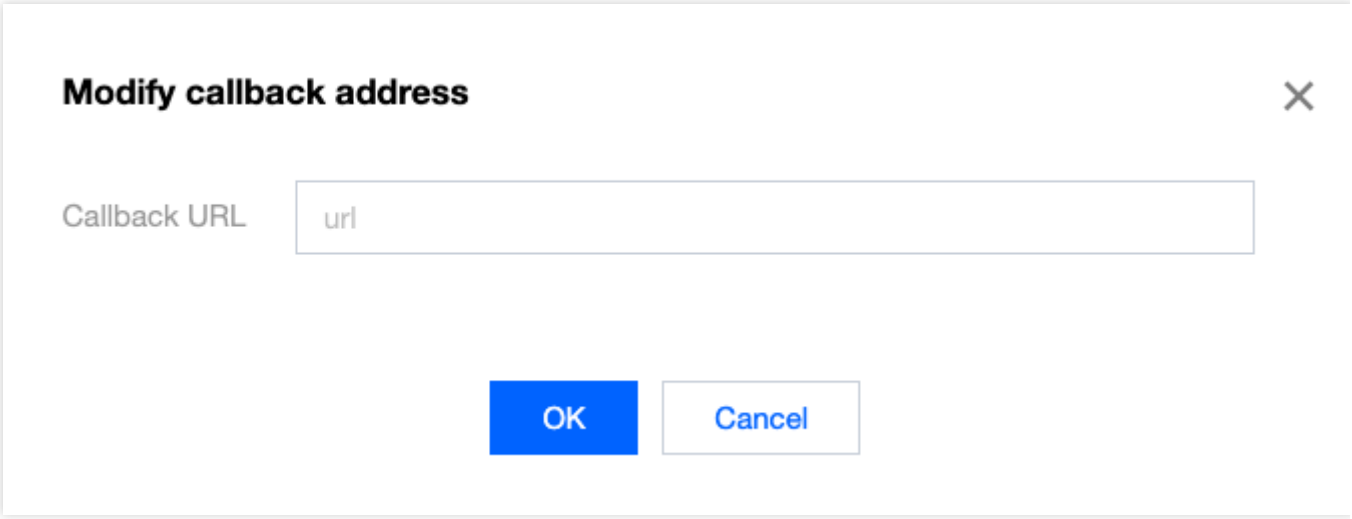
录音文件存储配置

在**录制文件存储桶**单击**绑定**。在**绑定存储桶**弹窗中，您可以绑定一个已有的 COS 存储桶（已有的存储桶需要您在 COS 控制台 先行创建），或新建一个存储桶。



录制事件回调配置（可选）

如果您希望接收录制服务的事件回调，可以配置回调地址。操作路径：进入应用详情页面，在页面中对**语音录制服务**单击**修改**，在**回调地址**单击**修改**，在弹窗中输入接收回调的 url 地址。目前仅针对录制任务完成状态推送事件回调消息。



录制范围配置

录制范围可选自定义录制或全量录制。您此时应选中自定义录制，否则调用服务端相关接口将会失败。
完成上述配置后，单击**保存**，录制服务即可开启。若您不再需要使用录制服务，请及时在控制台将录制服务开关置为**关闭**， 避免产生预期之外的费用。

步骤二：接收录制任务回调（可选）

如果您在**步骤一**中配置了回调地址，则可以接收录制任务事件回调。

步骤三：调用服务端接口，定义录制范围

您需要根据业务场景实际需求来调用相关接口。在调用时序和流程的设计上，请注意：
（1）仅支持对已存在的RoomId发起开始录制请求。若RoomId不存在，录制任务将创建失败
（2）若您没有主动调用停止录制，在房间内有用户的情况下，录制进程将一直保持。若房间内用户已全部退房，原录制任务进程将保持12小时。在这段保持的时间内，若用户重新进房，将自动开始录制。若已超过录制进程的保持时间，用户进房将不会自动开始录制

步骤四：查看/管理录制文件

一个录制任务结束后的数分钟内即可生成完成的音频文件。您需要登录您的 对象存储 COS 控制台 对录制文件进行查看和管理。

录制回调说明

最近更新时间：2024-01-18 14:14:28

服务端录制回调说明

说明

受网络影响，您的服务器收到的通知顺序和事件发生的顺序可能不完全一致。我们的服务有重试机制，但仍不能保证所有的消息都能到达。考虑到以上两点，不建议您业务的核心业务逻辑依赖消息通知服务。

网络协议

如果在控制台配置了回调地址，即一个 HTTP（S） 协议接口的 URL，则需要支持 POST 方法，传输数据编码采用 UTF-8。

HTTP 头参数

名称	类型	是否必需	描述
Signature	string	是	签名，具体见下方 签名生成 说明

签名生成

Signature = HMAC-SH1 (strContent, SecretKey)
strContent：签名原文串，为 body 的整个 JSON 内容（长度以 Content-Length 为准）。
body：回调给业务的 JSON 内容，下方 回调示例 中的全部内容即为 body。
SecretKey：密钥，为应用的权限密钥，可通过 控制台 > 应用详情 查看。
HMAC-SH1：签名算法。

回调参数

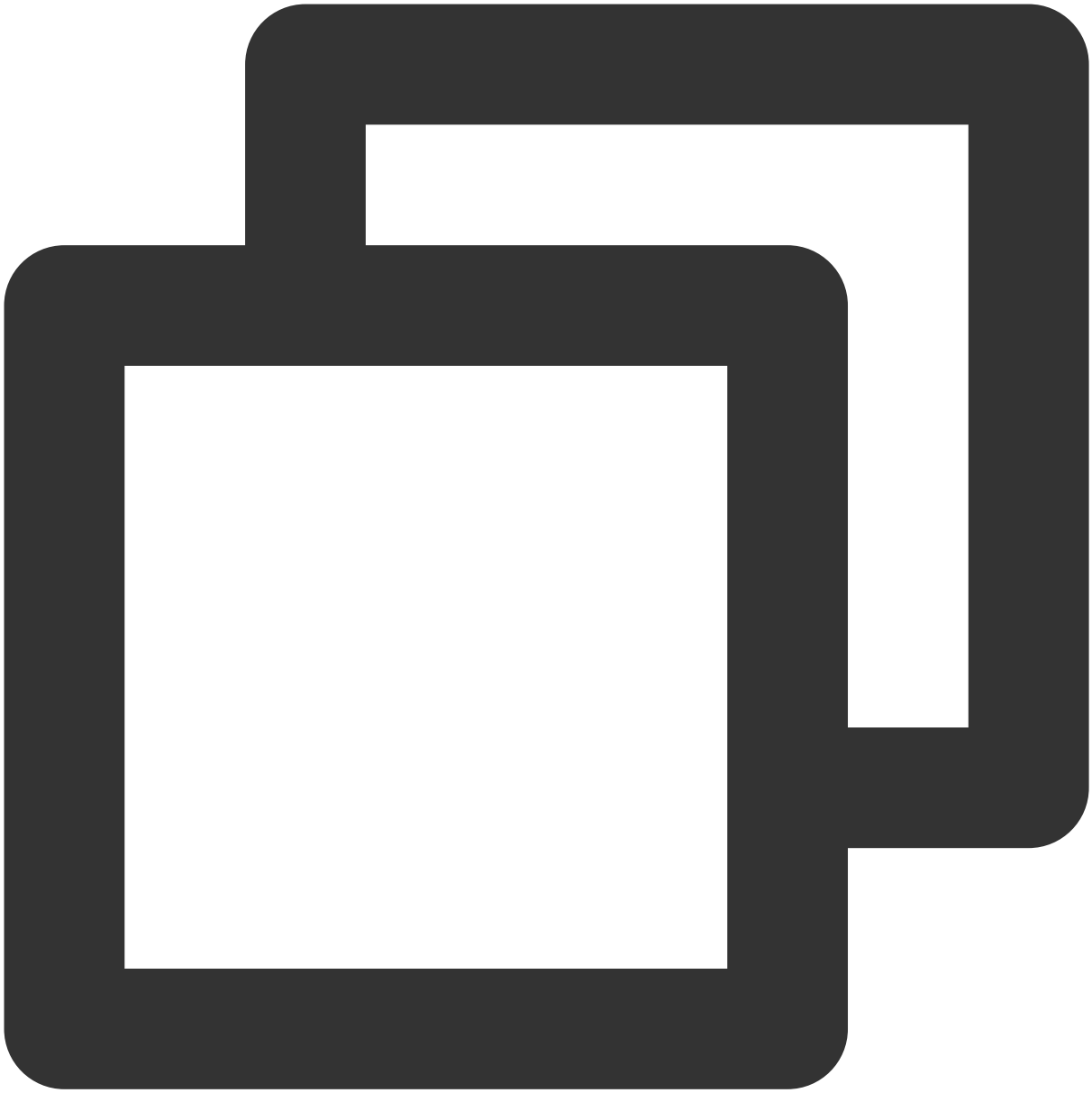
名称	类型	描述
BizID	Integer	应用的 AppID，可通过控制台 > 应用详情 查看。
RoomID	String	房间 ID
UserID	String	用户 ID
RecordMode	Integer	录制模式 0：单流 1：混流

Timestamp	Integer	发送回调时的时间戳（s）
TaskID	Integer	云录制服务分配的任务 ID。任务 ID 是对一次录制生命周期过程的唯一标识，结束录制时会失去意义。当您使用自定义录制模式时，任务 ID 可以在开始录制时通过响应参数获取，需要业务保存下来，作为下次针对这个录制任务操作的请求参数。
EventType	Integer	事件类型
Detail	EventDetail	事件详情，由 EventType 决定格式

EventDetail 事件详情说明

EventType	说明	Detail
1	音频文件启动录制	SeqNo：Number，分片序号 FileName：String，文件名
2	音频文件完成录制	SeqNo：Number，分片序号 FileName: String，文件名
3	音频文件上传完成	SeqNo：Number，分片序号 FileName：String，文件名

回调示例




```
{
    "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
```

```
        }
    }
```

万人范围语音

最近更新时间：2023-04-27 17:33:43

功能简介

在一个语音房间内，用户可以与一定距离内的其他用户进行实时语音通话。此功能通过业务层调用 GME 客户端接口更新声源方位，目的是告诉服务器本端位置，通过**本端世界坐标+本端接收音频的范围**，与**其他端世界坐标 + 其他端接收音频的范围**进行判断后，服务器对玩家范围内音频流进行转发，默认转发距离玩家最近的20路音频流，**最高可支持上万人同时在房间内开麦**。

应用场景

大逃杀游戏	提供**大逃杀类型游戏、生存射击类手游**中特有的“仅小队”或“所有人”的语音模式。在游戏设置中，根据玩家选择： 1、如果使用“仅小队”模式，则只能听见同队伍中队友说话的声音。 2、如果使用“所有人”模式，则可以听见队伍中队友声音，以及一定范围内其他对局玩家说话的声音。
沉浸式虚拟场景	例如**游戏演唱会**等虚拟场景中，可以通过范围语音实现演唱者在虚拟房间内唱歌，台下所有听众可以听到歌声的同时，能与自身一定范围内其他听众进行交流，最高可支持同房间内上万人同时开启麦克风。

体验效果

可以到 Demo 体验 中下载体验程序，体验 3D 音效及范围语音的效果。

基本概念

使用范围语音功能，涉及到**语音模式**、**语音接收范围**以及 **TeamID** 这三个概念。

语音模式

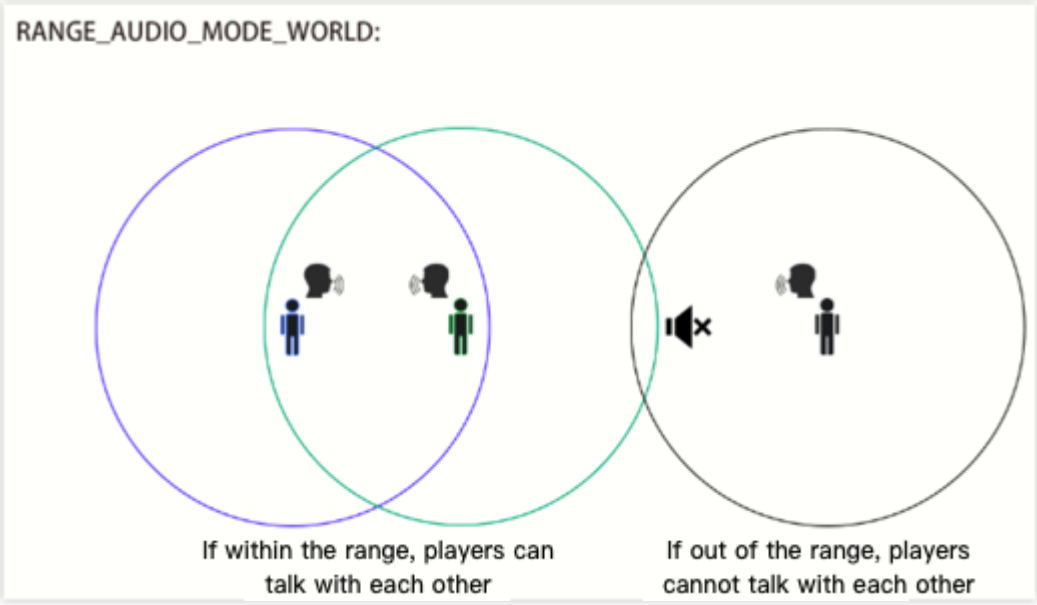
当进入范围语音房间时，有两种语音模式可供选择：

语音模式	参数名称	功能
所有人	RANGE_AUDIO_MODE_WORLD	设置后玩家附近一定范围的人都能听到该玩家讲话，如果范围内也有玩家设置为此模式，则也可以互相通话。 队友可以互相听到
仅小队	RANGE_AUDIO_MODE_TEAM	仅队友可以互相听到

注意

队友之间的通话不受距离以及语音模式的影响。

语音接收范围



如果设置的语音模式为**所有人(RANGE_AUDIO_MODE_WORLD)**，此时的语音接收范围受 UpdateAudioRecvRange 接口影响。

假设以下两个玩家 A 与 B 为不同的小队，且设置的语音模式为**所有人(RANGE_AUDIO_MODE_WORLD)**：

玩家坐标	语音接收范围	与另一玩家声音可达情况	与队友声音可达情况
A（0,0,0）	10米	可以听到玩家 B 的声音，因为 B 玩家距离 A 玩家在10米内	不影响同一小队成员互相通话
B（0,8,0）	5米	不可以听到玩家 A 的声音，因为 A 玩家与 B 玩家距离超过5米	不影响同一小队成员互相通话

说明

具体玩家声音可达情况请参见 附录。

TeamID

使用范围语音，需要调用 SetRangeAudioTeamID 接口设置小队号 TeamID，再调用 EnterRoom 接口进入语音房间。

当进入语音房间时指定的 TeamID != 0 时，进入范围语音房间模式。如果某成员使用 TeamID = 1 进入语音房间，当他设置语音模式为 RANGE_AUDIO_MODE_TEAM，则只有 TeamID = 1 的成员能听到他的声音；如果他设置的语音模式为 RANGE_AUDIO_MODE_WORLD，则除了 TeamID = 1 的成员，一定范围内的玩家也能听到他的声音。

TeamID 情况	语音模式	范围	声音可达情况
TeamID != 0，假设 TeamID = 1	RANGE_AUDIO_MODE_TEAM	10 米	声音只能和 TeamID = 1 的成员互通
	RANGE_AUDIO_MODE_WORLD	10 米	声音能和 TeamID = 1 的成员、以及语音模式设置为 RANGE_AUDIO_MODE_WORLD 的同房间10米范围内成员互通

如果某成员使用 TeamID = 0 进入语音房间，则为范围语音主持人模式，房间内所有人（不论语音模式是所有人还是仅小队）都可以听到该成员的声音。

TeamID 情况	TeamID 修改时机	范围	声音可达情况
TeamID = 0	进房前 TeamID != 0，进房后修改 TeamID = 0	10米	说话声音全房间成员（不论语音模式是所有人还是仅小队）都能听到 能与 TeamID = 0 的成员互相沟通 能听到设置为 RANGE_AUDIO_MODE_WORLD 的同房间10米范围内成员声音
	进房前 TeamID = 0，以 TeamID = 0 进入房间	10米	说话声音全房间成员（不论语音模式是所有人还是仅小队）都能听到 能与 TeamID = 0 的成员互相沟通 不能听到房间内其他人说话声音

示例场景

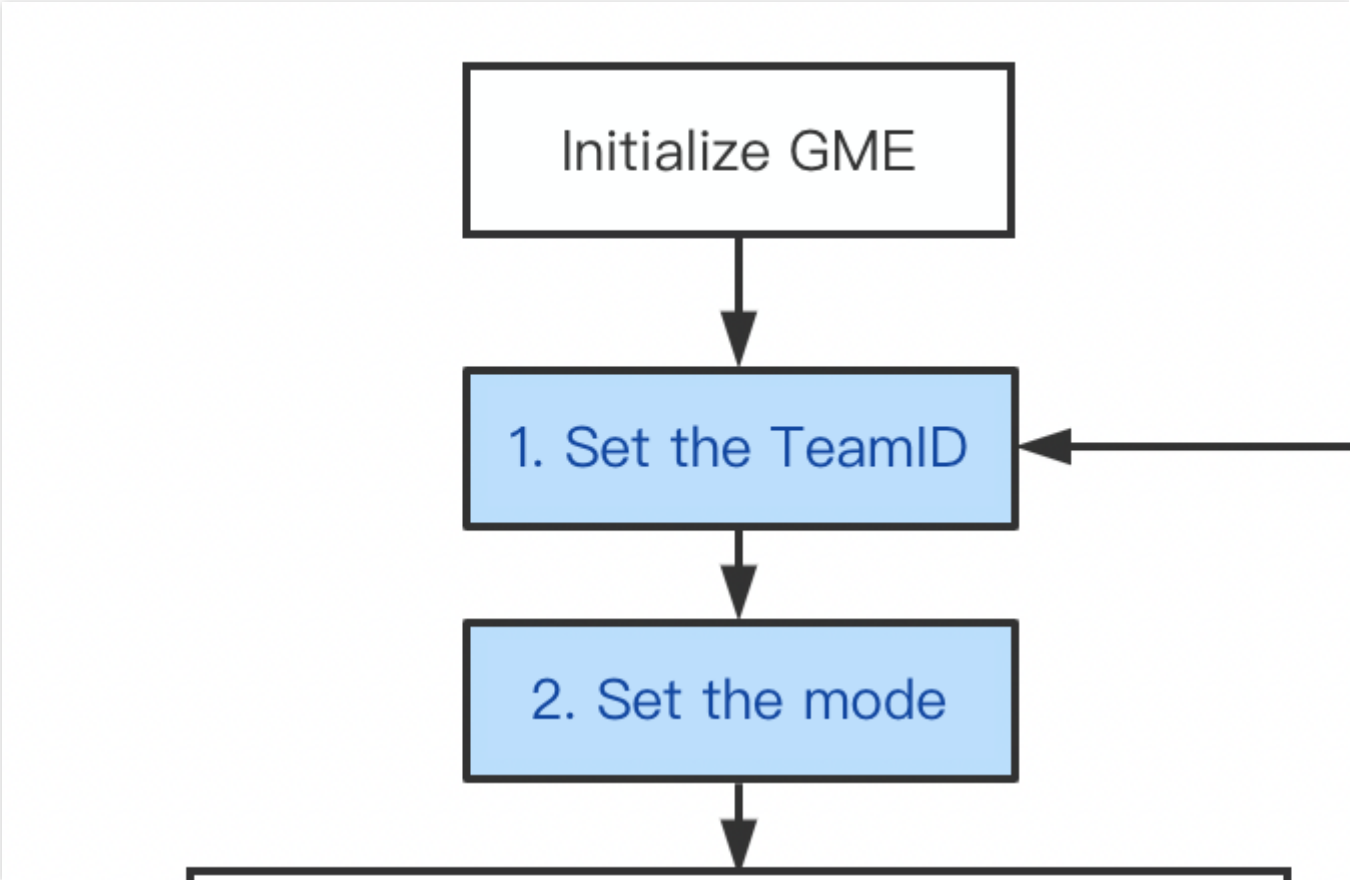
大逃杀游戏：例如一个大逃杀类型的游戏，每4个人为一个队伍，则这4个人需要设置一个小队号 TeamID，每100人为一个对局房间，一个对局共25个小队，则25个小队都进去一个语音房间。在对局中，如果某玩家想和10米范围内的陌生人沟通，则将语音距离范围设置为10，将语音模式设置为 RANGE_AUDIO_MODE_WORLD，同时打开麦克风及扬声器。如果他只想和小队成员沟通，不和非小队的成员沟通，则只需要将语音模式设置为 RANGE_AUDIO_MODE_TEAM。

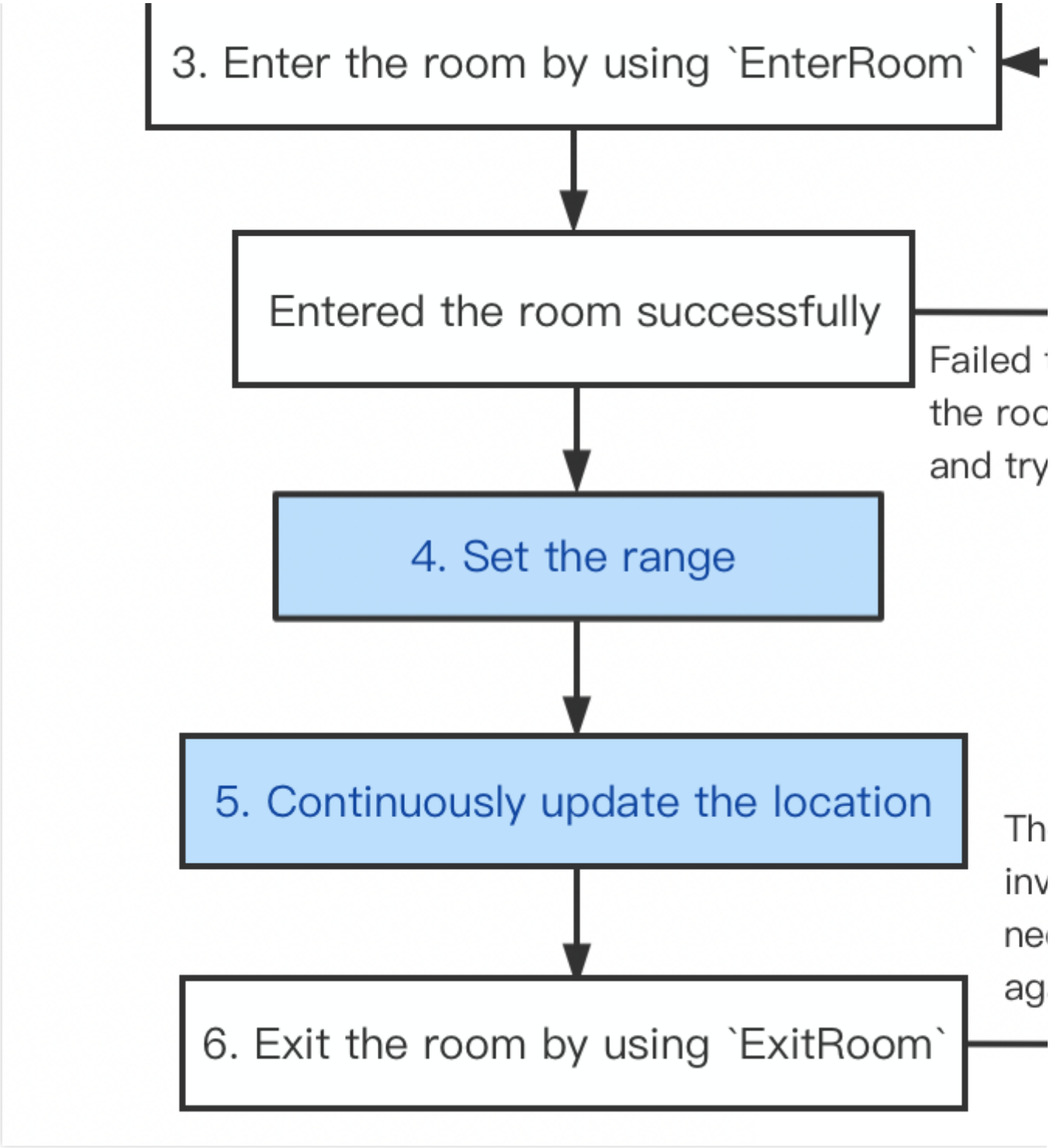
游戏演唱会：在游戏中如果需要举办演唱会，歌手不需要和游戏玩家有互动，可以让游戏玩家通过 TeamID = OpenID 进入范围语音房间，设置语音模式为 RANGE_AUDIO_MODE_WORLD，依照游戏玩法设置语音距离范围，这样游戏玩家可以和附近玩家沟通交流；歌手将 TeamID 设置为 0 后进入房间，歌手的声音整个房间的人都能听见，但歌手听不见其他人的声音。
主持人模式：在游戏中例如虚拟桌游场景，主持人说话声音即要房间内所有人听见，也要听见范围内玩家说话的声音，可以让主持人先以 TeamID != 0 的形式进入房间，进房后将 TeamID 设置为 0，此时主持人说话全房间的人都能听见，主持人也能听见范围内玩家说话的声音。

前提条件

已开通实时语音服务：可参见 语音服务开通指引。
已接入GME SDK：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入、Unity SDK 快速接入、Unreal SDK 快速接入。
万人范围语音：如果同房间内使用范围语音人数超过1千人，请 提交工单 联系 GME 开发者。

使用流程





注意

请务必参照此流程调用接口。

流程图中蓝色部分为范围语音所需流程。

有别于普通小队语音房间，在使用范围语音能力时，**必须使用流畅音质进房**。

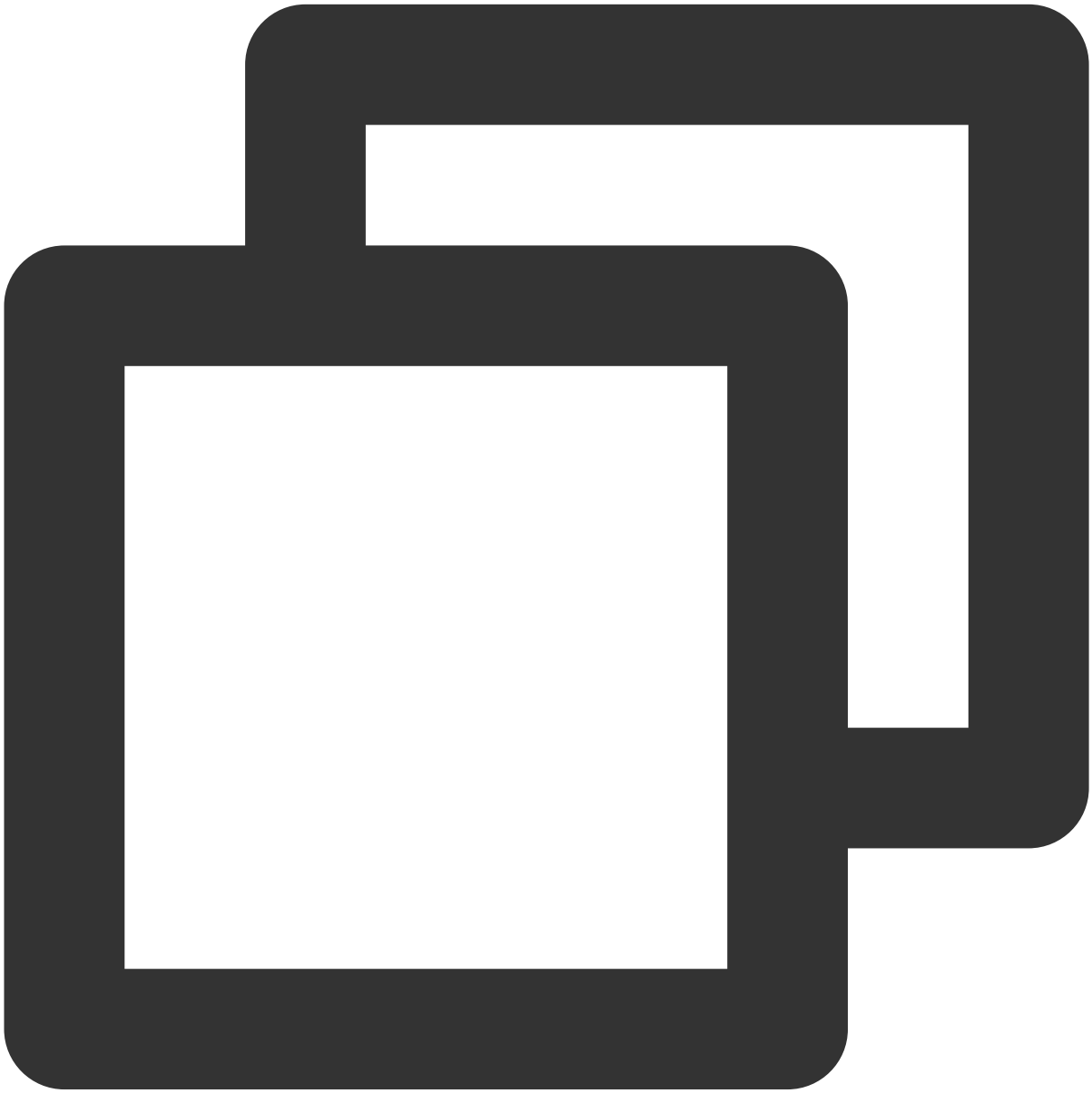
在进房成功后，调用 UpdateAudioRecvRange（至少一次），及每帧调用 UpdateSelfPosition。

1. 设置 TeamID

进房前，通过此接口设置队伍号，影响下一次进房。

进房后，可通过此接口修改队伍号，设置后立即生效。
退房后，TeamID 不会自动重置为0，所以一旦决定调用此语音模式，请在每次 EnterRoom 之前都调用此方法设置 TeamID。
退房后再进房，请在退房成功回调回来之后再调用设置队伍号接口。

函数原型



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

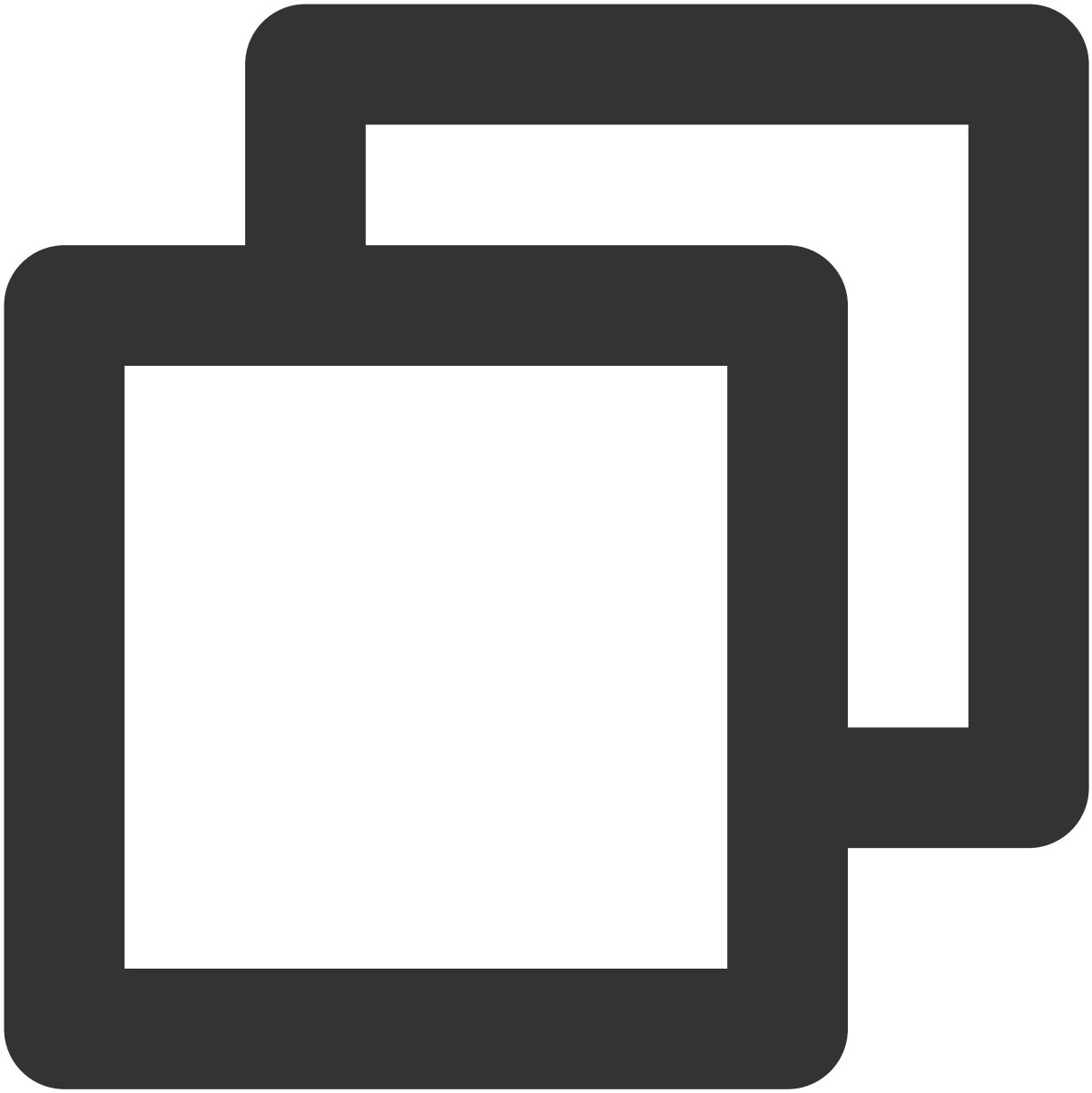
参数	类型	意义

teamID	int	队伍号，专供范围语音模式中使用。当 TeamID 为0时，通话模式为普通小队语音，默认0。

2. 设置语音模式

进房前，通过调用此接口修改语音模式，影响下一次进房。
进房后，通过调用此接口修改语音模式，将直接改变当前用户的语音模式。
退房后，此参数不会自动重置为 MODE_WORLD，所以一旦决定调用此方法，请在每次 EnterRoom 之前都调用此方法设置语音模式。

函数原型



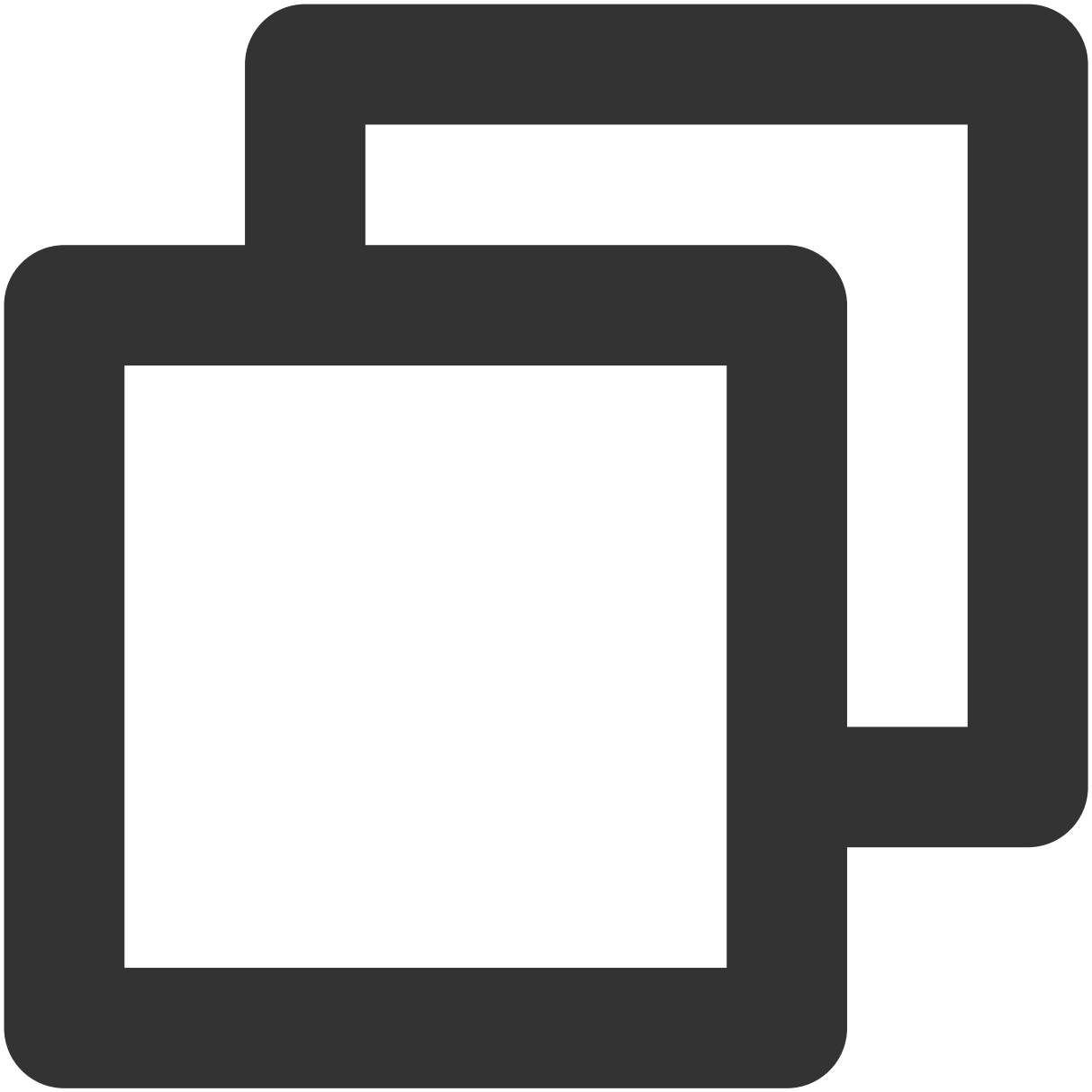
```
ITMGRoom int SetRangeAudioMode(RANGE_AUDIO_MODE rangeAudioMode)
```

参数	类型	意义
rangeAudioMode	int	0(MODE_WORLD) 代表“所有人”，1(MODE_TEAM) 代表"仅小队"

3. 进入语音房间

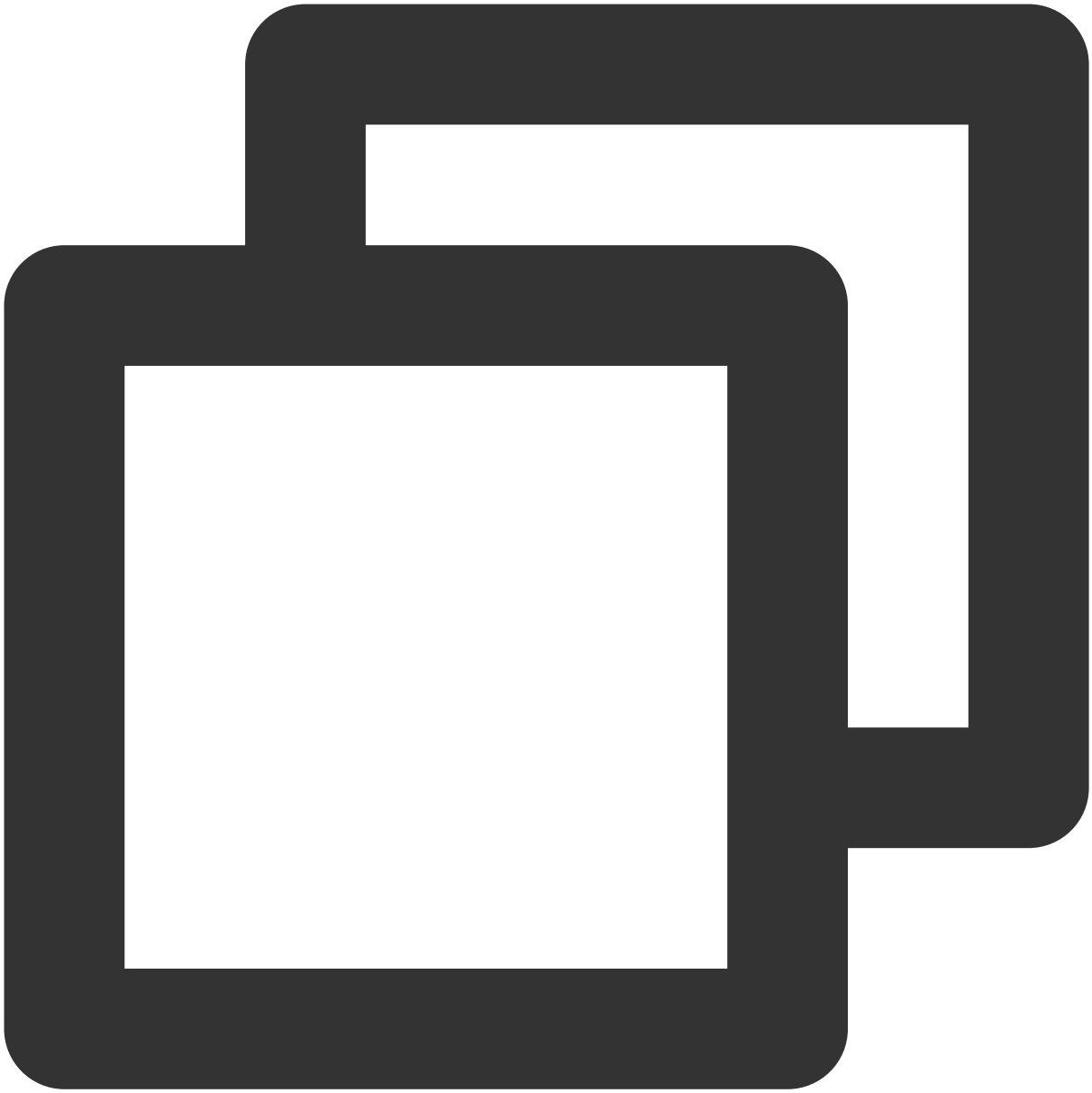
在调用 EnterRoom 之前，需调用以下两个 API：SetRangeAudioTeamID、 SetRangeAudioMode。

函数原型



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

一定要使用流畅音质进入语音房间，随后监听进房的回调并进行处理。



```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ENTER_ROOM == type)
        {
            //对事件返回的 Data 进行解析
            int nErrCode = data.getIntExtra("result" , -1);
            String strErrMsg = data.getStringExtra("error_info");

            if (nErrCode == AVError.AV_OK)
            {
                //收到进房信令，进房成功，可以操作设备
                ScrollView_ShowLog("EnterRoom success");
```

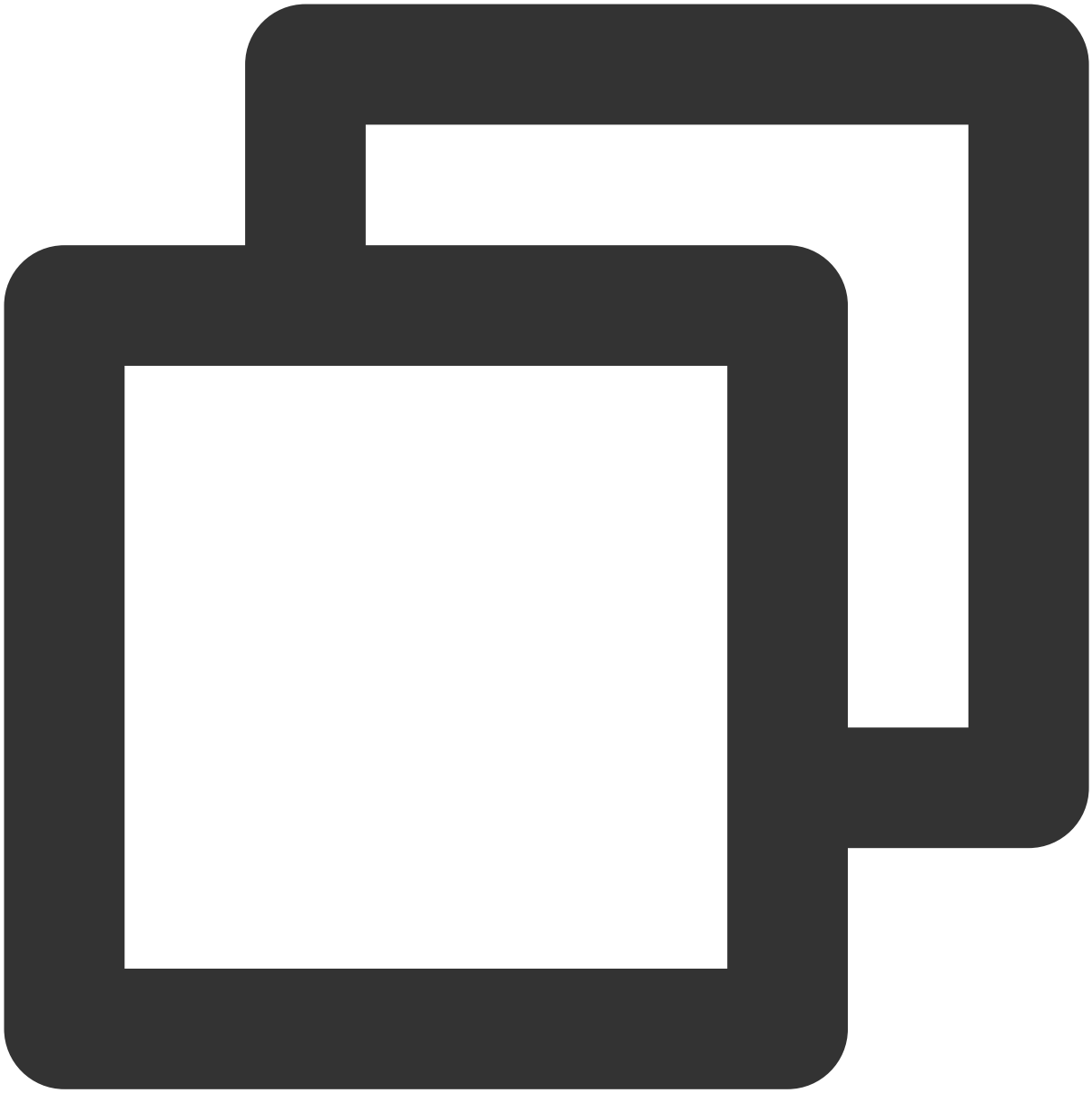
```
                Log.i(TAG,"EnterRoom success!");
            }
            else
            {
                //进房失败，需分析返回的错误信息
                ScrollView_ShowLog("EnterRoom fail :" + strErrMsg);
                Log.i(TAG,"EnterRoom fail!");
            }
        }
    }
```

在进房成功后，调用 UpdateAudioRecvRange（至少调用一次），及每帧调用 UpdateSelfPosition。

4. 设置接收语音距离范围

通过此方法用于设置接收的语音范围（距离以游戏引擎为准），只支持在**进房成功后调用**。
此方法必须配合 UpdateSelfPosition 更新声源方位联合使用。
此方法只需调用一次即可生效，支持修改。

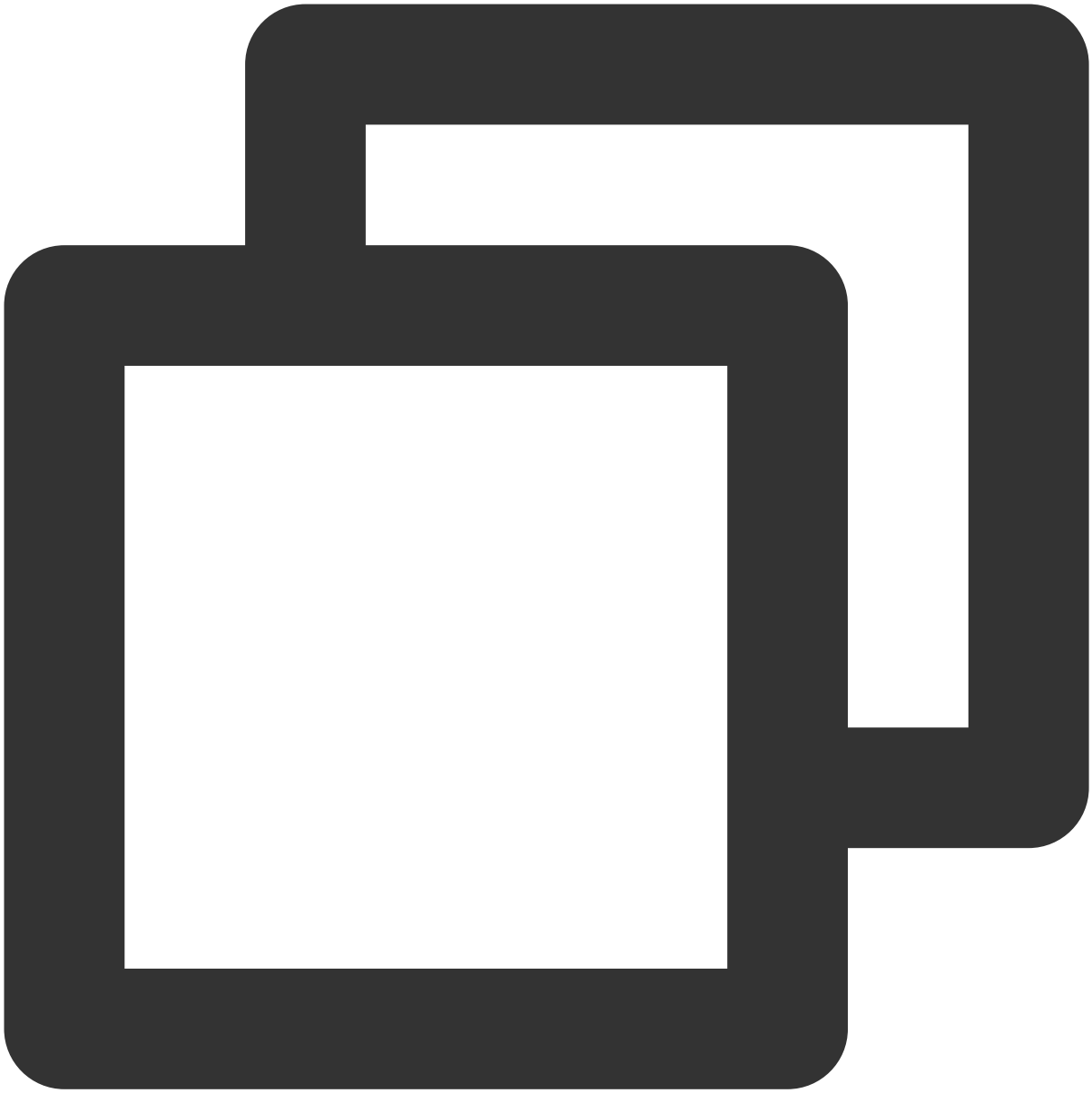
函数原型



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

参数	类型	意义
range	int	最大可以接收音频的范围，单位为引擎距离单位

示例代码



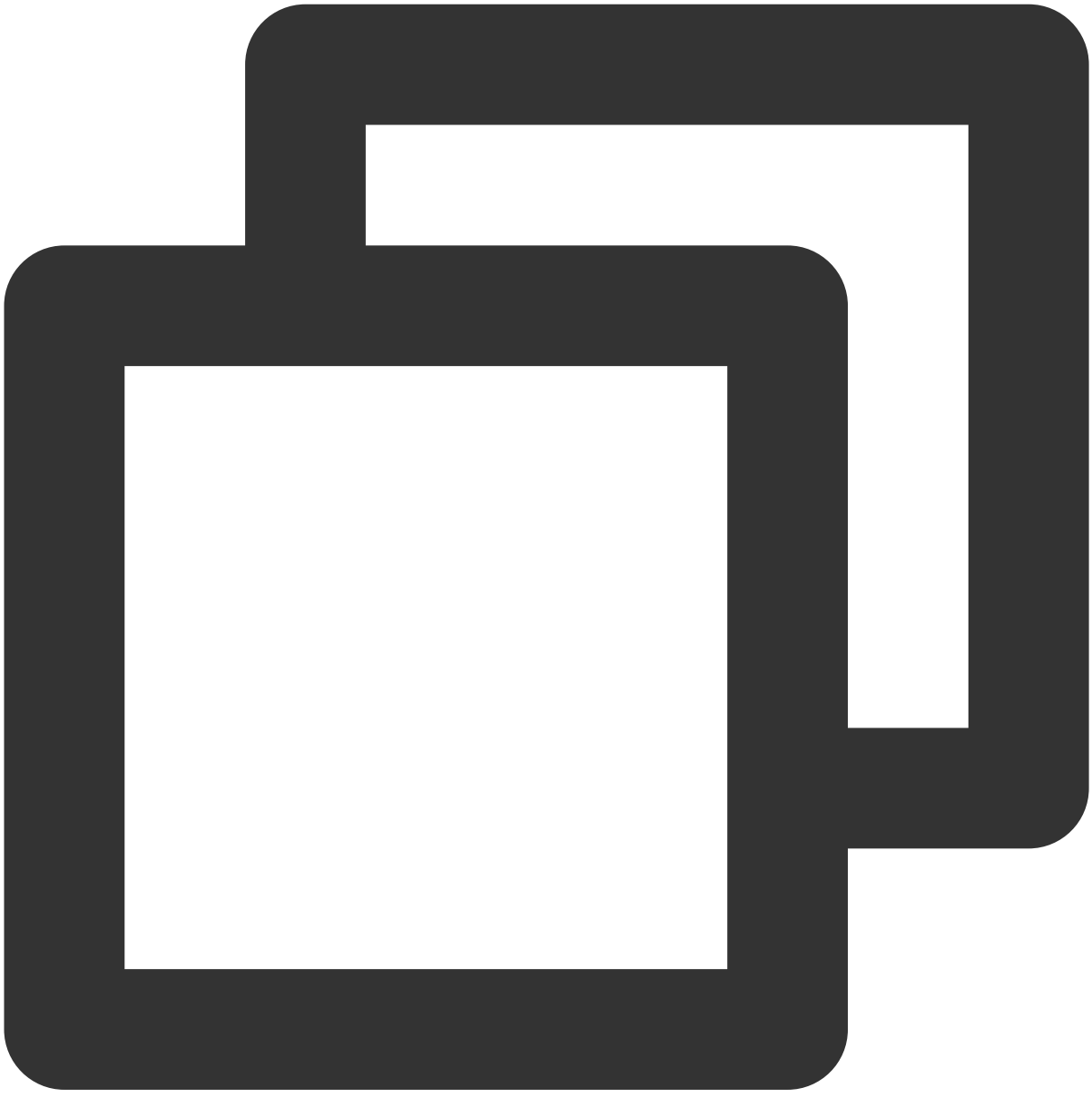
```
ITMGContext.GetInstance().GetRoom().UpdateAudioRecvRange(300);
```

5. 更新声源方位

更新声源方位，目的是告诉服务器本端位置，通过**本端世界坐标+本端接收音频的范围**，与**其他端世界坐标+其他端接收音频的范围**进行判断，达到范围语音效果。
此函数用于更新声源位置信息，只支持在**进房成功后调用**，且需要**每帧调用**，以 Unity 引擎为例，此接口需要在 Update 中调用。
使用范围语音一定要更新声源方位，如果不需要范围判断能力，**也需要进房后调用此接口一次**。

如果需要同时使用 3D 音效效果，此接口中的参数 axisForward、axisRight 及 axisUp 需要按照下文 3D 语音部分 进行设置。

函数原型



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

参数	类型	意义
position	int[]	自身在世界坐标系中的坐标，顺序是前、右、上

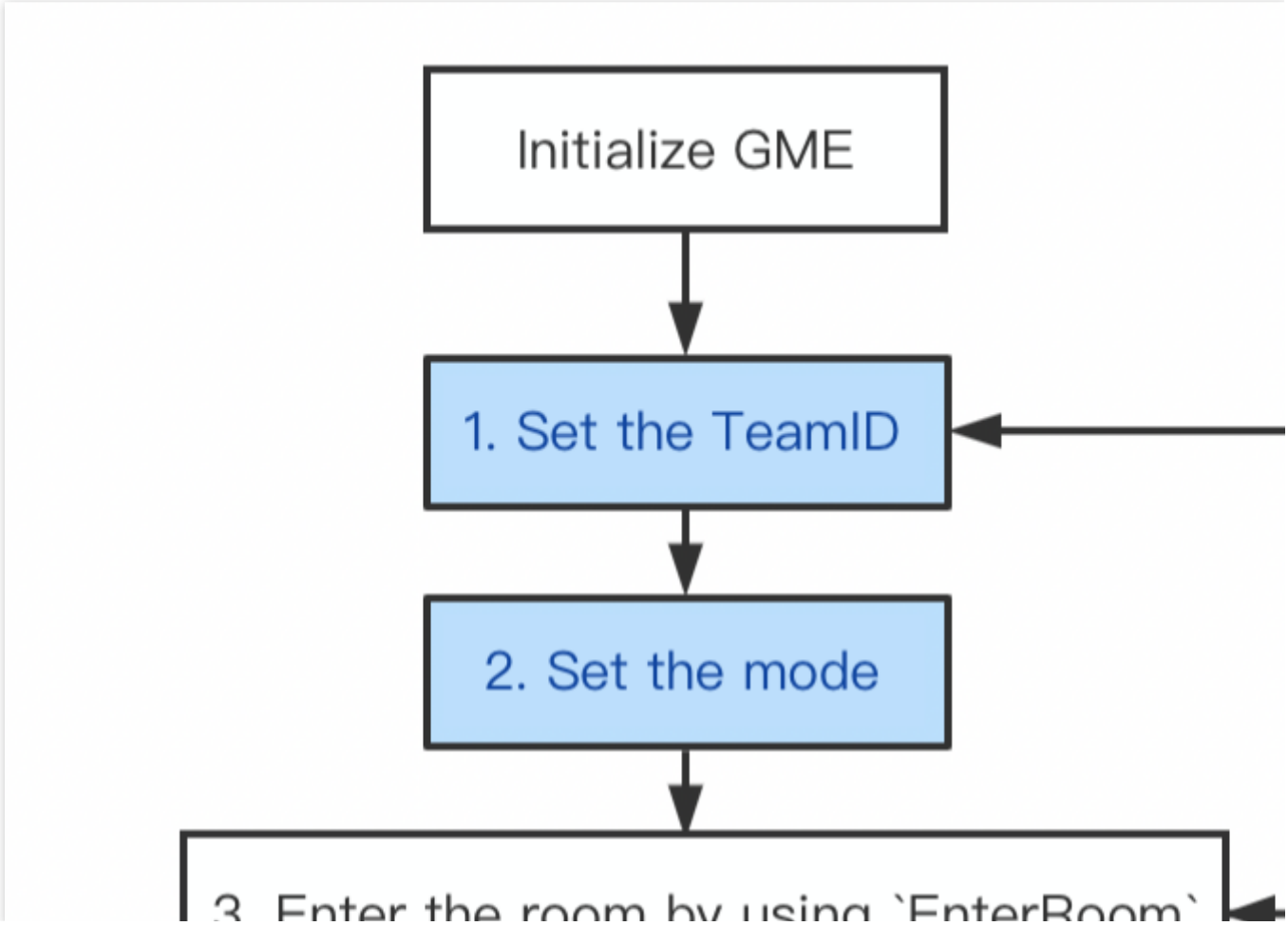
axisForward	float[]	在本产品中无需关注
axisRight	float[]	在本产品中无需关注
axisUp	float[]	在本产品中无需关注

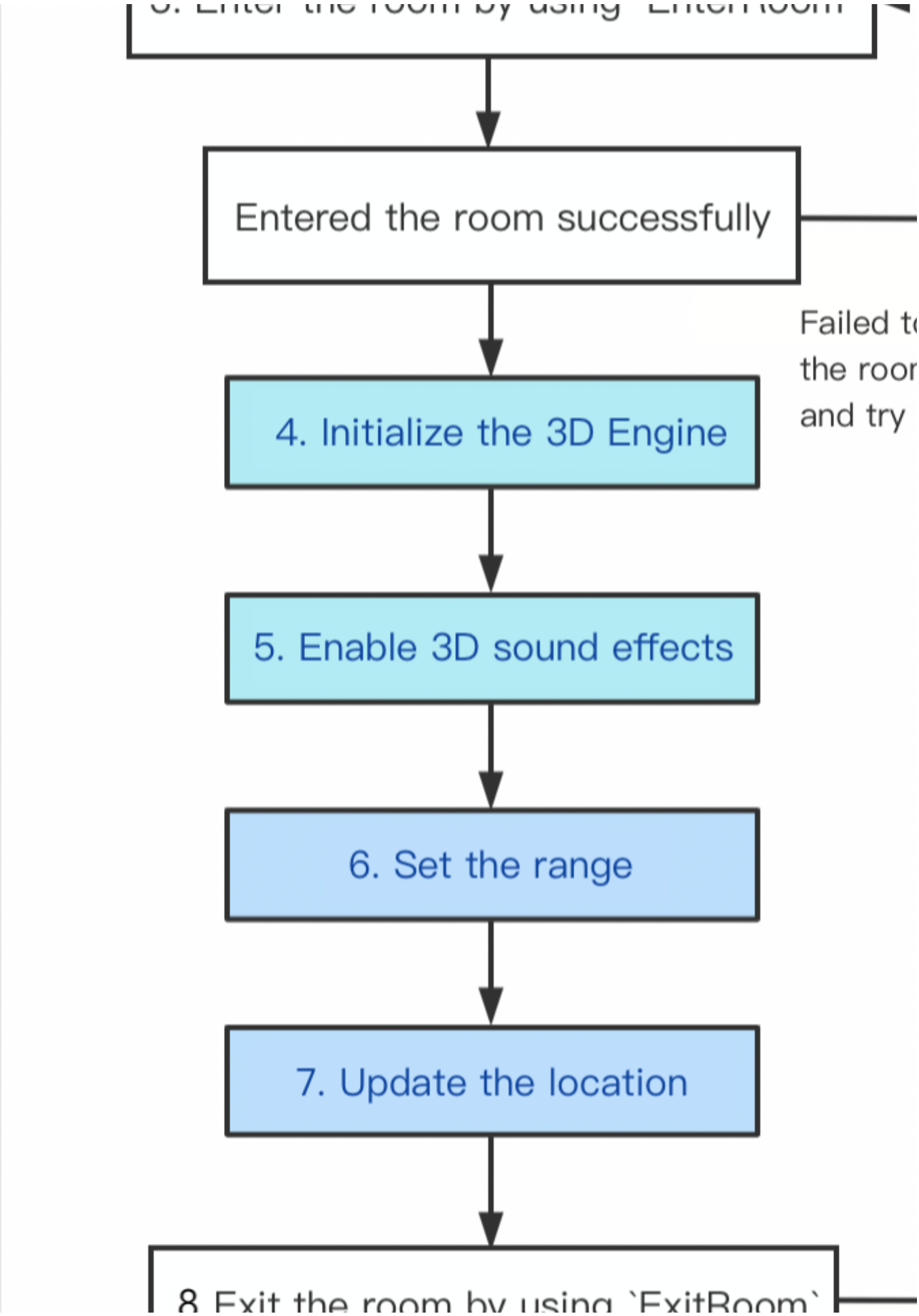
范围语音结合 3D 音效

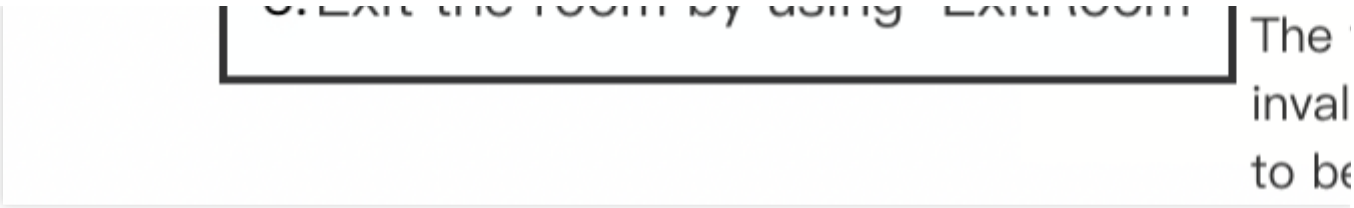
上文所介绍的范围语音功能，即通过距离来进行控制声音的可达性，如果需要更加沉浸式的体验，建议配合 3D 音效一起使用。

使用流程

使用范围语音的同时，如果需要同时使用 3D 音效功能，需要以下几个步骤，在进房后使用范围语音步骤1、2、3完成后，初始化 3D 引擎以及打开3D音效。




Entered the room successfully
Failed t
the roor
and try
4. Initialize the 3D Engine
5. Enable 3D sound effects
6. Set the range
7. Update the location
8 Exit the room by using `ExitRoom`



说明

流程图中绿色部分为3D语音所需流程。

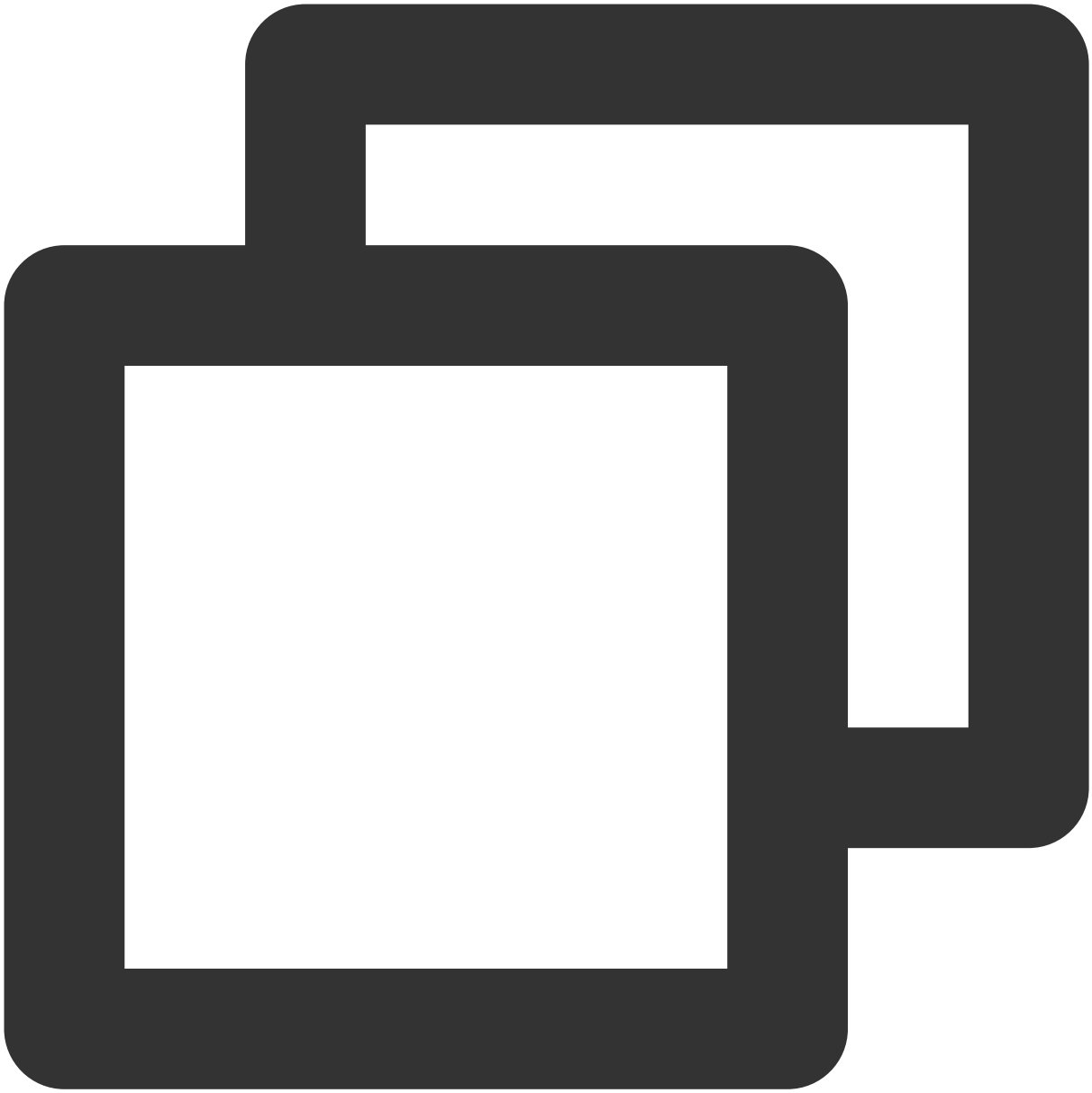
前提条件

参考 范围语音使用流程，完成步骤1、2、3。

4. 初始化 3D 音效引擎

此函数用于初始化 3D 音效引擎，在进房后调用。在使用 3D 音效之前必须先调用此接口，只接收 3D 音效而不发出 3D 音效的用户也需要调用此接口。

函数原型



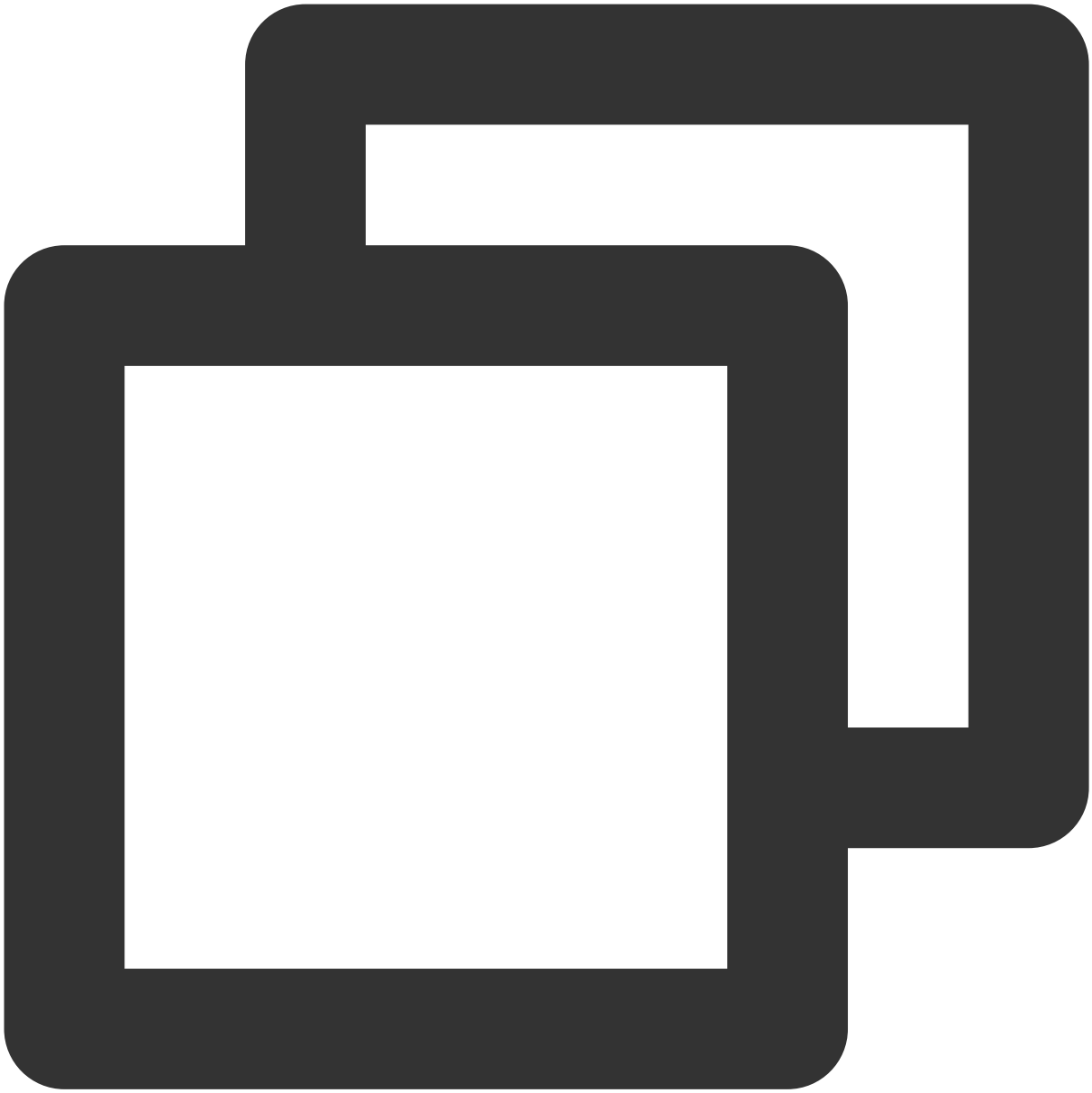
```
public abstract int InitSpatializer(string modelPath)
```

参数	类型	意义
modelPath	string	填空即可

5. 开启或关闭 3D 音效

此函数用于开启或关闭 3D 音效。开启之后可以听到 3D 音效。

函数原型



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

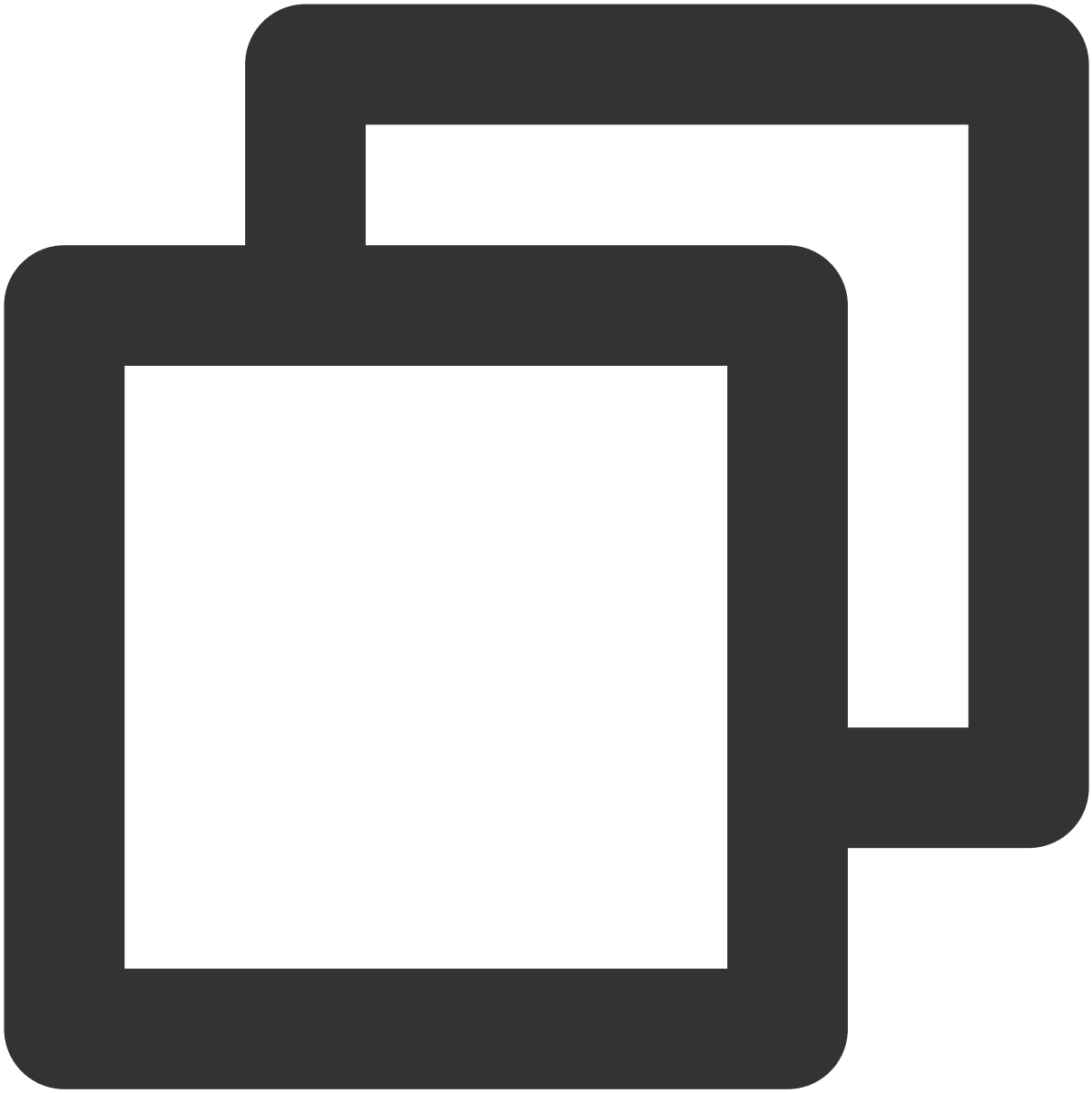
参数	类型	意义
enable	bool	开启之后可以听到 3D 音效
applyToTeam	bool	3D 语音是否作用于小队内部，仅 enble 为 true 时有效

通过 IsEnableSpatializer 接口获取 3D 音效状态。

6. 设置接收语音距离范围（3D）

通过此方法用于设置接收的语音范围（距离以游戏引擎为准），只支持在**进房成功后调用**。
此方法必须配合 UpdateSelfPosition 更新声源方位联合使用。
此方法只需调用一次即可生效。
3D 音效中，音源音量的大小与音源距离有一定的衰减关系。单位距离超过 range 之后，音量衰减到几乎为零。
距离与声音衰减的关系参考文档附录。

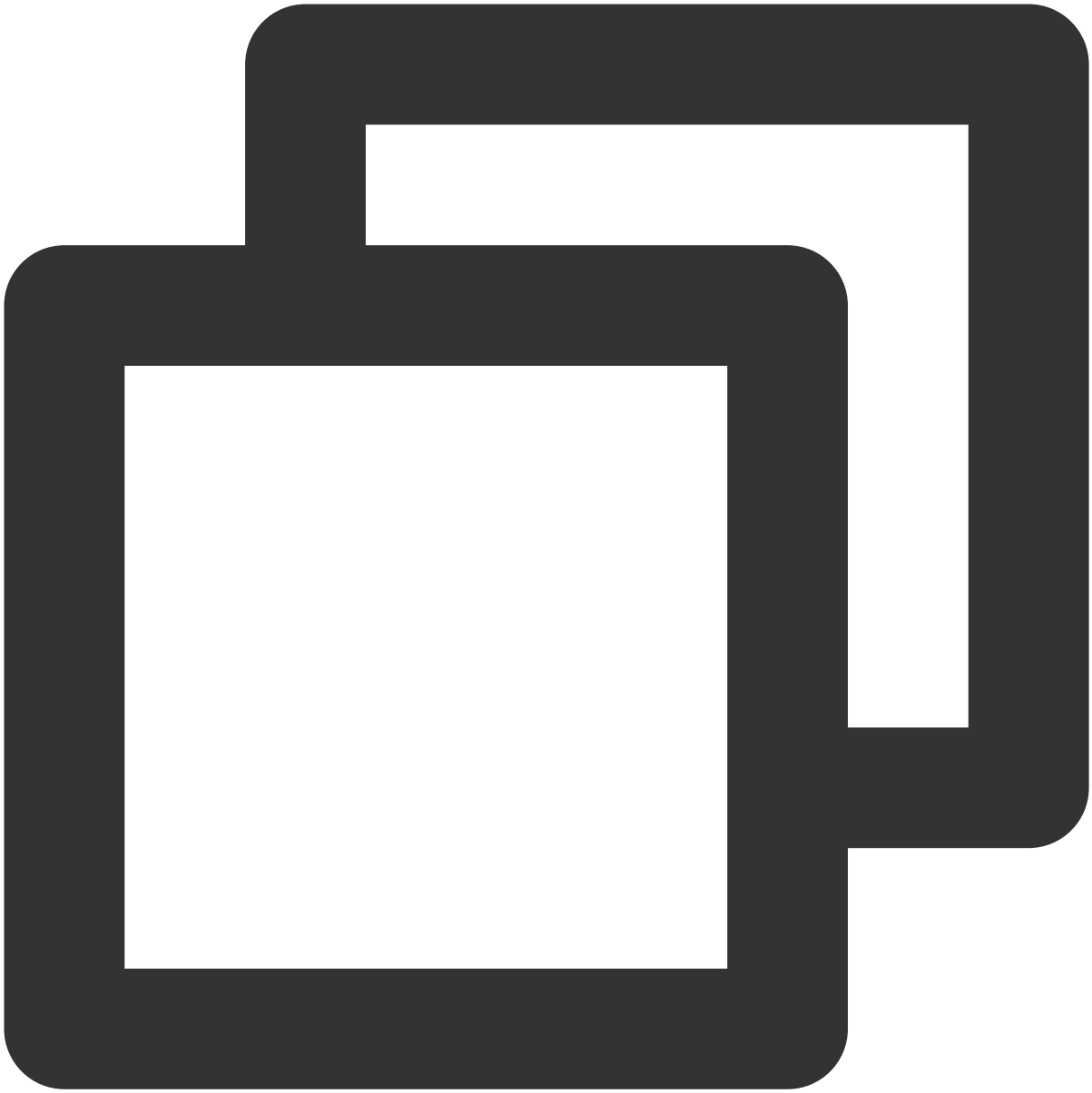
函数原型



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

参数	类型	意义
range	int	最大可以接收音频的范围，单位为引擎距离单位

示例代码



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

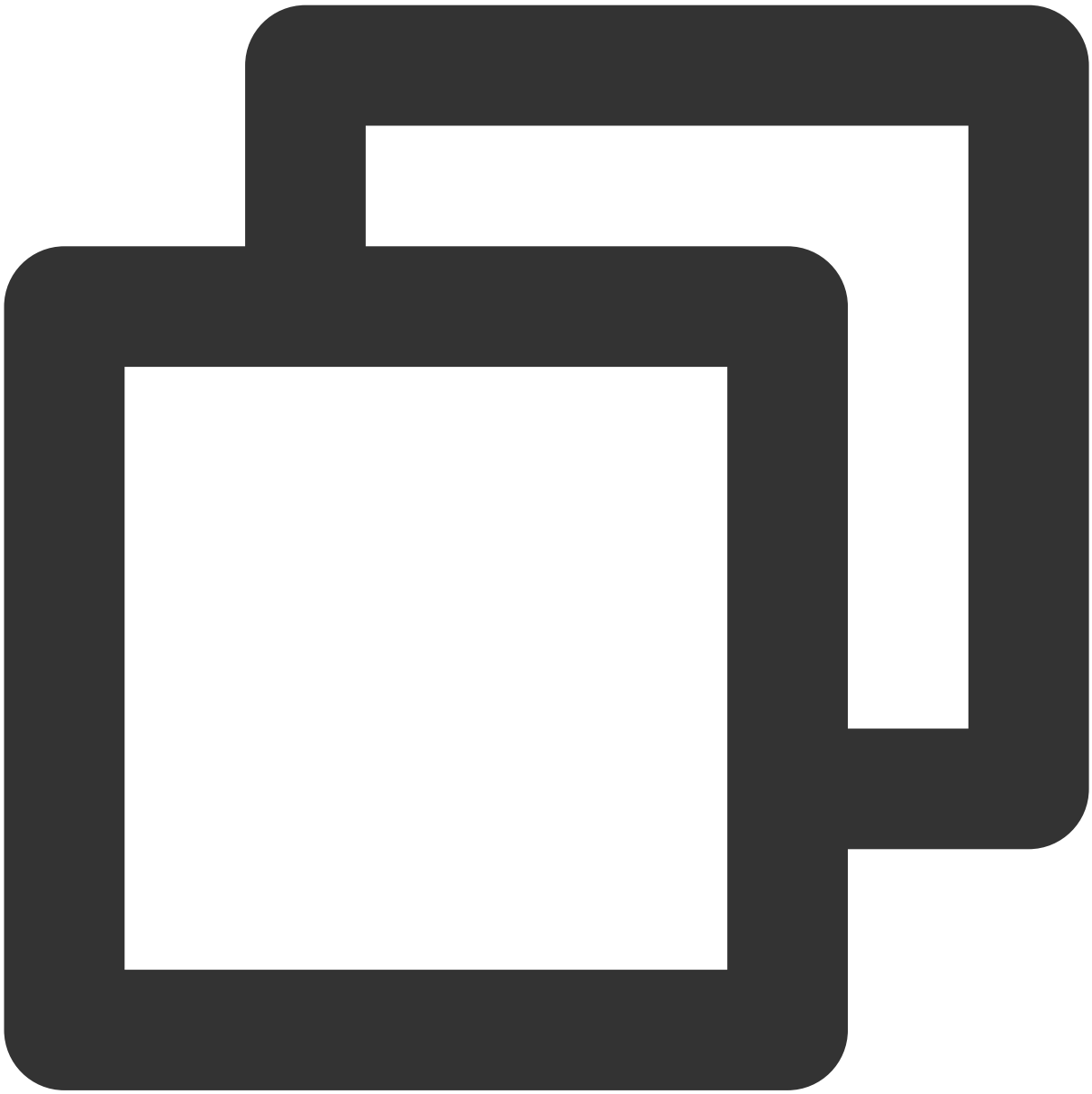
7. 更新声源方位（3D）

更新声源方位，目的是告诉服务器本端位置，通过**本端世界坐标+本端接收音频的范围**，与**其他端世界坐标+其他端接收音频的范围**进行判断，达到范围语音效果。

此函数用于更新声源位置信息，只支持在**进房成功后调用**，且需要**每帧调用**，以 Unity 引擎为例，此接口需要在 Update 中调用。

请直接复制并调用示例代码使用此功能。

函数原型



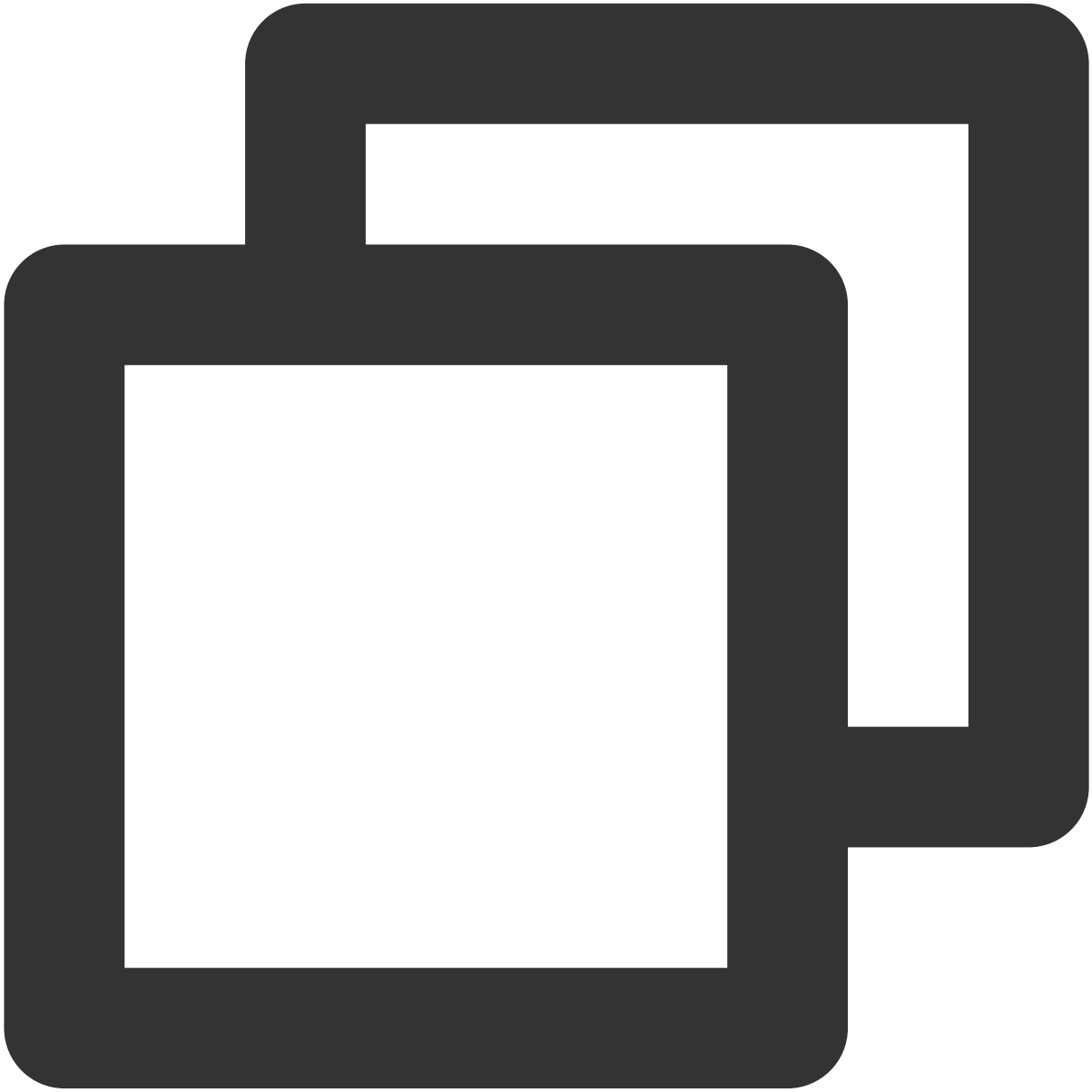
```
public abstract int UpdateSelfPosition(int position[3], float axisForward[3], float
```

参数	类型	意义
position	int[]	自身在世界坐标系中的坐标，顺序是前、右、上
axisForward	float[]	自身坐标系前轴的单位向量

axisRight	float[]	自身坐标系右轴的单位向量
axisUp	float[]	自身坐标系上轴的单位向量

示例代码

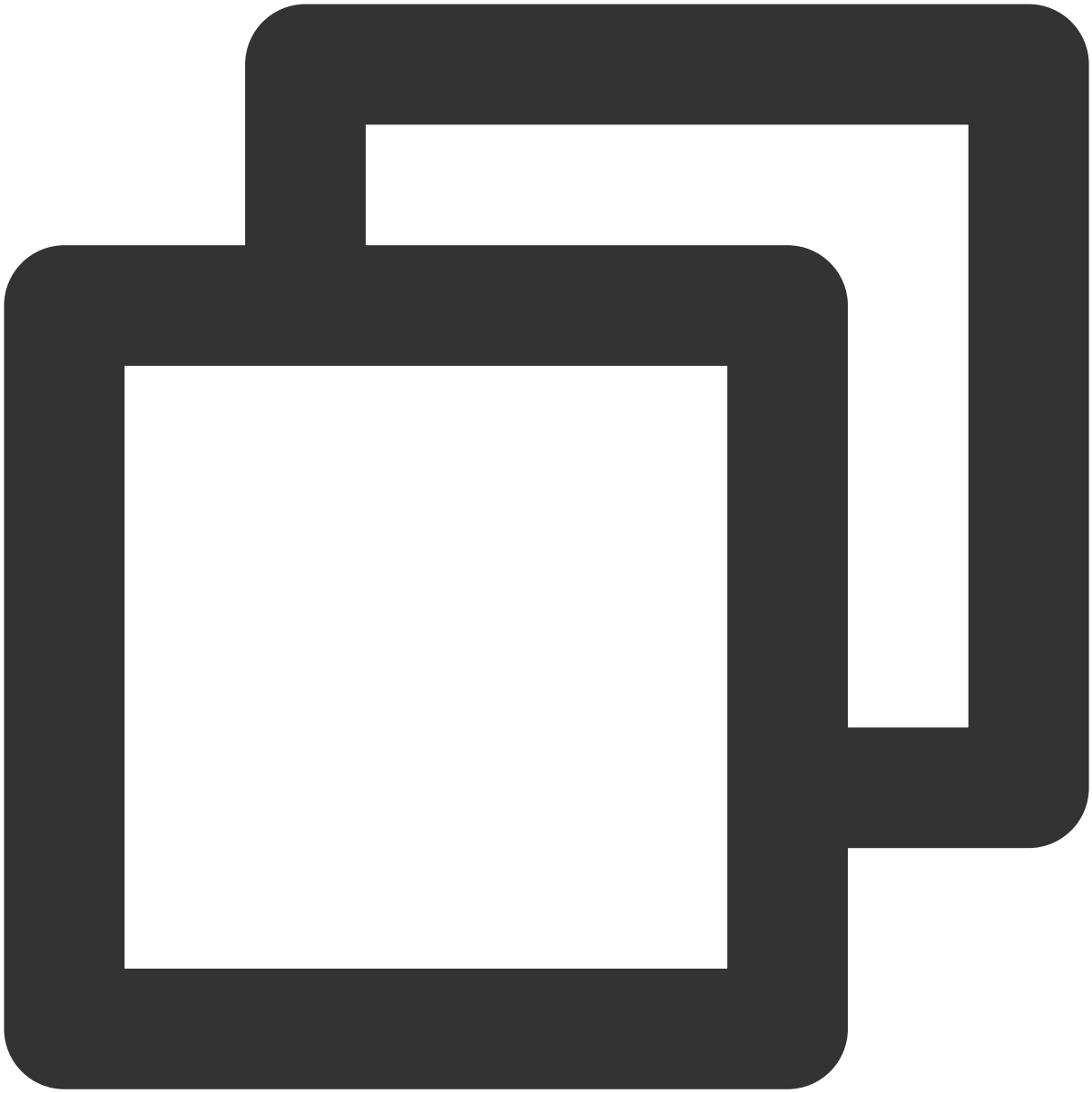
Unreal



```
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



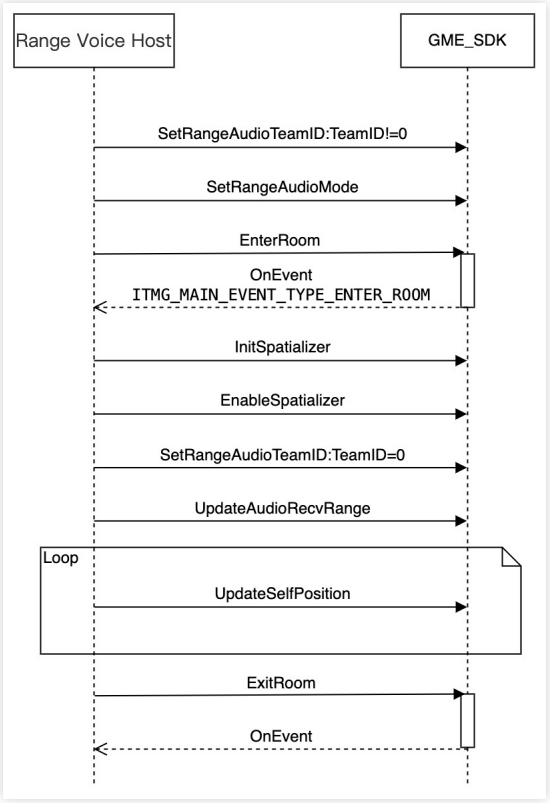
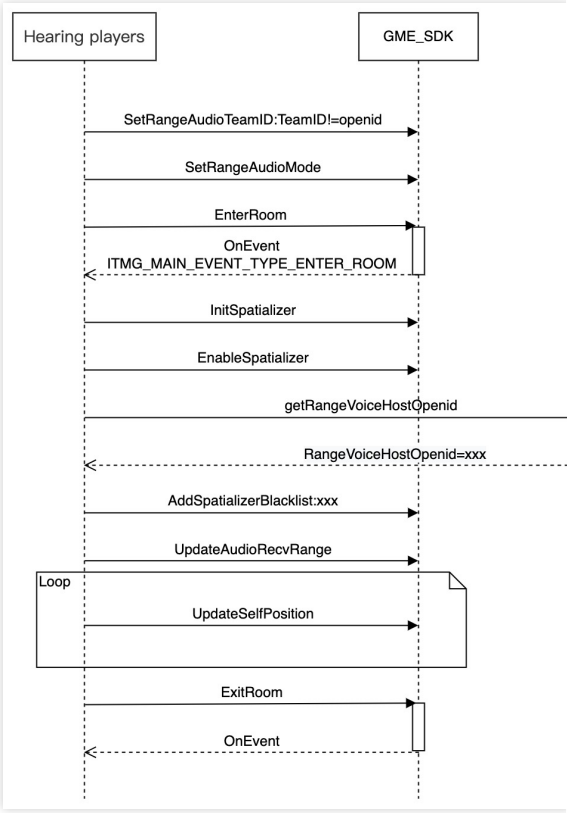
```
Transform selftrans = currentPlayer.gameObject.transform;
Matrix4x4 matrix = Matrix4x4.TRS(Vector3.zero, selftrans.rotation, Vector3.one);
int[] position = new int[3] {
    selftrans.position.z,
    selftrans.position.x,
    selftrans.position.y};
float[] axisForward = new float[3] {
    matrix.m22,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

对主持人禁止使用 3D 音效

如果场景中有玩家使用范围语音主持人模式，即他的声音需要全房间玩家都能收听到，需要参考 3D 语音黑名单接口，在所有收听端将主持人加入到收听端的 3D 语音黑名单中，避免出现 3D 语音功能衰减效果对主持人声音可达性的影响。

不同角色的 API 调用时序如下：

主持人 API 时序	听众 API 时序
	

附录

不同语音模式

不同语音模式下，玩家声音可达情况：

假设 A 玩家状态为“所有人”，对应 B 玩家在不同语音模式下声音可达情况：

是否同一小队	是否范围内	语音模式	A与B 是否能相互听到对方的声音
同一小队	是	MODE_WORLD	是
		MODE_TEAM	是
	否	MODE_WORLD	是
		MODE_TEAM	是
不同小队	是	MODE_WORLD	是
		MODE_TEAM	否
	否	MODE_WORLD	否
		MODE_TEAM	否

假设 A 玩家状态为“仅小队”，对应 B 玩家在不同语音模式下声音可达情况：

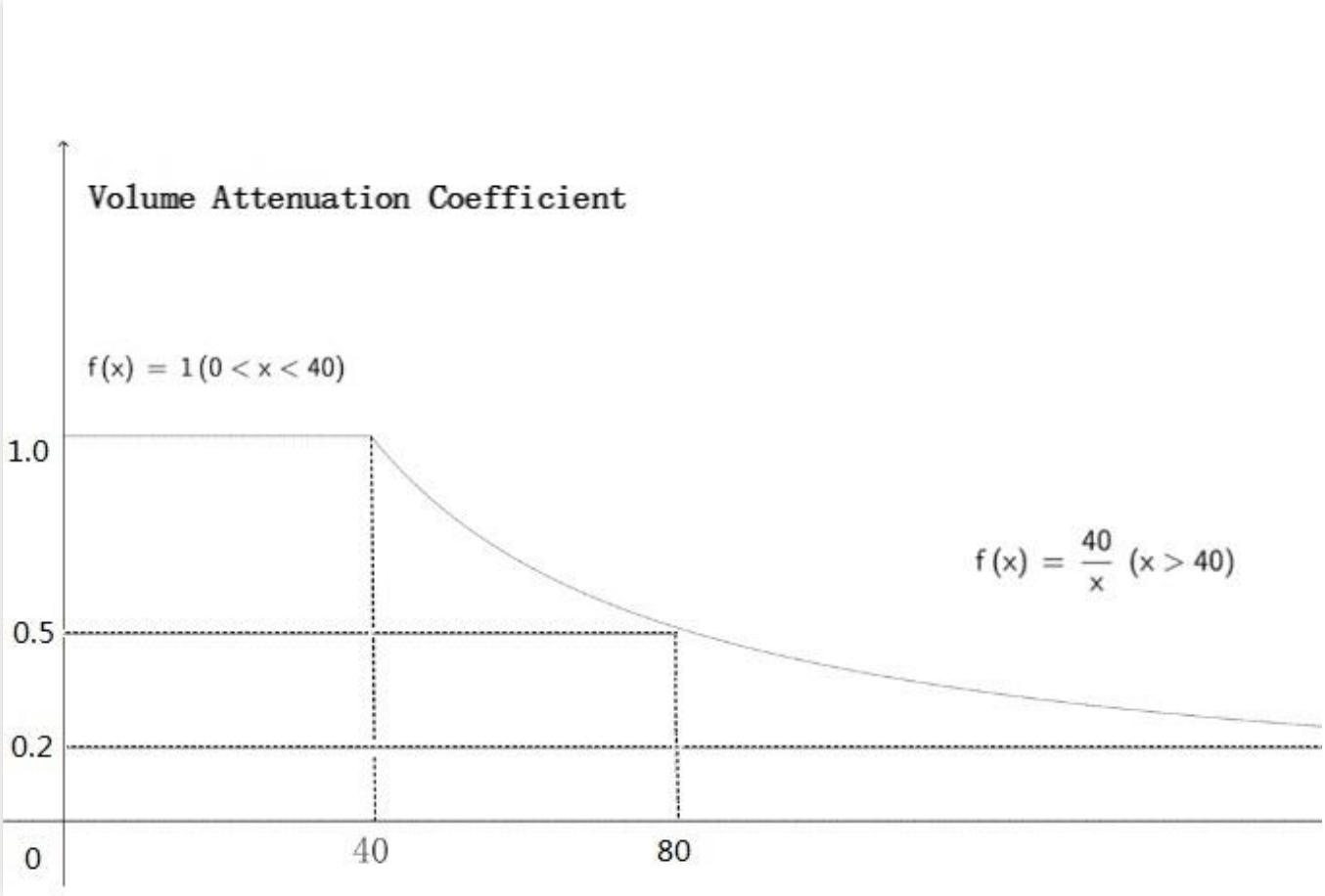
是否同一小队	是否范围内	语音状态	A与B 是否能相互听到对方的声音
同一小队	是	MODE_WORLD	是
		MODE_TEAM	是
	否	MODE_WORLD	是
		MODE_TEAM	是
不同小队	是	MODE_WORLD	否
		MODE_TEAM	否
	否	MODE_WORLD	否
		MODE_TEAM	否

距离与声音衰减的关系

3D 音效中，音源音量的大小与音源距离有一定的衰减关系。单位距离超过 range 之后，音量衰减到几乎为零。

距离范围（引擎单位）	衰减公式

0 < N < range/10	衰减系数：1.0 （音量无衰减）
N ≥ range/10	衰减系数：range/10/N



3D 音效

最近更新时间：2023-04-27 17:34:33

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文向您介绍游戏多媒体引擎 3D 音效的接入技术。

使用场景

在普通的进房实时语音中，玩家的声音不具有 3D 音效的效果，玩家之间只能进行很简单的互动；而引入 3D 位置语音之后，玩家在喊话过程中会暴露自己的方位和位置信息，玩家的声音也会根据位置变化而实时改变。可以说，3D 音效让《大逃杀》这种玩家间的沟通和战斗体验更真实，感受更加沉浸式、更加身临其境的吃鸡玩法。
点击 下载demo ，体验 3D 音效效果。

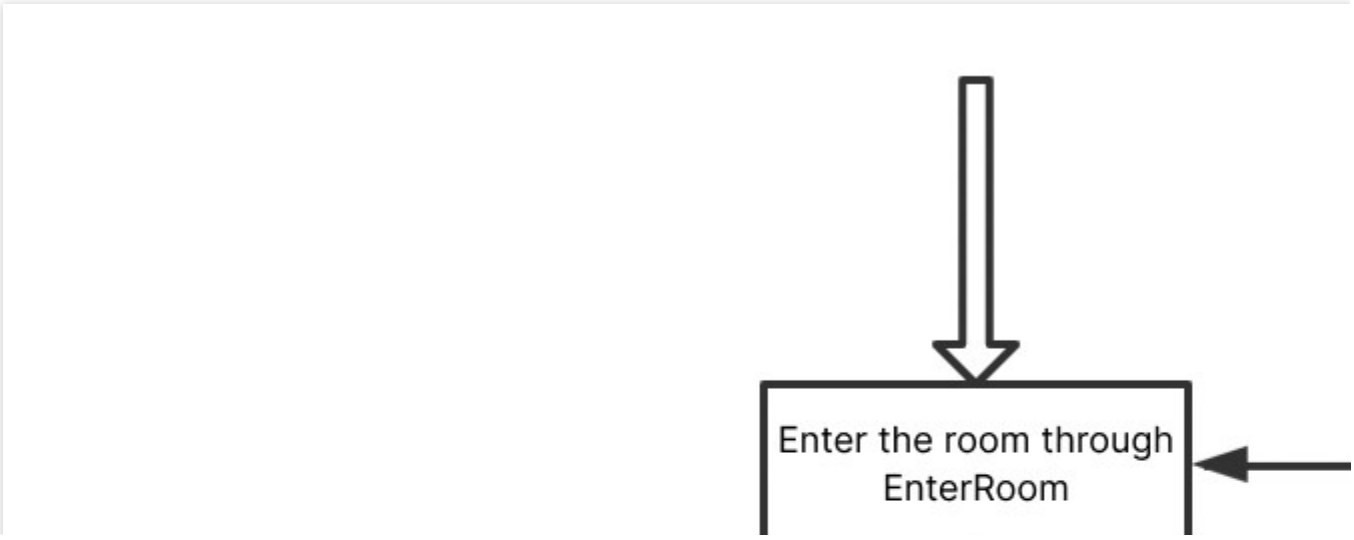
前提条件

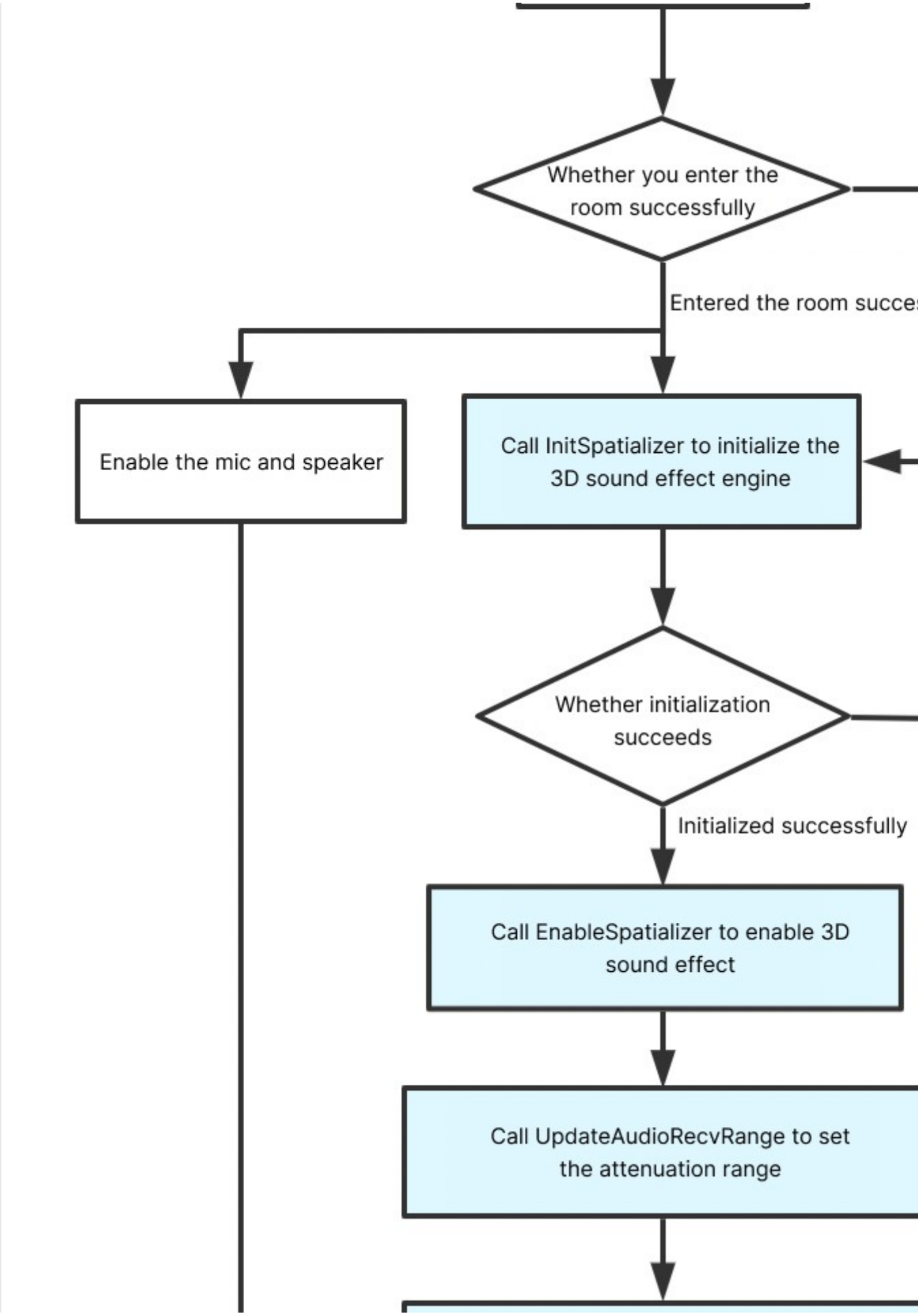
已开通实时语音服务：可参见 语音服务开通指引。
已接入 GME SDK：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入、Unity SDK 快速接入、Unreal SDK 快速接入。

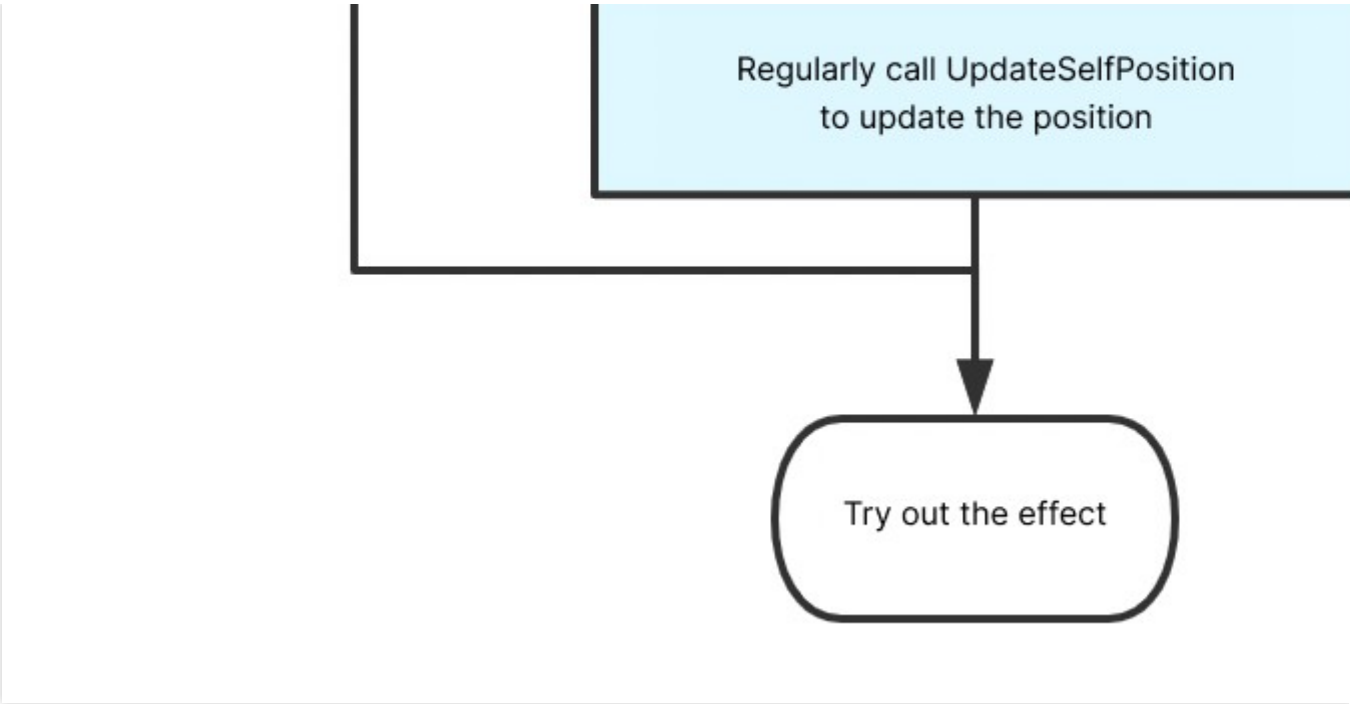
实现流程

实现流程图

下图为 3D 音效实现的流程图，其中蓝色部分为对比普通进房实时语音需要接入的步骤。



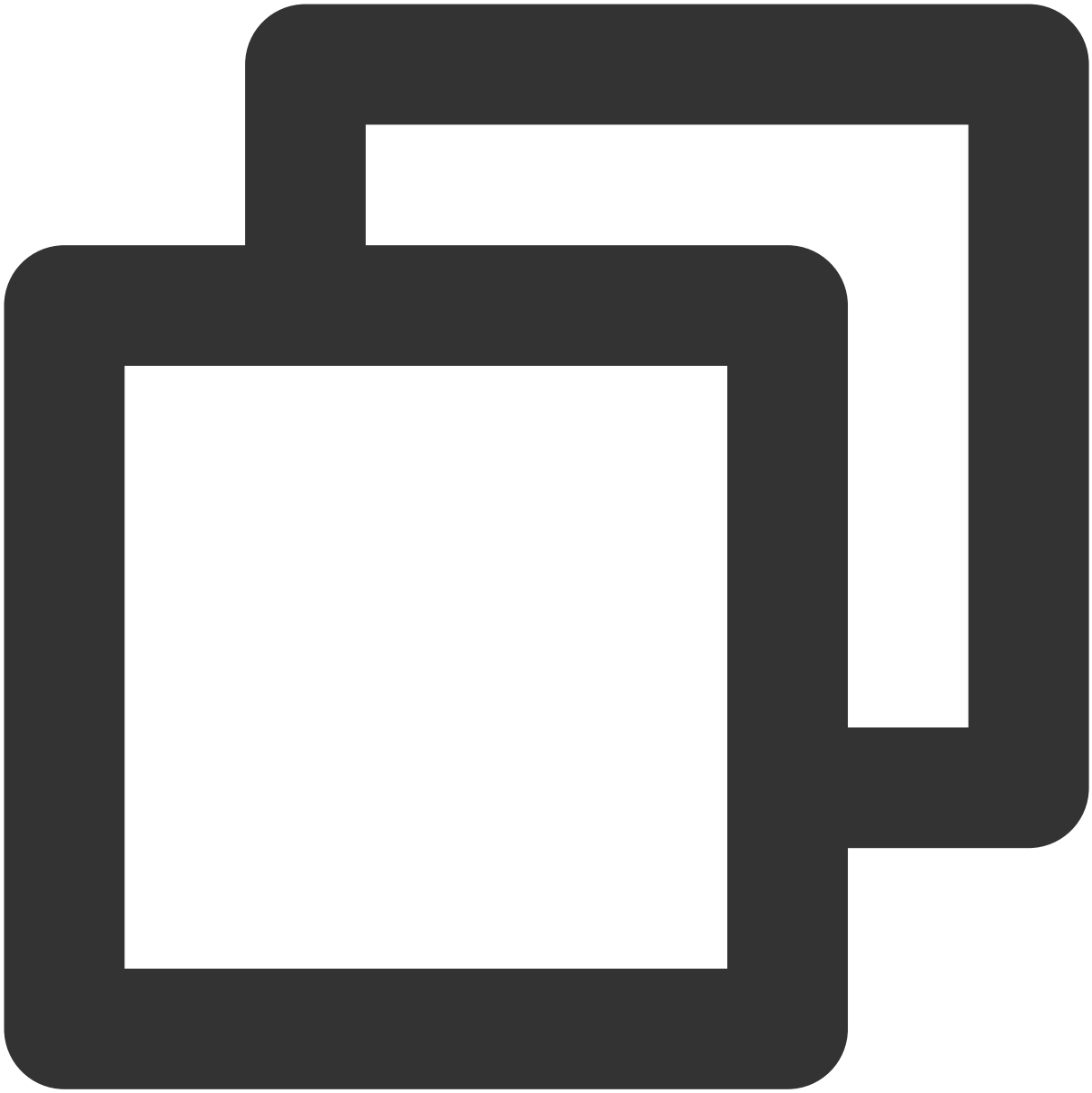

Whether you enter the room successfully
Entered the room succes
Enable the mic and speaker
Call InitSpatializer to initialize the 3D sound effect engine
Whether initialization succeeds
Initialized successfully
Call EnableSpatializer to enable 3D sound effect
Call UpdateAudioRecvRange to set the attenuation range



初始化 3D 音效引擎

此函数用于初始化 3D 音效引擎，在进房后调用。在使用 3D 音效之前必须先调用此接口，只接收 3D 音效而不发出 3D 音效的用户也需要调用此接口。

函数原型



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

参数	类型	意义
modelPath	string	3D 音效资源文件的绝对路径

参数中的 3D 音效资源文件，需要另外下载到本地，根据接入 SDK 的版本进行区分：
如果是v2.8以下版本，请单击 下载，md5: d0b76aa64c46598788c2f35f5a8a8694。
如果是v2.8-v2.9.5版本，请单击 下载，md5: 3d4d04b3949e267e34ca809e8a0b9243。

如果是v2.9.6及以上版本，由于3D音效资源文件已内置，此处 modelPath 可填空。
SDK 的版本发布历史请参见 产品动态。

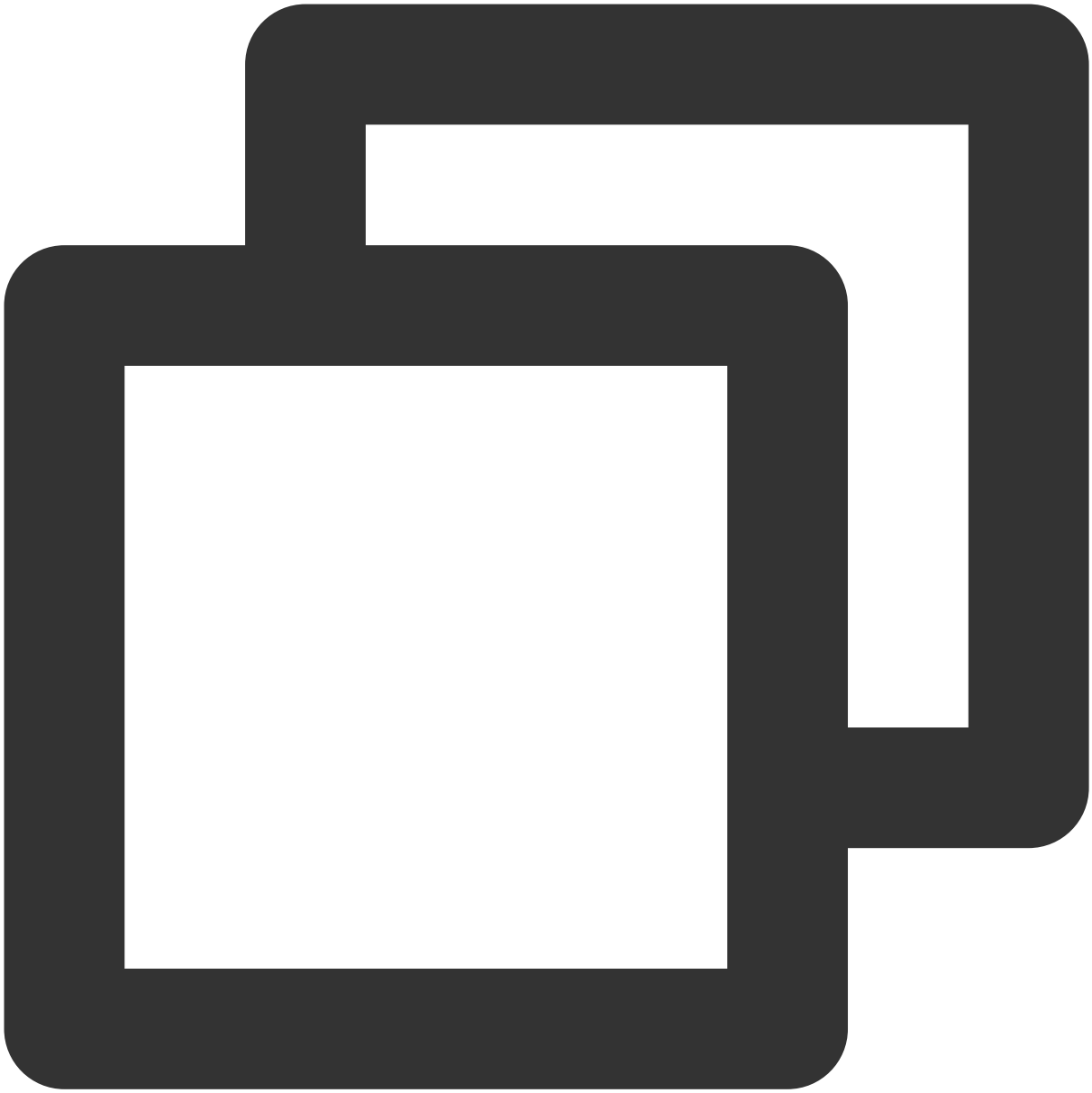
关于资源路径

以 Unity 为例，建议将 3D 文件放在项目的 StreamingAssets 目录下，并参考 SampleCode 中 **copyFileFromAssetsToPersistent** 函数写法，将资源文件拷贝到不同平台的相应目录中。
以 Unreal 为例，参考 SampleCode 中 **CopyAllAssetsToExternal** 函数写法，将 3D 模型文件拷贝后再读取路径。

开启或关闭 3D 音效

此函数用于开启或关闭 3D 音效。开启之后可以听到 3D 音效。

函数原型



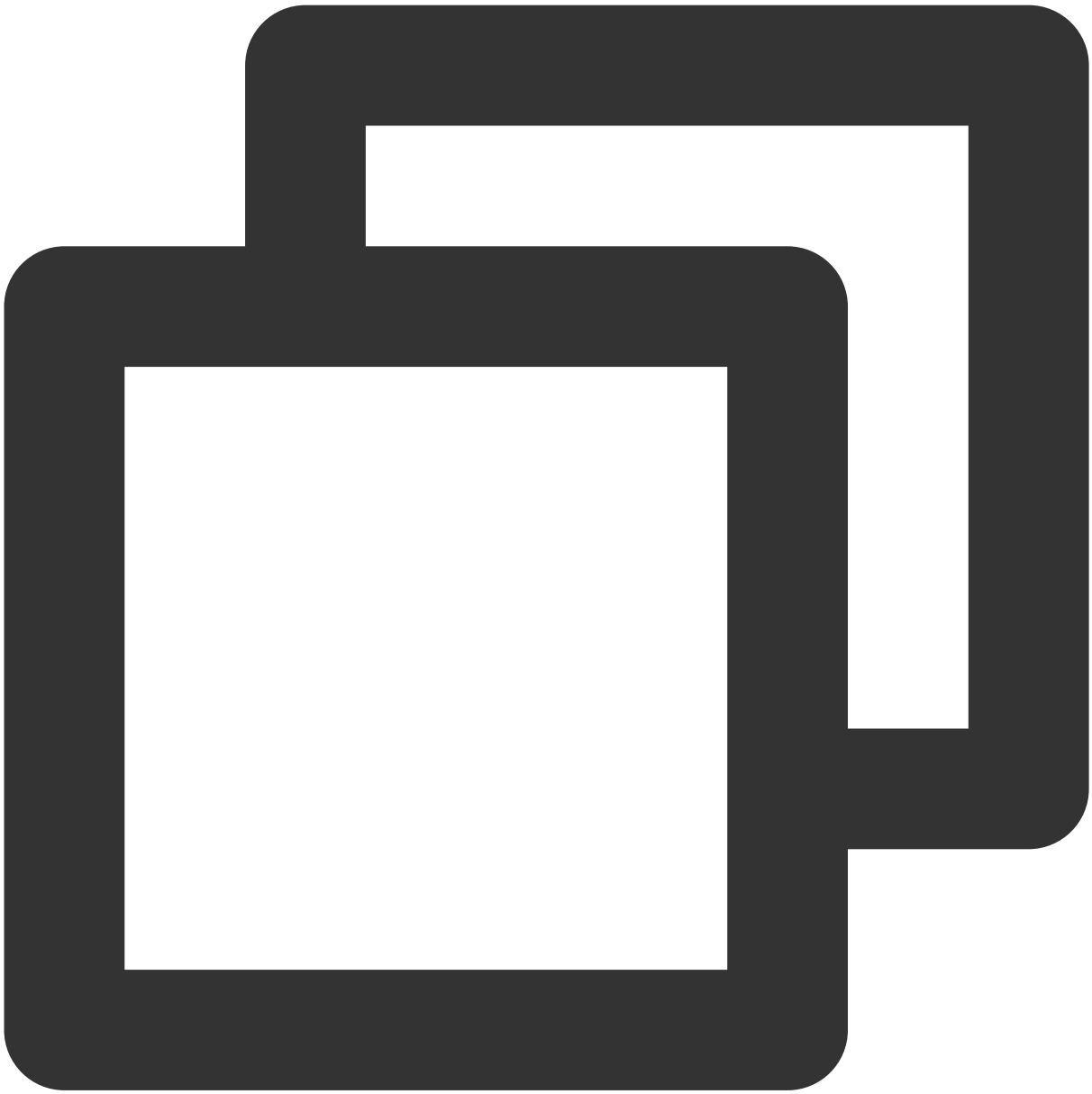
```
public abstract int EnableSpatializer(bool enable, bool applyToTeam)
```

参数	类型	意义
enable	bool	开启之后可以听到 3D 音效
applyToTeam	bool	3D 语音是否作用于小队内部，仅 enable 为 true 时有效

获取当前 3D 音效状态

此函数用于获取当前 3D 音效状态。

函数原型



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

返回值	意义
true	开启状态
false	关闭状态

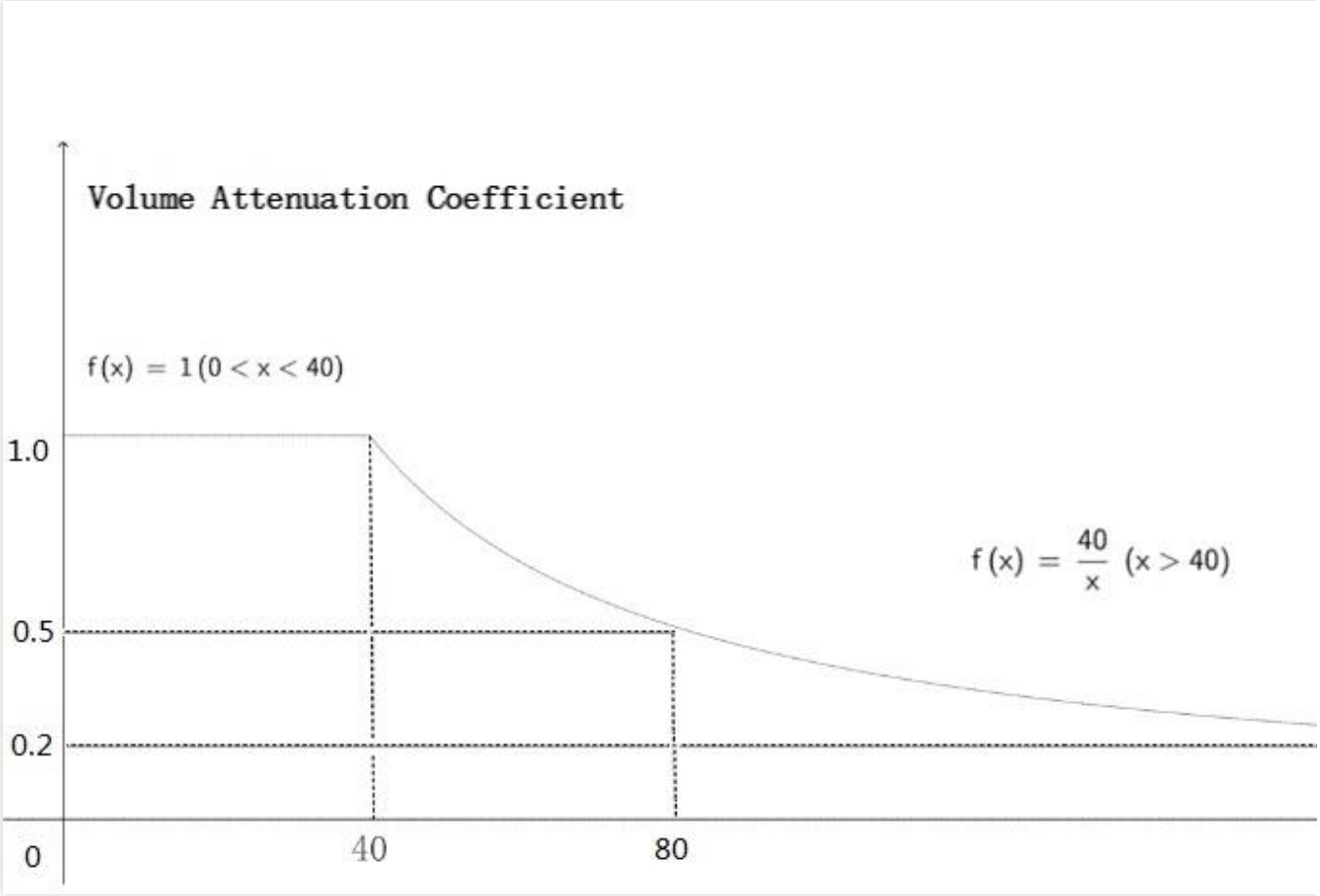
设置 3D 音效衰减距离

需要设置衰减距离，建议设置为 100。

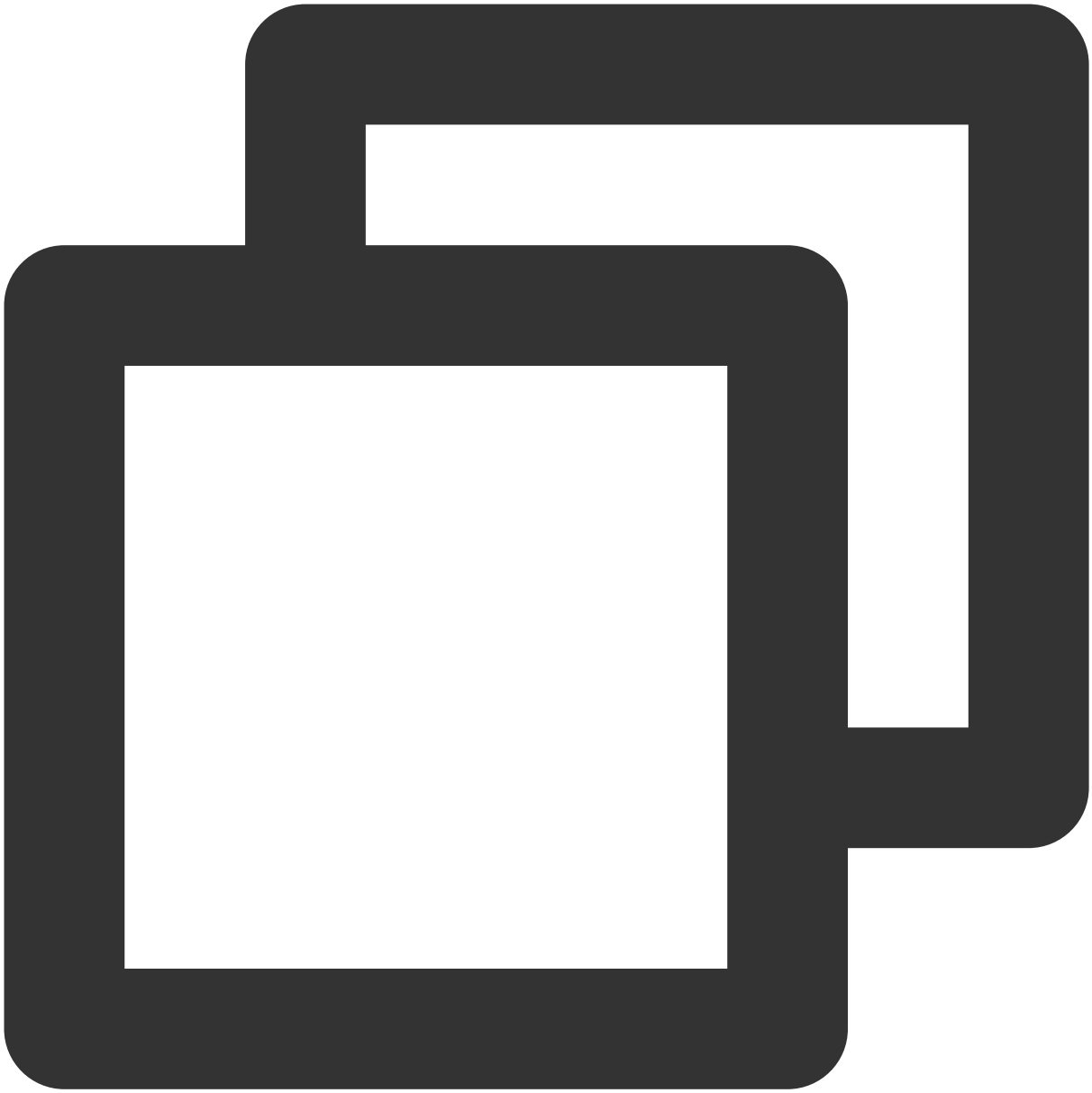
距离与声音衰减的关系

3D 音效中，音源音量的大小与音源距离有一定的衰减关系。单位距离超过 range 之后，音量衰减到几乎为零。

距离范围（引擎单位）	衰减公式
0 < N < range/10	衰减系数：1.0 （音量无衰减）
N ≥ range/10	衰减系数：range/10/N



函数原型



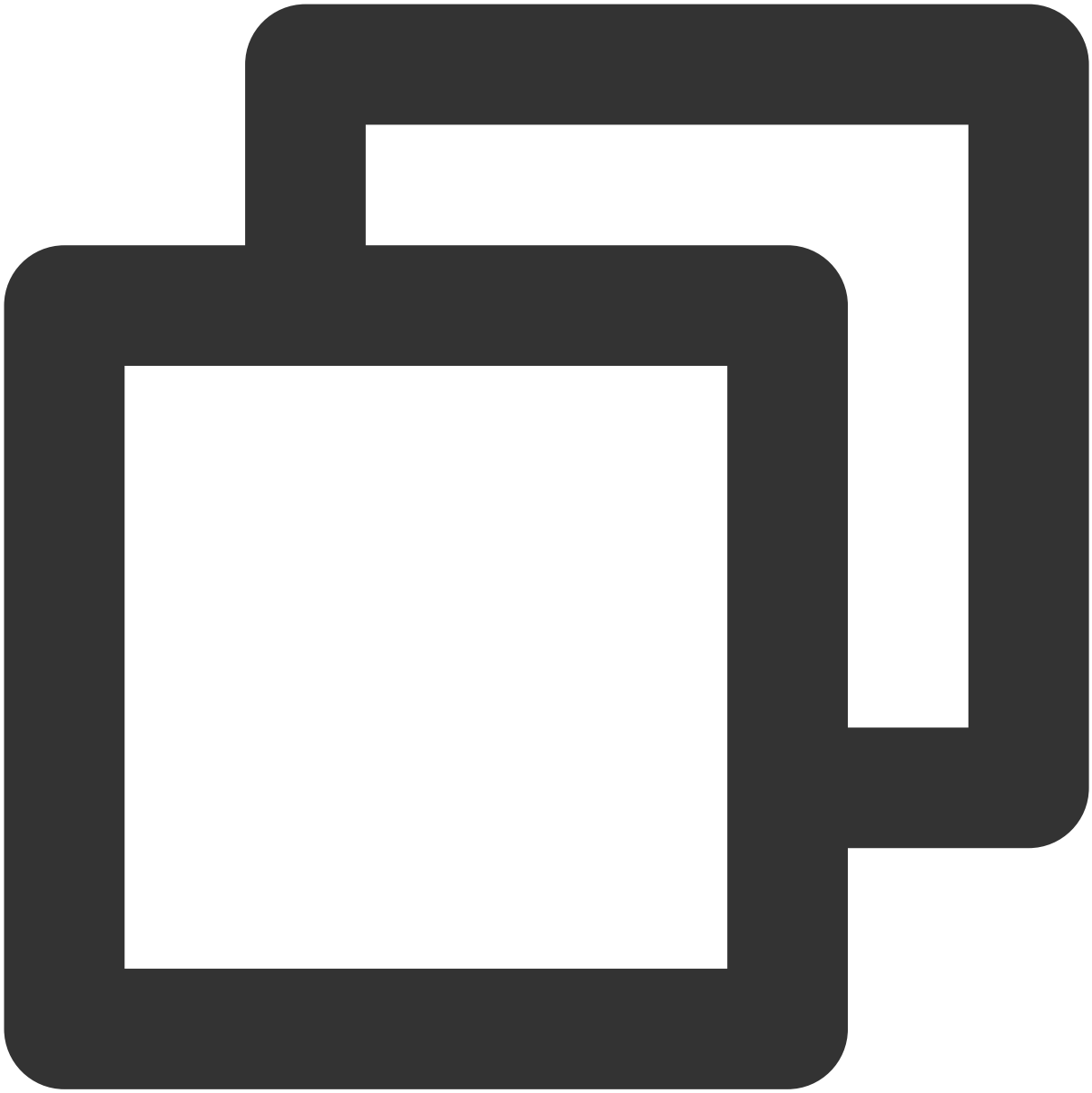
```
public abstract void UpdateAudioRecvRange(int range)
```

参数	类型	意义
range	int	设定音效可接收的范围，建议设置100，此处的距离单位为游戏引擎中的距离单位

更新声源方位（包含朝向）

此函数用于更新声源方位角信息，每帧调用便可实现 3D 音效效果。

函数原型



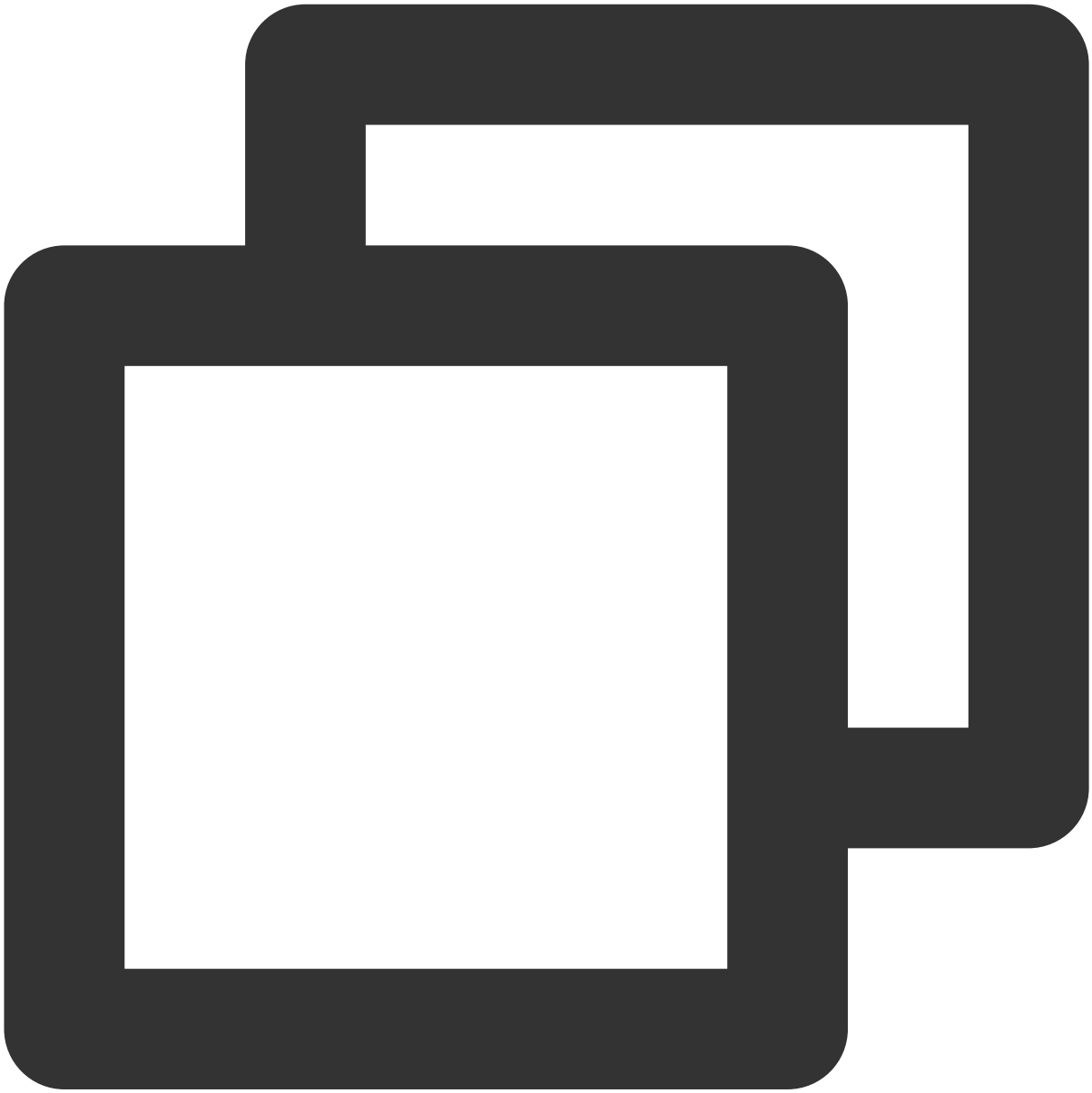
```
public abstract int UpdateSelfPosition(int position[3], float axisForward[3], float
```

参数	类型	意义
position	int[]	自身在世界坐标系中的坐标，顺序是前、右、上
axisForward	float[]	自身坐标系前轴的单位向量
axisRight	float[]	自身坐标系右轴的单位向量

axisUp	float[]	自身坐标系上轴的单位向量

示例代码

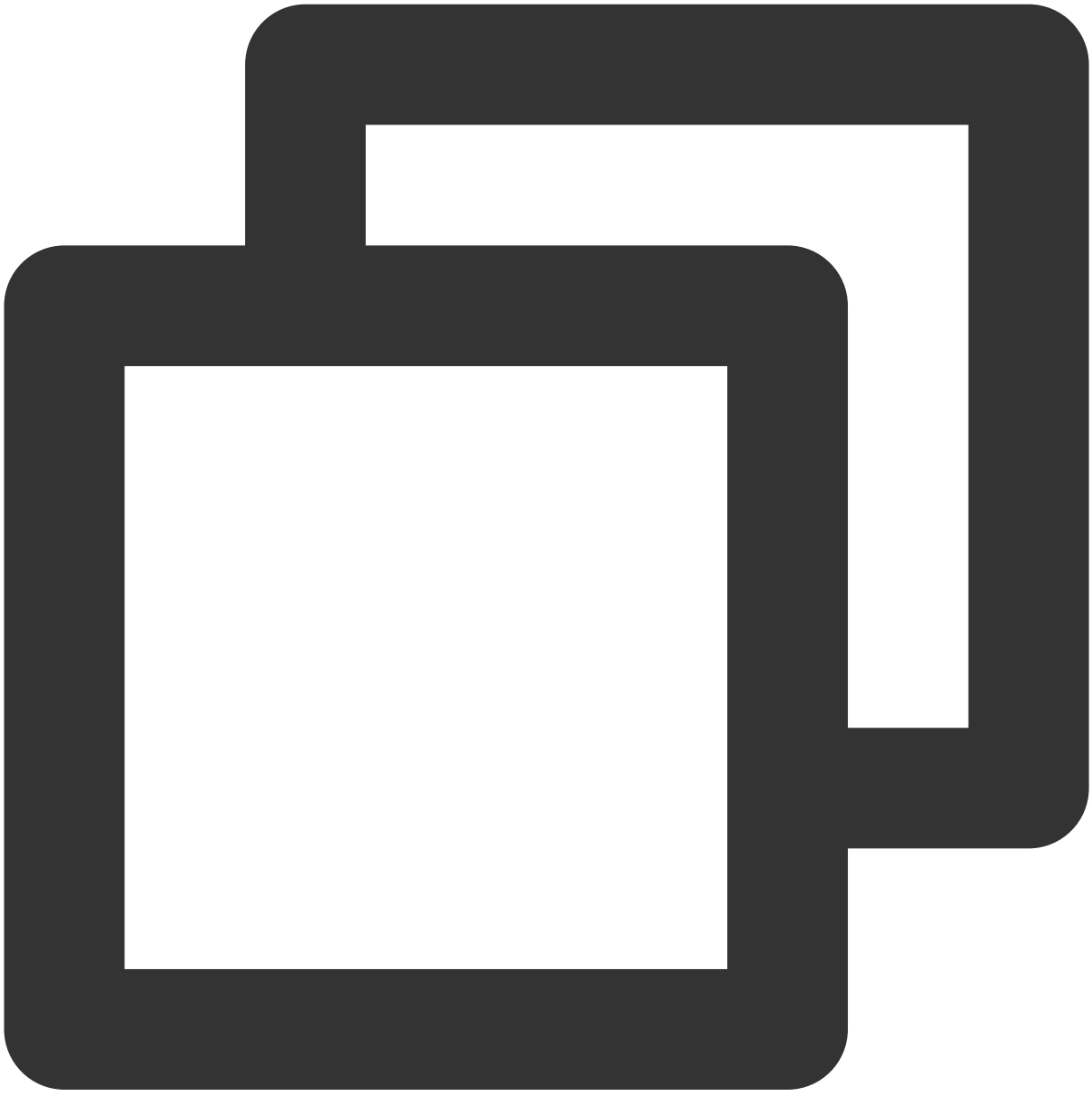
Unreal



```
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
```

本地方位接口（VR 场景）

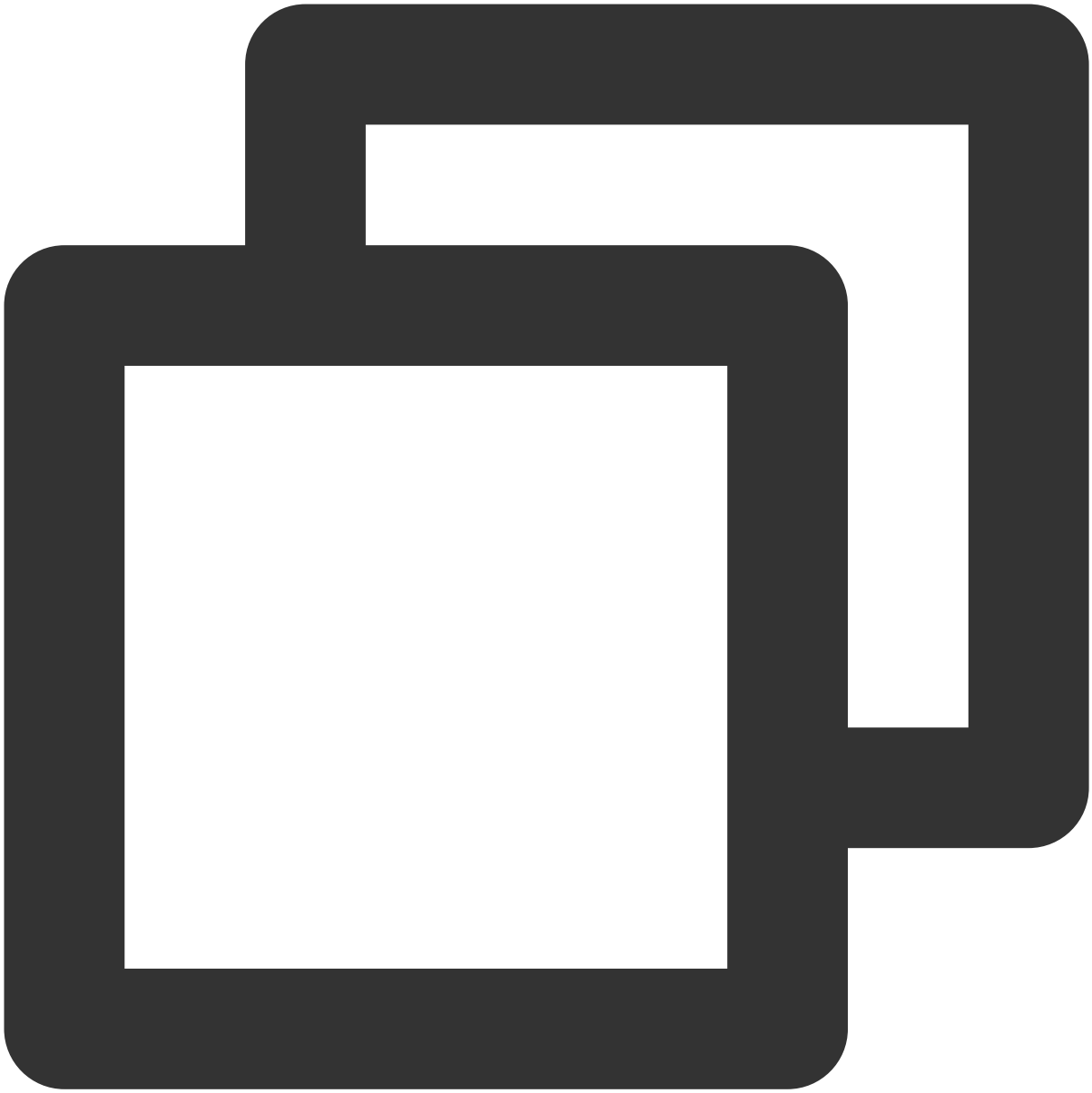
本接口适用于 VR 设备中，对 3D 位置变化有极高要求的场景。此功能为高级接口，需要在 GME 2.9.2 以上版本才可使用。

在一般 3D 的场景中，用户只需要通过函数 UpdateSelfPosition 更新自己的位置信息，然后经网络发送给其他用户。

UpdateOtherPosition支持本地传入其他玩家的位置信息，不经过网络发送，适用于 VR 游戏场景。

为了避免和远程更新的坐标发生冲突，只要您调用了UpdateOtherPosition，远程坐标便会被丢弃，这种影响持续到再一次进房。所以，**您如果要通过本地方式更新玩家坐标，请更新所有玩家的坐标**。

函数原型



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

参数	类型	意义
position	int[]	其他玩家在世界坐标系中的坐标，顺序是前、右、上

注意

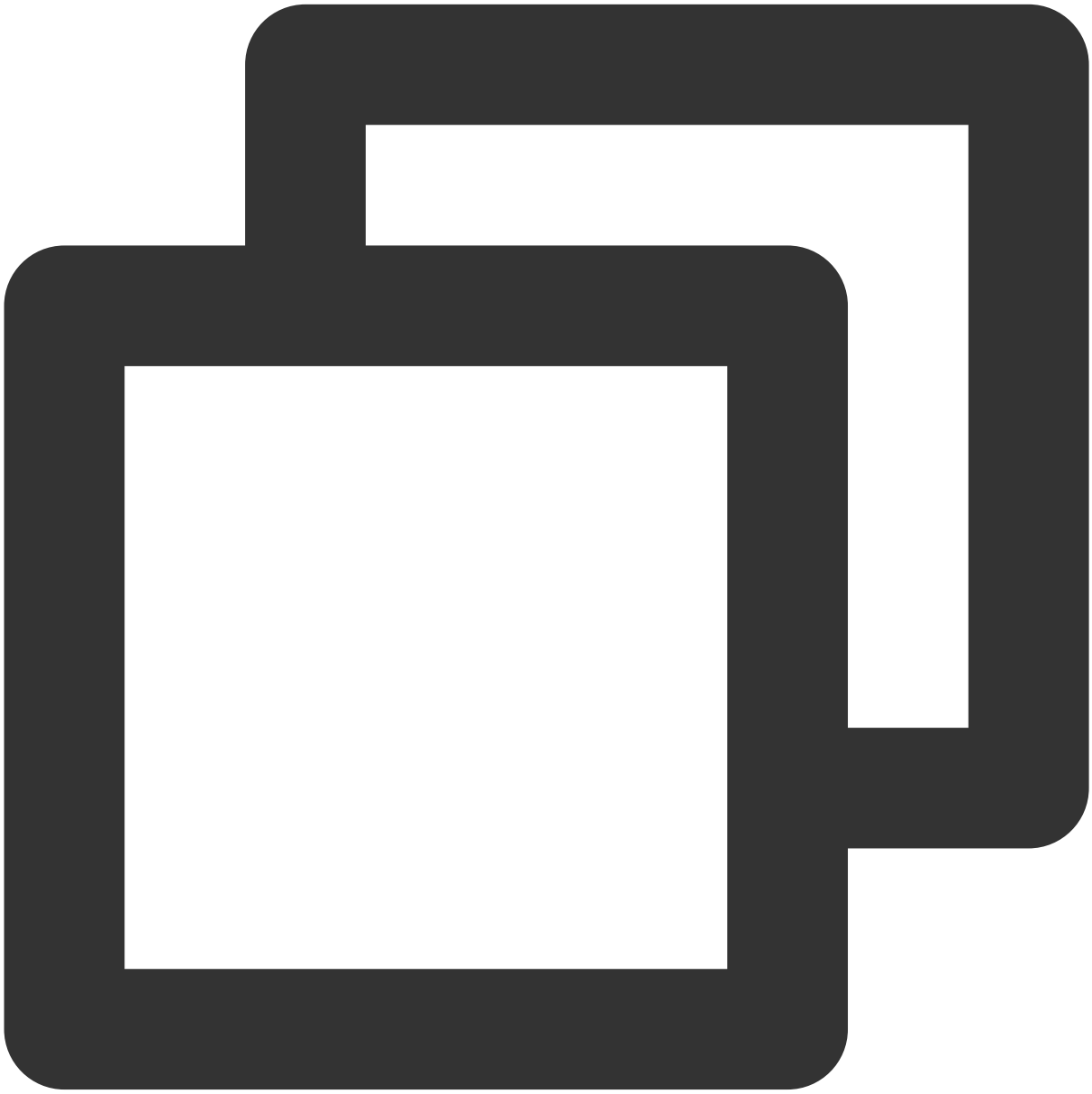
您如果要通过本地方式更新玩家坐标，请遍历**所有的玩家坐标**，通过此接口将坐标进行传入。

3D 语音黑名单接口

注意

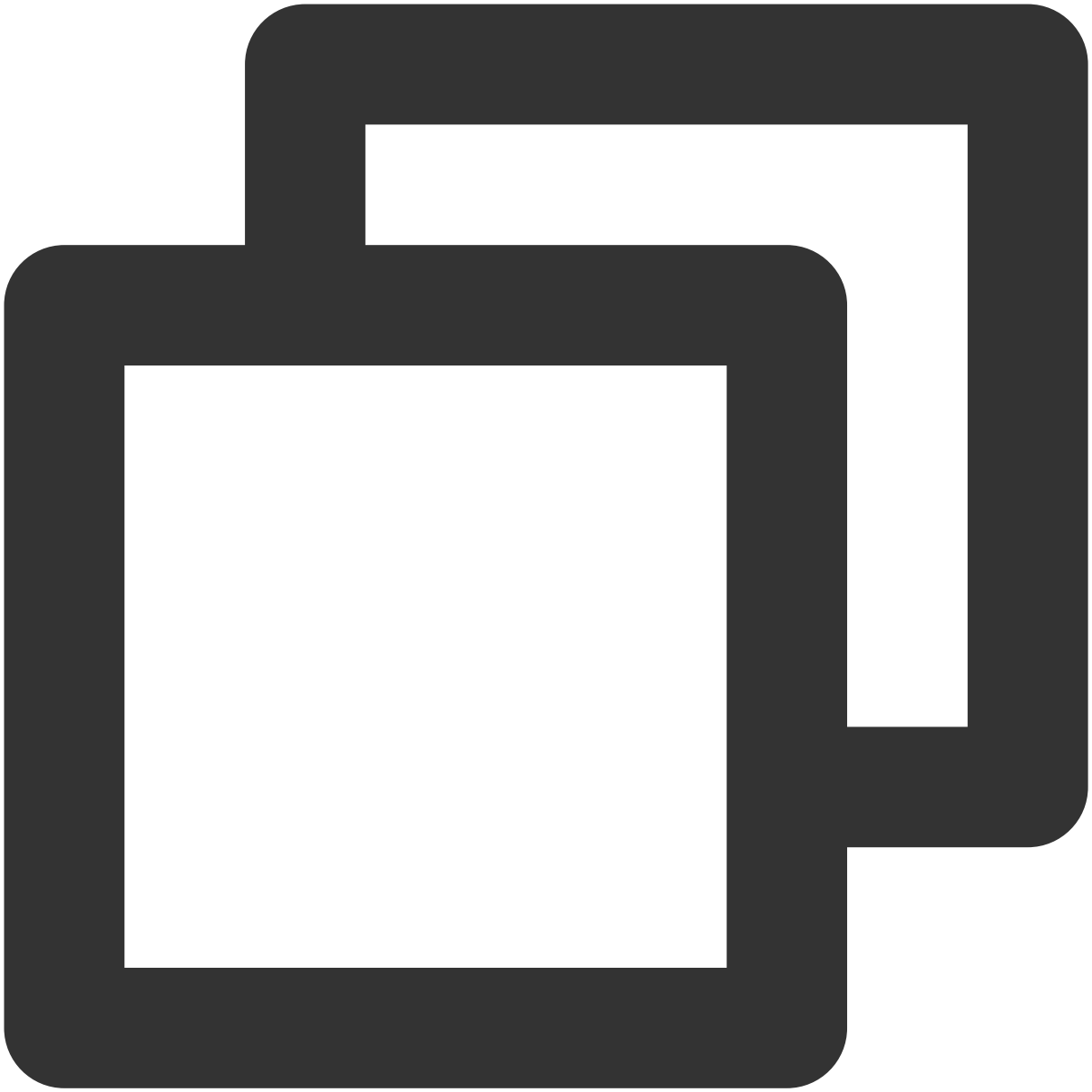
此接口在 GME 2.9.3 及以上版本 SDK 生效。

目前 3D 音效调用后对房间内所有人都会生效。由于某些特定场景下，不希望接受到的某个人声音会有因 3D 音效而产生的衰减，可以通过调用此接口，将某个人加入 3D 语音黑名单中，加入后此 openid 的声音不再有 3D 音效效果。



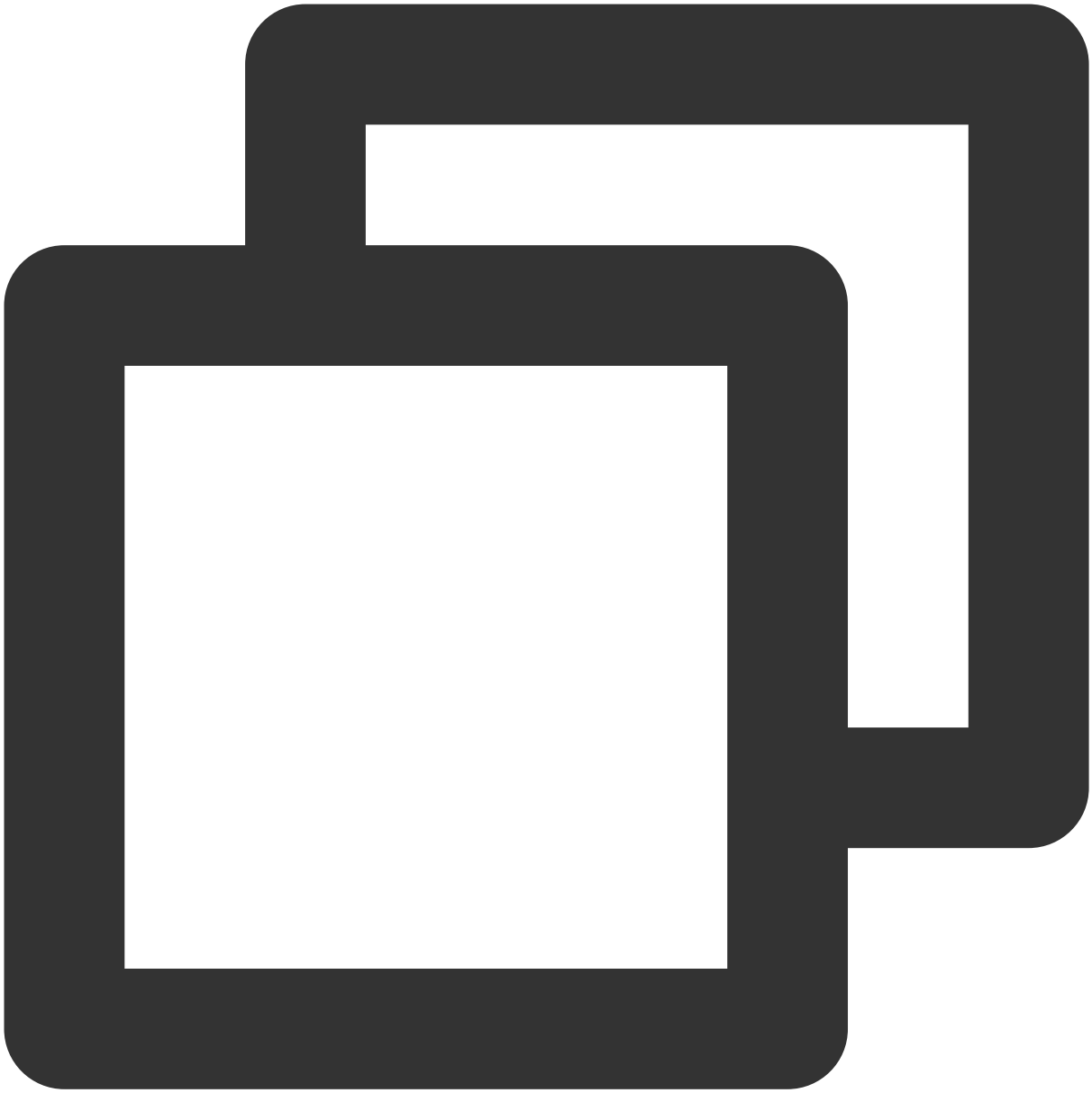
```
virtual int AddSpatializerBlacklist(const char* openId);
```

如果需要将此 openid 从黑名单中移除，需要调用以下接口：



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

如果需要清空黑名单，需要调用以下接口：



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

问题排查

如果接入后测试语音，没有 3D 音效效果，排查路径如下：

1. 确定是否已经进房成功及开启麦克风？双方能否听到声音？
2. 是否适用双声道耳机进行收听？

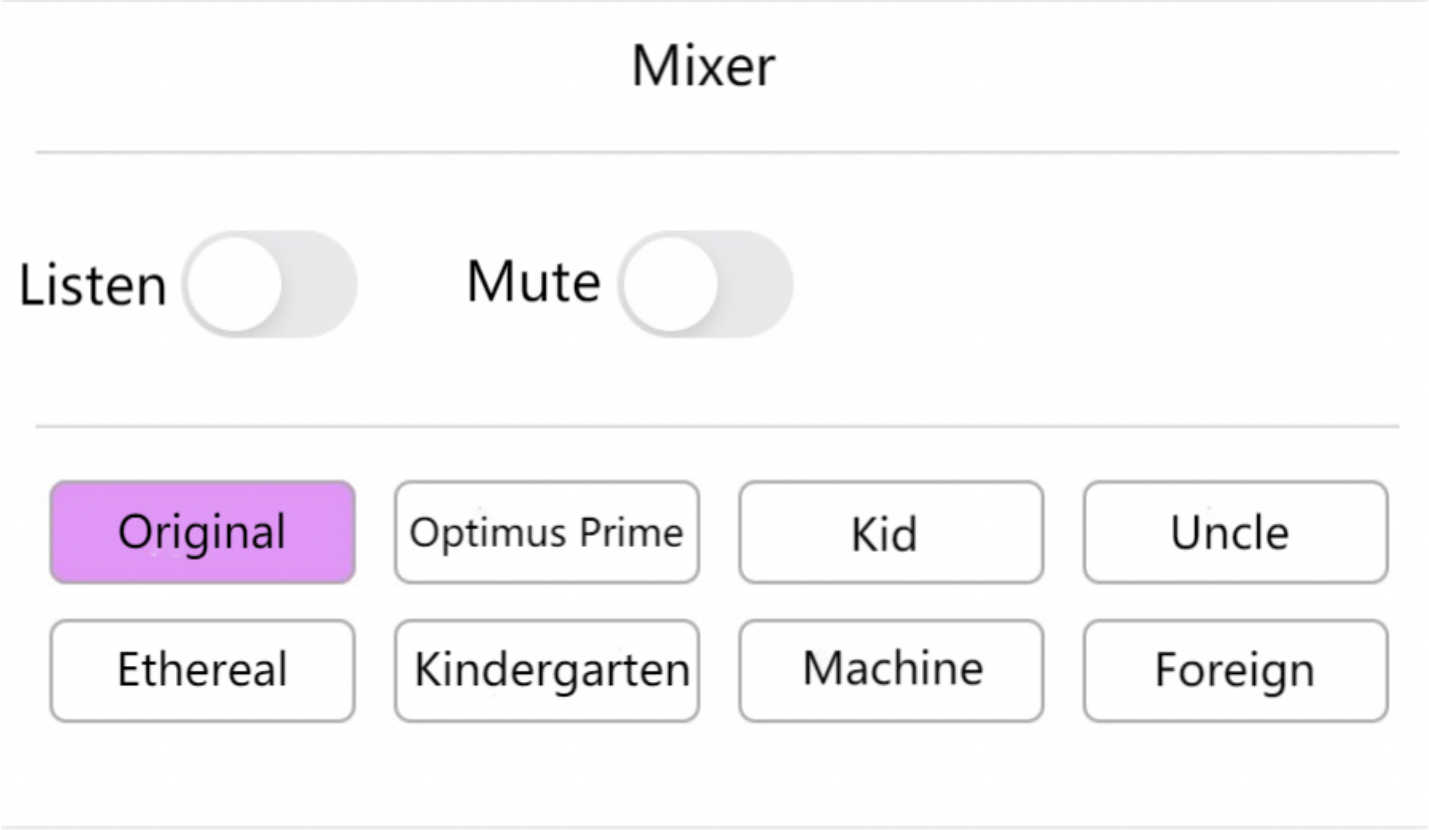
3. InitSpatializer 接口返回值是否为0？
4. UpdateAudioRecvRange 设置是否过小？
5. 是否有周期性的调用 UpdateSelfPosition 接口？
6. 通过 错误码文档 进行判断并解决。

音效与伴奏
语音变声

最近更新时间：2023-02-09 15:51:26

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文向您介绍游戏多媒体引擎变声特效的接入方法。

使用场景



前提条件

- **已开通实时语音服务**：可参见 服务开通指引。
- **已开通语音转文本服务**：可参见 服务开通指引。
- **已接入 GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入、Unity SDK 快速接入、Unreal SDK 快速接入。

- **已接入 GME SDK 库文件libgmesoundtouch**：需要保证工程库文件中带有 libgmesoundtouch，具体参见 库文件对应功能。

实时语音变声接入

变声接口

在已经进房成功且打开麦克风的情况下，调用 SetVoiceType 接口设置变声特效，接口返回0代表调用成功，房间内的人听到的本端发出的声音带有变声效果。如果要自测变声效果，请使用耳返功能（接口：EnableLoopBack）。

函数原型

- Android java
- iOS objectc
- Unity C#
- C++ 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);
```

参数	类型	意义
type	int	表示本端音频变声类型

类型参数	参数代表	意义
ITMG_VOICE_TYPE_ORIGINAL_SOUND	0	原声

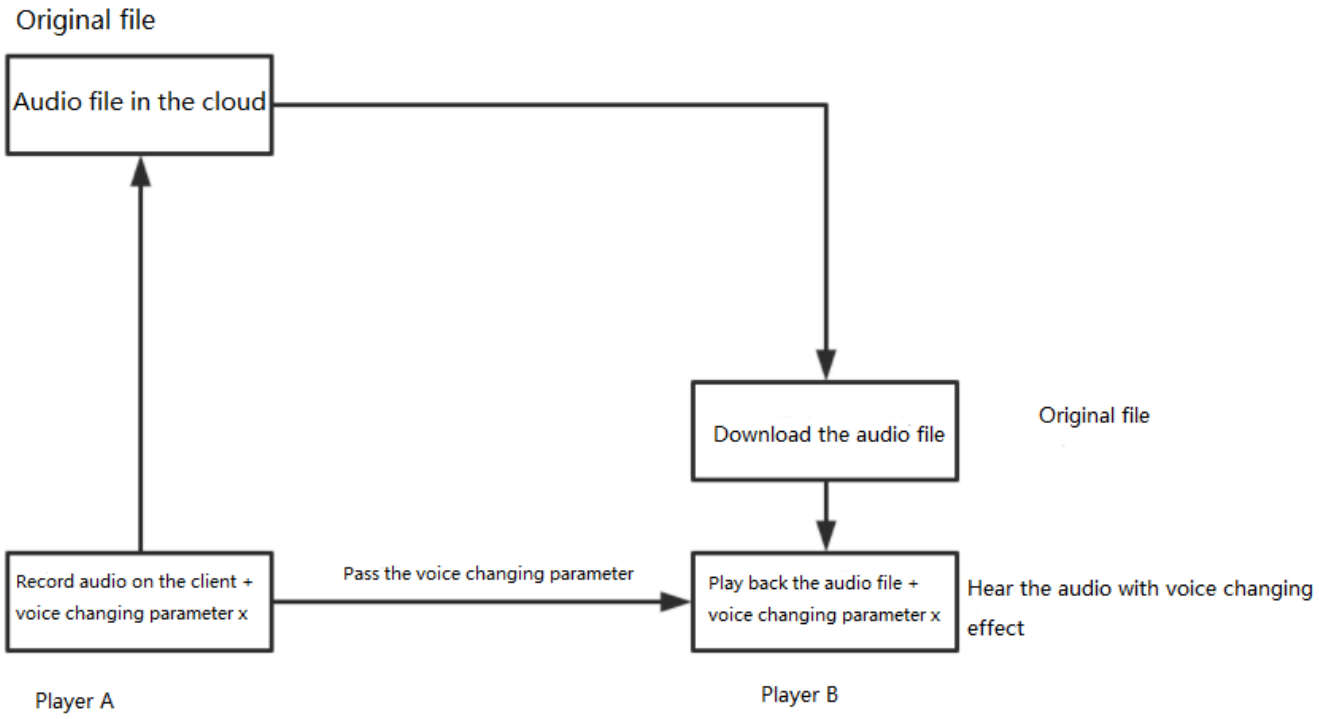
类型参数	参数代表	意义
ITMG_VOICE_TYPE_LOLITA	1	萝莉
ITMG_VOICE_TYPE_UNCLE	2	大叔
ITMG_VOICE_TYPE_INTANGIBLE	3	空灵
ITMG_VOICE_TYPE_DEAD_FATBOY	4	小胖子
ITMG_VOICE_TYPE_HEAVY_MENTA	5	重金属
ITMG_VOICE_TYPE_DIALECT	6	歪果仁
ITMG_VOICE_TYPE_INFLUENZA	7	感冒
ITMG_VOICE_TYPE_CAGED_ANIMAL	8	困兽
ITMG_VOICE_TYPE_HEAVY_MACHINE	9	重机器
ITMG_VOICE_TYPE_STRONG_CURRENT	10	强电流
ITMG_VOICE_TYPE_KINDER_GARTEN	11	幼稚园
ITMG_VOICE_TYPE_HUANG	12	小顽童

示例代码

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

语音消息变声接入

语音消息变声流程



语音消息变声不会影响原始的音频信息，在播放的时候才体现变声效果。

语音消息播放

语音消息播放接口，带有变声效果参数。

- Android java
- iOS objectc
- Unity c#
- C++ c++

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

参数	类型	含义
filePath	string	本地语音文件的路径
voicetype	int	变声类型

错误码

错误码值	原因	建议方案
20485	播放未开始	确保文件存在，文件路径的合法性

实时语音伴奏

最近更新时间：2024-01-18 14:26:28

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文为您介绍游戏多媒体引擎实时语音伴奏的接入技术文档。

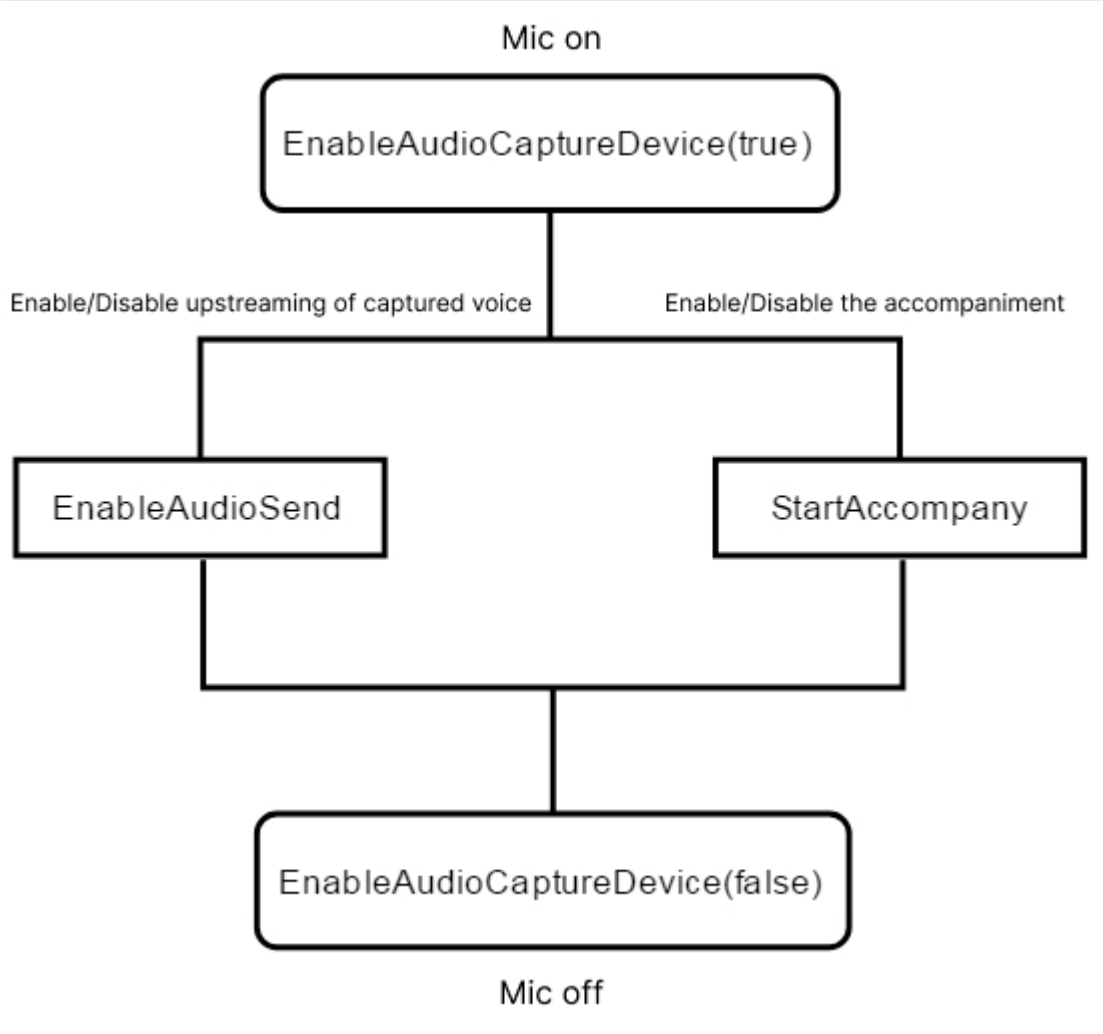
实时语音伴奏相关接口

接口	接口含义
StartAccompany	开始播放伴奏。
StopAccompany	停止播放伴奏。
IsAccompanyPlayEnd	伴奏是否播放完毕。
PauseAccompany	暂停播放伴奏。
ResumeAccompany	重新播放伴奏。
SetAccompanyVolume	设置伴奏音量。
GetAccompanyVolume	获取播放伴奏的音量。
SetAccompanyFileCurrentPlayedTimeByMs	设置播放进度。

说明：

如需使用实时语音伴奏，需要在接入 GME SDK 且能在进行实时语音通话的情况下，才可以使用实时语音伴奏。

流程图



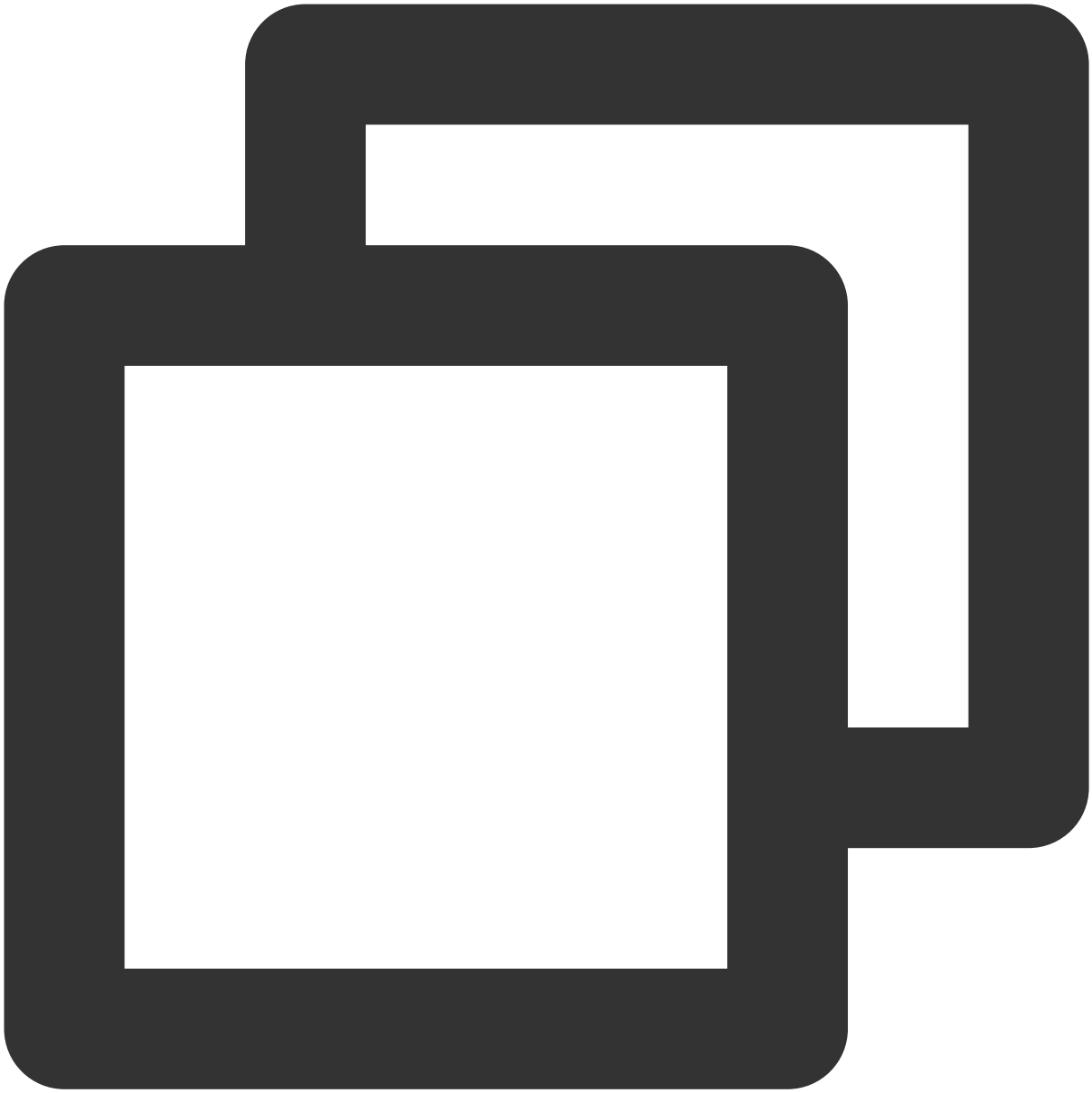
如何配合 EnableAudioCaputreDevice 使用

在进入实时语音房间成功之后，调用 EnableAudioCaputreDevice 打开采集设备，再调用 StartAccompany 播放伴奏。如果需要采集人声，可以调用 EnableAudioSend 实现开麦效果。

开始播放伴奏

调用 StartAccompany 接口开始播放伴奏。支持 m4a、wav、mp3 一共三种格式。调用此 API，音量会重置。

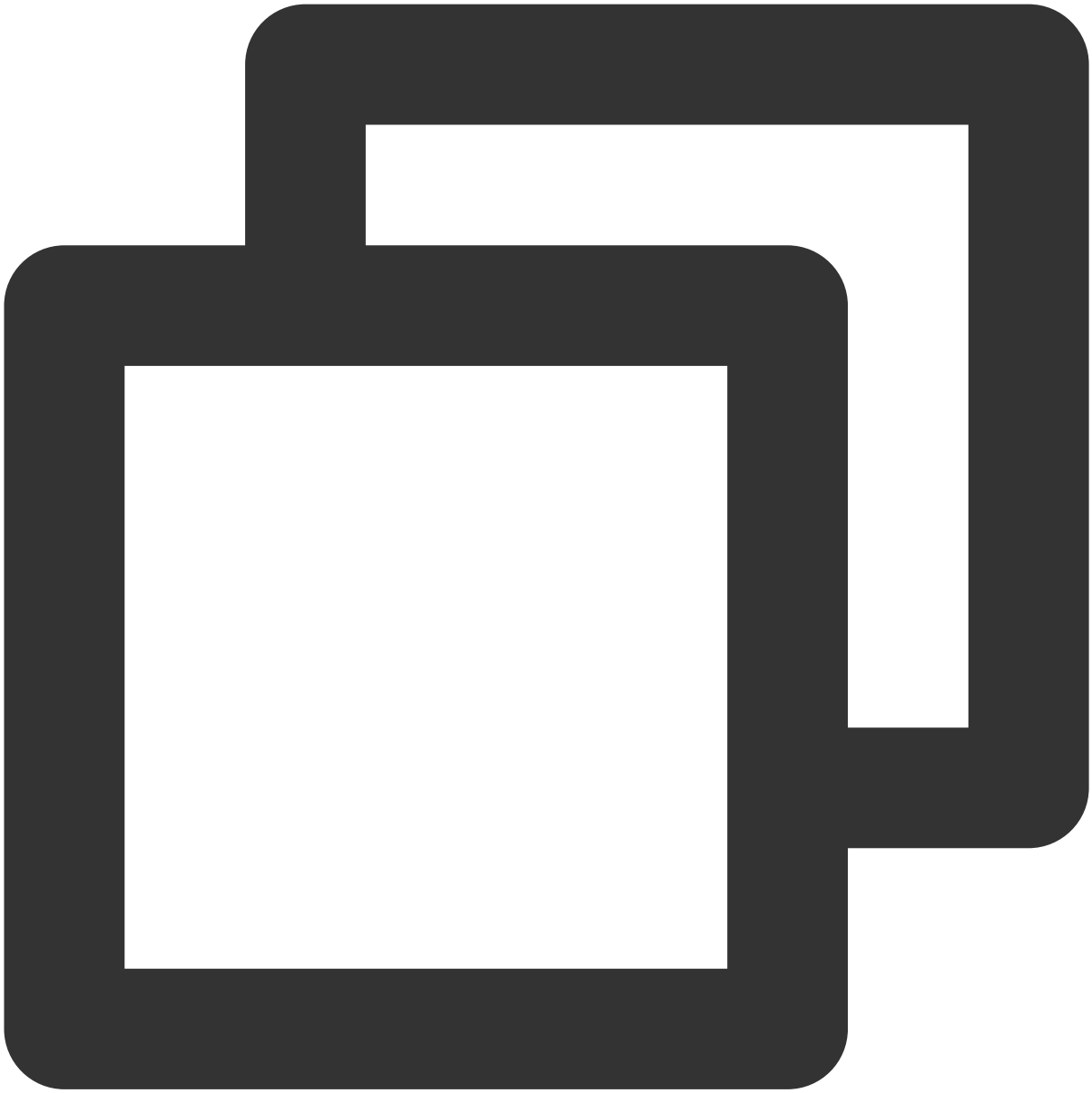
函数原型



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

参数	类型	意义
filePath	char*	播放伴奏的路径。
loopBack	bool	是否混音发送，一般都设置为 true，即其他人也能听到伴奏。
loopCount	int	循环次数，数值为-1表示无限循环。填0不播放。
msTime	int	延迟时间。

示例代码

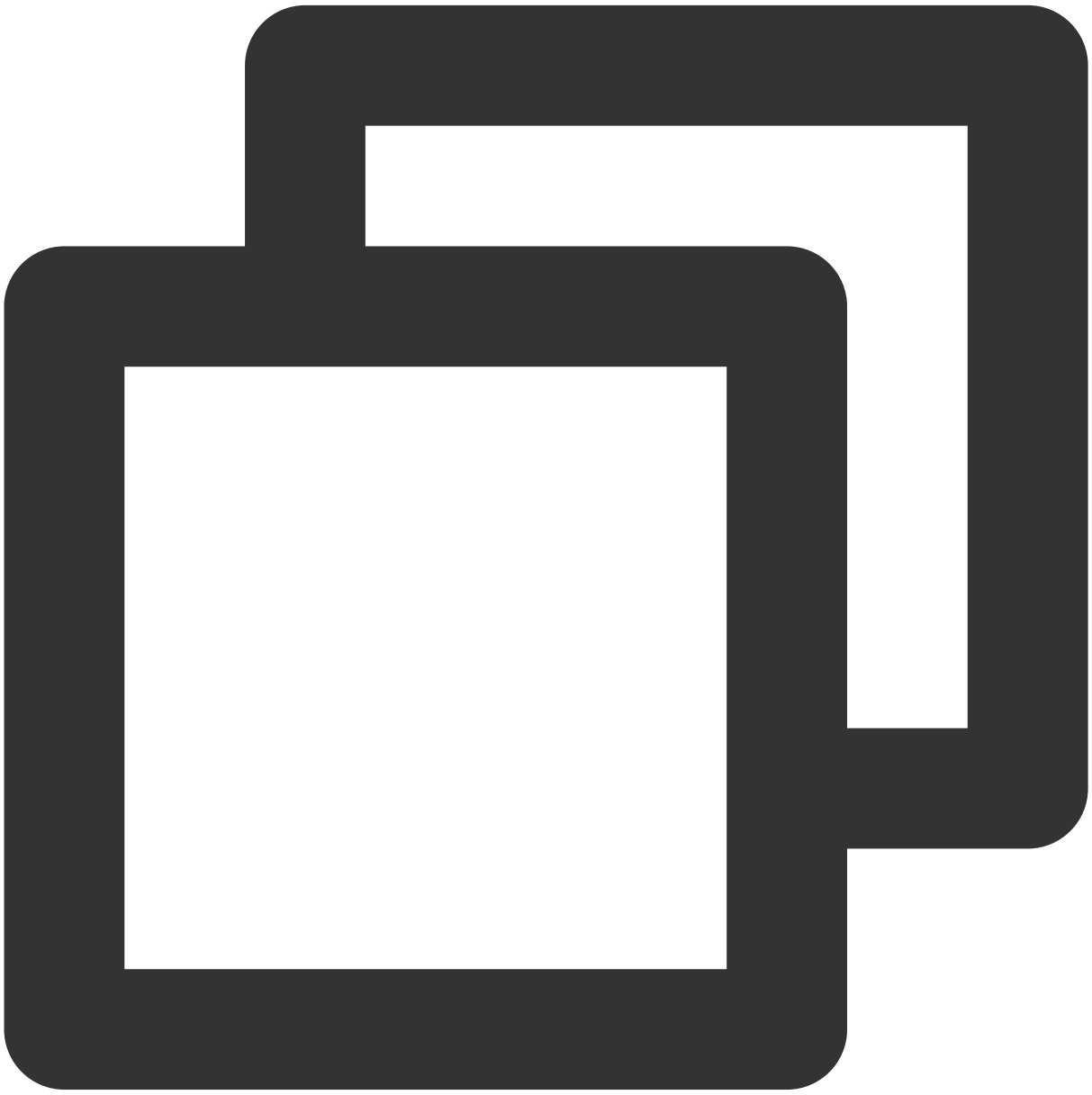


```
//Windows端代码
ITMGContextGetInstance()->GetAudioEffectCtrl()->StartAccompany(filePath,true,-1,0);
//Android端代码
ITMGContext.GetInstance(this).GetAudioEffectCtrl().StartAccompany(filePath,true,loo
//iOS端代码
[[[ITMGContext GetInstance] GetAudioEffectCtrl] StartAccompany:path loopBack:isLoop
```

开始播放伴奏（边下边播）

调用 StartAccompanyDownloading 接口开始边下载边播放伴奏。在代码中实现下载伴奏，未下载完时，可以先将文件路径作为参数传到 StartAccompanyDownloading 里面，可实现边下边播。fileSize 为预估的完整文件大小。调用此接口传入未下载完的文件时，先保证文件至少有10k以上。

函数原型



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

说明：

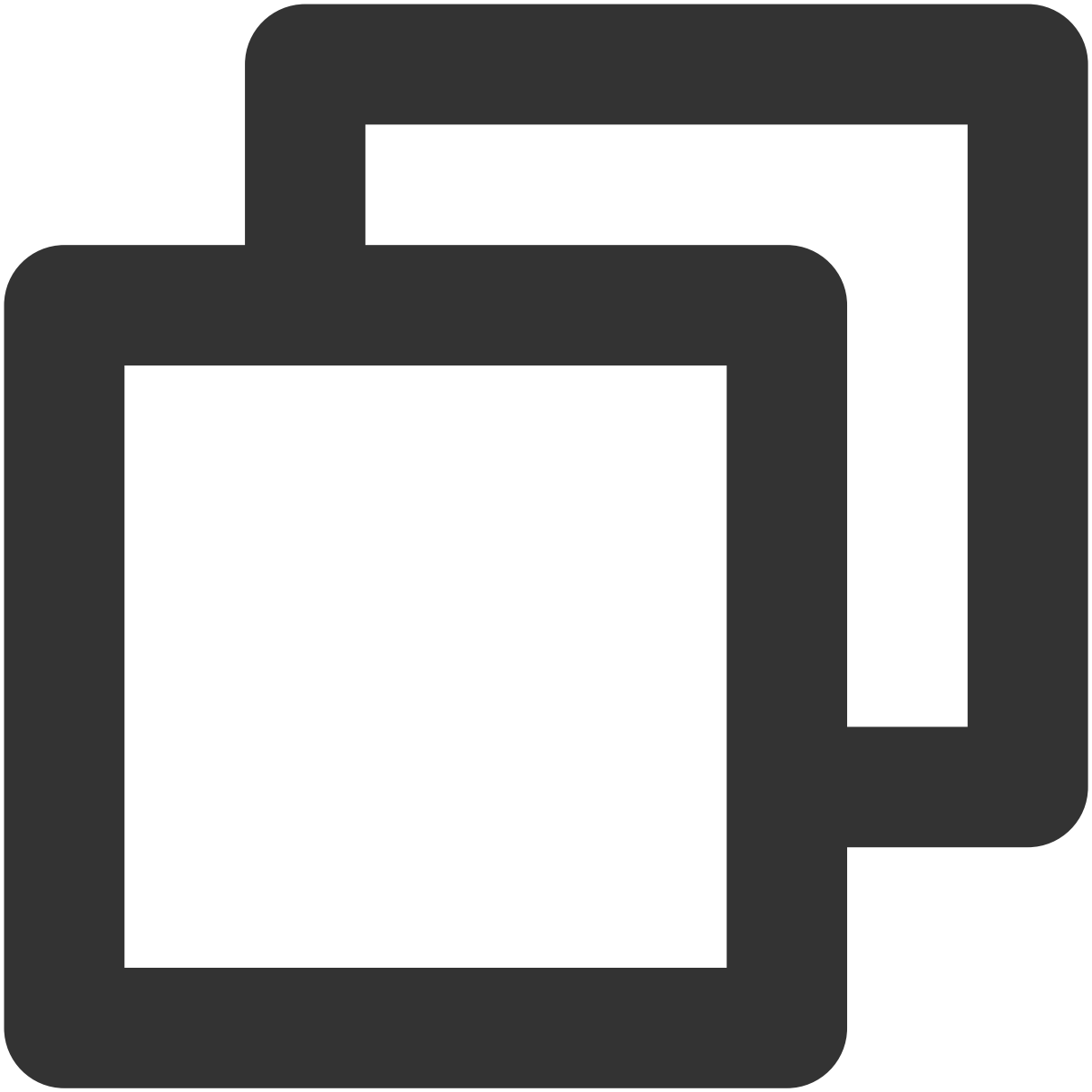
iOS 端需要以下配置。

1. 在 iOS 端使用此功能，需要将相关动态库引入工程中，单击下载 mp3 动态库。
2. 将下载好的文件引入到工程文件中。并在 Link Binary With Libraries 中添加此动态库。
3. 将头文件 TMGEngine_adv.h 加入工程中，与其他 SDK 头文件同目录下。

播放伴奏的回调

开始播放伴奏完成后，回调函数调用 OnEvent，事件消息为 ITMG_MAIN_EVENT_TYPE_ACCOMPANY_FINISH，在 OnEvent 函数中对事件消息进行判断。
传递的参数 data 包含两个信息，一个是 result，另一个是 file_path。

示例代码



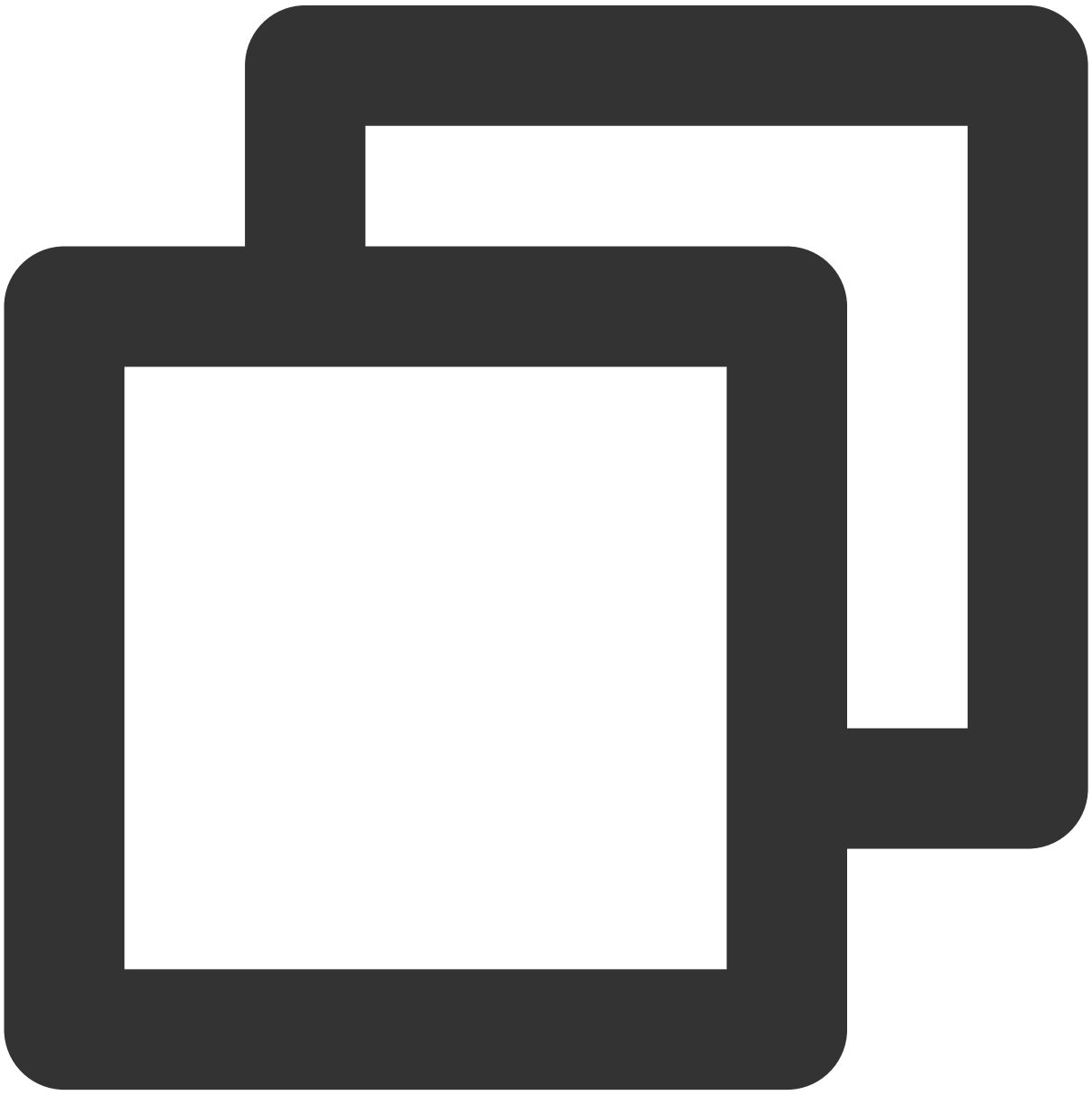
```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
        {
        //进行处理
        break;
        }
        ...
        case ITMG_MAIN_EVENT_TYPE_ACCOMPANY_FINISH:
        {
        //进行处理
```

```
            break;
            }
        }
    }
```

停止播放伴奏

调用 StopAccompany 接口停止播放伴奏。

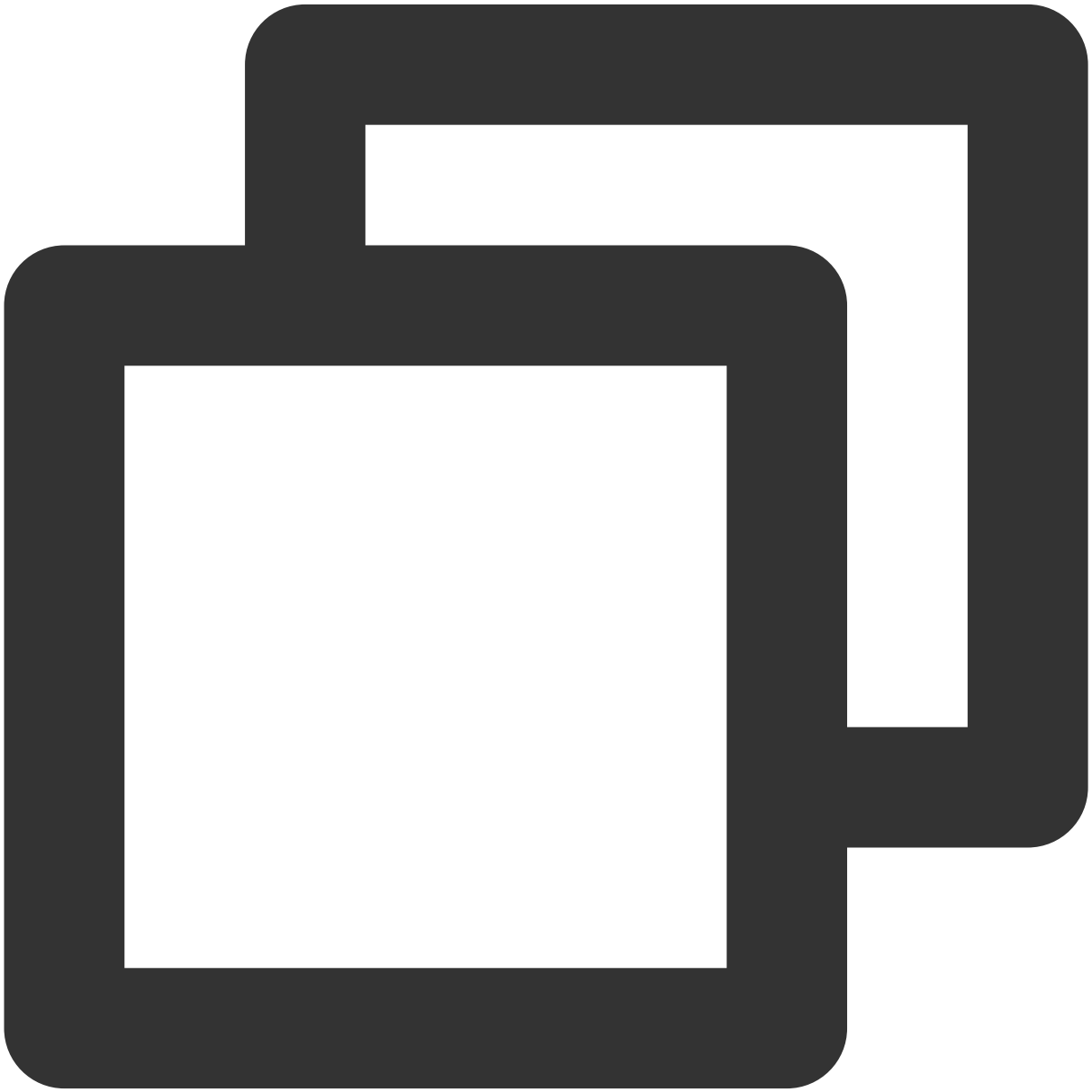
函数原型



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

参数	类型	意义
duckerTime	int	淡出时间。

示例代码

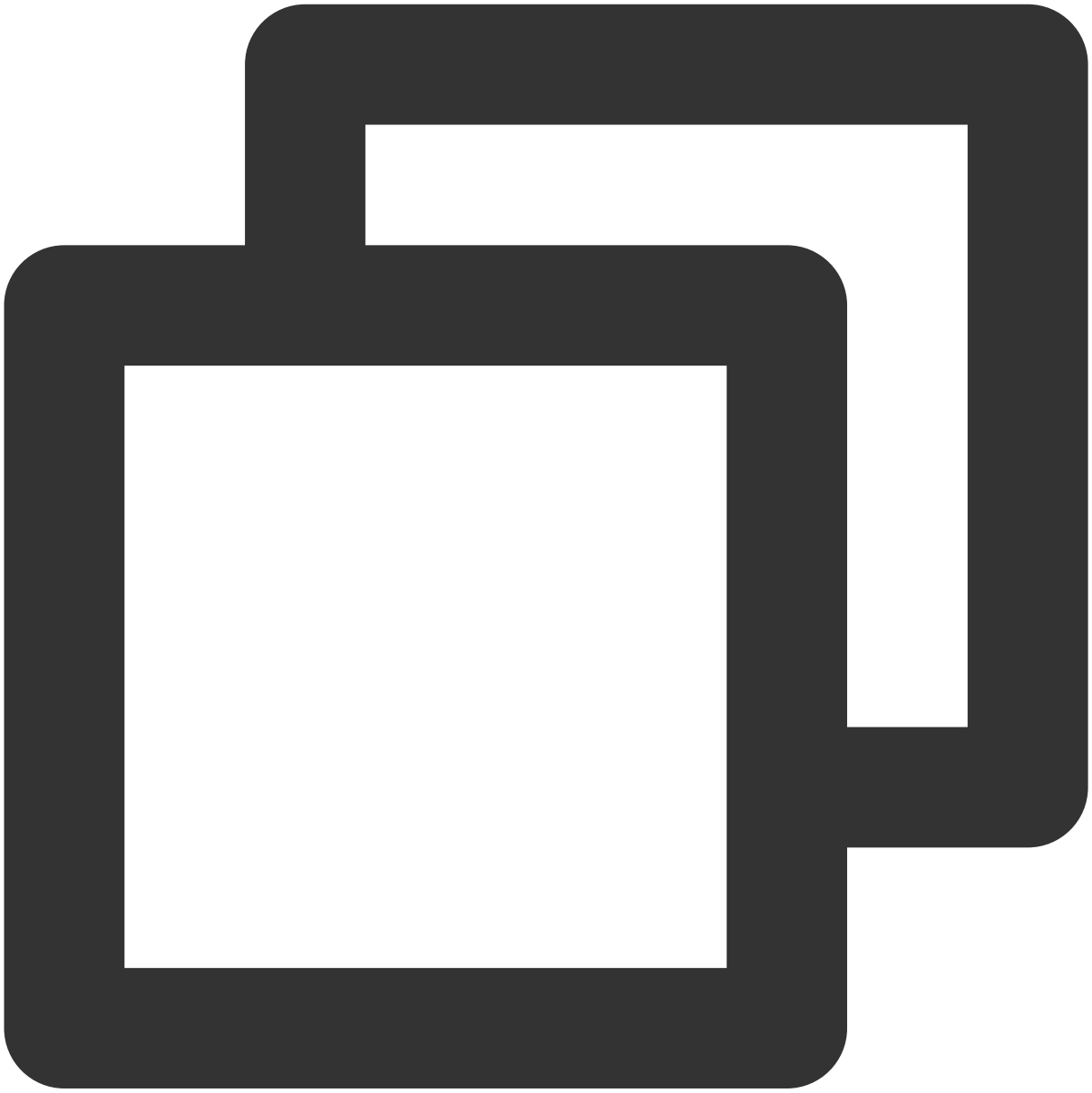


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->StopAccompany(0);
```

伴奏是否播放完毕

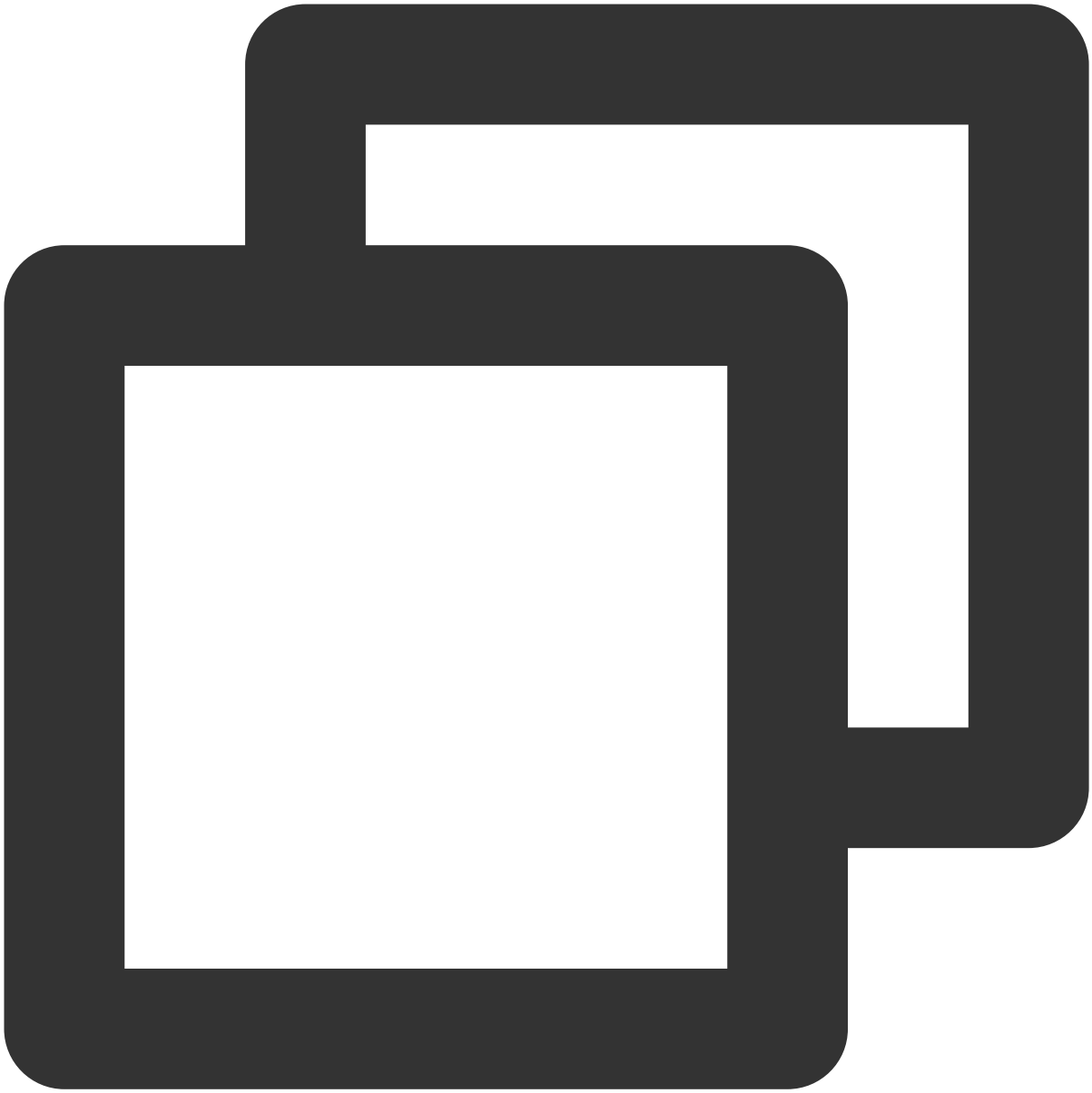
如果播放完毕，返回值为 true，如果没播放完，返回值为 false。

函数原型



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

示例代码

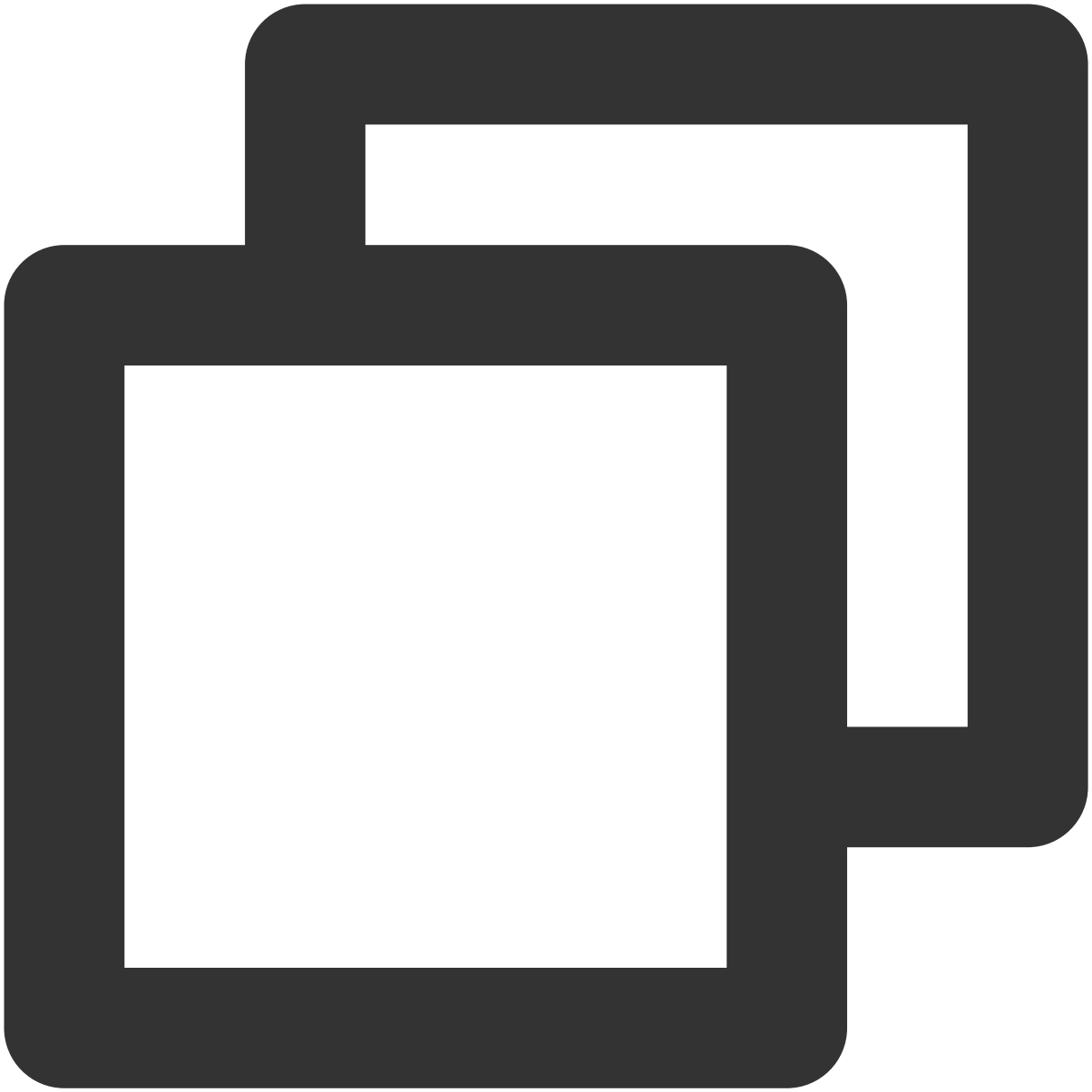


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->IsAccompanyPlayEnd();
```

暂停播放伴奏

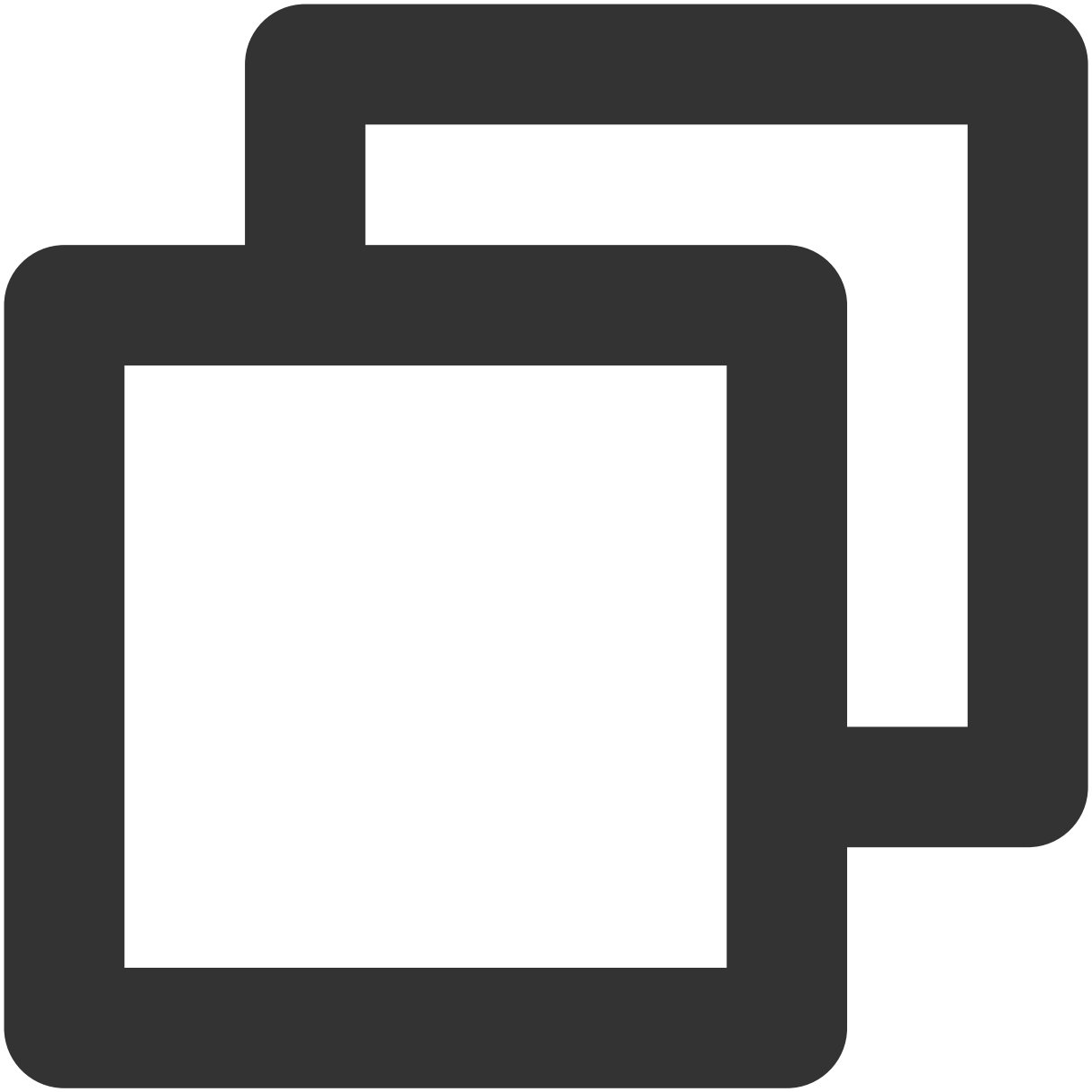
调用 PauseAccompany 接口暂停播放伴奏。

函数原型



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

示例代码

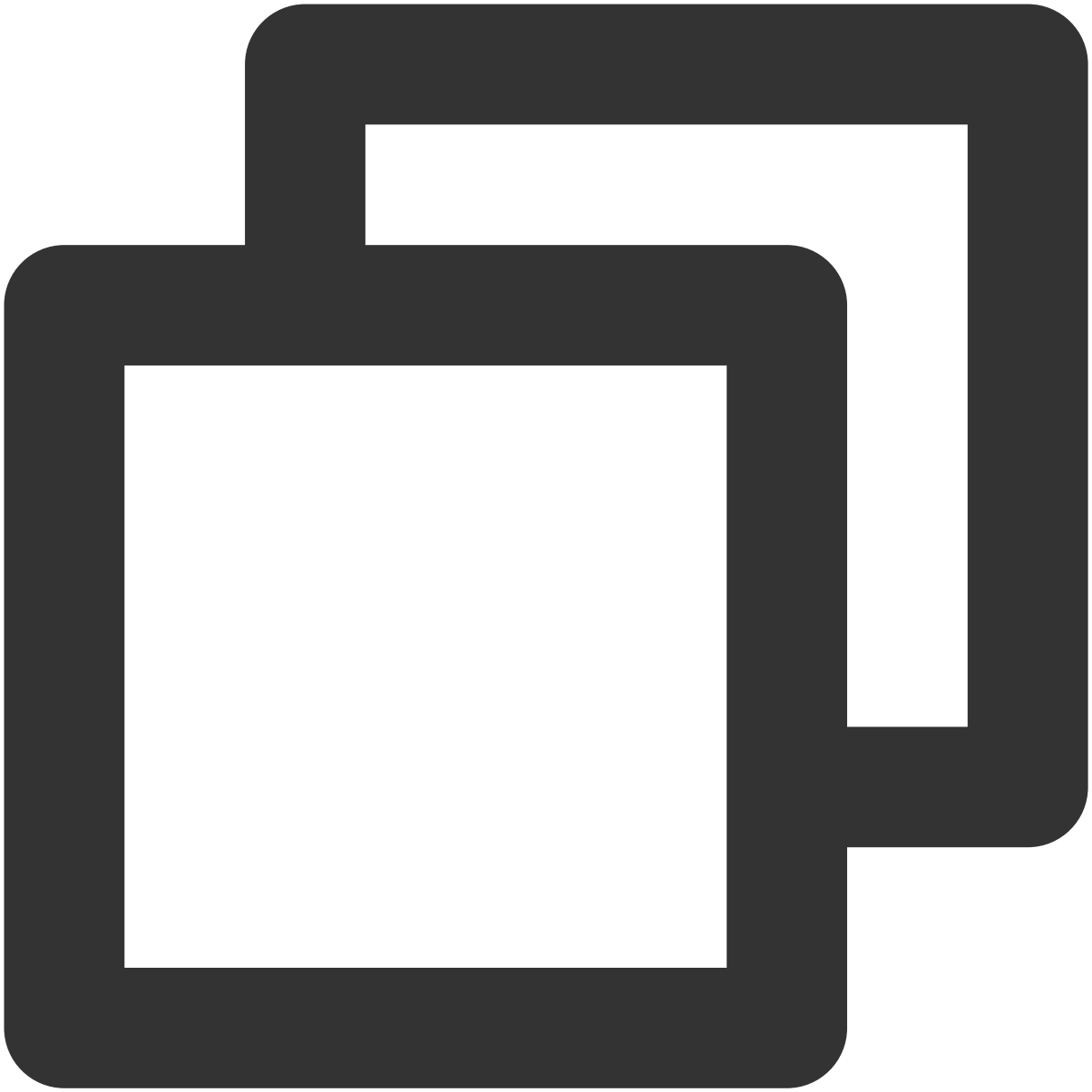


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->PauseAccompany();
```

恢复播放伴奏

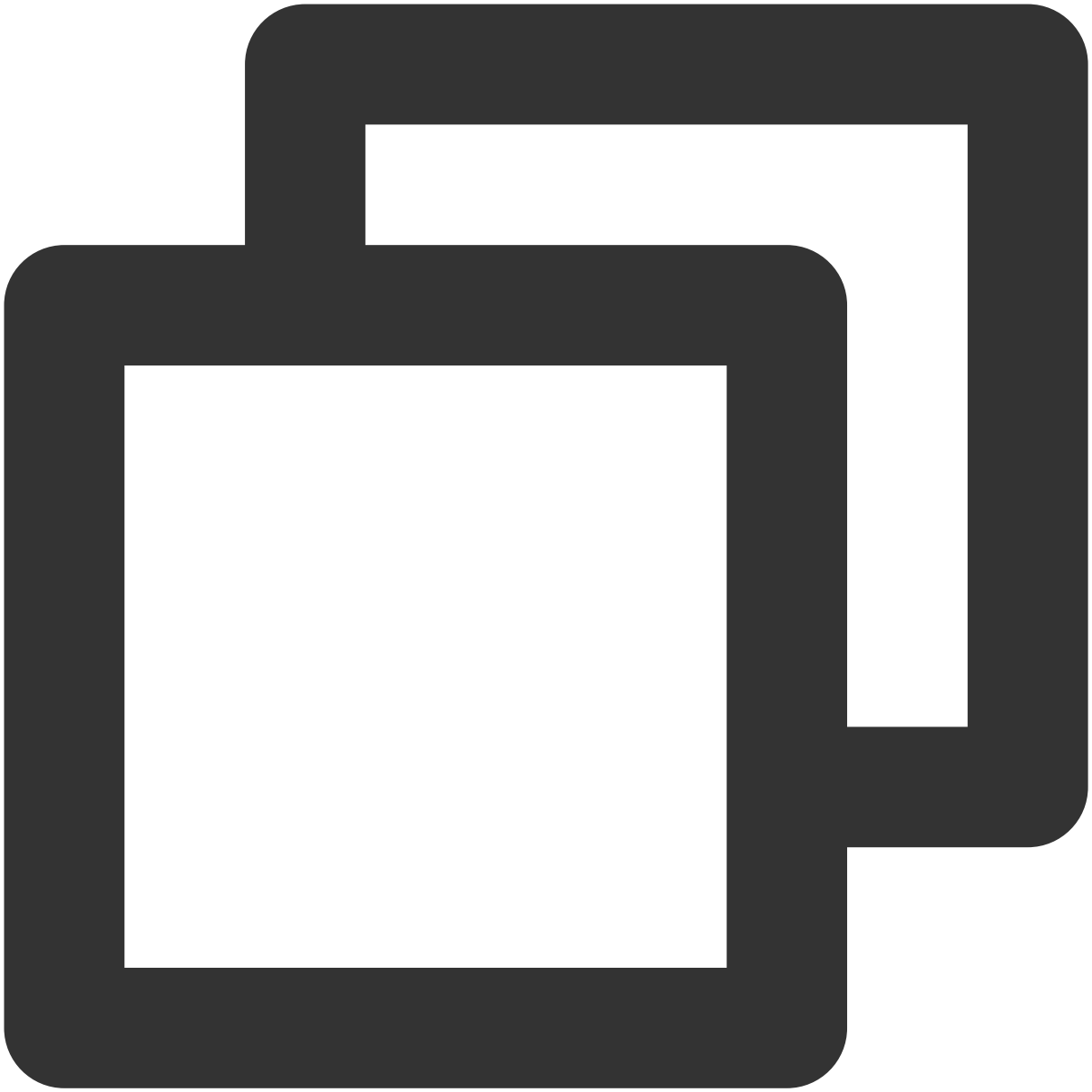
ResumeAccompany 接口用于恢复播放伴奏。

函数原型



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

示例代码

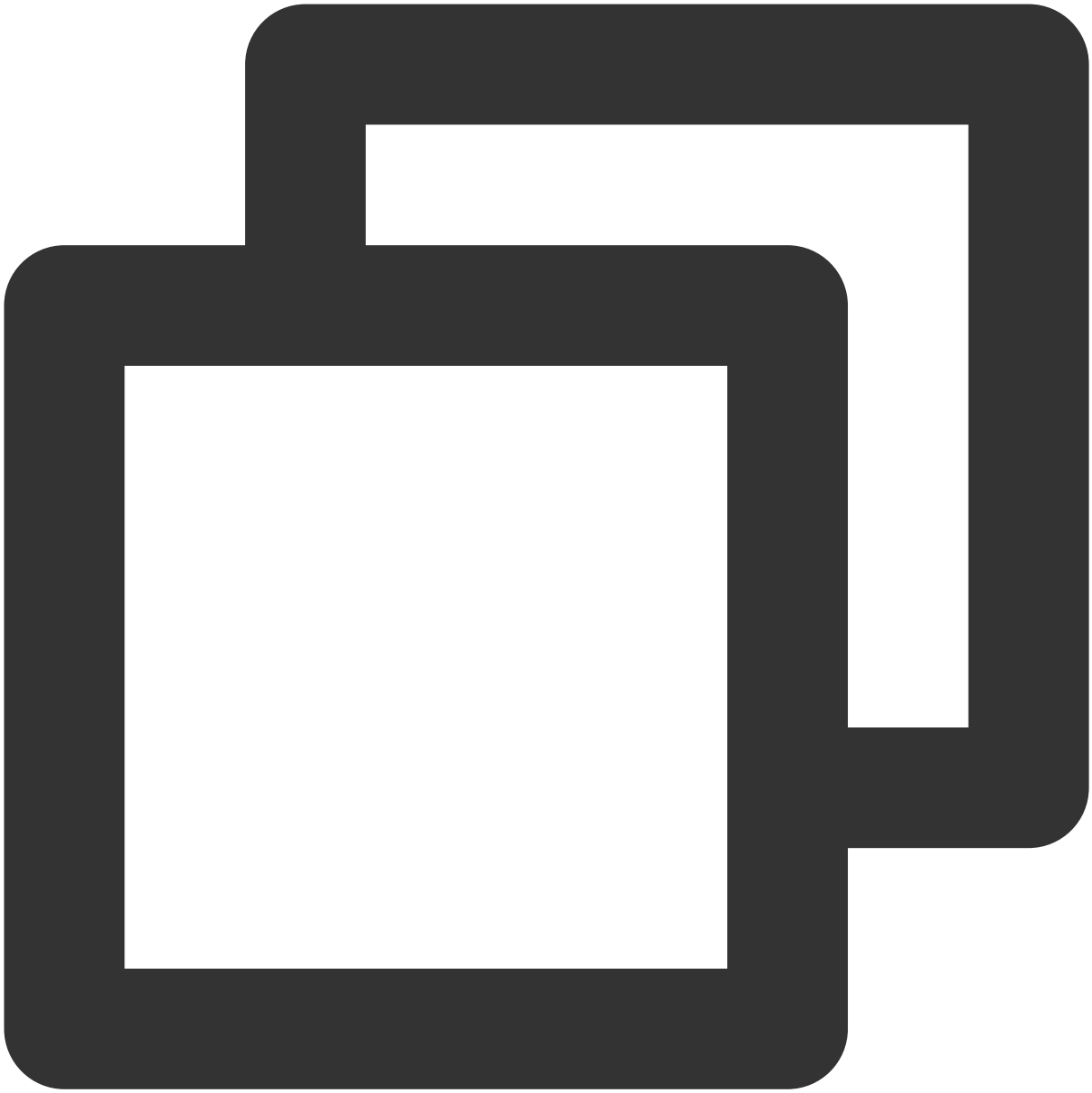


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->ResumeAccompany();
```

设置自己是否可以听到伴奏

此接口用于设置自己是否可以听到伴奏。

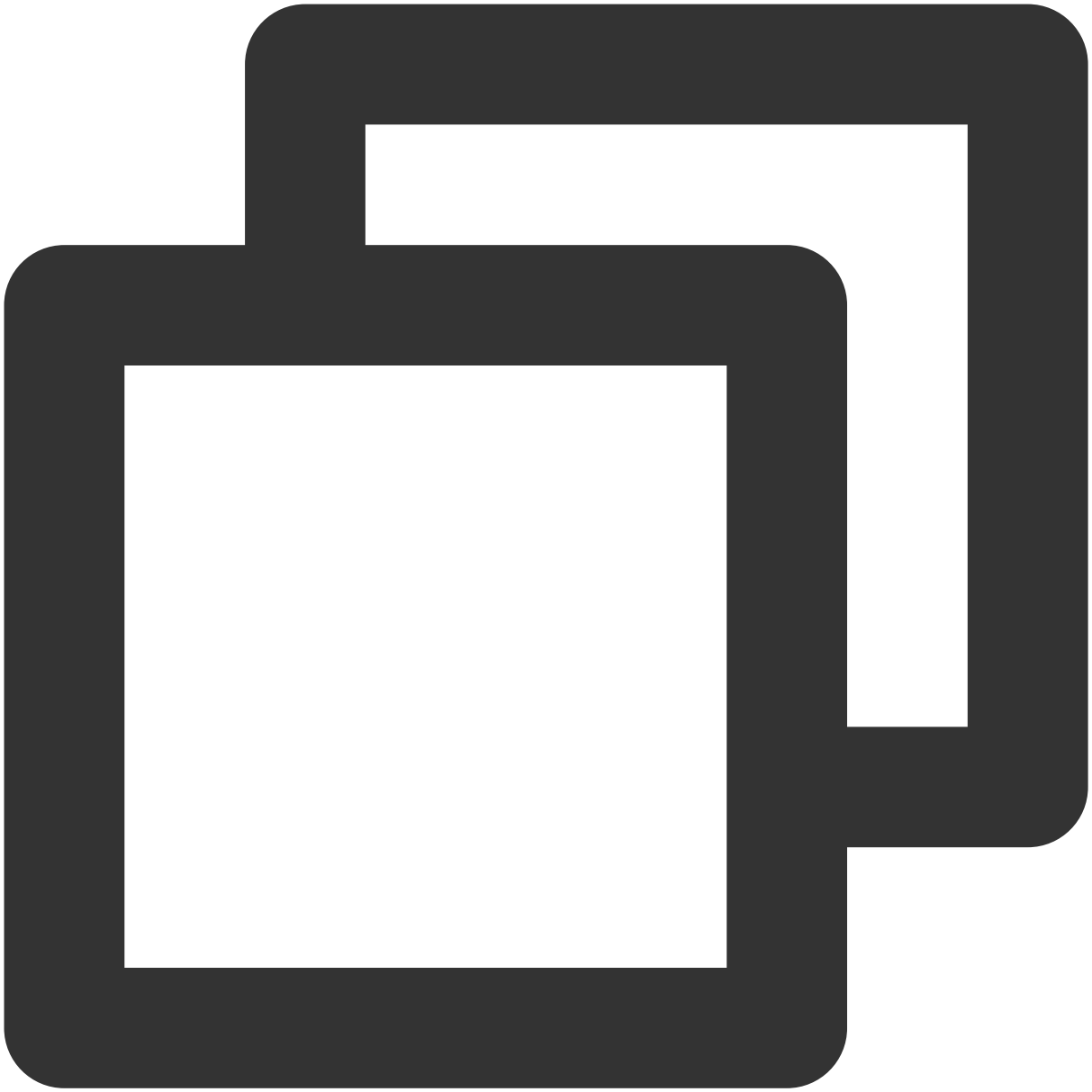
函数原型



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

参数	类型	意义
enable	bool	是否能听到。

示例代码

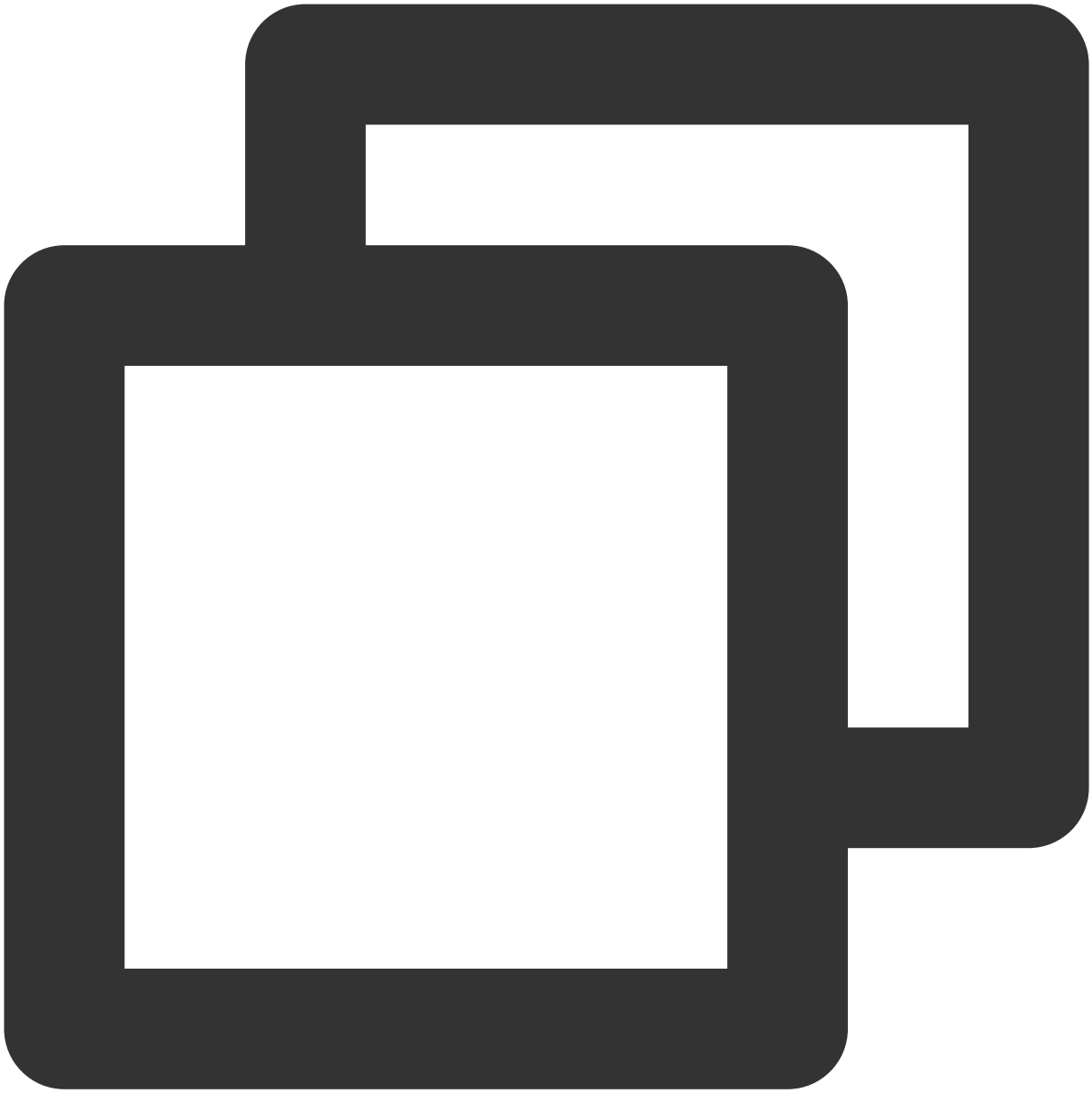


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->EnableAccompanyPlay(false);
```

设置他人是否也可以听到伴奏

设置他人是否也可以听到伴奏。

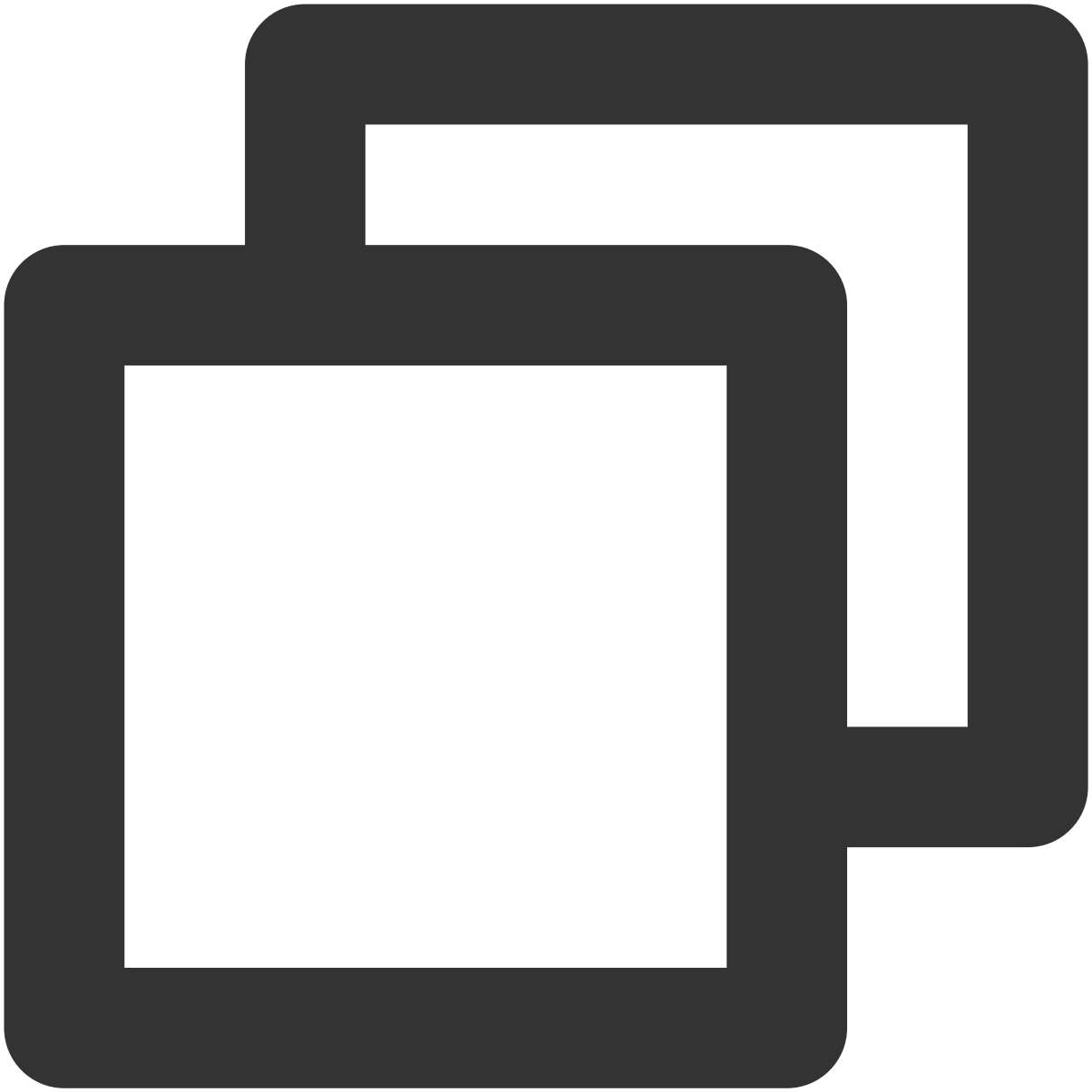
函数原型



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

参数	类型	意义
enable	bool	是否能听到。

示例代码

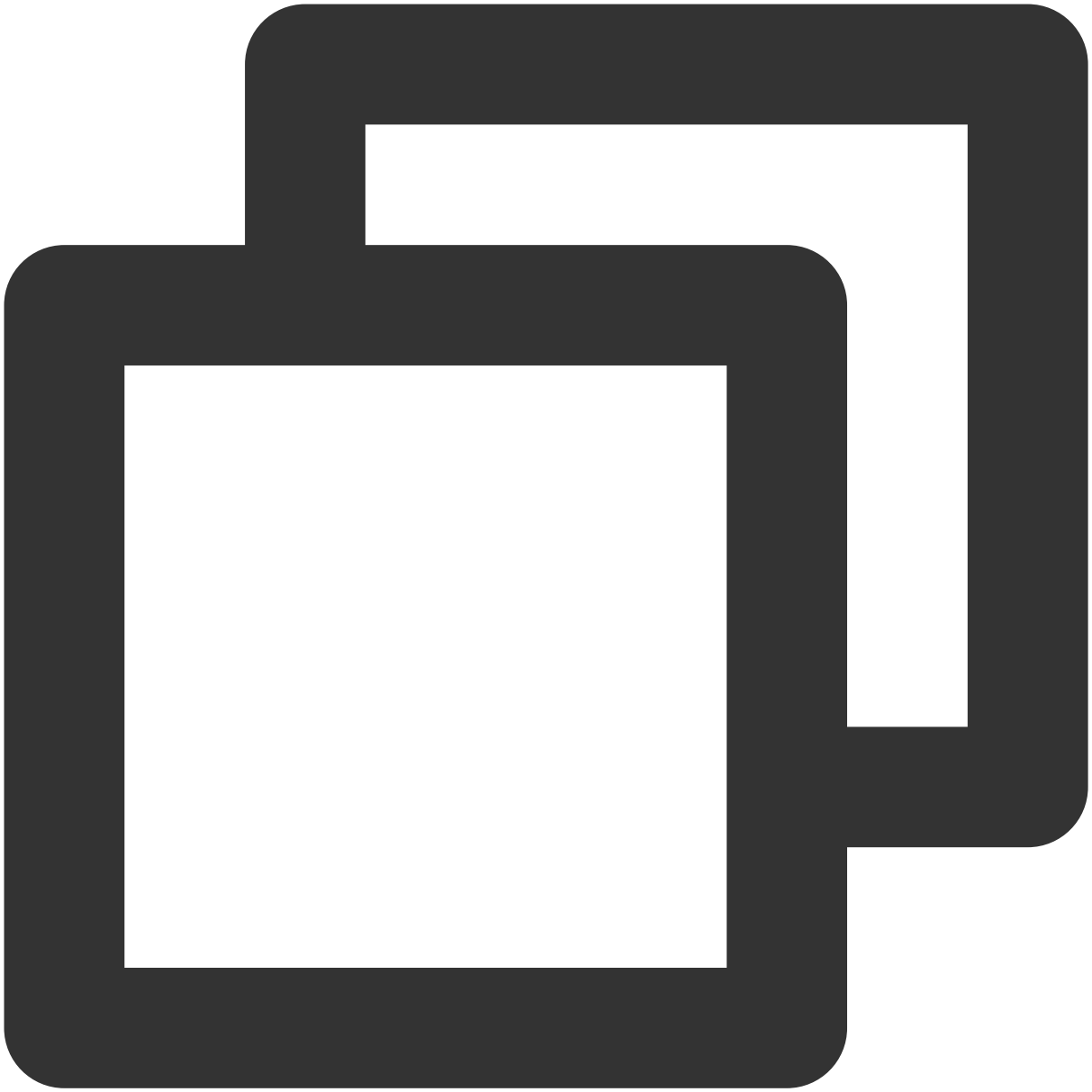


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->EnableAccompanyLoopBack(false);
```

设置伴奏音量

调用 SetAccompanyVolume 接口设置伴奏音量，默认值为100，数值大于100音量增益，数值小于100音量减益，值域为0 - 200。

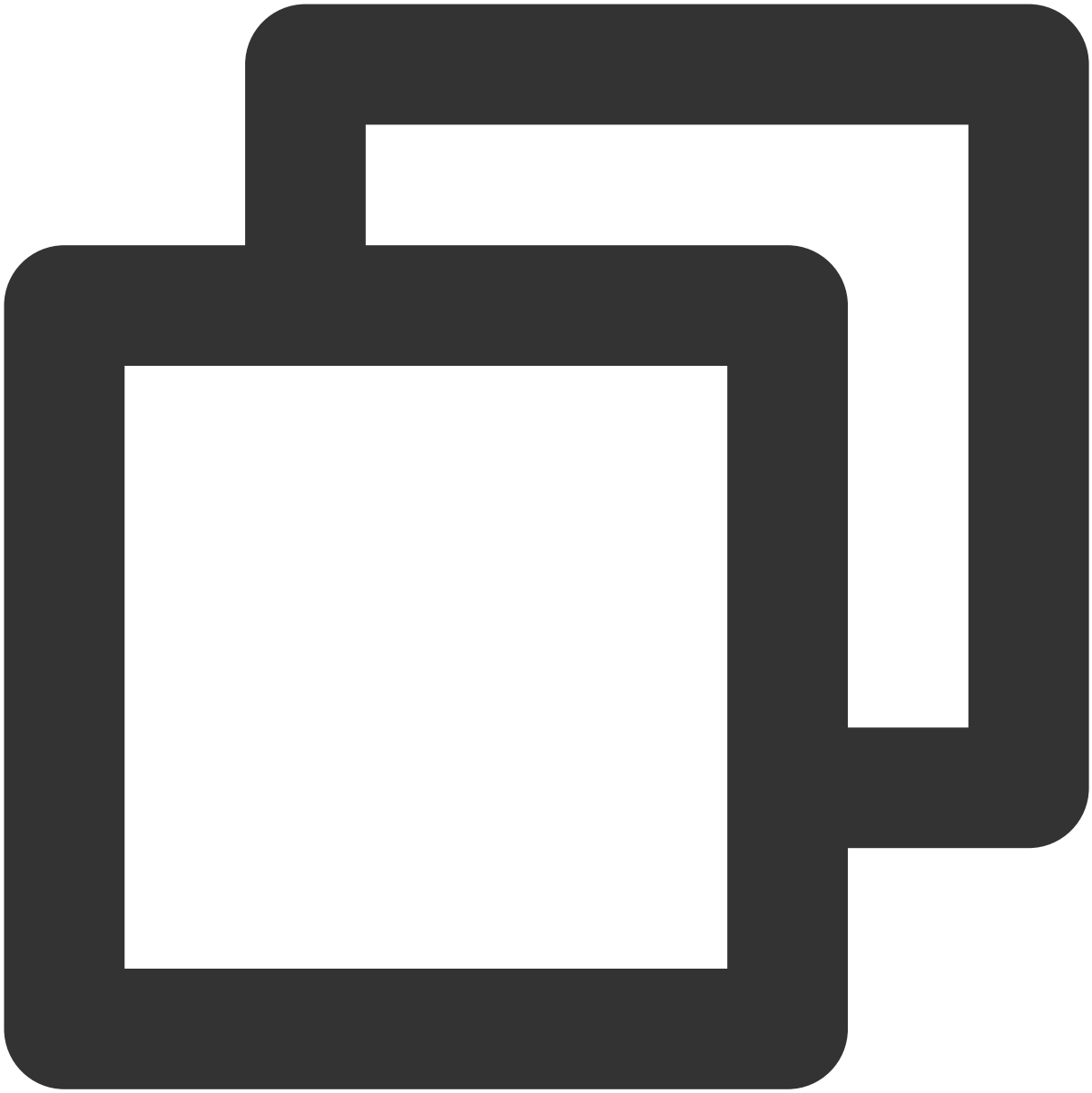
函数原型



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

参数	类型	意义
vol	int	音量数值。

示例代码

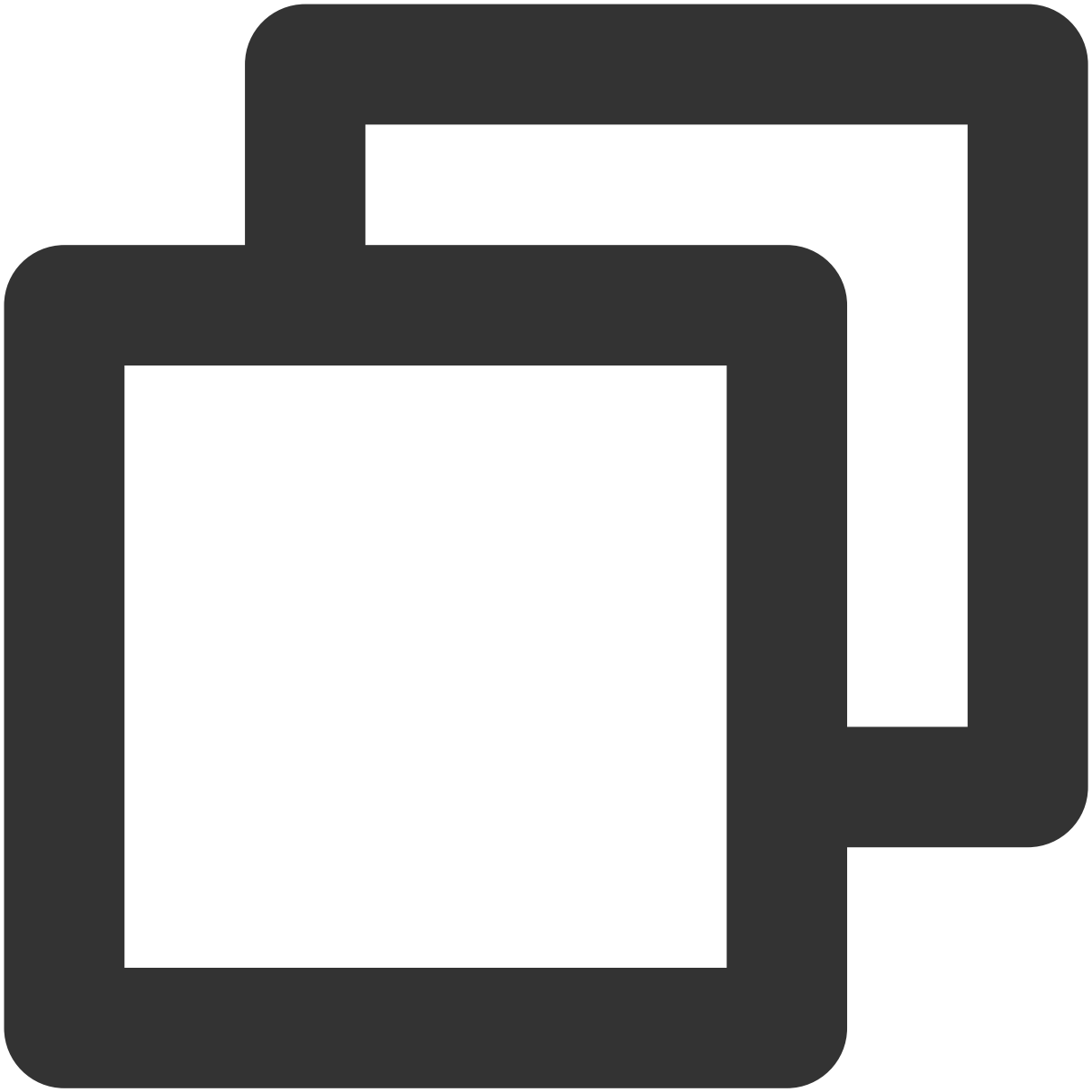


```
int vol=100;
ITMGContextGetInstance()->GetAudioEffectCtrl()->SetAccompanyVolume(vol);
```

获取播放伴奏的音量

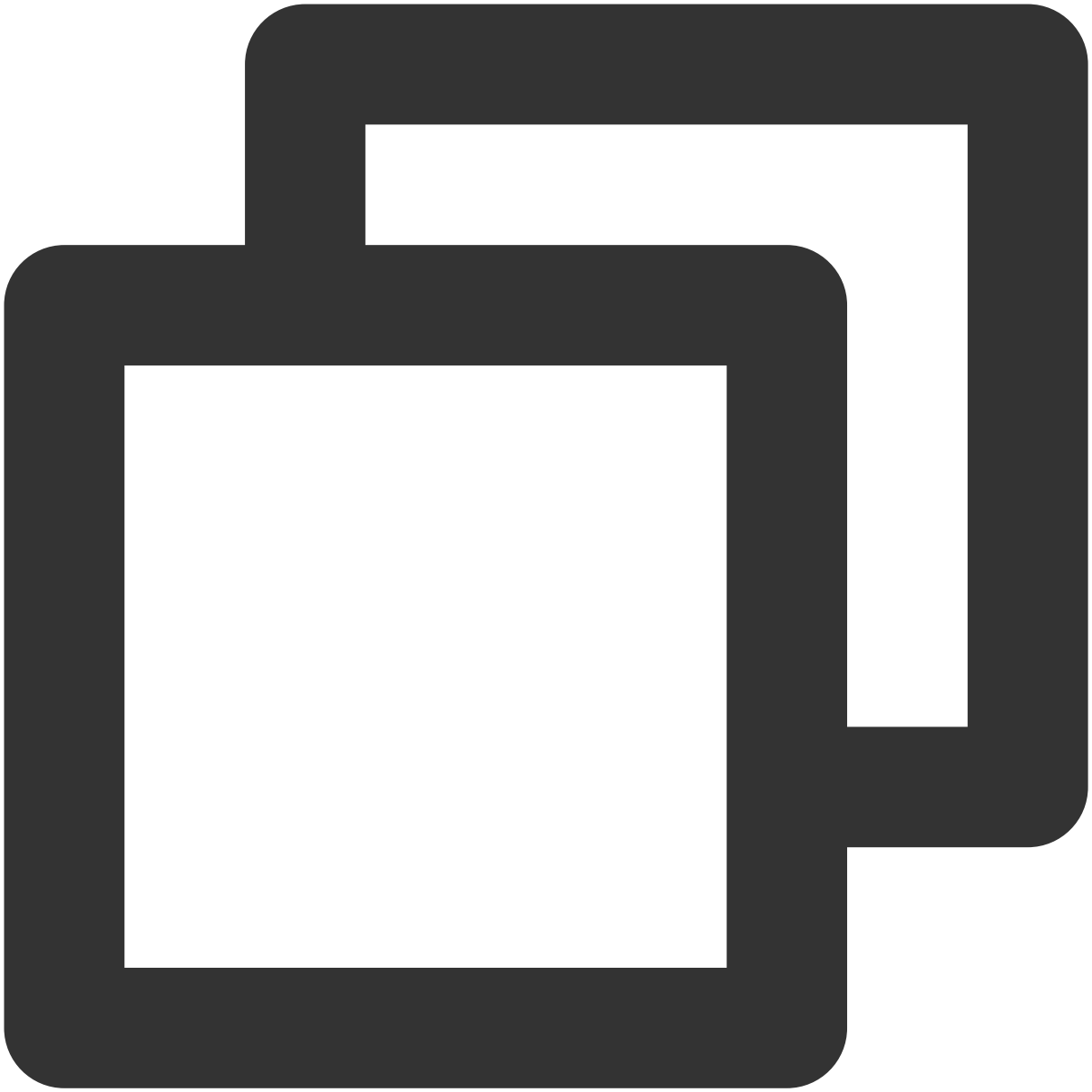
GetAccompanyVolume 接口用于获取伴奏音量。

函数原型



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

示例代码

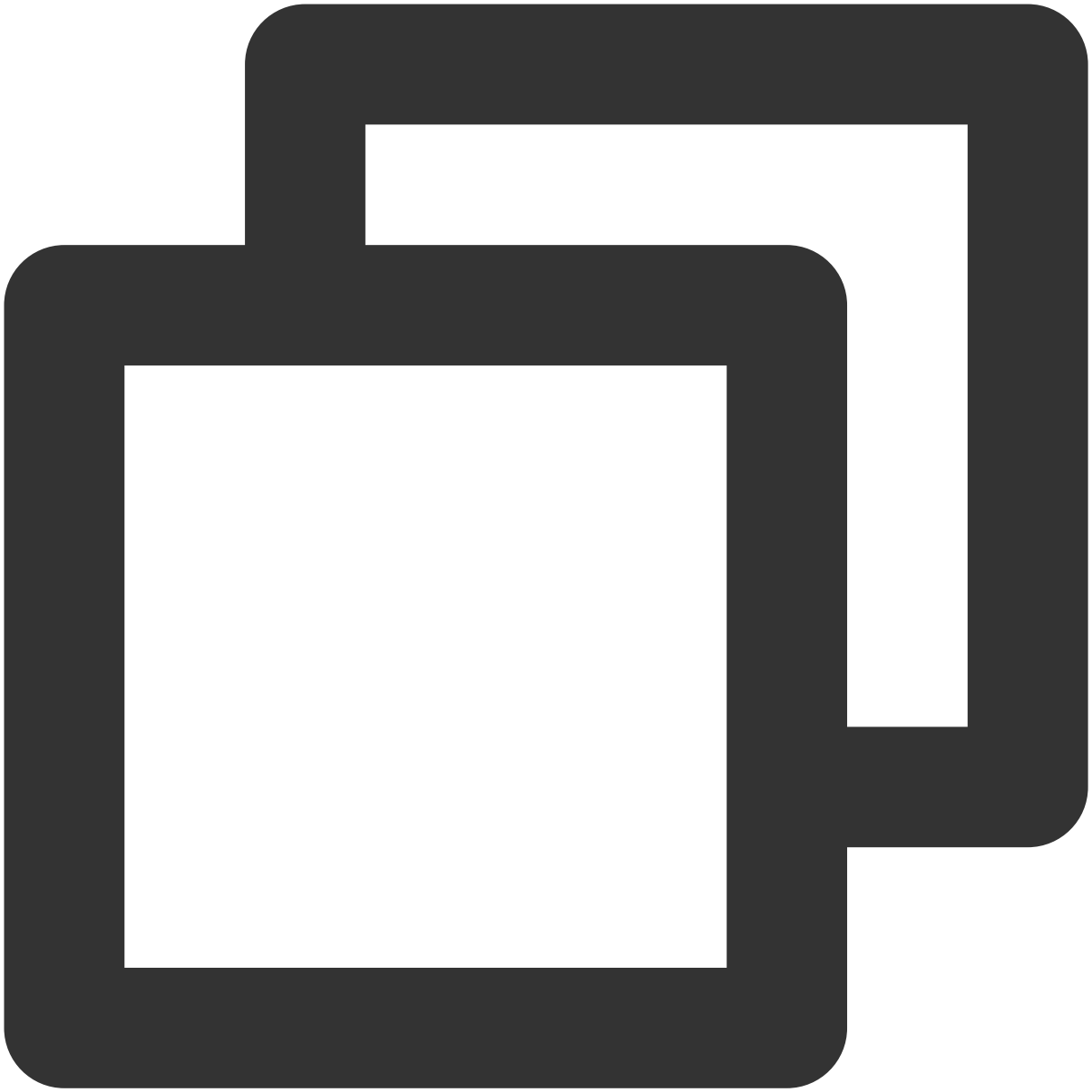


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->GetAccompanyVolume();
```

获得伴奏播放进度

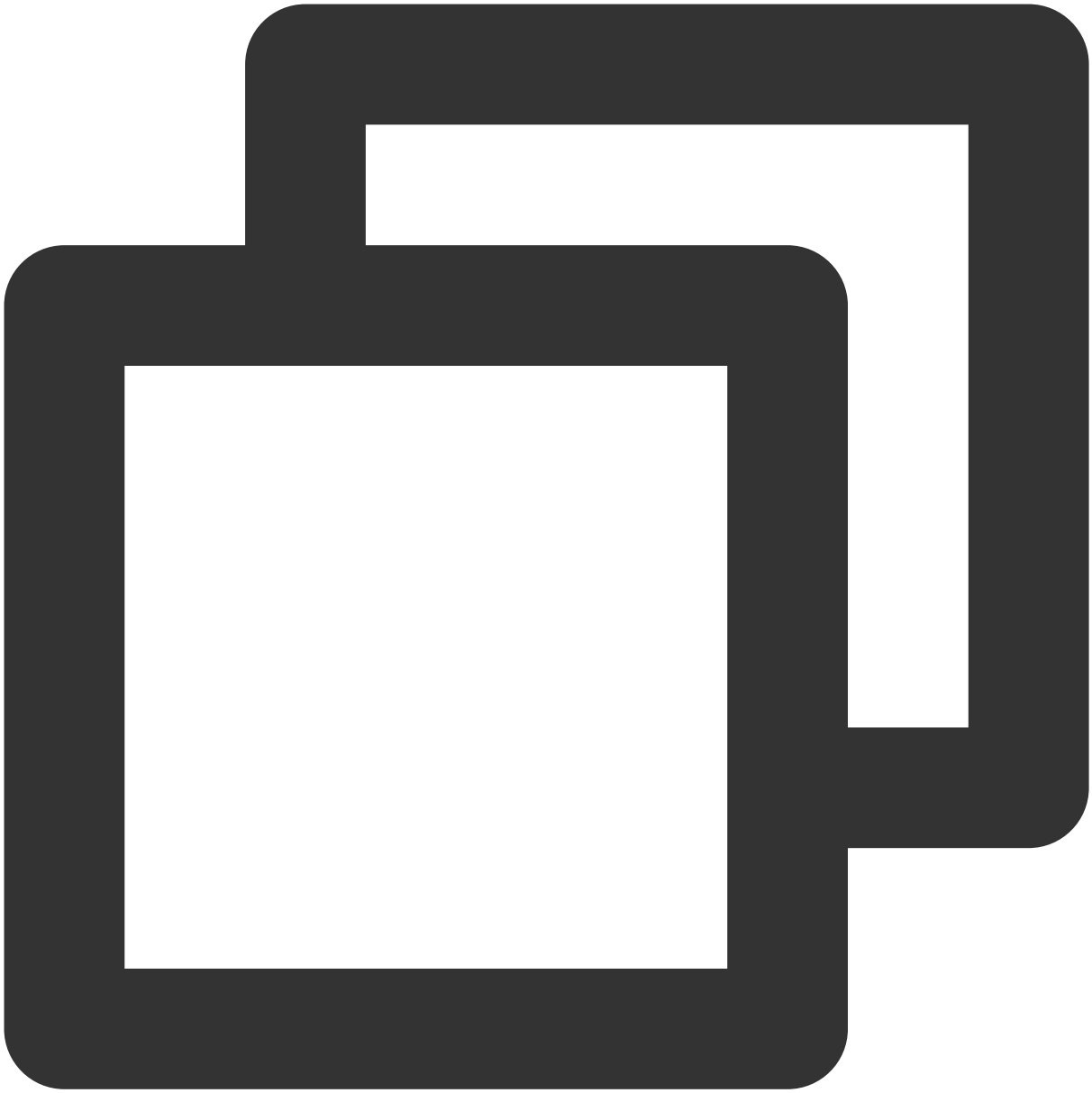
以下两个接口用于获得伴奏播放进度。需要注意：Current / Total = 当前循环次数，Current % Total = 当前循环播放位置。

函数原型



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

示例代码

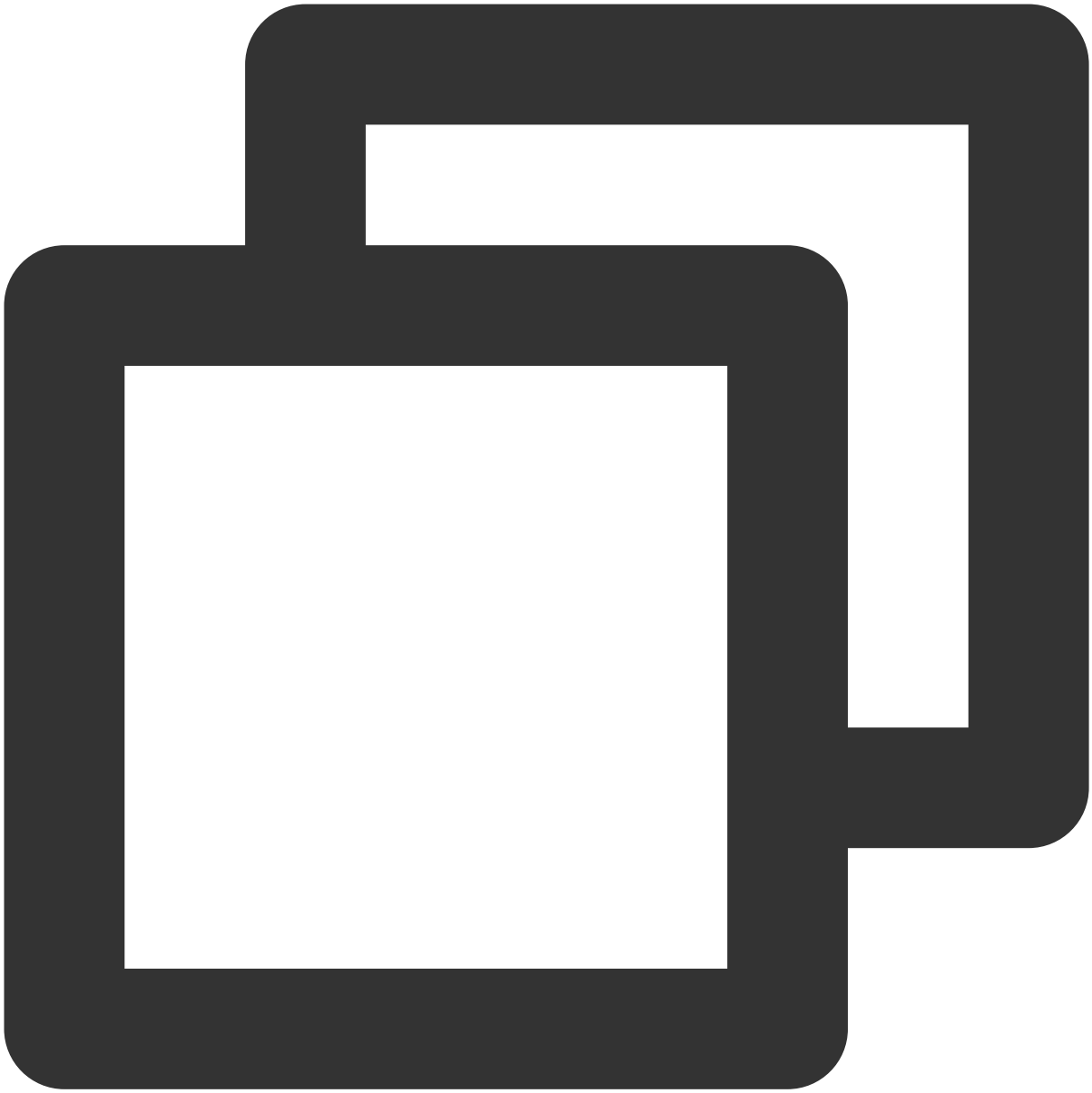


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->GetAccompanyFileTotalTimeByMs();
ITMGContextGetInstance()->GetAudioEffectCtrl()->GetAccompanyFileCurrentPlayedTimeBy
```

设置播放进度

SetAccompanyFileCurrentPlayedTimeByMs 接口用于设置播放进度。

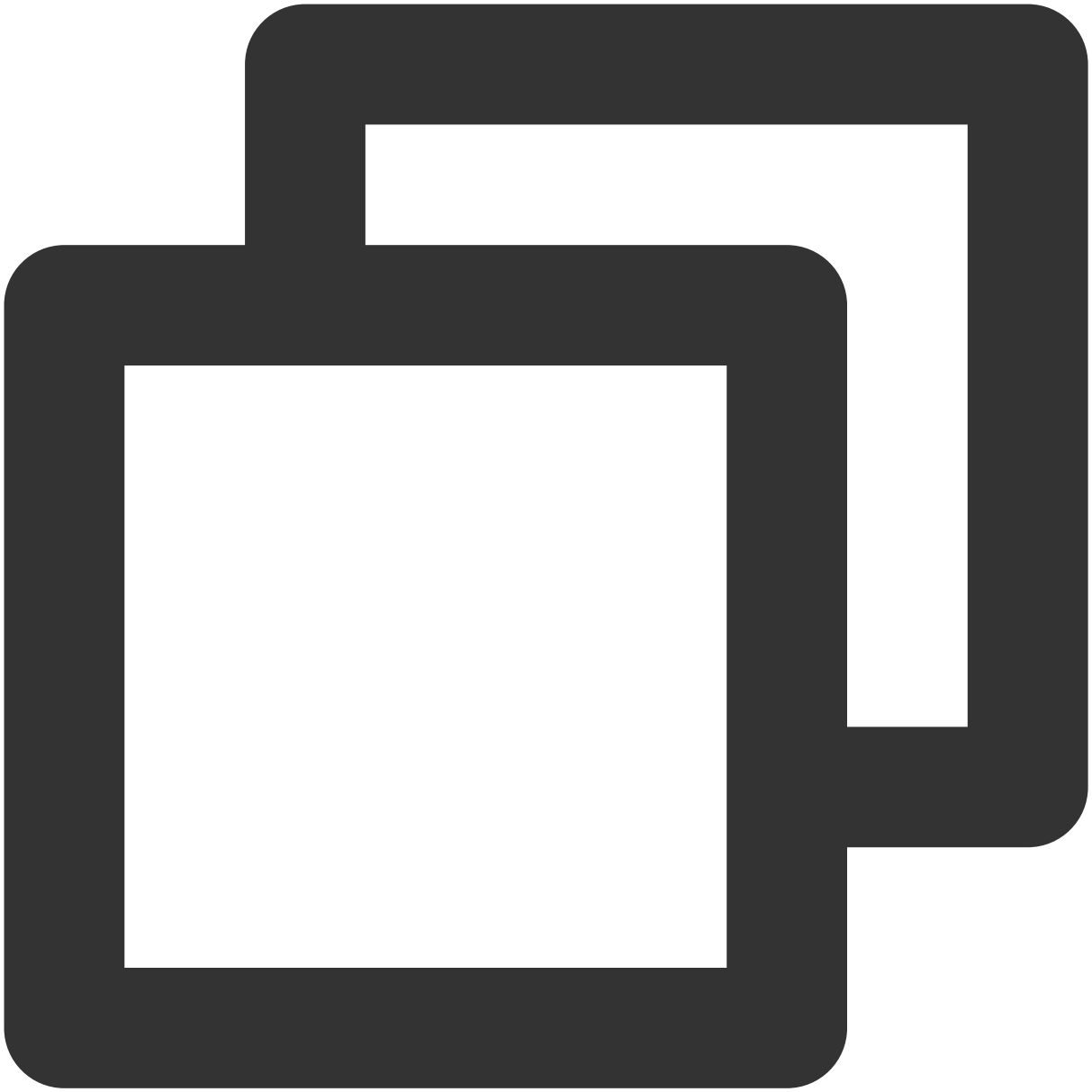
函数原型



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

参数	类型	意义
time	int	播放进度，以毫秒为单位。

示例代码

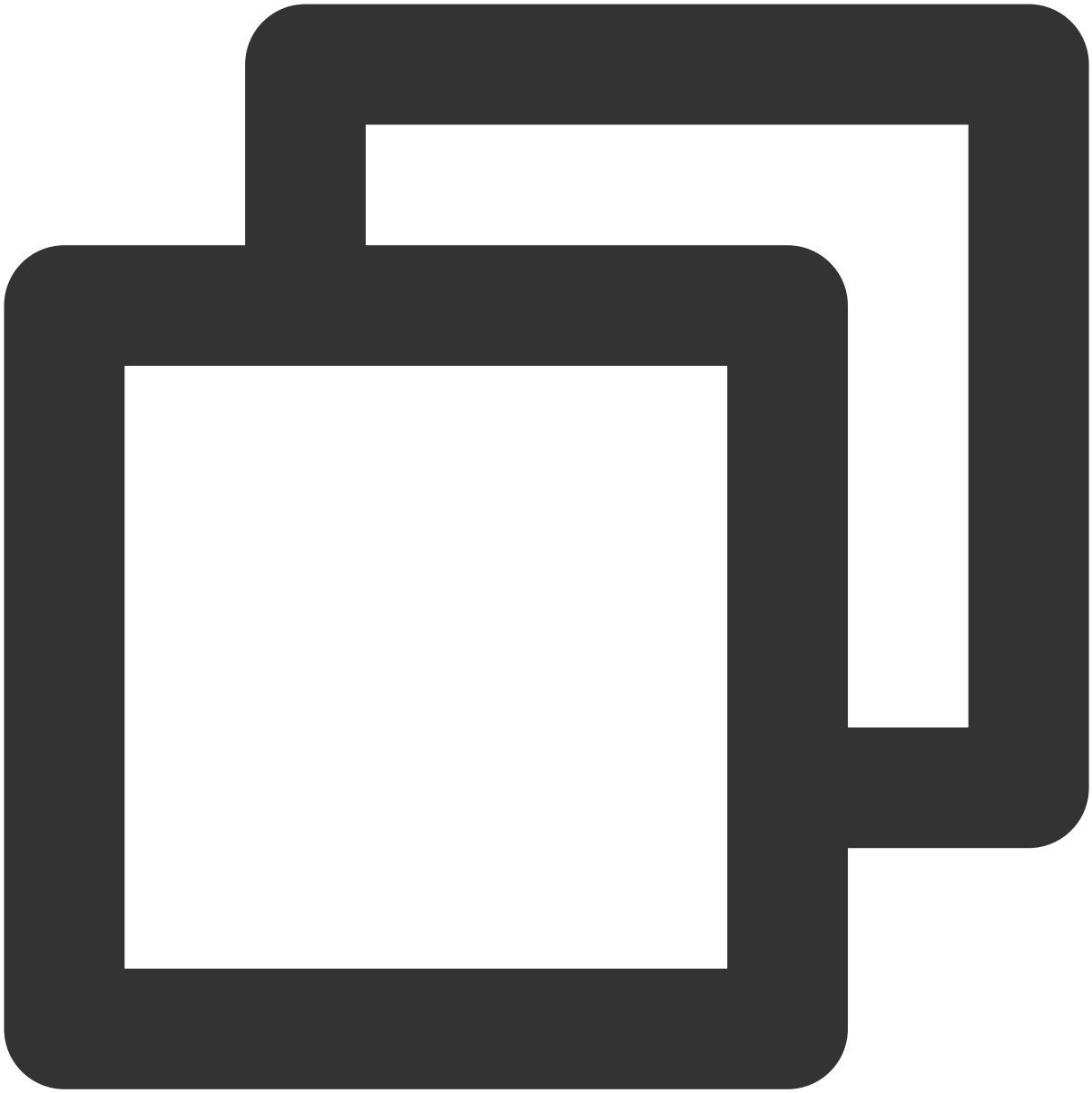


```
ITMGContextGetInstance()->GetAudioEffectCtrl()->SetAccompanyFileCurrentPlayedTimeBy
```

设置伴奏音调

SetAccompanyKey 接口用于调整伴奏的音调，在启动伴奏之前调用。

函数原型



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

参数	类型	意义
nKey	int	升降 Key，推荐范围-4到4。当设置为0时为原声调。

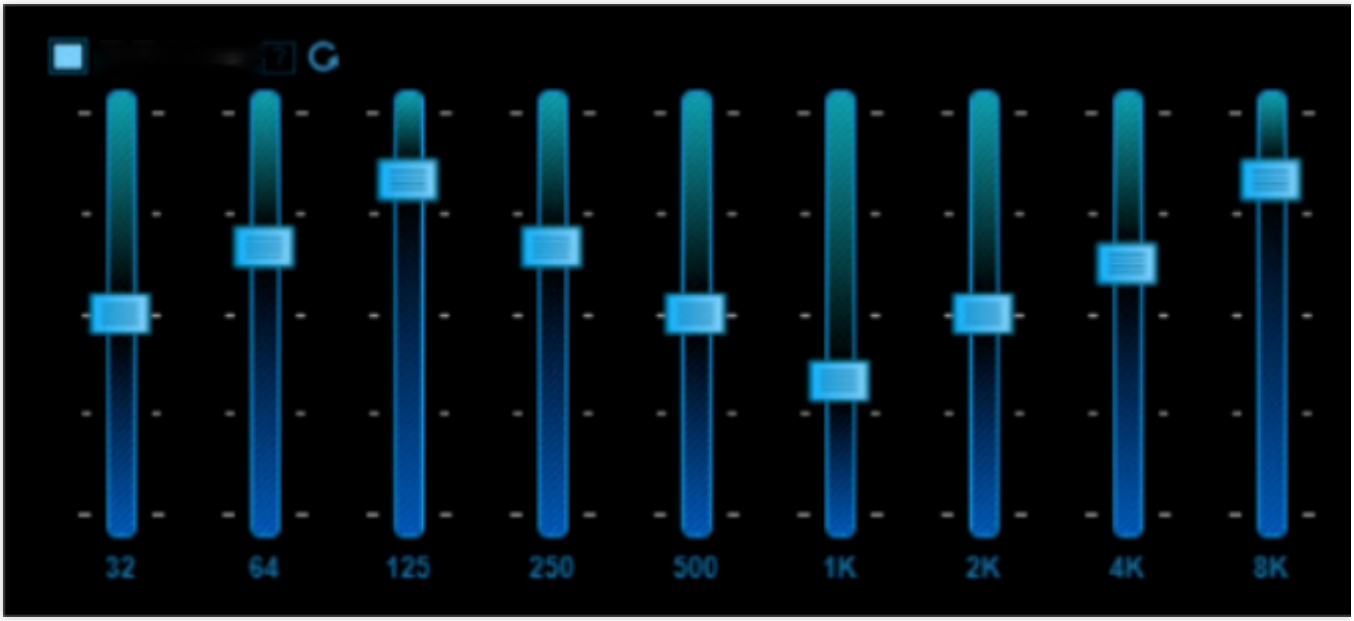
错误码列表

错误码名称	错误码值	错误码含义	解决方法
QAV_ERR_ACC_OPENFILE_FAILED	4001	打开文件失败	检查文件路径及文件是否存在，检查是否有访问文件的权限。
QAV_ERR_ACC_FILE_FORAMT_NOTSUPPORT	4002	不支持的文件格式	检查文件格式是否正确。
QAV_ERR_ACC_DECODER_FAILED	4003	解码失败	检查文件格式是否正确。
QAV_ERR_ACC_BAD_PARAM	4004	参数错误	检查代码中所填参数是否正确。
QAV_ERR_ACC_MEMORY_ALLOC_FAILED	4005	内存分配失败	系统资源耗尽，如果一直存在此错误码，请联系开发人员。
QAV_ERR_ACC_CREATE_THREAD_FAILED	4006	创建线程失败	系统资源耗尽，如果一直存在此错误码，请联系开发人员。
QAV_ERR_ACC_STATE_ILLIGAL	4007	状态非法	未处于某种状态，去调用需要处于这个状态才允许调用的接口时，则会产生这个错误。

实时语音均衡器

最近更新时间：2024-01-18 14:28:46

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文为您介绍游戏多媒体引擎实时语音均衡器功能的接入指引。



使用场景

GME 均衡器功能，可以将 GME SDK 采集的音频流实时进行均衡器调整。该功能可应用于线上K歌场景，玩家开始唱歌后，调用 **GME SDK** 的 **均衡器** 接口，即可将玩家的实时语音流做声音美化效果方面的调整。

前提条件

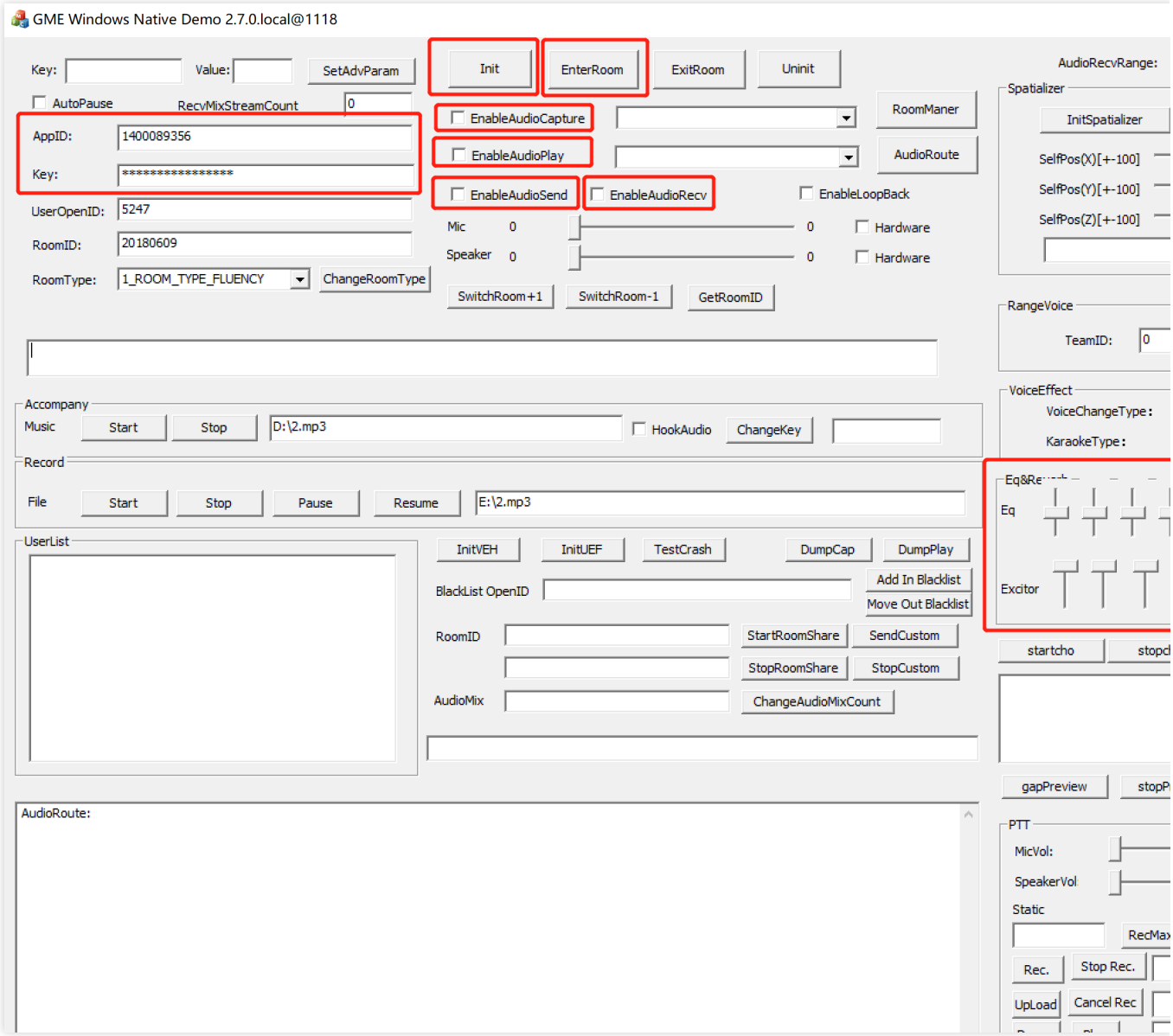
已开通实时语音服务：可参见 服务开通指引。
已接入 GME SDK：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入、Unity SDK 快速接入、Unreal SDK 快速接入。

体验 Demo

下载 Demo

下载地址 >>

此体验 Demo 为一个 Windows 可执行程序，界面如下：



配置参数

在 Appid 和 Key 的输入框中填入自己申请的 GME AppID 以及 Key。

如果需要，您也可以输入目标房间号以及 OpenID。

使用方法

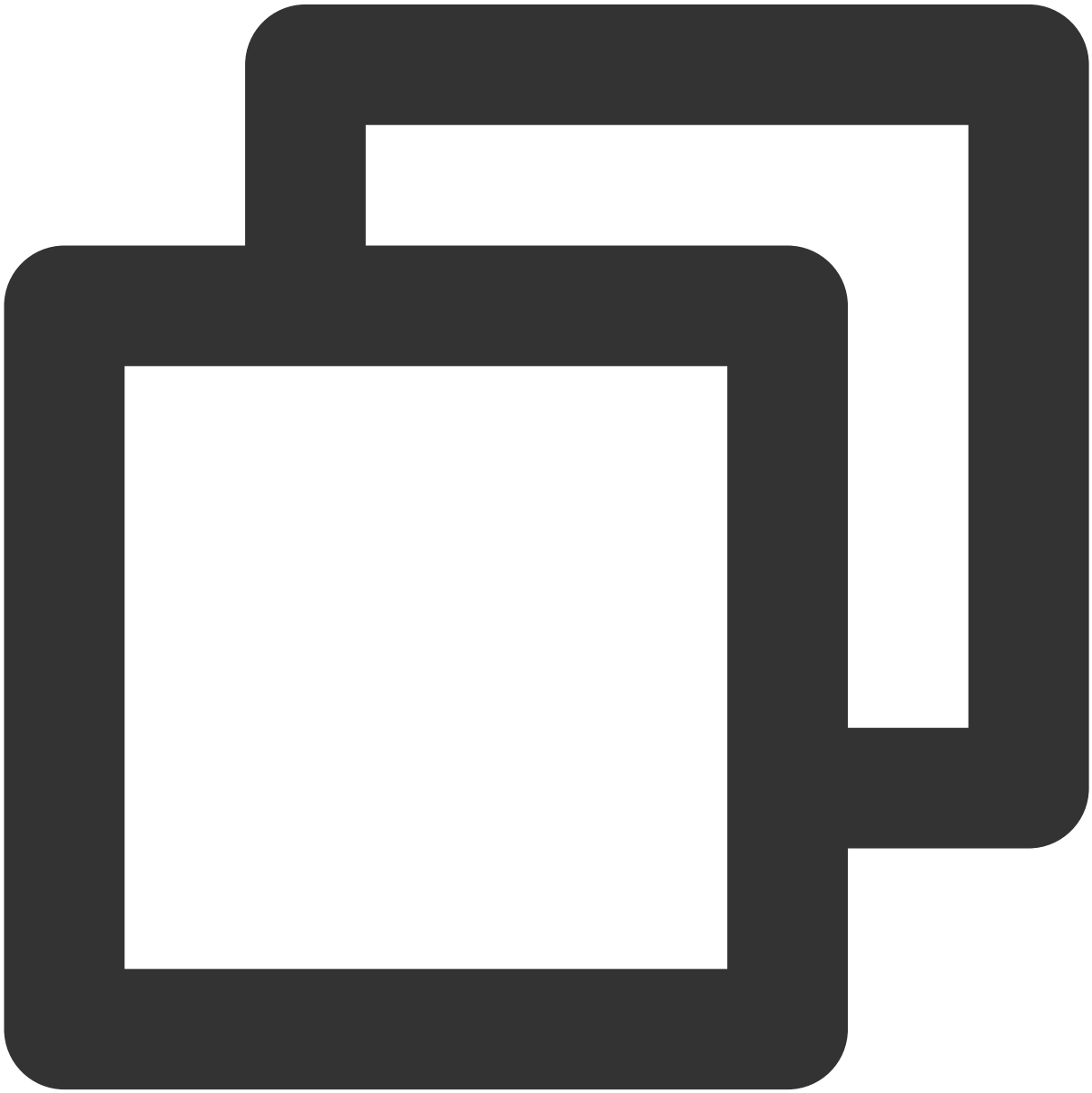
1. 进房打开麦克风扬声器流程如下：Init > EnterRoom > EnableCapture > EnablePlay > EnableSend > EnableRecv。
2. 进房成功之后，可以开启 **EnableLoopBack** 听到自己的声音。

3. 后续再调整红框中的 **EQ** 均衡器（EQ 是频段增益，Exditor 和 Reverb 是混响器）。

均衡器功能接入

使用此接口需要在进房成功的状态下，才能对本端采集的声音实现均衡器调节效果。

函数原型



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

参数	类型	意义
pEqualizer	ITMG_VOICE_TYPE_EQUALIZER	频段增益
pReverb	ITMG_VOICE_TYPE_REVERB	涵盖 HARMONIC 和 REVERB

结构体详细信息

ITMG_VOICE_TYPE_EQUALIZER 中结构体成员为 float 类型，数值范围为-12到12。

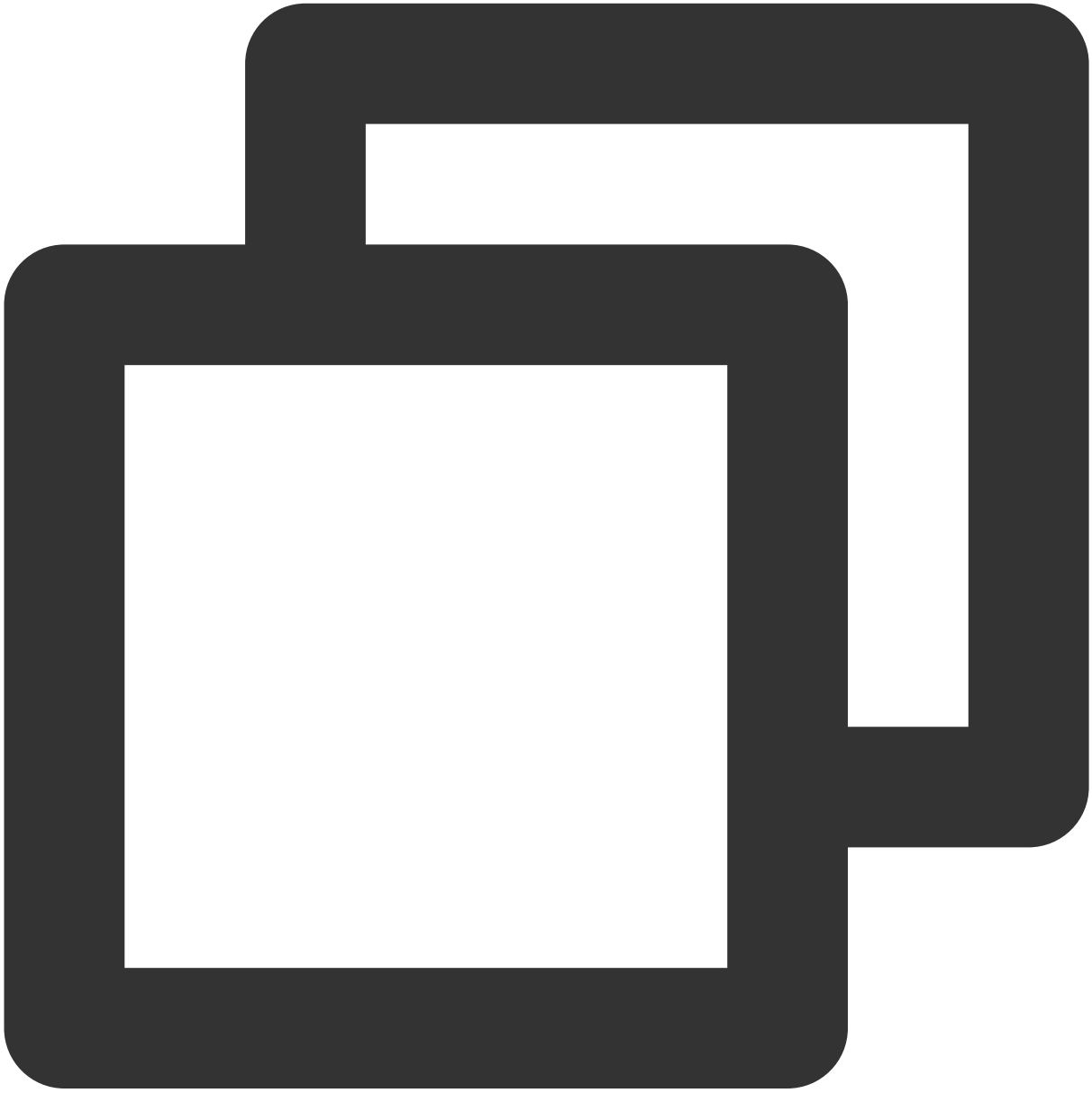
ITMG_VOICE_TYPE_EQUALIZER	意义
EQUALIZER_32HZ	32HZ频段上施加的增益
EQUALIZER_64HZ	64HZ频段上施加的增益
EQUALIZER_128HZ	128HZ频段上施加的增益
EQUALIZER_250HZ	250HZ频段上施加的增益
EQUALIZER_500HZ	500HZ频段上施加的增益
EQUALIZER_1KHZ	1KHZ频段上施加的增益
EQUALIZER_2KHZ	2KHZ频段上施加的增益
EQUALIZER_4KHZ	4KHZ频段上施加的增益
EQUALIZER_8KHZ	8KHZ频段上施加的增益
EQUALIZER_16KHZ	16KHZ频段上施加的增益
EQUALIZER_MASTER_GAIN	整体音量

ITMG_VOICE_TYPE_REVERB 中结构体成员为 float 类型，数值范围为0到 1。

ITMG_VOICE_TYPE_REVERB
HARMONIC_GAIN
HARMONIC_START_FREQUENCY
HARMONIC_BASS_CONTROL
REVERB_SIZE
REVERB_DEPTH

REVERB_GAIN
REVERB_ECHO_DEPTH

示例代码



```
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);
}
```

均衡器使用指引

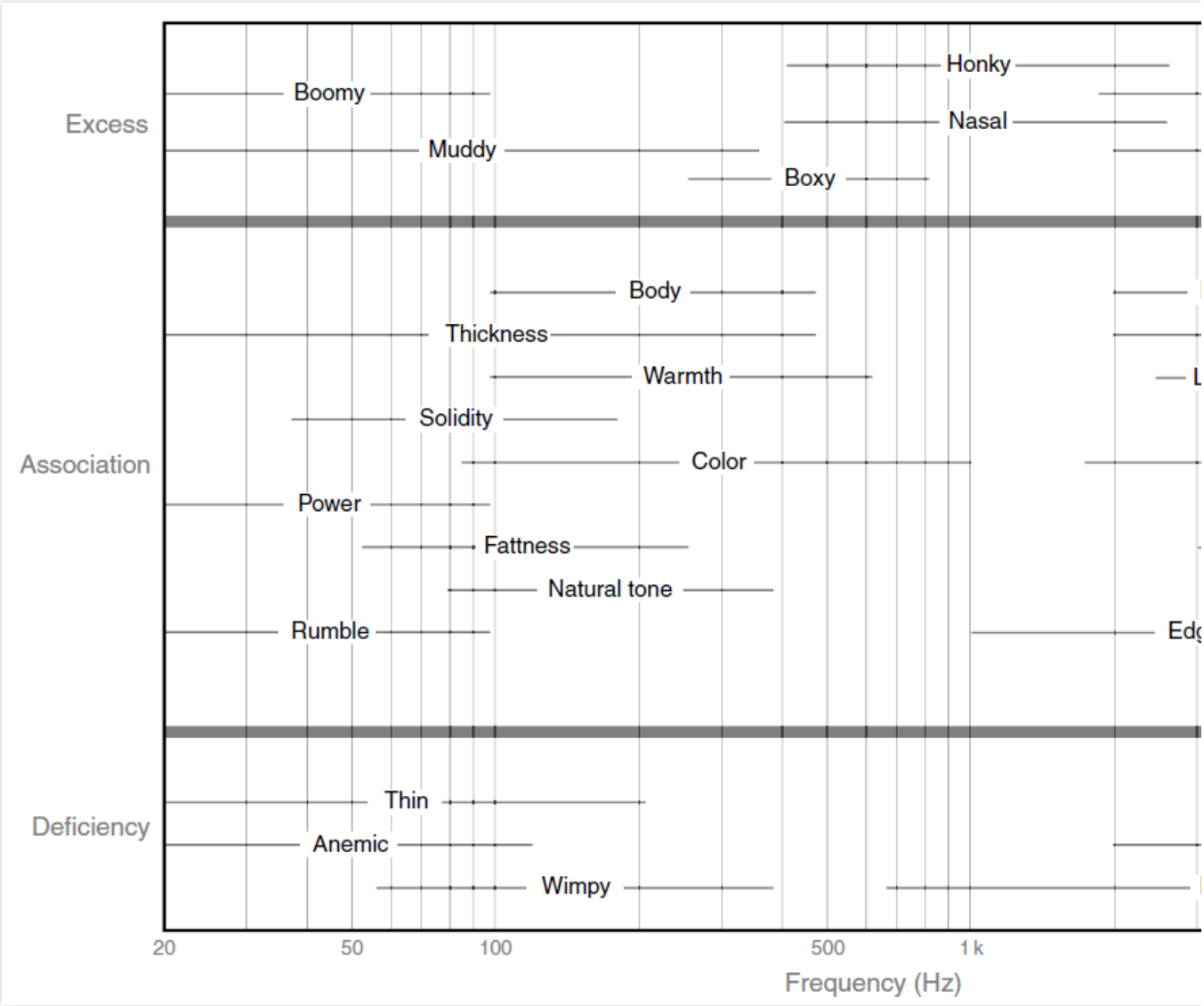
注意：

此处仅提供指引，如果需要更加深入的使用 EQ 效果，需要专业的调音师协作。

人耳可以听到的声音范围大致为20HZ - 20KHZ，人耳对于各频段的感知呈现对数关系，因此混音工程中常将人的可听频段分成10个八度分别进行调整，调音大致分为如下几个区域：

频段	区域	说明
20HZ - 32HZ	次声和超低音区	大部分频段低于人耳听觉下限，多用触觉感知，音乐中超大型管风琴以及电影中的爆炸和雷声音效可以达到这个频率，人声无法达到这个频段，一般 VOIP 通话建议调至z最低，去除直流干扰，将信号能量留给其他频段。
32HZ - 64HZ	重低音区	主要用于调整鼓和贝斯的下潜，音调感觉不明显，部分男低音音域可达这个频段。VOIP 通话人声调音一般建议调低，去除工频干扰，将能量留给其他频段。
64HZ - 125HZ	低音区	大部分管弦乐器的基频的范围，也决定打击乐的力度。
125HZ - 250HZ	低音区	人声的基频的范围，决定人声音调感知，过重会导致声音浑浊。
250HZ - 500HZ	中低音区	包含人声音色的重要的低次谐波所在频段，可用于调整男声音色，适当增强这一段使人声音色温暖，厚重；增强过度声音变浑浊。
500HZ - 1KHZ	中音区	调整女声音色，使其更饱满，调整太高会导致鼻音重。手机等移动播放设备的共振峰处于这个频段，建议避免调整太高容易引起结构振动杂音。
1KHZ - 2KHZ	中音区	人耳较敏感区域，影响响度和临场感，可以适当增强。
2KHZ - 4KHZ	中高音区	人耳最敏感的区域，提升这一判断可以提高声音的响度，提升语音可懂度相关，调整太高会导致齿音过重。
4KHZ - 8KHZ	高音区	表现镲等高频乐器，已经弦乐器摩擦音等声音细节。决定人声的高频细节，如唇齿音，摩擦音等，一般不建议提升。
8KHZ - 16KHZ	超高音和超声区	乐器的高频泛音，调整对语音作用不大。

具体调整也可以参考下图：


Excess
Association
Deficiency
Boomy
Muddy
Honky
Nasal
Boxy
Body
Thickness
Warmth
Solidity
Color
Power
Fattness
Natural tone
Rumble
Thin
Anemic
Wimpy
20
50
100
500
1 k
Frequency (Hz)

实时 K 歌功能

最近更新时间：2023-04-11 13:00:56

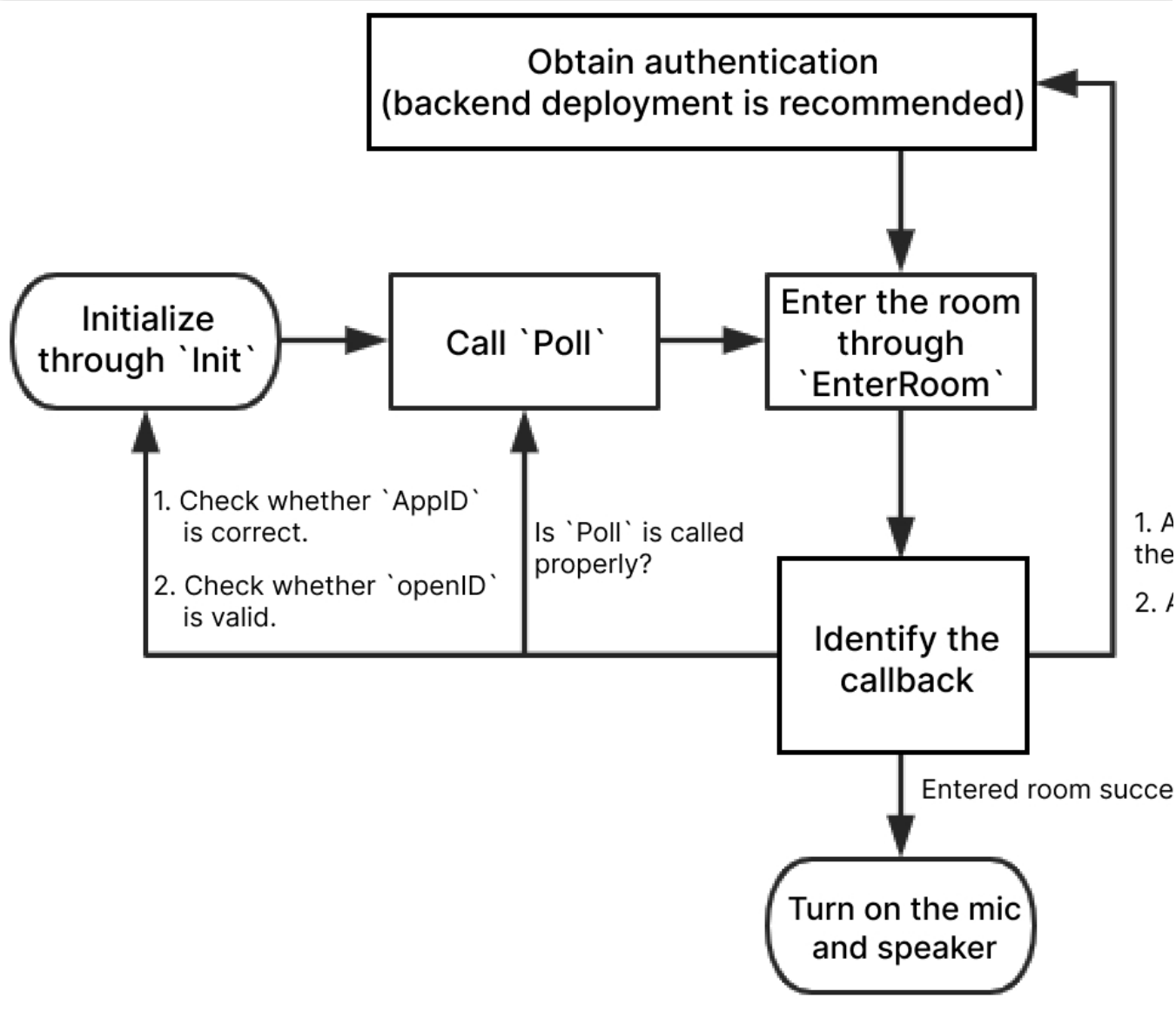
为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍游戏多媒体引擎实时语音 K 歌功能的接入技术文档。

使用前提

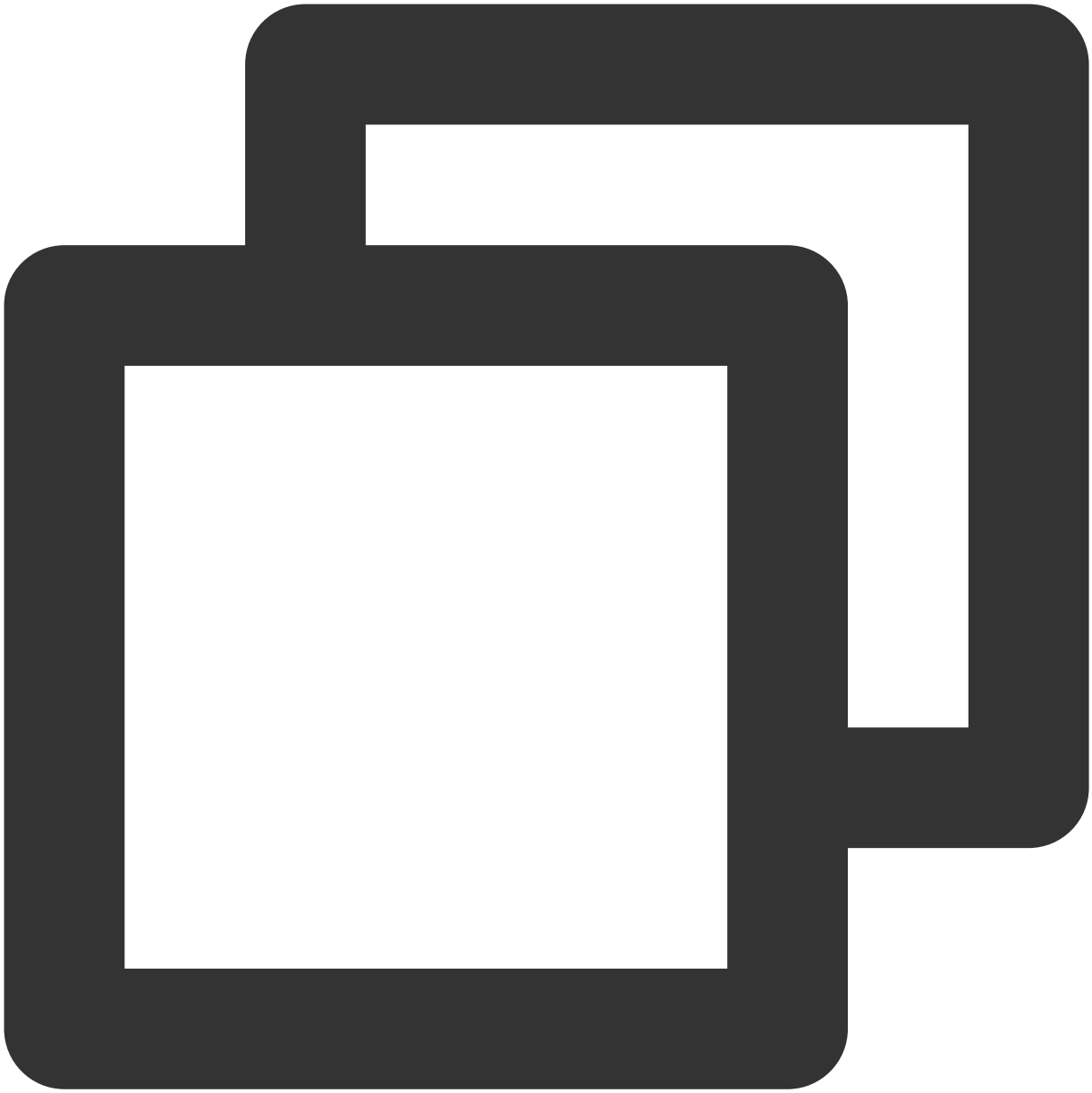
如需使用实时语音 K 歌功能，需要在接入 GME SDK 且能在进行实时语音通话的情况下，才可以使用此功能。
进房时候需要填入房间类型参数，推荐使用 RoomType = 2 进入房间（请使用2或者3）。
使用中如果有错误码提示，可参考 错误码文档 进行解决。

流程图

进房流程参考图如下：


Obtain authentication
(backend deployment is recommended)
Initialize through `Init`
Call `Poll`
Enter the room through `EnterRoom`
1. Check whether `AppID` is correct.
2. Check whether `openID` is valid.
Is `Poll` is called properly?
Identify the callback
Entered room succe
Turn on the mic and speaker

实时语音进房接口：



```
ITMGContext.GetInstance(this).Init(String.valueOf(mAppId), mUserId);//初始化sdk
ITMGContext.GetInstance(this).SetTMGDelegate(new MyDelegate());//设置代理类，用来接受各
EnginePollHelper.createEnginePollHelper();//周期性调用Poll函数，触发回调

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

说明

详细调用流程和接口详情请参考 各端 SDK 接口文档。

K 歌接口获取

1. 您需要在 下载指引 中下载标准 SDK 文件。
2. 根据不同的平台，下载 K 歌功能接口并导入相应接口文件。

说明

此功能支持 mp3、ogg 一共两种格式。
如果音乐文件为 ogg 格式，请 单击下载ogg动态库（iOS 端已包含 ogg 动态库，无需额外导入）。
如果音乐文件为 mp3 格式，请 单击下载mp3动态库（只有 iOS 平台才需导入此动态库，其他平台无需额外导入）。

Android 端配置

Android 对应的接口，已经包含在标准 jar 包内，不需要下载额外的接口文件。

iOS 端配置

1. 在 iOS 端使用 K 歌功能，需要将相关动态库引入工程中，单击下载 mp3 动态库。
2. 将下载好的文件引入到工程文件中。并在 Link Binary With Libraries 中添加此动态库。
3. 将头文件 TMGEngine_adv.h 加入工程中，与其他 SDK 头文件同目录下。

Windows 端配置

Windows 端使用 K 歌功能，需要下载头文件后，将头文件 tmg_sdk_adv.h、tmg_type_adv.h 导入工程中。

Unity 引擎配置

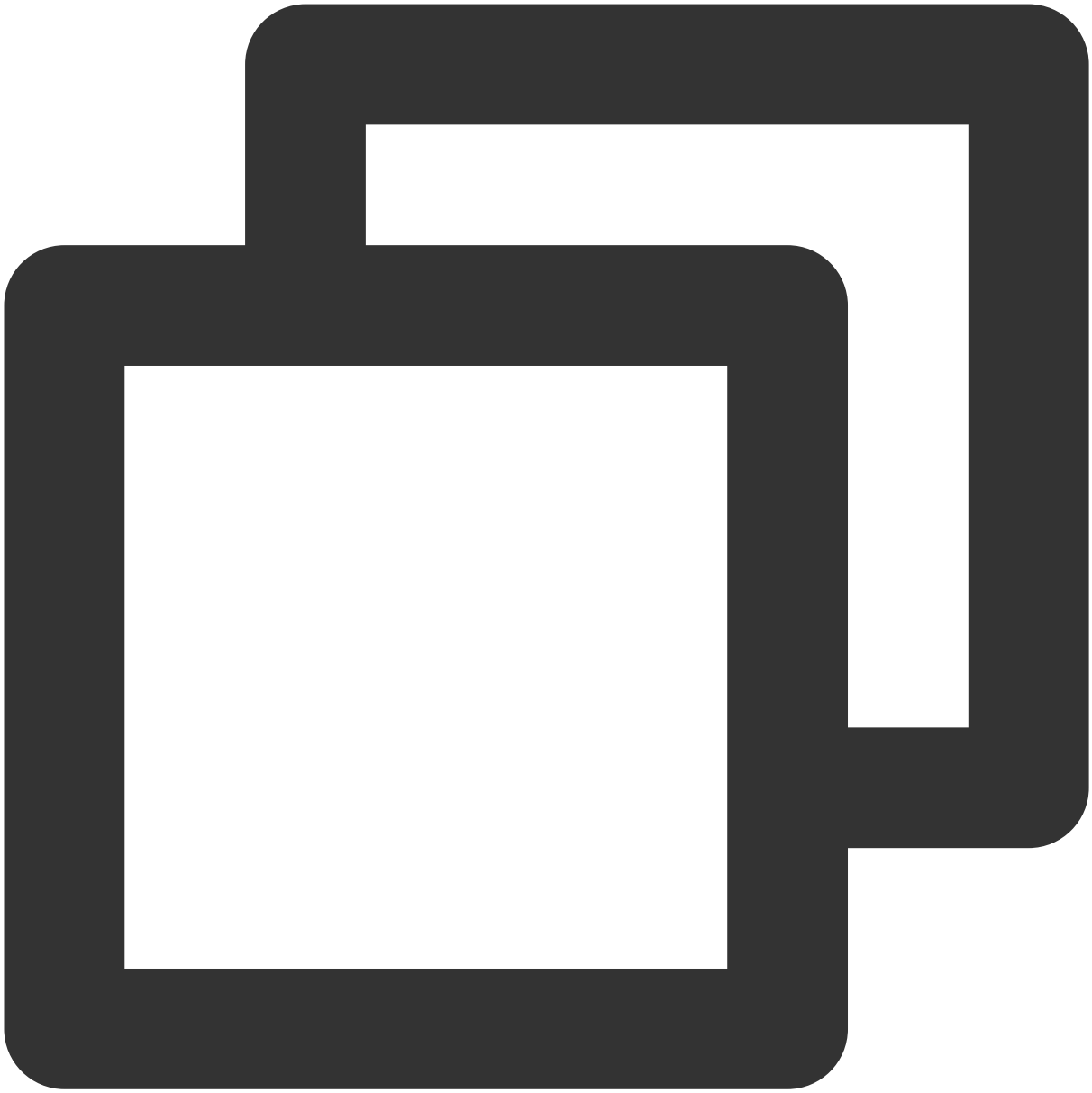
在 Unity 引擎中使用 K 歌功能，需要下载头文件，将 Unity 文件夹下的代码文件 TMGEngine_Adv.cs、ITMGEngine_Adv.cs 拷贝后导入工程中。
如需导出 iOS 平台，请参考上文进行 mp3 动态库导入。

录制相关接口

开始录制

调用 StartRecord 接口开始录制。录制完成会有回调函数，需要监听 ITMG_MAIN_EVENT_TYPE_RECORD_COMPLETED。
录制时候请保证麦克风已经打开（设备及上行都需要打开），另外保证文件路径是可访问的，SDK 不会主动创建文件夹。

函数原型

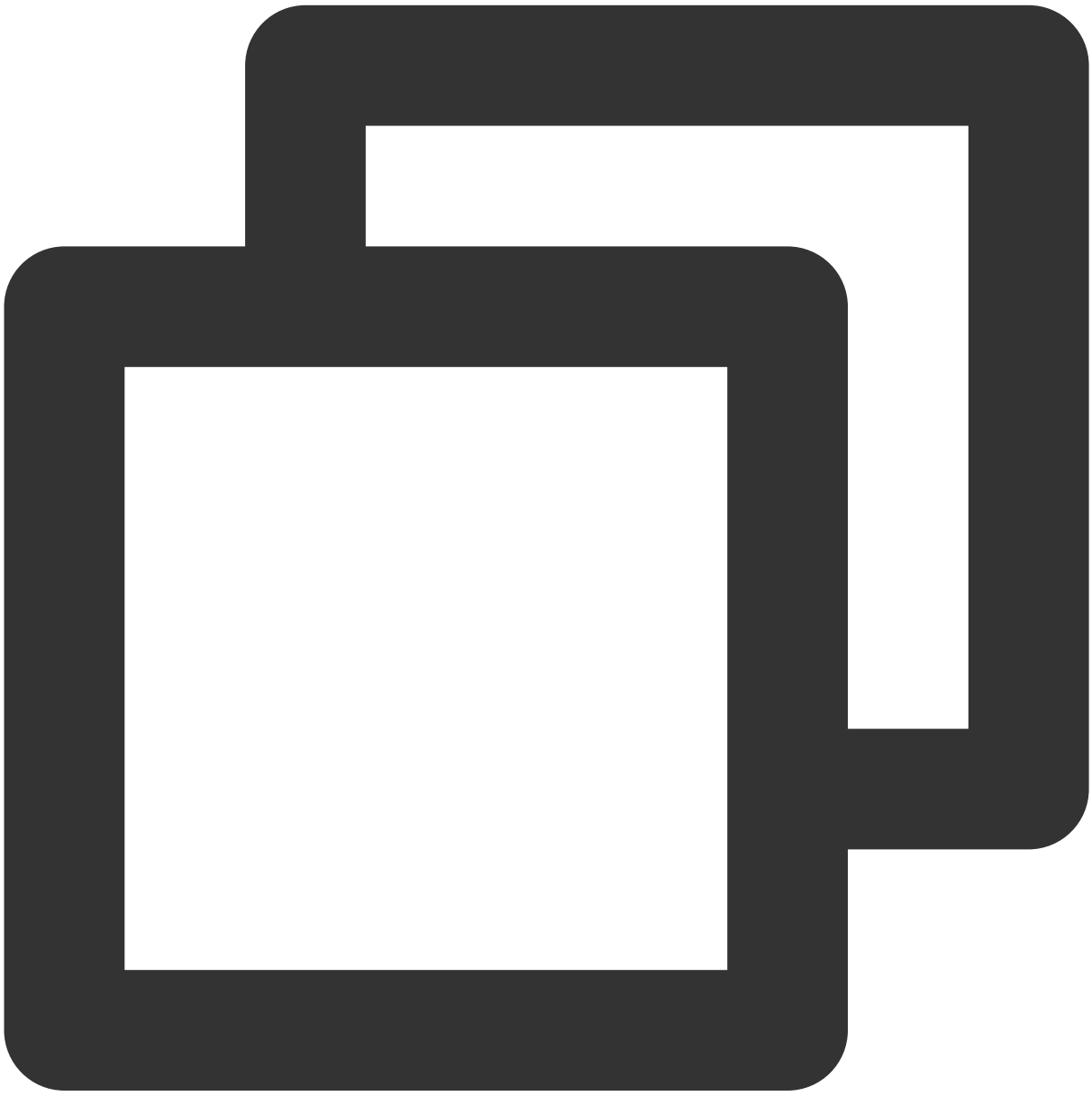


```
int StartRecord(int type, String dstFile, String accMixFile, String accPlayFile)
```

参数	类型	意义
type	int	K歌场景下，此参数传 ITMG_AUDIO_RECORDING_KTV。如果是纯录制 MP3 文件，请使用 ITMG_AUDIO_RECORDING_SELF
dstFile	String	目标文件路径，用于保存录制完成的音乐。
accMixFile	String	一般为没有原声的伴奏，用于和人声合成音乐文件。

accPlayFile	String	用于播放的音乐文件，正常情况下与 accMixFile 为同一个文件。但在用户不熟悉歌曲时，可以填入带原唱的音乐文件路径，此时播放内容为带原唱的音乐，而合成为不带原声的伴奏。

示例代码



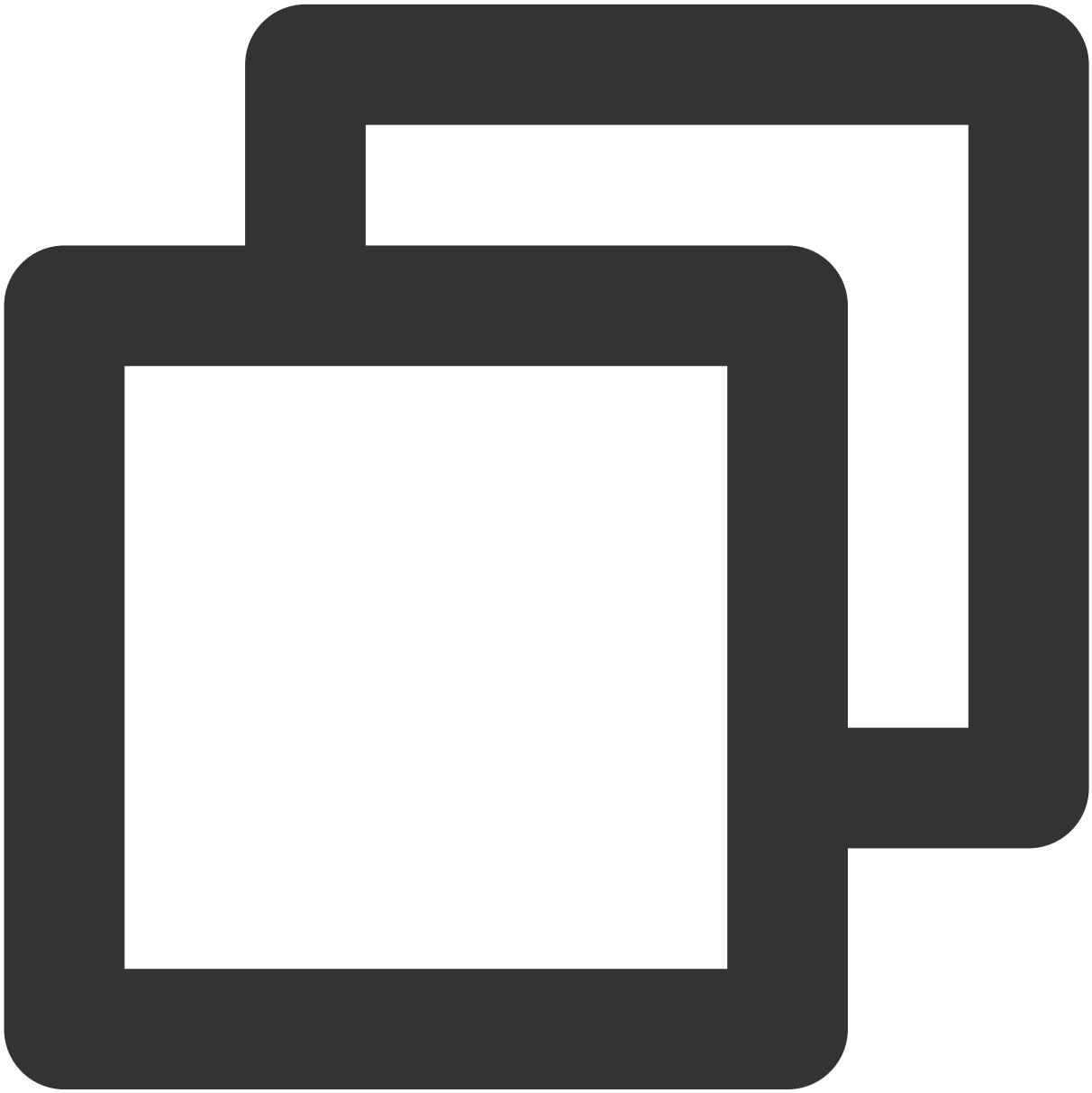
```
//Android
ITMGAudioRecordCtrl.GetInstance().StartRecord(ITMGAudioRecordCtrl.ITMG_AUDIO_RECORD
//iOS
#import "GMESDK/TMGEngine_adv.h"
```

```
[[ITMGAudioRecordCtrl GetInstance]StartPreview]
```

停止录制

调用 StopRecord 接口停止录制。

函数原型

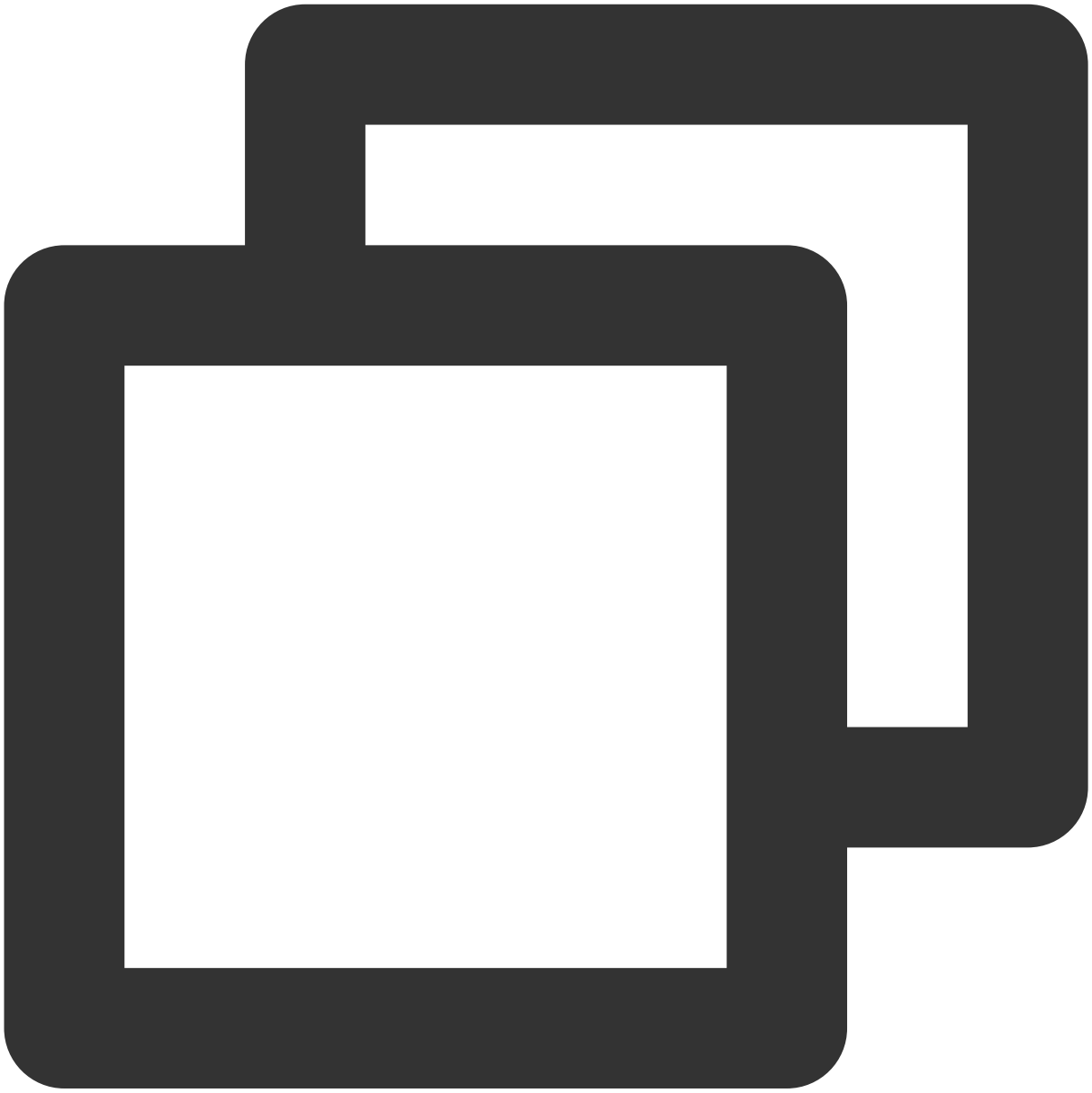


```
int StopRecord()
```

暂停录制

调用 PauseRecord 接口暂停录制。

函数原型

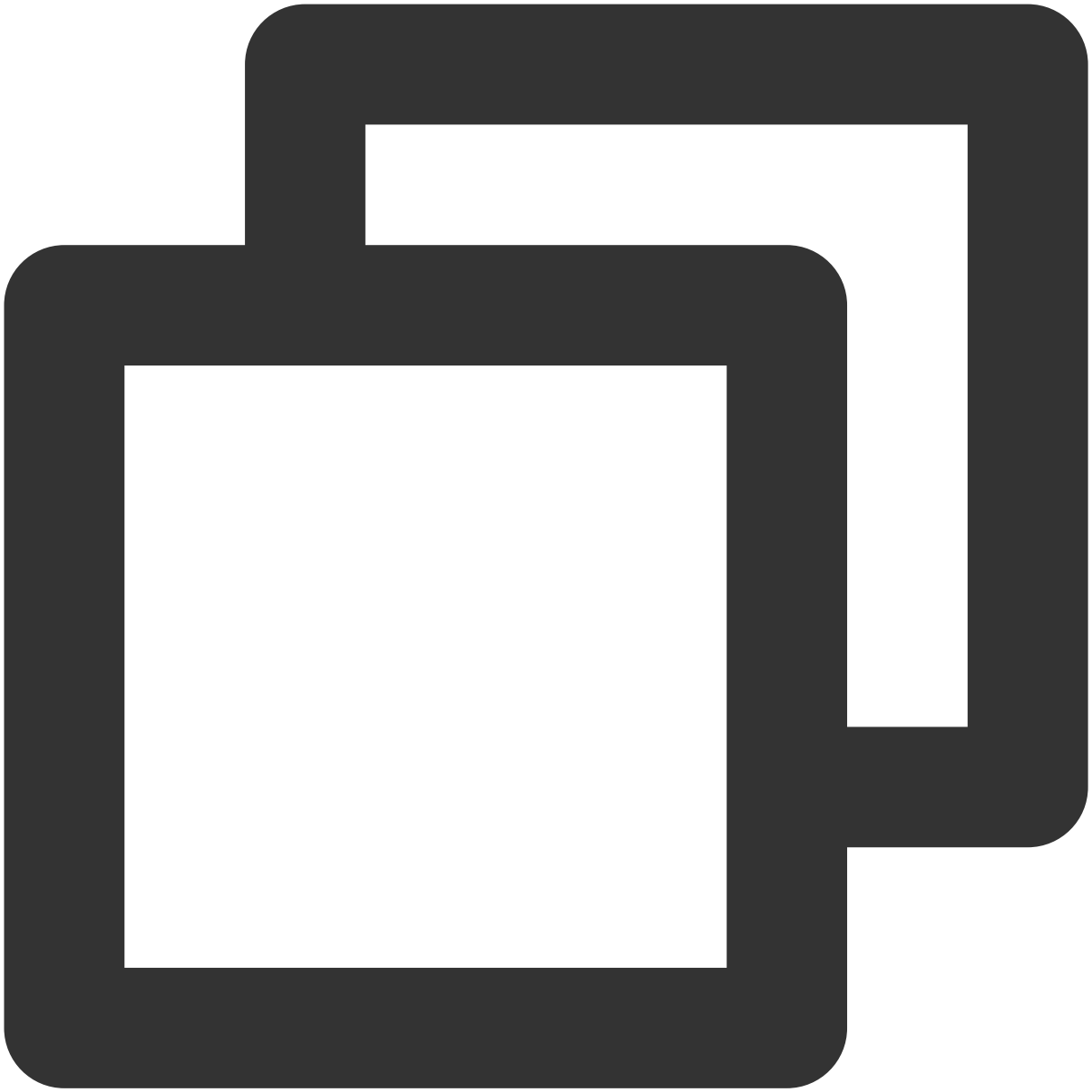


```
int PauseRecord()
```

恢复录制

调用 ResumeRecord 接口恢复录制。

函数原型



```
int ResumeRecord()
```

录制回调

ITMG_MAIN_EVENT_TYPE_RECORD_COMPLETED 录制完成的回调。伴奏播放结束或者调用 StopRecord 触发此回调。

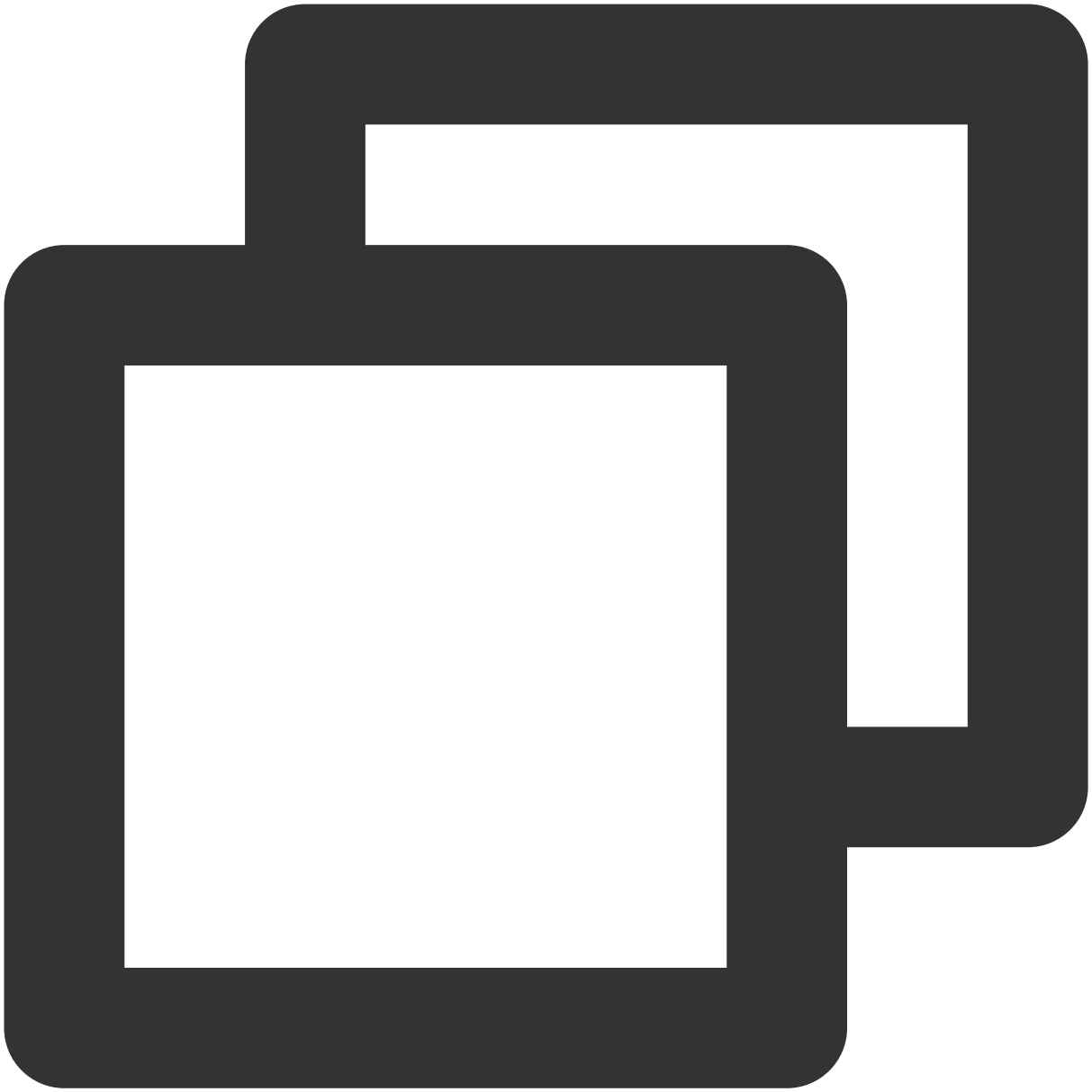
回调参数

参数	类型	意义
result	int	录制结果，0为成功。如果有其他错误码，可参考 错误码文档 进行解决。
filepath	String	目标文件的路径，StartRecord 传入的参数 dstFile。
duration	String	录制文件的长度，单位为毫秒。

设置播放文件

在调用 StartRecord 接口进行录制的时候，会设置播放的音乐文件。如果想要重新设置，可调用此接口重新设置播放文件。一般用于在原唱和纯音伴奏之间切换。

函数原型



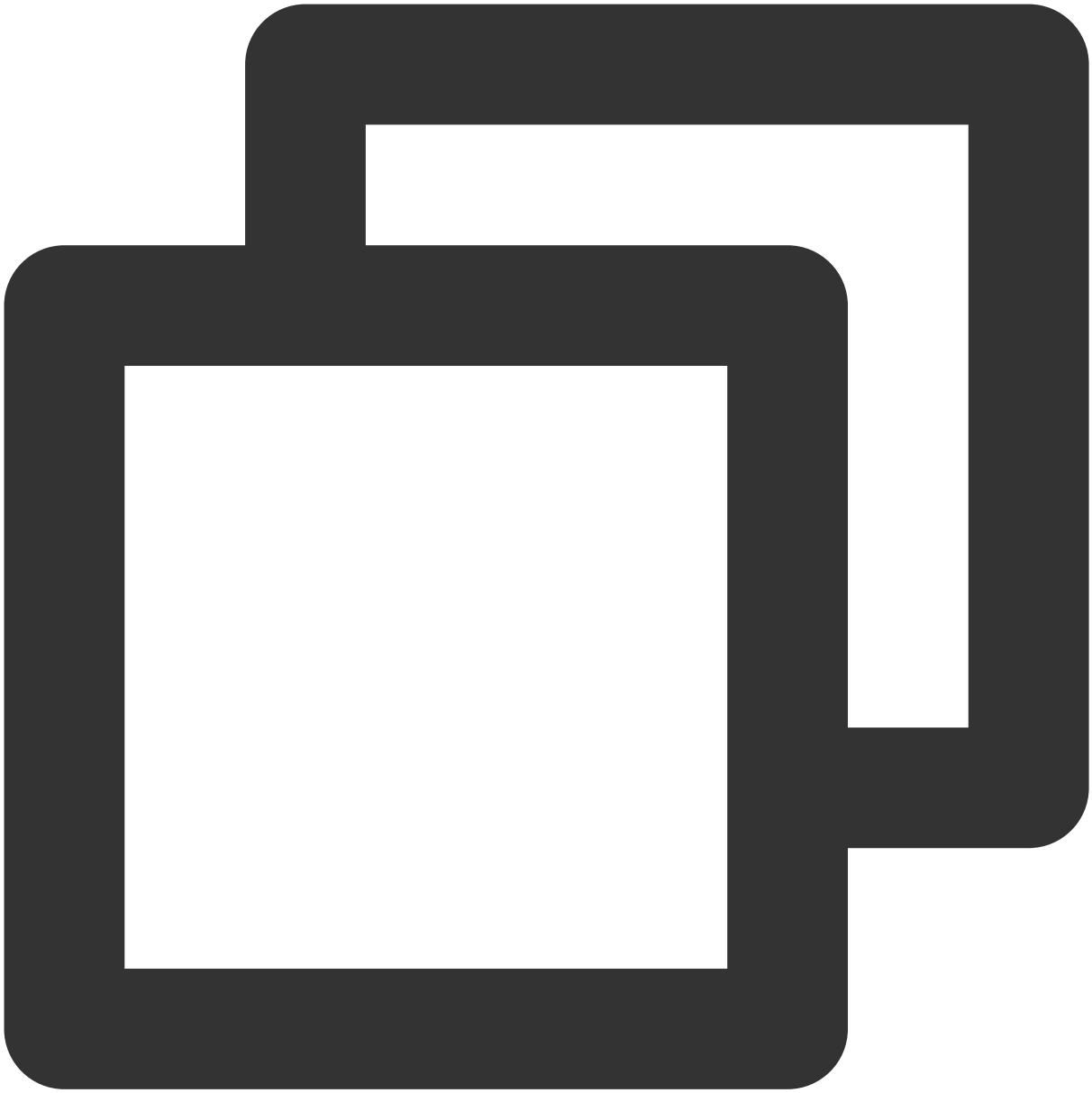
```
int SetAccompanyFile(String accPlayFile)
```

参数	类型	意义
accPlayFile	String	用于播放的音乐文件。

获取伴奏时长

调用此参数获取伴奏文件 accMixFile 的时长，返回的时长单位为毫秒。

函数原型

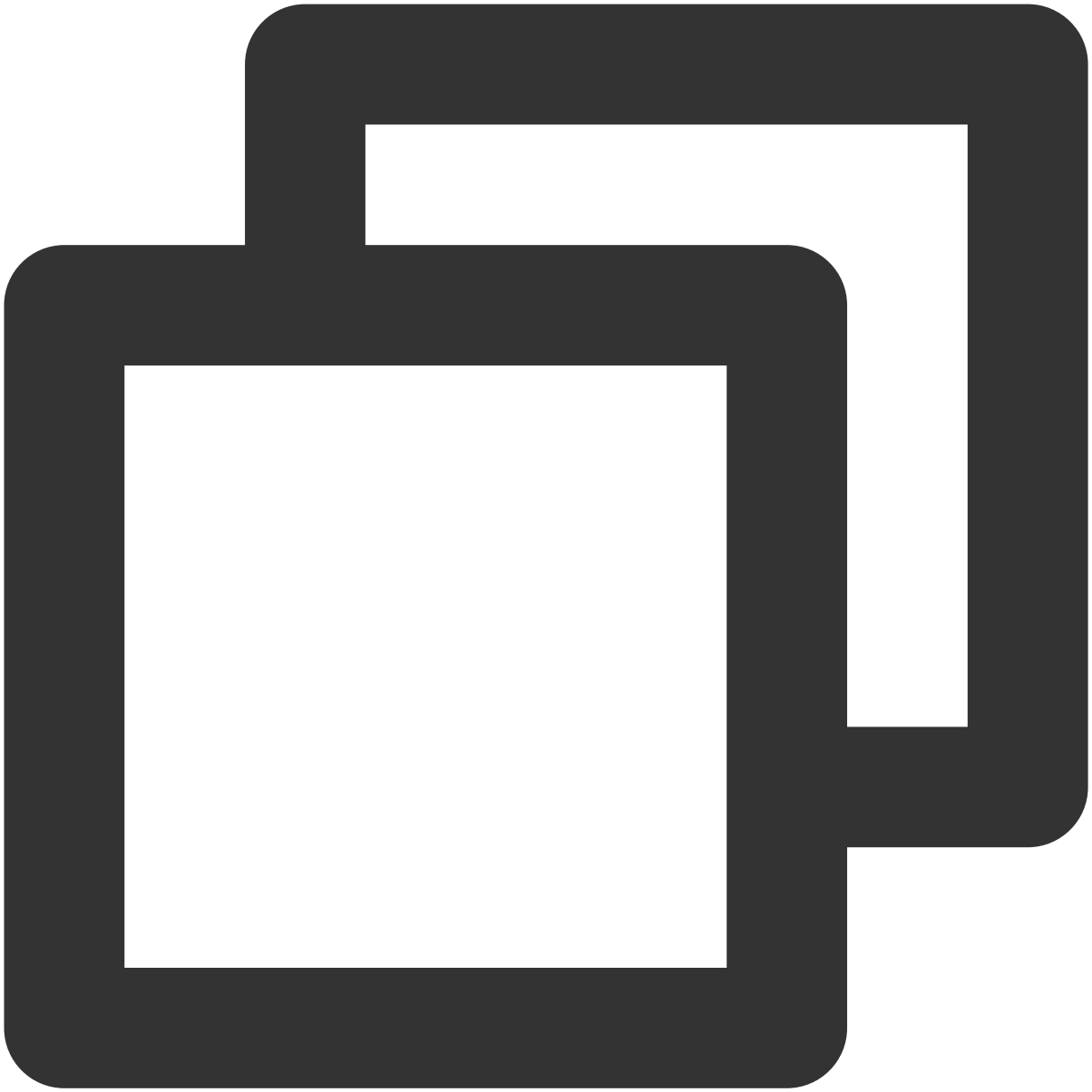


```
int GetAccompanyTotalTimeByMs()
```

获取当前录制时长

调用此参数获取当前录制时长，返回的时长单位为毫秒。

函数原型

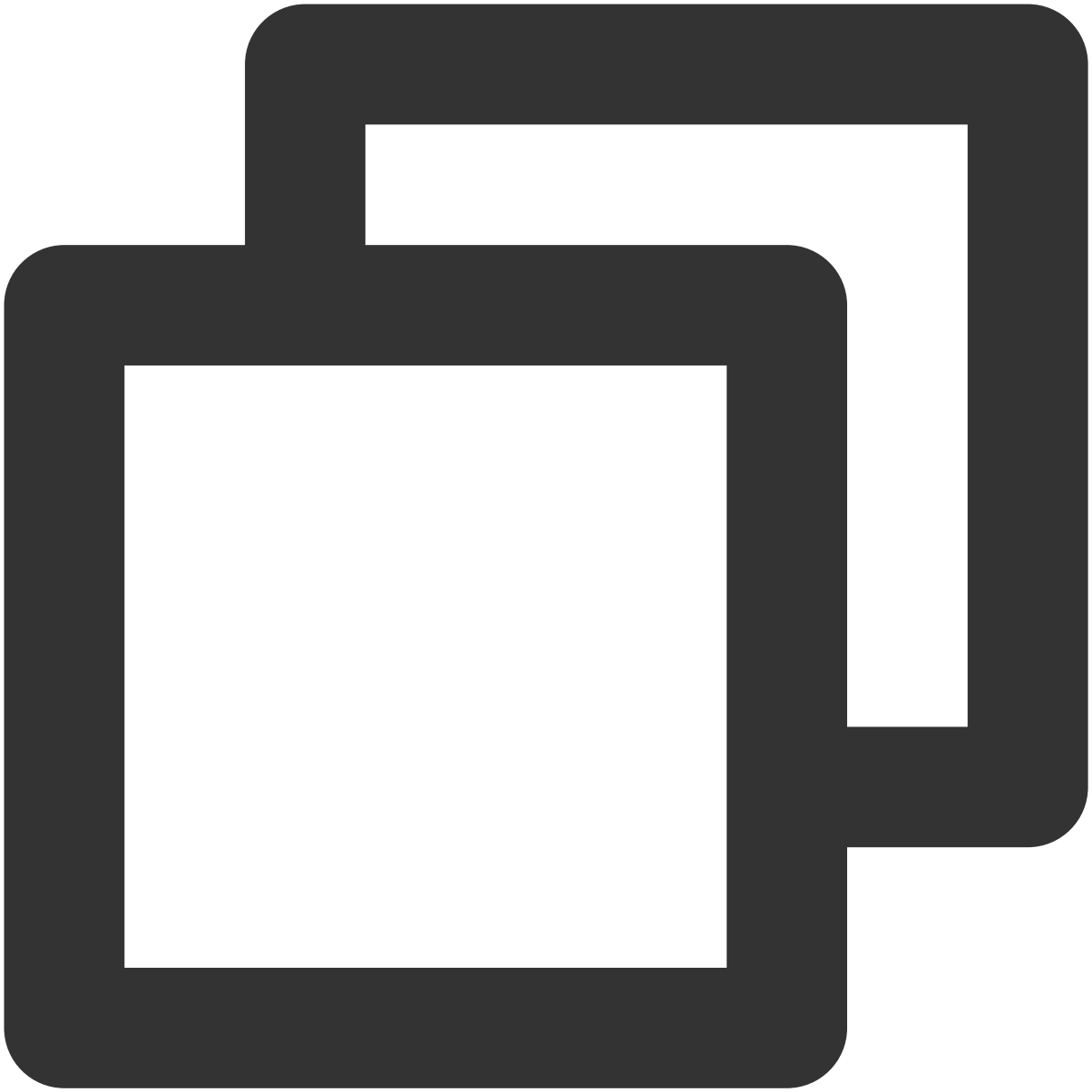


```
int GetRecordTimeByMs()
```

录制跳转

将录制时间跳转到指定时刻。如果参数比当前时间靠前，则重复的地方重新录制；相比当前时间靠后，则用静音数据填充没有录制的部分。

函数原型



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

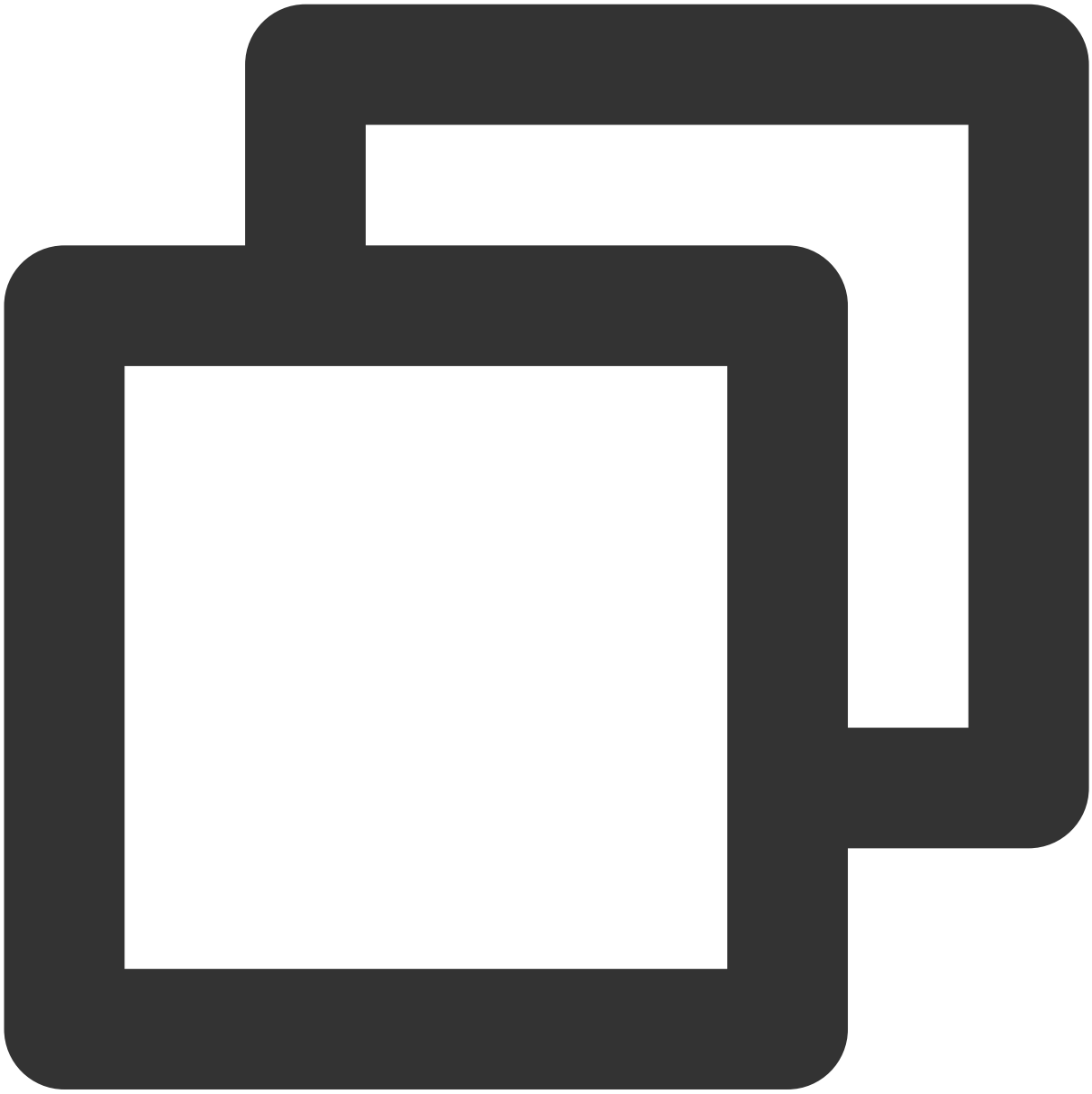
参数	类型	意义
timeMs	int	跳转的时刻，单位为毫秒。

K 歌文件预览

获取录制文件的时长

调用此参数获取录制文件的时长。

函数原型

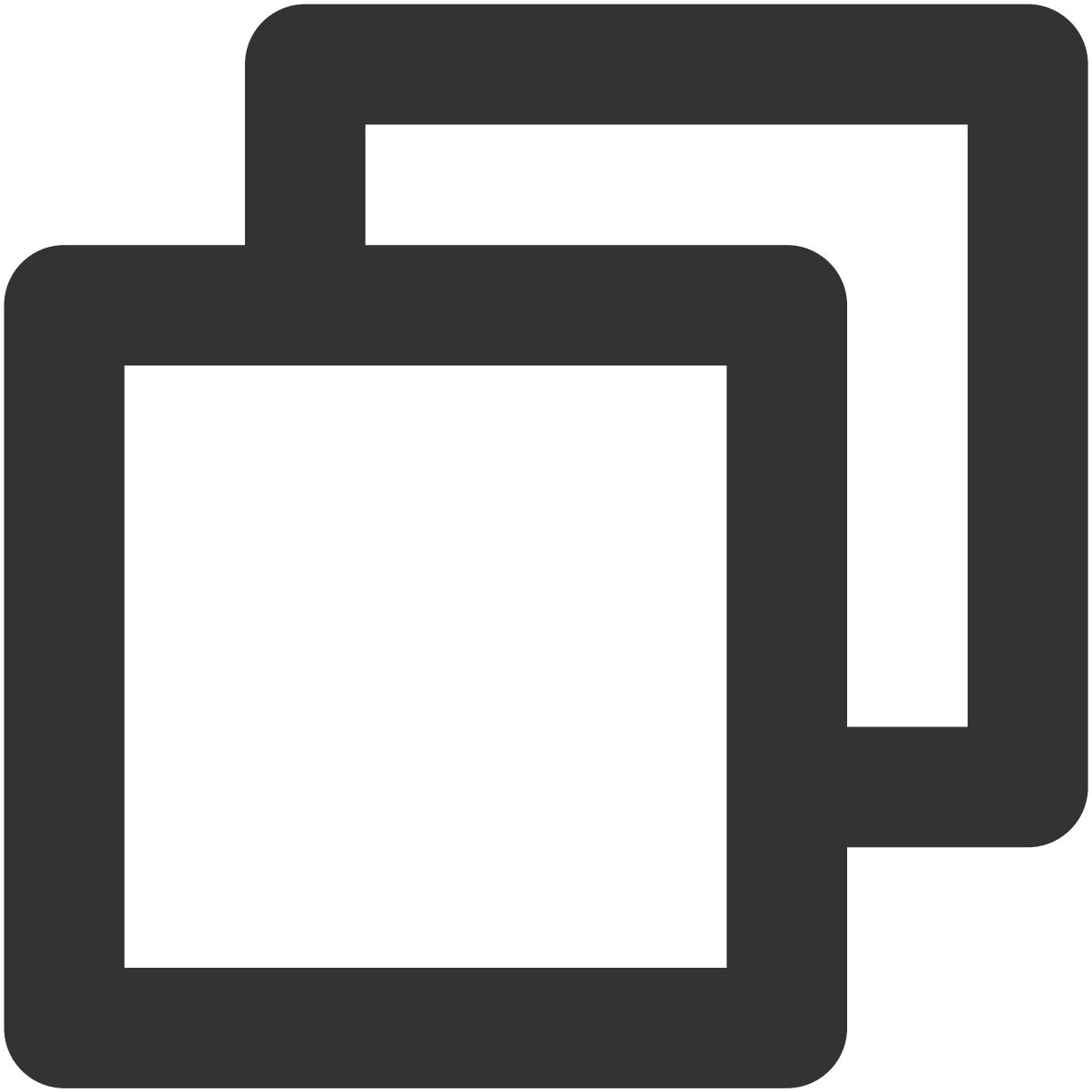


```
int GetRecordFileDurationByMs()
```

开始预览录制文件

调用此参数开始预览录制文件。

函数原型

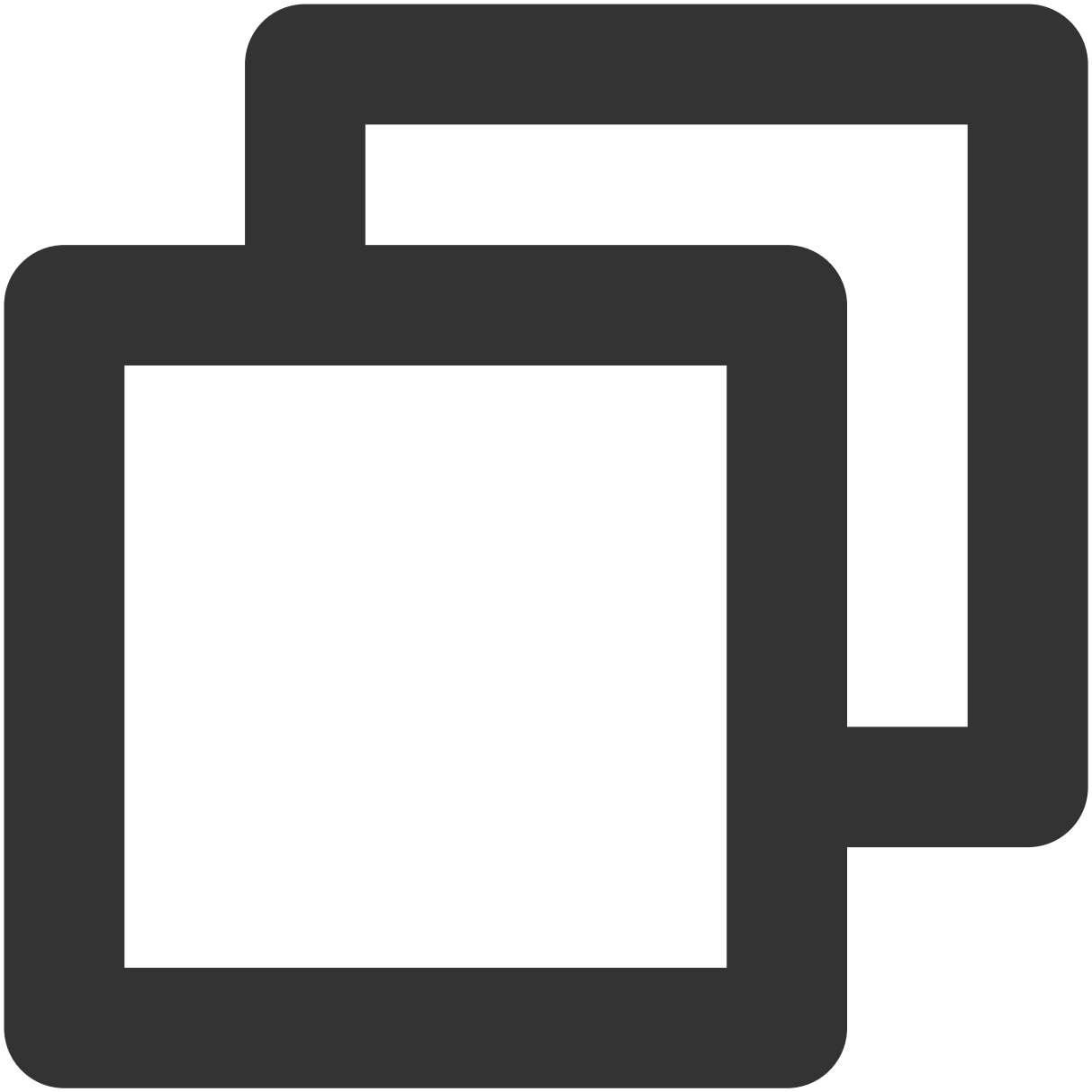


```
int StartPreview()
```

停止预览录制文件

调用此参数停止预览录制文件。

函数原型

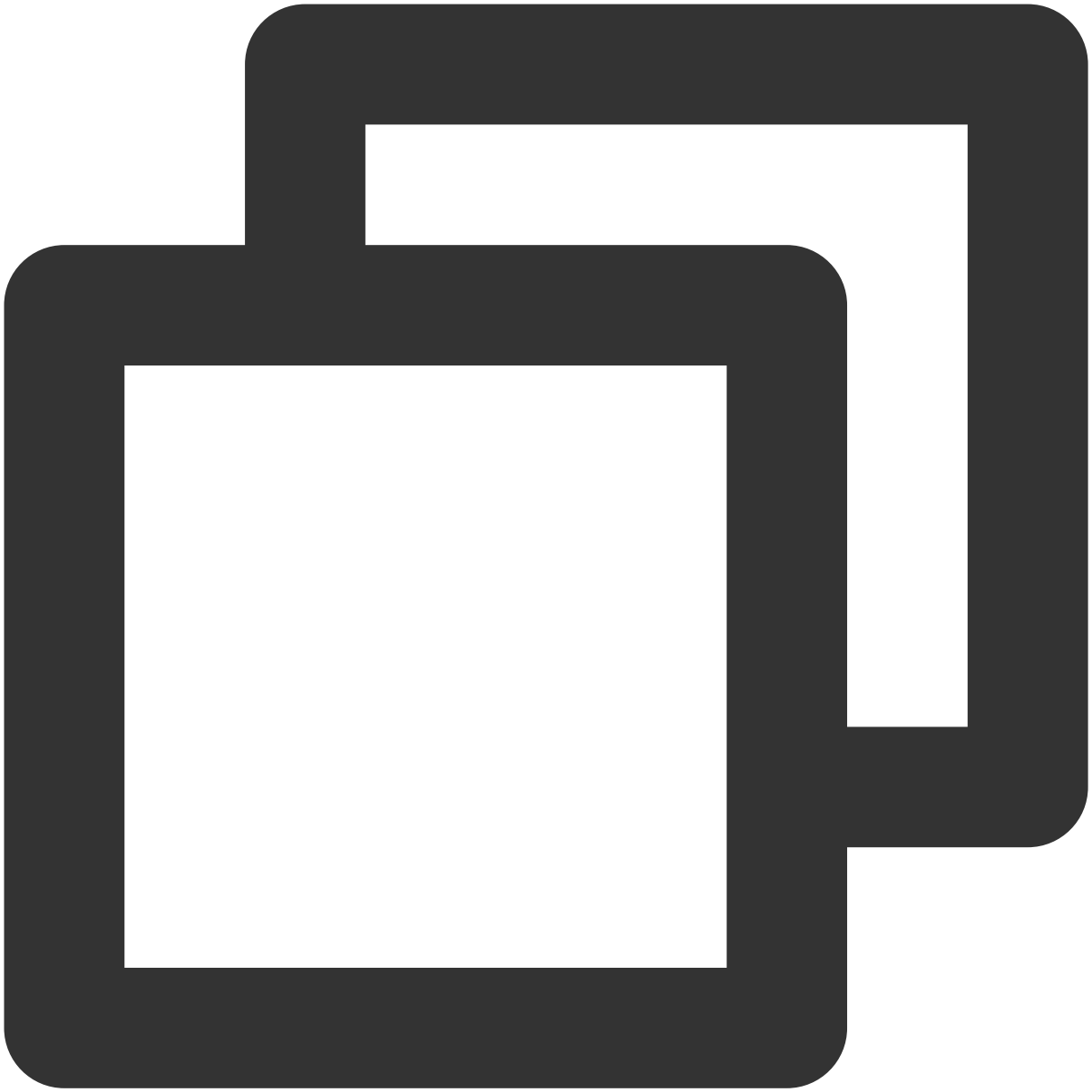


```
int StopPreview()
```

暂停预览录制文件

调用此参数暂停预览录制文件。

函数原型

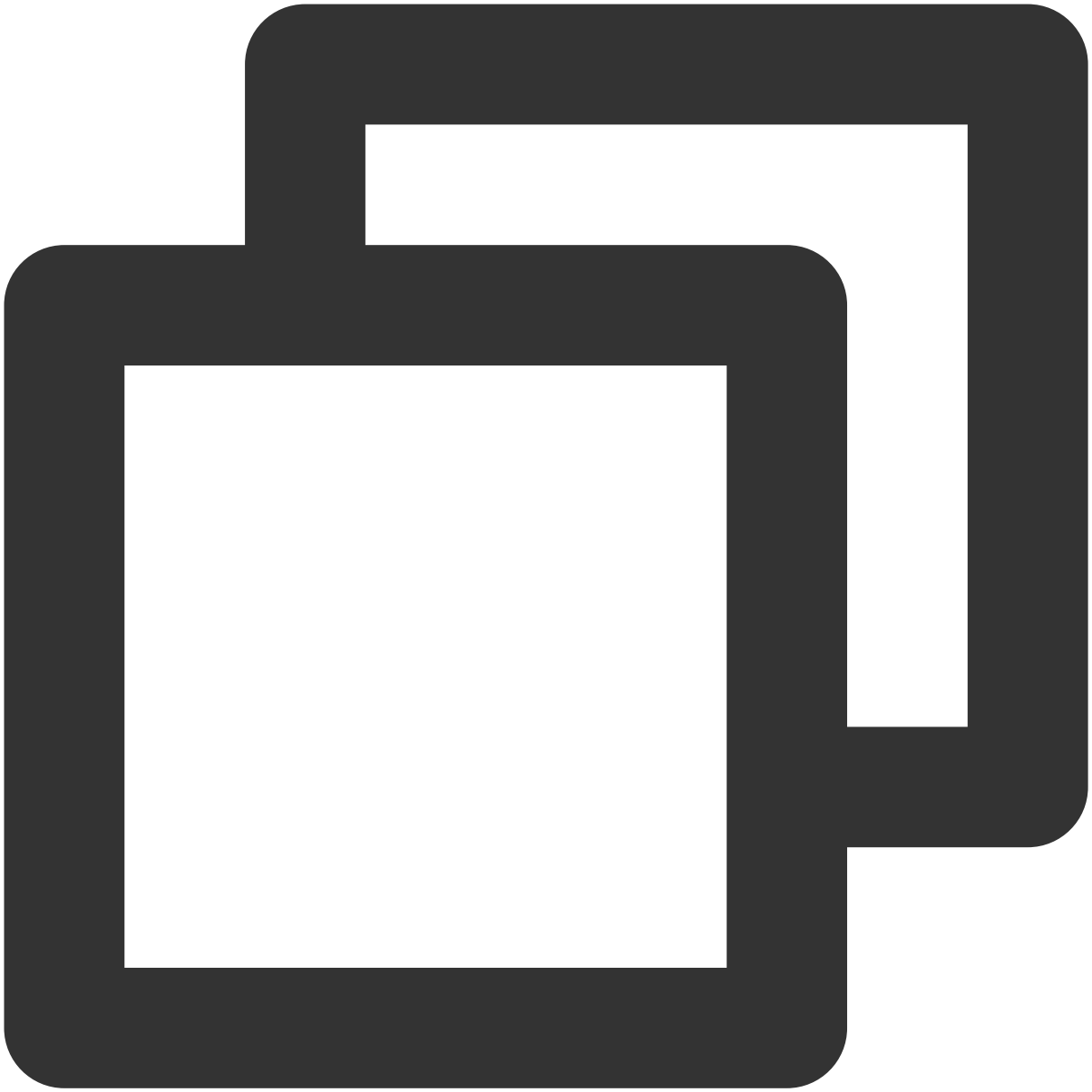


```
int PausePreview()
```

恢复预览录制文件

调用此参数恢复预览录制文件。

函数原型

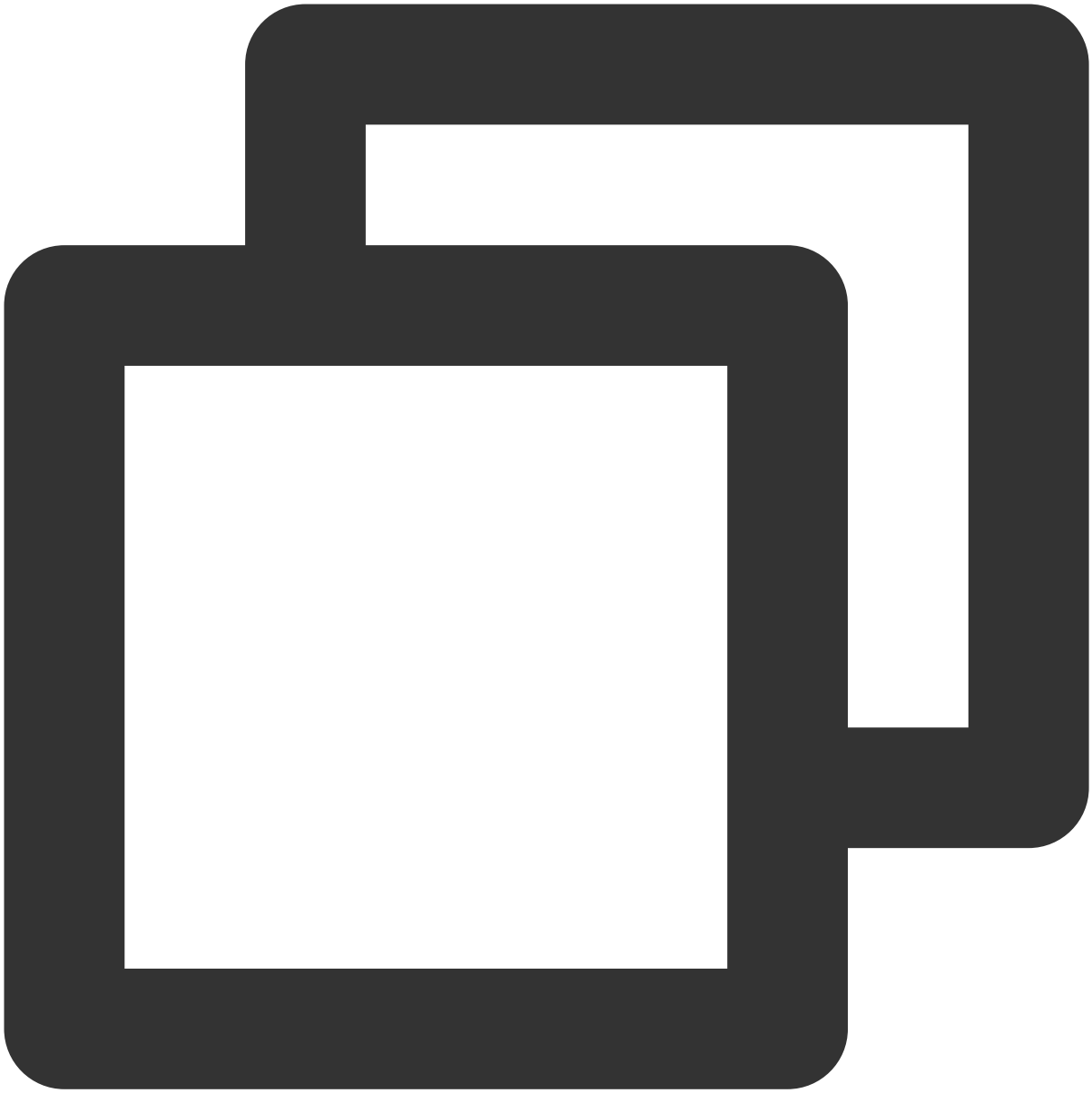


```
int ResumePreview()
```

设置当前预览的时间点

调用此参数设置当前预览的时间点。

函数原型



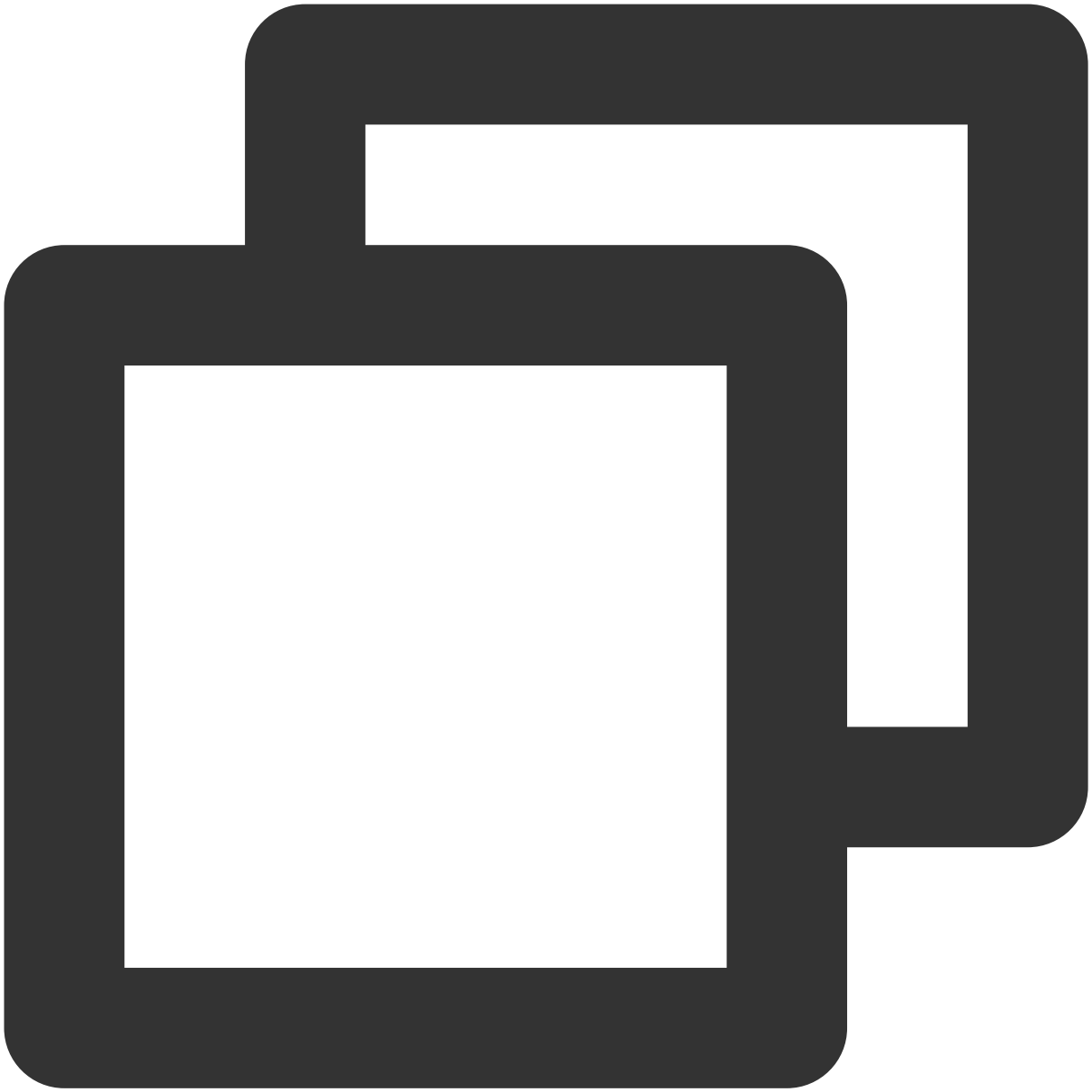
```
int SetPreviewTimeByMs(int time)
```

参数	类型	意义
time	int	预览文件的时间点，单位毫秒。

获取当前预览的时间点

调用此参数获取当前预览的时间点。

函数原型



```
int GetPreviewTimeByMs()
```

播放预览回调

ITMG_MAIN_EVENT_TYPE_RECORD_PREVIEW_COMPLETED 预览完成的回调。预览文件播放结束或者调用 StopPreview 接口触发

回调参数

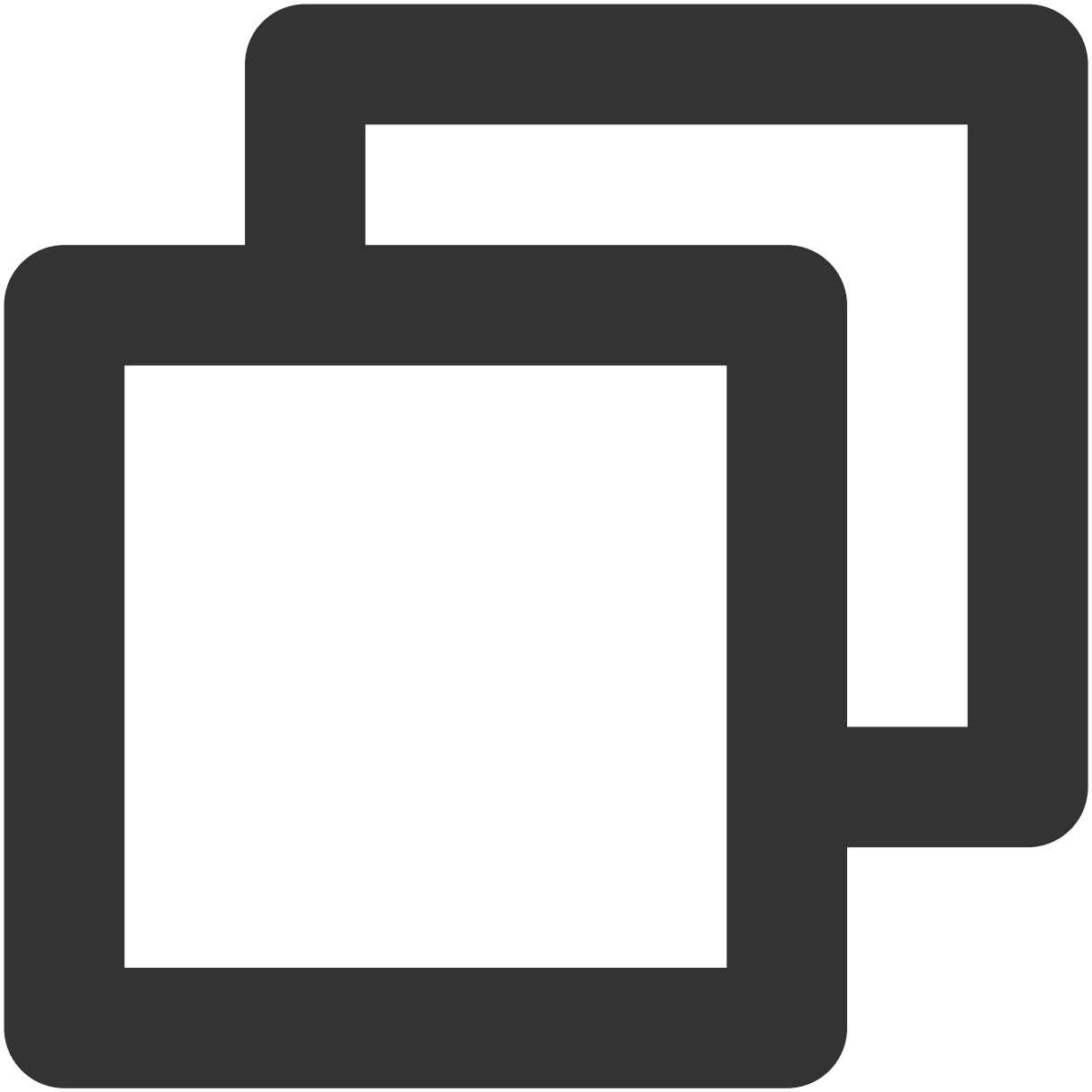
参数	类型	意义
result	int	播放结果，0为成功。

文件合成接口

合并文件

调用此参数将录制好的人声和伴奏合并成一个文件。

函数原型

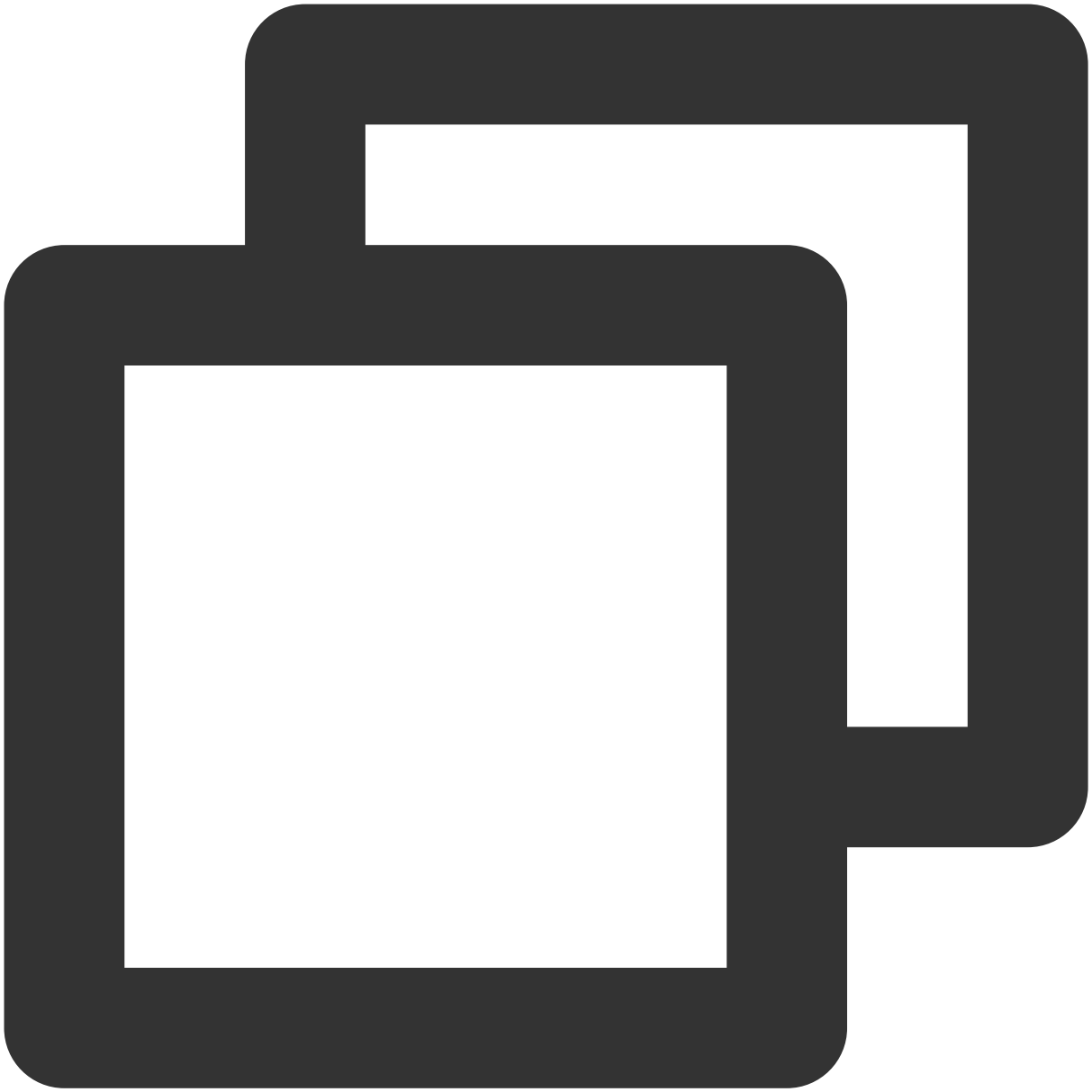


```
int MixRecordFile();
```

取消合并

调用此参数将取消合并操作。

函数原型



```
int CancelMixRecordFile();
```

合并文件回调

ITMG_MAIN_EVENT_TYPE_RECORD_MIX_COMPLETED 预览完成的回调。预览文件播放结束或者调用 StopPreview 接口触发。

回调参数

参数	类型	意义

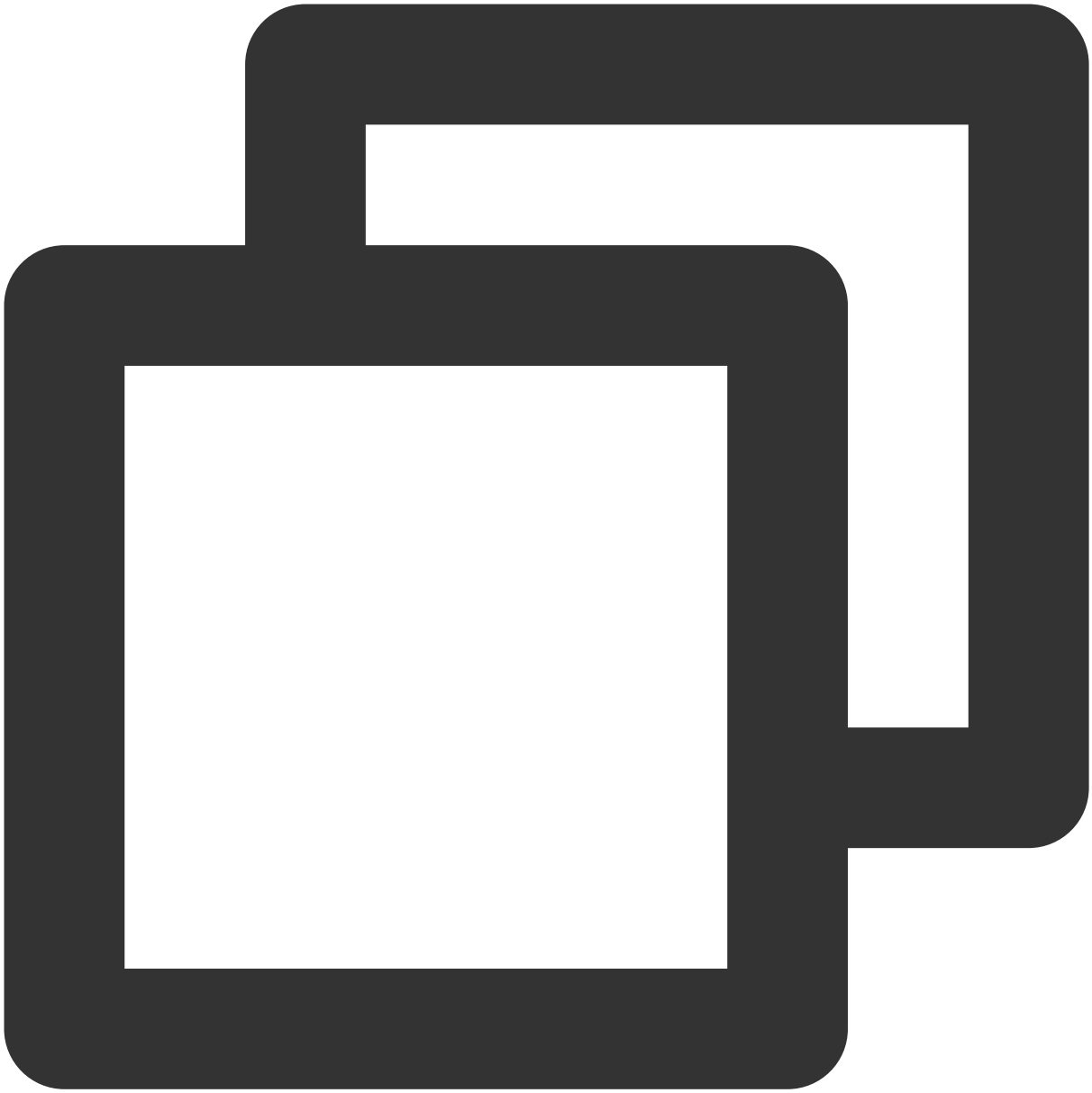
result	int	合成结果，0为成功。
filepath	String	目标文件的路径，由 StartRecord 接口中传入的 dstFile。
duration	String	录制文件的长度，单位为毫秒。

高级设置

设置伴奏缩放

调用此接口设置人声和伴奏的缩放比例。录制完成后可进行调整。

函数原型



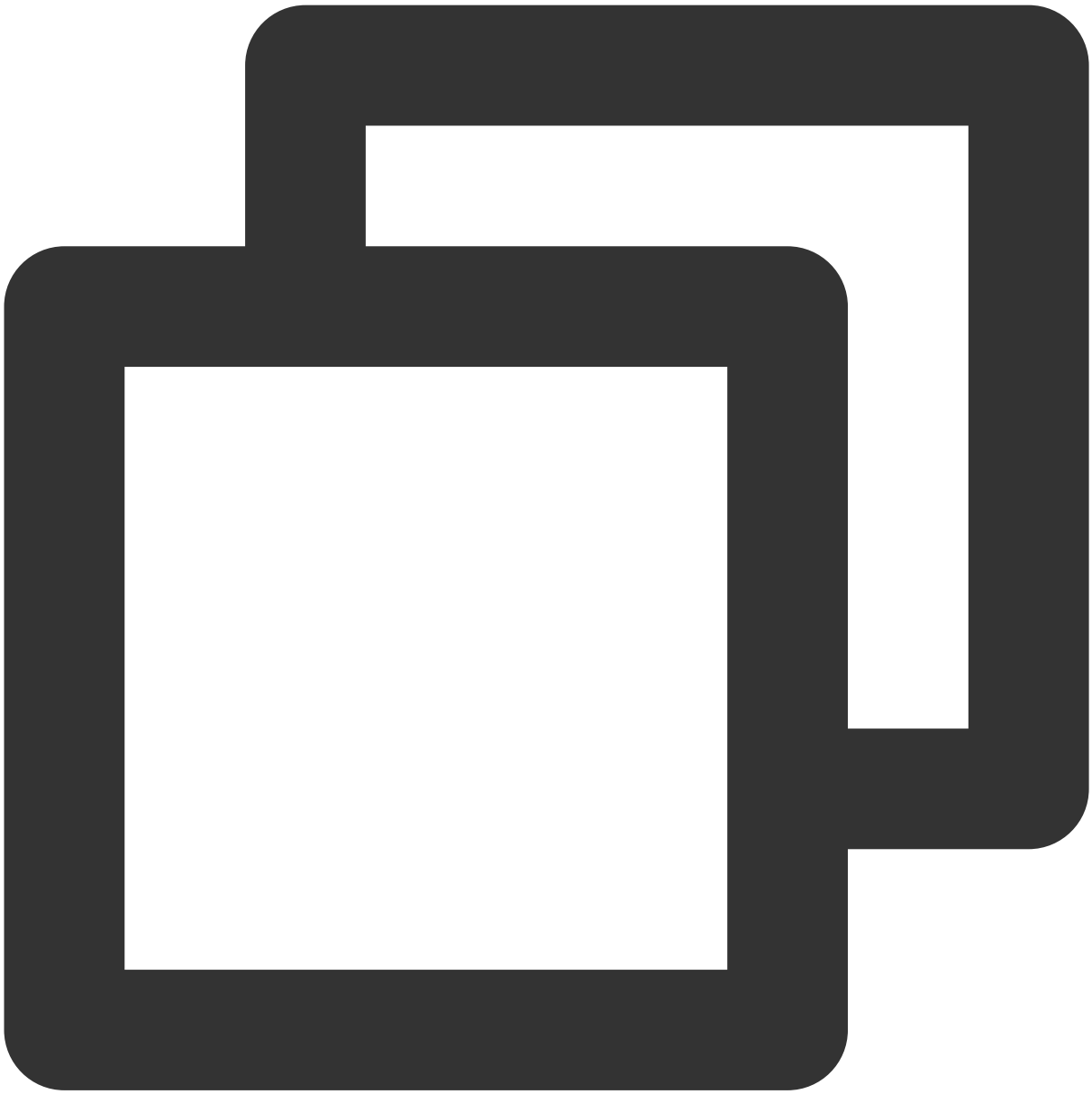
```
int SetMixWieghts(float mic, float acc)
```

参数	类型	意义
mic	float	人声的缩放比例，1.0为原来音量，小于1.0为缩小，大于1.0为放大，范围为0到2。
acc	float	伴奏的缩放比例，1.0为原来音量，小于1.0为缩小，大于1.0为放大，范围为0到2。

设置偏移量

调用此接口设置人声相对于伴奏的偏移，一般用于调整声音跟不上节拍的问题。录制完成后可进行调整。

函数原型



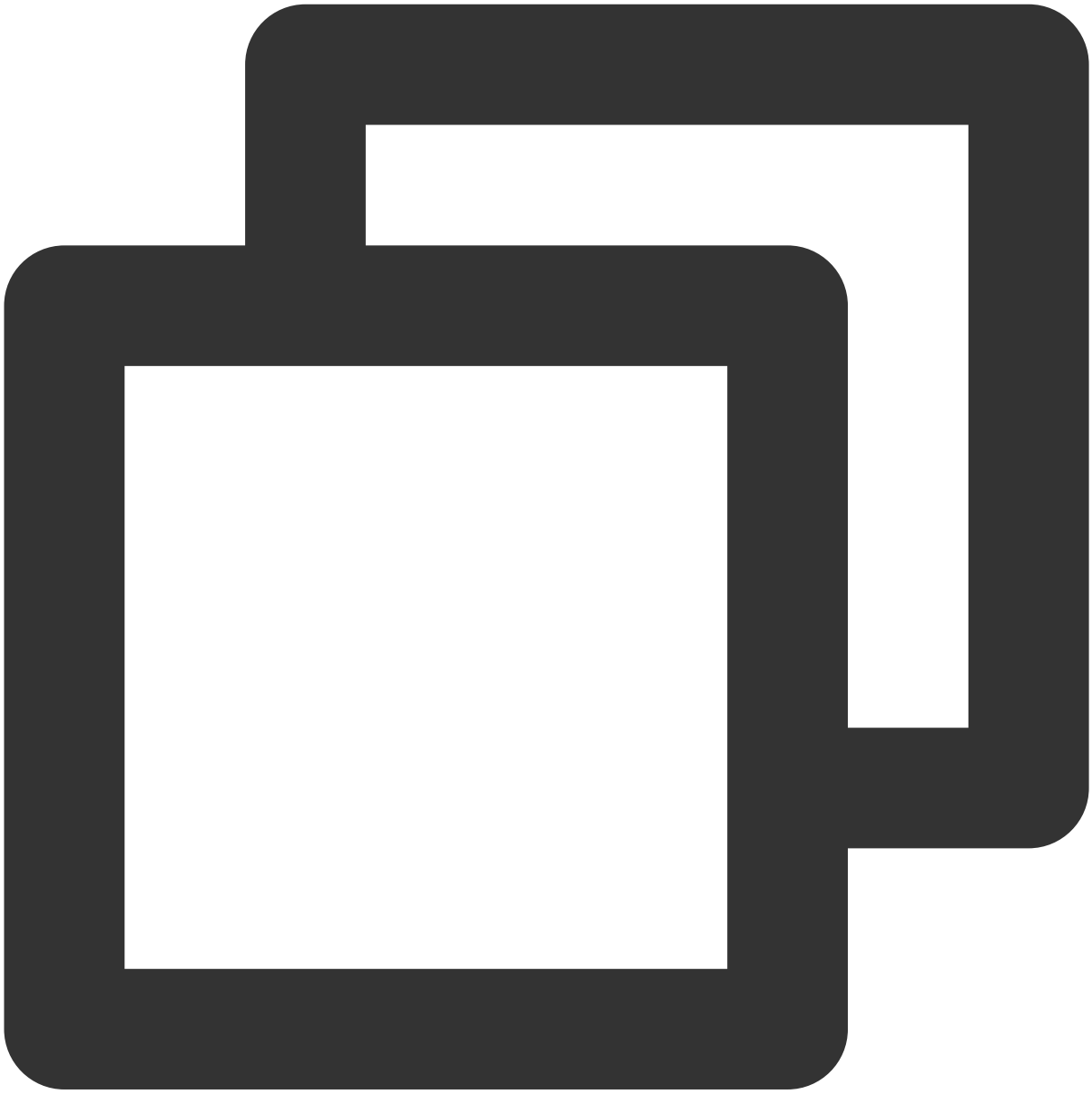
```
int AdjustAudioTimeByMs(int time)
```

参数	类型	意义
time	int	人声相对于伴奏的偏移时间，单位ms。大于0为向后移动，小于0为向前移动。

设置音效

设置音效类型，K 歌音效可参考 实时语音音效文档。录制完成后可进行调整，录制时也可使用。

函数原型



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

参数	类型	意义
type	int	此类型同等于实时语音音效中的K歌音效特效类型，具体可参考 K 歌音效特效。

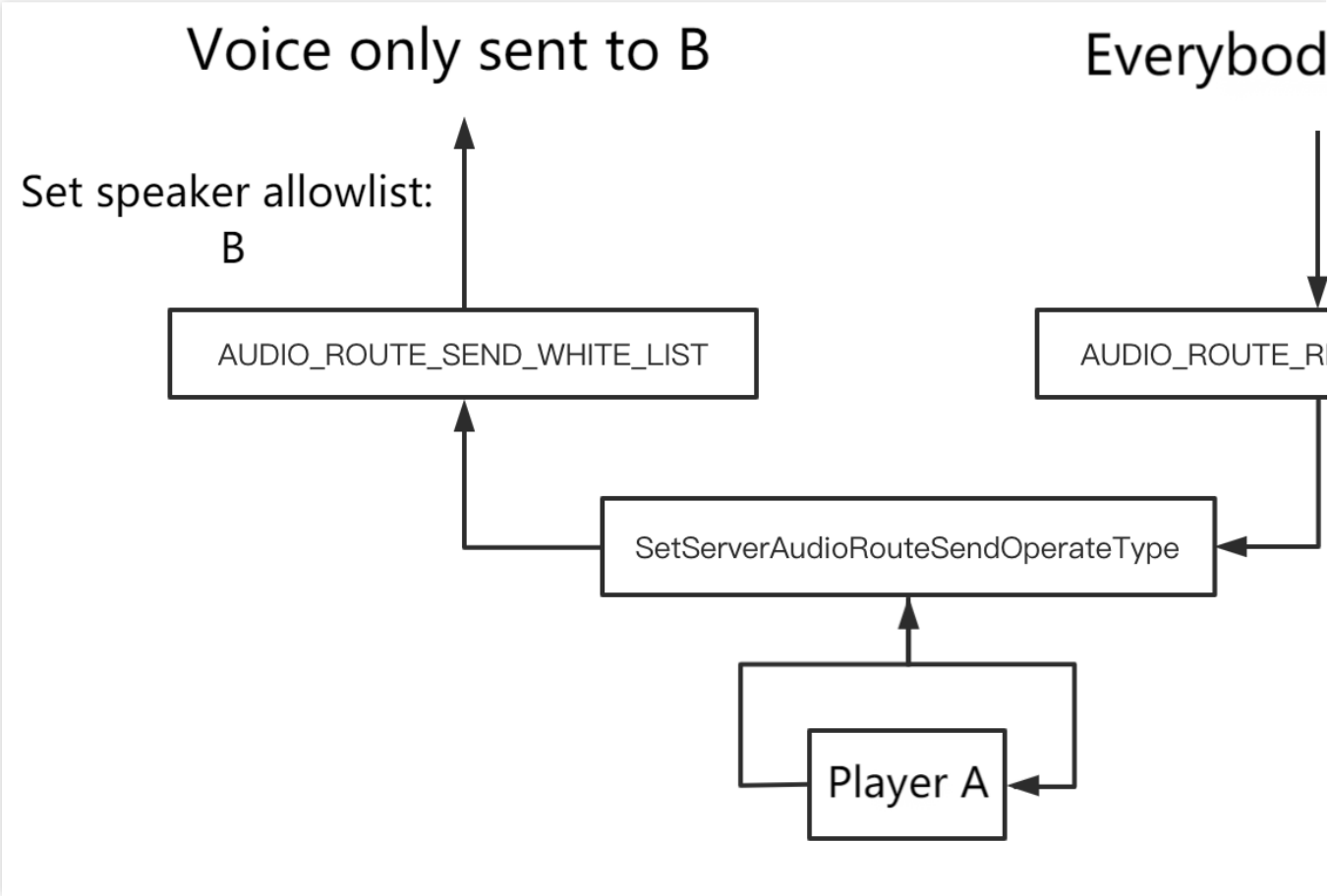
网络音频流转发路由

最近更新时间：2023-04-27 17:28:08

为方便 GME 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文为您介绍适用于 GME 自定义音频转发路由功能的使用参考文档。

使用场景

场景描述：2个好友组小队后，匹配了3个路人组了一个大队伍，需要实现听全队的声音，但只和小队的好友说话。
自定义音频路由功能可以实现。这里5个人都进入同一个语音房间，然后通过音频路由的接口设置，玩家可以设置只听见2人队伍的声音或者全房间的声音，也可以设置为说话只被2人队伍的人听到或者全房间的人听到。
音频规则距离：SetServerAudioRouteSendOperateType(AUDIO_ROUTE_SEND_WHITE_LIST,"2人队伍的list",ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE,"2人队伍的list");
这样声音就只发送给list里面的人，且只接收3人队伍的声音。



前提条件

已开通实时语音服务：可参见 服务开通指引。
已接入 GME SDK：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入、Unity SDK 快速接入、Unreal SDK 快速接入。
使用 GME 实时语音功能成功进入语音房间，并且打开了麦克风（EnableMic）、扬声器（EnableSpeaker）。

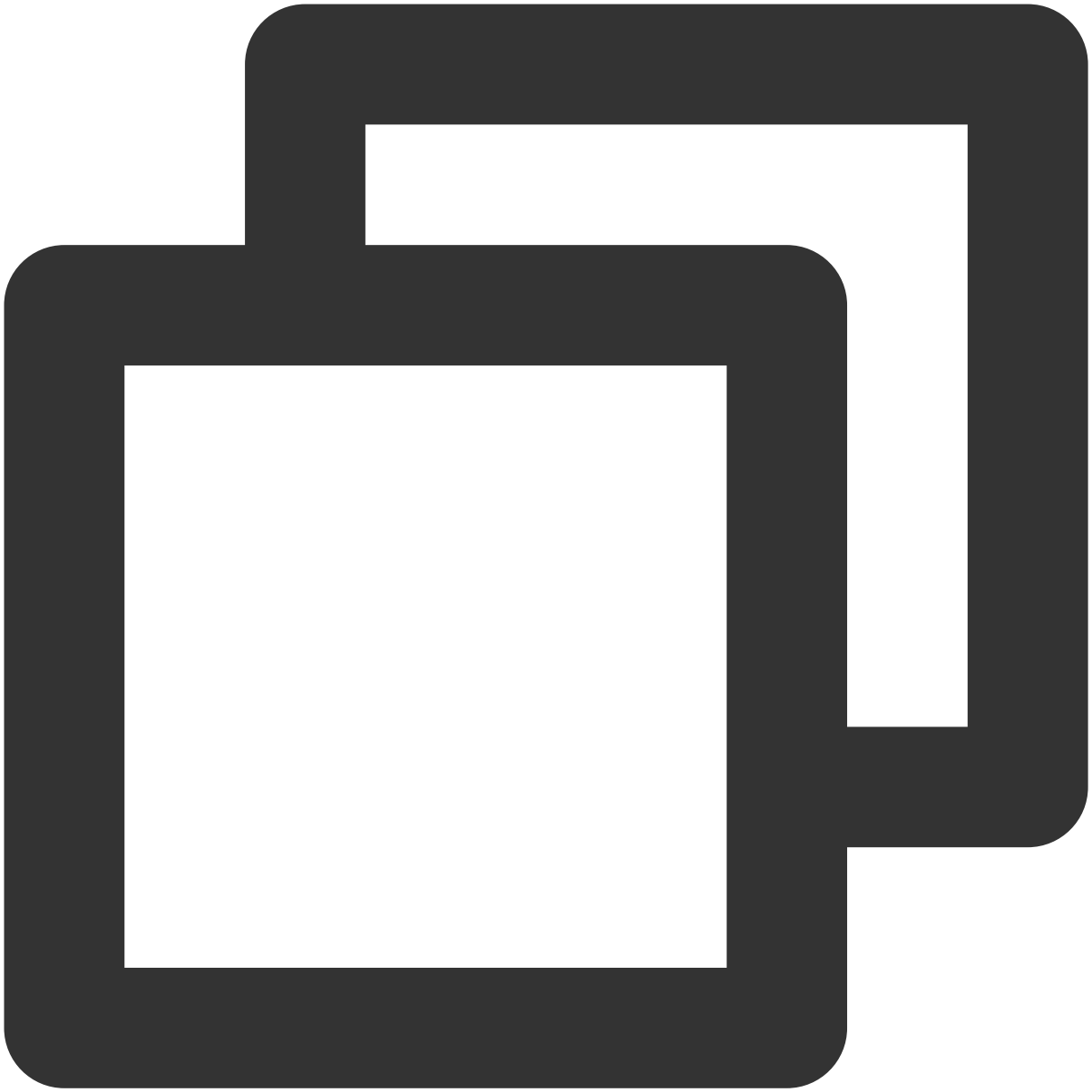
接入音频转发路由功能

设置音频转发规则

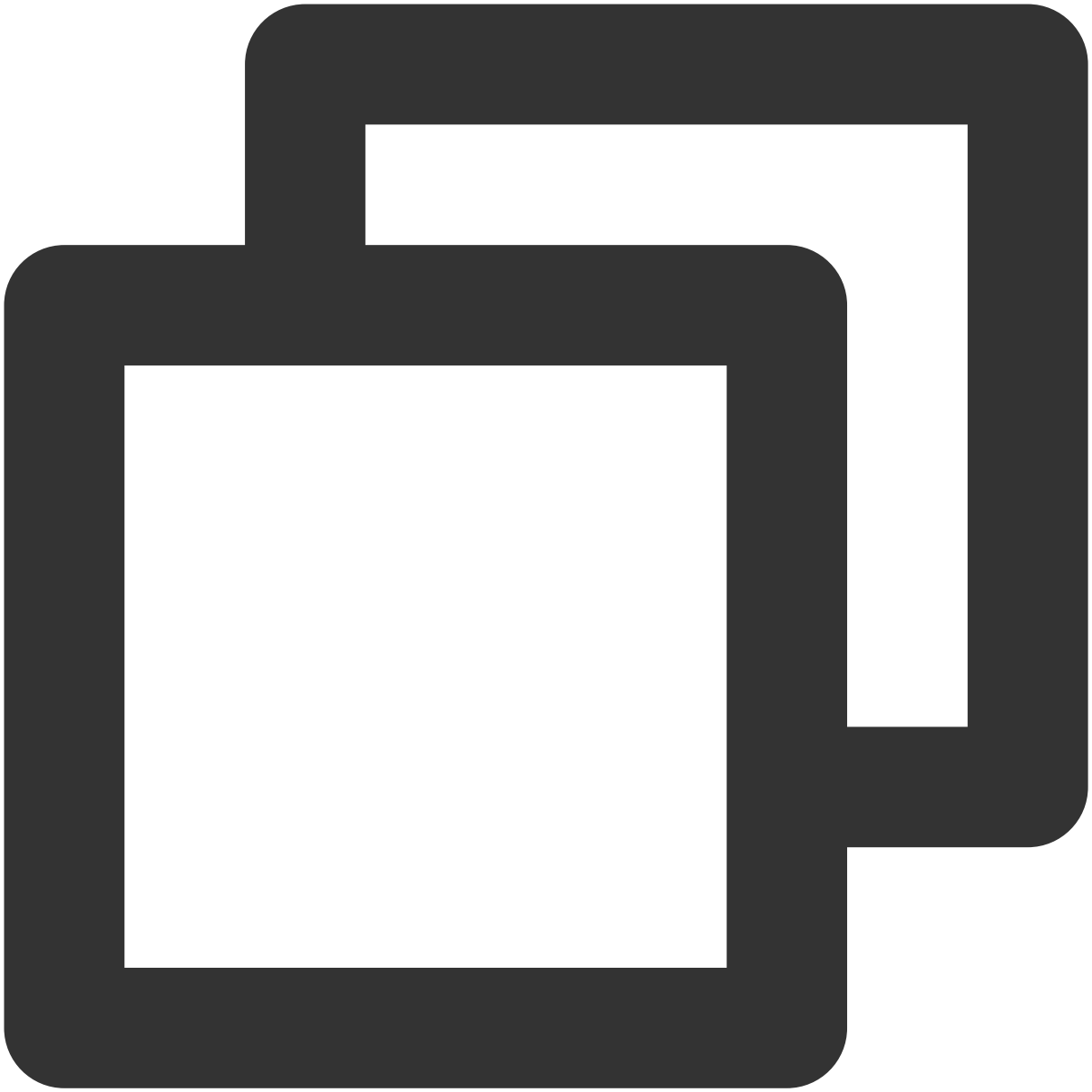
调用此接口设置音频转发规则，此接口在进房成功回调中调用，调用后此次进房生效，退房后失效。
注意
禁言功能 AddBlackList 为本端生效，优先级高于自定义音频路由。例如 A 通过 SetServerAudioRouteSendOperateType 设置了只听 B 说话，但是又调用了 AddBlackList 将B禁言，此时 A 将听不到 B 的声音。

接口原型

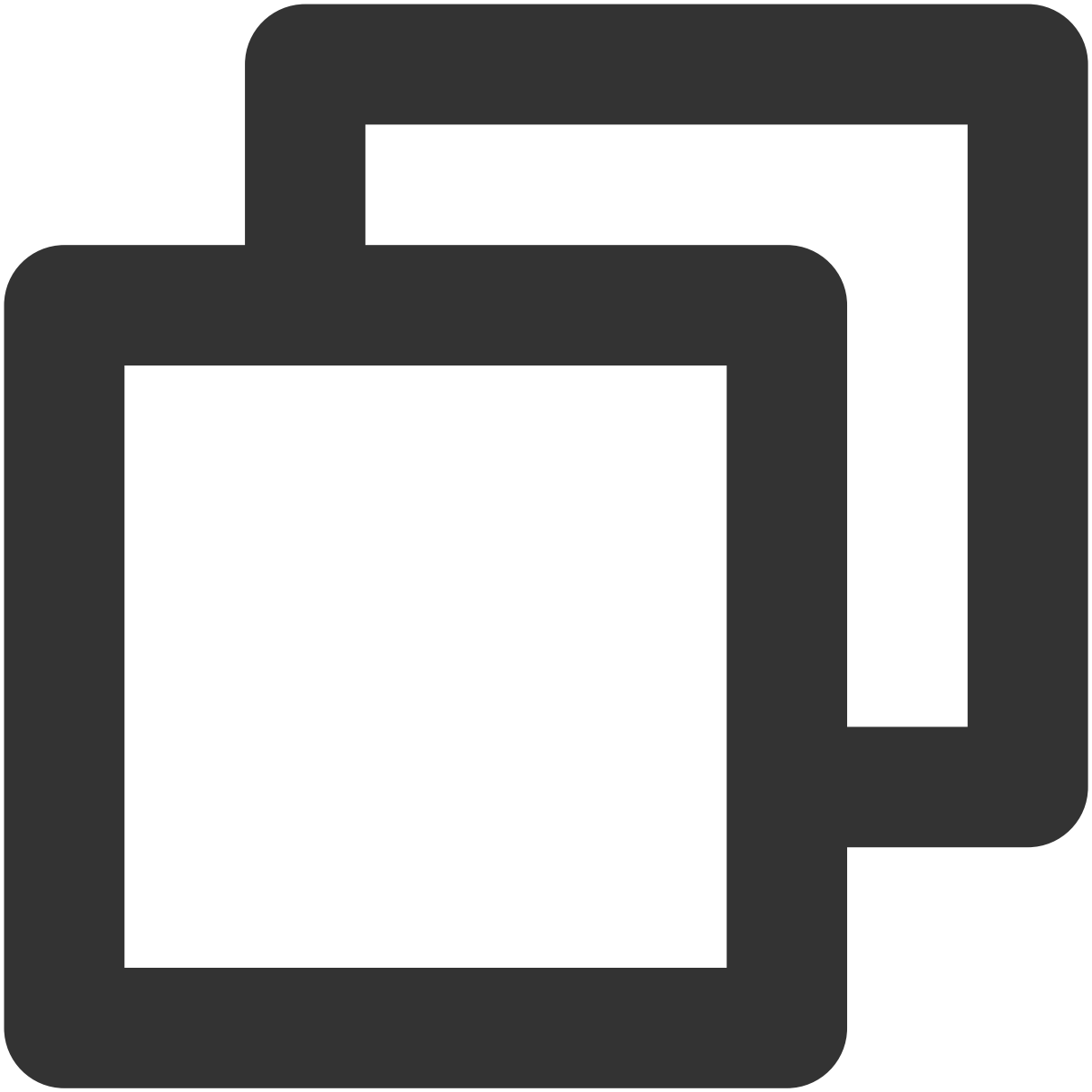
Unity
C++
Android
iOS



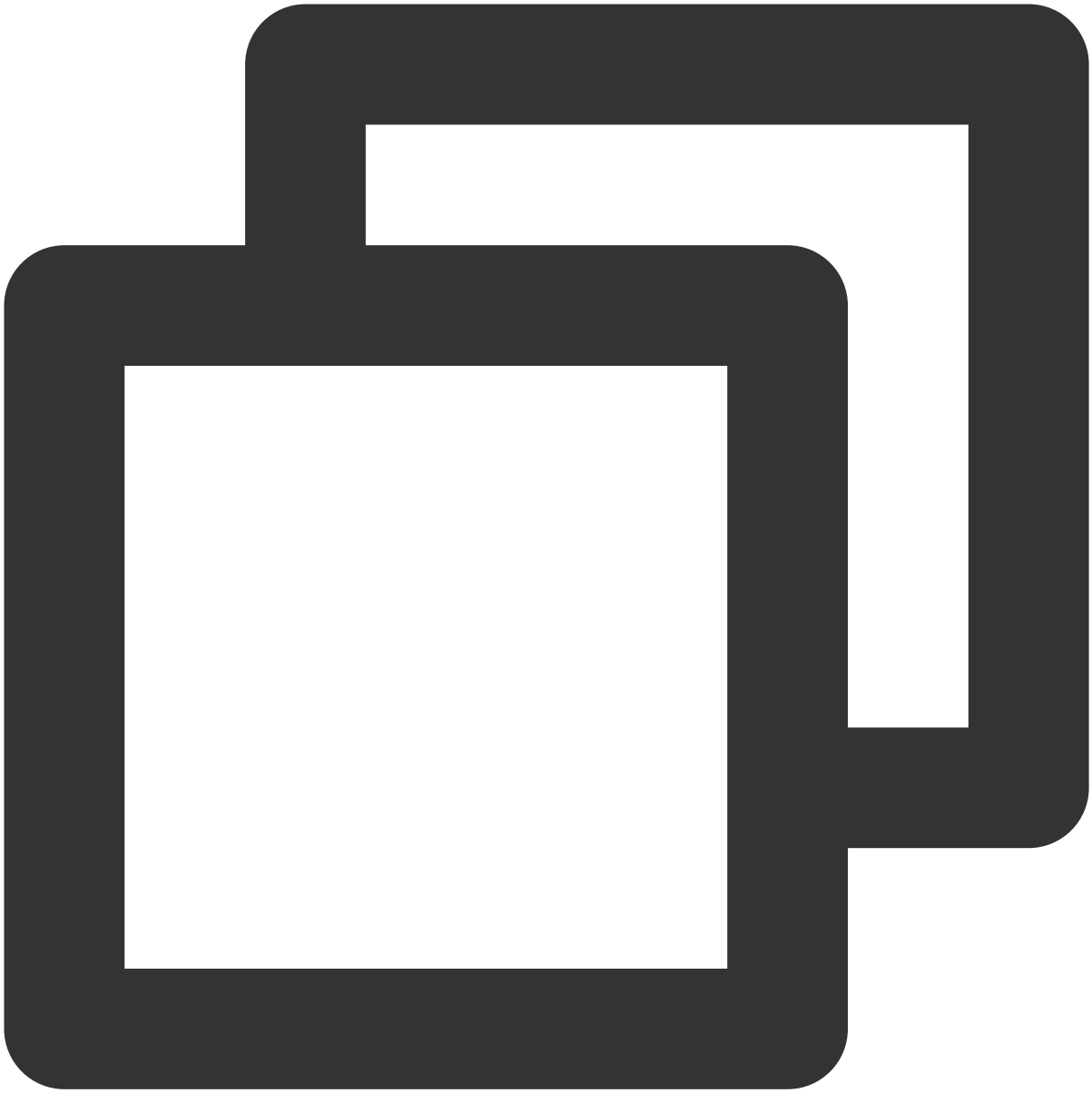
```
public abstract class ITMGRoom{
    public abstract int SetServerAudioRouteSendOperateType(ITMG_SERVER_AUDIO_ROUTE_
}
```



```
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
```

类型说明

ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE

设置发送音频规则，不同的规则填入后，会有不同的发送规则。

接收类型	效果
AUDIO_ROUTE_NOT_SEND_TO_ANYONE	本端音频上行发送到后台，但后台不转发给任何人，相当于将自己静音，此时参数 OpenIDForSend 无效，只需填 null

AUDIO_ROUTE_SEND_TO_ALL	本端音频上行将转发给所有人，此时参数 OpenIDForSend 无效，只需填 null
AUDIO_ROUTE_SEND_BLACK_LIST	本端音频上行将不转发给黑名单的人，黑名单由参数 OpenIDForSend 提供
AUDIO_ROUTE_SEND_WHITE_LIST	本端音频上行将只转发给白名单的人，白名单由参数 OpenIDForSend 提供

说明

如果类型传入 AUDIO_ROUTE_NOT_SEND_TO_ANYONE 以及 AUDIO_ROUTE_SEND_TO_ALL，此时的参数 OpenIDForSend 不生效，只需要填 null。

如果类型传入 AUDIO_ROUTE_SEND_BLACK_LIST，此时参数 OpenIDForSend 为黑名单列表，最多支持 10 个。

如果类型传入 AUDIO_ROUTE_SEND_WHITE_LIST，此时参数 OpenIDForSend 为白名单列表，最多支持 10 个。

ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE

设置接收音频规则，不同的规则填入后，会有不同的接收规则。

接收类型	效果
AUDIO_ROUTE_NOT_RECV_FROM_ANYONE	本端不接受任何音频，相当于关闭房间内扬声器效果，此时参数 OpenIDForSend 无效，只需填 null
AUDIO_ROUTE_RECV_FROM_ALL	本端接收所有人的音频，此时参数 OpenIDForSend 无效，只需填 null
AUDIO_ROUTE_RECV_BLACK_LIST	本端不接收黑名单的人的音频声音，黑名单由参数 OpenIDForSend 提供
AUDIO_ROUTE_RECV_WHITE_LIST	本端只接收白名单的人的音频声音，白名单由参数 OpenIDForSend 提供

说明

如果类型传入 AUDIO_ROUTE_NOT_RECV_FROM_ANYONE 以及 AUDIO_ROUTE_RECV_FROM_ALL OpenIDForSend 不生效。

如果类型传入 AUDIO_ROUTE_RECV_BLACK_LIST，此时参数 OpenIDForSend 为黑名单列表，最多支持 10 个。

如果类型传入 AUDIO_ROUTE_RECV_WHITE_LIST，此时参数 OpenIDForSend 为白名单列表，最多支持 10 个。

返回值

接口返回值为 QAV_OK 则表示成功。

若回调返回 1004 表示参数错误，建议重新检查参数是否正确。

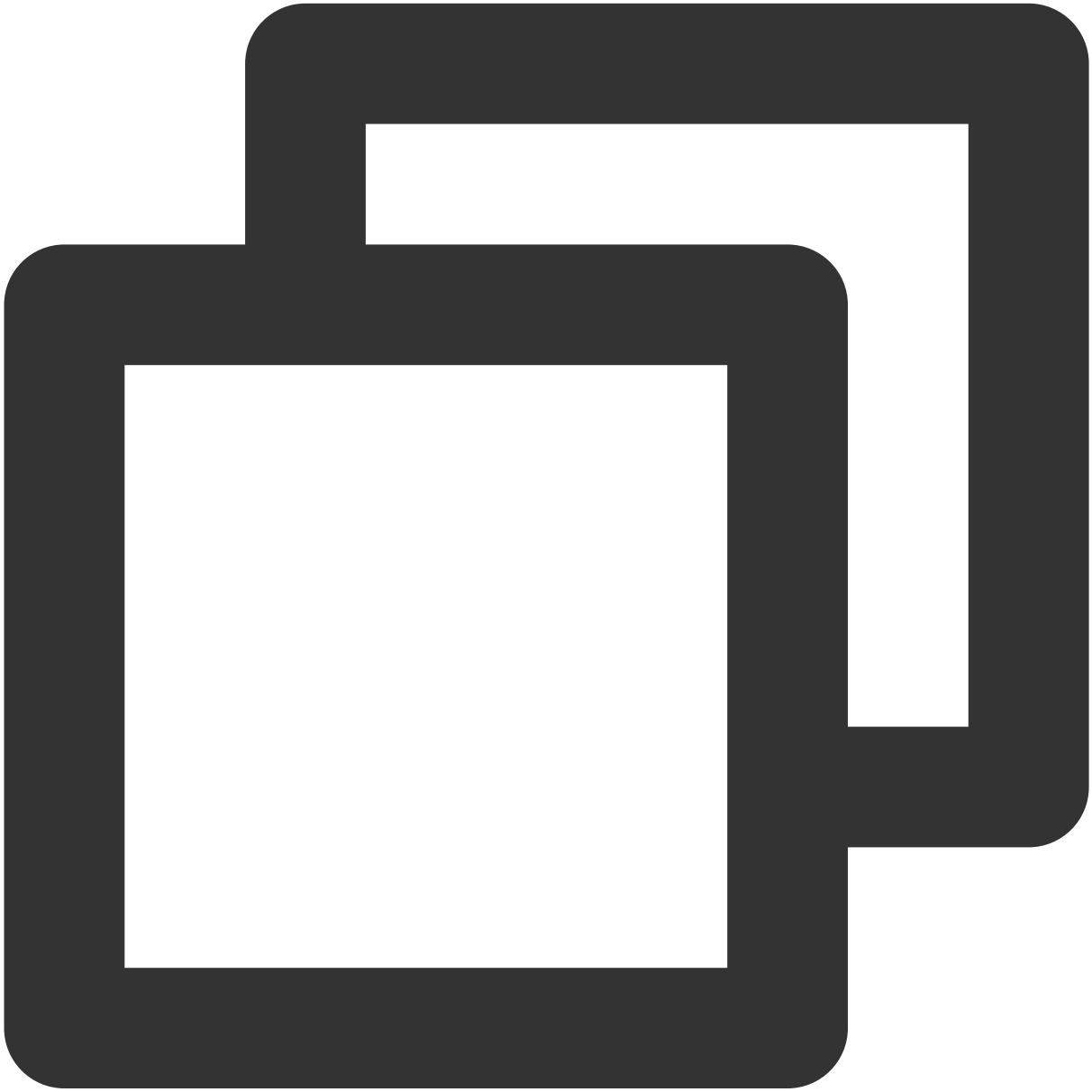
若回调返回 1001 表示重复操作。

若回调返回 1201 表示房间不存在，建议检查房间号是否正确。

若回调返回 10001 以及 1005，建议重新调用接口一次。
更多返回结果解释详情请参见 错误码 。

示例代码

执行语句

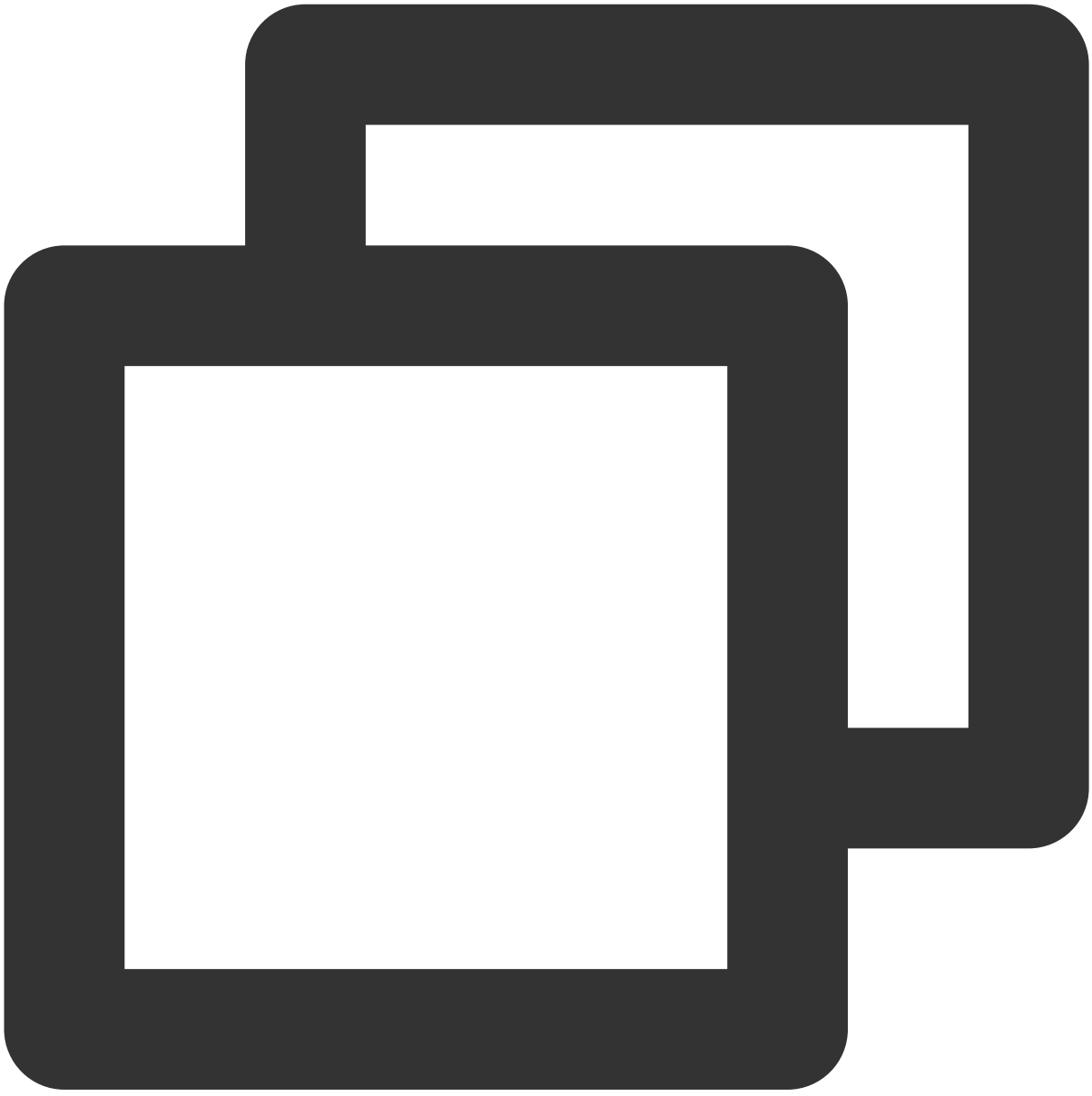


```
@synthesize _sendListArray;
@synthesize _recvListArray;

int ret =  [[[ITMGContext GetInstance] GetRoom] SetServerAudioRouteSendOperateType:
if (ret != QAV_OK) {
```

```
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"更新audioroute列表失败"
    [alert show];
}
```

回调



```
-(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:@"更新audior
```

```
                                [alert show];
            }
        }
        default:
            break;
    }
}
```

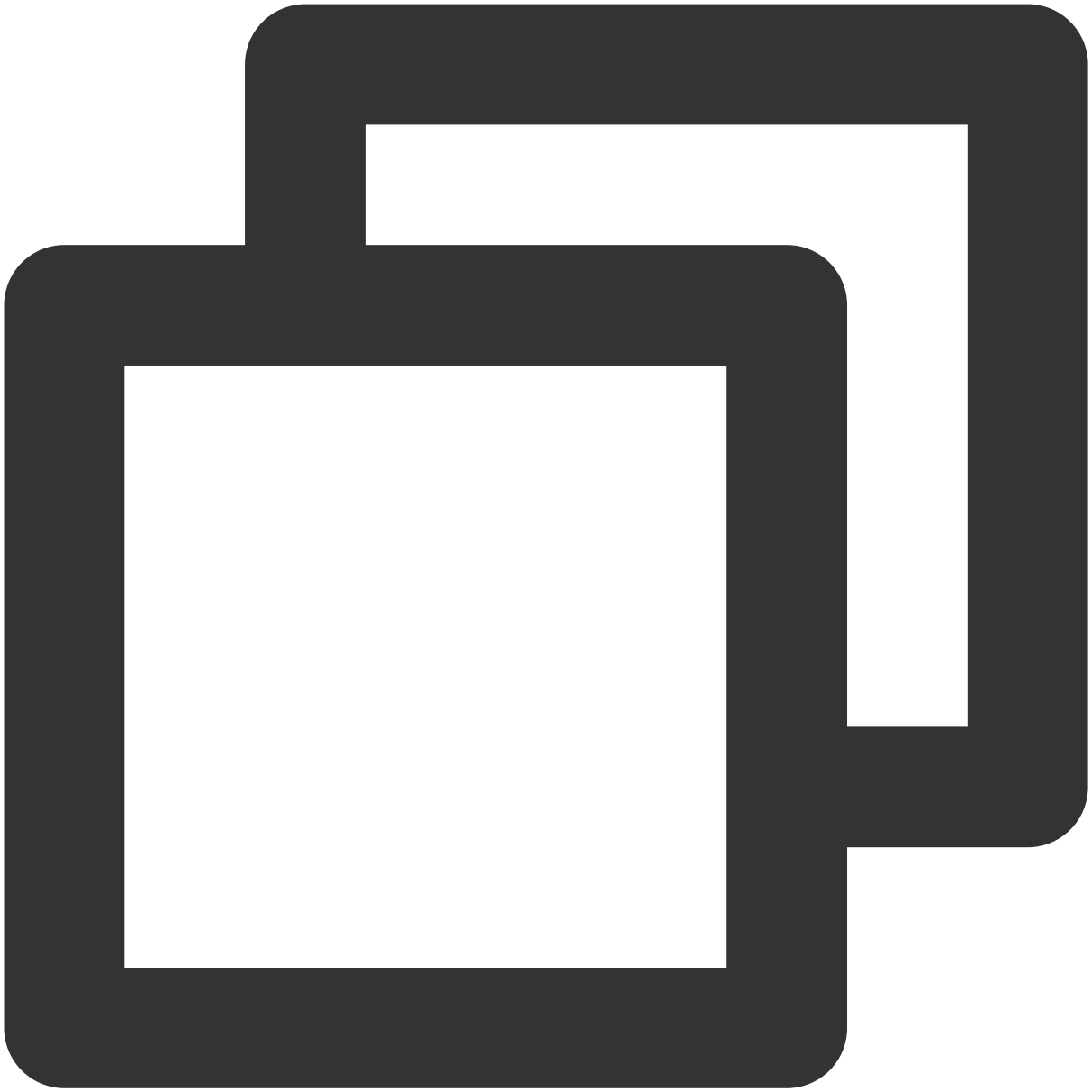
获取音频设置转发规则

调用此接口获取音频转发规则。调用后接口返回规则，传入的数组参数会返回相应规则的 openId。

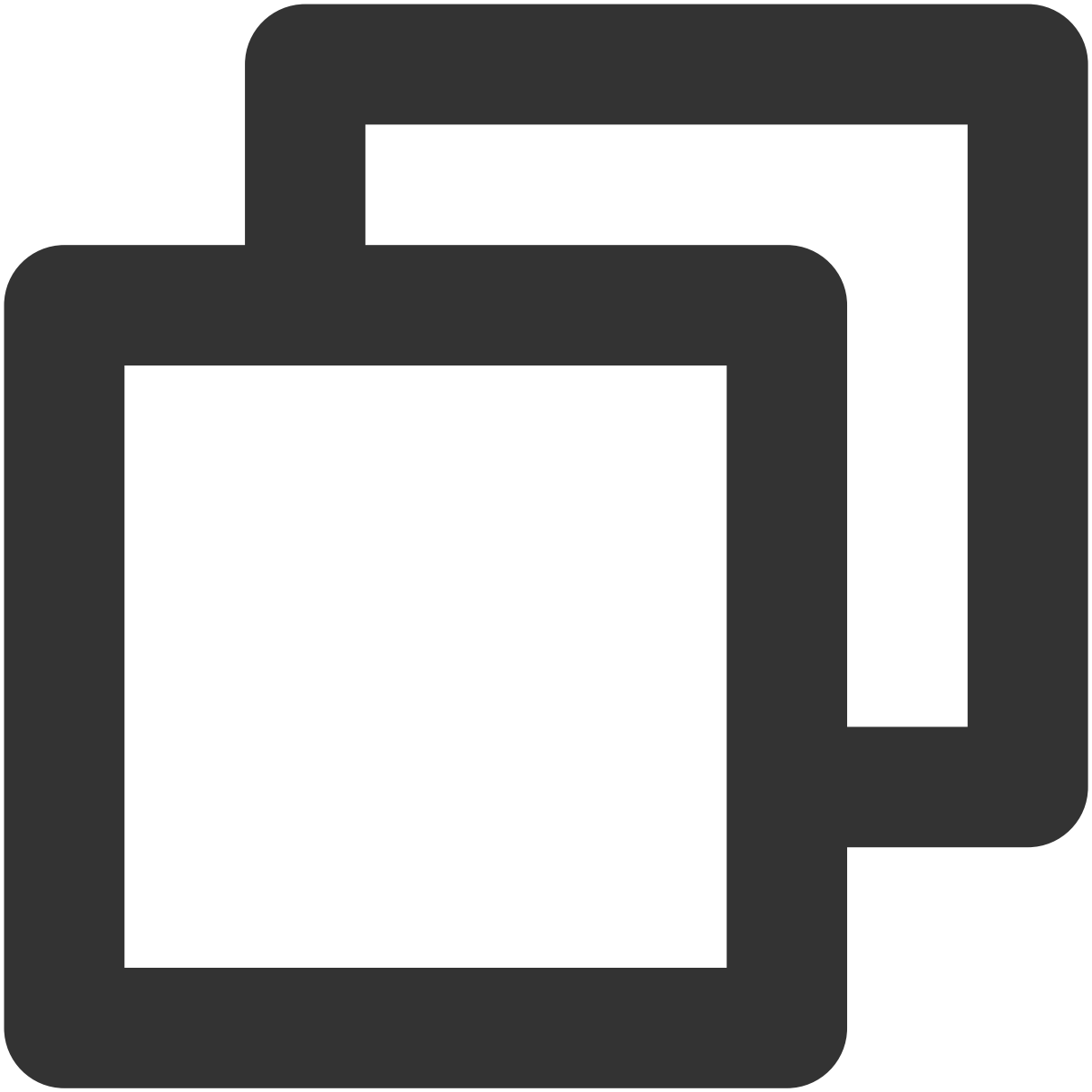
接口原型

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
```

返回规则

ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE

接收类型	效果
AUDIO_ROUTE_NOT_SEND_TO_ANYONE	本端音频上行发送到后台，但后台不转发给任何人，相当于将自己静音

AUDIO_ROUTE_SEND_TO_ALL	本端音频上行将转发给所有人
AUDIO_ROUTE_SEND_BLACK_LIST	本端音频上行将不转发给黑名单的人
AUDIO_ROUTE_SEND_WHITE_LIST	本端音频上行将只转发给白名单的人
AUDIO_ROUTE_RECV_INQUIRE_ERROR	获取出错，检查是否进入房间，是否已经初始化 SDK

ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE

接收类型	效果
AUDIO_ROUTE_NOT_RECV_FROM_ANYONE	本端不接受任何音频，相当于关闭房间内扬声器效果
AUDIO_ROUTE_RECV_FROM_ALL	本端接收所有人的音频
AUDIO_ROUTE_RECV_BLACK_LIST	本端不接收黑名单的人的音频声音
AUDIO_ROUTE_RECV_WHITE_LIST	本端只接收白名单的人的音频声音
AUDIO_ROUTE_RECV_INQUIRE_ERROR	获取出错，检查是否进入房间，是否已经初始化 SDK

注意

在 `SetServerAudioRouteSendOperateType` 接口中请勿使用 `AUDIO_ROUTE_RECV_INQUIRE_ERROR` 。

自定义消息通道

最近更新时间：2023-04-27 17:31:09

为方便 GME 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文为您介绍适用于 GME 用户自定义音频包附带消息功能的使用指引。

使用场景

GME 用户自定义音频包附带消息功能可以让开发者在 GME 音频包中携带自定义消息，作为信令传递广播给同房间的人。

前提条件

已开通实时语音服务：可参见 服务开通指引。
已接入 GME SDK：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入、Unity SDK 快速接入、Unreal SDK 快速接入。

使用限制

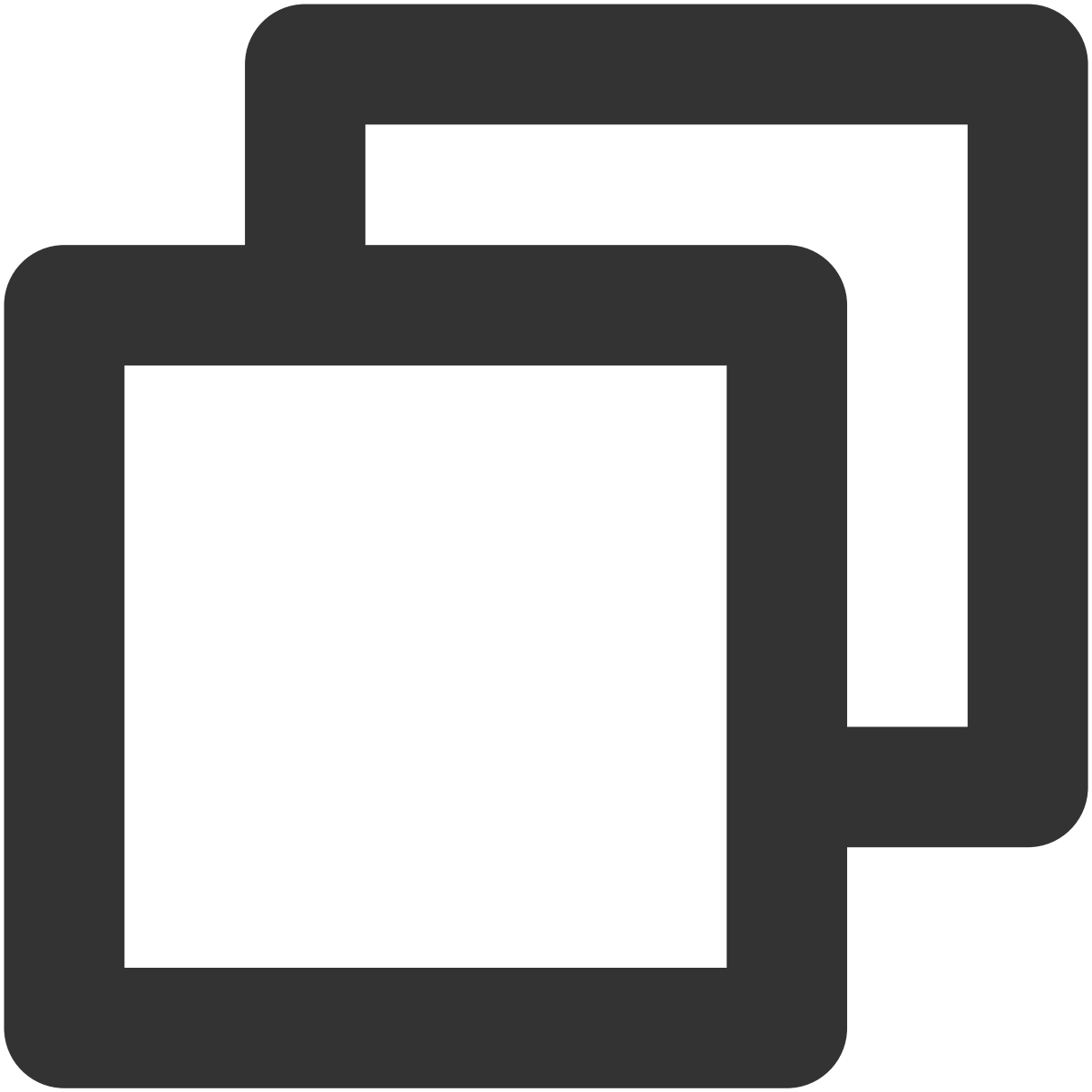
调用此接口需要在房间类型为标准及高清（ITMG_ROOM_TYPE_STANDARD 及 ITMG_ROOM_TYPE_HIGHQUALITY）的情况下，发送端需要打开麦克风，接收端需要打开扬声器。

自定义消息功能接入

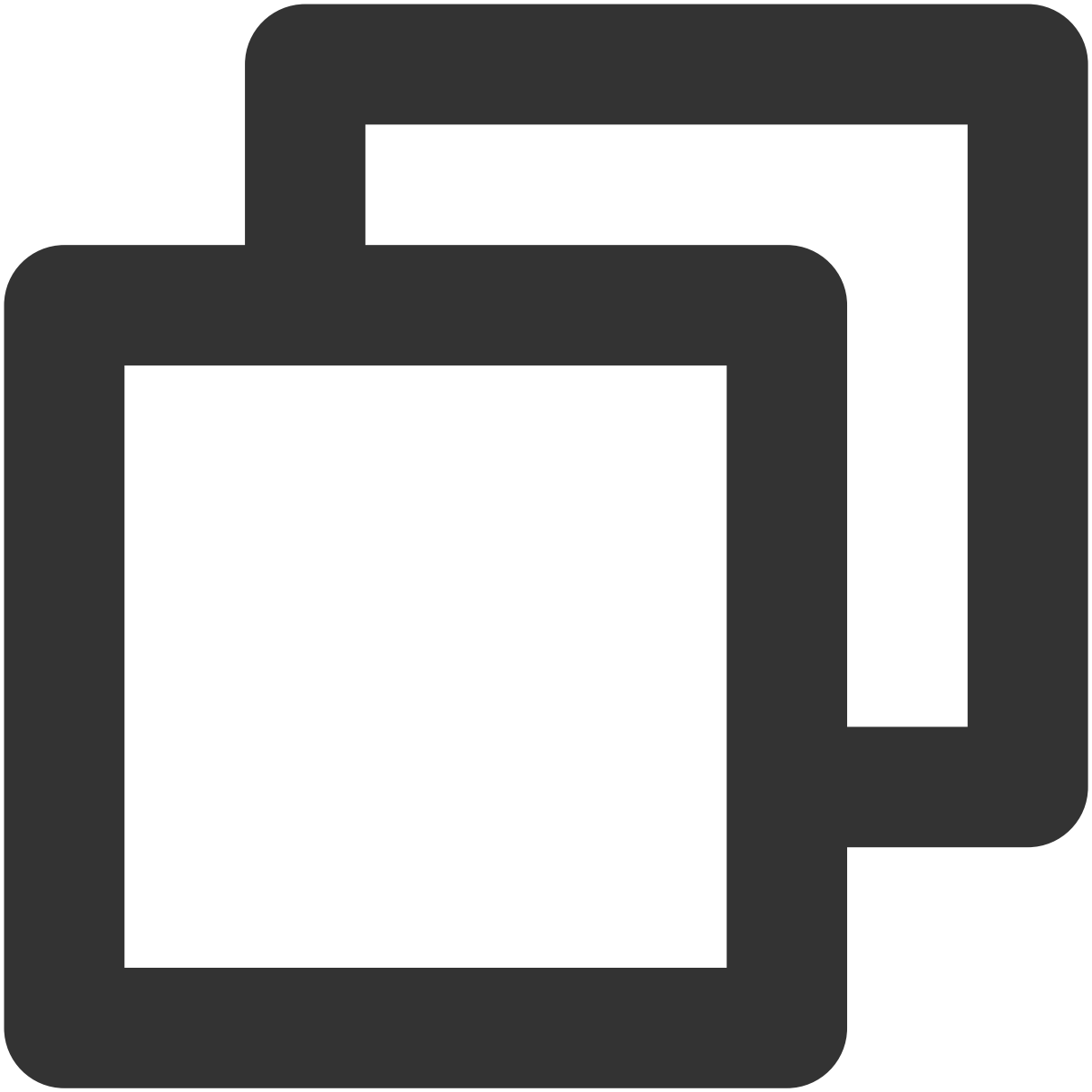
发送自定义消息

接口原型

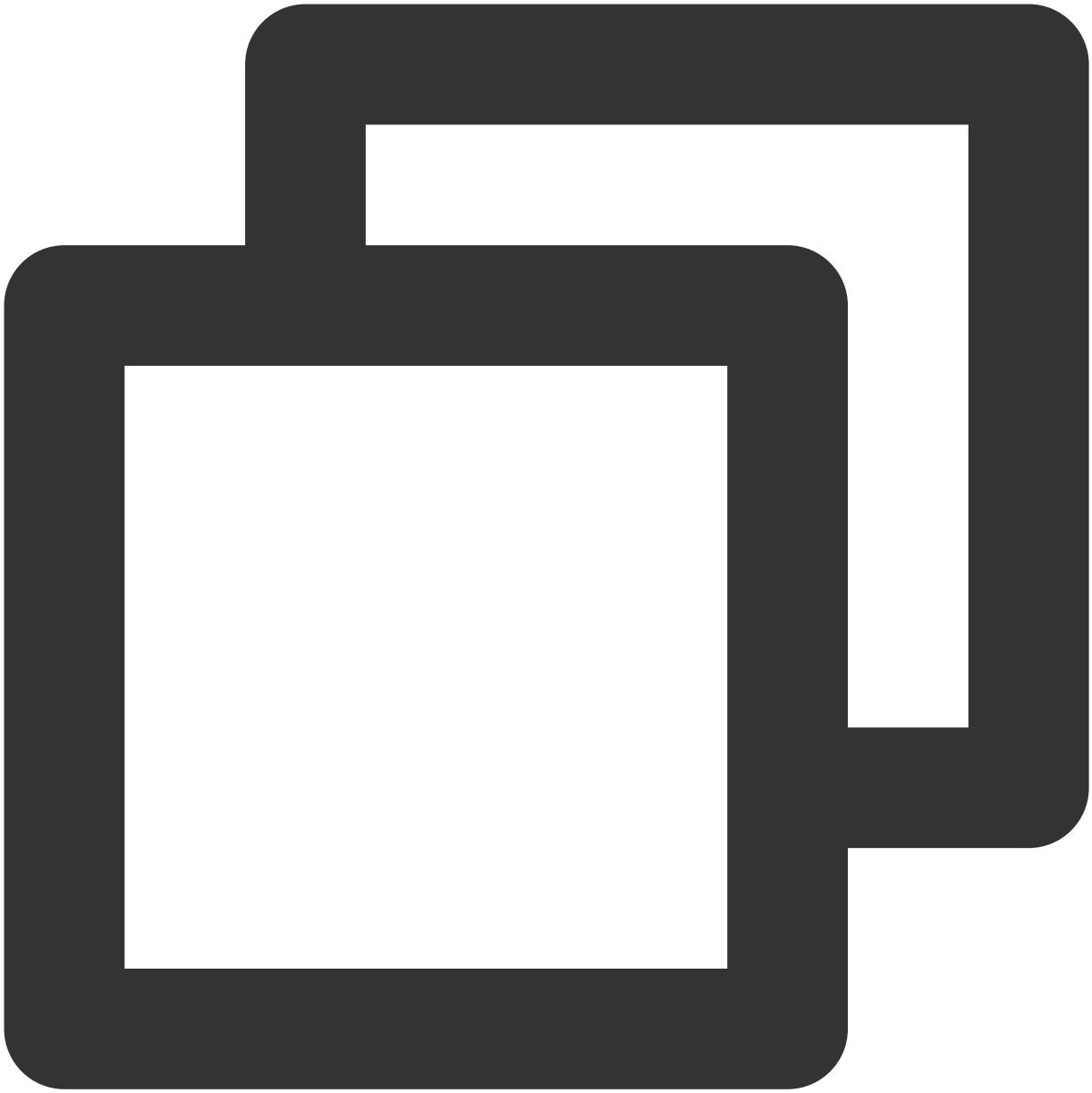
iOS
Android
Unity



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



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



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

参数说明

参数	类型	含义
data	NSData * 、byte[]	需要传递的信息
reaptCout	int	重复次数，填入-1为无限次重复发送

返回值

接口返回值为 QAV_OK 则表示成功。
回调返回1004表示参数错误，建议重新检查参数是否正确。返回1201表示房间不存在，建议检查房间号是否正确。
更多错误码请参见 错误码 文档。

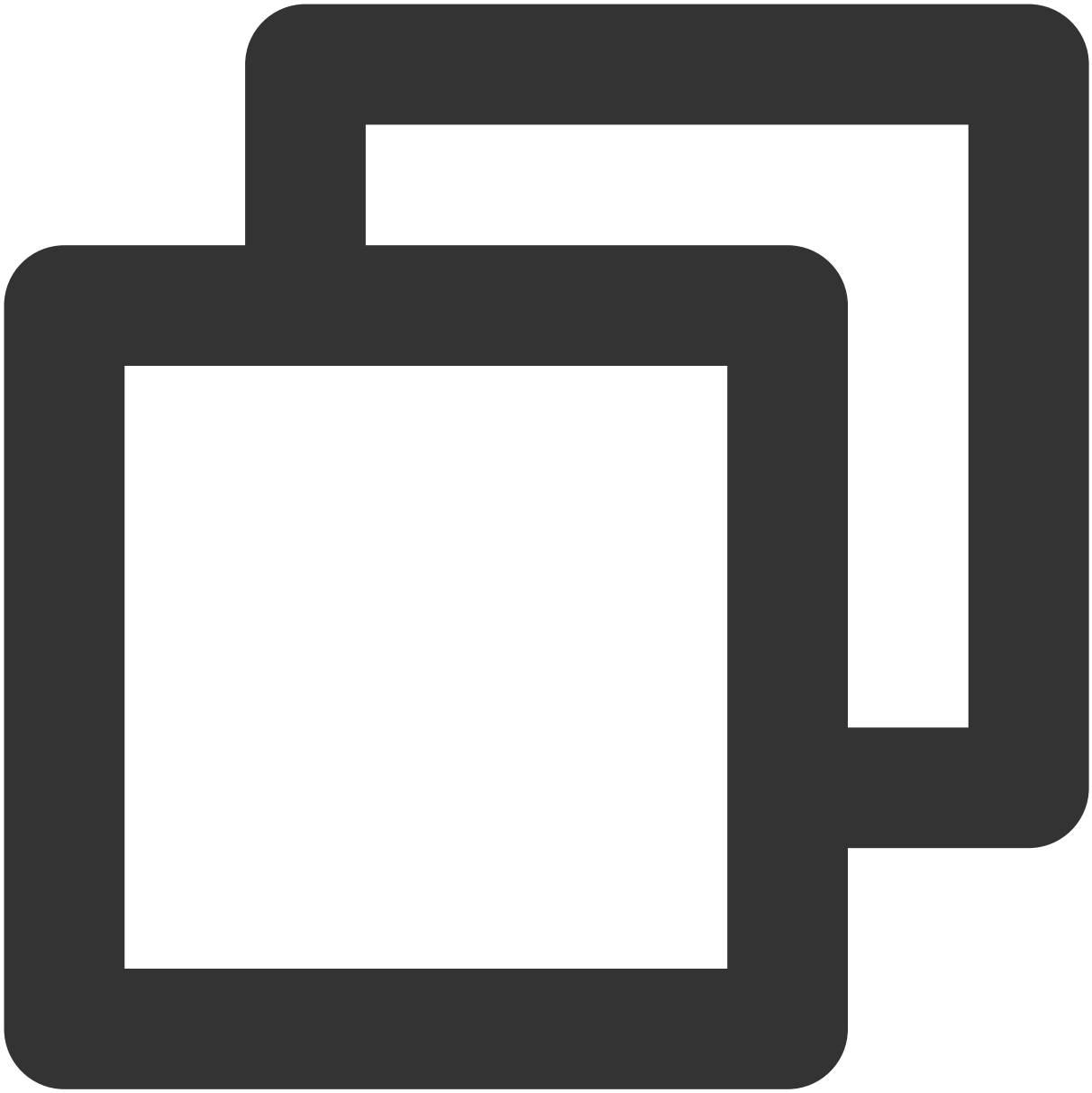
示例代码

执行语句

iOS

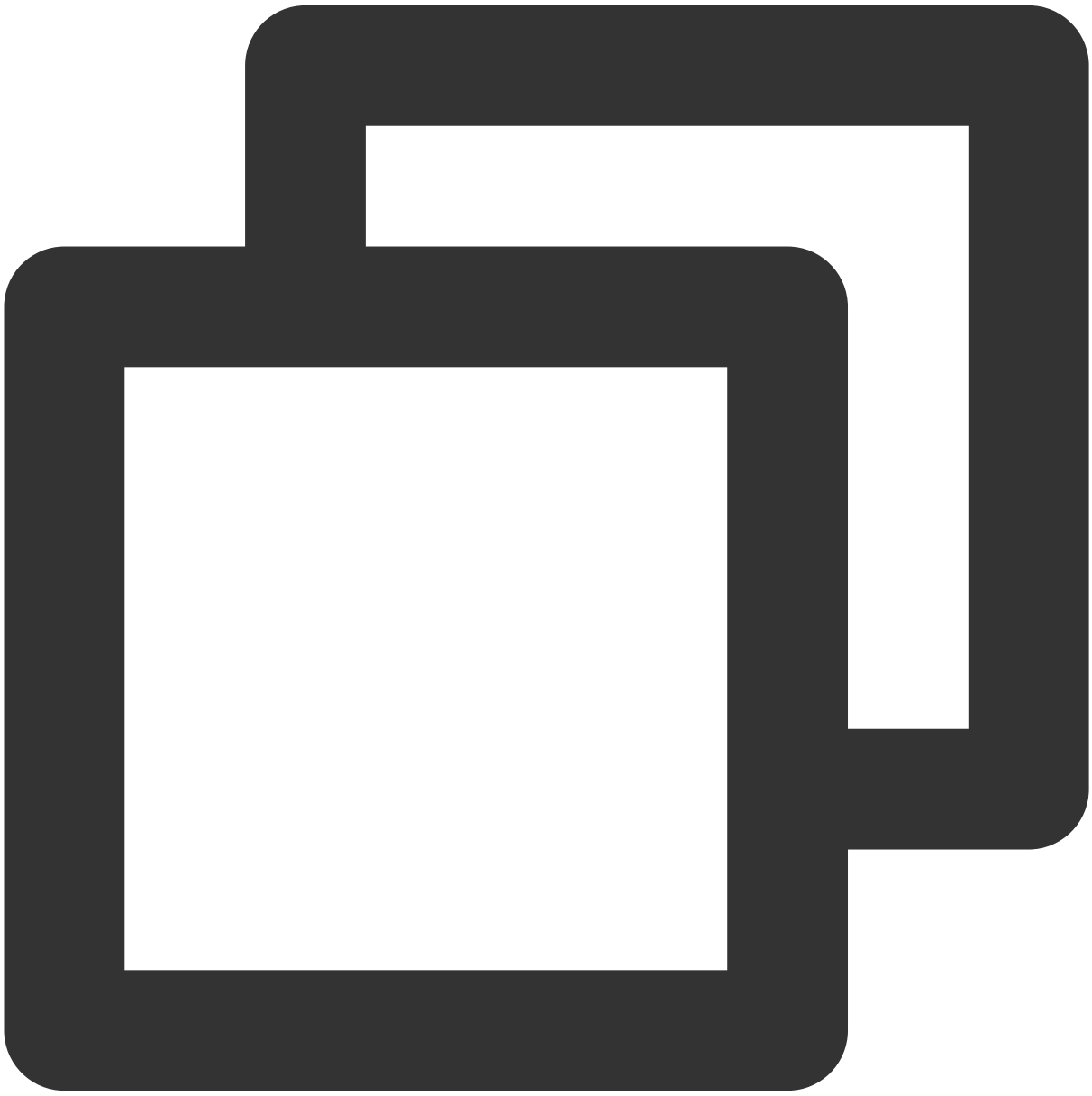
Android

Unity



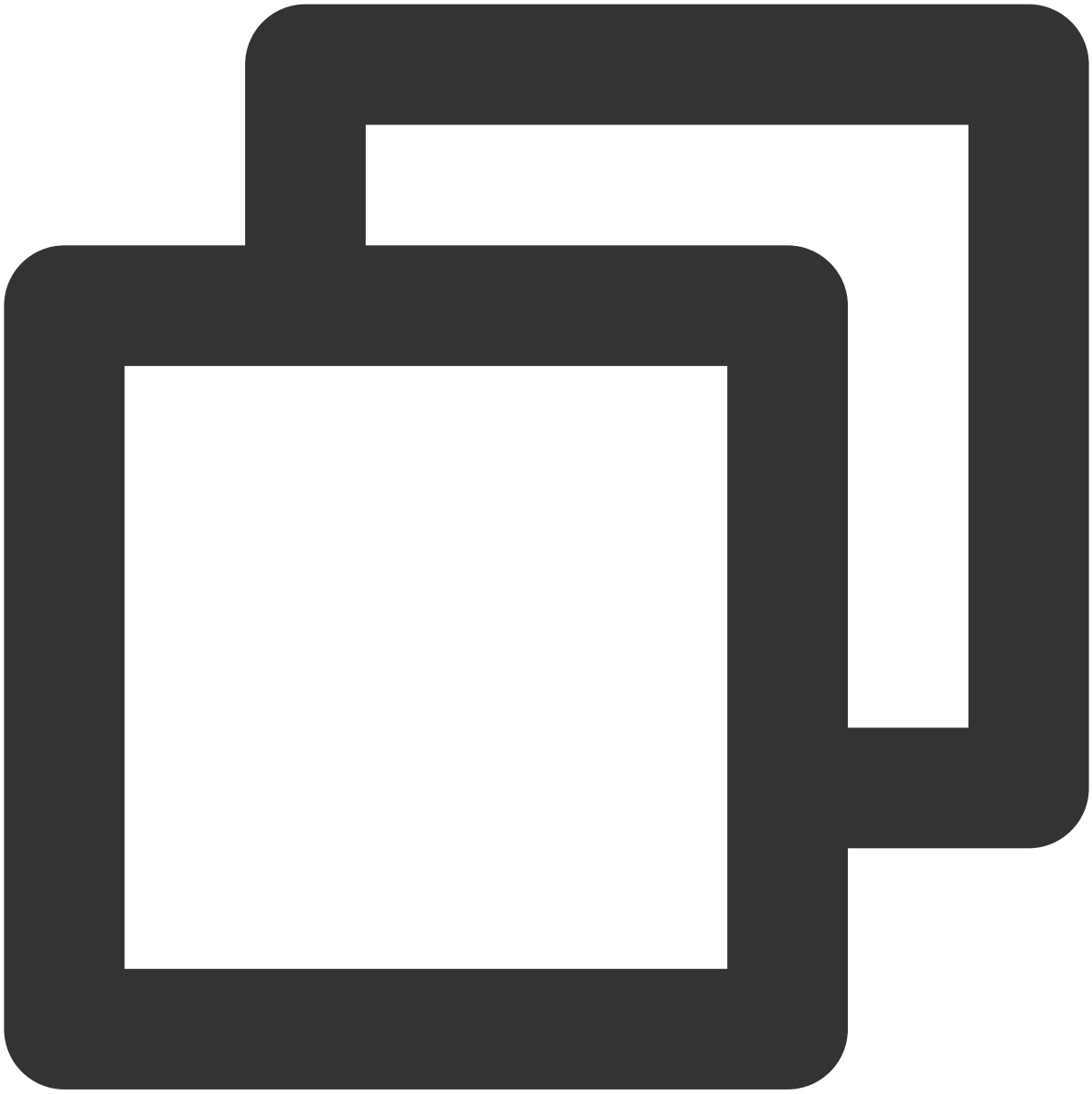
```
-(IBAction)SendCustData:(UIButton*)sender {
    int ret = 0;
    NSString *typeString;
    switch (sender.tag) {
        case 1:
            ret = [[[ITMGContext GetInstance] GetRoom] SendCustomData:[NSData dataW
            typeString = @"sendCustData";
            break;
          case 2:
    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

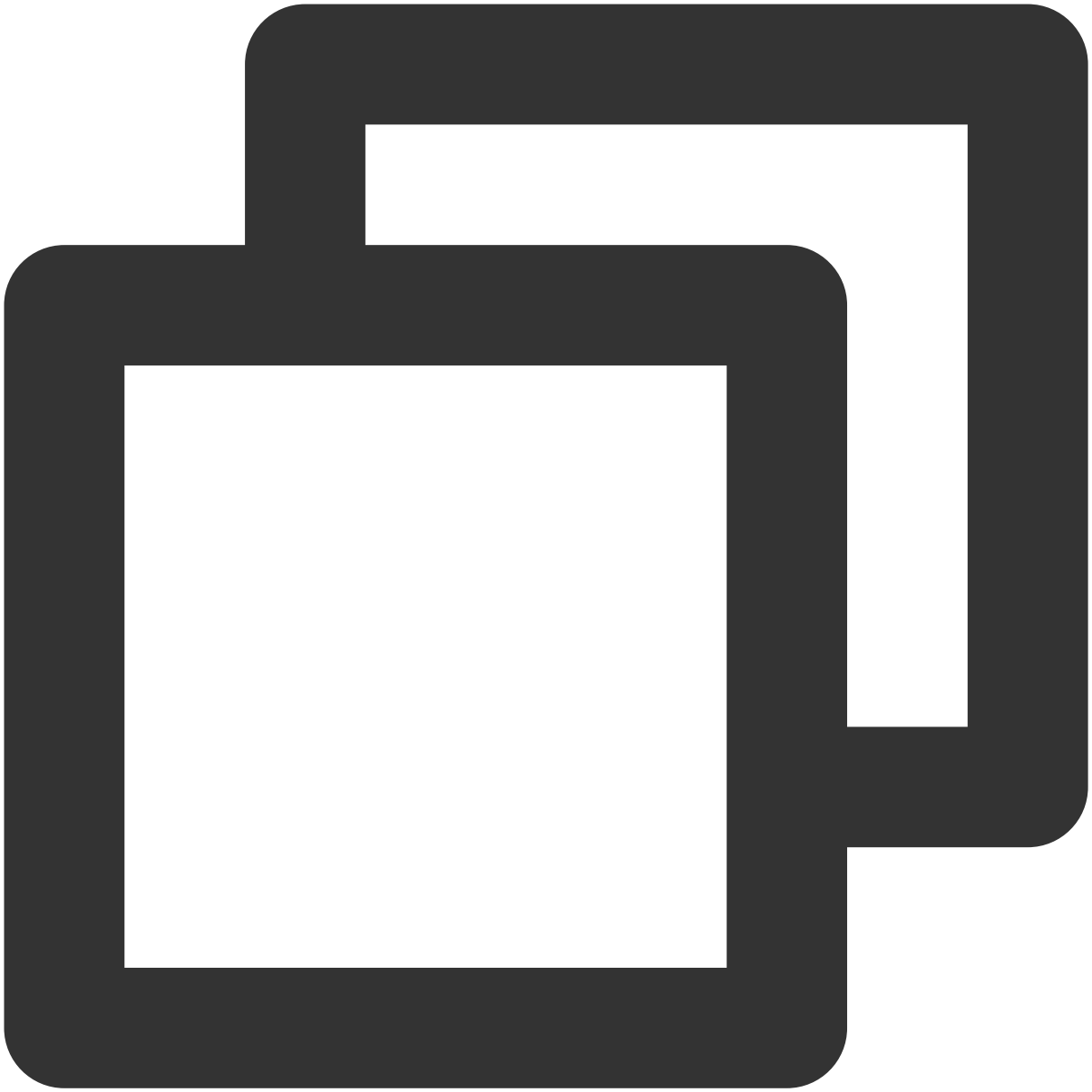
```
            int count = 0;
            if (int.TryParse(str_count, out count)) {
                Debug.Log(data+ count.ToString());
                byte[] byteData = Encoding.Default.GetBytes(data);
               int ret =  ITMGContext.GetInstance().GetRoom().SendCustomData(byteDat
            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

回调

iOS

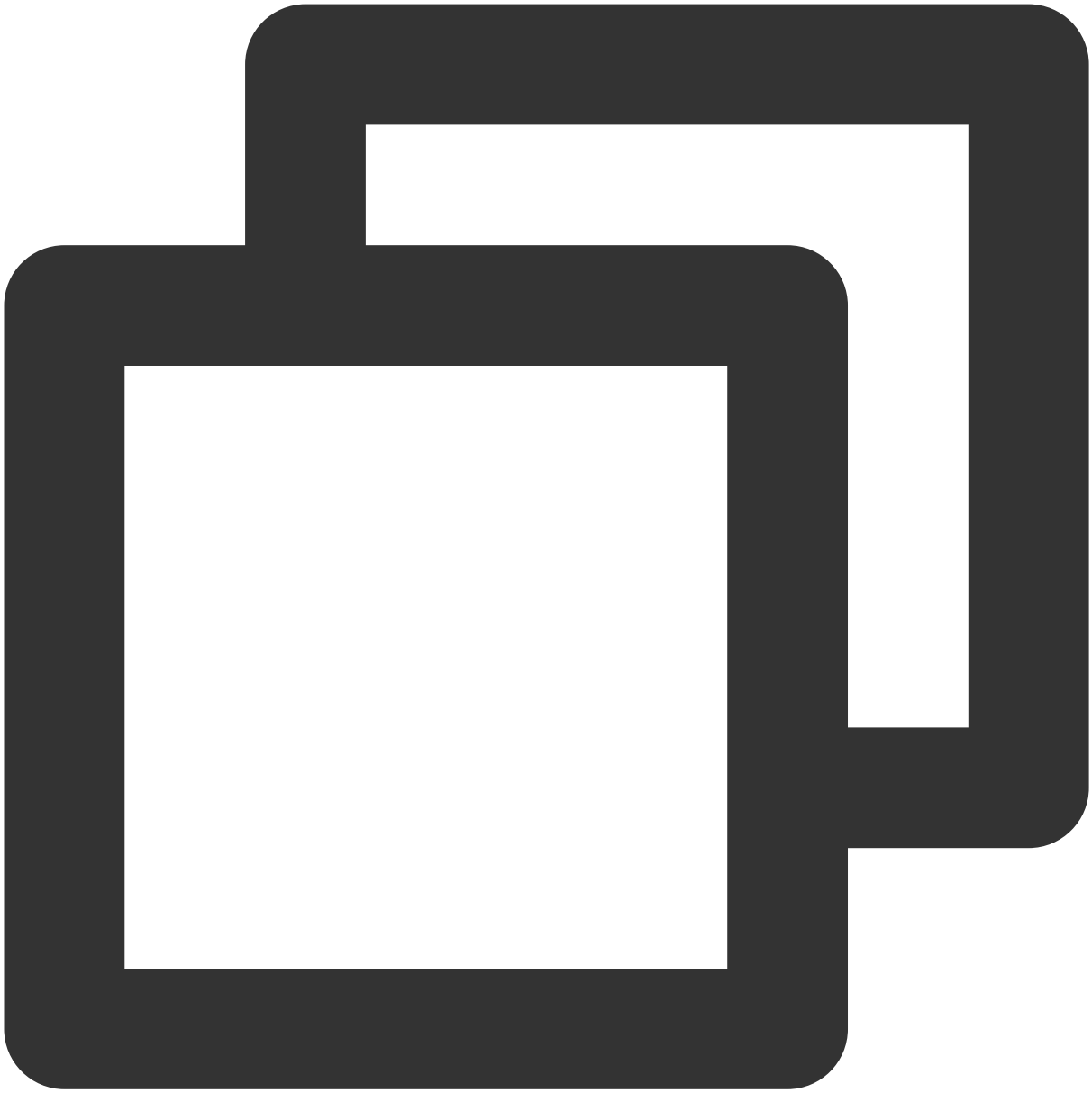
Android

Unity

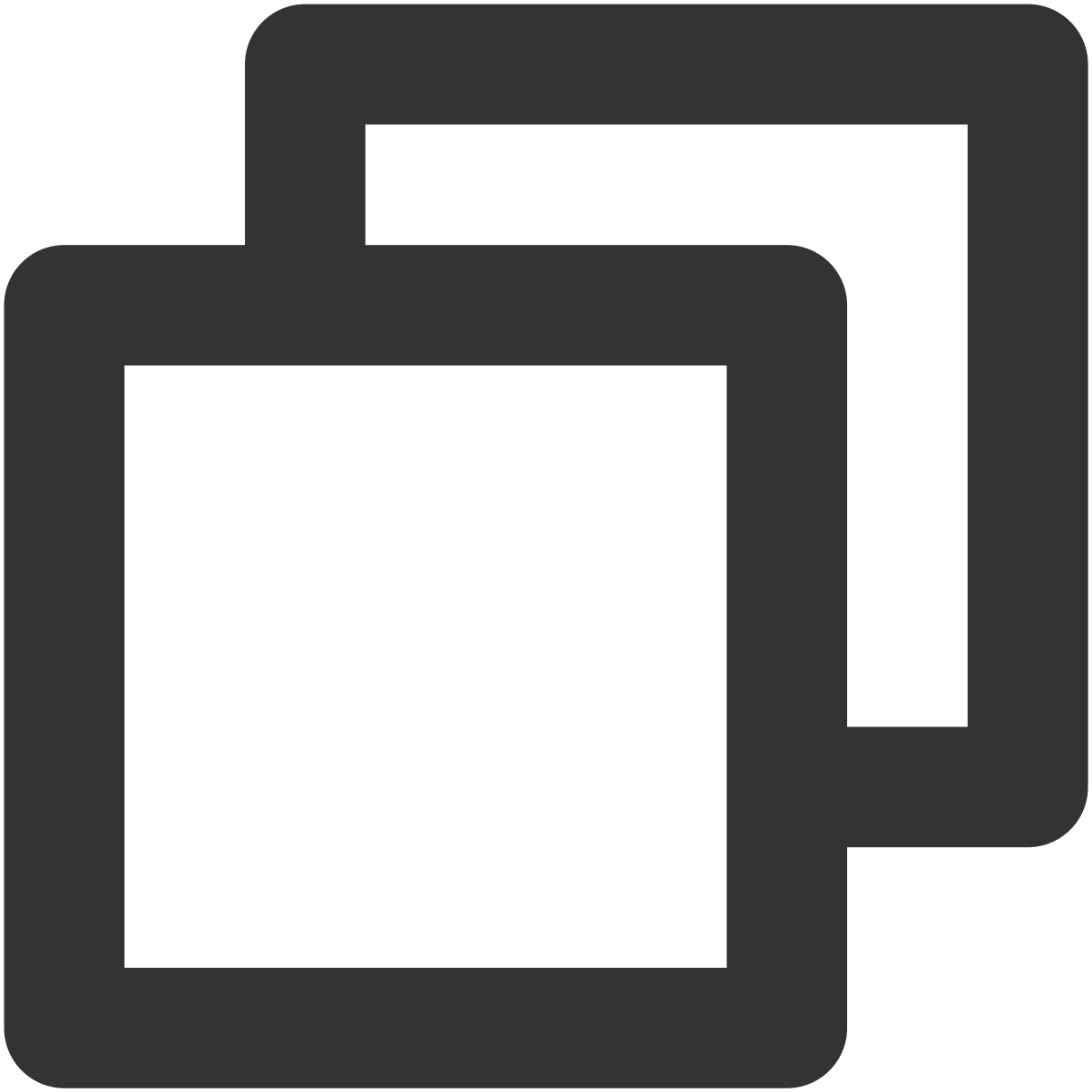


```
-(IBAction)SendCustData:(UIButton*)sender {
    int ret = ret = [[[ITMGContext GetInstance] GetRoom] SendCustomData:[NSData dat

    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;
    }
```

停止发送自定义消息

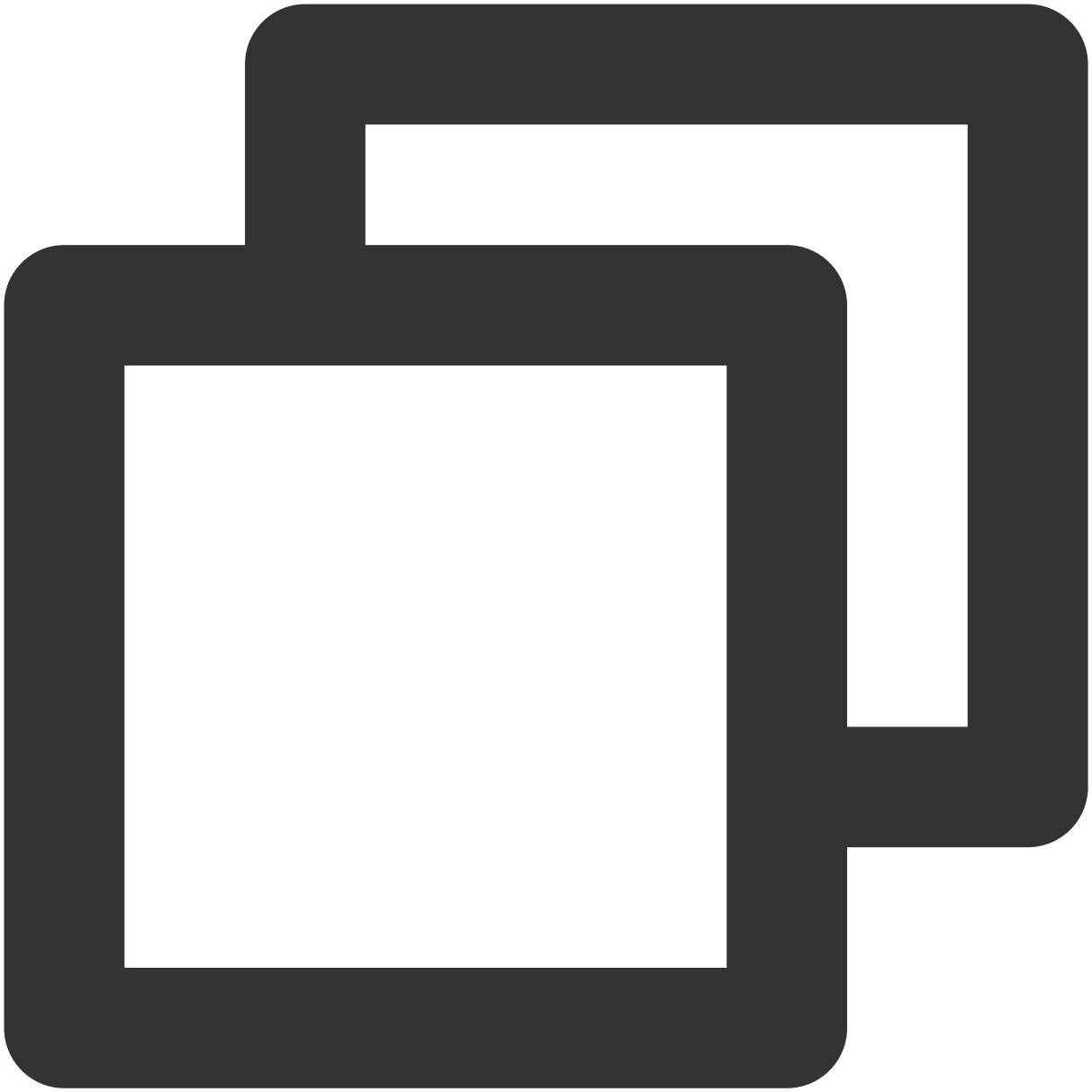
调用此接口停止发送自定义消息。

接口原型

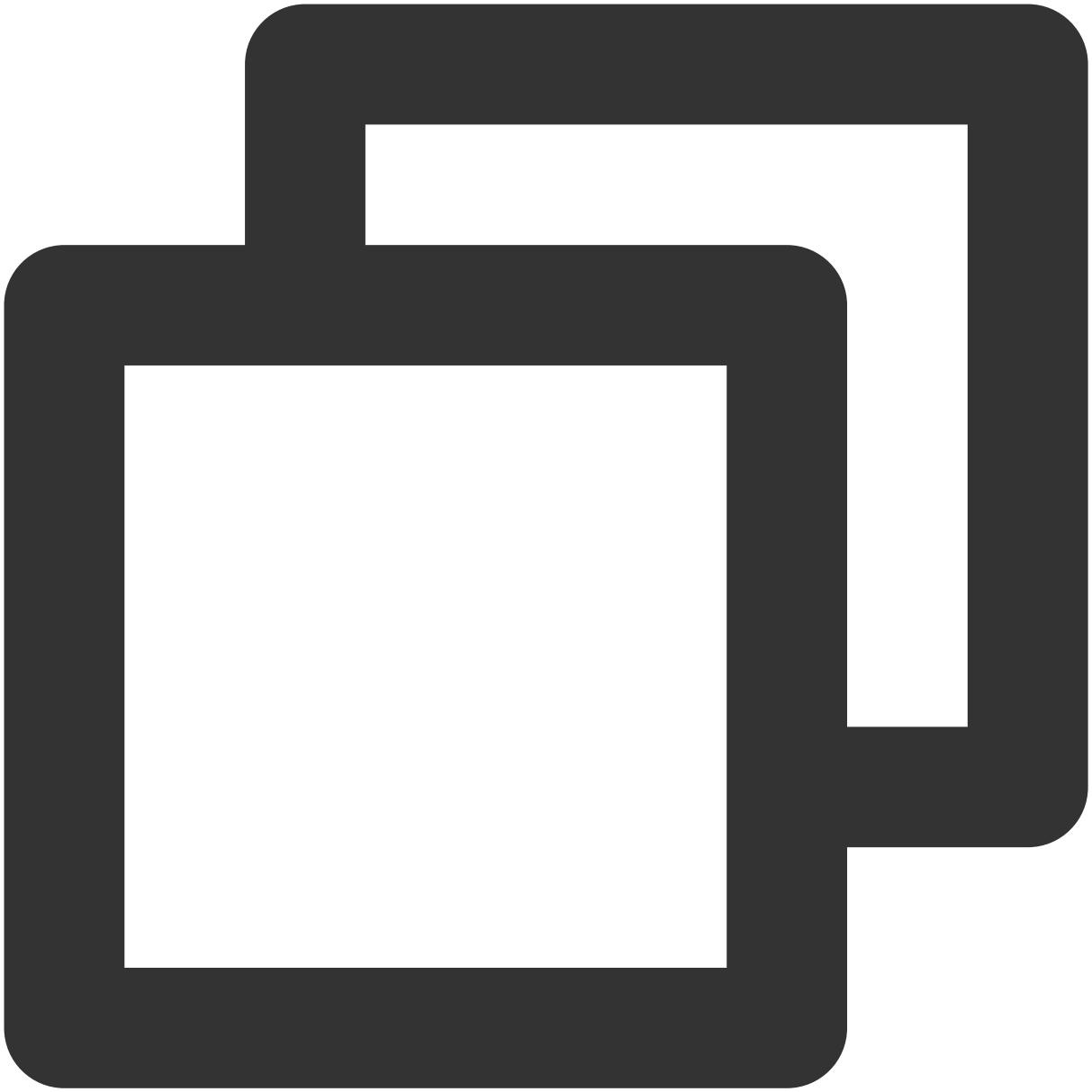
iOS

Android

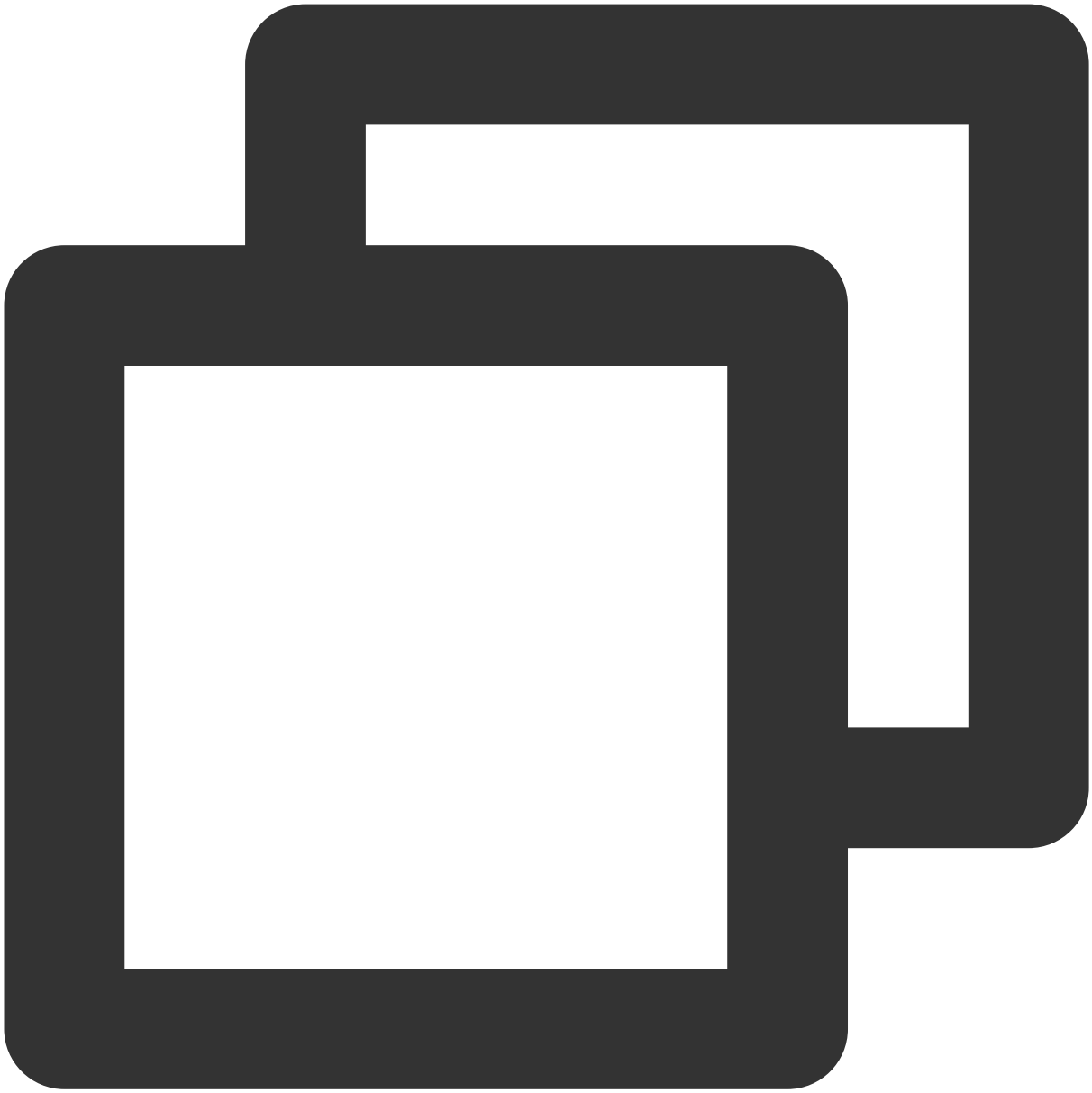
Unity



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



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



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

返回值

如果接口返回1003代表已经操作了 StopSendCustomData，SDK 正在进行这个操作，无需再次调用。

如何应对公司防火墙限制

最近更新时间：2024-01-18 14:32:55

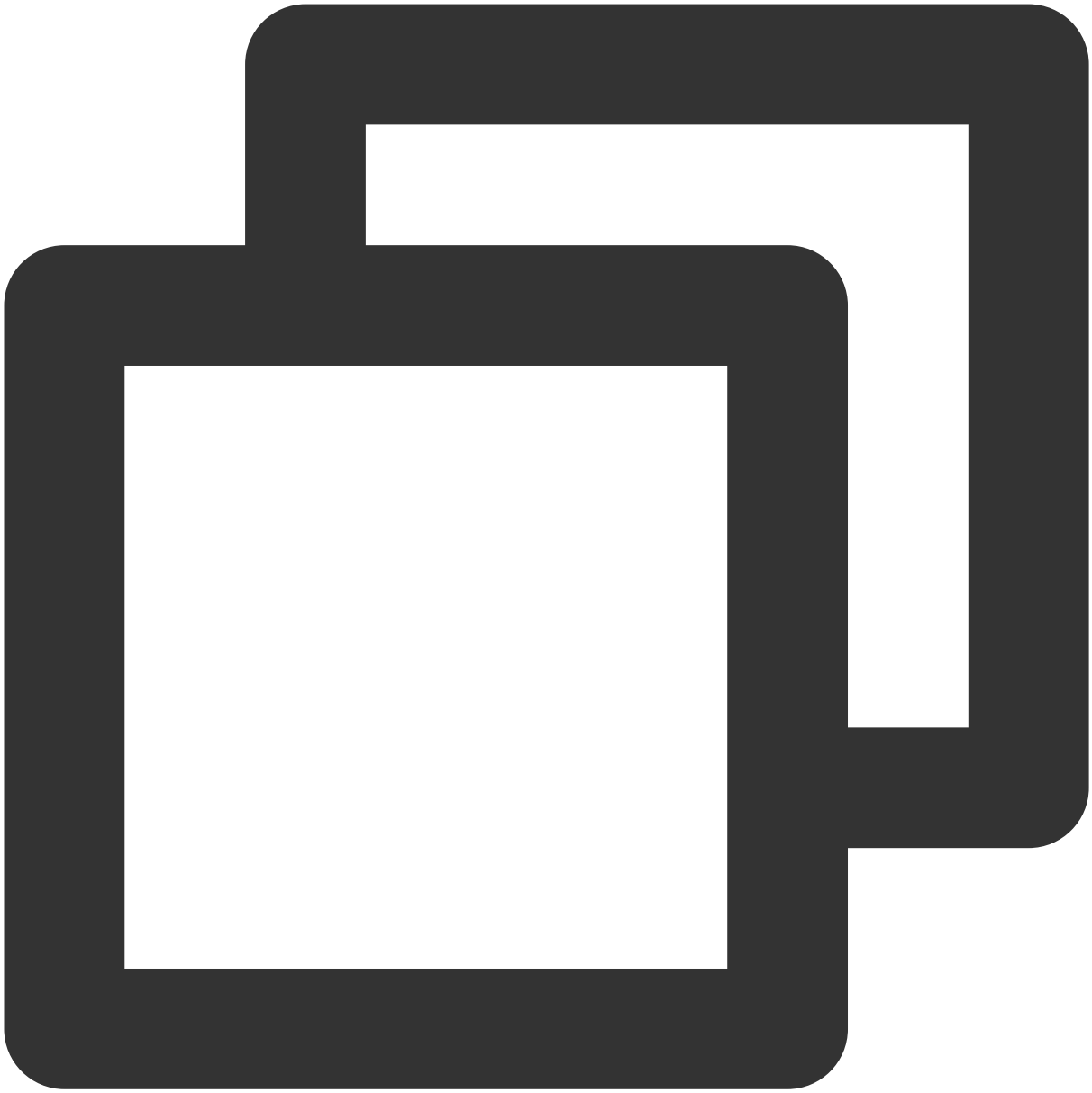
如果公司内部有外网访问限制，需要添加防火墙白名单才能访问，相关的规则如下：

客户端 Native SDK（版本号≥2.2）

防火墙端口：

端口类型	白名单项目
TCP 端口	443
UDP 端口	8000

域名白名单：



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

注意：

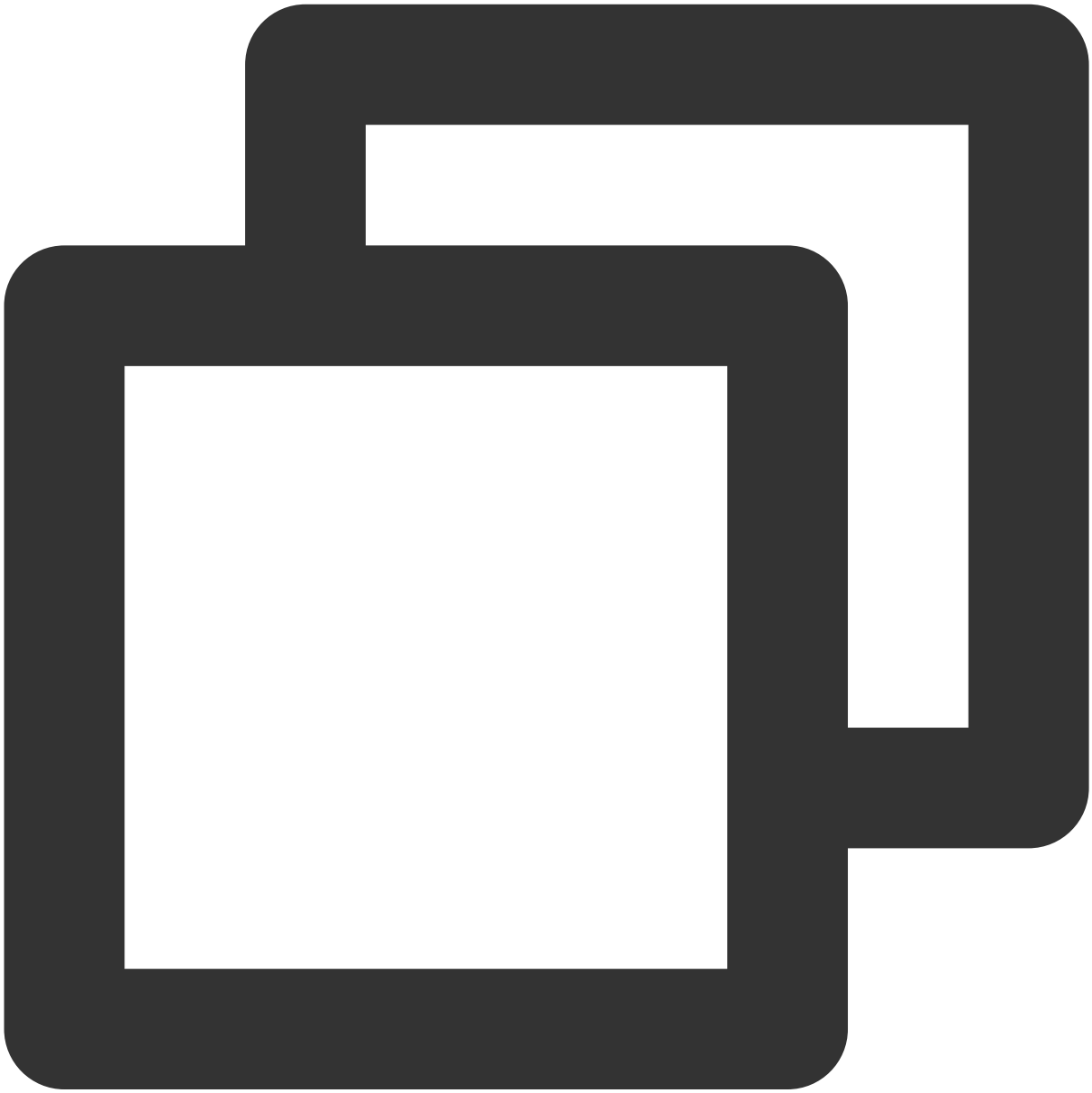
因为腾讯云服务端 IP 地址是动态更新的，并不是固定的一批 IP 地址，所以我们无法提供固定的一组 IP 列表给您。

Windows XP 下使用 GME SDK 还需添加以下防火墙白名单。

防火墙端口：

端口类型	白名单项目
TCP 端口	15000

域名白名单：



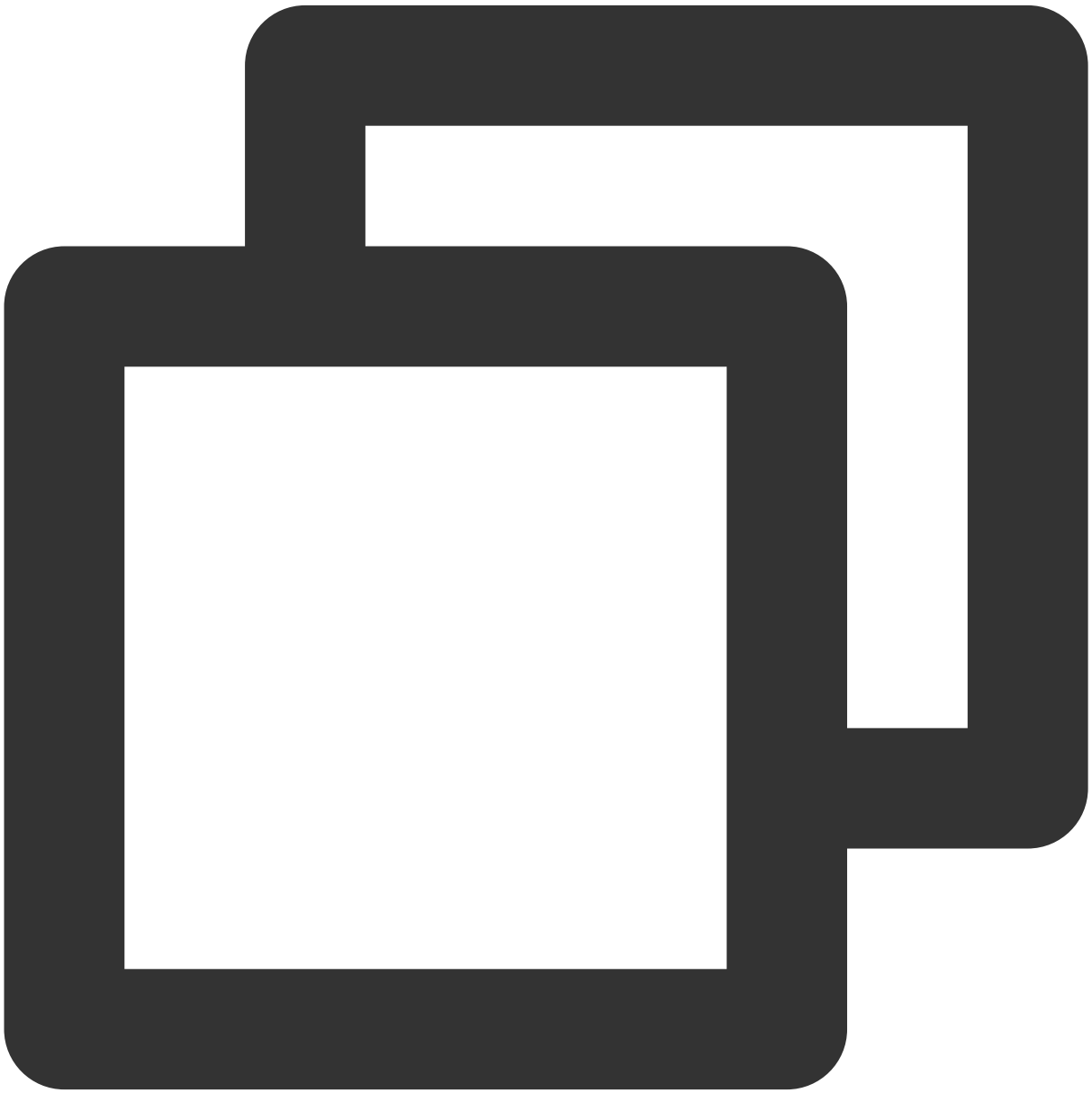
```
cloud.tim.qq.com
openmsf.3g.qq.com
```

使用 H5 SDK

防火墙端口：

端口类型	白名单项目
TCP 端口	443,8687
UDP 端口	8000；8800；843；443

域名白名单：



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

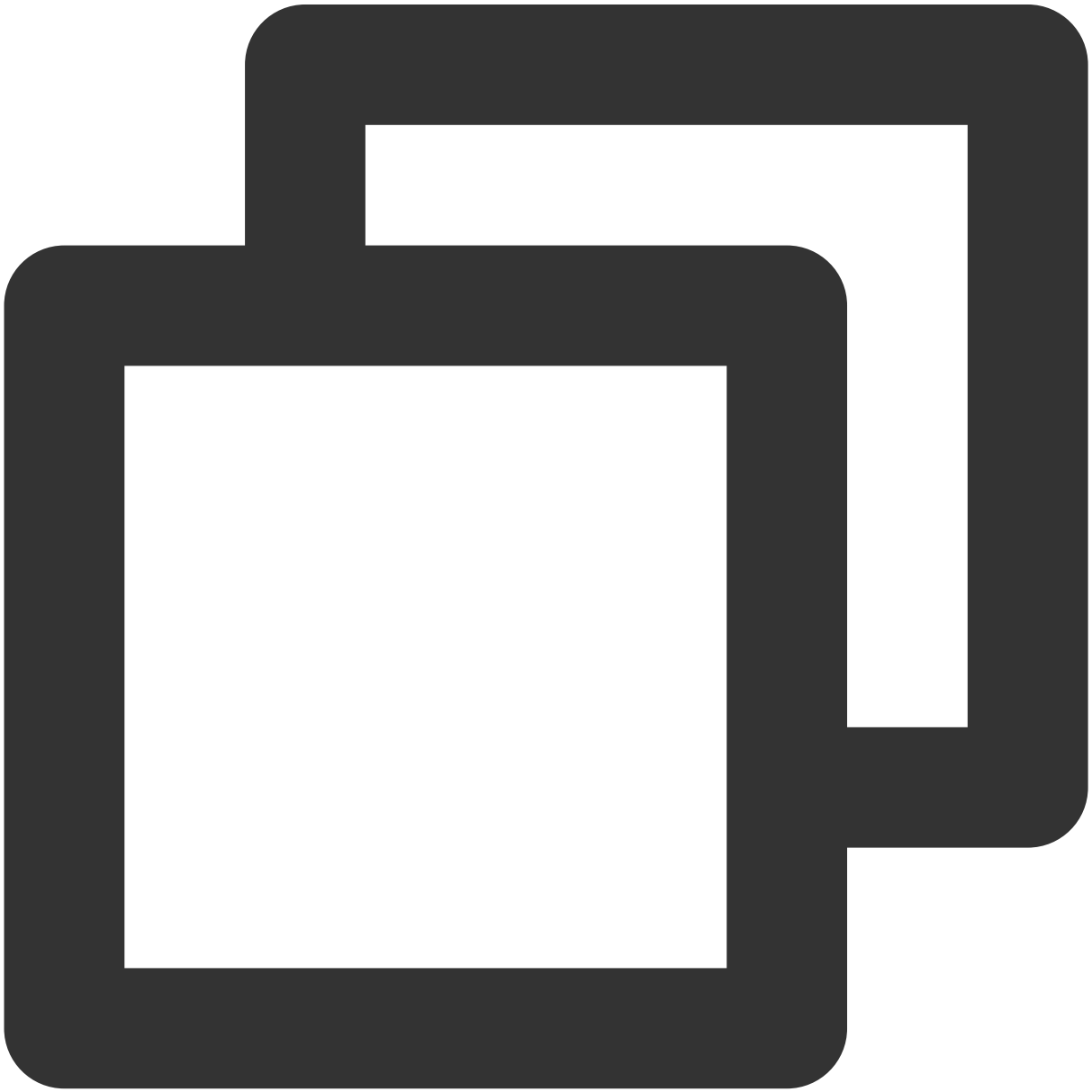
```
rtc.qcloud.qq.com
```

使用语音消息及转文字服务

防火墙端口：

端口类型	白名单项目
TCP 端口	80，443

域名白名单：



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

语言参数参考列表

最近更新时间：2023-04-04 15:35:18

本文主要为您介绍语音转文本服务、文本翻译服务以及文本转语音服务的语言参数参考列表，方便您调试和接入腾讯云游戏多媒体引擎服务。

语音转文本服务、文本翻译、文本转语音

语音转文本服务、文本翻译、文本转语音服务目前支持全语种，文本翻译对应的接口是 TranslateText，文本转语音对应的接口是 TextToSpeech。

注意

应用需使用文本翻译功能，或者文本转语音功能，请 提交工单 申请。

语言	参数	中文翻译
普通话 （中国大陆）	cmn-Hans-CN	中文，普通话（简体中文，中文）
國語 （中国台灣）	cmn-Hant-TW	中文，普通话（繁体，中国台湾）
廣東話 （中国香港）	yue-Hant-HK	中文，广东话（繁体，中国香港）
普通話 （中国香港）	cmn-Hans-HK	中文，普通话（简体，中国香港）
Afrikaans (Suid-Afrika)	af-ZA	南非荷兰语（南非）
አማርኛ (ኢትዮጵያ)	am-ET	阿姆哈拉语（埃塞俄比亚）
Հայ (Հայաստան)	hy-AM	亚美尼亚语（亚美尼亚）
Azərbaycan (Azərbaycan)	az-AZ	阿塞拜疆（阿塞拜疆）
Bahasa Indonesia (Indonesia)	id-ID	印度尼西亚（印度尼西亚）
Bahasa Melayu (Malaysia)	ms-MY	马来语（马来西亚）
বাংলা (বাংলাদেশ)	bn-BD	孟加拉语（孟加拉国）
বাংলা (ভারত)	bn-IN	孟加拉语（印度）
Català (Espanya)	ca-ES	加泰罗尼亚（西班牙）
Čeština (Česká republika)	cs-CZ	捷克（捷克共和国）

Dansk (Danmark)	da-DK	丹麦语（丹麦）
Deutsch (Deutschland)	de-DE	德语（德国）
English (Australia)	en-AU	英语（澳大利亚）
English (Canada)	en-CA	英语（加拿大）
English (Ghana)	en-GH	英语（加纳）
English (Great Britain)	en-GB	英语（英国）
English (India)	en-IN	英文（印度）
English (Ireland)	en-IE	英语（爱尔兰）
English (Kenya)	en-KE	英语（肯尼亚）
English (New Zealand)	en-NZ	英语（新西兰）
English (Nigeria)	en-NG	英语（尼日利亚）
English (Philippines)	en-PH	英语（菲律宾）
English (South Africa)	en-ZA	英语（南非）
English (Tanzania)	en-TZ	英语（坦桑尼亚）
English (United States)	en-US	英语（美国）
Español (Argentina)	es-AR	西班牙语
Español (Bolivia)	es-BO	西班牙语（玻利维亚）
Español (Chile)	es-CL	西班牙语（智利）
Español (Colombia)	es-CO	西班牙语（哥伦比亚）
Español (Costa Rica)	es-CR	西班牙语（哥斯达黎加）
Español (Ecuador)	es-EC	西班牙语（厄瓜多尔）
Español (El Salvador)	es-SV	西班牙语（萨尔瓦多）
Español (España)	es-ES	西班牙语（西班牙）
Español (Estados Unidos)	es-US	西班牙语（美国）
Español (Guatemala)	es-GT	西班牙语（危地马拉）

Español (Honduras)	es-HN	西班牙语（洪都拉斯）
Español (México)	es-MX	西班牙语（墨西哥）
Español (Nicaragua)	es-NI	西班牙语（尼加拉瓜）
Español (Panamá)	es-PA	西班牙语（巴拿马）
Español (Paraguay)	es-PY	西班牙语（巴拉圭）
Español (Perú)	es-PE	西班牙语（秘鲁）
Español (Puerto Rico)	es-PR	西班牙语（波多黎各）
Español (República Dominicana)	es-DO	西班牙语（多米尼加共和国）
Español (Uruguay)	es-UY	西班牙语（乌拉圭）
Español (Venezuela)	es-VE	西班牙语（委内瑞拉）
Euskara (Espainia)	eu-ES	巴斯克（西班牙）
Filipino (Pilipinas)	fil-PH	菲律宾（菲律宾）
Français (Canada)	fr-CA	法语（加拿大）
Français (France)	fr-FR	法国（法国）
Galego (España)	gl-ES	加利西亚（西班牙）
ქართული (საქართველო)	ka-GE	格鲁吉亚（格鲁吉亚）
ગુજરાતી (ભારત)	gu-IN	古吉拉特语（印度）
Hrvatski (Hrvatska)	hr-HR	克罗地亚（克罗地亚）
IsiZulu (Ningizimu Afrika)	zu-ZA	祖鲁（南非）
Íslenska (Ísland)	is-IS	冰岛（冰岛）
Italiano (Italia)	it-IT	意大利（意大利）
Jawa (Indonesia)	jv-ID	爪哇（印度尼西亚）
( )	kn-IN	卡纳达（印度）
( )	km-KH	高棉（柬埔寨）
( )	lo-LA	老挝（老挝）

Latviešu (latviešu)	lv-LV	拉脱维亚语（拉脱维亚）
Lietuvių (Lietuva)	lt-LT	立陶宛语（立陶宛）
Magyar (Magyarország)	hu-HU	匈牙利语（匈牙利）
മലയാളം (ഇന്ത്യ)	ml-IN	马拉雅拉姆语（印度）
मराठी (भारत)	mr-IN	马拉地语（印度）
Nederlands (Nederland)	nl-NL	荷兰（荷兰）
नेपाली (नेपाल)	ne-NP	尼泊尔语（尼泊尔）
Norsk bokmål (Norge)	nb-NO	挪威语Bokmål（挪威）
Polski (Polska)	pl-PL	波兰语（波兰）
Português (Brasil)	pt-BR	葡萄牙语（巴西）
Português (Portugal)	pt-PT	葡萄牙（葡萄牙）
Română (România)	ro-RO	罗马尼亚（罗马尼亚）
සිංහල (ශ්‍රී ලංකාව)	si-LK	僧伽罗语（斯里兰卡）
Slovenčina (Slovensko)	sk-SK	斯洛伐克（斯洛伐克）
Slovenščina (Slovenija)	sl-SI	斯洛文尼亚（斯洛文尼亚）
Urang (Indonesia)	su-ID	Sundanese（印度尼西亚）
Swahili (Tanzania)	sw-TZ	斯瓦希里语（坦桑尼亚）
Swahili (Kenya)	sw-KE	斯瓦希里语（肯尼亚）
Suomi (Suomi)	fi-FI	芬兰语（芬兰）
Svenska (Sverige)	sv-SE	瑞典语（瑞典）
தமிழ் (இந்தியா)	ta-IN	泰米尔语（印度）
தமிழ் (சிங்கப்பூர்)	ta-SG	泰米尔语（新加坡）
தமிழ் (இலங்கை)	ta-LK	泰米尔语（斯里兰卡）
தமிழ் (மலேசியா)	ta-MY	泰米尔语（马来西亚）
( )	te-IN	泰卢固语（印度）

Tiếng Việt (Việt Nam)	vi-VN	越南语（越南）
Türkçe (Türkiye)	tr-TR	土耳其语（土耳其）
(اردو (پاکستان	ur-PK	乌尔都语（巴基斯坦）
(اردو (بھارت	ur-IN	乌尔都语（印度）
Ελληνικά (Ελλάδα)	el-GR	希腊语（希腊）
Български (България)	bg-BG	保加利亚语（保加利亚）
Русский (Россия)	ru-RU	俄罗斯（俄罗斯）
Српски (Србија)	sr-RS	塞尔维亚语（塞尔维亚）
Українська (Україна)	uk-UA	乌克兰语（乌克兰）
(עברית (ישראל	he-IL	希伯来语（以色列）
(العربية (إسرائيل	ar-IL	阿拉伯语（以色列）
(العربية (الأردن	ar-JO	阿拉伯语（约旦）
(العربية (الإمارات	ar-AE	阿拉伯语（阿拉伯联合酋长国）
(العربية (البحرين	ar-BH	阿拉伯语（巴林）
(العربية (الجزائر	ar-DZ	阿拉伯语（阿尔及利亚）
(العربية (السعودية	ar-SA	阿拉伯语（沙特阿拉伯）
(العربية (العراق	ar-IQ	阿拉伯语（伊拉克）
(العربية (الكويت	ar-KW	阿拉伯语（科威特）
(العربية (المغرب	ar-MA	阿拉伯语（摩洛哥）
(العربية (تونس	ar-TN	阿拉伯语（突尼斯）
(العربية (عُمان	ar-OM	阿拉伯语（阿曼）
(العربية (فلسطين	ar-PS	阿拉伯语（巴勒斯坦国）
(العربية (قطر	ar-QA	阿拉伯语（卡塔尔）
(العربية (لبنان	ar-LB	阿拉伯语（黎巴嫩）
(العربية (مصر	ar-EG	阿拉伯语（埃及）

(ایران)فارسی	fa-IR	波斯语（伊朗）
हिन्दी (भारत)	hi-IN	印地语（印度）
ไทย (ประเทศไทย)	th-TH	泰国（泰国）
한국어 (대한민국)	ko-KR	韩国（韩国）

房间管理功能

最近更新时间：2023-04-27 17:17:14

注意：GME 3.x 版本暂不支持房间管理功能。
为方便开发者快速接入房间管理服务，这里向您介绍房间管理服务的使用场景以及接入流程。

功能简介

通过客户端房间管理接口，可以简单实现对房间内成员的管理、以及对房间内成员上下麦的管理。

使用场景

例如在狼人杀场景中，作为主持人可以通过 EnableMic 控制其他玩家打开麦克风发言；当某位玩家已经“死亡”，不需要听房间内声音或者操作麦克风说话，则通过 ForbidUserOperation 接口禁止该玩家操作设备。

前提条件

已开通实时语音服务：可参见 语音服务开通指引。
已接入GME SDK：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入、Unity SDK 快速接入、Unreal SDK 快速接入。
说明
此功能不支持 H5 SDK。

接入流程

类名：ITMGRoomManager

GME 房间管理功能在进房后才可调用，且只能修改房间内成员的状态。
所有接口的结果会通过 `ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR` 回调，回调详情请参考 回调处理。

接口列表

类型	接口
控制采集	EnableMic、 EnableAudioCaptureDevice、EnableAudioSend

控制播放	EnableSpeaker、 EnableAudioPlayDevice、EnableAudioRecv
设备状态获取	GetMicState、 GetSpeakerState
敏感接口	ForbidUserOperation

采集管理相关接口

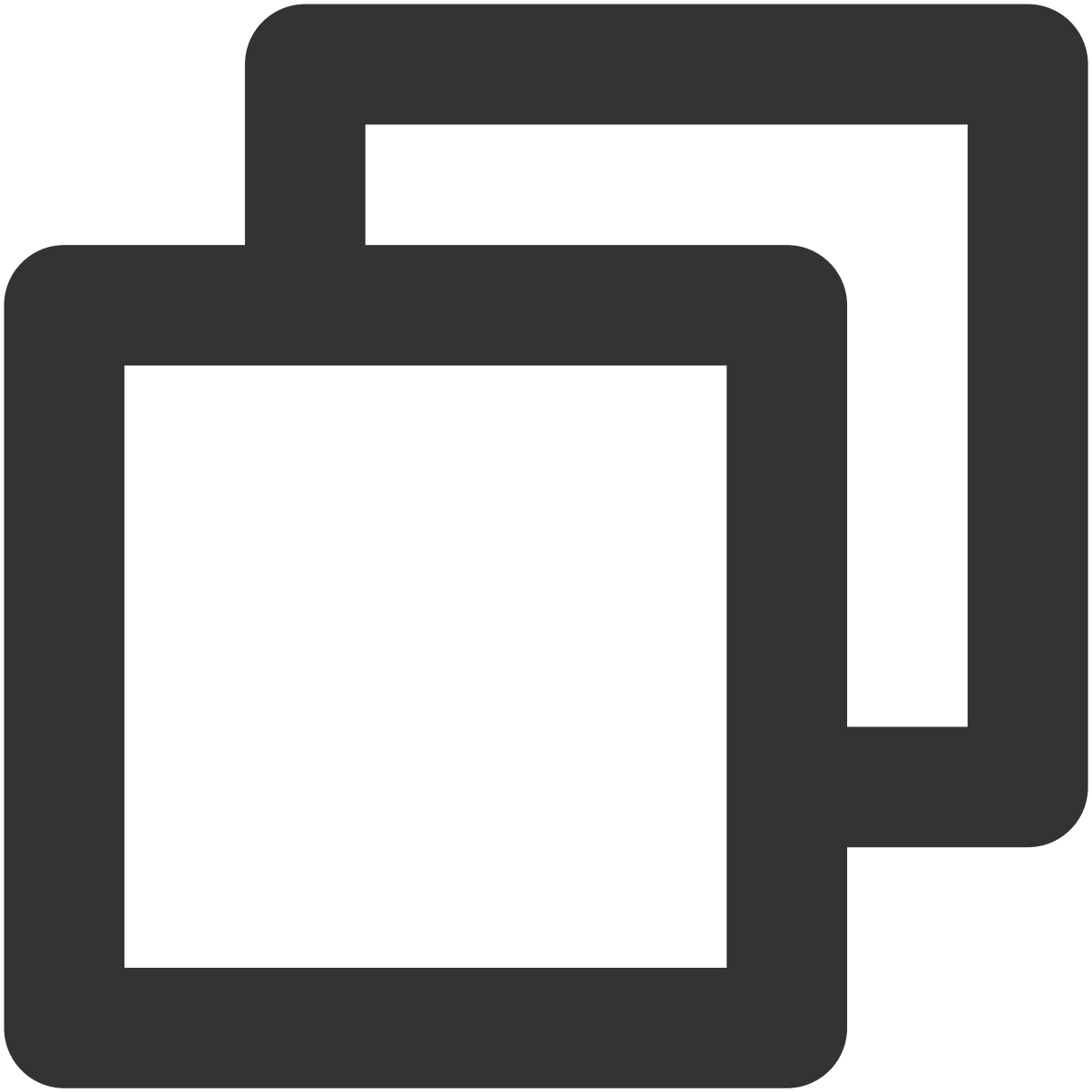
采集管理接口包含**麦克风管理**、**音频上行管理**以及**采集硬件设备管理**。其中麦克风管理相当于音频上行管理加上采集硬件设备管理。

采集管理

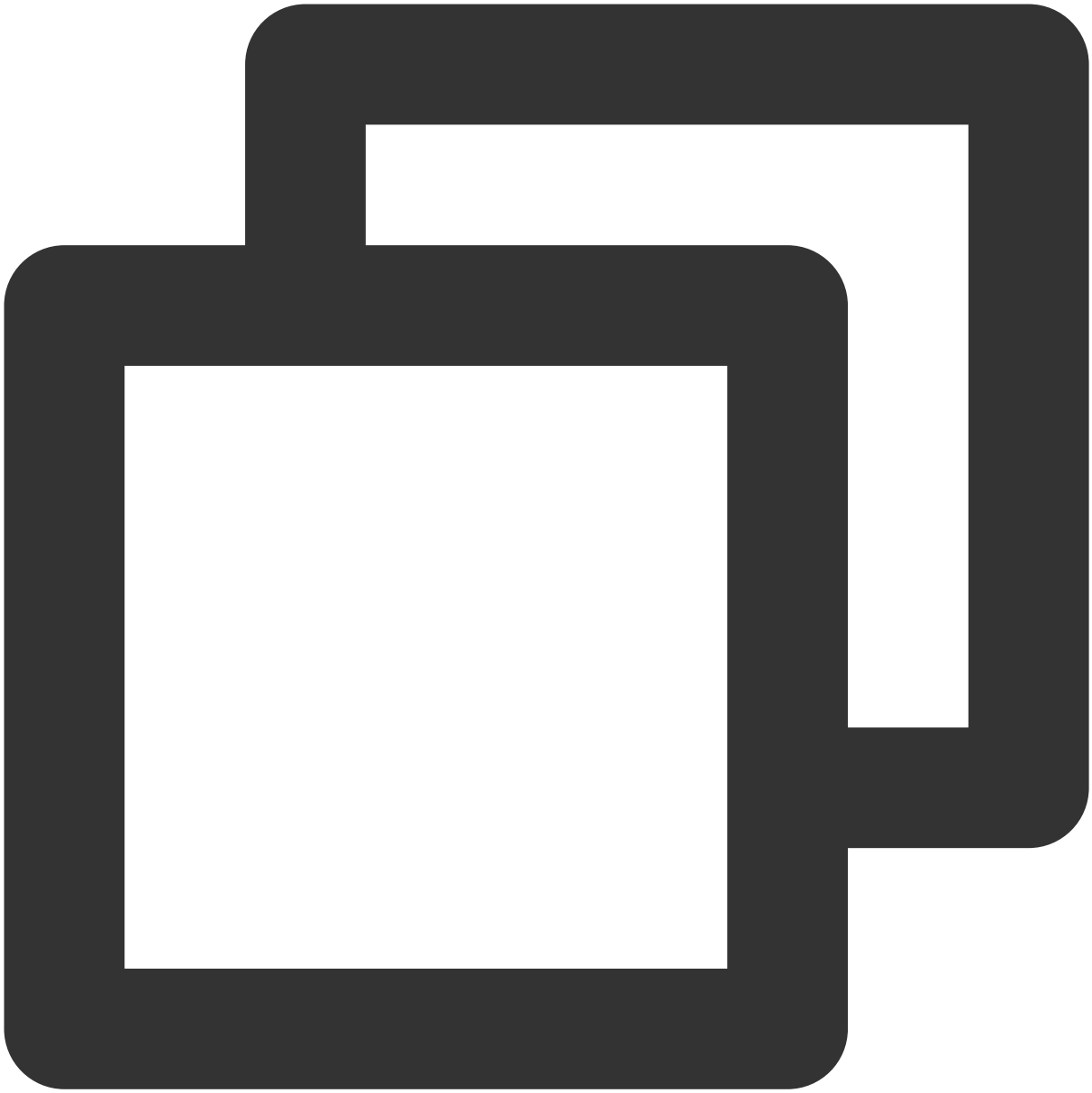
调用此接口打开或关闭房间内某用户的麦克风，调用成功后，该用户的麦克风将被关闭或打开。
EnableMic 相当于同时调用 EnableAudioSend 及 EnableAudioCaptureDevice。

函数原型

Android
iOS



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



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

参数	类型	含义
enable	BOOL	YES：打开某用户麦克风，NO：关闭某用户麦克风
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_MIC_OP。

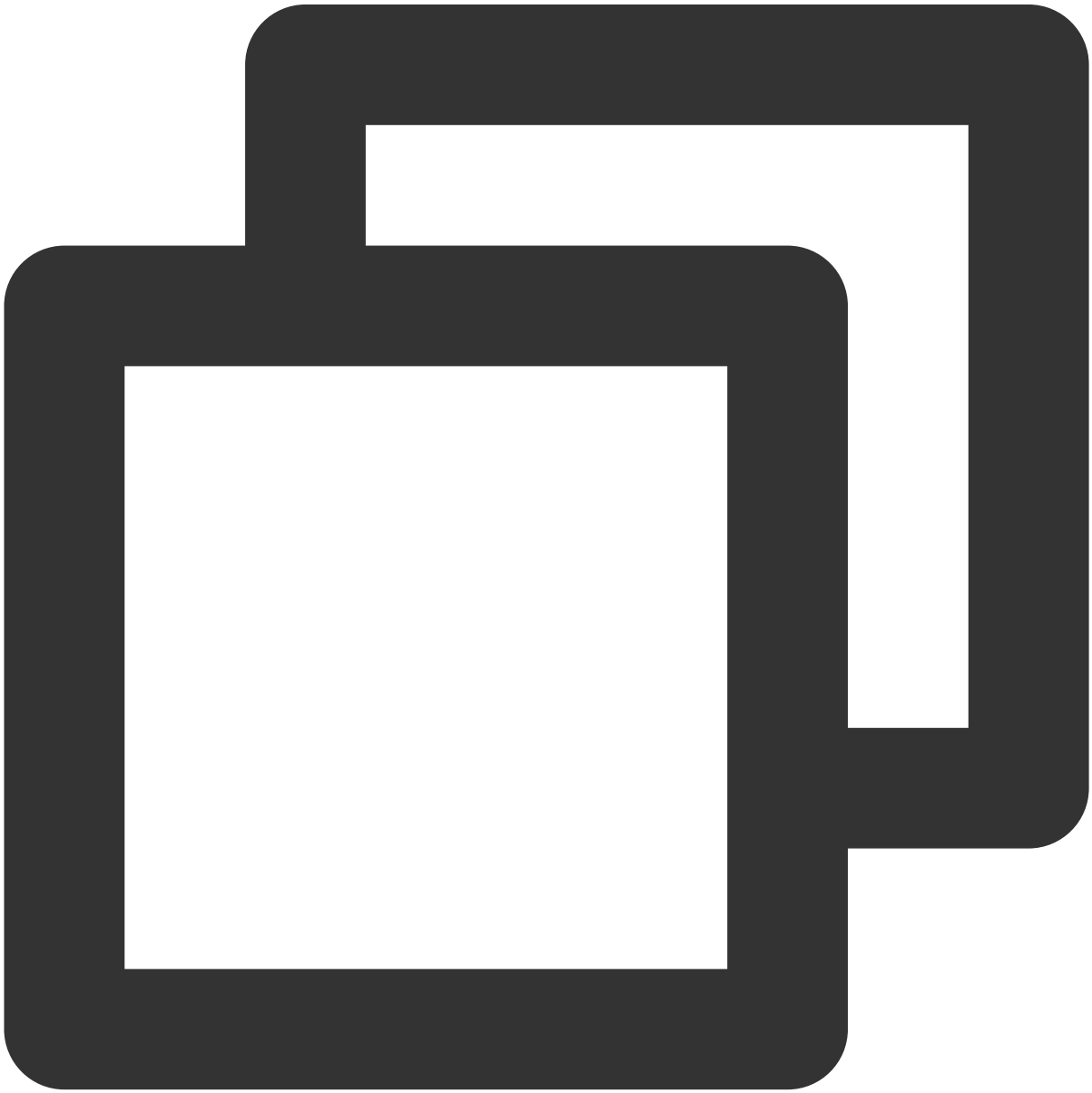
音频流发送管理

调用此接口打开或关闭房间内某用户的音频上行。调用成功后，该用户的音频上行将被关闭或打开，但不影响麦克风采集。

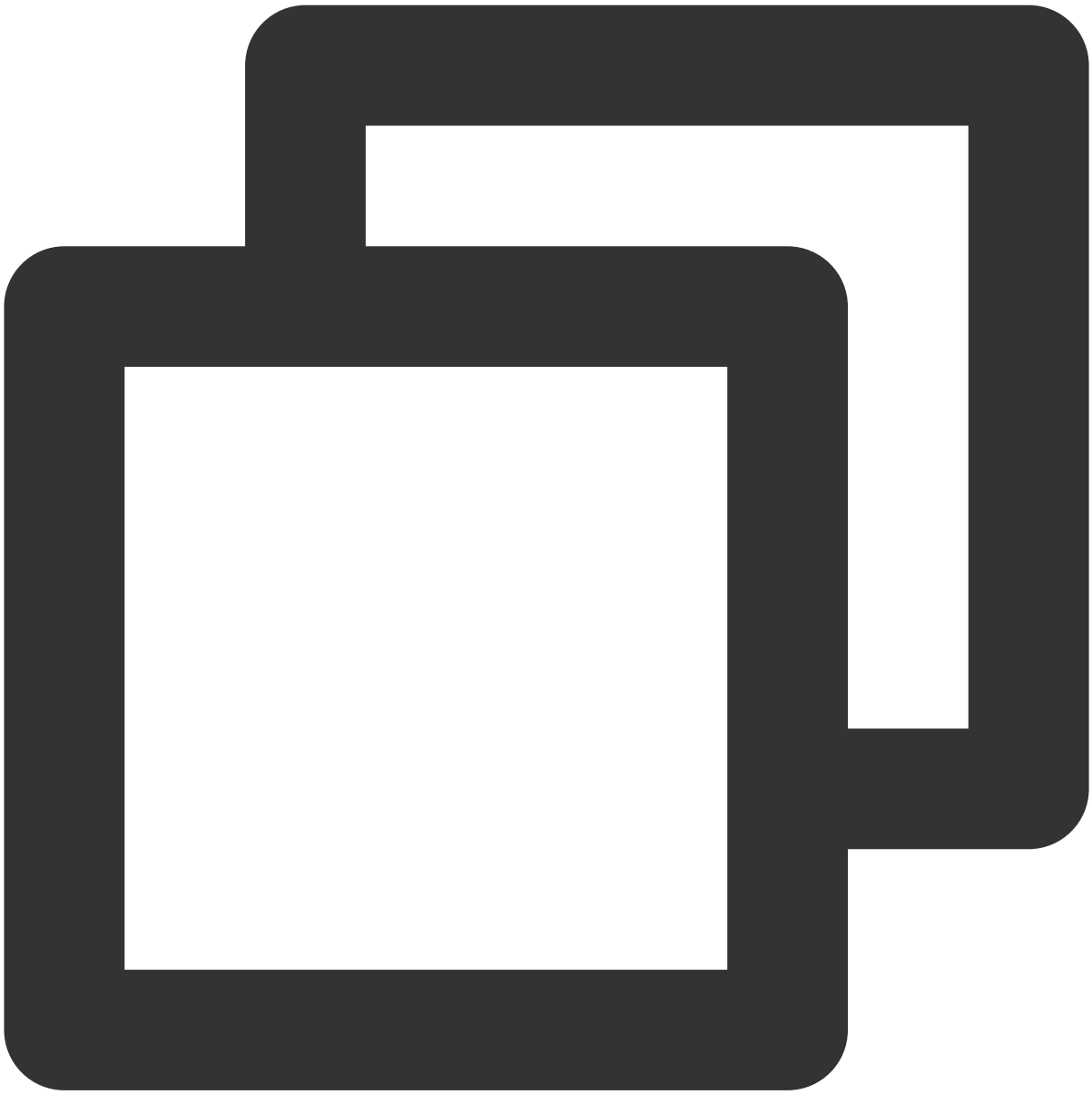
函数原型

Android

iOS



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



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

参数	类型	含义
enable	BOOL	YES：打开某用户上行，NO：关闭某用户上行
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP。

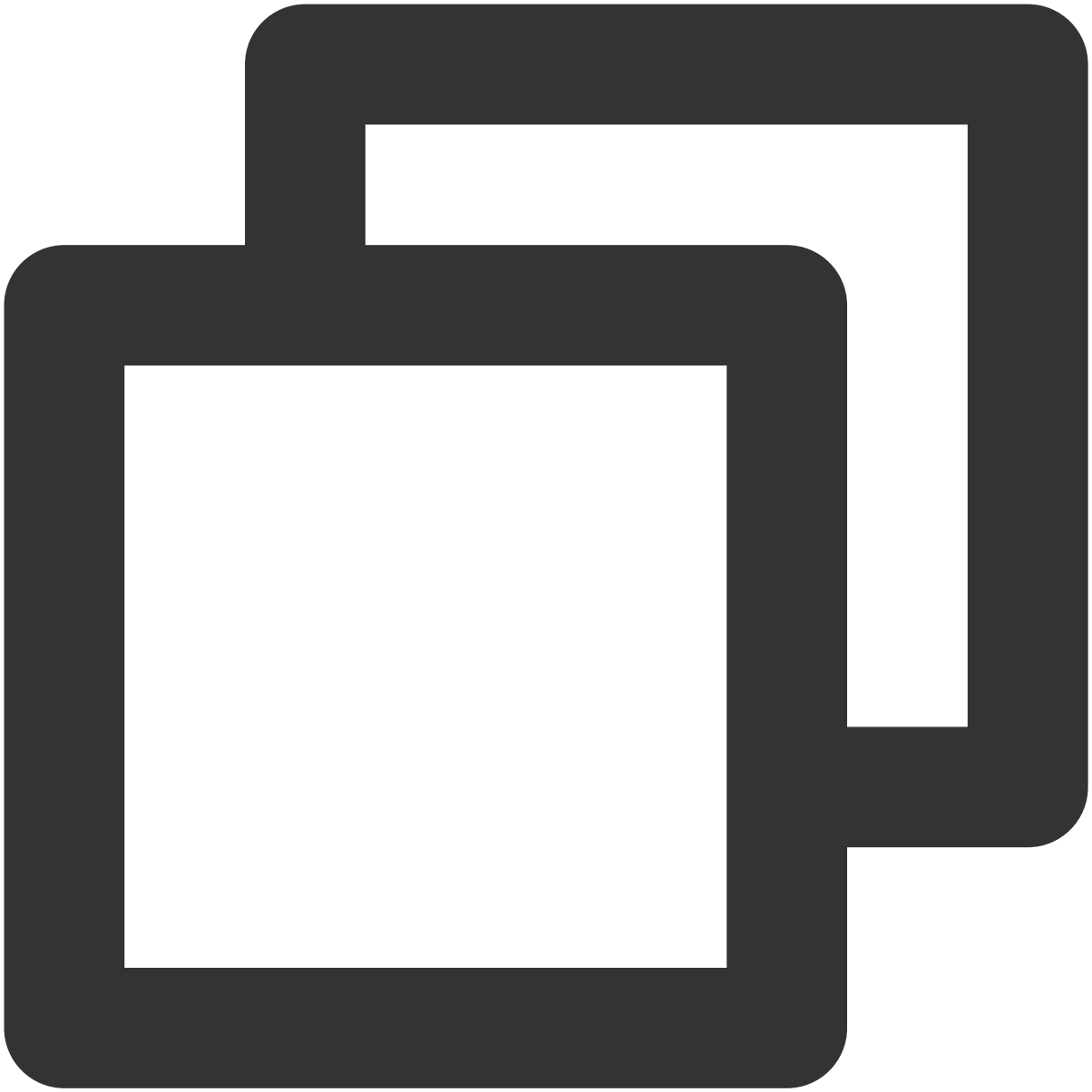
音频采集硬件管理

调用此接口打开或关闭房间内某用户的音频采集硬件设备。调用成功后，该用户的音频采集硬件设备将被关闭或打开，但不影响上行。

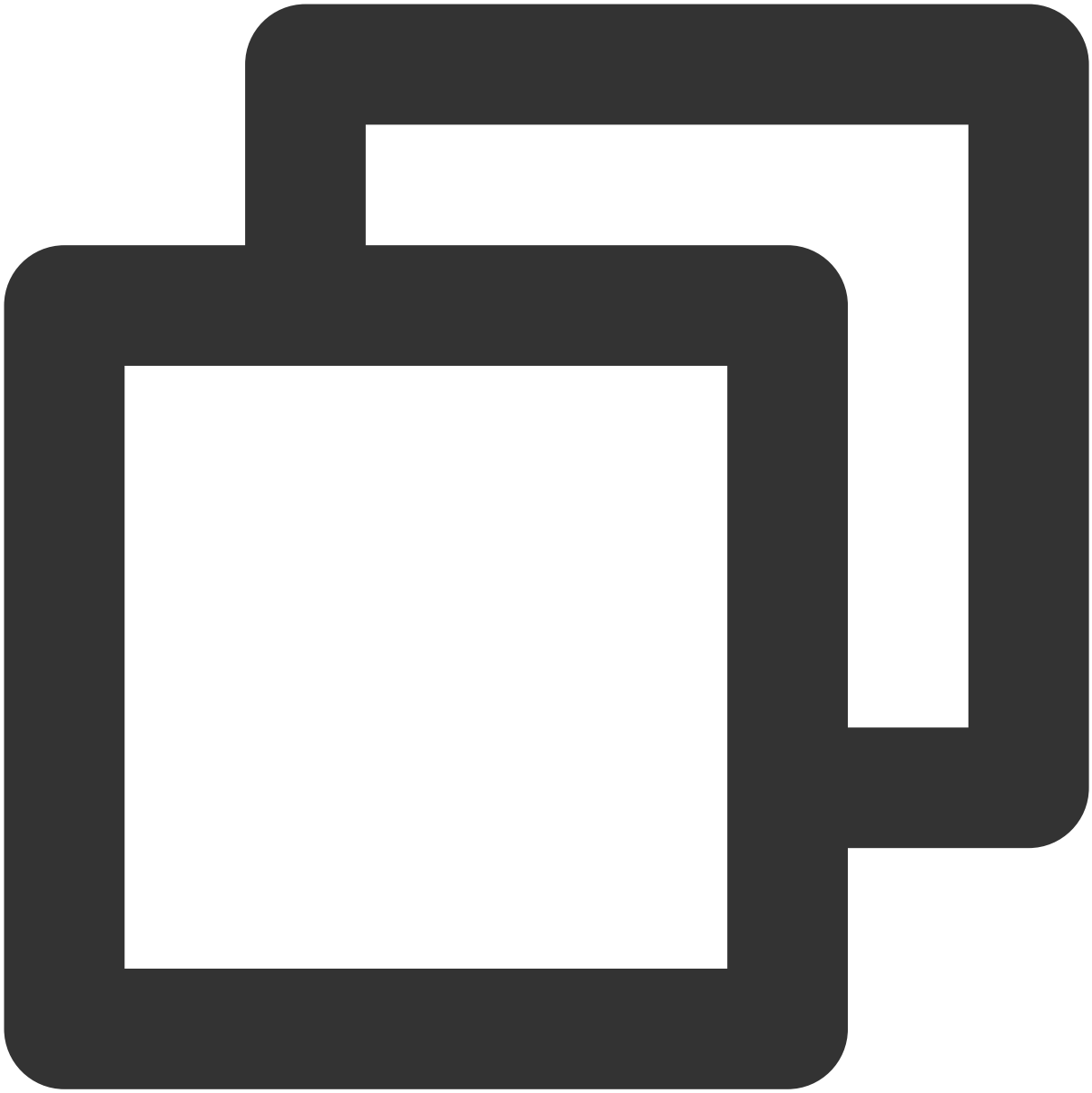
函数原型

Android

iOS



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



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

参数	类型	含义
enable	BOOL	YES：打开某用户音频采集硬件设备，NO：关闭某用户音频采集硬件设备
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_CAPTURE_OP。

播放管理相关接口

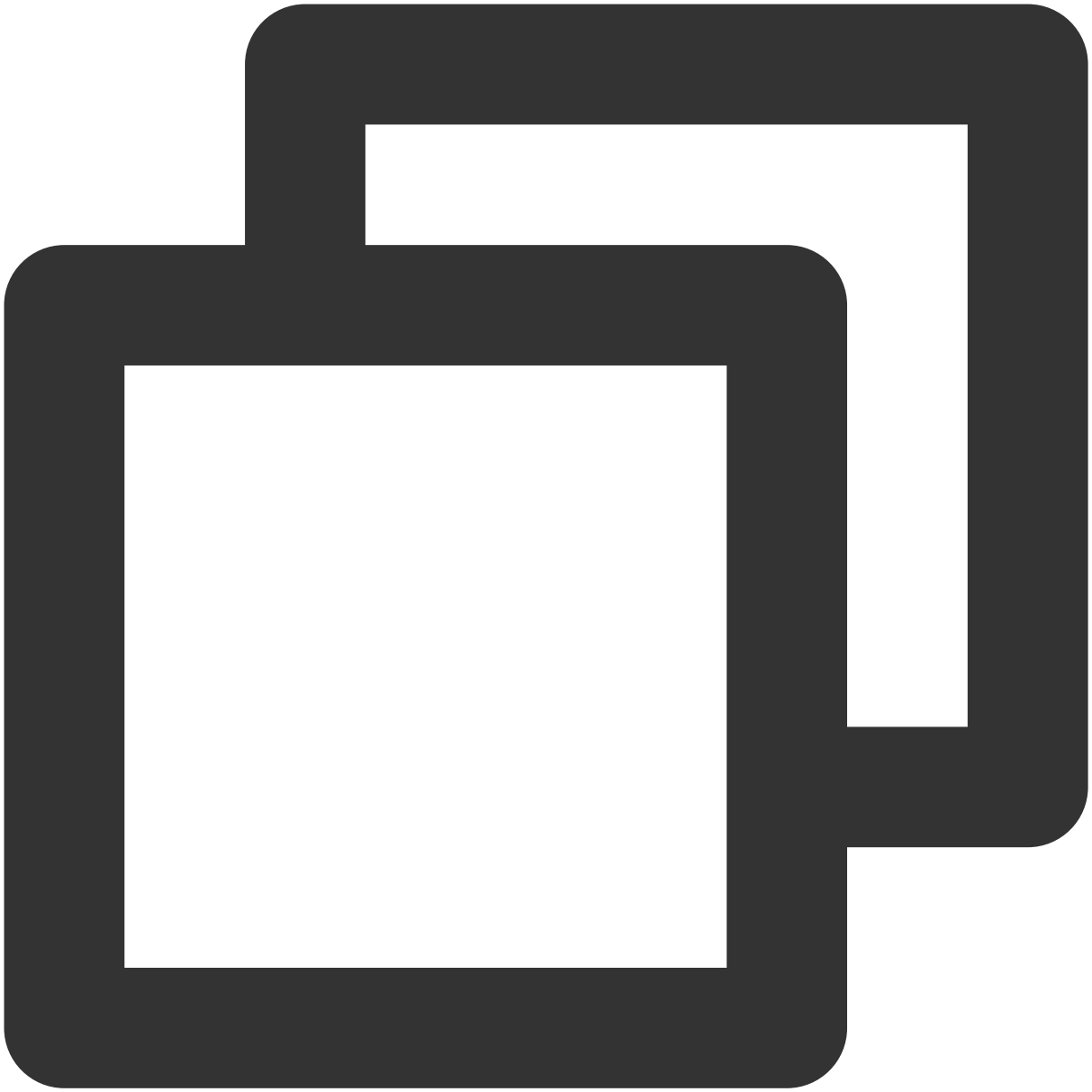
播放管理接口包含**扬声器管理**、**音频下行管理**以及**播放硬件设备管理**。其中扬声器管理相当于音频下行管理加上播放硬件设备管理。

播放管理

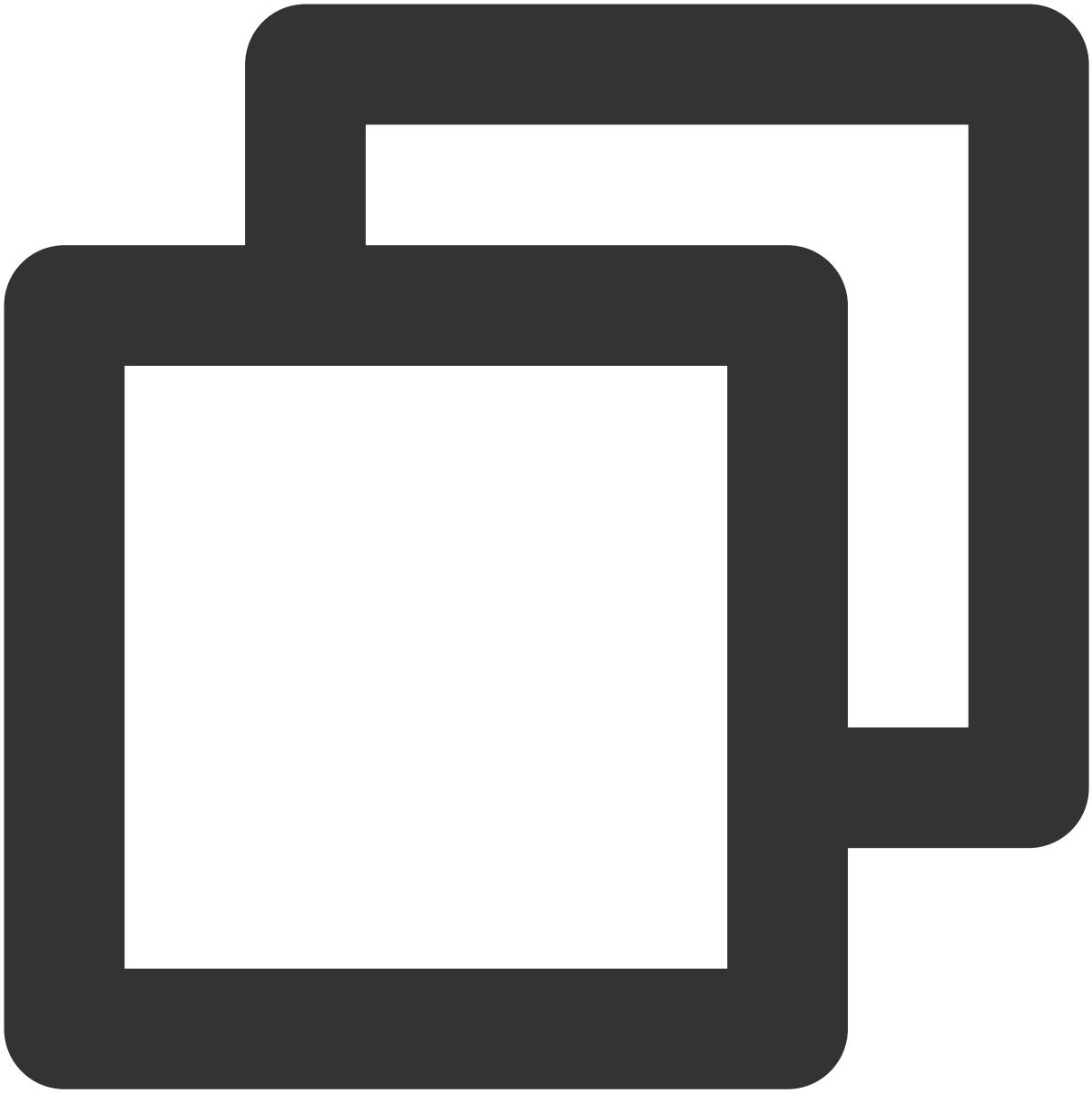
调用此接口打开或关闭房间内某用户的扬声器。调用成功后，该用户的扬声器将被关闭或打开，听到房间内的音频声音。
EnableSpeaker 相当于同时调用 EnableAudioRecv 及 EnableAudioPlayDevice。

函数原型

Android
iOS



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



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

参数	类型	含义
enable	BOOL	YES ：打开某用户扬声器，NO：关闭某用户扬声器
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_SPEAKER_OP。

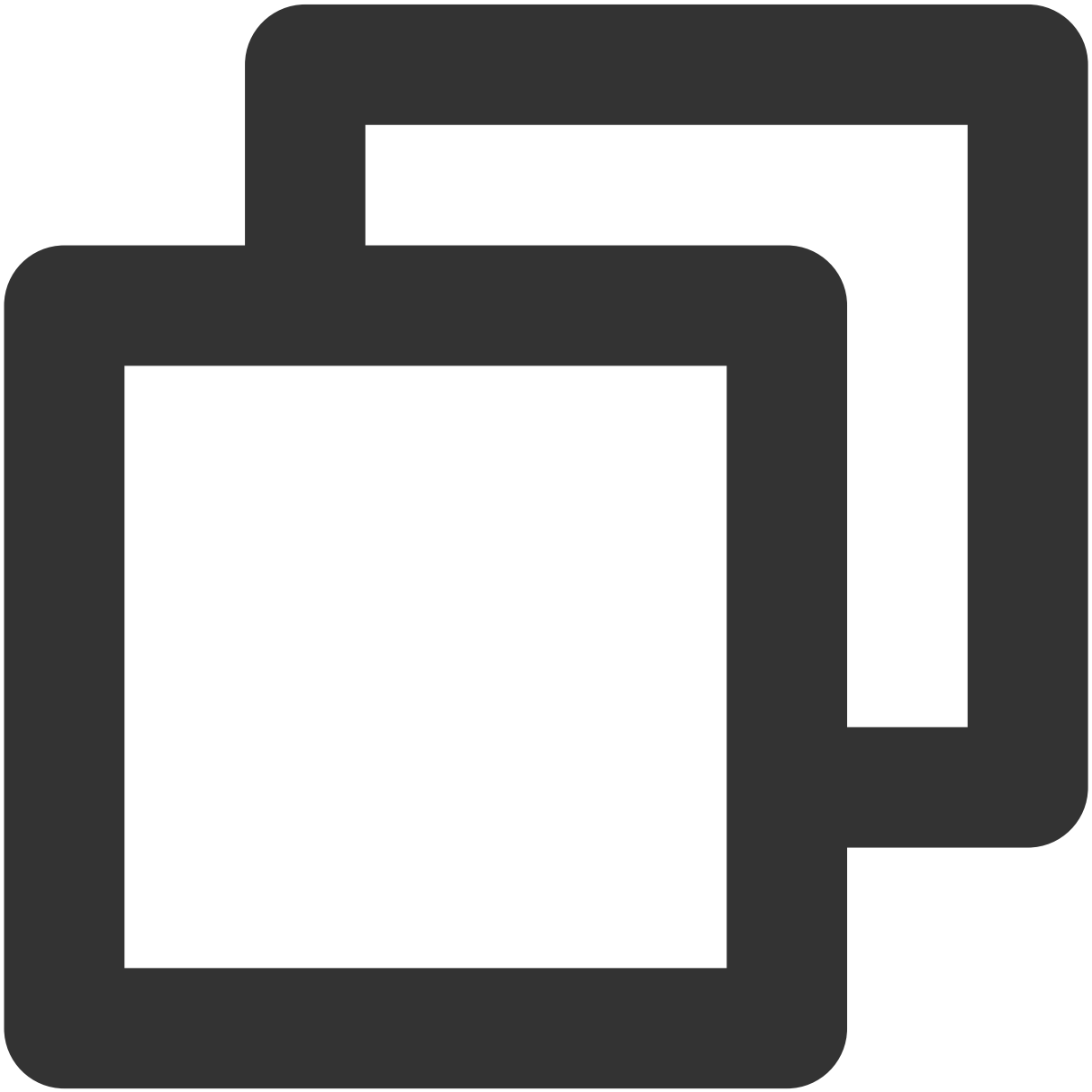
音频流接收管理

调用此接口打开或关闭房间内某用户的音频下行。调用成功后，该用户的音频下行将被关闭或打开，但不影响播放设备。

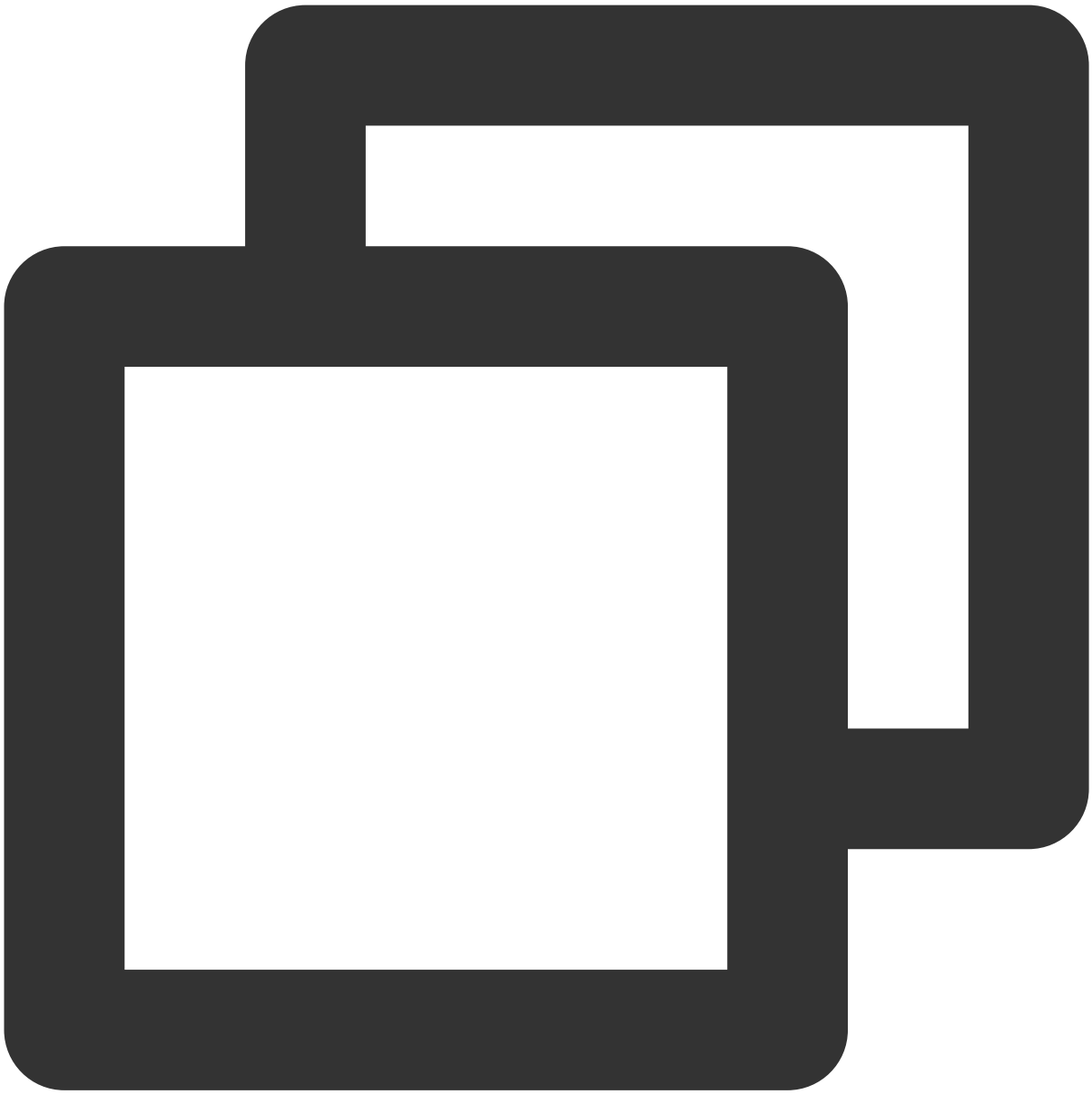
函数原型

Android

iOS



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



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

参数	类型	含义
enable	BOOL	YES ：打开某用户音频下行，NO：关闭某用户音频下行
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP。

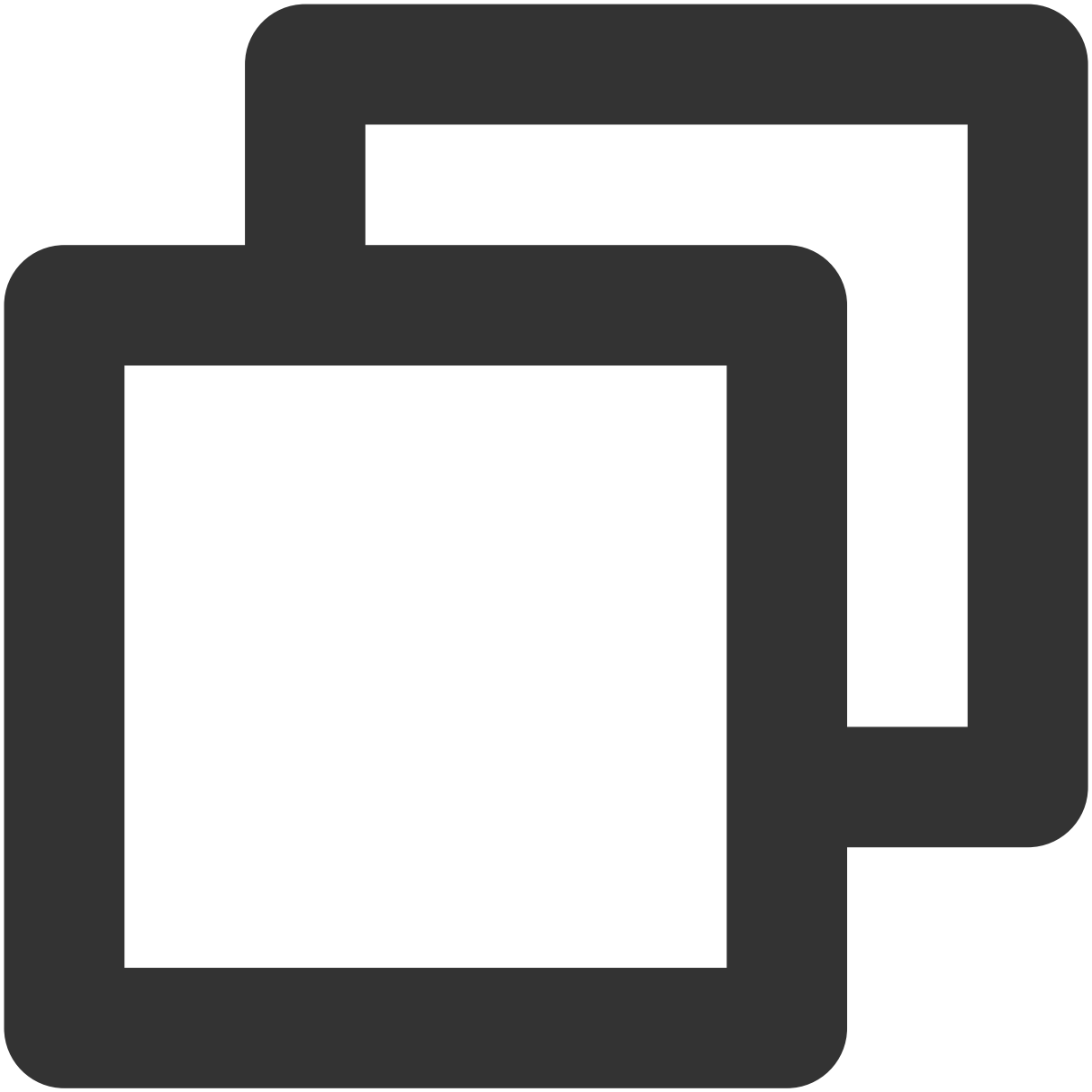
音频播放硬件管理

调用此接口打开或关闭房间内某用户的音频播放硬件设备。调用成功后，该用户的音频播放硬件设备将被关闭或打开，但不影响下行。

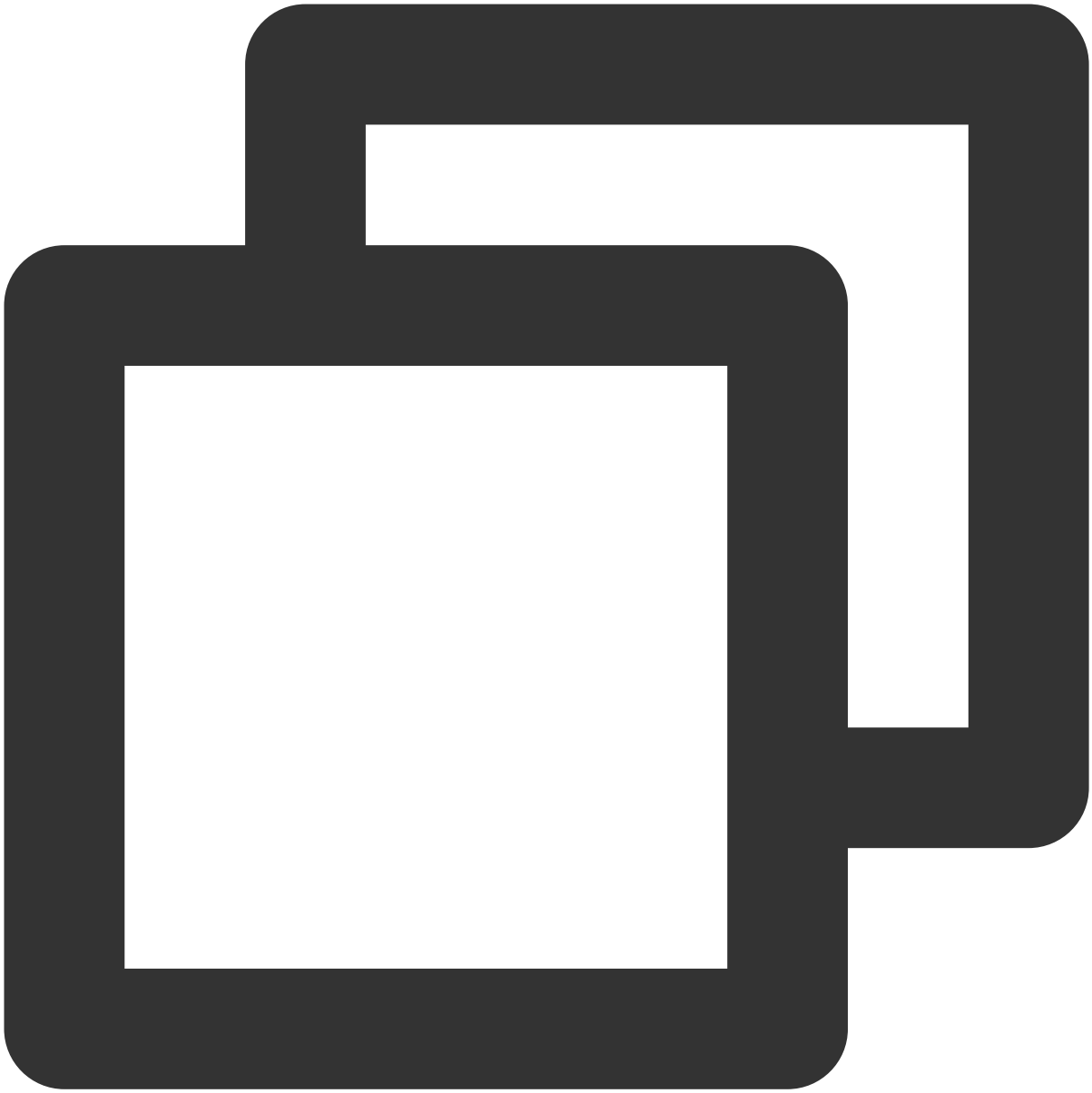
函数原型

Android

iOS



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



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

参数	类型	含义
enable	BOOL	YES ：打开某用户音频播放硬件设备，NO：关闭某用户音频播放硬件设备
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_PLAY_OP。

获取成员状态接口

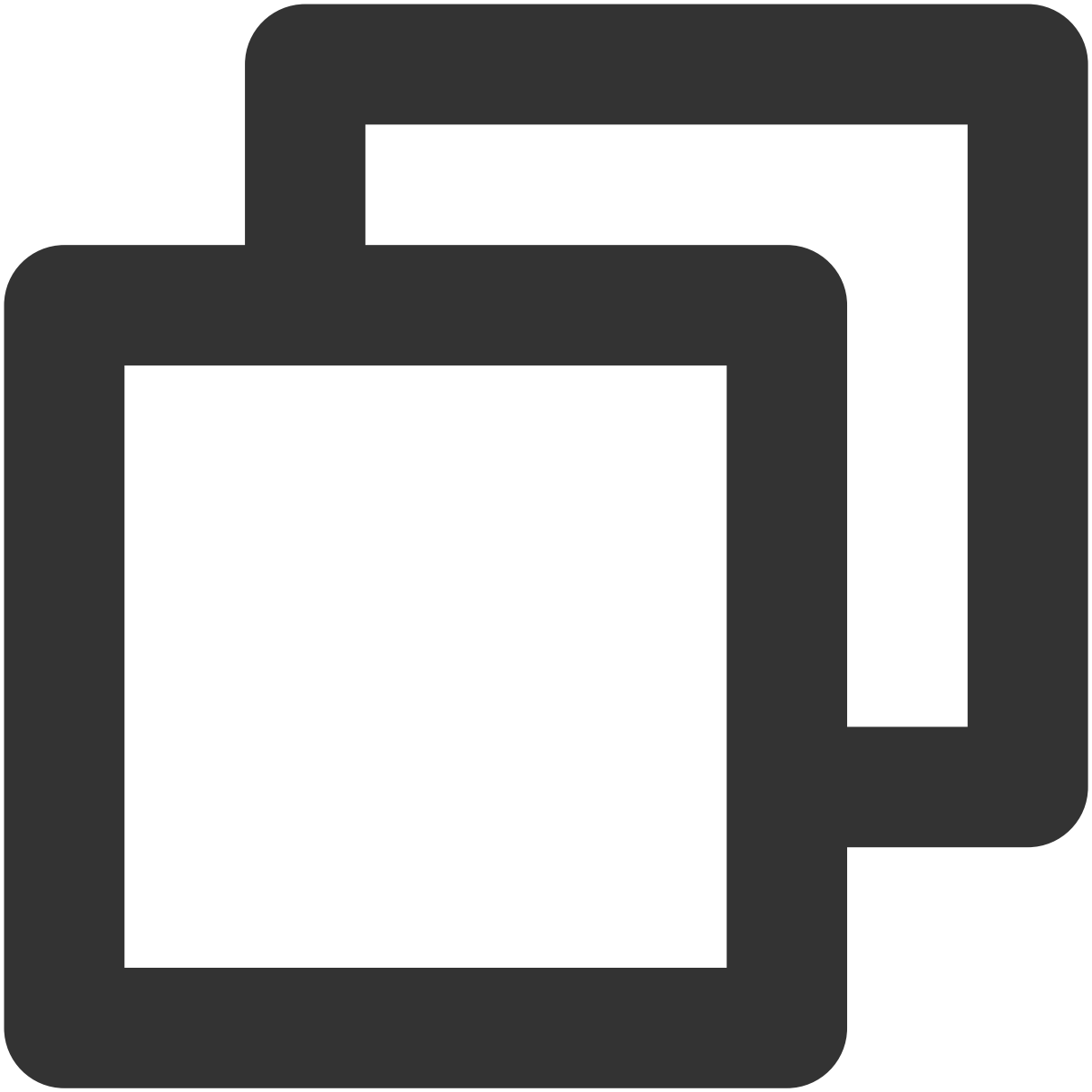
获取某人采集状态

调用此接口，获取房间内某成员的麦克风状态。

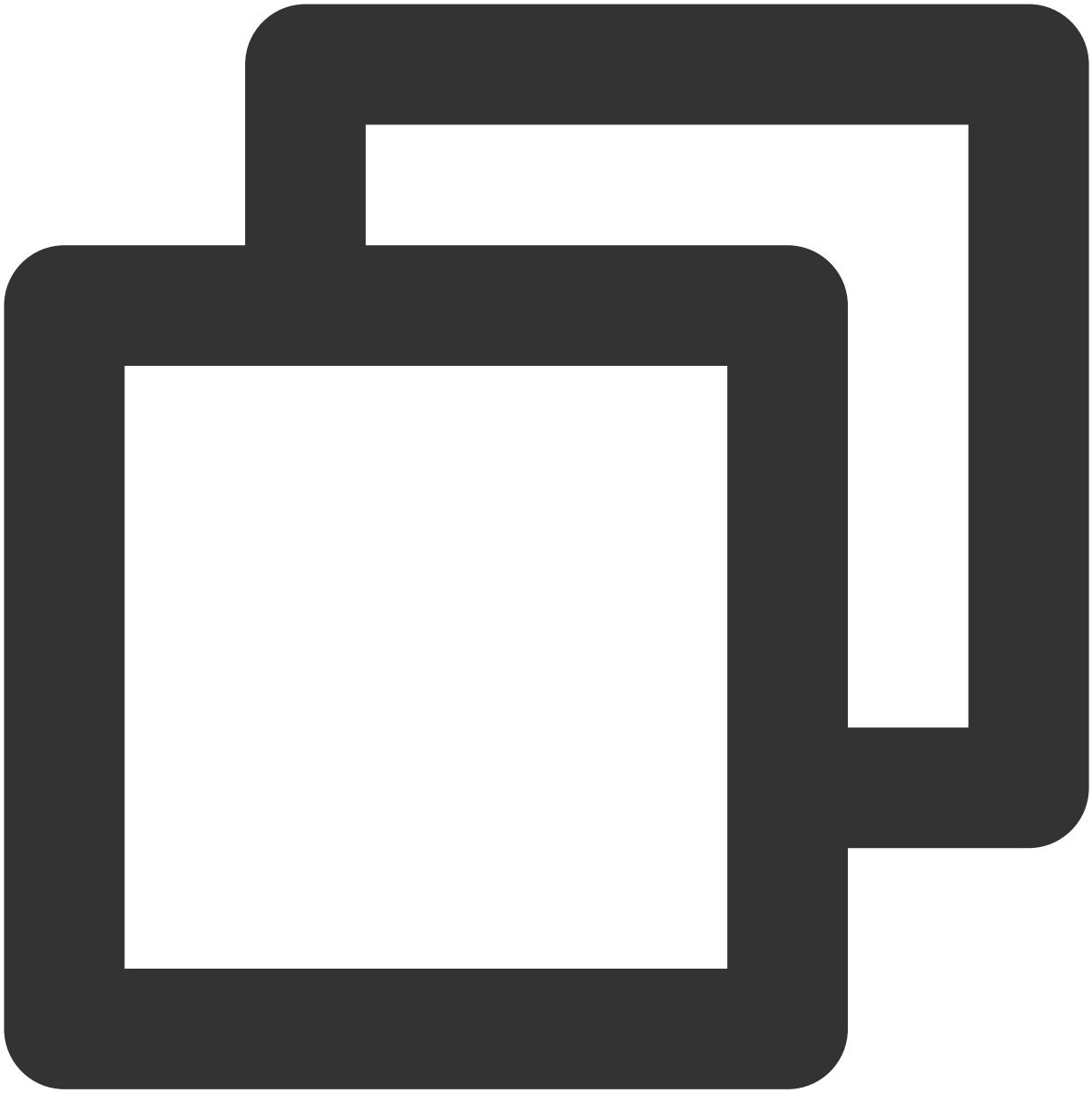
函数原型

Android

iOS



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



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

参数	类型	含义
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_GET_MIC_STATE。

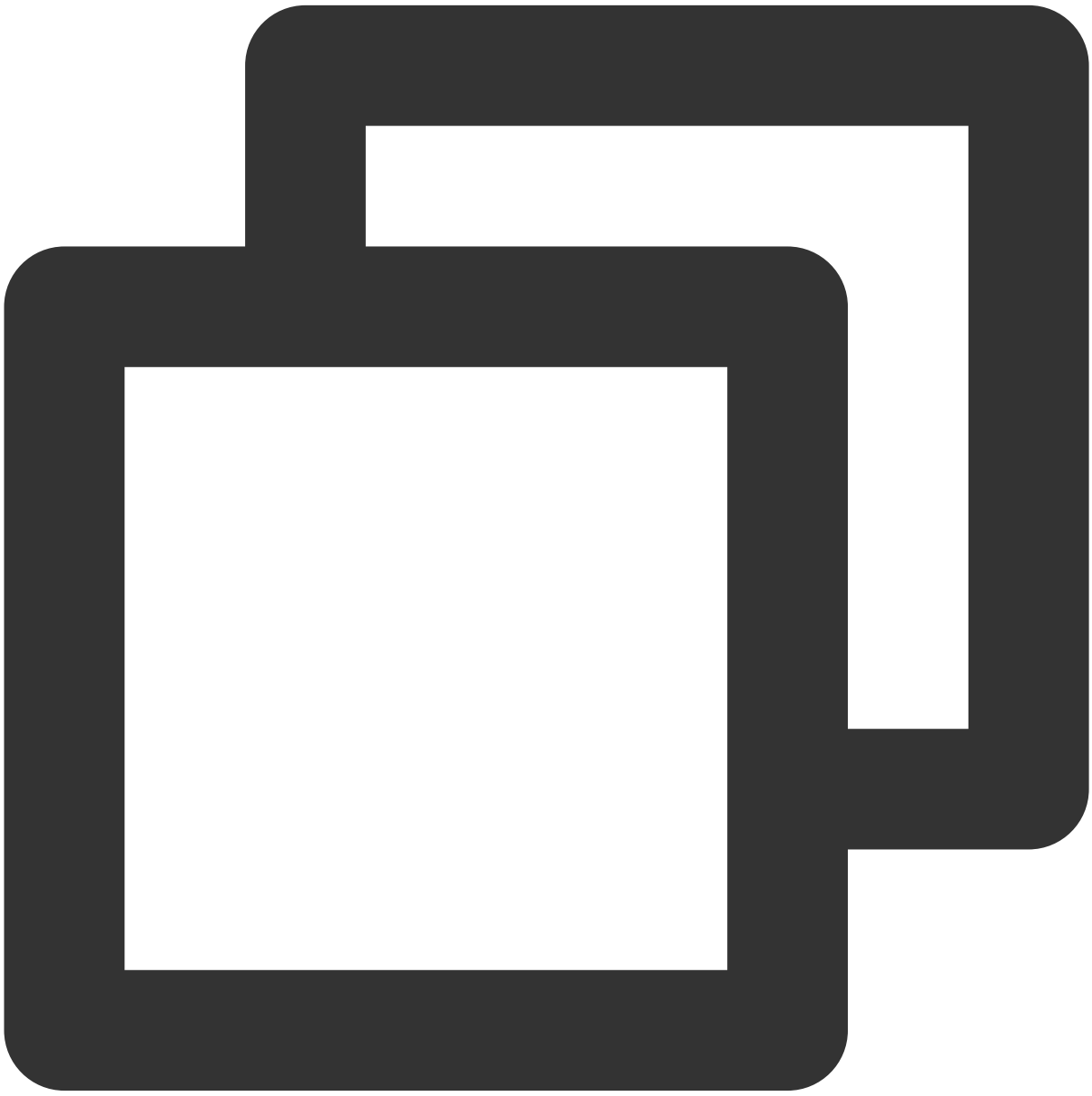
获取某人播放状态

调用此接口，获取房间内某成员的扬声器状态。

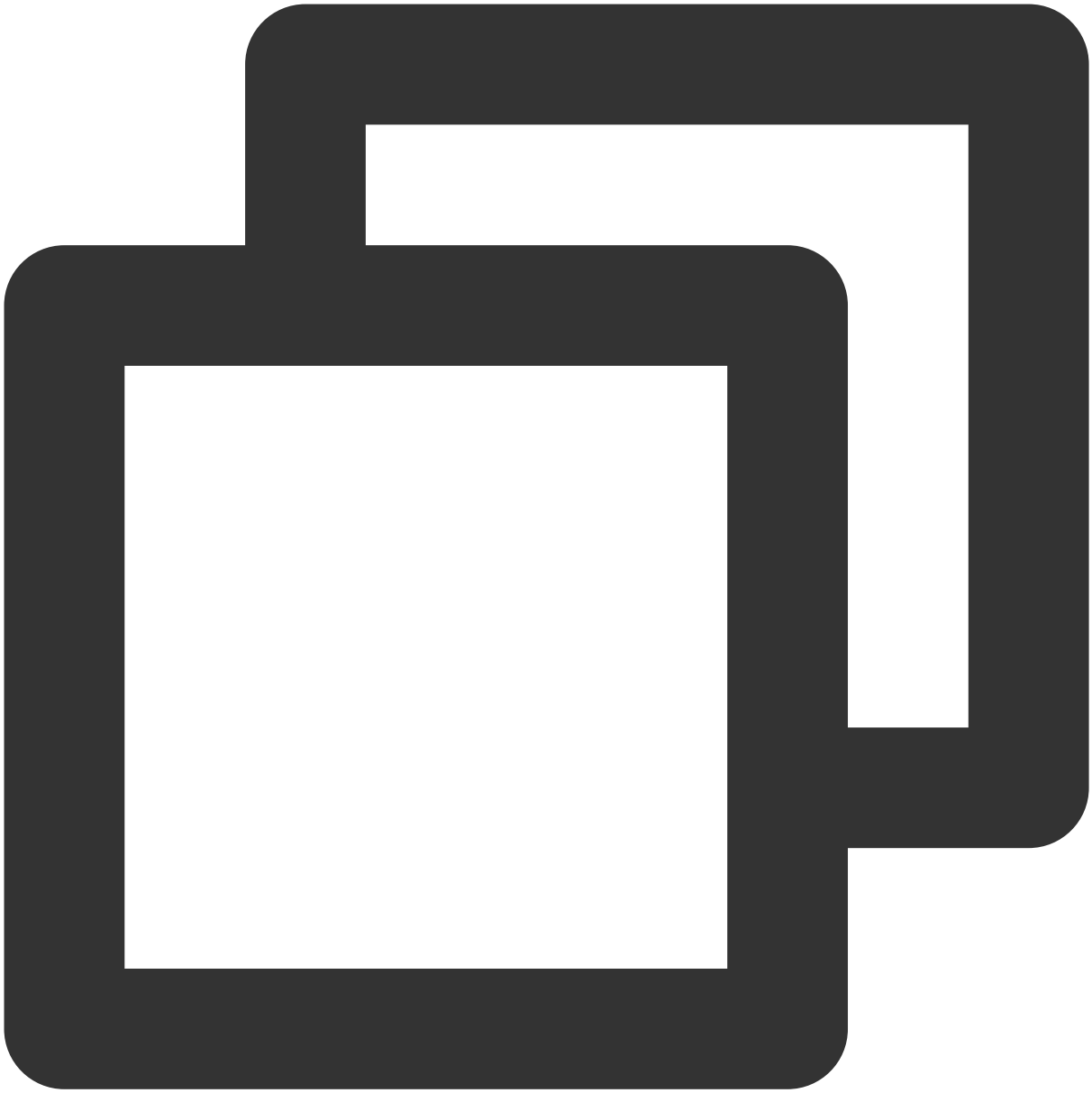
函数原型

Android

iOS



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



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

回调

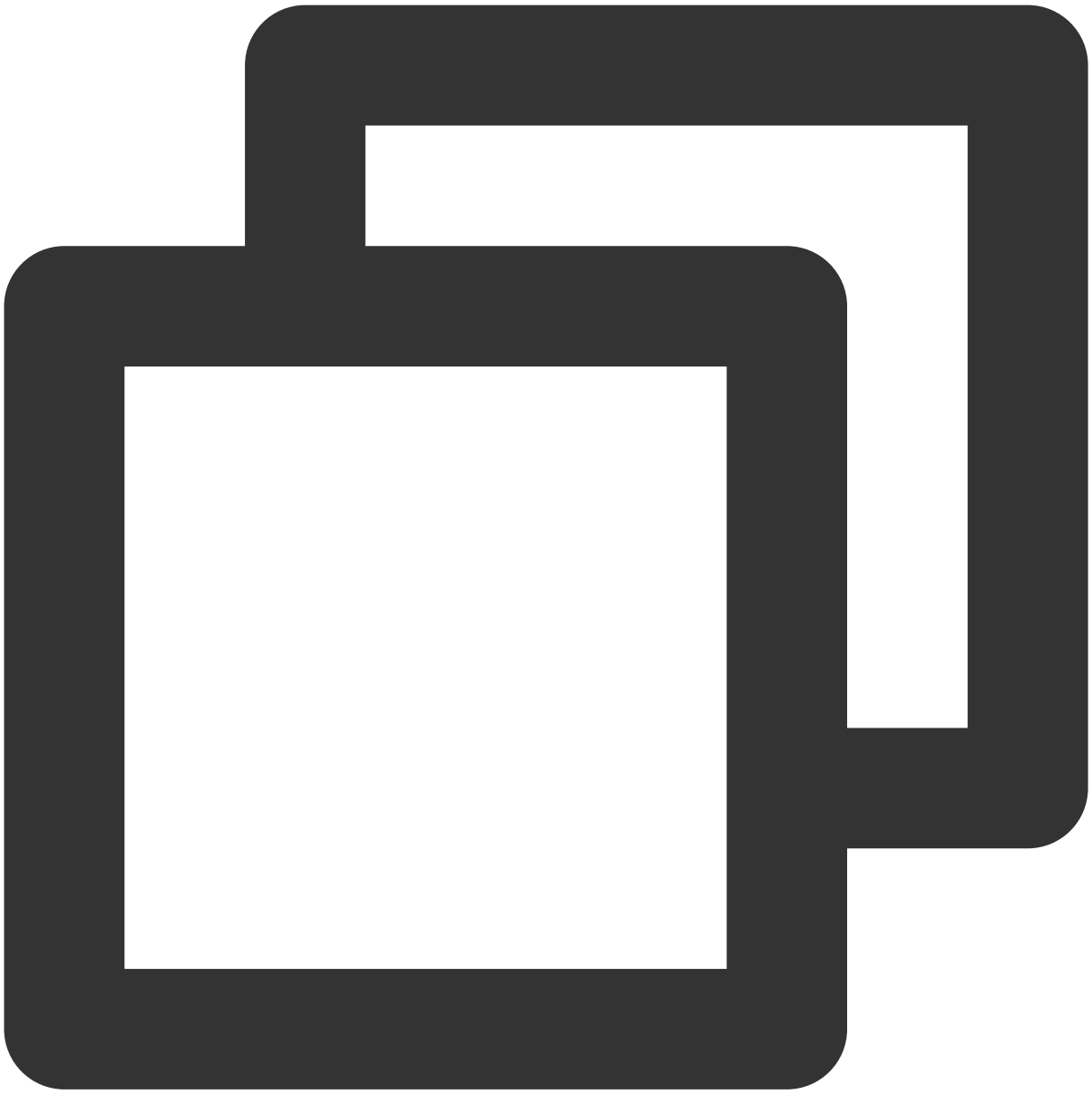
回调参数为 ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE。

禁止某位成员操作采集及播放

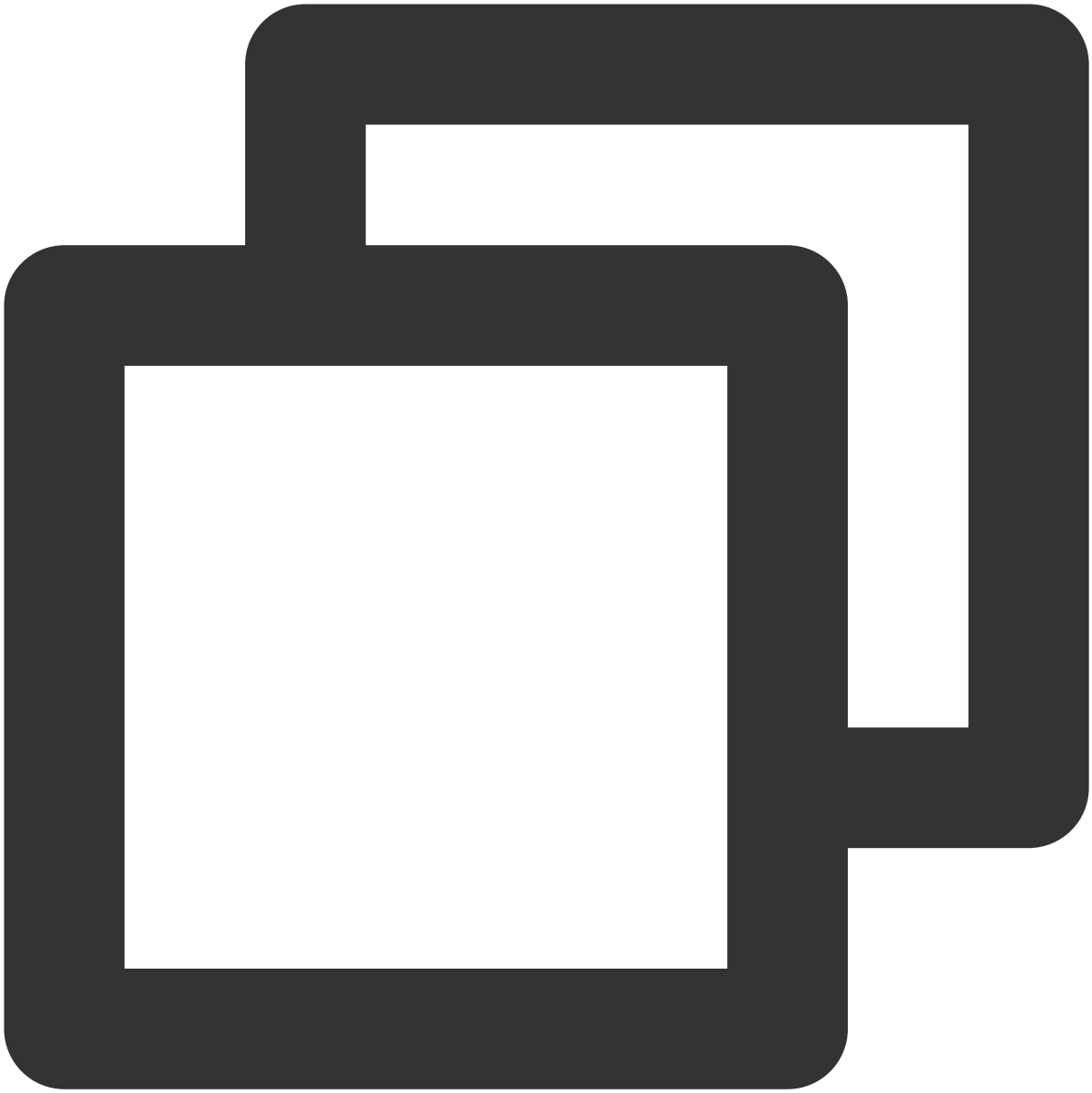
成员进房默认为允许操作麦克风及扬声器，调用此接口可以禁止房间内的某位成员操作麦克风及扬声器，在该成员退出房间后此功能失效。

Android

iOS



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



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

参数	类型	含义
enable	BOOL	YES：禁止某用户操作设备，NO：允许某用户操作设备
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGERMENT_FOBIN_OP。

回调处理

与 GME 其他回调相同，房间管理的回调也在 OnEvent 中处理，事件名称为 `ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR`，事件会返回一个结构体，具体包含的内容如下。

回调参数

参数	类型	含义
SenderID	NSString	事件发送者 ID，如果与自己 OpenId 相同，即为本端发送的命令
ReceiverID	NSString	事件接收者 ID，如果与自己 OpenId 相同，即为本端接收的命令
OperateType	NSNumber	事件类型
Result	NSNumber	事件结果，0为成功
OperateValue	NSNumber	命令详情

OperateType

数值	事件类型	含义
0	ITMG_ROOM_MANAGEMENT_CAPTURE_OP	控制采集设备硬件回调
1	ITMG_ROOM_MANAGEMENT_PLAY_OP	控制播放设备硬件回调
2	ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP	控制上行回调
3	ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP	控制下行回调
4	ITMG_ROOM_MANAGEMENT_MIC_OP	控制麦克风回调
5	ITMG_ROOM_MANAGEMENT_PLAY_OP	控制扬声器回调
6	ITMG_ROOM_MANAGEMENT_GET_MIC_STATE	获取麦克风状态
7	ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE	获取扬声器状态
8	ITMG_ROOM_MANAGERMENT_FOBIN_OP	禁止操作麦克风及扬声器事件

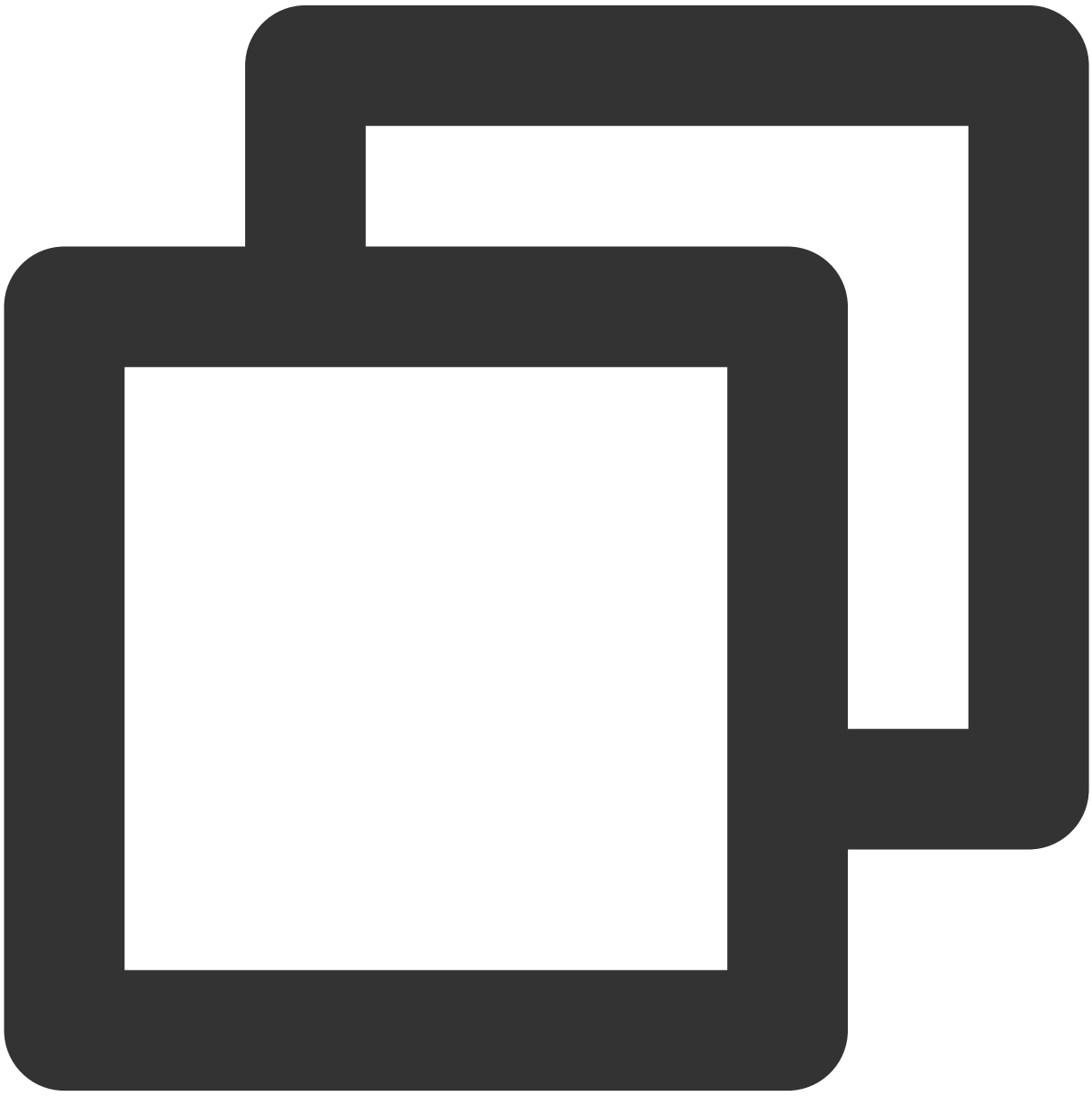
OperateValue

成员	含义
boolValue	0：关闭命令，1：打开命令

示例代码

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("采集");
operatorArr.add("播放");
operatorArr.add("上行");
operatorArr.add("下行");
operatorArr.add("采集上行");
operatorArr.add("播放下行");
operatorArr.add("mic状态");
operatorArr.add("spk状态");
operatorArr.add("禁止操作mic/speak");

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("发送给id:%s 的%s操作
    } else  {
        Toast.makeText(getActivity(), String.format("发送给id:%s 的%s%s操
    }

} 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("是否要打开设备采集");
                    dialog.setMessage("");
                    dialog.setCancelable(false);
                    dialog.setPositiveButton("开", new DialogInterface.
                        //设置确定按钮的点击事件
                        @Override
                        public void onClick(DialogInterface dialog, int
                            mSwitchCapture.setChecked(true);
                            ITMGContext.GetInstance(getActivity()).GetA
                        }
```

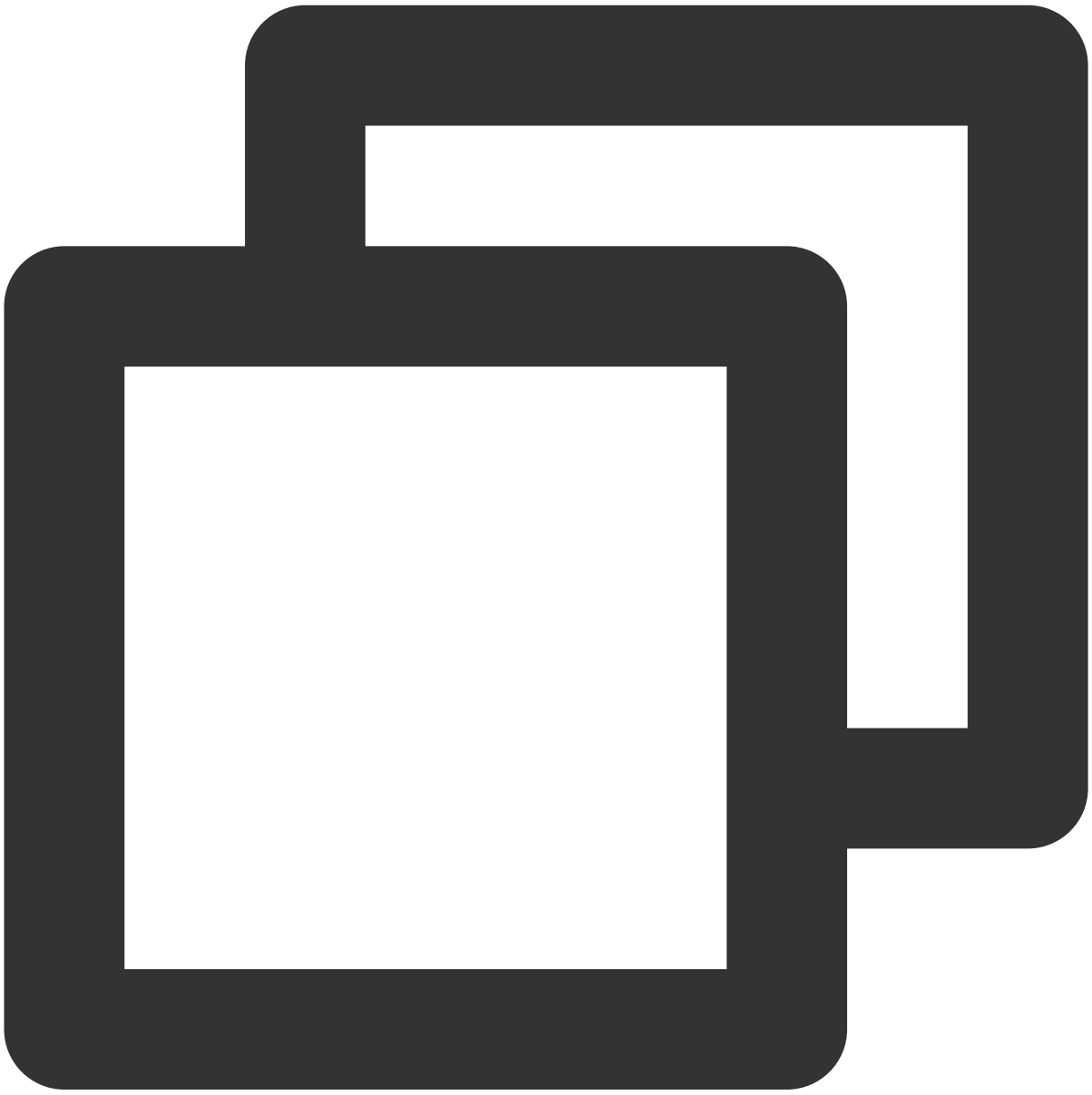
```
                });
                dialog.setNegativeButton("关", new DialogInterface.
                    //设置取消按钮的点击事件
                    @Override
                    public void onClick(DialogInterface dialog, int
                    }
                });
                dialog.show();
            }

        }
            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("是否要打开上行");
                dialog.setMessage("");
                dialog.setCancelable(false);
                dialog.setPositiveButton("开", new DialogInterface.
                    //设置确定按钮的点击事件
                    @Override
                    public void onClick(DialogInterface dialog, int
                        mSwitchSend.setChecked(true);
                        ITMGContext.GetInstance(getActivity()).GetA
                    }
                });
                dialog.setNegativeButton("关", new DialogInterface.
                    //设置取消按钮的点击事件
                    @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("是否要打开采集和上行");
                         dialog.setMessage("");
                         dialog.setCancelable(false);
                         dialog.setPositiveButton("开", new DialogInterface
                             //设置确定按钮的点击事件
                             @Override
                             public void onClick(DialogInterface dialog, in
                                 mSwitchCapture.setChecked(true);
                                 mSwitchSend.setChecked(true);
                                 ITMGContext.GetInstance(getActivity()).Get
                             }
                         });
                         dialog.setNegativeButton("关", new DialogInterface
                             //设置取消按钮的点击事件
                             @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("收到来自id:%s 的%s操
        }
        else if (OperateType == ITMGContext.ITMG_ROOM_MANAGEMENT_SPEAKER_OP
            Toast.makeText(getActivity(), String.format("收到来自id:%s 的%s%s
        } else if (OperateValue == false) {
```

```
                    Toast.makeText(getActivity(), String.format("收到来自id:%s 的%s%s
                }
            }
    }
```



```
-(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 = @[@"采集",@"播放",@"上行",@"下行",@"采集上行",@"播放
    // _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"];

    //自己发出去的命令
    if ([SenderID isEqualToString:_openId]) {
        if (OperateType.intValue == ITMG_ROOM_MANAGEMENT_GET_MIC_STATE || O
                NSString *alterString = [NSString stringWithFormat:@"发送给
                                    UIAlertView *alert = [[UIAlertView all
                                    [alert show];
        }
        else
        {
            NSString *alterString = [NSString stringWithFormat:@"发送给id:%@
                        UIAlertView *alert = [[UIAlertView alloc] initWithTi
                        [alert show];
        }


    }
    else if([ReceiverID isEqualToString:_openId] ){ //别人发过来的命令
        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:@"收到id:
                                UIAlertView *alert = [[UIAlertView alloc] initWithT
                                [alert show];
                    }
                else{
                        NSString *alterString = [NSString stringWithFormat:@"收到id:
                              UIAlertView *alert = [[UIAlertView alloc] initWithTit
                              [alert show];
                    }
                }
            }
        }
        break;
}
```