Tencent Real-Time Communication

Video Calling (Including UI)

Product Documentation

Contents

Video Calling (Including UI) Overview (TUICallKit)

Last updated：2024-06-17 17:41:52

Component Overview

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

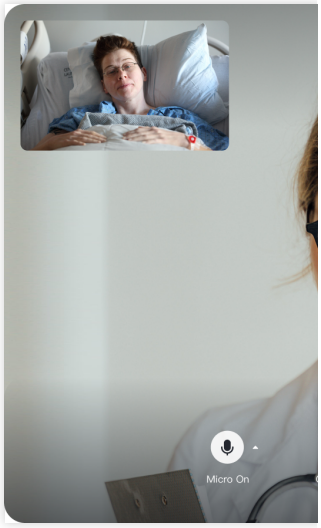
Supported Platform

Platform	Android	iOS	Web	Flutter
Supported				
Languages/Frameworks	Kotlin Java	Swift Objective-C	Vue3 React	Dart

Description of the Feature

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

Applicable Scenario

Online Social Interaction	Online Consultation
	

Online Education	Online Sales
	

Trying It Online

Platform	Web	Android	iOS	Flutter
Online Experience				/
Demo Integration	Web Demo	Andorid Demo	iOS Demo	Flutter Demo

Exchange and Feedback

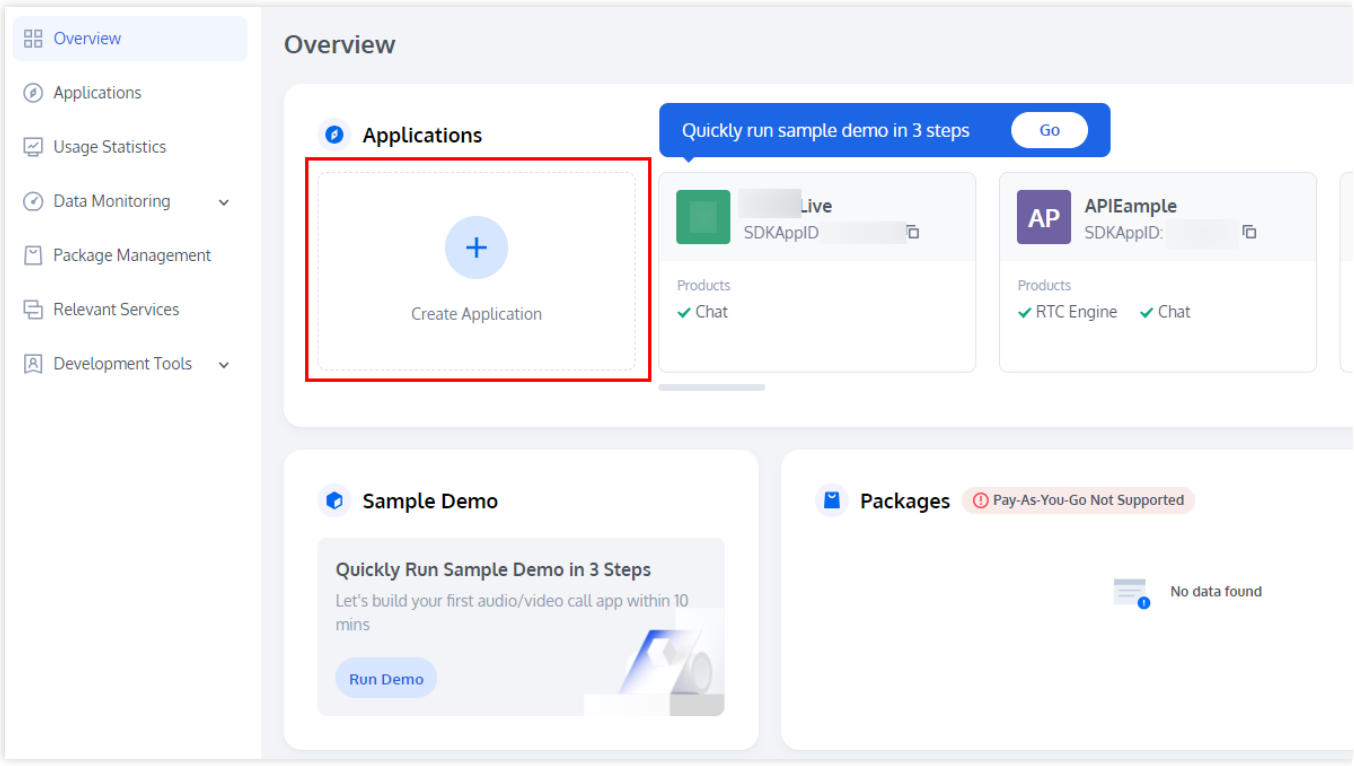
If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

Activate the Service（TUICallKit）

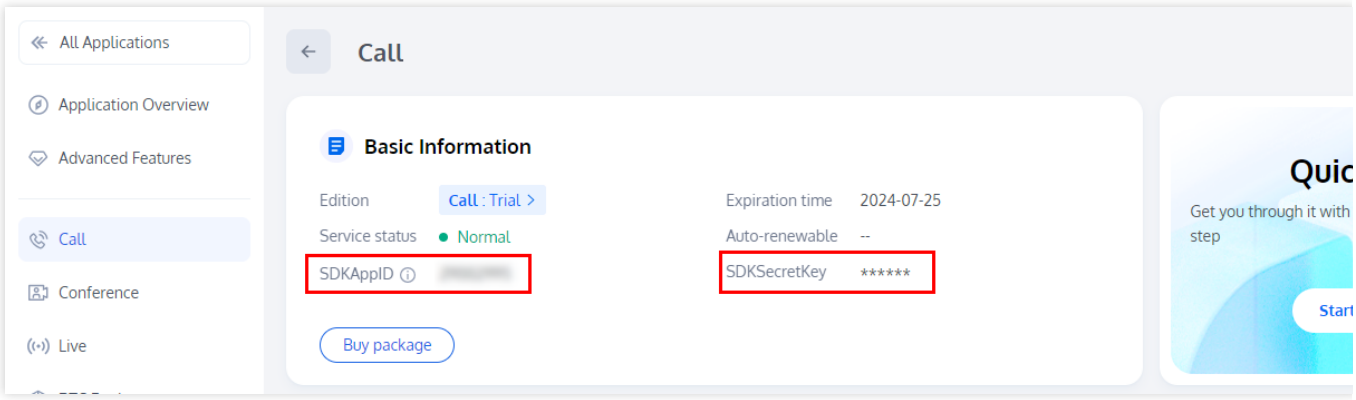
Last updated：2024-07-19 11:13:55

Activate Trial Service

In order for you to better experience the features of TRTC Call, we provide a 7-day free trial. Each SDKAppID can try TRTC Call twice for free, each time for 7 days. Each account can try out TRTC Call 10 times in total.
You can refer to the following guidelines to activate the trial edition of Call.
1. Log in to the Tencent RTC Console, and click on **Create Application.**



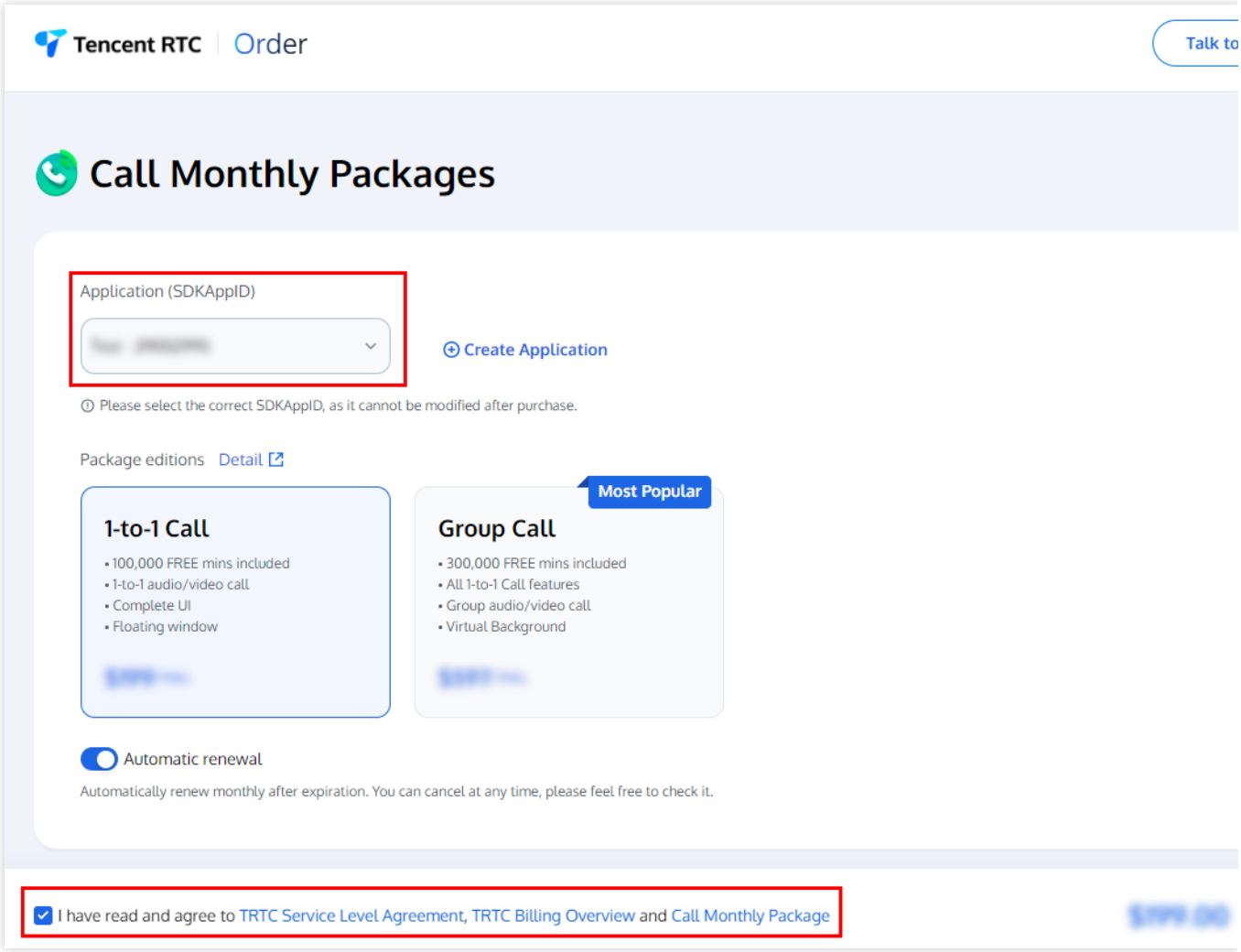
2. After the application has been created, you will automatically be taken to the Call application details page. At this point, you have quickly created an application and successfully received the TRTC Call (TUICallKit) trial version. You can view information on the current **Call Application Details page** or **Application Overview**, and refer to the Integration Guide for integration. The `SDKAppID` and `SDKSecretKey` will be used in the Integration Guide.



Purchasing the official edition

For the price and feature comparison details of the Call monthly subscription package, please refer to Call Monthly Packages Billing Instructions,**If you need to purchase a Call monthly subscription, please follow the steps below.**

1. Visit the Tencent RTC Purchase Page, select the Application (SDKAppID) and Call Package you want to purchase. After Confirming purchase information and agreeing to the relevant agreement, check the agreement content and click **Subscribe now**.



2. After the purchase is complete, you can go to the Tencent RTC Console to view the application information.

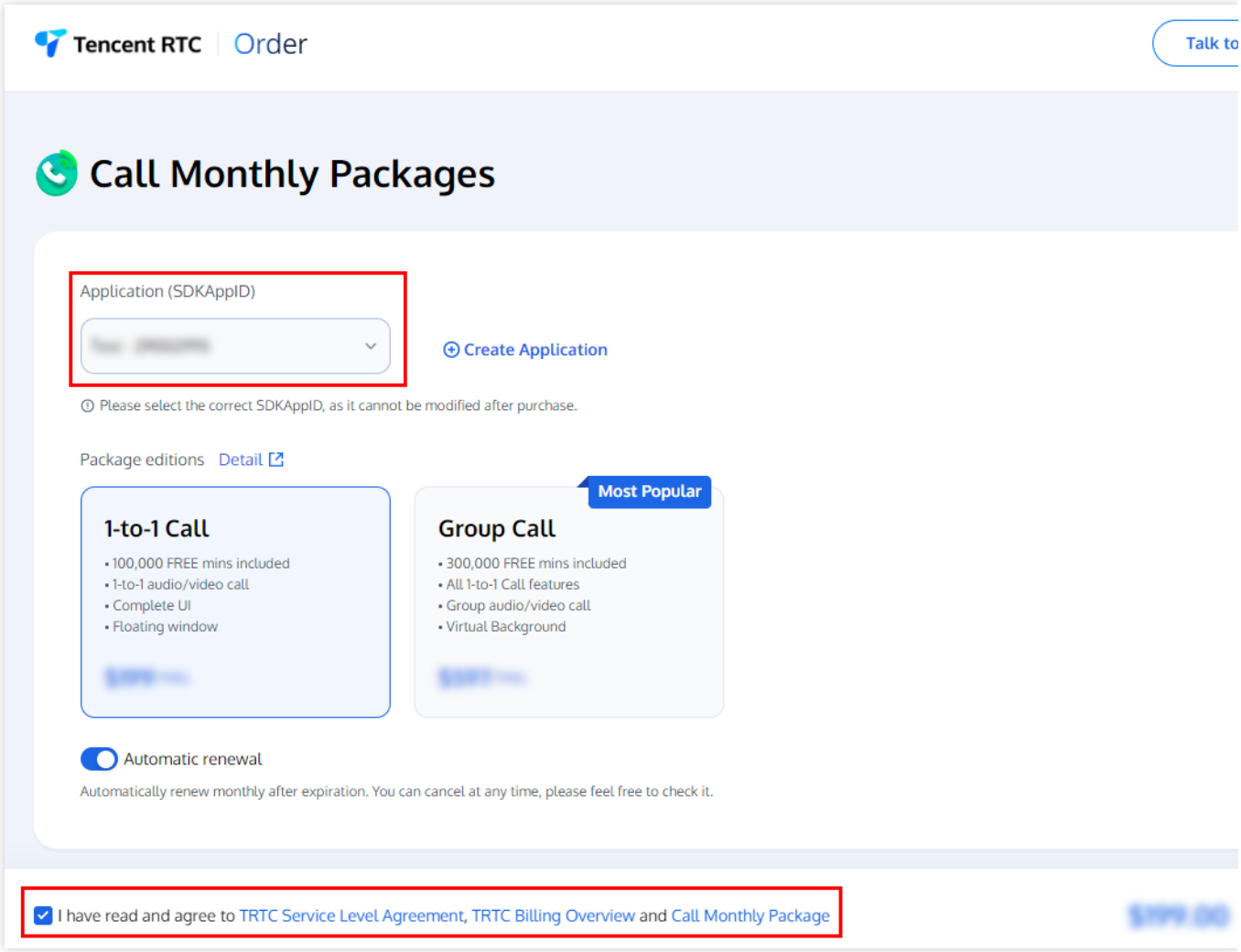
Renew the official edition

Refer to the Purchasing the official edition of Call and purchase the same edition of the Call monthly package again to complete the renewal.

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

Note：

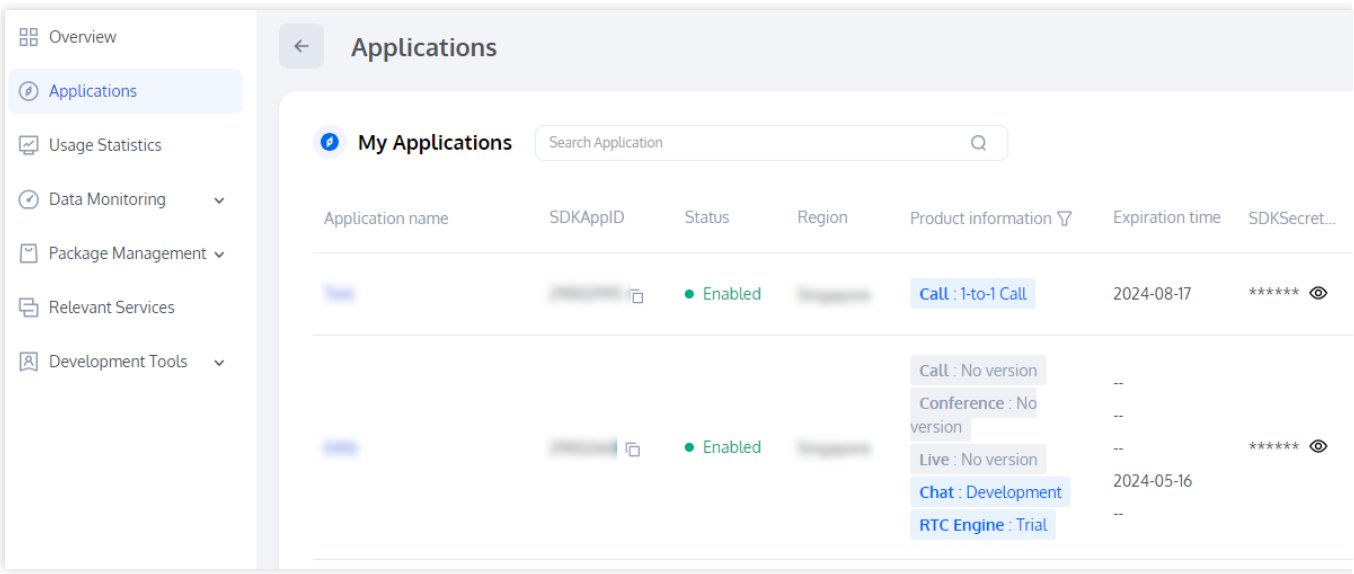
Please select the SDKAppID of the application you wish to renew.



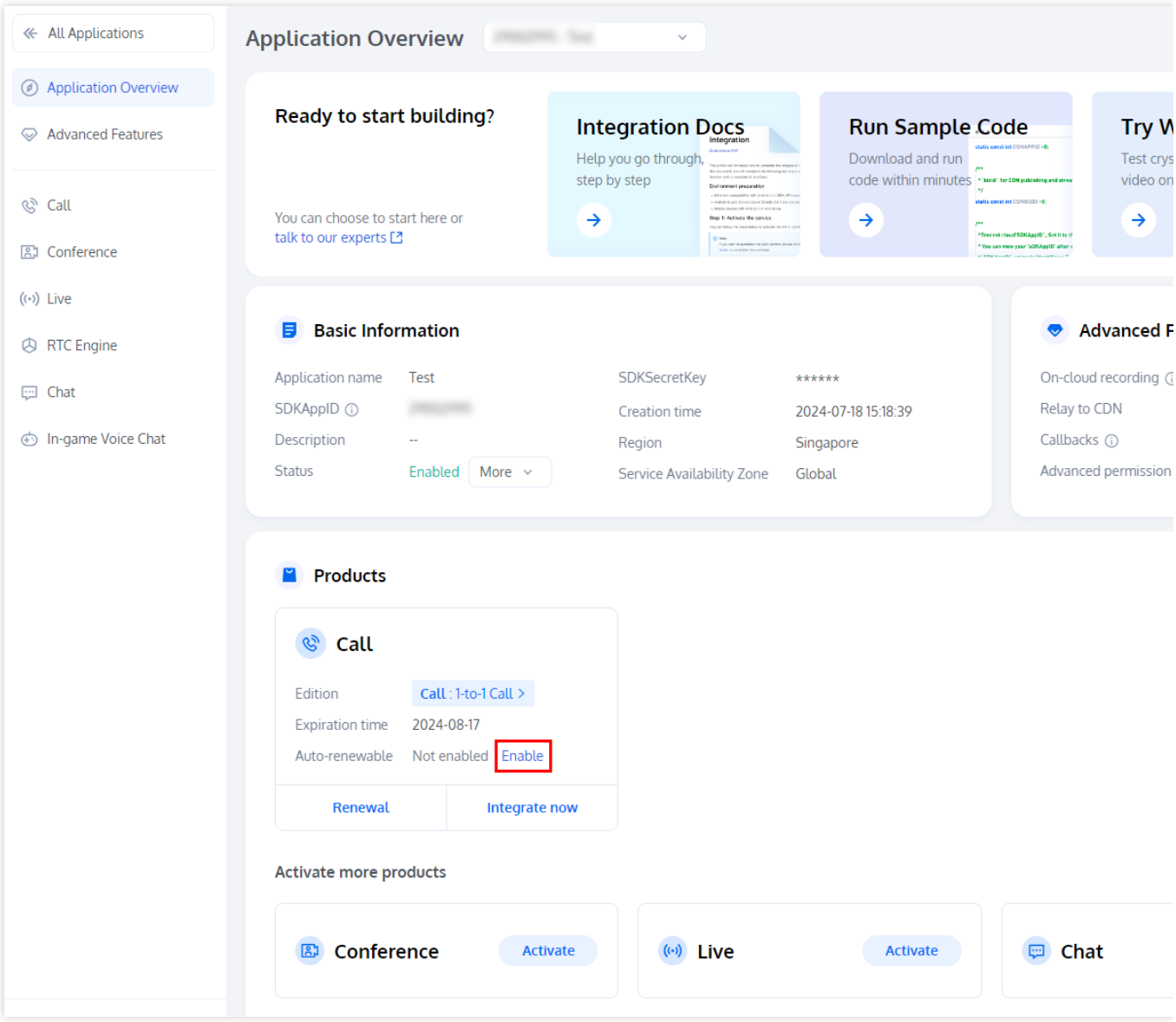
Auto-renewal

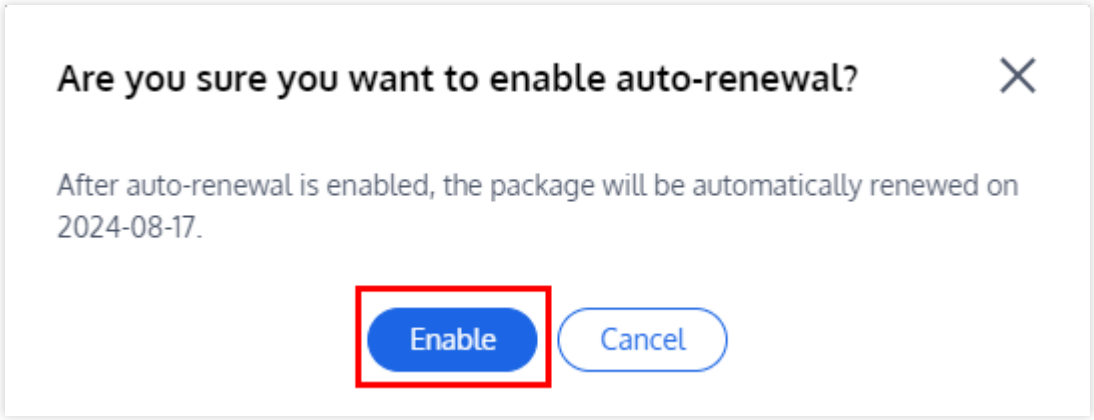
The specific steps to enable auto-renewal in the console are as follows:

1. Access the Tencent RTC Console > Applications, find the application and click **manage** to enter the application details page.



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

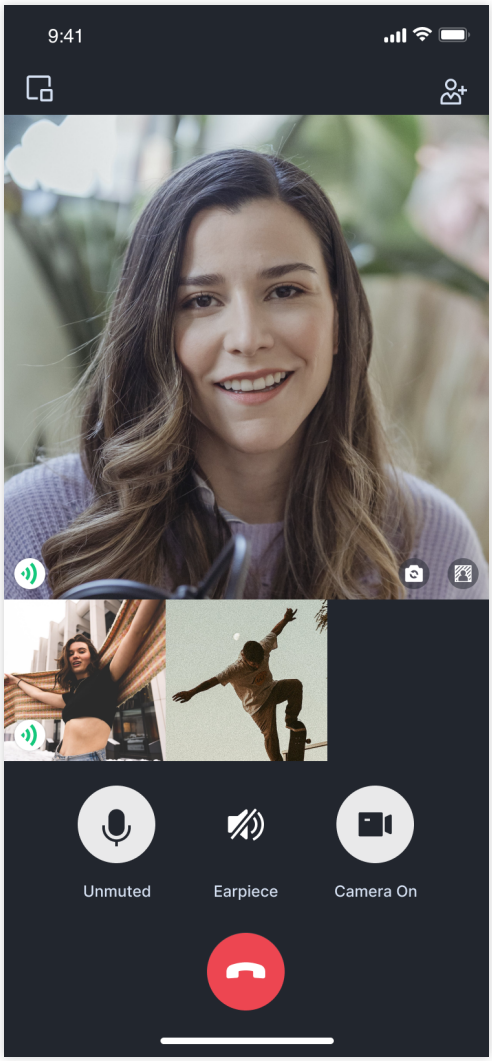




Run Demo（TUICallKit）
Android

Last updated：2024-06-18 17:27:24

This article will guide you on how to quickly run through the Audio and Video Call Demo. By following this document, you can have the Demo up and running in 10 minutes, and ultimately experience an Audio and Video Call feature with a complete UI interface.

1v1 Video Call	Group call
	

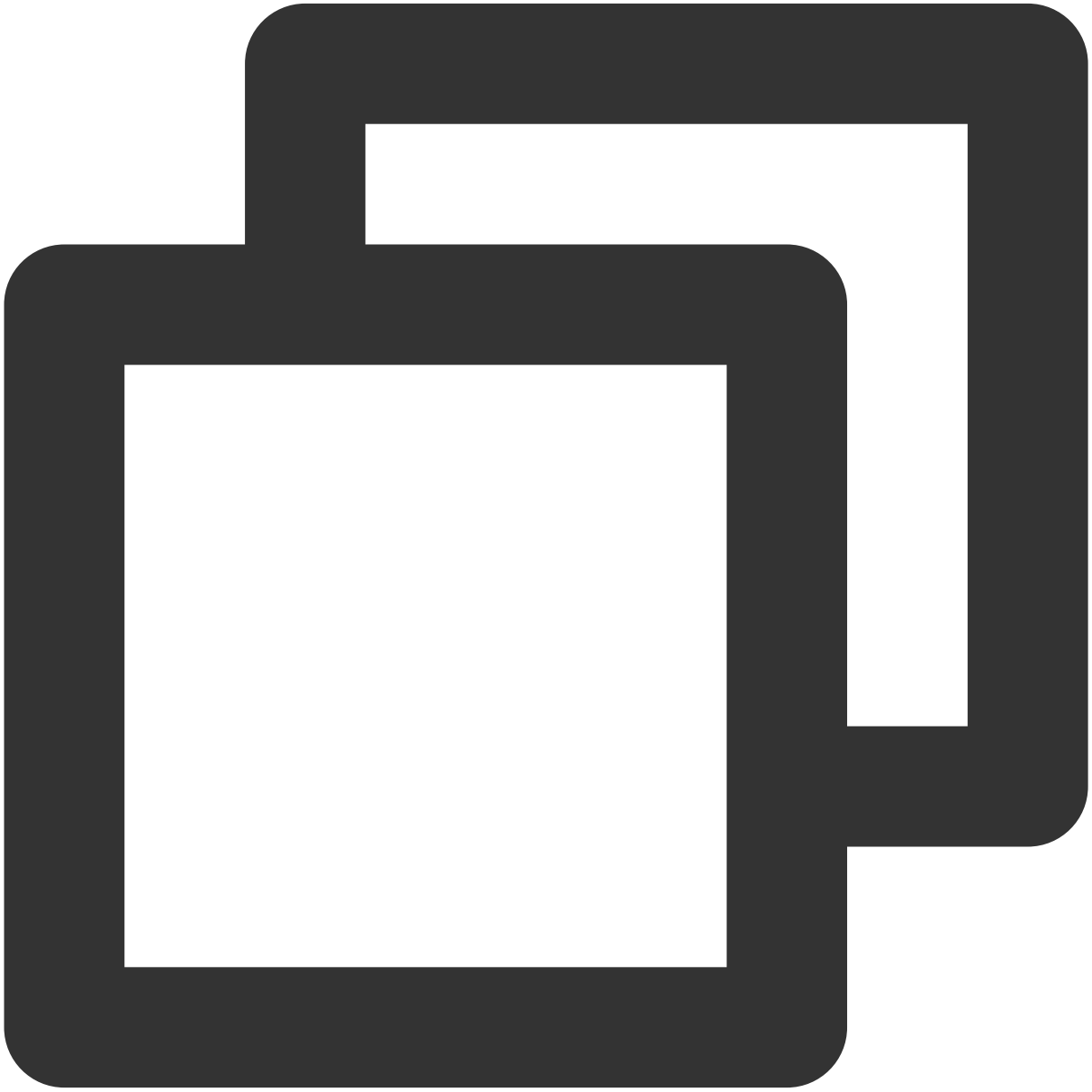
Environment preparations

Android Studio

Two devices with Android 5.0 or higher

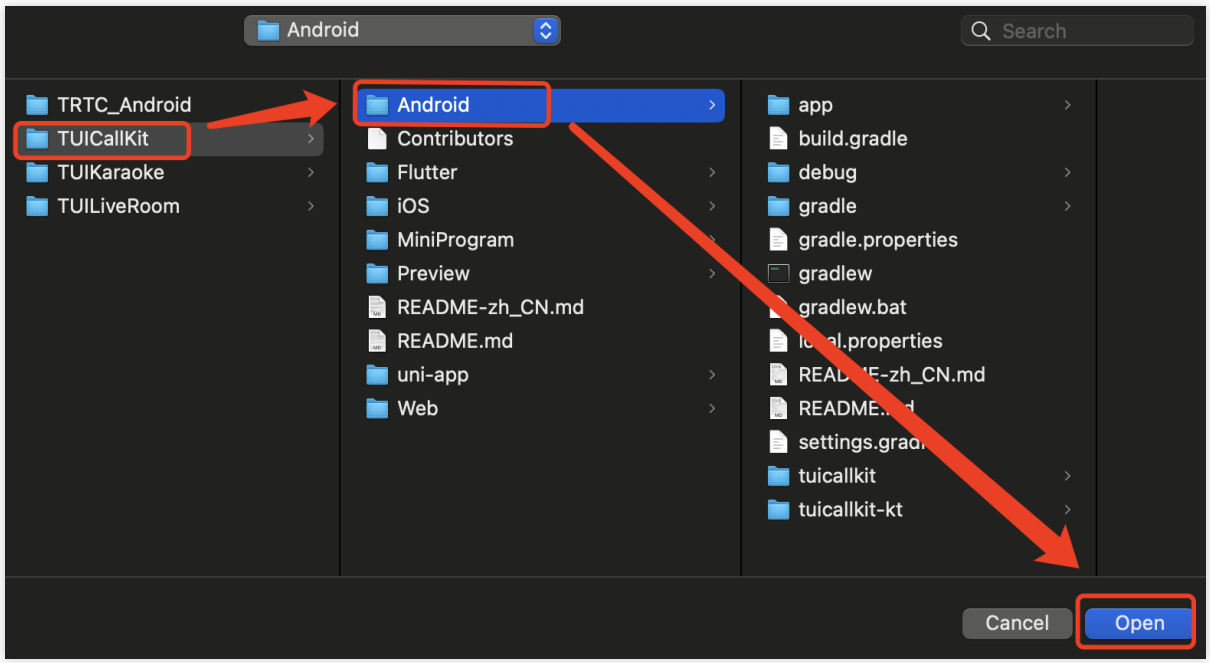
Step 1: Download the Demo

1. Download the TUICallKit Demo source code from GitHub, or directly run the following command in the command line:



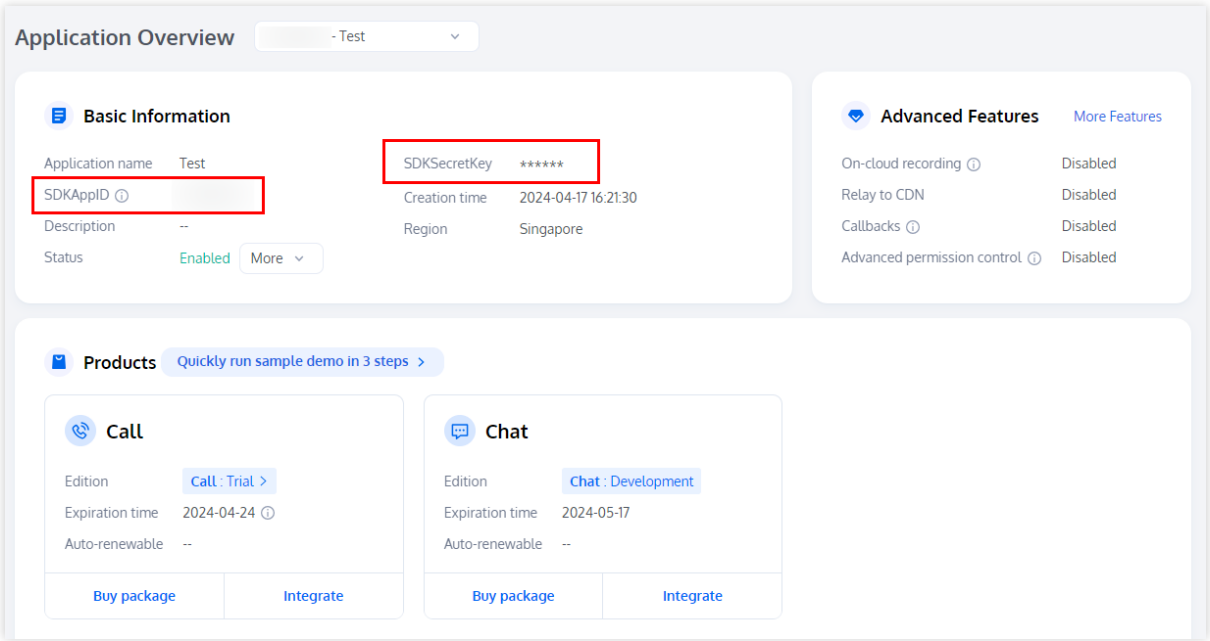
```
git clone https://github.com/Tencent-RTC/TUICallKit.git
```

2. Open the TUICallKit **Android** project through Android Studio:

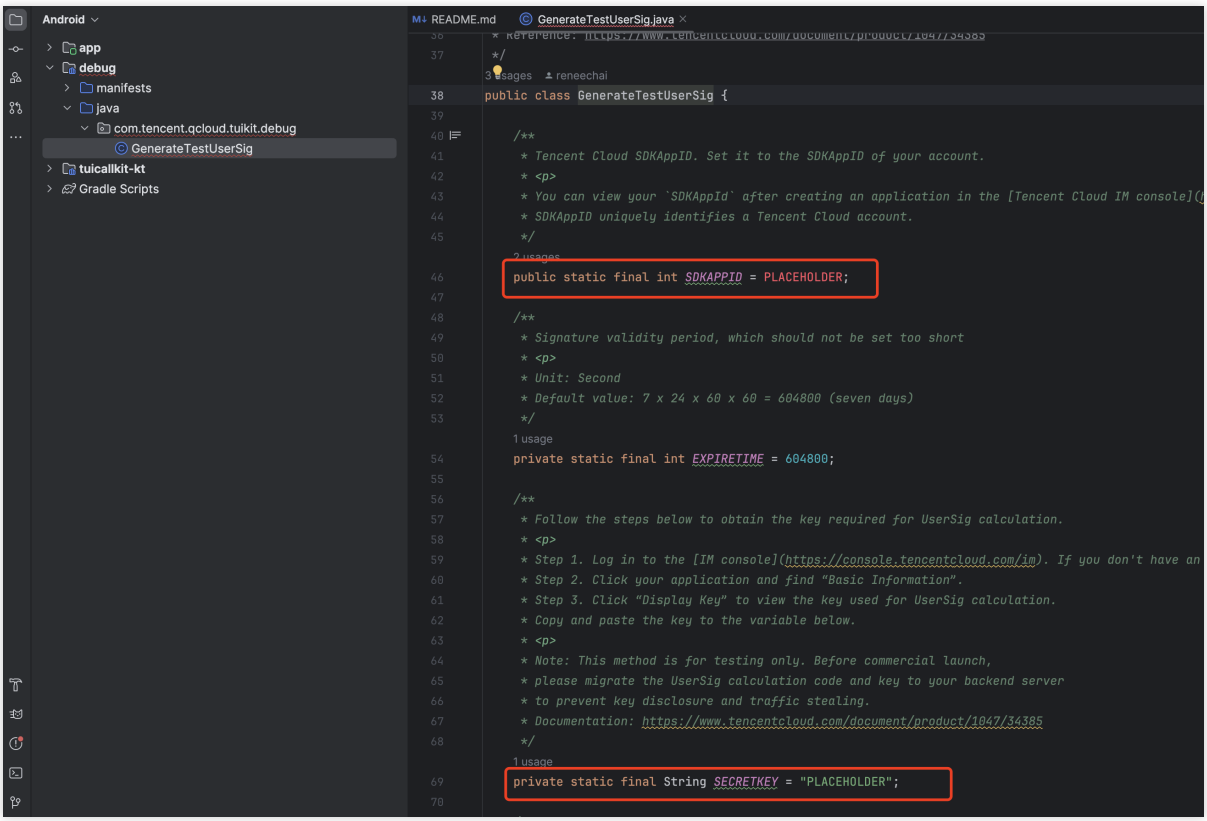


Step 2: Configure the Demo

1. Activate the TRTC service, obtain the SDKAppID and SDKSecretKey.

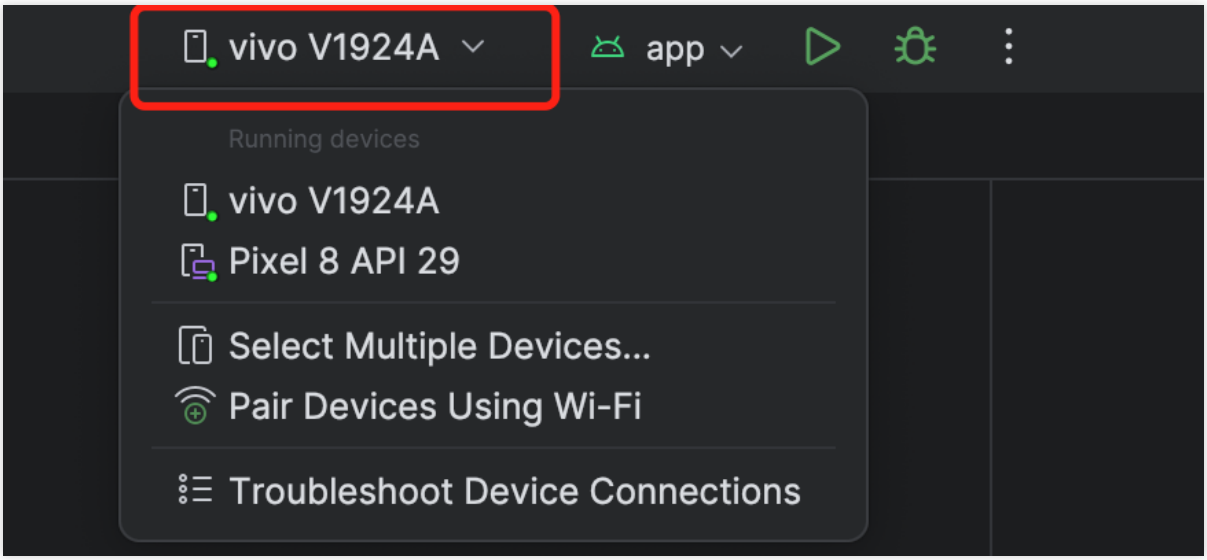


2. Open the `Android/debug/src/main/java/com/tencent/qcloud/tuikit/debug/GenerateTestUserSig.java` file, and enter the SDKAppID and SDKSecretKey obtained when activating the service:

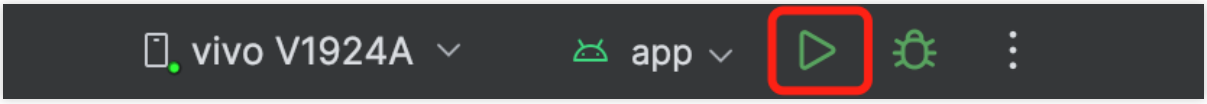


Step 3: Running the Demo

1. In the top right corner of Android Studio, select the device you want to run the Demo on as shown below:



2. After selecting, click Run to execute the TUICallKit Android Demo on the target device.



3. Once the Demo successfully runs on the device, you can initiate a call by following these steps:

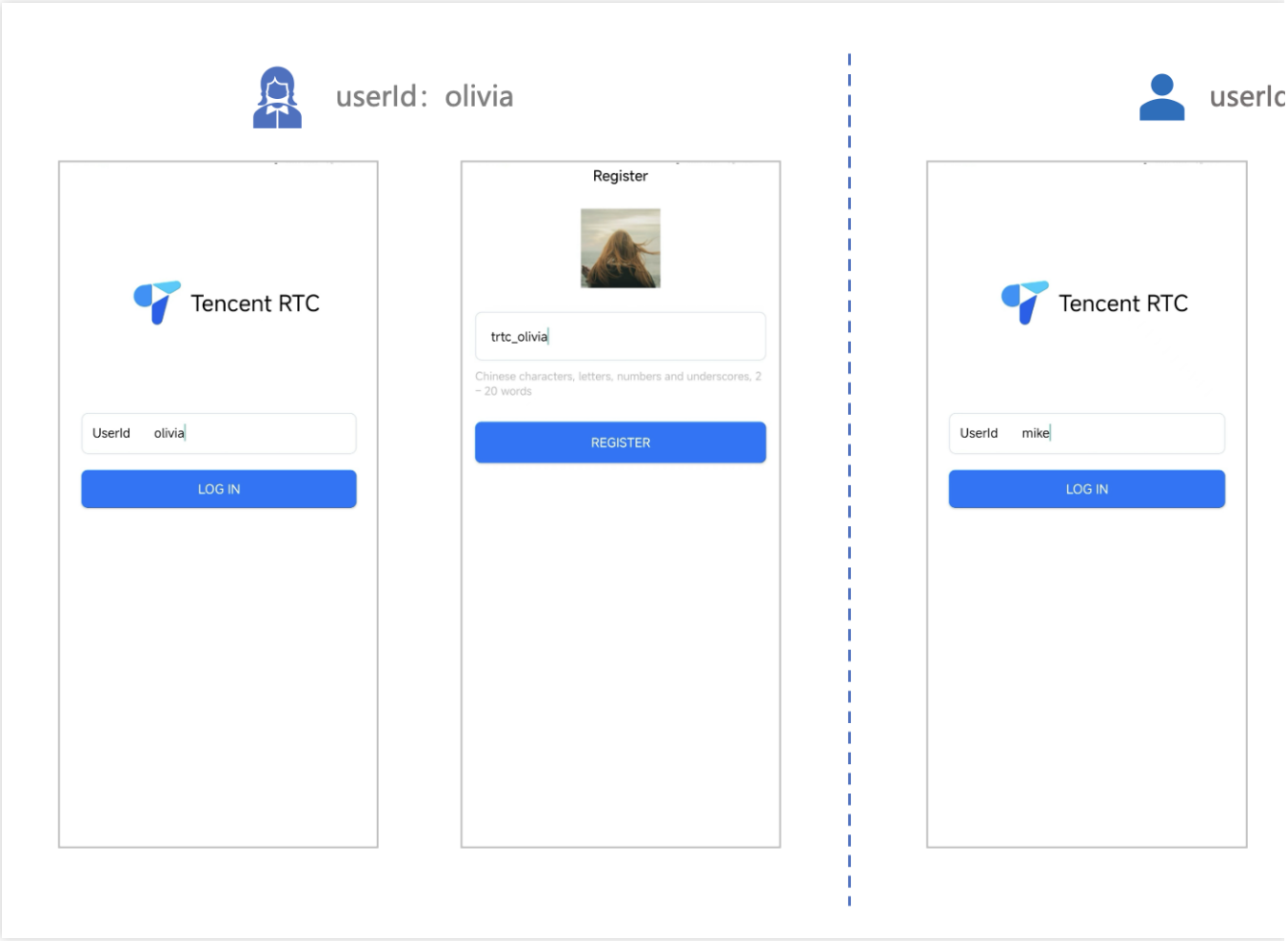
Make the first call

Note:

To experience the complete audio and video calling process, please log into the Demo on two devices as two different users, with one acting as the caller and the other as the callee.

1. Log in & Signup

Please enter the ID at the `User ID` . If your current User ID has not been used before, you will be taken to the **Signup** screen, where you can set your own avatar and nickname.



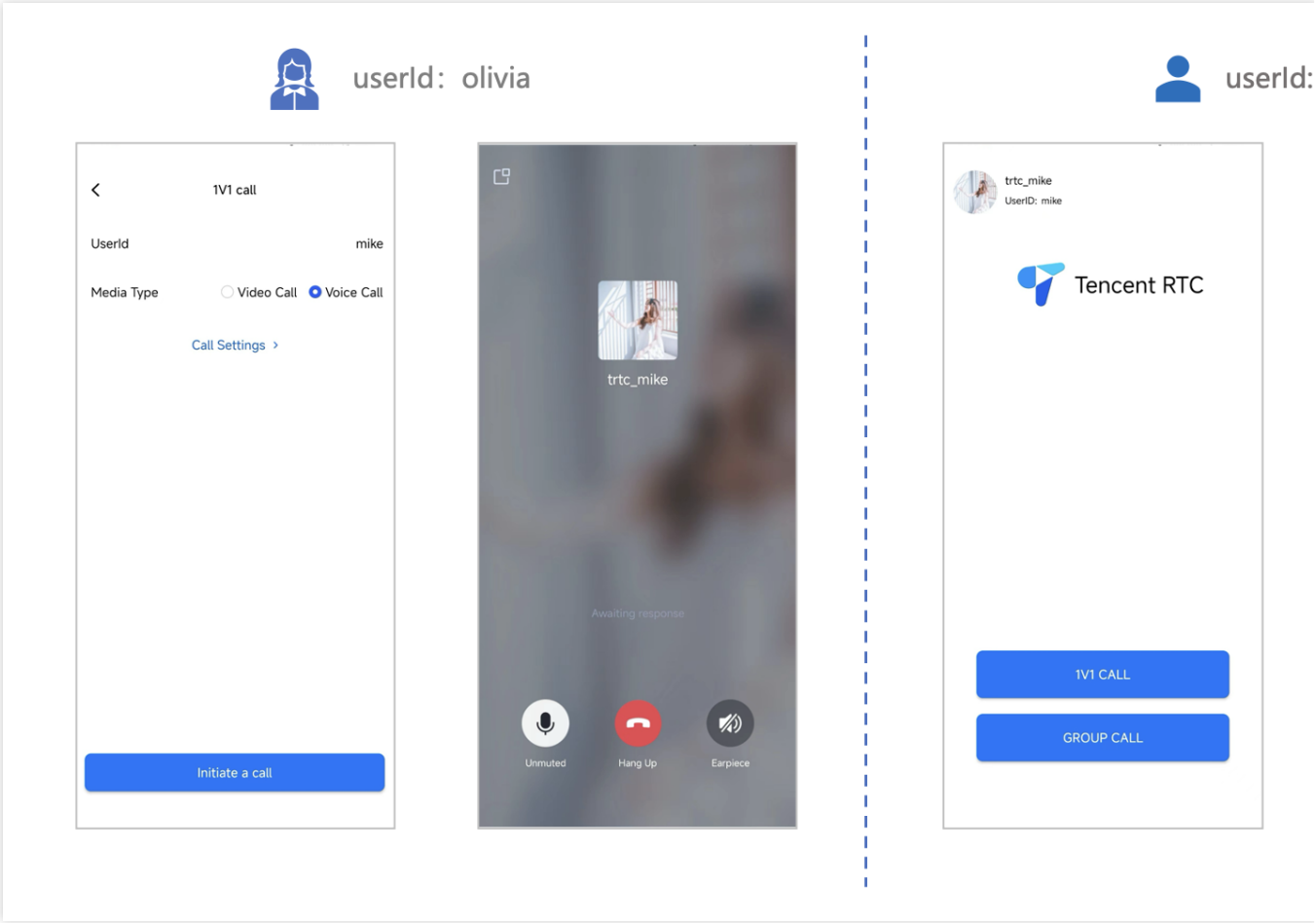
Note:

Try to avoid setting your User ID to simple strings like "1", "123", "111", as TRTC does not support the same User ID being logged into from multiple devices. Such User IDs like "1", "123", "111" are easily occupied by your colleagues during collaborative development, leading to login failures. Therefore, we recommend setting highly recognizable User IDs while debugging.

2. Make a phone call

2.1 The caller should click **1V1 Call** on the interface, enter the callee's User ID in the pop-up interface, and select the desired call type.

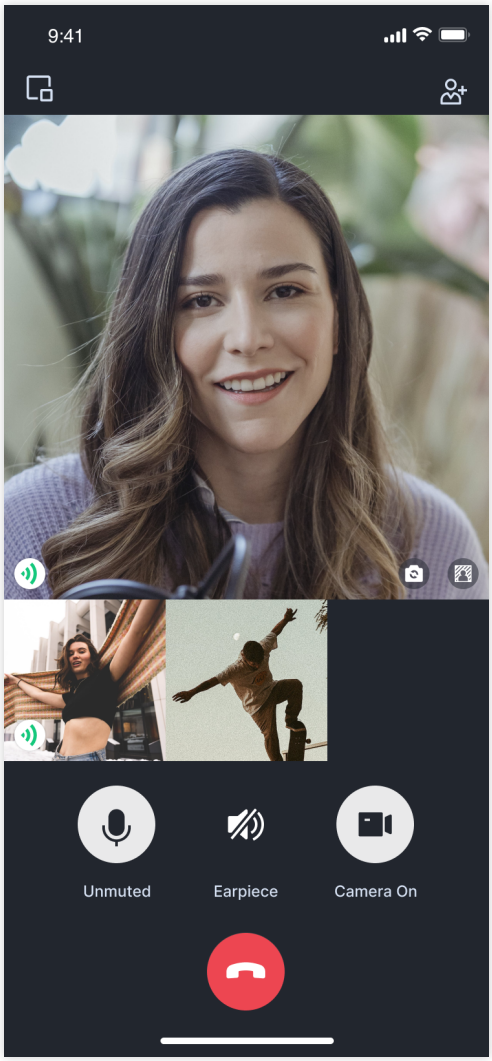
2.2 Click **Initiate Call**.



iOS

Last updated：2024-07-19 18:10:53

This article will guide you on how to quickly run through the Audio and Video Call Demo. By following this document, you can have the Demo up and running in 10 minutes, and ultimately experience an Audio and Video Call feature with a complete UI interface.

1v1 Video Call	Group call
	

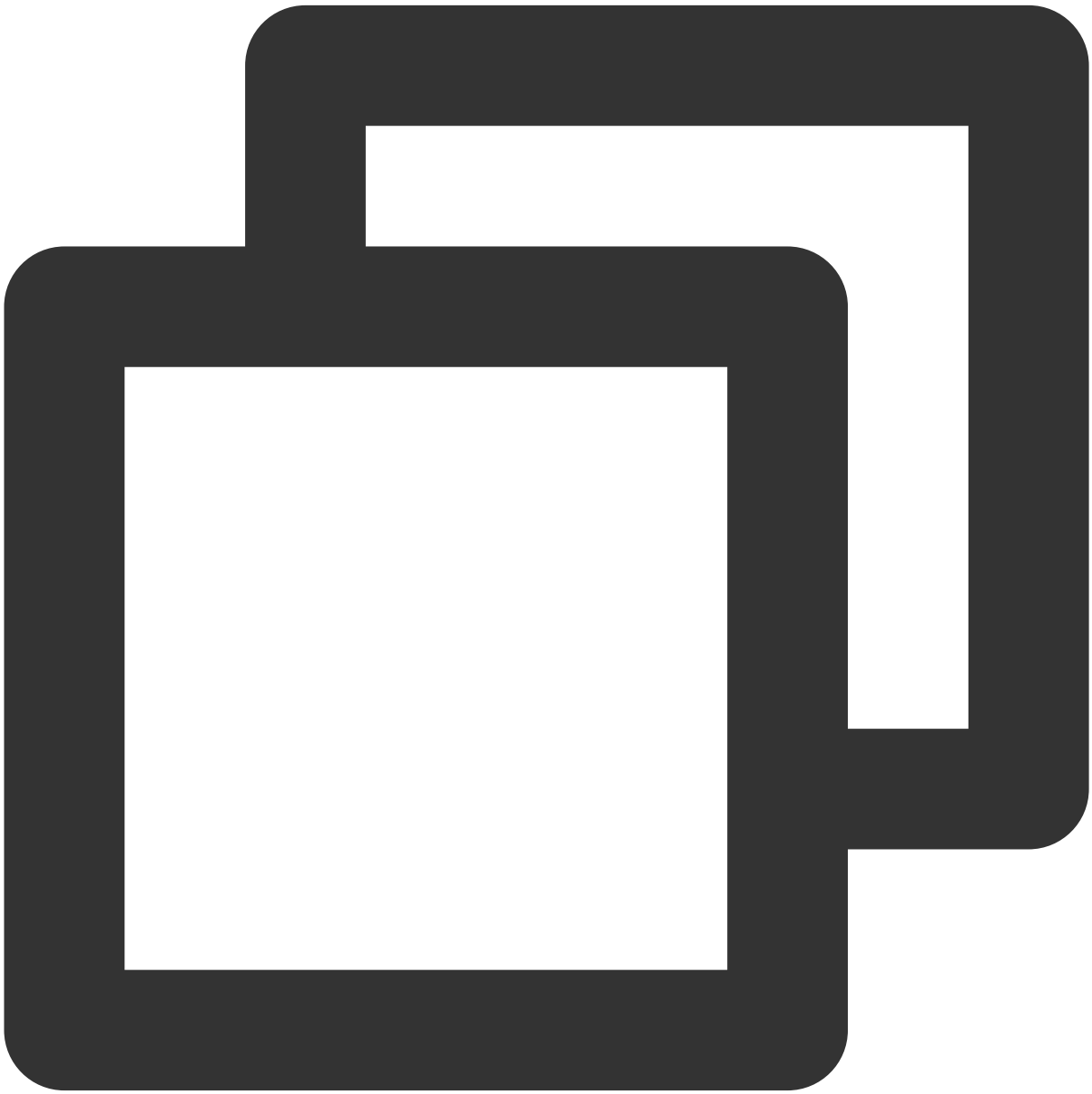
Environment preparations

Xcode 13 or later.

Two iOS 13.0 or later devices.

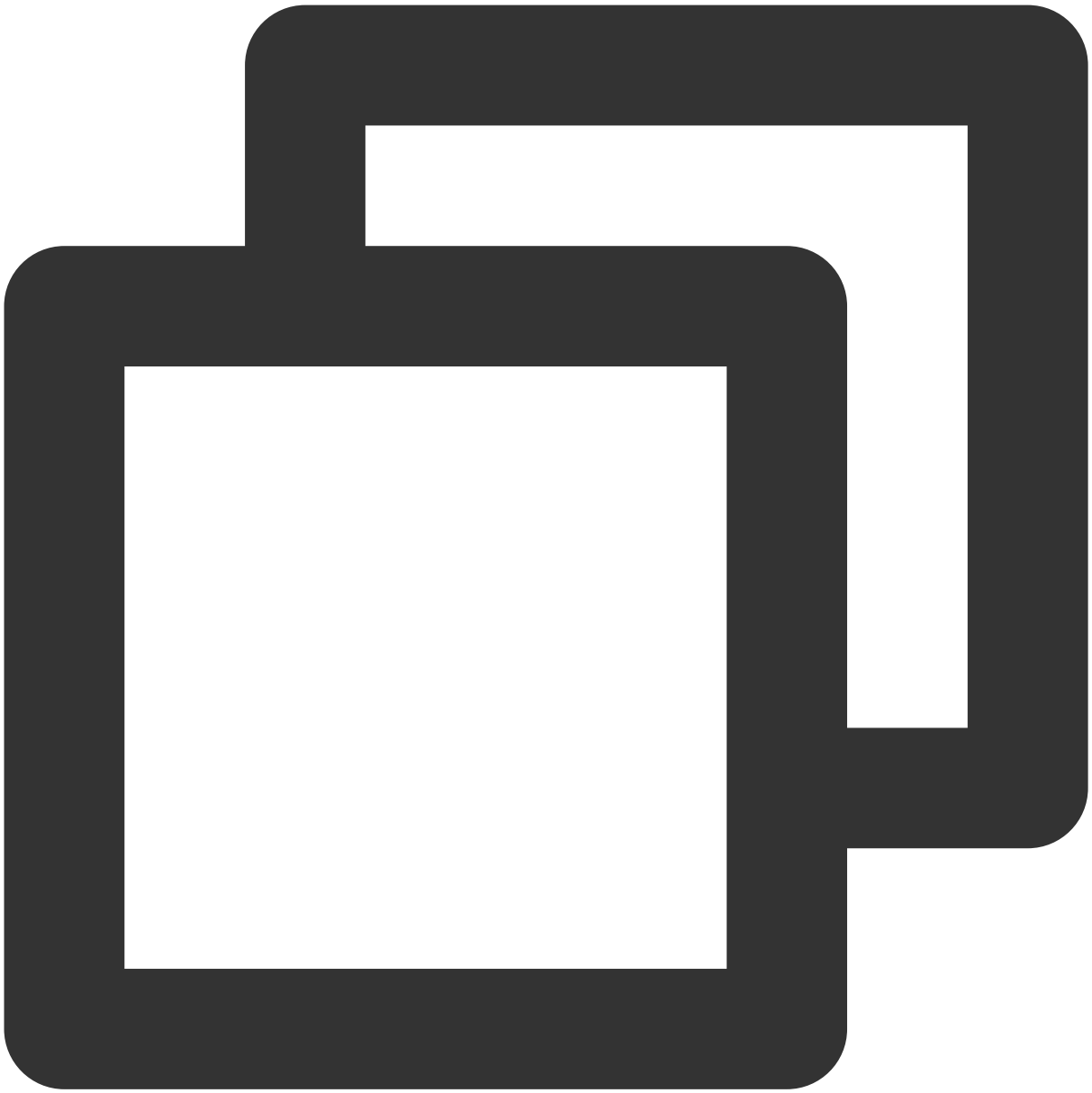
Step 1: Download the Demo

1. Download the TUICallKit Demo source code from GitHub, or directly run the following command in the command line:



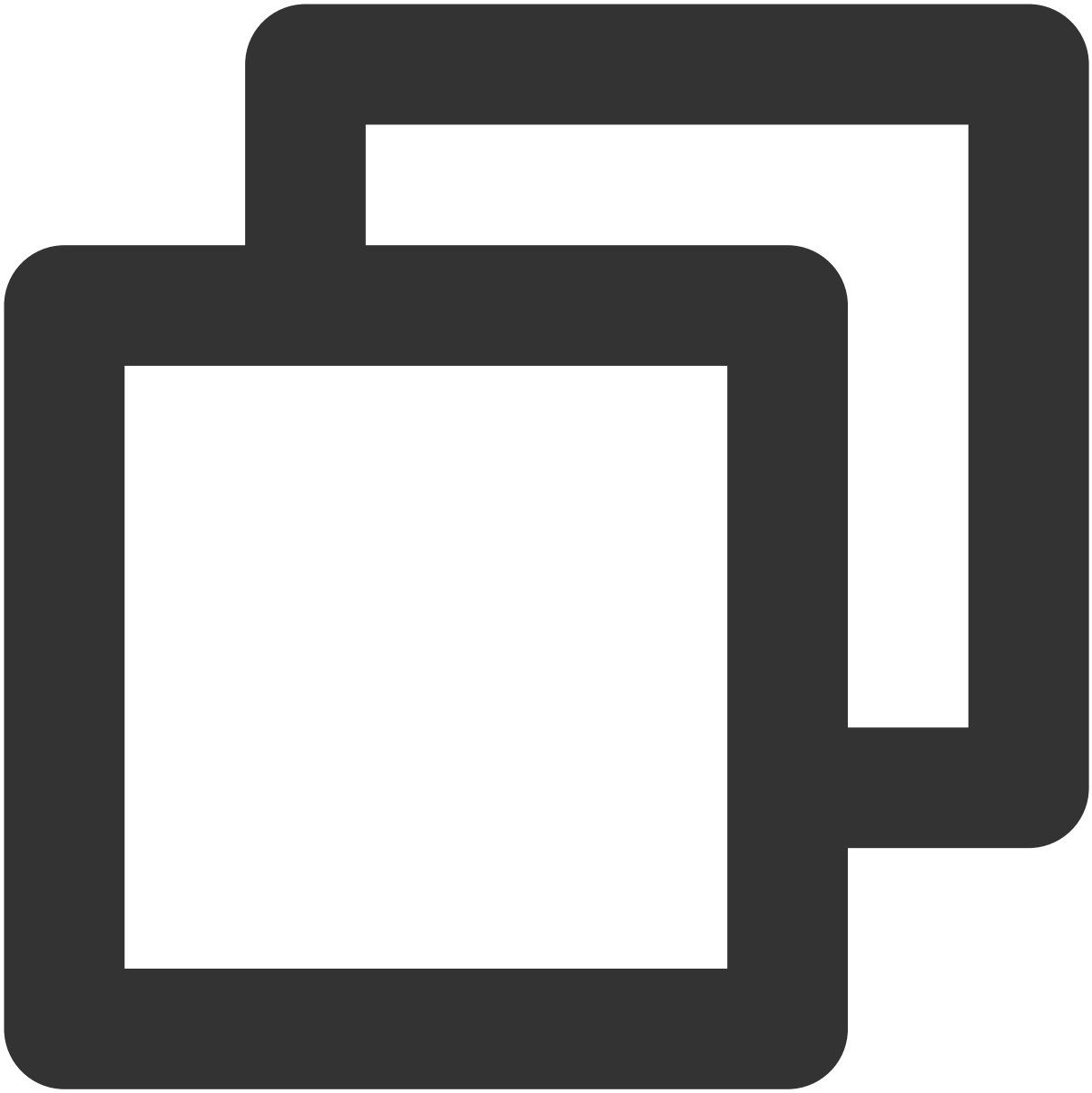
```
git clone https://github.com/Tencent-RTC/TUICallKit.git
```

2. Enter the iOS project directory in the command line:



```
cd TUICallKit/iOS/Example
```

3. Load the dependency library:



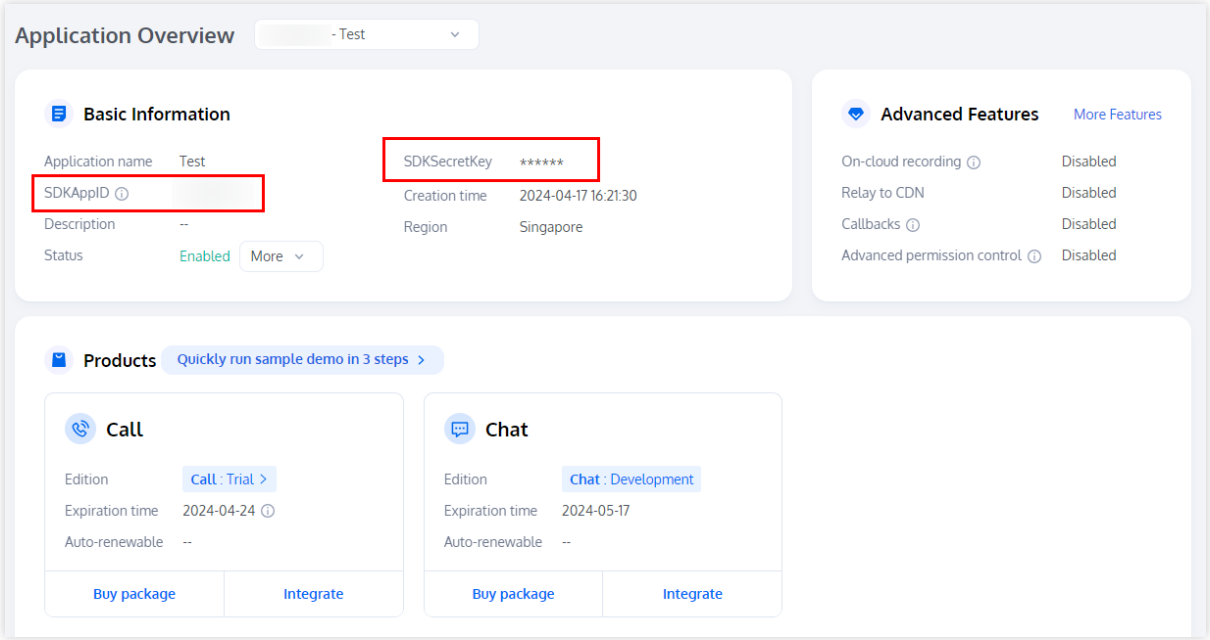
```
pod install
```

Note:

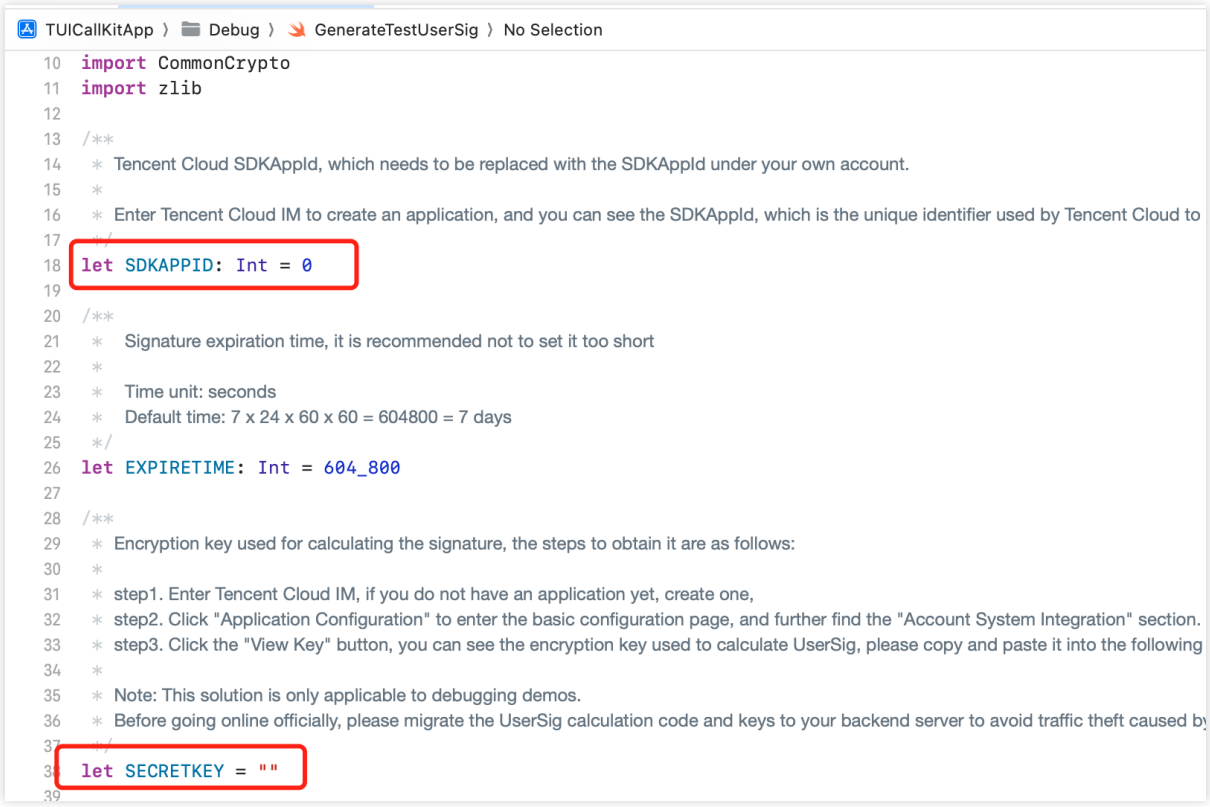
If you haven't installed CocoaPods, you can refer to this for instructions on how to install.

Step 2: Configure the Demo

1. Activate the audio and video services, to obtain the **SDKAppID** and **SDKSecretKey**.

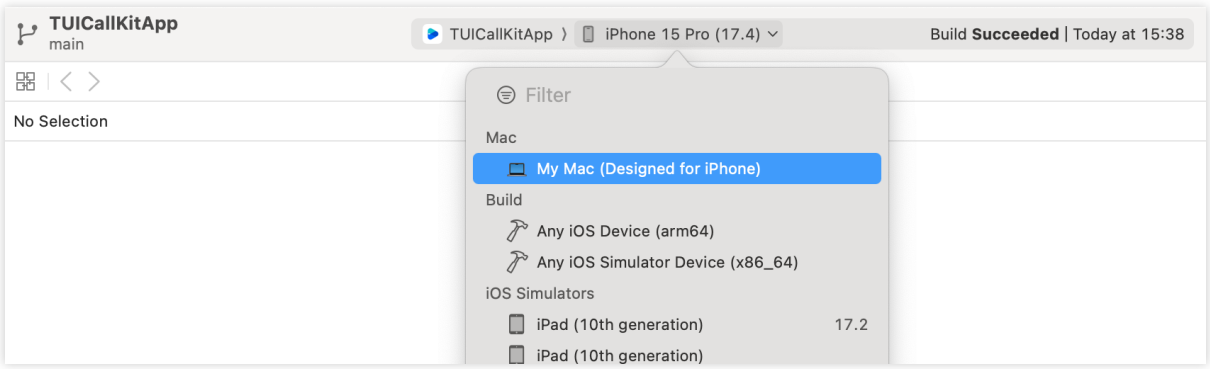


2. Open the `/iOS/Example/Debug/GenerateTestUserSig.swift` file and enter the SDKAppID and SDKSecretKey obtained when activating the service:



Step 3: Running the Demo

1. In XCode, select the device you want to run the Demo on as shown below:



2. After selection, click run to deploy our TUICallKit iOS Demo to the target device.

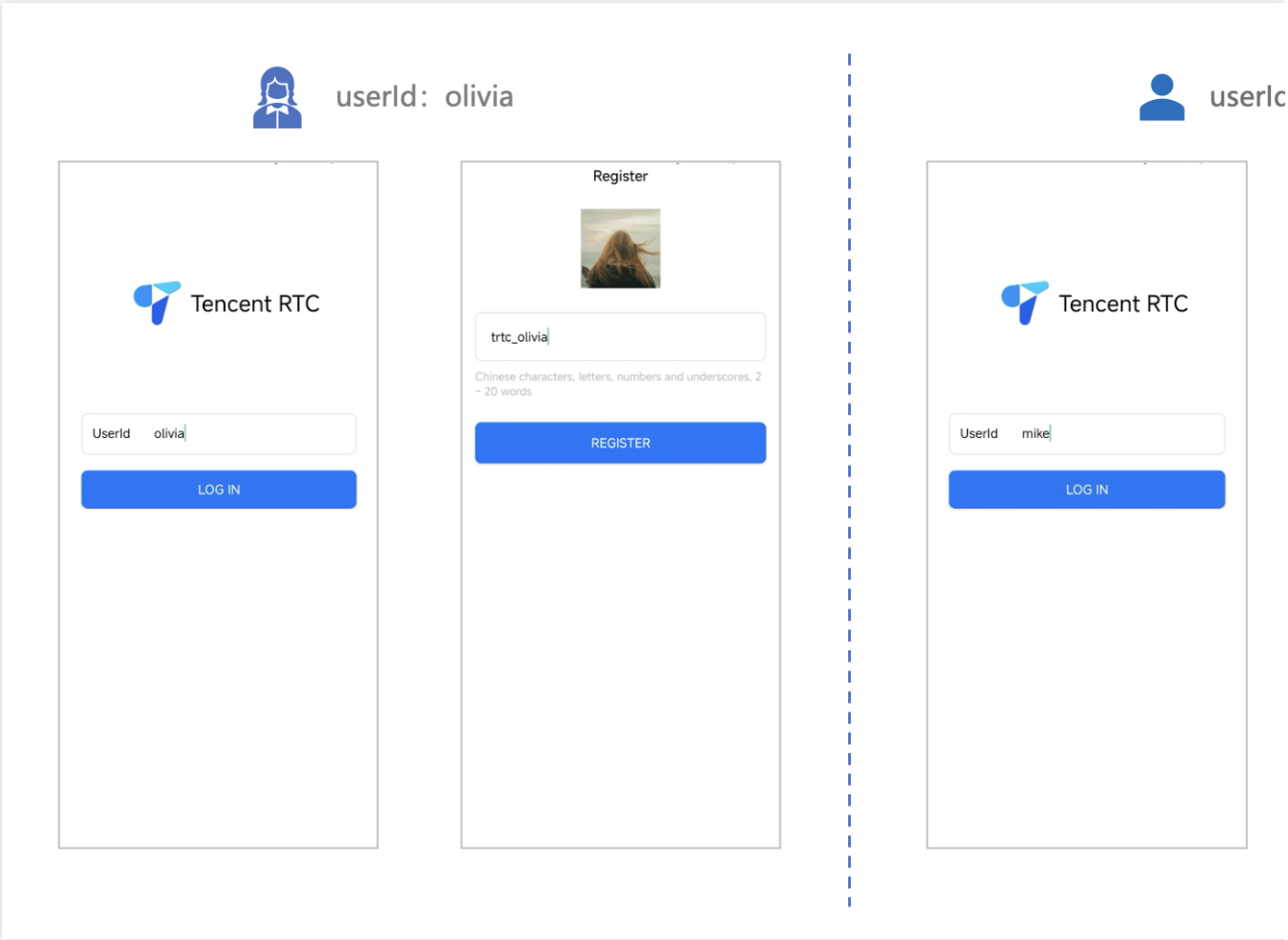
Make the first call

Note:

To experience the complete audio and video calling process, please log into the Demo on two devices as two different users, with one acting as the caller and the other as the callee.

1. Log in & Signup

Please enter the ID at the `User ID` . If your current User ID has not been used before, you will be taken to the **Signup** screen, where you can set your own avatar and nickname.



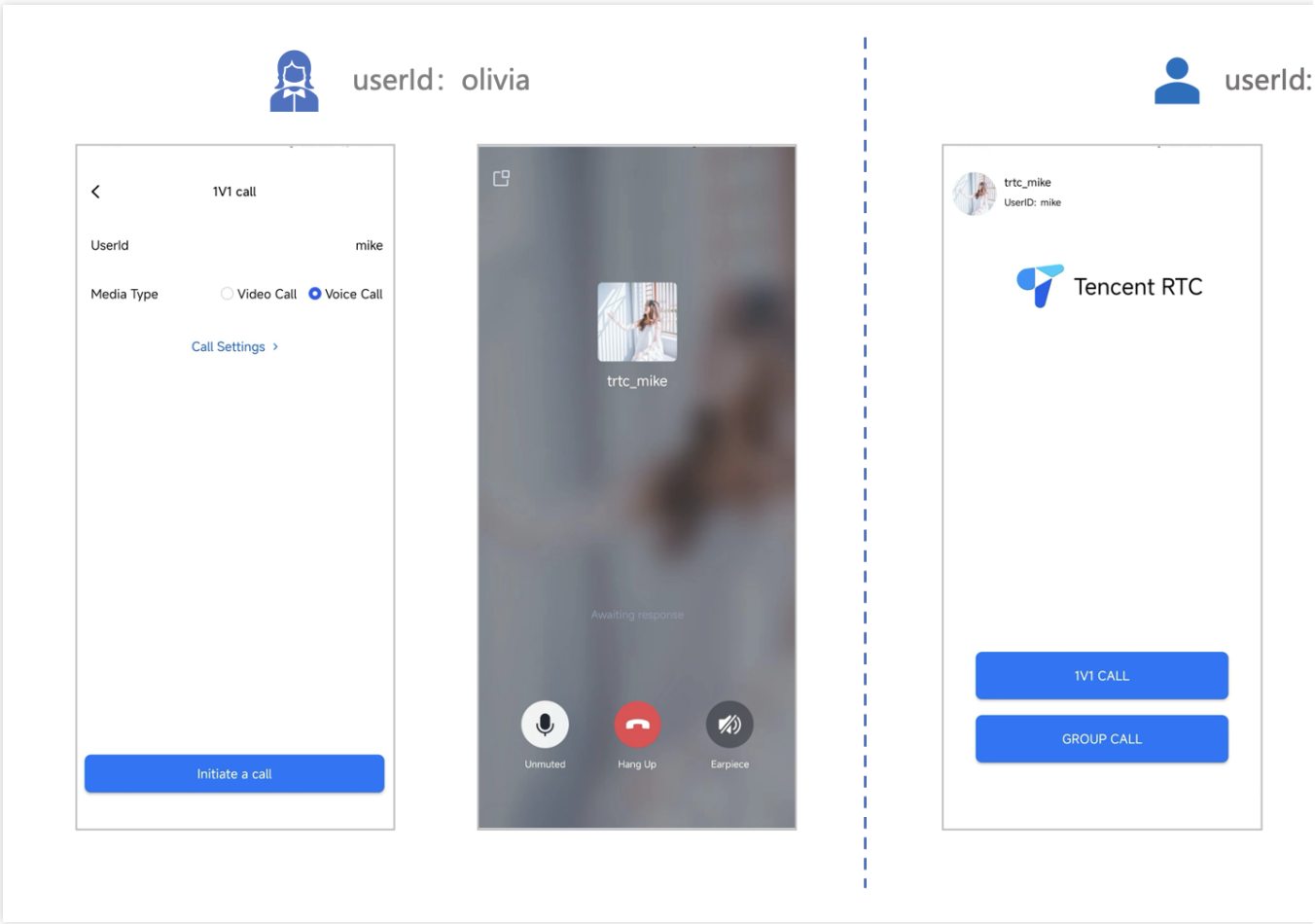
Note:

Try to avoid setting your User ID to simple strings like "1", "123", "111", as TRTC does not support the same User ID being logged into from multiple devices. Such User IDs like "1", "123", "111" are easily occupied by your colleagues during collaborative development, leading to login failures. Therefore, we recommend setting highly recognizable User IDs while debugging.

2. Make a phone call

2.1 The caller should click **1V1 Call** on the interface, enter the callee's User ID in the pop-up interface, and select the desired call type.

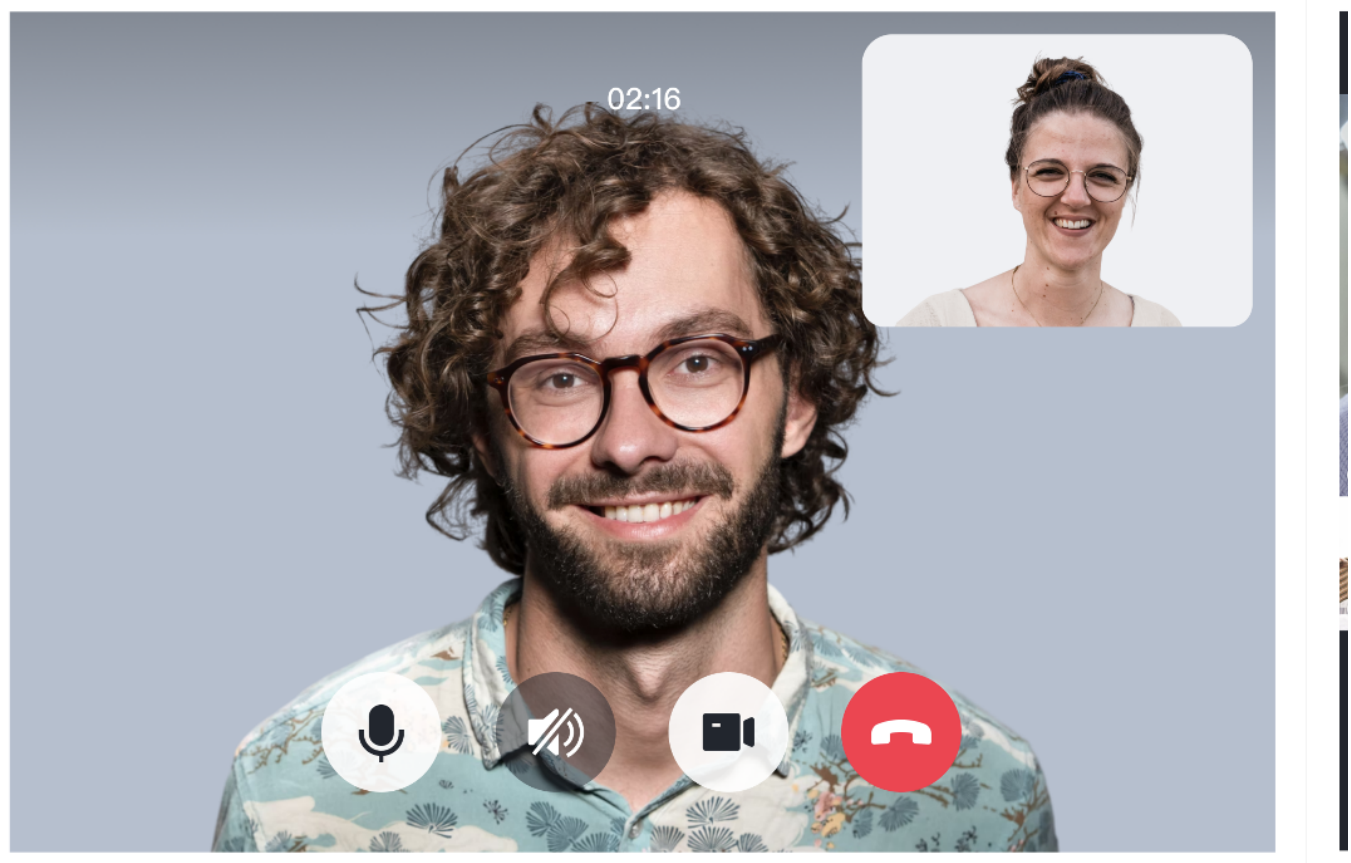
2.2 Click **Initiate Call**.



Web

Last updated：2024-08-09 22:25:01

This article will introduce how to quickly implement an audio and video call demo. You will complete the following key steps within 10 minutes and ultimately obtain a video call feature with a full user interface.

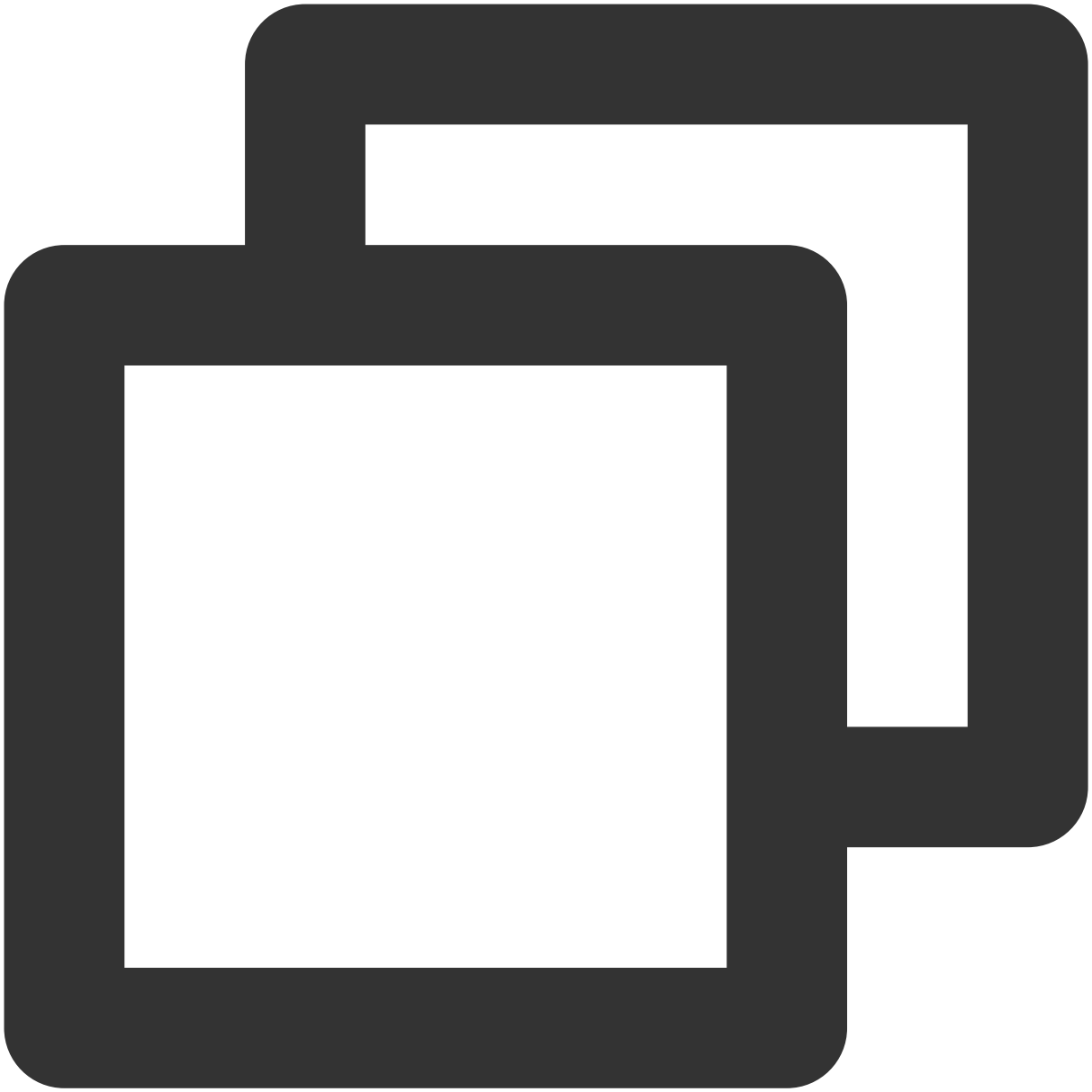


Environment preparations

Node.js version 16+.
Modern browser, supporting WebRTC API.

Step 1: Download the demo

1. Open the terminal and clone the repository.

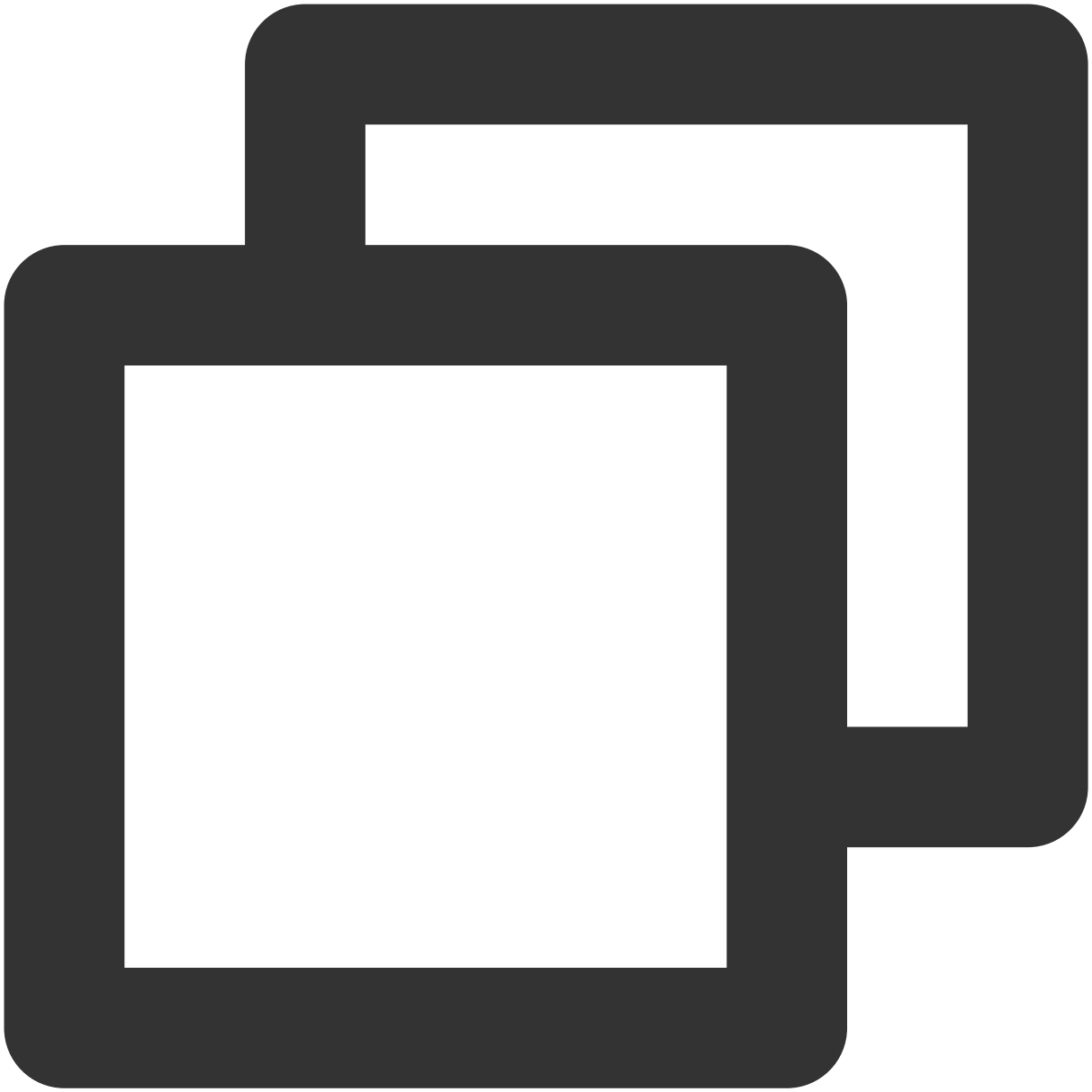


```
git clone https://github.com/Tencent-RTC/TUICallKit.git
```

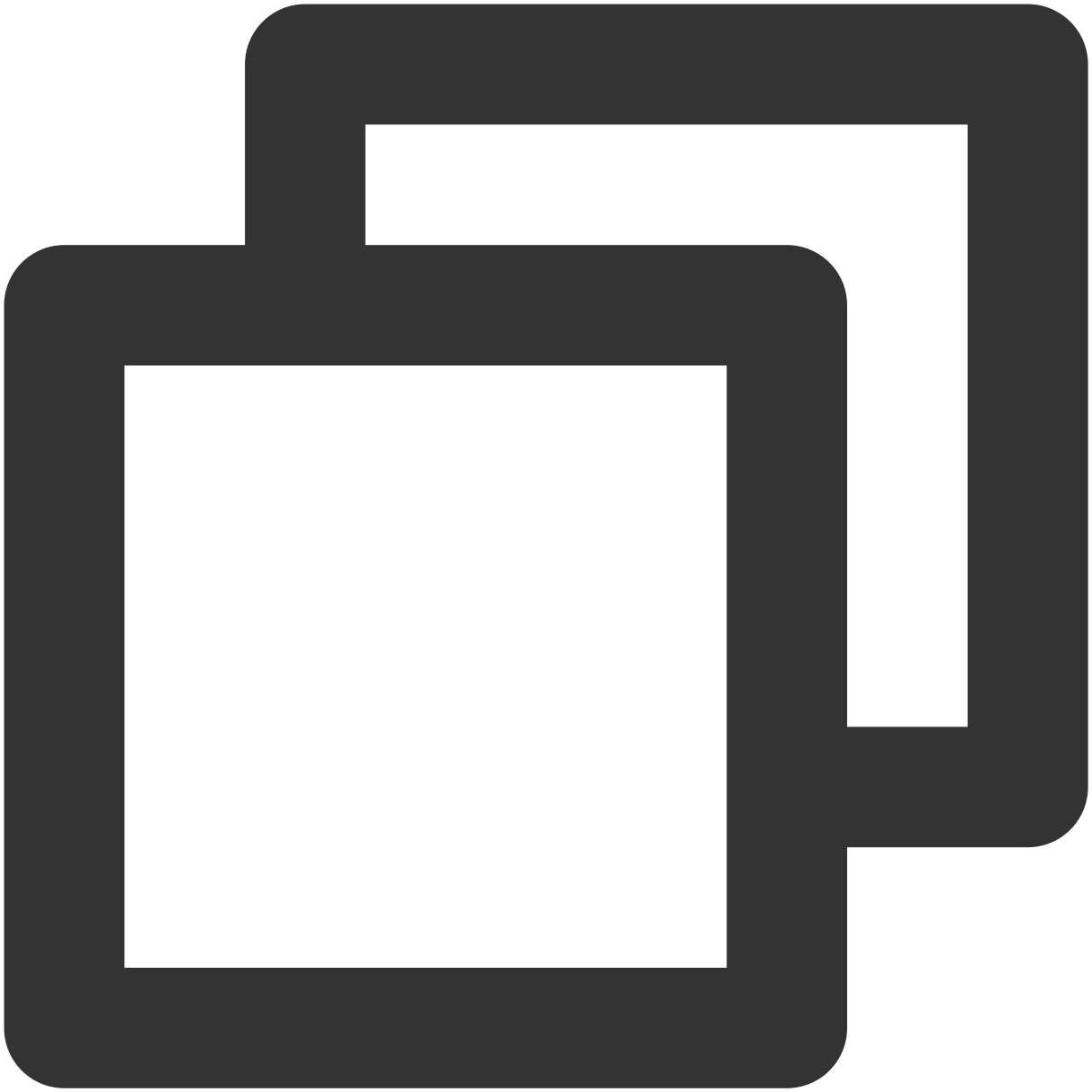
2. Install dependencies.

React

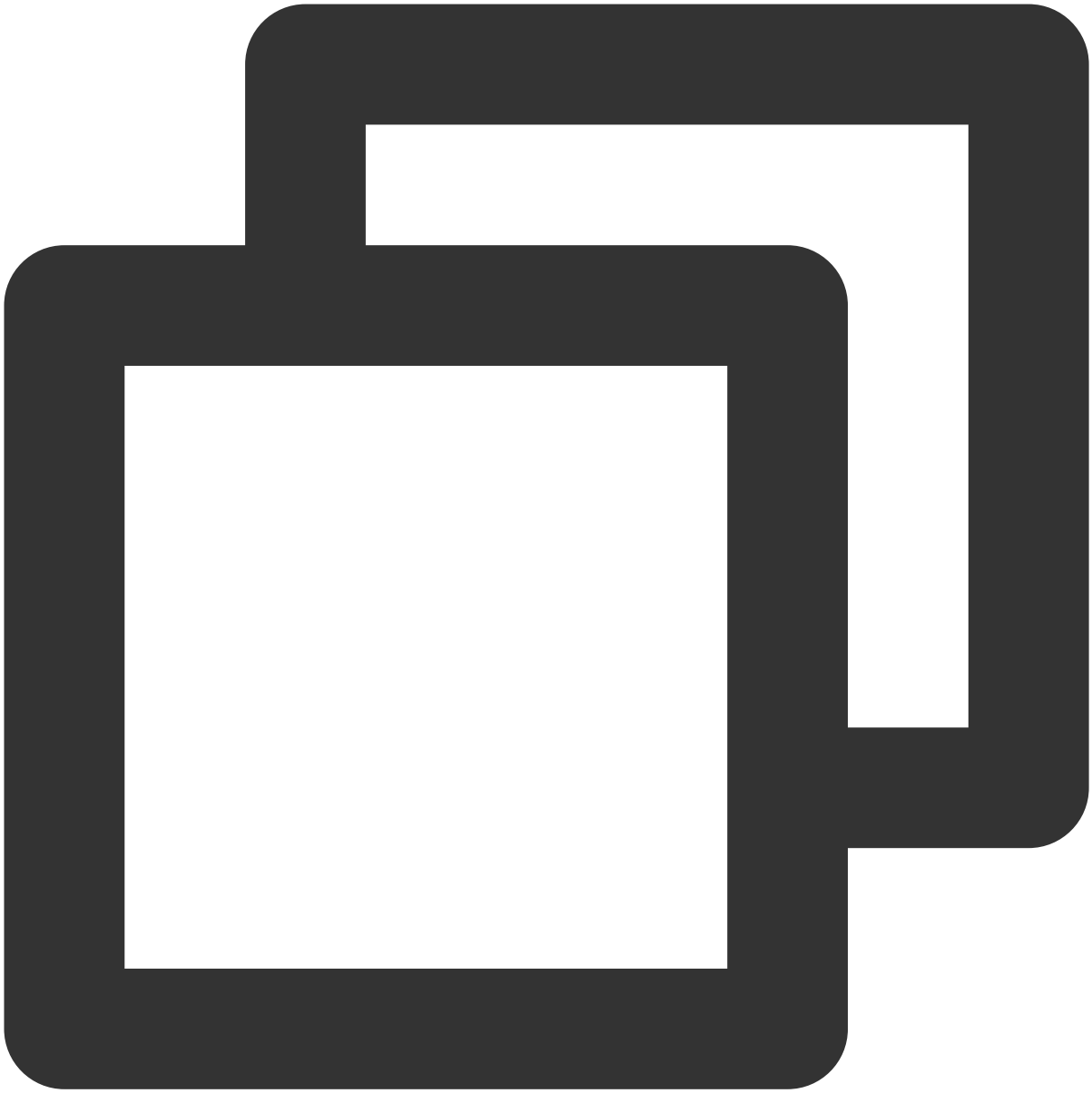
Vue3



```
cd ./TUICallKit/Web/basic-react
```



```
cd ./TUICallKit/Web/basic-vue3
```



```
npm install
```

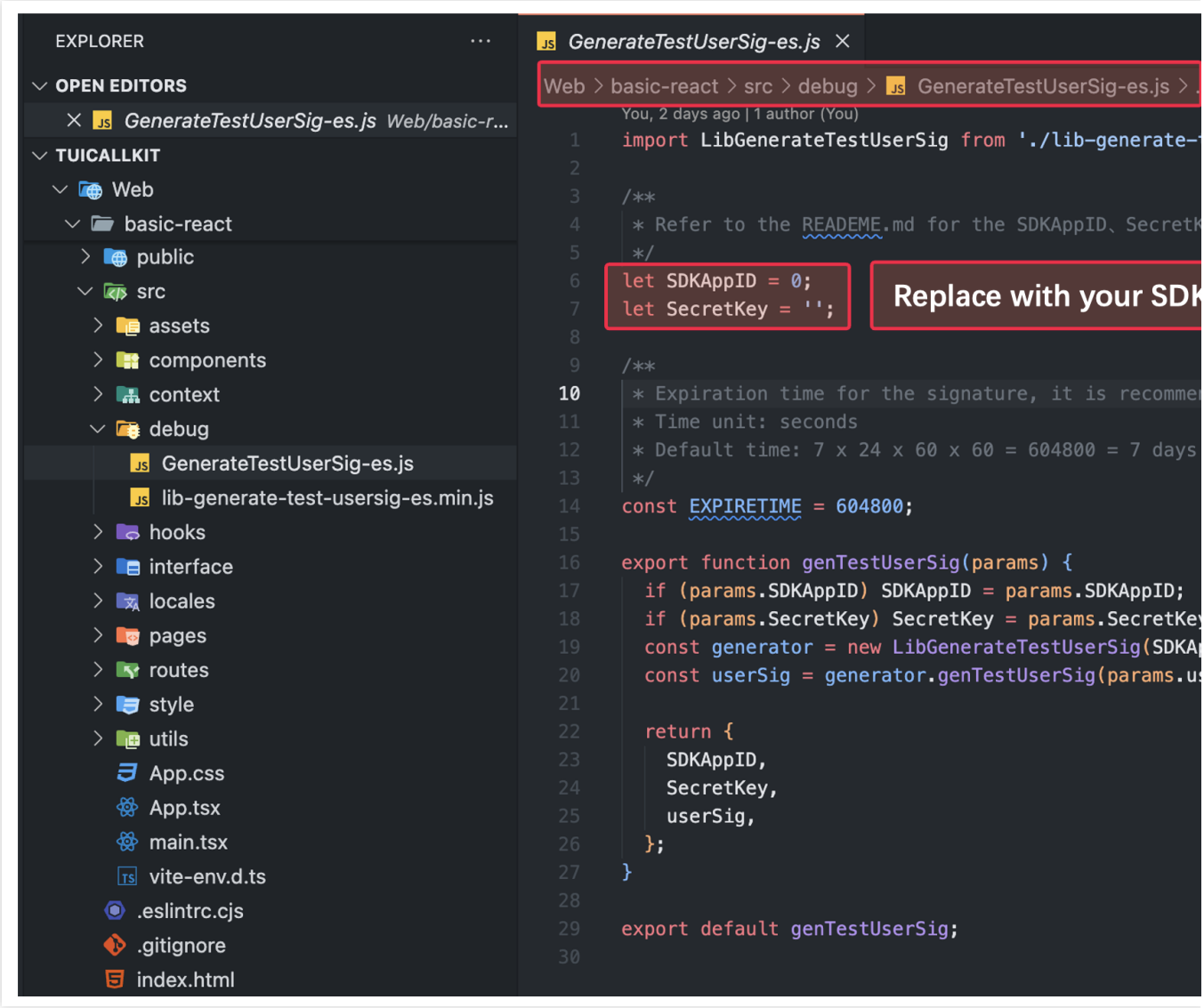
Step 2: Configure the demo

Go to the Activate Service page and get the **`SDKAppID and SDKSecretKey`**, **then fill them in the** `GenerateTestUserSig-es.js` file.

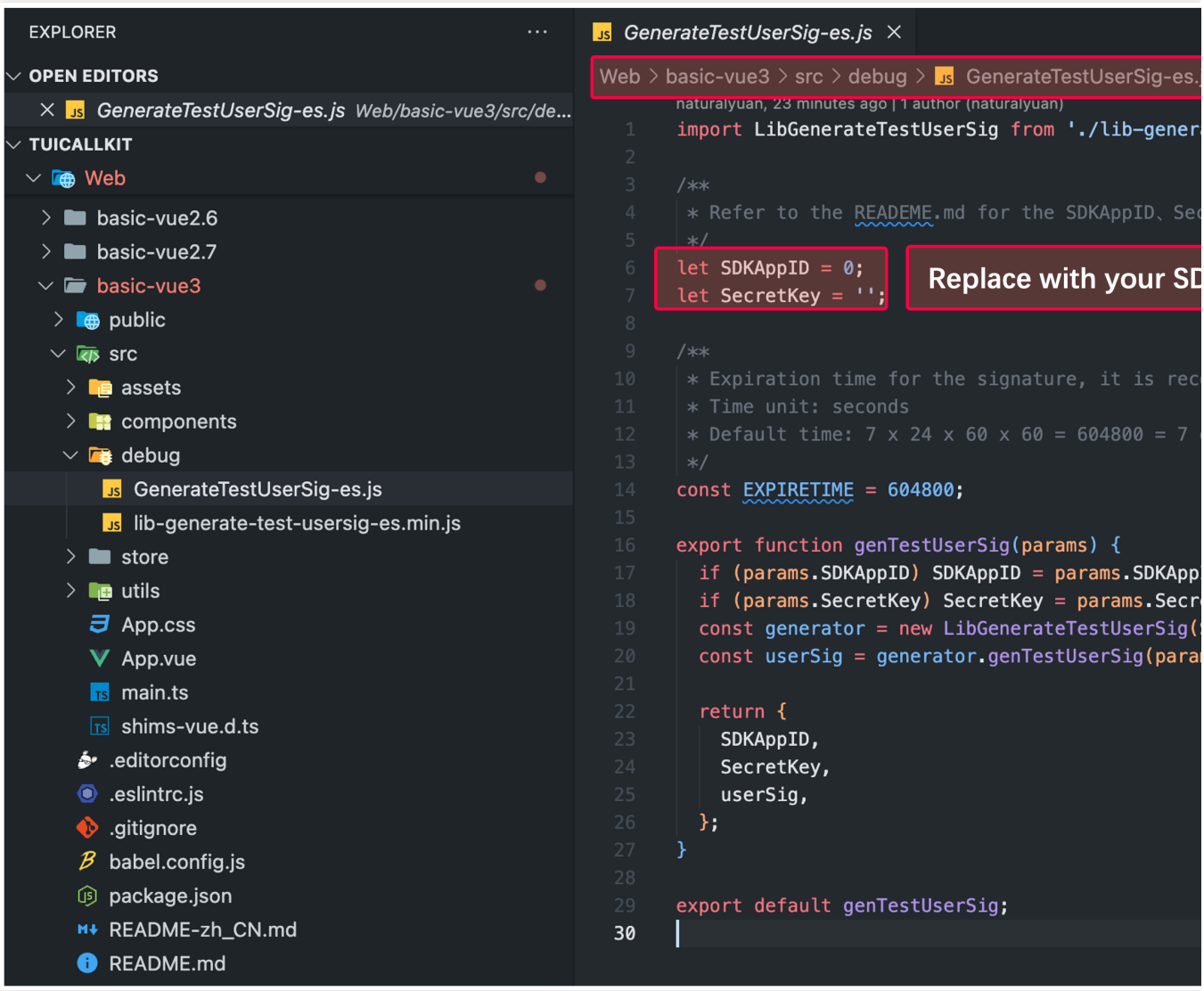
React

Vue3

File path: `TUICallKit/Web/basic-react/src/debug/GenerateTestUserSig-es.js`



File path: `TUICallKit/Web/basic-vue3/src/debug/GenerateTestUserSig-es.js`

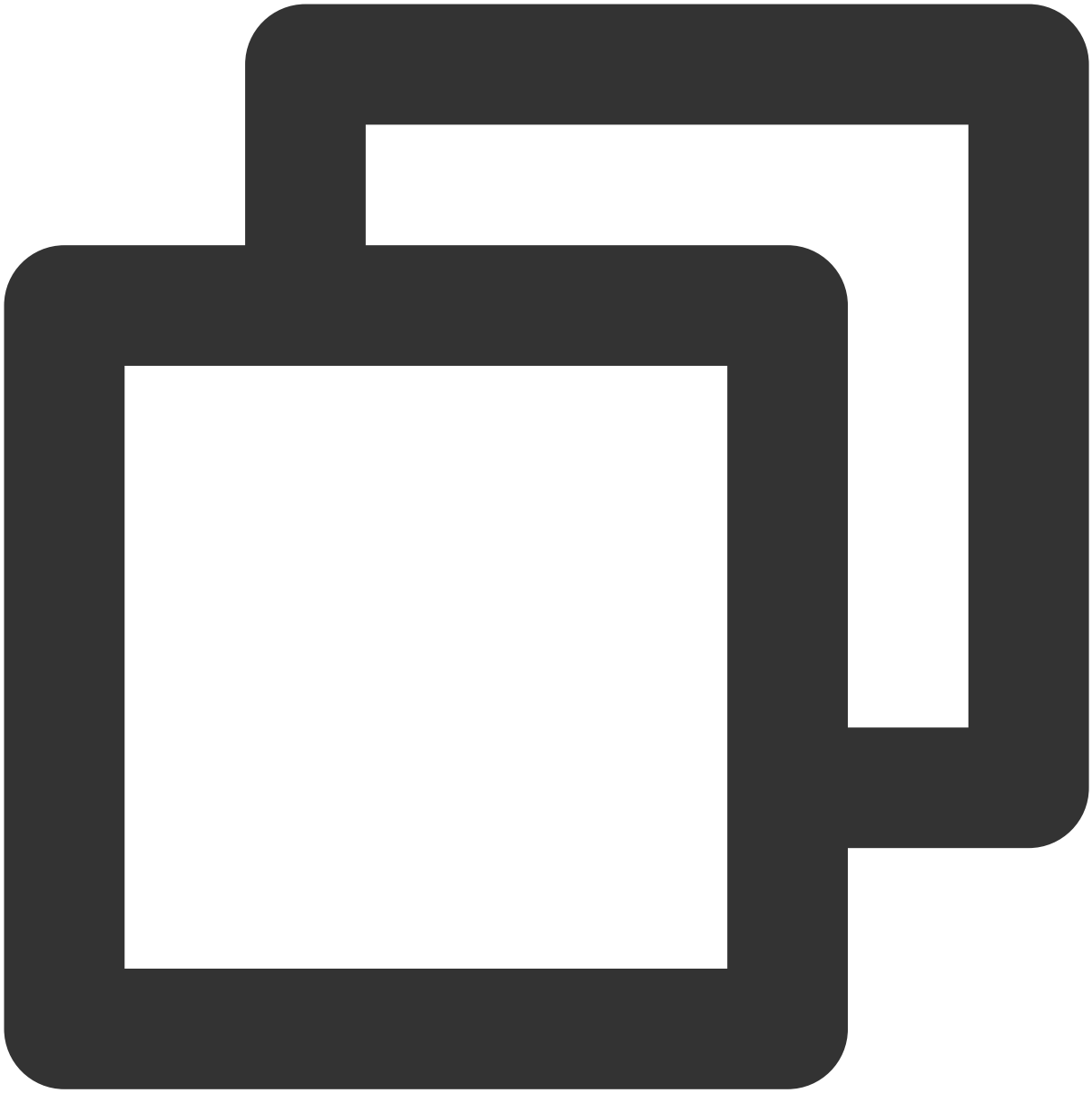


Step 3: Run the demo

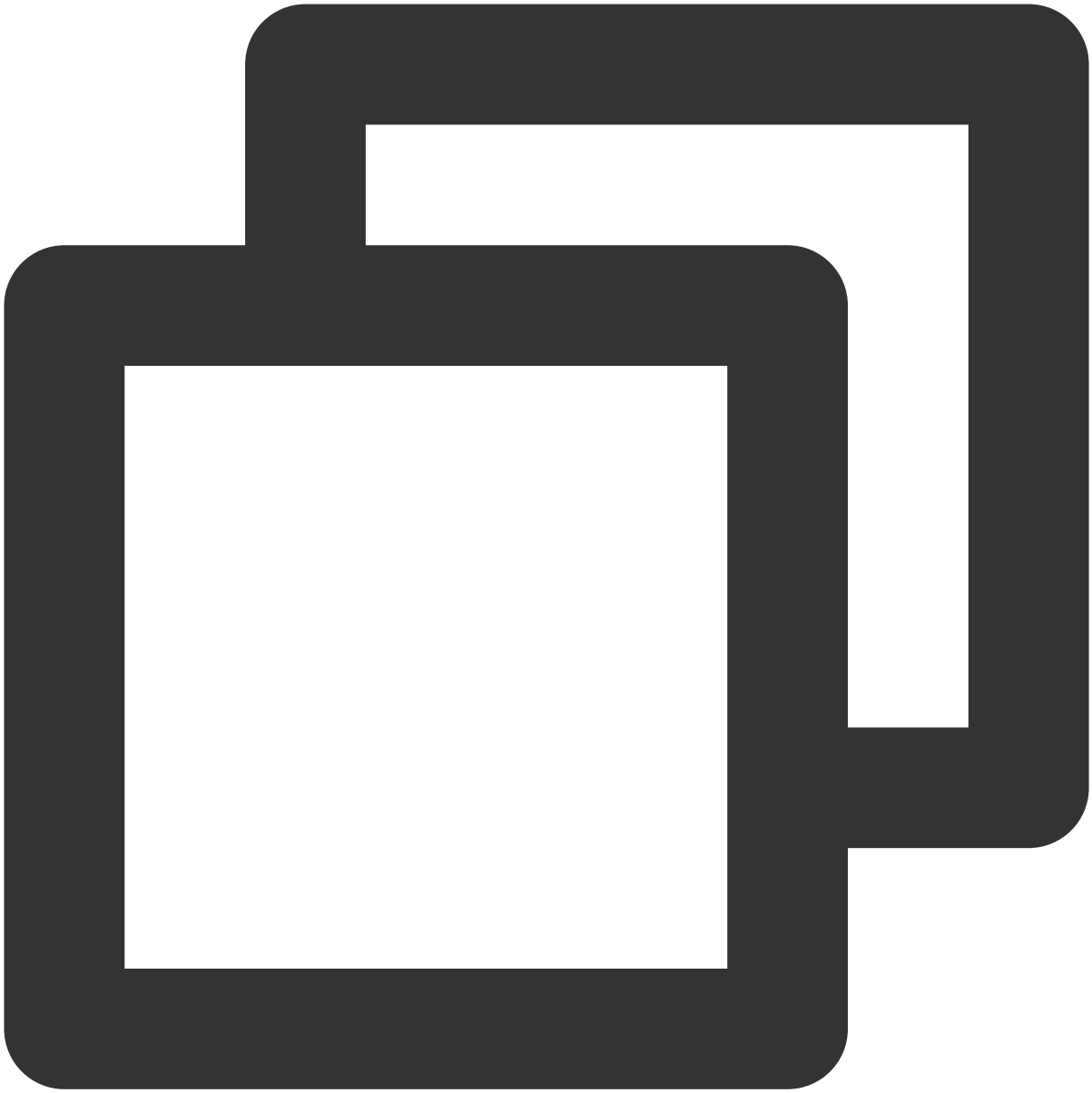
Open the terminal, copy the sample command to run the demo.

TUICallKit/Web/basic-react

TUICallKit/Web/basic-vue3



```
npm run dev
```



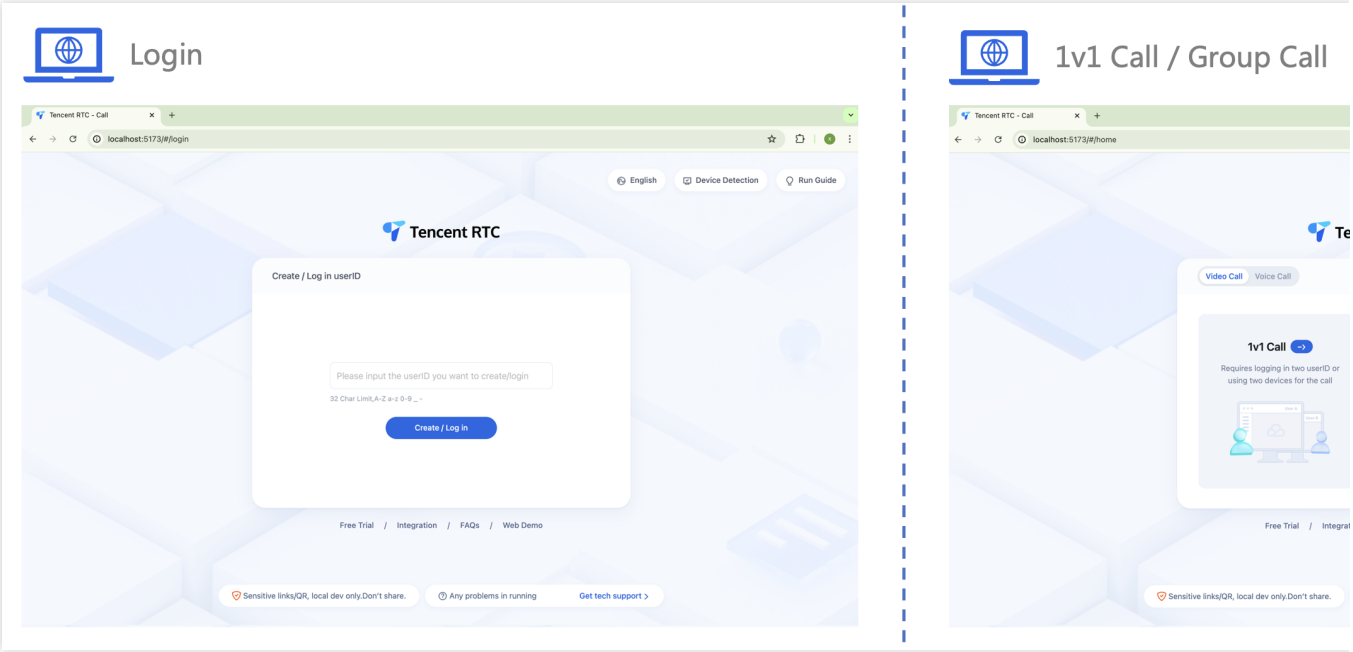
```
npm run serve
```

Warning:

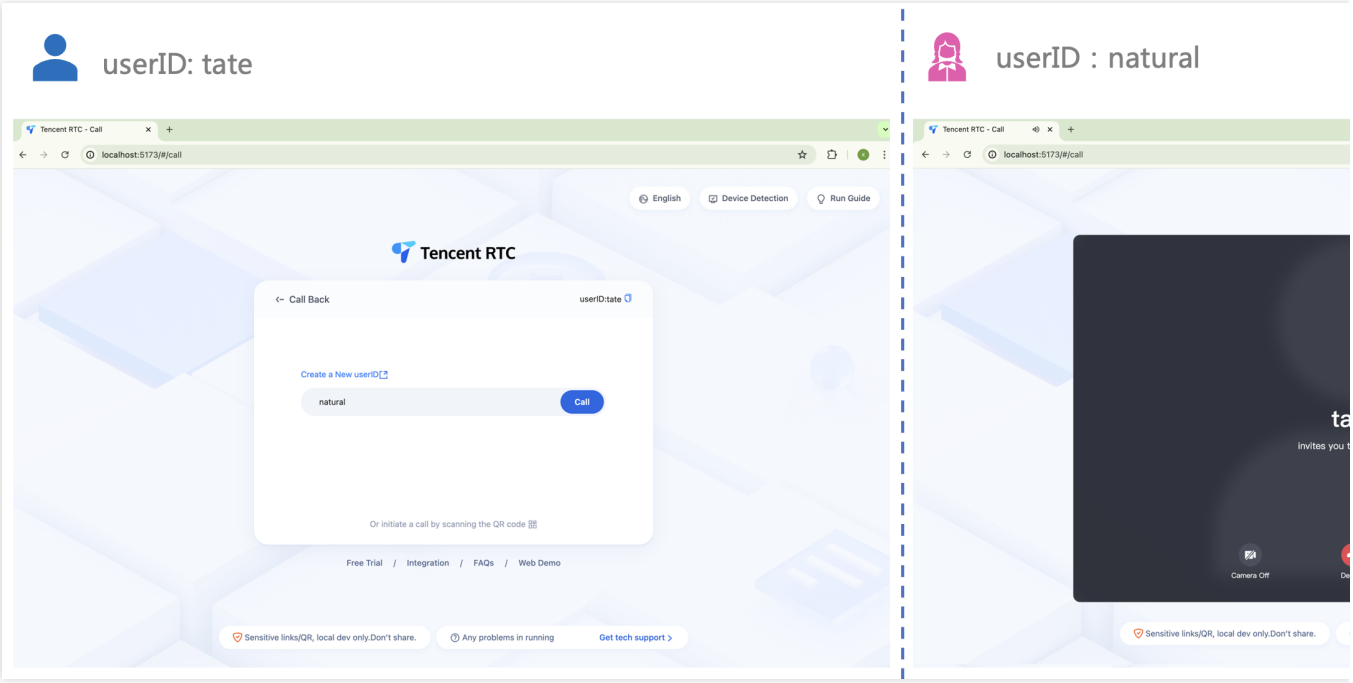
For local environments, access under the localhost protocol. For public network experience, access under the HTTPS protocol. For details, refer to Network Access Protocol Instructions**.**

Step 4: Make your first call

1. Open the browser page, enter the project run address, and log in to userID (defined by you).



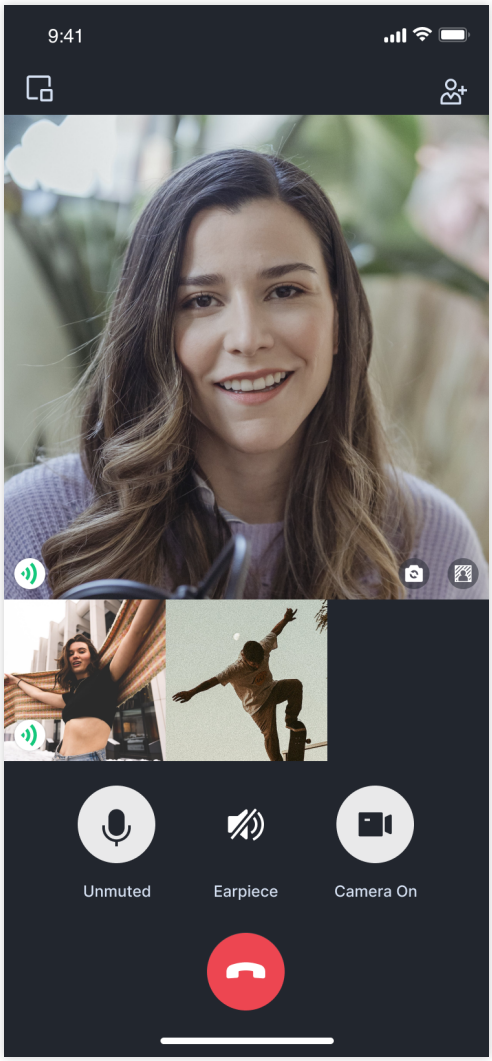
2. Input the callee's userID and click call to experience your first call.



Flutter

Last updated：2024-06-18 17:27:24

This article will guide you on how to quickly run through the Audio and Video Call Demo. By following this document, you can have the Demo up and running in 10 minutes, and ultimately experience an Audio and Video Call feature with a complete UI interface.

1v1 Video Call	Group call
	

Environment preparations

Flutter

Flutter 3.0 or later.

Android

Android Studio 3.5 or above.

Android devices with Android 4.1 or higher versions.

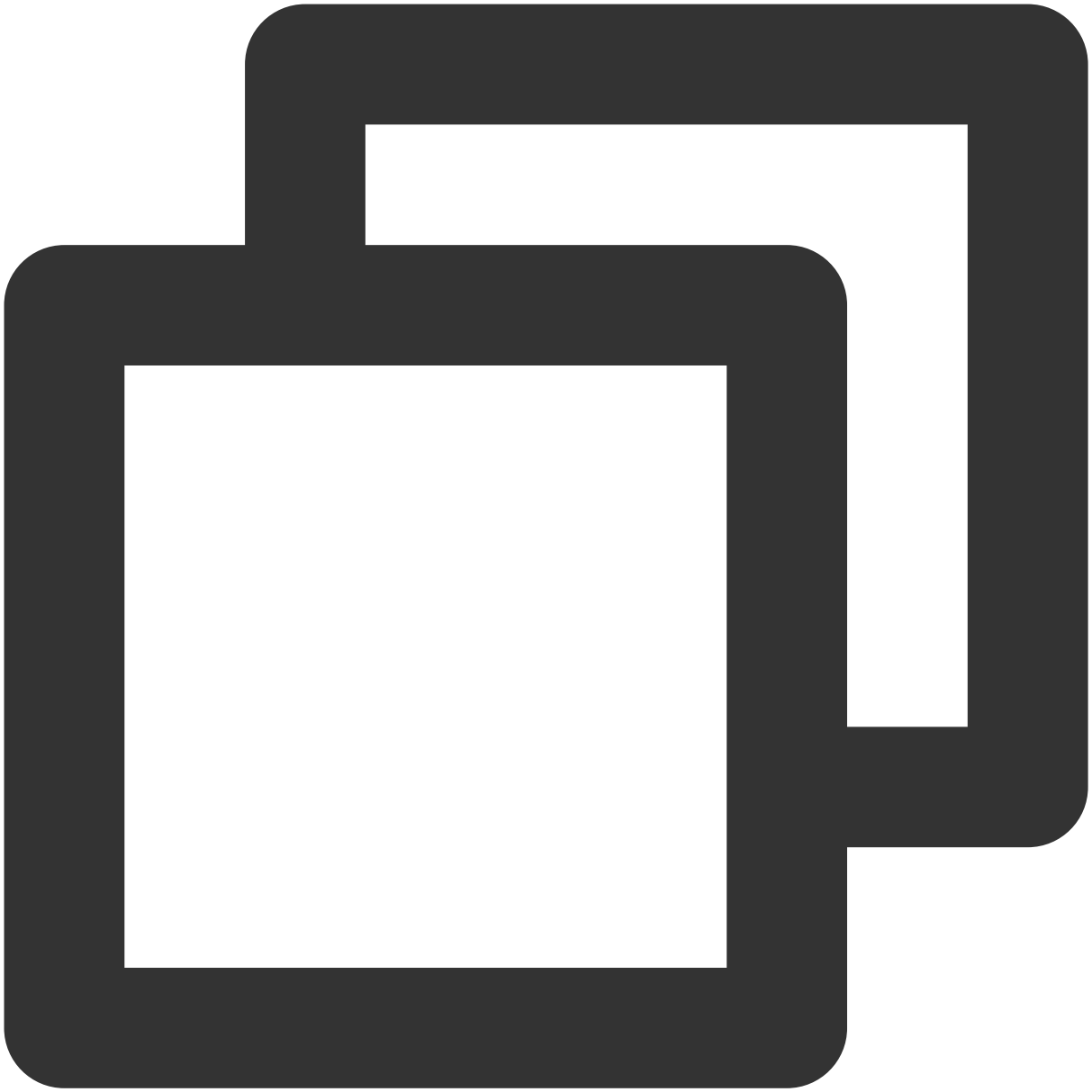
iOS

Xcode 13.0 or above.

Please ensure your project is set up with a valid developer signature.

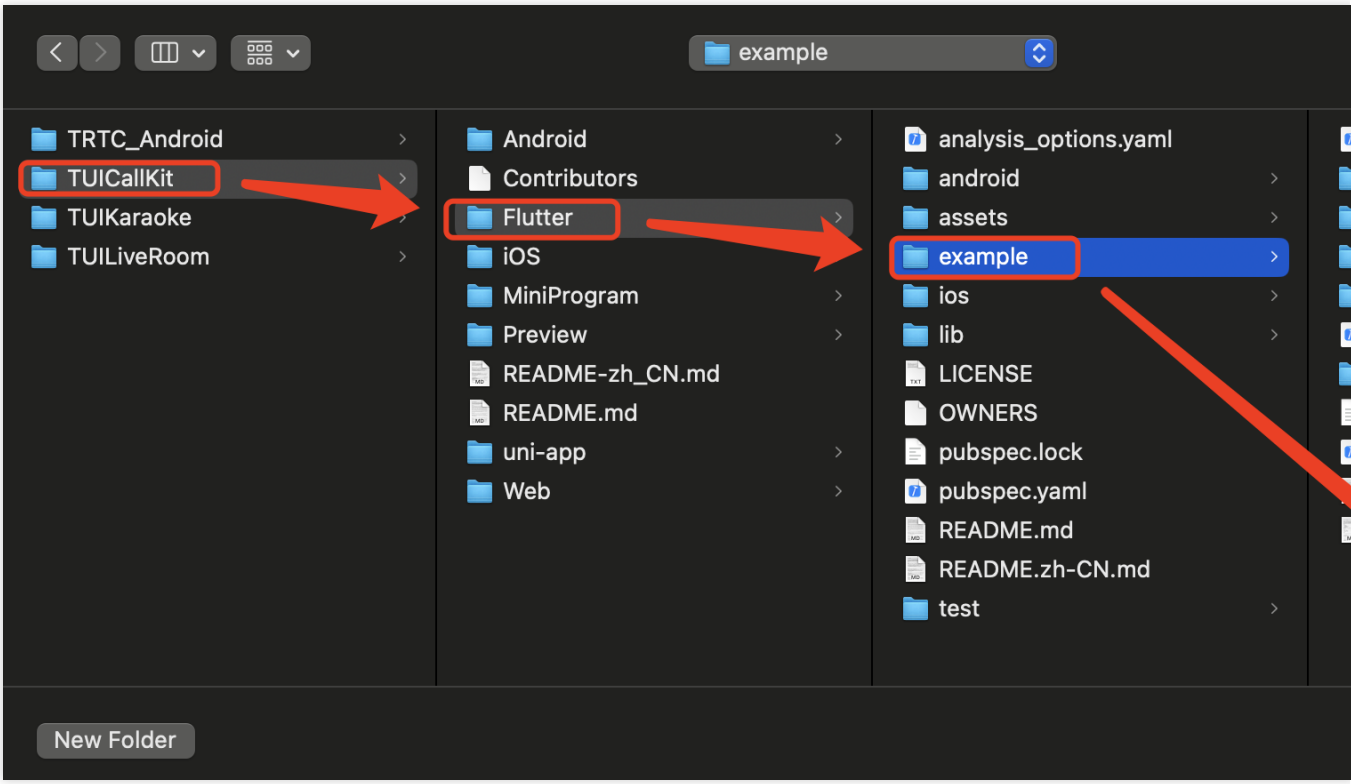
Step 1: Download the Demo

1. Download the TUICallKit Demo source code from GitHub, or directly run the following command in the command line:



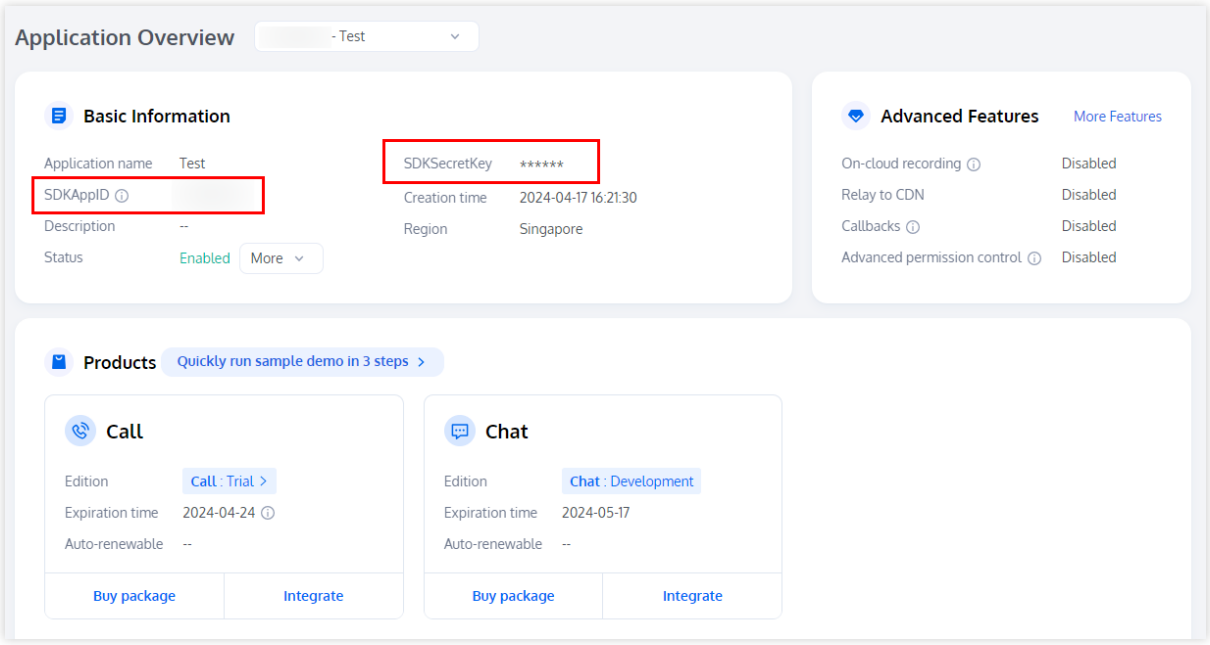
```
git clone https://github.com/Tencent-RTC/TUICallKit.git
```

2. Open the TUICallKit Flutter example in Android Studio or VSCode. The following process will take Android Studio as an example:

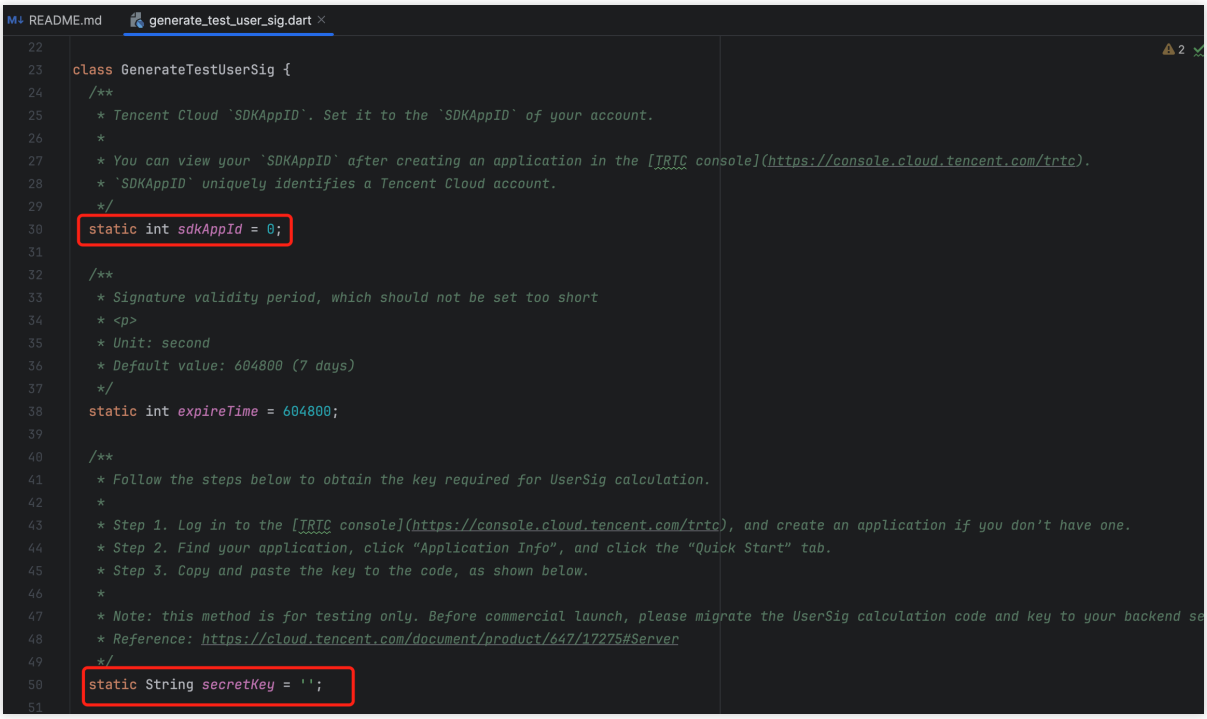


Step 2: Configure the Demo

1. Activate the audio and video services, to obtain the **SDKAppID** and **SDKSecretKey**.

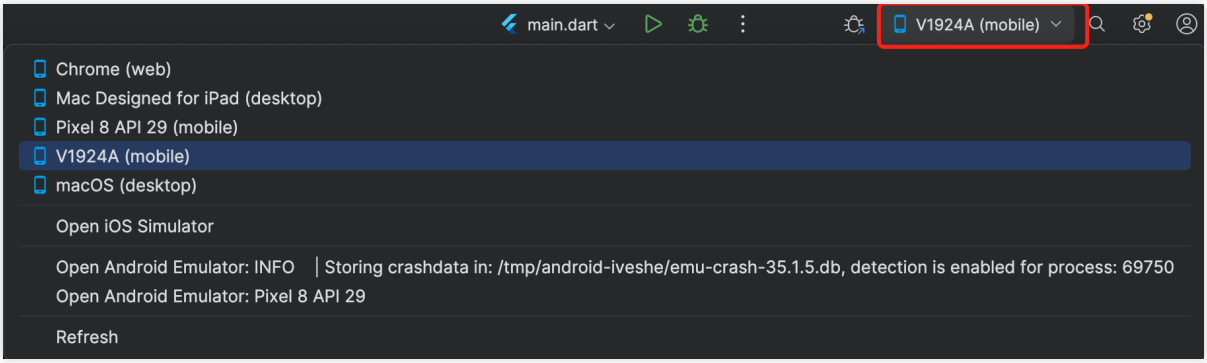


2. Open the `Flutter/example/lib/debug/generate_test_user_sig.dart` file and fill in the SDKAppID and SDKSecretKey obtained when Activating the Service:

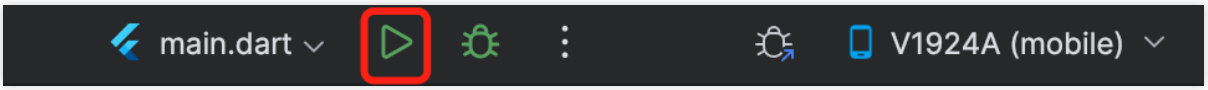


Step 3: Running the Demo

1. In the top right corner of Android Studio, select the device you want to run the Demo on as shown below:



2. After selecting, click Run to execute the TUICallKit Android Demo on the target device.



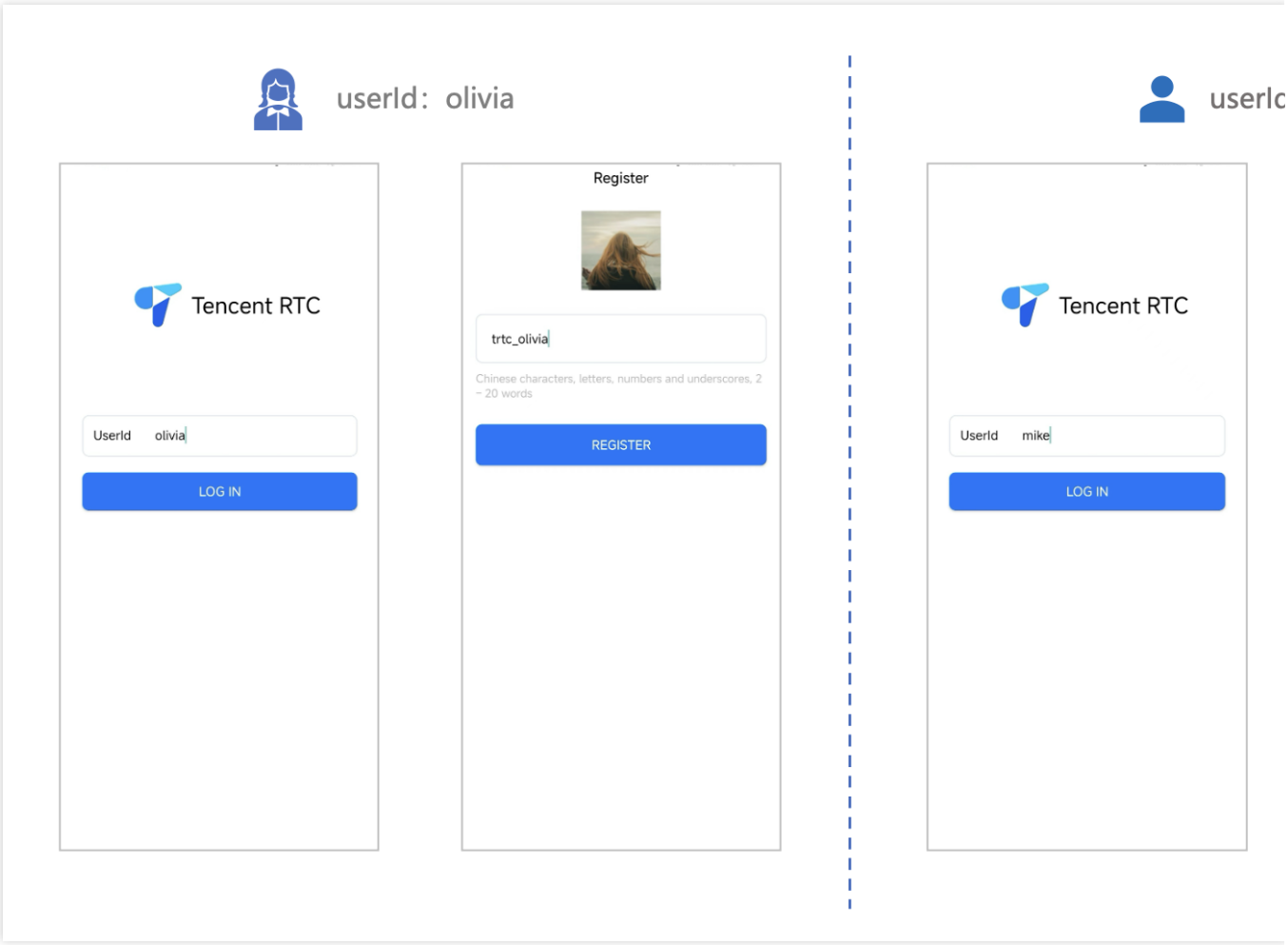
Make the first call

Note:

To experience the complete audio and video calling process, please log into the Demo on two devices as two different users, with one acting as the caller and the other as the callee.

1. Log in & Signup

Please enter the ID at the `User ID`. If your current User ID has not been used before, you will be taken to the **Signup** screen, where you can set your own avatar and nickname.



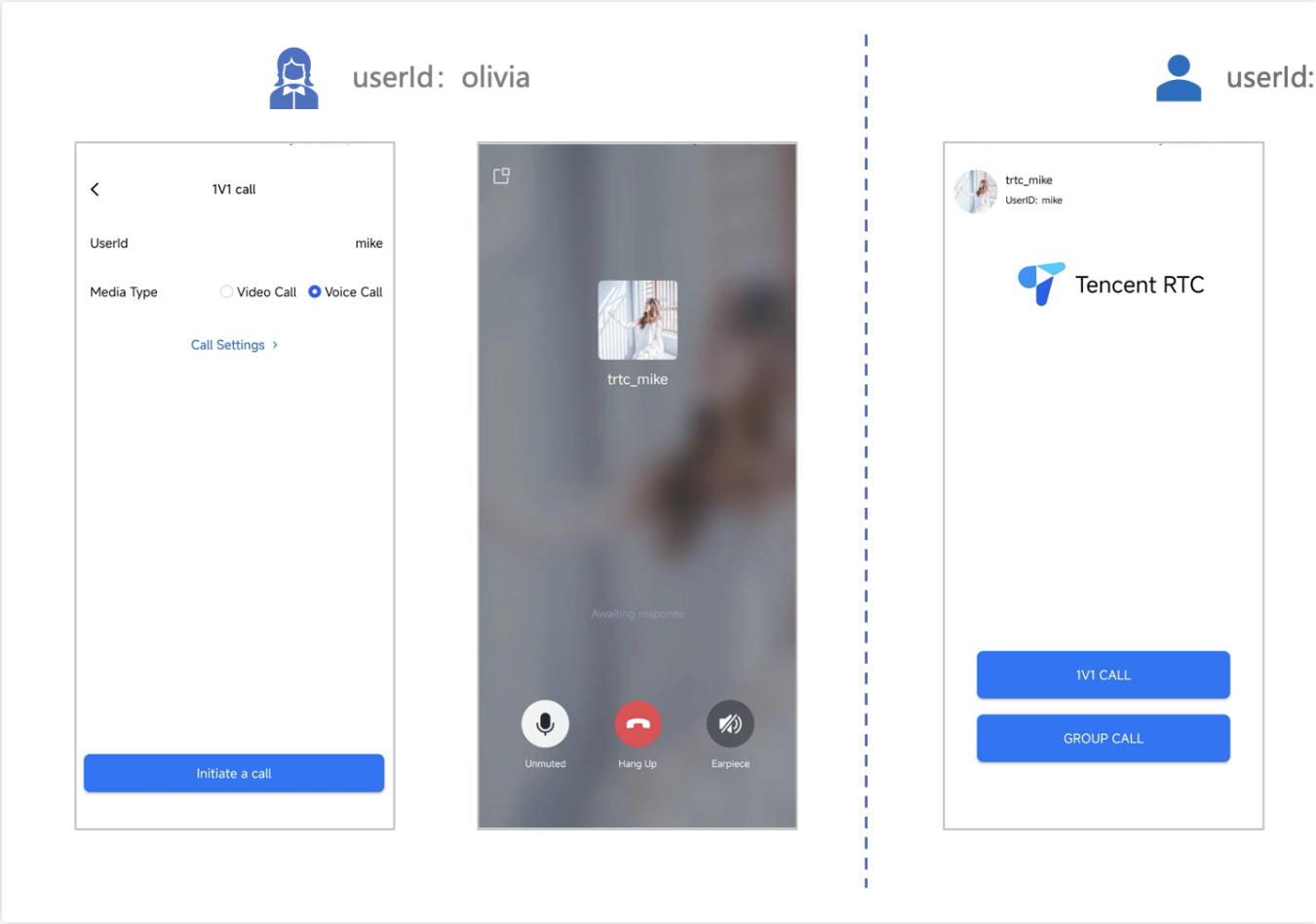
Note:

Try to avoid setting your User ID to simple strings like "1", "123", "111", as TRTC does not support the same User ID being logged into from multiple devices. Such User IDs like "1", "123", "111" are easily occupied by your colleagues during collaborative development, leading to login failures. Therefore, we recommend setting highly recognizable User IDs while debugging.

2. Make a phone call

2.1 The caller should click **1V1 Call** on the interface, enter the callee's User ID in the pop-up interface, and select the desired call type.

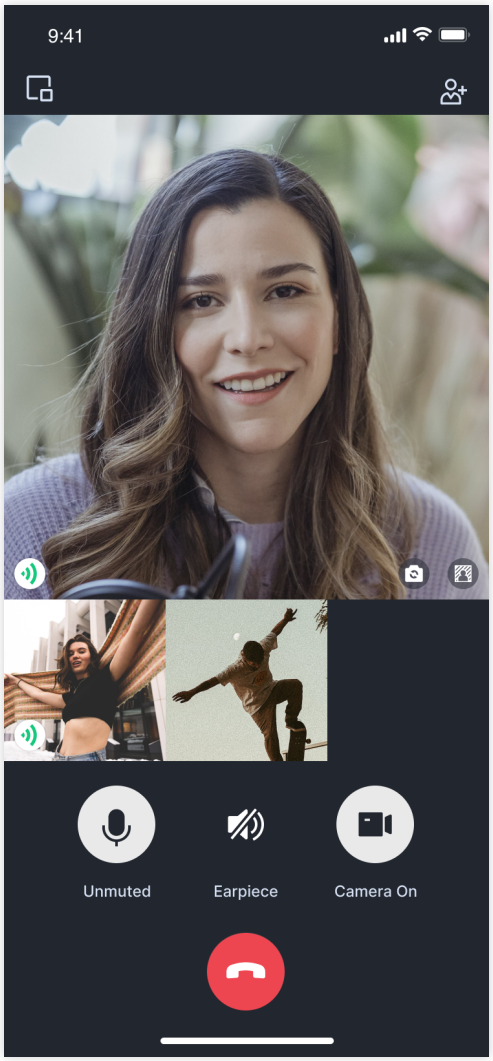
2.2 Click **Initiate Call**.



Integration (TUICallKit)
Android

Last updated：2024-05-31 16:42:21

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

1v1 Video Call	Group call
	

Environment Preparations

Android 5.0 (SDK API level 21) or later.
Gradle 4.2.1 or later
Mobile phone on Android 5.0 or later.

Step 1. Activate the service

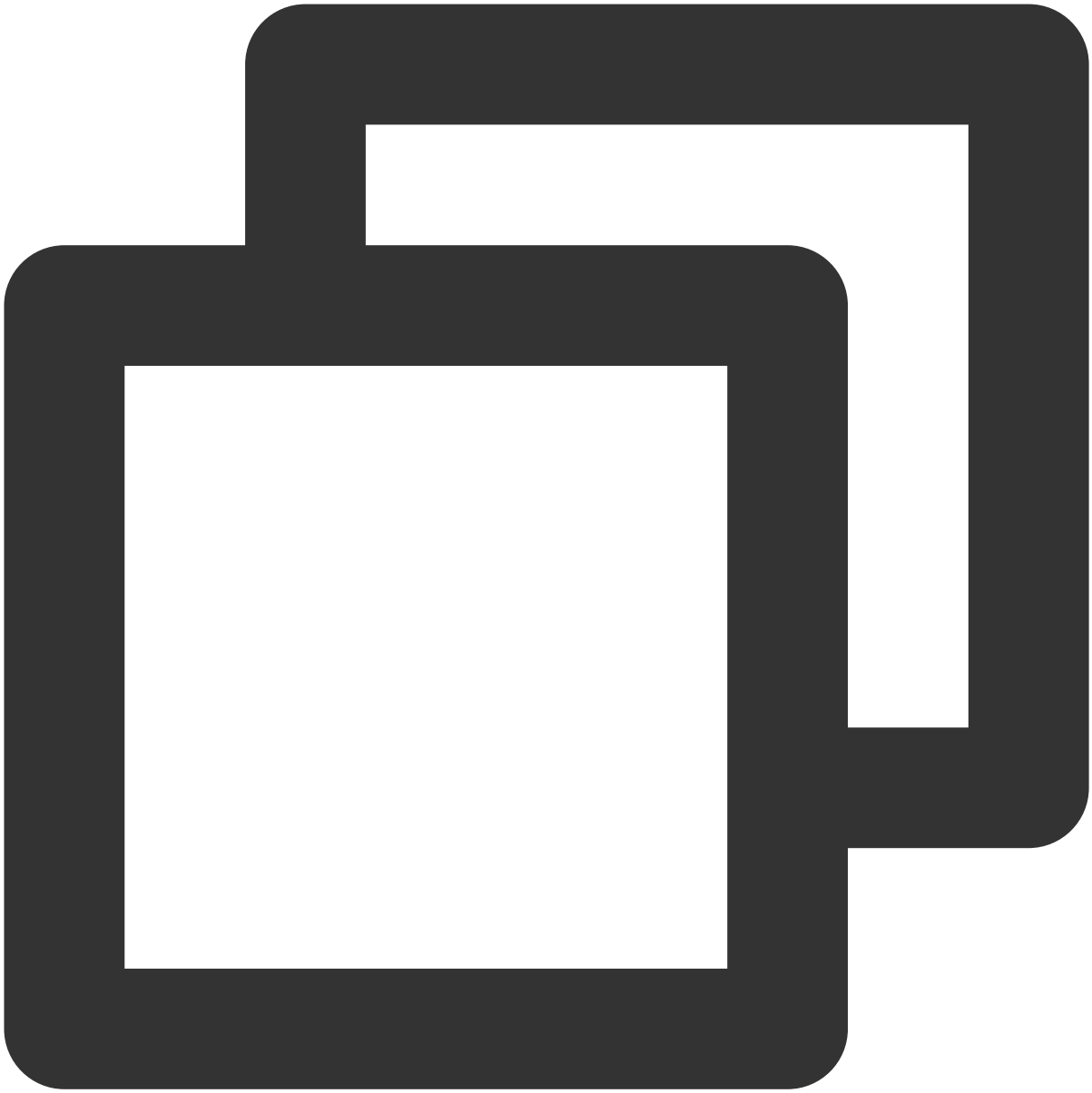
Before using the Audio and Video Services provided by Tencent Cloud, you need to go to the Console and activate the service for your application. For detailed steps, refer to Activate Service. Please record the `SDKAppID` and `SDKSecretKey`, they will be used for the following steps.

Step 2. Download and import the component

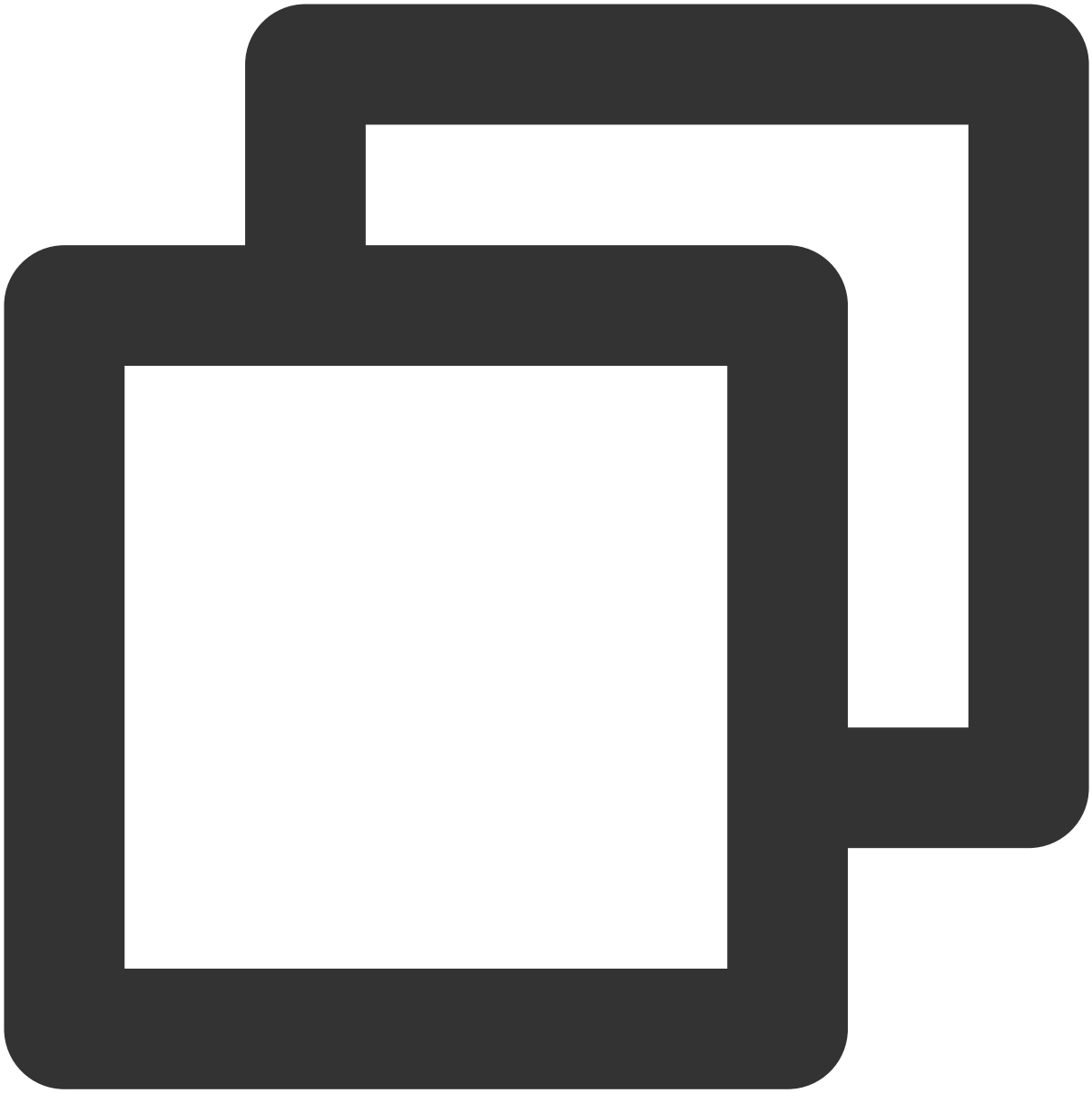
Go to GitHub, clone or download the code, and copy the `tuicallkit-kt` subdirectory in the `Android` directory to the directory at the same level as `app` in your current project, as shown below.

Step 3. Configure the project

1. Find the `settings.gradle.kts(or settings.gradle)` file in the project root directory and add the following code to import the `tuicallkit-kt` component.
setting.gradle.kts
settings.gradle



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

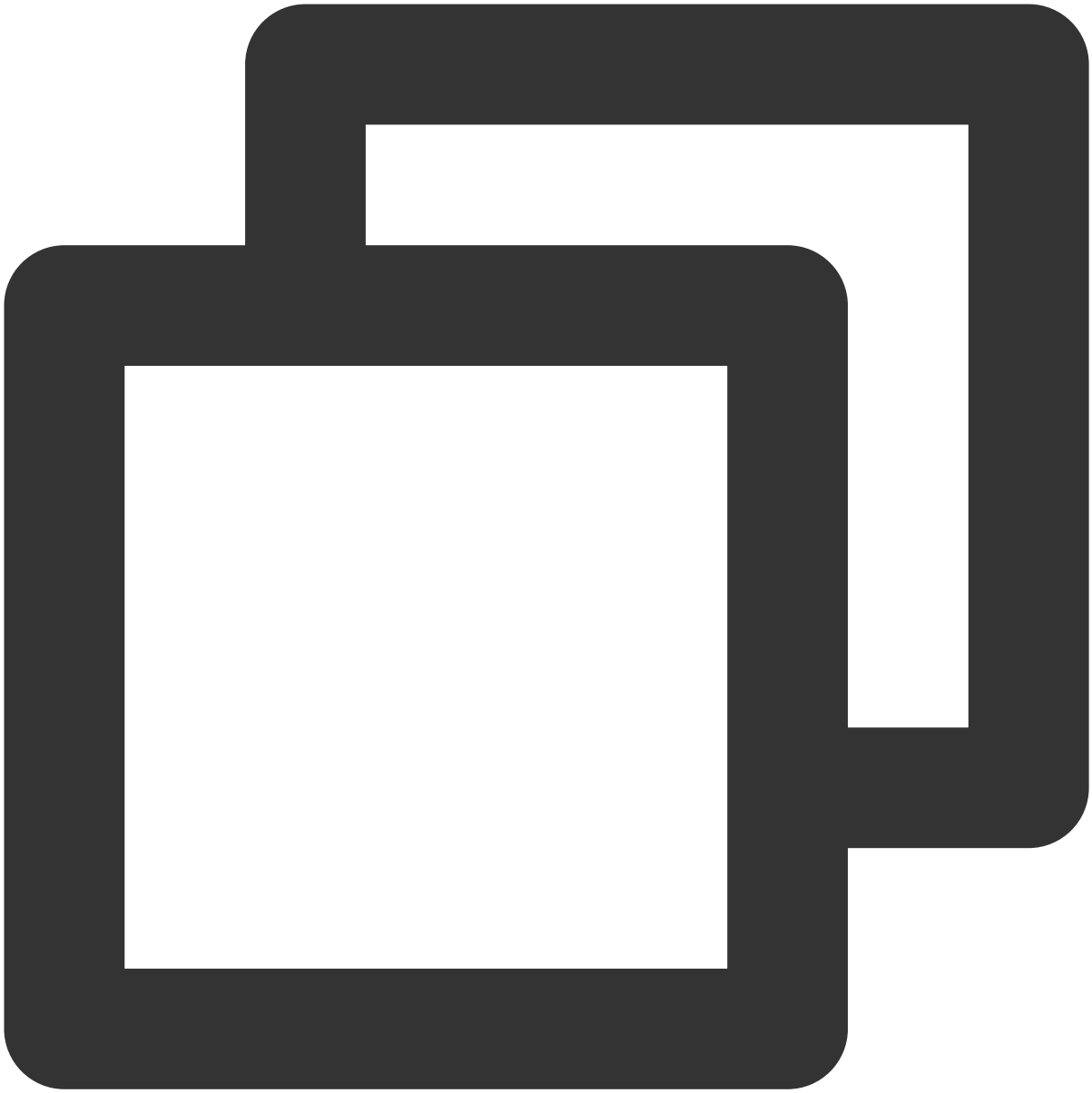


```
include ':tuicallkit-kt'
```

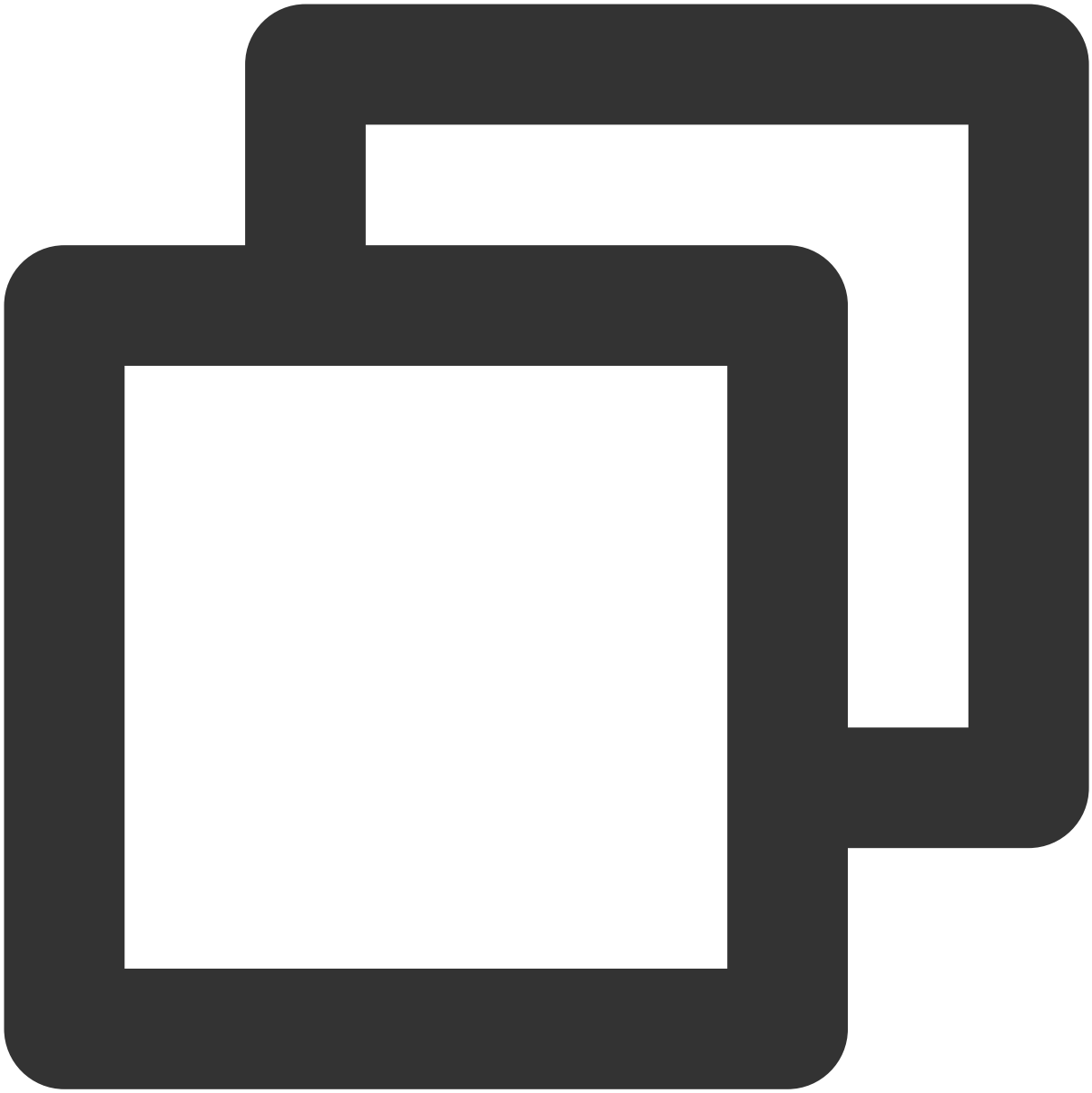
2. Find the `build.gradle.kts` (or `build.gradle`) file in the `app` directory, add the following code to its `dependencies` section. This is to declare the current app's dependency on the newly added components.

build.gradle.kts

build.gradle



```
dependencies {
    api(project(":tuicallkit-kt"))
}
```

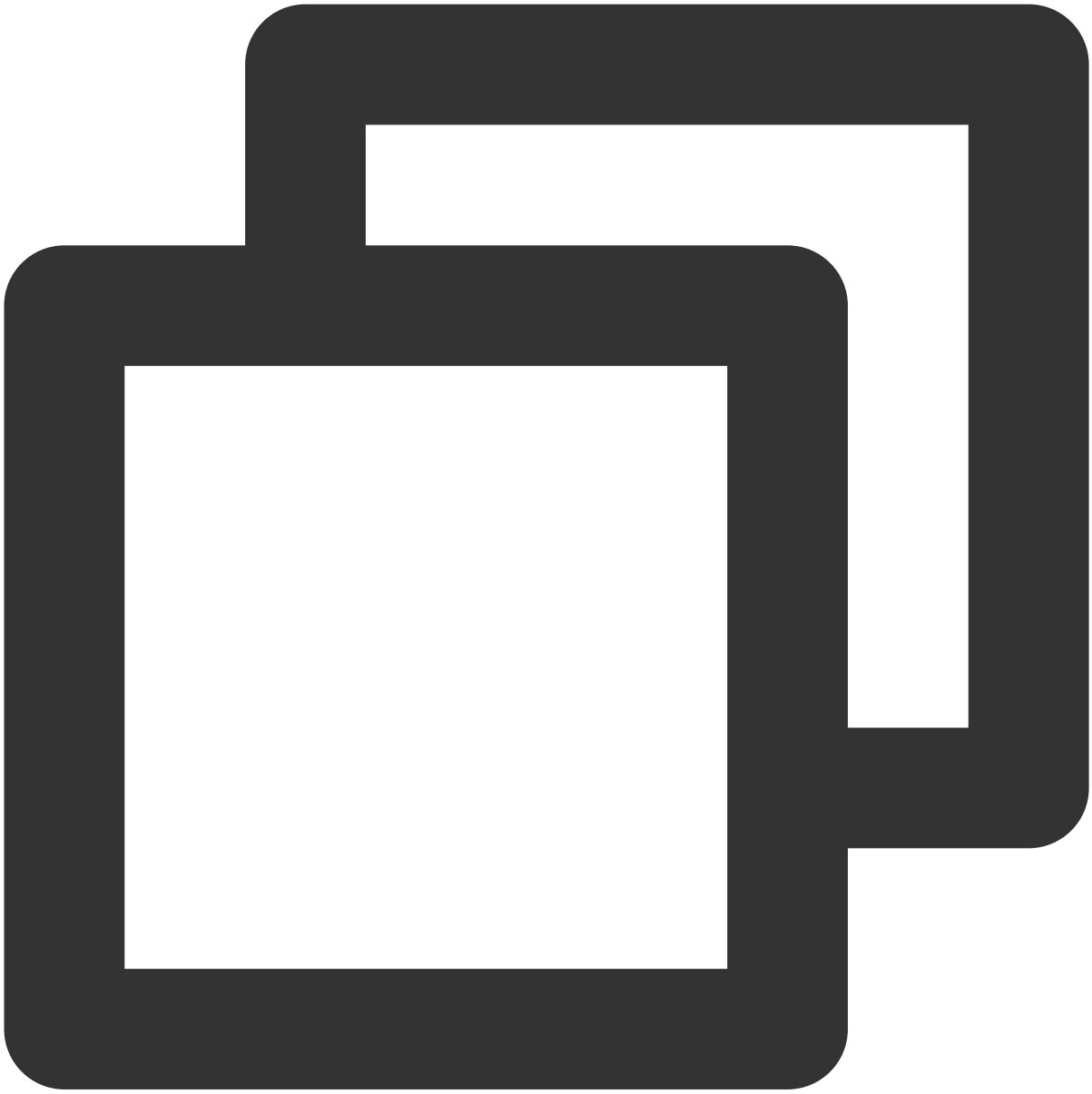


```
dependencies {
    api project(':tuicallkit-kt')
}
```

Note:

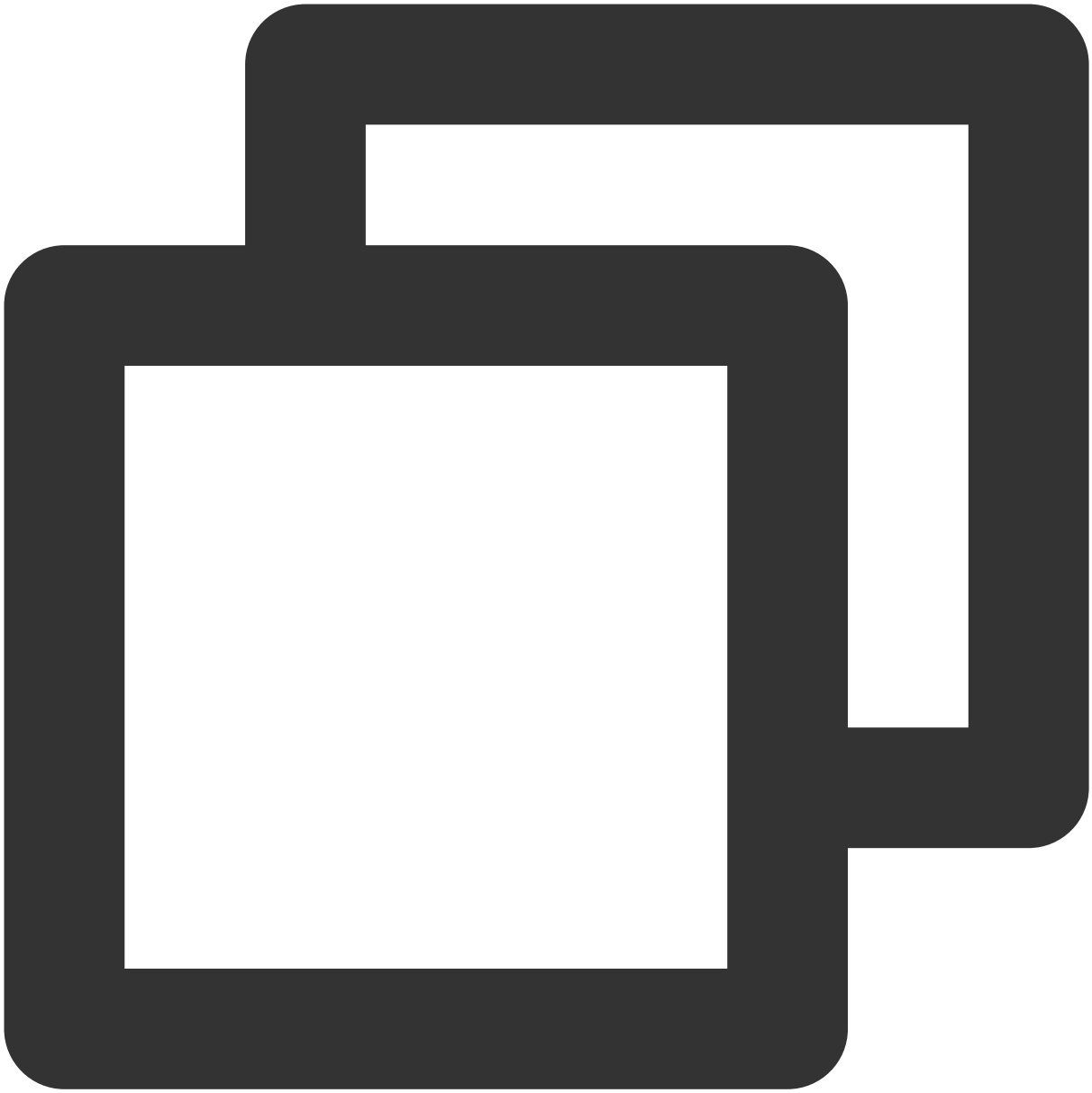
The `TUICallKit` project depends on `TRTC SDK`, `Chat SDK`, `tuicallengine`, and the `tuicore` public library internally by default. To upgrade the version, modify the version in `tuicallkit-kt/build.gradle` file.

3. Since the SDK uses Java's reflection feature internally, you need to add certain classes in the SDK to the obfuscation allowlist. Therefore, you should add the following code to the `proguard-rules.pro` file in the `app` directory. After adding, click on the upper right corner "**Sync Now**" to synchronize the code.



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

4. In the app directory, find the `AndroidManifest.xml` file and add `tools:replace="android:allowBackup"` within the application node to override the settings inside the components, using your own settings.



```
// app/src/main/AndroidManifest.xml

<application
  android:name=".DemoApplication"
  android:allowBackup="false"
  android:icon="@drawable/app_ic_launcher"
  android:label="@string/app_name"
  android:largeHeap="true"
  android:theme="@style/AppTheme"
  tools:replace="android:allowBackup">
```

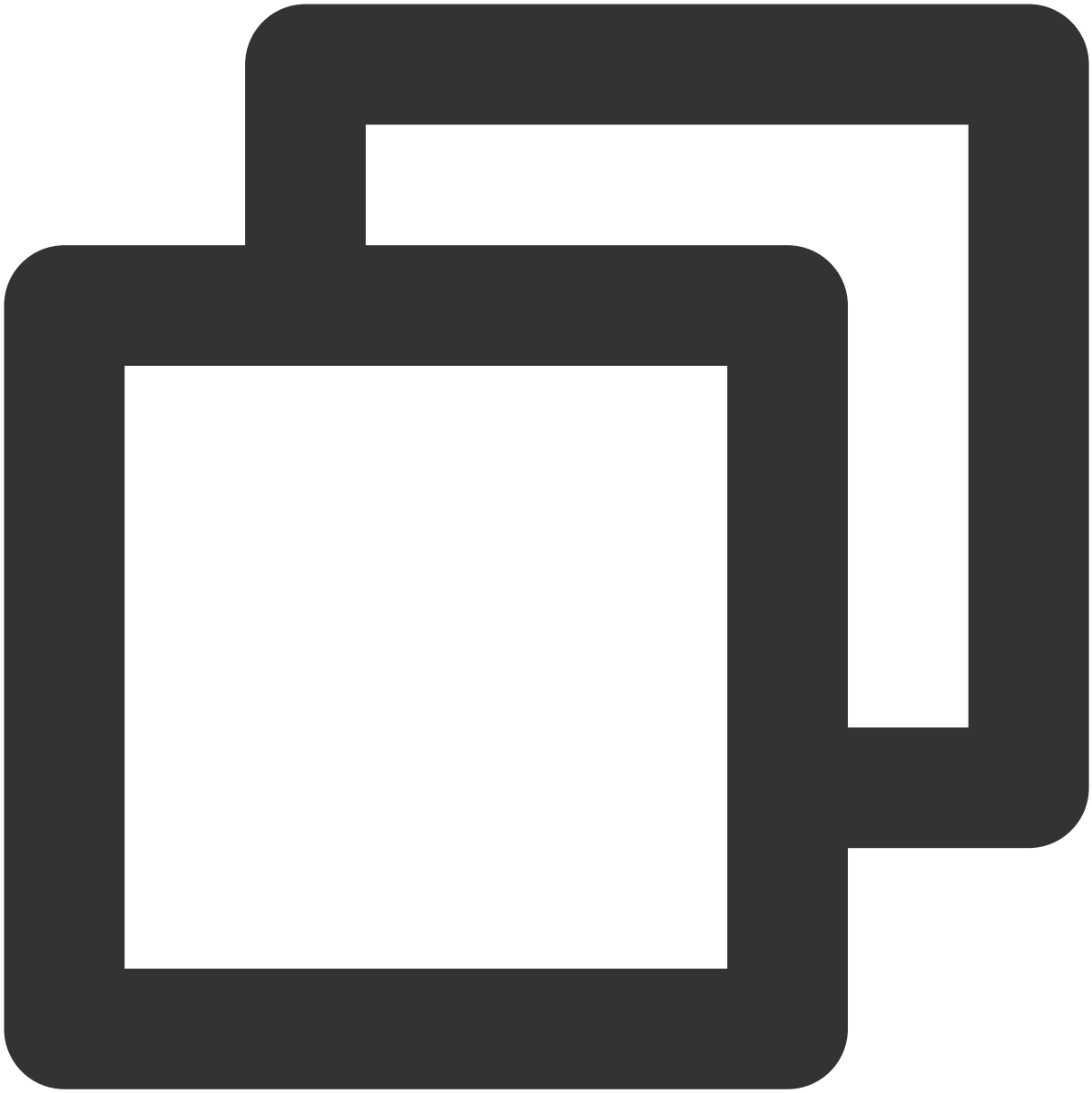
5. We suggest you compile and run once. If you encounter any issues, you may try our Github demo. By comparison, you can identify potential differences and resolve encountered issues. During integration and use, if any issues arise, you are welcome to provide feedback to us.

Step 4. Log in to the TUICallKit component

Add the following code to your project. It works by calling the relevant interfaces in TUICore to complete the login to TUI Component. This step is very important, only after successfully logging in, you can normally use the features offered by TUICallKit.

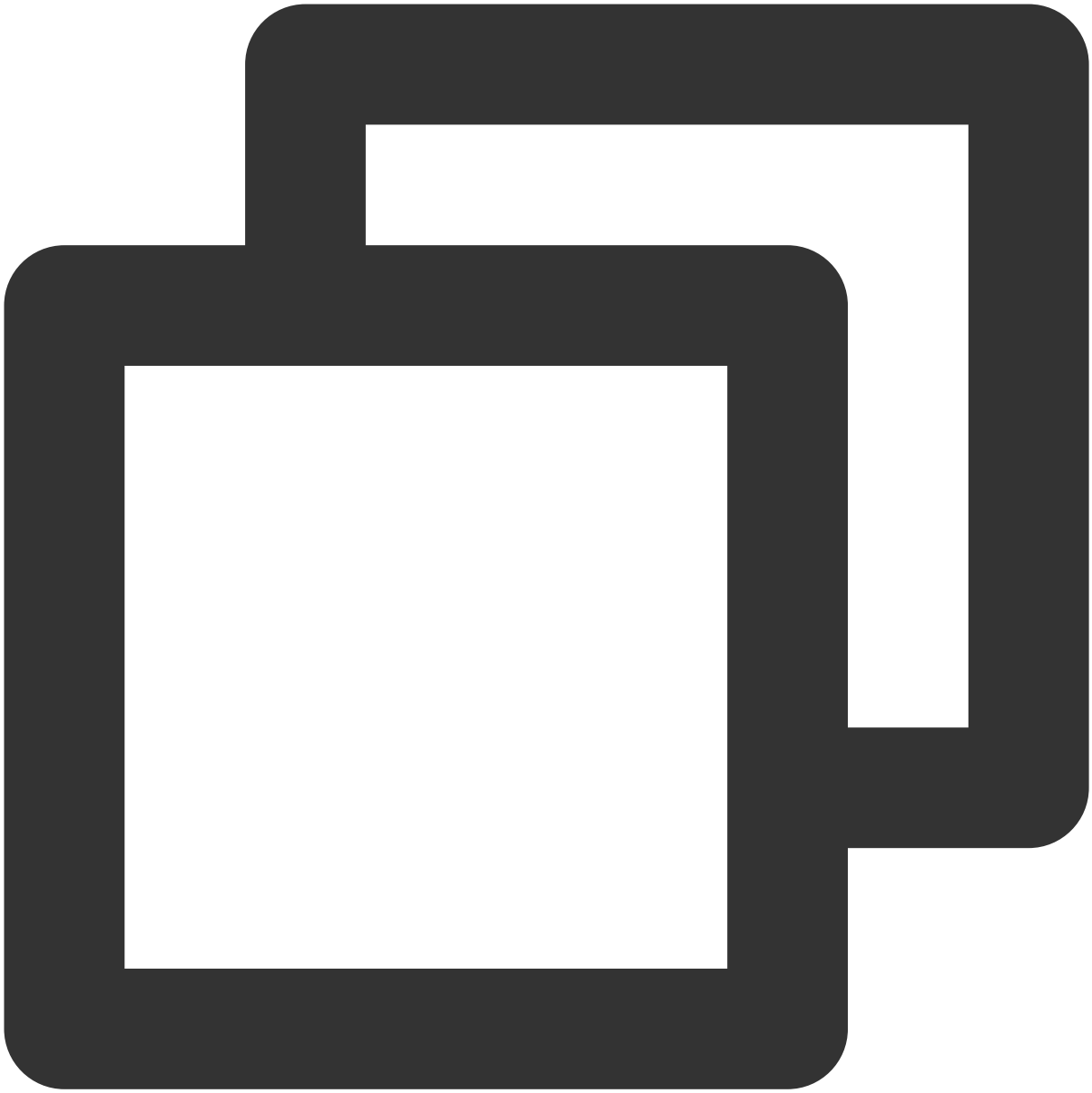
Kotlin

Java



```
import com.tencent.qcloud.tuicore.TUILogin
import com.tencent.qcloud.tuicore.interfaces.TUICallback
import com.tencent.qcloud.tuikit.tuicallkit.debug.GenerateTestUserSig

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // begin
        val userID = "denny"     // Please replace with your UserId
        val sdkAppID = 0         // Please replace with the SDKAppID obtained from
```

```
        val secretKey = "****"   // Please replace with the SecretKey obtained from

        val userSig = GenerateTestUserSig.genTestUserSig(userId, sdkAppId, secretKe

        TUILogin.login(this, sdkAppId, userId, userSig, object : TUICallback() {
            override fun onSuccess() {
            }
            override fun onError(errorCode: Int, errorMessage: String) {
            }
         })
        // end
    }
}
```



```
import com.tencent.qcloud.tuicore.TUILogin;
import com.tencent.qcloud.tuicore.interfaces.TUICallback;
import com.tencent.qcloud.tuikit.tuicallkit.debug.GenerateTestUserSig;

public class MainActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //begin
        String userID = "denny";     // Please replace with your UserId
```

```
        int sdkAppID = 0;            // Please replace with the SDKAppID obtained f
        String secretKey = "****";   // Please replace with the SecretKey obtained

        String userSig = GenerateTestUserSig.genTestUserSig(userId, sdkAppId, secre

        TUILogin.login(this, sdkAppId, userId, userSig, new TUICallback() {
            @Override
            public void onSuccess() {
            }

            @Override
            public void onError(int errorCode, String errorMessage) {
            }
        });
        //end
    }
}
```

Parameter	Type	Description
userID	String	your own User ID based on your business. It can only include letters (a-z, A-Z), digits (0-9), underscores, and hyphens.
sdkAppID	int	The unique identifier SDKAppID for the audio and video application created in Tencent RTC Console.
secretKey	String	SDKSecretKey for the audio and video application created in Tencent RTC Console.
userSig	String	A security signature for user login to verify identity and prevent unauthorized access to cloud services.

Note:

Development Environment: During local development and debugging, use the local `GenerateTestUserSig.genTestSig` function to create a userSig. But be careful, the SDKSecretKey can be decompiled and reverse-engineered. If leaked, it could allow theft of your Tencent Cloud traffic.

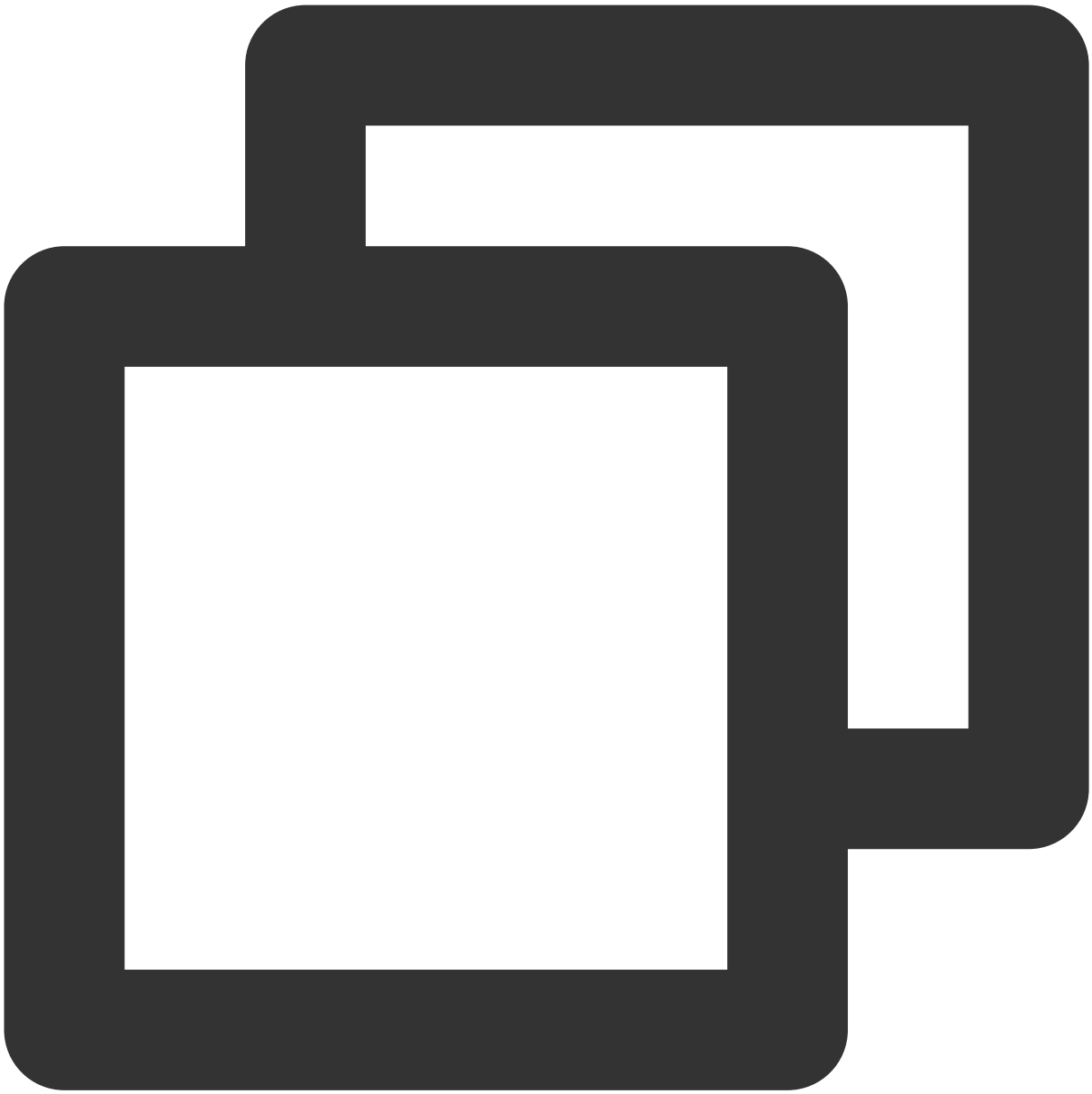
Production Environment: If your project is going to be launched, please adopt the method of Server-side Generation of UserSig.

Step 5. Make your first phone call

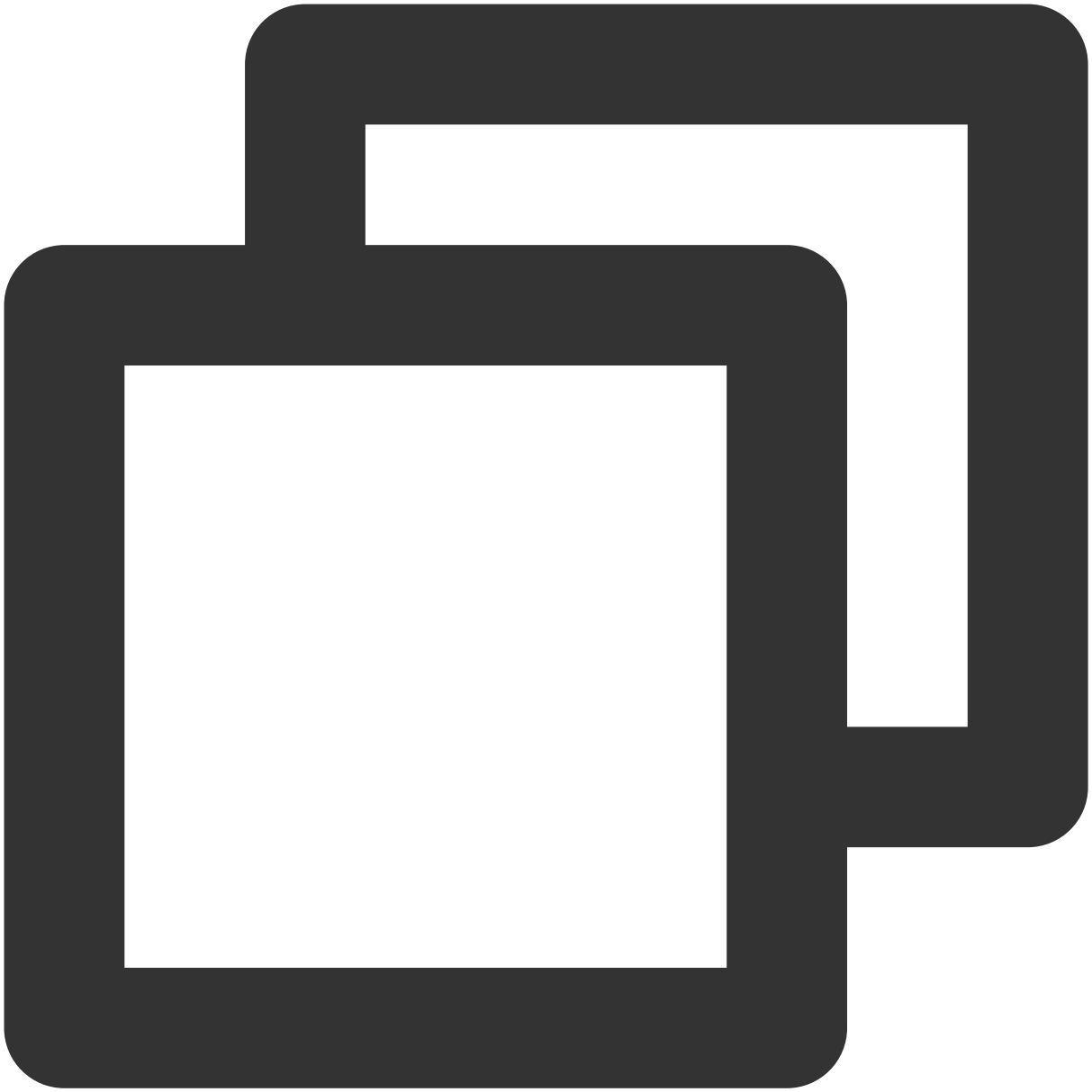
After user login success, invoke the call method of TUICallKit, specifying the callee's userID and the type of call, to initiate an audio/video call. The callee will receive an incoming call invitation.

Kotlin

Java

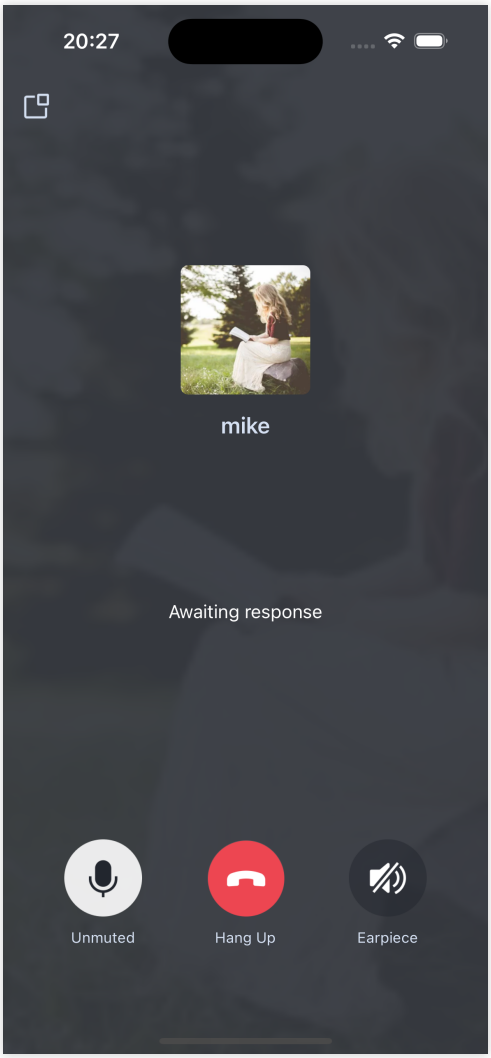


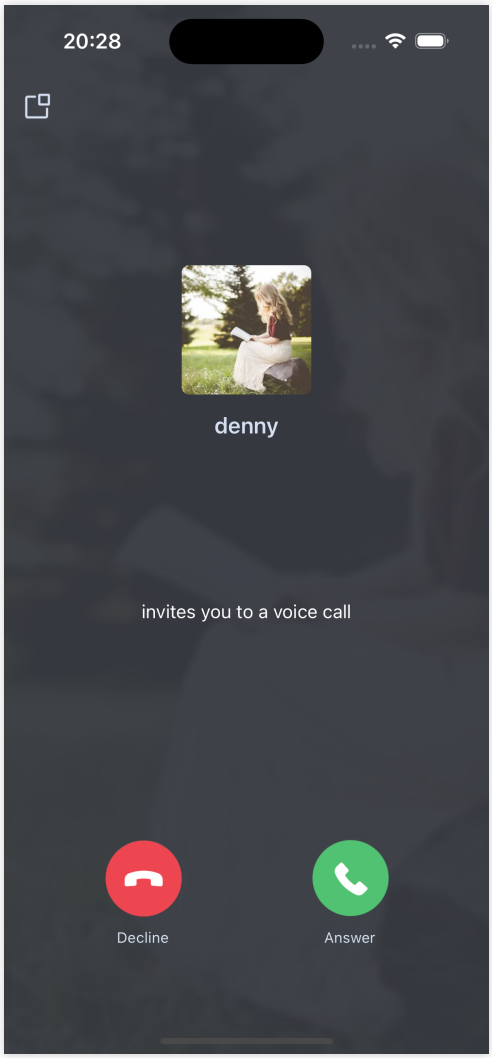
```
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

// Initiating a one-to-one audio call (assuming the callee's userID is mike)
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Audio)
```



```
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine;
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit;

// Initiating a one-to-one audio call (assuming the callee's userID is mike)
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Audio);
```

Caller initiates an audio call	Callee receives an audio call request





Additional Features

Setting Nickname, Avatar
Customize Interface
Offline Push
Group Call
Floating Window
Custom Ringtone
Call Status Monitoring
Cloud Recording

FAQs

If you encounter any issues during integration and use, please refer to Frequently Asked Questions.

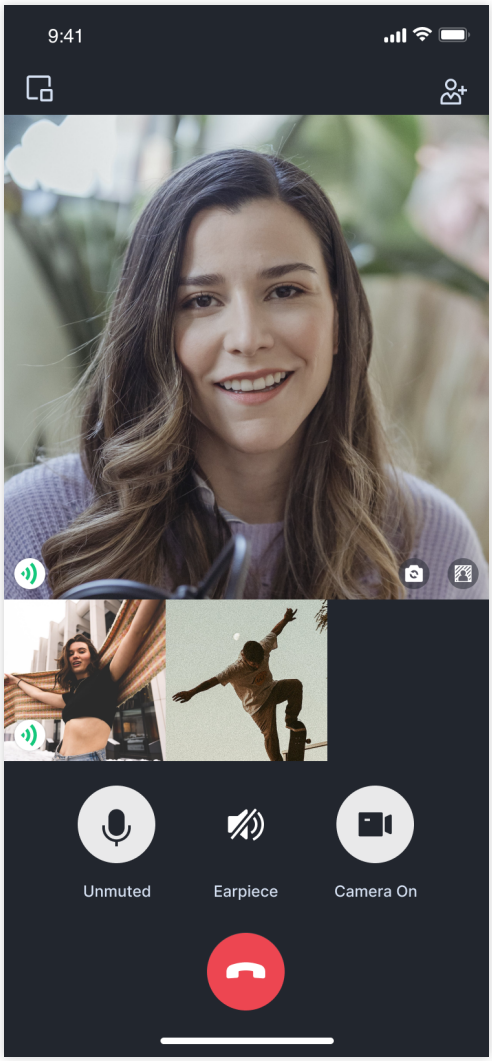
Suggestions and Feedback

If you have any suggestions or feedback, please contact `info_rtc@tencent.com` .

iOS

Last updated：2024-05-24 18:35:11

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

1v1 Video Call	Group call
	

Environment Preparations

Xcode 13 or later.

iOS 13.0 or later.

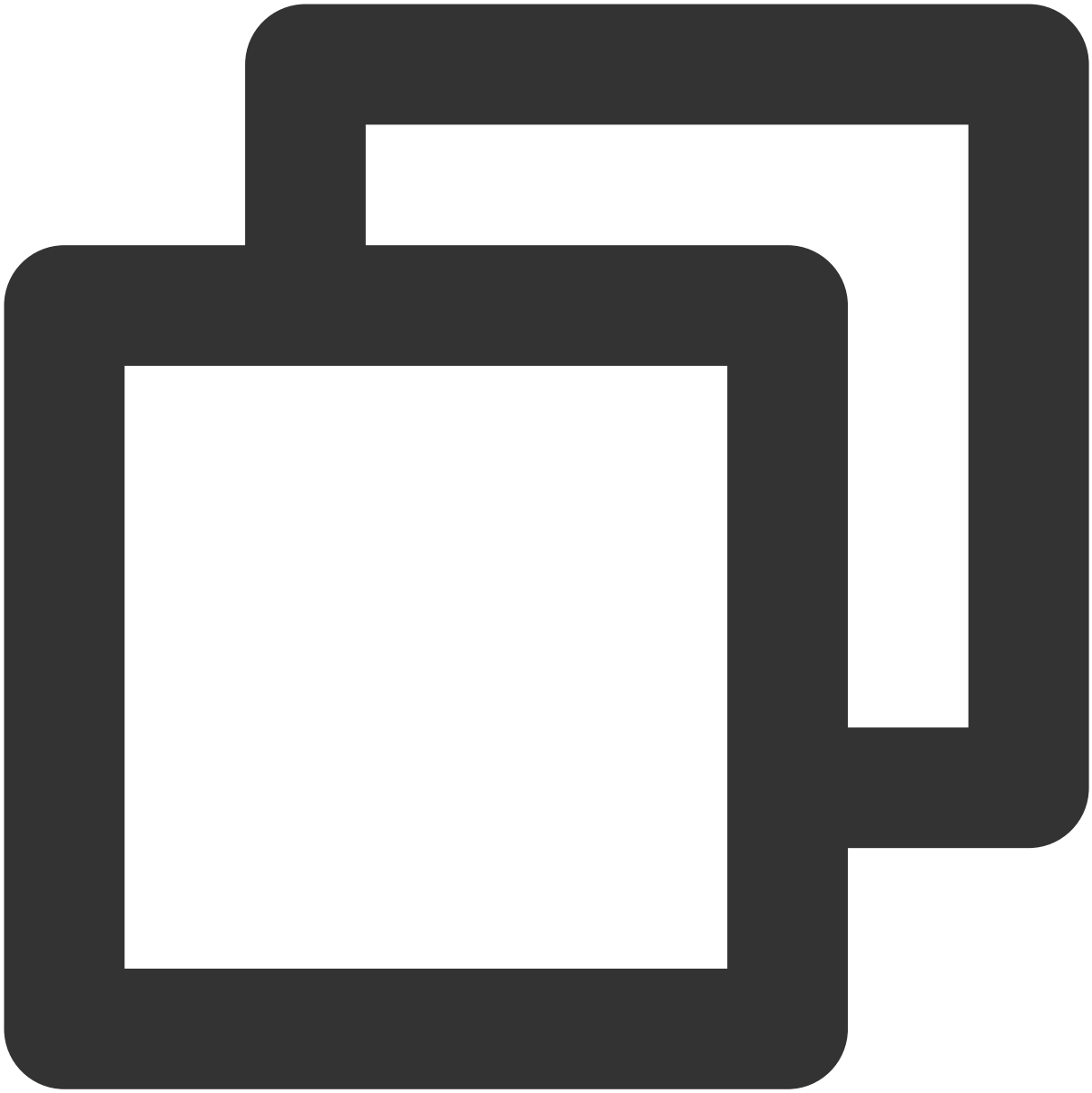
CocoaPods environment installation, Click to view.
If you experience any issues with access or usage, please consult the FAQs.

Step 1. Activate the service

Refer to Activate the Service to obtain `SDKAppID, SDKSecretKey` , which will be used as **Mandatory Parameters in** Step 4: Log in to the TUICallKit component**.**

Step 2. Import the component

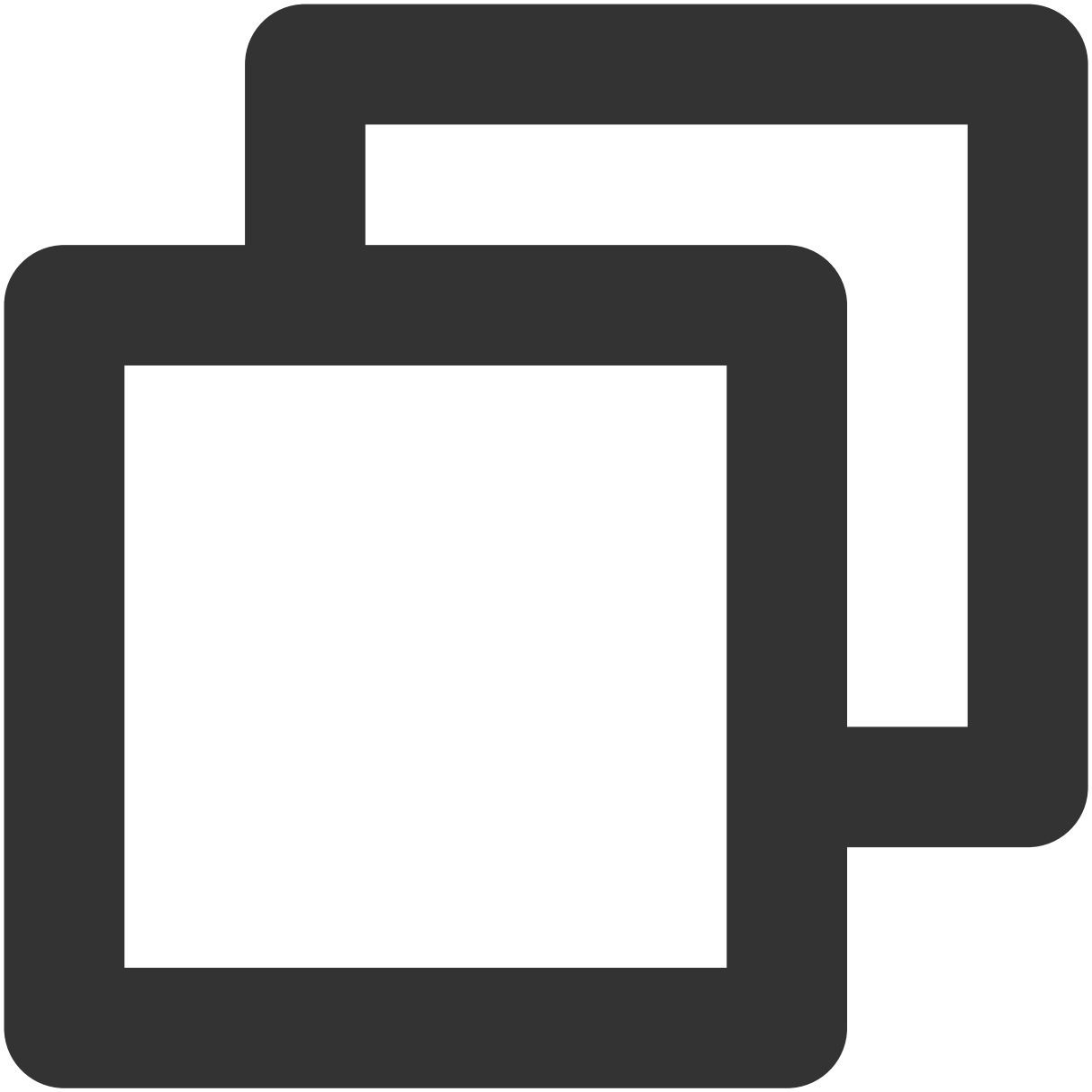
Use CocoaPods to import the component. If you encounter any issues, please refer to Environment Preparation first.
Detailed steps for importing the component are as follows:
1. Please add the dependency `pod 'TUICallKit_Swift'` to your `Podfile` . If you encounter any problems, please refer to the Example project.



```
target 'xxxx' do
  ...
  pod 'TUICallKit_Swift'
end
```

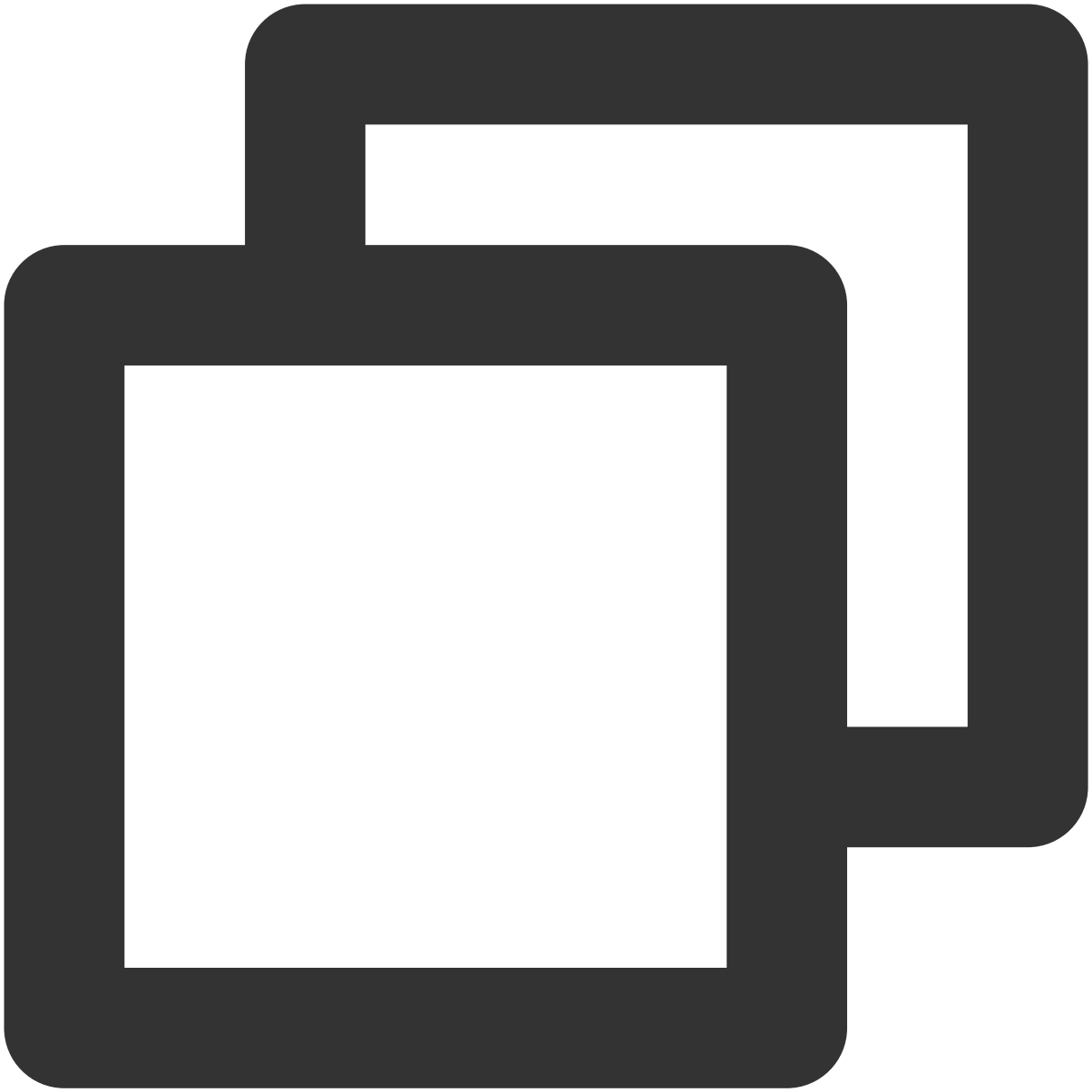
Note:

If your project lacks a `Podfile`, you need to `cd` into the `xxxx.xcodeproj` directory in Terminal, and then create a `Podfile` by executing the following command:



```
pod init
```

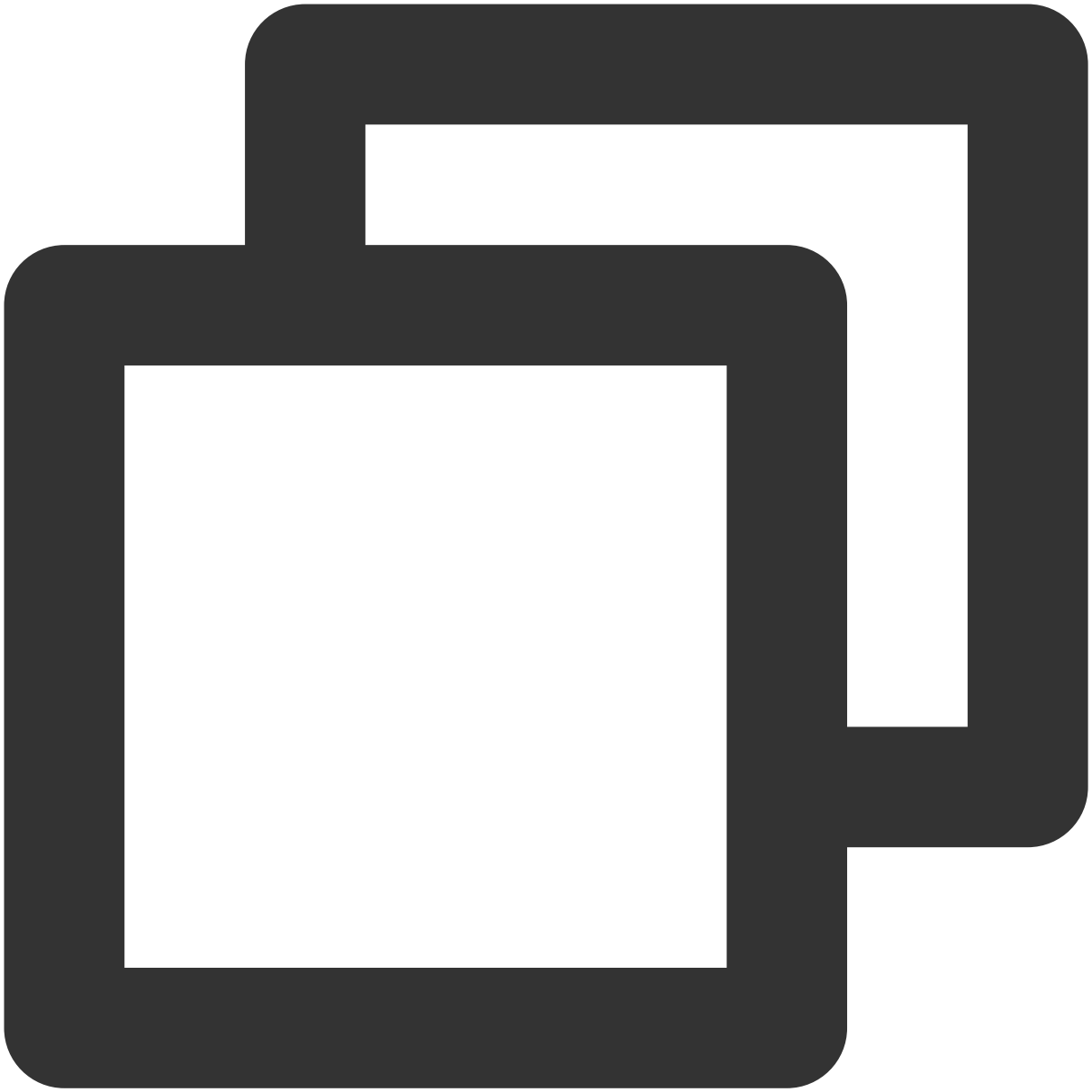
2. In Terminal, first `cd` into the `Podfile` directory and then run the following command to install components.



```
pod install
```

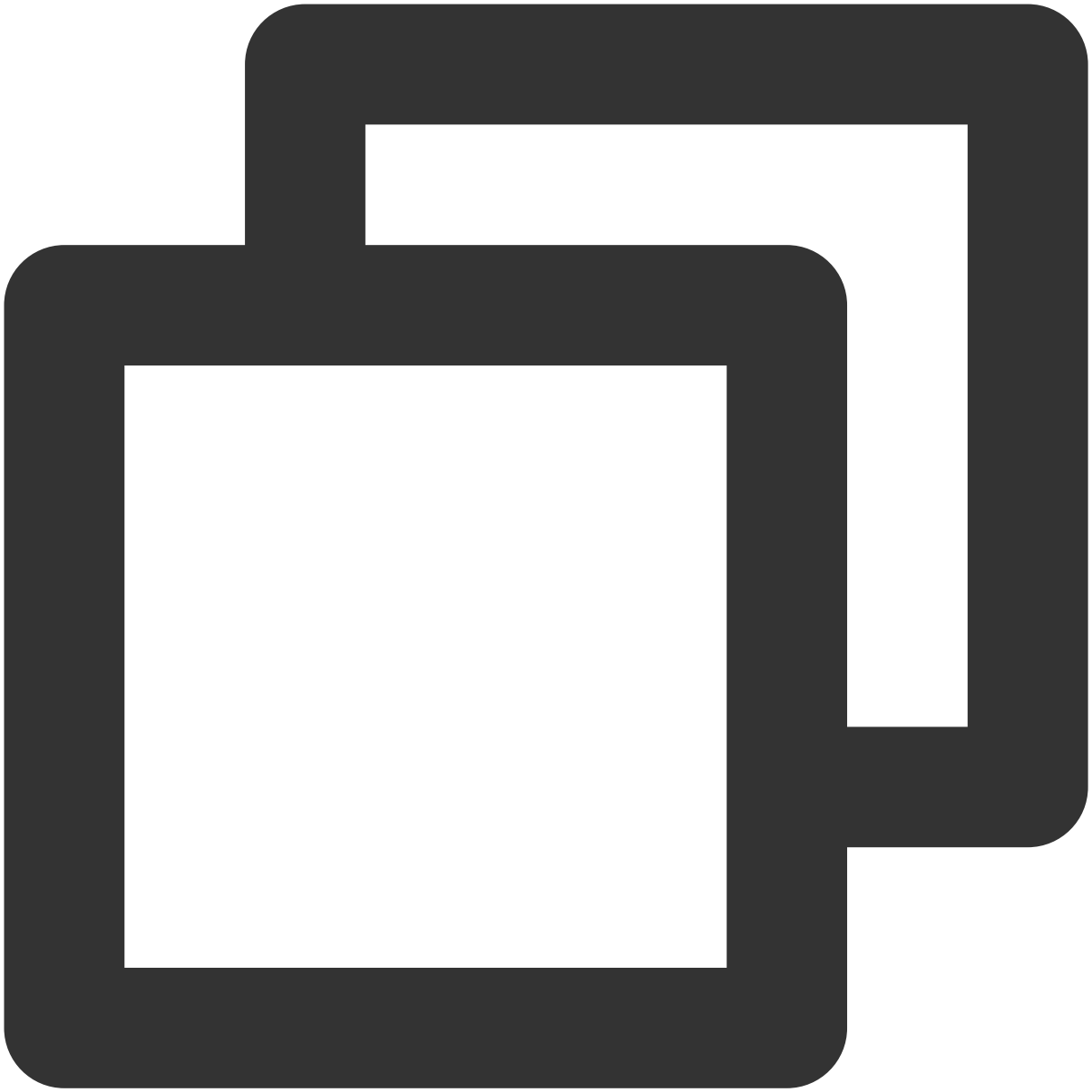
Note:

If you cannot install the latest version of TUICallKit, you may first delete **Podfile.lock** and **Pods**. Then run the following command to update the local CocoaPods repository list.



```
pod repo update
```

Then, run the following command to update the Pod version of the component library.



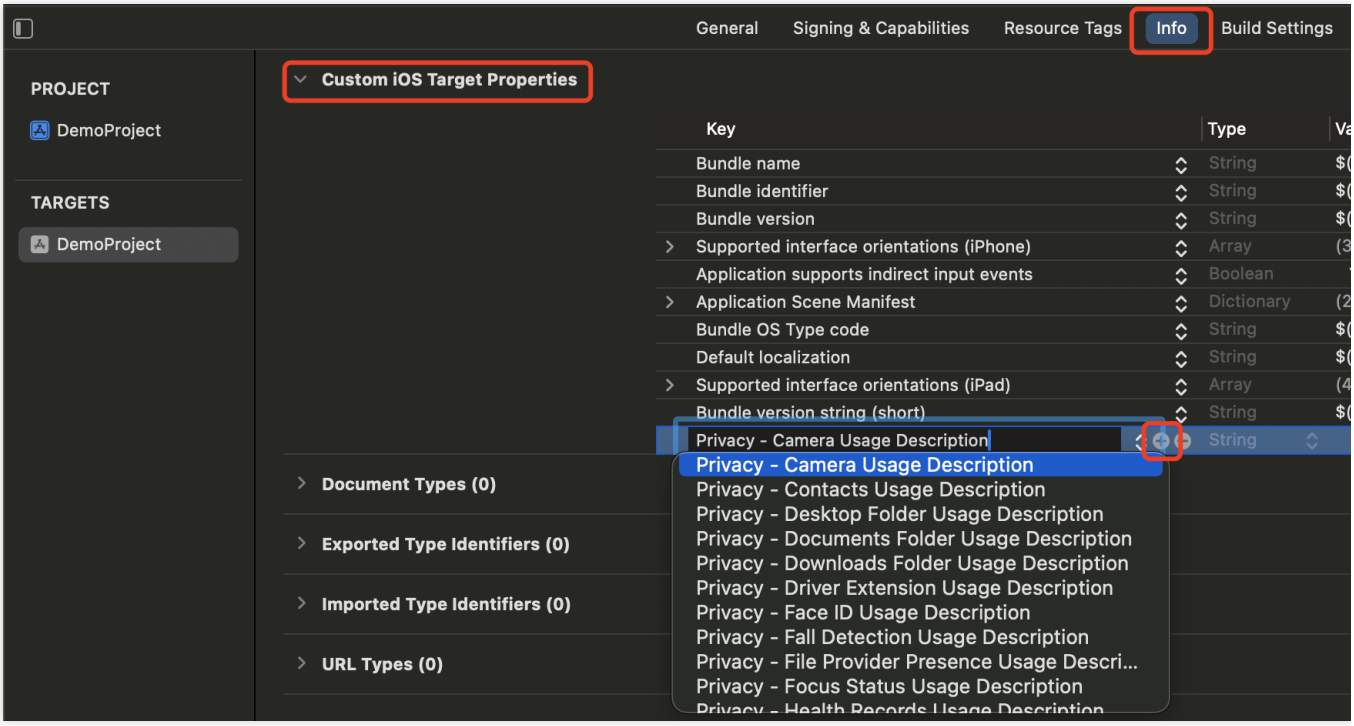
```
pod update
```

3. We suggest you compile and run once. If you encounter any problems, you can refer to our FAQs. If the problem remains unresolved, you may try running our Example project. If you encounter any issues during integration and use, you are welcome to provide feedback to us.

Step 3: Configure the Project

To use audio and video features, you need to authorize the usage of the camera and microphone. Please set the required permissions according to the actual needs of the project.

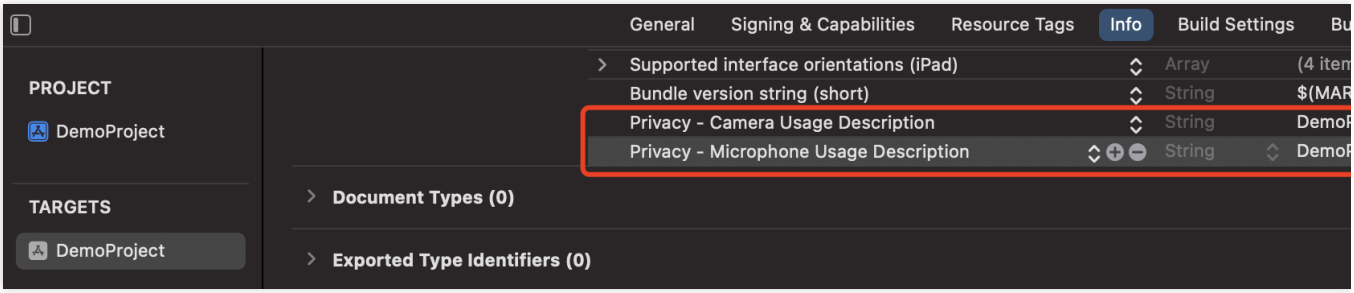
1. In **Xcode**, select **TARGETS** > **Info** > **Custom iOS Target Properties**" from the menu.



2. Click the **+** button to add camera and microphone permissions.

`Privacy - Camera Usage Description`

`Privacy - Microphone Usage Description`



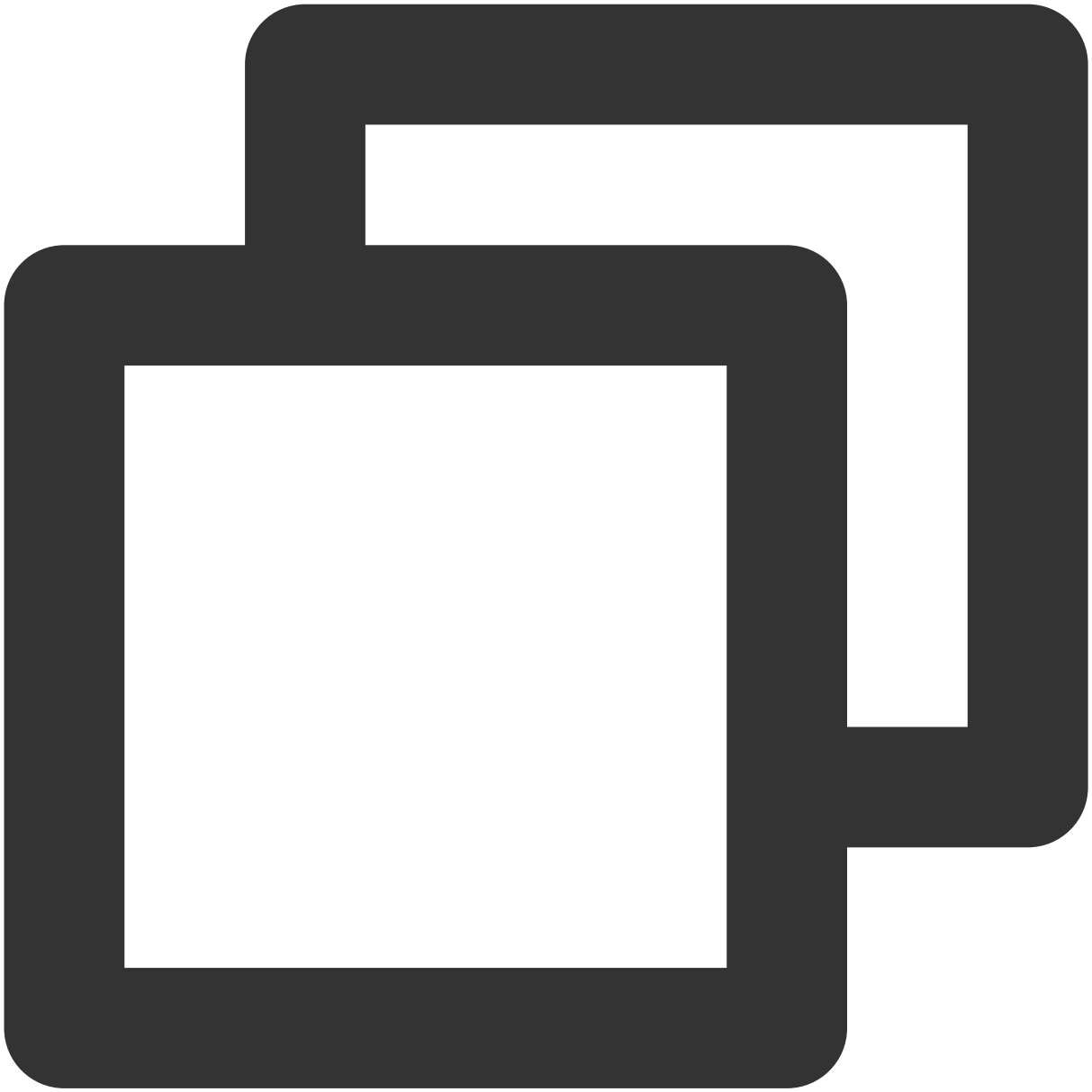
Step 4: Log in to the `TUICallKit` component

Add the following code to your project. It works by calling the relevant interfaces in TUICore to complete the login to TUI Component. This step is very important, only after successfully logging in, you can normally use the features

offered by TUICallKit.

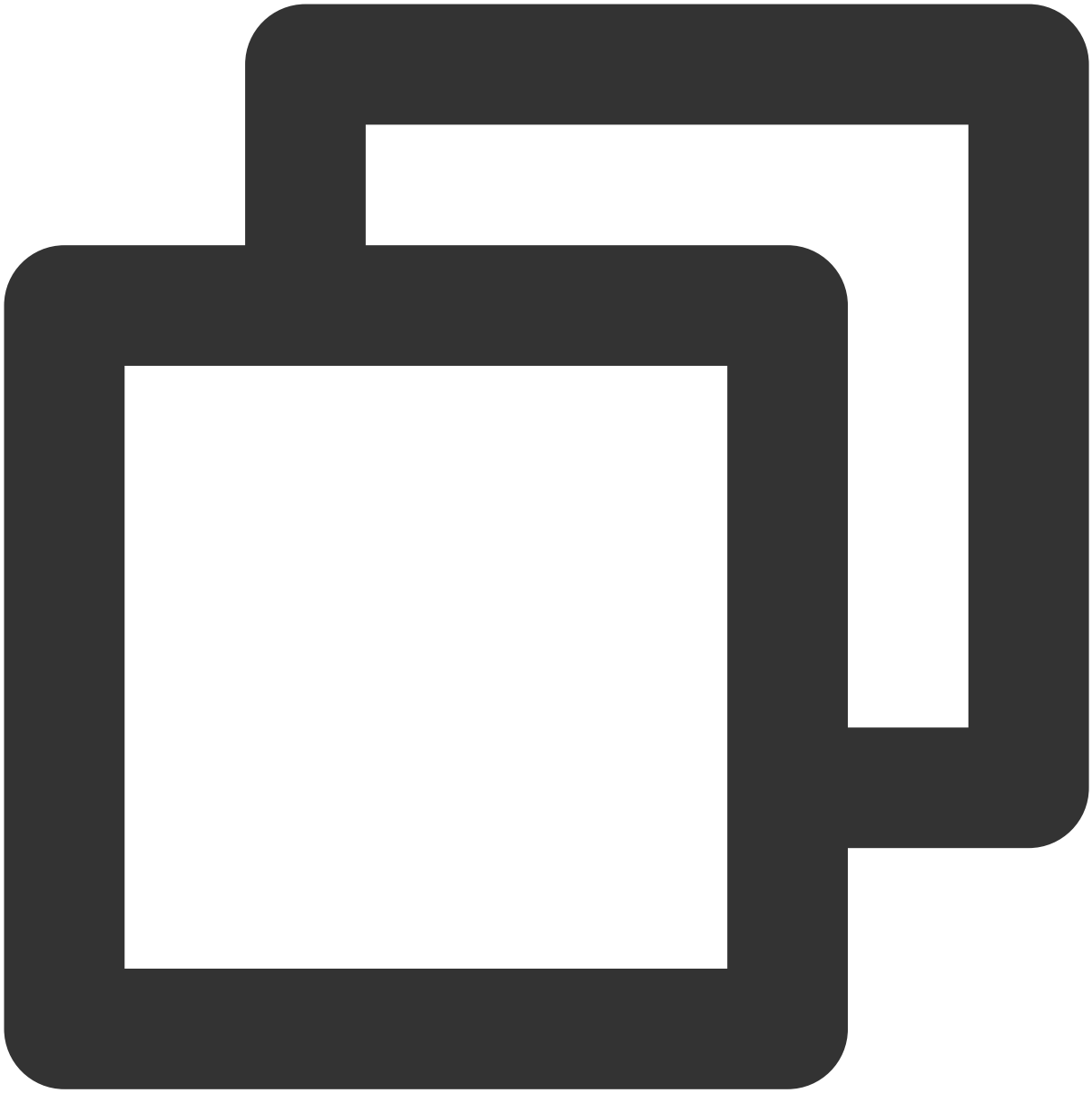
Swift

Objective-C



```
import TUICore
import TUICallKit_Swift

func application(_ application: UIApplication, didFinishLaunchingWithOptions launch

    let userID = "denny"       // Please replace with your UserId
    let sdkAppID: Int32 = 0    // Please replace with the SDKAppID obtained from th
```

```
    let secretKey = "****"     // Please replace with the SecretKey obtained from t

    let userSig = GenerateTestUserSig.genTestUserSig(userID: userID, sdkAppID: sdkA

    TUILogin.login(sdkAppID, userID: userID, userSig: userSig) {
      print("login success")
    } fail: { code, message in
      print("login failed, code: \\(code), error: \\(message ?? "nil")")
    }

    return true
}
```



```
#import <TUICore/TUILogin.h>
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD

    NSString *userID = @"denny";      // Please replace with your UserId
    int sdkAppID = 0;                 // Please replace with the SDKAppID obtained f
    NSString *secretKey = @"****";    // Please replace with the SecretKey obtained

    NSString *userSig = [GenerateTestUserSig genTestUserSigWithUserID:userID sdkApp
```

```
    [TUILogin login:sdkAppID
             userID:userID
            userSig:userSig
               succ:^{
        NSLog(@"login success");
    } fail:^(int code, NSString * _Nullable msg) {
        NSLog(@"login failed, code: %d, error: %@", code, msg);
    }];

    return YES;
}
```

Parameter	Type	Description
userID	String	Your own User ID based on your business. It can only include letters (a-z, A-Z), digits (0-9), underscores, and hyphens.
sdkAppID	Int32	The unique identifier SDKAppID for the audio and video application created in Tencent RTC Console.
secretKey	String	SDKSecretKey for the audio and video application created in Tencent RTC Console.
userSig	String	A security signature for user login to verify identity and prevent unauthorized access to cloud services.

Note:

Development Environment: During local development and debugging, use the local GenerateTestUserSig.genTestSig function to create a userSig. But be careful, the SDKSecretKey can be decompiled and reverse-engineered. If leaked, it could allow theft of your Tencent Cloud traffic.

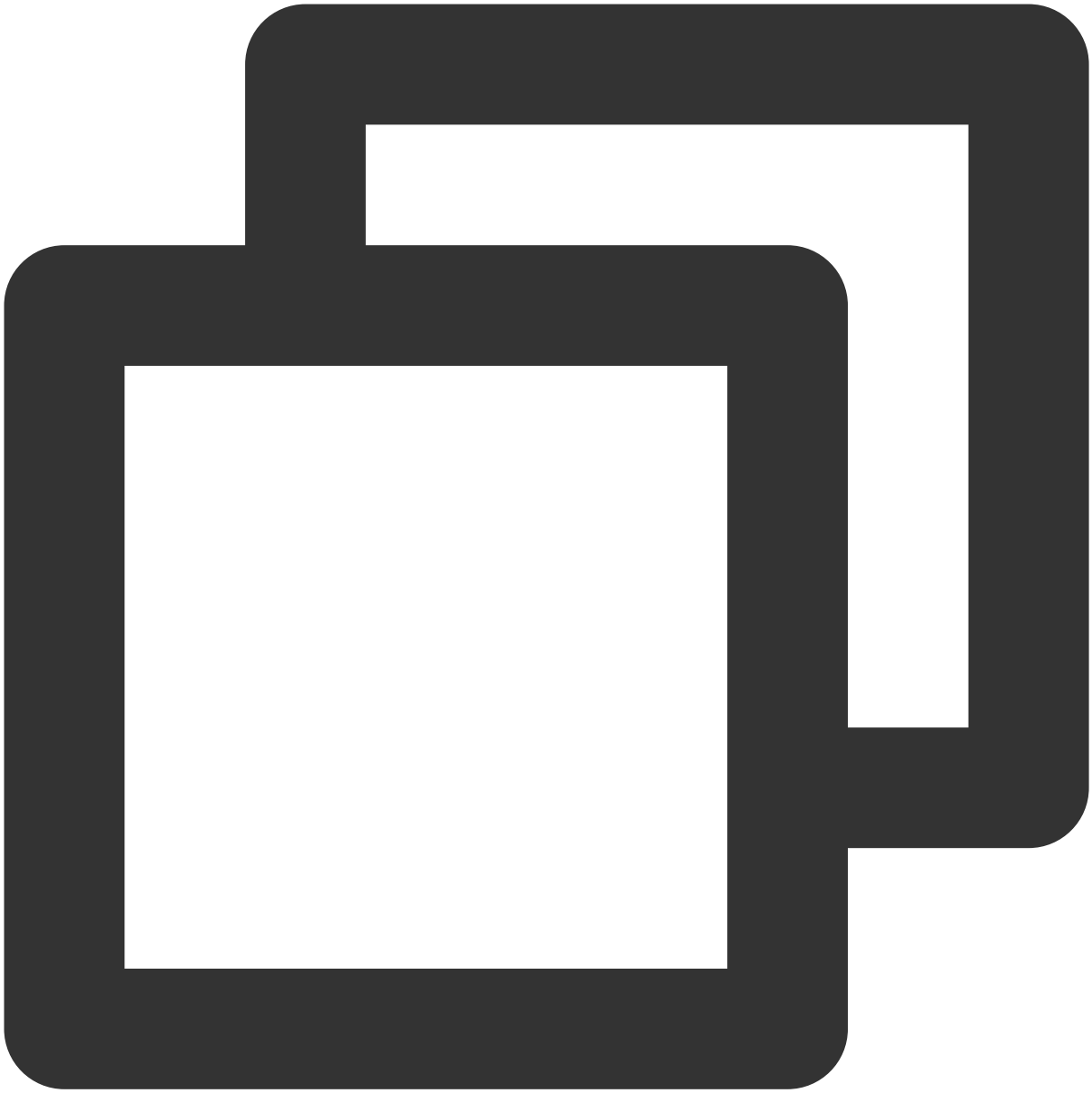
Production Environment: For project launch, use the Server-side Generation of UserSig method.

Step Five: Make your first call

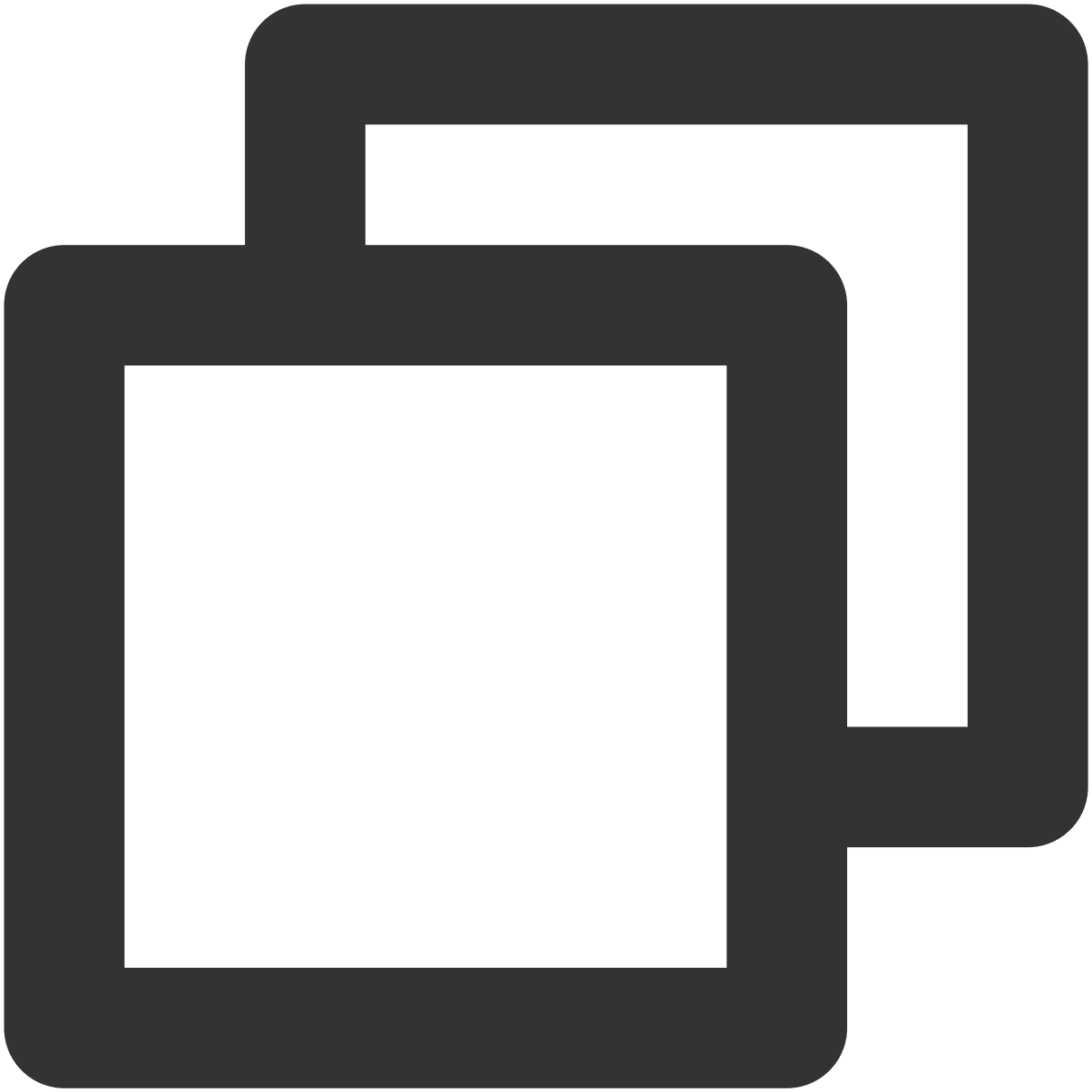
By calling the call function of TUICallKit and specifying the call type and callee's userId, you can initiate an audio or video call.

Swift

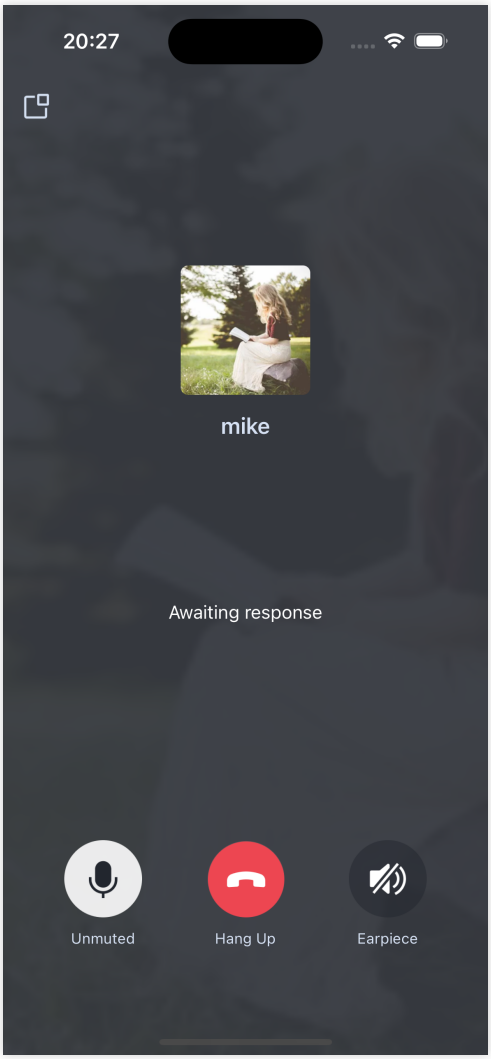
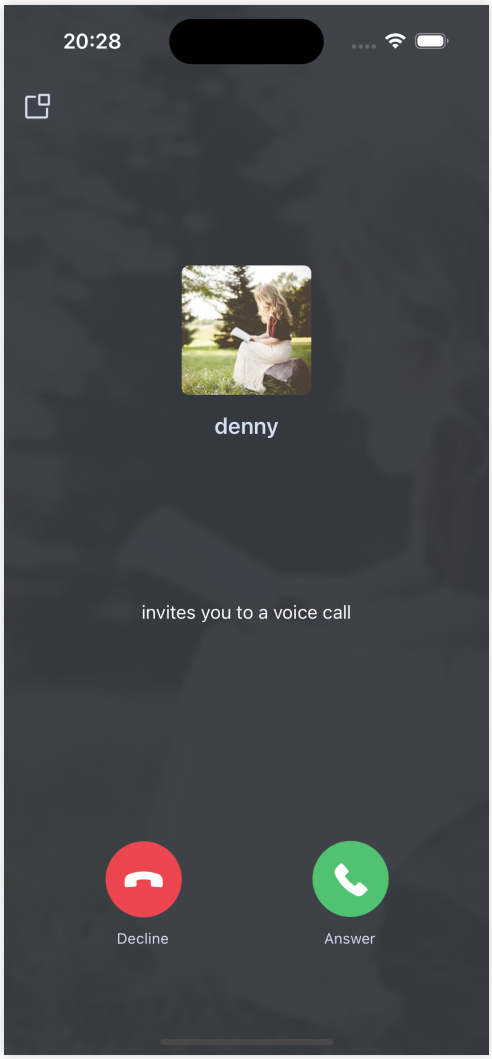
Objective-C



```
import TUICallKit_Swift

// Initiating a 1-to-1 audio call (assuming userId is mike)
TUICallKit.createInstance().call(userId: "mike", callMediaType: .audio)
```



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

// Initiating a 1-to-1 audio call (assuming userId is mike)
[[TUICallKit createInstance] callWithUserId:@"mike" callMediaType:TUICallMediaTypeA
```

	
Caller	Callee

Additional Features

Setting Nickname, Avatar

Customize Interface

Offline Push

Group Call

Floating Window

Custom Ringtone

Call Status Monitoring

Cloud Recording

FAQs

If you encounter any issues during integration and use, please refer to Frequently Asked Questions.

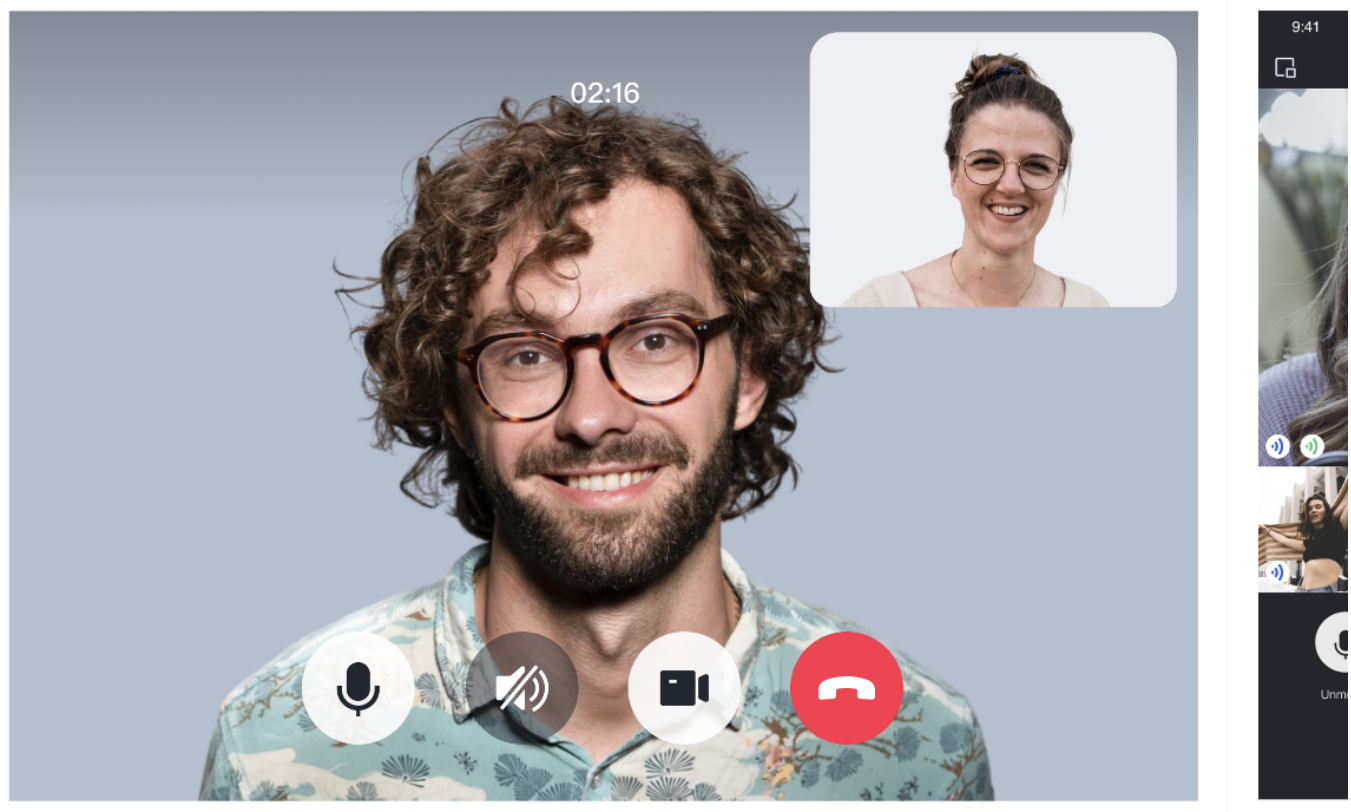
Suggestions and Feedback

If you have any suggestions or feedback, please contact info_rtc@tencent.com.

Web&H5 (React)

Last updated：2024-07-22 17:21:41

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



Prerequisites

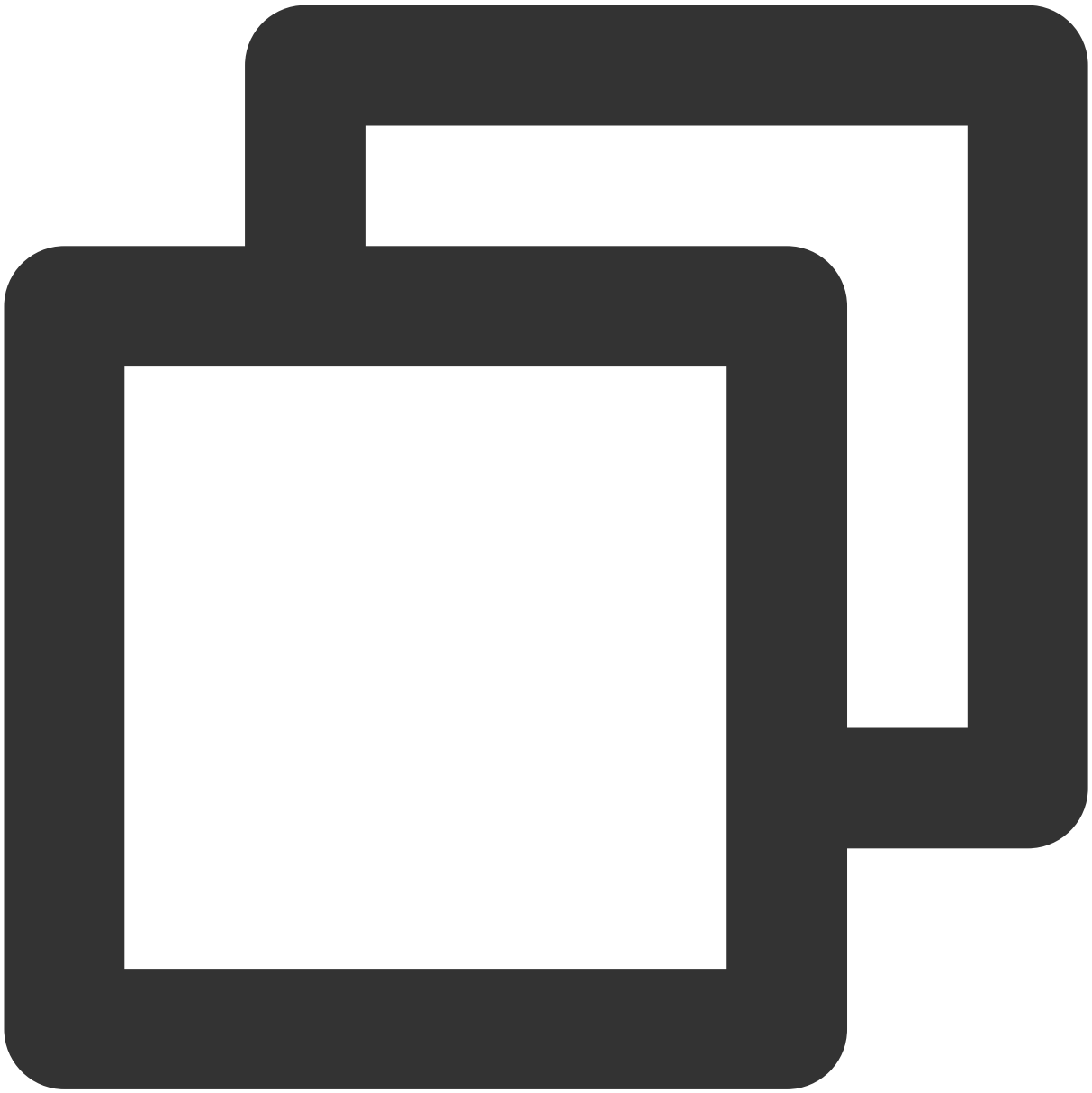
React version 18+.
Node.js version 16+.
Modern browser, supporting WebRTC APIs.

Step 1. Activate the service

Refer to Activate the Service to obtain `SDKAppID, SDKSecretKey` , which will be used as **Mandatory Parameters in** Initialize the TUICallKit**.**

Step 2. Download the TUICallKit

1. Download the @tencentcloud/call-uikit-react component.

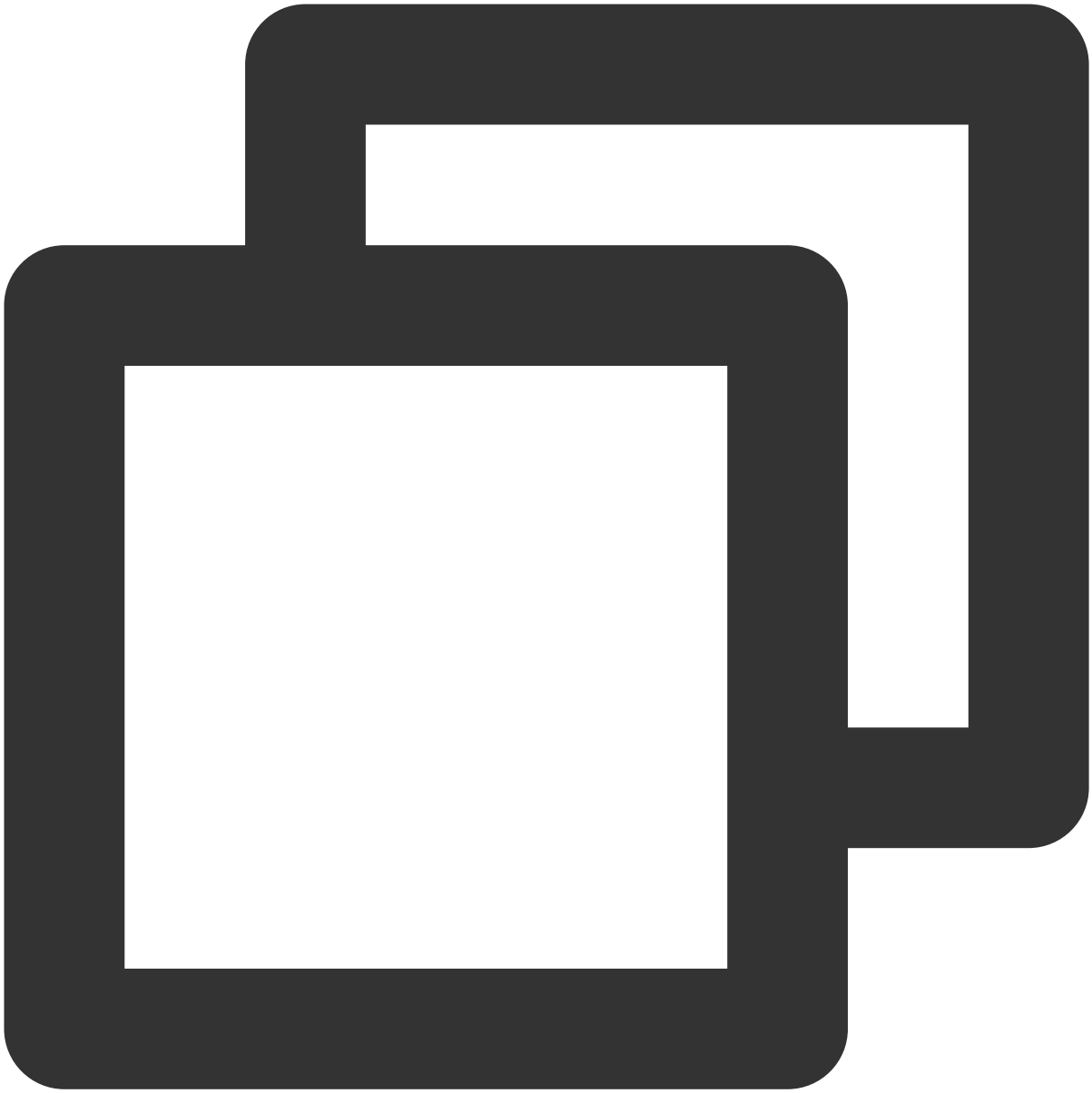


```
npm install @tencentcloud/call-uikit-react
```

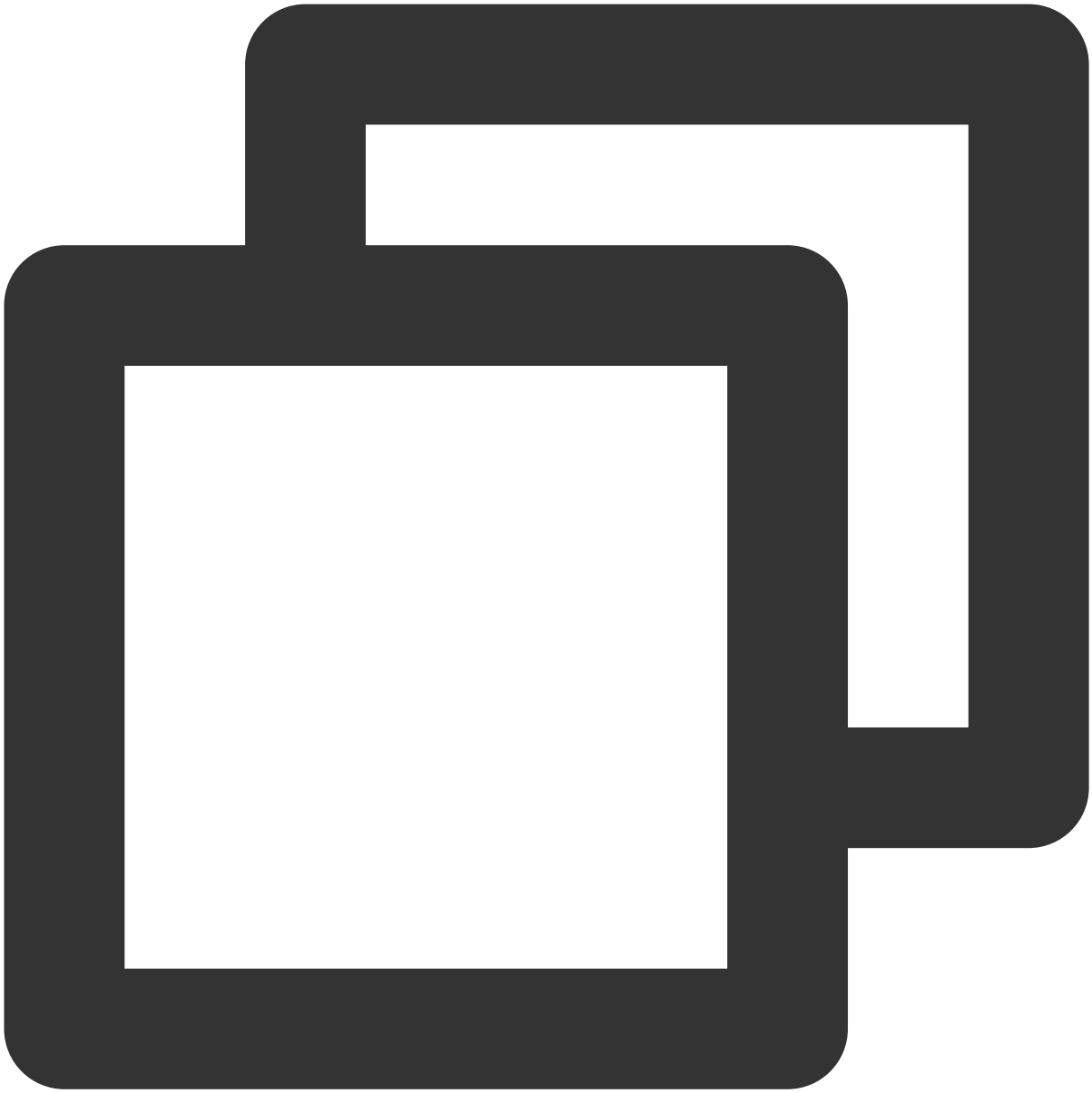
2. Copy the `debug` directory to your project directory `src/debug` , it is necessary when generating userSig locally.

MacOS

Windows



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

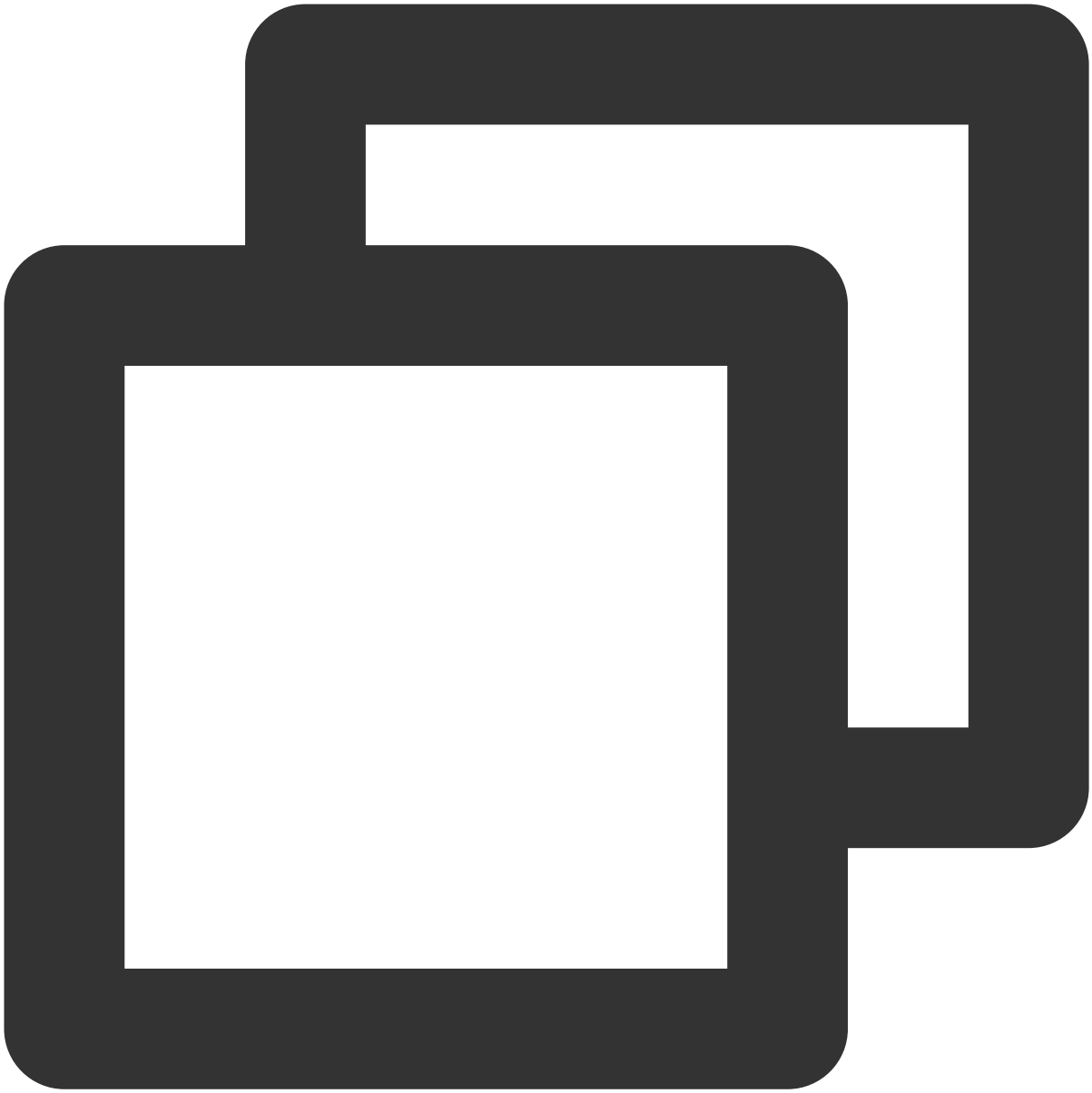


```
xcopy node_modules\\@tencentcloud\\call-uikit-react\\debug .\\src\\debug /i /e
```

Step 3. Initialize the TUICallKit

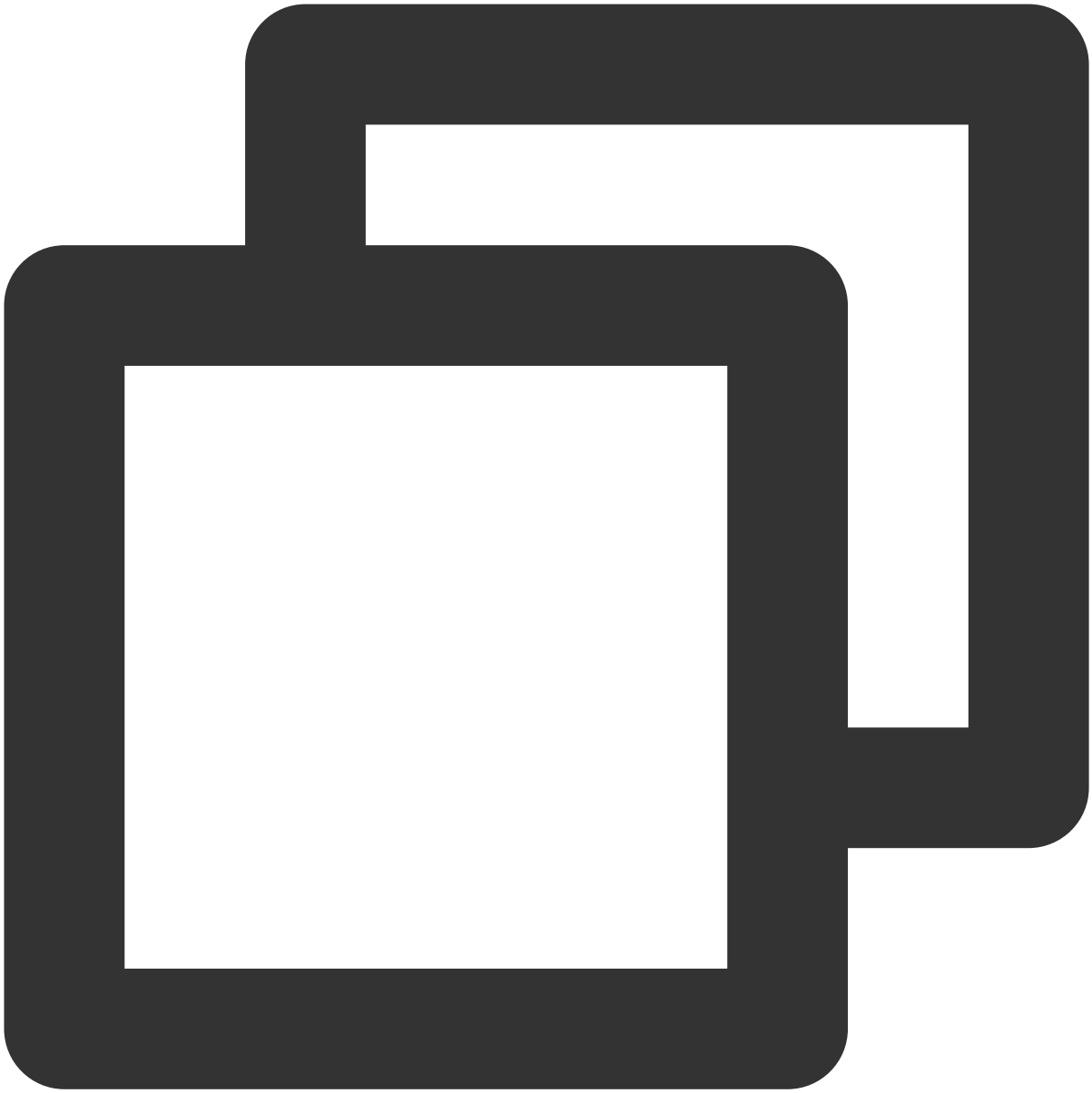
You can choose to import the sample code in the `/src/App.tsx` file.

1. Import the call uikit.



```
import { useState } from 'react';
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es"; // Refer to
```

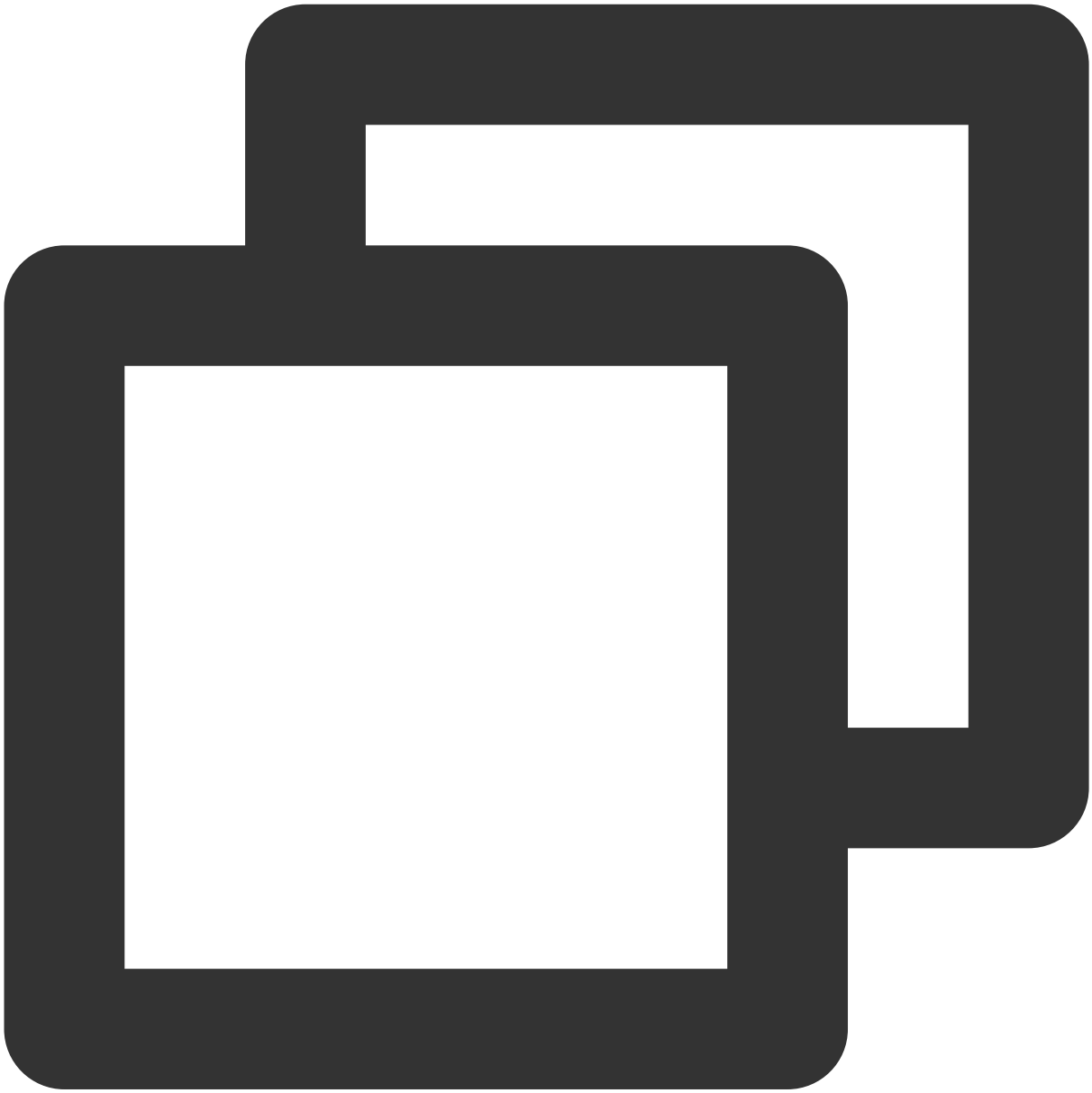
2. using the <TUICallKit />, which contains the complete UI interaction during a call.



```
return (
  <>
    <span> caller's ID: </span>
    <input type="text" placeholder='input caller userID' onChange={(event) => setCa
    <button onClick={init}> step1. init </button> <br />
    <span> callee's ID: </span>
    <input type="text" placeholder='input callee userID' onChange={(event) => setCa
    <button onClick={call}> step2. call </button>

    {/* 【1】Import the TUICallKit component: Call interface UI */}
    <TUICallKit />
```

```
    </>
  );
```

3. using the TUICallKitServer.init API to log in to the component, you need to `fill in SDKAppID, SDKSecretKey` as two parameters in the code.



```
const SDKAppID = 0;        // TODO: Replace with your SDKAppID (Notice: SDKAppID is
const SDKSecretKey = '';   // TODO: Replace with your SDKSecretKey

const [callerUserID, setCallerUserID] = useState('');
const [calleeUserID, setCalleeUserID] = useState('');
```

```
// 【2】Initialize the TUICallKit component
const init = async () => {
  const { userSig } = GenerateTestUserSig.genTestUserSig({
    userID: callerUserID,
    SDKAppID,
    SecretKey: SDKSecretKey,
  });
  await TUICallKitServer.init({
    userID: callerUserID,
    userSig,
    SDKAppID,
  });
  alert('TUICallKit init succeed');
}
```

Parameter	Type	Note
userID	String	**Unique identifier of the user,** `defined by you` **,** it is allowed to contain only upper and lower case letters (a-z, A-Z), numbers (0-9), underscores, and hyphens.
SDKAppID	Number	The unique identifier for the audio and video application created in the Tencent RTC Console.
SDKSecretKey	String	The SDKSecretKey of the audio and video application created in the Tencent RTC Console.
userSig	String	A security protection signature used for user log in authentication to confirm the user's identity and prevent malicious attackers from stealing your cloud service usage rights.

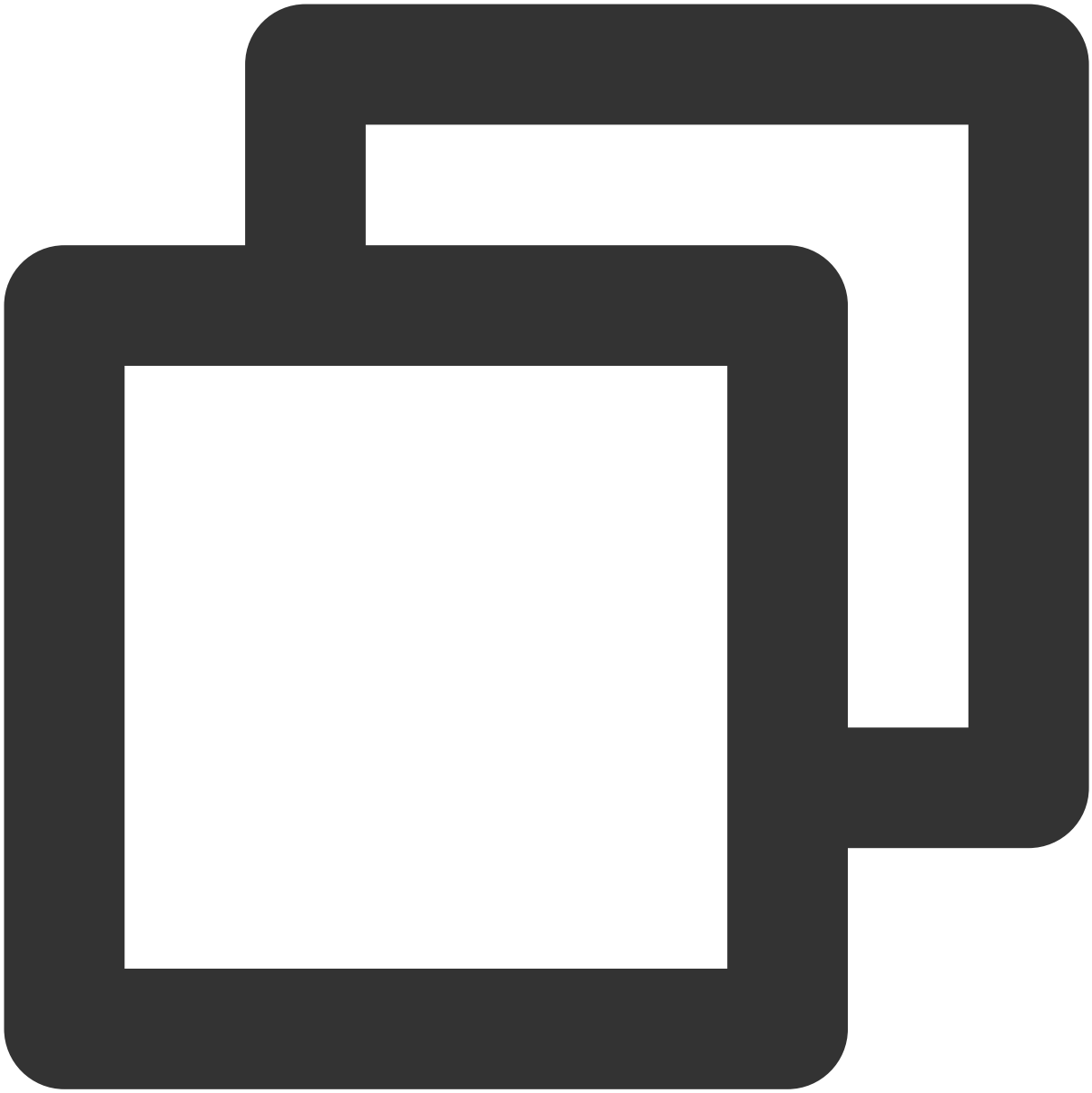
Explanation of userSig:

Development environment: If you are running a demo locally and developing debugging, you can use the `genTestUserSig` (Refer to Step 3.2) function in the debug file to generate a `userSig`. In this method, SDKSecretKey is vulnerable to decompilation and reverse engineering. Once your key is leaked, attackers can steal your Tencent Cloud traffic.

Production environment: If your project is going live, please use the Server-side Generation of UserSig method.

Step 4. Make your first call

1. using the TUICallKitServer.call API to make a call.



```
// 【3】Make a 1v1 video call
const call = async () => {
  await TUICallKitServer.call({
    userID: calleeUserID,
    type: TUICallType.VIDEO_CALL,
  });
};
```

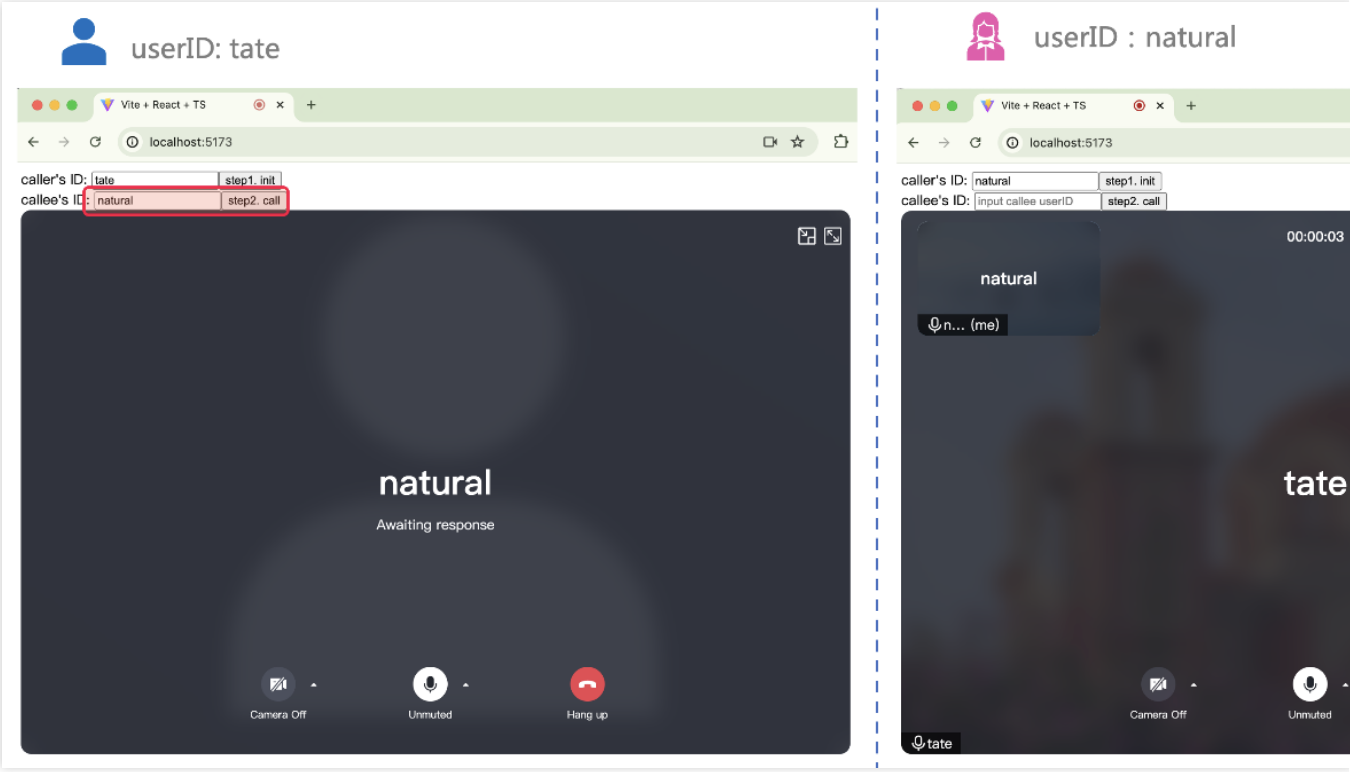
2. Run the project.

Warning:

For local environment, please access under localhost protocol. For public network experience, please access under HTTPS protocol. For details, see Description of Network Access Protocol.

3. Open two browser pages, **enter different userID (defined by you)** click `step1. init` to login (caller and callee).

4. **After both userID init to successfully**, click on `step2. call` to make a call. If you have a call problem, refer to FAQs.



Additional Features

Setting Nickname, Avatar
Group Call
Floating Window
Custom Ringtone
Call Status Monitoring, Component Callback Event
Setting Resolution, Fill Pattern
Customize Interface

FAQs

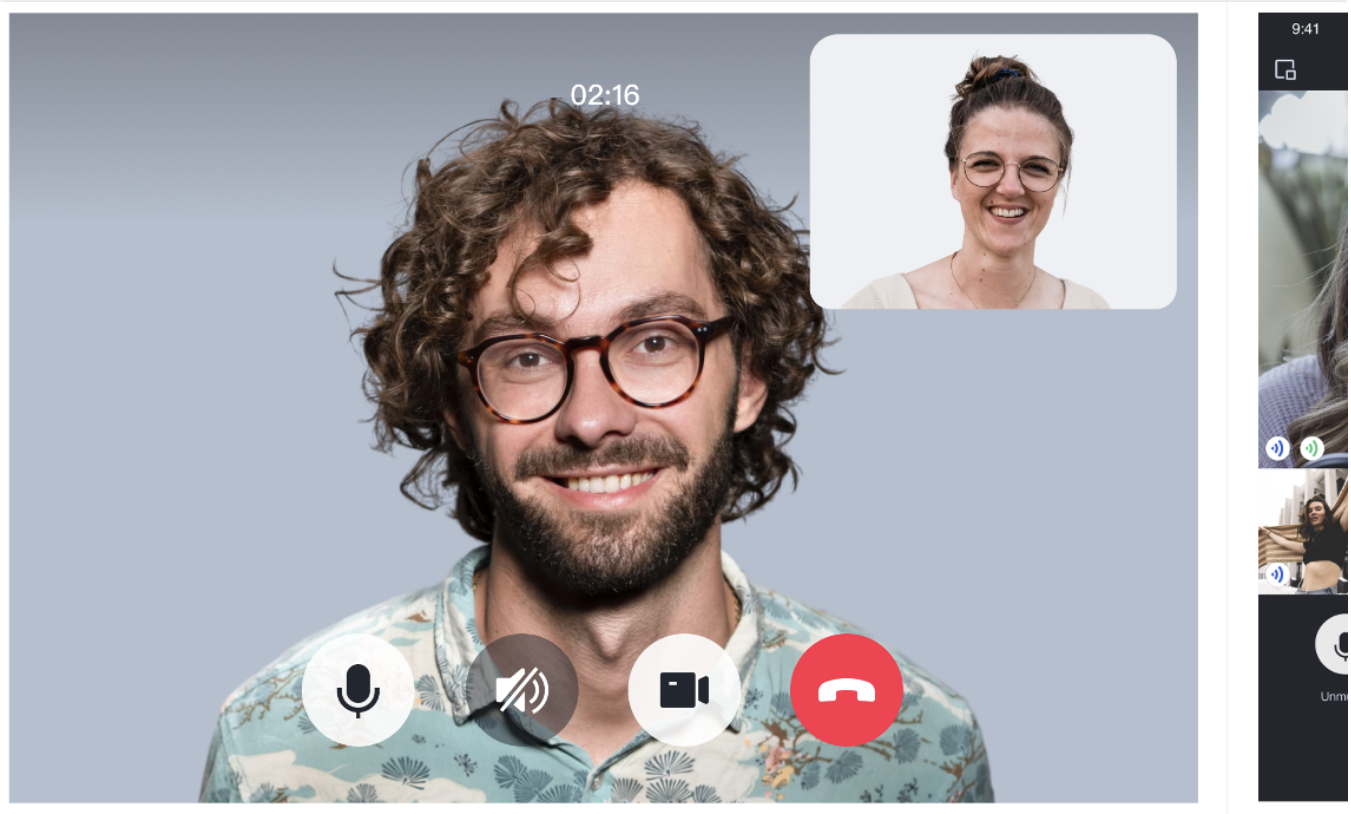
If you encounter any problems with access and use, please refer to FAQs.

If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

Web&H5 (Vue3)

Last updated：2024-07-22 17:21:41

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



Prerequisites

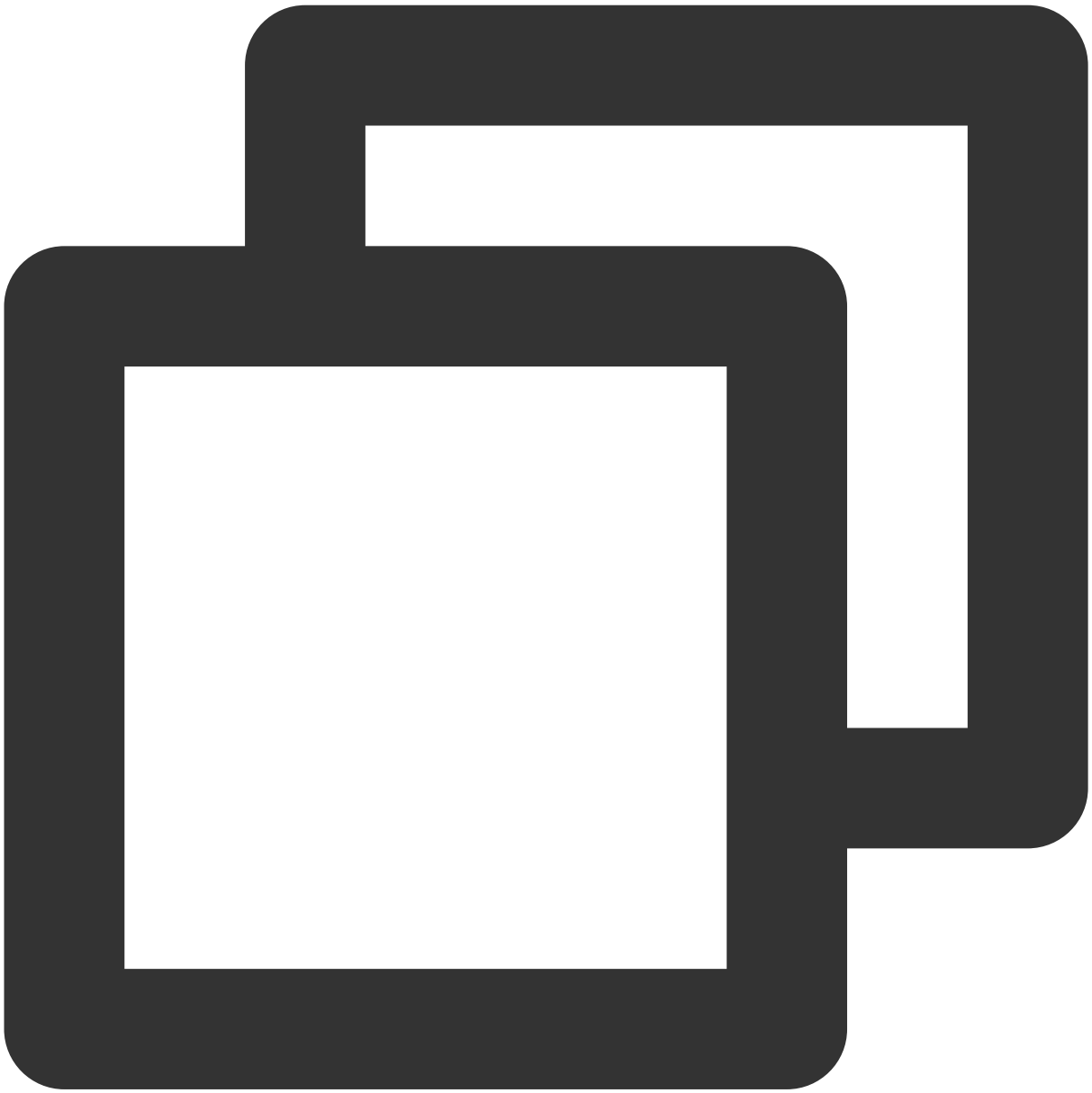
Node.js version 16+.
Modern browser, supporting WebRTC APIs.

Step 1. Activate the service

Refer to Activate the Service to obtain `SDKAppID, SDKSecretKey` , which will be used as **Mandatory Parameters in** Initialize the TUICallKit.

Step 2. Download the TUICallKit

1. Download the @tencentcloud/call-uikit-vue component.

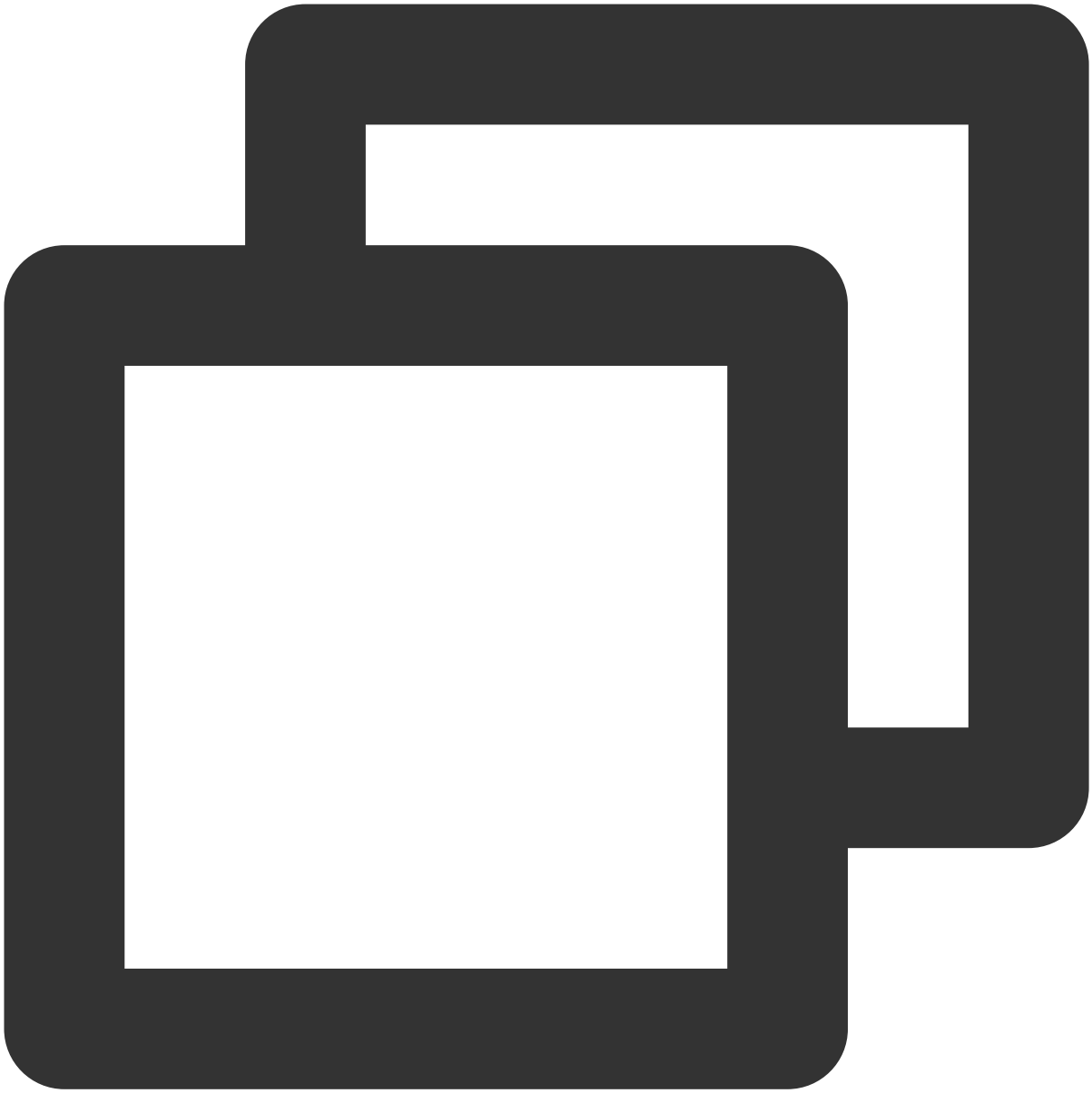


```
npm install @tencentcloud/call-uikit-vue
```

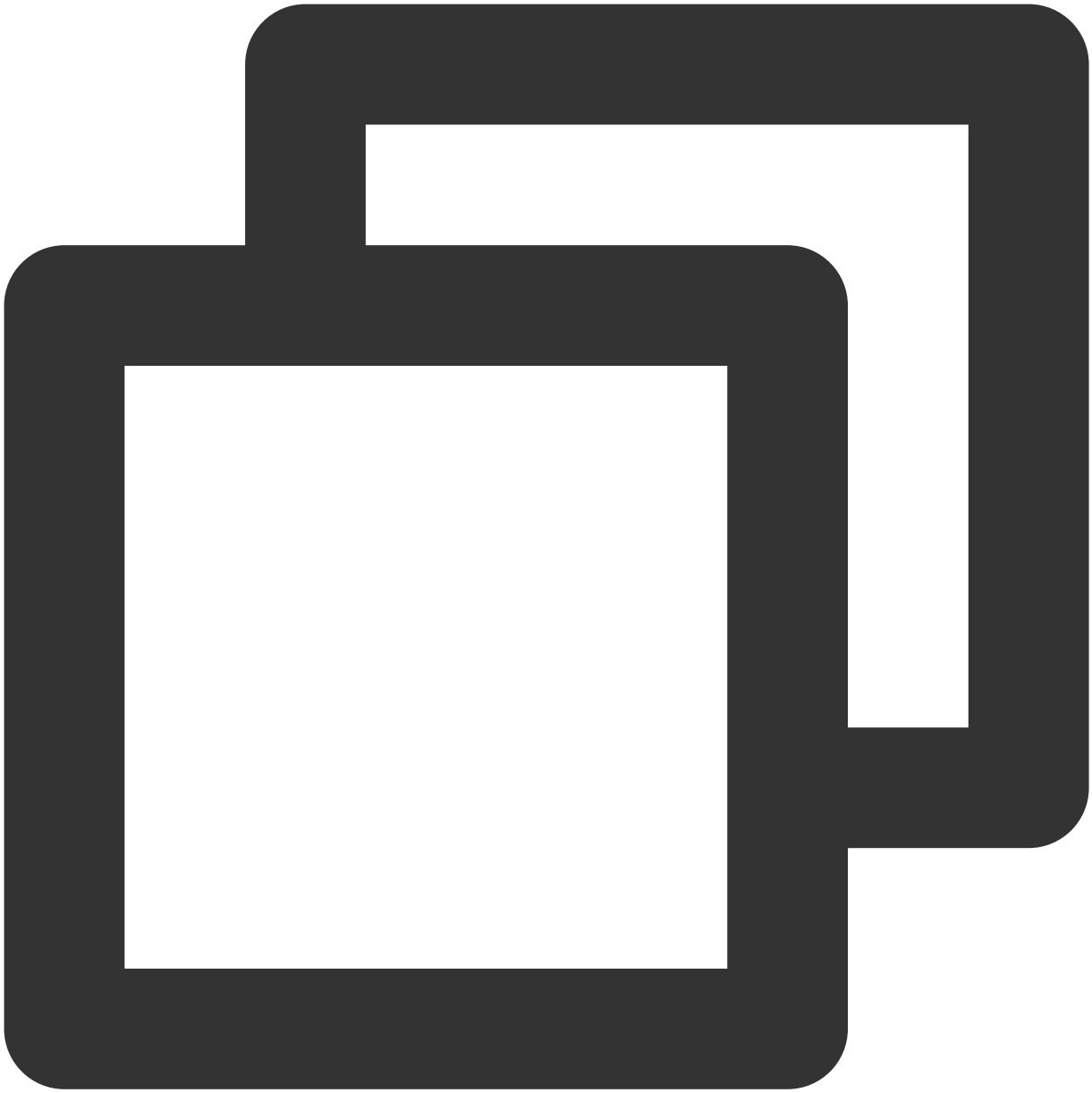
2. Copy the **debug** directory to your project directory `src/debug` , it is necessary when generating userSig locally.

MacOS

Windows



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

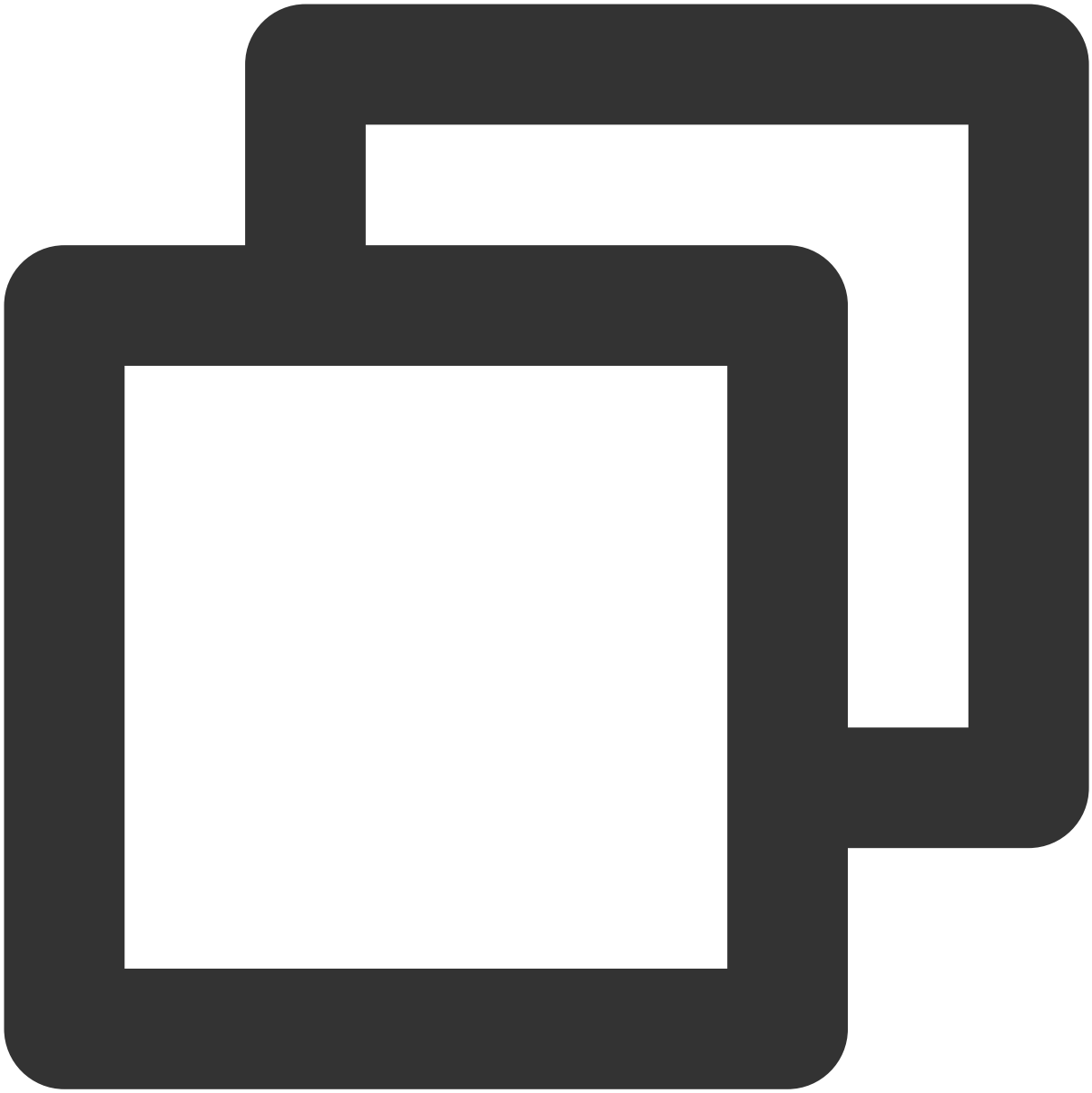


```
xcopy node_modules\\@tencentcloud\\call-uikit-vue\\debug .\\src\\debug /i /e
```

Step 3. Initialize the TUICallKit

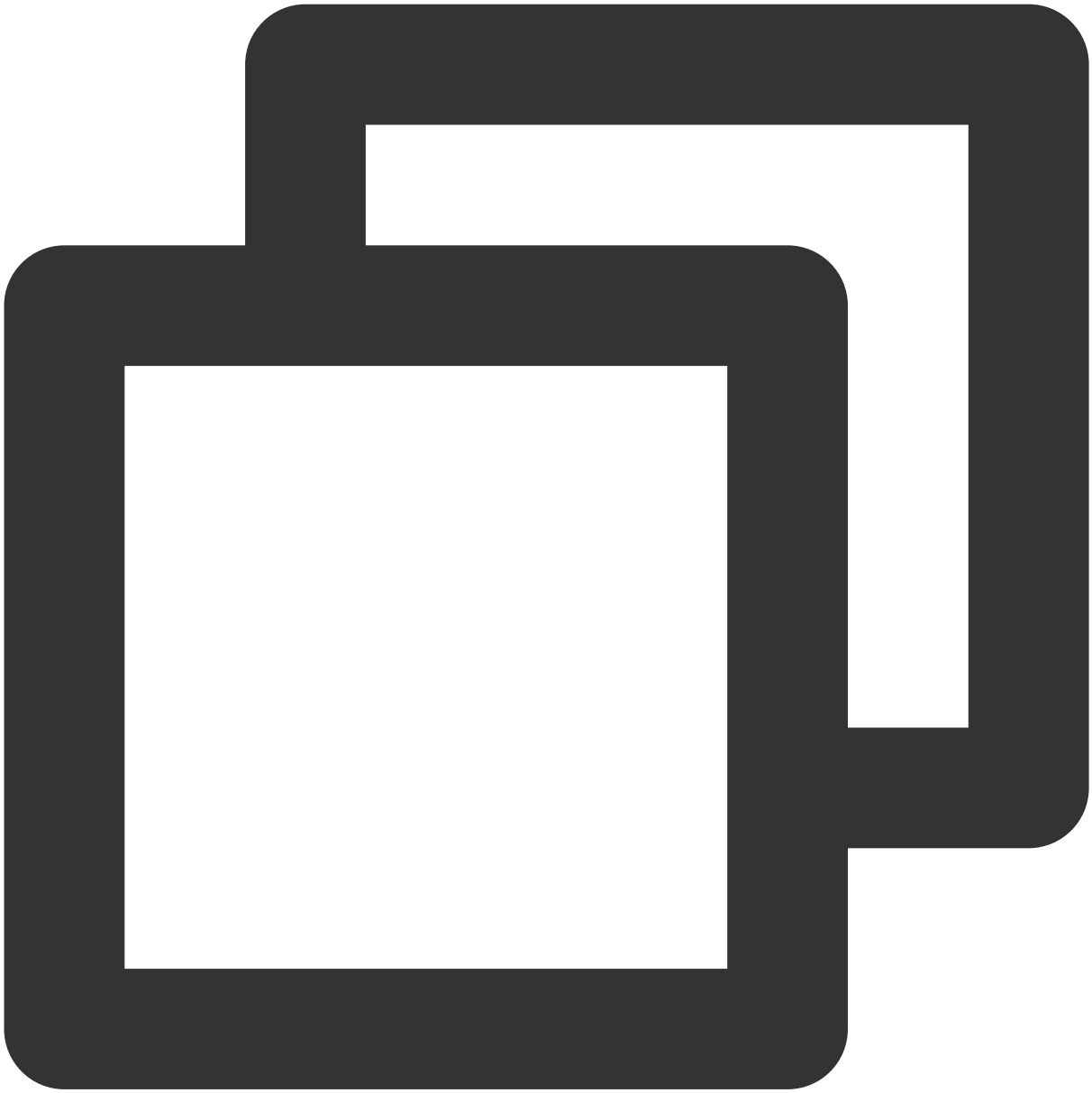
You can choose to import the sample code in the `src/App.vue` file.The example code uses the **`Composition API`** approach.

1. using <TUICallKit />, which contains the complete UI interaction during a call.



```
<template>
  <span> caller's ID: </span>
  <input type="text" v-model="callerUserID">
  <button @click="init"> step1. init </button> <br>
  <span> callee's ID: </span>
  <input type="text" v-model="calleeUserID">
  <button @click="call"> step2. call </button>

  <!-- 【1】Import the TUICallKit component: Call interface UI -->
  <TUICallKit style="width: 650px; height: 500px " />
</template>
```

2. using the TUICallKitServer.init API to log in to the component, you need to `fill in SDKAppID, SDKSecretKey` **as two parameters in the code.**



```
<script setup> // lang='ts'
import { ref } from 'vue';
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es"; // Refer to

const SDKAppID = 0;       // TODO: Replace with your SDKAppID (Notice: SDKAppID is
const SDKSecretKey = '';  // TODO: Replace with your SDKSecretKey
```

```
const callerUserID = ref('');
const calleeUserID = ref('');

// 【2】Initialize the TUICallKit component
const init = async () => {
  const { userSig } = GenerateTestUserSig.genTestUserSig({
    userID: callerUserID.value,
    SDKAppID,
    SecretKey: SDKSecretKey
  });
  await TUICallKitServer.init({
    userID: callerUserID.value,
    userSig,
    SDKAppID,
  });
  alert('TUICallKit init succeed');
}
</script>
```

Parameter	Type	Note
userID	String	**Unique identifier of the user,** `defined by you` **,** it is allowed to contain only upper and lower case letters (a-z, A-Z), numbers (0-9), underscores, and hyphens.
SDKAppID	Number	The unique identifier for the audio and video application created in the Tencent RTC Console.
SDKSecretKey	String	The SDKSecretKey of the audio and video application created in the Tencent RTC Console.
userSig	String	A security protection signature used for user log in authentication to confirm the user's identity and prevent malicious attackers from stealing your cloud service usage rights.

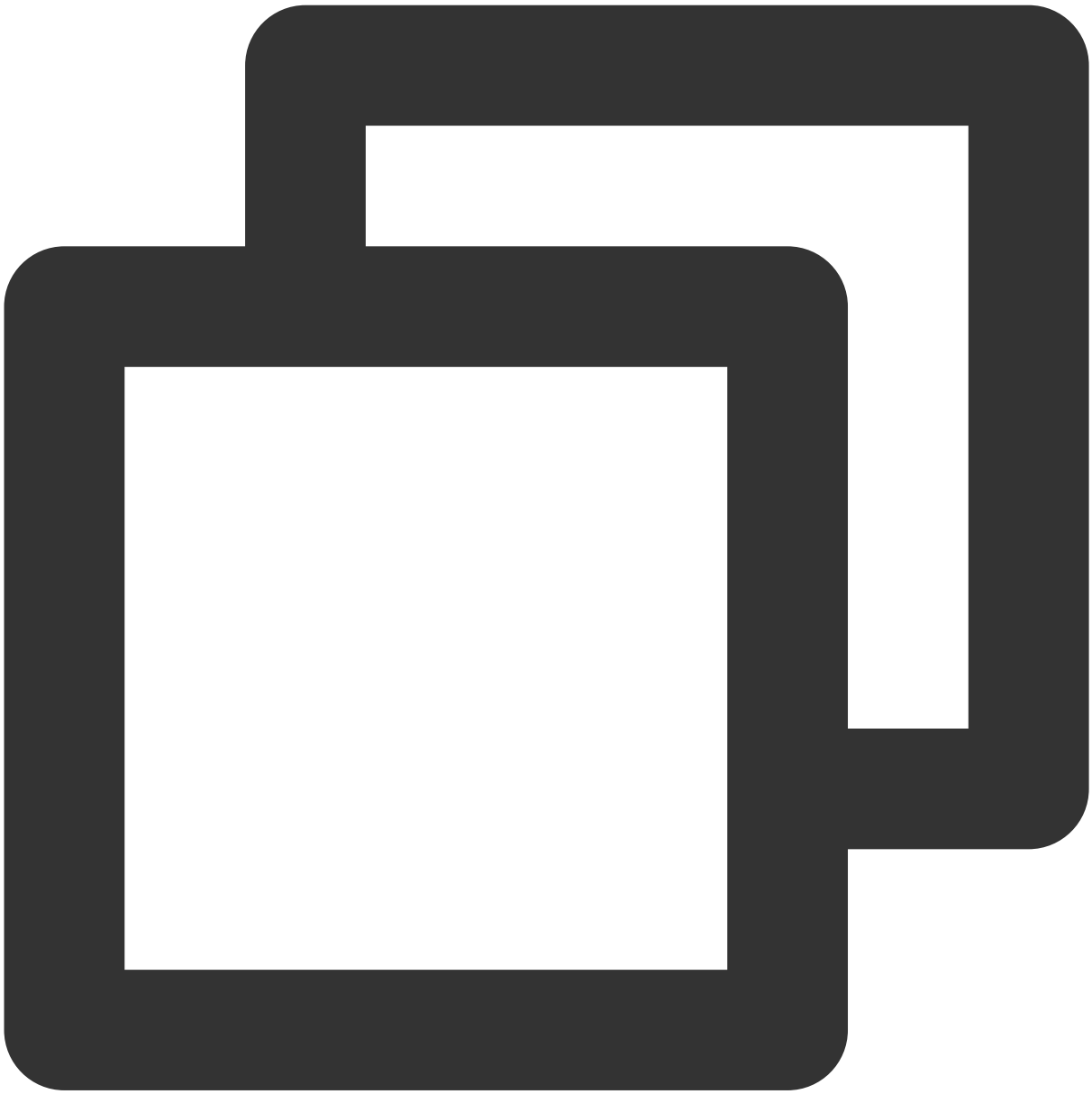
Explanation of userSig:

Development environment: If you are running a demo locally and developing debugging, you can use the `genTestUserSig` (Refer to Step 3.2) function in the debug file to generate a `userSig`. In this method, SDKSecretKey is vulnerable to decompilation and reverse engineering. Once your key is leaked, attackers can steal your Tencent Cloud traffic.

Production environment: If your project is going live, please use the Server-side Generation of UserSig method.

Step 4. Make your first call

1. using the TUICallKitServer.call API to make a call.



```
//【3】Make a 1v1 video call
const call = async () => {
  await TUICallKitServer.call({
    userID: calleeUserID.value,
    type: TUICallType.VIDEO_CALL,
  });
};
```

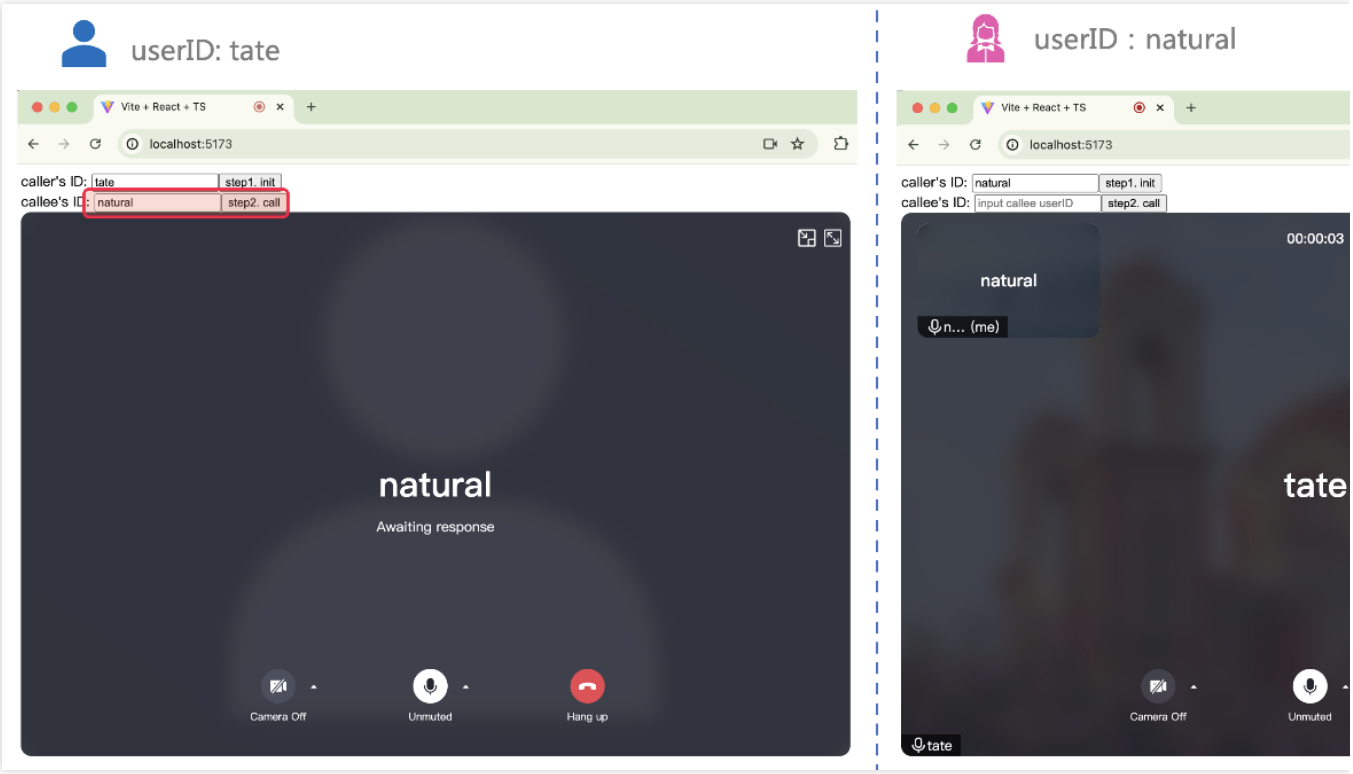
2. Run the project.

Warning:

For local environment, please access under `localhost protocol`. For public network experience, please access under HTTPS protocol. For details, see Description of Network Access Protocol**.**

3. Open two browser pages, **enter different userID(defined by you)** click `step1. init` to login (caller and callee).

4. **After both userID init to successfully**, click on `step2. call` to make a call. If you have a call problem, refer to FAQs.



Additional Features

Setting Nickname, Avatar
Group Call
Floating Window
Custom Ringtone
Call Status Monitoring, Component Callback Event
Setting Resolution, Fill Pattern
Customize Interface

FAQs

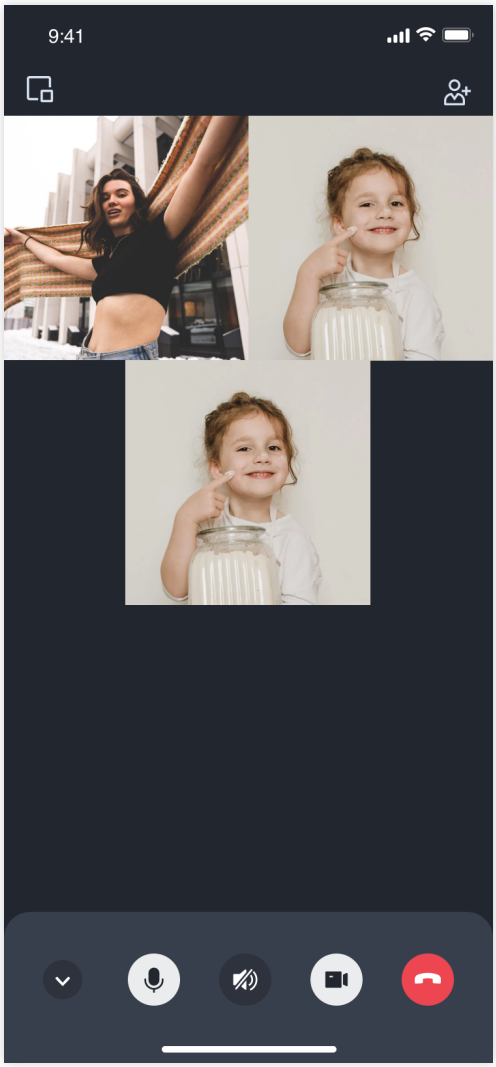
If you encounter any problems with access and use, please refer to FAQs.

If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

Flutter

Last updated：2024-06-20 16:55:01

This article will guide you through the process of integrating the TUICallKit component quickly. By following this documentation, you can complete the access work in just 10 minutes and ultimately obtain an application with a complete user interface as well as audio and video calling features.

Video Call	Group call
	

Environment Preparations

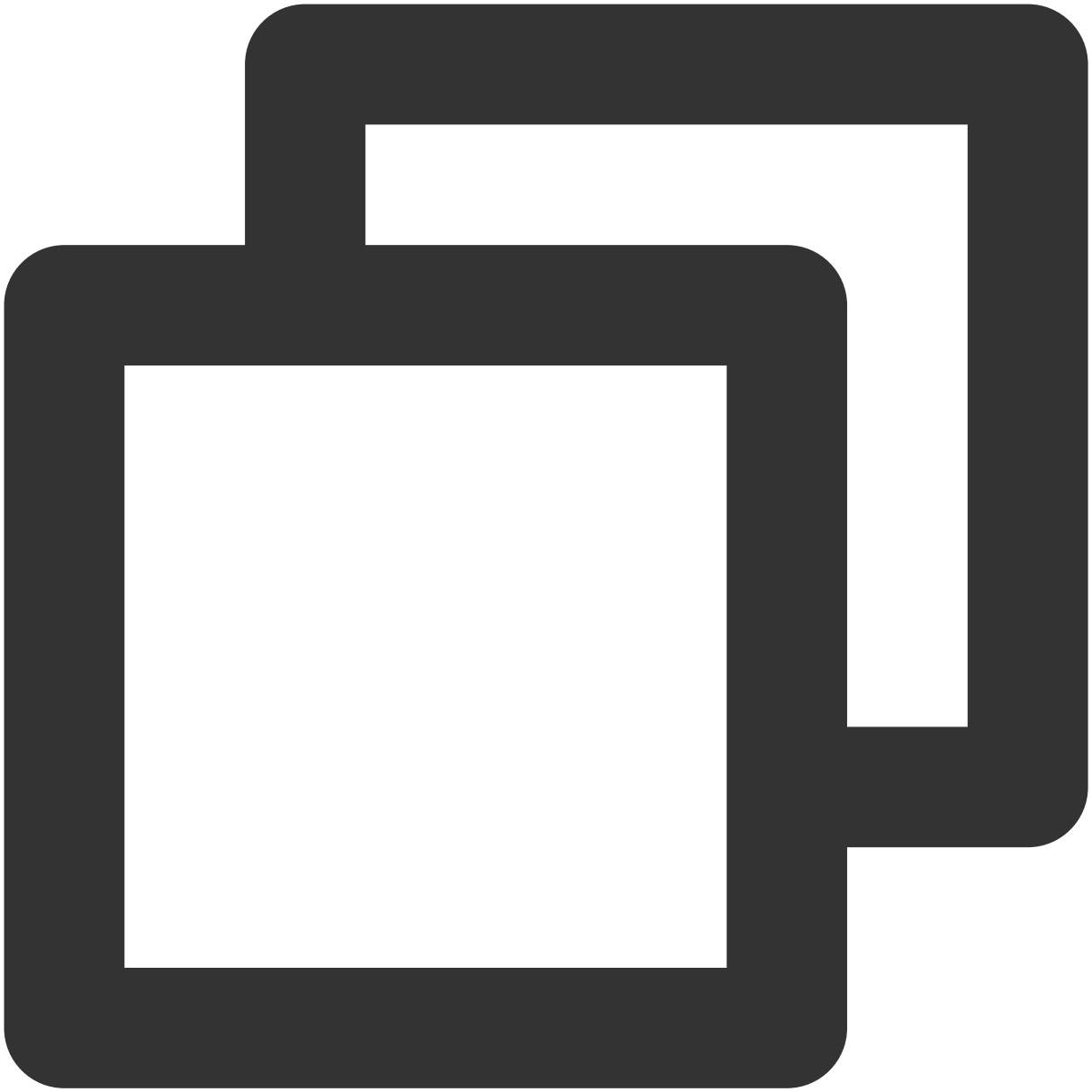
Flutter 3.0 or higher version.

Step 1. Activate the service

Before using the audio and video services provided by Tencent Cloud, you need to go to the console to activate the audio and video services for your application, and obtain `SDKAppID, SDKSecretKey` . They will be used in Step 5. For specific steps, please refer to activate the Service.

Step 2. Import the component

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



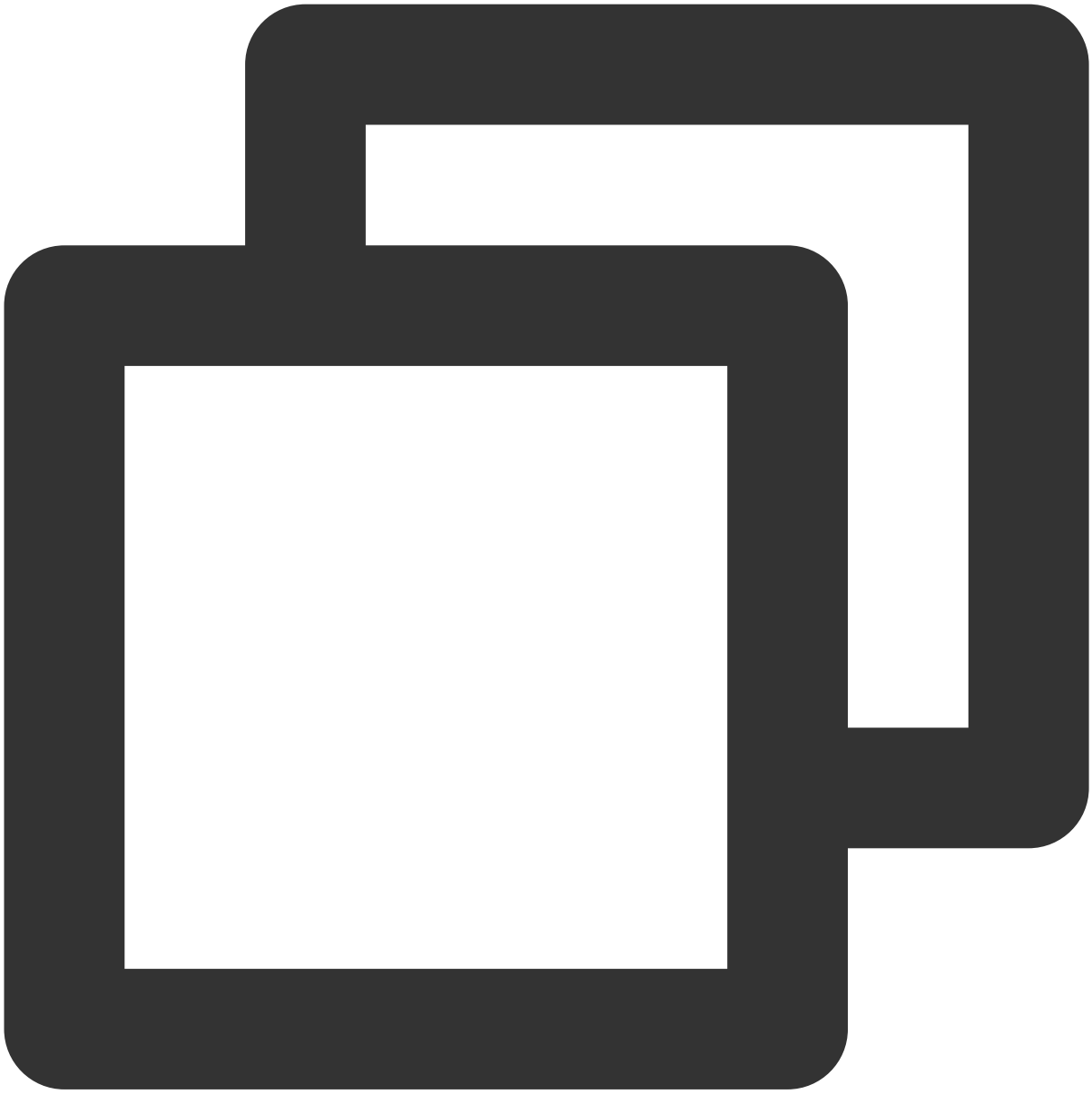
```
flutter pub add tencent_calls_uikit
```

Step 3. Configure the project

Android

iOS

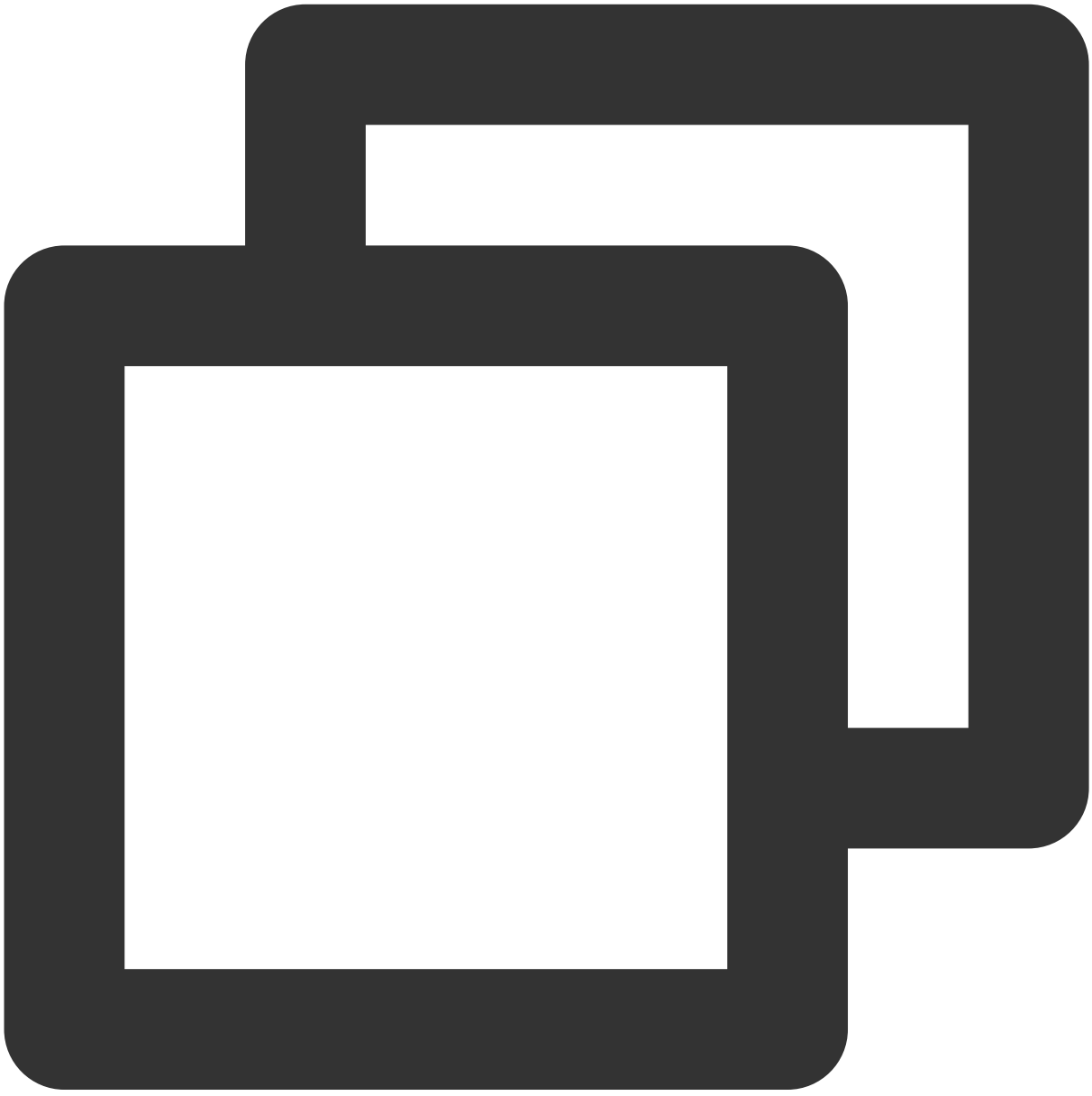
1. If you need to compile and run on the Android platform, since the SDK uses Java's reflection feature internally, certain classes in the SDK must be added to the non-aliasing list.
First, configure and enable obfuscation rules in the `android/app/build.gradle` file of the project:



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

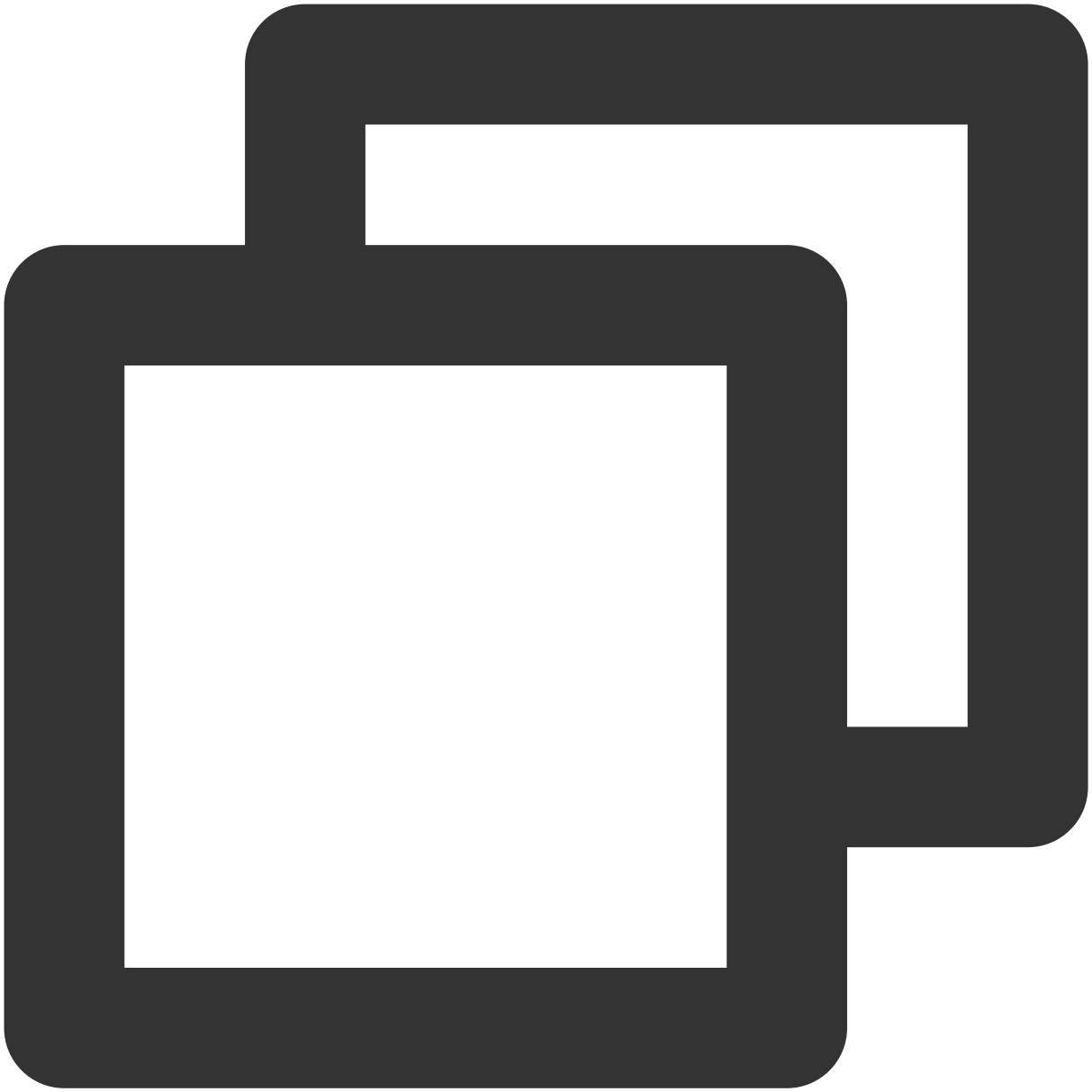
```
        }
    }
}
```

Create a `proguard-rules.pro` file in the `android/app` directory of the project, and add the following code in the `proguard-rules.pro` file:



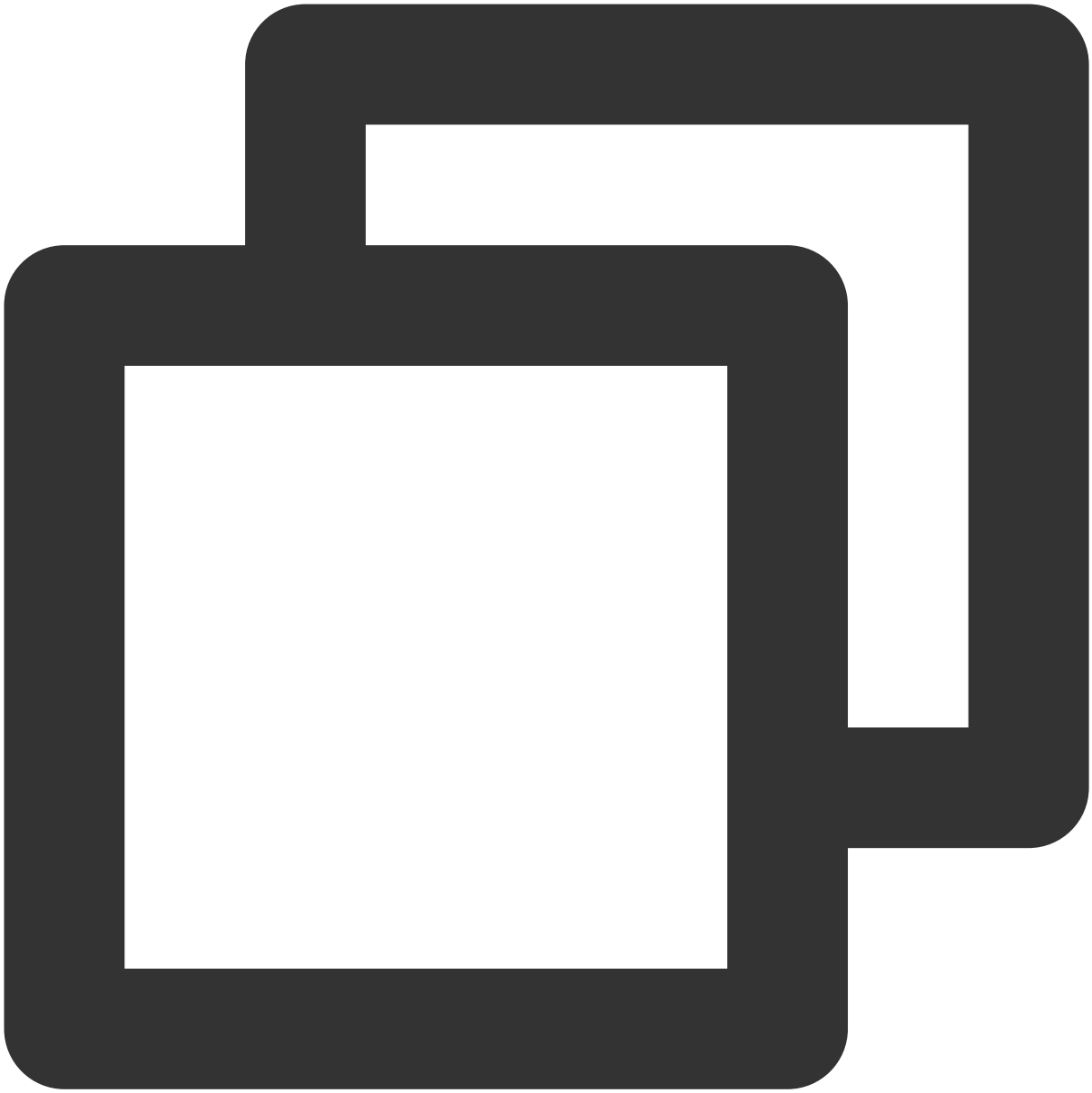
```
-keep class com.tencent.** { *; }
```

2. Configure to enable Multidex support in the `android/app/build.gradle` file of your project.



```
  android {
    ......
    defaultConfig {
      ......
      multiDexEnabled true
    }
}
```

1. **Optional** If you need to debug on an iOS Simulator and your Mac Computer uses an Intel Chip, you need to add the following code in the `ios/Podfile` file:

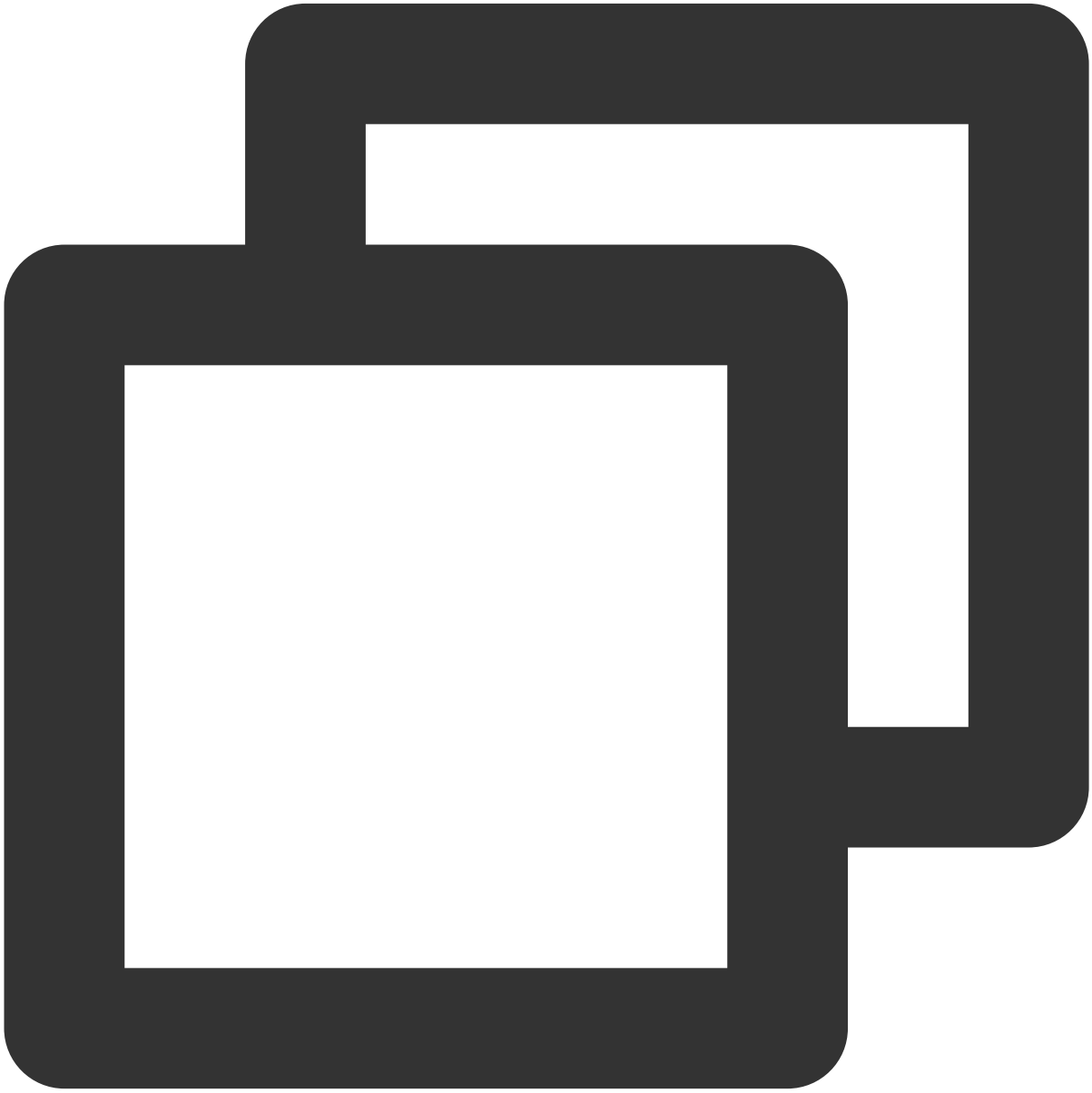


```
target 'xxxx' do
  ......
end
......

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

```
    end
  end
end
```

2. Since TUICallKit uses iOS's audio and video features, you need to grant permissions for the use of the microphone and camera.

Authorization Operation Method: In your iOS project's `Info.plist` , under the first-level `<dict>` directory, add the following two items. They correspond to the system's prompt messages when asking for microphone and camera permissions.

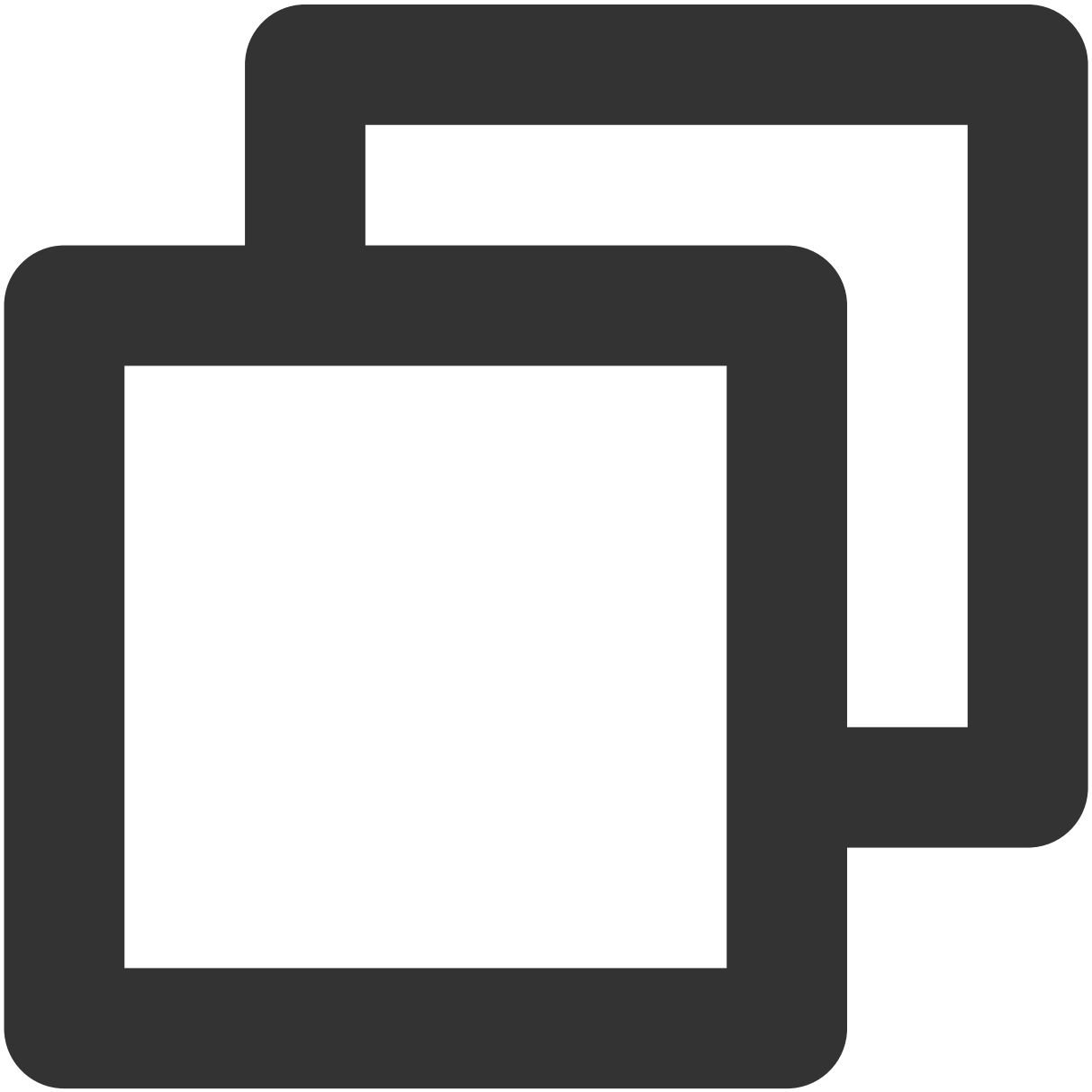


```
<key>NSCameraUsageDescription</key>
```

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

Step 4: Set up navigatorObservers

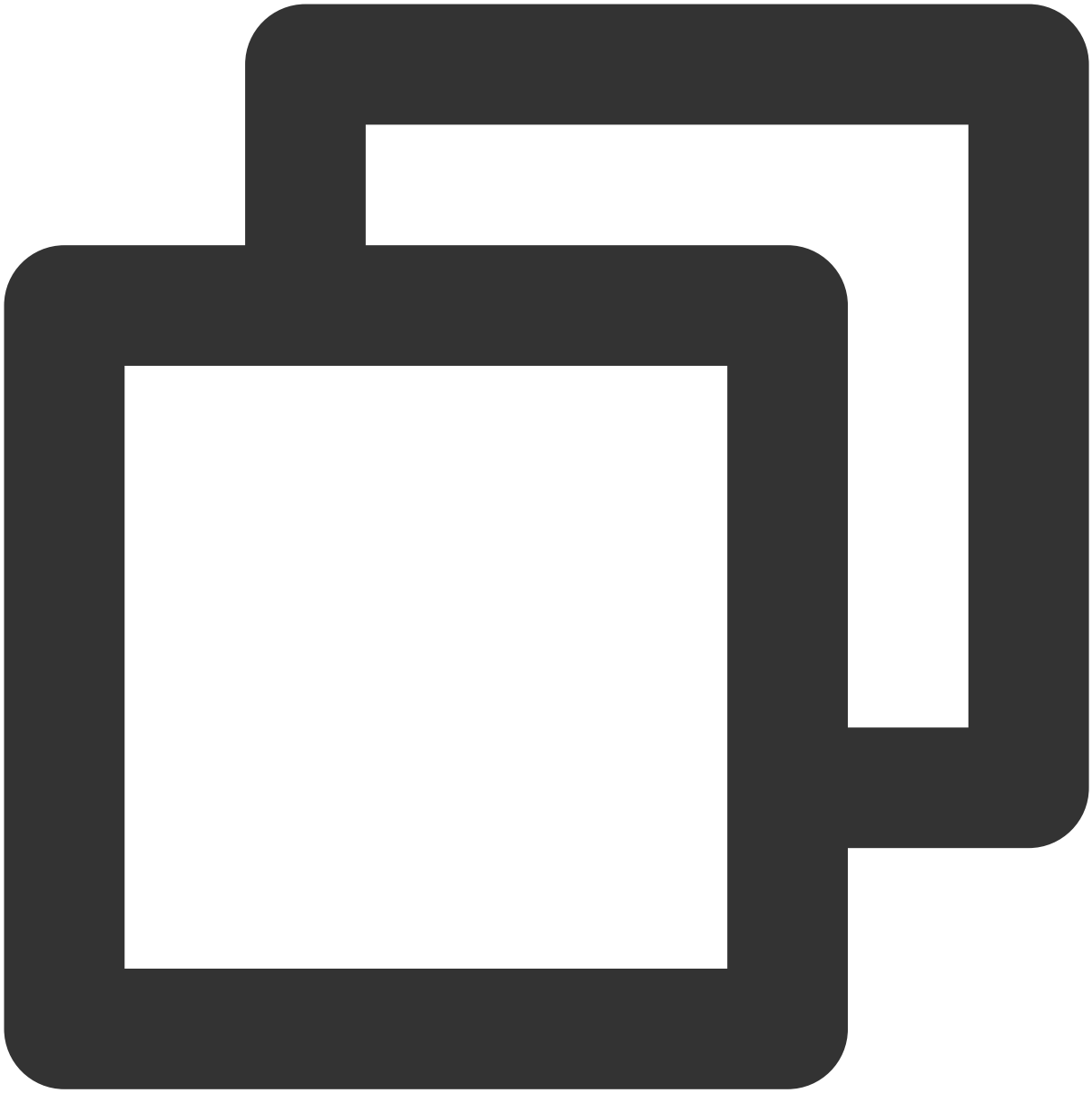
1. In the Flutter application framework, add TUICallKit.navigatorObserver to navigatorObservers. For example, using the MaterialApp framework, the code is as follows:



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

 ......

class XXX extends StatelessWidget {
  const XXX({super.key});

 @override
  Widget build(BuildContext context) {
    return MaterialApp(
      navigatorObservers: [TUICallKit.navigatorObserver],
      ......
    );
  }
}
```

Step 5: Log in to the TUICallKit Component

Use the login interface to complete the log-in. For specific usage, refer to the following code:



```
import 'package:tencent_calls_uikit/tencent_calls_uikit.dart';
import 'package:tencent_calls_uikit/debug/generate_test_user_sig.dart';
......

final String userID    = 'xxxxx'; // Please replace with your UserId
final int    sdkAppID  = 0;       // Please replace with the SDKAppID you got from
final String secretKey = 'xxxx';  // Please replace with the SecretKey you got from

void login() async {
    String userSig  = GenerateTestUserSig.genTestSig(userID, sdkAppID, secretKey);
    TUIResult result = await TUICallKit.instance.login(sdkAppID, userID, userSig);
```

```
    if (result.code.isEmpty) {
      print('Login success');
    } else {
      print('Login failed: ${result.code} ${result.message}');
    }
}
```

Parameter	Type	Description
userID	String	Customers define their own User ID based on their business. You can only include letters (a-z, A-Z), digits (0-9), underscores, and hyphens.
sdkAppID	int	The unique identifier of the audio and video application created in the Tencent RTC Console.
secretKey	String	SDKSecretKey for the audio and video application created in Tencent RTC Console.
userSig	String	A security protection signature used for user log in authentication to confirm the user's identity and prevent malicious attackers from stealing your cloud service usage rights.

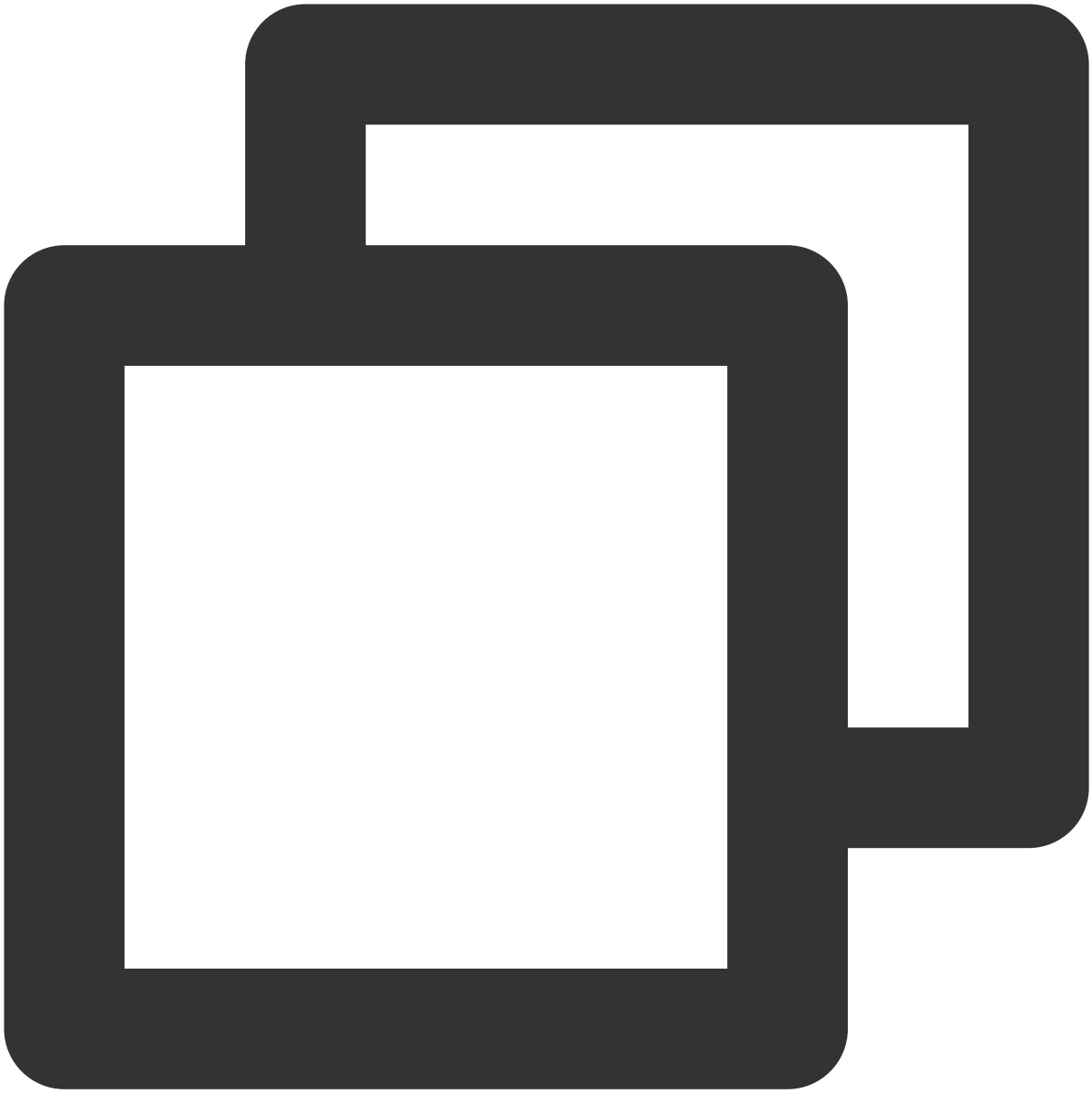
Note:

Development Environment: If you are in the local development and debugging stage, you can use the local `GenerateTestUserSig.genTestSig` function to generate userSig. In this method, the SDKSecretKey is vulnerable to decompilation and reverse engineering, and once your key is leaked, attackers can steal your Tencent Cloud traffic.

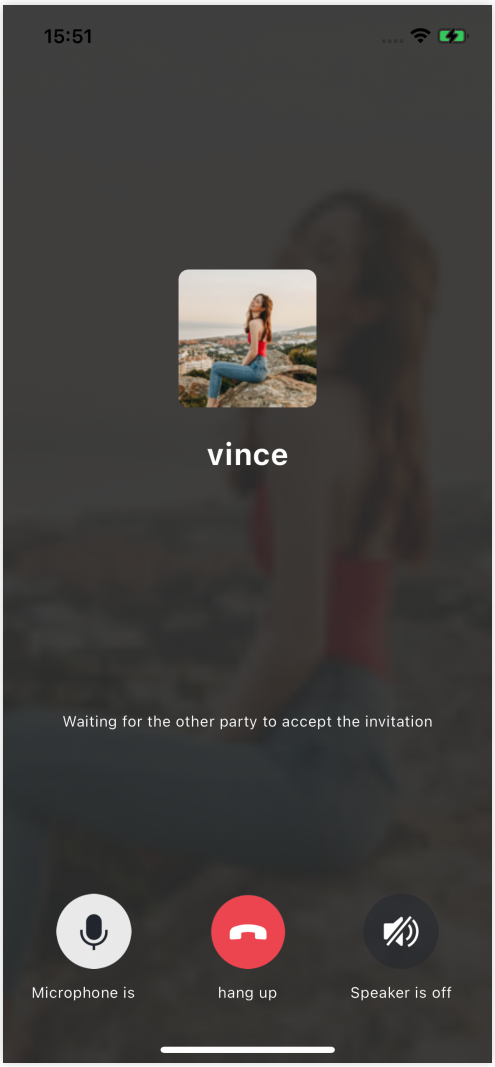
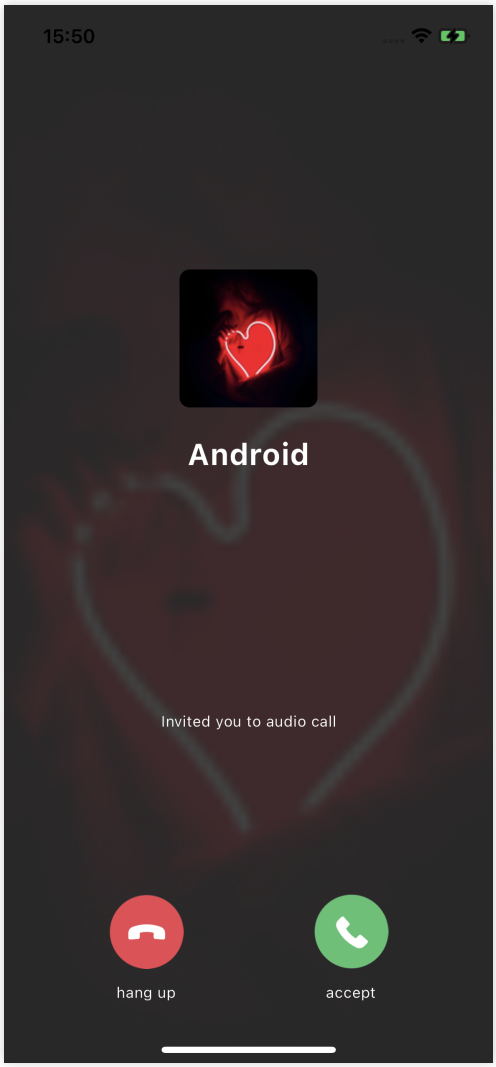
Production Environment: If your project is going to be launched, please adopt the method of Server-side Generation of UserSig.

Step 6. Make your first phone call

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



```
TUICallKit.instance.call('Android', TUICallMediaType.audio);import 'package:tencent
......

void call() {
    TUICallKit.instance.call('Android', TUICallMediaType.audio);
}
```

	
Caller	Callee

Additional Features

Customize Interface
Offline Push
Group Call
Floating Window
Custom Ringtone
Call Status Monitoring
Cloud Recording
Beauty Effects

FAQs

If you encounter any issues during integration and use, please refer to Frequently Asked Questions.

Suggestions and Feedback

If you have any suggestions or feedback, please contact info_rtc@tencent.com.

uniapp（Android&iOS）

Last updated：2024-04-28 14:56:43

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

TUICallKit Demo Experience

TUICallKit Plugin Address: TUICallKit Plugin Link .
If you want to quickly run a new project, please read uni-app Demo Quick Start directly.

Environment Requirements

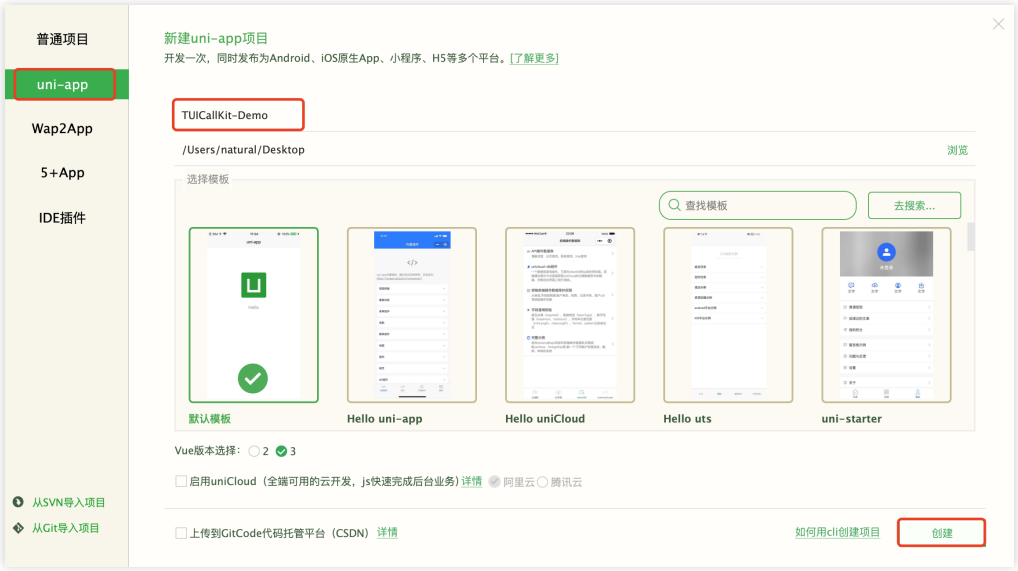
HbuilderX version requirement: HbuilderX version ≥ 3.94.
Plugin Debugging Notes: Native plugins do not currently support simulator debugging.
iOS Device Requirements: iOS system ≥ 9.0, supports audio and video calls on actual devices.
Android Device Requirements: Android system ≥ 5.0 (SDK API Level 21), supports audio and video calls on actual devices, Allow USB Debugging.

Step 1. Activate the service

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

Step 2: Create a uni-app Project

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



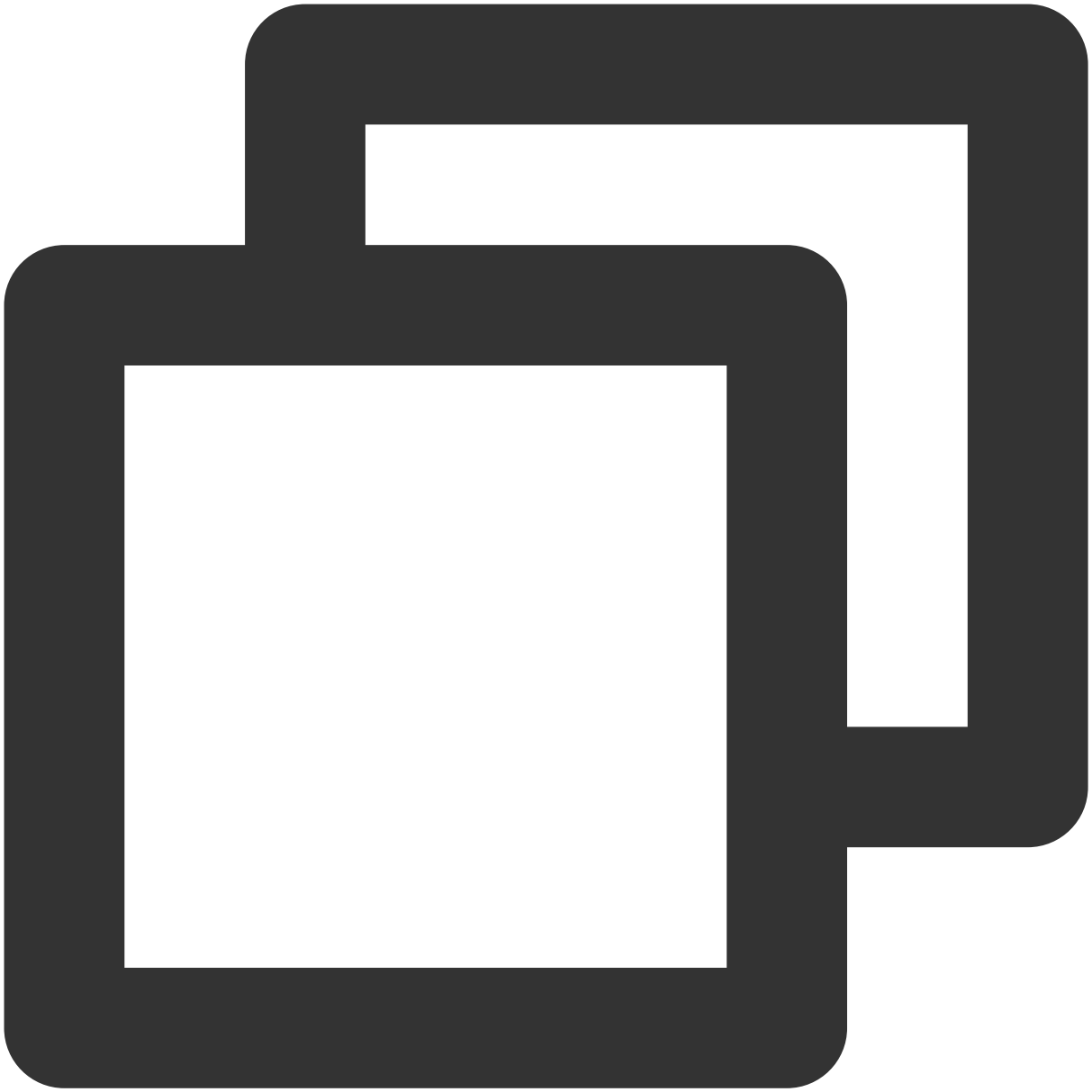
Step 3: Download and import the TUICallKit Plugin

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

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

Step 4: Use the TUICallKit Plugin

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



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

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

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

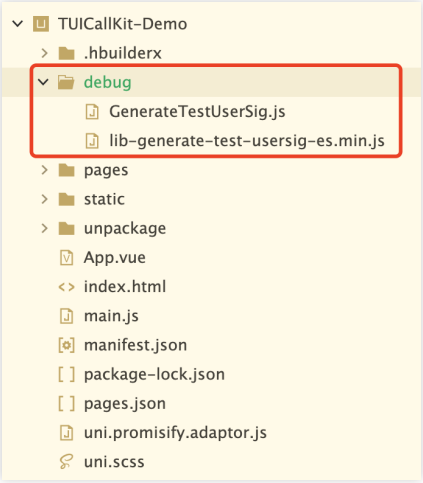
2. Enter the SDKAppID, SDKSecretKey, and userSig parameters.

Client-side userSig generation

Console-generated userSig

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

1. Click to download the debug folder, and copy the debug directory to your project, as shown below:

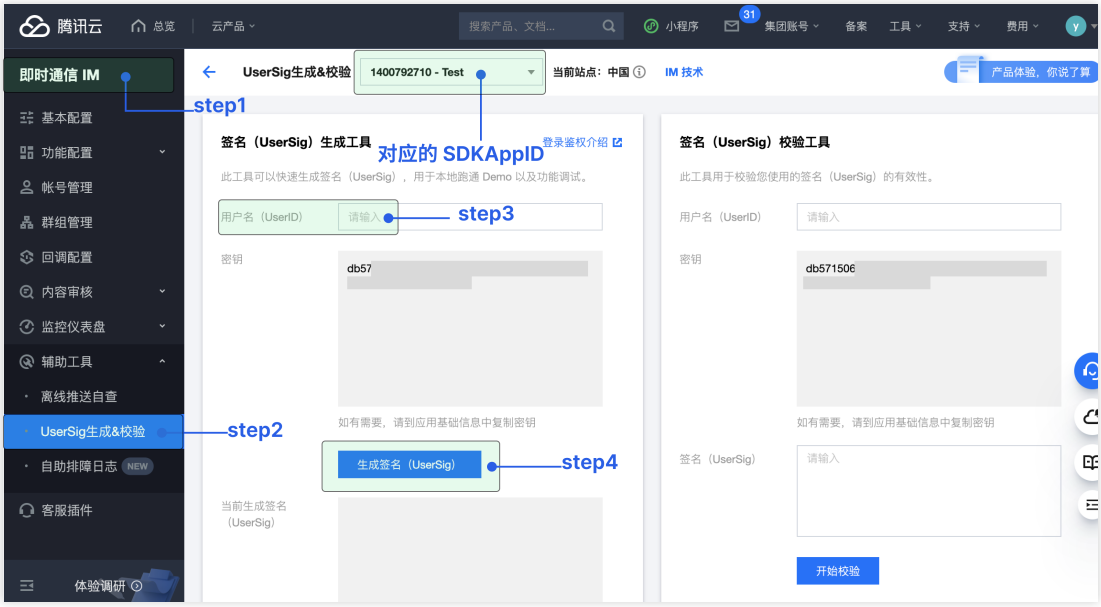


2. Fill in the `TUICallKit-Demo/debug/GenerateTestUserSig.js` file with `SDKAppID` and `SDKSecretKey` (refer to Activate Service)

GenerateTestUserSig.js

```
import LibGenerateTestUserSig from './lib-generate-test-usersig-es.min.js';
/** ... */

const SDKAPPID = 0;
/** ... */

const EXPIRETIME = 604800;
/** ... */

const SECRETKEY = '';
/* ... */

export function genTestUserSig(userID) {
  const generator = new LibGenerateTestUserSig(SDKAPPID, SECRETKEY, EXPIRETIME);
  const userSig = generator.genTestUserSig(userID);

  return {
    sdkAppID: SDKAPPID,
    userSig,
  };
}
```

If you want to quickly experience TUICallKit, you can generate a temporary UserSig available through the **Auxiliary Tools** in the console.



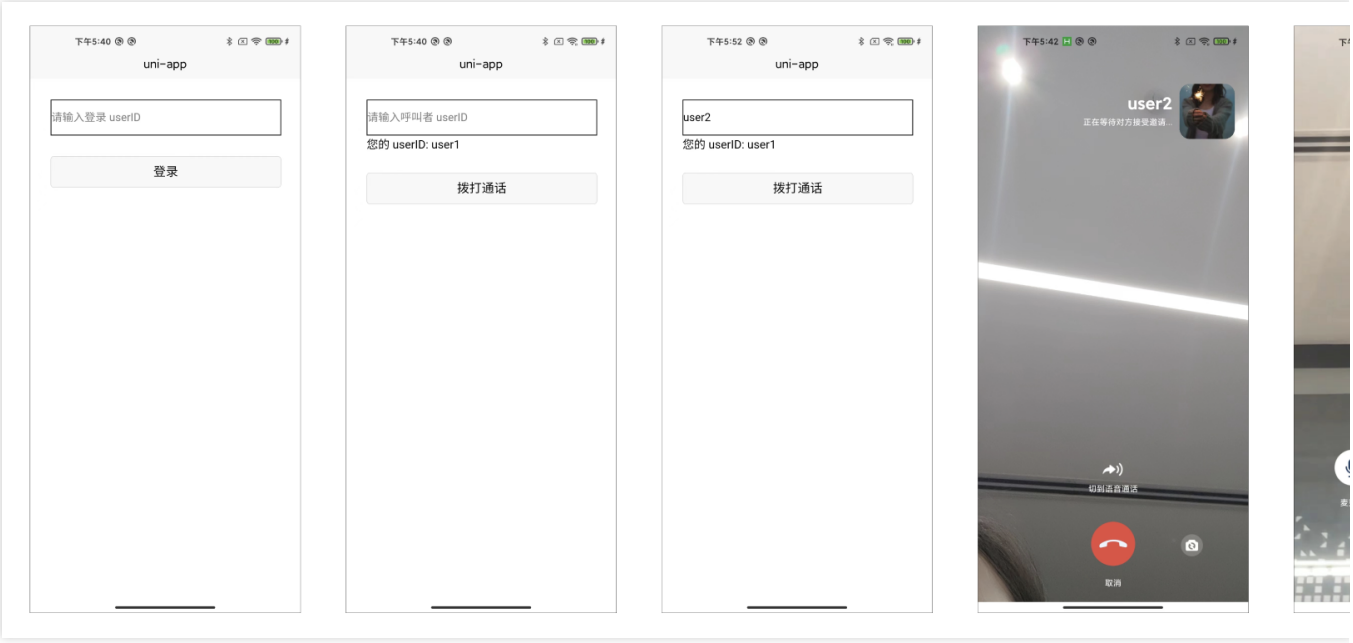
If you are using **Console Generation**, you will need to assign the `SDKAppID, userSig` values in the `TUICallKit-Demo/pages/index/index.vue` file.

Step Five: Make your first phone call

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

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

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



Additional Features

Setting Nickname, Avatar

Group Call

Floating Window

Custom Definition Ringtone

Call Status Monitoring

FAQs

If you encounter any issues during integration and use, please refer to Frequently Asked Questions.

Related Case - Online Customer Service Scenario

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

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

Supports two-person voice and video call features.

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

Technical Consultation

If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

UI Customization (TUICallKit)
Android

Last updated：2024-05-31 16:42:21

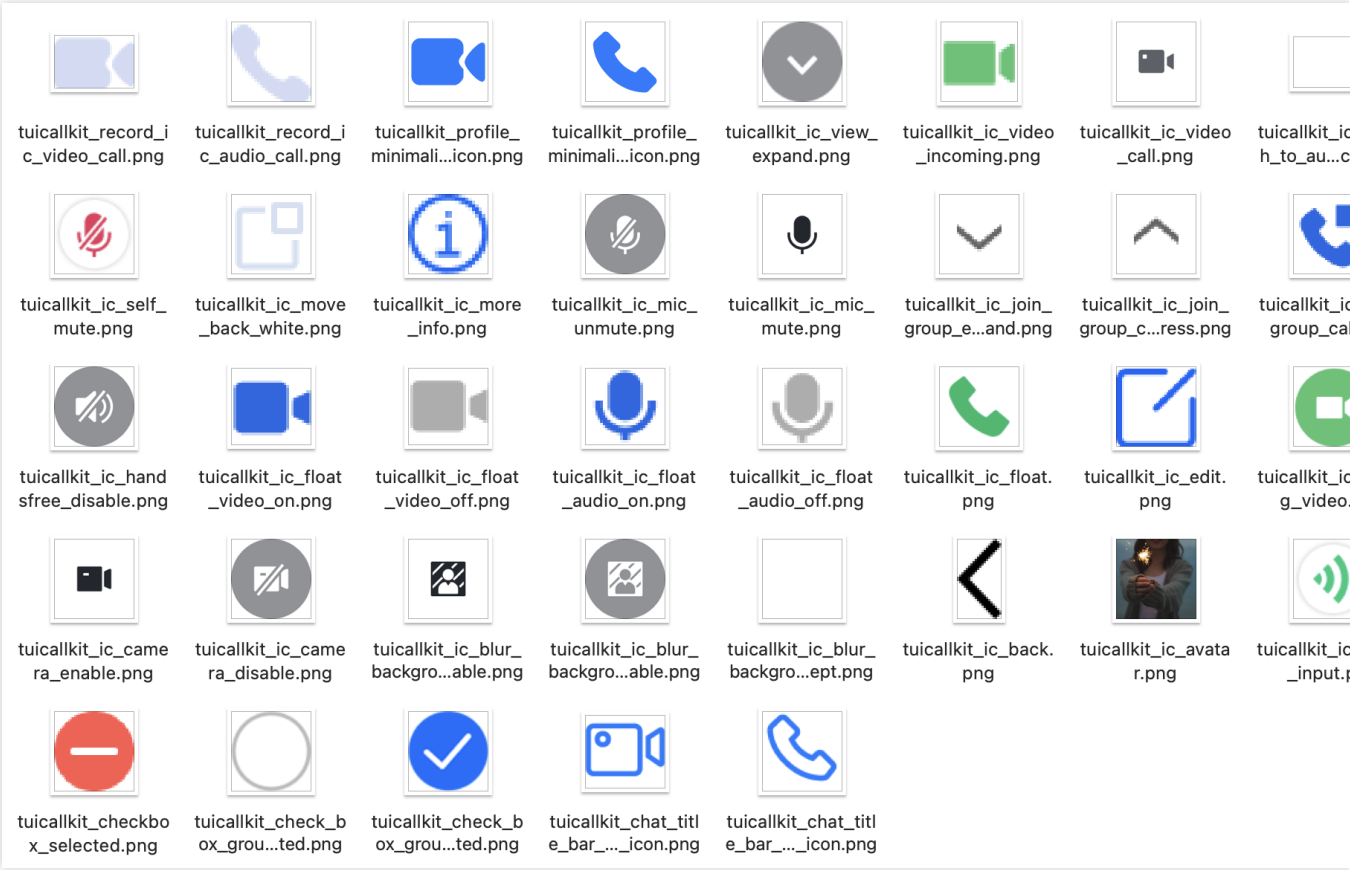
This document describes how to customize the UI of `TUICallKit` and provides two schemes for customization: **slight UI adjustment** and **custom UI implementation**.

Scheme 1. Slight UI Adjustment

You can adjust the UI of `TUICallKit` by directly modifying the UI source code in the `Android/tuicallkit-kt` folder in tencentyun/TUICallKit.

Replacing icons

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



Replacing ringtones

You can replace ringtones by replacing the three audio files in the `tuicallkit-kt/src/main/res/raw` folder.

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

Replacing text

You can modify the strings on the video call UI by modifying the `strings.xml` file in `tuicallkit-kt/src/main/res/values-**/` .

Scheme 2. Custom UI Implementation

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

TUICallEngine

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

API	Description
createInstance	Creates a `TUICallEngine` instance (singleton).
destroyInstance	Terminates a `TUICallEngine` instance (singleton).
init	Completes the authentication of basic audio/video call capabilities.
addObserver	Registers an event listener.
removeObserver	Unregisters an event listener.
call	Makes a one-to-one call.
groupCall	Makes a group call.
accept	Answers a call.
reject	Declines a call.
hangup	Hangs up a call.
ignore	Ignores a call.
inviteUser	Invites a user during a group call.
joinInGroupCall	Joins the current group call actively.
switchCallMediaType	Switches the call media type, such as from video call to audio call.
startRemoteView	Subscribes to the video stream of a remote user.
stopRemoteView	Unsubscribes from the video stream of a remote user.
openCamera	Enables the camera.
closeCamera	Disables the camera.
switchCamera	Switches between the front and rear cameras.
openMicrophone	Enables the mic.
closeMicrophone	Disables the mic.
selectAudioPlaybackDevice	Selects the audio playback device (receiver/speaker on the device).

setSelfInfo	Sets the user nickname and profile photo.
enableMultiDeviceAbility	Enables/Disables the multi-device login mode of `TUICallEngine` (supported by the premium plan).
setVideoRenderParams	Set the rendering mode of video image.
setVideoEncoderParams	Set the encoding parameters of video encoder.
getTRTCCloudInstance	Advanced features.
setBeautyLevel	Set beauty level, support turning off default beauty.

TUICallObserver

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

API	Description
onError	An error occurred during the call.
onCallReceived	A call was received.
onCallCancelled	The call was canceled.
onCallBegin	The call was connected.
onCallEnd	The call ended.
onCallMediaTypeChanged	The call media type changed.
onUserReject	A user declined the call.
onUserNoResponse	A user didn't respond.
onUserLineBusy	A user was busy.
onUserJoin	A user joined the call.
onUserLeave	A user left the call.
onUserVideoAvailable	Whether a user had a video stream.
onUserAudioAvailable	Whether a user had an audio stream.
onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.

Definitions of Key Types

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

iOS

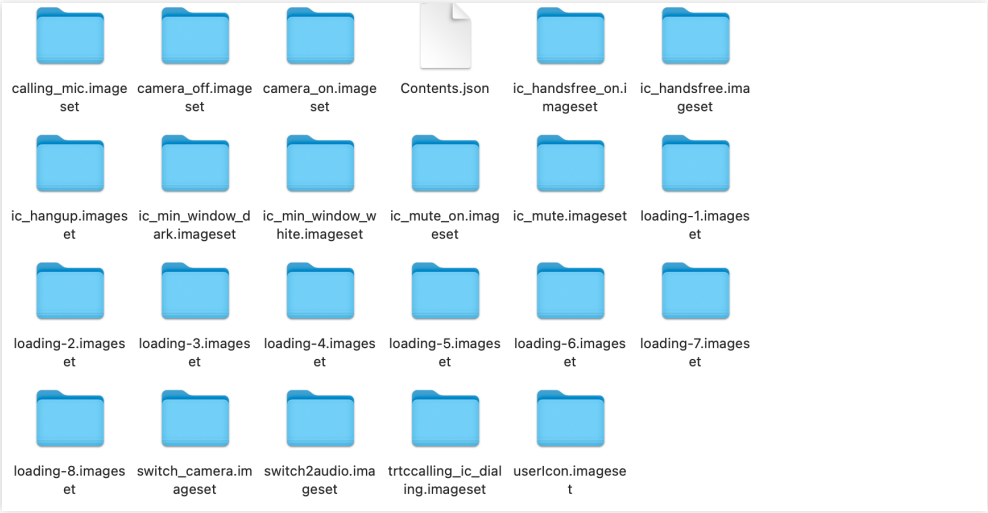
Last updated：2024-05-24 18:35:12

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

Scheme 1. Slight UI Adjustment

You can adjust the UI of `TUICallKit` by directly modifying the UI source code in the `iOS/TUICallKit-Swift` folder in tencentyun/TUICallKit.

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



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

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

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

Scheme 2. Custom UI Implementation

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

TUICallEngine

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

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

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

TUICallObserver

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

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

onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.

Definitions of key classes

API	Description
TUICallMediaType	The call media type. Enumeration: Video and Audio.
TUICallRole	The call role. Enumeration: Call and Called.
TUICallStatus	The call status. Enumeration: None, Waiting, and Accept.
TUIRoomId	The audio/video room ID, which can be a number or string.
TUICamera	The camera type. Enumeration: Front and Back.
TUIAudioPlaybackDevice	The audio playback device type. Enumeration: Speakerphone and Earpiece.
TUINetworkQualityInfo	The information of the current network quality.

Web

Last updated：2024-07-29 15:40:46

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

Scheme 1: Slight UI Adjustment

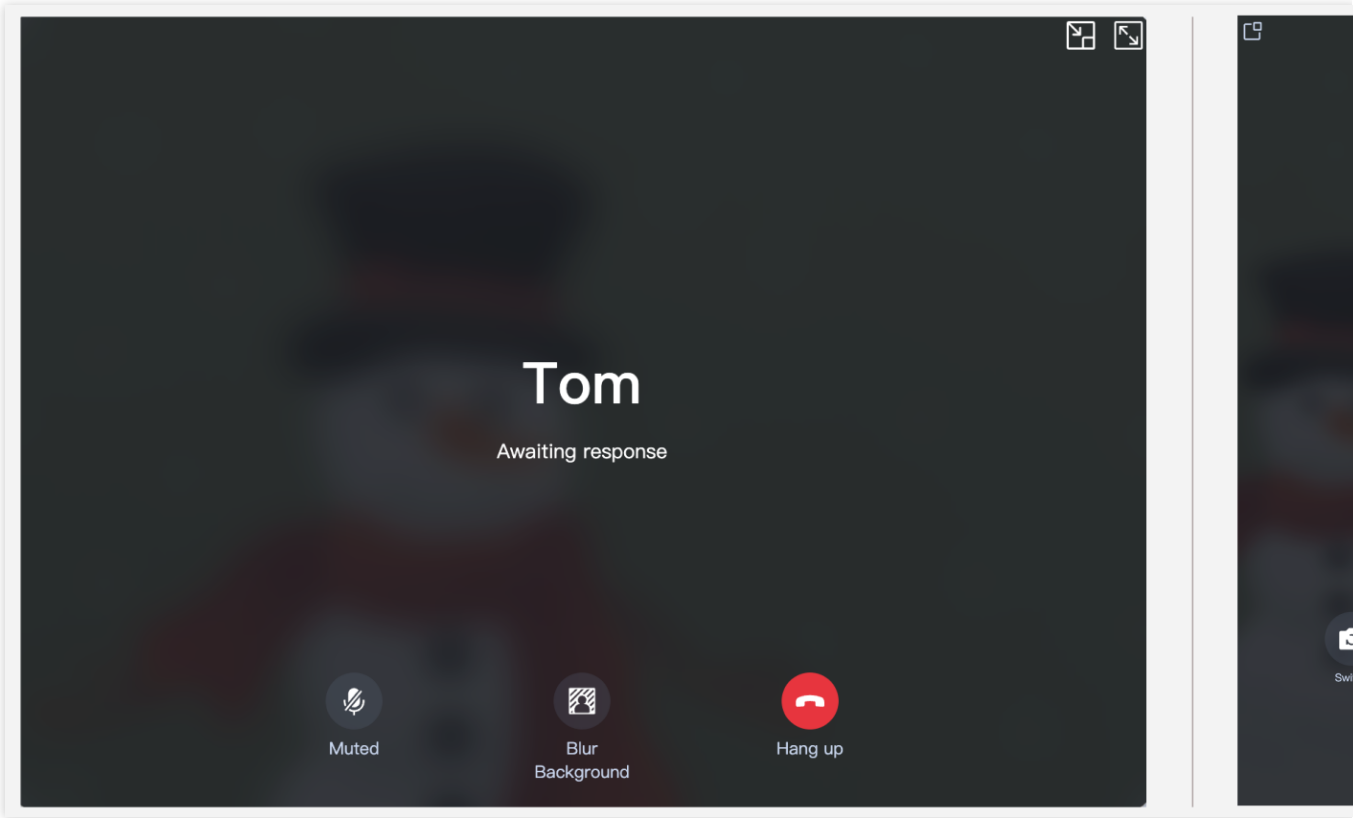
Button Hiding

Invoke the hideFeatureButton interface to hide buttons, currently supporting Camera, Microphone, SwitchCamera, InviteUser. For details, see the enumeration type FeatureButton.

Note:

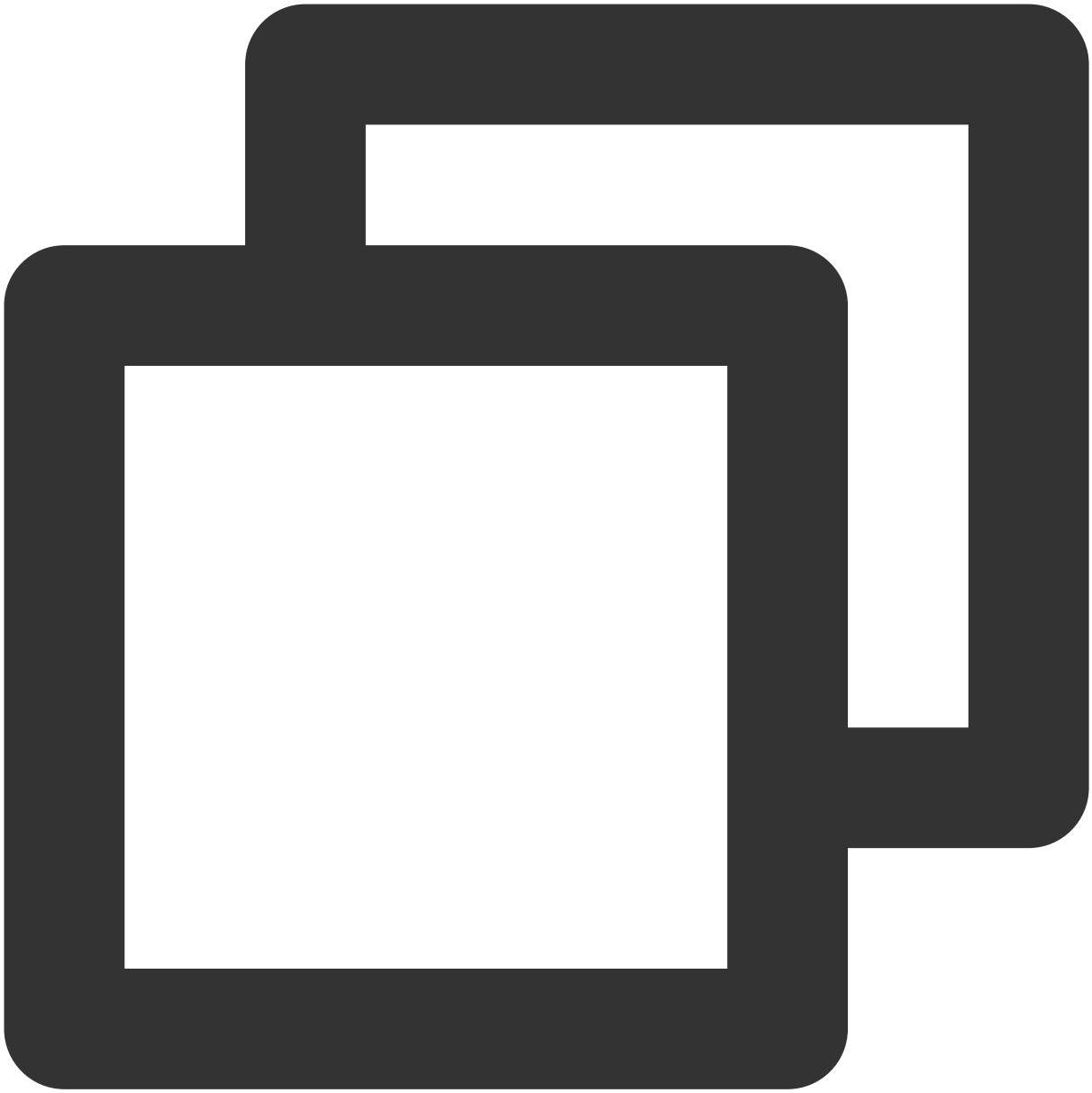
v3.2.9+ support.

Taking the hiding of the **Camera Button** as an example.

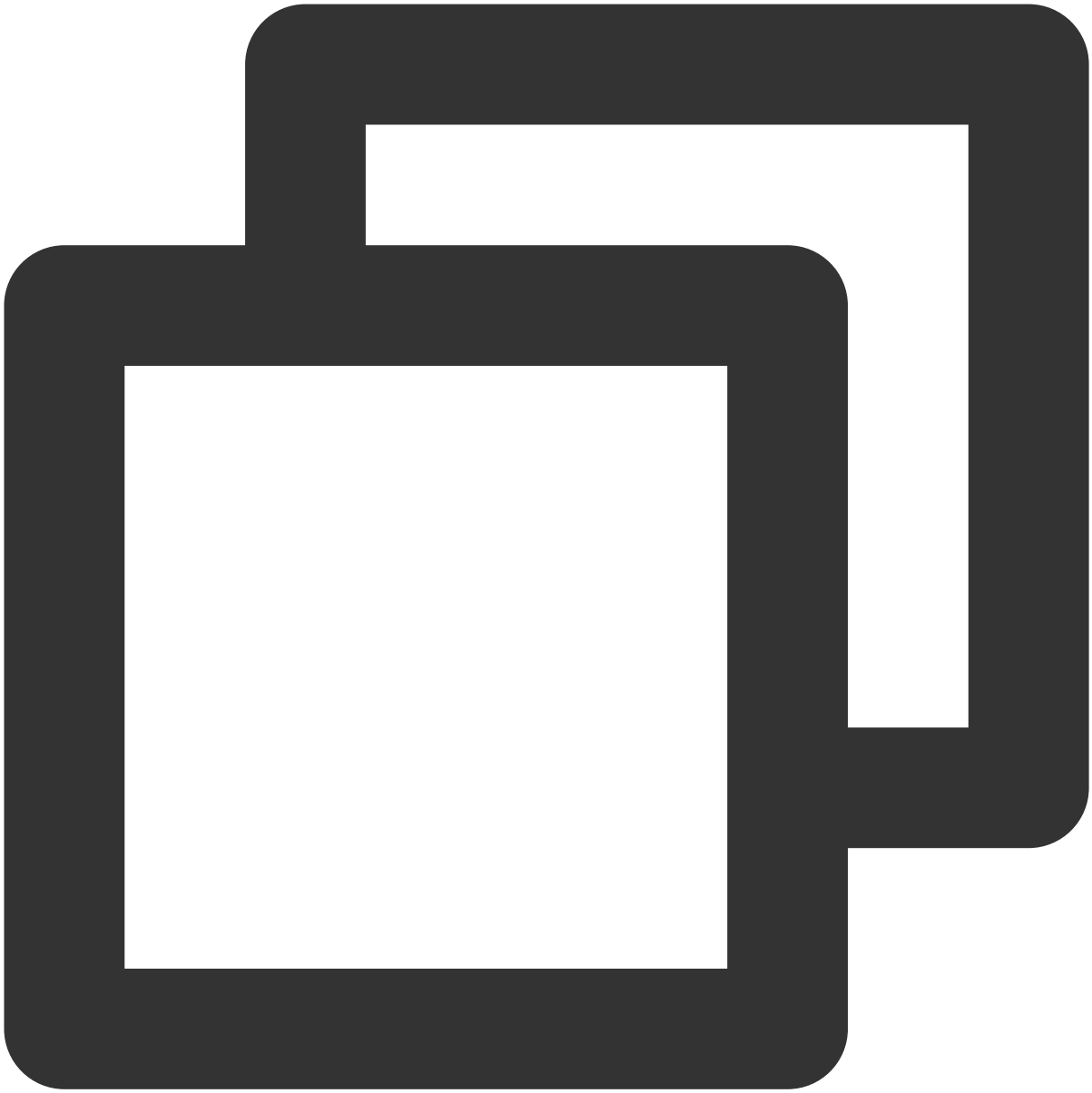


Vue3

React



```
import { TUICallKitServer, FeatureButton } from "@tencentcloud/call-uikit-vue";

TUICallKitServer.hideFeatureButton(FeatureButton.Camera);
```



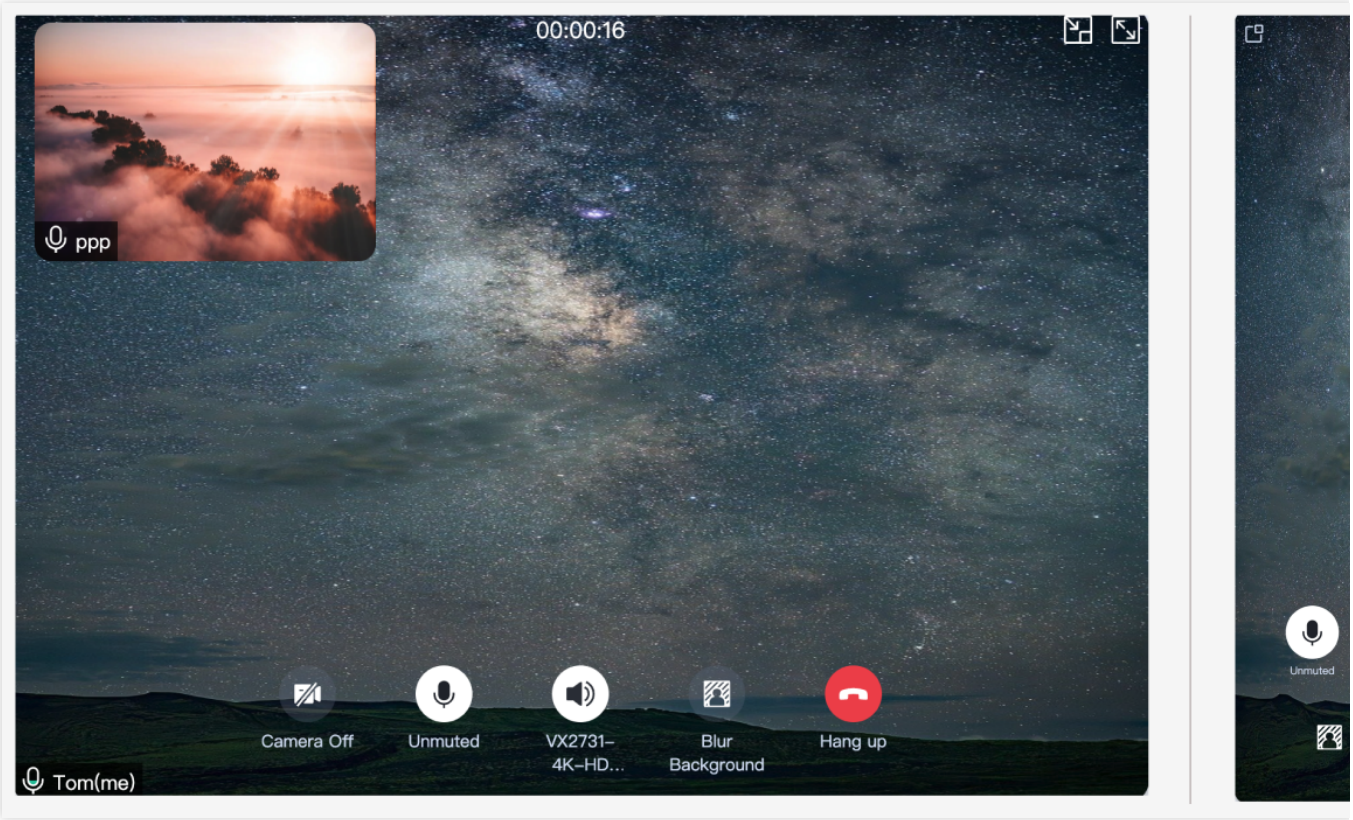
```
import { TUICallKitServer, FeatureButton } from "@tencentcloud/call-uikit-react";

TUICallKitServer.hideFeatureButton(FeatureButton.Camera);
```

Custom Call Background Image

The call background image appears when the camera is turned off during a voice or video call. Modify the local user's call interface background image by calling setLocalViewBackgroundImage, and modify the remote user's call interface background image with setRemoteViewBackgroundImage.

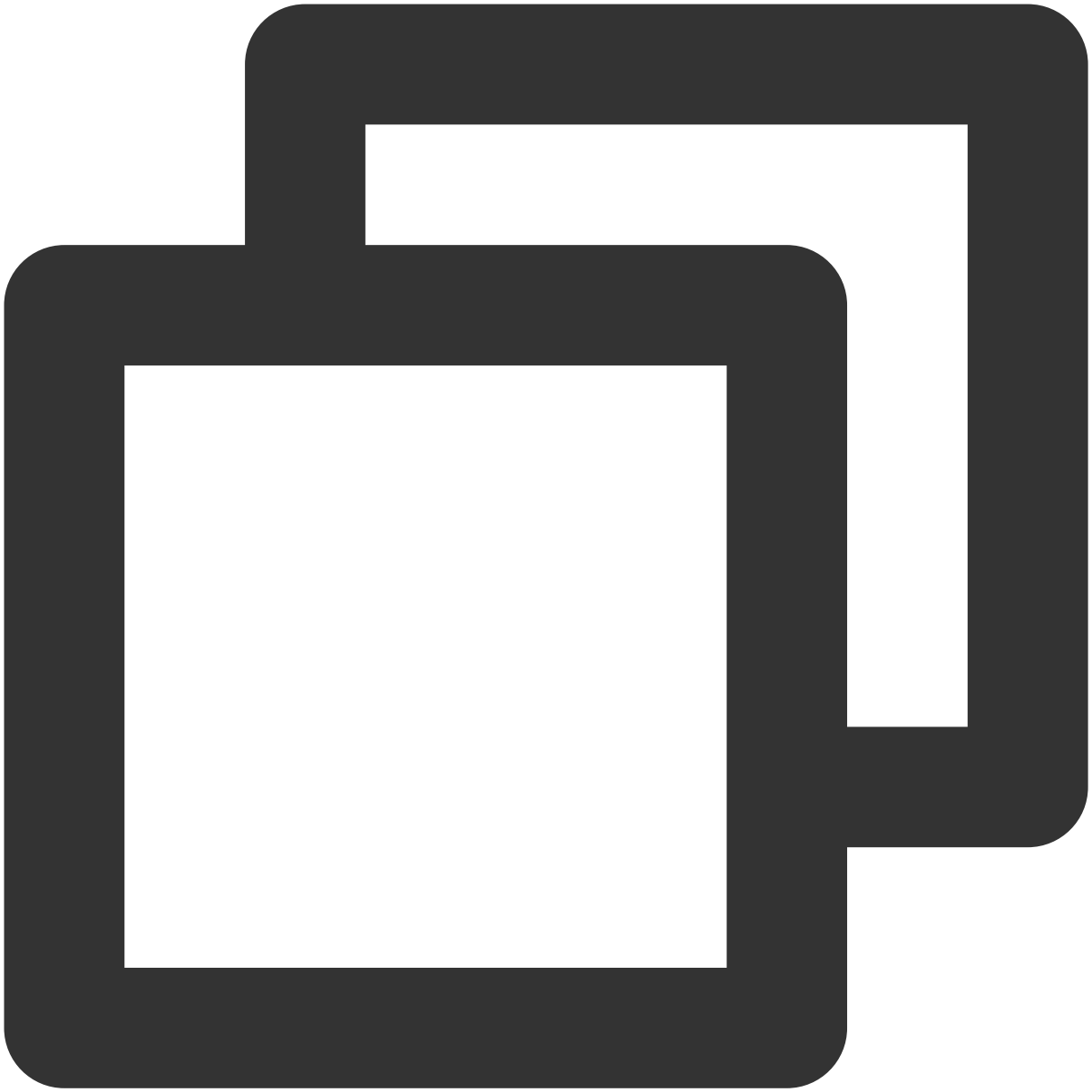
Note:

v3.2.9+ support.

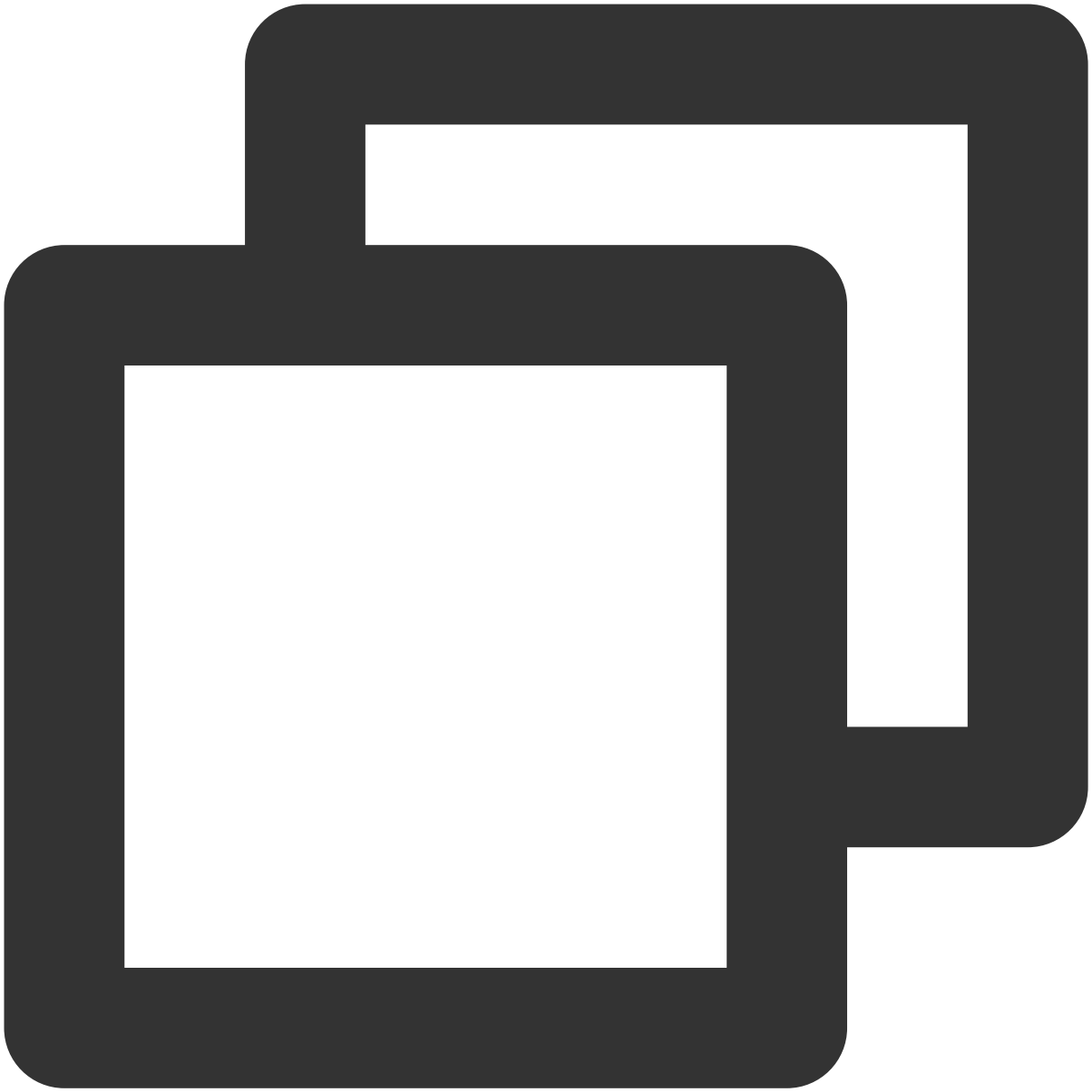


Vue3

React



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-vue";

TUICallKitServer.setLocalViewBackgroundImage('http://xxx.png');
TUICallKitServer.setRemoteViewBackgroundImage('remoteUserId', 'http://xxx.png');
```



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";

TUICallKitServer.setLocalViewBackgroundImage('http://xxx.png');
TUICallKitServer.setRemoteViewBackgroundImage('remoteUserId', 'http://xxx.png');
```

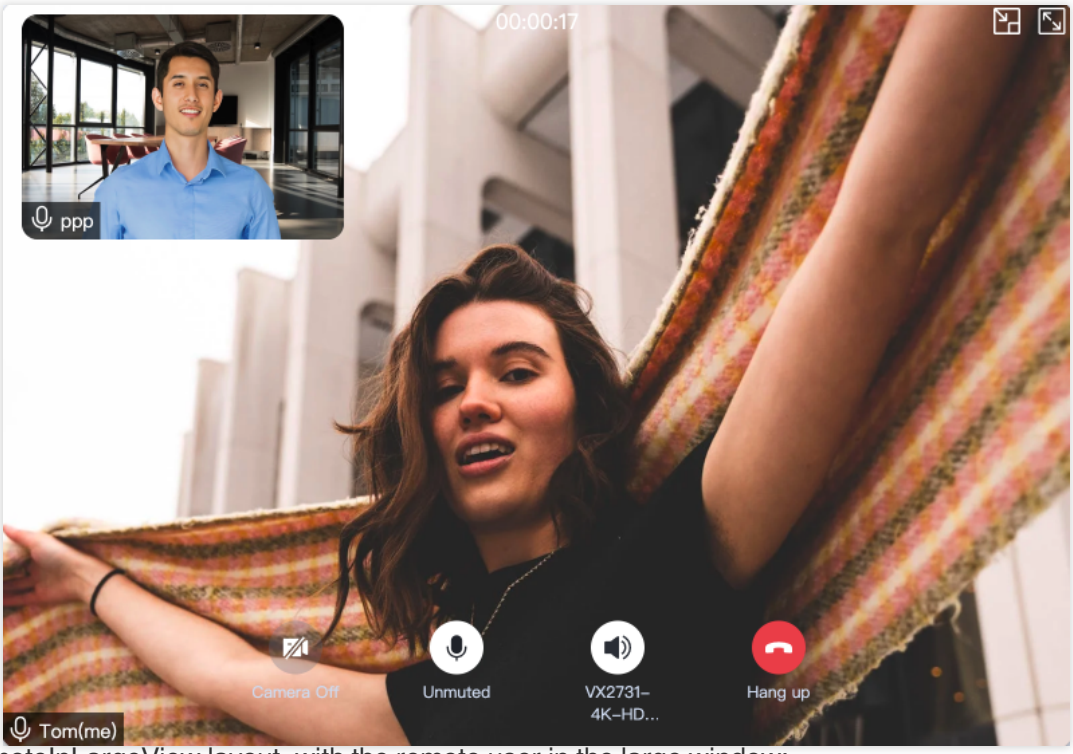
Set Layout

Note：

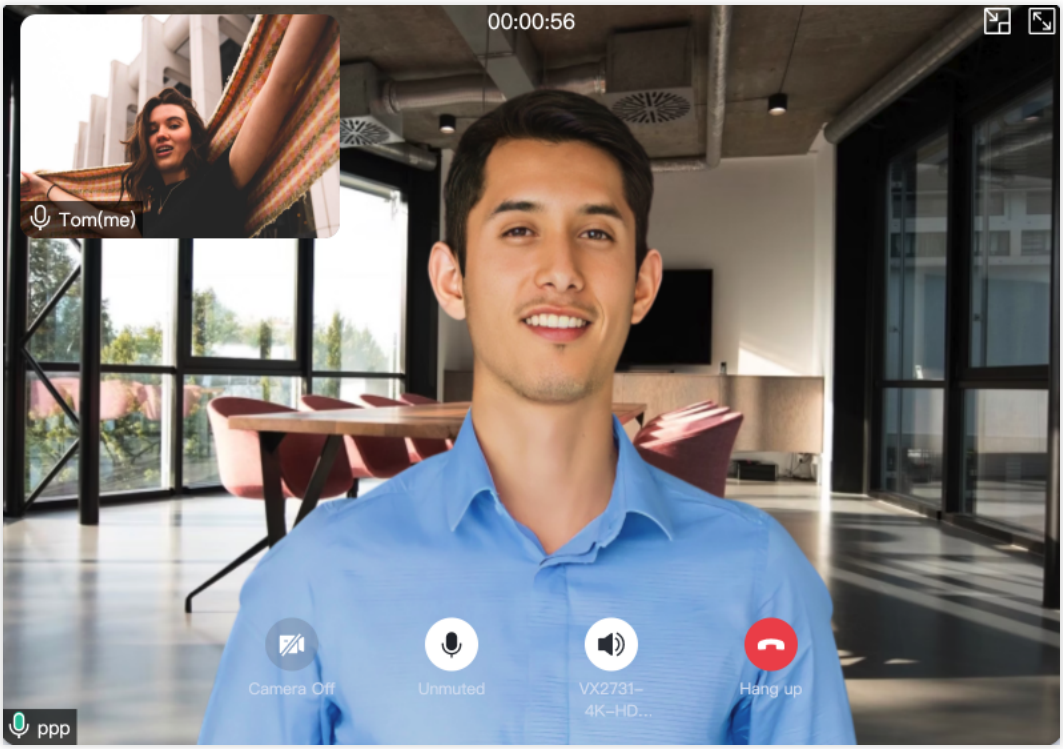
Only available for 1V1 video calls, supported from v3.3.0+.

Use setLayoutMode to set the call interface layout, currently only supports LocalInLargeView and RemoteInLargeView, see the LayoutMode enum for details.

1. LocalInLargeView layout, with the local user in the large window:

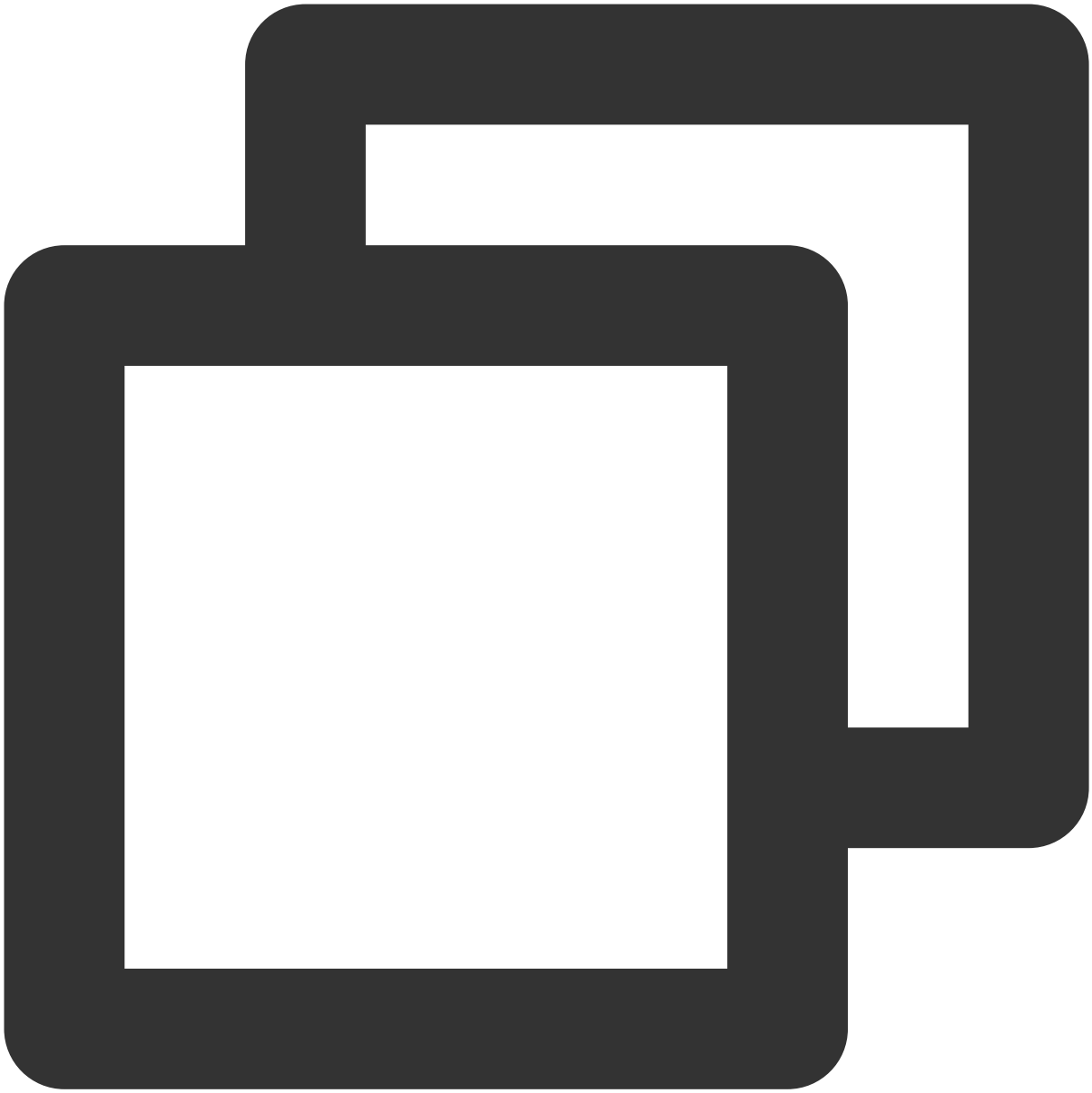


2. RemoteInLargeView layout, with the remote user in the large window:

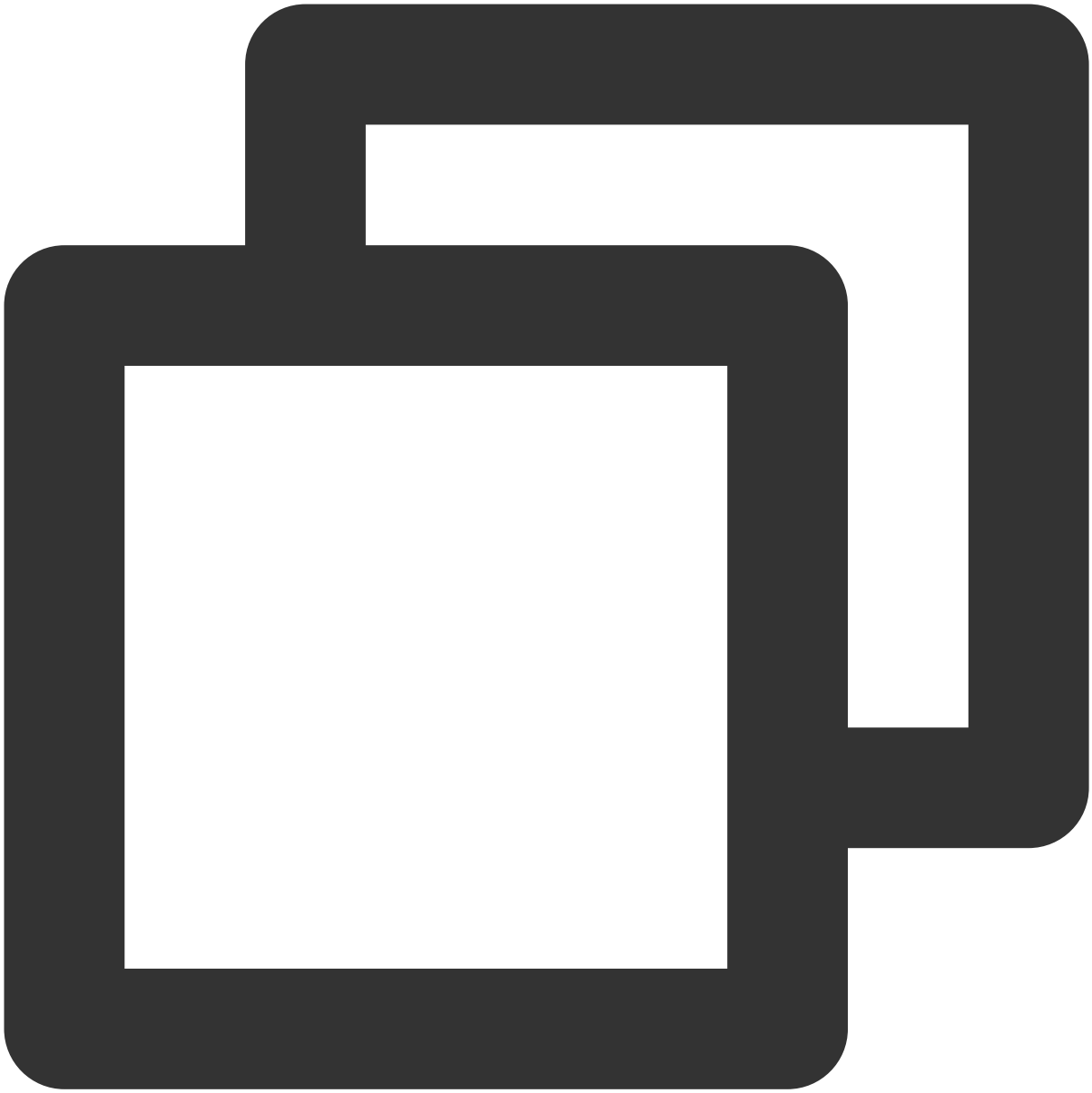


Vue3

React



```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-vue";

TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```



```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-react";

TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```

Set the initial state of the camera

Note：

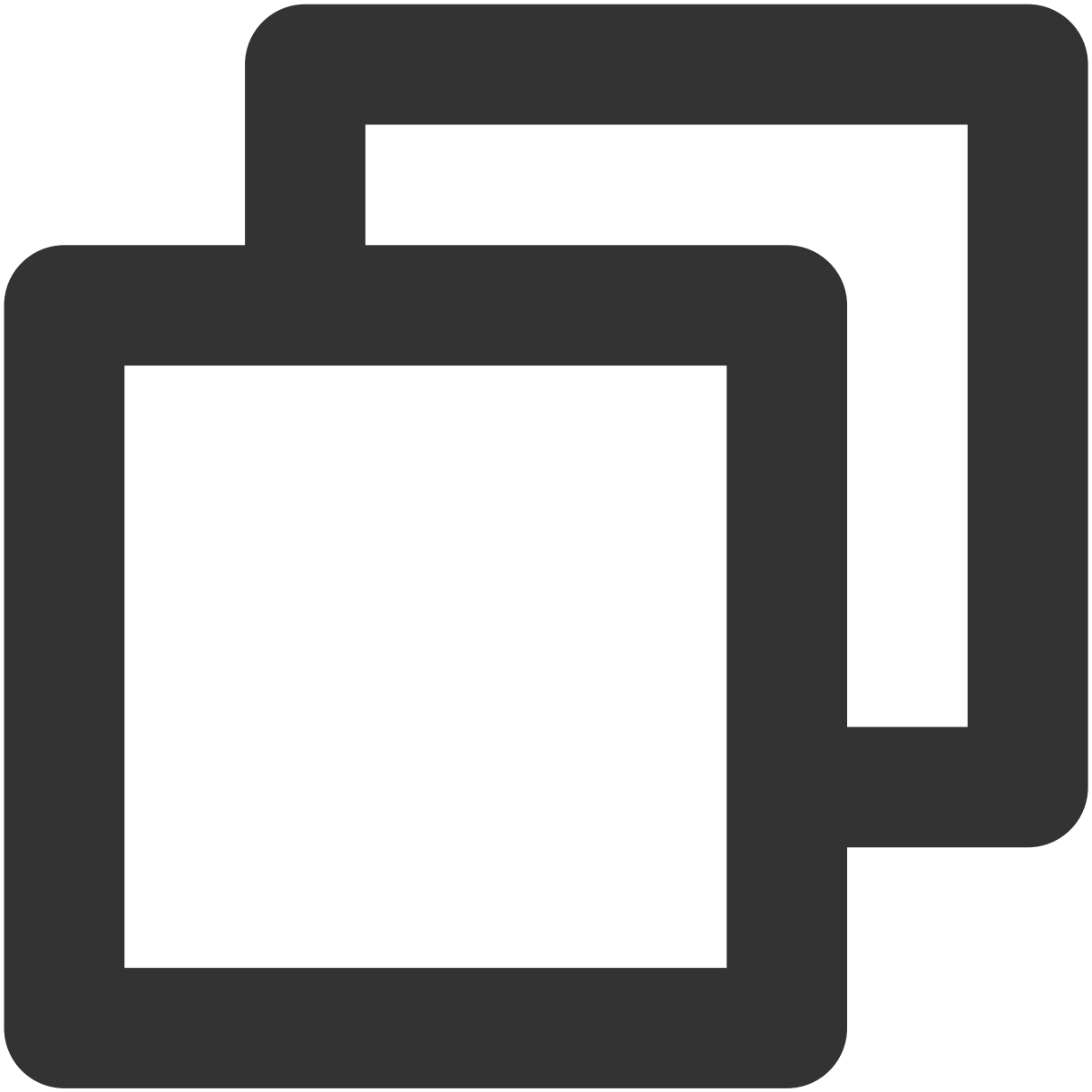
Supported from v3.3.0+.

Use setCameraDefaultState to set the initial state of the camera button, currently supports Enabled and Off.

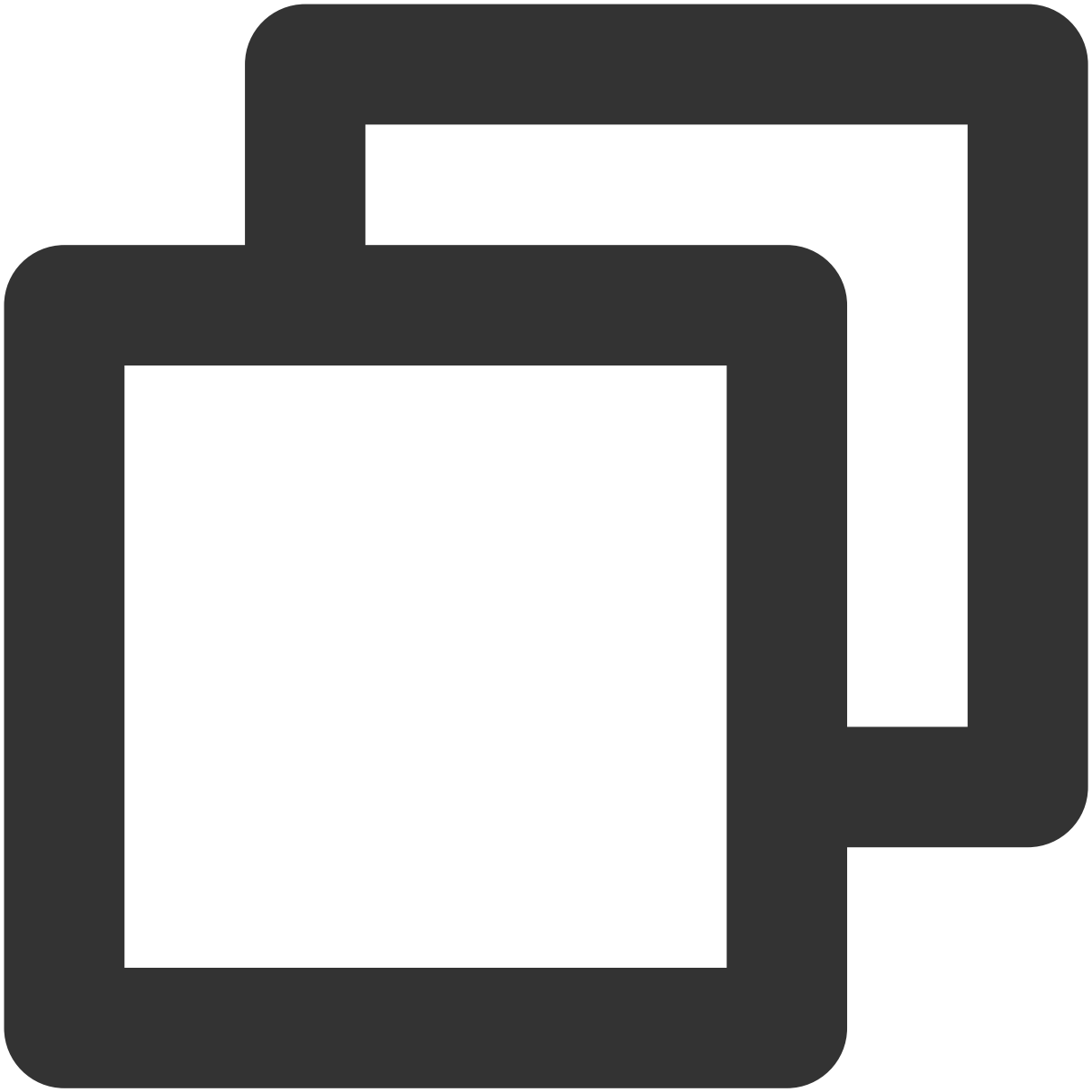
Taking the default Off state of the camera as an example:

Vue3

React



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-vue";

TUICallKitServer.setCameraDefaultState(false);
```



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";

TUICallKitServer.setCameraDefaultState(false);
```

Replacing icons

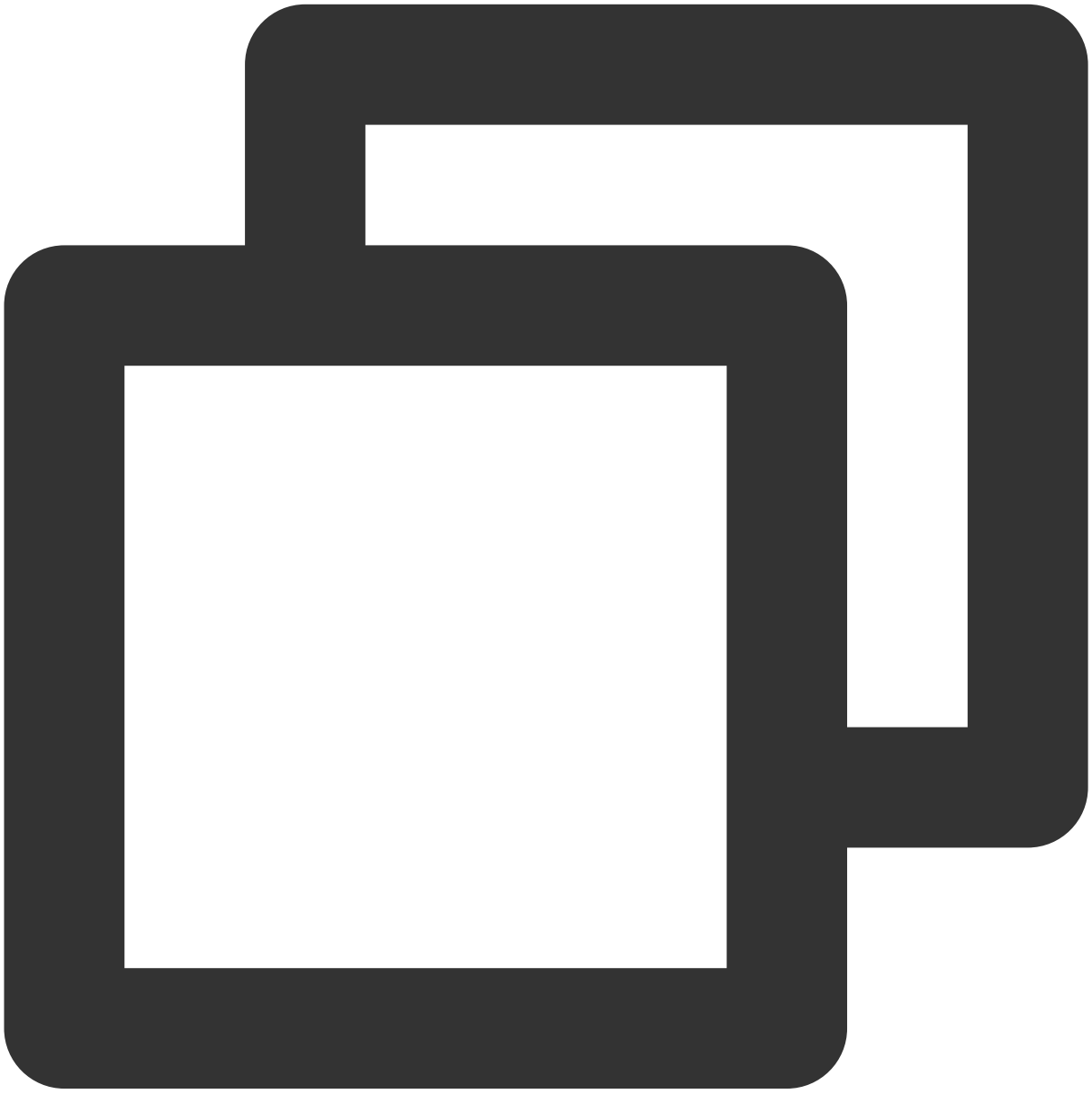
To replace an icon, source code import is required first. Copy the component to your project (the source code is in TypeScript version).

Note：

The Interface Replacing icons Plan is suitable for `Vue3 + TypeScript` and `@tencentcloud/call-uikit-vue` version number is 3.2.2 or later projects. If you are using other languages or technology stacks, please use the Custom UI Implementation.

1. **Download Source Code**

Vue3

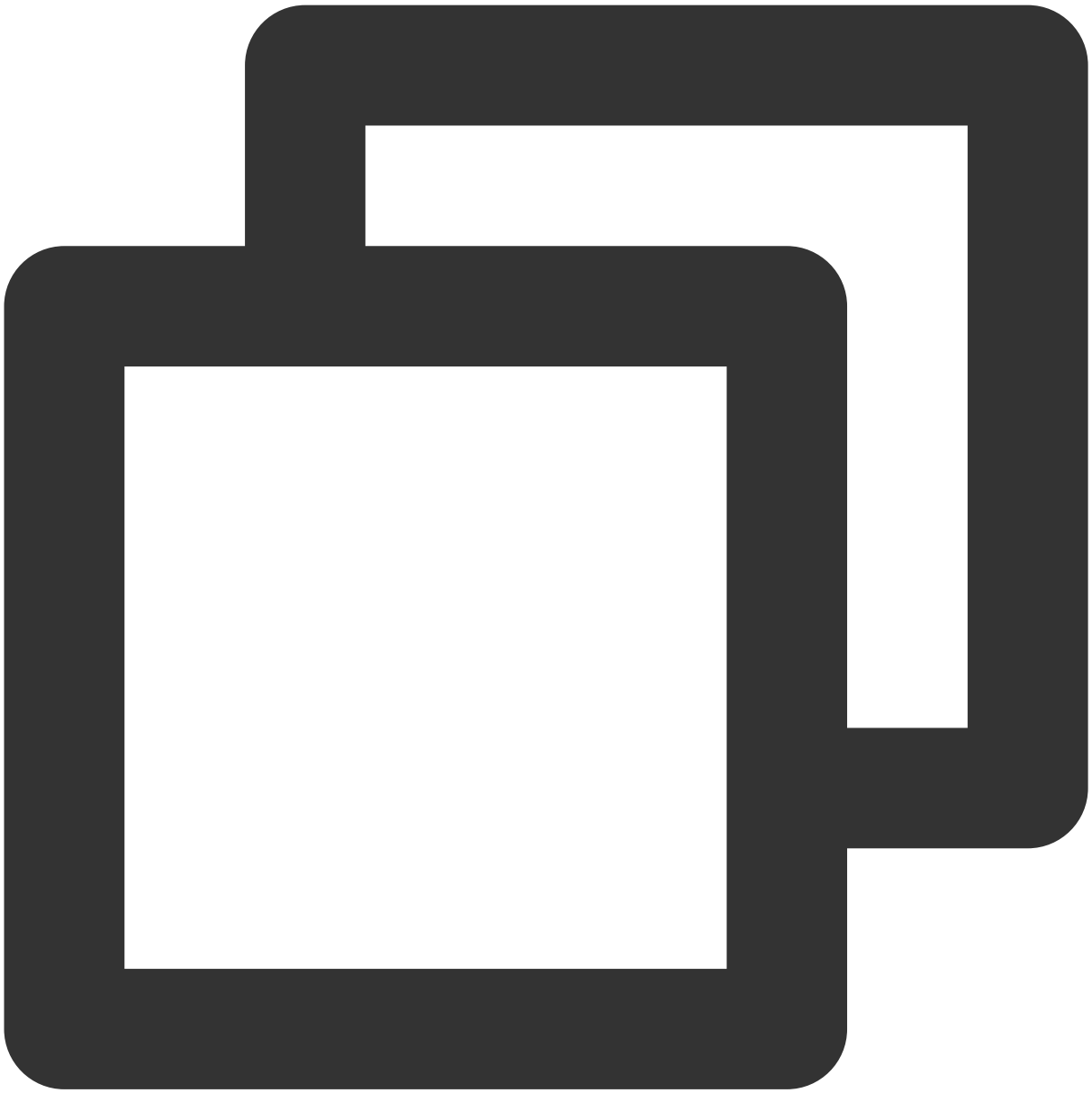


```
npm install @tencentcloud/call-uikit-vue
```

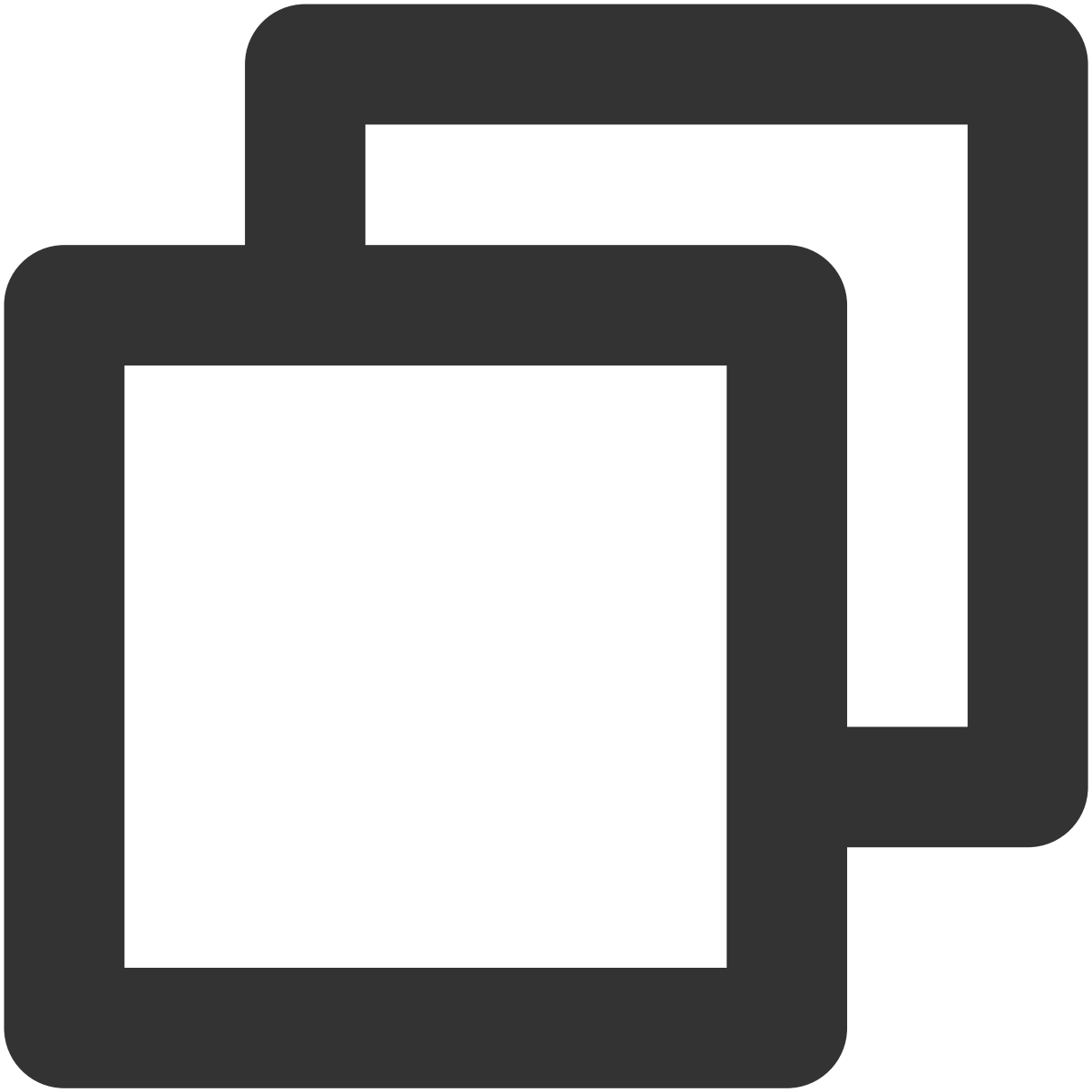
2. **Copy the source code into your own project, taking copying into the `src/components/` directory as an example:**

macOS + Vue3

Windows + Vue3



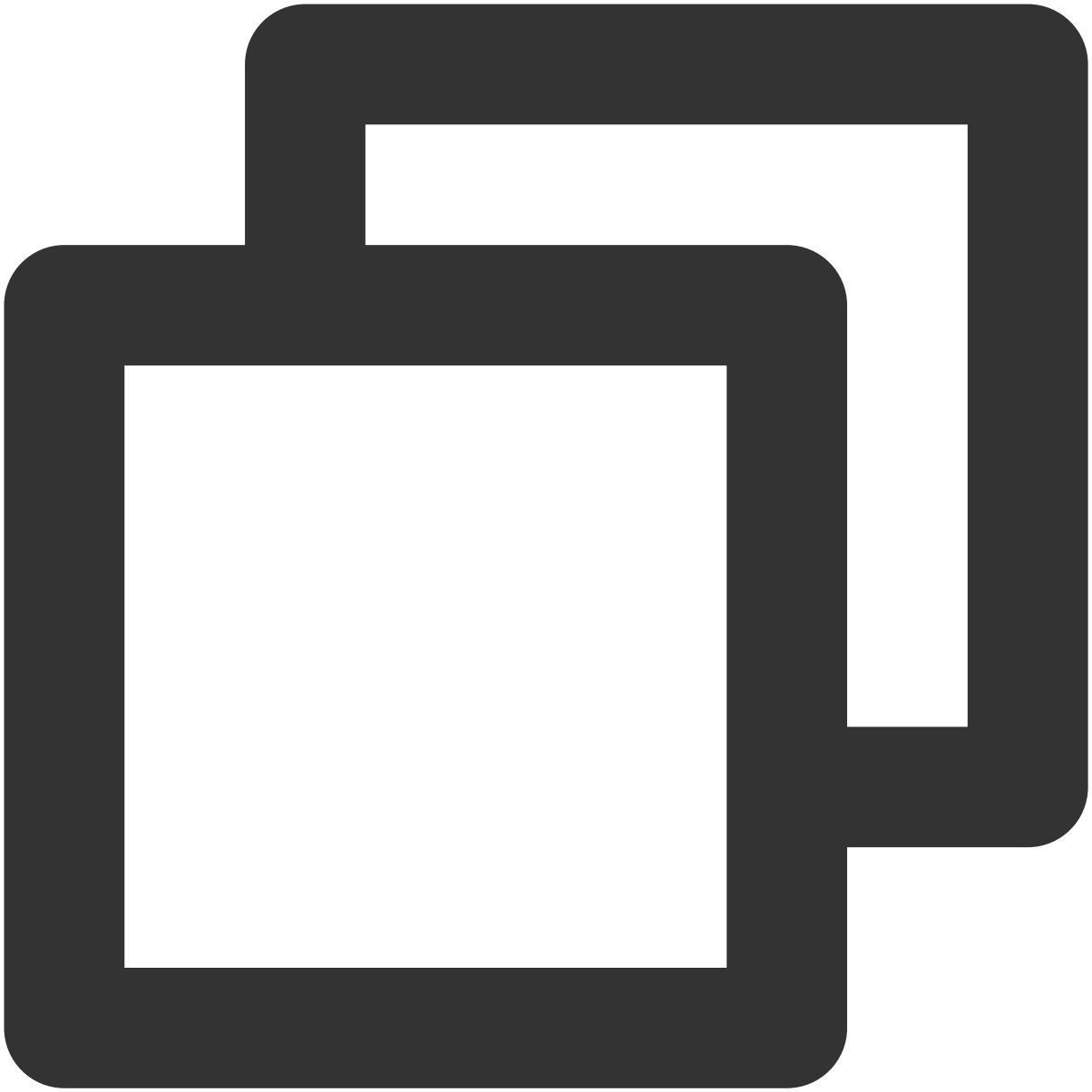
```
mkdir -p ./src/components/TUICallKit && cp -r ./node_modules/@tencentcloud/call-uik
```



```
xcopy .\\node_modules\\@tencentcloud\\call-uikit-vue  .\\src\\components\\TUICallKi
```

3. **Modify Import Path**

It's necessary to change CallKit to be imported from a local file, as shown below. For other usage details, refer to TUICallKit Quick Integration.



```
import { TUICallKit, TUICallKitServer, TUICallType } from "./components/TUICallKit/
```

4.

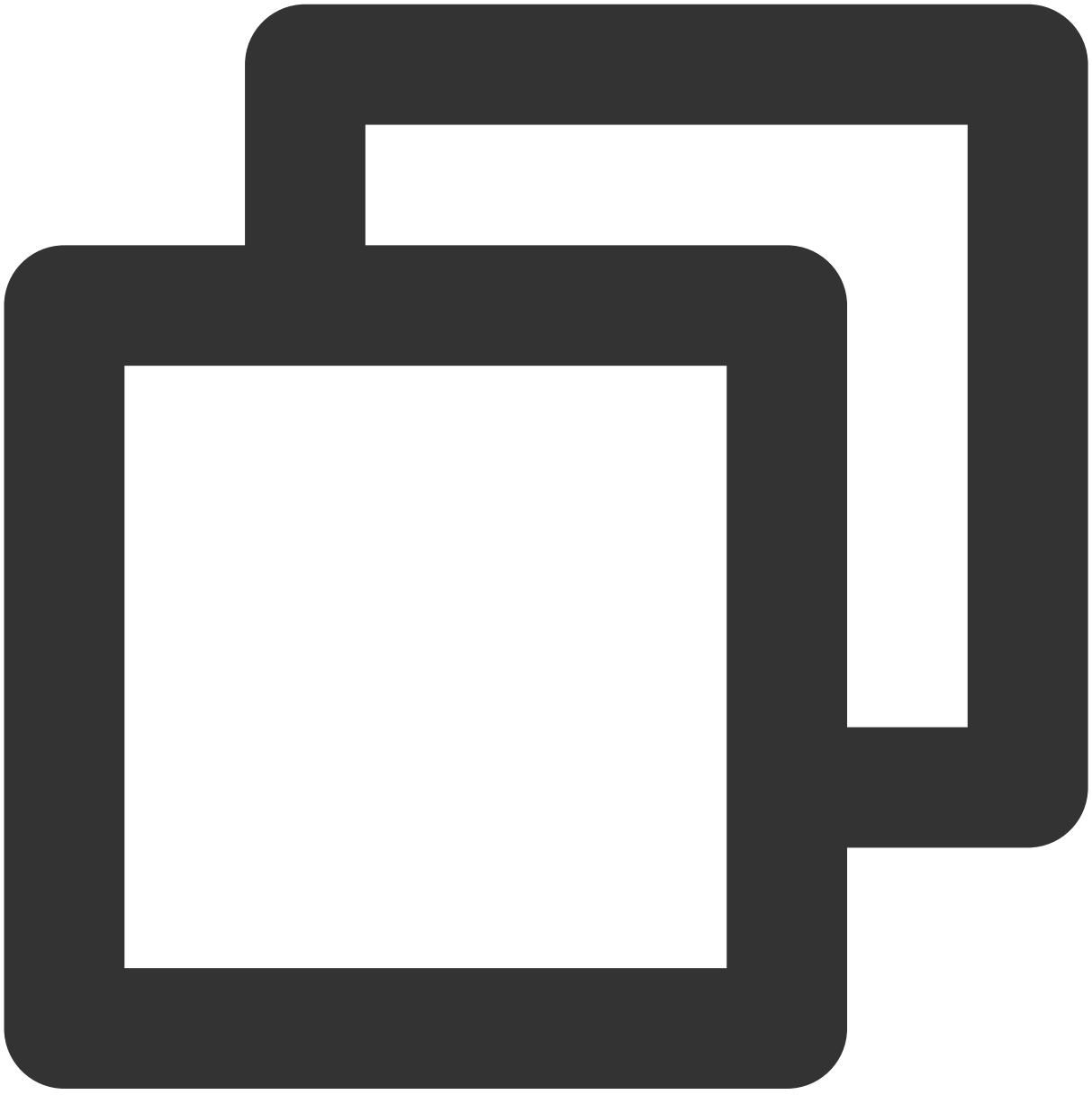
Solve Errors That May Be Caused by Copying Source Code

If you encounter an error while using the TUICallKit component, please don't worry. In most cases, this is due to inconsistencies between ESLint and TSConfig configurations. You can consult the documentation and configure correctly as required. If you need help, please feel free to contact us, and we will ensure that you can successfully use this component. Here are some common issues:

ESLint Error

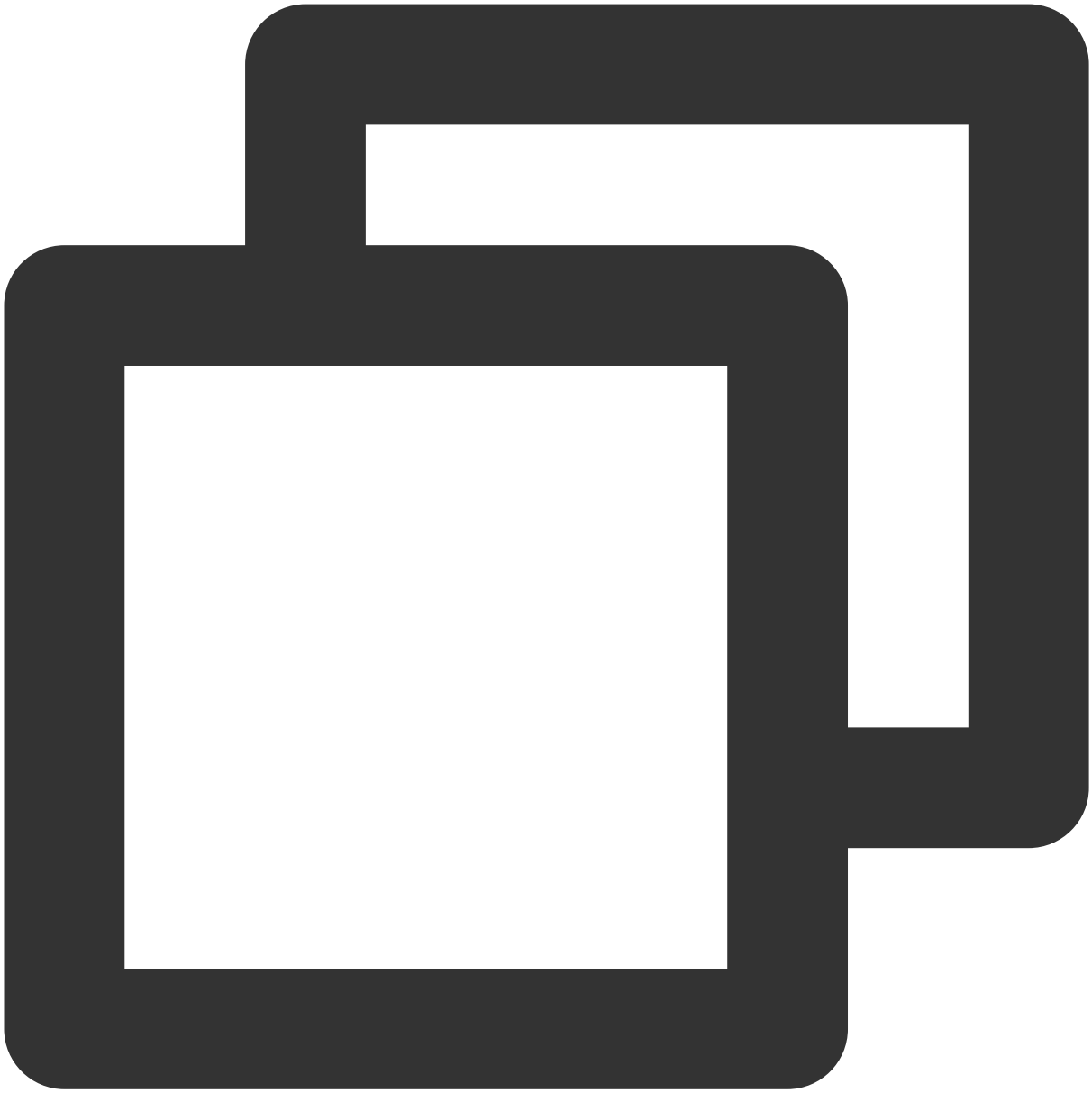
TypeScript Error

If the TUICallKit causes an error due to inconsistency with your project's code style, you can block this component directory by adding a `.eslintignore` file in the root directory of your project, for example:



```
# .eslintignore
src/components/TUICallKit
```

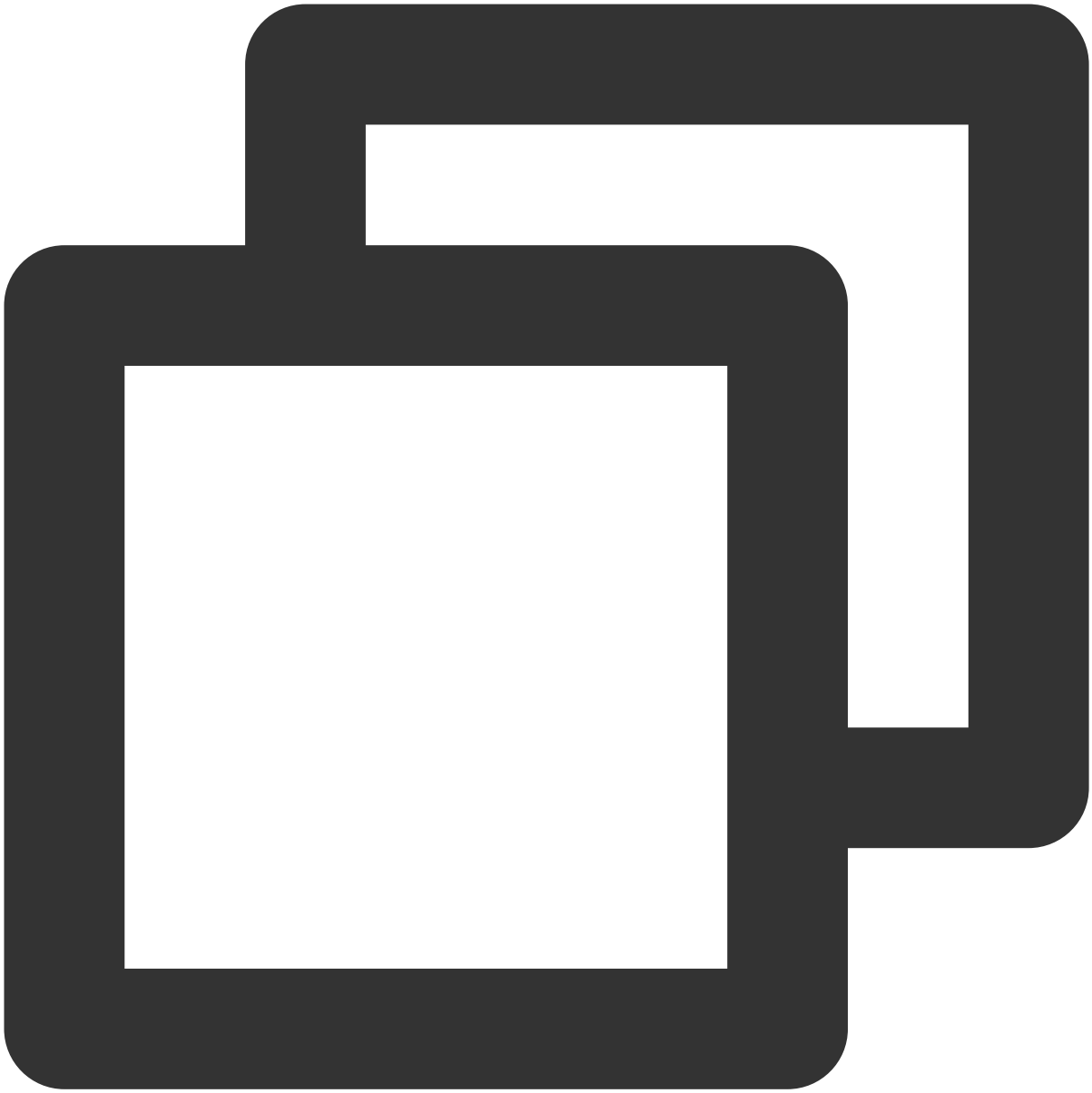
1. If you encounter the 'Cannot find module '../package.json'' error, it's because TUICallKit references a JSON file. You can add the related configuration in tsconfig.json, example:



```
{
  "compilerOptions": {
    "resolveJsonModule": true
  }
}
```

For other TSConfig issues, please refer to TSConfig Reference.

2. If you encounter the 'Uncaught SyntaxError: Invalid or unexpected token' error, it's because TUICallKit uses decorators. You can add the related configuration in tsconfig.json, example:



```
{
  "compilerOptions": {
    "experimentalDecorators": true
  }
}
```

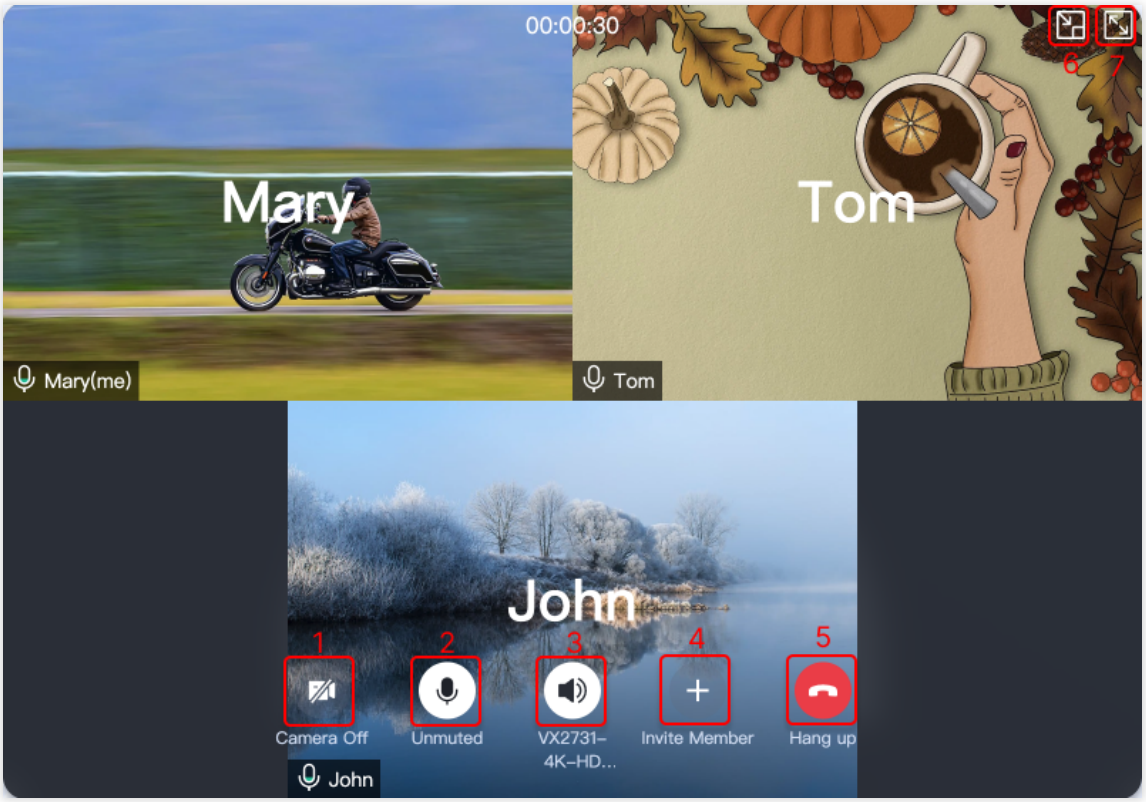
5. **Modify the icon components in the TUICallKit/Components/assets folder**

Note:

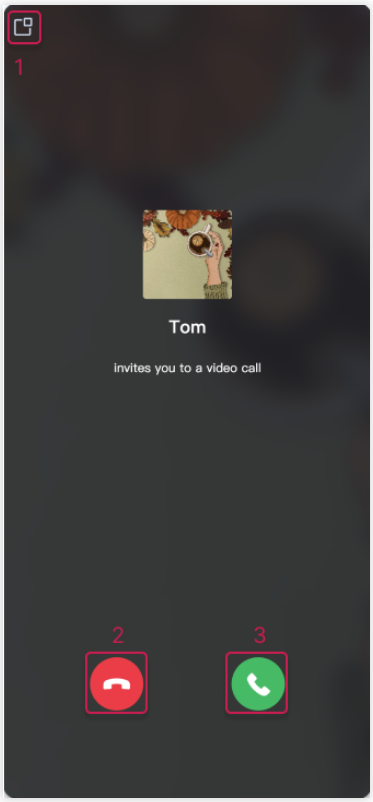
To ensure the icon color and style remain consistent throughout the application, please keep the icon file name unchanged when replacing.

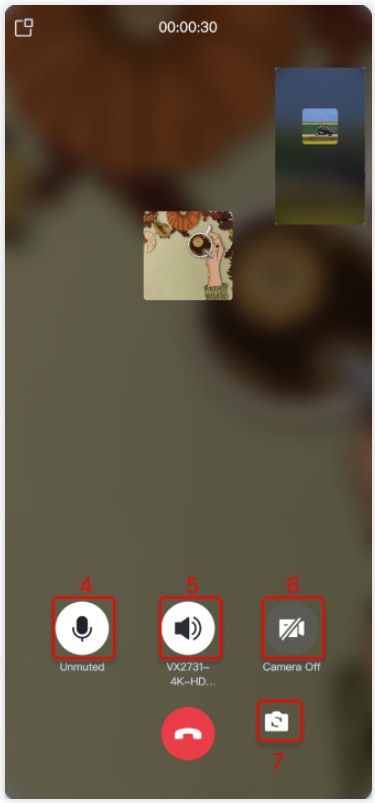
Desktop

Mobile



Serial number	Resource Path
1	/TUICallKit/Components/assets/button/camera-close.svg
2	/TUICallKit/Components/assets/button/microphone-open.svg
3	/TUICallKit/Components/assets/button/speaker-open.svg
4	/TUICallKit/Components/assets/button/desktop/inviteUser.svg
5	/TUICallKit/Components/assets/button/hangup.svg
6	/TUICallKit/Components/assets/button/desktop/minimize.svg
7	/TUICallKit/Components/assets/button/desktop/fullScreen.svg





Serial number	Resource Path

1	/TUICallKit/Components/assets/button/mobile/minimize.svg
2	/TUICallKit/Components/assets/button/hangup.svg
3	/TUICallKit/Components/assets/button/accept.svg
4	/TUICallKit/Components/assets/button/microphone-open.svg
5	/TUICallKit/Components/assets/button/speaker-open.svg
6	/TUICallKit/Components/assets/button/camera-close.svg
7	/TUICallKit/Components/assets/button/switchCamera.svg

Replacing ringtones

You can replace ringtones by replacing the three audio files in the `TUICallKit/src/TUICallService/assets/` folder.

Filename	Description
phone_dialing.mp3	The sound of making a call
phone_ringing.mp3	The ringtone for incoming calls

Scheme 2: Custom UI Implementation

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

TUICallEngine integration guide

TUICallEngine APIs

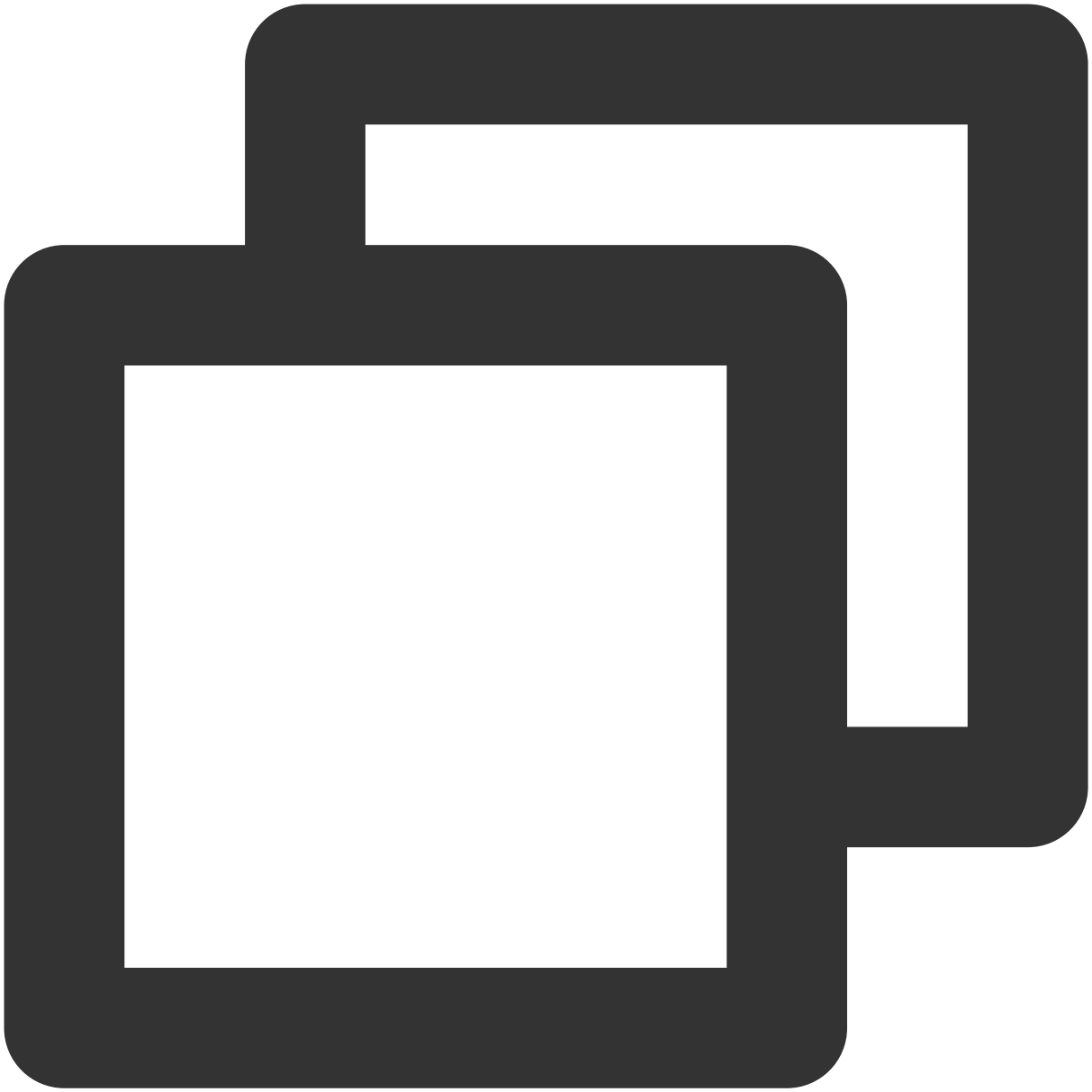
Flutter

Last updated：2024-03-12 14:51:07

This article will introduce how to customize the user interface of TUICallKit. We provide two solutions for you to choose: **interface fine-tuning solution** and **self-implementation UI** solution.
Note: The page customization solution needs to use the tencent_calls_uikit plugin version 1.8.0 or later.

Scheme 1. Slight UI Adjustment

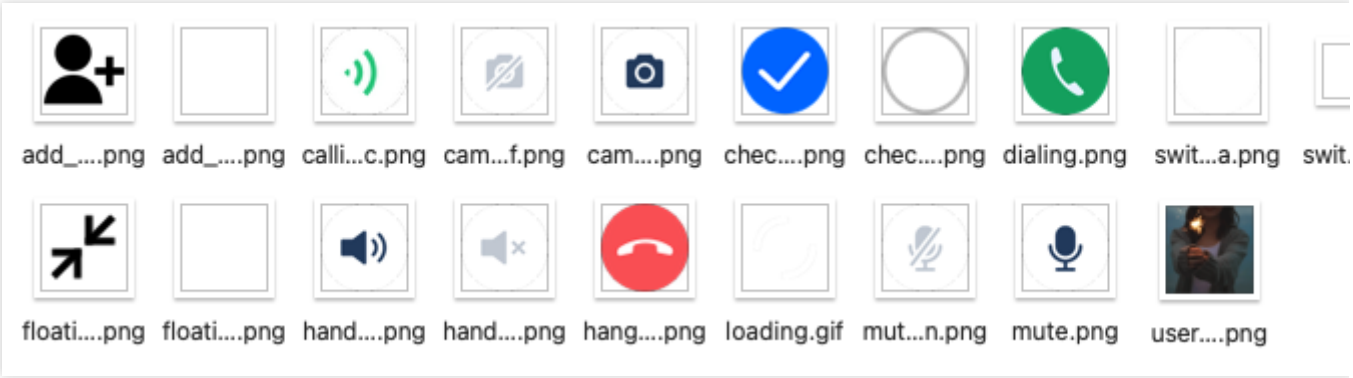
You can download the latest version of the tencent_calls_uikit plugin locally, and then use the local dependency method to access the plugin in your project. The local dependency method is as follows:
Under the **dependencies** node in the project **pubspec.yaml** file, add the **tencent_calls_uikit** plugin dependency, as shown below:



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

replace icon

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



replace ringtone

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

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

Replacing text

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

Scheme 2. Custom UI Implementation

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

TUICallEngine

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

TUICallObserver

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

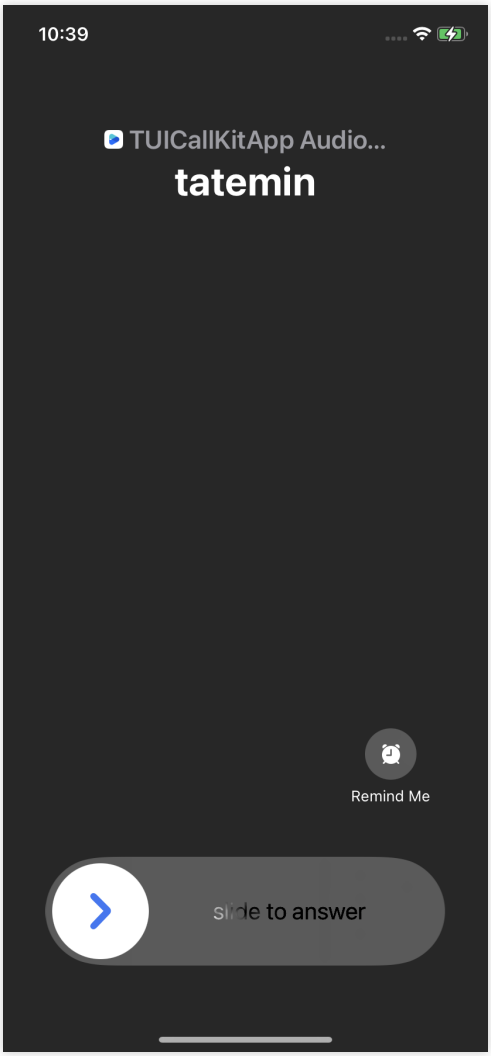
Offline Call Push (TUICallKit)
iOS
VoIP

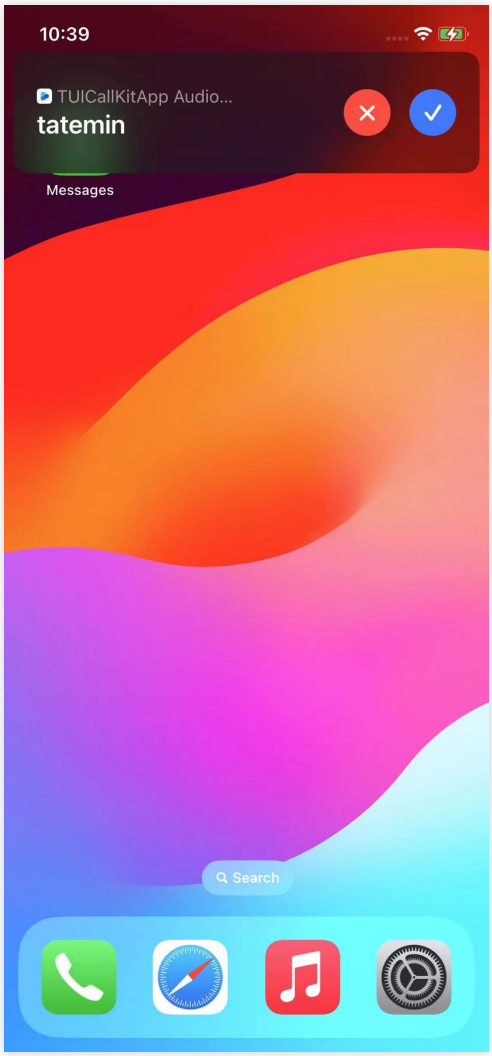
Last updated：2024-07-19 18:07:32

VoIP (Voice over IP) Push is a notification mechanism provided by Apple for responding to VoIP calls. Combining Apple's PushKit.framework and CallKit.framework can achieve system-level call effects.
Since the TIMPush plugin currently does not support VoIP notifications on iOS, we provide you with a free alternative solution to help you achieve VoIP push on iOS devices.
Note:
Apple requires that VoIP Push be used in conjunction with the CallKit.framework starting with iOS 13.0; otherwise, the app will crash after running.
VoIP Push cannot reuse the general APNs push certificates, so a separate VoIP Push certificate needs to be applied for on the Apple Developer website.

Integration effect

Lock screen effects	Effects of Applications in the Background	





Configuring VoIP Push

To receive VoIP Push notifications, follow these steps:

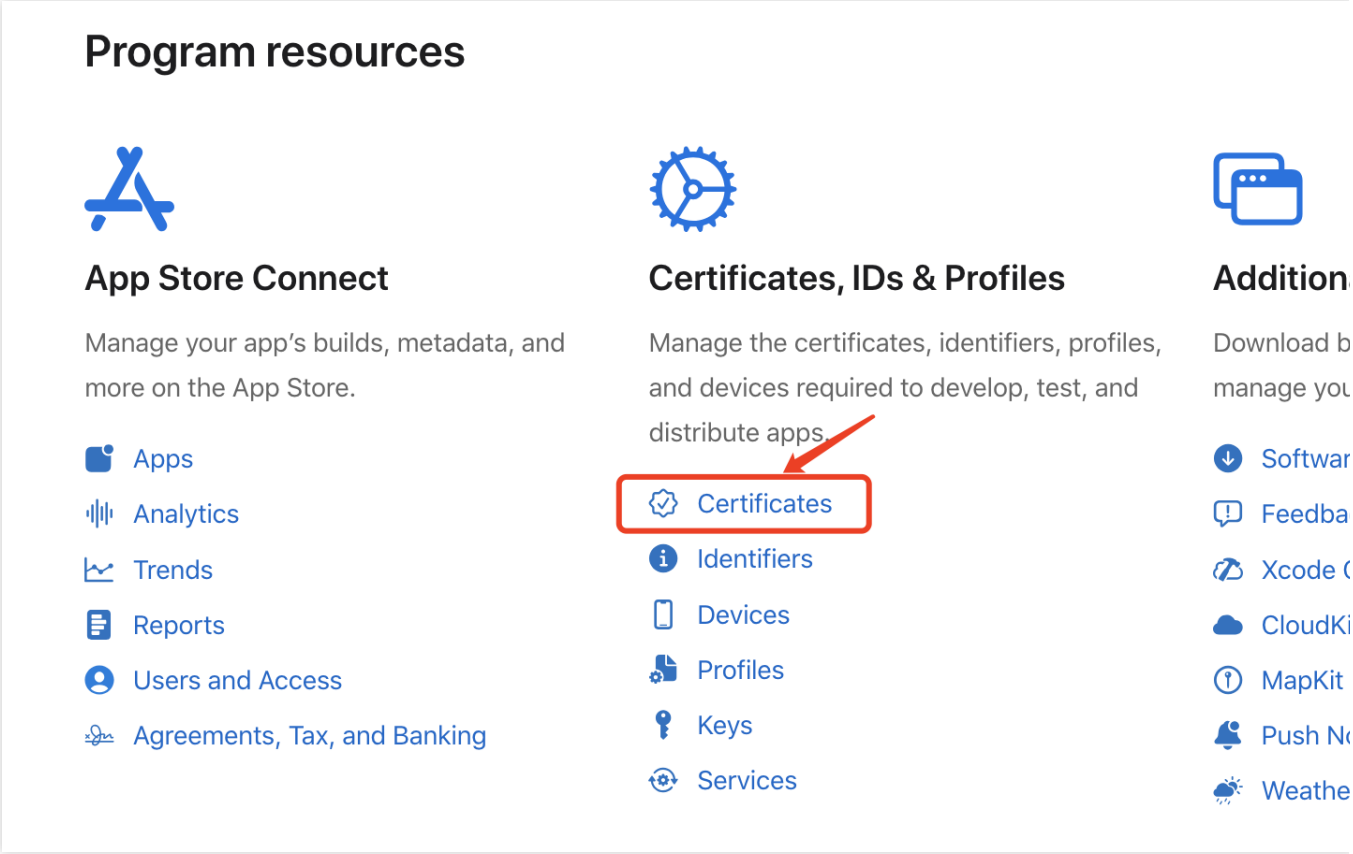
1. Apply for a VoIP Push certificate.
2. Upload the certificate to the Chat console.
3. Complete the project configuration.
4. Integrate the TUICallKitVoIPExtension component.

Step 1: Apply for a VoIP Push certificate

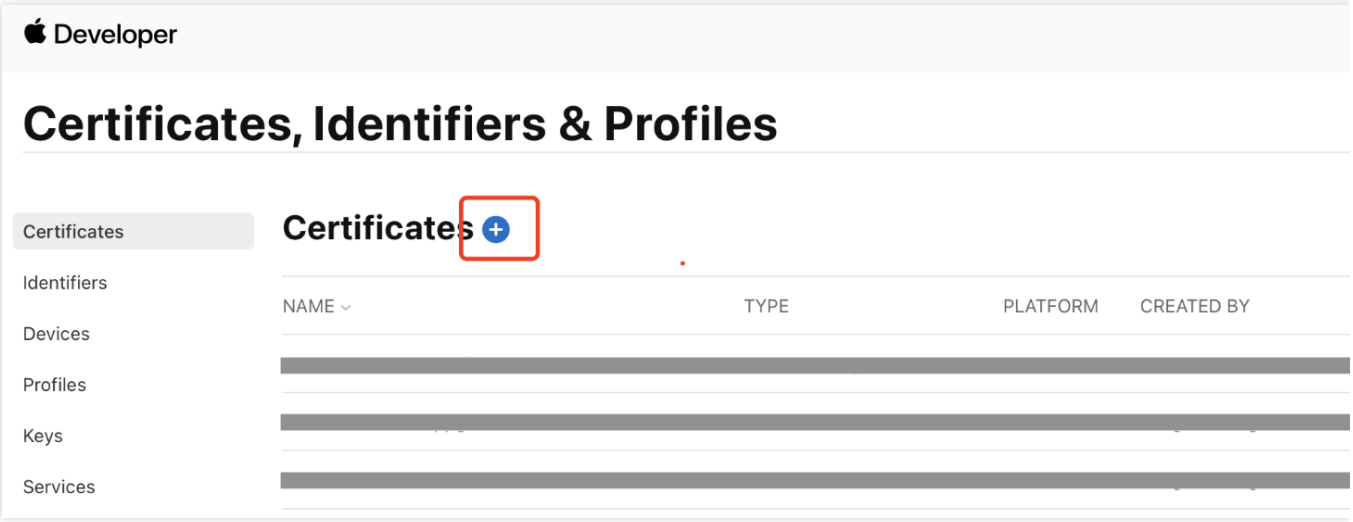
Before applying for a VoIP Push certificate, log in to the Apple Developer Center, enable your app's remote push capabilities.

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

1. Log in to the Apple Developer Center website, click "Certificates, IDs & Profiles" in the sidebar, and go to the "Certificates, Identifiers & Profiles" page.

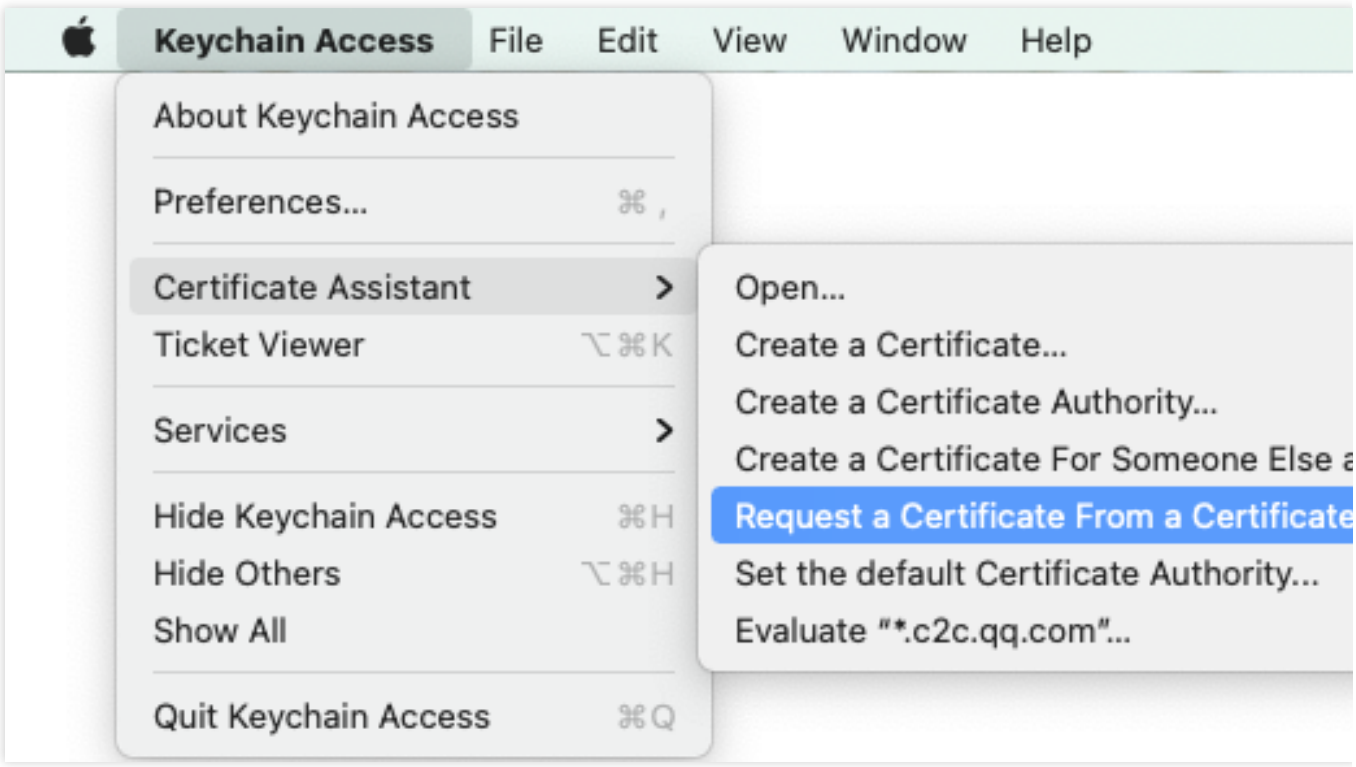


2. Click + next to Certificates.

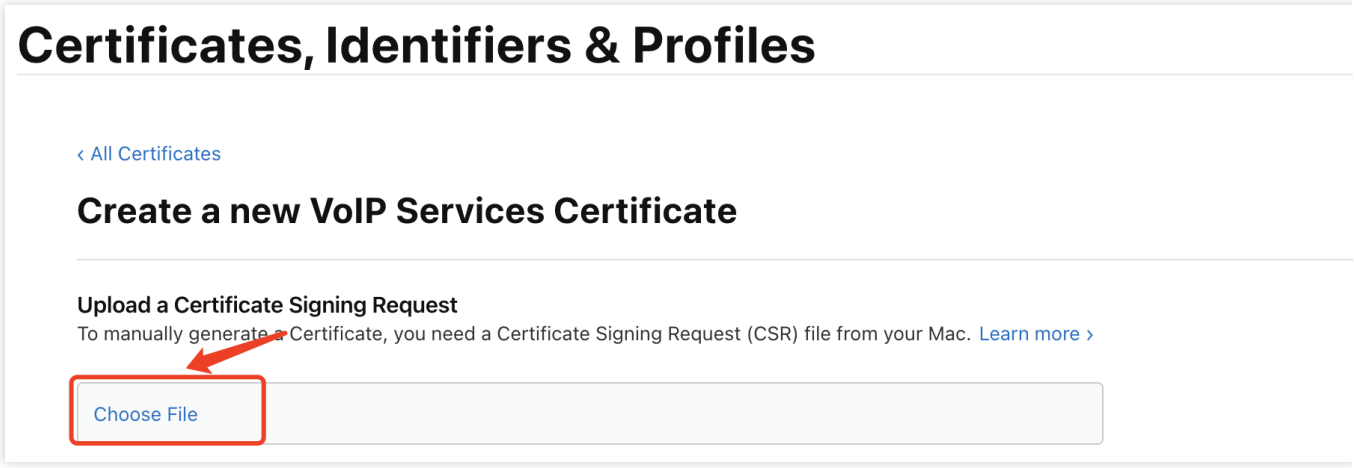


3. In the **Create a New Ceritificate** tab, select **VoIP Services Certificate** and click **Continue**.

4. In the **Select an App ID for your VoIP Service Certificate** tab, select your app's BundleID, and click **Continue**.
5.
The
system will prompt you for a Certificate Signing Request (CSR).
6. Next, create a CSR file. Open **Keychain Access** on your Mac, and in the menu, choose **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority**.



7. Enter your email address, common name (your name or company name), choose **Save to disk**, click **Continue**, and the system will generate a `*.certSigningRequest` file.
Go back to the Apple Developer website in Step5 , click "Choose File" and upload the generated `*.certSigningRequest` file.



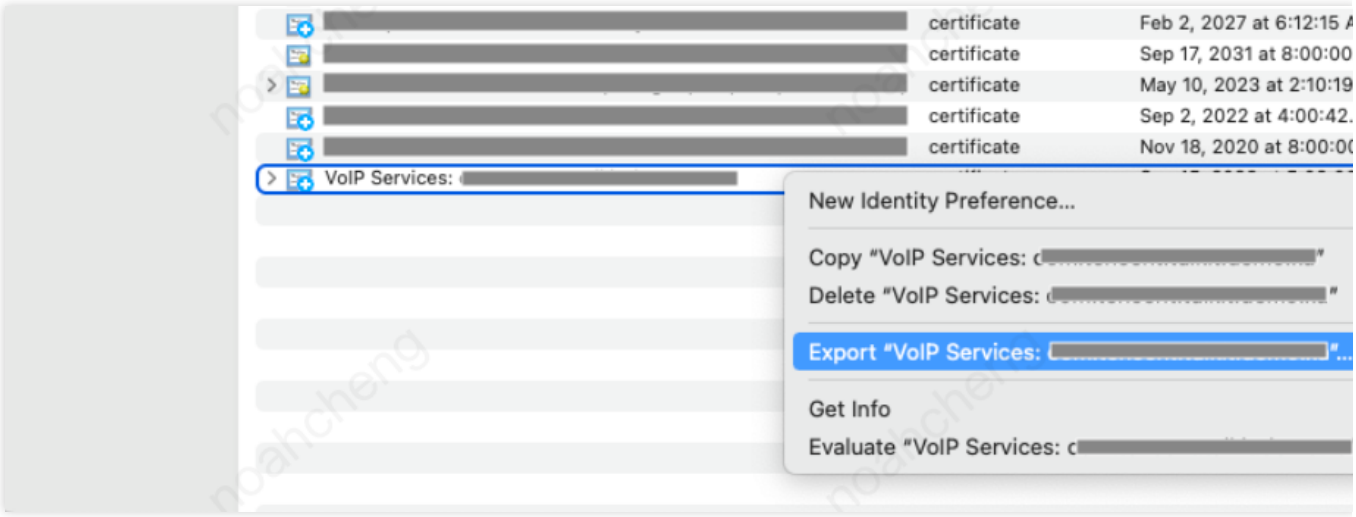
8. Click **Continue** to generate the certificate, and click **Download** to download the corresponding certificate to your local device.

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

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

Note:

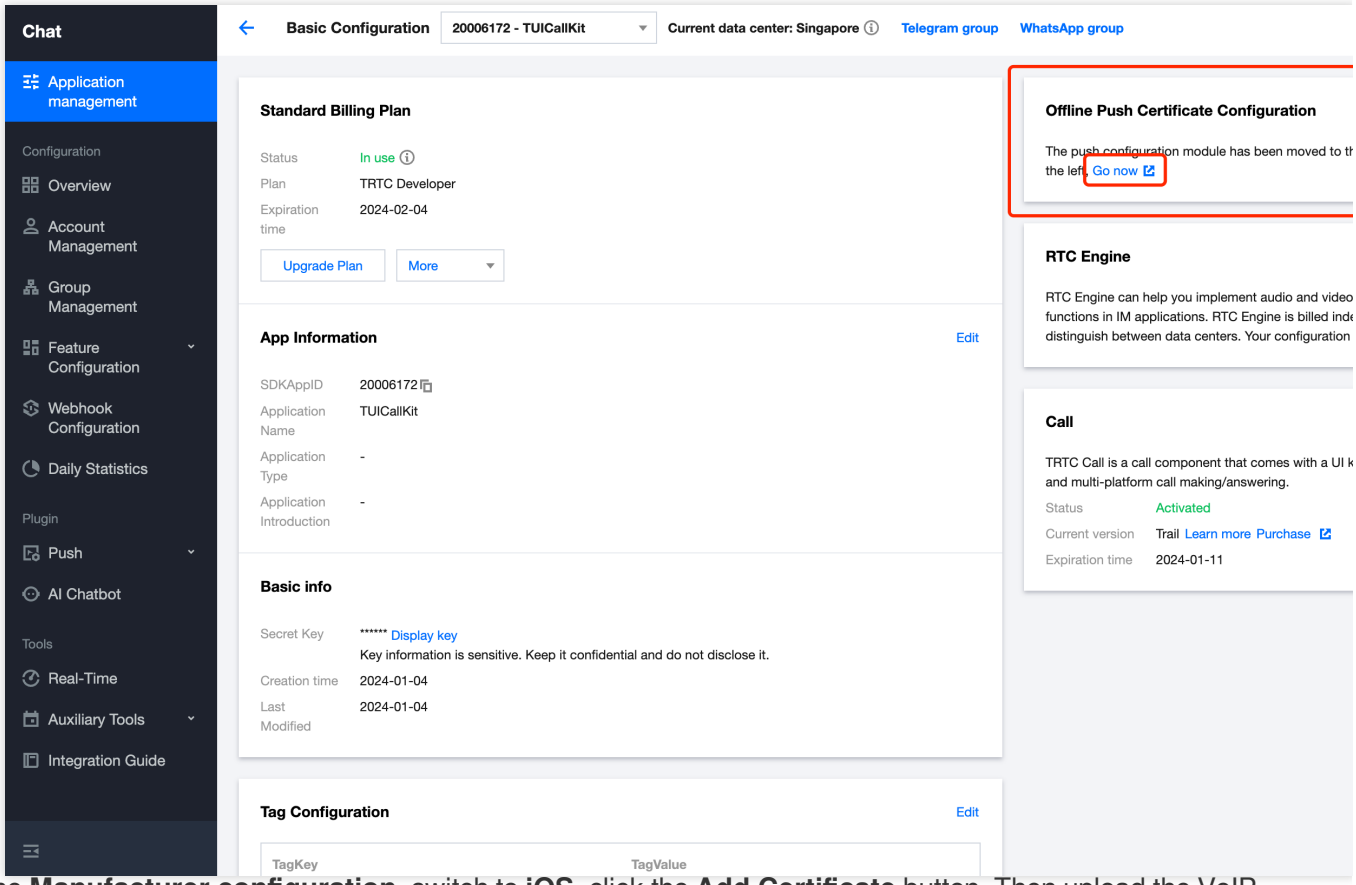
When saving the `P12` file, be sure to set a password for it.



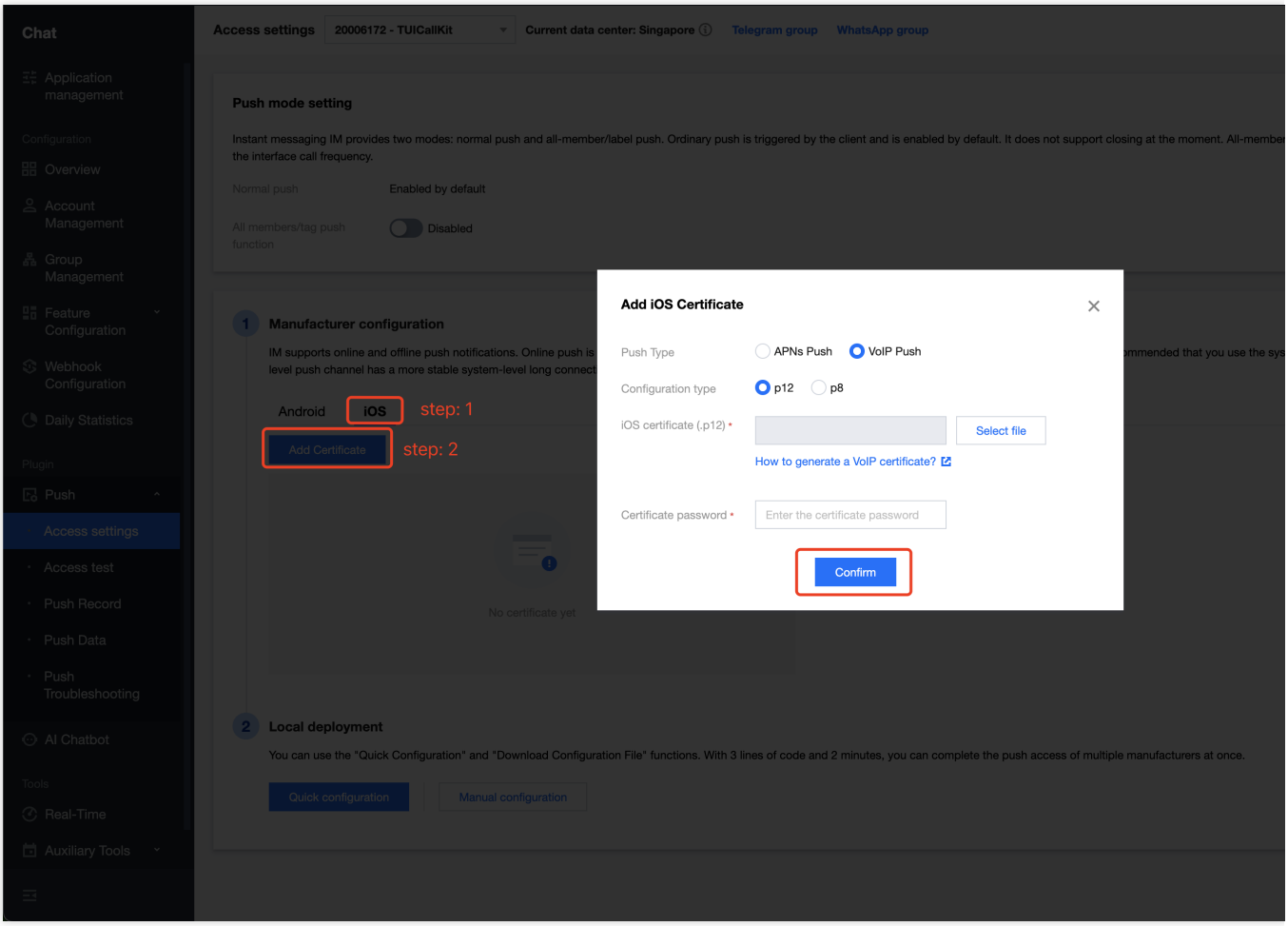
Step 2: Upload certificates to the Chat Console

Open the Chat Console, select your created Chat application, and follow the steps below to upload your certificates:

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



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



Note:

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

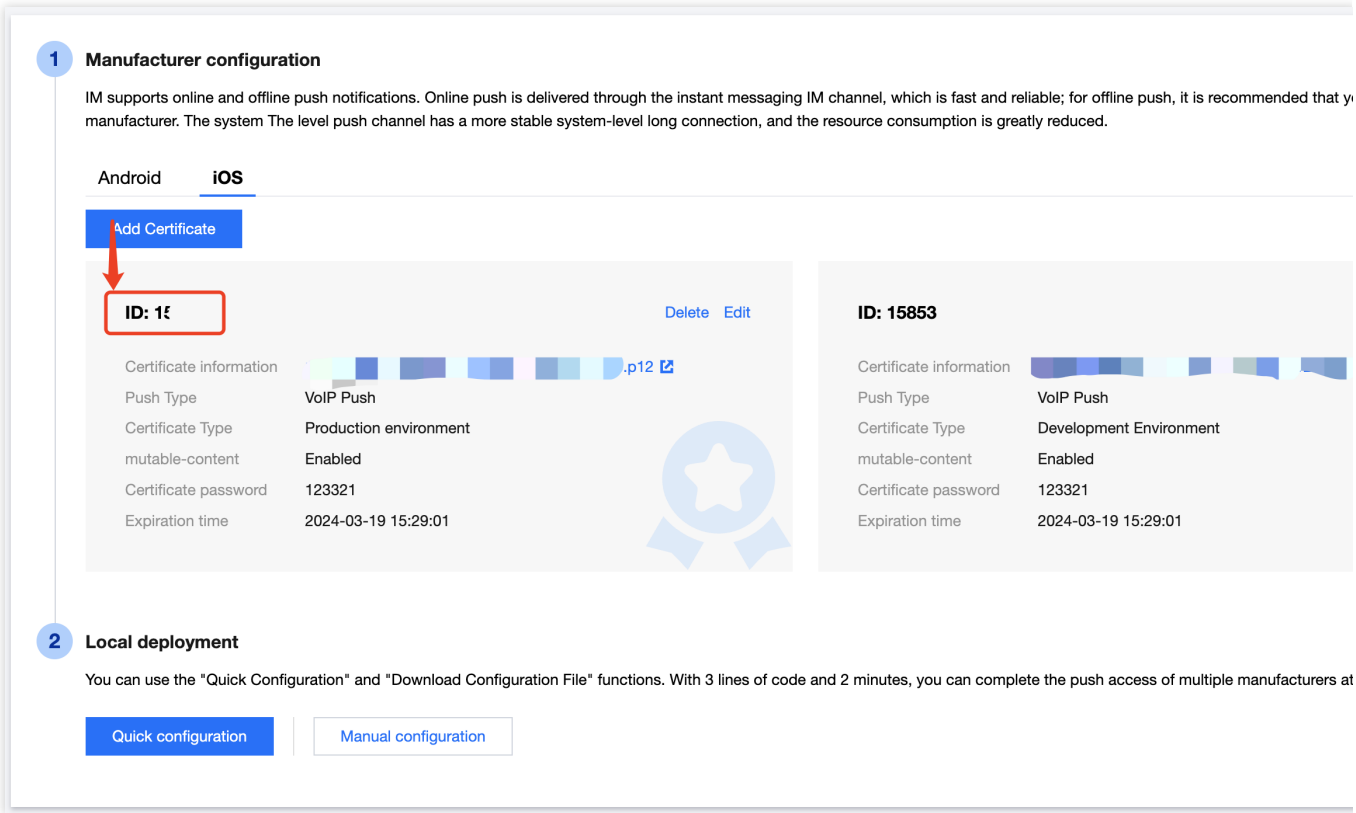
The uploaded certificate name should be in English (especially avoiding brackets and other special characters).

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

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

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

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

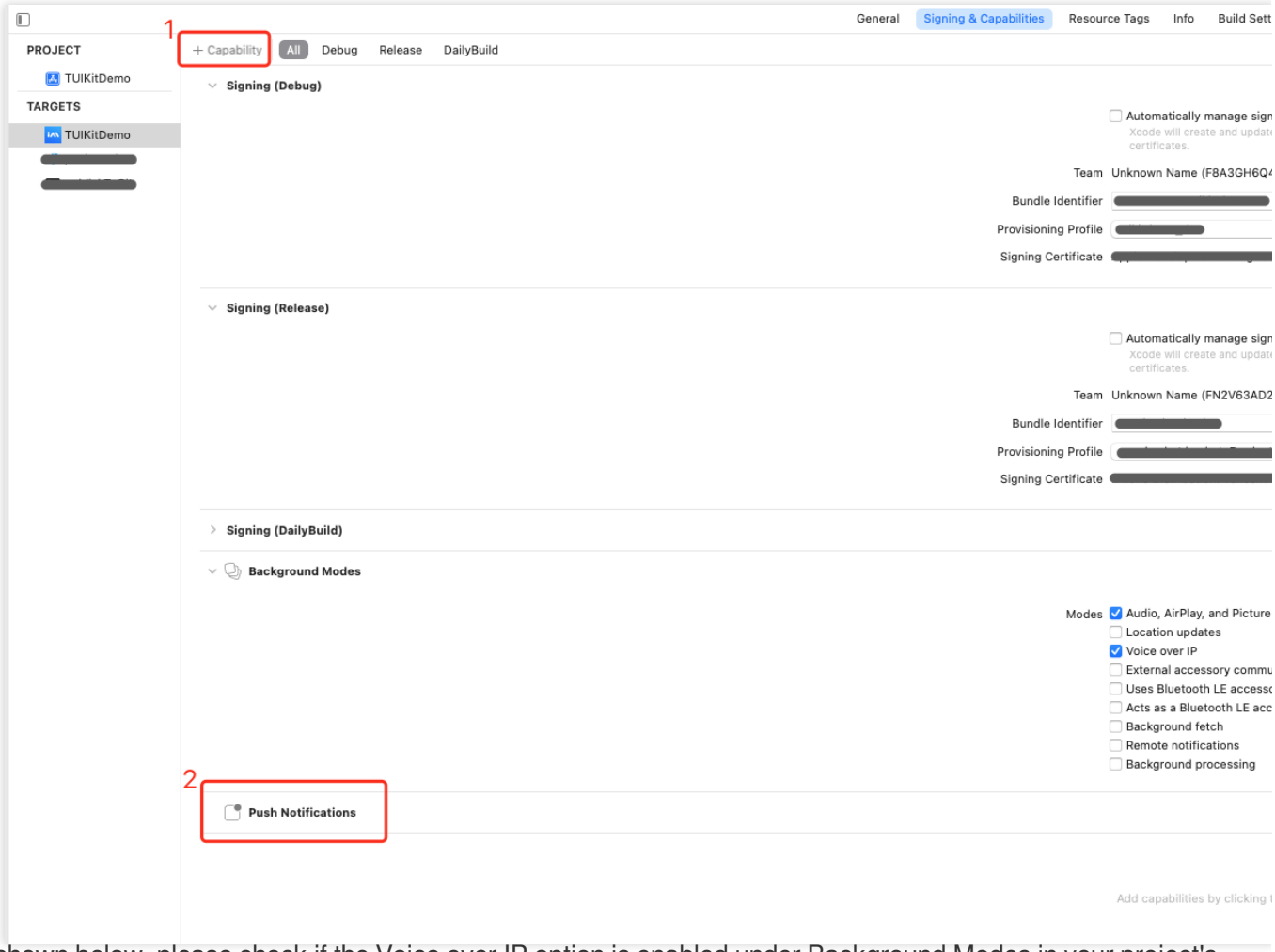


Note:

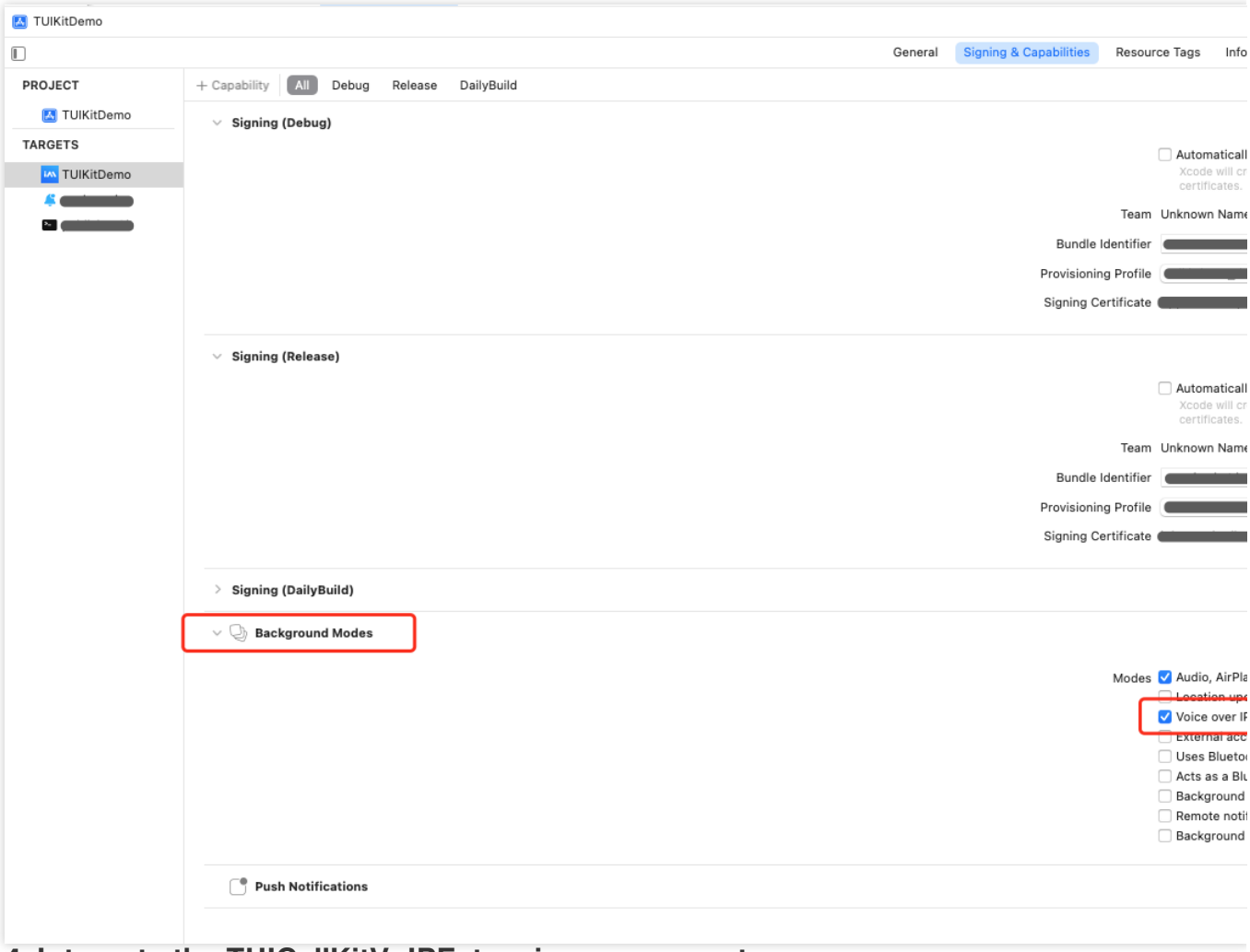
The certificate IDs for the development and production environments must be strictly distinguished and filled in according to the actual environment in Step 4: Integrate the TUICallKitVoIPExtension component.

Step 3: Complete the project configuration

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



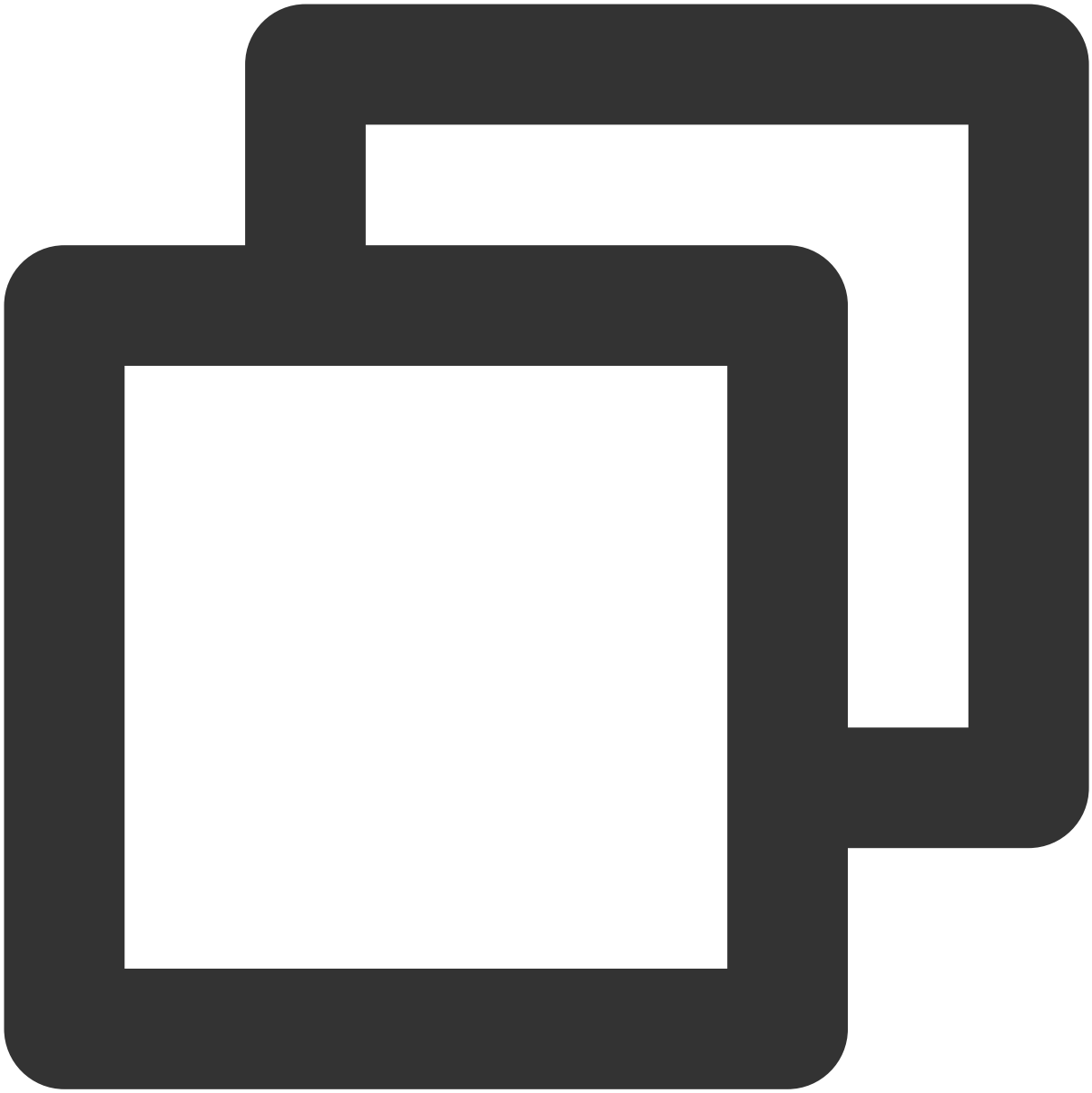
2. As shown below, please check if the Voice over IP option is enabled under Background Modes in your project's capabilities.



Step 4: Integrate the TUICallKitVoIPExtension component

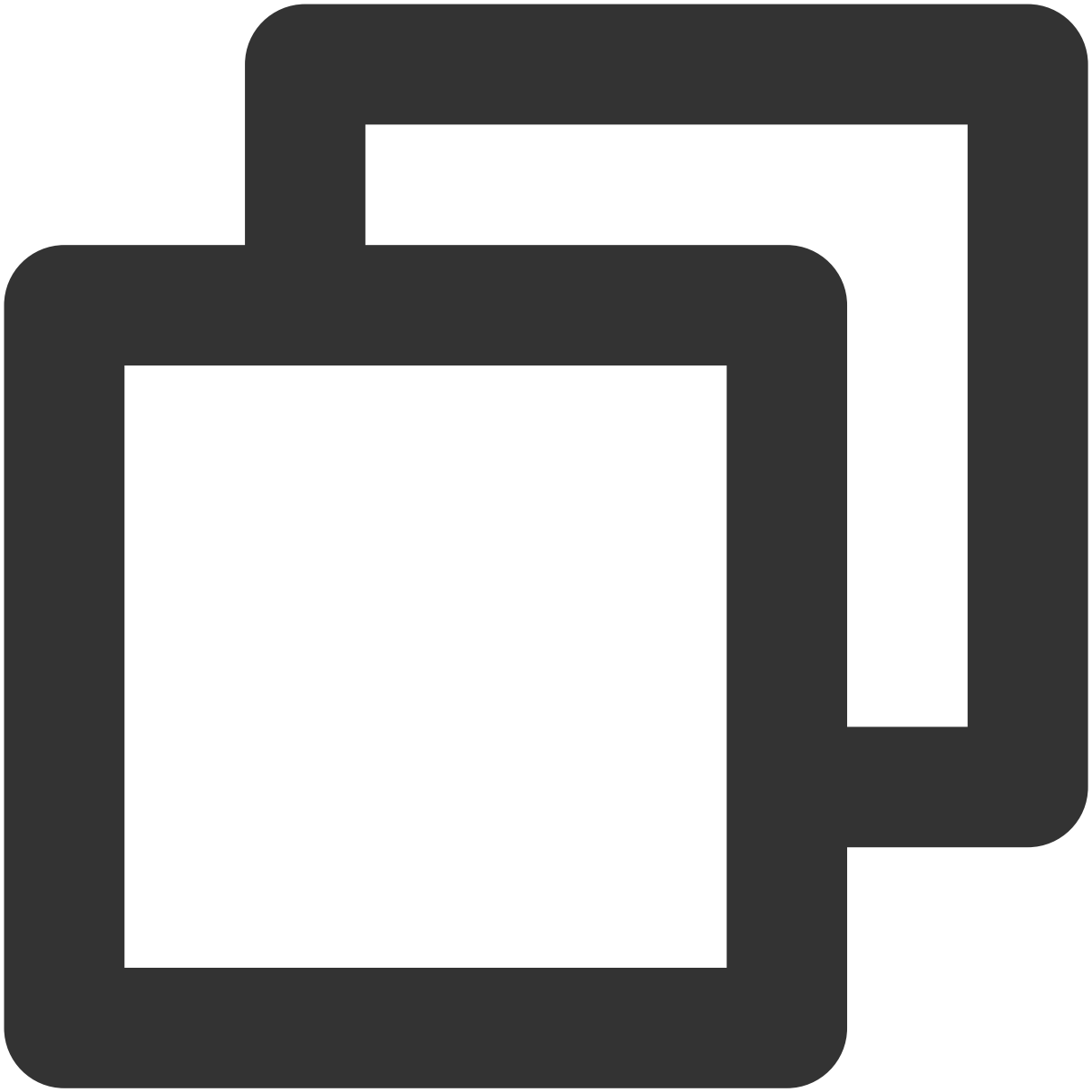
Use CocoaPods to import the component, following these steps:

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



```
pod 'TUICallKitVoIPExtension/Professional'
```

2. Execute the command below to install the component.

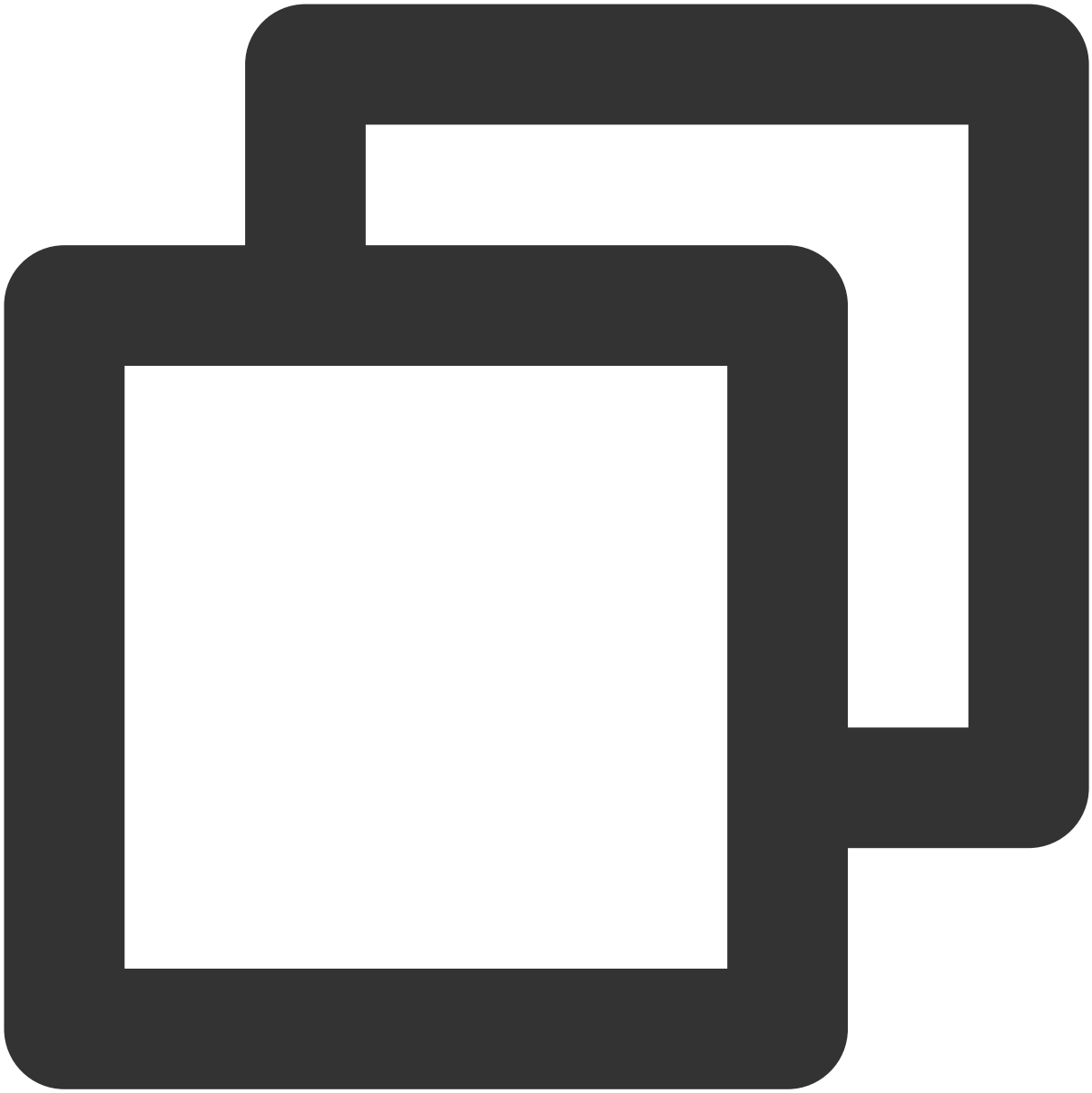


```
pod install
```

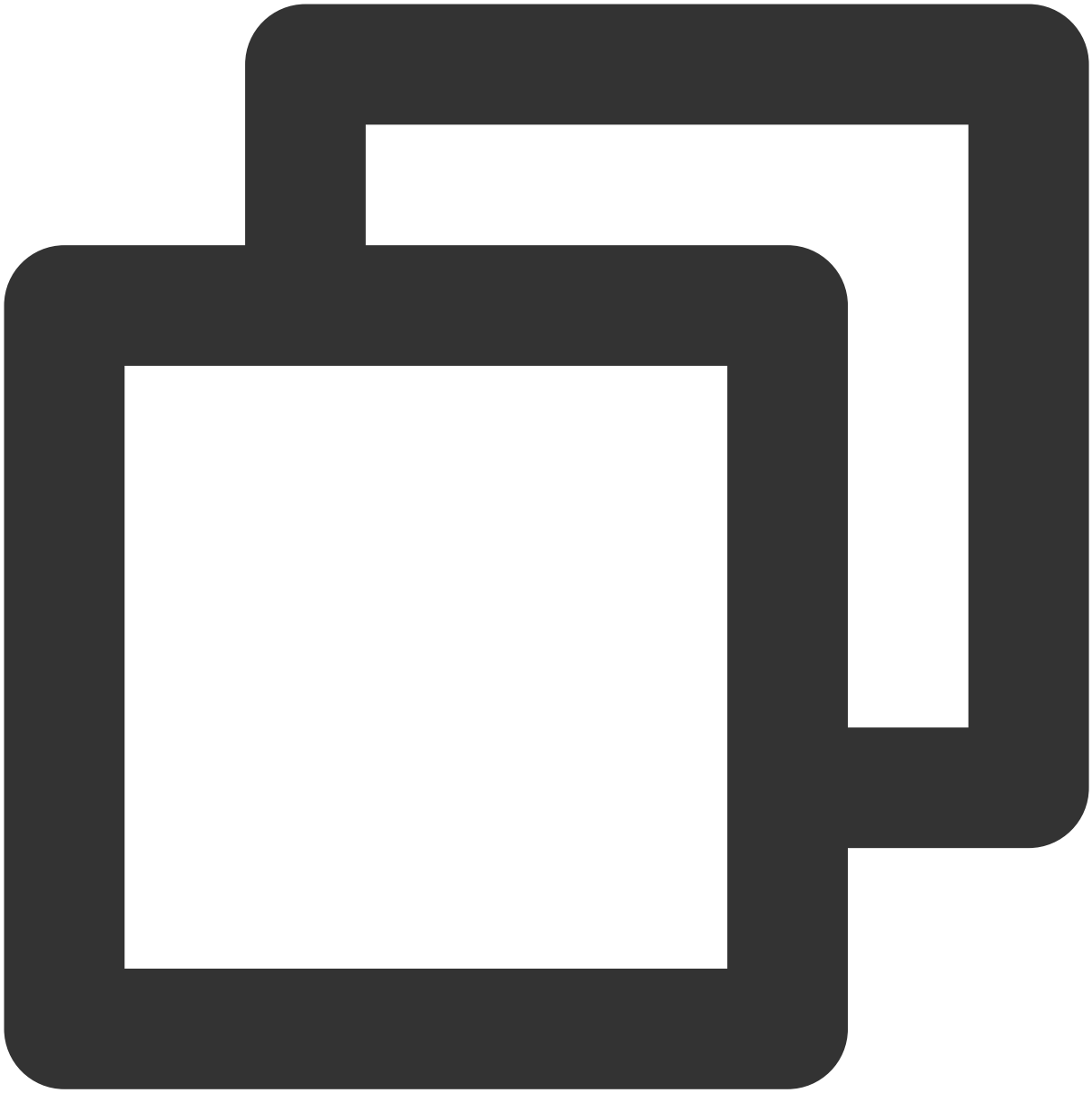
3. Report the Certificate ID recorded in Step 2.

Swift

Objective-C



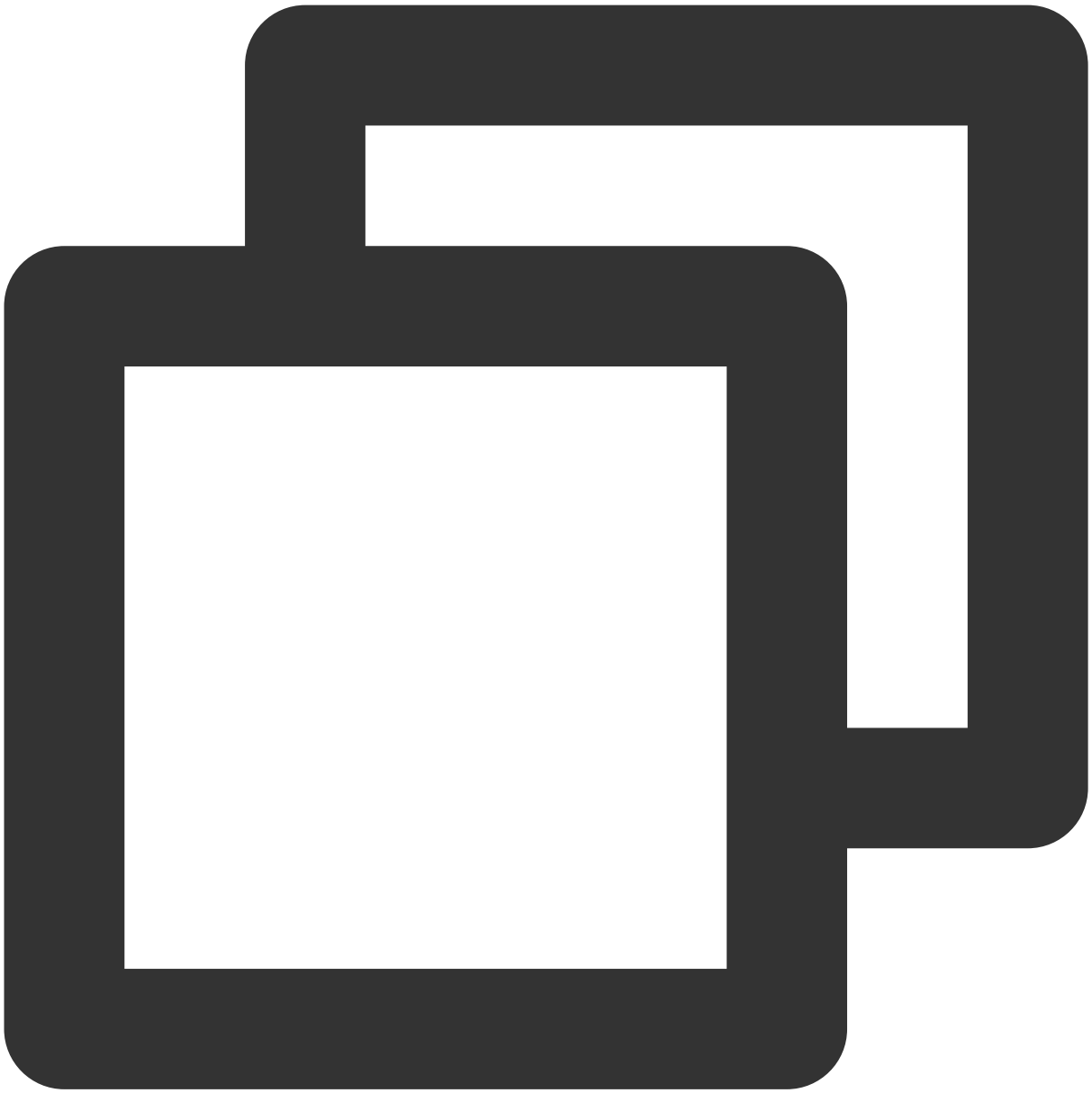
```
import TUICallKitVoIPExtension

func application(_ application: UIApplication, didFinishLaunchingWithOptions launch

    // Report certificate ID
    TUICallKitVoIPExtension.setCertificateID(1234)

    return true
}
```



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

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD

    // Report certificate ID
    [TUICallKitVoIPExtension setCertificateID:1234];

    return YES;
}
```

Note:

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



```
pod repo update
```

Make a VoIP call

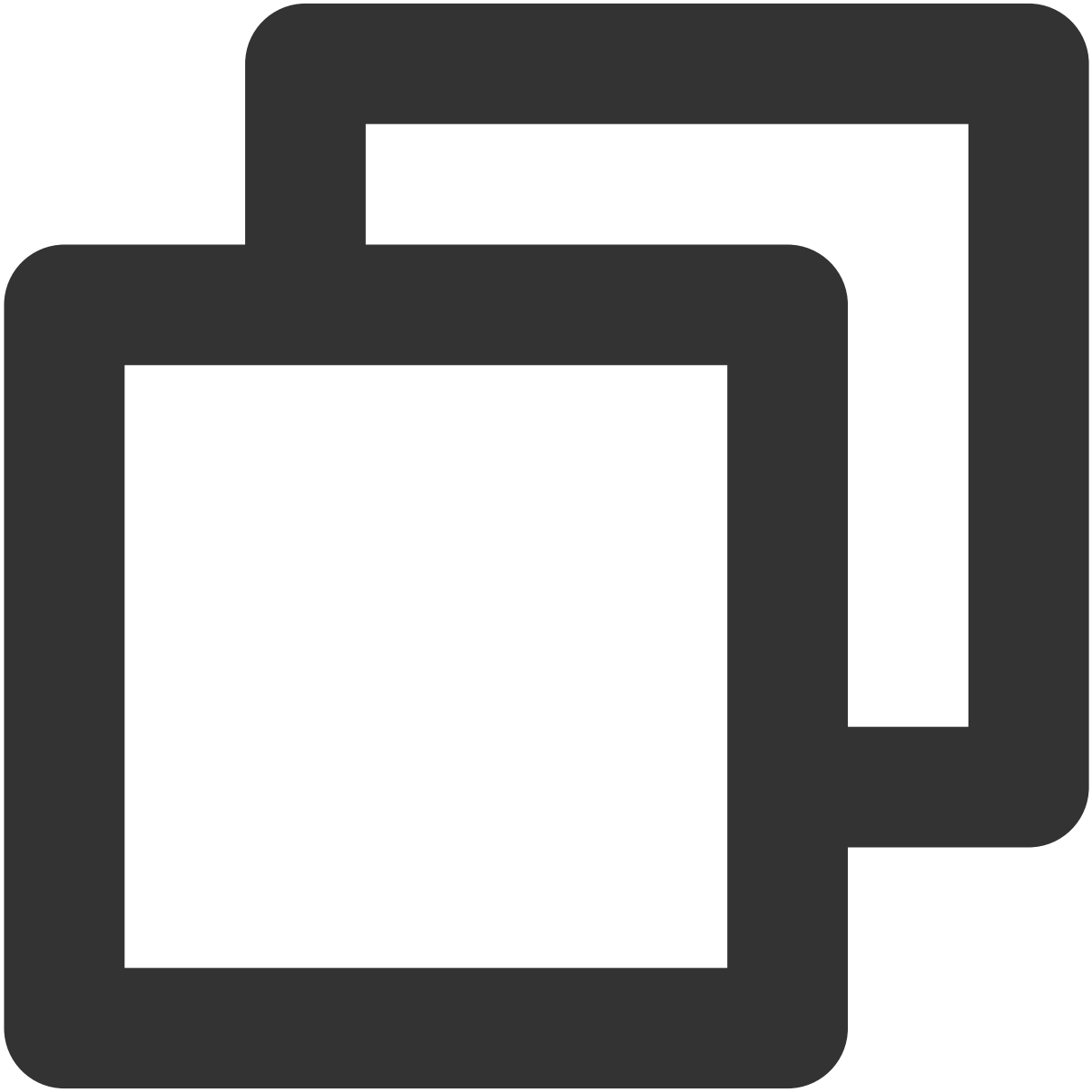
If you need to make a VoIP call, you need to set the iOSPushType field of the OfflinePushInfo to `TUICallIOSOfflinePushTypeVoIP` when calling call. The default value is `TUICallIOSOfflinePushTypeAPNs` .

Swift

Java

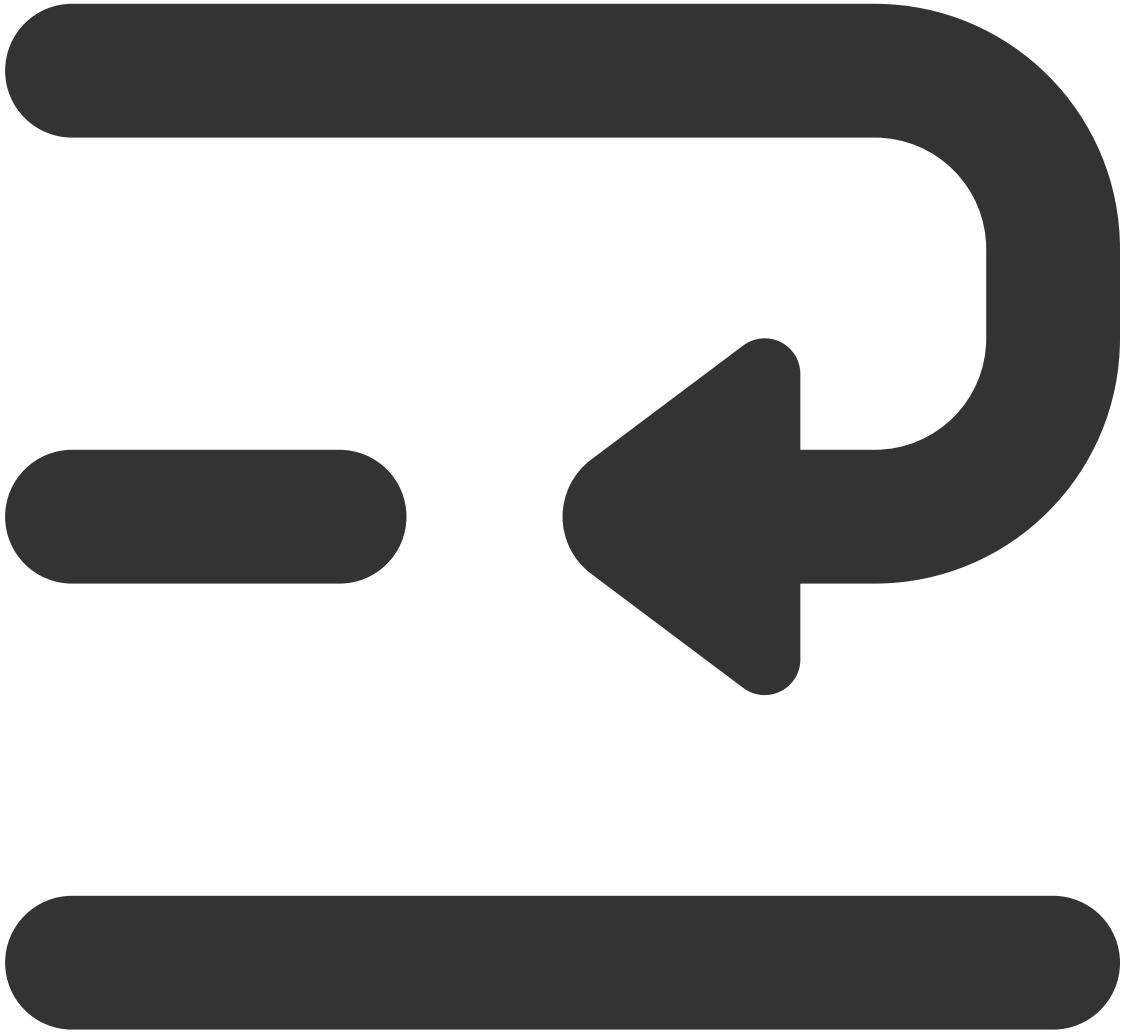
Dart

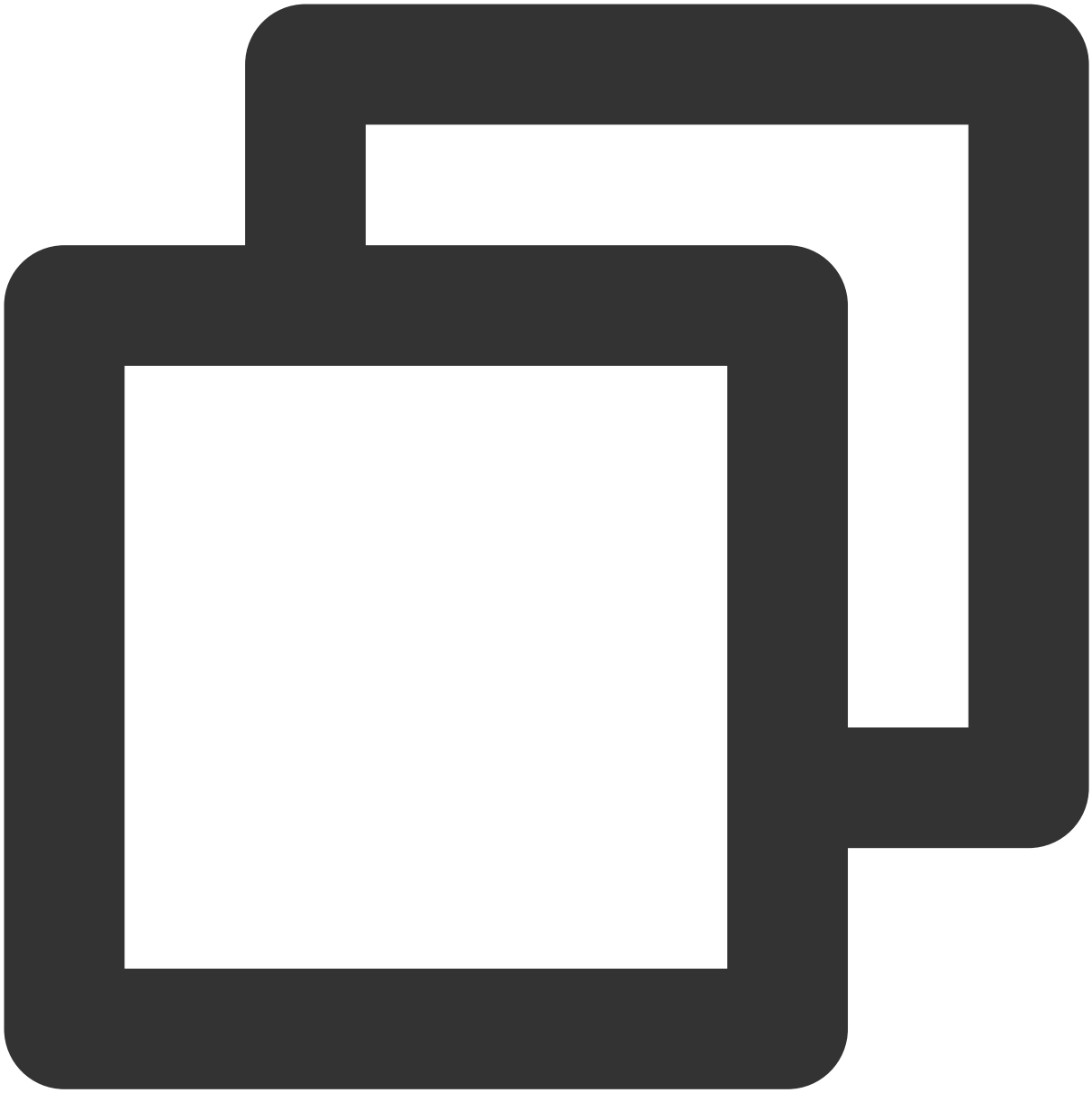
Objective-C



```
import TUICallKit_Swift
import TUICallEngine
```

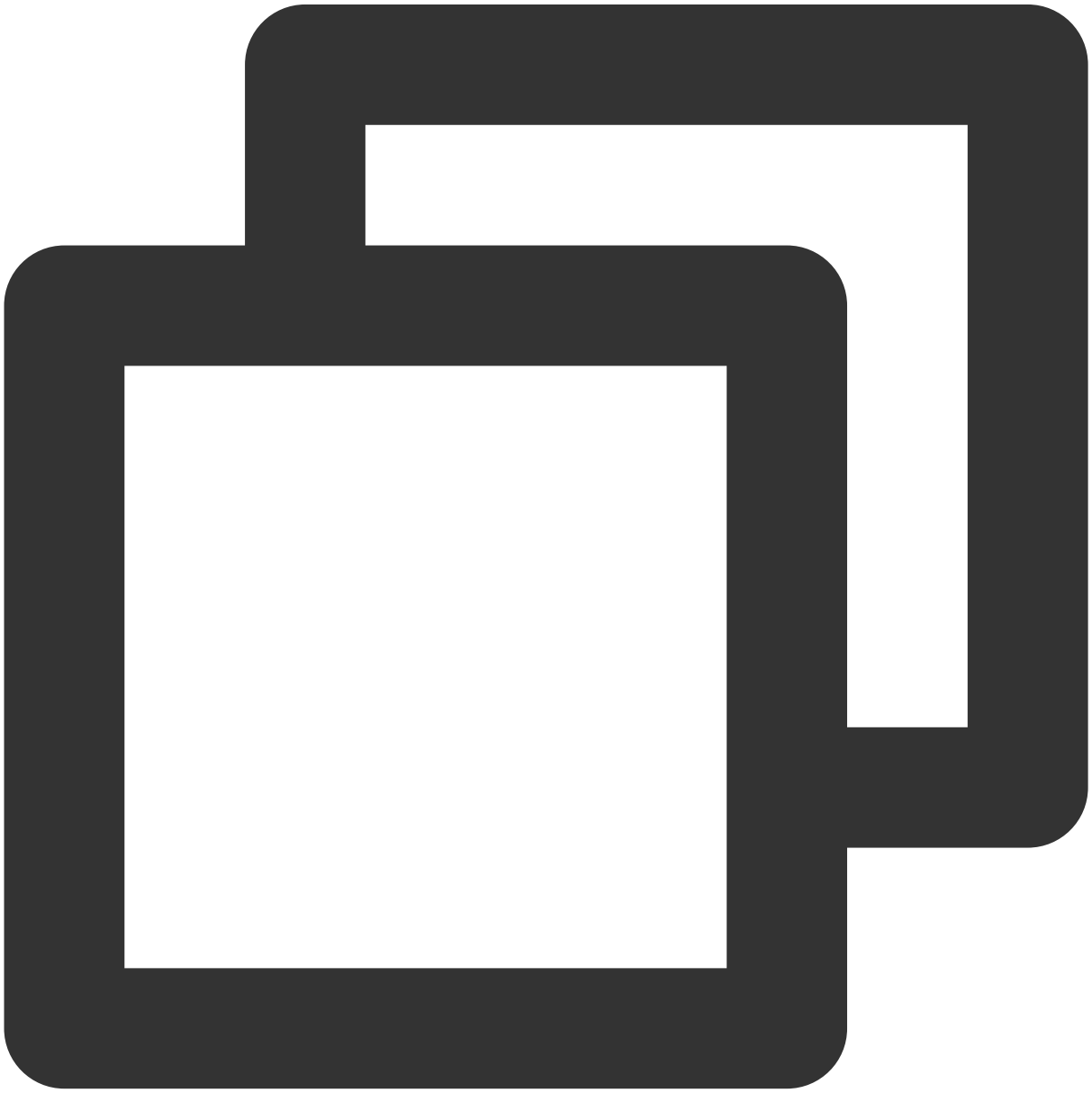
```
let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = ""
pushInfo.desc = "You have a new call"
pushInfo.iOSPushType = .voip
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO must set a ChannelID to receive push messages. This channelID needs to be t
pushInfo.androidOPPOChannelID = "tuikit"
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HM
pushInfo.androidHuaWeiCategory = "IM"

let params = TUICallParams()
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo

TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: p

} fail: { code, message in

}
```



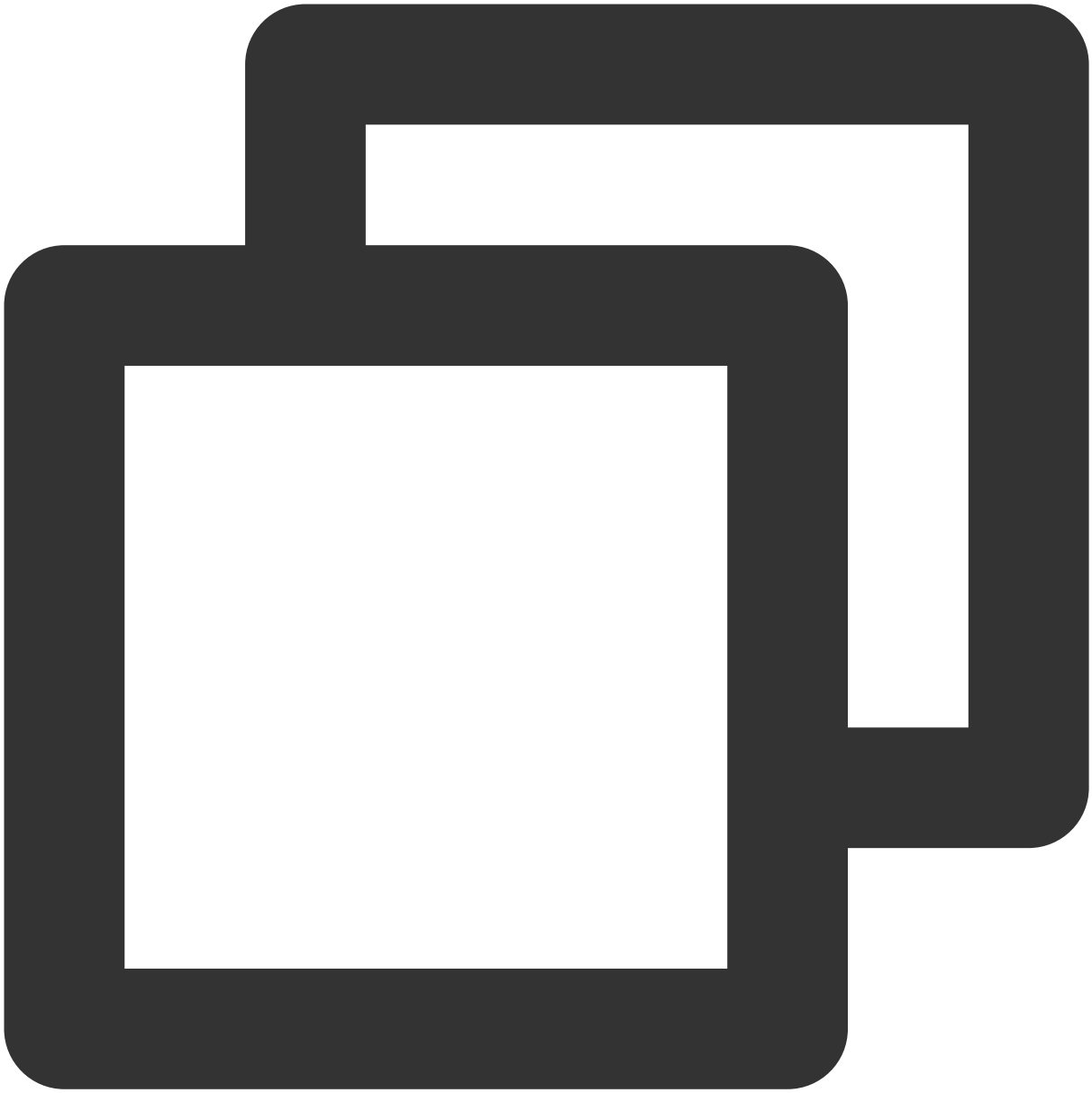


```
TUICallDefine.OfflinePushInfo offlinePushInfo = new TUICallDefine.OfflinePushInfo()
offlinePushInfo.setTitle("");
offlinePushInfo.setDesc("You have receive a new call");
// For OPPO, you must set the `ChannelID` to receive push messages. The `ChannelID`
// OPPO must set a ChannelID to receive push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

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

TUICallDefine.CallParams params = new TUICallDefine.CallParams();
params.offlinePushInfo = offlinePushInfo;

TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Video, para
```



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

 TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);

 TUICallKit.instance.call(callUserId, TUICallMediaType.audio, params);
```



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <TUICallEngine/TUICallEngine.h>

- (TUICallParams *)getCallParams {
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}
```

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

[[TUICallKit createInstance] callWithUserId:@"123456"
                                callMediaType:TUICallMediaTypeAudio
                                       params:[self getCallParams] succ:^{

} fail:^(int code, NSString * _Nullable errMsg) {

}];Objective-C
```

Call from the system call log

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

Note:

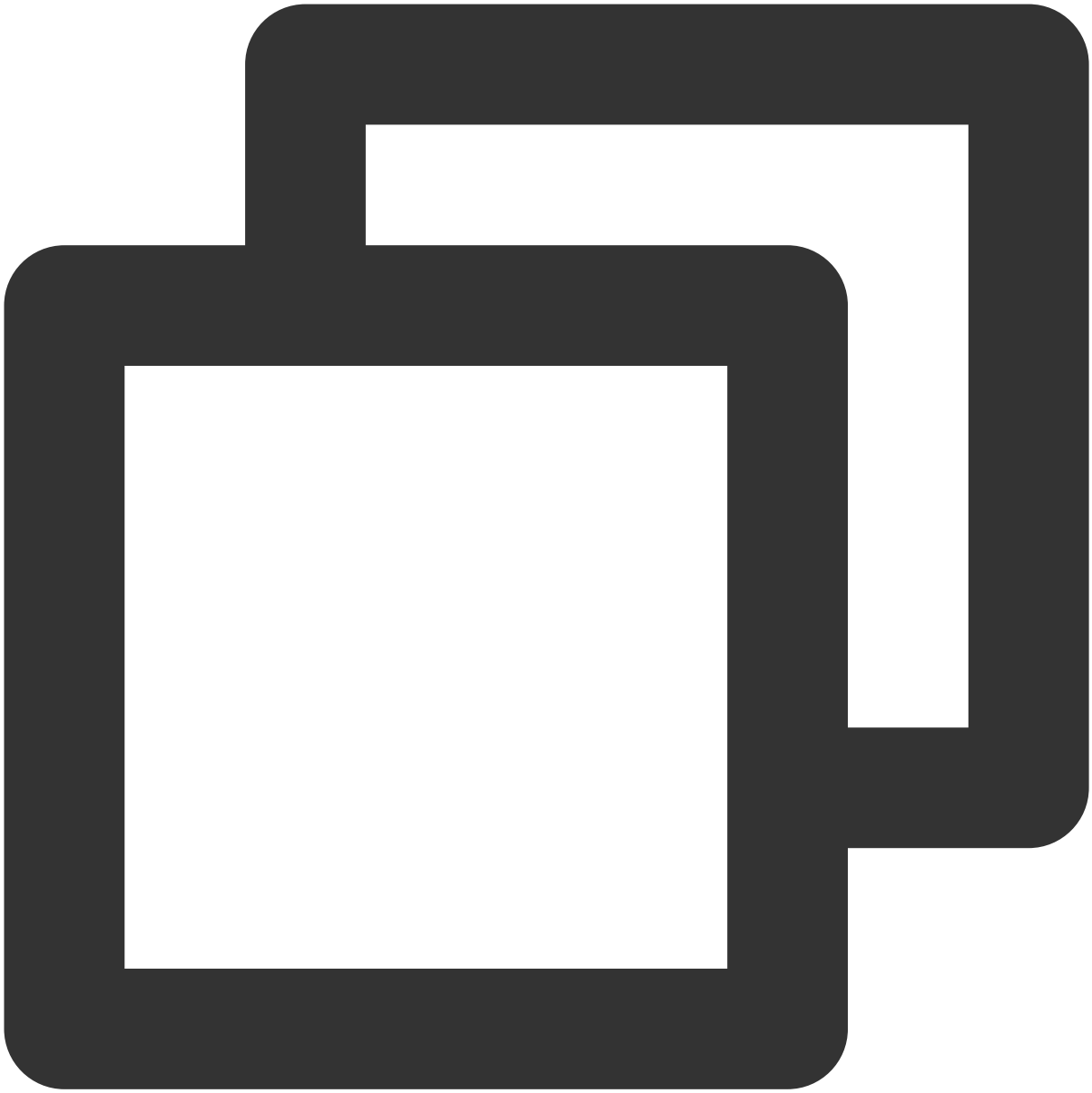
Only supports dialing one-on-one audio and video calls directly.

The logged-in account must be the same account.

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

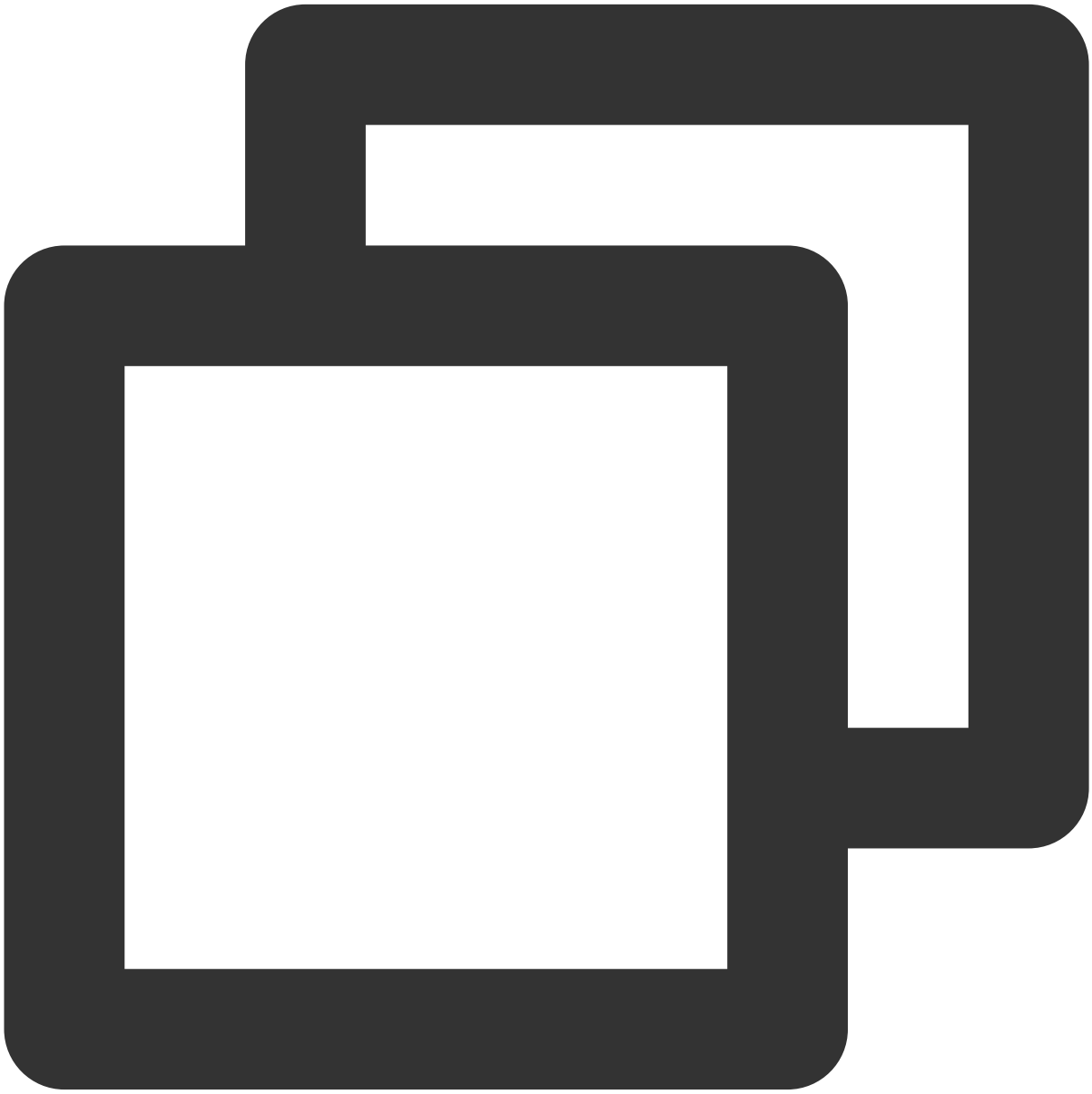
Swift

Objective-C



```
import TUICallKitVoIPExtension

/// Implementation in AppDelegate, for versions before iOS 13
func application(_ application: UIApplication, continue userActivity: NSUserActivit
    TUICallKitVoIPExtension.call(with: userActivity)
    return true
}

/// Implementation in SceneDelegate
func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connect
    for userActivity in connectionOptions.userActivities {
```

```
        TUICallKitVoIPExtension.call(with: userActivity);
    }
}

func scene(_ scene: UIScene, continue userActivity: NSUserActivity) {
    TUICallKitVoIPExtension.call(with: userActivity);
}
```



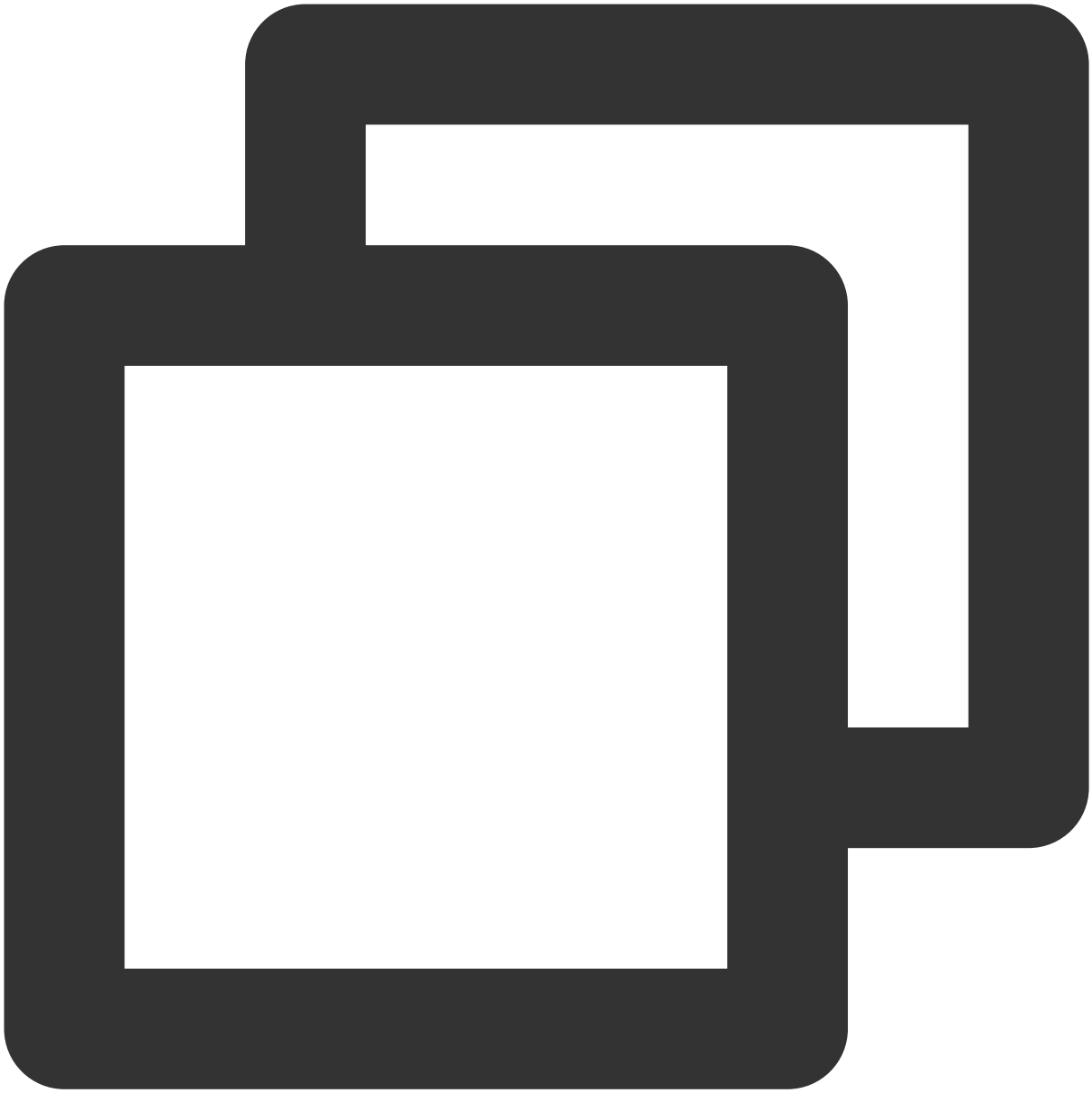
```
#import <TUICallKitVoIPExtension/TUICallKitVoIPExtension.h>

/// Implementation in AppDelegate, for versions before iOS 13
```

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

/// Implementation in SceneDelegate
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)sessionptions
    [connectionOptions.userActivities enumerateObjectsUsingBlock:^(NSUserActivity *
        [TUICallKitVoIPExtension callWith:userActivity];
    }];
}

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

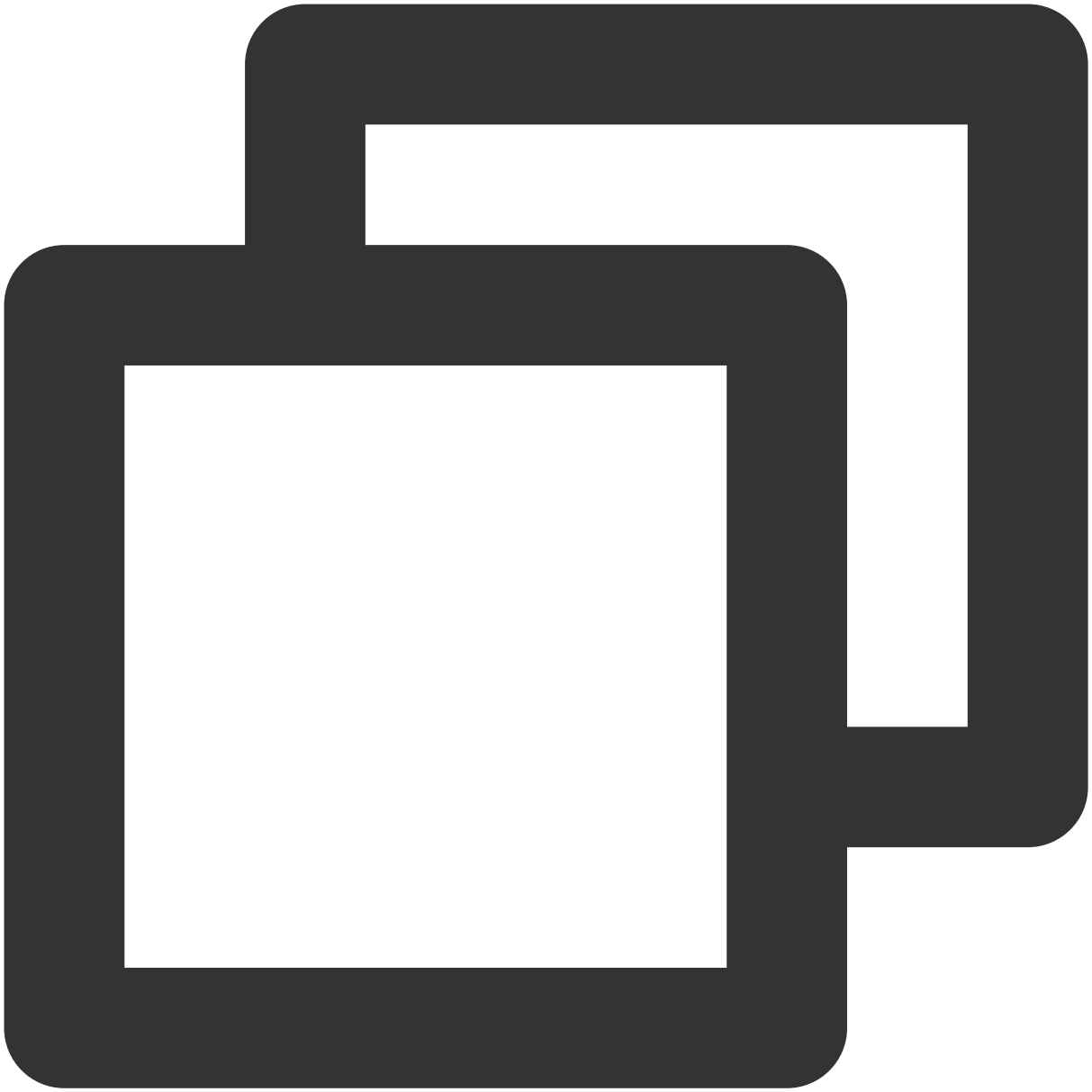
2. On iOS 13 (and later versions) without **SceneDelegate**, you only need to implement the following method in **AppDelegate**.

Swift

Objective-C



```
import TUICallKitVoIPExtension

func application(_ application: UIApplication, continue userActivity: NSUserActivit
    TUICallKitVoIPExtension.call(with: userActivity)
    return true
}
```



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

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

Frequently Asked Questions

Can't receive VoIP Push ?

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

Notification

Last updated：2024-07-29 15:23:09

TUICallKit components support offline push through the integration of the push plugin. To help developers easily implement offline push in their projects, we recommend using the TIMPush plugin (paid). TIMPush push plugin has the following advantages:
The integration cycle is short, with full-manufacturer access expected to take only 30 minutes.
Supports data statistics and link tracking, making it convenient for you to view various indicators such as push reach rate, click-through rate, and conversion rate.
Supports full-staff/Tag push, allowing you to push marketing ads, notifications, news information, and other content to all users or specified groups.
Supports cross-platform frameworks like uni-app and Flutter.

Integration effect

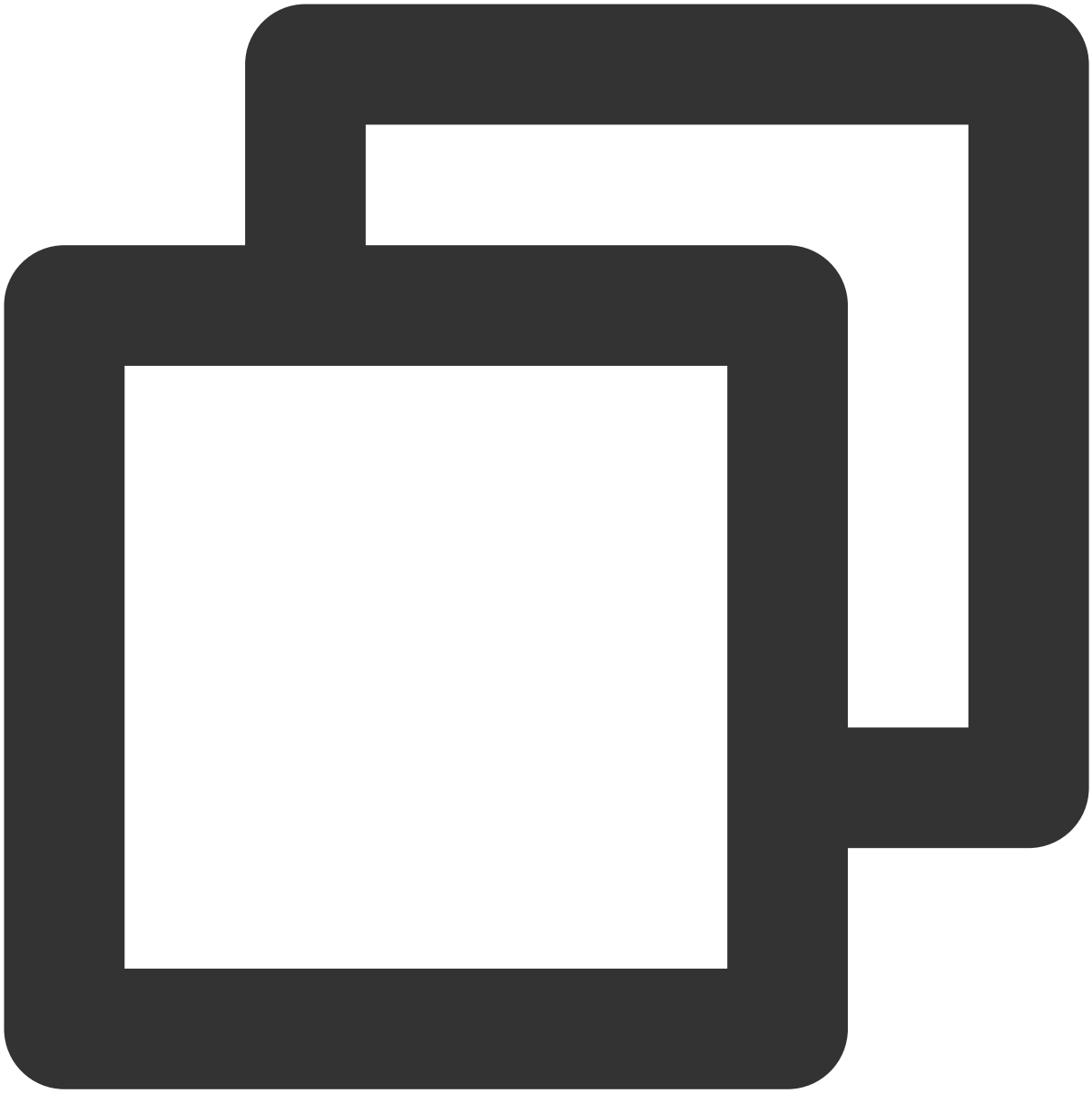
Effect when the screen is locked	Effect in the background

Configuring Offline Push

1. Integrate TIMPush component.
2. Activate remote push for the app.
3. Generate certificate.
4. Upload certificates to the Tencent RTC Console.
5. Complete Project Configuration
6. Configure push parameters.
7. Dial offline push calls.

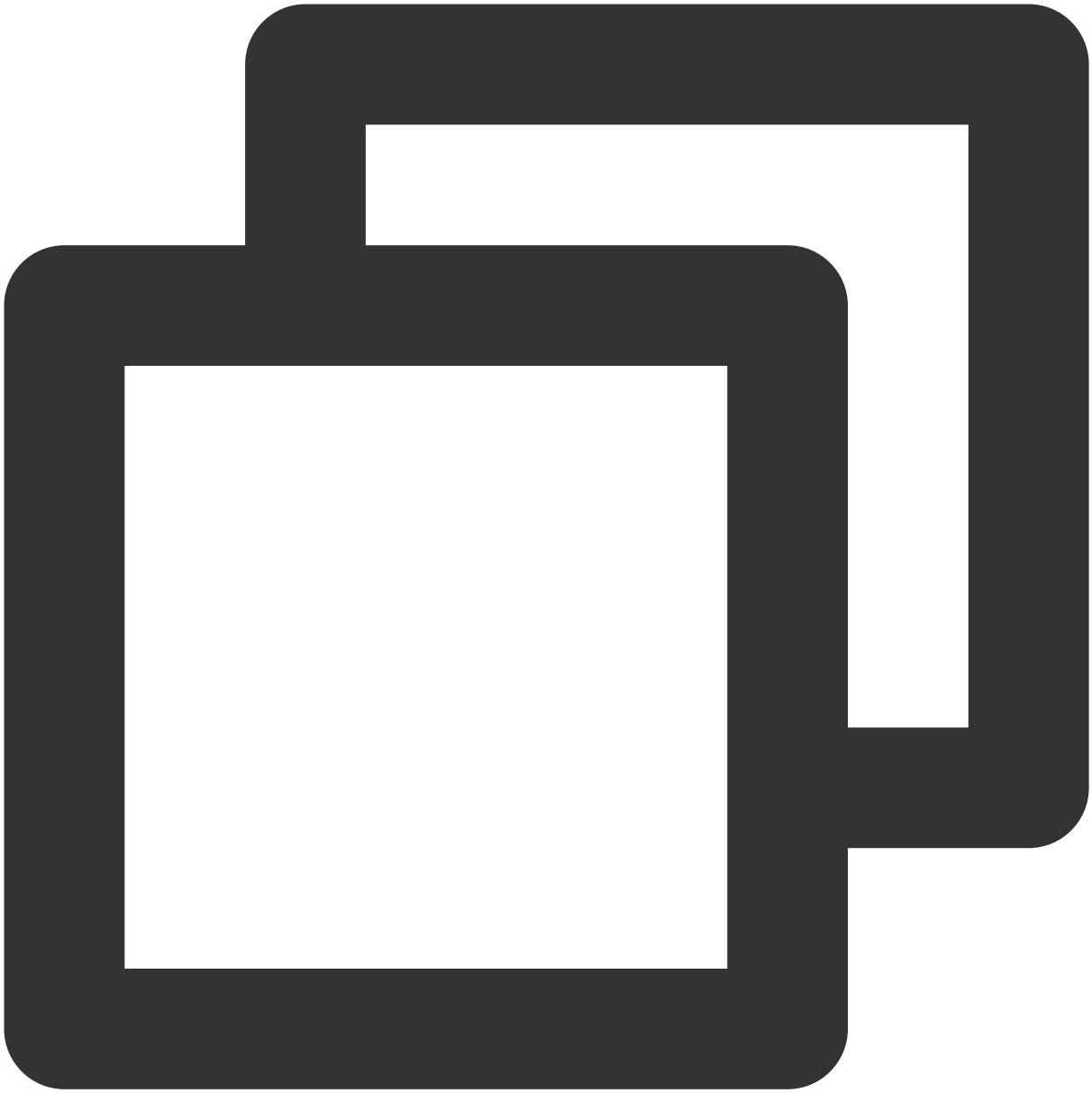
Step 1. Integrate the TIMPush component

1. The TIMPush component supports CocoaPods integration. You need to add the component dependencies in the Podfile.



```
target 'YourAppName' do
  # Uncommment the next line if you're using Swift or would like to use dynamic fra
  use_frameworks!
  use_modular_headers!

  # Pods for Example
  pod 'TIMPush', '7.9.5668'
end
```

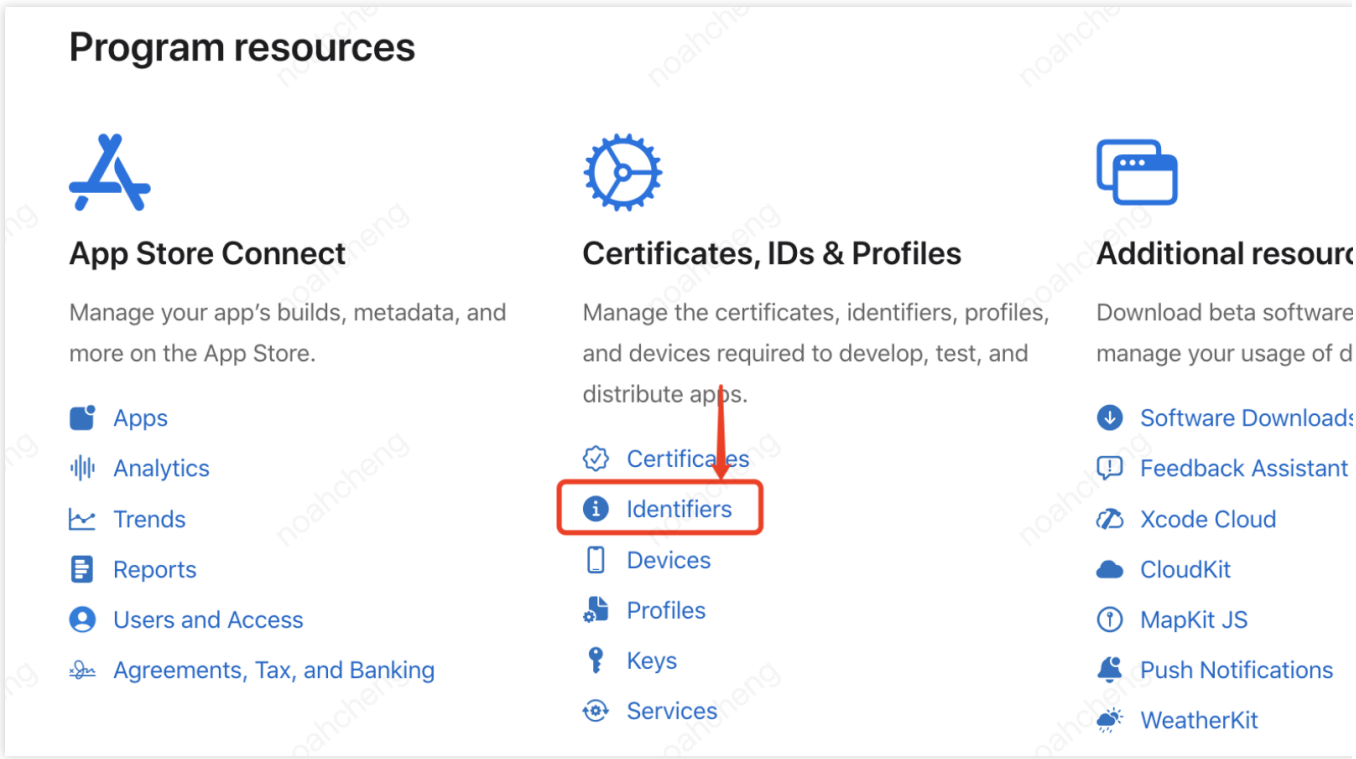
2. Run the following command to install the TIMPush component.



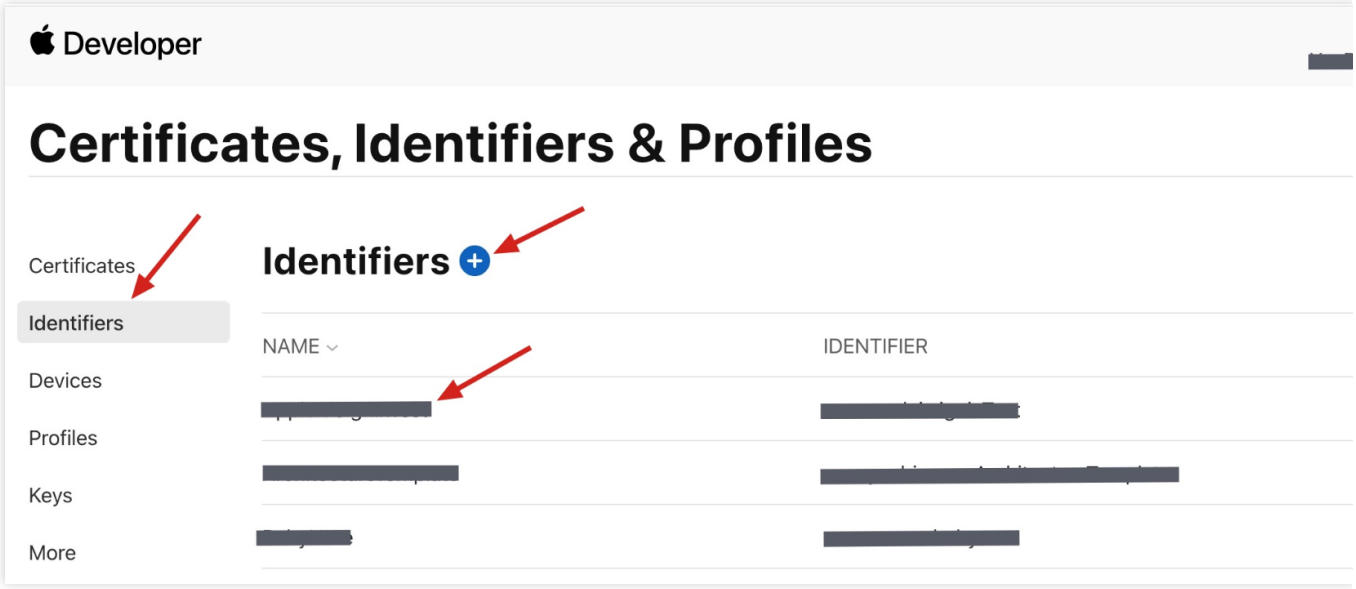
```
pod install
# If you cannot install the latest version of TUIKit, run the following command to
pod repo update
```

Step 2: Activate remote push for the app

1. Log in to the Apple Developer Center website, click on the **Certificates, IDs & Profiles** tab, then **Identifiers**, and enter **Certificates**, **Identifiers & Profiles** page.



2. Click the **+** next to **Identifiers**.



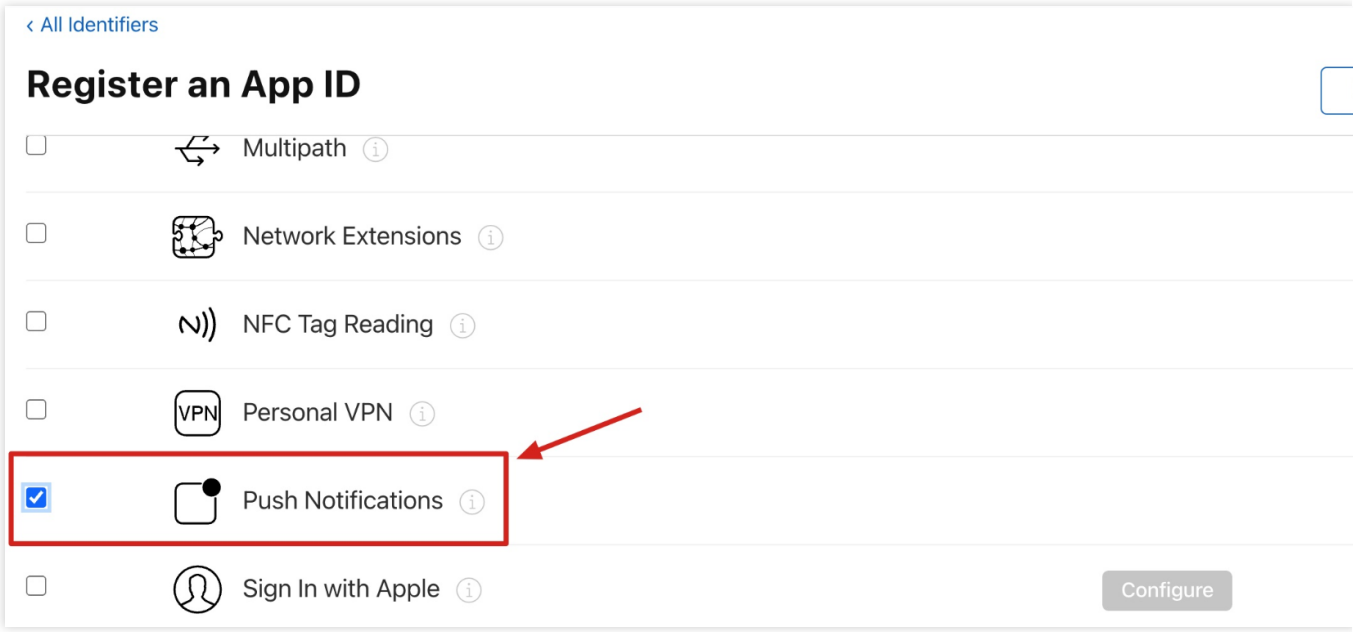
3. You can follow the steps below to create a new AppID, or add the `Push Notification Service` to your existing AppID.

Note:

It's important to note that your app's `Bundle ID` must not use the wildcard *, otherwise, the remote push service cannot be used.

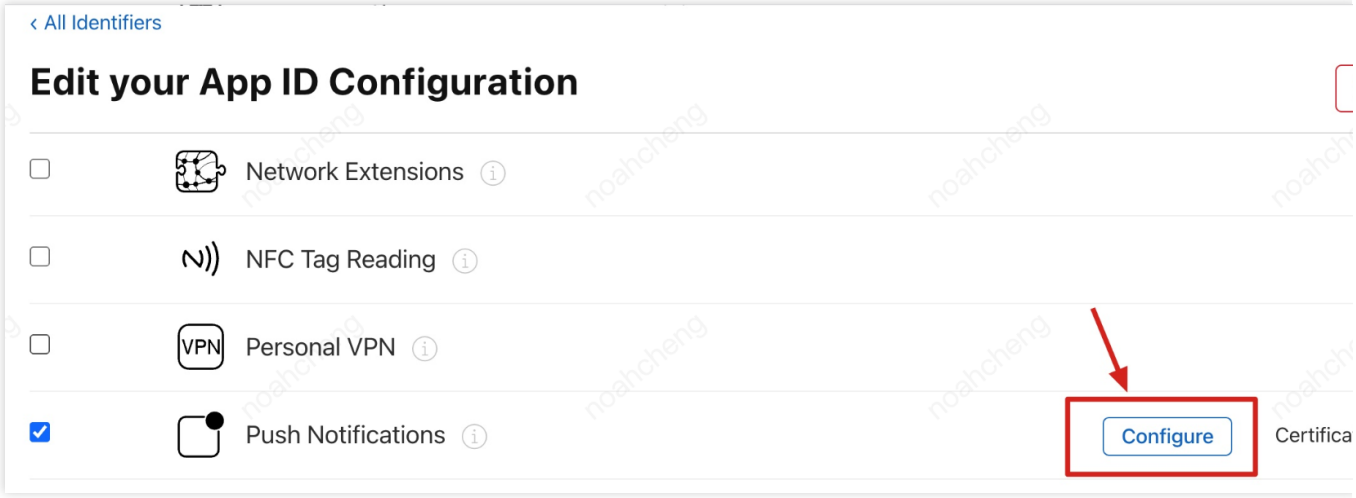
4. Select **App IDs**, click **Continue** to proceed to the next step.

5. Select **App**, click **Continue** to proceed to the next step.
6. Configure the `Bundle ID` and other information, click **Continue** to proceed to the next step.
7. Select **Push Notifications** to activate the remote push service.



Step 3: Generate the push certificate

1. Select your **AppID**, find the **Push Notifications** configuration option, and choose **Configure**.

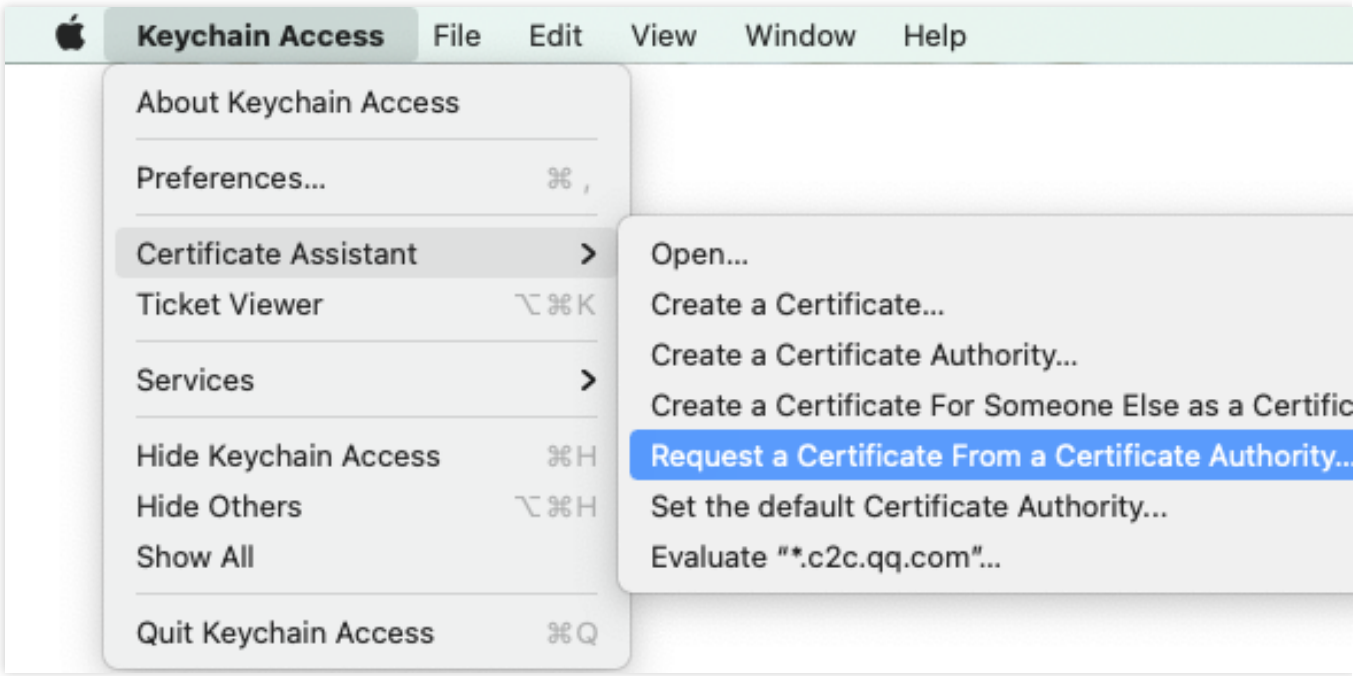


2. In the **Apple Push Notification service SSL Certificates** window, you will see two `SSL Certificates`, one for the Development environment and one for the Production environment.
3.

We

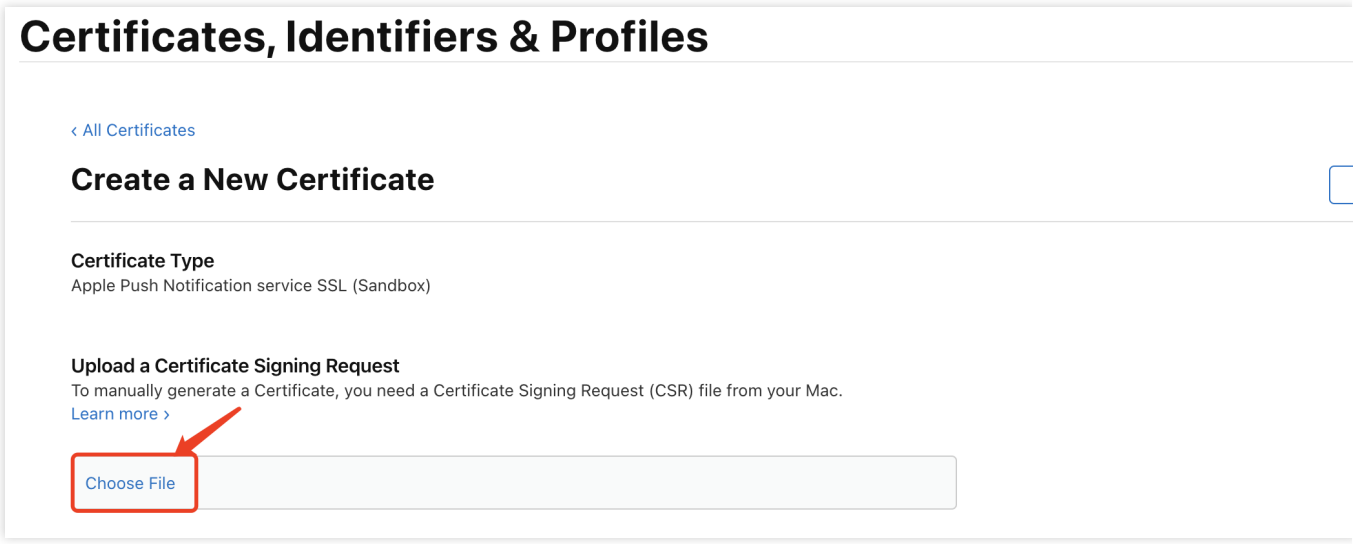
first select the development environment (Development) option **Create Certificate**, and the system will prompt us that a **Certificate Signing Request (CSR)** is needed.

4. On a Mac, open the Keychain Access tool (**Keychain Access**), and from the menu, select **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority** (`Keychain Access > Certificate Assistant > Request a Certificate From a Certificate Authority`).



5. Enter your email address, common name (your name or company name), select **Saved to disk**, and click **Continue**. The system will generate a `*.certSigningRequest` file.

6. Go back to the page on the Apple Developer website from Step 3 just now and click "Choose File" to upload the generated `*.certSigningRequest` file.



7. Click **Continue** to generate the push certificate.
8. Click **Download** to download the `Development SSL Certificate` for the development environment to your local machine.
9. Follow the steps 1-8 again to download the `Production SSL Certificate` for your production environment to your local system.

Note:

The production environment certificate is actually a merged certificate of `Sandbox` and `Production`, which can be used for both the development and production environments.

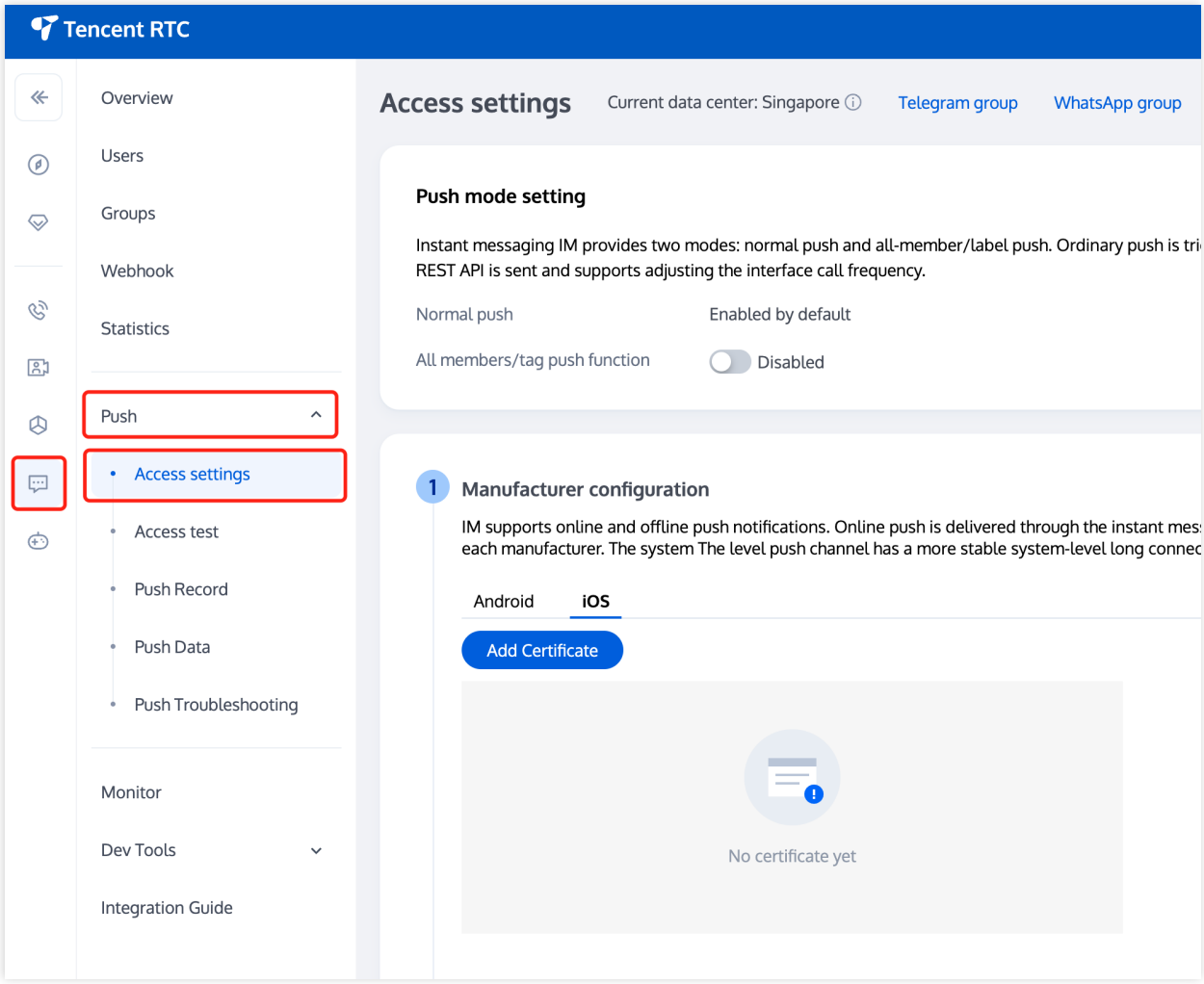
10. Double-click the downloaded `SSL Certificate` for both the development and production environments, and the system will import it into your keychain.
11. Open the Keychain Access app, **log in to My Certificates**, right-click to export the P12 files for both the newly created development environment (`Apple Development IOS Push Service`) and the production environment (`Apple Push Services`).

Note:

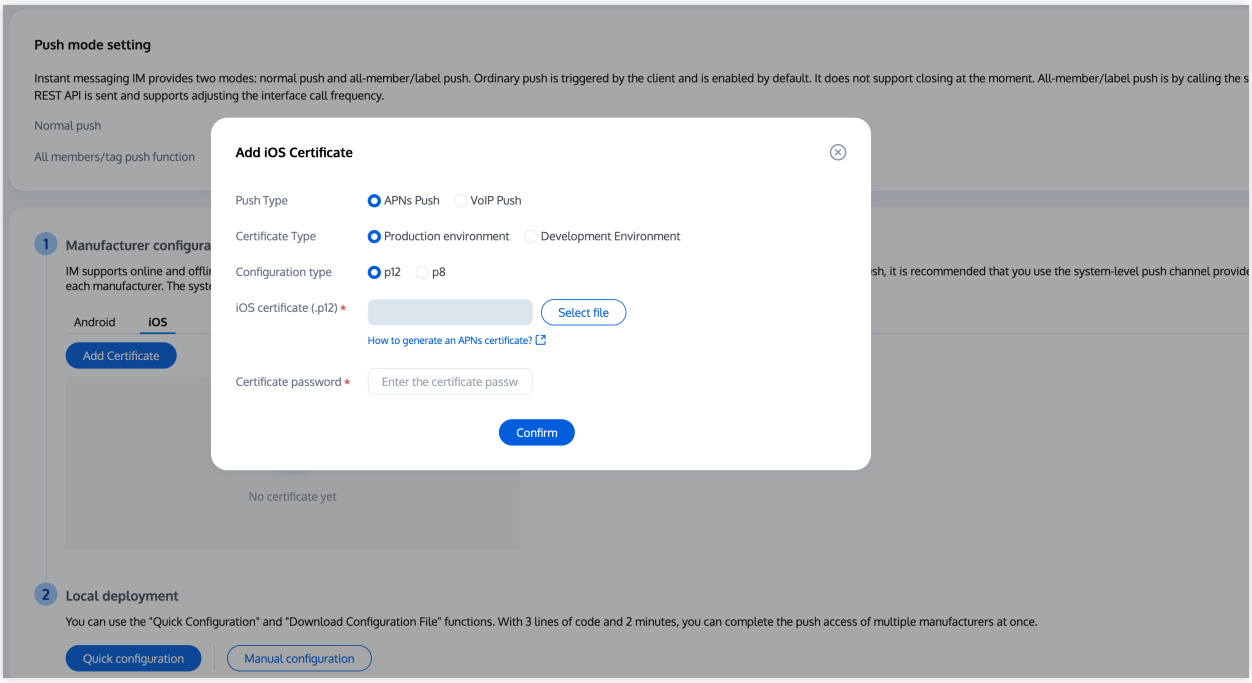
When saving the `P12` file, be sure to set a password.

Step 4: Upload the certificate to the Tencent RTC console

1. Log in to the Tencent RTC Console.
2. Click on the Target Application Card, select the **Chat** Tag on the left, click on **Push**, then click on **Access settings**.



3. Click on **iOS Native Offline Push Settings** on the right side to **Add Certificate**.
4. Select Certificate Type, upload the iOS Certificate (p12), set the Certificate Password, and click on **Confirm**.



Note:

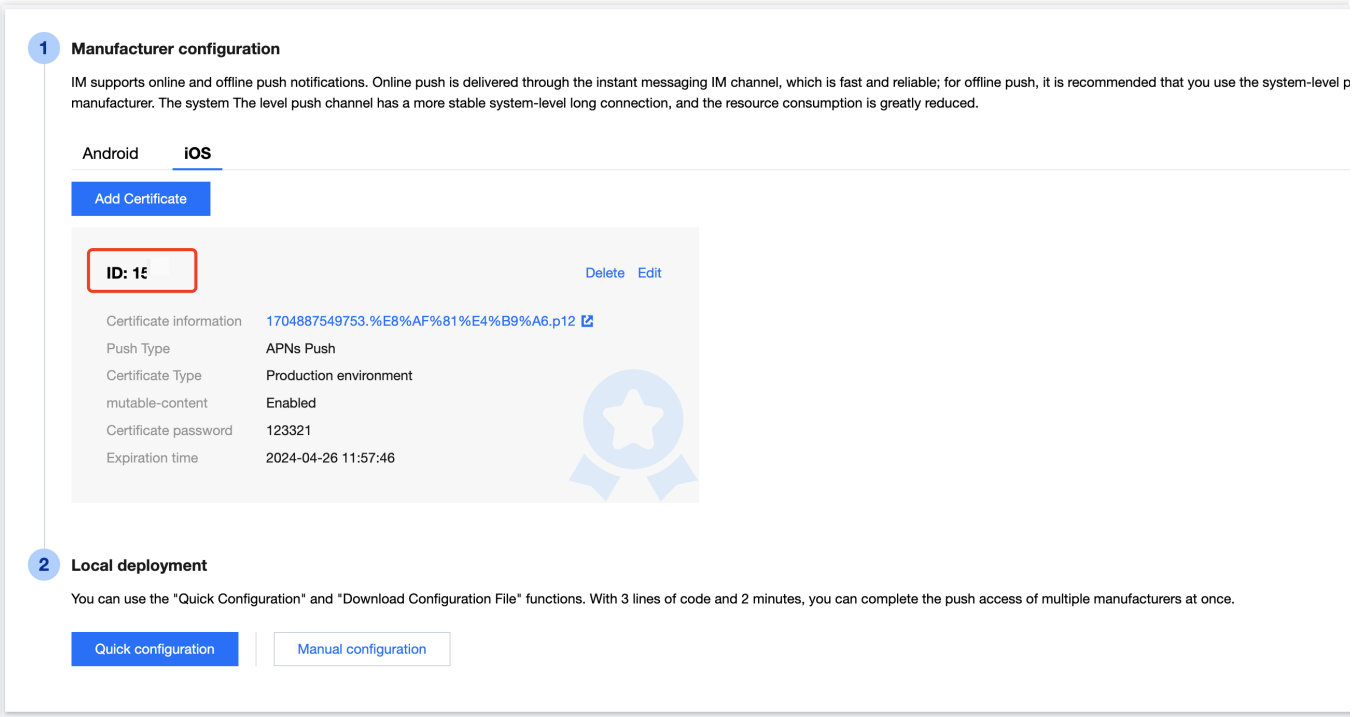
We recommend naming the uploaded certificate in English (special characters such as brackets are not allowed).

You need to set a password for the uploaded certificate. Without a password, push notifications cannot be received.

For an app published on App Store, the environment of the certificate must be the production environment. Otherwise, push notifications cannot be received.

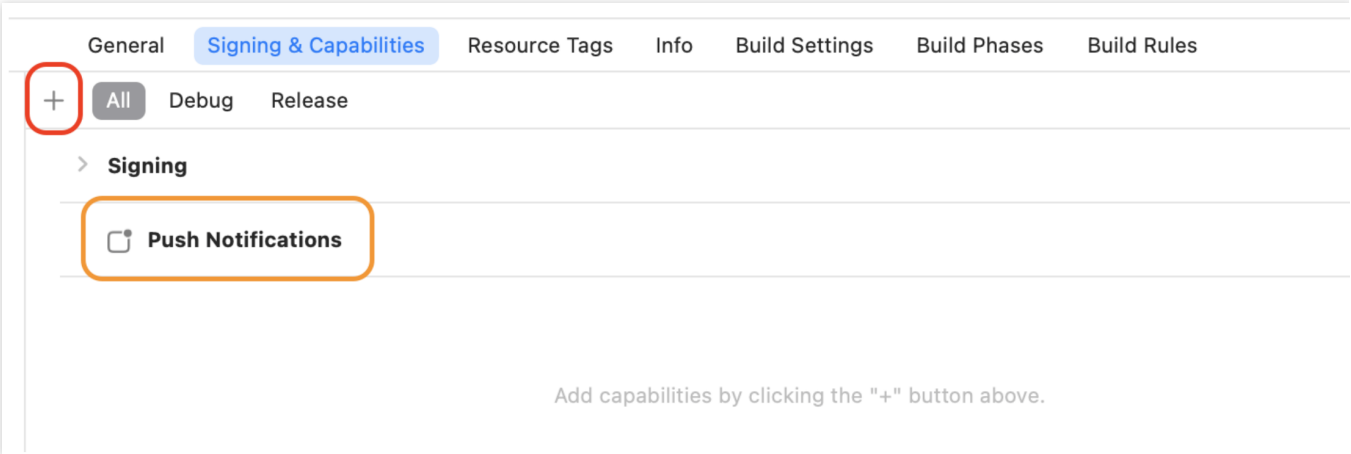
The uploaded .p12 certificate must be your own authentic and valid certificate.

5. After the push certificate information is generated, record the certificate ID. It will be used as a **mandatory parameter** in Step 6: Configure push parameters.



Step 5: Complete the project configuration

To add the required permissions to your application, enable the push notification feature in your Xcode project. Open the **Xcode** project, go to **Project** > **Target** > **Capabilities** page, click the **+** inside the red frame, then select and add **Push Notifications**. The result after adding is shown in the yellow frame in the picture:

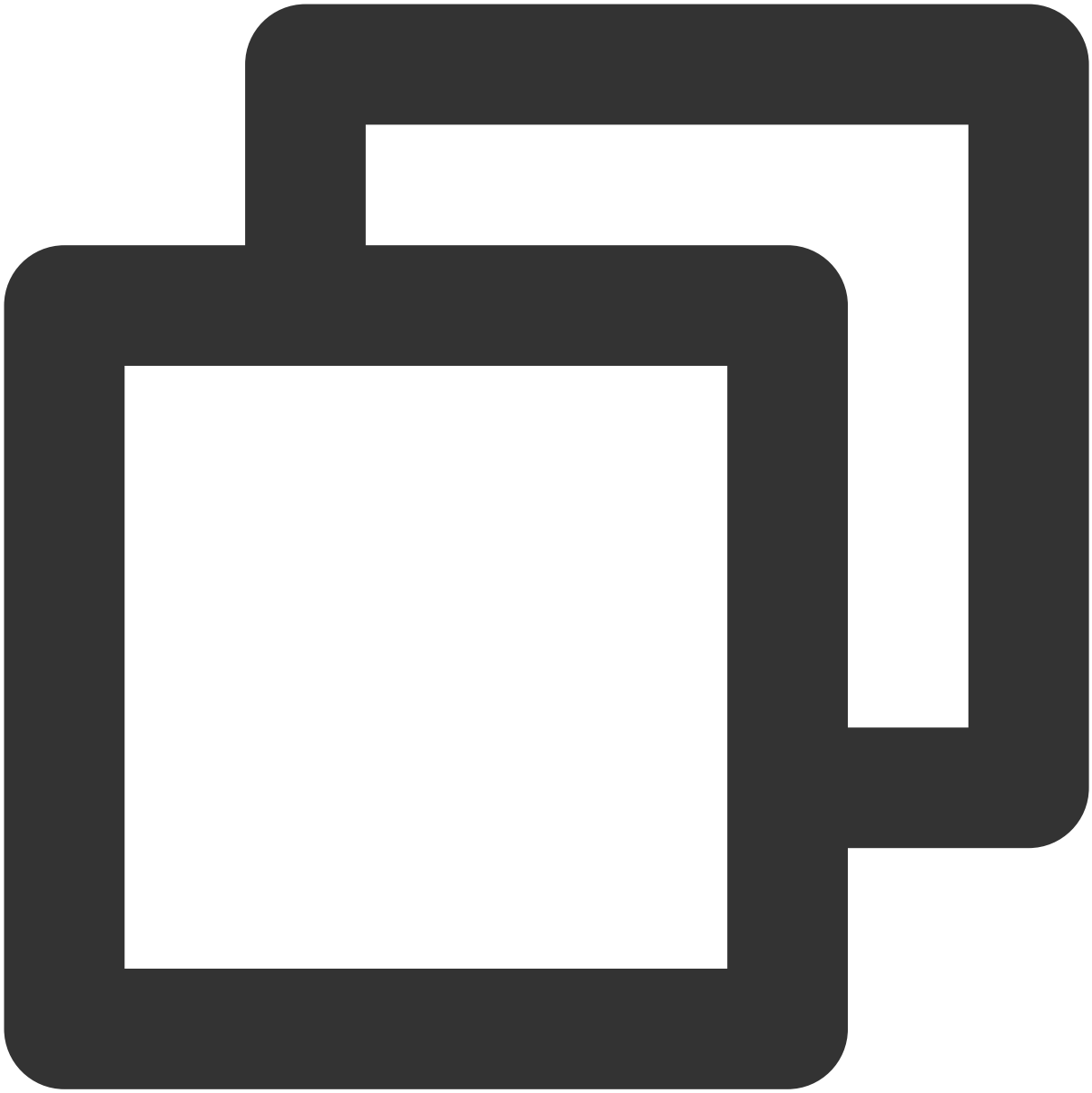


Step 6: Configure push parameters

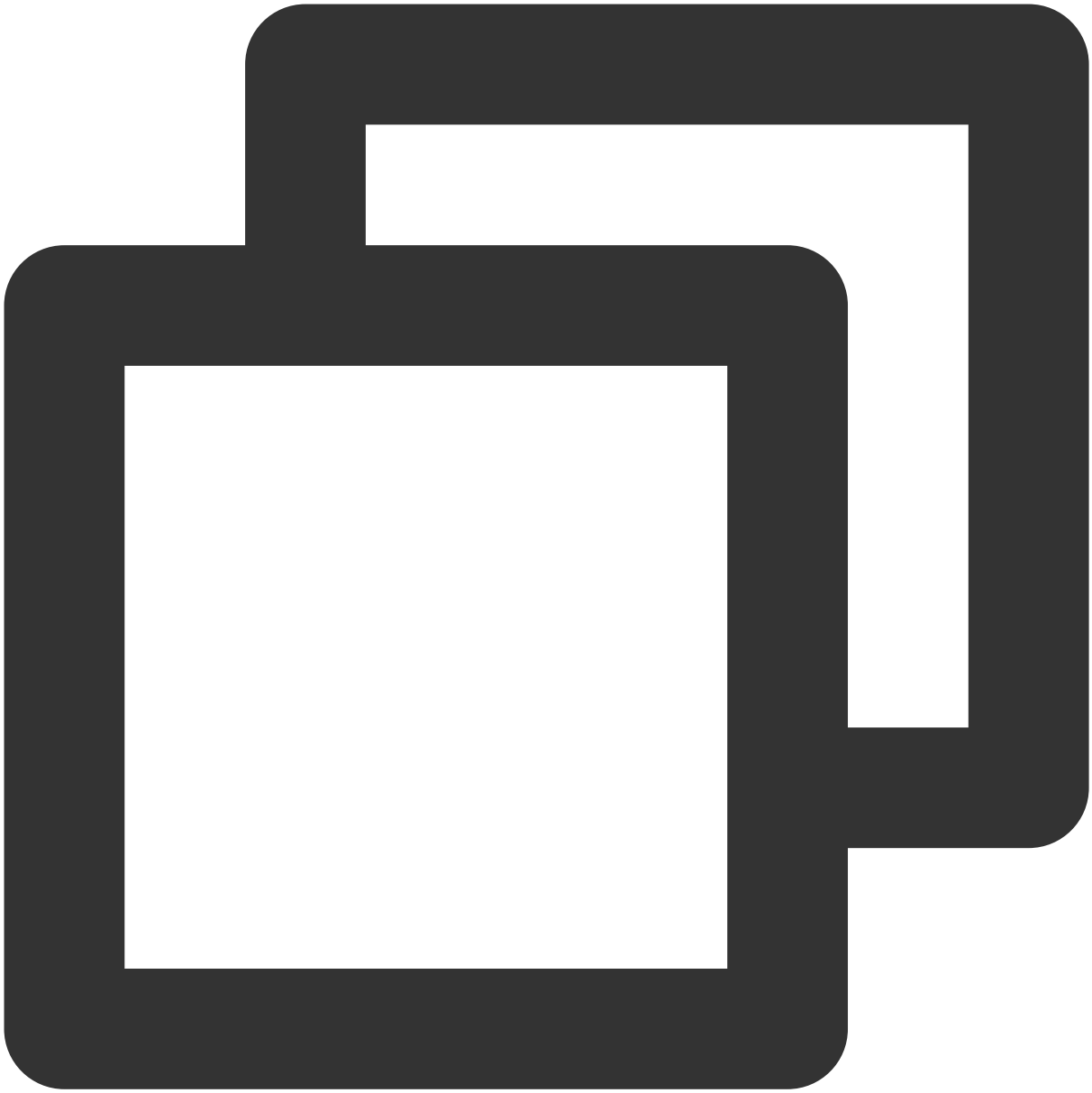
You need to implement the protocol method `offlinePushCertificateID` in AppDelegate to return the Certificate ID.

Swift

Objective-C



```
import TIMPush

func offlinePushCertificateID() -> Int32 {
    return kAPNSBusiId
}
```



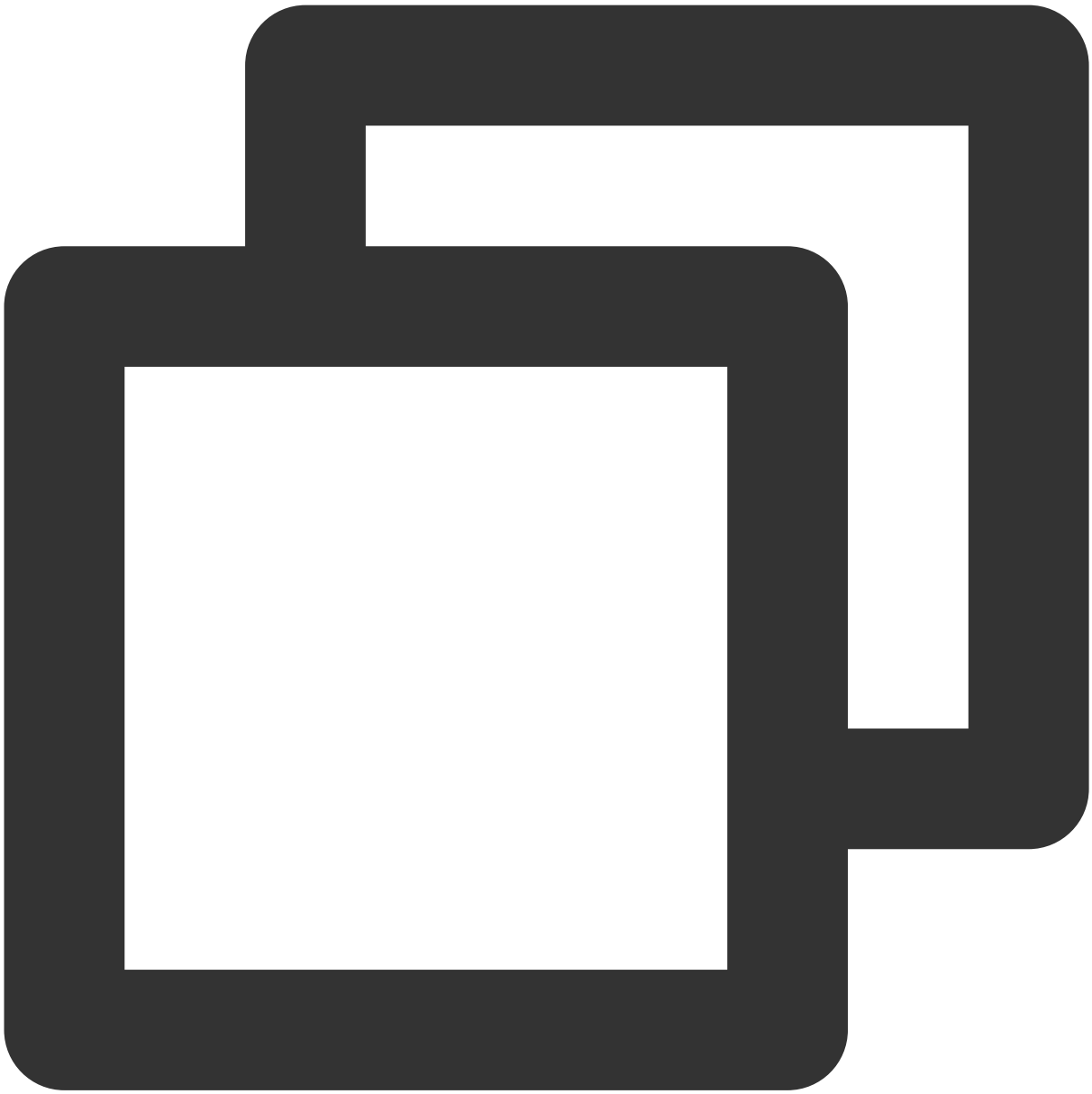
```
#pragma mark - TIMPush

- (int)offlinePushCertificateID {
    return kAPNSBusiId;
}
```

Step 7: Dial offline push calls

Please set the offlinePushInfo field of params when making a call using call or groupCall.

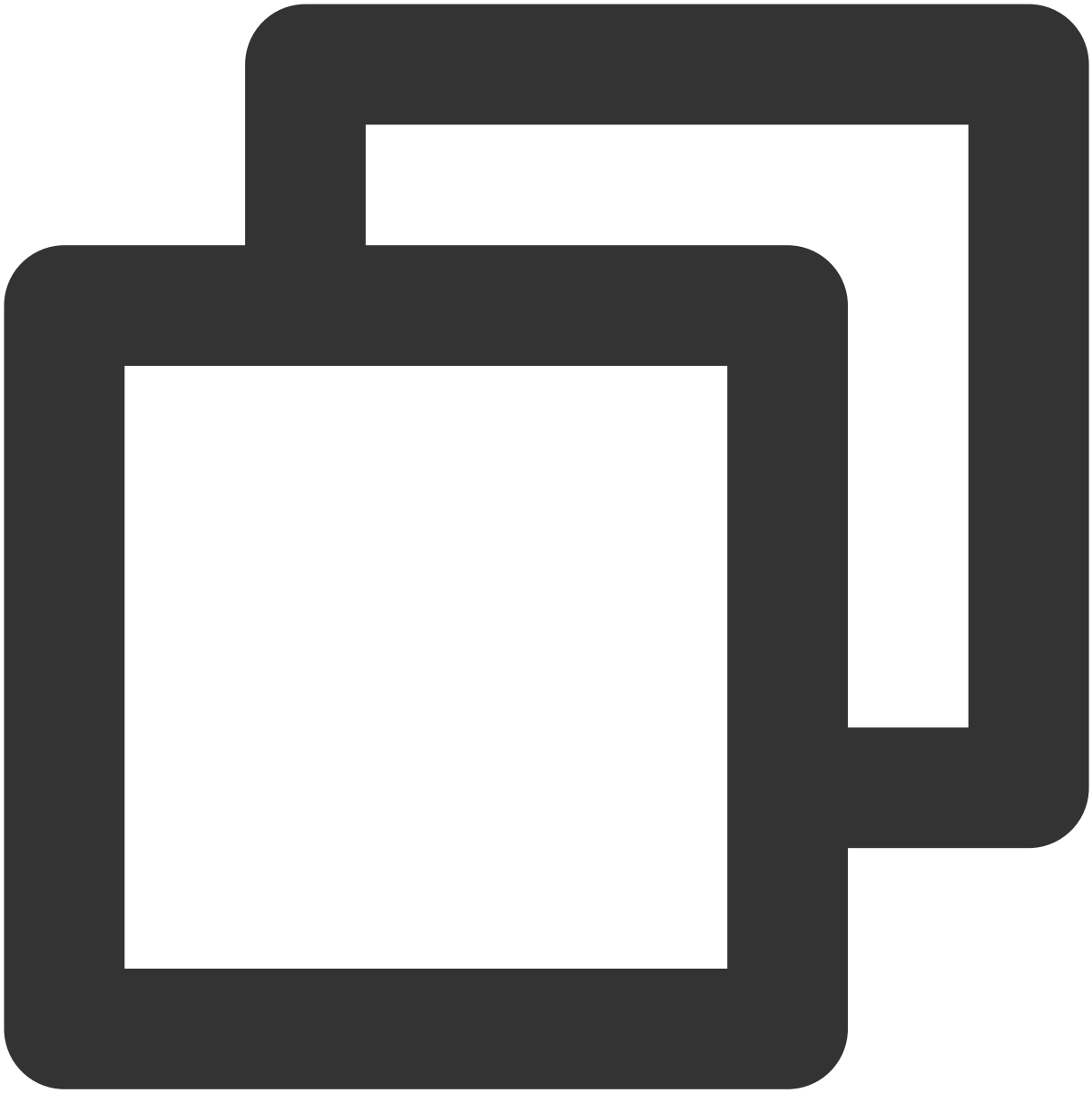
Swift

Objective-C



```
import TUICallKit_Swift
import TUICallEngine

let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = ""
pushInfo.desc = "You have a new call"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
```

```
pushInfo.androidSound = "phone_ringing"
// For OPPO, you must set the `ChannelID` to receive push messages. The `ChannelID`
// OPPO must set a ChannelID to receive push messages. This channelID needs to be t
pushInfo.androidOPPOChannelID = "tuikit"
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HM
pushInfo.androidHuaWeiCategory = "IM"

let params = TUICallParams()
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo

// Single person call example, similar for group call
TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: p

} fail: { code, message in

}
```



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <TUICallEngine/TUICallEngine.h>

- (TUICallParams *)getCallParams {
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}
```

```
- (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = @"You have a new call";
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeAPNs;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // For OPPO, you must set the `ChannelID` to receive push messages. The `Channe
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChanne
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery r
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/developmen
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}

// Single person call example, similar for group call
[[TUICallKit createInstance] callWithUserId:@"123456"
                                callMediaType:TUICallMediaTypeAudio
                                       params:[self getCallParams] succ:^{

} fail:^(int code, NSString * _Nullable errMsg) {

}];
```

FAQs

Not receiving push notifications, and the backend reports a bad deviceToken?

Apple's deviceToken is related to the current compilation environment. If the certificate ID and token used to log in to IMSDK and upload the deviceToken to Tencent Cloud do not match, an error will occur.
If compiled in the Release environment, the `-application:didRegisterForRemoteNotificationsWithDeviceToken:` callback returns a token for the deployment environment. In this case, the businessID should be set to the production environment's Certificate ID.
If compiled in the Debug environment, the `-application:didRegisterForRemoteNotificationsWithDeviceToken:` callback returns a token for the development environment. In this case, the businessID should be set to the development environment's Certificate ID.

In the iOS development environment, is there an occasional lack of return for the deviceToken or a failure in the APNs request for the token?

This problem is caused by instability of APNs. You can fix it in the following ways:

1. Insert a SIM card into the phone and use the 4G network.
2. Uninstall and reinstall the application, restart the application, or shut down and restart the phone.
3. Use a package for the production environment.
4. Use another iPhone.

Android

Last updated：2024-07-24 10:20:03

The TIMPush plugin is a powerful paid feature that brings VoIP call notification mechanisms to the Google platform. Combined with the data messaging capabilities of `Firebase Cloud Messaging (FCM)` and the `TUICallKit` component, it provides an incoming call display interface with a customizable layout.

Note:

Phones with Google Mobile Services.

The FCM data message feature is only available in TIMPush **7.9.5668** and later versions.

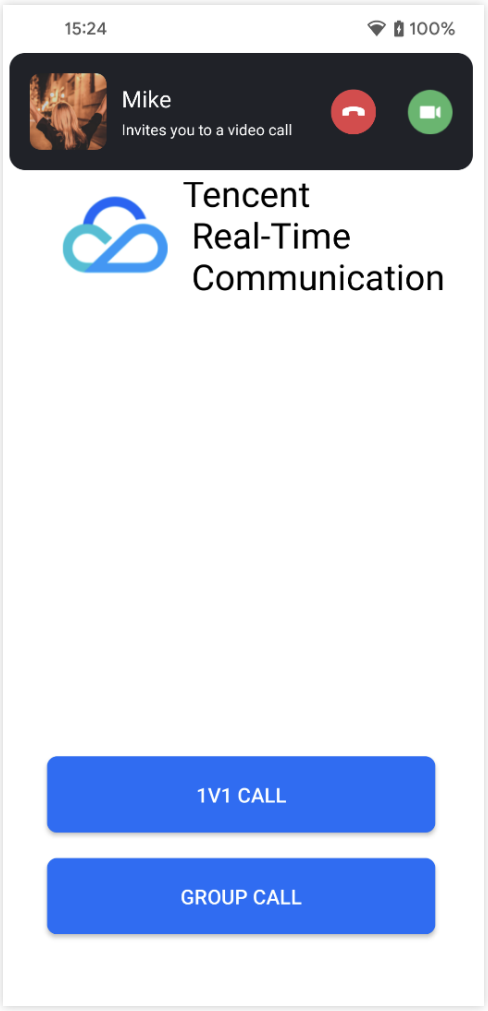
Support for TUICallKit is limited to **2.3** and later versions.

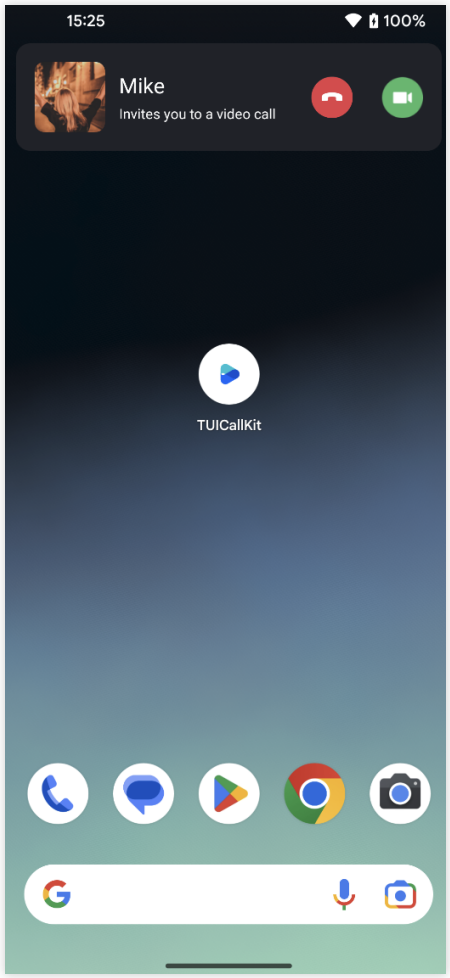
This article provides a detailed introduction on how to integrate the TIMPush plugin into the TUICallKit component, utilize FCM's data messages, and achieve banner views for audio and video incoming calls.

Integration effect

TUICallKit has successfully integrated FCM data messages on the GitHub sample project. You can quickly implement the feature's normal operation by referring to the Call UIKit sample project. The following diagram shows the incoming call views after receiving an call invitation when the **application is in the foreground, background**, or when the **application process does not exist.**

The display view when the **application is in the foreground**	The display view when the **application is in the background** or **offline**





Note:

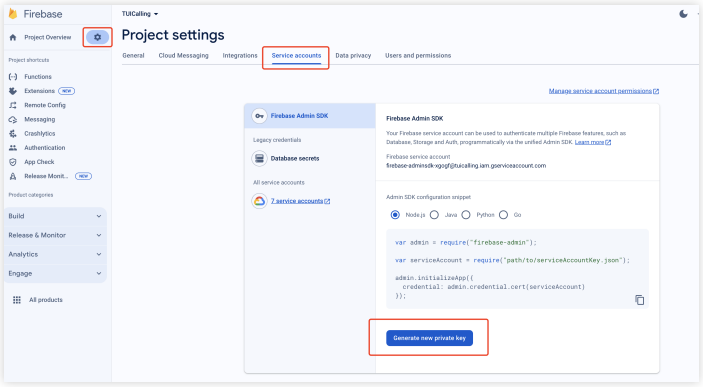
To achieve a good call experience as shown above, it is recommended to enable "notification", "Display over other apps" and "Background pop-ups" permissions in your application. For more details, refer to: Relevant permissions enabled.

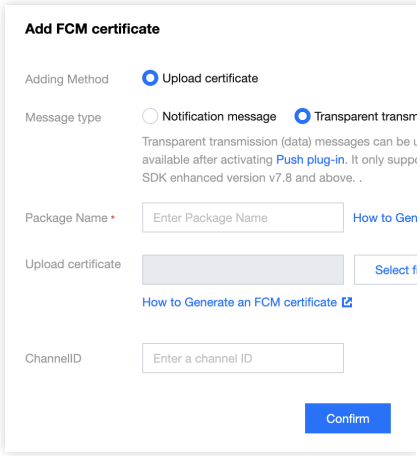
Preparation Requirements

1. Register your application to the FCM Push Platform to obtain parameters such as **AppID** and **AppKey**, as well as the `google-services.json` file.

2. log in to the Instant Messaging Chat Console, go to **Push Management** > **Access Settings** feature tab, select FCM, add FCM's certificate, and select **Transparent transmission(data) message**.

FCM Platform	Configuring in the Chat console

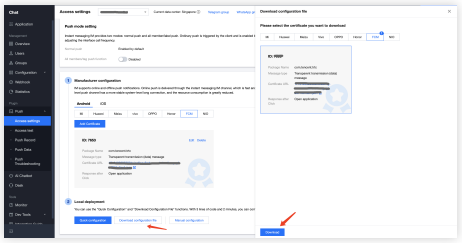
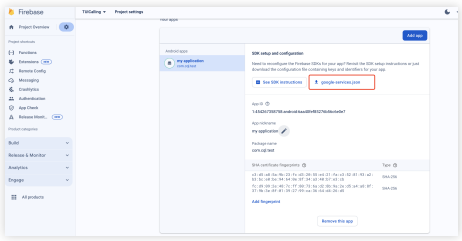




Quick Integration

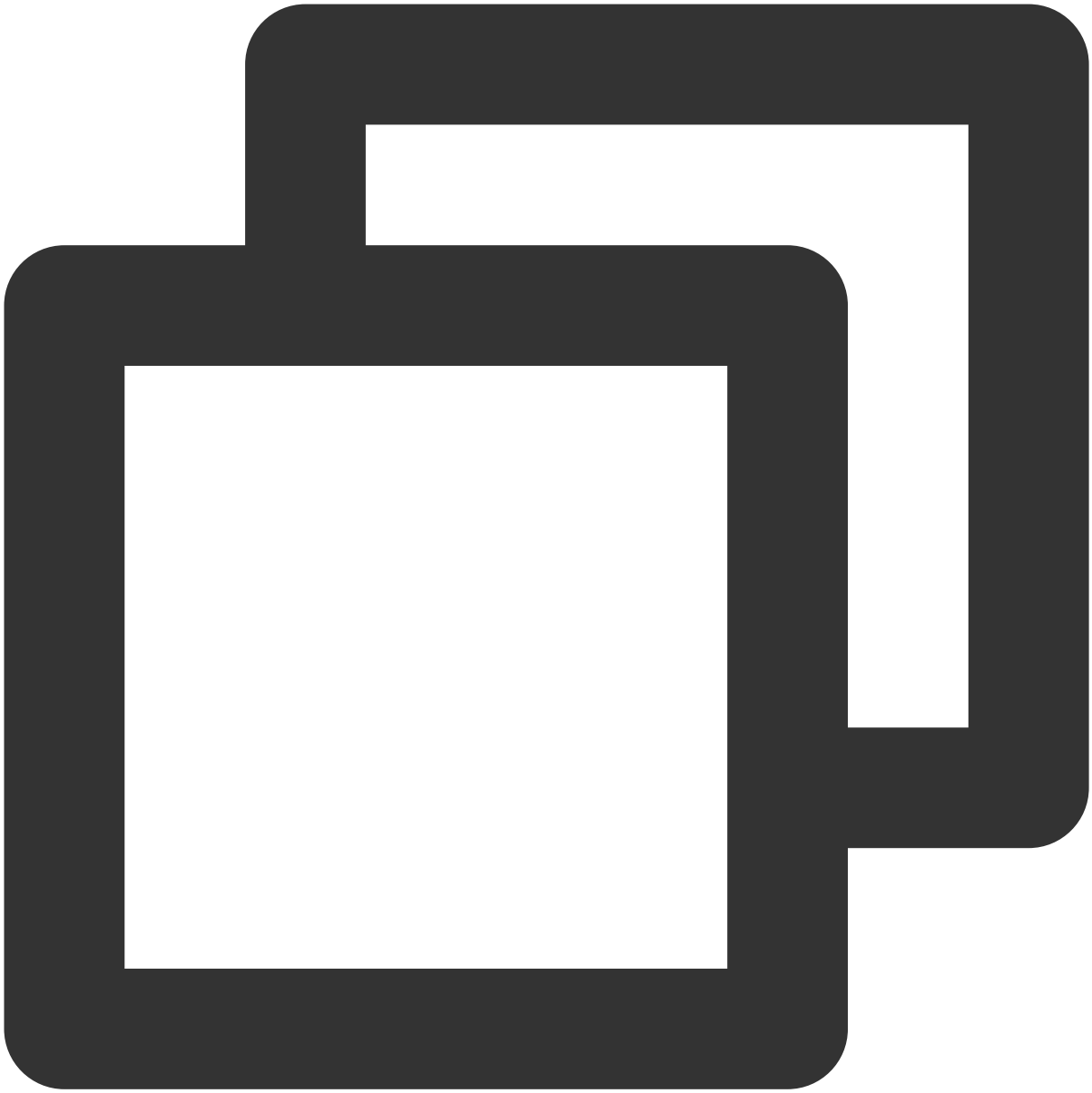
1. Download and add the configuration file

After completing the push information in the console, download `timpush-configs.json` file and add the file to the `assets` directory of the application module, and add the `google-services.json` to the project app directory.

Download the file timpush-configs.json	Download the file google-services.json	Add t
		

2. Integrate the TIMPush plugin

In the `app/build.gradle` file, add the following dependency.



```
implementation "com.tencent.timpush:timpush:latest.release"
implementation "com.tencent.timpush:fcm:latest.release"
```

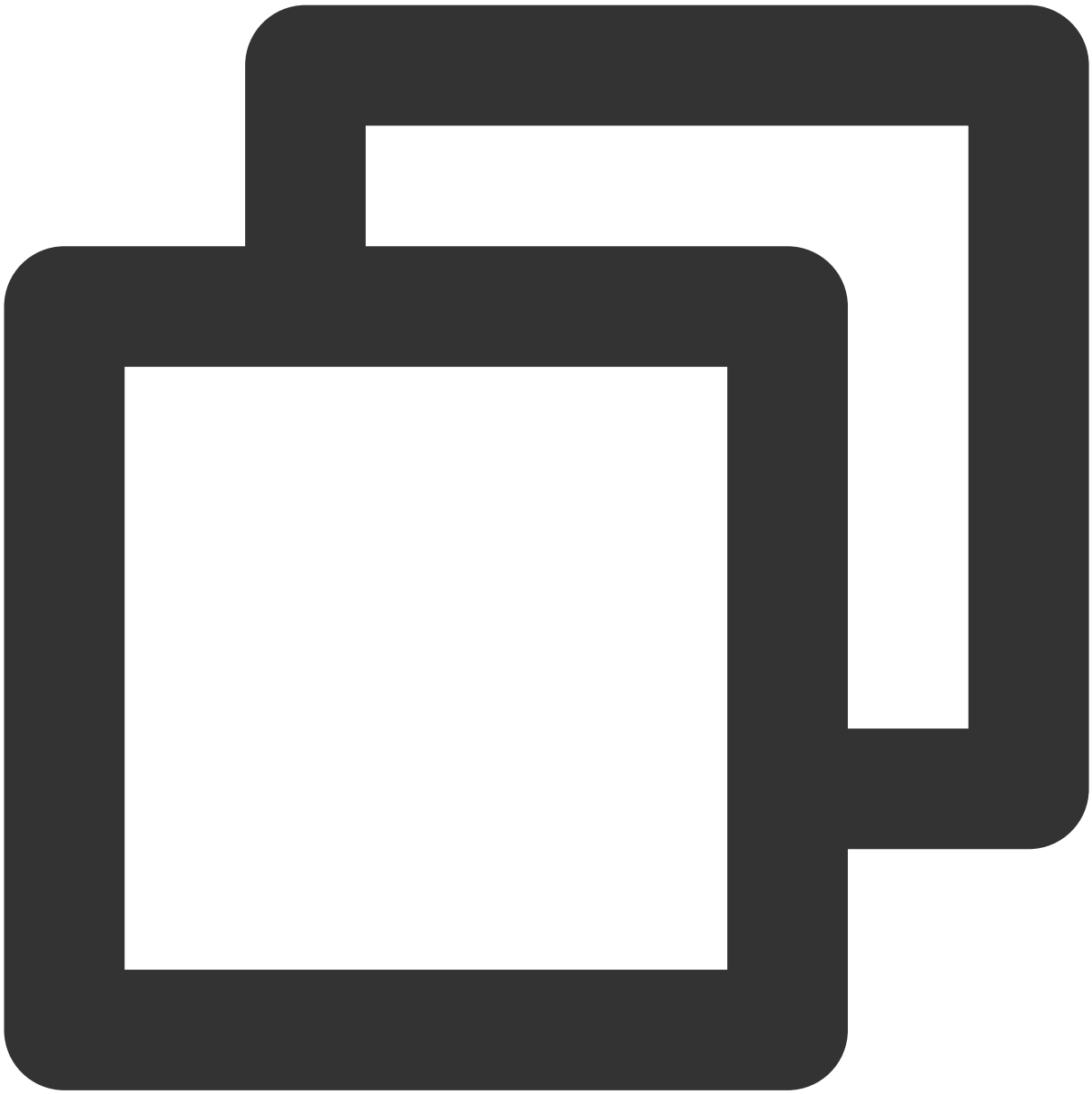
Note:

TIMPush requires integration with Chat SDK in version **7.9.5668** and above.

You can modify the version of the Chat SDK in the `tuicallkit-kt/build.gradle` file.

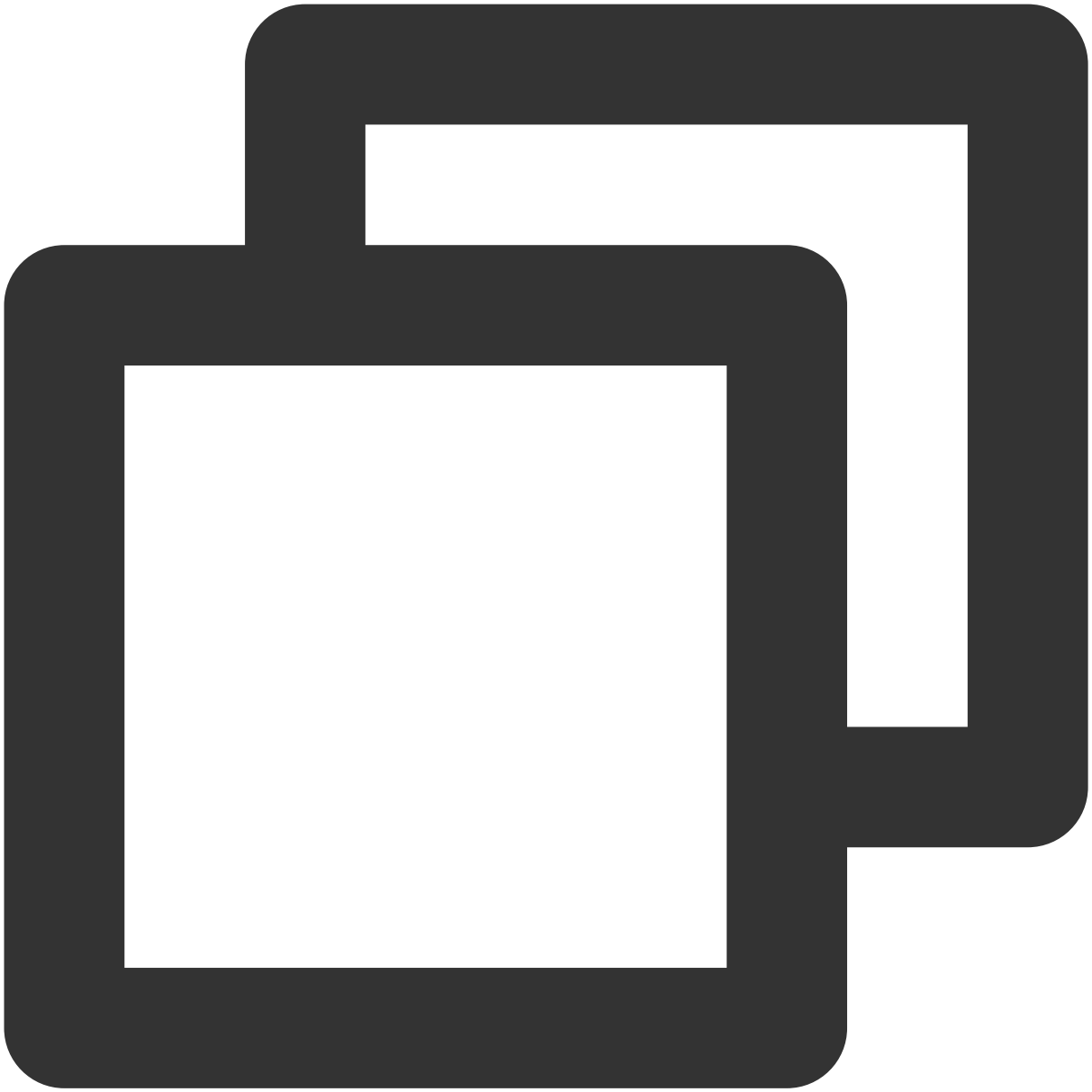
3. Complete project configuration

In the project-level `build.gradle` file, under `buildscript -> dependencies` section, add the following configuration.



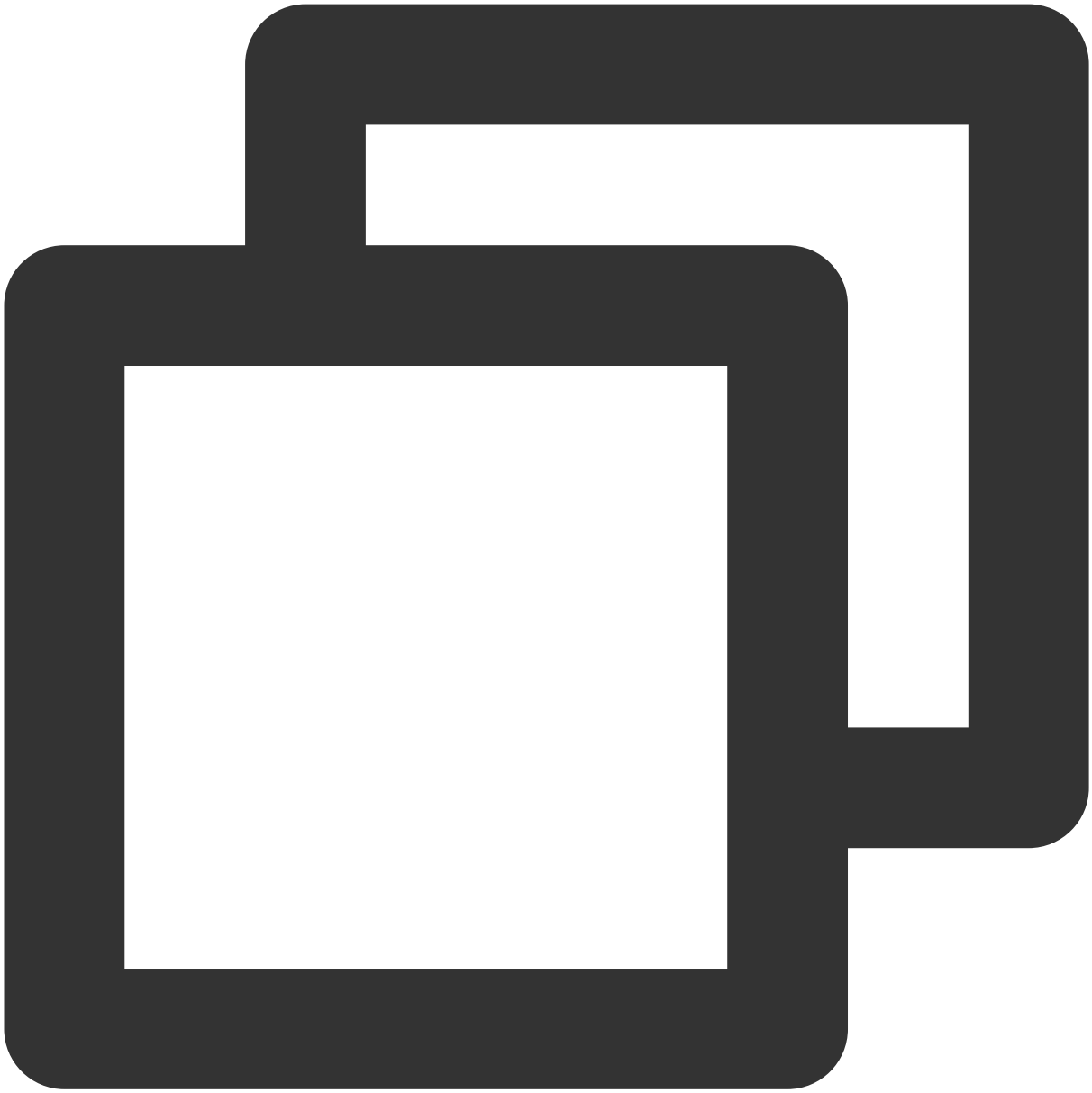
```
buildscript {
    dependencies {
        classpath 'com.google.gms:google-services:4.3.15'
    }
}
```

In the `app/build.gradle` file, add the following configuration.



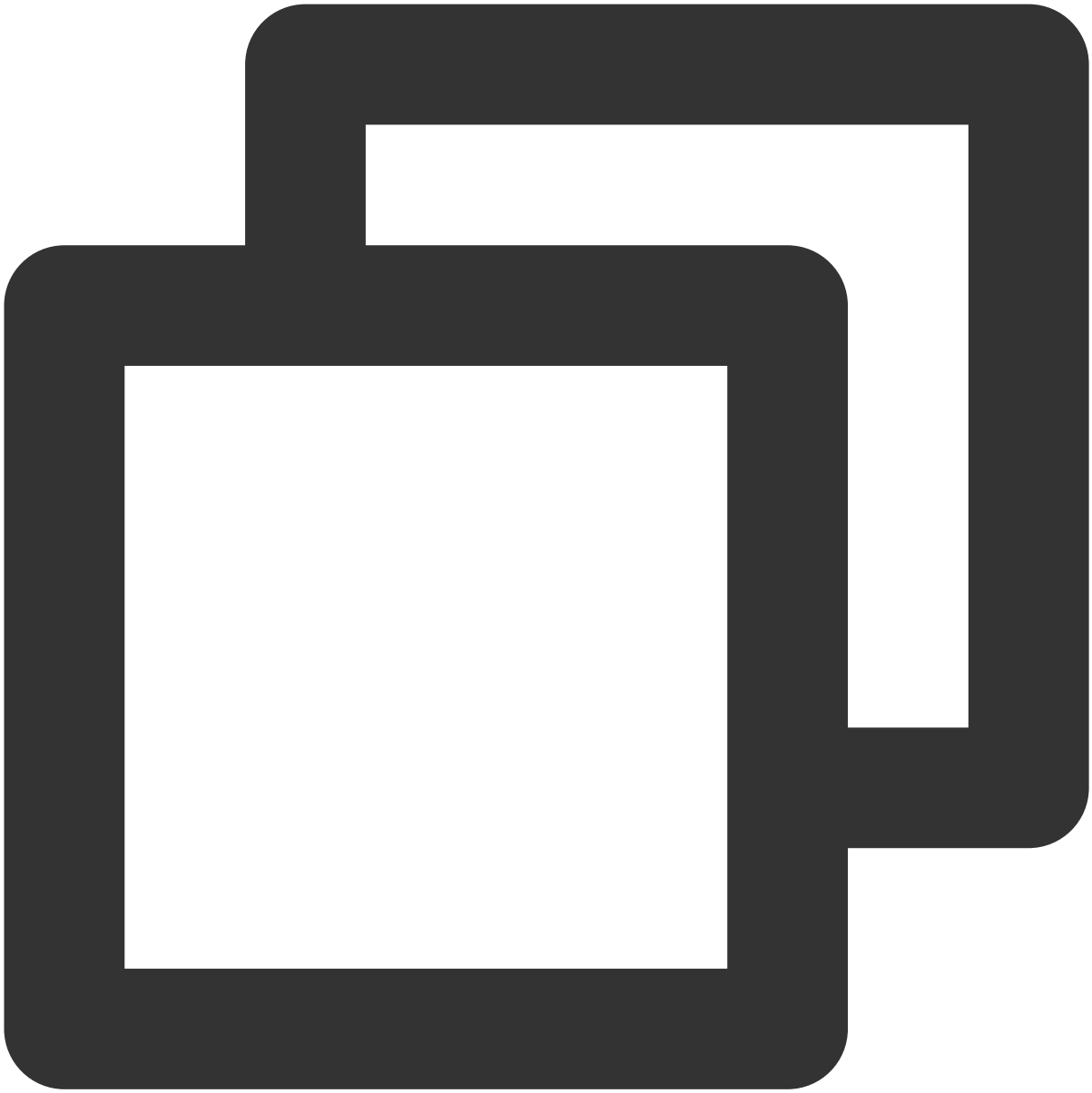
```
apply plugin: 'com.google.gms.google-services'
```

In the `app/proguard-rules.pro` file, add TIMPush related classes to the non-aliasing list.



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

In the `app/build.gradle` file, change the application package name to your actual application package name.



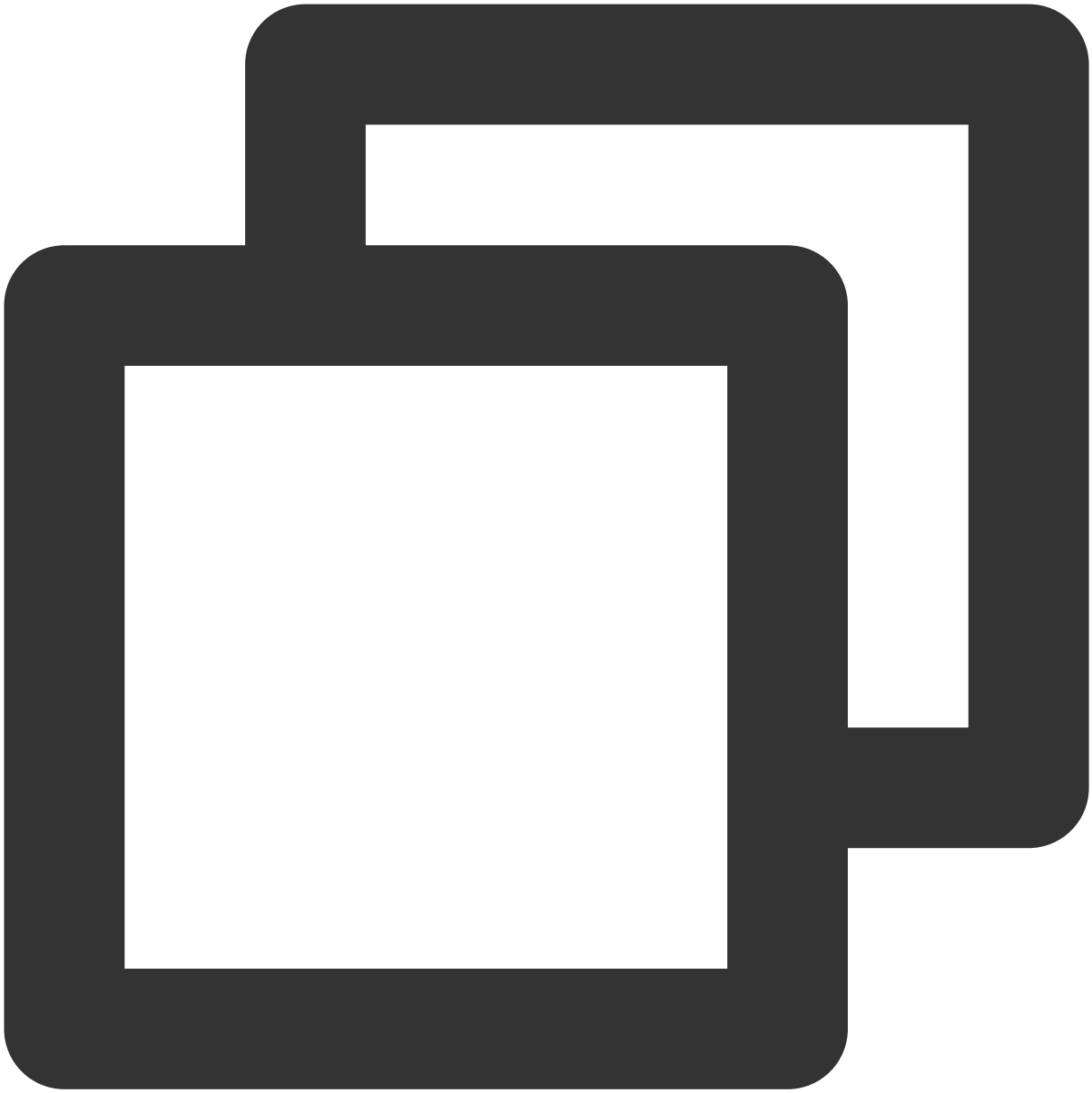
```
applicationId 'com.****.callkit'
```

4. Comlete automatic login

In your `application` class, listener to the TIMPush event and implement automatic login method by your self.

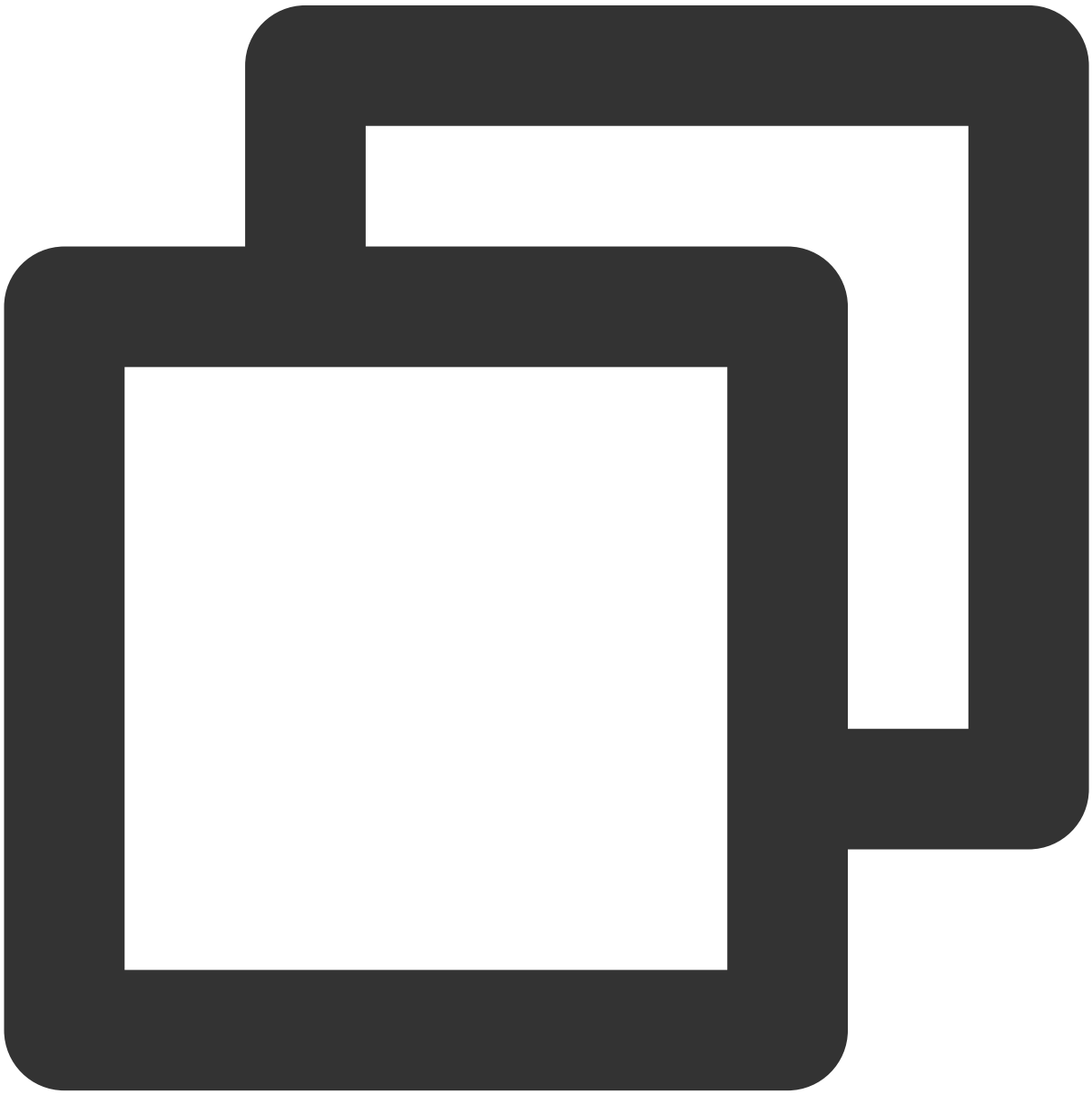
Kotlin

Java



```
import com.tencent.qcloud.tuicore.TUIConstants
import com.tencent.qcloud.tuicore.TUICore
import com.tencent.qcloud.tuicore.interfaces.ITUINotification

class BaseApplication : Application() {
    override fun onCreate() {
        super.onCreate()

        TUICore.registerEvent(TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_
            TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_SUB_KEY) { key, su
```

```
            if (TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_KEY == key
                && TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_SUB_KEY ==
                //you need to login again to launch call activity, please implement
                autoLogin()
            }
        }
    }
}
```



```
import com.tencent.qcloud.tuicore.TUIConstants;
import com.tencent.qcloud.tuicore.TUICore;
import com.tencent.qcloud.tuicore.interfaces.ITUINotification;

public class BaseApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

        TUICore.registerEvent(TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_
                TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_SUB_KEY, new I
                    @Override
                    public void onNotifyEvent(String key, String subKey, Map<String
                        if (TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_KE
                                && TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WA
                            //you need to login again to launch call activity, plea
                            autoLogin();
                        }
                    }
                });
    }
}
```

After completing the above steps, you can use the offline push capability in TUICallKit.

Note:

If your Android application encounters issues when receiving push notifications or pulling up pages, you can refer to the Called party's call display policy for troubleshooting.

Advanced features

Custom Ringtone

If you wish to customize the ringtone, you can replace the `tuicallkit-kt/src/main/res/raw/phone_ringing.mp3` file.

Note:

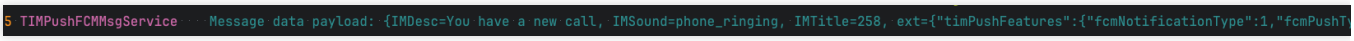
After replacing the ringtone, no matter if the application is in the foreground, background, or offline, the ringtone will be the one that has been replaced.

After replacement, the ringtone received by other manufacturer's mobile phones upon invitation will also be the replaced one.

FAQs

1. Unable to pop up the incoming call UI after the application is killed

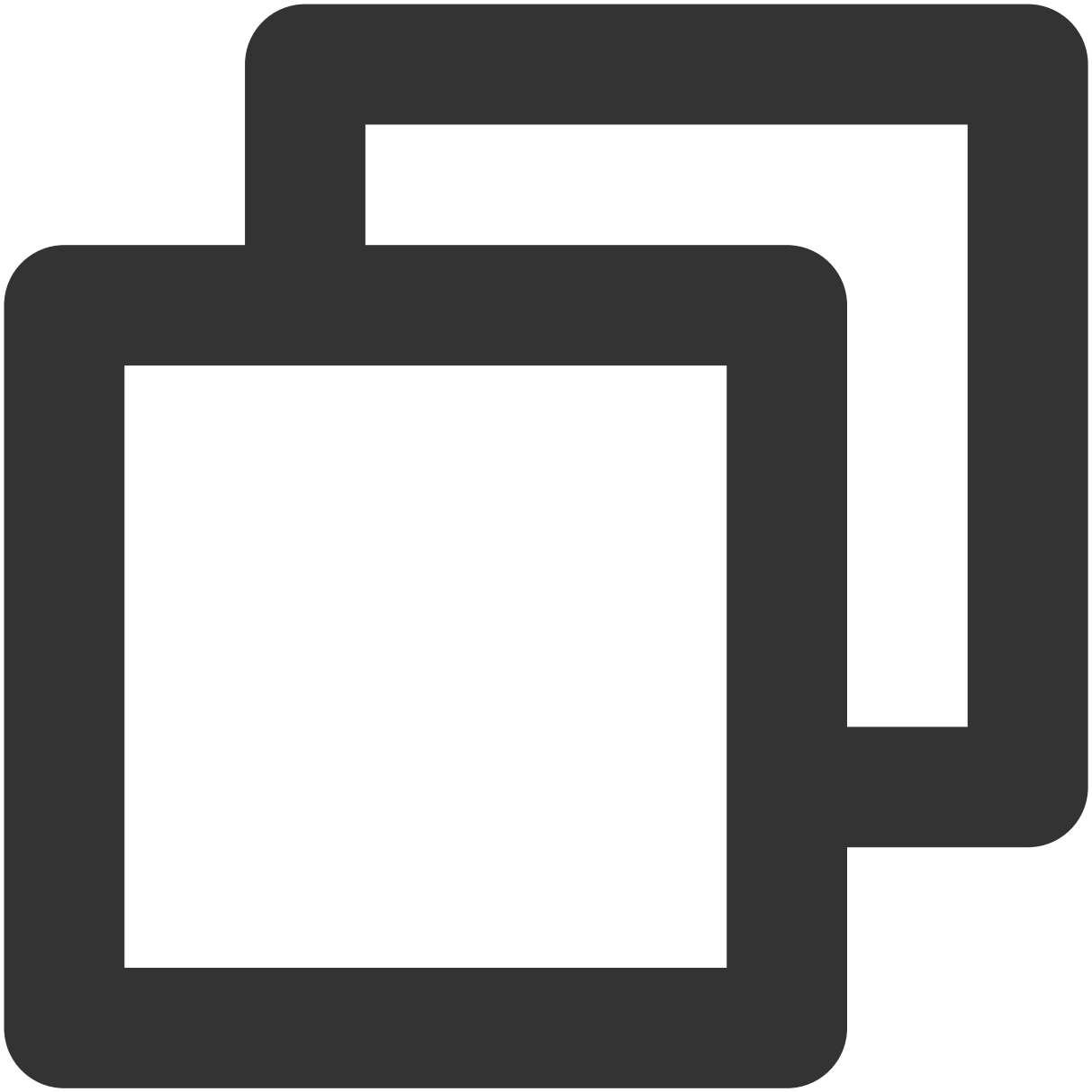
Confirm receipt of push. If push cannot be received, refer to step 1 in the above document **Quick Integration** to see if the certificates have been correctly added to the Chat cosole.
Make sure the console has selected FCM **Transparent transmission(data) message**, in accordance with the second step of the above **Preparation Requirements.**
Confirm receipt of data messages, filter logs (keyword: TIMPush), and check the following logs for printing (if messages are not received.



Confirm implementation of **automatic login**. It is only after **automatic login** that call requests are pulled and the incoming call UI is displayed.

2. Relevant permissions enabled

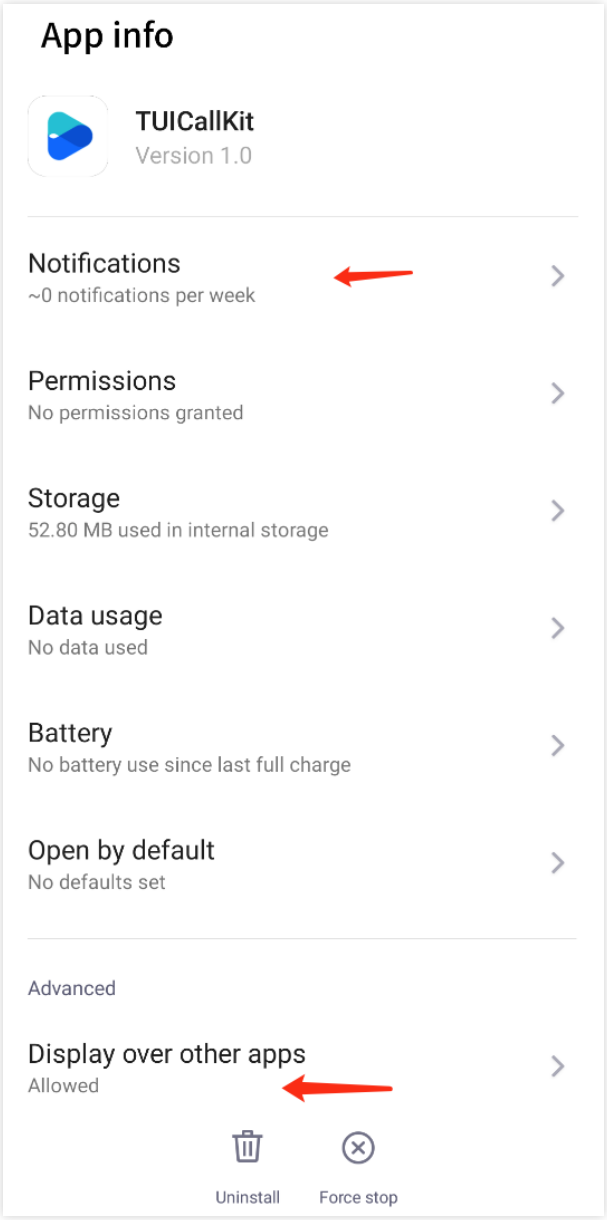
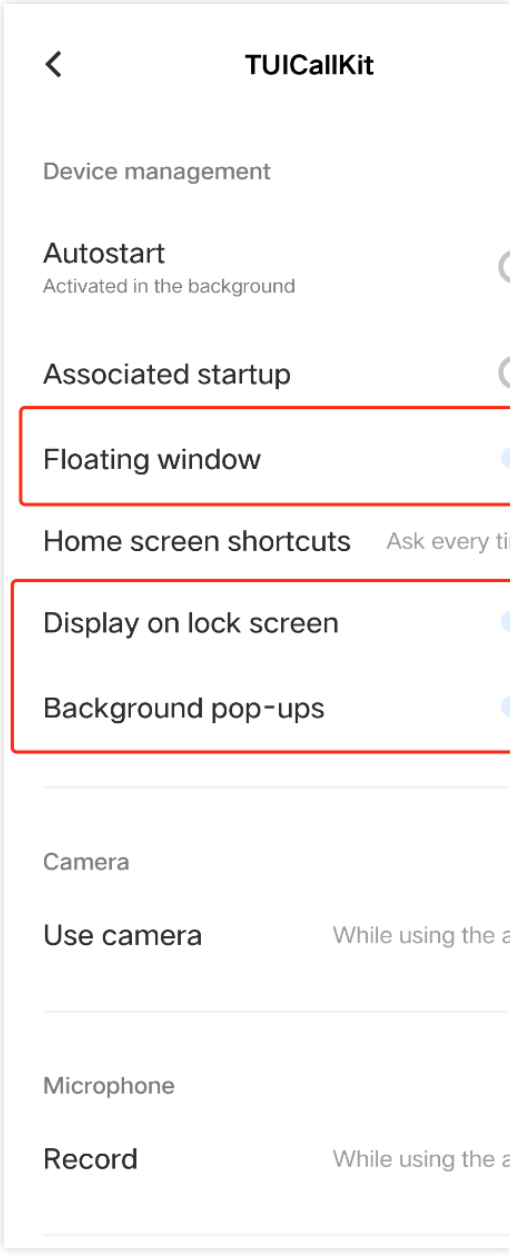
To ensure a good calling experience, it is recommended to enable "notification" permissions, "Display over other apps(Floating window)" and "Background pop-ups" permissions in your application. The specific methods are as follows:
Code guidance
Manually Enabled
Notification permissions: For showcasing push notifications: please refer to Notification runtime permissions and Request runtime permissions and implement according to business needs.
Floating window permission: Used for displaying custom incoming call notifications and call floating windows.
Background pop-ups: Used to pop up the interface when the application is in the background (for example: on a VIVO phone).



```
fun requestPermission(context: Context?) {
    //In TUICallKit,Please open both OverlayWindows and Background pop-ups permissi
    PermissionRequester.newInstance(
         PermissionRequester.FLOAT_PERMISSION, PermissionRequester.BG_START_PERMISS
        .request()
}
```

After the application is installed, you can long-press the application Icon, select "Application Information", then enable "notification" permission, "Display over other applications," and "Background pop-ups" permissions. Alternatively, you

can go to `Mobile Phone > System Settings > Application Management > Application` to manually enable these permissions.

Pixel 4a	VIVO
	

Suggestions and Feedback

If you have any suggestions or feedback, please contact `info_rtc@tencent.com` .

Flutter Notification

Last updated：2024-07-22 17:00:45

To help developers easily implement the offline push feature in Flutter projects, we recommend using the TIMPush plugin (paid). Compared to integrating separately on Android and iOS, using the TIMPush plugin offers the following advantages:

Short integration period, it is estimated that integration with all vendors only takes 30 minutes.

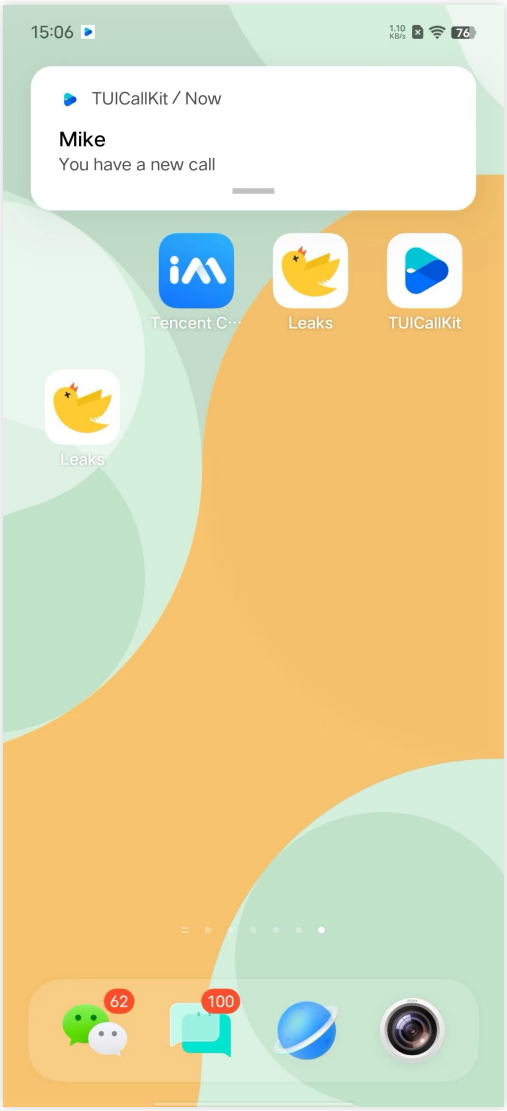
Supports data statistics and link tracking, making it easy for you to view various metrics such as reach rate, click-through rate, and conversion rate.

Supports All-staff/Tag Push, making it convenient to push marketing ads, notifications, news information, etc., to all users or specific groups.

Supports cross-platform frameworks like uni-app and Flutter.

This document will detail how to integrate the TIMPush plugin in the TUICallKit component to achieve offline push capability for audio and video calls.

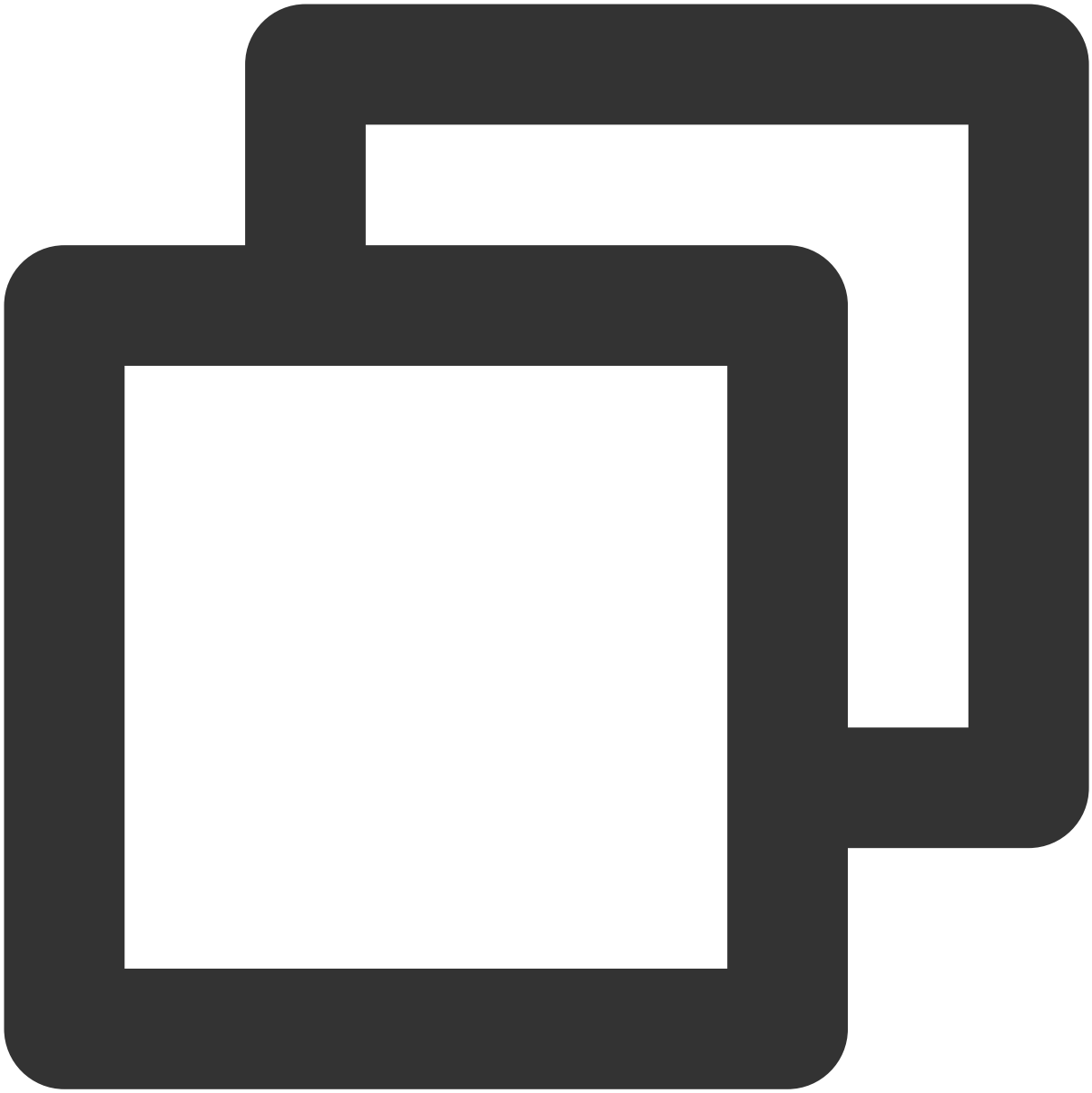
Android	iOS



Operation step

Step 1: Incorporate a message push plugin

This plugin's package name on pub.dev is: tencent_cloud_chat_push. You can include it in the pubspec.yaml dependencies directory, or execute the following command for automatic installation.



```
flutter pub add tencent_cloud_chat_push
```

Step 2: Manufacturer Configuration

iOS

Android

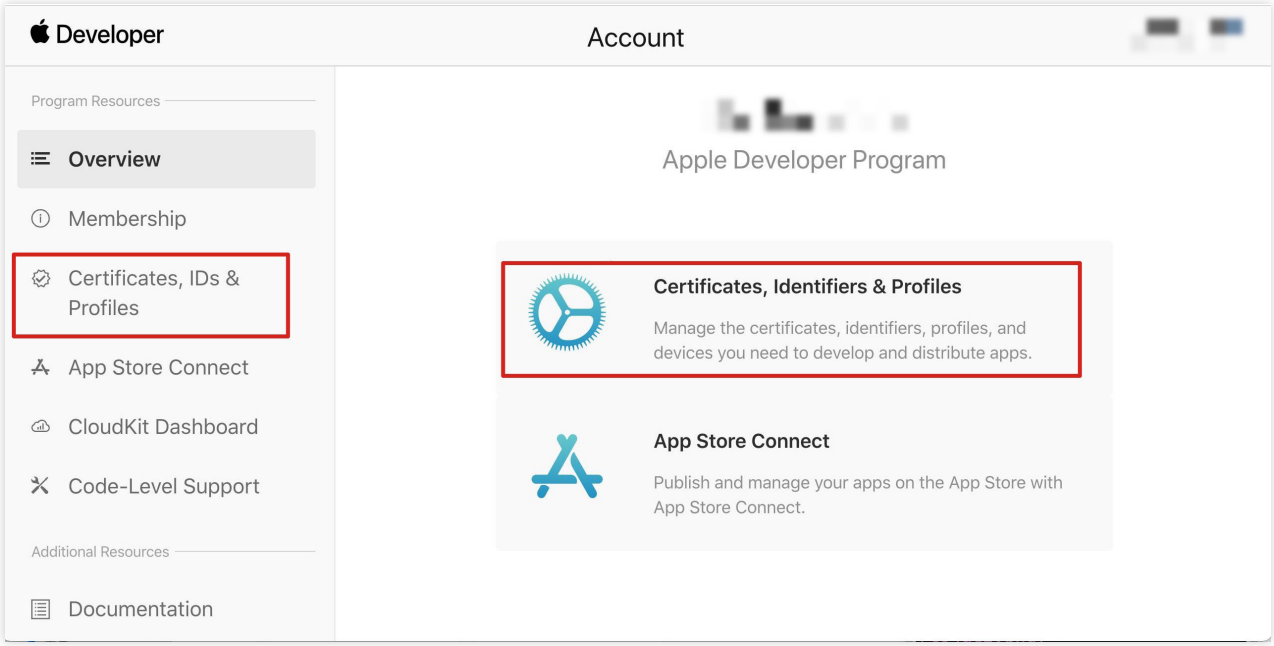
Before integrating the message push plugin, you need to apply to Apple for an APNs push certificate, then upload the push certificate to the Tencent RTC Console.

Operation step

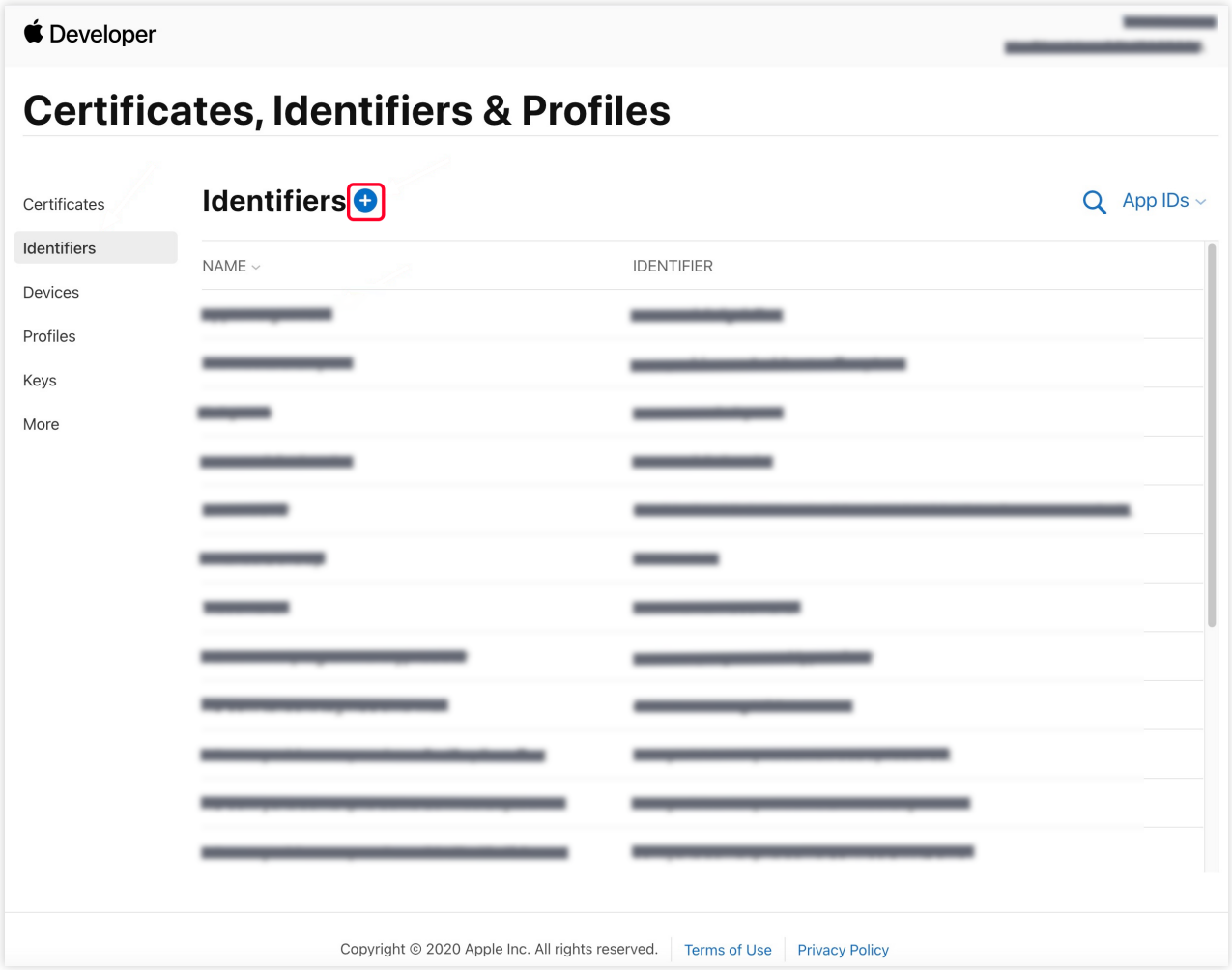
Step 1: Apply for an APNs certificate

Activate remote push for the App

1. log in to Apple Developer Center Web Sites, click **Certificates, Identifiers & Profiles** or the sidebar's **Certificates, IDs & Profiles**, to enter the **Certificates, IDS & Profiles** page.



2. click on the identifier's right **+**.

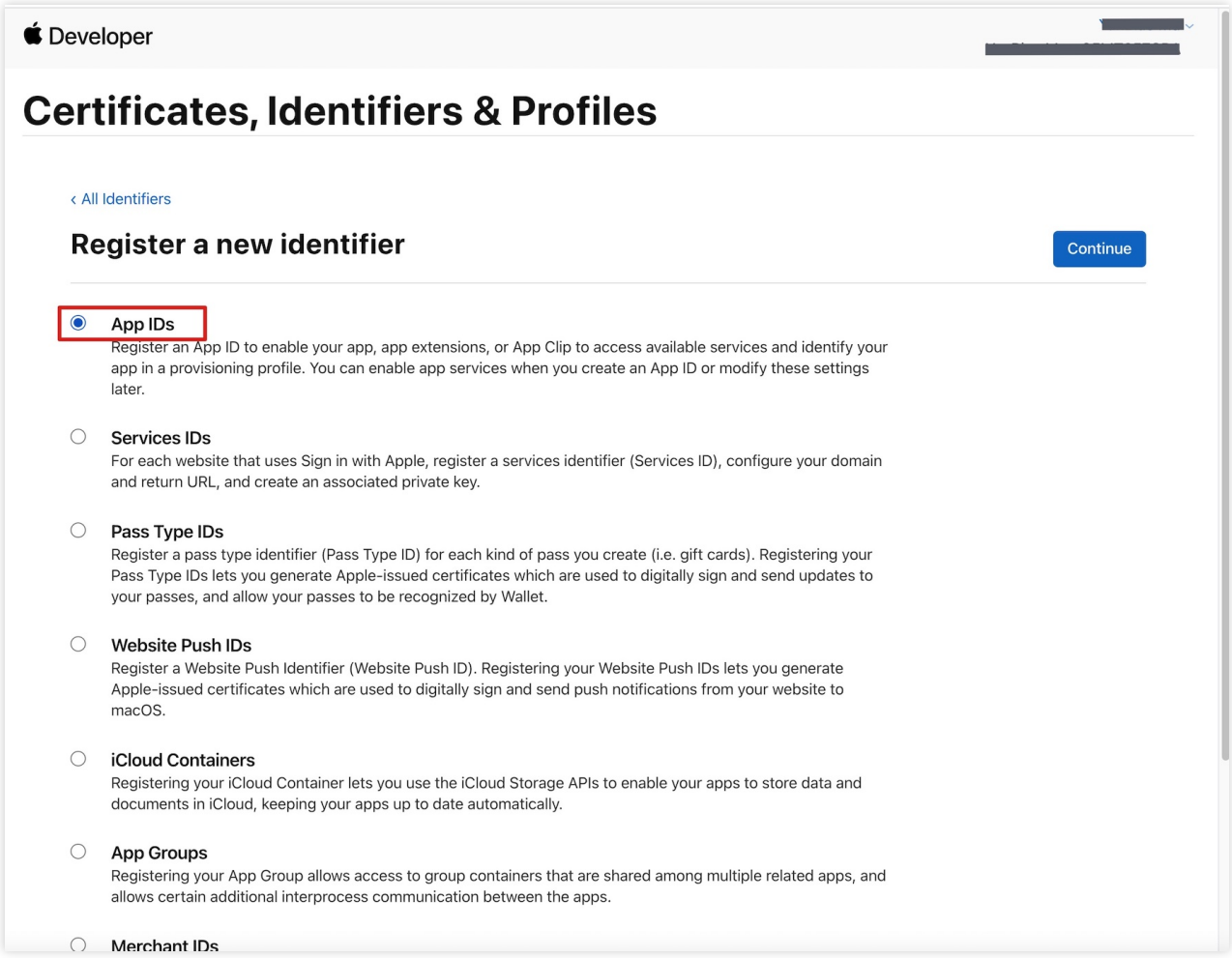


3. You can follow the steps below to create a new AppID, or add the `Push Notification Service` to your existing AppID.

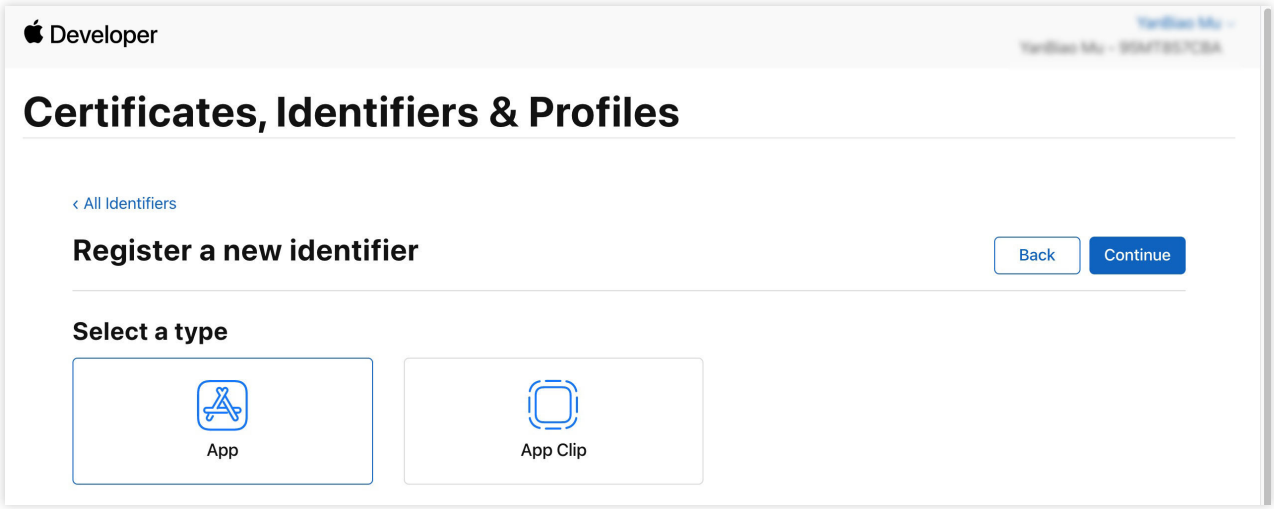
Note:

Your App's `Bundle ID` cannot use the wildcard `*`; otherwise, the remote push service cannot be used.

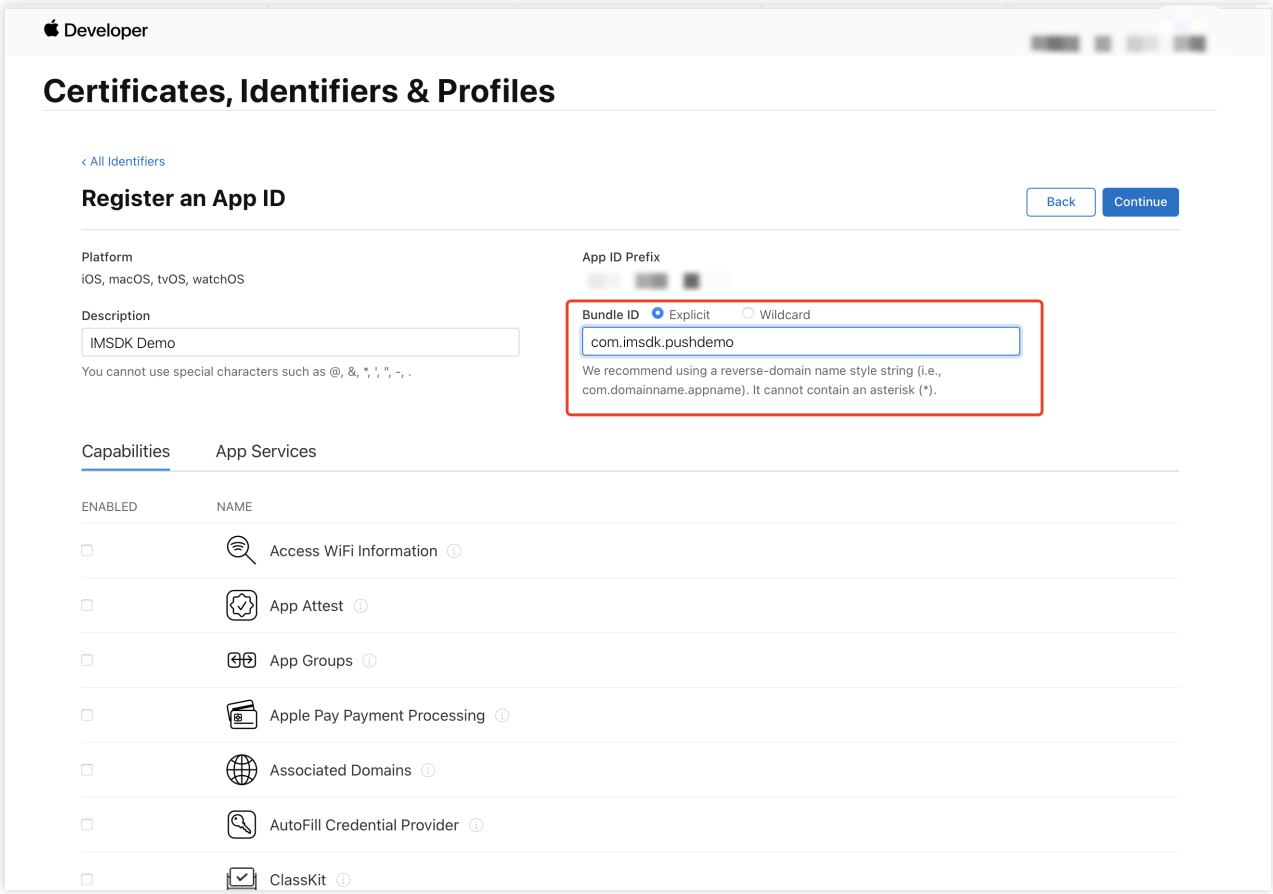
4. Select **App IDs**, click **Continue** to proceed to the next step.



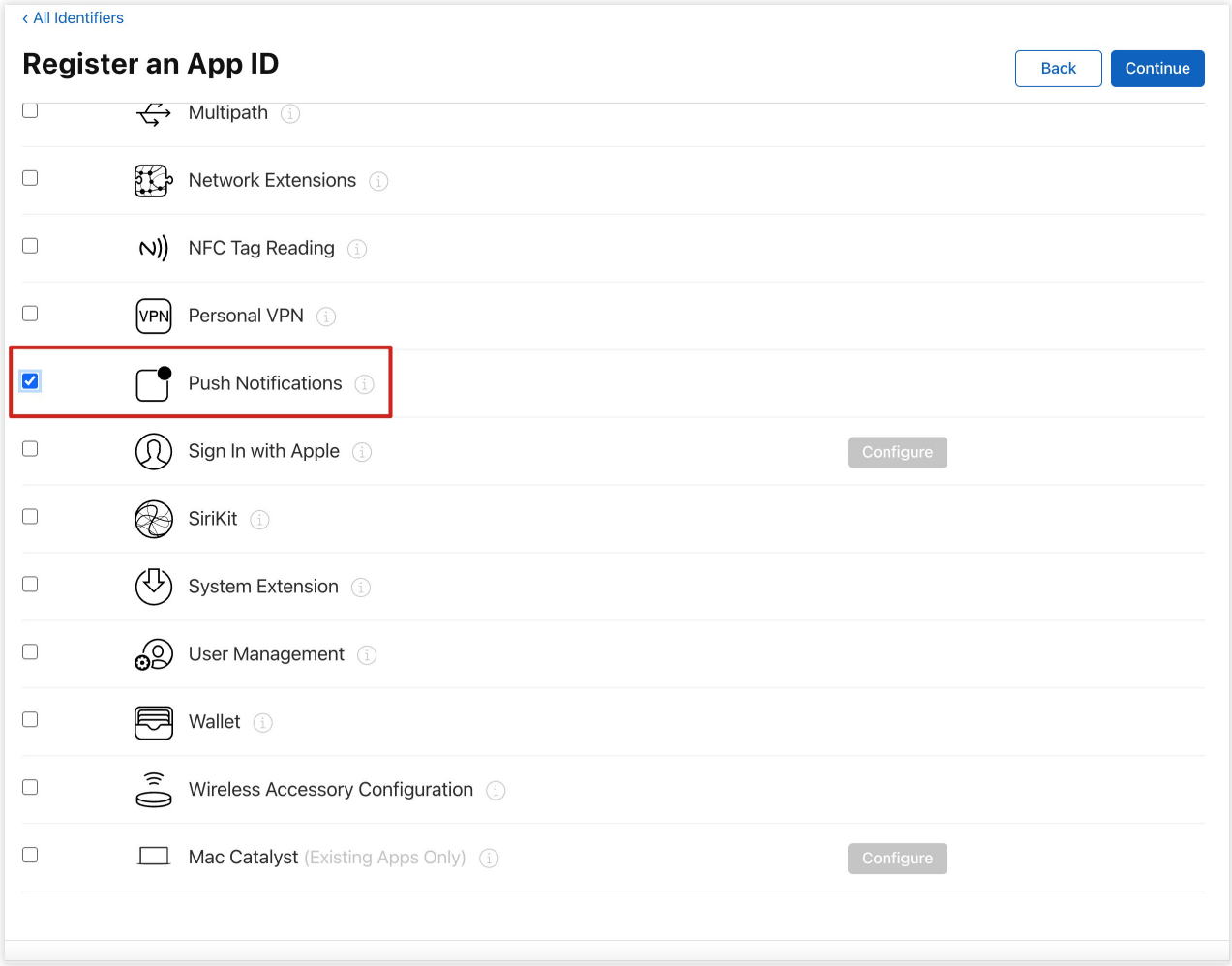
5. Select **App**, click **Continue** to proceed to the next step.



6. Configure `Bundle ID` and other information, click **Continue** to proceed to the next step.

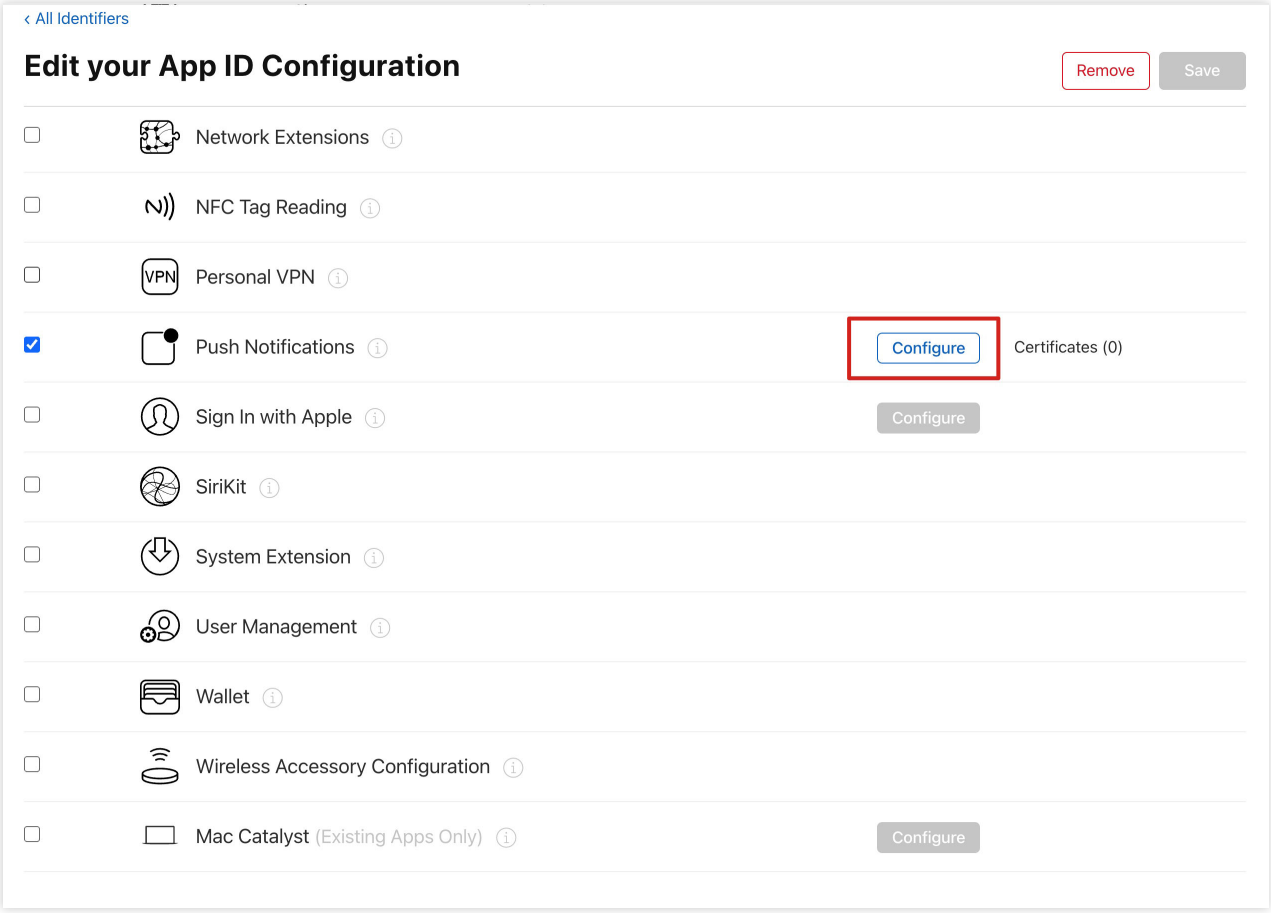


7. Select **Push Notifications** to activate the remote push service.

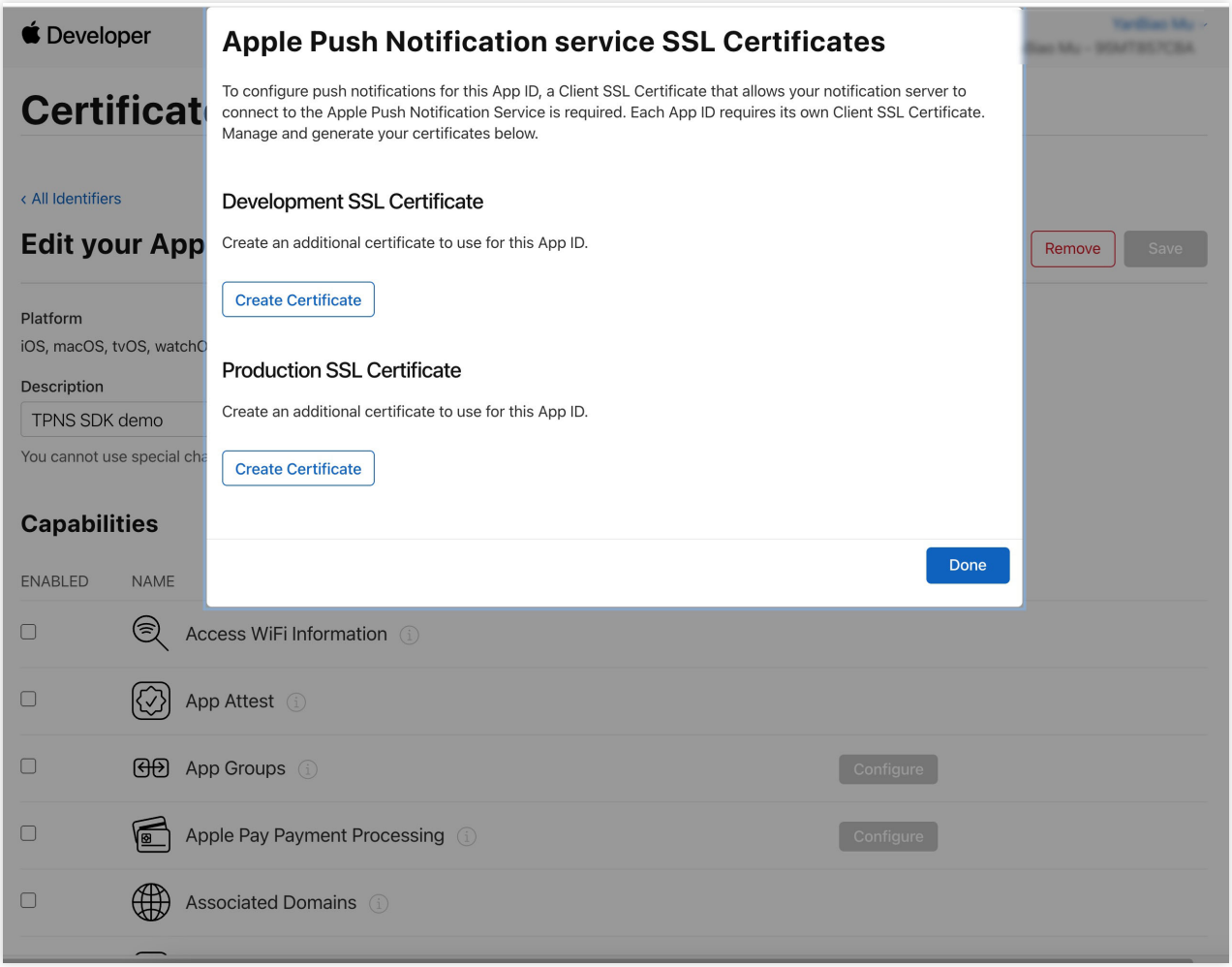


Certificate Generation

1. Select your AppID and choose **Configure**.



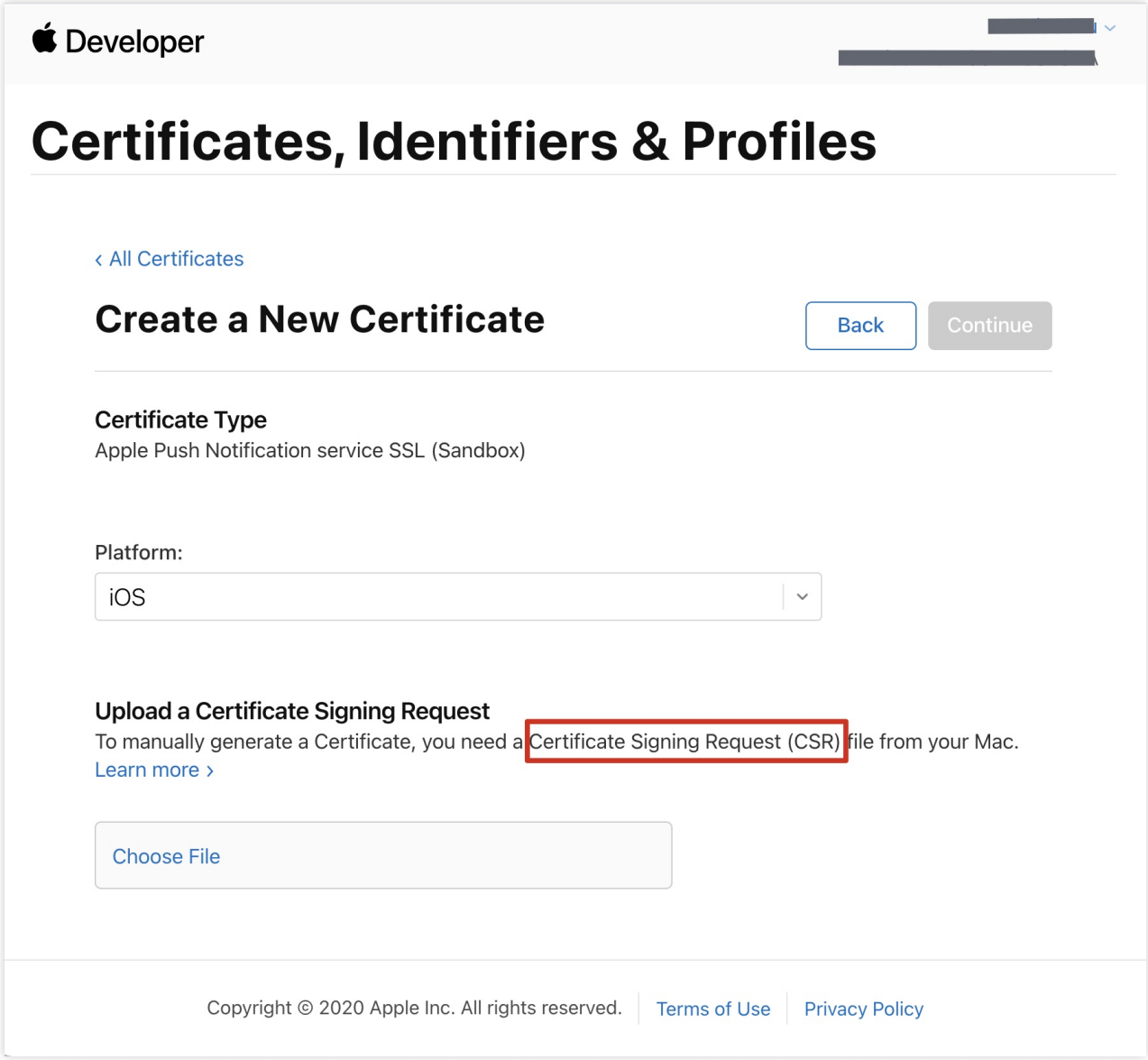
2. You can see in the **Apple Push Notification Service SSL Certificates** window, there are two `SSL Certificates` , one for the development environment (Development) and the other for the production environment (Production) remote push certificates, as shown below:



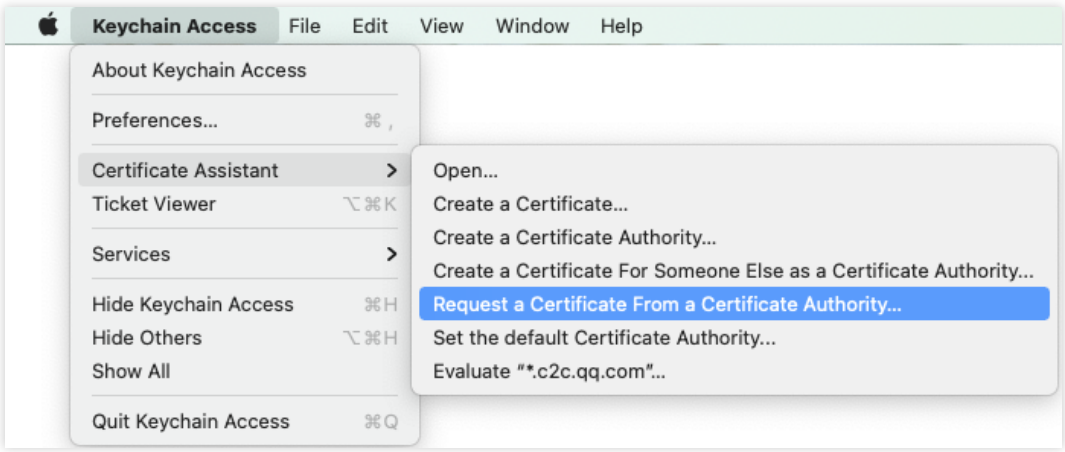
3.

We

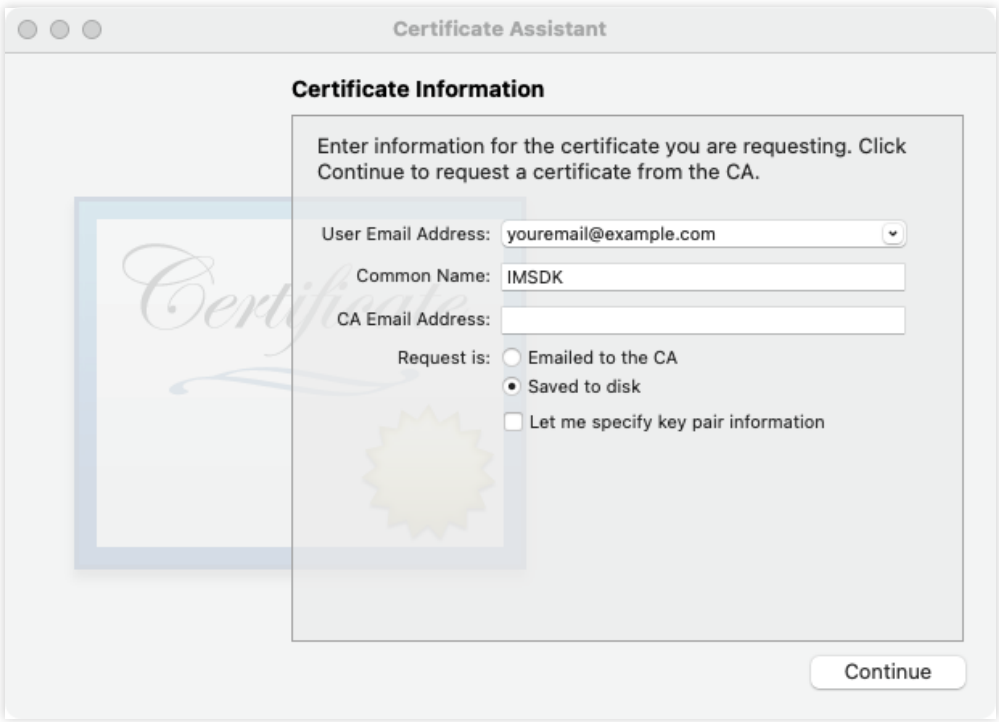
first select the development (Development) environment's **Create Certificate**, and the system will prompt you that a Certificate Signing Request (CSR) is required.



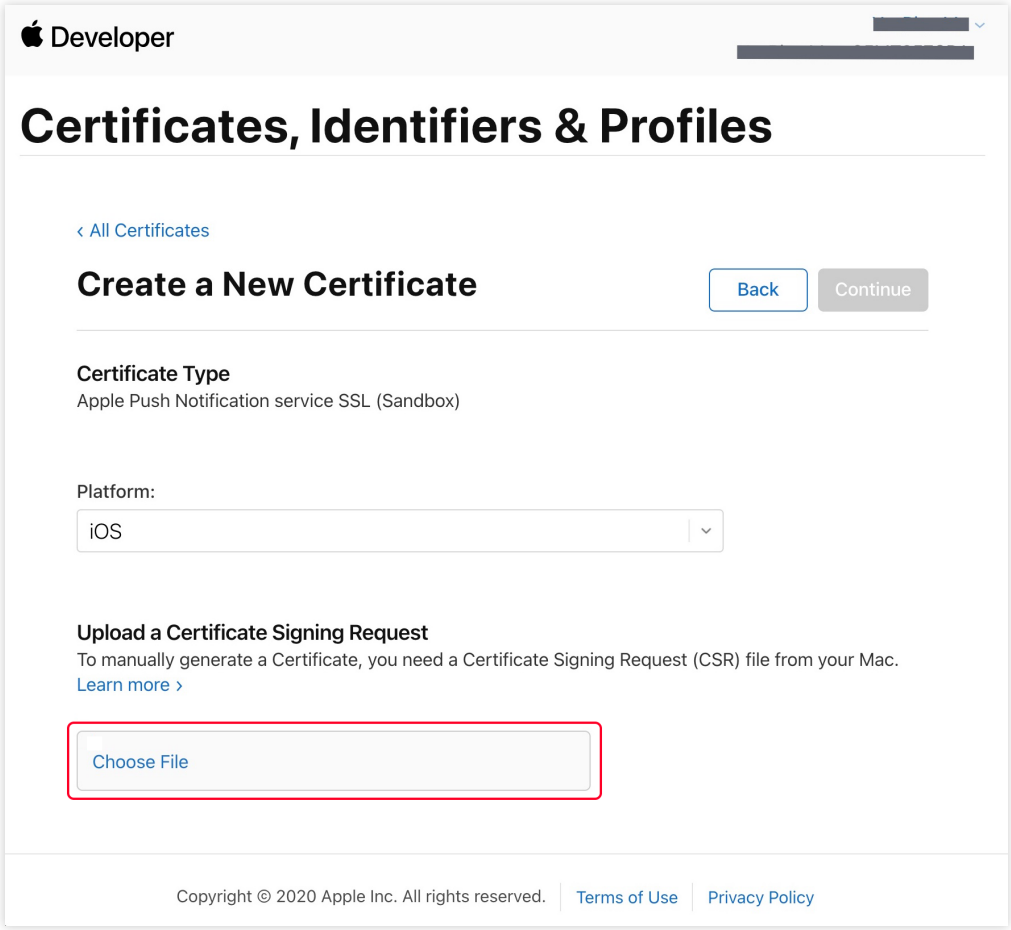
4. Open **Keychain Access tool (Keychain Access)** on Mac, select **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority** (`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`).



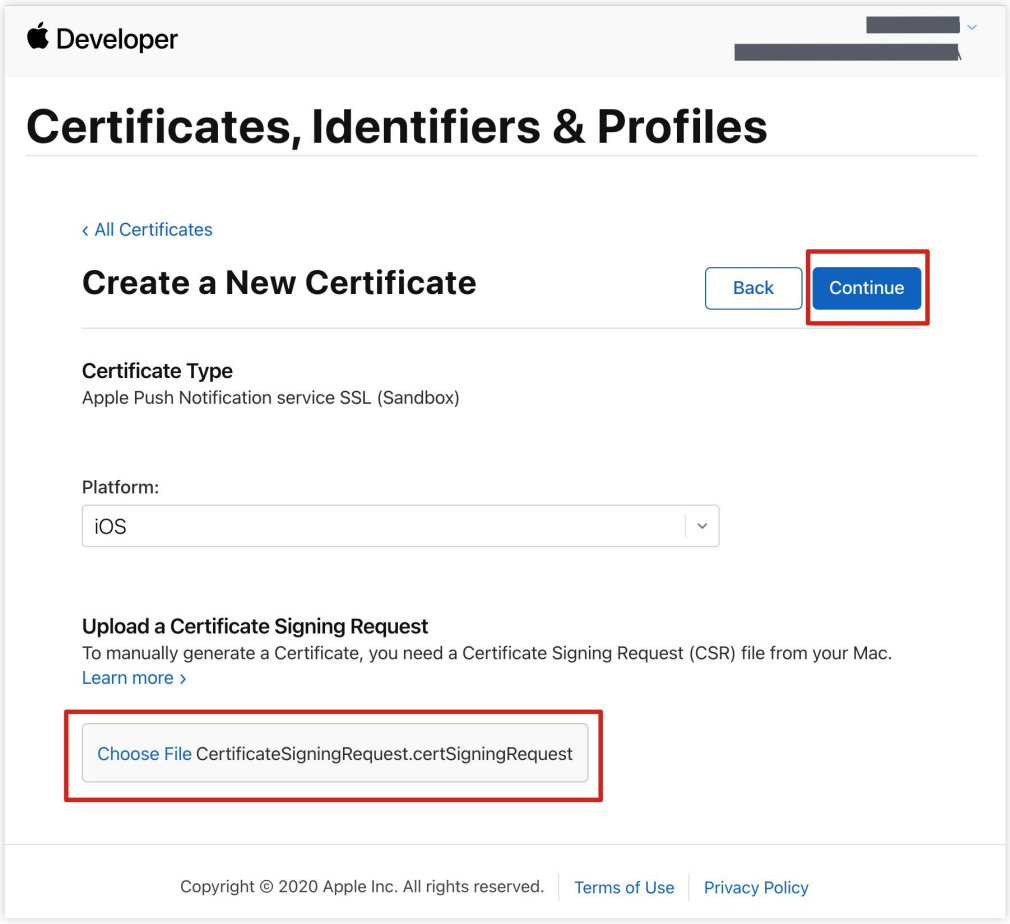
5. Enter your email address (Your email), Common Name (Your name or company name), choose **Save to disk**, click **continue**, and the system will generate a `*.certSigningRequest` file.



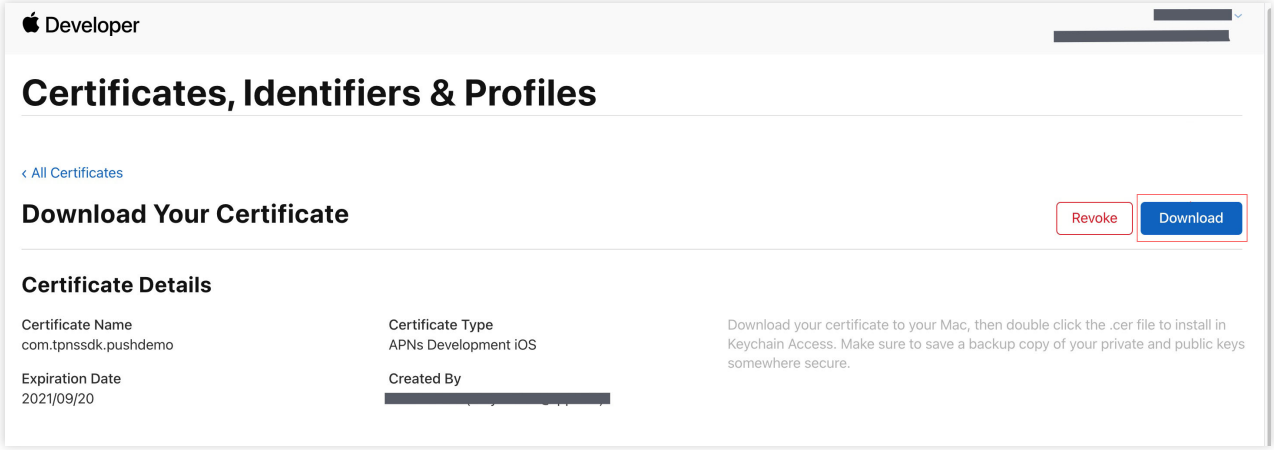
6. Return to the Apple Developer Website page mentioned in Step 3 above, click **Choose File** to upload the generated `*.certSigningRequest` file.



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



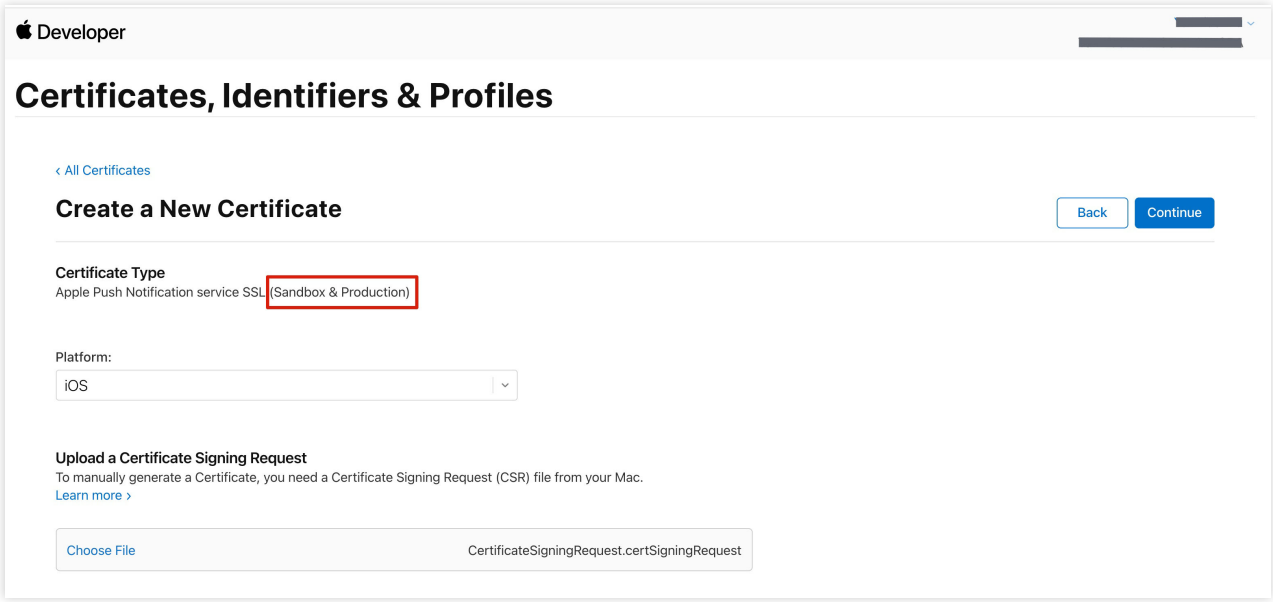
8. Click **Download** to download the `Development SSL Certificate` for your development environment to your local system.

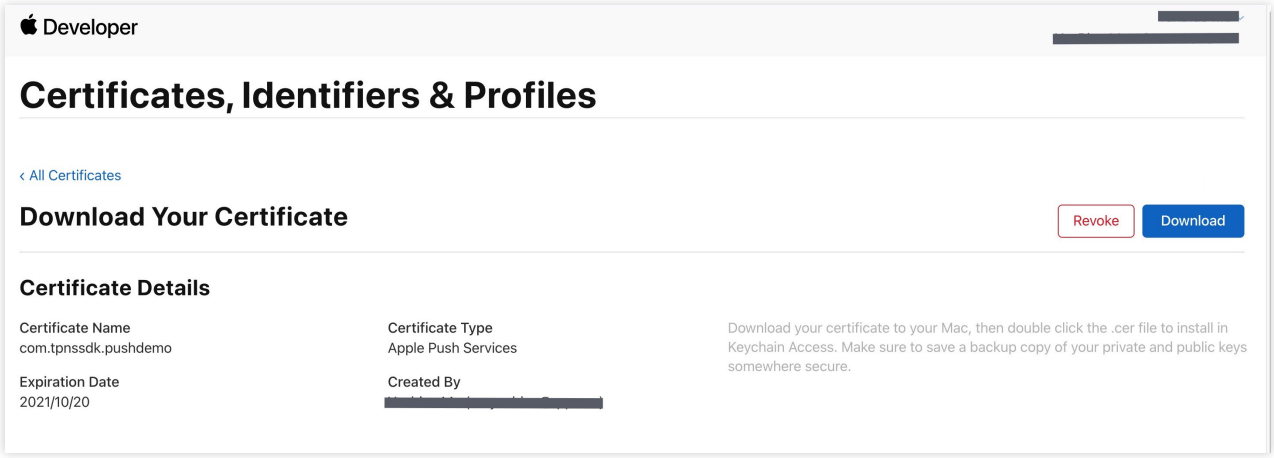


9. Follow the steps 1-8 again to download the `Production SSL Certificate` for your production environment to your local system.

Note:

The certificate for the production environment is actually a merged Sandbox and Production certificate, which can be used for both development and production environments.





10. Double-click the downloaded `SSL Certificate` for both the development and production environments, and the system will import it into your keychain.

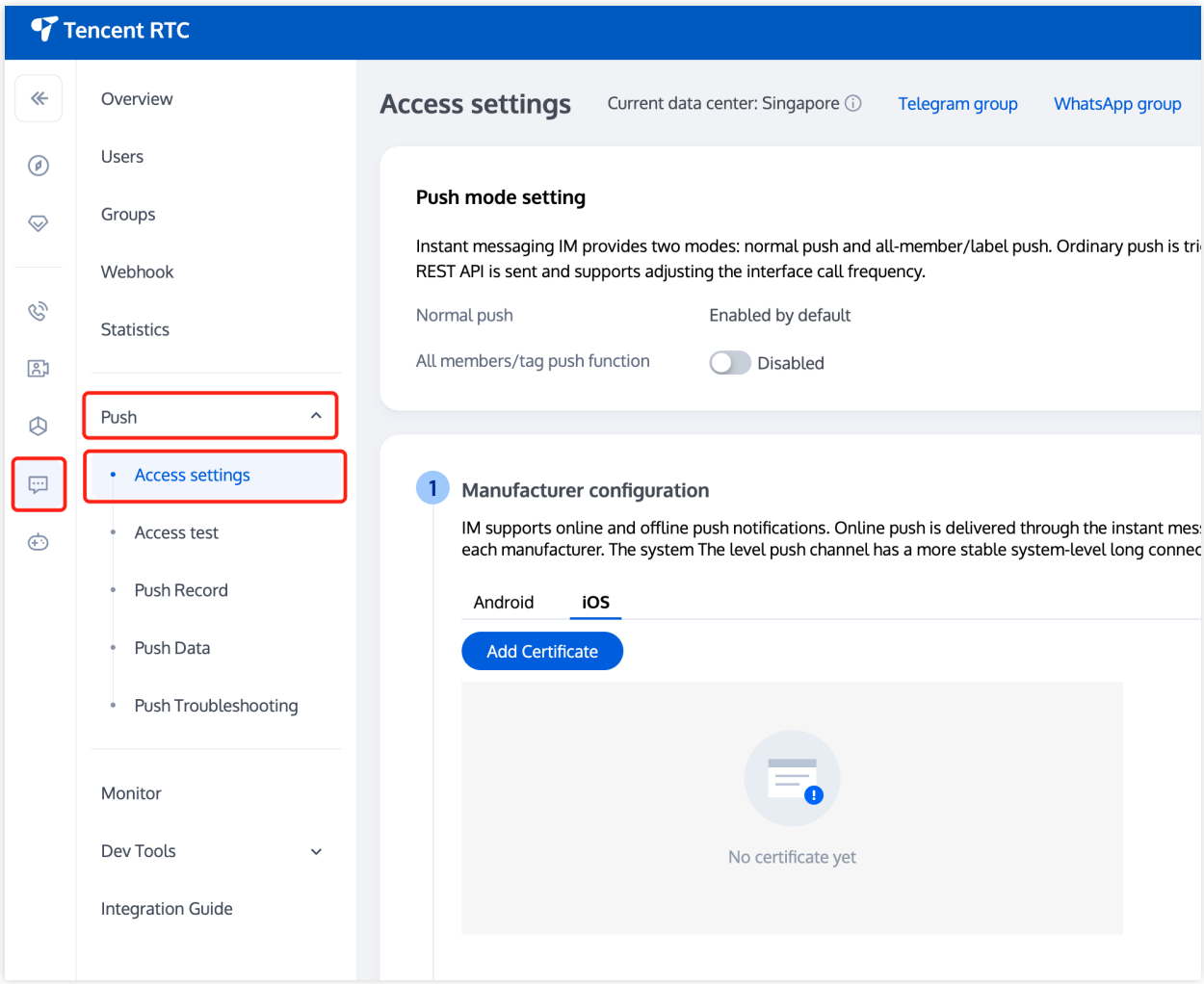
11. Open the Keychain application, navigate to **log in to My Certificates**, and right-click to export the newly created Development (`Apple Development IOS Push Service`) and Production (`Apple Push Services`) environment's `P12` files.

Note:

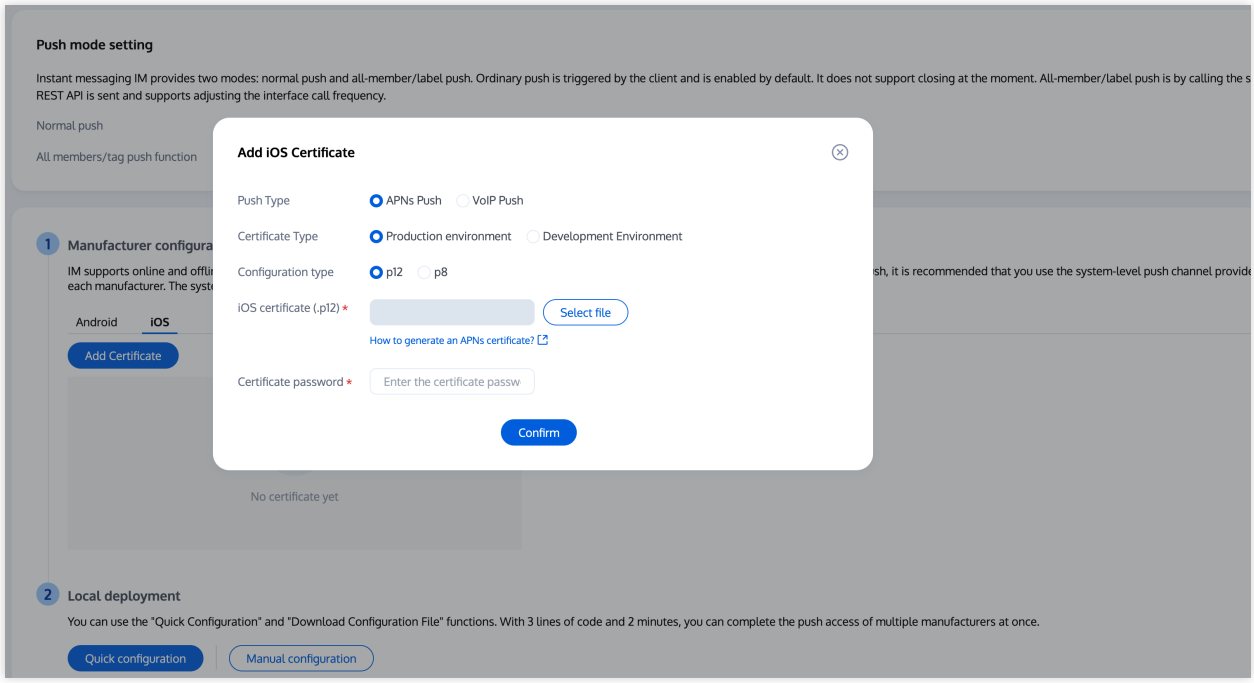
When saving the `P12` file, be sure to set a password.

Step 2: Upload the certificate to the console

1. Log in to the Tencent RTC Console.
2. Click on the Target Application Card, select the **Chat** Tag on the left, click on **Push**, then click on **Access settings**.



3. Click on **iOS Native Offline Push Settings** on the right side to **Add Certificate**.
4. Select Certificate Type, upload the iOS Certificate (p12), set the Certificate Password, and click on **Confirm**.



Note：

We recommend naming the uploaded certificate in English (special characters such as brackets are not allowed).

You need to set a password for the uploaded certificate. Without a password, push notifications cannot be received.

For an app published on App Store, the environment of the certificate must be the production environment. Otherwise, push notifications cannot be received.

The uploaded .p12 certificate must be your own authentic and valid certificate.

Operation step

Step 1. Register your app with vendor push platforms

Offline push requires registering your own app with each vendor's push platform to obtain parameters like AppID and AppKey, to implement the offline push feature.

Step 2: TRTC Console Configuration

log in to Tencent Cloud Tencent RTC Console, go to **Chat > Push > Access settings** feature bar to add push certificates from various vendors, and configure the AppId, AppKey, AppSecret and other parameters obtained in Step 1 to the added push certificates.

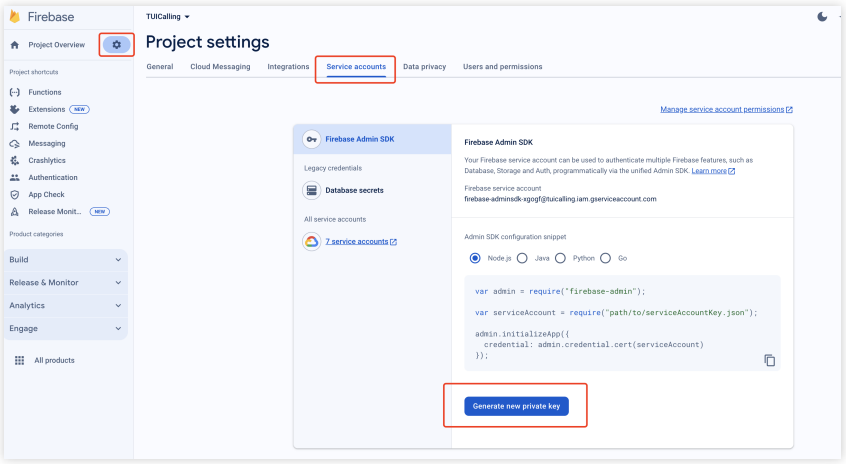
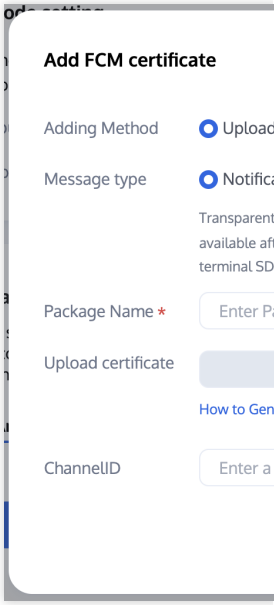
Note:

Regarding the **Click for Subsequent Actions** option:

if you wish to use this plugin's capability to redirect clicks, please keep the default values unchanged, which typically means `Opening a specified page within the application` with default settings.

If you also wish to use the Report Statistics feature, please keep this option at its default value,

Google FCM

Vendor Push Platform	Configuring in the IM console
	

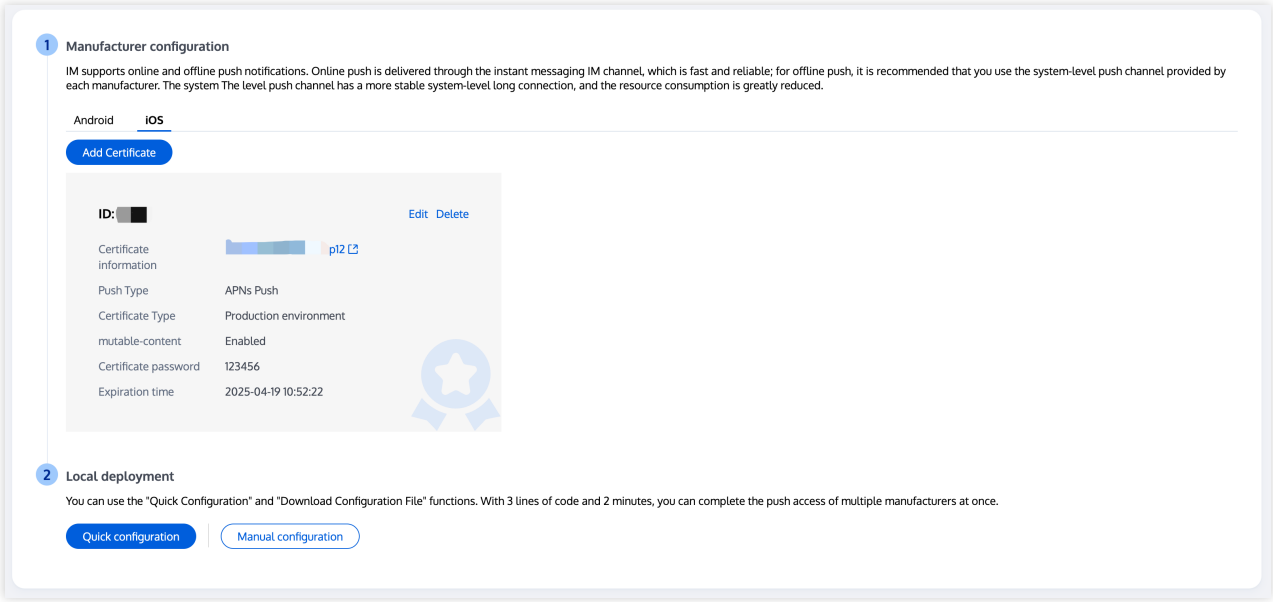
Step 3: Push Parameter Configuration

iOS

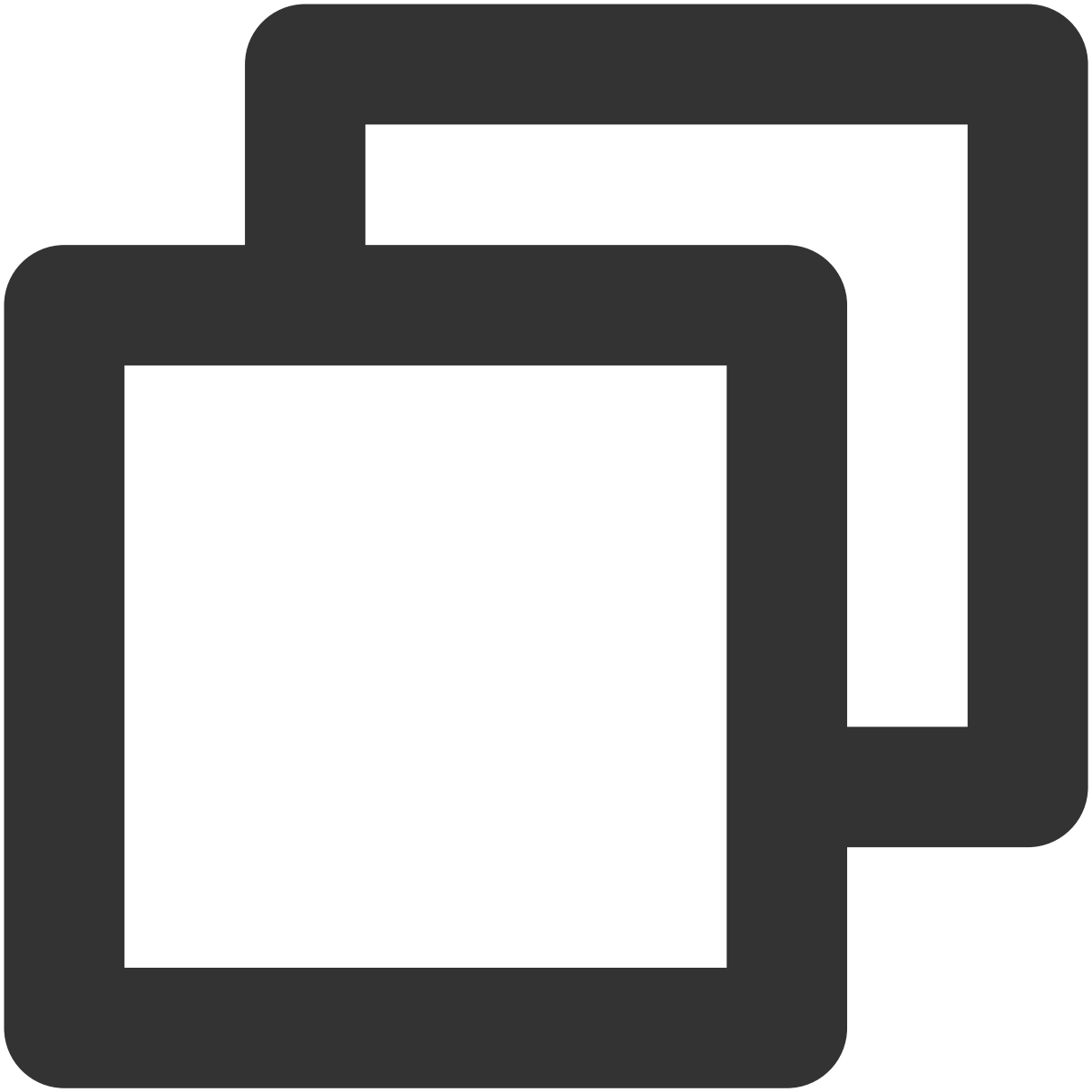
Android

Please upload the iOS APNs push certificate you accessed in the Manufacturer Configuration step to the Chat Console.

The Chat console will allocate a Certificate ID for you, as shown in the figure below:



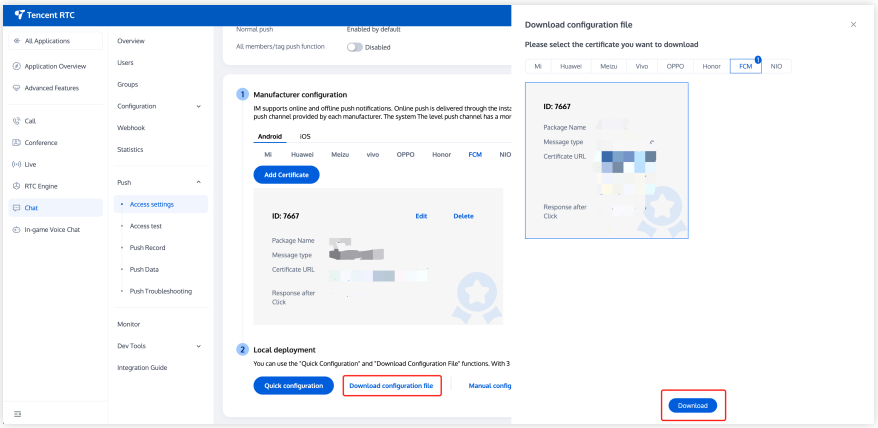
As soon as possible after your application starts, call the `TencentCloudChatPush().setApnsCertificateID` method to pass in this Certificate ID.

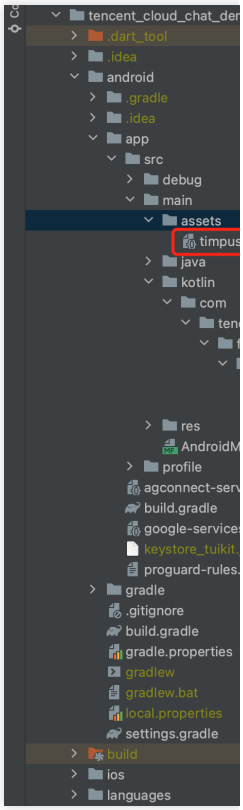


```
TencentCloudChatPush().setApnsCertificateID(apnsCertificateID: Certificate ID);
```

After completing the push info fill-in via console, download and add the configuration file to the project. Place the downloaded timpush-configs.json file under the `android/app/src/main/assets` directory. If the directory is non-existent, manually create it.

1.Select to download the configuration file timpush-configs.json	2.Add it to your project





Step 4: Client code configuration

In this step, several native codes are to be written, such as: Swift, Java, XML, etc.

Don't fret, simply follow the instructions and copy the code we provide into the specified files.

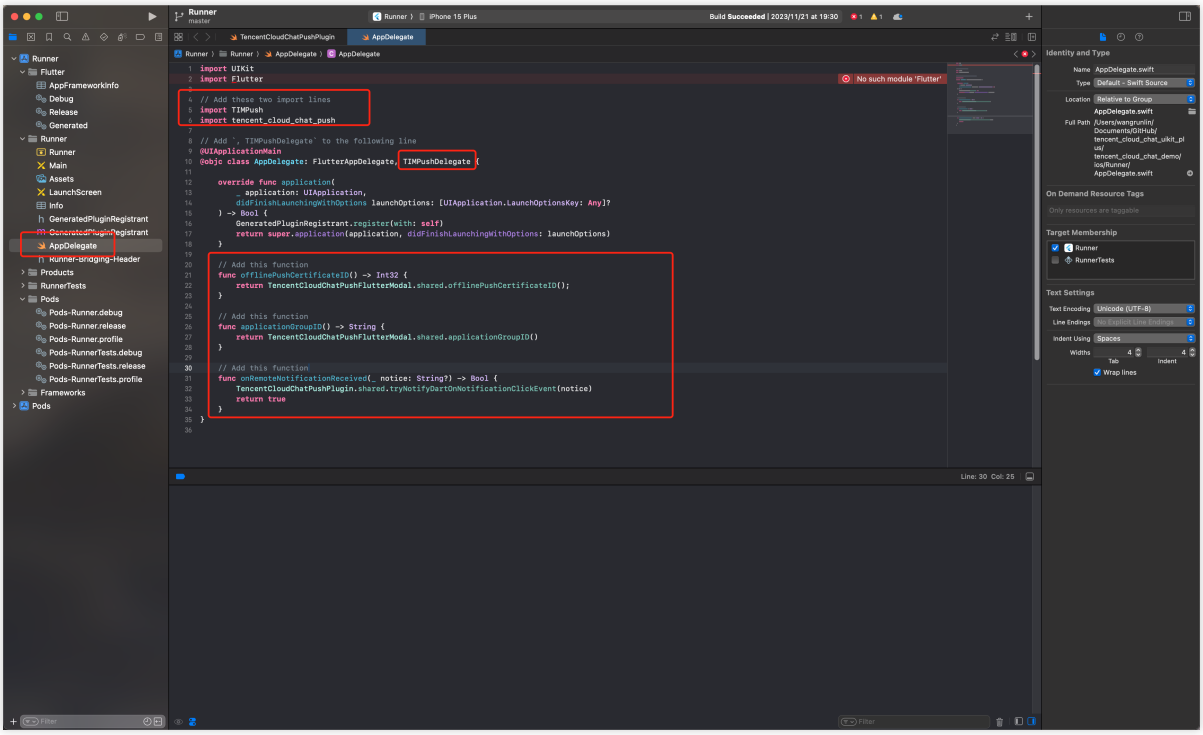
iOS

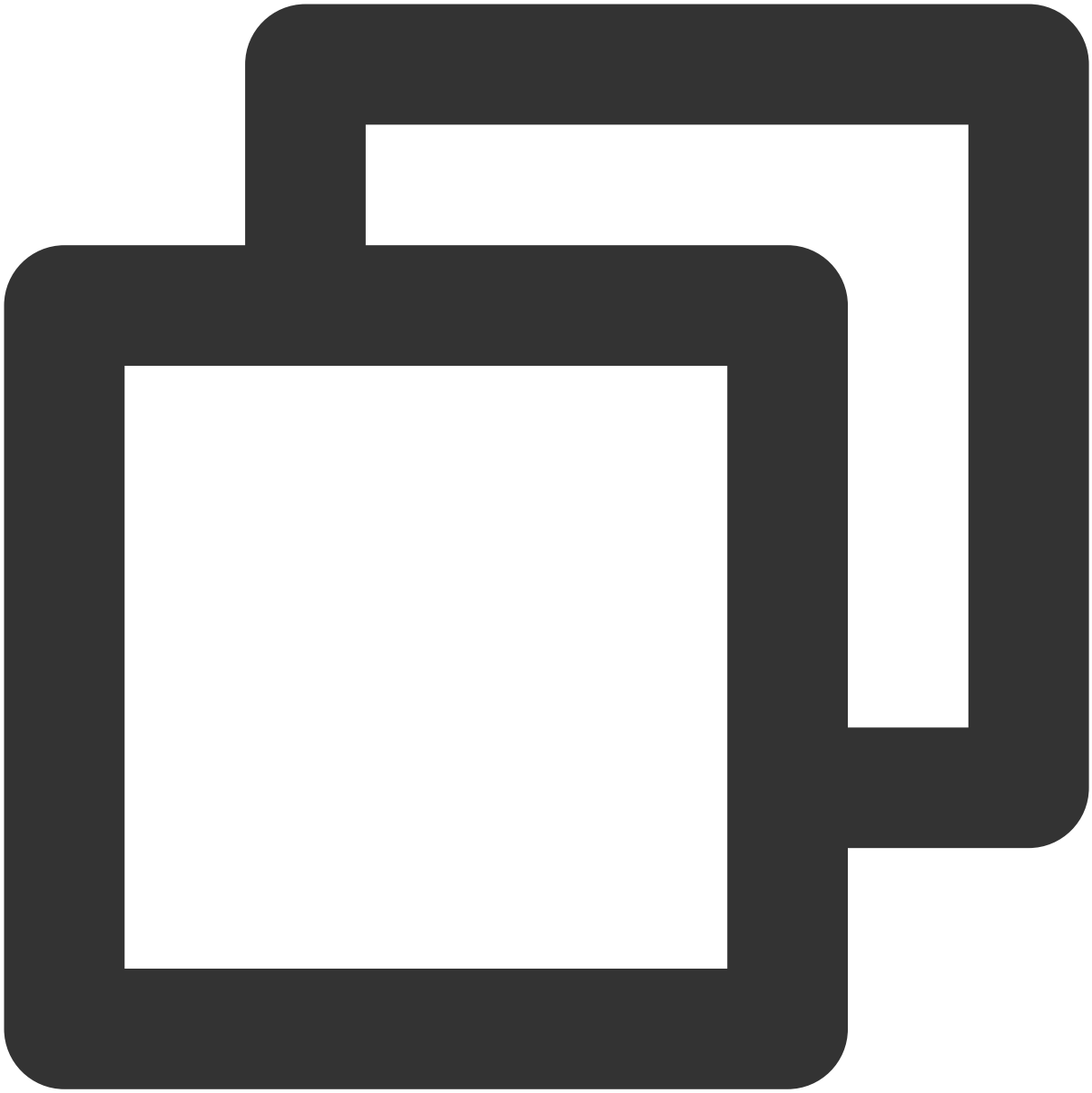
Android

You may use Xcode for editing, or directly edit in Visual Studio Code or Android Studio.

Open the `ios/Runner/AppDelegate.swift` file, paste the circled code below into it, as shown in the figure.

The code is attached after the image.





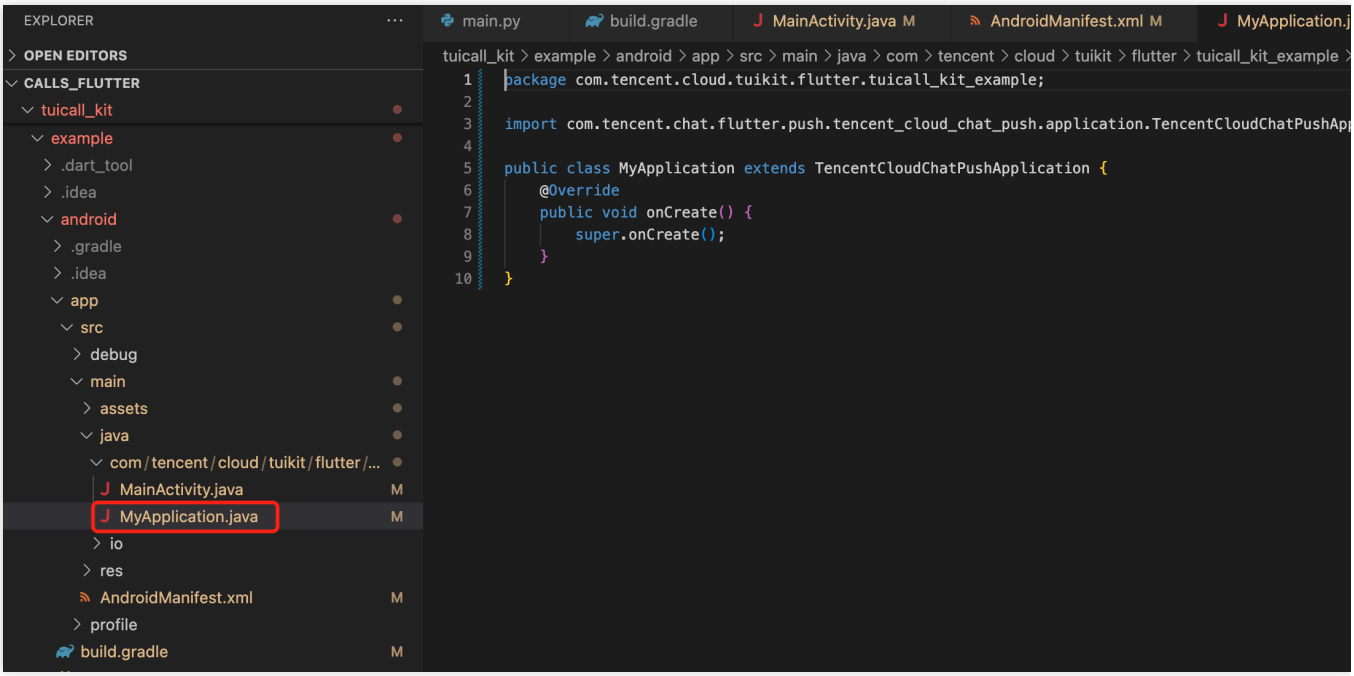
```
import UIKit
import Flutter

// Add these two import lines
import TIMPush
import tencent_cloud_chat_push

// Add `, TIMPushDelegate` to the following line
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate, TIMPushDelegate {
    override func application(
```

```
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKe
    ) -> Bool {
        GeneratedPluginRegistrant.register(with: self)
        return super.application(application, didFinishLaunchingWithOptions: launch
    }

    // Add this function
    func offlinePushCertificateID() -> Int32 {
        return TencentCloudChatPushFlutterModal.shared.offlinePushCertificateID();
    }

    // Add this function
    func applicationGroupID() -> String {
        return TencentCloudChatPushFlutterModal.shared.applicationGroupID()
    }

    // Add this function
    func onRemoteNotificationReceived(_ notice: String?) -> Bool {
        TencentCloudChatPushPlugin.shared.tryNotifyDartOnNotificationClickEvent(not
        return true
    }
}
```

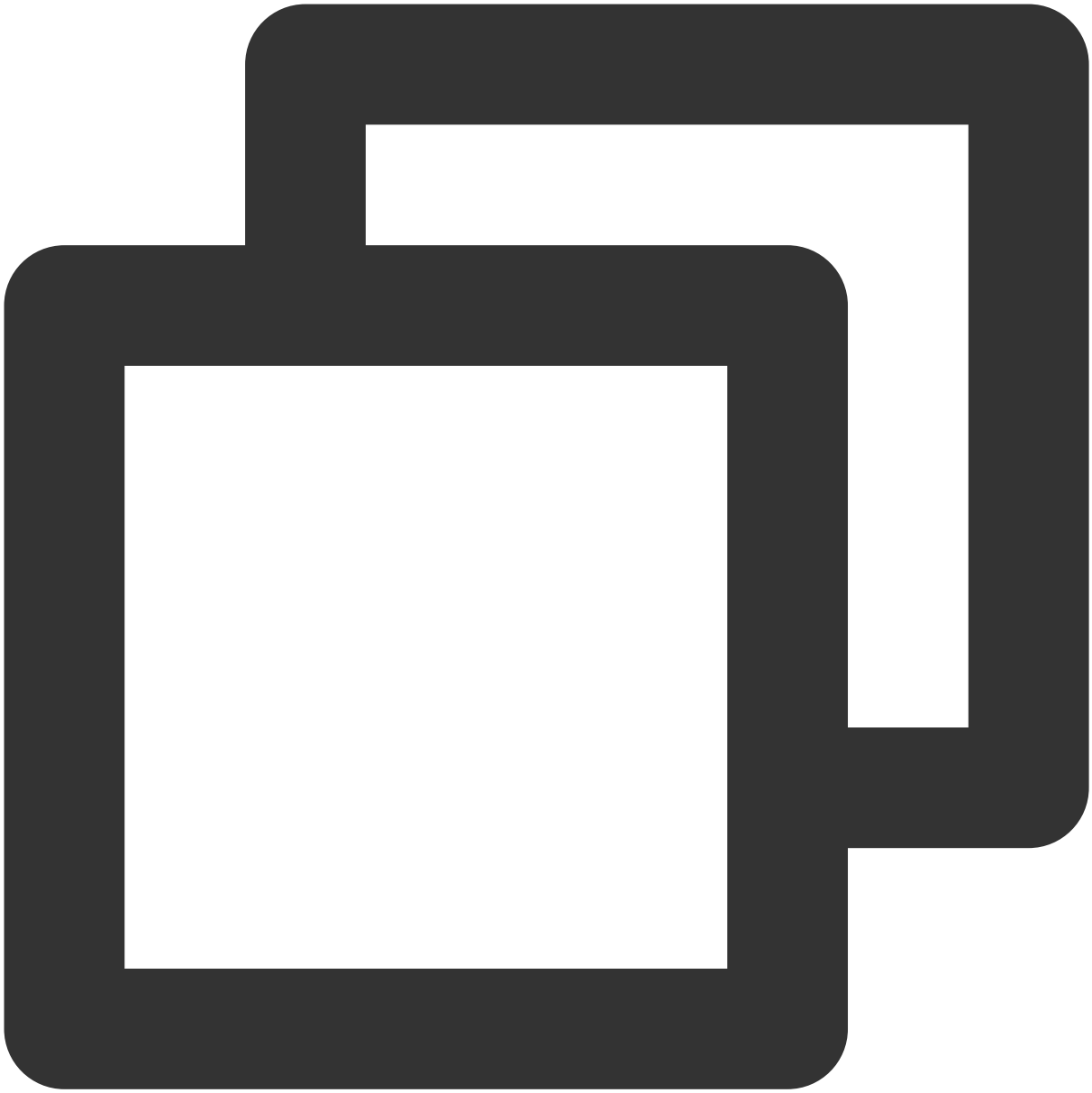
We advise utilizing Android Studio for completing this part of editing.

In the same directory as `MainActivity` under your project's android path, create a new Application file category, which could be named `MyApplication` .

If you have already defined your own Application class, you can directly reuse it, without the need for recreating.



Embed the following code into the file, as demonstrated above:

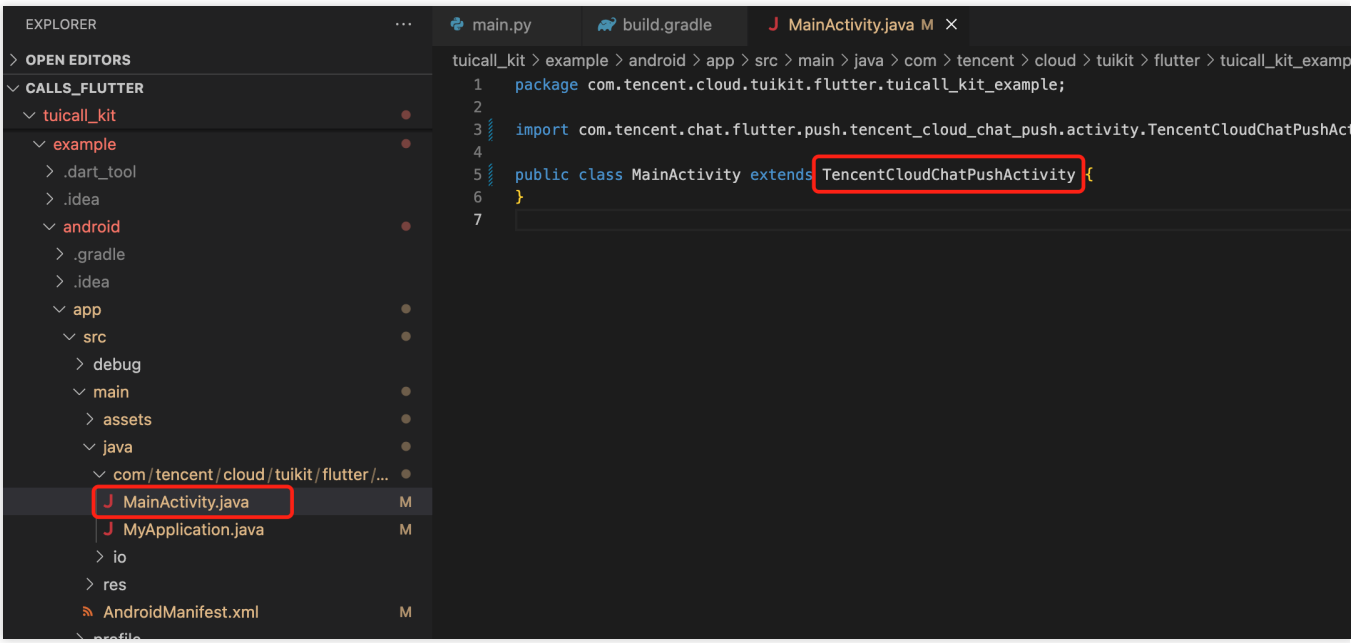


```
package xxxx.xxxx.xx

import com.tencent.chat.flutter.push.tencent_cloud_chat_push.application.TencentClo

public class MyApplication extends TencentCloudChatPushApplication {
    @Override
    public void onCreate() {
        super.onCreate();
    }
}
```

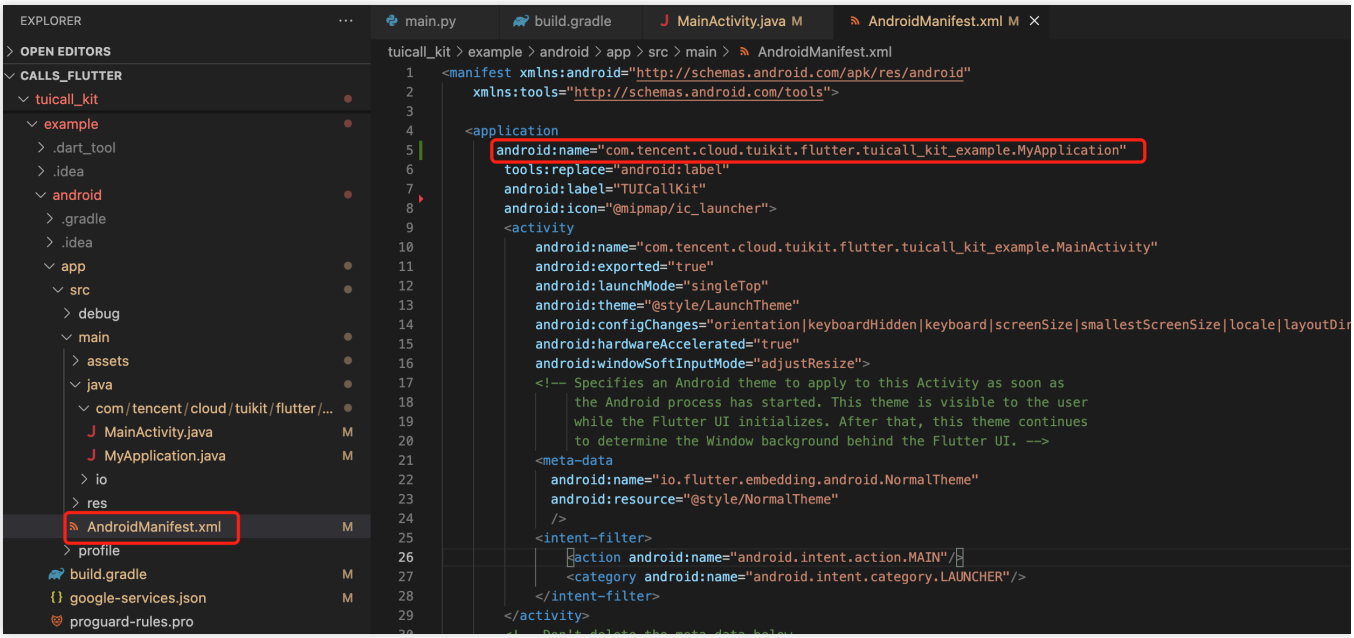
Note:

If you have already created your own Application for other purposes, simply `extend TencentCloudChatPushApplication` and ensure that `onCreate()` method is called in `super.onCreate();` .

Meanwhile, you also need to modify your MainActivity File:



Open the `android/app/src/main/AndroidManifest.xml` file, then add a specific `android:name` parameter to the `<application>` Tag, which is linked to your newly created Definition `Application` class as shown in the figure:



Step 5: Client Vendor Configuration

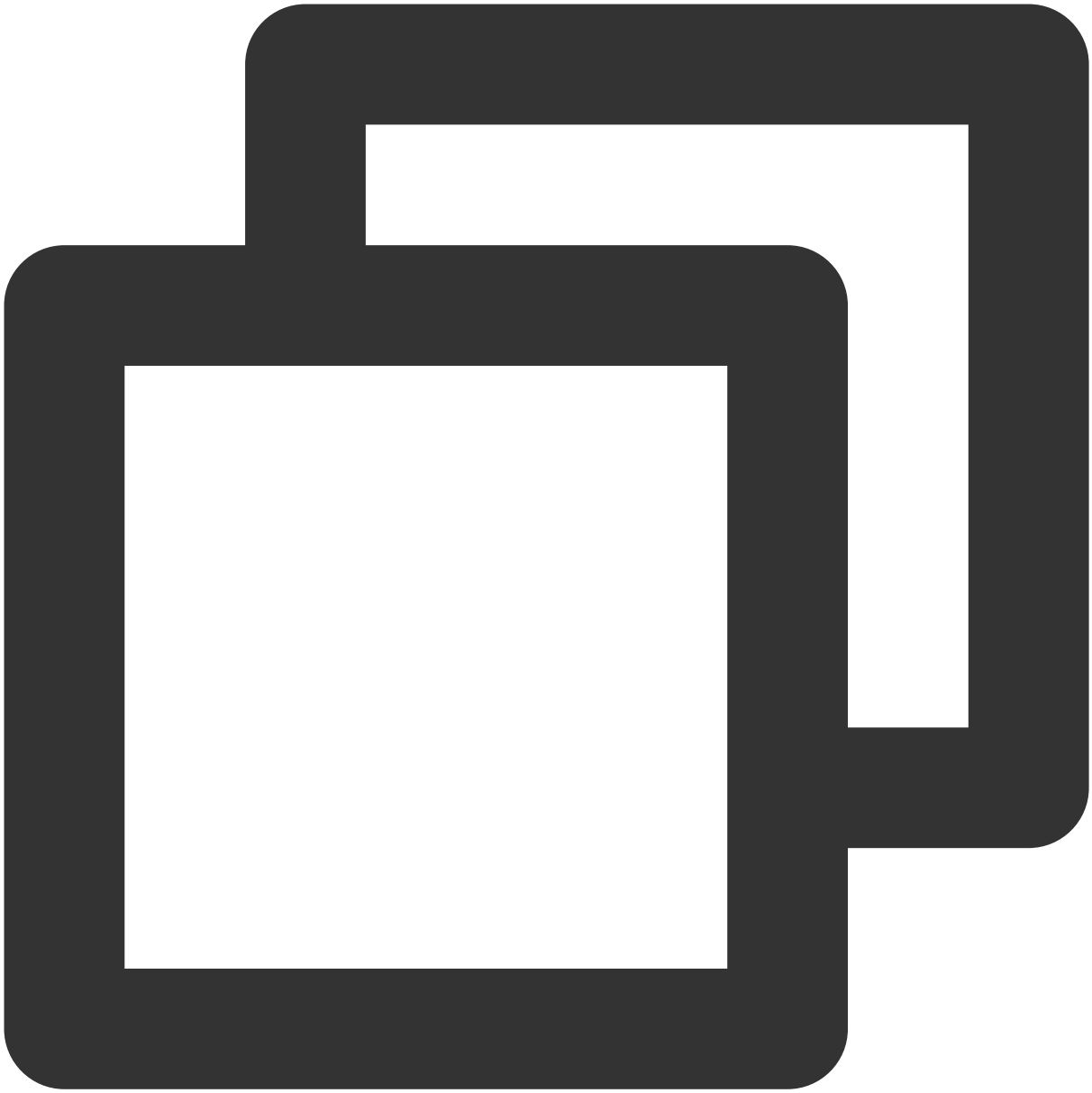
iOS

Android

No need to perform this step on the iOS side.

Open the android/app/build.gradle file and add a new dependencies configuration at the end. Based on your requirements, incorporate the push packages from any or all of the vendors listed below. Only by including the push package from the respective vendor can you enable the native push capability of that vendor.

The version number described below should be consistent with that of this Flutter push plugin (tencent_cloud_chat_push).



```
dependencies {
     // Google Firebase Cloud Messaging (Google FCM)
     implementation 'com.tencent.timpush:fcm:${version number of push plugin}'
}
```

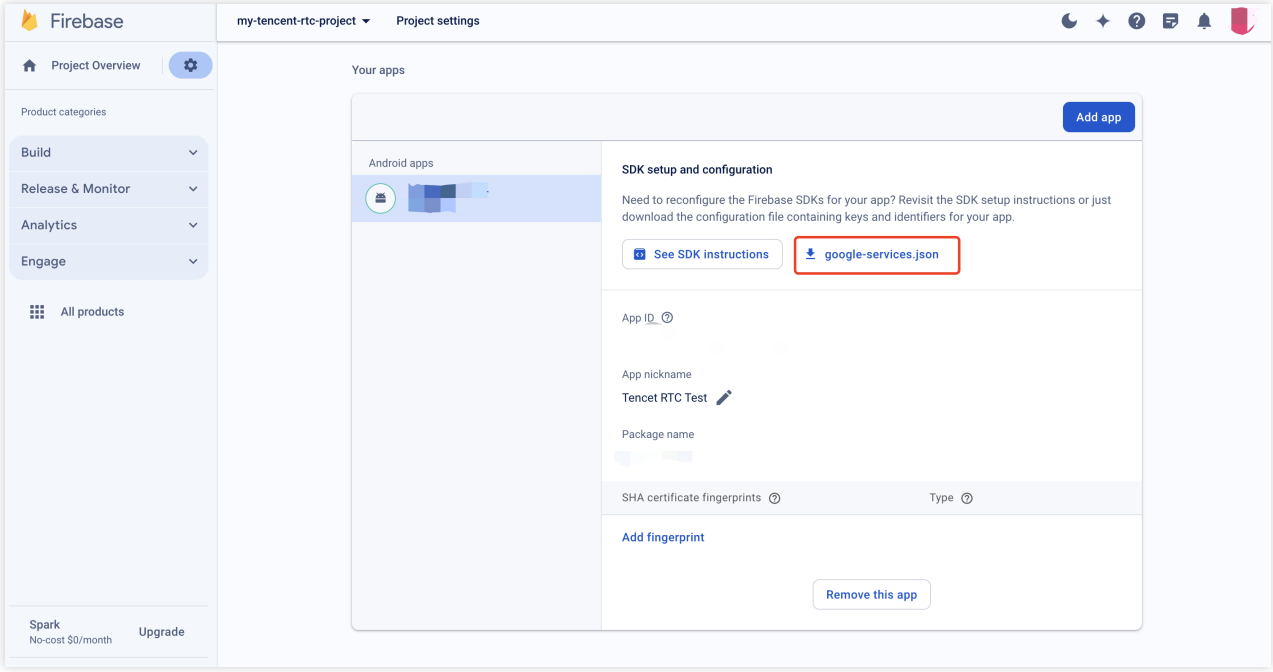
Google FCM Adaptation

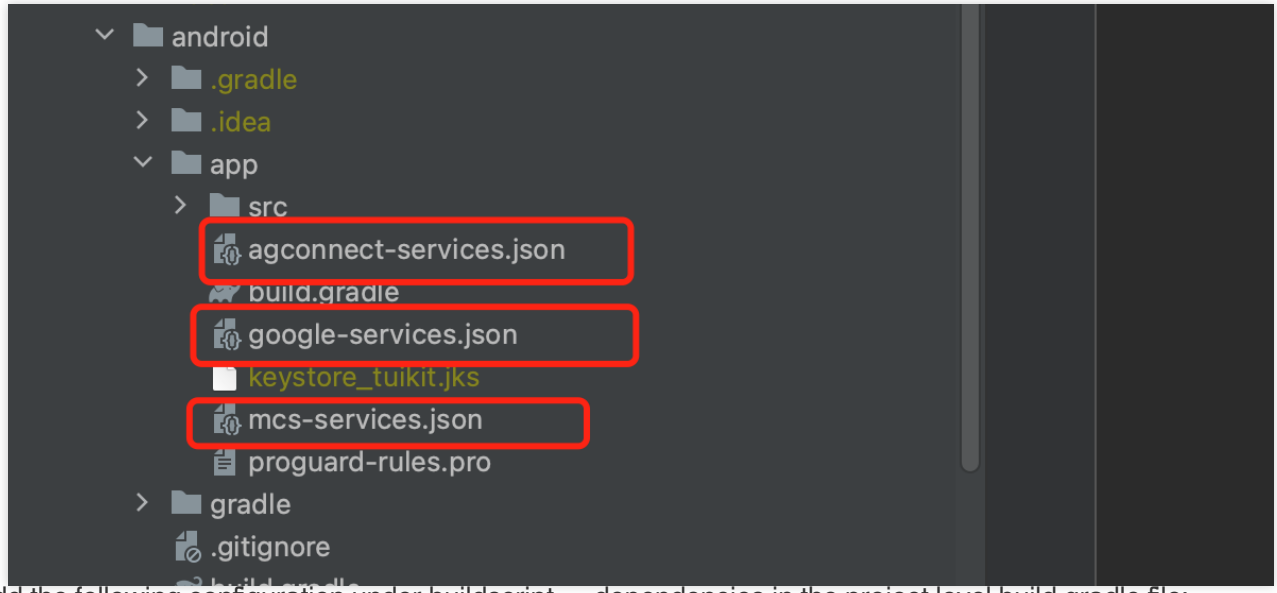
Integrate the corresponding plugins and json configuration files according to the vendor's methods.

1.1 Download the configuration file and insert it into the project root directory/Android/app.

Google FCM

Operation Path





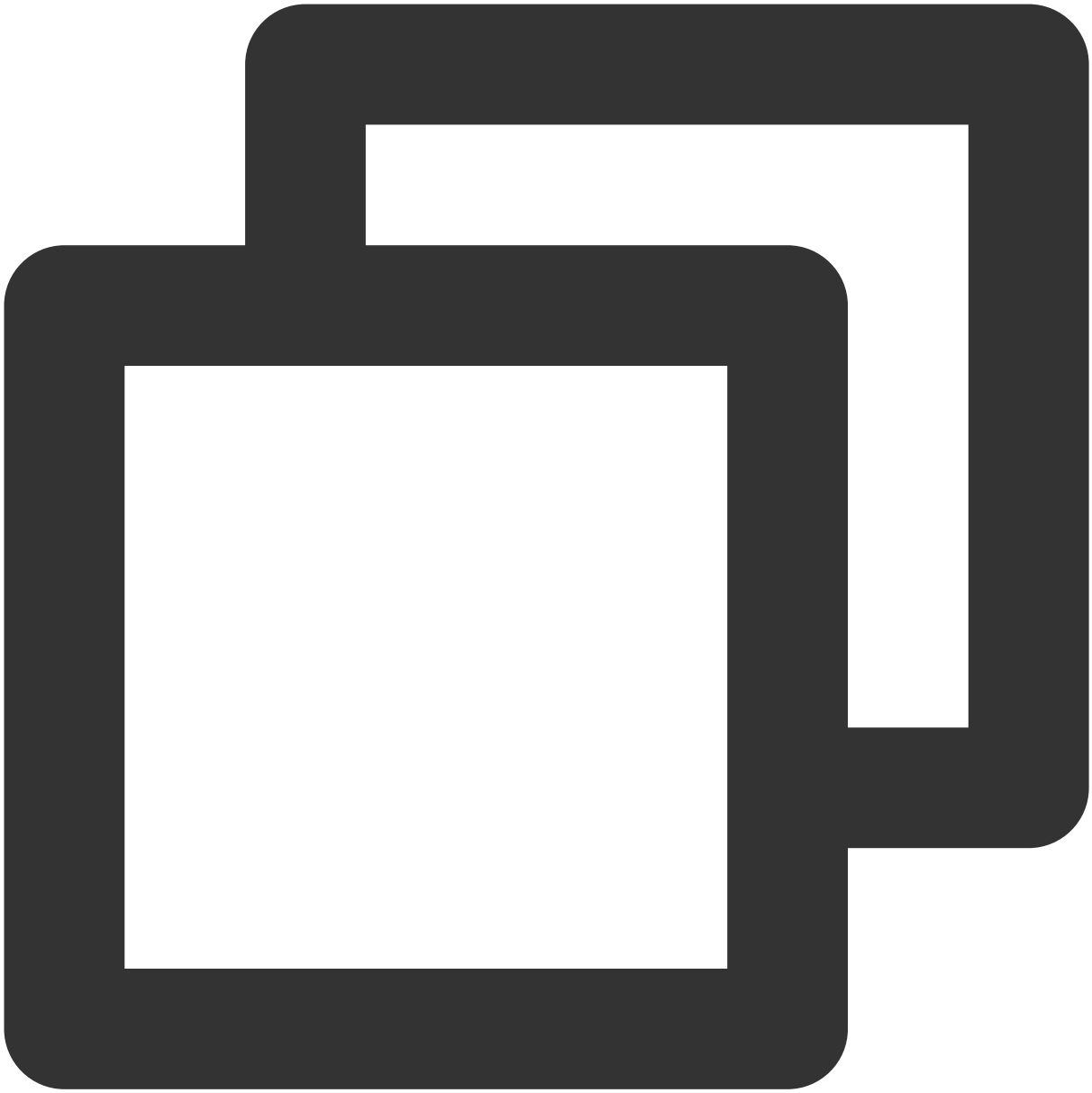
1.2 Add the following configuration under buildscript -> dependencies in the project level build.gradle file:

Gradle 7.1 and above versions

Version 7.0 of Gradle

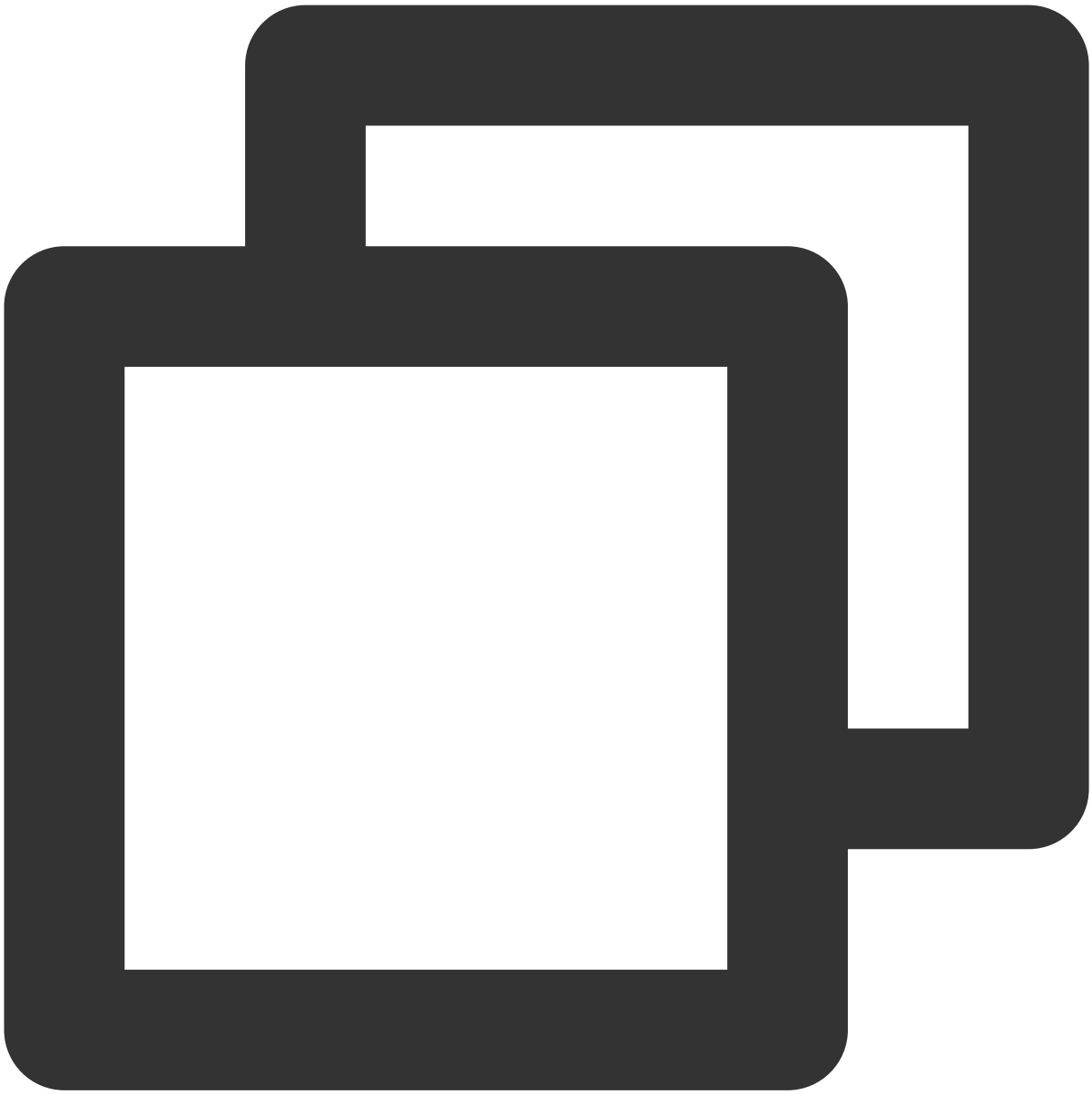
Versions below Gradle 7.0

Add the following configuration under buildscript -> dependencies in the project level build.gradle file:



```
buildscript {
    dependencies {
        ...
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
        classpath 'com.google.gms:google-services:4.4.0'
    }
}
```

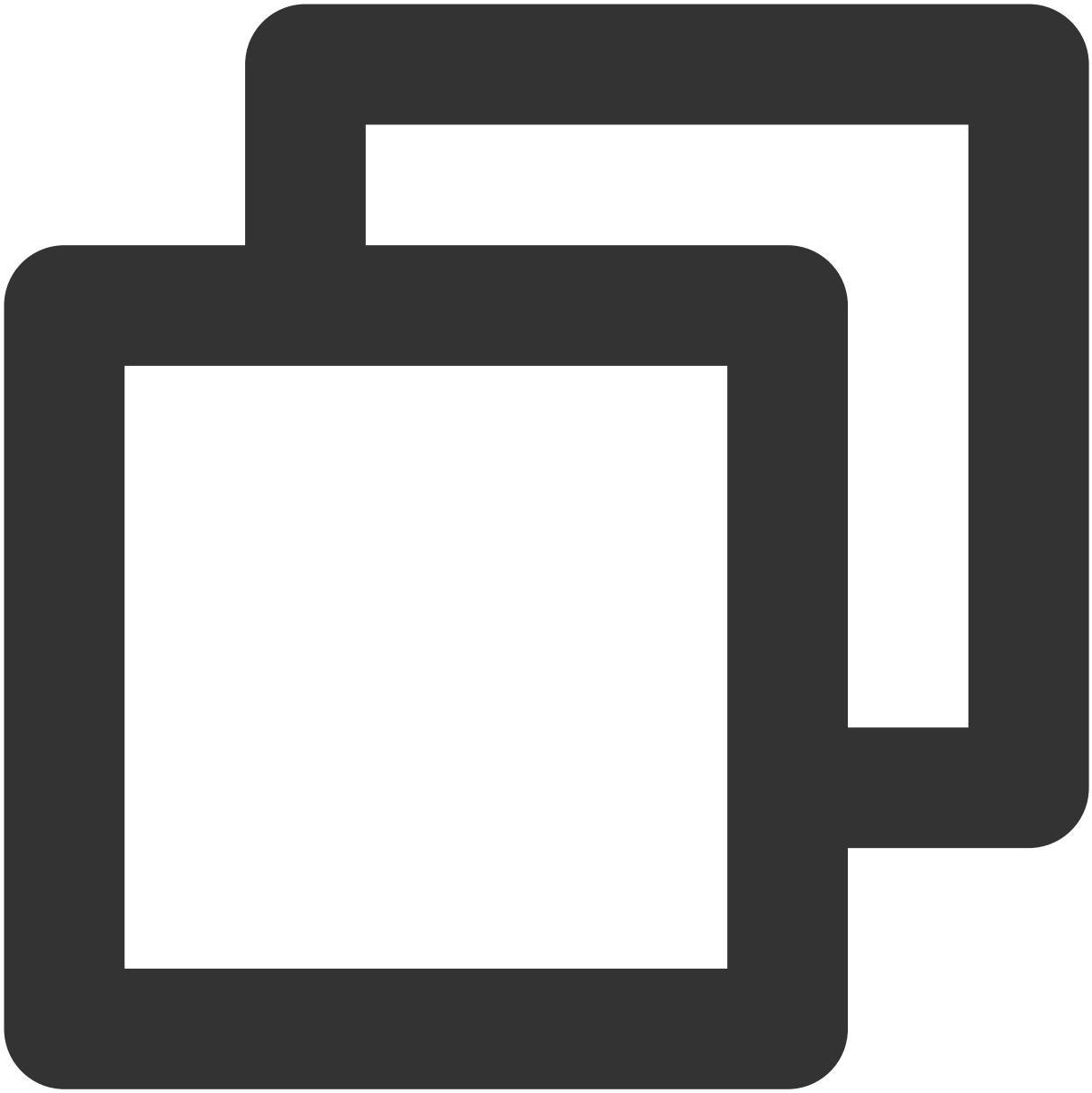
In your project-level settings.gradle file, add the following repositories under buildscript -> repositories and allprojects -> repositories configuration:



```
pluginManagementbuildscript {
    repositories {
        gradlePluginPortal()
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // The Maven repository address of HMS Core SDK.
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
```

```
    }
}
allprojects {
    ...
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // Configuration of the Maven repository address for HMS Core SDK.
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    }
}
```

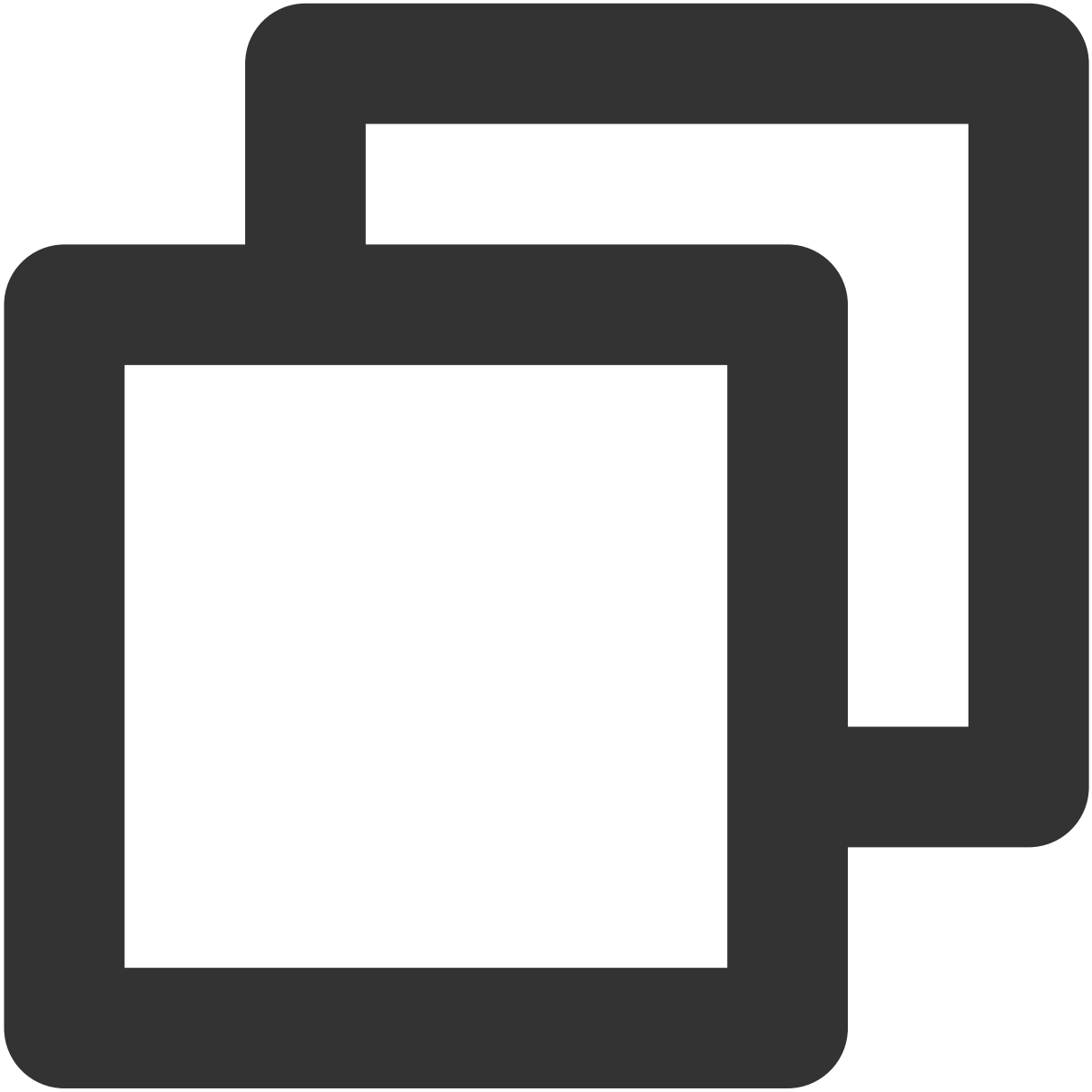
Add the following configuration under buildscript in the project-level build.gradle file:



```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // Configuration of the Maven repository address for HMS Core SDK.
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.2.0'
```

```
        classpath 'com.huawei.agconnect:agcp:1.4.1.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
```

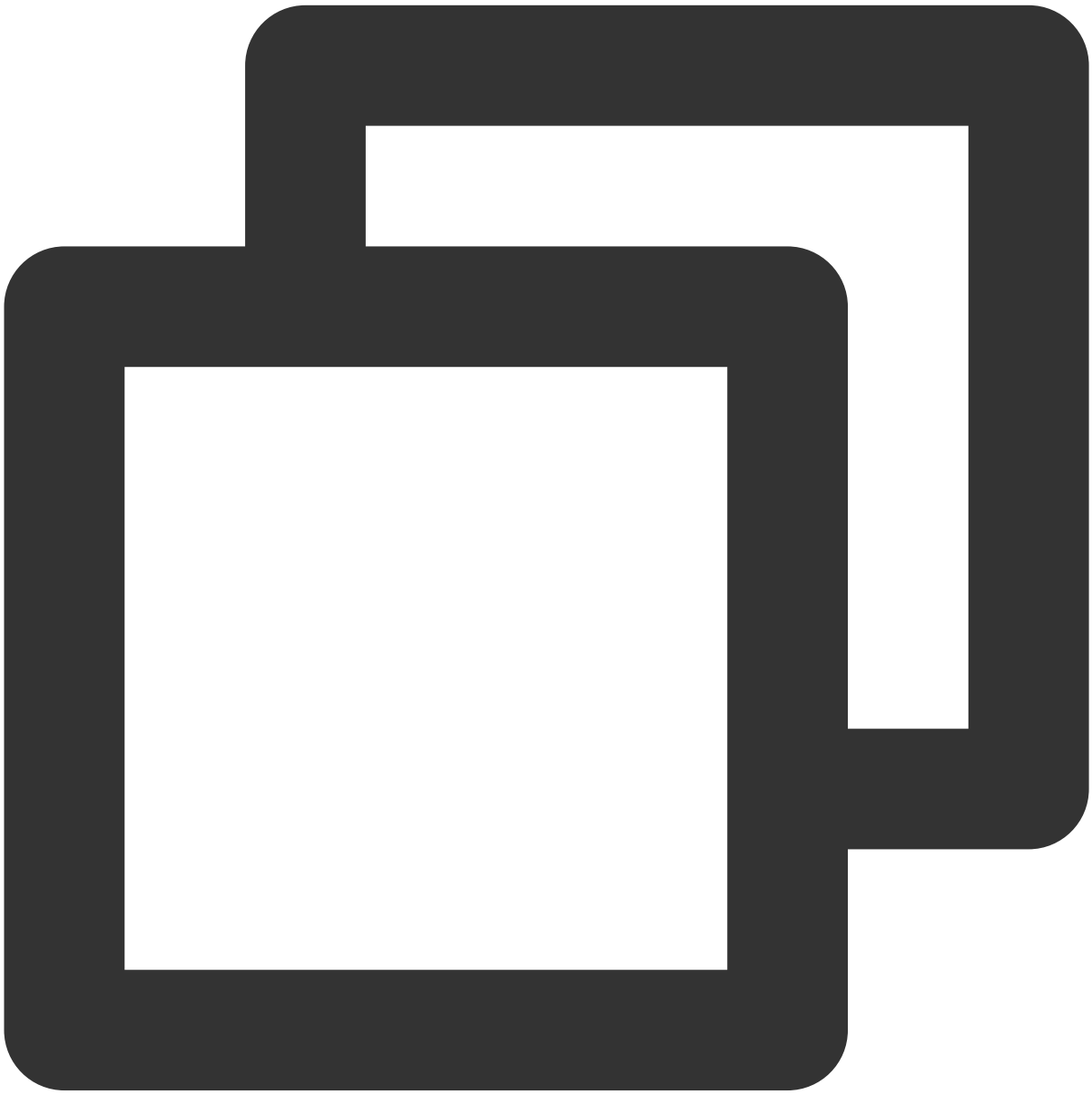
In the project-level settings.gradle file, add the following repository configuration under allprojects -> repositories:



```
allprojects {
    ...
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
```

```
        // Configuration of the Maven repository address for HMS Core SDK.
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
}
```

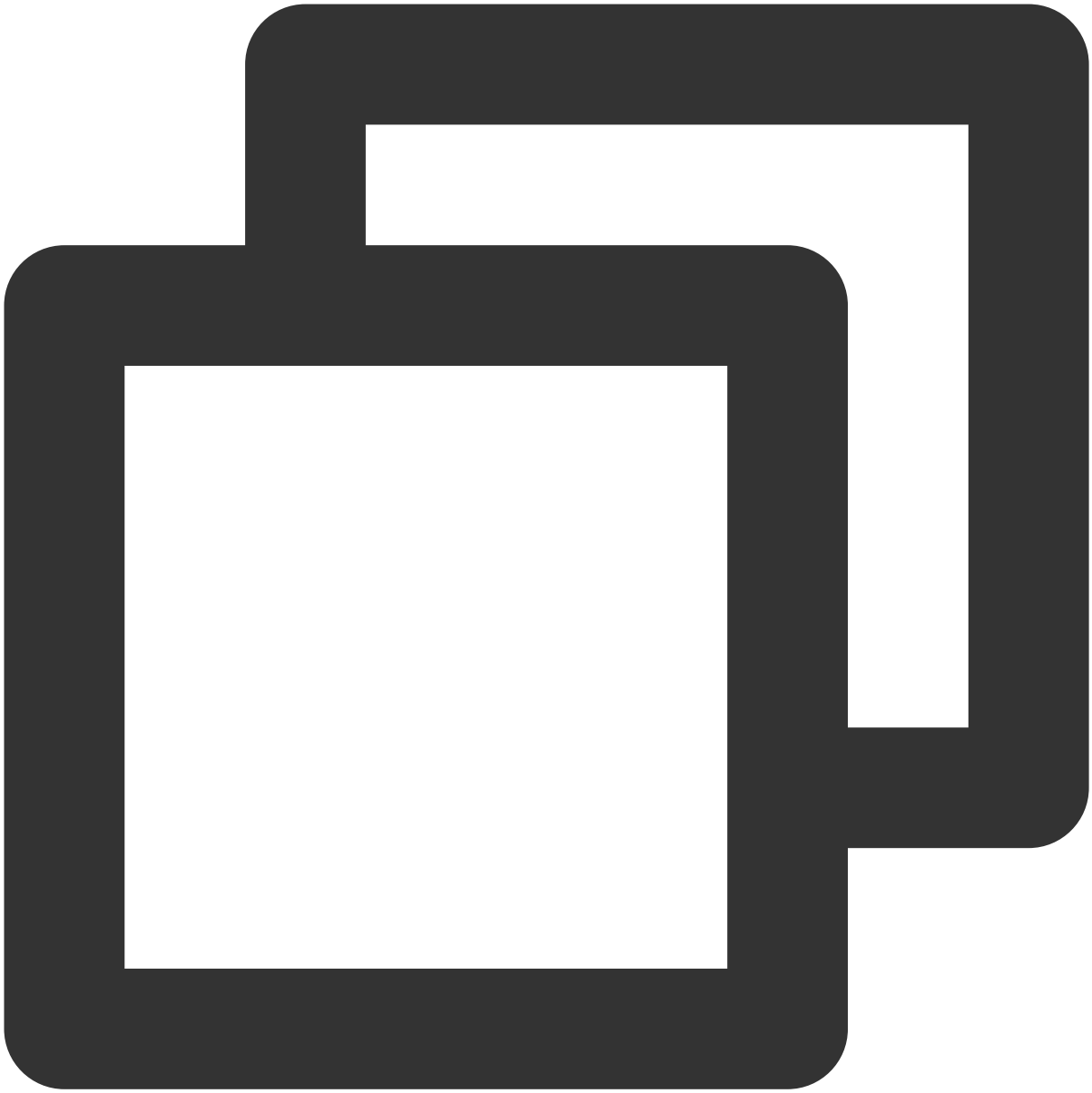
Incorporate the following configuration under buildscript and allprojects in the project level build.gradle file:



```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
```

```
        // Configuration of the Maven repository address for HMS Core SDK.
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.2.0'
        classpath 'com.huawei.agconnect:agcp:1.4.1.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}

allprojects {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // Configuration of the Maven repository address for HMS Core SDK.
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
}
```

1.3 In the application level build.gradle file, add the following configuration:



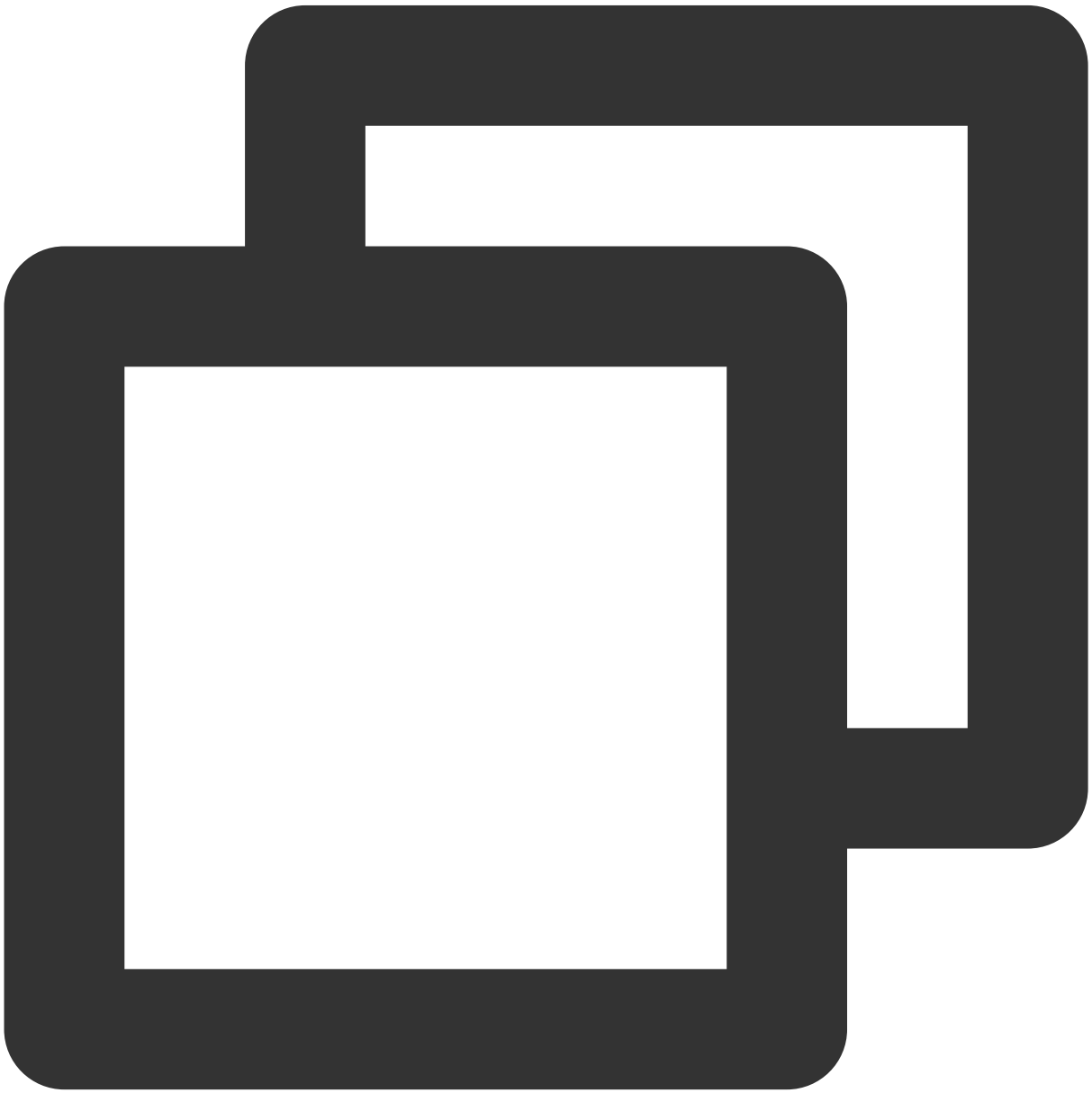
```
apply plugin: 'com.google.gms.google-services'
apply plugin: 'com.huawei.agconnect'
apply plugin: 'com.hihonor.mcs.asplugin'
```

Step 6: Process message click callbacks and parse parameters

Please define a function to accept the callback events of push message clicks.

Please define this function in the parameter format of `{required String ext, String? userID, String? groupID}` .

Among them, the ext field represents the complete ext information carried by this message, designated by the sender. If not specified, a default value exists. You can parse this field and navigate to the corresponding page.
The userID and groupID fields, in this plugin, automatically attempt to parse the ext Json String, carrying the userID of the individual chat counterpart and group chat groupID information in the access. If you have not self-defined the ext field, the ext field is default designated by the SDK or UIKit, then you can use the default parsing herein. If the parsing attempt fails, it becomes null.
You can define a function to receive this callback, and navigate to the corresponding session page or your business page based on it.
Following is an example:



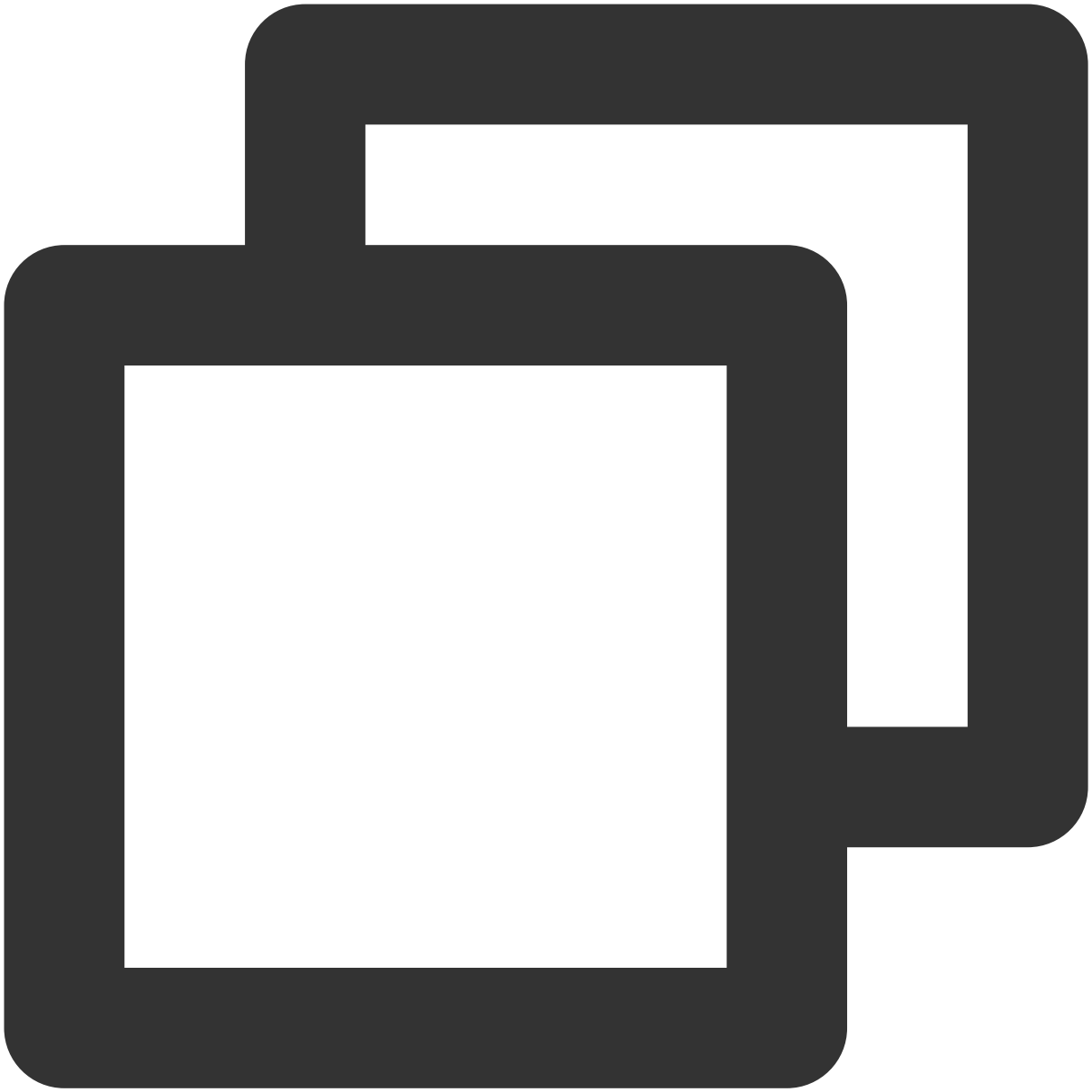
```
void _onNotificationClicked({required String ext, String? userID, String? groupID})
  print("_onNotificationClicked: $ext, userID: $userID, groupID: $groupID");
  if (userID != null || groupID != null) {
    // Navigate to the corresponding Message page based on `userID` or `groupID`.
  } else {
    // Based on the `ext` field, write your own parsing method to navigate to the c
  }
}
```

Step 7: Register the push plugin

Please register the push plugin immediately after logging in.

Invoke `TencentCloudChatPush().registerPush` method, it necessitates the transmission of BackDefinition's click callback function.

Moreover, you may also choose to input `apnsCertificateID` , the iOS push certificate ID, and `androidPushOEMConfig` , the Android push vendor configuration. These two configurations were previously specified in the initial steps; should there be no need for amendment, there is no requirement to input them again.

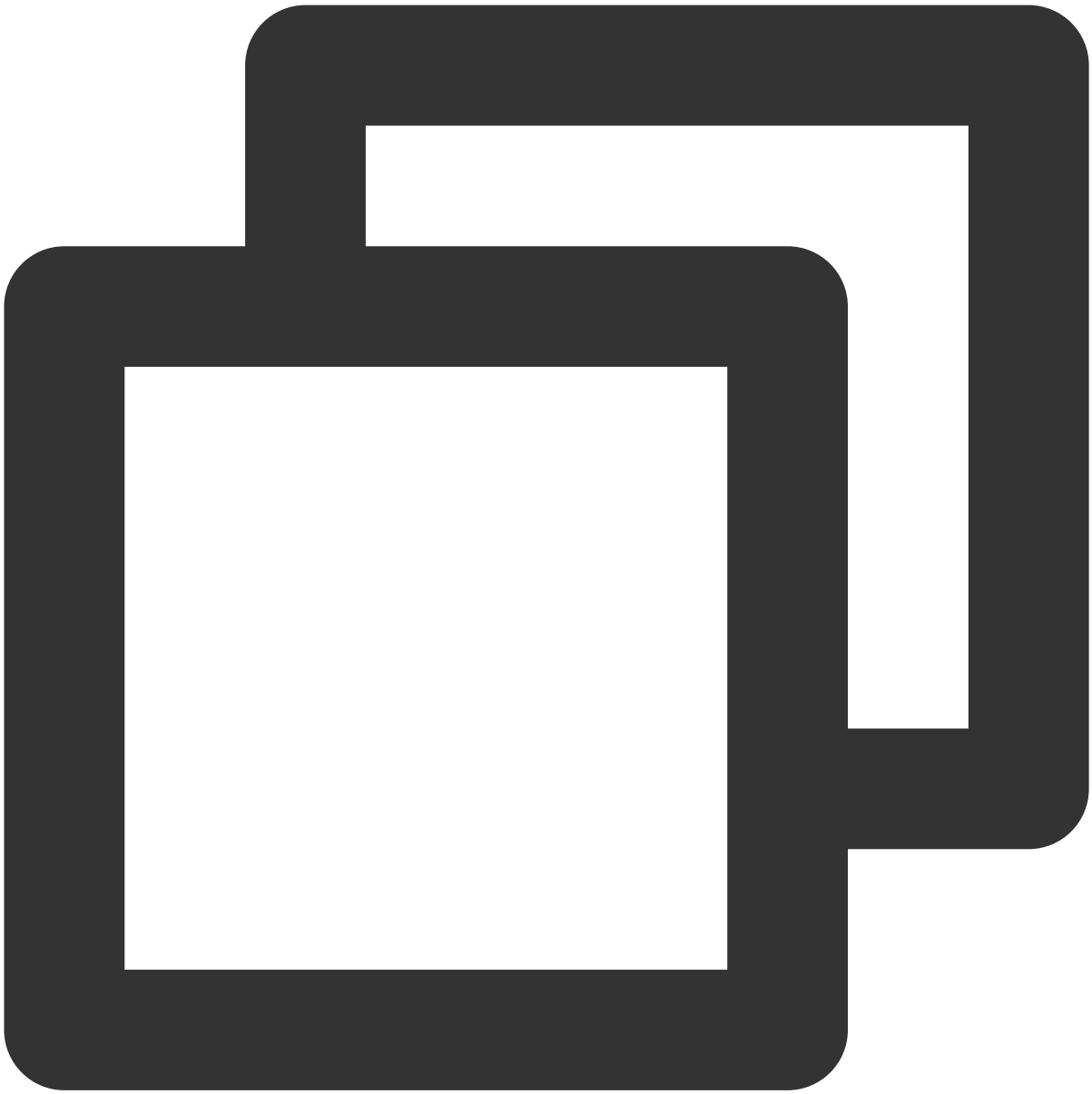


```
TencentCloudChatPush().registerPush(onNotificationClicked: _onNotificationClicked);
```

Note:

If your application needs to use **push plugin for business message notifications** , and does not immediately initiate and sign into the Chat module **after launching** , or if you need to handle business navigation through the access message click callback **before signing into the Chat module** , we suggest that you promptly invoke the `TencentCloudChatPush().registerOnNotificationClickedEvent` method, manually mount the message click callback, to access the message parameters in a timely manner.

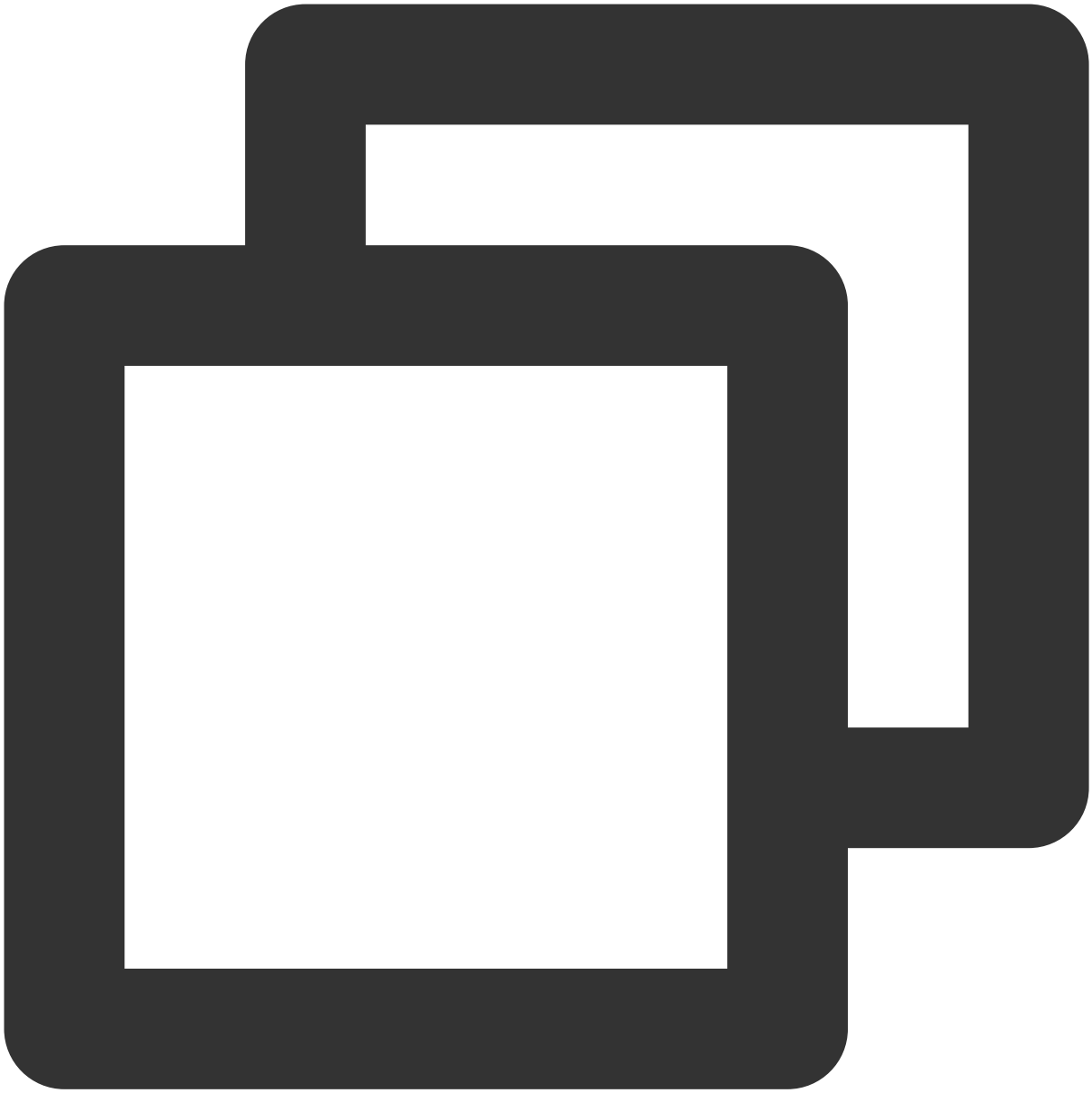
Under such circumstances, it is advisable to run this function before invoking `TencentCloudChatPush().registerPush` and position it as early as feasible in the code.



```
TencentCloudChatPush().registerOnNotificationClickedEvent(onNotificationClicked: _o
```

Step 8: Making an Offline Push Call

If you want to make an offline push call, you need to set OfflinePushInfo when calling call or groupCall .



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

```
TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);

TUICallKit.instance.call(callUserId, TUICallMediaType.audio, params);
```

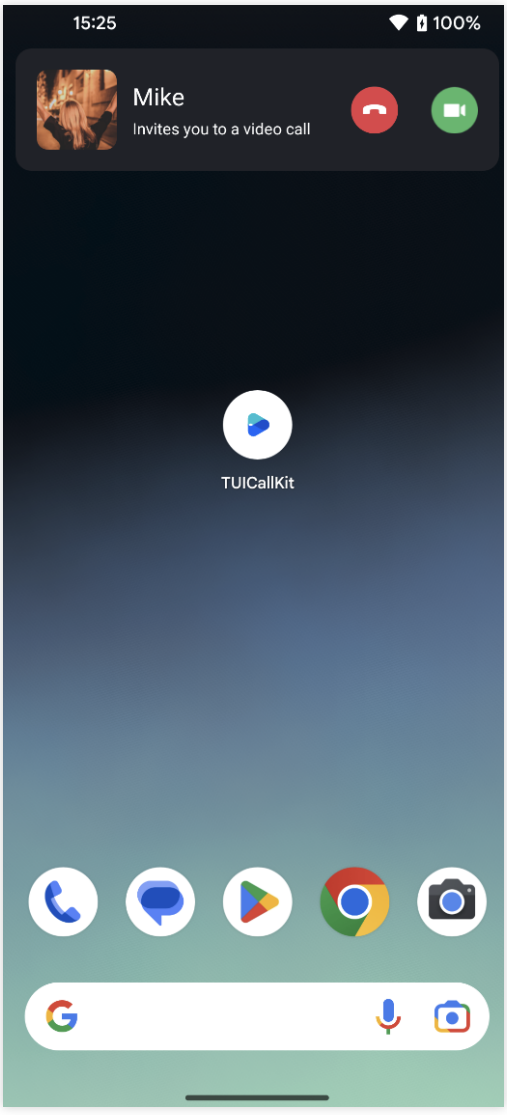
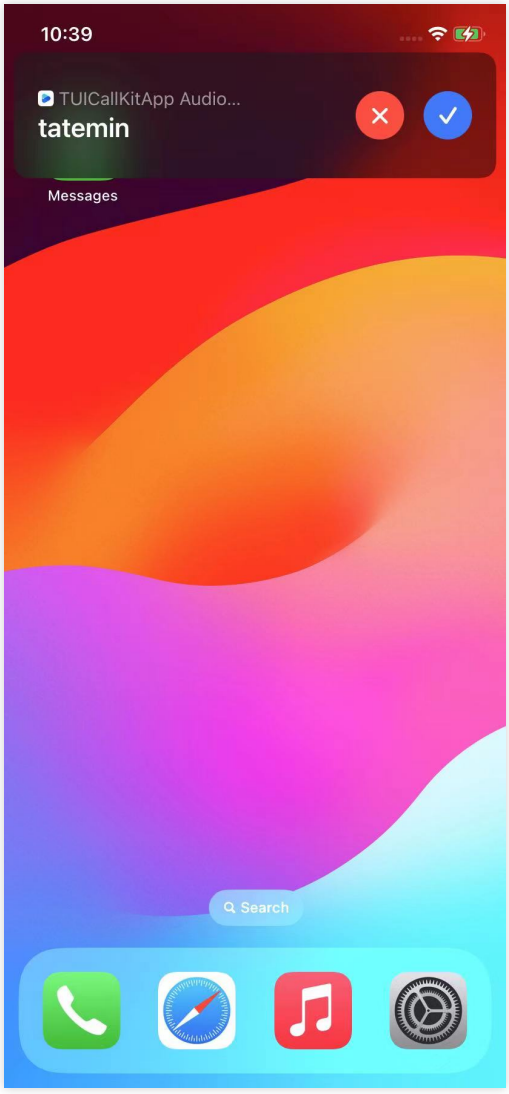
Note:

If your Android application encounters issues when receiving push notifications or pulling up pages, you can refer to the Called party's call display policy for troubleshooting.

VoIP (Optional)

Last updated：2024-07-22 17:00:45

VoIP calls are a technology for transmitting digital voice data over the internet or IP networks. They offer advantages such as low cost, high sound quality, strong flexibility, and the integration of other Communication Services.

Android	iOS
	

configuration VoIP

iOS

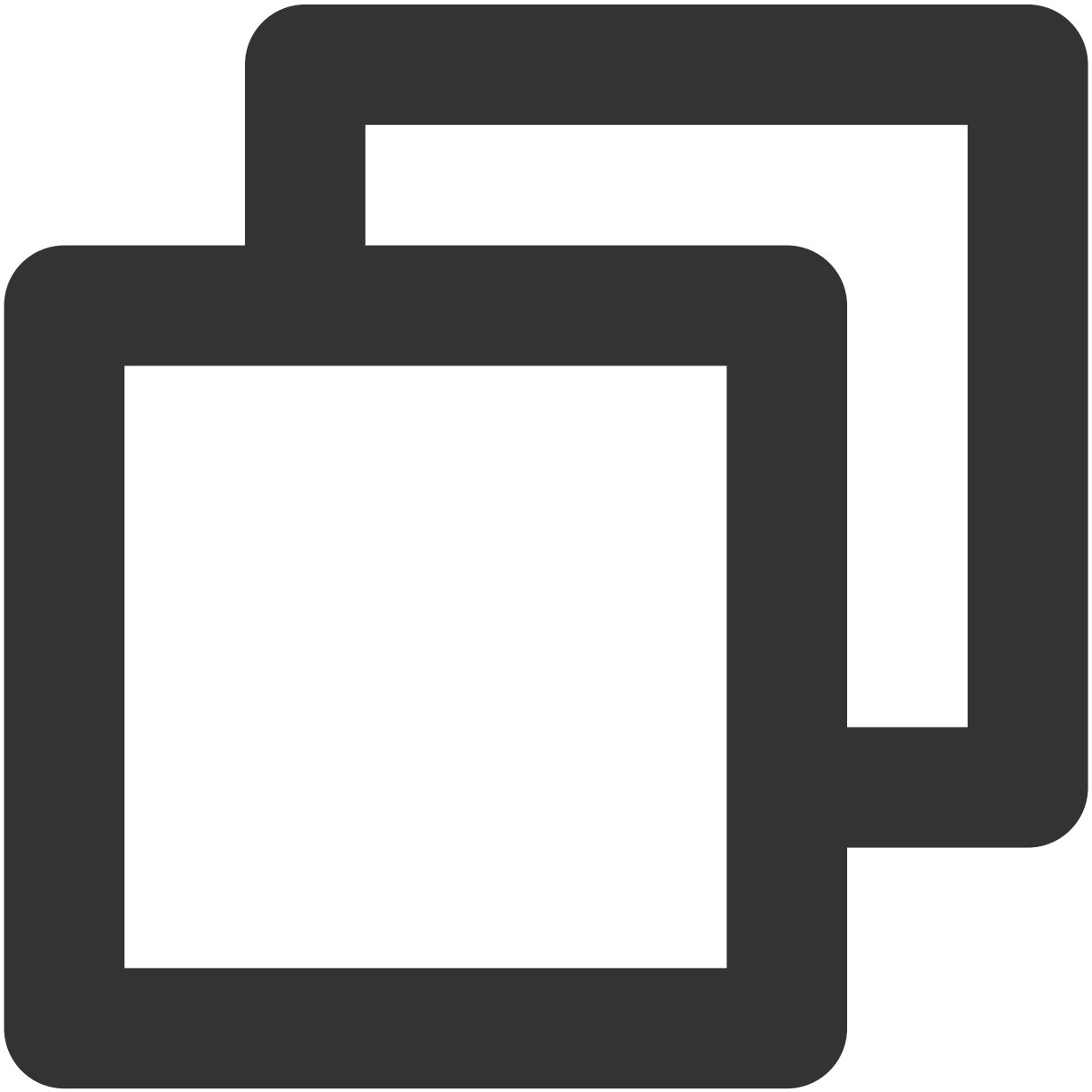
Since TIMPush currently does not support the VoIP push feature on iOS, we provide a free alternative solution to help you implement VoIP push on iOS devices: VoIP Offline Wakeup Configuration Process

Android

You can use our TIMPush plugin (paid) to implement VoIP notifications on Android devices.

Integrated message push plugin

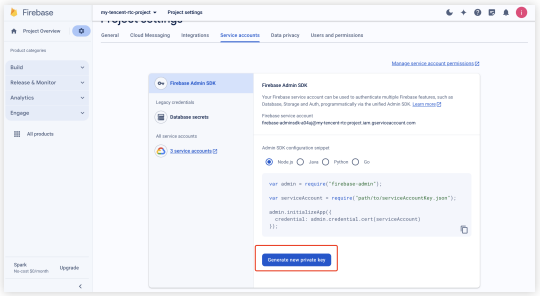
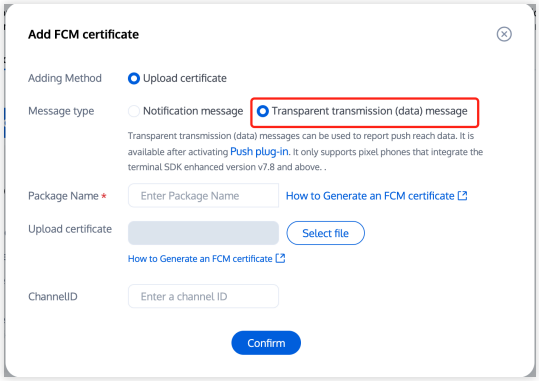
This plugin's package name on pub.dev is: tencent_cloud_chat_push. You can include it in the pubspec.yaml dependencies directory, or execute the following command for automatic installation.



```
flutter pub add tencent_cloud_chat_push
```

Preparation Requirements

1. Register your application to the FCM Push Platform to obtain parameters such as **AppID** and **AppKey**, as well as the `google-services.json` file, in order to implement the offline push feature.
2. Log in to the Tencent RTC Console, select your application, in the **Chat > Push > Access settings > Android** feature tab, choose FCM, add the FCM certificate, where the message type is selected as **Transparent transmission (data) message**.

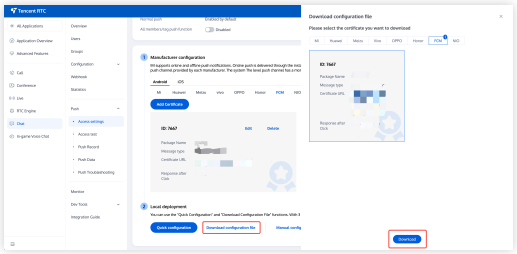
Vendor Push Platform	Configuring in the Chat console
	

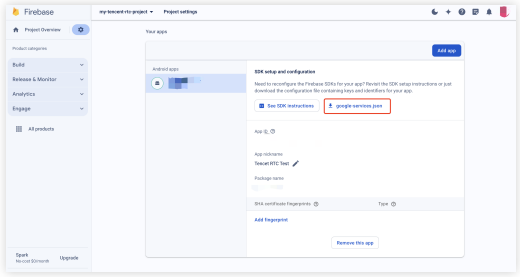
Quick Integration

1. Download and add the configuration file

After completing the vendor push information in the console, download and add the configuration file to your project. Add the downloaded `timpush-configs.json` file to the `assets` directory of the application module, and add the `google-services.json` to the project app directory.

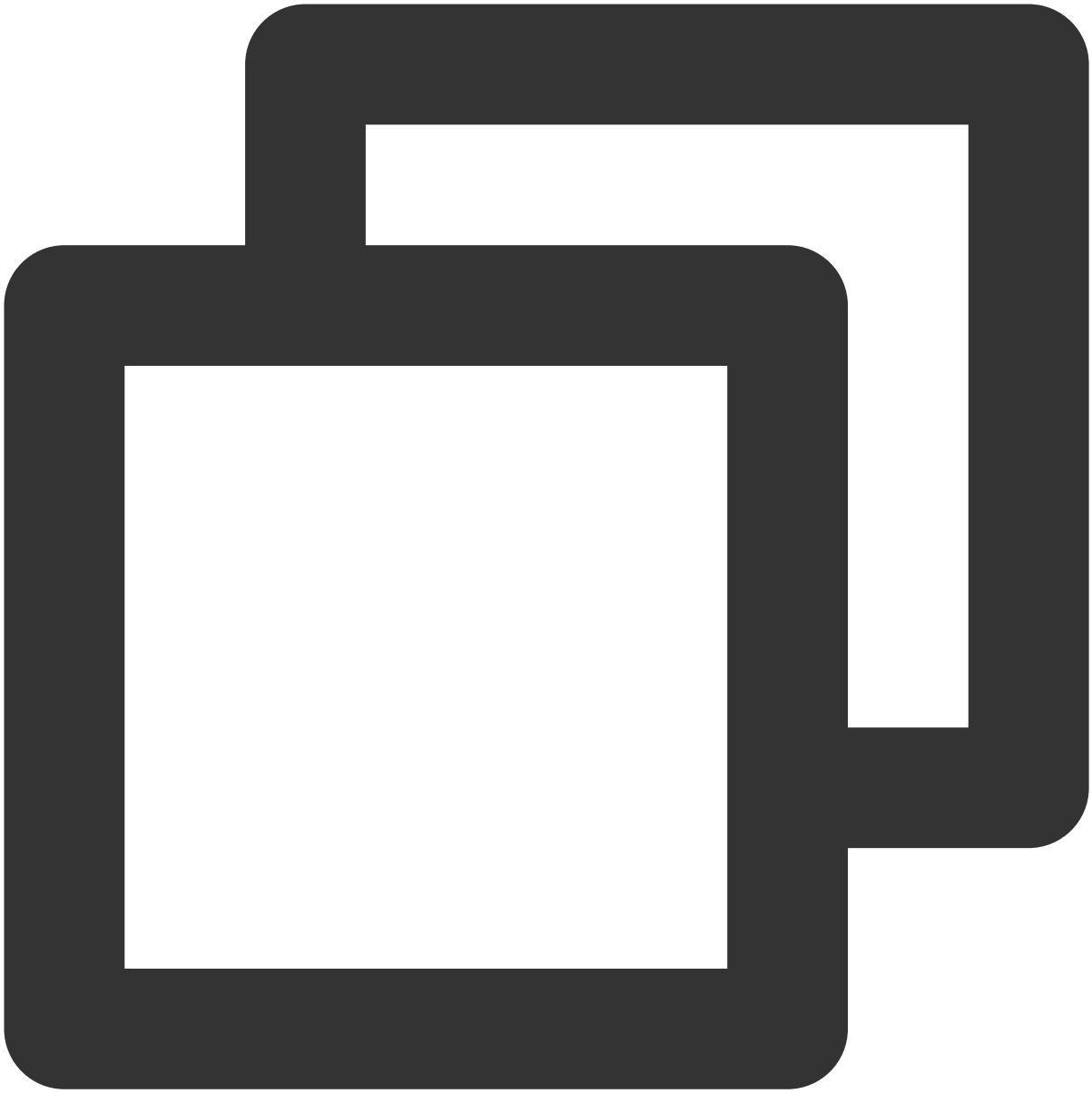
Select and download the configuration file timpush-configs.json	Download the file google-services.json





2. Integrate the TIMPush plugin

In the `build.gradle` file under the project's app directory, add the following dependency:



```
implementation "com.tencent.timpush:fcm: xxxxxx"
```

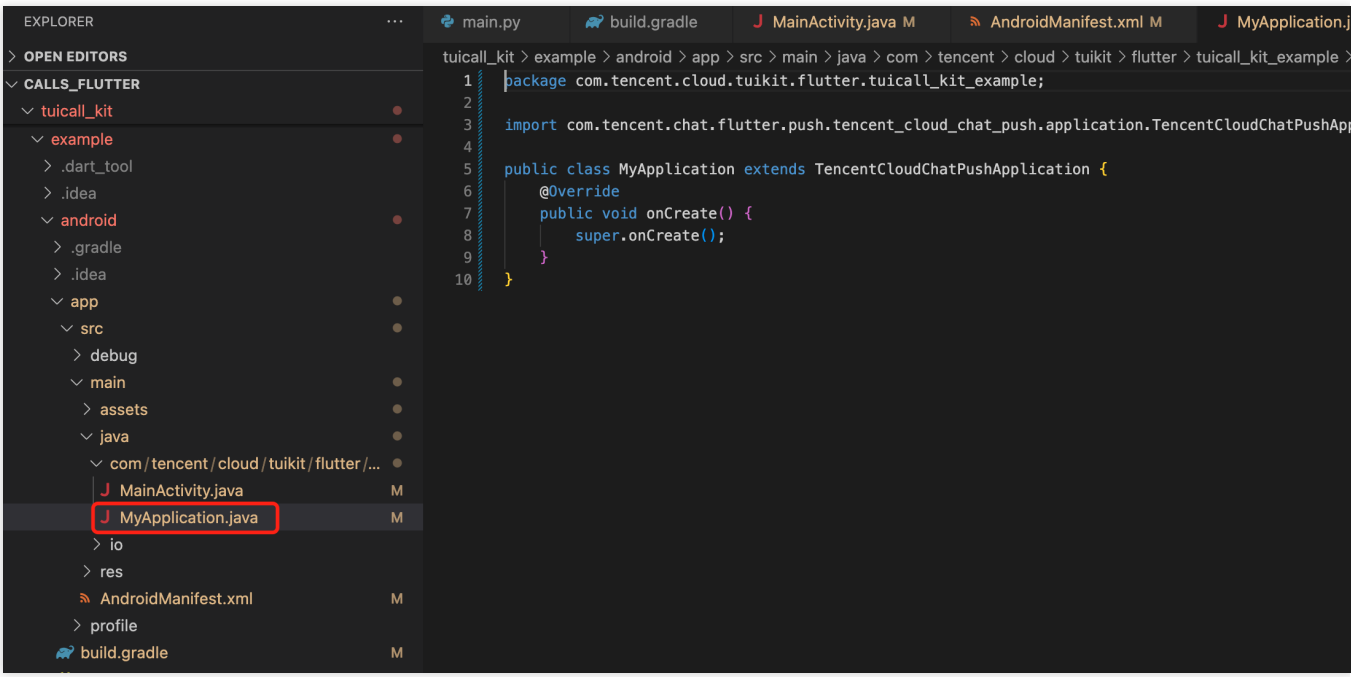
Note:

TIMPush requires integration with the Chat SDK version **7.9.5666** or above.

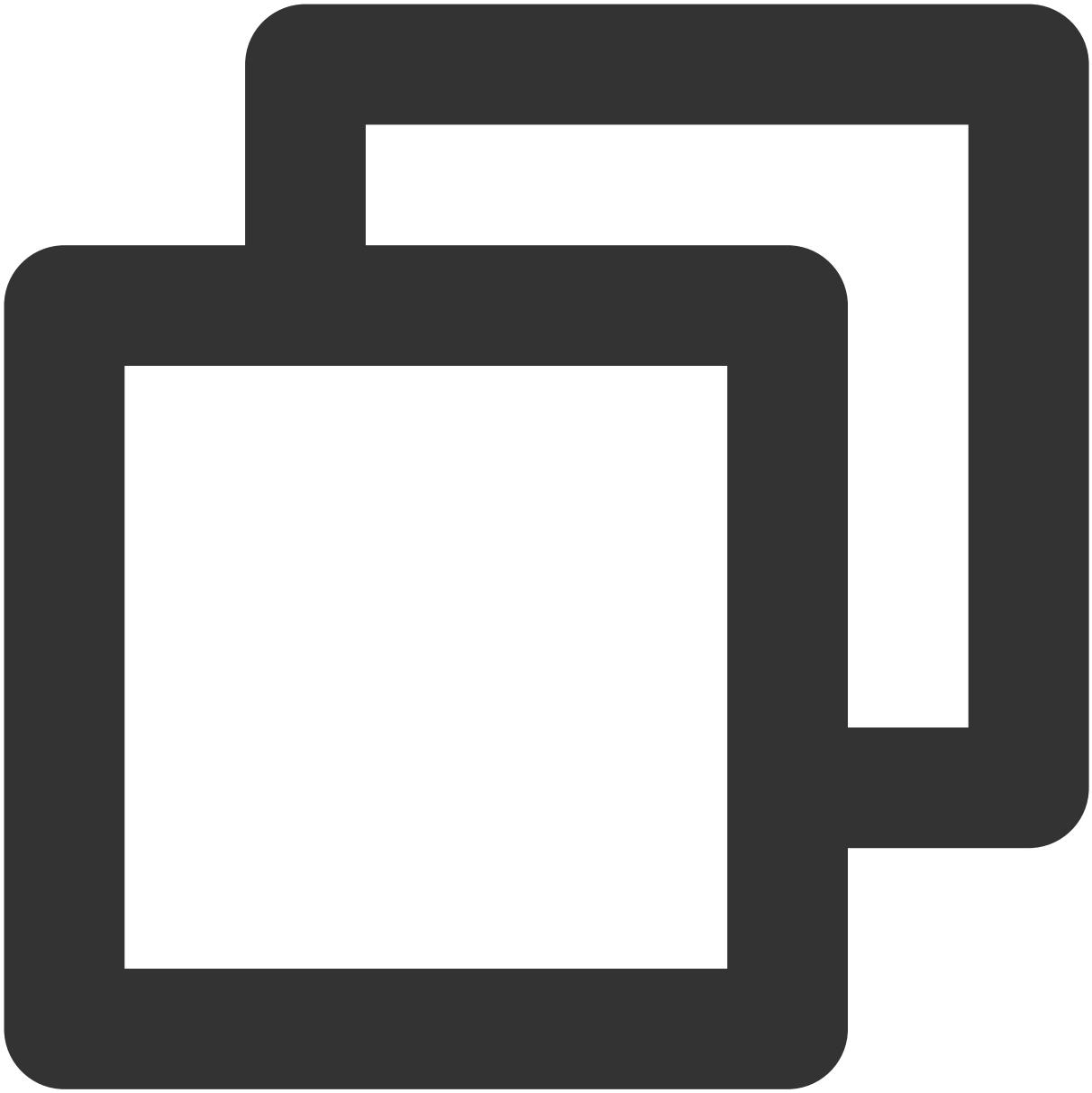
The version of the dependency added in `build.gradle` needs to correspond with the `tencent_cloud_chat_push` version.

3. Client Code Configuration

In the same directory as `MainActivity` under your project's android path, create a new Application file category, which could be named `MyApplication`.
If you have already defined your own Application class, you can directly reuse it, without the need for recreating.



Embed the following code into the file, as demonstrated above:

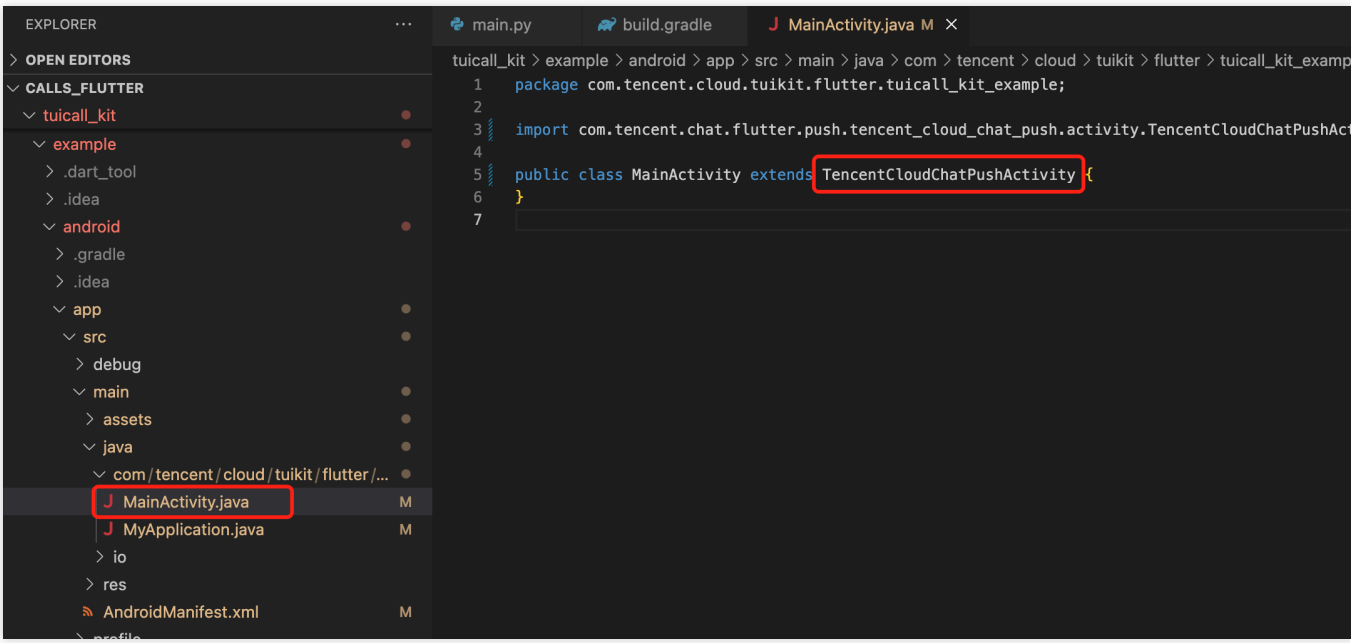


```
package xxxx.xxxx.xx

import com.tencent.chat.flutter.push.tencent_cloud_chat_push.application.TencentClo

public class MyApplication extends TencentCloudChatPushApplication {
    @Override
    public void onCreate() {
        super.onCreate();
    }
}
```

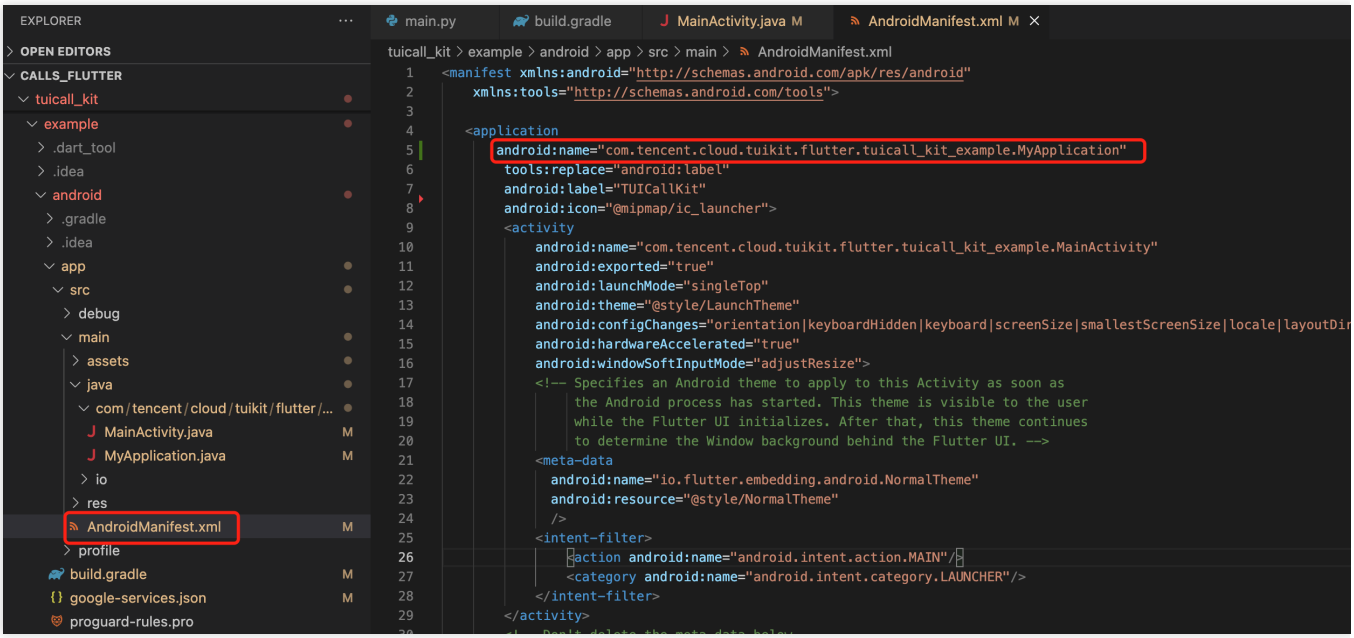
Note:

If you have already created your own Application for other purposes, simply `extend TencentCloudChatPushApplication` and ensure that `onCreate()` method is called in `super.onCreate();` .

Meanwhile, you also need to modify your MainActivity File:

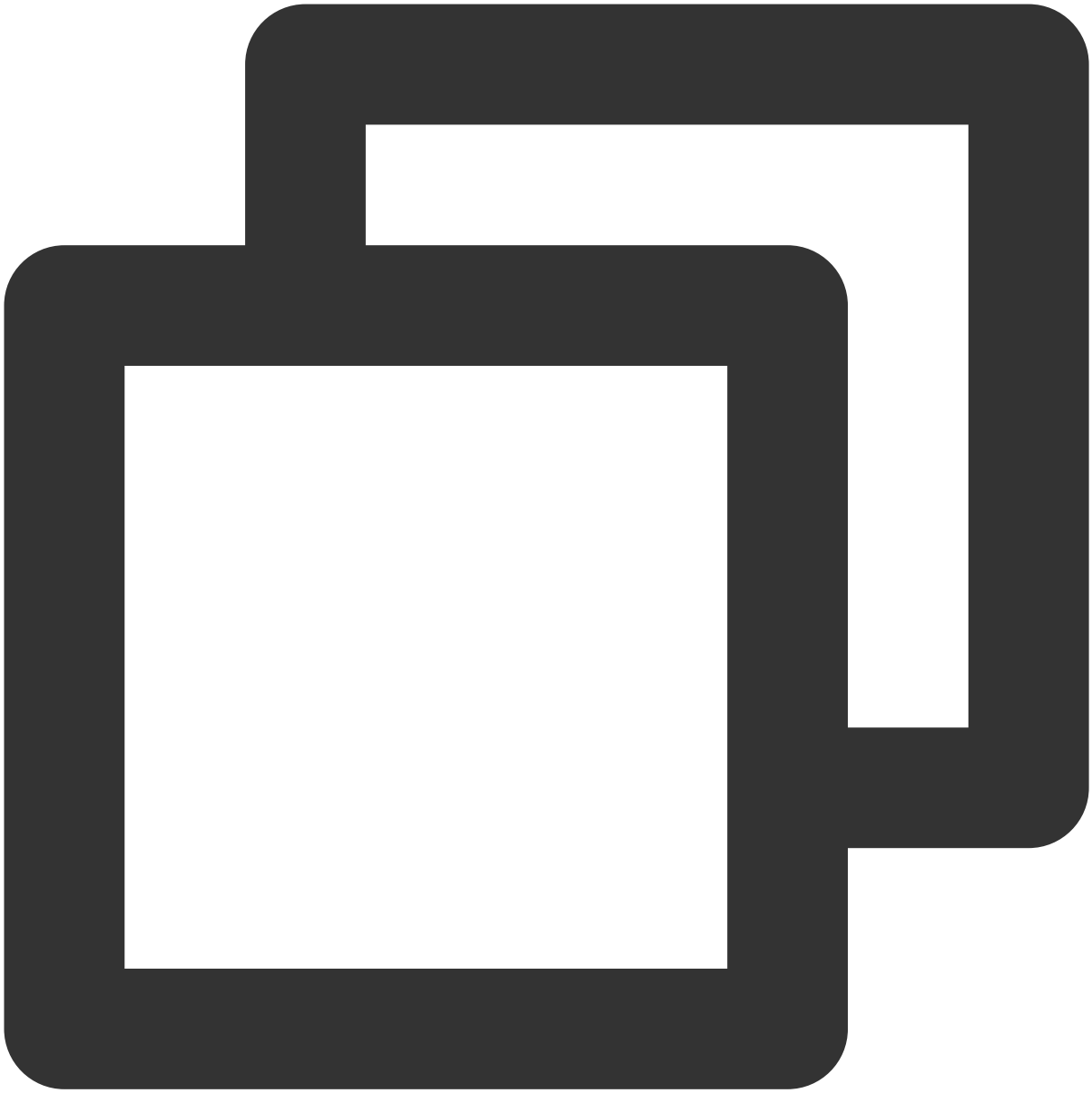


Open the `android/app/src/main/AndroidManifest.xml` file, then add a specific `android:name` parameter to the `<application>` Tag, which is linked to your newly created Definition `Application` class as shown in the figure:



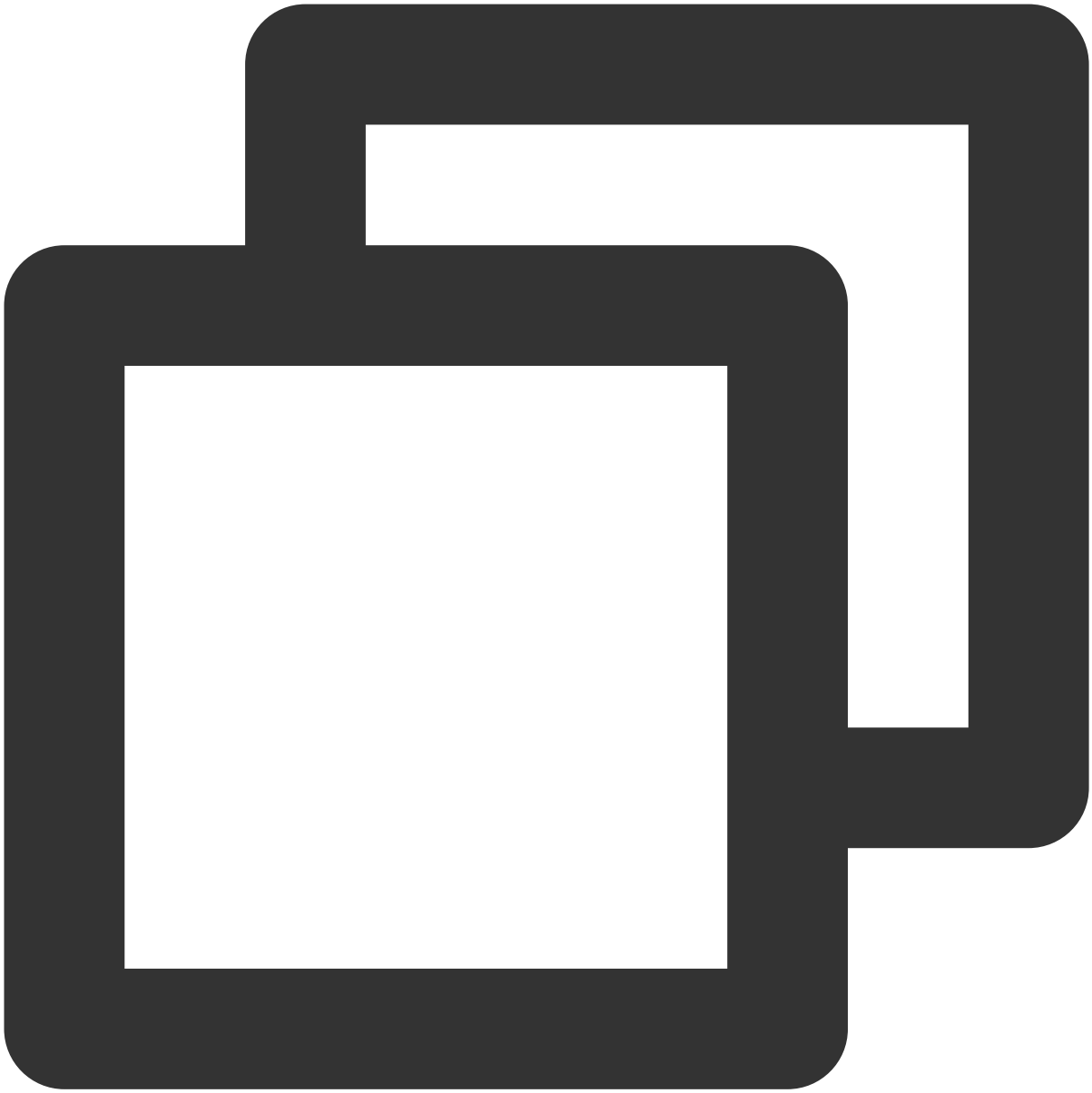
4. Complete project configuration

In the project-level `build.gradle` file, under buildscript -> dependencies, add the following configuration:



```
buildscript {
    dependencies {
        classpath 'com.google.gms:google-services:4.3.15'
    }
}
```

In the `build.gradle` file under the project's app directory, add the following configuration:



```
apply plugin: 'com.google.gms.google-services'
```

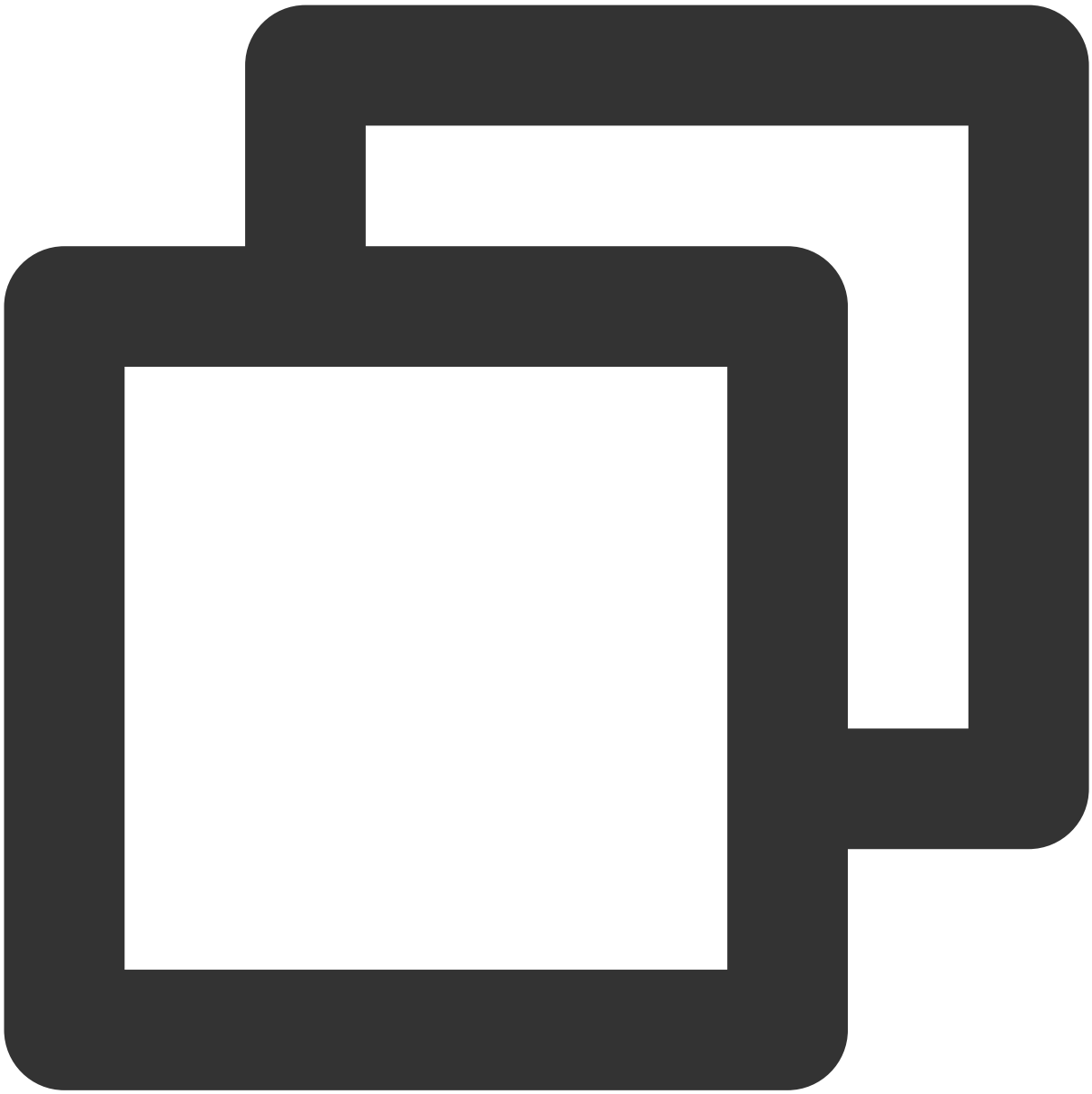
5. Register push plugin

Please register the push plugin immediately after logging in.

Invoke `TencentCloudChatPush().registerPush` method, it necessitates the transmission of BackDefinition's click callback function.

Moreover, you may also choose to input `apnsCertificateID` , the iOS push certificate ID, and `androidPushOEMConfig` , the Android push vendor configuration. These two configurations were previously

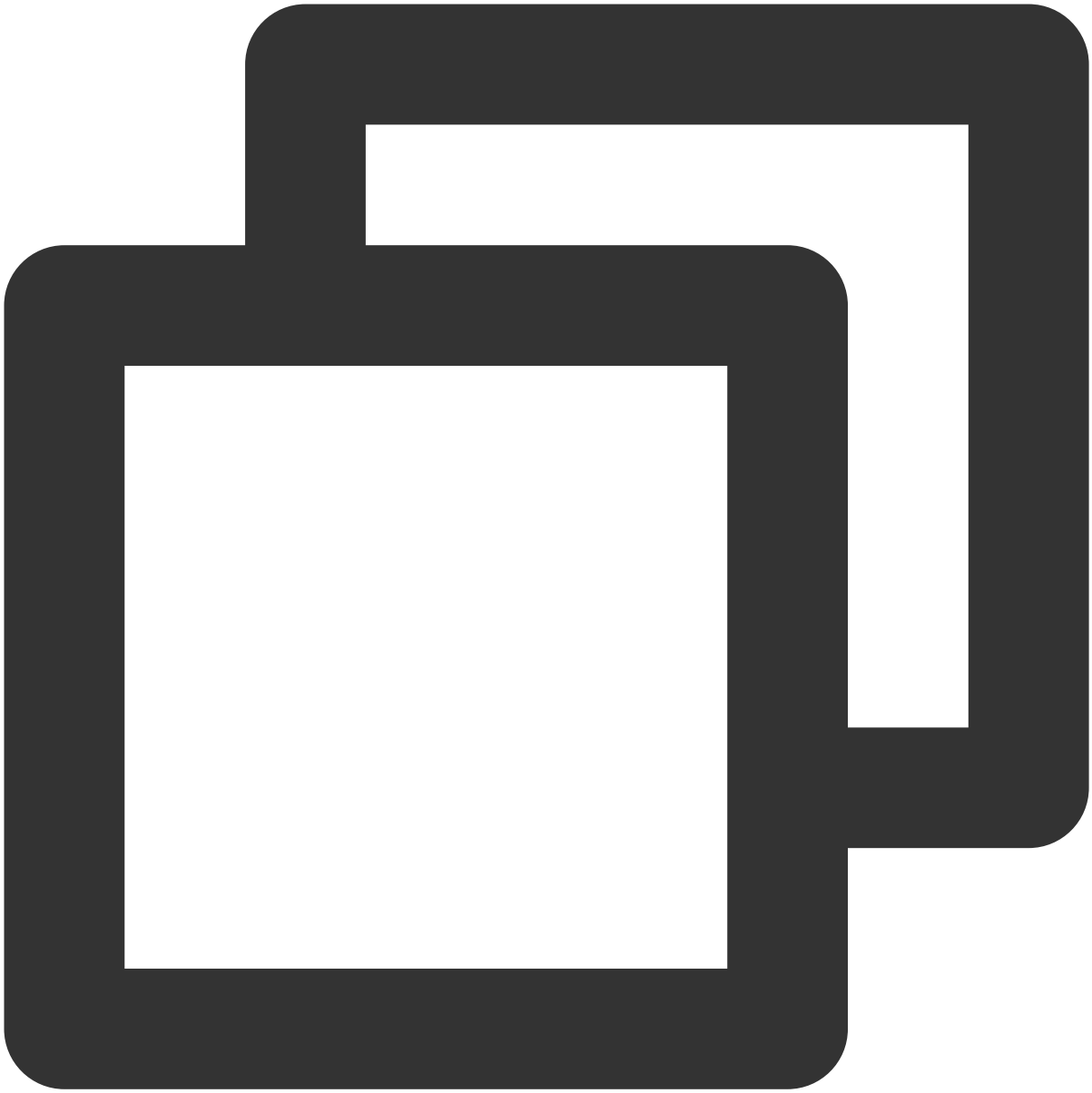
specified in the initial steps; should there be no need for amendment, there is no requirement to input them again.



```
TencentCloudChatPush().registerPush(onNotificationClicked: _onNotificationClicked);
```

6. Implement auto-log in

The data mode of FCM can only wake up the offline app, so you also need to implement login and app launching in the `onAppWakeUpEvent` .

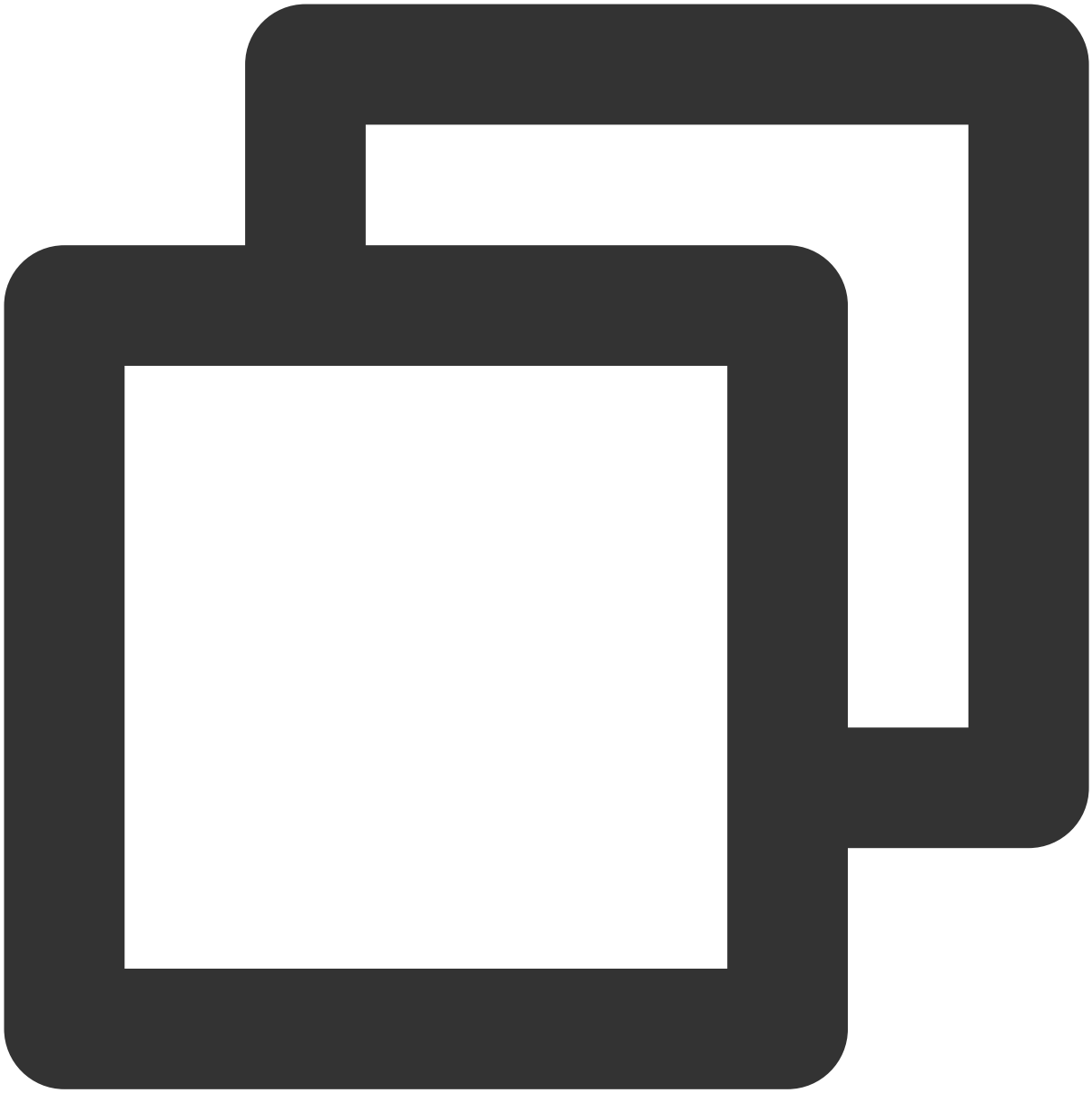


```
TencentCloudChatPush().registerOnAppWakeUpEvent(onAppWakeUpEvent: () {
      // TODO: log in operation
});
```

After completing the above steps, you can use the offline push capability of TIMPush in conjunction with TUICallKit.

7. Making an Offline Push Call

If you want to make an offline push call, you need to set OfflinePushInfo when calling call.



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

 TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);
```

```
TUICallKit.instance.call(callUserId, TUICallMediaType.audio, params);
```

Note:

If your Android application encounters issues when receiving push notifications or pulling up pages, you can refer to the Called party's call display policy for troubleshooting.

FAQs

If the application is terminated, the incoming call UI cannot be displayed?

Confirm receipt of push. If push cannot be received, check if the certificates are correctly uploaded to the Chat console. Refer to the first step in the above document **Quick Integration** to see if it has been added correctly.
Confirm that FCM Data Message was selected in the console, in accordance with the second step of the above **Preparation Requirements**;
Confirm receipt of data message, filter logs (keyword: TIMPush), check the following logs for printouts (if messages are not received, you can use the Troubleshooting Tool to investigate reasons);

Confirm implementation of **auto-log in**. It is only after auto-login that call requests are pulled and the incoming call UI is displayed.

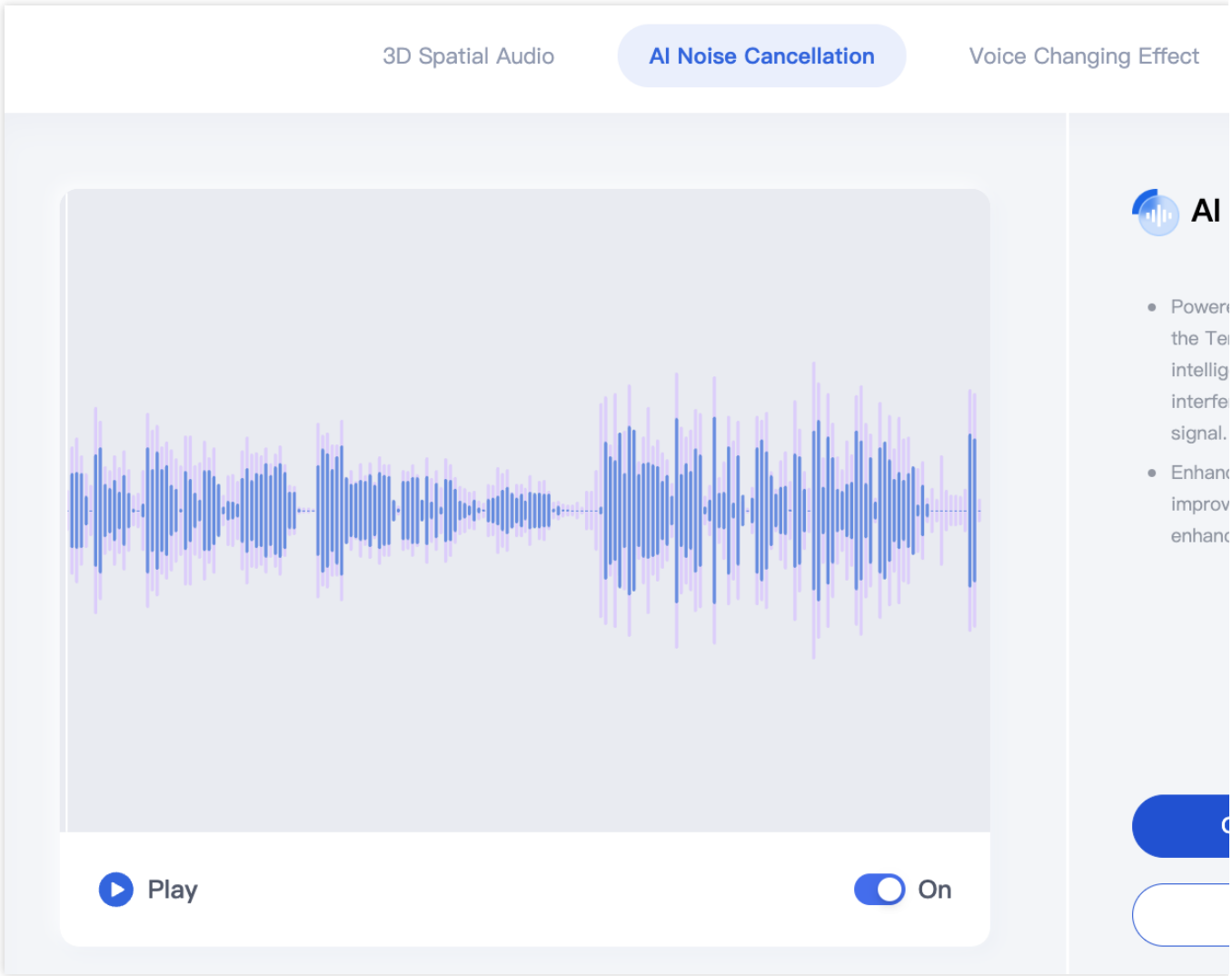
AI Noise Reduction (TUICallKit)

Last updated：2024-06-17 17:41:52

AI Noise Suppression originates from the AI algorithms of Tencent Tianlai Laboratory. It is capable of intelligently detecting and eliminating noise interference mixed in the transmitted signal, thereby significantly improving the quality of voices, enhancing clarity, and improving user listening experience.
The application of AI Noise Suppression features within TUICallKit enables users to experience clear and stable sound in various environments such as offices, internet cafes, malls, outdoors, etc.

Trying It Online

You can also enter our Tencent RTC Demo to try the excellent sound effects brought by AI Noise Reduction online.



Activate AI Noise Suppression

TUICallKit now has AI Noise Suppression feature activated by default. Users don't need to make any extra settings or operations to enjoy high-quality noise reduction effects in the app.

Virtual Background (TUICallKit)
iOS

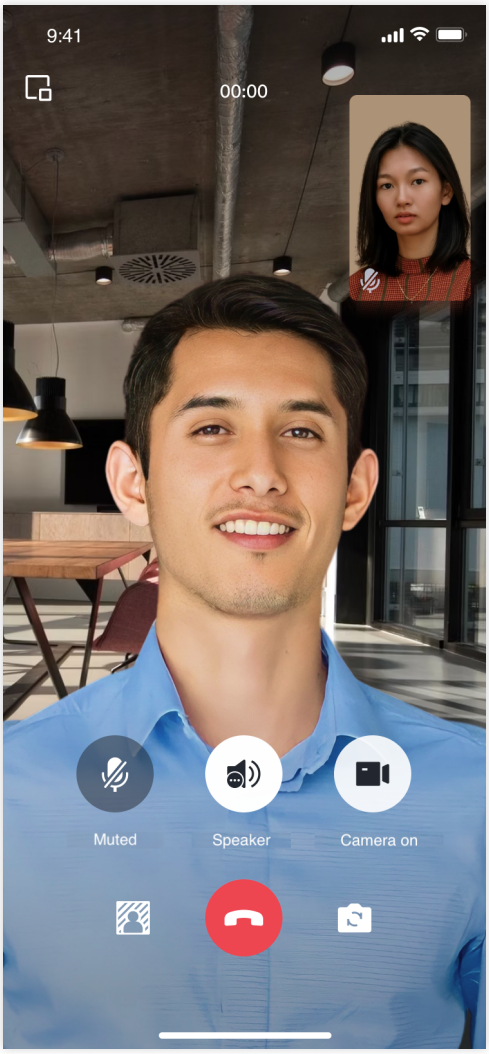
Last updated：2024-07-22 17:00:45

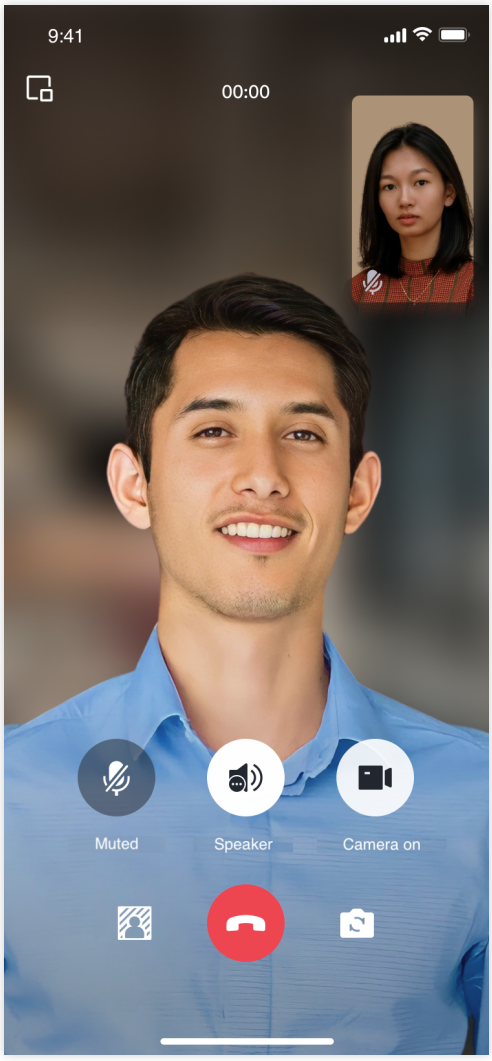
TUICallKit has launched a new feature for virtual backgrounds, allowing users to set a blurry or image background during video calls. This hides the real calling environment, protects privacy, and makes the call more interesting. Next, this article will detail how to use this feature in the TUICallKit component.

Integration effect

The display effect of the TUICallKit component after integrating the virtual background feature is as follows:

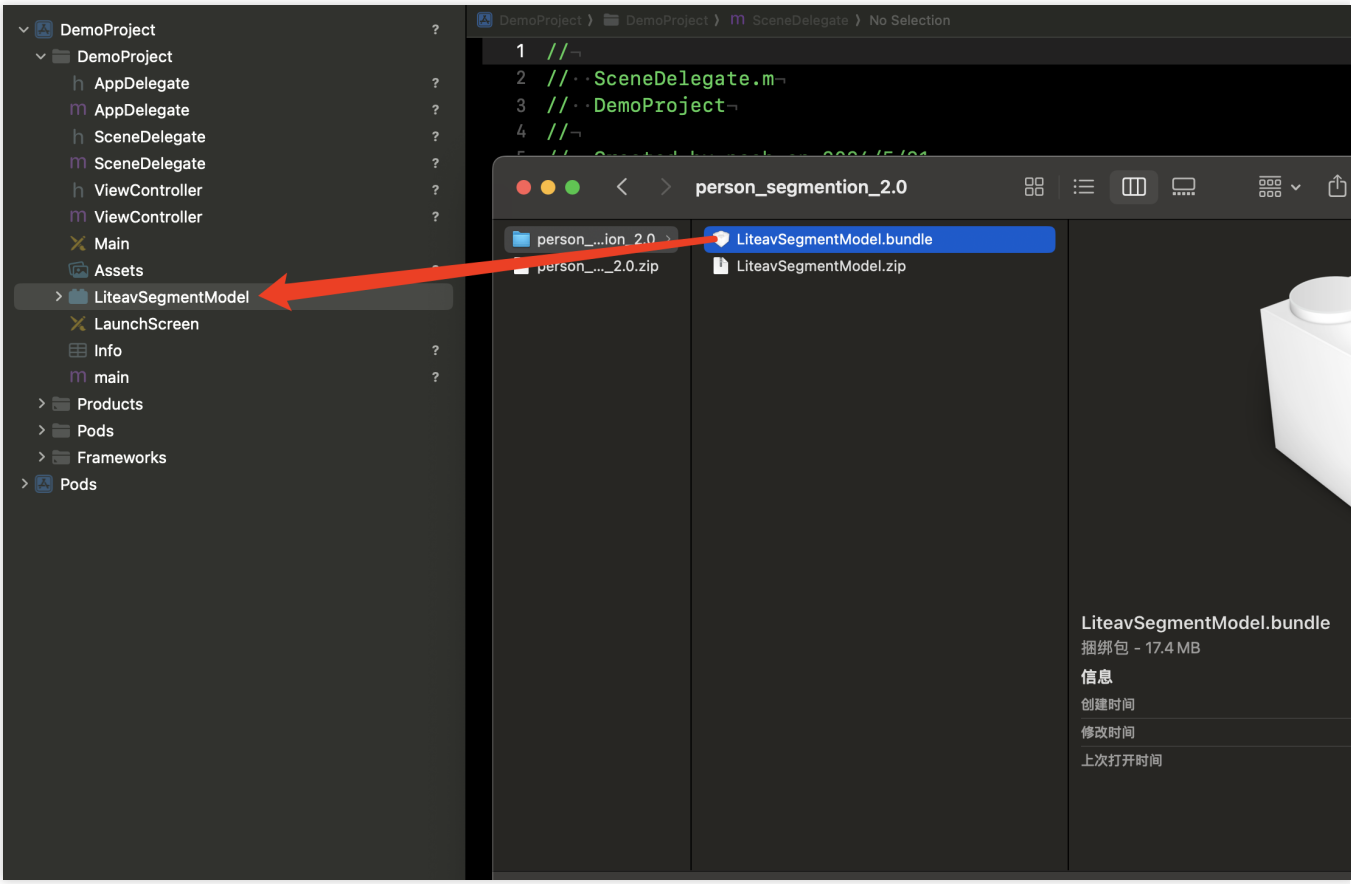
Original Camera	Blurry Background Effect	I



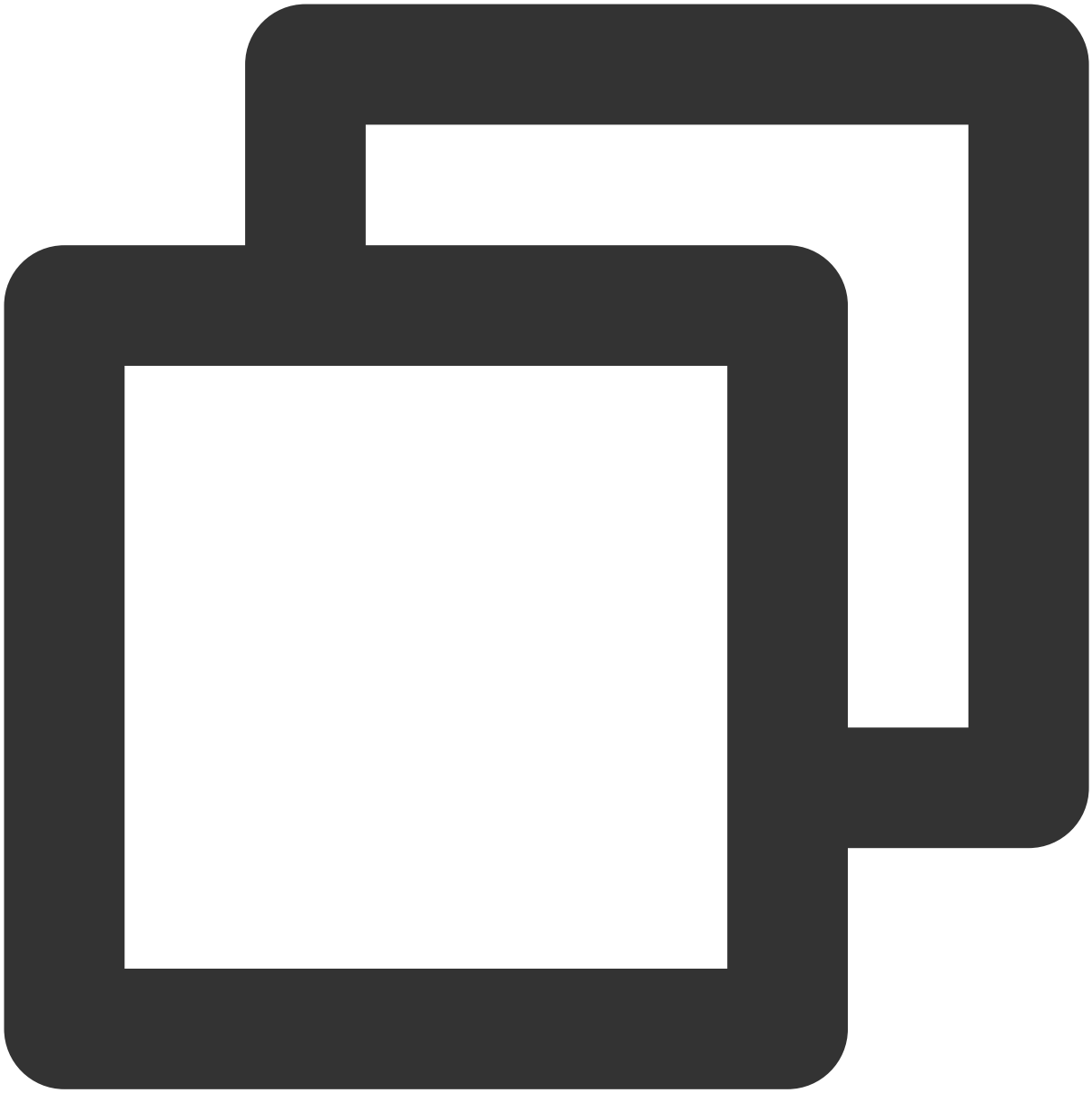


Preparation Requirements

1. Before using the virtual background feature provided by Tencent Cloud, you need to go to the Console to activate Audio and Video Services for your application, and purchase the `Group Call` package. For detailed steps, please refer to Activate Service.
2. Download the Virtual Background Model file, extract it, and drag the `LiteavSegmentModel.bundle` file into your project, ensuring it's in the `MainBundle` .



3. Modify the dependencies in your Podfile to replace them with the `Professional` version, and specify the version of TXLiteAVSDK_Professional as `11.8.15669` .



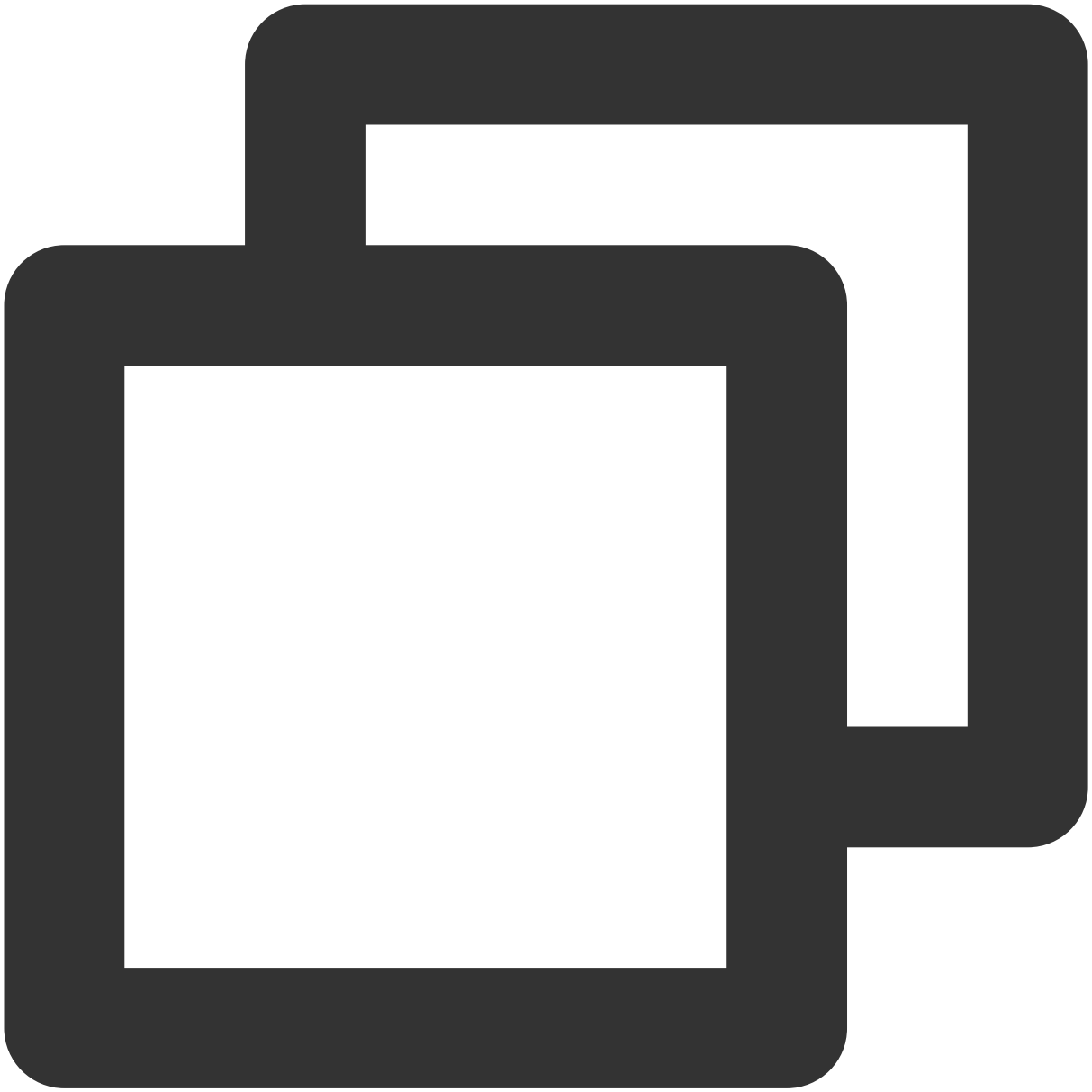
```
pod 'TUICallKit_Swift/Professional'
pod 'TXLiteAVSDK_Professional', '11.8.15669'
```

Note:

There is a corresponding relationship between the version of the TRTC SDK and the model file, please ensure the version number matches the model, see below: Model File Matching with SDK.

Enable Blurry Background

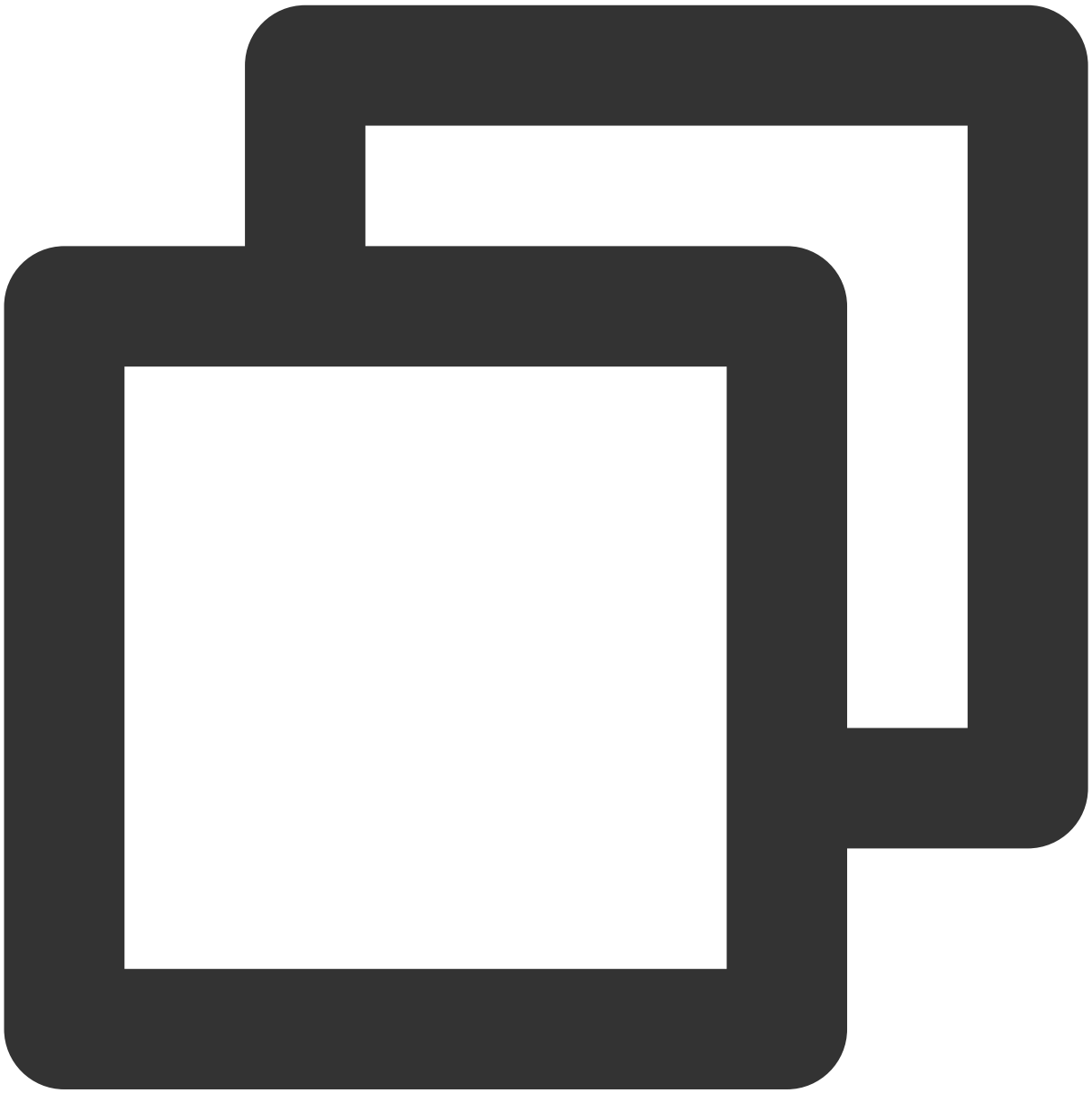
TUICallKit's UI design supports setting a blurry background. By calling the following interface, you can display a feature button for the blurry background on the UI. Clicking the button will directly enable the blurry background feature.



```
import TUICallKit

TUICallKit.createInstance().enableVirtualBackground(enable: true)
```

Setting Image Background (Optional)

Implementing the image background requires users to save the image locally. After saving, call the following interface for setting (currently, only images with a local path are supported, network images are not supported yet).



```
import TUICallEngine

TUICallEngine.createInstance().setVirtualBackground("imagePath") { code, message in
}
```

FAQs

Enabling blurry background has no response or is delayed?

Ensure you have purchased the `Group Call` package, see Activate Service for more details.
Ensure the model file is downloaded to local.
If a model file is not added to the local path, when enabling the blurry background feature, the SDK will then download the model file. Under normal network conditions, the download takes 1~3s; the poorer the network, the longer it will take.
Check if the model file and SDK are a match.

Matching model files with SDK?

TUICallKit is a video and audio call scenario implemented based on the C SDK and TRTC SDK. The virtual background is a distinctive feature provided by TRTC SDK. It's important to note that the virtual background model file needs to match the version of the TRTC SDK; otherwise, the blurry background feature may not function properly. The table below lists the relationships between the model files and the TRTC SDK versions:

SDK Version	Virtual background model file Download address·
pod 'TXLiteAVSDK_Professional', '11.8.15669'	Download version_2.0
pod 'TXLiteAVSDK_Professional', '11.7.12001'	Download version_1.0

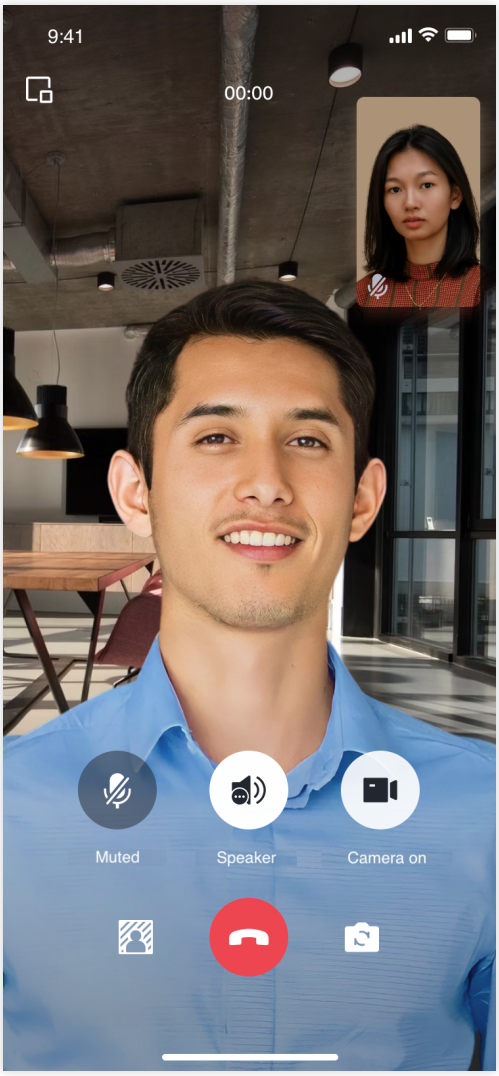
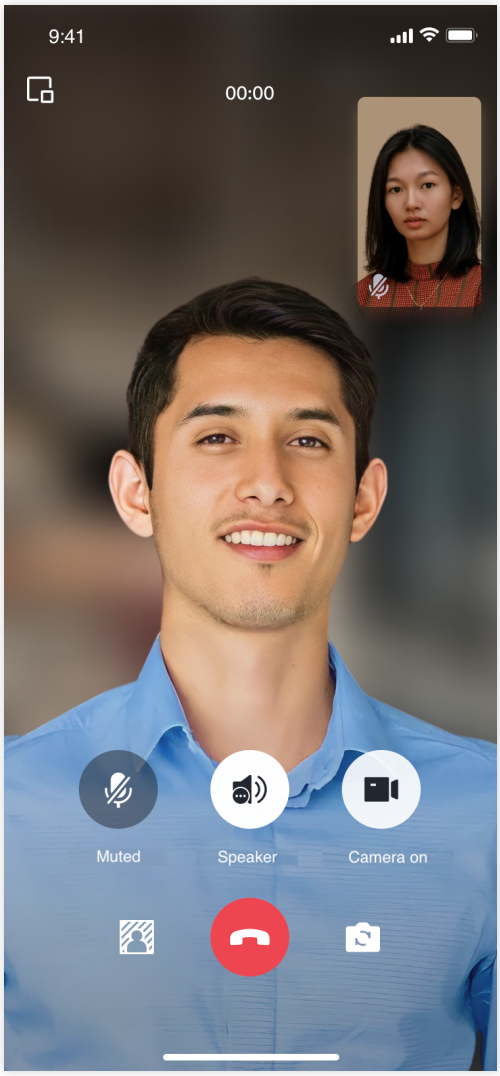
Android

Last updated：2024-07-22 17:00:45

TUICallKit has launched a new feature for virtual backgrounds, allowing users to set a blurry or image background during video calls. This hides the real calling environment, protects privacy, and makes the call more interesting. Next, this article will detail how to use this feature in the TUICallKit component.

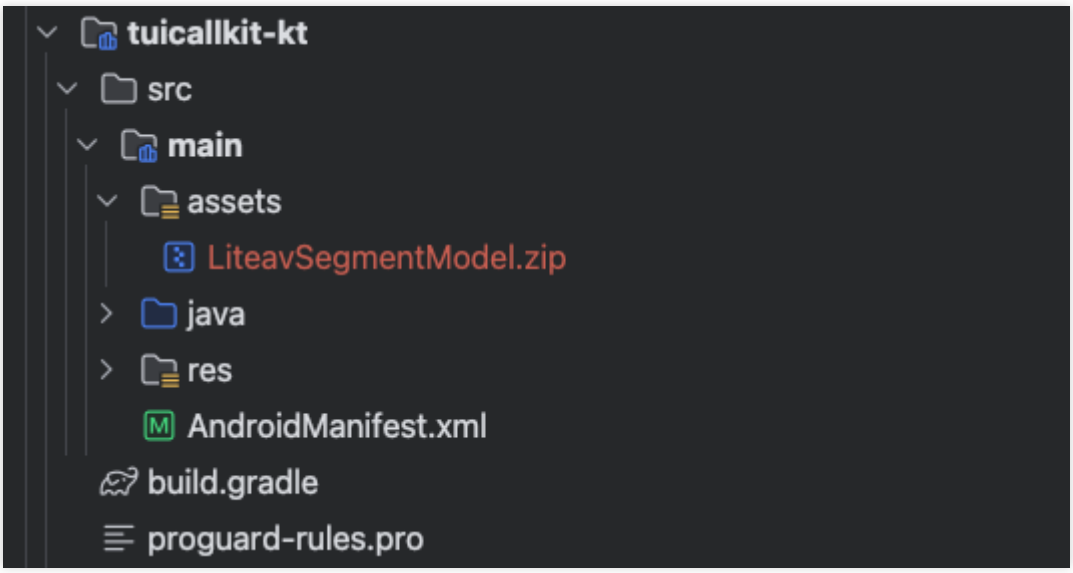
Integration effect

The display effect of the TUICallKit component after integrating the virtual background feature is as follows:

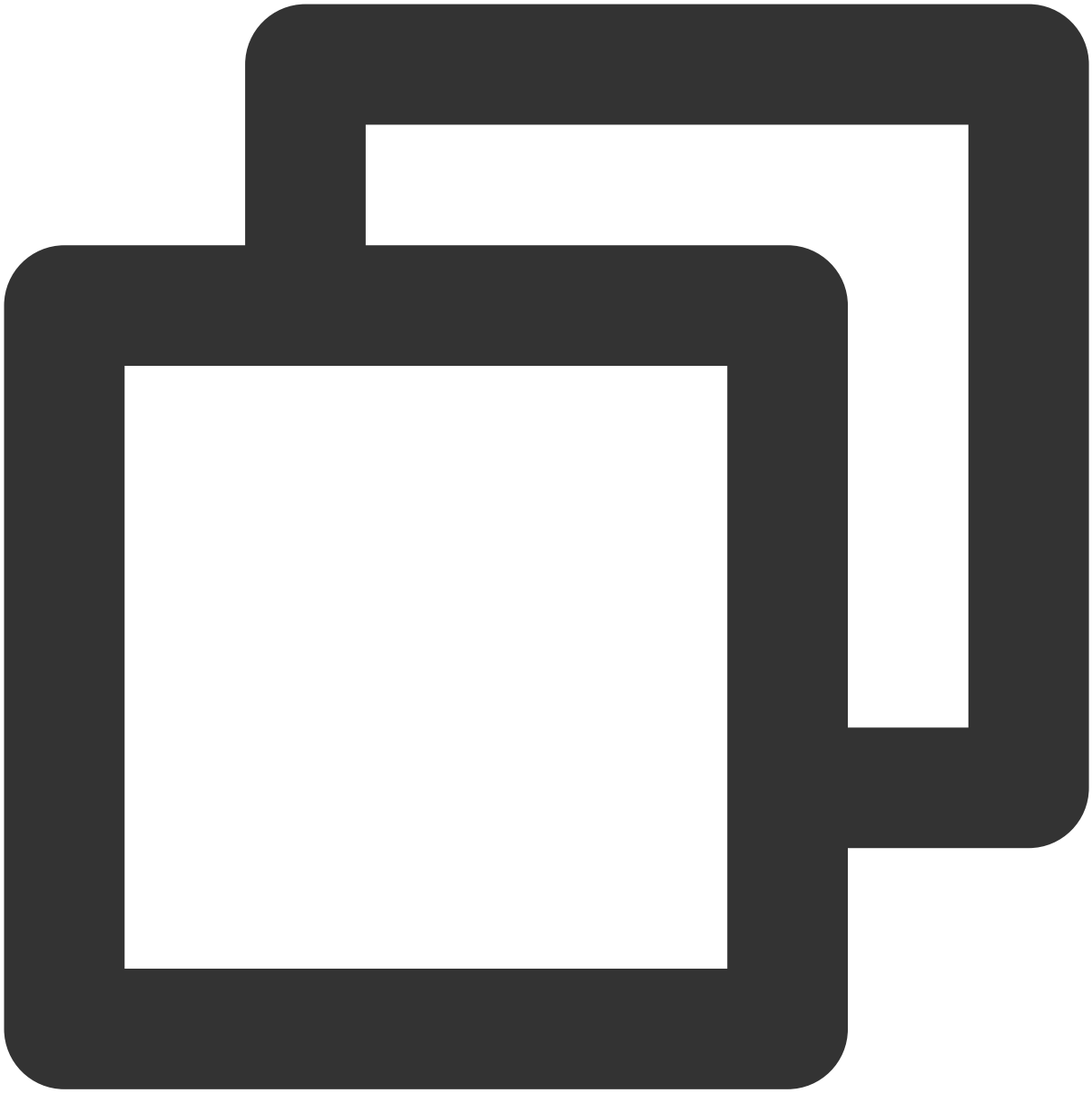
Original Camera	Blurry Background Effect
	

Preparation Requirements

1. Before using the Virtual Background feature provided by Tencent Cloud, you need to visit the Console to activate Audio and Video Services for your application and purchase the `Group Call` package. For specific steps, please see Activate Service.
2. Download the Virtual Background Model file, unzip it, and copy the `LiteavSegmentModel.zip` file to the `assets` directory of your project.



3. In the `tuicallkit-kt` directory of the project, find the `build.gradle` file and replace the `TRTC SDK` version with the Professional Version.



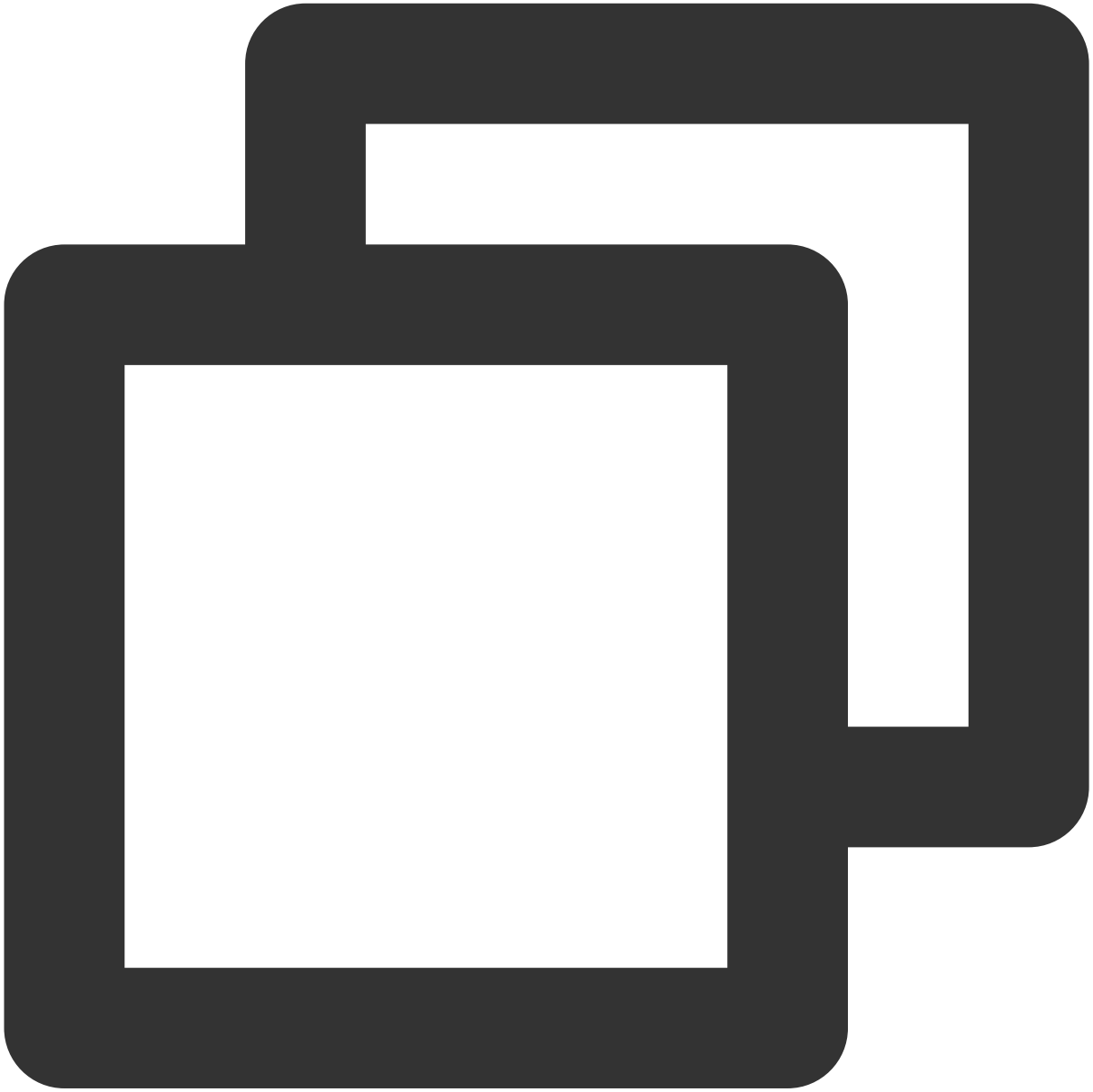
```
api "com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176"
```

Note:

There is a corresponding relationship between the version of the TRTC SDK and the model file, please ensure the version number matches the model, see below: Model File Matching with SDK.

Enable Blurry Background

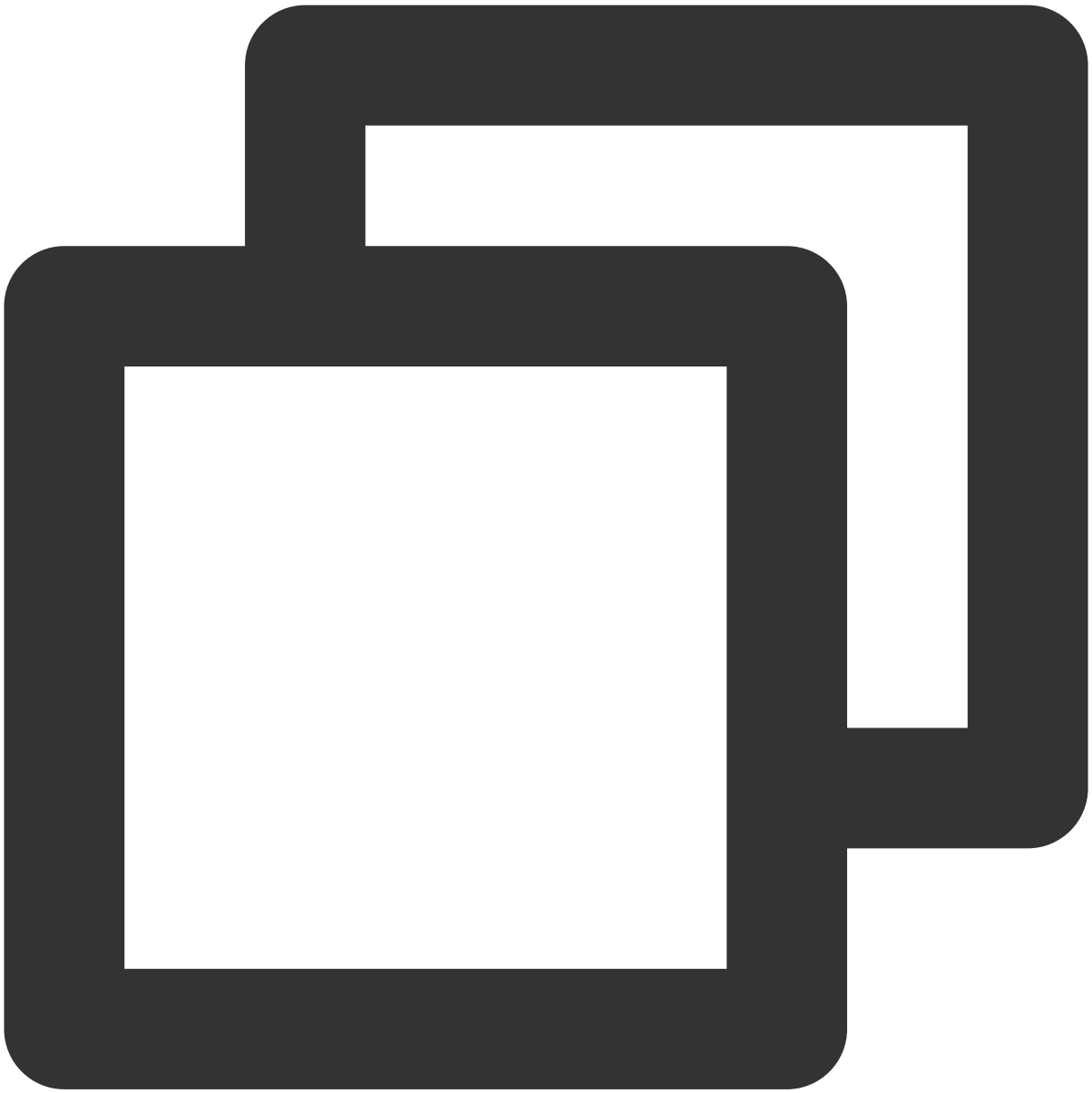
TUICallKit's UI design supports setting a blurry background. By calling the following interface, you can display a feature button for the blurry background on the UI. Clicking the button will directly enable the blurry background feature.



```
TUICallKit.createInstance(getApplicationContext()).enableVirtualBackground(true);
```

Setting Image Background (Optional)

Implementing the image background requires users to save the image locally. After saving, call the following interface for setting (currently, only images with a local path are supported, images of uri are not supported yet).



```
TUICallEngine.createInstance(context).setVirtualBackground("imagePath", null)
```

FAQs

Enabling blurry background has no response or is delayed?

Ensure you have purchased the audio and video call `Group Call` package, see Activate Service.
Ensure the model file is downloaded to local.
If a model file is not added to the local path, when enabling the blurry background feature, the SDK will then download the model file. Under normal network conditions, the download takes 1~3s; the poorer the network, the longer it will take.
Check if the model file and SDK are a match.

Matching model files with SDK?

TUICallKit is based on the Chat SDK and TRTC SDK for audio and video call scenarios. Virtual background is a feature provided by TRTC SDK. There is a matching relationship between the virtual background model file and the TRTC SDK version. If they do not match, enabling the blurry background might be ineffective. The relationship between the model file and TRTC SDK is as below:

SDK Version	Virtual background model file Download address
com.tencent.liteav:LiteAVSDK_Professional:11.7.0.12001	segmention_1.0
com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176	segmention_2.0

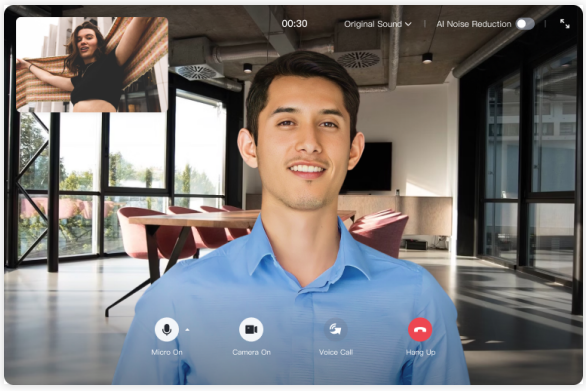
Web

Last updated：2024-07-09 16:08:59

TUICallKit has launched a new feature for virtual backgrounds, allowing users to set a blurry or image background during video calls. This hides the real calling environment, protects privacy, and makes the call more interesting. Next, this article will detail how to use this feature in the TUICallKit component.

Integration effect

The display effect of the TUICallKit component after integrating the virtual background feature is as follows.

Original Camera	Blurry Background Effect
	

Preparation Requirements

Before using the Virtual Background feature provided by Tencent Cloud, you need to visit the Console to activate Audio and Video Services for your application and purchase the `Group Call` package. For specific steps, please see Activate Service.

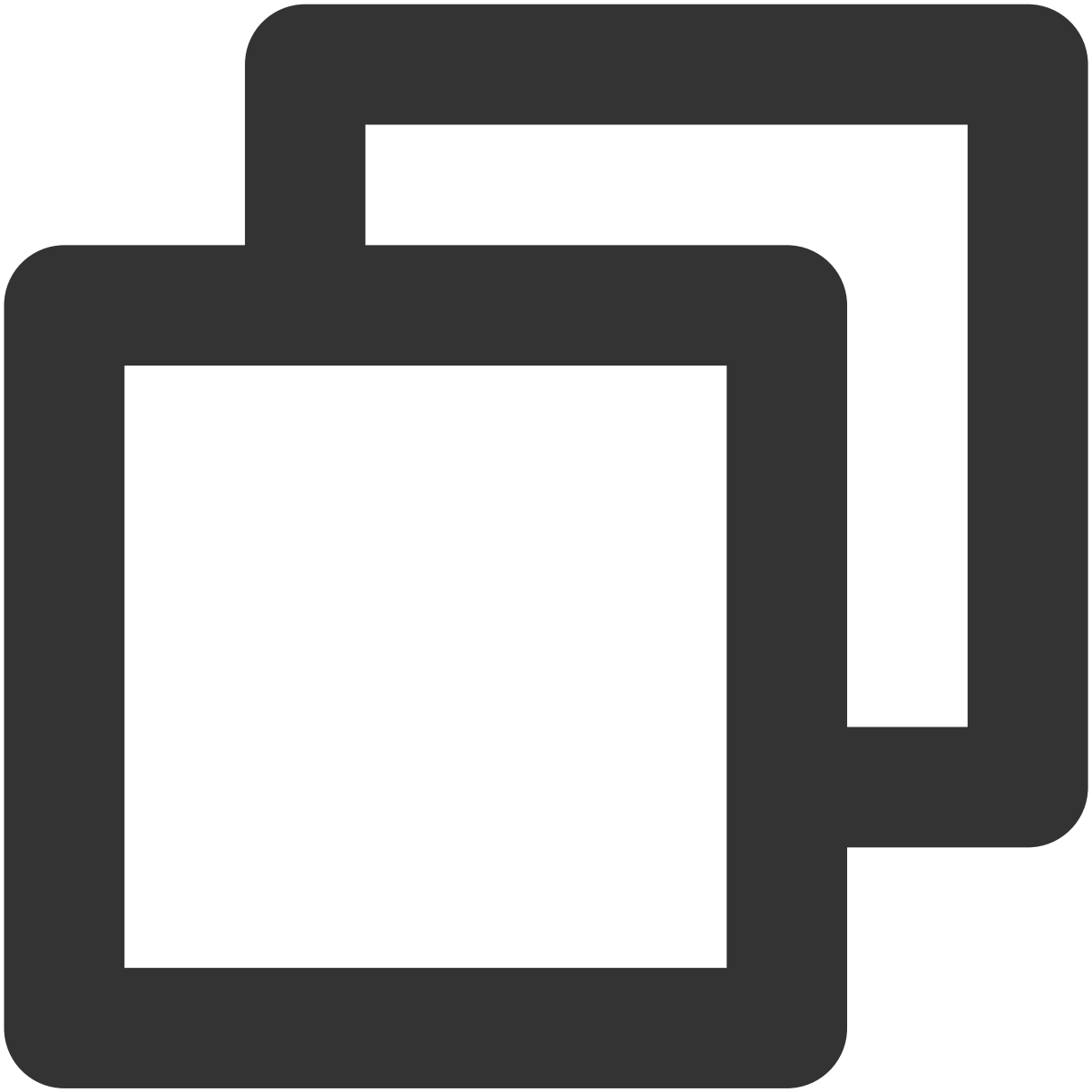
Enable Blurry Background

TUICallKit's UI design supports setting a blurry background. By calling the following interface, you can display a feature button for the blurry background on the UI. Clicking the button will directly enable the blurry background feature.

Note：

H5 is not supported yet, only PC supports it.

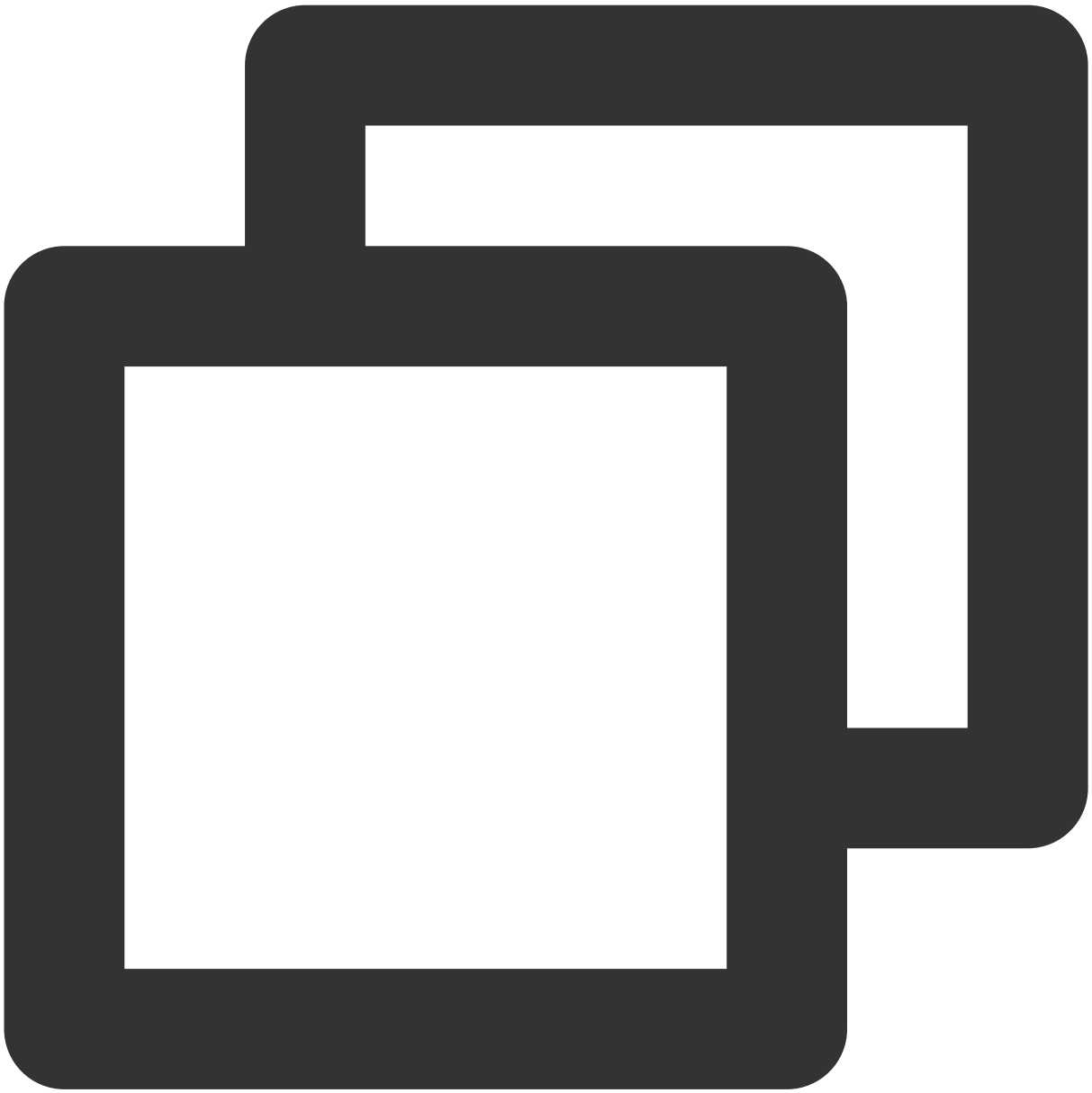
v3.2.4+ support.



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-vue"; // take @tencentcl
TUICallKitServer.enableVirtualBackground(true);
```

Setting Image Background (Optional)

Implementing the image background requires users to save the image locally. After saving, call the following interface for setting (currently, only images with a local path are supported, images of uri are not supported yet).



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-vue"; // take @tencentcl
TUICallKitServer.getTUICallEngineInstance().setVirtualBackground(imagePath: string)
```

FAQs

Enabling blurry background has no response or is delayed

Ensure you have purchased the audio and video call `Group Call` package, see Activate Service.
When the network is poor, the virtual background model file may not be completely downloaded, resulting in failure to open the virtual background.

Can the virtual background be turned on if the camera is turned off?

Not available.

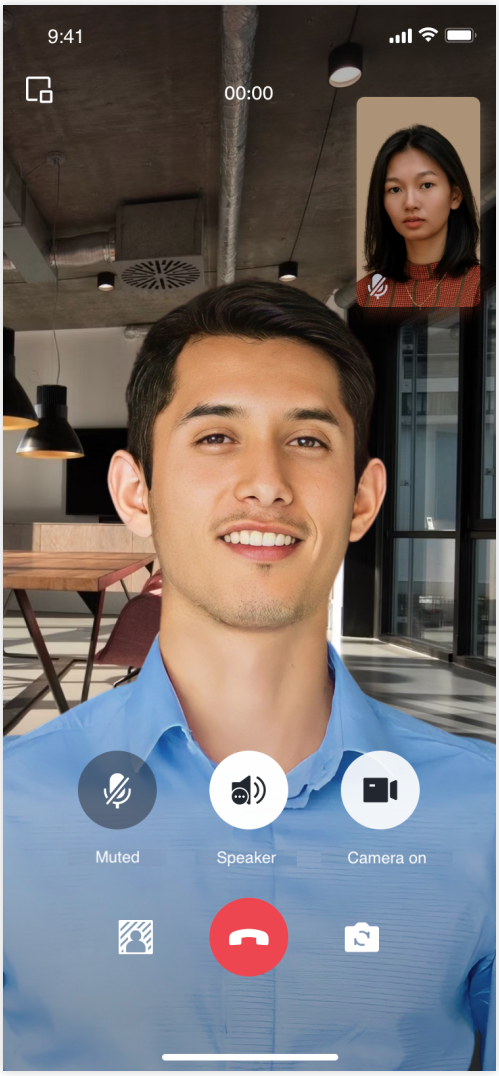
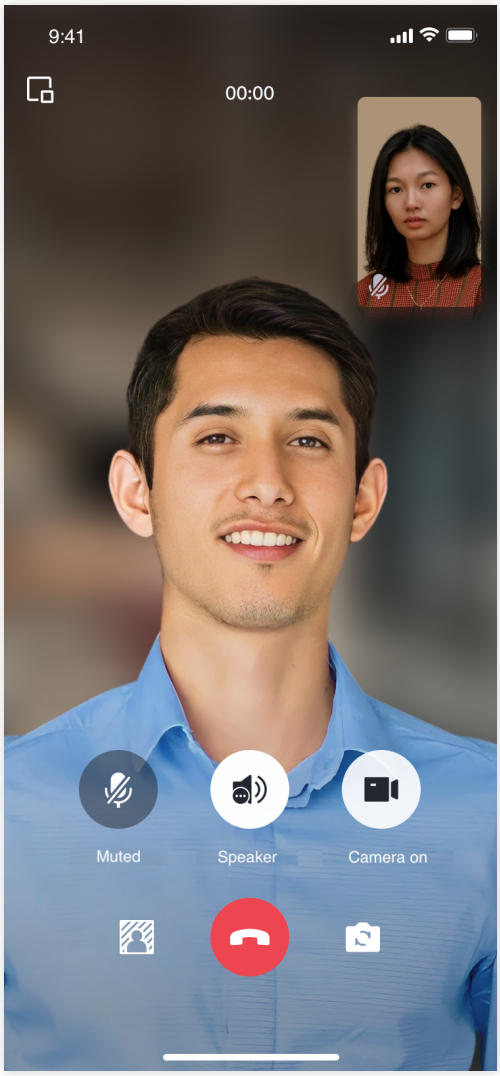
Flutter

Last updated：2024-07-22 17:00:45

TUICallKit has launched a new feature for virtual backgrounds, allowing users to set a blurry or image background during video calls. This hides the real calling environment, protects privacy, and makes the call more interesting. Next, this article will detail how to use this feature in the TUICallKit component.

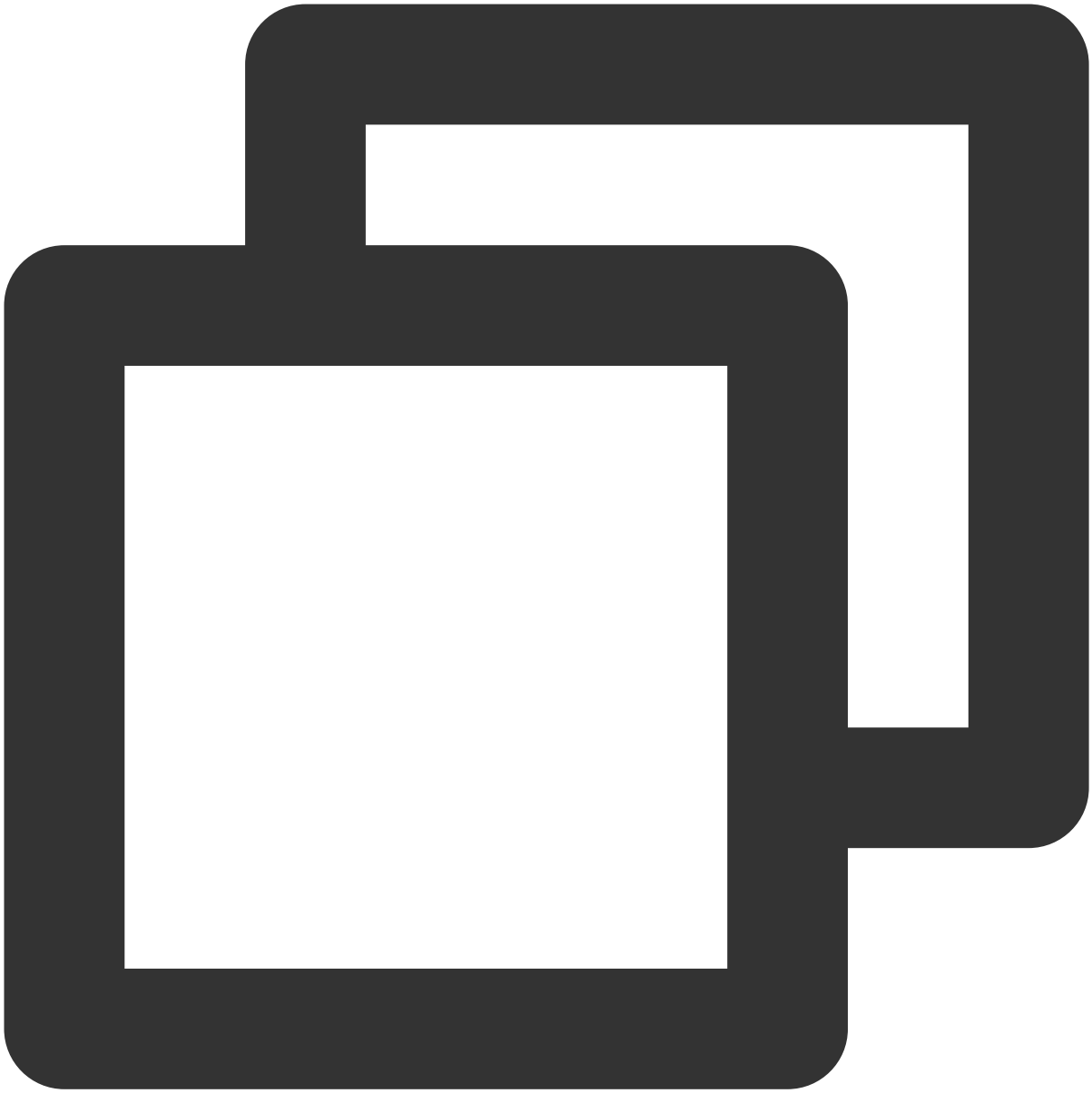
Integration effect

The display effect of the TUICallKit component after integrating the virtual background feature is as follows:

Original Camera	Blurry Background Effect
	

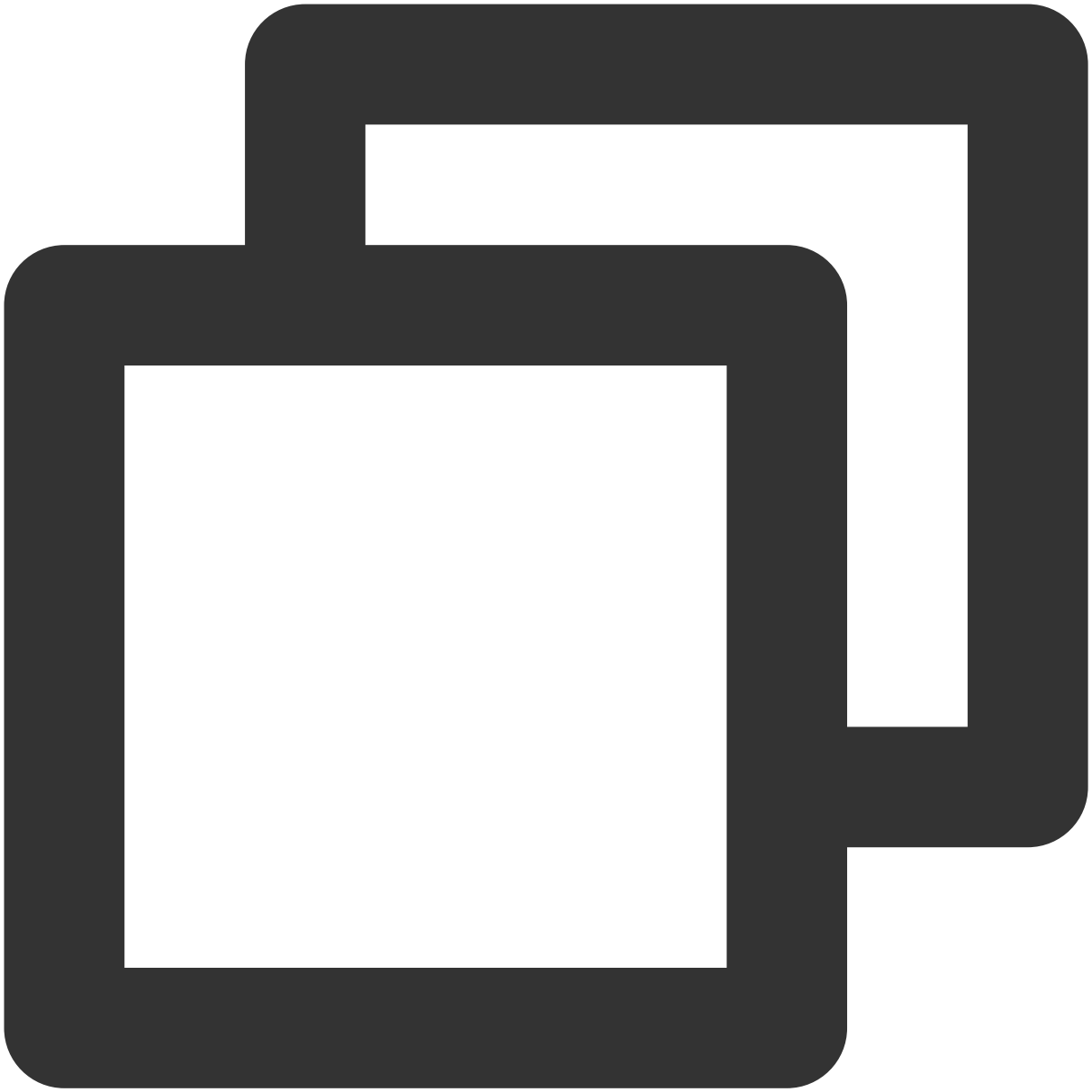
Preparation Requirements

1. Before using the Virtual Background feature provided by Tencent Cloud, you need to visit the Console to activate Audio and Video Services for your application and purchase the `Group Call` package. For specific steps, please see Activate Service.
2. Specify LiteAVSDK_Professional SDK version.
Virtual Background support starts from tencent_calls_uikit: 2.3.2 (LiteAVSDK_Professional 11.7.0.12001), different LiteAVSDK_Professional SDK versions require different model files.
Android
iOS
In the `build.gradle` file, specify the TXLiteAVSDK_Professional version, for example, set it to `11.8.0.14176`, which can be modified according to needs and version iterations.



```
api "com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176"
```

Modify the dependencies in your Podfile to specify the TXLiteAVSDK_Professional version, for example, set it to `11.8.15669` , which can be modified according to needs and version iterations.



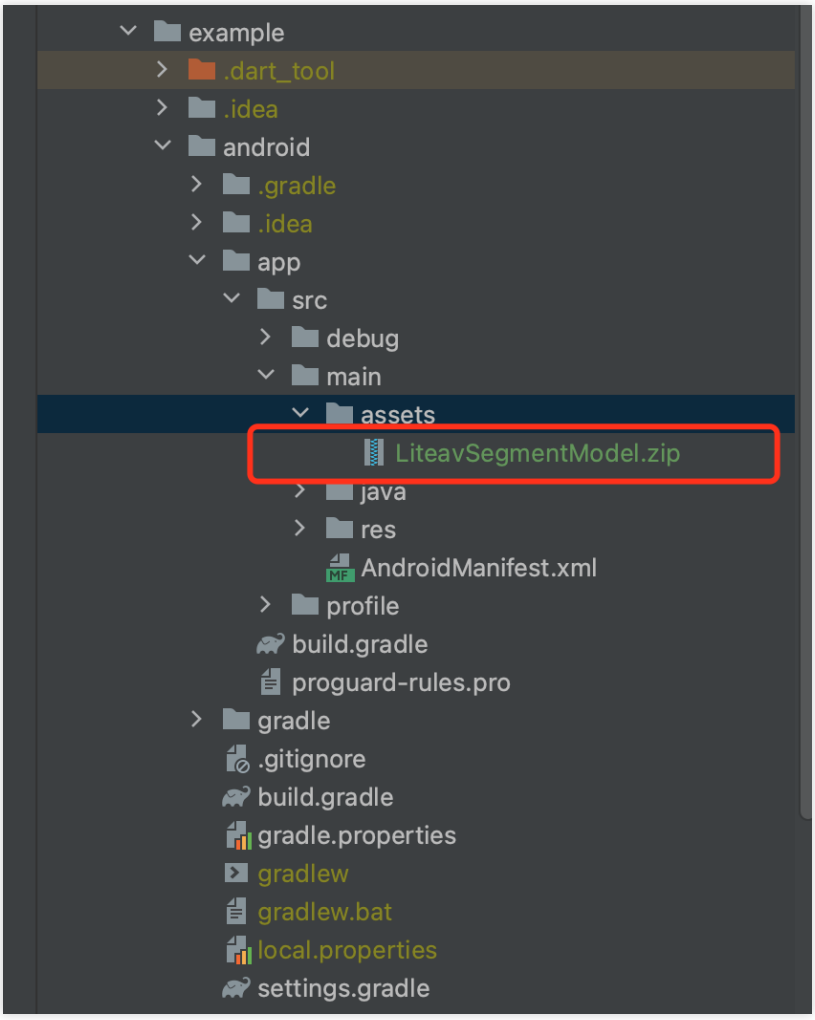
```
pod 'TXLiteAVSDK_Professional', '11.8.15669'
```

3. Download the model files according to the model file compatibility with SDK situation, and add them to the Android Studio and Xcode projects.

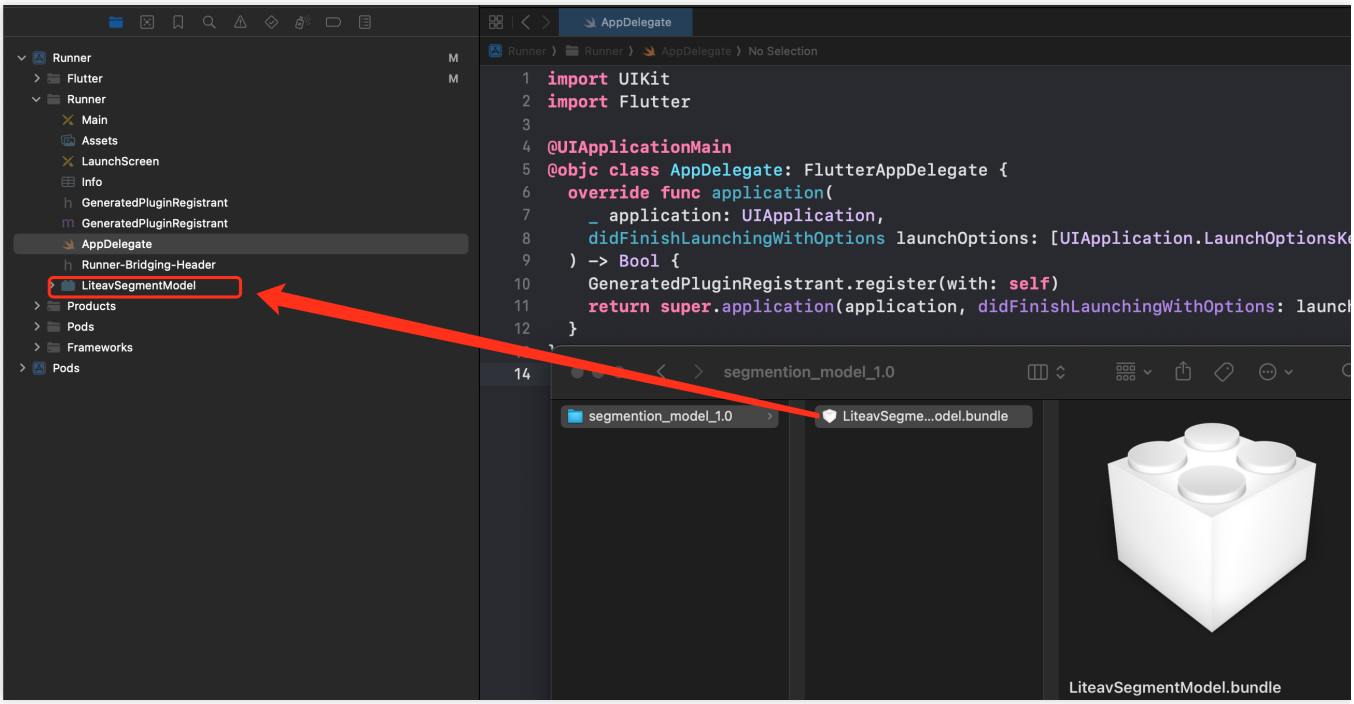
Android

iOS

After decompressing, copy the `LiteavSegmentModel.zip` file to the `assets` directory in your Android project.

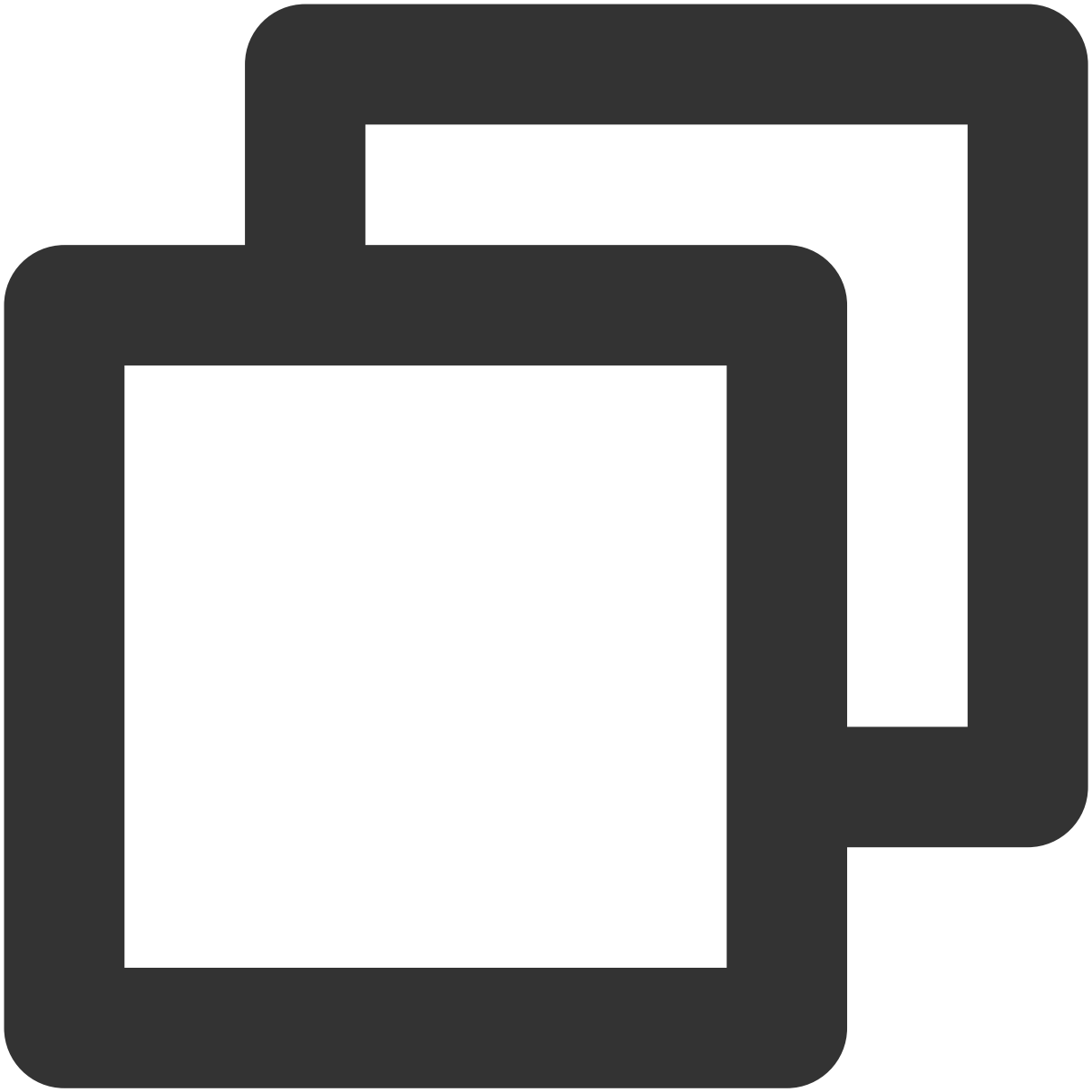


After decompression, drag and drop the `LiteavSegmentModel.bundle` file into your Xcode project.



Enable Blurry Background

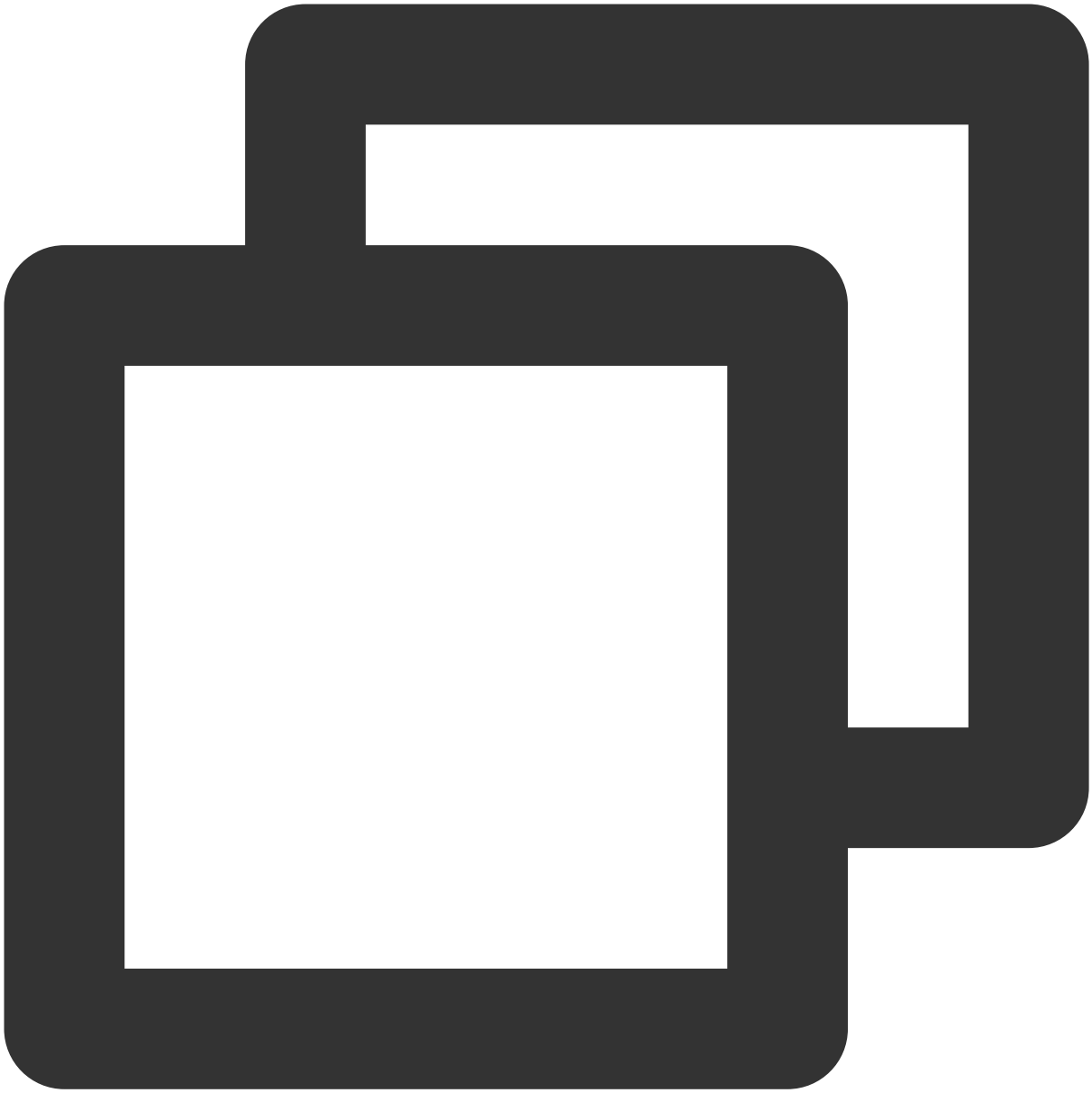
TUICallKit's UI design supports setting a blurry background. By calling the following interface, you can display a feature button for the blurry background on the UI. Clicking the button will directly enable the blurry background feature.



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

Setting Image Background (Optional)

The implementation of a picture background needs to be done by the user. Add the picture file to the Flutter project (you need to add resources in the pubspec.yaml file) and call the interface to set the background picture (currently, only local path pictures are supported, network pictures are not supported yet).



```
TUICallEngine.instance.setVirtualBackground("***.png", (code, message) { });
```

FAQs

Blurry background not responding or delayed?

Ensure you have purchased the `Group Call` package, see Activate Service for more details.

Ensure the model file is downloaded to local.

If a model file is not added to the local path, when enabling the blurry background feature, the SDK will then download the model file. Under normal network conditions, the download takes 1~3s; the poorer the network, the longer it will take.
Check if the model file and SDK are a match.

How to match the model file with the SDK?

TUICallKit is a video and audio call scenario implemented based on the Chat SDK and TRTC SDK. The virtual background is a distinctive feature provided by TRTC SDK. It's important to note that the virtual background model file needs to match the version of the TRTC SDK; otherwise, the blurry background feature may not function properly. The table below lists the relationships between the model files and the TRTC SDK versions:

SDK Version	Virtual background model file Download address
com.tencent.liteav:LiteAVSDK_Professional:11.7.0.12001	segmention_1.0
com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176	segmention_2.0

On-Cloud Recording (TUICallKit)

Last updated：2024-06-25 14:16:05

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

Note:

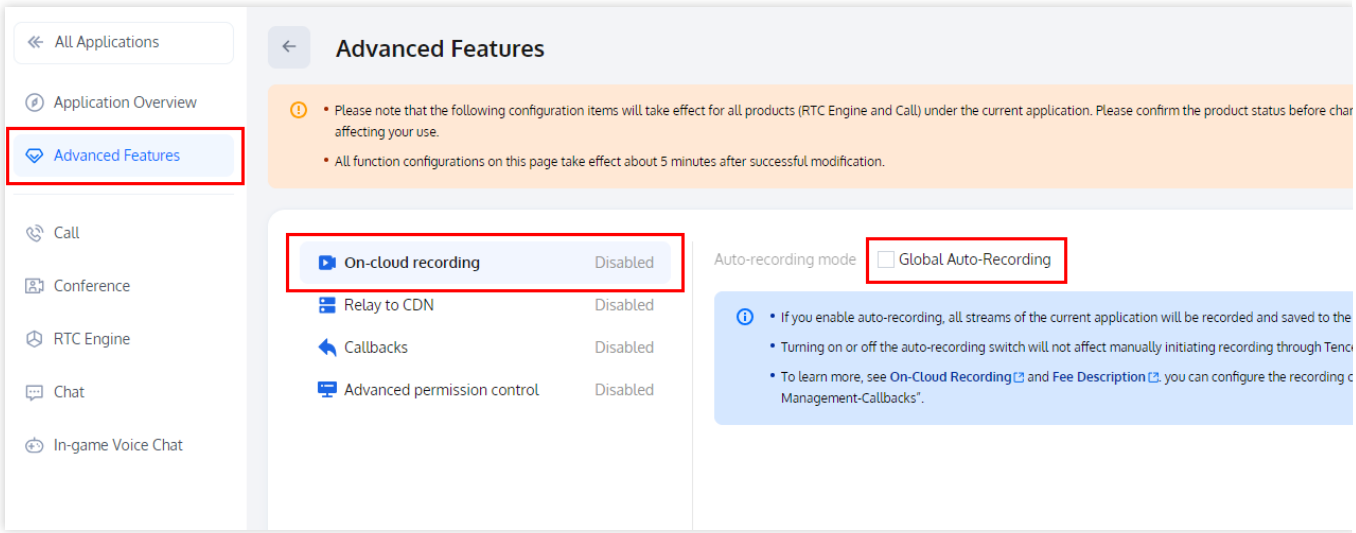
`TUICallKit` integrates multiple basic Tencent Cloud PaaS services, and its audio/video capabilities rely on TRTC. To use the on-cloud recording feature of `TUICallKit`, you need to configure it in the TRTC console .

Scheme 1. Automatic Recording (Recommended)

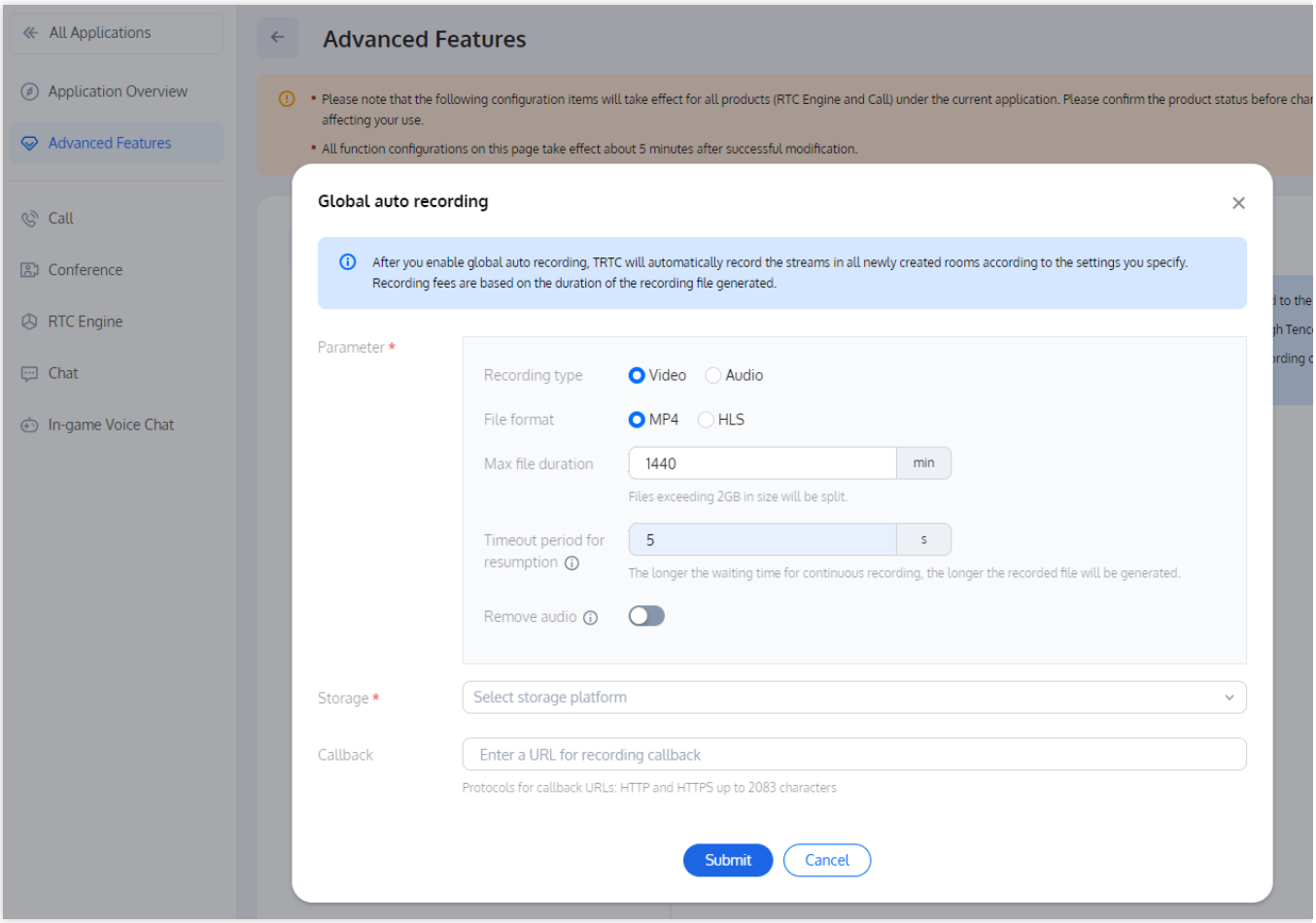
We recommend you use the automatic recording scheme. In this scheme, recording is not manually started or stopped by you. Instead, recording tasks are managed on the TRTC backend, and a call will be recorded automatically when there is an audio/video stream being upstreamed. You can integrate it quickly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

1. Find the application of the target `SDKAppId` in **Application Management** in the TRTC console and enter the **Advanced Feature** page.

2. In the **Advanced Function** configuration page, you can see the option for **On-cloud recording** configuration. Click on **Global Auto-Recording** to enter the configuration popup window.



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



Note:

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

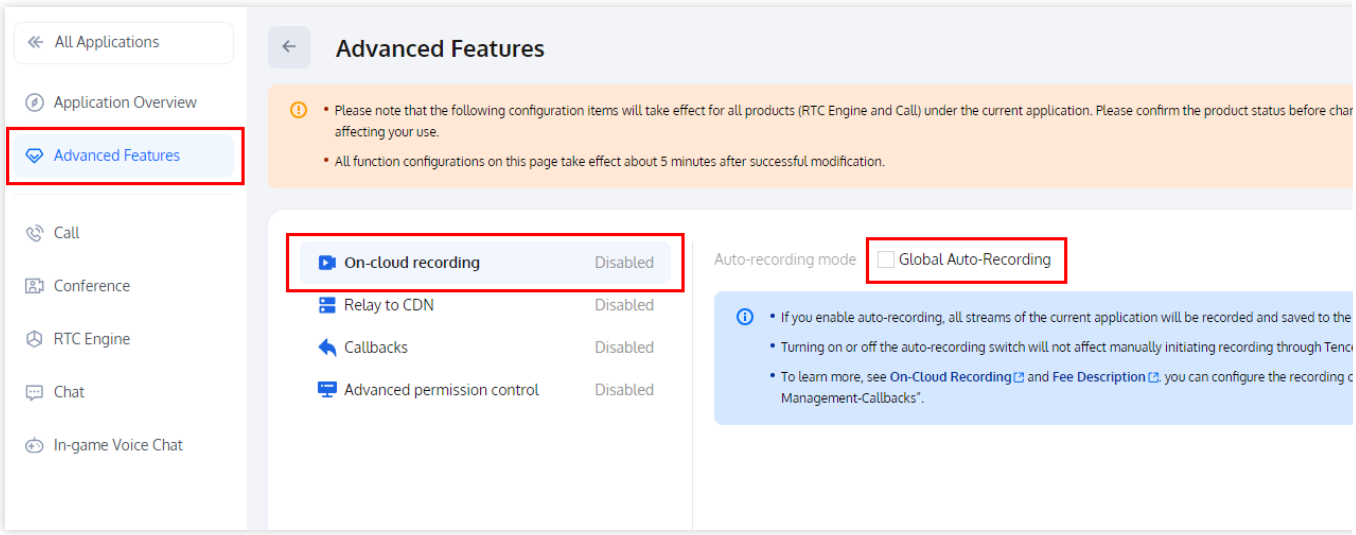
After the global automatic recording feature is enabled, when a call is answered and there is audio/video being upstreamed, a recording task will be triggered automatically, and recording will stop automatically when the call ends. If a user leaves the room due to poor network conditions or other exceptions,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

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

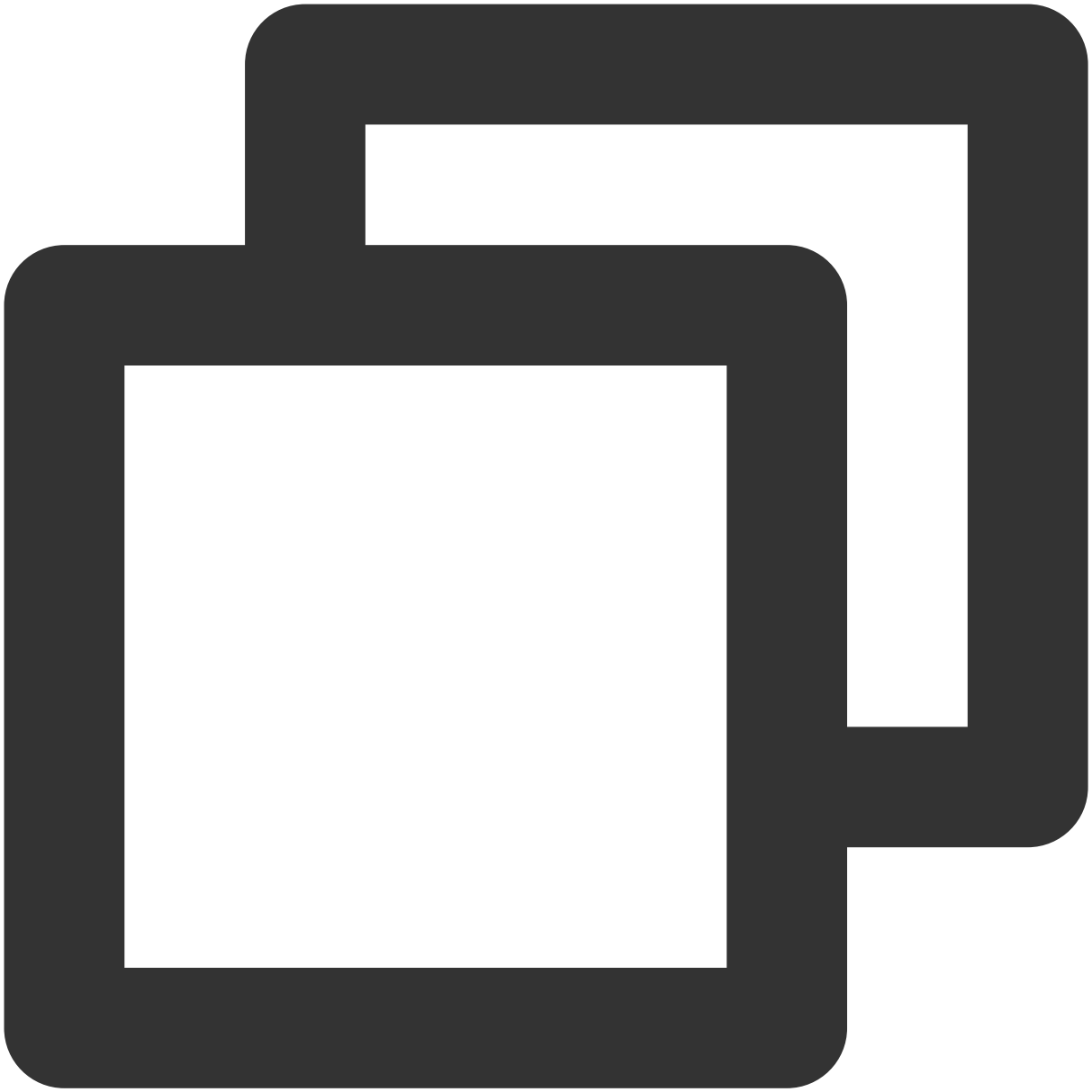
Scheme 2. RESTful API-Based Recording

If the automatic recording scheme doesn't meet your needs, you can also use the more flexible RESTful API-based recording scheme. In this scheme, you can record and subscribe to a specified anchor in the room, customize the layout of mixed streams, and update the layout and subscription during recording. However, **using its features requires using it together with the business backend service and performing complex integration operations**:

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



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



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

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

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

FAQs

1. How do I view the detailed recording durations?

You can view the detailed recording durations on the On-Cloud Recording page in the TRTC console.

2. How do I view recorded files?

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

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

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

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

Additional Features（TUICallKit）Configuring Nicknames and Avatars (All Platform)

Last updated：2024-05-24 18:35:12

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

Setting Avatar, Nickname

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

Android（Kotlin）

Android（Java）

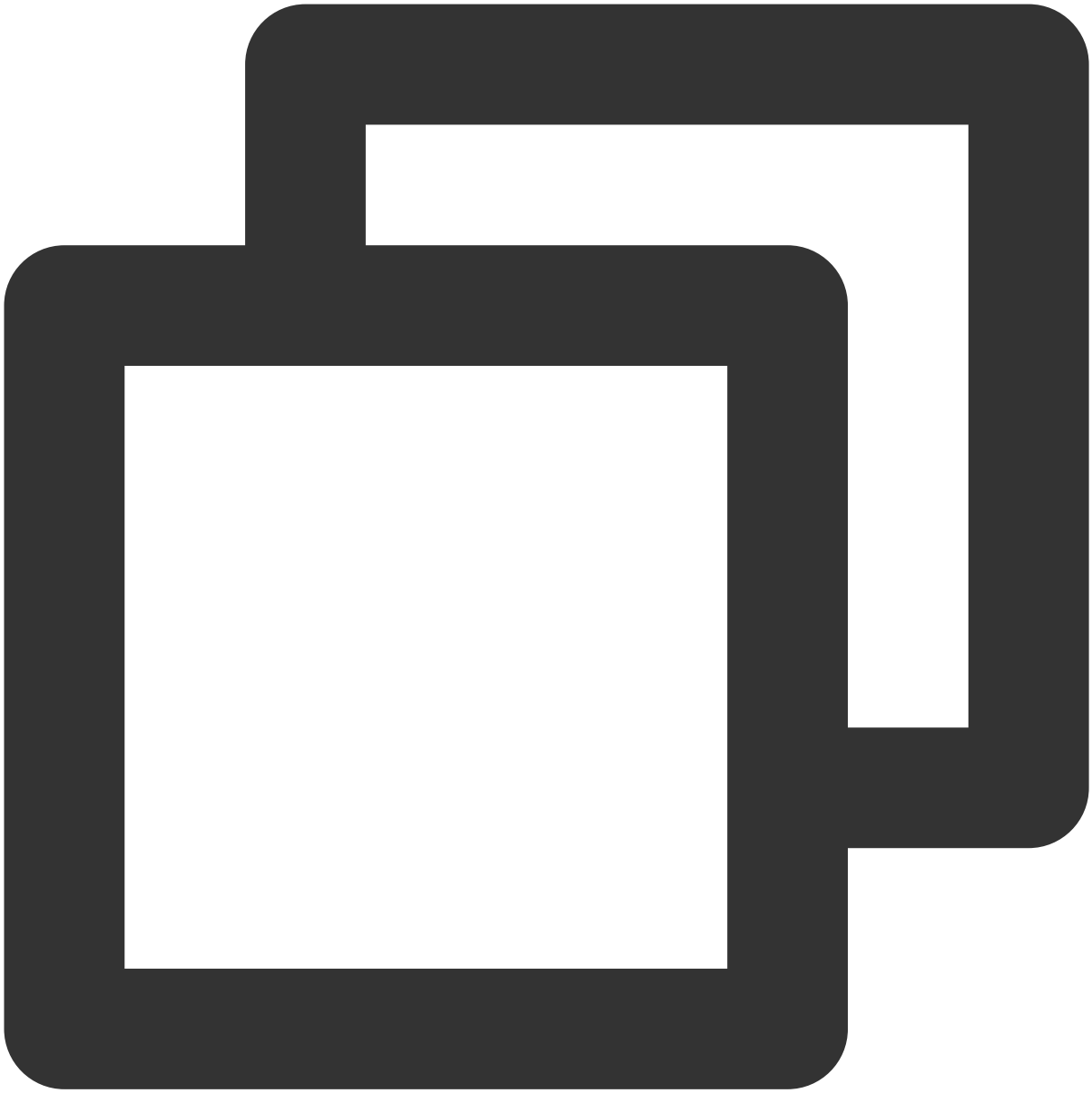
iOS(Swift)

iOS(Objective-C)

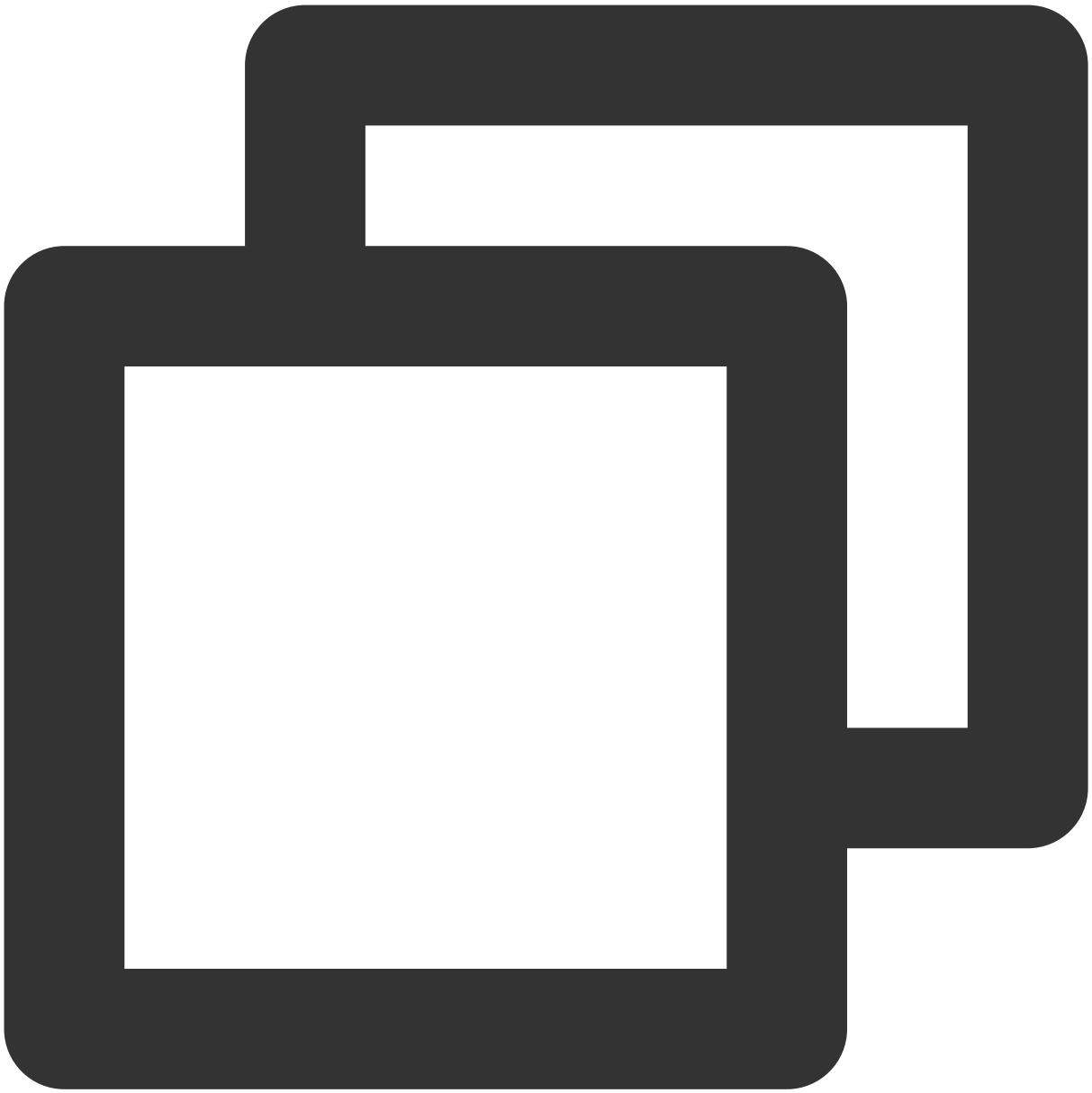
Flutter（Dart）

Web&H5

uni-app(Andorid&iOS)

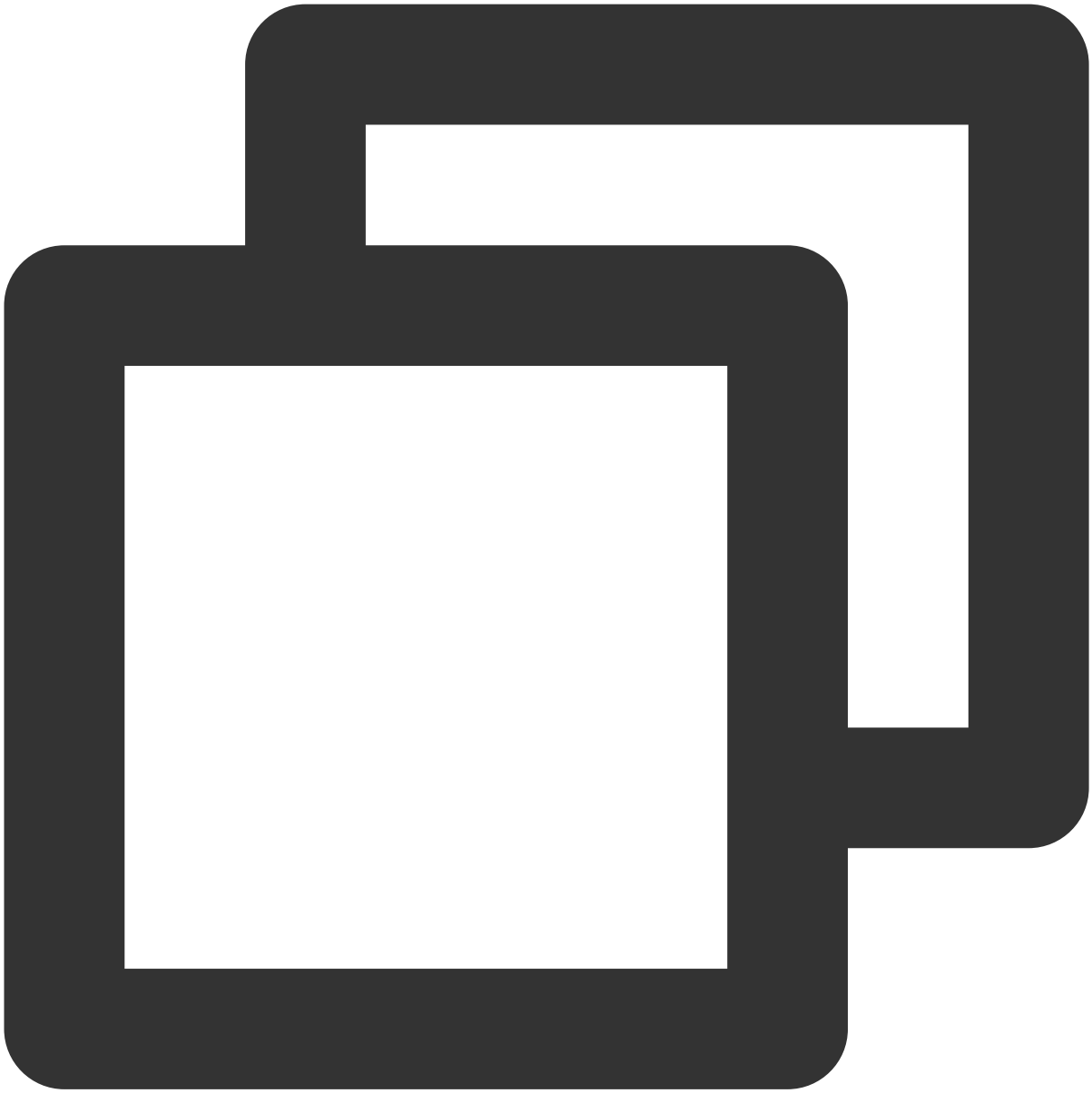


```
import com.tencent.qcloud.tuikit.TUICommonDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

TUICallKit.createInstance(context).setSelfInfo("jack", "https:/****/user_avatar.png
    object : TUICommonDefine.Callback {
        override fun onSuccess() {
        }

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

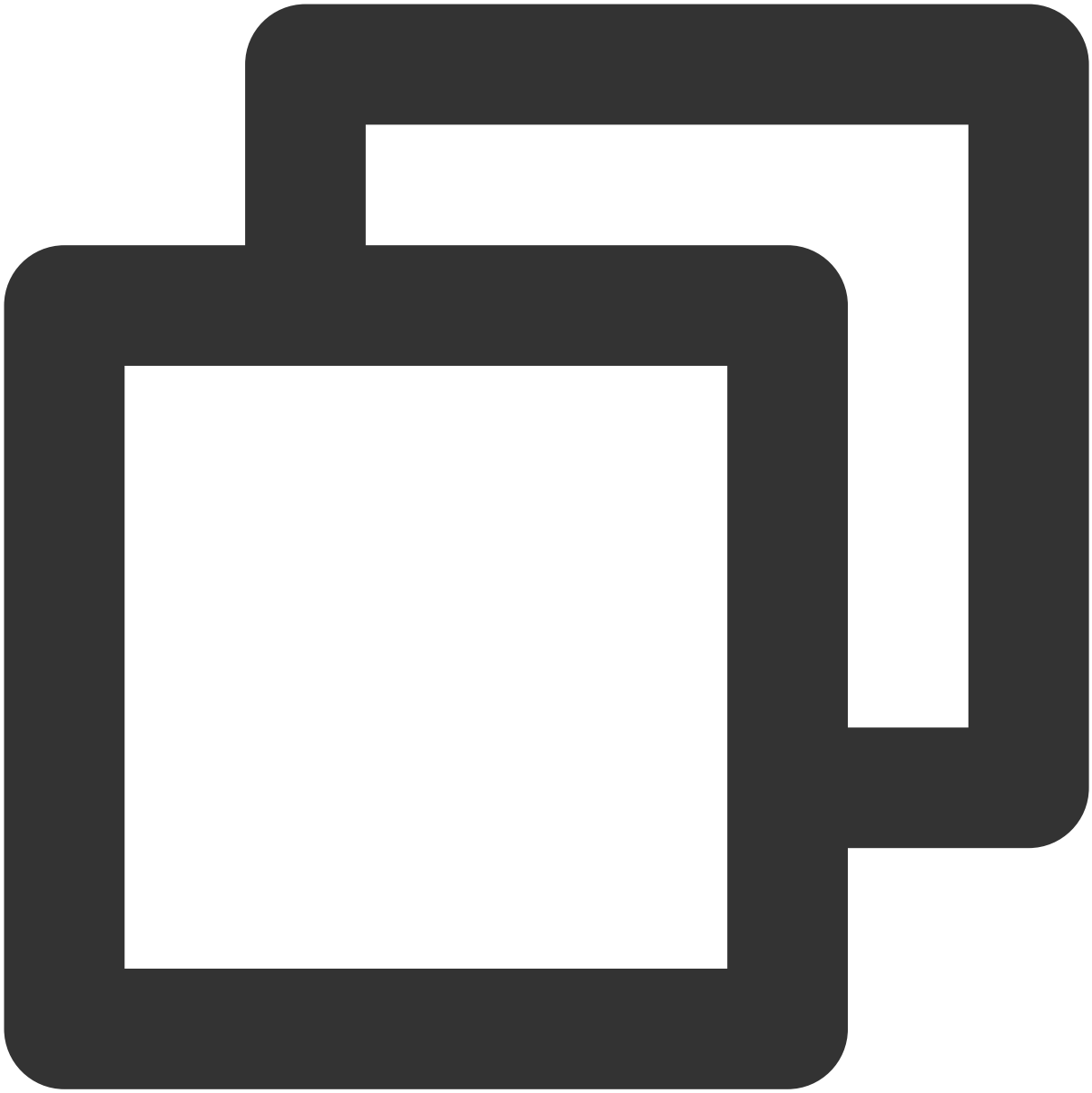


```
import com.tencent.qcloud.tuikit.TUICommonDefine;
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit;

TUICallKit.createInstance(context).setSelfInfo("jack", "https:/****/user_avatar.png
    @Override
    public void onSuccess() {
    }

    @Override
    public void onError(int errorCode, String errorMessage) {
```

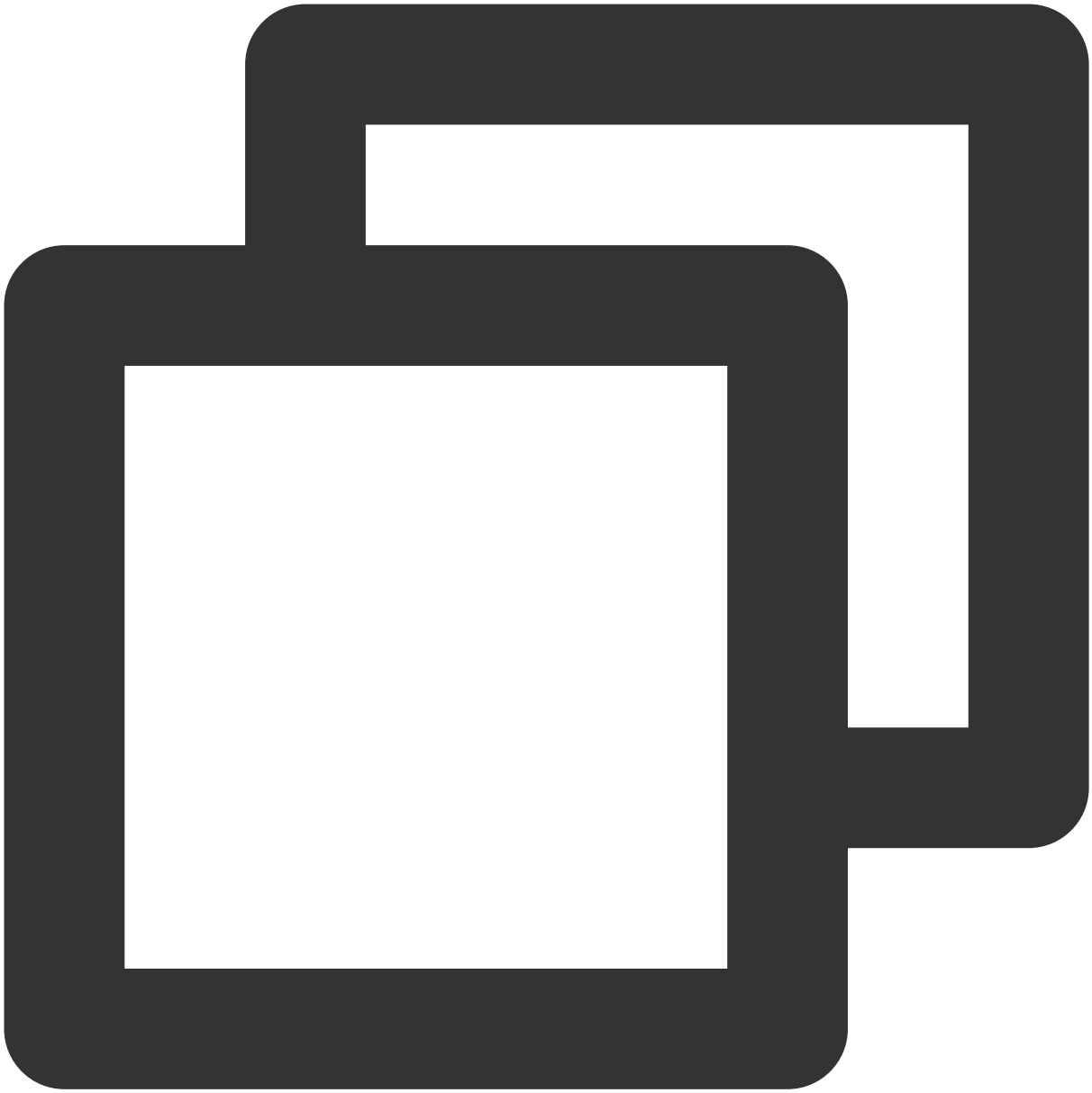
```
    }
});
```



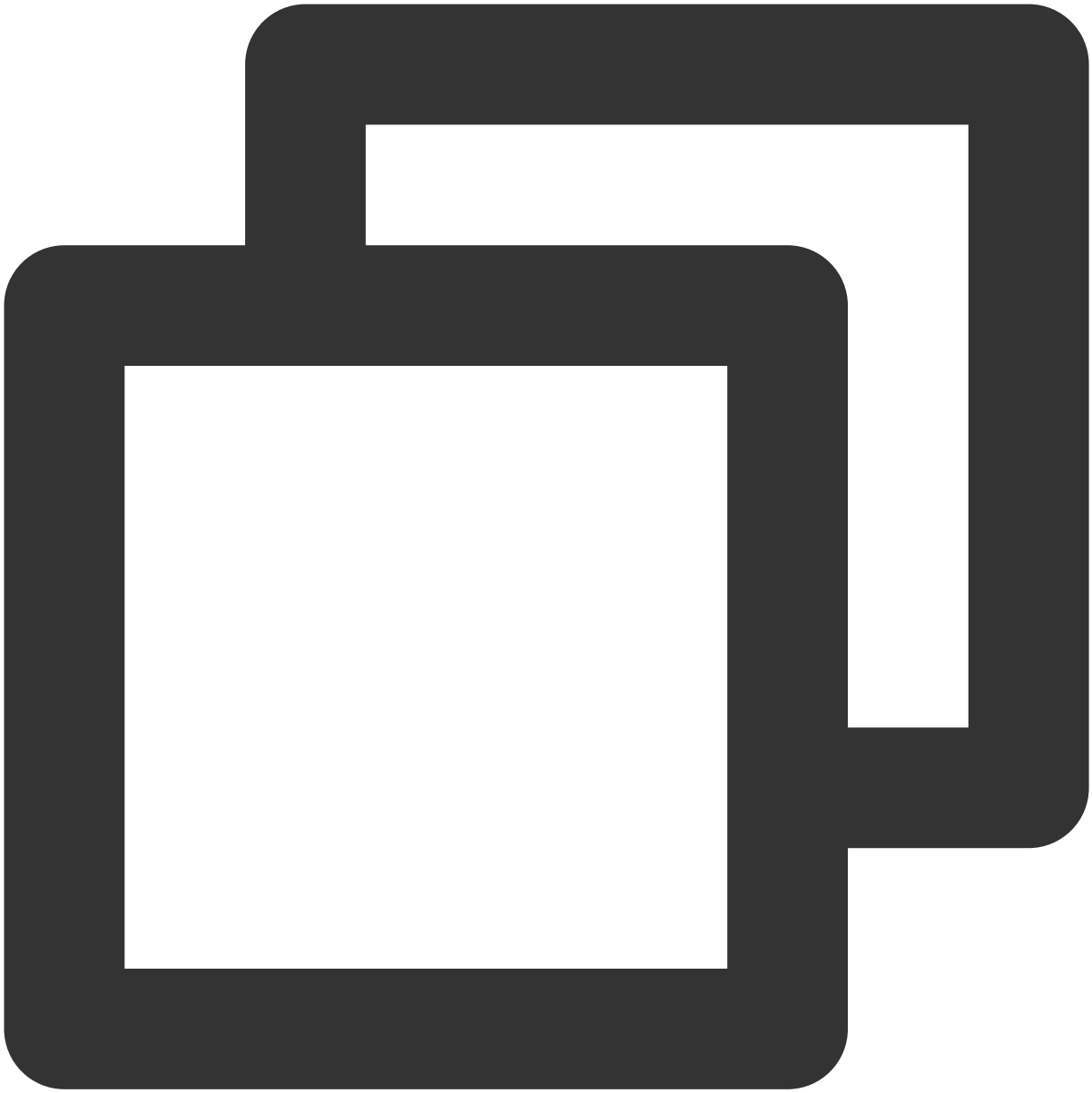
```
import TUICallKit_Swift
import TUICallEngine

TUICallKit.createInstance().setSelfInfo(nickname: "", avatar: "") {

} fail: { code, message in

}
```



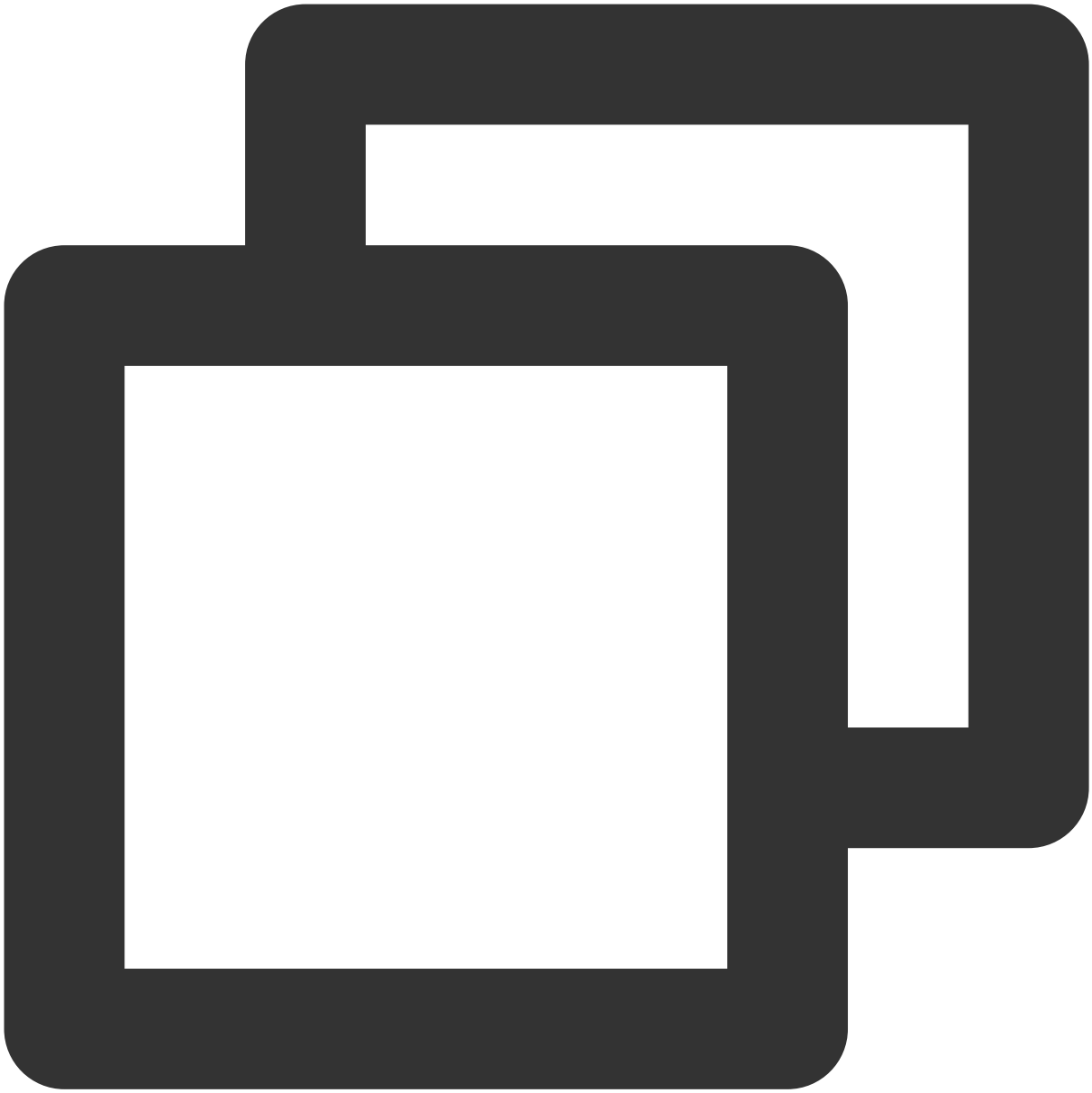
```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

[[TUICallKit createInstance] setSelfInfoWithNickname:@"" avatar:@"" succ:^{

} fail:^(int code, NSString * _Nullable errMsg) {

}];
```



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

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



```
import { TUICallKitServer } from '@tencentcloud/call-uikit-vue';
// import { TUICallKitServer } from '@tencentcloud/call-uikit-react';
try {
  await TUICallKitServer.setSelfInfo({ nickName: "jack", avatar: "http://xxx" });
} catch (error) {
  console.error(`[TUICallKit] Failed to call the setSelfInfo API. Reason: ${error}`
}
```



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

Note

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

Configure Resolution and Fill Mode (Web)

Last updated：2024-04-03 17:23:11

Introduction to how to set resolution, fill pattern.

Setting resolution, fill pattern

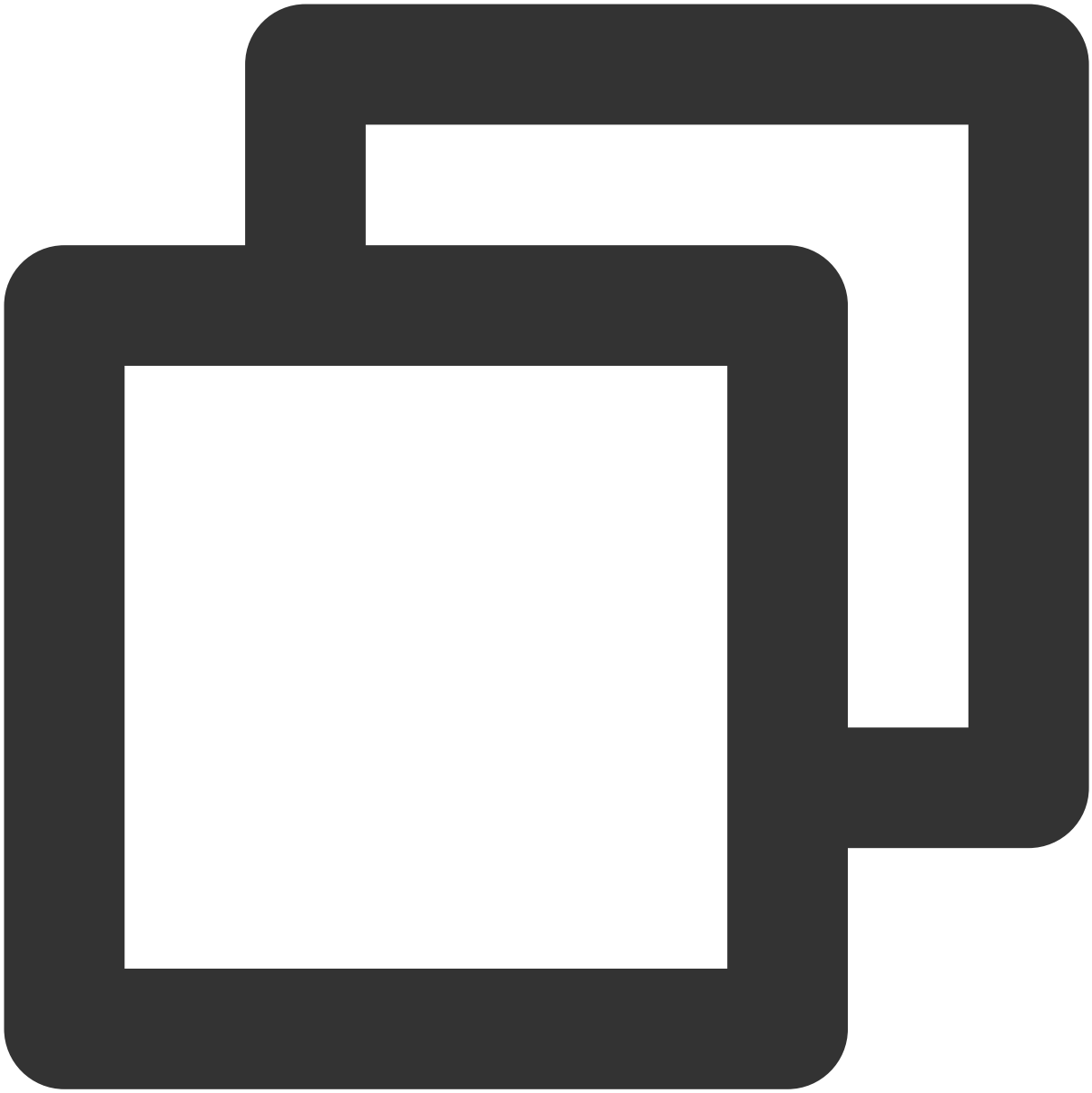
The TUICallKit component provides numerous feature switches, allowing for selective enabling or disabling as needed. For specifics, refer to Introduction to TUICallKit Attributes .

`VideoDisplayMode` : Display mode of the frame.

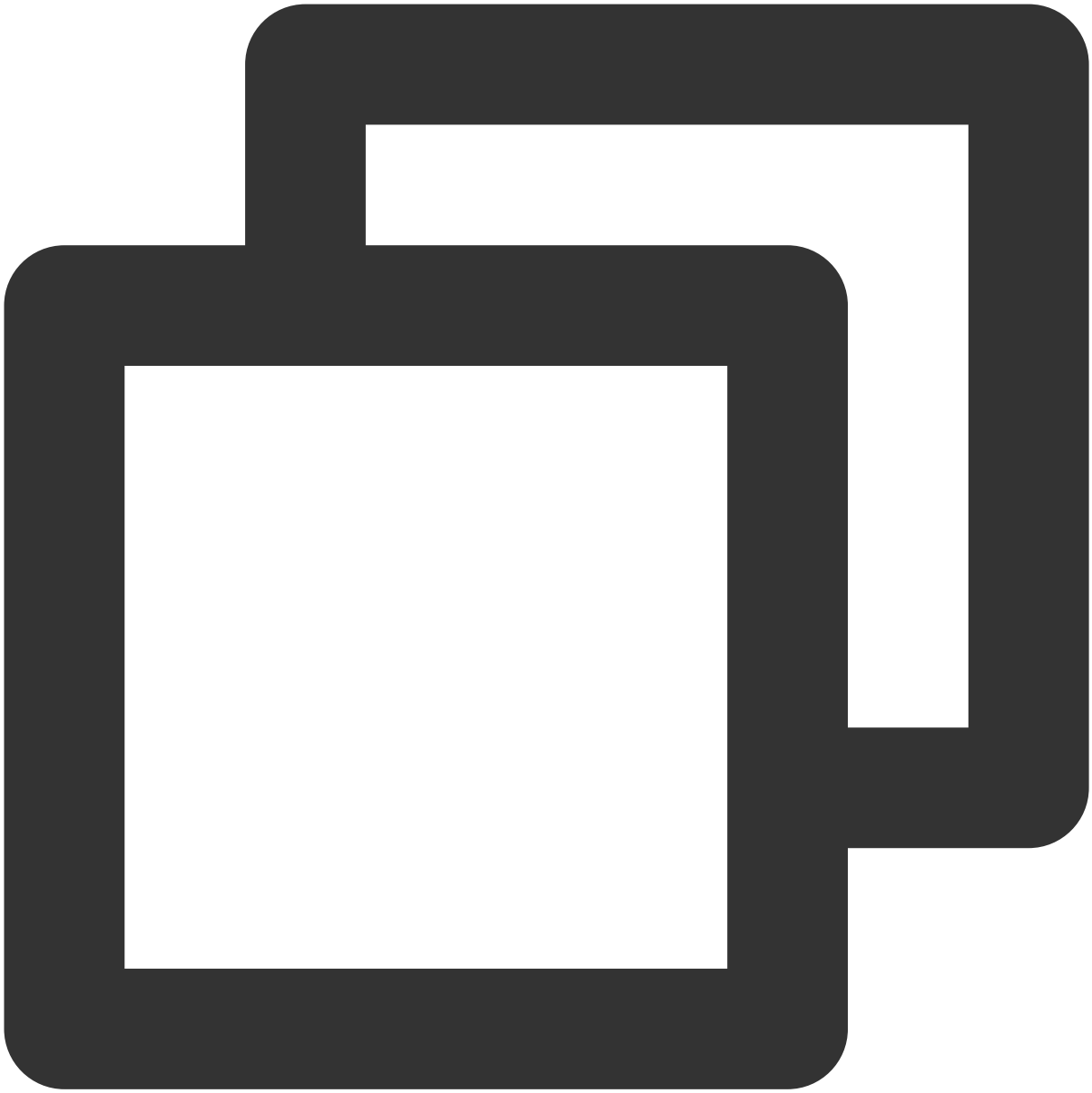
`VideoResolution` : Call resolution.

React

Vue



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

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



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

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

videoDisplayMode

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

`VideoDisplayMode.CONTAIN`

`VideoDisplayMode.COVER`

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

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

videoResolution

The resolution `videoResolution` has three possible values:

`VideoResolution.RESOLUTION_480P`

`VideoResolution.RESOLUTION_720P`

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

Resolution Explanation:

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

Frequently Asked Questions:

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

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

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

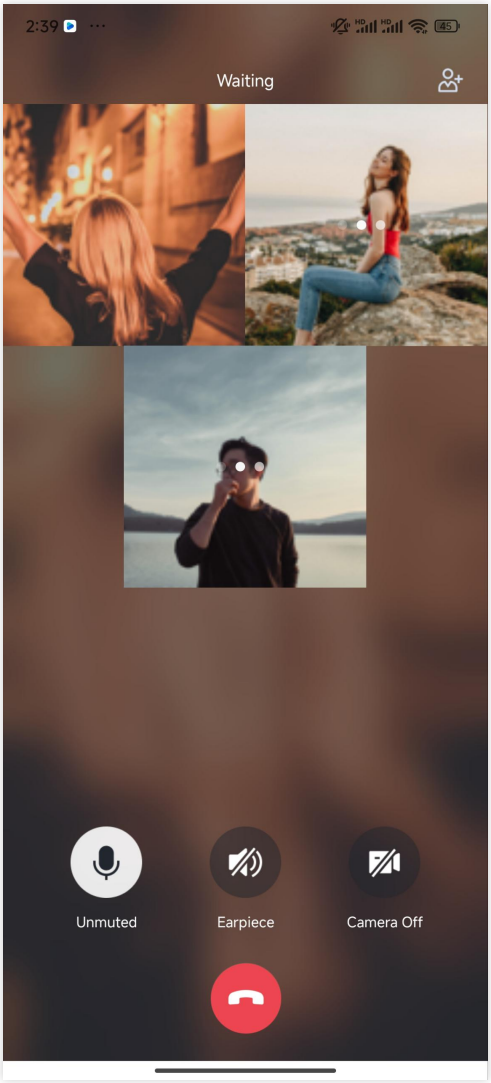
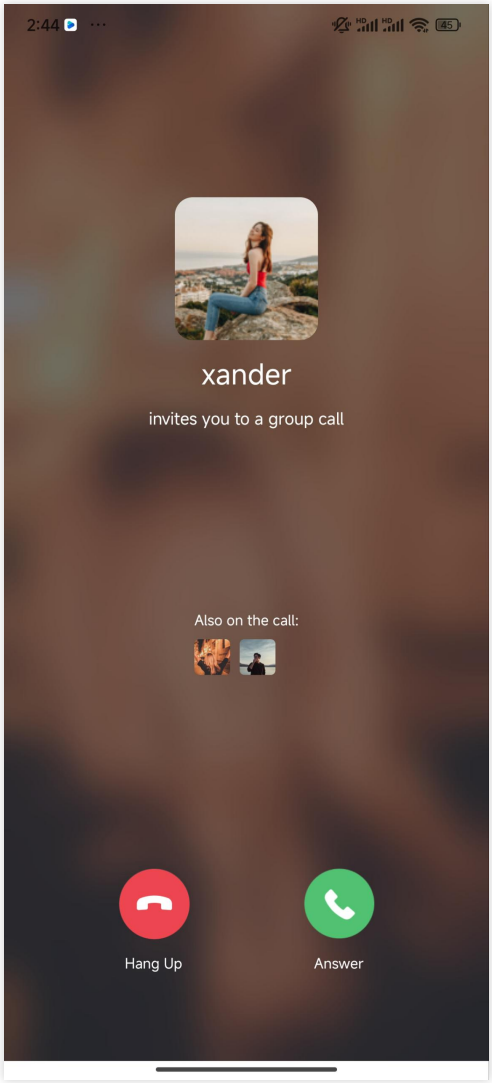
Group Calls Android&iOS&Flutter

Last updated：2024-06-17 17:41:52

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

Expected outcome

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

Initiate a group call	Receive a group call request	
		

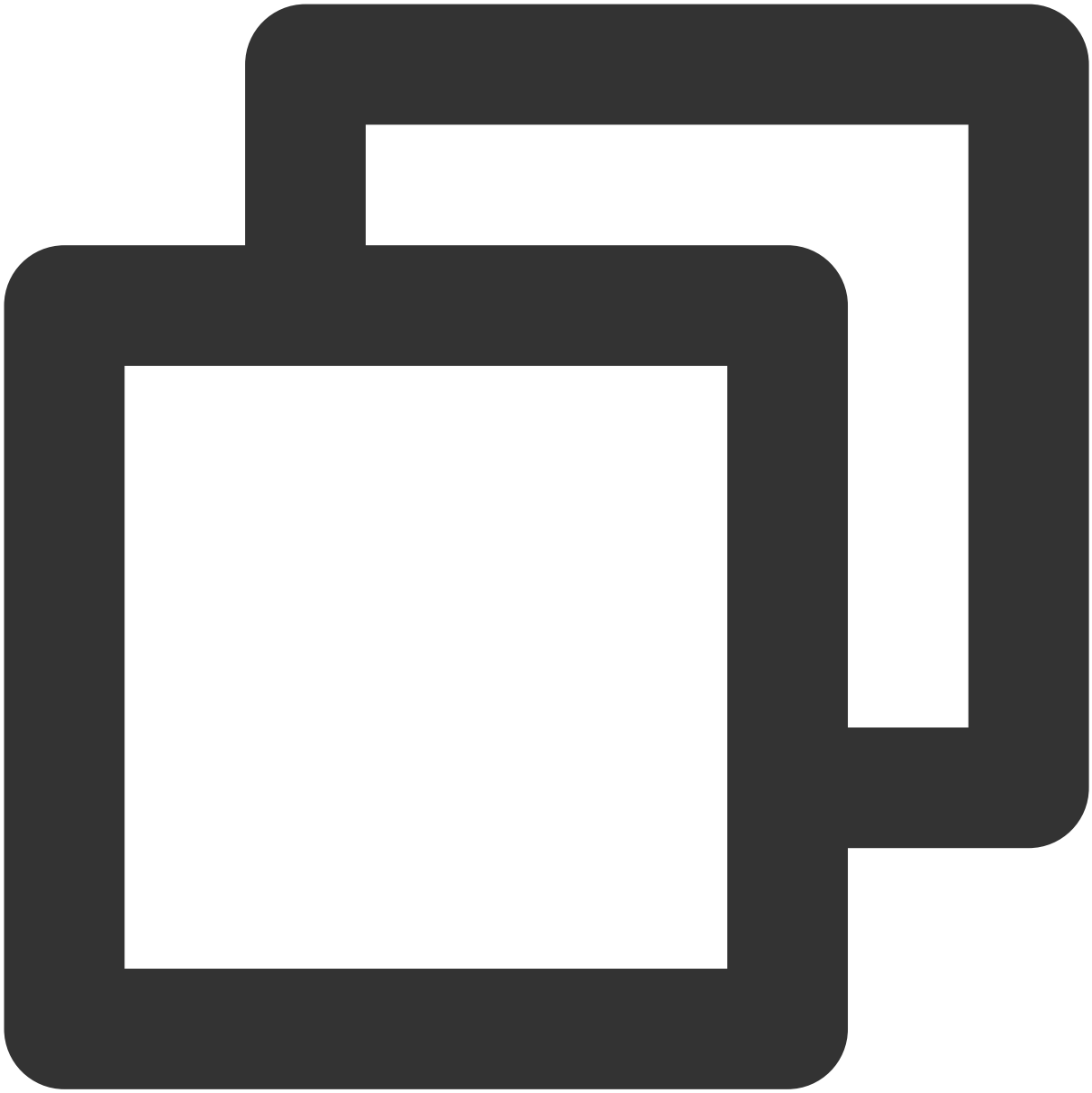
Create groupID

Before using the group call feature, you need to create a group first and initiate a group call in an existing group.
Method one: Create a group by calling the Chat API, see Chat Group Management.
Method two: Create a group manually through the console, see Console group management.

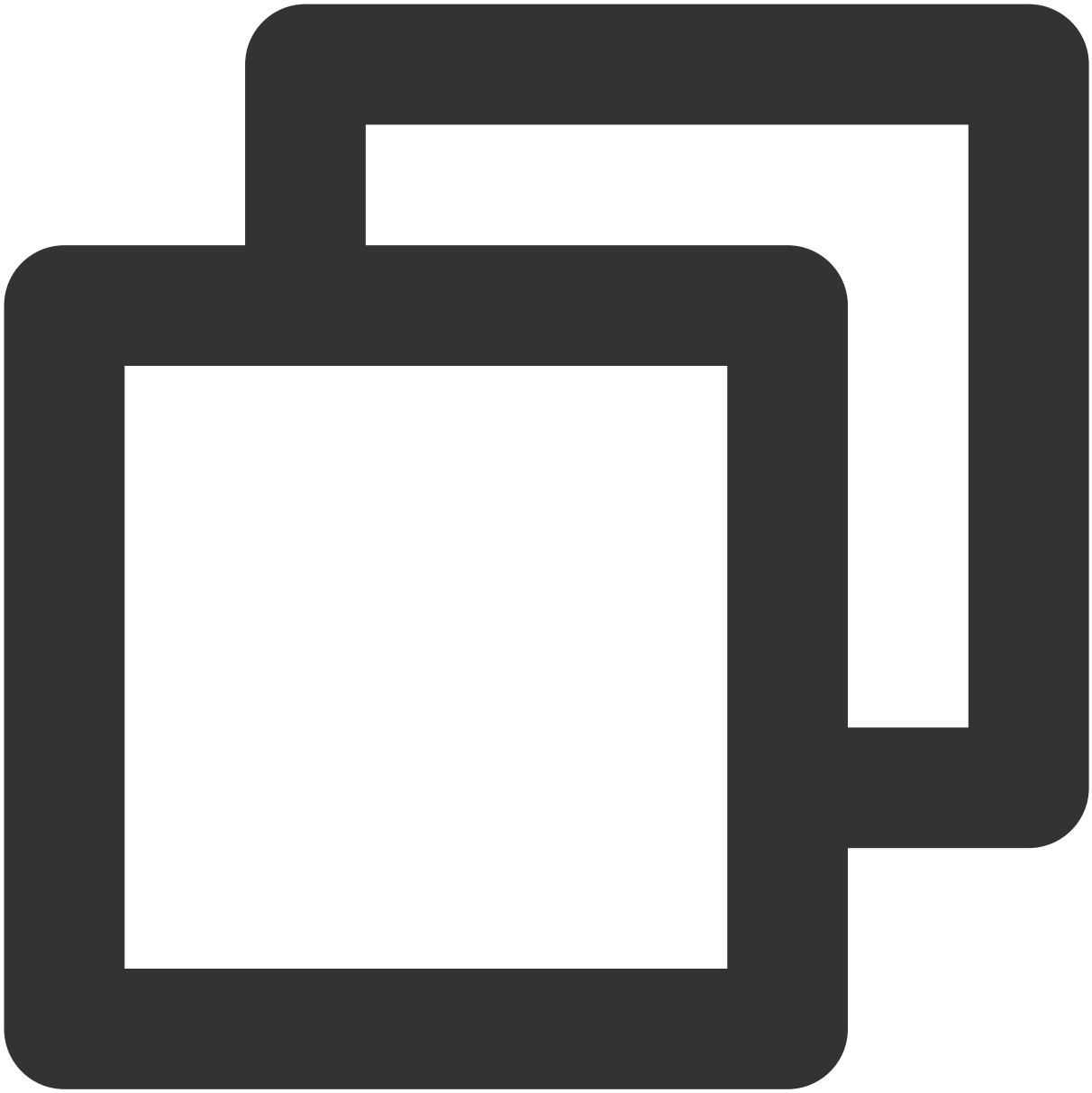
Group call

Initiate a group call

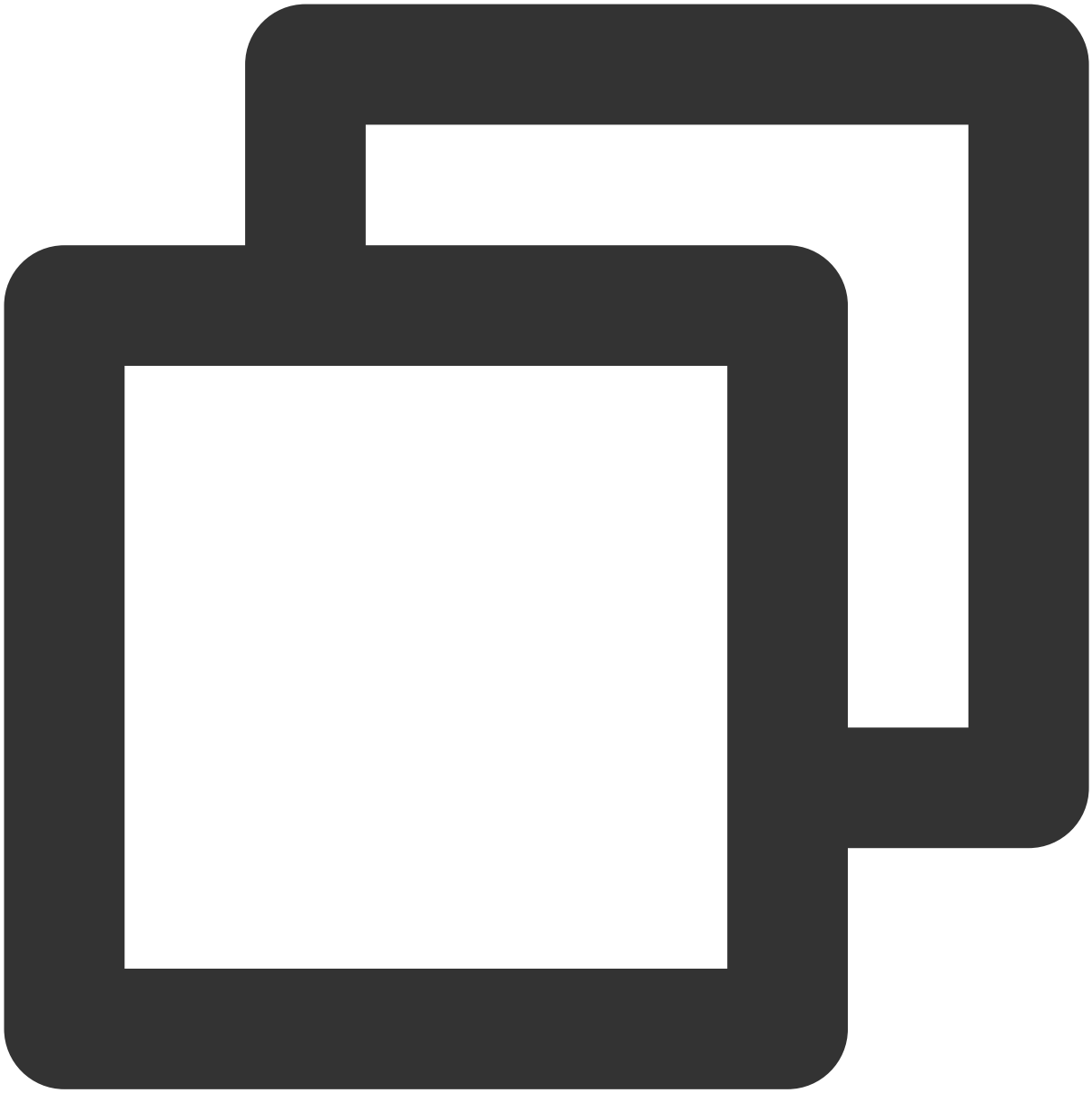
Launch a group call using the groupCall API.
Android(Kotlin)
Android(Java)
iOS(Swift)
iOS(Objective-C)
Flutter(Dart)



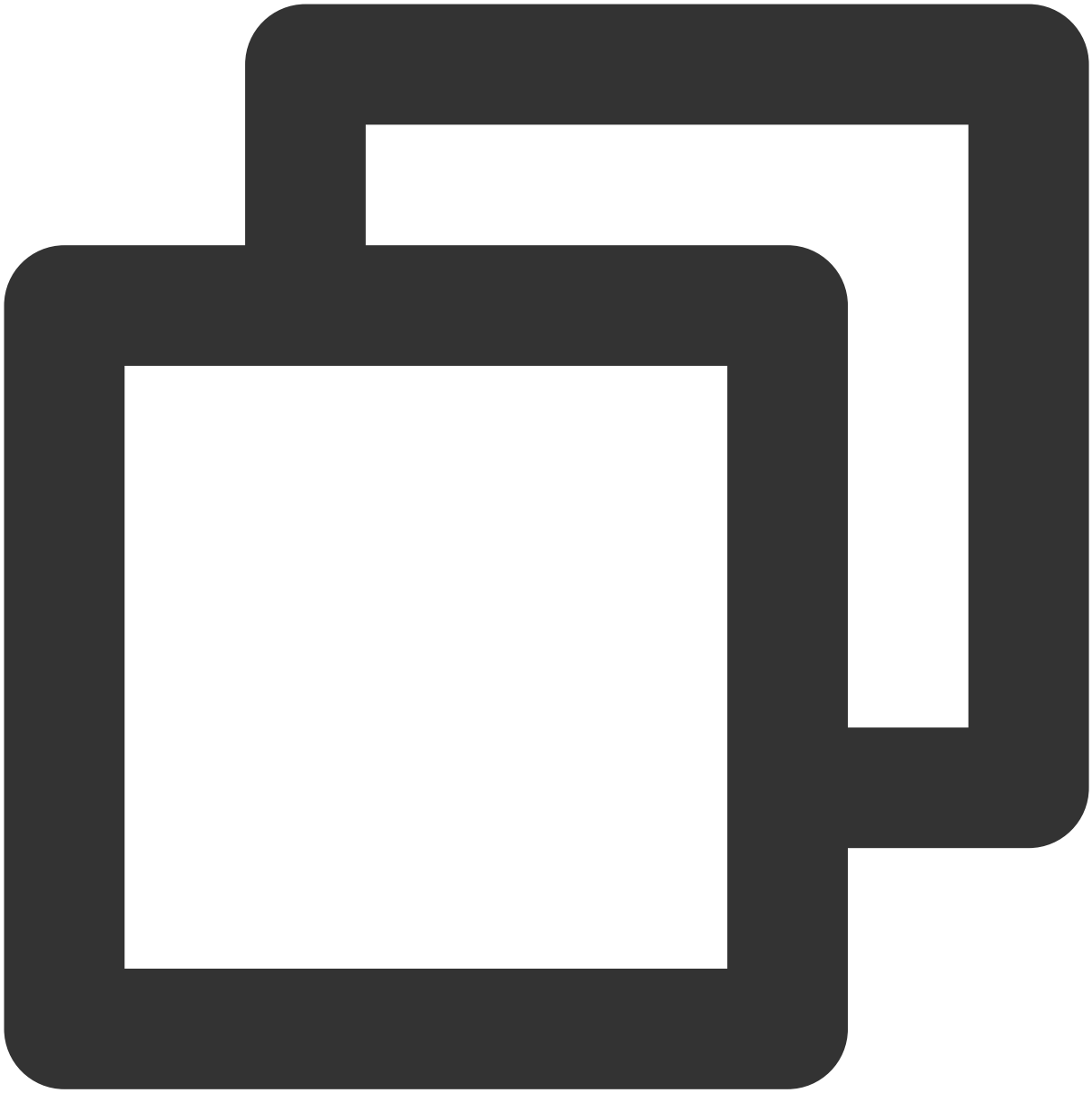
```
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

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



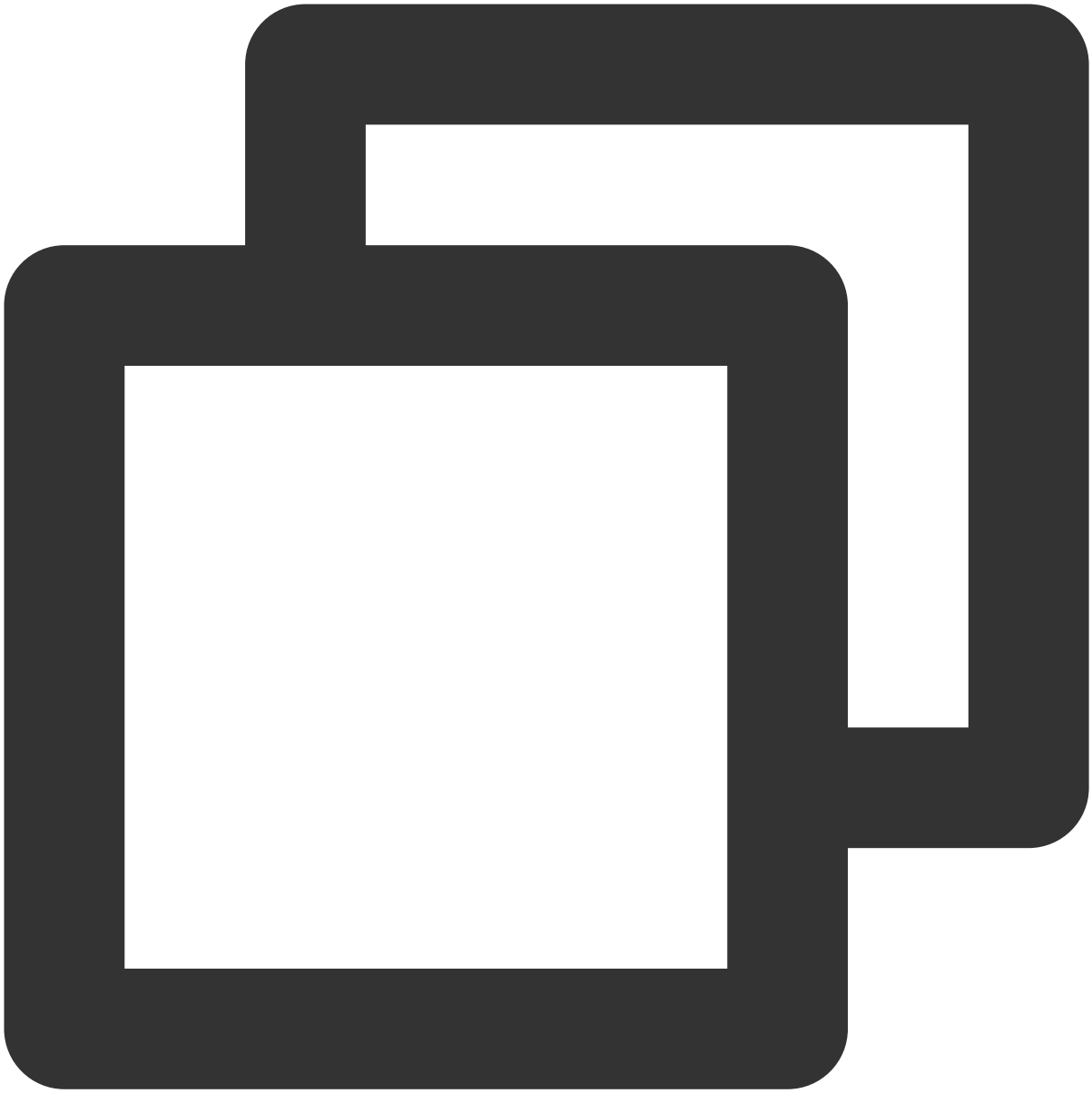
```
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine;
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit;

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



```
import TUICallKit_Swift
import TUICallEngine

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



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <TUICallEngine/TUICallEngine.h>

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



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

void groupCall() {
    TUICallKit.instance.groupCall('0001', ['denny', 'mike', 'tommy'], TUICallMediaT
}
```

Join a group call

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

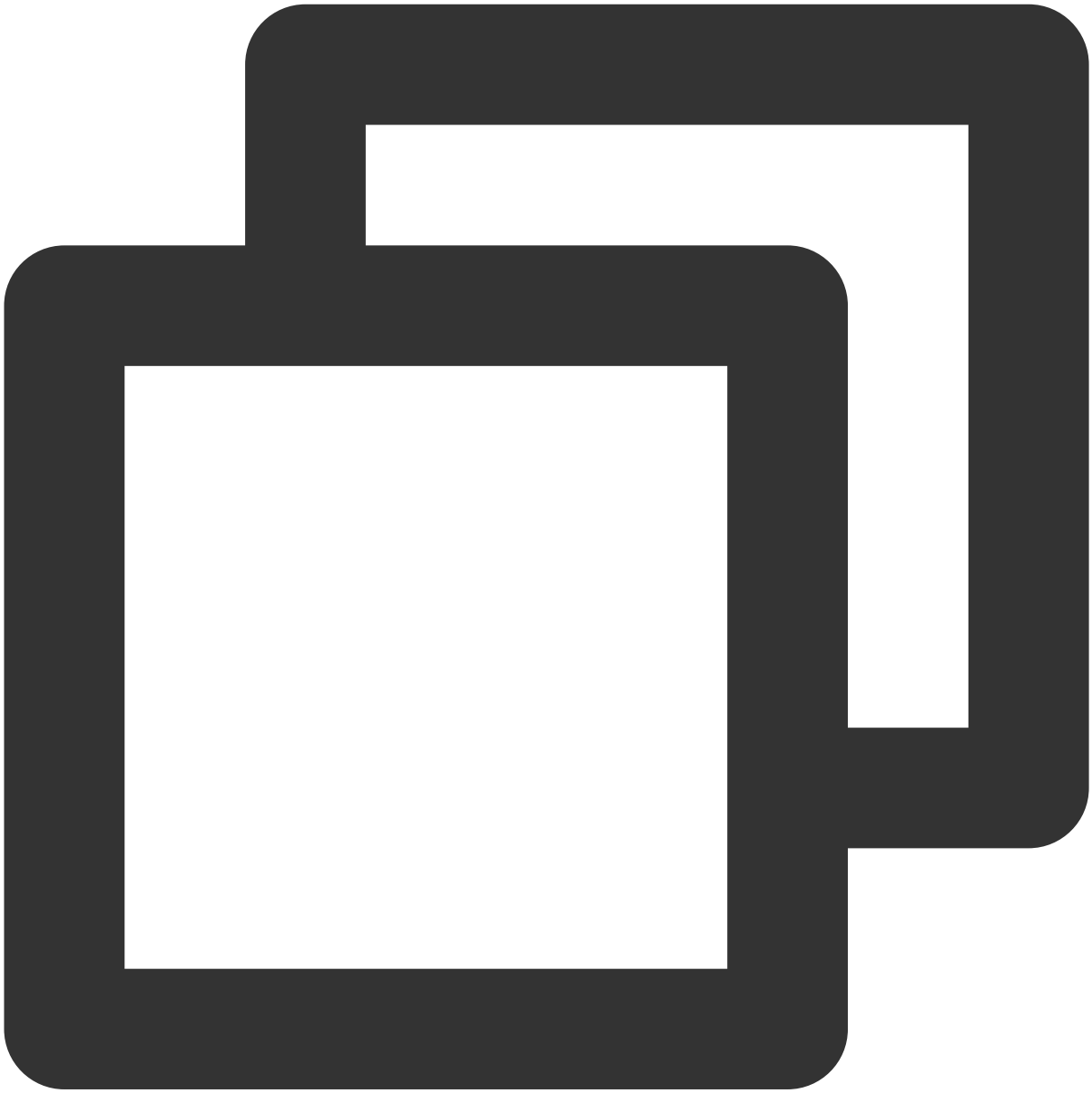
Android(Kotlin)

Android(Java)

iOS(Swift)

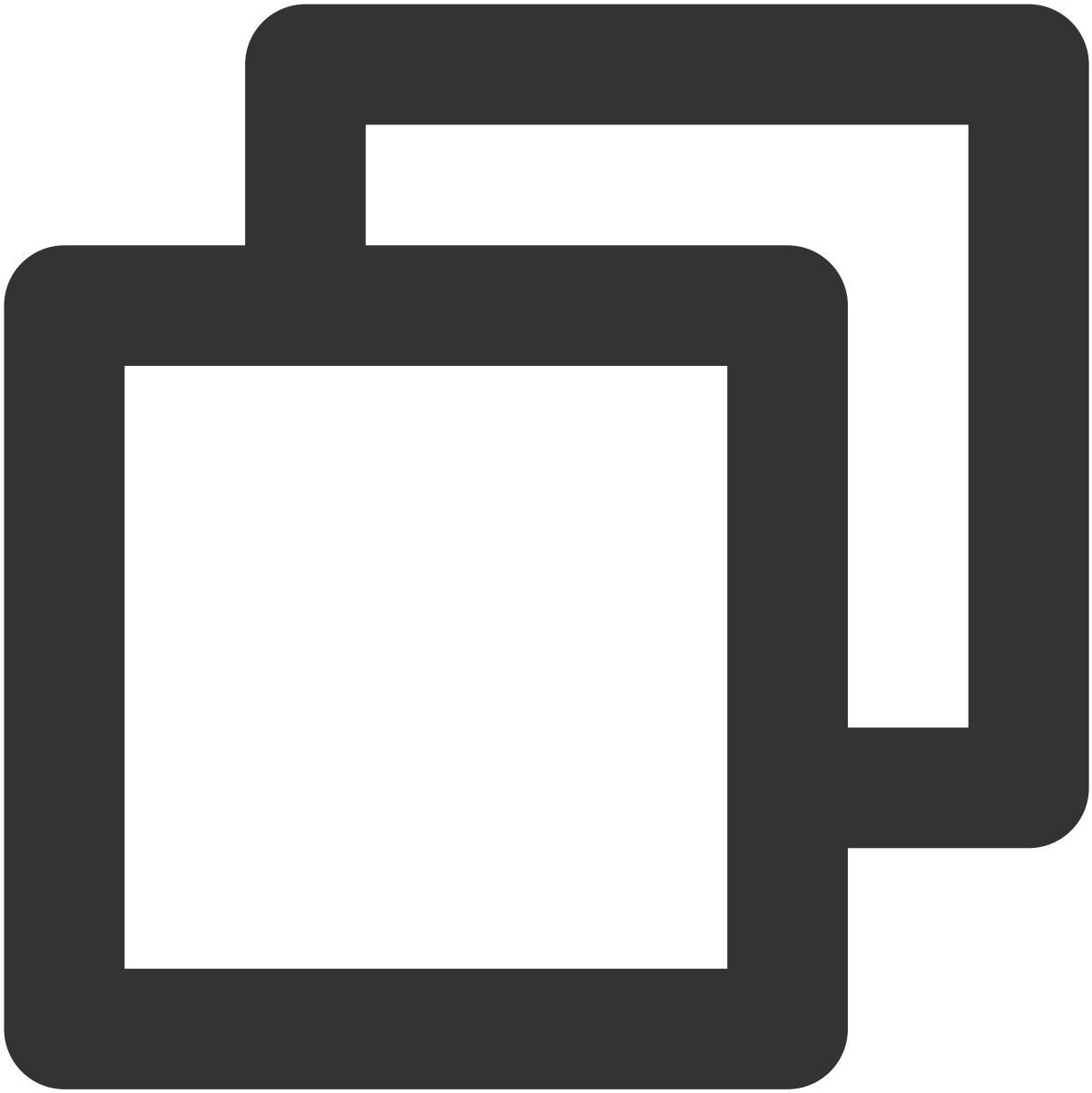
iOS(Objective-C)

Flutter(Dart)

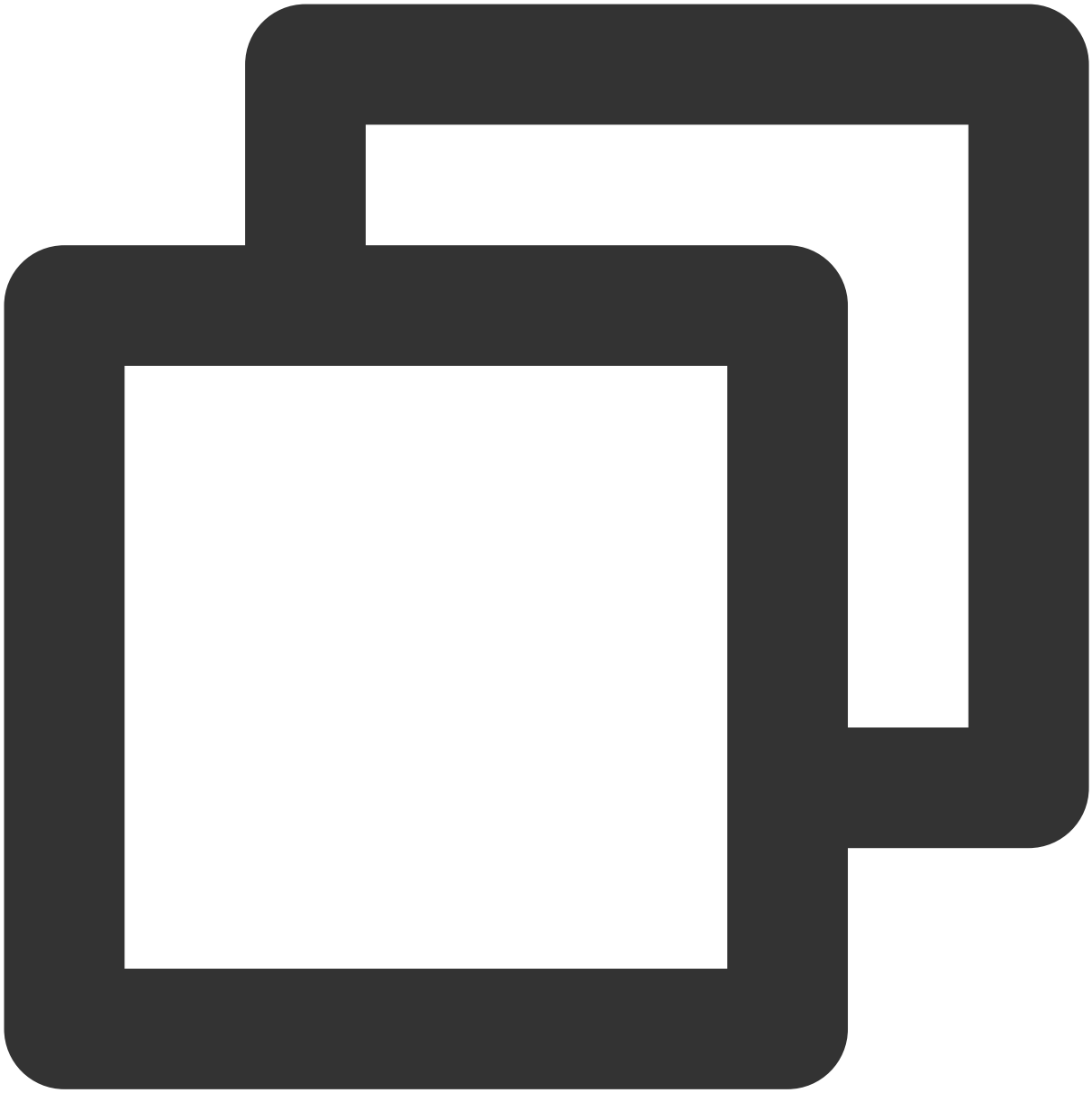


```
import com.tencent.qcloud.tuikit.TUICommonDefine
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

var roomId = TUICommonDefine.RoomId()
roomId.intRoomId = 123321
```

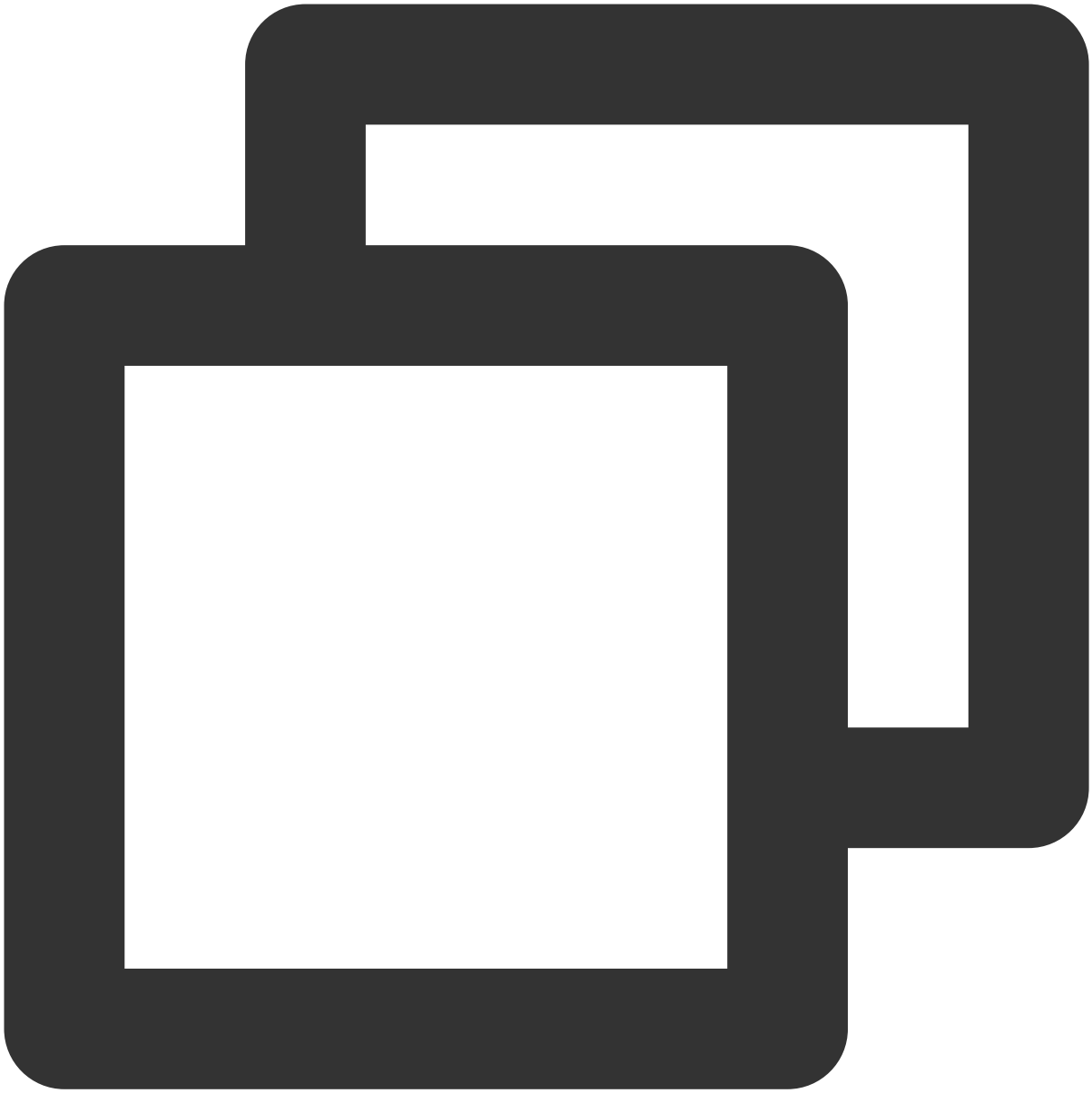
```
var groupId = "12345678"
TUICallKit.createInstance(context).joinInGroupCall(roomId, groupId, TUICallDefine.M
```



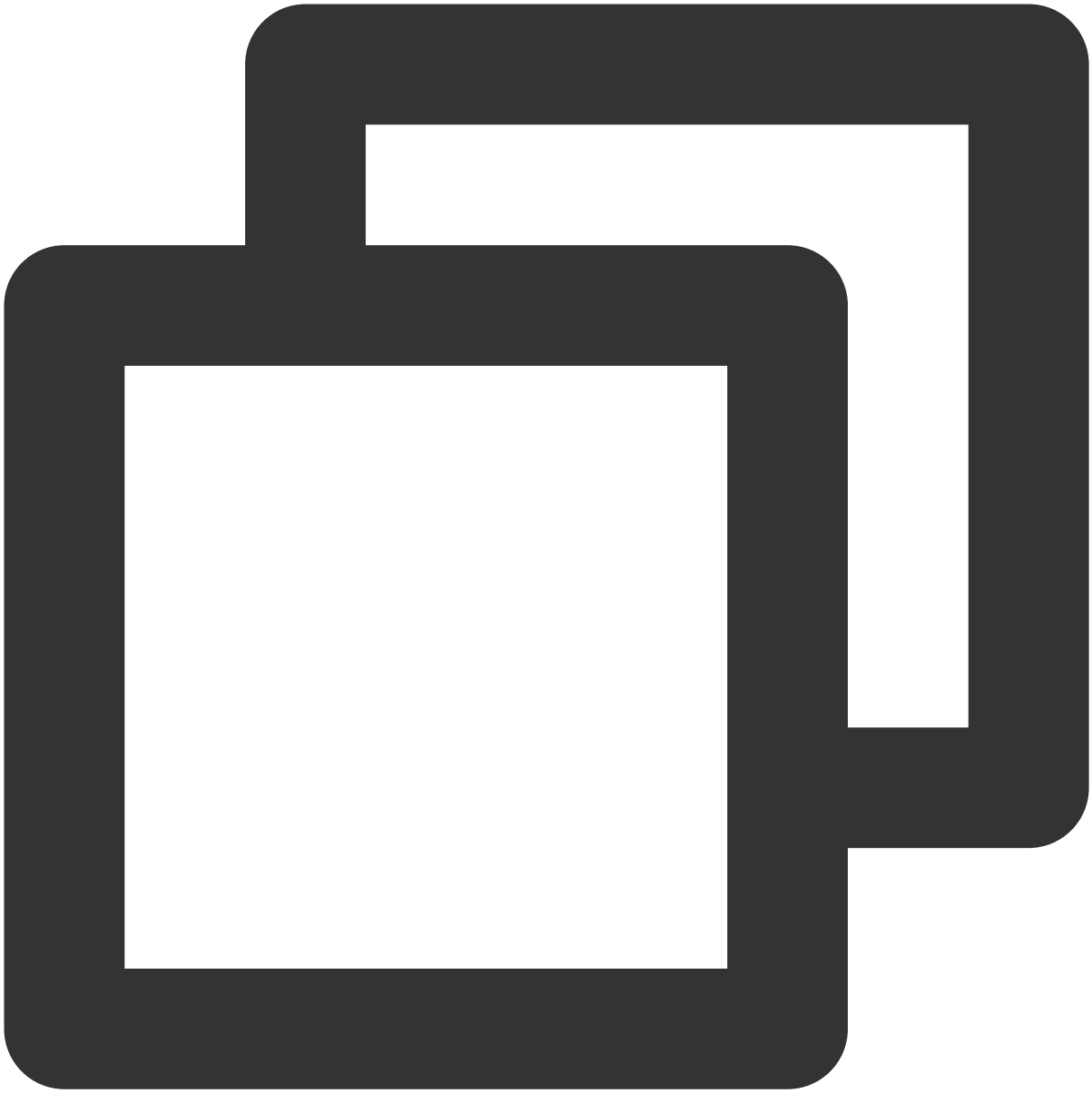
```
import com.tencent.qcloud.tuikit.TUICommonDefine;
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine;
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit;

TUICommonDefine.RoomId roomId = new TUICommonDefine.RoomId();
roomId.intRoomId = 123321;
String groupId = "12345678";
TUICallKit.createInstance(context).joinInGroupCall(roomId, groupId, TUICallDefine.M
```



```
import TUICallKit_Swift
import TUICallEngine

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



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

TUIRoomId *roomId = [[TUIRoomId alloc] init];
roomId.intRoomId = 123321;
[[TUICallKit createInstance] joinInGroupCallWithRoomId:roomId
                                               groupId:@"223344"
                                         callMediaType:TUICallMediaTypeVideo];
```



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

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

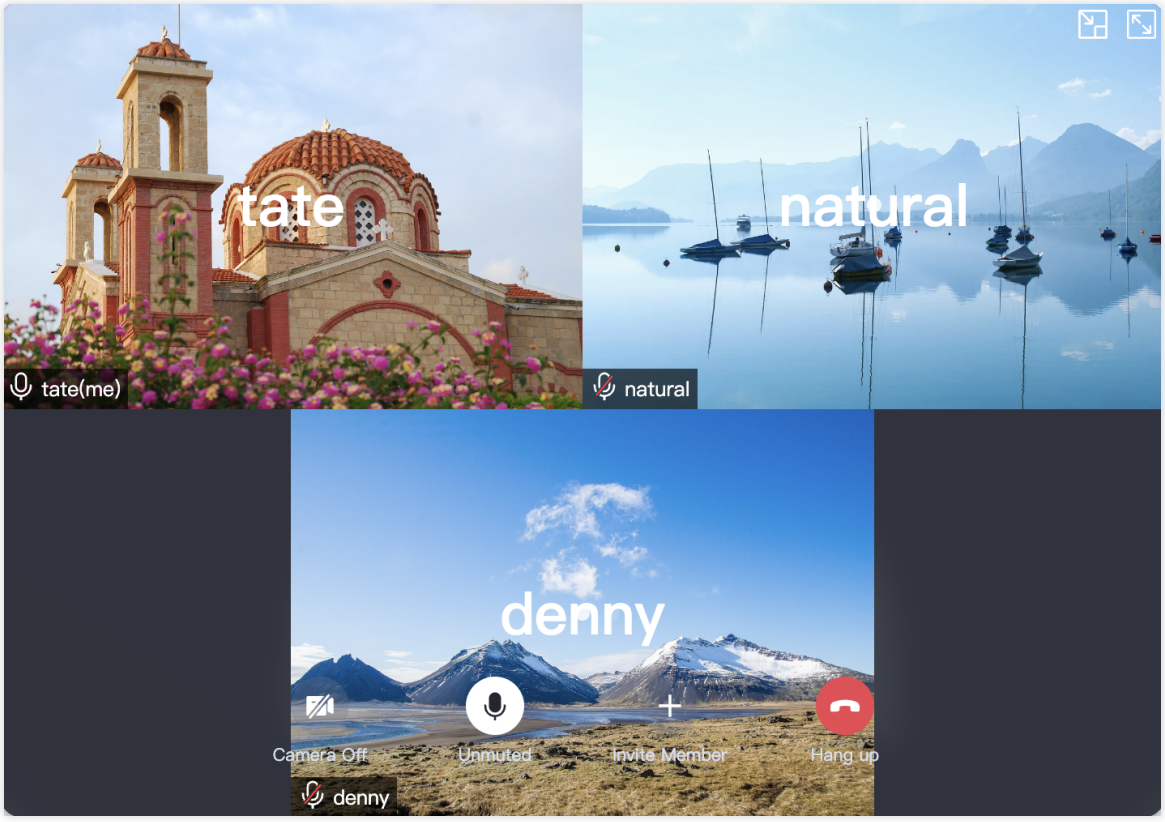
Web&H5

Last updated：2024-06-17 17:41:52

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

Expected outcome

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

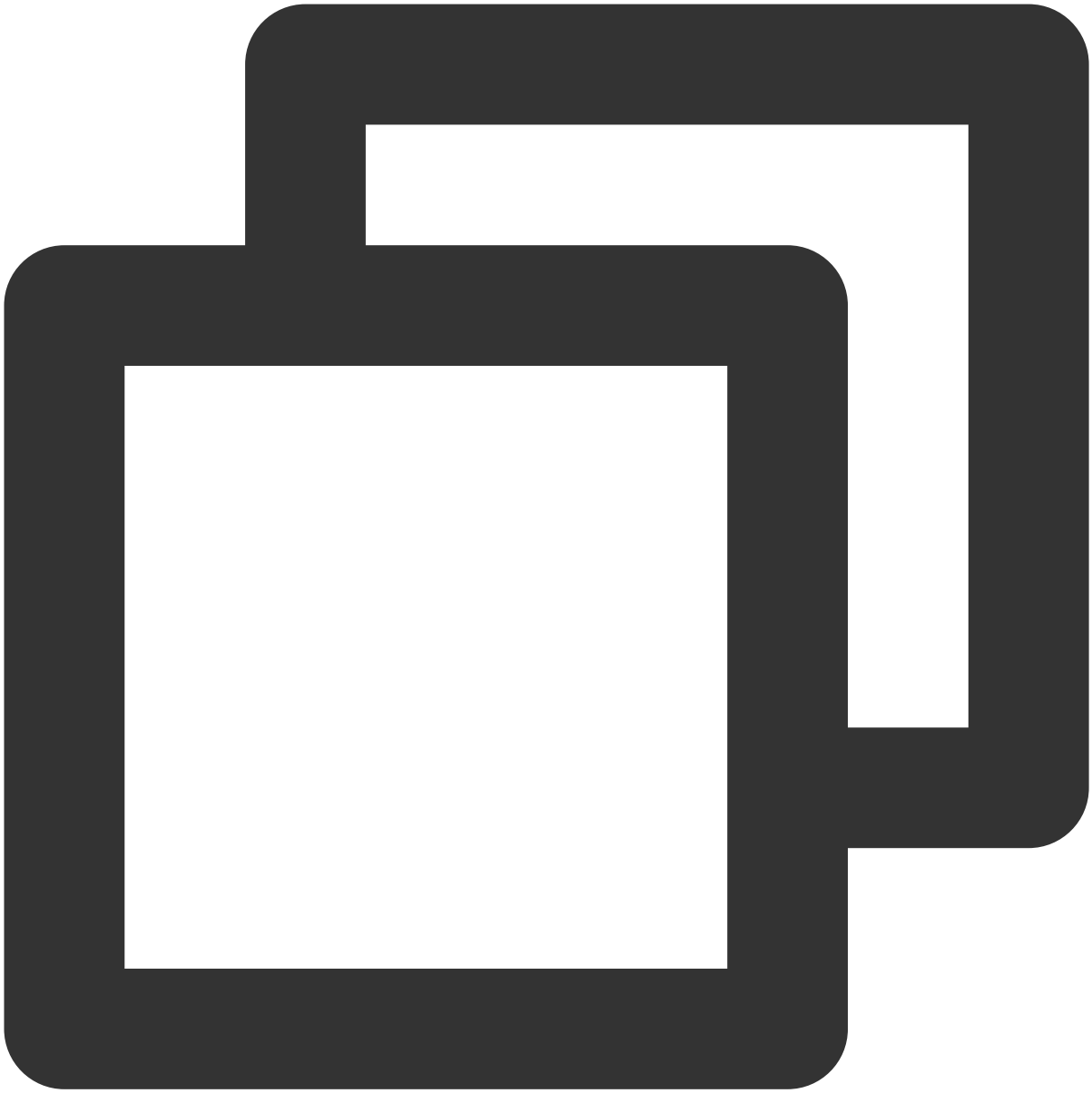
Web	M
	

Create groupID

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

Method one: Create a group by calling the Chat API, see Chat Group Management for details.

Method two: Manually create a group through the console, see Console group management for details.



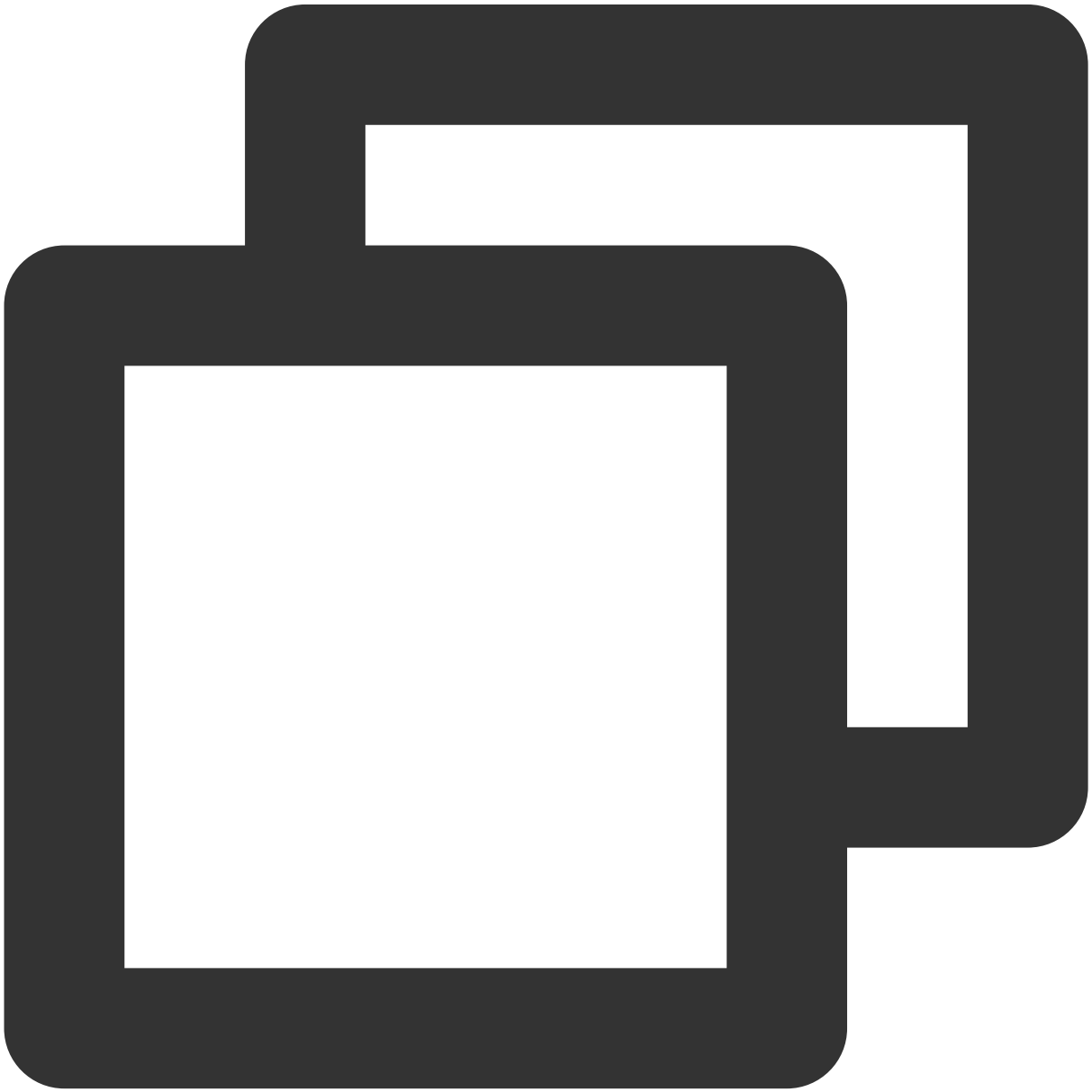
```
import Chat from "@tencentcloud/chat";   // npm i @tencentcloud/chat

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

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

Group call

Initiate a group call

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



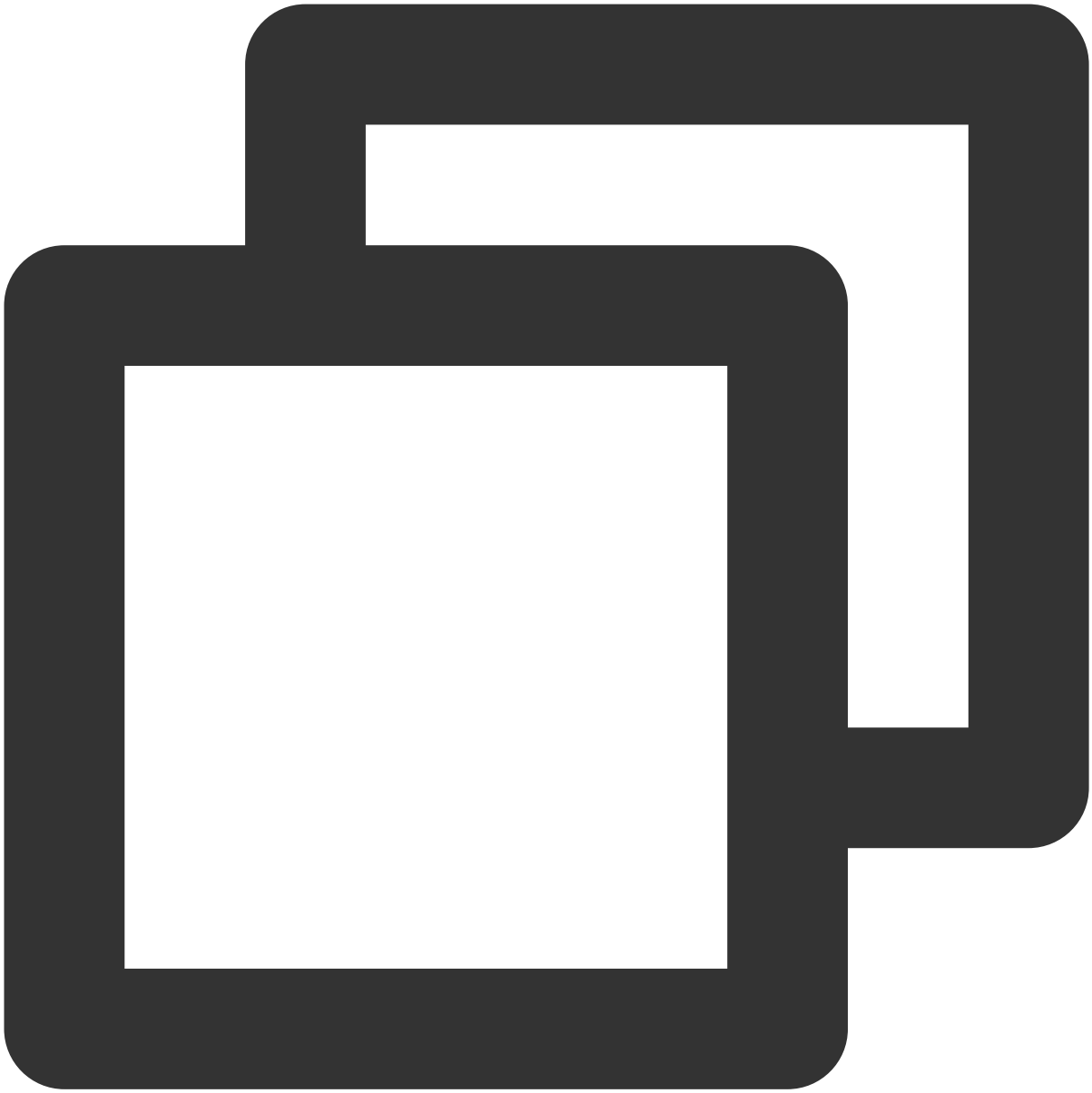
```
import { TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit-react";
// Replace it with the call-uikit npm package you are currently using

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

Join a group call

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

Note:

v3.1.2+ is supported.



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

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

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

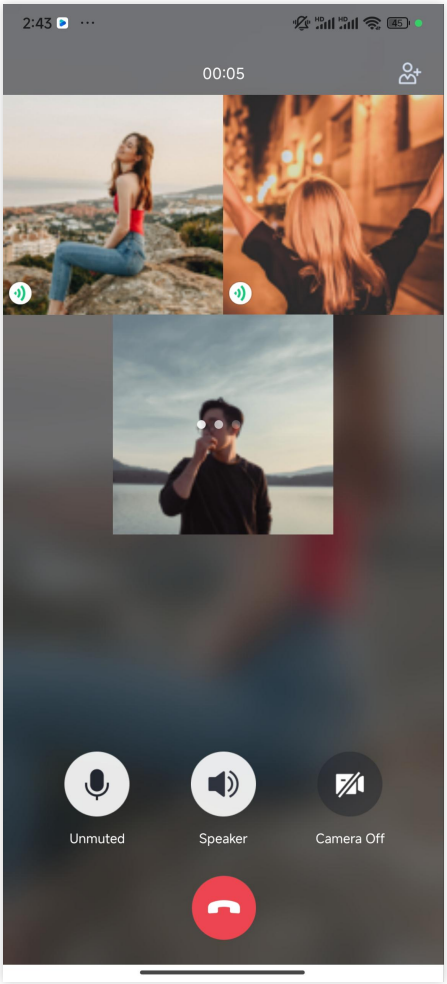
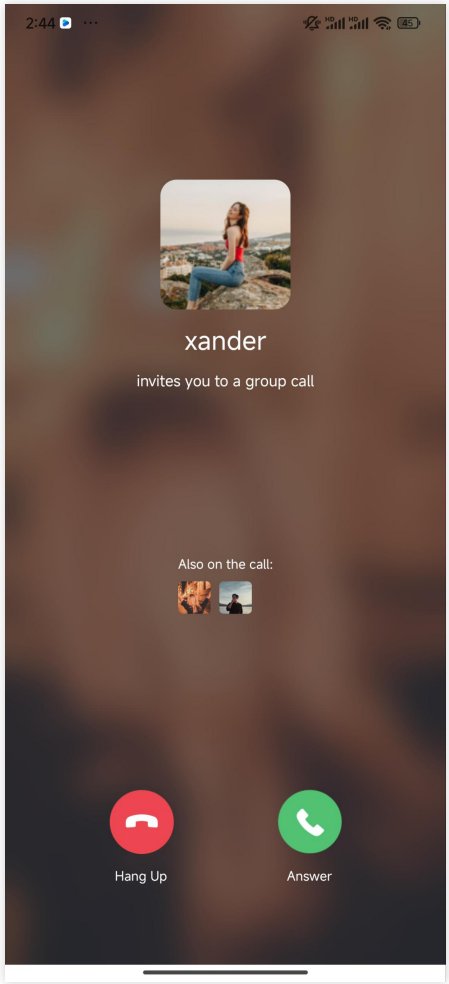
uni-app (Anroid&iOS)

Last updated：2024-04-03 17:23:11

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

Expected outcome

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

Initiate a group call	Received Group Call Invitation	Accept
		

Create groupID

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

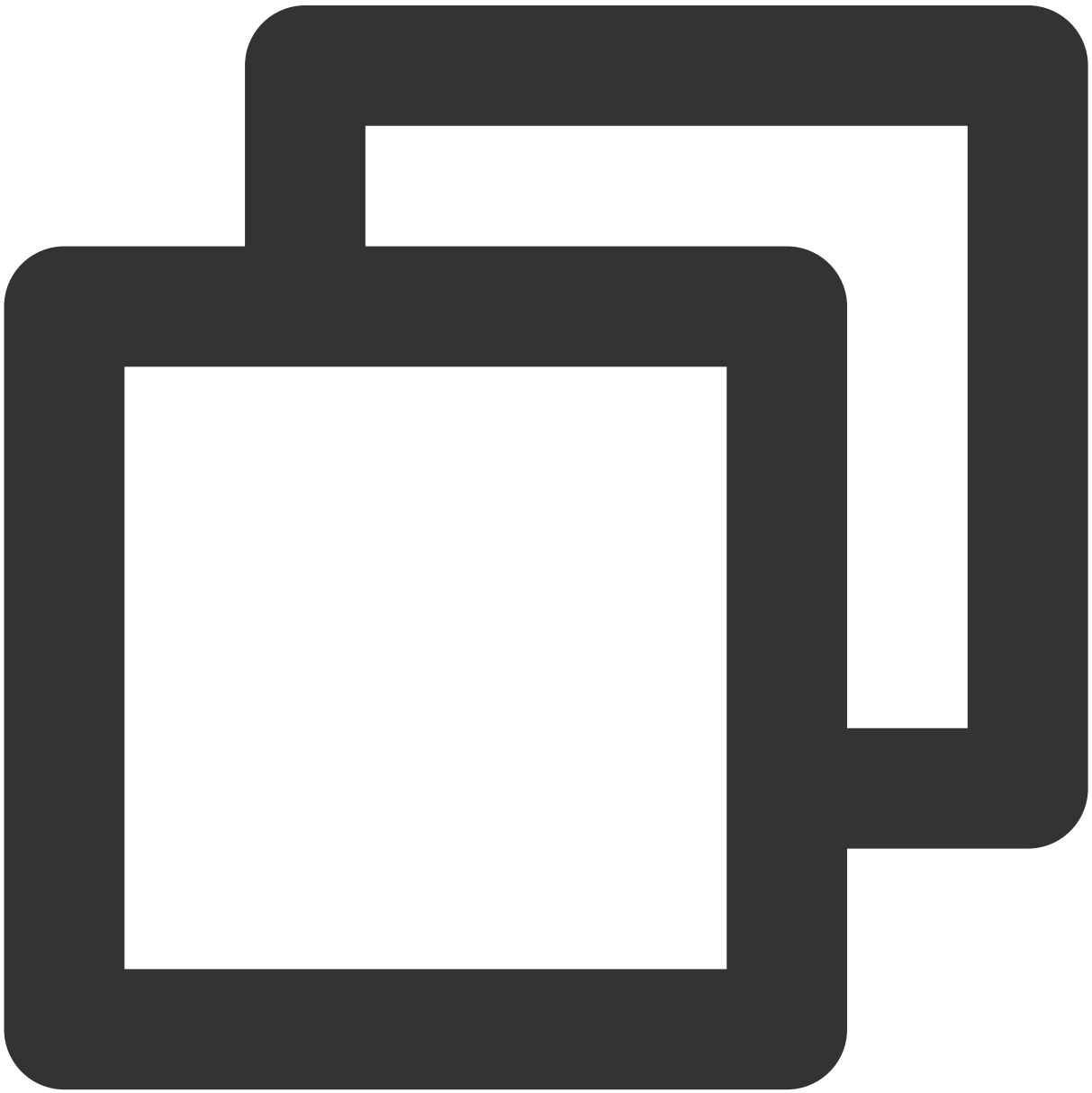
Method one: Create a group by calling the IM API, see IM Group Management for details.

Method two: Manually create a group through the console, see Console group management for details.

Group call

Initiate a group call

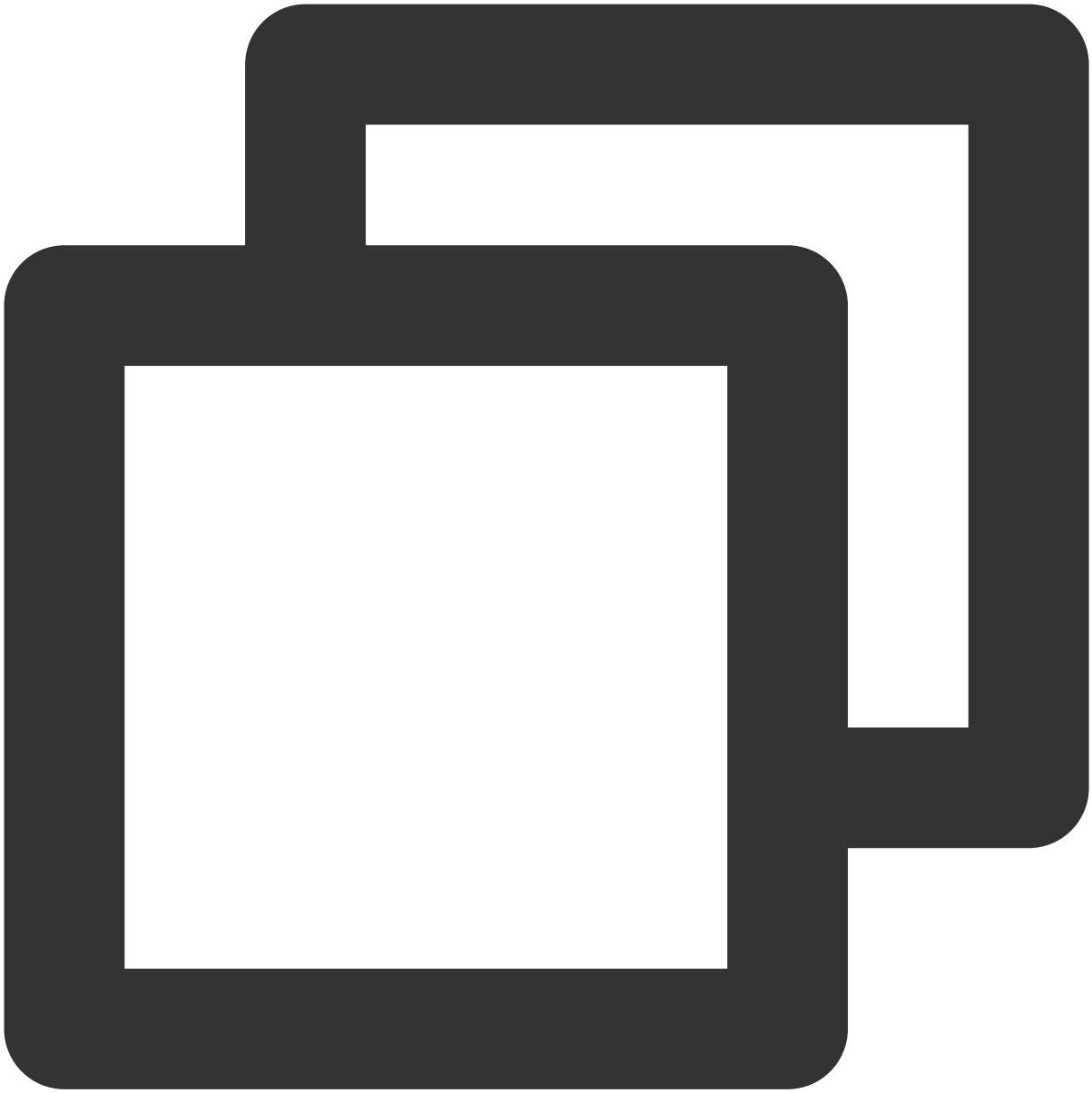
Launch a group call using the groupCall API.



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

Join a group call

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



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

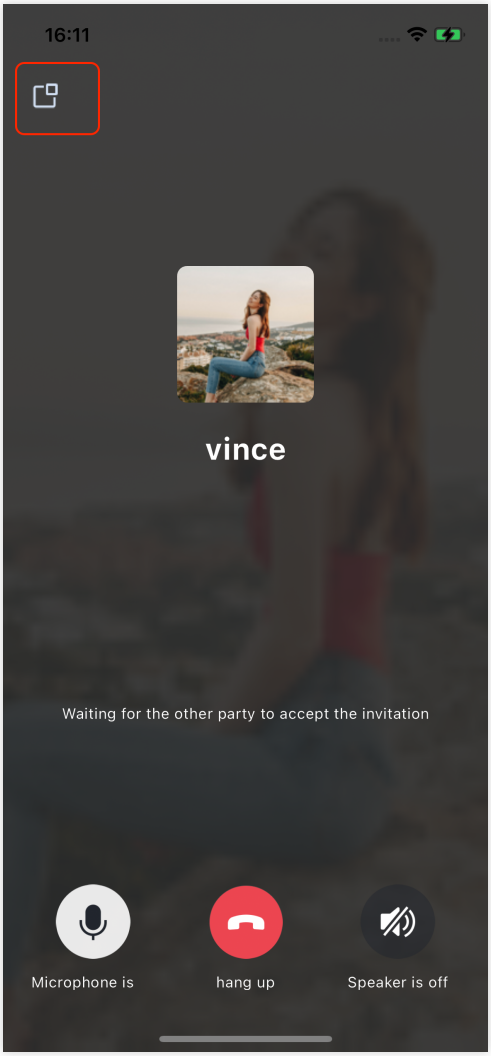
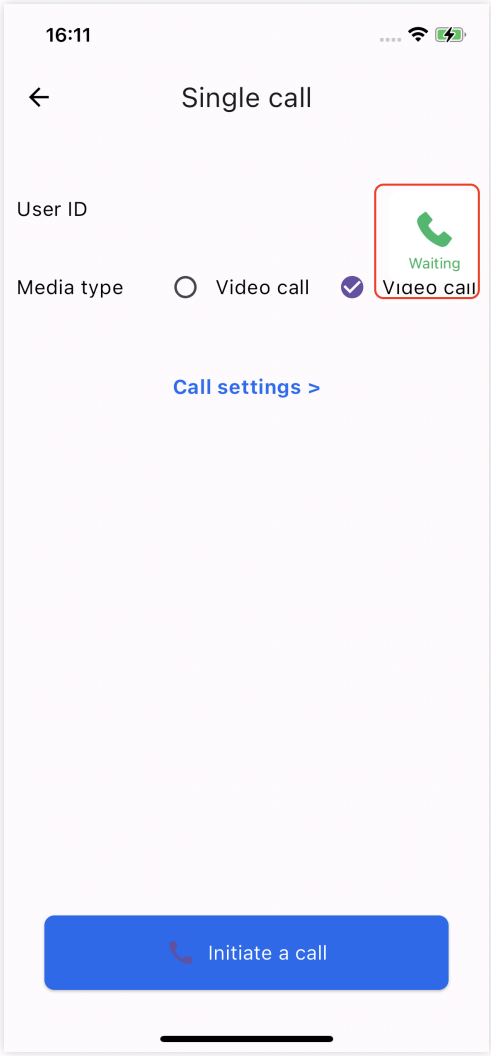
```
    }
});
```

Floating Window Android&iOS&Flutter

Last updated：2024-05-24 18:35:12

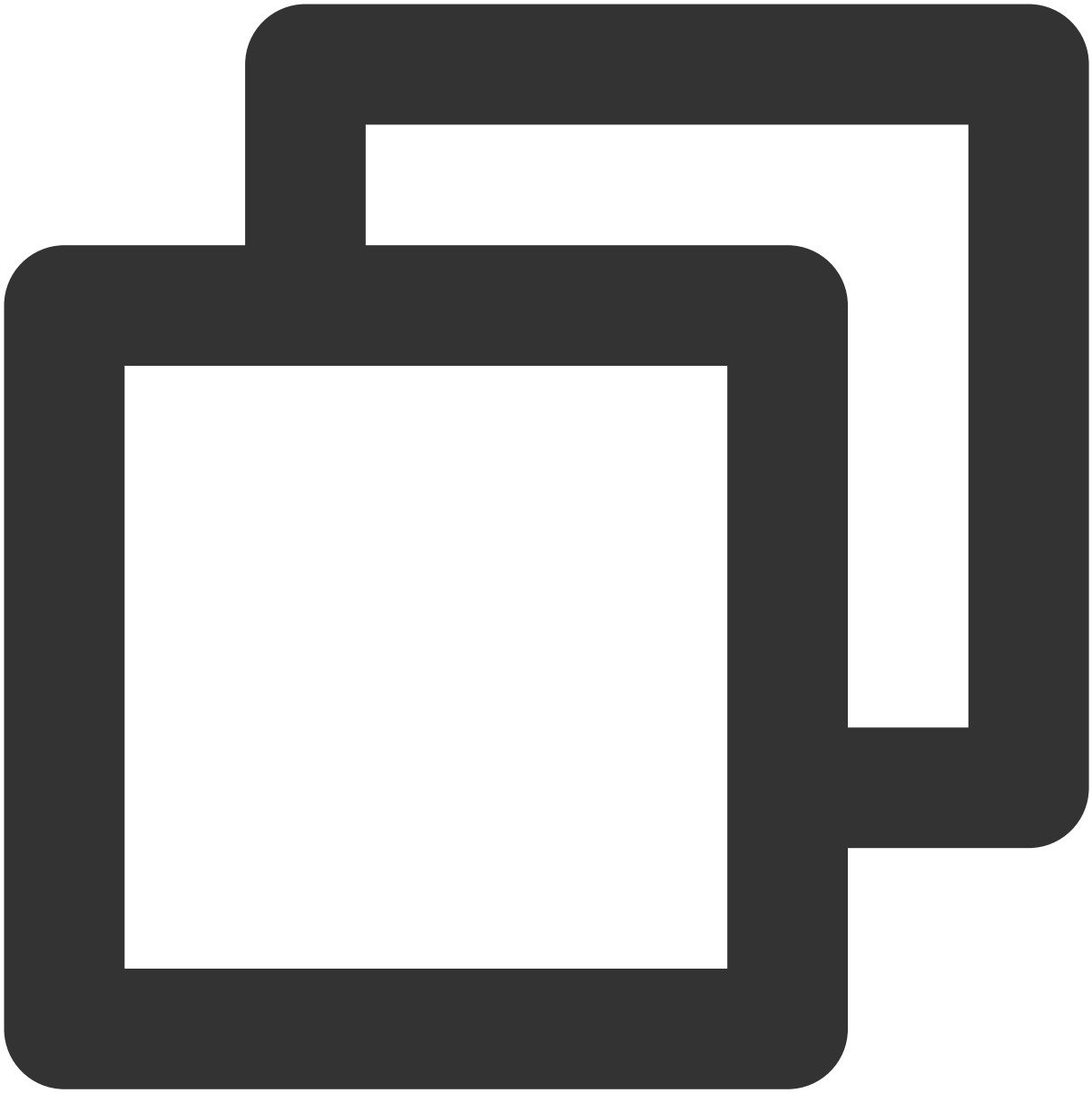
This article explains how to use the Floating Window feature.

Expected outcome

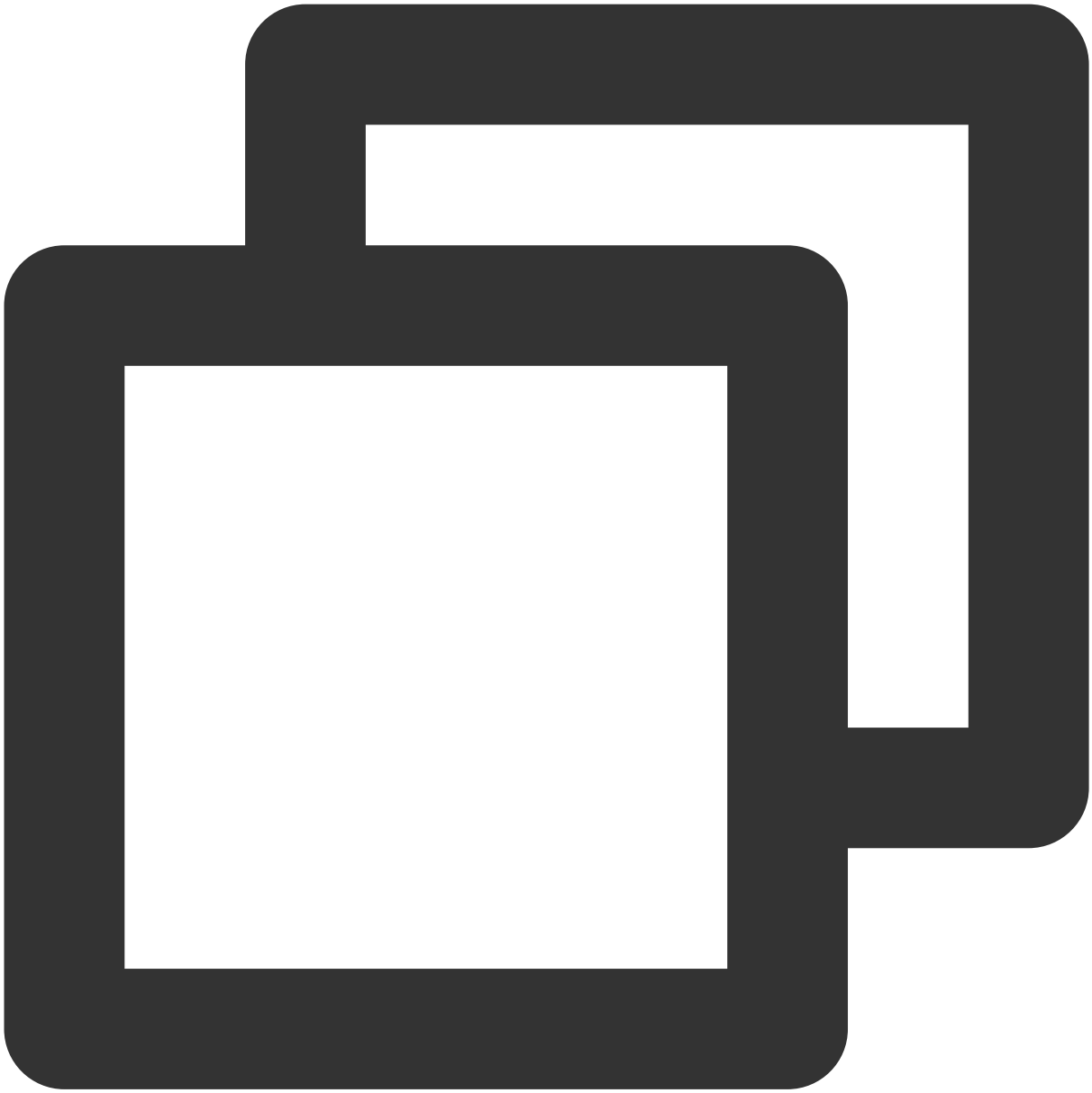
Activate floating button	Voice call floating window	
		

Floating Window feature

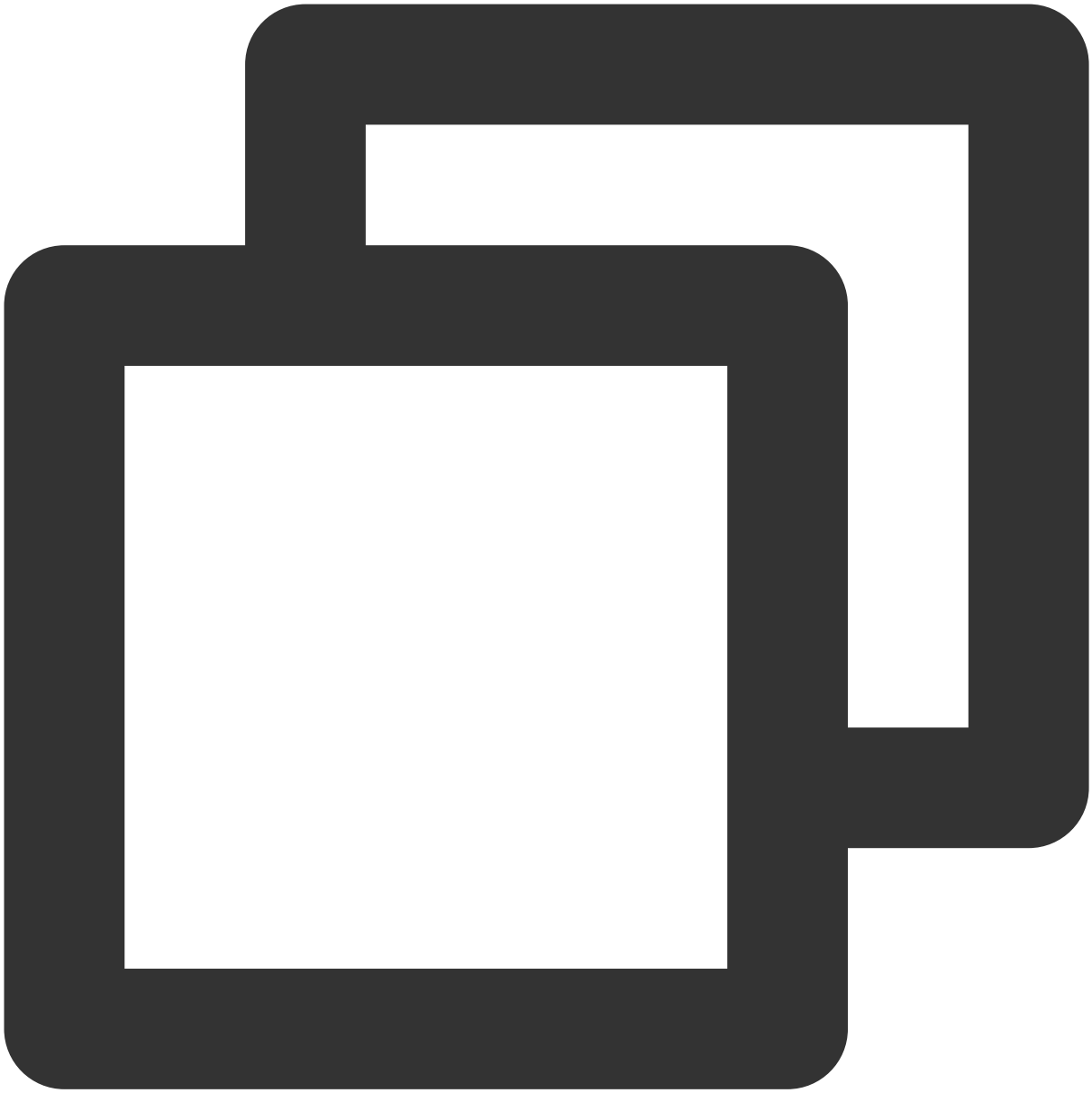
TUICallKit allows users to minimize the call interface into a floating window using the floating window button at the top left corner of the call interface during a call.
If your business needs to enable this feature, you can use the enableFloatWindow method to activate this feature during the initialization of the TUICallKit component:
Android(Kotlin)
Android(Java)
iOS(Swift)
iOS(Objective-C)
Flutter(Dart)



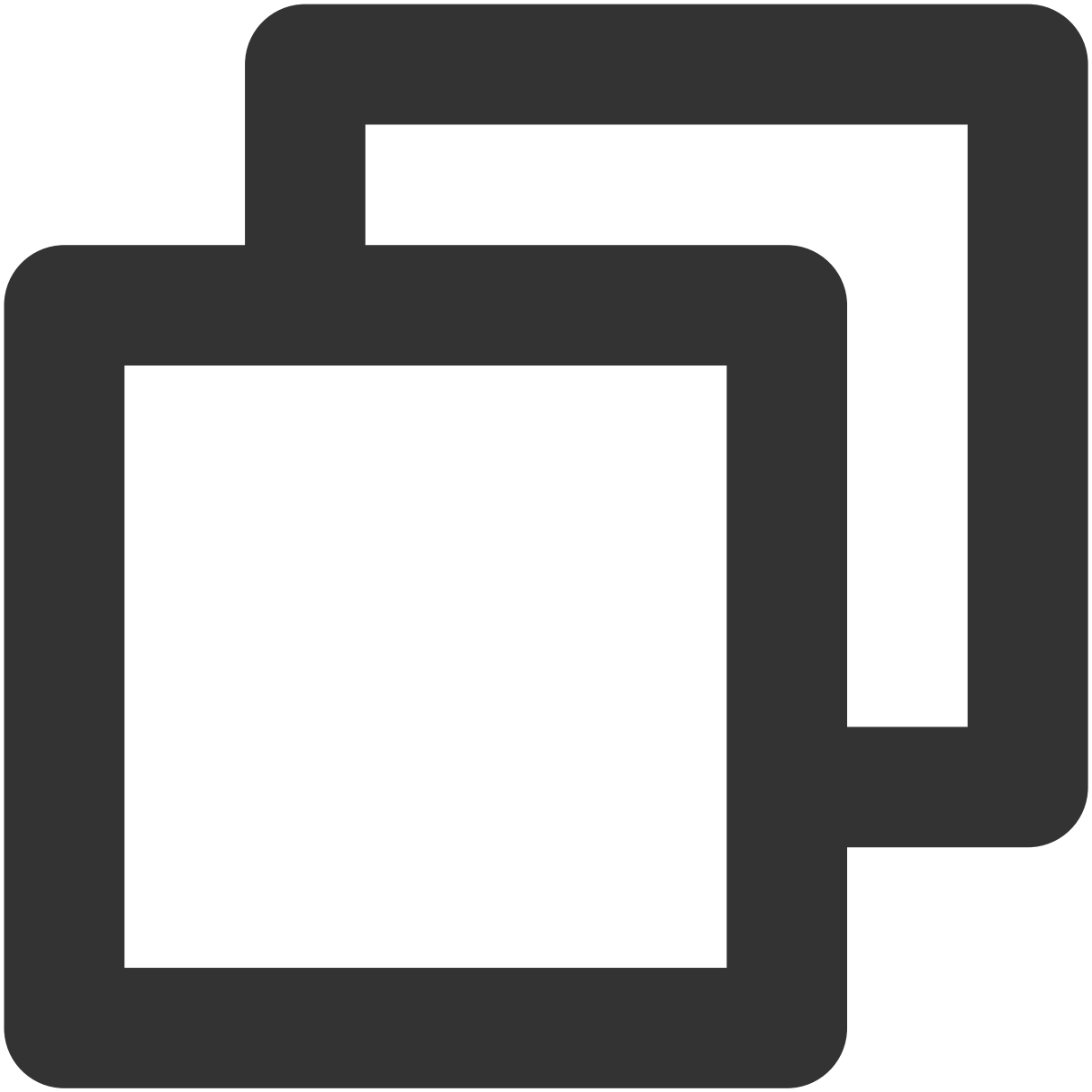
```
TUICallKit.createInstance(context).enableFloatWindow(true)
```



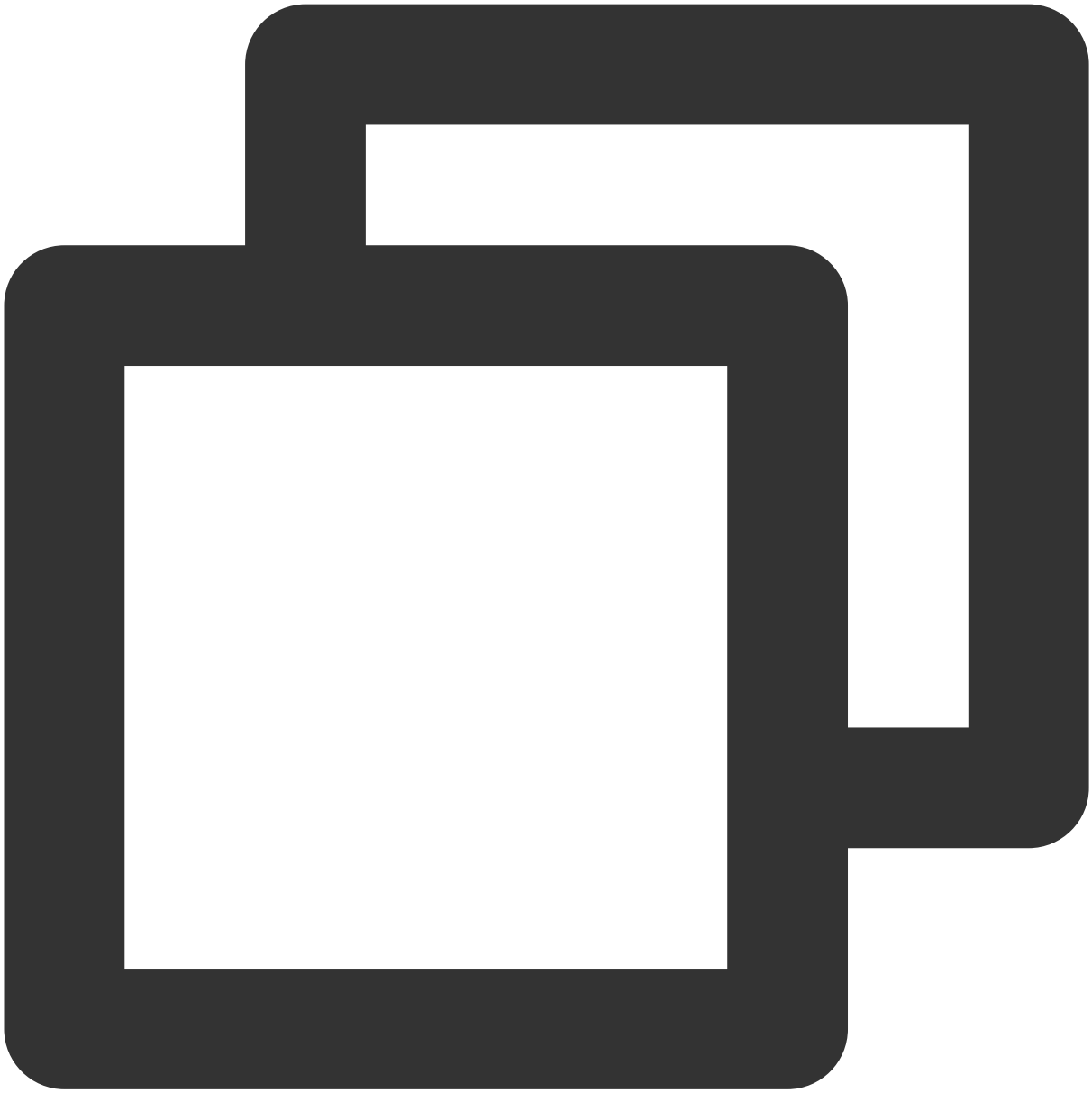
```
TUICallKit.createInstance(context).enableFloatWindow(true);
```



```
import TUICallKit_Swift

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



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

[[TUICallKit createInstance] enableFloatWindowWithEnable:YES];
```



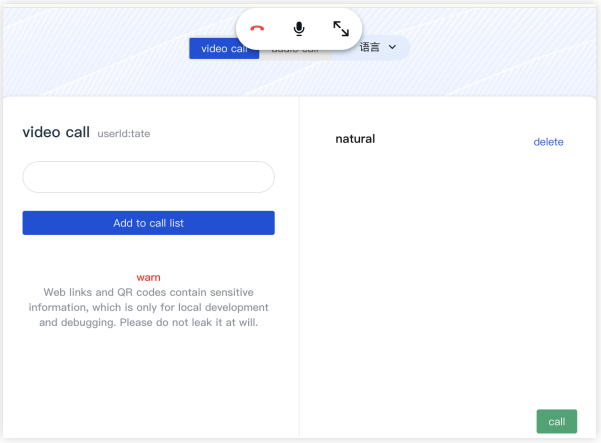
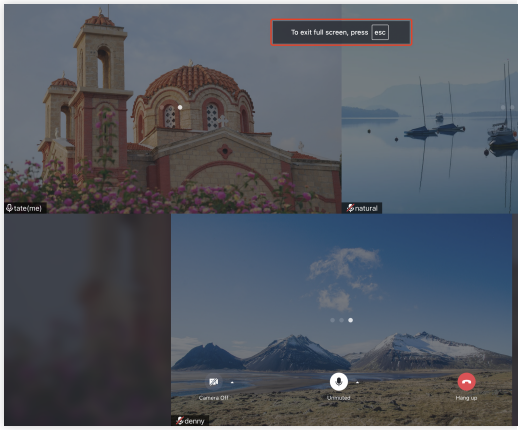
```
import 'package:tencent_calls_uikit/tencent_calls_uikit.dart';

void enableFloatWindow() {
    TUICallKit.instance.enableFloatWindow(true);
}
```

Web&H5

Last updated：2024-05-24 18:35:12

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

Expected outcome

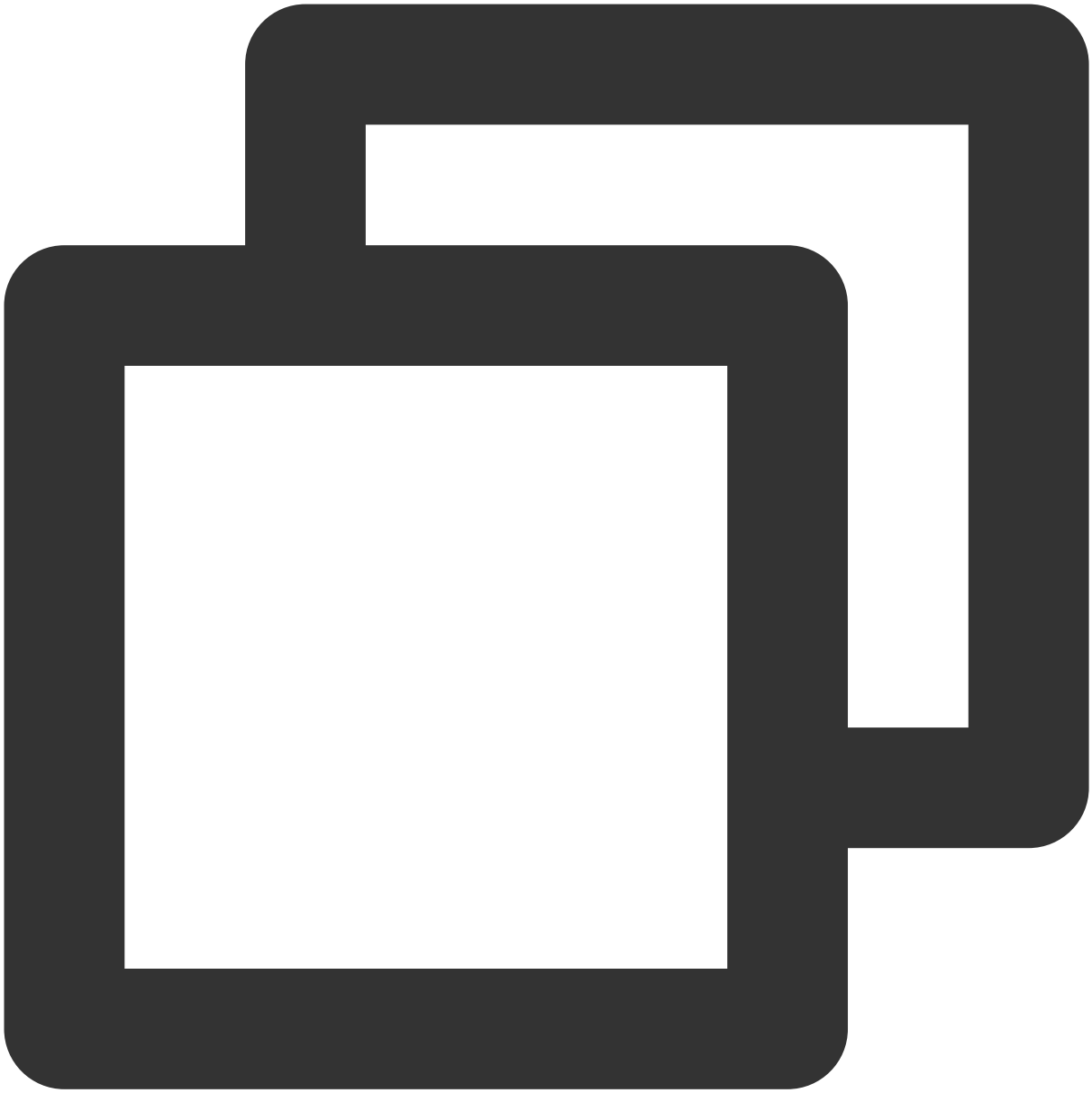
Web Floating Window	Web Full Screen
	

Floating Window feature

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

Note:

Vue ≥ v3.1.0 is supported.



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

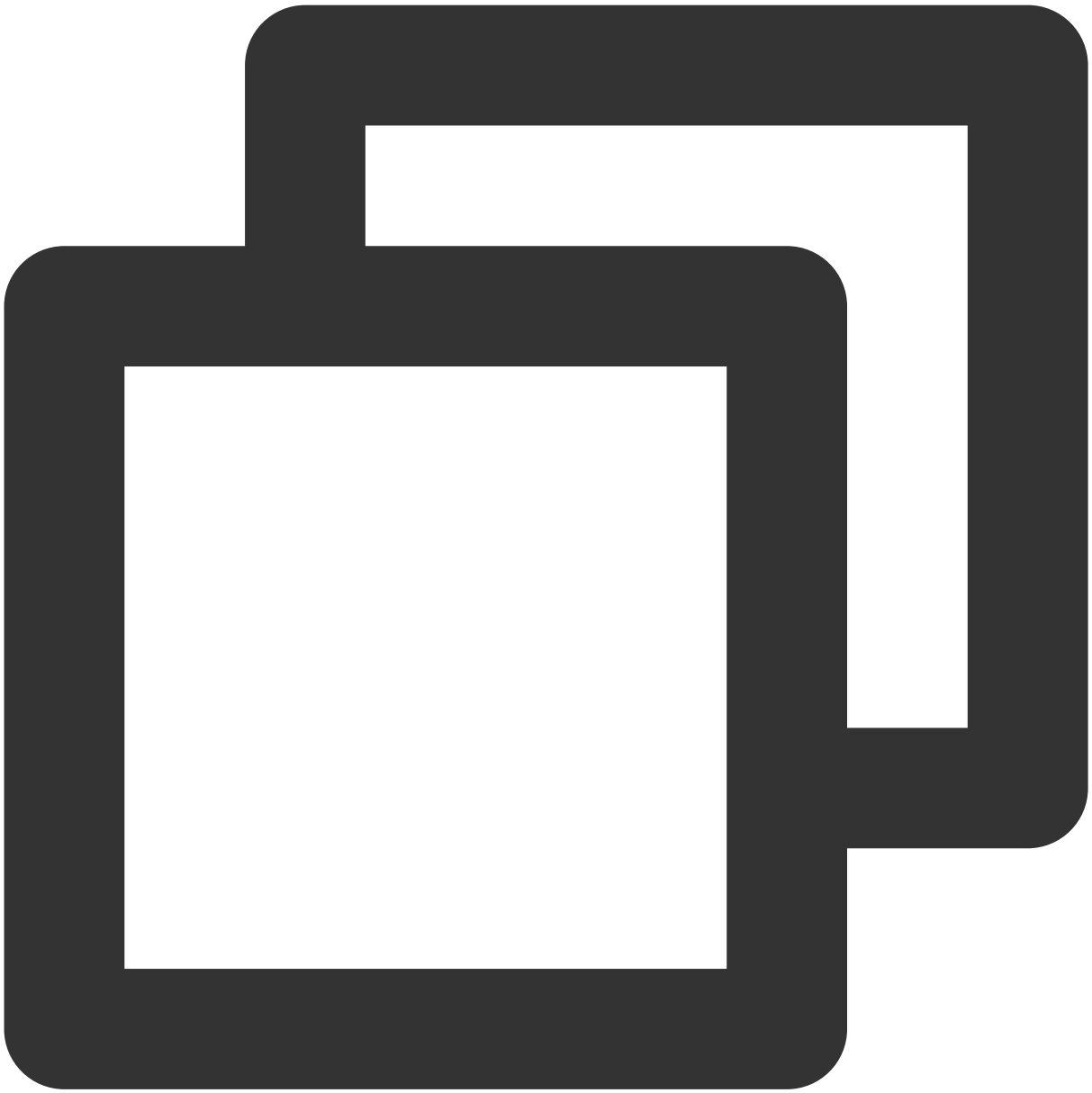
Method 2: Control the Floating Window and the Full Screen on/off through attribute control.

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

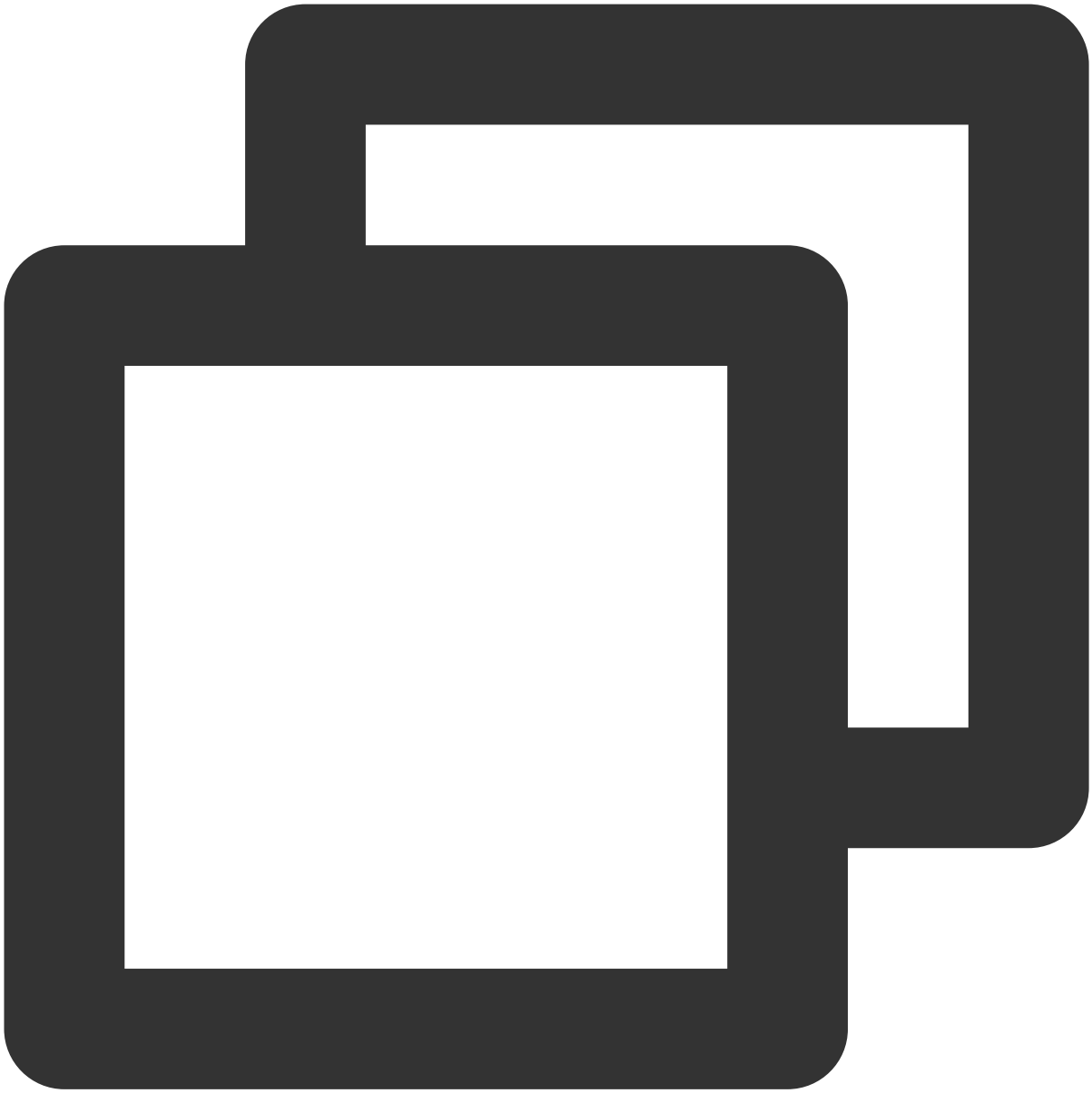
The `allowedMinimized` attribute controls the enabling/disabling of the Full Screen.

React

Vue



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



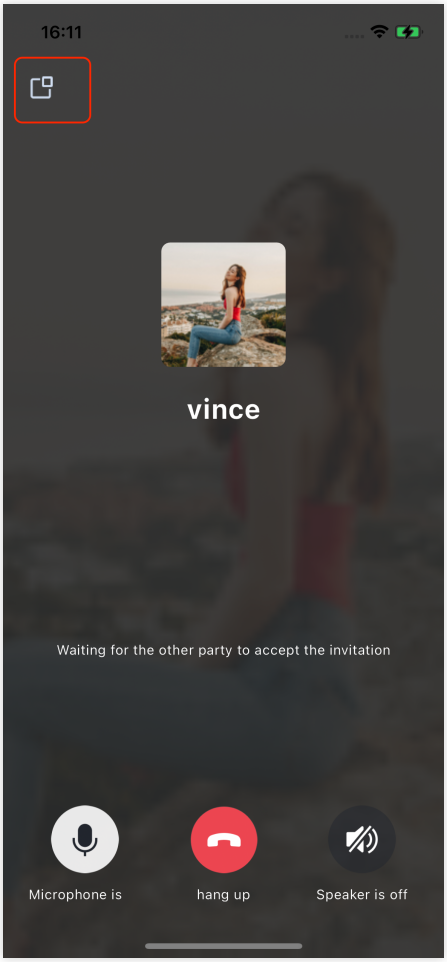
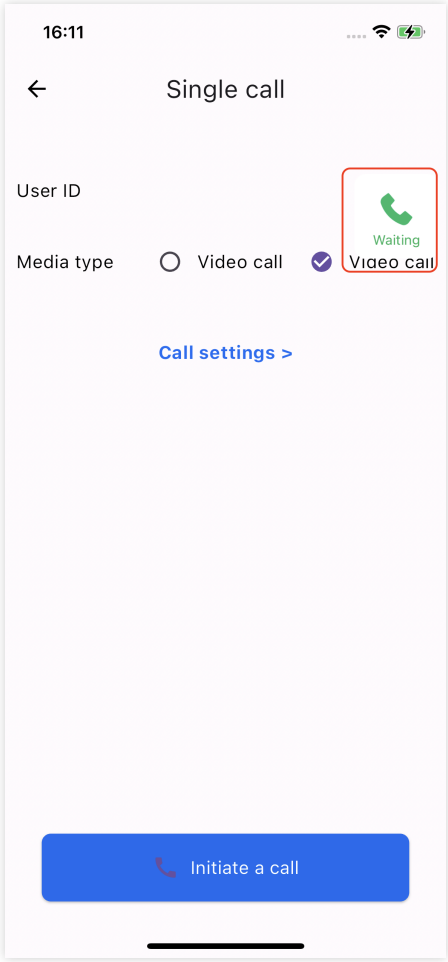
```
<TUICallKit
    :allowedMinimized="true"
    :allowedFullScreen="true"
   />
```

uni-app (Anroid&iOS)

Last updated：2024-04-03 17:23:11

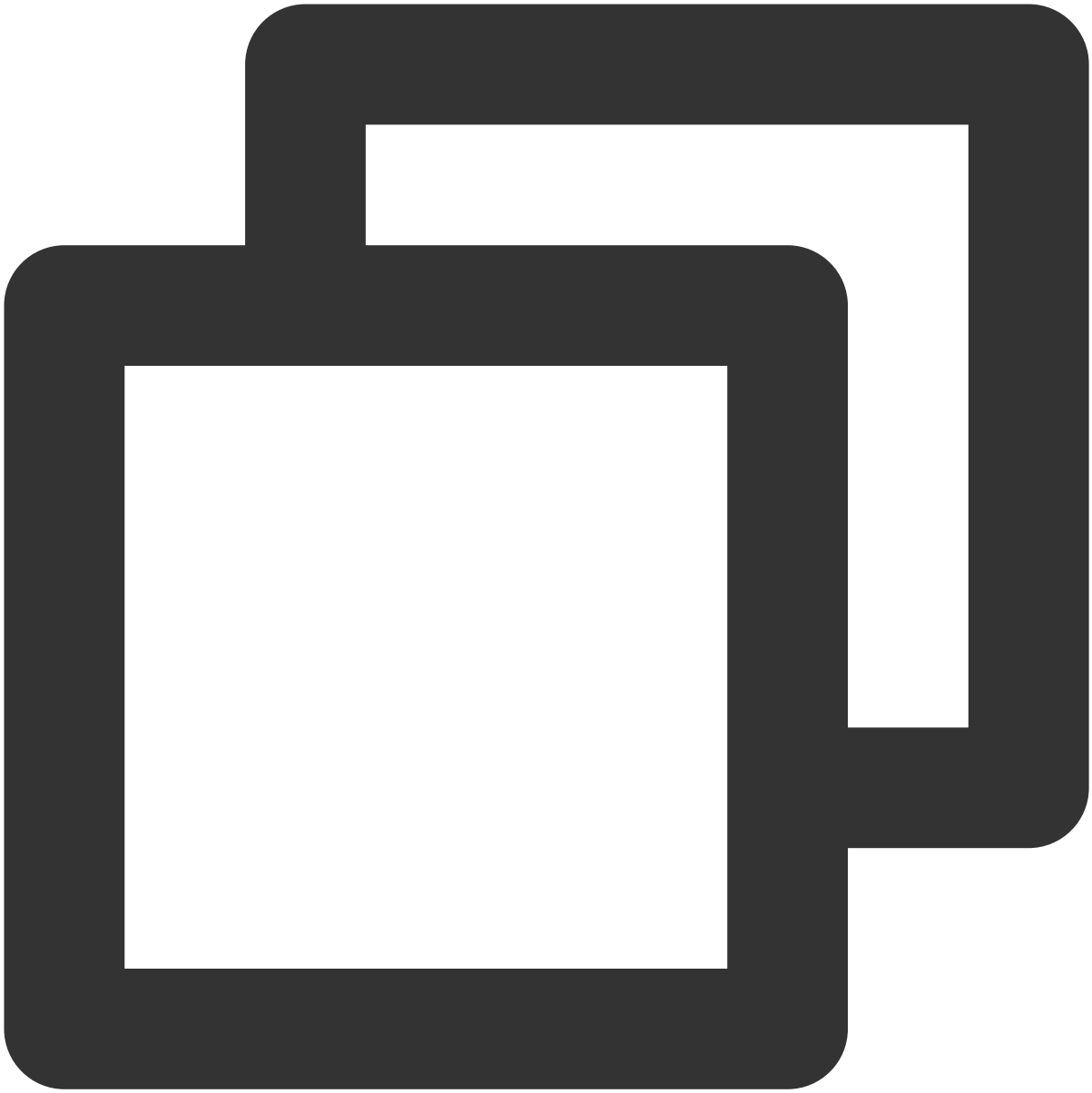
This article explains how to use the Floating Window feature.

Expected outcome

Activate floating button	Voice call floating window	Video
		

Floating Window feature

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

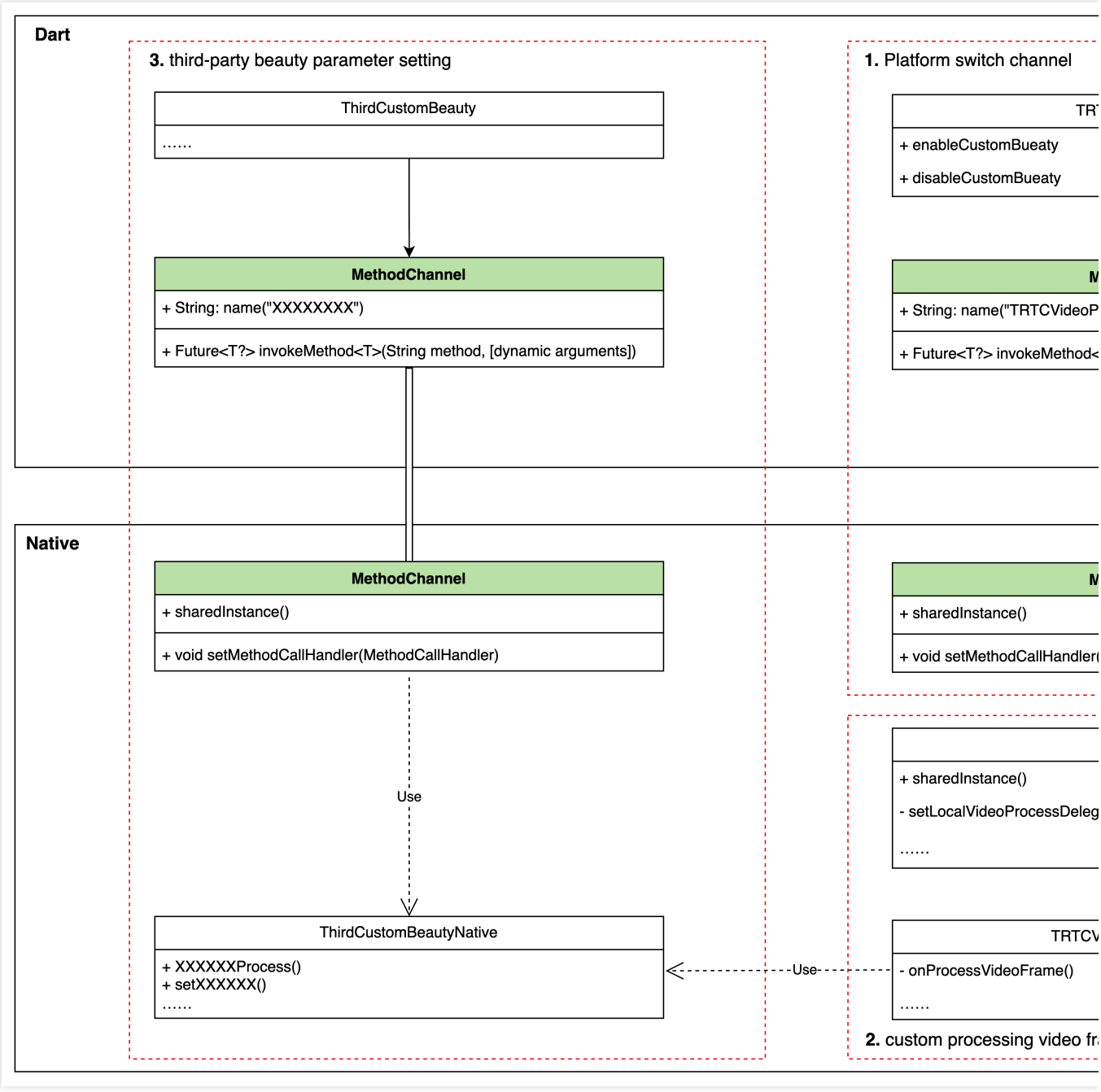


```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableFloatWindow(enable);
```

Beauty Effects
Flutter

Last updated：2024-04-03 17:23:11

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



The access plan is divided into 3 steps:

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

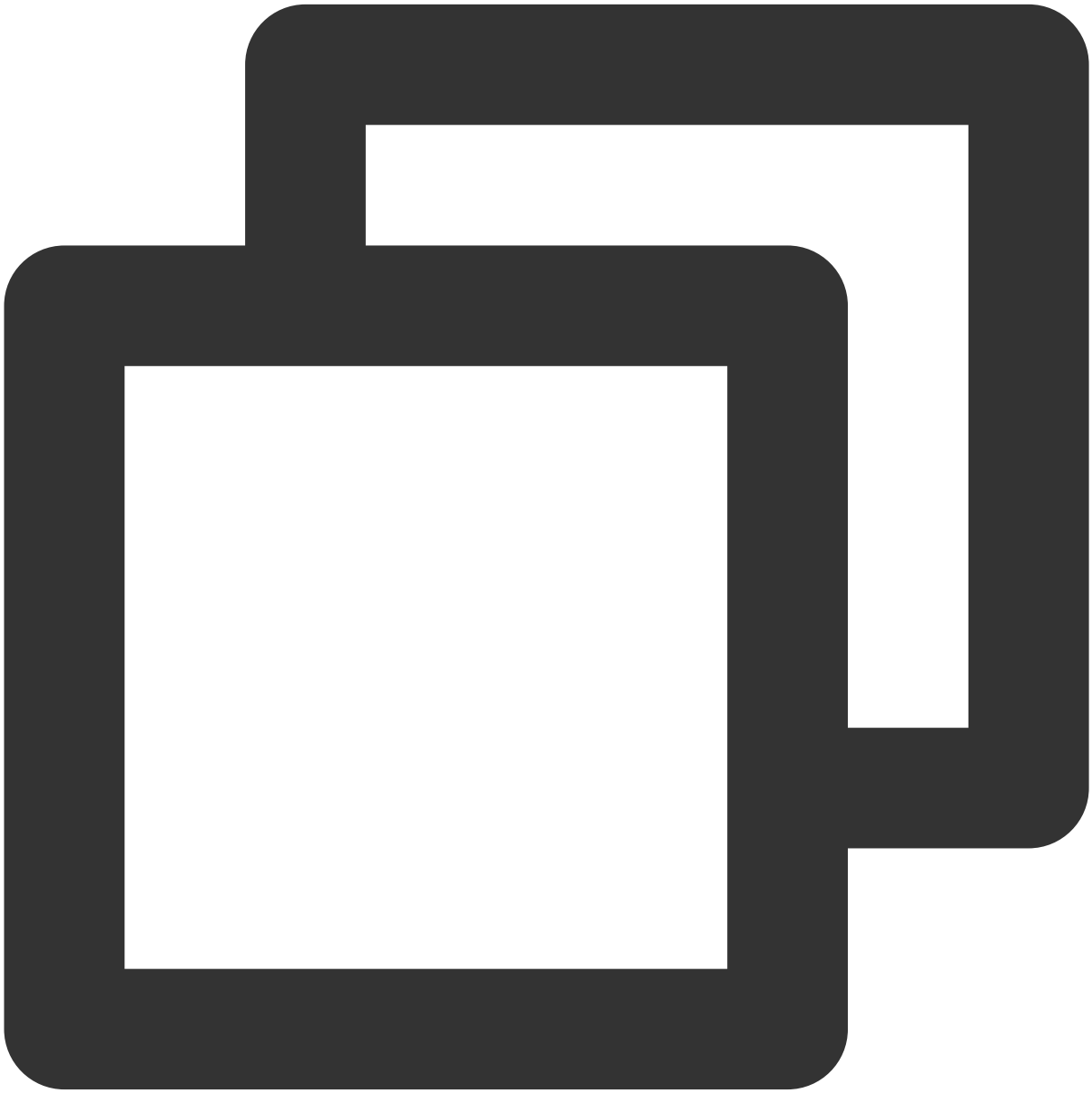
Step 2: Use the beauty processing module in TRTC's custom rendering processing logic onProcessVideoFrame() to process the original video frame.

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

Integrate third-party beauty effects

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

Implement Dart layer interface:



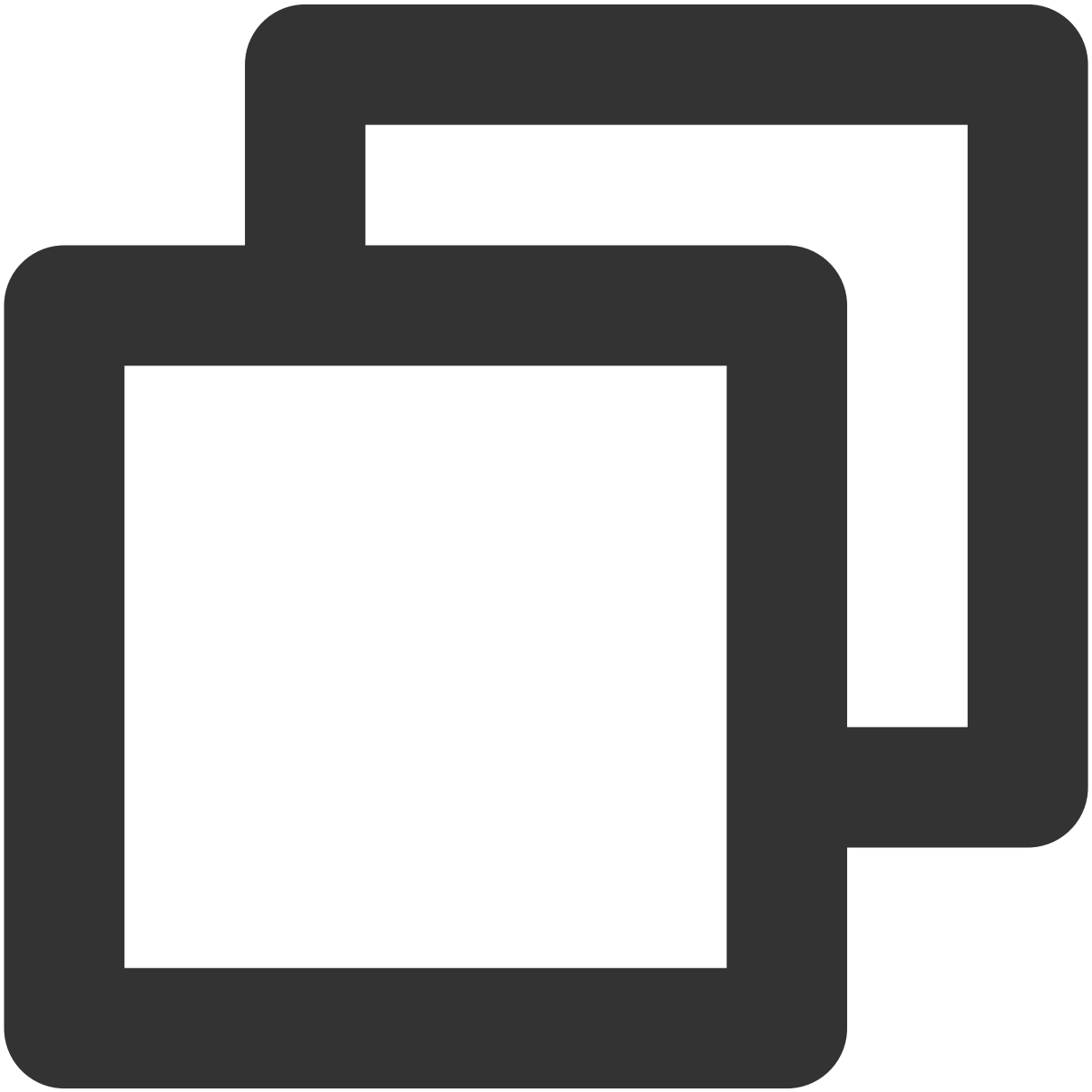
```
final channel = MethodChannel('TUICallKitCustomBeauty');

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

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

Implement the corresponding Native layer interface:

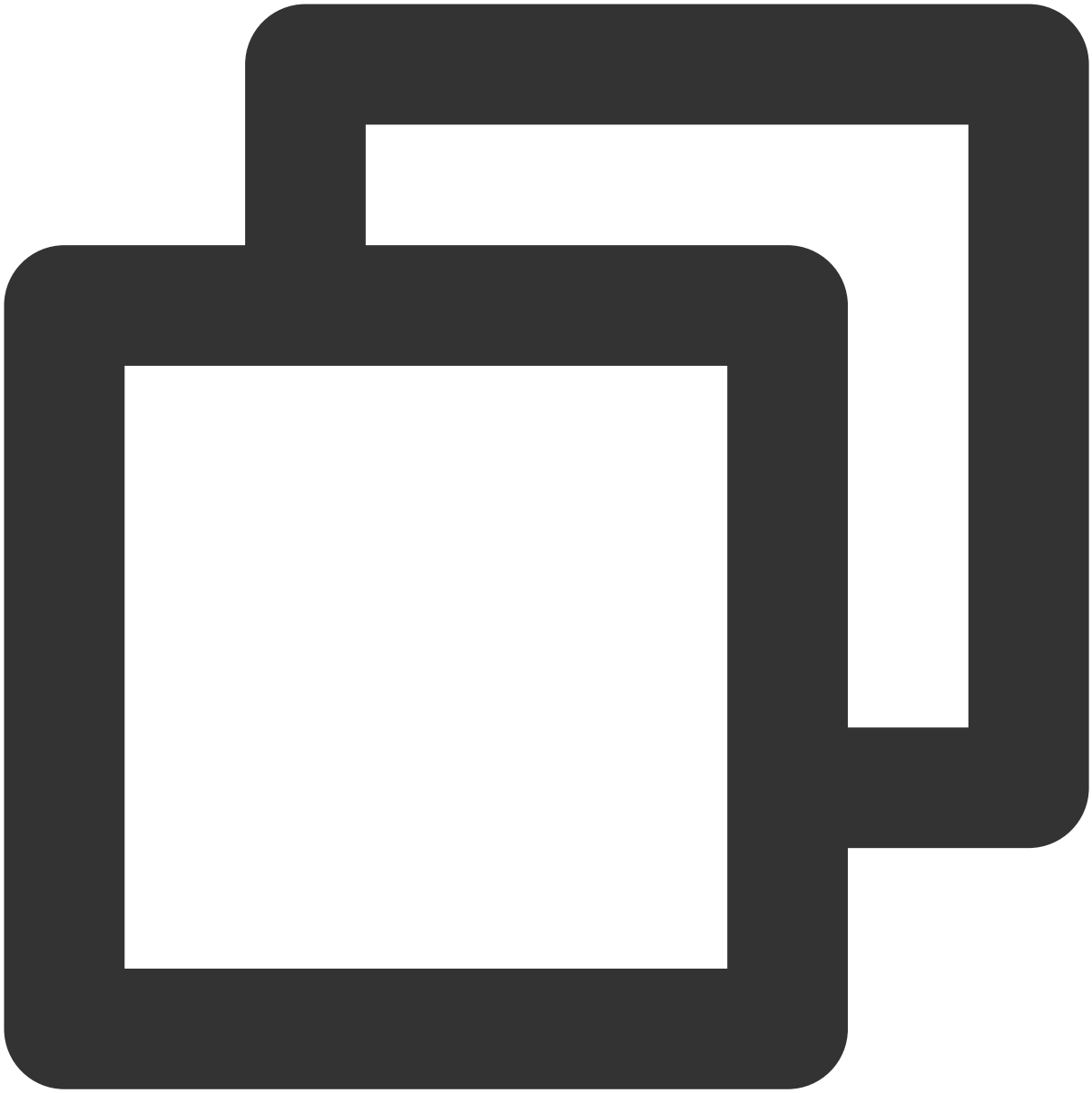
java

swift



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

```
    private MethodChannel channel;

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

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

    public void enableTUICallKitCustomBeauty() {

    }

    public void disableTUICallKitCustomBeauty() {

    }
}
```



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

    override func application(_ application: UIApplication,
                             didFinishLaunchingWithOptions launchOptions: [UIAppli
        GeneratedPluginRegistrant.register(with: self)

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

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

    func enableTUICallKitCustomBeauty() {

    }

    func disableTUICallKitCustomBeauty() {

    }
}
```

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

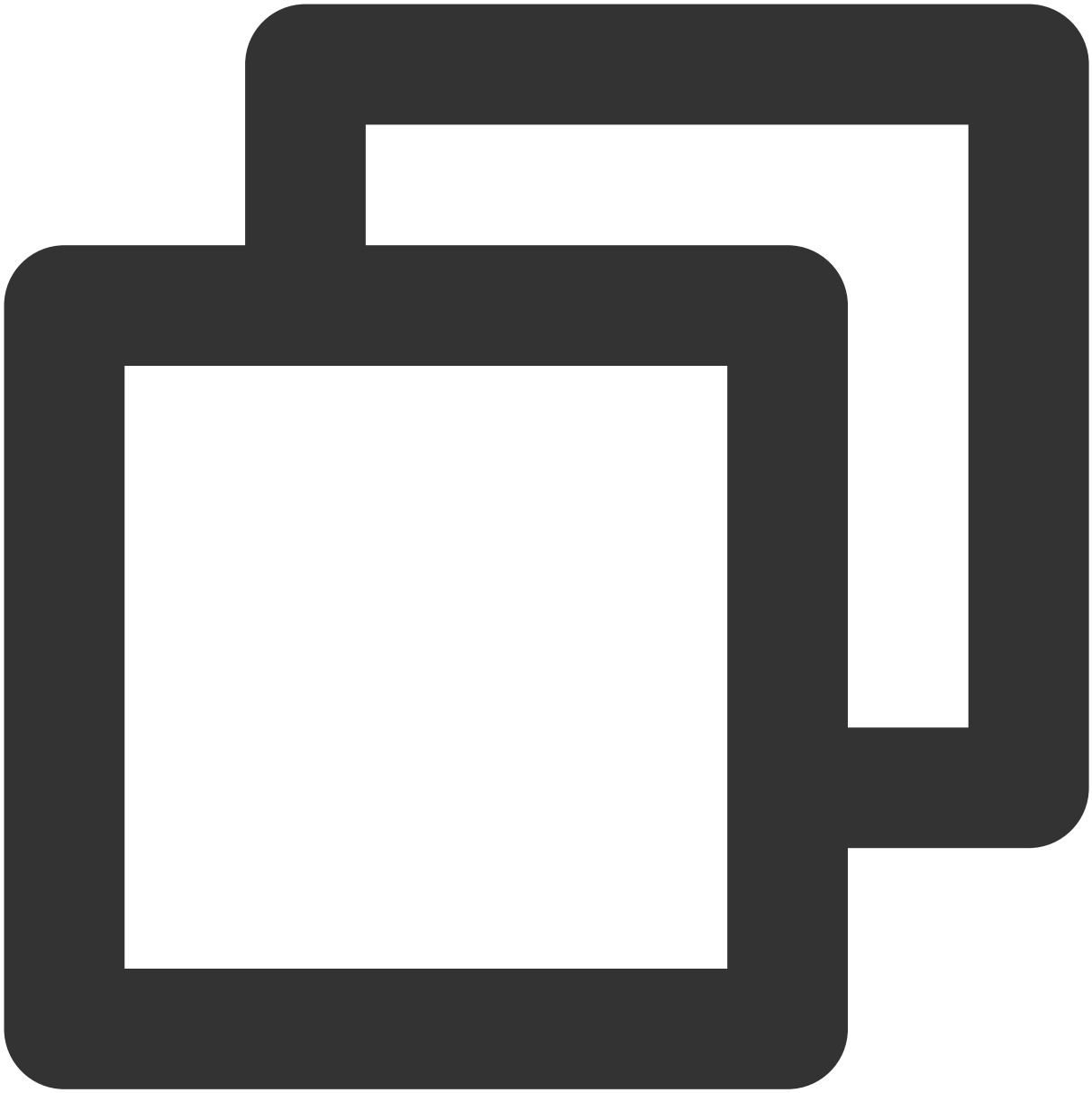
Note：

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

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

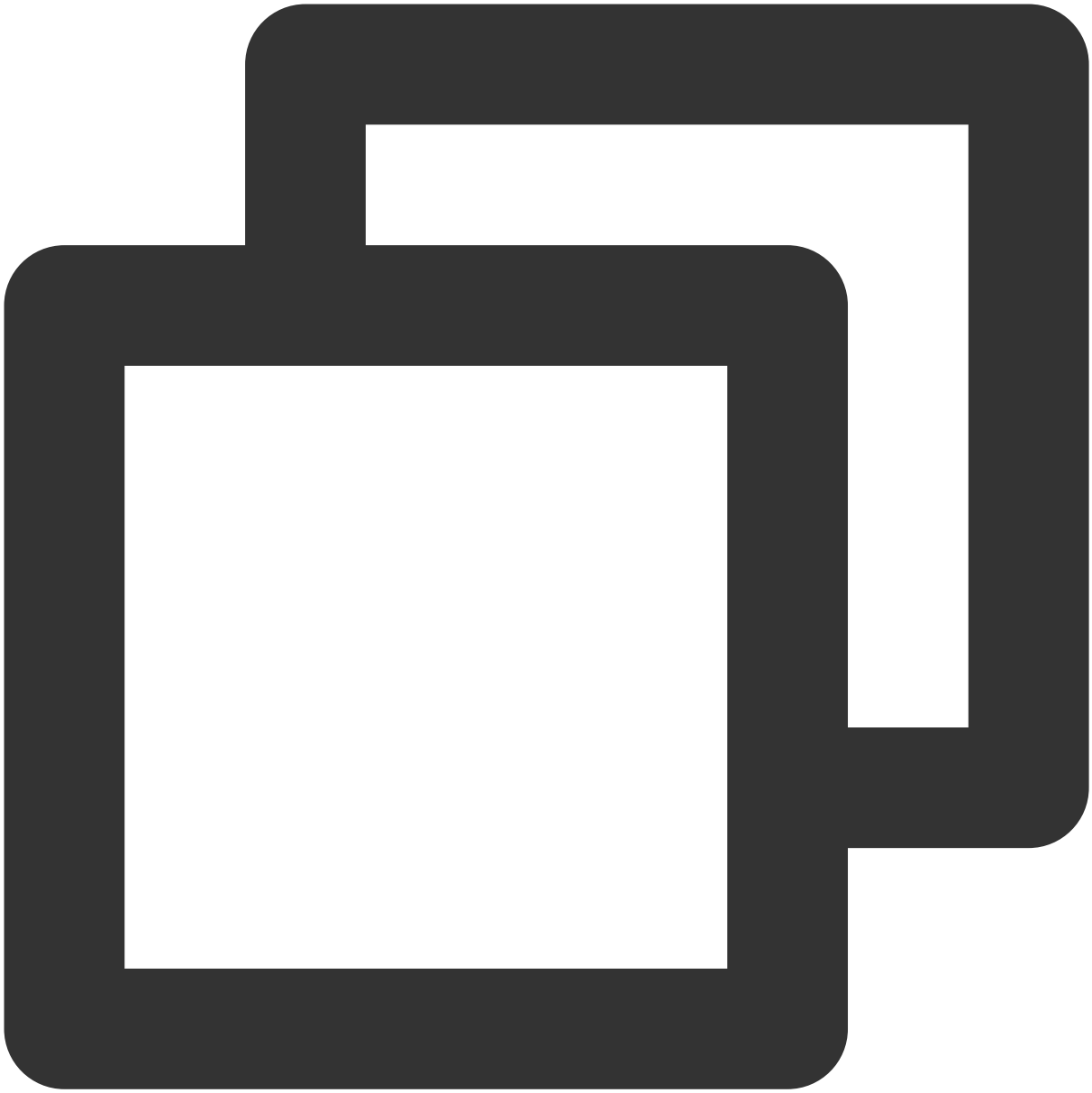
java

swift



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

}

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

```
}

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

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

        return 0;
    }

    @Override
    public void onGLContextCreated() {
    }

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



```
let videoFrameListener: TRTCVideoFrameListener = TRTCVideoFrameListener()

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

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

class TRTCVideoFrameListener: NSObject, TRTCVideoFrameDelegate {
```

```
    let bueutyModel = XXXXBeautyModel.shareIntance()
    func onProcessVideoFrame(_ srcFrame: TRTCVideoFrame, dstFrame: TRTCVideoFrame)
        // Beauty processing logic
        bueutyModel.onProcessVideoFrame(srcFrame, dstFrame)
        ……

        return 0
    }
}
```

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

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

Integrating Tencent Beauty Effects

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

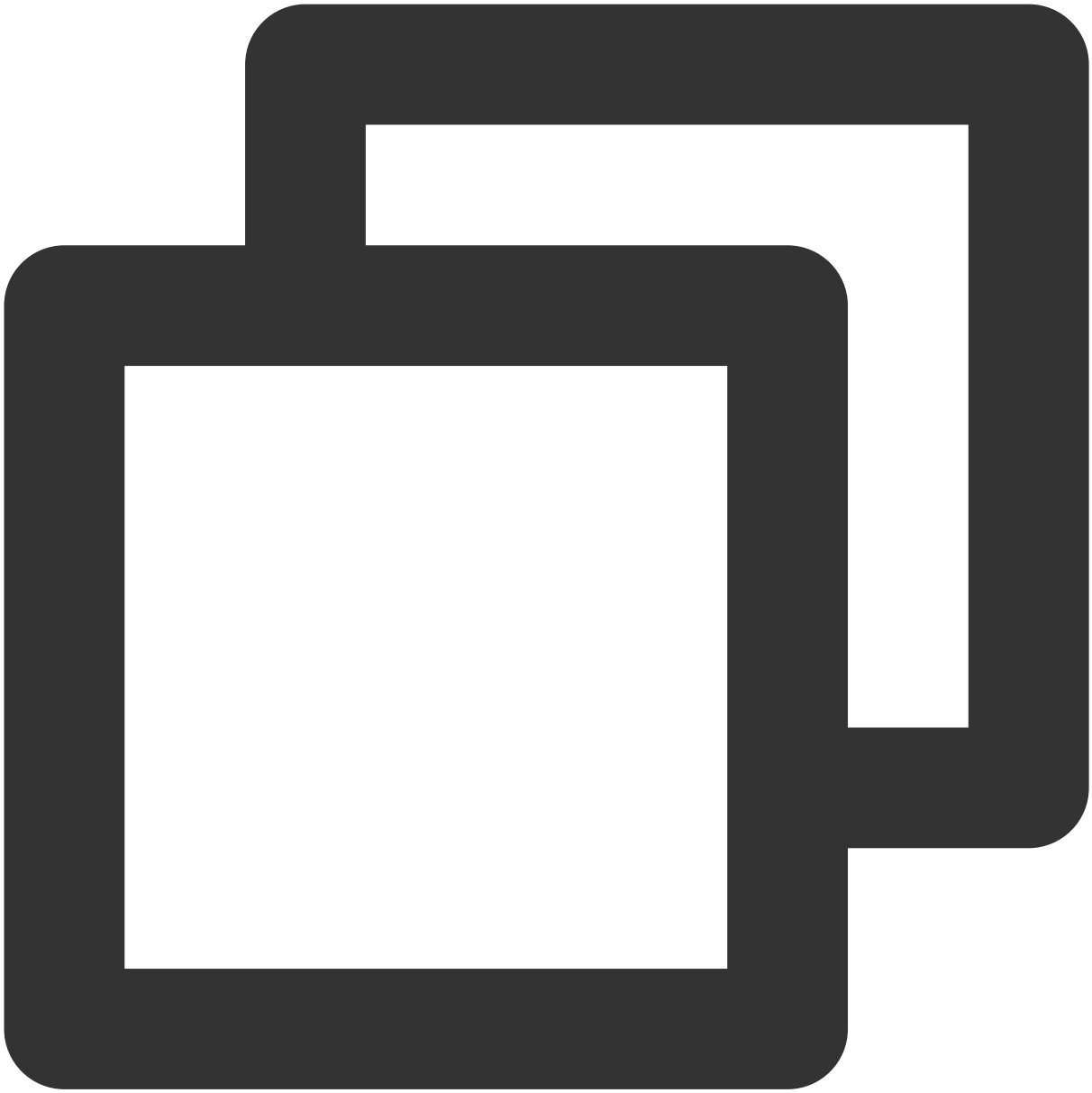
Step 1: Beauty resource download and integration

1. Download the SDK according to the package you purchased.
2. Add files to your project:

Android

iOS

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

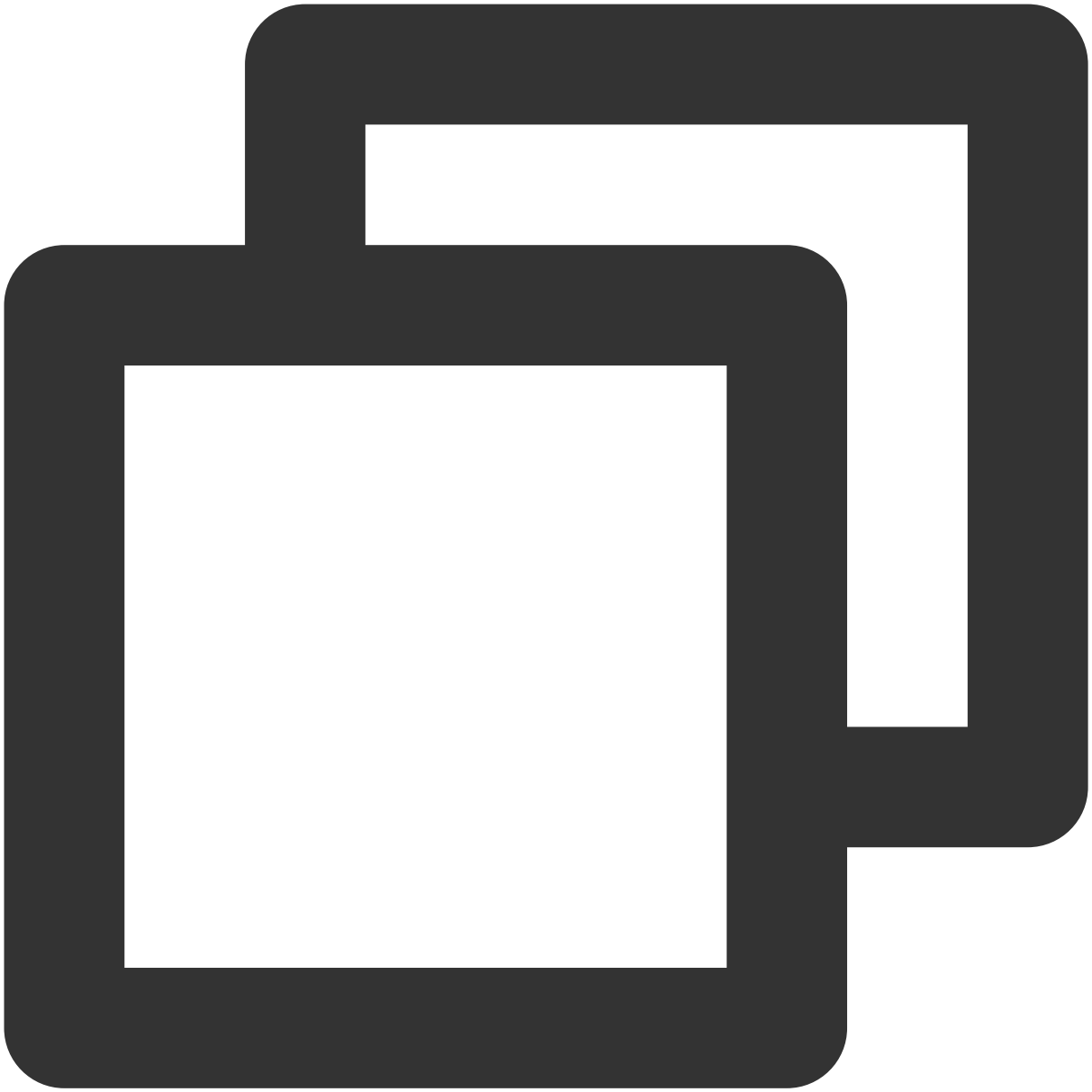


```
dependencies {
   implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

For the maven addresses corresponding to each package, please refer to the documentation.

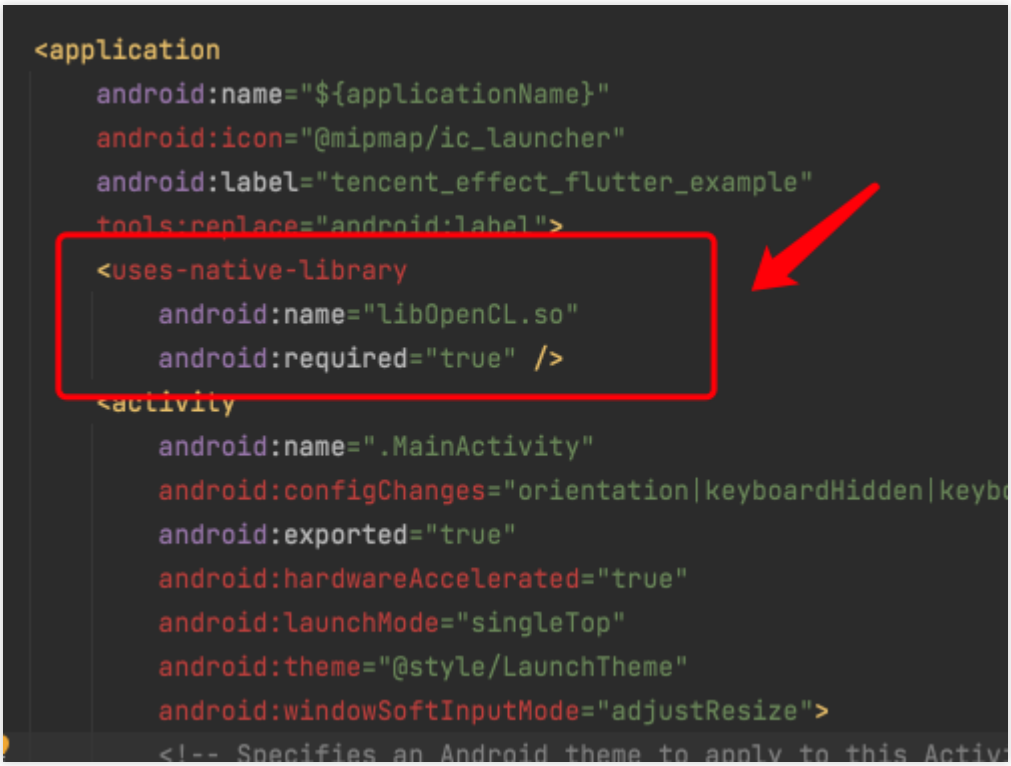
2. Find the src/main/assets folder under the app module. If it doesn't exist, create it. Check if there is a MotionRes folder in the downloaded SDK package. If so, copy this folder to the ../src/main/assets directory.

3. Find the AndroidManifest.xml file under the app module and add the following tag in the application form



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

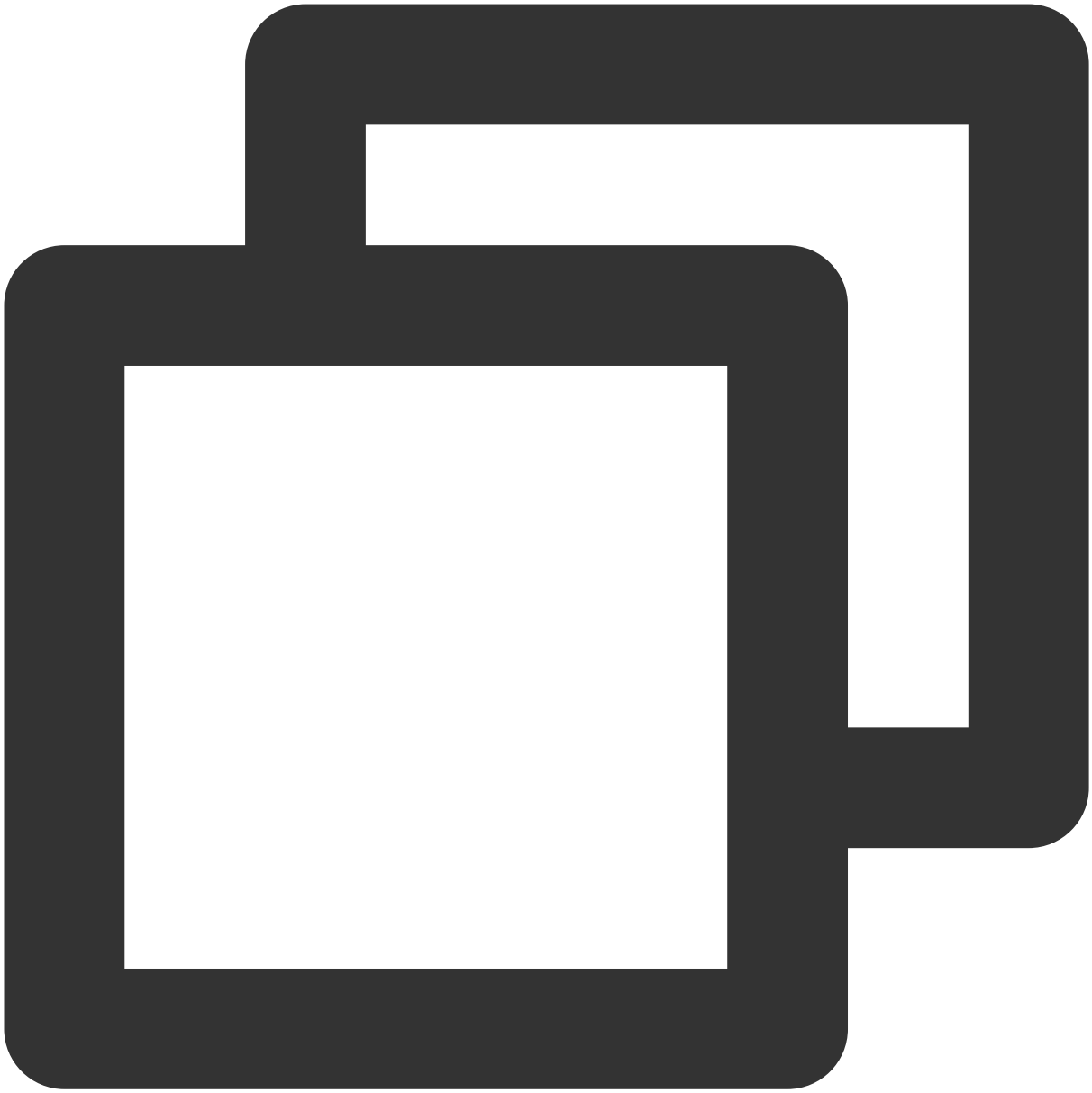
Add as shown in the following figure:



4. Obfuscation configuration

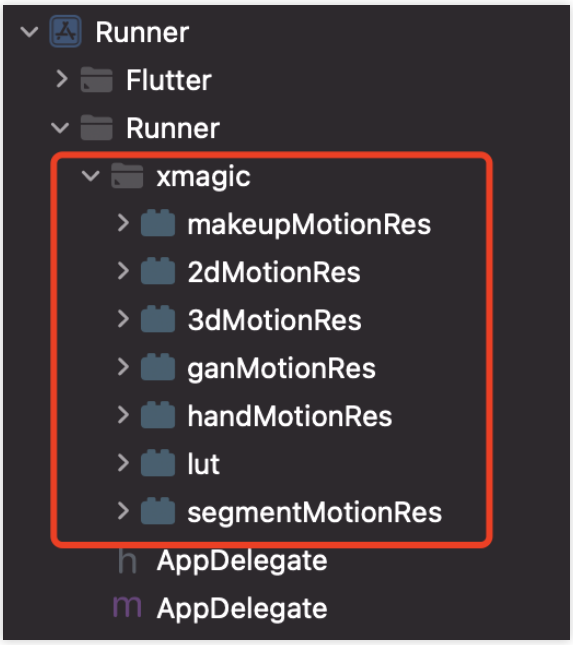
If you enable compile optimization when building a release package (set minifyEnabled to true), some code that is not called in the java layer will be cut off, and this code may be called by the native layer, causing a no xxx method exception.

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



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

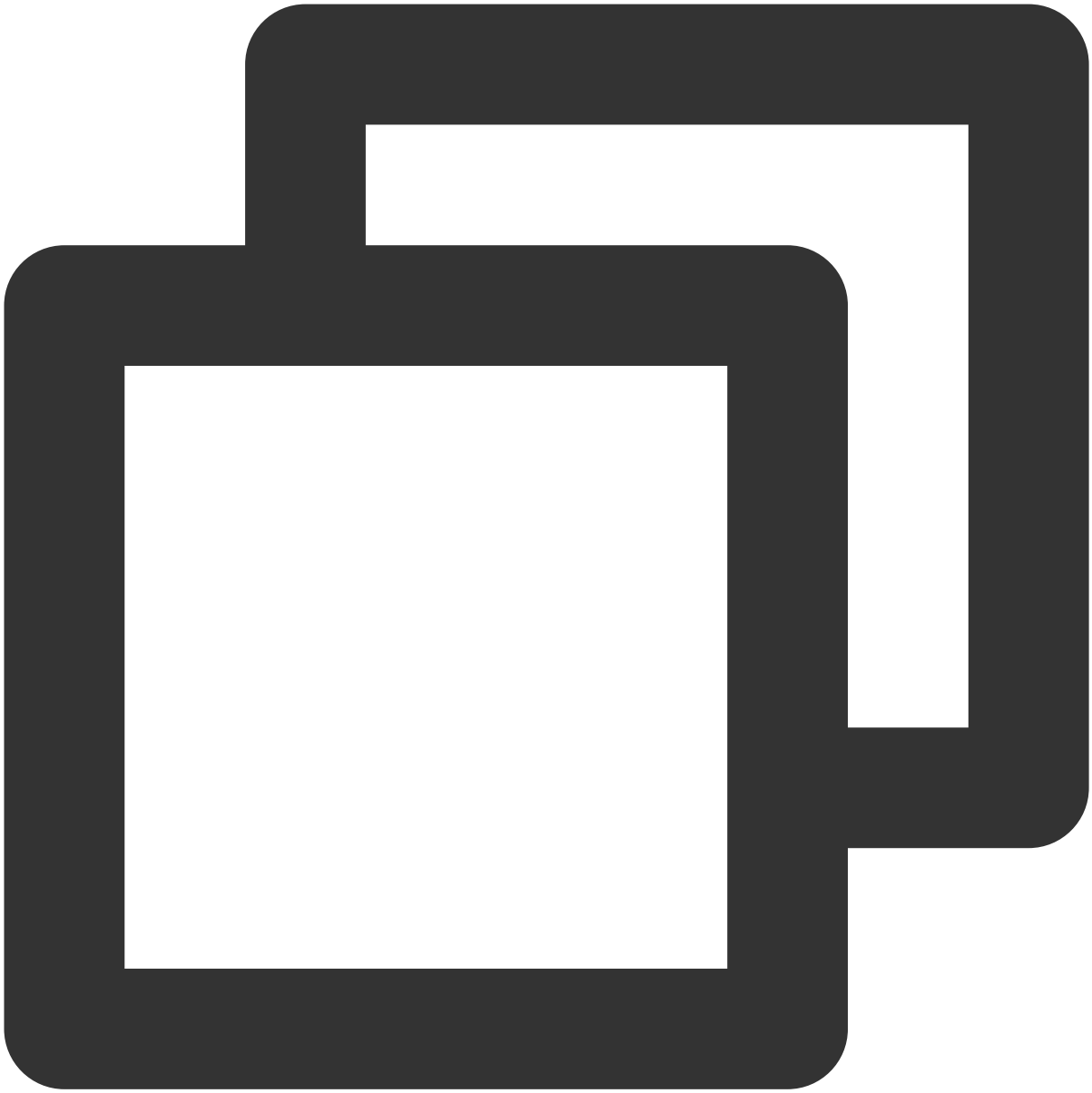
1. Add beauty resources to your project. After adding, it will be shown as in the following figure (the types of resources you have may not be exactly the same as in the figure):



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

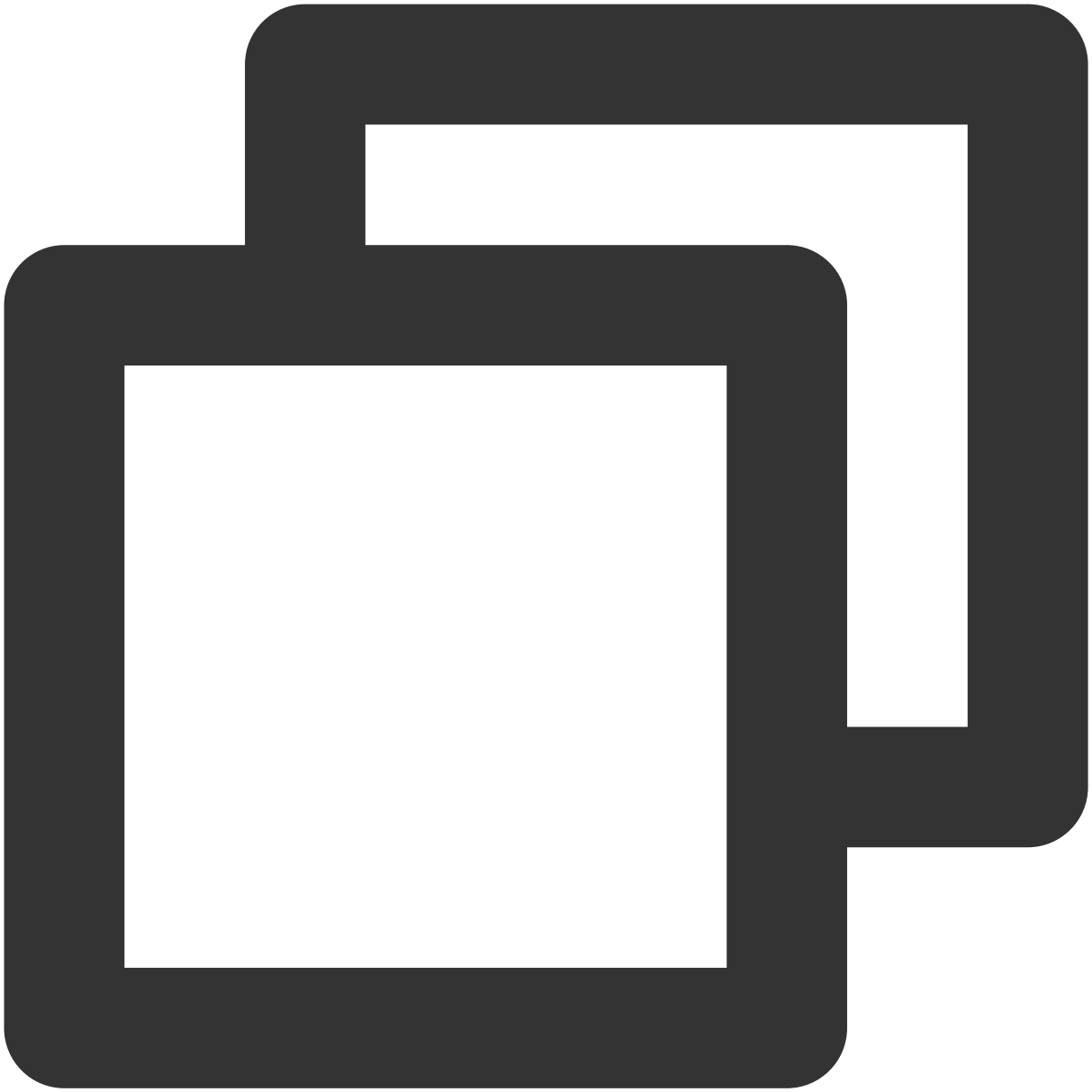
Step 2: Reference the Flutter version SDK

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



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

Local reference: Download the latest version of tencent_effect_flutter from tencent_effect_flutter, and then add the folders android, ios, lib, and the files pubspec.yaml, tencent_effect_flutter.iml to the project directory. Then, add the following reference to the project's pubspec.yaml file (refer to the demo):



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

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

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

Android

iOS

Execute the command `flutter pub upgrade` in the project directory, or click "`Pub upgrade`" in the upper right corner of the subspec.yaml page.
Execute the command `flutter pub upgrade` in the project directory, and then execute the command `pod update` in the iOS directory.

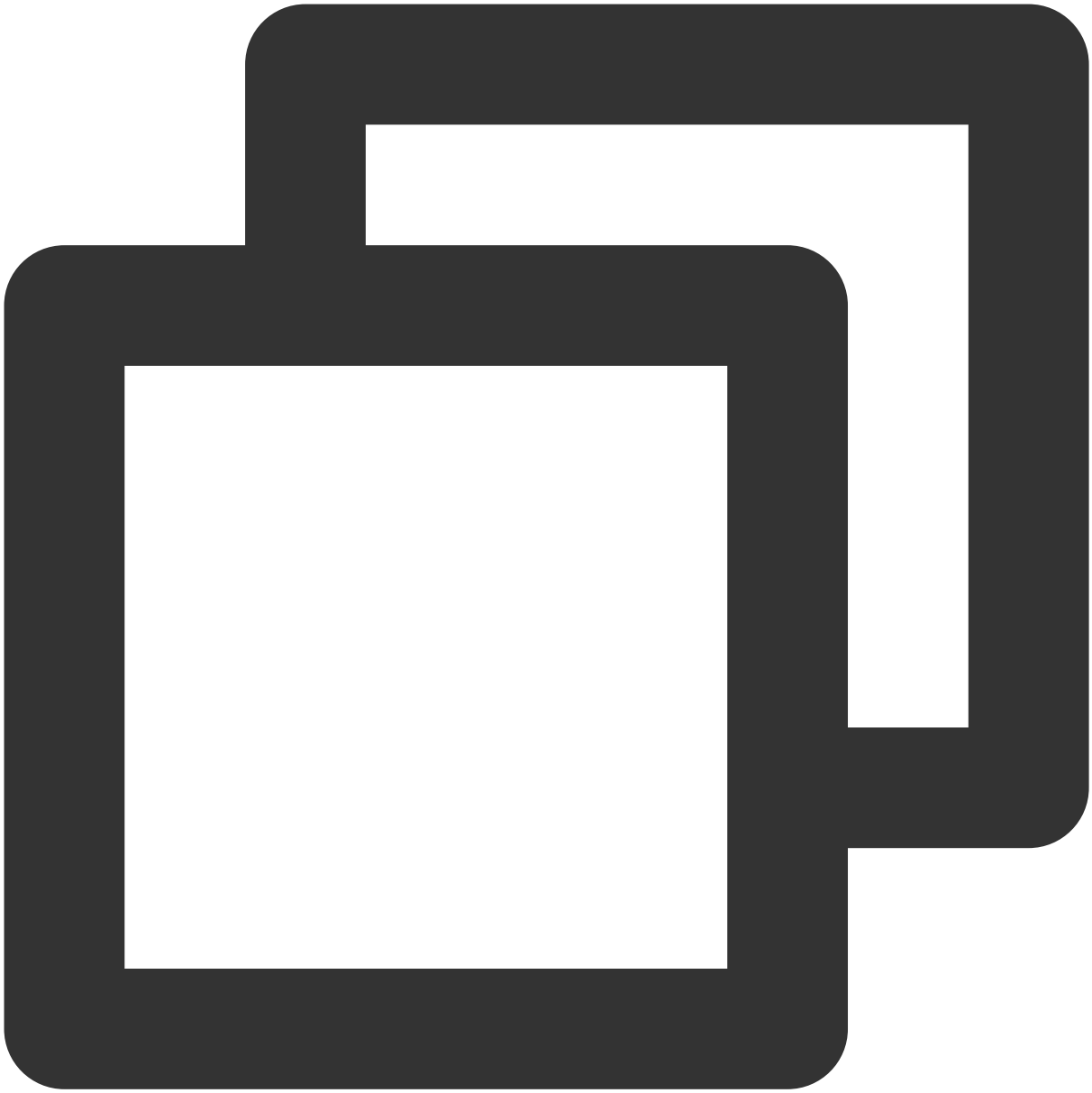
Step 3: Implement the Dart layer to Native beauty control interface start/end

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

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

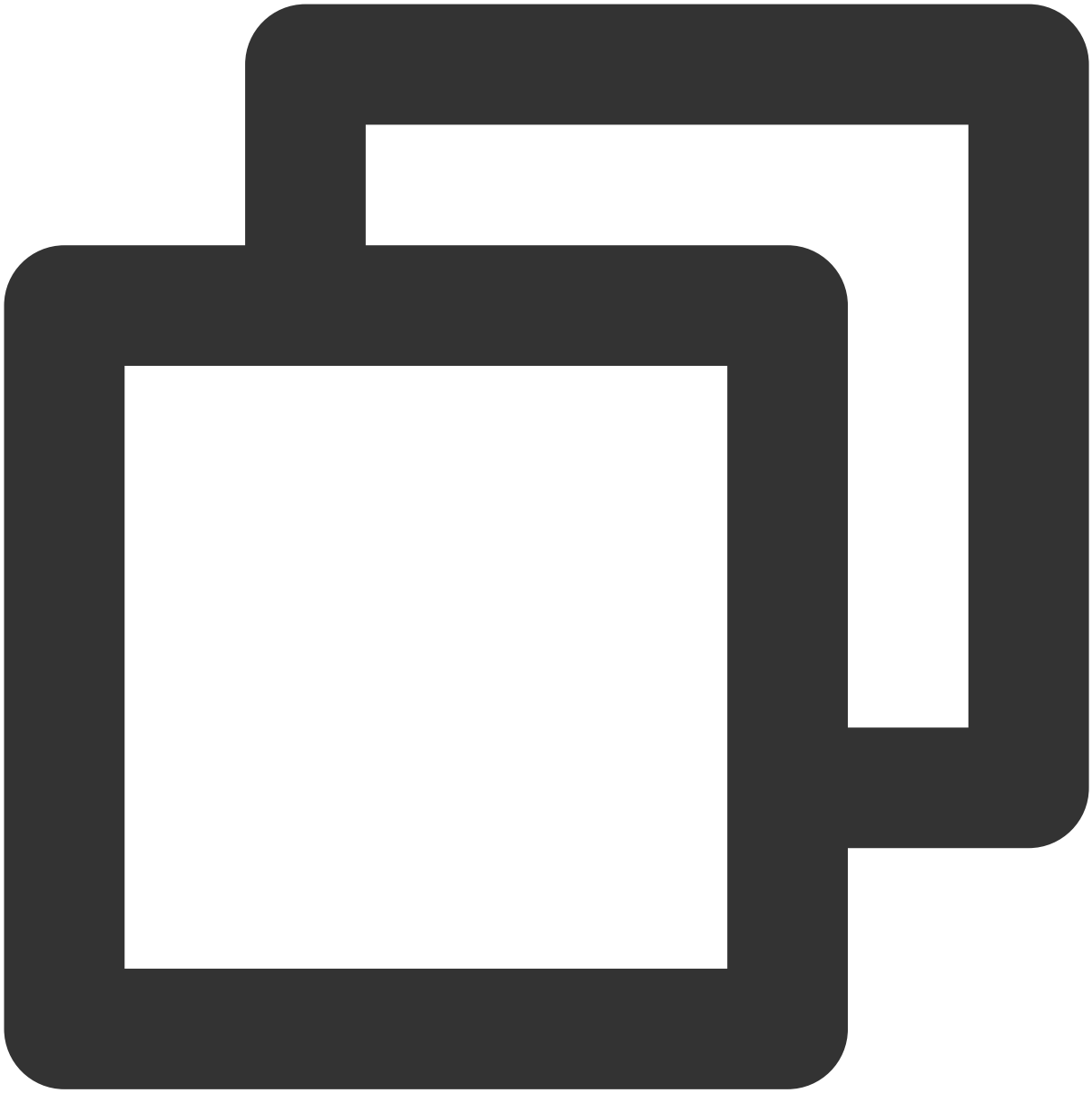
Android
iOS
Android needs to rely on `LiteAVSDK_Professional` first when accessing beauty. Add the following dependencies in the `app/build.gradle` of the Android project:

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



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

}

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

```
}

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

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

    @Override
    public void onGLContextCreated() {
    }

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



```
import TXLiteAVSDK_Professional
import tencent_effect_flutter

let videoFrameListener: TRTCVideoFrameListener = TRTCVideoFrameListener()

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

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

```
}

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


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

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

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

Step 5: Enable beauty and set beauty parameters

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

Custom Ringtone
Android

Last updated：2024-05-24 18:35:12

This article introduces how to replace the incoming call ringtone in TUICallKit. The incoming call ringtone includes **application ringtone** and **offline push ringtone**.

Setting the application ringtone

There are two methods to set the application ringtone: replace ringtone audio and call `setCallingBell` interface.You can also disable the application ringtone.

1. Replace Audio File

If you integrate the TUICallKit component via source code dependency, you can replace the audio files in the tuicallkit-kt/src/main/res/raw folder to customize the ringtone.

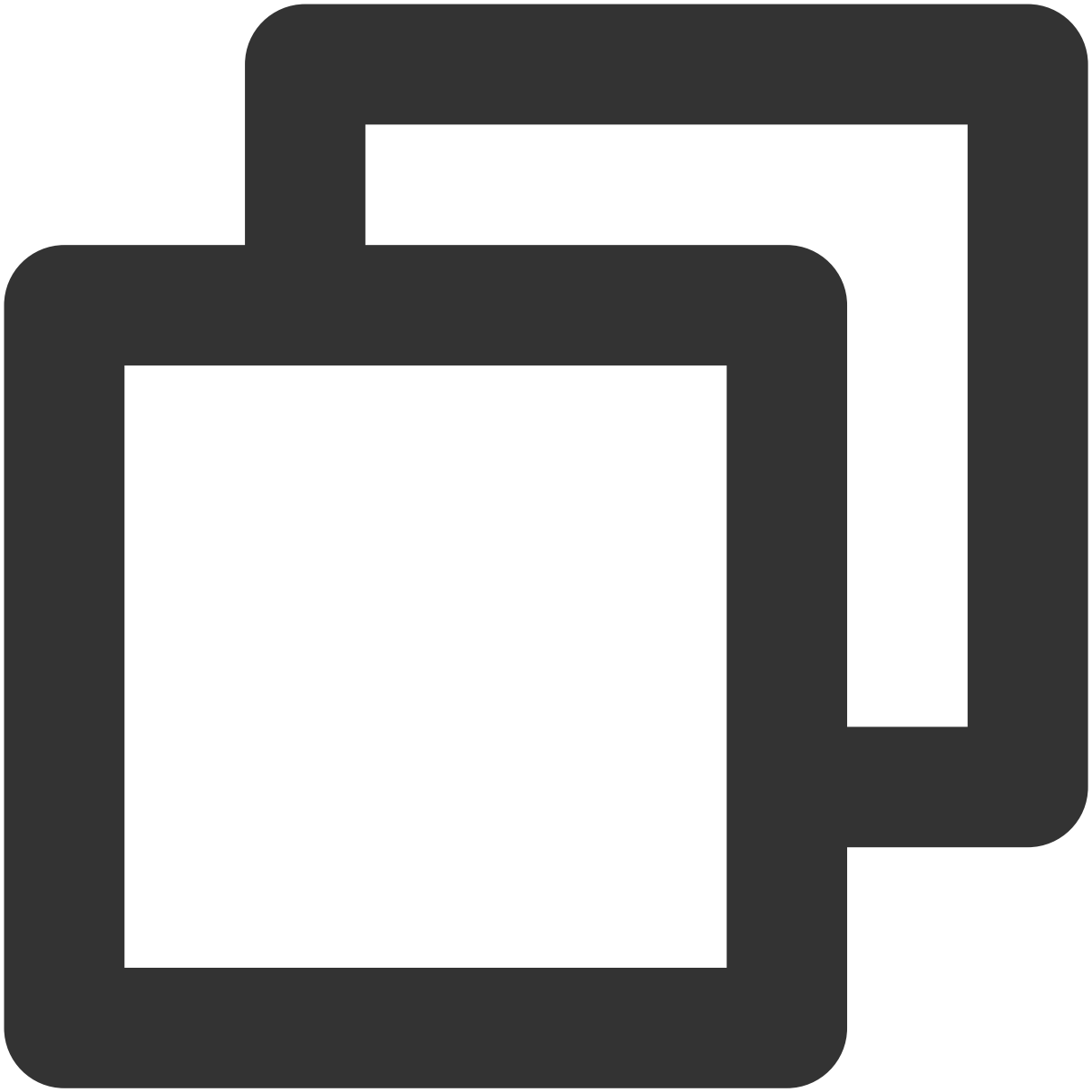
File Name	Use
phone_dialing.mp3	Ringtone when initiating a call
phone_hangup.mp3	Ringtone when the call is disconnected
phone_ringing.mp3	Ringtone when receiving a call

2. Call setCallingBell Interface

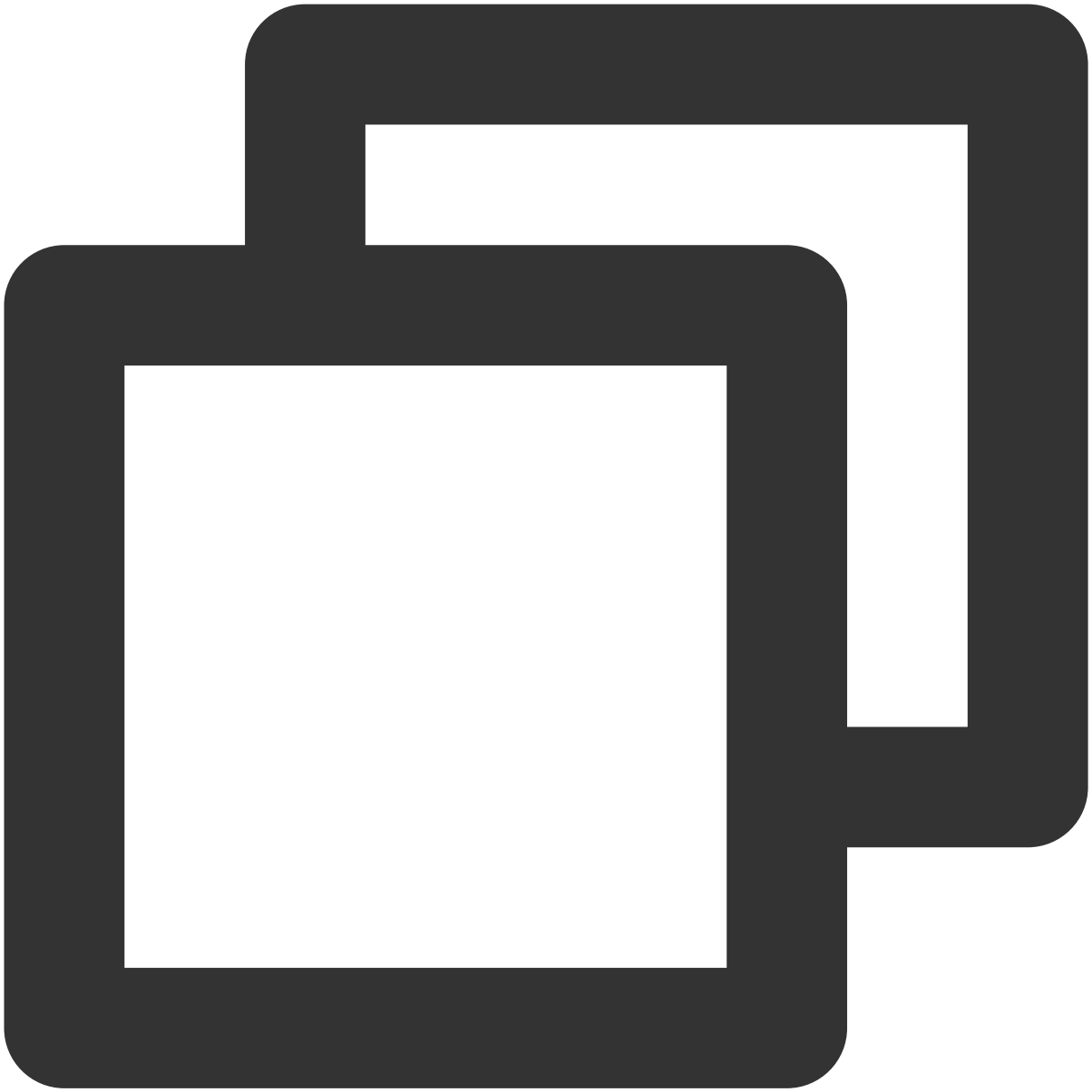
You can also customize the incoming call ringtone through the setCallingBell interface.

Kotlin

Java



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



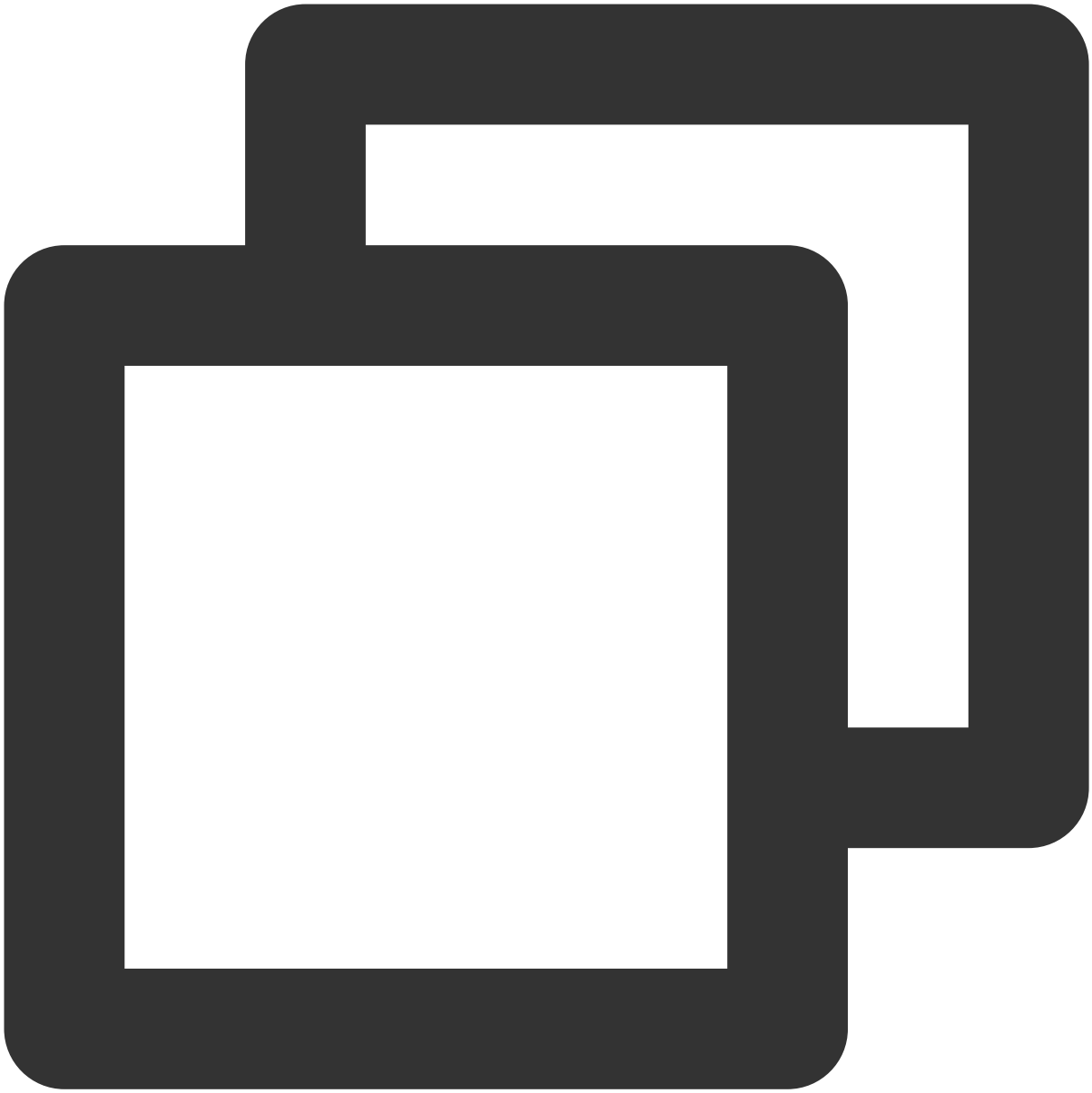
```
TUICallKit.createInstance(context).setCallingBell(filePath);
```

3. Set Mute Mode

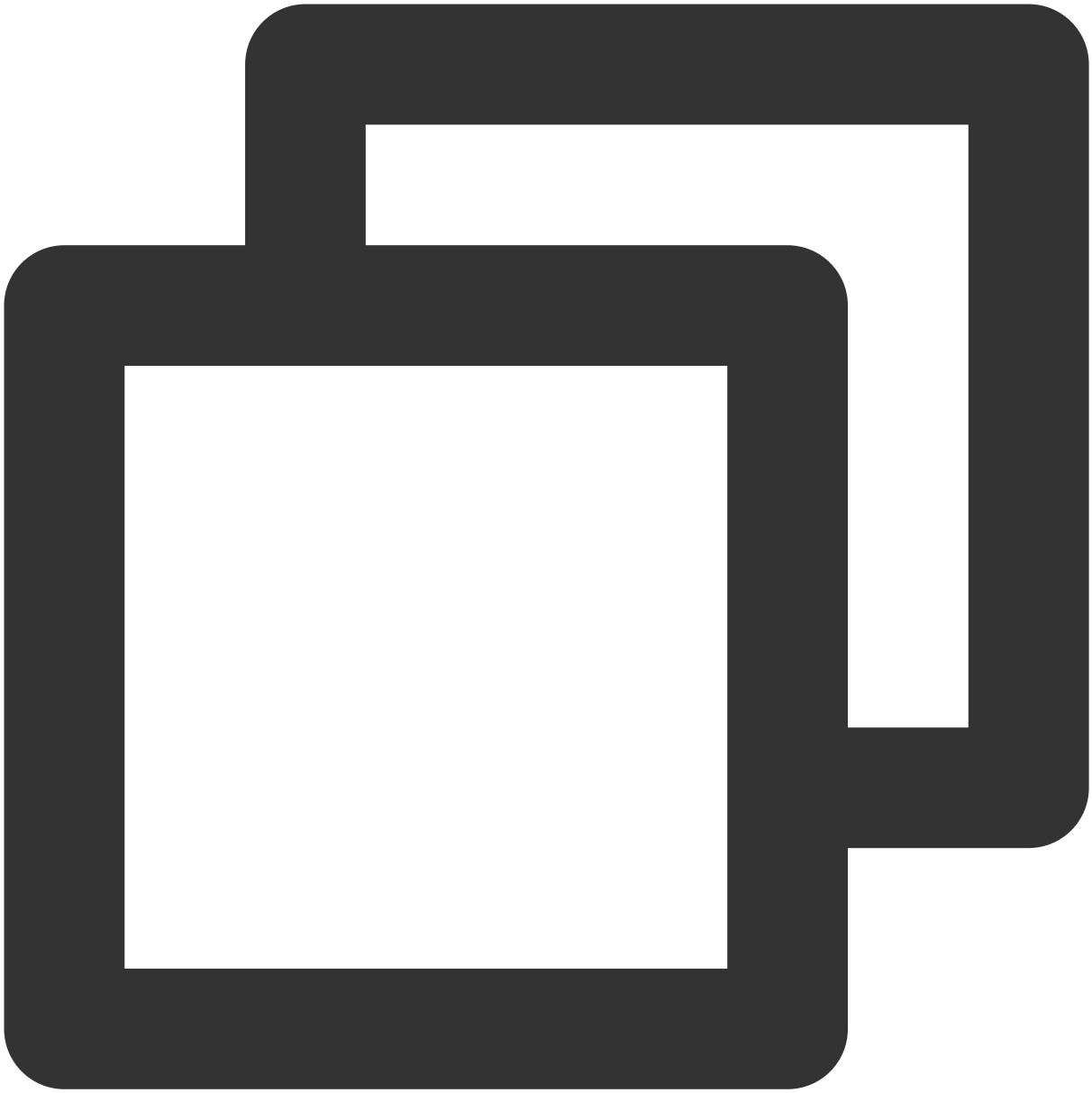
If you do not need the phone to ring, you can enable the mute mode using the enableMuteMode interface.

Kotlin

Java



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



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

Setting Offline Push Ringtone

For Offline Push Ringtone Settings, please refer to: FCM Offline Push customizes incoming call ringtone.

iOS

Last updated：2024-05-24 18:35:12

This article explains how to replace the incoming call ringtone for TUICallKit, which includes **application ringtone** and **offline push ringtone**.

Setting application Ringtone

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

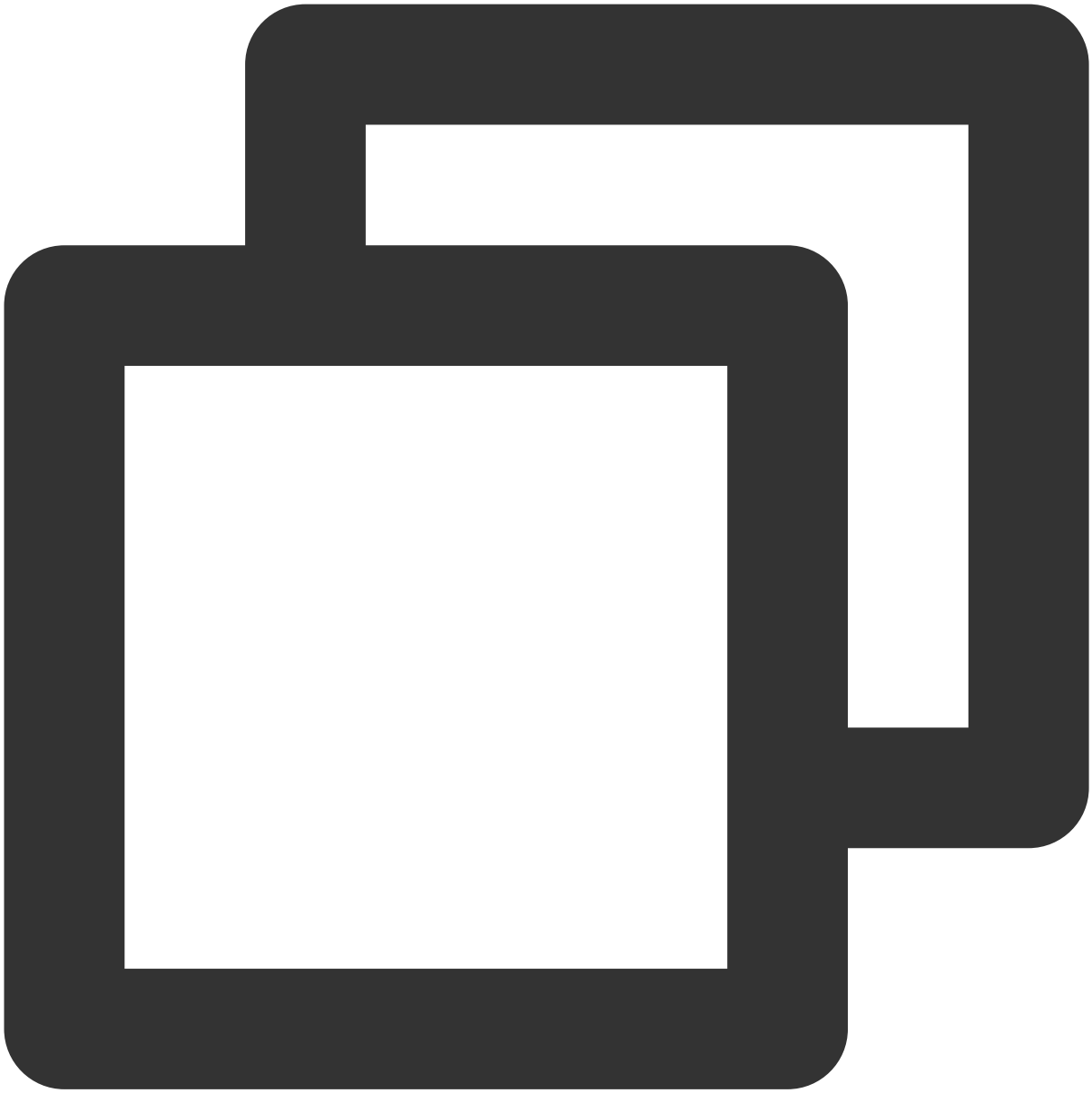
1. Replace Audio File

If you integrate the TUICallKit component via source code dependency, you can achieve the goal of replacing the ringtone by swapping out the three audio files under the `Resources\\AudioFile` folder:

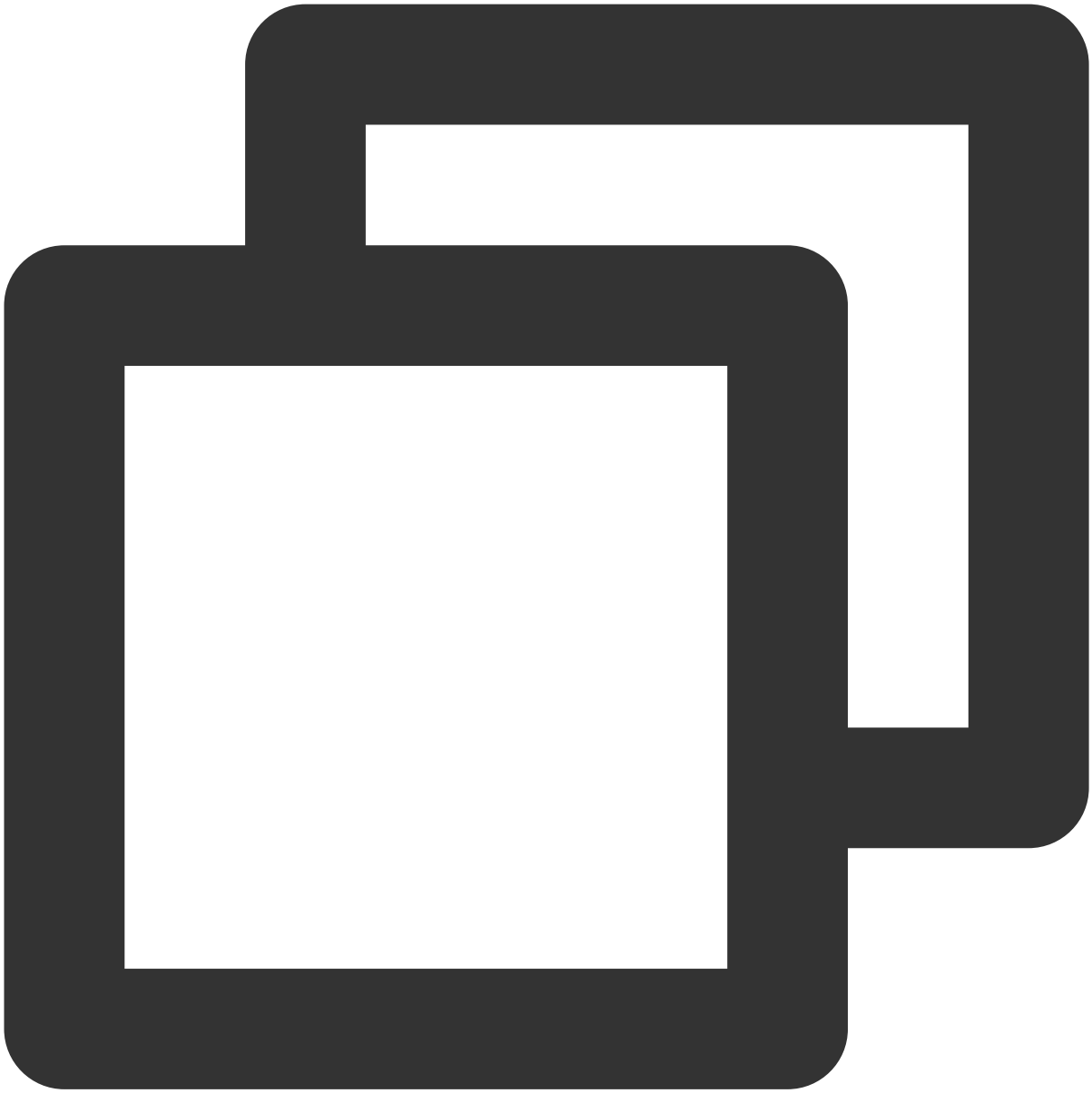
File Name	Use
phone_dialing.mp3	Ringtone when initiating a call
phone_hangup.mp3	Ringtone when the call is disconnected
phone_ringing.mp3	Ringtone when receiving a call

2. Set Ringtone Interface

You can also set the incoming call ringtone via the setCallingBell interface.
Swift
Objective-C



```
import TUICallKit_Swift

TUICallKit.createInstance().setCallingBell(filePath: "")
```



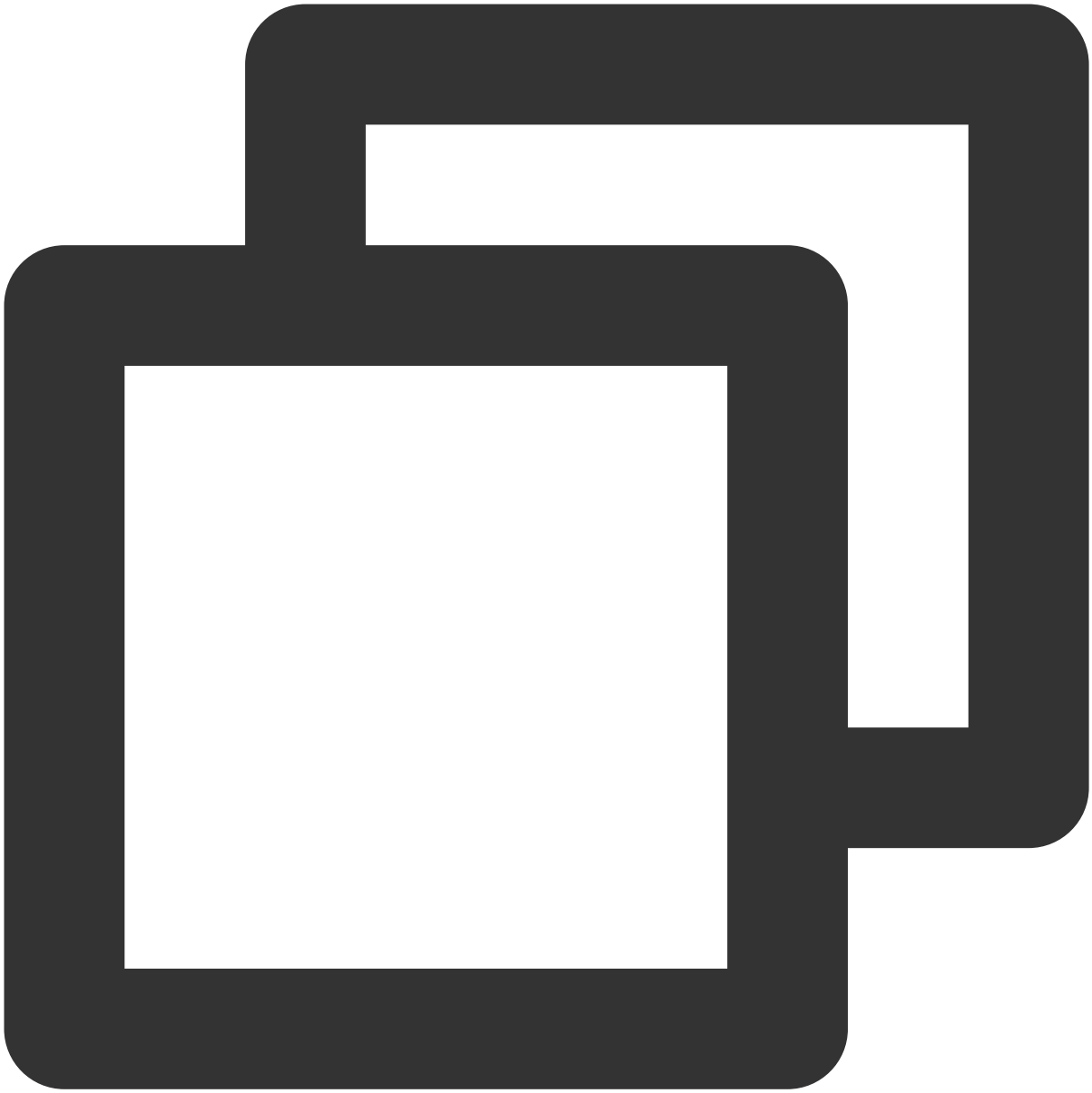
```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

[[TUICallKit createInstance] setCallingBellWithFilePath:@""];
```

3. Setting Mute Mode

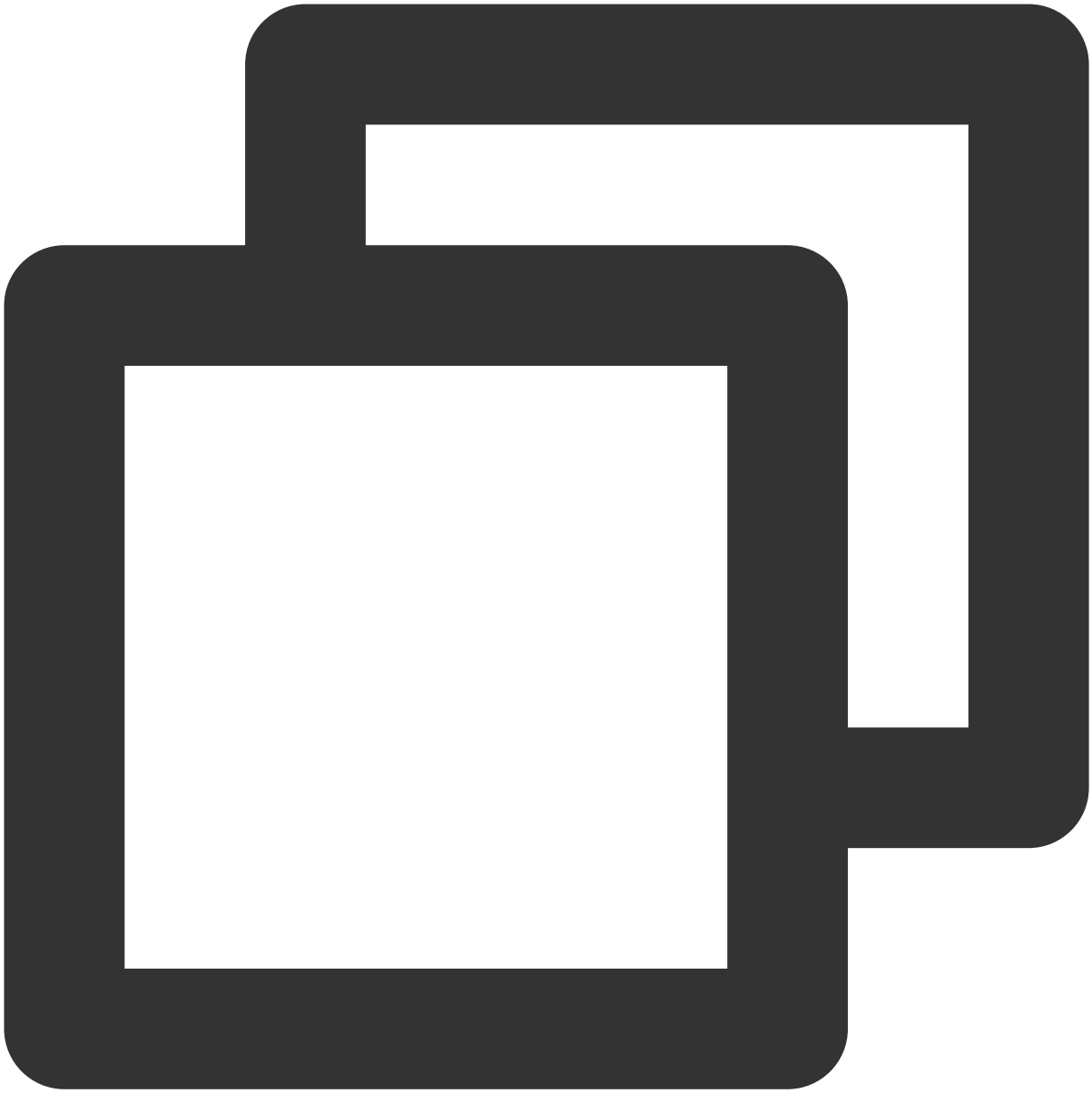
If you do not require a ringtone, you can set the mute mode via enableMuteMode.

Swift

Objective-C



```
import TUICallKit_Swift

TUICallKit.createInstance().enableMuteMode(enable: true)
```



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

[[TUICallKit createInstance] enableMuteModeWithEnable:YES];
```

Setting Offline Push Ringtone

VoIP push does not support custom push ringtones. APNs push can be set by specifying the iOSSound field in the offlinePushInfo params when making a call via the Call Interface. iOSSound should be passed the audio file name.

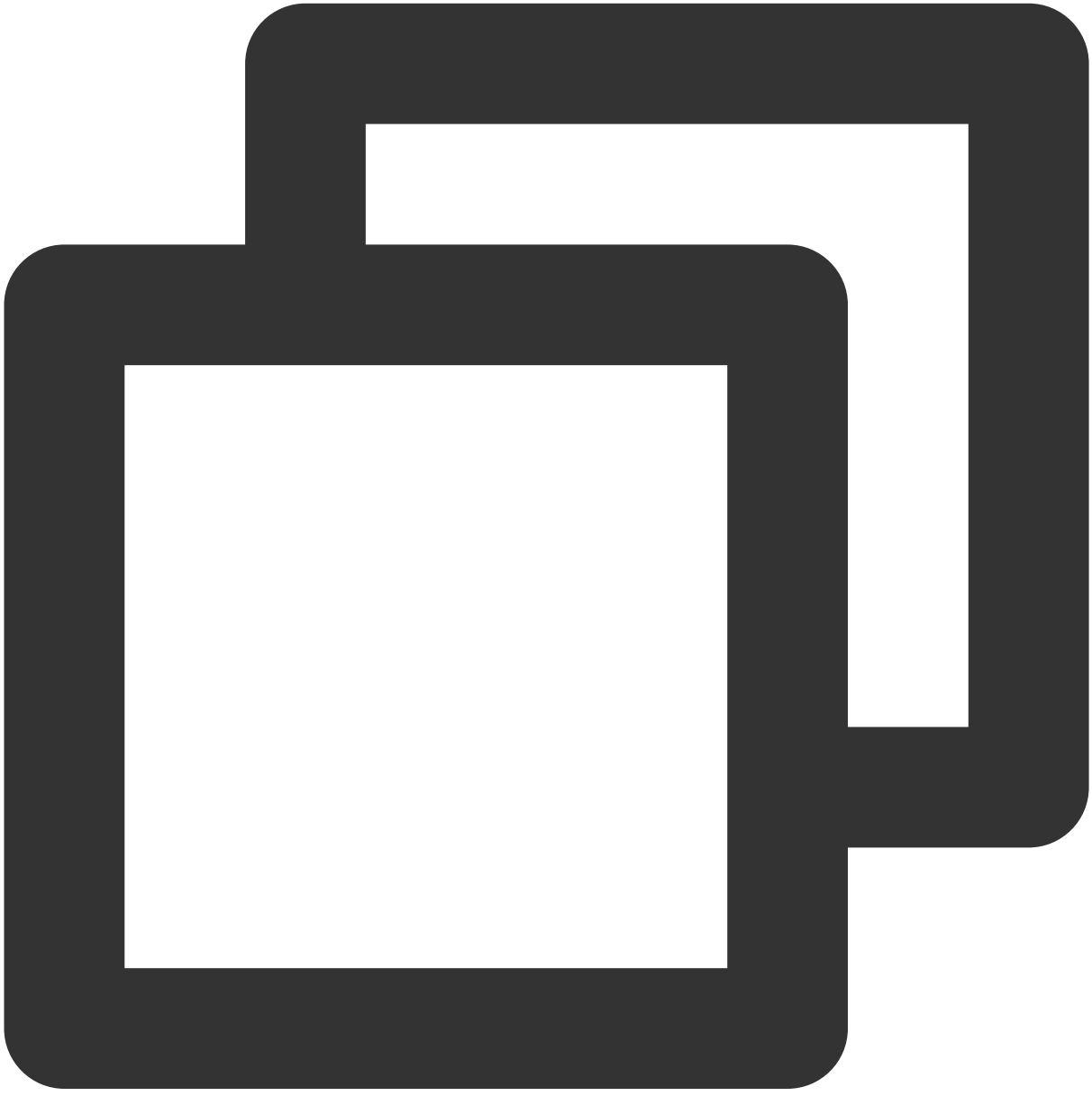
Note:

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

Ringtone duration should be less than 30s.

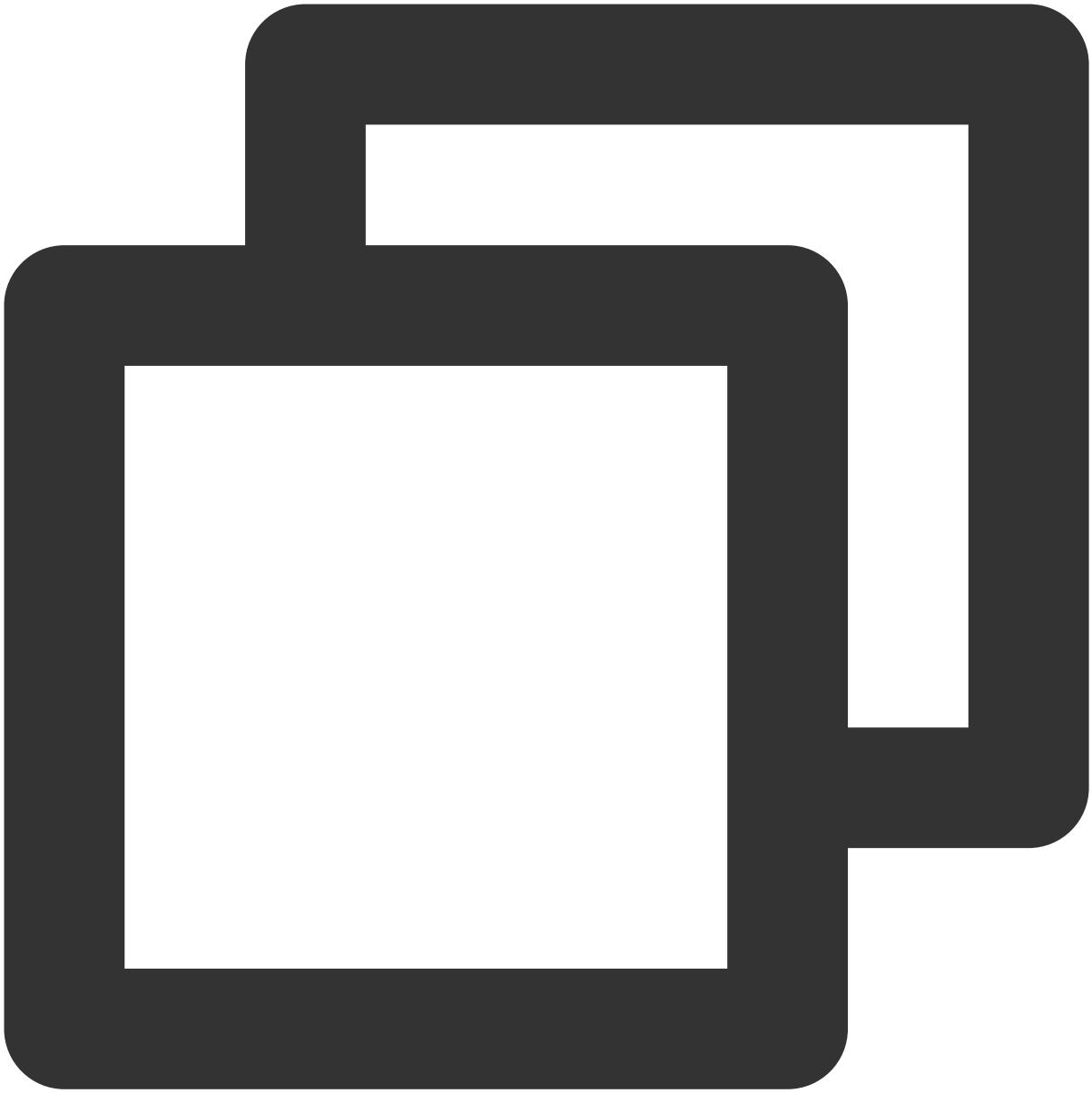
Swift

Objective-C



```
import TUICallKit_Swift
import TUICallEngine
```

```
let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = ""
pushInfo.desc = "You have a new call"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO must set a ChannelID to receive push messages. This channelID needs to be t
pushInfo.androidOPPOChannelID = "tuikit"
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HM
pushInfo.androidHuaWeiCategory = "IM"

let params = TUICallParams()
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo

TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: p

} fail: { code, message in

}
```



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <TUICallEngine/TUICallEngine.h>

- (TUICallParams *)getCallParams {
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}
```

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

[[TUICallKit createInstance] callWithUserId:@"123456"
                                callMediaType:TUICallMediaTypeAudio
                                       params:[self getCallParams] succ:^{

} fail:^(int code, NSString * _Nullable errMsg) {

}];C
```

Web&H5

Last updated：2024-05-24 18:19:55

This article introduces how to use the custom ringtone and silent incoming call ringtone feature from the definition.

Setting incoming call ringtone

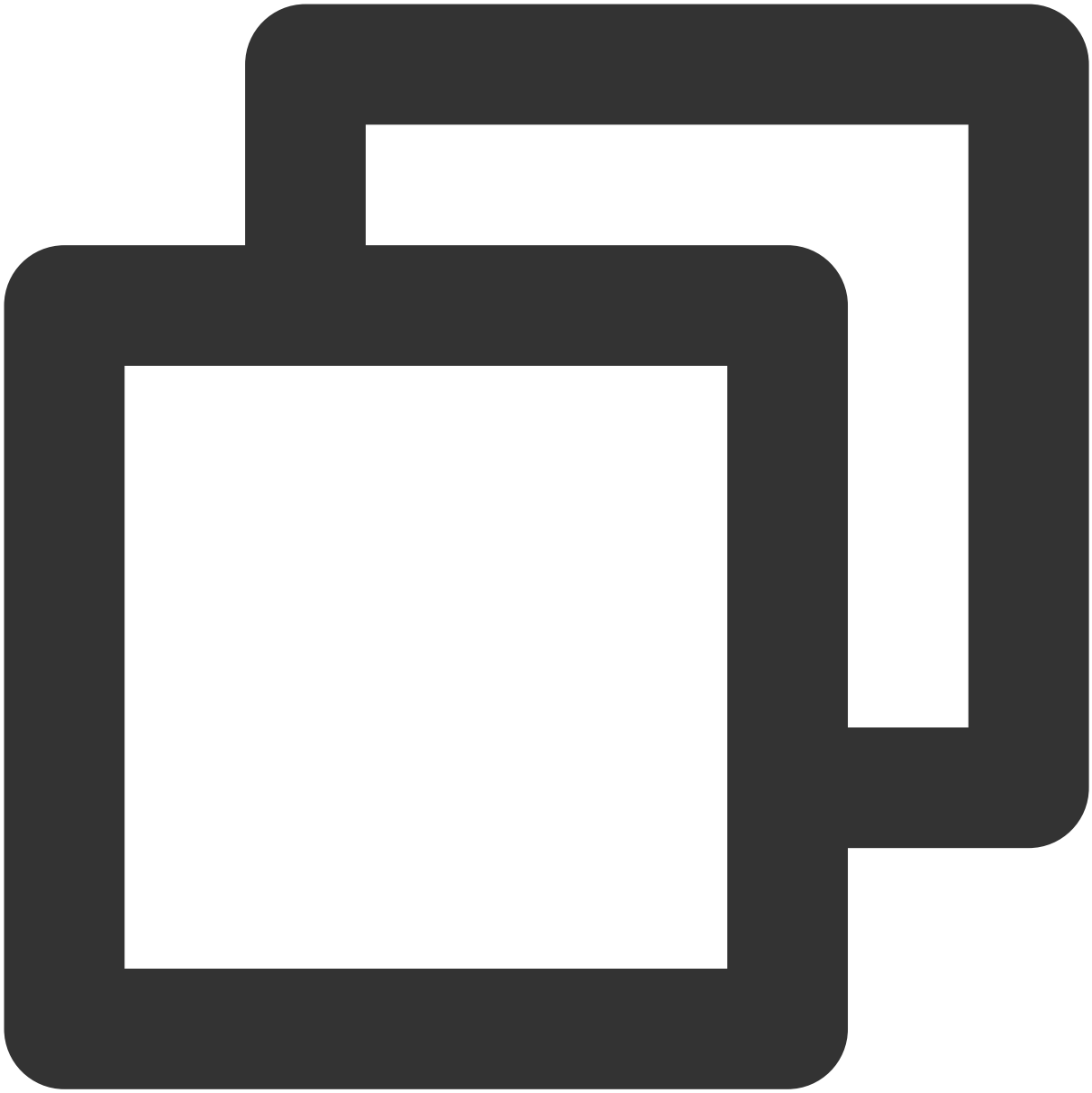
Only local MP3 format file addresses can be used, ensuring that the file is accessible.

To reset the ringtone, pass in an empty string for filePath.

Use the ES6 import method to import the ringtone file.

Note:

v3.0.0+ supported.



```
import filePath from '../public/ring.mp3';

try {
  await TUICallKitServer.setCallingBell(filePath?: string);
} catch (error: any) {
  alert(`[TUICallKit] setCallingBell API failed. Reason: ${error}`);
}
```

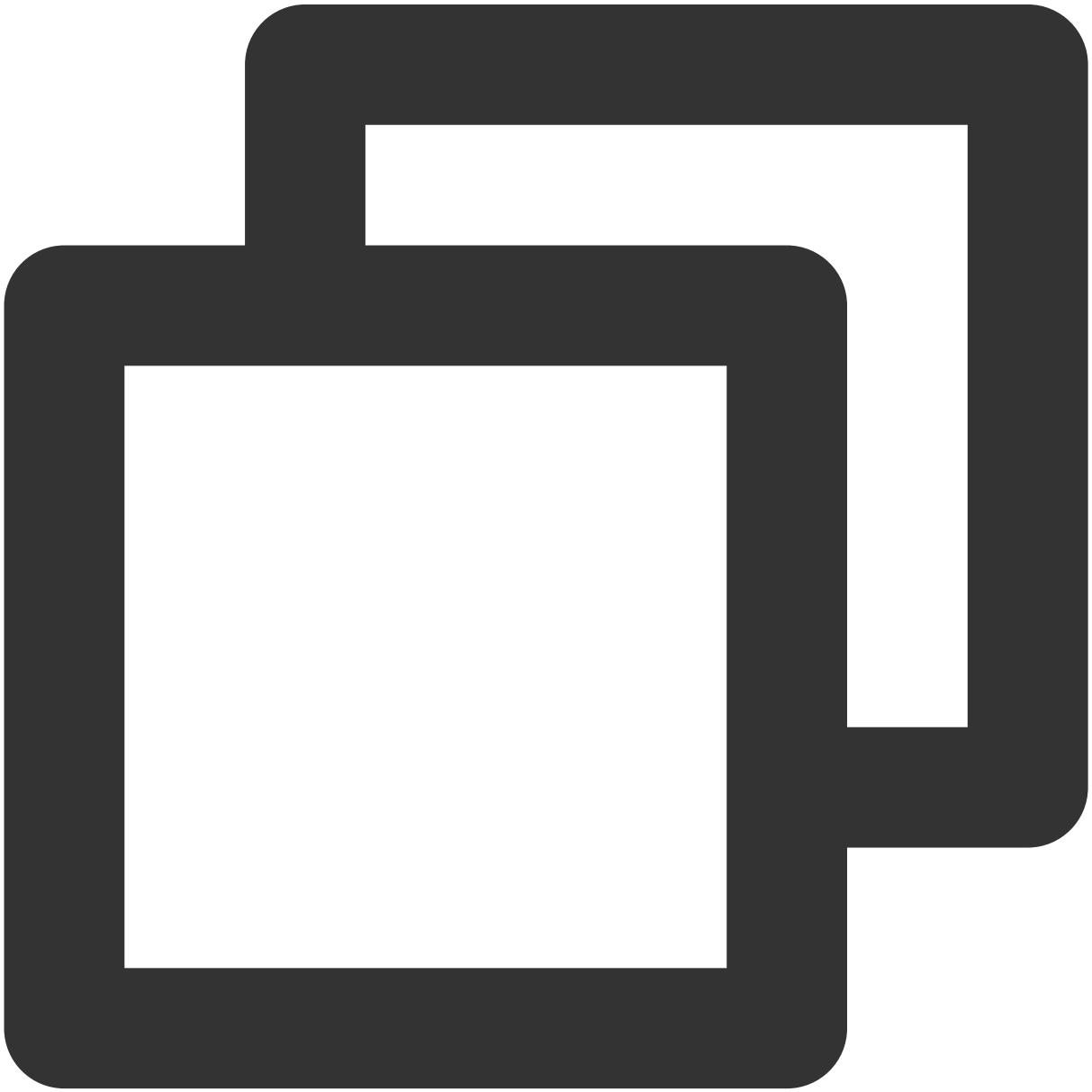
Silent incoming call ringtone

Enable/Disable incoming call ringtone.

After enabling, the incoming call ringtone will not be played when a call request is received.

Note:

v3.1.2+ supported.



```
try {
  await TUICallKitServer.enableMuteMode(enable: boolean);
} catch (error: any) {
```

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

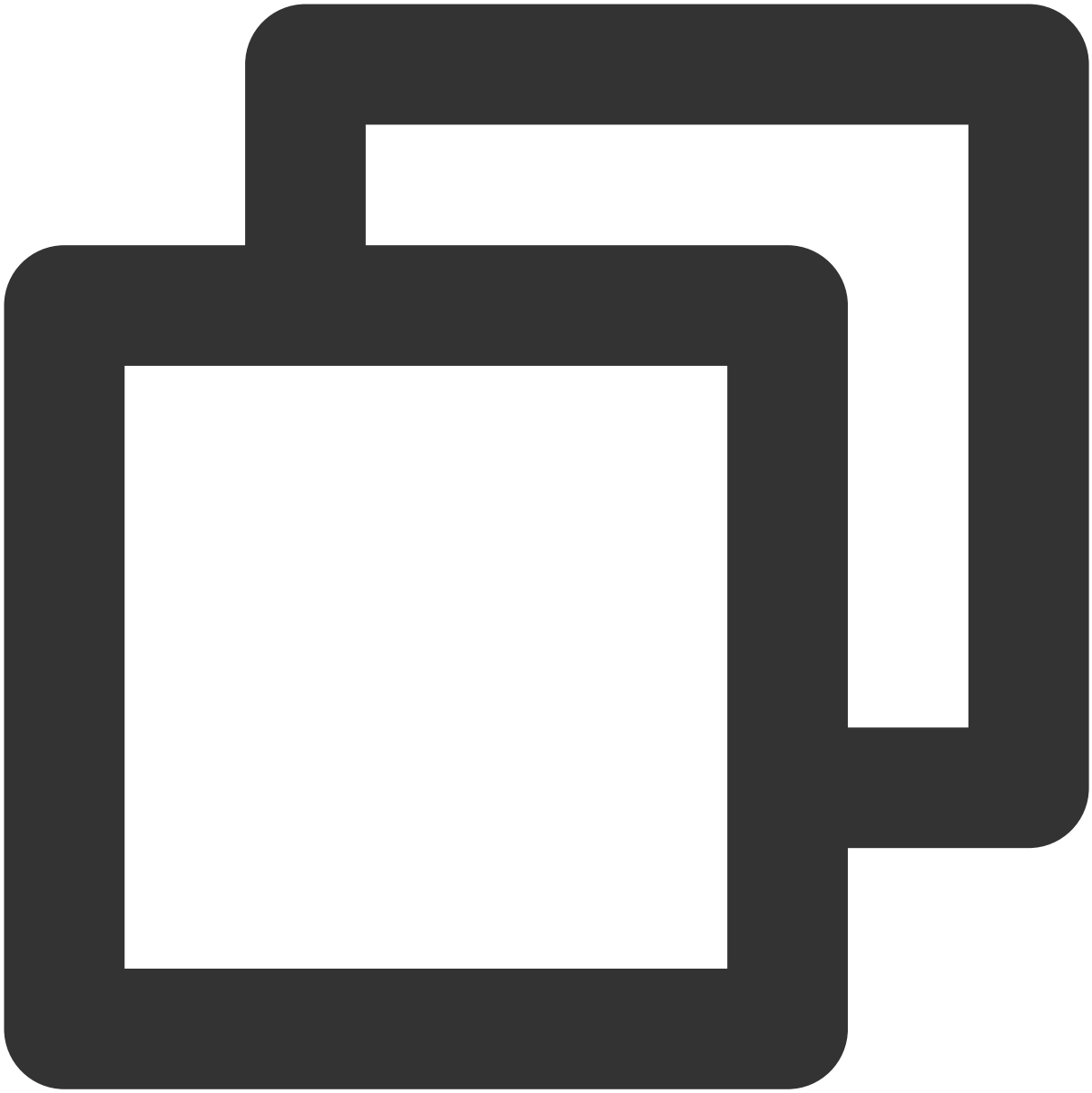
uni-app (Anroid&iOS)

Last updated：2024-04-03 17:23:11

This article introduces how to use the custom ringtone and silent incoming call ringtone feature from the definition.

Customize Incoming Call Ringtone

Setting Custom incoming call ringtone, here **only local file addresses can be passed in**, **it is required to ensure the file directory is accessible by the application**.



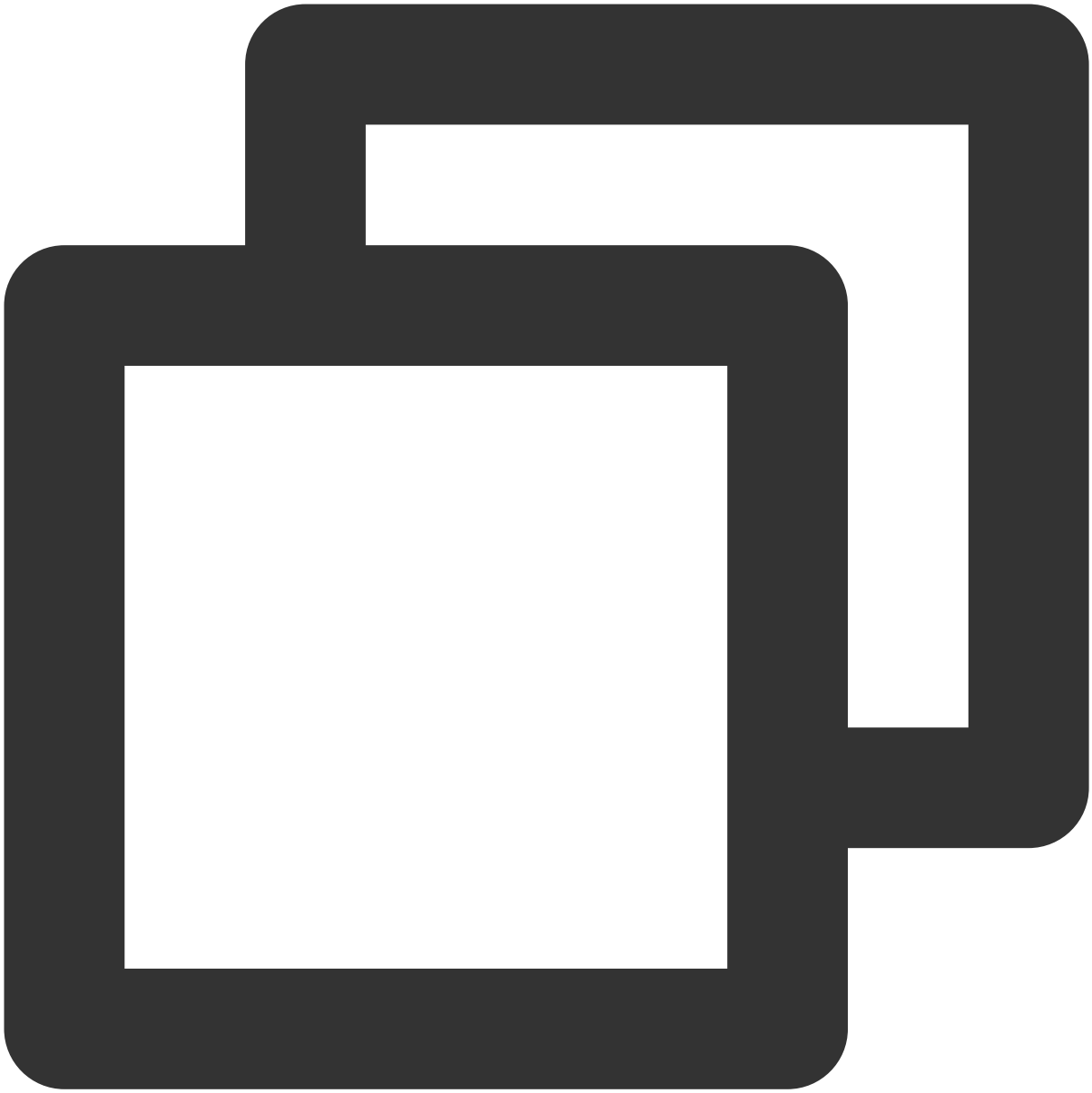
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');

const tempFilePath = './static/rain.mp3'; // Locally stored audio files
let musicFilePath = '';
uni.saveFile({
  tempFilePath: tempFilePath,
  success: (res) => {
    console.warn(JSON.stringify(res));
    musicFilePath =  res.savedFilePath;

    musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath);
```

```
    // Set ringtone
    TUICallKit.setCallingBell(musicFilePath, (res) => {
      if (res.code === 0) {
        console.log('setCallingBell success');
      } else {
        console.log(`setCallingBell failed, error message = ${res.msg}`);
      }
    });
  },
  fail: (err) => {
    console.error(err);
  },
});
```

Silent incoming call ringtone

Enable/Disable incoming call ringtone.

After enabling, the incoming call ringtone will not be played when a call request is received.



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

Flutter

Last updated：2024-05-24 18:46:03

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

Setting application ringtone

There are two ways to set an application ringtone: replace the ringtone audio or call the Setting ringtone interface.

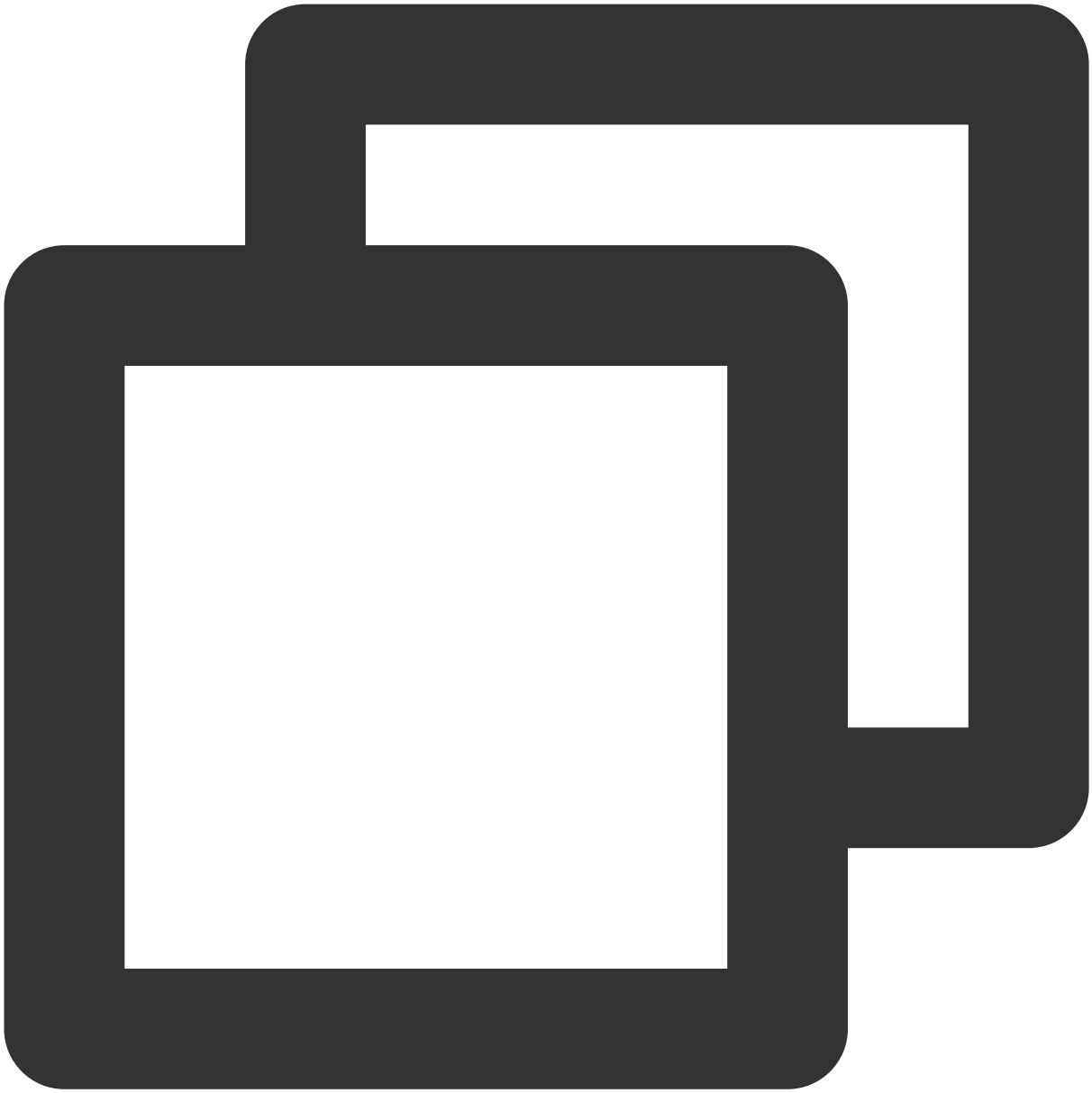
1. Replace Audio File

If you integrate the TUICallKit component via source code dependency, you can replace the three audio files in the **assets\\audios** folder to achieve the purpose of ringtone replacement:

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

2. Set Ringtone Interface

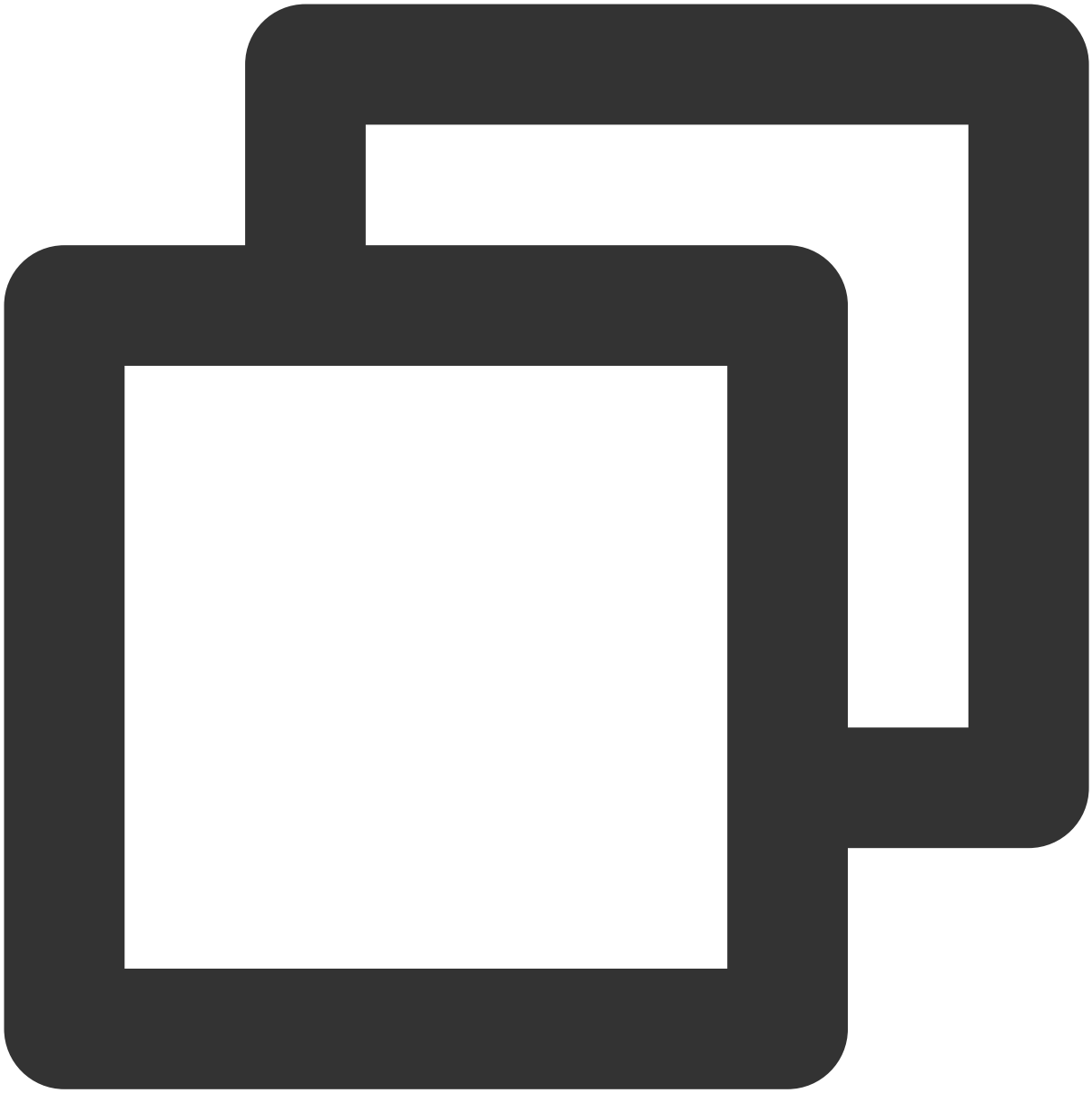
You can also set the incoming call ringtone through the setCallingBell interface.



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

void setCallingBell() {
    TUICallKit.instance.setCallingBell('flie path');
}
```

3. Setting Silent Mode

If you do not need ringing, you can enable the silent mode through enableMuteMode.



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

void enableMuteMode() {
    TUICallKit.instance.enableMuteMode(true);
}
```

Setting Offline Push Ringtone

1. iOS

Voip push does not support custom Definition push ringtones. APNs push allows modifying the parameters `call` and `groupcall` in the interface `params` including `TUIOfflinePushInfo.iOSSound` Setting on the `iOS` platform for offline message ringtones.

2. Android

Note:

The interface supports Huawei, Xiaomi, FCM, and APNs.

FCM's push ringtone is set as the application ringtone.

For Huawei, Xiaomi, and APNs push ringtone settings, please set the `TUIOfflinePushInfo.iOSSound` 's `iOSSound` and `androidSound` fields when calling Call and GroupCall.

Monitoring Call Status Android&iOS&Flutter

Last updated：2024-05-24 18:35:12

This article describes how to use call status callbacks with the TUICallKit component.

Call State Monitoring

If your business requires **monitoring call status**, such as the start and end of a call, and other events during the call, you can refer to the following code:

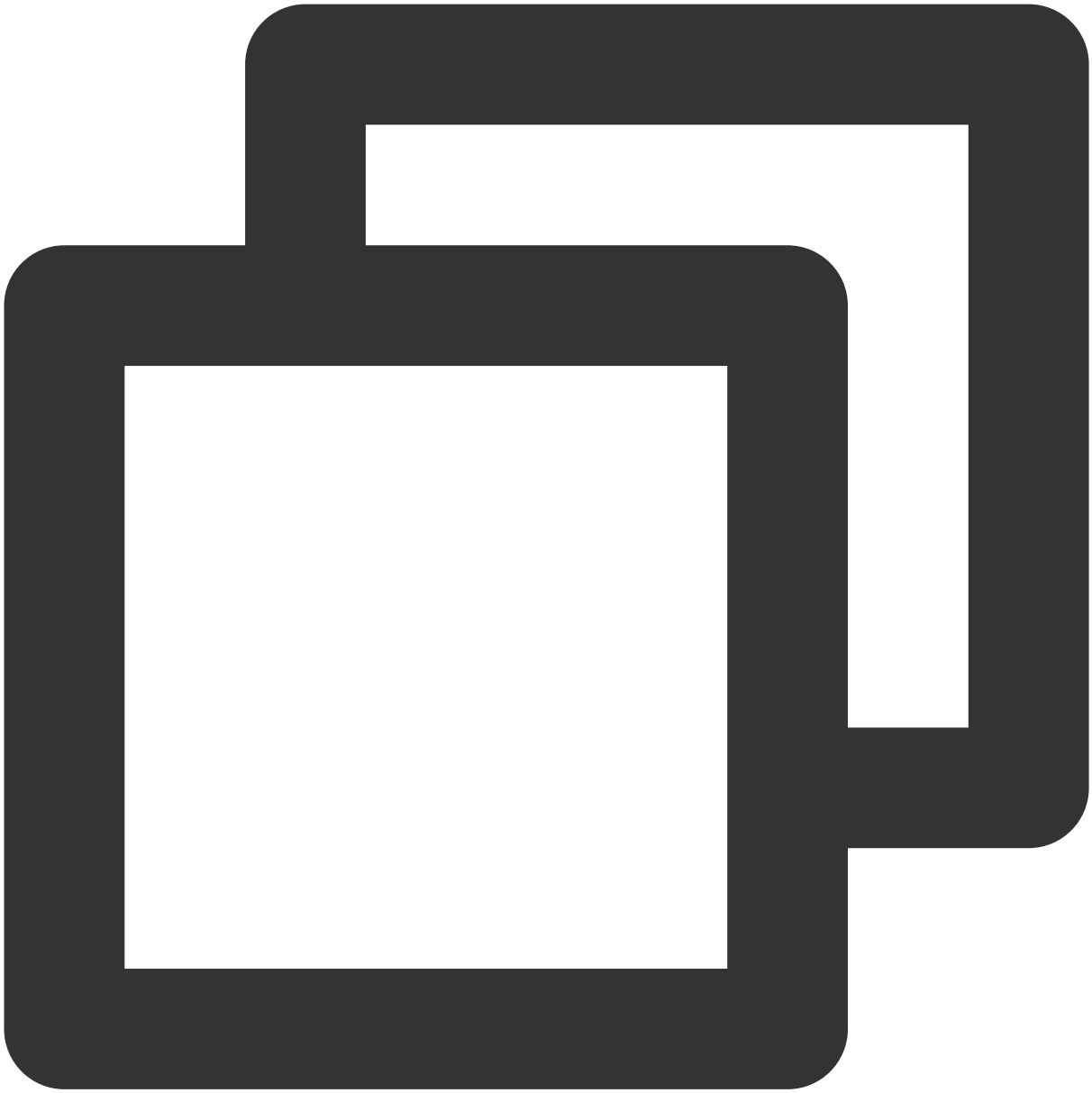
Android(Kotlin)

Android(Java)

iOS(Swift)

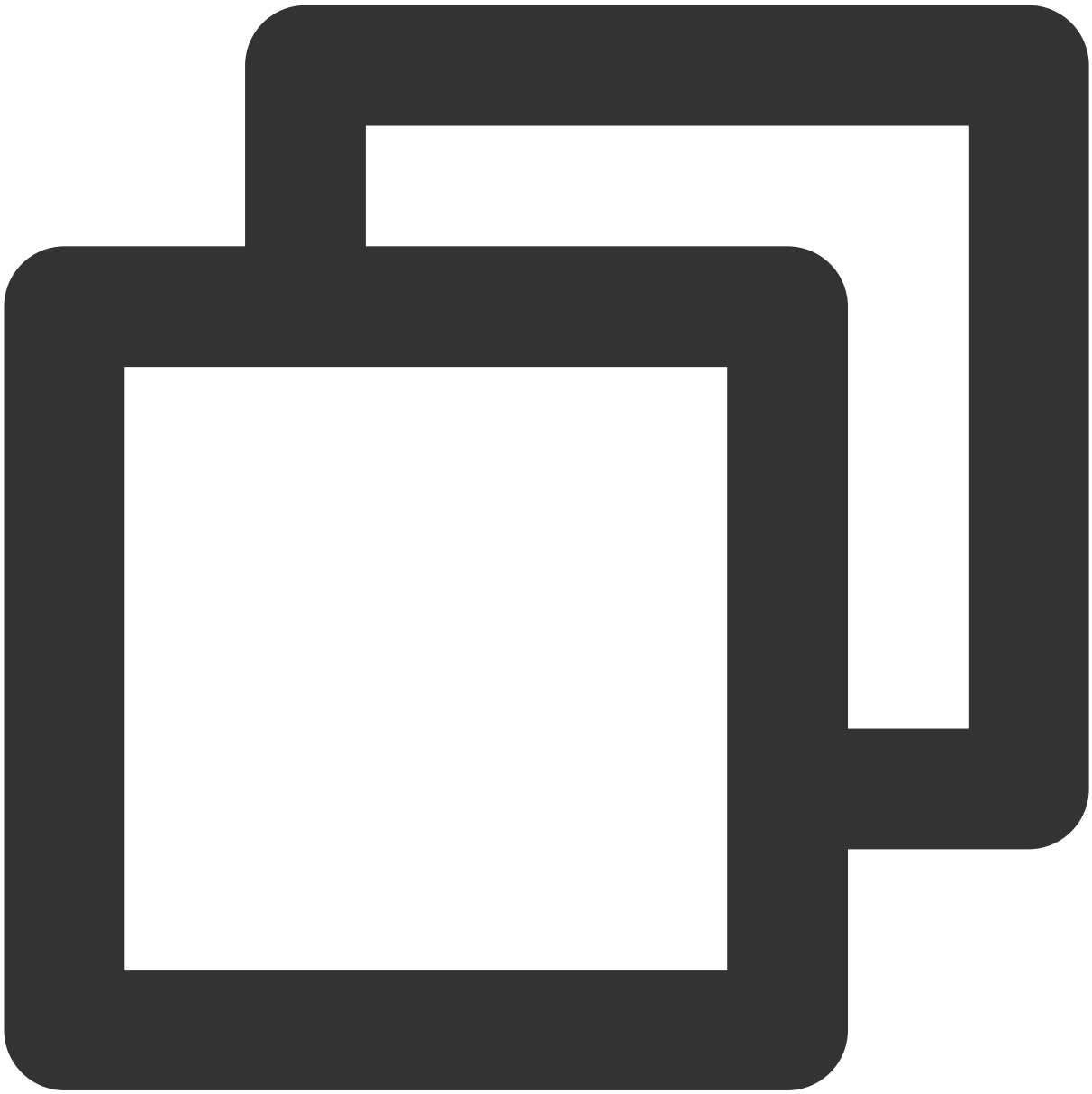
iOS(Objective-C)

Flutter(Dart)



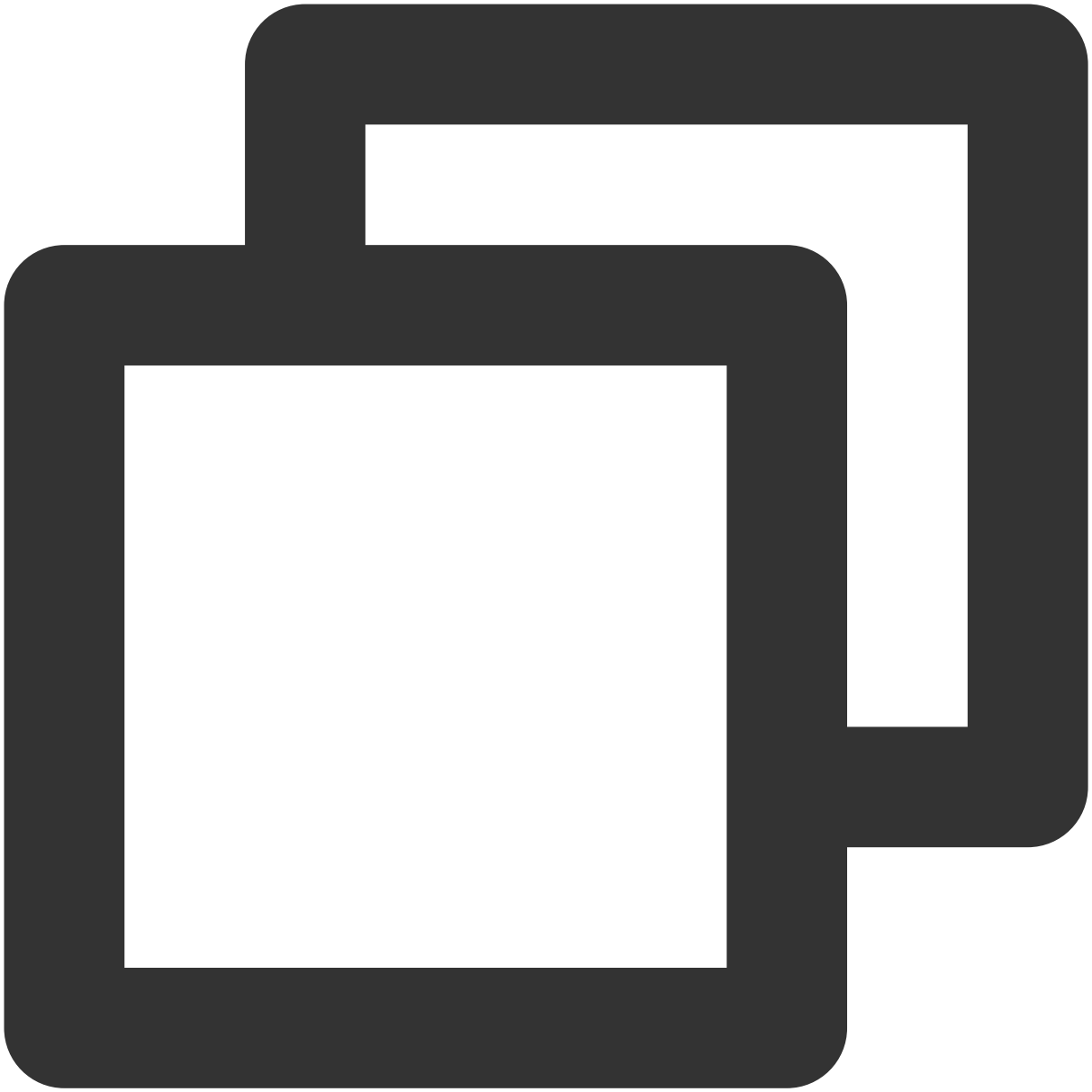
```
import com.tencent.qcloud.tuikit.TUICommonDefine
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallengine.TUICallEngine
import com.tencent.qcloud.tuikit.tuicallengine.TUICallObserver

private val observer: TUICallObserver = object : TUICallObserver() {
    override fun onCallBegin(roomId: TUICommonDefine.RoomId?, callMediaType: TUICal
    }
    override fun onCallEnd(roomId: TUICommonDefine.RoomId?, callMediaType: TUICallD
    }
    override fun onUserNetworkQualityChanged(networkQualityList: MutableList<TUICom
```

```
    }
}
private fun initData() {
    TUICallEngine.createInstance(context).addObserver(observer)
}
```



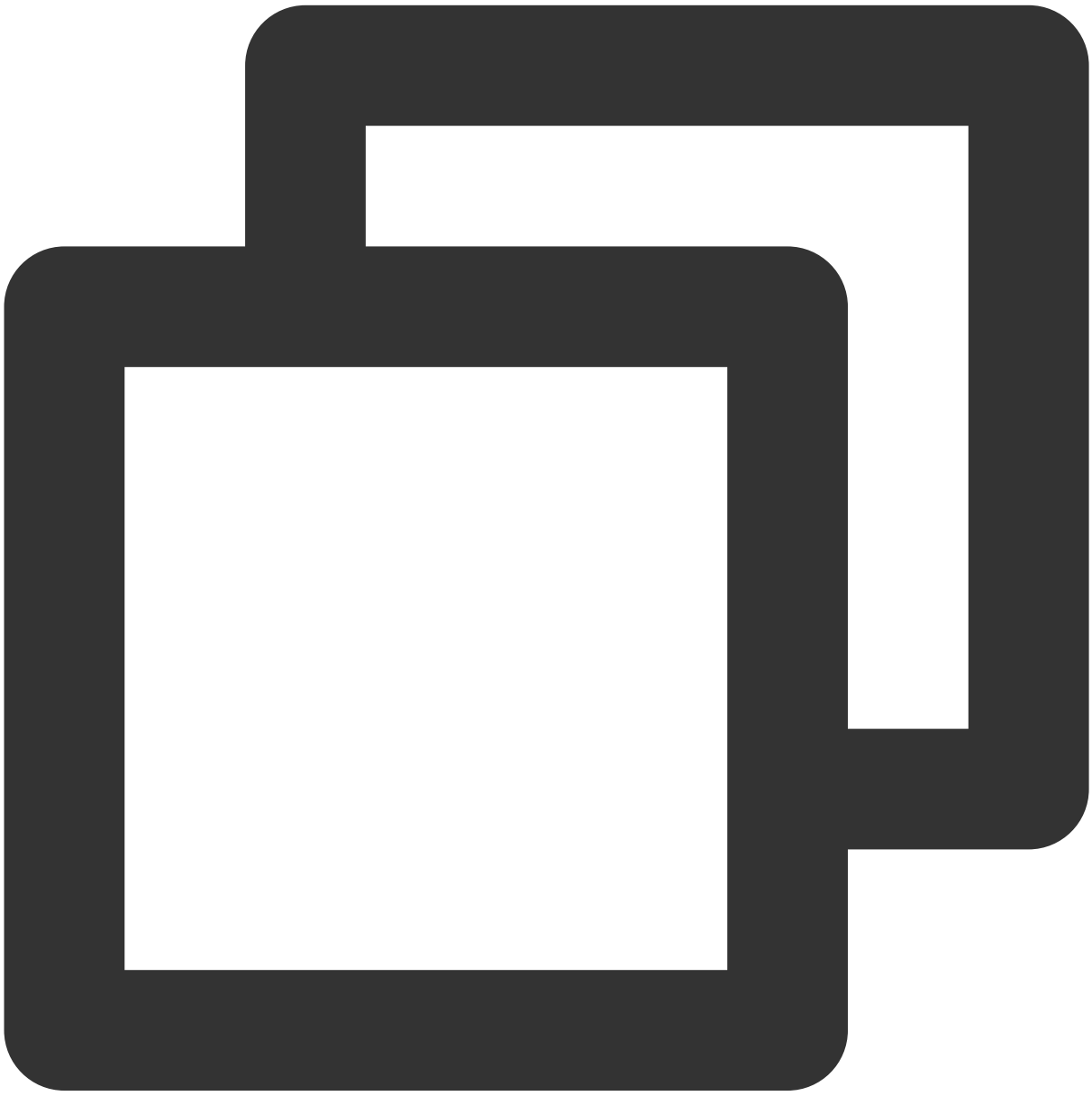
```
import com.tencent.qcloud.tuikit.TUICommonDefine;
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine;
import com.tencent.qcloud.tuikit.tuicallengine.TUICallEngine;
import com.tencent.qcloud.tuikit.tuicallengine.TUICallObserver;
```

```
private TUICallObserver observer = new TUICallObserver() {
    @Override
    public void onCallBegin(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType
    }
    @Override
    public void onCallEnd(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType ca
    }
    @Override
    public void onUserNetworkQualityChanged(List<TUICommonDefine.NetworkQualityInfo
    }
};

private void initData(){
    TUICallEngine.createInstance(context).addObserver(observer);
}
```



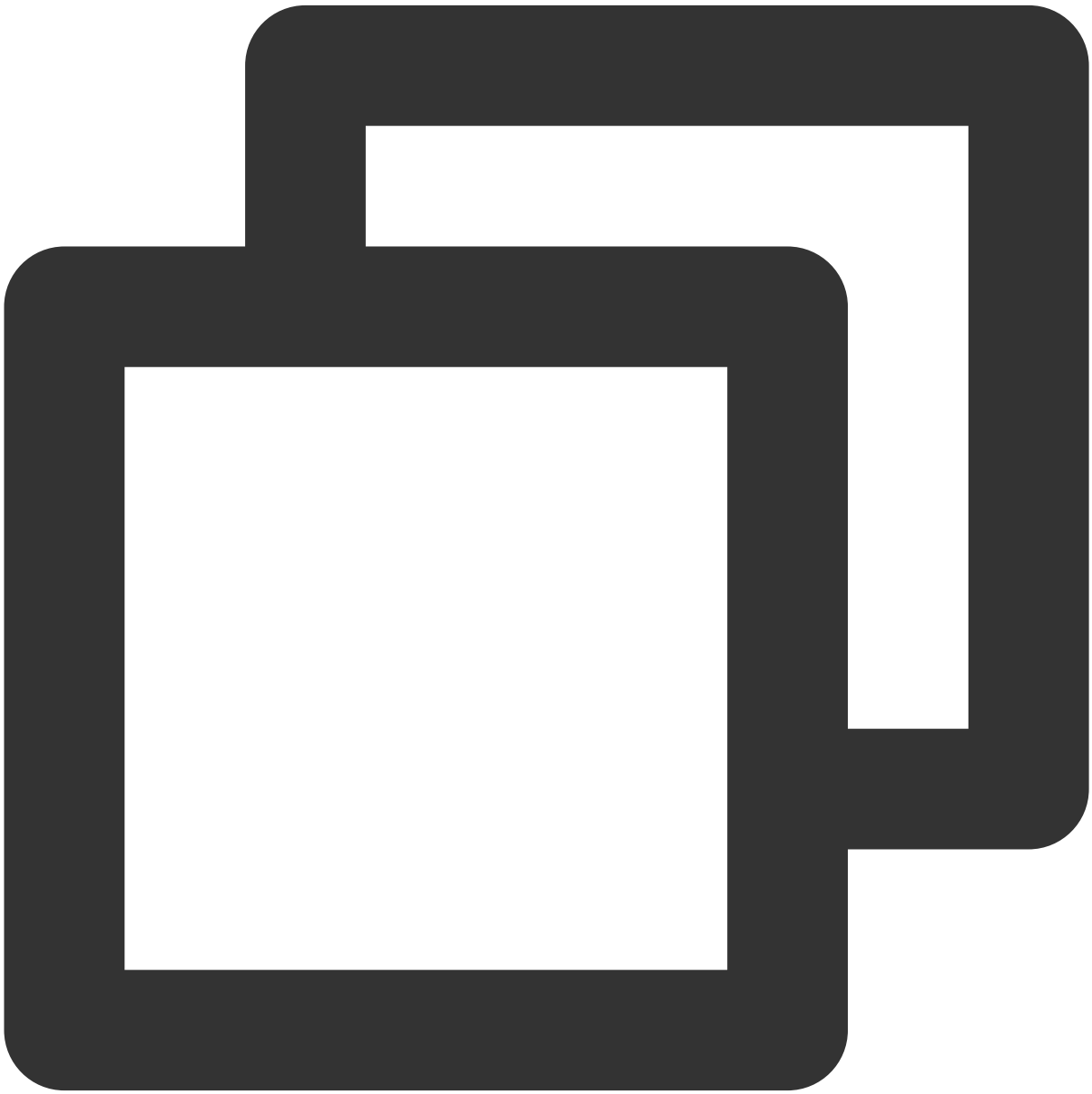
```
import TUICallEngine

TUICallEngine.createInstance().addObserver(self)

func onCallBegin(roomId: TUIRoomId, callMediaType: TUICallMediaType, callRole: TUIC

}

func onCallEnd(roomId: TUIRoomId, callMediaType: TUICallMediaType, callRole: TUICal

}
```

```
func onUserNetworkQualityChanged(networkQualityList: [TUINetworkQualityInfo]) {

}
```



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

[[TUICallEngine createInstance] addObserver:self];

- (void)onCallBegin:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaTy
```

```
}

- (void)onCallEnd:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaType

}

- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkQualityInfo *> *)networkQual

}
```



```
import 'package:tencent_calls_engine/tencent_calls_engine.dart';
```

```
TUICallObserver observer = TUICallObserver(
    onError: (int code, String message) {

    }, onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole

    }, onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole ca

    }, onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList

    }, onCallReceived: (String callerId, List<String> calleeIdList, String groupId,

    }
))

void addObserver() {
    TUICallEngine.instance.addObserver(observer);
}
```

Note:

On the Android platform, when setting TUICallObserver to listen for callbacks, ensure that the class hosting the callback will not be cleared. For example, it is not recommended to add the observer in LoginActivity, as when LoginActivity is destroyed, the callback will also be cleared; it is suggested to observe in the application's Application class or the main application interface.

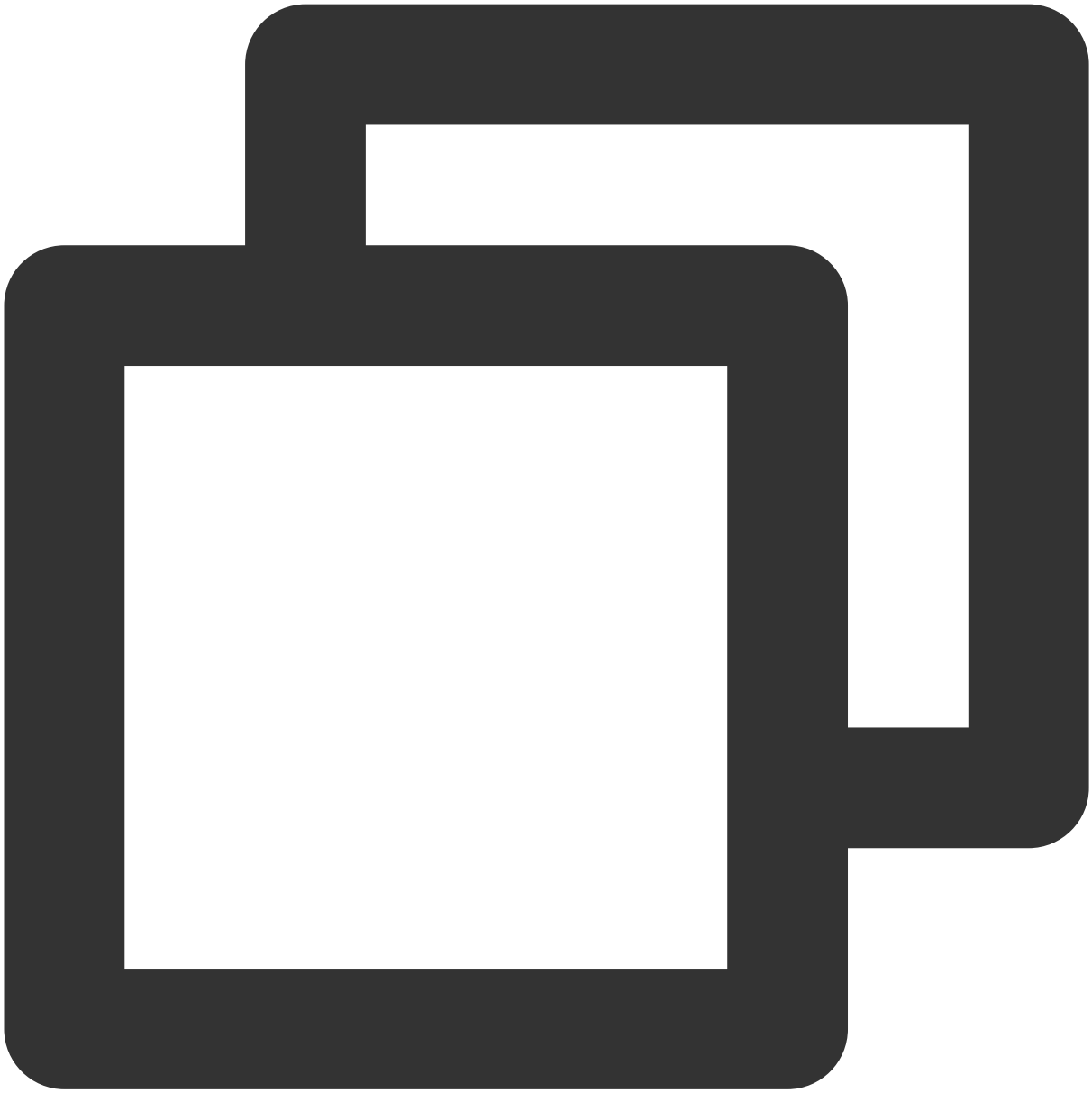
Web&H5

Last updated：2024-04-03 17:23:11

This article describes how to use call status callbacks with the TUICallKit component.

Call State Monitoring

If your business requires **monitoring the call status**, such as events during the call process (TUICallEvent for details), you can refer to the following code:



```
import { TUICallEvent } from 'tuicall-engine-webrtc';

let handleUserEnter = function(event) {
    console.log('TUICallEvent.USER_ENTER: ', event);
};

TUICallKitServer.getTUICallEngineInstance().on(TUICallEvent.USER_ENTER, handleUserE
TUICallKitServer.getTUICallEngineInstance().off(TUICallEvent.USER_ENTER, handleUser
```

Component Callback Event

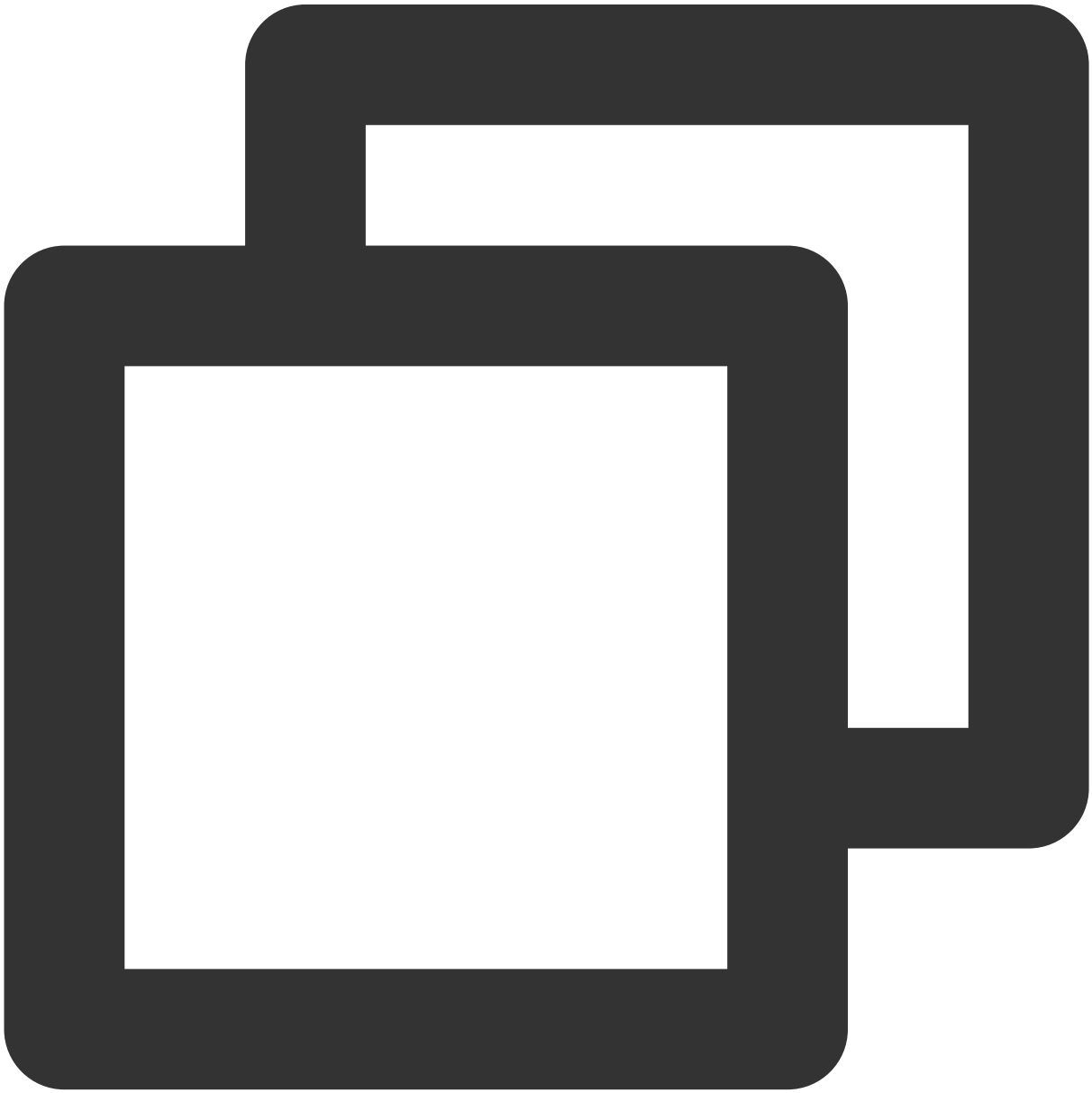
The TUICallKit component offers call status callbacks, which can be used to implement more interaction logic on the business side. Please refer to the TUICallKit Attribute Overview for more details.

`beforeCalling` : Executed prior to the commencement of the call.

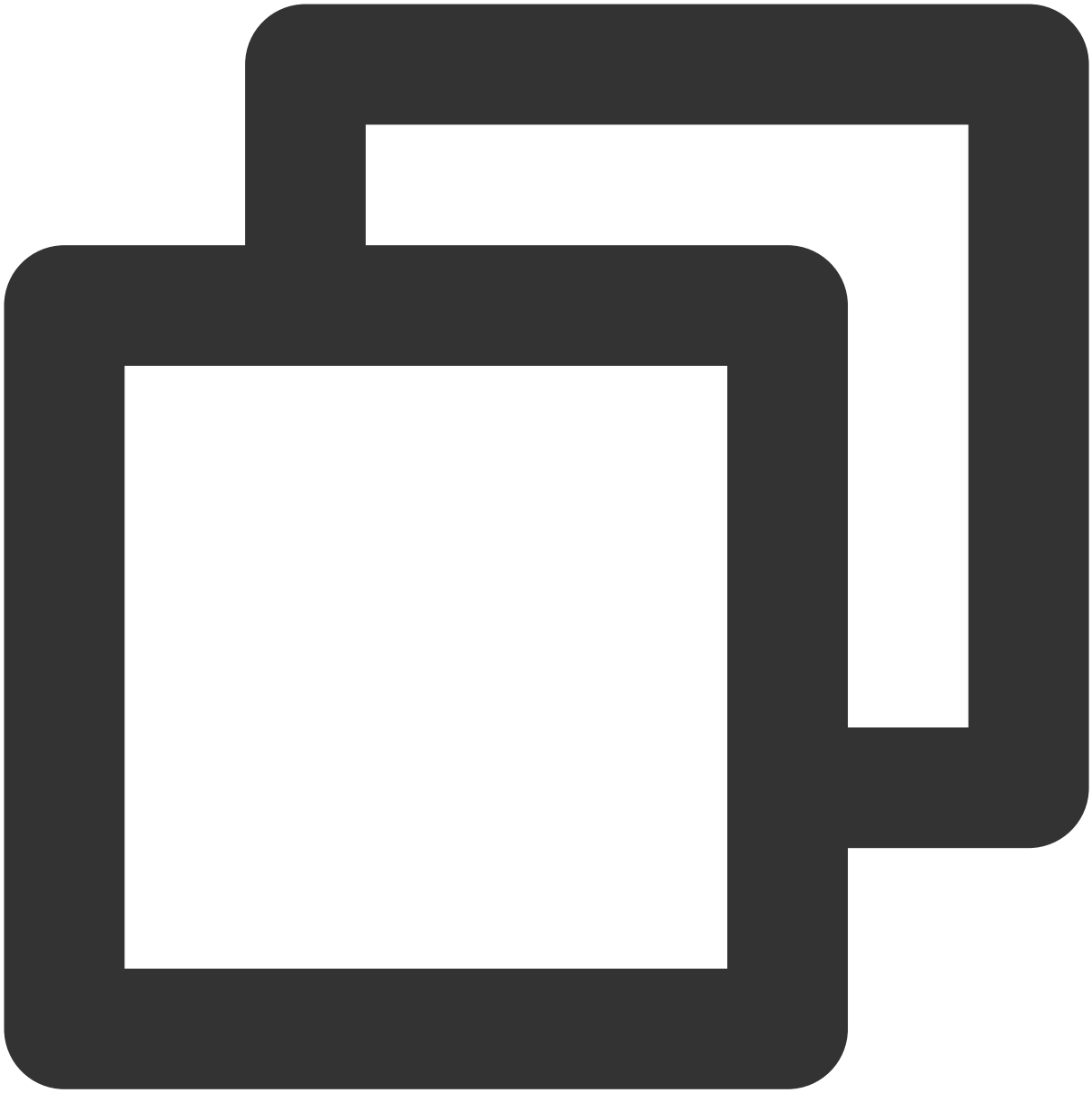
`afterCalling` : Executed upon the completion of the call.

React

Vue



```
function App() {
  const handleBeforeCalling = () => {
    console.log("[TUICallKit Demo] beforeCalling");
  };
  const handleAfterCalling = () => {
    console.log("[TUICallKit Demo] afterCalling");
  };
  return (
    <TUICallKit
      beforeCalling={handleBeforeCalling}
      afterCalling={handleAfterCalling} />
  )
}
```



```
<template>
  <TUICallKit
    :beforeCalling="handleBeforeCalling"
    :afterCalling="handleAfterCalling" />
</template>
<script setup>
function handleBeforeCalling() {
  console.log("[TUICallKit Demo] beforeCalling");
}
function handleAfterCalling() {
  console.log("[TUICallKit Demo] afterCalling");
```

```
}
</script>
```

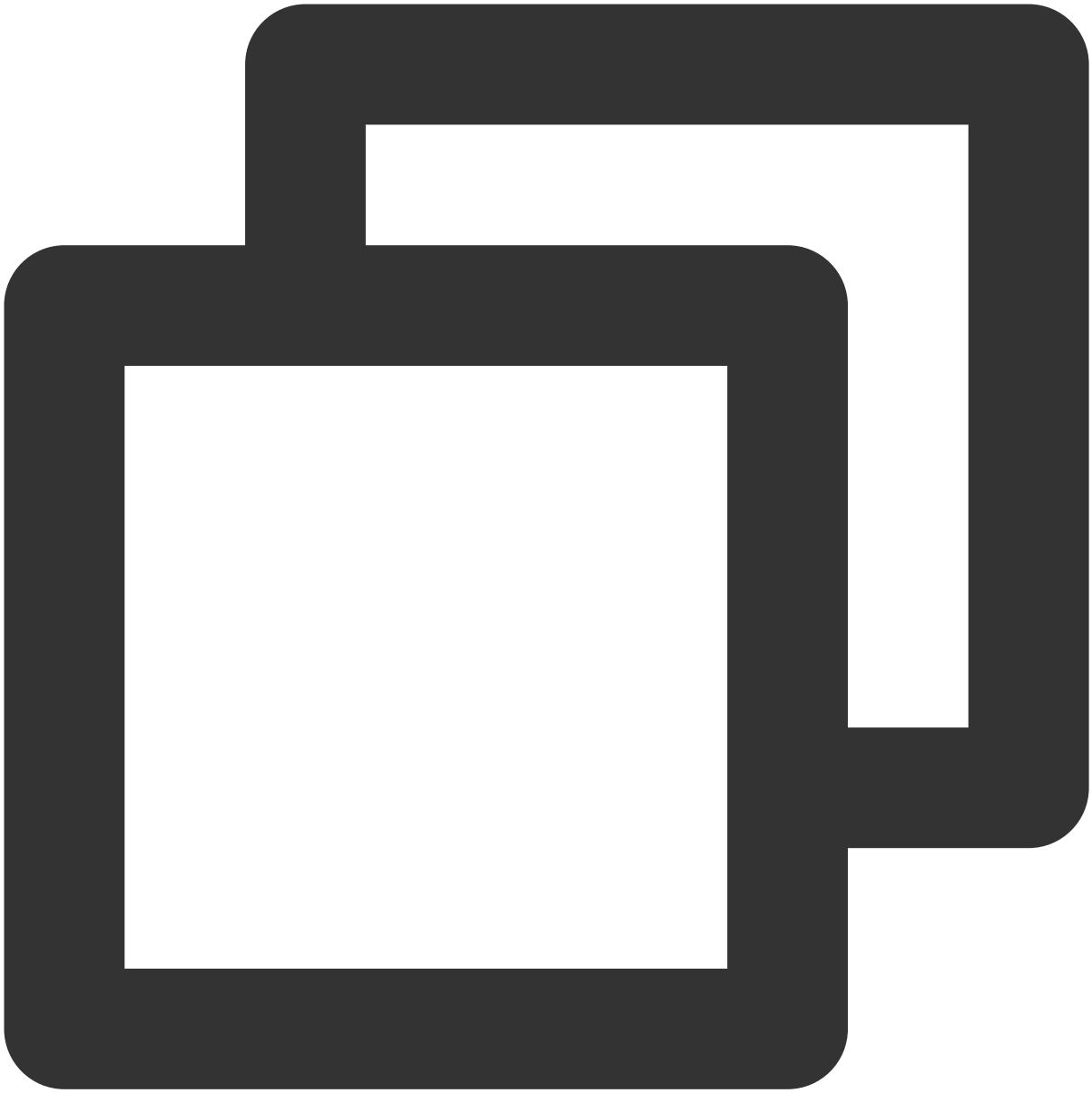
uni-app（Android & iOS）

Last updated：2024-04-30 10:43:45

This article describes how to use call status callbacks with the TUICallKit component.

Call State Monitoring

If your business requires **monitoring call status**, such as events that occur during a call like starting and ending (see TUICallEvent for details), please refer to the following code.



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin

function handleError(res) {
    console.log('onError', JSON.stringify(res);
}
TUICallEngine.addEventListener('onError', handleError);
TUICallEngine.removeEventListener('onError', handleError);
```

Language Settings
Web&H5

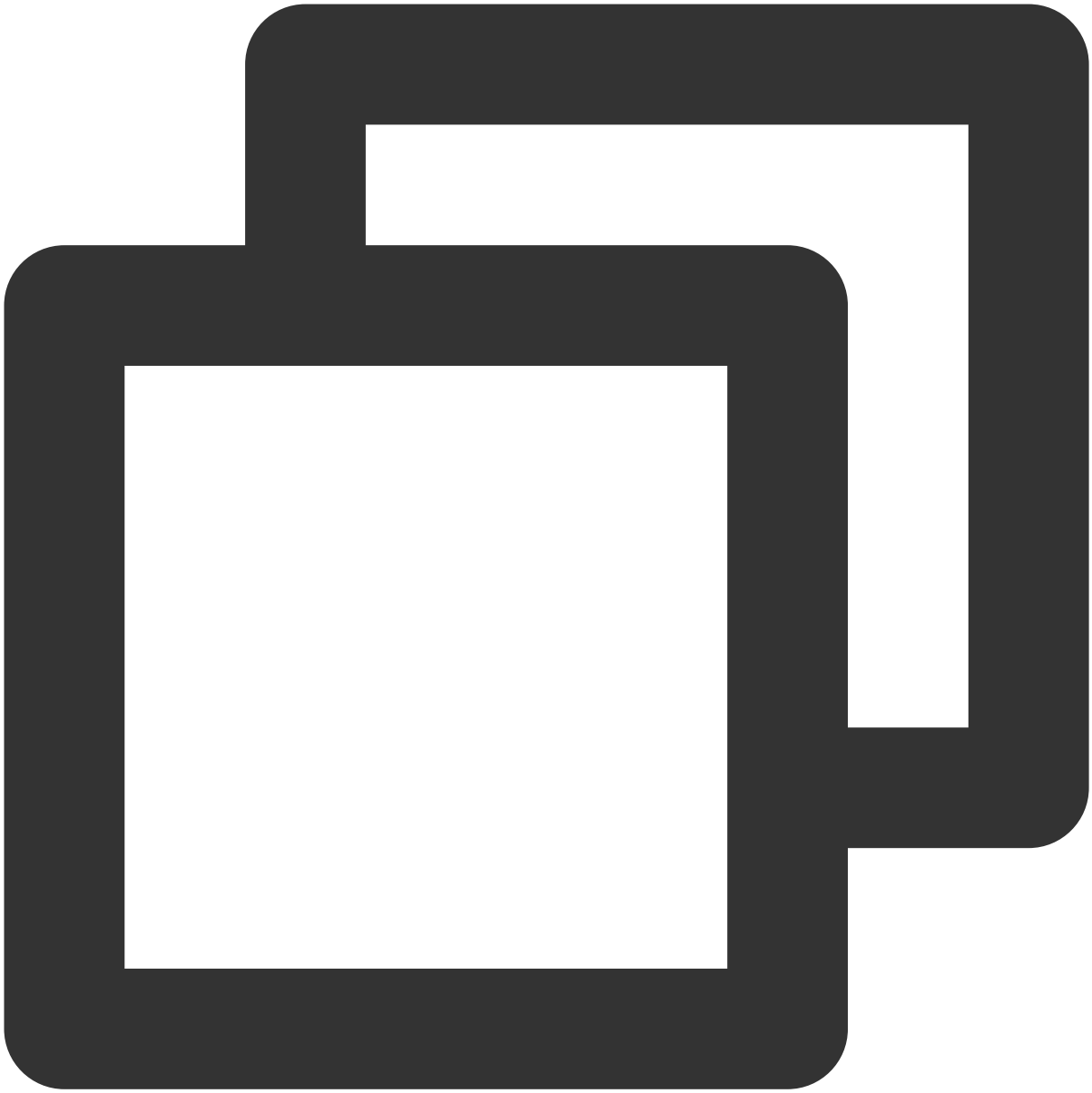
Last updated：2024-07-26 10:11:03

Supported Languages

Currently supports **Simplified Chinese, English, Japanese**.

Switch Language

TUICallKit will prioritize the browser language. If it is one of Chinese, English, or Japanese, the browser language will be used. Otherwise, English will be used. If you want to switch languages, you can use the setLanguage interface.



```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-vue';

TUICallKitServer.setLanguage("zh-cn"); // "en" | "zh-cn" | "ja_JP"
```

Add New Language

If you need support for other languages, you can modify the language source file via source code integration.

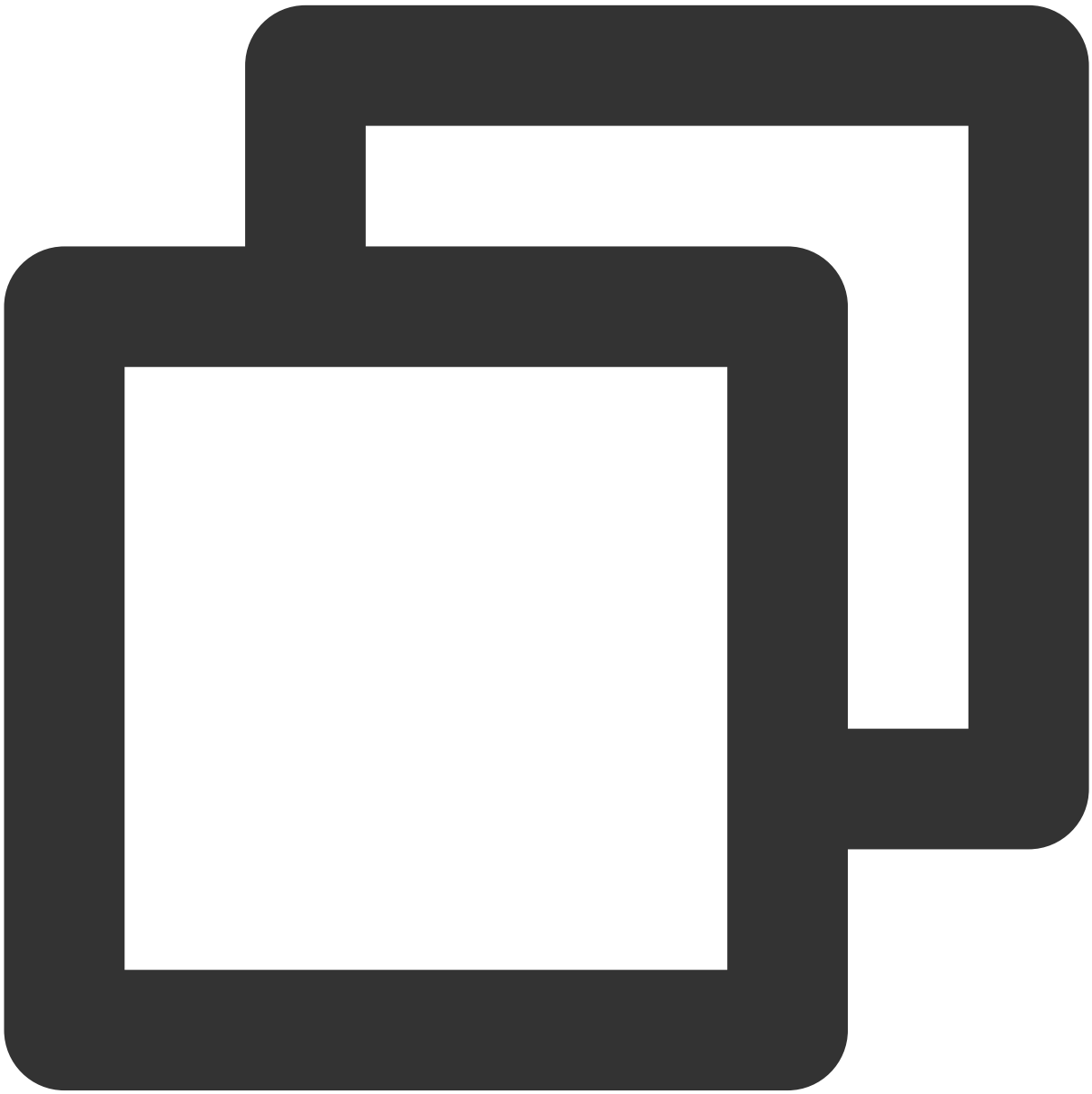
Step 1: Source Code Integration

Note:

Source code integration is suitable for `Vue + TypeScript` projects and `TUICallKit` version 3.2.2 or above.

1. **Download Source Code**

Vue3

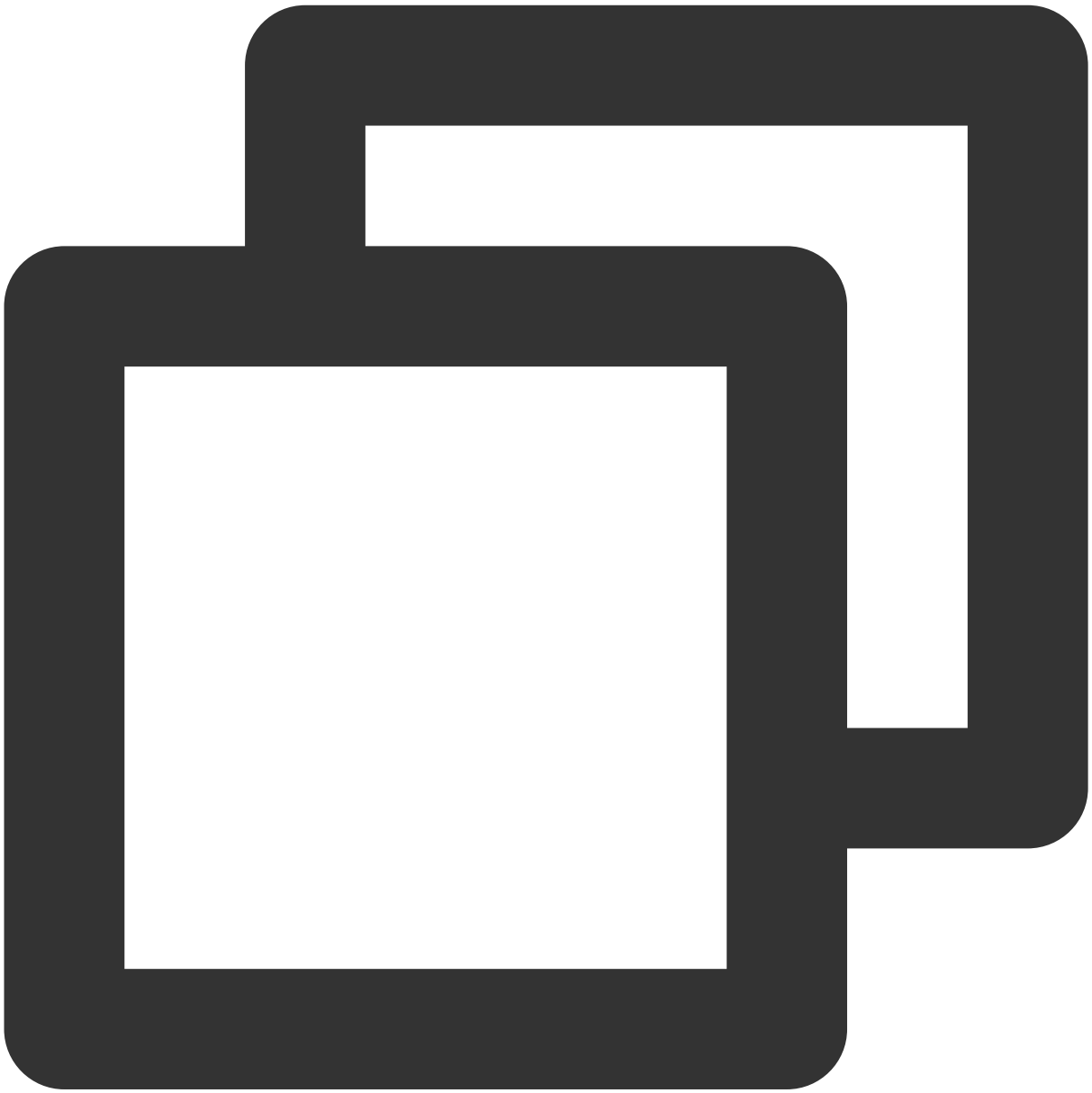


```
npm install @tencentcloud/call-uikit-vue
```

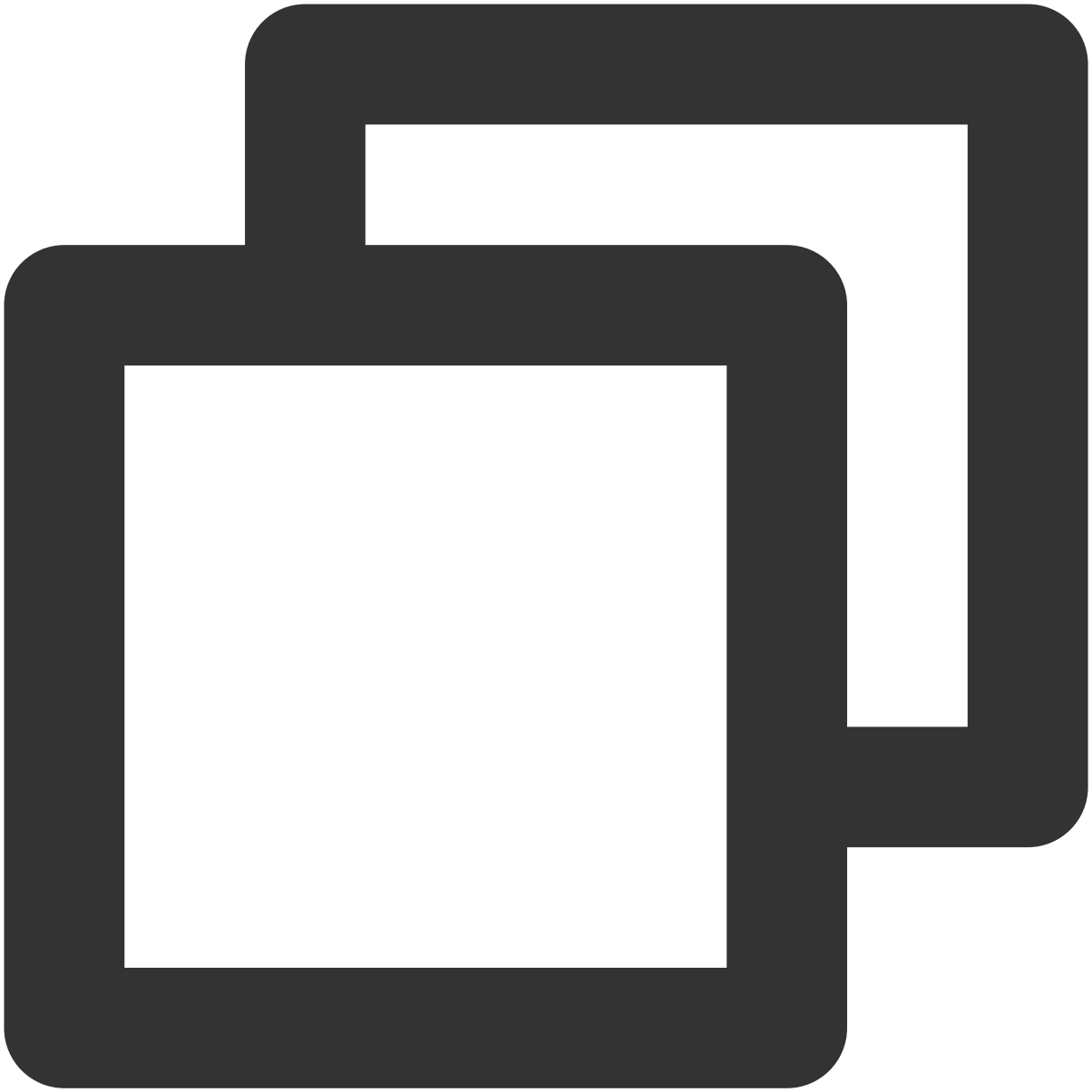
2. **Copy the source code into your own project, taking copying into the `src/components/` directory as an example:**

macOS + Vue3

Windows + Vue3



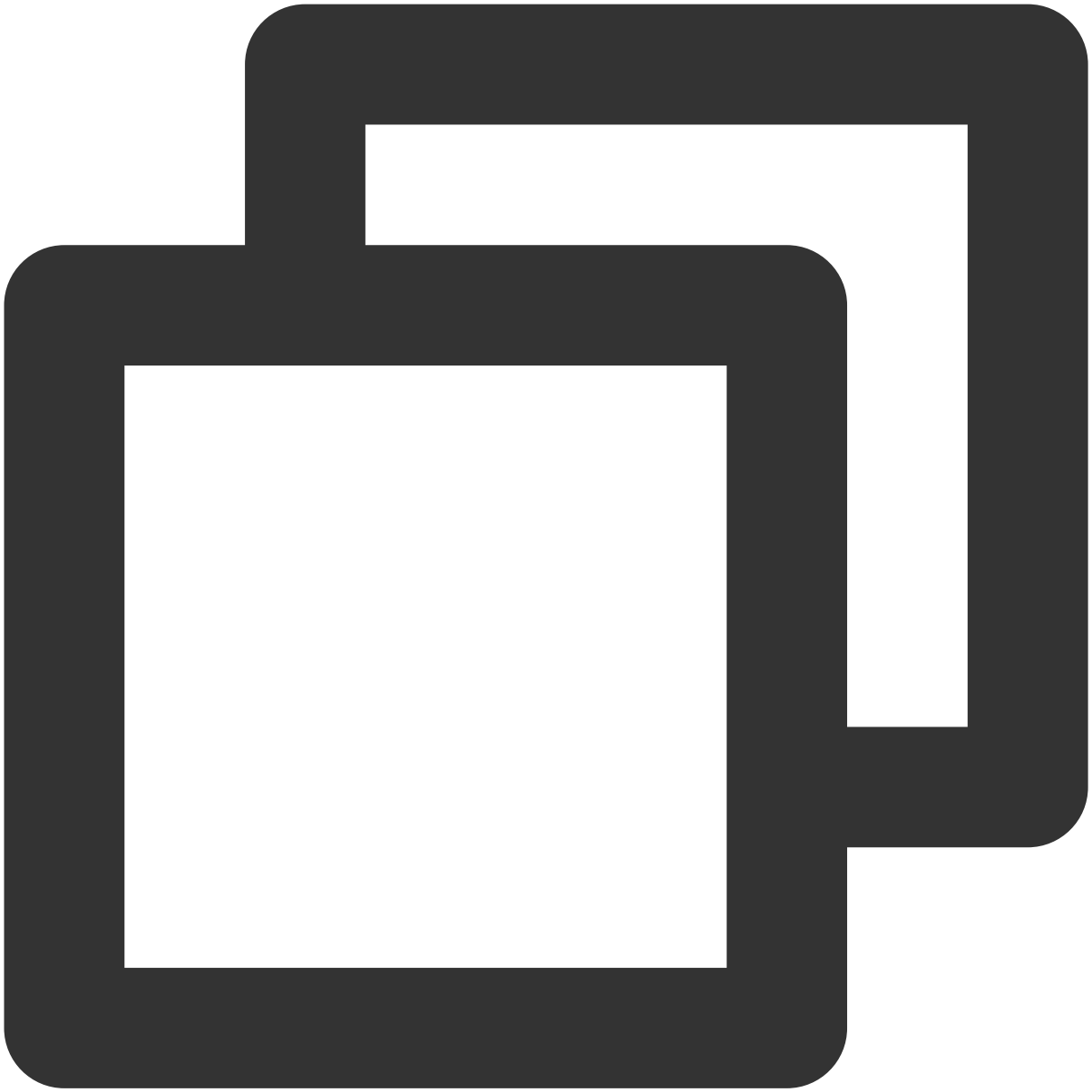
```
mkdir -p ./src/components/TUICallKit && cp -r ./node_modules/@tencentcloud/call-uik
```



```
xcopy .\\node_modules\\@tencentcloud\\call-uikit-vue  .\\src\\components\\TUICallKi
```

3. **Modify Import Path**

You need to change TUICallKit to import from a local file, as shown in the code below. For other usage details, refer to TUICallKit Quick Integration.



```
import { TUICallKit, TUICallKitServer, TUICallType } from "./components/TUICallKit/
```

4.

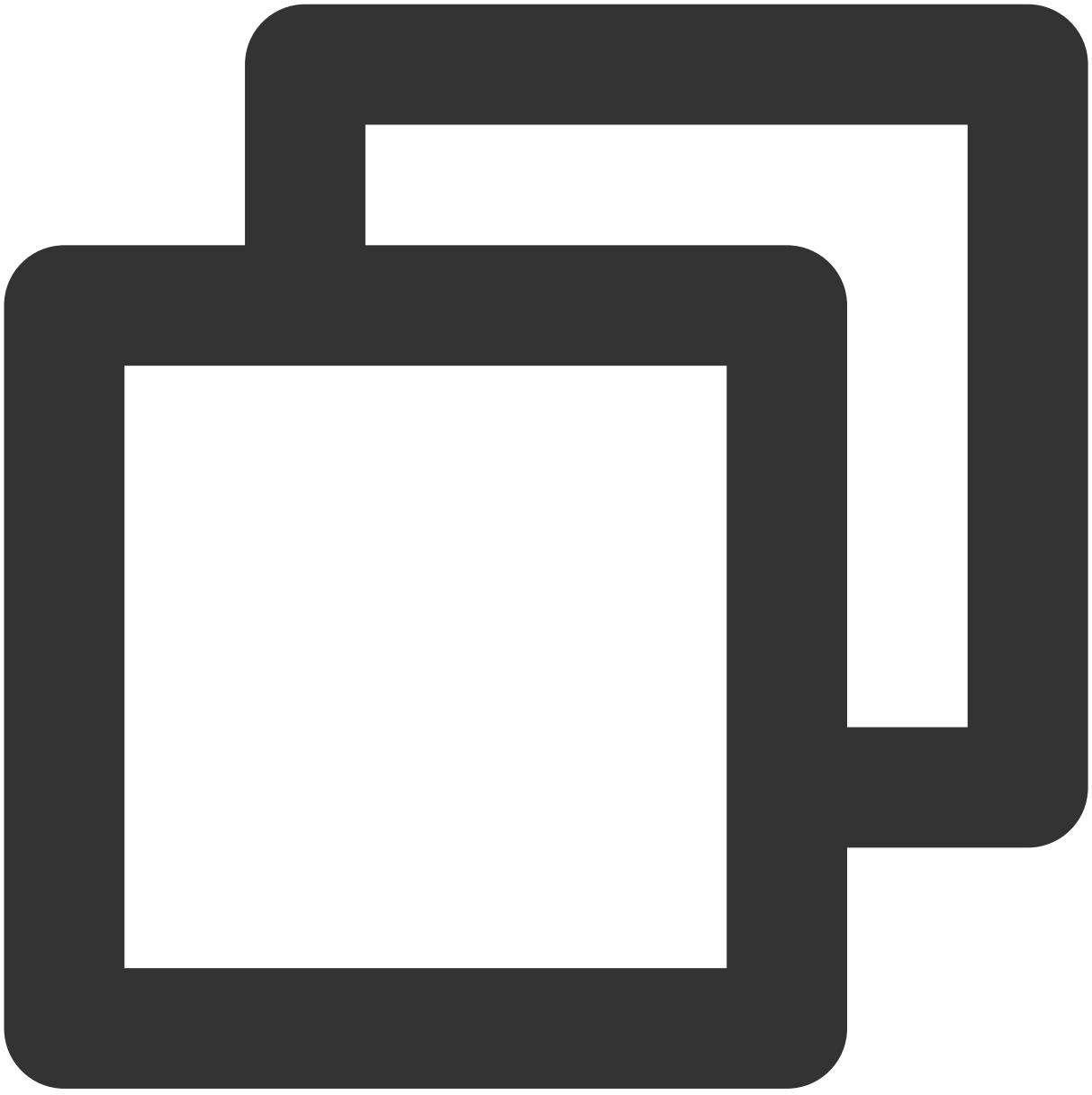
Solve Errors That May Be Caused by Copying Source Code

If you encounter an error while using the TUICallKit component, please don't worry. In most cases, this is due to inconsistencies between ESLint and TSConfig configurations. You can consult the documentation and configure correctly as required. If you need help, please feel free to contact us, and we will ensure that you can successfully use this component. Here are some common issues:

ESLint Error

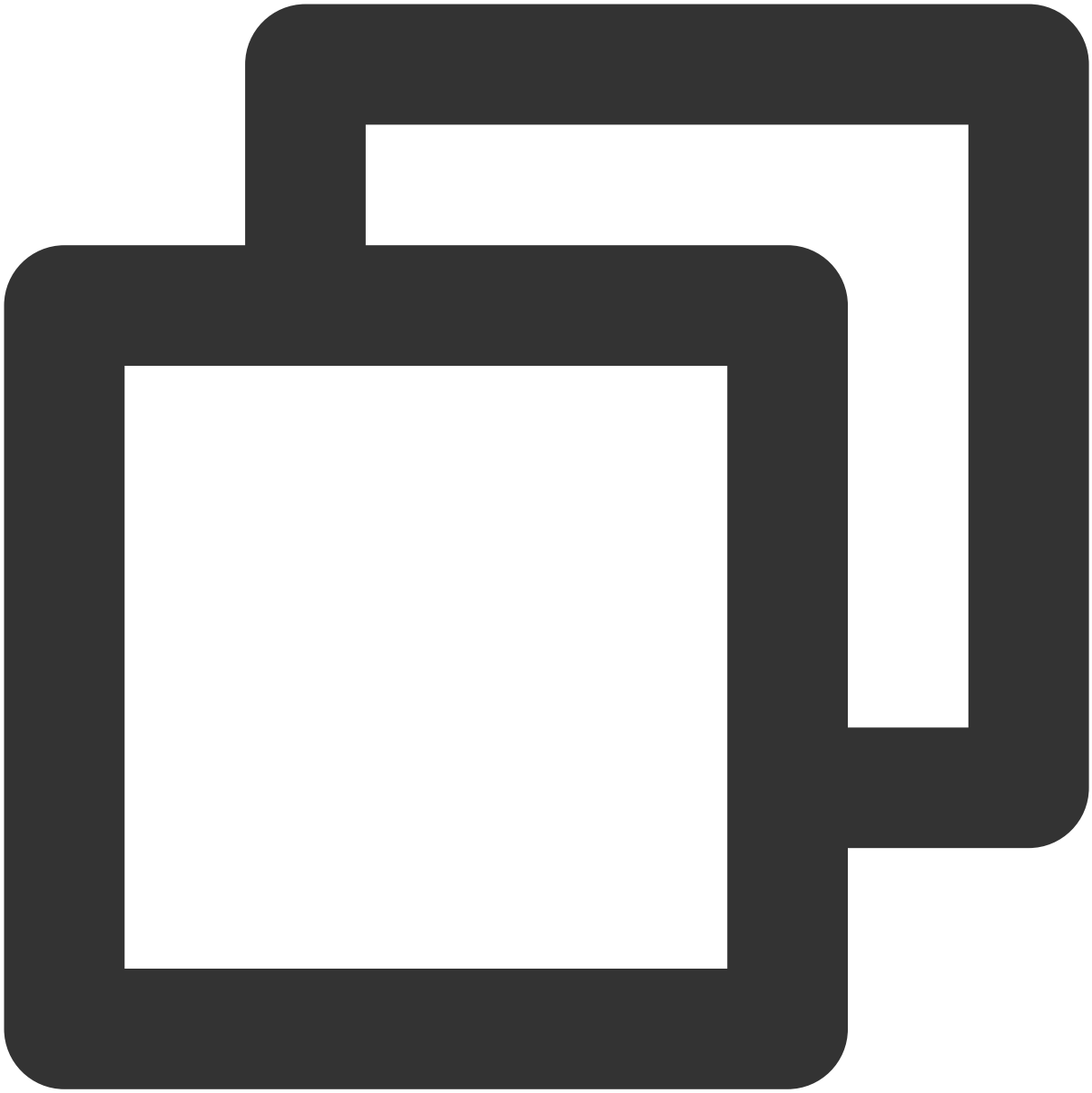
TypeScript Error

If the TUICallKit causes an error due to inconsistency with your project's code style, you can block this component directory by adding a `.eslintignore` file in the root directory of your project, for example:



```
# .eslintignore
src/components/TUICallKit
```

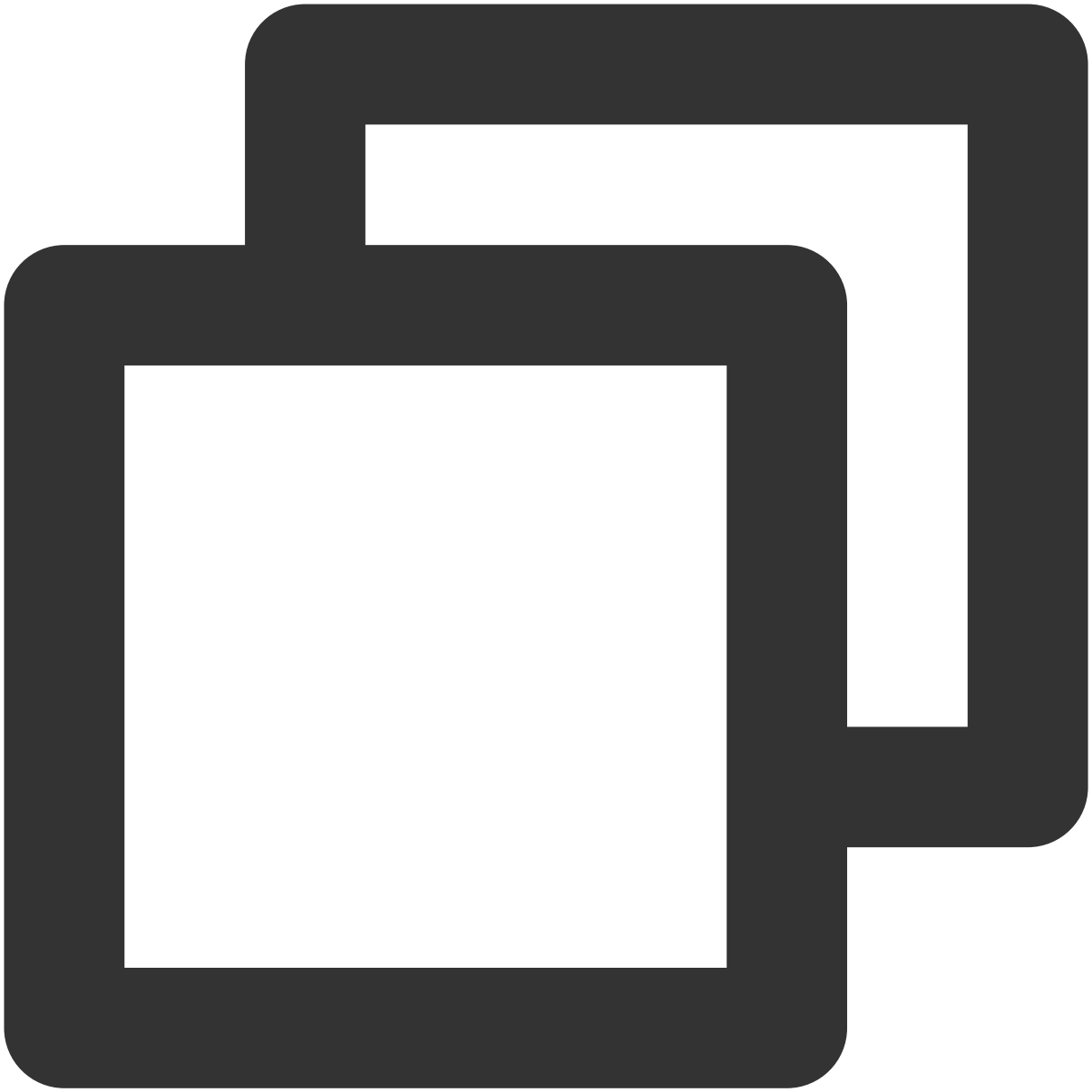
1. If you encounter the 'Cannot find module '../package.json'' error, it's because TUICallKit references a JSON file. You can add the related configuration in tsconfig.json, example:



```
{
  "compilerOptions": {
    "resolveJsonModule": true
  }
}
```

For other TSConfig issues, please refer to TSConfig Reference.

2. If you encounter the 'Uncaught SyntaxError: Invalid or unexpected token' error, it's because TUICallKit uses decorators. You can add the related configuration in tsconfig.json, example:



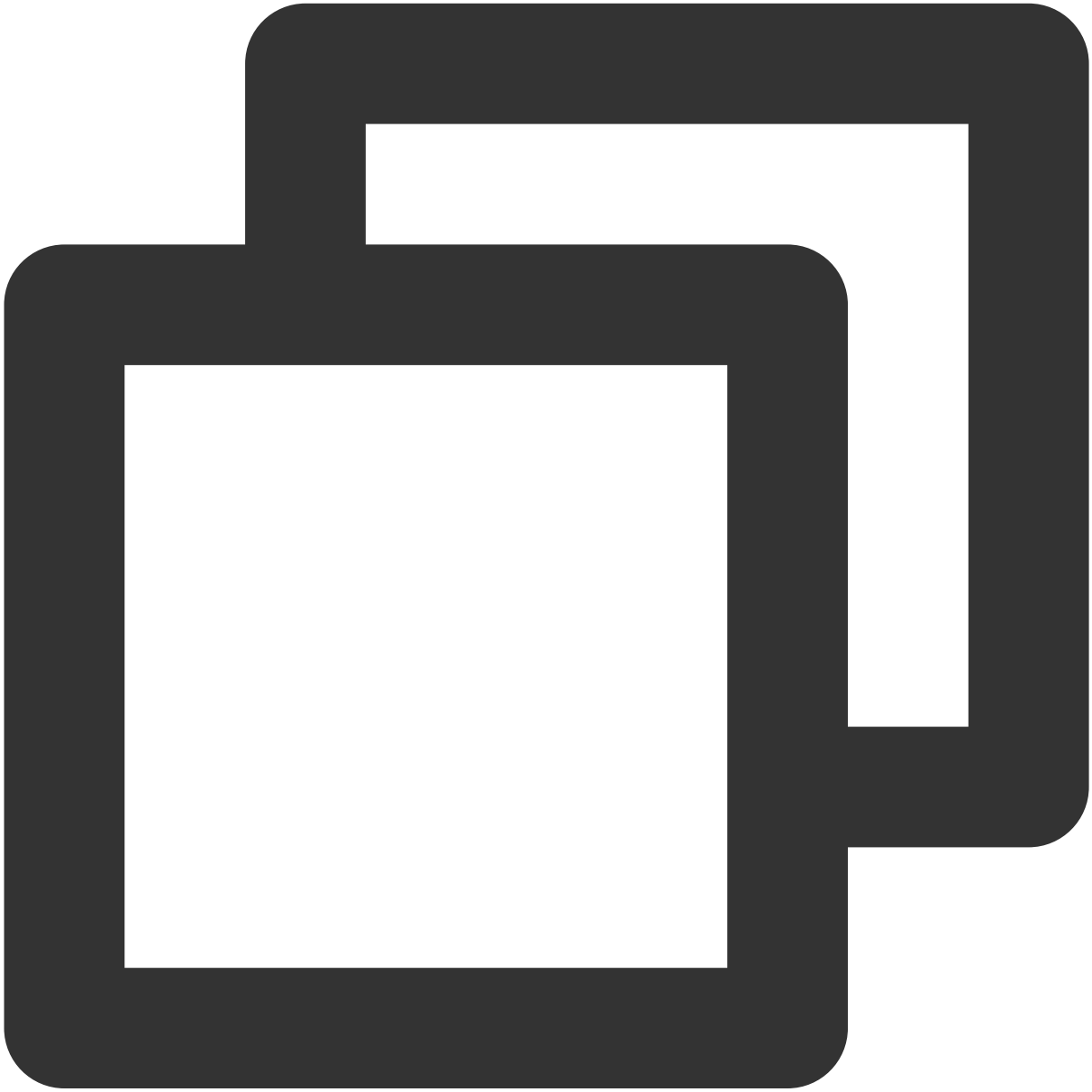
```
{
  "compilerOptions": {
    "experimentalDecorators": true
  }
}
```

Step 2: Add a new language pack

For example, adding Vietnamese:

1. Create the target language source file.

Add a new file named vi.ts in the src/components/TUICallKit/src/TUICallService/locales directory. Copy the contents of src/components/TUICallKit/src/TUICallService/locales/zh-cn.ts to vi.ts and then translate the JSON values into Vietnamese.

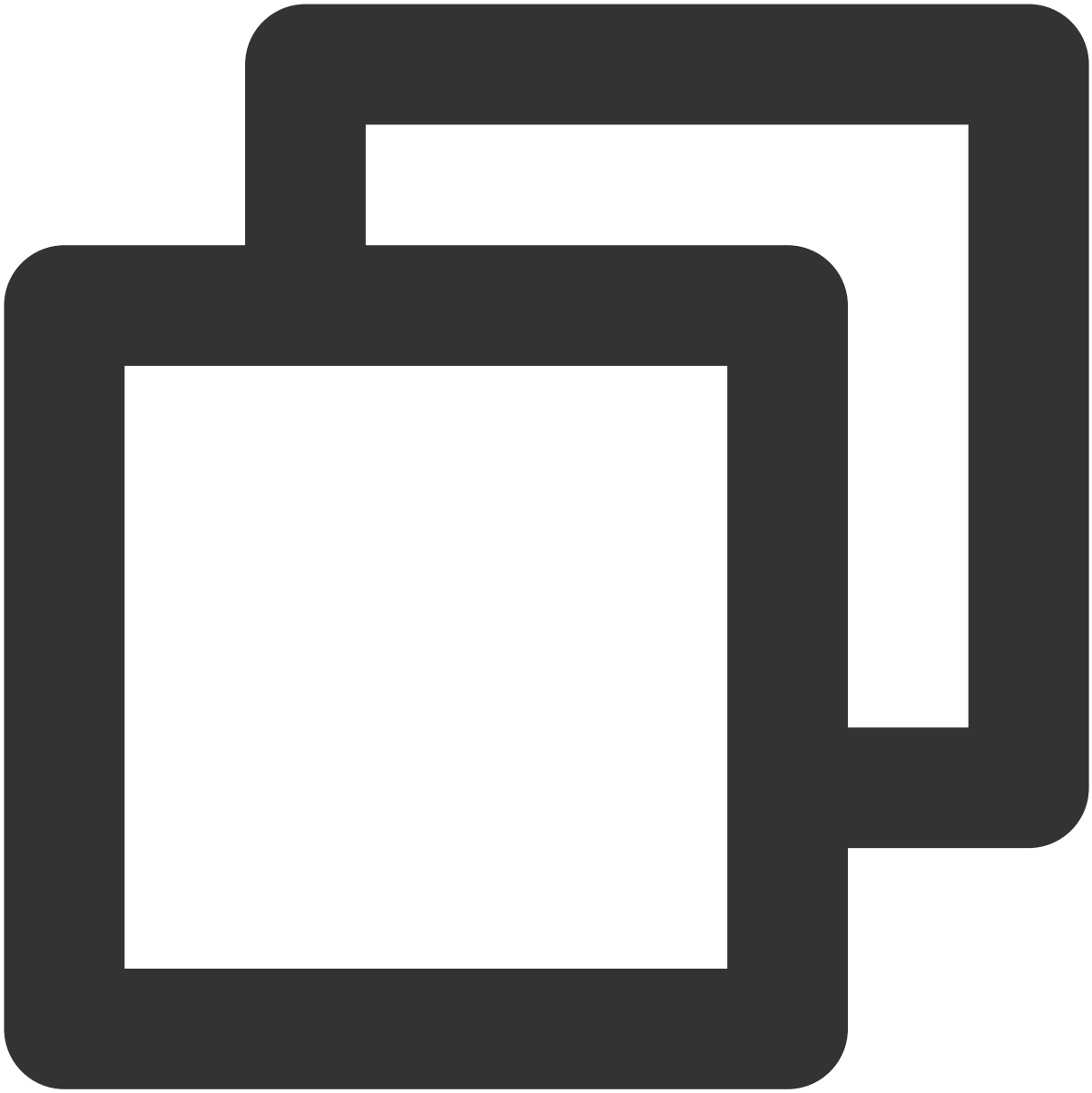


```
export const vi = {    // Note the export variable here
  'hangup': 'Hang Up',
  'reject': 'Reject',
  'accept': 'Acceptance',
  'camera': 'Camera',
  'microphone': 'Microphone',
  'speaker': 'Speaker',
```

```
  'open camera': 'Turn on the camera',
  'close camera': 'Turn off the camera',
  'open microphone': 'Open microphone',
  'close microphone': 'Turn off the microphone',
  'video-to-audio': 'Switch to audio call',
  'virtual-background': 'Blur background',
  'other side reject call': 'The other side has rejected',
  'reject call': 'Reject call',
  'cancel': 'Cancel call',
  ...
};
```

2. Export from index.ts

Modify the src/components/TUICallKit/src/TUICallService/locales/index.ts file.

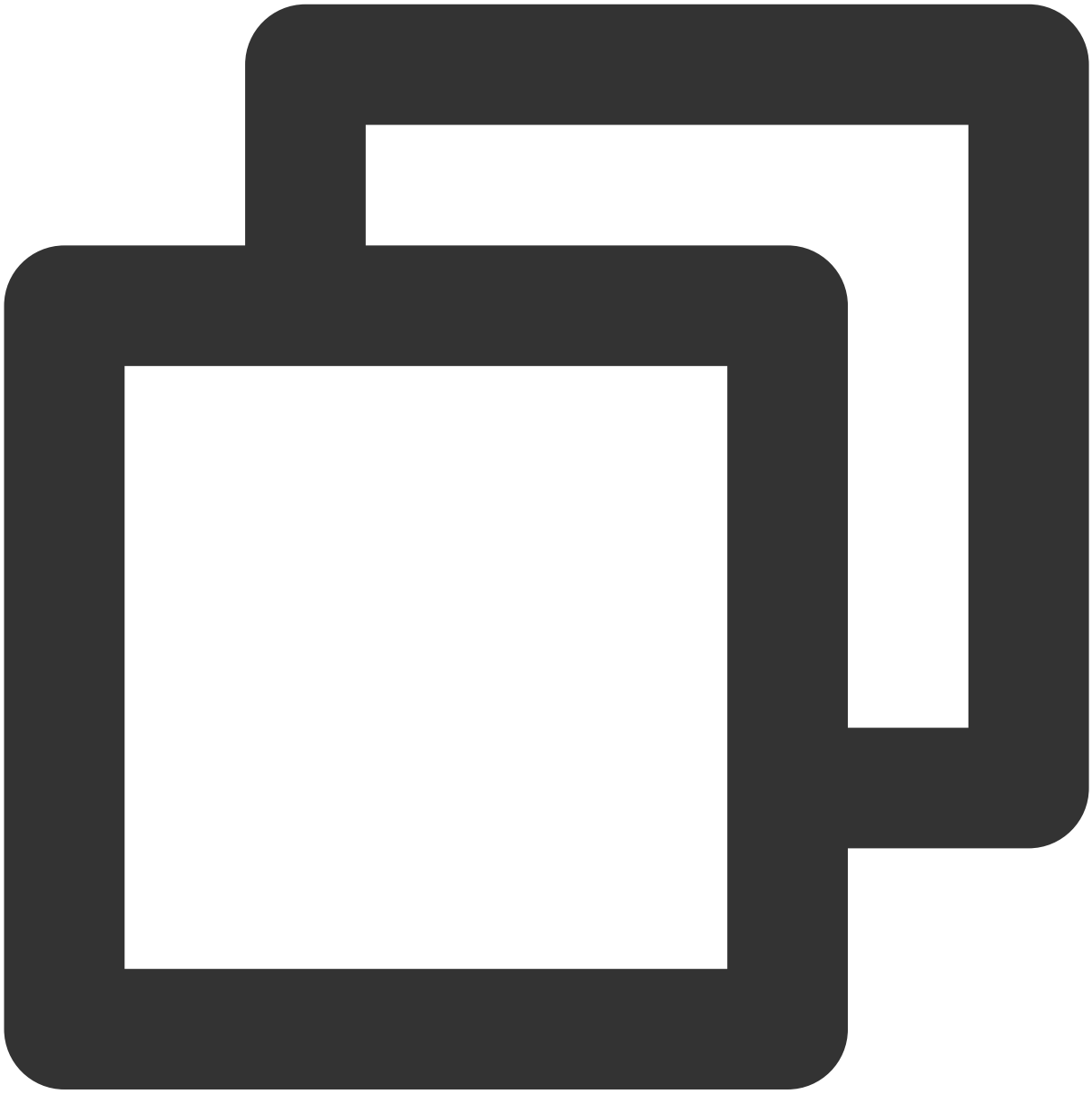


```
import { TUIStore } from '../CallService/index';
import { NAME, StoreName } from '../const/index';
import { en } from './en';
import { zh } from './zh-cn';
import { ja_JP } from './ja_JP';
import { vi } from './vi';  // Import new language file

.....

export const languageData: languageDataType = {
  en,
```

```
  'zh-cn': zh,
  ja_JP,
  vi,       // Export new language file
};
```

3. Add new LanguageType enum.

Modify the src/components/TUICallKit/src/TUICallService/const/call.ts

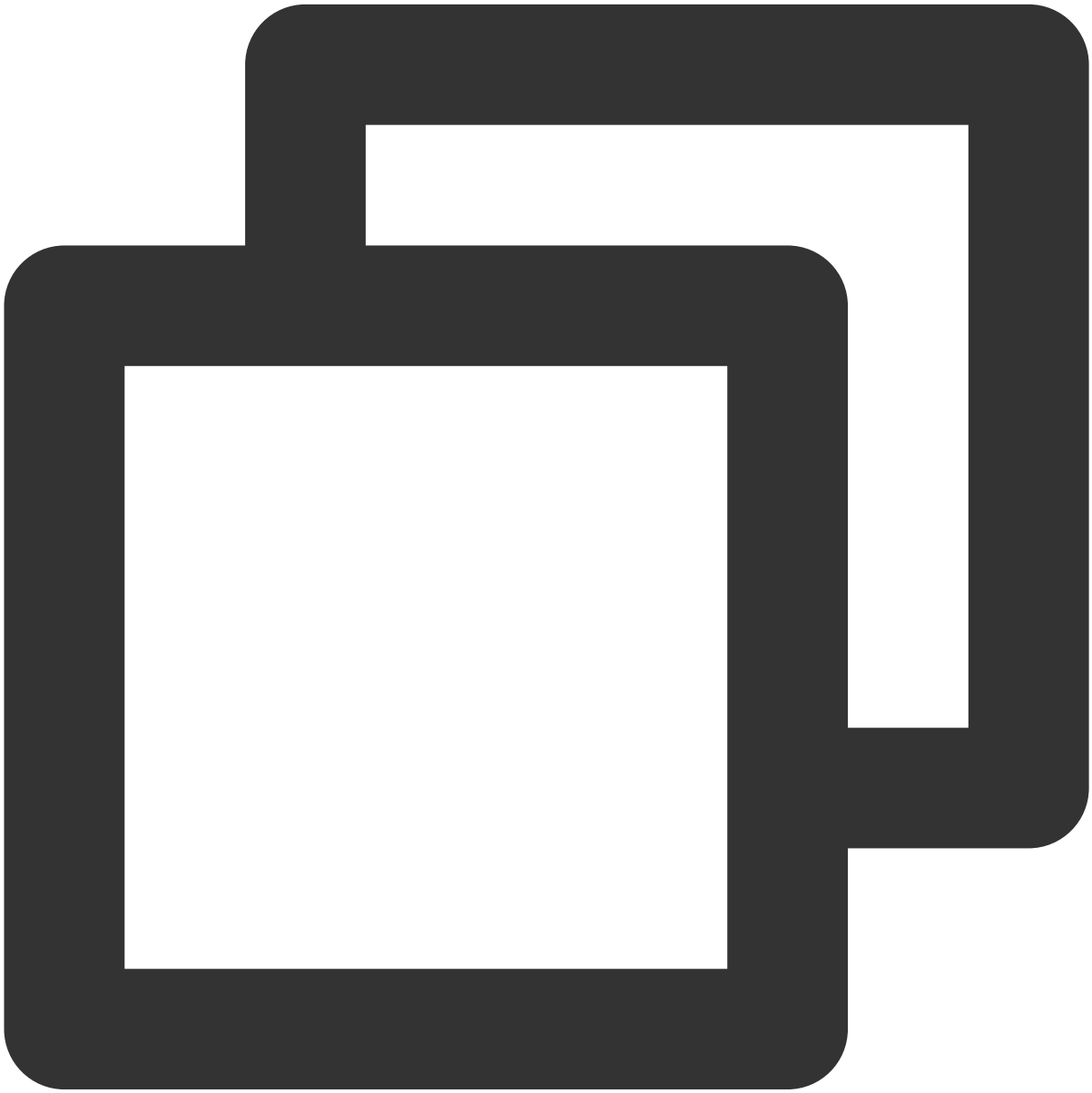


```
export enum LanguageType {
  EN = 'en',
  'ZH-CN' = 'zh-cn',
```

```
  JA_JP = 'ja_JP',
  VI = 'vi',   // Add new enum type
}
```

4. Switch Language

Switch languages in the project by calling the setLanguage interface.



```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-vue';

TUICallKitServer.setLanguage("vi");
```

iOS

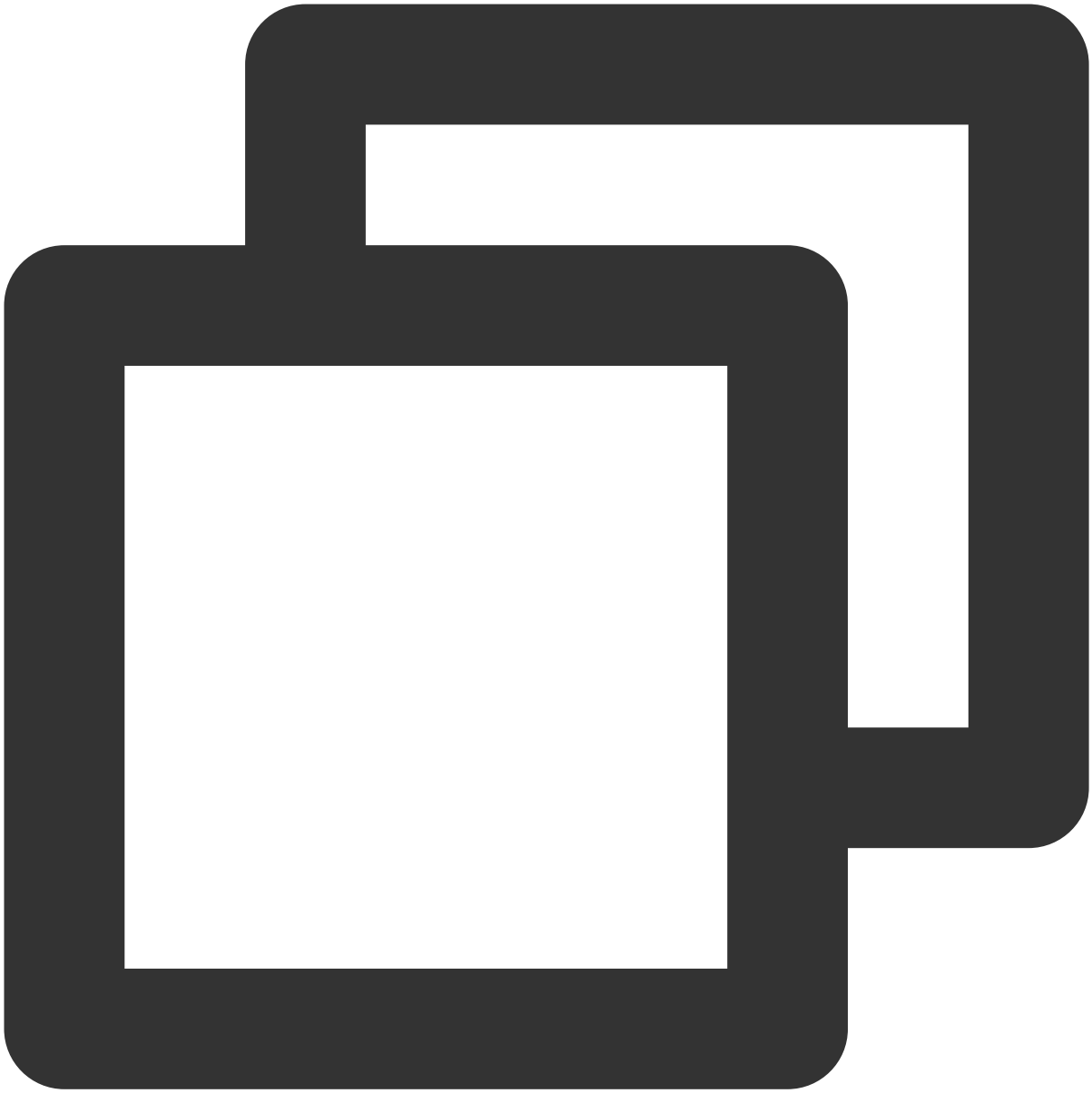
Last updated：2024-08-15 11:22:55

Supported Languages

Currently supports **Simplified Chinese, English, Japanese, and Arabic.**

Switch Language

TUICallKit defaults to the same language as the phone system . If you need to switch languages, you can use `TUIGlobalization.setPreferredLanguage` to switch languages, taking switching to English as an example:

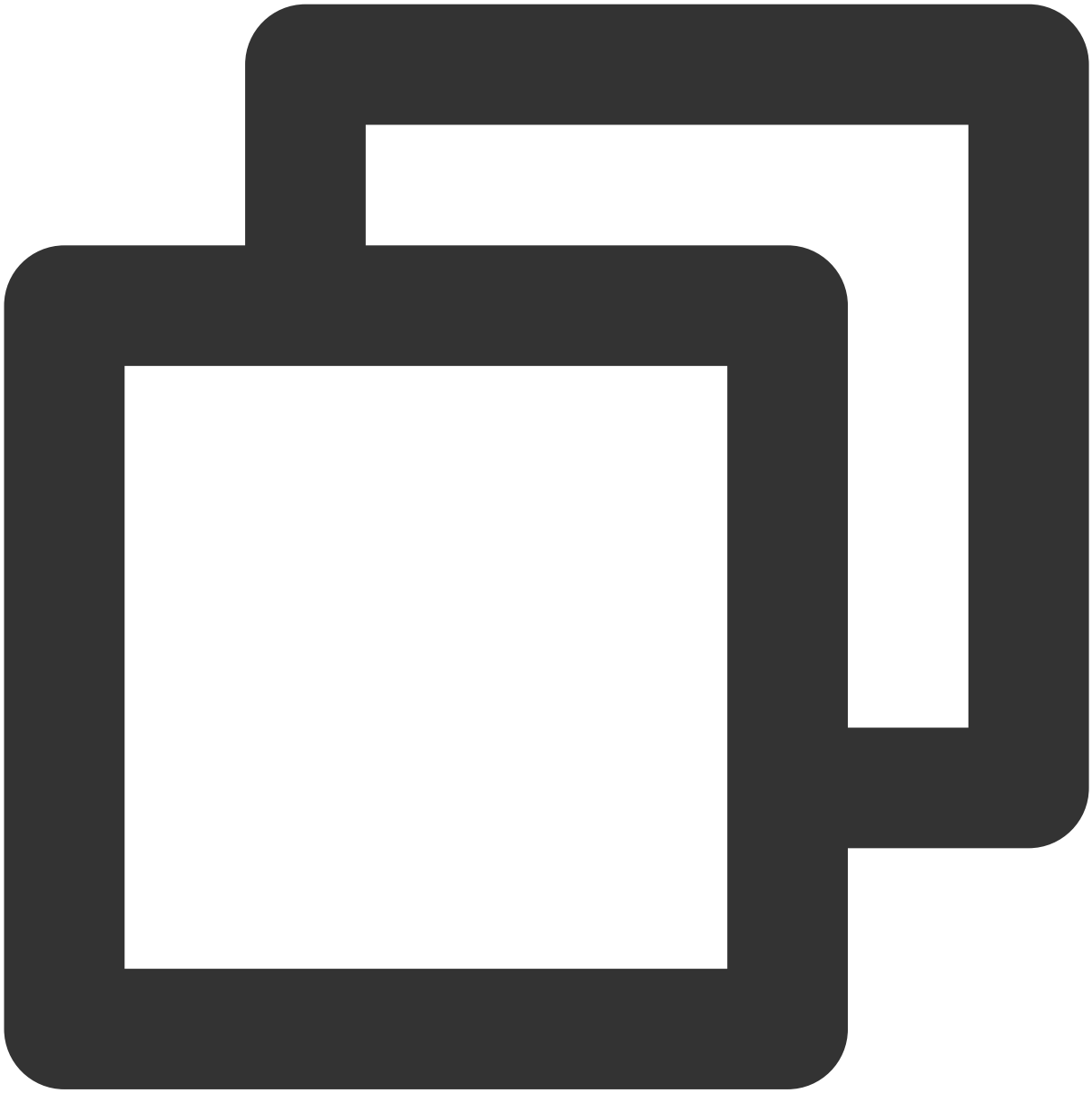


```
…
import TUICore

func steLanguage() {
    TUIGlobalization.setPreferredLanguage("en")
}
```

Add New Language

Step 1: Source Code Integration

1. Clone/download the code from GitHub.
2. Download the TUICallKit source code dependency to local in the Podfile file of the Application project.



```
target 'TUICallKitApp' do
  use_frameworks!
  …
  pod 'TUICallKit-Swift/Professional', :path => "Your Download Path/TUICallKit/iOS"

end
```

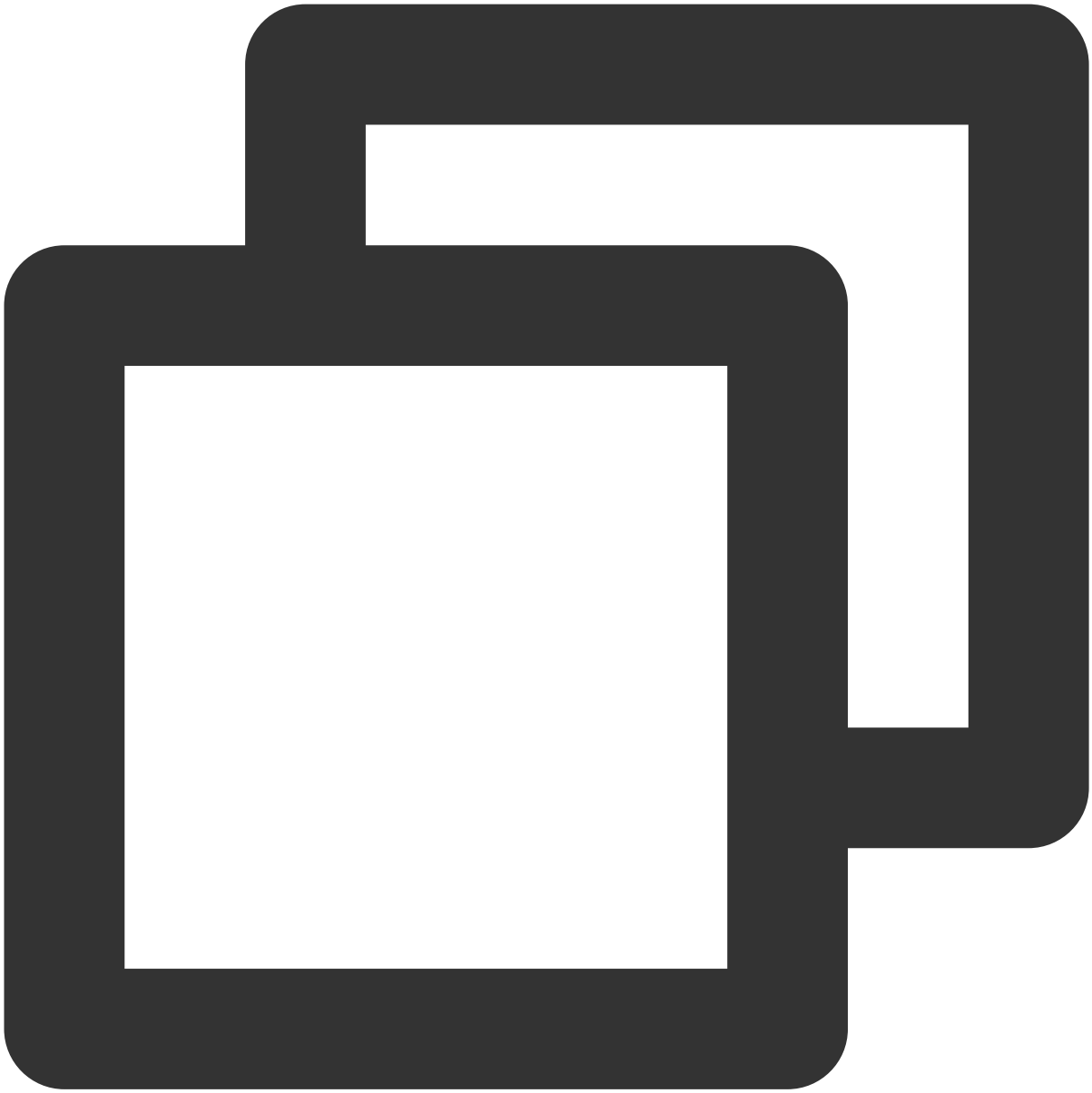
```
```

3. Run the `pod update` command to update dependencies.

Step 2: Add a new language pack

Using Spanish as an example:

1. Add a new Spanish language file.
Navigate to the `TUICallKit` source file directory, under iOS/TUICallKit-Swift/Resources, and create a new `es.lproj/Localized.strings` file.
2. Copy the content in `iOS/TUICallKit-Swift/Resources/en.lproj/Localized.strings` into the newly added `iOS/TUICallKit-Swift/Resources/es.lproj/Localized.strings` file.
3. Translate the English content in `iOS/TUICallKit-Swift/Resources/es.lproj/Localized.strings` to Spanish.
4. Navigate to the directory containing the Application project's Podfile and execute the `pod install` command to update dependencies.



```
  …
import com.tencent.qcloud.tuicore.TUIThemeManager;

public class MainActivity extends BaseActivity {
    @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      Locale locale = new Locale("es");
      TUIThemeManager.addLanguage("es", locale);
      …
```

```
        }
        ...
}
```

Android

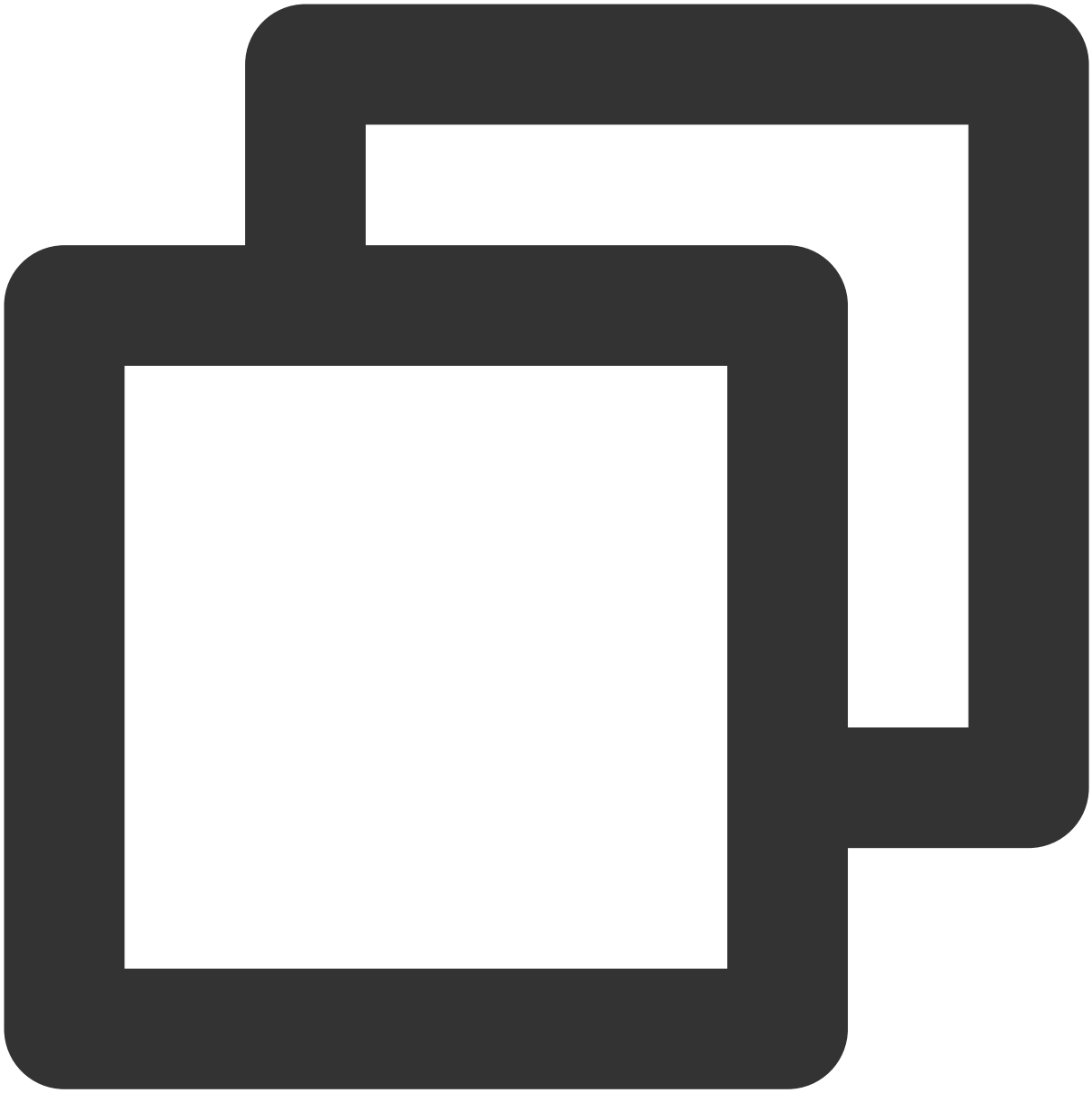
Last updated：2024-08-15 11:22:55

Supported Languages

Currently supports **Simplified Chinese, English, Japanese, and Arabic.**

Switch Language

TUICallKit Default Language matches the mobile system. If you need to switch languages, you can use `TUIThemeManager.getInstance().changeLanguage` to change the language. For example, to switch to English:



```
…
import com.tencent.qcloud.tuicore.TUIThemeManager;

public class MainActivity extends BaseActivity {
    @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      TUIThemeManager.getInstance().changeLanguage(getApplicationContext(), "en");
      …
    }
```

```
        ...
}
```

Add New Language

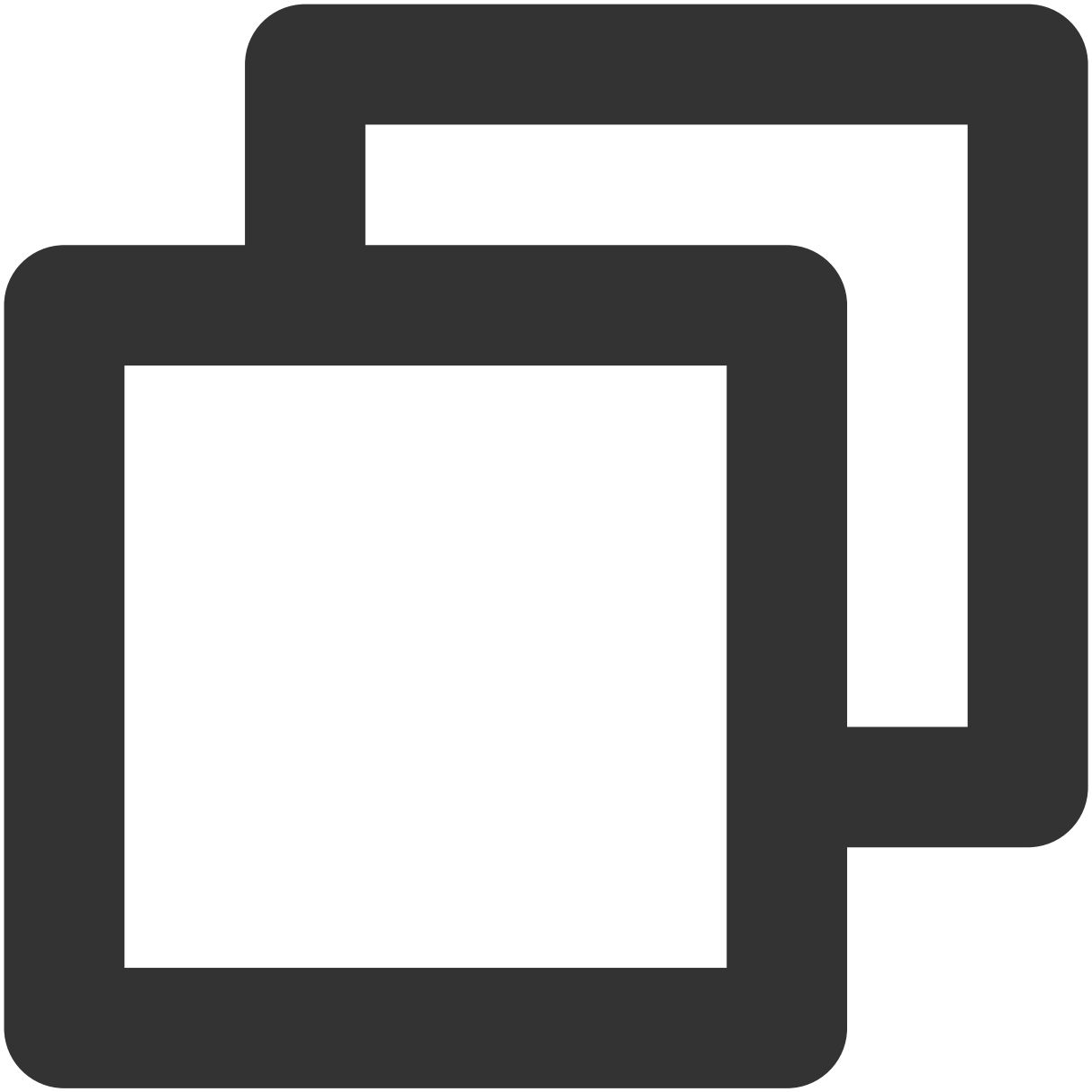
Step 1: Source Code Integration

1. Clone or download the code from GitHub, then copy the tuicallkit-kt subdirectory under the Android directory to the same level under your current project's app directory, as shown below.

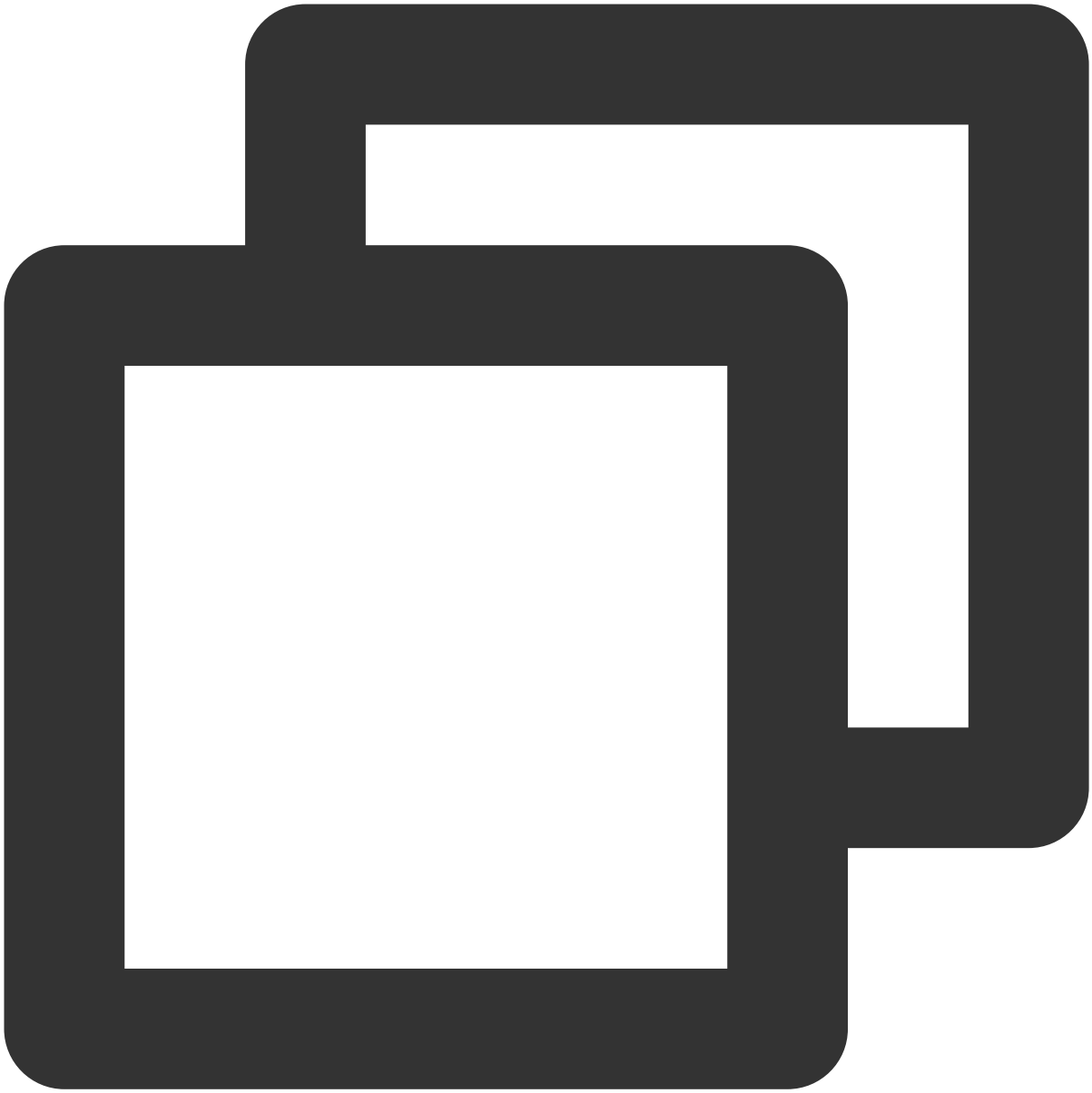
2. Find the `settings.gradle.kts (or settings.gradle)` file in your project's root directory, add the following code to import the `tuicallkit-kt` component into your project.

setting.gradle.kts

settings.gradle



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

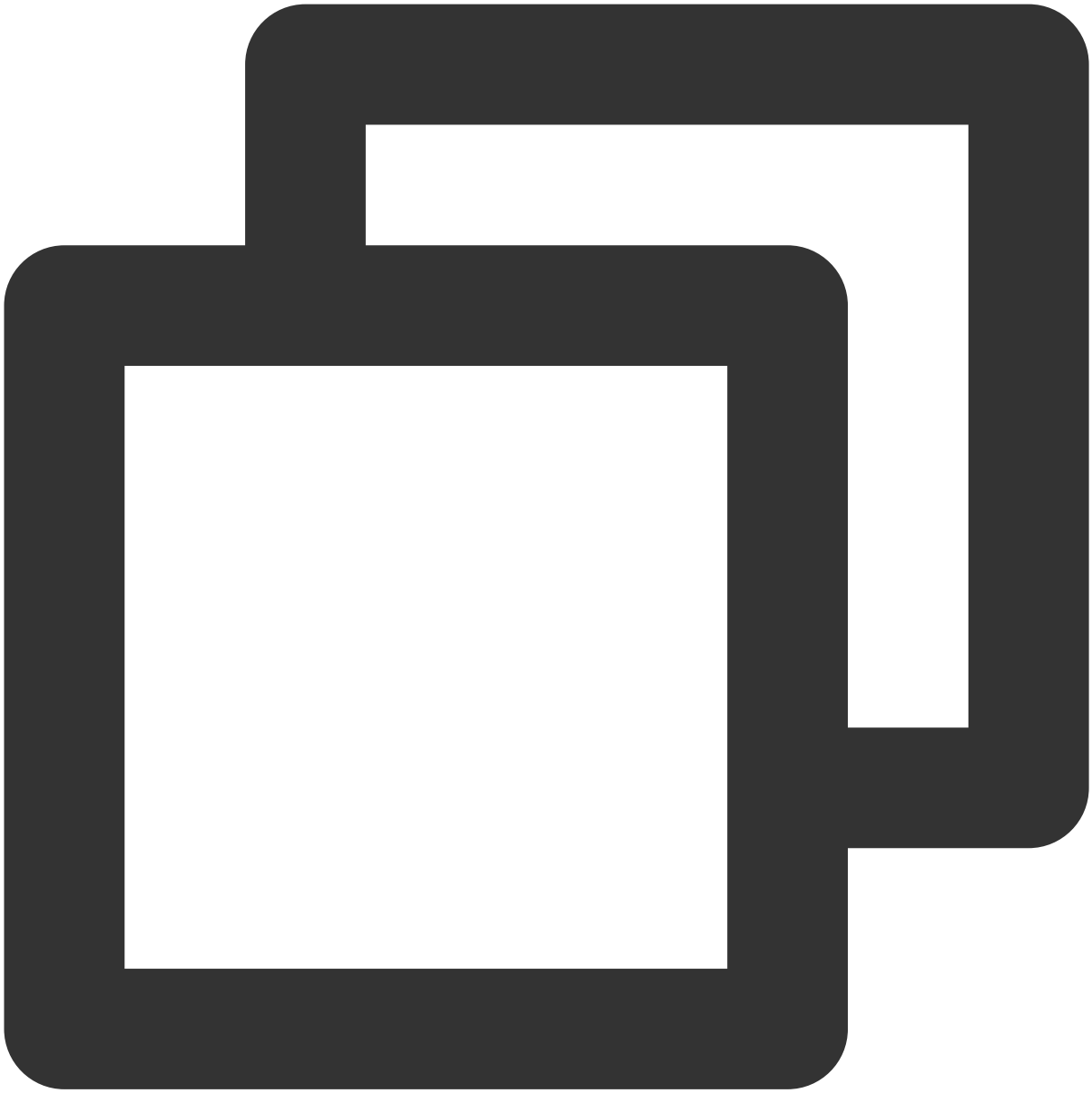


```
include ':tuicallkit-kt'
```

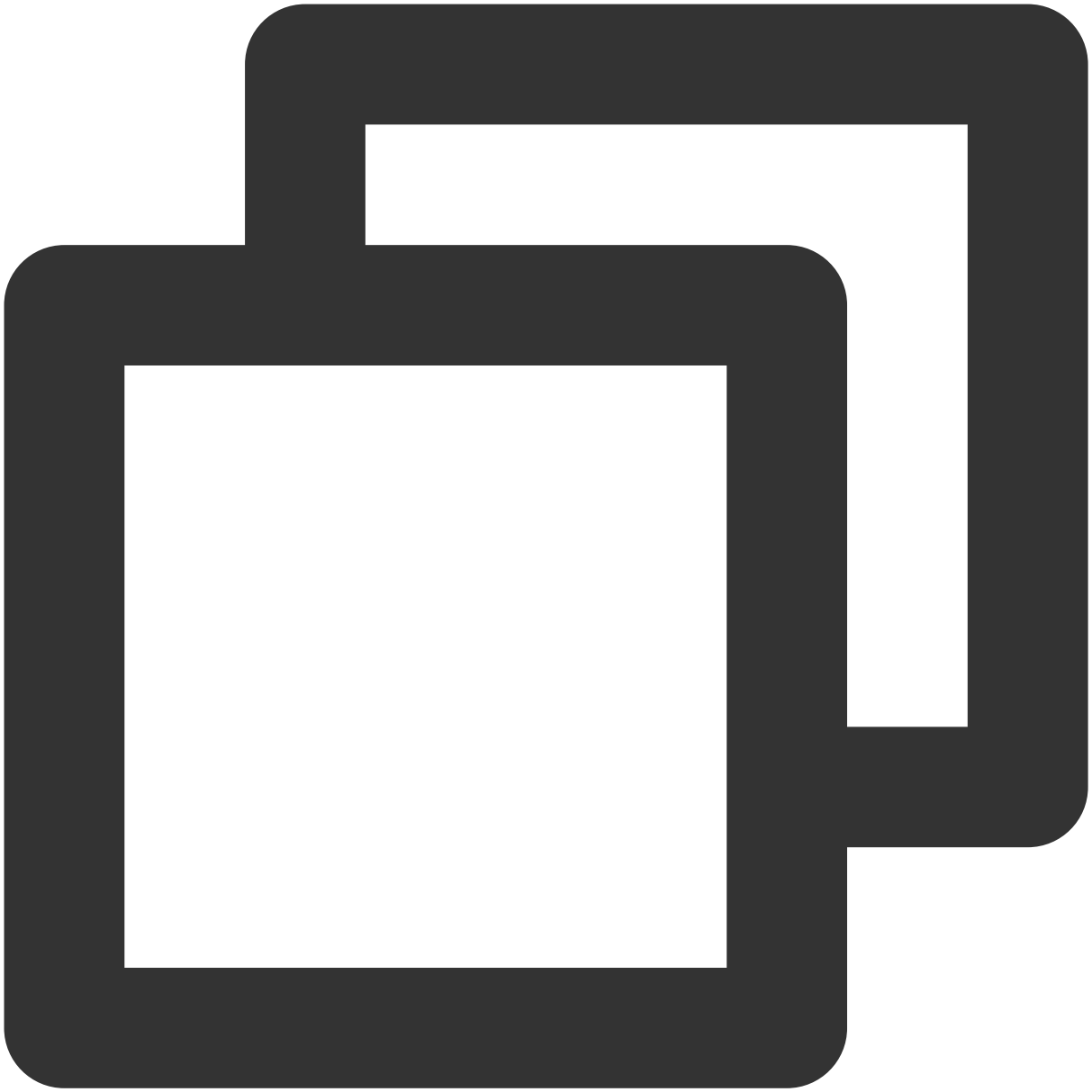
3. In the app directory, find the `build.gradle.kts (or build.gradle)` file, add the following code in the `dependencies` section to declare the current app's dependency on the newly added component.

build.gradle.kts

build.gradle



```
dependencies {
    api(project(":tuicallkit-kt"))
}
```



```
dependencies {
    api project(':tuicallkit-kt')
}
```

Step 2: Add a new language pack

Using Spanish as an example:

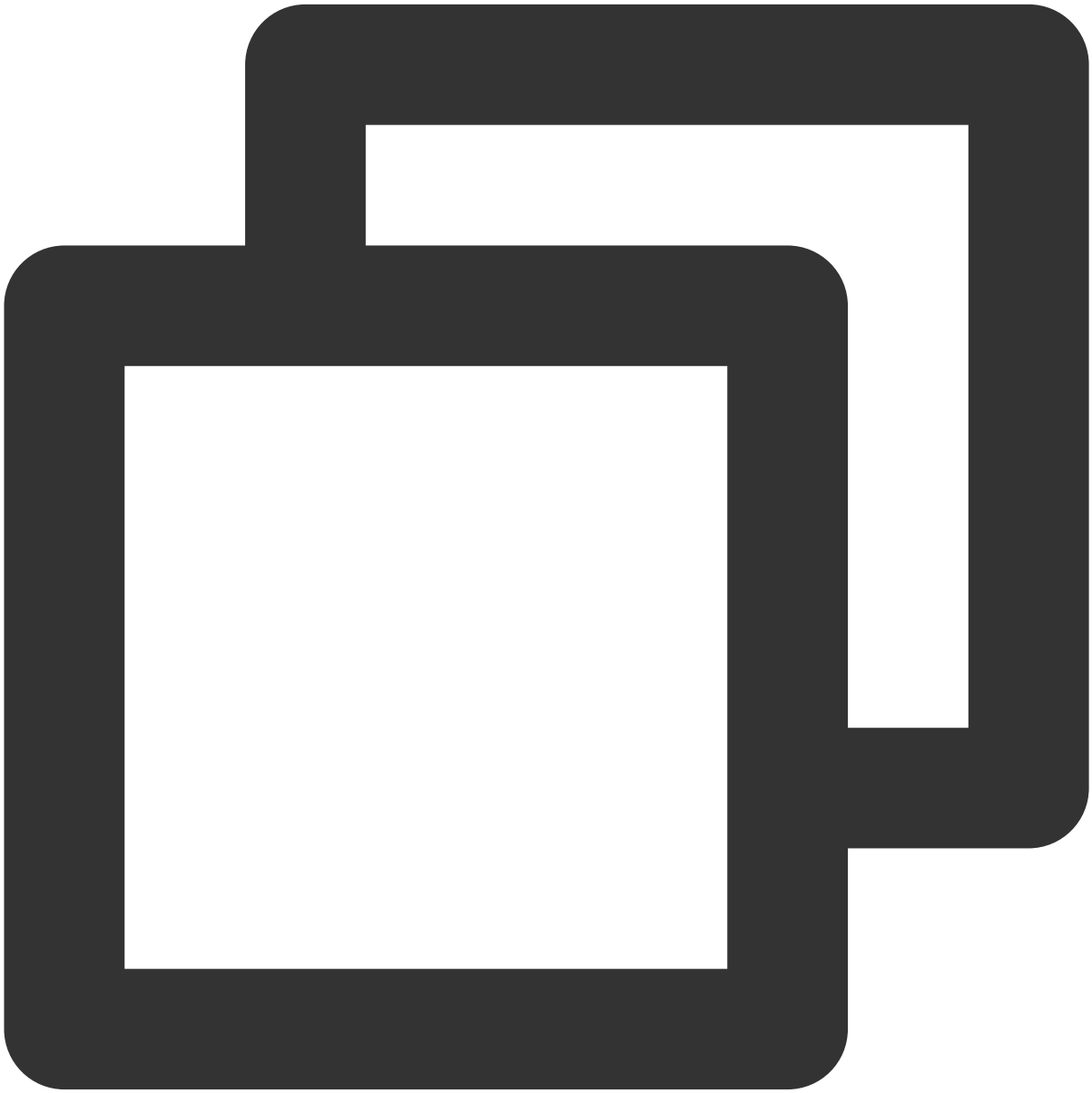
1. Add a new Spanish language file.

Navigate to the `TUICallKit` source code file directory under the `src/main/res` directory, and add a new `value-es/strings.xml` file .

2. Copy the contents of `src/main/res/values-en/strings.xml` to the newly added `src/main/res/values-es/strings.xml` file.

3. Translate the English in `src/main/res/values-es/strings.xml` to Spanish.

4. Add new language.



```
    ...
import com.tencent.qcloud.tuicore.TUIThemeManager;
```

```
public class MainActivity extends BaseActivity {
    @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      Locale locale = new Locale("es");
      TUIThemeManager.addLanguage("es", locale);
      …
    }
    …
}
```

Flutter

Last updated：2024-08-15 11:22:55

Supported Languages

Currently supported languages are **Simplified Chinese, English, and Japanese,** with the default language being **English .**

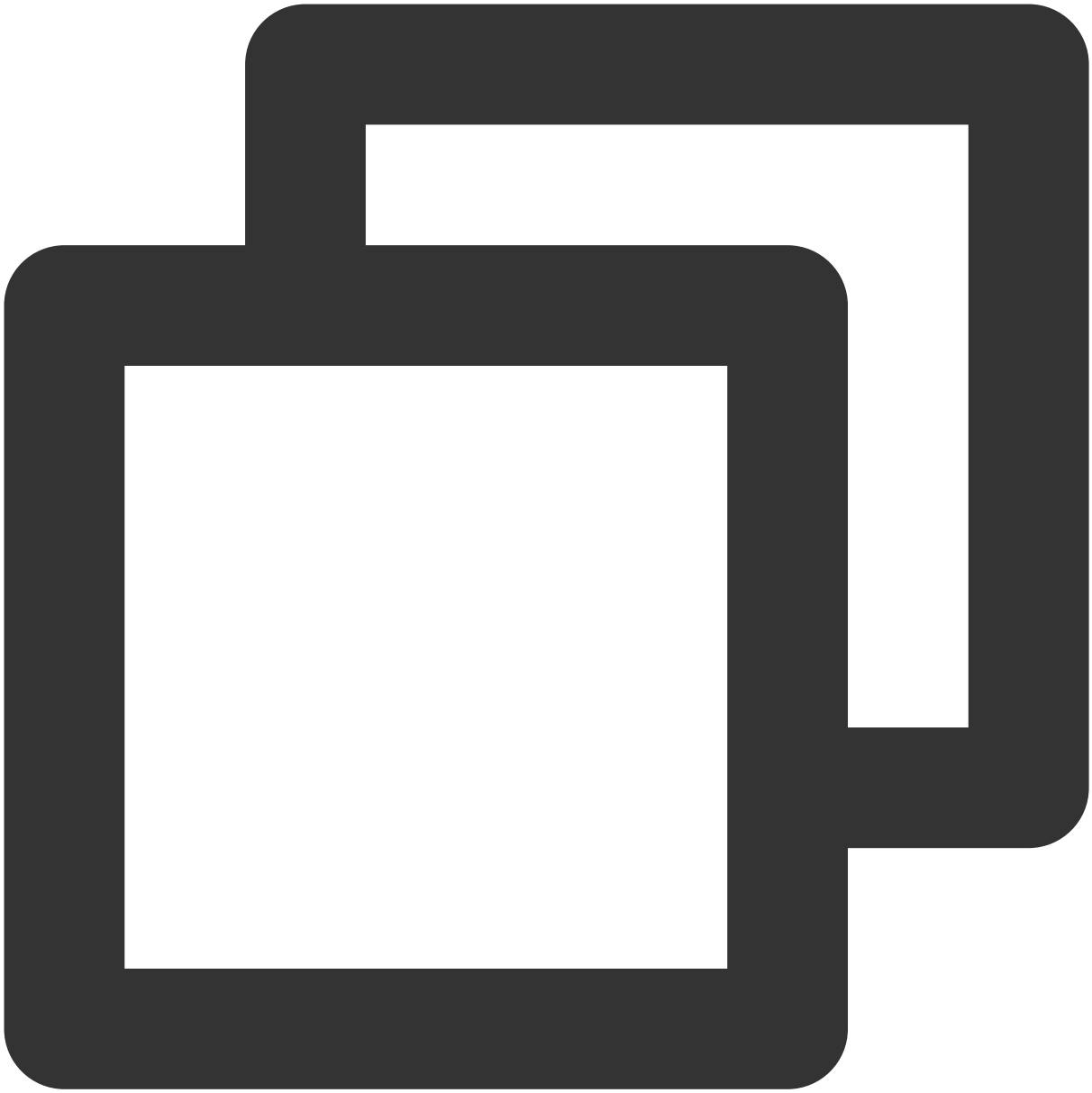
Switch Language

`TUICallKit` does not provide a separate interface for language switching, `TUICallKit` automatically switches languages based on the current `Application` 's `MaterialApp` (or CupertinoApp, etc. style components) language setting. Simply change the language used by `MaterialApp` (or CupertinoApp, etc. style components).

Add New Language

Step 1: Source Code Integration

1. Download Source Code
Go to `https://pub.dev/packages/tencent_calls_uikit` to download the latest `TUICallKit` source code.
2. Depend on Local Source Code
In the `Application` project's `pubspec.yaml` file, modify `TUICallKit` to local dependency:



```
dependencies:
  tencent_calls_uikit:
    path: /TUICallKit local_path/
```

Step 2: Add a new language pack

Using Spanish as an example:

1. Add a new Spanish language file.

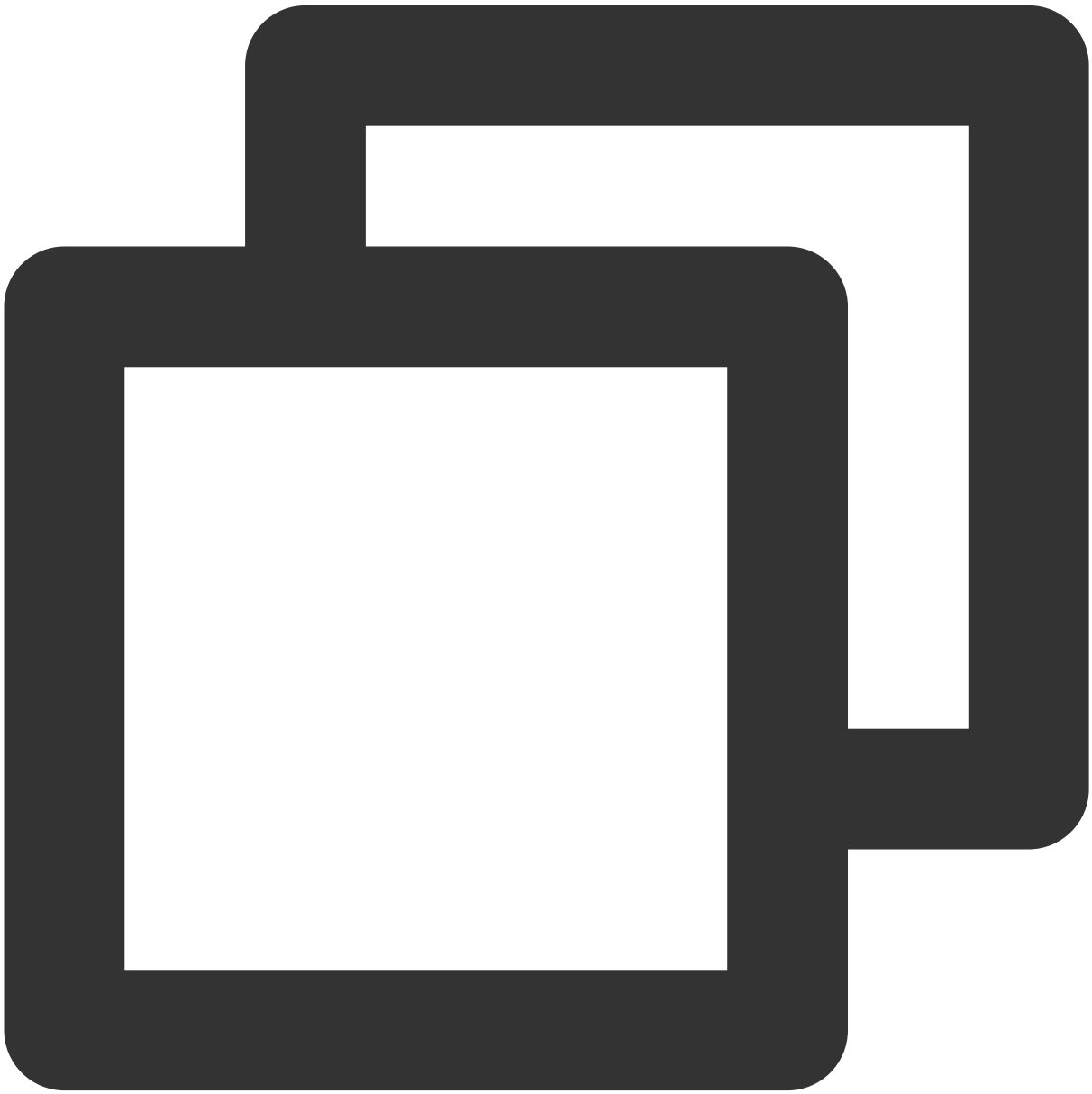
Go to the `TUICallKit` source code directory's `lib/src/i18n` folder and add `strings_es.i18n.json` .

2. Copy the contents from `lib/src/i18n/strings.i18n.json` to the newly added `lib/src/i18n/strings_es.i18n.json` file.

3. Translate the English content in `lib/src/i18n/strings_es.i18n.json` to Spanish.

4. Update Translation Package

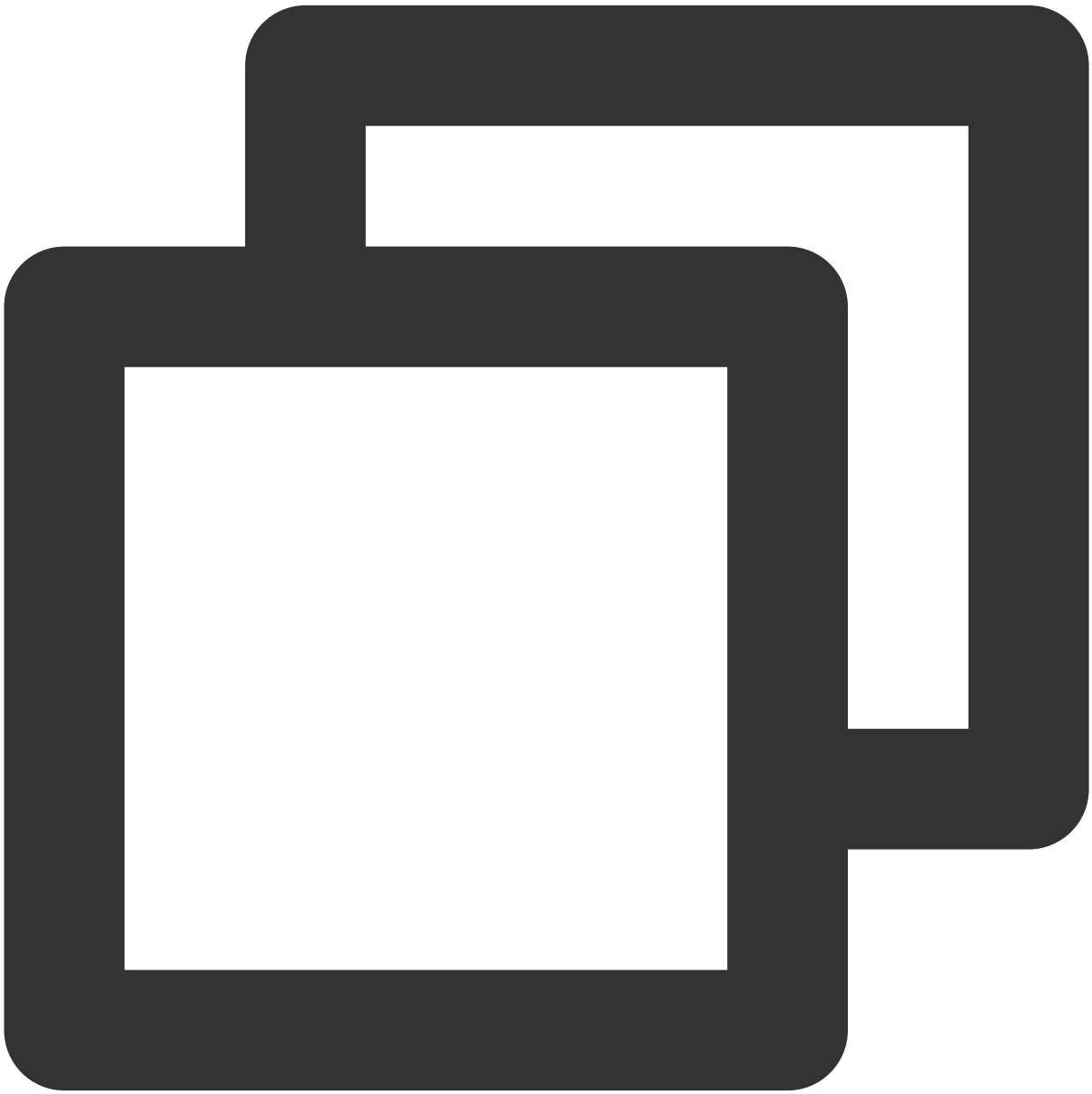
In the `TUICallKit` source code directory, go to TCCLI , and run the following commands to update the translation package:



```
flutter pub add fast_i18n
flutter pub run fast_i18n
```

5. Update the language adaptation method for `TUICallKit` .

Navigate to `lib/src/i18n/i18n_utils.dart` source file and modify the `setLanguage` method as follows:



```
static setLanguage(Locale currentLocale) {
switch (currentLocale.languageCode) {
```

```
    case 'zh':
      {
        CallKitI18nUtils(null, 'zh');
        break;
      }
    case 'en':
      {
        CallKitI18nUtils(null, 'en');
        break;
      }
    case 'ja':
      {
        CallKitI18nUtils(null, 'ja');
        break;
      }
    // Add case 'es'
    case 'es':
      {
        CallKitI18nUtils(null, 'es');
        break;
      }
  }
}
```

uni-app（Android&iOS）

Last updated：2024-08-15 11:22:55

Supported Languages

Currently supports **Simplified Chinese, English, Japanese, and Arabic.**

Switch Language

TUICallKit Language is consistent with the mobile system .

Add New Language

uni-app(client) does not support adding new languages at the moment.

If you need to add a new language, please send your feedback via email: info_rtc@tencent.com .

Server APIs (TUICallKit)
Call Status Callback
Call Status Callback

Last updated：2024-04-18 16:25:27

To facilitate refined control of your call services, the TRTC Call (TUICallKit) offers call status callbacks. Your business backend can use these callbacks to peek at users' call results in real-time, such as missed calls, rejections, etc., and based on this, perform real-time data analytics and other operations. For calling configuration methods, see Callback Configuration API List.

Use Conditions

Only applications (SDKAppId) that have activated the **Group Call Version** of TRTC Call can use the call status callback. You can also activate the trial version for free to test. For version descriptions and activation instructions, refer to Activate the Service.
Currently, call status callbacks are available only in specific versions of TUICallKit across various platforms, as detailed in the table below:

Platform/Framework	Version number
Android/iOS/Flutter/uni-app (client)	≥ 1.7.1
Web	≥ 1.4.6
WeChat Mini Program	≥ 1.5.1

Note:
Only after all participating platforms/frameworks are upgraded to the versions mentioned above, can the corresponding call information be peeked at in the console.

Notes

To enable the callback, you must configure the callback URL and enable the switch for this command. Please refer to Create Callback Configuration.
The direction of the callback is from the Callkit backend to the app backend via an HTTP POST request.

After receiving the callback request, the app backend must check whether the SDKAppID contained in the request URL is consistent with its own SDKAppID.

Scenarios that may trigger this callback

Actions performed by app users through the client during a call, such as hanging up, etc.

Timing of the callback

After the call ends.

Possible callback results

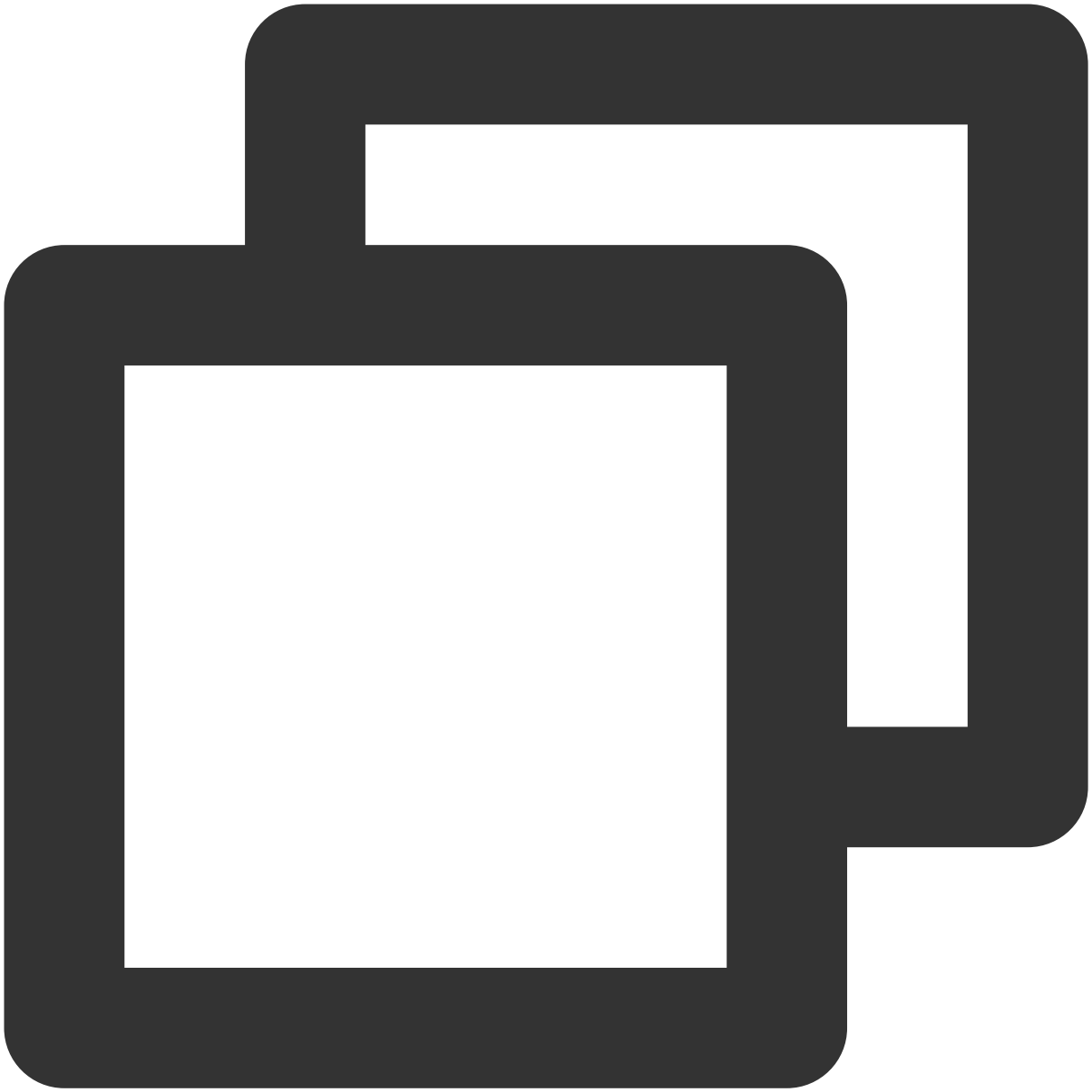
Callback status	Result indication
Missed call: Recipient timeout before answering	not_answer
Declined call: Recipient declined the call	reject
Busy Line: Busy line	call_busy
Cancel: Caller canceled the call before connecting	cancel
Completion: Call connected and ended normally	normal_end
Interruption: Call interrupted due to network or other reasons	interrupt

API description

Request URL

In the following example, the callback URL configured in the app is `https://www.example.com` .

Example:



```
$http://www.example.com?sdkappid=$sdkappid&command=$command&contenttype=json&client
```

Request parameters

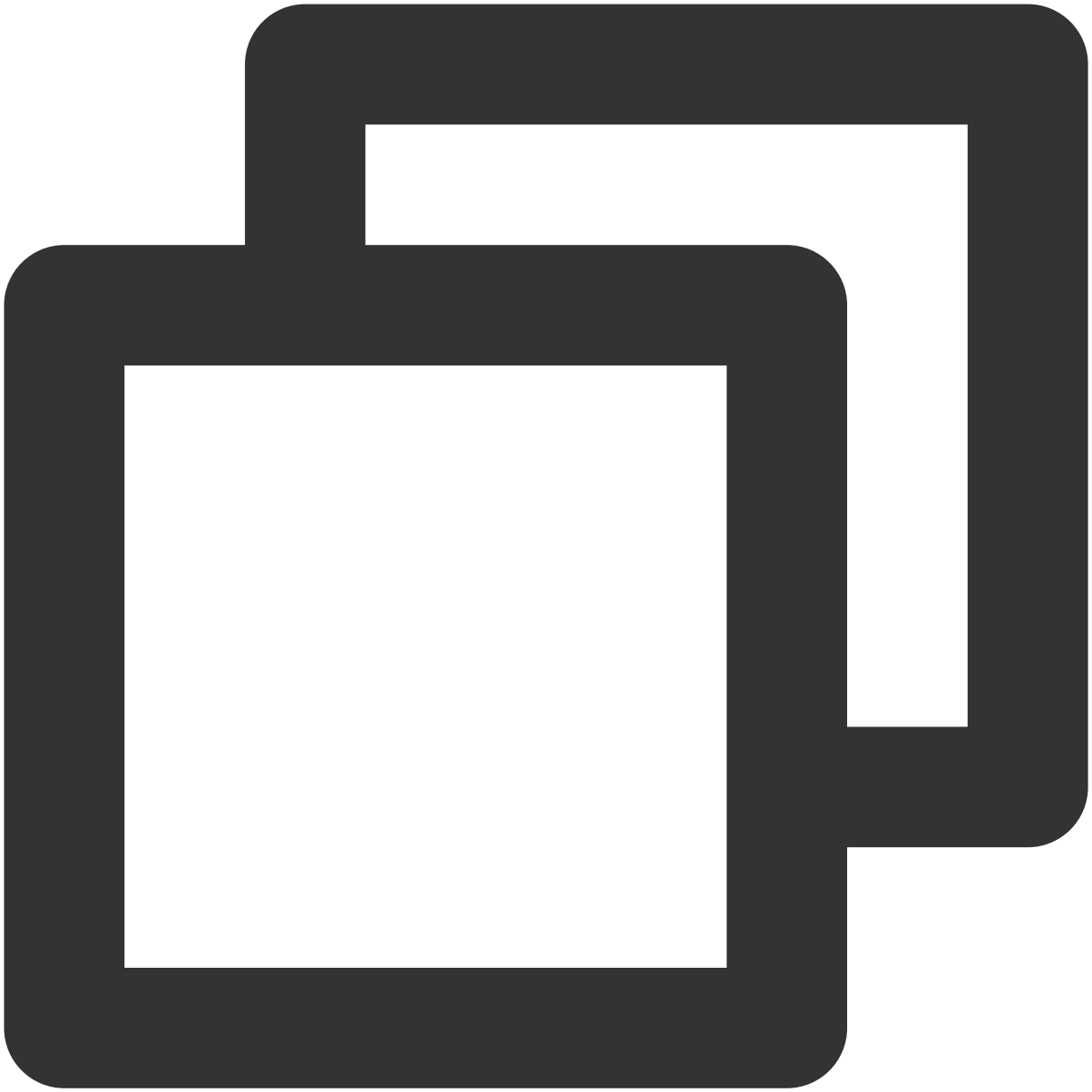
Parameter	Description
http	Request protocol is HTTPS or HTTP, request method is POST
www.example.com	Callback URL

sdkappid	SDKAppID assigned by the Instant Messaging console when an application is created
command	Please refer to: Callback command list
contenttype	Fixed value: json
clientip	Client IP, such as 127.0.0.1
optplatform	Client platforms may include iOS, Android, Web, miniProgram

The specific callback content is included in the HTTP request body. See the callback examples below for details.

Callback Example

Webhook request example:



```
POST /?sdkappid=8888888&command=call_end&contenttype=json&clientip=127.0.0.1&optpla
Host: www.example.com
Content-Length: 337
{
    "UserId": "Alice",
        "RoomId": "Alice's Room",
    "TotalNum": 2,
    "MediaType": "audio",
    "CallType": "single",
    "CallId": "aheahfo-eqwnknoihfsd-qweqf",
    "Role": "caller",
```

```
    "CallResult": "normal_end",
        "EventTime": 170485688,
    "StartCallTs": 1704856873,
    "AcceptTs": 1704856876,
    "EndTs": 1704856885
}
```

Request packet fields

Field	Type	Description
UserId	String	Operating User ID
RoomId	String	Operating Room ID
TotalNum	Integer	Number of Participants in Call
MediaType	String	Media Type: `Audio` Audio Call `Video` Video Call
CallType	String	Call Type: `Single` Audio Call `Group` Video Call
CallId	String	Call Unique ID
Role	String	role : `Caller` User ID `Callee` User ID
CallResult	String	Call Result, only filled in during the end event, otherwise empty: `Cancel` : Caller canceled the call before connection `Reject` Declined: Recipient declined the call `Not_answer` Missed call: Recipient did not answer in time `Normal_end` Completion: Call connected and ended normally `Call_busy` Busy line: Busy line during call `Interrupt` Interruption: Call interrupted due to network or other reasons
EventTime	Integer	Timestamp of operation (second-level)
StartCallTs	Integer	Timestamp when call is initiated (second-level) only returned on normal_end
AcceptTs	Integer	Timestamp when call is answered (second-level) only returned on normal_end

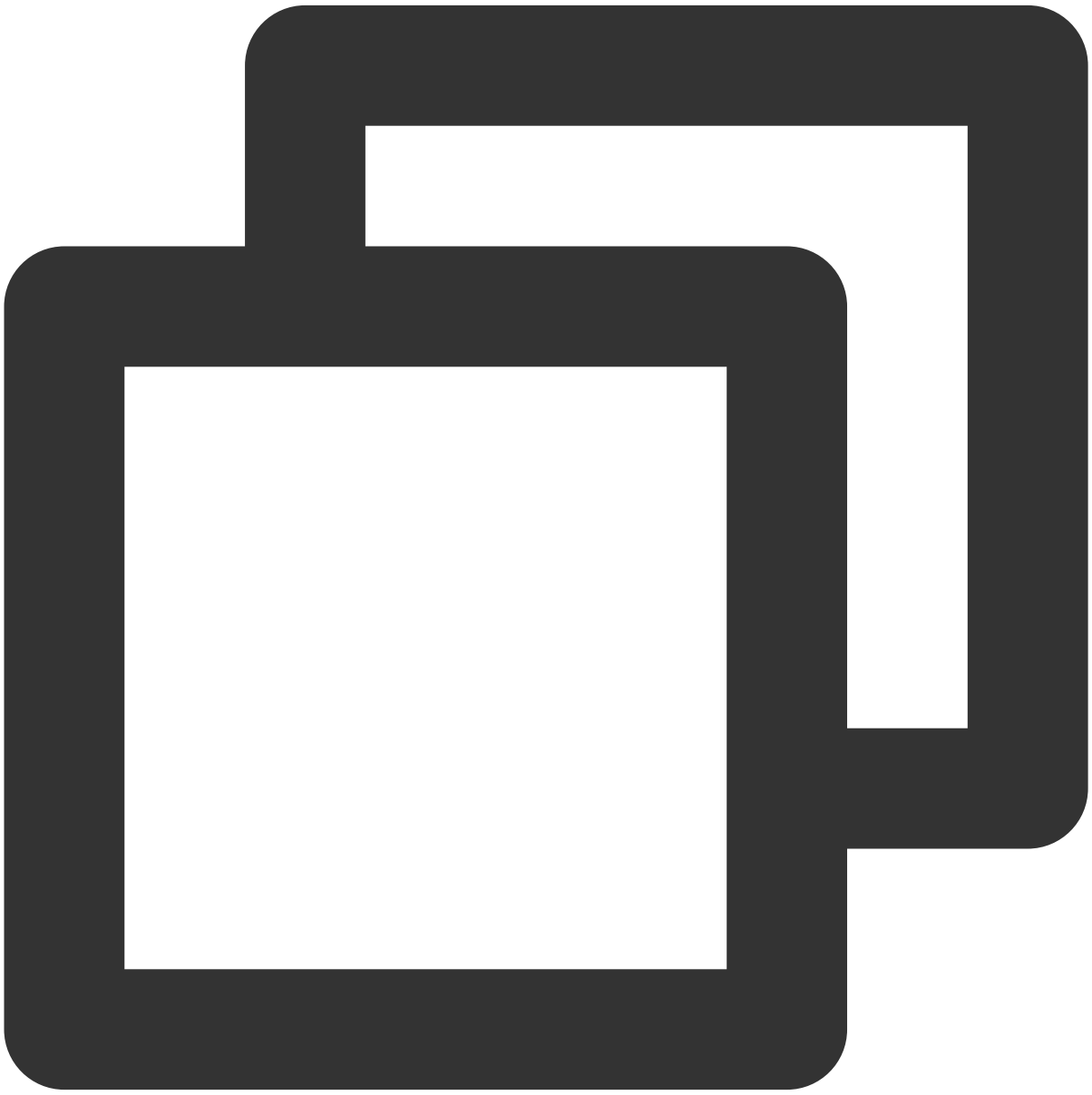
EndTs	Integer	Timestamp when call ends (second-level) only returned on normal_end

Note:

CallResult field shows all types only for one-on-one calls, group chat only has `normal_end` .

Callback Response Example

A callback response packet is sent after the app backend synchronizes the data.



```
{
    "ErrorCode": 0,
    "ErrorMessage": "Success"
}
```

Response Packet Field Description

Field	Type	Description
ErrorCode	Integer	0 indicates success, all other values indicate failure
ErrorMessage	String	Your server response can carry error information as defined

Note:

It is recommended that your server responds with the correct response packet promptly after receiving the request packet correctly, otherwise, it will trigger a system retry. The retry mechanism is as follows.

Retry mechanism

When to trigger a retry

ErrorCode = 0 is considered a successful callback, otherwise, it will trigger a retry.

An HTTP request error to your appServer will also trigger a retry.

Retry interval

Retry interval: `0s, 1s, 1s, 2s, 3s, 5s` . If all attempts fail, the process will be abandoned.

Error code

Error code	Description
0	Request succeeded
50001	The current application needs to purchase the TUICallKit Group Call Version Package to use
999	Callback does not exist
70001	Invalid request parameters, please check whether mandatory parameters are missing or incorrectly entered
70002	UserSig is invalid

70003	UserSig has expired
70004	Requesting user is not a Super Administrator
70005	Request frequency limited
Unknown error code	System internal error, please Submit a ticket to contact technical personnel

Call Event Callback

Last updated：2024-04-18 16:26:28

To facilitate refined control of your call services, the TRTC Call (TUICallKit) offers call status callbacks. Your business backend can use these callbacks to peek at users' call results in real-time, such as missed calls, rejections, etc., and based on this, perform real-time data analytics and other operations. For calling configuration methods, see Callback Configuration API List.

Use Conditions

Only applications (SDKAppId) that have activated the **Group Call Version** of TRTC Call can use the call status callback. You can also activate the trial version for a free trial. For version descriptions and activation instructions, see Activate the Service.
Currently, call status callbacks are available only in specific versions of TUICallKit across various platforms, as detailed in the table below:

Platform/Framework	Version number
Android/iOS/Flutter/uni-app (client)	≥ 1.7.1
Web	≥ 1.4.6
WeChat Mini Program	≥ 1.5.1

Note:
Only after all participating platforms/frameworks are upgraded to the versions mentioned above, can the corresponding call information be peeked at in the console.

Notes

To enable the callback, you must configure the callback URL and enable the switch for this command. Please refer to Create Callback Configuration.
The direction of the callback is from the Callkit backend to the app backend via an HTTP POST request.
After receiving the callback request, the app backend must check whether the SDKAppID contained in the request URL is consistent with its own SDKAppID.

Scenarios that may trigger this callback

Actions generated by an app user during a call via the client, such as answering and hanging up.

Timing of the callback

After operations by the user such as answering and hanging up.

Possible callback results

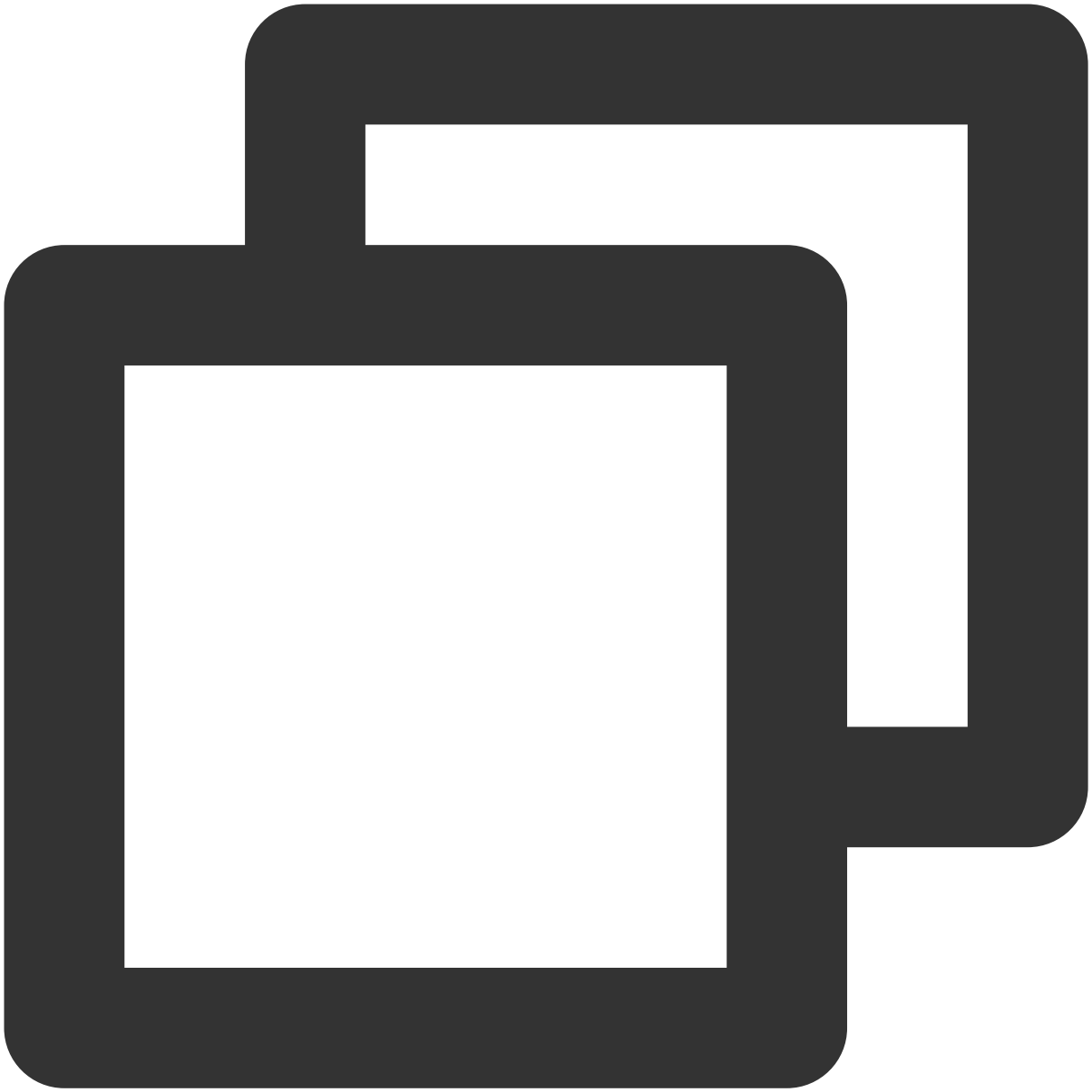
Please refer to Callback Command - Call Event Callback.

API description

Request URL

In the following example, the callback URL configured in the app is `https://www.example.com` .

Example:



```
$http://www.example.com?sdkappid=$sdkappid&command=$command&contenttype=json&client
```

Request parameters

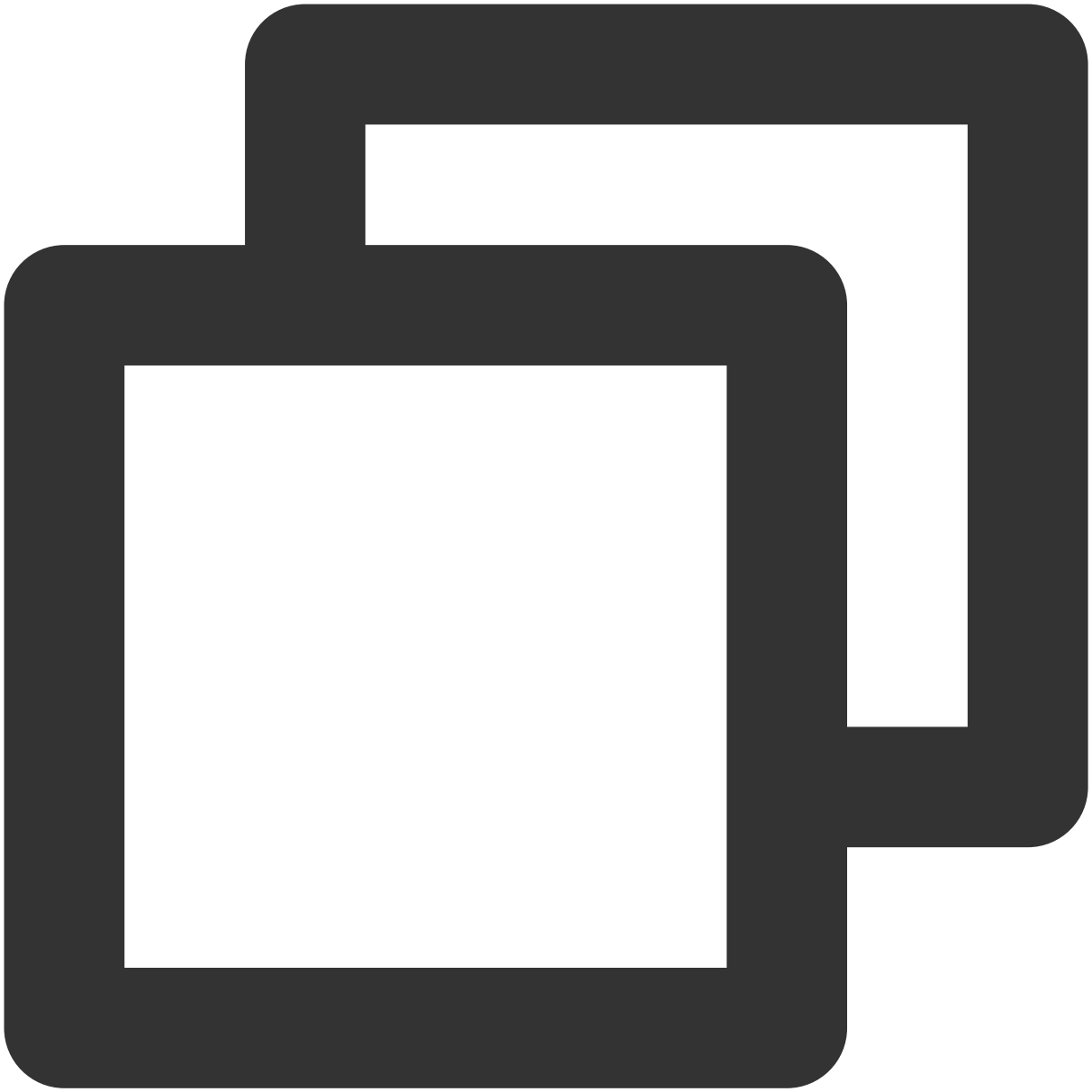
Parameter	Description
http	Request protocol is HTTPS or HTTP, request method is POST
www.example.com	Callback URL

sdkappid	SDKAppID assigned by the Instant Messaging console when an application is created
command	Please refer to: Callback command list
contenttype	Fixed value: json
clientip	Client IP, such as 127.0.0.1
optplatform	Client platforms may include iOS, Android, Web, miniProgram

The specific callback content is included in the HTTP request body. See the callback examples below for details.

Callback Example

Webhook request example:



```
POST /?sdkappid=8888888&command=caller_start_call&contenttype=json&clientip=127.0.0
Host: www.example.com
Content-Length: 337
{
    "UserId": "Alice",
        "RoomId": "Alice's Room",
    "TotalNum": 2,
    "MediaType": "audio",
    "CallType": "single",
    "CallId": "aheahfo-eqwnknoihfsd-qweqf",
    "Role": "caller",
```

```
    "Event": "start_call",
    "CallResult": "",
        "EventTime": 1704695566,
    "StartCallTs": 1704856873,
    "AcceptTs": 1704856876,
    "EndTs": 1704856885
}
```

Request packet fields

Field	Type	Description
UserId	String	Operating User ID
RoomId	String	Operating Room ID
TotalNum	Integer	Number of Participants in Call
CallType	String	Call Type: `Single` Audio Call `Group` Video Call
CallId	String	Call Unique ID
Role	String	role : `Caller` User ID `Callee` User ID
Event	String	Call Event: `start_call` Caller initiates call `call_accepted` Caller answers call `call_missed` Caller missed call `call_rejected` Caller rejects call `call_busy` Caller line busy `cancel_call` Caller cancels call `call_failed` Caller failed to initiate call `call_end` Caller call ended normally `call_interrupted` Caller call interrupted `receive_call` Callee receives call `accept_call` Callee answers call `not_answer_call` Callee does not answer `reject_call` Callee rejects call `ignore_call` Called party ignores call `call_canceled` Called party cancels call `call_end` Normal termination of called party's call

		`call_interrupted` Called party's call abnormally interrupted `invite_user` Midway invite user `join_in_group_call` Join the call midway
StartCallTs	Integer	Timestamp when call is initiated (second-level) only returned on normal_end
AcceptTs	Integer	Timestamp when call is answered (second-level) only returned on normal_end
EndTs	Integer	Timestamp when call ends (second-level) only returned on normal_end

Note:

CallResult field shows all types only for one-on-one calls, group chat only has `normal_end` .

Callback Configuration
API List for Callback Configuration

Last updated：2024-04-18 16:28:42

Callback Information Configuration

Feature Overview	API
Create Callback	v1/callback/set
Query Callback	v1/callback/get
Update Callback	v1/callback/update
Deleting callback	v1/callback/delete

Callback Command Word List

Call Status Callback	Call Event Callback
`cancel` Cancel: Caller cancels the call before it is answered `reject` Declined: The callee rejects the call `not_answer` Missed call: The callee does not answer within the timeout period `normal_end` Complete: The call is connected and ends normally `call_busy` Busy Line: The call is on a busy line `interrupt` Interruption: The call is interrupted due to network or other reasons	`caller_start_call` Originator Initiates Call `caller_call_accepted` Originator Answers Call `caller_call_missed` Originator Misses Call `caller_call_rejected` Caller Rejects Call `caller_call_busy` Caller Line is Busy `caller_cancel_call` Caller Cancels Call `caller_call_failed` Caller Fails to Initiate Call `caller_call_end` Caller Ends Call Normally `caller_call_interrupted` Caller's Call Interrupted `callee_receive_call` Callee Receives Call `callee_accept_call` Callee Answers Call `callee_not_answer_call` Callee Does Not Answer `callee_reject_call` Callee Rejects Call `callee_ignore_call` Callee Ignores Call `callee_call_canceled` Callee Cancels Call `callee_call_end` Callee Ends Call Normally `callee_call_interrupted` Callee's Call Interrupted `invite_user` Midway invite user

	`join_in_group_call` Join the call midway

Establishing Callback Configuration

Last updated：2024-07-24 10:20:03

Feature Overview

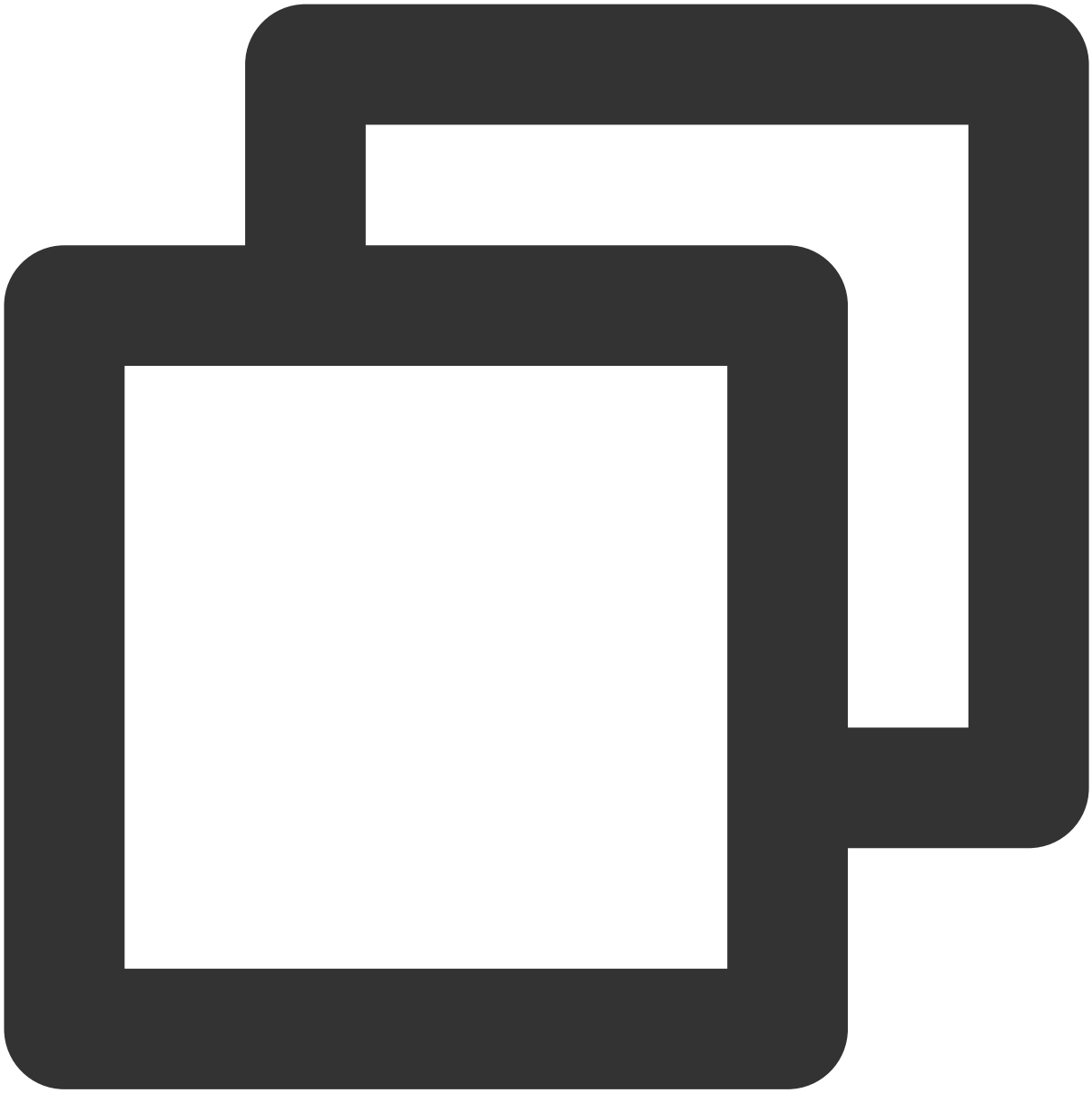
Administrators can create callbacks through this API.

API description

Note:

If the API is called multiple times, the last result will prevail.

Sample request URL



```
https://xxxxxx/v1/callback/set?sdkappid=88888888&identifier=admin&usersig=xxx&rando
```

Request parameters

The table below only lists the parameters modified when calling this API and their description. For more information, please refer to REST API Overview.

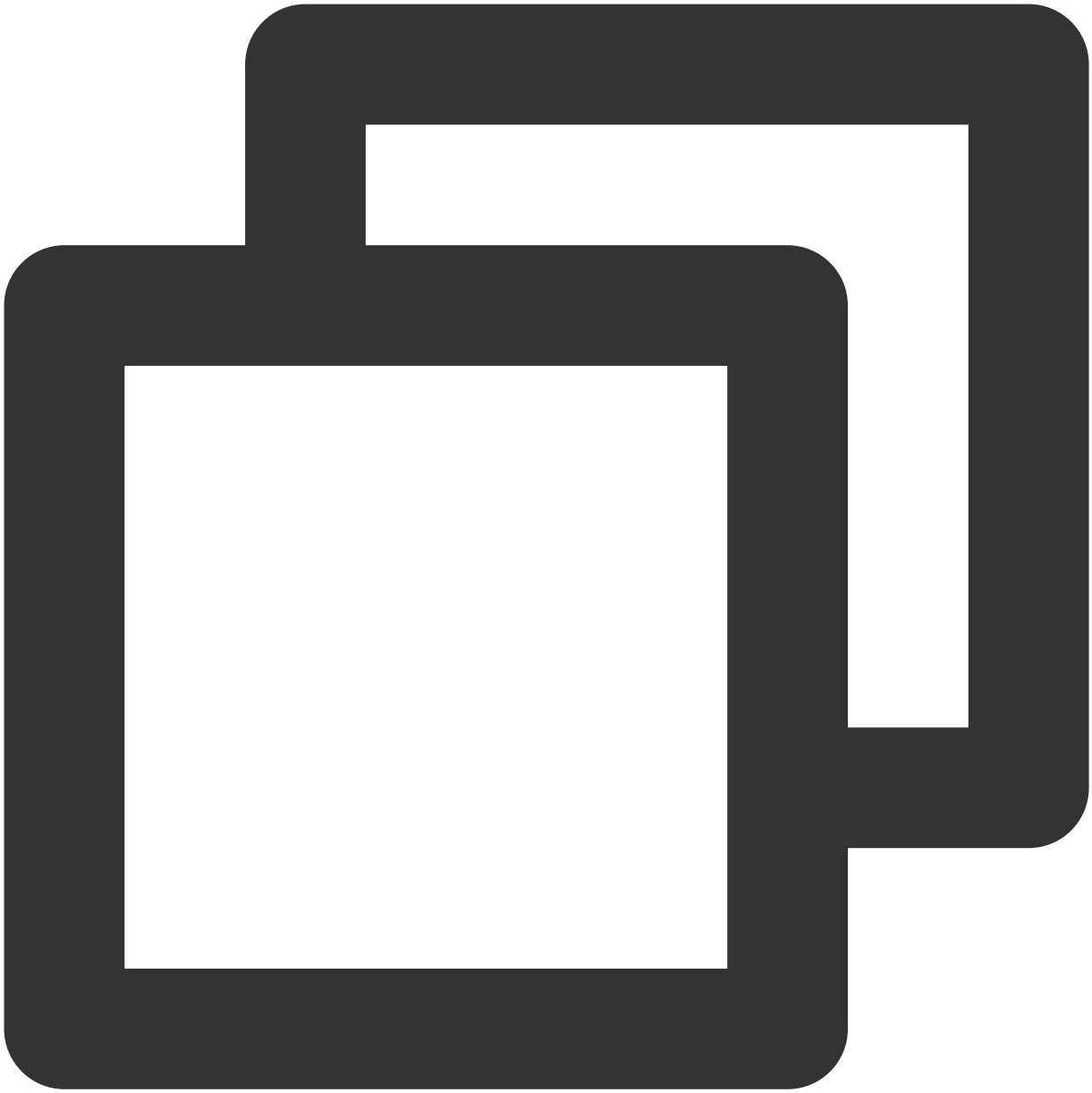
Parameter	Description
xxxxxx	The reserved domain for the country/region where the SDKAppID is located:

	`callkit-intl.trtc.tencent-cloud.com`
v1/callback/set	Request API
sdkappid	The sdkappid assigned by the console when creating an application
identifier	Must be an Chat App Administrator Account
usersig	The Signature generated by the App Administrator account, for detailed operations, please refer to Generating UserSig
random	Enter a random 32-bit unsigned integer, range 0 to 4294967295
contenttype	The request format fixed value is `json`

Maximum calling frequency

10 times per second.

Sample request packets

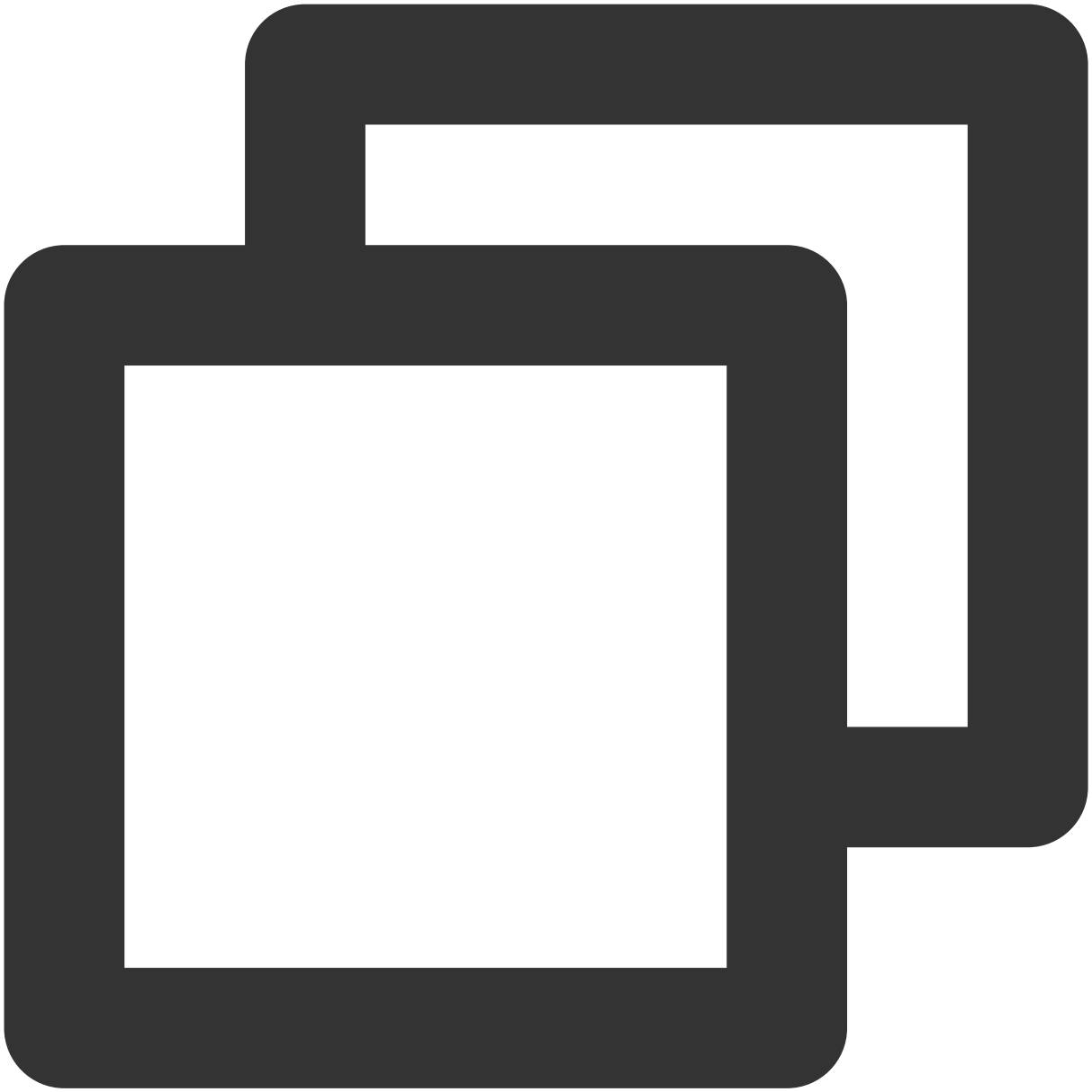


```
{
    "address":"http://www.example.com/callback",
    "actions": [
        "call_busy",
        "normal_end",
        "caller_start_call",
        "invite_user",
        "callee_reject_call"
    ]
}
```

Request field description

Field	Type	Attribute	Description
address	String	Mandatory	Callback address, must start with http/https, it is recommended to use the more secure https
actions	Array	Mandatory	Scenarios that require triggering a callback, see the Callback Command List for the list

Sample response packets



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "a1d8543a9b1daef5d0f0c21517a4bc0a",
    "data": "http://www.example.com/callback"
}
```

Response Packet Field Description

Field	Type	Description

errorCode	Integer	Error code, 0 indicates success
errorMessage	String	Error message
requestId	String	Unique Request ID
data	String	Successful callback address configuration

Error codes

Error code	Description
0	Request succeeded
50001	The current application needs to purchase the TUICallKit Group Call Version Package to use
70001	Callback address must start with http or https

Retrieving Callback Configuration

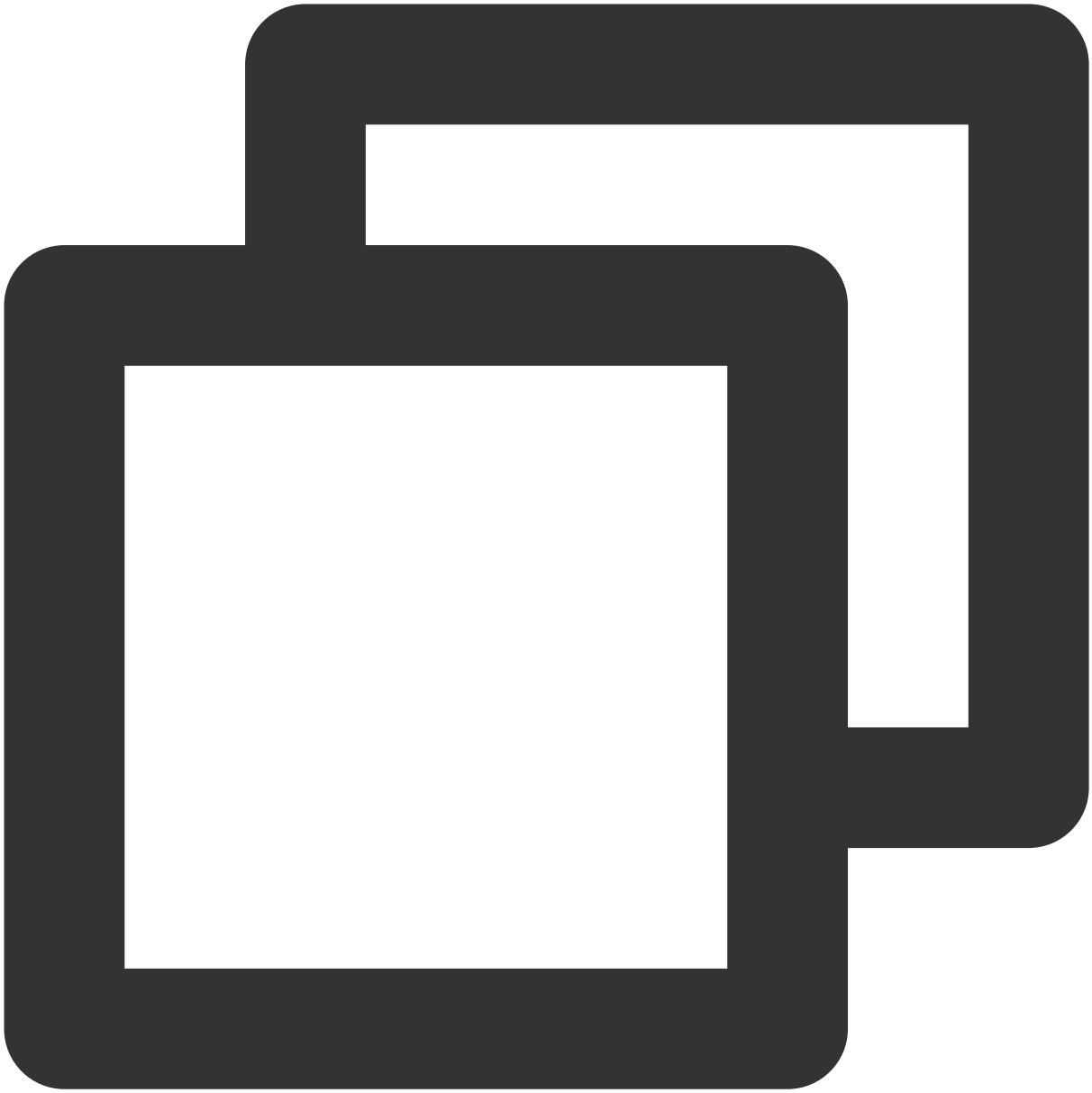
Last updated：2024-07-24 10:20:03

Feature Overview

Administrators can query configured callbacks through this interface.

API description

Sample request URL



```
https://xxxxxx/v1/callback/get?sdkappid=88888888&identifier=admin&usersig=xxx&rando
```

Request parameters

The table below only lists the parameters modified when calling this API and their description. For more information, please refer to REST API Overview.

Parameter	Description
xxxxxx	The reserved domain for the country/region where the SDKAppID is located:

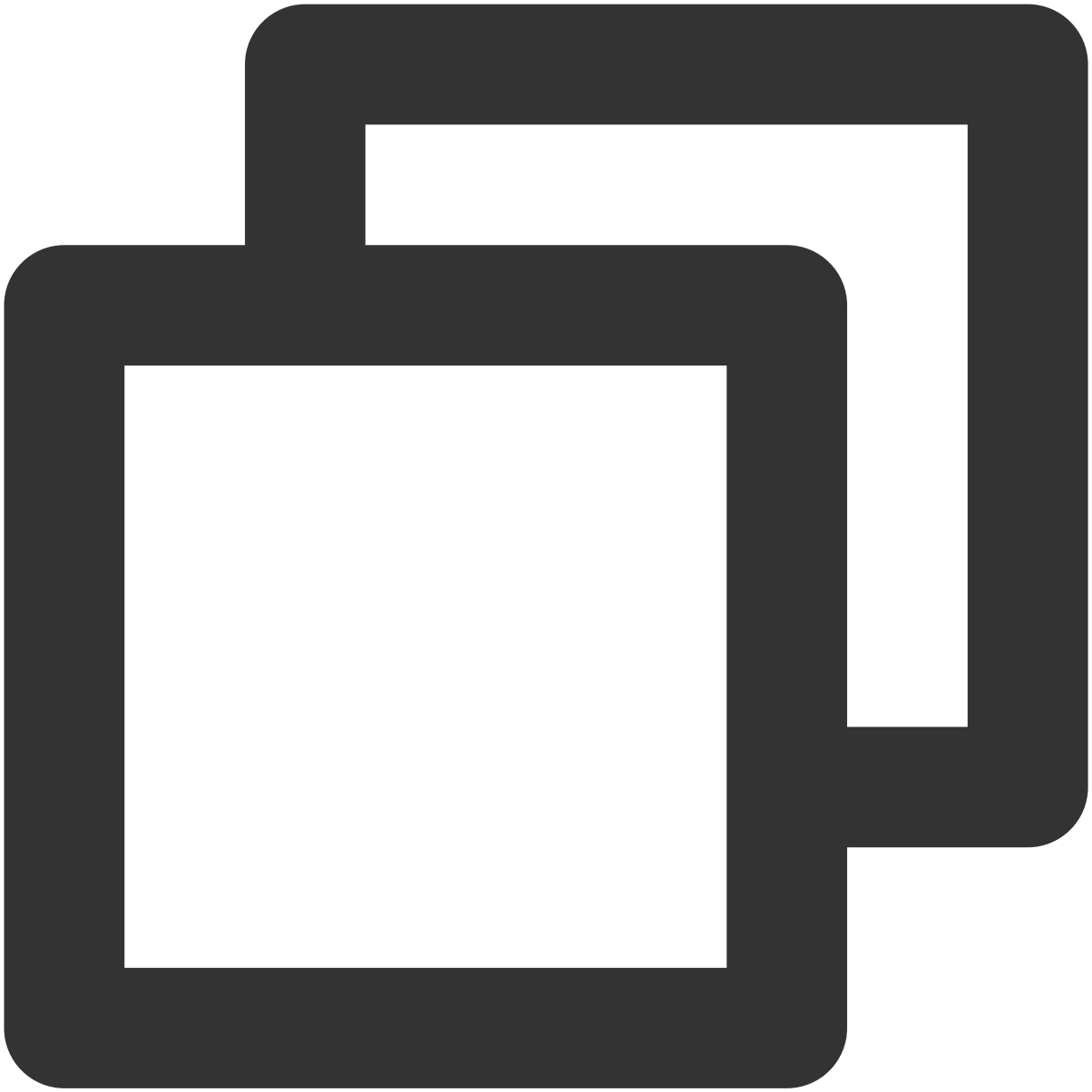
	`callkit-intl.trtc.tencent-cloud.com`
v1/callback/get	Request API
sdkappid	The sdkappid assigned by the console when creating an application
identifier	Must be an Chat App Administrator Account
usersig	The Signature generated by the App Administrator account, for detailed operations, please refer to Generating UserSig
random	A random 32-bit unsigned integer ranging from 0 to 4294967295
contenttype	The request format fixed value is `json`

Maximum calling frequency

10 times per second.

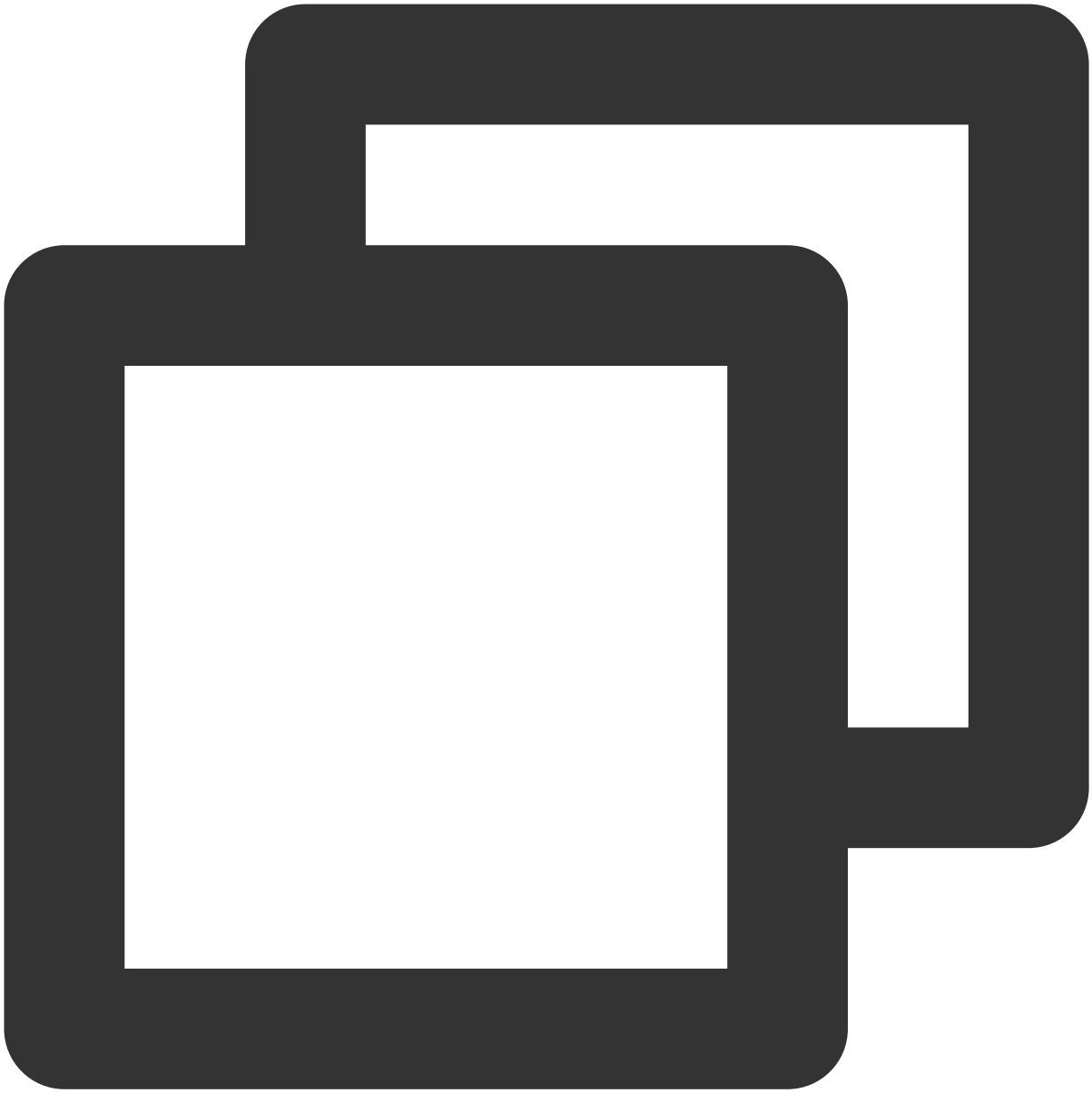
Sample request packets

Passing empty is sufficient.



```
{

}
```

Sample response packets



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "355c3c394f0602ed81a13c34999abebb",
    "data": {
        "actions": [
            "call_busy",
            "normal_end"
        ],
        "address": "http://www.example.com/callback"
    }
```

```
}
```

Response Packet Field Description

Field	Type	Description
errorCode	Integer	Error code, 0 indicates success
errorMessage	String	Error message
requestId	String	Unique Request ID
actions	Array	Configured callback actions
address	String	Configured callback address

Error codes

Error code	Description
0	Request succeeded
50001	The current application needs to purchase the TUICallKit Group Call Version Package to use
70001	Callback address must start with http or https

Update Callback Configuration

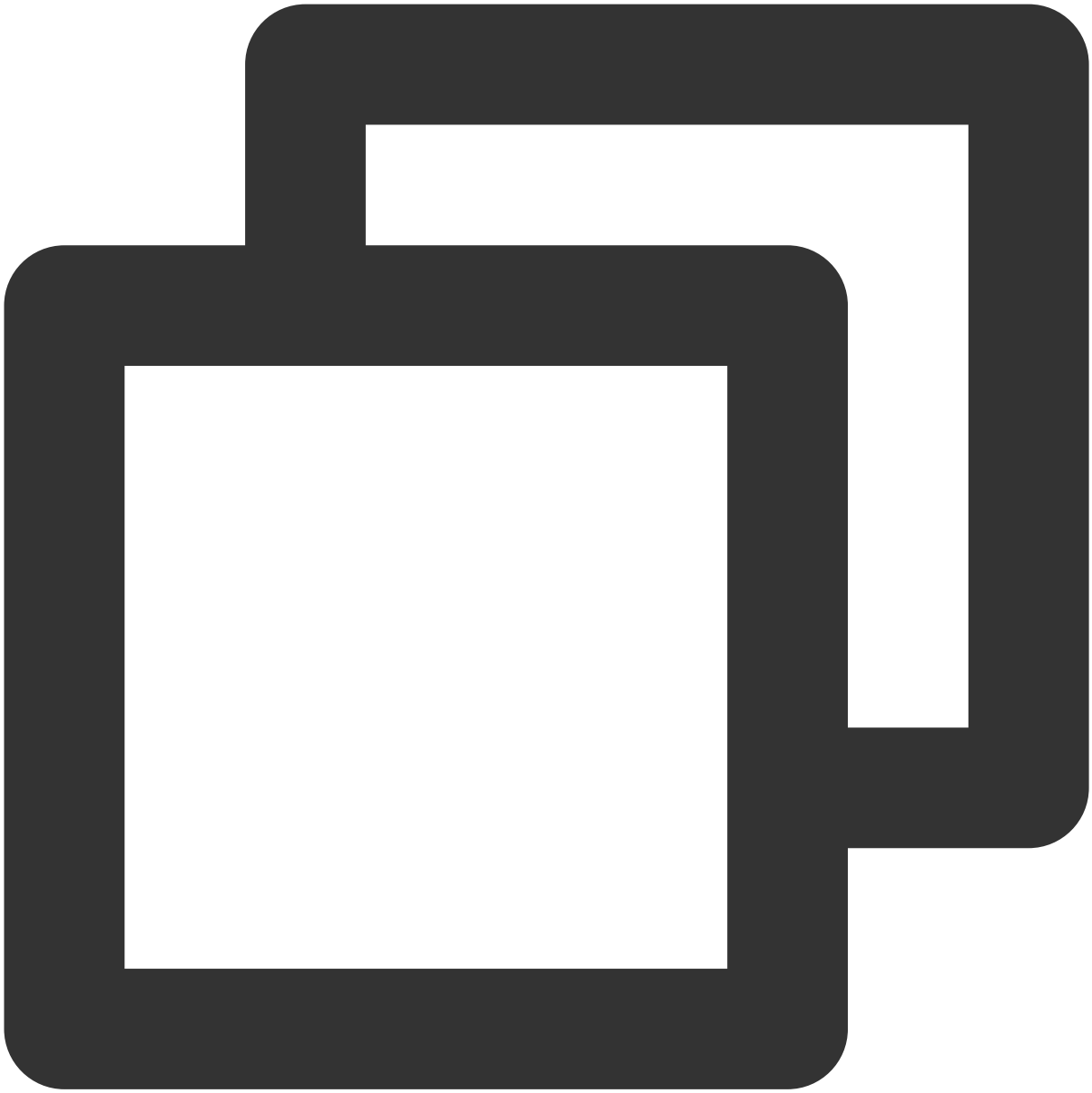
Last updated：2024-07-24 10:20:03

Feature Overview

Administrators can update callbacks through this interface.

API description

Sample request URL



```
https://xxxxxx/v1/callback/update?sdkappid=88888888&identifier=admin&usersig=xxx&ra
```

Request parameters

The table below only lists the parameters modified when calling this API and their description. For more information, please refer to REST API Overview.

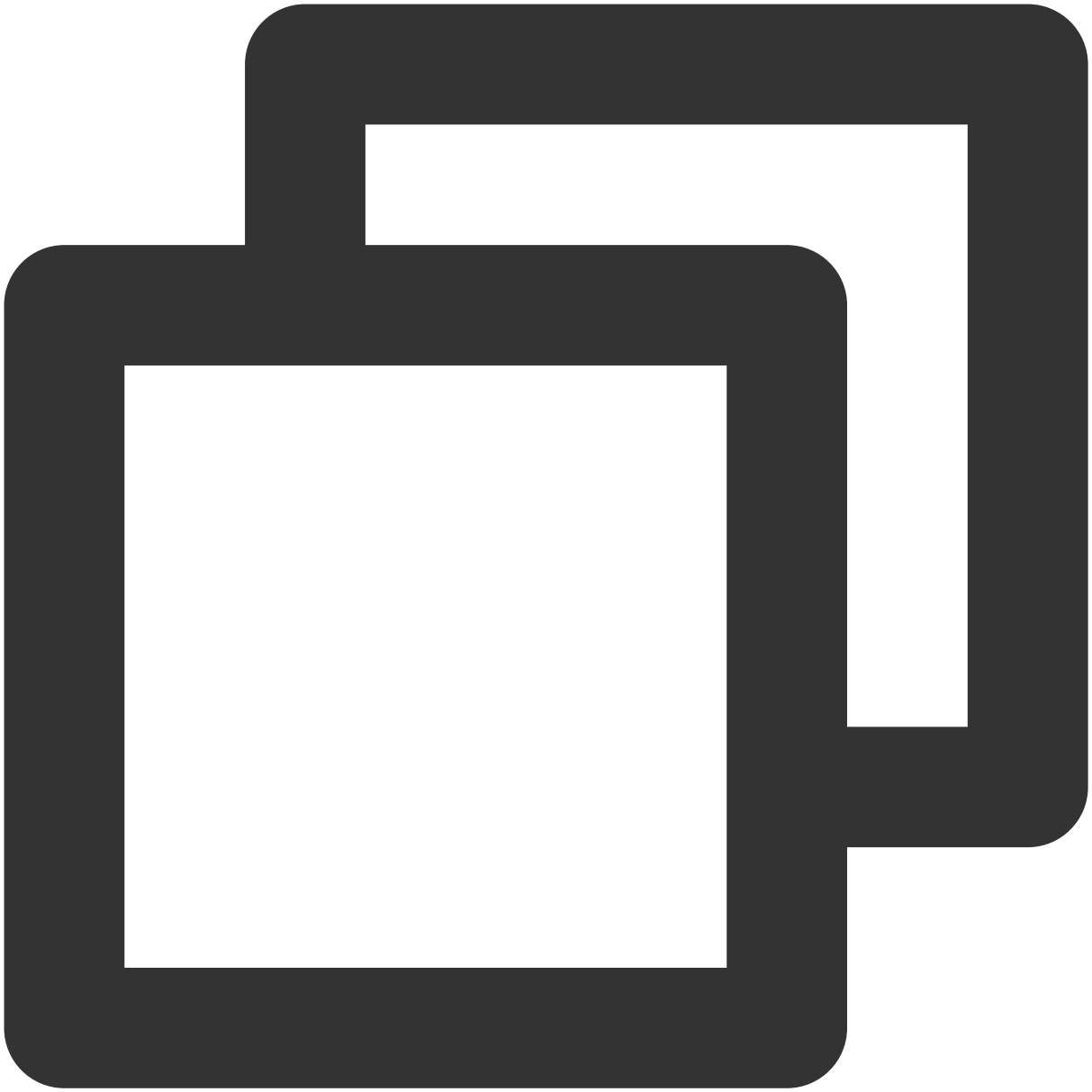
Parameter	Description
xxxxxx	The reserved domain for the country/region where the SDKAppID is located:

	`callkit-intl.trtc.tencent-cloud.com`
v1/callback/update	Request API
sdkappid	The sdkappid assigned by the console when creating an application
identifier	Must be an Chat App Administrator Account
usersig	The Signature generated by the App Administrator account, for detailed operations, please refer to Generating UserSig
random	A random 32-bit unsigned integer ranging from 0 to 4294967295
contenttype	The request format fixed value is `json`

Maximum calling frequency

10 times per second.

Sample request packets

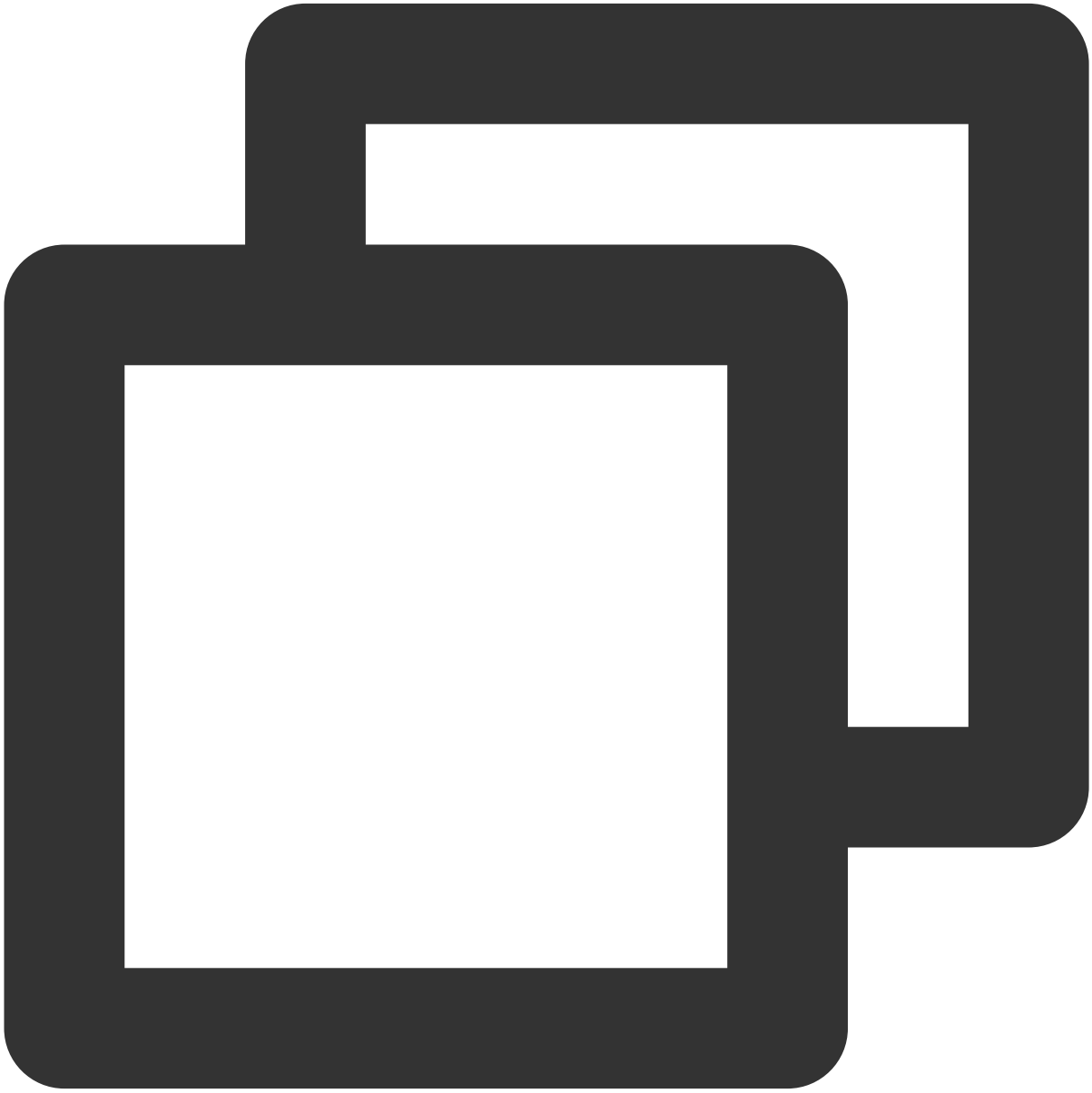


```
{
    "address": "https://www.example2.com/callback",
    "actions": [
        "call_busy",
        "cancel",
        "normal_end"
    ]
}
```

Request field description

Field	Type	Attribute	Description
address	String	Mandatory	Callback address, must start with http/https, it is recommended to use the more secure https
actions	Array	Mandatory	Scenarios requiring callback triggering, please refer to the Callback Command Word List

Sample response packets



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "5b0fa500064397cad3554506e27e18e1",
    "data": "https://www.example2.com/callback"
}
```

Response Packet Field Description

Field	Type	Description
errorCode	Integer	Error code, 0 indicates success
errorMessage	String	Error message
requestId	String	Unique Request ID
data	String	Callback address updated successfully

Error codes

Error code	Description
0	Request succeeded
50001	The current application needs to purchase the TUICallKit Group Call Version Package to use
999	No configuration information

Remove Callback Configuration

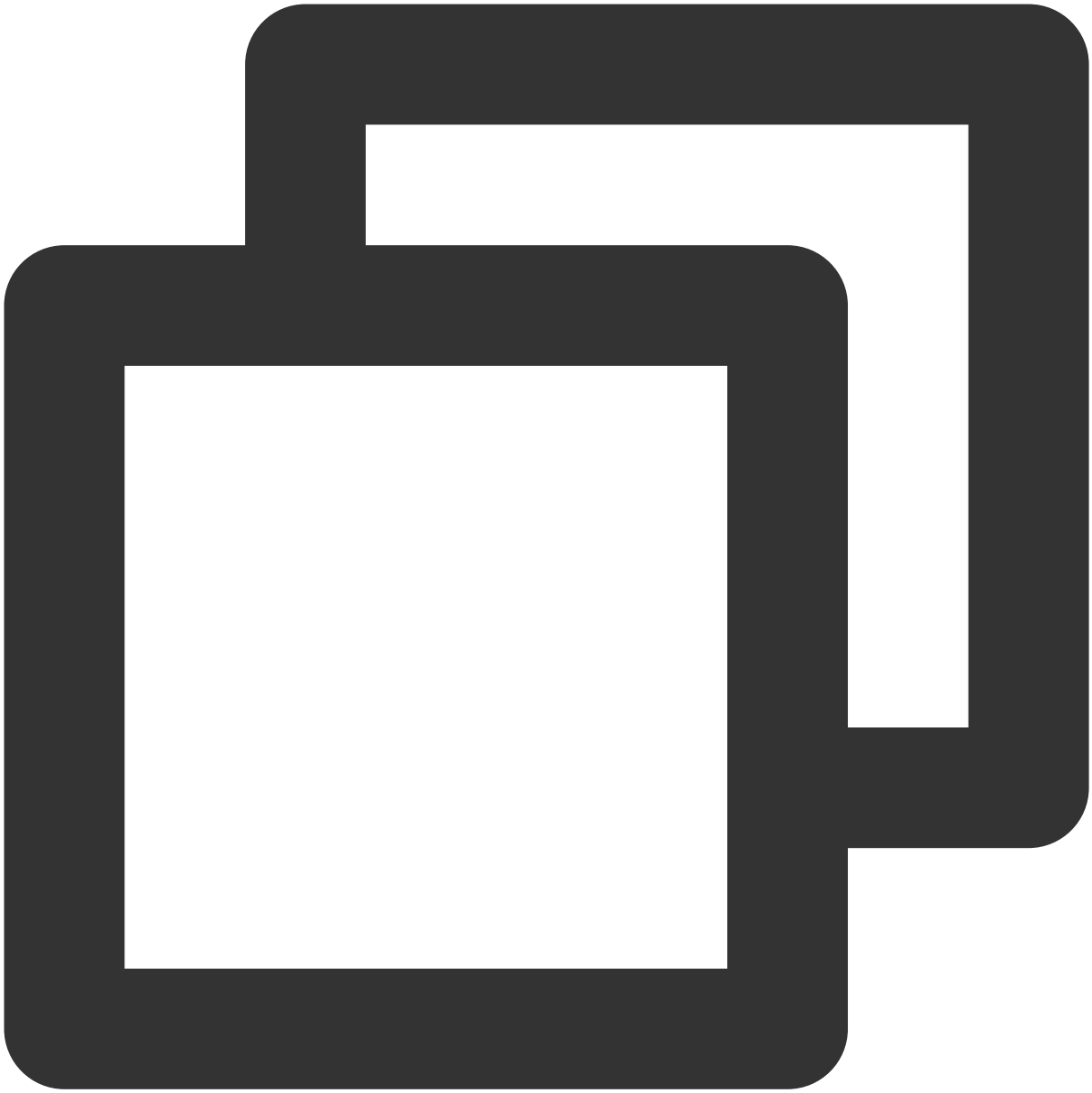
Last updated：2024-07-24 10:20:03

Feature Overview

The administrator can delete the callback through this interface.

API description

Sample request URL



```
https://xxxxxx/v1/callback/delete?sdkappid=88888888&identifier=admin&usersig=xxx&ra
```

Request parameters

The table below only lists the parameters modified when calling this API and their description. For more information, please refer to REST API Overview.

Parameter	Description
xxxxxx	The reserved domain for the country/region where the SDKAppID is located:

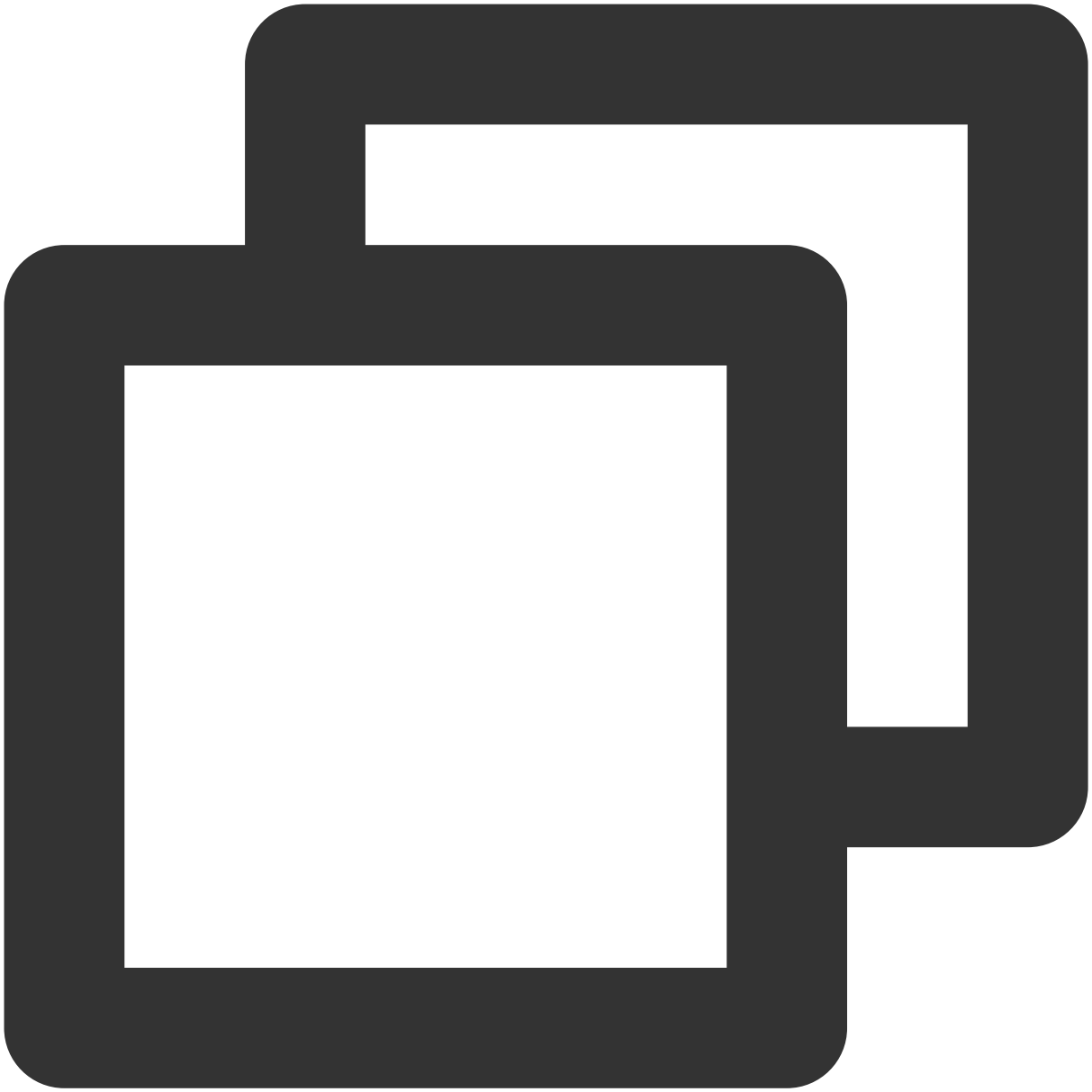
	`callkit-intl.trtc.tencent-cloud.com`
v1/callback/delete	Request API
sdkappid	The sdkappid assigned by the console when creating an application
identifier	Must be an Chat App Administrator Account
usersig	The Signature generated by the App Administrator account, for detailed operations, please refer to Generating UserSig
random	Enter a random 32-bit unsigned integer, range 0 to 4294967295
contenttype	The request format fixed value is `json`

Maximum calling frequency

10 times per second.

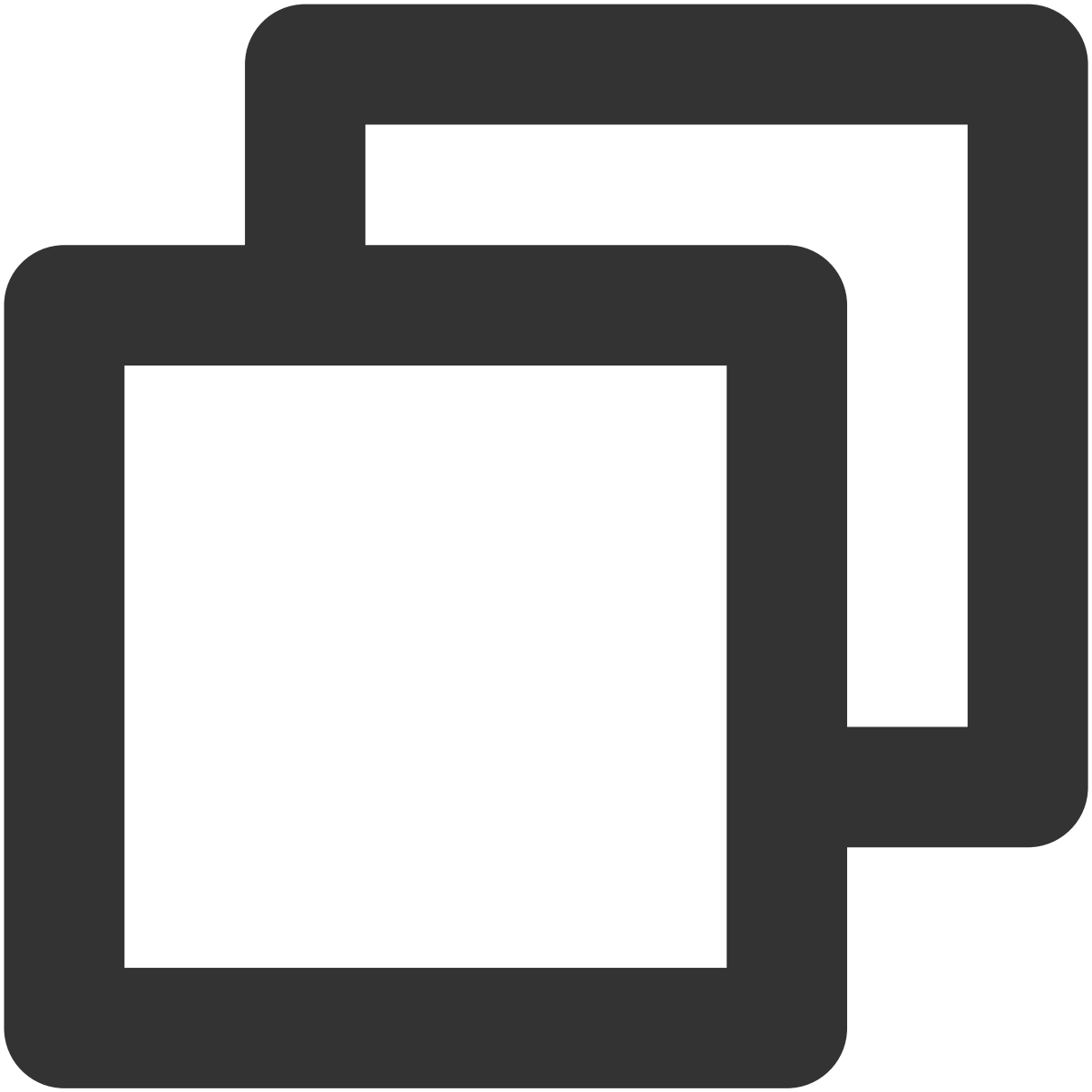
Sample request packets

Passing empty is sufficient.



```
{

}
```

Sample response packets



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "5b0fa500064397cad3554506e27e18e1",
    "data": ""
}
```

Response Packet Field Description

Field	Type	Description

errorCode	Integer	Error code, 0 indicates success
errorMessage	String	Error message
requestId	String	Unique Request ID
data	String	No data

Error codes

Error code	Description
0	Request succeeded
50001	The current application needs to purchase the TUICallKit Group Call Version Package to use

REST API

Introduction to REST API

Last updated：2024-07-24 10:20:03

The RESTful API is a part of the TRTC Call's backend HTTP Hub Management Interface Hub, providing developers with a simplified management entry. For the RESTful APIs currently supported by the TRTC Call, please refer to REST API List.
For security reasons, the RESTful API is only available via HTTPS Interface.

Use Conditions

The REST API is currently in beta. During the beta period, applications (SDKAppId) that have enabled the **Group Call Version** of TRTC Call can use the REST API. You can also activate a free trial of the experience version. For version descriptions and activation guidelines, refer to Activate the Service.
Currently, the RESTful API is available only in specific versions of TUICallKit across different platforms, as detailed in the table below:

Platform/Framework	Version number
Android/iOS/Flutter/uni-app (client)	≥ 1.7.1
Web	≥ 1.4.6
WeChat Mini Program	≥ 1.5.1

Note:
Only after all participating platforms/frameworks are upgraded to the versions mentioned above, can the corresponding call information be peeked at in the console.
During the Beta Testing Period, the RESTful API supports querying data from the past 7 days.

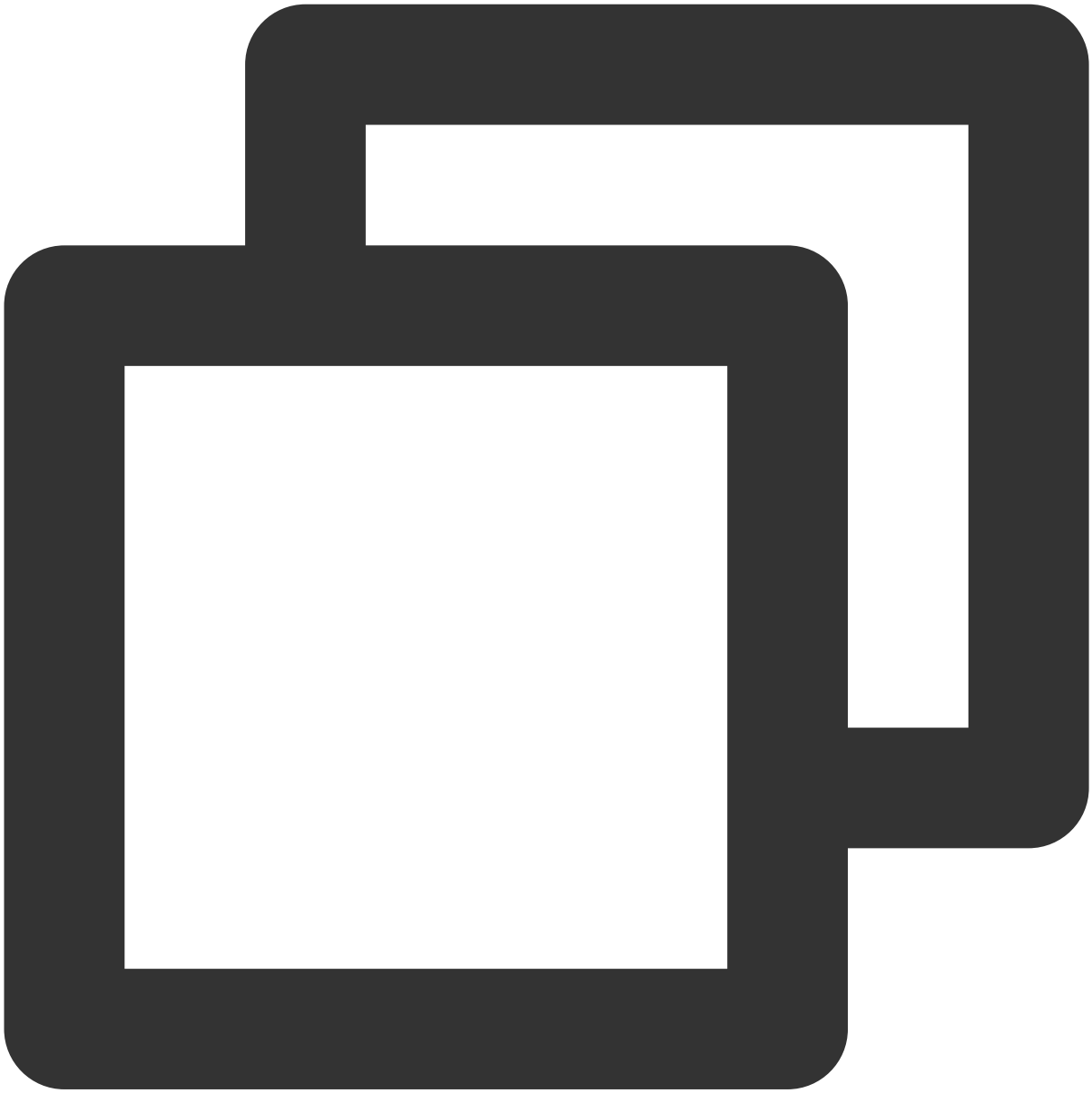
RESTful API List

Feature Overview	API
Access records through callId	v1/records/get_record_by_callId
Access records by condition	v1/records/get_records_by_filter

Calling method

Request URL

The URL format of the RESTful API is as follows:



```
https://xxxxxx/$version/$kind/$command?sdkappid=$SDKAppID&identifier=$identifier&us
```

The meanings and values of each parameter are as follows (both parameter names and their values are case-sensitive):

Parameter	Meaning	Fetching Value
https	Request protocol	The request protocol is HTTPS, and the request method is POST
xxxxxx	reserved domain name	`callkit-intl.trtc.tencent-cloud.com`
version	Protocol Version Number	Fixed as `v1`
kind	Management Classification	Example: `v1/records/get_record_by_callId` , where `records` is a kind
command	The word `command`, combined with `kind`, is used to indicate a specific business feature	Example: `v1/records/get_record_by_callId` , where `get_record_by_callId` is a command
sdkappid	The application identifier accessed in the console	Obtained when applying for integration
identifier	username, must be an App Administrator Account when calling RESTful APIs	Using the Admin account of Chat
usersig	password corresponding to username	Refer to Generating UserSig
random	Identifies the random number parameter for the current request	32-bit unsigned integer random number, ranging from 0 to 4294967295
contenttype	Request Format	Fixed value: json

Note:

After obtaining or purchasing package bundles, an administrator account named `administrator` will be created in the Chat account system. Please use `administrator` for the `identifier` parameter in requests. If you cancel or delete an administrator in Chat Account Management, please correctly specify the administrator account and corresponding UserSig.

Apps can either generate a UserSig for the administrator account at every RESTful API call, or generate a fixed UserSig for repeated use. However, it is crucial to pay attention to the UserSig's validity period.

During operations such as creating or entering a room, the system will automatically import Chat accounts into the Chat System. Please be aware.

HTTP Request Body Format

The RESTful API only supports the POST method, and its request body is in JSON format. For specific body formats, refer to the detailed description of each API.

Note:

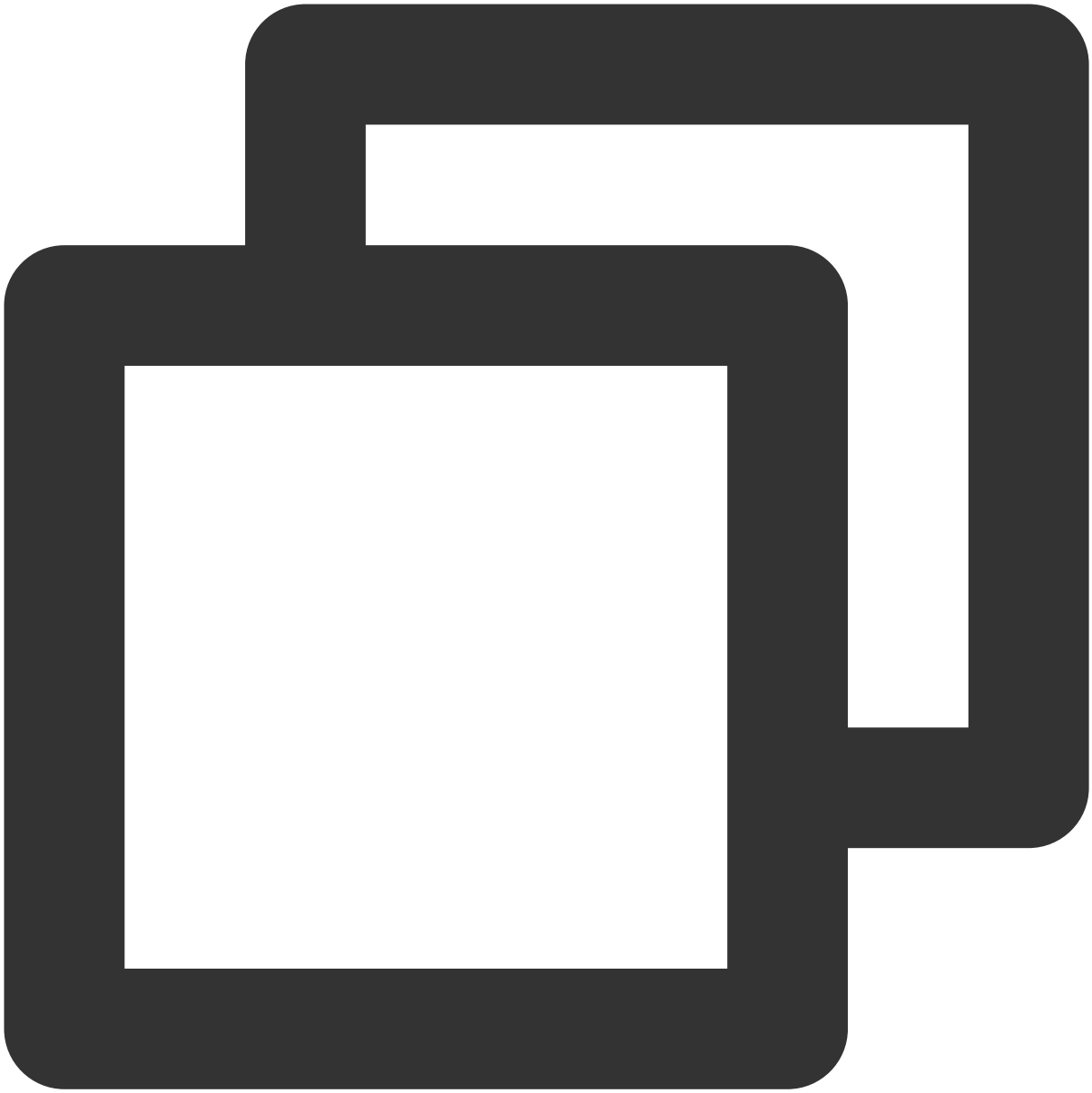
The POST body cannot be empty. Even if a protocol does not require any information to be carried, it must still include an empty JSON object, namely `{}` .

HTTP return code

Unless a network error occurs (e.g., 502 error), the call result of the RESTful API is always 200. The actual error code and error message of the API call are returned in the HTTP response body.

HTTP Response Body Format

The response body of the RESTful API is also in JSON format, and its format conforms to the following characteristics:



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "1c8960ac38d61be38b6fb219db0182d1",
    "data": {}
}
```

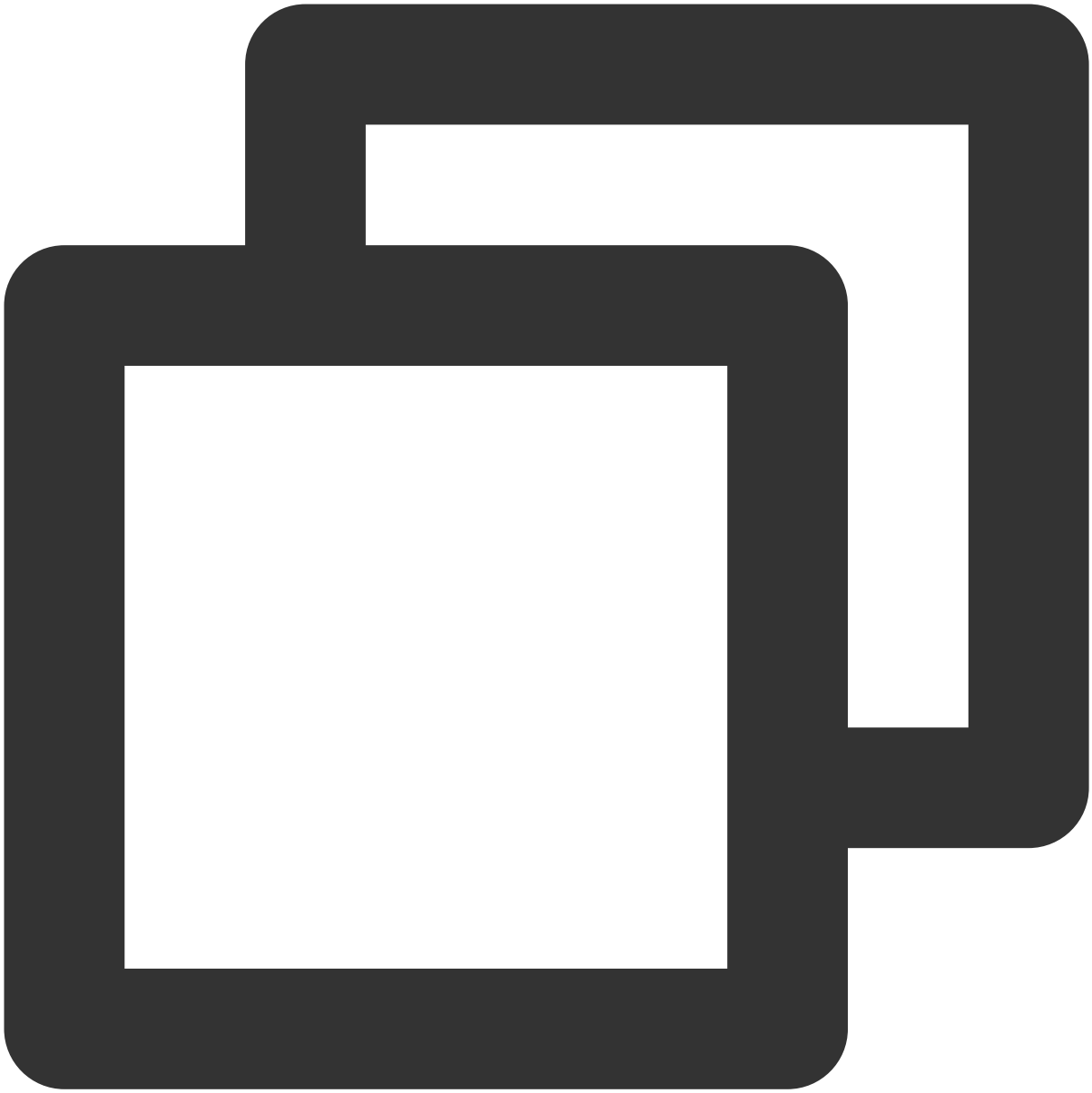
The response body must contain three attributes: errorCode, errorMessage, requestId. Their meanings are as follows:

Field	Type	Description

errorCode	String	Error code, 0 for success, others for failure
errorMessage	String	Error message
requestId	Integer	Request Unique Identifier

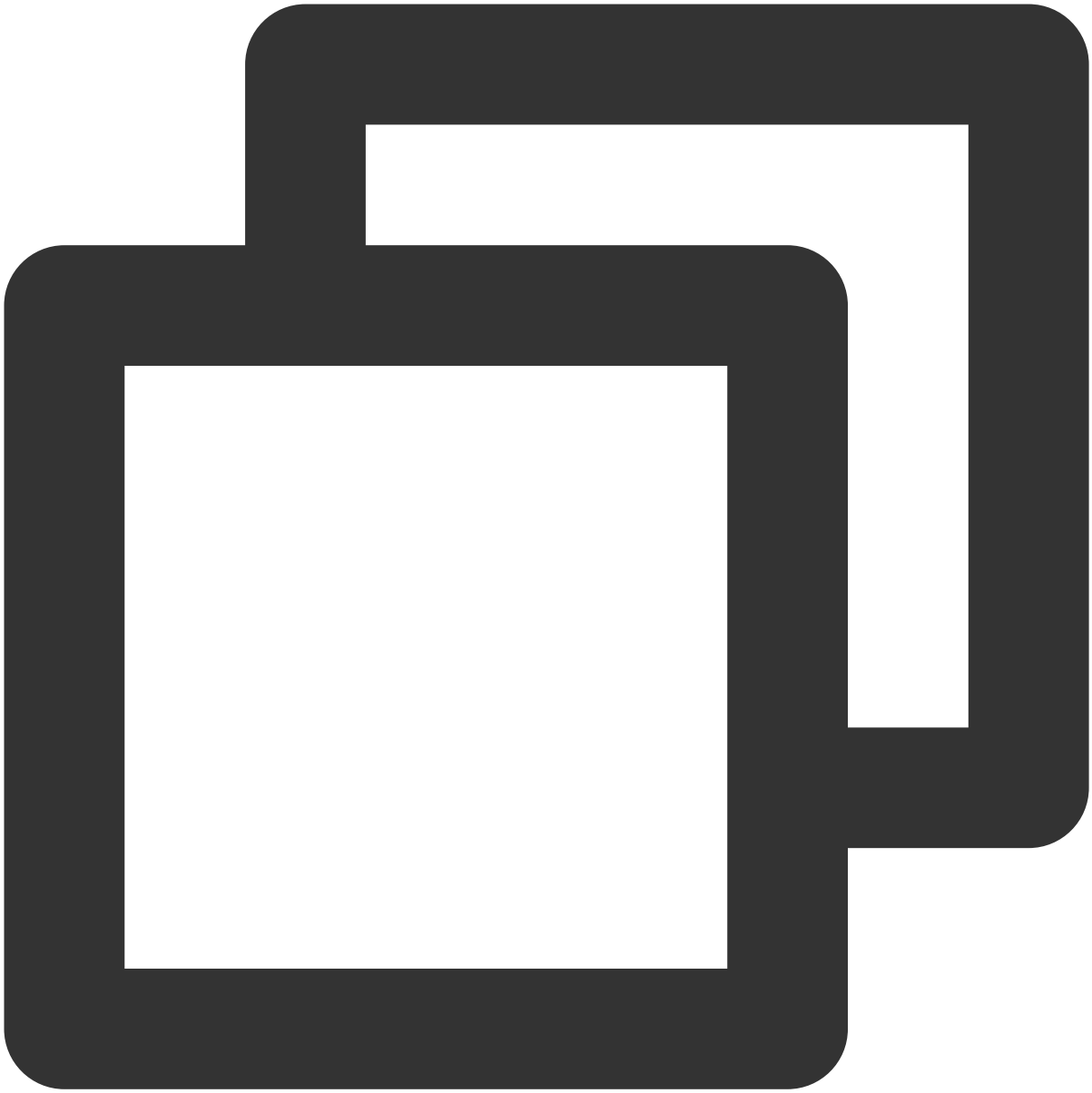
Sample call

Below is an example of accessing call records for a specified callId through RESTful APIs.
HTTPS Request:



```
POST /records/get_record_by_callId?usersig=xxx&identifier=admin&sdkappid=88888888&r
Host: callkit-intl.trtc.tencent-cloud.com
Content-Length: 36
{
    "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582"
}
```

HTTPS Response:



```
HTTP/2 200 OK
Date: Fri, 21 Apr 2023 06:06:16 GMT
Content-Length: 112
Connection: keep-alive

{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "9f01db503f2006ad10c45f4f4609e38d",
    "data": {
        "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582",
```

```
        "sdkAppId": 88888888,
        "mediaType": "video",
        "roomId": "123456",
        "startCallTs": 1688705638,
        "acceptTs": 1688705641,
        "endTs": 1688705668,
        "caller": "alice",
        "totalUserNumber": 2,
        "callType": "single",
        "callResult": "normal_end",
        "callees": [
            "bob1",
            "bob2"
        ]
    }
}
```

Common Error Codes for RESTful APIs

Error code	Description
0	Request succeeded
50001	The current application needs to purchase the TUICallKit Group Call Version Package to use
70001	Invalid request parameters, please check whether mandatory parameters are missing or incorrectly entered
70002	UserSig is invalid
70003	UserSig has expired
70004	Requesting user is not a Super Administrator
70005	Request frequency limited
70009	Error in parsing request body, please check if the request parameter type is correct
Unknown error code	System internal error, please Submit a ticket to contact technical personnel

FAQs

Is there a chance that REST API requests timeout, receiving no response?

You can confirm from the following aspects:

1. The backend REST interface's timeout setting is 3s, the caller's timeout setting should be longer than 3s.
2. `telnet callkit-intl.trtc.tencent-cloud.com 443` to confirm if the service port can be connected.
3. Use `curl -I https://callkit-intl.trtc.tencent-cloud.com` for a simple test to see if the status code is 200.
4. Confirm whether the machine's DNS server configuration is an internal DNS server or a public DNS server. If it is an internal DNS server, ensure that the DNS server's network egress and the region ISP of the machine's network egress IP match.
5. It is recommended for business callers to use the **long connection + connection pool** pattern.

Retrieve records via callId

Last updated：2024-07-24 10:20:03

Feature Overview

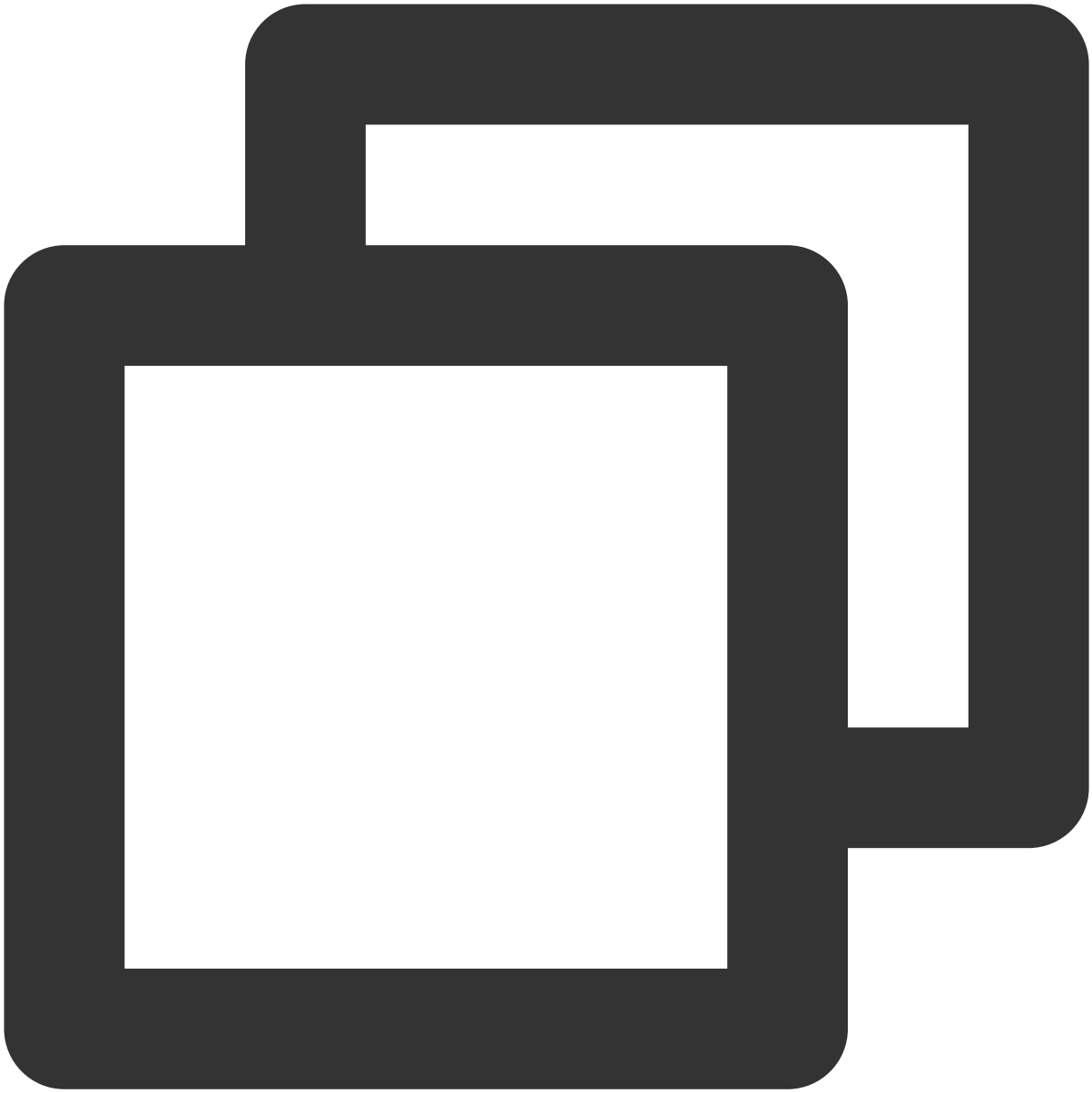
Administrators can access call records by callId through this interface.

Note:

The RESTful API is currently in beta. You can query call data created within the last 7 days.

API Calling Description

Sample request URL



```
https://xxxxxx/v1/records/get_record_by_callId?sdkappid=88888888&identifier=admin&u
```

Request parameters

The table below only lists the parameters modified when calling this API and their description. For more information, please refer to REST API Overview.

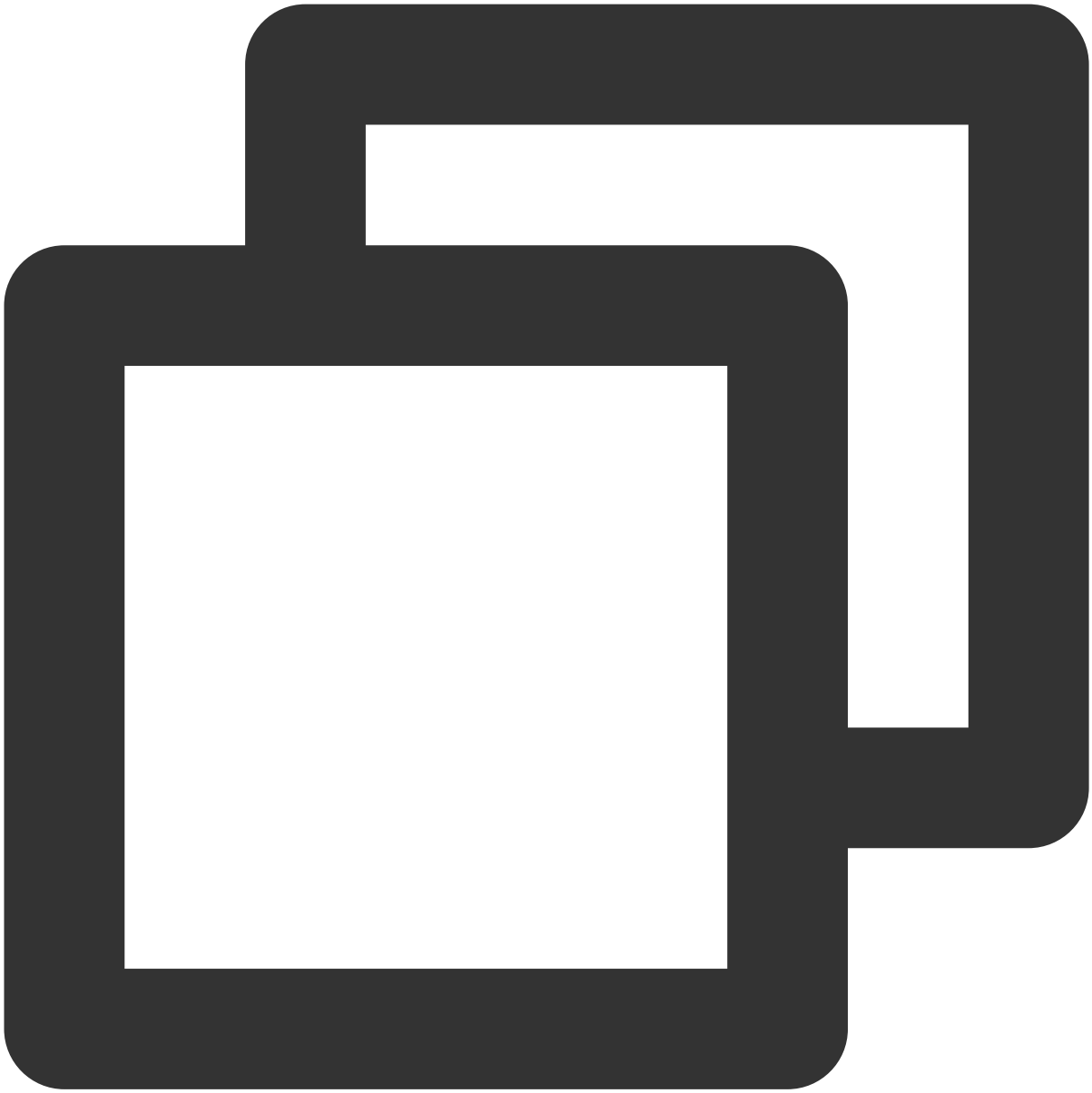
Parameter	Description

xxxxxx	The reserved domain for the country/region where the SDKAppID is located: `callkit-intl.trtc.tencent-cloud.com`
v1/records/get_record_by_callId	Request API
sdkappid	SDKAppID assigned by the console when creating an application
identifier	Using the Admin account of Chat
usersig	The Signature generated by the App Administrator account, for detailed operations, please refer to Generating UserSig
random	Enter a random 32-bit unsigned integer, range 0 to 4294967295
contenttype	The request format fixed value is `json`

Maximum calling frequency

20 times per second.

Sample request packets



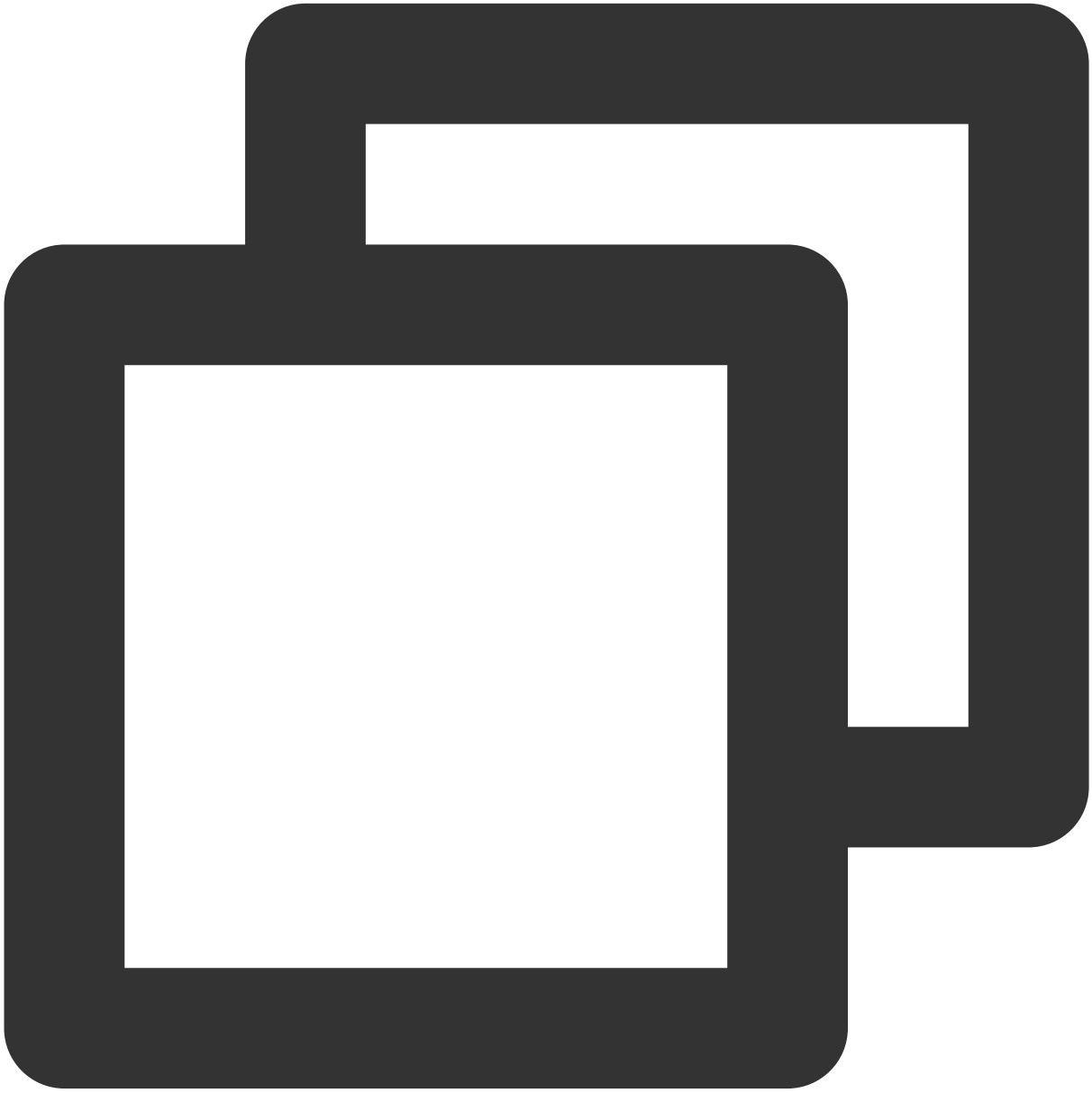
```
{
    "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582"
}
```

Request packet fields

Field	Type	Attribute	Description

callId	String	Mandatory	Call Unique ID

Sample response packets



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "3fdc344c7a67461c6af3cfc3e77744b5",
```

```
    "data": {
        "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582",
        "sdkAppId": 88888888,
        "mediaType": "video",
        "roomId": "123456",
        "startCallTs": 1688705638,
        "acceptTs": 1688705641,
        "endTs": 1688705668,
        "caller": "alice",
        "totalUserNumber": 2,
        "callType": "single",
        "callResult": "normal_end",
        "callees": [
            "bob1",
            "bob2"
        ]
    }
}
```

Response Packet Field Description

Field	Type	Description
errorCode	Integer	Error code, 0 indicates success
errorMessage	String	Error message
requestId	String	Unique Request ID
callId	String	Unique ID of the call
sdkAppId	Integer	Your sdkAppId
mediaType	String	Media type `video` Video Call `audio` Audio Call
roomId	String	Room ID of the call
startCallTs	Integer	Call Initiation Timestamp (in seconds)
acceptTs	Integer	Call Connection Timestamp (in seconds)

endTs	Integer	Call End Timestamp (in seconds)
caller	String	Caller userId
totalUserNumber	Integer	Total Number of Participants in the Call
callType	String	Call Type: `single` One-on-one call `group` Group Call
callResult	String	Call Result: `cancel` Cancel: The caller cancelled the call before connection. `reject` Declined call: The callee rejected. `not_answer` Not Answered: The callee did not answer in time. `normal_end` Completed: The call was connected and ended normally. `call_busy` Busy: The line was busy during the call. `interrupt` Interrupts: The call was interrupted due to network or other reasons.
callees	Array	List of callee user IDs who participated in the call

Note:

The callResult field may display all types for one-on-one calls only, while group chats only have `normal_end` .

Error codes

Unless there is a network error (e.g., a 502 error), the HTTP return code for this interface is always 200. The actual error code and error message are conveyed through errorCode and errorMessage in the response body.

For common error codes (70000 to 79999), please see Error Code.

Error code	Description
0	Request succeeded
50001	The current application needs to purchase the TUICallKit Group Call Version Package to use
70011	No records found
Unknown error code	Unknown error, please submit a ticket to contact technical staff

Retrieve Records Based on Conditions

Last updated：2024-07-24 10:20:03

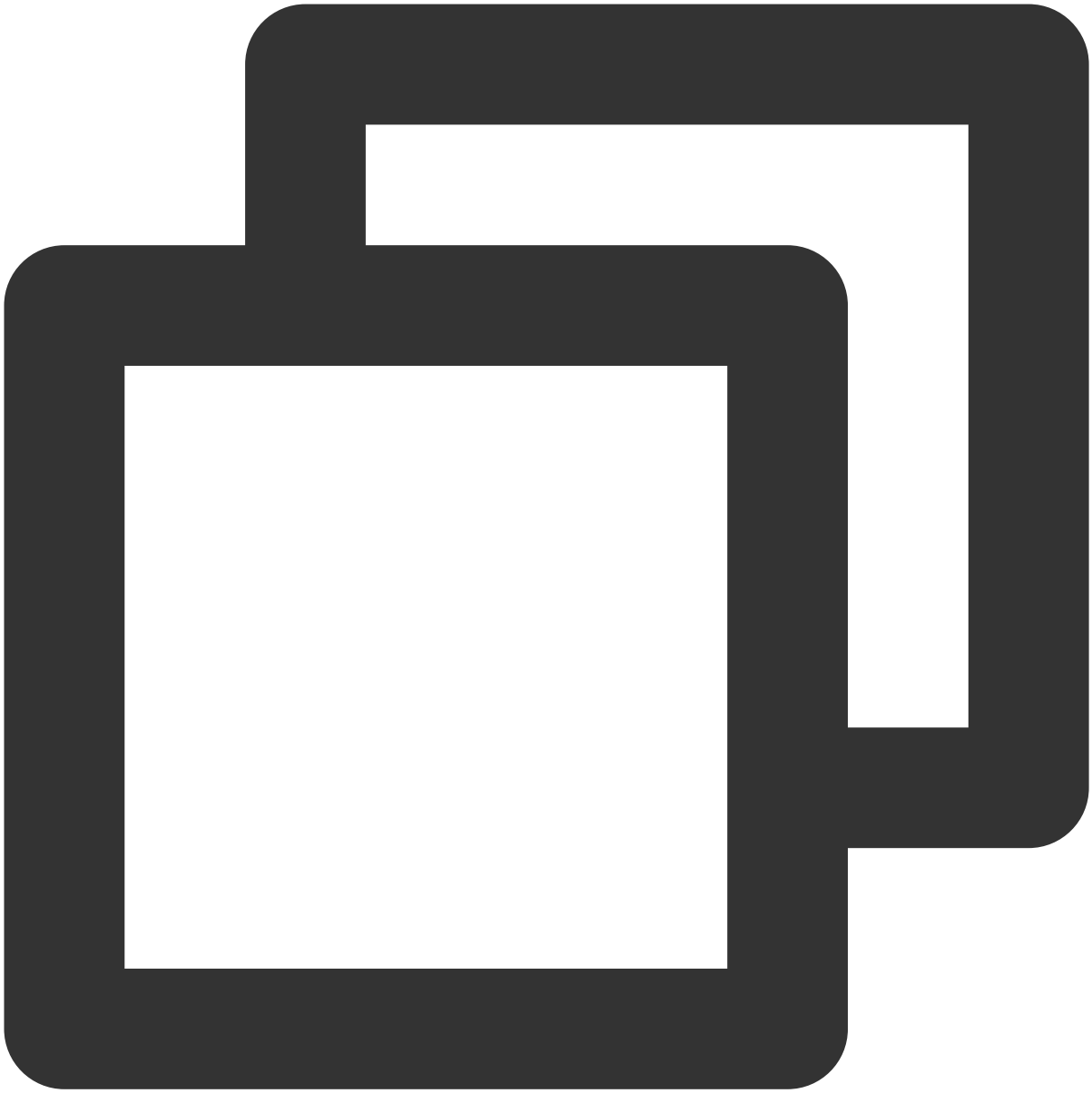
Administrators can access call records based on specified conditions through this interface.

Note:

The RESTful API is currently in beta. You can query call data created within the past seven days.

API description

Sample request URL



```
https://xxxxxx/v1/records/get_records_by_filter?sdkappid=88888888&identifier=admin&
```

Request parameters

The table below only lists the parameters modified when calling this API and their description. For more information, please refer to REST API Overview.

Parameter	Description
xxxxxx	The reserved domain for the country/region where the SDKAppID is located:

	`callkit-intl.trtc.tencent-cloud.com`
v1/records/get_records_by_filter	Request API
sdkappid	SDKAppID assigned by the console when creating an application
identifier	Using the Admin account of Chat
usersig	The Signature generated by the App Administrator account, for detailed operations, please refer to Generating UserSig
random	Enter a random 32-bit unsigned integer, range 0 to 4294967295
contenttype	The request format fixed value is `json`

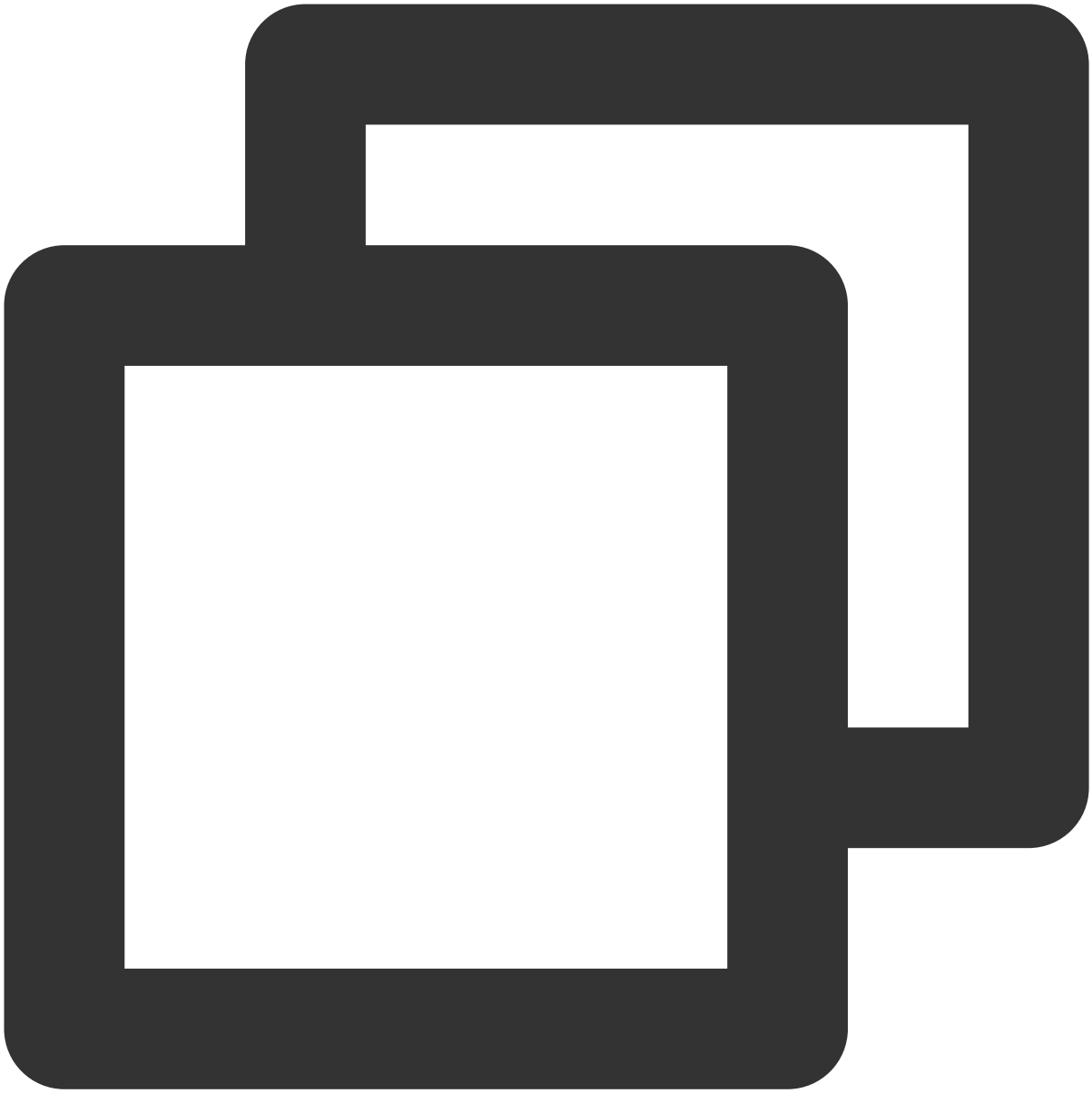
Maximum calling frequency

200 queries/sec.

Sample request packets

Basic form

Create room

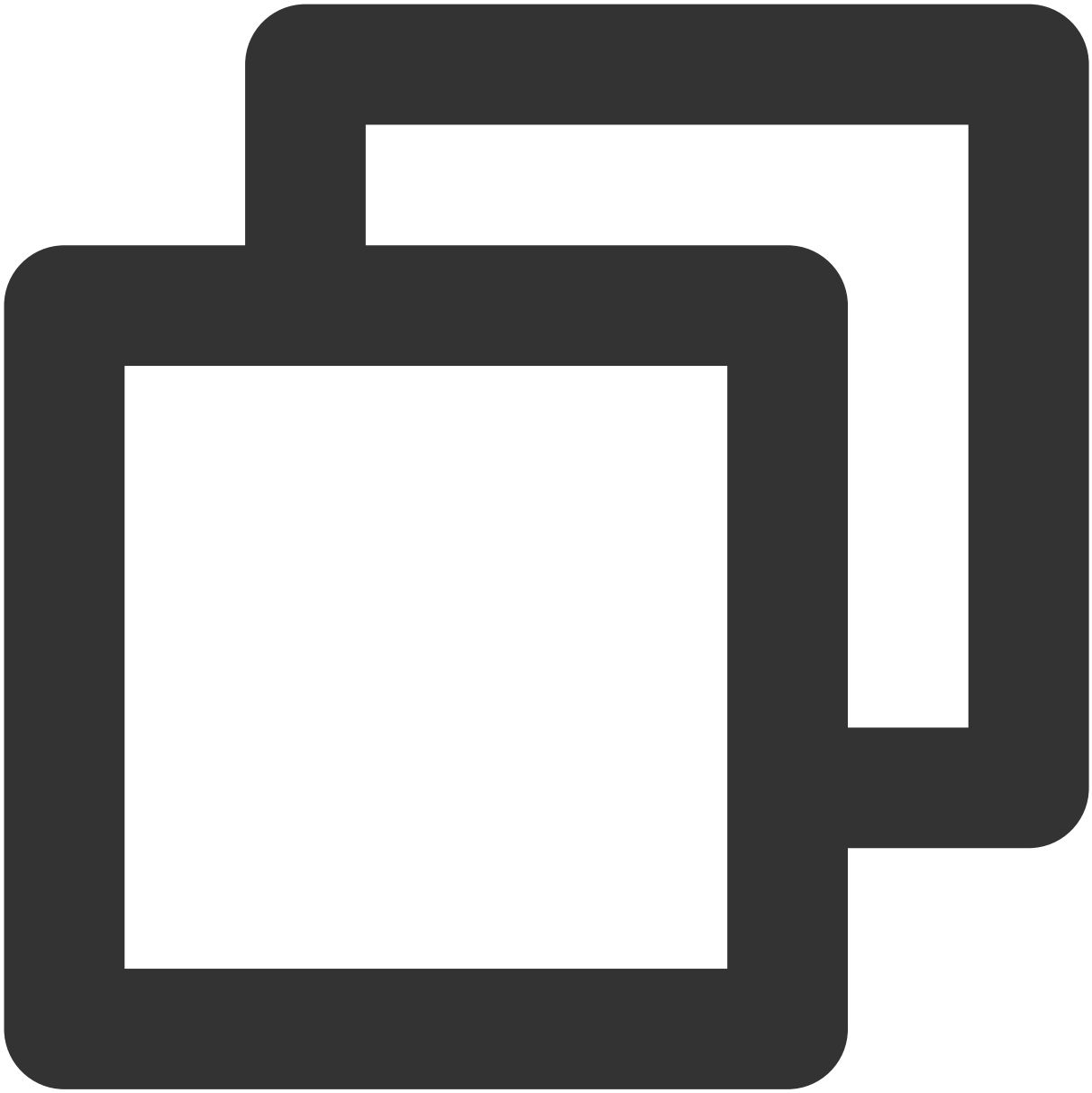


```
{
    "startTimestamp": 1618705638,
    "endTimestamp": 1618705738,
    "callResult": "normal_end",
    "callType": "single",
    "numberPerPage": 20,
    "page": 2
}
```

Request packet fields

Field	Type	Attribute	Description
startTimestamp	Integer	Mandatory	Call start timestamp (in seconds)
endTimestamp	Integer	Optional	Call end Timestamp (in seconds), if not specified, the default is 7 days
callResult	String	Optional	Call result, if not specified, defaults to all results
callType	String	Optional	Call type, if not specified, defaults to all types
numberPerPage	Integer	Optional	Number of queries per page, default is: 100
page	Integer	Optional	Query page number, if not specified, defaults to the first page

Response Packet Field Example




```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "8ba14f64a1d92d38705eec775e1f3deb",
    "data": {
        "totalNum": 3,
        "page": 2,
        "callRecordList": [
            {
                "callId": "2ae7d549-c441-4a9b-87c0-61810fe19581",
                "sdkAppId": 88888888,
```

```
            "mediaType": "video",
            "roomId": "456",
            "startCallTs": 1688709003,
            "acceptTs": 1688709003,
            "endTs": 1689150030,
            "caller": "123",
            "totalUserNumber": 0,
            "callType": "single",
            "callResult": "offline",
            "callees": [
                "111",
                "123"
            ]
        },
        {
            "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582",
            "sdkAppId": 88888888,
            "mediaType": "video",
            "roomId": "456",
            "startCallTs": 1688709303,
            "acceptTs": 1688709303,
            "endTs": 1689150030,
            "caller": "123",
            "totalUserNumber": 0,
            "callType": "single",
            "callResult": "offline",
            "callees": [
                "111",
                "123"
            ]
        },
        {
            "callId": "2ae7d549-c441-4a9b-87c0-61810fe19583",
            "sdkAppId": 88888888,
            "mediaType": "video",
            "roomId": "456",
            "startCallTs": 1688709903,
            "acceptTs": 1688709903,
            "endTs": 1689150030,
            "caller": "123",
            "totalUserNumber": 0,
            "callType": "single",
            "callResult": "offline",
            "callees": [
                "111",
                "123"
            ]
```

```
                }
            ]
        }
    }
```

Response Packet Field Description

Field	Type	Description
errorCode	Integer	Error code, 0 indicates success
errorMessage	String	Error message
requestId	String	Unique Request ID
totalNum	Integer	Total Quantity for This Query
page	Integer	When page > 0 it indicates that there is more data, incrementing page by 1 will request subsequent data.
callRecordList	Array	Please refer to: Single Call Record Description

Single Call Record Description

Field	Type	Description
callId	String	Unique ID of the call
sdkAppId	Integer	Your sdkAppId
mediaType	String	Media type `video` Video Call `audio` Audio Call
roomId	String	Room ID of the call
startCallTs	Integer	Call Initiation Timestamp (in seconds)
acceptTs	Integer	Call Connection Timestamp (in seconds)
endTs	Integer	Call End Timestamp (in seconds)
caller	String	Caller userId
totalUserNumber	Integer	Total Number of Participants in the Call
callType	String	Call type

		`single` One-on-one call `group` Group Call
callResult	String	Call Result: `cancel` Cancel: The caller cancelled the call before connection. `reject` Declined call: The callee rejected. `not_answer` Not Answered: The callee did not answer in time. `normal_end` Completed: The call was connected and ended normally. `call_busy` Busy: The line was busy during the call. `interrupt` Interrupts: The call was interrupted due to network or other reasons.
callees	Array	List of callee user IDs who participated in the call

Note:

The callResult field may display all types for one-on-one calls only, while group chats only have `normal_end` .

Error codes

Unless there is a network error (e.g., a 502 error), the HTTP return code for this interface is always 200. The actual error code and error message are conveyed through errorCode and errorMessage in the response body.

For common error codes (70000 to 79999), please see Error Code.

Error code	Description
0	Request succeeded
50001	The current application needs to purchase the TUICallKit Group Call Version Package to use
70011	No records found
Unknown error code	Unknown error, please submit a ticket to contact technical staff

Client APIs (TUICallKit)
Android
API Overview

Last updated：2024-07-25 17:49:30

TUICallKit (UI Included)

TUICallKit is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat.

API	Description
createInstance	Create a TUICallKit instance (singleton mode).
setSelfInfo	Set user's avatar and nickname.
call	Make a one-to-one call.
call	Makes a one-to-one call, Support for custom room ID, call timeout, offline push content, etc
groupCall	Make a group call.
groupCall	Makes a group call, Support for custom room ID, call timeout, offline push content, etc
joinInGroupCall	Join a group call.
setCallingBell	Set the ringtone.
enableMuteMode	Set whether to turn on the mute mode.
enableFloatWindow	Set whether to enable floating windows.
enableIncomingBanner	Sets whether to display incoming banner.

TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description
createInstance	Create a `TUICallEngine` instance (singleton).
destroyInstance	Destroy `TUICallEngine` instance (singleton).
init	Authenticates the basic audio/video call capabilities.
addObserver	Add listener.
removeObserver	Remove listener.
call	Make a one-to-one call.
groupCall	Make a group call.
accept	Accept call.
reject	Reject call.
hangup	Hang up call.
ignore	Ignore call.
inviteUser	Invite users to the current group call.
joinInGroupCall	Join a group call.
switchCallMediaType	Switch the call media type, such as from video call to audio call.
startRemoteView	Subscribes to the video stream of a remote user.
stopRemoteView	Unsubscribes from the video stream of a remote user.
openCamera	Turns the camera on.
closeCamera	Turns the camera off.
switchCamera	Switches the camera.
openMicrophone	Enables the mic.
closeMicrophone	Disables the mic.
selectAudioPlaybackDevice	Selects the audio playback device (Earpiece/Speakerphone).
setSelfInfo	Set user's avatar and nickname.
enableMultiDeviceAbility	Sets whether to enable multi-device login for `TUICallEngine` (supported by the Group Call package).

setVideoRenderParams	Set the rendering mode of video.
setVideoEncoderParams	Set the encoding parameters of video encoder.
getTRTCCloudInstance	Advanced features.
setBeautyLevel	Set beauty level, support turning off default beauty.

TUICallObserver

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

API	Description
onError	An error occurred during the call.
onCallReceived	A call was received.
onCallCancelled	The call was canceled.
onCallBegin	The call was connected.
onCallEnd	The call ended.
onCallMediaTypeChanged	The call type changed.
onUserReject	A user declined the call.
onUserNoResponse	A user didn't respond.
onUserLineBusy	A user was busy.
onUserJoin	A user joined the call.
onUserLeave	A user left the call.
onUserVideoAvailable	Whether a user has a video stream.
onUserAudioAvailable	Whether a user has an audio stream.
onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.
onKickedOffline	The current user was kicked offline.
onUserSigExpired	The user sig is expired.

Definitions of Key Types

API	Description
TUICallDefine.MediaType	Call media type, Enumeration type: Unknown, Video, and Audio.
TUICallDefine.Role	Call role, Enumeration type: None, Caller, and Called.
TUICallDefine.Status	Call status, Enumeration type: None, Waiting, and Accept.
TUICommonDefine.RoomId	The room ID, which can be a number or string.
TUICommonDefine.Camera	The camera type. Enumeration type: Front and Back.
TUICommonDefine.AudioPlaybackDevice	The audio playback device type. Enumeration type: Earpiece and Speakerphone.
TUICommonDefine.NetworkQualityInfo	The current network quality.

TUICallKit

Last updated：2024-07-24 10:20:03

TUICallKit APIs

`TUICallKit` is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat. For directions on integration, see Integrating TUICallKit.

API Overview

API	Description
createInstance	Creates a `TUICallKit` instance (singleton mode).
setSelfInfo	Sets the alias and profile photo.
call	Makes a one-to-one call.
call	Makes a one-to-one call, Support for custom room ID, call timeout, offline push content, etc
groupCall	Makes a group call.
groupCall	Makes a group call, Support for custom room ID, call timeout, offline push content, etc
joinInGroupCall	Joins a group call.
setCallingBell	Sets the ringtone.
enableMuteMode	Sets whether to turn on the mute mode.
enableFloatWindow	Sets whether to enable floating windows.
enableIncomingBanner	Sets whether to display incoming banner.

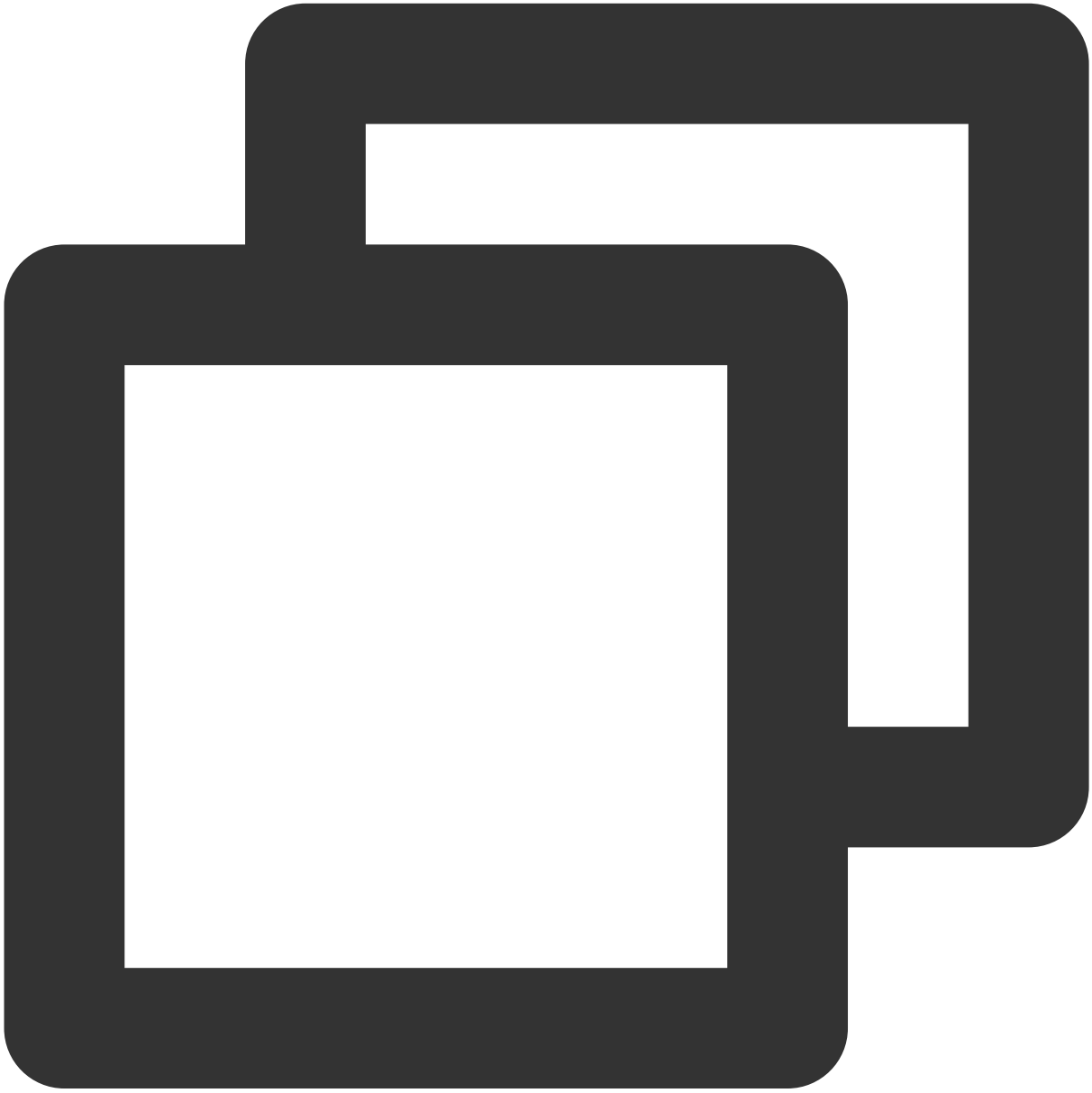
Details

createInstance

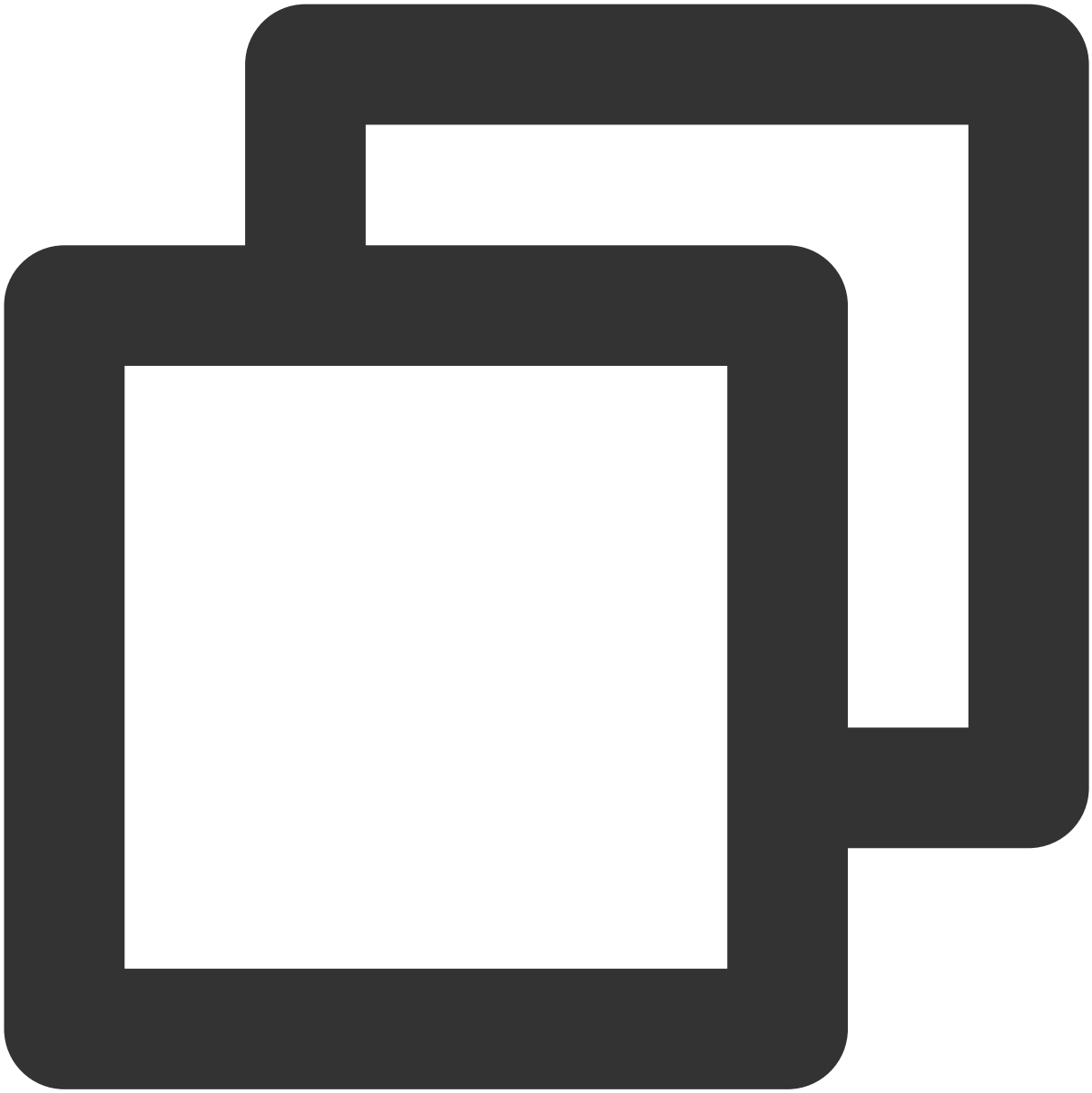
This API is used to create a `TUICallKit` singleton.

Kotlin

Java



```
fun createInstance(context: Context): TUICallKit
```



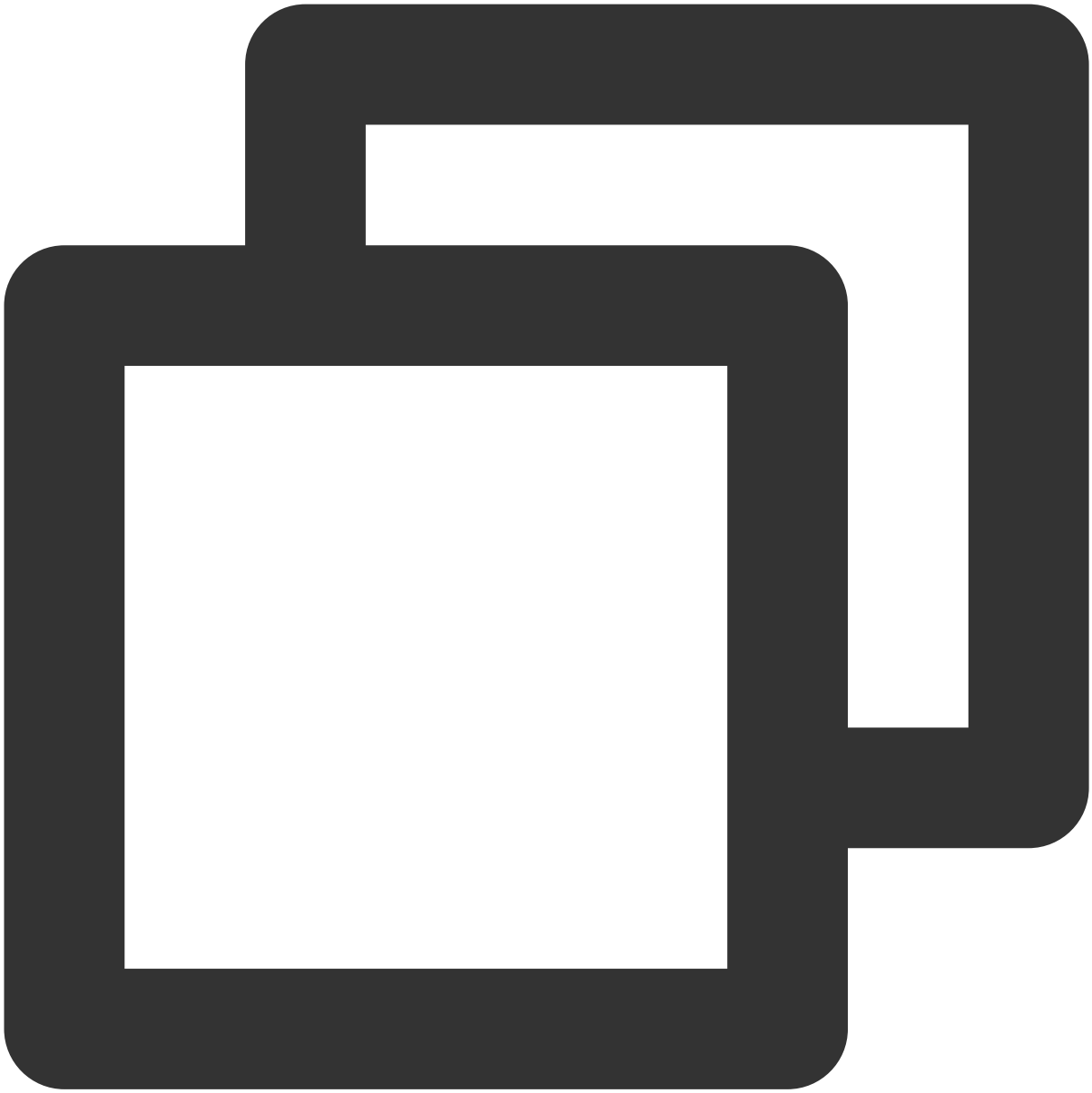
```
TUICallKit createInstance(Context context)
```

setSelfInfo

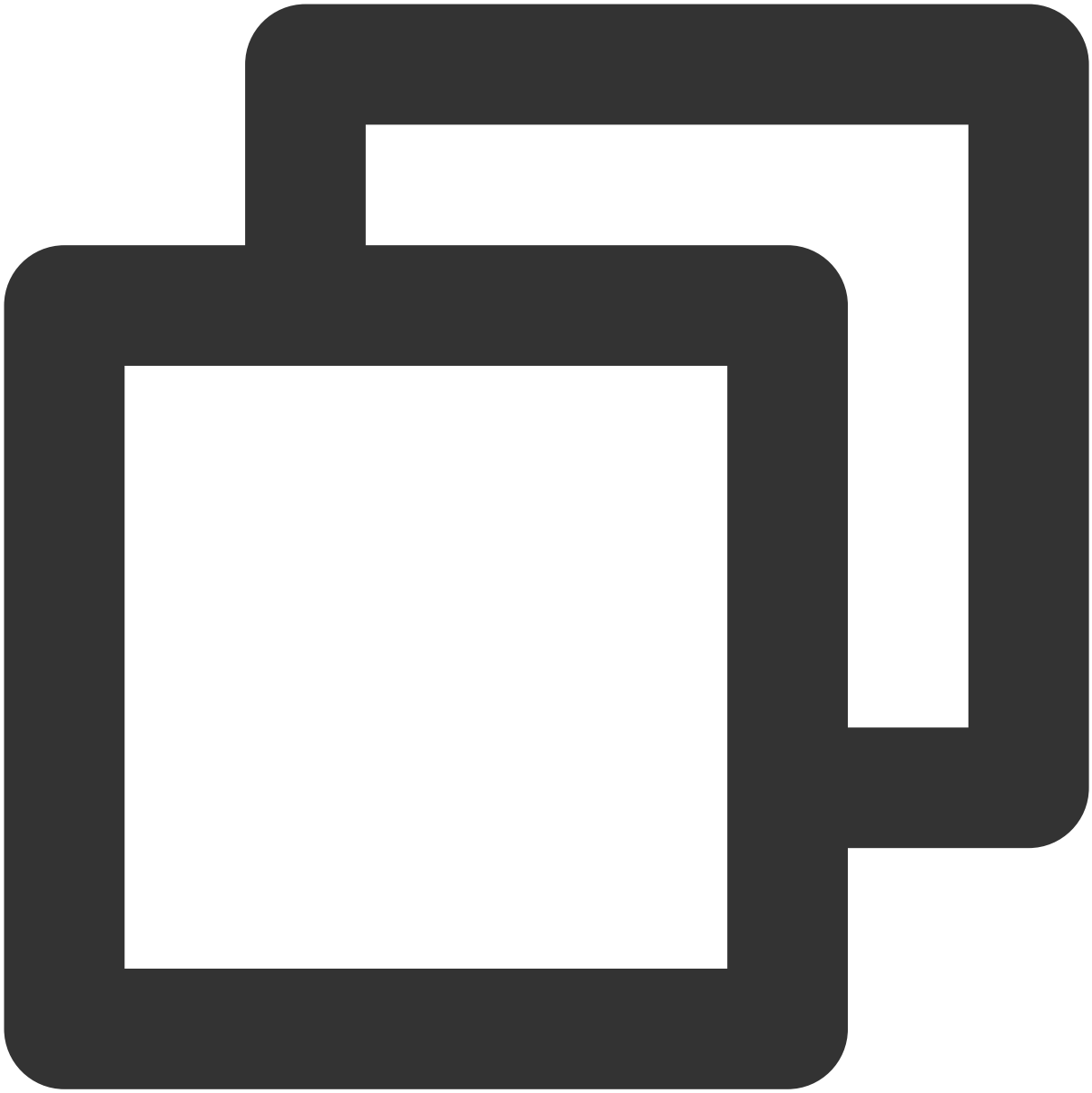
This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.

Kotlin

Java



```
fun setSelfInfo(nickname: String?, avatar: String?, callback: TUICommonDefine.Callb
```



```
void setSelfInfo(String nickname, String avatar, TUICommonDefine.Callback callback)
```

The parameters are described below:

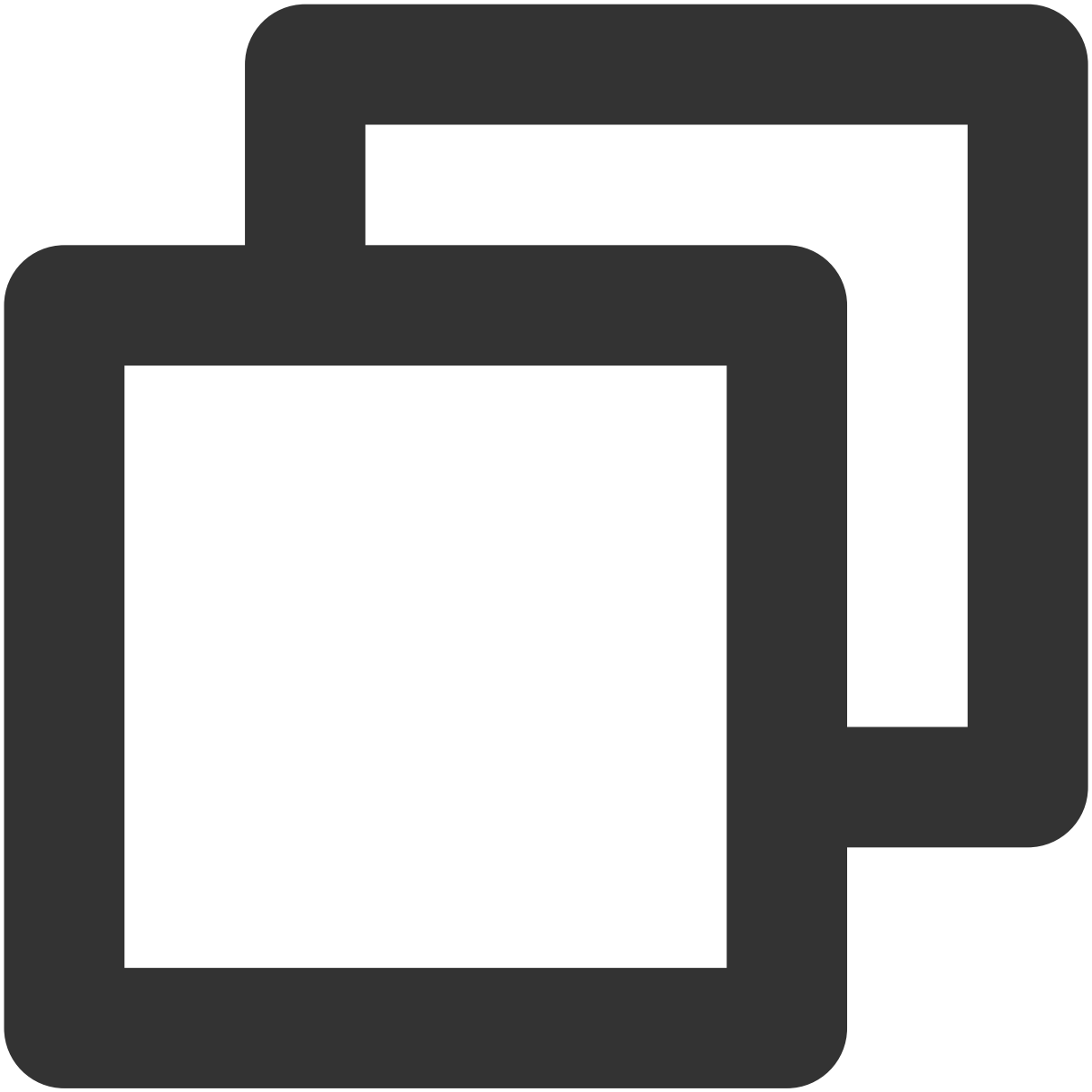
Parameter	Type	Description
nickname	String	The alias.
avatar	String	The profile photo.

call

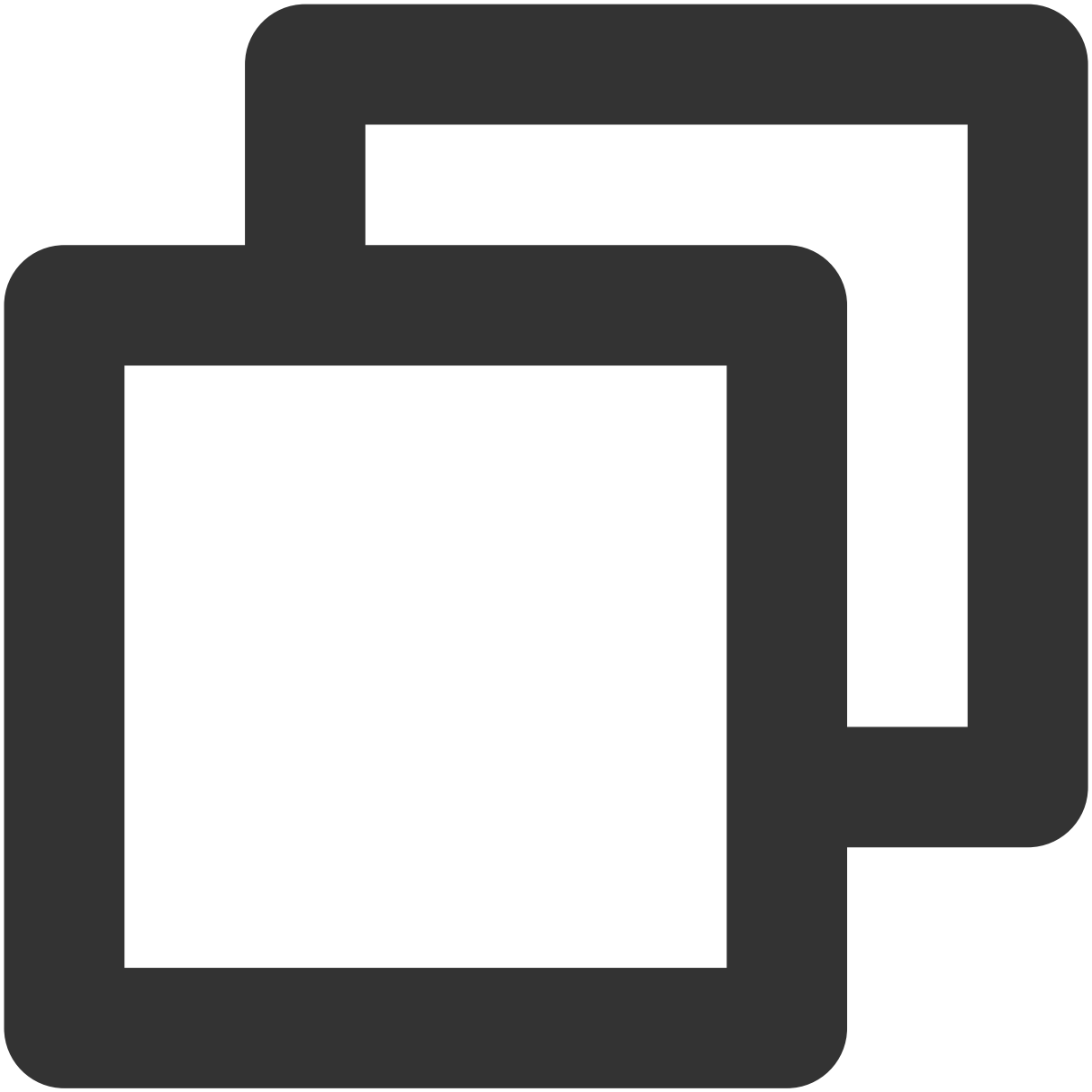
This API is used to make a (one-to-one) call.

Kotlin

Java



```
fun call(userId: String, callMediaType: TUICallDefine.MediaType)
```



```
void call(String userId, TUICallDefine.MediaType callMediaType)
```

The parameters are described below:

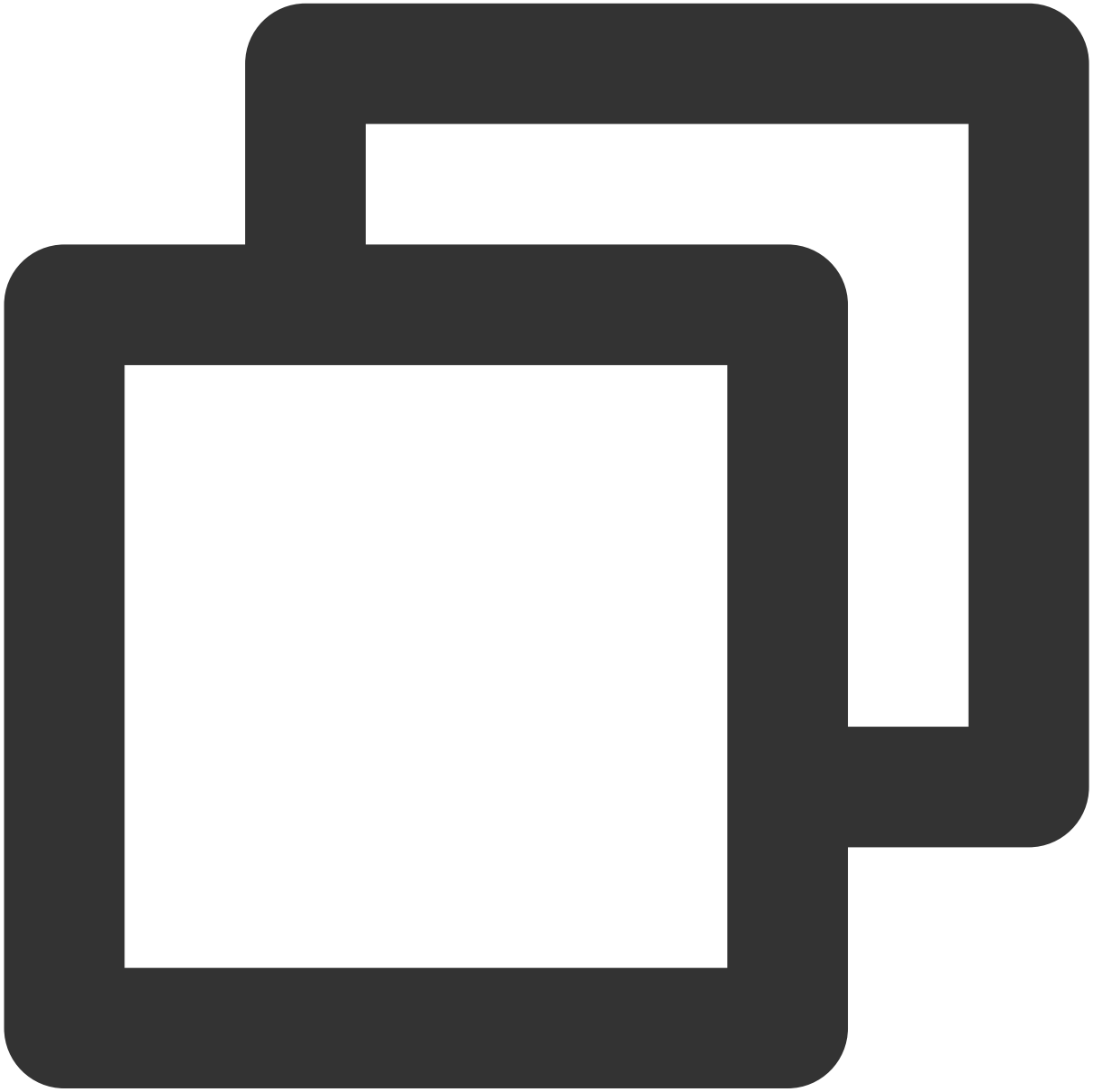
Parameter	Type	Description
userId	String	The target user ID.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.

call

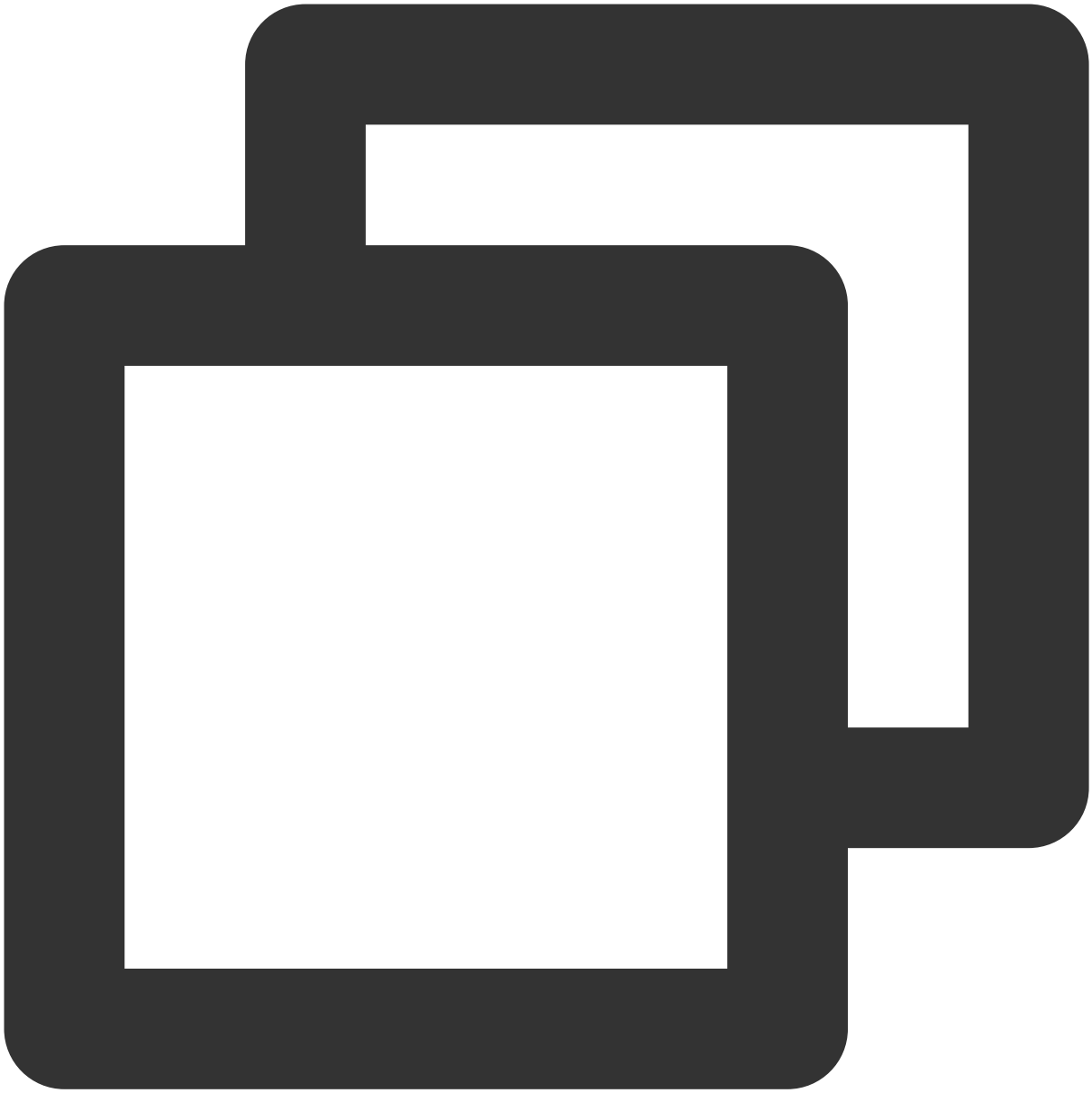
This API is used to make a (one-to-one) call, Support for custom room ID, call timeout, offline push content, etc.

Kotlin

Java



```
fun call(
    userId: String, callMediaType: TUICallDefine.MediaType,
    params: CallParams?, callback: TUICommonDefine.Callback?
)
```



```
void call(String userId, TUICallDefine.MediaType callMediaType,
          TUICallDefine.CallParams params, TUICommonDefine.Callback callback)
```

The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.

params	TUICallDefine.CallParams	Call extension parameters, such as roomID, call timeout, offline push info,etc

groupCall

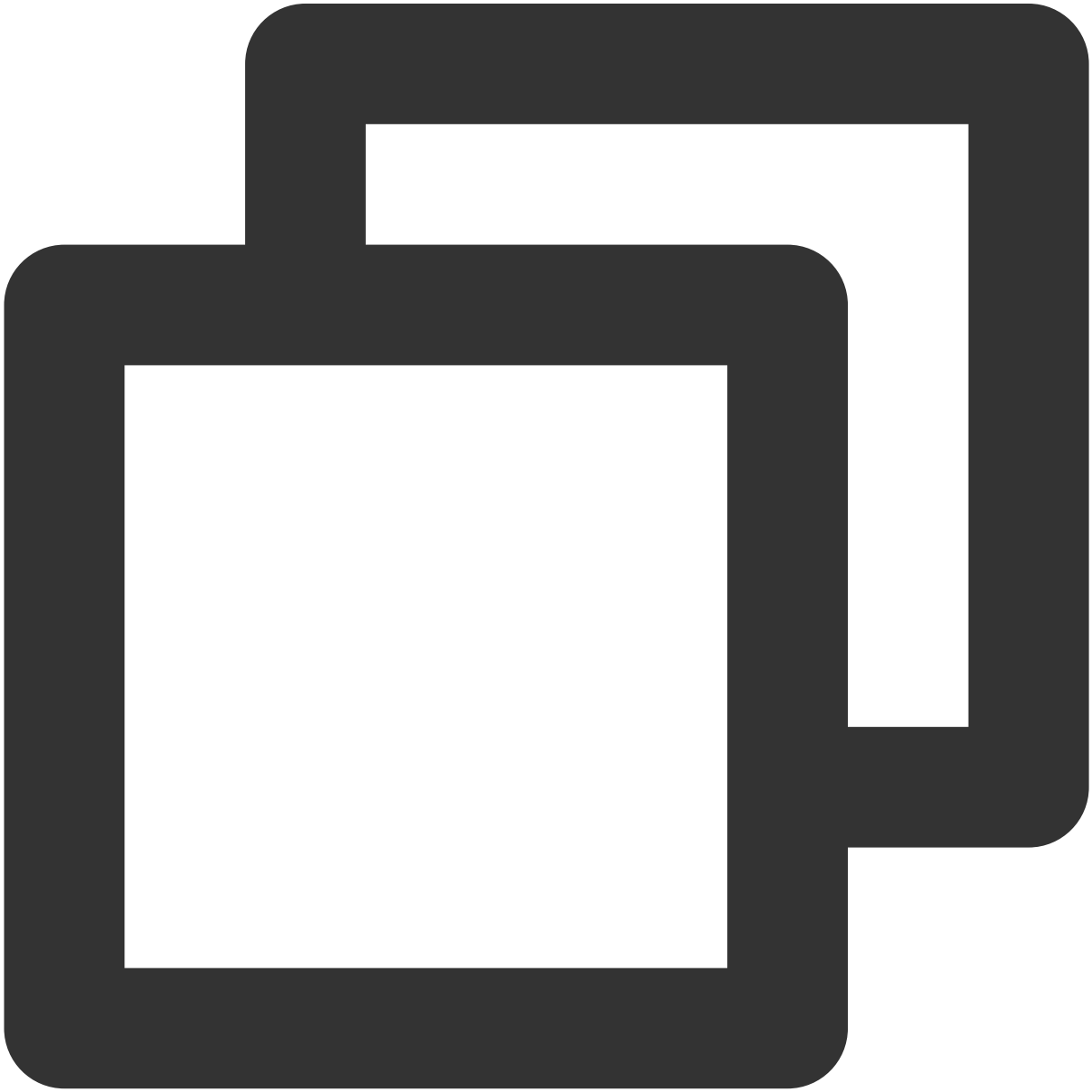
This API is used to make a group call.

Note:

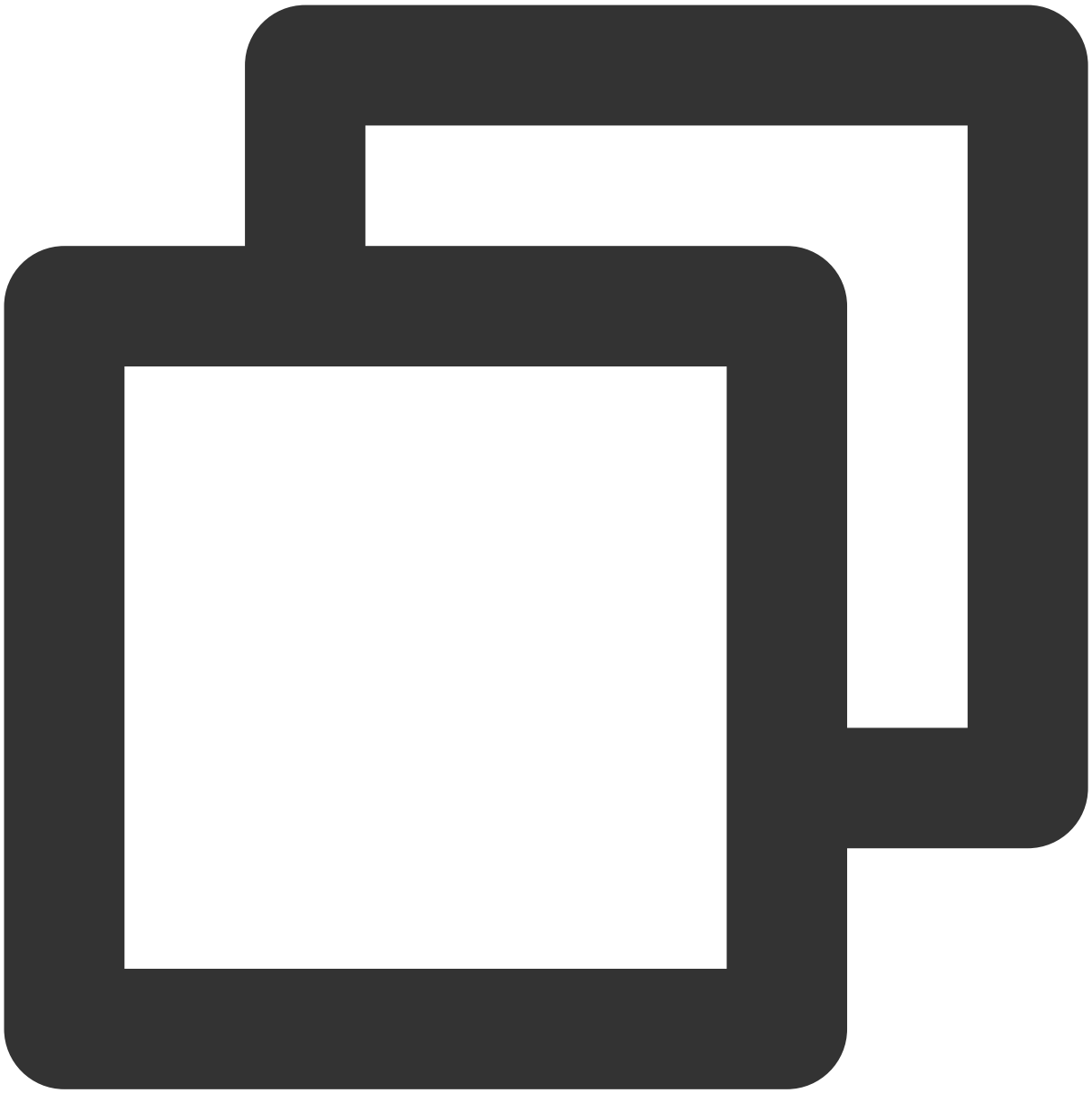
Before making a group call, you need to create an Chat group first.

Kotlin

Java



```
fun groupCall(groupId: String, userIdList: List<String?>?, callMediaType: TUICallDe
```



```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal
```

The parameters are described below:

Parameter	Type	Description
groupId	String	The group ID.
userIdList	List	The target user IDs.

callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.

groupCall

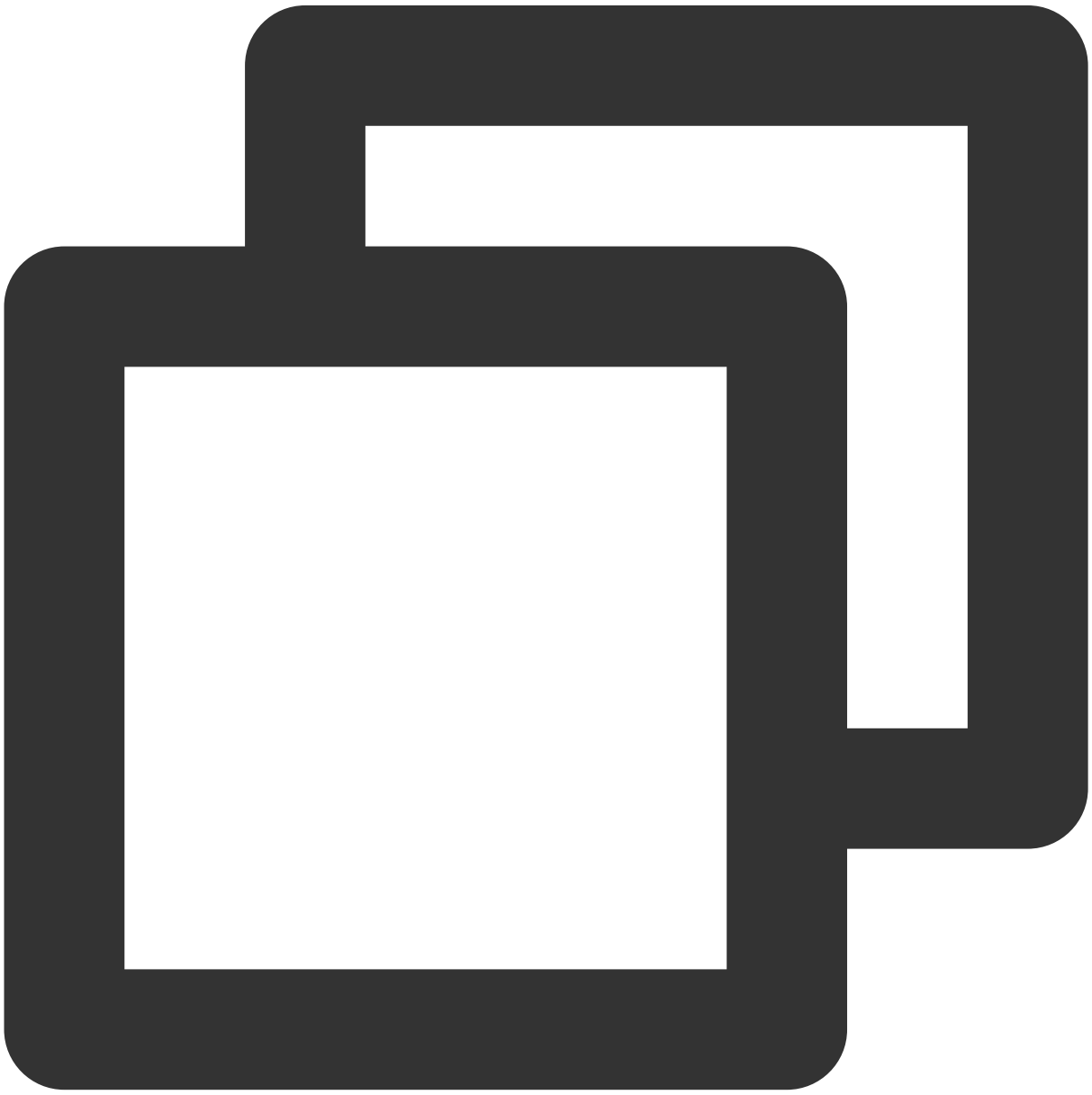
This API is used to make a group call, Support for custom room ID, call timeout, offline push content, etc.

Note:

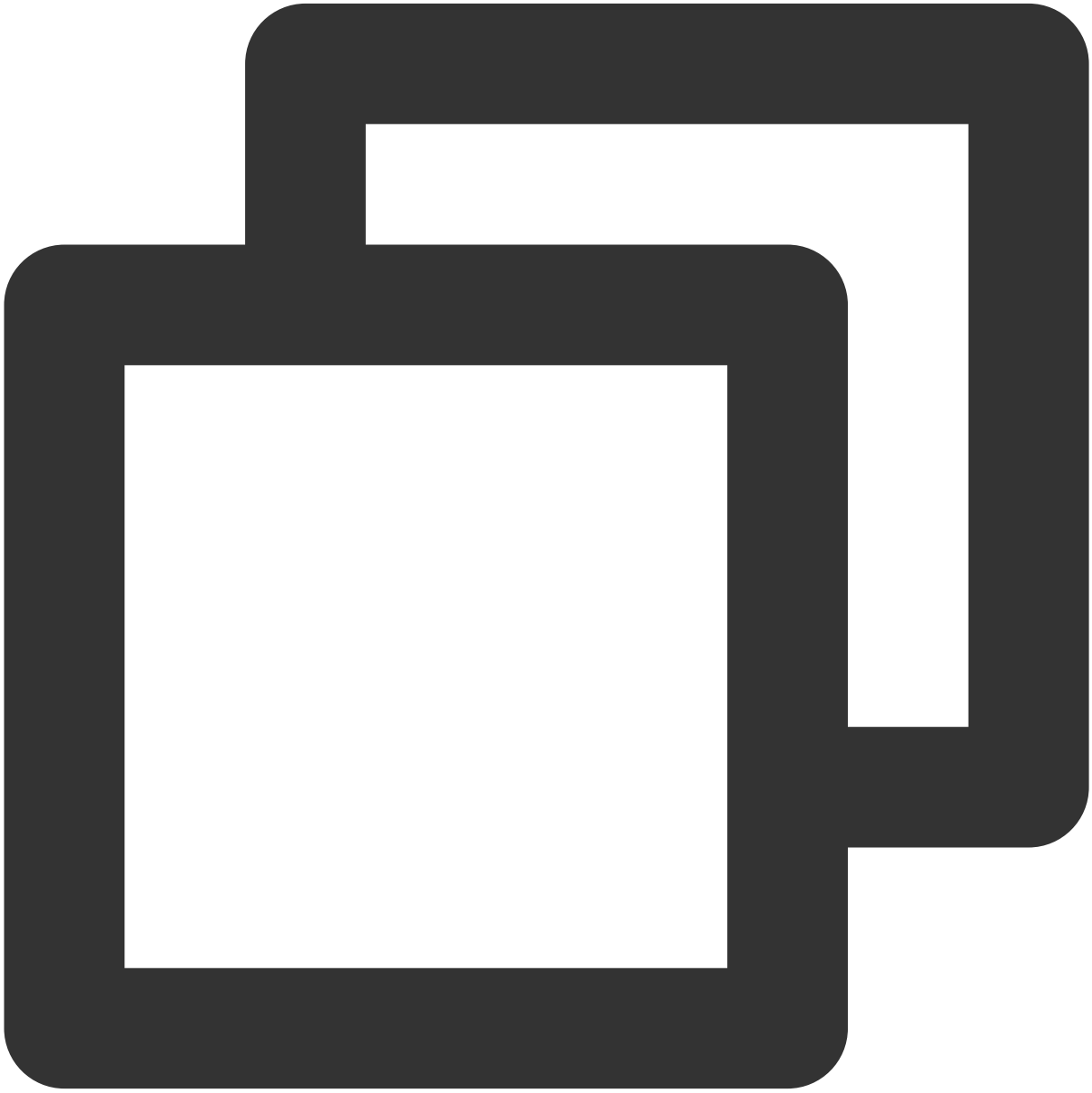
Before making a group call, you need to create an Chat group first.

Kotlin

Java



```
fun groupCall(
    groupId: String, userIdList: List<String?>?,
    callMediaType: TUICallDefine.MediaType, params: CallParams?,
    callback: TUICommonDefine.Callback?
)
```



```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal
               TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

The parameters are described below:

Parameter	Type	Description
groupId	String	The group ID.
userIdList	List	The target user IDs.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.
params	TUICallDefine.CallParams	Call extension parameters, such as roomID, call timeout, offline push info,etc

joinInGroupCall

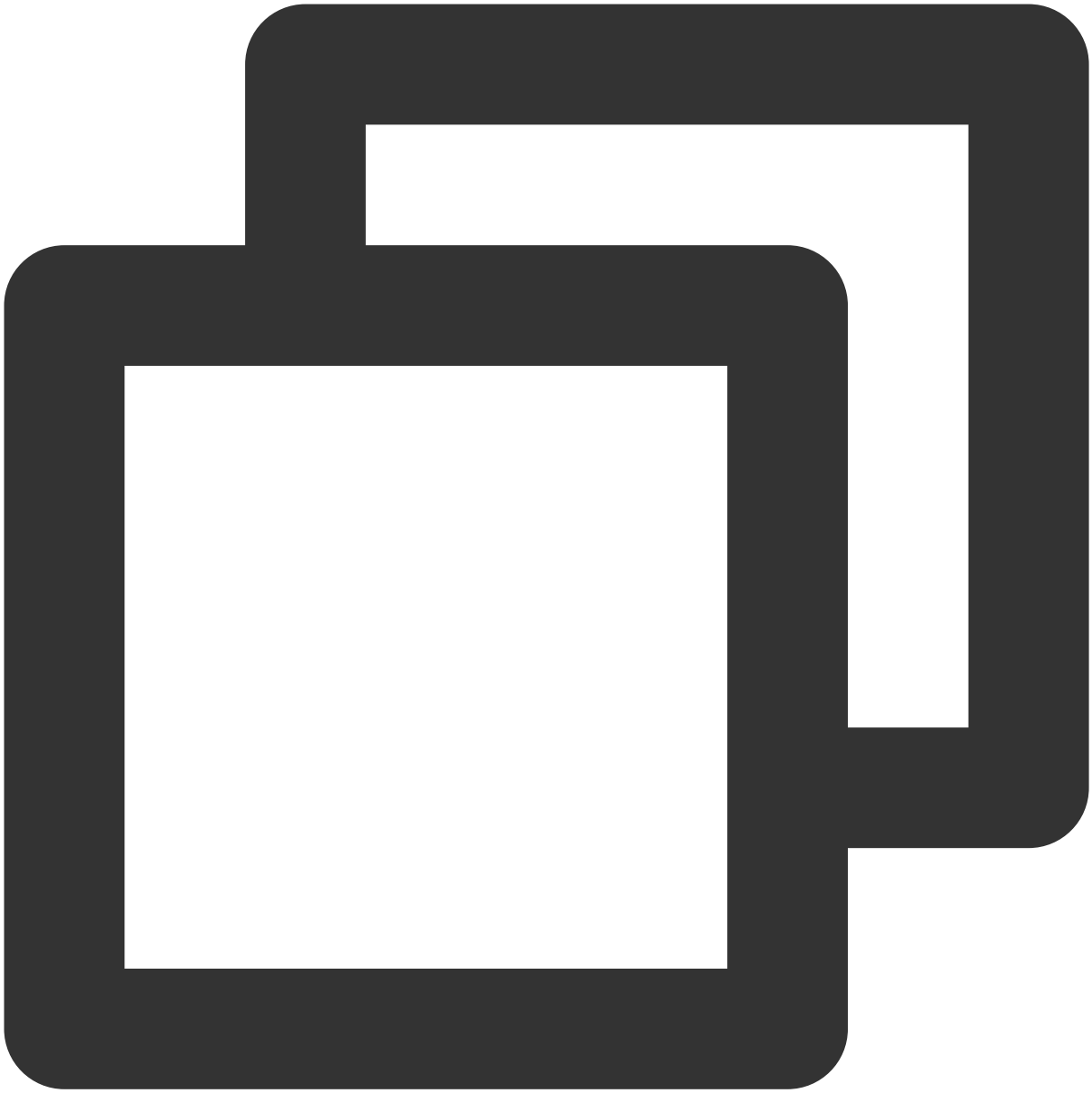
This API is used to join a group call.

Note:

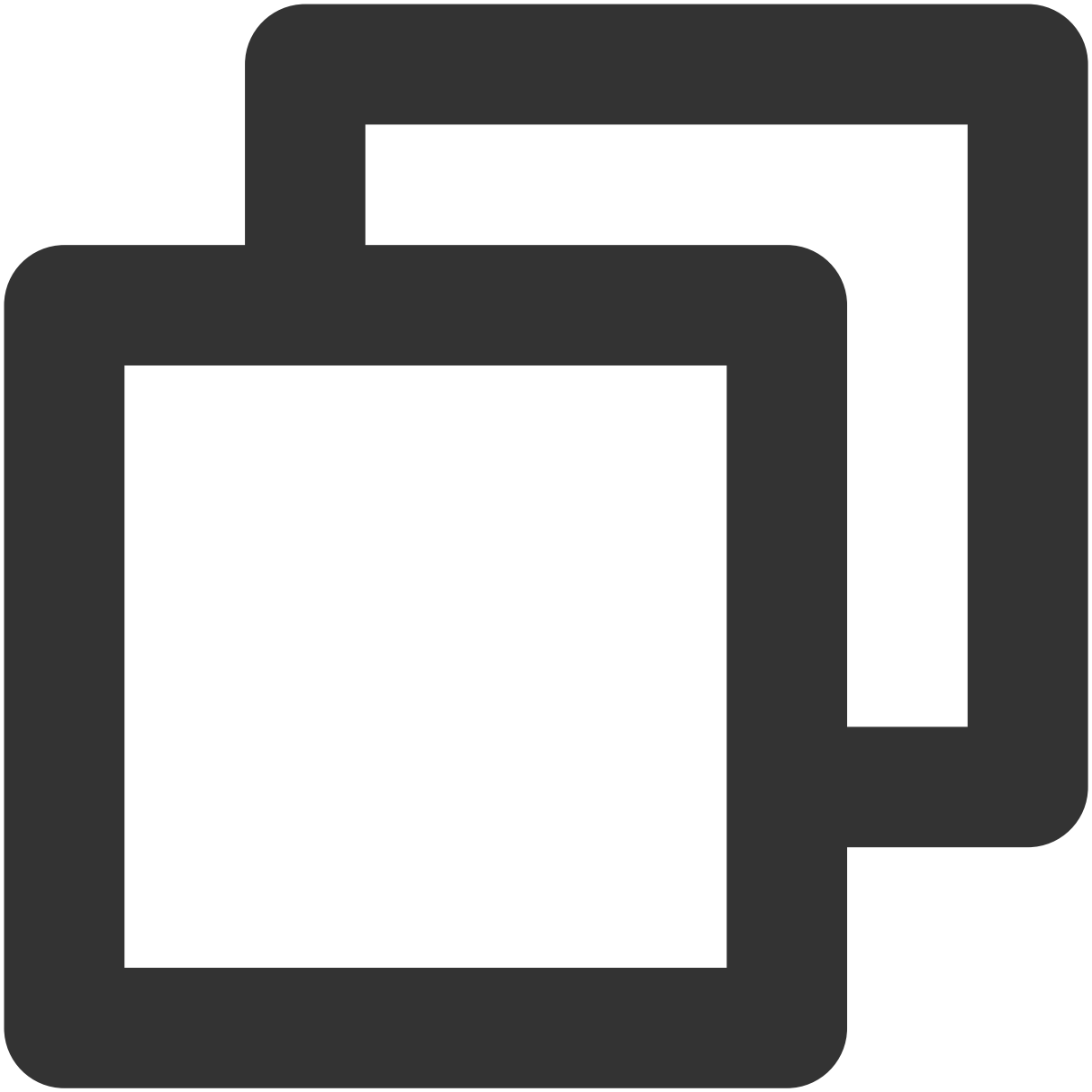
Before joining a group call, you need to create or join an Chat group in advance, and there are already users in the group who are in the call.

Kotlin

Java



```
fun joinInGroupCall(roomId: RoomId?, groupId: String?, callMediaType: TUICallDefine
```



```
void joinInGroupCall(TUICommonDefine.RoomId roomId, String groupId, TUICallDefine.M
```

The parameters are described below:

Parameter	Type	Description
roomId	TUICommonDefine.RoomId	The room ID.
groupId	String	The group ID.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.

setCallingBell

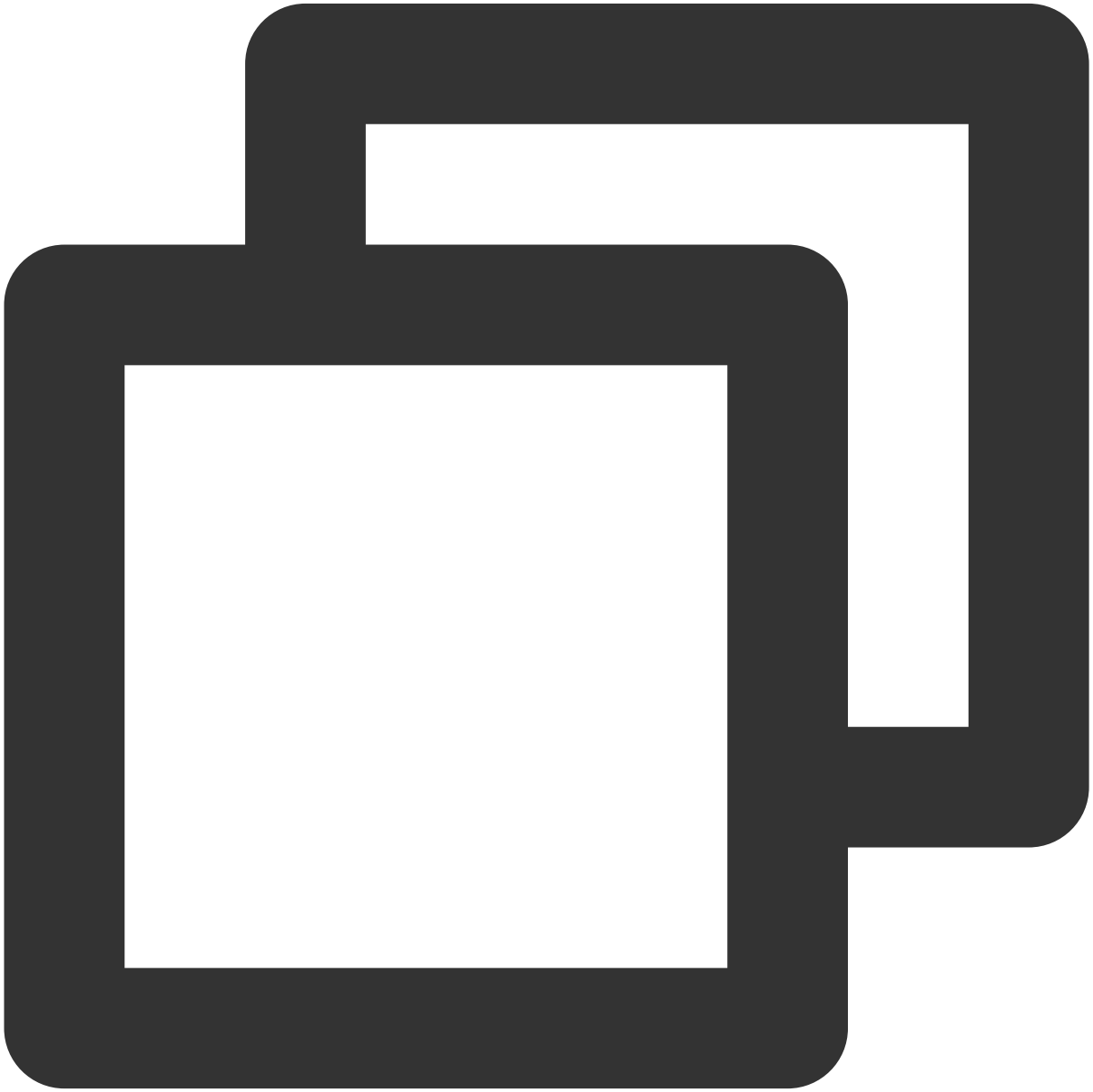
This API is used to set the ringtone. `filePath` must be an accessible local file URL.

The ringtone set is associated with the device and does not change with user.

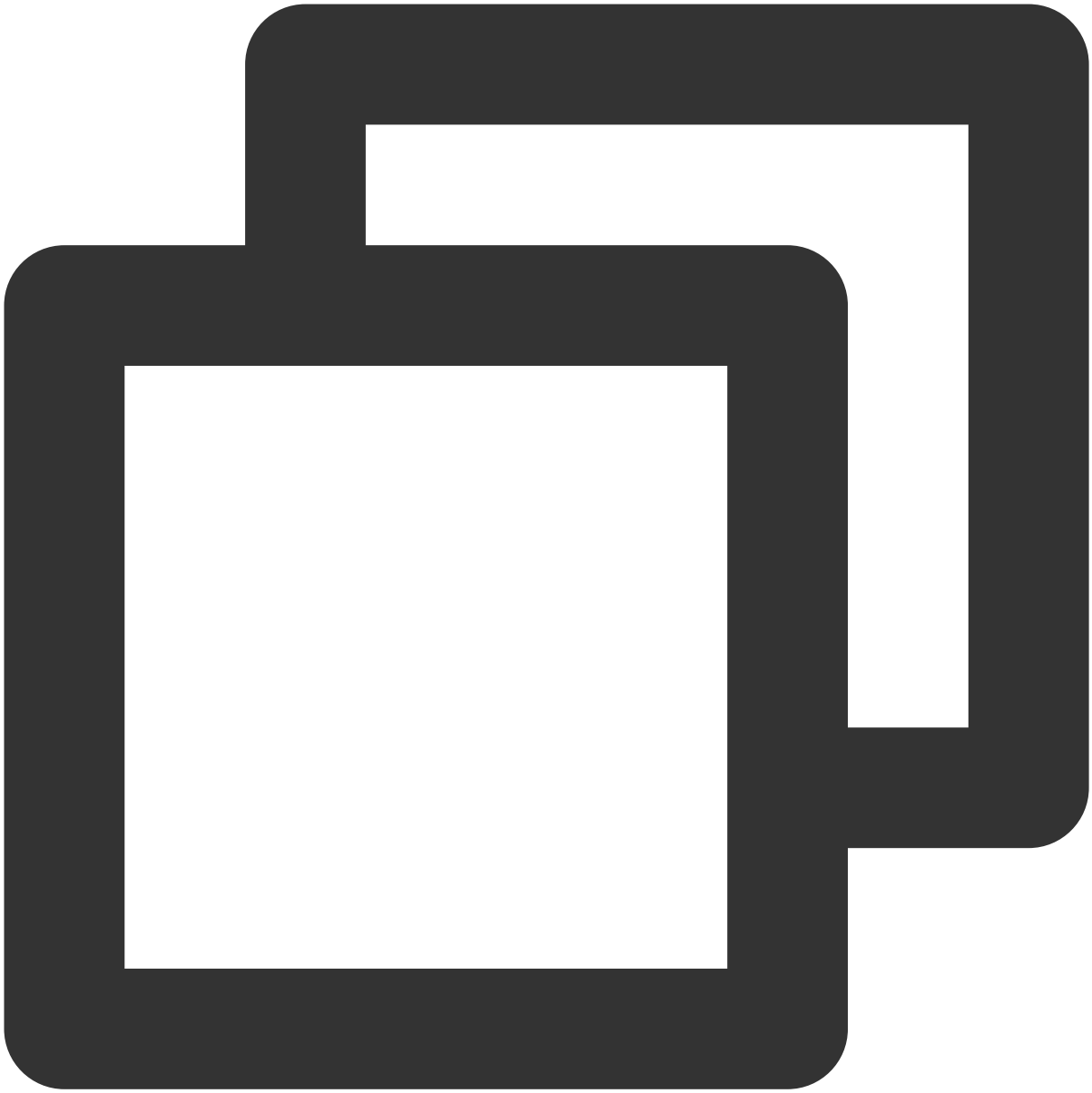
To reset the ringtone, pass in an empty string for `filePath` .

Kotlin

Java



```
fun setCallingBell(filePath: String?)
```



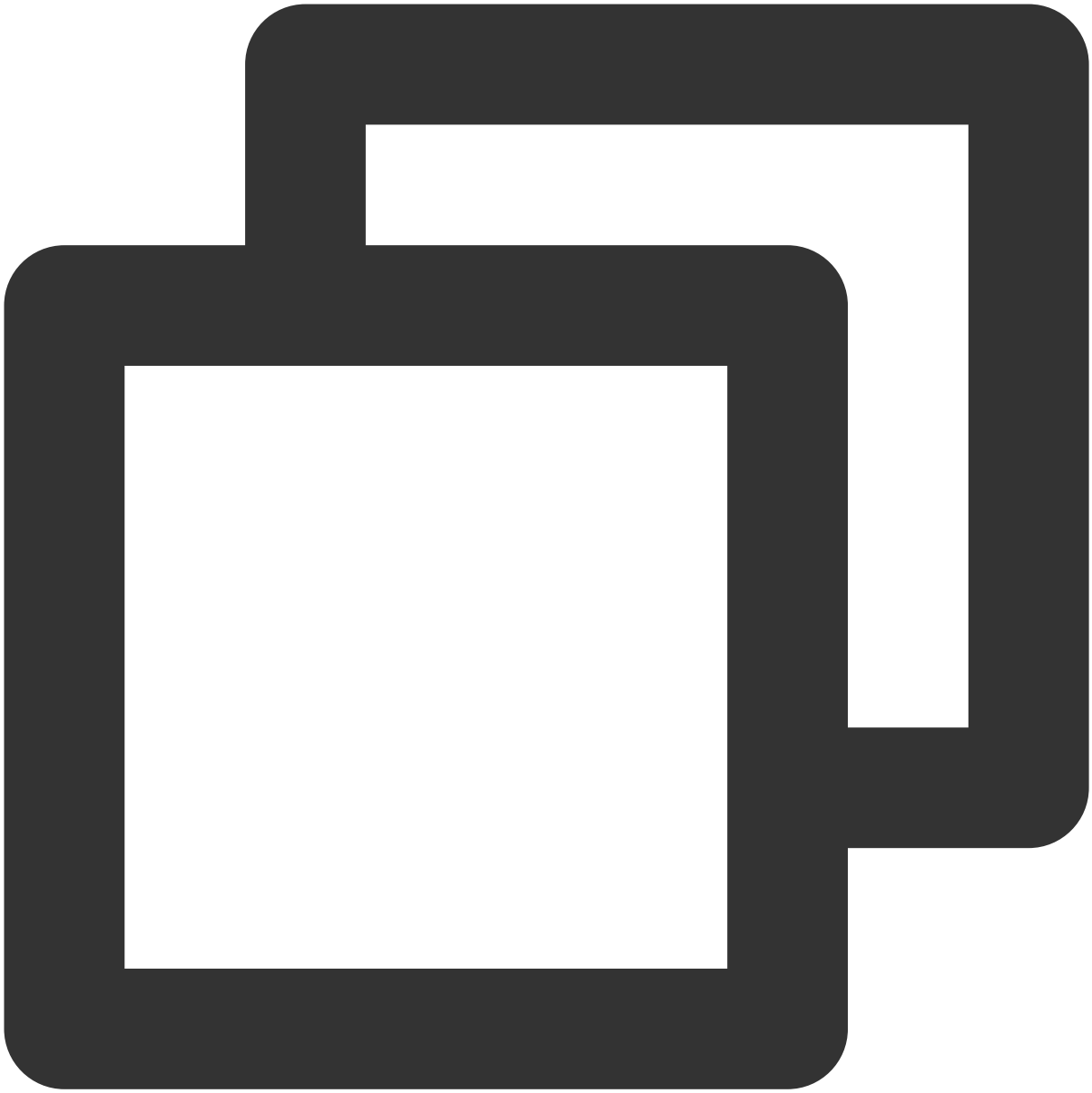
```
void setCallingBell(String filePath);
```

enableMuteMode

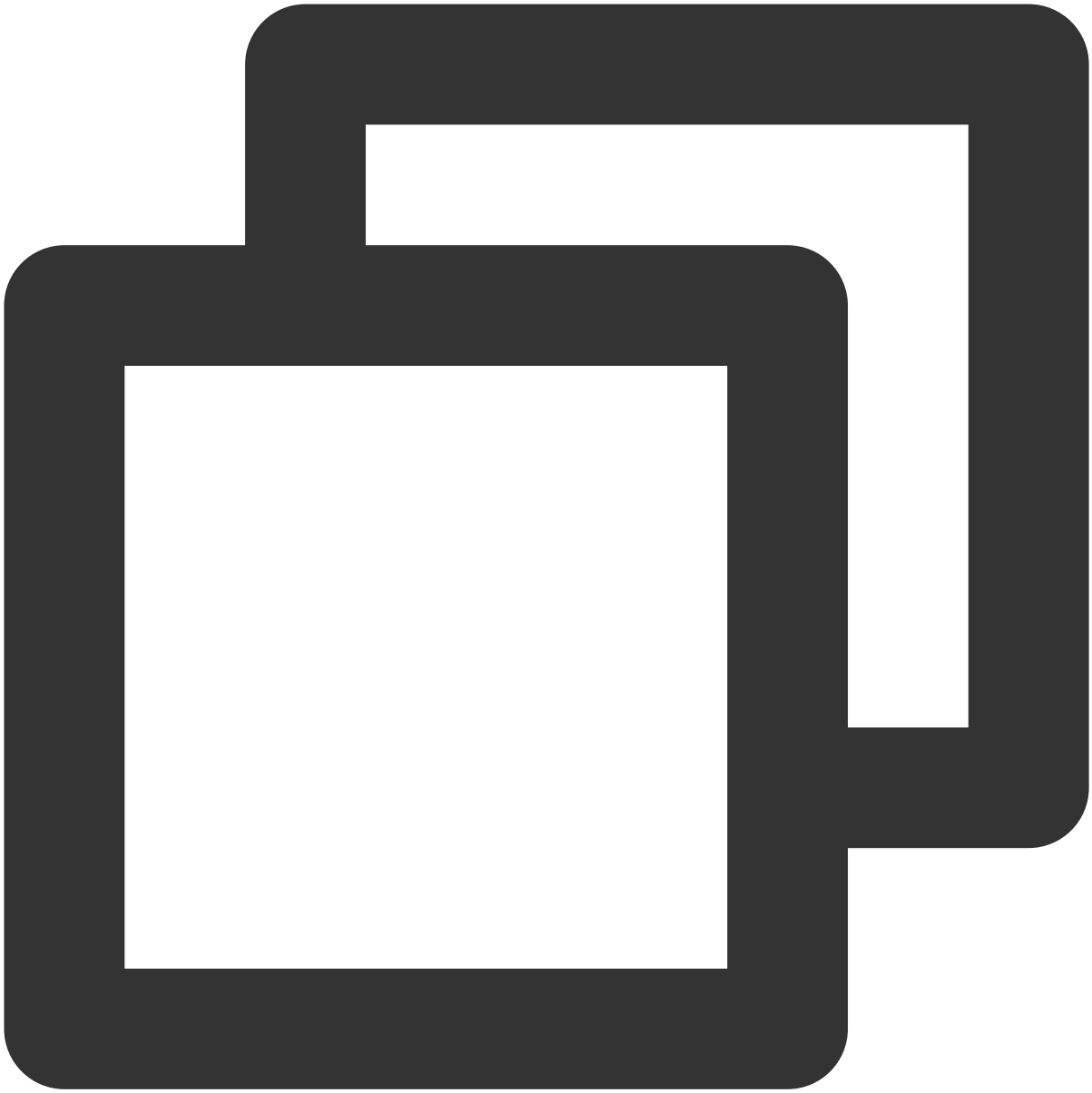
This API is used to set whether to play the music when user received a call.

Kotlin

Java



```
fun enableMuteMode(enable: Boolean)
```



```
void enableMuteMode(boolean enable);
```

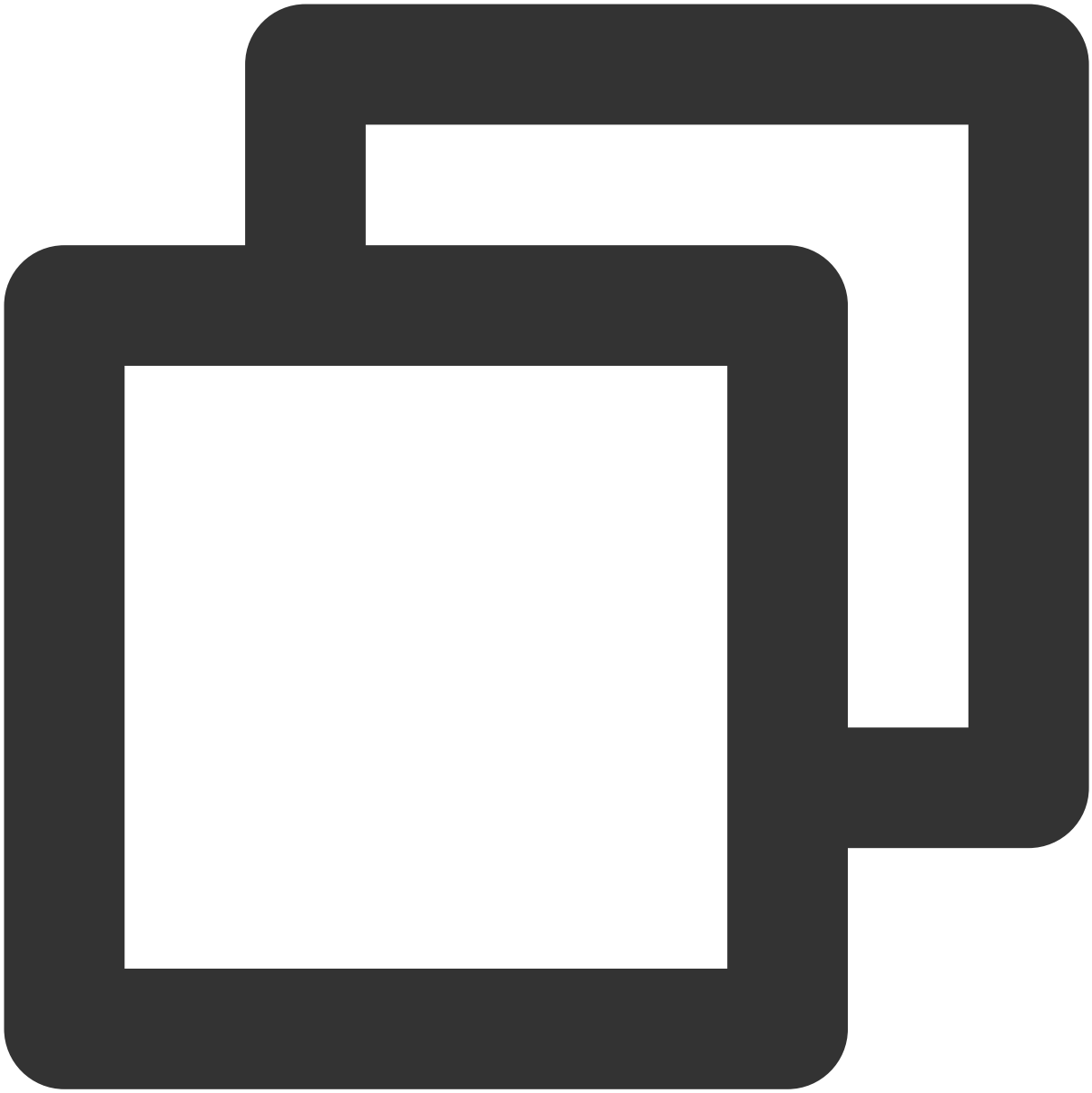
enableFloatWindow

This API is used to set whether to enable floating windows.

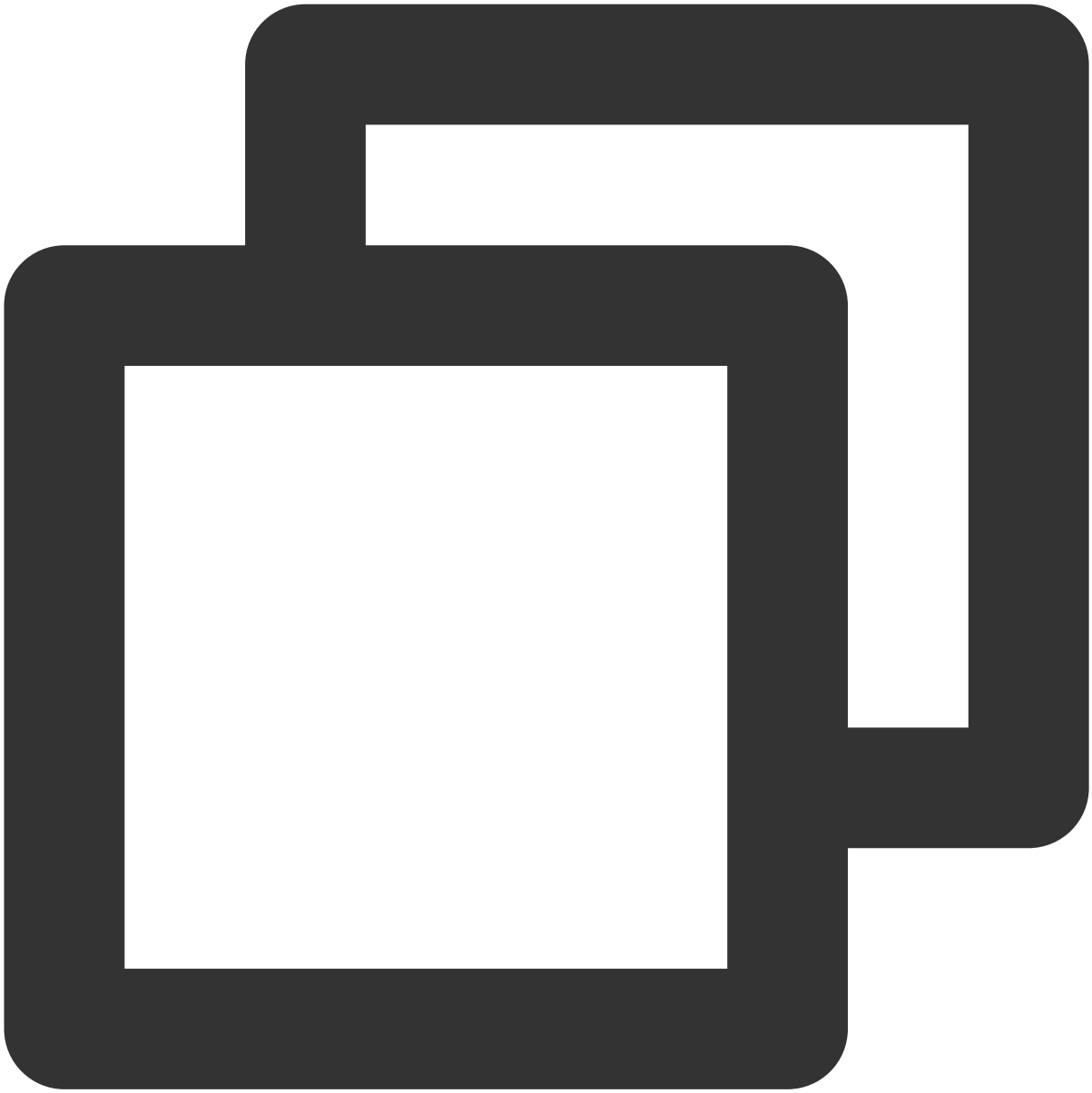
The default value is `false`, and the floating window button in the top left corner of the call view is hidden. If it is set to `true`, the button will become visible.

Kotlin

Java



```
fun enableFloatWindow(enable: Boolean)
```

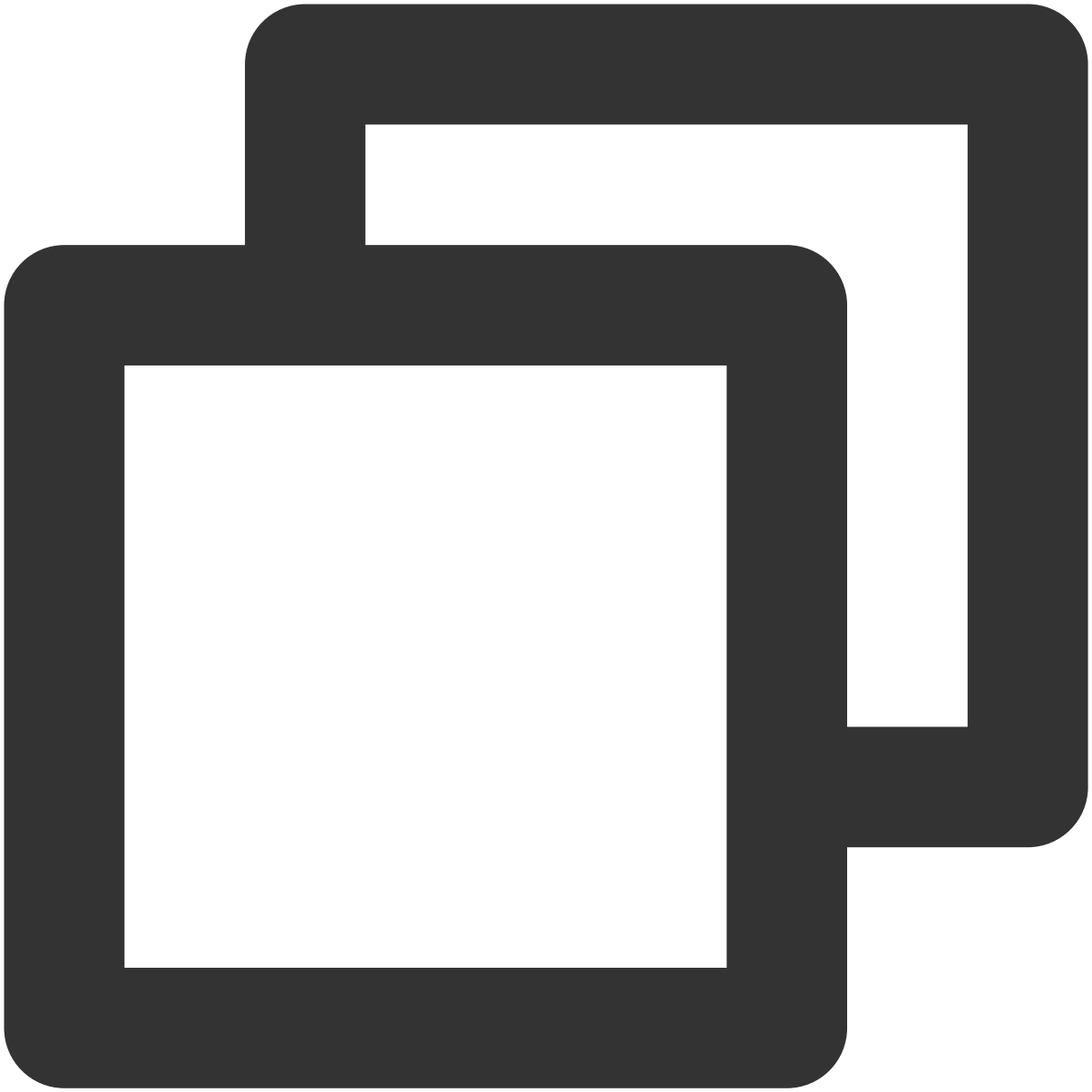


```
void enableFloatWindow(boolean enable);
```

enableIncomingBanner

The API is used to set whether show incoming banner when user received a new call invitation.

The default value is `false` , The callee will pop up a full-screen call view by default when receiving the invitation. If it is set to `true` , the callee will display a banner first.



```
fun enableIncomingBanner(enable: Boolean)
```

TUICallEngine

Last updated：2024-07-25 17:49:30

TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

Overview

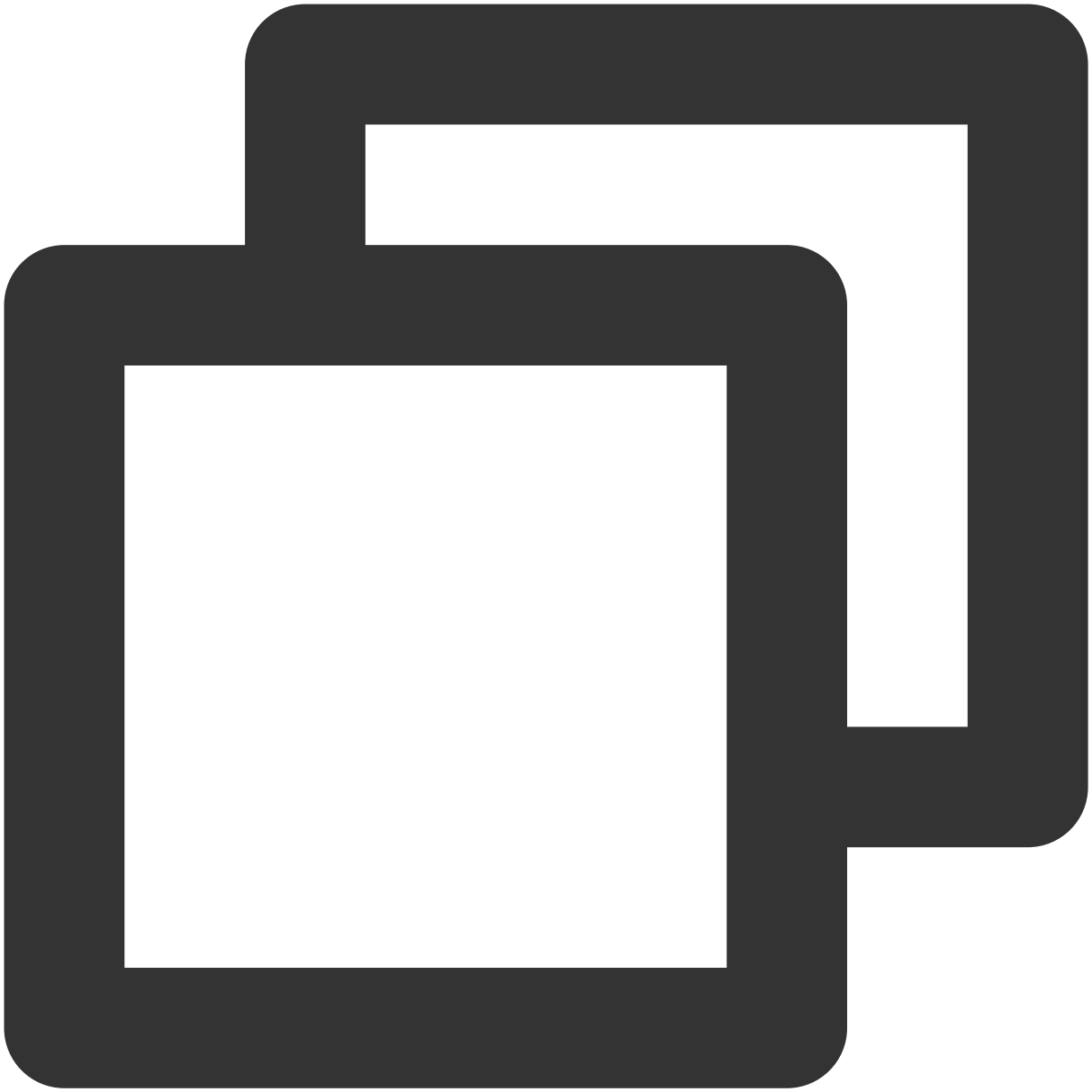
API	Description
createInstance	Creates a `TUICallEngine` instance (singleton mode).
destroyInstance	Terminates a `TUICallEngine` instance (singleton mode).
init	Authenticates the basic audio/video call capabilities.
addObserver	Registers an event listener.
removeObserver	Unregisters an event listener.
call	Makes a one-to-one call.
groupCall	Makes a group call.
accept	Accepts a call.
reject	Rejects a call.
hangup	Ends a call.
ignore	Ignores a call.
inviteUser	Invites users to the current group call.
joinInGroupCall	Joins a group call.
switchCallMediaType	Changes the call type, for example, from video call to audio call.
startRemoteView	Subscribes to the video stream of a remote user.
stopRemoteView	Unsubscribes from the video stream of a remote user.

openCamera	Turns the camera on.
closeCamera	Turns the camera off.
switchCamera	Switches between the front and rear cameras.
openMicrophone	Turns the mic on.
closeMicrophone	Turns the mic off.
selectAudioPlaybackDevice	Selects the audio playback device (receiver or speaker).
setSelfInfo	Sets the alias and profile photo.
enableMultiDeviceAbility	Sets whether to enable multi-device login for `TUICallEngine` (supported by the Group Call package).
setVideoRenderParams	Set the rendering mode of video image.
setVideoEncoderParams	Set the encoding parameters of video encoder.
getTRTCCloudInstance	Advanced features.
setBeautyLevel	Set beauty level, support turning off default beauty.

Details

createInstance

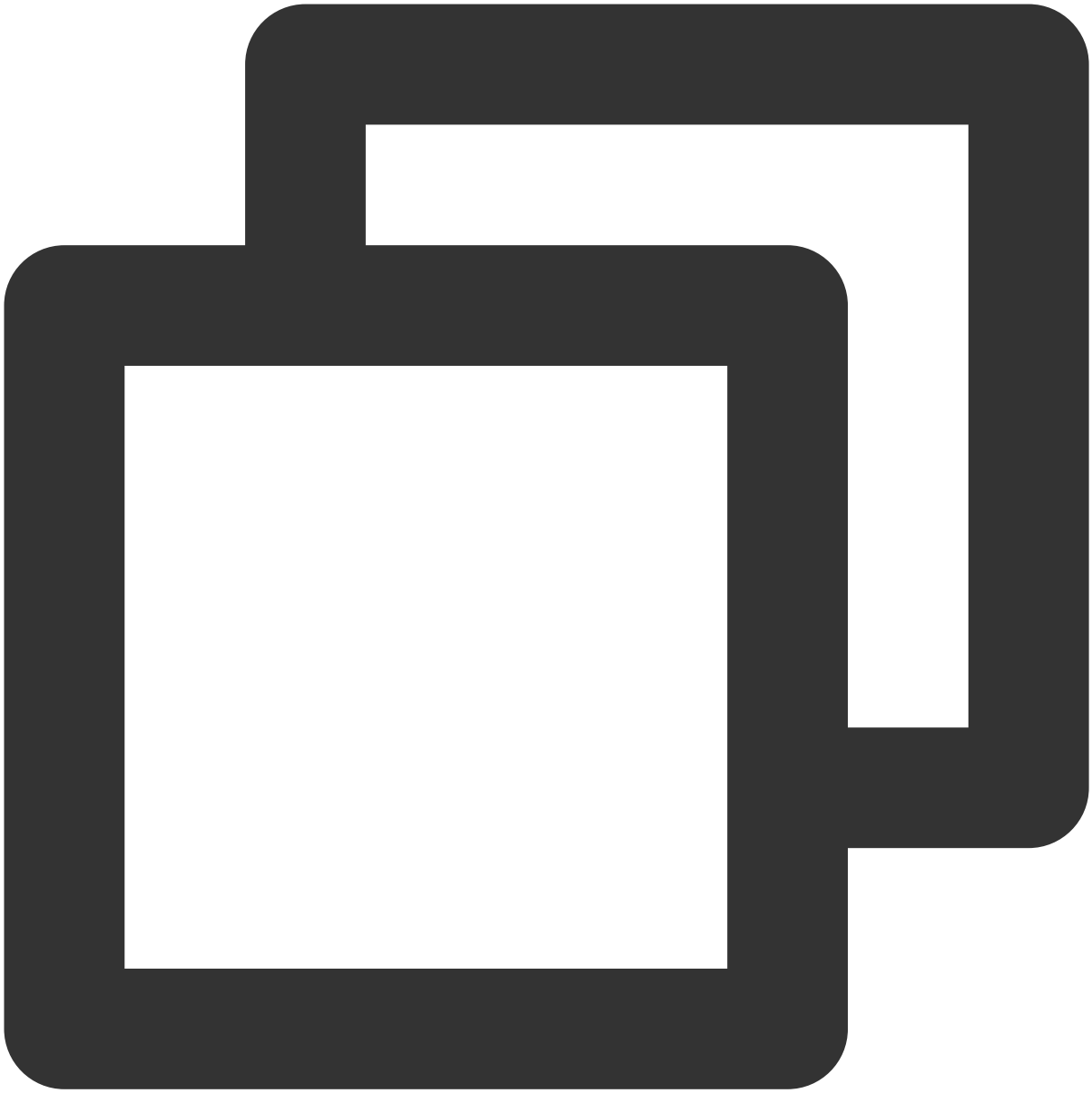
This API is used to create a `TUICallEngine` singleton.



```
TUICallEngine createInstance(Context context)
```

destroyInstance

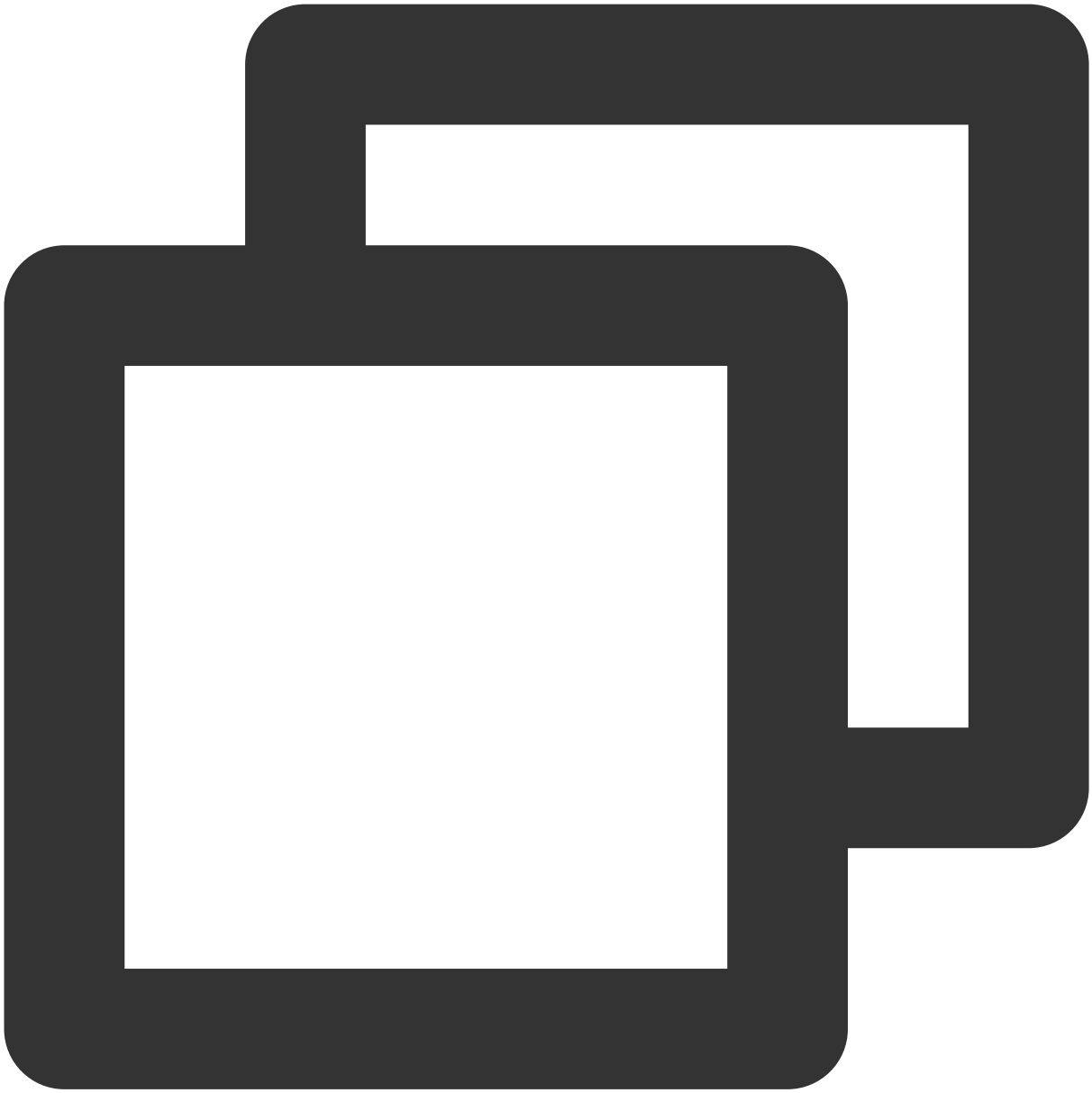
This API is used to terminate a `TUICallEngine` singleton.



```
void destroyInstance();
```

Init

This API is used to initialize `TUICallEngine` . Call it to authenticate the call service and perform other required actions before you call other APIs.



```
void init(int sdkAppId, String userId, String userSig, TUICommonDefine.Callback cal
```

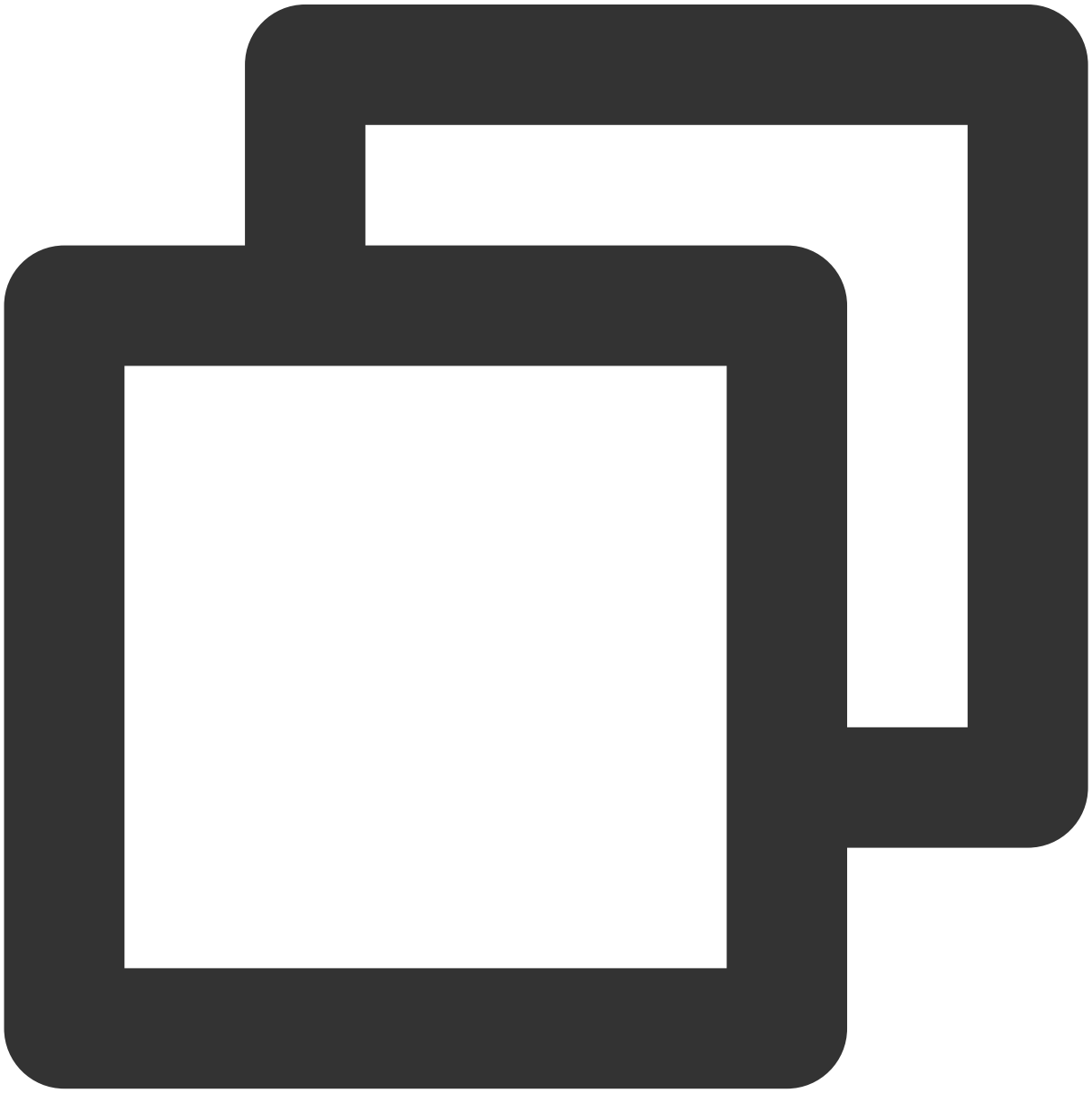
The parameters are described below:

Parameter	Type	Description
sdkAppId	int	You can view `SDKAppID` in Application Management > **Application Info** of the TRTC console.
userId	String	The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores

		(_).
userSig	String	Tencent Cloud's proprietary security signature. For how to calculate and use it, see UserSig.
callback	TUICommonDefine.Callback	The initialization callback. `onSuccess` indicates initialization is successful.

addObserver

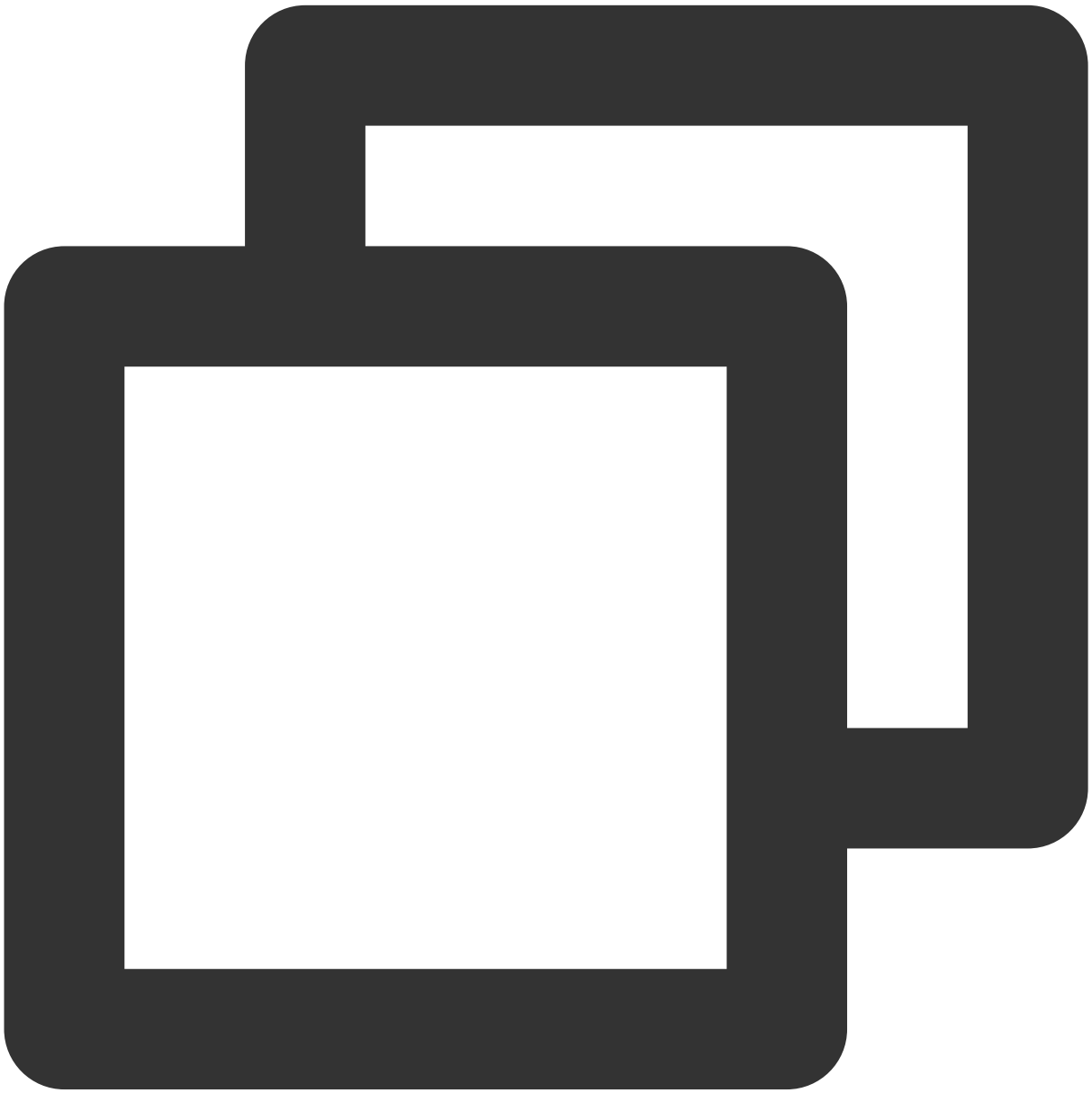
This API is used to register an event listener to listen for `TUICallObserver` events.



```
void addObserver(TUICallObserver observer);
```

removeObserver

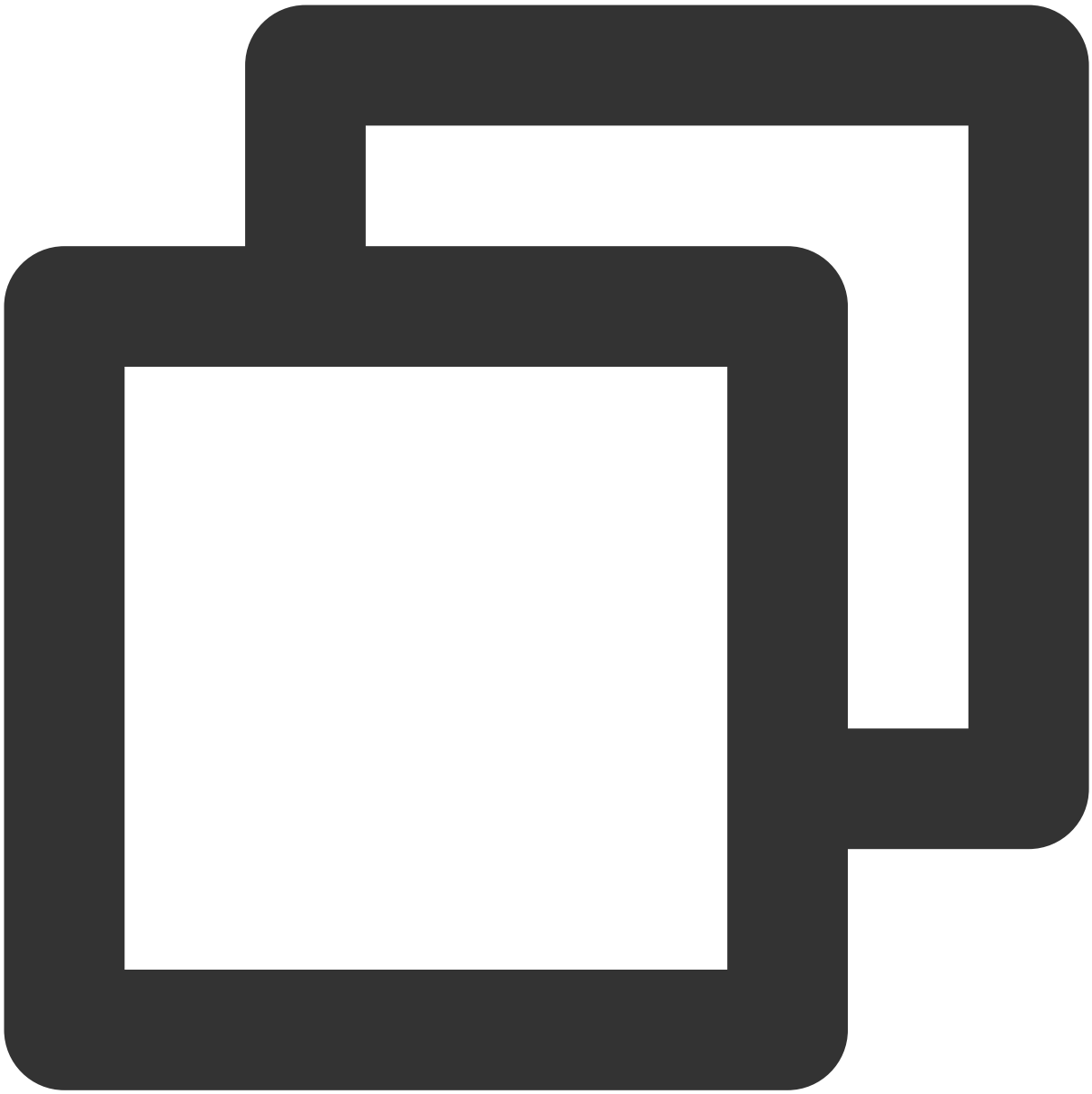
This API is used to unregister an event listener.



```
void removeObserver(TUICallObserver observer);
```

call

This API is used to make a (one-to-one) call.



```
void call(String userId, TUICallDefine.MediaType callMediaType,
          TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.

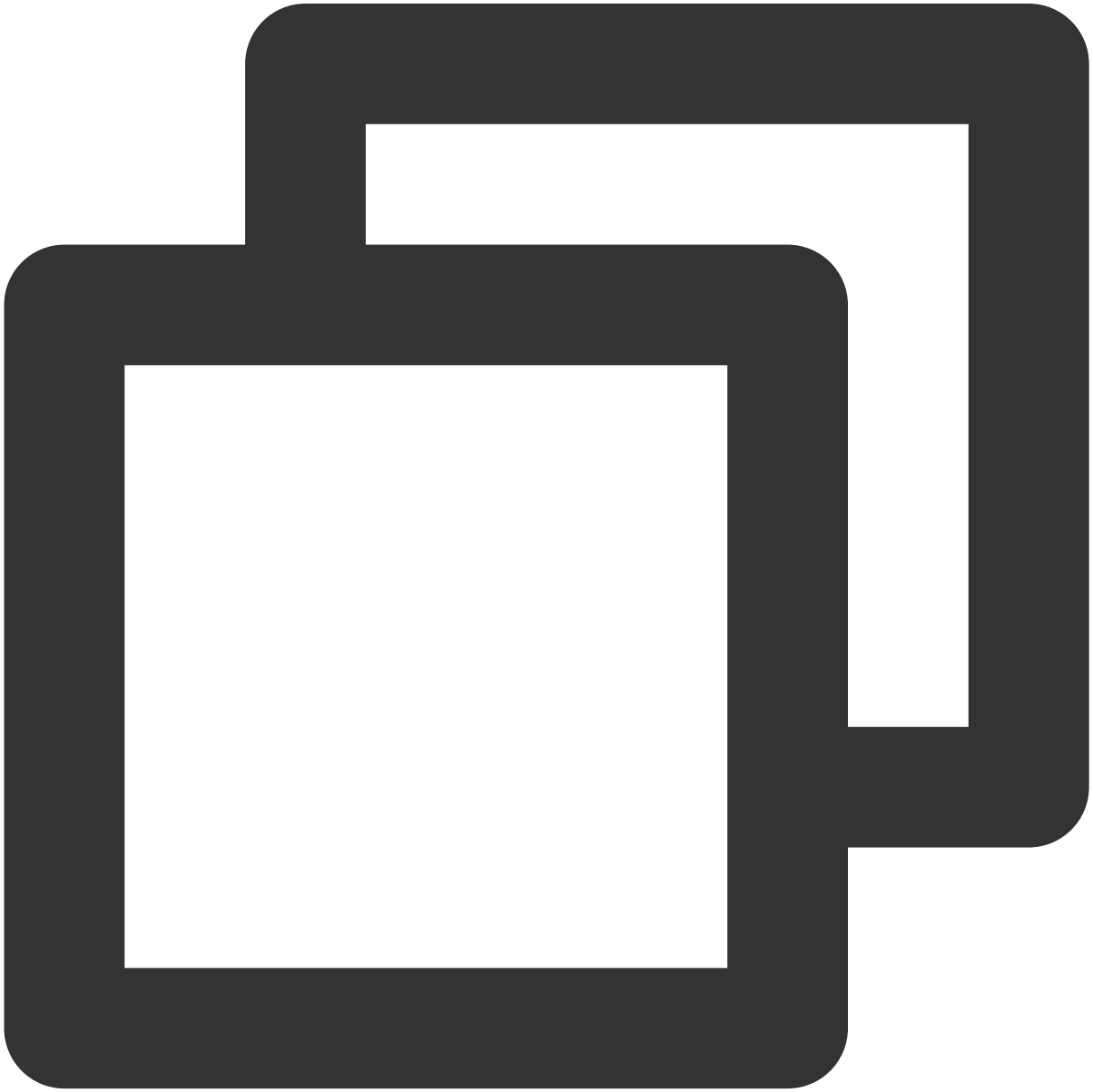
params	TUICallDefine.CallParams	An additional parameter, such as roomID, call timeout, offline push info,etc

groupCall

This API is used to make a group call.

Note：

Before making a group call, you need to create an Chat group first.



```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal
```

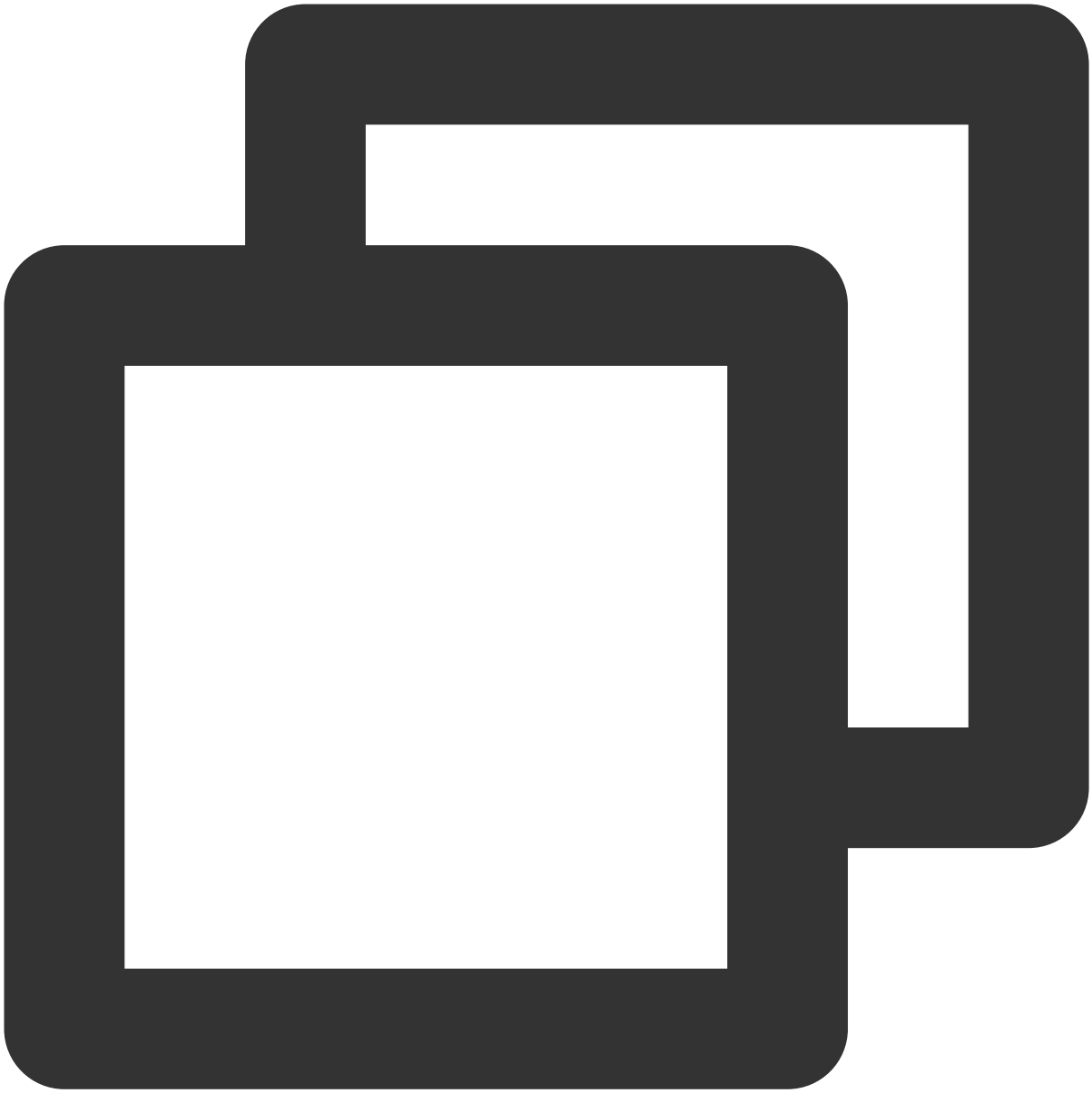
```
                TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

The parameters are described below:

Parameter	Type	Description
groupId	String	The group ID.
userIdList	List	The target user IDs.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.
params	TUICallDefine.CallParams	An additional parameter. such as roomID, call timeout, offline push info,etc

accept

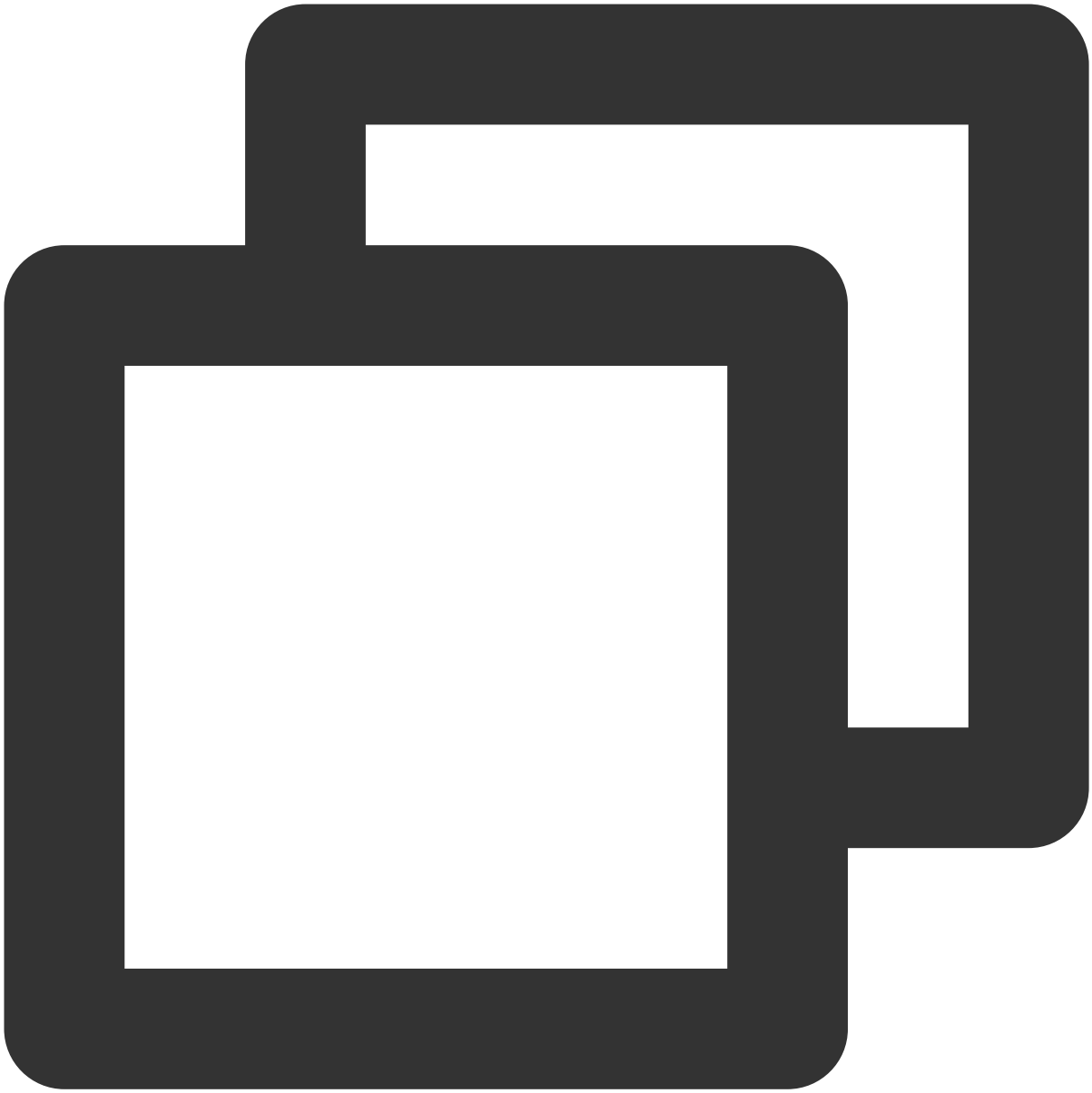
This API is used to accept a call. After receiving the `onCallReceived()` callback, you can call this API to accept the call.



```
void accept(TUICommonDefine.Callback callback);
```

reject

This API is used to reject a call. After receiving the `onCallReceived()` callback, you can call this API to reject the call.

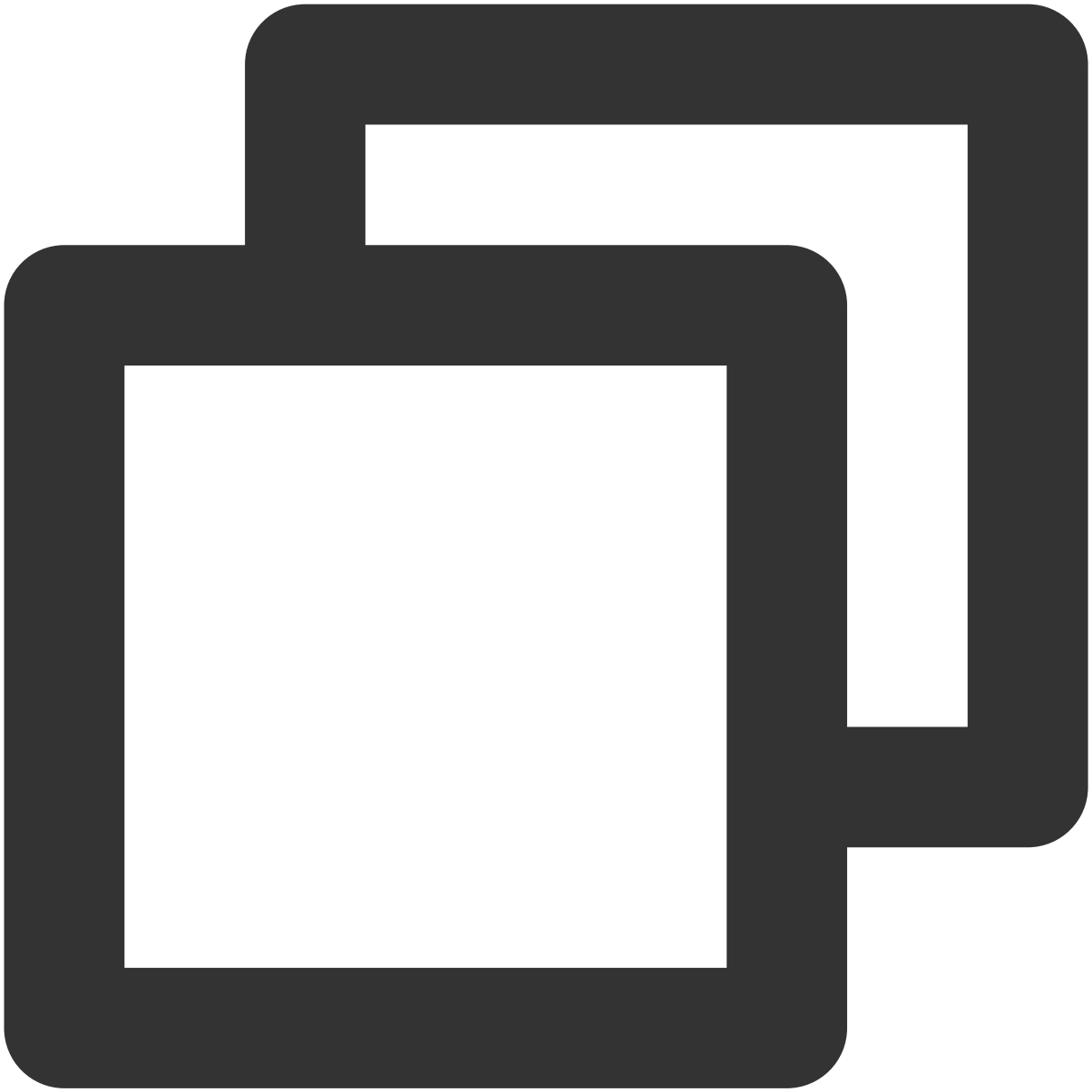


```
void reject(TUICommonDefine.Callback callback);
```

ignore

This API is used to ignore a call. After receiving the `onCallReceived()` , you can call this API to ignore the call. The caller will receive the `onUserLineBusy` callback.

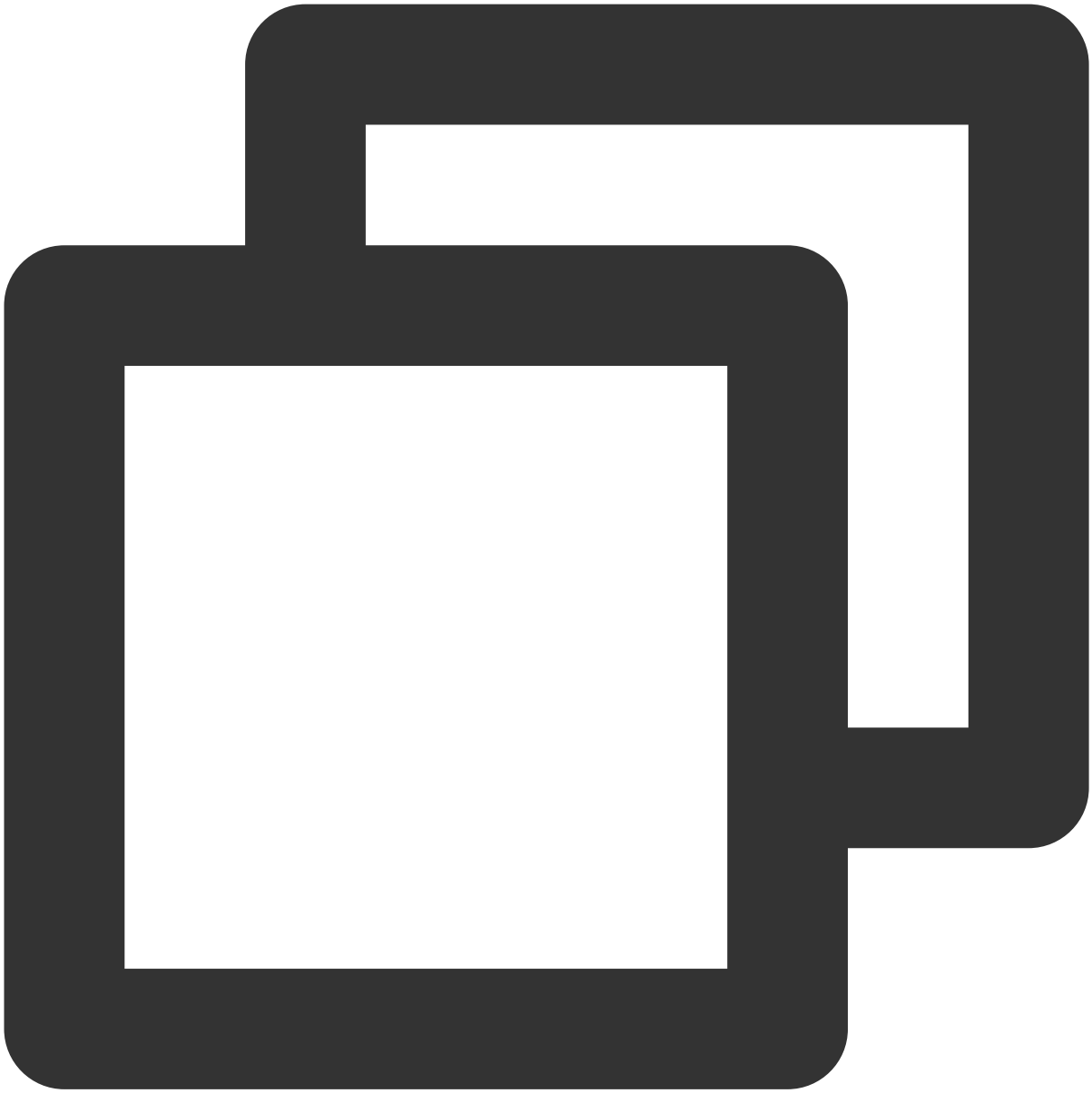
Note: If your project involves live streaming or conferencing, you can also use this API to implement the “in a meeting” or “on air” feature.



```
void ignore(TUICommonDefine.Callback callback);
```

hangup

This API is used to end a call.

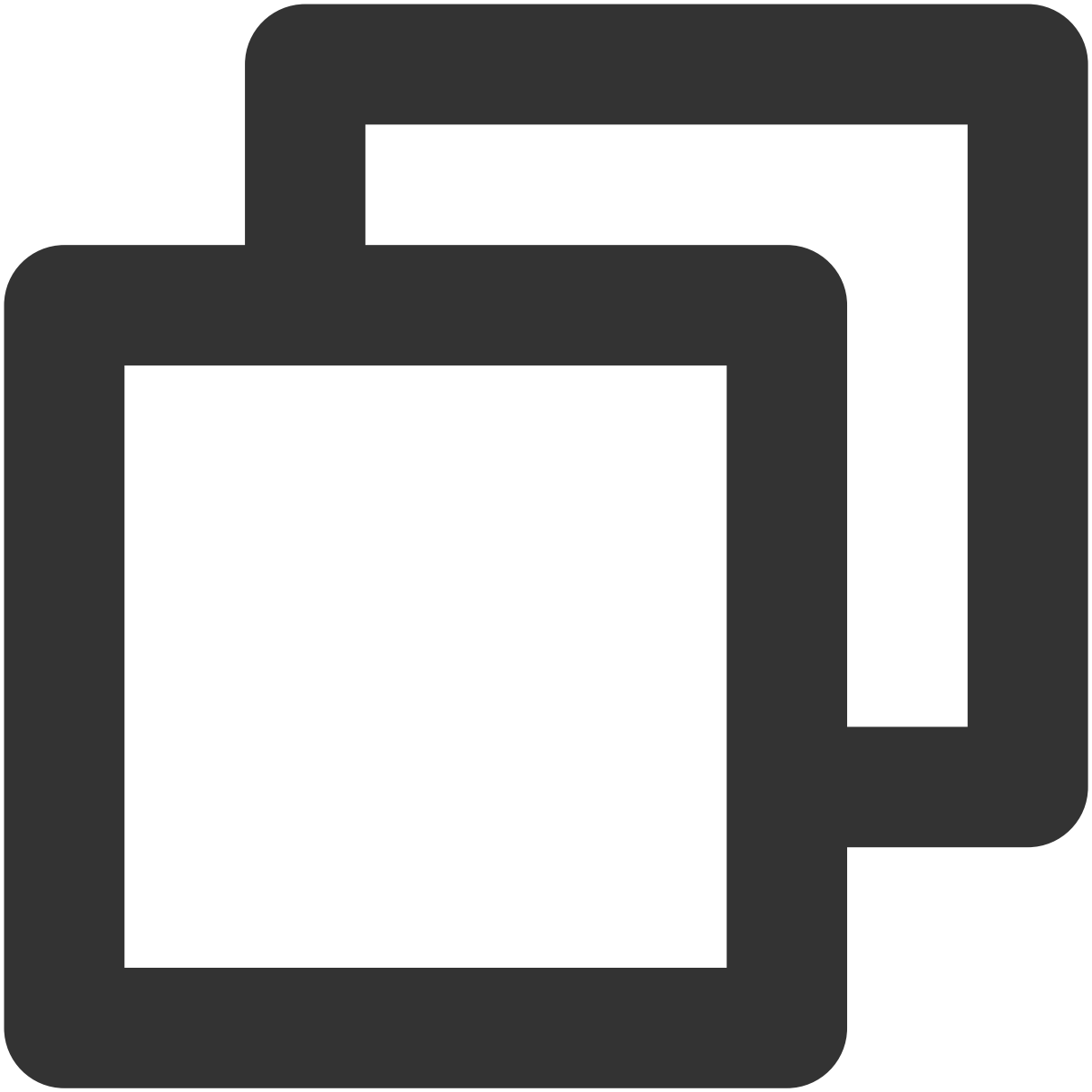


```
void hangup(TUICommonDefine.Callback callback);
```

inviteUser

This API is used to invite users to the current group call.

This API is called by a participant of a group call to invite new users.



```
void inviteUser(List<String> userIdList, TUICallDefine.CallParams params,
                TUICommonDefine.ValueCallback callback);
```

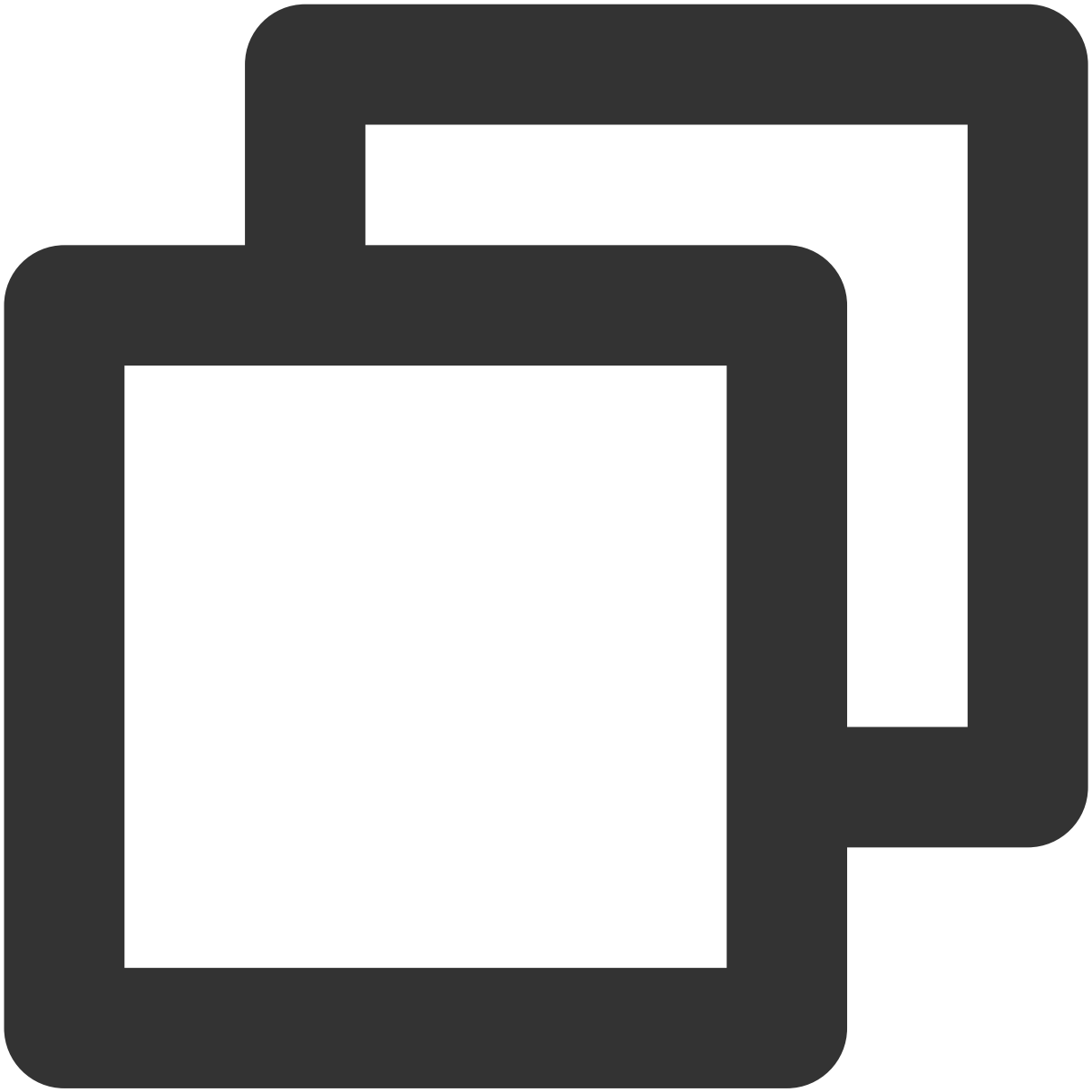
The parameters are described below:

Parameter	Type	Description
userIdList	List	The target user IDs.
params	TUICallDefine.CallParams	An additional parameter. such as roomID, call timeout, offline push info,etc

joinInGroupCall

This API is used to join a group call.

This API is called by a group member to join the group's call.



```
void joinInGroupCall(TUICommonDefine.RoomId roomId, String groupId,
                     TUICallDefine.MediaType callMediaType, TUICommonDefine.Callbac
```

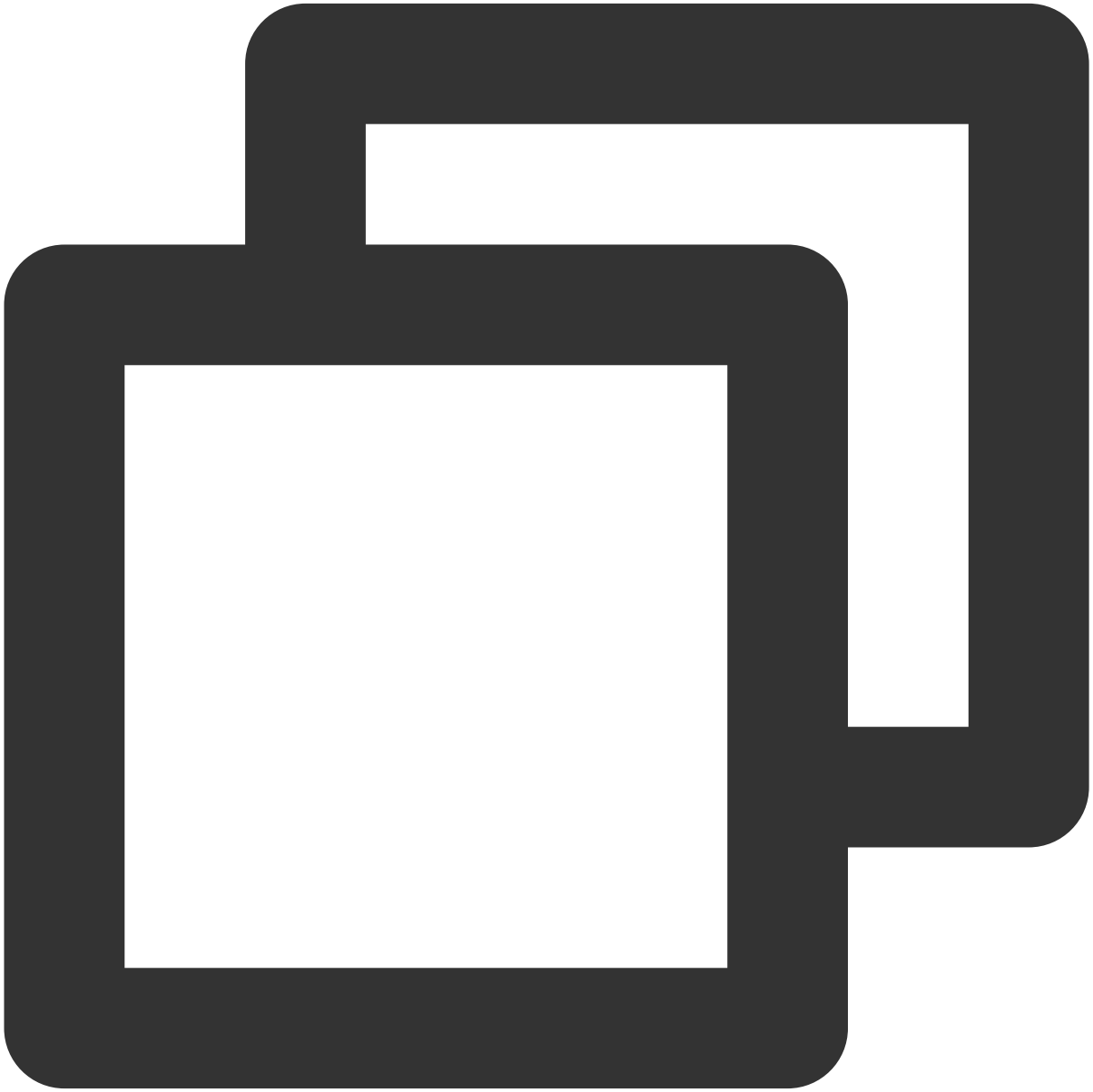
The parameters are described below:

Parameter	Type	Description

roomId	TUICommonDefine.RoomId	The room ID.
groupId	String	The group ID.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.

switchCallMediaType

This API is used to change the call type.



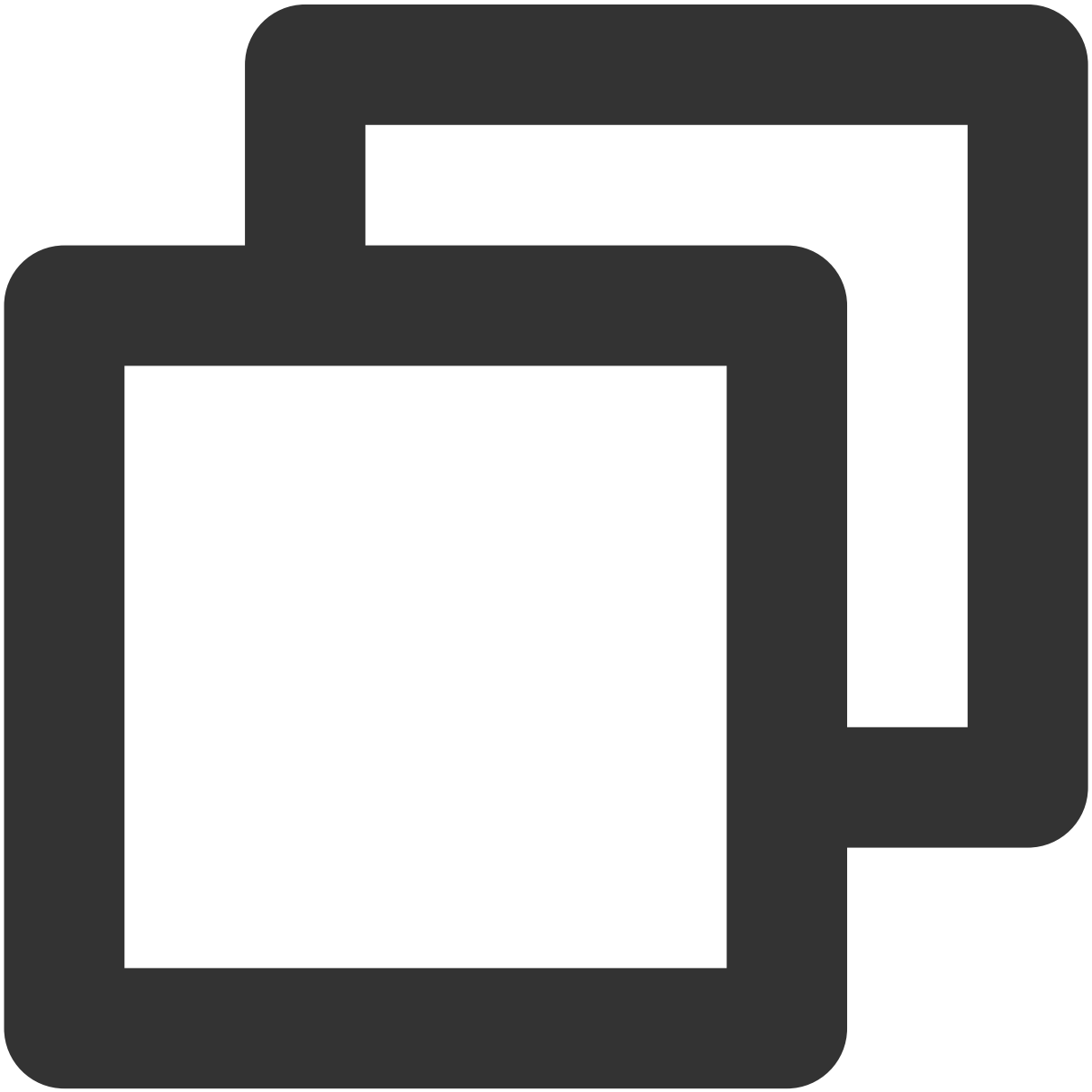
```
void switchCallMediaType(TUICallDefine.MediaType callMediaType);
```

The parameters are described below:

Parameter	Type	Description
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.

startRemoteView

This API is used to subscribe to the video stream of a remote user. For it to work, make sure you call it after `setRenderView`.



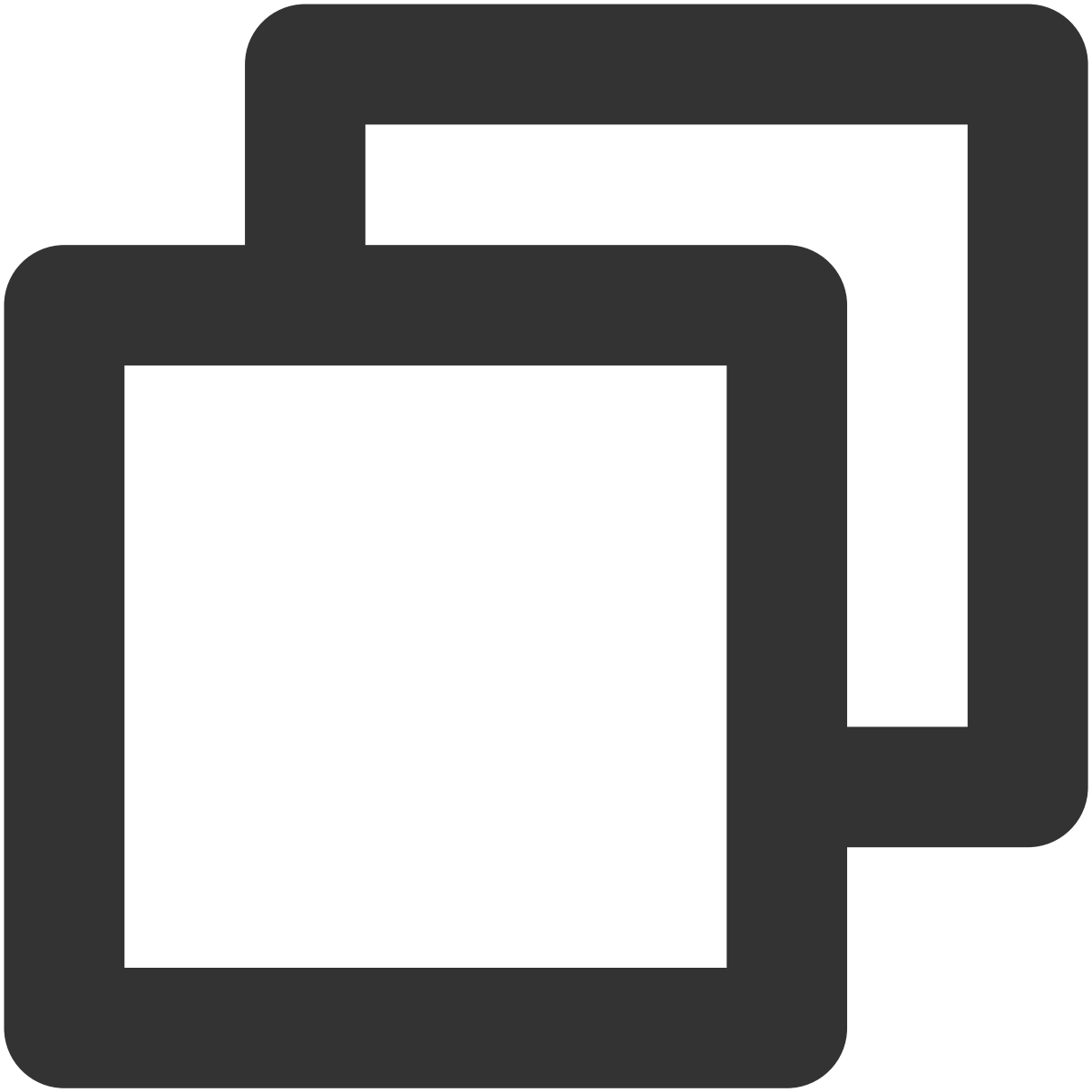
```
void startRemoteView(String userId, TUIVideoView videoView, TUICommonDefine.PlayCal
```

The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
videoView	TUIVideoView	The view to be rendered.

stopRemoteView

This API is used to unsubscribe from the video stream of a remote user.



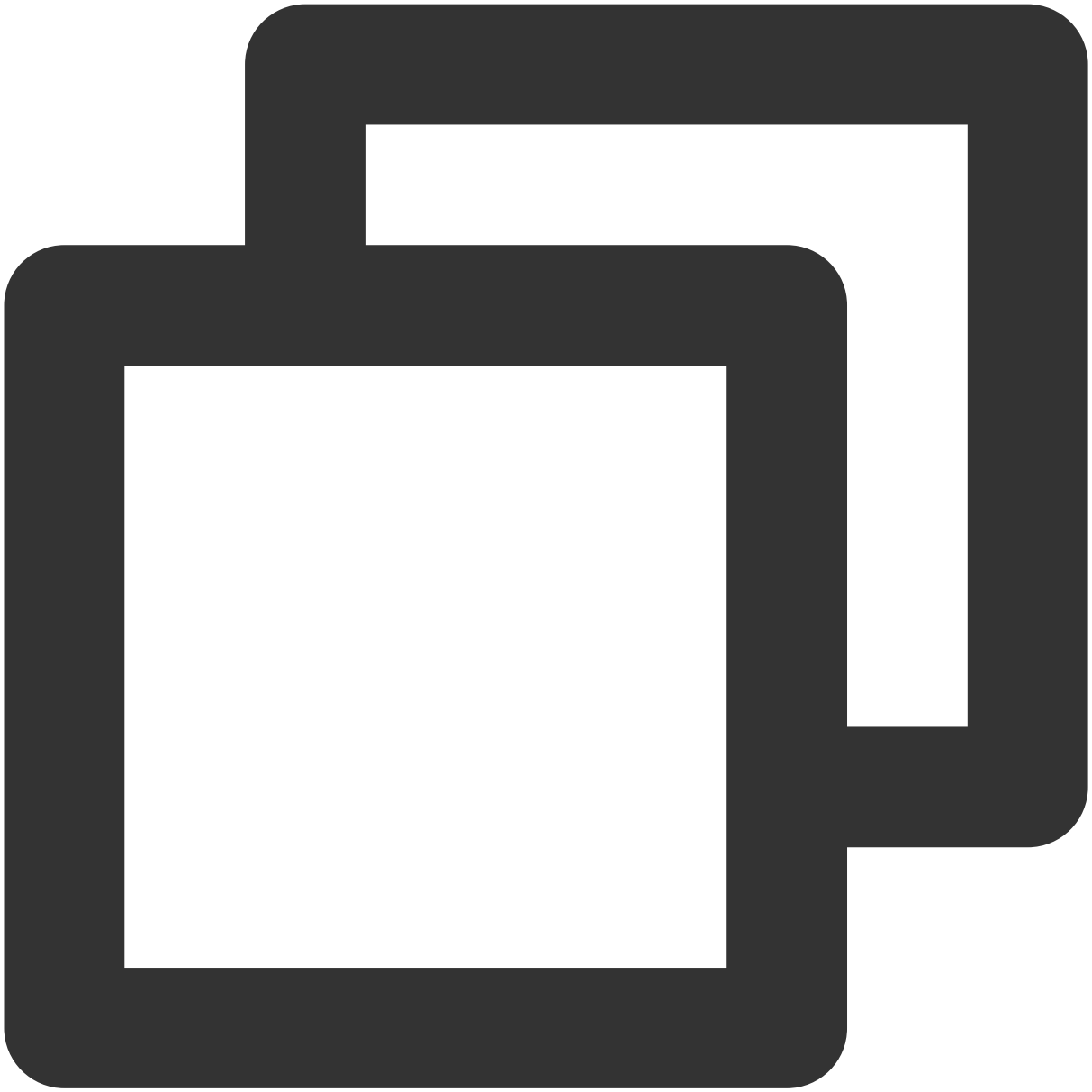
```
void stopRemoteView(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.

openCamera

This API is used to turn the camera on.



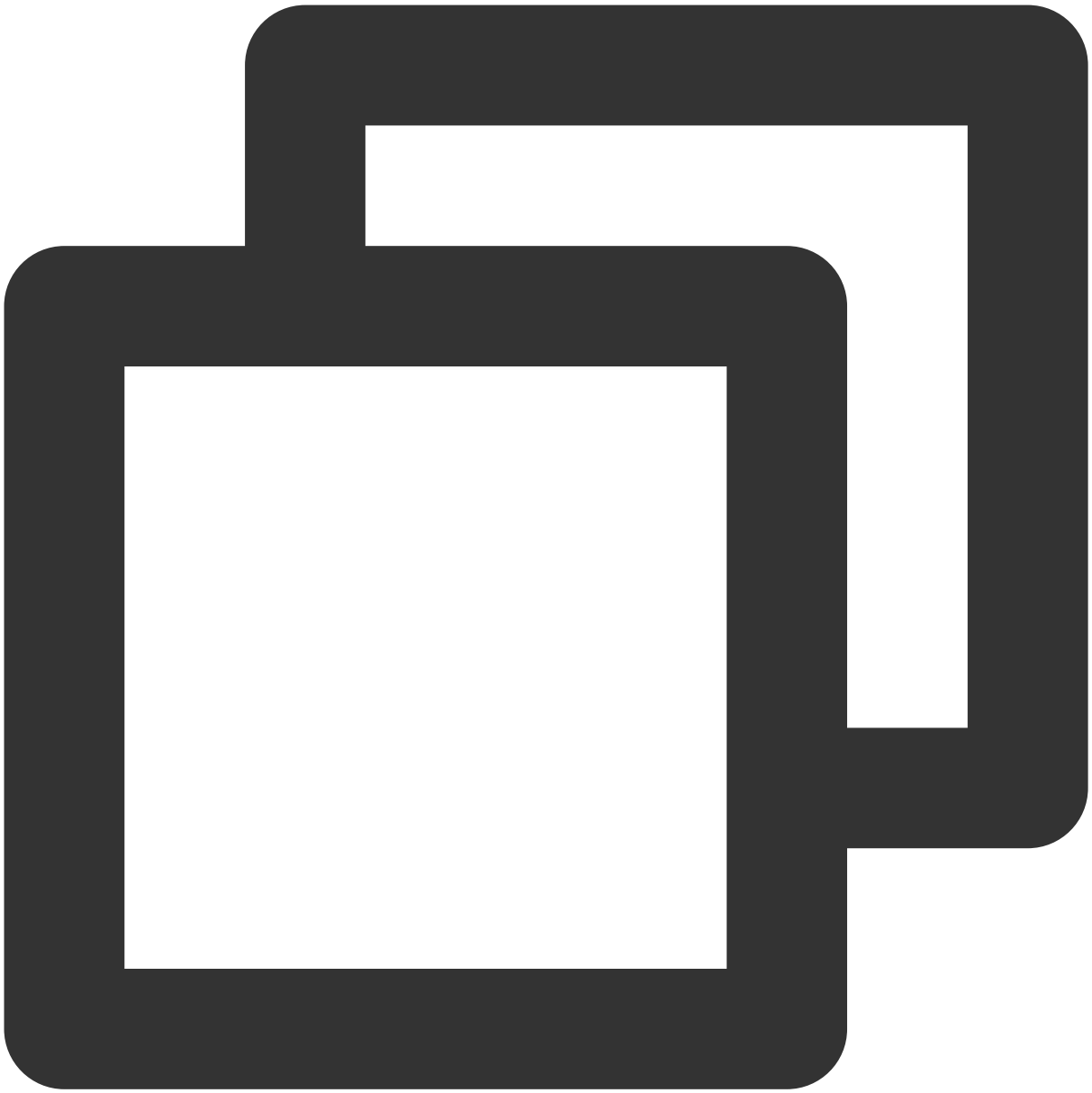
```
void openCamera(TUICommonDefine.Camera camera, TUIVideoView videoView, TUICommonDef
```

The parameters are described below:

Parameter	Type	Description
camera	TUICommonDefine.Camera	The front or rear camera.
videoView	TUIVideoView	The view to be rendered.

closeCamera

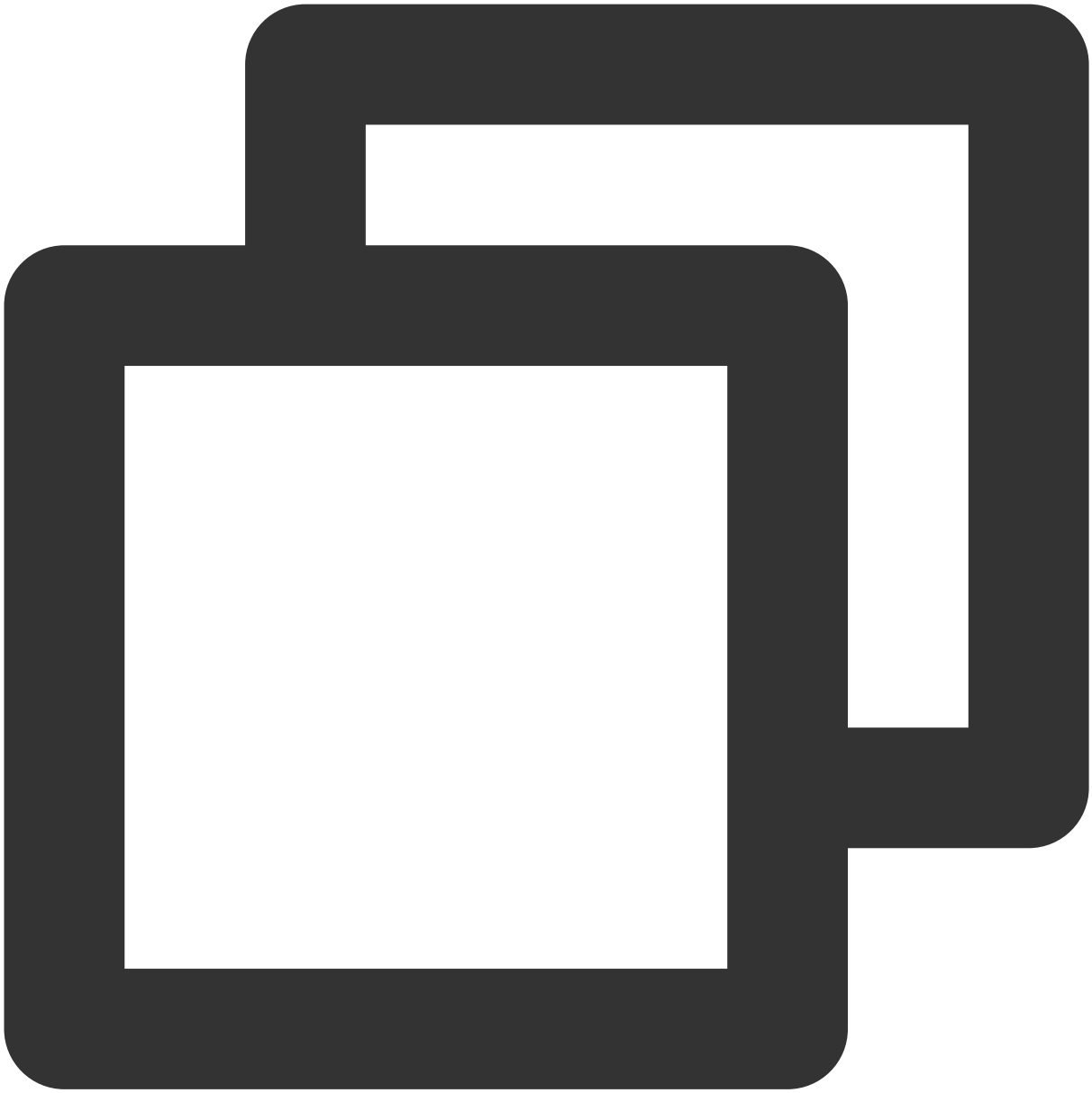
This API is used to turn the camera off.



```
void closeCamera();
```

switchCamera

This API is used to switch between the front and rear cameras.



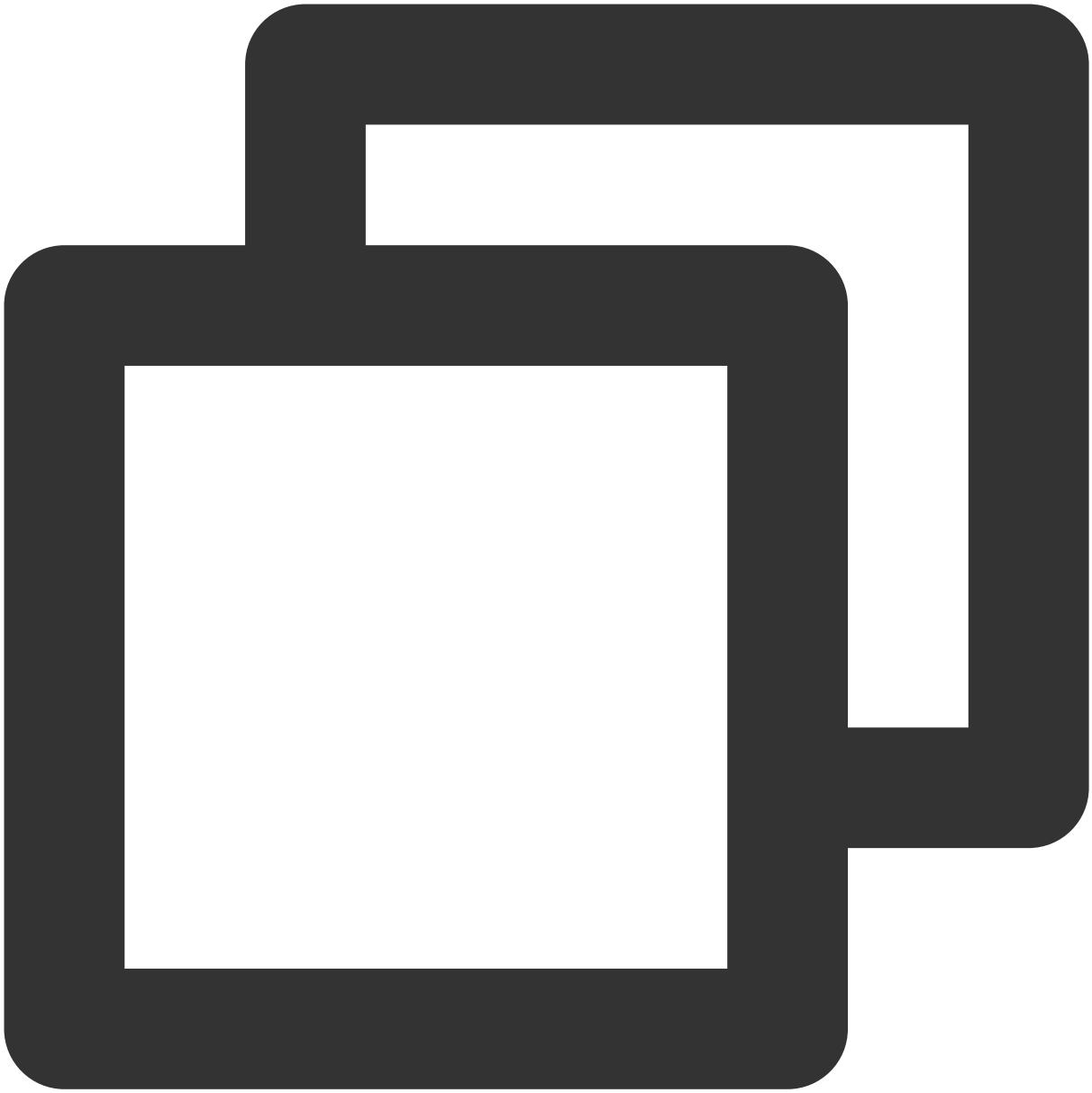
```
void switchCamera(TUICommonDefine.Camera camera);
```

The parameters are described below:

Parameter	Type	Description
camera	TUICommonDefine.Camera	The front or rear camera.

openMicrophone

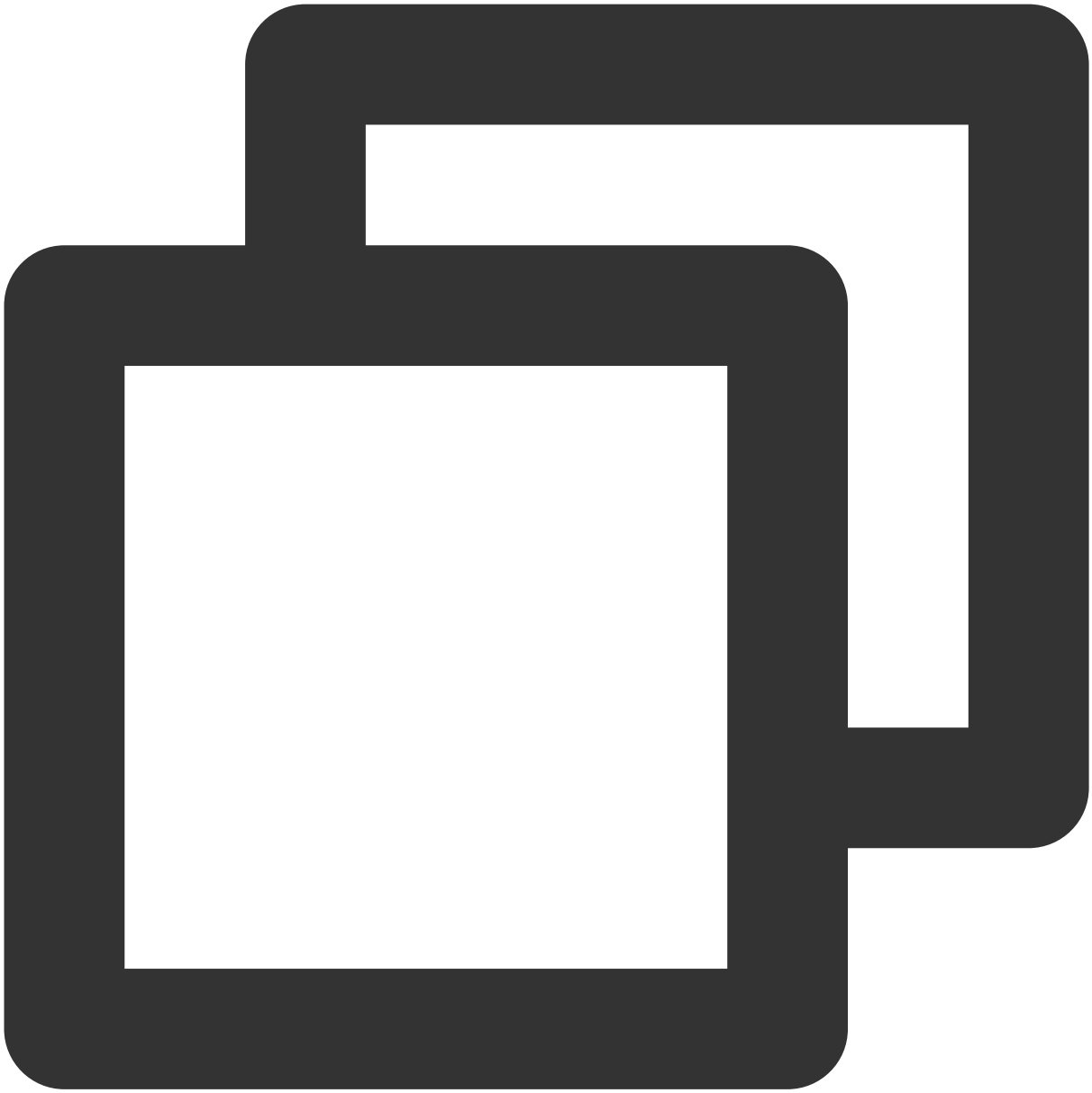
This API is used to turn the mic on.



```
void openMicrophone(TUICommonDefine.Callback callback);
```

closeMicrophone

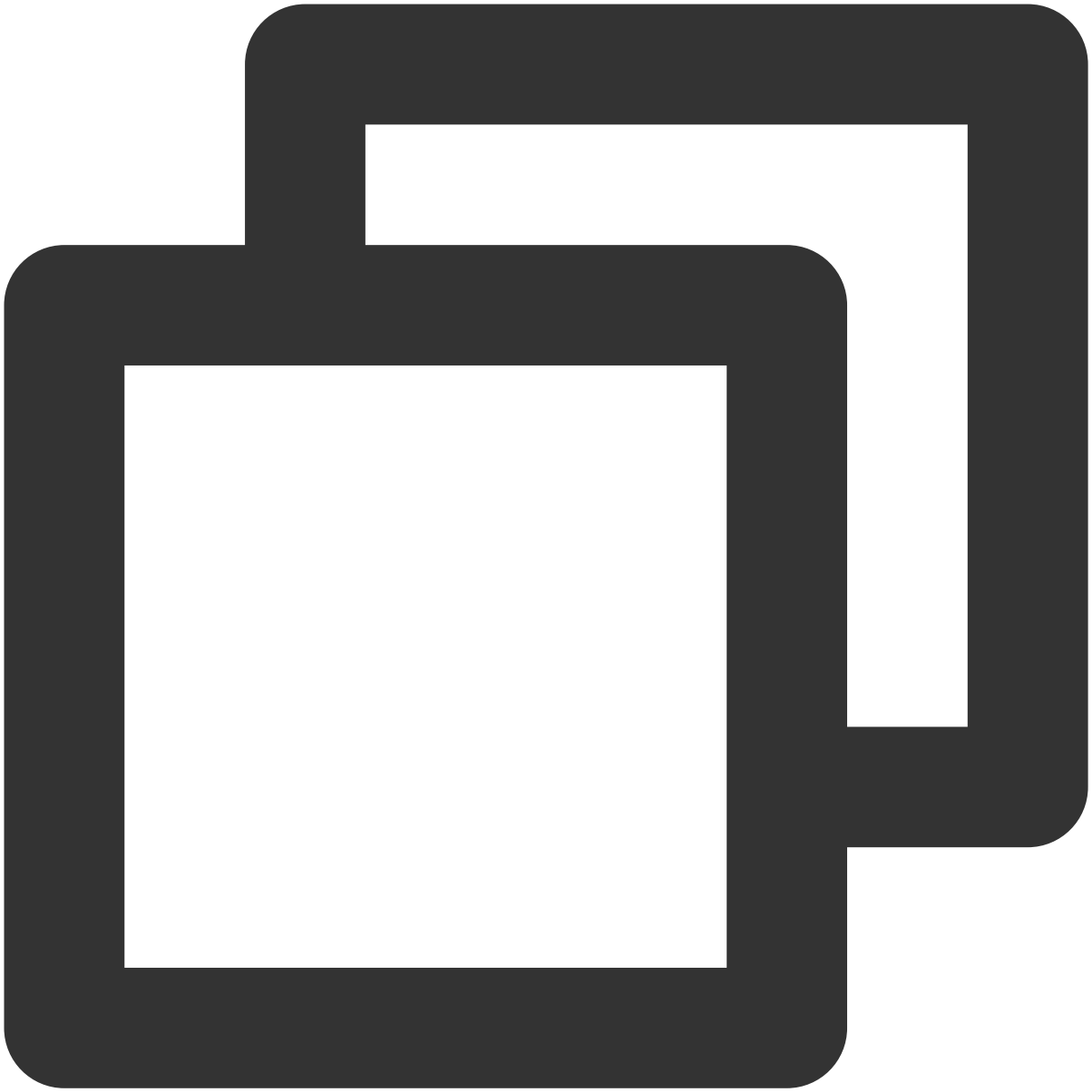
This API is used to turn the mic off.



```
void closeMicrophone();
```

selectAudioPlaybackDevice

This API is used to select the audio playback device (receiver or speaker). In call scenarios, you can use this API to turn on/off hands-free mode.



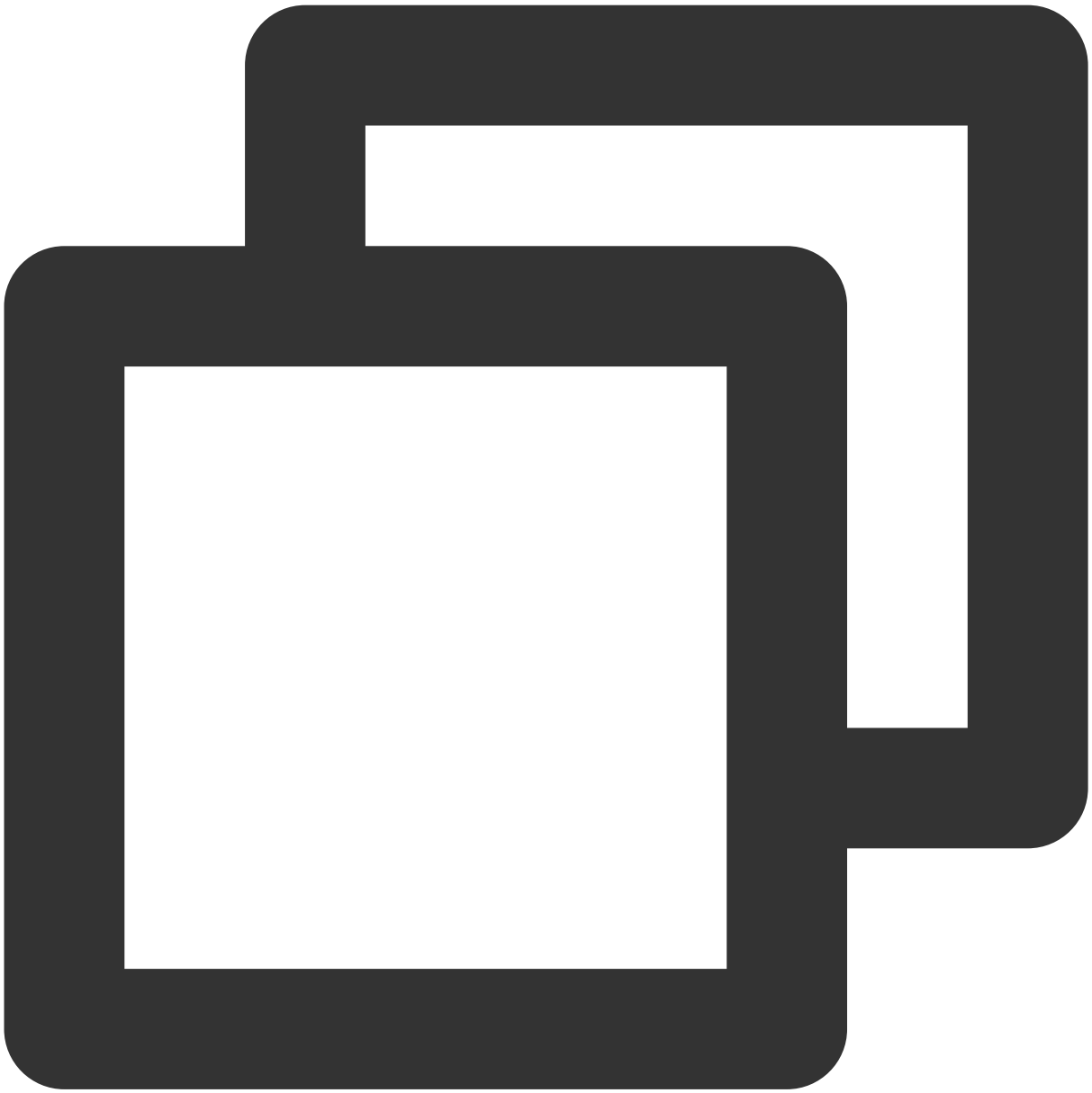
```
void selectAudioPlaybackDevice(TUICommonDefine.AudioPlaybackDevice device);
```

The parameters are described below:

Parameter	Type	Description
device	TUICommonDefine.AudioPlaybackDevice	The speaker or receiver.

setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.



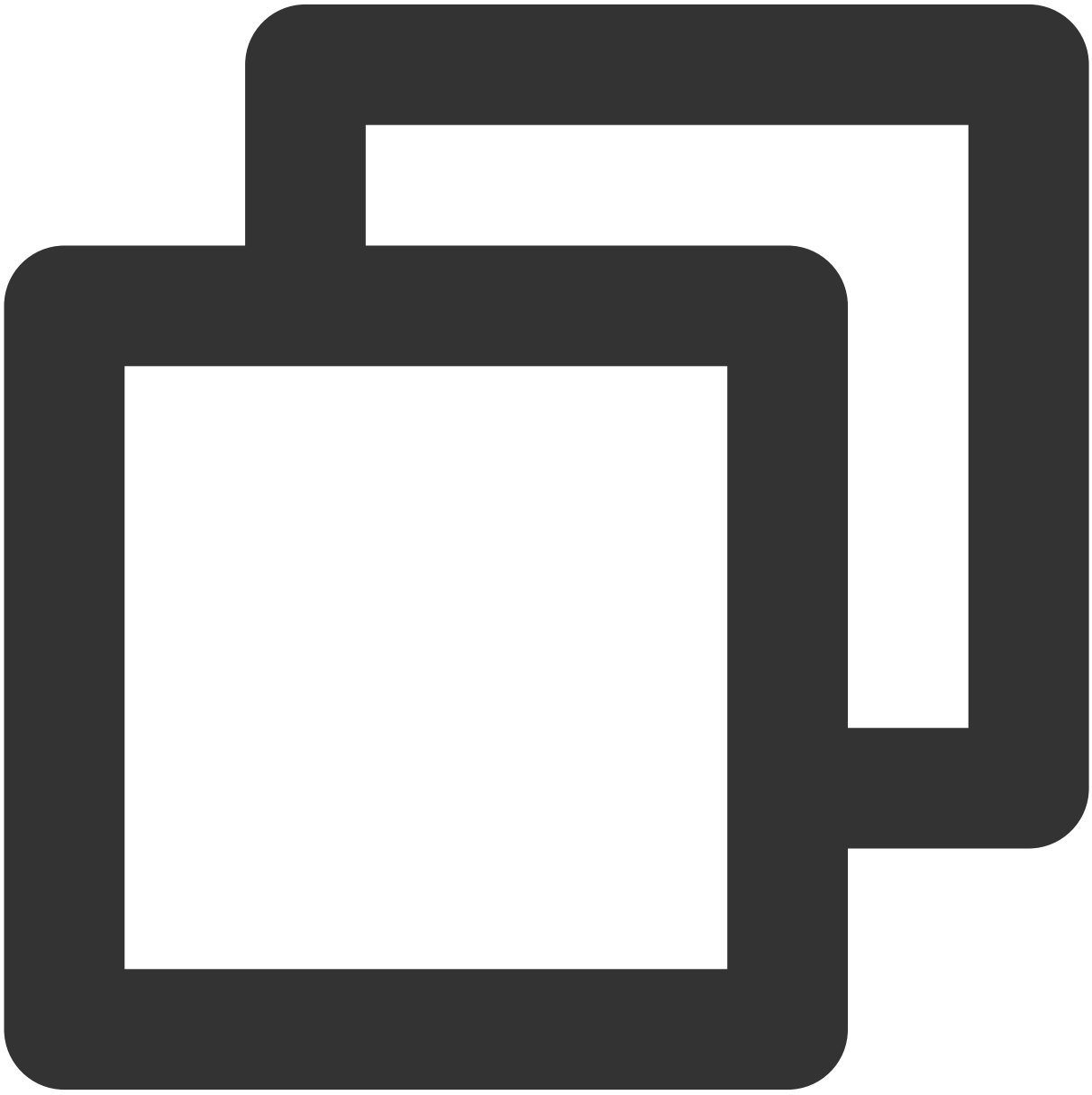
```
void setSelfInfo(String nickname, String avatar, TUICommonDefine.Callback callback)
```

The parameters are described below:

Parameter	Type	Description
nickname	String	The alias.
avatar	String	The URL of the profile photo.

enableMultiDeviceAbility

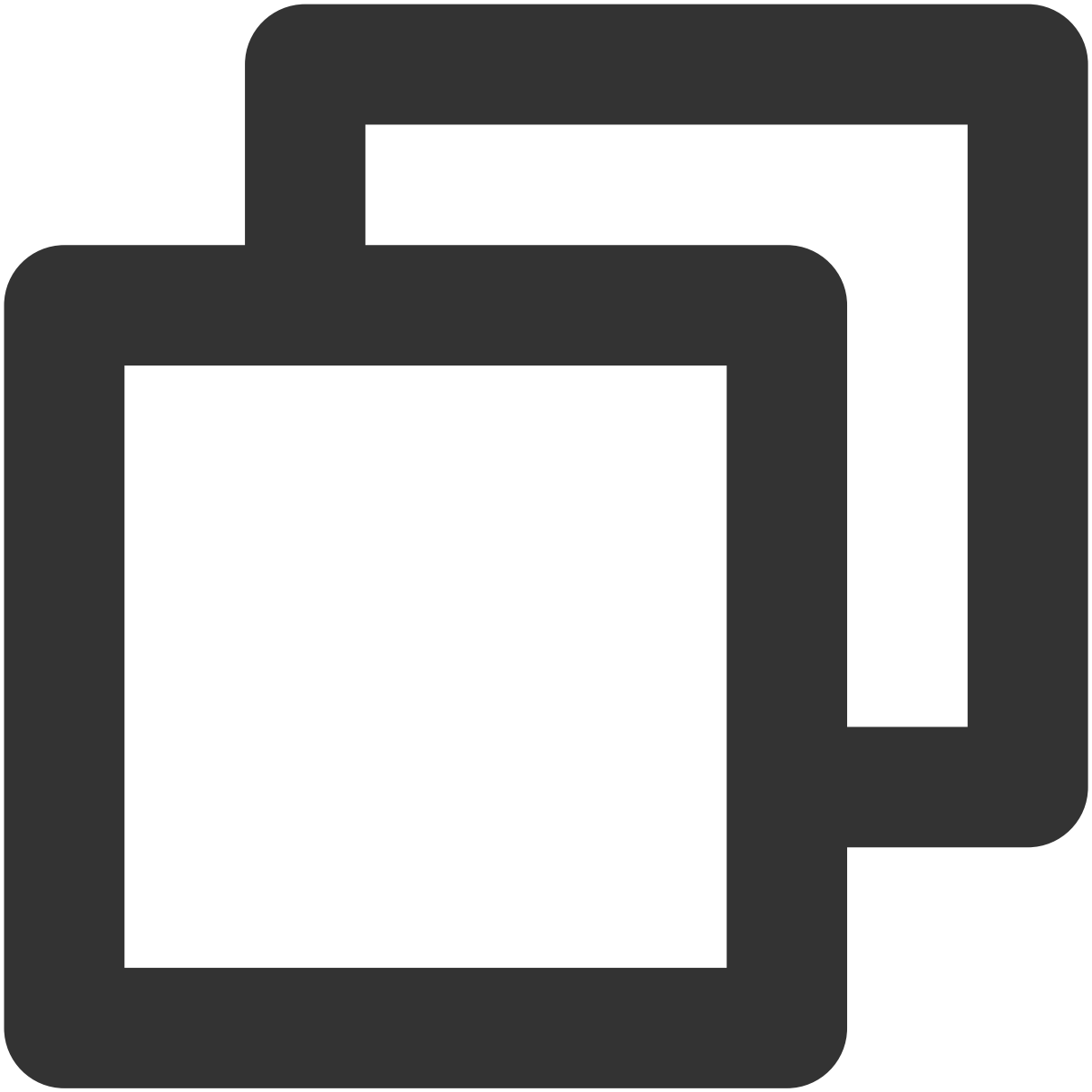
This API is used to set whether to enable multi-device login for `TUICallEngine` (supported by the Group Call package).



```
void enableMultiDeviceAbility(boolean enable, TUICommonDefine.Callback callback);
```

setVideoRenderParams

Set the rendering mode of video image.



```
void setVideoRenderParams(String userId, TUICommonDefine.VideoRenderParams params,
```

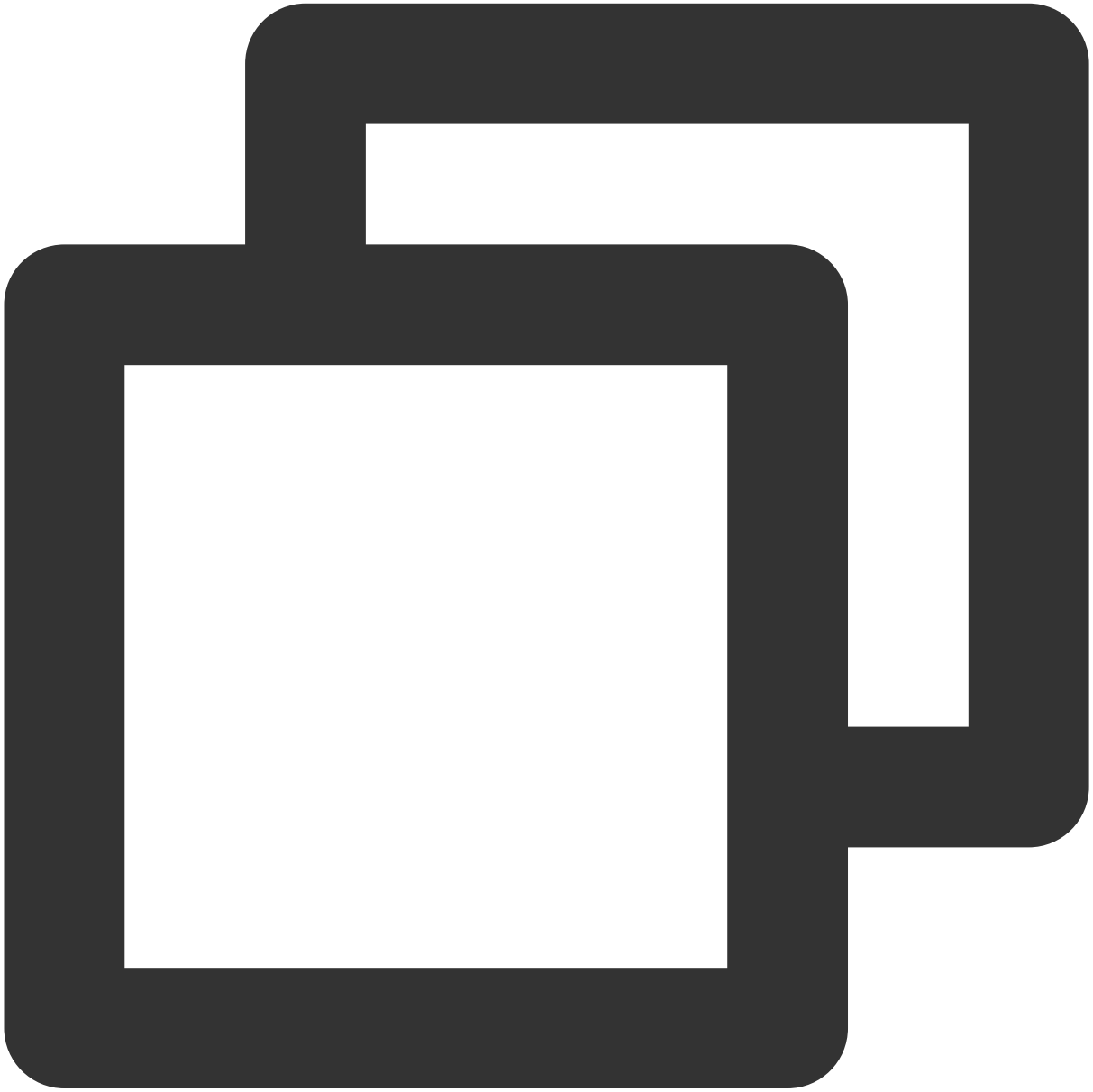
The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
params	TUICommonDefine.VideoRenderParams	Video render parameters.

setVideoEncoderParams

Set the encoding parameters of video encoder.

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.



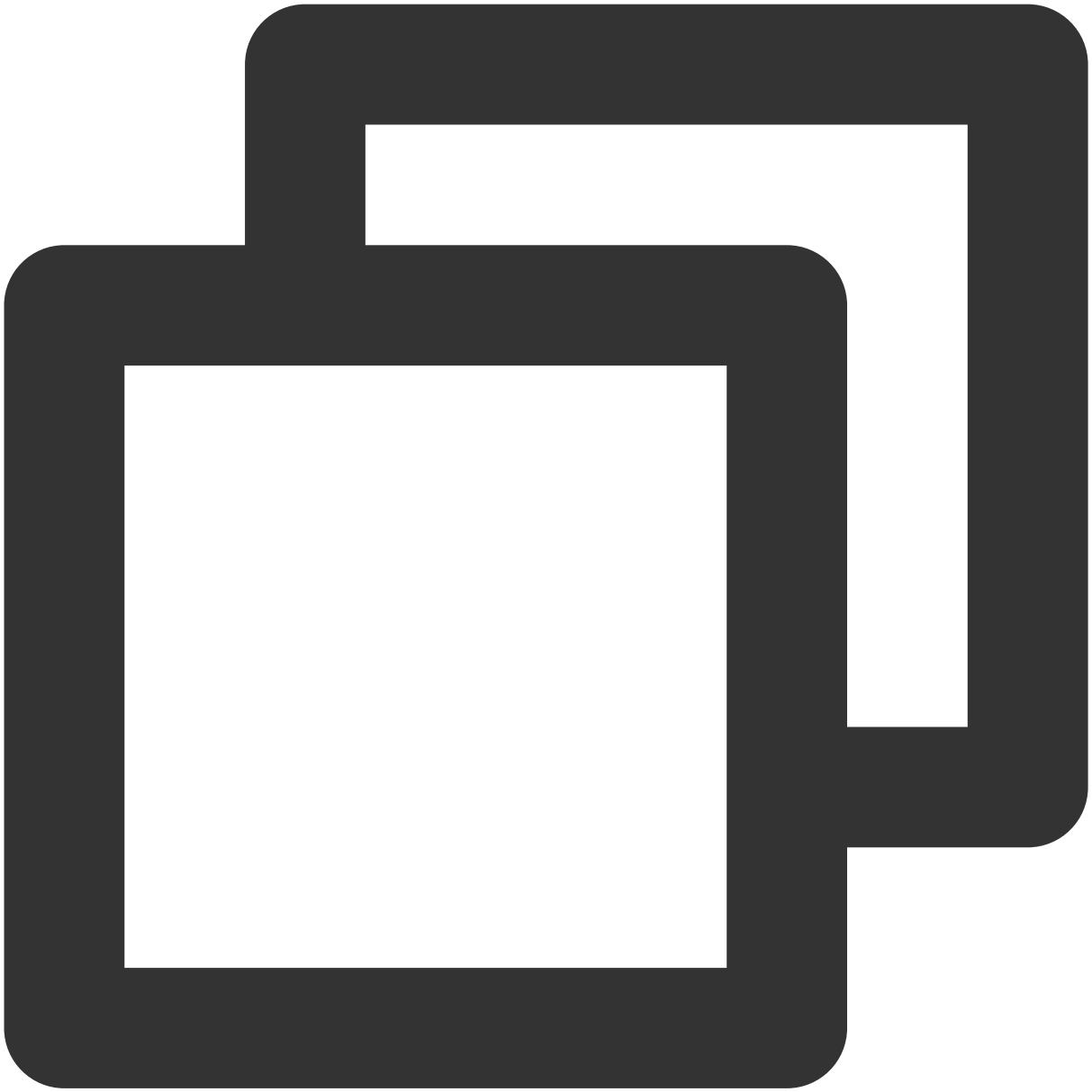
```
void setVideoEncoderParams(TUICommonDefine.VideoEncoderParams params, TUICommonDefi
```

The parameters are described below:

Parameter	Type	Description
params	TUICommonDefine.VideoEncoderParams	Video encoding parameters

getTRTCCloudInstance

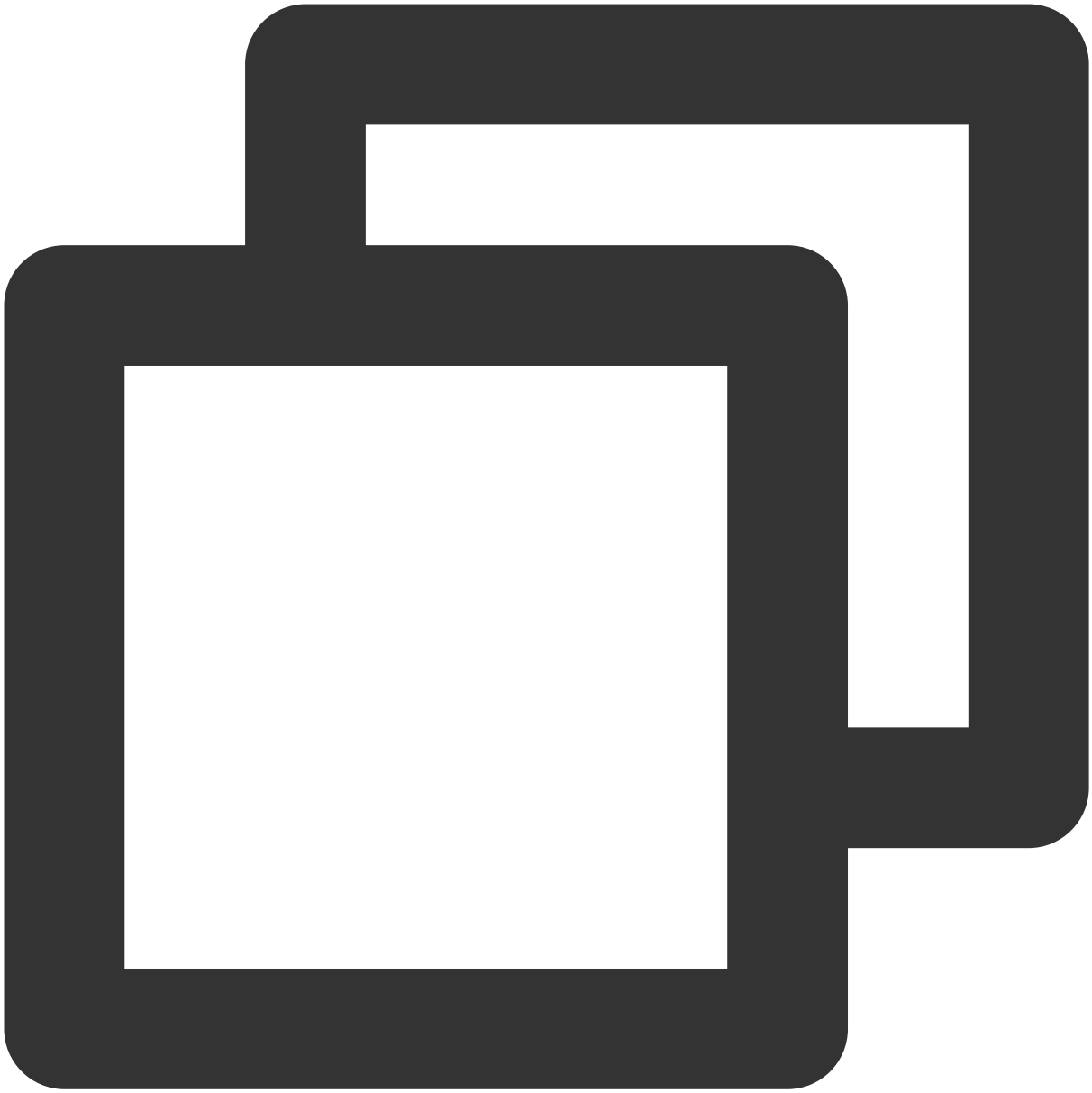
Advanced features.



```
TRTCCloud getTRTCCloudInstance();
```

setBeautyLevel

Set beauty level, support turning off default beauty.



```
void setBeautyLevel(float level, TUICommonDefine.Callback callback);
```

The parameters are described below:

Parameter	Type	Description
level	float	Beauty level, range: 0 - 9； 0 means turning off the effect, 9 means the most obvious effect.

TUICallObserver

Last updated：2024-01-17 11:58:53

TUICallObserver APIs

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

Overview

API	Description
onError	A call occurred during the call.
onCallReceived	A call invitation was received.
onCallCancelled	The call was canceled.
onCallBegin	The call was connected.
onCallEnd	The call ended.
onCallMediaTypeChanged	The call type changed.
onUserReject	A user declined the call.
onUserNoResponse	A user didn't respond.
onUserLineBusy	A user was busy.
onUserJoin	A user joined the call.
onUserLeave	A user left the call.
onUserVideoAvailable	Whether a user had a video stream.
onUserAudioAvailable	Whether a user had an audio stream.
onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.
onKickedOffline	The current user was kicked offline.

onUserSigExpired	The user sig is expired.

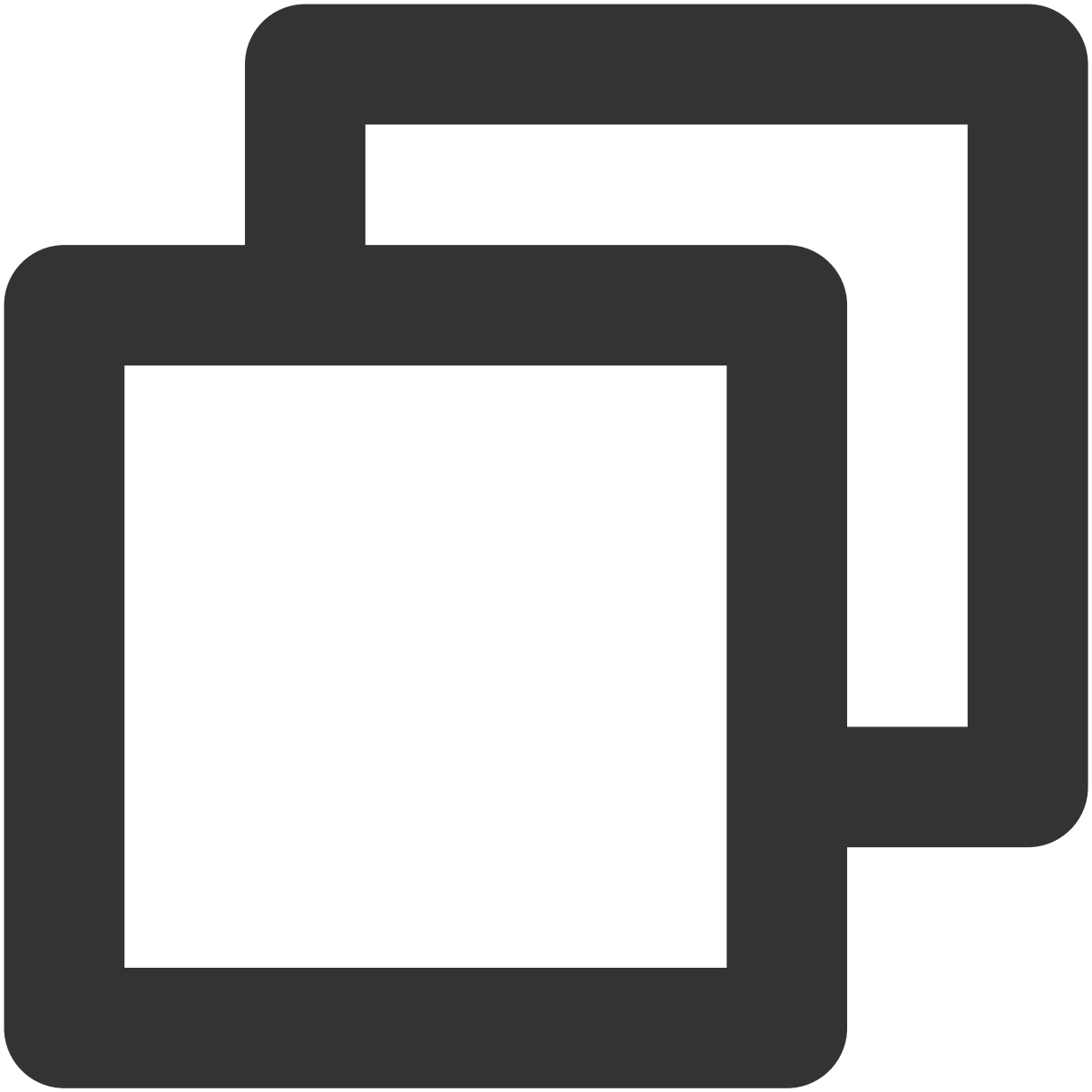
Details

onError

An error occurred.

Note

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



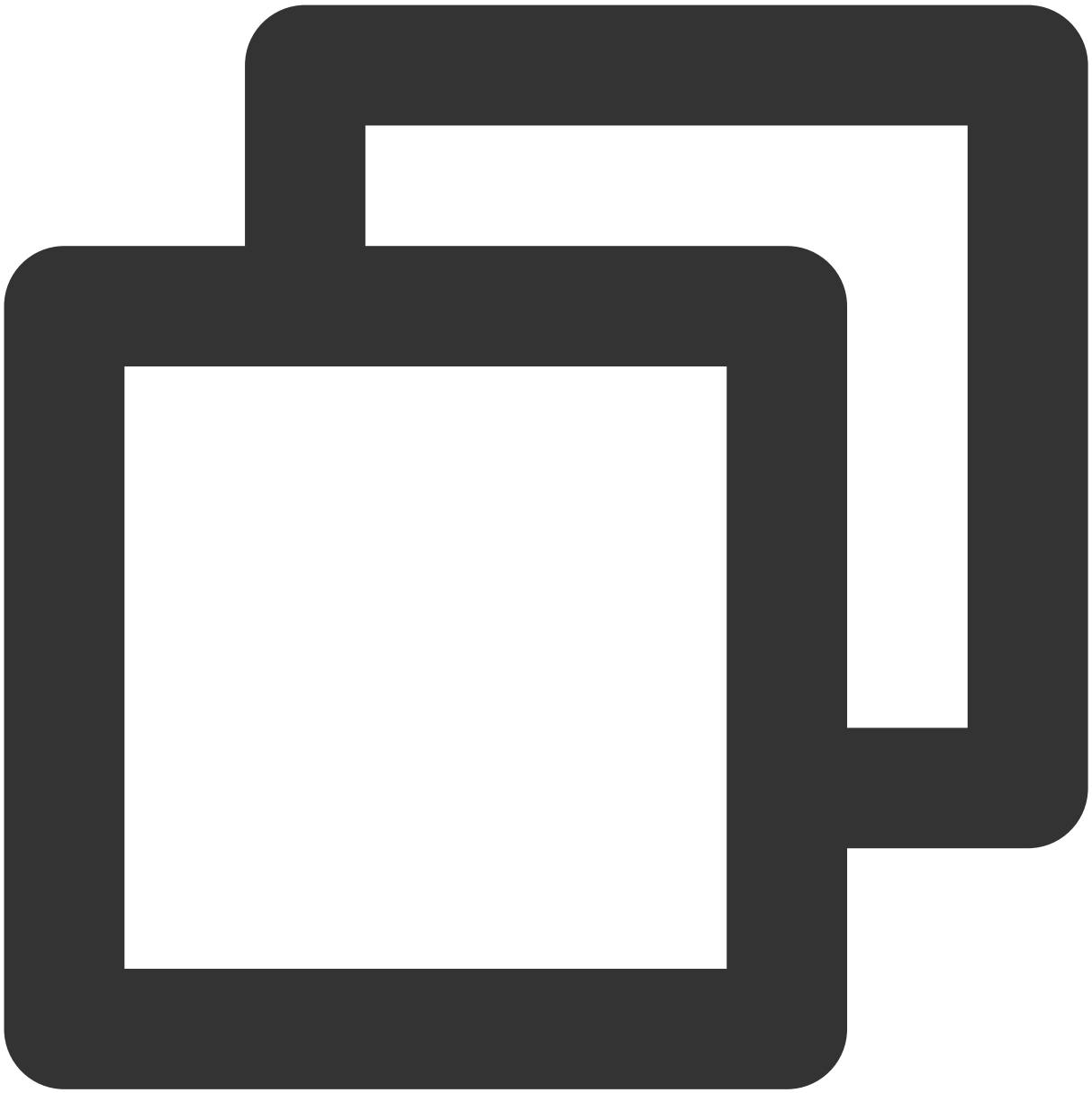
```
void onError(int code, String msg);
```

The parameters are described below:

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

onCallReceived

A call invitation was received. This callback is received by an invitee. You can listen for this event to determine whether to display the incoming call view.



```
void onCallReceived(String callerId, List<String> calleeIdList, String groupId,
                    TUICallDefine.MediaType callMediaType, String userData);
```

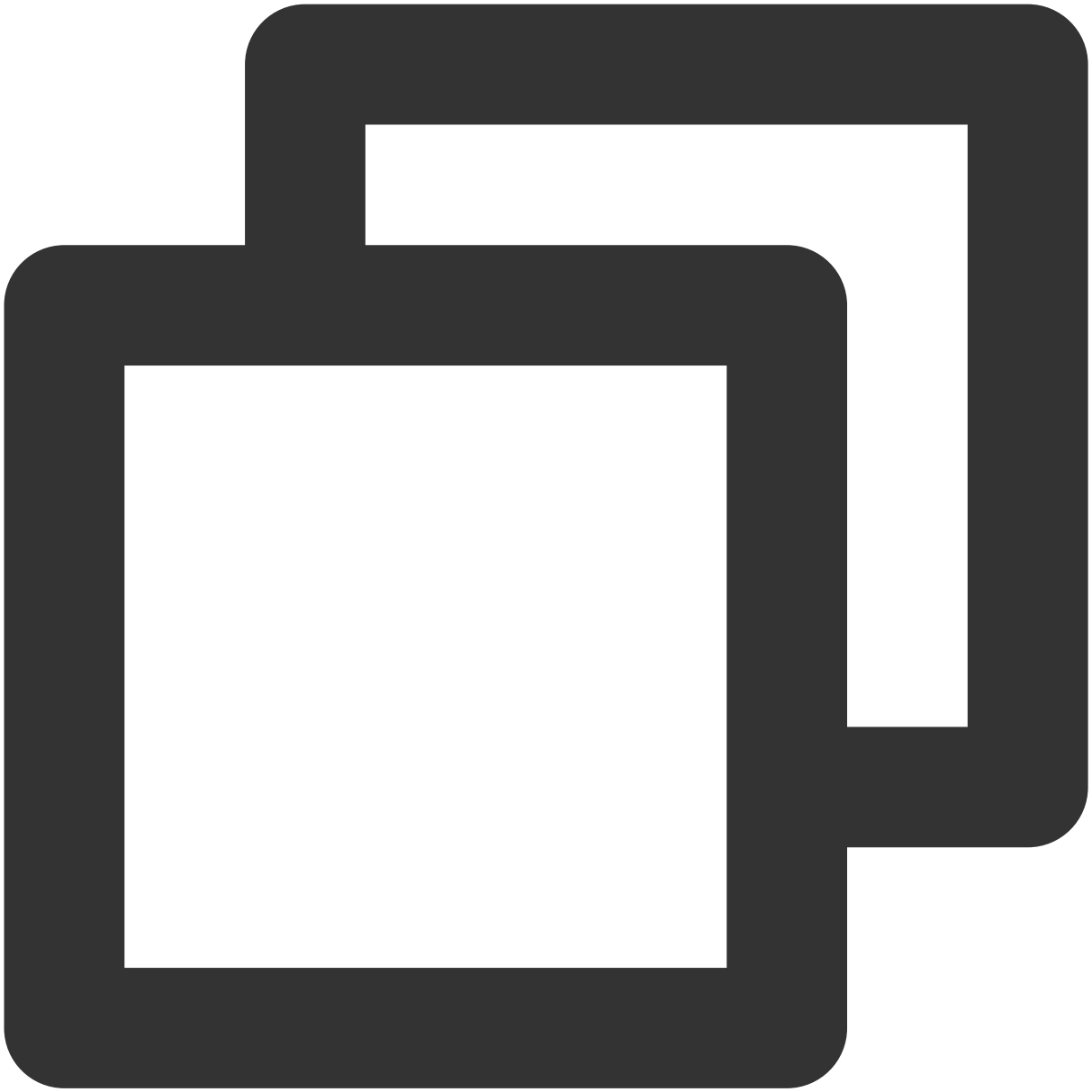
The parameters are described below:

Parameter	Type	Description
callerId	String	The user ID of the inviter.

calleeIdList	List	The invitee list.
groupId	String	The group ID.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.
userData	String	User-added extended fields., Please refer to: TUICallDefine.CallParams

onCallCancelled

The call was canceled by the inviter or timed out. This callback is received by an invitee. You can listen for this event to determine whether to show a missed call message.
This indicates that the call was canceled by the caller, timed out by the callee, rejected by the callee, or the callee was busy. There are multiple scenarios involved. You can listen to this event to achieve UI logic such as missed calls and resetting UI status.
Call cancellation by the caller: The caller receives the callback (userId is himself); the callee receives the callback (userId is the ID of the caller)
Callee timeout: the caller will simultaneously receive the onUserNoResponse and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)
Callee rejection: The caller will simultaneously receive the onUserReject and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)
Callee busy: The caller will simultaneously receive the onUserLineBusy and onCallCancelled callbacks (userId is his own ID);
Abnormal interruption: The callee failed to receive the call , he receives this callback (userId is his own ID).



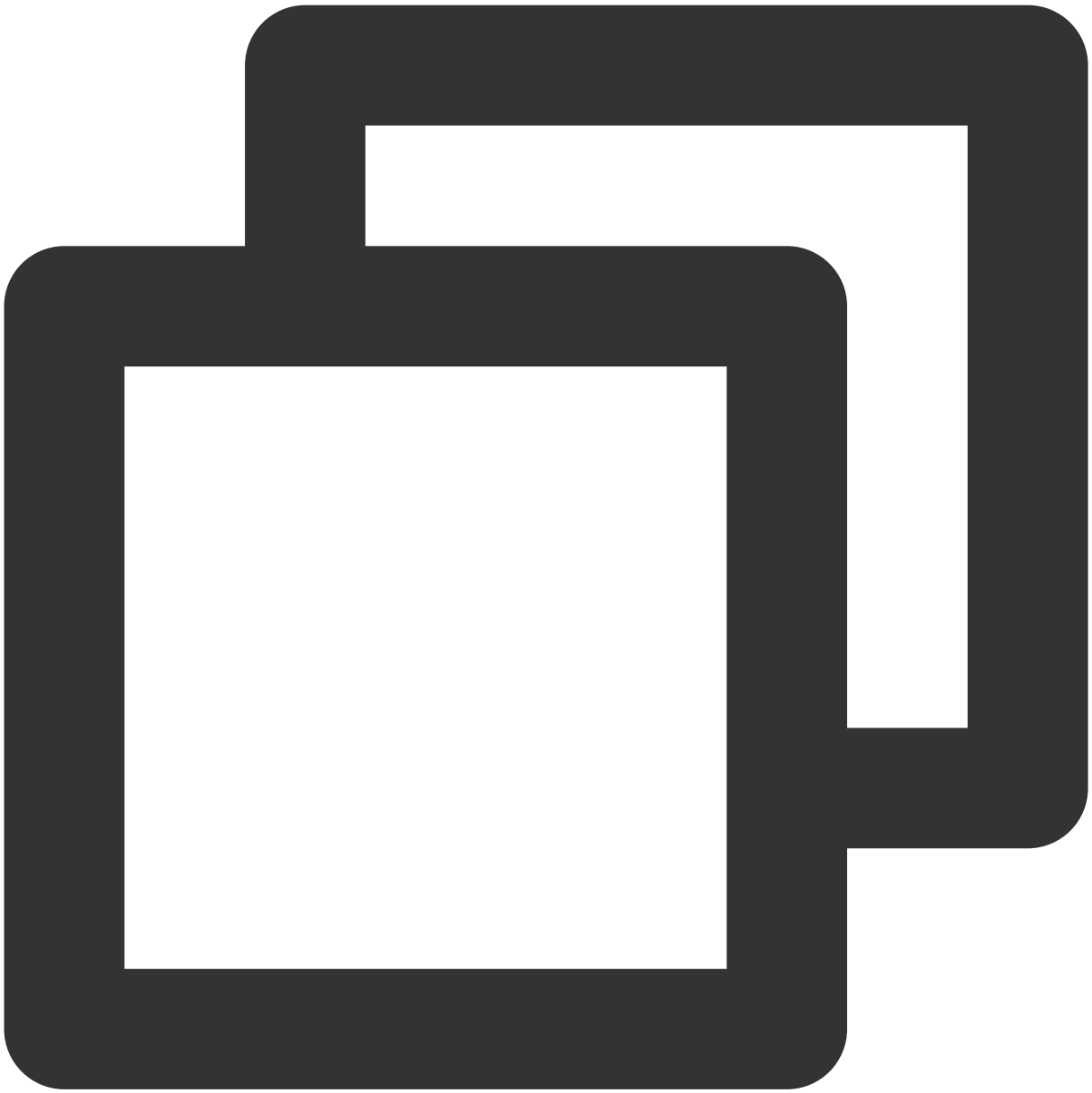
```
void onCallCancelled(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the inviter.

onCallBegin

The call was connected. This callback is received by both the inviter and invitees. You can listen for this event to determine whether to start on-cloud recording, content moderation, or other tasks.



```
void onCallBegin(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType callMediaTy
```

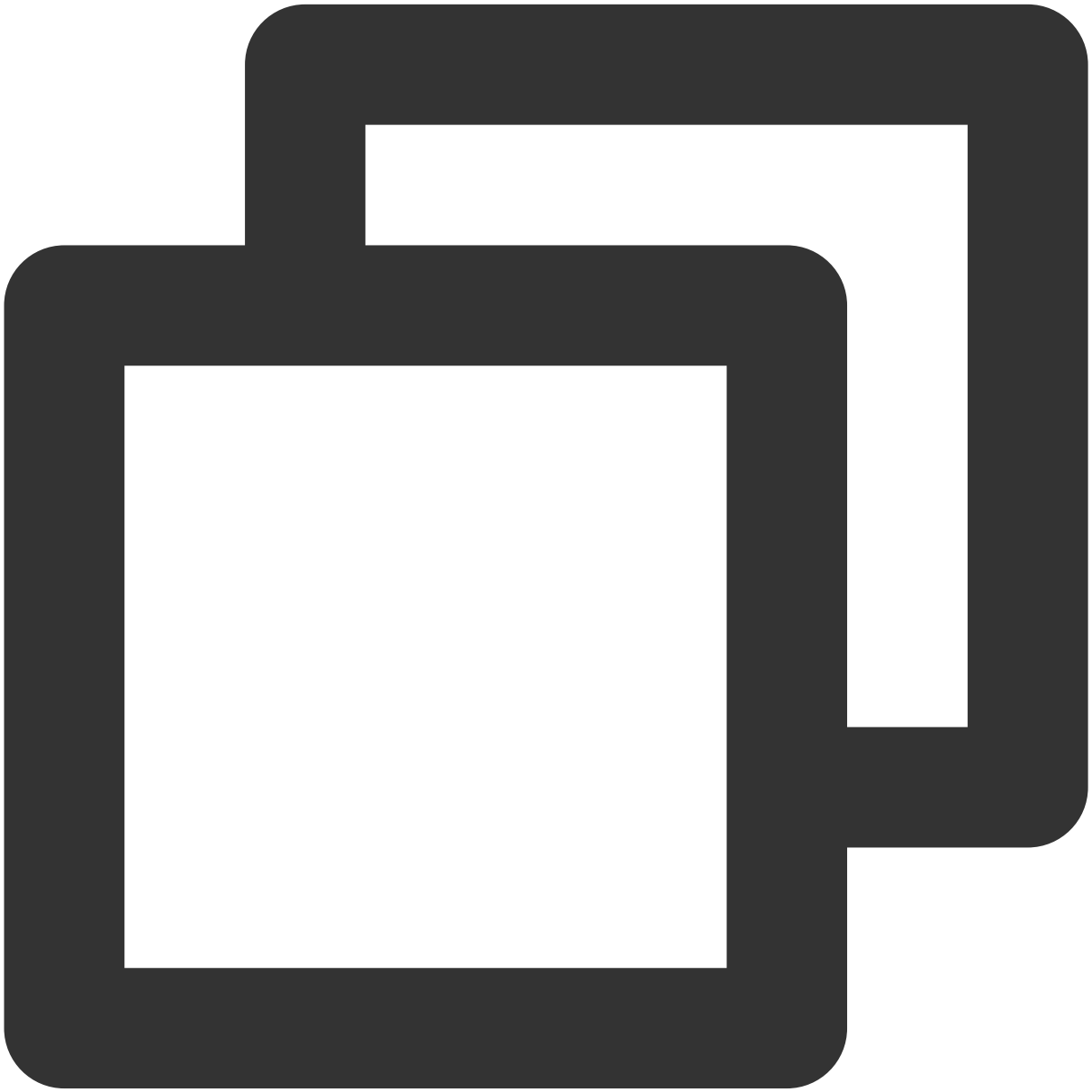
The parameters are described below:

Parameter	Type	Description
roomId	TUICommonDefine.RoomId	The room ID.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.

callRole	TUICallDefine.Role	The role, which can be caller or callee.

onCallEnd

The call ended. This callback is received by both the inviter and invitees. You can listen for this event to determine when to display call information such as call duration and call type, or stop on-cloud recording.



```
void onCallEnd(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType callMediaType
```

The parameters are described below:

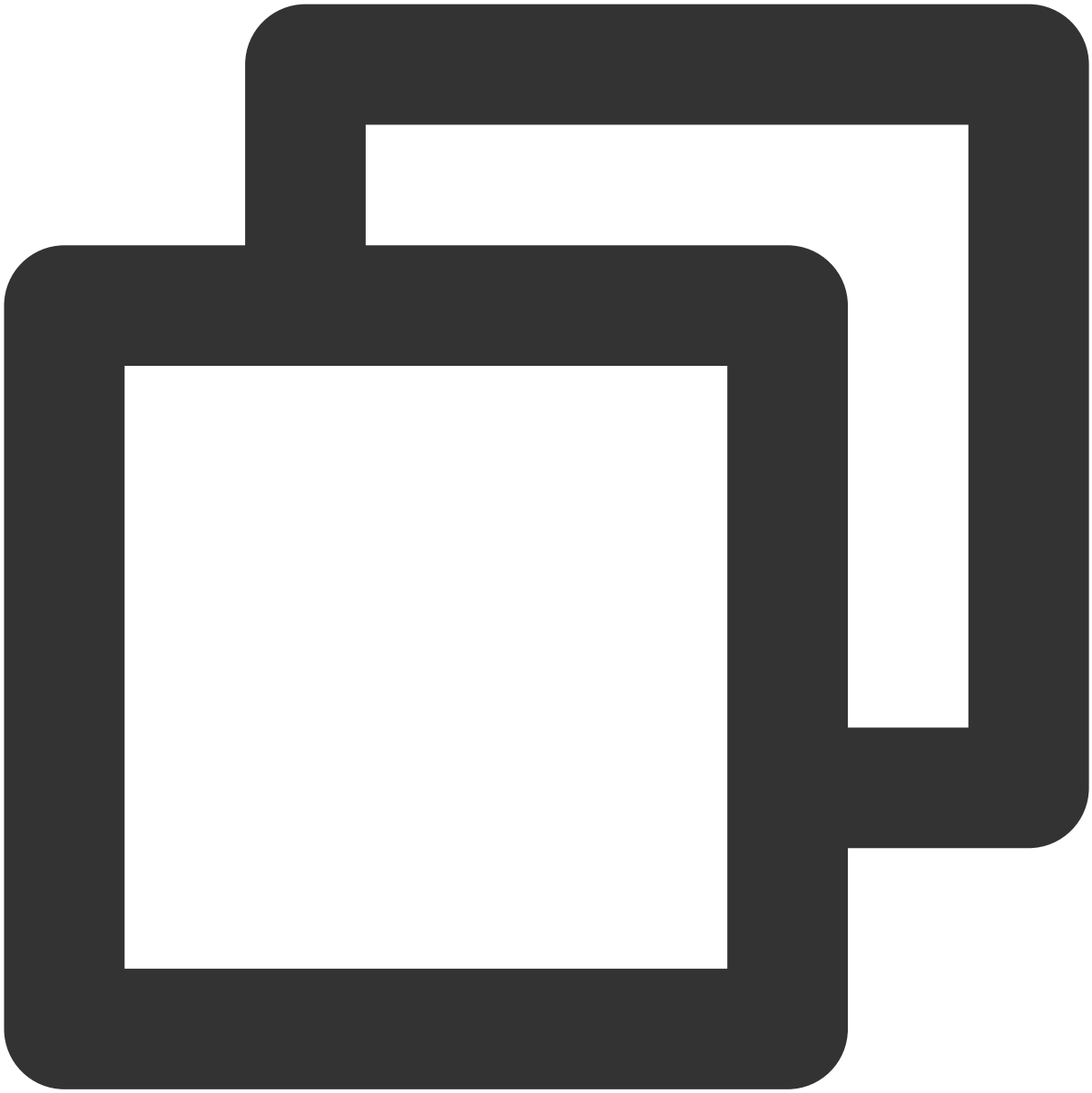
Parameter	Type	Description
roomId	TUICommonDefine.RoomId	The room ID.
callMediaType	TUICallDefine.MediaType	The call type, which can be video or audio.
callRole	TUICallDefine.Role	The role, which can be caller or callee.
totalTime	long	The call duration.

Note：

Client-side callbacks are often lost when errors occur, for example, when the process is closed. If you need to measure the duration of a call for billing or other purposes, we recommend you use the RESTful API.

onCallMediaTypeChanged

The call type changed.



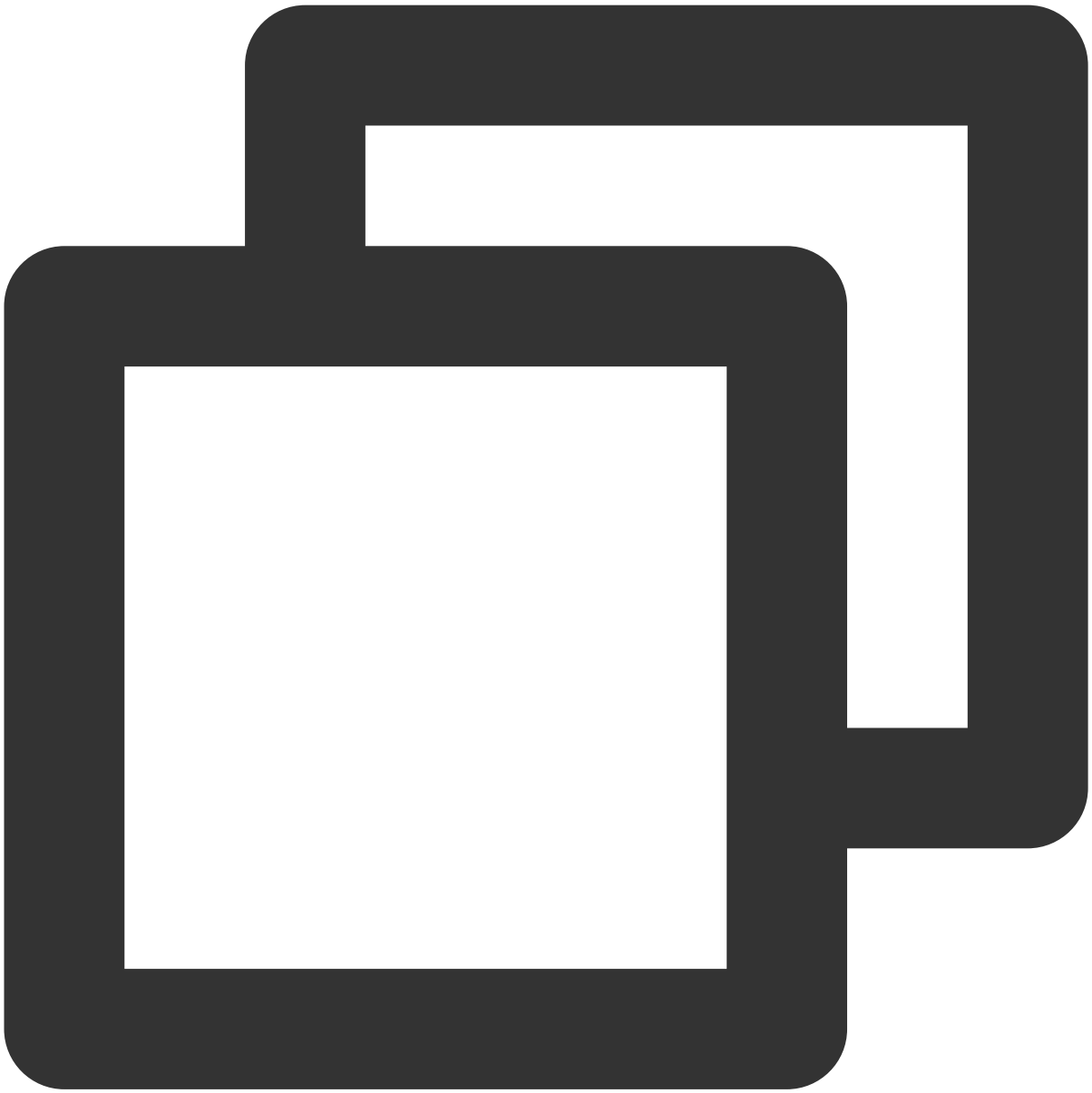
```
void onCallMediaTypeChanged(TUICallDefine.MediaType oldCallMediaType,TUICallDefine.
```

The parameters are described below:

Parameter	Type	Description
oldCallMediaType	TUICallDefine.MediaType	The call type before the change.
newCallMediaType	TUICallDefine.MediaType	The call type after the change.

onUserReject

The call was rejected. In a one-to-one call, only the inviter will receive this callback. In a group call, all invitees will receive this callback.



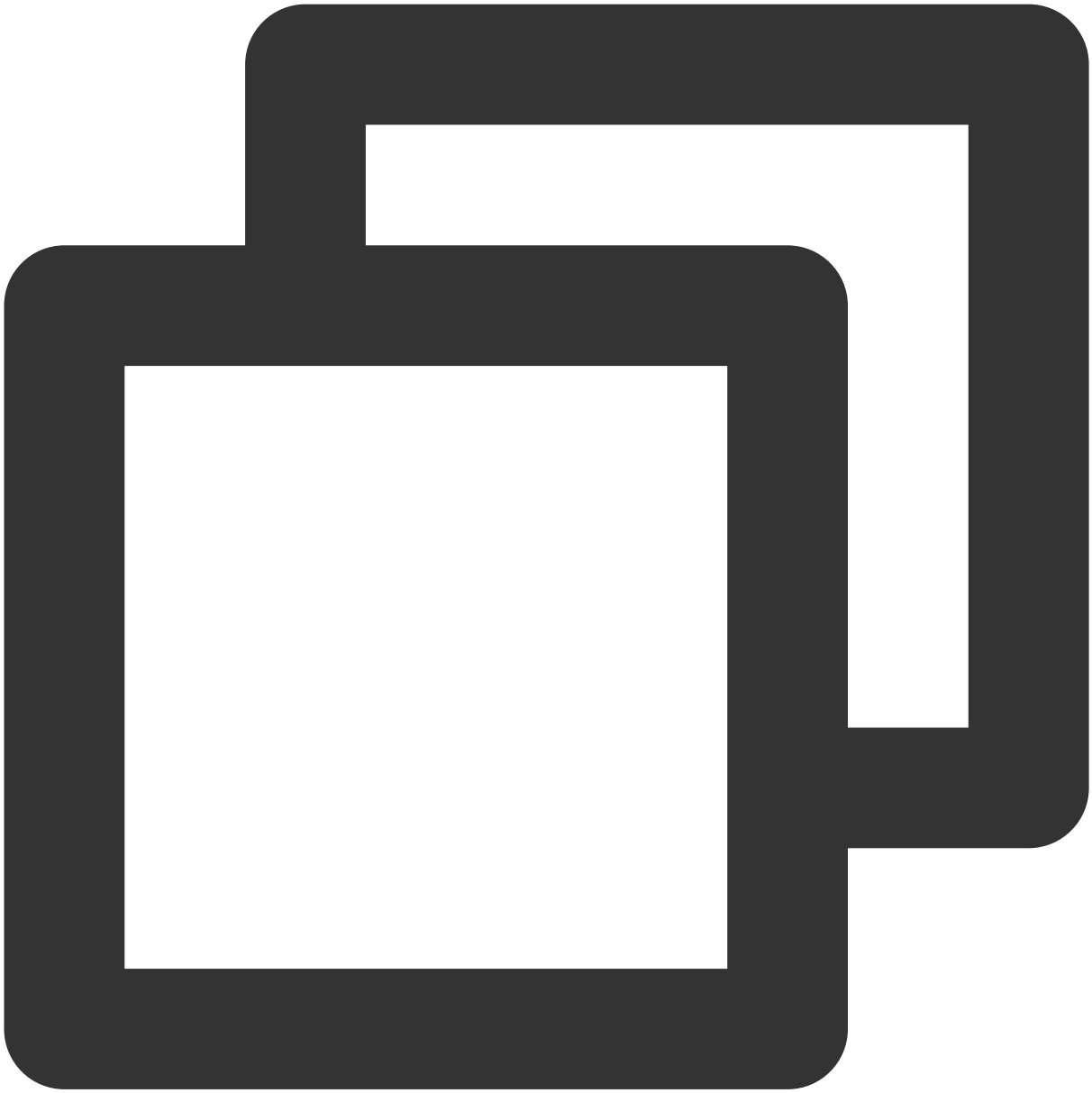
```
void onUserReject(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who rejected the call.

onUserNoResponse

A user did not respond.



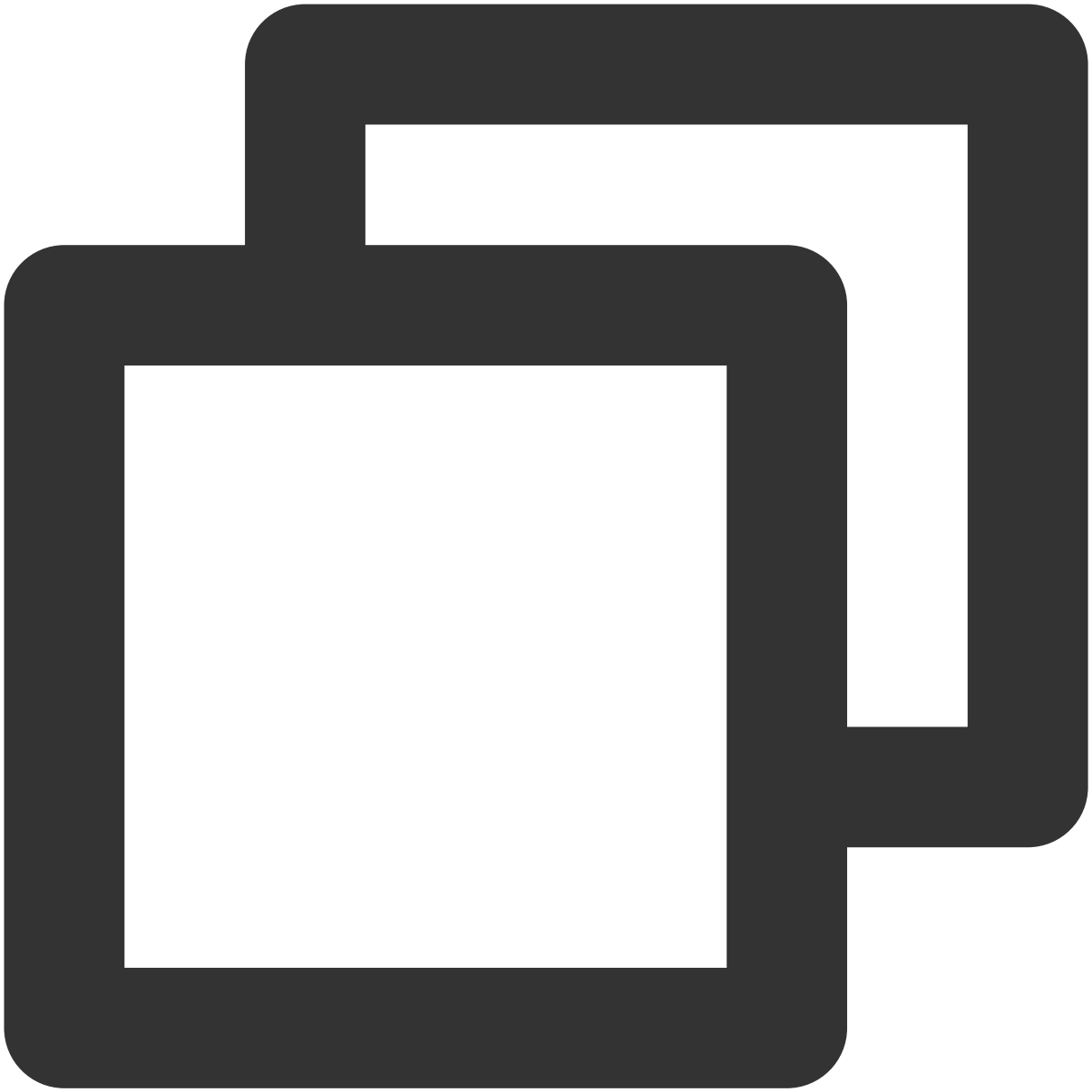
```
void onUserNoResponse(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who did not answer.

onUserLineBusy

A user is busy.



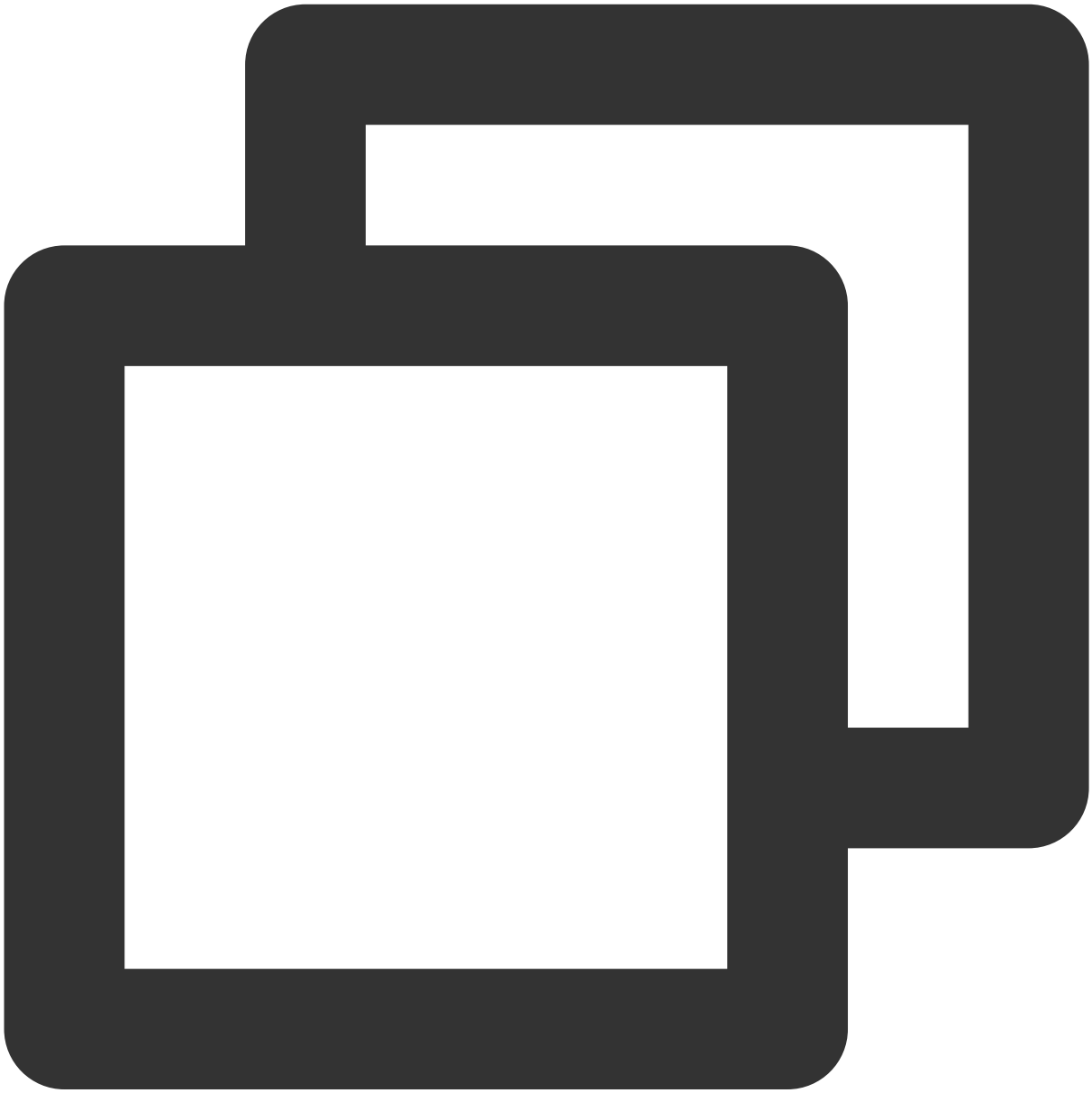
```
void onUserLineBusy(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who is busy.

onUserJoin

A user joined the call.



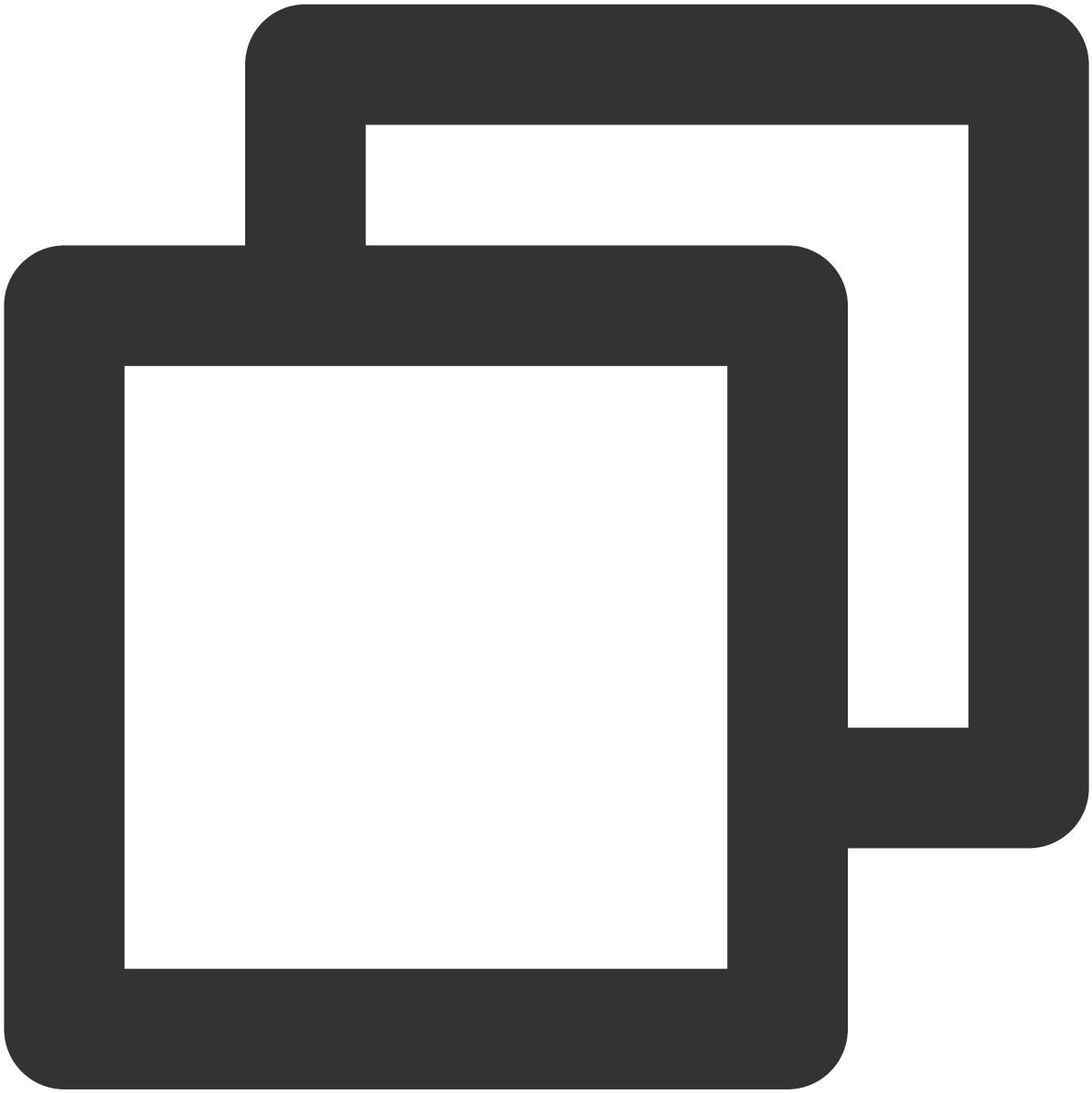
```
void onUserJoin(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The ID of the user who joined the call.

onUserLeave

A user left the call.



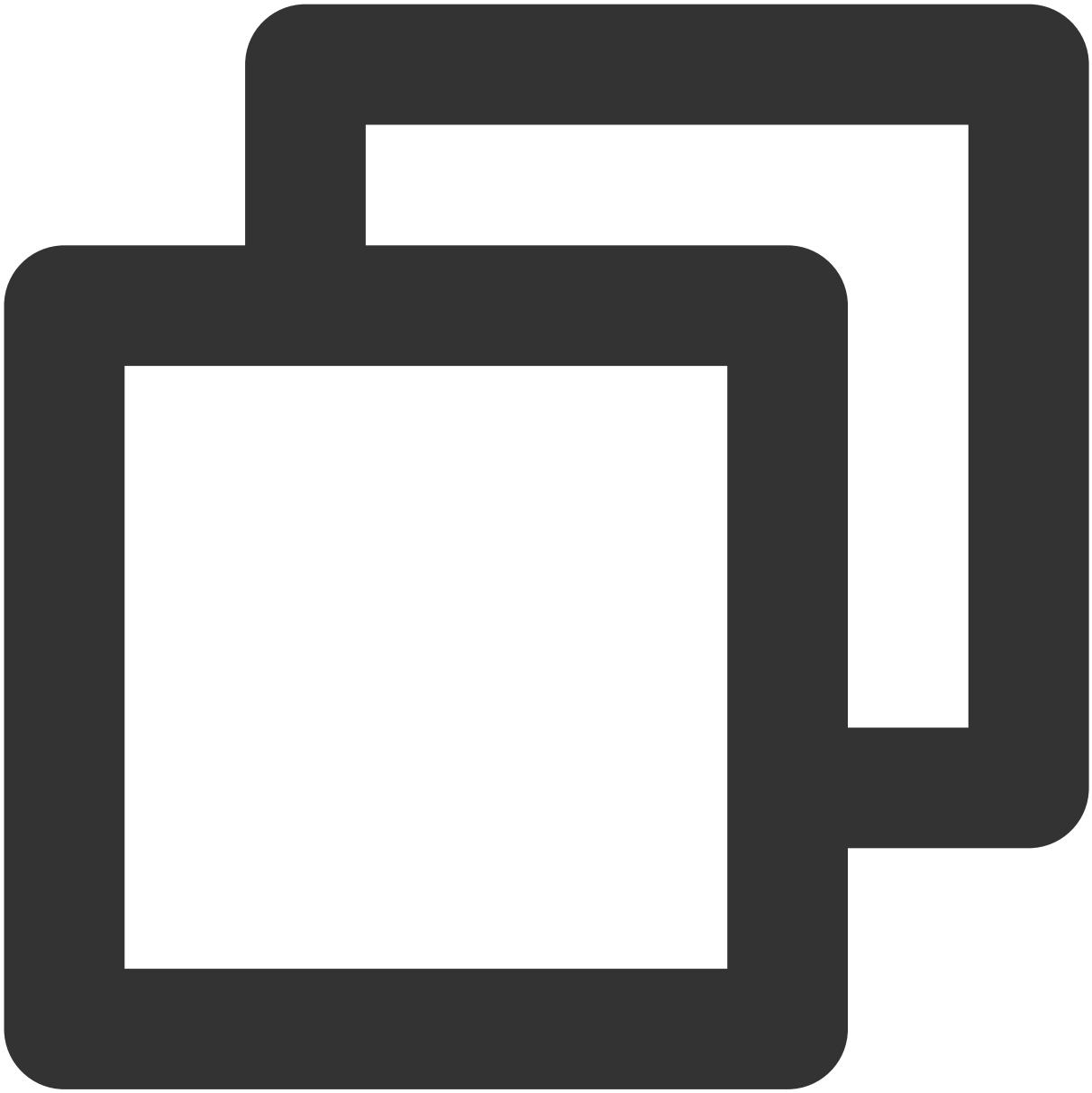
```
void onUserLeave(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The ID of the user who left the call.

onUserVideoAvailable

Whether a user is sending video.



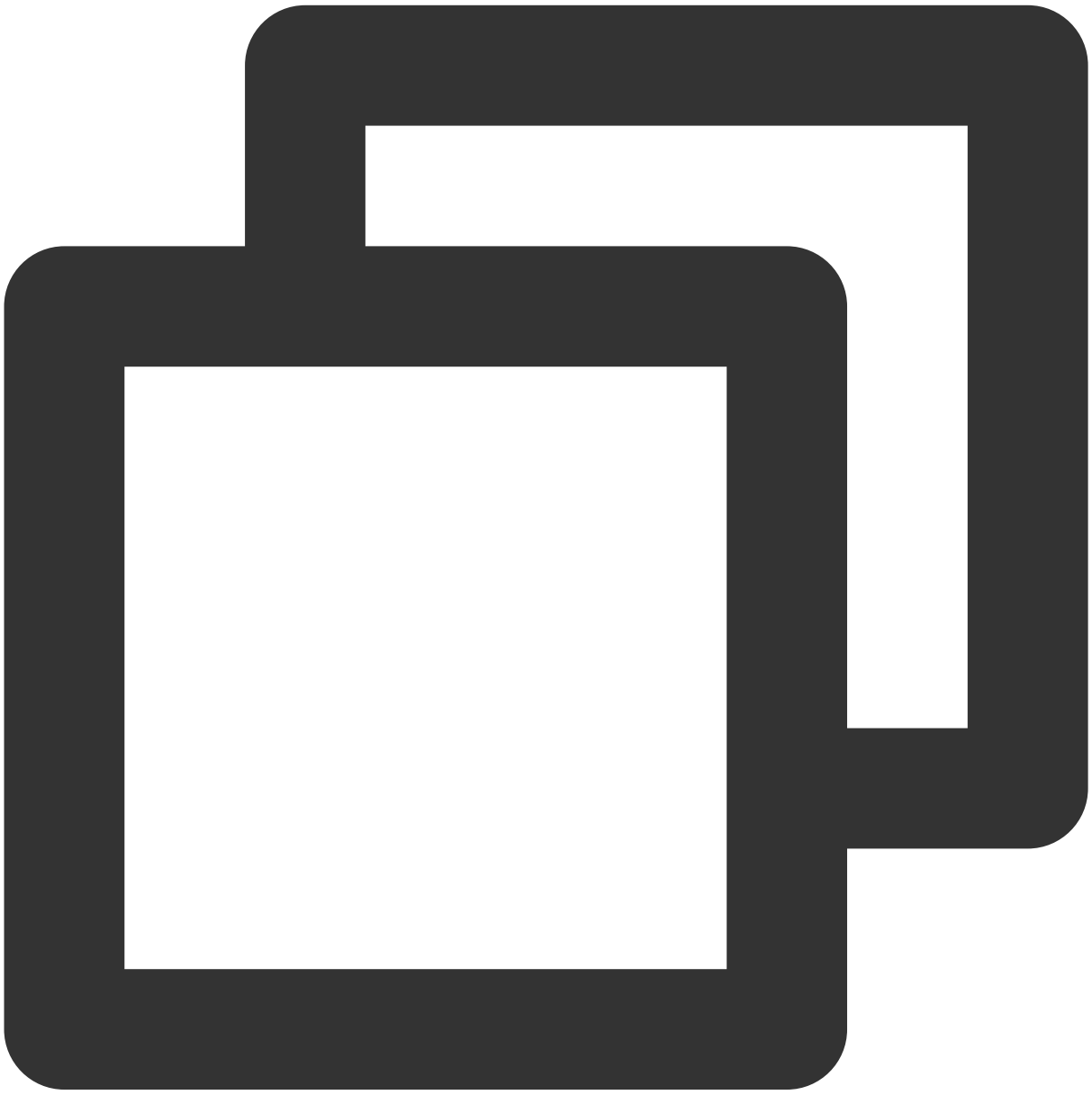
```
void onUserVideoAvailable(String userId, boolean isVideoAvailable);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
isVideoAvailable	boolean	Whether the user has video.

onUserAudioAvailable

Whether a user is sending audio.



```
void onUserAudioAvailable(String userId, boolean isAudioAvailable);
```

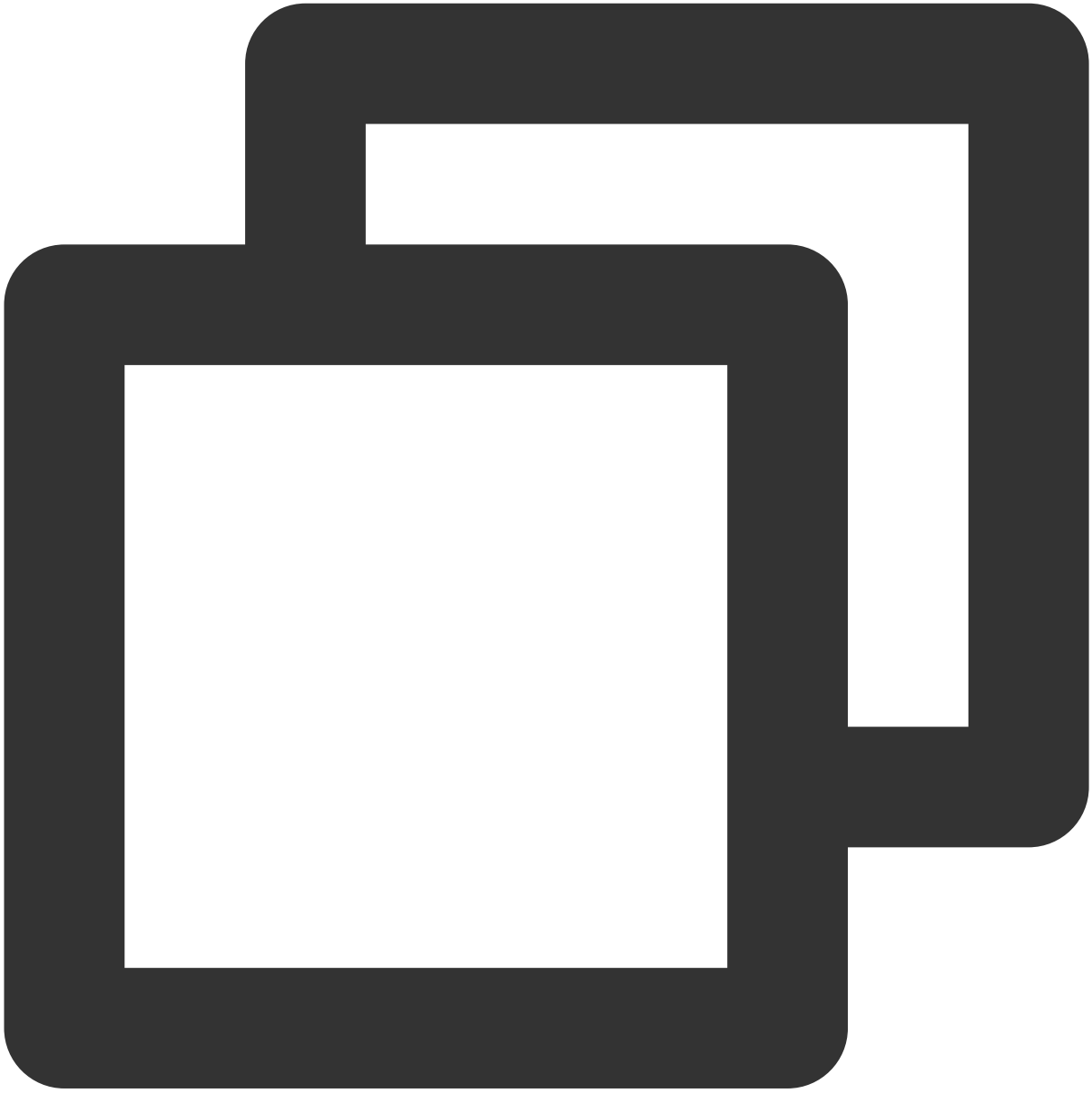
The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.

isAudioAvailable	boolean	Whether the user has audio.

onUserVoiceVolumeChanged

The volumes of all users.



```
void onUserVoiceVolumeChanged(Map<String, Integer> volumeMap);
```

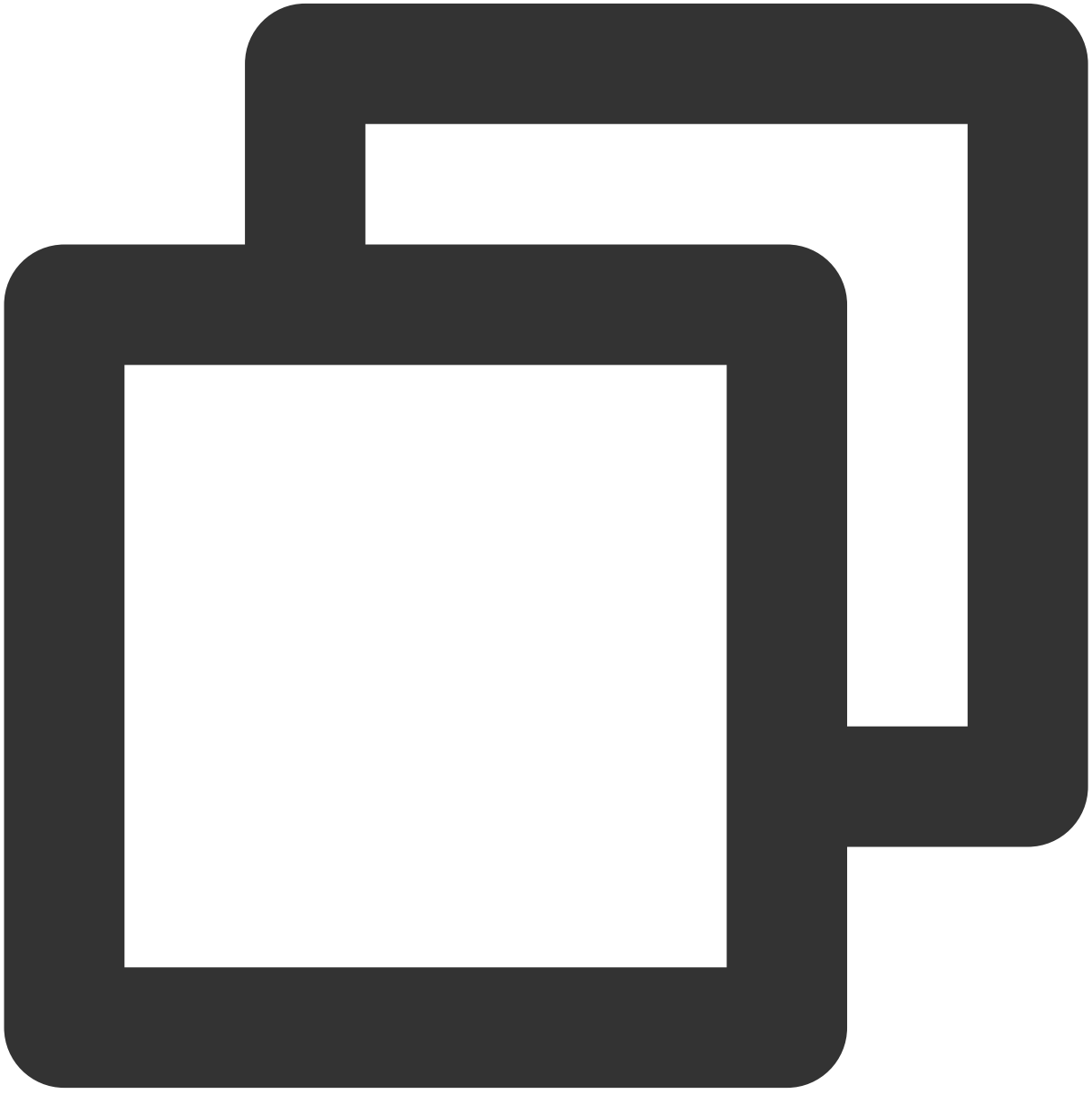
The parameters are described below:

Parameter	Type	Description

volumeMap	Map<String, Integer>	The volume table, which includes the volume of each user (`userId`). Value range: 0-100.

onUserNetworkQualityChanged

The network quality of all users.



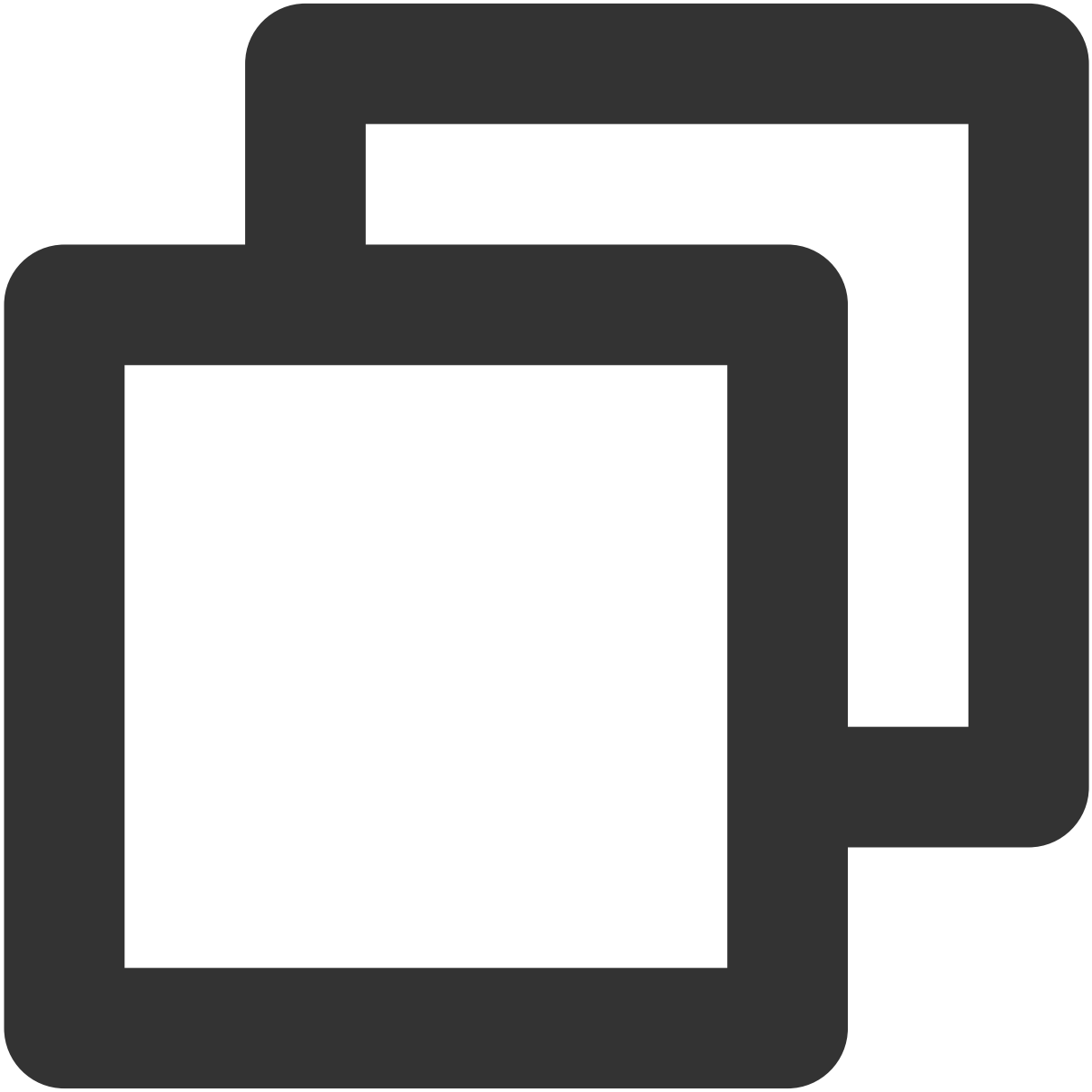
```
void onUserNetworkQualityChanged(List<TUICallDefine.NetworkQualityInfo> networkQual
```

The parameters are described below:

Parameter	Type	Description
networkQualityList	List	The current network conditions for all users (`userId`).

onKickedOffline

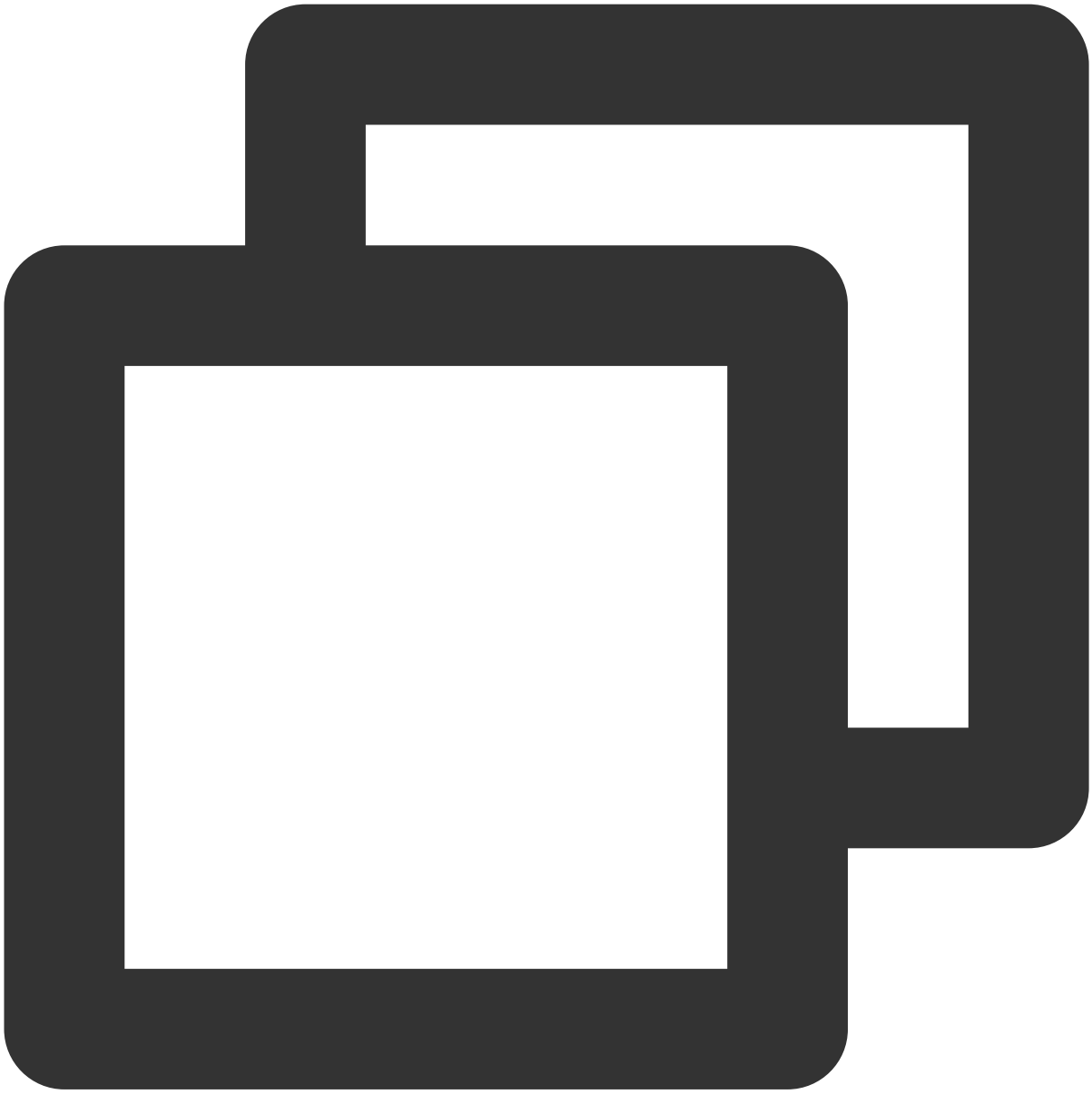
The current user was kicked offline：At this time, you can prompt the user with a UI message and then invoke `init` again.



```
void onKickedOffline();
```

onUserSigExpired

The user sig is expired：At this time, you need to generate a new `userSig` , and then invoke `init` again.



```
void onUserSigExpired();
```

Type Definition

Last updated：2023-09-19 15:29:26

Common structures

TUICallDefine

Type	Description
CallParams	An additional parameter.
OfflinePushInfo	Offline push vendor configuration information.

TUICommonDefine

Type	Description
RoomId	Room ID for audio and video in a call.
NetworkQualityInfo	Network quality information
VideoRenderParams	Video render parameters
VideoEncoderParams	Video encoding parameters

Enum definition

TUICallDefine

Type	Description
MediaType	Media type in a call
Role	Roles of individuals in a call.
Status	The call status
Scene	The call scene
IOSOfflinePushType	iOS offline push type

TUICommonDefine

Type	Description

AudioPlaybackDevice	Audio route
Camera	Camera type
NetworkQuality	Network quality
FillMode	Video image fill mode
Rotation	Video image rotation direction
ResolutionMode	Video aspect ratio mode
Resolution	Video resolution

CallParams

Call params

Value	Type	Description
offlinePushInfo	OfflinePushInfo	Offline push vendor configuration information.
timeout	int	Call timeout period, default: 30s, unit: seconds.
userData	String	An additional parameter. Callback when the callee receives onCallReceived

OfflinePushInfo

Offline push vendor configuration information, please refer to：Offline call push

Value	Type	Description
title	String	offlinepush notification title
desc	String	offlinepush notification description
ignoreIOSBadge	boolean	Ignore badge count for offline push (only for iOS), if set to true, the message will not increase the unread count of the app icon on the iOS receiver's side.
iOSSound	String	Offline push sound setting (only for iOS). When `sound = IOS_OFFLINE_PUSH_NO_SOUND`, there will be no sound played when the message is received. When `sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND`, the system sound will be played when the message is received. If you want to customize the iOSSound, you need to link

		the audio file into the Xcode project first, and then set the audio file name (with extension) to the iOSSound.
androidSound	String	Offline push sound setting (only for Android, supported by IMSDK 6.1 and above). Only Huawei and Google phones support setting sound prompts. For Xiaomi phones, please refer to: Xiaomi custom ringtones. In addition, for Google phones, in order to set sound prompts for FCM push on Android 8.0 and above systems, you must call `setAndroidFCMChannelID` to set the channelID for it to take effect.
androidOPPOChannelID	String	Set the channel ID for OPPO phones with Android 8.0 and above systems.
androidVIVOClassification	int	Classification of VIVO push messages (deprecated interface, VIVO push service will optimize message classification rules on April 3, 2023. It is recommended to use setAndroidVIVOCategory to set the message category). 0: Operational messages, 1: System messages. The default value is 1.
androidXiaoMiChannelID	String	Set the channel ID for Xiaomi phones with Android 8.0 and above systems.
androidFCMChannelID	String	Set the channel ID for google phones with Android 8.0 and above systems.
androidHuaWeiCategory	String	Classification of Huawei push messages, please refer to: Huawei message classification standard.
isDisablePush	boolean	Whether to turn off push notifications (default is on).
iOSPushType	IOSOfflinePushType	iOS offline push type， default is APNs

RoomId

Room ID for audio and video in a call.

Note：

(1) `intRoomId` and `strRoomId` are mutually exclusive. If you choose to use `strRoomId`, `intRoomId` needs to be filled in as 0. If both are filled in, the SDK will prioritize `intRoomId`.

(2) Do not mix `intRoomId` and `strRoomId` because they are not interchangeable. For example, the number 123 and the string "123" represent two completely different rooms.

Value	Type	Description
intRoomId	int	Numeric room ID. **range**：1 - 2147483647(2^31-1)
strRoomId	String	String room ID. **range**： Limited to 64 bytes in length. The supported character set range is as follows (a total of 89 Lowercase and uppercase English letters.（a-zA-Z）； Number（0-9）； `Spaces`、`!`、`#`、`$`、`%`、`&`、`(`、`)`、`+`、`-`、`:`、`;`、`<` 。

Note：

Currently, string room number is only supported on Android and iOS platforms. Support for other platforms such as Web, Mini Programs, Flutter, and Uniapp will be available in the future. Please stay tuned!

NetworkQualityInfo

User network quality information

Value	Type	Description
userId	String	user ID
quality	NetworkQuality	network quality

VideoRenderParams

Video render parameters

Value	Type	Description
fillMode	VideoRenderParams.FillMode	Video image fill mode
rotation	VideoRenderParams.Rotation	Video image rotation direction

VideoEncoderParams

Video encoding parameters

Value	Type	Description
resolution	VideoEncoderParams.Resolution	Video resolution

resolutionMode	VideoEncoderParams.ResolutionMode	Video aspect ratio mode

MediaType

Call media type

Value	Type	Description
Unknown	0	Unknown
Audio	1	Audio call
Video	2	Video call

Role

Call role

Value	Type	Description
None	0	Unknown
Caller	1	Caller（inviter）
Called	2	Callee（invitee）

Status

Call status

Value	Type	Description
None	0	Unknown
Waiting	1	The call is currently waiting
Accept	2	The call has been accepted

Scene

Call scene

Value	Type	Description
GROUP_CALL	0	Group call
MULTI_CALL	1	Anonymous group calling (not supported at this moment, please stay tuned).

SINGLE_CALL	2	one to one call

IOSOfflinePushType

iOS offline push type

Type	Value	Description
APNs	0	APNs
VoIP	1	VoIP

AudioPlaybackDevice

Audio route

Type	Value	Description
Earpiece	0	Earpiece
Speakerphone	1	Speakerphone

Camera

Front/Back camera

Type	Value	Description
Front	0	Front camera
Back	1	Back camera

NetworkQuality

Network quality

Type	Value	Description
Unknown	0	Unknown
Excellent	1	Excellent
Good	2	Good
Poor	3	Poor
Bad	4	Bad
Vbad	5	Vbad

Down	6	Down

FillMode

If the aspect ratio of the video display area is not equal to that of the video image, you need to specify the fill mode:

Type	Value	Description
Fill	0	Fill mode: the video image will be centered and scaled to fill the entire display area, where parts that exceed the area will be cropped. The displayed image may be incomplete in this mode.
Fit	1	Fit mode: the video image will be scaled based on its long side to fit the display area, where the short side will be filled with black bars. The displayed image is complete in this mode, but there may be black bars.

Rotation

We provides rotation angle setting APIs for local and remote images. The following rotation angles are all clockwise.

Type	Value	Description
Rotation_0	0	No rotation
Rotation_90	1	Clockwise rotation by 90 degrees
Rotation_180	2	Clockwise rotation by 180 degrees
Rotation_270	3	Clockwise rotation by 0 degrees

ResolutionMode

Video aspect ratio mode

Type	Value	Description
Landscape	0	Landscape resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360.
Portrait	1	Portrait resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640.

Resolution

Video resolution

Type	Value	Description
Resolution_640_360	108	Aspect ratio: 16:9；resolution: 640x360；recommended bitrate: 500kbps
Resolution_640_480	62	Aspect ratio: 4:3；resolution: 640x480；recommended bitrate: 600kbps
Resolution_960_540	110	Aspect ratio: 16:9；resolution: 960x540；recommended bitrate: 850kbps
Resolution_960_720	64	Aspect ratio: 4:3；resolution: 960x720；recommended bitrate: 1000kbps
Resolution_1280_720	112	Aspect ratio: 16:9；resolution: 1280x720；recommended bitrate: 1200kbps
Resolution_1920_1080	114	Aspect ratio: 16:9；resolution: 1920x1080；recommended bitrate: 2000kbps

iOS
API Overview

Last updated：2024-07-25 17:49:30

TUICallKit (UI Included)

TUICallKit is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat.

API	Description
createInstance	Create a TUICallKit instance (singleton mode).
setSelfInfo	Set the user's profile picture and nickname.
call	Make a one-to-one call.
call	Make a one-to-one call, Support for custom room ID, call timeout, offline push content, etc
groupCall	Make a group call.
groupCall	Make a group call, Support for custom room ID, call timeout, offline push content, etc
joinInGroupCall	Join a group call.
setCallingBell	Set the ringtone.
enableMuteMode	Set whether to turn on the mute mode.
enableFloatWindow	Set whether to enable floating windows.
enableIncomingBanner	Set whether to display incoming banner.

TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description

createInstance	Create a TUICallEngine instance (singleton).
destroyInstance	Destroy TUICallEngine instance (singleton).
init	Authenticates the basic audio/video call capabilities.
addObserver	Add listener.
removeObserver	Remove listener.
call	Make a one-to-one call.
groupCall	Make a group call.
accept	Accept call.
reject	Reject call.
hangup	Hang up call.
ignore	Ignore call.
inviteUser	Invite users to the current group call.
joinInGroupCall	Join a group call.
switchCallMediaType	Switch the call media type, such as from video call to audio call.
startRemoteView	Subscribe to the video stream of a remote user.
stopRemoteView	Unsubscribe from the video stream of a remote user.
openCamera	Turn on the camera.
closeCamera	Turn off the camera.
switchCamera	Switch camera.
openMicrophone	Enable microphone.
closeMicrophone	Disable the microphone.
selectAudioPlaybackDevice	Select the audio playback device (Earpiece/Speakerphone).
setSelfInfo	Set the user's profile picture and nickname.
enableMultiDeviceAbility	Sets whether to enable multi-device login for TUICallEngine (supported by the Group Call package).
setVideoRenderParams	Set the rendering mode of video.

setVideoEncoderParams	Set the encoding parameters of video encoder.
getTRTCCloudInstance	Advanced features.
setBeautyLevel	Set beauty level, support turning off default beauty.

TUICallObserver

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

API	Description
onError	An error occurred during the call.
onCallReceived	A call was received.
onCallCancelled	The call was canceled.
onCallBegin	The call was connected.
onCallEnd	The call ended.
onCallMediaTypeChanged	The call type changed.
onUserReject	A user declined the call.
onUserNoResponse	A user didn't respond.
onUserLineBusy	A user was busy.
onUserJoin	A user joined the call.
onUserLeave	A user left the call.
onUserVideoAvailable	Whether a user has a video stream.
onUserAudioAvailable	Whether a user has an audio stream.
onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.
onKickedOffline	The current user was kicked offline.
onUserSigExpired	The user sig is expired.

Definitions of Key Typ

API	Description
TUICallMediaType	Call media type, Enumeration type: Unknown, Video, and Audio.
TUICallRole	Call role, Enumeration type: None, Call, and Called.
TUICallStatus	Call status, Enumeration type: None, Waiting, and Accept.
TUIRoomId	The room ID, which can be a number or string.
TUICallCamera	The camera type. Enumeration type: Front and Back.
TUIAudioPlaybackDevice	The audio playback device type. Enumeration type: Earpiece and Speakerphone.
TUINetworkQualityInfo	The current network quality.

TUICallKit

Last updated：2024-07-24 10:20:03

TUICallKit APIs

`TUICallKit` is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat. For directions on integration, see Integrating TUICallKit.

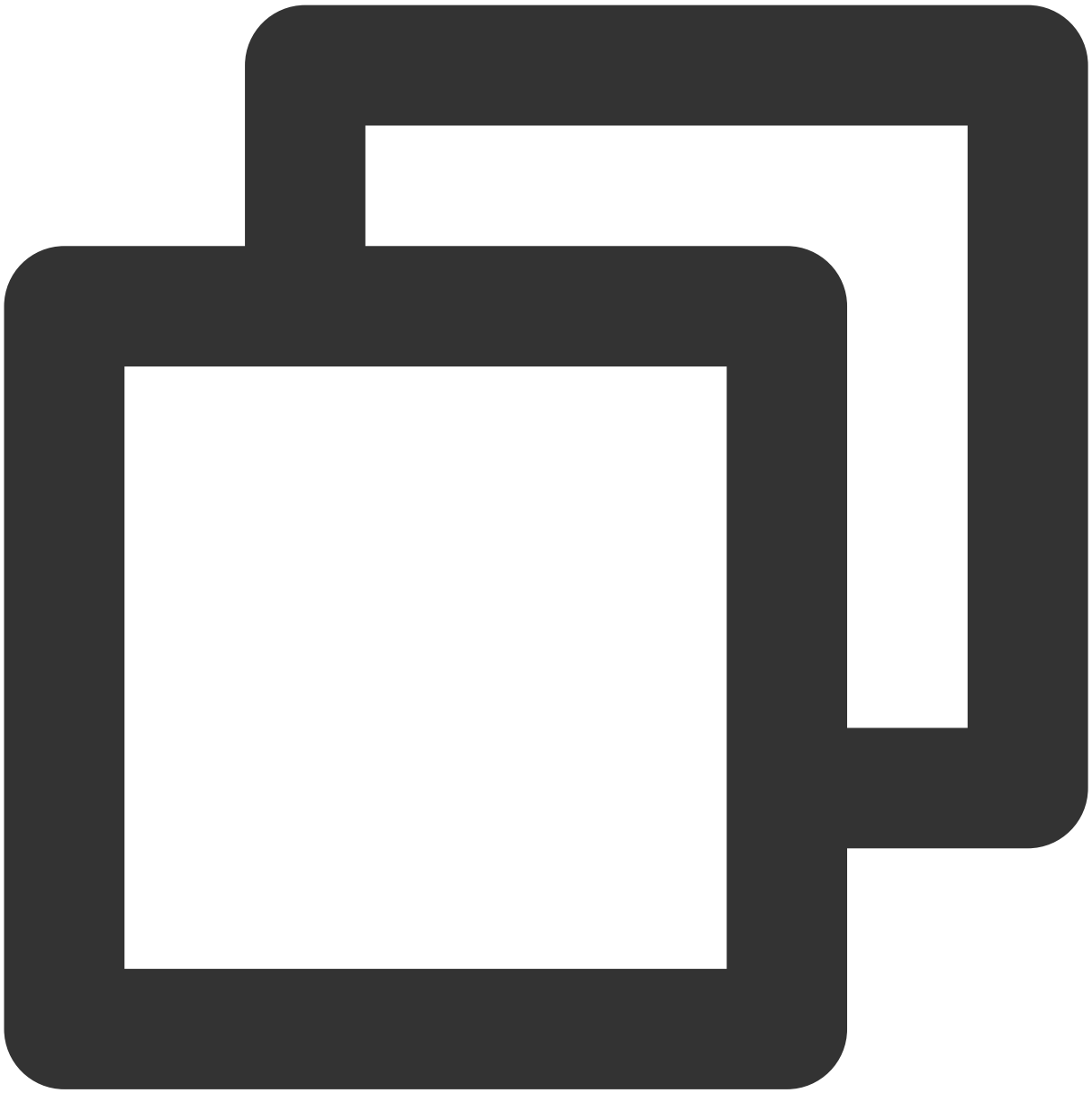
Overview

API	Description
createInstance	Create a TUICallKit instance (singleton mode).
setSelfInfo	Set the user's profile picture and nickname.
call	Make a one-to-one call.
call	Make a one-to-one call, Support for custom room ID, call timeout, offline push content, etc
groupCall	Make a group call.
groupCall	Make a group call, Support for custom room ID, call timeout, offline push content, etc
joinInGroupCall	Join a group call.
setCallingBell	Set the ringtone.
enableMuteMode	Set whether to turn on the mute mode.
enableFloatWindow	Set whether to enable floating windows.
enableIncomingBanner	Set whether to display incoming banner.

Details

createInstance

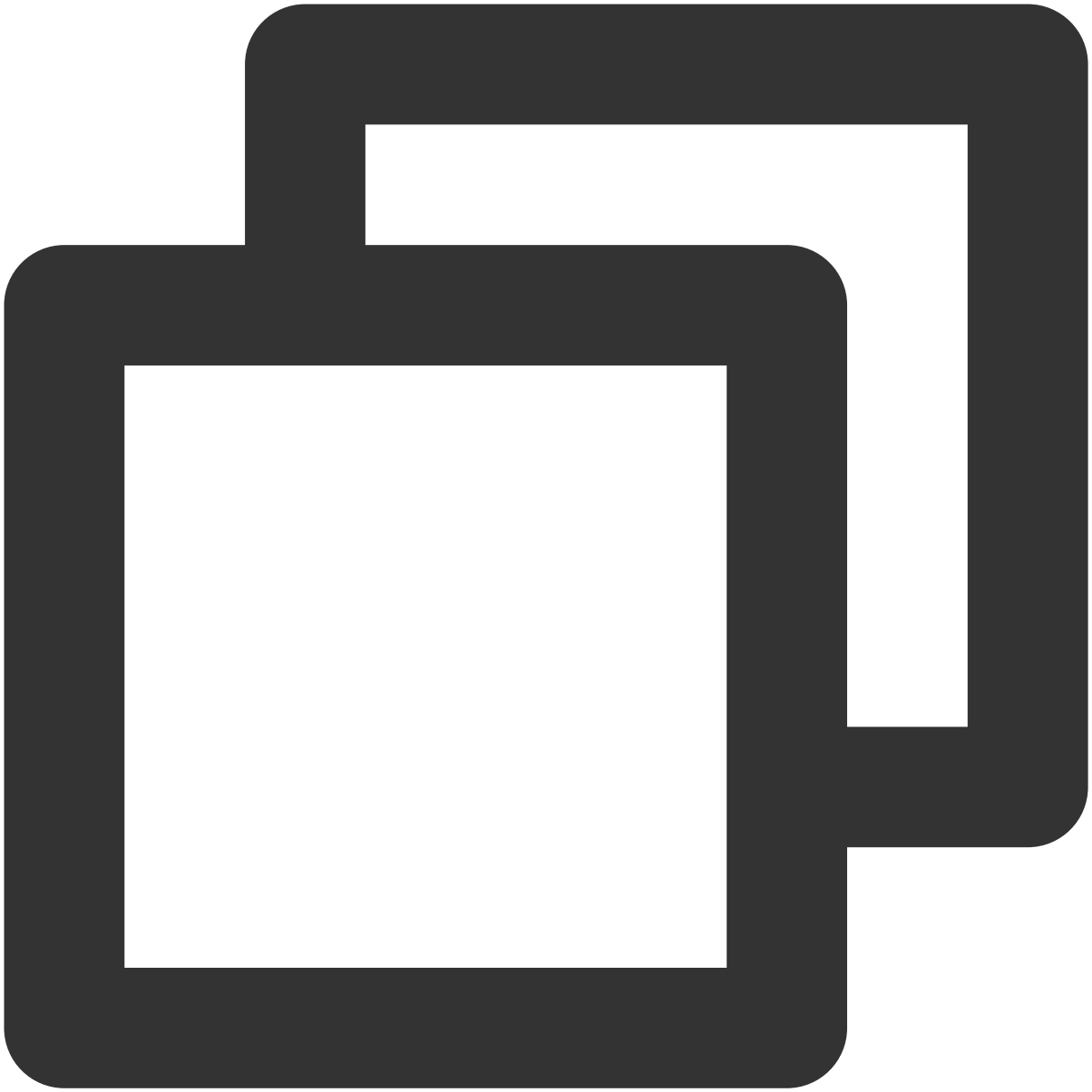
This API is used to create a `TUICallKit` singleton.



```
public static func createInstance() -> TUICallKit
```

setSelfInfo

This API is used to set the user's profile picture and nickname. The nickname cannot exceed 500 bytes, and the profile picture is specified by a URL.

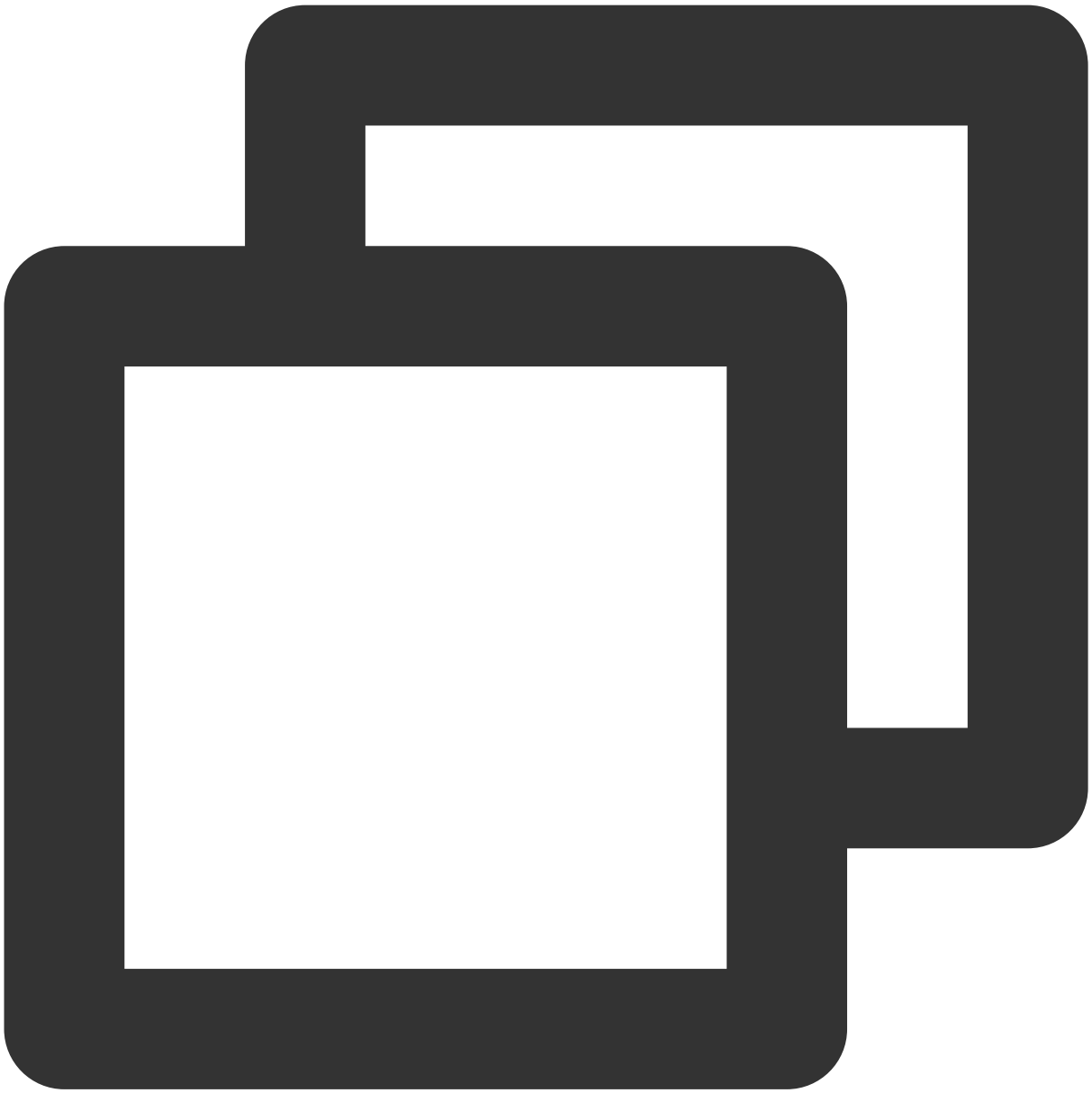


```
public func setSelfInfo(nickname: String, avatar: String, succ:@escaping TUICallSuc
```

Parameter	Type	Description
nickname	String	The nickname.
avatar	String	The profile picture.

call

This API is used to make a (one-to-one) call.

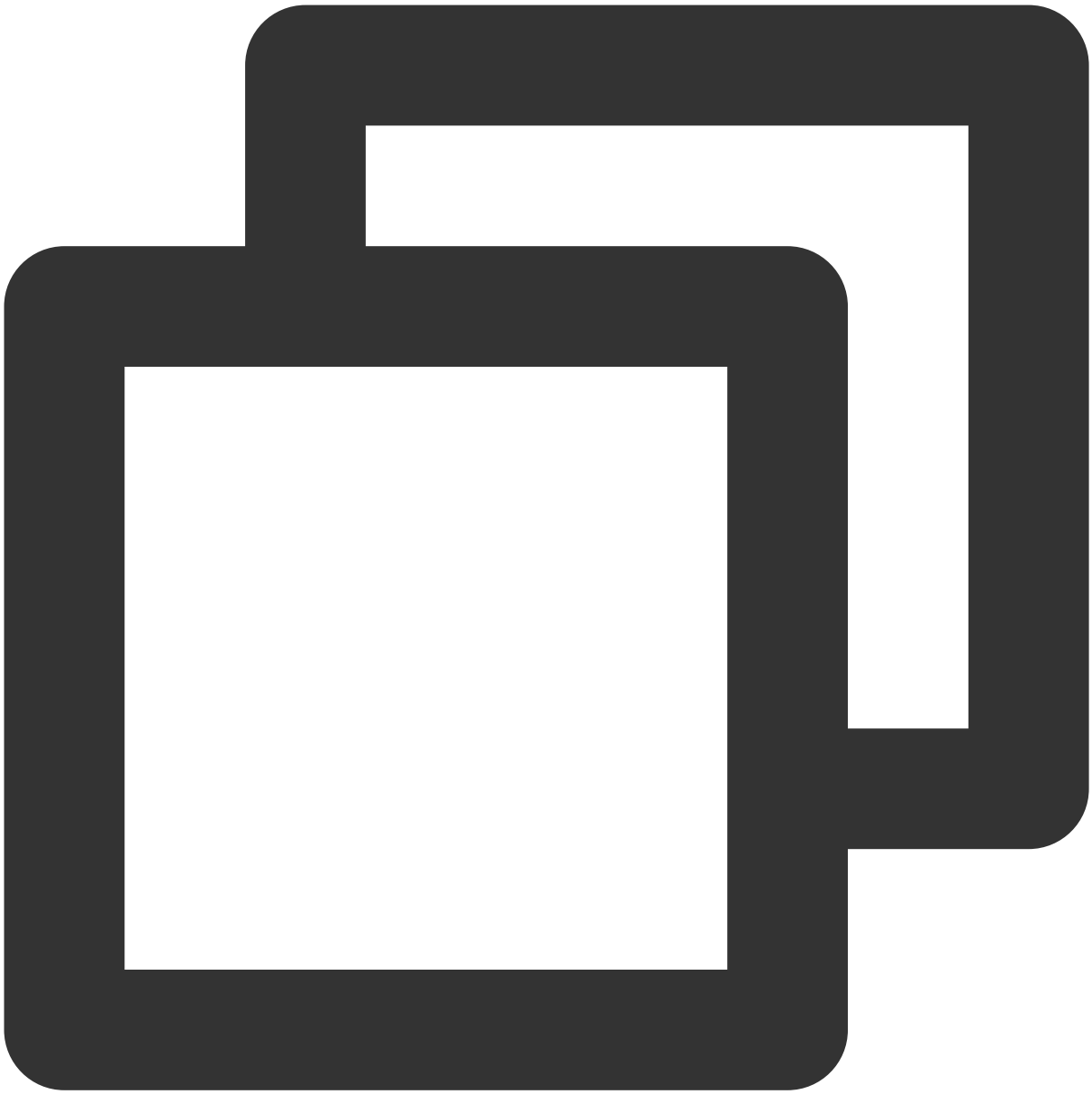


```
public func call(userId: String, callMediaType: TUICallMediaType)
```

Parameter	Type	Description
userId	String	The target user ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.

call

This API is used to make a (one-to-one) call, Support for custom room ID, call timeout, offline push content, etc.



```
public func call(userId: String, callMediaType: TUICallMediaType, params: TUICallPa
                 succ: @escaping TUICallSucc, fail: @escaping TUICallFail)
```

Parameter	Type	Description
userId	String	The target user ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.

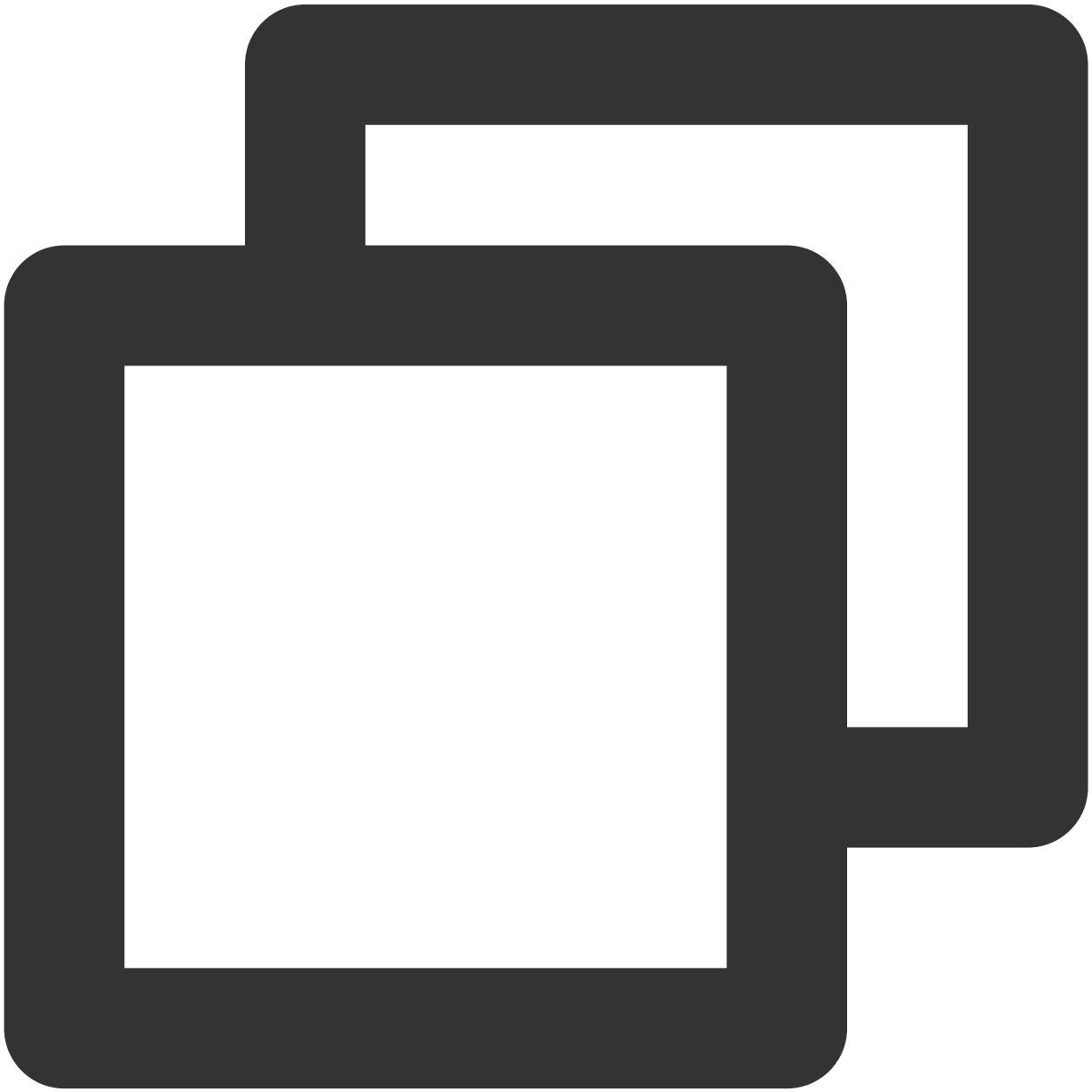
params	TUICallParams	Call extension parameters, such as roomID, call timeout, offline push info,etc

groupCall

This API is used to make a group call.

Note:

Before making a group call, you need to create an Chat group first.



```
public func groupCall(groupId: String, userIdList: [String], callMediaType: TUICall
```

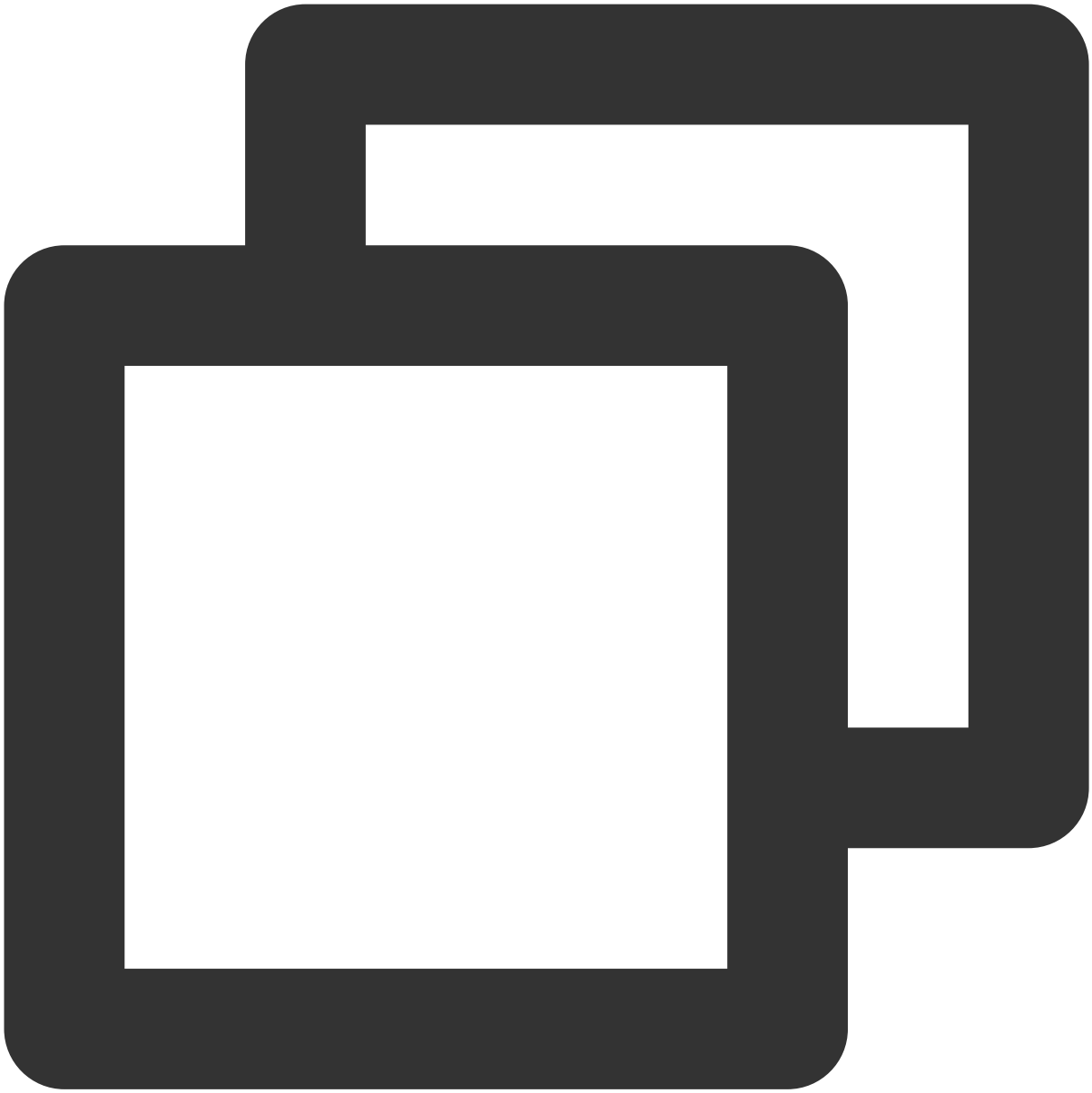
Parameter	Type	Description
groupId	String	The group ID.
userIdList	Array	The target user IDs.
callMediaType	TUICallMediaType	The call type, which can be video or audio.

groupCall

This API is used to make a group call, Support for custom room ID, call timeout, offline push content, etc.

Note:

Before making a group call, you need to create an Chat group first.



```
public func groupCall(groupId: String, userIdList: [String], callMediaType: TUICall
                      succ: @escaping TUICallSucc, fail: @escaping TUICallFail)
```

Parameter	Type	Description
groupId	String	The group ID.
userIdList	Array	The target user IDs.
callMediaType	TUICallMediaType	The call type, which can be video or audio.

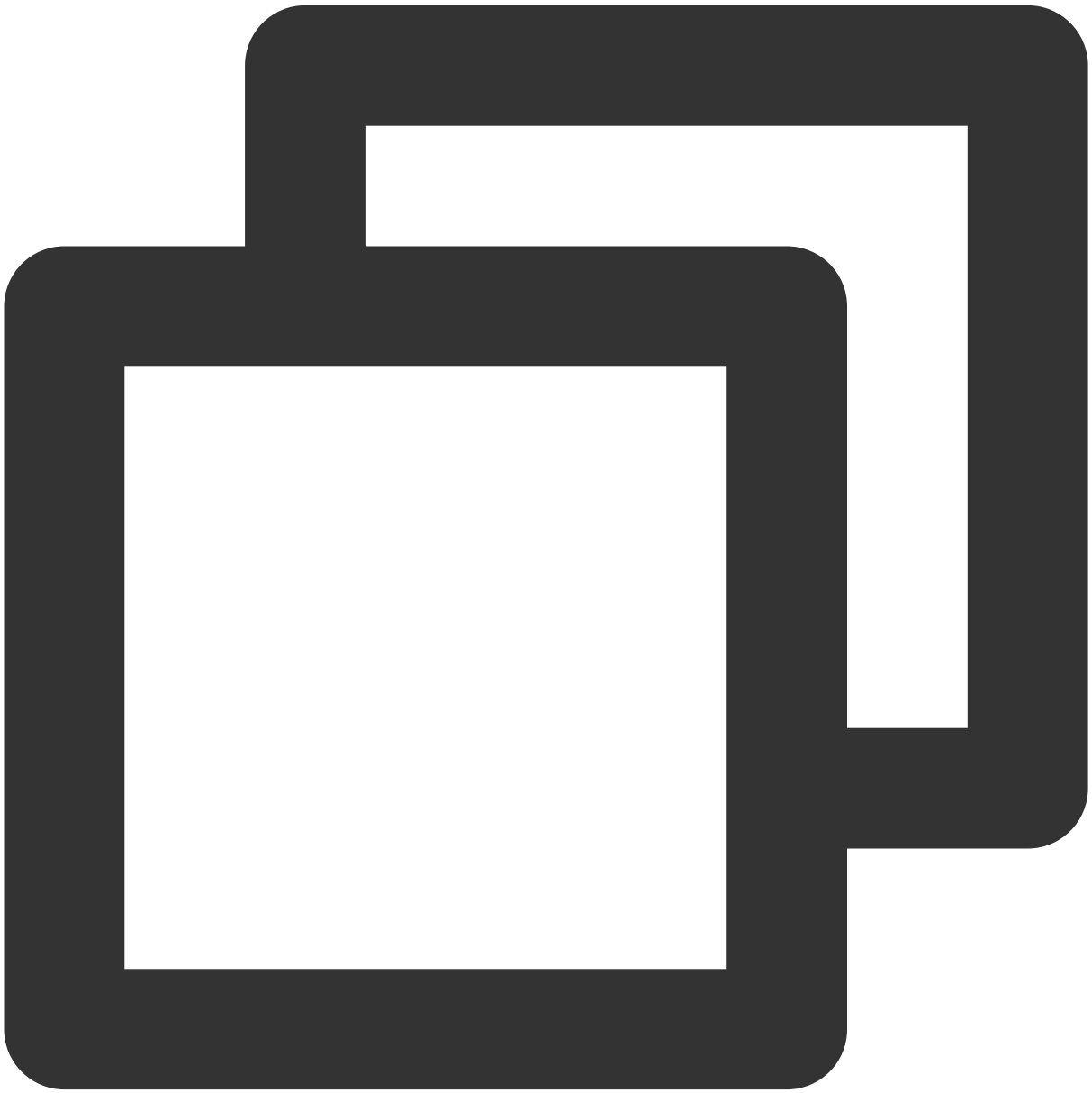
params	TUICallParams	Call extension parameters, such as roomID, call timeout, offline push info, etc

joinInGroupCall

This API is used to join a group call.

Note:

Before joining a group call, you need to create or join an Chat group in advance, and there are already users in the group who are in the call.



```
public func joinInGroupCall(roomId: TUIRoomId, groupId: String, callMediaType: TUIC
```

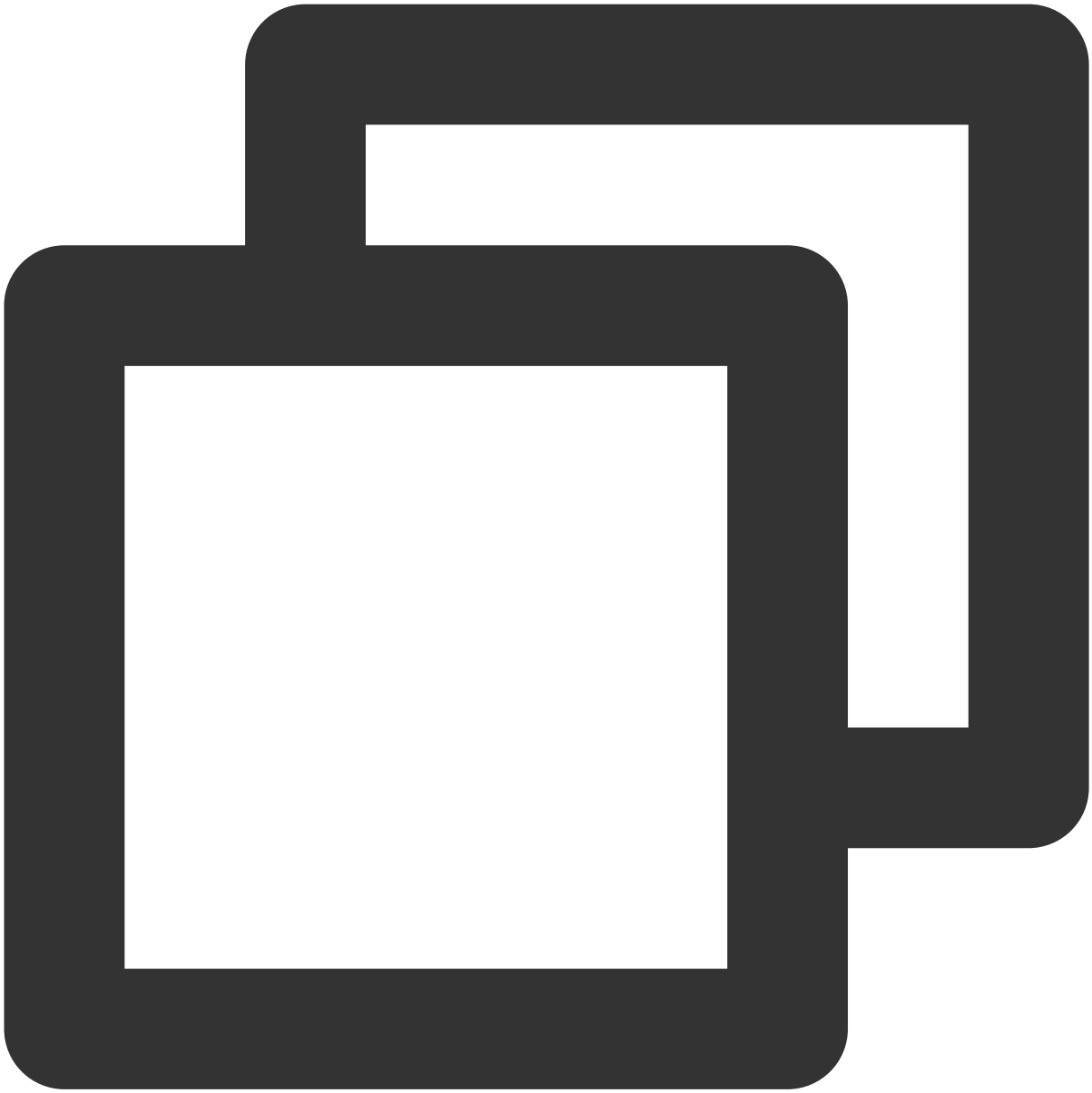
Parameter	Type	Description
roomId	TUIRoomId	The room ID.
groupId	String	The group ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.

setCallingBell

This API is used to set the ringtone. `filePath` must be an accessible local file URL.

The ringtone set is associated with the device and does not change with user.

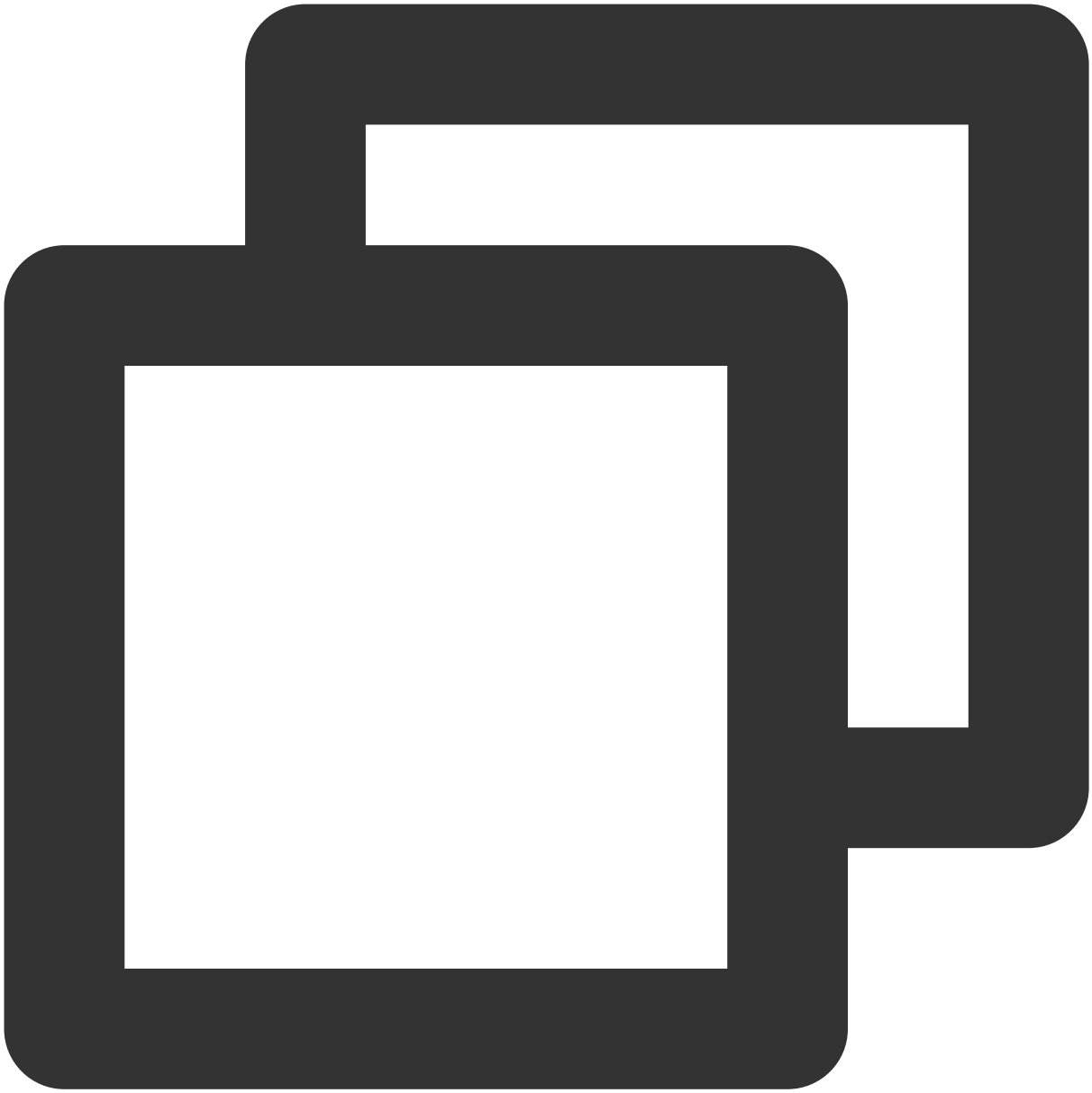
To reset the ringtone, pass in an empty string for `filePath` .



```
public func setCallingBell(filePath: String)
```

enableMuteMode

This API is used to set whether to play the music when user received a call.

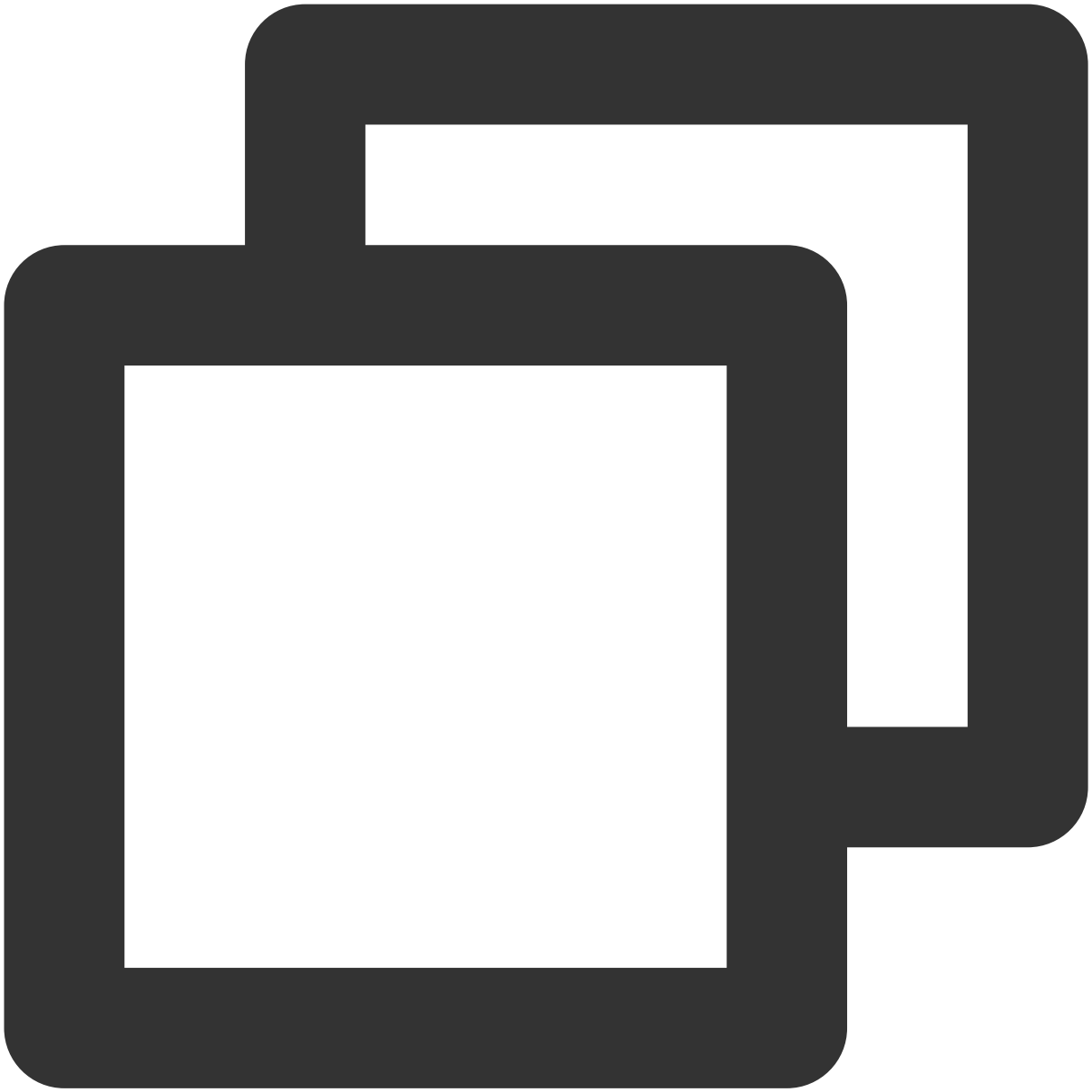


```
public func enableMuteMode(enable: Bool)
```

enableFloatWindow

This API is used to set whether to enable floating windows.

The default value is `false`, and the floating window button in the top left corner of the call view is hidden. If it is set to `true`, the button will become visible.

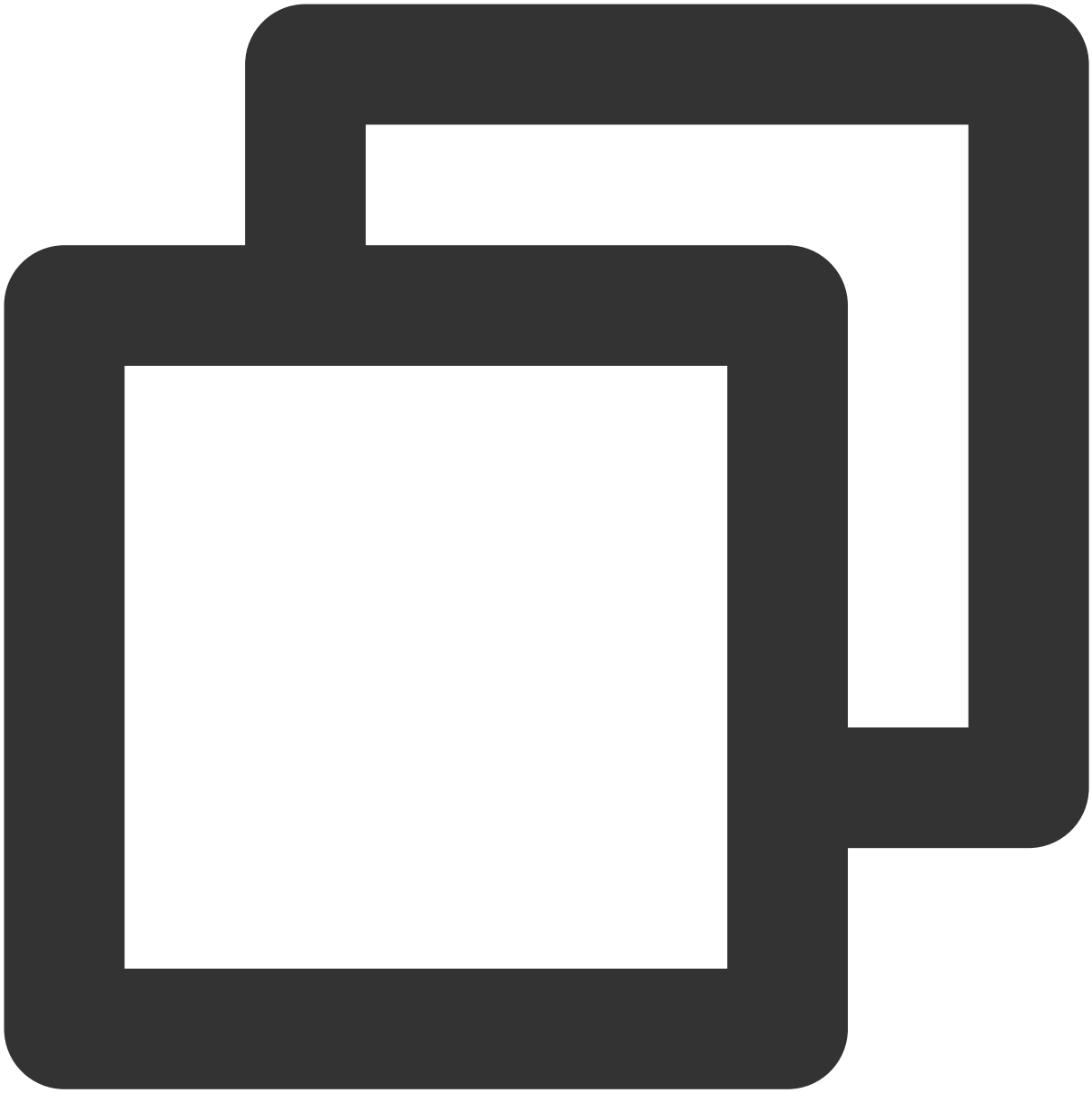


```
public func enableFloatWindow(enable: Bool)
```

enableIncomingBanner

The API is used to set whether show incoming banner when user received a new call invitation.

The default value is `false` , The callee will pop up a full-screen call view by default when receiving the invitation. If it is set to `true` , the callee will display a banner first.



```
public func enableIncomingBanner(enable: Bool)
```

TUICallEngine

Last updated：2024-07-25 17:49:30

TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

Overview

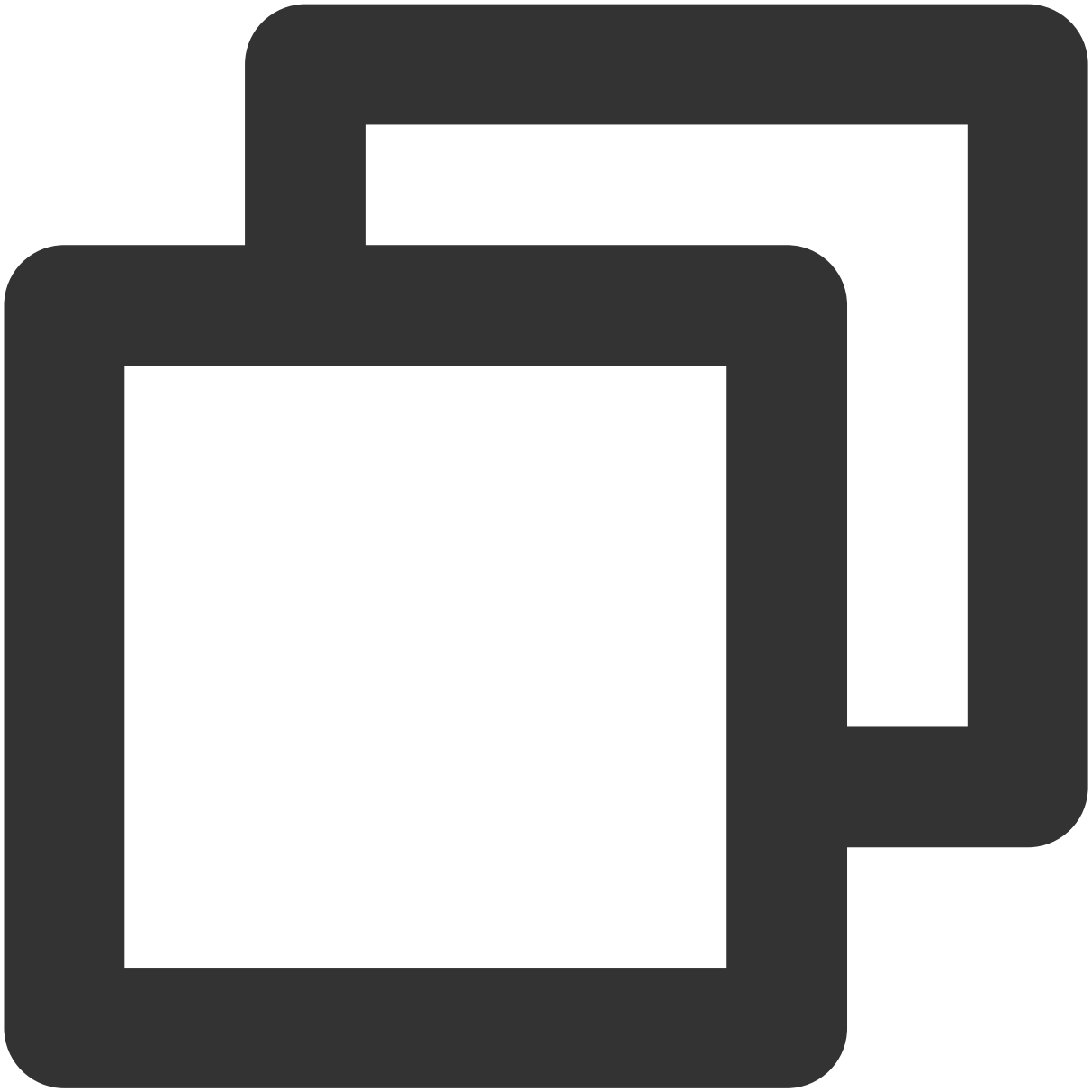
API	Description
createInstance	Create a TUICallEngine instance (singleton).
destroyInstance	Destroy TUICallEngine instance (singleton).
init	Authenticates the basic audio/video call capabilities.
addObserver	Add listener.
removeObserver	Remove listener.
call	Make a one-to-one call.
groupCall	Make a group call.
accept	Accept call.
reject	Reject call.
hangup	Hang up call.
ignore	Ignore call.
inviteUser	Invite users to the current group call.
joinInGroupCall	Join a group call.
switchCallMediaType	Switch the call media type, such as from video call to audio call.
startRemoteView	Subscribe to the video stream of a remote user.
stopRemoteView	Unsubscribe from the video stream of a remote user.

openCamera	Turn on the camera.
closeCamera	Turn off the camera.
switchCamera	Switch camera.
openMicrophone	Enable microphone.
closeMicrophone	Disable the microphone.
selectAudioPlaybackDevice	Select the audio playback device (Earpiece/Speakerphone).
setSelfInfo	Set the user's profile picture and nickname.
enableMultiDeviceAbility	Sets whether to enable multi-device login for TUICallEngine (supported by the Group Call package).
setVideoRenderParams	Set the rendering mode of video.
setVideoEncoderParams	Set the encoding parameters of video encoder.
getTRTCCloudInstance	Advanced features.
setBeautyLevel	Set beauty level, support turning off default beauty.

Details

createInstance

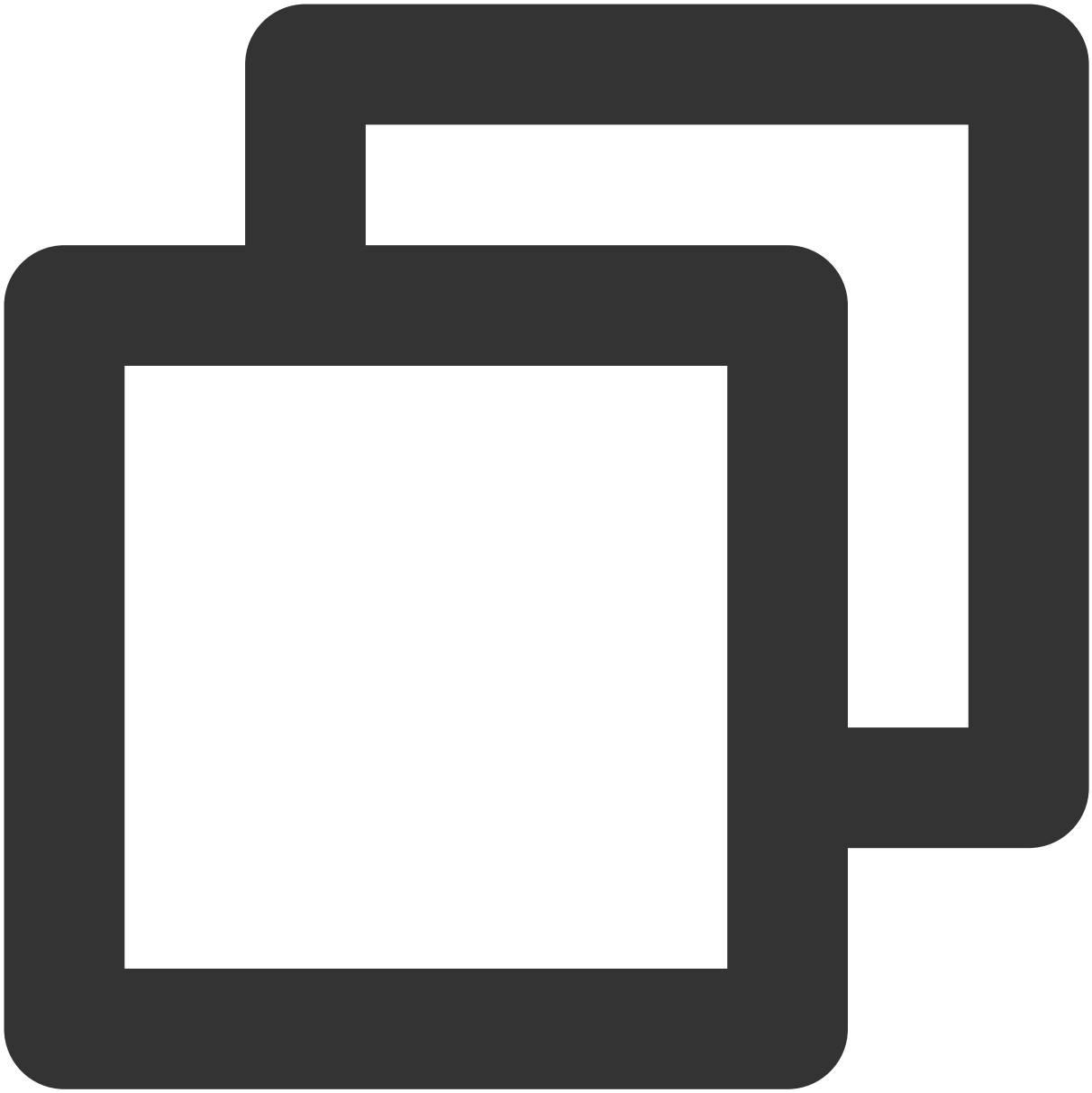
This API is used to create a `TUICallEngine` singleton.



```
- (TUICallEngine *)createInstance;
```

destroyInstance

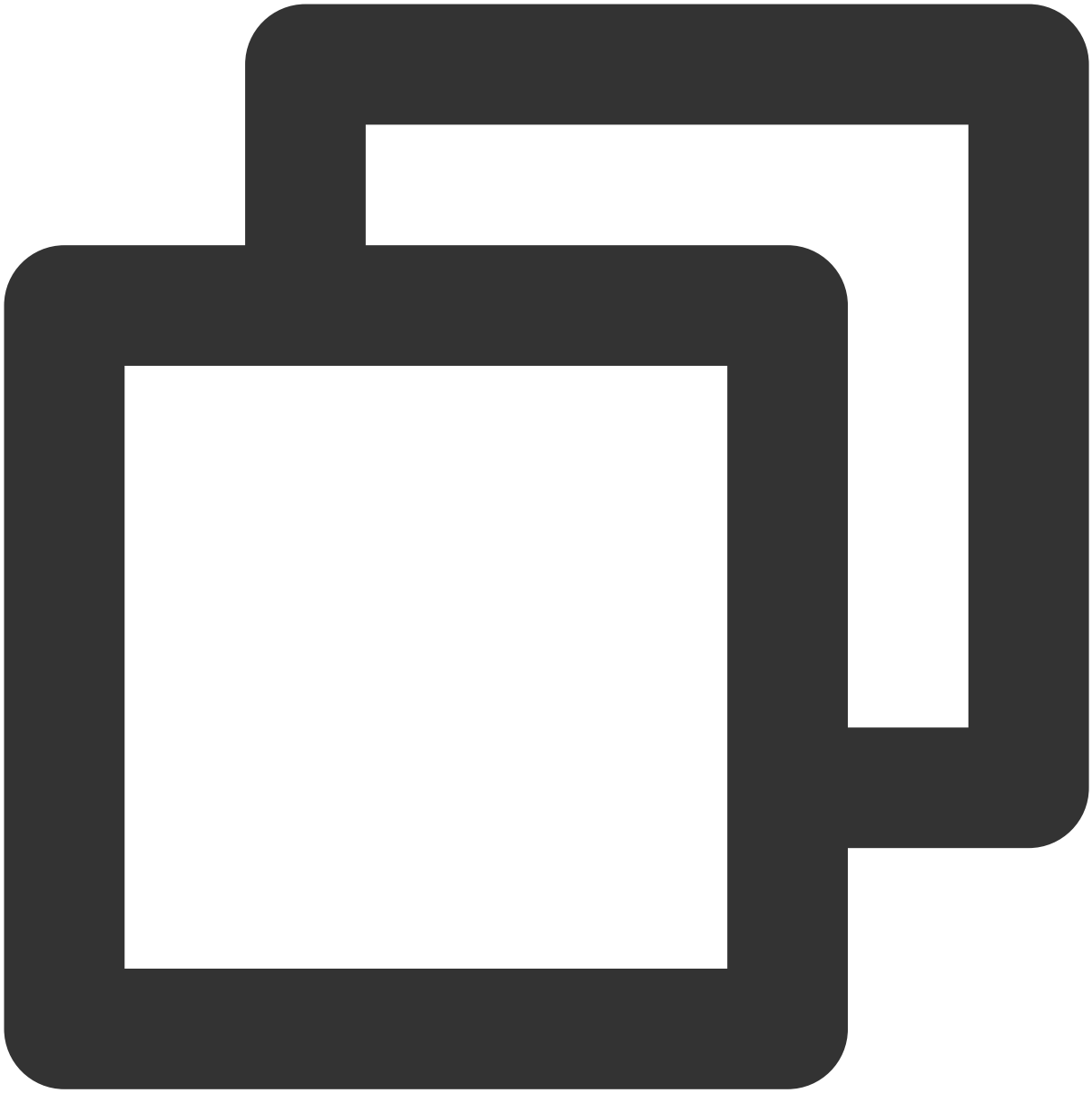
This API is used to Destroy `TUICallEngine` singleton.



```
- (void)destroyInstance;
```

Init

This API is used to initialize `TUICallEngine` . Call it to authenticate the call service and perform other required actions before you call other APIs.



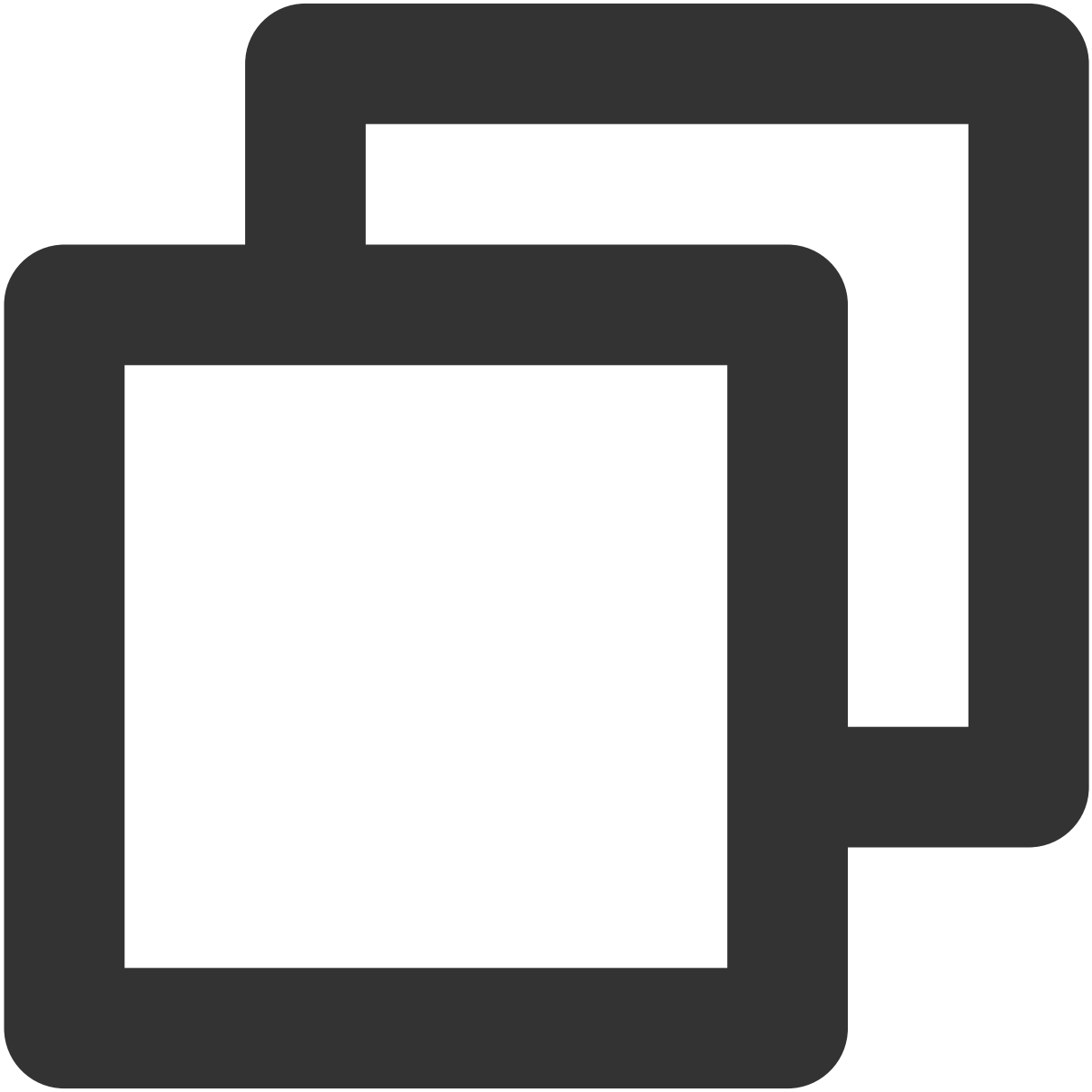
```
- (void)init:(NSString *)sdkAppID userId:(NSString *)userId userSig:(NSString *)use
```

Parameter	Type	Description
sdkAppID	NSString	You can view `SDKAppID` in Application Management > **Application Info** of the TRTC console.
userId	NSString	The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_).
userSig	NSString	Tencent Cloud's proprietary security signature. For how to calculate and

		use it, see UserSig.

addObserver

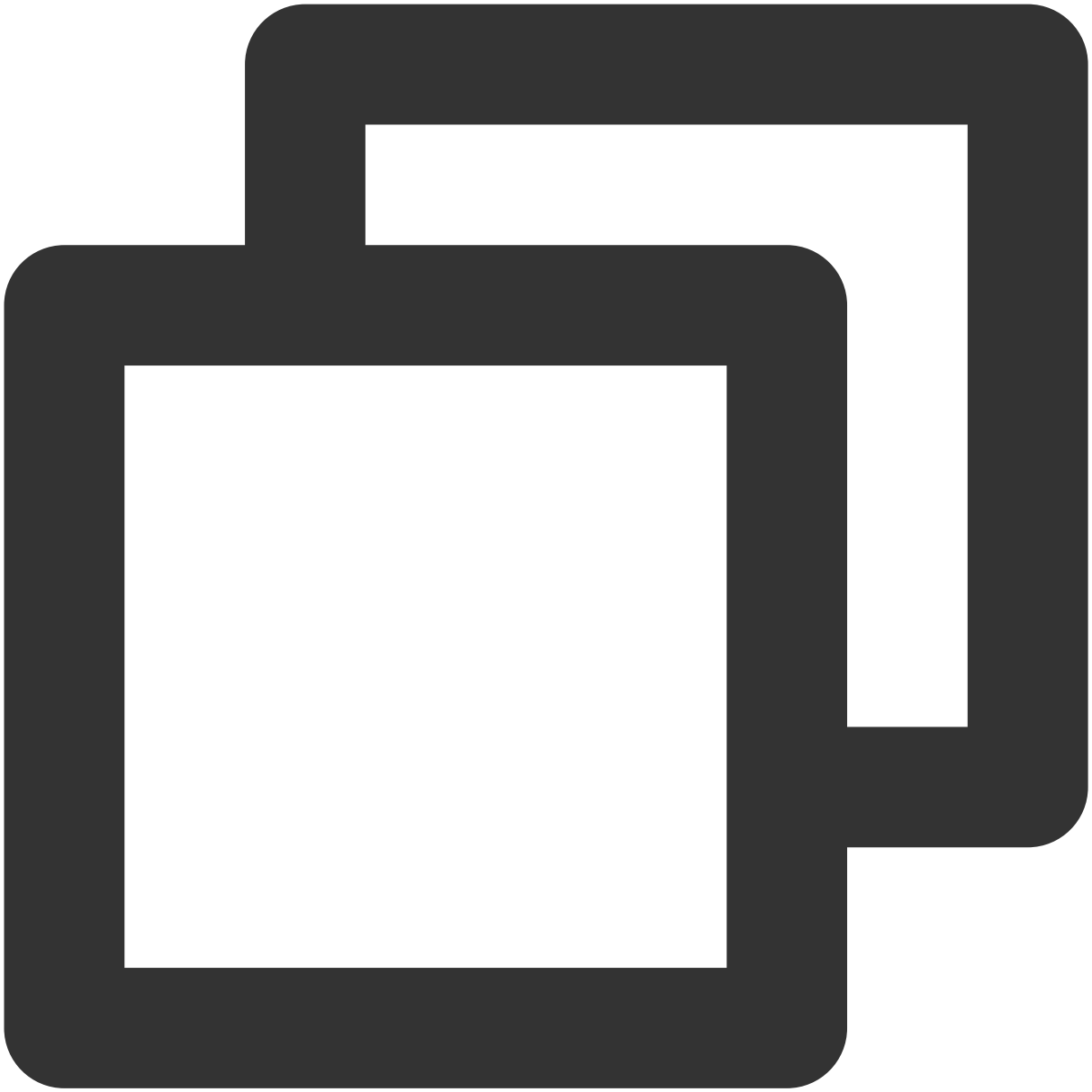
This API is used to register an event listener to listen for `TUICallObserver` events.



```
- (void)addObserver:(id<TUICallObserver>)observer;
```

removeObserver

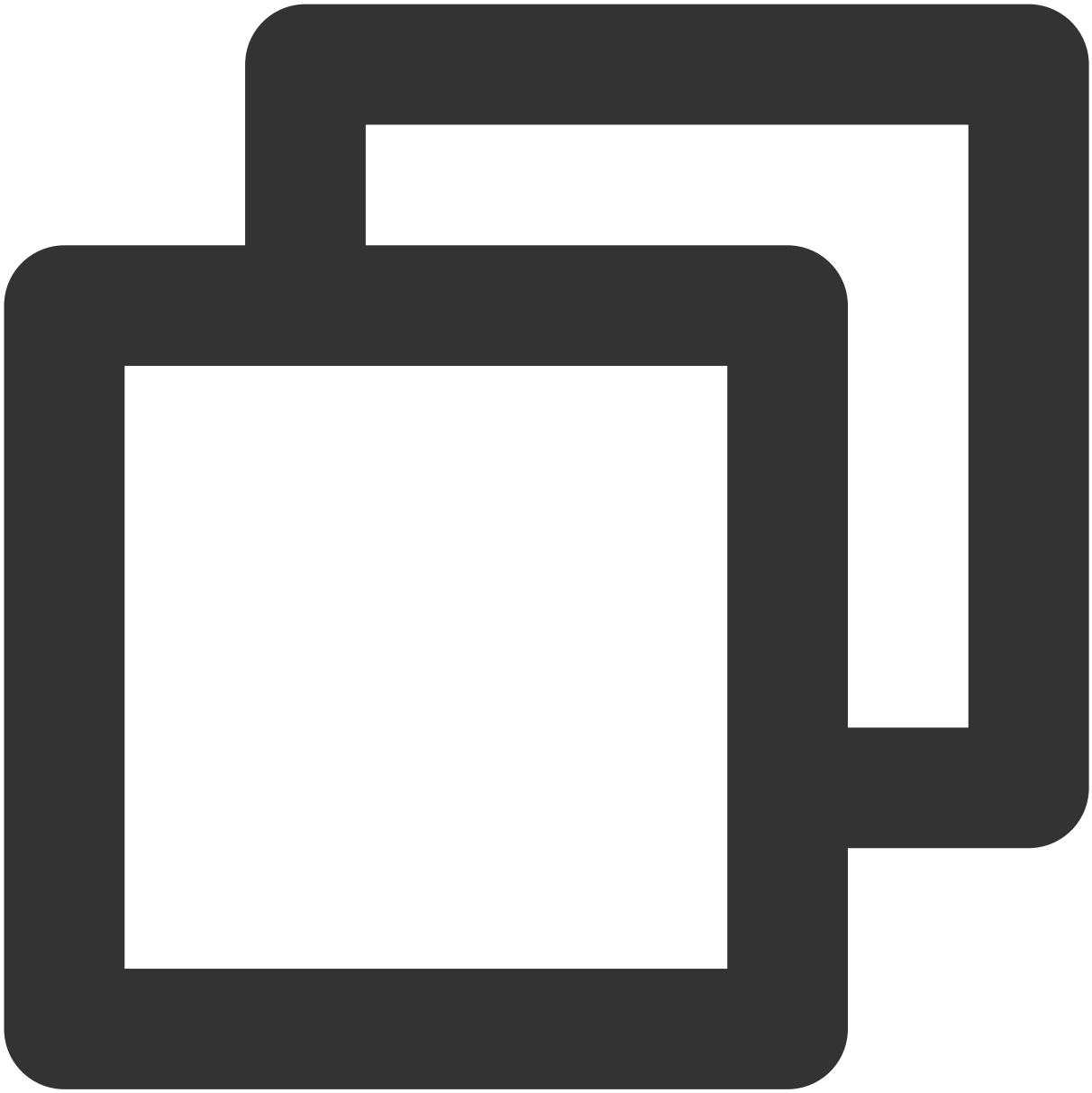
This API is used to unregister an event listener.



```
- (void)removeObserver:(id<TUICallObserver>)observer;
```

call

This API is used to make a (one-to-one) call.



```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType param
```

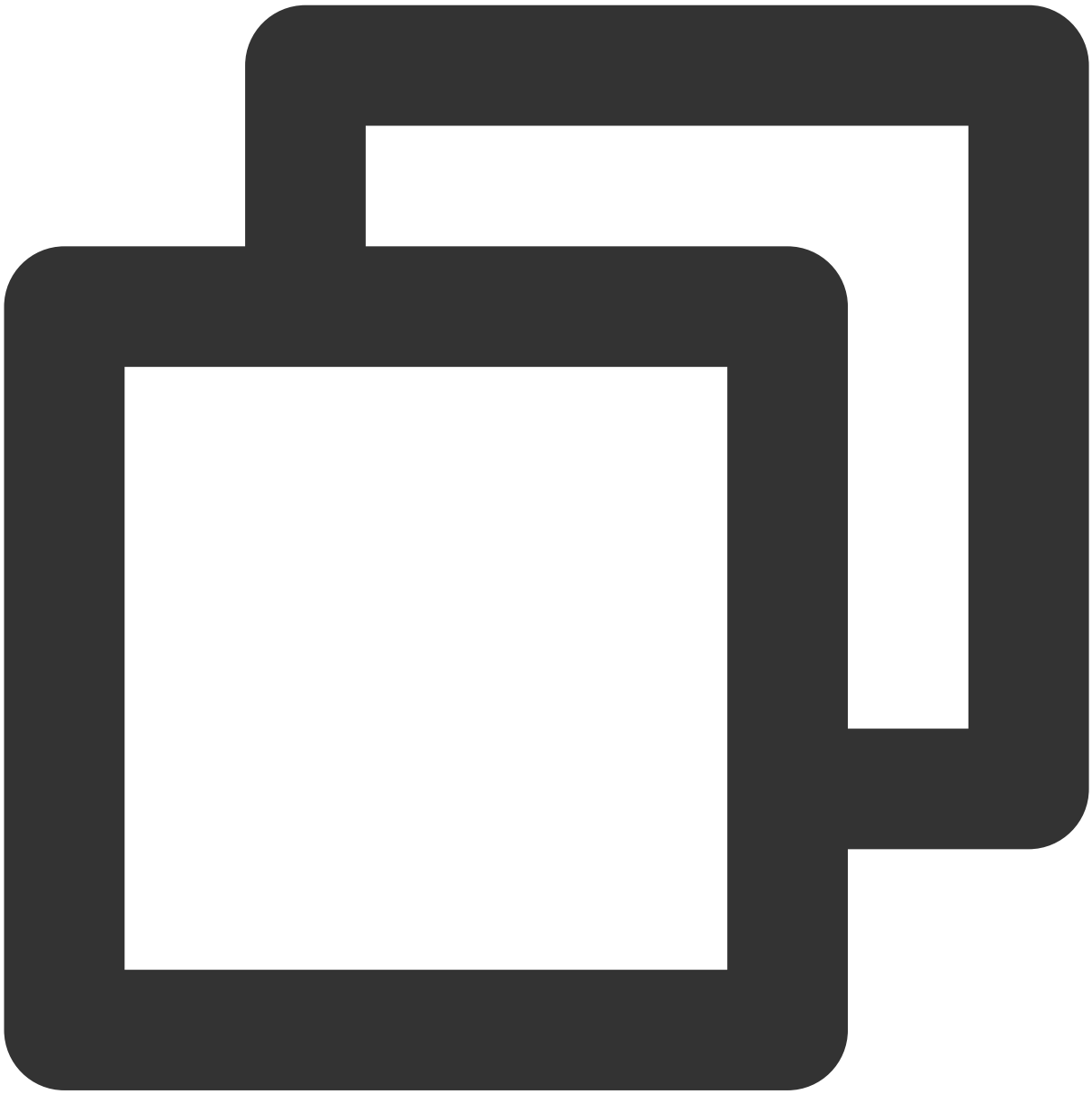
Parameter	Type	Description
userId	NSString	The target user ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.
params	TUICallParams	An additional parameter, such as roomID, call timeout, offline push info, etc

groupCall

This API is used to make a group call.

Note：

Before making a group call, you need to create an Chat group first.



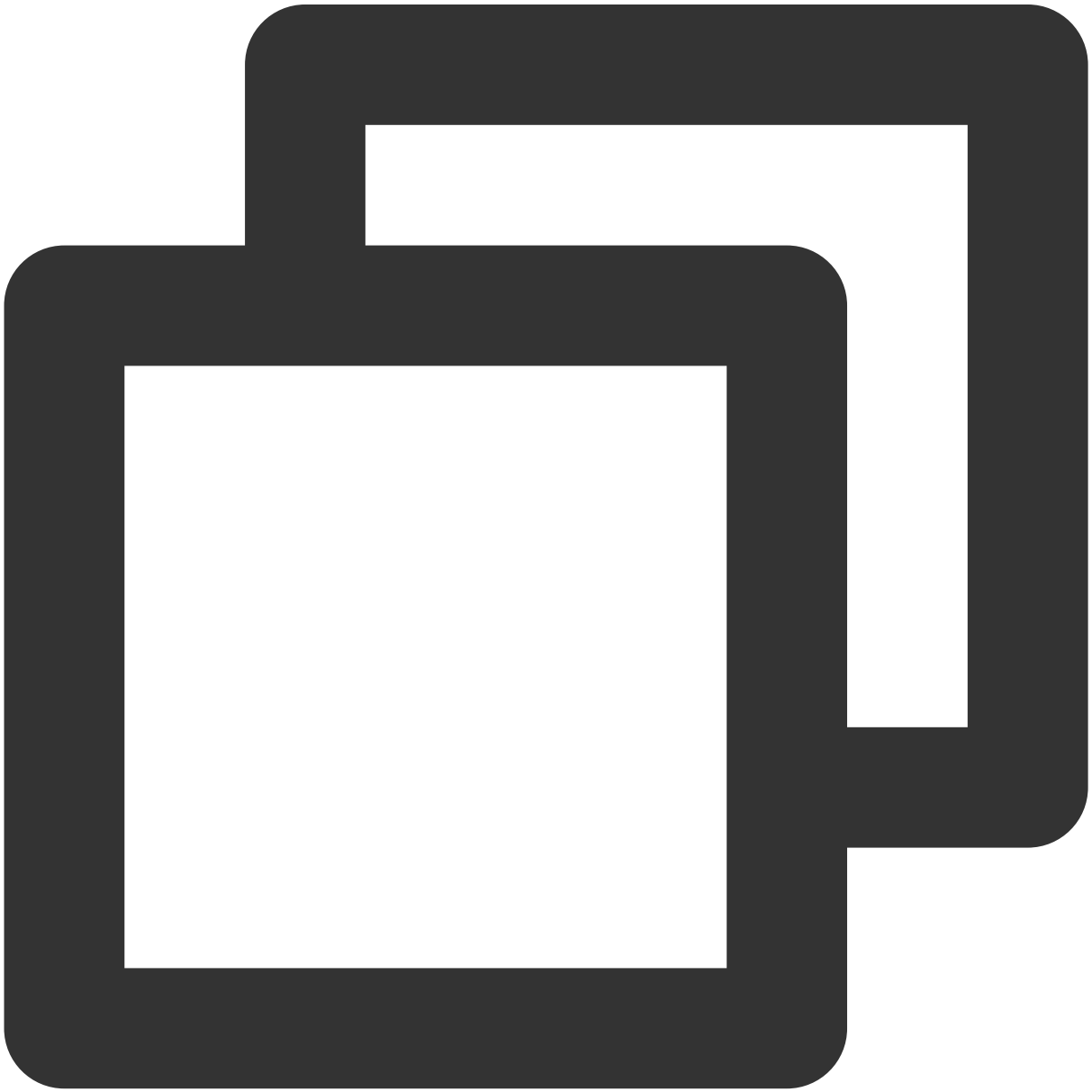
```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray <NSString *> *)userIdList
```

Parameter	Type	Description
groupId	NSString	The group ID.

userIdList	NSArray	The target user IDs.
callMediaType	TUICallMediaType	The call type, which can be video or audio.
params	TUICallParams	An additional parameter. such as roomID, call timeout, offline push info, etc

accept

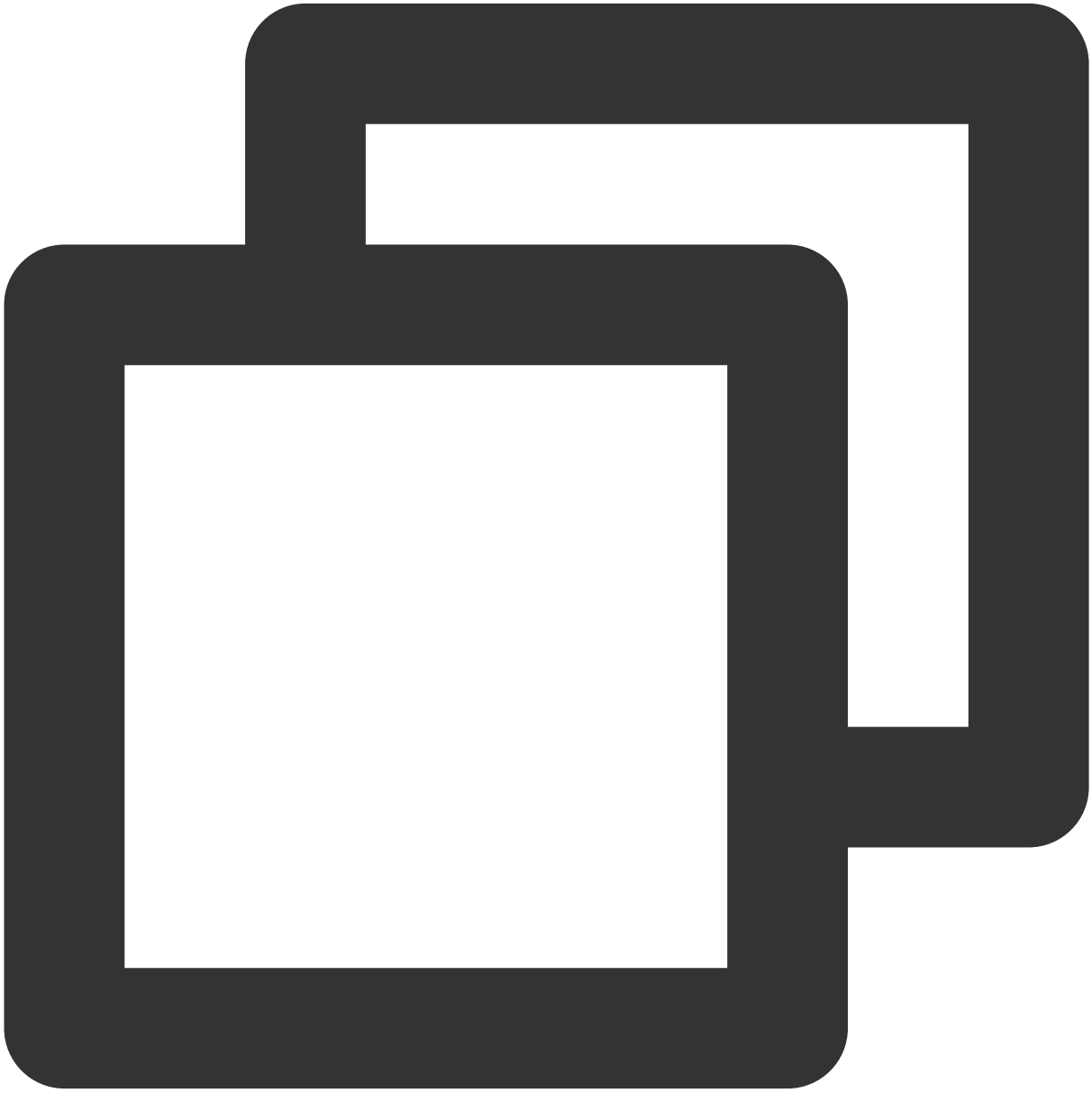
This API is used to accept a call. After receiving the `onCallReceived()` callback, you can call this API to accept the call.



```
- (void)accept:(TUICallSucc)succ fail:(TUICallFail)fail;
```

reject

This API is used to reject a call. After receiving the `onCallReceived()` callback, you can call this API to reject the call.

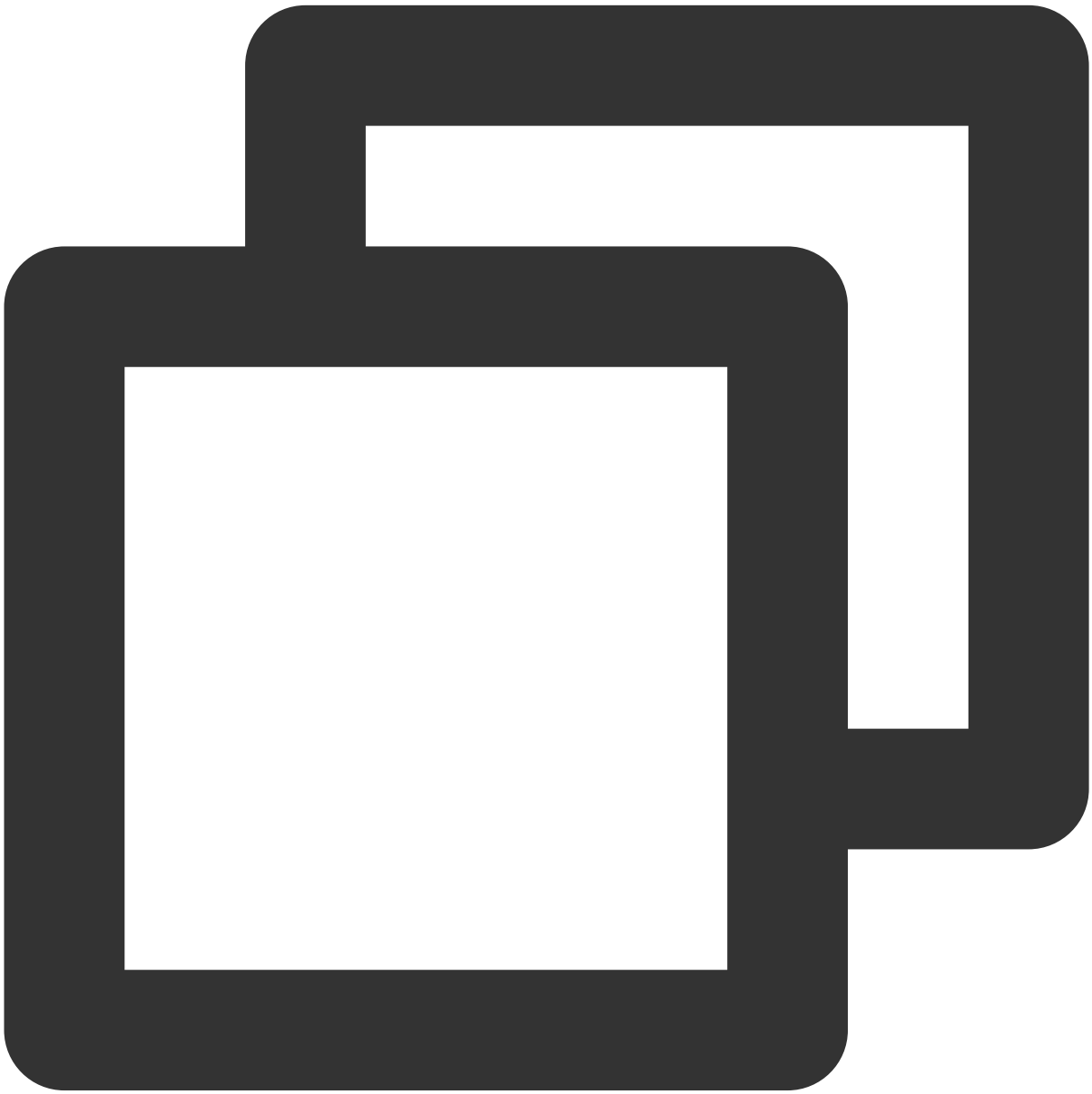


```
- (void)reject:(TUICallSucc)succ fail:(TUICallFail)fail;
```

ignore

This API is used to ignore a call. After receiving the `onCallReceived()` , you can call this API to ignore the call. The caller will receive the `onUserLineBusy` callback.

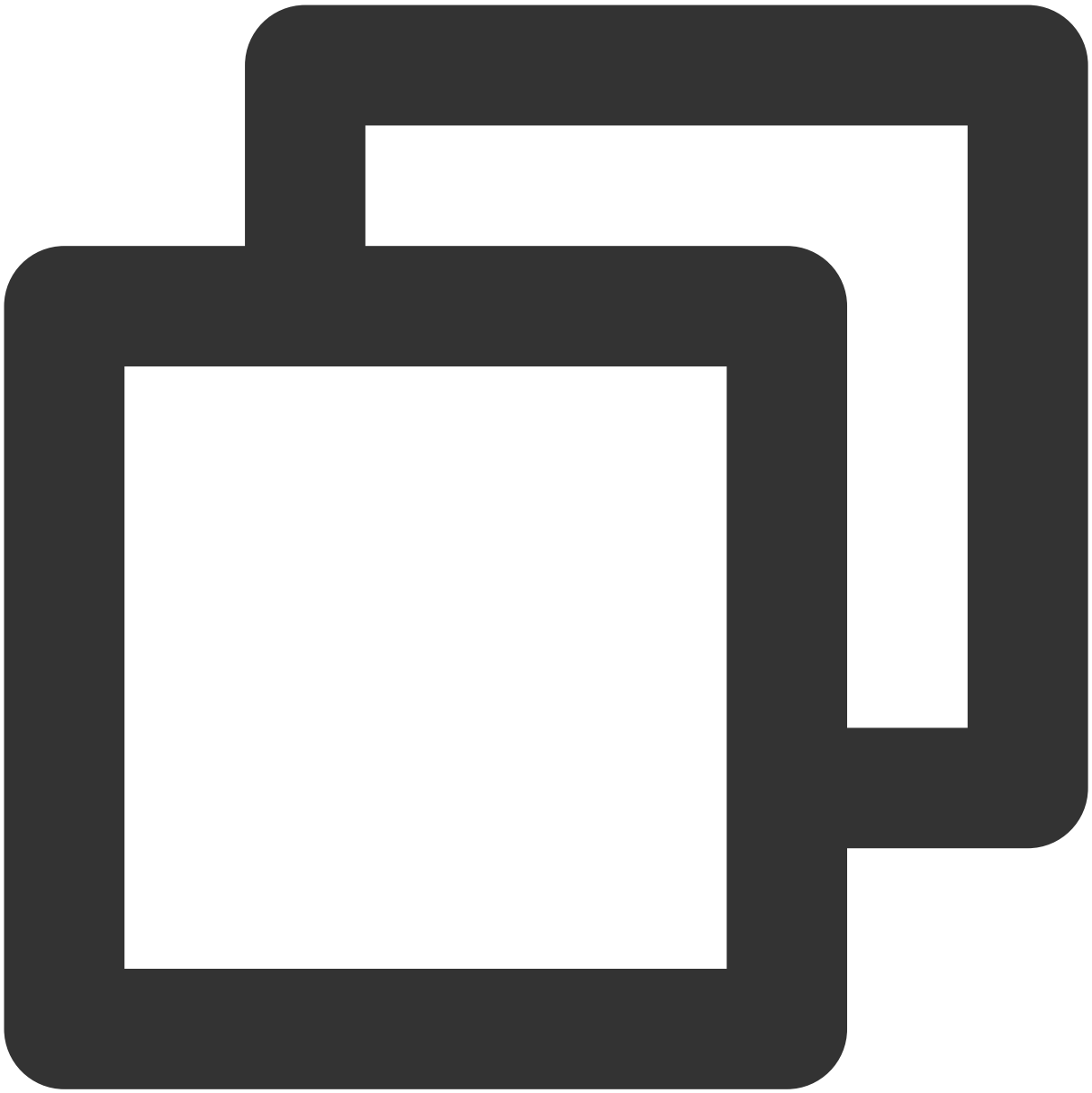
Note: If your project involves live streaming or conferencing, you can also use this API to implement the “in a meeting” or “on air” feature.



```
- (void)ignore:(TUICallSucc)succ fail:(TUICallFail)fail;
```

hangup

This API is used to end a call.

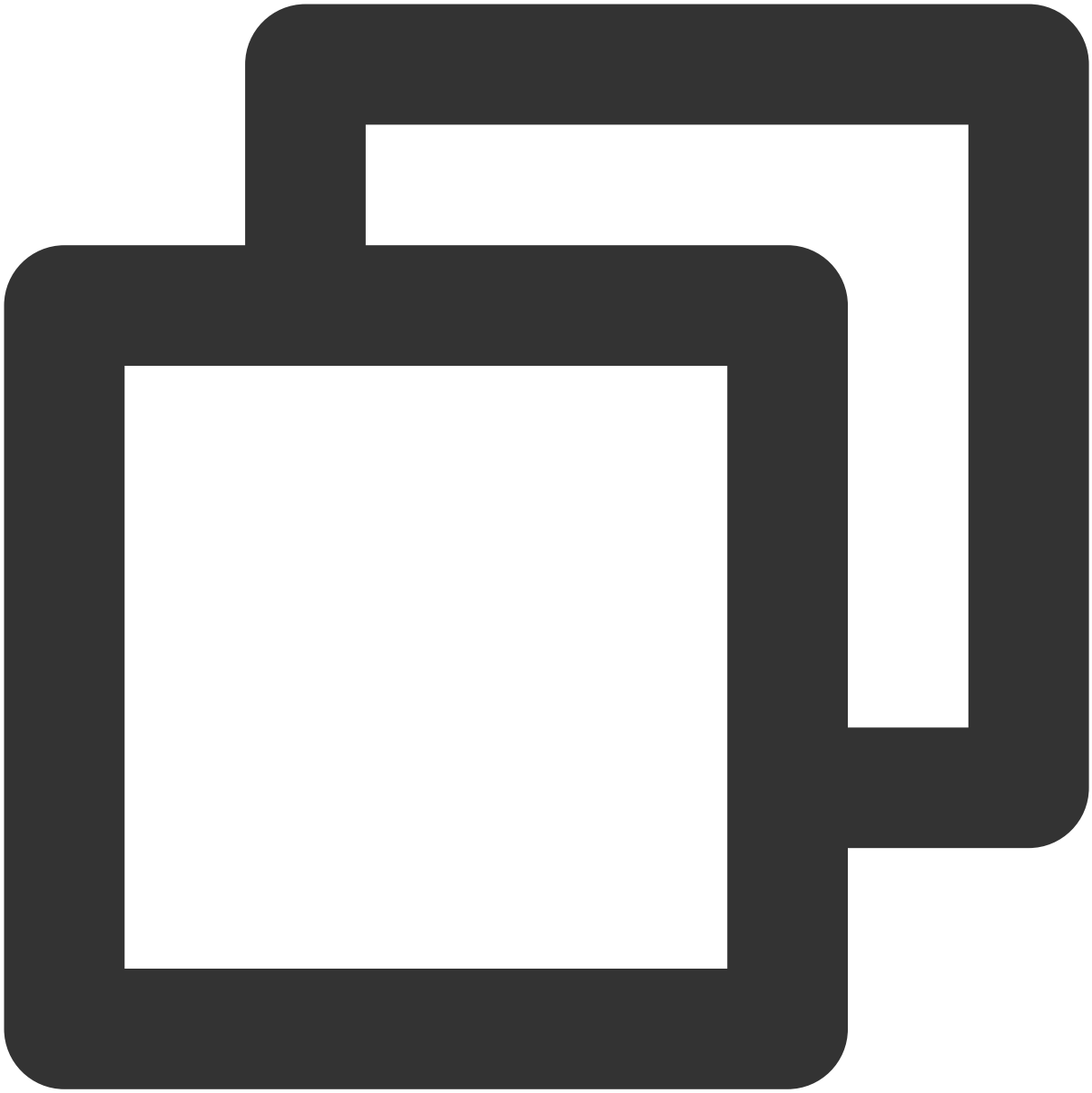


```
- (void)hangup:(TUICallSucc)succ fail:(TUICallFail)fail;
```

inviteUser

This API is used to invite users to the current group call.

This API is called by a participant of a group call to invite new users.



```
- (void)inviteUser:(NSArray<NSString *> *)userIdList params:(TUICallParams *)params
```

Parameter	Type	Description
userIdList	NSArray	The target user IDs.
params	TUICallParams	An additional parameter. such as roomID, call timeout, offline push info, etc

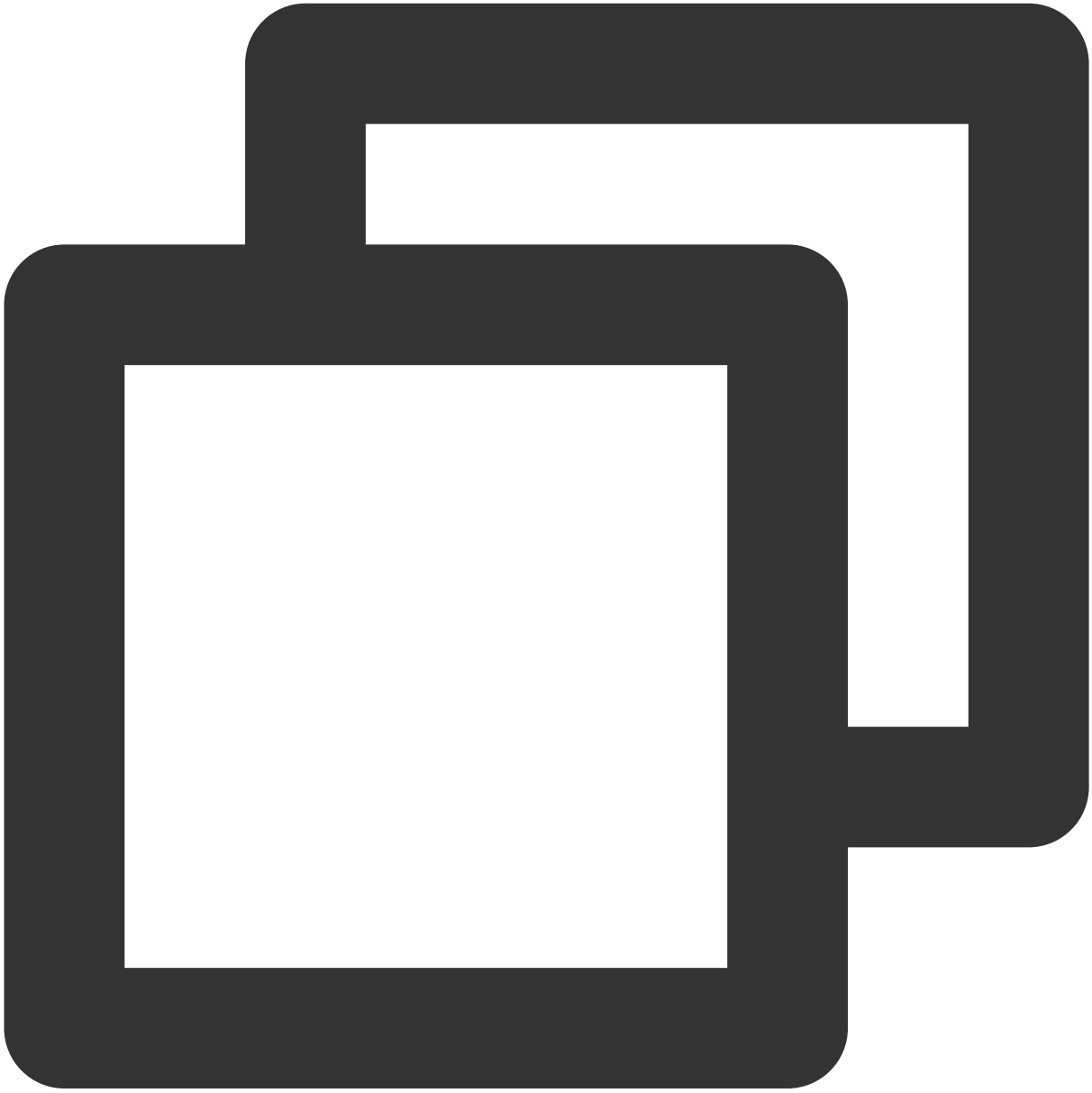
Note：

In this case, the custom RoomId is invalid. The SDK will invite others to join the room where the inviter is currently located.

joinInGroupCall

This API is used to join a group call.

This API is called by a group member to join the group's call.



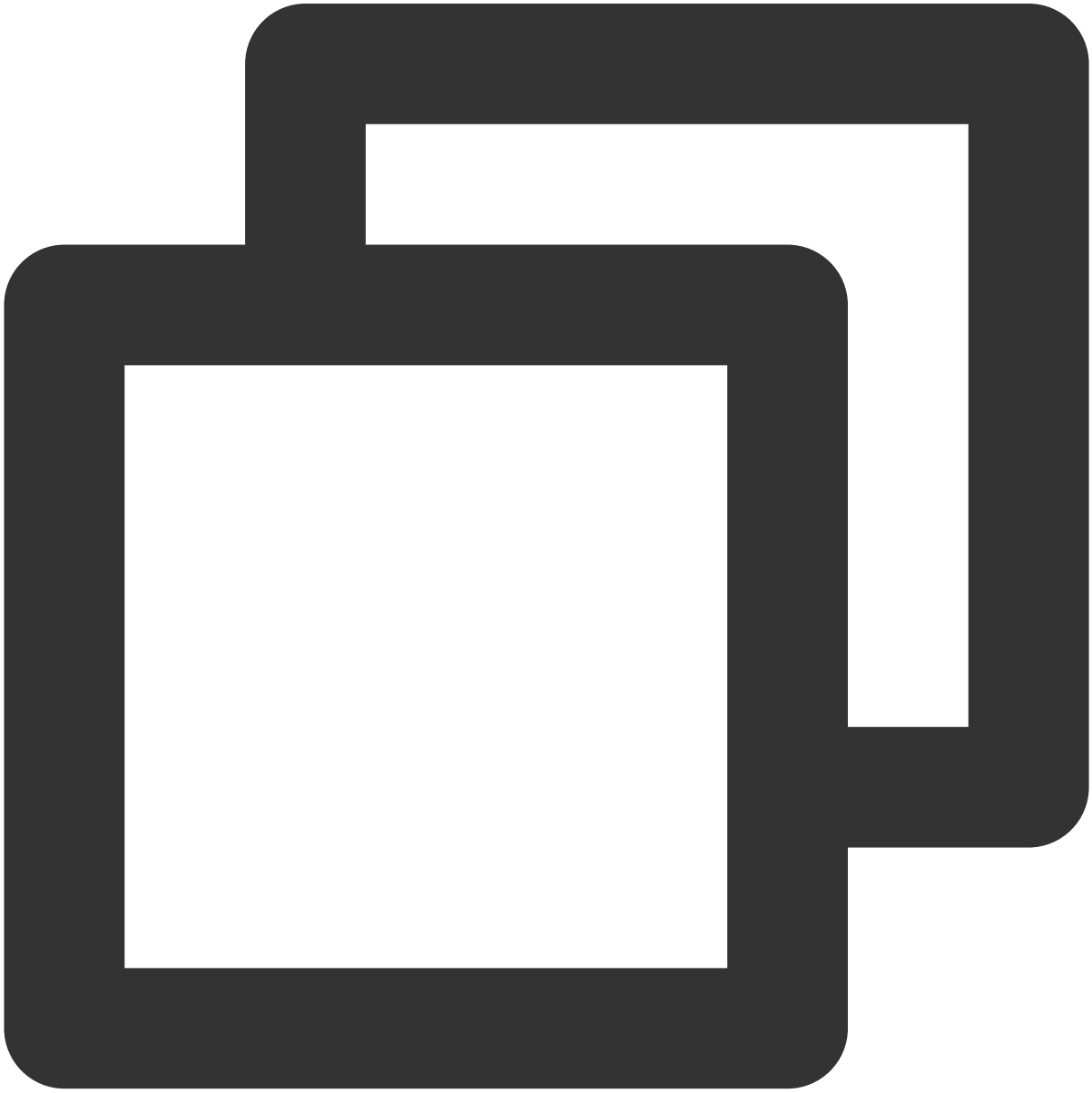
```
- (void)joinInGroupCall:(TUIRoomId *)roomId groupId:(NSString *)groupId callMediaTy
```

Parameter	Type	Description

roomId	TUIRoomId	The room ID.
groupId	NSString	The group ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.

switchCallMediaType

This API is used to change the call type.

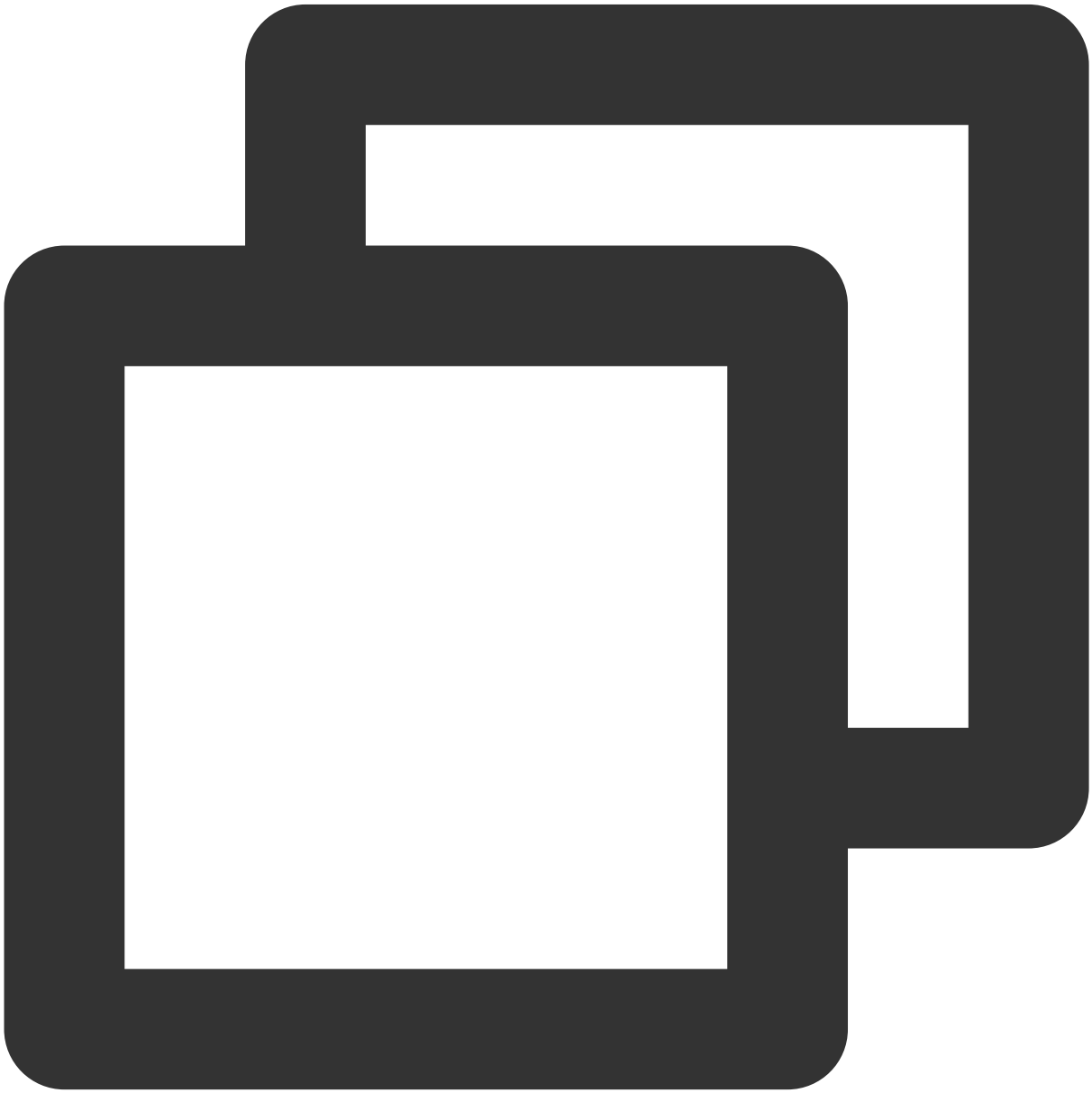


```
- (void)switchCallMediaType:(TUICallMediaType)newType;
```

Parameter	Type	Description
callMediaType	TUICallMediaType	The call type, which can be video or audio.

startRemoteView

This API is used to set the view object to display a remote video.

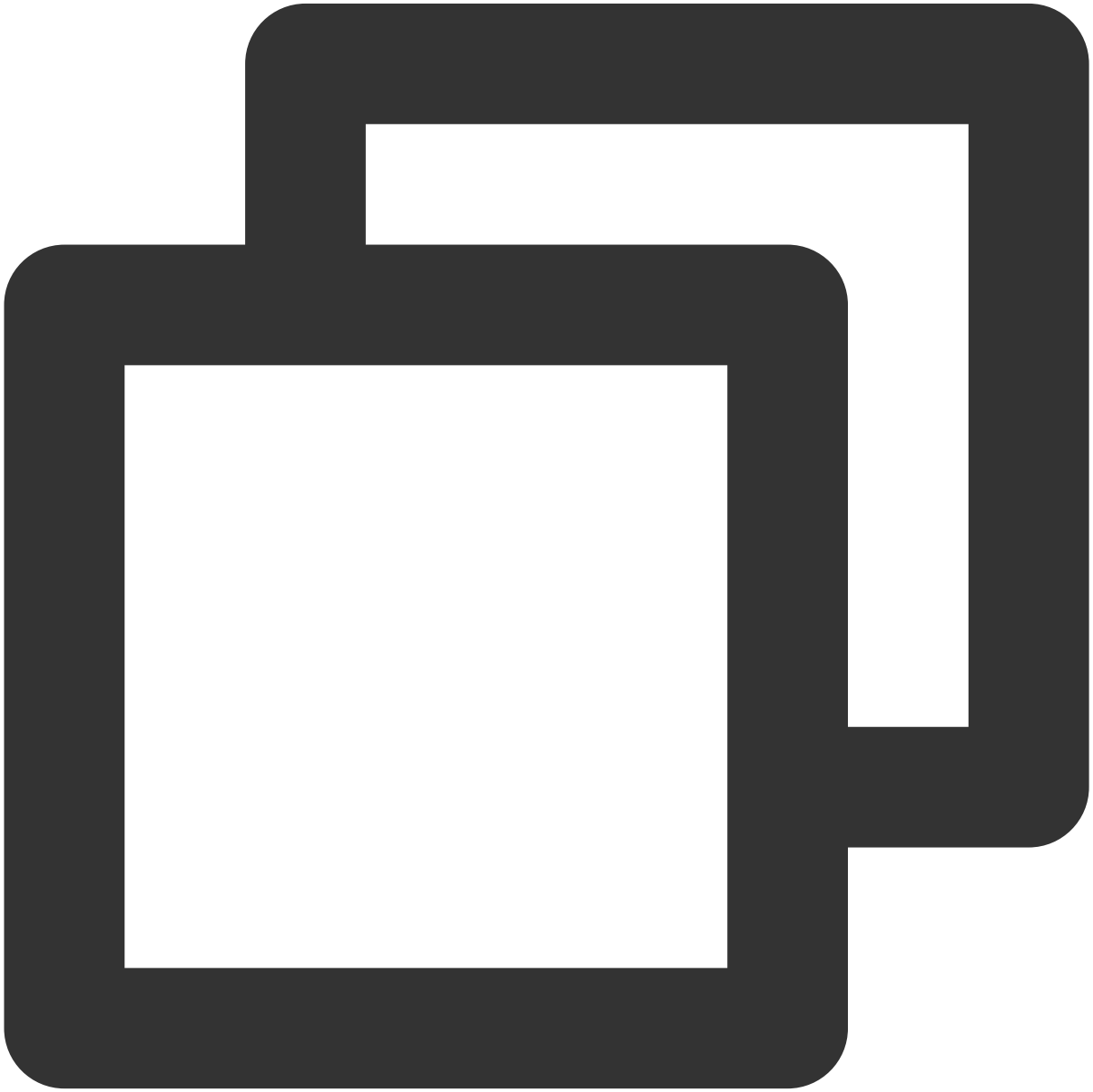


```
- (void)startRemoteView:(NSString *)userId videoView:(TUIVideoView *)videoView onPl
```


Parameter	Type	Description
userId	NSString	The target user ID.
videoView	TUIVideoView	The view to be rendered.

stopRemoteView

This API is used to unsubscribe from the video stream of a remote user.

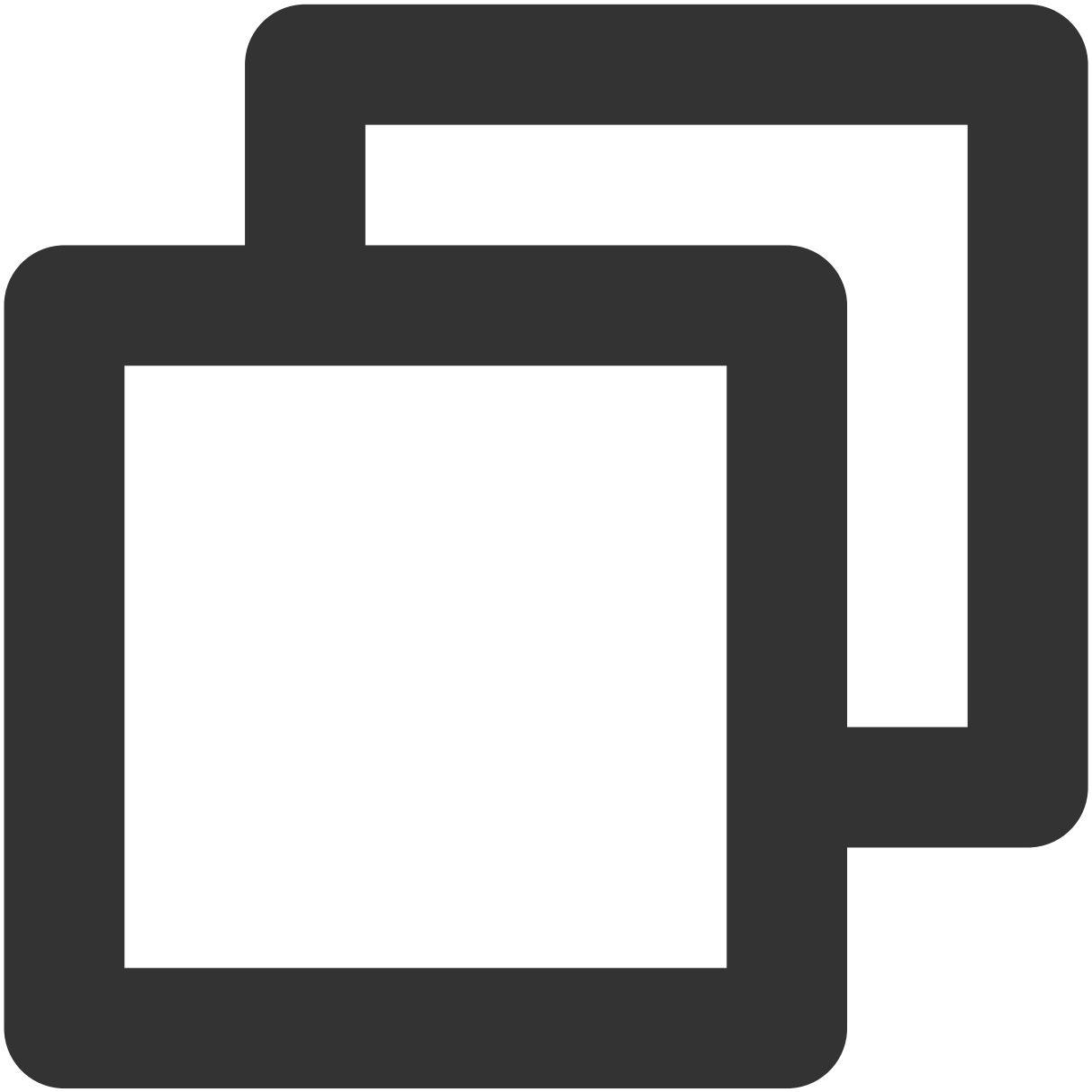


```
- (void)stopRemoteView:(NSString *)userId;
```

Parameter	Type	Description
userId	NSString	The target user ID.

openCamera

This API is used to turn the camera on.



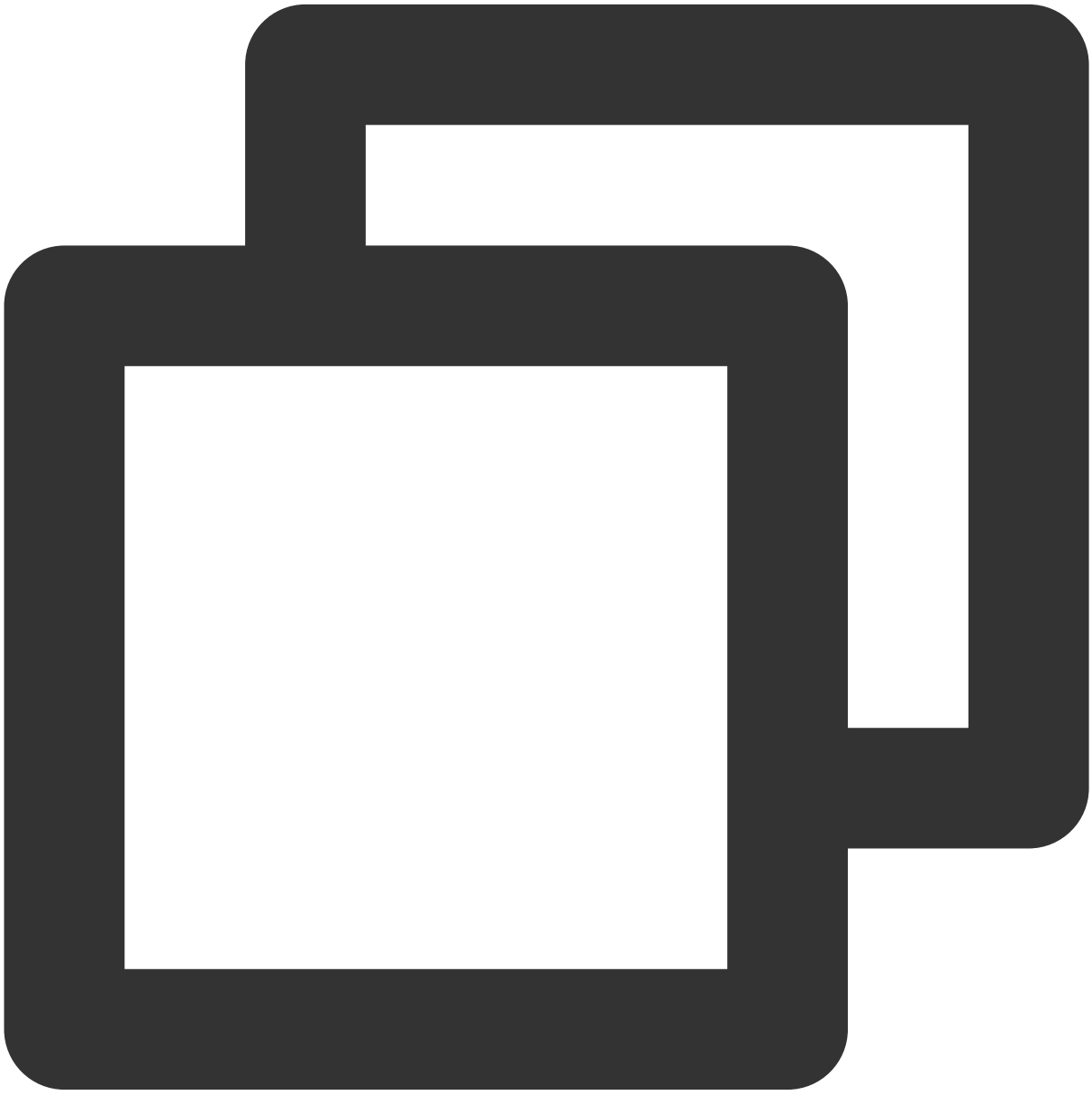
```
- (void)openCamera:(TUICamera)camera videoView:(TUIVideoView *)videoView succ:(TUIC
```

Parameter	Type	Description

camera	TUICamera	The front or rear camera.
videoView	TUIVideoView	The view to be rendered.

closeCamera

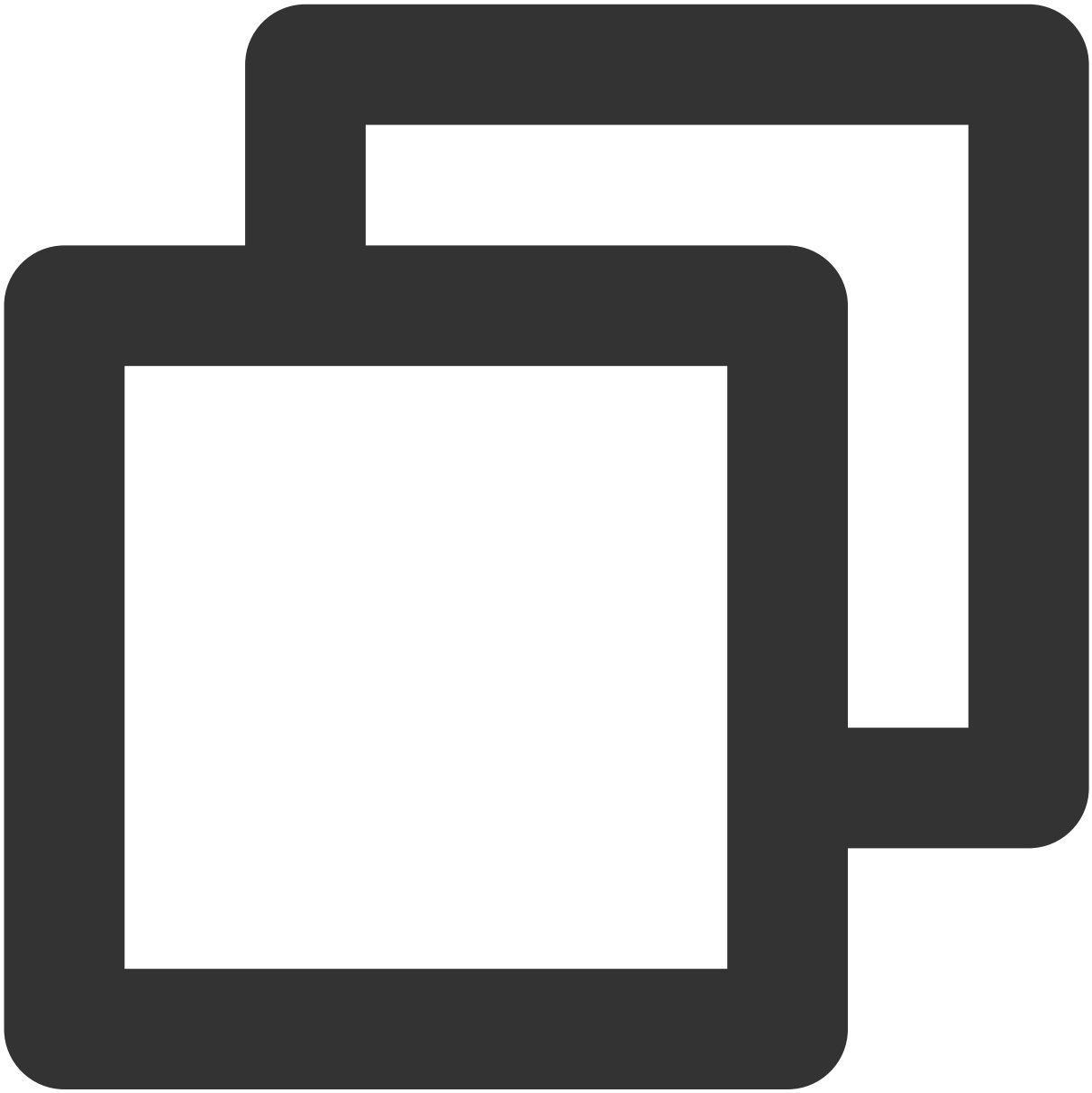
This API is used to turn the camera off.



```
- (void)closeCamera;
```

switchCamera

This API is used to switch between the front and rear cameras.

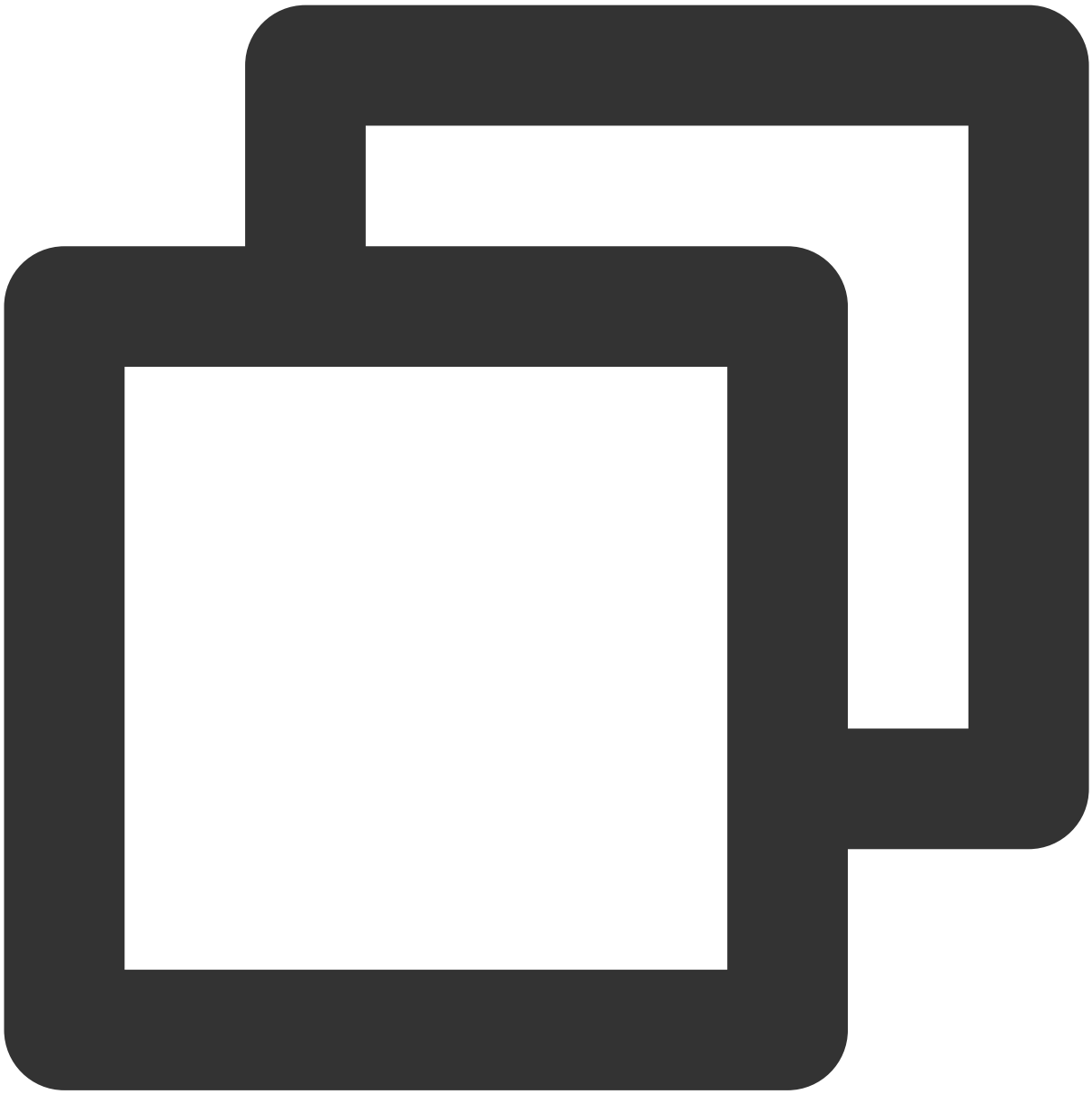


```
- (void)switchCamera:(TUICamera)camera;
```

Parameter	Type	Description
camera	TUICamera	The front or rear camera.

openMicrophone

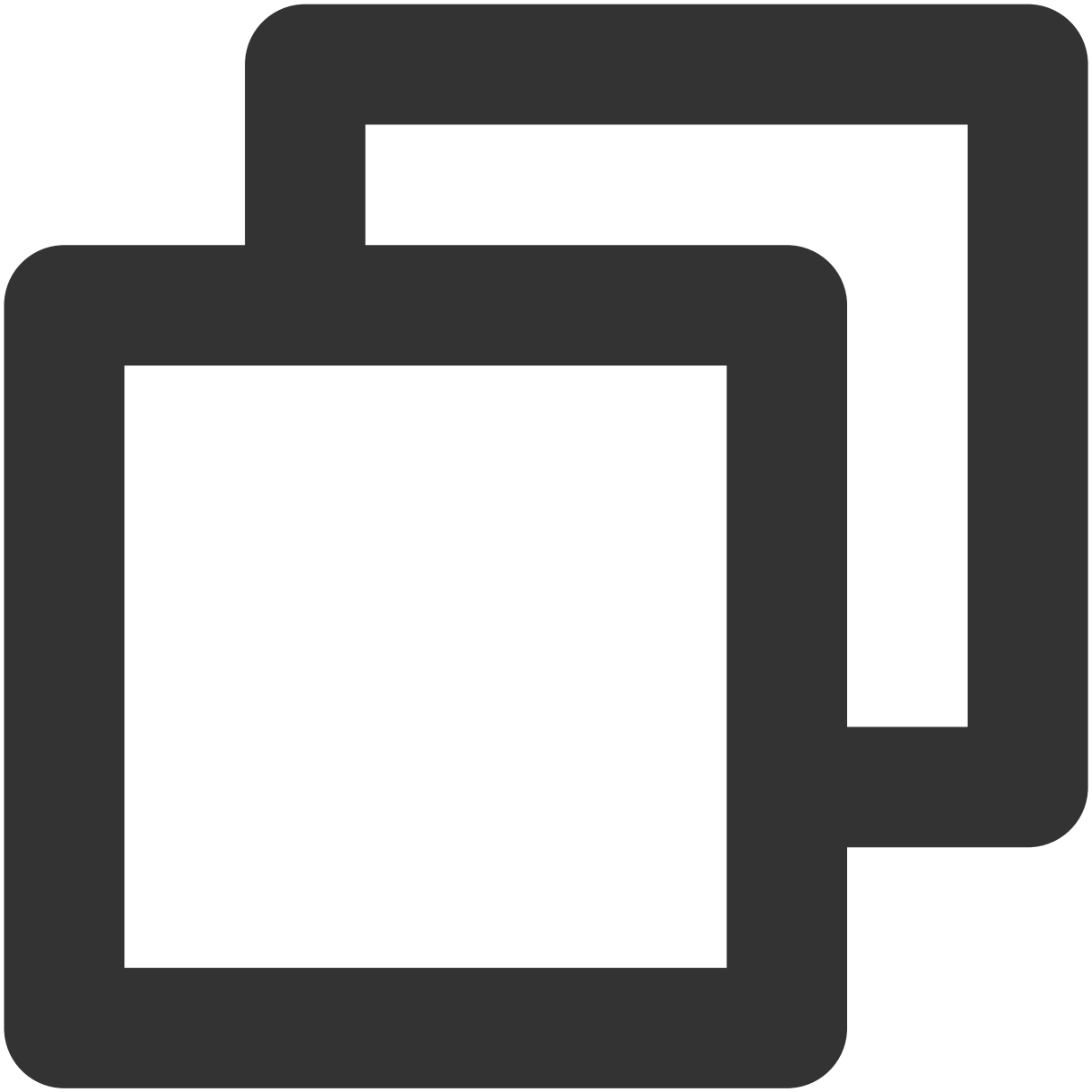
This API is used to turn the microphone on.



```
- (void)openMicrophone:(TUICallSucc)succ fail:(TUICallFail)fail;
```

closeMicrophone

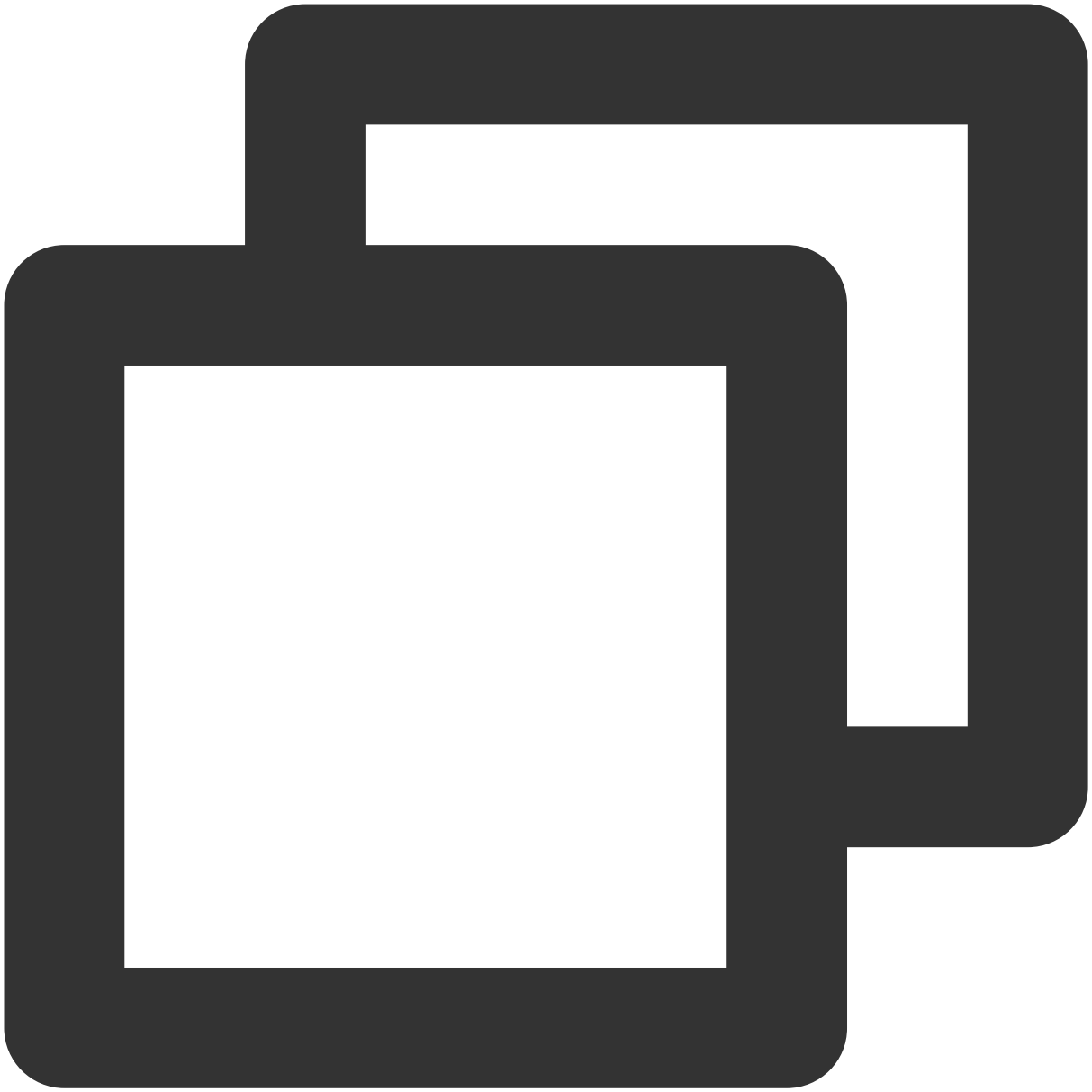
This API is used to turn the microphone off.



```
- (void)closeMicrophone;
```

selectAudioPlaybackDevice

This API is used to select the audio playback device (Earpiece and Speakerphone). In call scenarios, you can use this API to turn on/off hands-free mode.

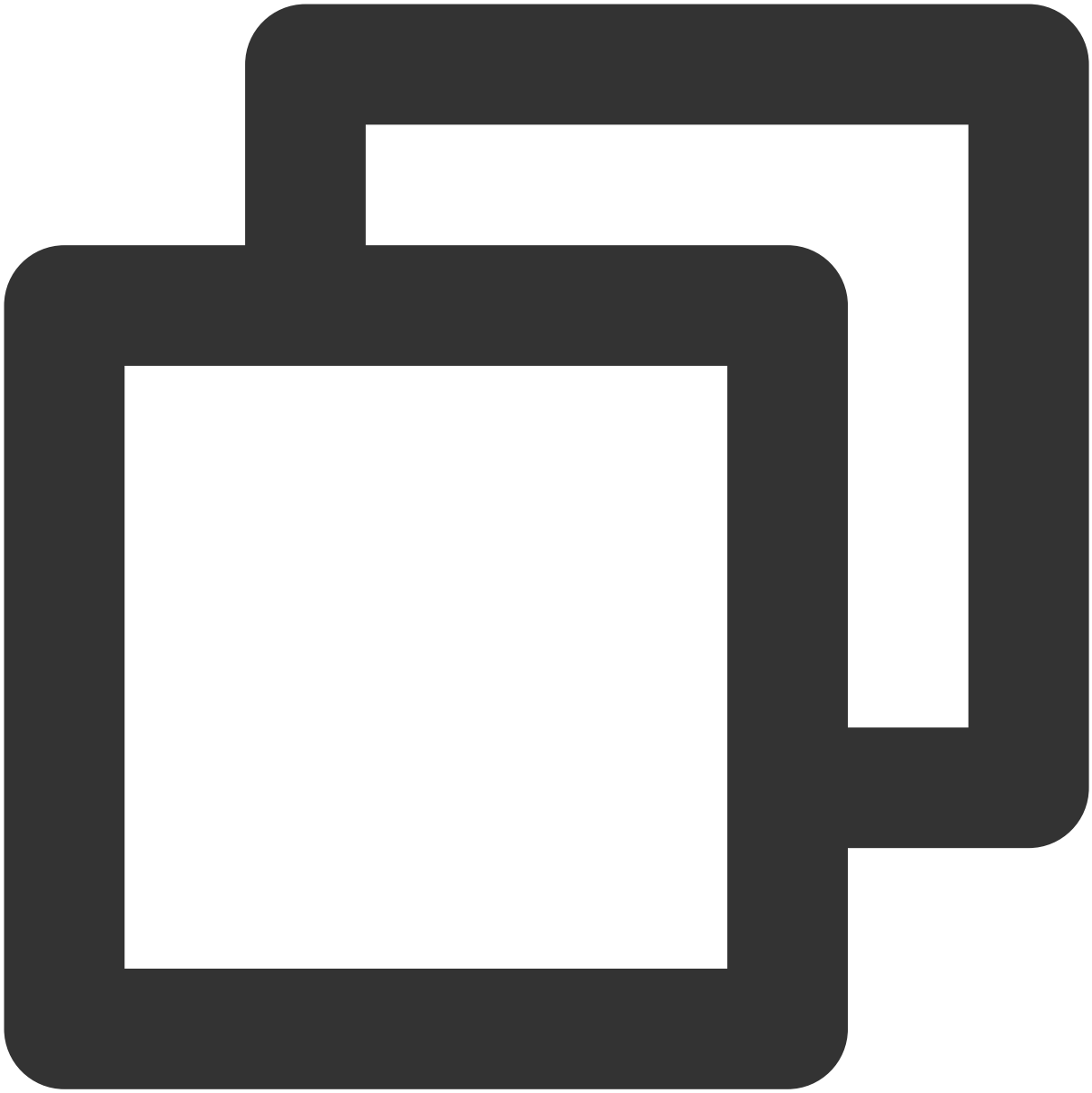


```
- (void)selectAudioPlaybackDevice:(TUIAudioPlaybackDevice)device;
```

Parameter	Type	Description
device	TUIAudioPlaybackDevice	The Earpiece and Speakerphone.

setSelfInfo

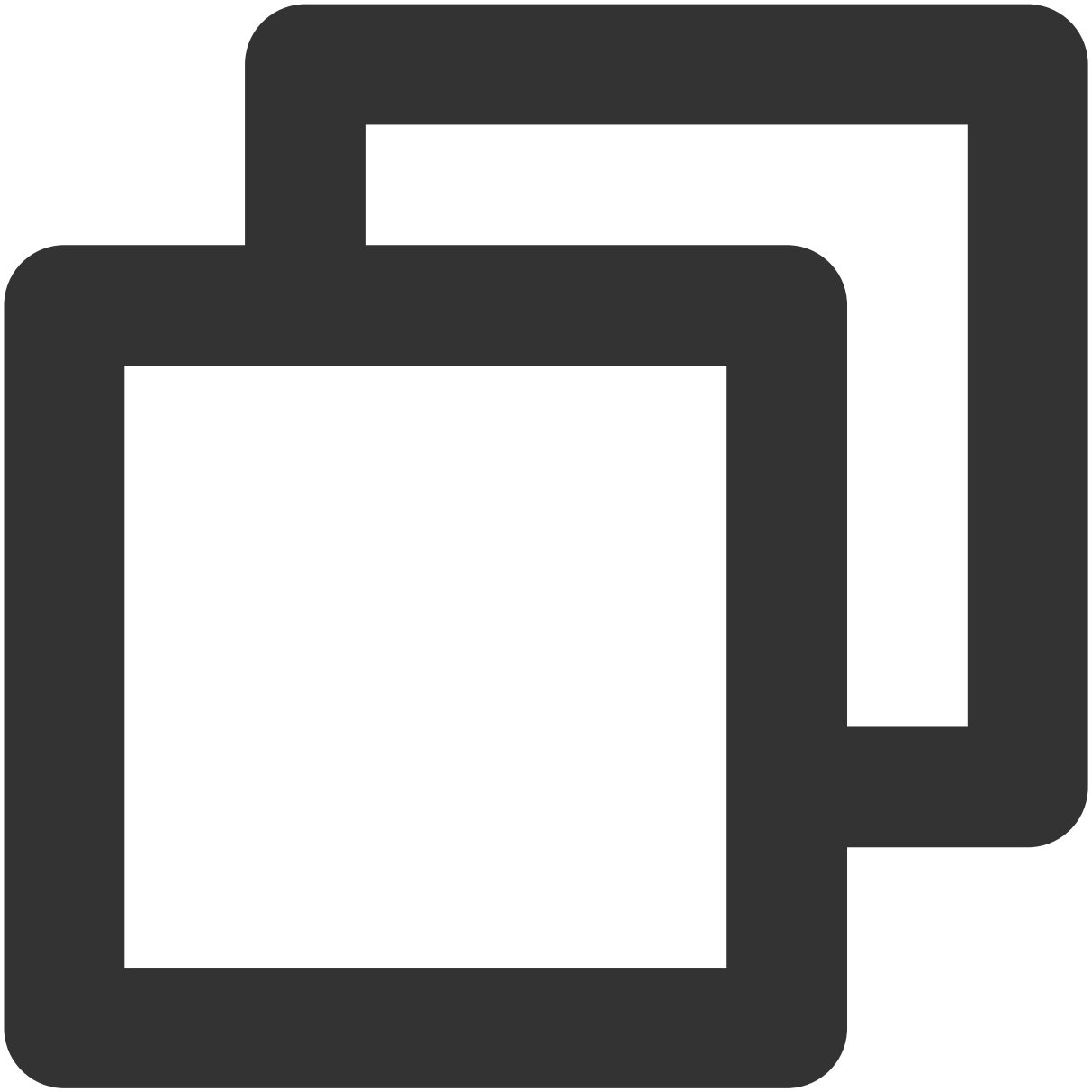
This API is used to set the nickname and profile picture. The nickname cannot exceed 500 bytes, and the profile picture is specified by a URL.



```
- (void)setSelfInfo:(NSString * _Nullable)nickName avatar:(NSString * _Nullable)ava
```

enableMultiDeviceAbility

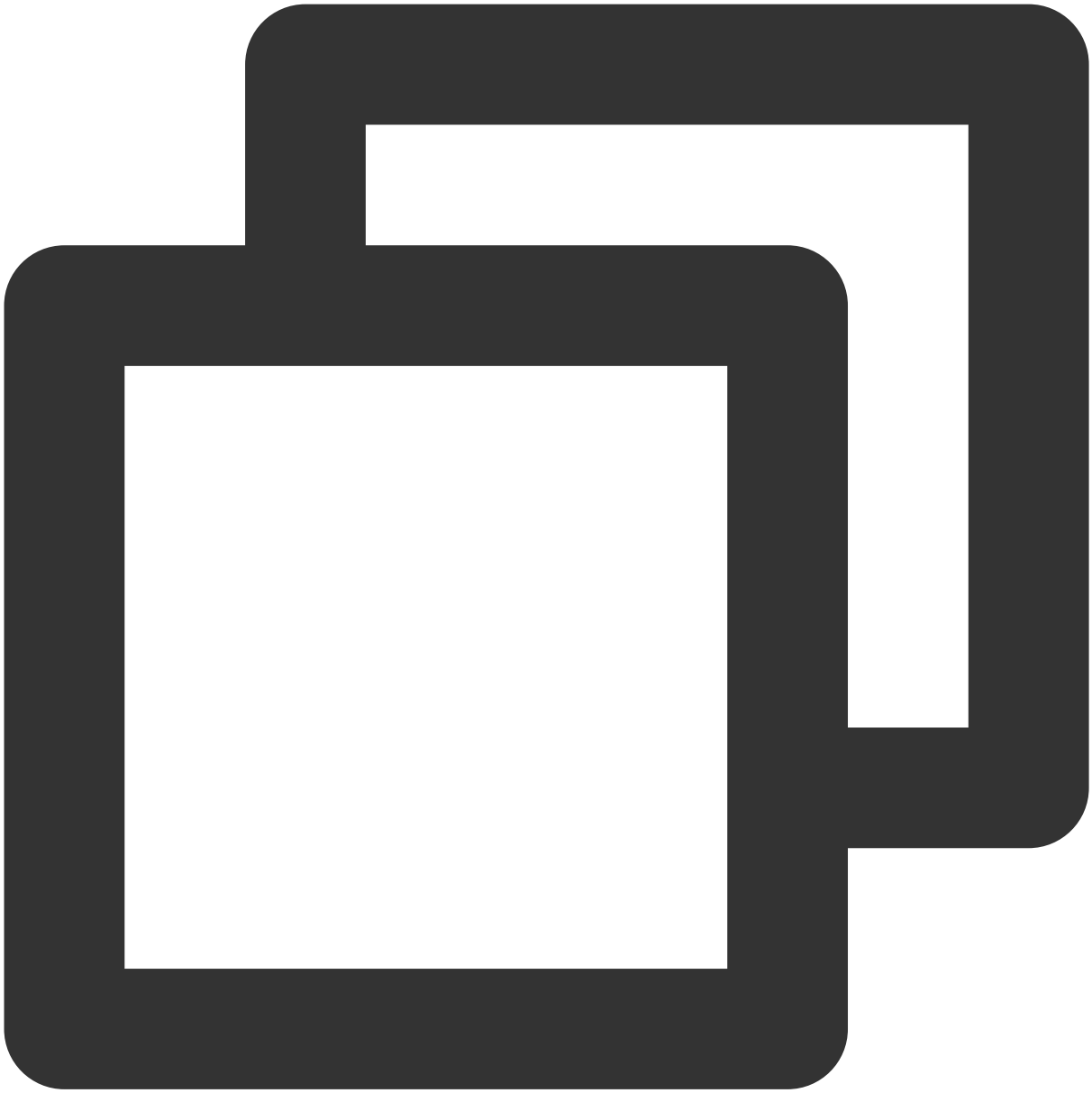
This API is used to set whether to enable multi-device login for `TUICallEngine` (supported by the Group Call package).



```
- (void)enableMultiDeviceAbility:(BOOL)enable succ:(TUICallSucc)succ fail:(TUICallF
```

setVideoRenderParams

Set the rendering mode of video image.

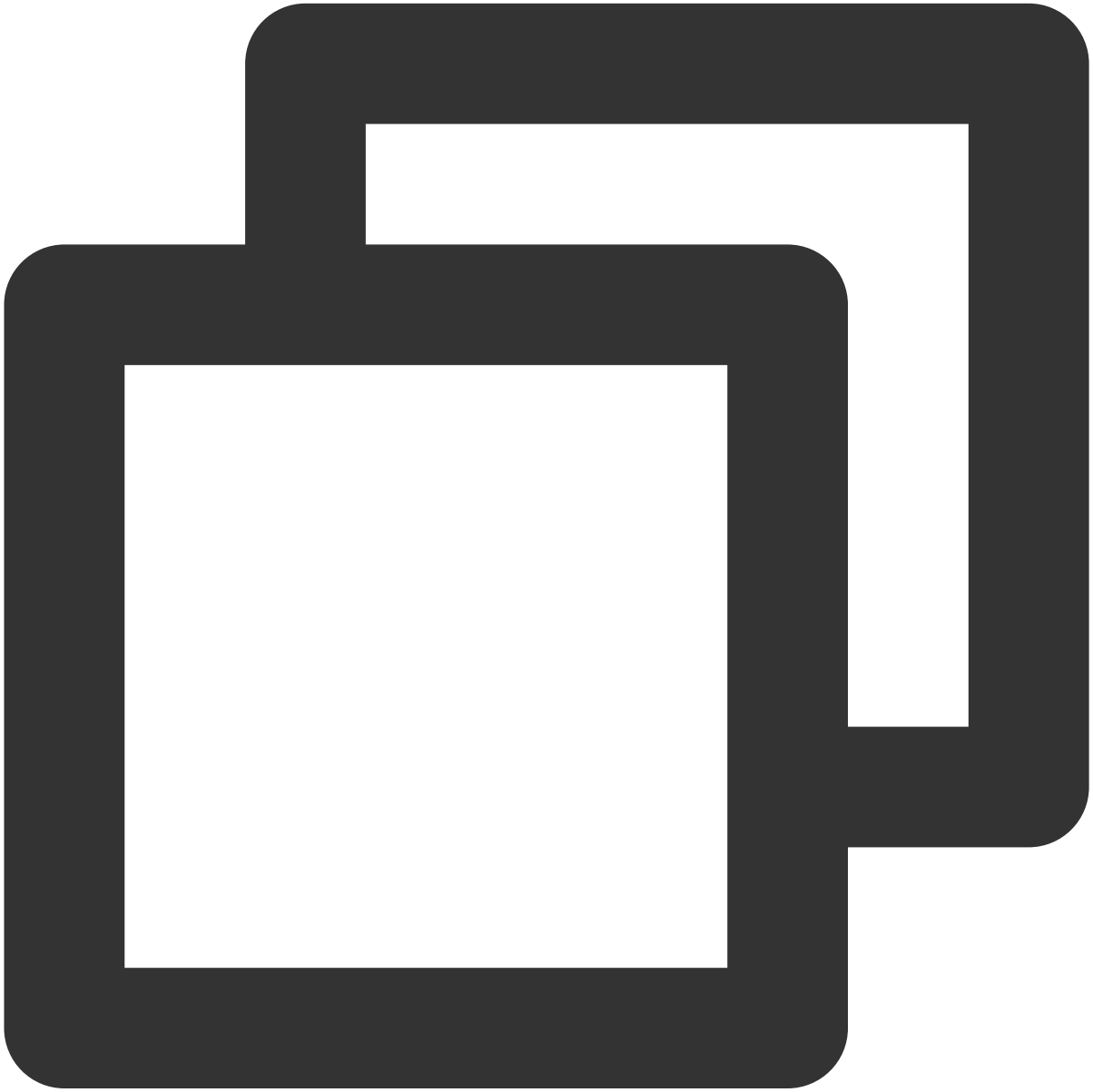


```
- (void)setVideoRenderParams:(NSString *)userId params:(TUIVideoRenderParams *)para
```

Parameter	Type	Description
userId	NSString	The target user ID.
params	TUIVideoRenderParams	Video render parameters.

setVideoEncoderParams

Set the encoding parameters of video encoder.
This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.

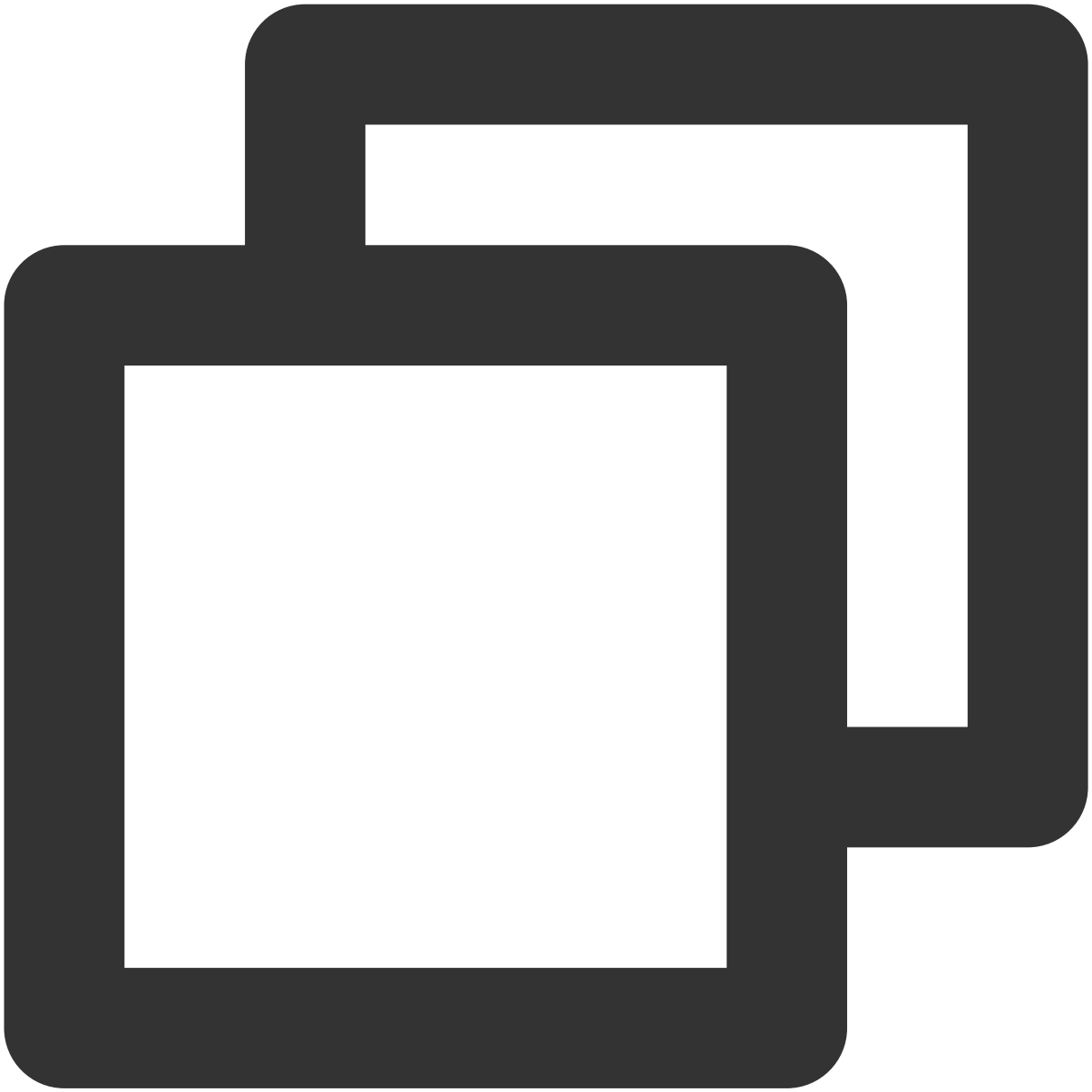


```
- (void)setVideoEncoderParams:(TUIVideoEncoderParams *)params succ:(TUICallSucc)suc
```

Parameter	Type	Description
params	TUIVideoEncoderParams	Video encoding parameters

getTRTCCloudInstance

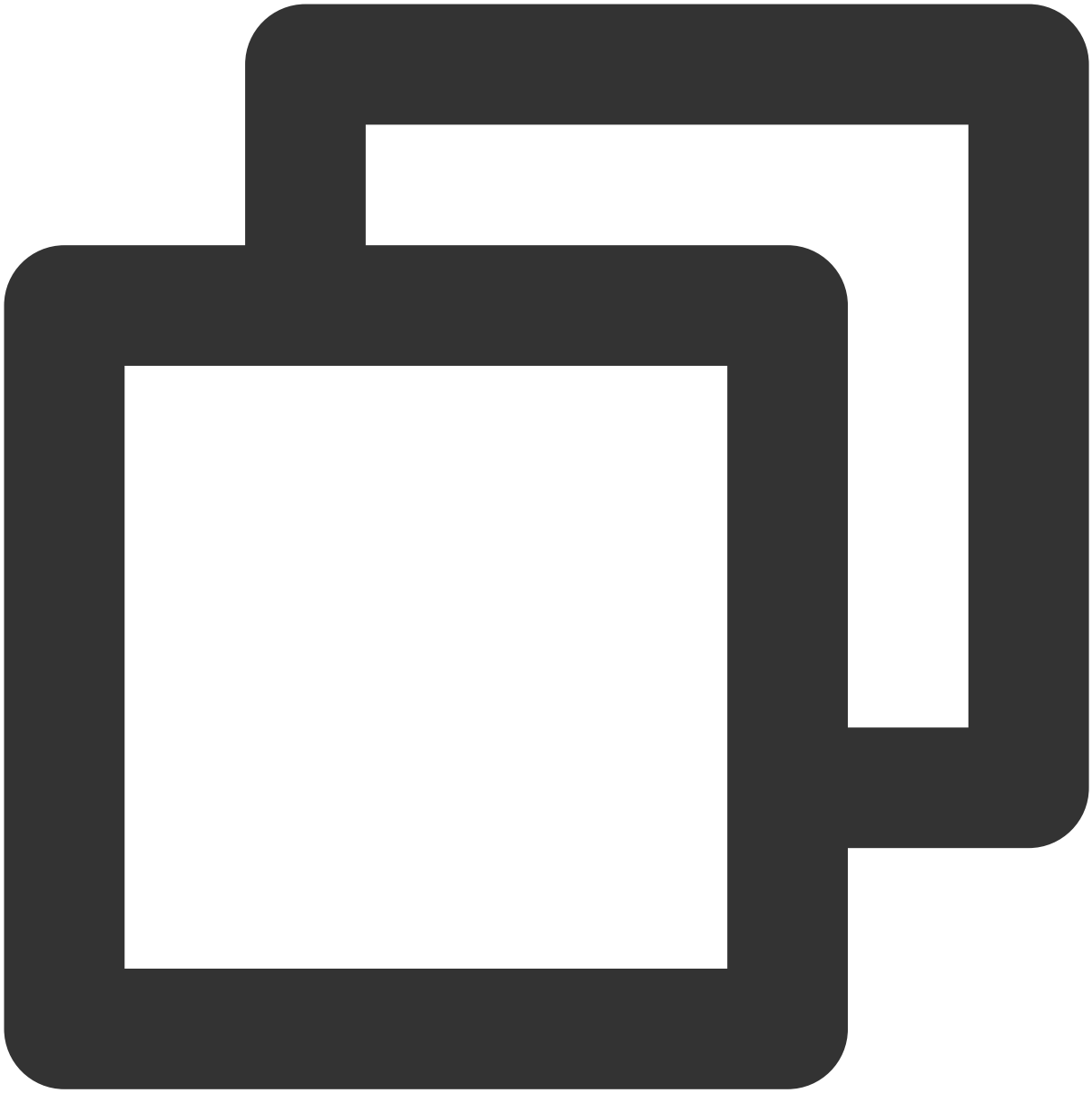
Advanced features.



```
- (TRTCCloud *)getTRTCCloudInstance;
```

setBeautyLevel

Set beauty level, support turning off default beauty.



```
- (void)setBeautyLevel:(CGFloat)level succ:(TUICallSucc)succ fail:(TUICallFail)fail
```

Parameter	Type	Description
level	CGFloat	Beauty level，range: 0 - 9； 0 means turning off the effect, 9 means the most obvious effect.

TUICallObserver

Last updated：2024-05-28 18:02:41

TUICallObserver APIs

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

Overview

API	Description
onError	An error occurred during the call.
onCallReceived	A call was received.
onCallCancelled	The call was canceled.
onCallBegin	The call was connected.
onCallEnd	The call ended.
onCallMediaTypeChanged	The call type changed.
onUserReject	A user declined the call.
onUserNoResponse	A user didn't respond.
onUserLineBusy	A user was busy.
onUserJoin	A user joined the call.
onUserLeave	A user left the call.
onUserVideoAvailable	Whether a user had a video stream.
onUserAudioAvailable	Whether a user had an audio stream.
onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.
onKickedOffline	The current user was kicked offline.

onUserSigExpired	The user sig is expired.

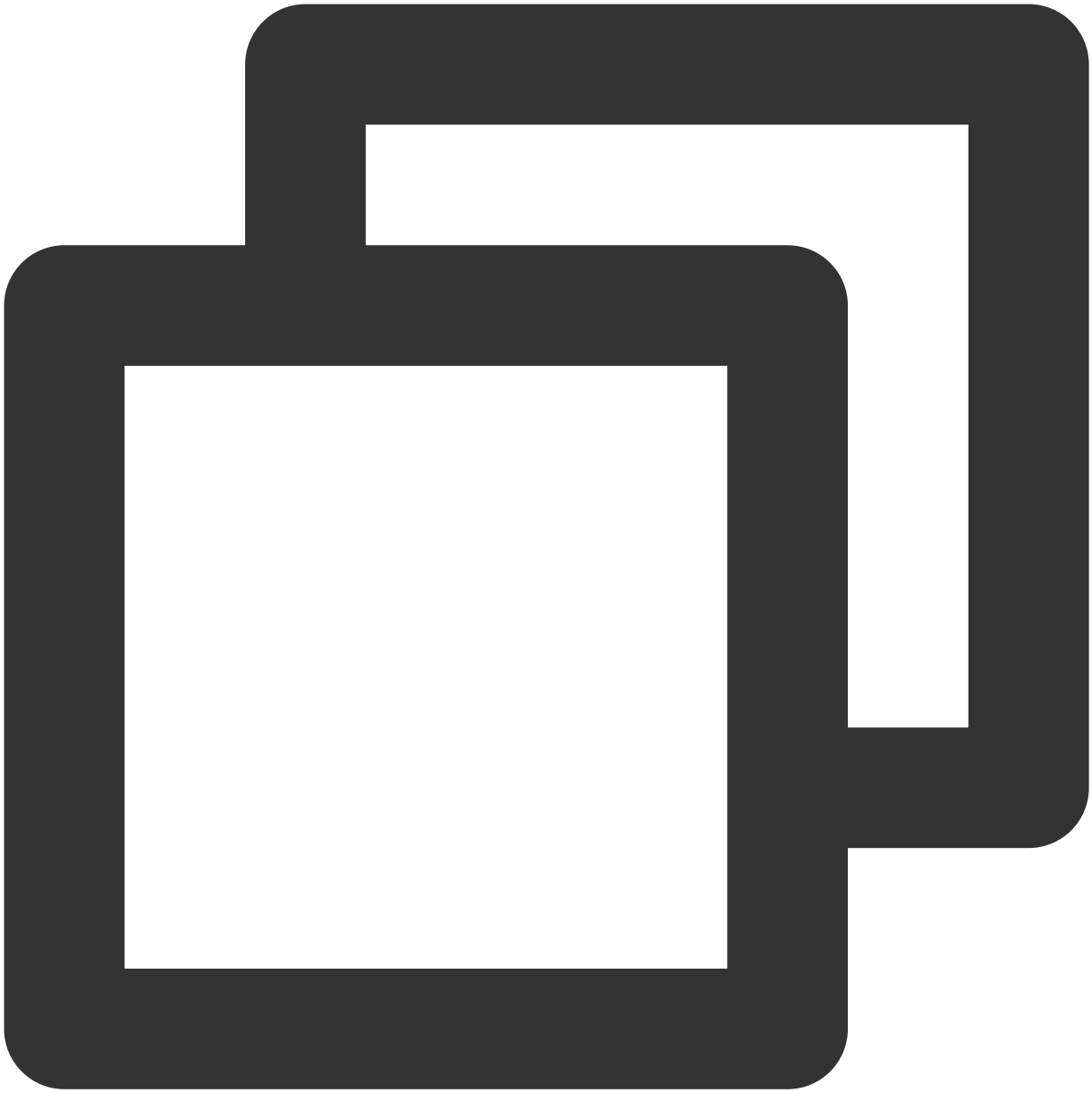
Details

onError

An error occurred.

Note

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



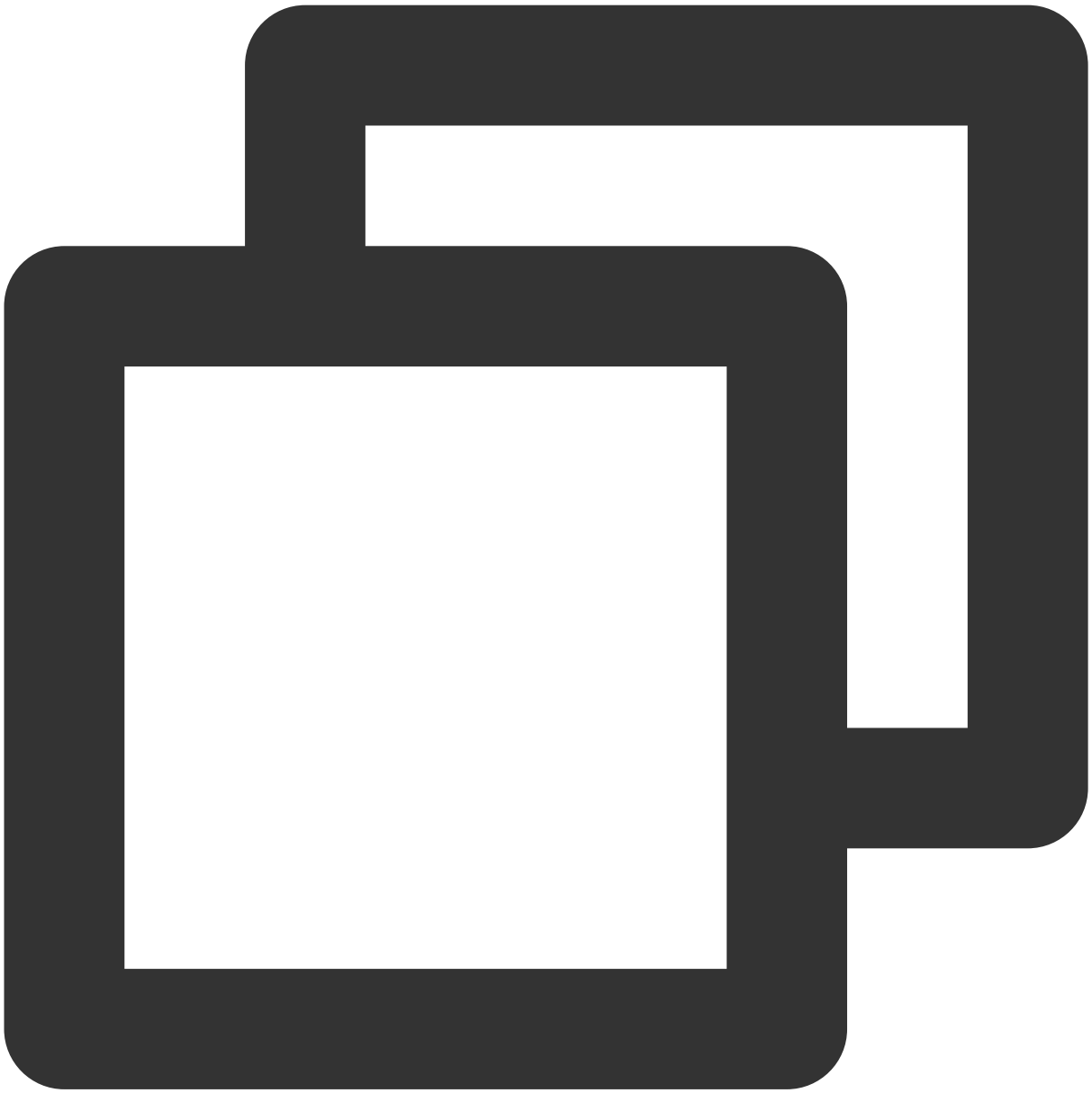
```
- (void)onError:(int)code message:(NSString * _Nullable)message;
```

The parameters are described below:

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

onCallReceived

A call invitation was received. This callback is received by an invitee. You can listen for this event to determine whether to display the incoming call view.



```
- (void)onCallReceived:(NSString *)callerId calleeIdList:(NSArray<NSString *> *)cal
```

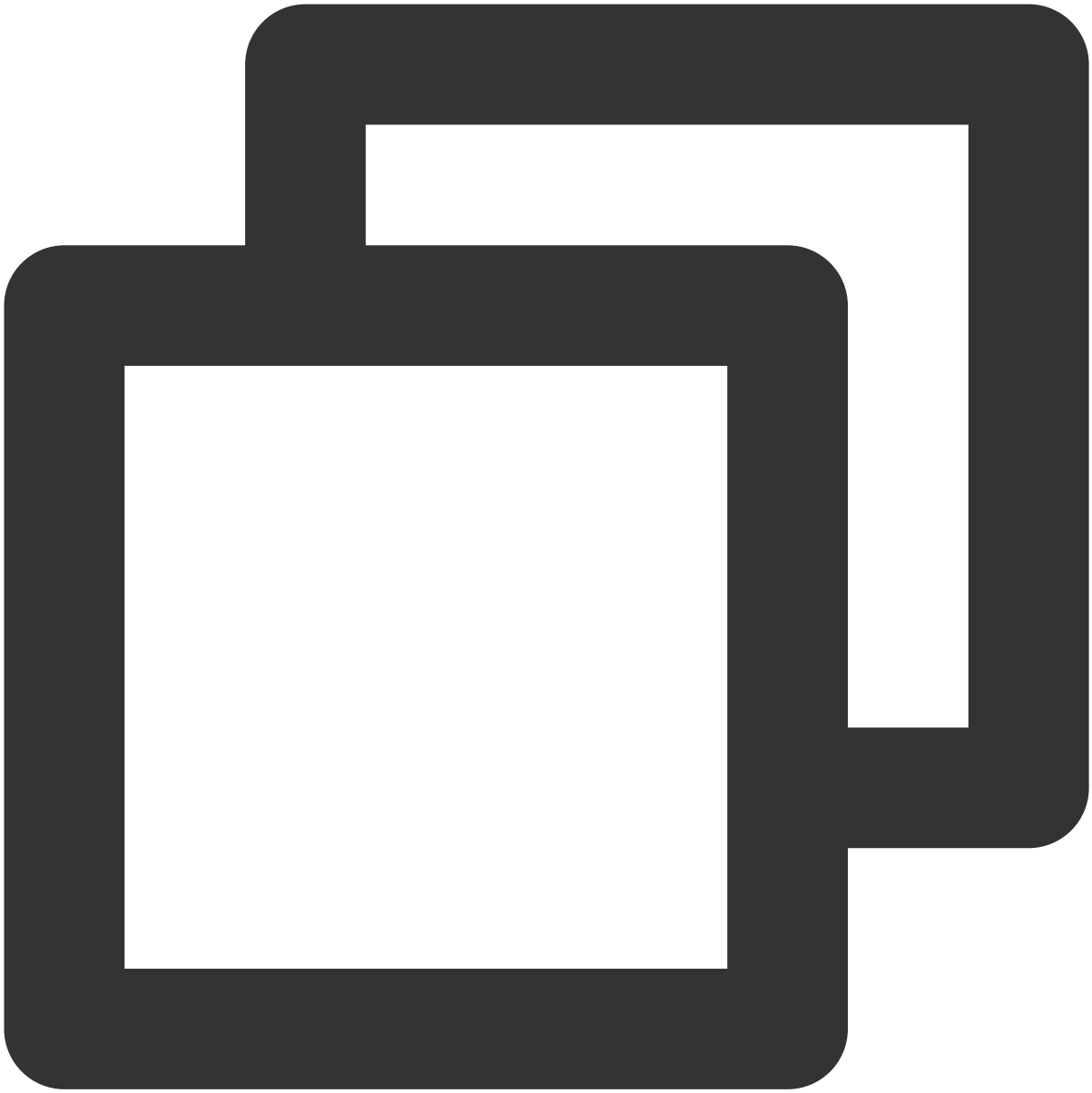
The parameters are described below:

Parameter	Type	Description
callerId	NSString	The user ID of the inviter.
calleeIdList	NSArray	The invitee list.

groupId	NSString	The group ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.
userData	NSString	User-added extended fields., Please refer to:

onCallCancelled

The call was canceled by the inviter or timed out. This callback is received by an invitee. You can listen for this event to determine whether to show a missed call message.
This indicates that the call was canceled by the caller, timed out by the callee, rejected by the callee, or the callee was busy. There are multiple scenarios involved. You can listen to this event to achieve UI logic such as missed calls and resetting UI status.
Call cancellation by the caller: The caller receives the callback (userId is himself); the callee receives the callback (userId is the ID of the caller)
Callee timeout: the caller will simultaneously receive the onUserNoResponse and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)
Callee rejection: The caller will simultaneously receive the onUserReject and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)
Callee busy: The caller will simultaneously receive the onUserLineBusy and onCallCancelled callbacks (userId is his own ID);
Abnormal interruption: The callee failed to receive the call , he receives this callback (userId is his own ID).



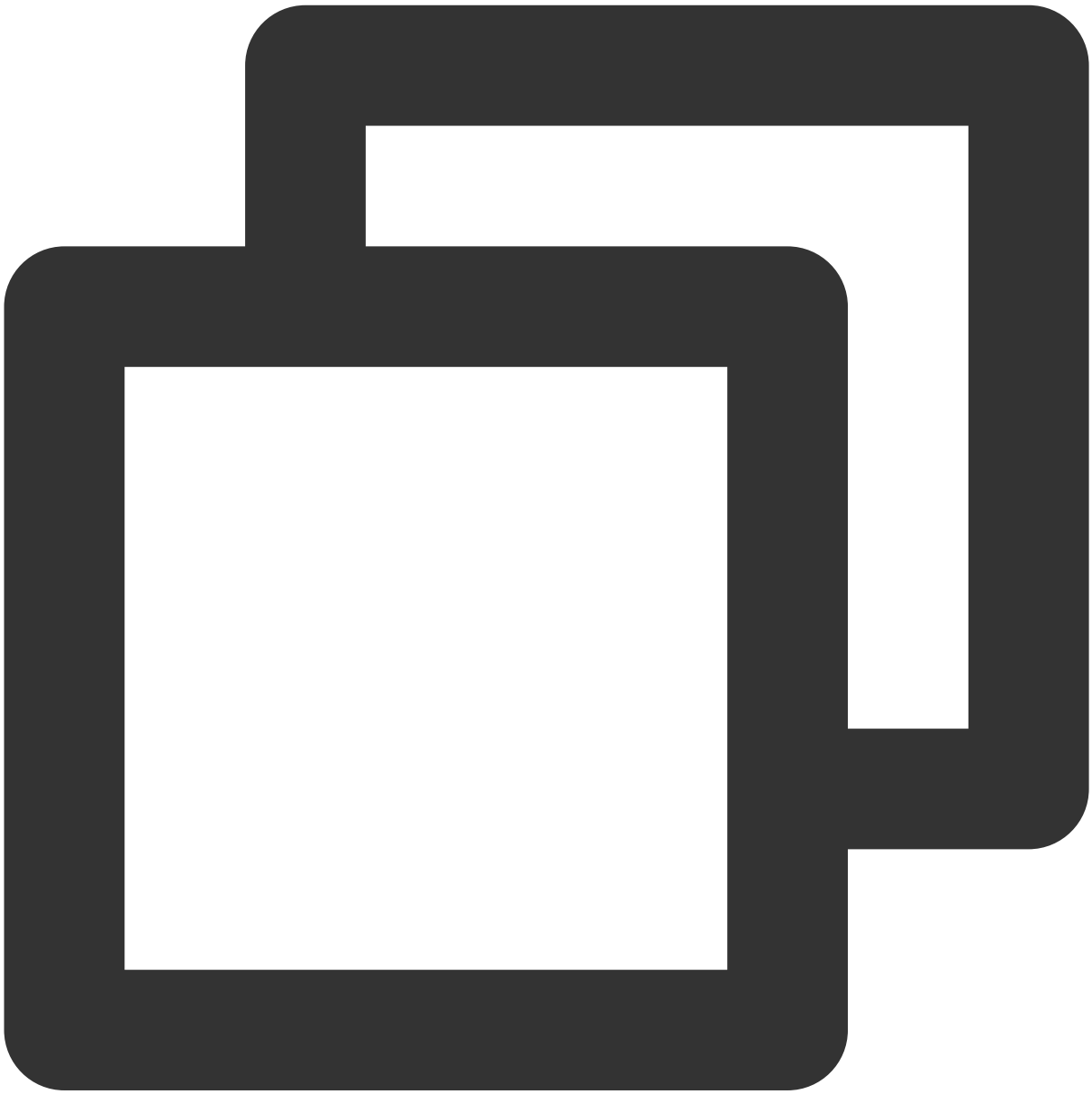
```
- (void)onCallCancelled:(NSString *)callerId;
```

The parameters are described below:

Parameter	Type	Description
callerId	NSString	The user ID of the inviter.

onCallBegin

The call was connected. This callback is received by both the inviter and invitees. You can listen for this event to determine whether to start on-cloud recording, content moderation, or other tasks.



```
- (void)onCallBegin:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaTy
```

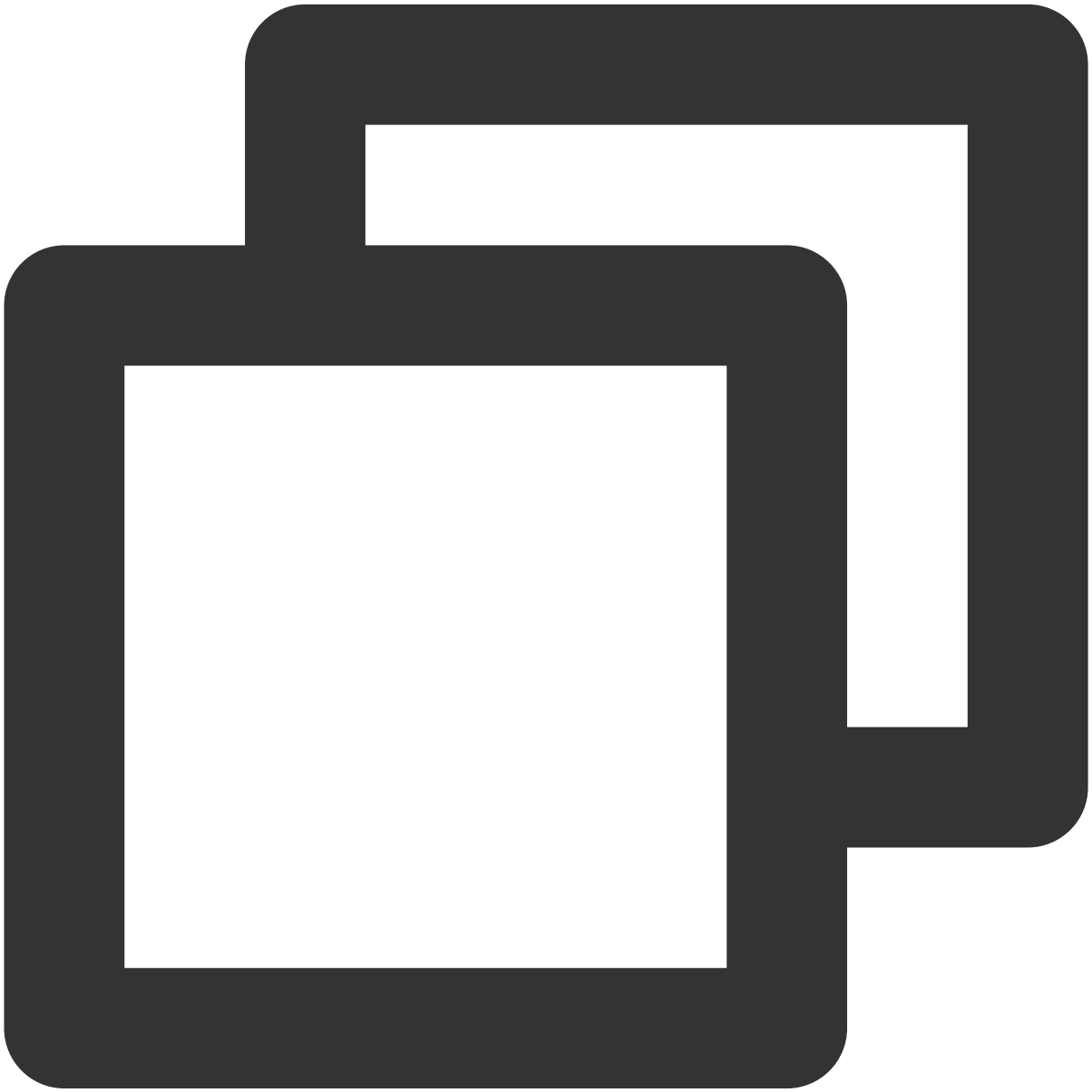
The parameters are described below:

Parameter	Type	Description
roomId	TUIRoomId	The room ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.

callRole	TUICallRole	The role, which can be caller or callee.

onCallEnd

The call ended. This callback is received by both the inviter and invitees. You can listen for this event to determine when to display call information such as call duration and call type, or stop on-cloud recording.



```
- (void)onCallEnd:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaType
```

The parameters are described below:

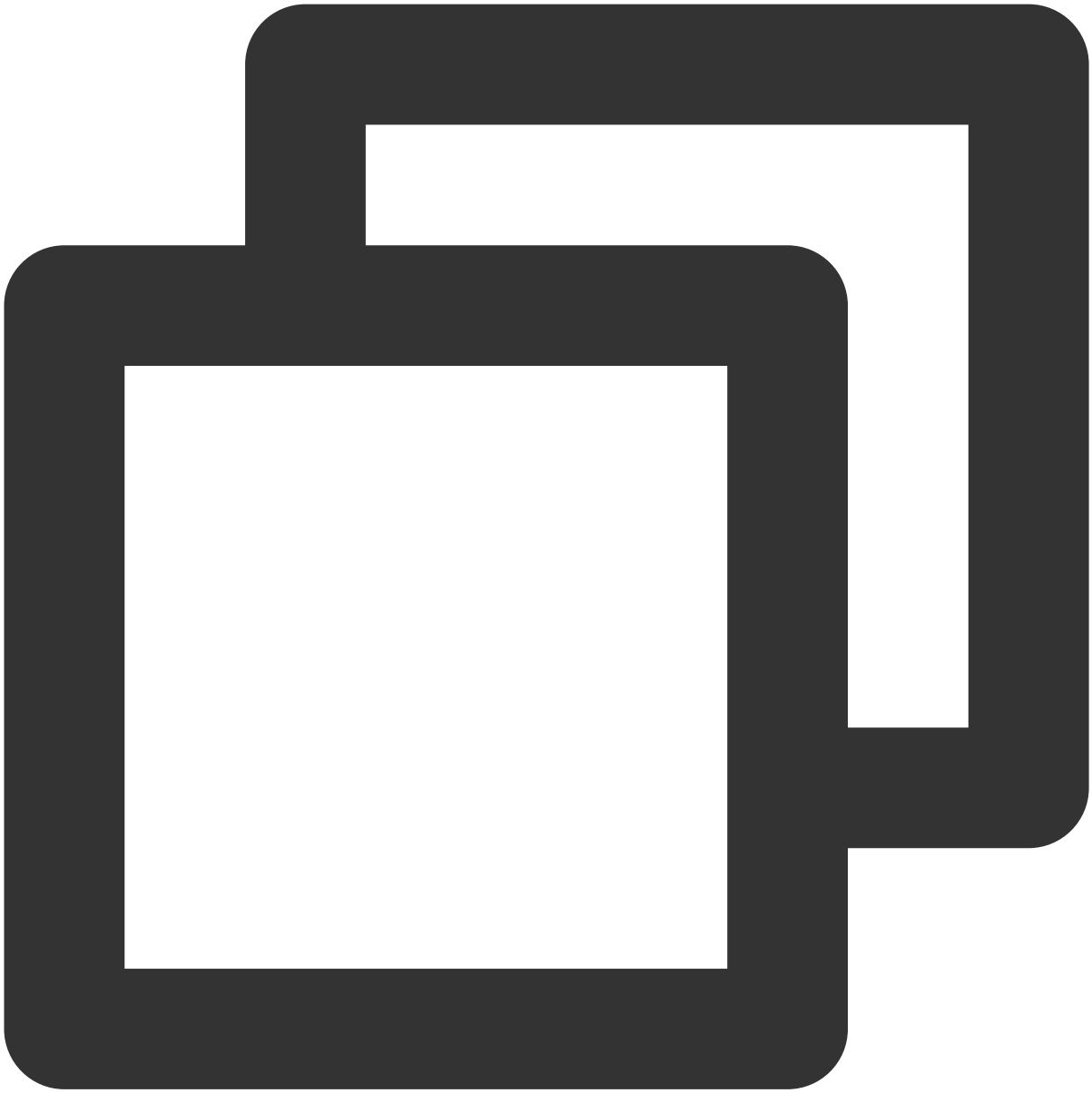
Parameter	Type	Description
roomId	TUIRoomId	The room ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.
callRole	TUICallRole	The role, which can be caller or callee.
totalTime	float	The call duration.

Note

Client-side callbacks are often lost when errors occur, for example, when the process is closed. If you need to measure the duration of a call for billing or other purposes, we recommend you use the RESTful API.

onCallMediaTypeChanged

The call type changed.



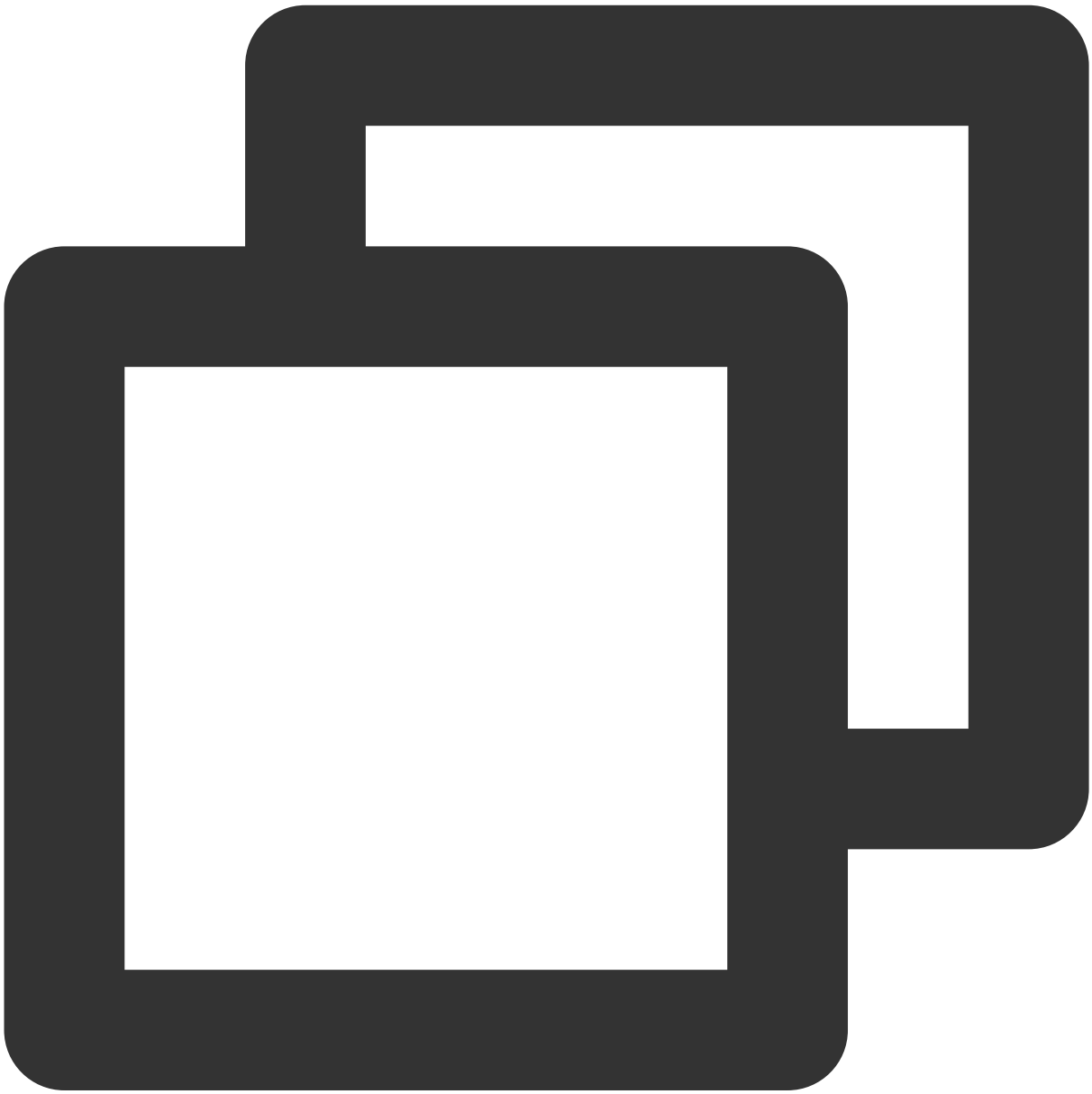
```
- (void)onCallMediaTypeChanged:(TUICallMediaType)oldCallMediaType newCallMediaType:
```

The parameters are described below:

Parameter	Type	Description
oldCallMediaType	TUICallMediaType	The call type before the change.
newCallMediaType	TUICallMediaType	The call type after the change.

onUserReject

The call was rejected. In a one-to-one call, only the inviter will receive this callback. In a group call, all invitees will receive this callback.



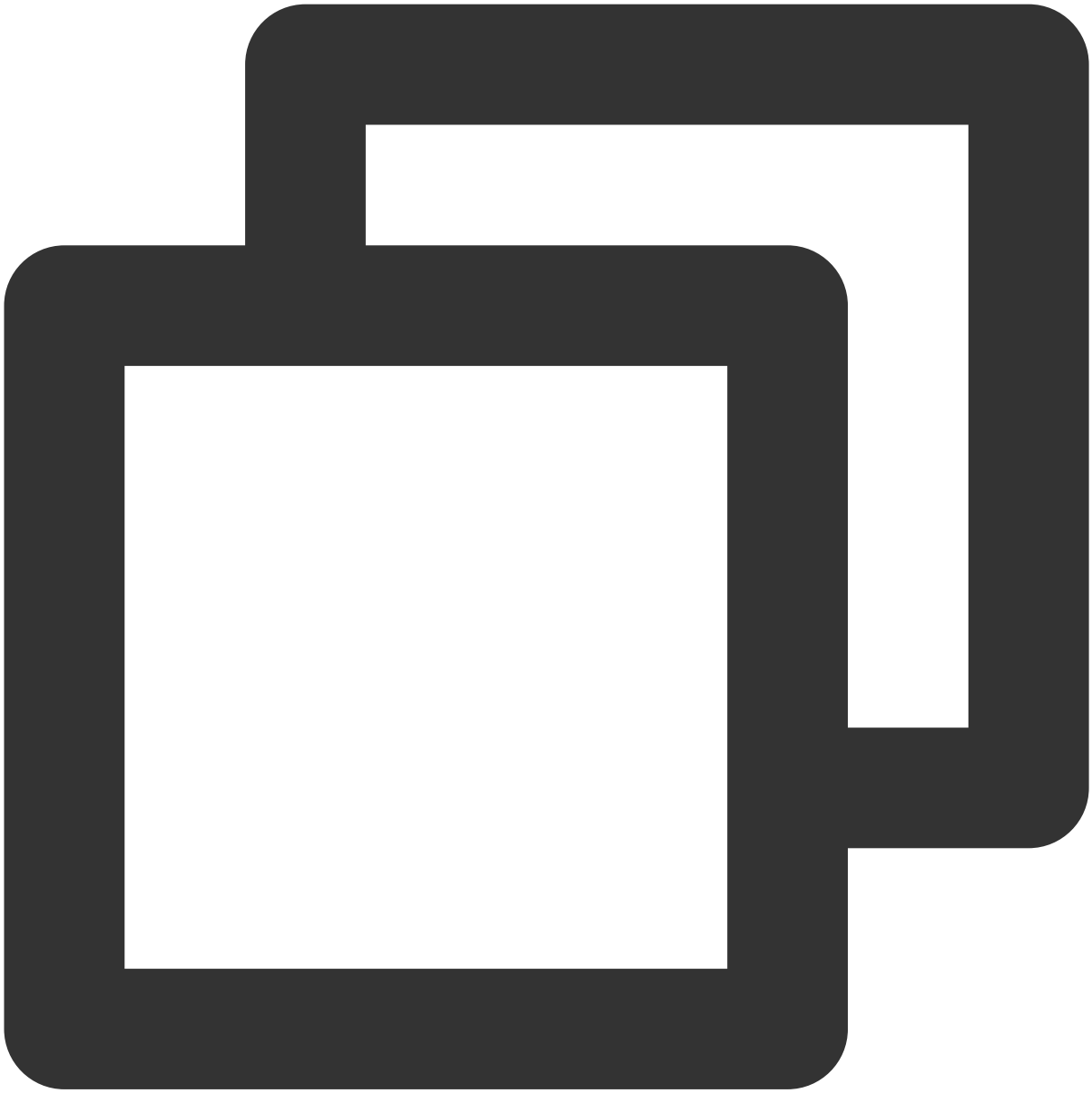
```
- (void)onUserReject:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID of the invitee who rejected the call.

onUserNoResponse

A user did not respond.



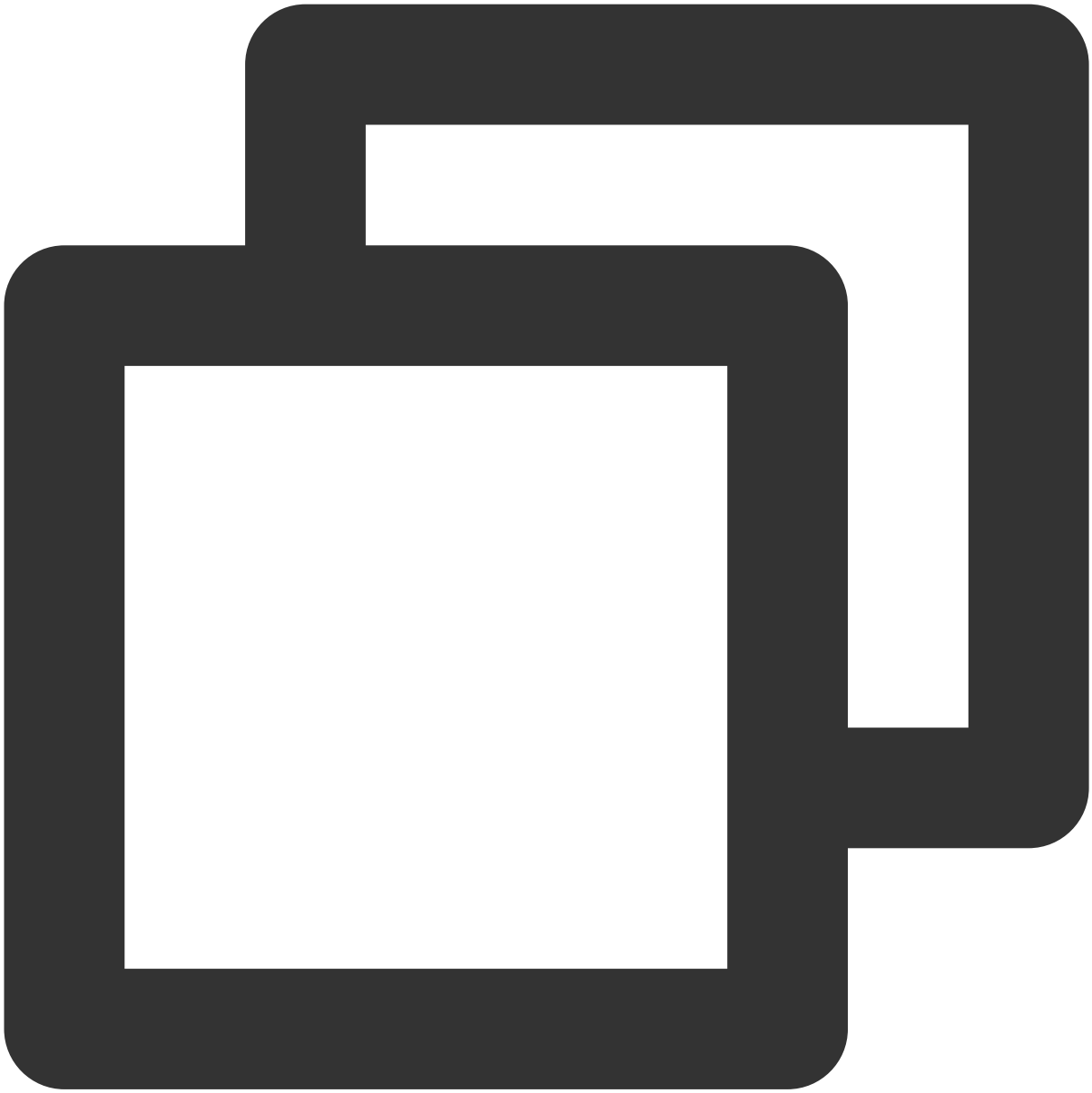
```
- (void)onUserNoResponse:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID of the invitee who did not answer.

onUserLineBusy

A user is busy.



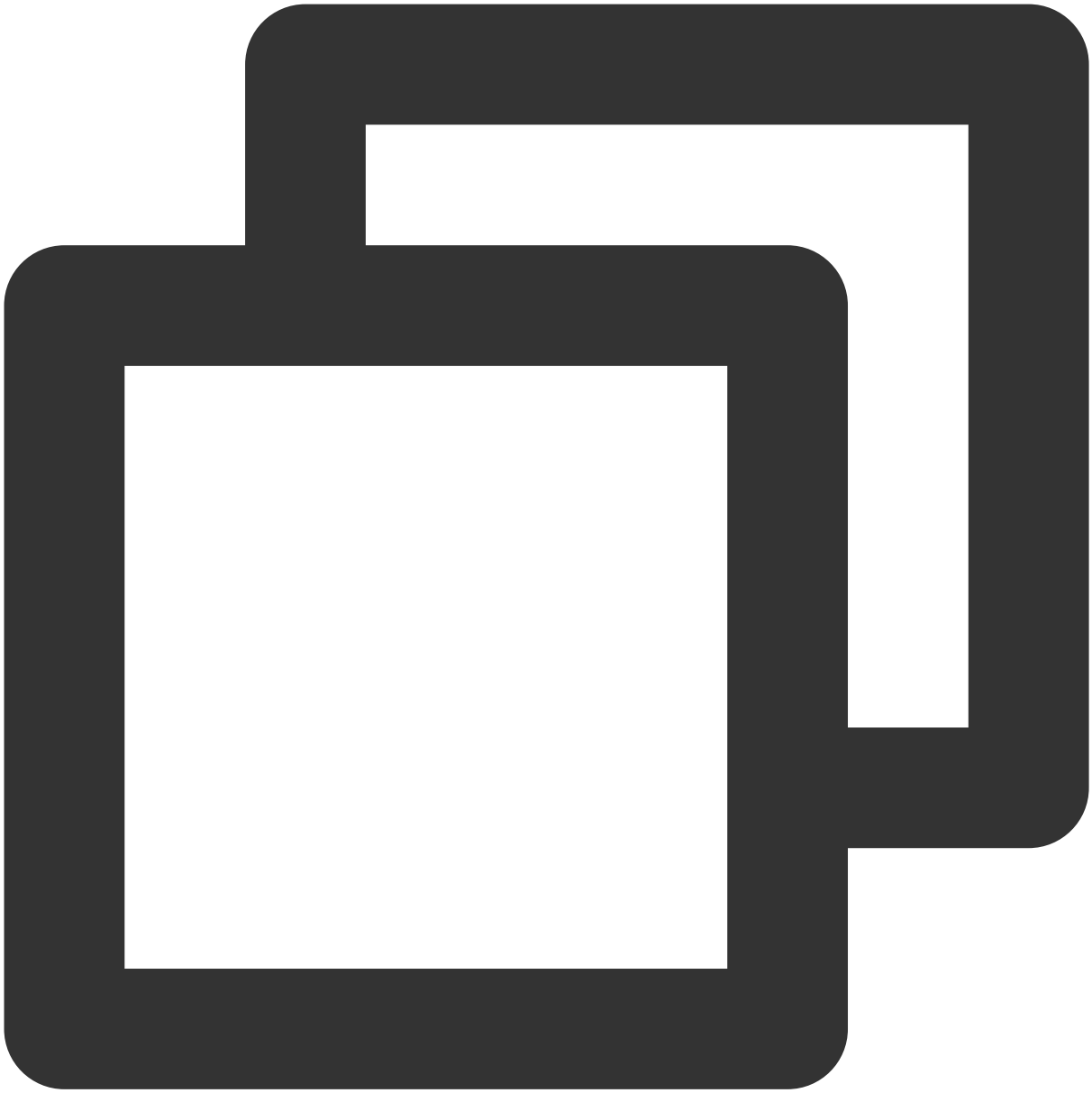
```
- (void)onUserLineBusy:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID of the invitee who is busy.

onUserJoin

A user joined the call.



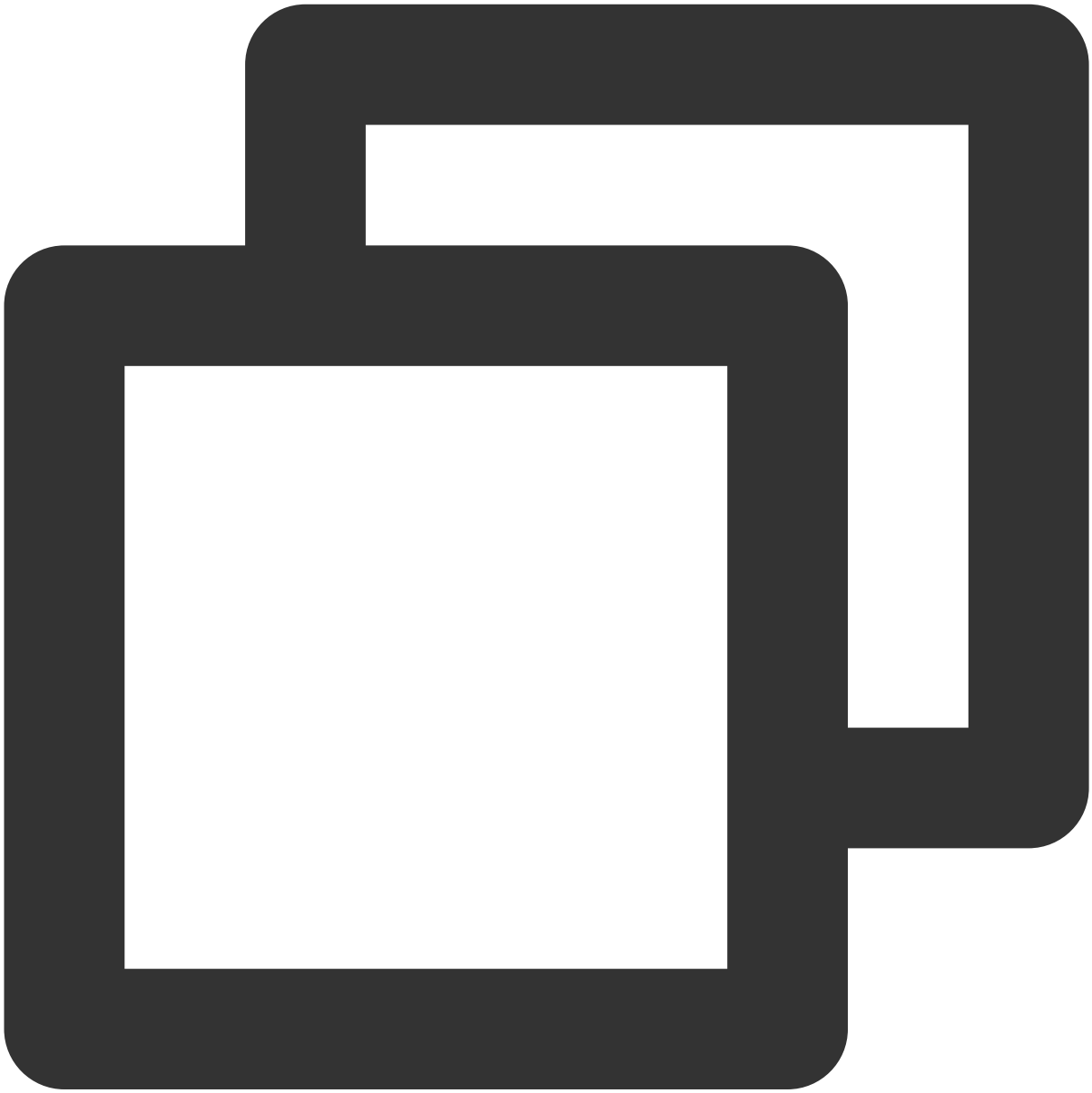
```
- (void)onUserJoin:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The ID of the user who joined the call.

onUserLeave

A user left the call.



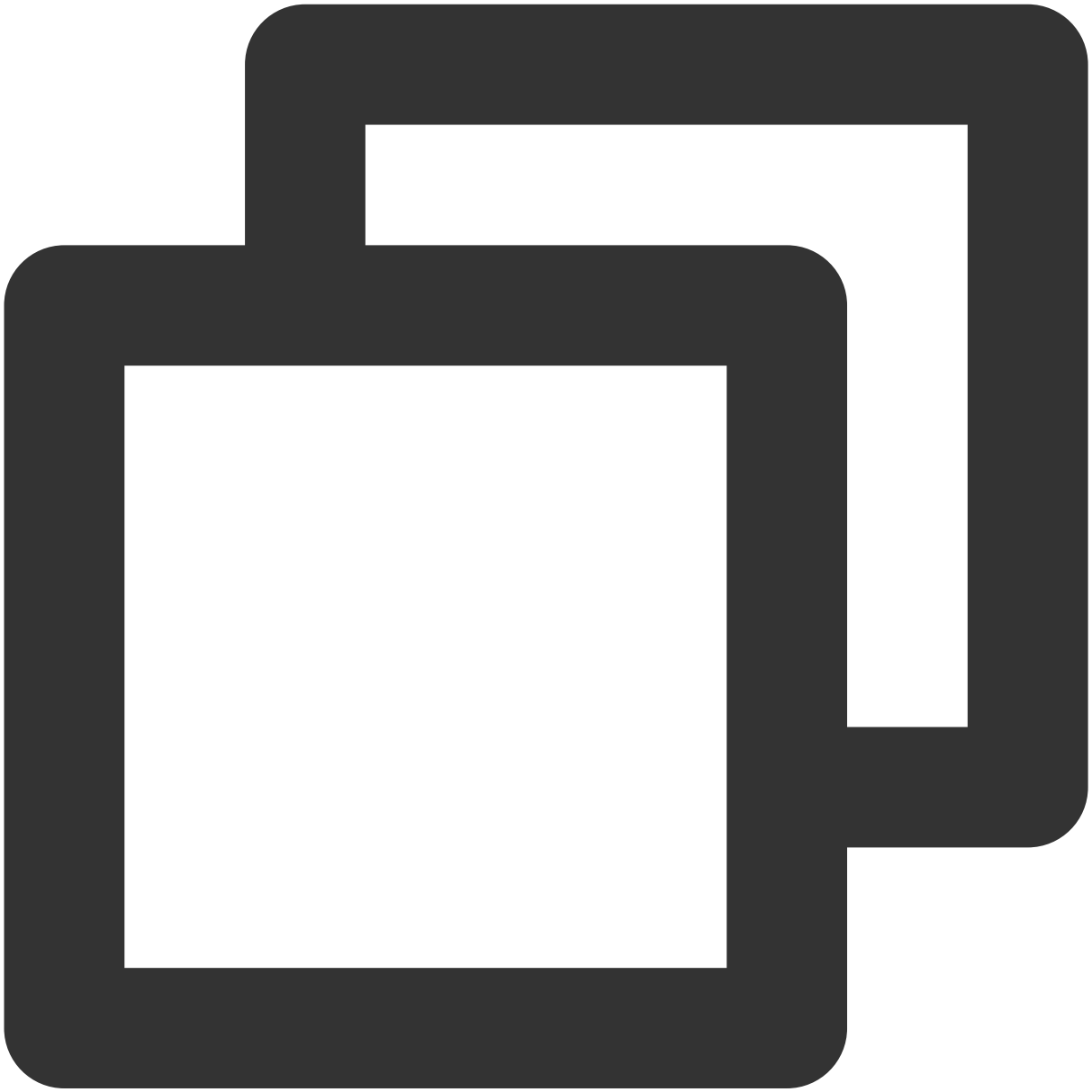
```
- (void)onUserLeave:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The ID of the user who left the call.

onUserAudioAvailable

Whether a user is sending audio.



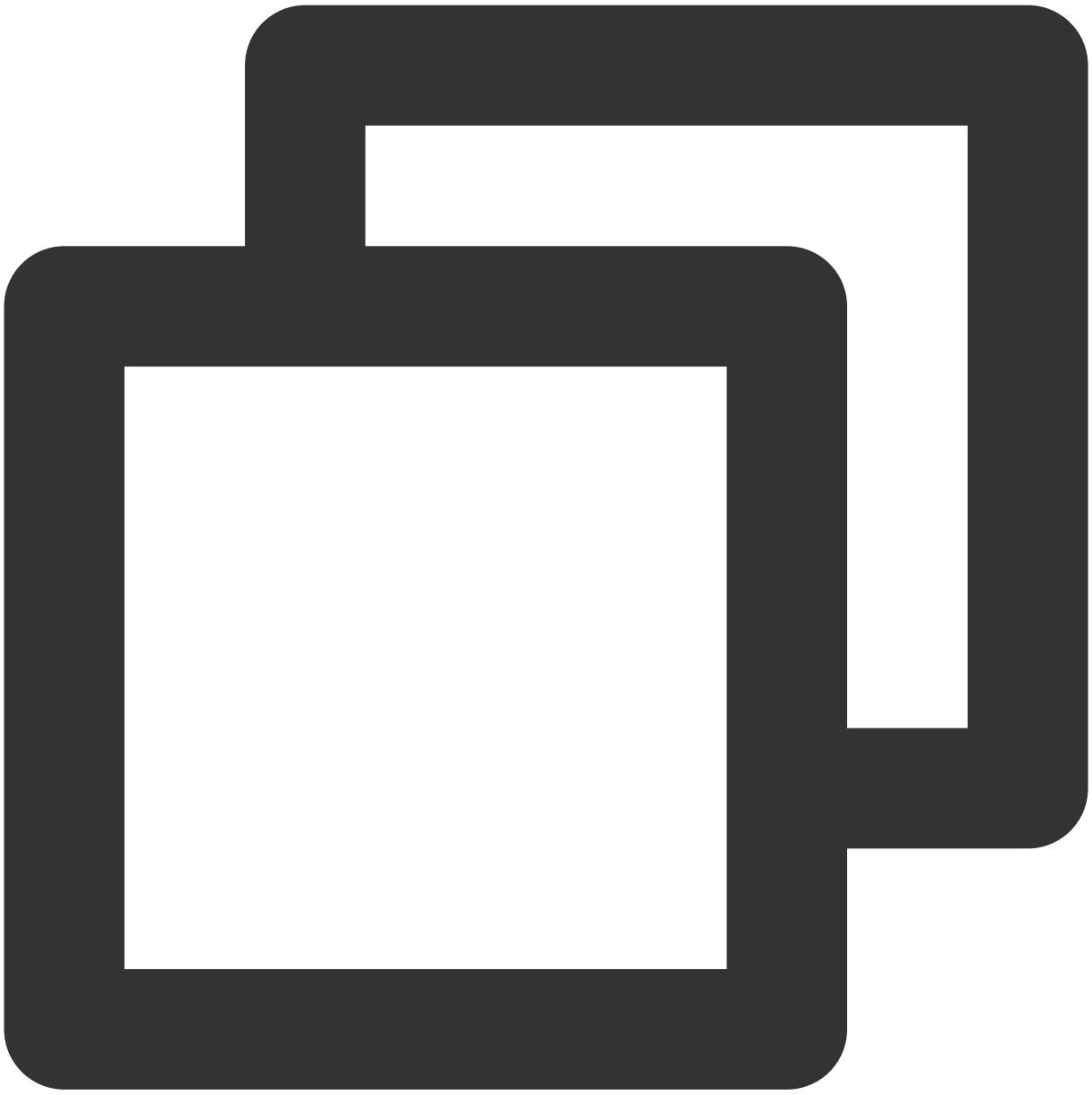
```
- (void)onUserAudioAvailable:(NSString *)userId isAudioAvailable:(BOOL)isAudioAvail
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.
isAudioAvailable	BOOL	Whether the user has audio.

onUserVideoAvailable

Whether a user is sending video.



```
- (void)onUserVideoAvailable:(NSString *)userId isVideoAvailable:(BOOL)isVideoAvail
```

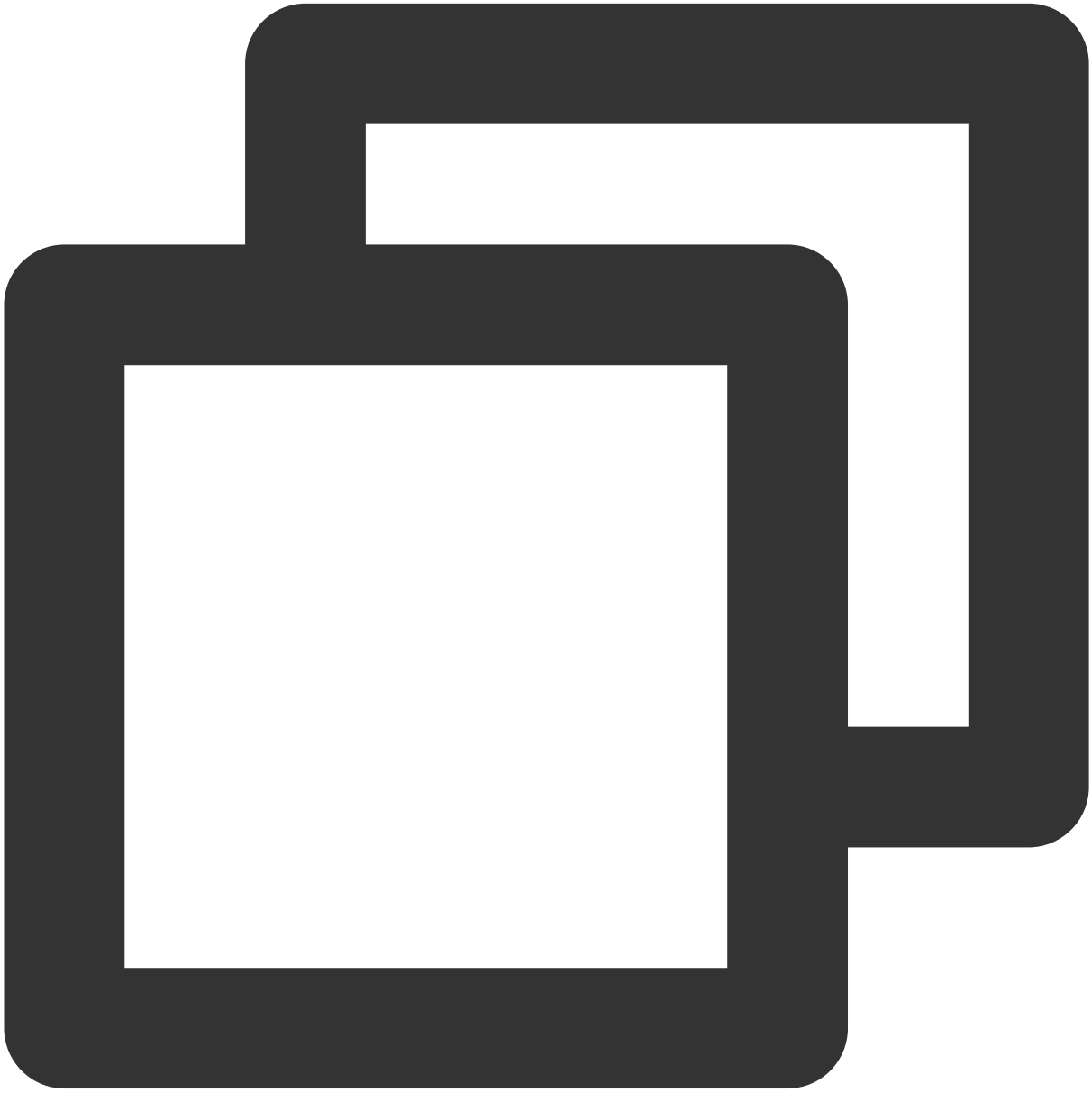
The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.

isVideoAvailable	BOOL	Whether the user has video.

onUserVoiceVolumeChanged

The volumes of all users.



```
- (void)onUserVoiceVolumeChanged:(NSDictionary <NSString *, NSNumber *> *)volumeMap
```

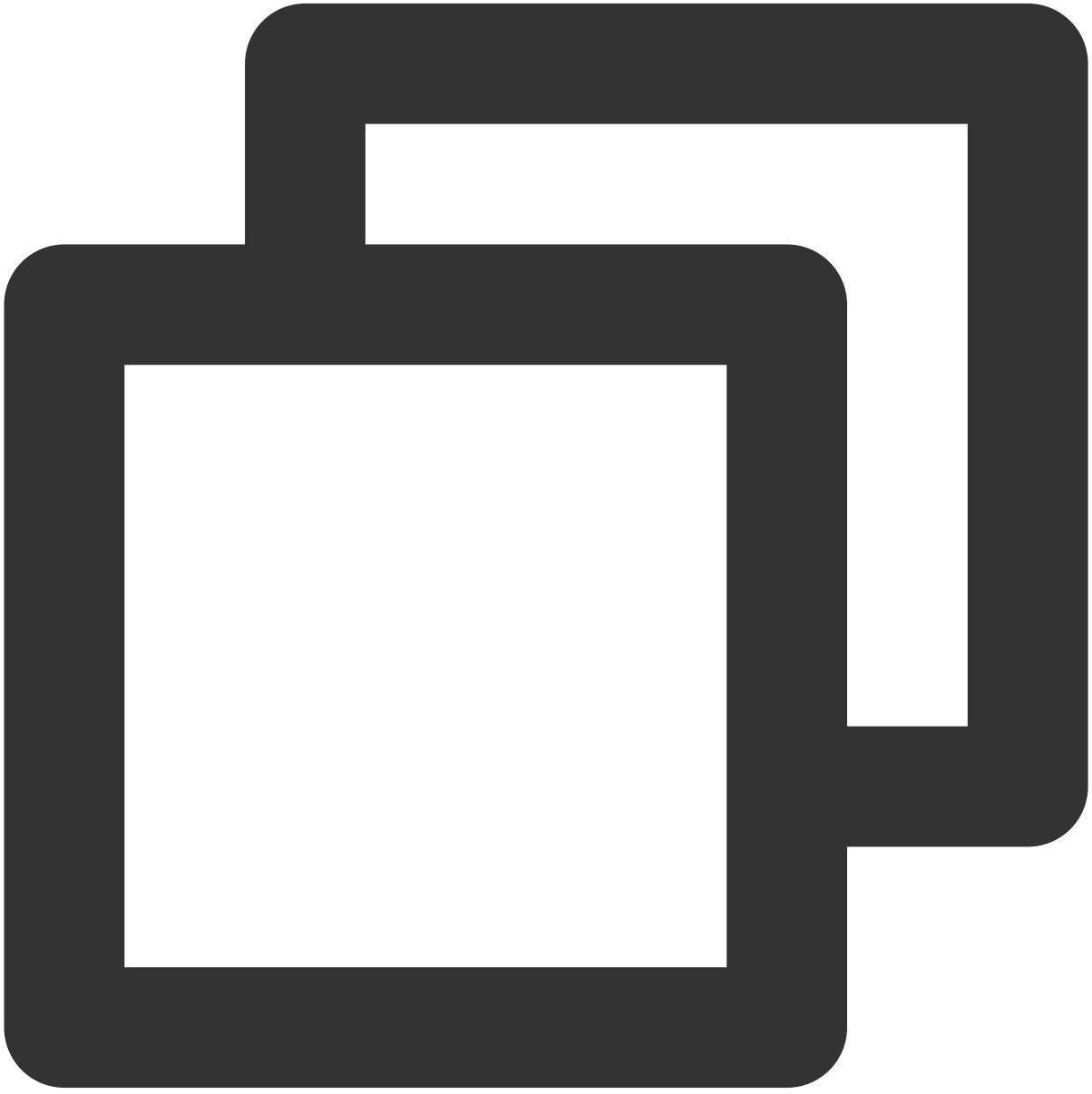
The parameters are described below:

Parameter	Type	Description

volumeMap	NSDictionary	The volume table, which includes the volume of each user (`userId`). Value range: 0-100.

onUserNetworkQualityChanged

The network quality of all users.



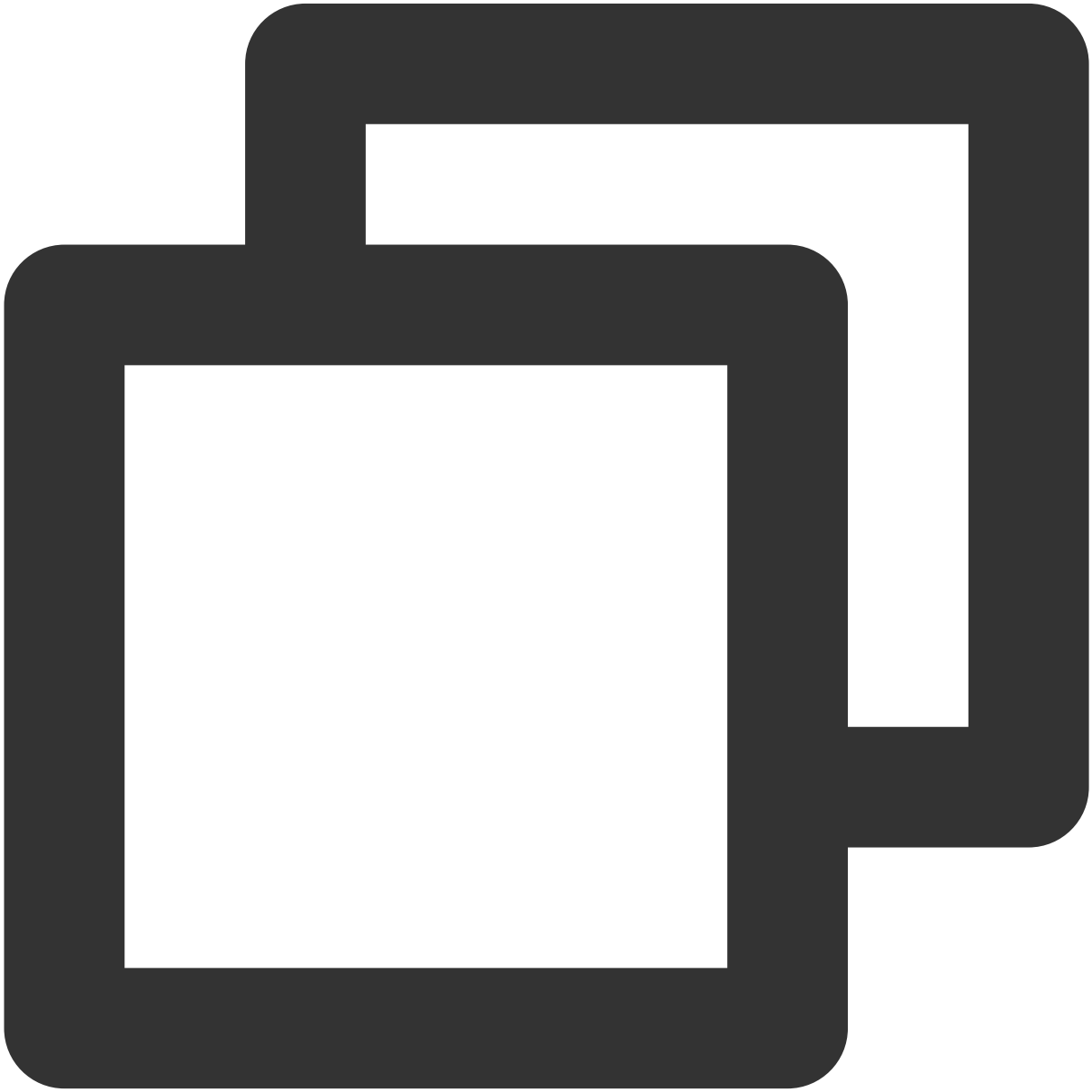
```
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkQualityInfo *> *)networkQual
```

The parameters are described below:

Parameter	Type	Description
networkQualityList	NSArray	The current network conditions for all users (`userId`).

onKickedOffline

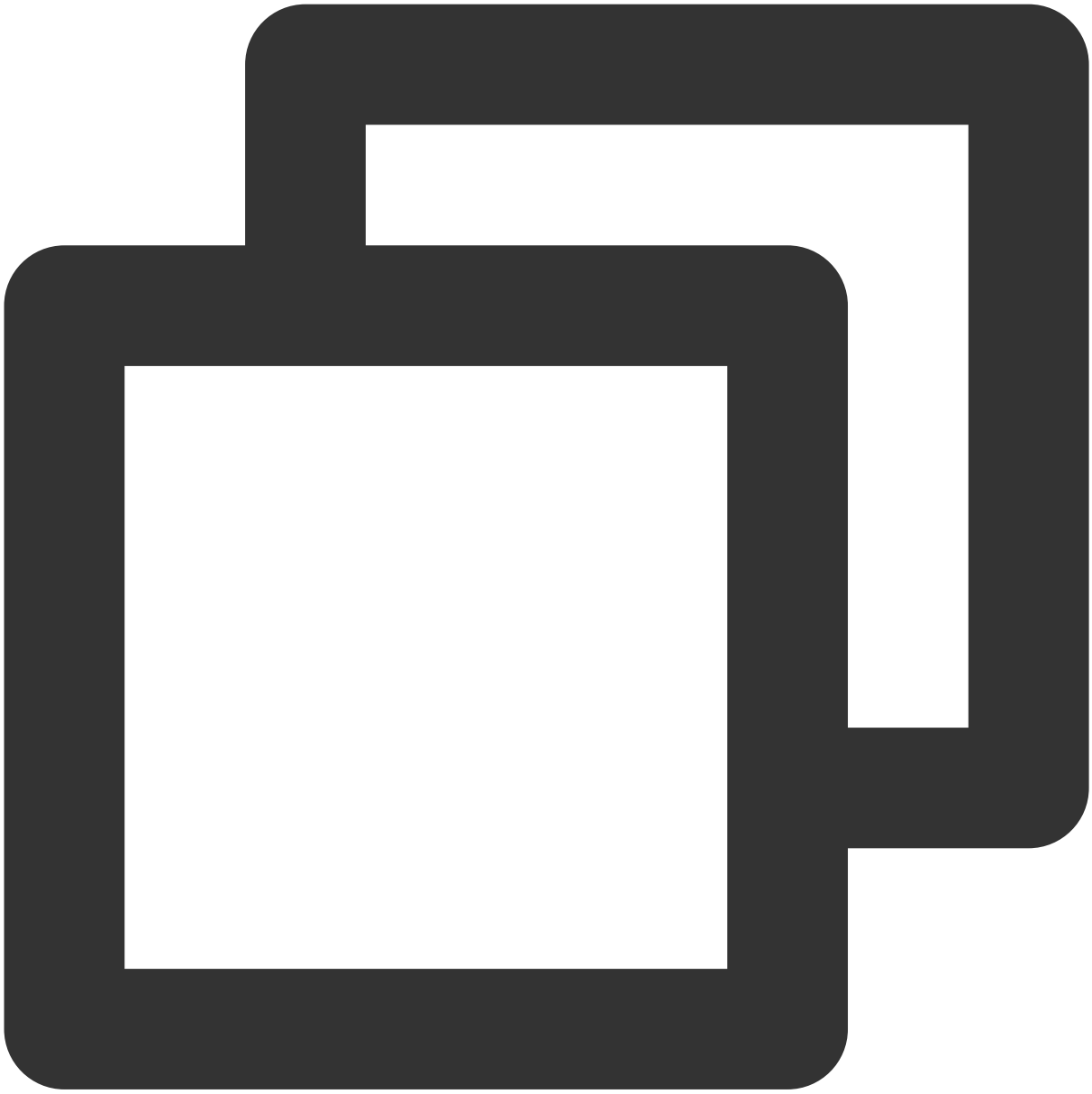
The current user was kicked offline：At this time, you can prompt the user with a UI message and then invoke `init` again.



```
- (void)onKickedOffline;
```

onUserSigExpired

The user sig is expired：At this time, you need to generate a new `userSig` , and then invoke `init` again.



```
- (void)onUserSigExpired;
```

Type Definition

Last updated：2024-05-28 18:03:34

Common structures

TUICallDefine

Type	Description
TUICallParams	An additional parameter.
TUIOfflinePushInfo	Offline push vendor configuration information.

TUICommonDefine

Type	Description
TUIRoomId	Room ID for audio and video in a call.
TUINetworkQuality	Network quality information
TUIVideoRenderParams	Video render parameters
TUIVideoEncoderParams	Video encoding parameters

Enum definition

TUICallDefine

Type	Description
TUICallMediaType	Media type in a call
TUICallRole	Roles of individuals in a call.
TUICallStatus	The call status
TUICallScene	The call scene
TUICallIOSOfflinePushType	iOS offline push type

TUICommonDefine

Type	Description

TUIAudioPlaybackDevice	Audio route
TUICamera	Camera type
TUINetworkQuality	Network quality
TUIVideoRenderParamsFillMode	Video image fill mode
TUIVideoRenderParamsRotation	Video image rotation direction
TUIVideoEncoderParamsResolutionMode	Video aspect ratio mode
TUIVideoEncoderParamsResolution	Video resolution

TUICallParams

Call params

Value	Type	Description
roomId	TUIRoomId	Room ID for audio and video in a call.
offlinePushInfo	TUIOfflinePushInfo	Offline push vendor configuration information.
timeout	int	Call timeout period, default: 30s, unit: seconds.
userData	NSString	An additional parameter. Callback when the callee receives onCallReceived

TUIOfflinePushInfo

Offline push vendor configuration information, please refer to：Offline call push

Value	Type	Description
title	NSString	offlinepush notification title
desc	NSString	offlinepush notification description
ignoreIOSBadge	BOOL	Ignore badge count for offline push (only for iOS), if set to true, the message will not increase the unread count of the app icon on the iOS receiver's side.
iOSSound	NSString	Offline push sound setting (only for iOS). When sound = **IOS_OFFLINE_PUSH_NO_SOUND**, there will be no sound played when the message is received. When sound = **IOS_OFFLINE_PUSH_DEFAULT_SOUND**,

		the system sound will be played when the message is received. If you want to customize the iOSSound, you need to link the audio file into the Xcode project first, and then set the audio file name (with extension) to the iOSSound.
androidSound	NSString	Offline push sound setting (only for Android, supported by IMSDK 6.1 and above). Only Huawei and Google phones support setting sound prompts. For Xiaomi phones, please refer to: Xiaomi custom ringtones. In addition, for Google phones, in order to set sound prompts for FCM push on Android 8.0 and above systems, you must call setAndroidFCMChannelID to set the channelID for it to take effect.
androidOPPOChannelID	NSString	Set the channel ID for OPPO phones with Android 8.0 and above systems.
androidVIVOClassification	NSInteger	Classification of VIVO push messages (deprecated interface, VIVO push service will optimize message classification rules on April 3, 2023. It is recommended to use setAndroidVIVOCategory to set the message category). 0: Operational messages, 1: System messages. The default value is 1.
androidXiaoMiChannelID	NSString	Set the channel ID for Xiaomi phones with Android 8.0 and above systems.
androidFCMChannelID	NSString	Set the channel ID for google phones with Android 8.0 and above systems.
androidHuaWeiCategory	NSString	Classification of Huawei push messages, please refer to: Huawei message classification standard.
isDisablePush	BOOL	Whether to turn off push notifications (default is on).
iOSPushType	TUICallIOSOfflinePushType	iOS offline push type， default is APNs

TUIRoomId

Room ID for audio and video in a call.

Note：

(1) `intRoomId` and `strRoomId` are mutually exclusive. If you choose to use `strRoomId`, `intRoomId` needs to be filled in as 0. If both are filled in, the SDK will prioritize `intRoomId`.
(2) Do not mix `intRoomId` and `strRoomId` because they are not interchangeable. For example, the number 123 and the string "123" represent two completely different rooms.

Value	Type	Description
intRoomId	UInt32	Numeric room ID. **range:** 1 - 2147483647(2^31-1)
strRoomId	NSString	String room ID. **range**：Limited to 64 bytes in length. The supported character set range is as follows (Lowercase and uppercase English letters. (a-zA-Z) Number (0-9) Spaces、`!`、`#`、`$`、`%`、`&`、`(`、`)`、`+`、`-`、`:`、`;`、`<`、

Note：
Currently, string room number is only supported on Android，iOS，Flutter and Uniapp platforms. Support for other platforms such as Web and Mini Programs will be available in the future. Please stay tuned!

TUIVideoRenderParams

Video render parameters

Value	Type	Description
fillMode	TUIVideoRenderParamsFillMode	Video image fill mode
rotation	TUIVideoRenderParamsRotation	Video image rotation direction

TUINetworkQualityInfo

User network quality information

Value	Type	Description
userId	NSString	user ID
quality	NetworkQuality	network quality

TUIVideoEncoderParams

Video encoding parameters

Value	Type	Description
resolution	TUIVideoEncoderParamsResolution	Video resolution
resolutionMode	TUIVideoEncoderParamsResolutionMode	Video aspect ratio mode

TUICallMediaType

Call media type

Type	Value	Description
TUICallMediaTypeUnknown	0	Unknown
TUICallMediaTypeAudio	1	Audio call
TUICallMediaTypeVideo	2	Video call

TUICallRole

Call role

Type	Value	Description
TUICallRoleNone	0	Unknown
TUICallRoleCall	1	Caller（inviter）
TUICallRoleCalled	2	Callee（invitee）

TUICallStatus

Call status

Type	Value	Description
TUICallStatusNone	0	Unknown
TUICallStatusWaiting	1	The call is currently waiting
TUICallStatusAccept	2	The call has been accepted

TUICallScene

Call scene

Type	Value	Description

TUICallSceneGroup	0	Group call
TUICallSceneMulti	1	Anonymous group calling (not supported at this moment, please stay tuned).
TUICallSceneSingle	2	one to one call

TUICallIOSOfflinePushType

iOS offline push type

Type	Value	Description
TUICallIOSOfflinePushTypeAPNs	0	APNs
TUICallIOSOfflinePushTypeVoIP	1	VoIP

TUIAudioPlaybackDevice

Audio route

Type	Value	Description
TUIAudioPlaybackDeviceSpeakerphone	0	Speakerphone
TUIAudioPlaybackDeviceEarpiece	1	Earpiece

TUICamera

Front/Back camera

Type	Value	Description
TUICameraFront	0	Front camera
TUICameraBack	1	Back camera

TUINetworkQuality

Network quality

Type	Value	Description
TUINetworkQualityUnknown	0	Unknown
TUINetworkQualityExcellent	1	Excellent
TUINetworkQualityGood	2	Good

TUINetworkQualityPoor	3	Poor
TUINetworkQualityBad	4	Bad
TUINetworkQualityVbad	5	Vbad
TUINetworkQualityDown	6	Down

TUIVideoRenderParamsFillMode

If the aspect ratio of the video display area is not equal to that of the video image, you need to specify the fill mode:

Type	Value	Description
TUIVideoRenderParamsFillModeFill	0	Fill mode: the video image will be centered and scaled to fill the entire display area, where parts that exceed the area will be cropped. The displayed image may be incomplete in this mode.
TUIVideoRenderParamsFillModeFit	1	Fit mode: the video image will be scaled based on its long side to fit the display area, where the short side will be filled with black bars. The displayed image is complete in this mode, but there may be black bars.

TUIVideoRenderParamsRotation

We provides rotation angle setting APIs for local and remote images. The following rotation angles are all clockwise.

Type	Value	Description
TUIVideoRenderParamsRotation_0	0	No rotation
TUIVideoRenderParamsRotation_90	1	Clockwise rotation by 90 degrees
TUIVideoRenderParamsRotation_180	2	Clockwise rotation by 180 degrees
TUIVideoRenderParamsRotation_270	3	Clockwise rotation by 270 degrees

TUIVideoEncoderParamsResolutionMode

Video aspect ratio mode

Type	Value	Description
TUIVideoEncoderParamsResolutionModeLandscape	0	Landscape resolution, such as：TUIVideoEncoderParamsResolution_640_360

		TUIVideoEncoderParamsResolutionModeLands = 640 × 360
TUIVideoEncoderParamsResolutionModePortrait	1	Portrait resolution, such as： TUIVideoEncoderParamsResolution_640_360 - TUIVideoEncoderParamsResolutionModePortra 360 × 640

TUIVideoEncoderParamsResolution

Video resolution

Type	Value	Description
TUIVideoEncoderParamsResolution_640_360	108	Aspect ratio: 16:9；resolution: 640x360；recommended bitrate: 500kbps
TUIVideoEncoderParamsResolution_640_480	62	Aspect ratio: 4:3；resolution: 640x480；recommended bitrate: 600kbps
TUIVideoEncoderParamsResolution_960_540	110	Aspect ratio: 16:9；resolution: 960x540；recommended bitrate: 850kbps
TUIVideoEncoderParamsResolution_960_720	64	Aspect ratio: 4:3；resolution: 960x720；recommended bitrate: 1000kbps
TUIVideoEncoderParamsResolution_1280_720	112	Aspect ratio: 16:9；resolution: 1280x720；recommended bitrate: 1200kbps
TUIVideoEncoderParamsResolution_1920_1080	114	Aspect ratio: 16:9；resolution: 1920x1080；recommended bitrate: 2000kbps

Web
API Overview

Last updated：2024-08-09 22:25:01

TUICallKit (Includes UI Components)

TUICallKit is an audio and video call component that **includes a UI component**. You can quickly implement a WhatsApp-like audio and video calling scenario with this component.

API	Description
init	Initialize TUICallKit.
call	Makes a one-to-one call, Support for custom room ID, call timeout, offline push content, etc
groupCall	Makes a group call, Support for custom room ID, call timeout, offline push content, etc
joinInGroupCall	Join a group call.
setCallingBell	Customize user's ringtone.
setSelfInfo	Set your own nickname and avatar.
enableMuteMode	Turn on/off ringtone
enableFloatWindow	Activate/Deactivate the floating window function Turn on/off the floating window function
enableVirtualBackground	Turn on/off the blurred background function button. **v3.2.4+ support**
setLanguage	Set the call language for the TUICallKit component
hideFeatureButton	Hidden Button, **v3.2.9+ support**
setLocalViewBackgroundImage	Set the background image for the local user's call interface, **v3.2.9+ support**
setRemoteViewBackgroundImage	Set the background image for the remote user's call interface, **v3.2.9+ support**
setLayoutMode	Set the call interface layout mode, **supported from v3.3.0+**

setCameraDefaultState	Set whether the camera is opened by default, **supported from v3.3.0+**
destroyed	Terminating TUICallKit Destroy TUICallKit
getTUICallEngineInstance	Get TUICallEngine instance,**v2.4.3+ supported**

TUICallEngine (No UI)

TUICallEngine API is an audio and video call component that **offers a No UI interface**. You can use this set of APIs to custom encapsulate according to your business needs.

API	Description
createInstance	Creating a TUICallEngine Instance (Singleton Pattern)
destroyInstance	Terminating a TUICallEngine Instance (Singleton Pattern)
on	Listening on events
off	Canceling Event Listening
login	Sign in Interface
logout	Logout Interface
setSelfInfo	Configure the user's nickname and profile photo
call	Initiate a one-on-one call
groupCall	Group Chat Invitation Call
accept	Answer Calls
reject	Decline Call
hangup	End Calls
switchCallMediaType	Switch Audio and Video Calls
startRemoteView	Initiate Remote Screen Rendering
stopRemoteView	Stop Remote Screen Rendering
startLocalView	Start Local Screen Rendering, **Note: This will be deprecated; use openCamera instead**
stopLocalView	Stop Local Screen Rendering, **Note: This will be deprecated; use**

	closeCamera instead
openCamera	Enable the camera
closeCamara	Turn Off Camera
switchCamera	Switch between front and rear cameras, note: only supported on mobile devices. **v3.0.0+ supported**
openMicrophone	Enable Microphone
closeMicrophone	Turn off the microphone
setVideoQuality	Set video quality
getDeviceList	Access device list
switchDevice	Switch camera or microphone devices
enableAIVoice	Enable/disable AI noise reduction
enableMultiDeviceAbility	Turn on/off the multi-device login mode of TUICallEngine. **v2.1.1+ supported**
setBlurBackground	Switch/set background blur, **v3.0.6+ supported**
setVirtualBackground	Switch/set image background blur, **v3.0.6+ supported**

Event Types

TUICallEvent is the callback event class corresponding to TUICallEngine. Through this callback, you can listen to the callback events of interest.

EVENT	Description
TUICallEvent.ERROR	An error occurred during the call.
TUICallEvent.SDK_READY	This callback is received when the SDK enters the Ready State
TUICallEvent.KICKED_OUT	Duplicate Sign in, receiving this callback indicates being Kicked out of Room
TUICallEvent.USER_ACCEPT	If a user answers, this callback will be received
TUICallEvent.USER_ENTER	A user joined the call.

TUICallEvent.USER_LEAVE	A user left the call.
TUICallEvent.REJECT	A user declined the call.
TUICallEvent.NO_RESP	A user didn't respond.
TUICallEvent.LINE_BUSY	A user was busy.
TUICallEvent.USER_VIDEO_AVAILABLE	Whether a user has a video stream.
TUICallEvent.USER_AUDIO_AVAILABLE	Whether a user has an audio stream.
TUICallEvent.USER_VOICE_VOLUME	The volume levels of all users.
TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE	Group Chat Update, Invitation List this callback will be received
TUICallEvent.ON_CALL_BEGIN	Call connected event, **v1.4.6+ supported**
TUICallEvent.INVITED	A call was received. **It will be discarded later and it is recommended to use TUICallEvent.ON_CALL_RECEIVED**
TUICallEvent.ON_CALL_RECEIVED	Call request event, **v1.4.6+ supported**
TUICallEvent.CALLING_CANCEL	Call cancellation event, **It will be abandoned later and it is recommended to use TUICallEvent.ON_CALL_CANCELED**
TUICallEvent.ON_CALL_CANCELED	Call canceled event, **v1.4.6+ supported**
TUICallEvent.CALLING_END	The call ended.
TUICallEvent.DEVICED_UPDATED	Device list update, this callback will be received
TUICallEvent.CALL_TYPE_CHANGED	This callback is received when switching call types
TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED	All user network quality events, **v3.0.7+ supported**

Document Link

TUICallEngine

TUICallEvent

TUICallKit

Last updated：2024-08-09 22:25:01

API Introduction

The TUICallKit API is the **audio and video call component that includes a UI interface**. With the TUICallKit API, you can swiftly develop audio and video call scenarios reminiscent of WeChat through simple interfaces. For further comprehensive steps to access this, please refer to: Integrating TUICallKit.

API Overview

TUICallKit is the **audio and video call component with a UI**, which enables you to swiftly create scenarios akin to WeChat for voice and video calls.
<TUICallKit/>: The core UI call component.
TUICallKitServer is the call instance, offering the following API interfaces.

API	Description
init	Initialize TUICallKit.
call	Makes a one-to-one call, Support for custom room ID, call timeout, offline push content, etc
groupCall	Makes a group call, Support for custom room ID, call timeout, offline push content, etc
joinInGroupCall	Join a group call.
setCallingBell	Customize user's ringtone.
setSelfInfo	Set your own nickname and avatar.
enableMuteMode	Turn on/off ringtone
enableFloatWindow	Turn on/off the floating window function
enableVirtualBackground	Turn on/off the blurred background function button. **v3.2.4+ support**
setLanguage	Set the call language for the TUICallKit component
hideFeatureButton	Hidden Button, **v3.2.9+ support**

setLocalViewBackgroundImage	Set the background image for the local user's call interface, **v3.2.9+ support**
setRemoteViewBackgroundImage	Set the background image for the remote user's call interface, **v3.2.9+ support**
setLayoutMode	Set the call interface layout mode, **v3.3.0+ support**
setCameraDefaultState	et whether the camera is opened by default, **v3.3.0+ support**
destroyed	Destroy TUICallKit
getTUICallEngineInstance	Get TUICallEngine instance,**v2.4.3+ supported**

Introduction to `<TUICallKit/>` attributes

Attribute Overview

Attribute	Description	Type	Required	Default Value
allowedMinimized	Is the floating window permitted?	boolean	No	false
allowedFullScreen	Whether to permit full screen mode for the call interface	boolean	No	true
videoDisplayMode	Display mode for the call interface	VideoDisplayMode	No	VideoDisplayMode.COVER
videoResolution	Call Resolution	VideoResolution	No	VideoResolution.RESOLUTION_48
beforeCalling	This function will be executed prior to making a call and before	function(type, error)	No	-

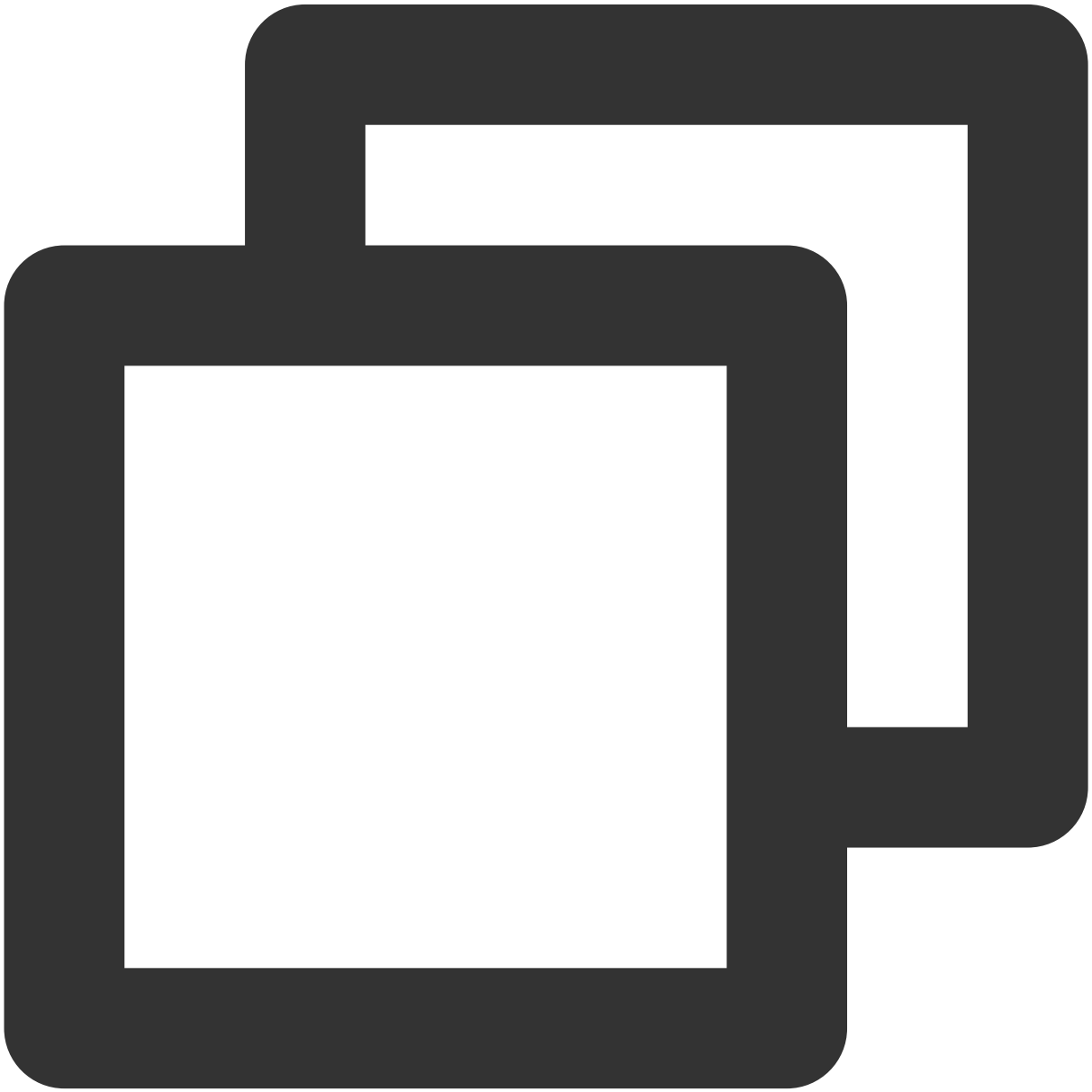
	receiving an invitation to talk			
afterCalling	This function will be executed after the termination of the call	function()	No	-
onMinimized	This function will be executed when the component switches to a minimized state. The explanation for the STATUS value is	function(oldStatus, newStatus)	No	-
kickedOut	The events thrown by the component occur when the current logged-in user is ejected. The call will also automatically terminate	function()	No	-
statusChanged	Event thrown by the component; this event is triggered when the call status changes. For detailed	function({oldStatus, newStatus})	No	-

	types of call status, refer to STATUS value description			

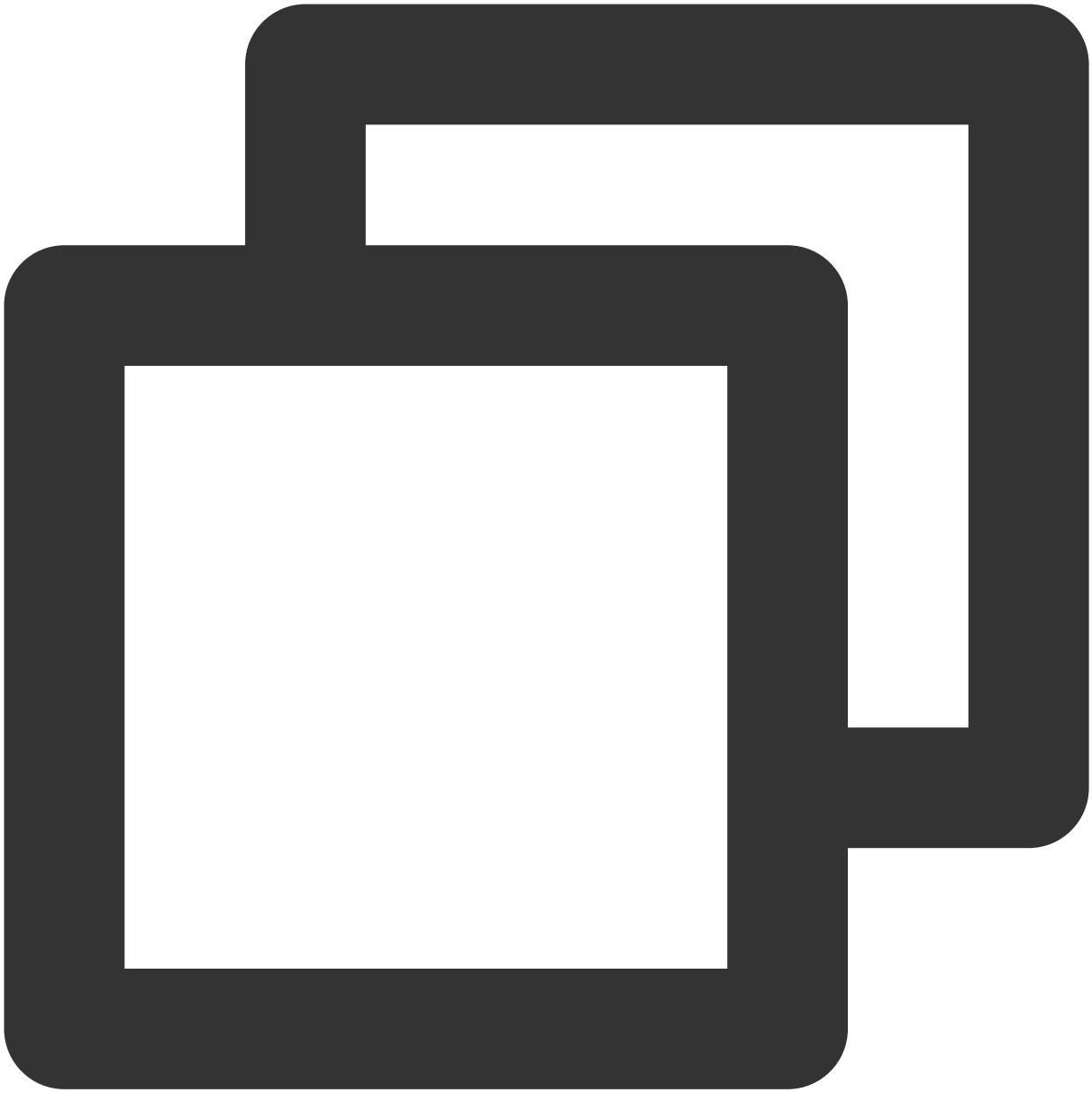
Sample code

React

Vue



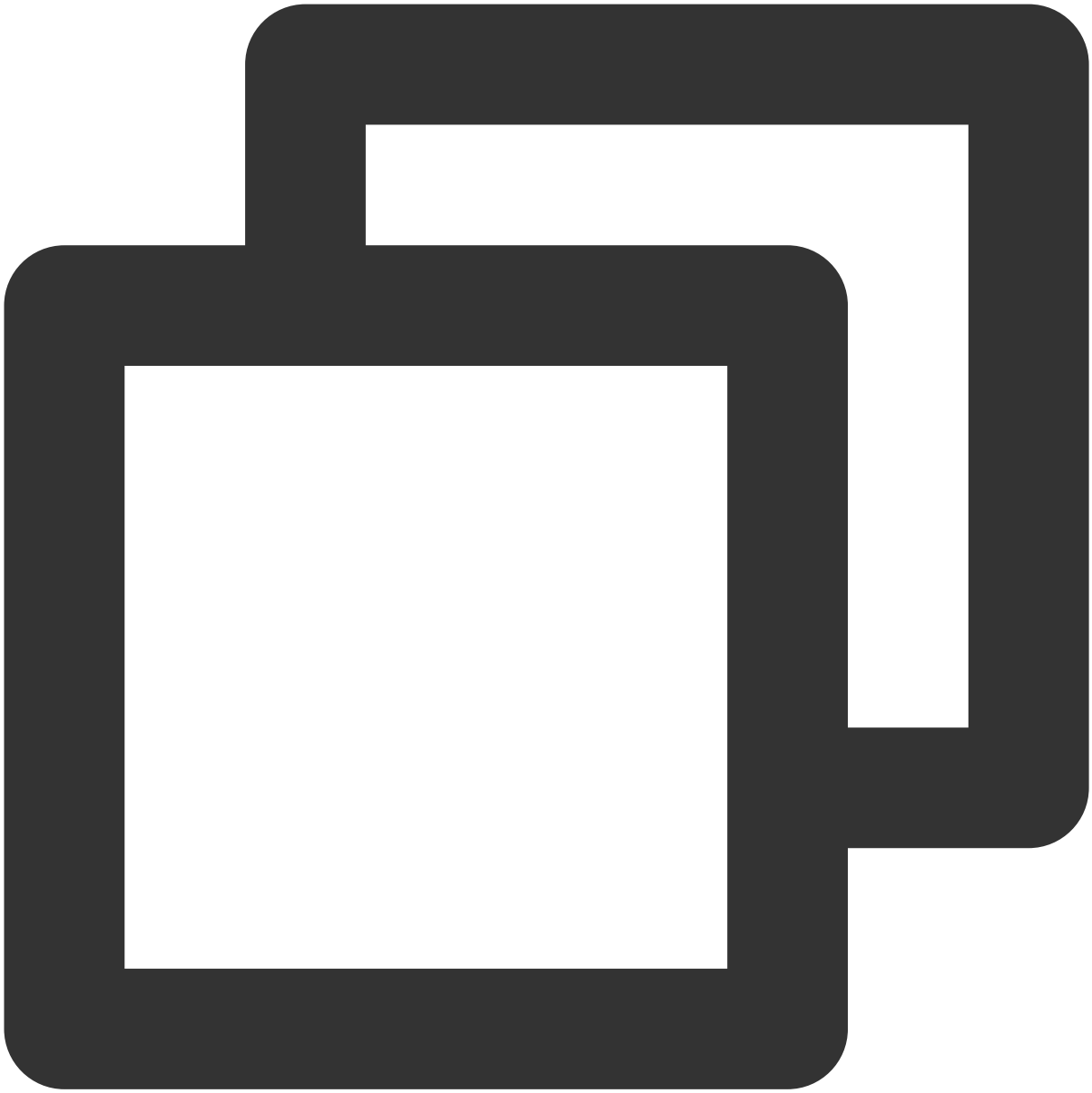
```
import { TUICallKit, VideoDisplayMode, VideoResolution } from "@tencentcloud/call-u

<TUICallKit
    videoDisplayMode={VideoDisplayMode.CONTAIN}
    videoResolution={VideoResolution.RESOLUTION_1080P}
    beforeCalling={handleBeforeCalling}
    afterCalling={handleAfterCalling}
/>

function handleBeforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] handleBeforeCalling:", type, error);
}
function handleAfterCalling() {
  console.log("[TUICallkit Demo] handleAfterCalling");
}
```



```
<template>
   <TUICallKit
    :allowedMinimized="true"
    :allowedFullScreen="true"
    :videoDisplayMode="VideoDisplayMode.CONTAIN"
    :videoResolution="VideoResolution.RESOLUTION_1080P"
    :beforeCalling="beforeCalling"
    :afterCalling="afterCalling"
    :onMinimized="onMinimized"
    :kickedOut="handleKickedOut"
    :statusChanged="handleStatusChanged"
```

```
  />
</template>
<script lang="ts" setup>
import { TUICallKit, TUICallKitServer, VideoDisplayMode, VideoResolution, STATUS }

function beforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] beforeCalling:", type, error);
}
function afterCalling() {
  console.log("[TUICallkit Demo] afterCalling");
}
function onMinimized(oldStatus: string, newStatus: string) {
  console.log("[TUICallkit Demo] onMinimized: " + oldStatus + " -> " + newStatus);
}
function kickedOut() {
  console.log("[TUICallkit Demo] kickedOut");
}
function statusChanged(args: { oldStatus: string; newStatus: string; }) {
  const { oldStatus, newStatus } = args;
  if (newStatus === STATUS.CALLING_C2C_VIDEO) {
    console.log(`[TUICallkit Demo] statusChanged: ${oldStatus} -> ${newStatus}`);
  }
}
</script>
```

Detailed information on TUICallKitServer API



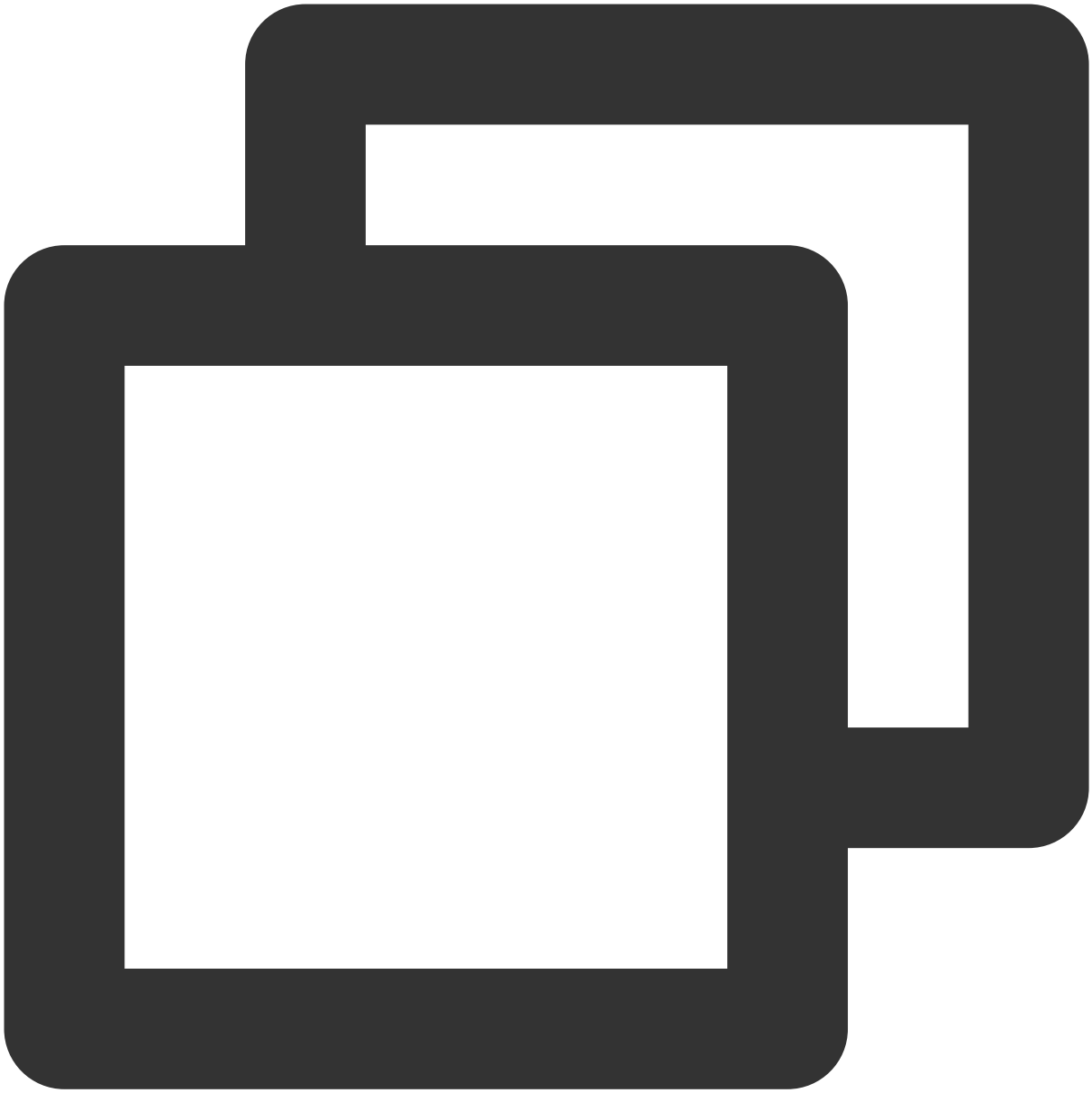
```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";
// Replace it with the call-uikit npm package you are currently using
```

init

Initialize TUICallKit.

Note：

Init initialization needs to be completed before functions such as call and groupCall can be used.



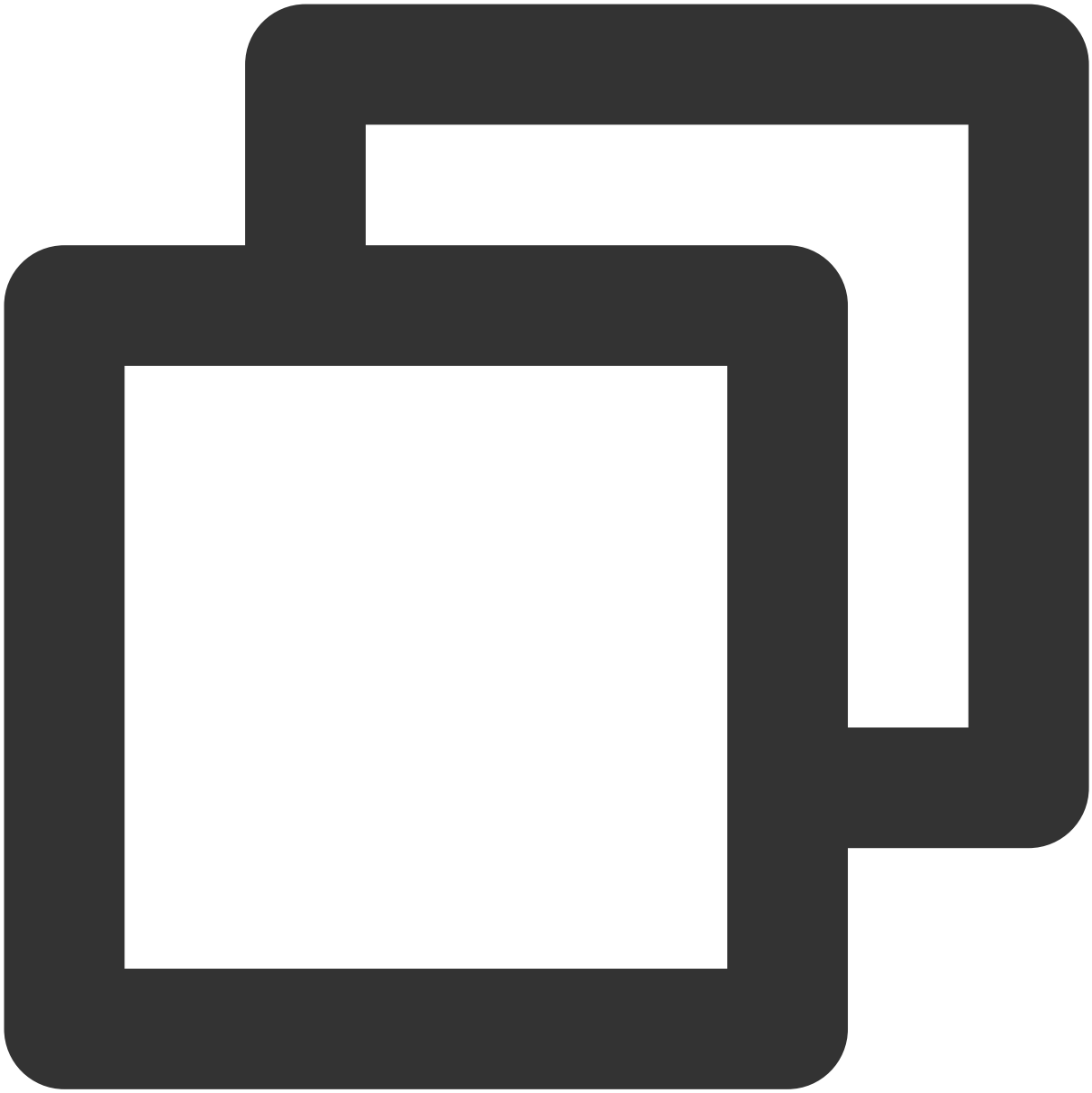
```
try {
  await TUICallKitServer.init({ SDKAppID, userID, userSig });
  // If you already have a tim instance in your project, you need to pass it in her
  // await TUICallKitServer.init({ tim, SDKAppID, userID, userSig});
  console.log("[TUICallKit] Initialization succeeds.");
} catch (error: any) {
  console.error(`[TUICallKit] Initialization failed. Reason: ${error}`);
}
```

The parameters are described below:

Parameter	Type	Required	Meaning
SDKAppID	Number	Yes	The SDKAppID of your app, you can find your SDKAppID in the Live Audio and Video console. For details, see Activating Services
userID	String	Yes	The current user's ID is of string type, only allowing for the inclusion of English letters (a-z and A-Z), digits (0-9), hyphens (-) and underscores (_)
userSig	String	Yes	Use SDKSecretKey to encrypt SDKAppID, UserID and other information to obtain userSig. It is an authentication ticket used by Tencent Cloud to identify whether the current user can use TRTC services. For how to obtain it, please refer to How to Calculate UserSig
tim	TencentCloudChat	No	tim is an instance of TencentCloudChat SDK.

call

Makes a one-to-one call, Support for custom room ID, call timeout, offline push content, etc.



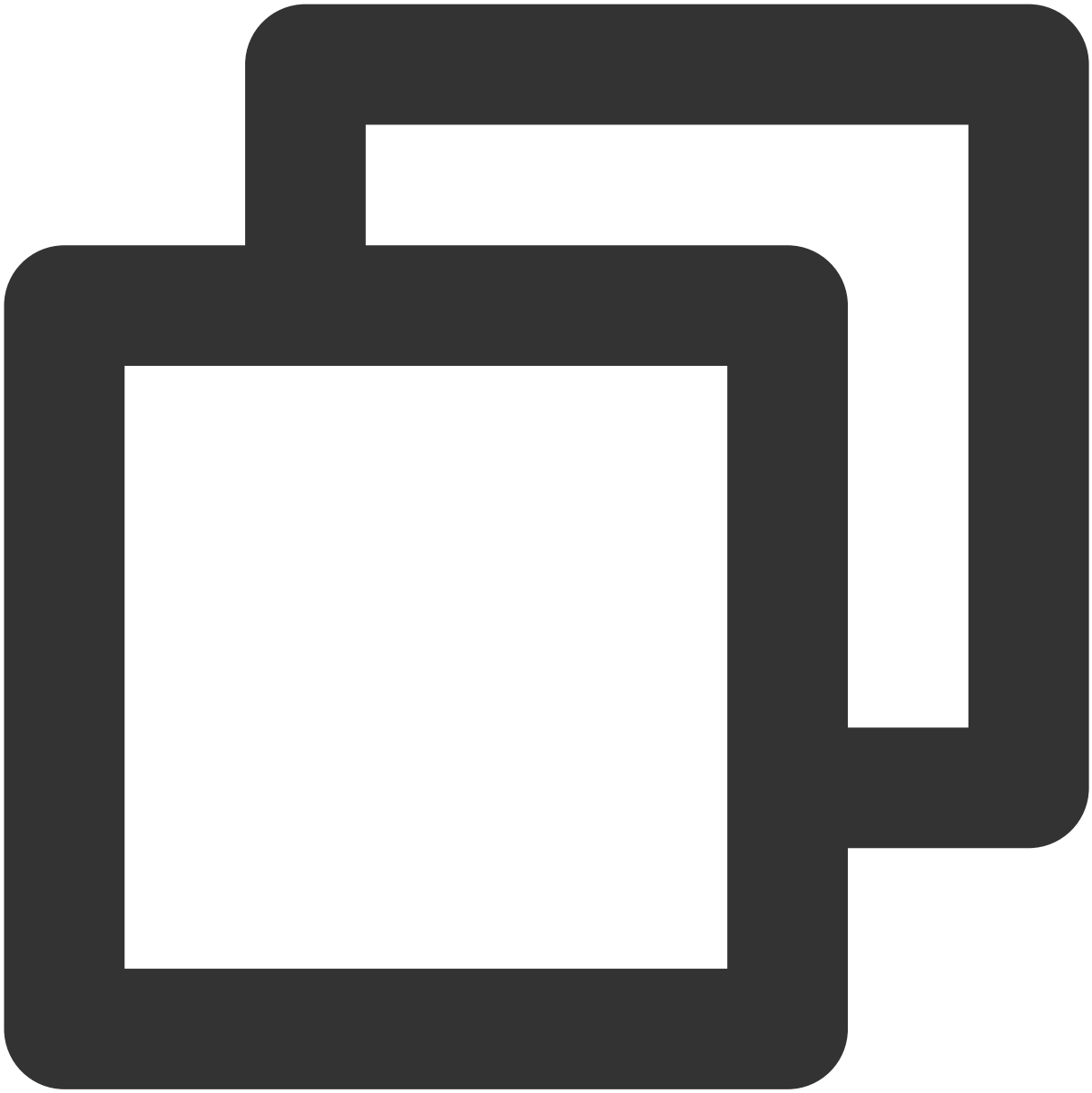
```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-react';
try {
  await TUICallKitServer.call({
    userID: 'mike',
    type: TUICallType.VIDEO_CALL,
  });
} catch (error: any) {
  console.error(`[TUICallKit] Call failed. Reason: ${error}`);
}
```

The parameters are described below:

Parameter	Type	Required	Meaning
userID	String	Yes	The username of the called user
type	TUICallType	Yes	The media type of the call, see TUICallType call type for parame
roomID	Number	No	Numerical Room ID, range [1, 2147483647]
strRoomID	String	No	String room ID. **v3.3.1+ supported** **range：** Limited to 64 bytes in length. The supported character set range Lowercase and uppercase English letters.（a-zA-Z）； Number（0-9）； `Spaces`、`!`、`#`、`$`、`%`、`&`、`(`、`)`、`+` 。 1. roomID and strRoomID are mutually exclusive, if you use strRo 2. don't mix roomID and strRoomID, because they are not interch
timeout	Number	No	Call timeout, default: 30s, unit: seconds. timeout = 0, set to no tim
userData	String	No	Customize the extended fields when initiating a call. The called u
offlinePushInfo	Object	No	Customize offline message push parameters

groupCall

Makes a group call, Support for custom room ID, call timeout, offline push content, etc.



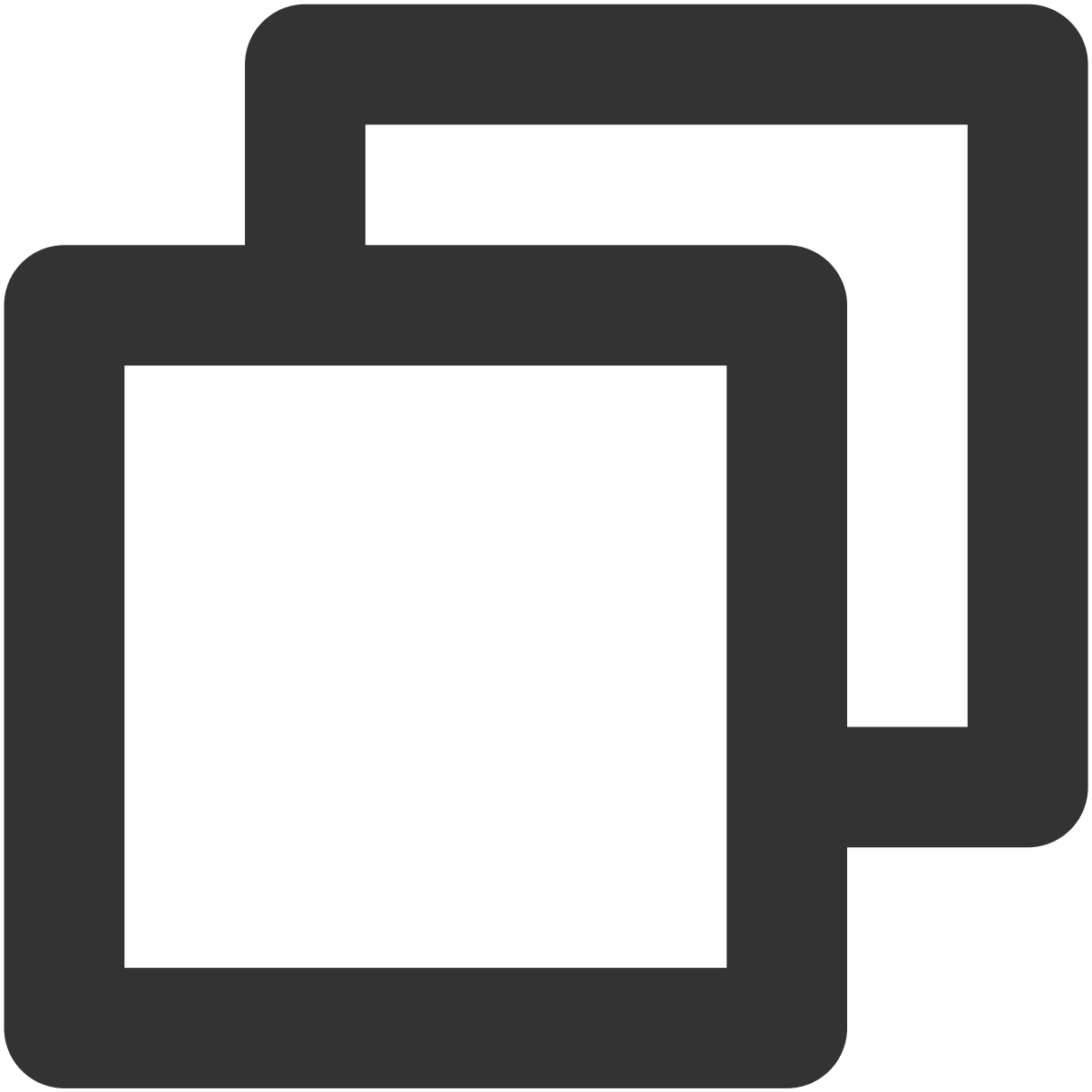
```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-react';
try {
  await TUICallKitServer.groupCall({
    userIDList: ['jack', 'tom'],
    groupID: "xxx",
    type: TUICallType.VIDEO_CALL
  });
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the groupCall API. Reason:${error}`);
}
```

The parameters are described below:

Parameter	Type	Required	Meaning
userIDList	Array<String>	Yes	List of called users
type	TUICallType	Yes	The type of media for the call, you can refer to TUICallType for
groupID	String	Yes	Call group ID, the creation of groupID can be referred to chat-cr
roomID	Number	No	Numerical Room ID, range [1, 2147483647]
strRoomID	String	No	String room ID. **v3.3.1+ supported** **range：** Limited to 64 bytes in length. The supported character set range Lowercase and uppercase English letters.（a-zA-Z）； Number（0-9）； `Spaces`、`!`、`#`、`$`、`%`、`&`、`(`、`)`、`+` 。 1. roomID and strRoomID are mutually exclusive, if you use strF 2. don't mix roomID and strRoomID, because they are not interc
timeout	Number	No	Call timeout, default: 30s, unit: seconds. timeout = 0, set to no ti
userData	String	No	Customize the extended fields when initiating a call. The called
offlinePushInfo	Object	No	Customize offline message push parameters

setLanguage

Set language, currently supports: Chinese, English, Japanese.



```
TUICallKitServer.setLanguage("zh-cn"); // "en" | "zh-cn" | "ja_JP"
```

The parameters are described below:

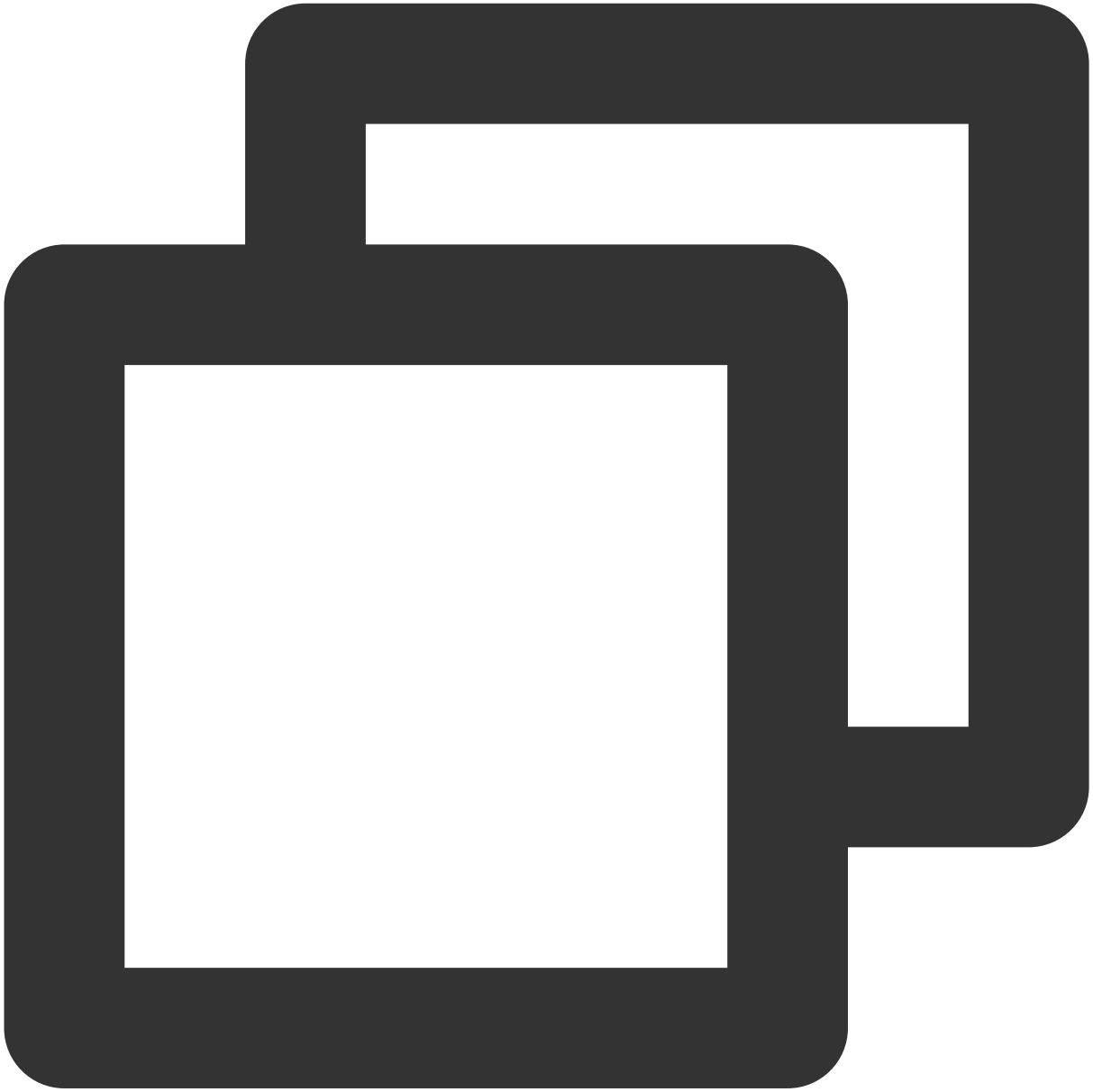
Parameter	Type	Required	Meaning
lang	String	Yes	Language type `en` , `zh-cn` and `ja_JP` .

setSelfInfo

Set your own nickname and avatar.

Note:

v2.2.0+ supported. If you use this interface to modify user information during a call, the UI will not be updated immediately, and you will need to wait until the next call to see the changes.



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

The parameters are described below:

Parameter	Type	Required	Meaning
nickName	String	Yes	own nickname
avatar	String	Yes	own avatar address

setCallingBell

Note:

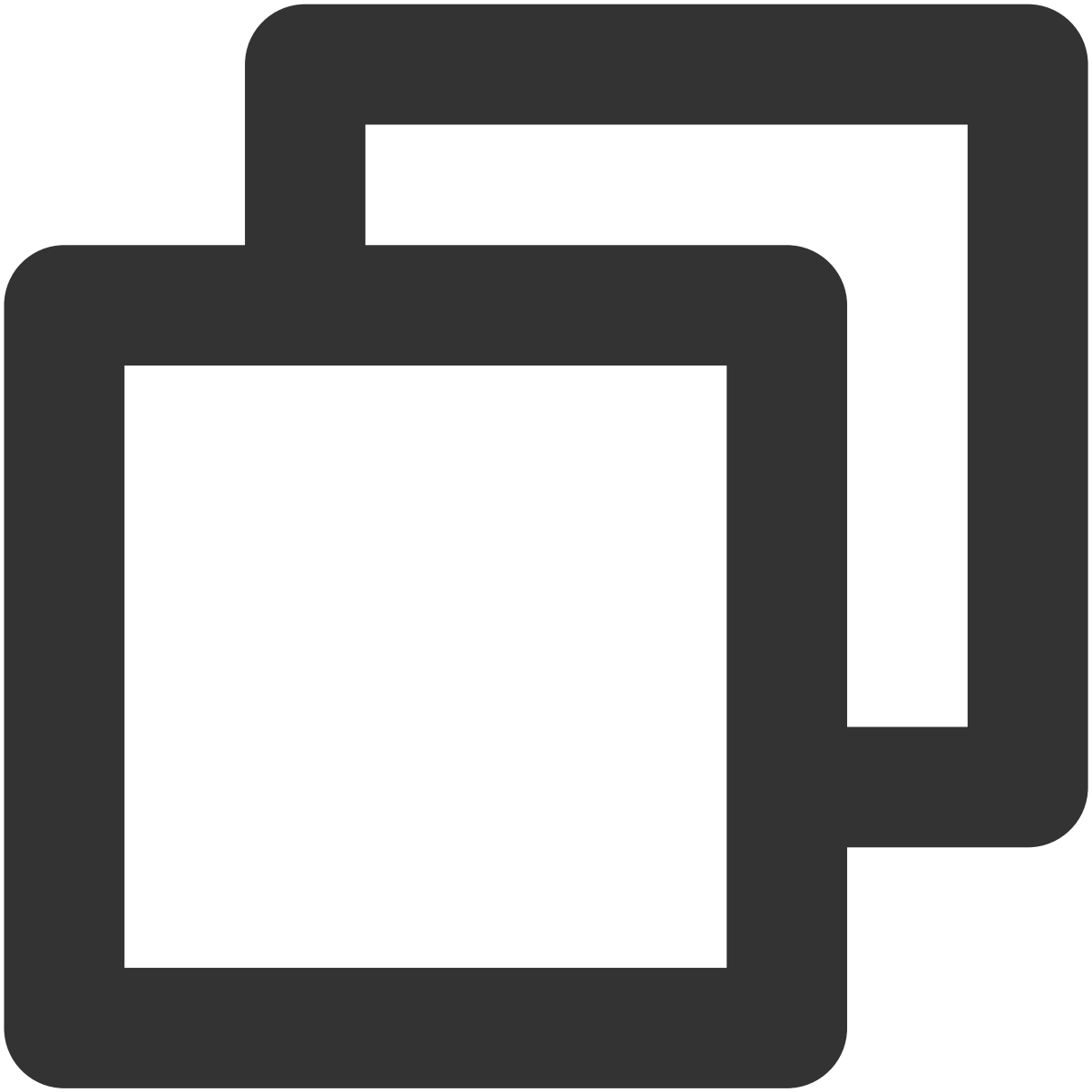
v3.0.0+ supported.

Customize the user's incoming call ringtone.

The input is restricted to the local MP3 format file address. It is imperative to ensure that the application has access to this file directory.

Use the import method to import the ringtone file.

If you need to restore the default ringtone, just pass empty filePath.



```
import filePath from '../assets/phone_ringing.mp3';
try {
  await TUICallKitServer.setCallingBell(filePath);
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the setCallingBell API. Reason: ${erro
}
```

The parameters are described below:

Parameter	Type	Required	Meaning

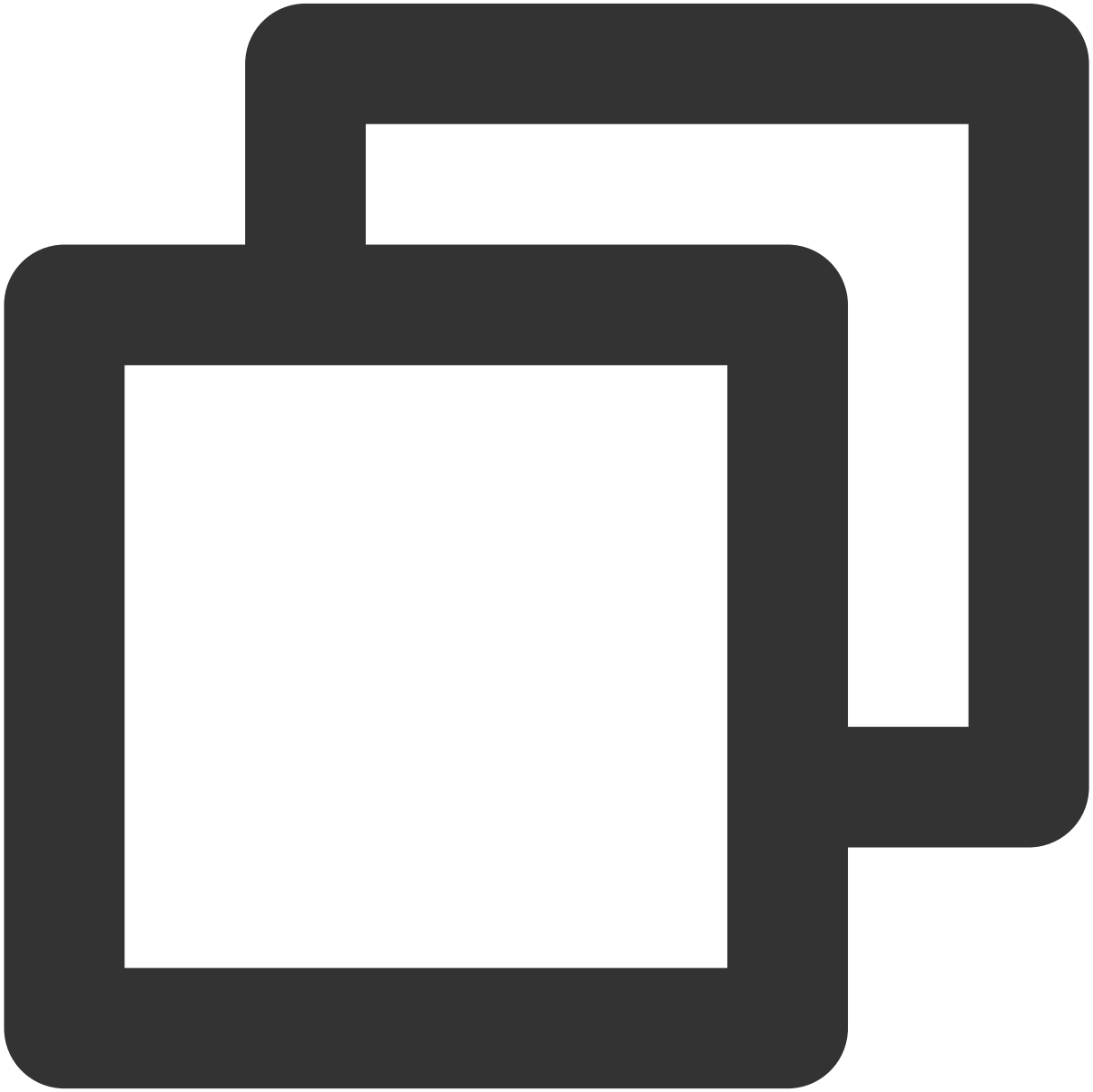
filePath	String	Yes	Ringtone file path

enableFloatWindow

Turn on/off the floating window function. The default is false. The floating window button in the upper left corner of the call interface is hidden. It will be displayed after setting it to true.

Note:

v3.1.0+ supported.



```
try {
```

```
    const enable = true;
    await TUICallKitServer.enableFloatWindow(enable);
  } catch (error: any) {
    console.error(`[TUICallKit] Failed to call the enableFloatWindow API. Reason: ${e
  }
```

The parameters are described below:

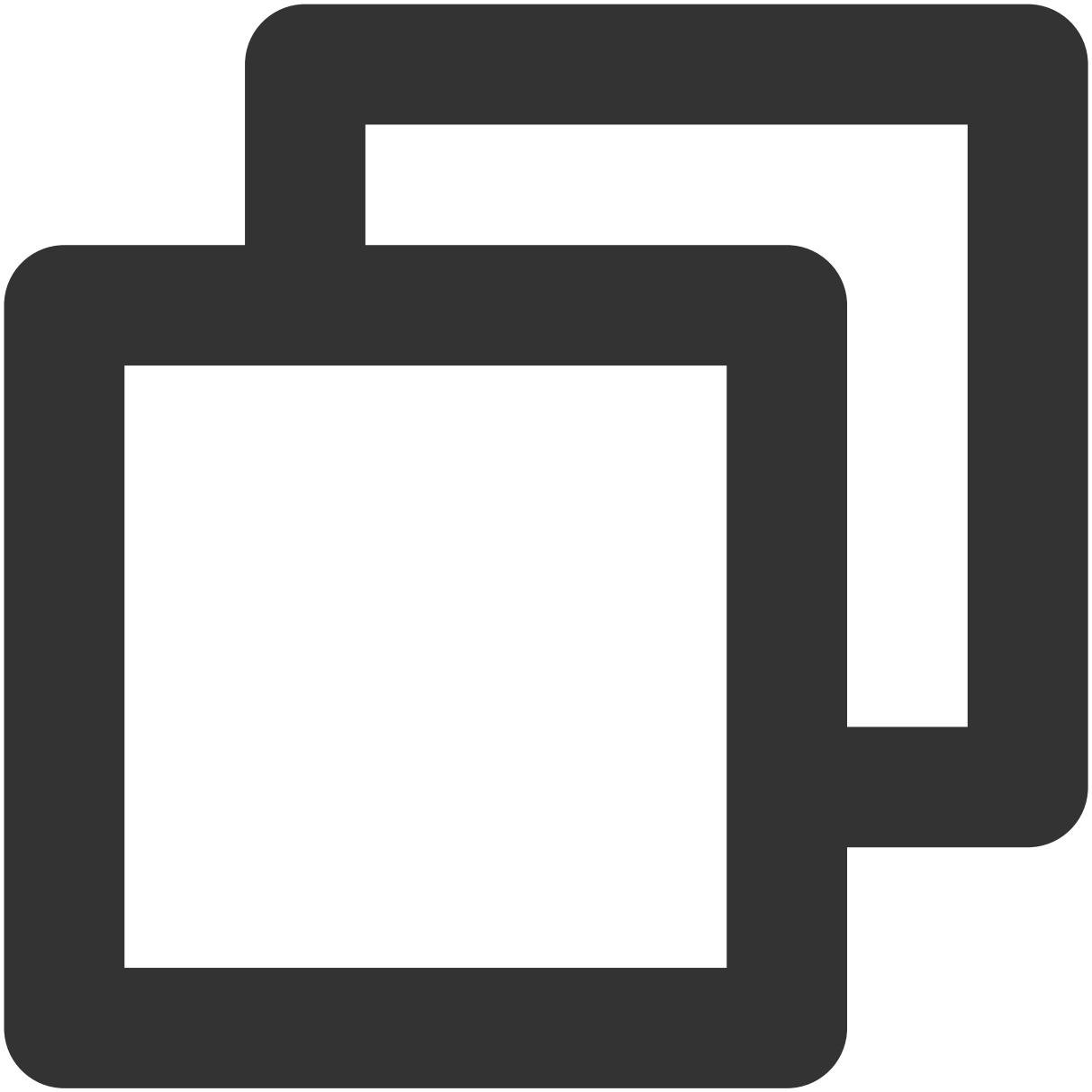
Parameter	Type	Required	Meaning
enable	Boolean	Yes	Turn on/off the floating window function. default false.

enableMuteMode

Turn on/off the ringtone for incoming calls. When turned on, the incoming call ringtone will not be played when a call request is received.

Note:

v3.1.2+ supported.



```
try {
  const enable = true;
  await TUICallKitServer.enableMuteMode(enable);
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${erro
}
```

joinInGroupCall

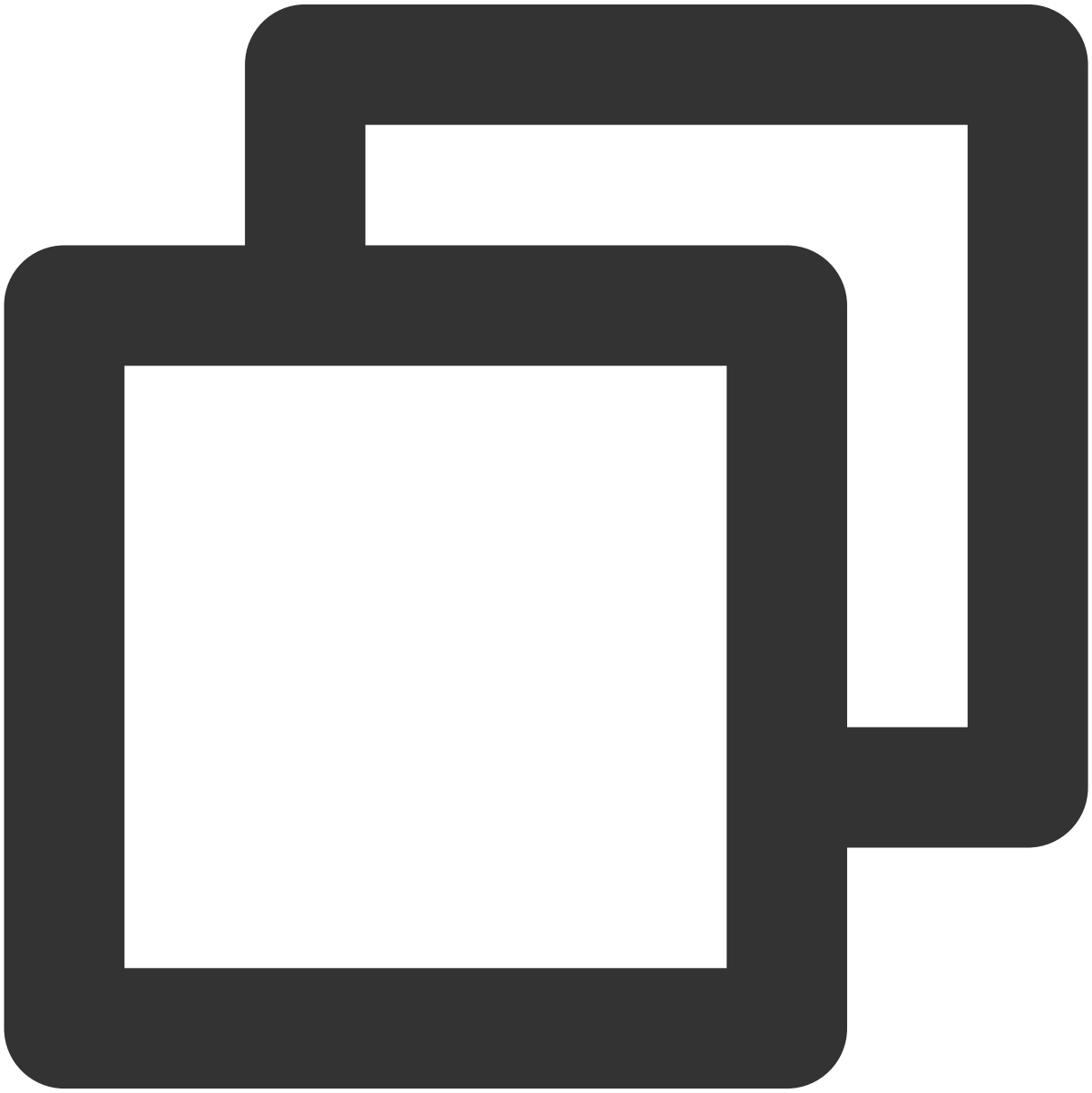
Join an existing audio-video call in a group.

Note:

v3.1.2+ supported.

Note:

Before joining an existing audio-video call in the group, an Chat group must be pre-established or joined, and users in the group must already be engaged in a call. If the group has already been formed, please ignore this requirement. Instructions for creating a group can be found at Chat Group Management. Alternatively, you may directly utilize Chat TUIKit for an all-in-one integration of chat, call and other scenarios.



```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-react';
try {
```

```
    const params = {
      type: TUICallType.VIDEO_CALL,
      groupID: "xxx",
      roomID: 234,
    };
    await TUICallKitServer.joinInGroupCall(params);
  } catch (error: any) {
    console.error(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${erro
  }
```

The parameters are described below:

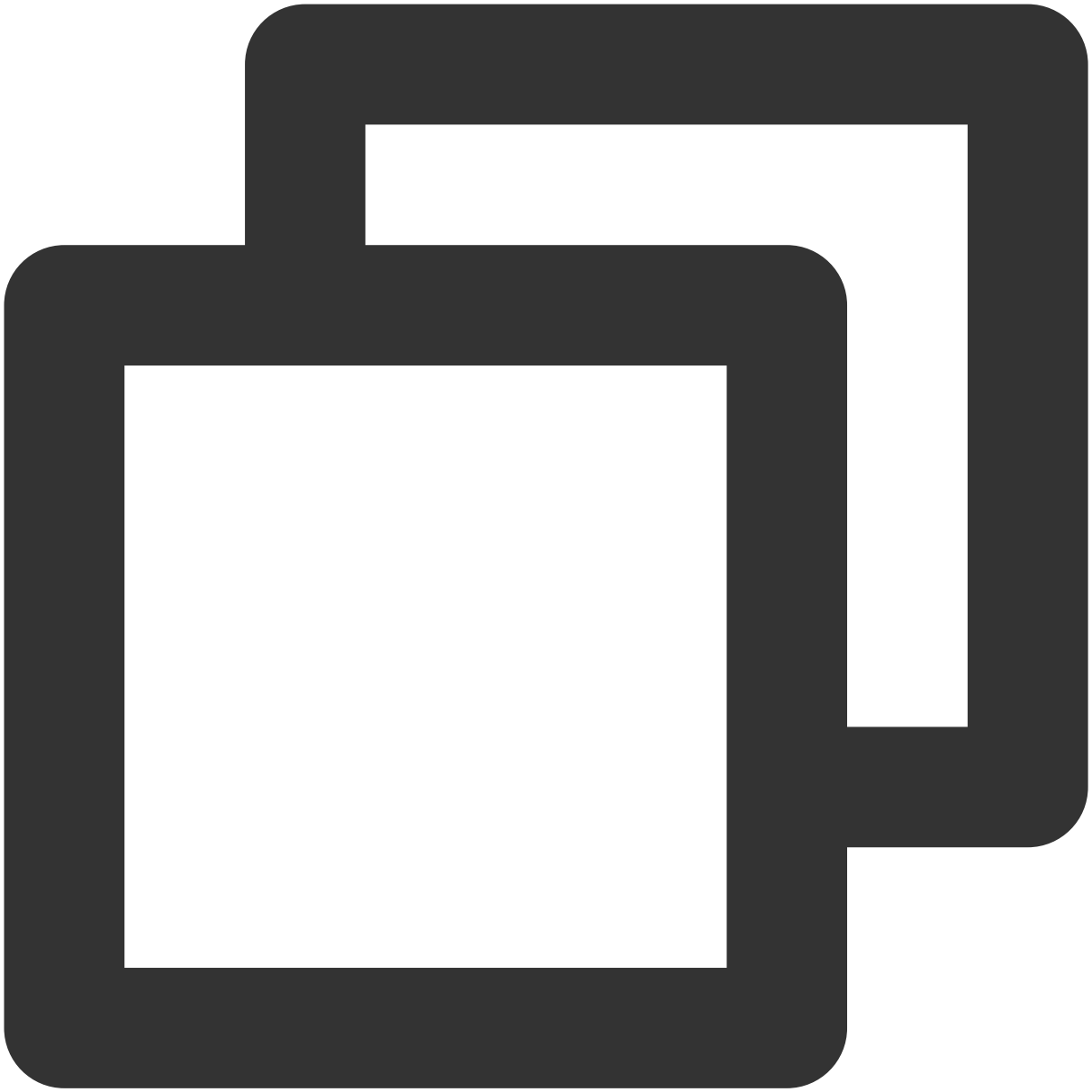
Parameter	Type	Required	Meaning
type	TUICallType	Yes	The media type of communication, for instance, video calls, voice calls
groupID	string	Yes	The group ID for this group call
roomID	number	Yes	Audio and video room ID for this call

enableVirtualBackground

Turn on/off the blurred background function. If you want to set the picture background to be blurry see Web. By calling the interface, you can display the blurred background function button on the UI, and click the button to directly enable the blurred background function.

Note：

v3.2.4+ supported.



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";
const enable = true;
TUICallKitServer.enableVirtualBackground(enable);
```

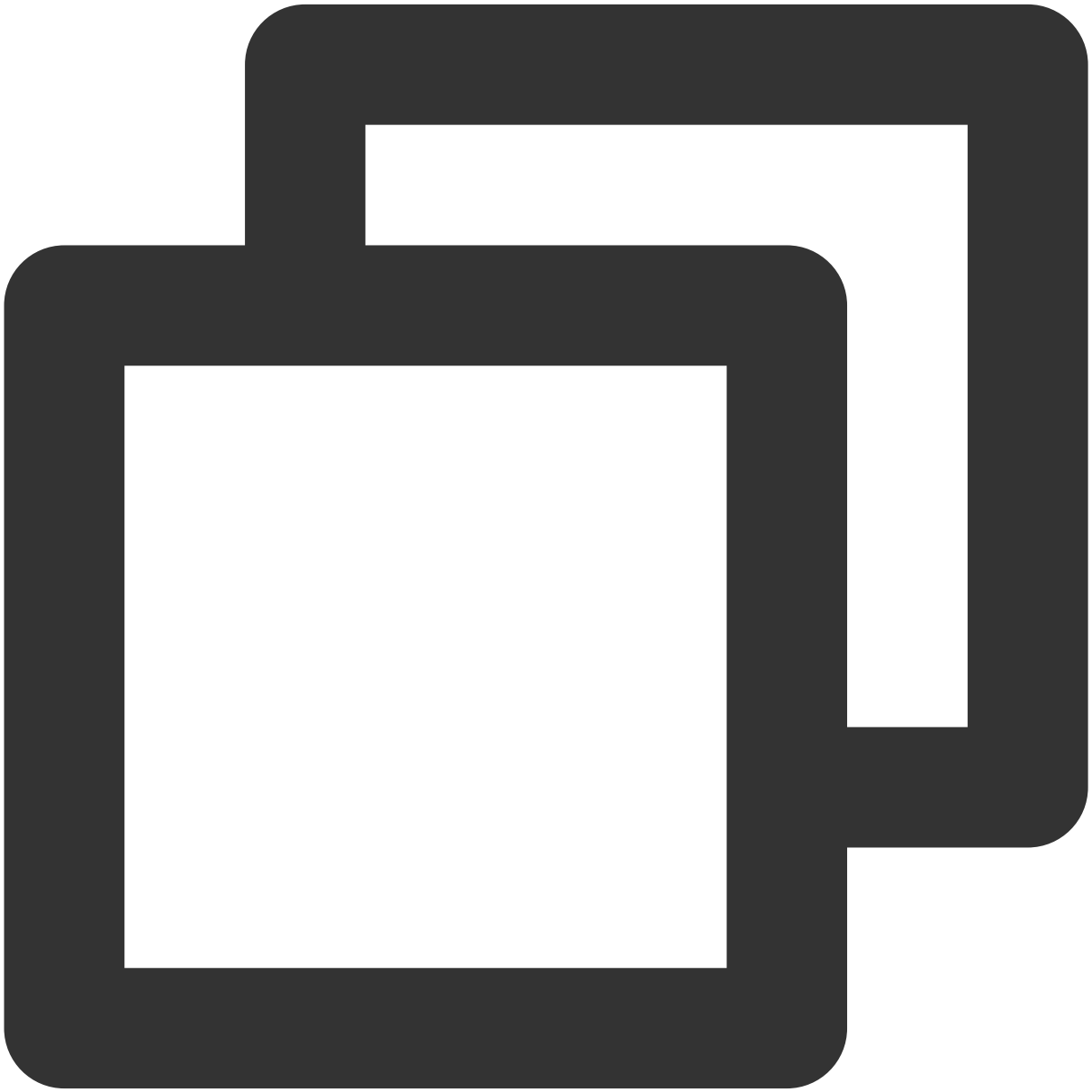
The parameters are described below:

Parameter	Type	Required	Meaning
enable	boolean	Yes	enable = true, show blur background button enable = false, don't show blur background button

destroyed

Terminate the TUICallKit instance.

This method won't automatically log out of `tim`, manual logging out is required: `tim.logout();`.



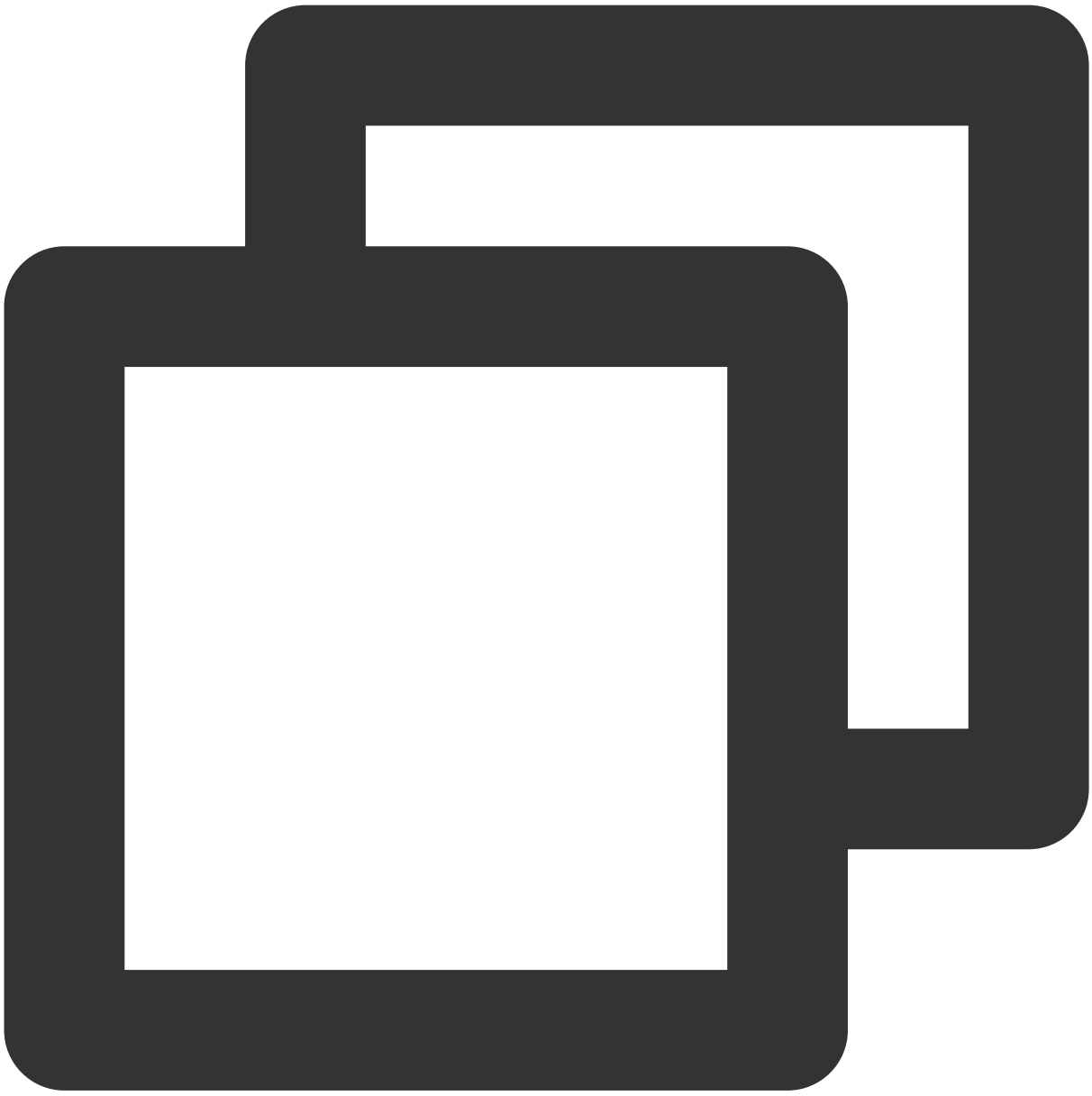
```
try {
  await TUICallKitServer.destroyed();
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the destroyed API. Reason: ${error}`);
}
```

hideFeatureButton

Hidden feature buttons, currently only support Camera, Microphone, and Switch Camera Button.

Note：

v3.2.9+ supported.



```
TUICallKitServer.hideFeatureButton(buttonName: FeatureButton);
```

The parameters are described below:

Parameter	Type	Required	Meaning

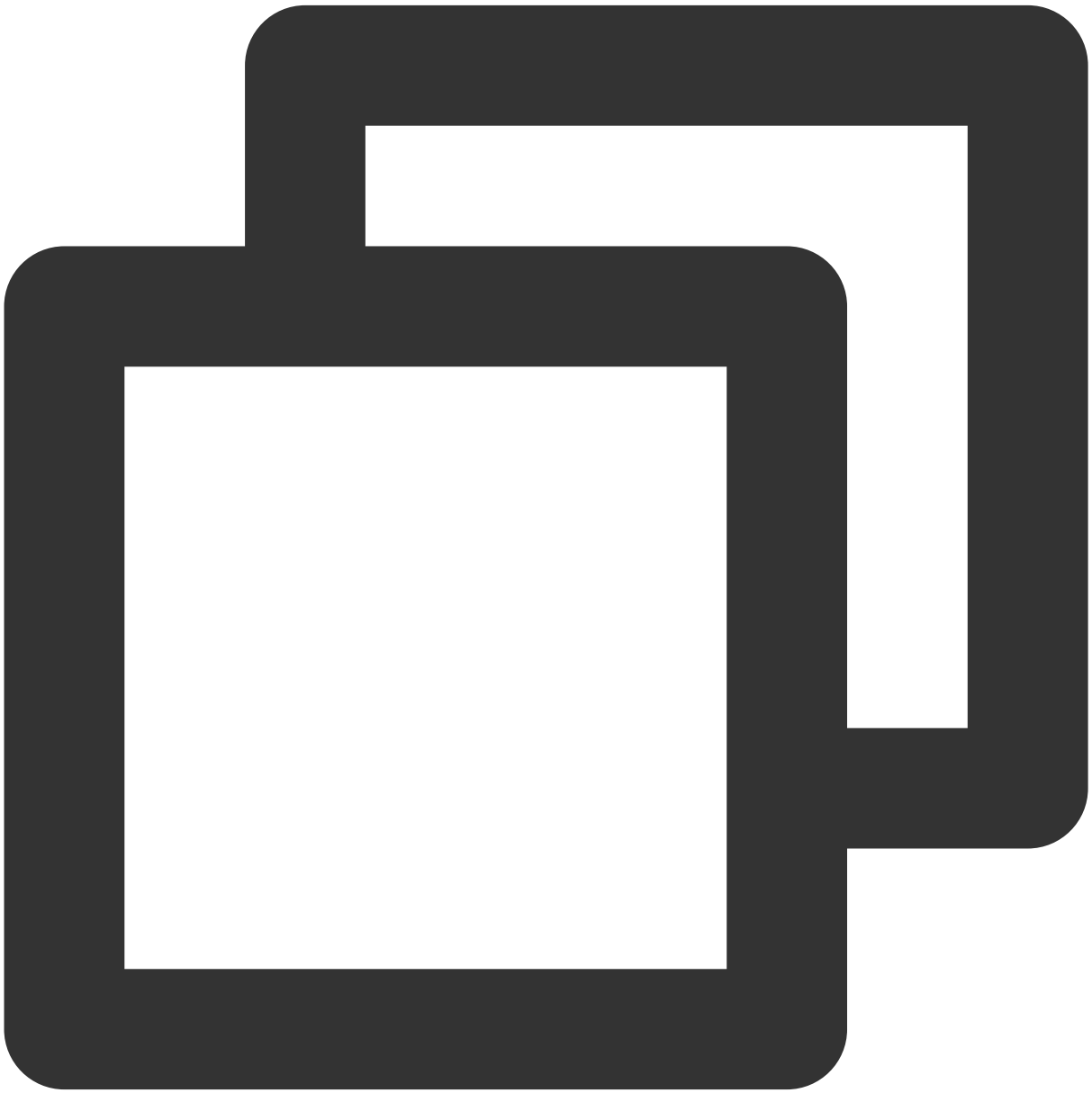
buttonName	FeatureButton	Yes	Button Name

setLocalViewBackgroundImage

Set the background image for the local user's call interface.

Note：

v3.2.9+ supported.



```
TUICallKitServer.setLocalViewBackgroundImage(url: string);
```

The parameters are described below:

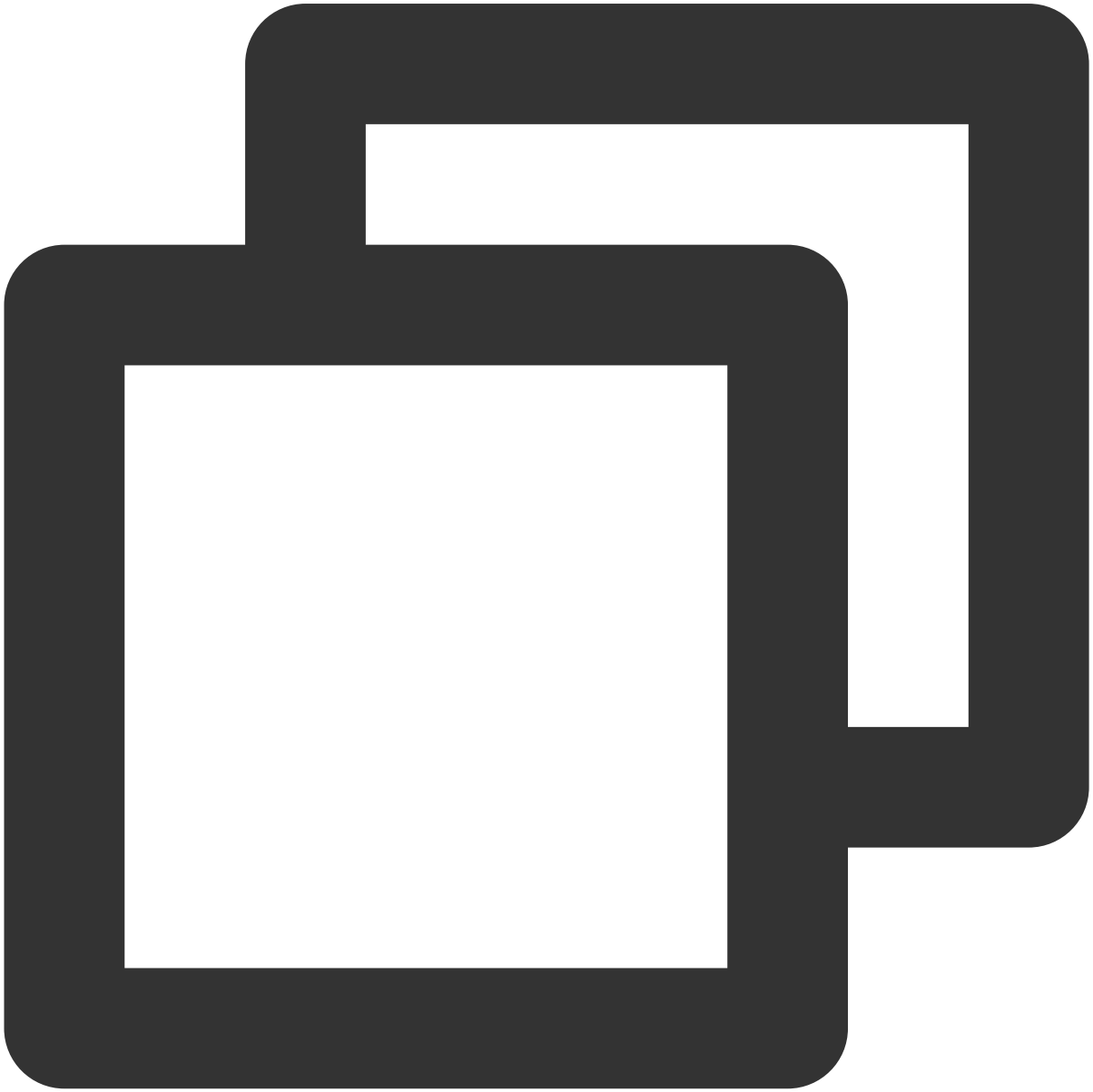
Parameter	Type	Required	Meaning
url	string	Yes	Image Address (supports Local Path and Network Address)

setRemoteViewBackgroundImage

Set the background image for the remote user's call interface.

Note：

v3.2.9+ supported.



```
TUICallKitServer.setRemoteViewBackgroundImage(userId: string, url: string);
```

The parameters are described below:

Parameter	Type	Required	Meaning
userId	string	Yes	Remote User userId, setting to '*' means it applies to all Remote Users
url	string	Yes	Image Address (supports Local Path and Network Address)

setLayoutMode

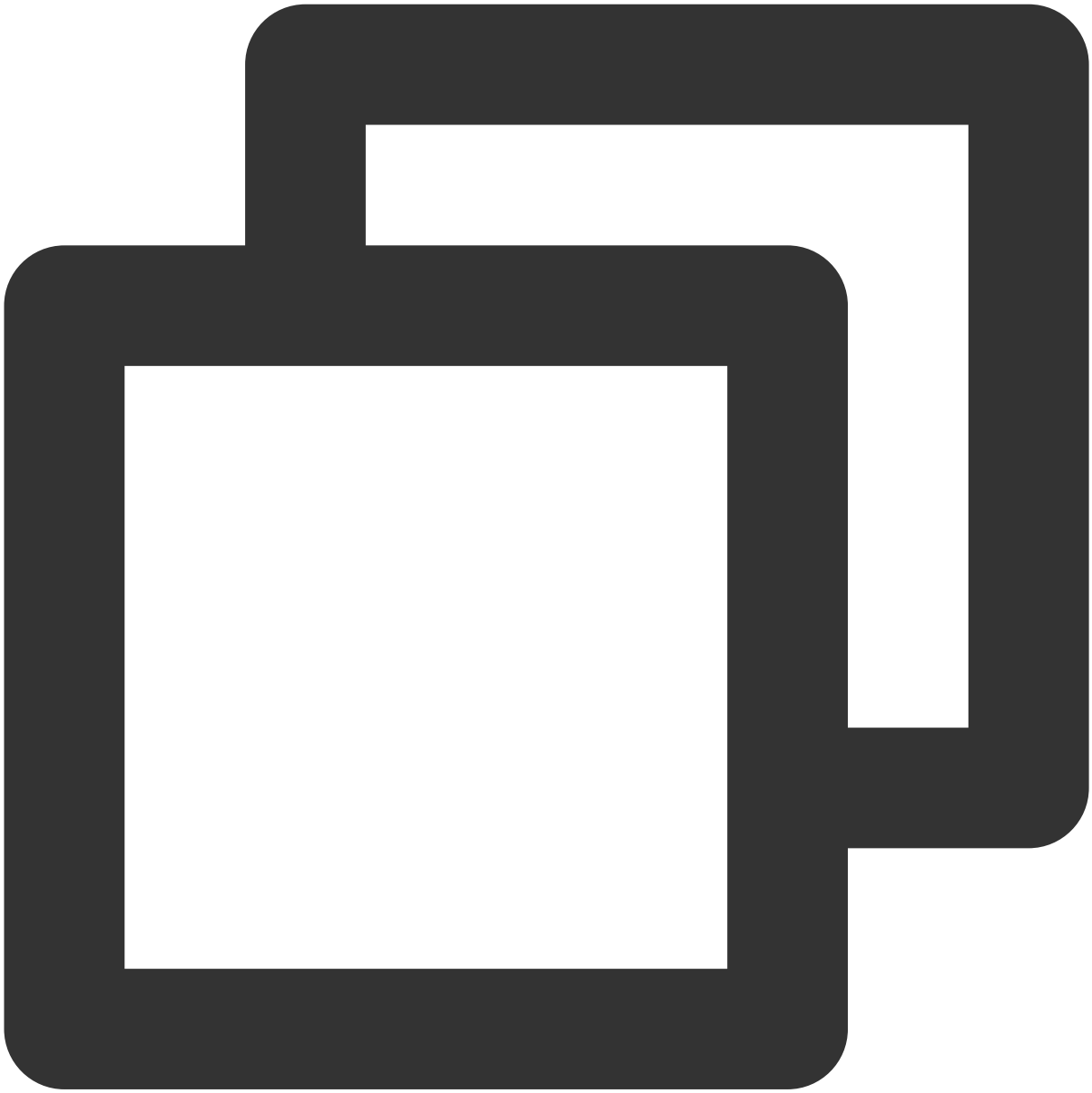
Set the call interface layout mode.

Note：

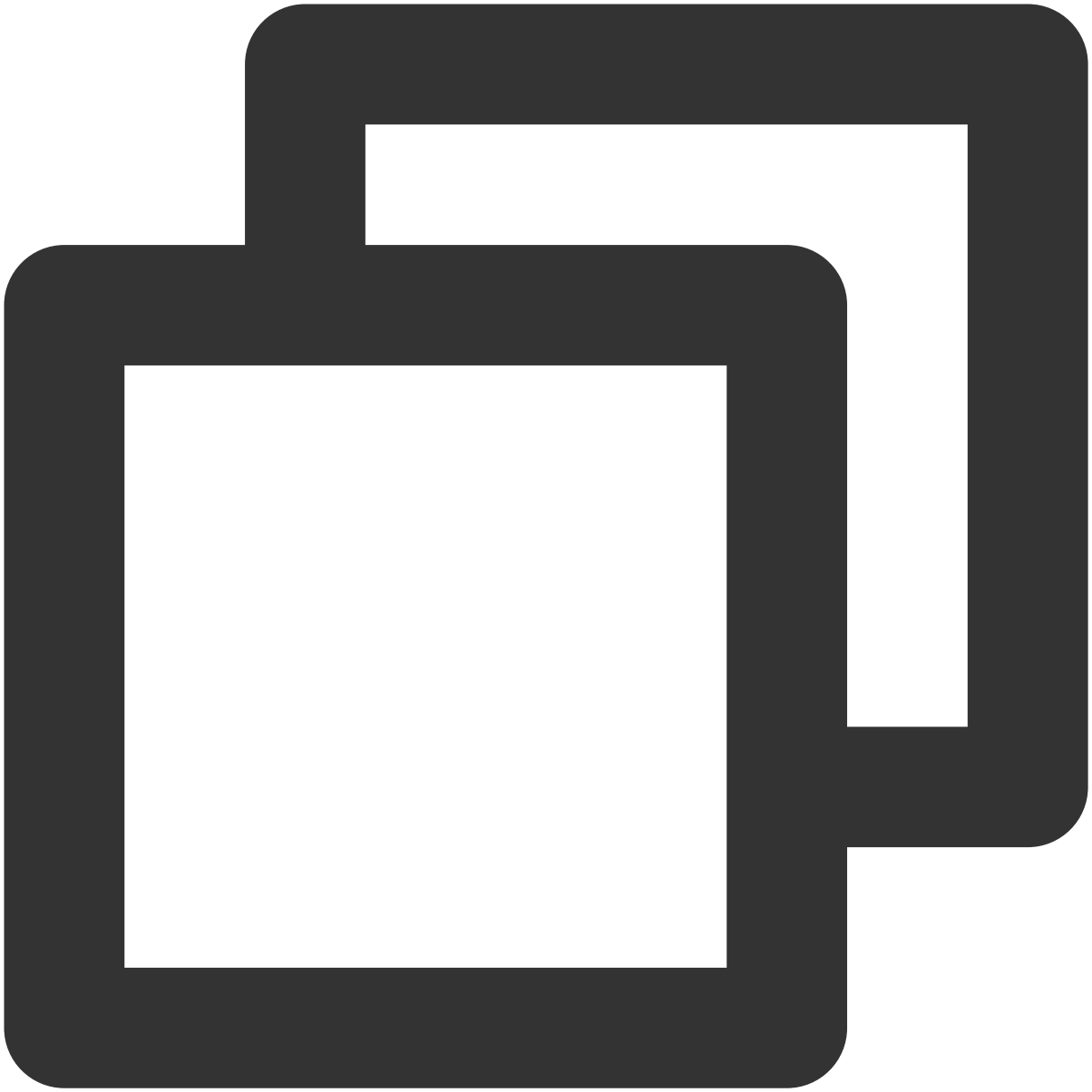
upported from v3.3.0+.

Vue

React



```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-vue";
TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```



```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-react";
TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```

Parameter list:

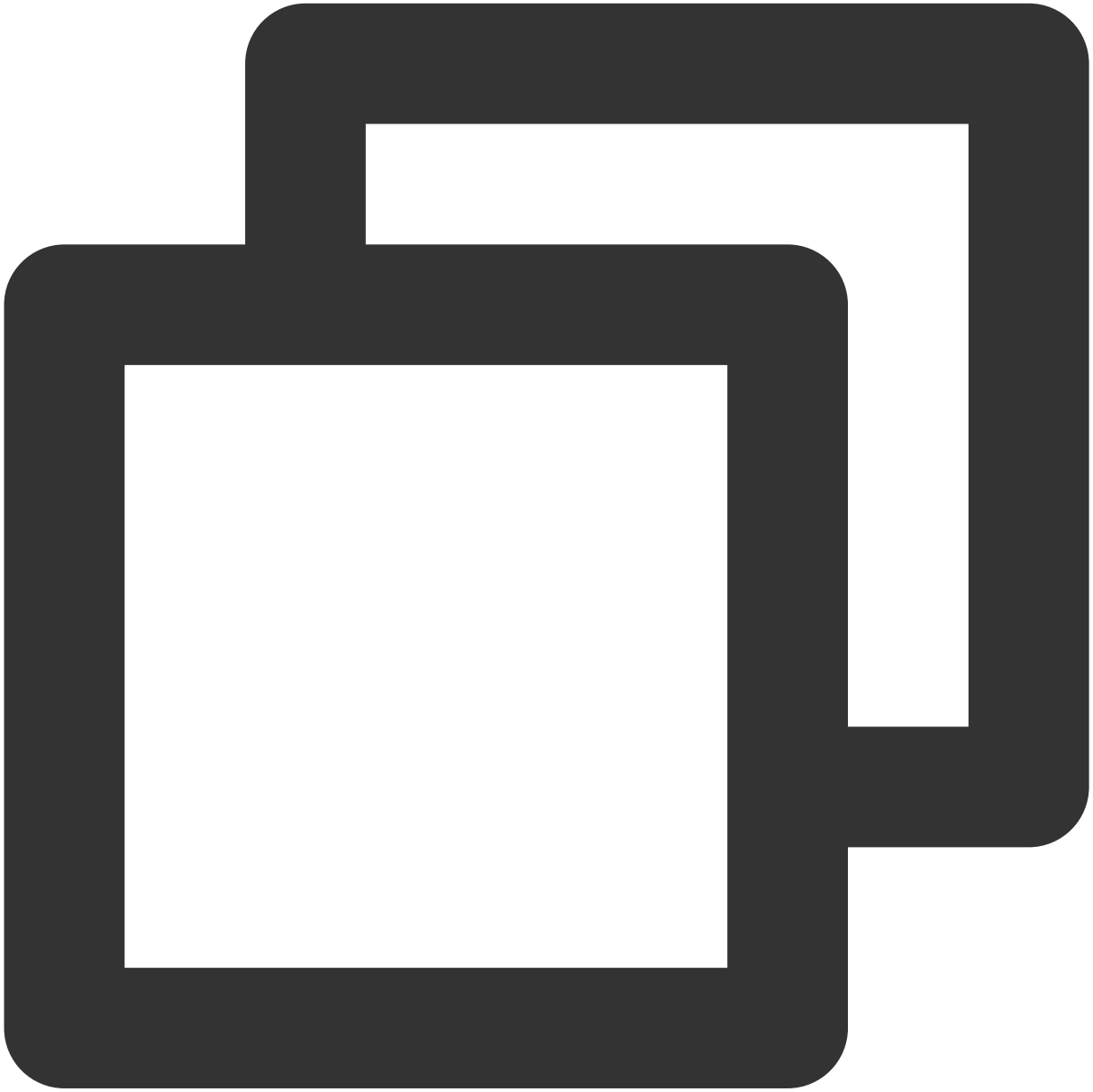
Parameter	Type	Required	Meaning
layoutMode	LayoutMode	Yes	User flow layout mode

setCameraDefaultState

Set whether the camera is on by default.

Note：

upported from v3.3.0+.



```
TUICallKitServer.setCameraDefaultState(true);
```

Parameter list:

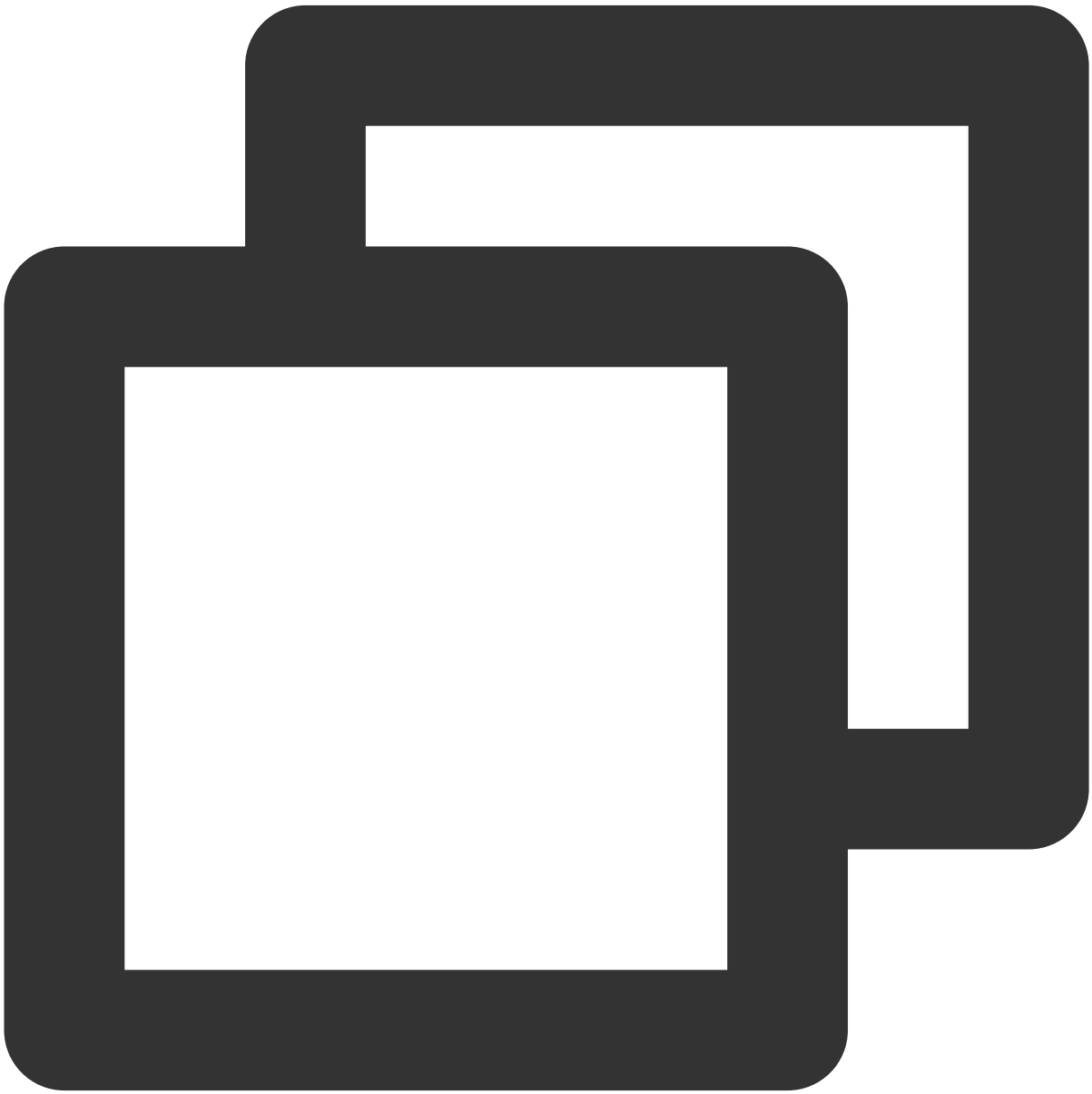
Parameter	Type	Required	Meaning
isOpen	boolean	Yes	Whether to enable the camera

getTUICallEngineInstance

Get TUICallEngine instance.

Note:

v2.4.3+ supported



```
TUICallKitServer.getTUICallEngineInstance();
```

TUICallKit Type Definition

videoDisplayMode

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

`VideoDisplayMode.CONTAIN`

`VideoDisplayMode.COVER`

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

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

videoResolution

The resolution `videoResolution` has three possible values:

`VideoResolution.RESOLUTION_480P`

`VideoResolution.RESOLUTION_720P`

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

Resolution Explanation:

Video Profile	Resolution (W x H)	Frame Rate (fps)	Bitrate (Kbps)
480p	640 × 480	15	900

720p	1280 × 720	15	1500
1080p	1920 × 1080	15	2000

Frequently Asked Questions:

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

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

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

STATUS

STATUS attribute value	Description
STATUS.IDLE	Idle status
STATUS.BE_INVITED	Received an Audio/Video Call Invite
STATUS.DIALING_C2C	Initiating a one-to-one call
STATUS.DIALING_GROUP	Initiating a group call
STATUS.CALLING_C2C_AUDIO	Engaged in a 1v1 Audio Call
STATUS.CALLING_C2C_VIDEO	In the midst of a one-to-one video call
STATUS.CALLING_GROUP_AUDIO	Engaged in Group Audio Communication
STATUS.CALLING_GROUP_VIDEO	Engaged in group video call

TUICallType

TUICallType Type	Description
TUICallType.AUDIO_CALL	Audio Call
TUICallType.VIDEO_CALL	Video Call

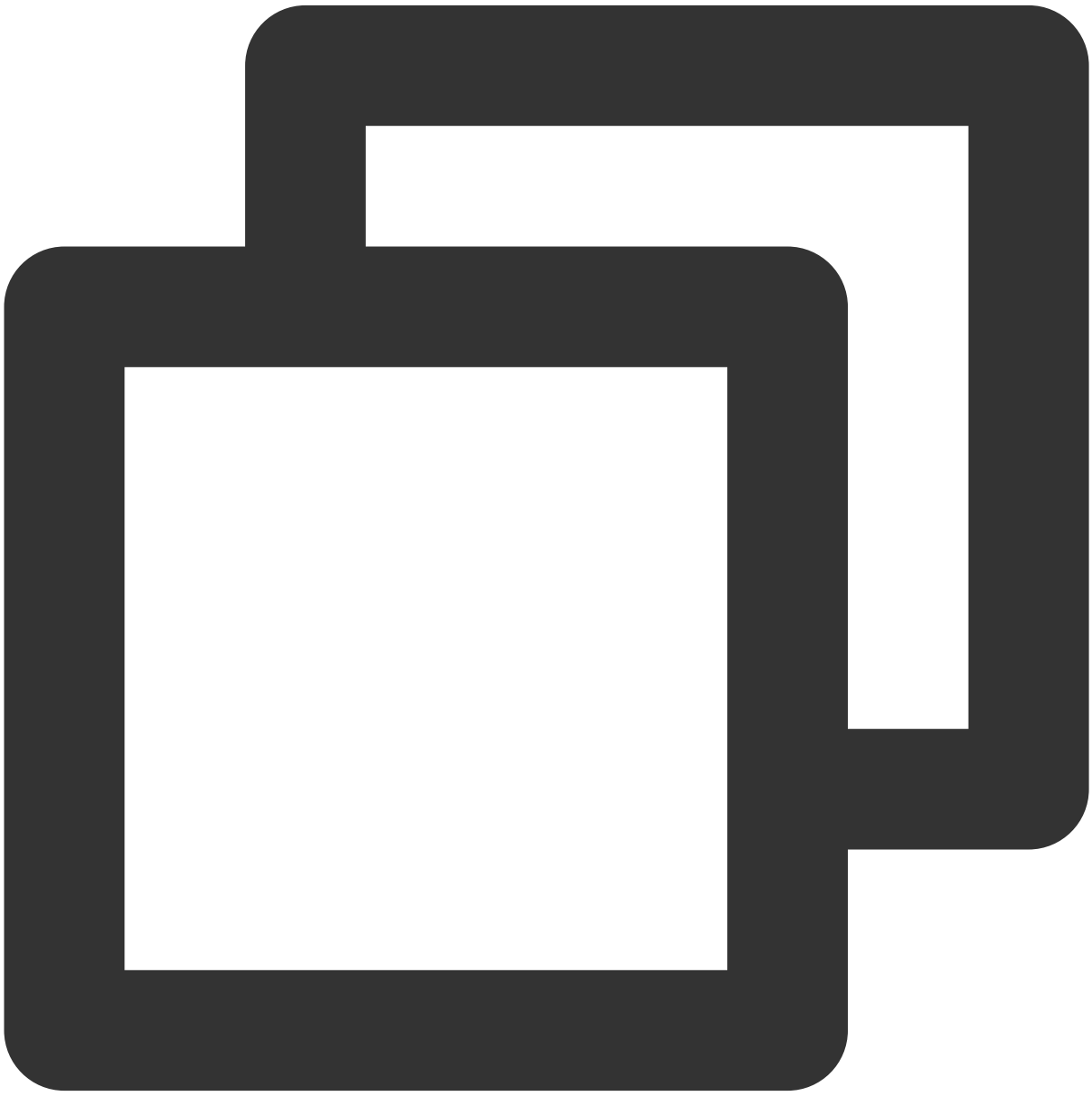
offlinePushInfo

Parameter	Type	Required	Meaning
offlinePushInfo.title	String	No	Offline Push Title (Optional)

offlinePushInfo.description	String	No	Offline Push Content (Optional)
offlinePushInfo.androidOPPOChannelID	String	No	Setting the channel ID for OPPO phones with 8.0 system and above for offline pushes (Optional)
offlinePushInfo.extension	String	No	Offline push through content. Can be used to set Android Notification mode and VoIP mode. Default: Notification mode, it will be a notification from the system; VoIP mode is required to pass the field.
offlinePushInfo.ignoreIOSBadge	Boolean	No	Ignore badge count for offline push (only for iOS), if set to true, the message will not increase the unread count of the app icon on the iOS receiver's side. **v3.3.0+ supported**
offlinePushInfo.iOSSound	String	No	Offline push sound setting (only for iOS). **v3.3.0+ supported**
offlinePushInfo.androidSound	String	No	Offline push sound setting. **v3.3.0+ supported**
offlinePushInfo.androidVIVOClassification	Number	No	Classification of VIVO push messages (deprecated interface, VIVO push service will optimize message classification rules on April 3, 2023. It is recommended to use setAndroidVIVOCategory to set the message category). 0: Operational messages, 1: System messages. The default value is 1. **v3.3.0+ supported**
offlinePushInfo.androidXiaoMiChannelID	String	No	Set the channel ID for Xiaomi phones with Android 8.0 and above systems. **v3.3.0+ supported**
offlinePushInfo.androidFCMChannelID	String	No	Set the channel ID for google phones with Android 8.0 and above systems. **v3.3.0+ supported**
offlinePushInfo.androidHuaWeiCategory	String	No	Classification of Huawei push messages. **v3.3.0+ supported**

offlinePushInfo.isDisablePush	Boolean	No	Whether to turn off push notifications (default is on). **v3.3.0+ supported**
offlinePushInfo.iOSPushType	Number	No	iOS offline push type，default is 0。0-APNs；1-VoIP. **v3.3.0+ supported**

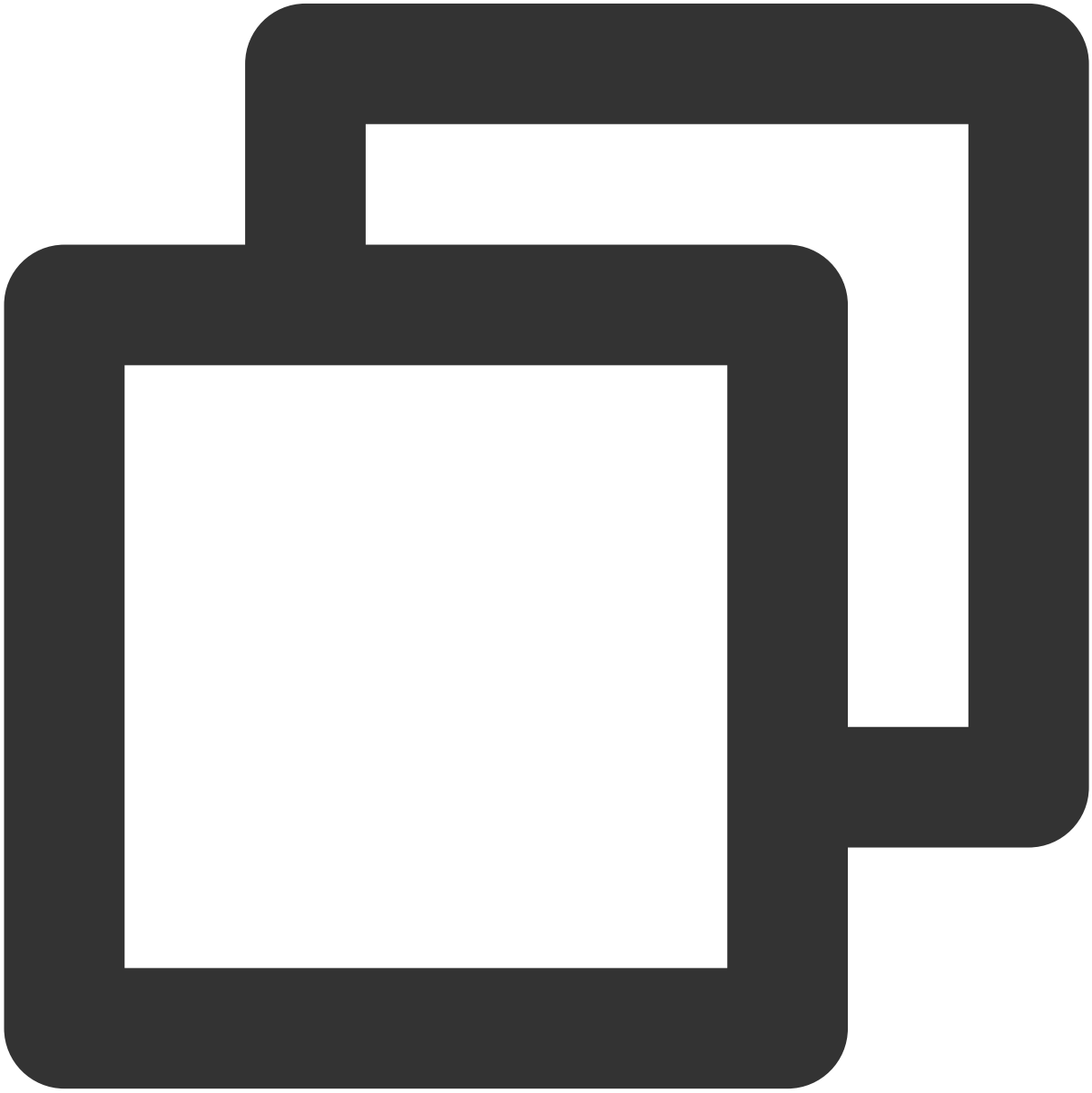
Android Notification Mode



```
const extension = {
  timPushFeatures: {
    fcmPushType: 0, // 0, VoIP; 1, notification
```

```
  }
};
offlinePushInfo.extension = JSON.stringify(extension);
```

Android VoIP Mode



```
const extension = {
  timPushFeatures: {
    fcmPushType: 0, // 0, data; 1, notification
    fcmNotificationType: 1, // 0, TIMPush implementation; 1, business implementati
  }
```

```
};
offlinePushInfo.extension = JSON.stringify(extension);
```

FeatureButton

FeatureButton Type	Description
FeatureButton.Camera	Camera Button
FeatureButton.Microphone	Microphone Button
FeatureButton.SwitchCamera	Switches between the front and rear cameras.
FeatureButton.InviteUser	Invite users button

LayoutMode

LayoutMode type	Description
LayoutMode.LocalInLargeView	Local user in large window display
LayoutMode.RemoteInLargeView	Remote user in large window display

TUICallEngine

Last updated：2024-05-24 17:39:32

TUICallEngine APIs

TUICallEngine API is the **No UI Interface** of the Audio and Video Call Components.

API Overview

API	Description
createInstance	Creating a TUICallEngine Instance (Singleton Pattern)
destroyInstance	Terminating a TUICallEngine Instance (Singleton Pattern)
on	Listening on events
off	Canceling Event Listening
login	Sign in Interface
logout	Logout Interface
setSelfInfo	Configure the user's nickname and profile photo
call	Initiate a one-on-one call
groupCall	Group Chat Invitation Call
accept	Answer Calls
reject	Decline Call
hangup	End Calls
switchCallMediaType	Switch Audio and Video Calls
startRemoteView	Initiate Remote Screen Rendering
stopRemoteView	Stop Remote Screen Rendering
startLocalView	Start Local Screen Rendering, **Note: This will be deprecated; use openCamera instead**

stopLocalView	Stop Local Screen Rendering, **Note: This will be deprecated; use closeCamera instead**
openCamera	Enable the camera
closeCamara	Turn Off Camera
switchCamera	Switch between front and rear cameras, note: only supported on mobile devices. **v3.0.0+ supported**
openMicrophone	Enable Microphone
closeMicrophone	Turn off the microphone
setVideoQuality	Set video quality
getDeviceList	Access device list
switchDevice	Switch camera or microphone devices
enableAIVoice	Enable/disable AI noise reduction
enableMultiDeviceAbility	Turn on/off the multi-device login mode of TUICallEngine. **v2.1.1+ supported**
setBlurBackground	Switch/set background blur, **v3.0.6+ supported**
setVirtualBackground	Switch/set image background blur, **v3.0.6+ supported**

TUICallEvent

Last updated：2024-06-28 14:38:07

TUICallEvent API Introduction

TUICallEvent API is the **Event Interface** of the Audio and Video Call Components.

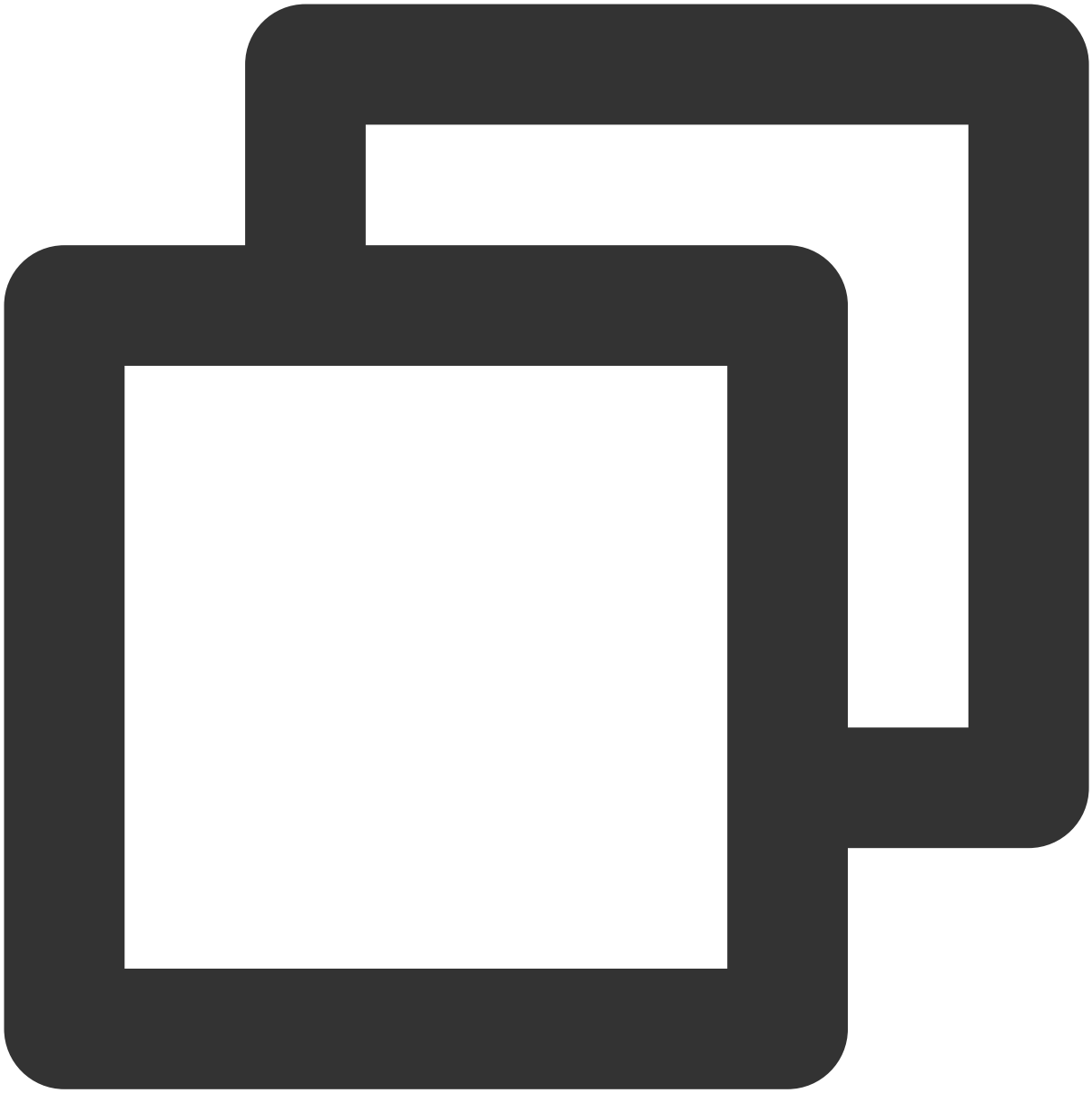
Event List

EVENT	Description
TUICallEvent.ERROR	An error occurred during the call.
TUICallEvent.SDK_READY	This event is received when the SDK enters the ready state
TUICallEvent.KICKED_OUT	Receiving this event after a duplicate sign-in indicates that the user has been removed from the room
TUICallEvent.USER_ACCEPT	If a user answers, this event will be received
TUICallEvent.USER_ENTER	A user joined the call.
TUICallEvent.USER_LEAVE	A user left the call.
TUICallEvent.REJECT	A user declined the call.
TUICallEvent.NO_RESP	A user didn't respond.
TUICallEvent.LINE_BUSY	A user was busy.
TUICallEvent.USER_VIDEO_AVAILABLE	Whether a user has a video stream.
TUICallEvent.USER_AUDIO_AVAILABLE	Whether a user has an audio stream.
TUICallEvent.USER_VOICE_VOLUME	The volume levels of all users.
TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE	Group Chat Update, Invitation List this callback will be received
TUICallEvent.ON_CALL_BEGIN	Call connected event, **v1.4.6+ supported**

TUICallEvent.INVITED	A call was received. **It will be discarded later and it is recommended to use TUICallEvent.ON_CALL_RECEIVED**
TUICallEvent.ON_CALL_RECEIVED	Call request event, **v1.4.6+ supported**
TUICallEvent.CALLING_CANCEL	Call cancellation event, **It will be abandoned later and it is recommended to use TUICallEvent.ON_CALL_CANCELED**
TUICallEvent.ON_CALL_BEGIN	Call connected event, **v1.4.6+ supported**
TUICallEvent.CALLING_END	The call ended.
TUICallEvent.DEVICED_UPDATED	Device list update, this event will be received
TUICallEvent.CALL_TYPE_CHANGED	Call type switching, this event will be received
TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED	All user network quality events, **v3.0.7+ supported**

ERROR

Error event during the call. You can capture internal errors during the call by monitoring this event.



```
let onError = function(error) {
  console.log(error.code, error.msg);
};
tuiCallEngine.on(TUICallEvent.ERROR, onError);
```

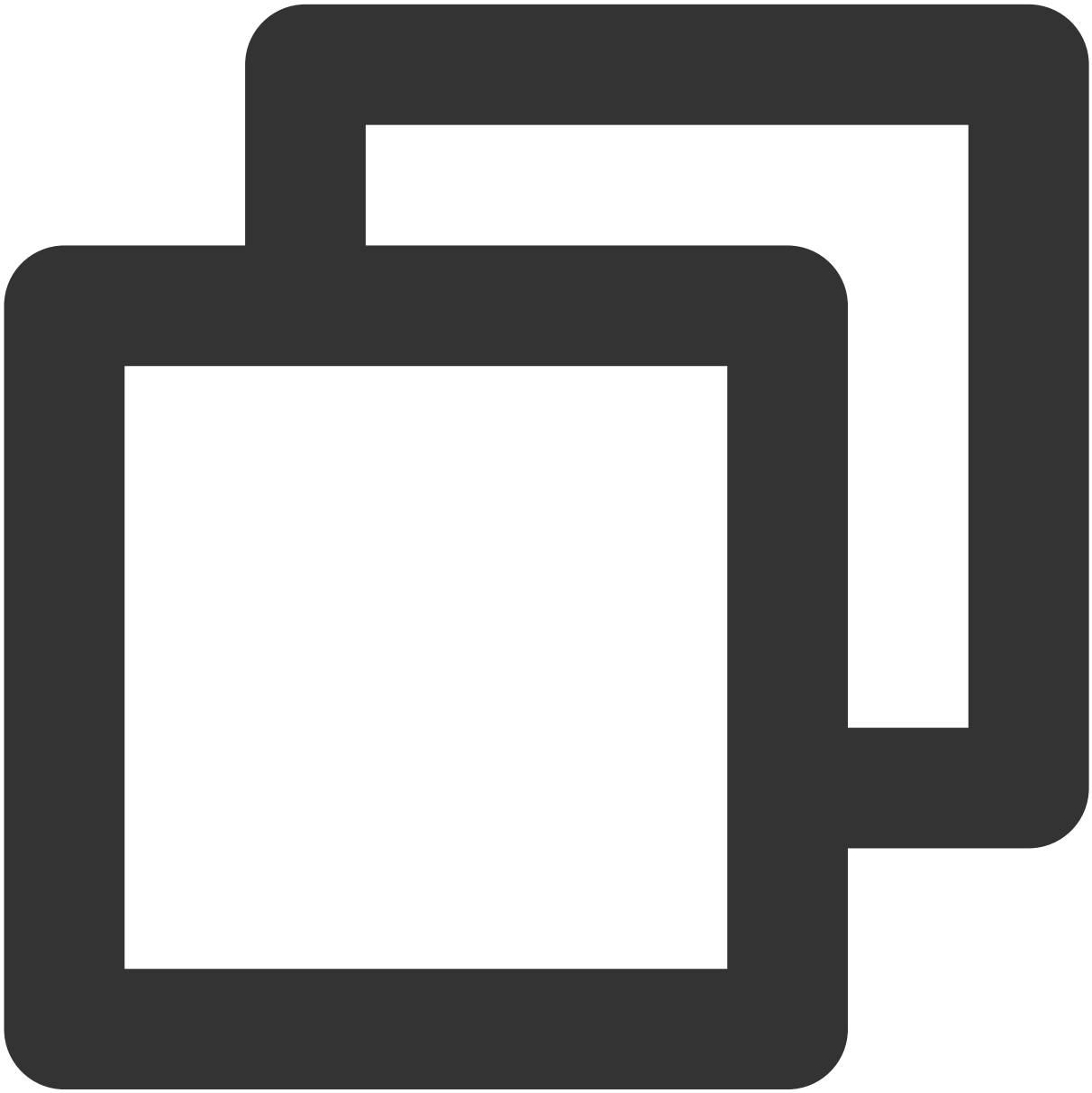
The parameters are described below:

Parameter	Type	Meaning
code	Number	Error Code

msg	String	Error message

SDK_READY

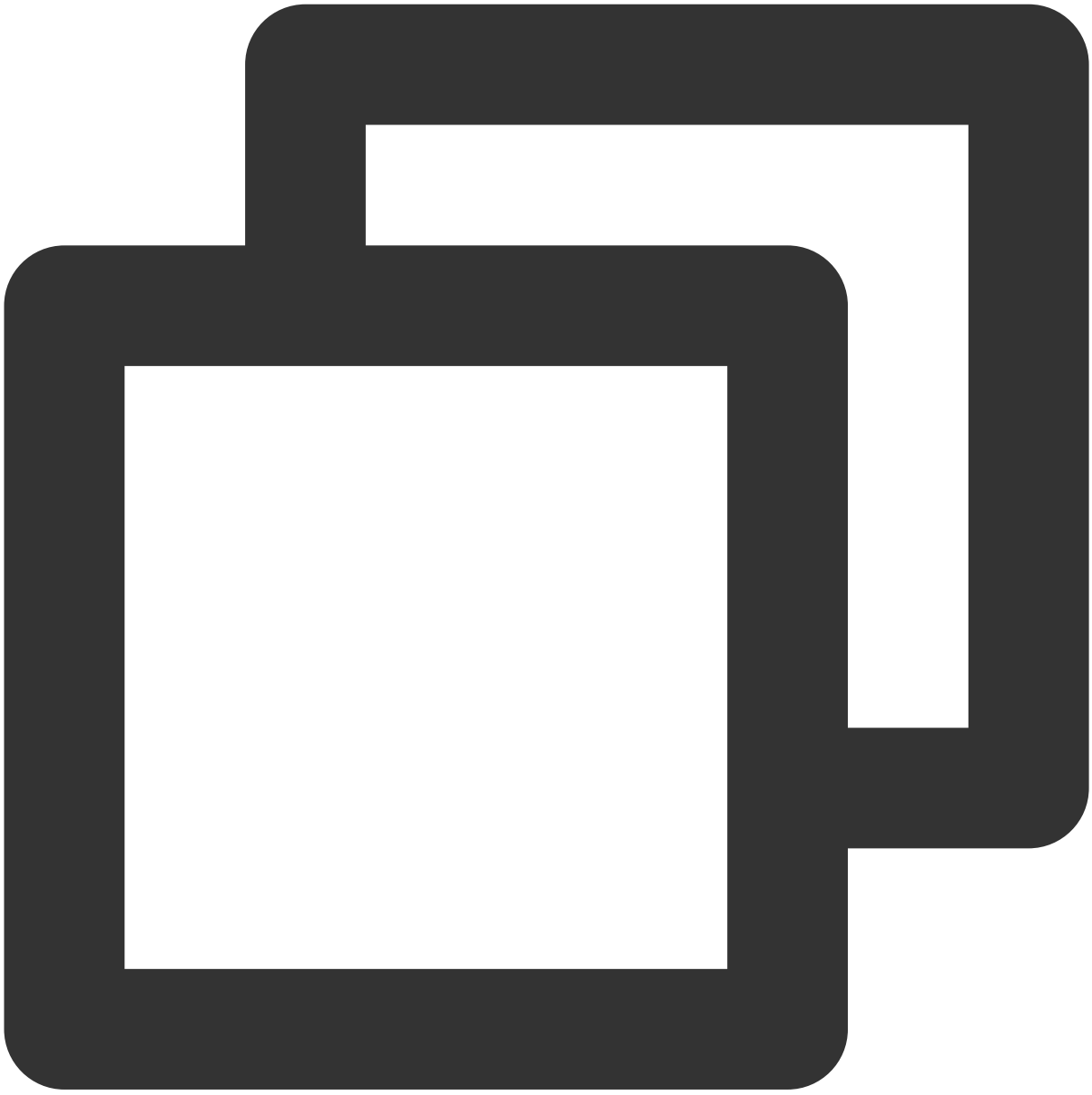
TUICallEngine relies on @tencentcloud/chat SDK. The SDK_READY event will be triggered only after successful login, and then you can use various functions of the SDK.



```
let onSDKReady = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.SDK_READY, onSDKReady);
```

KICKED_OUT

The current user was kicked offline：At this time, you can prompt the user with a UI message and then invoke `login` again.



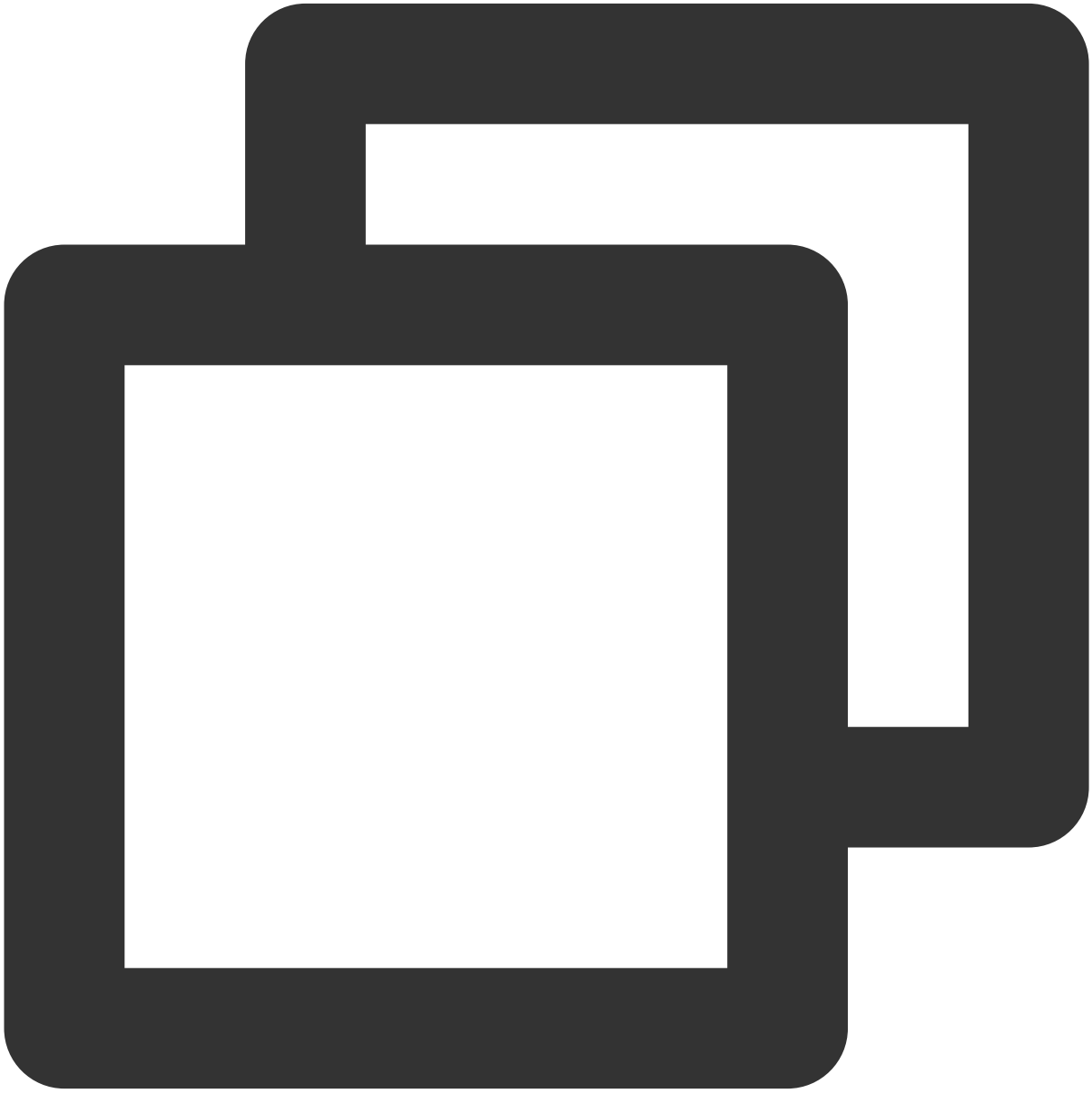
```
let handleOnKickedOut = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.KICKED_OUT, handleOnKickedOut);
```

USER_ACCEPT

If a user answers, all other users will receive this event, where `userID` is the user who answered.

1. In a 1v1 call: when the callee answers, the caller will throw this event.

2. In group calls: if A calls B and C, and B answers, both A and C will throw this event, with the event's `userID` being B. Similarly, if C answers, both A and B will throw this event, with the event's `userID` being C.



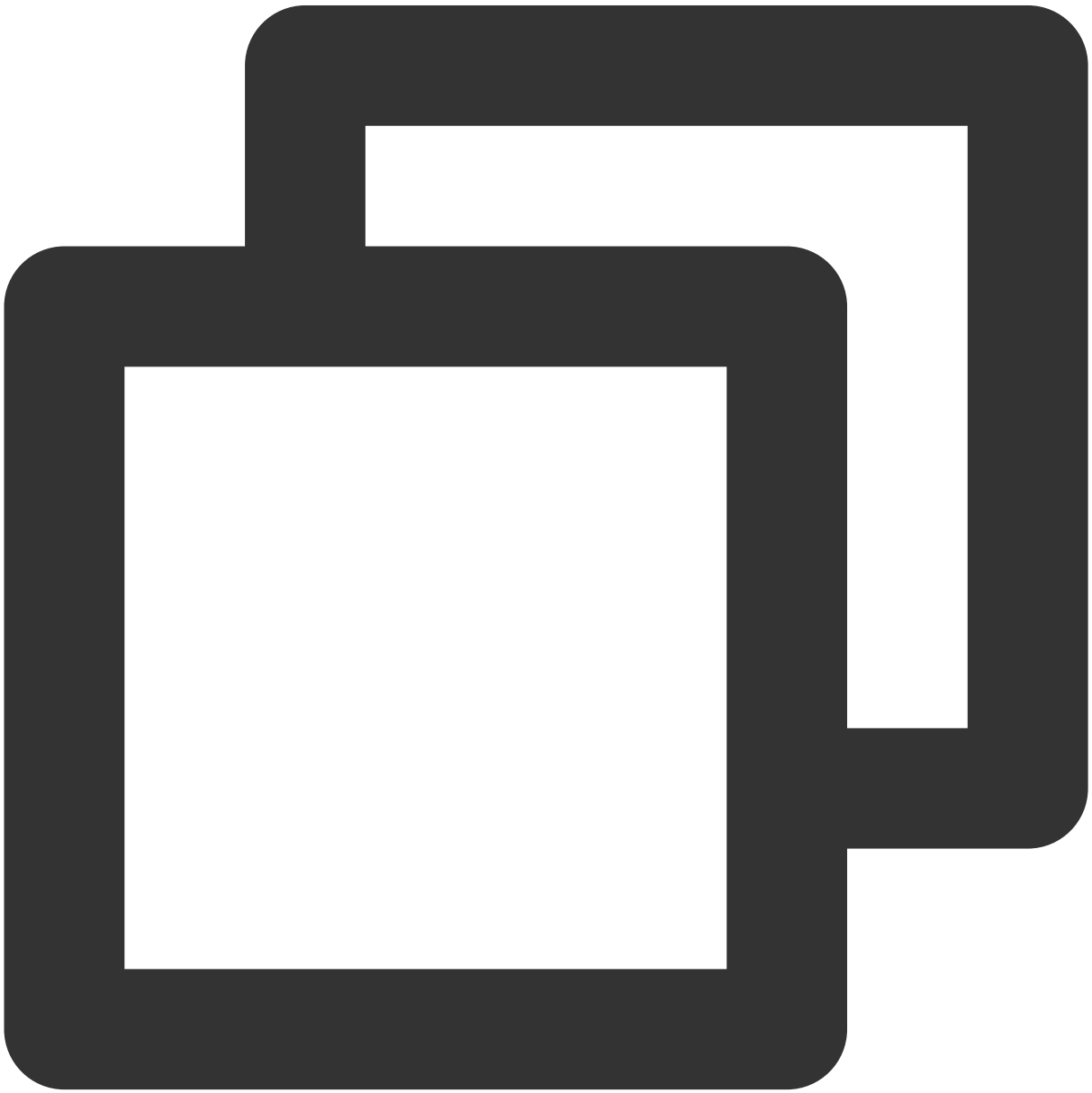
```
let handleUserAccept = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_ACCEPT, handleUserAccept);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Answering User ID

USER_ENTER

If a user enters the call, other users will throw this event, and userID is the user name who entered the call.



```
let handleUserEnter = function(event) {
```

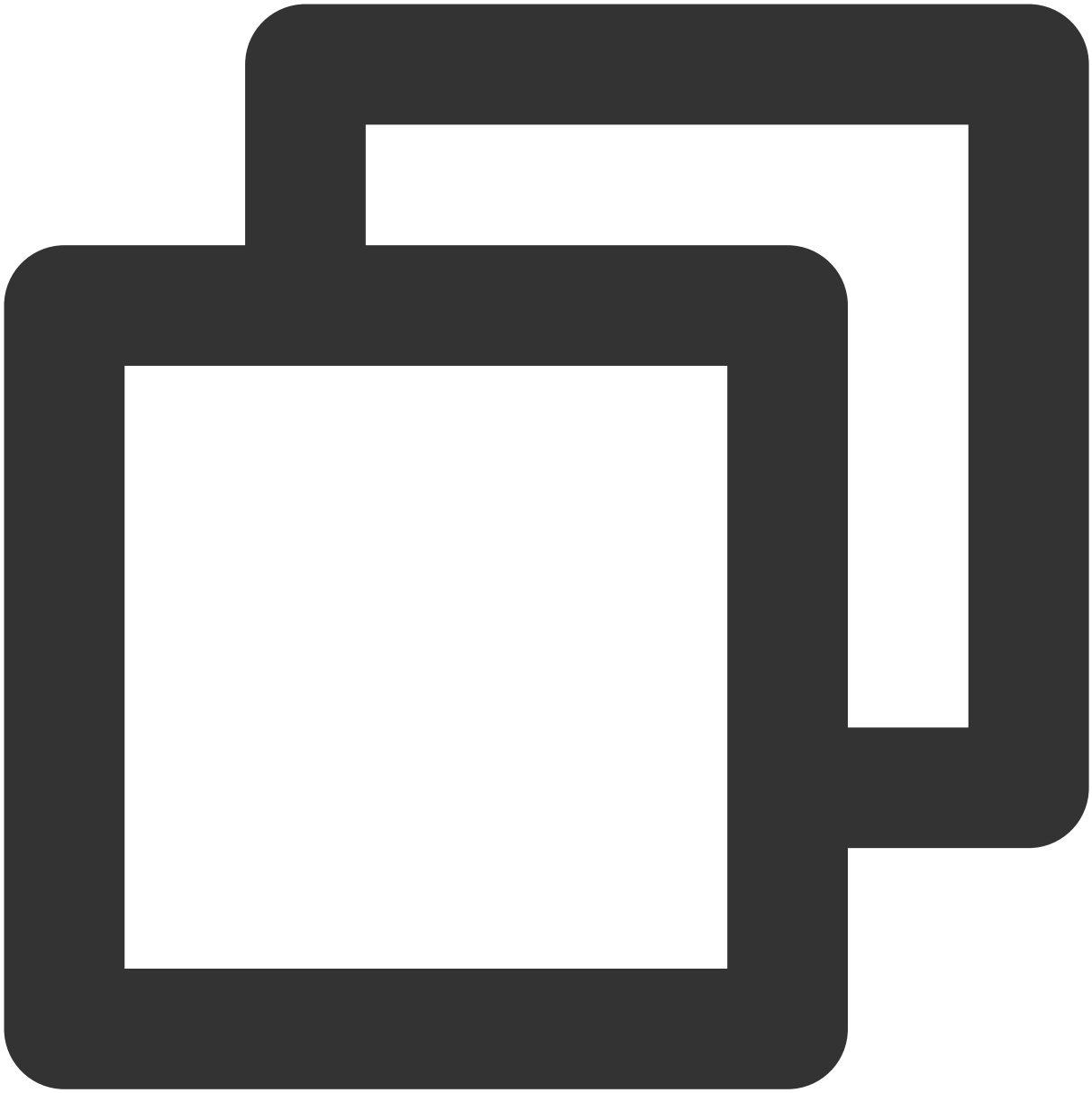
```
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_ENTER, handleUserEnter);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Entering User ID

USER_LEAVE

When a user leaves the call, this event will be thrown by other users in the call. The userID is the name of the user who left the call.



```
let handleUserLeave = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_LEAVE, handleUserLeave);
```

The parameters are described below:

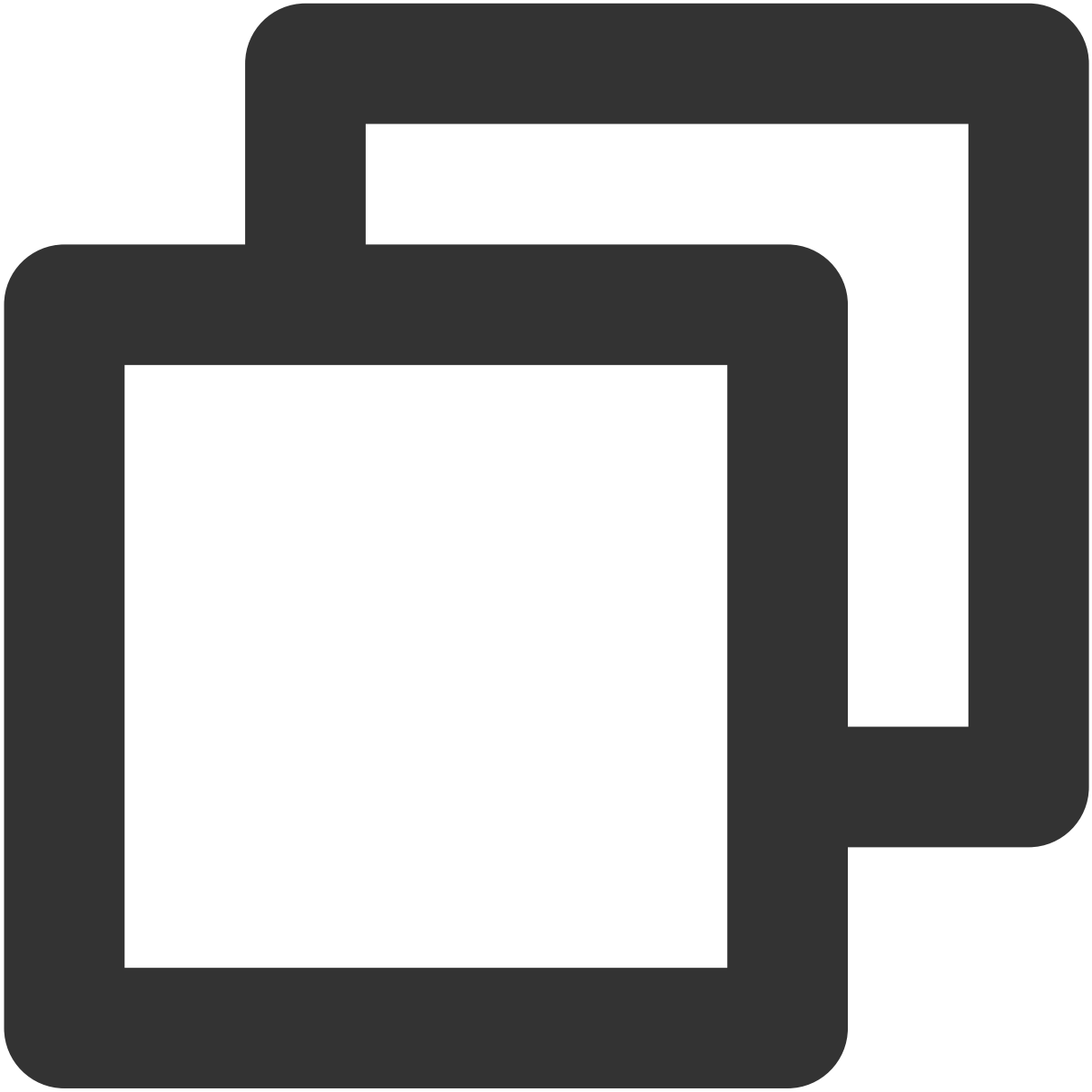
Parameter	Type	Meaning
userID	String	Exiting User ID

REJECT

This event is thrown when the call is rejected

1. In a 1v1 call, only the calling party will receive the rejection event, and userID is the called username.

2. In a group call, when an invitee refuses the call, this event will be thrown by other people in the group call. The userID is the name of the user who refused the call.



```
let handleInviteeReject = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.REJECT, handleInviteeReject);
```

The parameters are described below:

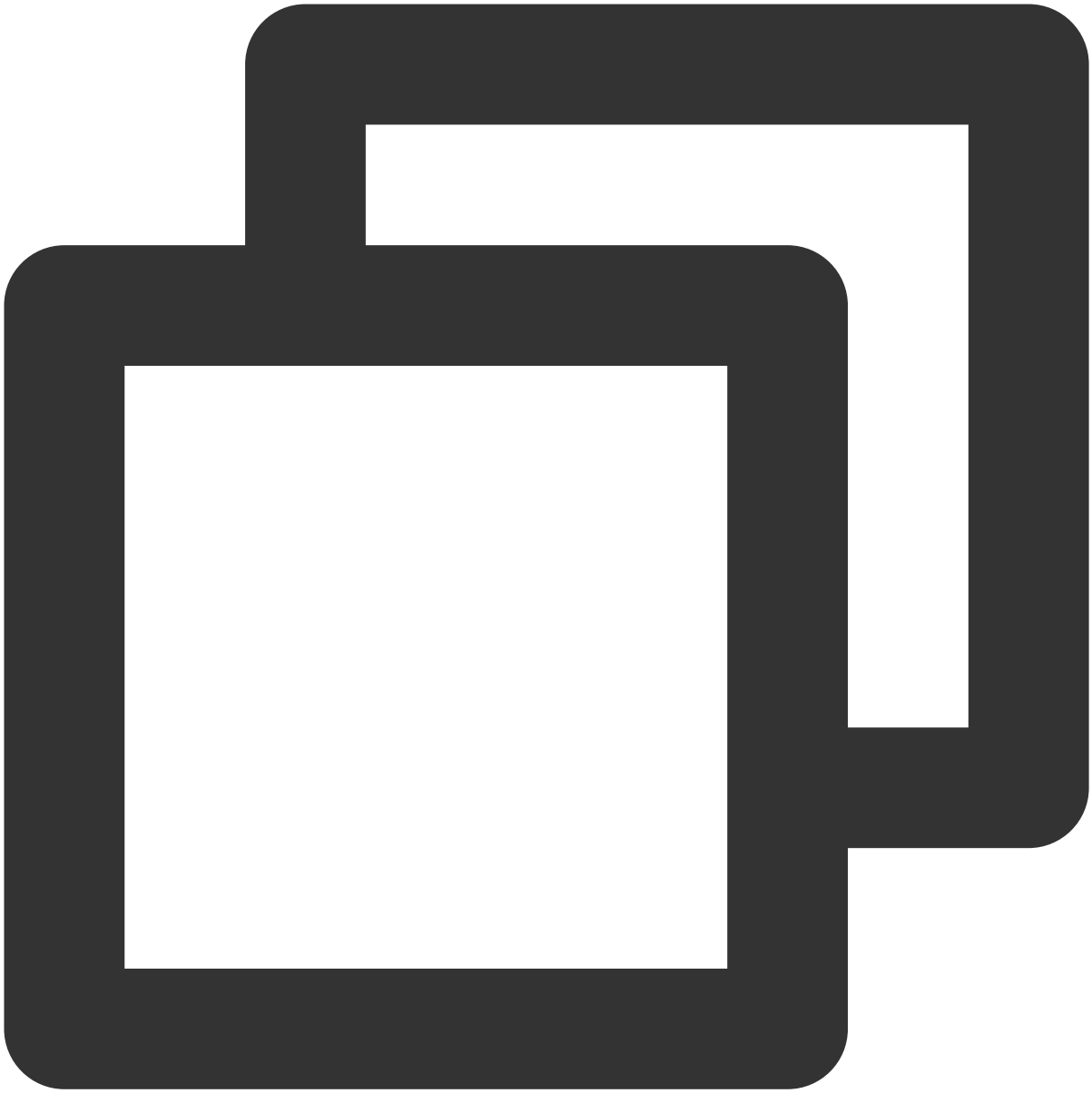
Parameter	Type	Meaning
userID	String	Rejecting User ID

NO_RESP

This event will be thrown by other calling users when the callee does not respond.

In a 1v1 call, only the initiator will receive the event of no answer. For example, A invites B, B does not answer, A can receive this event.

In a group call, when an invitee does not respond, this event will be thrown by everyone else in the group call. For example, if A invites B and C to join the call, but B does not respond, both A and C will throw this event.



```
let handleNoResponse = function(event) {
console.log(event.sponsor, event.userIDList);
};
tuiCallEngine.on(TUICallEvent.NO_RESP, handleNoResponse);
```

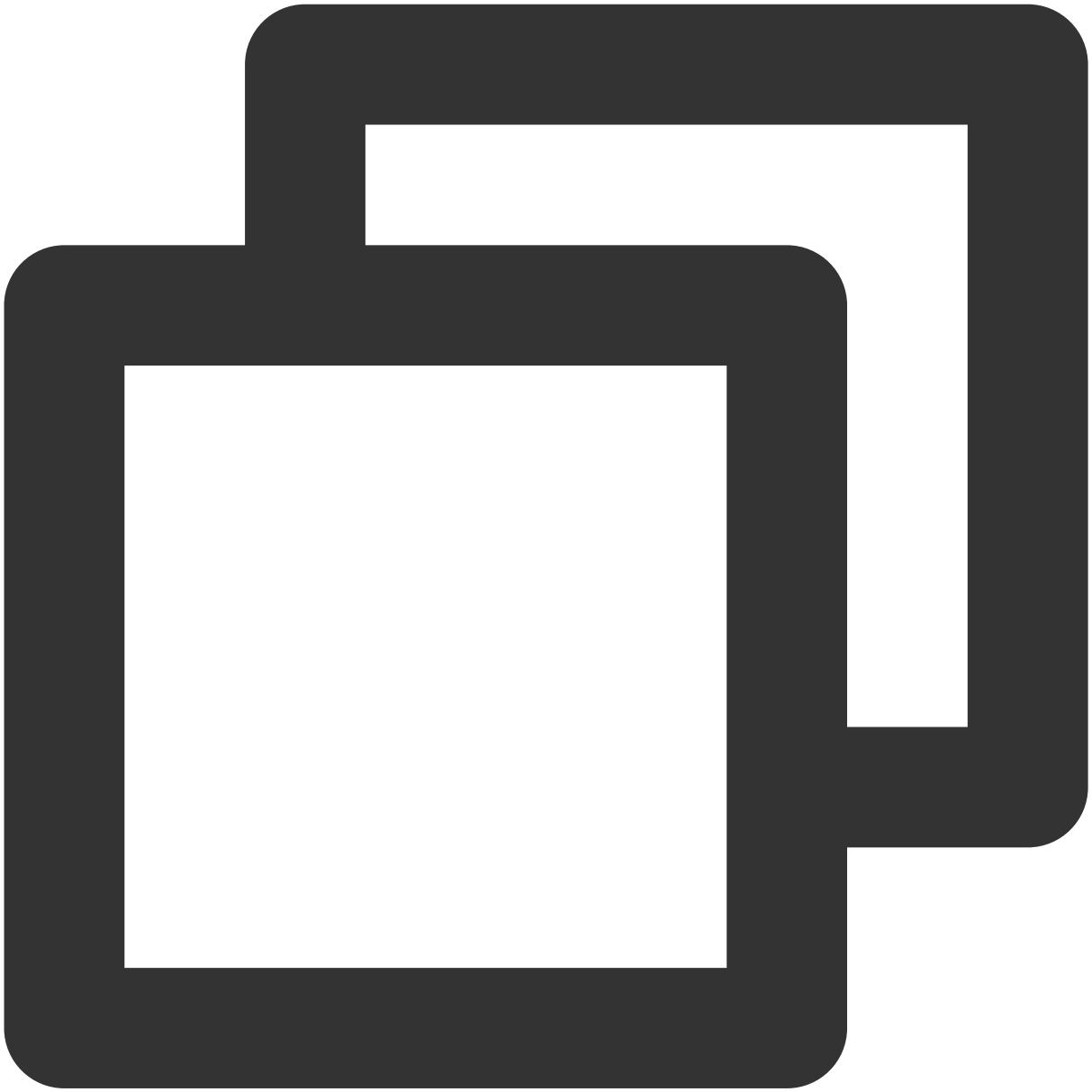
The parameters are described below:

Parameter	Type	Meaning
sponsor	String	Caller's User ID

userIDList	Array<String>	List of Users Who Triggered Timeout Due to No Response

LINE_BUSY

Call busy event. For example: when B is on a call, and A calls B, A will throw an event.



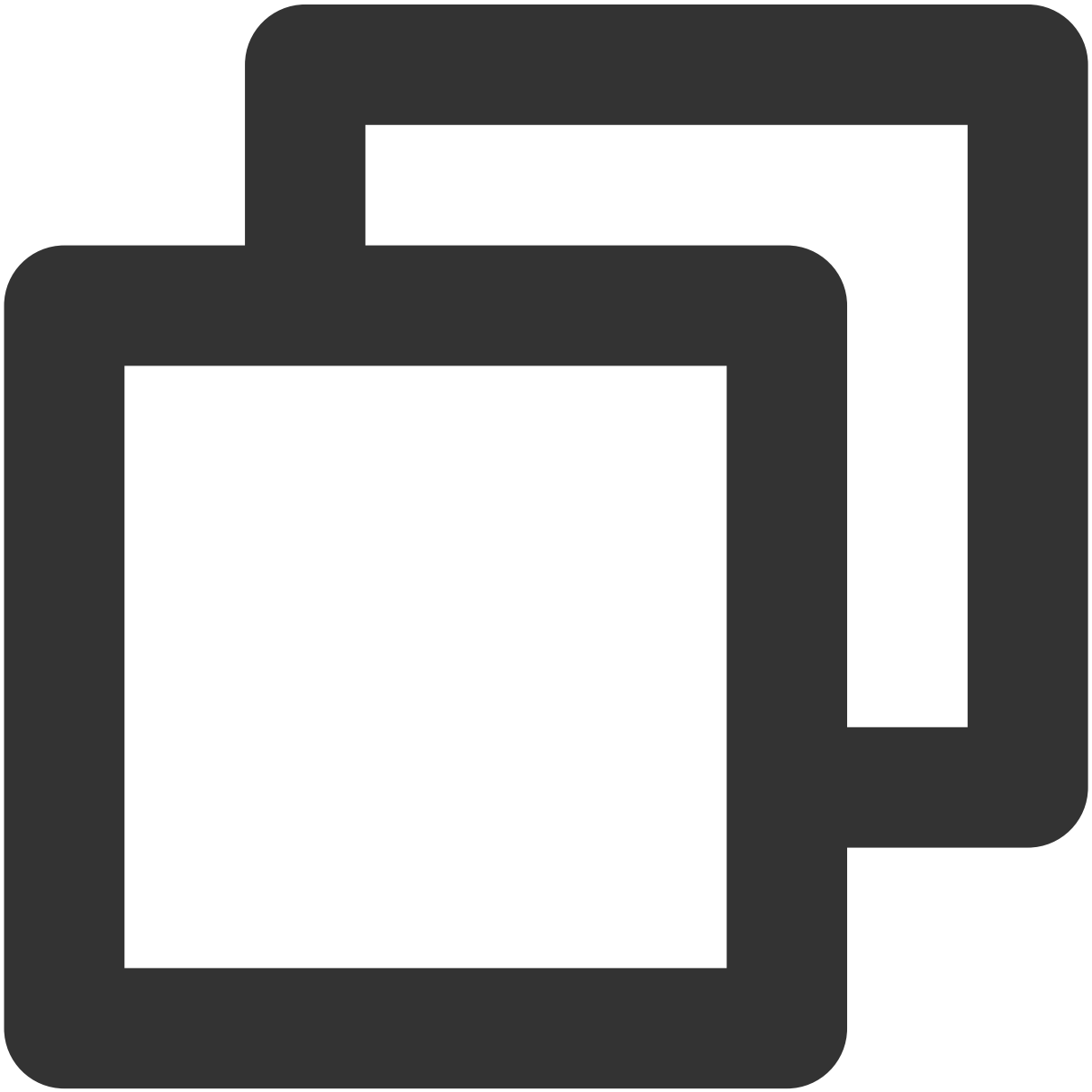
```
let handleLineBusy = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.LINE_BUSY, handleLineBusy);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Busy User ID

USER_VIDEO_AVAILABLE

If a user turns on/off the camera during a video call, this event will be thrown by other users in the call. For example: A and B are on a video call, A turns on/off the camera, and B will throw this event.



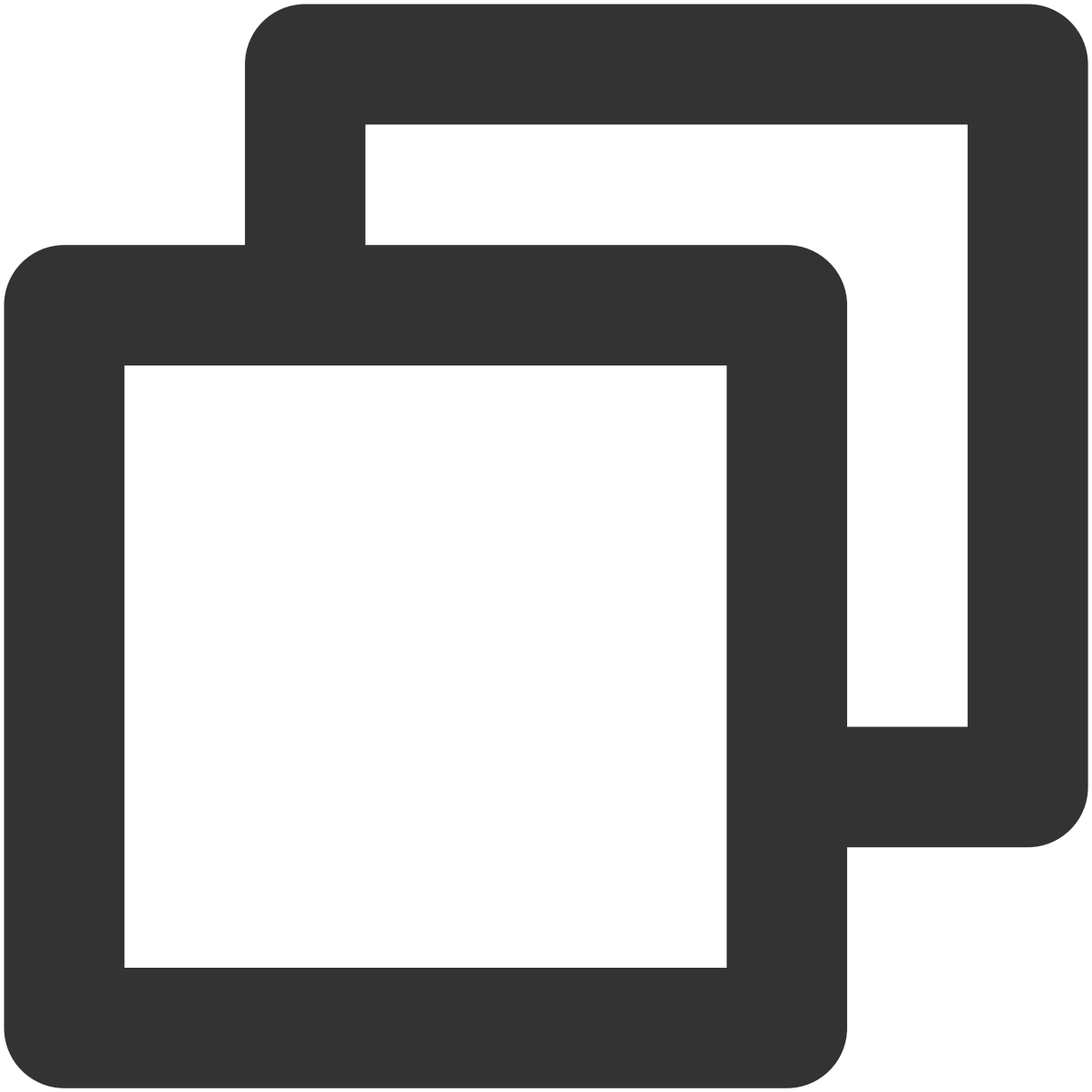
```
let handleUserVideoChange = function(event) {
  console.log(event.userID, event.isVideoAvailable);
};
tuiCallEngine.on(TUICallEvent.USER_VIDEO_AVAILABLE, handleUserVideoChange);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Remote User ID
isVideoAvailable	Boolean	true: Remote User turns Camera On; false: Remote User turns Camera Off

USER_AUDIO_AVAILABLE

If a user turns on/off the microphone during an audio or video call, this event will be thrown by other users on the call. For example: A and B are having an audio and video call, and A turns on/off the microphone, and B will throw this event.



```
let handleUserAudioChange = function(event) {
  console.log(event.userID, event.isAudioAvailable);
};
tuiCallEngine.on(TUICallEvent.USER_AUDIO_AVAILABLE, handleUserAudioChange);
```

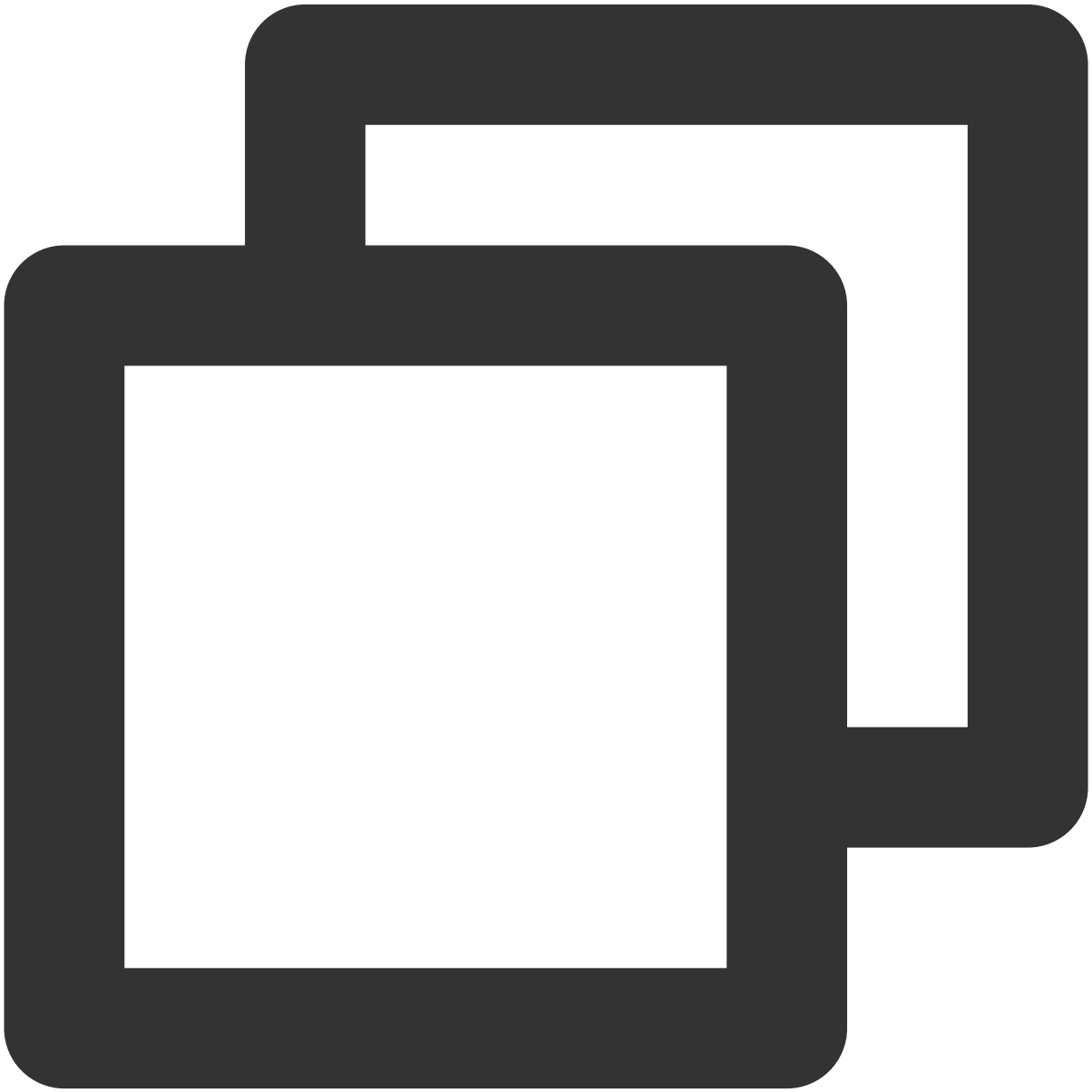
The parameters are described below:

Parameter	Type	Meaning
userID	String	User ID to turn microphone on/off

isAudioAvailable	Boolean	true the user turns on the microphone; false the user turns off the microphone

USER_VOICE_VOLUME

When the user's volume changes during an audio or video call, this event will be thrown by other users on the call. For example: A and B are having an audio and video call, and if A's volume changes, B will throw this event.



```
let handleUserVoiceVolumeChange = function(event) {
  console.log(event.volumeMap);
};
```

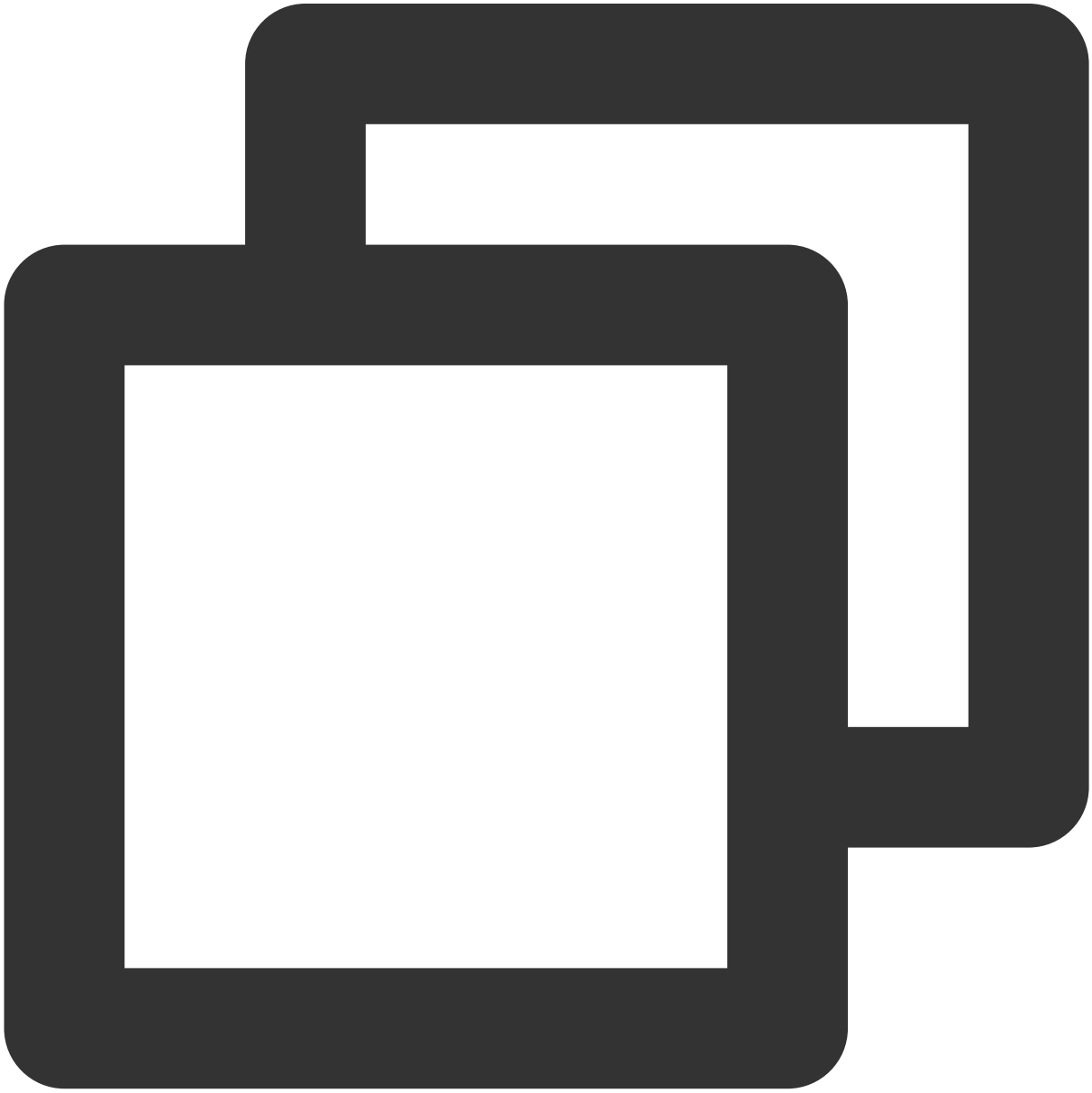
```
tuiCallEngine.on(TUICallEvent.USER_VOICE_VOLUME, handleUserVoiceVolumeChange);
```

The parameters are described below:

Parameter	Type	Meaning
volumeMap	Array<Object>	Volume meter, the corresponding volume can be obtained according to each userid, volume range: [0, 100]

GROUP_CALL_INVITEE_LIST_UPDATE

Group chat update invitation list, this event will be received.



```
let handleGroupInviteeListUpdate = function(event) {
  console.log(event.userIDList);
};
tuiCallEngine.on(TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE, handleGroupInviteeLis
```

The parameters are described below:

Parameter	Type	Meaning
userIDList	Array<String>	Group update invitation list

INVITED

Receiving a new incoming call event, the called party will be notified. By listening to this event, you can decide whether to display the call answering interface.

Note:

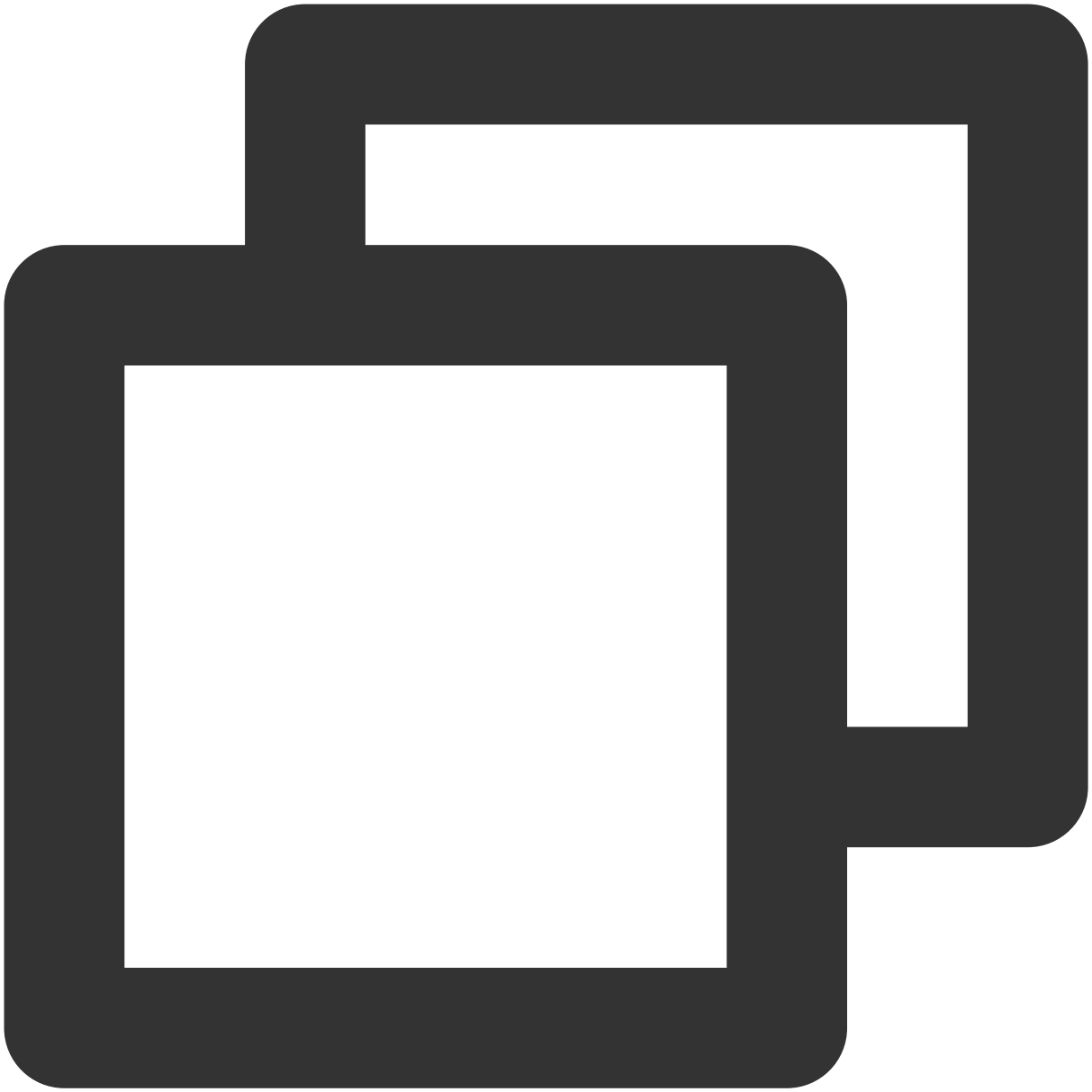
Plan to deprecate in subsequent versions. Recommended: ON_CALL_RECEIVED.

ON_CALL_RECEIVED

Receiving a new incoming call event, the called party will be notified. By listening to this event, you can decide whether to display the call answering interface.

Note:

v1.4.6+ supported.



```
let handleOnCallReceived = function(event) {
    console.log(event);
};
tuiCallEngine.on(TUICallEvent.ON_CALL_RECEIVED, handleOnCallReceived);
```

The parameters are described below:

Parameter	Type	Meaning
sponsor	String	Inviter

userIDList	Array<String>	Also Invited Persons
isFromGroup	Boolean	Is it a Group Call
inviteData	Object	Call Data
inviteID	String	Invitation ID, identifying one invitation
userData	String	Extended field: Utilized for amplifying details in the invitation signaling
callId	String	Unique ID for this call
roomID	Number	Audio-Video Room ID for this call
callMediaType	Number	Media Type of the call, Video Call, Voice Call
callRole	String	role, Enumeration Type: Caller, Called

CALLING_CANCEL

If the call is not established, this event will be thrown. By listening to this event, you can implement display logic similar to missed calls, reset UI state, etc. Scenarios where the call is not established include:

Note:

Plan to deprecate in subsequent versions. Recommended: ON_CALL_CANCELED.

ON_CALL_CANCELED

If the call is not established, this event will be thrown. By listening to this event, you can implement display logic similar to missed calls, reset UI state, etc. Scenarios where the call is not established include:

Caller Cancelled: The caller throws this event, userID is the caller; the called also throws this event, userID is the called;

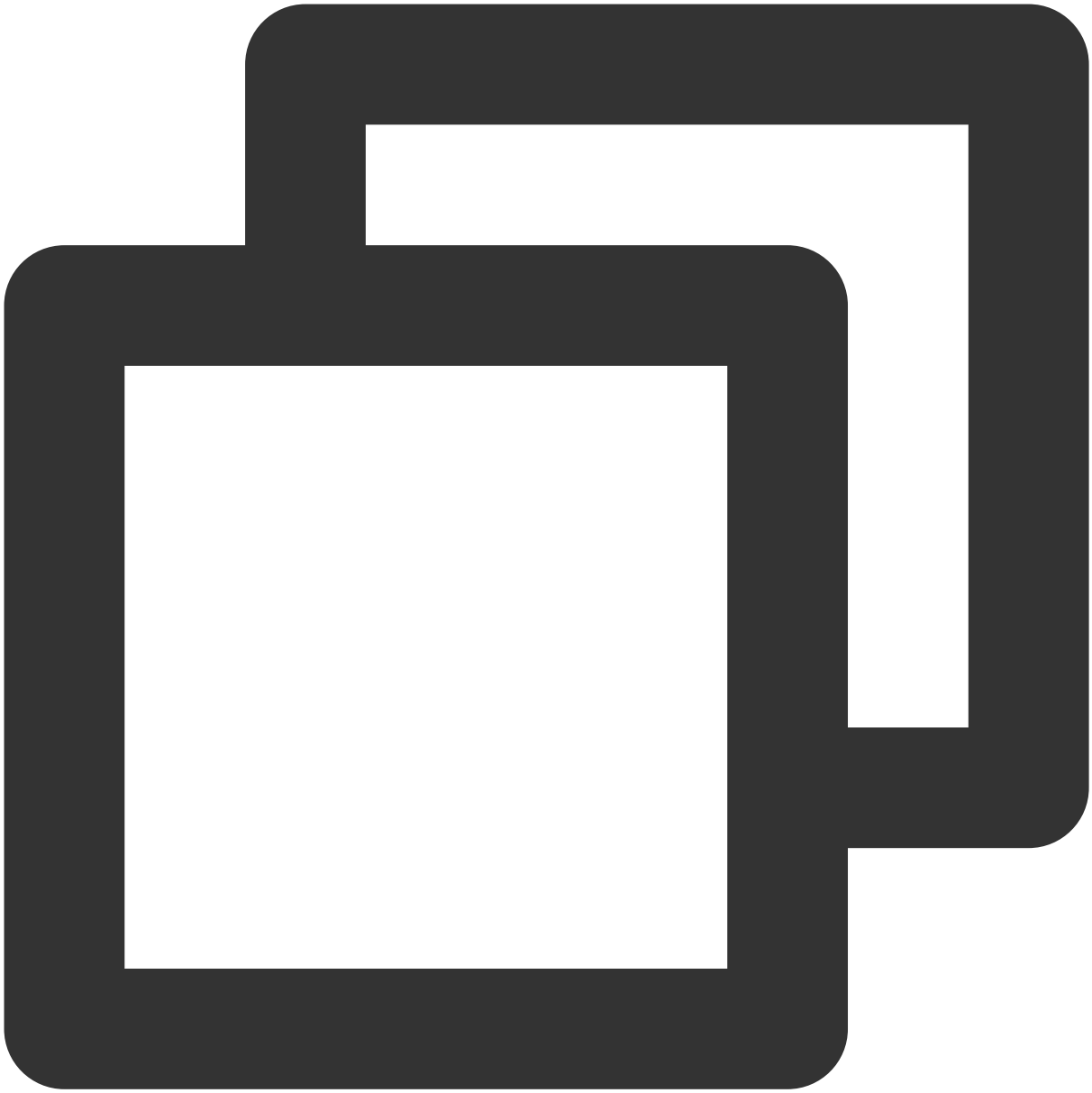
Callee Timeout: The caller will throw both NO_RESP and CALLING_CANCEL events, userID is the caller; the called throws the CALLING_CANCEL event, userID is the called;

Callee Rejected: The caller will throw both REJECT and CALLING_CANCEL events, userID is the caller; the called throws the CALLING_CANCEL event, userID is the called;

Callee Busy: The caller will throw both LINE_BUSY and CALLING_CANCEL events, userID is the caller; the callee throws the CALLING_CANCEL event, userID is the callee;

Note:

Supported from version v1.4.6+ .



```
let handleOnCallCanceled = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.ON_CALL_CANCELED, handleOnCallCanceled);
```

The parameters are described below:

Parameter	Type	Meaning
userID	String	Cancelled User ID

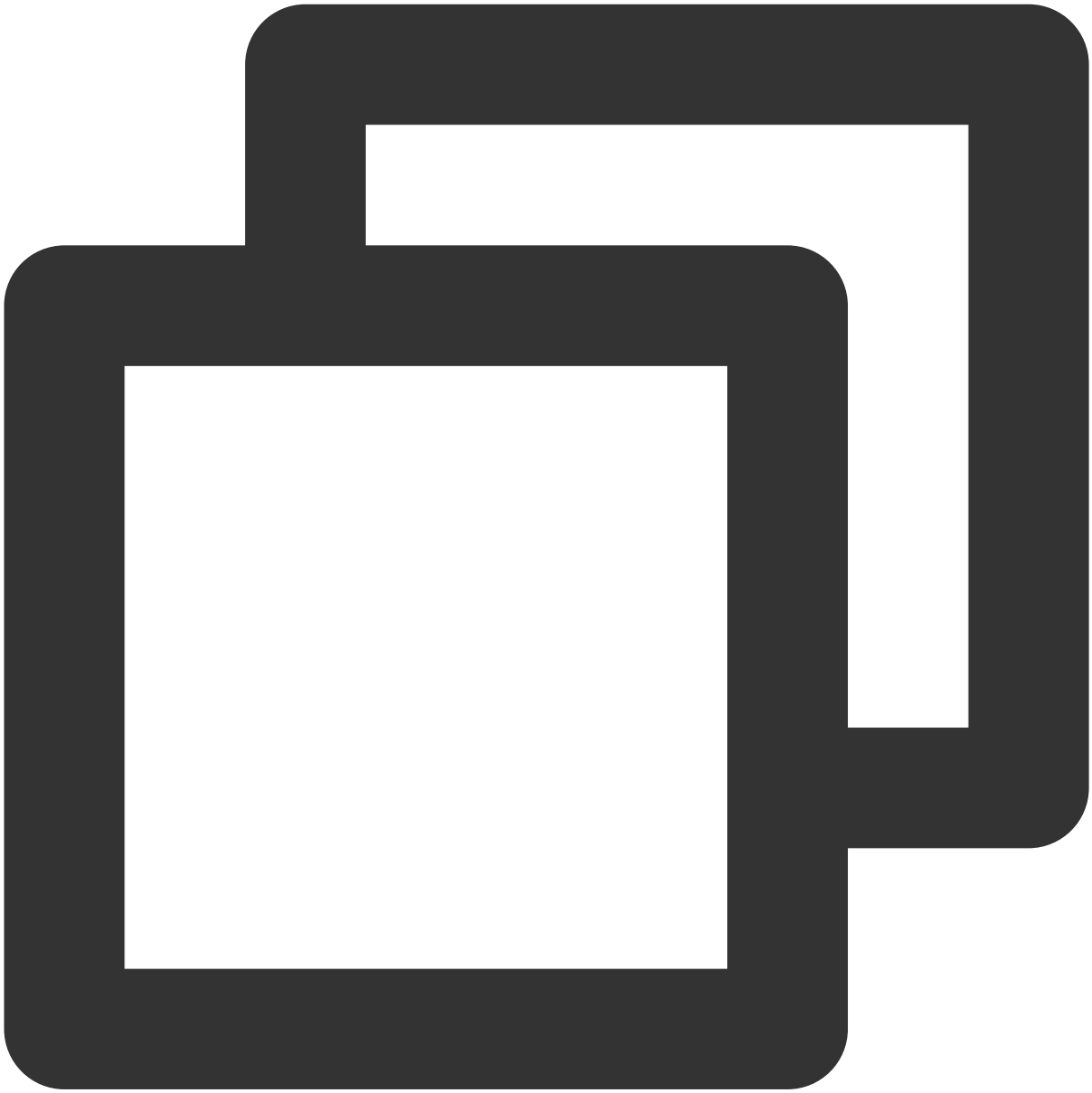
callId	String	Unique ID for this call
roomID	Number	Audio-Video Room ID for this call
callMediaType	Number	Media Type of the call, Video Call, Voice Call
callRole	String	role, Enumeration Type: Caller, Called

ON_CALL_BEGIN

Indicates call connection. Both caller and called can receive it. You can start cloud recording, content review, etc., by listening to this event.

Note:

Supported from version v1.4.6+ .



```
let handleOnCallBegin = function(event) {
    console.log(event);
};
tuiCallEngine.on(TUICallEvent.ON_CALL_BEGIN, handleOnCallBegin);
```

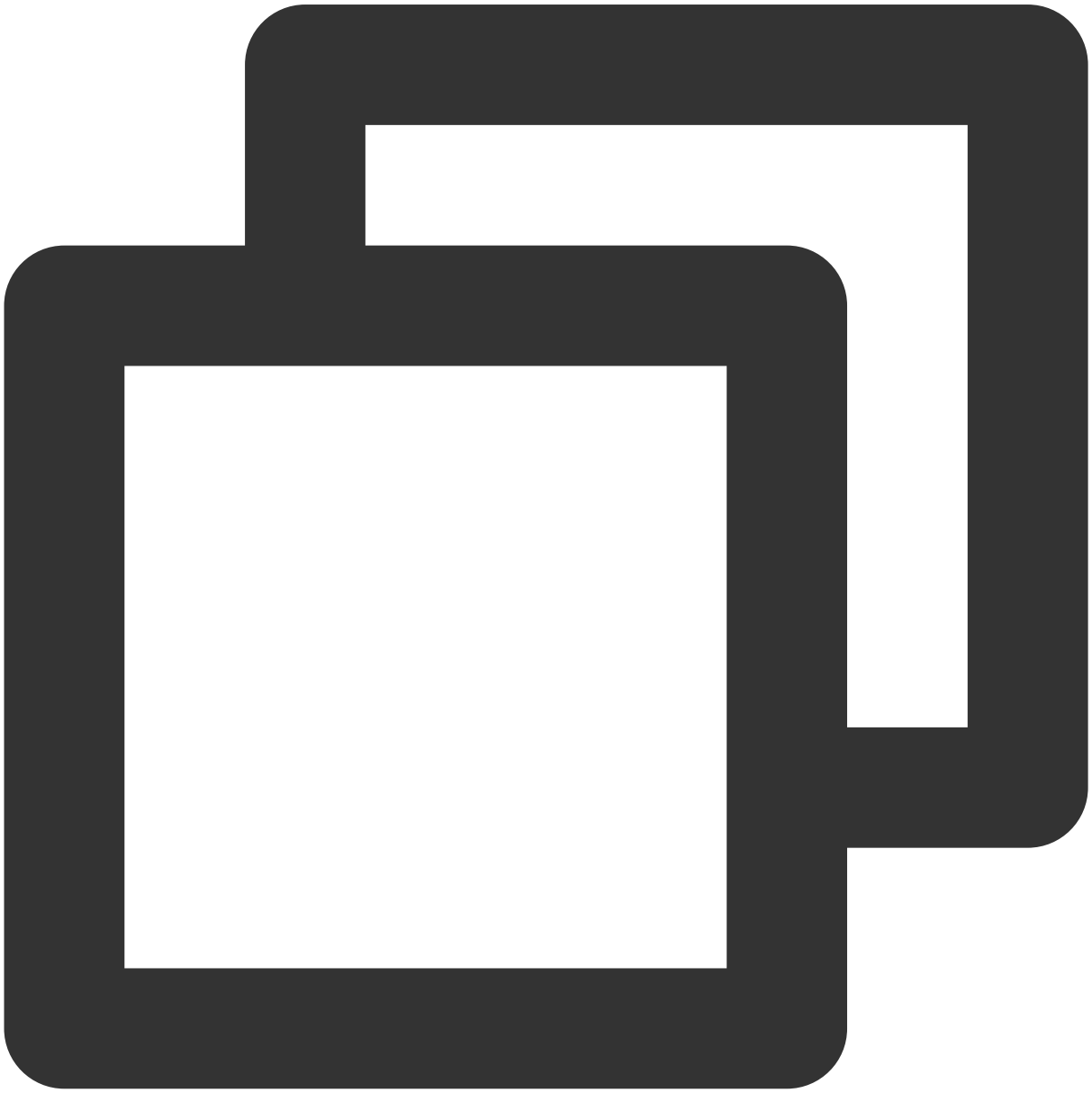
The parameters are described below:

Parameter	Type	Meaning
callId	String	Unique ID for this call

roomID	Number	Audio-Video Room ID for this call
callMediaType	Number	Media Type of the call, Video Call, Voice Call
callRole	String	role, Type: Caller, Called

CALLING_END

Indicates call termination. Both caller and called can trigger this event. You can display information such as call duration, call type, or stop the cloud recording process by listening to this event.



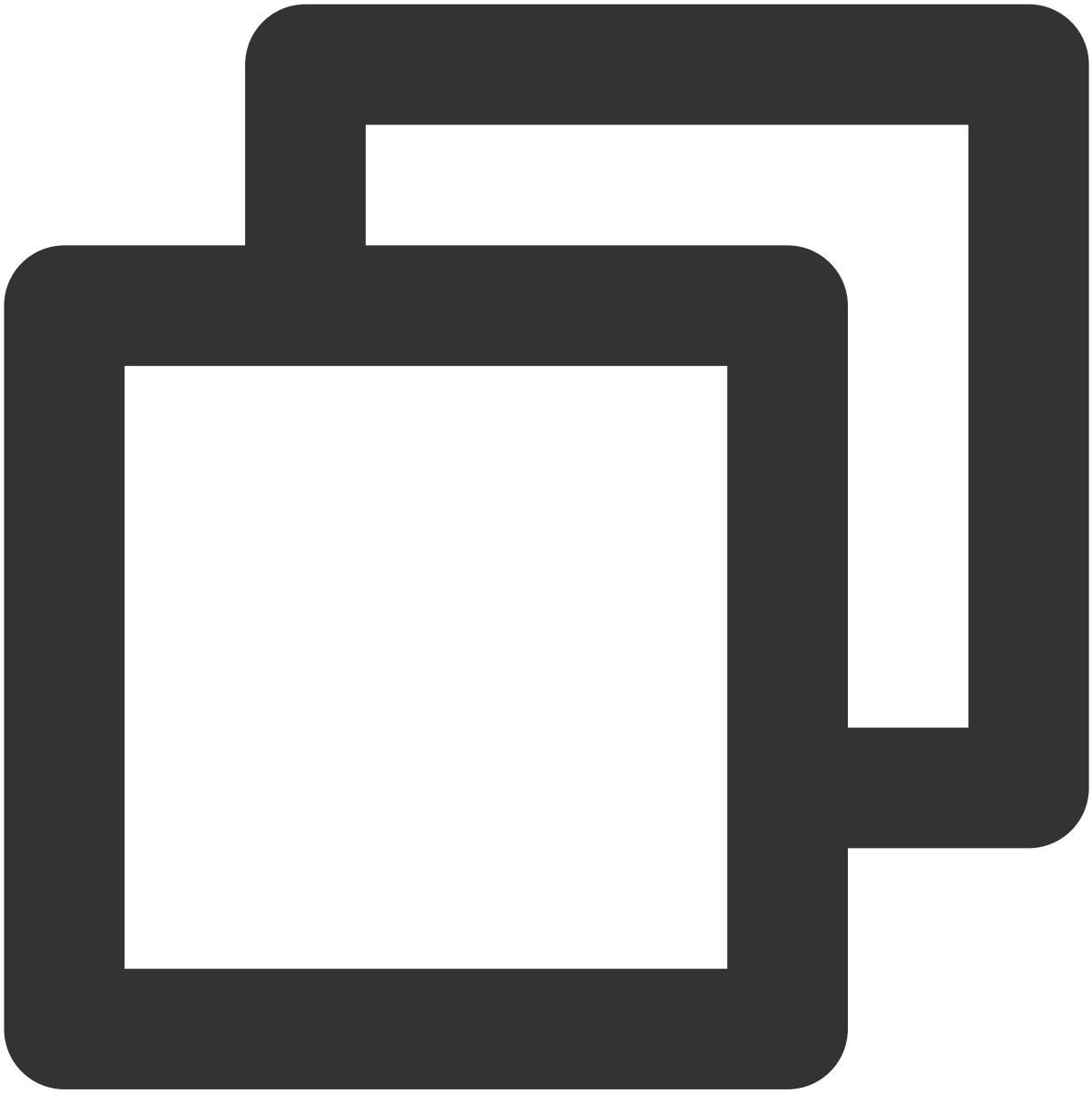
```
let handleCallingEnd = function(event) {
  console.log(event.userID, event.);
};
tuiCallEngine.on(TUICallEvent.CALLING_END, handleCallingEnd);
```

The parameters are described below:

Parameter	Type	Meaning
roomID	Number	Audio-Video Room ID for this call, currently only supports numeric room number, future versions will support character string room numbers
callMediaType	Number	Media Type of the call, Video Call, Voice Call
callRole	String	role, Enumeration Type: Caller ('inviter'), Called ('invitee'), Unknown ('')
totalTime	Number	The duration of this call in seconds
userID	String	userID of the call termination.
callId	String	The unique ID for this call. **v1.4.6+ Supported**
callEnd	Number	The duration of this call (**will be deprecated, Please use totalTime**) in seconds

DEVICED_UPDATED

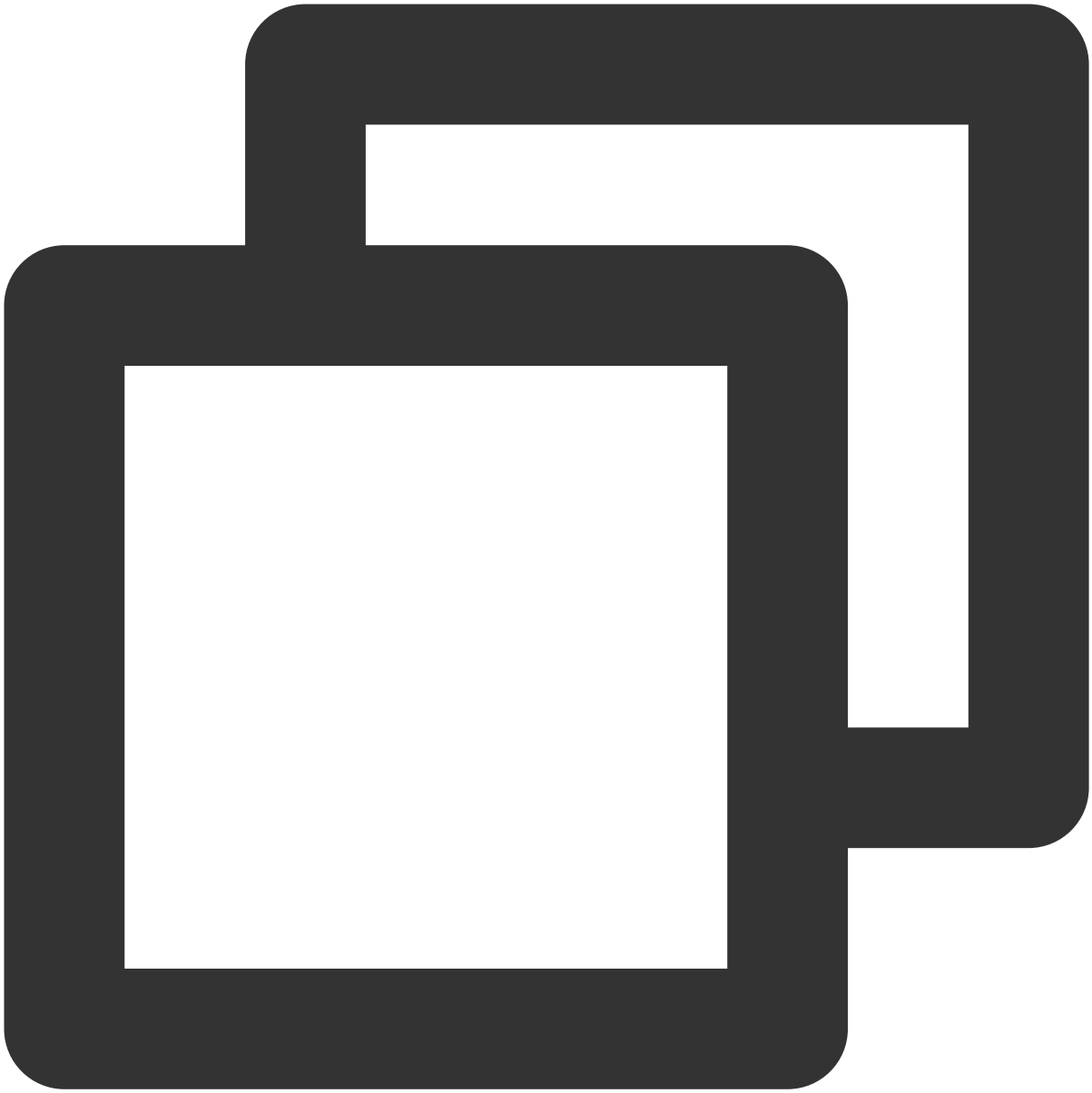
Device list update, this event will be received.



```
let handleDeviceUpdated = function({ microphoneList, cameraList, currentMicrophoneI
  console.log(microphoneList, cameraList, currentMicrophoneID, currentCameraID)
};
tuiCallEngine.on(TUICallEvent.DEVICED_UPDATED, handleDeviceUpdated);
```

CALL_TYPE_CHANGED

Call type switching, this event will be received.



```
let handleCallTypeChanged = function({ oldCallType, newCallType }) {
  console.log(oldCallType, newCallType)
};
tuiCallEngine.on(TUICallEvent.CALL_TYPE_CHANGED, handleDeviceUpdated);
```

The parameters are described below:

Parameter	Type	Meaning
oldCallType	Number	Old call type

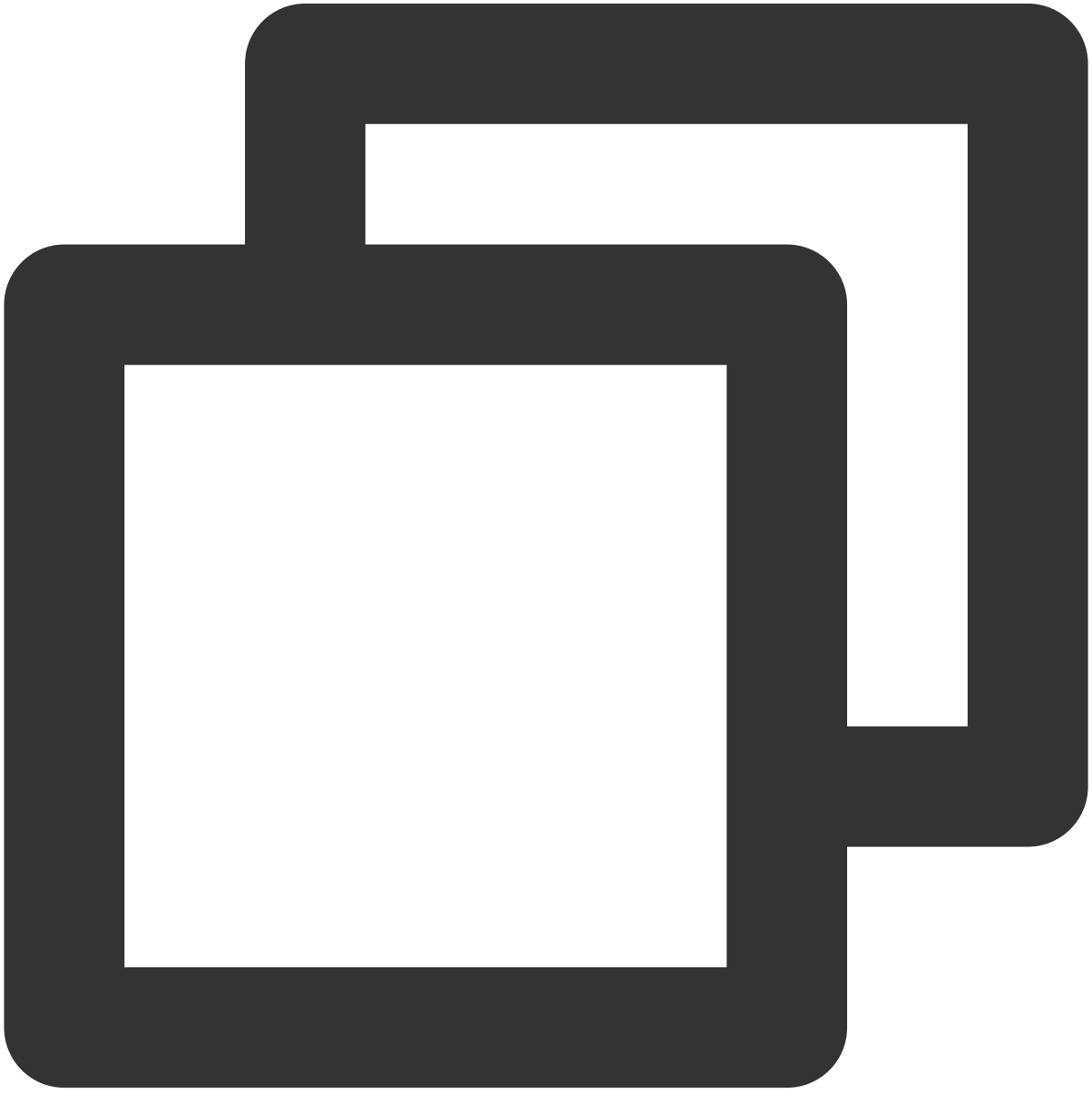
newCallType	Number	New call type

ON_USER_NETWORK_QUALITY_CHANGED

All user network quality events

Note：

v3.0.7+ supported.



```
let handleOnUserNetworkQualityChange = function(event) {
  console.log(event.networkQualityList);
};
```

```
tuiCallEngine.on(TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED, handleOnUserNetworkQ
```

The parameters are described below:

Parameter	Type	Meaning
networkQualityList	Array<Object>	Network status, according to userID, you can get the current network quality of the corresponding user (only local uplink and downlink). For example: `networkQualityList: [{ userId: quality }]` 。 **Network Quality Description:** quality = 0, Network state is unknown quality = 1, Network state is excellent quality = 2, Network state is good quality = 3, Network state is average quality = 4, Network state is poor quality = 5, Network state is very poor quality = 6, Network connection is disconnected

Flutter
API Overview

Last updated：2024-05-23 16:24:18

TUICallKit (UI Included)

TUICallKit is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat.

API	Description
login	Log in
logout	Sign out
setSelfInfo	Sets the user nickname and profile photo.
call	Makes a one-to-one call.
groupCall	Makes a group call.
joinInGroupCall	Joins a group call.
enableMuteMode	Sets whether to turn on the mute mode.
enableFloatWindow	Sets whether to enable floating windows.
setCallingBell	Custom ringtone.
enableVirtualBackground	Turn On/Off the Virtual Background feature

TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description
init	Authenticates the basic audio/video call capabilities.
unInit	The destructor function, which releases resources used by TUICallEngine.

addObserver	Registers an event listener.
removeObserver	Unregisters an event listener.
call	Makes a one-to-one call.
groupCall	Makes a group call.
accept	Accepts a call.
reject	Rejects a call.
hangup	Ends a call.
ignore	Ignores a call.
inviteUser	Invites users to the current group call.
joinInGroupCall	Joins a group call.
switchCallMediaType	Changes the call type, for example, from video call to audio call.
startRemoteView	Subscribes to the video stream of a remote user.
stopRemoteView	Unsubscribes from the video stream of a remote user.
openCamera	Turns the camera on.
closeCamera	Turns the camera off.
switchCamera	Switches between the front and rear cameras.
openMicrophone	Turns the mic on.
closeMicrophone	Turns the mic off.
selectAudioPlaybackDevice	Selects the audio playback device (receiver or speaker).
setSelfInfo	Sets the alias and profile photo.
enableMultiDeviceAbility	Sets whether to enable multi-device login for TUICallEngine (supported by the premium package).
setVideoRenderParams	Set the rendering mode of video image.
setVideoEncoderParams	Set the encoding parameters of video encoder.
queryRecentCalls	Query call record.
deleteRecordCalls	Delete call record.

setBlurBackground	Set Blurry Video Effect
setVirtualBackground	Set Virtual Background Image
setBeautyLevel	Set beauty level, support turning off default beauty.

TUICallObserver

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

API	Description
onError	An error occurred during the call.
onCallReceived	A call was received.
onCallCancelled	The call was canceled.
onCallBegin	The call was connected.
onCallEnd	The call ended.
onCallMediaTypeChanged	The call type changed.
onUserReject	A user declined the call.
onUserNoResponse	A user didn't respond.
onUserLineBusy	A user was busy.
onUserJoin	A user joined the call.
onUserLeave	A user left the call.
onUserVideoAvailable	Whether a user has a video stream.
onUserAudioAvailable	Whether a user has an audio stream.
onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.
onKickedOffline	The current user is kicked offline
onUserSigExpired	Ticket expires while online

TUICallKit

Last updated：2024-05-23 16:24:18

TUICallKit APIs

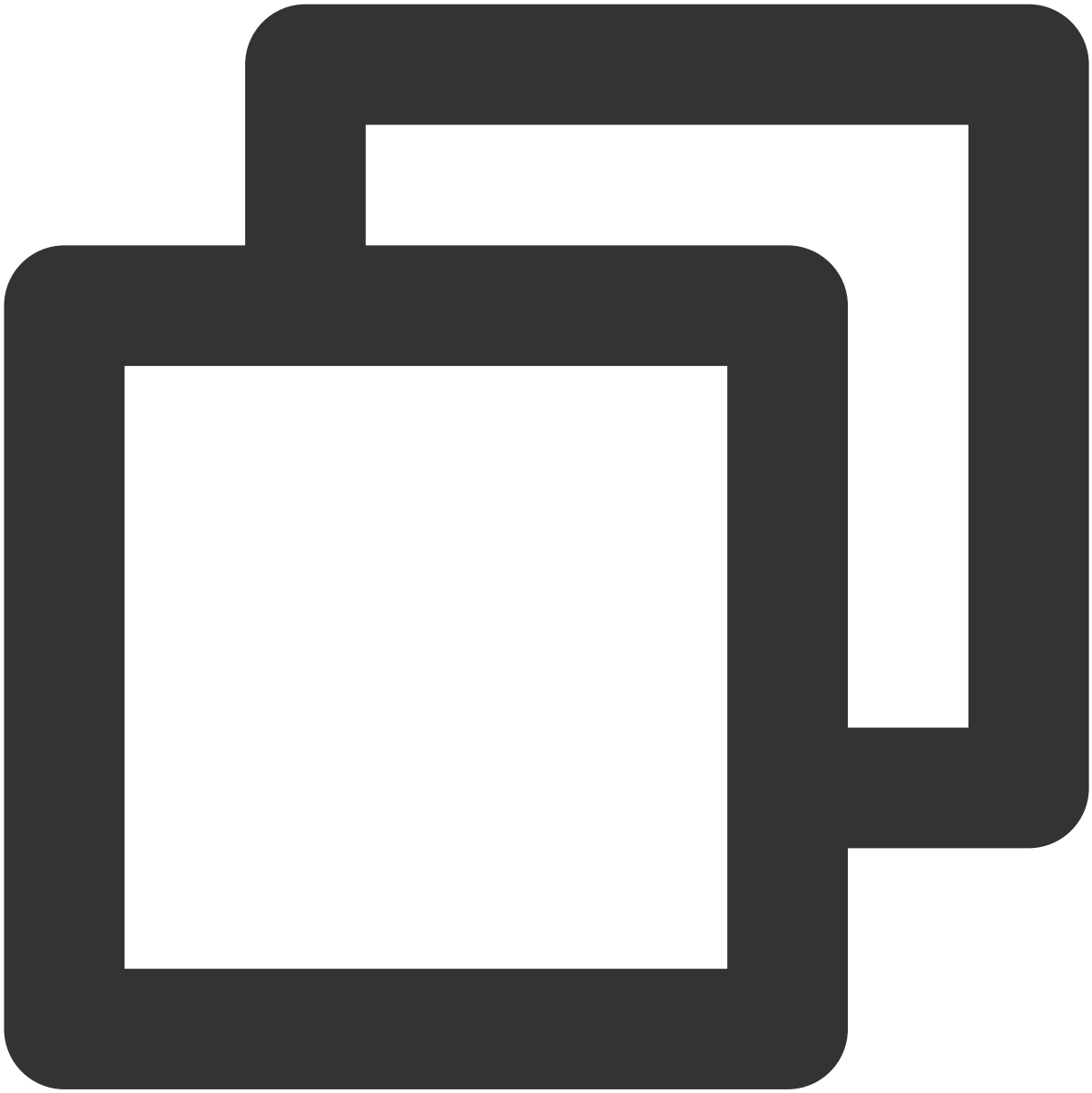
`TUICallKit` is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat. For directions on integration, see Integrating TUICallKit.

API overview

API	Description
login	Log in
logout	Sign out
setSelfInfo	Sets the user nickname and profile photo.
call	Makes a one-to-one call.
groupCall	Makes a group call.
joinInGroupCall	Joins a group call.
enableMuteMode	Sets whether to turn on the mute mode.
enableFloatWindow	Sets whether to enable floating windows.
setCallingBell	Custom ringtone.
enableVirtualBackground	Turn On/Off the Virtual Background feature

Details

login

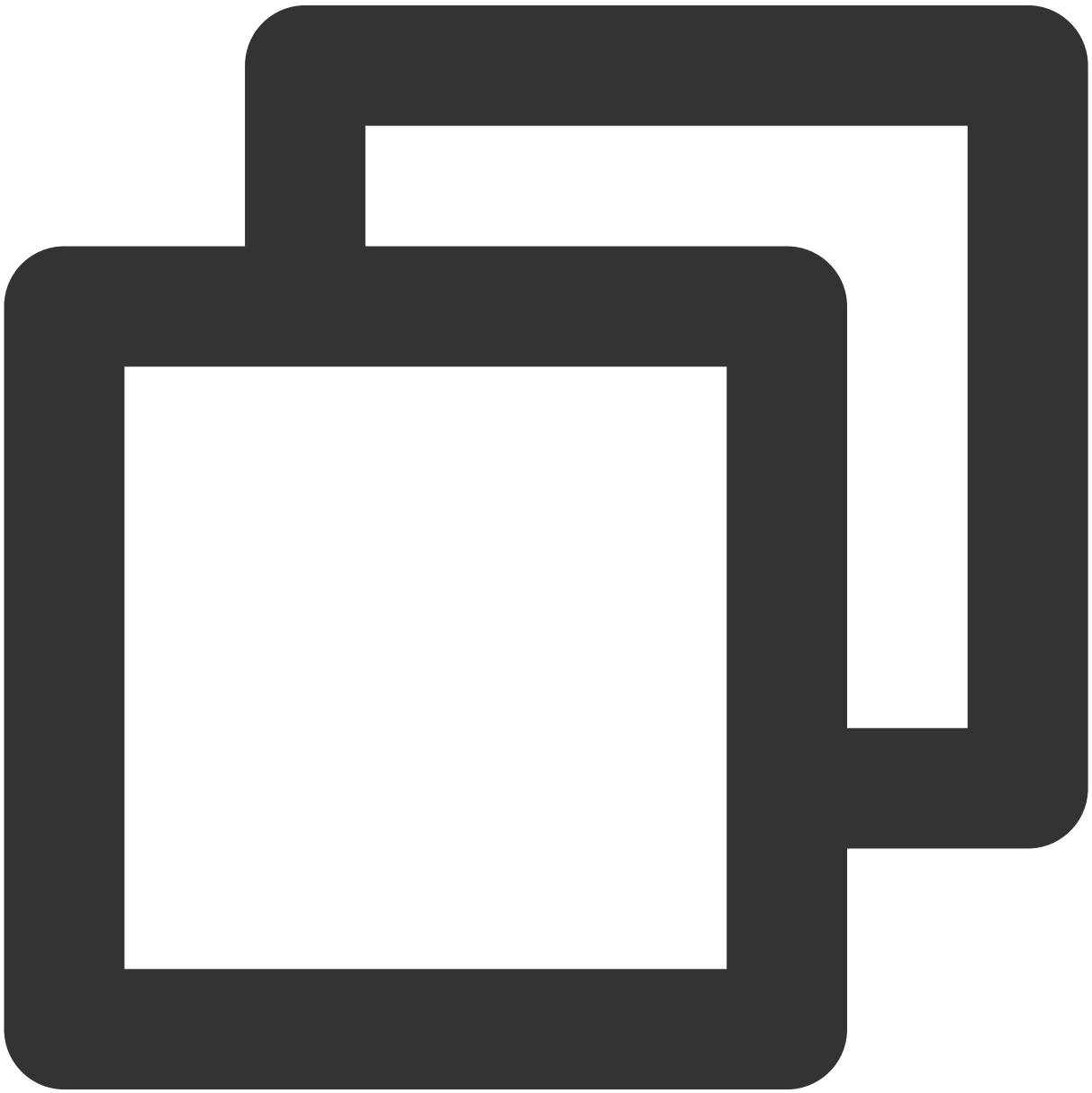


```
Future<TUIResult> login(int sdkAppId, String userId, String userSig)
```

Parameter	Type	Description
sdkAppId	int	User SDKAppID
userId	String	User ID, a string type, can only include English letters (a-z and A-Z), numbers (0-9), hyphens (-), and underscores (_).
userSig	String	User Signature. UserSig is obtained by encrypting information such as sdkAppId and userId

		using the SDKSecretKey(Signature calculation method). It serves as a ticket for authentication, enabling Tencent Cloud to determine if the current user is authorized to use TRTC services.
return value	TUIResult	Contains code and message information: code is empty ("") means the call is successful; code is not empty ("") means the call failed, see message for the reason of failure

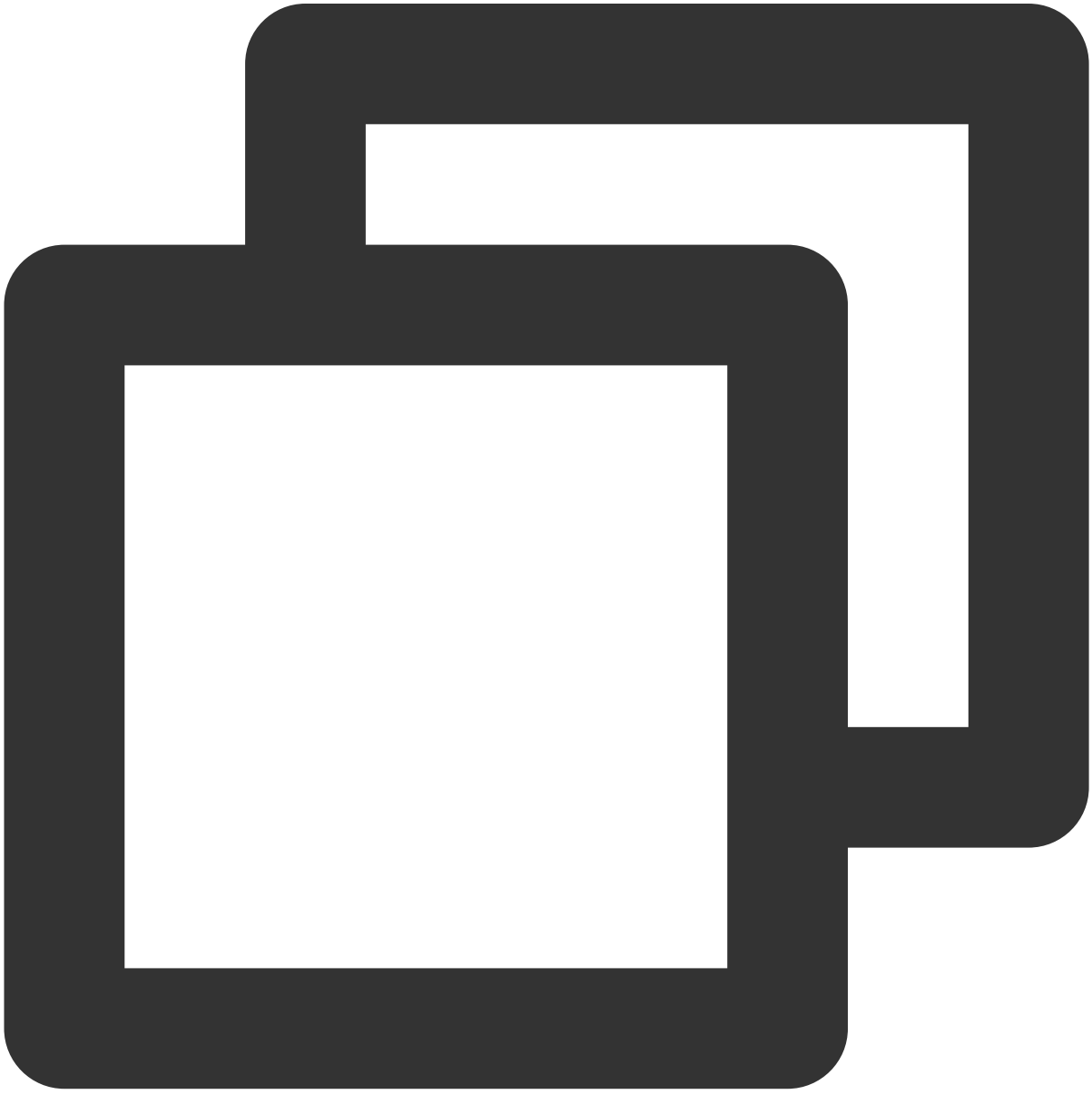
logout



```
Future<void> logout()
```

setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.



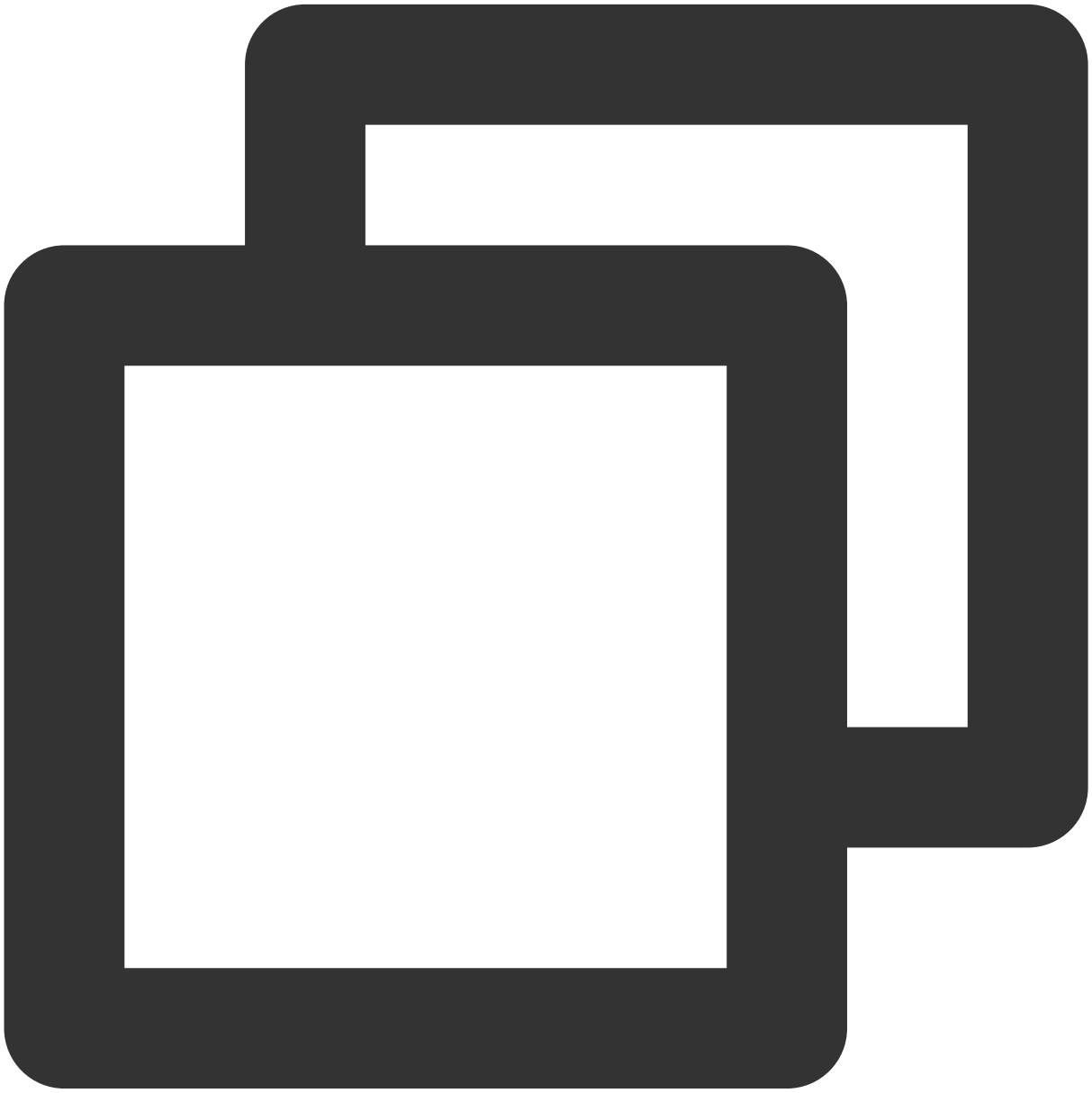
```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

Parameter	Type	Description

nickName	String	The nick name.
avatar	String	The profile photo.
return value	TUIResult	Contains code and message information: code is empty ("") means the call is successful; code is not empty ("") means the call failed, see message for the reason of failure

call

This API is used to make a (one-to-one) call.



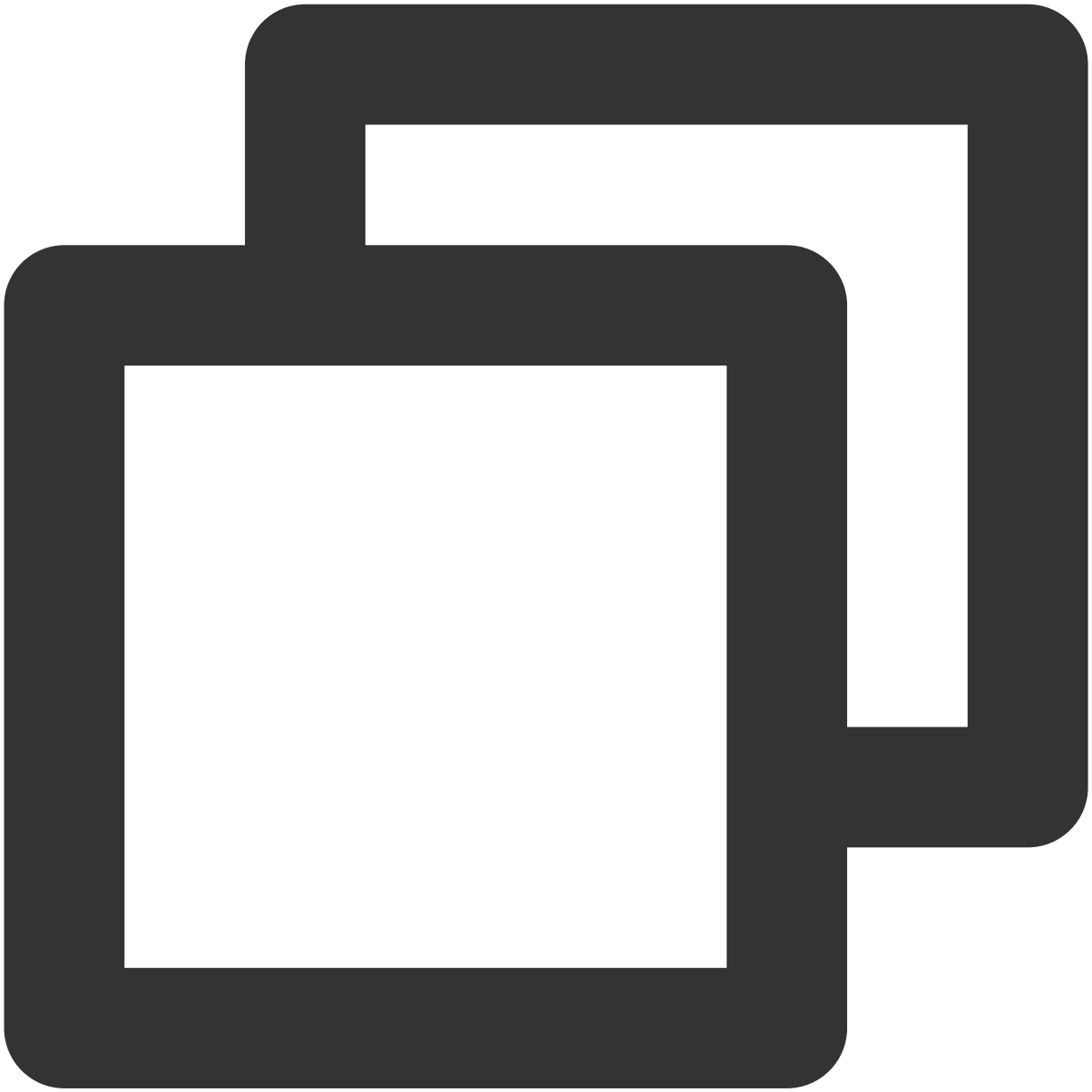
```
Future<void> call(String userId, TUICallMediaType callMediaType, [TUICallParams? pa
```

The parameters are described below:

Parameter	Type	Description
userId	String	The target user ID.
callMediaType	TUICallMediaType	`The call type, which can be video or audio.`

groupCall

This API is used to make a group call.



```
Future<void> groupCall(String groupId, List<String> userIdList, TUICallMediaType ca
```

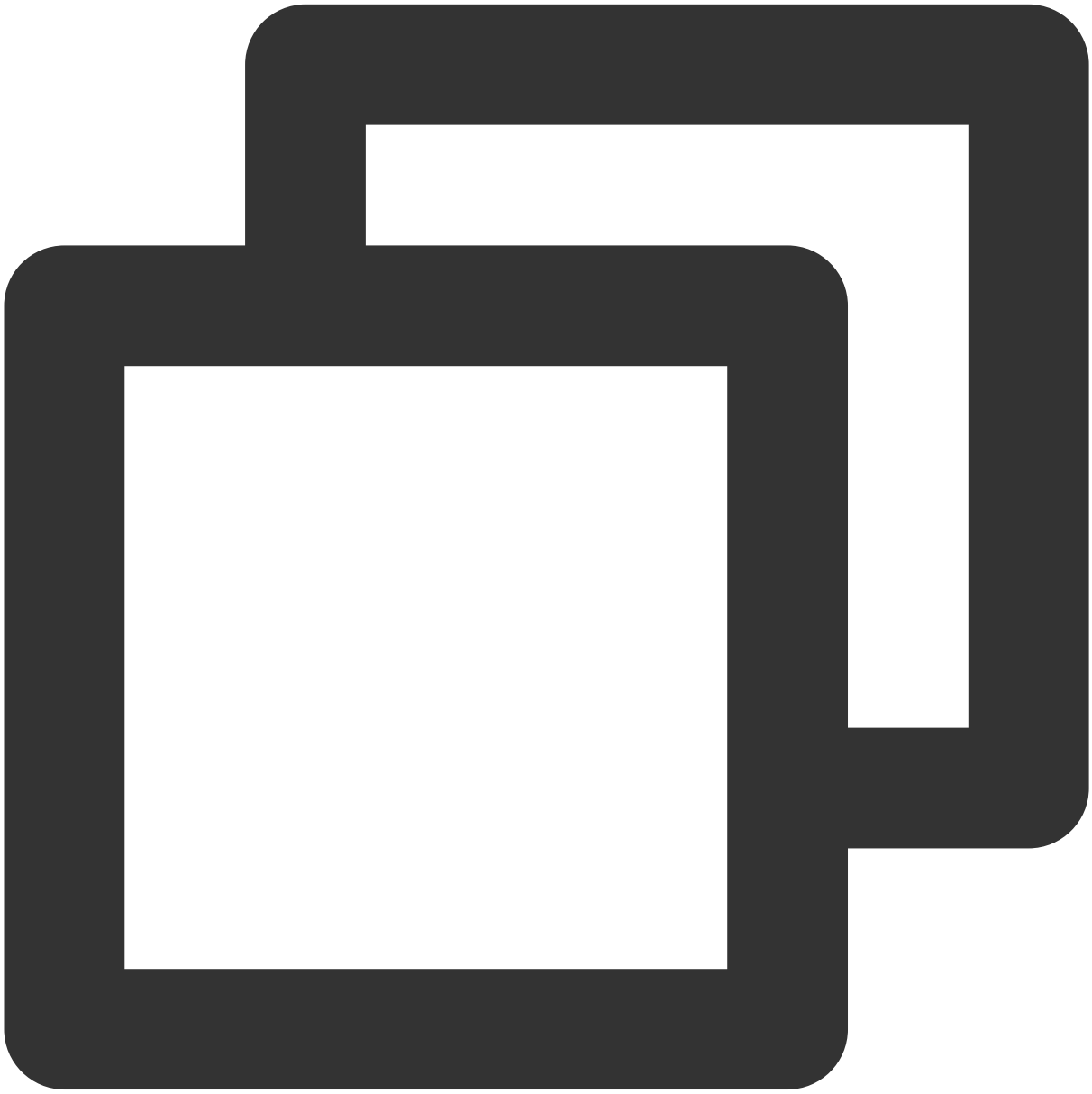
The parameters are described below:

Parameter	Type	Description
groupId	String	The group ID.
userIdList	List<String>	The target user IDs.
callMediaType	TUICallMediaType	`The call type, which can be video or`

		`audio.`

joinInGroupCall

This API is used to join a group call.



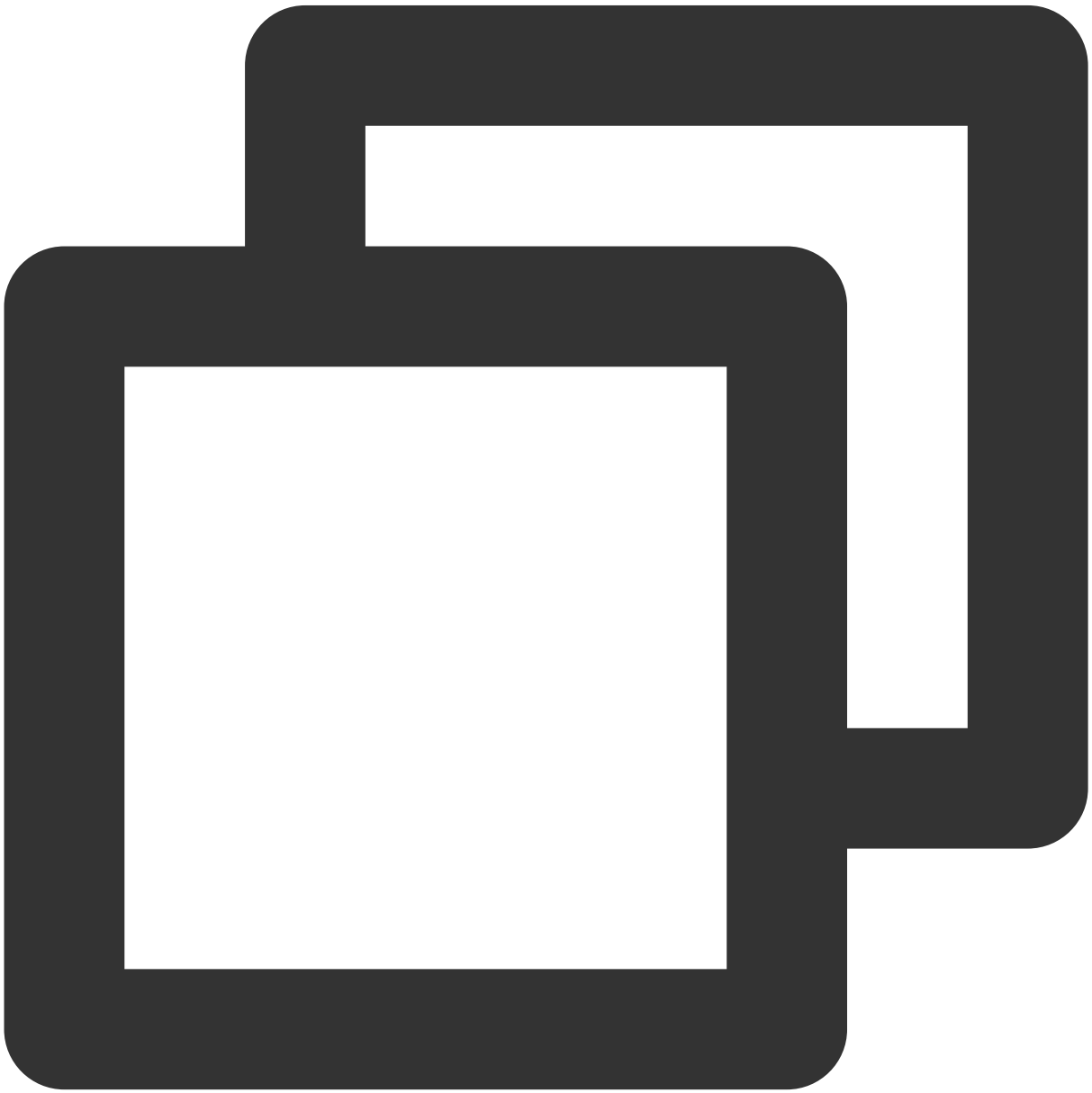
```
Future<void> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaType cal
```

Parameter	Type	Description
roomId	TUIRoomID	The room ID.

groupId	String	The group ID.
callMediaType	TUICallMediaType	`The call type, which can be video or audio.`

enableMuteMode

This API is used to set whether to turn on the mute mode.

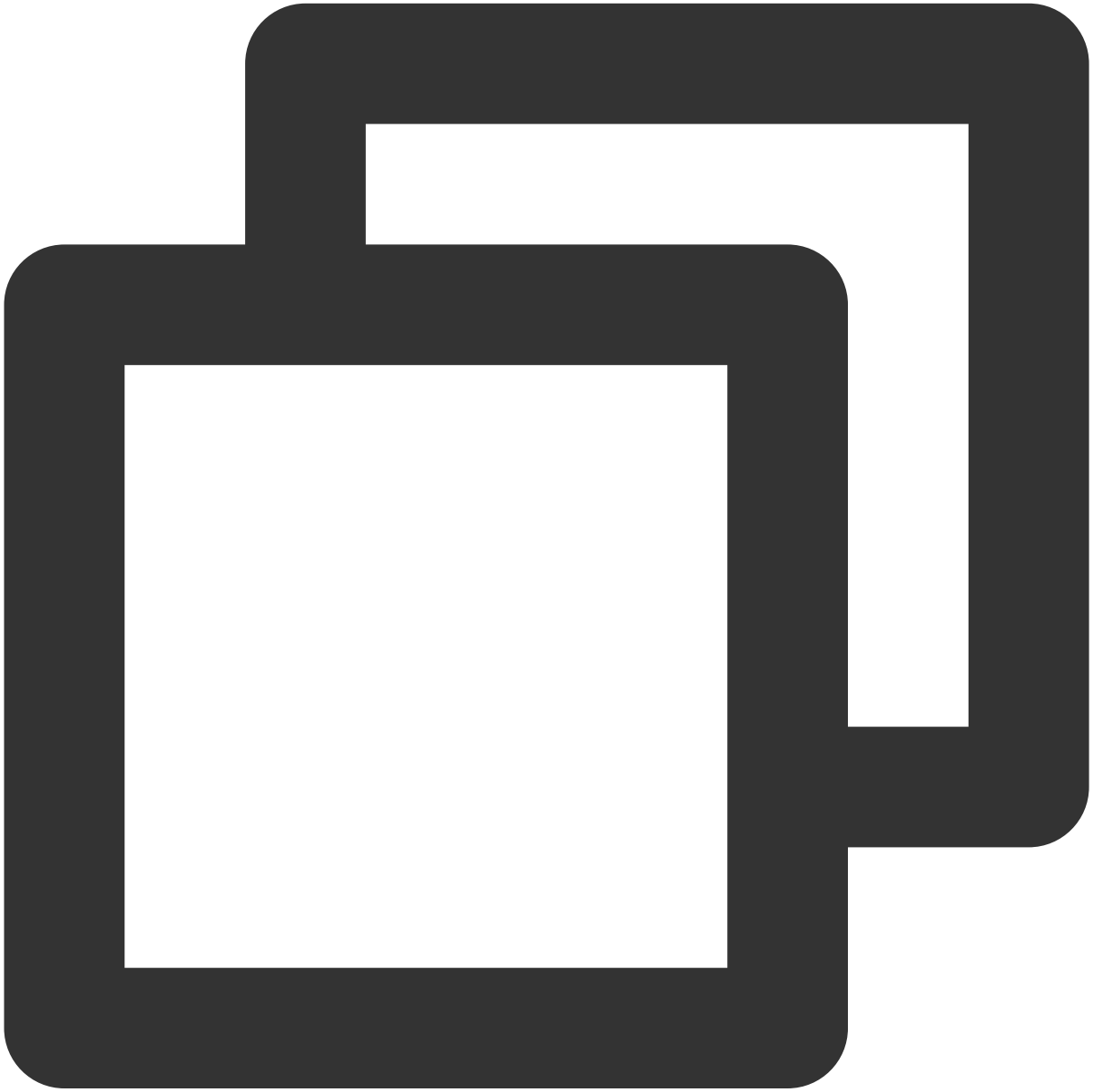


```
Future<void> enableMuteMode(bool enable)
```

Parameter	Type	Description
enable	bool	Turn on and off the mute; true means to turn on the mute

enableFloatWindow

This API is used to set whether to enable floating windows. The default value is `false`, and the floating window button in the top left corner of the call view is hidden. If it is set to `true`, the button will become visible.

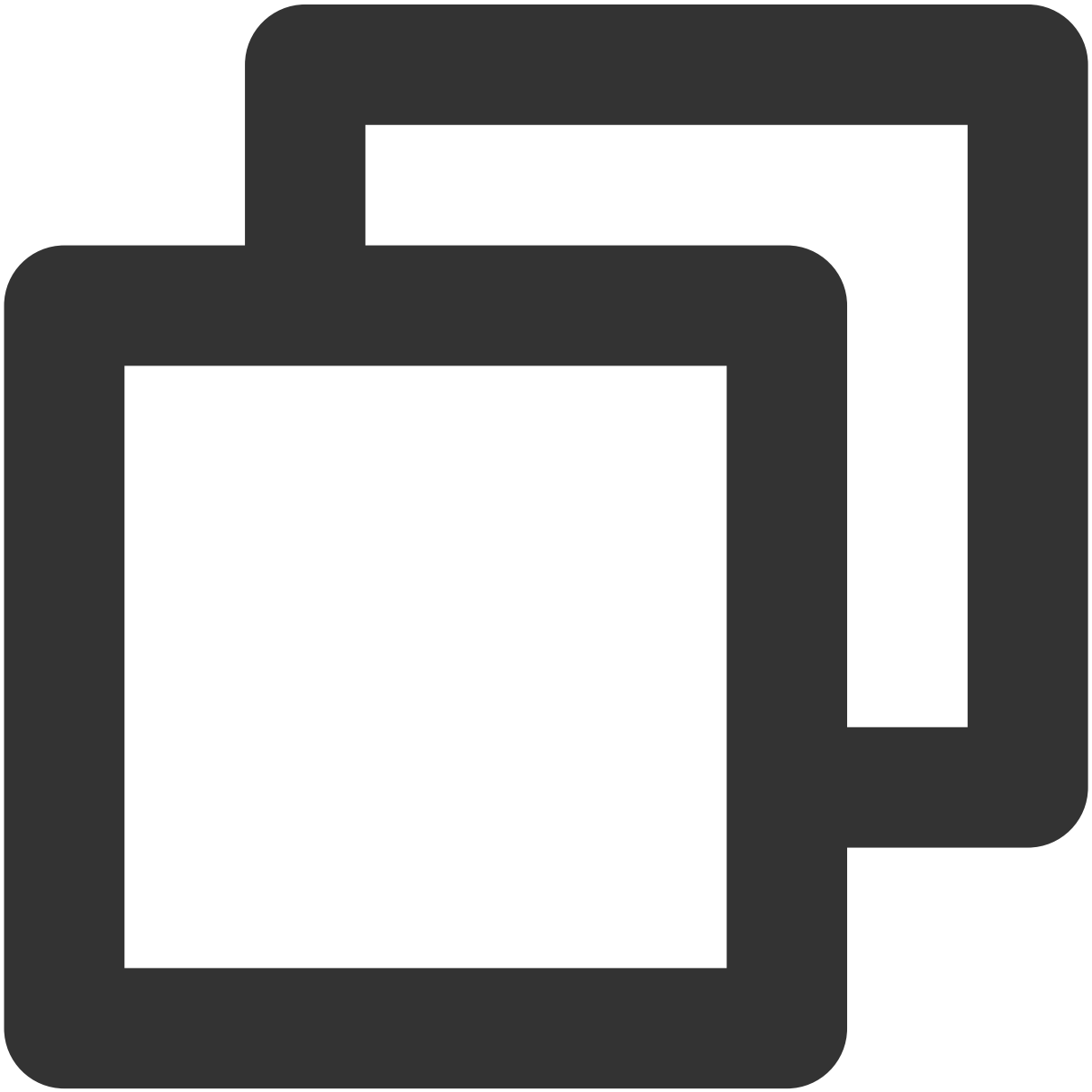


```
Future<void> enableFloatWindow(bool enable)
```

Parameter	Type	Description
enable	bool	The default value is false, and the floating window button in the top left corner of the call view is hidden. If it is set to true, the button will become visible.

setCallingBell

Custom ringtone.

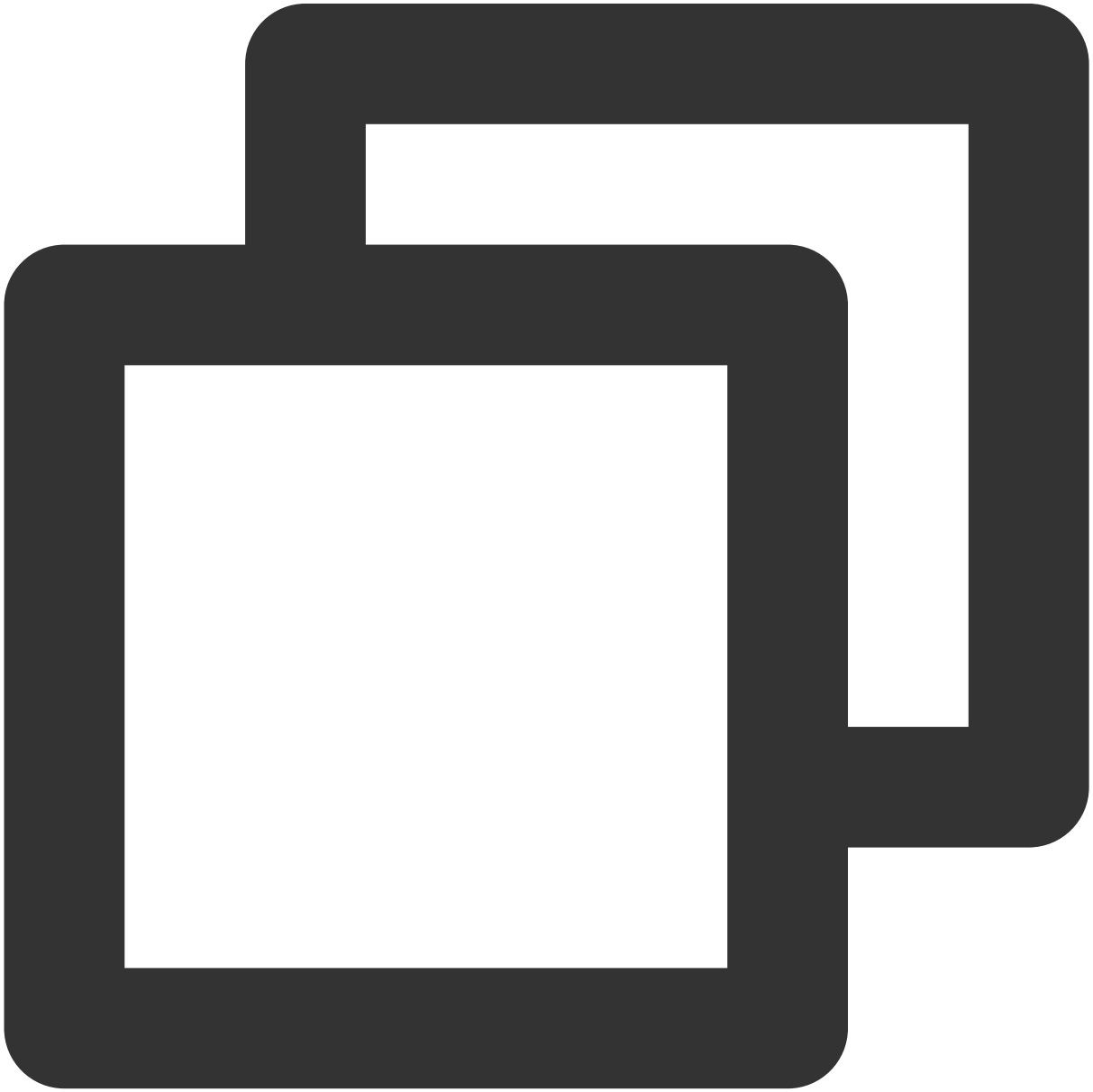


```
Future<void> setCallingBell(String assetName)
```

Parameter	Type	Description
assetName	String	The path of the ringtone. The ringtone file needs to be added to the assets resource of the main project.

enableVirtualBackground

Turn On/Off the Virtual Background feature. After enabling the Virtual Background feature, you can display the Blurry Background feature button on the UI. Clicking the button will directly enable the Blurry Background feature.



```
Future<void> enableVirtualBackground(bool enable)
```

Parameter	Type	Meaning
enable	bool	Turn on, turn off mute; true means mute is on

TUICallEngine

Last updated：2024-05-23 16:24:18

TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

Overview

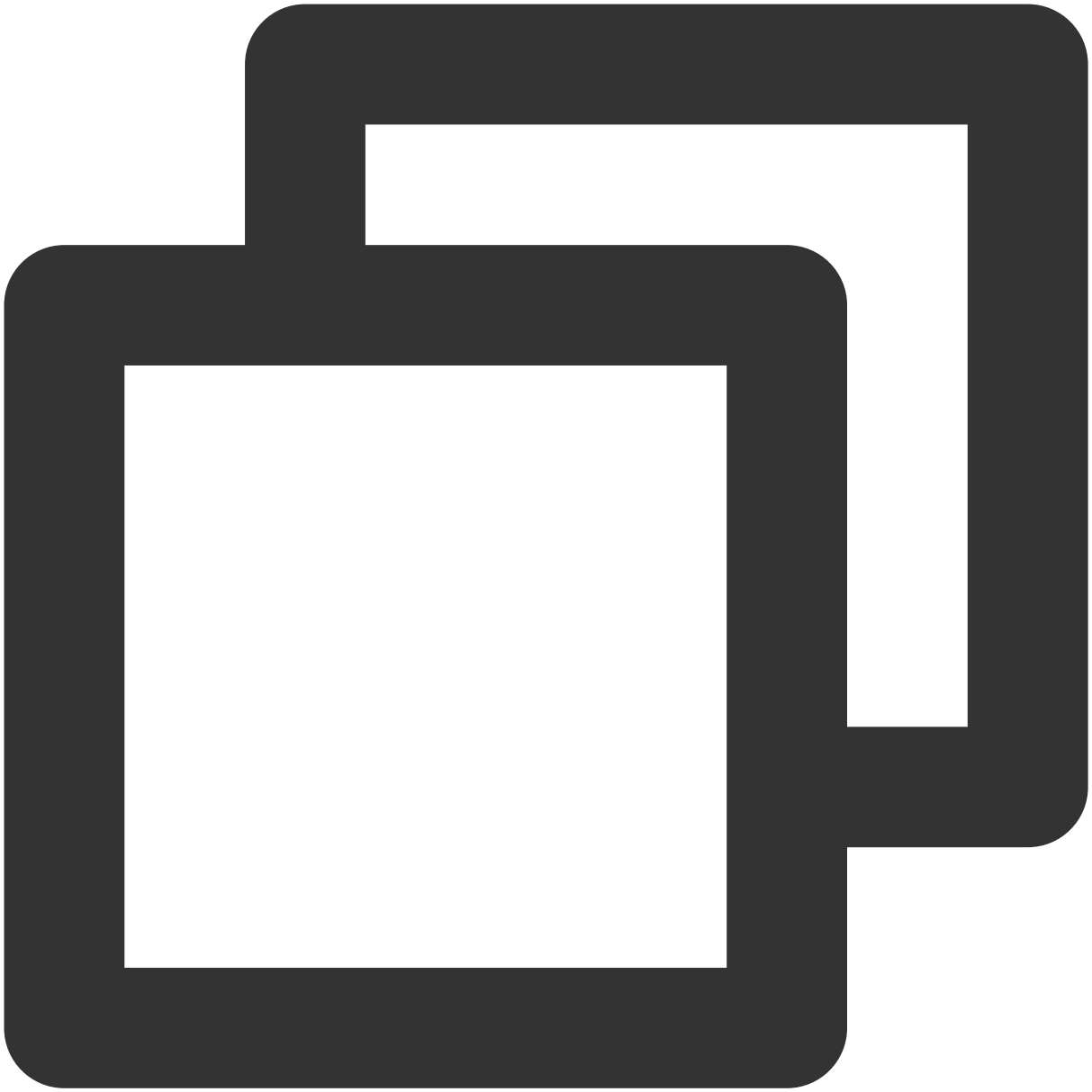
API	Description
init	Authenticates the basic audio/video call capabilities.
unInit	The destructor function, which releases resources used by TUICallEngine.
addObserver	Registers an event listener.
removeObserver	Unregisters an event listener.
call	Makes a one-to-one call.
groupCall	Makes a group call.
accept	Accepts a call.
reject	Rejects a call.
hangup	Ends a call.
ignore	Ignores a call.
inviteUser	Invites users to the current group call.
joinInGroupCall	Joins a group call.
switchCallMediaType	Changes the call type, for example, from video call to audio call.
startRemoteView	Subscribes to the video stream of a remote user.
stopRemoteView	Unsubscribes from the video stream of a remote user.
openCamera	Turns the camera on.

closeCamera	Turns the camera off.
switchCamera	Switches between the front and rear cameras.
openMicrophone	Turns the mic on.
closeMicrophone	Turns the mic off.
selectAudioPlaybackDevice	Selects the audio playback device (receiver or speaker).
setSelfInfo	Sets the alias and profile photo.
enableMultiDeviceAbility	Sets whether to enable multi-device login for TUICallEngine (supported by the premium package).
setVideoRenderParams	Set the rendering mode of video image.
setVideoEncoderParams	Set the encoding parameters of video encoder.
queryRecentCalls	Query call record.
deleteRecordCalls	Delete call record.
setBlurBackground	Set Blurry Video Effect
setVirtualBackground	Set Virtual Background Image
setBeautyLevel	Set beauty level, support turning off default beauty.

Details

init

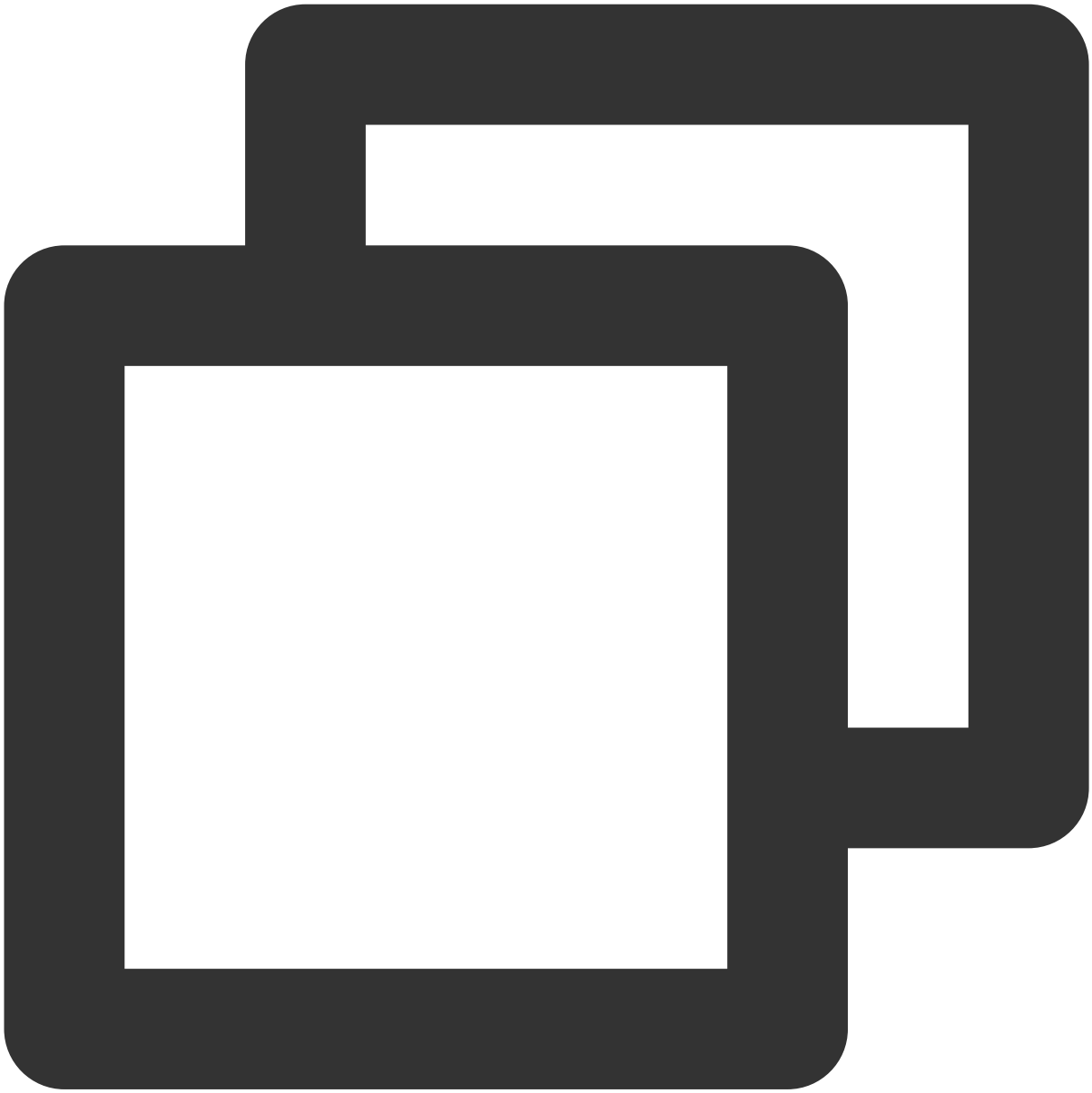
This API is used to initialize `TUICallEngine`. Call it to authenticate the call service and perform other required actions before you call other APIs.



```
Future<TUIResult> init(int sdkAppID, String userId, String userSig)
```

unInit

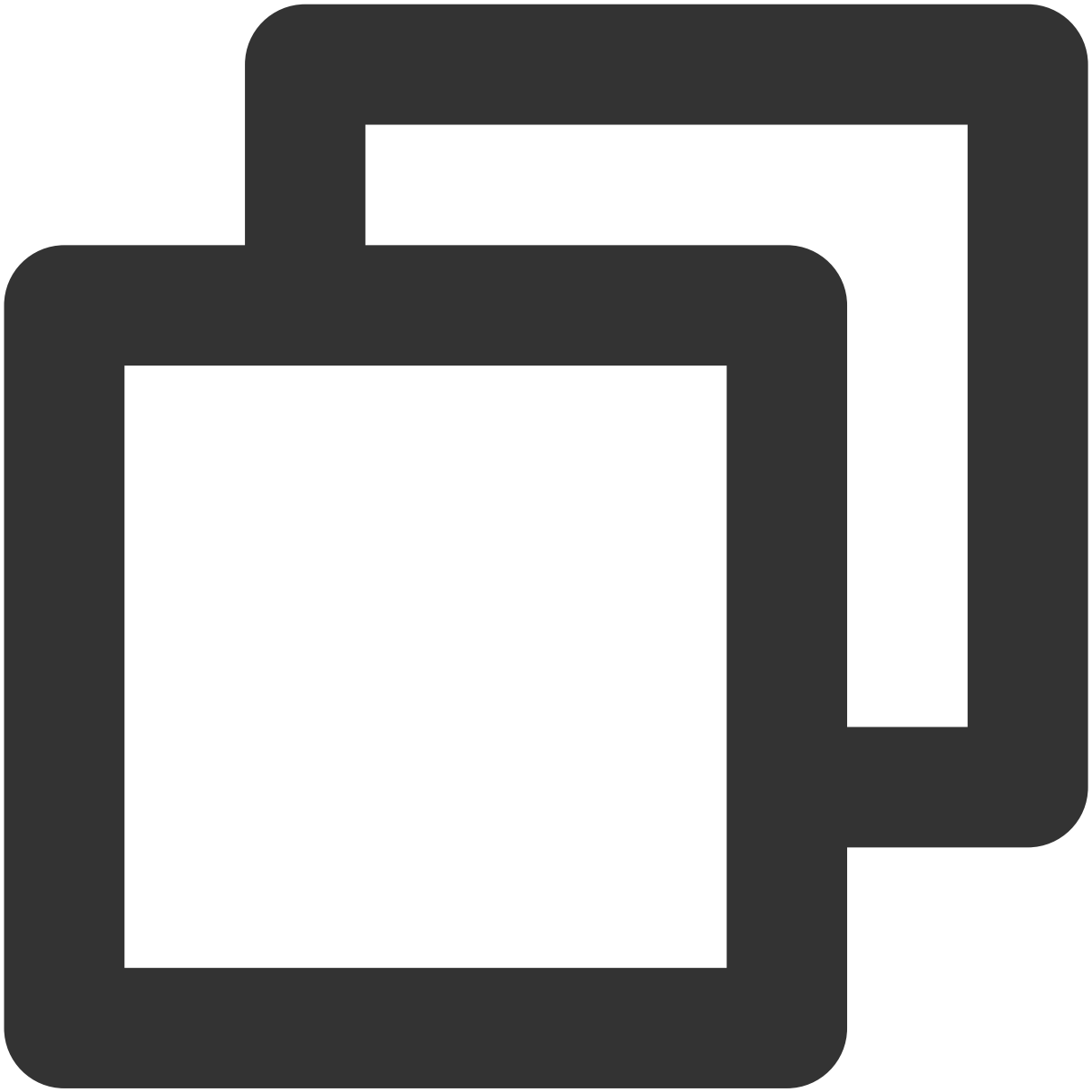
The destructor function, which releases resources used by TUICallEngine.



```
Future<TUIResult> unInit()
```

addObserver

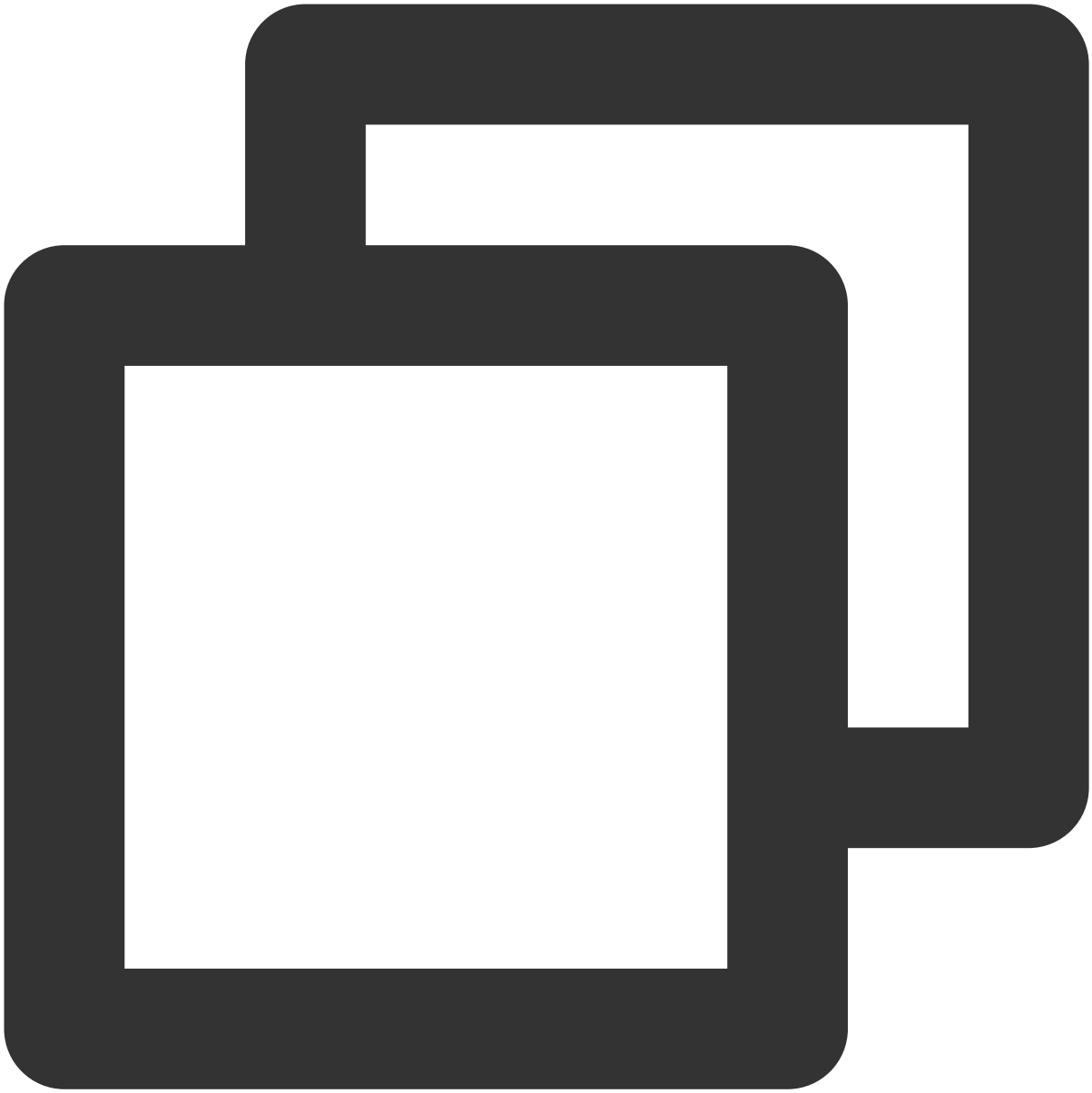
This API is used to register an event listener to listen for `TUICallObserver` events.



```
Future<void> addObserver(TUICallObserver observer)
```

removeObserver

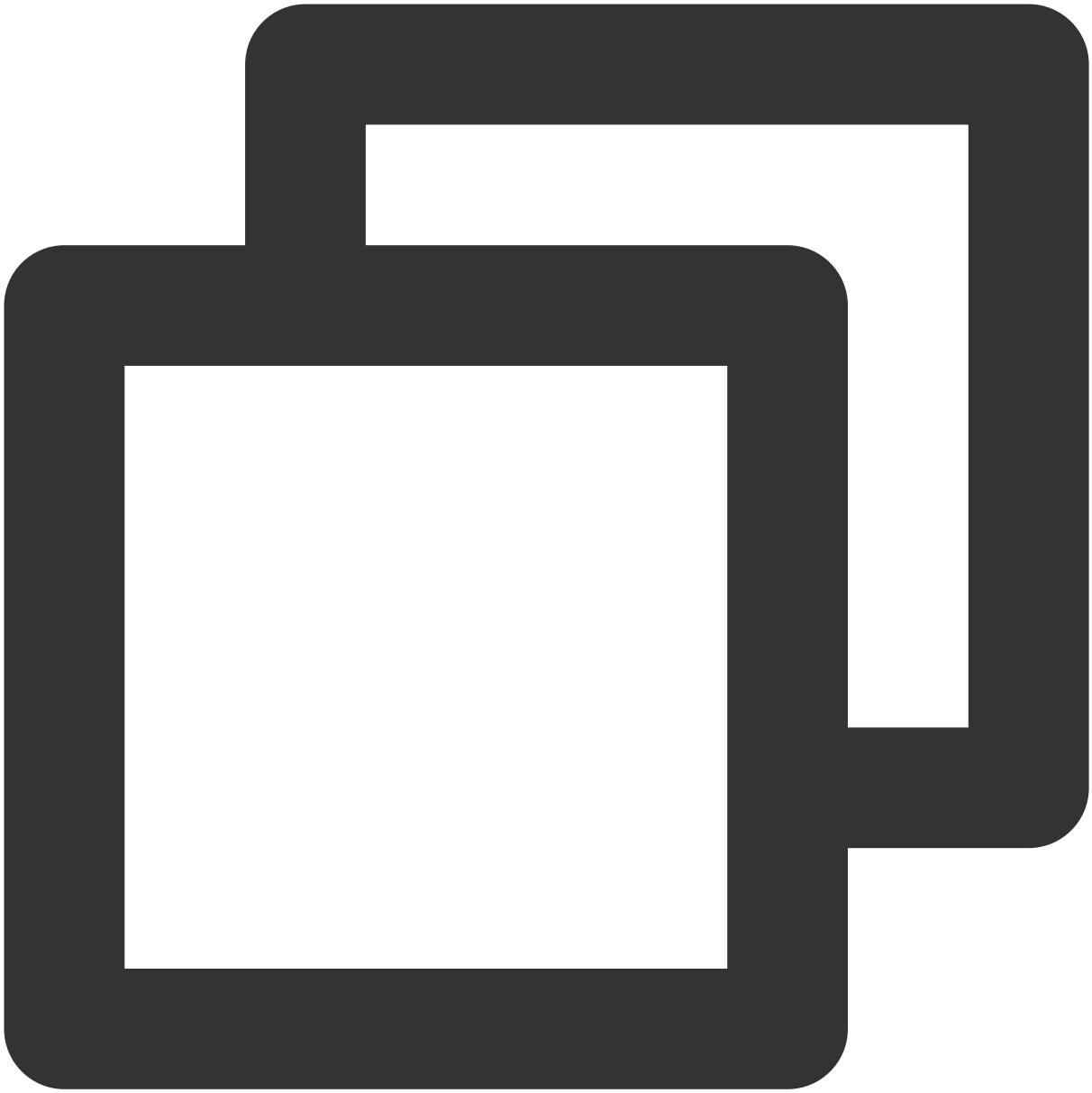
This API is used to unregister an event listener.



```
Future<void> removeObserver(TUICallObserver observer)
```

call

This API is used to make a (one-to-one) call.



```
Future<TUIResult> call(String userId, TUICallMediaType mediaType, TUICallParams par
```

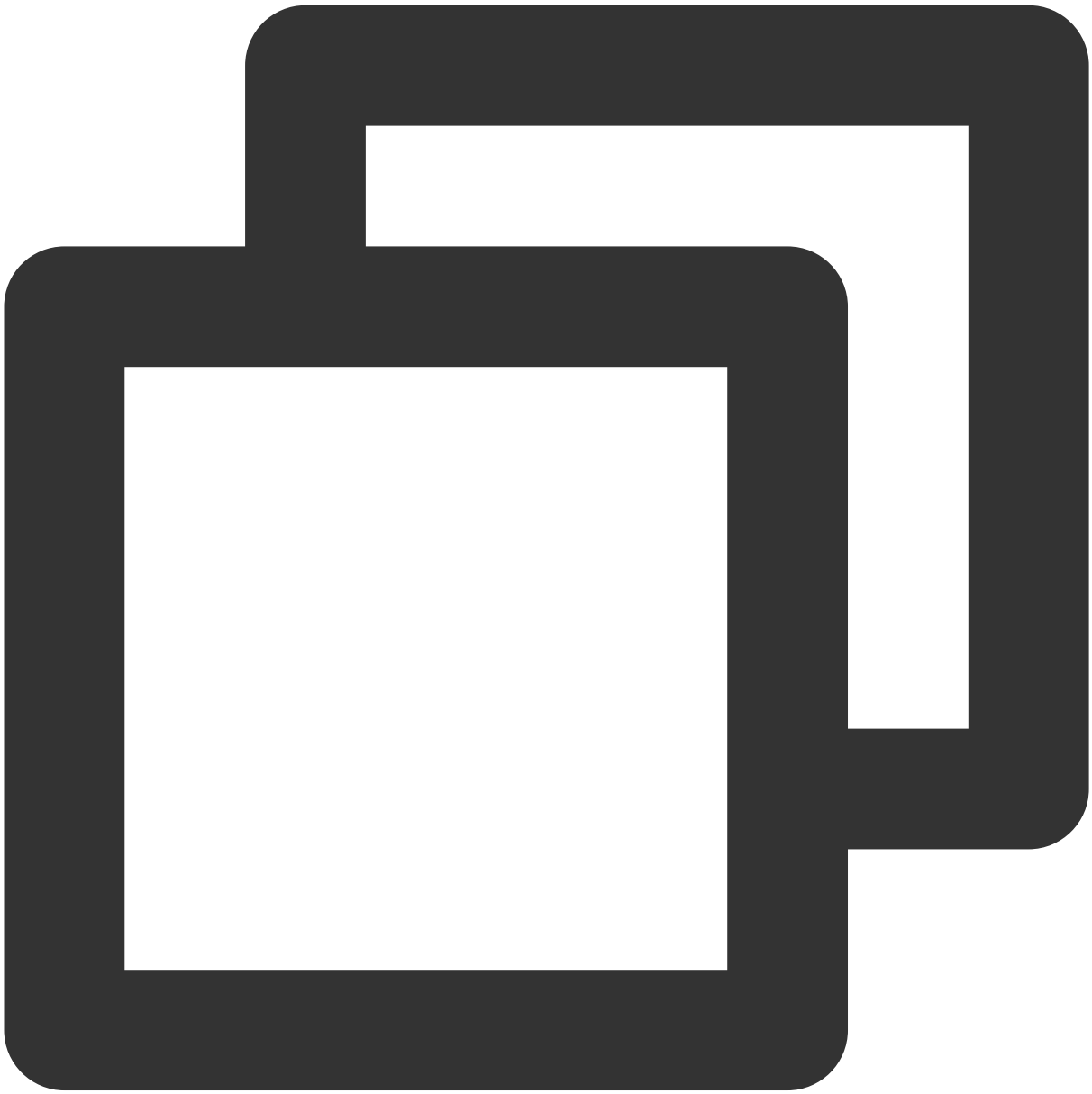
Parameter	Type	Description
userId	String	The target user ID.
mediaType	TUICallMediaType	The call type, which can be video or audio.
params	TUICallParams	An additional parameter, such as roomID, call timeout, offline push info, etc

groupCall

This API is used to make a group call.

Note：

Before making a group call, you need to create an IM group first.



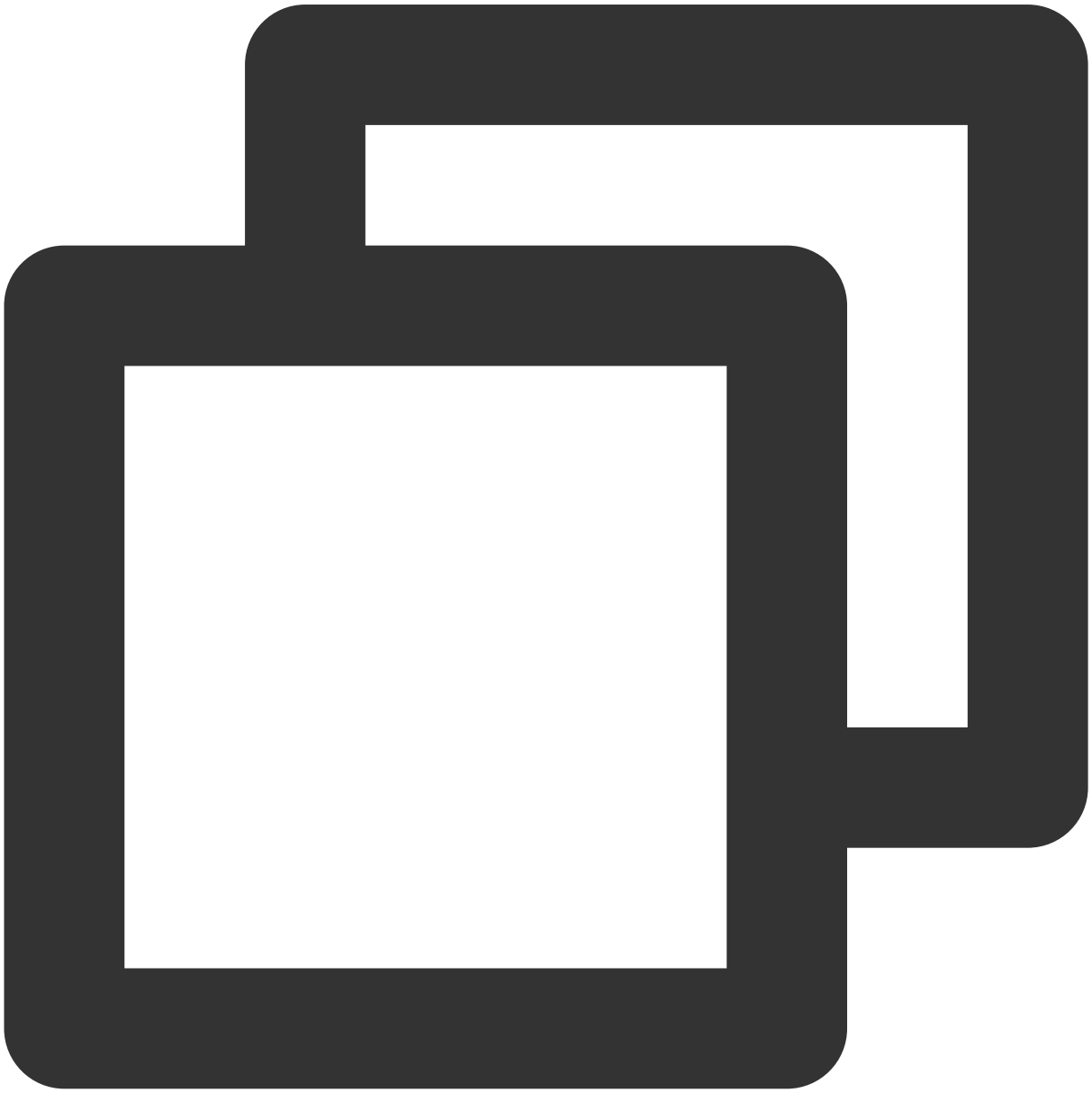
```
Future<TUIResult> groupCall(String groupId, List<String> userIdList, TUICallMediaTy
```

Parameter	Type	Description
groupId	String	The group ID.

userIdList	List<String>	The target user IDs.
mediaType	TUICallMediaType	The call type, which can be video or audio.
params	TUICallParams	An additional parameter. such as roomID, call timeout, offline push info, etc

accept

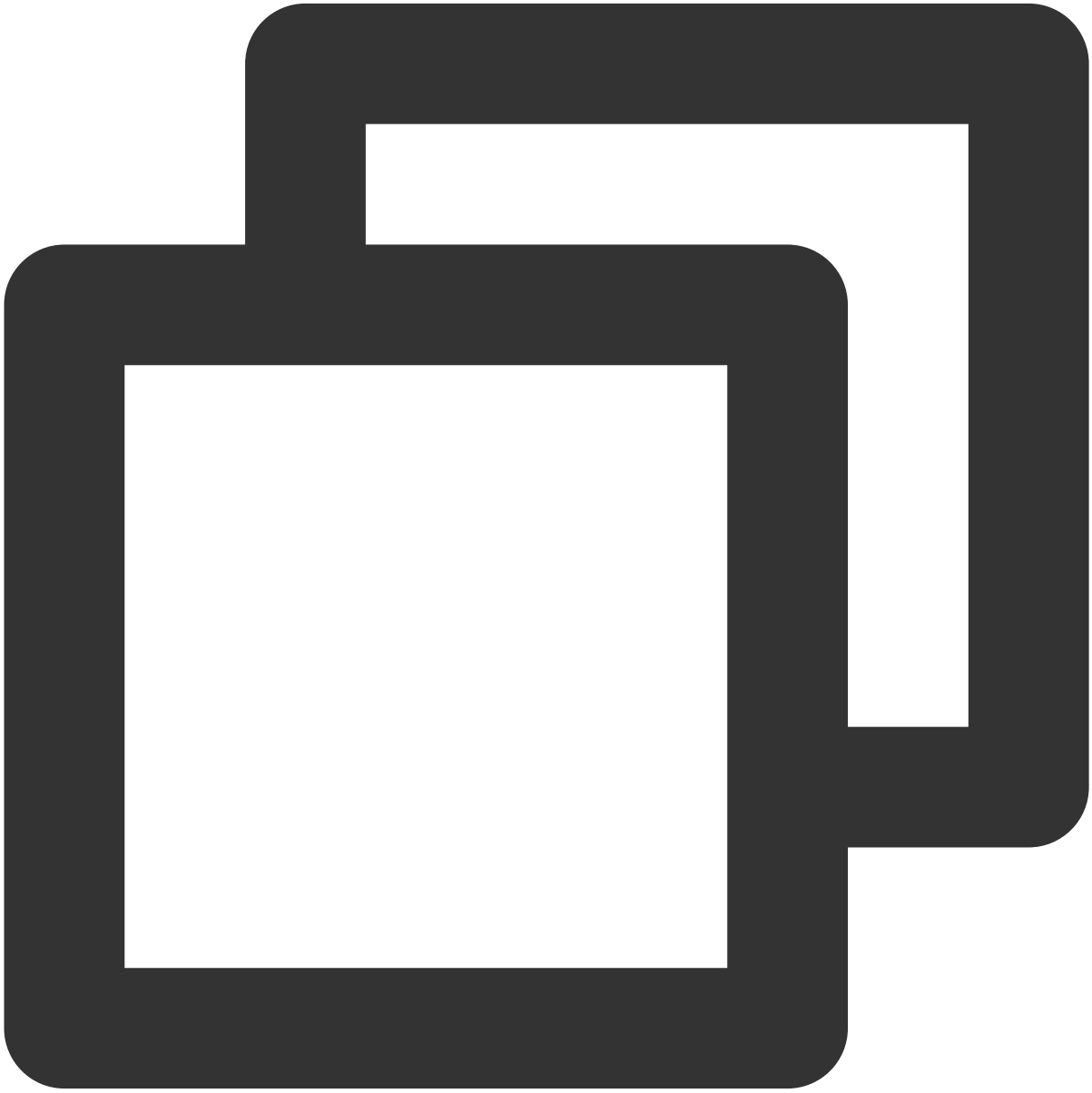
This API is used to accept a call. After receiving the `onCallReceived()` callback, you can call this API to accept the call.



```
Future<TUIResult> accept()
```

reject

This API is used to reject a call. After receiving the `onCallReceived()` callback, you can call this API to reject the call.

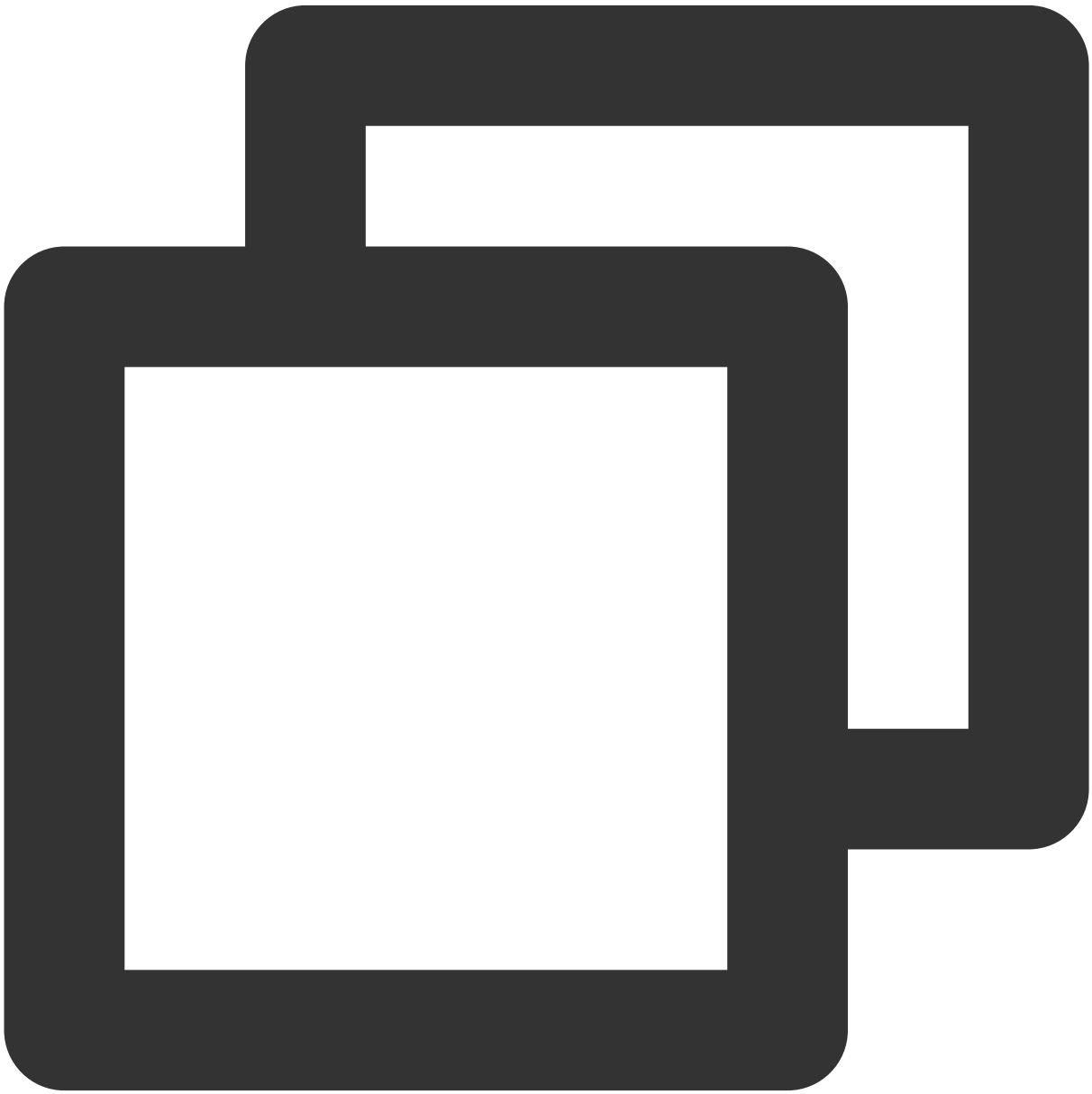


```
Future<TUIResult> reject()
```

ignore

This API is used to ignore a call. After receiving the `onCallReceived()` , you can call this API to ignore the call. The caller will receive the `onUserLineBusy` callback.

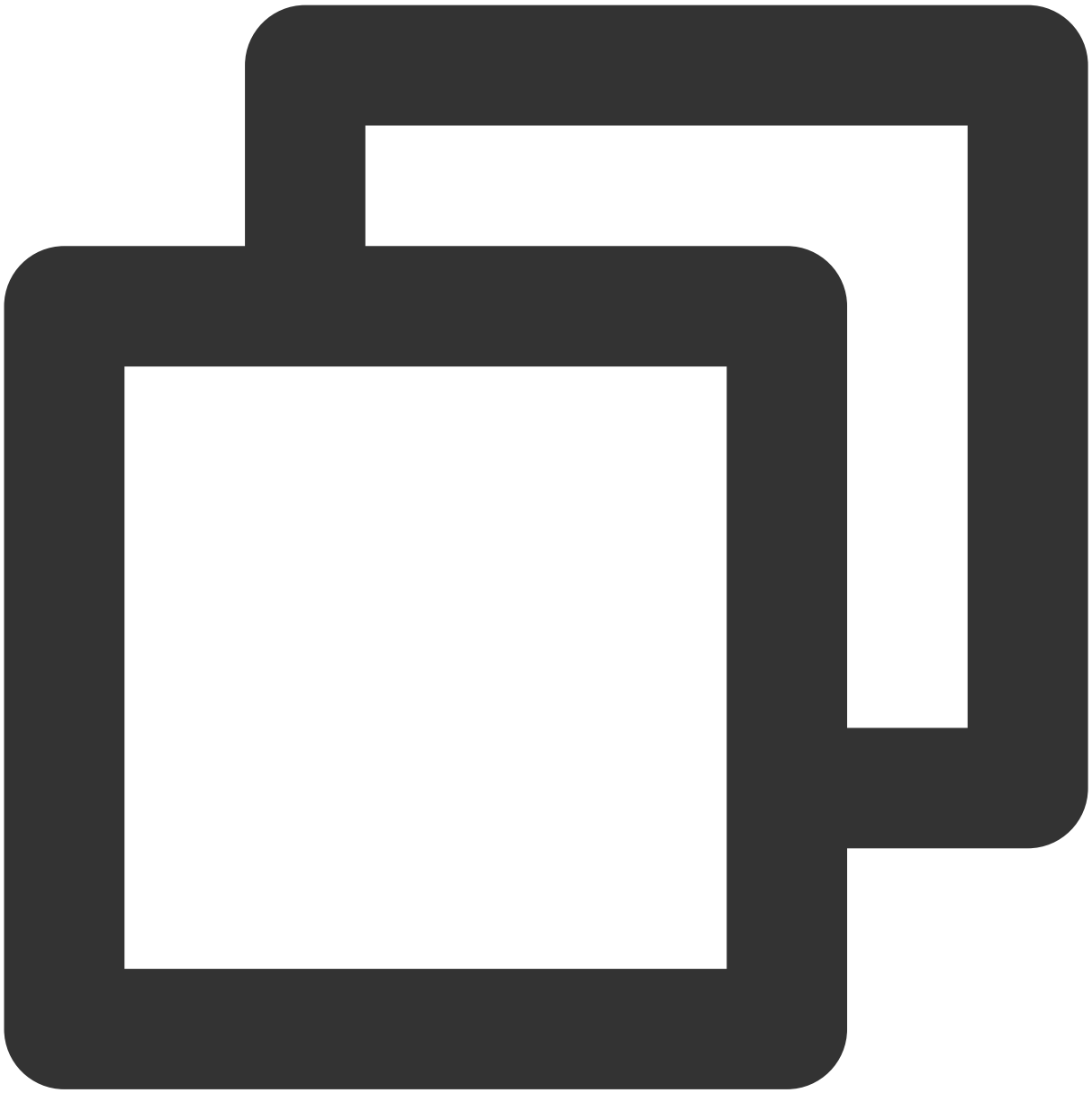
Note: If your project involves live streaming or conferencing, you can also use this API to implement the “in a meeting” or “on air” feature.



```
Future<TUIResult> ignore()
```

hangup

This API is used to end a call.

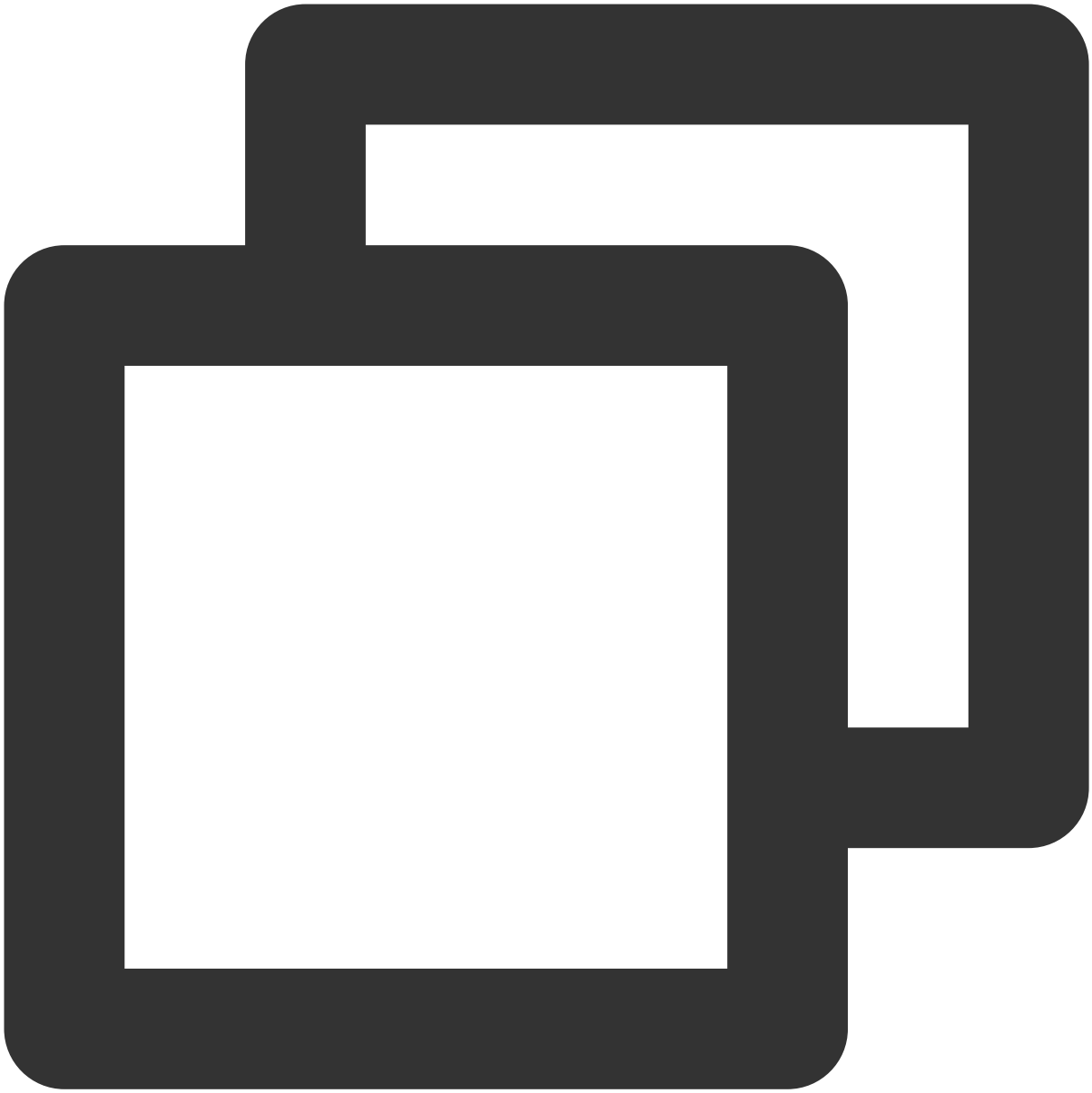


```
Future<TUIResult> hangup()
```

inviteUser

This API is used to invite users to the current group call.

This API is called by a participant of a group call to invite new users.



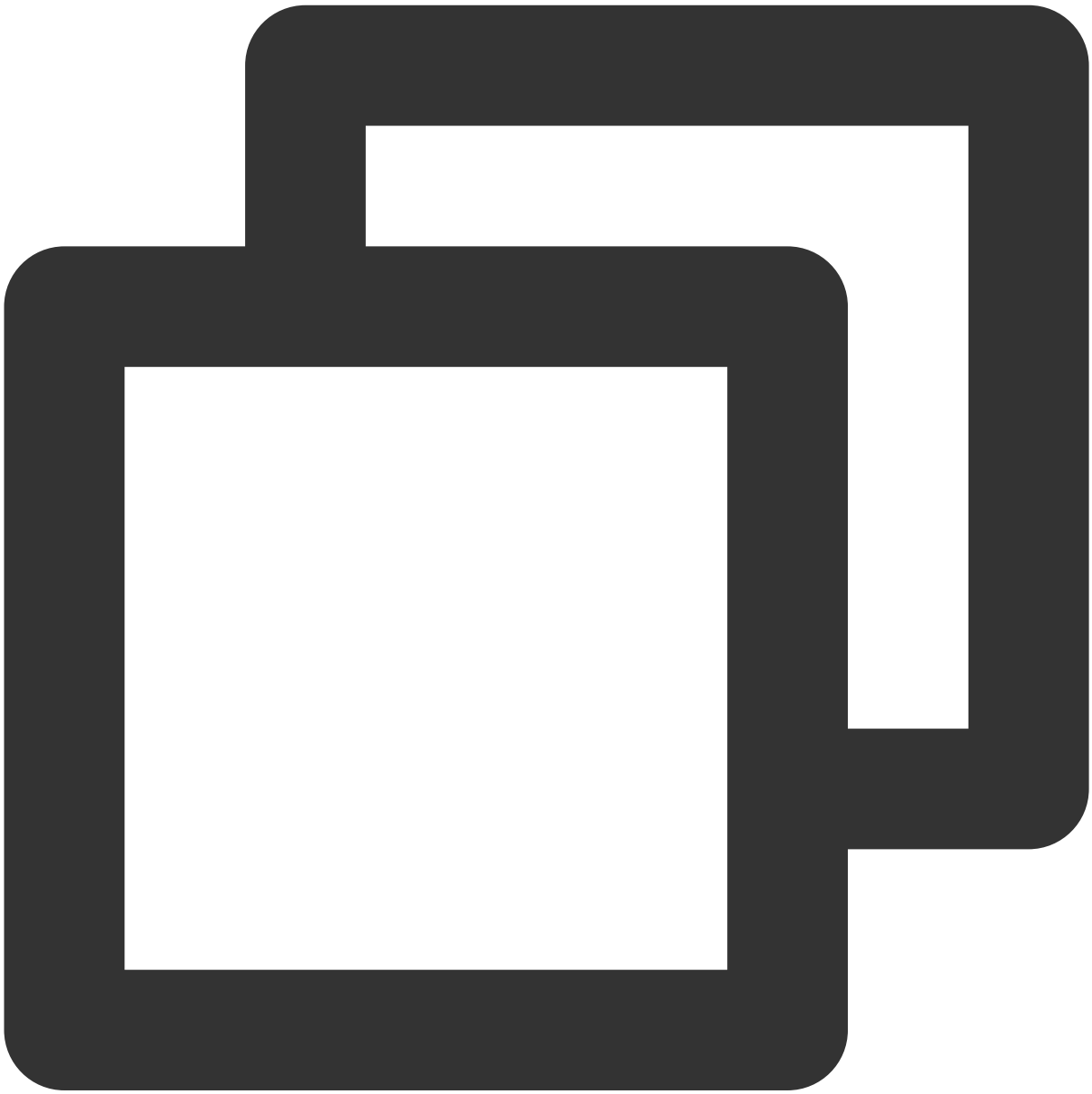
```
Future<void> iniviteUser(List<String> userIdList, TUICallParams params, TUIValueCal
```

Parameter	Type	Description
userIdList	List<String>	The target user IDs.
params	TUICallParams	An additional parameter. such as roomID, call timeout, offline push info, etc.

joinInGroupCall

This API is used to join a group call.

This API is called by a group member to join the group's call.



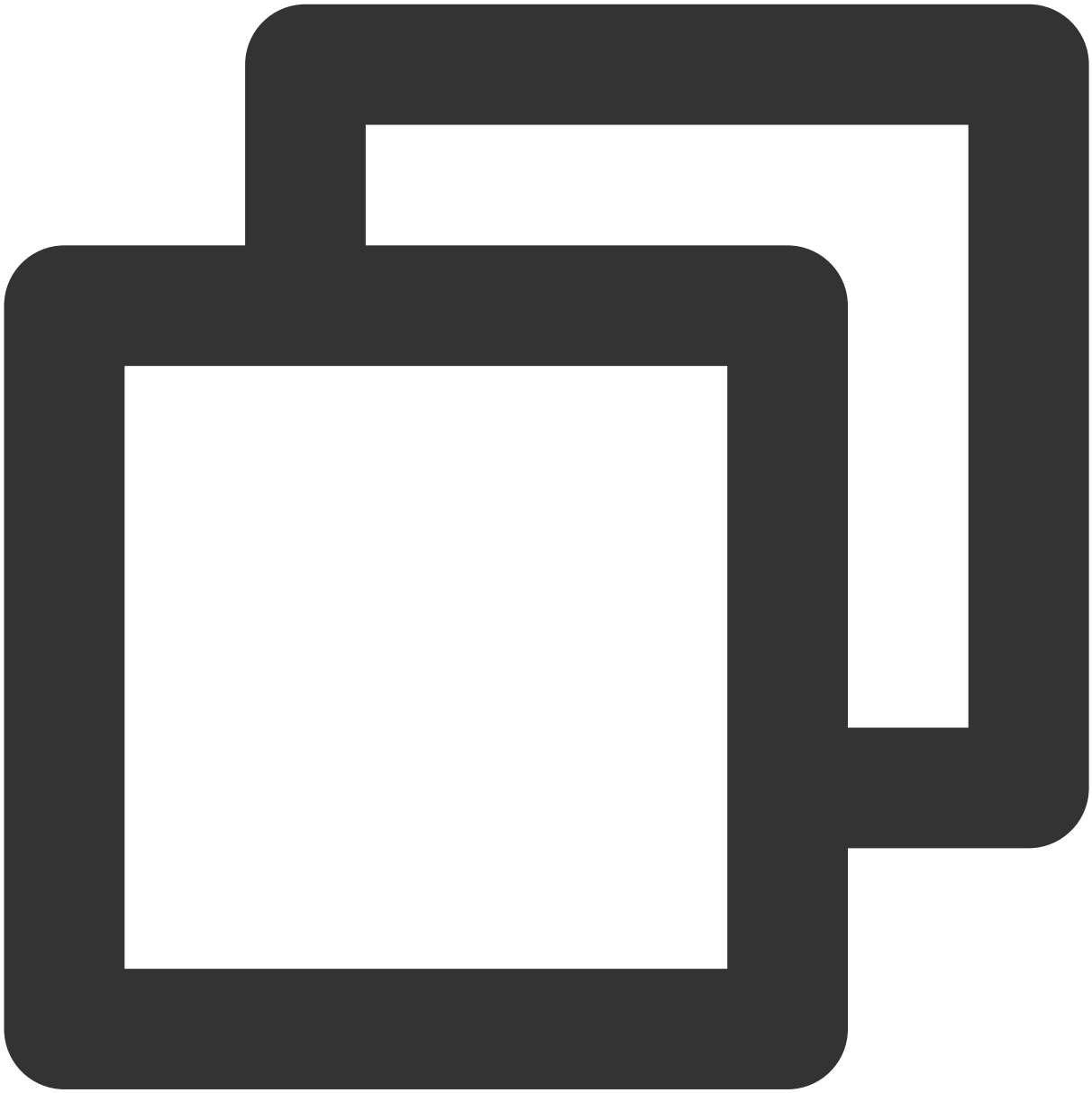
```
Future<TUIResult> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaTyp
```

Parameter	Type	Description
roomId	TUIRoomId	The room ID.
groupId	String	The group ID.

mediaType	TUICallMediaType	The call type, which can be video or audio.

switchCallMediaType

This API is used to change the call type.

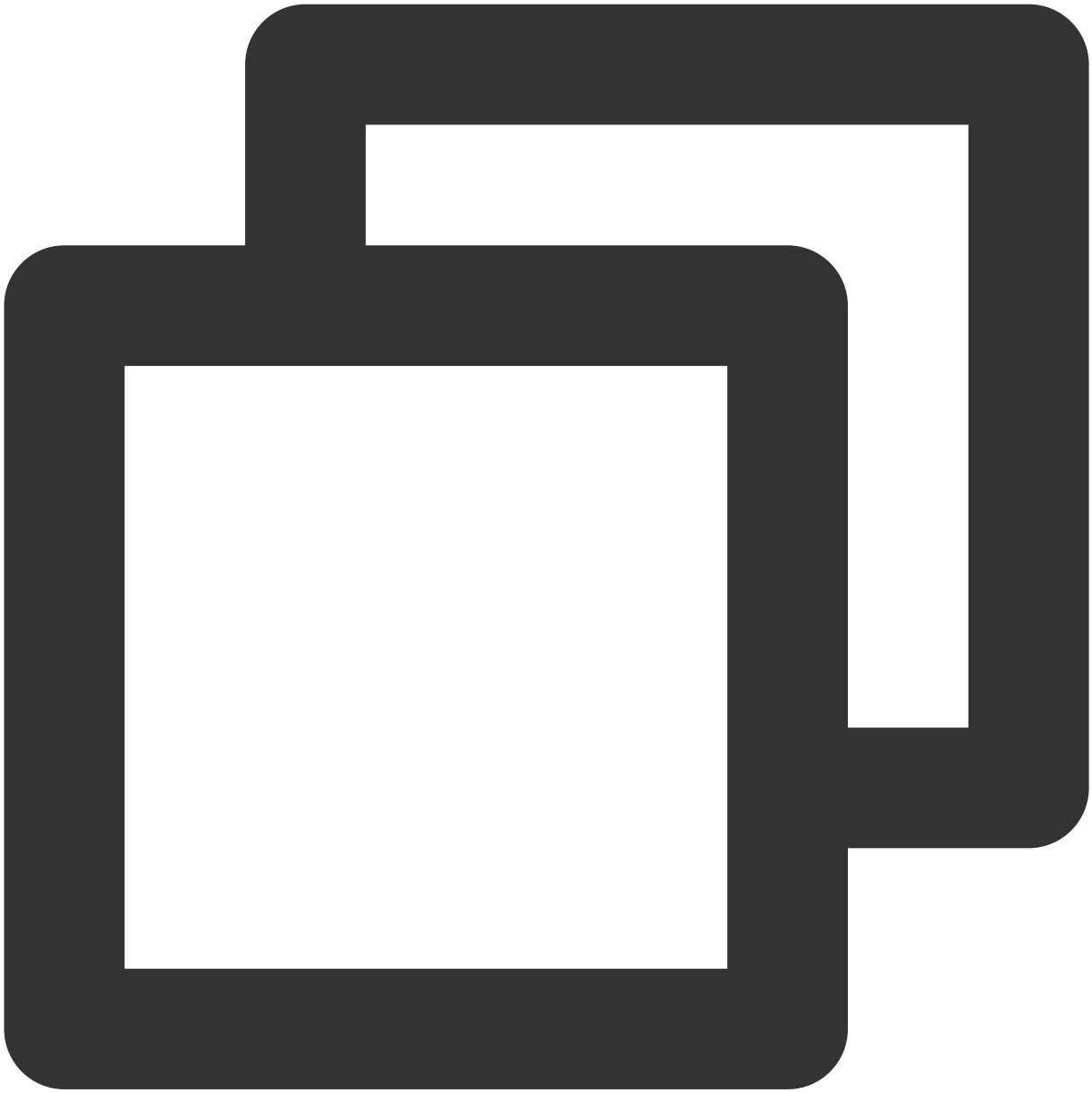


```
Future<void> switchCallMediaType(TUICallMediaType mediaType)
```

Parameter	Type	Description
mediaType	TUICallMediaType	The call type, which can be video or audio.

startRemoteView

This API is used to set the view object to display a remote video.

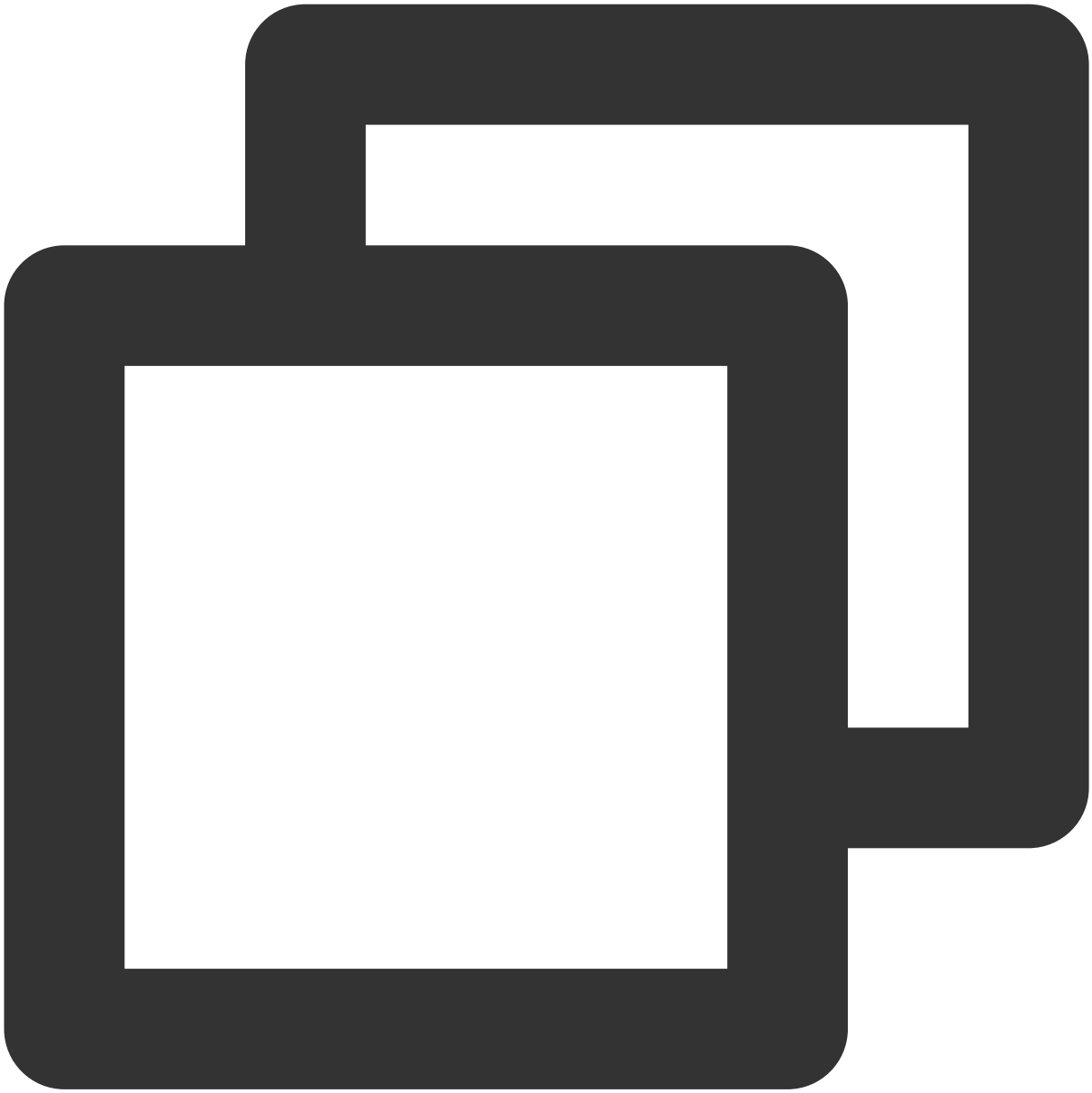


```
Future<void> startRemoteView(String userId, intviewId)
```

Parameter	Type	Description
userId	String	The target user ID.
intviewId	int	The ID of the widget in the video rendering screen

stopRemoteview

This API is used to unsubscribe from the video stream of a remote user.

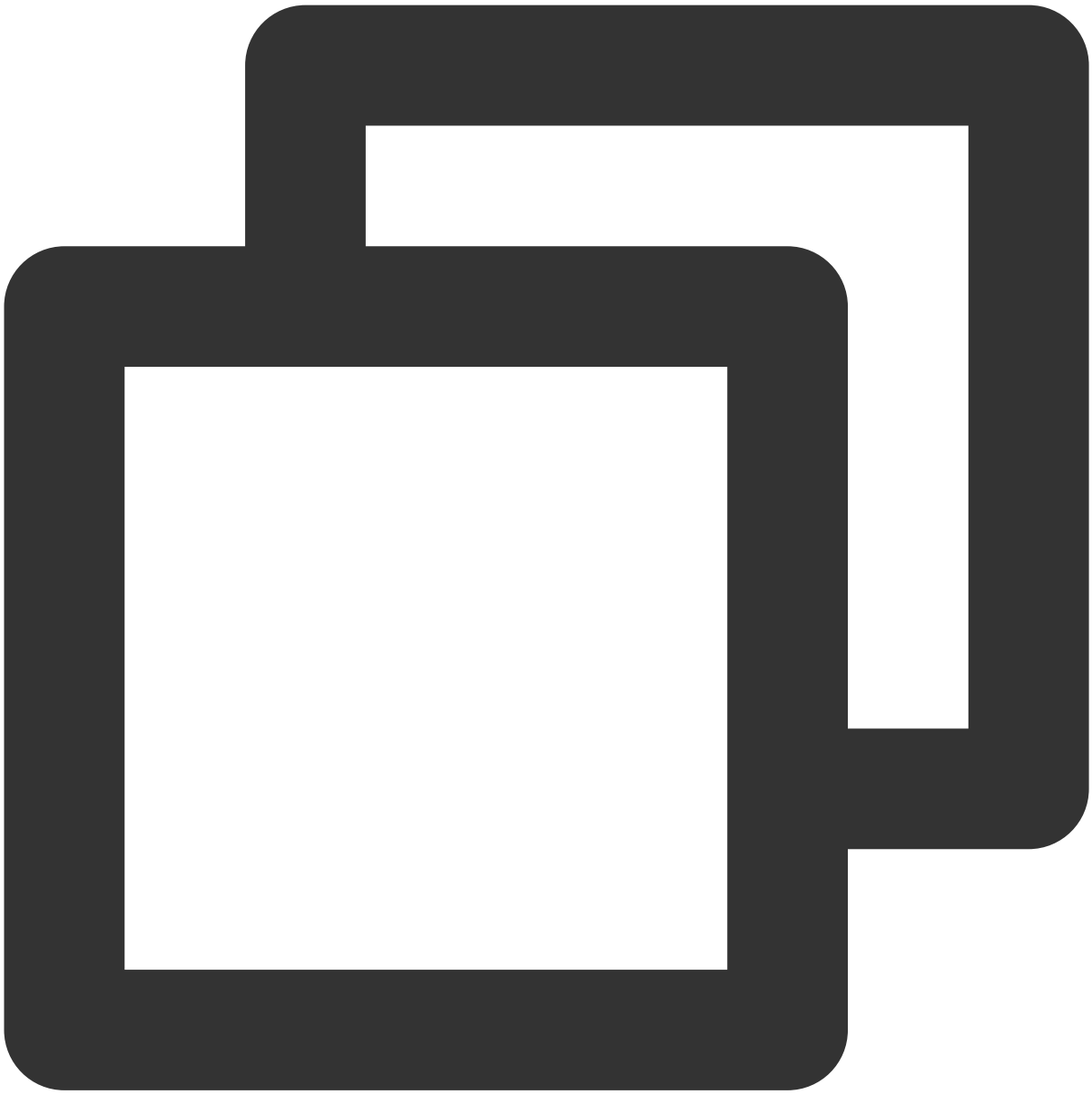


```
Future<void> stopRemoteView(String userId)
```

Parameter	Type	Description
userId	String	The target user ID.

openCamera

This API is used to turn the camera on.

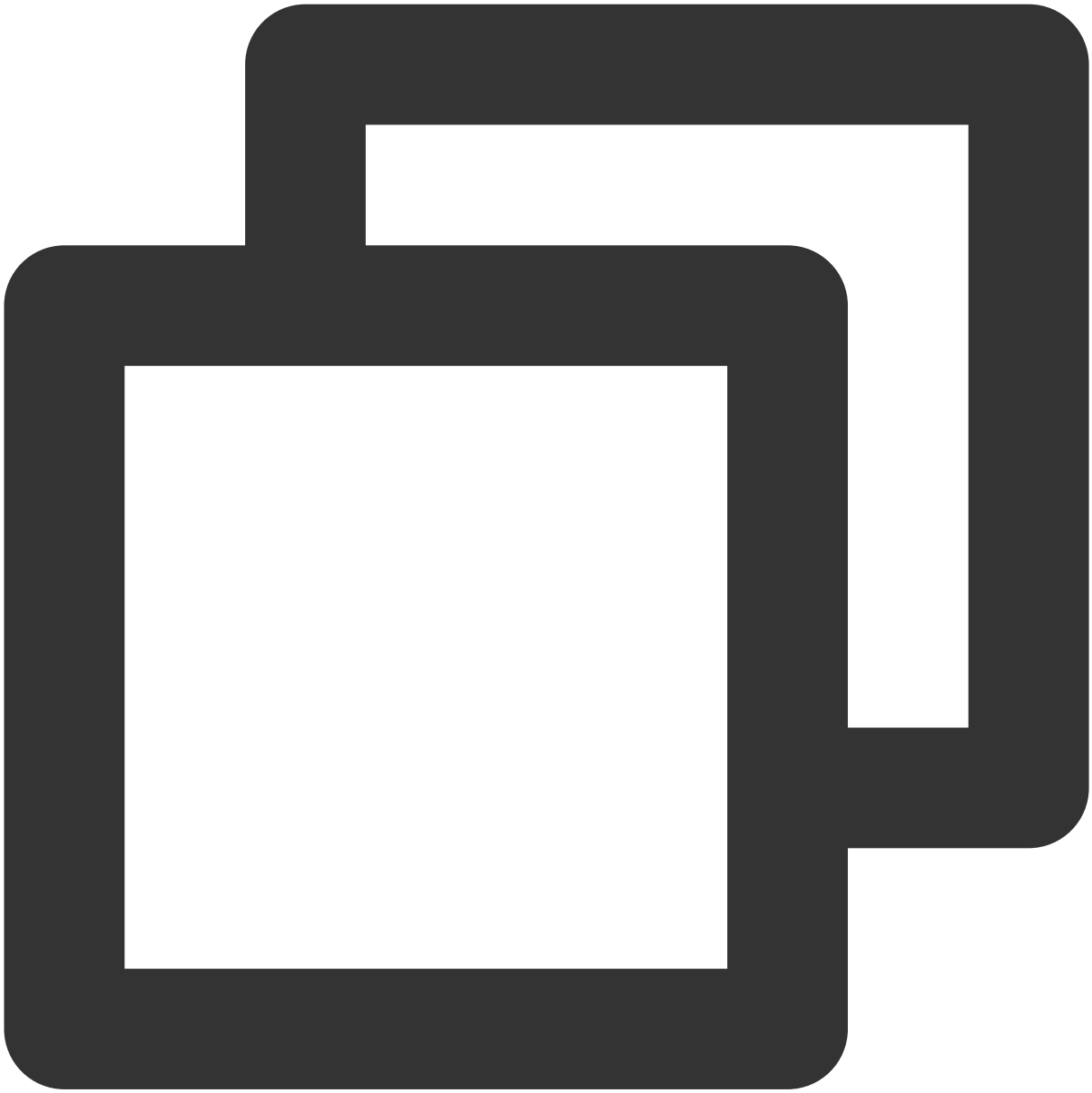


```
Future<TUIResult> openCamera(TUICamera camera, int? viewId)
```

Parameter	Type	Description
camera	TUICamera	The front or rear camera.
viewId	int	The ID of the widget in the video rendering screen

closeCamera

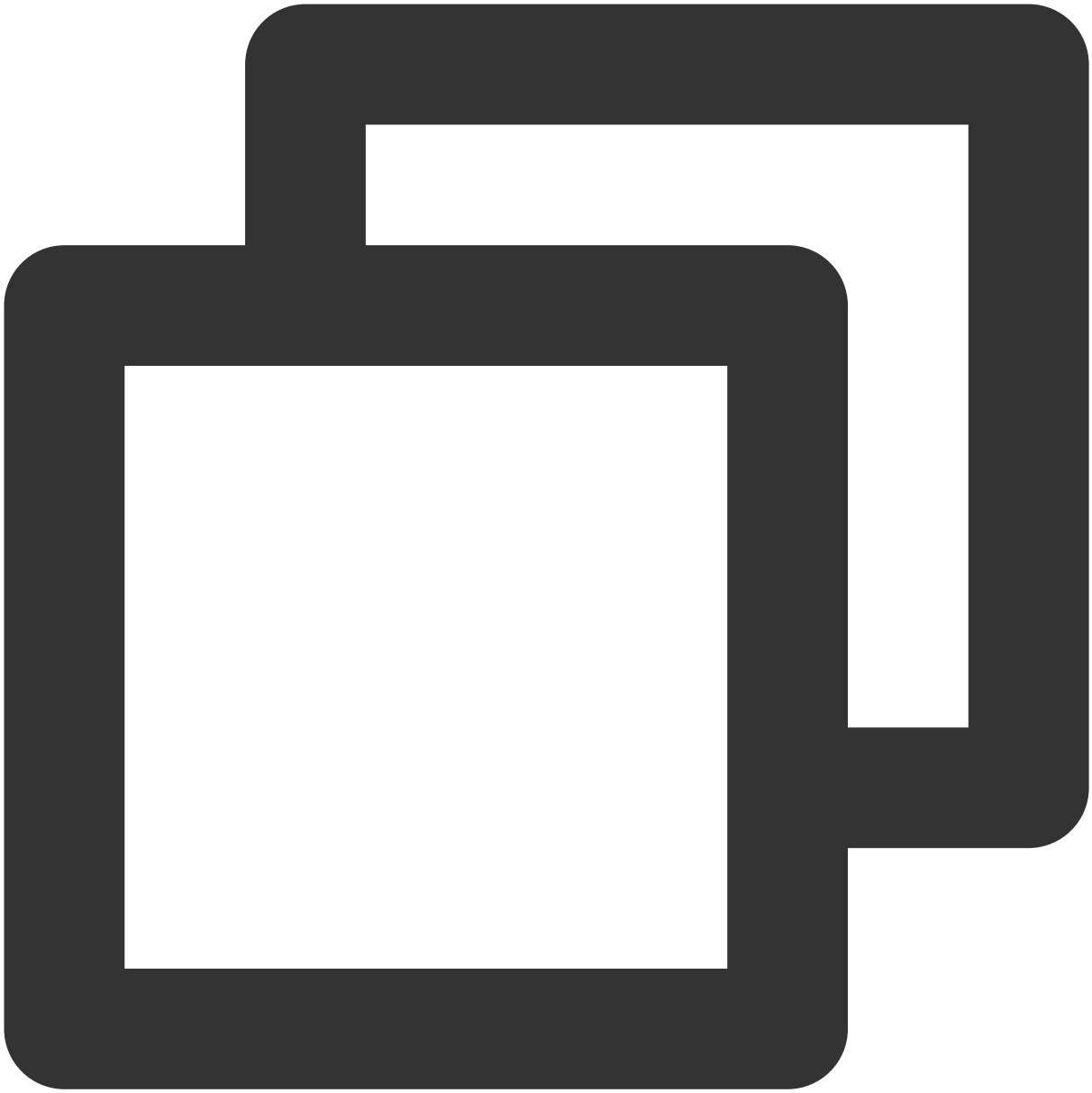
This API is used to turn the camera off.



```
Future<void> closeCamera()
```

switchCamera

This API is used to switch between the front and rear cameras.

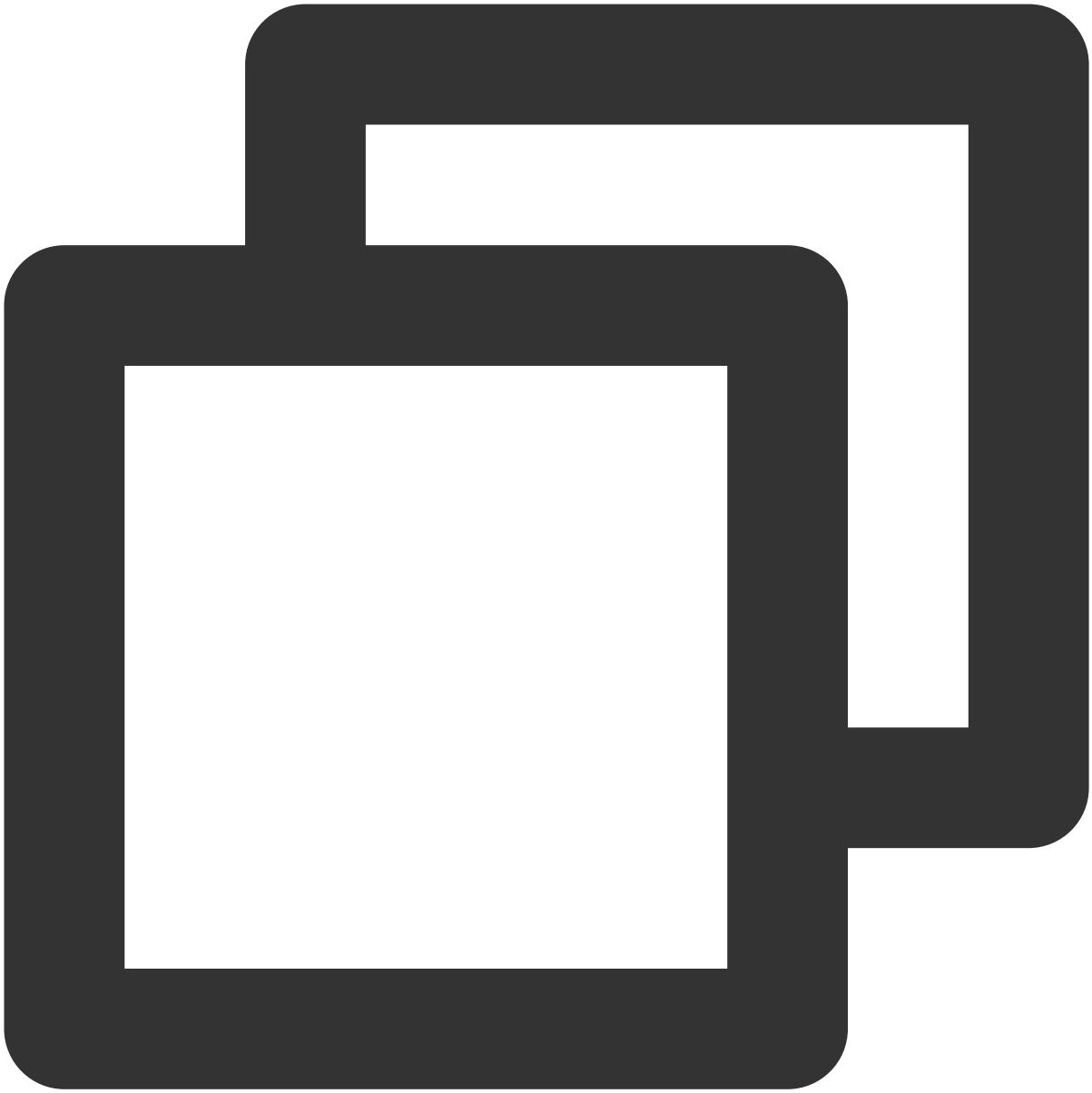


```
Future<void> switchCamera(TUICamera camera)
```

Parameter	Type	Description
camera	TUICamera	The front or rear camera.

openMicrophone

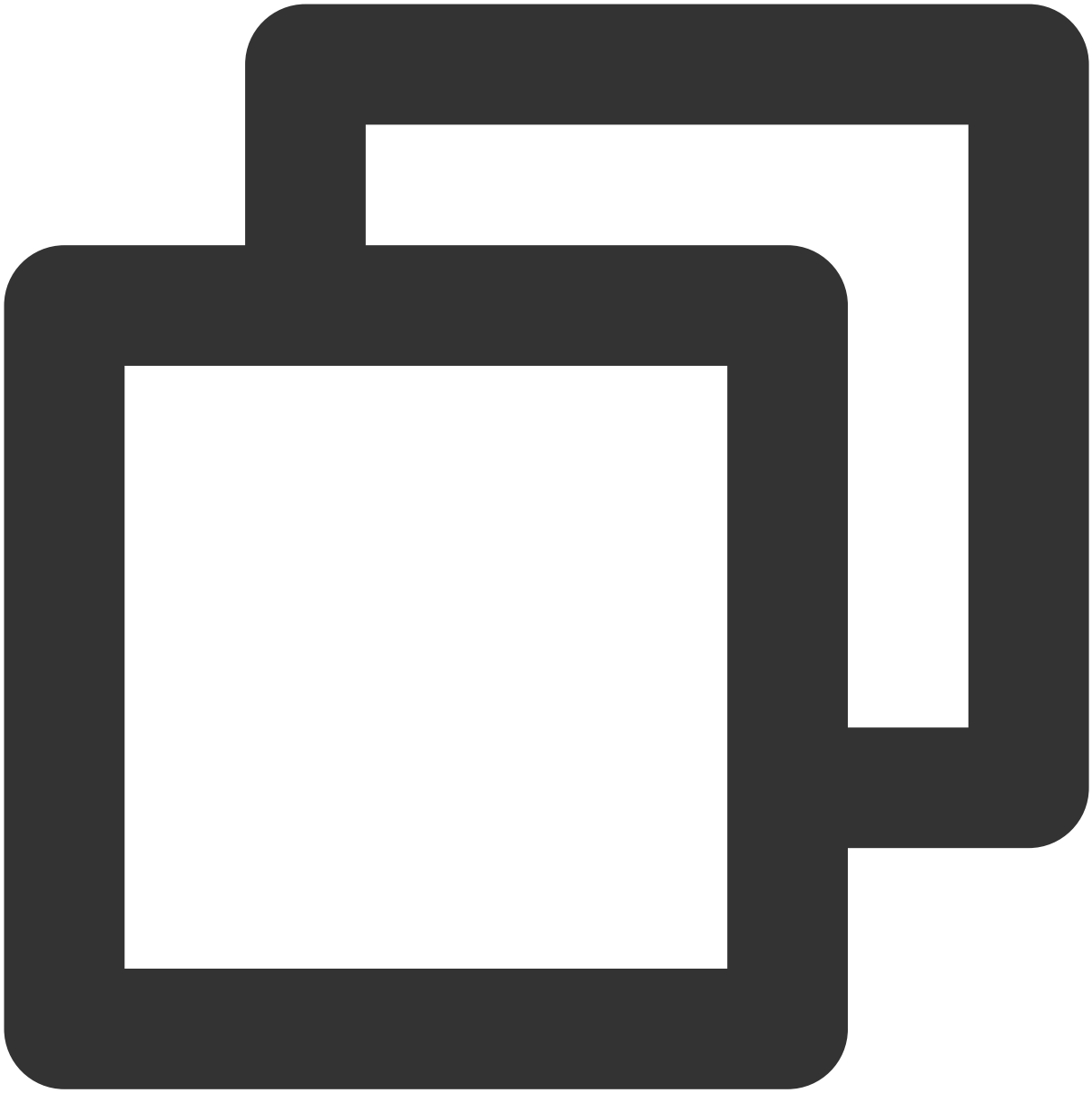
This API is used to turn the mic on.



```
Future<TUIResult> openMicrophone()
```

closeMicrophone

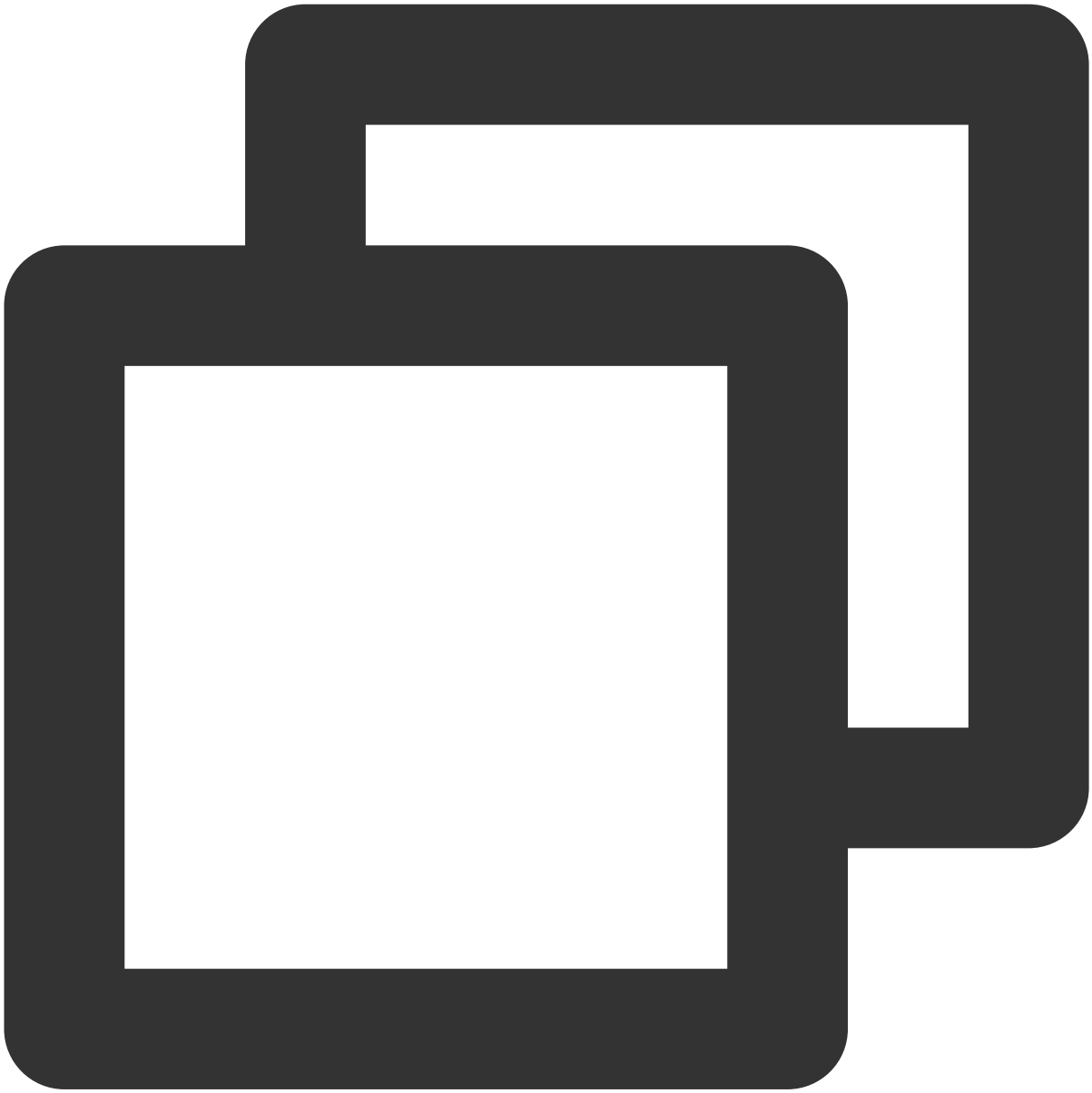
This API is used to turn the mic off.



```
Future<void> closeMicrophone()
```

selectAudioPlaybackDevice

This API is used to select the audio playback device (receiver or speaker). In call scenarios, you can use this API to turn on/off hands-free mode.

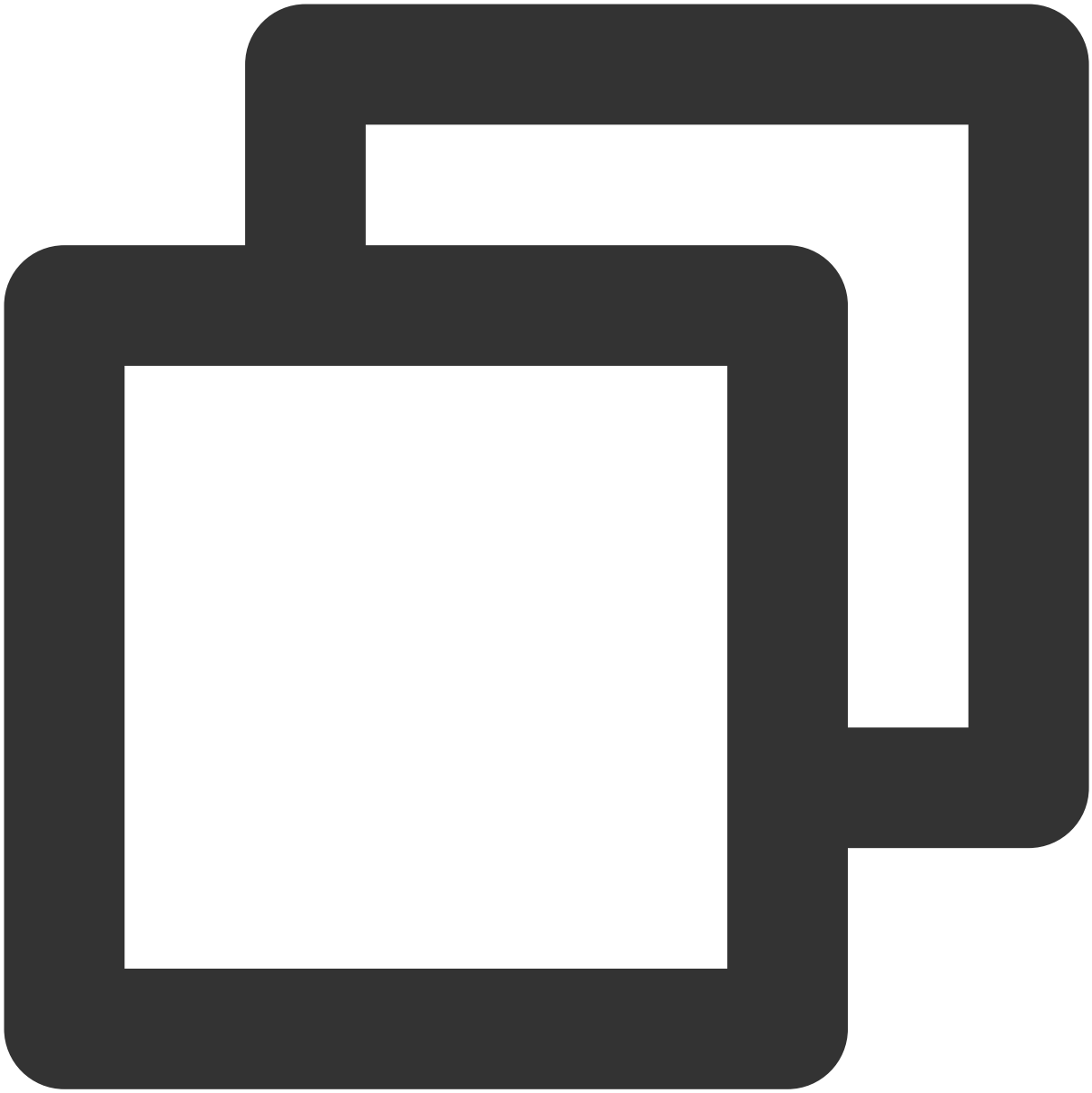


```
Future<void> selectAudioPlaybackDevice(TUIAudioPlaybackDevice device)
```

Parameter	Type	Description
device	TUIAudioPlaybackDevice	The speaker or receiver.

setSelfInfo

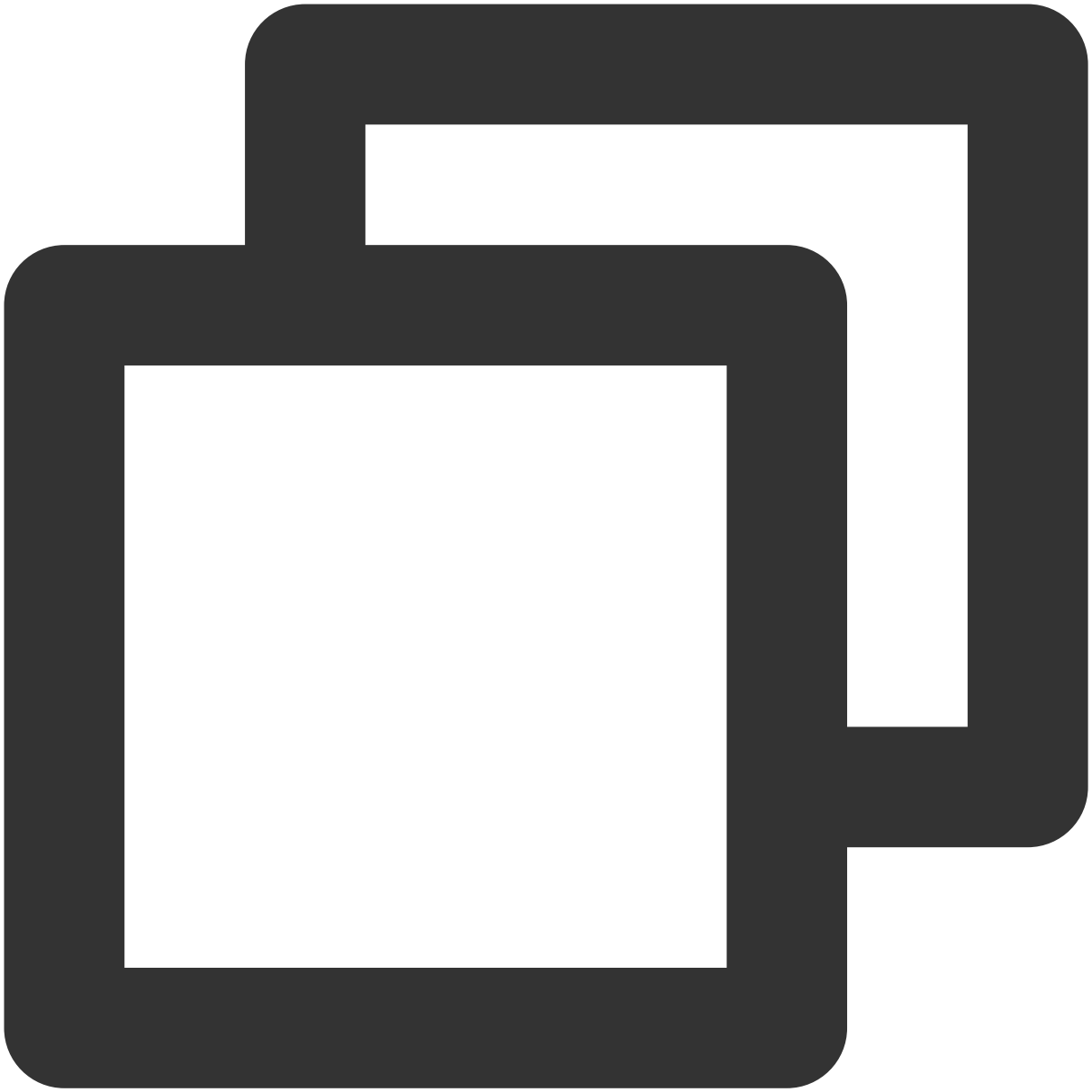
This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.



```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

enableMultiDeviceAbility

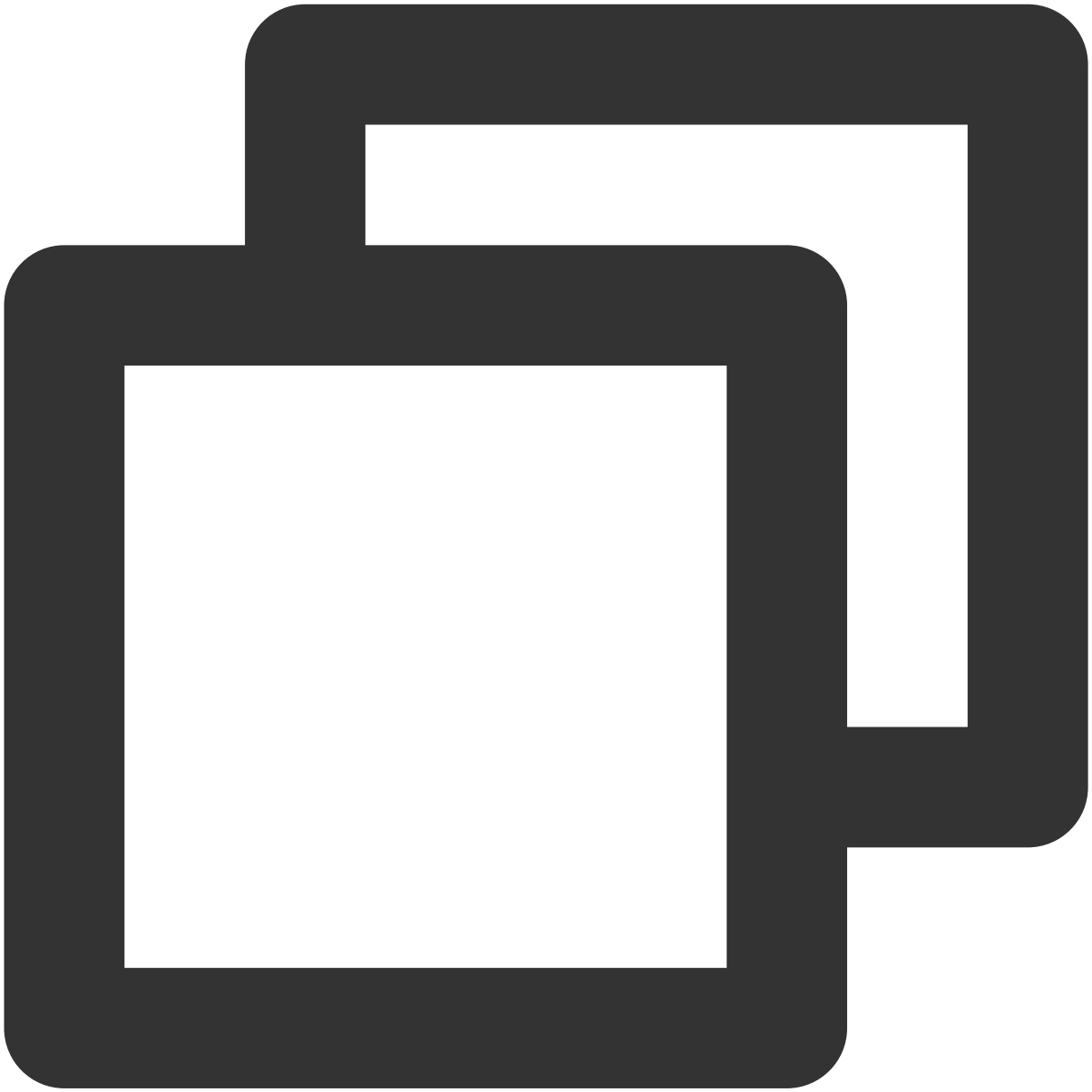
This API is used to set whether to enable multi-device login for `TUICallEngine` (supported by the premium package).



```
Future<TUIResult> enableMultiDeviceAbility(bool enable)
```

setVideoRenderParams

Set the rendering mode of video image.



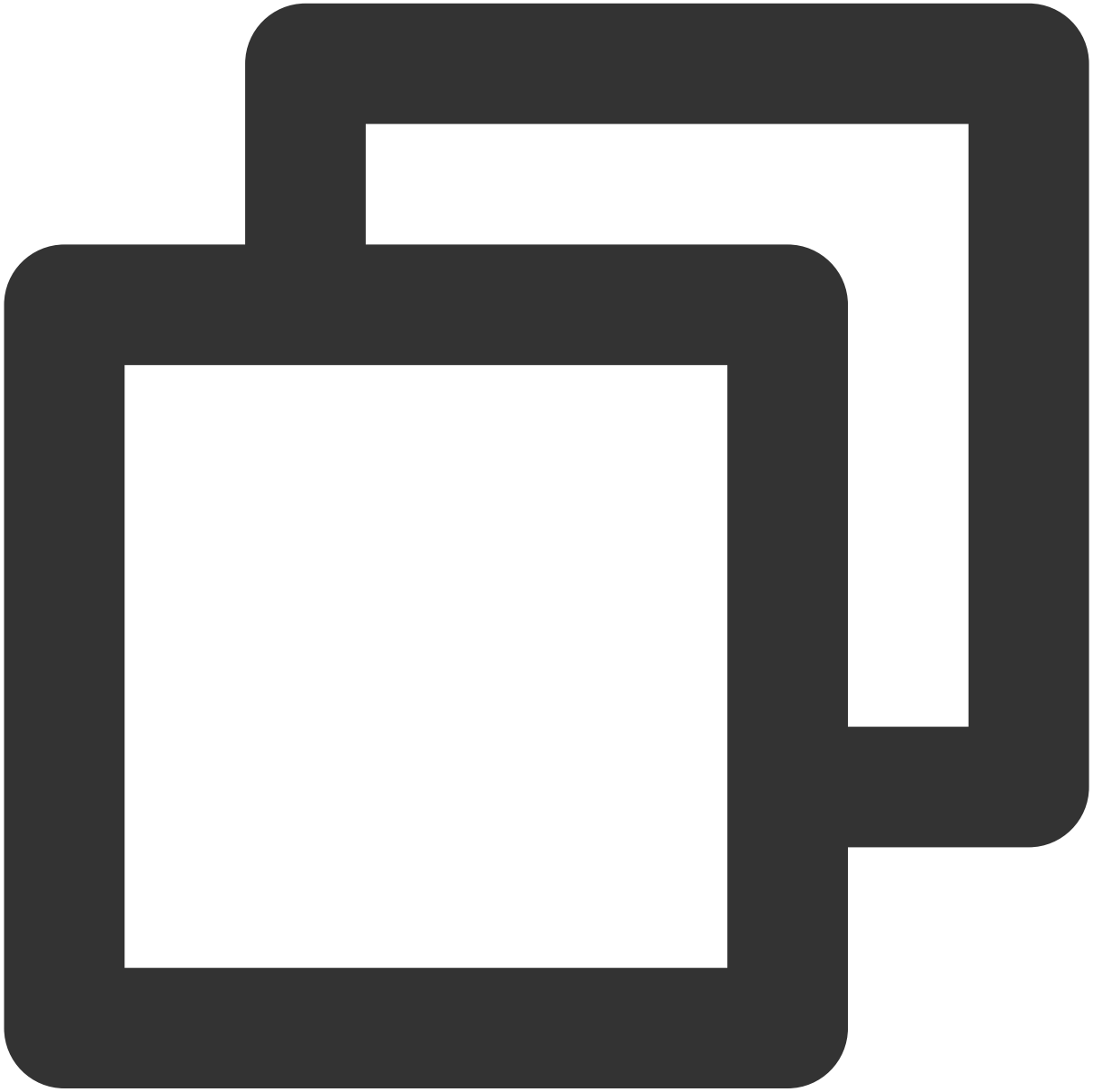
```
Future<TUIResult> setVideoRenderParams(String userId, VideoRenderParams params)
```

Parameter	Type	Description
userId	String	The target user ID.
params	VideoRenderParams	Video render parameters.

setVideoEncoderParams

Set the encoding parameters of video encoder.

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.

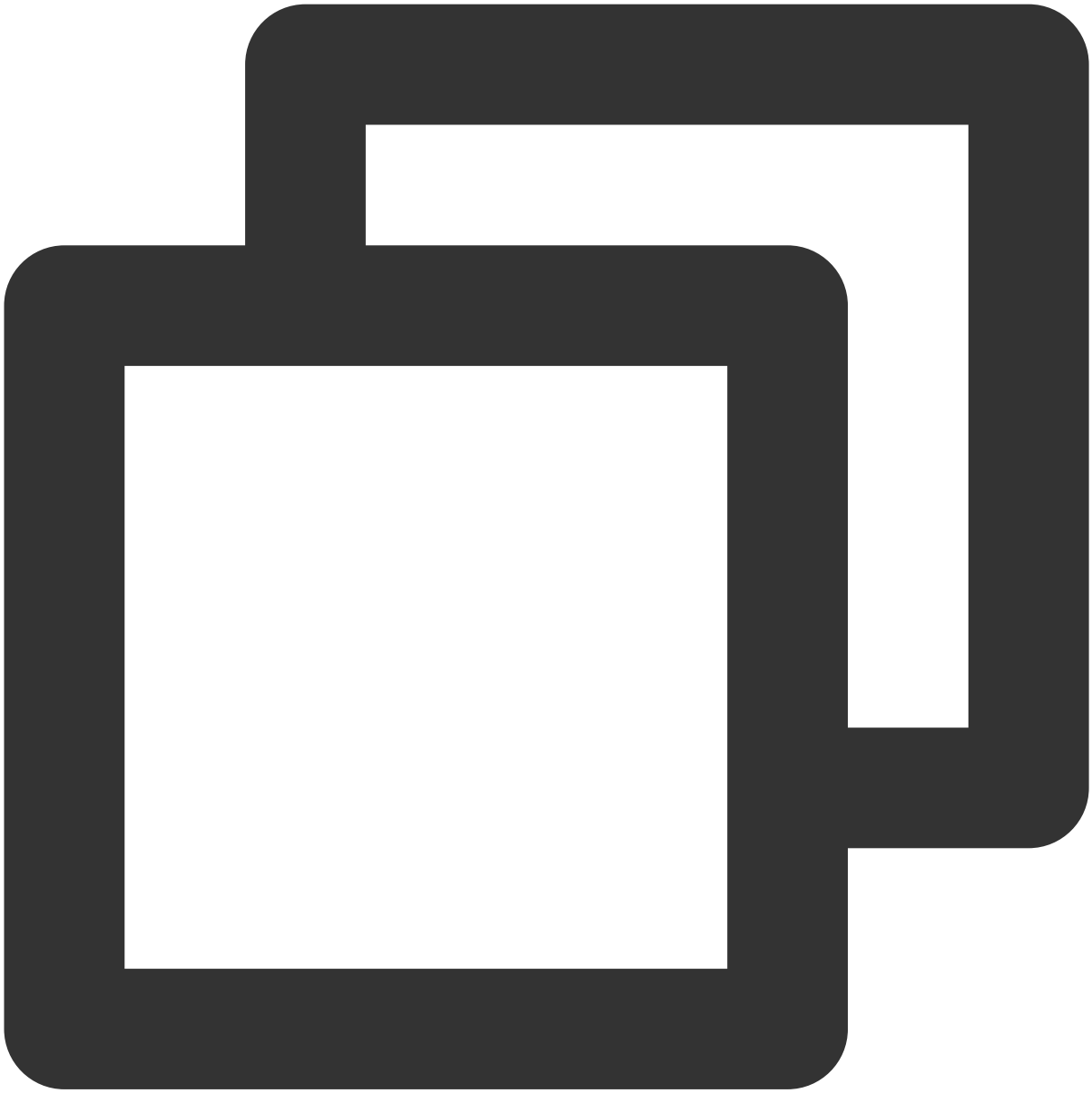


```
Future<TUIResult> setVideoEncoderParams(VideoEncoderParams params)
```

Parameter	Type	Description
params	VideoEncoderParams	Video encoding parameters

queryRecentCalls

Query call record.

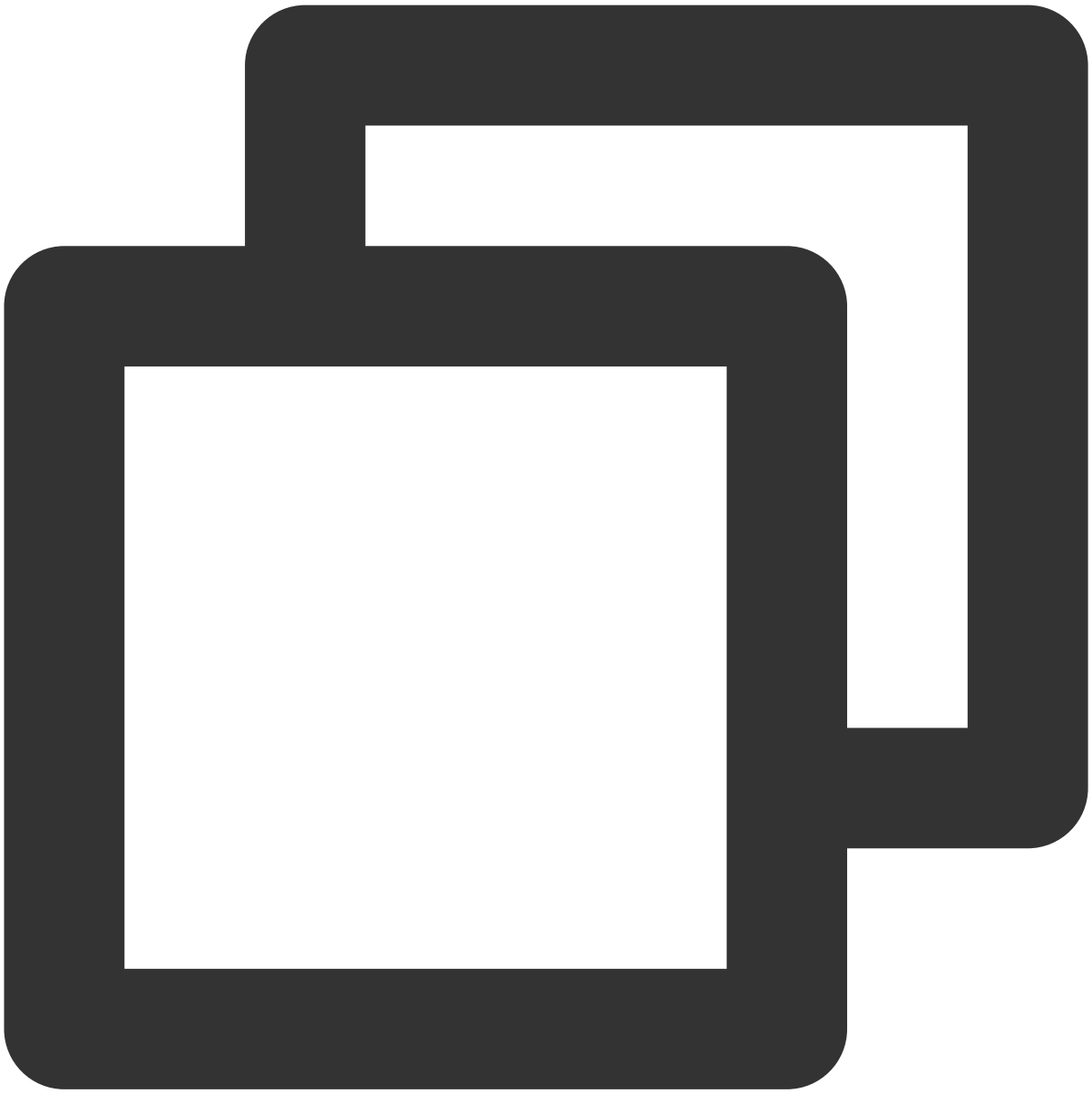


```
Future<void> queryRecentCalls(TUICallRecentCallsFilter filter, TUIValueCallback cal
```

Parameter	Type	Description
filter	TUICallRecentCallsFilter	Filter condition

deleteRecordCalls

Delete call record.

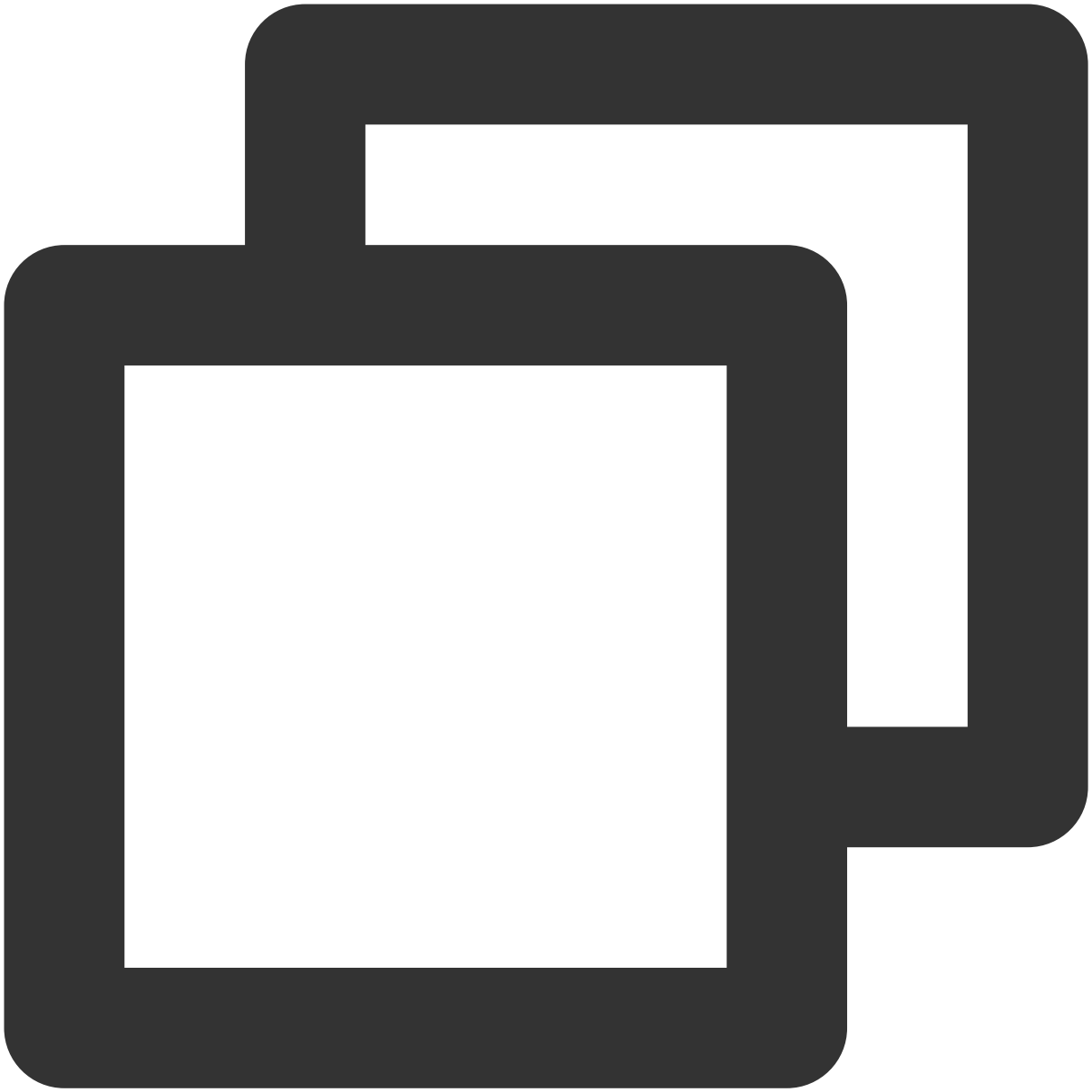


```
Future<void> deleteRecordCalls(List<String> callIdList, TUIValueCallback callback)
```

Parameter	Type	Description
callIdList	List<String>	List of IDs of records to be deleted.

setBeautyLevel

Set beauty level, support turning off default beauty.

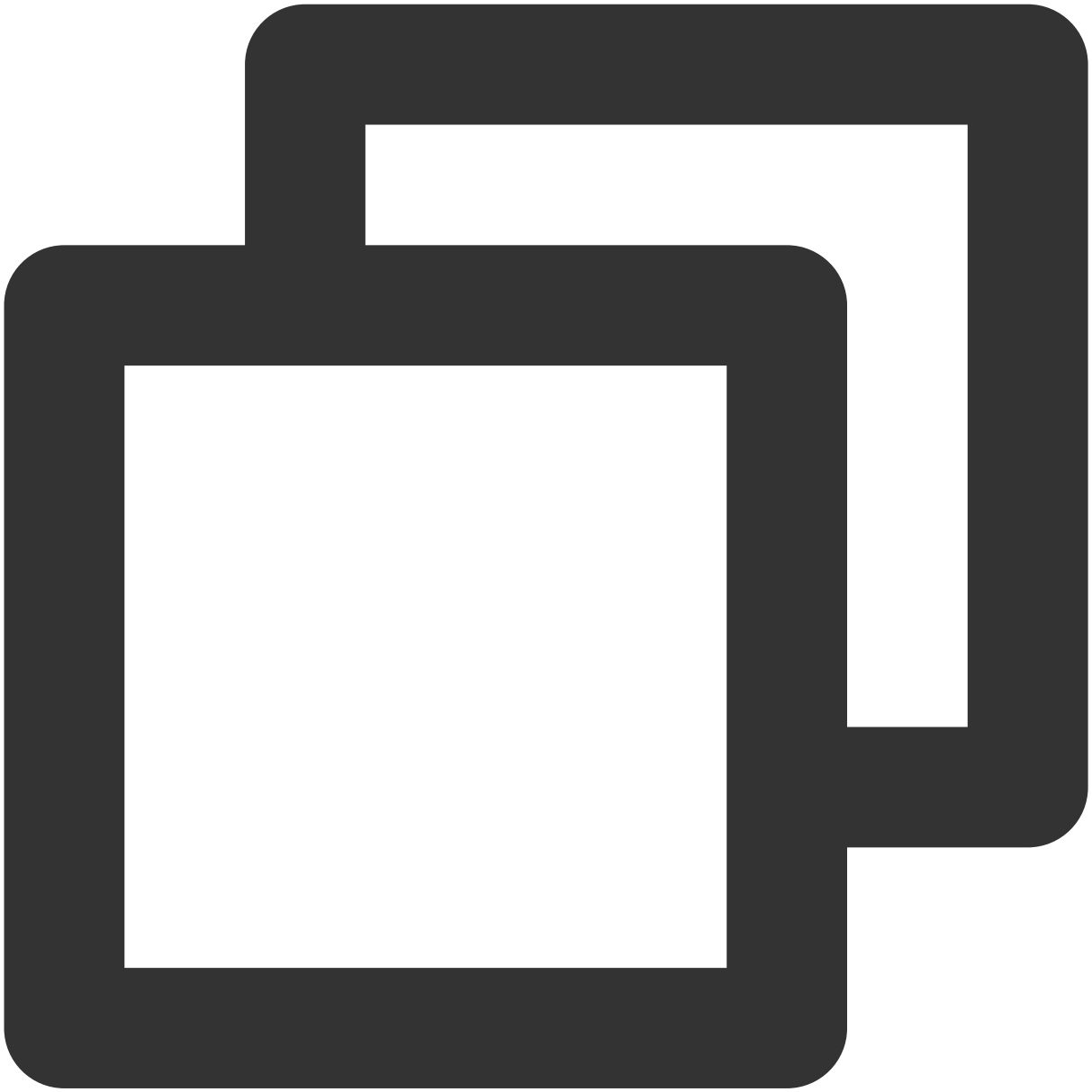


```
Future<TUIResult> setBeautyLevel(double level)
```

Parameter	Type	Description
level	double	Beauty level, range 0.0 to 9.0.

setBlurBackground

Setting Blurry Video Effect.



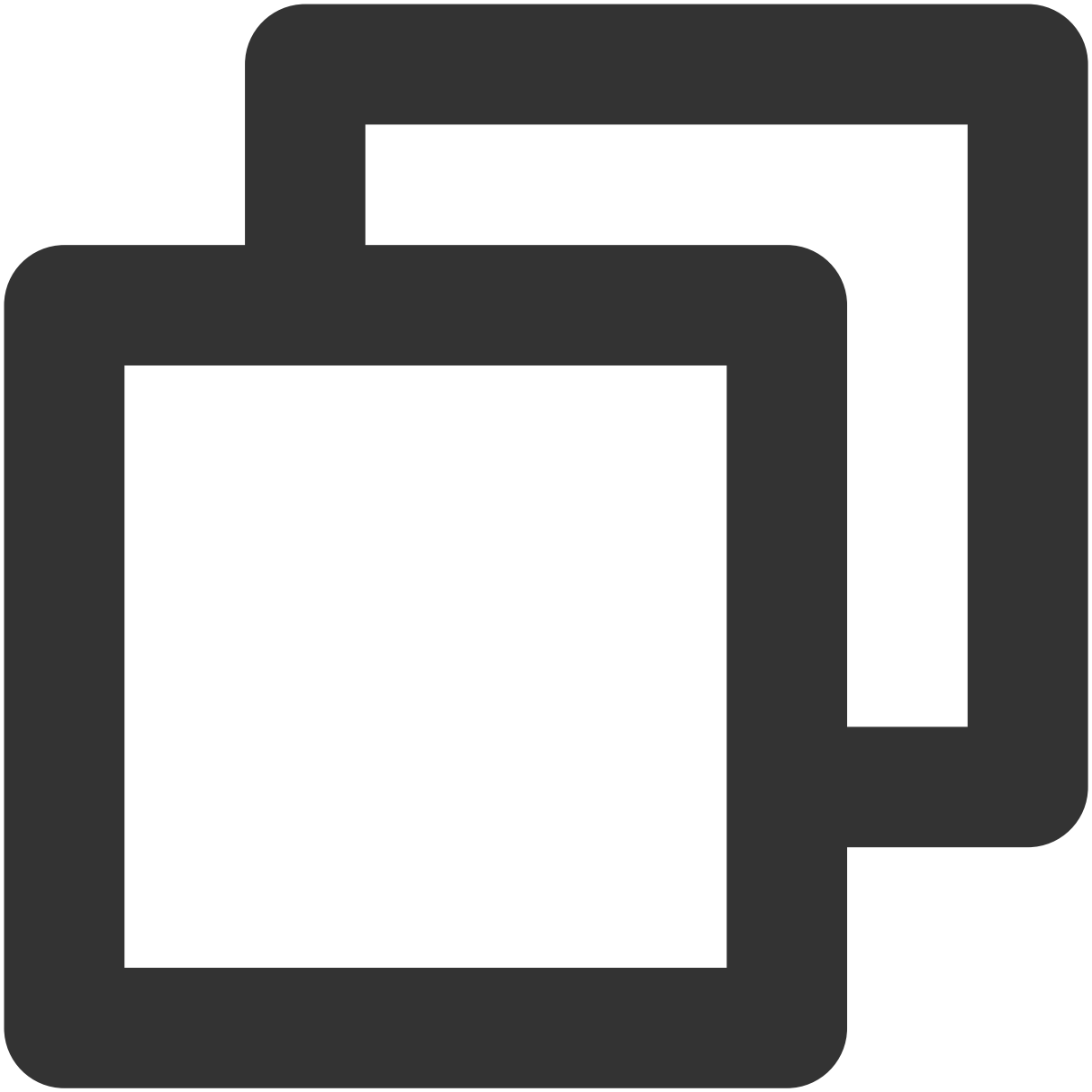
```
void setBlurBackground(int level, Function(int code, String message)? errorCallback
```

The parameters are described below:

Parameter	Type	Meaning
level	int	0: Off, 1: Low, 2: Medium, 3: High.

setVirtualBackground

Setting Virtual Background Image.



```
void setVirtualBackground(String imagePath, Function(int code, String message)? err
```

Parameter	Type	Meaning
imagePath	String	Image Filename. The file needs to be added to the assets of the main project.

TUICallObserver

Last updated：2024-01-23 13:01:17

TUICallObserver API

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

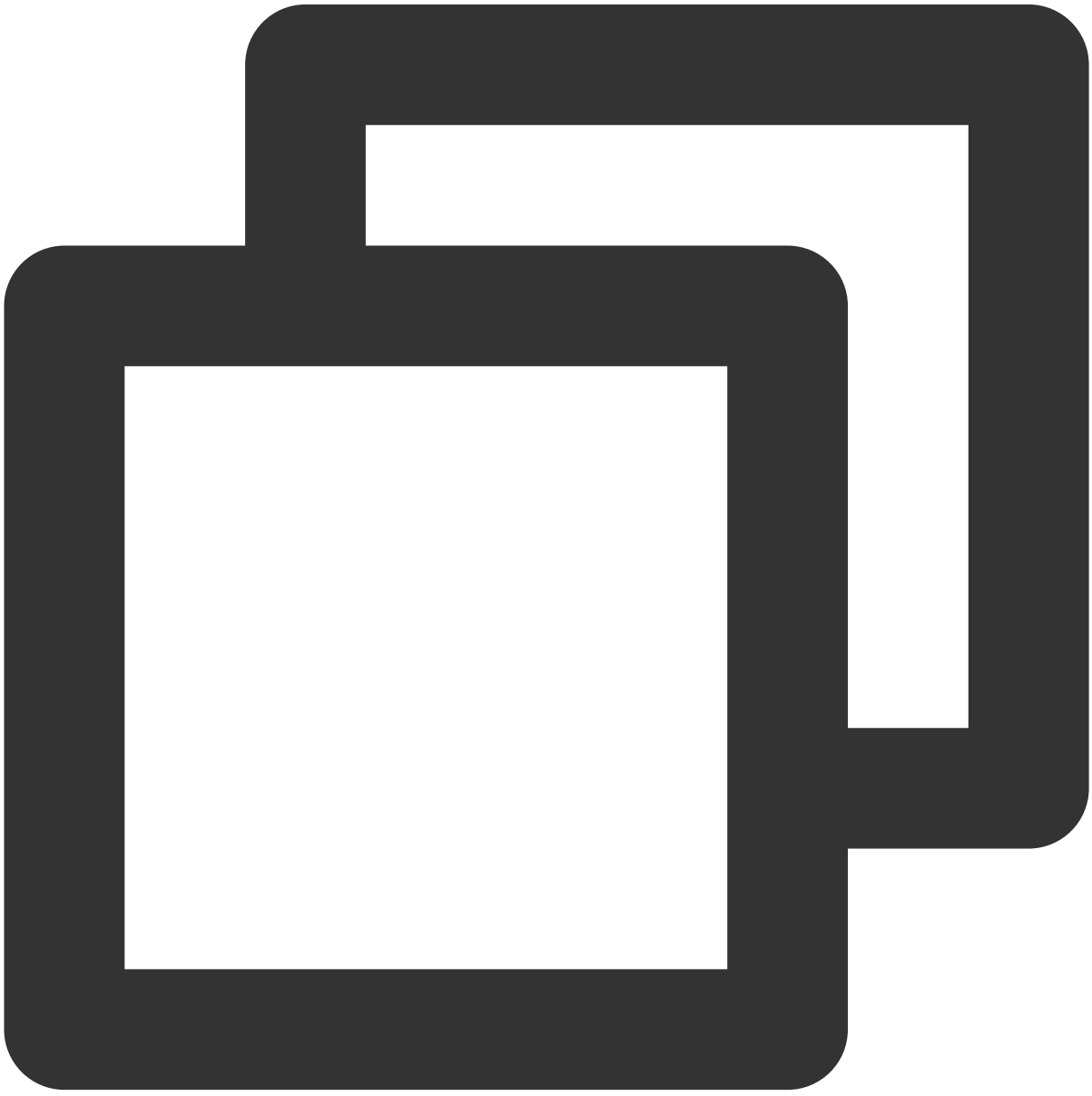
Overview

API	Description
onError	A call occurred during the call.
onCallReceived	A call invitation was received.
onCallCancelled	The call was canceled.
onCallBegin	The call was connected.
onCallEnd	The call ended.
onCallMediaTypeChanged	The call type changed.
onUserReject	A user declined the call.
onUserNoResponse	A user didn't respond.
onUserLineBusy	A user was busy.
onUserJoin	A user joined the call.
onUserLeave	A user left the call.
onUserVideoAvailable	Whether a user had a video stream.
onUserAudioAvailable	Whether a user had an audio stream.
onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.
onKickedOffline	The current user was kicked offline.

onUserSigExpired	The user sig is expired.

Details

Listen to the events thrown by the Flutter plugin through addObserver.



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onError: (int code, String message) {
```

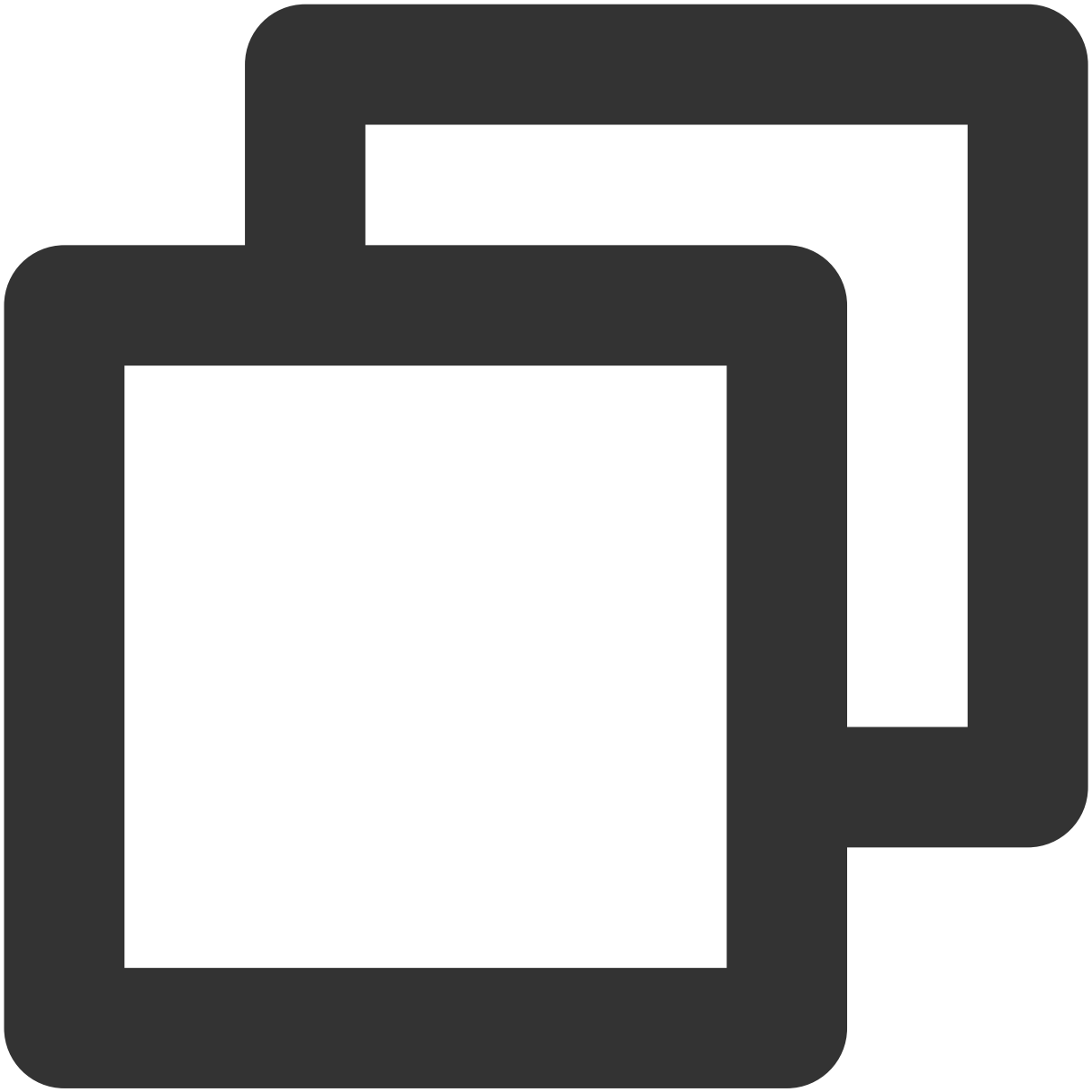
```
    },onCallCancelled: (String callerId) {

    }, onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole

    }, onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole ca

    }, onCallMediaTypeChanged: (TUICallMediaType oldCallMediaType, TUICallMediaType

    }, onUserReject: (String userId) {

    }, onUserNoResponse: (String userId) {

    }, onUserLineBusy: (String onUserLineBusy) {

    }, onUserJoin: (String userId) {

    }, onUserLeave: (String userId) {

    }, onUserVideoAvailable: (String userId, bool isVideoAvailable) {

    }, onUserAudioAvailable: (String userId, bool isAudioAvailable) {

    }, onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList

    }, onCallReceived: (String callerId, List<String> calleeIdList, String groupId,

    }, onUserVoiceVolumeChanged: (Map<String, int> volumeMap) {

    }, onKickedOffline: () {

    }, onUserSigExpired: () {

    }
));
```

onError

An error occurred.

Note

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onError: (int code, String message) {
    }
));
```

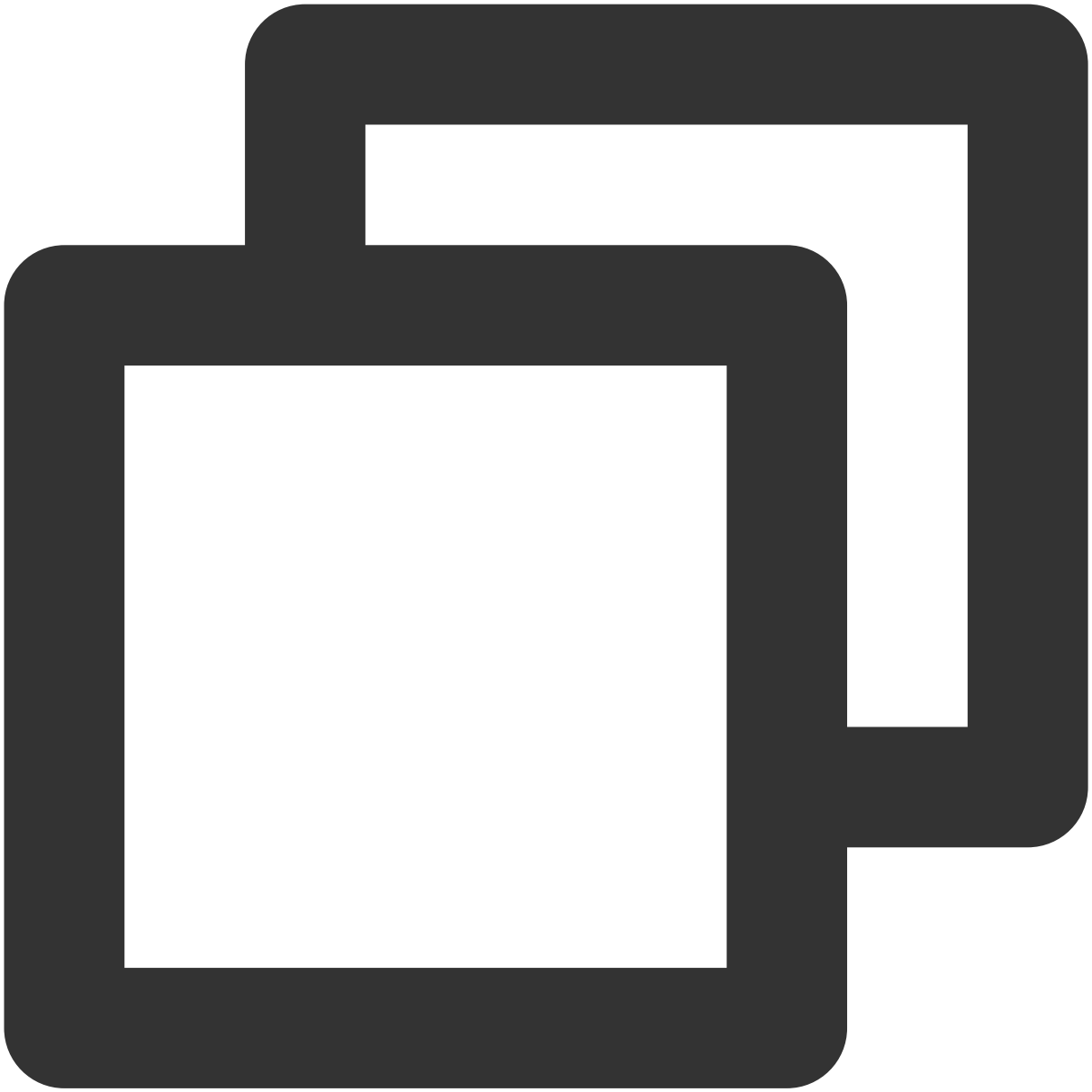
The parameters are described below:

Parameter	Type	Description
code	int	The error code.

message	String	The error message.

onCallReceived

A call invitation was received. This callback is received by an invitee. You can listen for this event to determine whether to display the incoming call view.



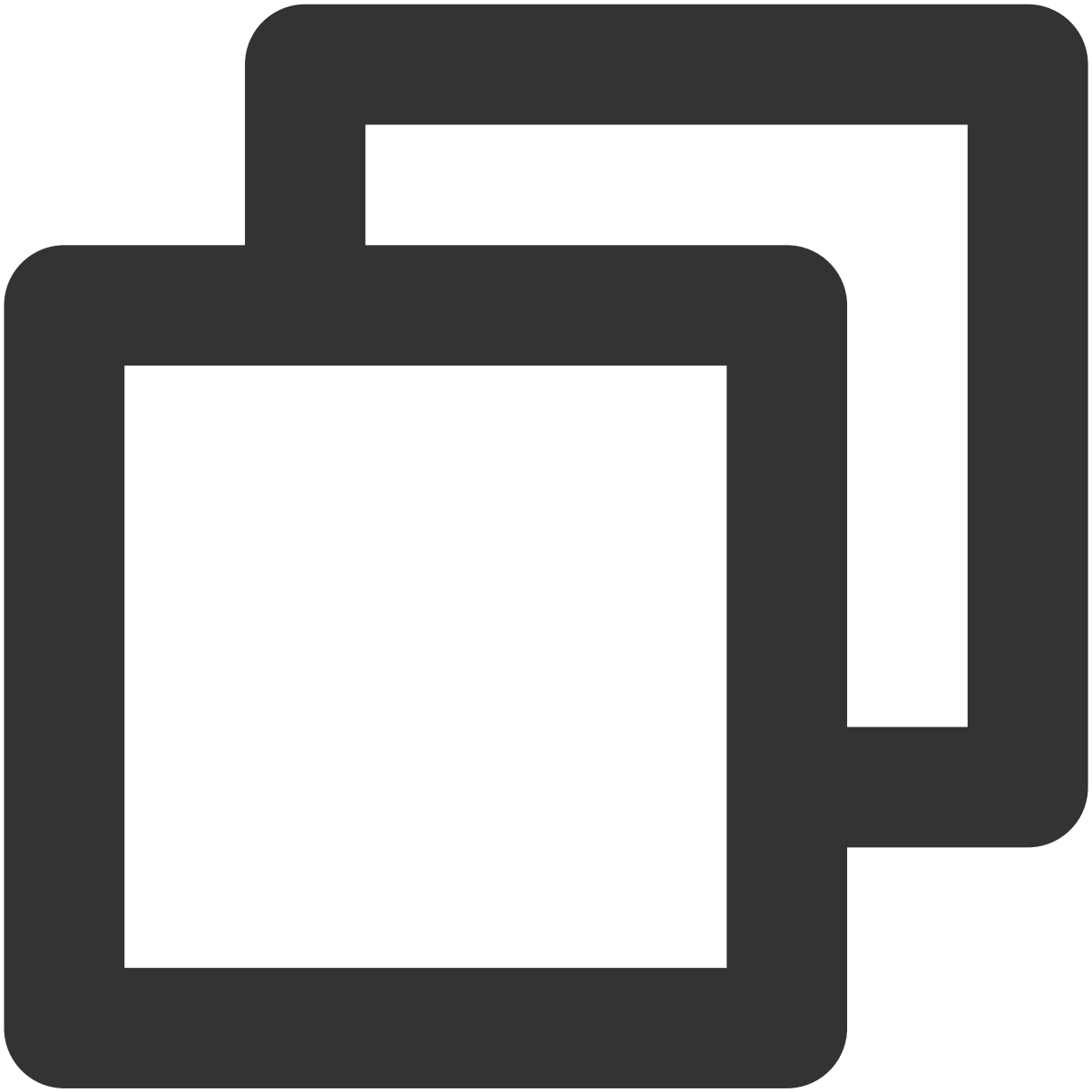
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallReceived: (String callerId, List<String> calleeIdList, String groupId, TU
    }
));
```

The parameters are described below:

Parameter	Type	Description
callerId	String	The user ID of the inviter.
calleeIdList	List<String>	The invitee list.
groupId	String	The group ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.

onCallCancelled

The call was canceled by the inviter or timed out. This callback is received by an invitee. You can listen for this event to determine whether to show a missed call message.
This indicates that the call was canceled by the caller, timed out by the callee, rejected by the callee, or the callee was busy. There are multiple scenarios involved. You can listen to this event to achieve UI logic such as missed calls and resetting UI status.
Call cancellation by the caller: The caller receives the callback (userId is himself); the callee receives the callback (userId is the ID of the caller) .
Callee timeout: the caller will simultaneously receive the onUserNoResponse and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID) .
Callee rejection: The caller will simultaneously receive the onUserReject and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID) .
Callee busy: The caller will simultaneously receive the onUserLineBusy and onCallCancelled callbacks (userId is his own ID);
Abnormal interruption: The callee failed to receive the call ， he receives this callback (userId is his own ID).



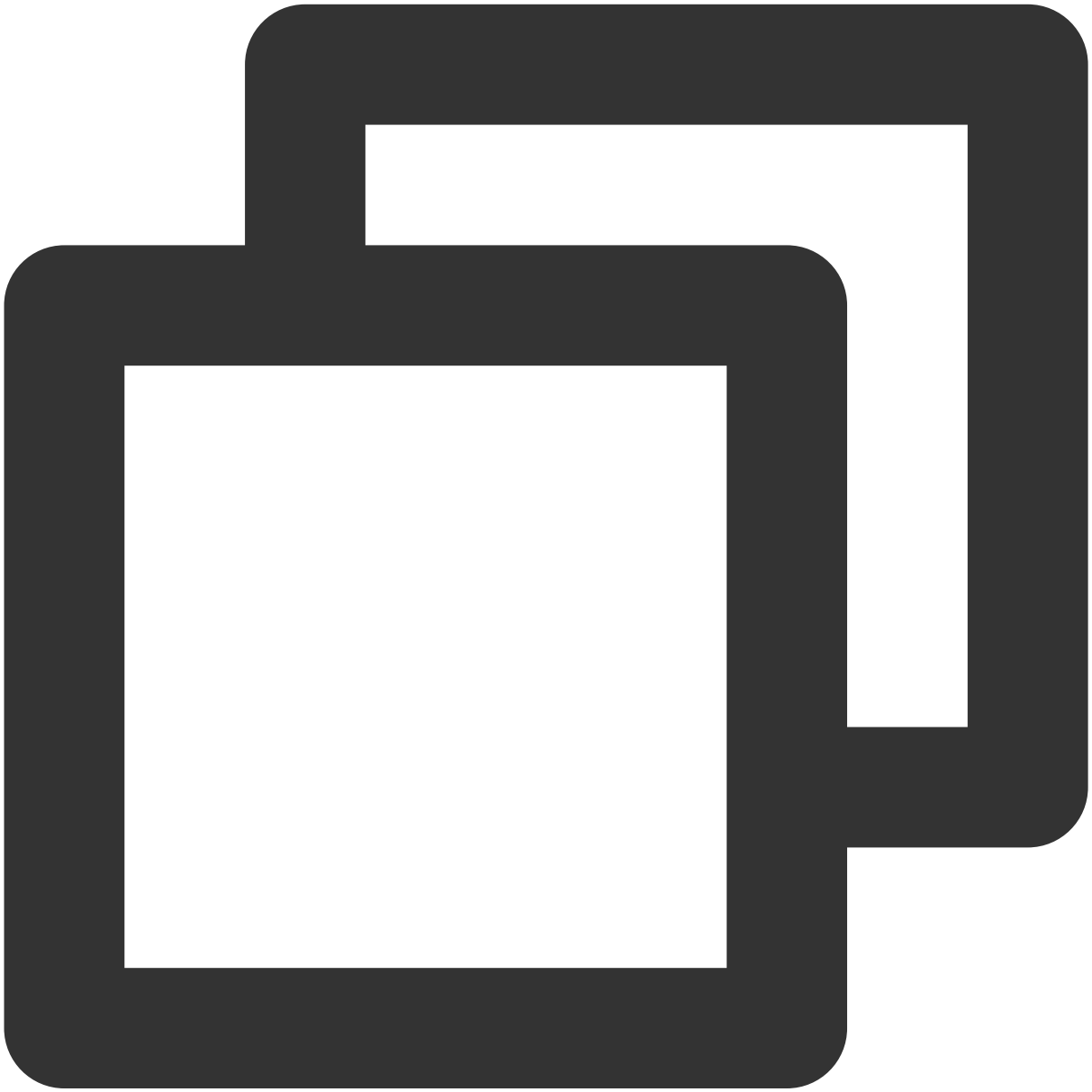
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallCancelled: (String userId) {
    }
));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the inviter.

onCallBegin

The call was connected. This callback is received by both the inviter and invitees. You can listen for this event to determine whether to start on-cloud recording, content moderation, or other tasks.



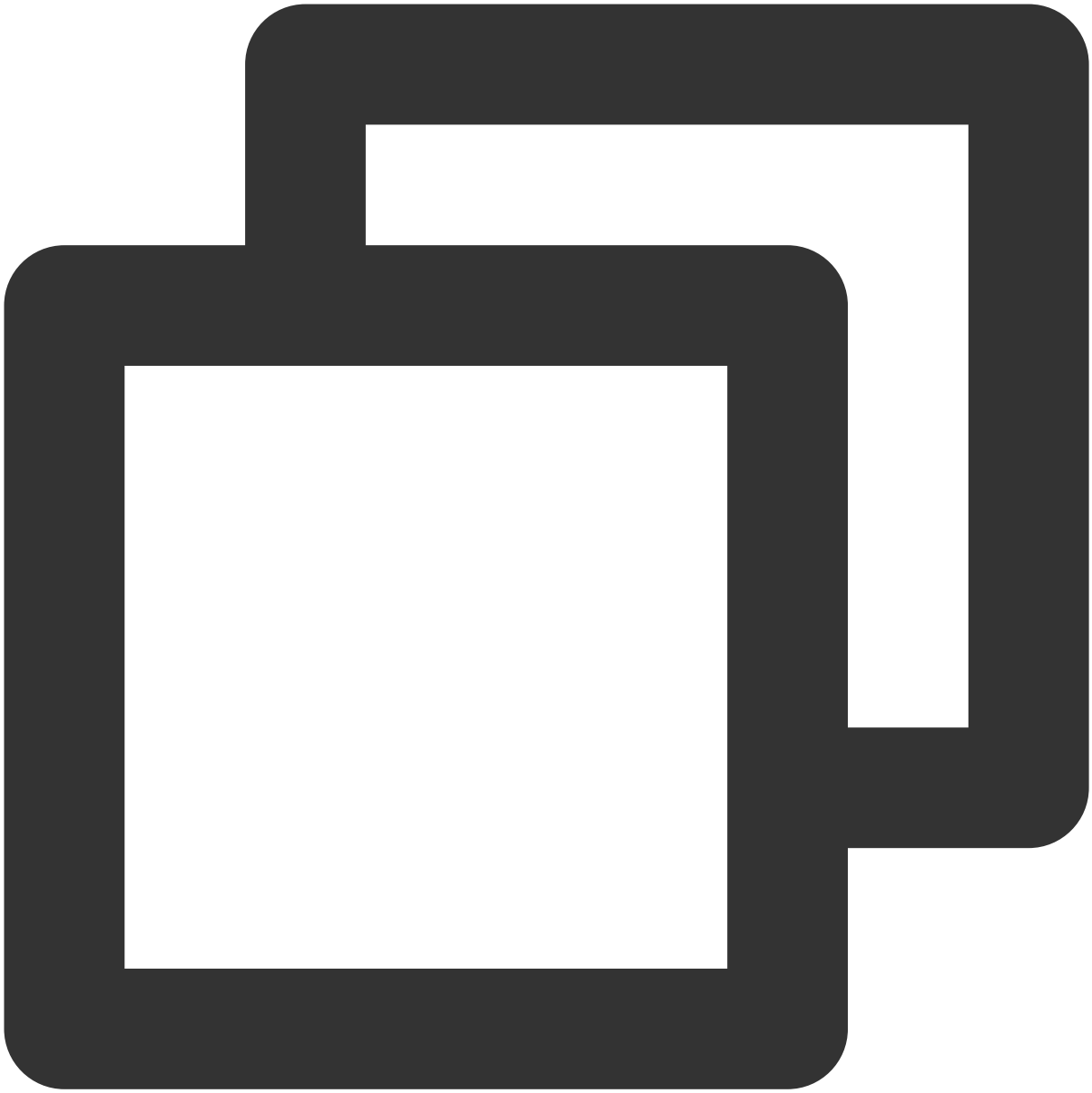
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole cal
    }
));
```

The parameters are described below:

Parameter	Type	Description
roomId	TUIRoomId	The room ID.
callMediaType	TUICallMediaType	The call type, which can be video or audio.
callRole	TUICallRole	The role, which can be caller or callee.

onCallEnd

The call ended. This callback is received by both the inviter and invitees. You can listen for this event to determine when to display call information such as call duration and call type, or stop on-cloud recording.



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole callR
    }
));
```

The parameters are described below:

Parameter	Type	Description
roomId	TUIRoomId	The room ID.

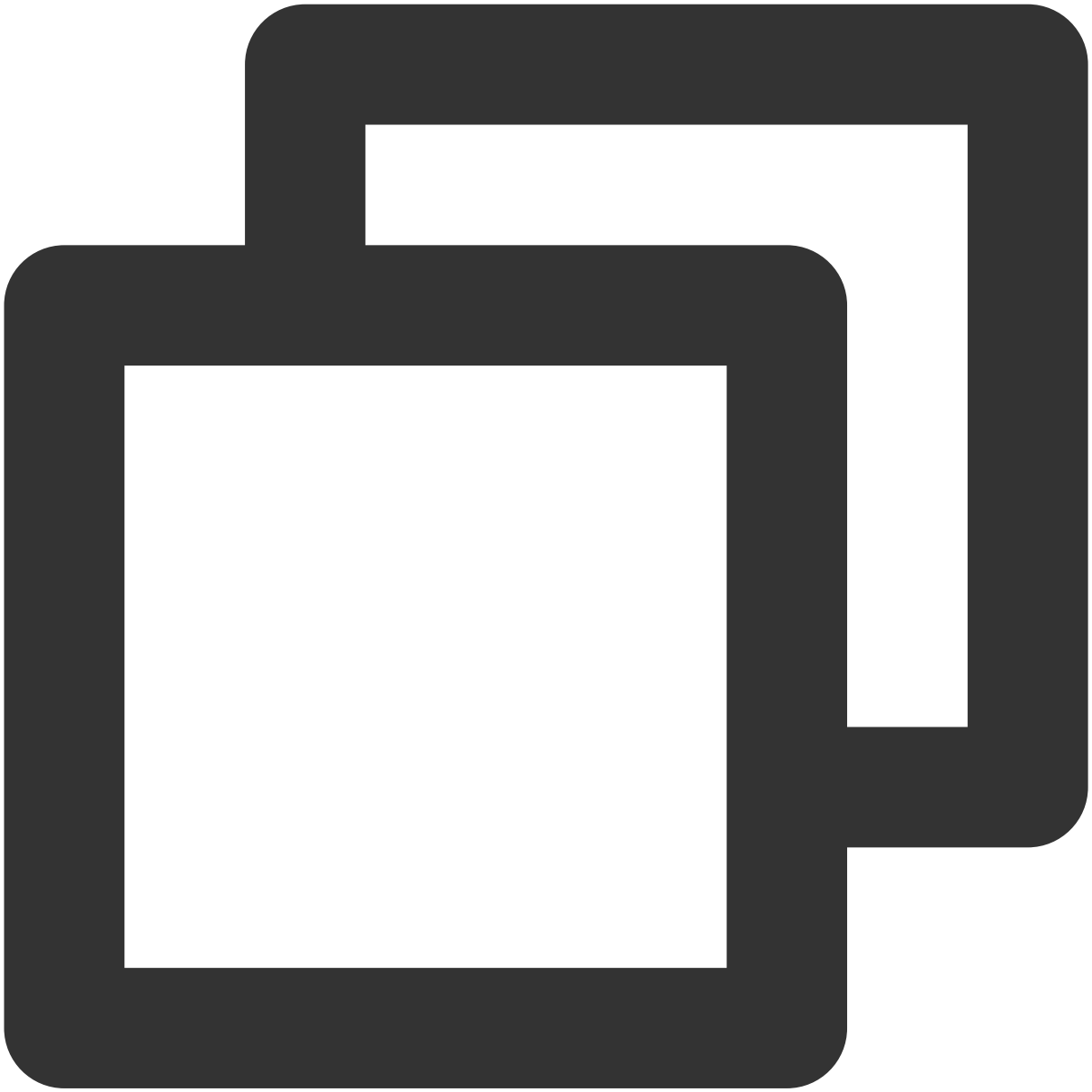
callMediaType	TUICallMediaType	The call type, which can be video or audio.
callRole	Number	The role, which can be caller or callee.
totalTime	double	The call duration.

Note

Client-side callbacks are often lost when errors occur, for example, when the process is closed. If you need to measure the duration of a call for billing or other purposes, we recommend you use the RESTful API.

onCallMediaTypeChanged

The call type changed.



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallMediaTypeChanged: (TUICallMediaType oldCallMediaType, TUICallMediaType ne
    }
));
```

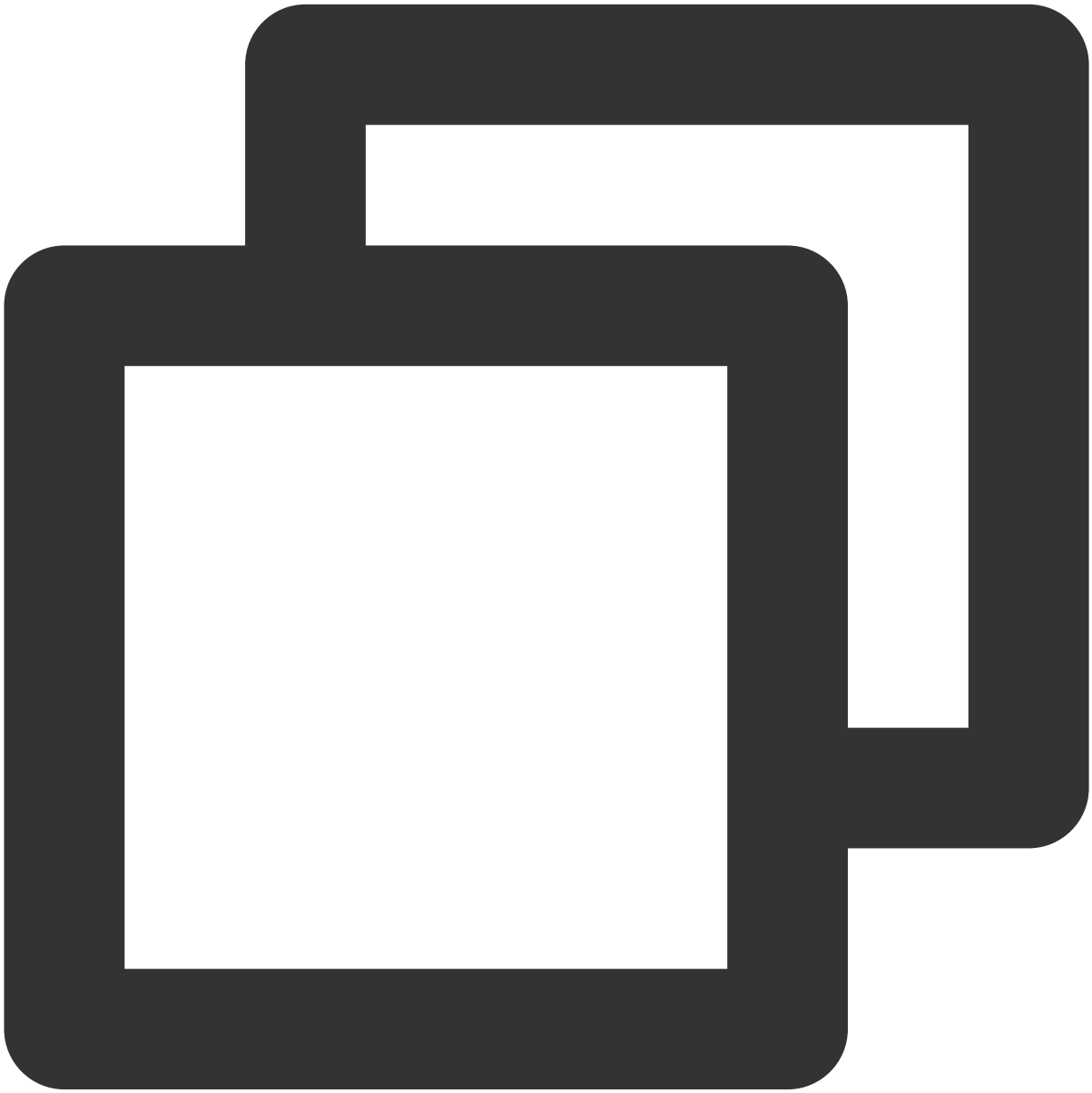
The parameters are described below:

Parameter	Type	Description
oldCallMediaType	TUICallMediaType	The call type before the change.

newCallMediaType	TUICallMediaType	The call type after the change.

onUserReject

The call was rejected. In a one-to-one call, only the inviter will receive this callback. In a group call, all invitees will receive this callback.



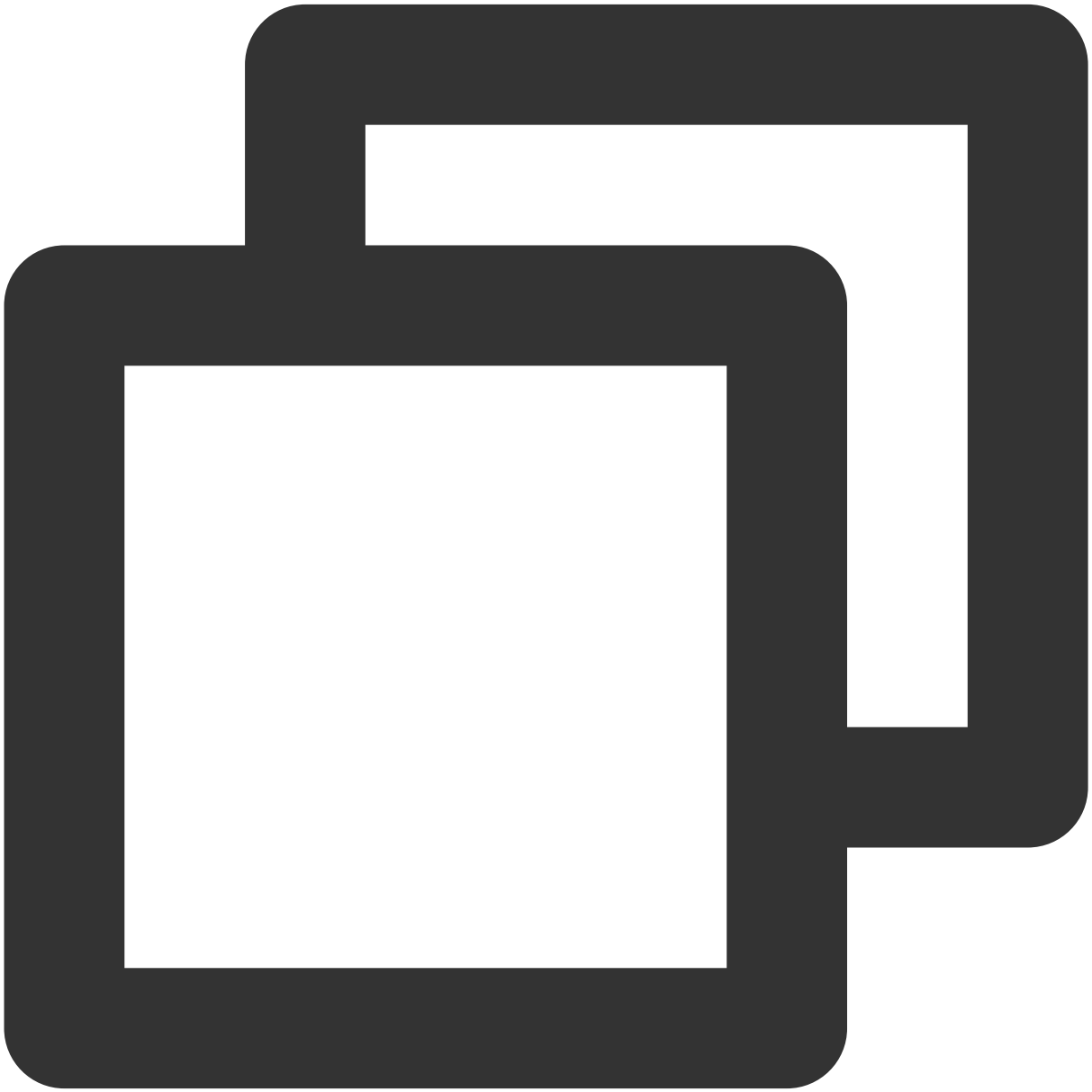
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserReject: (String userId) {
    }
));
```

The parameters are described below:

Parameter	Type	Description
res.userId	String	The user ID of the invitee who rejected the call.

onUserNoResponse

A user did not respond.



```
TUICallEngine.instance.addObserver(TUICallObserver(
```

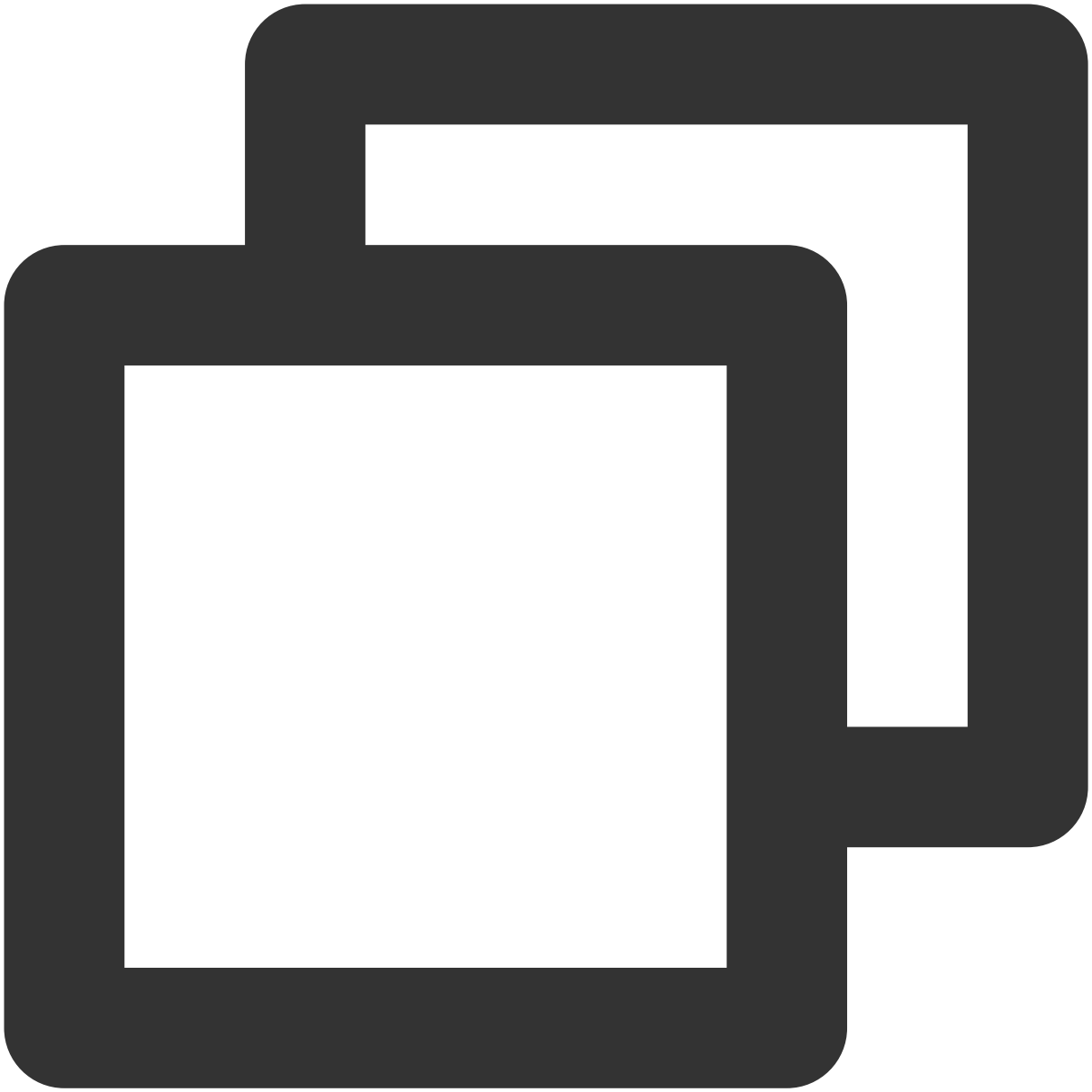
```
    onUserNoResponse: (String userId) {
    }
));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who did not answer.

onUserLineBusy

A user is busy.



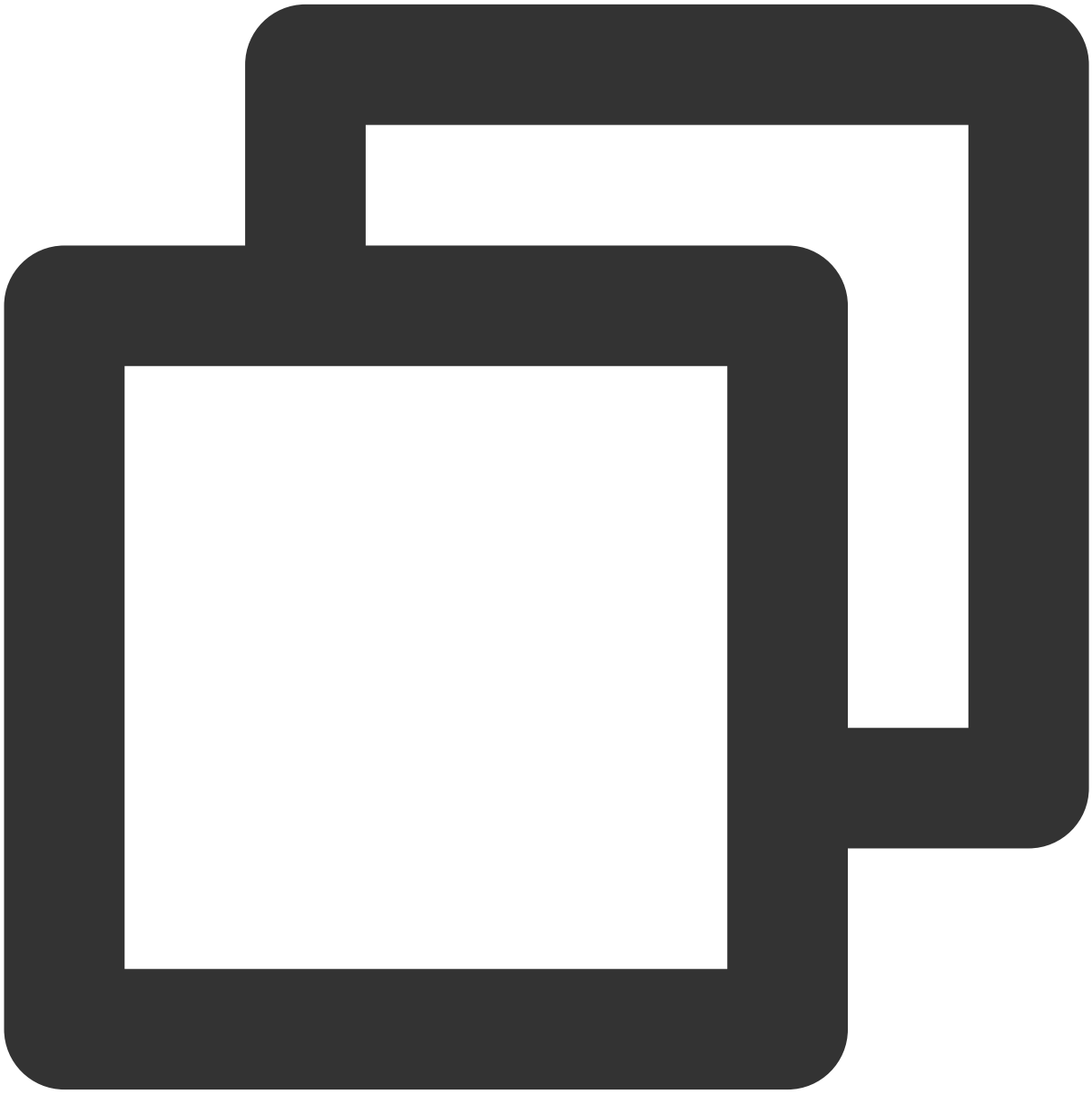
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserLineBusy: (String onUserLineBusy) {
    },
));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee who is busy.

onUserJoin

A user joined the call.



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserJoin: (String userId) {
    }
));
```

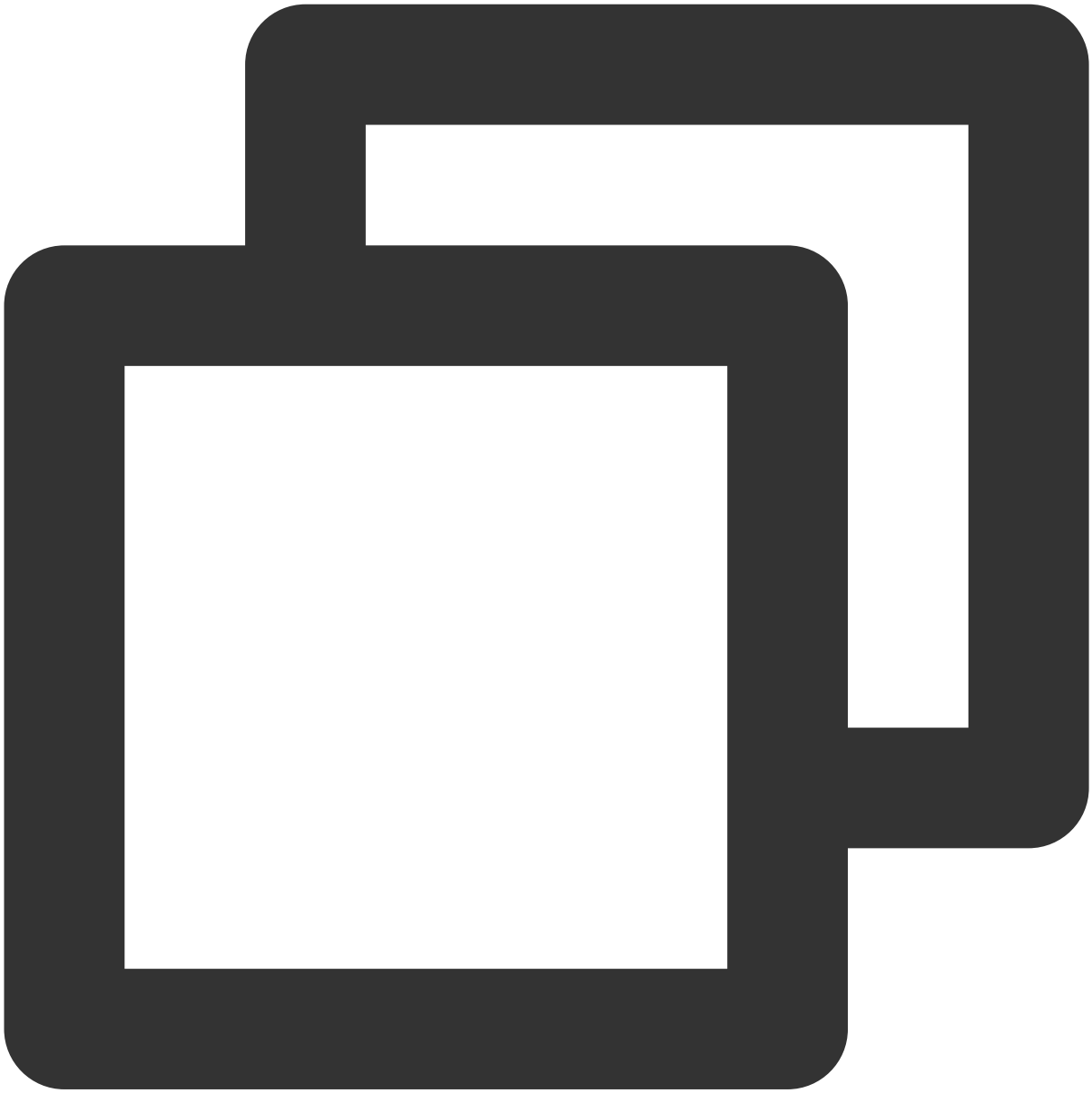
The parameters are described below:

Parameter	Type	Description

userId	String	The ID of the user who joined the call.

onUserLeave

A user left the call.



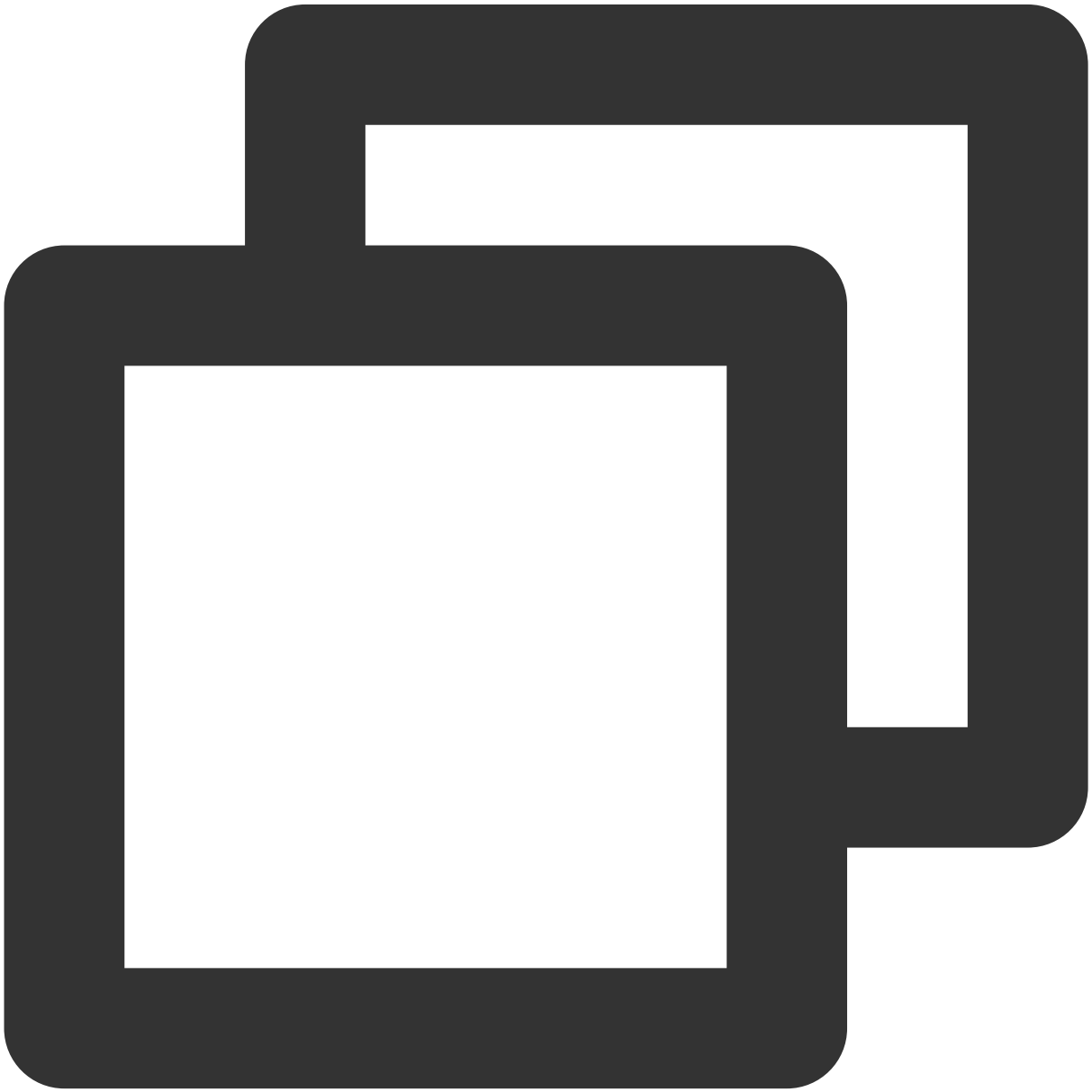
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserLeave: (String userId) {
    }
));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The ID of the user who left the call.

onUserVideoAvailable

Whether a user is sending video.



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserVideoAvailable: (String userId, bool isVideoAvailable) {
```

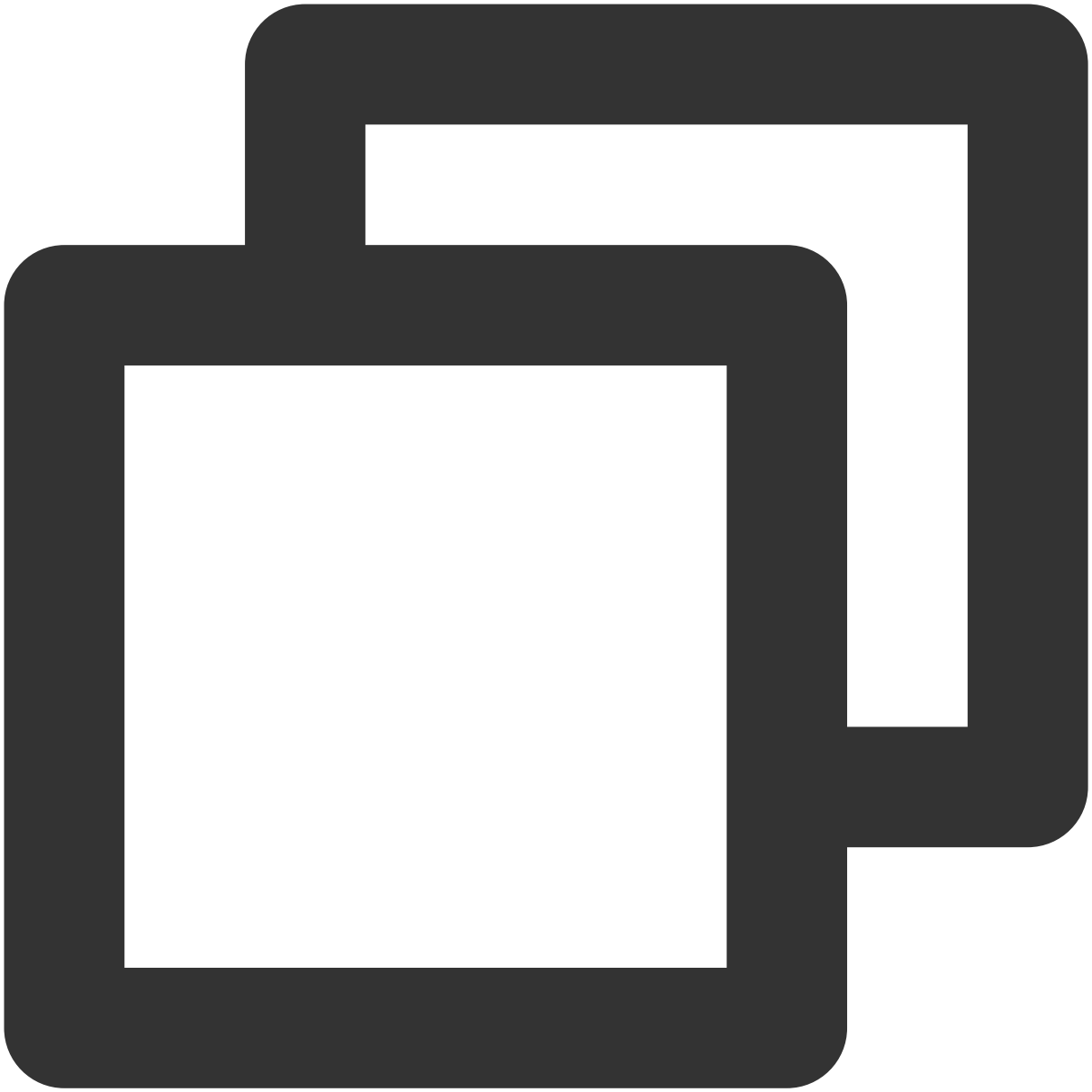
```
        }
    ));
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
isVideoAvailable	bool	Whether the user has video.

onUserAudioAvailable

Whether a user is sending audio.



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserAudioAvailable: (String userId, bool isAudioAvailable) {
    }
));
```

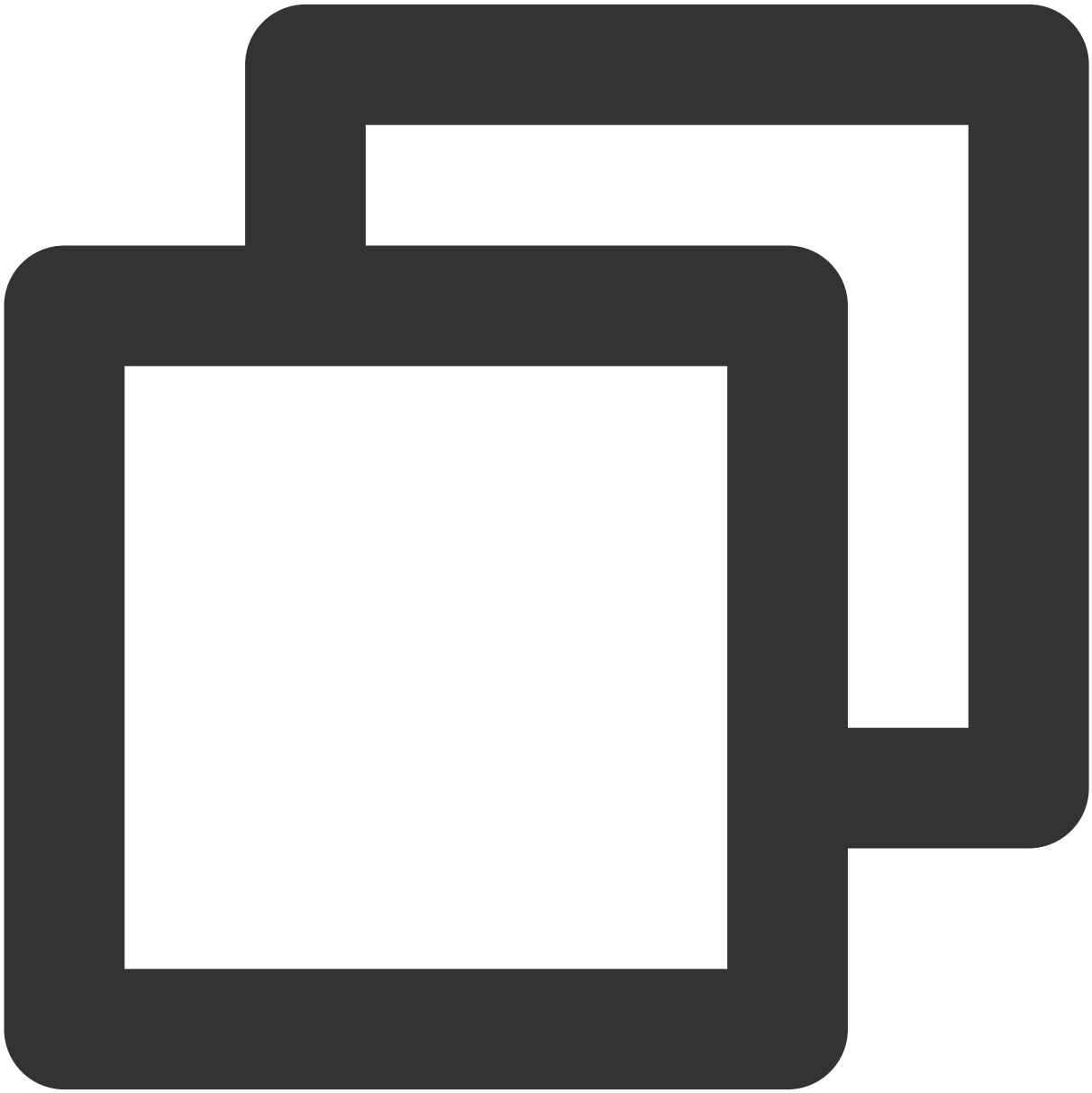
The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.

isAudioAvailable	bool	Whether the user has audio.

onUserVoiceVolumeChanged

The volumes of all users.



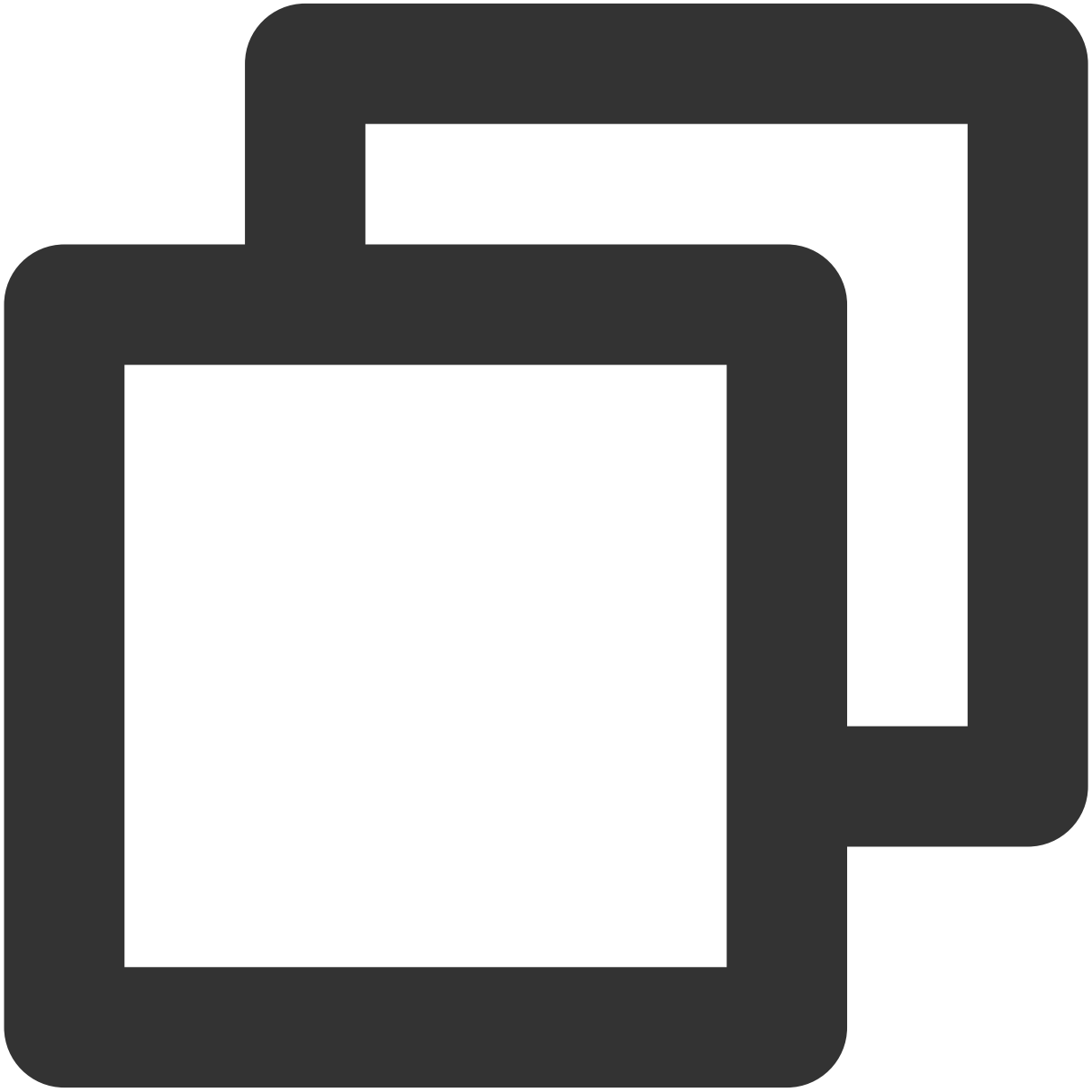
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserVoiceVolumeChanged: (Map<String, int> volumeMap) {
    }
));
```

The parameters are described below:

Parameter	Type	Description
volumeMap	Map<String, int>	The volume table, which includes the volume of each user (userId). Value range: 0-100.

onUserNetworkQualityChanged

The network quality of all users.



```
TUICallEngine.instance.addObserver(TUICallObserver(
```

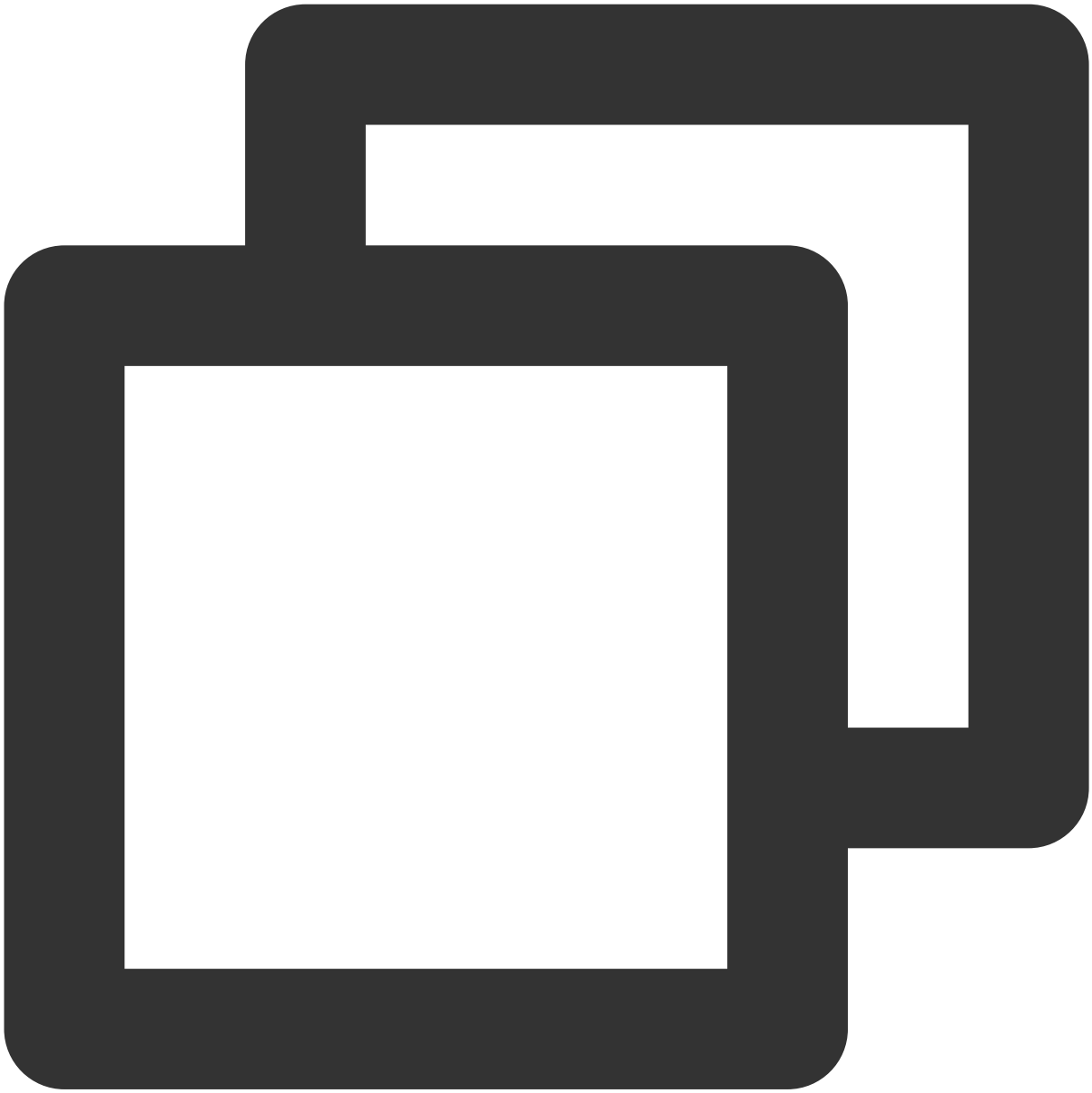
```
    onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList) {
    }
));

class TUINetworkQualityInfo {
    String userId;
    TUINetworkQuality quality;
    TUINetworkQualityInfo({required this.userId, required this.quality});
}

enum TUINetworkQuality {
  unknown,
  excellent,
  good,
  poor,
  bad,
  vBad,
  down
}
```

The parameters are described below:

Parameter	Type	Description
networkQualityList	List<TUINetworkQualityInfo>	The current network conditions for all users (userId).

onKickedOffline

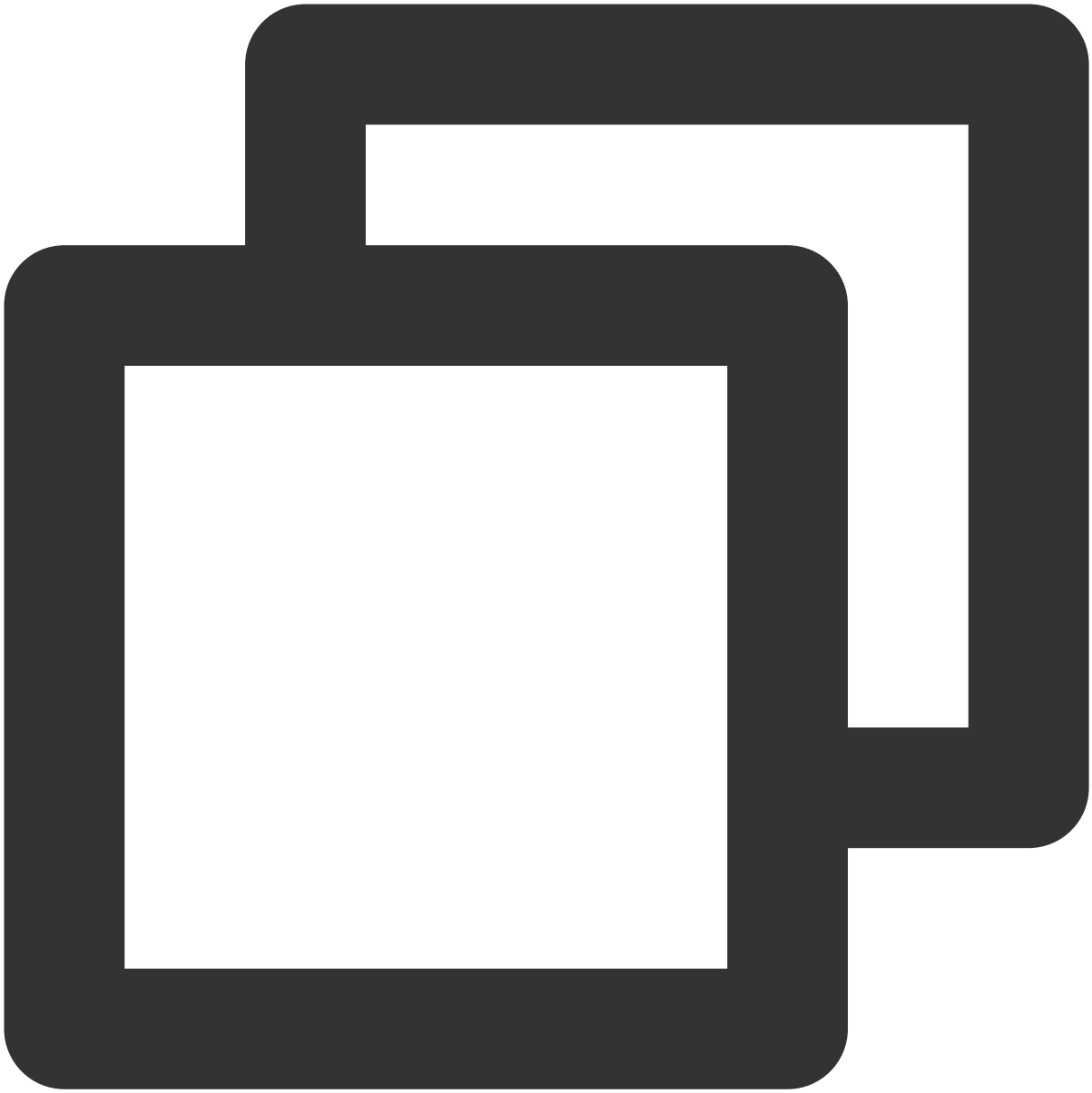
The current user was kicked offline：At this time, you can prompt the user with a UI message and then invoke `init` again.



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onKickedOffline: () {
    }
));
```

onUserSigExpired

The user sig is expired：At this time, you need to generate a new `userSig`, and then invoke `init` again.



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserSigExpired: () {
    }
));
```

Type Definition

Last updated：2023-09-21 15:37:01

Common structures

TUIResult

The return value of calling the API

Value	Type	Description
code	String	If the code is empty "", it means the call succeeded, if the code is not empty "", it means the call failed.
message	String?	Error message

TUIRoomId

Room ID for audio and video in a call.

Note：

(1) `intRoomId` and `strRoomId` are mutually exclusive. If you choose to use `strRoomId`, `intRoomId` needs to be filled in as 0. If both are filled in, the SDK will prioritize `intRoomId`.

(2) Do not mix `intRoomId` and `strRoomId` because they are not interchangeable. For example, the number 123 and the string "123" represent two completely different rooms.

Value	Type	Description
intRoomId	int	Numeric room ID.
strRoomId	String	String room number.

VideoRenderParams

Video render parameters

Value	Type	Description
fillMode	FillMode	Video image fill mode
rotation	Rotation	Video image rotation direction

VideoEncoderParams

Video encoding parameters

Value	Type	Description
resolution	Resolution	Video resolution
resolutionMode	ResolutionMode	Video aspect ratio mode

TUICallParams

Call params

Value	Type	Description
roomId	TUIRoomId	Room Id.
offlinePushInfo	TUIOfflinePushInfo	Offline push vendor configuration information.
timeout	String	Call timeout period, default: 30s, unit: seconds.
userData	String	An additional parameter.

TUIOfflinePushInfo

Offline push vendor configuration information, please refer to:Offline call push

Value	Type	Description
title	String	offlinepush notification title
desc	String	offlinepush notification description
ignoreIOSBadge	bool	Ignore badge count for offline push (only for iOS), if set to true, the message will not increase the unread count of the app icon on the iOS receiver's side.
iOSSound	String	Offline push sound setting (only for iOS). When sound = IOS_OFFLINE_PUSH_NO_SOUND , there will be no sound played when the message is received. When sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND , the system sound will be played when the message is received. If you want to customize the iOSSound, you need to link the audio file into

		the Xcode project first, and then set the audio file name (with extension) to the iOSSound.
androidSound	String	Offline push sound setting (only for Android, supported by IMSDK 6.1 and above). Only Huawei and Google phones support setting sound prompts. For Xiaomi phones, please refer to: Xiaomi custom ringtones . In addition, for Google phones, in order to set sound prompts for FCM push on Android 8.0 and above systems, you must call setAndroidFCMChannelID to set the channelID for it to take effect.
androidOPPOChannelID	String	Set the channel ID for OPPO phones with Android 8.0 and above systems.
androidVIVOClassification	int	Classification of VIVO push messages (deprecated interface, VIVO push service will optimize message classification rules on April 3, 2023. It is recommended to use setAndroidVIVOCategory to set the message category). 0: Operational messages, 1: System messages. The default value is 1.
androidXiaoMiChannelID	String	Set the channel ID for Xiaomi phones with Android 8.0 and above systems.
androidFCMChannelID	String	Set the channel ID for google phones with Android 8.0 and above systems.
androidHuaWeiCategory	String	Classification of Huawei push messages, please refer to: Huawei message classification standard.
isDisablePush	bool	Whether to turn off push notifications (default is on).
iOSPushType	TUICallIOSOfflinePushType	iOS offline push type，default is APNs

TUICallRecords

Call recording information

Value	Type	Description

callId	String	Call recording ID.
inviter	String	Inviter ID.
inviteList	List<String>	List of invited user IDs.
scene	TUICallScene	Call scenario.
mediaType	TUICallMediaType	Media type.
groupId	String	Group ID.
role	TUICallRole	Role.
result	TUICallResultType	Call result type.
beginTime	int	Start time.
totalTime	int	Total time.

TUICallRecentCallsFilter

Call recording filtering conditions.

Value	Type	Description
begin	double	Start time.
end	double	End time.
resultType	TUICallResultType	Call result type.

enum definition

TUICallMediaType

Call media type

Type	Value	Description
none	0	Unknown
audio	1	Audio call
video	2	Video call

TUICallRole

Call role

Type	Value	Description
none	0	Unknown
caller	1	Caller（inviter）
called	2	Callee（invitee）

TUICallStatus

Call status

Type	Value	Description
none	0	Unknown
waiting	1	The call is currently waiting
accept	2	The call has been accepted

TUICallScene

Call scene

Type	Value	Description
groupCall	0	Group call
multiCall	1	Anonymous group calling (not supported at this moment, please stay tuned).
singleCall	2	one to one call

TUINetworkQuality

Network quality

Type	Value	Description
unknown	0	Unknown
excellent	1	Excellent
good	2	Good
poor	3	Poor

bad	4	Bad
vBad	5	Vbad
down	6	Down

FillMode

If the aspect ratio of the video display area is not equal to that of the video image, you need to specify the fill mode:

Type	Value	Description
fill	0	Fill mode: the video image will be centered and scaled to fill the entire display area, where parts that exceed the area will be cropped. The displayed image may be incomplete in this mode.
fit	1	Fit mode: the video image will be scaled based on its long side to fit the display area, where the short side will be filled with black bars. The displayed image is complete in this mode, but there may be black bars.

Rotation

We provides rotation angle setting APIs for local and remote images. The following rotation angles are all clockwise.

Type	Value	Description
rotation_0	0	No rotation
rotation_90	1	Clockwise rotation by 90 degrees
rotation_180	2	Clockwise rotation by 180 degrees
rotation_270	3	Clockwise rotation by 270 degrees

ResolutionMode

Video aspect ratio mode

Type	Value	Description
landscape	0	Landscape resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360.
portrait	1	Portrait resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640.

Resolution

Video resolution

Type	Value	Description
resolution_640_360	0	Aspect ratio: 16:9；resolution: 640x360；recommended bitrate: 500kbps
resolution_640_480	1	Aspect ratio: 4:3；resolution: 640x480；recommended bitrate: 600kbps
resolution_960_540	2	Aspect ratio: 16:9；resolution: 960x540；recommended bitrate: 850kbps
resolution_960_720	3	Aspect ratio: 4:3；resolution: 960x720；recommended bitrate: 1000kbps
resolution_1280_720	4	Aspect ratio: 16:9；resolution: 1280x720；recommended bitrate: 1200kbps
resolution_1920_1080	5	Aspect ratio: 16:9；resolution: 1920x1080；recommended bitrate: 2000kbps

TUICallIOSOfflinePushType

iOS offline push type

Type	Value	Description
APNs	0	APNs
VoIP	1	VoIP

TUICamera

Camera type.

Type	Value	Description
front	0	Front camera.
back	1	Rear camera.

TUIAudioPlaybackDevice

Audio playback device.

Type	Value	Description

speakerphone	0	Speaker
earpiece	1	Earpiece

TUICallResultType

Call recording type.

Type	Value	Description
unknown	0	Unknown
missed	1	Missed
incoming	2	Incoming call
outgoing	3	Outgoing Call

uniapp（Android&iOS）
API Overview

Last updated：2024-07-19 14:15:49

TUICallKit (UI Included)

TUICallKit is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat.

API	Description
login	login
logout	logou
setSelfInfo	Sets the user nickname and profile photo.
call	Makes a one-to-one call.
groupCall	Makes a group call.
joinInGroupCall	Joins a group call.
setCallingBell	Sets the ringtone.
enableMuteMode	Sets whether to turn on the mute mode.
enableFloatWindow	Sets whether to enable floating windows.
enableIncomingBanner	Sets whether to display incoming banner. **v2.3.1+ supported.**

Event

TUICallKit throws the following events.

API	Description
onError	An error occurred during the call.
onCallReceived	A call was received.
onCallCancelled	The call was canceled.

onCallBegin	The call was connected.
onCallEnd	The call ended.
onCallMediaTypeChanged	The call type changed.
onUserReject	A user declined the call.
onUserNoResponse	A user didn't respond.
onUserLineBusy	A user was busy.
onUserJoin	A user joined the call.
onUserLeave	A user left the call.
onUserVideoAvailable	Whether a user has a video stream.
onUserAudioAvailable	Whether a user has an audio stream.
onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.
onKickedOffline	The current user was kicked offline.
onUserSigExpired	The user sig is expired.

TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description
hangup	Ends a call.
accept	Answers a call.
setVideoRenderParams	Set the rendering mode of video image.
setVideoEncoderParams	Set the encoding parameters of video encoder.

TUICallKit

Last updated：2024-07-19 14:15:49

TUICallKit APIs

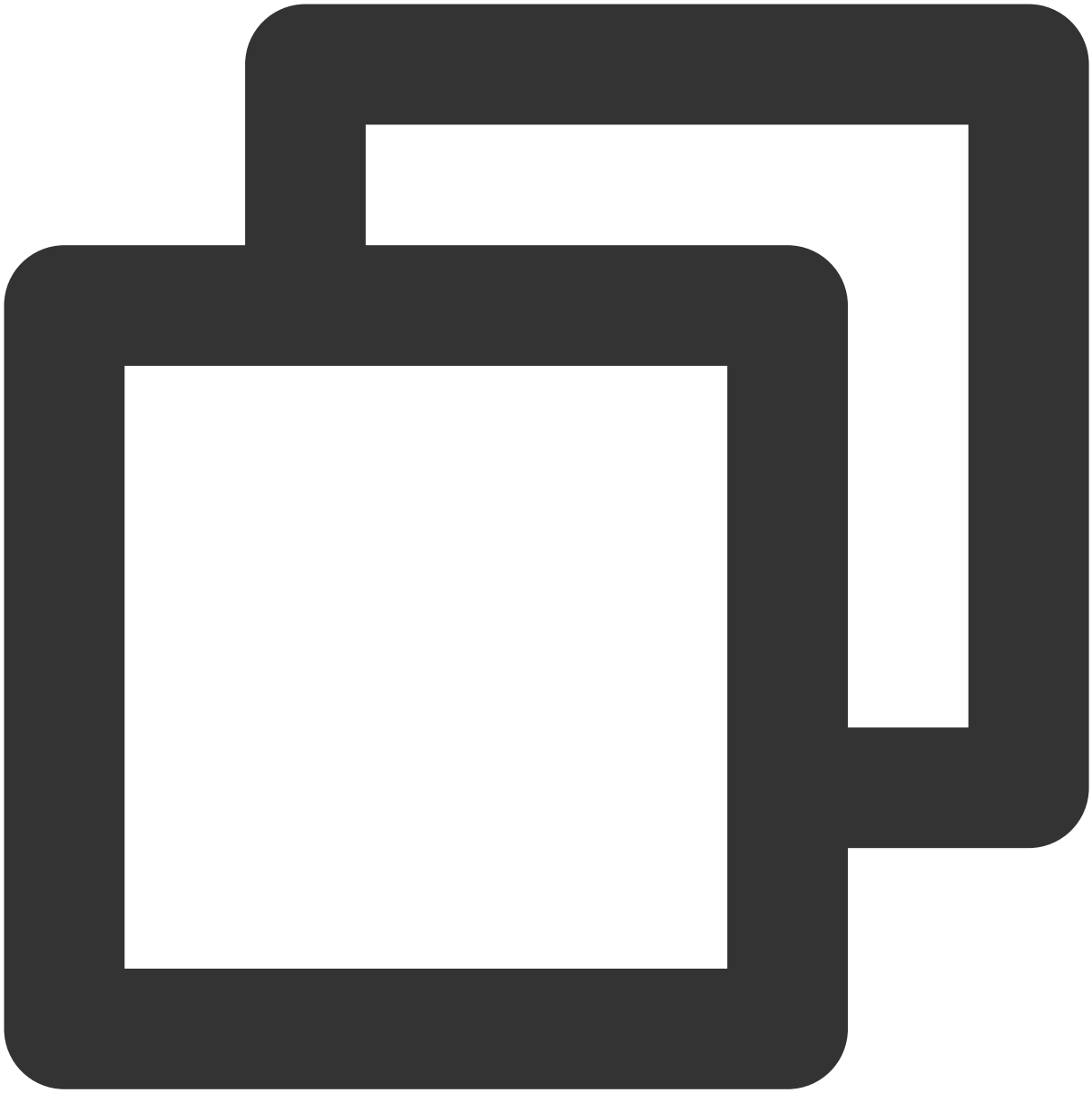
`TUICallKit` is an audio/video call component that **includes UI elements**. You can use its APIs to quickly implement an audio/video call application similar to WeChat. For directions on integration, see Integrating TUICallKit.

API Overview

API	Description
login	login
logout	logout
setSelfInfo	Sets the user nickname and profile photo.
call	Makes a one-to-one call.
groupCall	Makes a group call.
joinInGroupCall	Joins a group call.
setCallingBell	Sets the ringtone.
enableMuteMode	Sets whether to turn on the mute mode.
enableFloatWindow	Sets whether to enable floating windows.
enableIncomingBanner	Sets whether to display incoming banner. **v2.3.1 supported.**

API Detail

login

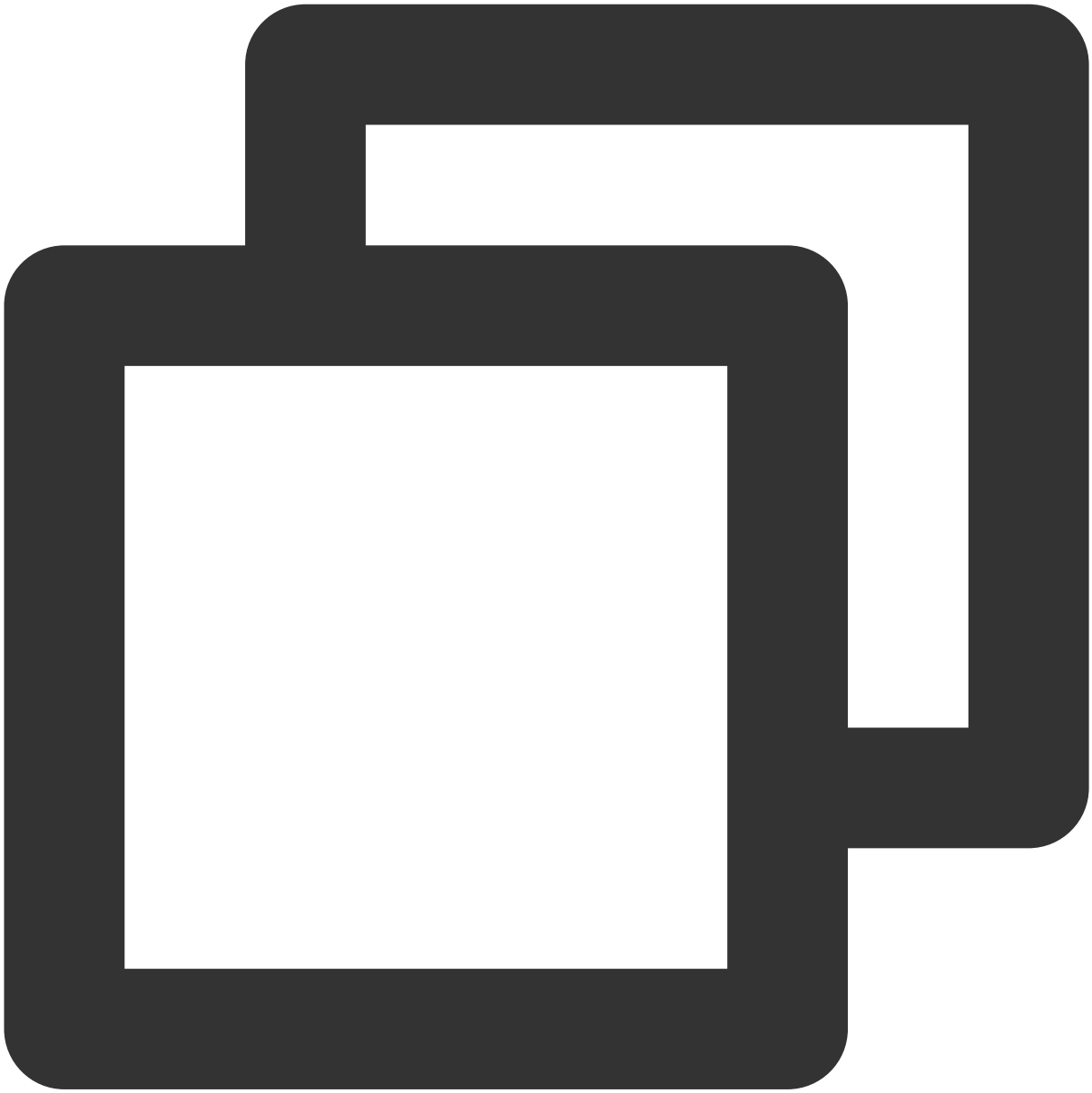


```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  SDKAppID: 0,
  userID: 'mike',
  userSig: '',
};
TUICallKit.login(options, (res) => {
  if (res.code === 0) {
    console.log('login success');
  } else {
    console.log(`login failed, error message = ${res.msg}`);
```

```
    }
});
```

Parameter	Type	Description
options	Object	Initialization parameters
options.SDKAppID	Number	User SDKAppID
options.userID	String	userID
options.userSig	String	User Signature
callback	Function	callback function, code = 0 means the call was successful; code ! = 0 means the call failed, see msg for the reason.

logout



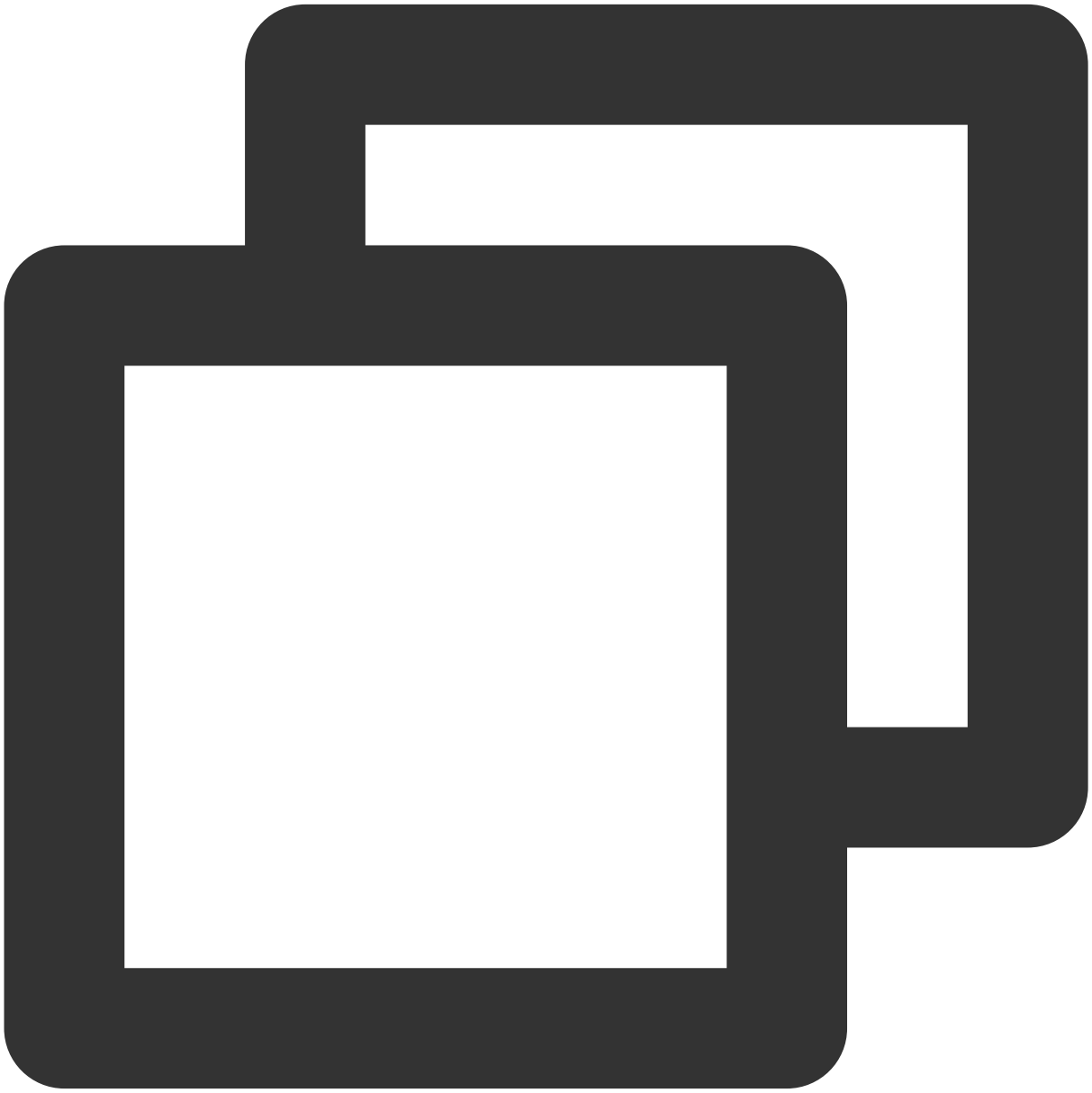
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
TUICallKit.logout((res) => {
  if (res.code === 0) {
    console.log('logout success');
  } else {
    console.log(`logout failed, error message = ${res.msg}`);
  }
});
```

Parameter	Type	Description

callback	Function	callback function, code = 0 means the call was successful; code ! = 0 means the call failed, see msg for the reason.

setSelfInfo

This API is used to set the alias and profile photo. The alias cannot exceed 500 bytes, and the profile photo is specified by a URL.



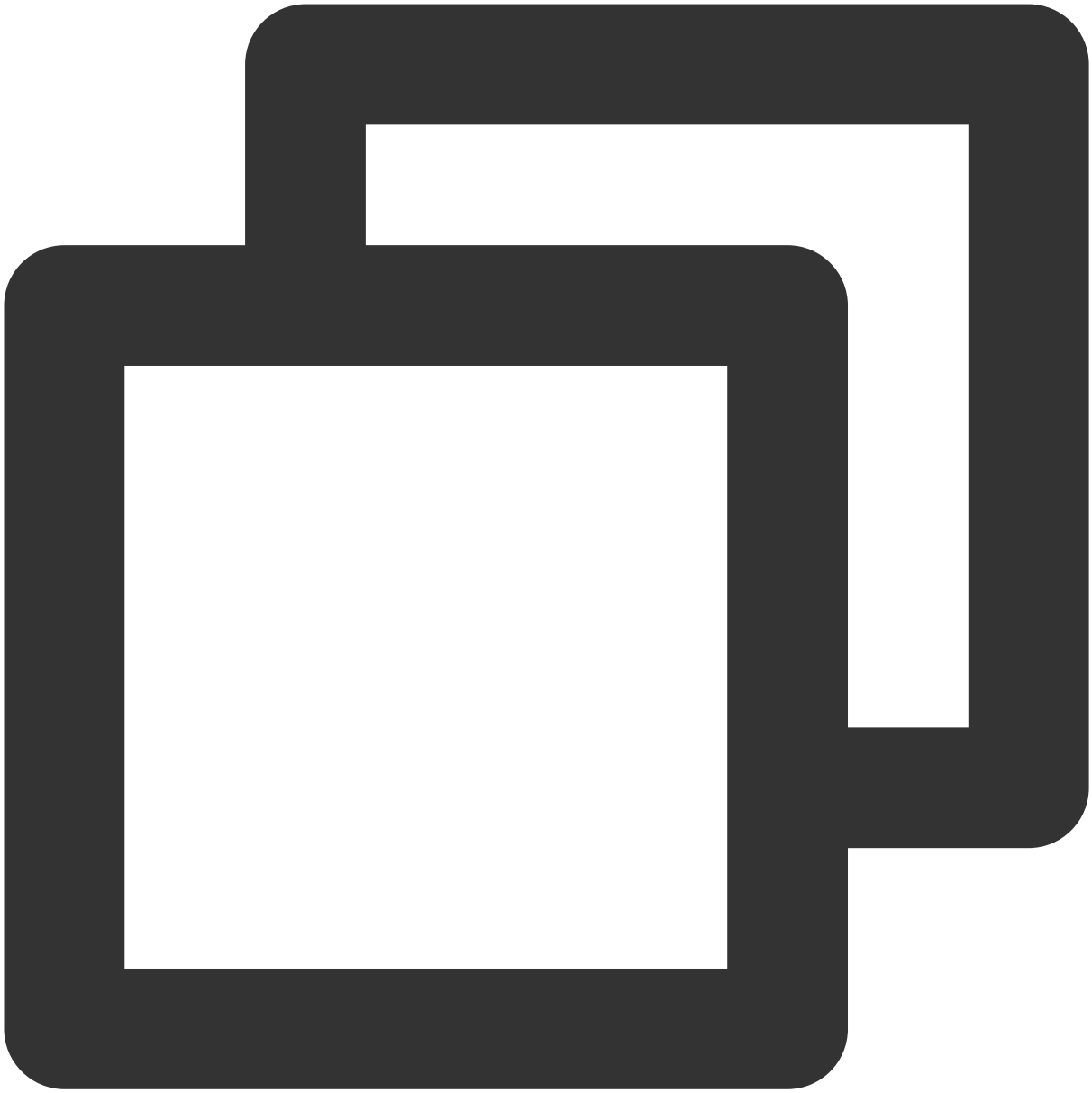
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  nickName: 'jack',
```

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

Parameter	Type	Description
options	Object	Initialization parameters
options.nickName	String	Nickname of the target user, not required
options.avatar	String	Target user's avatar, not required
callback	Function	callback function, code = 0 means the call was successful; code ! = 0 means the call failed, see msg for the reason.

call

This API is used to make a (one-to-one) call.



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  userID: 'mike',
  callMediaType: 1, // audio call(callMediaType = 1)、video call(callMediaType = 2)
  roomID: 0,
  strRoomID: '1223',
};
TUICallKit.call(options, (res) => {
  if (res.code === 0) {
    console.log('call success');
  } else {
```

```
    console.log(`call failed, error message = ${res.msg}`);
  }
});
```

The parameters are described below:

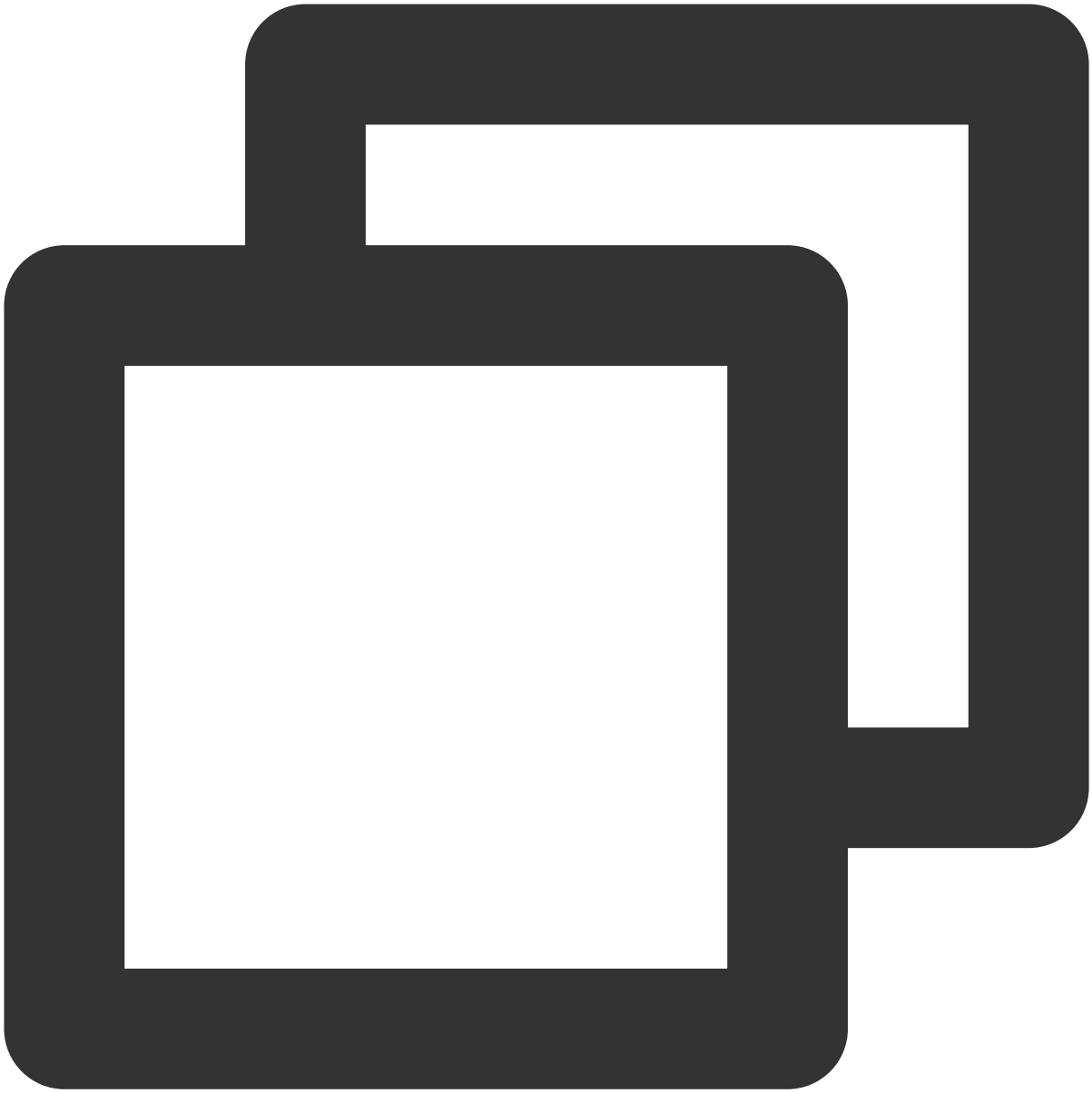
Parameter	Type	Description
options	Object	Initialization parameters
options.userID	String	The userID of the target user
options.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
options.roomID	Number	Customize the numeric room number. As long as the roomID is present, the numeric room number is used, even if strRoomID is present.
options.strRoomID	String	Customize the string room number. If you want to use a string room number, you need to set roomID = 0 after setting strRoomID.
callback	Function	callback function, code = 0 means the call was successful; code ! = 0 means the call failed, see msg for the reason.

groupCall

This API is used to make a group call.

Note:

Before making a group call, you need to create an IM group first.



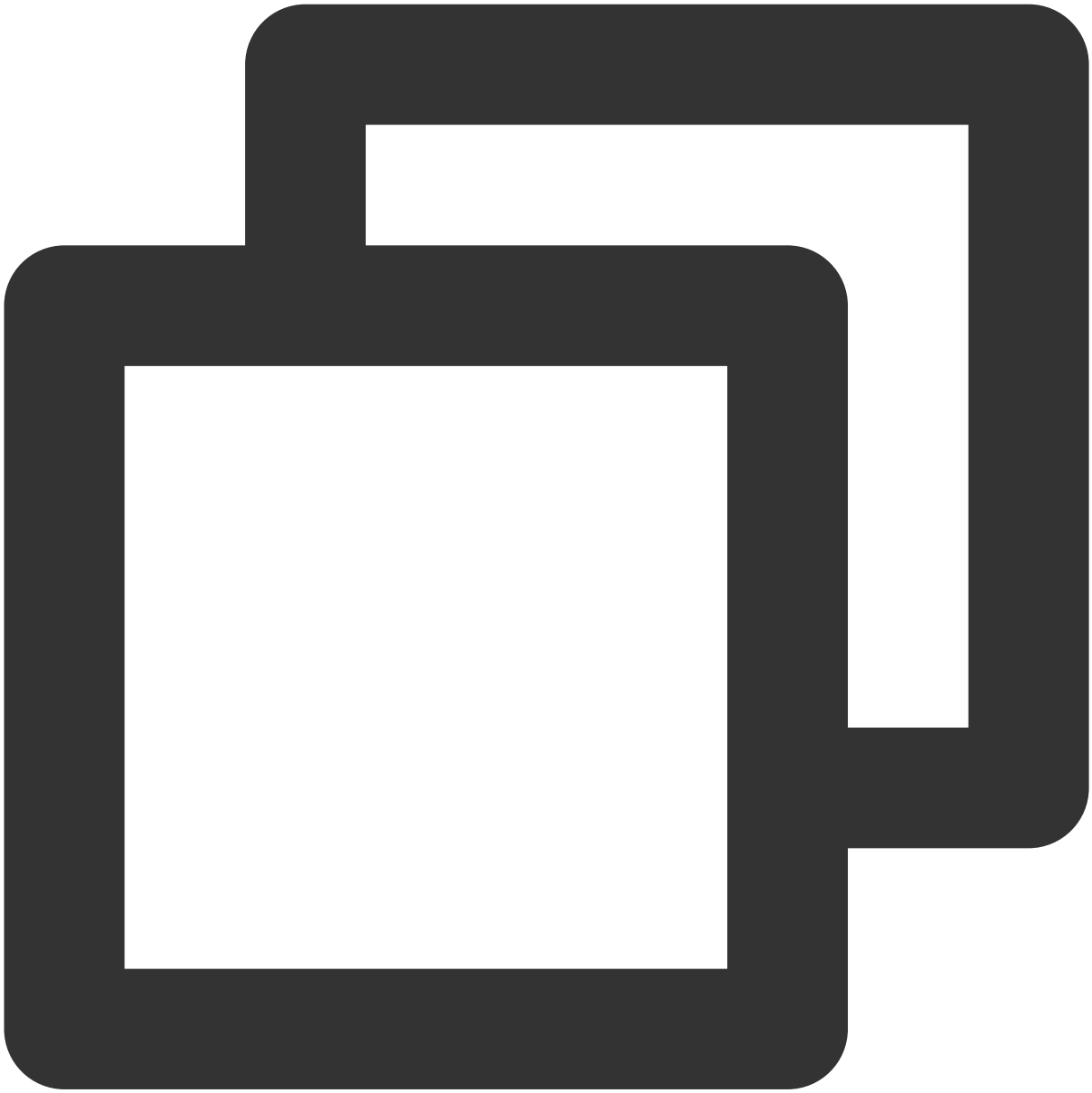
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  groupID: 'myGroup',
  userIDList: ['mike', 'tom'],
  callMediaType: 1, // audio call(callMediaType = 1)、video call(callMediaType = 2)
};
TUICallKit.groupCall(options, (res) => {
  if (res.code === 0) {
   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

```
    }
});
```

Parameter	Type	Description
options	Object	Initialization parameters
options.groupID	String	Group ID for this group cal
options.userIDList	List	The target user IDs.
options.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
options.roomID	Number	Customize the numeric room number. As long as the roomID is present, the numeric room number is used, even if strRoomID is present.
options.strRoomID	String	Customize the string room number. If you want to use a string room number, you need to set roomID = 0 after setting strRoomID.

joinInGroupCall

This API is used to join a group call.



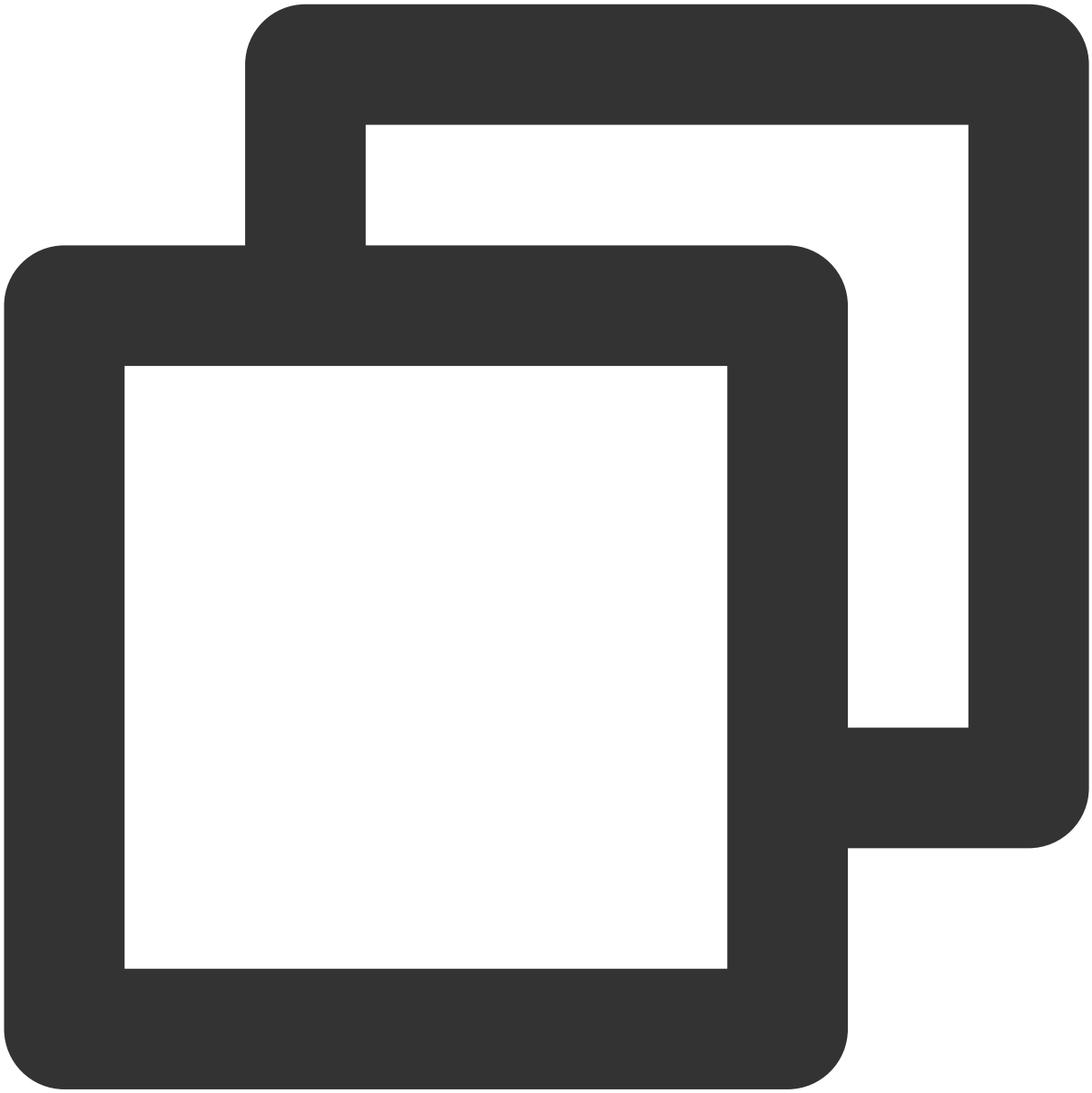
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  roomID: 9898,
  groupID: 'myGroup',
  callMediaType: 1, // audio call(callMediaType = 1)、video call(callMediaType = 2)
};
TUICallKit.joinInGroupCall(options, (res) => {
  if (res.code === 0) {
    console.log('joinInGroupCall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

```
    }
});
```

Parameter	Type	Description
options	Object	Initialization parameters
options.roomID	Number	Customize the numeric room number. As long as the roomID is present, the numeric room number is used, even if strRoomID is present.
options.strRoomID	String	Customize the string room number. If you want to use a string room number, you need to set roomID = 0 after setting strRoomID.
options.groupID	String	Group ID for this group cal
options.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
callback	Function	callback function, code = 0 means the call was successful; code ! = 0 means the call failed, see msg for the reason.

setCallingBell

To set a customized incoming call tone, here you are limited to passing in the local file address, and you need to make sure that the file directory is accessible to the application.



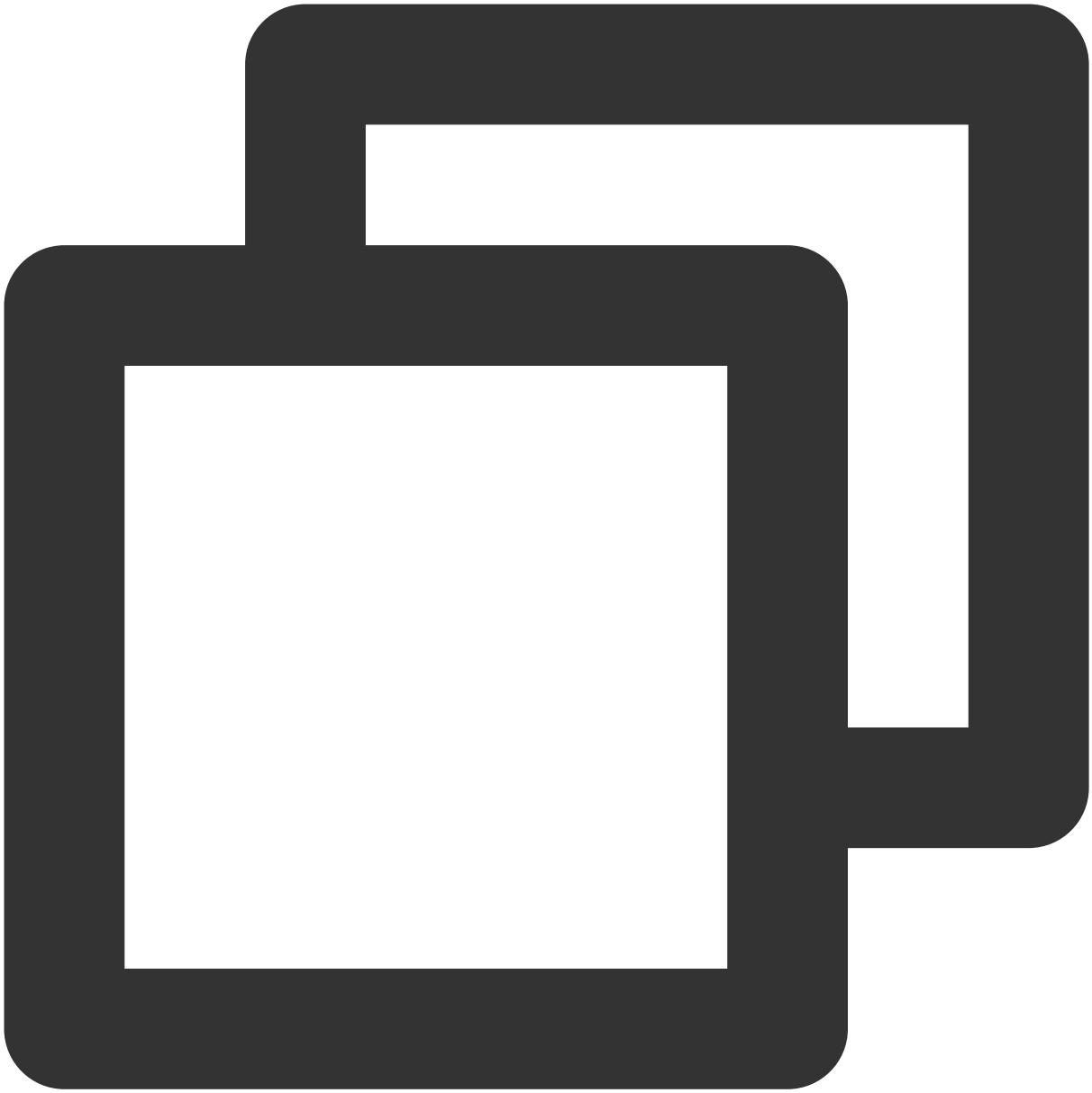
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');

// 【1】Save audio files locally through uni.saveFile. Reference.: https://zh.uniapp
const tempFilePath = './static/rain.mp3';
let musicFilePath = '';
uni.saveFile({
    tempFilePath: tempFilePath,
    success: (res) => {
        musicFilePath =  res.savedFilePath;

        // 【2】Convert relative path to absolute path, otherwise access will not be
```

```
        musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath);

        // 【3】set ringtone
        TUICallKit.setCallingBell(musicFilePath, (res) => {
          if (res.code === 0) {
            console.log('setCallingBell success');
          } else {
            console.log(`setCallingBell failed, error message = ${res.msg}`);
          }
        });
    },
    fail: (err) => {
      console.error('save failed');
    },
});
```

Parameter	Type	Description
filePath	String	Ringtone local file address
callback	Function	callback function, code = 0 means the call was successful; code ! = 0 means the call failed, see msg for the reason.

enableMuteMode

This API is used to set whether to turn on the mute mode.

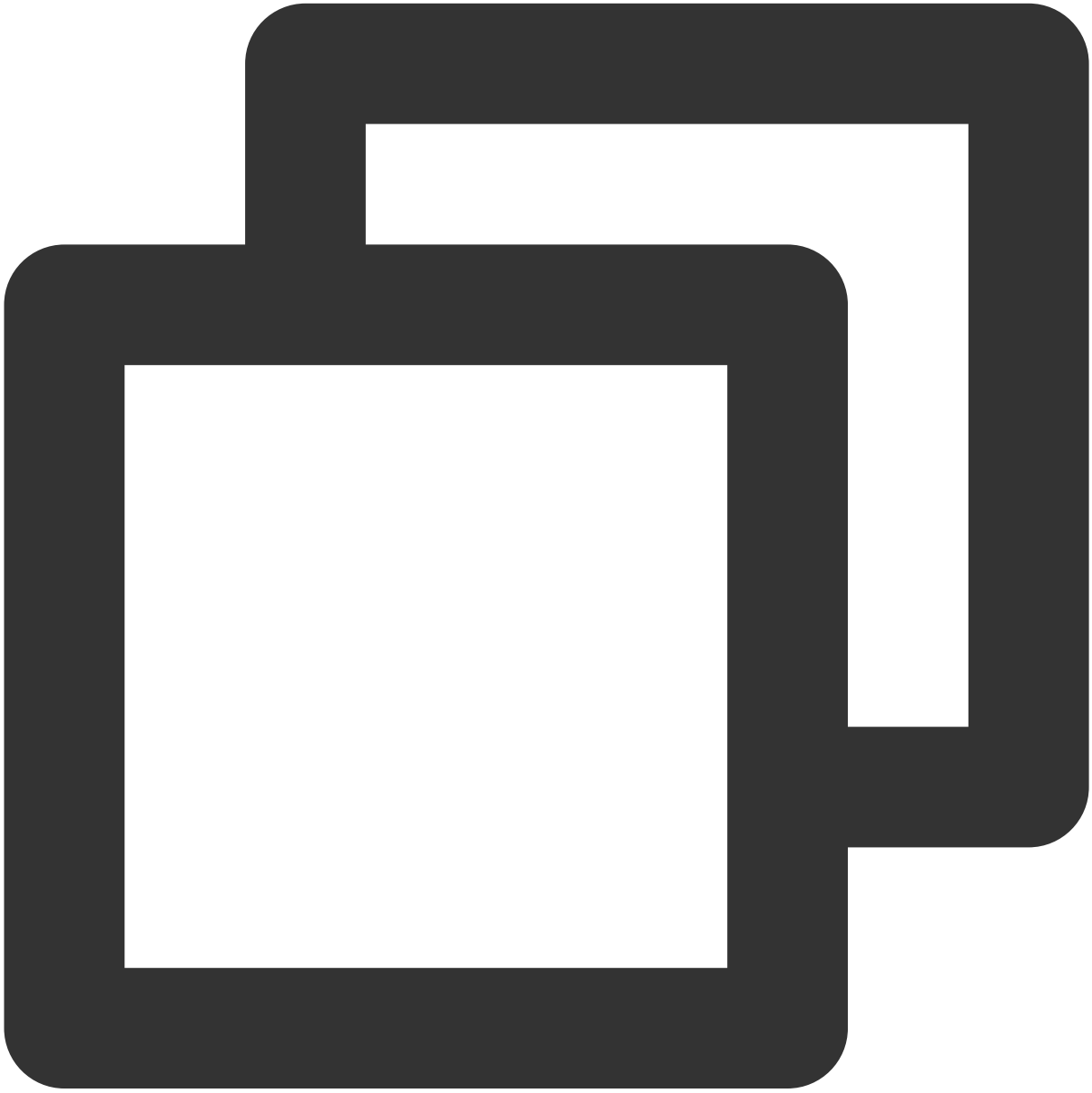


```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableMuteMode(enable);
```

Parameter	Type	Description
enable	Boolean	Mute on, mute off; true means mute on

enableFloatWindow

This API is used to set whether to enable floating windows.
The default value is `false`, and the floating window button in the top left corner of the call view is hidden. If it is set to `true`, the button will become visible.



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

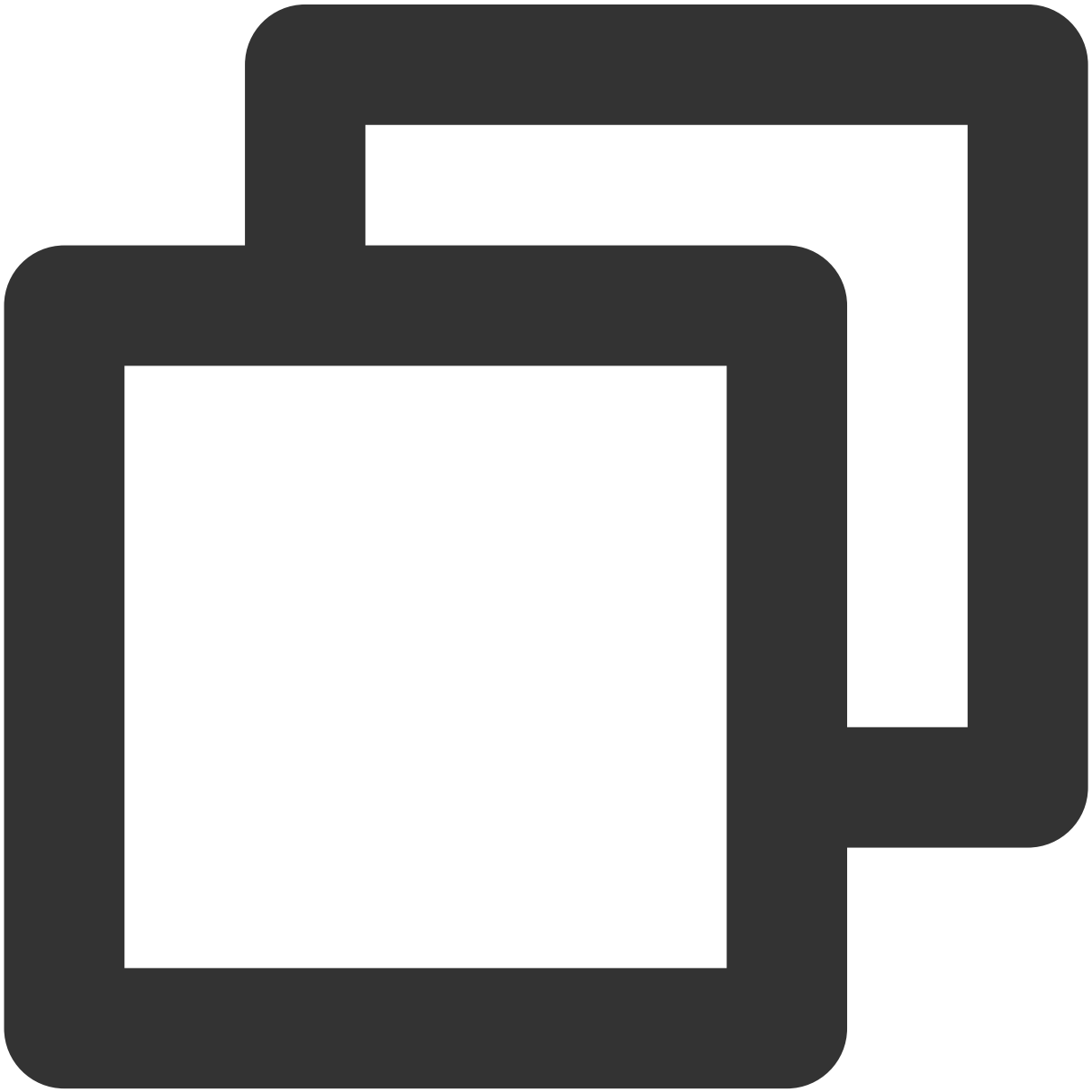
Parameter	Type	Description
enable	Boolean	Enable/disable the floating window function; true means floating window is enabled.

enableIncomingBanner

The API is used to set whether show incoming banner when user received a new call invitation.
The default value is `false` , The callee will pop up a full-screen call view by default when receiving the invitation. If it is set to `true` , the callee will display a banner first.

Note：

v2.3.1 supported



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

```
const enable = true;
TUICallKit.enableIncomingBanner(enable);
```

TUICallEngine

Last updated：2024-03-07 10:46:00

TUICallEngine API

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

Note

It needs to be used with TencentCloud-TUICallKit plugin and cannot be used alone.

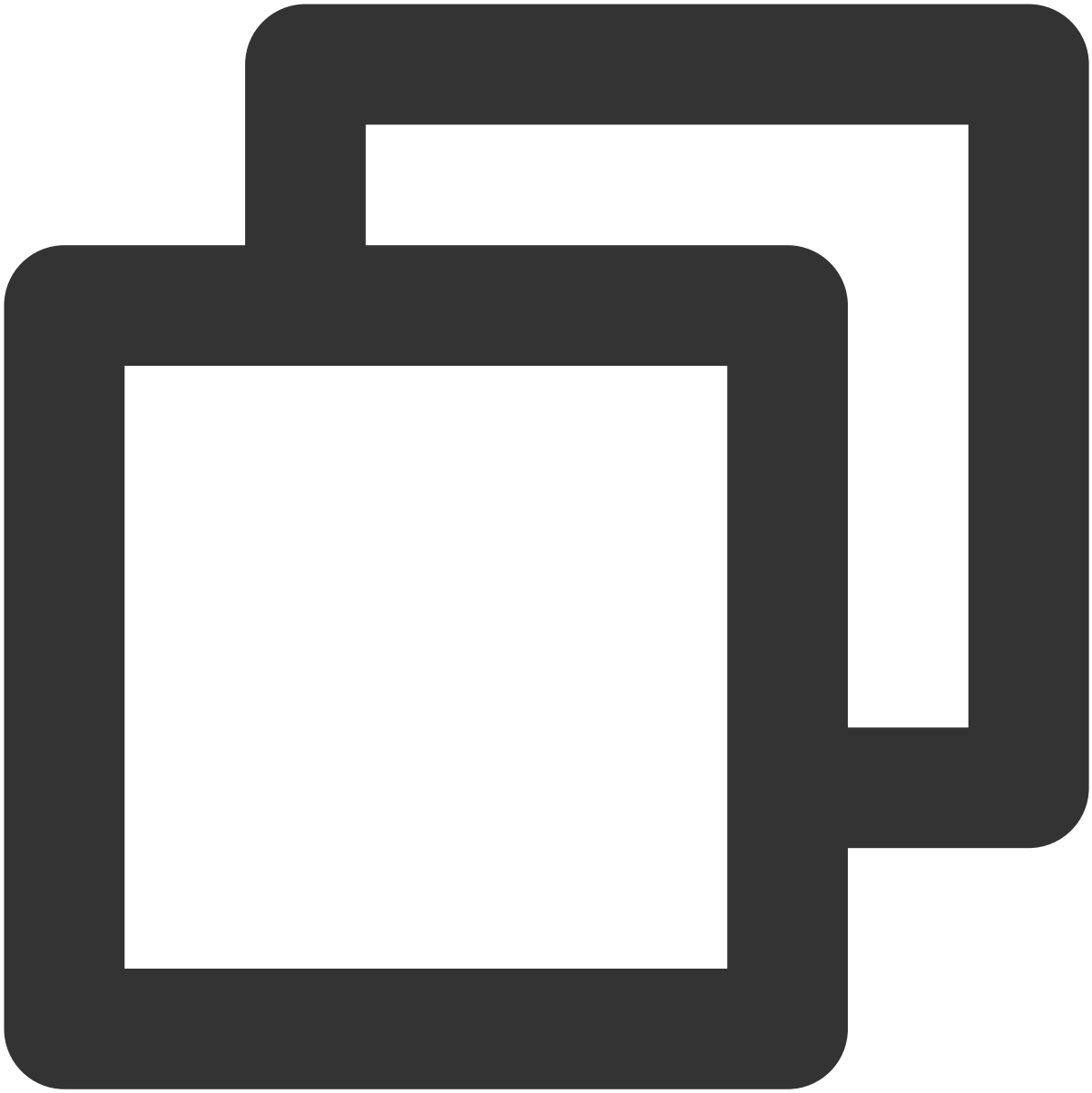
API Overview

API	Description
hangup	Ends a call.
accept	Accepts a call.
setVideoRenderParams	Set the rendering mode of video image.
setVideoEncoderParams	Set the encoding parameters of video encoder.

API Details

hangup

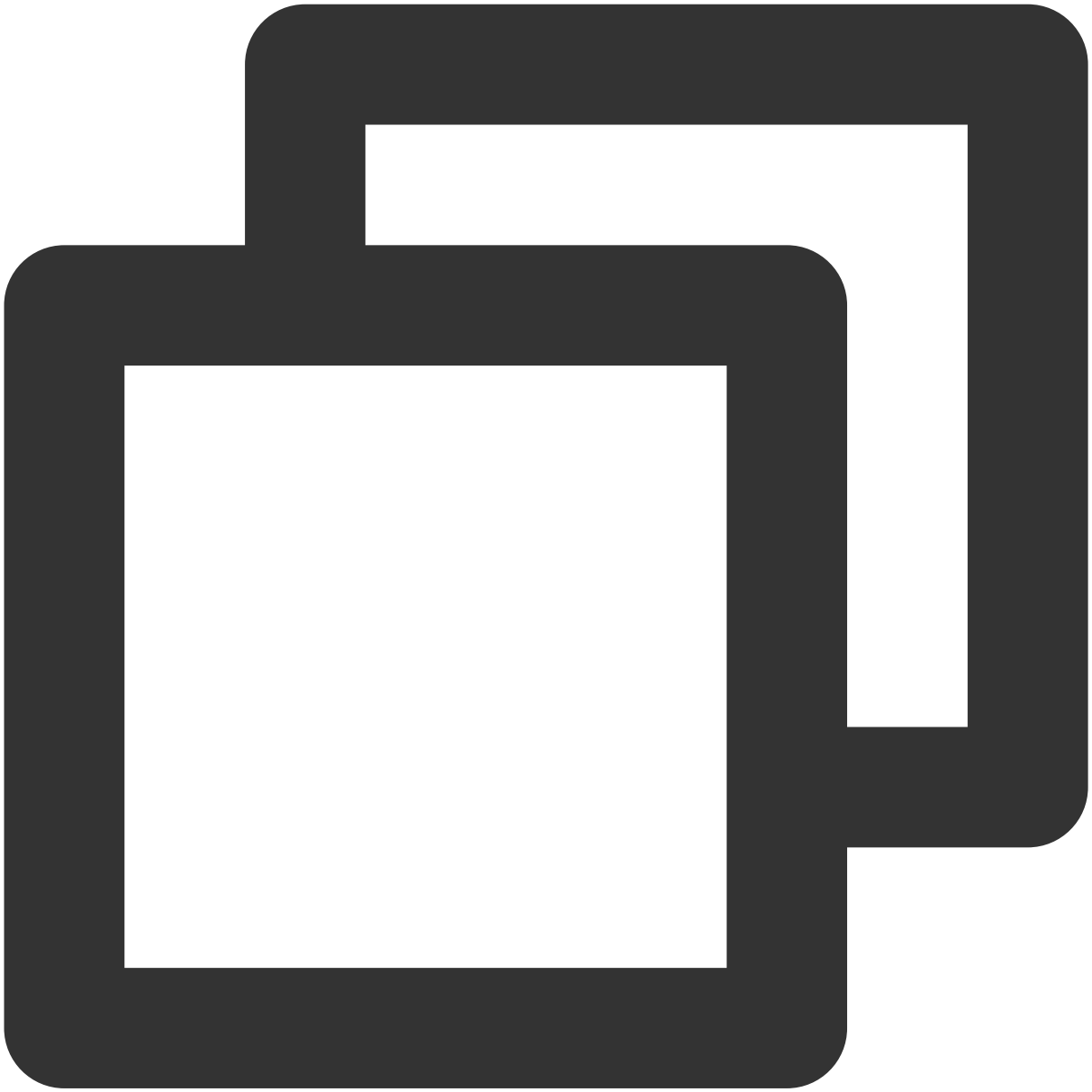
This API is used to end a call.



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
TUICallEngine.hangup();
```

accept

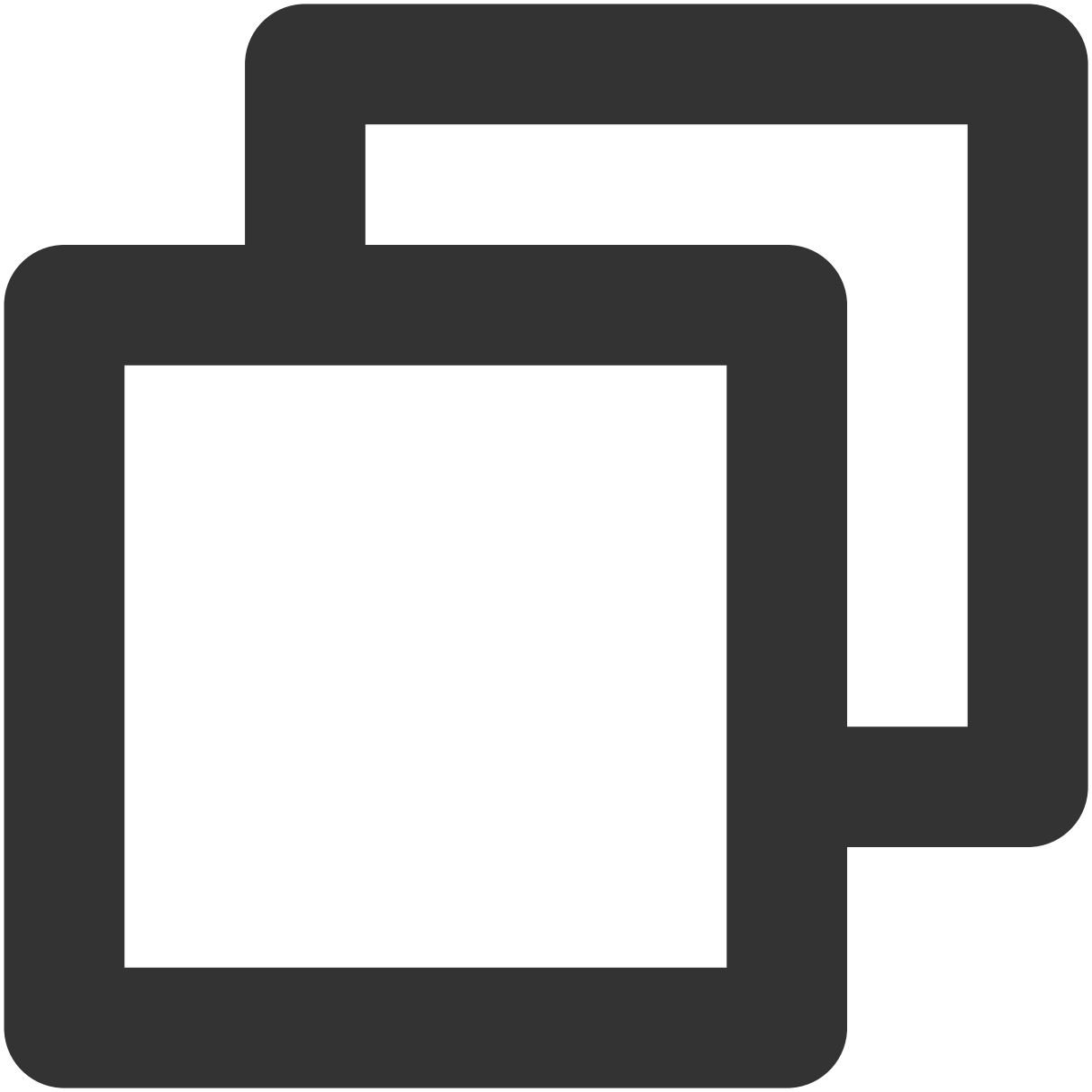
This API is used to accept a call. After receiving the `onCallReceived()` callback, you can call this API to accept the call.



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
TUICallEngine.accept();
```

setVideoRenderParams

Set the rendering mode of video image.



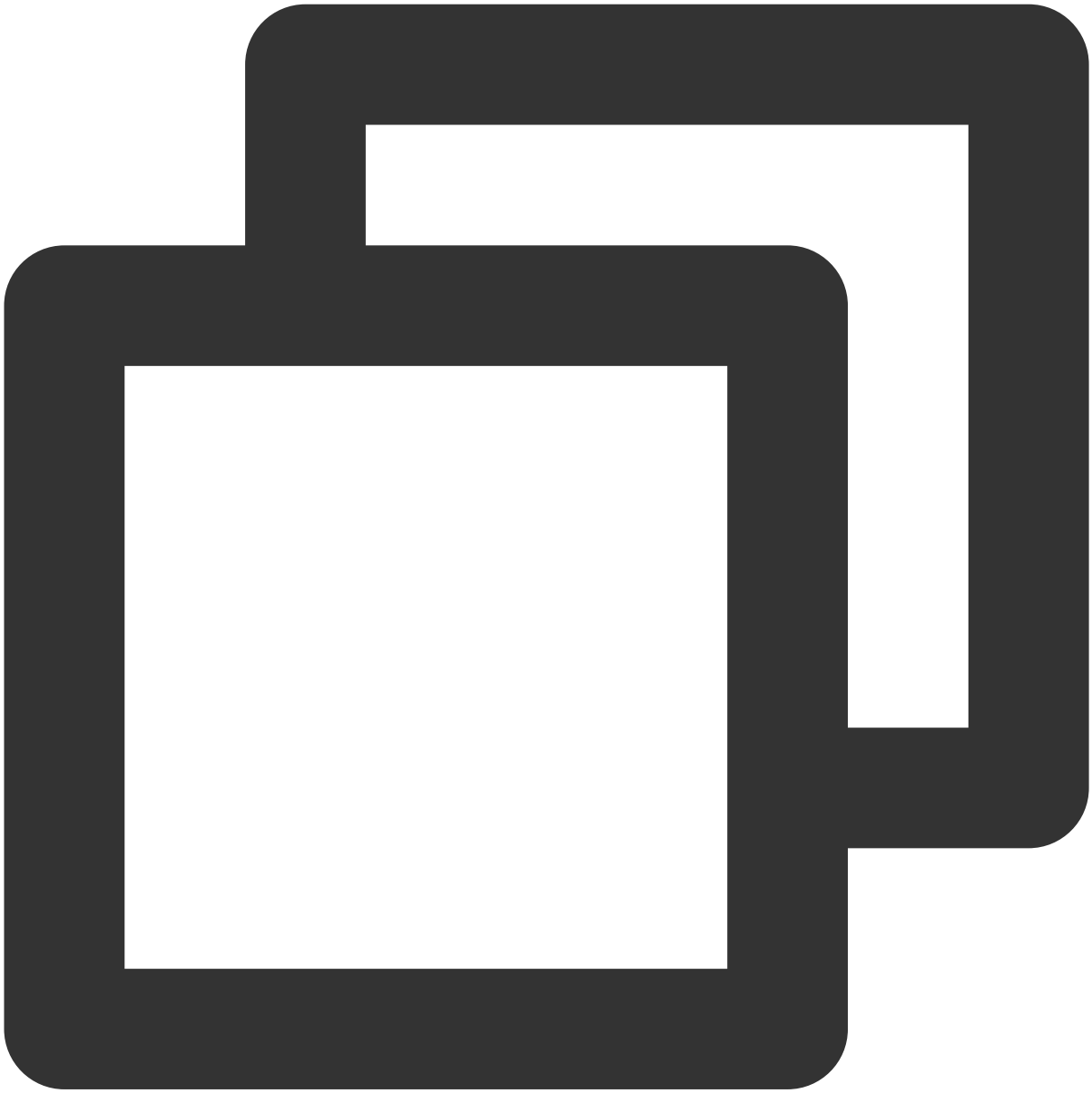
```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
const params = {
        userID: '234',
        fillMode: 0, // 0-fill mode, 1-adapter mode
        rotation: 1, // 0：Rotation_0; 1: Rotation_90; 2: Rotation_180; 3: Rotation_
};
TUICallEngine.setVideoRenderParams(params, (res) => {
        console.warn('res = ', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
userID	String	target userId
params	Object	Video frame rendering parameters, e.g. frame rotation angle, fill mode

setVideoEncoderParams

Set the encoding parameters of video encoder.
This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
const params = {
        resolution: 108,
        resolutionMode: 0, // 0--landscape, 1--portrait
};
TUICallEngine.setVideoEncoderParams(params, (res) => {
        console.warn('res = ', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
resolution	Number	video resolution 62: aspect ratio 16:9；resolution 640x360； 64: aspect ratio 4:3；resolution 960x720； 108: aspect ratio 16:9；resolution 640x360； 110: aspect ratio 16:9；resolution 960x540； 112: aspect ratio 16:9；resolution 1280x720； 114: aspect ratio 16:9；resolution 1920x1080；
resolutionMode	Number	resolution mode 0: Landscape 1: Portrait

TUICallEvent

Last updated：2024-04-30 10:43:45

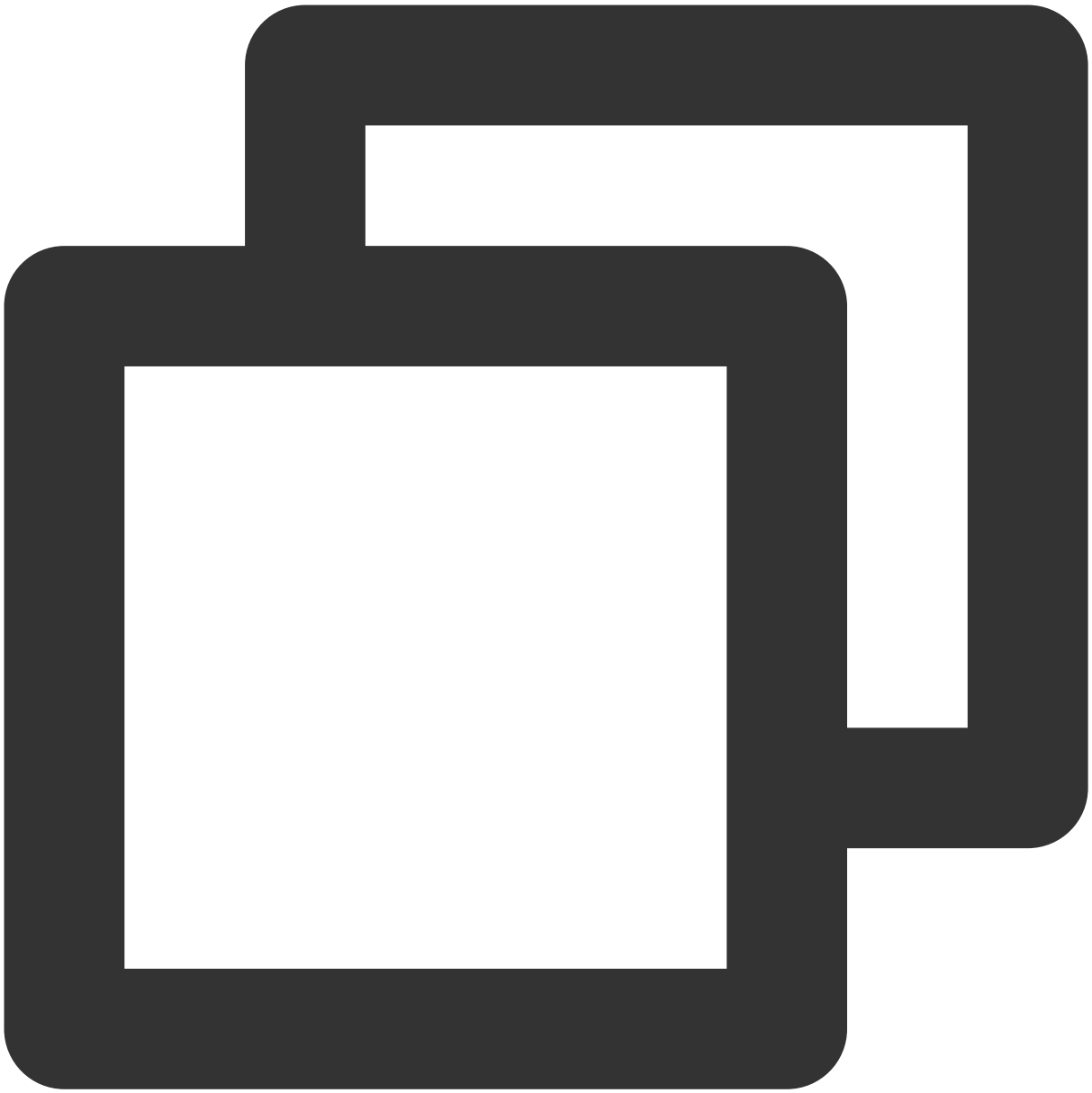
You can listen to TUICallKit's corresponding events for prompts and other interactions.

Event Overview

API	Description
onError	A call occurred during the call.
onCallReceived	A call invitation was received.
onCallCancelled	The call was canceled.
onCallBegin	The call was connected.
onCallEnd	The call ended.
onCallMediaTypeChanged	The call type changed.
onUserReject	A user declined the call.
onUserNoResponse	A user didn't respond.
onUserLineBusy	A user was busy.
onUserJoin	A user joined the call.
onUserLeave	A user left the call.
onUserVideoAvailable	Whether a user had a video stream.
onUserAudioAvailable	Whether a user had an audio stream.
onUserVoiceVolumeChanged	The volume levels of all users.
onUserNetworkQualityChanged	The network quality of all users.
onKickedOffline	The current user was kicked offline.
onUserSigExpired	The user sig is expired.

Details

Listen to events thrown by the native plugin via globalEvent.



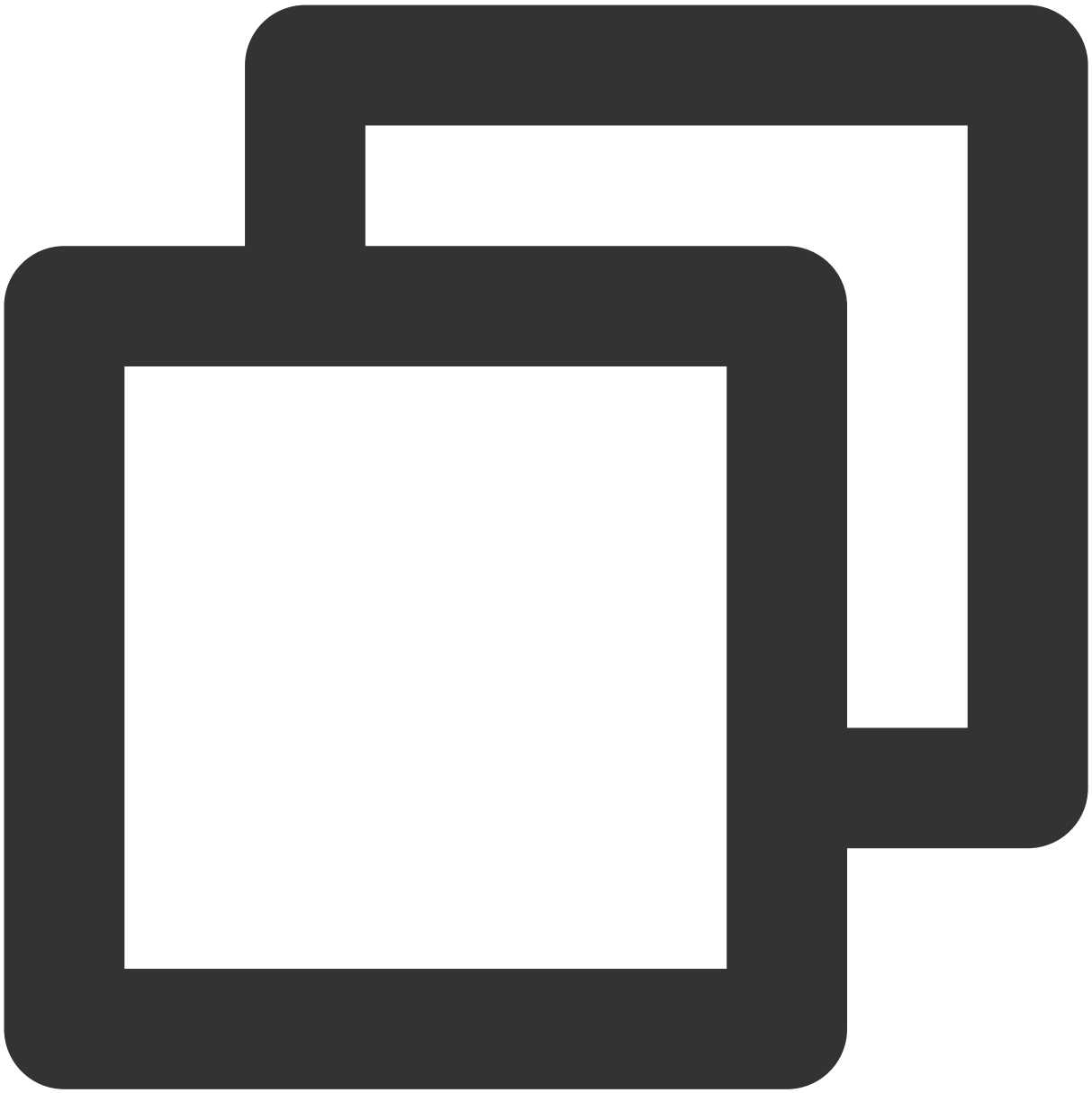
```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
```

onError

An error occurred.

Note

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.



```
TUICallEngine.addEventListener('onError', (res) => {
  console.log('onError', JSON.stringify(res));
});
```

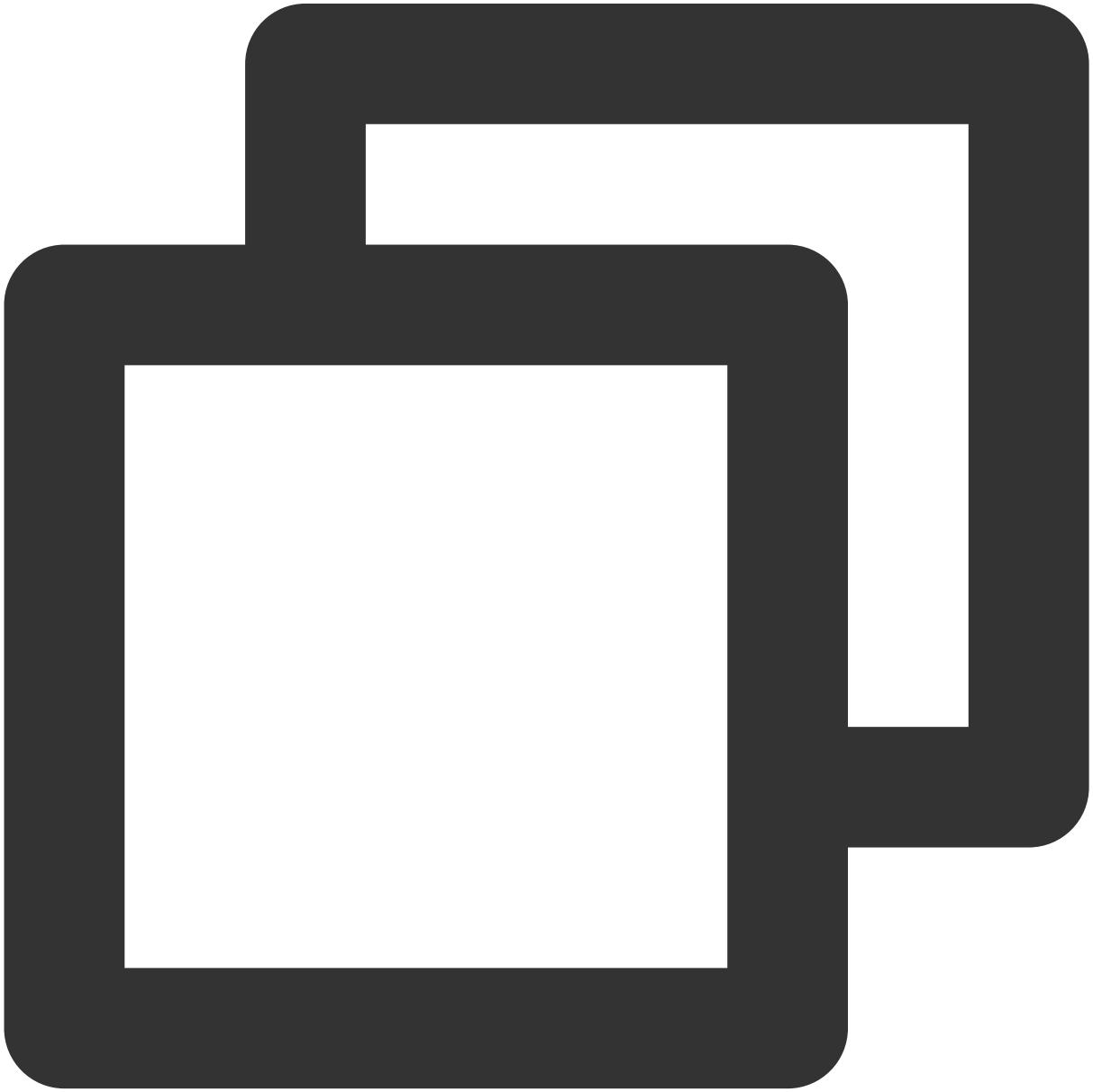
The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter

res.code	Number	The error code.
res.msg	String	The error message.

onCallReceived

A call invitation was received. This callback is received by an invitee. You can listen for this event to determine whether to display the incoming call view.



```
TUICallEngine.addEventListener('onCallReceived', (res) => {
```

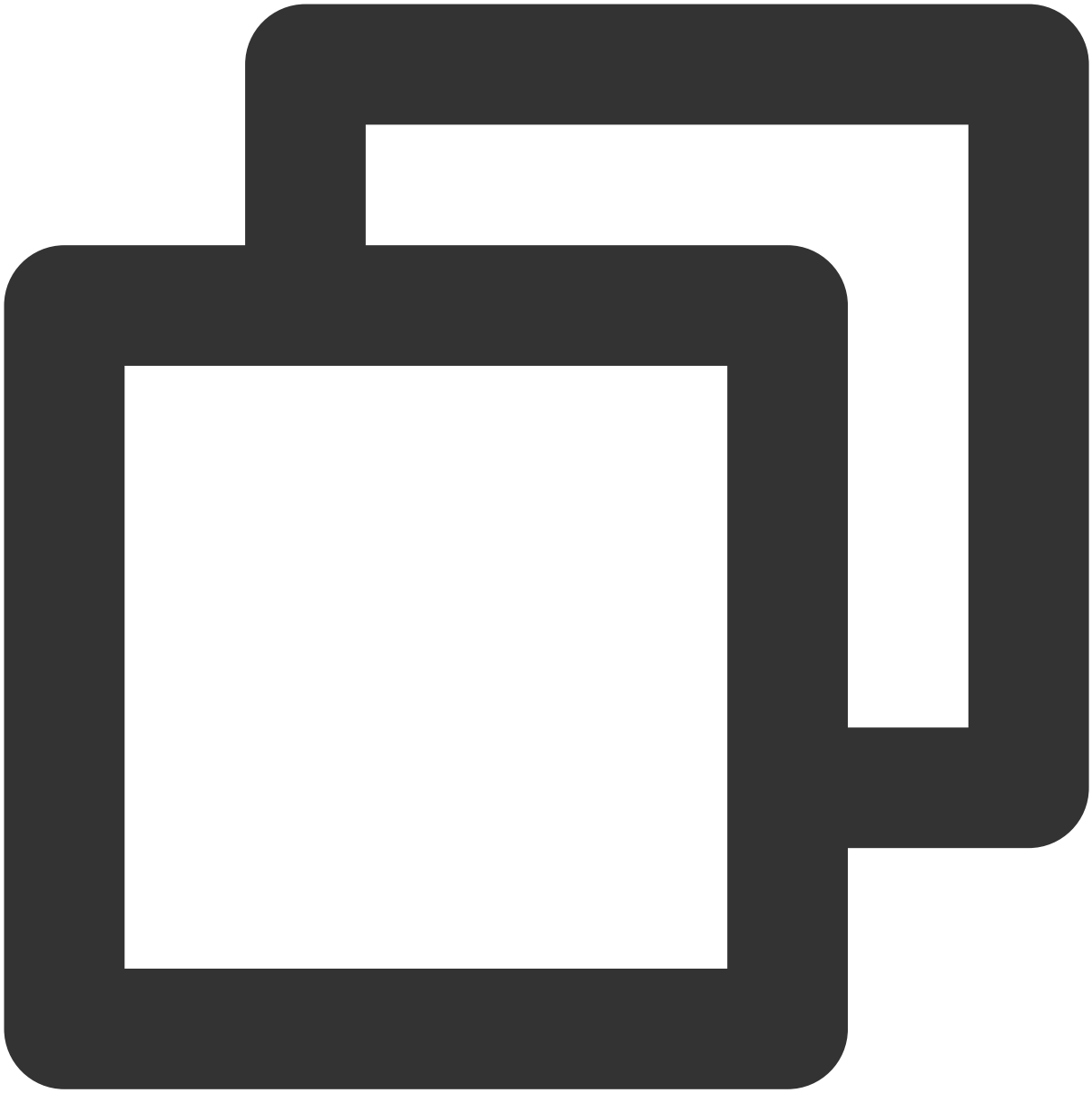
```
  console.log('onCallReceived', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.callerId	String	The user ID of the inviter.
res.calleeIdList	Array<String>	The invitee list.
res.groupId	String	The group ID.
res.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
res.userData	String	User-added extended fields.

onCallCancelled

The call was canceled by the inviter or timed out. This callback is received by an invitee. You can listen for this event to determine whether to show a missed call message.
This indicates that the call was canceled by the caller, timed out by the callee, rejected by the callee, or the callee was busy. There are multiple scenarios involved. You can listen to this event to achieve UI logic such as missed calls and resetting UI status.
Call cancellation by the caller: The caller receives the callback (userId is himself); the callee receives the callback (userId is the ID of the caller)
Callee timeout: the caller will simultaneously receive the onUserNoResponse and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)
Callee rejection: The caller will simultaneously receive the onUserReject and onCallCancelled callbacks (userId is his own ID); the callee receives the onCallCancelled callback (userId is his own ID)
Callee busy: The caller will simultaneously receive the onUserLineBusy and onCallCancelled callbacks (userId is his own ID);
Abnormal interruption: The callee failed to receive the call ， he receives this callback (userId is his own ID).



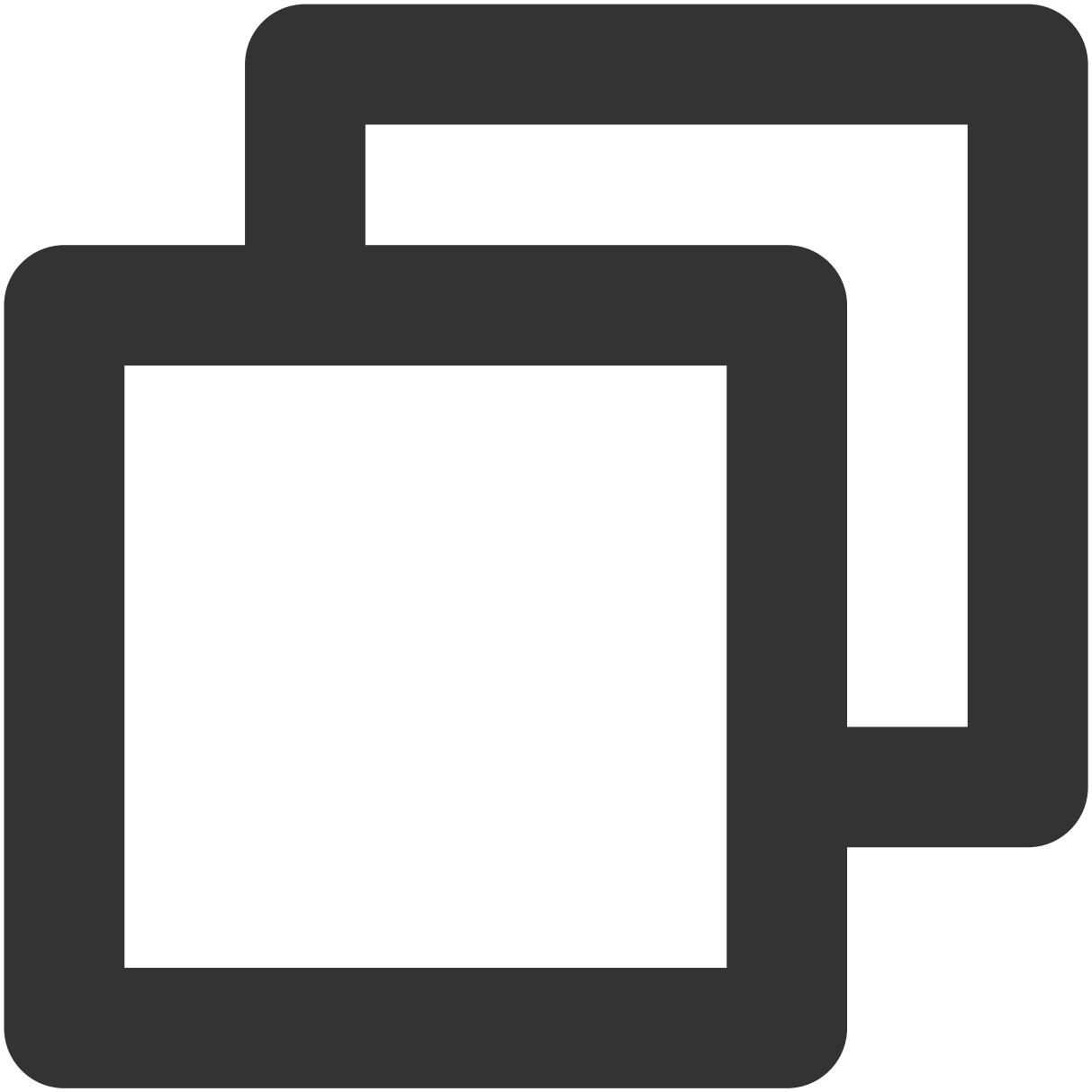
```
TUICallEngine.addEventListener('onCallCancelled', (res) => {
  console.log('onCallCancelled', res);
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID of the inviter.

onCallBegin

The call was connected. This callback is received by both the inviter and invitees. You can listen for this event to determine whether to start on-cloud recording, content moderation, or other tasks.



```
TUICallEngine.addEventListener('onCallBegin', (res) => {
  console.log('onCallBegin', JSON.stringify(res));
});
```

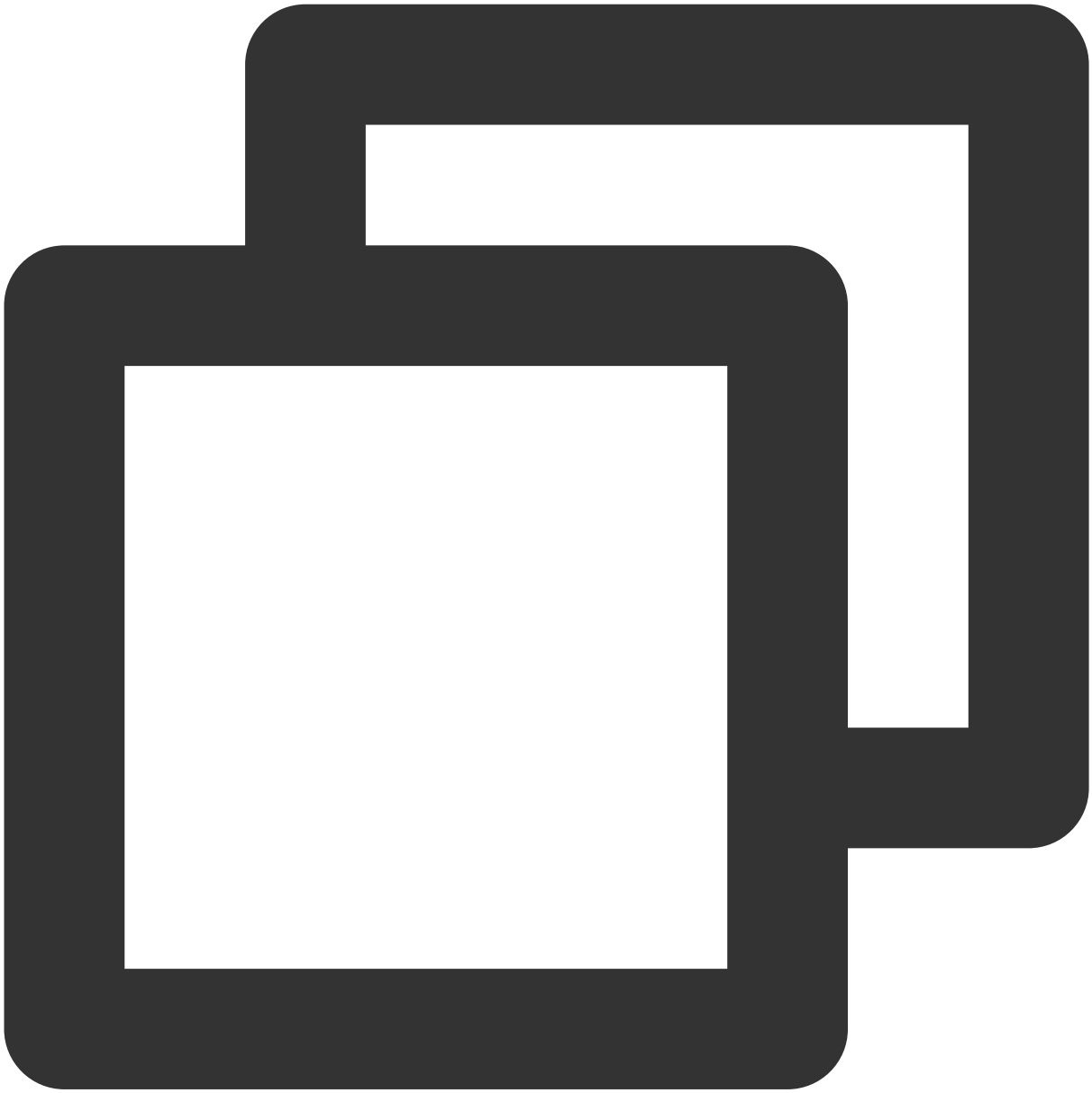
The parameters are described below:

Parameter	Type	Description

res	Object	callback parameter
res.roomID	Number	The room ID.
res.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
res.callRole	Number	The role, which can be caller or callee.

onCallEnd

The call ended. This callback is received by both the inviter and invitees. You can listen for this event to determine when to display call information such as call duration and call type, or stop on-cloud recording.



```
TUICallEngine.addEventListener('onCallEnd', (res) => {
  console.log('onCallEnd', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.roomID	Number	The room ID.

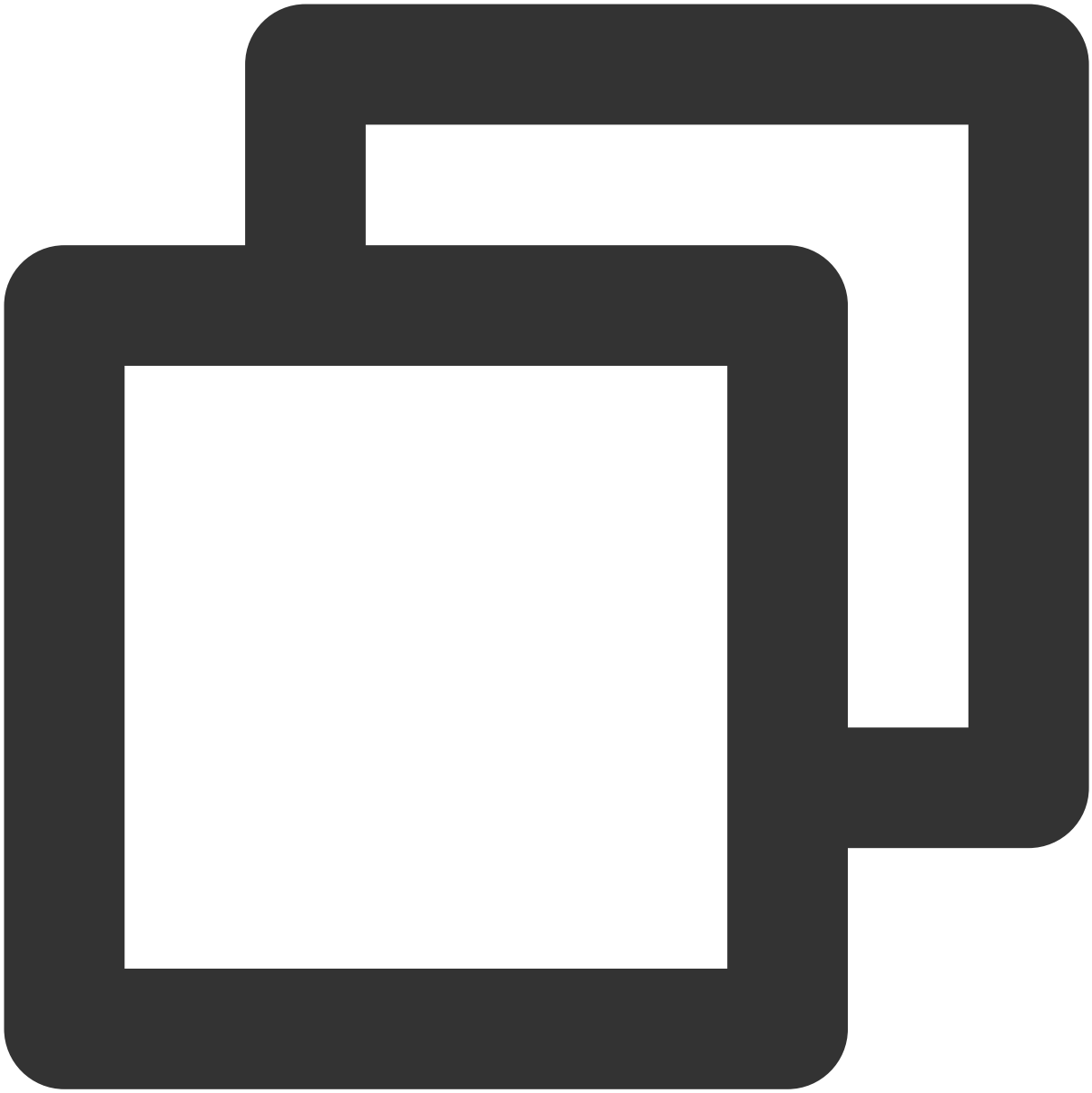
res.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
res.callRole	Number	The role, which can be caller or callee.
res.totalTime	Number	The call duration. unit: second

Note

Client-side callbacks are often lost when errors occur, for example, when the process is closed. If you need to measure the duration of a call for billing or other purposes, we recommend you use the RESTful API.

onCallMediaTypeChanged

The call type changed.



```
TUICallEngine.addEventListener('onCallMediaTypeChanged', (res) => {
  console.log('onCallMediaTypeChanged', JSON.stringify(res));
});
```

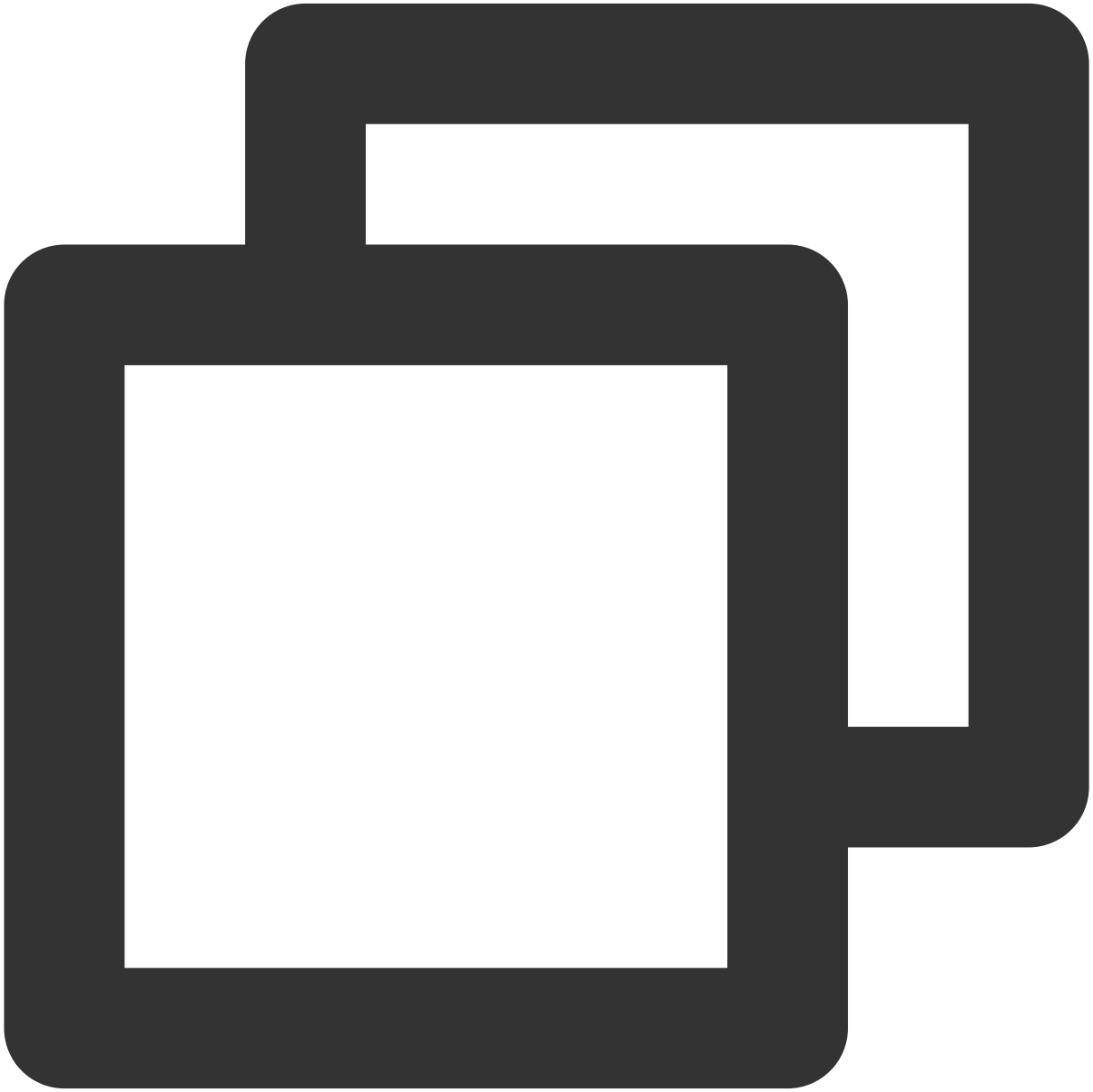
The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.oldCallMediaType	Number	The call type before the change. eg. audio call(callMediaType =

		1), video call(callMediaType = 2)
res.newCallMediaType	Number	The call type after the change. eg. audio call(callMediaType = 1), video call(callMediaType = 2)

onUserReject

The call was rejected. In a one-to-one call, only the inviter will receive this callback. In a group call, all invitees will receive this callback.



```
TUICallEngine.addEventListener('onUserReject', (res) => {
```

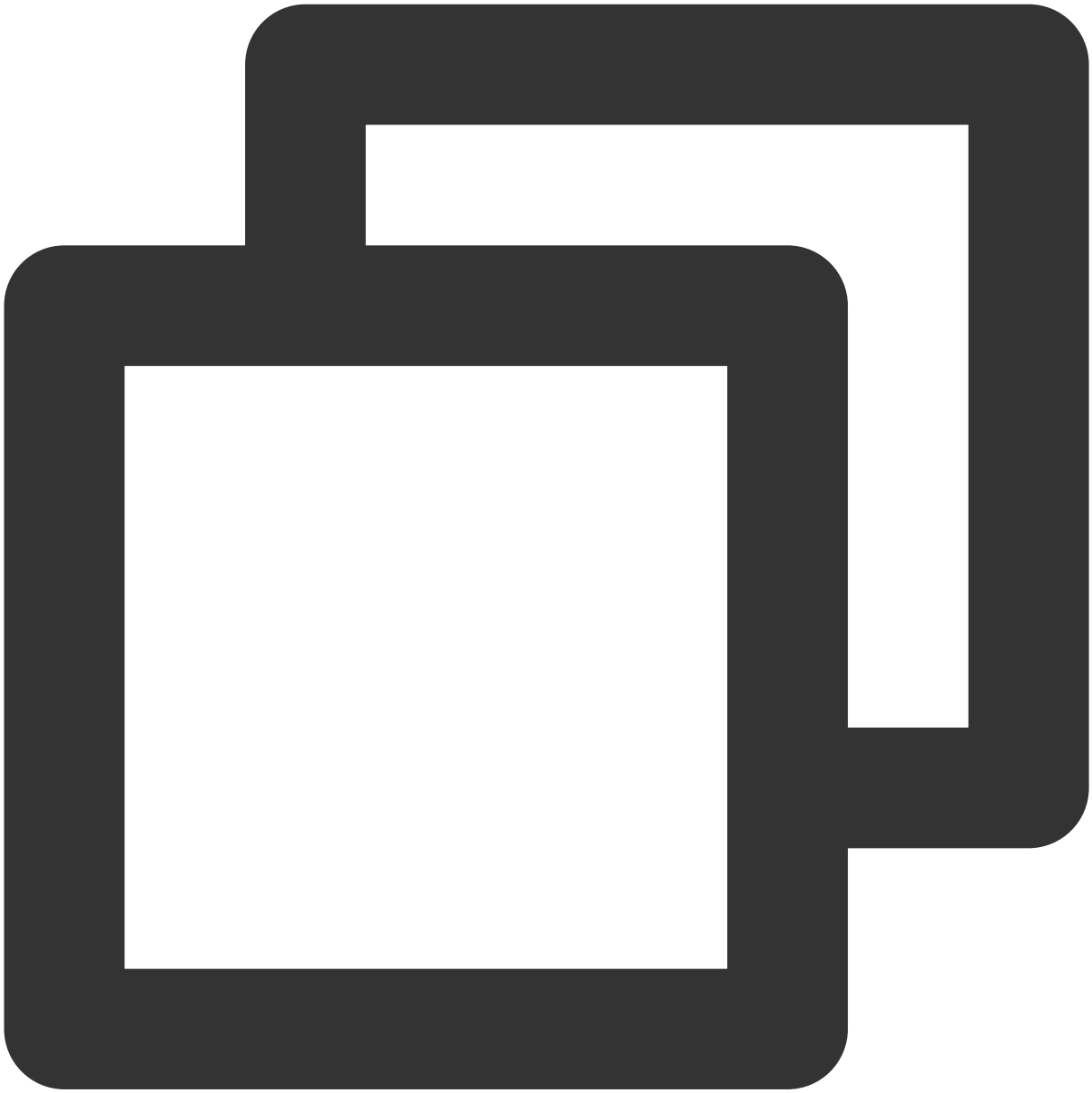
```
  console.log('onUserReject', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID of the invitee who rejected the call.

onUserNoResponse

A user did not respond.



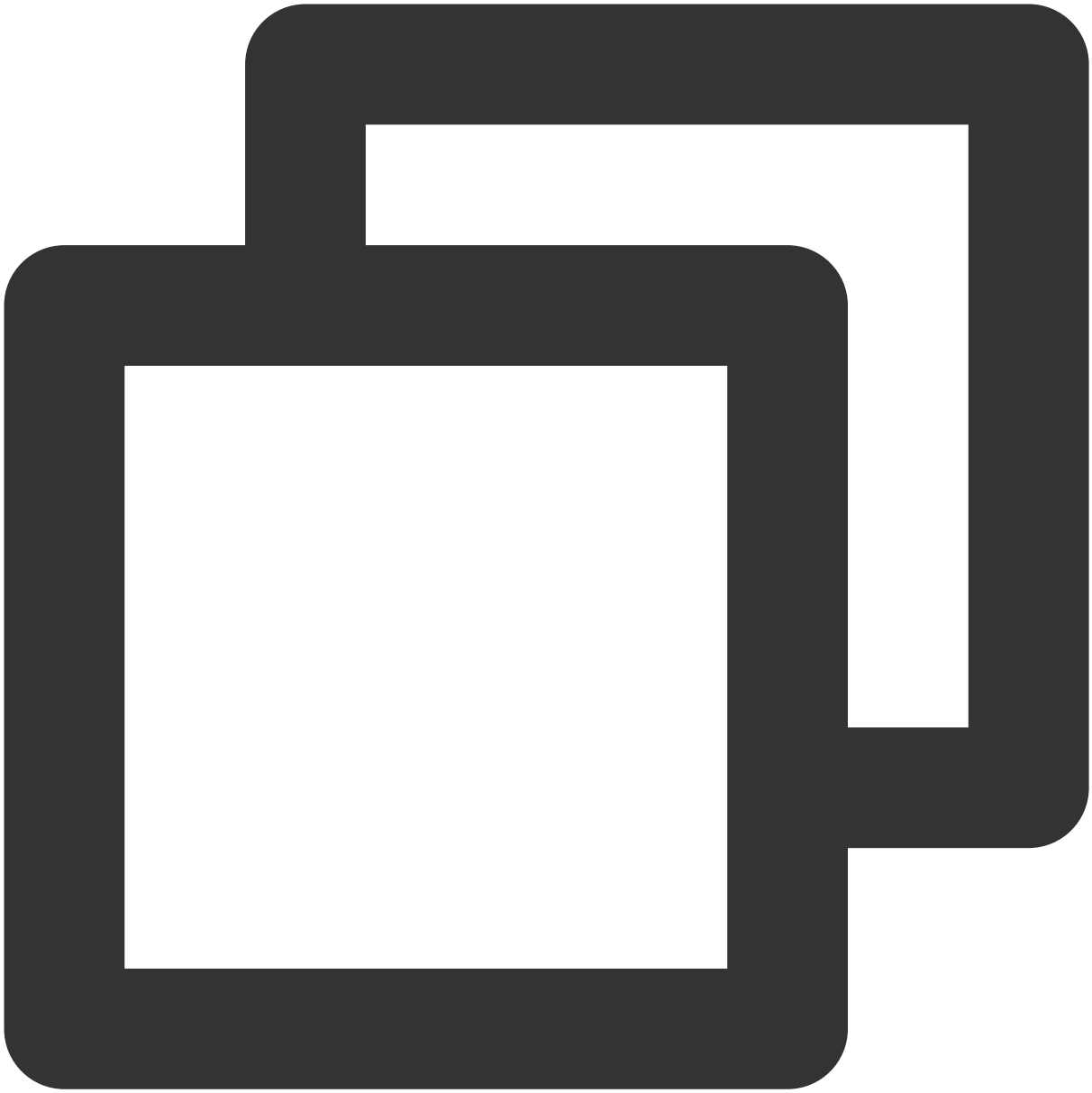
```
TUICallEngine.addEventListener('onUserNoResponse', (res) => {
  console.log('onUserNoResponse', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID of the invitee who did not answer.

onUserLineBusy

A user is busy.



```
TUICallEngine.addEventListener('onUserLineBusy', (res) => {
  console.log('onUserLineBusy', JSON.stringify(res));
});
```

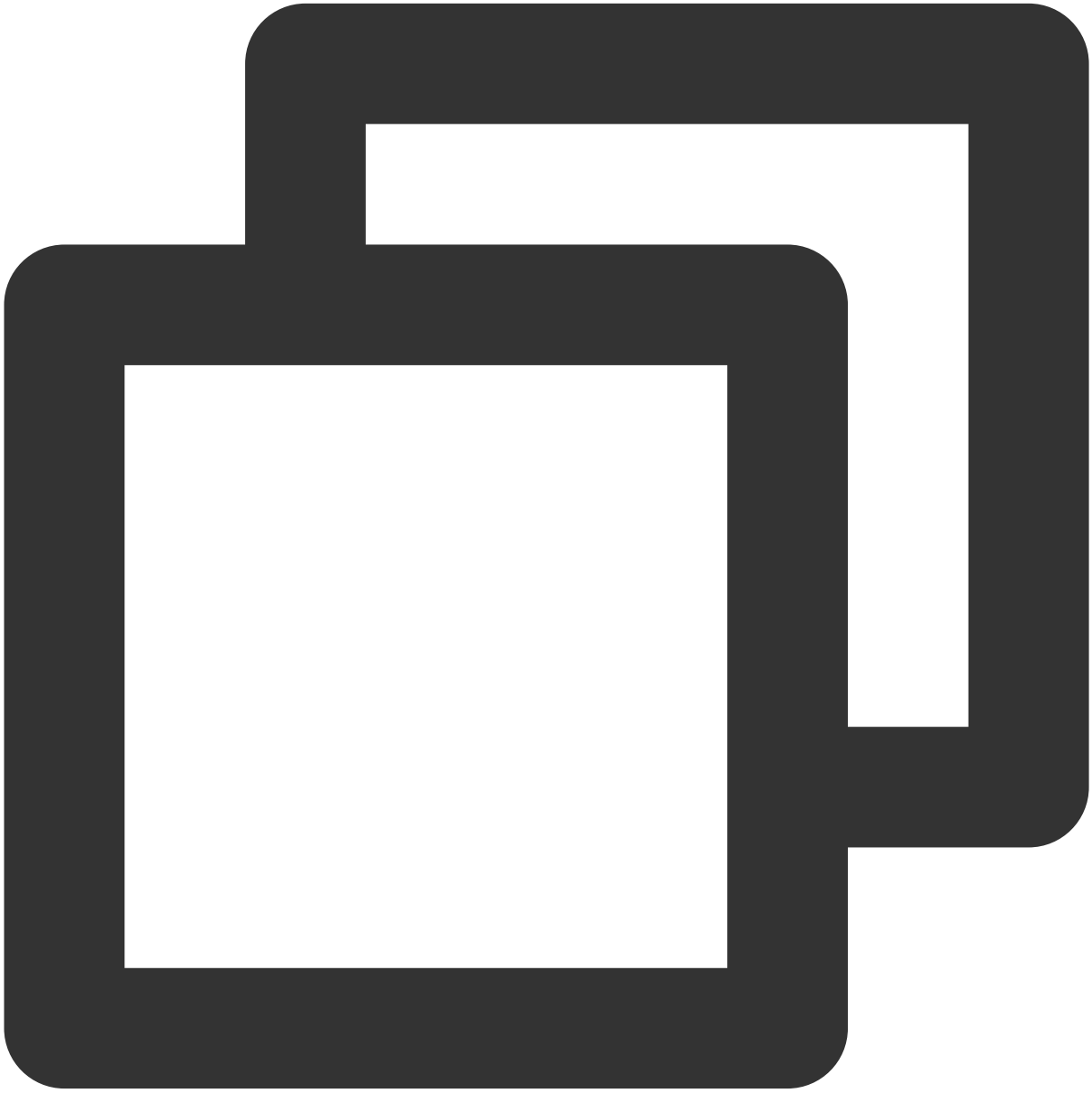
The parameters are described below:

Parameter	Type	Description

res	Object	callback parameter
res.userId	String	The user ID of the invitee who is busy.

onUserJoin

A user joined the call.



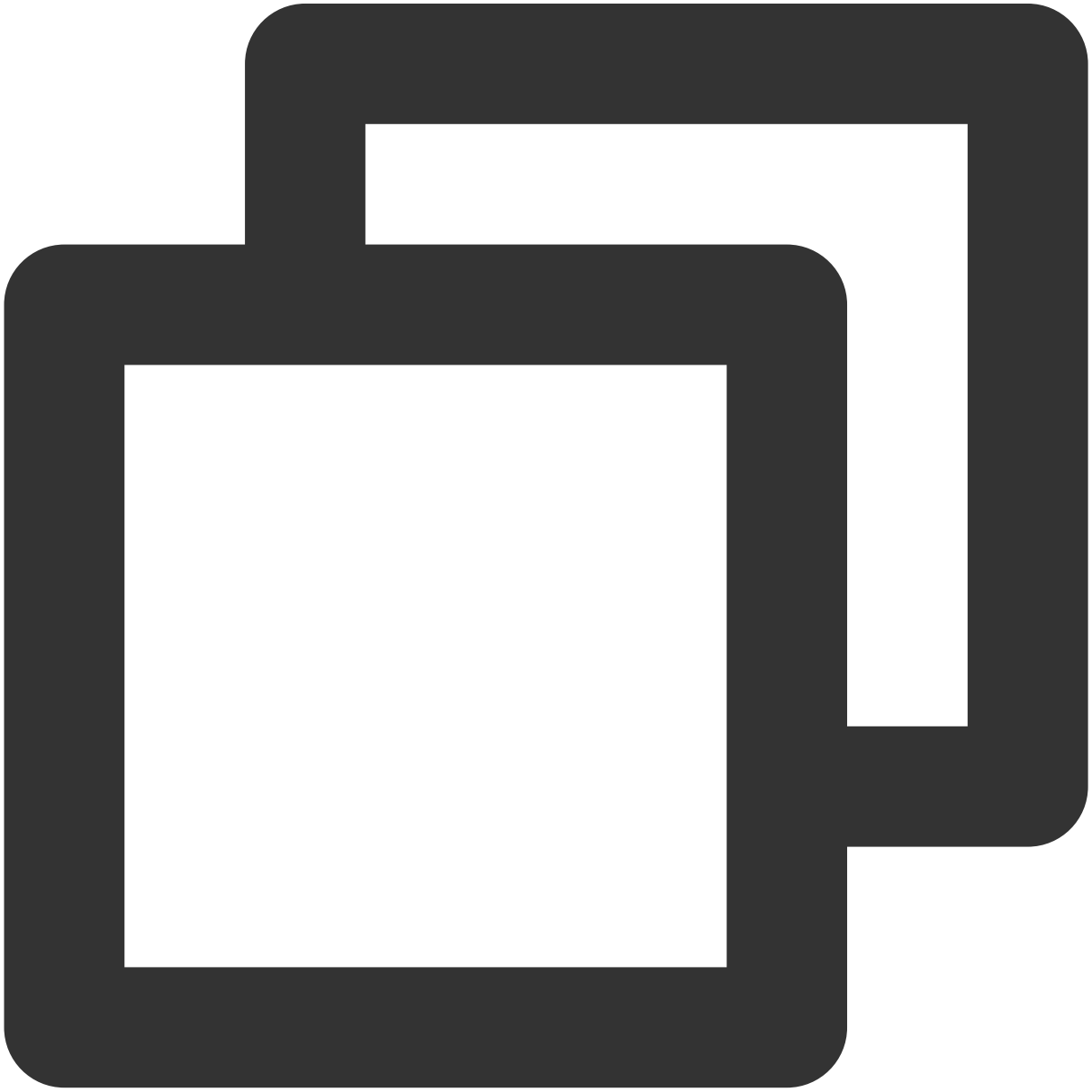
```
TUICallEngine.addEventListener('onUserJoin', (res) => {
  console.log('onUserJoin', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The ID of the user who joined the call.

onUserLeave

A user left the call.



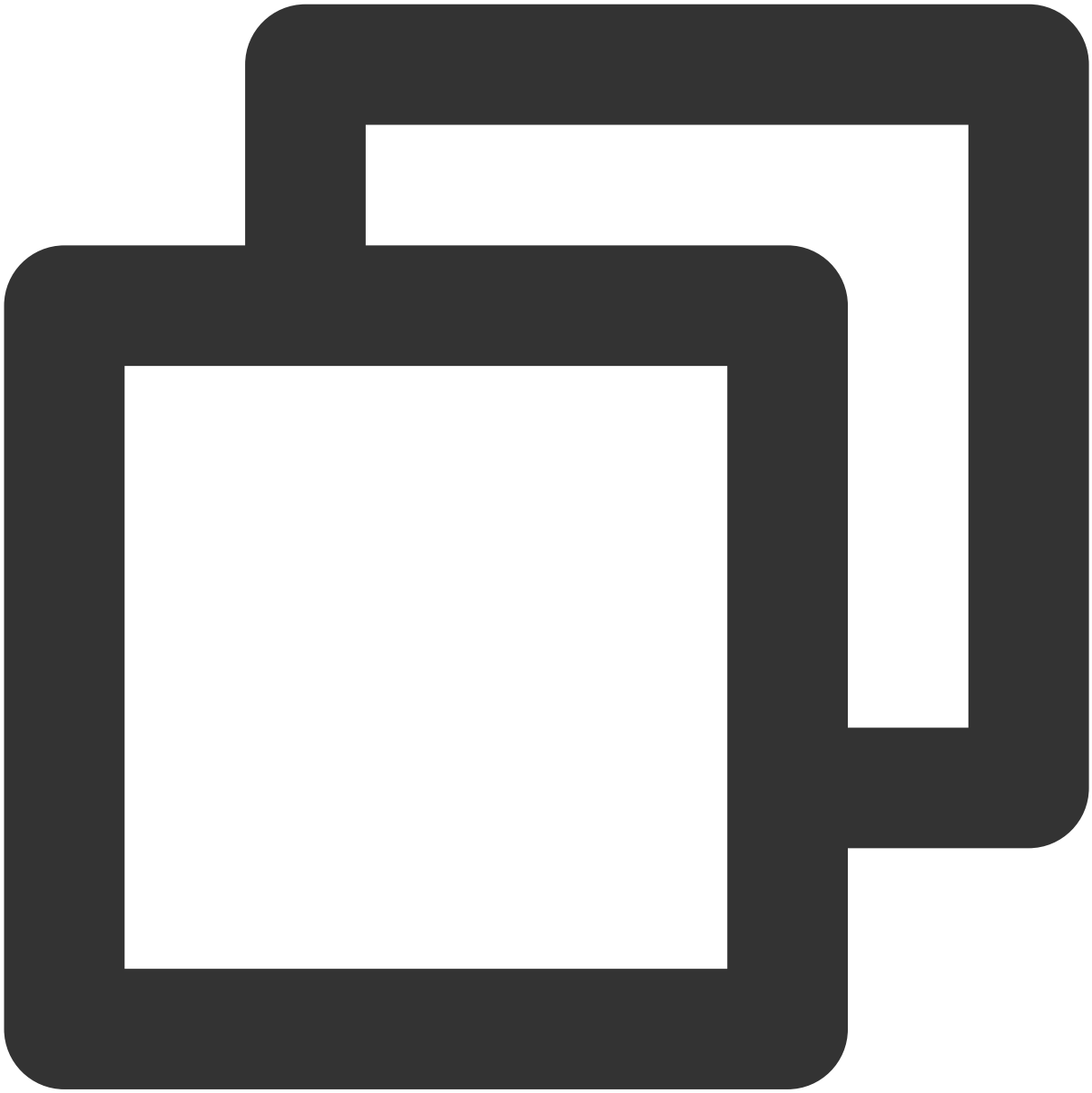
```
TUICallEngine.addEventListener('onUserLeave', (res) => {
  console.log('onUserLeave', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The ID of the user who left the call.

onUserVideoAvailable

Whether a user is sending video.



```
TUICallEngine.addEventListener('onUserVideoAvailable', (res) => {
  console.log('onUserVideoAvailable', JSON.stringify(res));
});
```

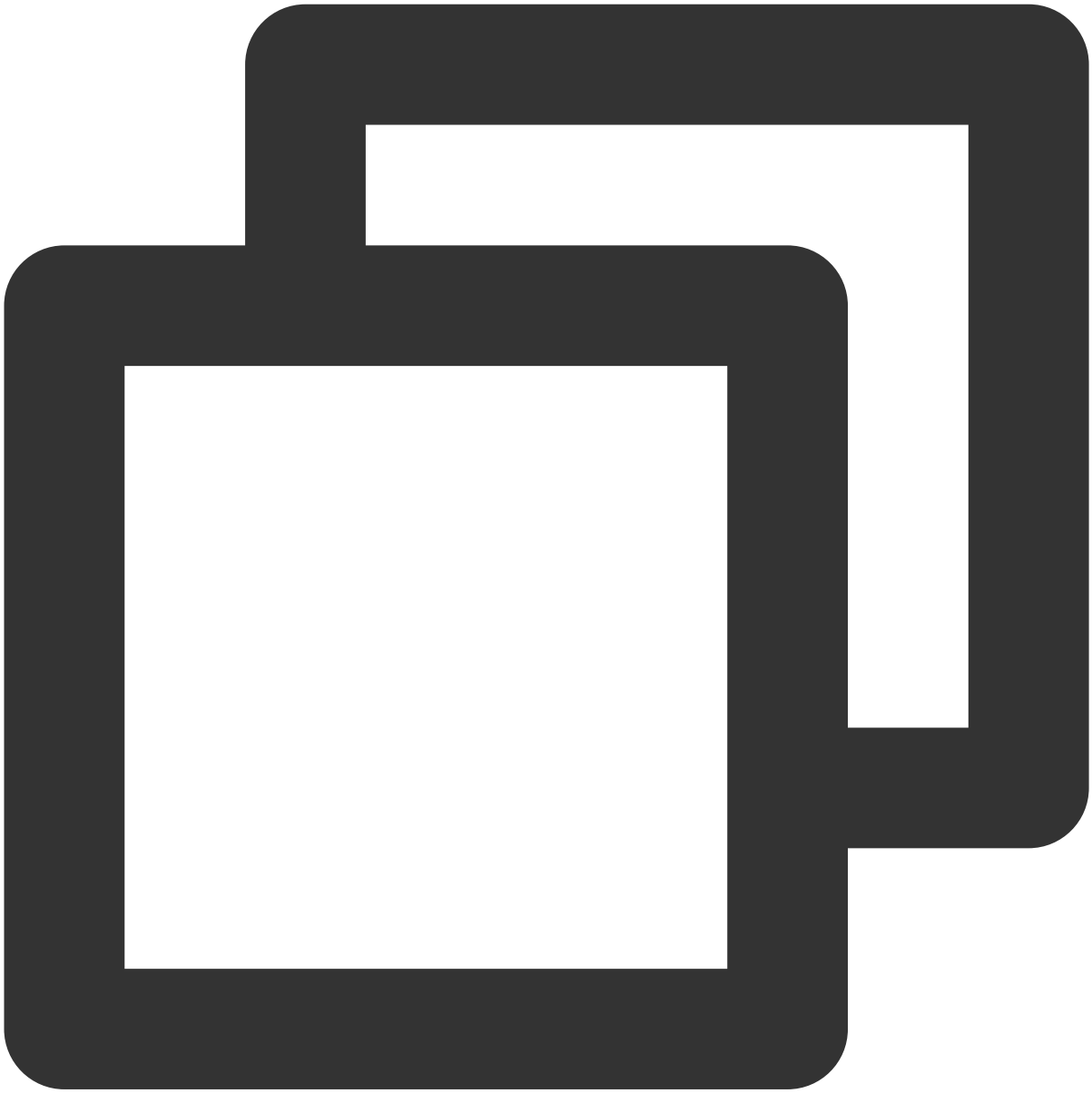
The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID.

res.isVideoAvailable	boolean	Whether the user has video.

onUserAudioAvailable

Whether a user is sending audio.



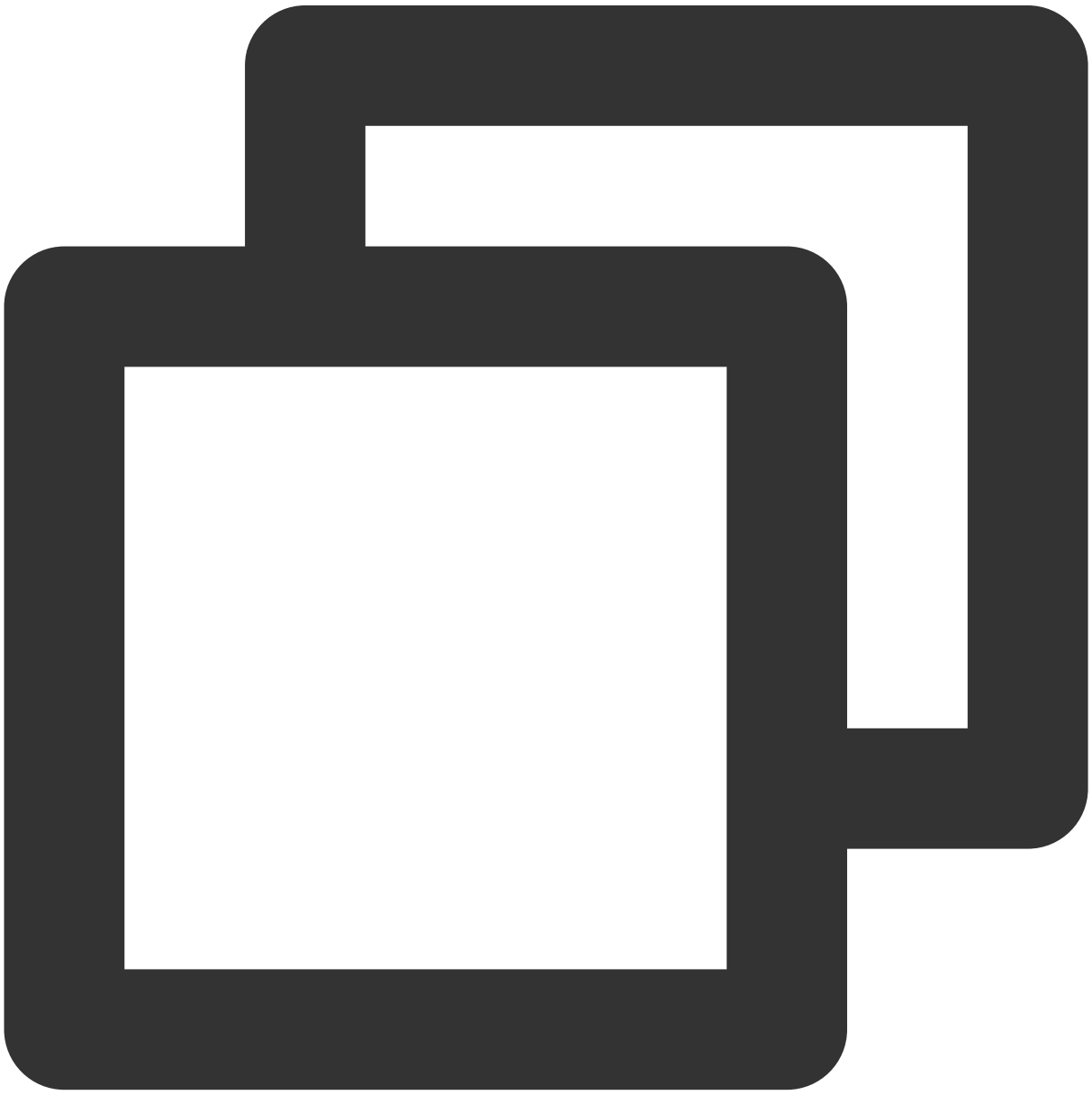
```
TUICallEngine.addEventListener('onUserAudioAvailable', (res) => {
  console.log('onUserAudioAvailable', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.userId	String	The user ID.
res.isAudioAvailable	boolean	Whether the user has audio.

onUserVoiceVolumeChanged

The volumes of all users.



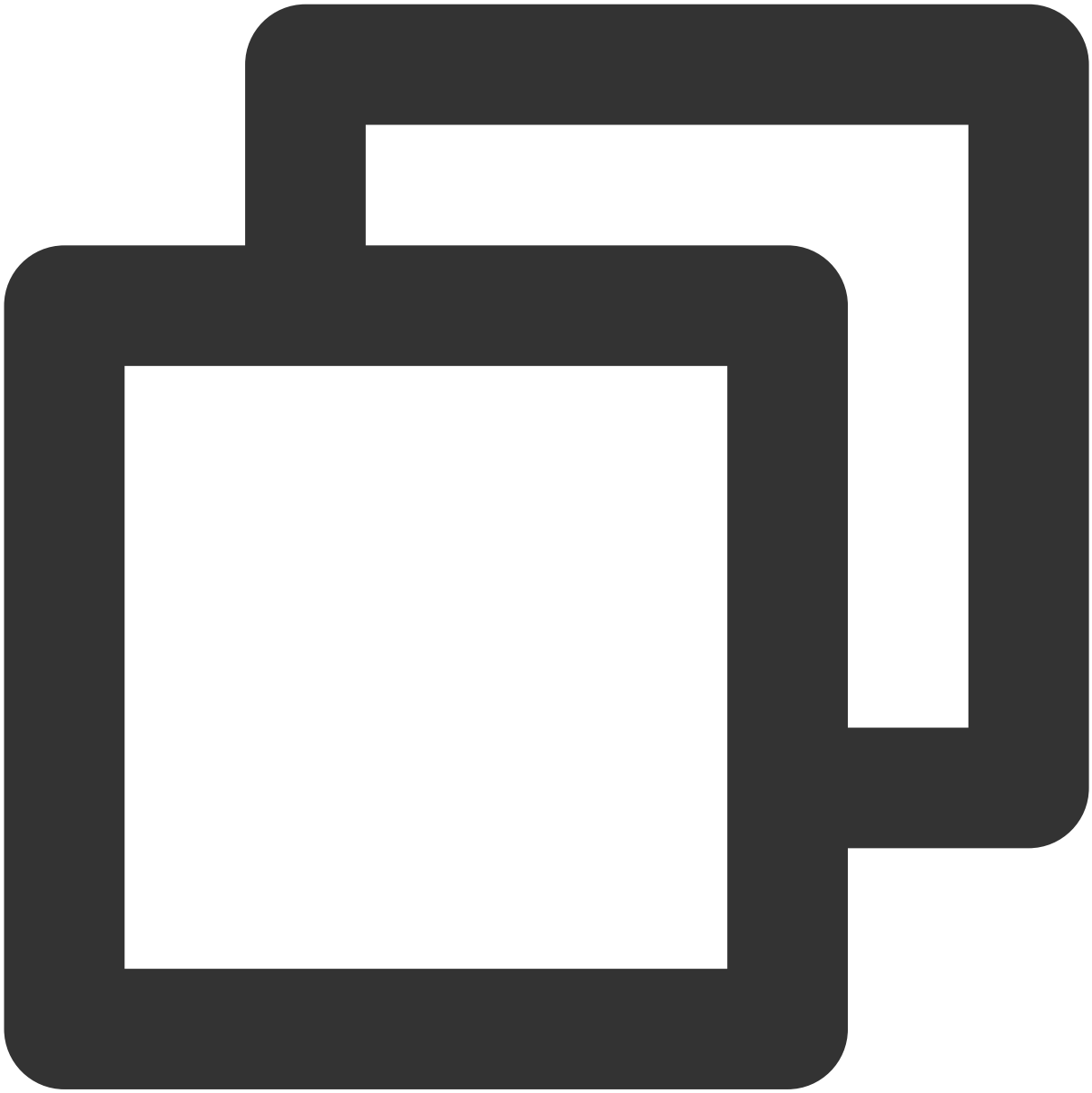
```
TUICallEngine.addEventListener('onUserVoiceVolumeChanged', (res) => {
  console.log('onUserVoiceVolumeChanged', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.volumeMap	Map<String, Integer>	The volume table, which includes the volume of each user (userId). Value range: 0-100.

onUserNetworkQualityChanged

The network quality of all users.



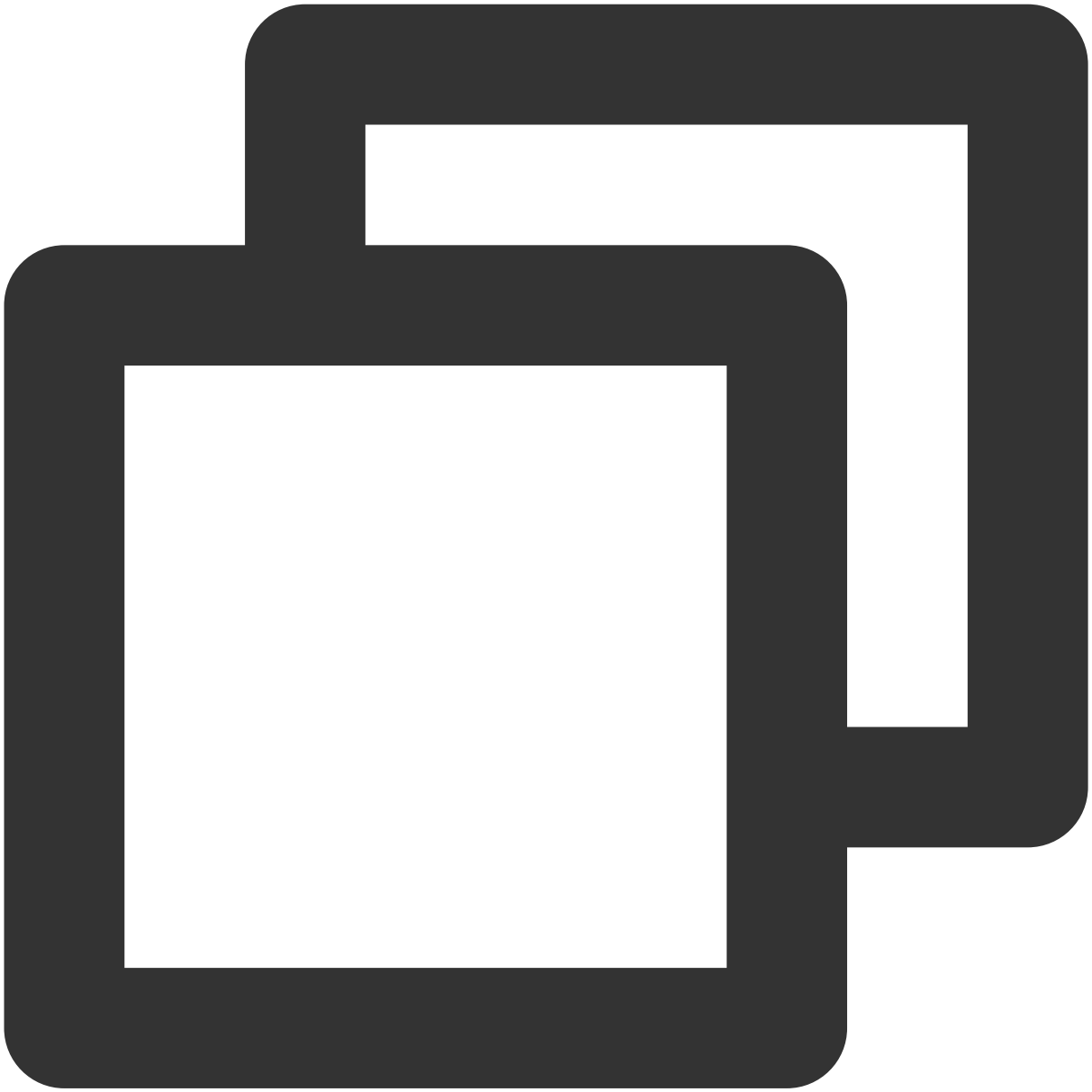
```
TUICallEngine.addEventListener('onUserNetworkQualityChanged', (res) => {
  console.log('onUserNetworkQualityChanged', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
res	Object	callback parameter
res.networkQualityList	List	The current network conditions for all users (userId).

onKickedOffline

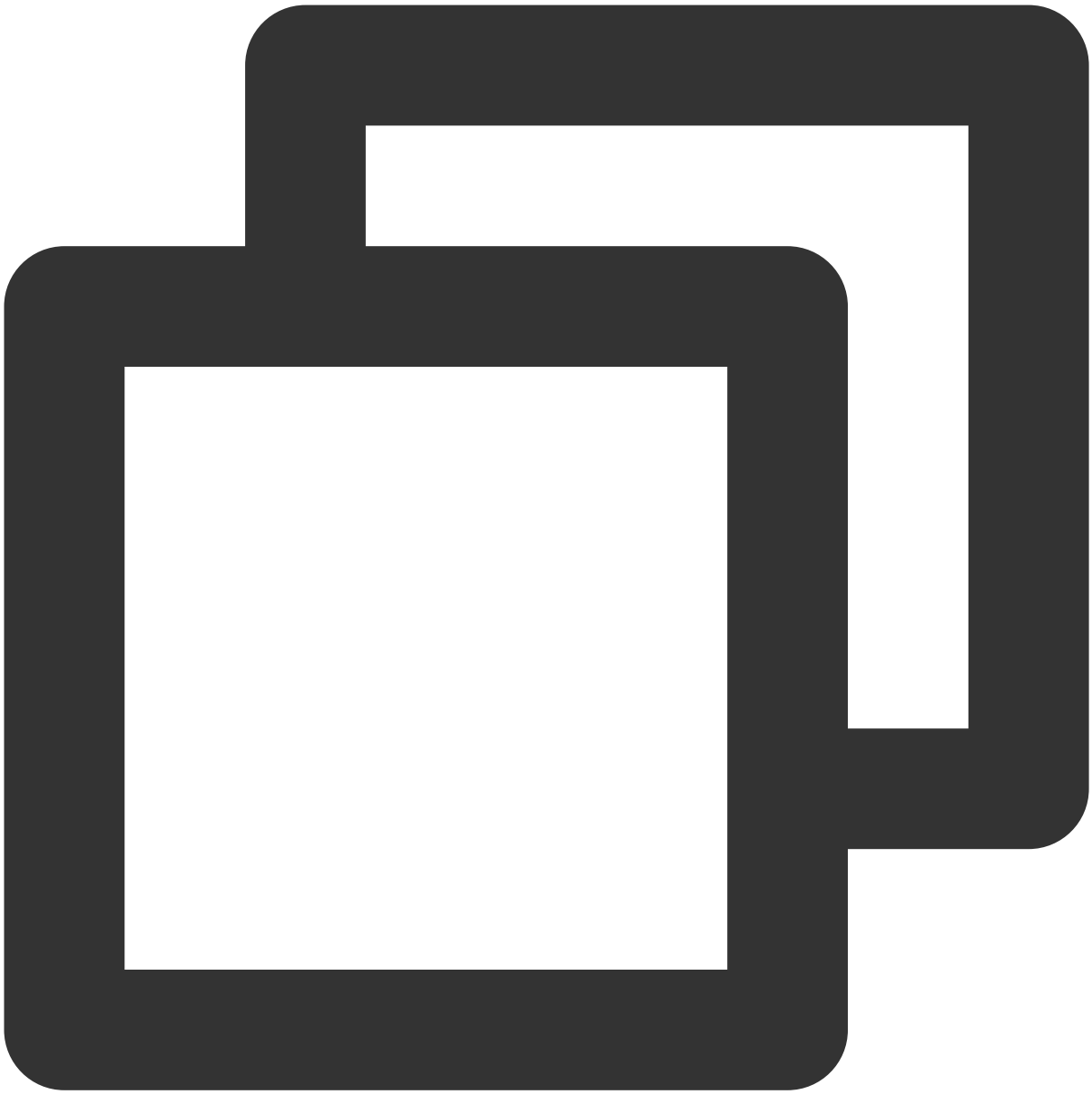
The current user was kicked offline：At this time, you can prompt the user with a UI message and then invoke `login` again.



```
TUICallEngine.addEventListener('onKickedOffline', (res) => {
  console.log('onKickedOffline', JSON.stringify(res));
});
```

onUserSigExpired

The user sig is expired：At this time, you need to generate a new `userSig` , and then invoke `login` again.



```
TUICallEngine.addEventListener('onUserSigExpired', (res) => {
  console.log('onUserSigExpired', JSON.stringify(res));
});
```

ErrorCode(TUICallKit)

Last updated：2024-07-23 18:11:32

Notify users of warnings and errors that occur during audio and video calls.

TUICallDefine Error Code

Definition	Value	Description
ERROR_PACKAGE_NOT_PURCHASED	-1001	You do not have TUICallKit package, please open the free experience in the console or purchase the official package.
ERROR_PACKAGE_NOT_SUPPORTED	-1002	The package you purchased does not support this ability. You can refer to console to purchase Upgrade package.
ERROR_TIM_VERSION_OUTDATED	-1003	The Chat SDK version is too low, please upgrade the Chat SDK version to ≥ 6.6; Find and modify in the build.gradle file.："com.tencent.imsdk:imsdk-plus:7.1.3925"
ERROR_PERMISSION_DENIED	-1101	Failed to obtain permission. The audio/video permission is not authorized. Check if the device permission is enabled.
ERROR_GET_DEVICE_LIST_FAIL	-1102	Failed to get the device list (only supported on web platform).
ERROR_INIT_FAIL	-1201	The init method has not been called for initialization. The TUICallEngine API should be used after initialization.
ERROR_PARAM_INVALID	-1202	param is invalid.
ERROR_REQUEST_REFUSED	-1203	The current status can't use this function.
ERROR_REQUEST_REPEATED	-1204	The current status is waiting/accept, please do not call it repeatedly.
ERROR_SCENE_NOT_SUPPORTED	-1205	The current calling scene does not support this feature.
ERROR_SIGNALING_SEND_FAIL	-1406	Failed to send signaling. You can check the specific

		error message in the callback of the method.

Chat Error Code

Video and audio calls use Tencent Cloud's Chat SDK as the basic service for communication, such as the core logic of call signaling and busy signaling. Common error codes are as follows:

Error Code	Description
6014	You have not logged in to the Chat SDK or have been forcibly logged out. Log in to the Chat SDK first and try again after a successful callback. To check whether you are online, use TIMManager getLoginUser.
6017	Invalid parameter. Check the error information to locate the invalid parameter.
6206	UserSig has expired. Get a new valid UserSig and log in again. For more information about how to get a UserSig, see Generating UserSig.
7013	The current package does not support this API. Please upgrade to the Flagship Edition package.
8010	The signaling request ID is invalid or has been processed.

Explanation：
More Chat SDK error codes are available at：Chat Error Code

TRTC Error Code

Video and audio calls use Tencent Cloud's Chat SDK as the basic service for calling, such as the core logic of switching camera and microphone on or off. Common error codes are as follows:

Enum	Value	Description
ERR_CAMERA_START_FAIL	-1301	Failed to turn the camera on. This may occur when there is a problem with the camera configuration program (driver) on Windows or macOS. Disable and reenable the camera, restart the camera, or update the configuration program.
ERR_CAMERA_NOT_AUTHORIZED	-1314	No permission to access to the camera. This usually occurs on mobile devices and may be because the user denied access.

ERR_CAMERA_SET_PARAM_FAIL	-1315	Incorrect camera parameter settings (unsupported values or others).
ERR_CAMERA_OCCUPY	-1316	The camera is being used. Try another camera.
ERR_MIC_START_FAIL	-1302	Failed to turn the mic on. This may occur when there is a problem with the mic configuration program (driver) on Windows or macOS. Disable and reenable the mic, restart the mic, or update the configuration program.
ERR_MIC_NOT_AUTHORIZED	-1317	No permission to access to the mic. This usually occurs on mobile devices and may be because the user denied access.
ERR_MIC_SET_PARAM_FAIL	-1318	Failed to set mic parameters.
ERR_MIC_OCCUPY	-1319	The mic is being used. The mic cannot be turned on when, for example, the user is having a call on the mobile device.
ERR_TRTC_ENTER_ROOM_FAILED	-3301	Failed to enter the room. For the reason, refer to the error message for -3301.

Explanation：

More TRTC SDK error codes are available at：TRTC Error Code

Release Notes (TUICallKit)
Web

Last updated：2024-08-09 22:25:01

Note:

TUICallKit Vue3 Github version address: Github TUICallKit Web.

Version 3.3.3 @2024.08.06

New features

Added feature to turn on/off the speaker .

Feature Optimization

react: Optimized button loading effect.

Optimized the creation and termination of ringtone instances.

Optimized the interaction of flipping the camera and virtual background button after turning off the camera.

Optimized the error content when directly calling call/groupCall API before the user init .

Optimized the warning phenomenon of setVideoResolution in the console when the user has not init and introduced the TUICallKit component on the page.

Bug Fixes

react: Fixed the spacing issue of the Virtual Background button.

react: Fixed the issue where the nickname was not truncated.

Version 3.3.2 @2024.07.12

New features

Added support for showing and hiding the invite others button.

Prompt when the network status is poor.

Bug Fixes

After selecting the device, redial. The selected device in the device list does not match the current device.

When the device list is empty, do not display the device selection list.

Fix the known issues of midway joining.

Version 3.3.1 @2024.06.25

Bug Fixes

Fix some issues with midway joining feature .
Delete the comments for debug files.

Version 3.3.0 @2024.06.14

New features

Added support for Custom Definition window size display and setting camera initial state feature.
Vue&React: Added support for passing in Custom Definition string Room Number.
React: Added react TUICallKit Virtual Background feature.

Feature Optimization

Extended offlinePushInfo parameters to support Offline Push Sound Settings and other features.

Bug Fixes

Fixed the issue where a call exception occurs immediately after a call is accepted and then timed out.
Fix the issue with group call, caller waiting, answer page, remote user, and nickname display.

Feature Optimization

UI Customization:Add Interface Log Reporting

Version 3.2.9 @2024.05.29

Feature Optimization

UI Customized Interface adds Log Reporting.

Version 3.2.8 @2024.05.27

Bug Fixes

Fixed SDK Import ref Path Error issue.

Version 3.2.7 @2024.05.17

New features

New Feature UI component for joining in group.
New Feat UI interface, supporting setting of call background and button hiding.
Adjusted parameter validation when initiating a call, supports string room numbers.

Version 3.2.4 @2024.05.06

New features

Video call supports background blur.

Bug Fixes

Fixed an issue in uni-app packaging for web projects where image loading icon failed.
Fixed the issue with the group call switch camera button.
Optimized the application getting stuck after a button click, addressing the abnormal issues caused by clicking the button again.

Version 3.2.3 @2024.04.19

New features

New Feature Group Call Supports Camera Switching.

Feature Optimization

Optimize TUICallKit SDK Readme.

Version 3.2.2 @2024.03.25

New features

Brand new UI visual effects, clearer functions, and better experience.

Feature Feature Optimization

Optimize data reporting using TUICallKit.

Version 3.2.1 @2024.03.08

New features

Language log reporting.

Version 3.2.0 @2024.02.23

New features

New features default offline push parameters.

Bug Fixes

Bug Fixes the issue where group calls have no nickname.

Version 3.1.9 @2024.01.30

Bug Fixes

Bug Fixes the issue where group calls do not display user information.
Fixed the issue where the 'Confirm' button was still clickable in the selector component when there were no members available.
Fixed the issue where muting the microphone during a call would prevent the audio stream from being transmitted in subsequent calls (upgrade trtc-cloud-js-sdk to v2.2.7+).

Version 3.1.8 @2024.01.19

Bug Fixes

Fixed the impact of the selector component's style on the page.

Version 3.1.7 @2024.01.12

Bug Fixes

New features a retry mechanism for interfaces and fixed the playback failure issue due to the inability to find the DOM node.
Fixed the device list selection style issue on PC.

Version 3.1.6 @2023.12.29

Feature Optimization

Optimized the prompt message during group calls.
Optimized the display issue when the nickname is too long.

Bug Fixes

Fixed the camera permission issue for voice call requests.
Fixed the issue with 'destroyed'.
Fixed the hang-up issue in the floating window under different call scenarios.
Fixed the display remote issue during the caller call status.
Fixed the incomplete fill style issue on PC.

Version 3.1.5 @2023.12.15

New features

Optimized the timing of accessing device permissions. No longer access device permissions during initialization, only access when using call.

Bug Fixes

Fixed @tencentcloud/call-uikit-vue2,@tencentcloud/call-uikit-vue2.6 components not having declare file issues.

Version 3.1.4 @2023.12.01

New features

Integrated into Chat, added isFromChat reporting.

Bug Fixes

Fixed the issue where the button is clickable under loading.

Version 3.1.3 @2023.11.17

New features

New features parameter validation to the interface.

Version 3.1.2 @2023.11.03

New features

Add the inviteUser feature for inviting others.
New features the feature to add people mid-call joinInGroupCall.
Introduced the feature to mute incoming call ringtones enableMuteMode.

Bug Fixes

Fixed the issue where remote stream microphone status was displayed incorrectly.

Version 3.1.0 @2023.10.20

New features

New features Floating Window feature.
New features enableFloatWindow interface for enabling/disabling Floating Window feature.
Desktop Terminal supports switching Camera and Microphone devices.
New features failure prompt message for calling blocked users.
New features support for Japanese.

Feature Optimization

During video calls, the big screen defaults to displaying the remote user.

Version 3.0.8 @2023.10.10

New features

New features reporting for version number, framework, and other information.

Version 3.0.7 @2023.10.08

New features

Add desktop video call duration display.

Feature Optimization

Optimized desktop video stream preview of rounded corners and black edges.

Optimized display priority for remote stream user information: Remarks > Nickname > userId.
Optimized TUICallKit component package size (Removed unused images and code).

Version 3.0.6 @2023.09.19

Bug Fixes

Fixed the message display issue integrated into TUIKit.

Version 3.0.5 @2023.09.15

Feature Optimization

Optimized mutual references in TUICallKit to avoid the stack overflow issue that occurs when packaging mini-programs in uniapp.

New features

New features a prompt for desktop devices when there is no permission, guiding customers on how to authorize devices.

Bug Fixes

Fixed setCallingBell where the called ringtone was overridden by the calling ringtone, leading to ringtone repetition issue.
Fixed styling issues on mobile devices.

Version 3.0.4 @2023.09.01

Bug Fixes

Fixed setCallingBell targeting incoming call ringtone (called ringtone).
Fixed destroyed error reporting problem.
Fixed the lack of Chinese and English in the error popup prompt.
Fixed the issue where it is impossible to switch between multi-screen support after turning off the camera during a 1v1 call.

Version 3.0.3 @2023.8.25

New features

Add @tencentcloud/call-uikit-vue2.6 compatible with Vue 2.6 version.

Feature Optimization

Optimize the default language of the component to the system default language.
Optimize the log information printed.
Optimize error message thrown by tuicall-engine-webrtc.
Optimize resource cleanup after component destruction.

Bug Fixes

Fixed an issue where videoDisplayMode,videoResolution did not take effect when calling again after hanging up.
Fixed the issue where statusChanged was not triggered during the call.
Fixed the issue of init being called multiple times.
Fixed the issue of being unable to switch between full and split screens when turning off the camera during a call.

Version 3.0.2 @2023.8.14

Bug Fixes

Fixed styling issues of the called component on the H5 platform.
Fixed the styling issues that occurred after switching to a small window during another call.

Version 3.0.1 @2023.8.8

Bug Fixes

Fixed the issue of the caller's local preview failing during a group call, and modified the component layer's default reading mode from the data layer.

Version 3.0.0 @2023.8.4

Breaking Change

Upgraded the underlying dependency tuicall-engine-webrtc to ^2.0.0. It no longer supports creating tim instances with tim-js-sdk. If you need to create a tim instance, please use @tencentcloud/chat.

Add

Add the custom ringtone feature setCallingBell.

Version 2.4.2 @2023.11.03

New features

Add the inviteUser feature for inviting others.

New features the feature to add people mid-call joinInGroupCall.

Introduced the feature to mute incoming call ringtones enableMuteMode.

Bug Fixes

Fixed the issue where remote stream microphone status was displayed incorrectly.

Version 2.4.0 @2023.10.20

New features

New features Floating Window feature.

New features enableFloatWindow interface for enabling/disabling Floating Window feature.

Desktop Terminal supports switching Camera and Microphone devices.

New features failure prompt message for calling blocked users.

New features support for Japanese.

Feature Optimization

During video calls, the big screen defaults to displaying the remote user.

Version 2.3.9 @2023.10.10

New features

New features reporting for version number, framework, and other information.

Version 2.3.8 @2023.10.08

New features

Add desktop video call duration display.

Feature Optimization

Optimized desktop video stream preview of rounded corners and black edges.

Optimized display priority for remote stream user information: Remarks > Nickname > userId.
Optimized TUICallKit component package size (Removed unused images and code).

Version 2.3.6 @2023.09.15

Feature Optimization

Optimized mutual references in TUICallKit to avoid the stack overflow issue that occurs when packaging mini-programs in uniapp.

New features

New features a prompt for desktop devices when there is no permission, guiding customers on how to authorize devices.

Bug Fixes

Fixed setCallingBell where the called ringtone was overridden by the calling ringtone, leading to ringtone repetition issue.
Fixed styling issues on mobile devices.

Version 2.3.5 @2023.9.5

Bug Fixes

Fixed the issue where the camera and microphone buttons were by default turned on before entering the room.

Version 2.3.4 @2023.9.1

New features

Add the feature to disable or enable the camera before answering a video call.

Bug Fixes

Fixed the issue where it is impossible to switch screen sizes after turning off the camera during a 1v1 call.
Fixed the issue where statusChanged was not triggered when switching from a video call to a voice call.

Version 2.3.3 @2023.8.22

Bug Fixes

Fixed an issue where videoDisplayMode,videoResolution did not take effect when calling again after hanging up.
Fixed the issue where statusChanged was not triggered during the call.

Version 2.3.2 @2023.7.26

Breaking Change

Removed the TUICallKitMini floating window component, merged it into the TUICallKit component.
The thrown @kicked-out event has been adjusted to the affinity callback `:kickedOut` .
The thrown @status-changed event has been adjusted to the affinity callback `:statusChanged` .

Add

Add animation effect when the call page appears.
Add group call layout on H5.

Feature Optimization

Optimize the problem prompt message during the call, prompt method.
Optimize support on H5 page, interaction.
Optimize the time it takes to bring up the call interface.
Optimize the @tencentcloud/call-uikit-vue package directory structure.

Bug Fixes

Fixed call issues under boundary operations such as immediately hanging up after connecting.
Fixed styling issues on H5 for some models, browsers.
Fixed call anomaly issues caused by repeated clicks.

Version 2.2.1 @2023.7.7

Add

@tencentcloud/call-uikit-vue2 Add detection and prompt for the Vue version.

Bug Fixes

Fixed the issue where repeatedly clicking the "Answer" button on the incoming call page causes the answer to fail.

Version 2.2.0 @2023.6.30

Add

call,groupCall support custom roomID parameter for digital room numbers.
call,groupCall support custom userData parameter for extended fields (used to add additional information in the invitation signaling).
Add setSelfInfo interface, supporting user configuration of aliases and profile photos.

Version 2.1.0 @2023.4.14

Add

In the H5 voice chat pattern, while calling, it supports displaying the opposite party's nickname.
When initiating a call fails, "Call initiation failed" will be displayed on the calling page.
When answering a call fails, "Answer failed" will be shown on the incoming call page.
Support for monitoring whether the current user is kicked out (e.g., due to being logged out), see TUICallKit Method - @kicked-out.
Support for listening to TUICallKit call status, see TUICallKit Method - @status-changed.
Support for business-side code to control the answering, canceling, and hanging up of calls, see More Features - Auto-answer through Interface Setting.
The Vue2 version adds TypeScript type declaration files, allowing normal compilation of types in TypeScript projects.

Bug Fixes

Fixed a warning about updating personal profile interface appearing in the console during component initialization.
Fixed background image misalignment issues of the callee answer button in the H5 pattern.

Interface Change

`TUICallKitServer.destroy()` New features invocation limit, can only be called in non-call status.

Version 2.0.1 @2023.03.31

Add

Optimized the rendering logic of 1v1 and group call videos to improve performance and stability.
Optimized UI presentation, support for displaying corresponding UI during the execution of `TUICallKitServer.call()` , which enables immediate UI display of `<TUICallKit/>` components upon clicking the call button.

Bug Fixes

Fixed the issue of incorrect nickname display in group calls.

Fixed the issue of CSS not being scoped properly, leading to global style pollution.

Version 2.0.0 @2023.03.21

Add

Support for importing the packaged CallKit file from npm.

Support for Vue projects in JavaScript version.

Supports all versions of Vue2 projects, applicable to the npm package for Vue2: call-uikit-vue2.

Bug Fixes

Fixed the issue where calls could not be initiated due to the absence of a camera device or permission.

Version 1.4.2 @2023.03.03

Add

Supports setting call resolution. See API Documentation for details.

Supports changing the display pattern. See API Documentation for details.

Optimized the integration steps.

Optimized error throwing.

Version 1.4.1 @2023.02.13

Add

Optimized the logic for previewing the local camera.

Optimized the rendering logic for remote video streams.

Version 1.4.0 @2023.01.06

Add

Supports importing in Vue2.7+ projects.

Call interface defaults to displaying nicknames.

Version 1.3.3 @2022.12.27

Add

New features null value detection for the call list when making calls in the Basic Demo.
New features a loading icon when making calls in the Basic Demo.
Optimized the logic for device detection in the Basic Demo, no longer proactively popping up after manually skipping.
Optimized the reference method for component icons.
Changed the default package management tool to npm.
Optimized the rendering method for videos, reducing the number of iterative renderings.

Bug Fixes

Fixed an error in the Basic Demo caused by outdated dependencies in vue-CLI.

Version 1.3.2 @2022.12.07

Add

Language switching is supported, see setLanguage for interface details.
Optimized the device detection logic in the Basic Demo; it will no longer pop up proactively after being manually skipped.

Bug Fixes

Fixed a warning caused by introducing `defineProps` .

Version 1.3.1 @2022.11.29

Note:
This version depends on the SDK version tuicall-engine-webrtc@1.2.1, please update promptly.

Add

Optimize style details.
Support monitoring the other party's modification of call type when the call is not answered.
Basic demo adds device detection feature.

Bug Fixes

Fixed errors caused by internal logic when hanging up the phone.

Version 1.3.0 @2022.11.14

Add

Supports automatic switching to vertical screen style when using mobile H5.
Supports previewing the local camera when making a phone call.
Basic demo adds device detection before making a phone call.

Bug Fixes

Fixed the issue where the tim instance did not fully log out after calling TUICallKitServer.destroyed().
Fixed the problem where a 'No response' message was displayed when the line was busy.
Fixed the issue where TypeScript types were not successfully packaged in a vite environment.

Interface Change

When actively calling TUICallKitServer.call() or TUICallKitServer.groupCall(), if an error occurs, the beforeCalling callback will not be invoked. Please use try catch to capture errors directly.

Version 1.2.0 @2022.11.03

Add

Adaptation to new versions of TUICallEngine SDK.

Version 1.1.0 @2022.10.21

Add

During a call, the call page can be displayed in full screen.
During a call, you can use `<TUICallKitMini/>` to minimize.

Bug Fixes

Fixed known issues, improved stability.

Version 1.0.3 @2022.10.14

Add

Basic demo adds quick copy UserID, one-click open new window.

Version 1.0.2 @2022.09.30

Add

Optimized access documentation, added demonstration images and detailed guides.

Bug Fixes

Fixed the issue where the device status bit became invalid when first entering the room.
Fixed the occasional failure of Icon loading when packaging with webpack.
Fixed known styling issues.

Version 1.0.1 @2022.09.26

Add

Hide the other party's microphone icon during a phone call.

Bug Fixes

Fixed the issue where the SDKAppID input box in the basic demo should be numeric.

Version 1.0.0 @2022.09.23

Quickly Run Through the TUICallKit Demo
Quick Integration of TUICallKit
TUICallKit API
TUICallKit Customizable Interface Guide
Frequently Asked Questions About TUICallKit (Web)

Android&iOS

Last updated：2024-08-09 20:13:49

Version 2.5.0.1025 @ 2024.8.7

Feature optimization

Android: Optimize the logic of joinInGroupCall to fix memory leaks.

Bug Fixes

Android&iOS: Fix the issue where web users do not receive group call invitations sent from Android and iOS.
Android: Fix the issue where during a group call, A voice calls B, and when B clicks the push notification to open the interface, it shows Speaker instead of Earpiece.

Version 2.4.0.970 Released June 15, 2024

Feature Optimization

Android&iOS: Show tips in weak network conditions.
Android: Optimize the incoming call strategy when the callee's screen is locked.
iOS: Fix the issue of abnormal memory growth in group calls.

Bug Fixes

Android&iOS: Fix the display issue when the joinInGroupCall interface is invoked.

Version 2.3.0.915 Released April 15, 2024

Feature Optimization

Android&iOS: Support for displaying call status at the top of the TUIChat group, and allowing group members to join the call actively.
Android&iOS: Optimized incoming call pop-up logic, default to display banner answer box.
Android&iOS: Video call supports background blur.

Bug Fixes

Android: Fixed the issue of no response when clicking the delete button in the call record editing interface.
iOS: Fixed the issue of ghosting during the switching process when clicking on the member view in the group call.

iOS: Fixed the issue of not displaying in specific scenarios of the audio and video interface.
Android&iOS: Fixed the issue of missing prompts after the call ends when calling a busy line user.

Version 2.2.0.860 Released February 1, 2024

Feature Optimization

Android&iOS: Brand new UI visual effects, clearer functions, and better experience.

Bug Fixes

Android&iOS: Fixed the issue of microphone and camera device mutual occupation after answering incoming calls in conference, live broadcast scenarios.

Version 2.1.0.810 Released December 19, 2023

Feature Optimization

Android: Optimized the prompt for abnormality when calling the TUICallKit interface without logging in.
Android: Optimized compatibility for Android 14 platform (API 34), for details, see: Android 14 behavior changes.
Android&iOS: Optimized the display of user nicknames, displayed in the following order: User Remarks > User Nickname > User Id, the default is userId.

Bug Fixes

iOS: Fixed the issue of overlapping group call avatars.
iOS: Fixed the problem that the keyboard cannot be retracted when returning to the call interface after opening the floating window to send messages during a video call.
iOS: Fixed the issue that the camera cannot be switched and moved when switching the camera, turning off the camera, and then switching back to the original camera and reopening it during a video call.
Android: Fixed the issue that the small window of a single video call cannot be moved under the right-to-left layout mode (RTL mode) such as Arabic.

Version 2.0.0.750 Released November 3, 2023

Functionality Enhancement

Android & iOS: The UIKit supports the Japanese language.
Android & iOS: Enhanced the display of call nicknames.
Android & iOS: Adjusted the default ringtone volume from 60% to 100%.

Defect Rectification

iOS: Corrected the slow image loading issue in Swift version.
iOS: Fixed the issue where the call invitation was automatically cancelled after the caller initiated the call and moved the application to the background.

Version 1.9.0.680 Released September 27, 2023

Functionality Enhancement

Android & iOS: Added support for the Arabic language.
Android & iOS: Optimized the package purchase prompts, enabling redirection to corresponding package purchase pages based on the provided links.
Android & iOS: Enhanced the default bitrate at different resolutions to ensure clearer output at higher resolutions. For details, refer to Resolution.
Android & iOS: The default bitrate for video calls has been set to 600kbps, with a beauty level of grade 4.

Defect Rectification

Android & iOS: Resolved inconsistencies between the rejection prompt when initiating a call to a user on the blacklist and the rejection prompt when sending a private chat message.
iOS: Rectified an anomaly in the video placement for the initiator, which occurred on a 4-person group video call interface when one member declined the call.
iOS: Addressed an issue where the resolution would be reset if the beauty feature was enabled immediately after successful login.

Version 1.8.0.620 Released August 14, 2023

Functionality Enhancement

Android & iOS: By default, call messages are excluded from the unread count.
Android: Enhanced the redirection page for floating window permissions on Xiaomi smartphones.

Defect Rectification

iOS: Remedied an issue where the onKickOffline callback interface became ineffective after being kicked offline.
iOS: Fixed an issue where, after clearing the call on the Missed Call interface, returning to the All Calls interface would result in an empty list.

Version 1.7.2.570 Released July 20, 2023

Functionality Enhancement

Android: Gravity sensor is turned off by default, optimizing the call experience on large-screen and customized devices.

Defect Rectification

Android & iOS: Rectified an issue where, after User A (online) calls User B (offline) and cancels the call, User A calls back User B who logs in thereafter, leading to abnormal cloud call records for user B.
Android: Resolved the crash issue of TUICallKit after upgrading the TRTC SDK version to 11.3.

Version 1.7.0.460 Released June 25, 2023

Functionality Enhancement

Android & iOS: Includes UI integration solutions, optimizes sample projects, and improves call setting items.
Android: Reduced the status preservation level during a call to only show standby prompts in the status bar; removed notifications and vibrations.

Version 1.6.1.410 Released on May 22, 2023

New features:

Android & iOS: The UI interface call() and groupCall() now support custom room ID.
Android & iOS: When initiating a call, a string-type room ID can be passed in, see CallParams for details.

Bug fixes:

Android: Fixed issue where an error would occur on the groupCall when generating list parameters using Arrays.asList.
Android: Fixed issue where the video call display was abnormal.
iOS: Fixed issue where it conflicted with TUIRoom component.
iOS: Fixed issue where initiating a call immediately after successful login would cause a crash.
iOS: Fixed issue where the invite page would not appear intermittently when clicking on a notification message to enter the app.

Version 1.6.0.360 Released on April 27, 2023

New features:

Android: TUICallKit added the Kotlin language version;

iOS: TUICallKit added the Swift language version;
Android & iOS: Added a page to display local call records.

Functional optimization:

Android: Optimized the display of video call avatars.
Android & iOS: In group calls, other group members can be invited to join the call by default.

Bug fixes:

Android: Fixed issues where devices running Android 12 or higher would have no sound after being connected to Bluetooth;
Android: Fixed intermittent issues where the muting setting on the callee side was not effective;
iOS: Fixed intermittent issues where devices could not receive incoming call invitations after relogging in;
iOS: Fixed the issue where the enableCustomViewRoute interface of TUICallKit was not valid;
iOS: Fixed the issue where the nickname was displayed incorrectly on the VoIP push page.

Version 1.5.1.310 Released on April 17, 2023

New features:

Android & iOS: Added VoIP message push function to provide a better call answering experience.
Android & iOS: Support custom extended fields when initiating a call, see TUICallDefine.CallParams parameter in the call() method for details.

Functional optimization:

Android & iOS: Optimized offline push capabilities for Huawei, Xiaomi, FCM, and other manufacturers, added manufacturer message categories, and channel ID settings.

Bug fixes:

Android & iOS: Fixed issue where the Chat custom property was overwritten after initiating a call.
Android: Fixed issue where the totalTime unit in the onCallEnd callback was incorrect.

Version 1.5.0.305 Released on March 09, 2023

Functional optimization:

Android & iOS: Optimized chat-message display.
Android: The ear-to-screen messaging function is turned off by default now.
Android: Upgraded gradle plugin and version.

Android: Optimized mediaPlayer class, supporting loop playback of ringtones.

Bug fixes:

Android & iOS: Fixed issue where the callee would not receive the onCallCancel callback when answering a call fails.
Android: Fixed issue where the caller would receive an exception when the callee fails to answer the call.
Android: Fixed issue where the caller cancels the call during the permission check of the first call and the callee pulls up the interface again.
Android: Fixed the issue where userId was empty when returning network quality to the upper callback.

Version 1.4.0.255 Released on January 06, 2023

New features:

Android & iOS: Support custom call timeout time, see TUICallDefine.CallParams parameter in the call() method for details.

Bug fixes:

Android & iOS: Fixed issue where joining a room actively (joinInGroupCall) would result in abnormal termination of the call.
Android: Fixed issue where there were abnormalities with call status when you exited the audio and video call answer interface and came back to the foreground again.

Version 1.3.0.205 Released on November 30, 2022

New features:

Android & iOS: Added beauty setting interface setBeautyLevel(), supporting turning off default beauty.

Functional optimization:

iOS: Optimized the framework size of TUICallKit.

Bug fixes:

Android & iOS: Fixed issue where the calling interface did not disappear when the server dissolves a room or kicks out a user.
Android: Fixed issue where if A called offline user B and then cancelled, then A called B again and B came online, the calling interface did not appear.

Version 1.2.0.153 Released on November 14, 2022

New features:

Android & iOS: Support for custom video encoding resolutions.
Android & iOS: Support setting rendering parameters for video: rendering direction and filling mode.
Android & iOS: Support integration of third-party beauty features.
Android & iOS: TUICallKit has added overloaded interfaces call() and groupCall(), supporting custom offline messages (see API documentation for details).

Functional optimization:

Android & iOS: Optimized some TUICallObserver callback exception issues.
Android & iOS: Optimized the video-to-audio switching function, supporting switching in offline state.
Android & iOS: Improved error codes and error prompts for TUICallKit.
iOS: Standardized TUICallEngine and TUICallKit Swift API names.

Bug fixes:

Android & iOS: Fixed issue where you would still receive previously rejected incoming calls after logging back in to your account.
Android: Fixed issue where you would encounter abnormal hang-ups in invites from group chats in one-to-one chats.
Android: Fixed issue where there were abnormalities with multiple-scene exits from live rooms, preventing the initiation of calls.
Android: Fixed issue where contacting person A while B and C were calling each other at the same time would cause C to enter A's room at random.

Version 1.1.0.103 Released on September 30, 2022

Android & iOS: Optimized the feature of inviting new members to the current group call.
Android & iOS: Optimized the call process to avoid the charging of recording, moderation, and other fees before a call is answered.
Android & iOS: Added support for custom offline notifications.
Android & iOS: Changed the parameters of some **TUICallEngine** APIs. For details, see call(), groupCall(), inviteUser(), and onCallReceived().
Android & iOS: Fixed occasional callback errors during a group call.
Android & iOS: Fixed the status abnormal issue caused by repeated login or expired `UserSig`.
iOS: Fixed the issue where, when a mixed-language `TUICallKit` project is built with Objective-C and Swift, an error occurs when `init` is called.

Android: Fixed the issue where an error occurs when the floating window feature is integrated for a Kotlin project.

Version 1.0.0.53 Released on August 15, 2022

First release:

Android & iOS: Supports one-to-one and group audio/video calls.

Android & iOS: Supports offline call push for mainstream devices on the market.

Android & iOS: Supports custom profile photos and aliases.

Android & iOS: Supports floating call windows.

Android & iOS: Supports custom ringtones.

Android & iOS: Supports receiving calls when the user is logged in on multiple platforms.

Flutter

Last updated：2024-04-28 10:40:58

Version 2.3.2 @2024.04.24

New features

Android & iOS: Optimizing incoming call popup logic, defaulting to display the banner answering box.
Android & iOS: Video calls support background blur.

Version 2.3.1 @2024.04.22

Fixes

Fixes the Method Channel error reporting problem that occurs on non-iOS & Android platforms.

Version 2.3.0 @2024.04.18

New features

Android & iOS: Supports displaying call status at the top of the group in tencent_cloud_chat_uikit and allows group members to initiate join a call.

Feature optimization

Android & iOS: Optimizing the log in method when used concurrently with tencent_cloud_chat_uikit.

dependency description

Upgrade tencent_cloud_uikit_core to version 1.6.0.
Upgrade the dependency client TUICallEngine SDK to version 2.3.0.915.

Version 2.2.3 @2024.03.08

dependency description

Upgrade tencent_cloud_uikit_core to version 1.5.2.

Version 2.2.2 @2024.03.07

New Features：

Android&iOS: TUICallObserver's onCallReceived callback adds user-defined parameter userData.

Version 2.2.1 @2024.02.06

Bug Fixes：

Android: After killing the process, the application received the FCM push but did not answer it, and then entered the application page with an exception.

Version 2.2.0 @2024.02.03

New Features：

Android: Supports FCM push and needs to be used together with tencent_cloud_chat_push

Version 2.1.1 @2024.01.06

Bug Fixes：

Android&iOS : Modify some UI 3.0 UI related bugs

Version 2.1.0 @2024.01.04

New Features：

Android&iOS : Use the new UI3.0.

Version 2.0.6 @2023.12.15

Bug Fixes：

iOS : The problem of missing some fields in iOS offline push information

Version 2.0.5 @2023.12.10

Bug Fixes：

iOS : Fixed the issue of abnormal ringtone when entering the call page during VoIP push and abnormal pulling of remote stream.
Android : Fixed an issue where the incoming call page would be abnormal when in the background when the floating window permission was not obtained.

Version 2.0.4 @2023.12.04

Bug Fixes：

Android&iOS : Fixed the intermittent issue of incoming call page not displaying sometimes after clicking on an incoming call notification.

Version 2.0.2 @2023.11.27

Bug Fixes

Android : Fix the issue of call failure under multiple Flutter Engine conditions.

Version 2.0.1 @2023.11.15

Bug Fixes

Android & iOS : Fixed an incompatibility issue caused by Tencent RTC Observer when using Tencent RTC components with other components that also use Tencent RTC.
Android : Fixed an incompatibility issue when using TUIRoom in conjunction with other devices.

Version 2.0.0 @2023.11.06

Dependency Description

Upgrade the dependent client SDK version:
Android&iOS TUICore:7.6.5011.
Android&iOS TUICallEngine:2.0.0.750.

Version 1.9.1 @2023.10.21

New Features：

iOS: Support Voip.

Bug Fixes

iOS: Fixed an issue where the call page would be abnormal when receiving a call in the background.

Version 1.9.0 @2023.10.09

New Features

Android&iOS: Add an interface for setting ringtones.

Function Optimization:

Android & iOS: Optimize package purchasing prompts.
Android & iOS: Optimize default bitrates for different resolutions, see details.

Bug Fixes

iOS: Fixed the issue where the same Observer object can be registered twice.

Dependency Description

Upgrade the dependent client SDK version:
Android&iOS TUICore:7.5.4852.
Android&iOS TUICallEngine:1.9.0.680.

Version 1.8.3 @2023.08.25

Bug Fixes

Android&iOS: Fixed the problem of no call message display when using tencent_cloud_chat_uikit.
Android&iOS: Fixed the problem of occasionally pulling up the group call page during a single-person call.
Android&iOS: Fixed the problem of occasionally pulling up the call page twice during a call.
Android&iOS: Fixed the problem of abnormal display of call duration.
Function Optimization:
Android: Optimized the problem of failing to pull up the interface in the background when receiving a call.

Dependency Description：

Upgrade tencent_cloud_uikit_core to version 1.1.1.

Version 1.8.2 @2023.08.19

Bug Fixes

iOS: Fixed the problem of some compilers failing to compile due to the use of deprecated Swift interfaces.

Version 1.8.1 @2023.08.18

Bug Fixes：

Android&iOS: Fixed the problem of the video stream of the other party being displayed during a group call voice call.

Version 1.8.0 @ 2023.08.17

New Features:

Android&iOS：Build a new TUICalkit based on the Dart language, which makes it easier to customize your own UI style.
Android&iOS：TUICallEngine adds multiple business interfaces such as hangup, accept, and reject.

Version 1.7.4 @ 2023.07.20

Function Optimization:

Android：By default, the gravity sensor is turned off to optimize the call experience on large screens and customized devices.

Bug Fixes:

Android&iOS：A calls B (offline) and then cancels, A calls B again, B logs in and goes online, and B's cloud call record is abnormal.

Version 1.7.3 @ 2023.07.19

Function Optimization:

Android: Supports development & debugging using the emulator.

Dependency Notes:

The client SDK version that the upgrade depends on: Android LiteAVSDK_Professional: 11.3.0.13176.

Version 1.7.2 @ 2023.07.09

Bug Fixes:

iOS: Upgrade the client SDK version to fix the problem of AppStore listing failure caused by Non-public API usage.

Version 1.7.1 @ 2023.07.03

New Features:

Android&iOS：Added cloud call records, you can activate the service on the console for experience query.

Function Optimization:

Android：Reduce the level of system keep-alive during calls, only display the keep-alive reminder in the status bar, and remove notifications and vibrations.

Version 1.6.3 @ 2023.06.03

Bug Fixes:

iOS: Fix the issue of an empty page when adding people halfway after calling joinInGroupCall.
iOS: Fix the issue of user screen overlap after calling joinInGroupCall.

Version 1.6.2 @ 2023.05.30

Bug Fixes:

Android: Fix the crash issue caused by calling the joinInGroupCall API.

Version 1.6.1 @ 2023.05.15

Bug Fixes:

Android: Fix the occasional crash when applying for floating window permissions on specific Vivo models.

Dependency Notes:

Upgrade the dependent client SDK version: Android LiteAVSDK_Professional: 11.1.0.13111, iOS TXLiteAVSDK_Professional: 11.1.14143.

Version 1.6.0 @ 2023.05.03

New Features:

Android&iOS: Add hangup interface.
Android&iOS: Add user-defined fields and user-defined call timeout duration.
Android&iOS: Add midway join page for group calls.

Function Optimization:

Android: Optimize single-user video call avatar display.
Android&iOS: By default, support inviting other group members to join the call in group calls.

Bug Fixes:

Android: Fix the issue of no sound on Android 12 and above devices after connecting to Bluetooth.
Android: Fix the occasional issue of mute settings not taking effect on the called party's side.
iOS: Fix the occasional issue of the device not receiving incoming call invitations after re-login.
iOS: Fix the issue of incorrect nickname display on VoIP push page.

Dependency Notes:

Upgrade the dependent client SDK version: Android LiteAVSDK_Professional: 11.1.0.13111, iOS TXLiteAVSDK_Professional: 11.1.14143.

Flutter Version 1.5.4 @ 2023.04.14

New Features:

Android&iOS: Add offline push parameters for Xiaomi, Huawei, and VIVO.
iOS: Supports VoIP message push function.
Android&iOS: Add resolution setting function.

Function Optimization:

Android: Optimize the unit of the totaltime parameter in the onCallEnd callback to milliseconds.

Bug Fixes:

Android&iOS: Fix onCallReceived callback exception issue.
iOS: Fix the issue of incomplete call page display when the screen is rotated.

Version 1.5.3 @ 2023.03.17

Bug Fixes:

Android: Fix the packaging failure issue.
Android&iOS: Fix the issue of throwing exceptions when callback methods are not implemented.

Version 1.5.2 @ 2023.03.13

Bug Fixes:

Android: Fix the compilation error caused by the API change of TUICallDefine.OfflinePushInfo.

Version 1.5.1 @ 2023.03.13

Bug Fixes:

Android&iOS: Fix the issue of incorrect version dependency of tencent_calls_engine.

Version 1.5.0 @ 2023.03.13

Function Optimization:

Android: Proximity wake-up function is turned off by default.
Android: Upgrade gradle plugin and version.
Android: Optimize the ringtone playback class, supporting loop playback.

Bug Fixes:

Android&iOS: Fix the issue that the onCallCancel callback is missing when the called party fails to answer the call.
Android: Fix the abnormal problem of the caller when the called party fails to answer the call.
Android: Fix the issue that the interface is pulled up again by the called party when checking permissions for the first call and the calling party cancels the call.

Android: Fix the issue of userId being empty when calling back the network quality to the upper layer.
iOS: Fix the issue of incorrect Observer registration timing in Example.

Version 1.4.2 @ 2023.02.24

Bug Fixes:

Android&iOS: Fix the issue of abnormal calls caused by the wrong Observer registration timing in Example.
iOS: Fix the occasional invalid setting issue of removeObserver API.

Version 1.4.1 @ 2023.02.20

Bug Fixes:

Android: Fix the issue of the OnCallEnd event being lost after the call ends.

Version 1.4.0 @ 2023.01.16

Bug Fixes:

Android&iOS: Fix the issue of abnormal call termination when actively joining a room (joinInGroupCall).
Android: Fix the issue of abnormal call status when entering the background during audio and video call answering and returning to the foreground.
Android: Fix the issue of call initiation failure caused by login status when integrating the tencent_cloud_chat_uikit plugin.
Android: Fix the issue of call initiation failure due to parameter check issues when initiating a group call.

Version 1.3.1 @ 2022.12.27

New Features:

Android&iOS: Support custom offline push message during a call.

Bug Fixes:

Android: Fix the issue of sdkappid is invalid prompt when integrating the tencent_cloud_chat_uikit plugin.

Version 1.3.0 @ 2022.12.02

Function Optimization:

iOS: Optimize the size of the TUICallKit Framework.

Bug Fixes:

Android&iOS: Fix the issue of the call interface not disappearing when the server-side dissolves the room or kicks out users.
Android: Fix the issue that the call interface does not display when user A calls offline user B, then cancels the call; A calls B again, and B's call interface does not display after going online.

Version 1.2.2 @ 2022.11.17

Bug Fixes:

iOS: Fix the compilation issue caused by static library linking.

Version 1.2.0 @ 2022.11.17

New Features:

Support 1v1 audio and video calls, group audio and video calls.
Support custom avatar and custom nickname.
Support setting custom ringtones.
Support enabling floating window during the call.
Support incoming call service for multi-platform login status.

FAQs(TUICallKit)
iOS

Last updated：2022-11-24 10:52:37

What should I do if I receive the error message "The package you purchased does not support this ability"?

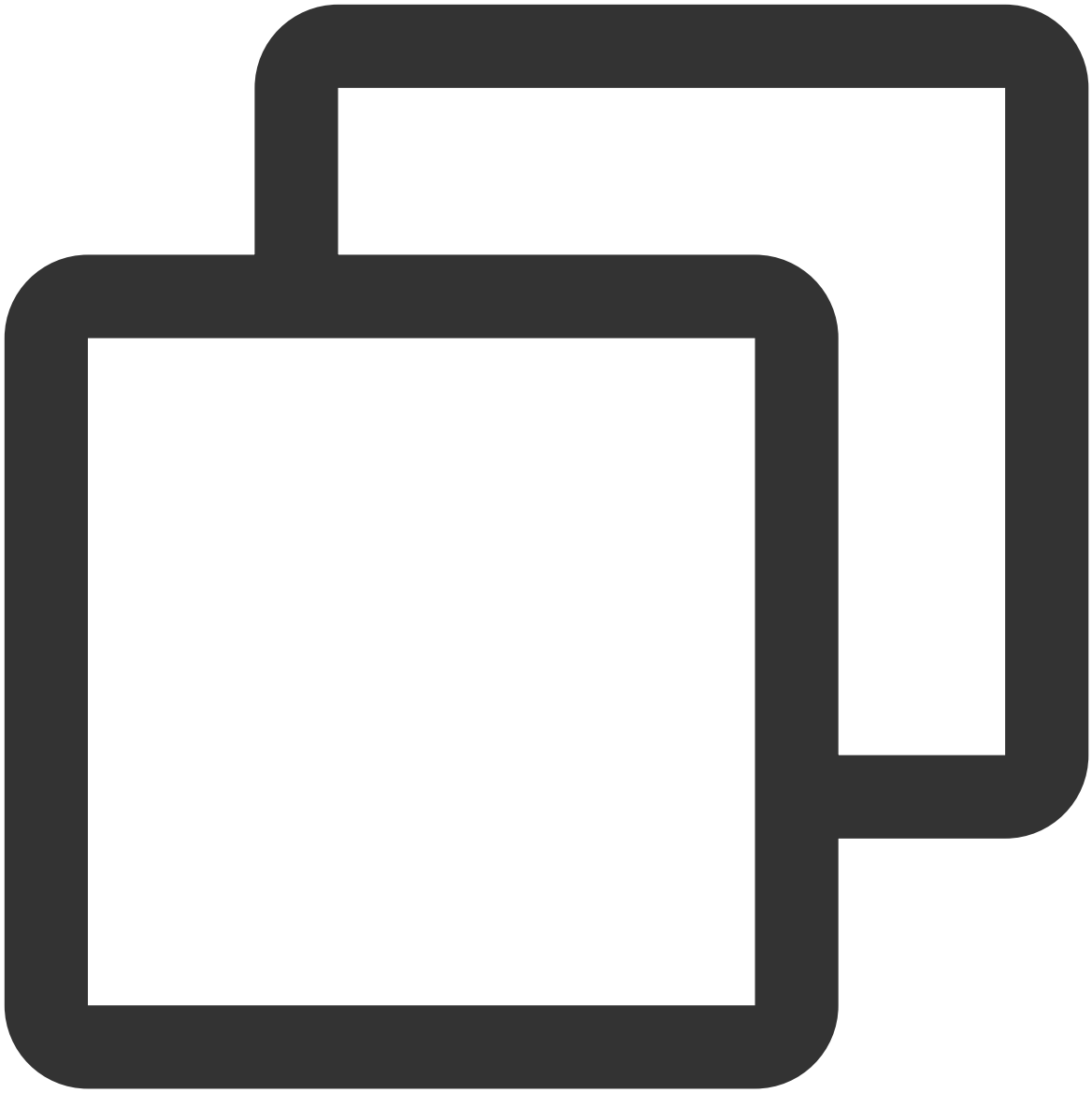
If the above error occurs, it indicates that you haven't activated TRTC or your TRTC package has expired. For directions on how to activate TRTC, see Step 1. Activate services. You can get a free trial package after activation.

How do I buy a TRTC package?

See Billing Overview. If you have any questions, please click the icon in the bottom right corner of the page to contact us.

How do I modify the source code of `TUICallKit` ?

Import the component using CocoaPods:
1. Create a `TUICallKit` folder in the same directory as `Podfile` in your project.
2. Go to the component's GitHub page, clone or download the code, and copy the `TUICallKit` and `Resources` folders and the `TUICallKit.podspec` file in iOS to the `TUICallKit` folder in your project.
3. Add the following dependency to your `Podfile` .



```
# :path => "The relative path of `TUICallKit.podspec`"
pod 'TUICallKit', :path => "TUICallKit/TUICallKit.podspec"
```

4. Execute `pod install` .

notice

Make sure `TUICallKit` , `Resources` , and `TUICallKit.podspec` are in the same directory.

After `TUICallKit` is imported, you will see this directory structure:

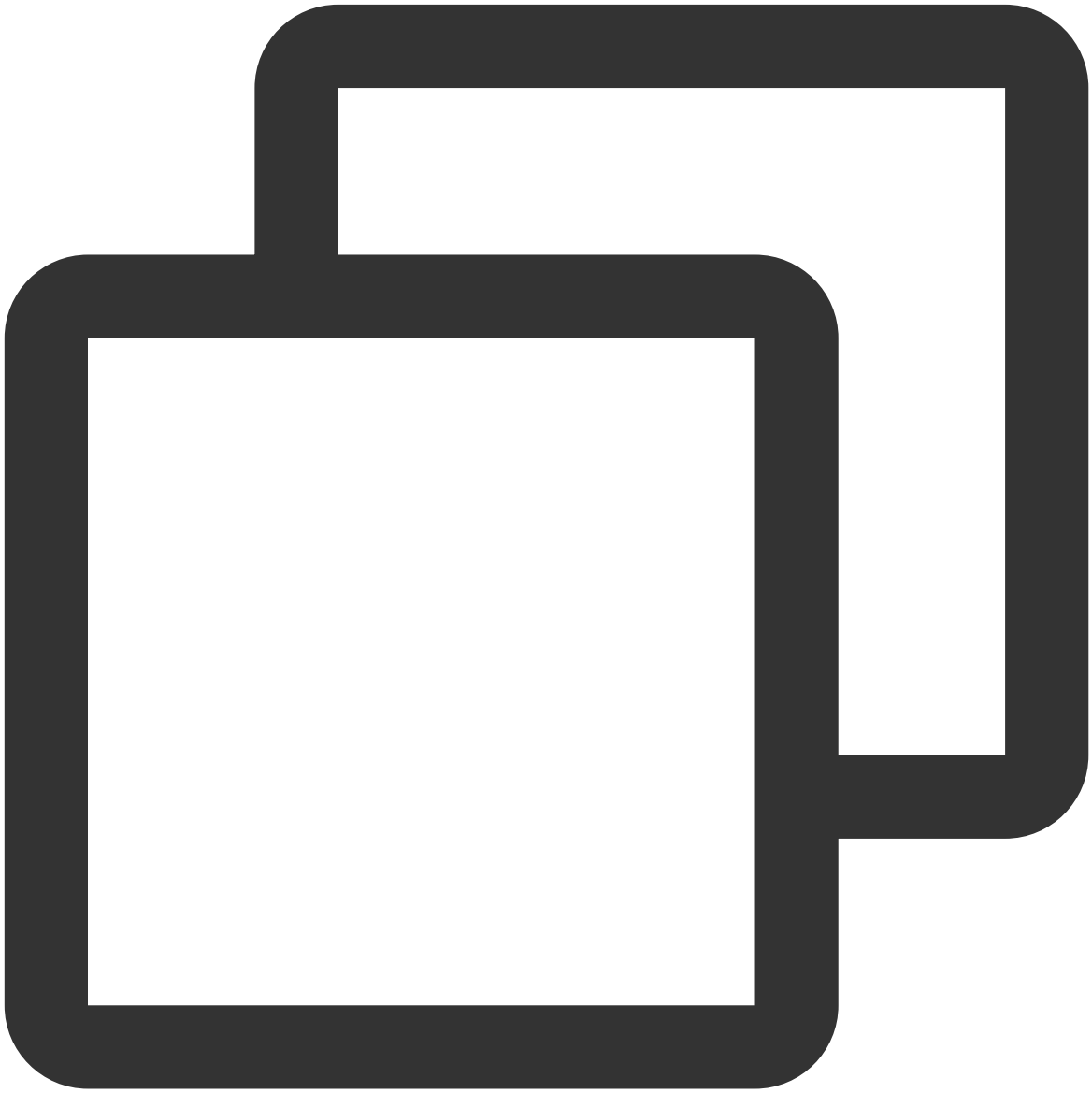
explain

The folders are organized hierarchically, making it easy for you to view and modify the source code.

What should I do if TUICallKit conflicts with an audio/video library that I have integrated?

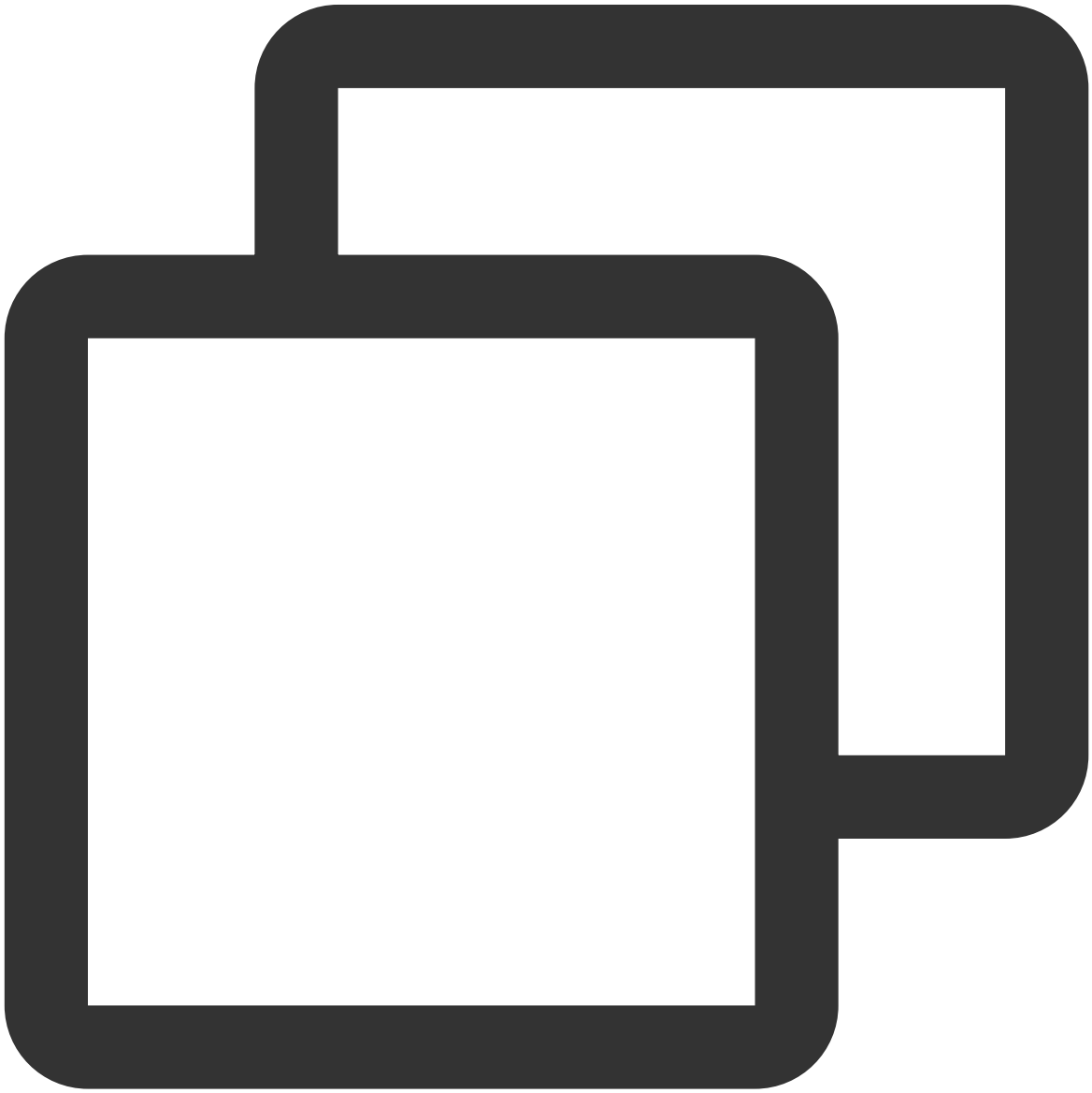
You cannot integrate multiple audio/video libraries of Tencent Cloud at the same time. Otherwise, a duplicate symbol error may occur. You can use the following methods to troubleshoot the issue:

1. If you have integrated `TXLiteAVSDK_TRTC`, a duplicate symbol error will not occur. Just add the following in `Podfile`:



```
pod 'TUICallKit'
```

2. If you have integrated `TXLiteAVSDK_Professional`, a duplicate symbol error will occur. You can add the following in `Podfile`:



```
pod 'TUICallKit/Professional'
```

3. If you have integrated `TXLiteAVSDK_Enterprise`, a duplicate symbol error will occur. We recommend you update to `TXLiteAVSDK_Professional` and use `TUICallKit/Professional`.

What should I do if the "ld: framework not found BoringSSL clang: error: linker command failed with exit code 1 sdk" error occurs when I integrate `TUICallKit`?

Currently, the audio/video libraries `TUICallKit` relies on do not support emulators. Please use a real device for demo run or debugging.

Can I not integrate the IM SDK if I use only TRTC features?

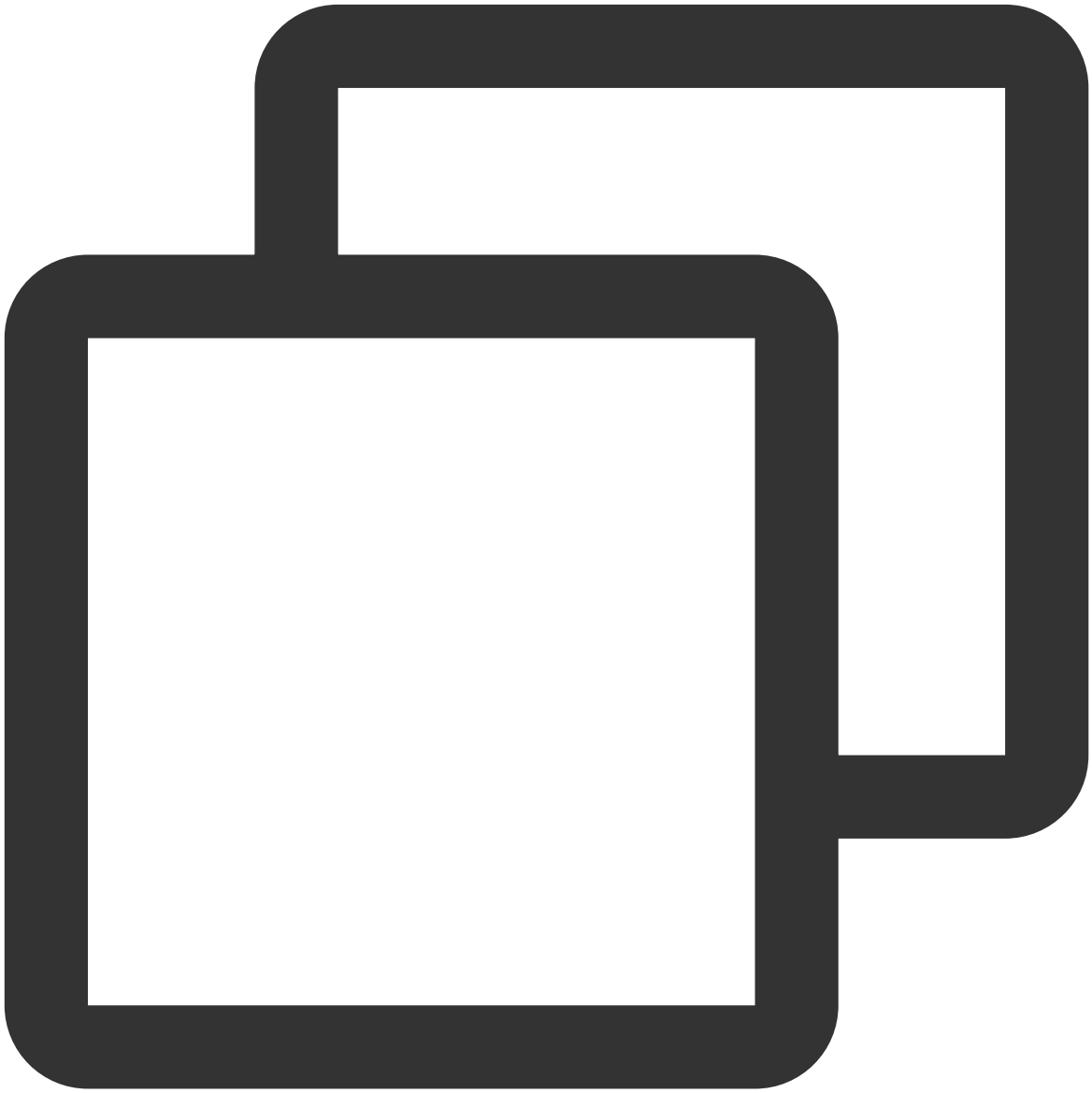
No, you can't. All components of `TUIKit` rely on the IM SDK for underlying logic such as outgoing call signaling and busy line signaling. If you have already purchased IM's services, you can use them according to `TUICallKit` logic.

Does `TUICallKit` support custom ringtones?

Yes. You can implement this by calling TUICalling#setCallingBell.

How do I install CocoaPods?

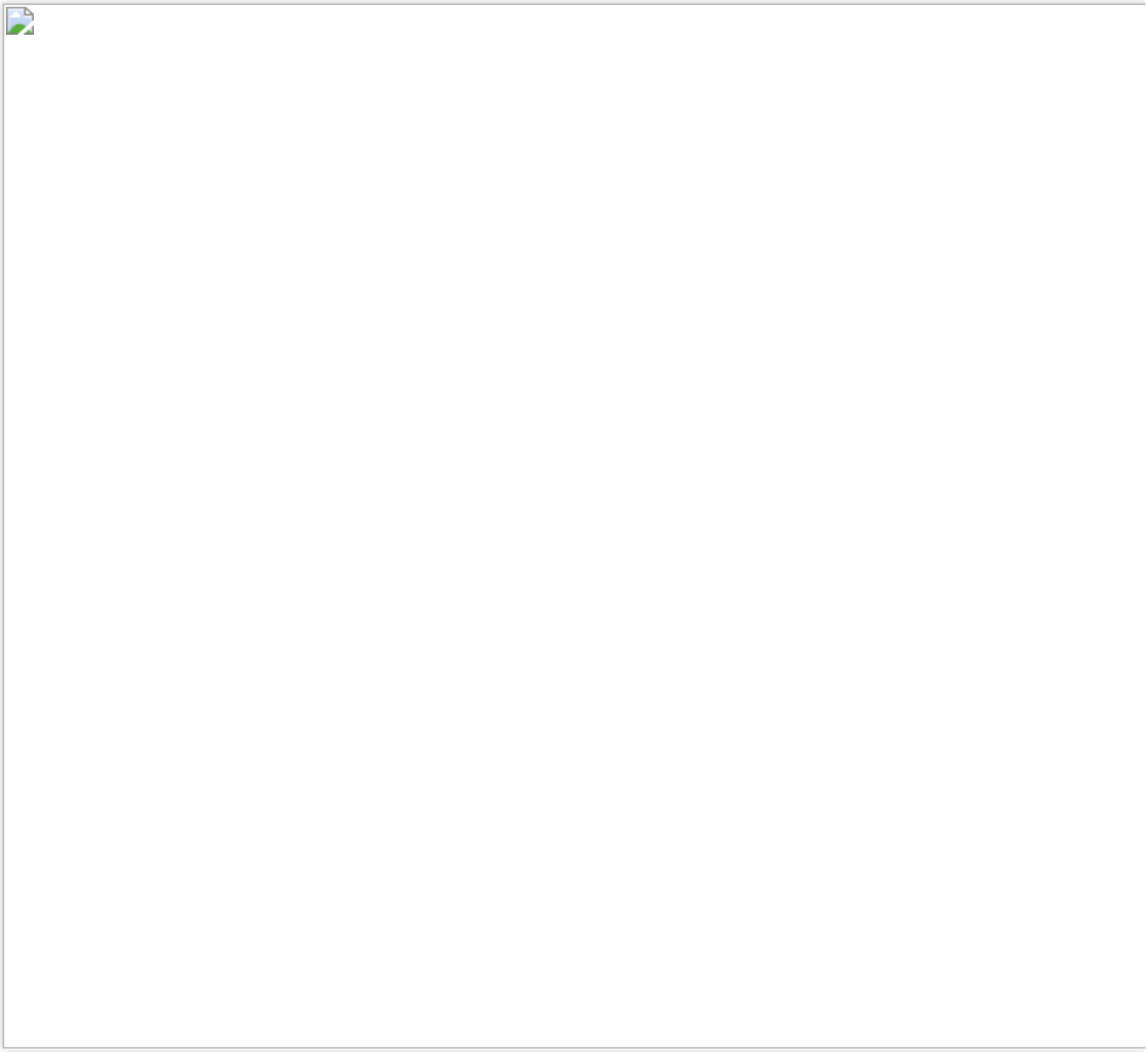
Enter the following command in a terminal window (you need to install Ruby on your Mac first):



```
sudo gem install cocoapods
```

Can `TUICallKit` run in the background?

Yes. If you want `TUICallKit` to run in the background, select your project, under the **Capabilities** tab, toggle on **Background Modes**, and select **Audio, AirPlay, and Picture in Picture**.



Android

Last updated：2024-08-15 14:42:54

The error message "The package you purchased does not support this ability"?

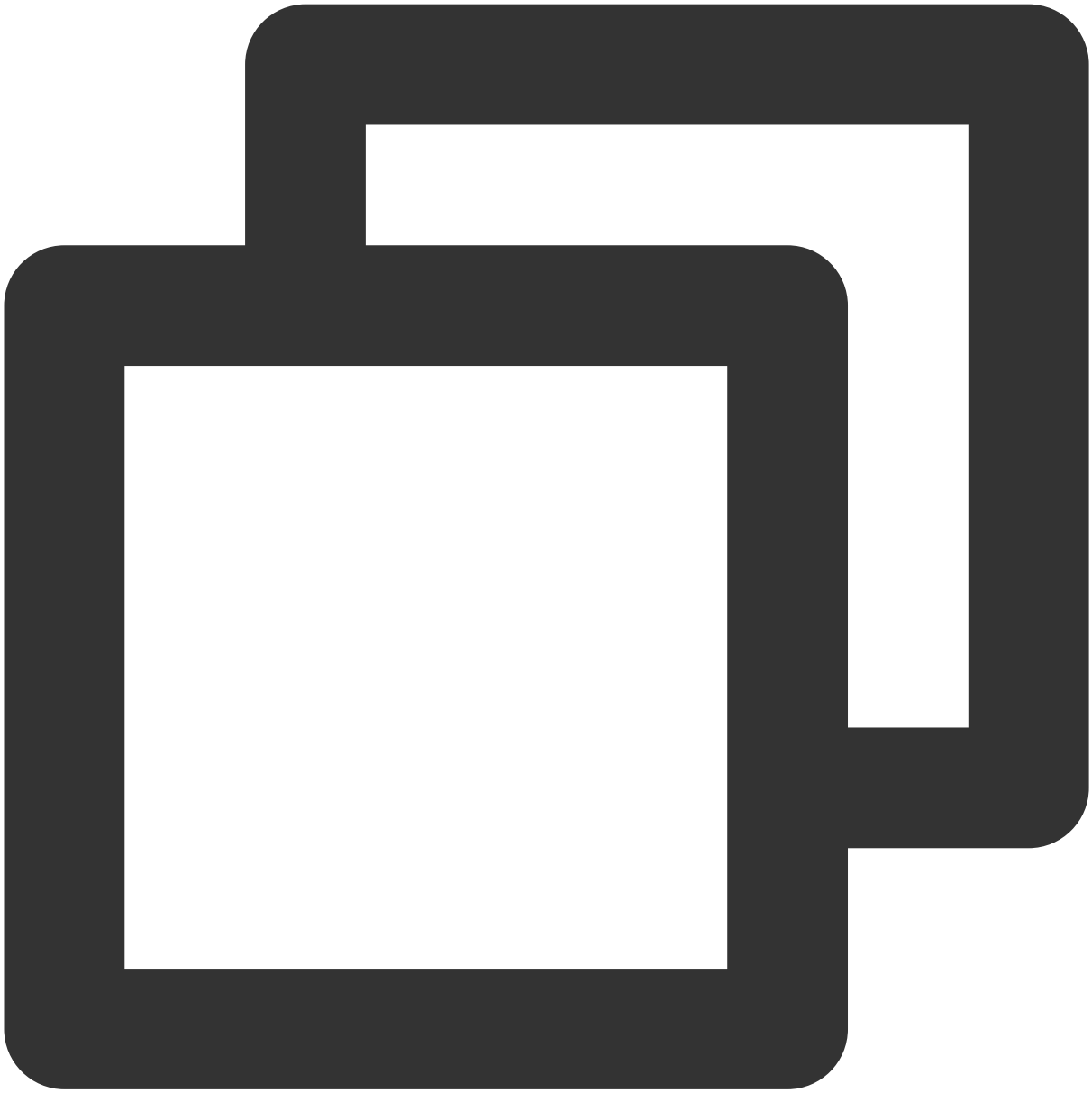
If you encounter the above error message, it is because the audio and video call capability package of your current application has expired or has not been activated. You can refer to the service activation guide to claim or activate the audio and video call capability, and then continue to use the TUICallKit component.

When the application is offline, Will it can pop up the call view?

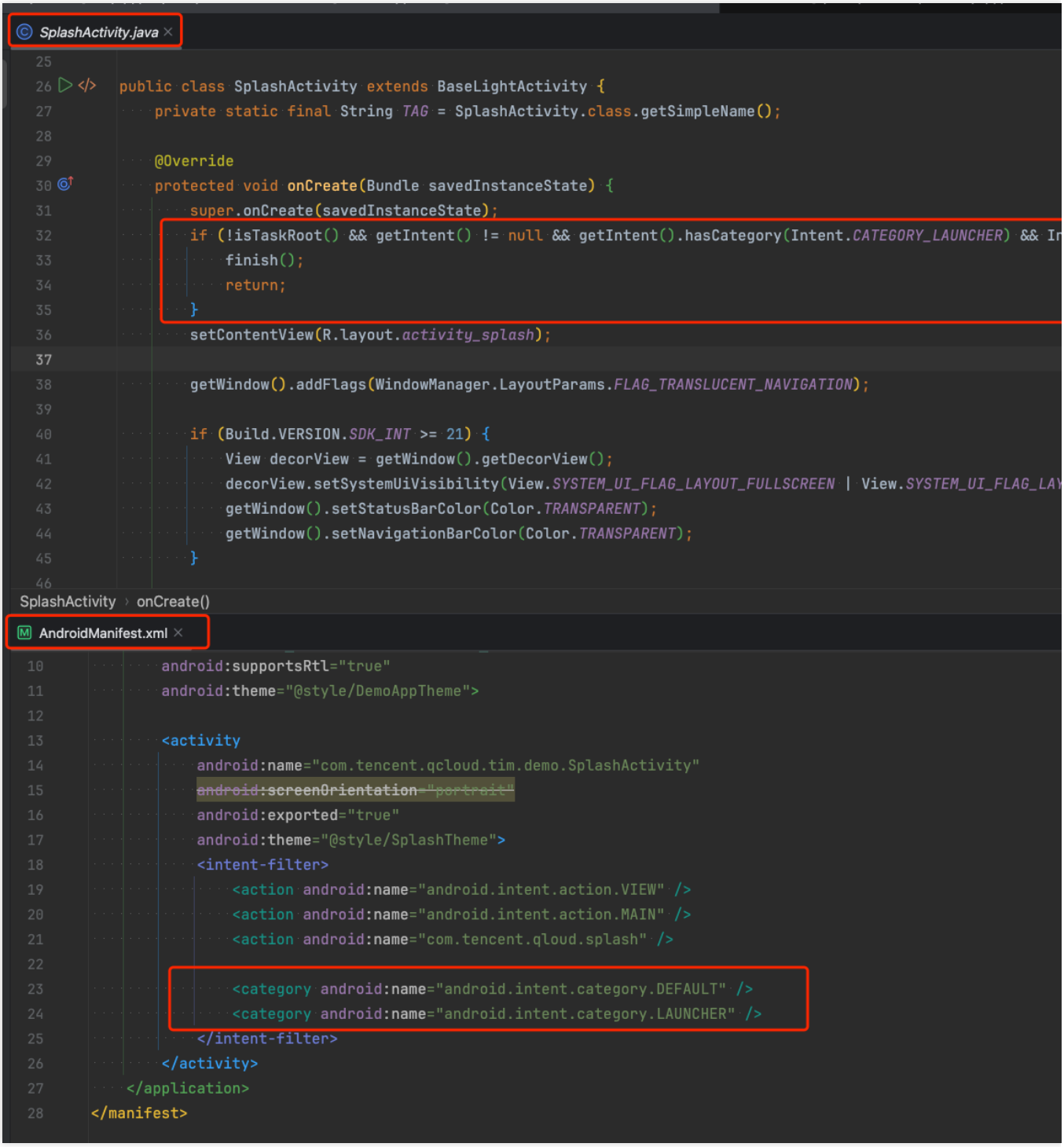
During a one-to-one call, if you come online within the timeout period, an incoming call view will be triggered; For group calls, if you come online within the timeout period, the last 20 unprocessed group messages will be pulled up, and if a call invitation exists, it will trigger an incoming call. The display strategy for incoming calls see: Incoming call display policy for the callee).

When the application is in the background, Will it can pop up the call view?

1. TUICallKit has two different display policy for the callee, see below: Incoming call display policy for the callee.
2. Before version 2.3 of TUICallKit, to automatically bring the application from the background to the foreground, it was necessary to check whether the app had enabled the "Background self-start" or "Floating window" permission. Different manufacturers, or even the same manufacturer with different Android versions, offer varying permissions and permission names for applications. For instance, Xiaomi 6 requires only the "Background pop-ups" permissions , while Redmi needs both the "Background pop-ups" and "Display over other applications" permissions.
How to enable permissions, see below: Relevant permissions enabled.
3. If you encounter the following scenarios where the call interface cannot be pulled up, the reason is that changes in the application's launch stack caused the CallKitActivity interface to be obscured or removed.
Scenario One: After accept call, if app goes to the background and then click the desktop icon to enter foreground, the CallKitActivity disappears.
Scenario Two: If the app is in the background and when you receive a IncomingNotificationView then click the desktop icon to enter the app, the CallKitActivity didn't show and the IncomingNotificationView disappears.
Scenario Three: When the app is in the background and an offline push notification is received, not clicking the push notification but entering the app via the desktop icon will either fail to bring up the CallKitActivity or cause the CallKitActivityto flash momentarily.
In the above scenarios, you need to add the following code to the default Activity of your app. Each app's default Activity may differ. Refer to the AndroidManifest.xml configuration for more details. Taking the launch page SplashActivity for most applications as an example:
Code
Location



```
if (!isTaskRoot() && getIntent() != null && getIntent().hasCategory(Intent.CATEGORY
        && Intent.ACTION_MAIN.equals(getIntent().getAction())) {
    finish();
    return;
}
```



```java
// SplashActivity.java
public class SplashActivity extends BaseLightActivity {
    private static final String TAG = SplashActivity.class.getSimpleName();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (!isTaskRoot() && getIntent() != null && getIntent().hasCategory(Intent.CATEGORY_LAUNCHER) && Ir
            finish();
            return;
        }
        setContentView(R.layout.activity_splash);

        getWindow().addFlags(WindowManager.LayoutParams.FLAG_TRANSLUCENT_NAVIGATION);

        if (Build.VERSION.SDK_INT >= 21) {
            View decorView = getWindow().getDecorView();
            decorView.setSystemUiVisibility(View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN | View.SYSTEM_UI_FLAG_LAY
            getWindow().setStatusBarColor(Color.TRANSPARENT);
            getWindow().setNavigationBarColor(Color.TRANSPARENT);
        }
```

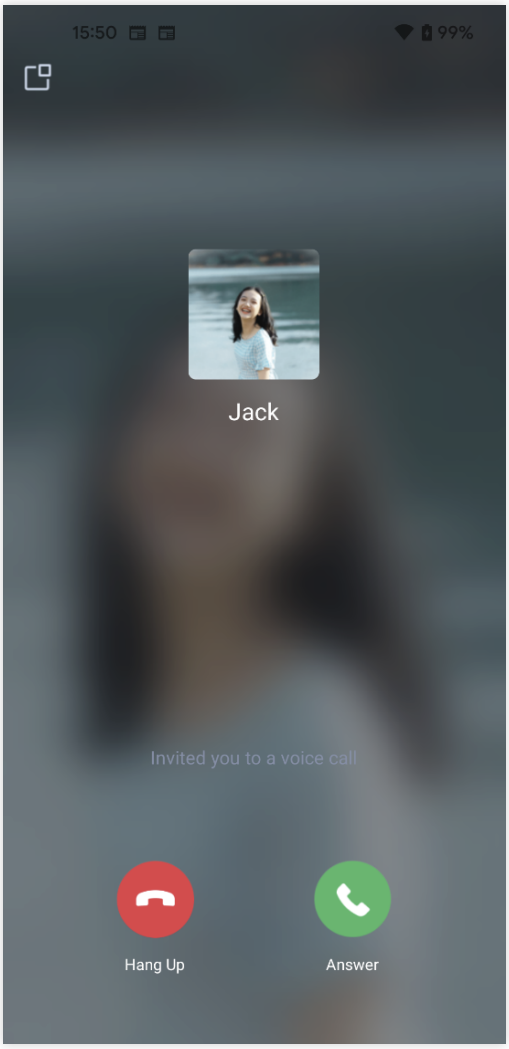
SplashActivity › onCreate()

```xml
<!-- AndroidManifest.xml -->
        android:supportsRtl="true"
        android:theme="@style/DemoAppTheme">

        <activity
            android:name="com.tencent.qcloud.tim.demo.SplashActivity"
            android:screenOrientation="portrait"
            android:exported="true"
            android:theme="@style/SplashTheme">
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <action android:name="android.intent.action.MAIN" />
                <action android:name="com.tencent.qloud.splash" />

                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

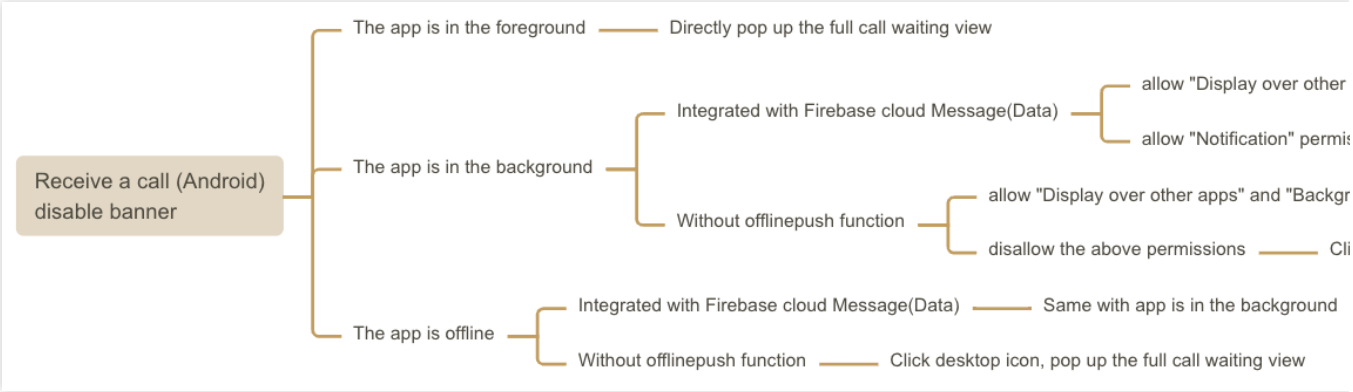
Incoming call display policy for the callee

To make TUICallKit adaptable to different business needs, add product features, and improve user experience, TUICallKit **version 2.3 and above** (Release Log) optimized the call page display pop-up strategy after receiving a call, as detailed below:

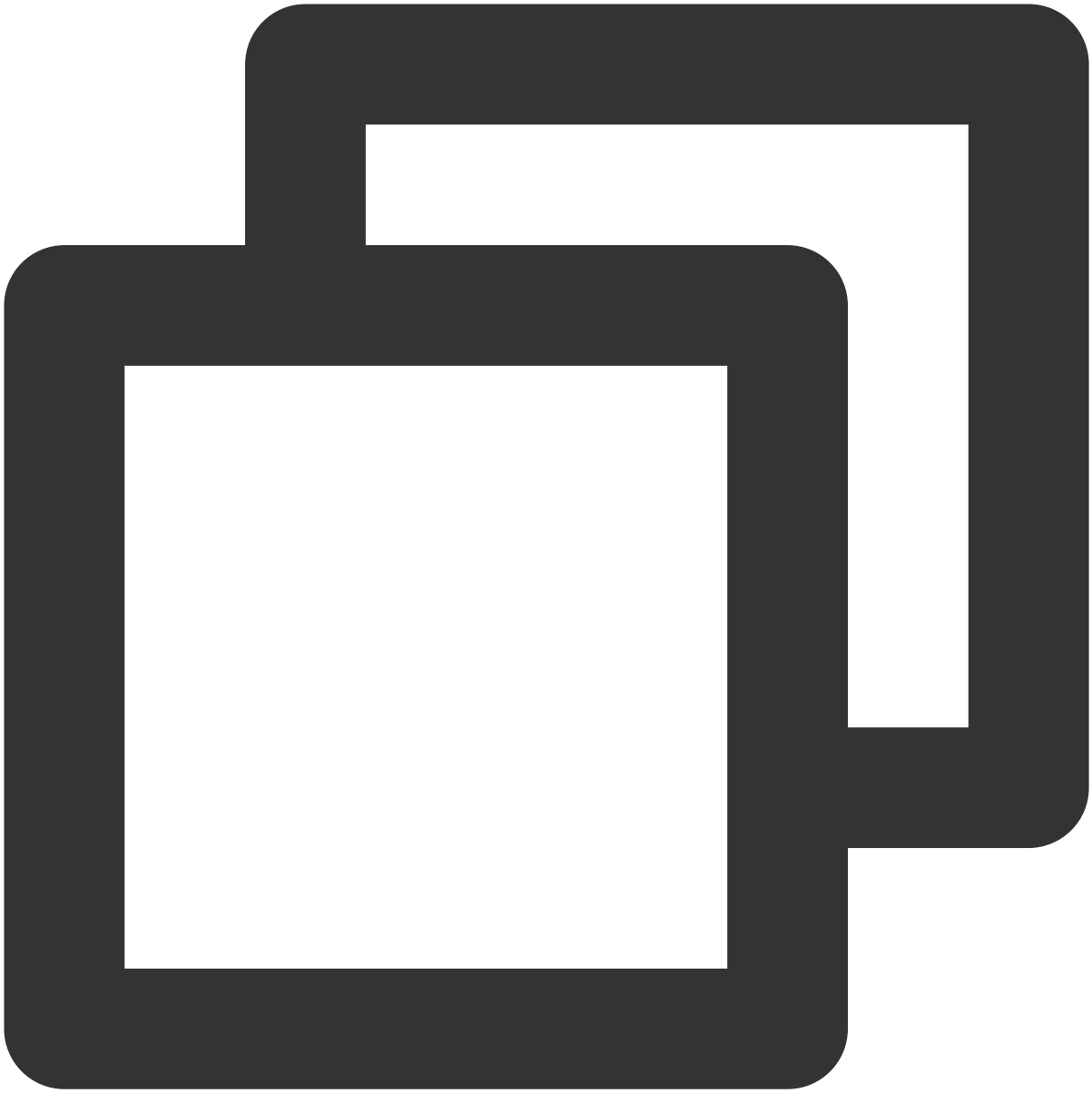
Full screen display of incoming call	Banner display of incoming call



1. If you want the callee to pull up the full-screen call view when receiving a call, you can integrate the tuicallkit-kt code. By default, the incoming call display strategy for the callee is as follows:

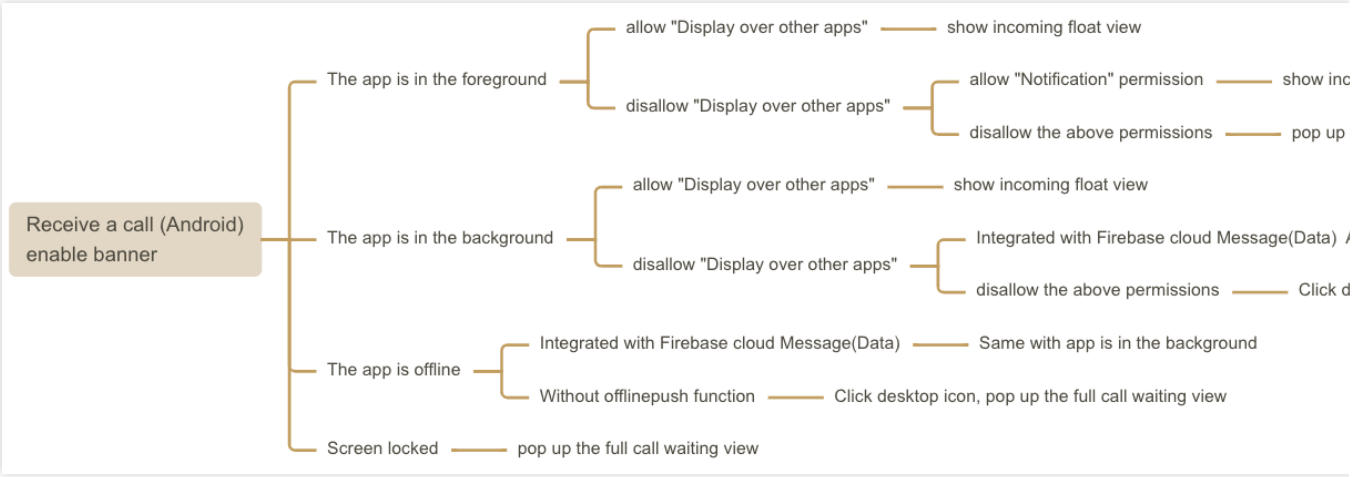


2. If you prefer to show a banner when receiving a call and then pull up the full-screen as needed, you can enable this feature by calling the following interface.



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

Once enabled, the callee's incoming call strategy is as shown below, as long as the **floating window permission** is enabled, the call banner will be displayed as much as possible.



Note:

For information on how to enable related permissions, see below: Relevant permissions enabled.

If the incoming call interface doesn't display according to the above strategy, please filter the `onCallReceived` log to check if a call invitation was received.

Relevant permissions enabled

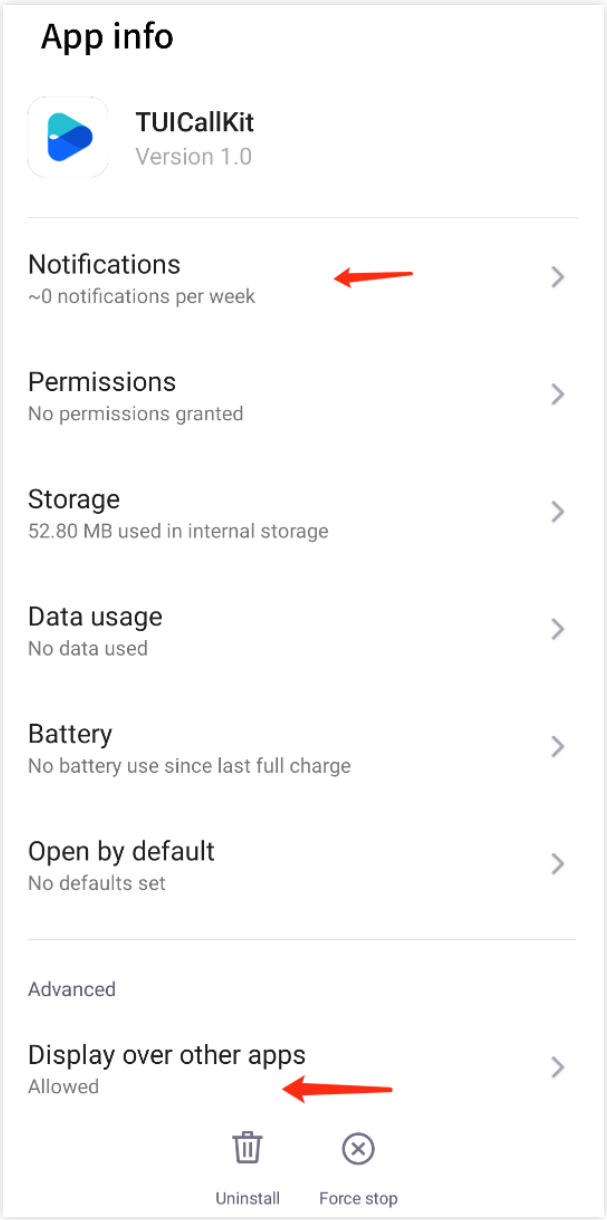
To ensure a good calling experience, it is recommended to enable "notification" permissions, "Display over other apps(Floating window)" and "Background pop-ups" permissions in your application. The specific methods are as follows:

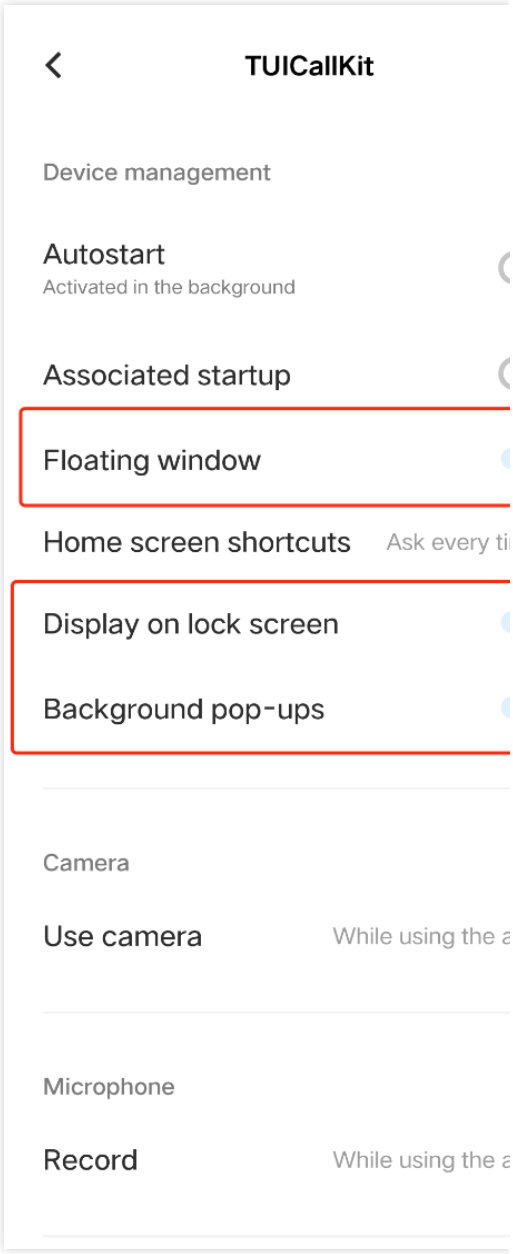
Manually Enabled

Code guidance

After the application is installed, you can long-press the application Icon, select "Application Information", then enable "notification" permission, "Display over other applications" and "Background pop-ups" permissions. Alternatively, you can go to `Mobile Phone > System Settings > Application Management > Application` to manually enable these permissions.

Pixel 4a	VIVO

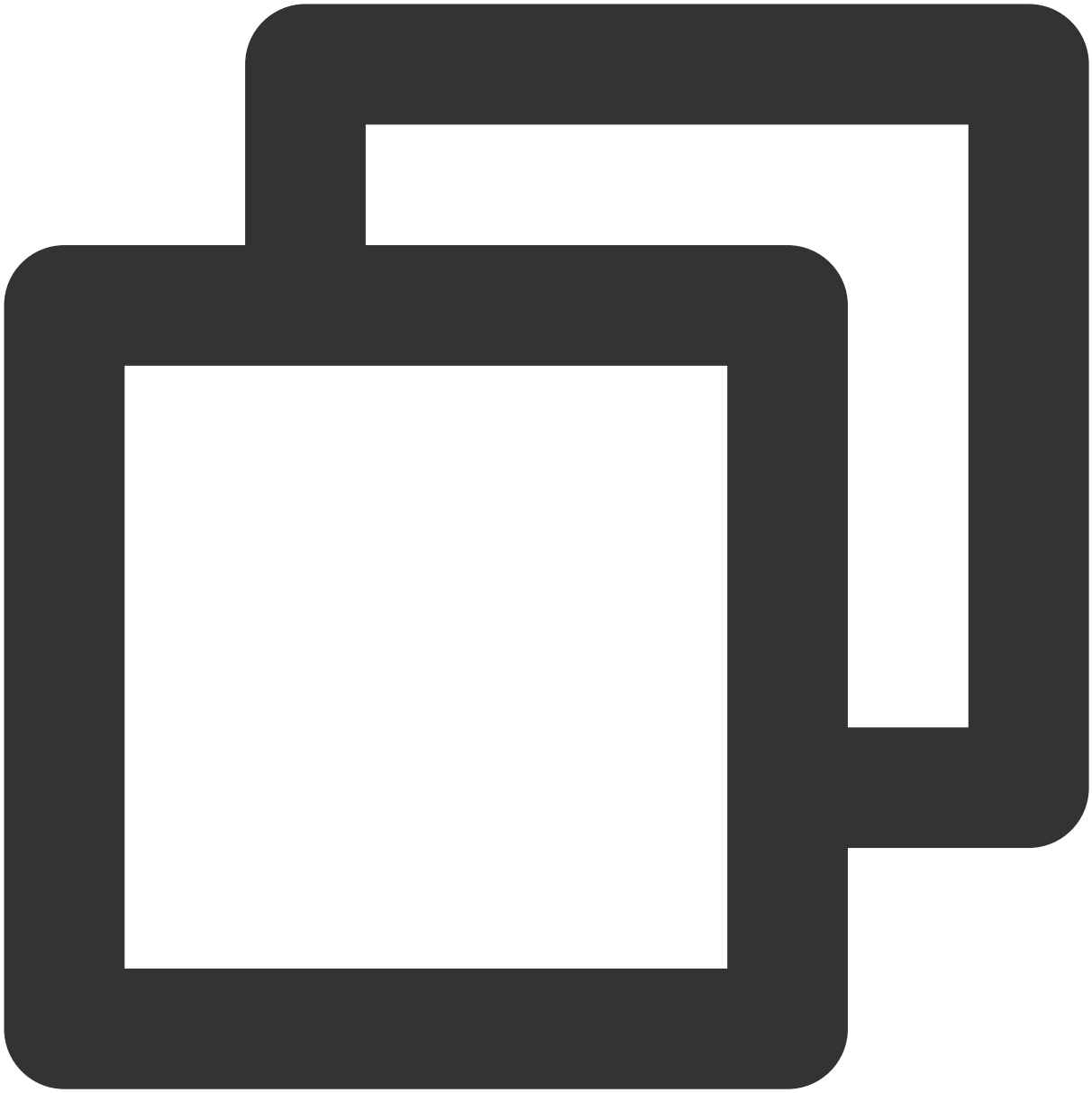




Notification permissions: For showcasing push notifications: please refer to Notification runtime permissions and Request runtime permissions and implement according to business needs.

Floating window permission: Used for displaying custom incoming call notifications and call floating windows.

Background pop-ups: Used to pop up the interface when the application is in the background (for example: on a VIVO phone).



```
fun requestPermission(context: Context?) {
    //In TUICallKit,Please open both OverlayWindows and Background pop-ups permissi
    PermissionRequester.newInstance(
         PermissionRequester.FLOAT_PERMISSION, PermissionRequester.BG_START_PERMISS
        .request()
}
```

Web

Last updated：2024-06-27 16:54:10

What should I do if I receive the error message "The package you purchased does not support this ability"?

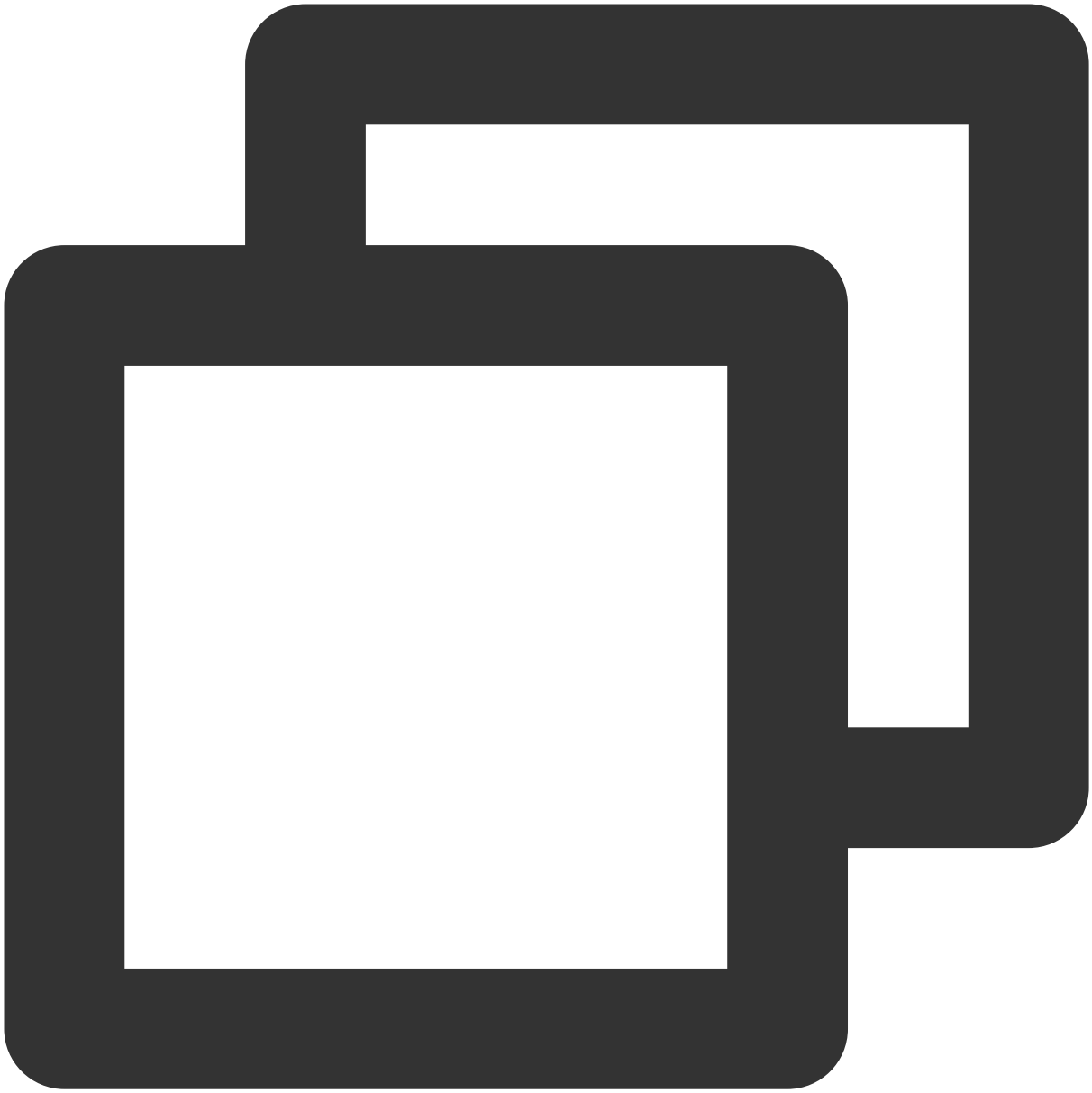
If you encounter the error message above, it's because the Audio and Video Calling Capability Package of your current application has either expired or not been activated. Please refer to Activate Service to claim or activate the audio and video calling capability, and continue to use the TUICallKit Component.

How do I purchase a package?

Please follow the link Purchase Official Version.

How can I generate a UserSig?

UserSig is a type of security signature designed by Tencent Cloud for its cloud services. It serves as a login credential, derived from the encrypted combination of information such as SDKAppID and SecretKey.
Method 1: Accessing from the control panel, refer to How to Obtain a Temporary UserSig.
Method 2: Deploying a temporary generation script.
Warning:
This approach involves configuring the SecretKey within the front-end code. Regrettably, in this method, the SecretKey can be easily decrypted through reverse engineering. In the event of your key being exposed, attackers can usurp your Tencent Cloud traffic. Therefore, **this method is only suitable for local functional debugging**. For a production environment, please refer to Method 3.
For easier initial debugging, `GenerateTestUserSig-es.js` in the `genTestUserSig(params)` function can be used temporarily to calculate userSig, for instance:



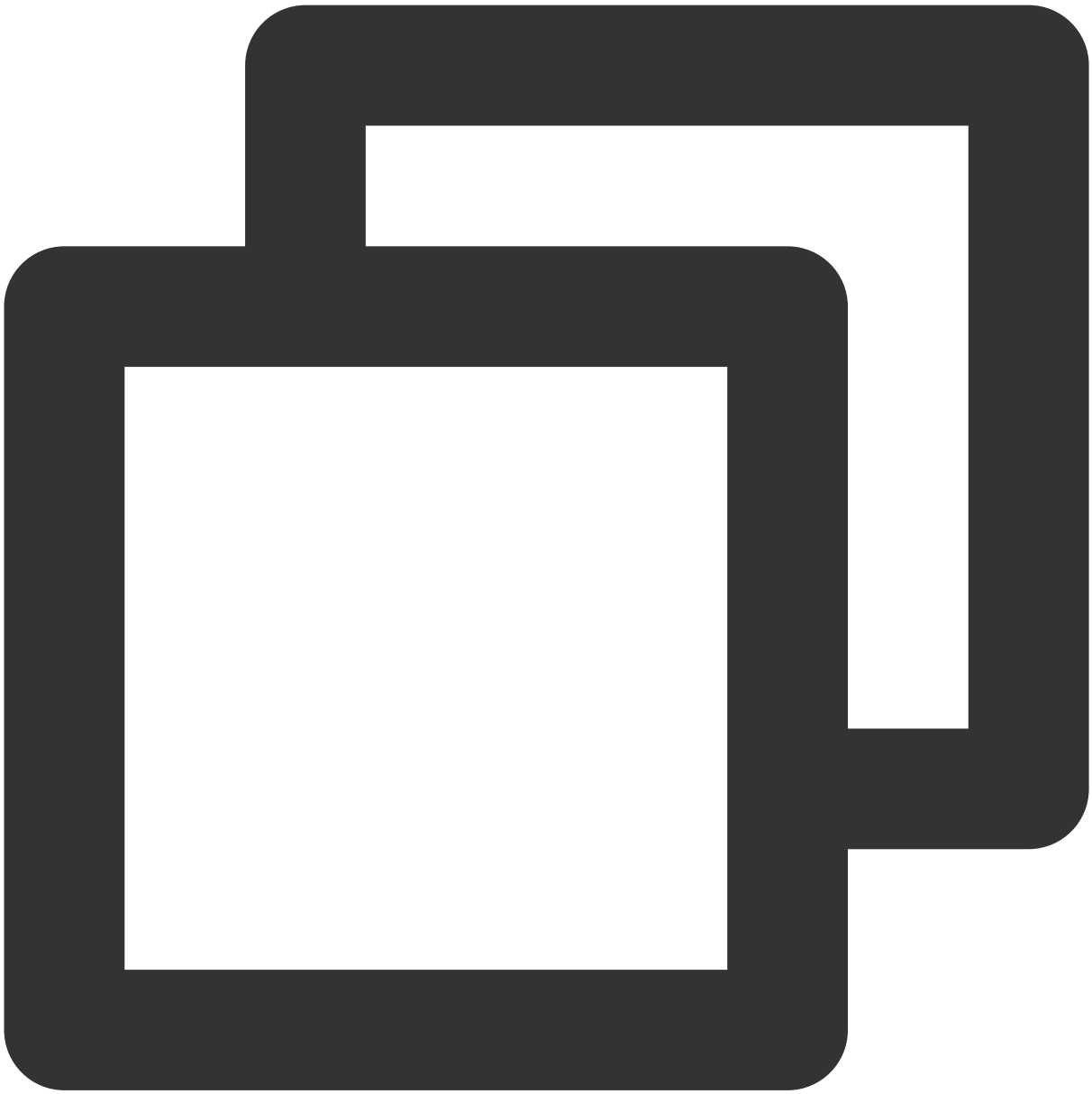
```
import { genTestUserSig } from "./debug/GenerateTestUserSig-es.js";
const { userSig } = genTestUserSig({ userID: "Alice", SDKAppID: 0, SecretKey: "YOUT
```

Method 3: Use in official environment.

The correct method of issuing UserSig is to integrate the calculation code of UserSig into your server-side, providing project-specific interfaces. When UserSig is needed, your project can launch requests to the business server to obtain dynamic UserSig. For detailed information, please see Generating UserSig on Server-side.

How is the groupID generated in a group call?

The generation of groupID requires integration of the @tencentcloud/chat package. For specifics, refer to the createGroup API; below is an example of the code to generate groupID.



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

const userIDList: string[] = ['user1', 'user2'];
async function createGroupID() {
  const chat = Chat.create({ SDKAppID });
  const memberList: any[] = userIDList.map(userId => ({ userID: userId }));
  const res = await chat.createGroup({
    type: Chat.TYPES.GRP_PUBLIC,
```

```
    name: 'WebSDK',
    memberList
  });
  return res.data.group.groupID;
}
```

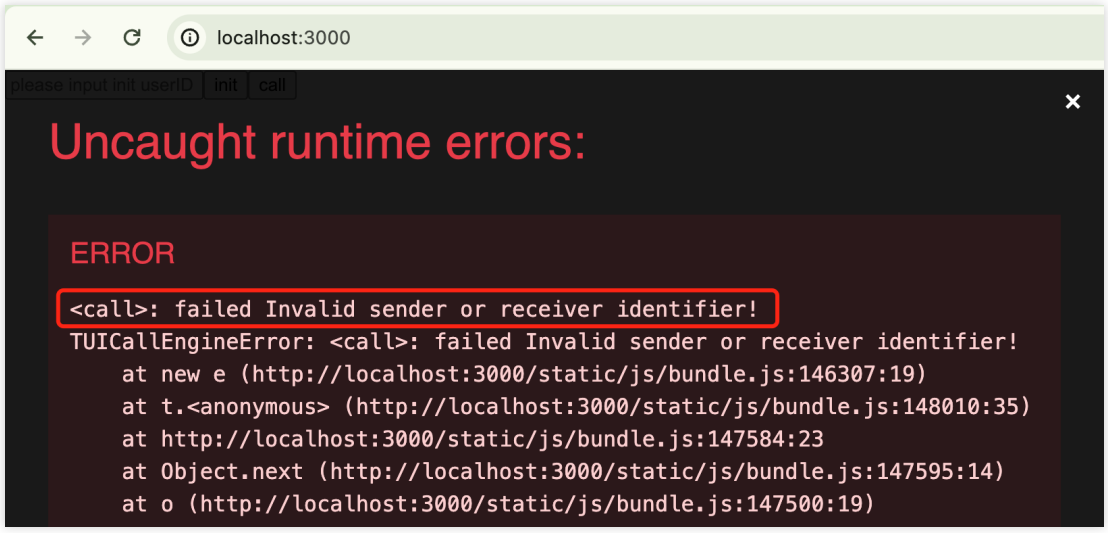
How do I create a userID?

Unique identifier of the user, `defined by you` **,** it is allowed to contain only upper and lower case letters (a-z, A-Z), numbers (0-9), underscores, and hyphens.

Signing in once with userID and userSig will automatically create the user.

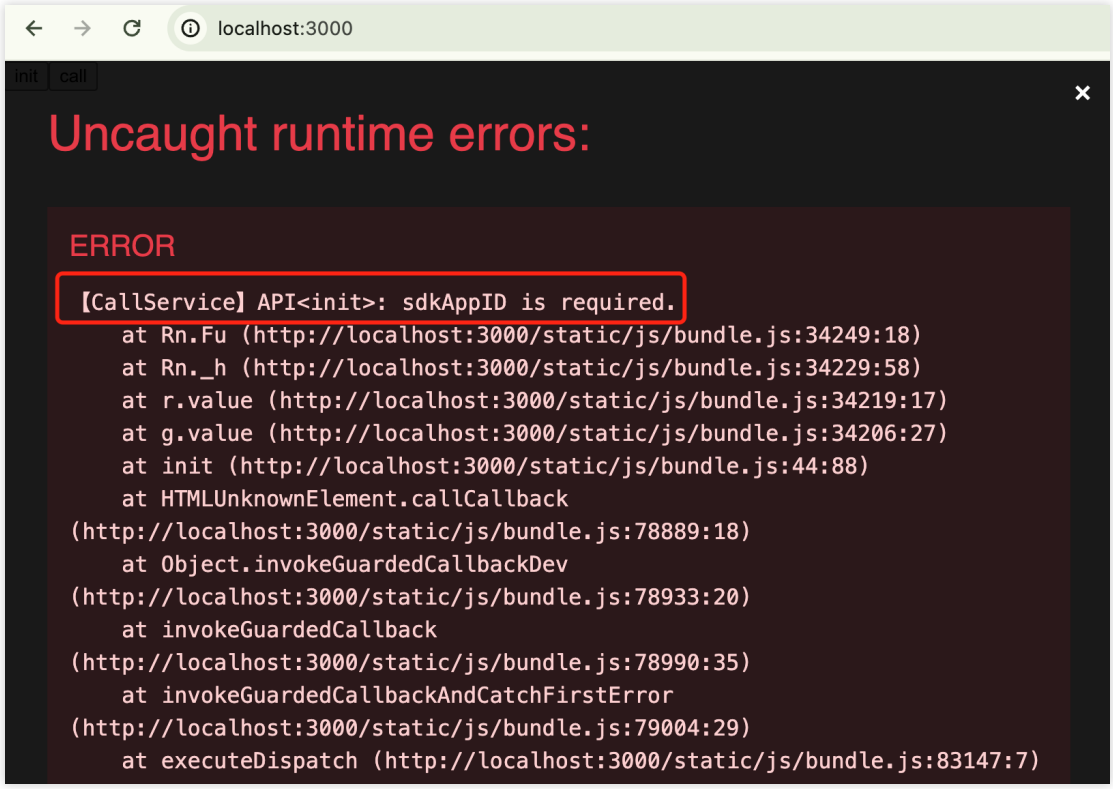
Create and get through the Tencent RTC Console.

Error <call>: failed Invalid sender or receiver identifier ?



This error occurs because the userID you called does not exist; ensure the userID has signed in at least once. See How do I create a userID for more details.

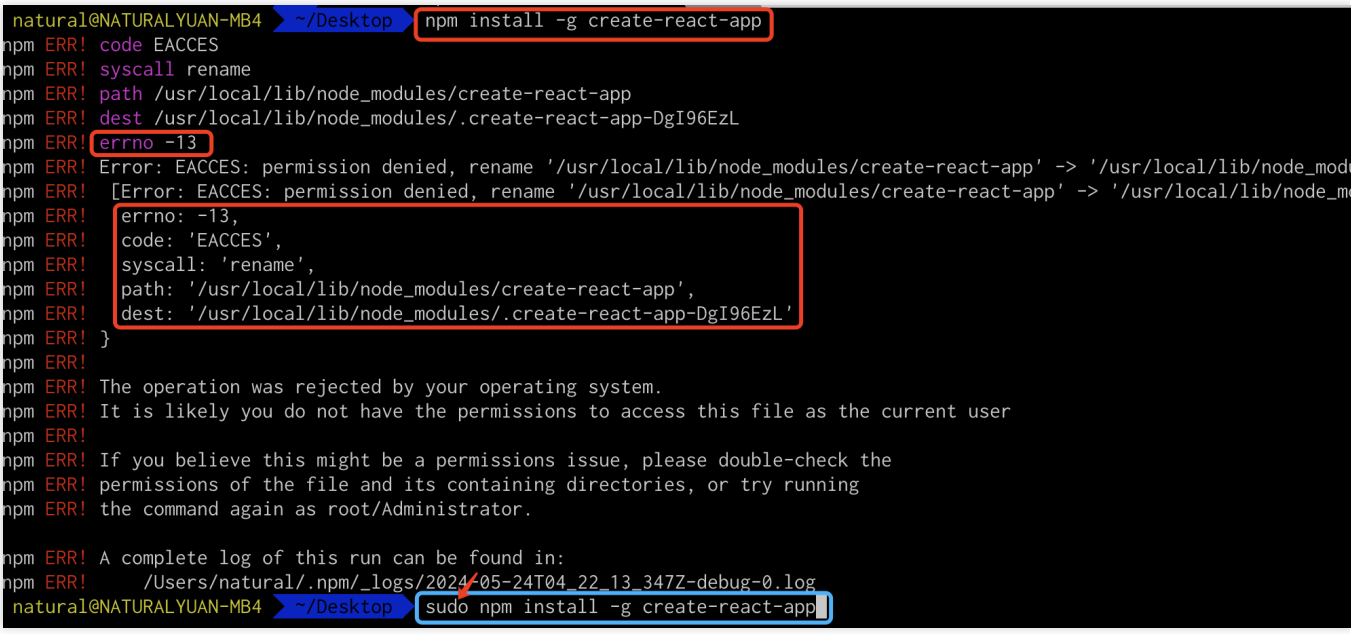
Error [CallService]API<init>: sdkAppID is required ?



This error occurs because you did not fill in the SDKAppID information during TUICallKitServer.init/ `GenerateTestUserSig.genTestUserSig`

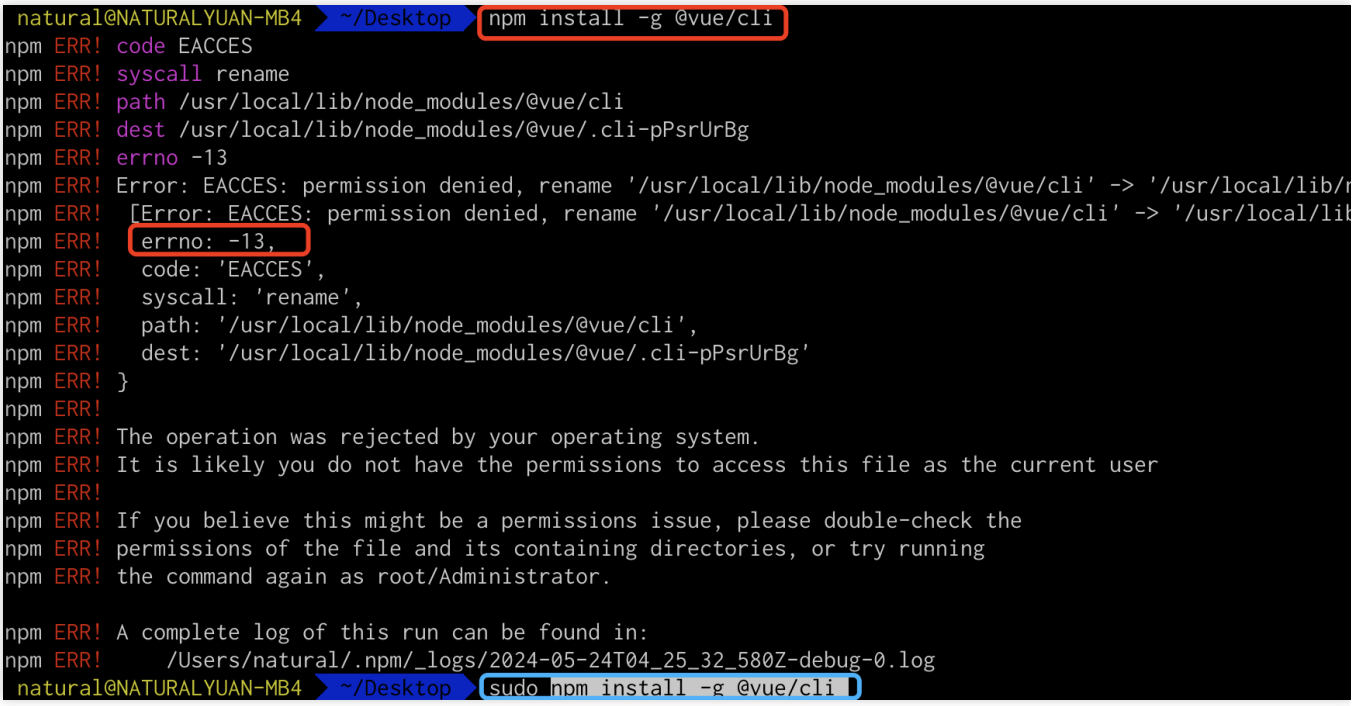
Please obtain and fill in from the Tencent RTC Console.

npm install -g create-react-app error: errno -13 ?



If this error occurs, it is because the current user does not have permission to globally install scaffolding. Please use `sudo npm install -g create-react-app` .

npm install -g @vue/cli package error: errno -13？



If this error occurs, it is because the current user does not have permission to globally install scaffolding. Please use `sudo npm install -g @vue/cli`.

All Platform

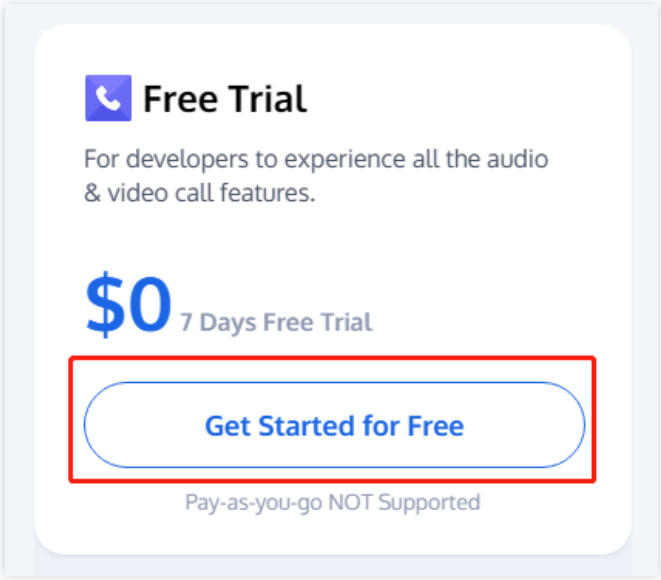
Last updated：2024-07-23 18:11:32

Is there an error message indicating that the audio and video call function is not enabled?

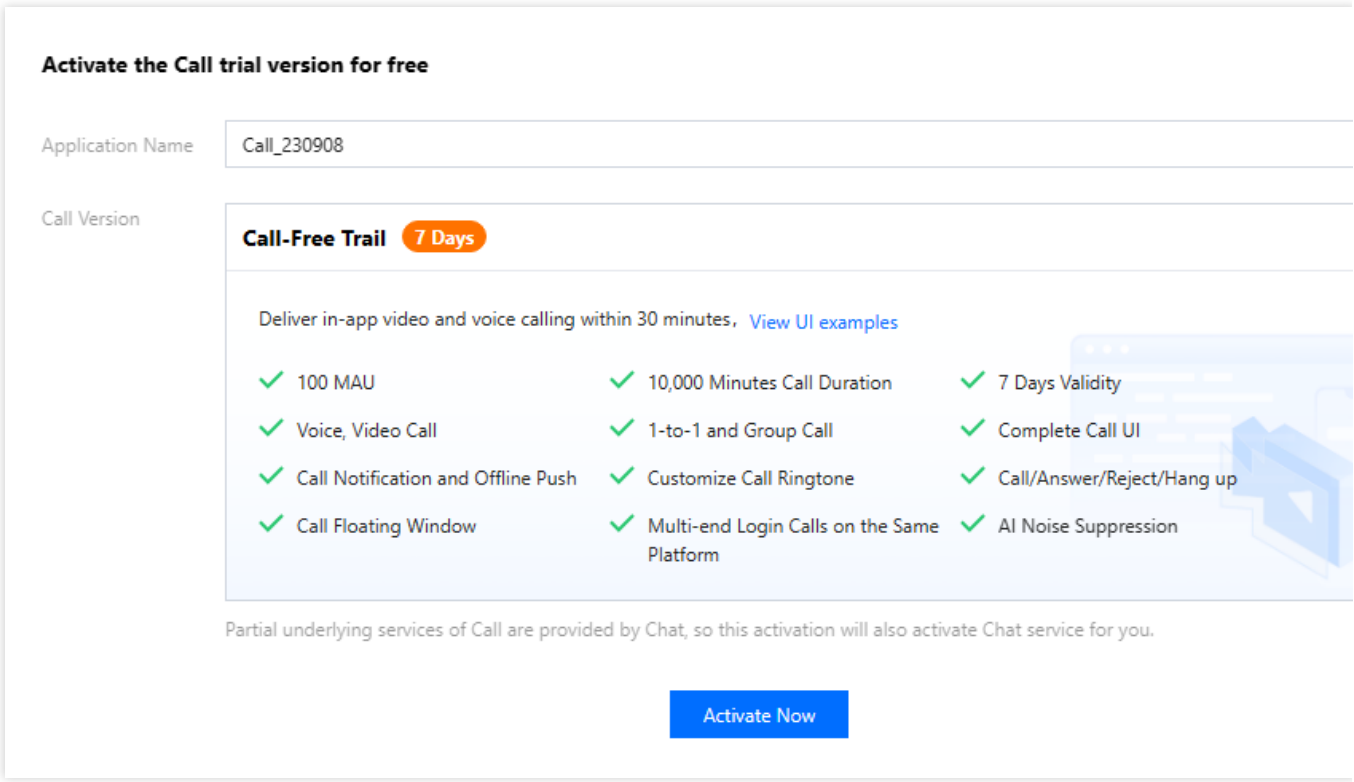
You can choose the Free Trial version (7 days trial) or purchase the official version.

The Free Trail

1. Go to the price page, read the specific services included in the Free Trail package, click **Get Started for Free**, and follow the documentation prompts to enter the console.



2. Read the **Call-Free Trail** package information in the console pop-up window and click **Activate Now** to activate the 7-day free trial of TUICallKit.

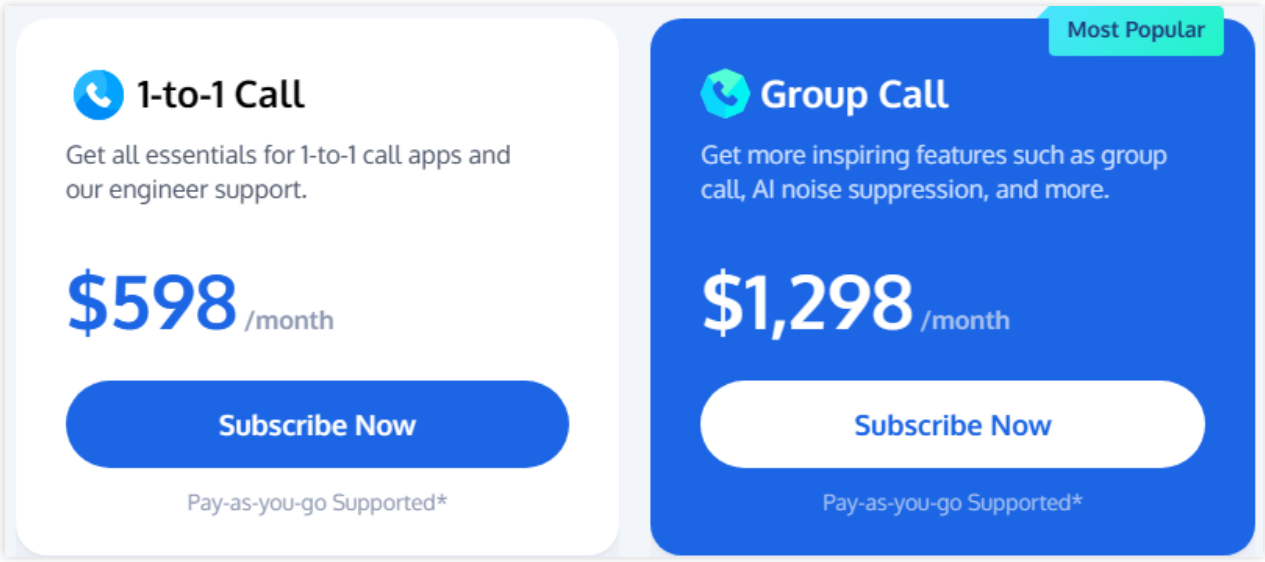


Note：

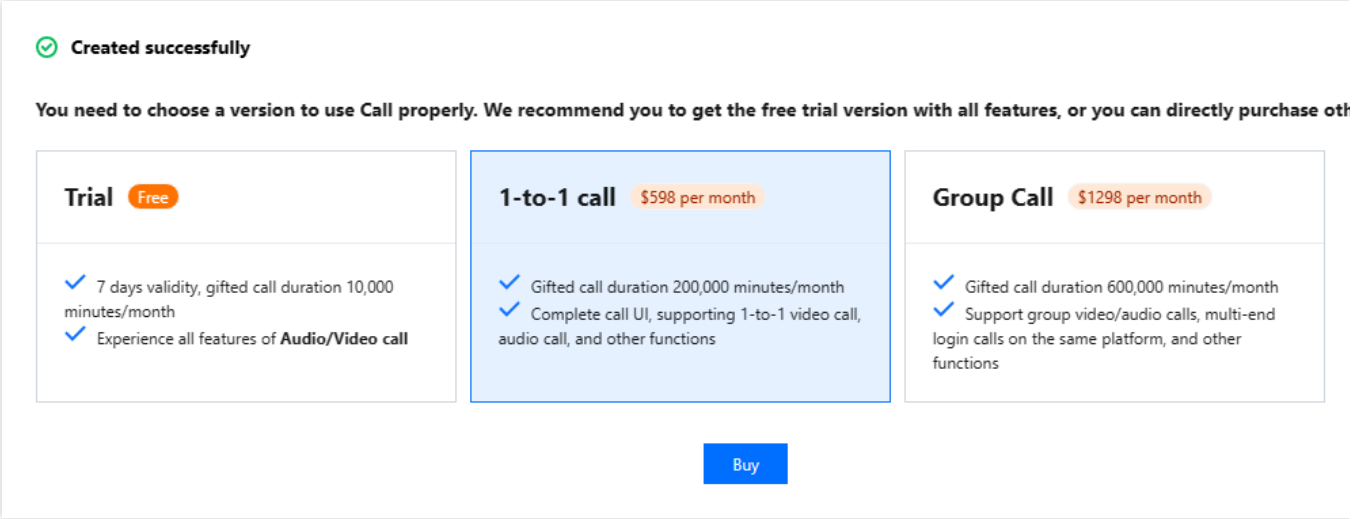
Partial underlying services of Call are provided by Chat, so this activation will also activate Chat service for you.

The Official Version

1. Go to the price page, read the specific services included in the Free Trail package, click **Subscribe Now**, and follow the documentation prompts to enter the console.



2. Click **Create application** in the console, and select **1-to-1 Call** or **Group Call** in the pop-up window after successful creation to open official service of TUICallKit and become the official version.



The error inviteID is invalid or the invitation has been processed

The inviteID has been cancelled by the calling party or the invitation has been accepted. Because signaling is affected by many factors such as network bandwidth, in extreme cases (repeated calls on the accept and reject interfaces), an error that inviteID is invalid may result.

The message sender or receiver UserID is invalid or does not exist

The user you are calling has not logged in to the Chat service, so the call fails. Please try to call a user who has logged in to the Chat service.

Can TUICallKit use TRTC without introducing Chat SDK?

No, all components of TUIKit use Tencent Cloud Chat SDK as the basic service for communication, such as call dialing signaling, call busy line signaling and other core logic, if you have purchased other Chat products, you can also refer to TUICallKit logic to adapt.

How to buy audio and video call plan?

Please refer to the purchase link TRTC Call Monthly Packages, if you have other questions, please click the top of the page Talk to us for pre-sale package consultation.

How do I get the Roomid of the call?

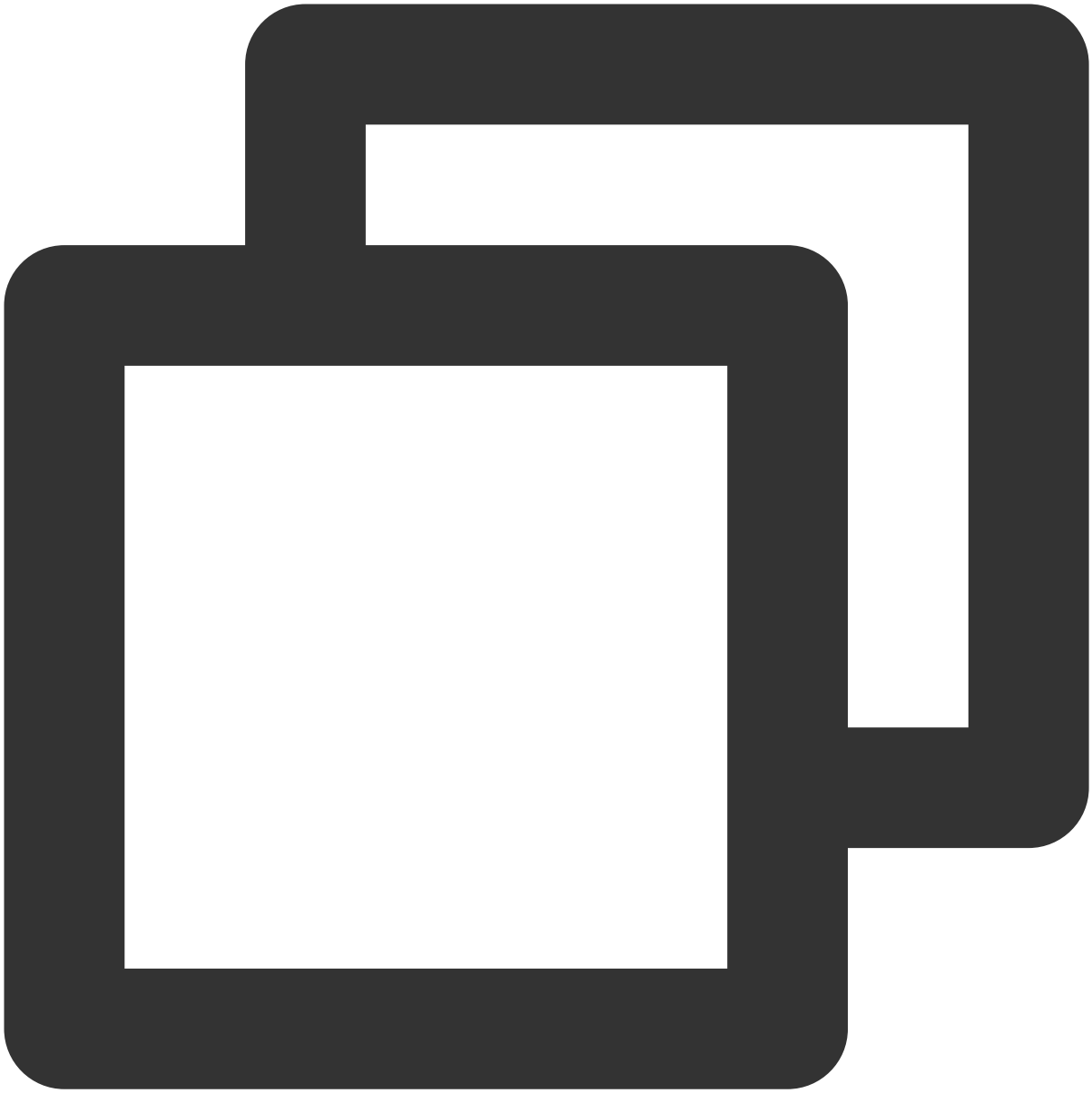
After the call is connected, you can use onCallBegin to return the roomid field to obtain the information.

Flutter

Last updated：2024-08-15 14:42:54

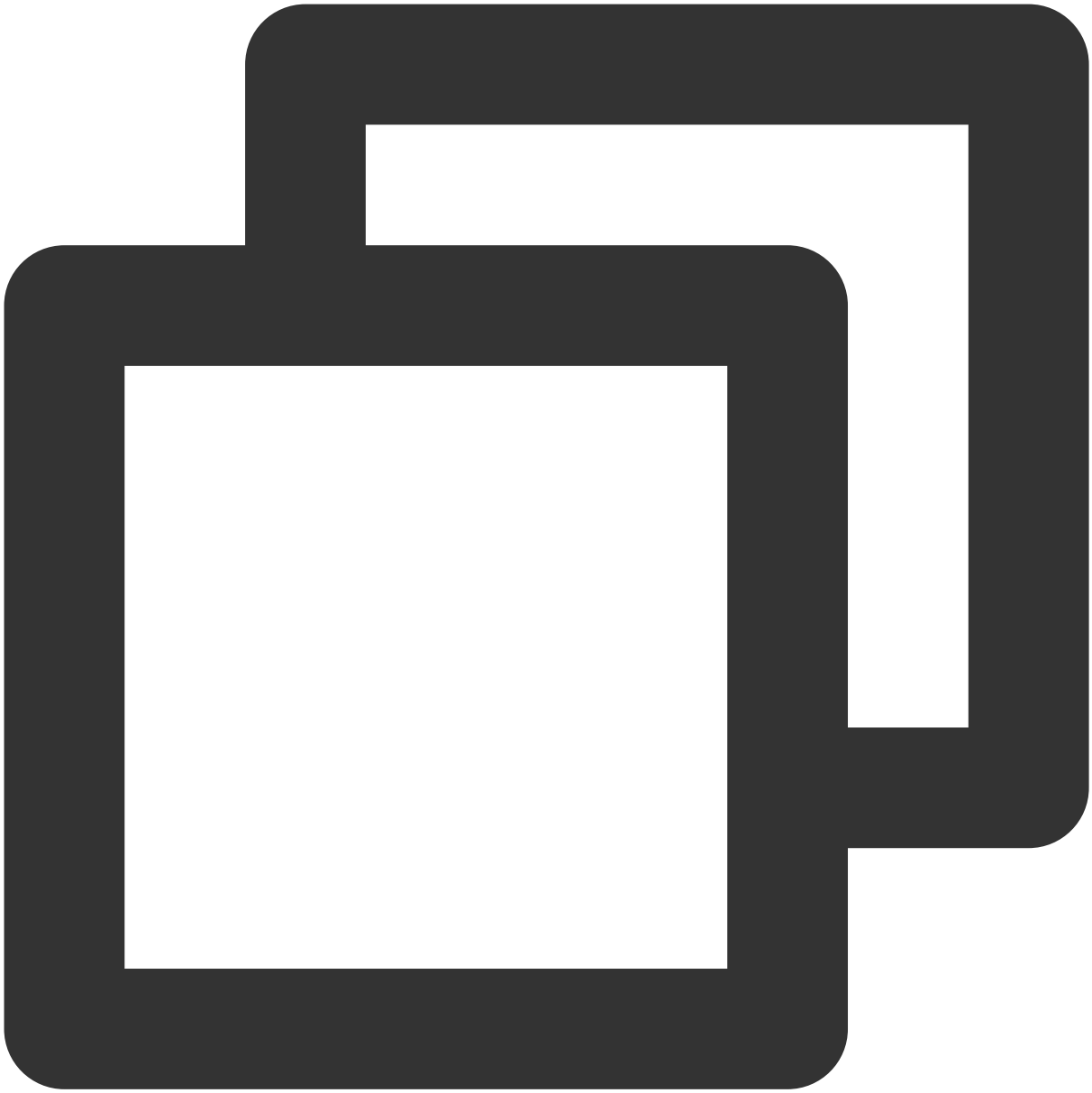
1. Integrate `tencent_calls_uikit` and `tencent_trtc_cloud` at the same time, or integrate `tencent_calls_uikit` and `live_flutter_plugin` at the same time, and the symbol conflict will occur, how to solve it?

Detials: When introducing flutter "tencent_calls_uikit" into our existing project, Android builds APK with the following error:



```
Duplicate class com.tencent.liteav.LiveSettingJni found in modules jetified-LiteAVS
(com.tencent.liteav:LiteAVSDK_Professional:10.7.0.13053) and jetified-LiteAVSDK_TRT
(com.tencent.liteav:LiteAVSDK_TRTC:10.3.0.11225)
```

The following error occurs when `pod install` is executed on iOS:

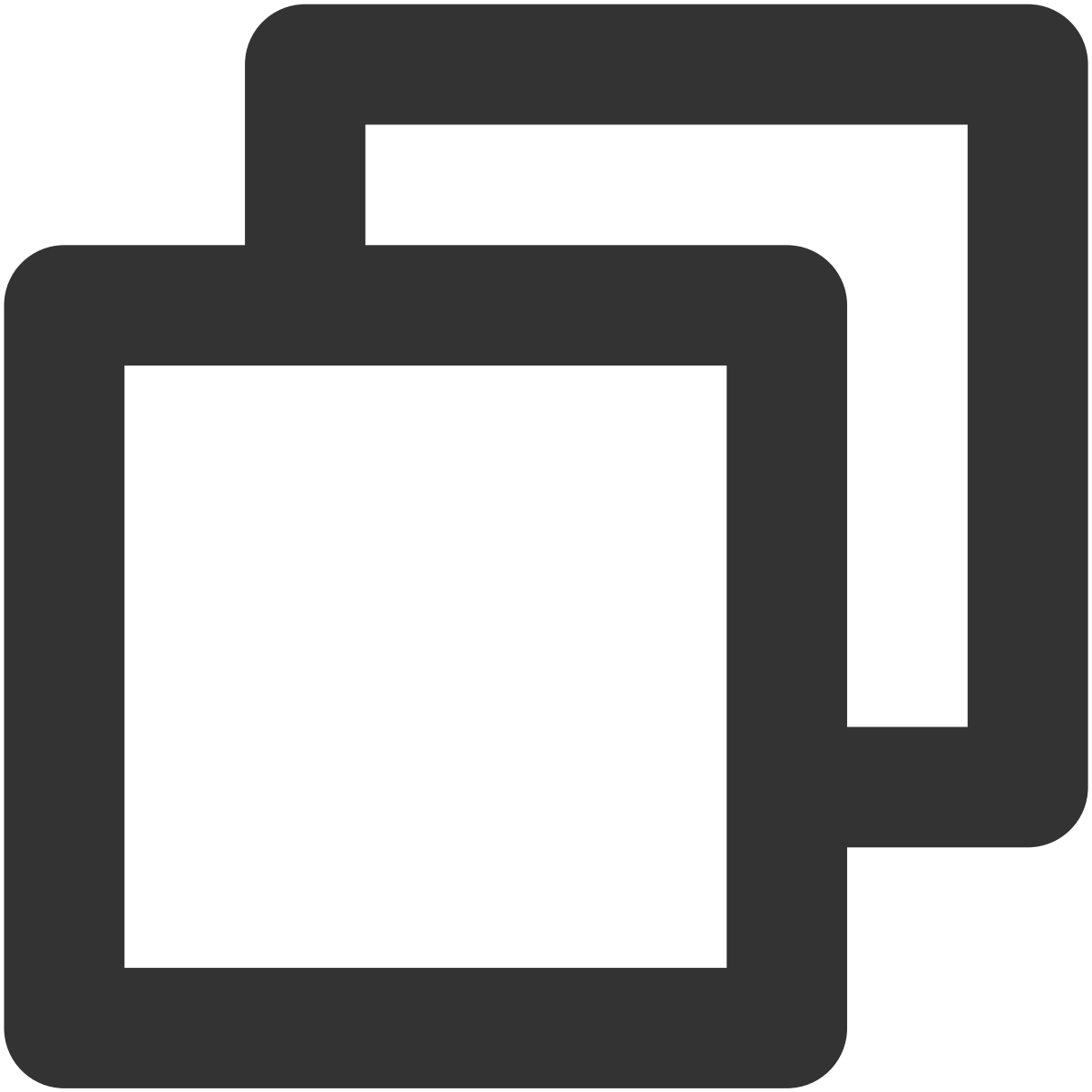


```
[!] The 'Pods-Runner' target has frameworks with conflicting names: txsoundtouch.xc
```

The issue arises because you are using tencent_calls_uikit and tencent_trtc_cloud which depend on the Pro and Lite versions of TRTC Android SDK , respectively. We have resolved this issue in the latest version. You just need to upgrade tencent_calls_uikit and tencent_trtc_cloud to the latest version.

2. Flutter Android does not add confusion Settings, how to set?

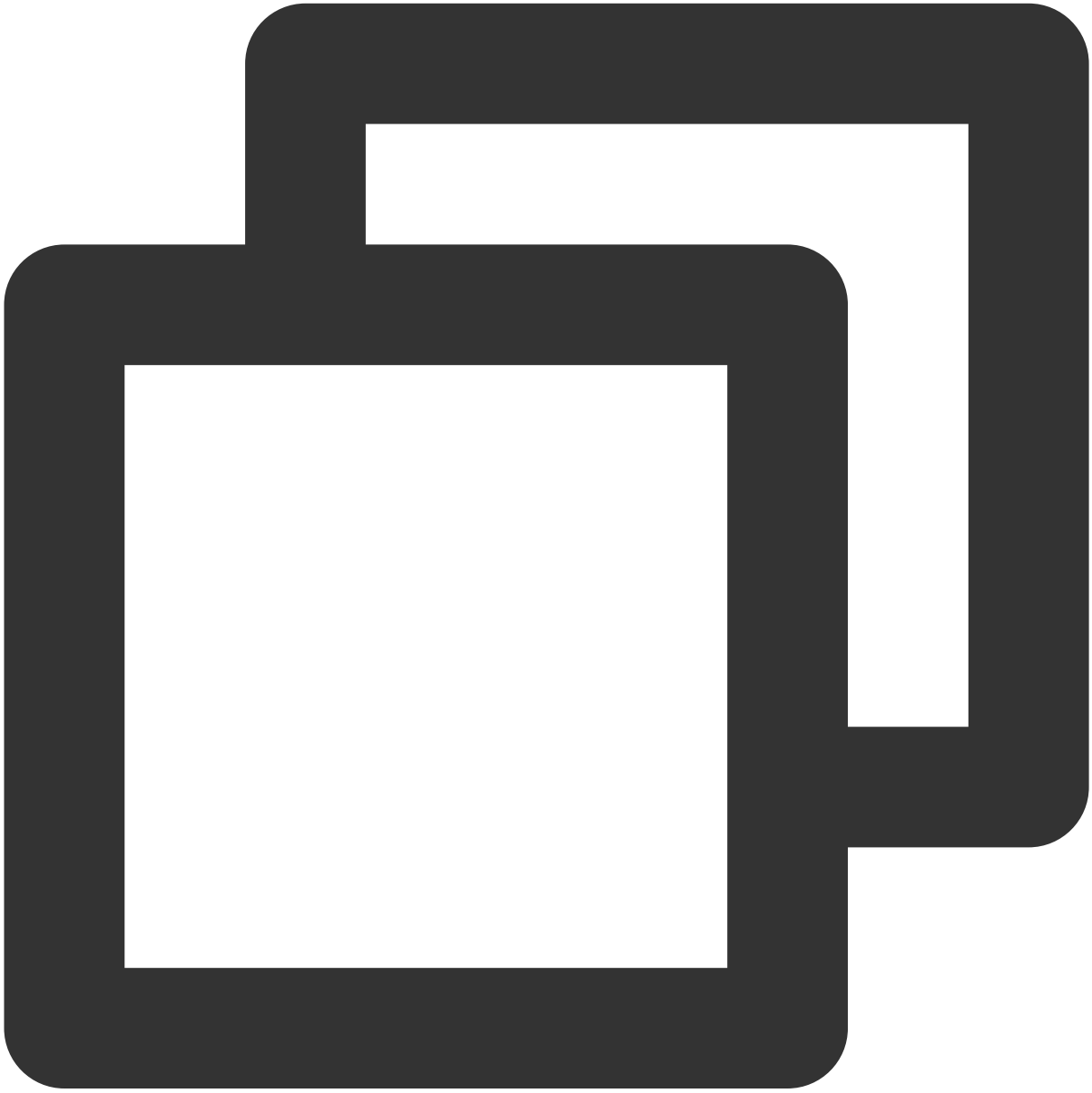
If you need to compile and run on the Android platform, because we use the reflection feature of Java inside the SDK, we need to add some classes in the SDK to the list of non-obviation, so you need to add the following code in the proguard-rules.pro file:



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

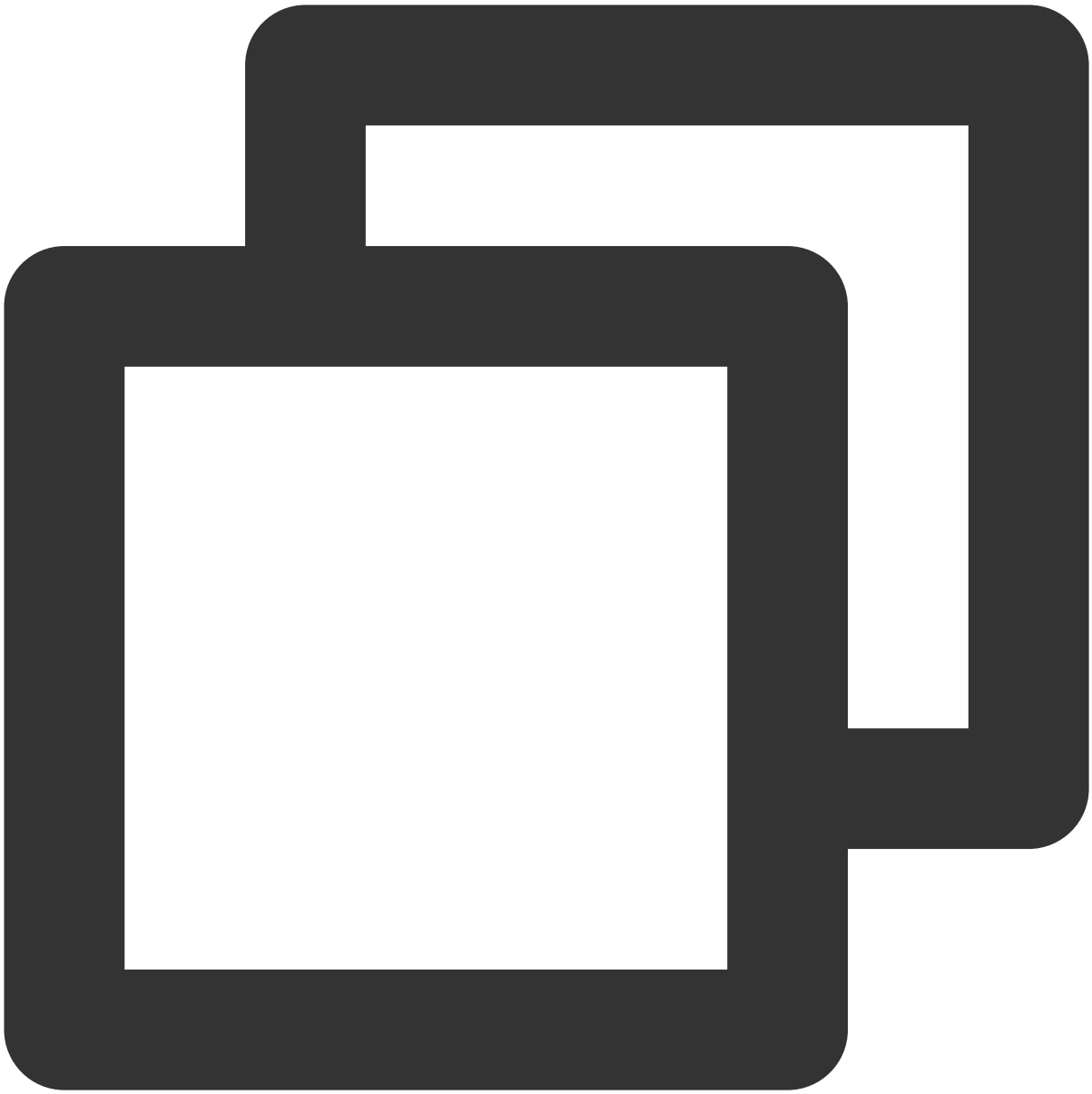
3. How can I fix a compilation error or page failure caused by an upgrade from a version earlier than 1.8.0 to 1.8.0 or later?

If you are upgrading from 1.8.0 or below to 1.8.0 or above, you need to check that the following steps are normal:
1. Add `navigatorObservers` to `MateriaApp` . The purpose is to navigate to the TUICallKit page when you receive a call invitation. Example code is as follows:



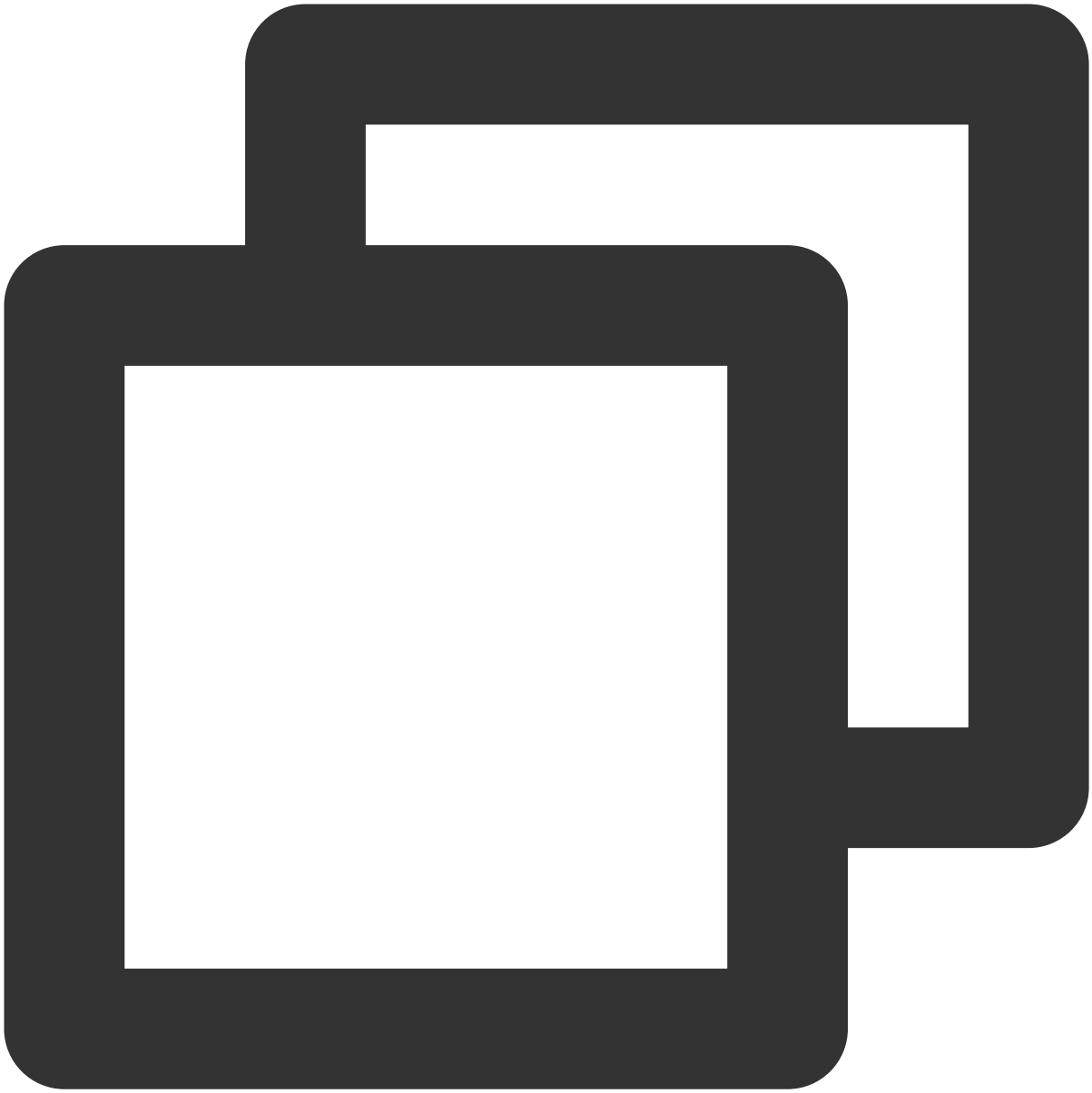
```
import 'package:tencent_calls_uikit/tuicall_kit.dart';

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

2. `tencent_calls_engine` Import files in plug-ins are uniformly replaced with new ones.



```
import 'package:tencent_calls_engine/tuicall_engine.dart';
import 'package:tencent_calls_engine/tuicall_observer.dart';
import 'package:tencent_calls_engine/tuicall_define.dart';
```

The above code block content is replaced with the following:



```
import 'package:tencent_calls_engine/tencent_calls_engine.dart';
```

3. The login API adjustment is more standardized and no parameters need to be specified.

4. Optimization of offline push parameter construction.