

Tencent Real-Time Communication

Video Calling (Including UI)

Product Documentation



Copyright Notice

©2013-2025 Tencent Cloud. All rights reserved.

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

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by the Tencent corporate group, including its parent, subsidiaries and affiliated companies, as the case may be. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

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

Contents

- Video Calling (Including UI)
 - Overview (TUICallKit)
 - Activate the Service (TUICallKit)
 - Run Demo (TUICallKit)
 - Android
 - iOS
 - Web
 - Flutter
 - React Native
 - Integration (TUICallKit)
 - Android
 - iOS
 - Web&H5 (React)
 - Web&H5 (Vue3)
 - Flutter
 - uniapp (Android&iOS)
 - React Native
 - Calls integration to Chat (TUICallKit)
 - Web&H5 (React)
 - Web&H5 (Vue3)
 - UI Customization (TUICallKit)
 - Android
 - iOS
 - Web
 - Flutter
 - Offline Call Push (TUICallKit)
 - iOS
 - VoIP
 - Notification
 - Android
 - Flutter
 - Notification
 - VoIP (Optional)
 - AI Noise Reduction (TUICallKit)
 - Virtual Background (TUICallKit)

iOS

Android

Web

Flutter

Conversational Chat (TUICallKit)

Android

iOS

Flutter

Vue

React

On-Cloud Recording (TUICallKit)

Additional Features (TUICallKit)

Configuring Nicknames and Avatars (All Platform)

Configure Resolution and Fill Mode (Web)

USB Camera Capture

Volume Increment

Multi-Person Call

Android&iOS&Flutter

Web&H5

uni-app (Anroid&iOS)

Floating Window

Android&iOS&Flutter

Web&H5

uni-app (Anroid&iOS)

Beauty Effects

Flutter

Custom Ringtone

Android

iOS

Web&H5

uni-app (Anroid&iOS)

Flutter

Monitoring Call Status

Android&iOS&Flutter

Web&H5

uni-app (Android & iOS)

Language Settings

Web&H5

iOS

Android

Flutter

uni-app (Android&iOS)

Solution (TUICallKit)

WhatsApp Clone

Server APIs (TUICallKit)

REST API

REST API Introduction

REST API List

Call Records

Get Records by CallId

Retrieve Records by Conditions

Get Real-Time Call Status

Third-Party Callback

Third-Party Callback Overview

Callback Command List

Callback Configuration

Query Callback Configuration

Setting Callback Configuration

Deleting Callback Configuration

Call Status Callback

Callback after Call Initiation

Callback after Call Ends

Callback after Member Changed

Legacy Documentation

Call Status Callback

Call Status Callback

Call Event Callback

Callback Configuration

API List for Callback Configuration

Establishing Callback Configuration

Retrieving Callback Configuration

Update Callback Configuration

Remove Callback Configuration

REST API

Call Records

Introduction to REST API

Retrieve records via callId

Retrieve Records Based on Conditions

End Calls

Client APIs (TUICallKit)

Android

API Overview

TUICallKit

TUICallEngine

TUICallObserver

Type Definition

iOS

API Overview

TUICallKit

TUICallEngine

TUICallObserver

Type Definition

Web

API Overview

TUICallKit

TUICallEngine

TUICallEvent

Flutter

API Overview

TUICallKit

TUICallEngine

TUICallObserver

Type Definition

uniapp (Android&iOS)

API Overview

TUICallKit

TUICallEngine

TUICallEvent

React Native

API Overview

TUICallKit

TUICallEngine

TUICallEvent

Definitions of Key Types

[ErrorCode\(TUICallKit\)](#)

[Release Notes \(TUICallKit\)](#)

[Web](#)

[Android&iOS](#)

[Flutter](#)

[FAQs\(TUICallKit\)](#)

[iOS](#)

[Android](#)

[Web](#)

[All Platform](#)

[Flutter](#)

Video Calling (Including UI)

Overview (TUICallKit)

Last updated : 2025-06-23 11:26:47

Component Overview

TUICallKit is an audio and video call UI component launched by Tencent RTC, supporting real-time communication with up to nine participants simultaneously. 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 Platforms

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

Features

Basic Features	Advanced Features	Feature Advantages
1v1 voice/video call Group call (up to 9 people) Invite/join ongoing calls Customize incoming call ringtone Customize nickname, avatar Floating window Mute/Unmute the ringtone for incoming calls	Offline push Virtual background On-cloud recording AI noise suppression Global Interconnectivity Adapt to poor network conditions Call records	Comprehensive UI interaction Cross-platform interconnection Multi-device login

Applicable Scenarios

Online Social Interaction	Online Consultation

Online Education	Online Sales

Try Our Product Online

Platform	Web	Android	iOS	Flutter
Demo				/
Demo Integration	Web Demo	Android Demo	iOS Demo	Flutter Demo

Suggestions and Feedback

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

Activate the Service (TUICallKit)

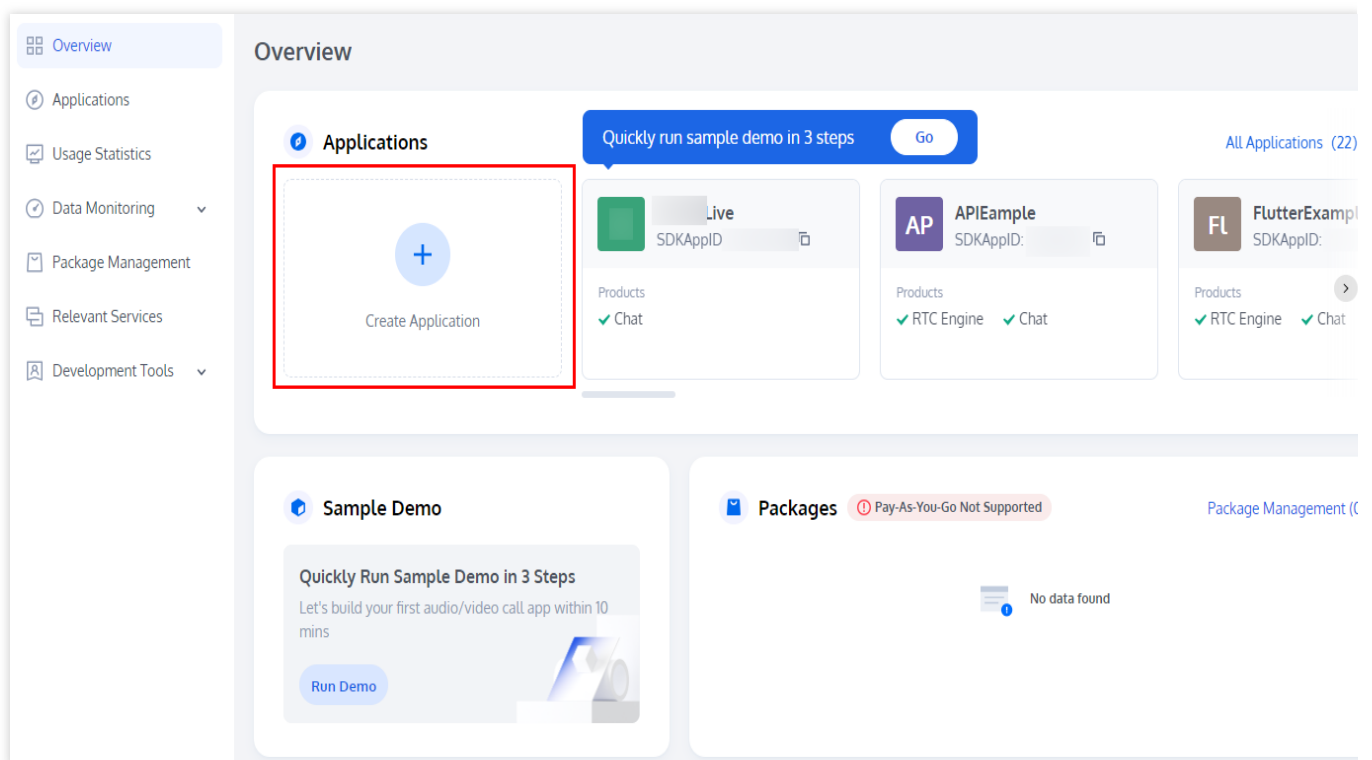
Last updated : 2025-02-14 17:45:16

Activate Trial Service

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 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. In the popup, enter an **application name**, select **Call**, and choose the appropriate **deployment region**. Then click **Create**. This will create a TRTC application bound to the trial edition of TRTC Call.

Create application

Application name

Enter application name

The application name can contain only digits, letters, and underscores.

Select product


☒ Call UIKit

☐ Conference UIKit

☐ Live UIKit

☐ Chat UIKit

☐ RTC Engine



Version

Free Trial 7 Days Free for 10,000 minutes every month [Version Details](#)

Deployment Region

Singapore (Globally communicable)

All our services are globally communicable, regardless of region selection. [Regions only specify Chat service deployment and data storage.](#)

☒ I have read and agree to [TRTC Service Level Agreement](#), [TRTC Billing Overview](#) and [Call Monthly Package](#).

Create

3. 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.

All Applications

Application Overview

Advanced Features

Call

Conference

Live

Call

Basic Information

Edition

Call: Trial

Service status

Normal

SDKAppID

Expiration time

2024-07-25

Auto-renewable

--

SDKSecretKey

Buy package

Quick start

Get you through it with documentation, step by step

Start Building

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**.

The screenshot shows the 'Tencent RTC | Order' page. At the top right are 'Talk to us' and 'Console' buttons. The main heading is 'Call Monthly Packages'. Below this is a section for 'Application (SDKAppID)' with a dropdown menu (highlighted with a red box) and a 'Create Application' link. A note states: 'Please select the correct SDKAppID, as it cannot be modified after purchase.' Below this are 'Package editions' with a 'Detail' link. Two packages are shown: '1-to-1 Call' and 'Group Call' (marked 'Most Popular'). The '1-to-1 Call' package includes 100,000 FREE mins, 1-to-1 audio/video call, Complete UI, and Floating window. The 'Group Call' package includes 300,000 FREE mins, All 1-to-1 Call features, Group audio/video call, and Virtual Background. Below the packages is an 'Automatic renewal' toggle (checked) with a note: 'Automatically renew monthly after expiration. You can cancel at any time, please feel free to check it.' At the bottom, there is a checkbox (checked) for 'I have read and agree to TRTC Service Level Agreement, TRTC Billing Overview and Call Monthly Package' (highlighted with a red box), a price tag of '\$99.00', and a 'Subscribe now' button (highlighted with a red box).

2. After the purchase is complete, go to [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.

The screenshot shows the 'Order' page for 'Call Monthly Packages' on the Tencent RTC console. At the top, there's a header with the Tencent RTC logo and 'Order' text, along with 'Talk to us' and 'Console' buttons. The main title is 'Call Monthly Packages'. Below this, there's a section for 'Application (SDKAppID)' with a dropdown menu and a 'Create Application' link. A note below states: 'Please select the correct SDKAppID, as it cannot be modified after purchase.' There are two package options: '1-to-1 Call' and 'Group Call'. The 'Group Call' package is marked as 'Most Popular'. Both packages list features like 'FREE mins included', 'audio/video call', 'Complete UI', and 'Floating window'. Below the packages, there's a toggle for 'Automatic renewal' which is currently turned on. A note below the toggle says: 'Automatically renew monthly after expiration. You can cancel at any time, please feel free to check it.' At the bottom, there's a checkbox for 'I have read and agree to TRTC Service Level Agreement, TRTC Billing Overview and Call Monthly Package' which is checked. To the right of the checkbox is a price tag '\$99.00' and a 'Subscribe now' button.

Tencent RTC | Order Talk to us Console

Call Monthly Packages

Application (SDKAppID)
[Dropdown menu] [Create Application](#)

Please select the correct SDKAppID, as it cannot be modified after purchase.

Package editions [Detail](#)

1-to-1 Call

- 100,000 FREE mins included
- 1-to-1 audio/video call
- Complete UI
- Floating window

\$99.00

Group Call

- 300,000 FREE mins included
- All 1-to-1 Call features
- Group audio/video call
- Virtual Background

\$99.00

☒ Automatic renewal

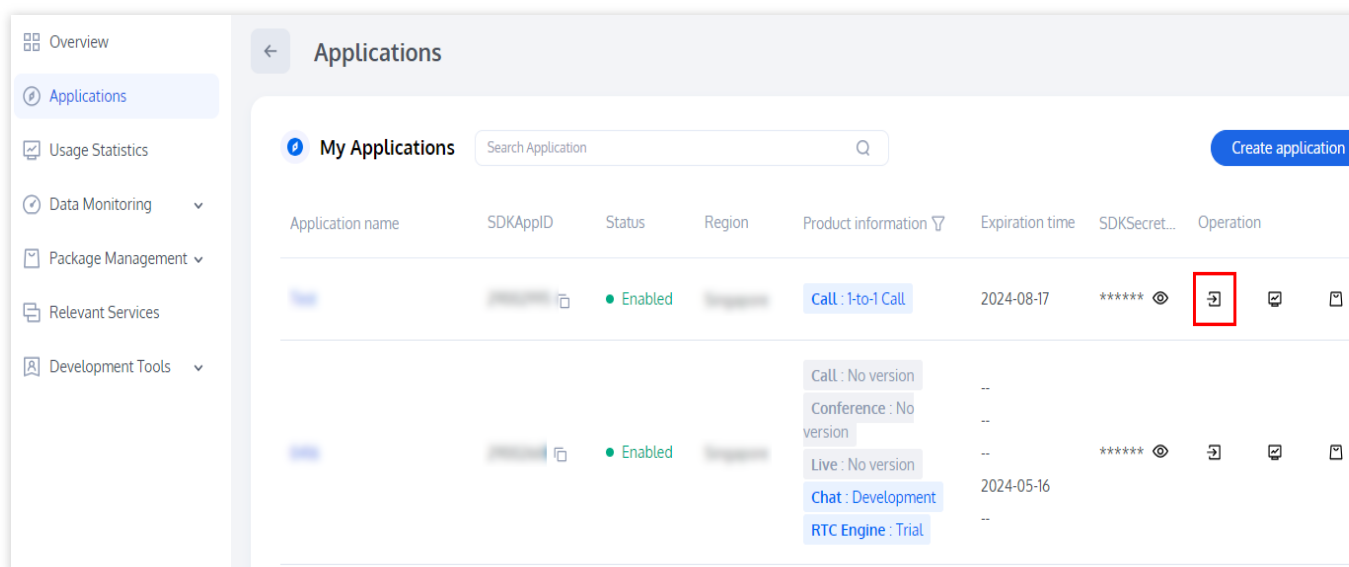
Automatically renew monthly after expiration. You can cancel at any time, please feel free to check it.

☒ I have read and agree to [TRTC Service Level Agreement](#), [TRTC Billing Overview](#) and [Call Monthly Package](#) \$99.00 Subscribe now

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**.

All Applications

Application Overview

Advanced Features

Call

Conference

Live

RTC Engine

Chat

In-game Voice Chat

Application Overview

Ready to start building?

Integration Docs

Run Sample Code

Try Web Demo

Basic Information

Application name	Test	SDKSecretKey	*****
SDKAppID		Creation time	2024-07-18 15:18:39
Description	--	Region	Singapore
Status	Enabled	Service Availability Zone	Global

Advanced Features

On-cloud recording	Disabled
Relay to CDN	Disabled
Callbacks	Disabled
Advanced permission control	Disabled

Products

Call

Edition

Call: 1-to-1 Call

Expiration time

2024-08-17

Auto-renewable

Not enabled

Enable

Renewal

Integrate now

Activate more products

Conference

Activate

Live

Activate

Chat

Activate

Are you sure you want to enable auto-renewal?



After auto-renewal is enabled, the package will be automatically renewed on 2024-08-17.

Enable


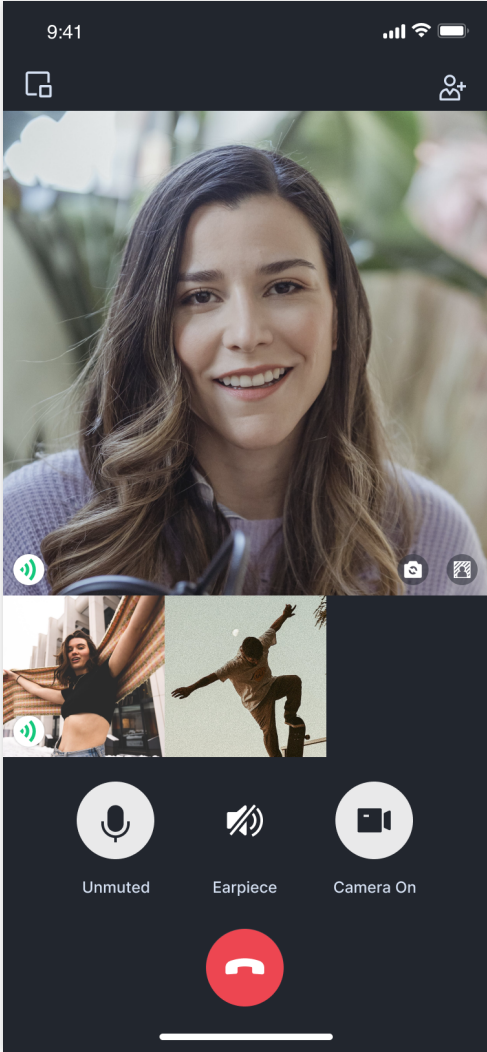
Cancel

Run Demo（TUICallKit）

Android

Last updated：2024-08-15 17:04:03

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

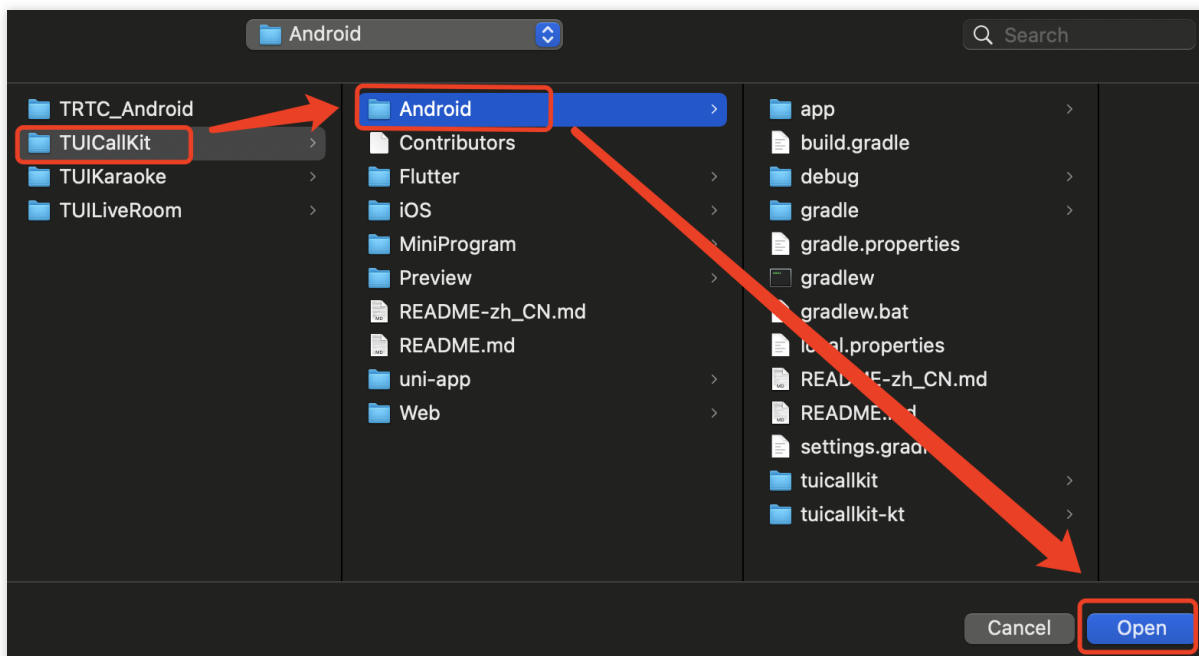
Two devices with Android 5.0 or higher

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.

Application Overview - Test

Basic Information

Application name	Test	SDKSecretKey	*****
SDKAppID		Creation time	2024-04-17 16:21:30
Description	--	Region	Singapore
Status	Enabled		

Advanced Features [More Features](#)

On-cloud recording	Disabled
Relay to CDN	Disabled
Callbacks	Disabled
Advanced permission control	Disabled

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

Call

Edition: [Call: Trial >](#)

Expiration time: 2024-04-24

Auto-renewable: --

[Buy package](#) [Integrate](#)

Chat

Edition: [Chat: Development](#)

Expiration time: 2024-05-17

Auto-renewable: --

[Buy package](#) [Integrate](#)

2. Open the

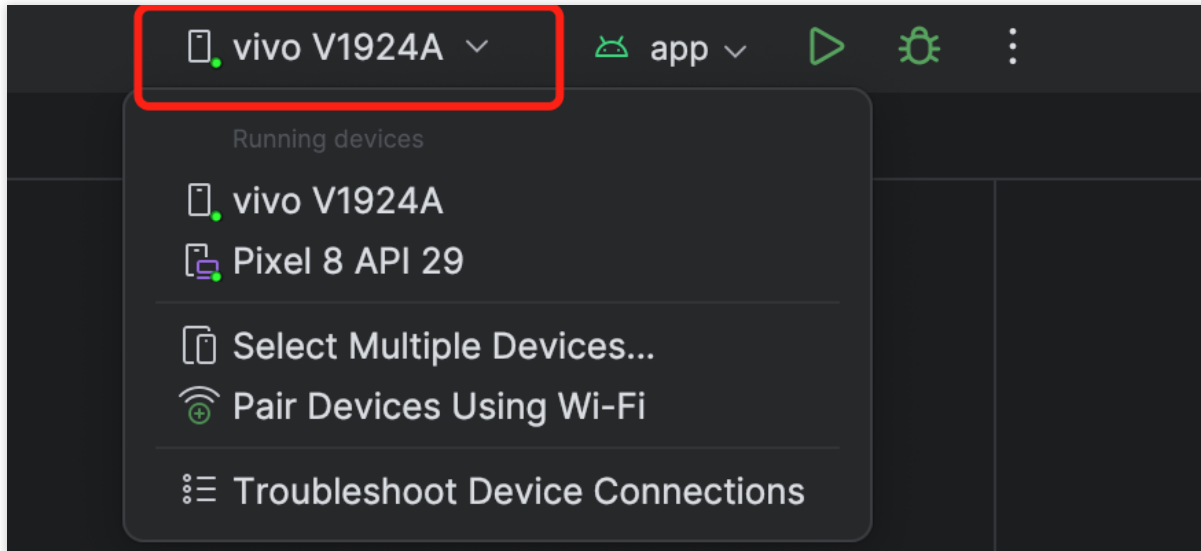
Android/debug/src/main/java/com/tencent/qcloud/tuikit/debug/GenerateTestUserSig.java

- a file, and enter the SDKAppID and SDKSecretKey obtained when [activating the service](#):

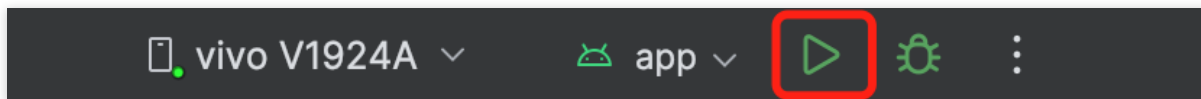
```
37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70  
/*  
 * Tencent Cloud SDKAppID. Set it to the SDKAppID of your account.  
 * <p>  
 * You can view your 'SDKAppID' after creating an application in the [Tencent Cloud IM console](https://console.tencentcloud.com/im).  
 * SDKAppID uniquely identifies a Tencent Cloud account.  
 */  
public static final int SDKAPPID = PLACEHOLDER;  
  
/**  
 * Signature validity period, which should not be set too short  
 * <p>  
 * Unit: Second  
 * Default value: 7 x 24 x 60 x 60 = 604800 (seven days)  
 */  
private static final int EXPIRETIME = 604800;  
  
/**  
 * Follow the steps below to obtain the key required for UserSig calculation.  
 * <p>  
 * Step 1. Log in to the [IM console](https://console.tencentcloud.com/im). If you don't have an account, please create one.  
 * Step 2. Click your application and find "Basic Information".  
 * Step 3. Click "Display Key" to view the key used for UserSig calculation.  
 * Copy and paste the key to the variable below.  
 * <p>  
 * Note: This method is for testing only. Before commercial launch,  
 * please migrate the UserSig calculation code and key to your backend server  
 * to prevent key disclosure and traffic stealing.  
 * Documentation: https://www.tencentcloud.com/document/product/1847/34385  
 */  
private static final String SECRETKEY = "PLACEHOLDER";
```


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:

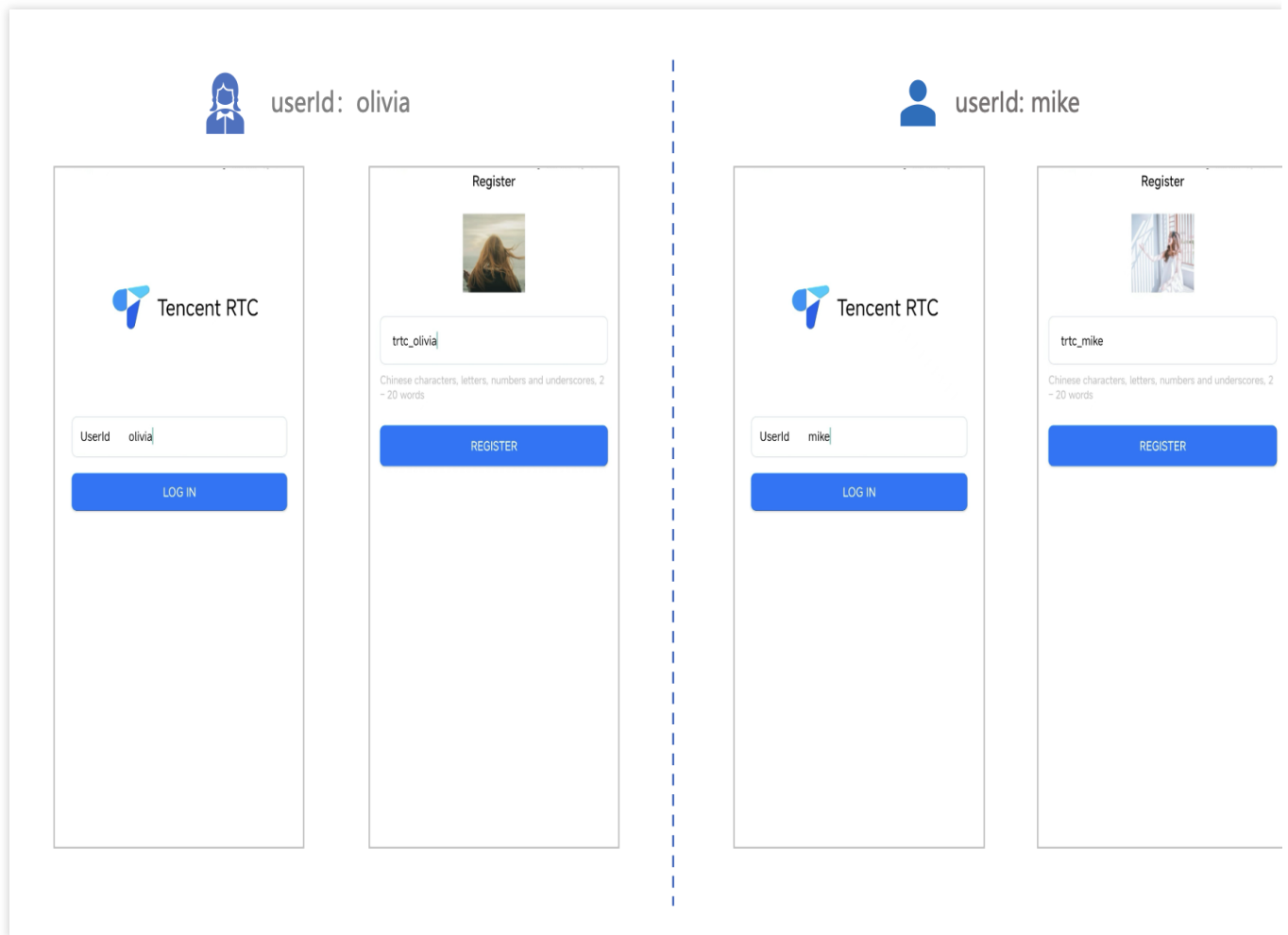
Make the first call

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.

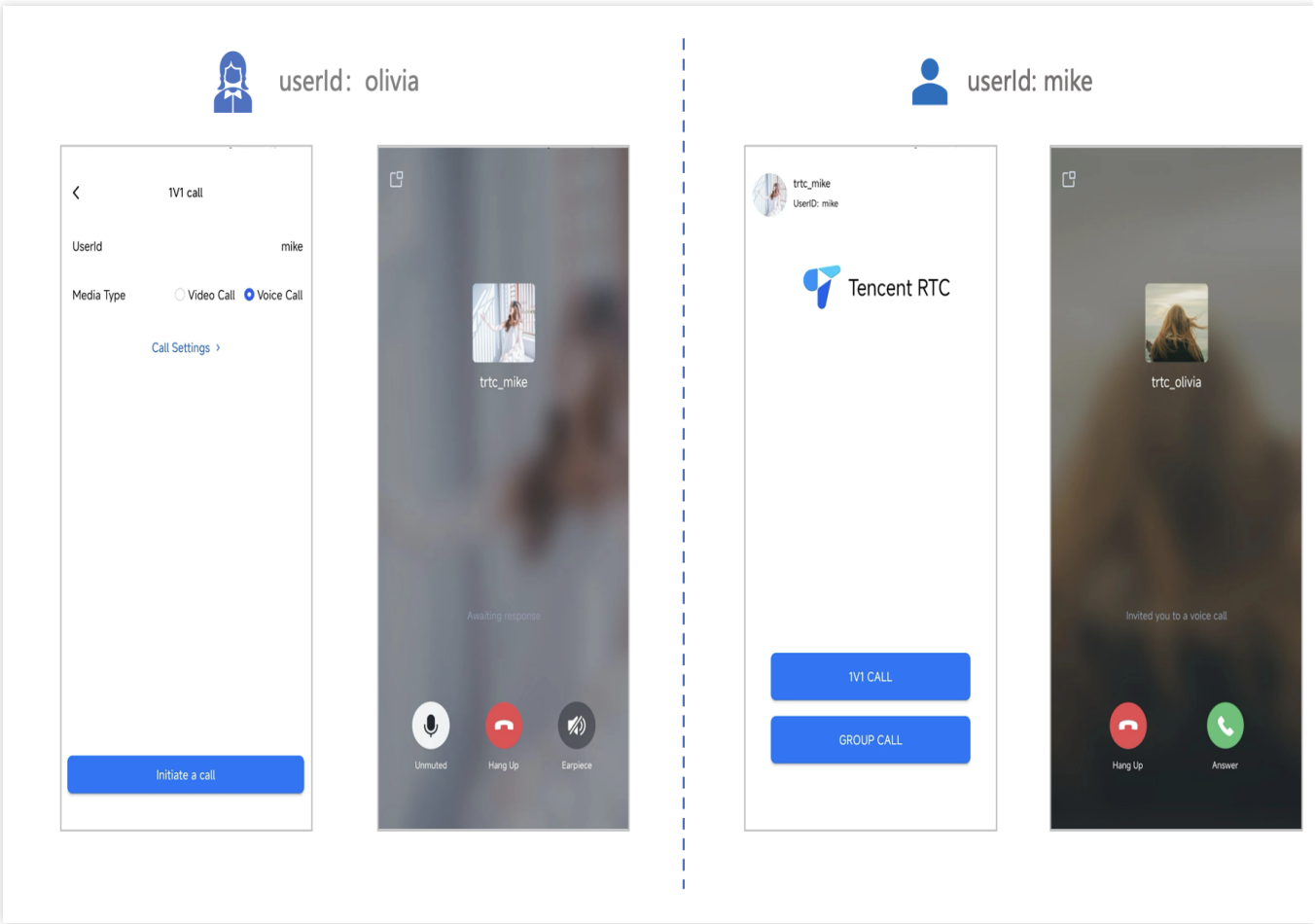
**Note:**

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.


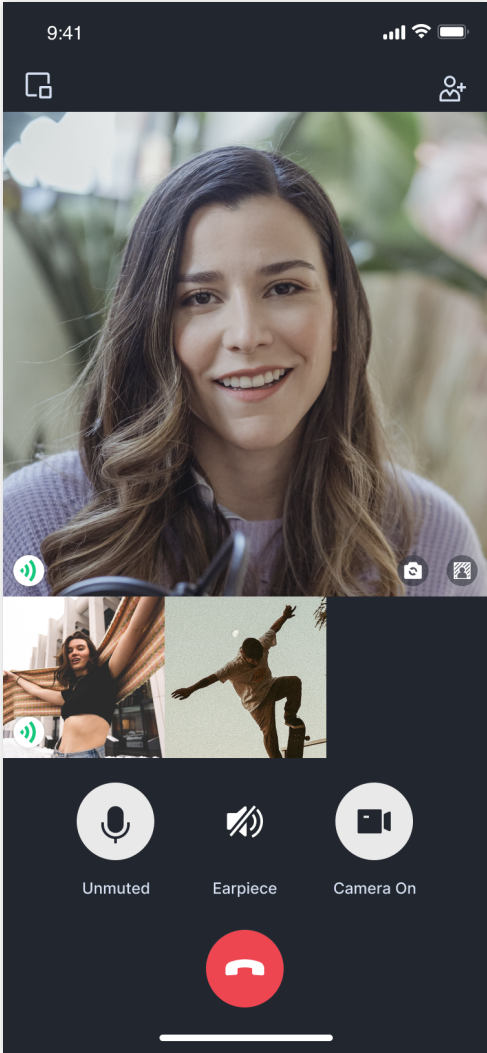
2.2 Click **Initiate Call**.



iOS

Last updated : 2024-08-15 17:03:36

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**.

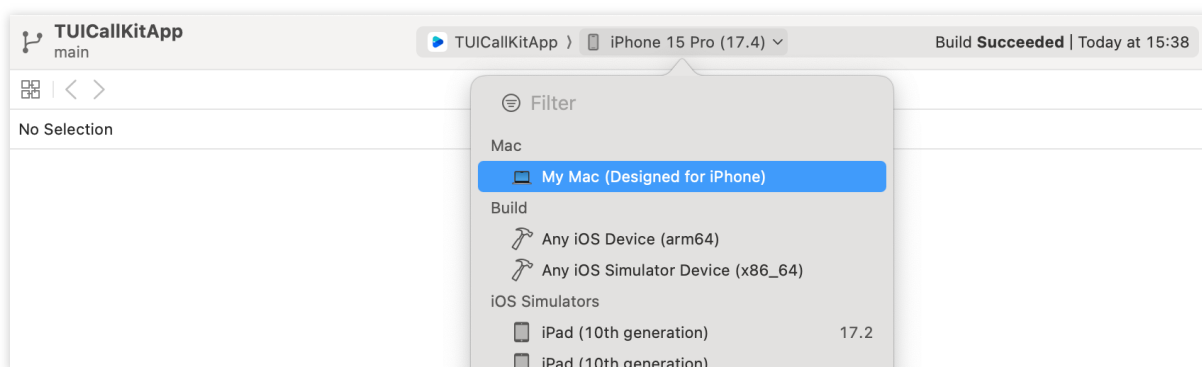
The screenshot displays the 'Application Overview' page in the Tencent Cloud console. The 'Basic Information' section shows the application name 'Test', the SDKAppID (highlighted with a red box), the SDKSecretKey (also highlighted with a red box), the creation time '2024-04-17 16:21:30', and the region 'Singapore'. The status is 'Enabled'. The 'Advanced Features' section shows various features like 'On-cloud recording', 'Relay to CDN', 'Callbacks', and 'Advanced permission control', all of which are 'Disabled'. Below these sections, there are two product cards: 'Call' and 'Chat'. The 'Call' card shows 'Edition: Call: Trial >', 'Expiration time: 2024-04-24', and 'Auto-renewable: --'. The 'Chat' card shows 'Edition: Chat: Development', 'Expiration time: 2024-05-17', and 'Auto-renewable: --'. Both cards have 'Buy package' and 'Integrate' buttons.

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

```
TUICallKitApp > Debug > GenerateTestUserSig > No Selection
10 import CommonCrypto
11 import zlib
12
13 /**
14  * Tencent Cloud SDKAppId, which needs to be replaced with the SDKAppId under your own account.
15  *
16  * Enter Tencent Cloud IM to create an application, and you can see the SDKAppId, which is the unique identifier used by Tencent Cloud to
17  */
18 let SDKAPPID: Int = 0
19
20 /**
21  * Signature expiration time, it is recommended not to set it too short
22  *
23  * Time unit: seconds
24  * Default time: 7 x 24 x 60 x 60 = 604800 = 7 days
25  */
26 let EXPIRETIME: Int = 604_800
27
28 /**
29  * Encryption key used for calculating the signature, the steps to obtain it are as follows:
30  *
31  * step1. Enter Tencent Cloud IM, if you do not have an application yet, create one,
32  * step2. Click "Application Configuration" to enter the basic configuration page, and further find the "Account System Integration" section.
33  * step3. Click the "View Key" button, you can see the encryption key used to calculate UserSig, please copy and paste it into the following
34  *
35  * Note: This solution is only applicable to debugging demos.
36  * Before going online officially, please migrate the UserSig calculation code and keys to your backend server to avoid traffic theft caused by
37  */
38 let SECRETKEY = ""
39
```

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.

The screenshot displays two side-by-side mobile app interfaces for Tencent RTC, separated by a vertical dashed line. The left interface is for user 'olivia' and the right is for user 'mike'. Each interface has a 'Login' screen and a 'Register' screen. The 'Login' screen features the Tencent RTC logo, a 'Userid' input field with the user's name, and a 'LOG IN' button. The 'Register' screen features a profile picture placeholder, a 'trtc_' prefix input field, a text constraint 'Chinese characters, letters, numbers and underscores, 2 ~ 20 words', and a 'REGISTER' button.

User	Screen	Input Field	Action
olivia	Login	Userid olivia	LOG IN
	Register	trtc_olivia	REGISTER
mike	Login	Userid mike	LOG IN
	Register	trtc_mike	REGISTER

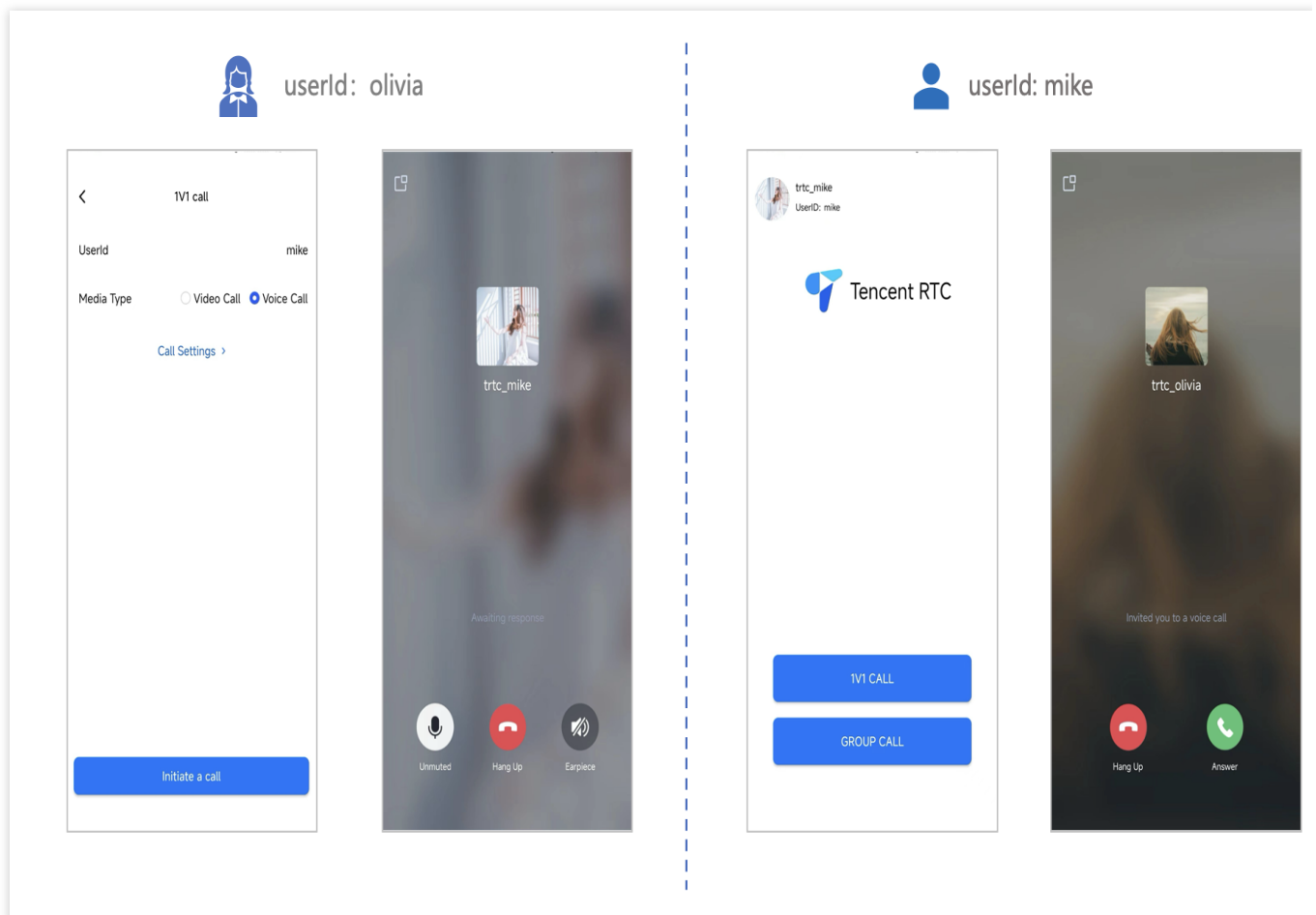
Note:

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.

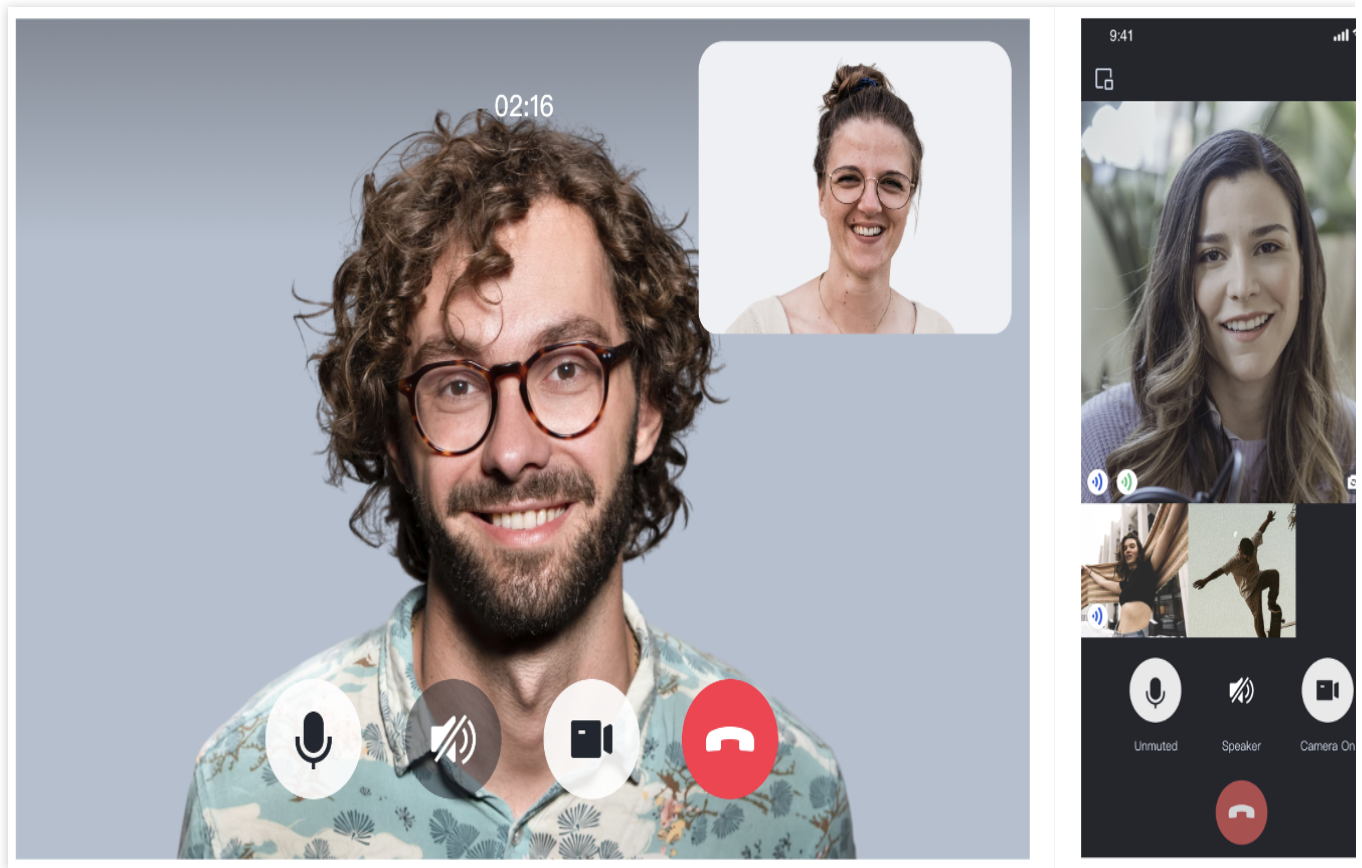
2.2 Click **Initiate Call**.



Web

Last updated : 2024-09-06 15:37:02

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`

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the following structure:

- OPEN EDITORS
 - GenerateTestUserSig-es.js Web/basic-r...
- TUICALLKIT
 - Web
 - basic-react
 - public
 - src
 - assets
 - components
 - context
 - debug
 - GenerateTestUserSig-es.js
 - lib-generate-test-usersig-es.min.js
 - hooks
 - interface
 - locales
 - pages
 - routes
 - style
 - utils
 - App.css
 - App.tsx
 - main.tsx
 - vite-env.d.ts
 - .eslintrc.cjs
 - .gitignore
 - index.html

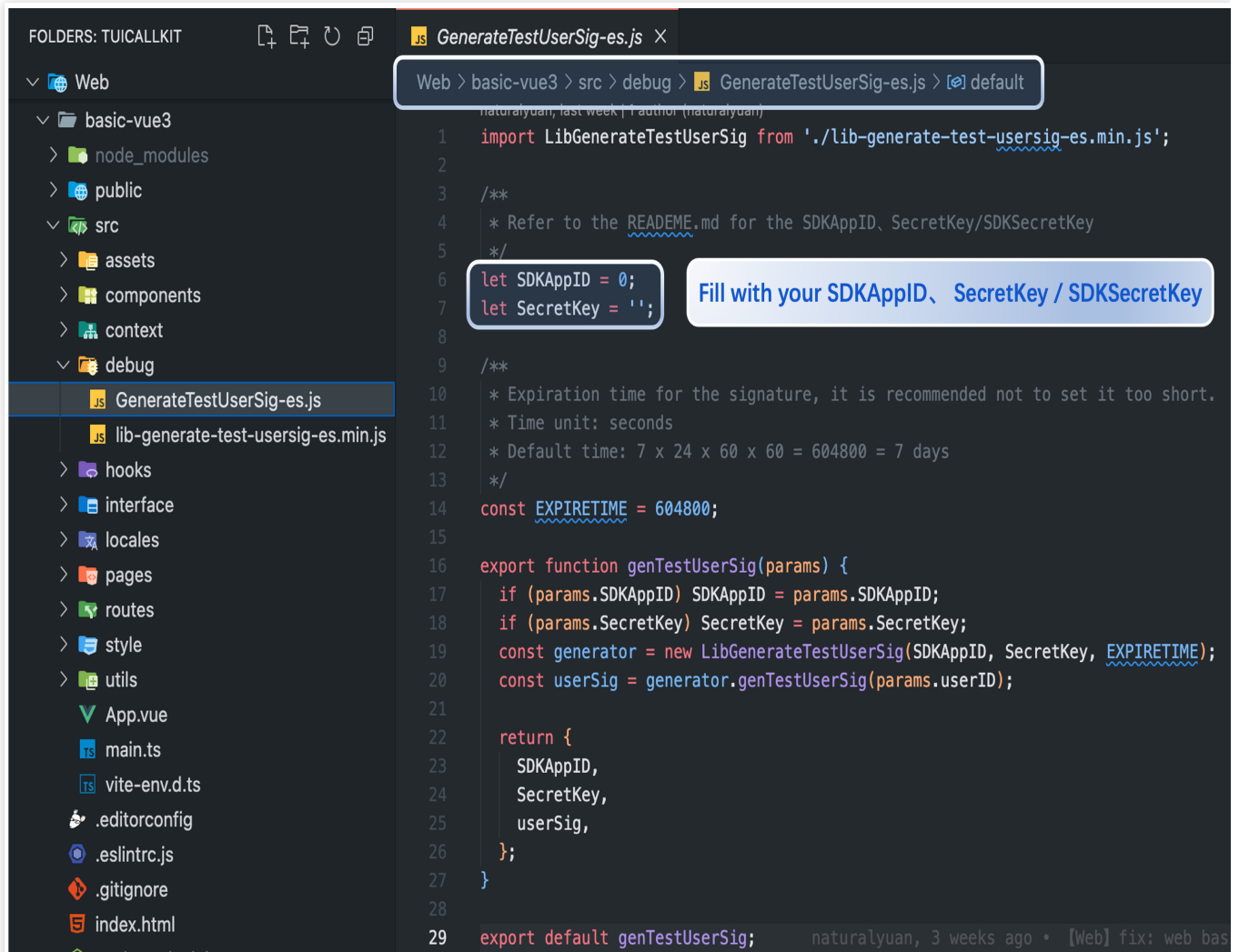
The code editor shows the content of `GenerateTestUserSig-es.js`:

```
1 import LibGenerateTestUserSig from './lib-generate-test-usersig-es.min.js';
2
3 /**
4  * Refer to the README.md for the SDKAppID, SecretKey
5  */
6 let SDKAppID = 0;
7 let SecretKey = '';
8
9 /**
10  * Expiration time for the signature, it is recommended not to set it too short.
11  * Time unit: seconds
12  * Default time: 7 x 24 x 60 x 60 = 604800 = 7 days
13  */
14 const EXPIRETIME = 604800;
15
16 export function genTestUserSig(params) {
17   if (params.SDKAppID) SDKAppID = params.SDKAppID;
18   if (params.SecretKey) SecretKey = params.SecretKey;
19   const generator = new LibGenerateTestUserSig(SDKAppID, SecretKey, EXPIRETIME);
20   const userSig = generator.genTestUserSig(params.userID);
21
22   return {
23     SDKAppID,
24     SecretKey,
25     userSig,
26   };
27 }
28
29 export default genTestUserSig;
30
```

Annotations in the image:

- A breadcrumb path at the top: `Web > basic-react > src > debug > GenerateTestUserSig-es.js > ...`
- A callout box around lines 6-7: `let SDKAppID = 0;` and `let SecretKey = '';`
- A callout box on the right: `Fill with your SDKAppID、 SecretKey / SDKSecretKey`

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
```

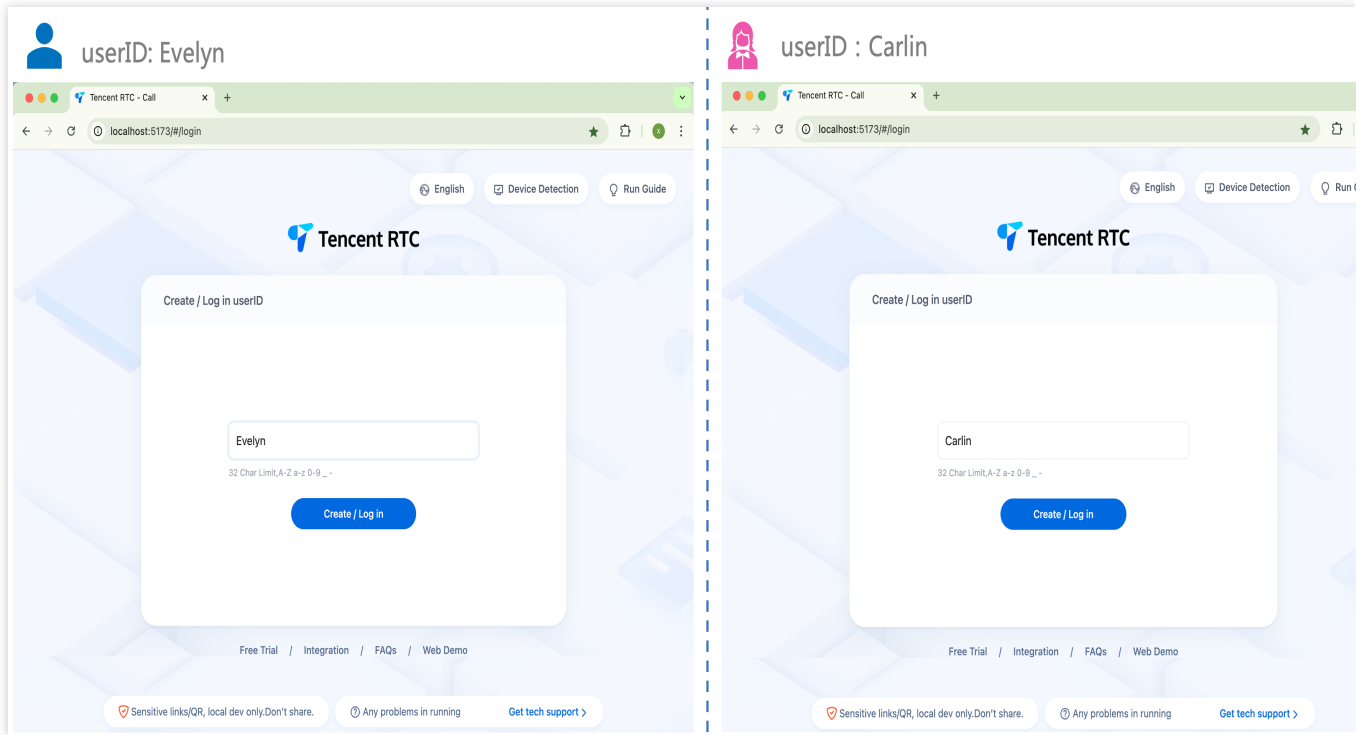
```
npm run dev
```

Warning:

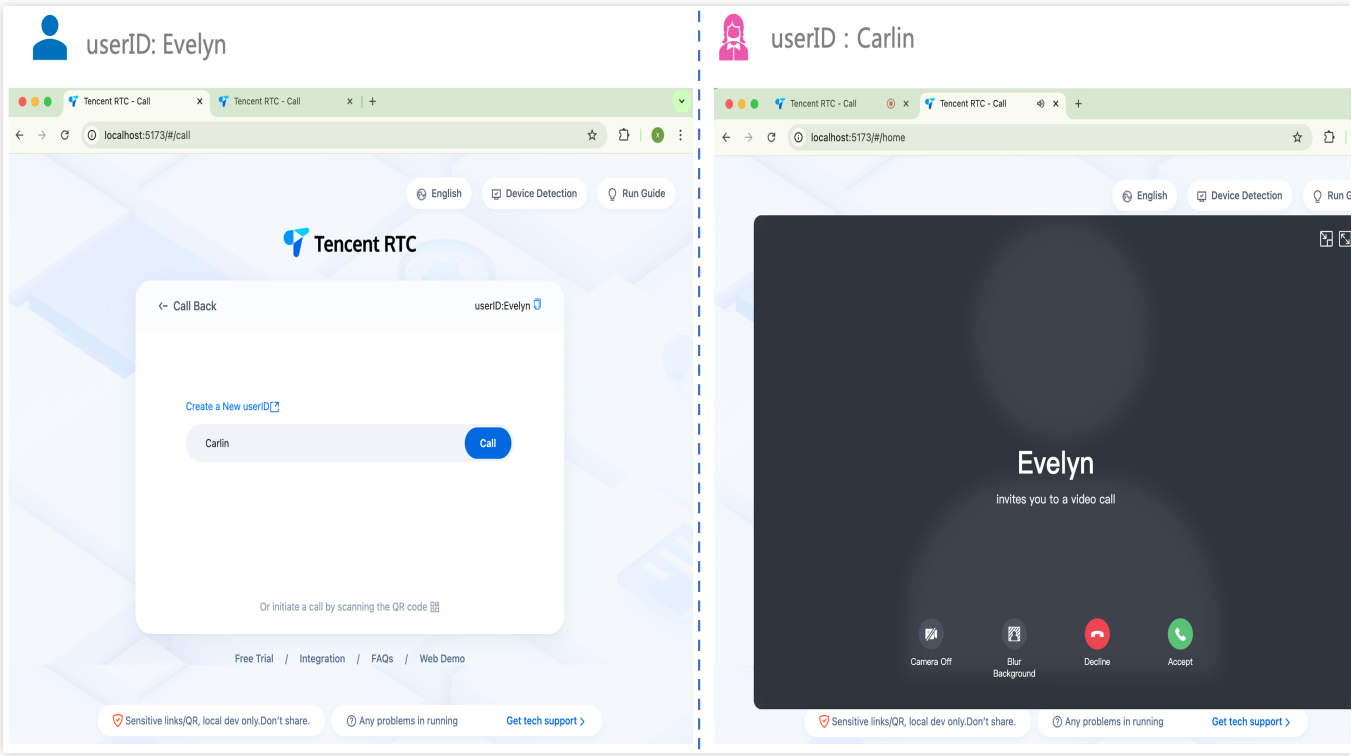
For local environments, access under the localhost protocol. For public network experience, access under the HTTPS protocol. For details, refer to [Network Access Protocol Instructions](#).

Step 4: Make your first call

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




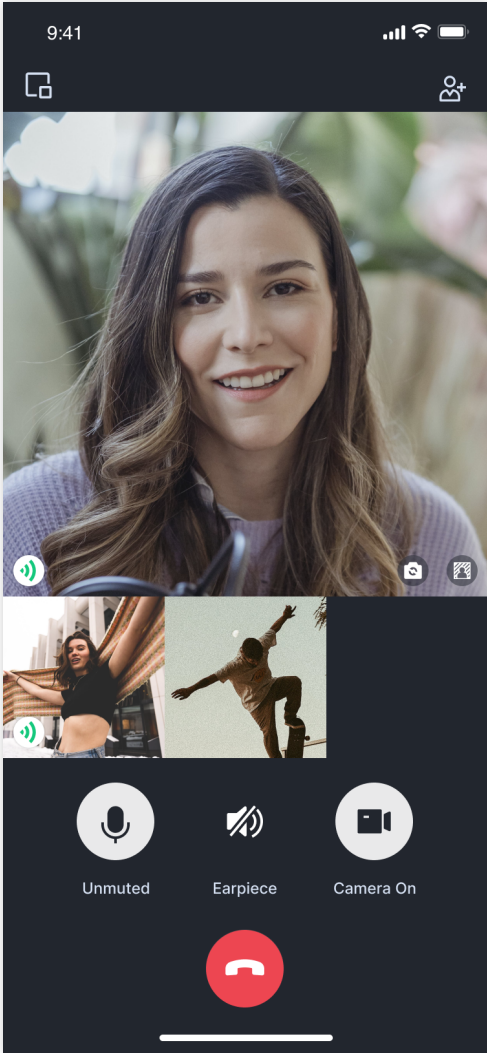
2. Input the callee's userID and click all to experience your first call.



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.

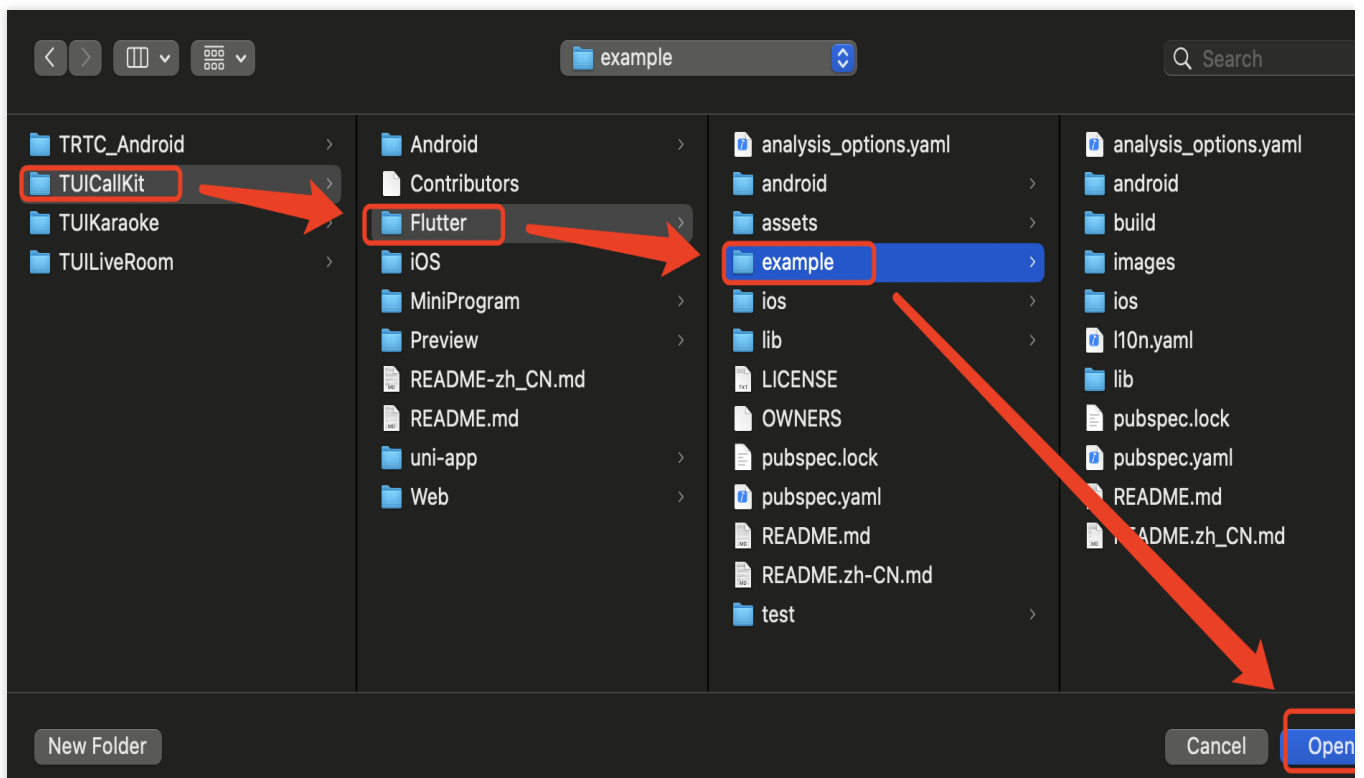
Please ensure your project is set up with a valid developer signature.

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**.

The screenshot shows the 'Application Overview' page for a Tencent Cloud application. The page is divided into several sections:

- Basic Information:** This section contains a table with the following data:

Field	Value
Application name	Test
SDKAppID	[Redacted]
SDKSecretKey	*****
Description	--
Status	Enabled
Creation time	2024-04-17 16:21:30
Region	Singapore
- Advanced Features:** This section lists several features and their status:

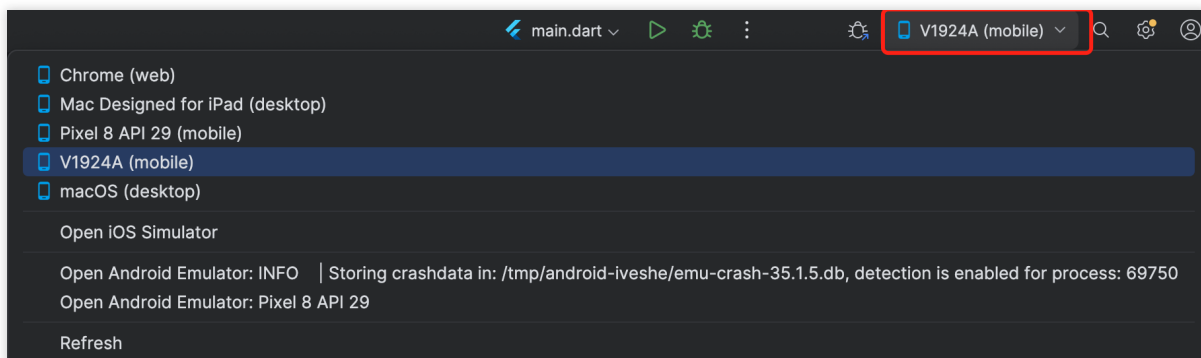
Feature	Status
On-cloud recording	Disabled
Relay to CDN	Disabled
Callbacks	Disabled
Advanced permission control	Disabled
- Products:** This section contains two product cards: 'Call' and 'Chat'. Each card has an 'Edition' (Call: Trial, Chat: Development), an 'Expiration time' (2024-04-24 for Call, 2024-05-17 for Chat), and an 'Auto-renewable' status (both are '--'). Below each card are 'Buy package' and 'Integrate' buttons.

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](#):

```
22
23 class GenerateTestUserSig {
24   /**
25    * Tencent Cloud 'SDKAppID'. Set it to the 'SDKAppID' of your account.
26    *
27    * You can view your 'SDKAppID' after creating an application in the [TRTC console](https://console.cloud.tencent.com/trtc).
28    * 'SDKAppID' uniquely identifies a Tencent Cloud account.
29    */
30   static int sdkAppId = 0;
31
32   /**
33    * Signature validity period, which should not be set too short
34    * <p>
35    * Unit: second
36    * Default value: 604800 (7 days)
37    */
38   static int expireTime = 604800;
39
40   /**
41    * Follow the steps below to obtain the key required for UserSig calculation.
42    *
43    * Step 1. Log in to the [TRTC console](https://console.cloud.tencent.com/trtc), and create an application if you don't have one.
44    * Step 2. Find your application, click "Application Info", and click the "Quick Start" tab.
45    * Step 3. Copy and paste the key to the code, as shown below.
46    *
47    * Note: this method is for testing only. Before commercial launch, please migrate the UserSig calculation code and key to your backend se
48    * Reference: https://cloud.tencent.com/document/product/647/17275#Server
49    */
50   static String secretKey = '';
51 }
```

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.



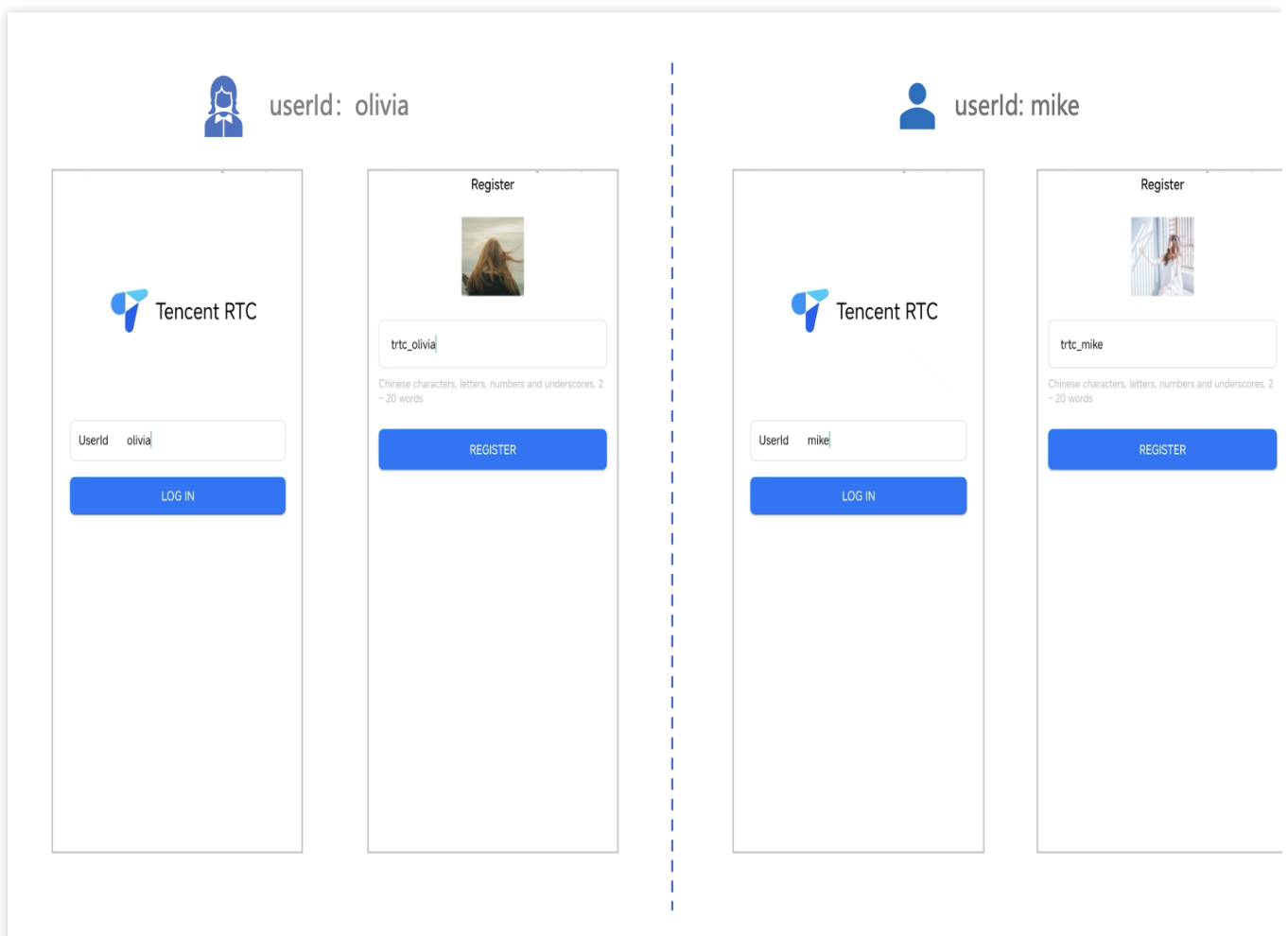
Make the first call

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.



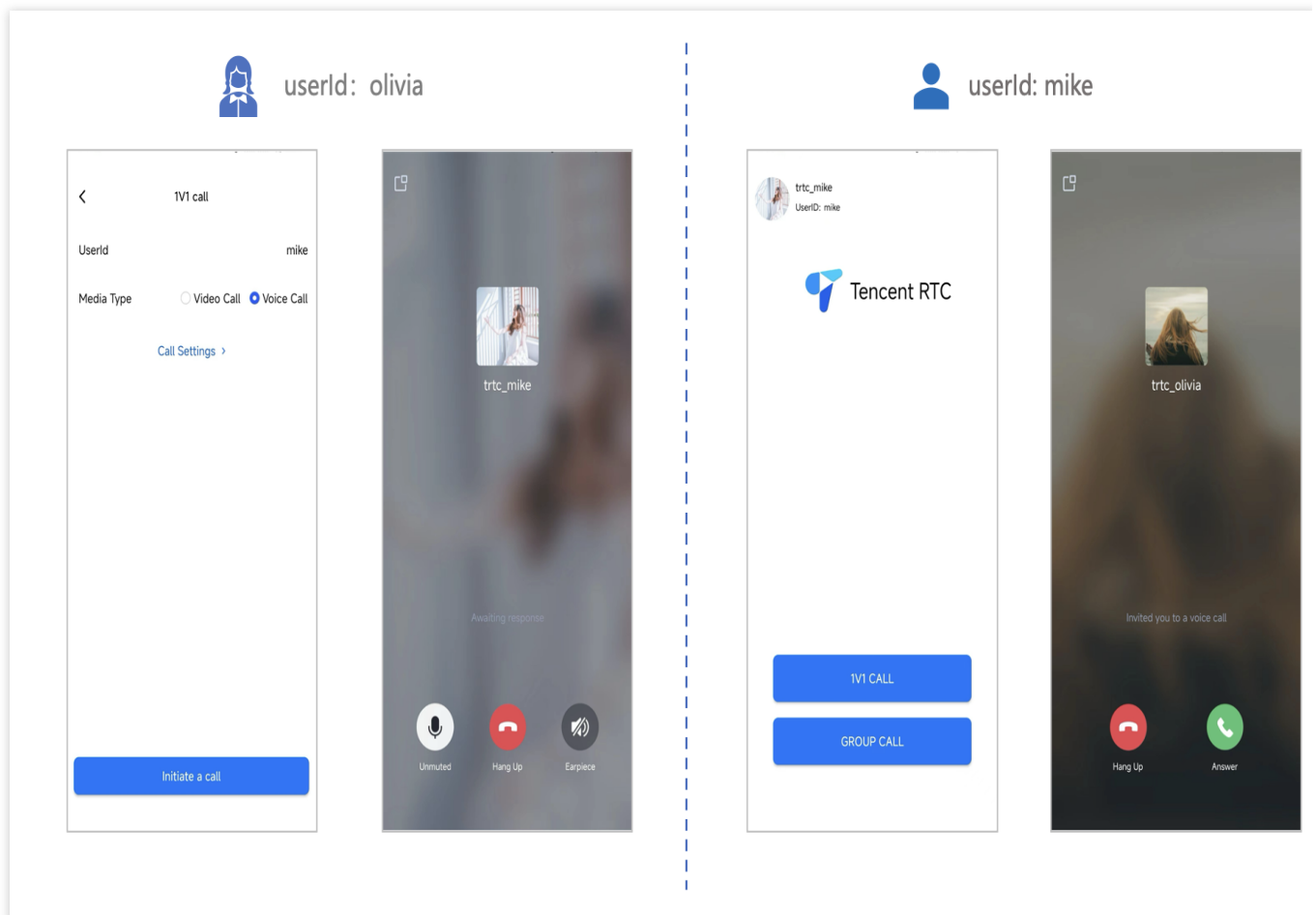
Note:

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.


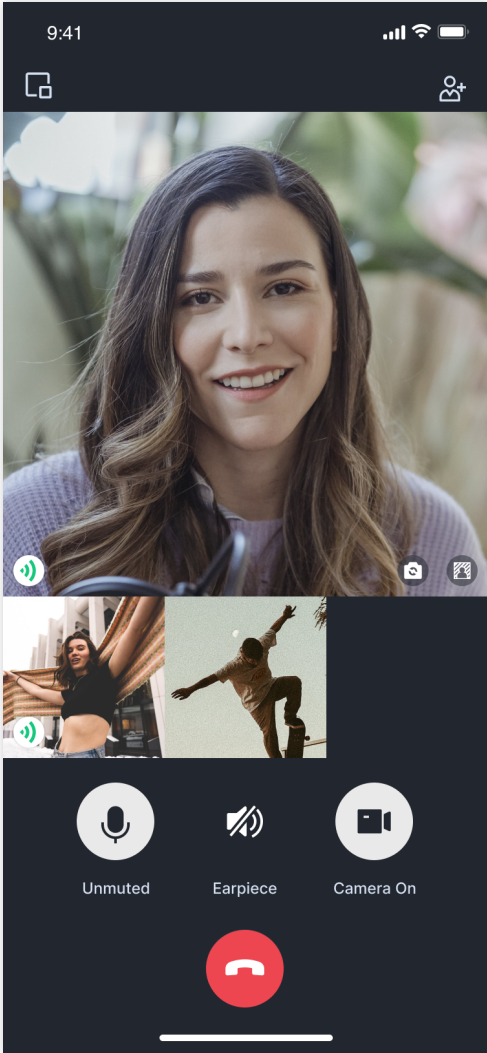
2.2 Click **Initiate Call**.



React Native

Last updated : 2024-12-17 14:36:39

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
	

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.

```
cd ./TUICallKit/ReactNative
yarn install
```

Step 2: Configure the demo

1. Go to the [Activate Service](#) page and get the SDKAppID and SDKSecretKey

The screenshot shows the 'Application Overview' page for a service named 'Test'. The page is divided into two main sections: 'Basic Information' and 'Advanced Features'.

Basic Information:

- Application name: Test
- SDKAppID: (highlighted with a red box)
- SDKSecretKey: ***** (highlighted with a red box)
- Description: --
- Status: Enabled (with a 'More' dropdown)
- Creation time: 2024-04-17 16:21:30
- Region: Singapore

Advanced Features:

- On-cloud recording: Disabled
- Relay to CDN: Disabled
- Callbacks: Disabled
- Advanced permission control: Disabled

Products:

Quickly run sample demo in 3 steps >

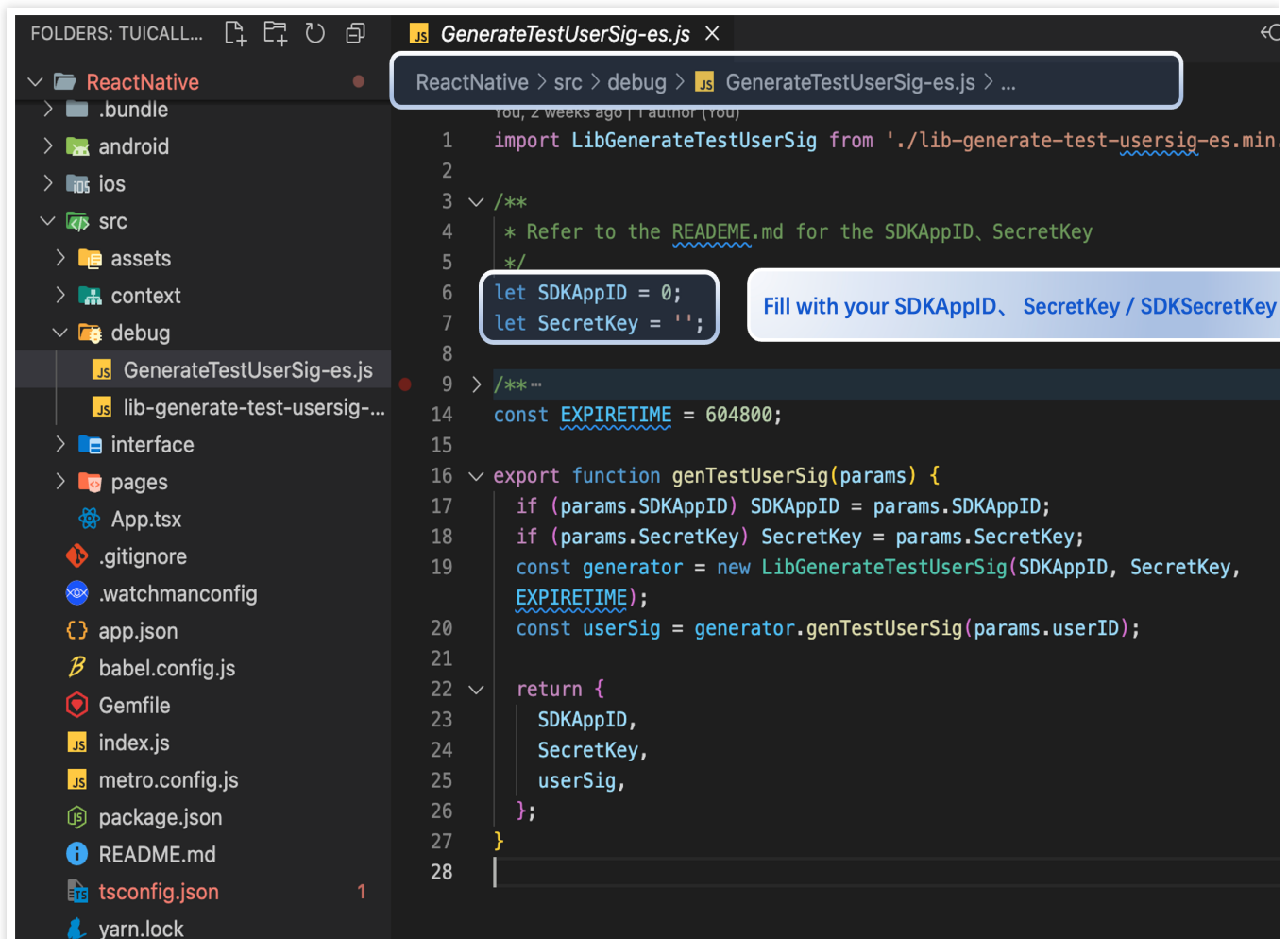
Call:

- Edition: Call : Trial >
- Expiration time: 2024-04-24
- Auto-renewable: --
- Buttons: Buy package, Integrate

Chat:

- Edition: Chat : Development
- Expiration time: 2024-05-17
- Auto-renewable: --
- Buttons: Buy package, Integrate

2. Fill them in the `TUICallKit/ReactNative/src/debug/GenerateTestUserSig-es.js` file.



Step 3: Run the demo


```
# TUICallKit/ReactNative
yarn start
```


Step 4: 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.

log in to userID (defined by you).

 userID: olivia

 Tencent RTC


Video Call

Voice Call

32 Char Limit,A-Z a-z 0-9 _ -


Create / Log in


userID: olivia


 Tencent RTC


Video Call

Voice Call

 1v1 Call

 Group Call

 userID : mike

 Tencent RTC


Video Call

Voice Call

32 Char Limit,A-Z a-z 0-9 _ -


Create / Log in


userID: mike

 Tencent RTC

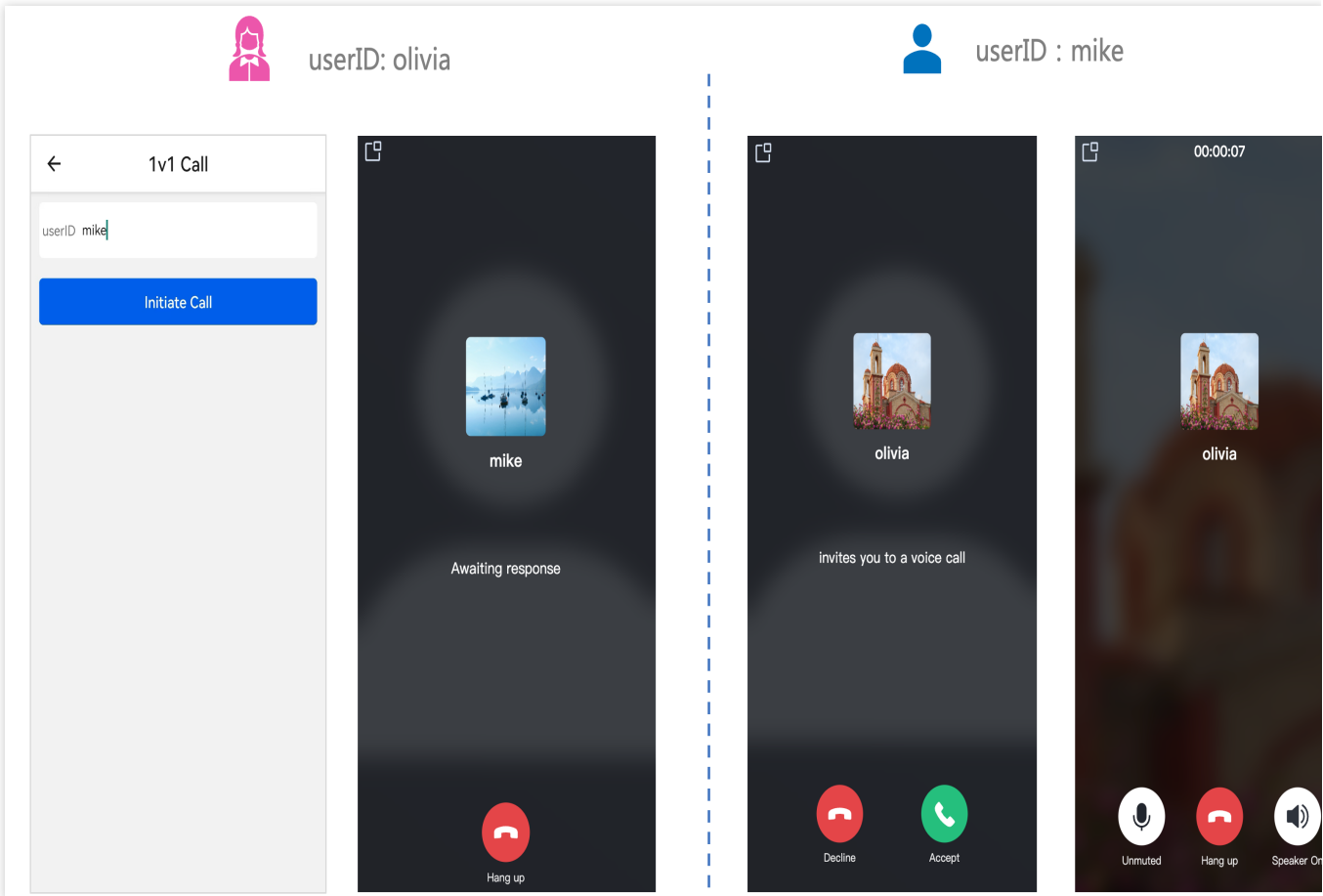
Video Call

Voice Call

 1v1 Call

 Group Call

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


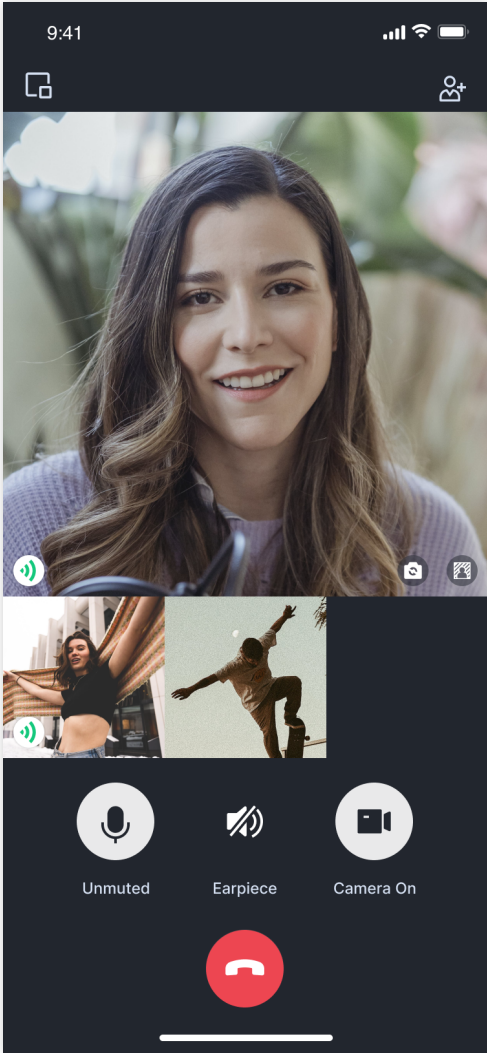


Integration (TUICallKit)

Android

Last updated : 2025-02-19 10:28:20

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

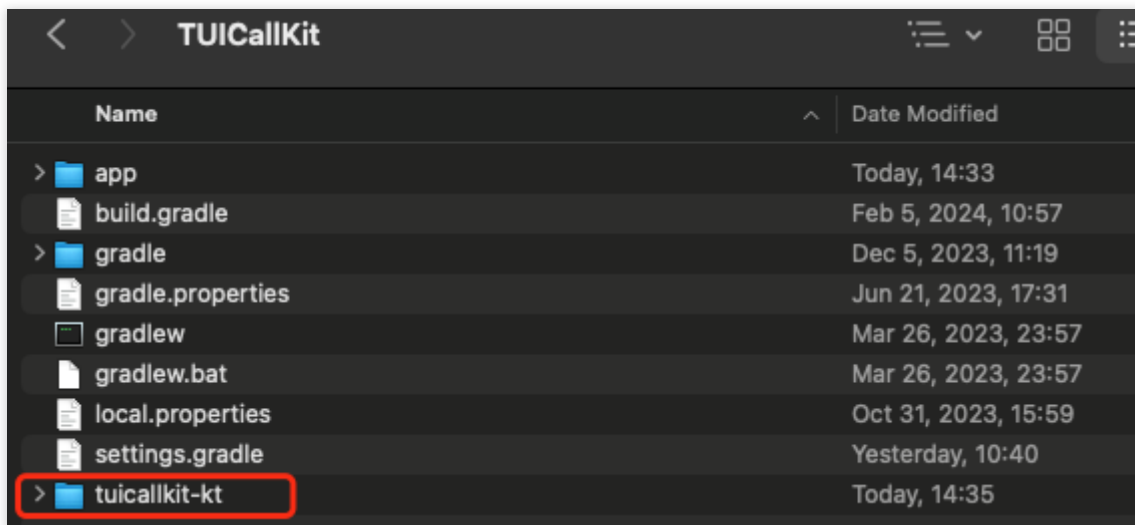
Mobile phone on Android 5.0 or later.

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.

`settings.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, secretKey)

        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 [calls](#) 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

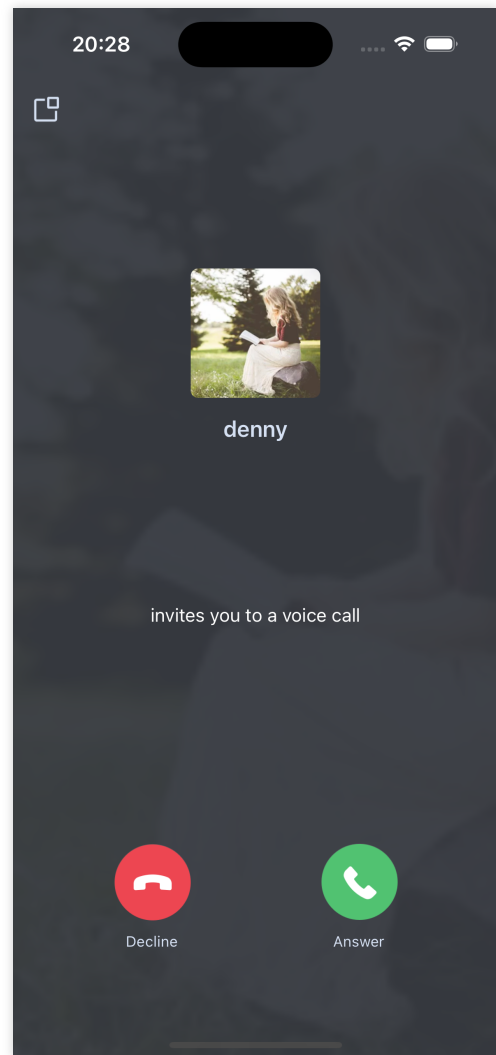
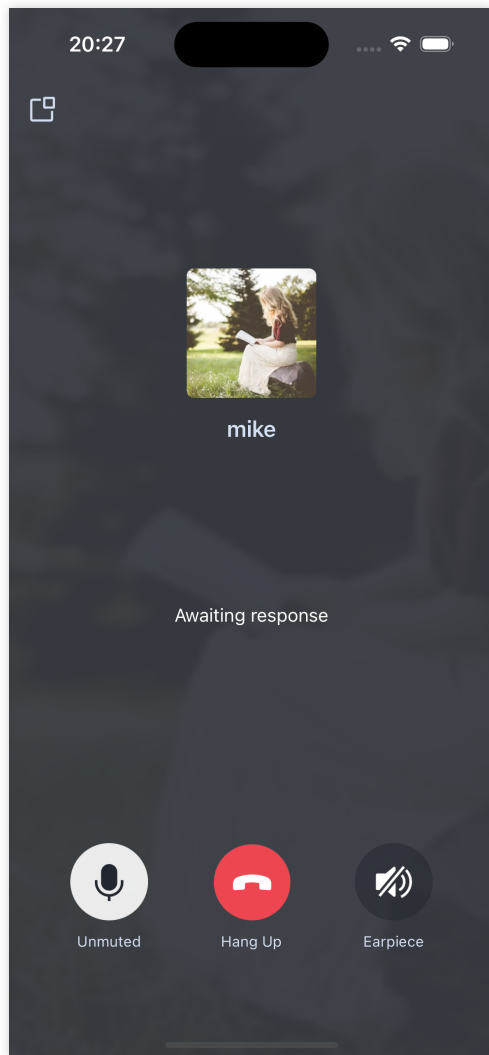
```
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

val list = mutableListOf<String>()
list.add("mike")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, null,

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

List<String> list = new ArrayList<>();
list.add("mike")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, null,
```

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](#).


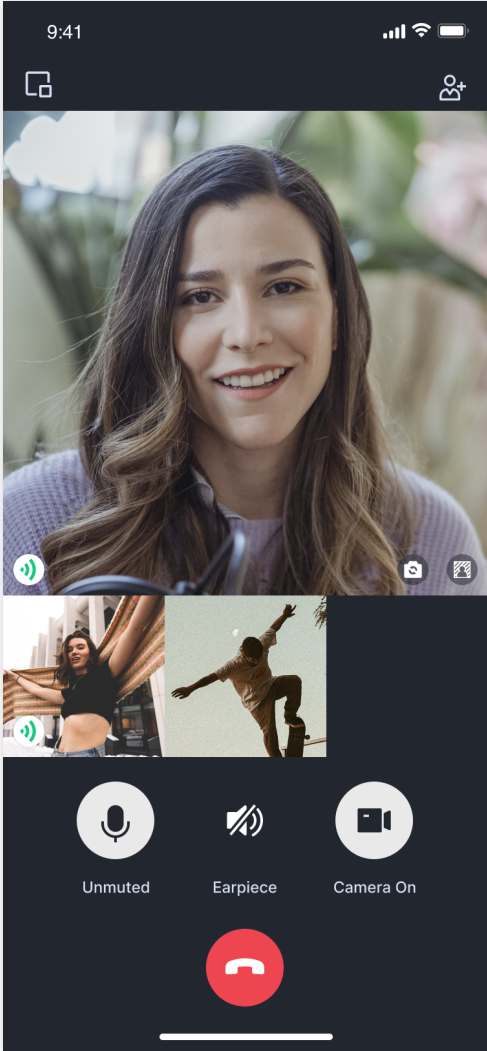
Suggestions and Feedback

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

iOS

Last updated : 2025-02-18 20:58:21

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

- 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/Professional'
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.

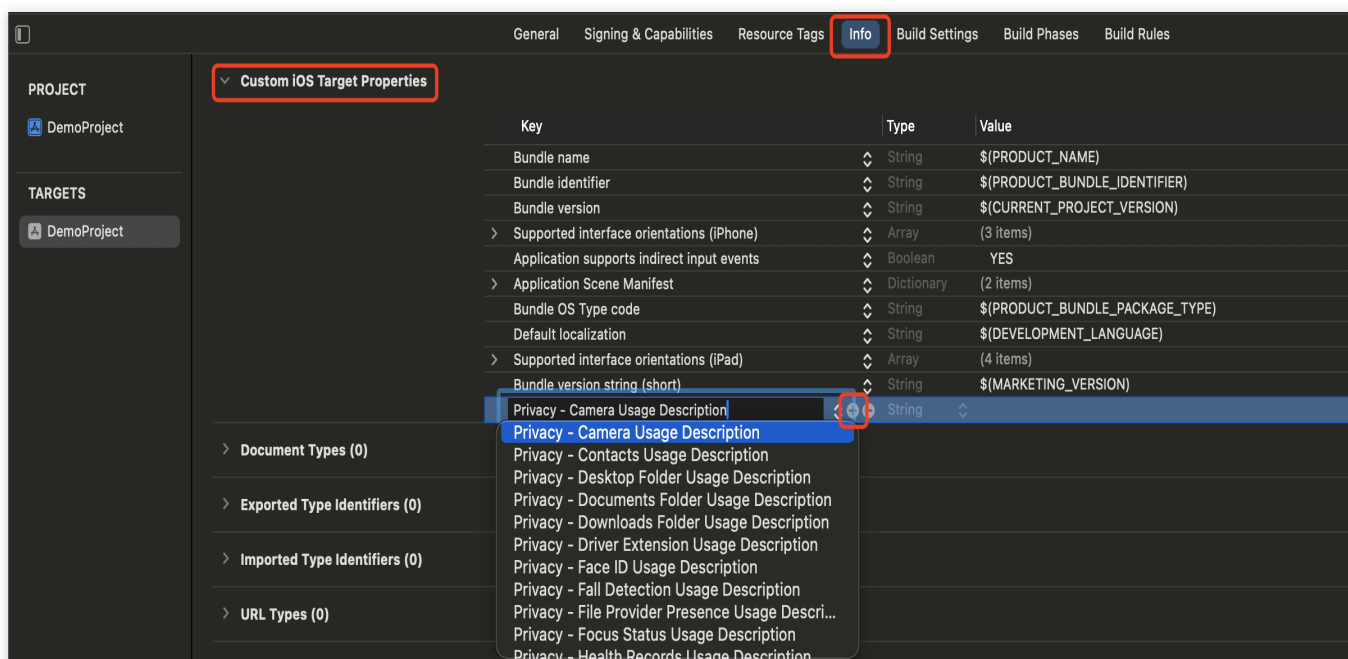
```
pod update
```

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.

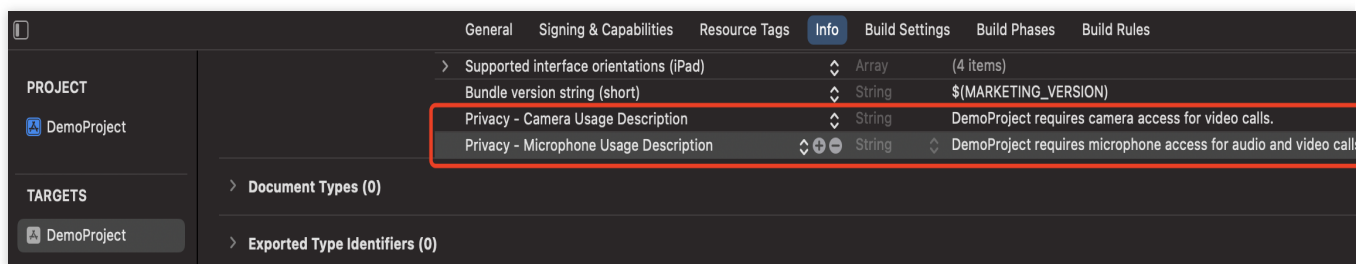
1. In **Xcode**, select **TARGETS > Info > Custom iOS Target Properties** from the menu.



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 launchOptions: [UIApplication.LaunchOptionsKey: Any]?) throws -> Bool {
    let userID = "denny"           // Please replace with your UserId
    let sdkAppID: Int32 = 0         // Please replace with the SDKAppID obtained from the console
    let secretKey = "****"         // Please replace with the SecretKey obtained from the console

    let userSig = GenerateTestUserSig.genTestUserSig(userID: userID, sdkAppID: sdkAppID, secretKey: secretKey)

    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:(NSDictionary *)launchOptions {
    NSString *userID = @"denny";    // Please replace with your UserId
    int sdkAppID = 0;               // Please replace with the SDKAppID obtained from the console
    NSString *secretKey = @"****";  // Please replace with the SecretKey obtained from the console

    NSString *userSig = [GenerateTestUserSig genTestUserSigWithUserID:userID sdkAppID:sdkAppID secretKey:secretKey];

    [TUILogin login:sdkAppID
                userID:userID
                userSig:userSig
                succ:^(int code, NSString * _Nullable msg) {
                    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 calls type and callee's `userId`, you can initiate an audio or video call.

Swift

Objective-C

```
import TUICallKit_Swift
import RTCRoomEngine

// Initiating a 1-to-1 audio call (assuming userId is mike)
TUICallKit.createInstance().calls(userIdList: ["mike"], callMediaType: .audio, para

} fail: { code, message in

}
```

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

// Initiating a 1-to-1 audio call (assuming userId is mike)
[[TUICallKit sharedInstance] calls:@[@"mike"] callMediaType:TUICallMediaTypeAudio p
} fail:^(int code, NSString * _Nullable errMsg) {
}];
```

Caller	Callee

Additional Features

[Setting Nickname, Avatar](#)

[Customize Interface](#)[Offline Push](#)[Multiplayer 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](#).

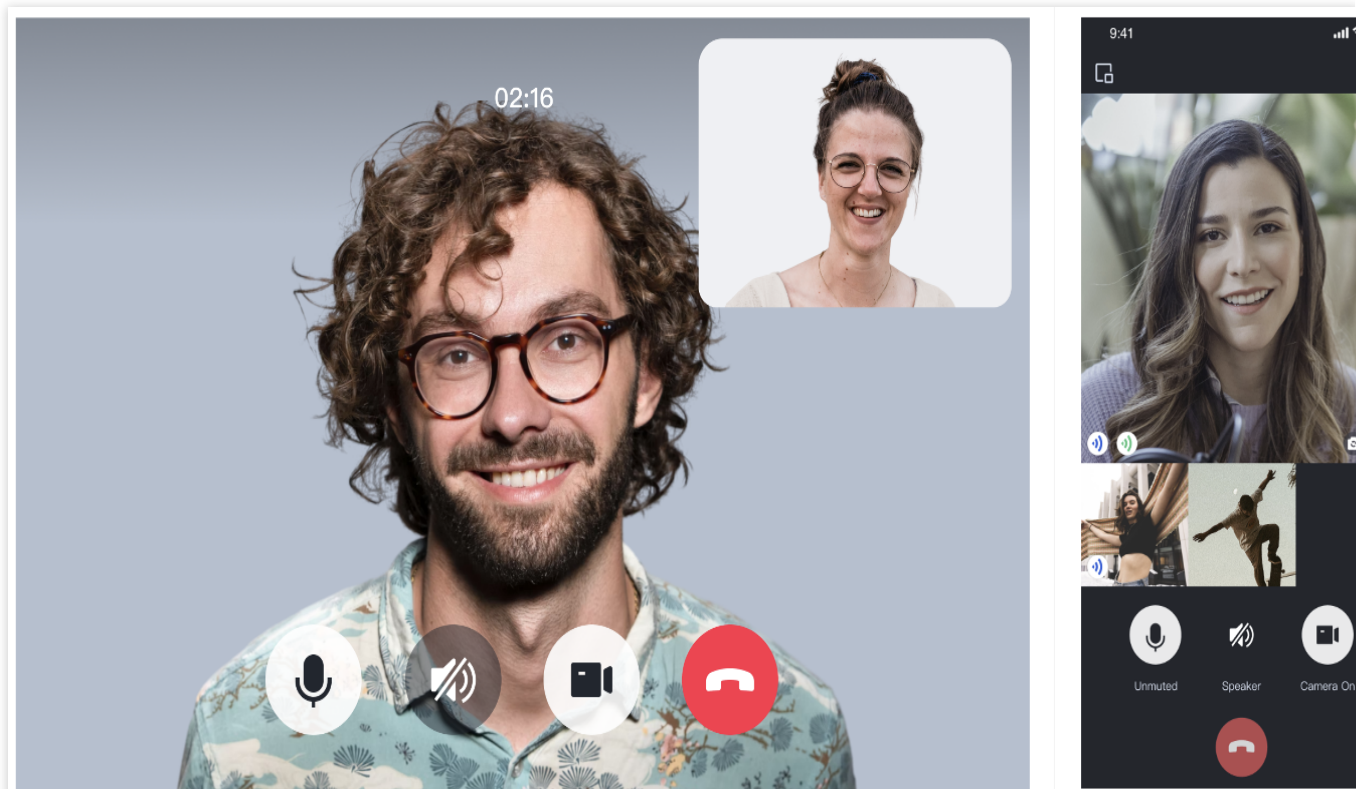
Suggestions and Feedback

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

Web&H5 (React)

Last updated : 2025-02-26 15:43:00

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-react";
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.calls API](#) to make a call.

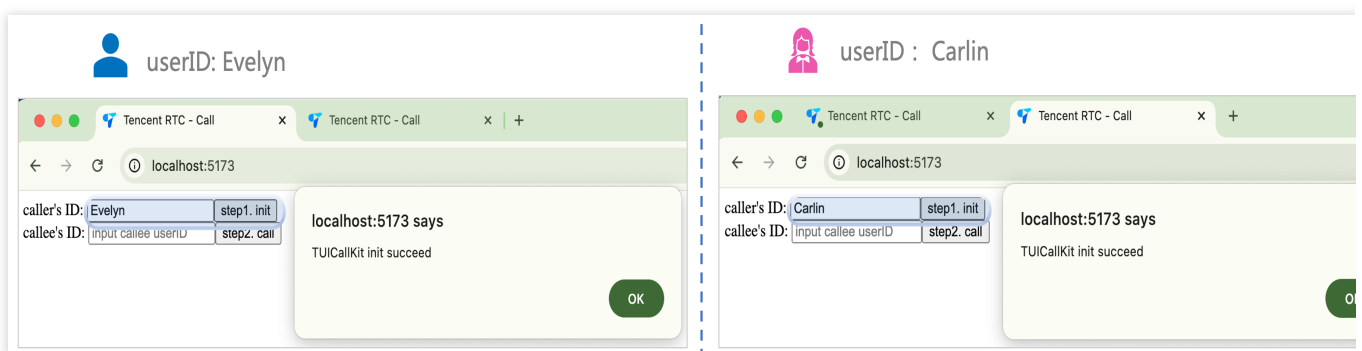
```
// 【3】 Make a 1v1 video call
const call = async () => {
  await TUICallKitServer.calls({
    userIDList: [calleeUserID],
    type: TUICallType.VIDEO_CALL,
  });
};
```

2. Run the project.

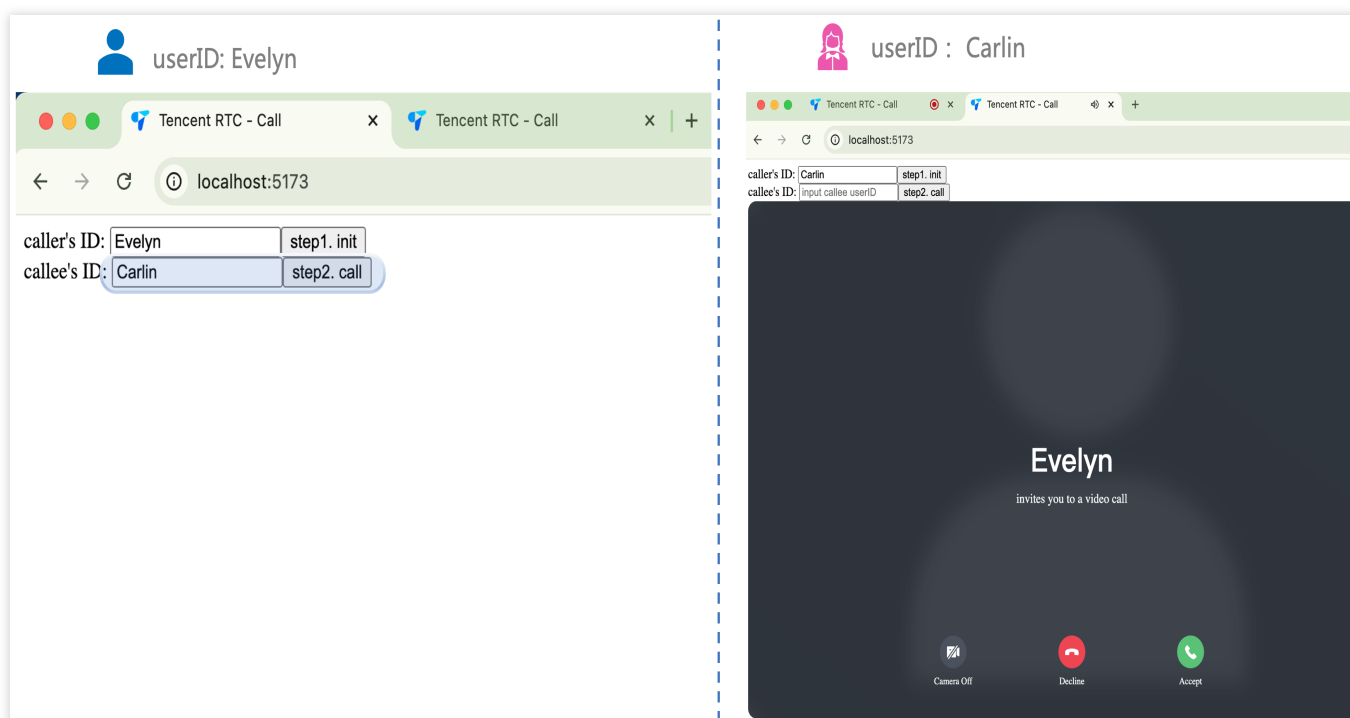
Warning:

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

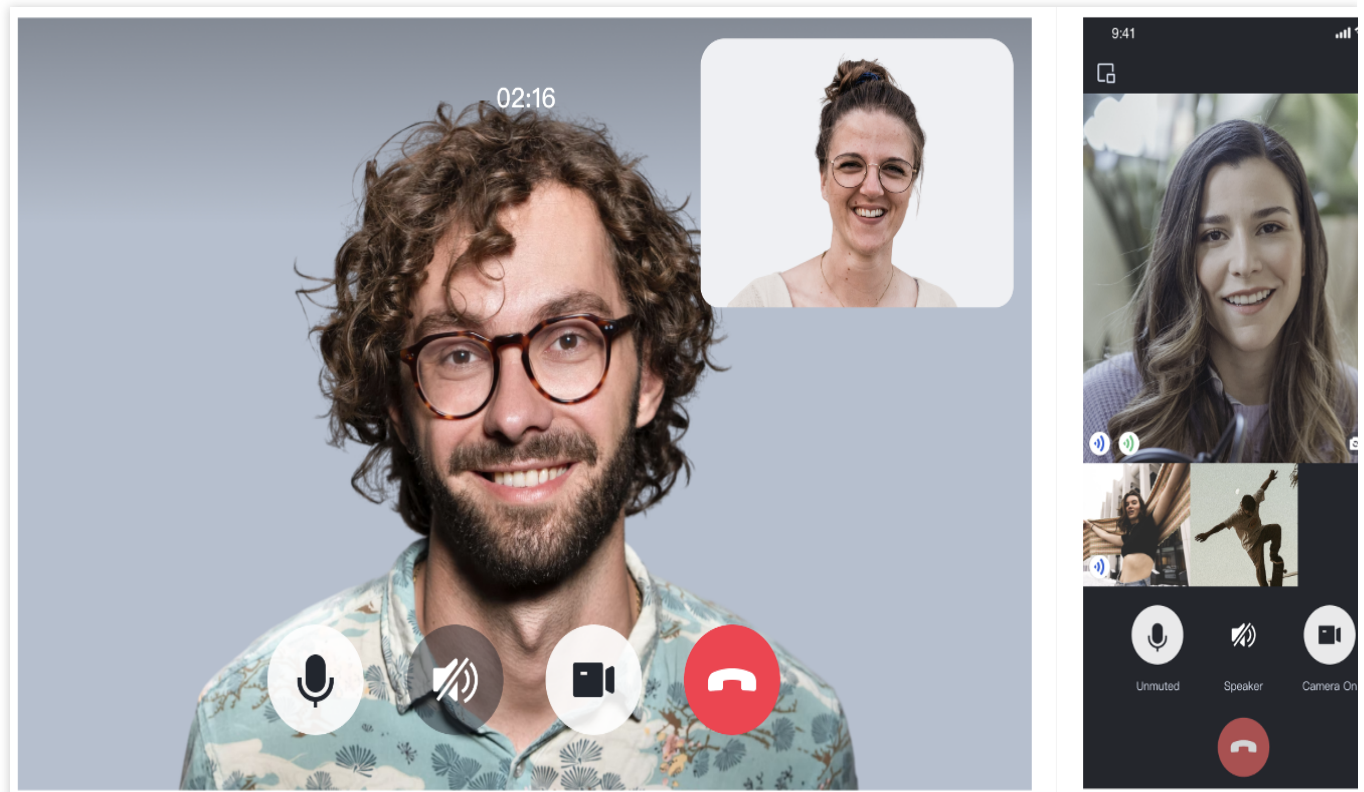
If you encounter any problems with access and use, please refer to [FAQs](#).

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

Web&H5 (Vue3)

Last updated : 2025-02-26 15:43:00

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.calls API](#) to make a call.

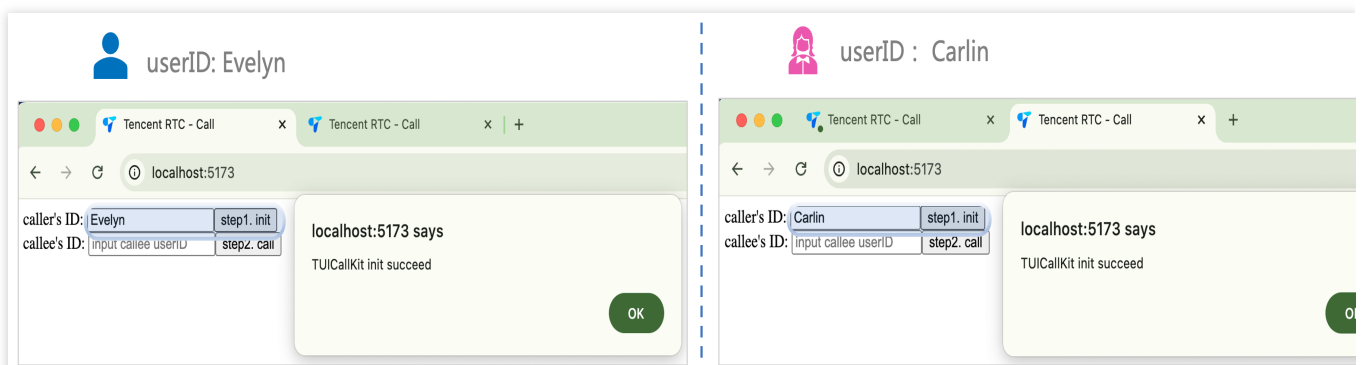
```
// 【3】 Make a 1v1 video call
const call = async () => {
  await TUICallKitServer.calls({
    userIDList: [calleeUserID.value],
    type: TUICallType.VIDEO_CALL,
  });
};
```

2. Run the project.

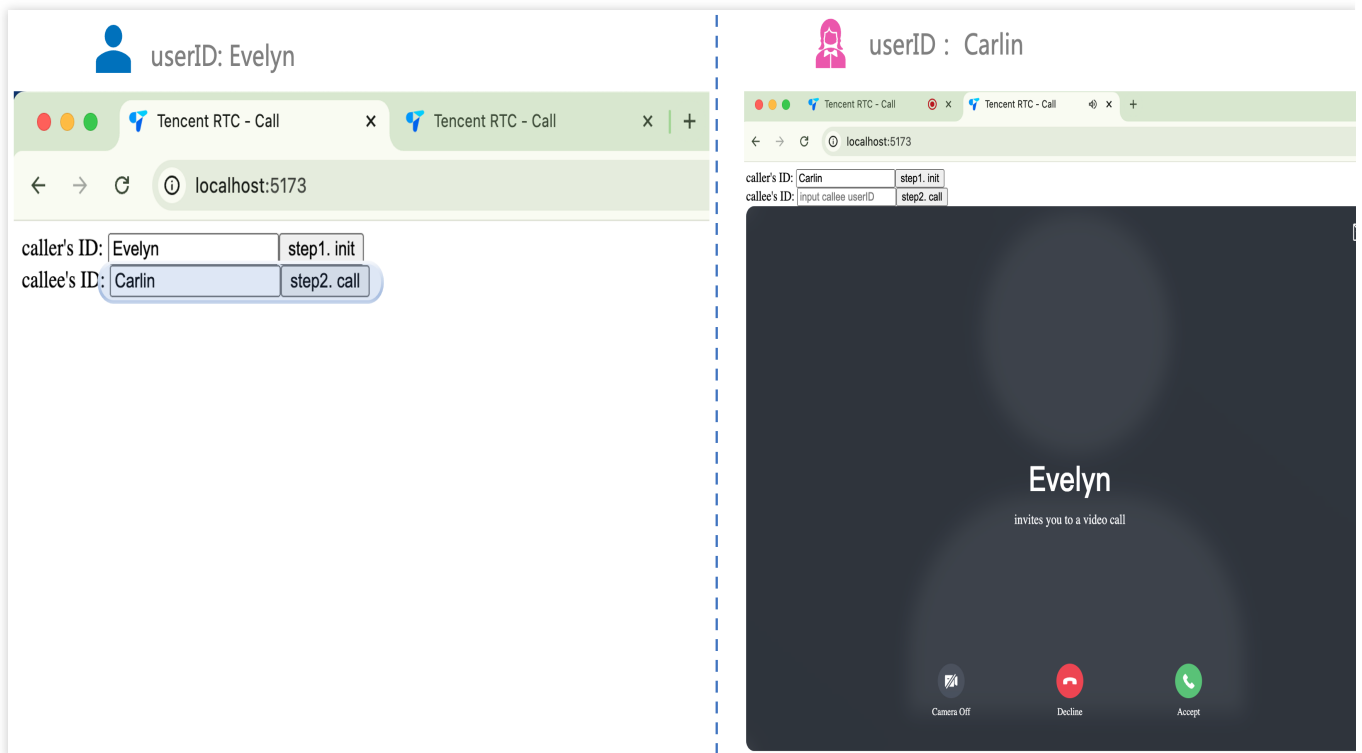
Warning:

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


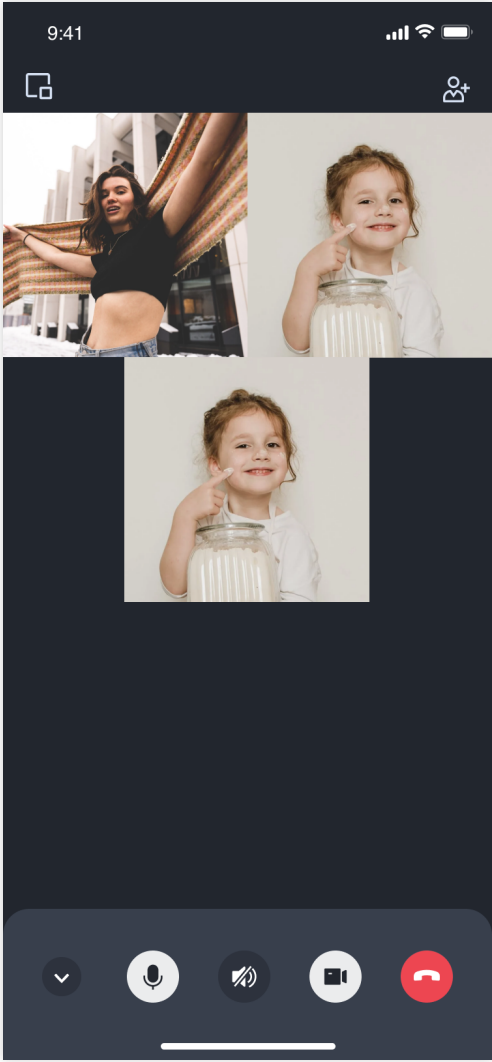
If you encounter any problems with access and use, please refer to [FAQs](#).

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

Flutter

Last updated : 2025-02-20 14:16:06

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  
}  
}
```

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.

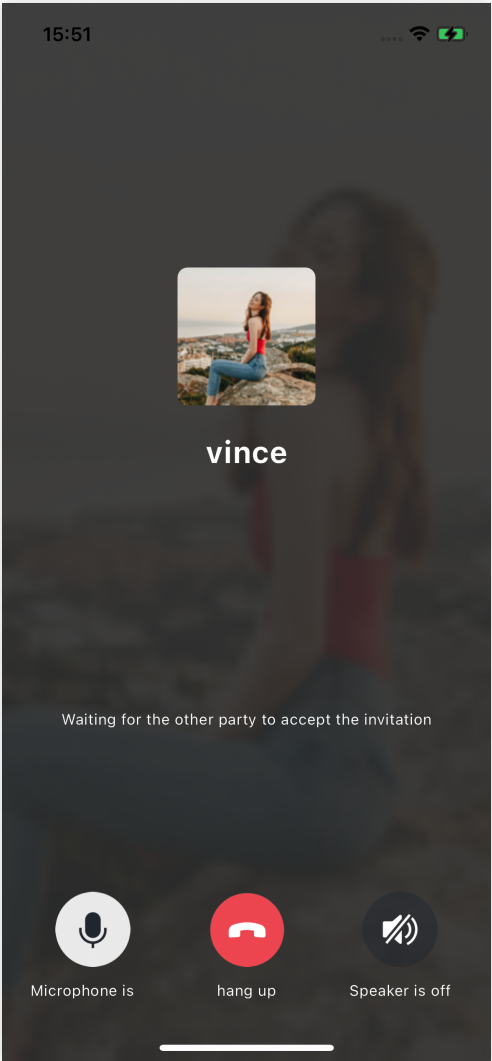
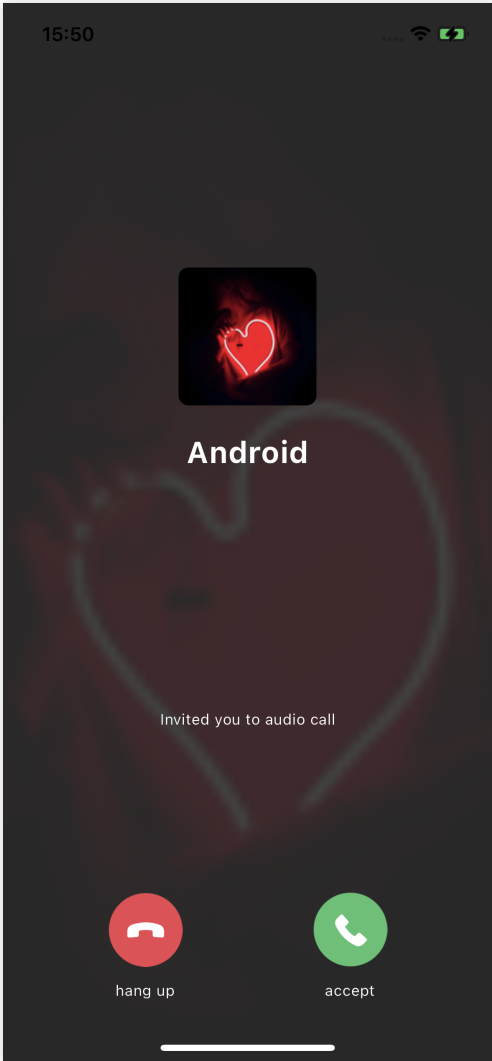
Production Environment: If your project is going to be launched, please adopt the method of [Server-side Generation of UserSig](#).

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.

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

void call() {
  List<String> userIdList = ['Android'];
  TUICallKit.instance.calls(userIdList, 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 : 2025-05-27 17:58:09

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. Create project, generate uin-app Application ID (AppID)

Open HbuilderX, click the bottom left corner to login to your uni-app account (if you don't have an account, please register first). After logging in, click manifest.json file of the project to generate the uni-app AppID.

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

3. 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'); // 【
  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.$TUICallKit.login( loginParams, res => {
                if (res.code === 0) {
                    this.isLogin = true;
                    this.inputID = '';
                    console.log('[TUICallKit] l
                } else {
                    console.error('[TUICallKit] login failed, failed messag
                }
            }
        );
    },
    handleCall() {
        try {
            const callParams = {
                userIDList: [this.inputID],
                callMediaType: 2,    // 1 -- audio c
                callParams: { roomID: 234, strRoomI
            };
// 【3】 start 1v1 video call
            uni.$TUICallKit.calls( 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](#))

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.


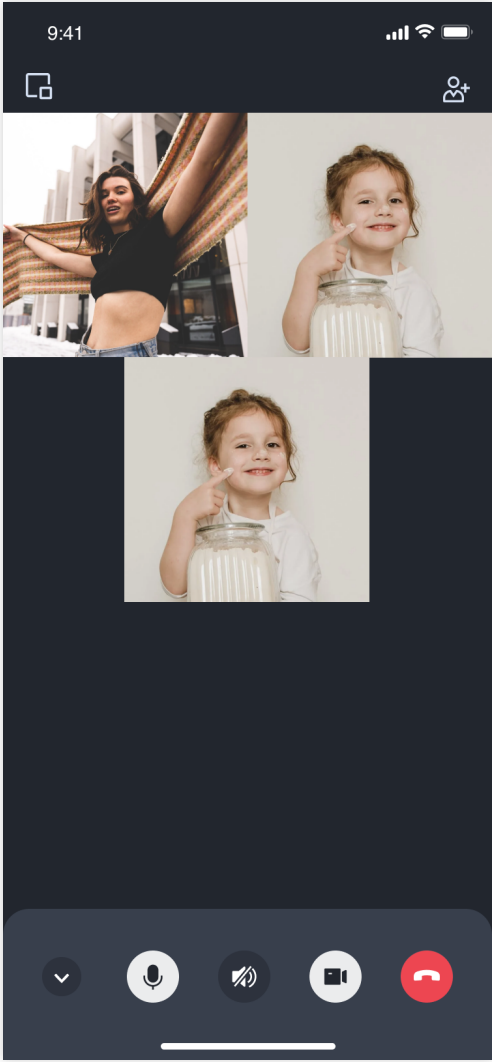
Technical Consultation

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

React Native

Last updated : 2025-03-07 15:28:30

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

Node.js version 16+.

two Mobile phone.

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-native](#) component.

```
yarn add @tencentcloud/call-uikit-react-native
```

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-native/src/debug ./src
```

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

Step 3. Login the TUICallKit

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

1. Import the call uikit.

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

2. using the [TUICallKit.login](#) API to log in to the component, you need to `fill in` `SDKAppID`, `SDKSecretKey` as two parameters in the code.

```
const handleLogin = async () => {  
  const userId = "denny"; // Please replace with your userId  
  const SDKAppID = 0; // Please replace with the SDKAppID obtained from s  
  const SecretKey = "****"; // Please replace with the SDKSecretKey obtained fr
```

```
const { userSig } = genTestUserSig({ userID: userId, SDKAppID, SecretKey });

TUICallKit.login(
  {
    sdkAppId: SDKAppID,
    userId,
    userSig,
  },
  (res) => {},
  (errCode, errMsg) => {}
);
```

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 .
SecretKey	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

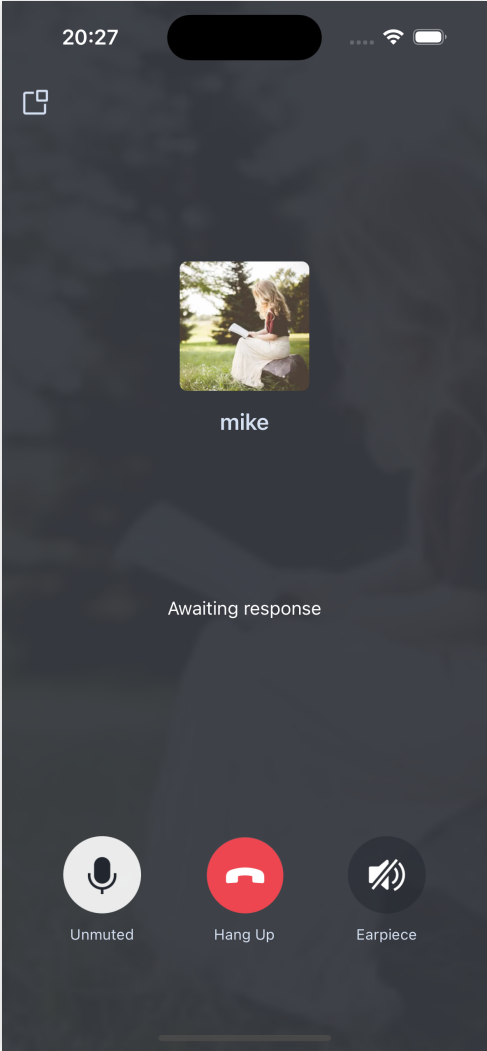
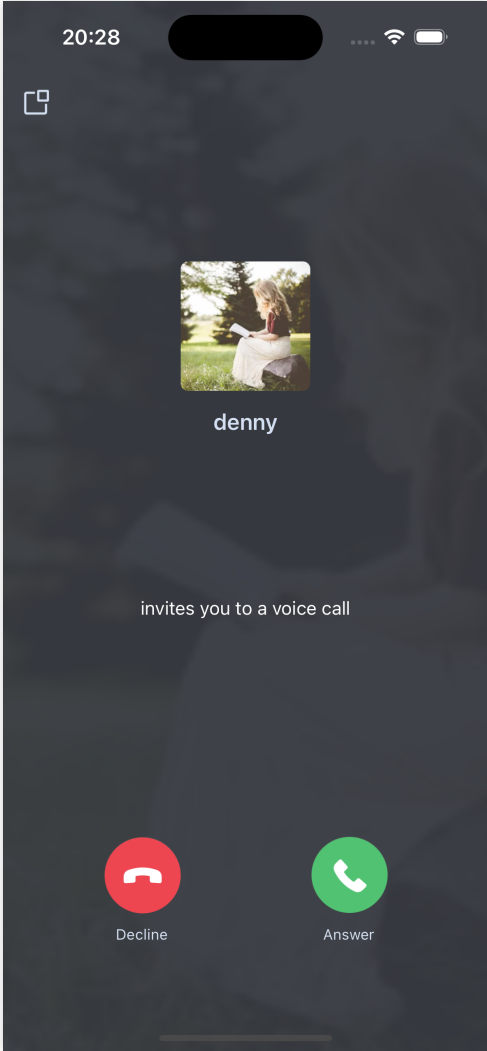
1. using the [TUICallKit.call API](#) to make a call.

```
// 【3】 Make a 1v1 video call
const users: string[] = ["mike"];
const call = async () => {
```



```
await TUICallKit.calls({
  userID: users,
  mediaType: MediaType.Video,
});
};
```

2. After both userID login to successfully, make a call..

Caller initiates an audio call	Callee receives an audio call request
	

FAQs

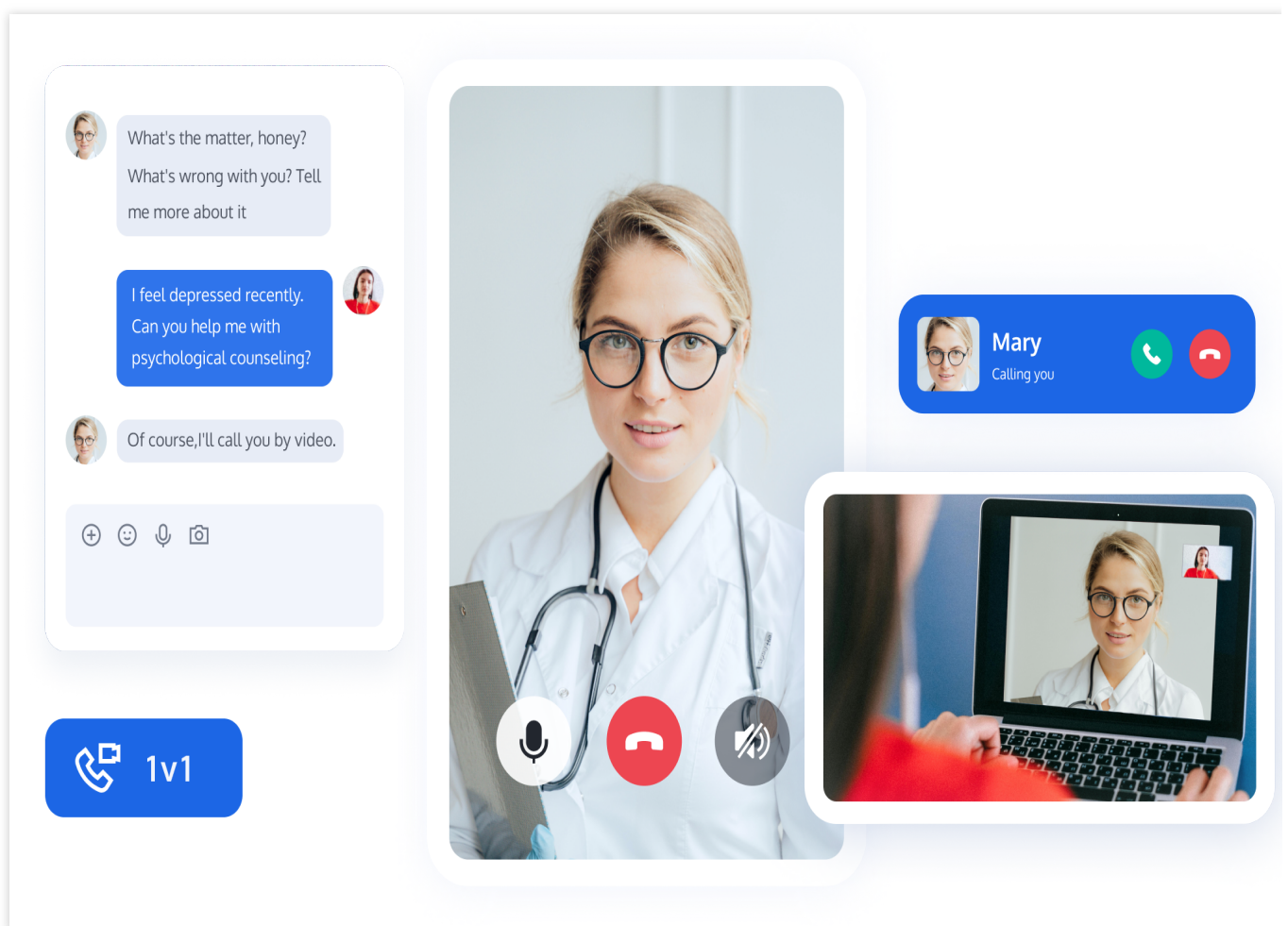
If you encounter any problems with access and use, please refer to [FAQs](#).
If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

Calls integration to Chat (TUICallKit)

Web&H5 (React)

Last updated : 2024-10-18 15:13:04

Chat offers multi-platform chat APIs, UI components, server-side APIs, and webhooks, enabling you to build a fully featured chat application in ten minutes. You can [experience the Chat UIKit Demo online](#). In Chat, you can integrate TUICallKit and add audio and video call features to your chat application with just a few lines of code.



React version 18+ (17.x versions are not supported)

TypeScript

[Node.js](#) version 16+

npm (use a version that matches the Node version in use)

Step 1: Quick Integration of Chat

For detailed steps, please refer to: [Quick Integration of Chat](#).

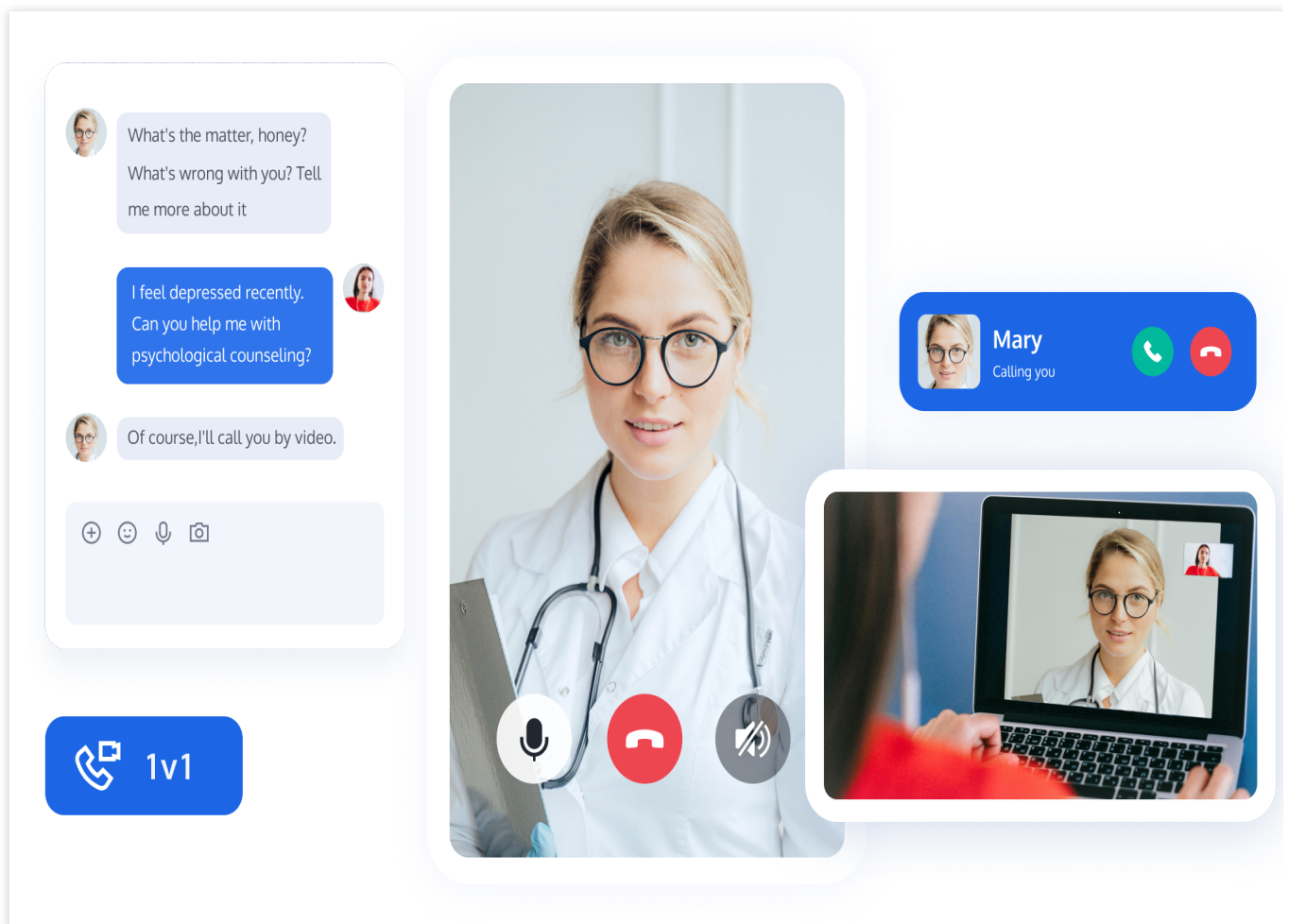
Step 2: Chat Integrates TUICallKit

For detailed steps, please refer to: [Chat Integrates TUICallKit](#).

Web&H5 (Vue3)

Last updated : 2024-10-18 15:13:45

Chat provides multi-platform chat APIs, UI components, server-side APIs, and webhooks, allowing you to build a fully-featured chat application in ten minutes. TUICallKit can be integrated into Chat, enabling audio and video call features with just a few lines of code.



Environment Requirements

Vue (Fully compatible with both Vue2 & Vue3. While incorporating below, please select the Vue version guide that matches your needs)

TypeScript (If you have a JS project, please refer to [How to Integrate TUIKit Components in a JS Project?](#) to configure progressive support for TS)

Sass (The version of sass-loader should be $\leq 10.1.1$)

node(node.js \geq 16.0.0)

npm (use a version that matches the Node version in use)

Quick Integration of Chat

For detailed steps, please refer to: [Quick Integration of Chat](#).

UI Customization (TUICallKit)

Android

Last updated : 2024-05-31 16:42:21

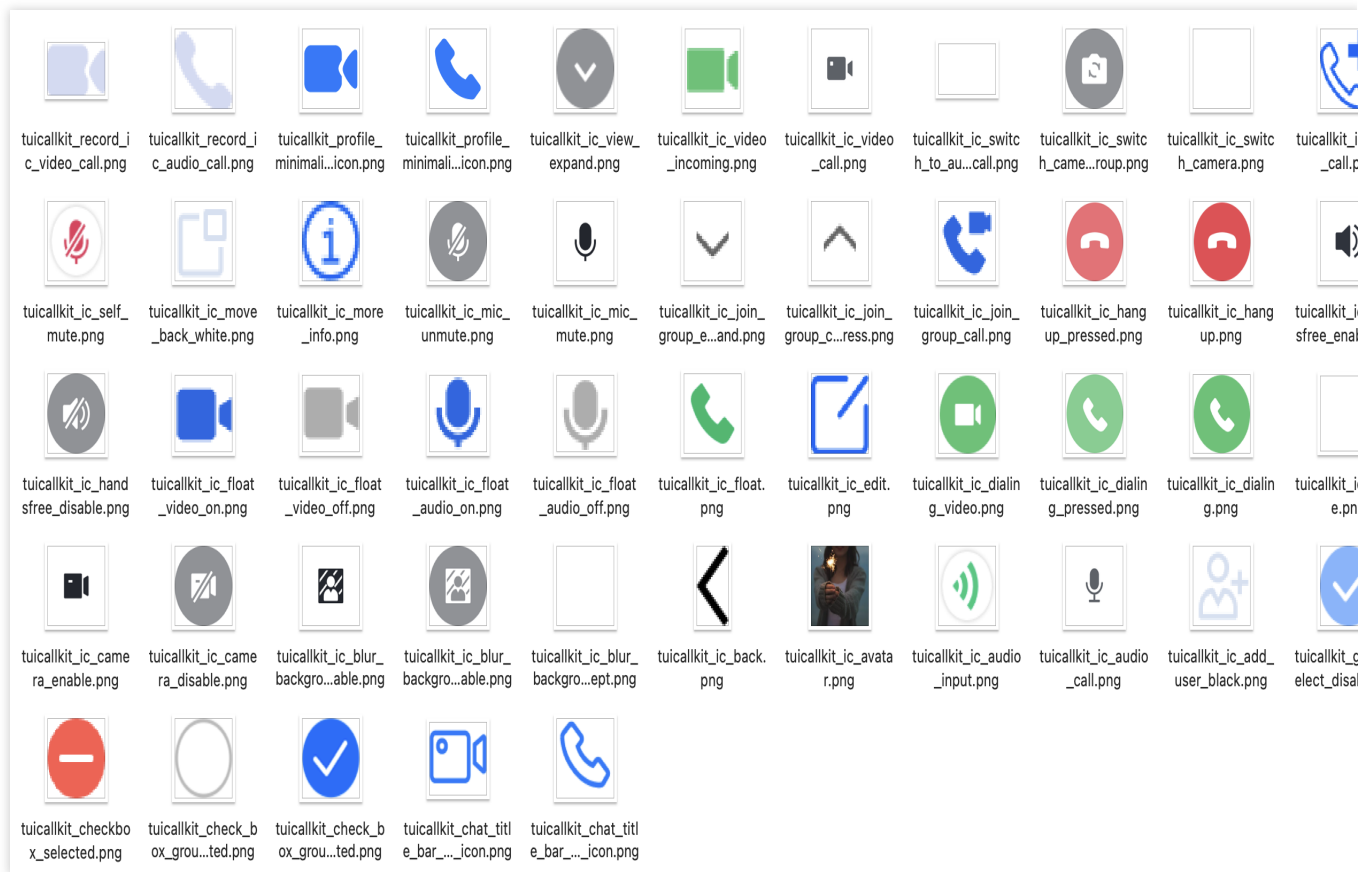
This document describes how to customize the UI of `TUICallKit` and provides two schemes for customization: **slight UI adjustment** and **custom UI implementation**.

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

setSelfInfo	Sets the user nickname and profile photo.
enableMultiDeviceAbility	Enables/Disables the multi-device login mode of <code>TUICallEngine</code> (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

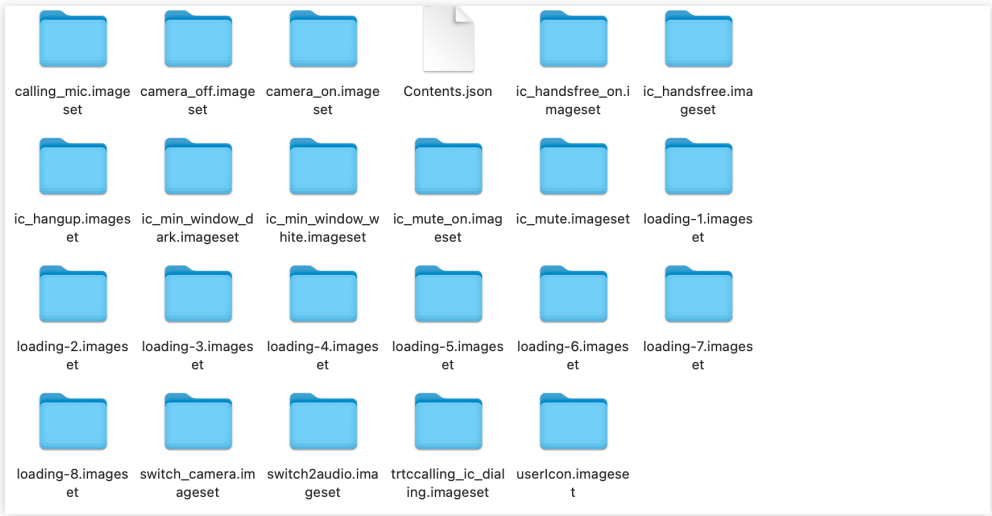
Last updated : 2024-05-24 18:35:12

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

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

TUICallObserver

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

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

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

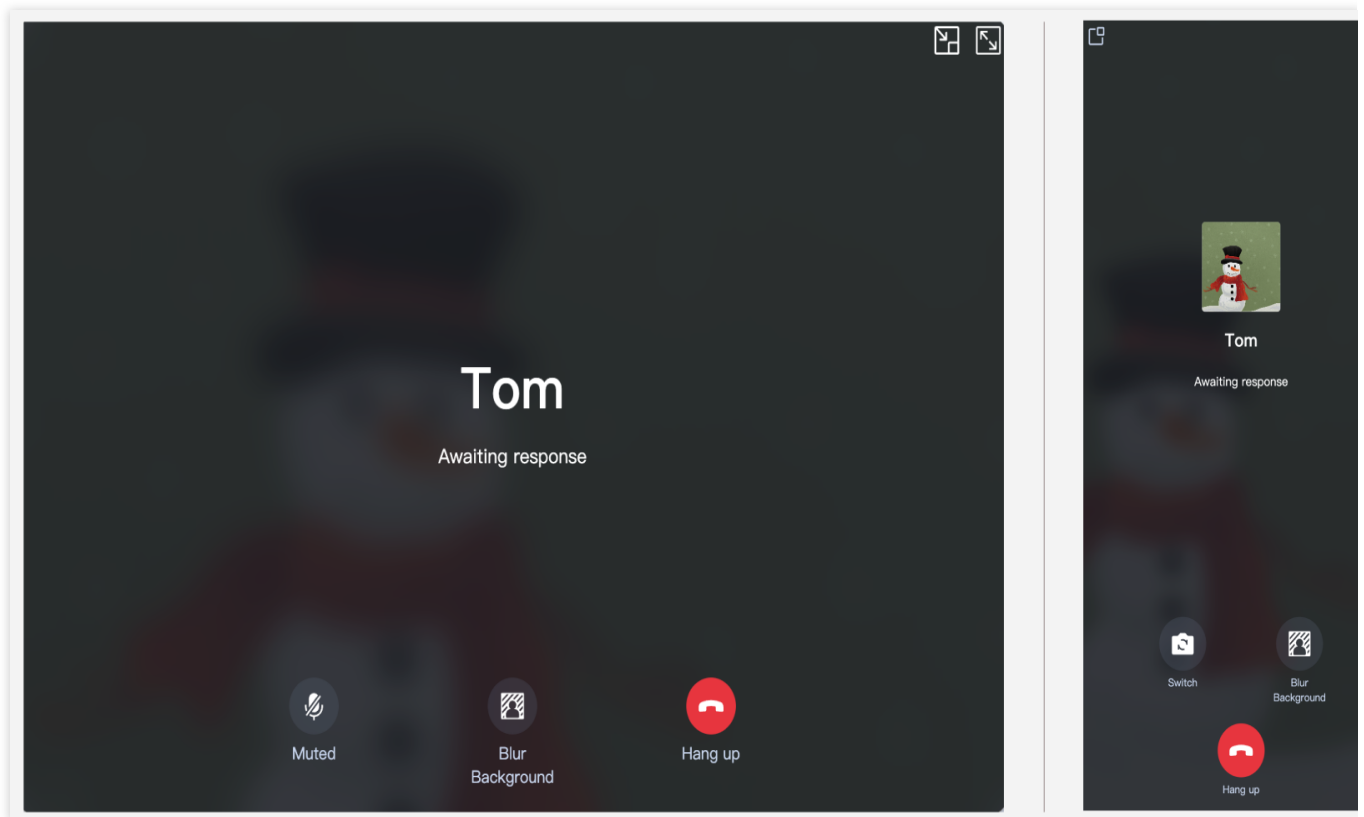
Button Hiding

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";

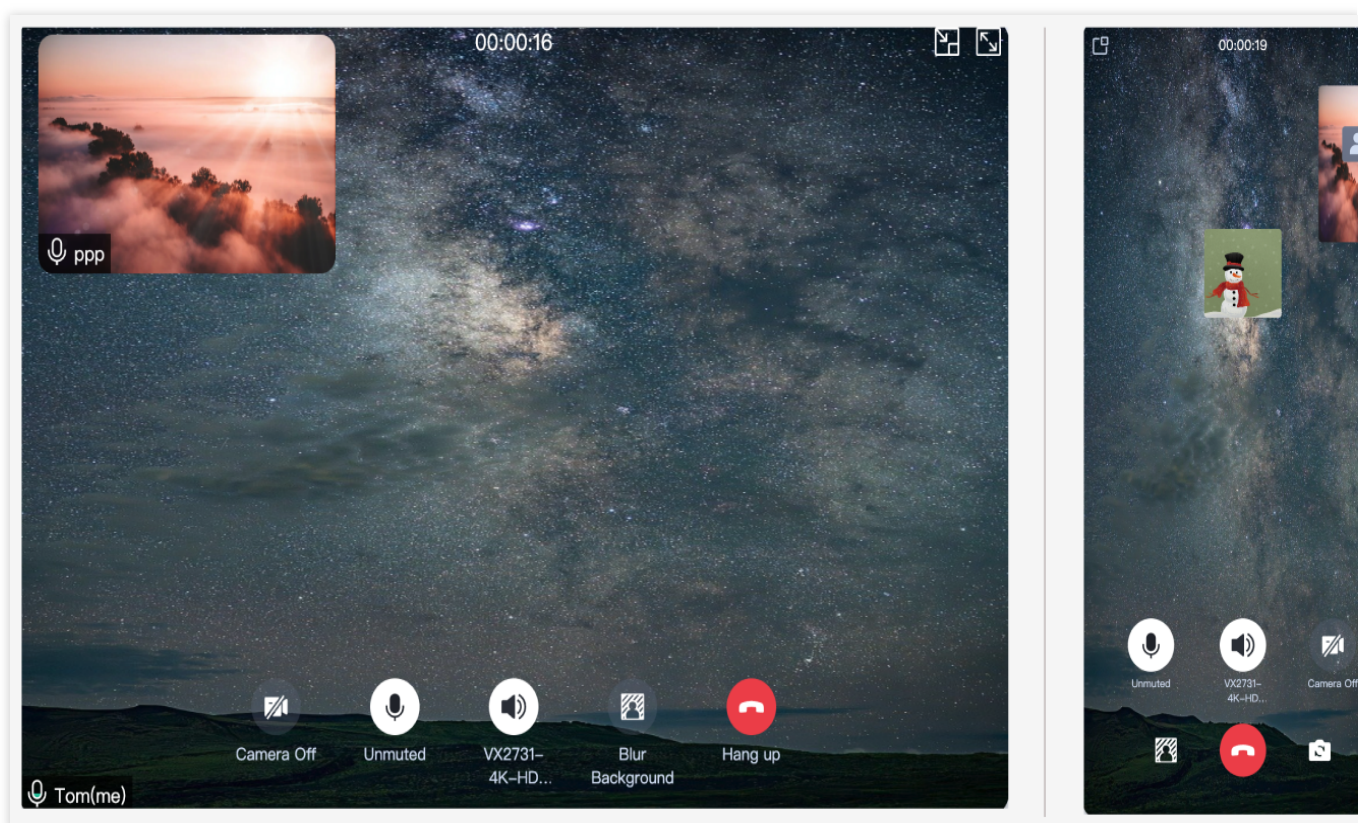
TUICallKitServer.hideFeatureButton(FeatureButton.Camera);
```

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');
```

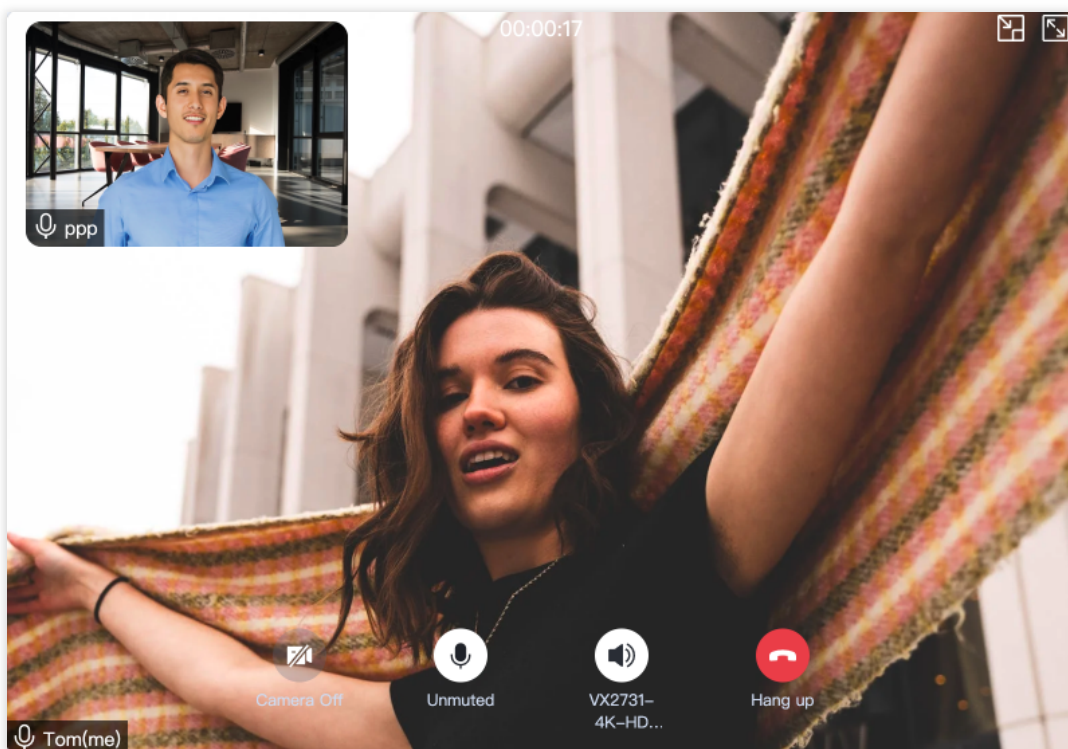
Set Layout

Note :

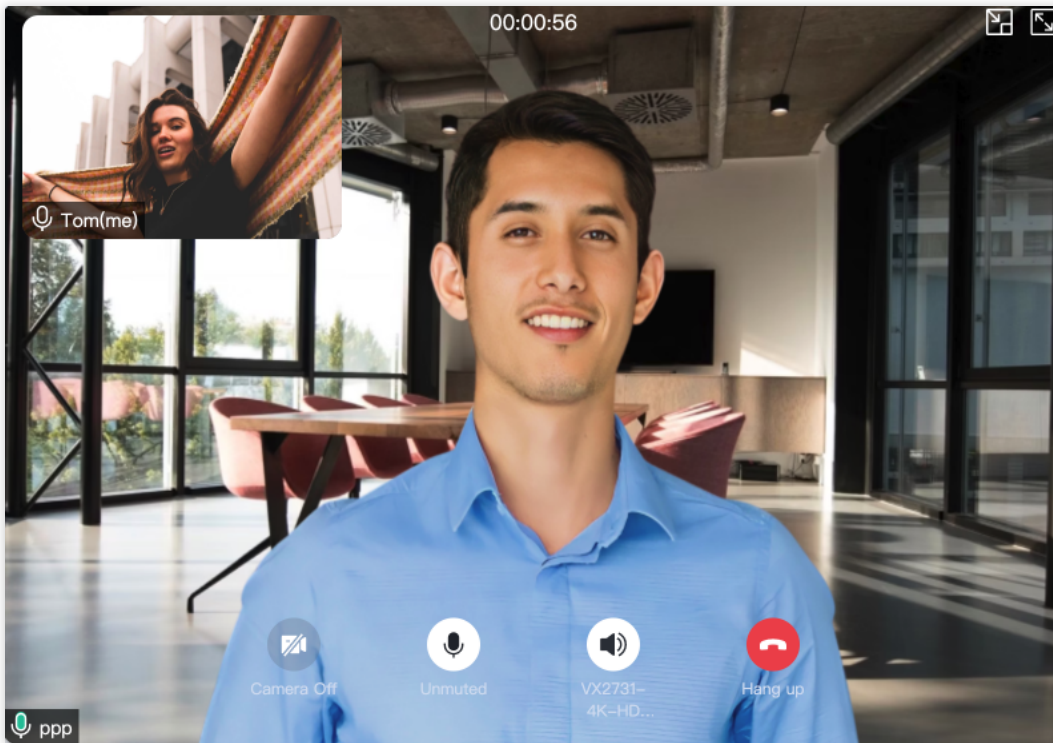
Only available for 1V1 video calls, supported from v3.3.0+.

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-uikit-vue/* ./src/components/TUICallKit
```

```
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
  }
}
```

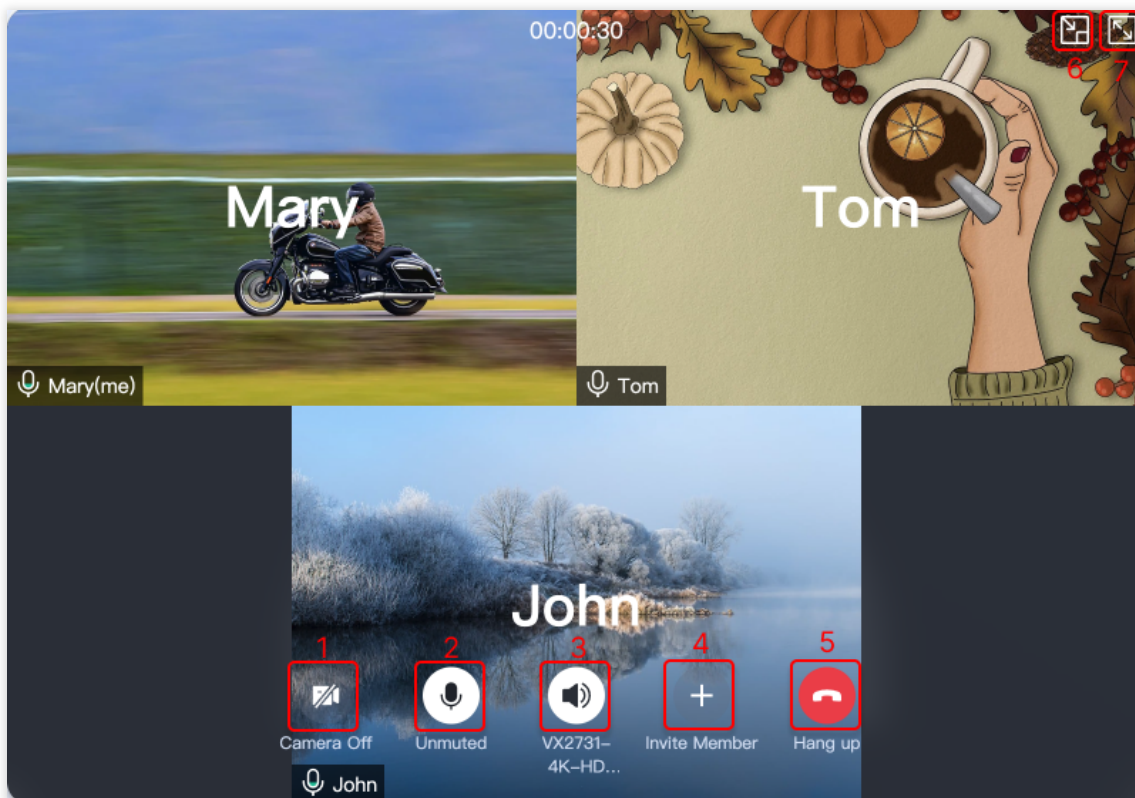
5. Modify the icon components in the TUICallKit/Components/assets folder

Note:

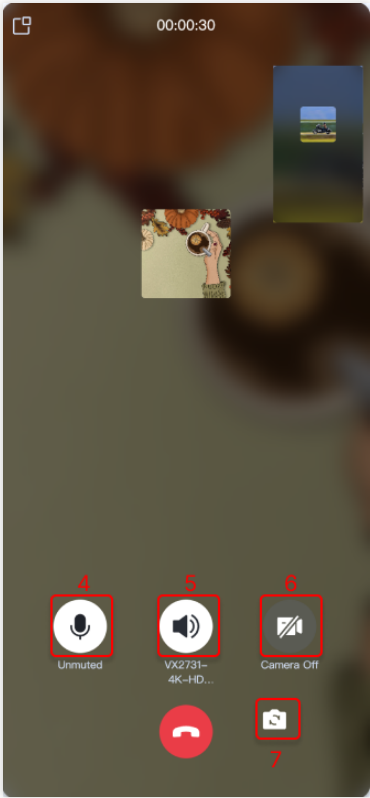
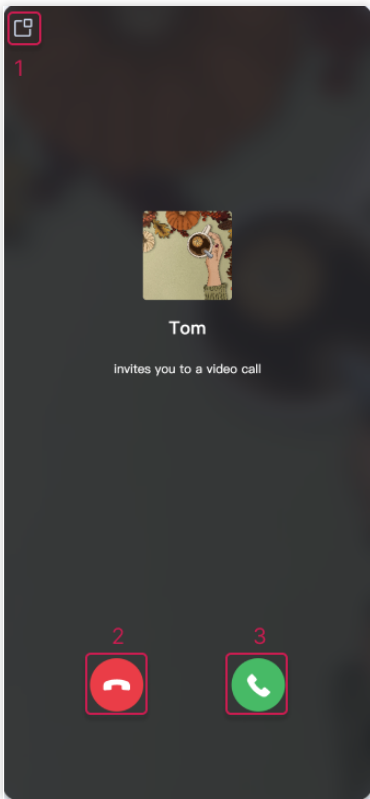
To ensure the icon color and style remain consistent throughout the application, please keep the icon file name unchanged when replacing.

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

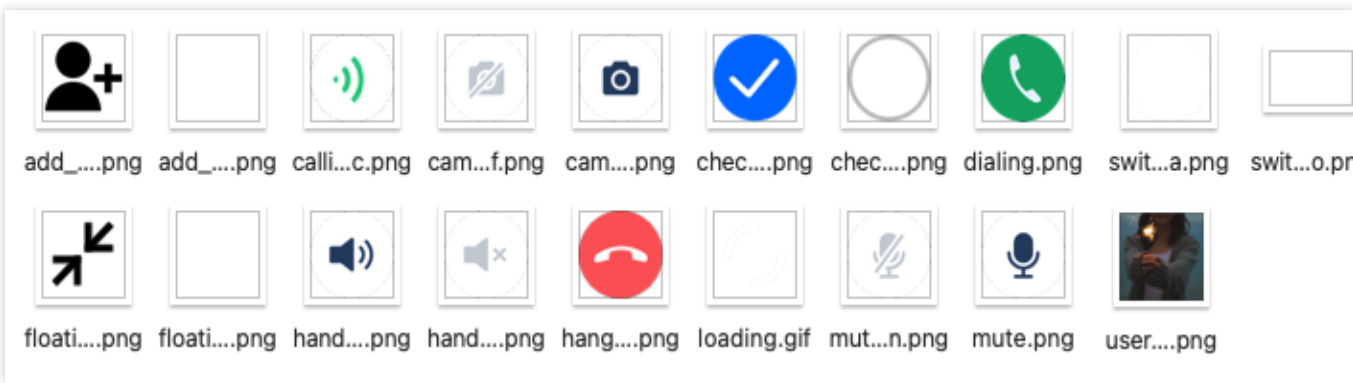
You can download the latest version of the [tencent_calls_uikit](#) plugin locally, and then use the local dependency method to access the plugin in your project. The local dependency method is as follows:

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 : 2025-02-26 14:45:27

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.

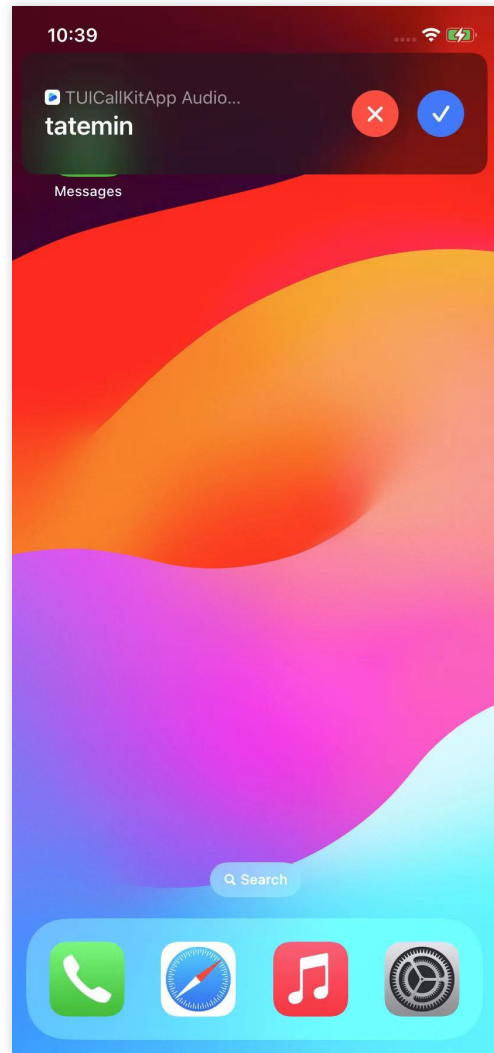
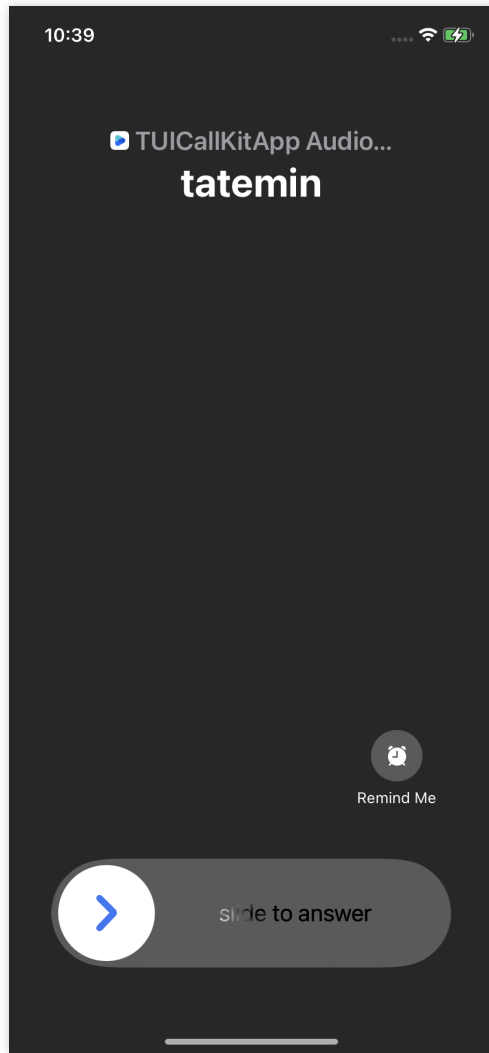
Note:

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.


Program resources



App Store Connect

Manage your app's builds, metadata, and more on the App Store.


- Apps
- Analytics
- Trends
- Reports
- Users and Access
- Agreements, Tax, and Banking



Certificates, IDs & Profiles

Manage the certificates, identifiers, profiles, and devices required to develop, test, and distribute apps.

- Certificates**
- Identifiers
- Devices
- Profiles
- Keys
- Services



Additional resources

Download beta software, and view and manage your usage of developer services.

- Software Downloads
- Feedback Assistant
- Xcode Cloud
- CloudKit
- MapKit JS
- Push Notifications
- WeatherKit

2. Click + next to Certificates.

Apple Developer

Certificates, Identifiers & Profiles

Certificates

Identifiers

Devices

Profiles

Keys

Services

Certificates

+

NAME

TYPE

PLATFORM

CREATED BY

EXPIRATION

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

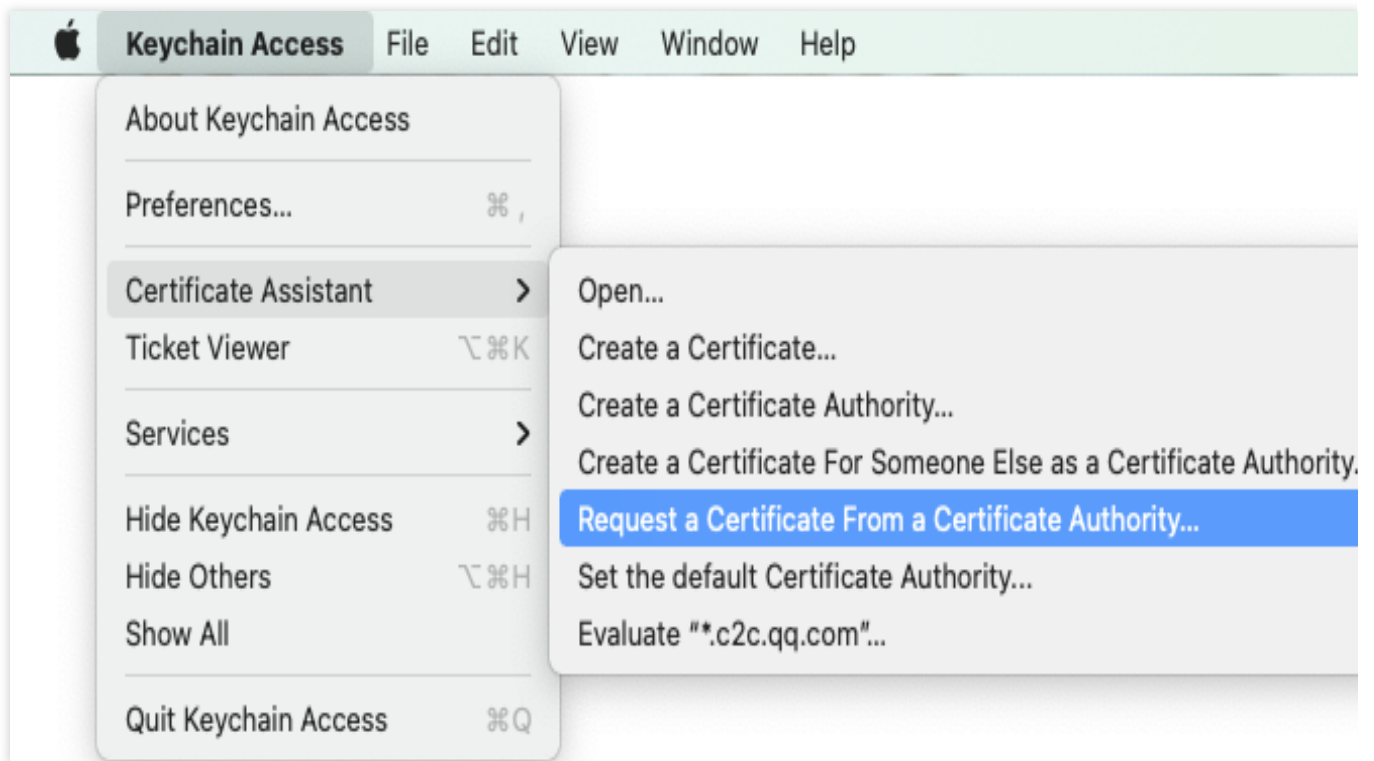
4. In the **Select an App ID for your VoIP Service Certificate** tab, select your app's BundleID, and click **Continue**.

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.

Certificates, Identifiers & Profiles

[< All Certificates](#)

Create a new VoIP Services Certificate

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

Upload a Certificate Signing Request

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

[Choose File](#)

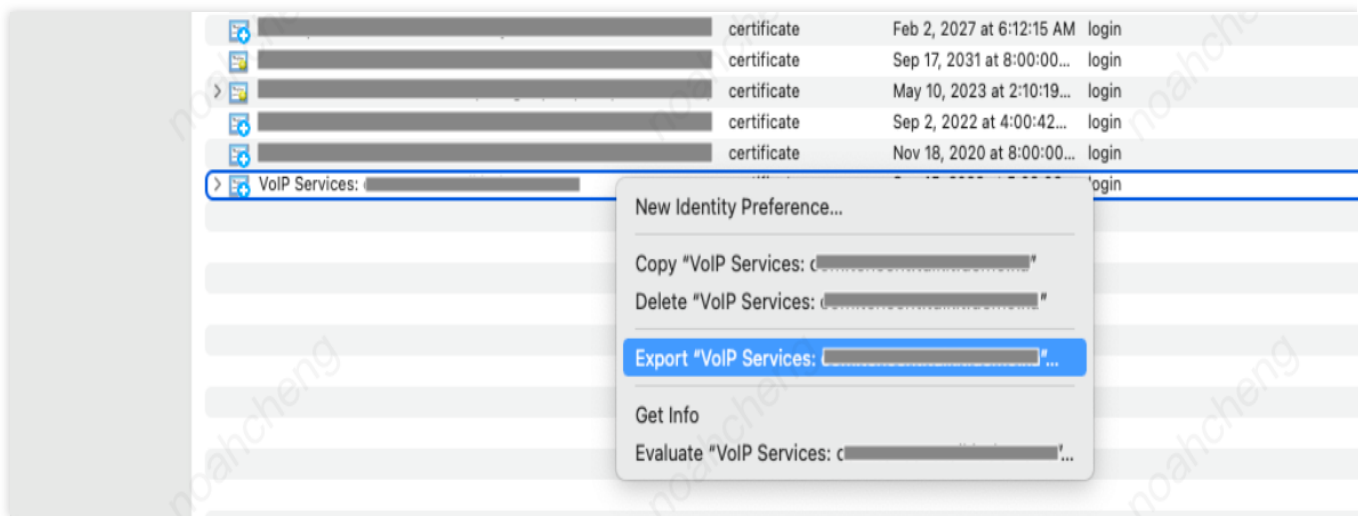
8. Click **Continue** to generate the certificate, and click **Download** to download the corresponding certificate to your local device.

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.

Chat

Application management

Configuration

Overview

Account Management

Group Management

Feature Configuration

Webhook Configuration

Daily Statistics

Plugin

Push

AI Chatbot

Tools

Real-Time

Auxiliary Tools

Integration Guide

Basic Configuration 20006172 - TUICallKit Current data center: Singapore Telegram group WhatsApp group

Standard Billing Plan

Status In use

Plan TRTC Developer

Expiration time 2024-02-04

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

App Information [Edit](#)

SDKAppID 20006172

Application Name TUICallKit

Application Type -

Application Introduction -

Basic info

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

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

Creation time 2024-01-04

Last Modified 2024-01-04

Tag Configuration [Edit](#)

TagKey TagValue

Offline Push Certificate Configuration [What is offline pus](#)

The push configuration module has been moved to the [Access Settings] page under the [Push Management] directory, the left [Go now](#)

RTC Engine [Act](#)

RTC Engine can help you implement audio and video calls, multi-person voice chats, video conferencing and other functions in IM applications. RTC Engine is billed independently, see [Price Overview](#) for details. RTC Engine does not distinguish between data centers. Your configuration and log data using RTC Engine will be stored in Singapore by defa

Call [Integration Guide](#) [Product Overvie](#)

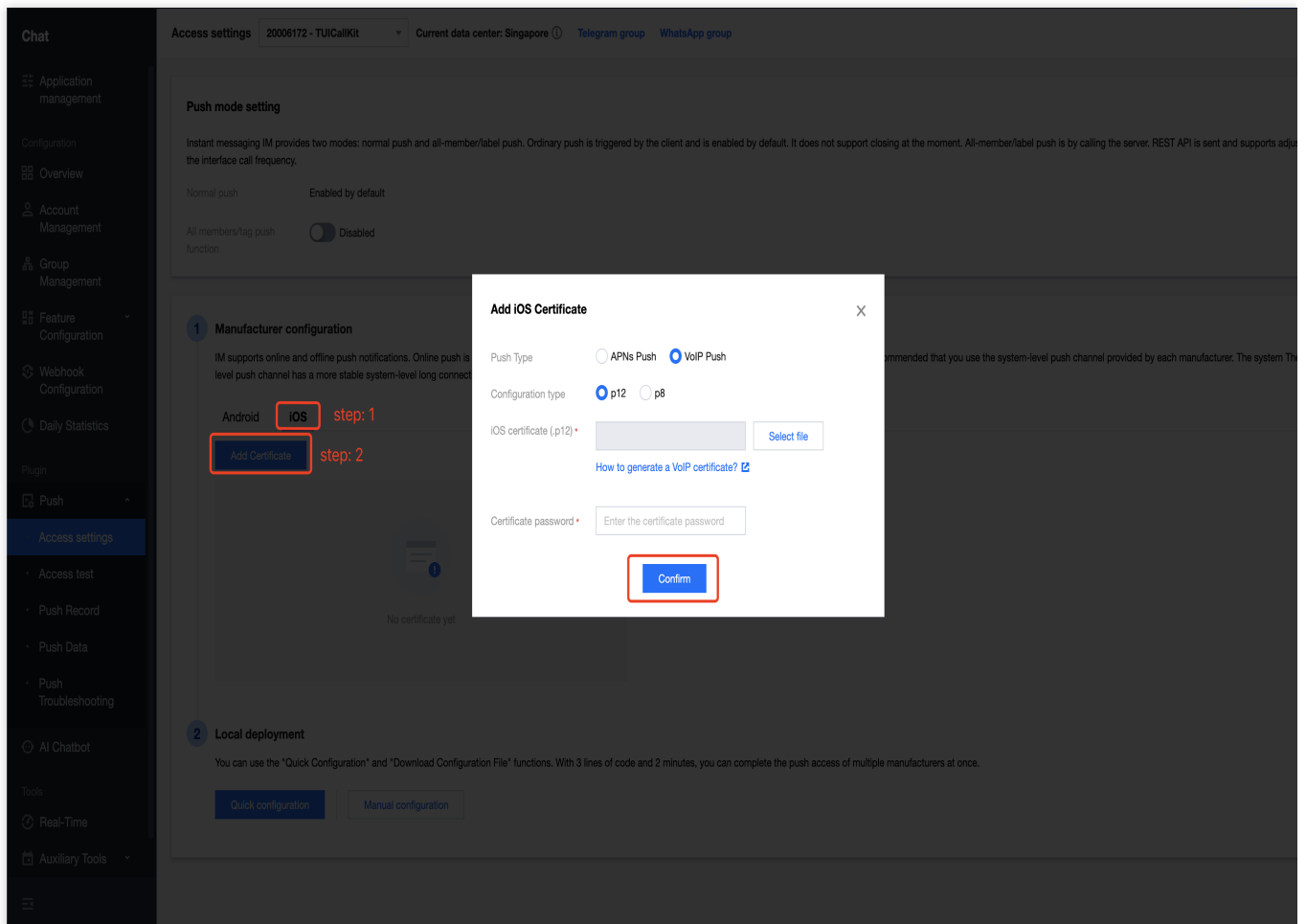
TRTC Call is a call component that comes with a UI kit. You can use it to implement features including 1-to-1/group call and multi-platform call making/answering.

Status Activated

Current version [Trail](#) [Learn more](#) [Purchase](#)

Expiration time 2024-01-11

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

**Note:**

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

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

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

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

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

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

1

Manufacturer configuration

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

Android

iOS

Add Certificate

ID: 1f

Delete Edit

Certificate information

p12

Push Type

VoIP Push

Certificate Type

Production environment

mutable-content

Enabled

Certificate password

123321

Expiration time

2024-03-19 15:29:01

ID: 15853

Delete Edit

Certificate information

p12

Push Type

VoIP Push

Certificate Type

Development Environment

mutable-content

Enabled

Certificate password

123321

Expiration time

2024-03-19 15:29:01

2

Local deployment

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

Quick configuration

Manual configuration

Note:

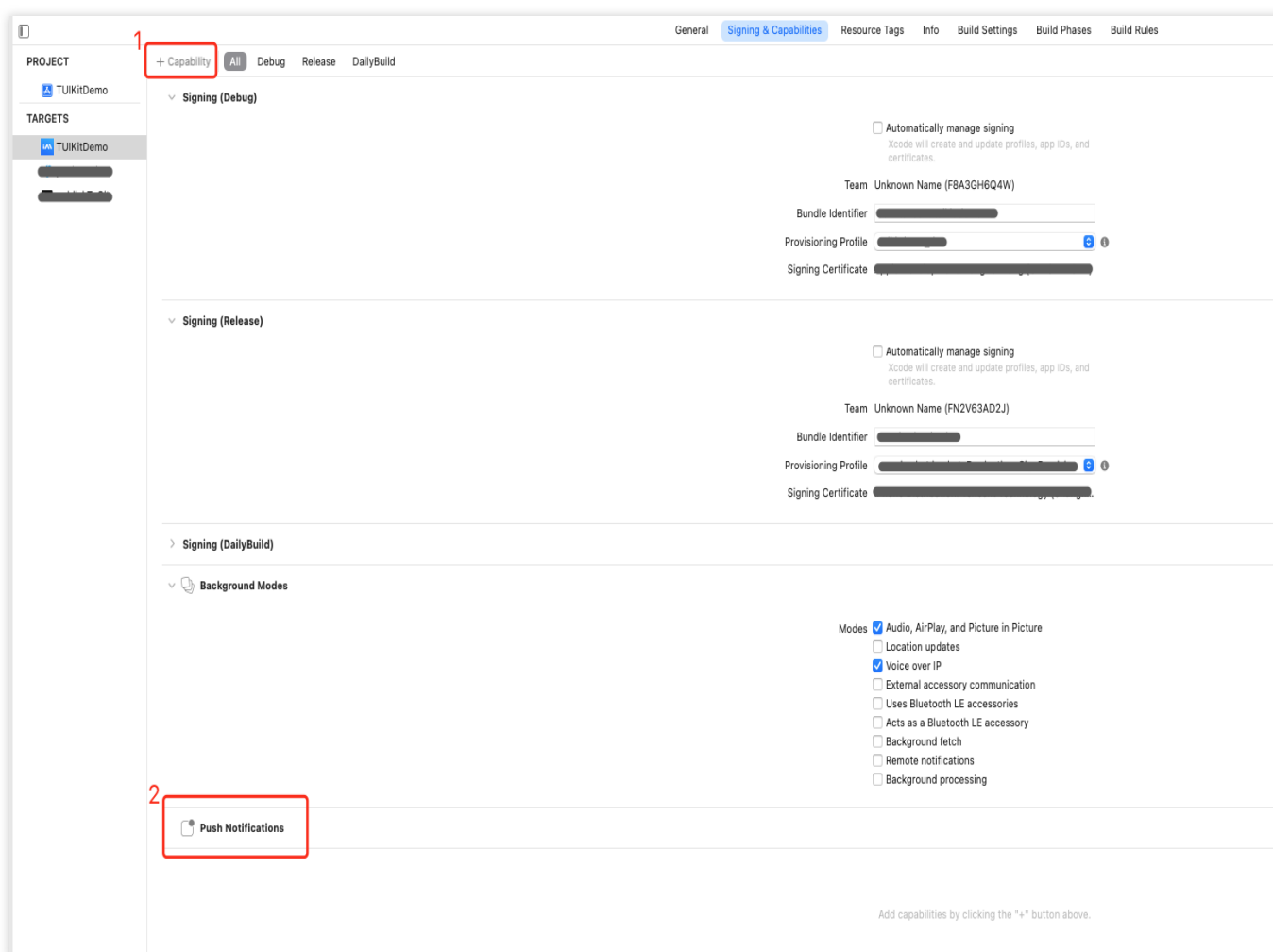
The certificate IDs for the development and production environments must be strictly distinguished and filled in according to the actual environment in [Step 4: Integrate the TUICallKitVoIPExtension component](#).

Step 3: Complete the project configuration

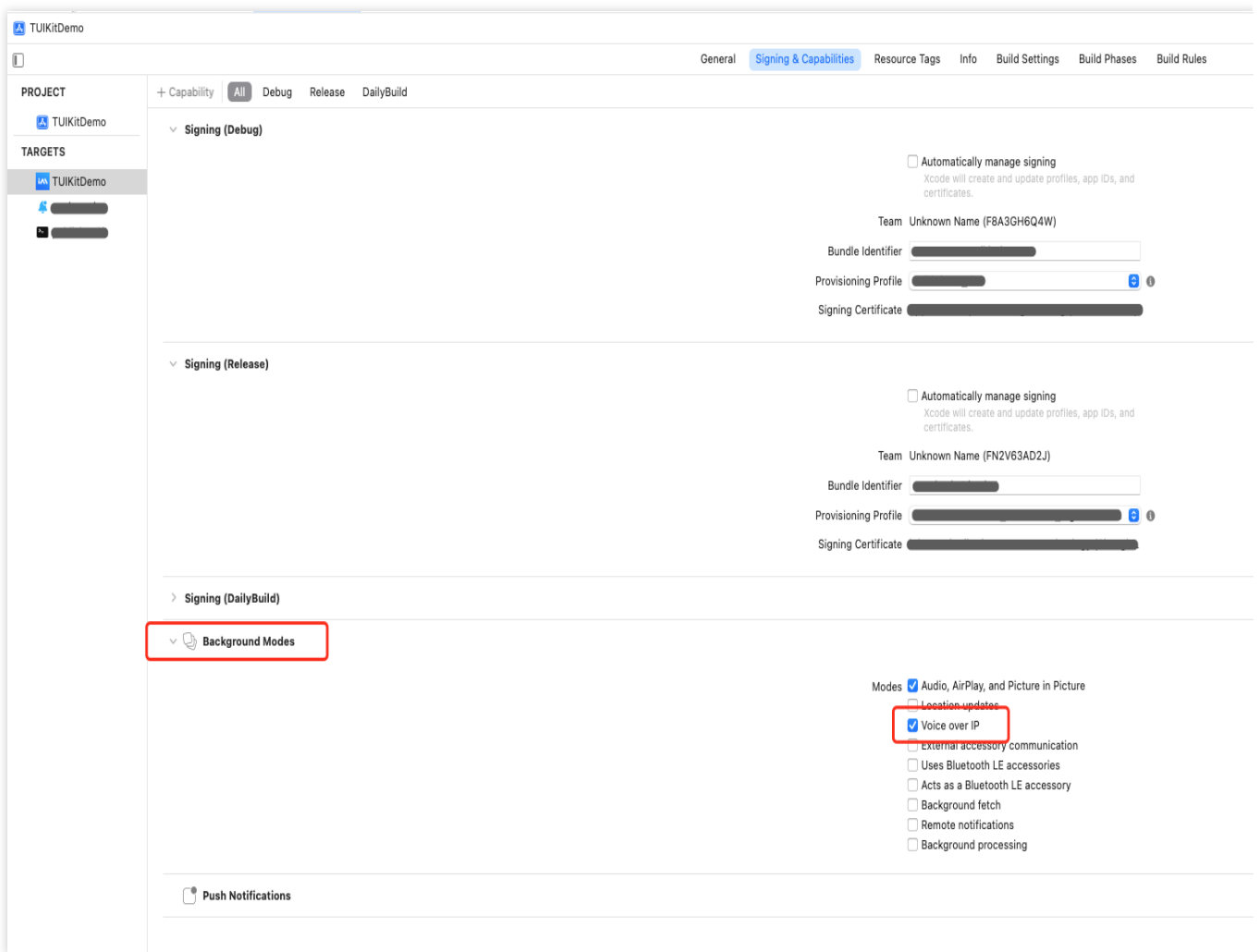
1. As shown below, confirm whether the Push Notification capability has been added to your project's capabilities.

©2013-2025 Tencent Cloud International Pte. Ltd.

Page 115 of 582



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 'TUIVoIPExtension'
```

Note :

Please make sure to specify the same `Subspecs` for `TUICallKit_Swift` and `TUICallKitVoIPExtension` components in your `Podfile`.

2. Execute the command below to install the component.

```
pod install
```

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

Swift

Objective-C

```
import TUIVoIPExtension
```

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) throws Bool {
    // Report certificate ID
    TUIVoIPExtension.setCertificateID(1234)

    return true
}

#import <TUIVoIPExtension/TUIVoIPExtension.h>

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Report certificate ID
    [TUIVoIPExtension 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 calls. 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().calls(userIdList: ["123456"], callMediaType: .audio, pa

} 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;

List<String> list = new ArrayList<>();
list.add("mike")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Video, param
```

```

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.calls(['vince'], TUICallMediaType.audio, params);

#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <RTCRoomEngine/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 `ChannelID`
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelID"
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery r
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/developmen
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}

```

```
[[TUICallKit sharedInstance] calls:@[@"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 **TUICallKitVoIPEExtension** 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 TUIVoIPEExtension

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

/// Implementation in SceneDelegate
func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connect
    for userActivity in connectionOptions.userActivities {
        TUIVoIPEExtension.call(with: userActivity);
    }
}

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

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

```

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

/// Implementation in SceneDelegate
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)sessionoptions
    [connectionOptions.userActivities enumerateObjectsUsingBlock:^(NSUserActivity *userActivity, NSUInteger idx, BOOL *stop) {
        [TUIVoIPEExtension callWith:userActivity];
    }];

- (void)scene:(UIScene *)scene continueUserActivity:(NSUserActivity *)userActivity
    [TUIVoIPEExtension 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 TUIVoIPEExtension

func application(_ application: UIApplication, continue userActivity: NSUserActivity withActivityInfo: UIUserActivityInfo?) -> Bool {
    TUIVoIPEExtension.call(with: userActivity)
    return true
}

#import <TUIVoIPEExtension/TUIVoIPEExtension.h>

- (BOOL)application:(UIApplication *)application continueUserActivity:(nonnull NSUserActivity *)userActivity
    [TUIVoIPEExtension 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 : 2025-04-07 10:56:30

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
```

```
# Uncomment 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 `calls`.

Swift

Objective-C

```
import TUICallKit_Swift
import RTCRoomEngine

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

TUICallKit.createInstance().calls(userIdList: ["123456"], callMediaType: .audio, pa

} fail: { code, message in

}
```

```

#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <RTCRoomEngine/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 `ChannelID`
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelID"
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}

// Single person call example, similar for group call
[[TUICallKit sharedInstance] calls:@[@"123456"]
                             callMediaType:TUICallMediaTypeAudio
                             params:[self getCallParams] succ:^(
} fail:^(int code, NSString * _Nullable errMsg) {

}];

```

Advanced features: continuous vibration, ringing

APNs (Apple Push Notification service) push can specify features such as pushing audio files, but it cannot vibrate or ring continuously. The notification duration is short, and inbound call information is easy to miss in

audio/video call scenarios. You can use our [Demo experience APNs continuous vibration and ringing feature](#).

1. Ring

APNs push can set the `iOSSound` field in `TUIOfflinePushInfo` of params when calling `call`, `calls`, `groupCall` APIs to make a phone call. Pass the audio filename to `iOSSound`.

Note:

If you want to customize `iOSSound`, you need to first link the audio file into the Xcode project, and then set the audio filename (with extension) to `iOSSound`.

Ringtone duration should be less than 30 s.

Objective-C

Swift

```
[[TUICallKit sharedInstance] call:@"mike's id" params:[self getCallParams] callMedi

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

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

let params = TUICallParams()
```



```
let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = "TUICallKit"
pushInfo.desc = "TUICallKit.have.new.invitation"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO must set ChannelID before it can receive push messages. This channelID need
// OPPO must set a ChannelID to receive push messages. This channelID needs to be t
pushInfo.androidOPPOChannelID = "tuikit"
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HM
pushInfo.androidHuaWeiCategory = "IM"
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo
TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: p

} fail: {
    code, message in
}
```

2. Continuous Vibration

Notification Service Extension (abbreviated as Extension) is an Extension in iOS used to intercept and modify push notification content. With the aid of the Extension, long-term vibration can be achieved.

The primary application and Extension are two independent processes. After receiving an APNs push, the primary application will not be activated, while the Extension will be activated. Turn on continuous vibration in the Extension. When the application switches to the foreground, inform the Extension via inter-process communication to turn off the vibration and ringtone.

2.1 Enable Mutable Content

Use `Notification Service Extension` to achieve continuous vibration. First, enable the `mutable-content` switch in the console.

2.2 Create an Extension Project

In the Xcode project, select `Editor > Add Target > Notification Service Extension` to create an Extension. The newly created Extension Target needs to configure the Apple certificate. For specific methods, you can

[refer here.](#)

The newly created Extension contains a source code file named **NotificationService**, with the following content:

```
// NotificationService.swift
import UserNotifications

class NotificationService: UNNotificationServiceExtension {
    var contentHandler: ((UNNotificationContent) -> Void)?
    var bestAttemptContent: UNMutableNotificationContent?

    override func didReceive(_ request: UNNotificationRequest, withContentHandler contentHandler: UNNotificationContentHandler?) {
        self.contentHandler = contentHandler
        bestAttemptContent = (request.content.mutableCopy() as? UNMutableNotificationContent)
        if let bestAttemptContent = bestAttemptContent {
            // Modify the notification content here...
            bestAttemptContent.title = "\\(bestAttemptContent.title) [modified]"
            contentHandler(bestAttemptContent)
        }
    }

    override func serviceExtensionTimeWillExpire() {
        // Called just before the extension will be terminated by the system.
        // Use this as an opportunity to deliver your "best attempt" at modified content.
        if let contentHandler = contentHandler, let bestAttemptContent = bestAttemptContent {
            contentHandler(bestAttemptContent)
        }
    }
}
```

didReceive: When a push notification is received and the Extension is activated, the didReceive callback is executed. Users can complete the modification of push notification information in didReceive. After completing the modification, call the contentHandler() method. The Extension execution ends.

serviceExtensionTimeWillExpire: When a push notification is received and the Extension is activated, if the Extension does not respond (does not execute the contentHandler() method) for more than 30 seconds, the serviceExtensionTimeWillExpire callback method will be called, and the contentHandler() method will be called within this method. The Extension execution ends.

2.3 Vibrate When a Push Notification Is Received

1. When a push notification is received and the Extension is activated, turn on vibration when the didReceive callback is executed.
2. Turn off vibration and ringtone after receiving serviceExtensionTimeWillExpire.

The modified code is as follows:

```
// NotificationService.swift
```

```
import UserNotifications

class NotificationService: UNNotificationServiceExtension {
    var contentHandler: ((UNNotificationContent) -> Void)?
    var bestAttemptContent: UNMutableNotificationContent?

    override func didReceive(_ request: UNNotificationRequest, withContentHandler c
        self.contentHandler = contentHandler
        bestAttemptContent = (request.content.mutableCopy() as? UNMutableNotificati
        if let bestAttemptContent = bestAttemptContent {
            contentHandler(bestAttemptContent)
        }

        // start ringing, VibratorFeature.start() needs to be implemented by yourse
        VibratorFeature.start()
    }

    override func serviceExtensionTimeWillExpire() {
        if let contentHandler = contentHandler, let bestAttemptContent = bestAttem
            /// If there is no response for more than 30 seconds, the vibration and
            // stop ringing, RingingFeature.shared.stop() needs to be implemented b
            VibratorFeature.stop()
            contentHandler(bestAttemptContent)
        }
    }
}
```

Note:

The methods of starting vibration `VibratorFeature.start()` and stopping vibration

`VibratorFeature.stop()` used in the above code need to be implemented by the customer.

2.4 Turn Off Vibration in Main Application

Upon receiving a push, start vibration and ringing. When the user enters the APP, the vibration in the Extension should be turned off. The primary application and the Extension are two independent processes.

`CFNotificationCenterGetDarwinNotifyCenter` can be used to send notifications. When the APP enters the foreground, the primary application sends a notification to the Extension. The Extension listens for the notification and turns off the vibration upon receipt.

The modified code is as follows:

```
// Versions prior to iOS 13 monitor the foreground or background status through App
// AppDelegate.swift
class AppDelegate: UIResponder, UIApplicationDelegate {
    func applicationWillEnterForeground(_ application: UIApplication) {
        ...
        sendStopRingingToExtension()
    }
}
```

```
}
func sendStopRingingToExtension() {
    CFNotificationCenterPostNotification(CFNotificationCenterGetDarwinNotifyCen
        CFNotificationName("APNsStopRinging" a
    }
}

// Starting from iOS 13, monitor the foreground or background status through Scened
// SceneDelegate.swift
class SceneDelegate: UIResponder, UIWindowSceneDelegate {
    func sceneWillEnterForeground(_ scene: UIScene) {
        ...
        sendStopRingingToExtension()
    }

    func sendStopRingingToExtension() {
        CFNotificationCenterPostNotification(CFNotificationCenterGetDarwinNotifyCen
            CFNotificationName("APNsStopRinging" a
        }
    }
}

// NotificationService.swift
import UserNotifications

class NotificationService: UNNotificationServiceExtension {
    var contentHandler: ((UNNotificationContent) -> Void)?
    var bestAttemptContent: UNMutableNotificationContent?

    override func didReceive(_ request: UNNotificationRequest, withContentHandler c
        self.contentHandler = contentHandler
        bestAttemptContent = (request.content.mutableCopy() as? UNMutableNotificati
        if let bestAttemptContent = bestAttemptContent {
            contentHandler(bestAttemptContent)
        }

        // start ringing, VibratorFeature.start() needs to be implemented by yourse
        VibratorFeature.start()
        registerObserver()
    }

    override func serviceExtensionTimeWillExpire() {
        if let contentHandler = contentHandler, let bestAttemptContent = bestAttem
            /// If there is no response for more than 30 seconds, the vibration and
            // stop ringing, RingingFeature.shared.stop() needs to be implemented b
            VibratorFeature.stop()
        }
    }
}
```

```
        completionHandler(bestAttemptContent)
    }
}

private func registerObserver() {
    CFNotificationCenterAddObserver(CFNotificationCenterGetDarwinNotifyCenter(),
                                    Unmanaged.passUnretained(self).toOpaque(),
                                    VibratorFeature.stop(),
                                    }, "APNsStopRinging" as CFString, nil, .deliverImmediately)
}
}
```

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:

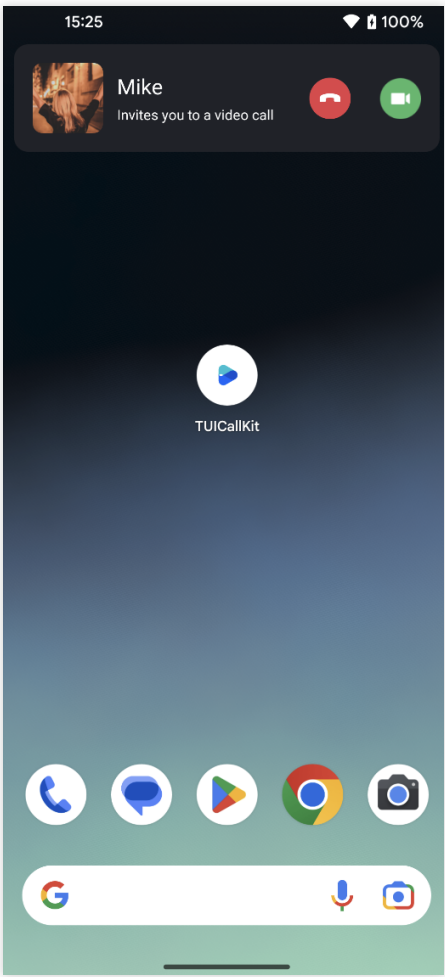
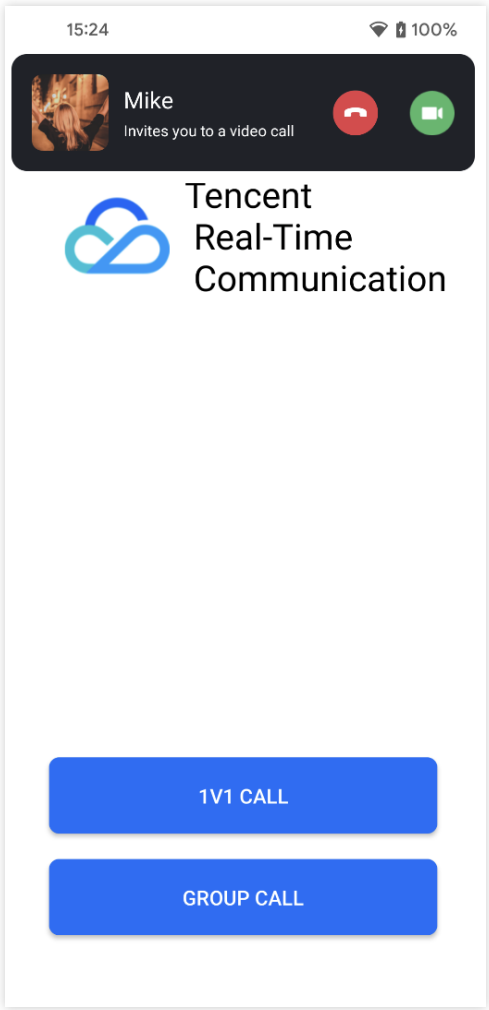
Phones with Google Mobile Services.
The FCM data message feature is only available in TIMPush **7.9.5668** and later versions.
Support for TUICallKit is limited to **2.3** and later versions.

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



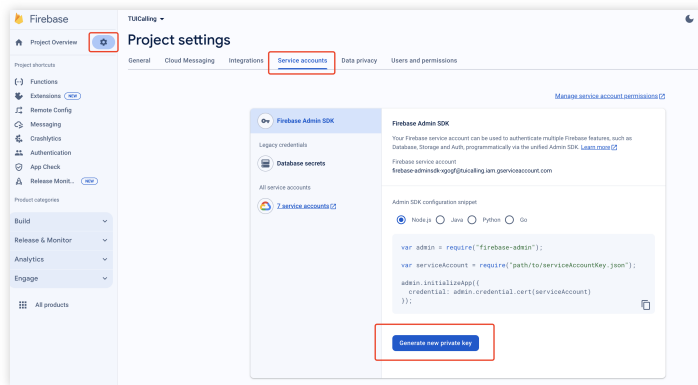
Note:

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



Add FCM certificate

Adding Method: ☒ Upload certificate

Message type: ☐ Notification message ☒ Transparent transmission

Transparent transmission (data) messages can be available after activating [Push plug-in](#). It only supports SDK enhanced version v7.8 and above.

Package Name: [How to Generate](#)

Upload certificate: [Select file](#)

[How to Generate an FCM certificate](#)

ChannelID:

[Confirm](#)

Quick Integration

1. Download and add the configuration file

After completing the push information in the console, download the `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 <code>timpush-configs.json</code>	Download the file <code>google-services.json</code>	Add to

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:fcml: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. Complete 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).

```
5 TIMPushFCMMsgService Message data payload: {IMDesc=You have a new call, IMSound=phone_ringing, IMTitle=258, ext={"timPushFeatures":{"fcmNotificationType":1,"fcmPushType":0},"entity":{"action":1,"type":1}}
```

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
<div><div>App info</div><div><div> TUICallKit Version 1.0</div><div><div>Notifications</div><div>~0 notifications per week</div><div>→</div></div><div><div>Permissions</div><div>No permissions granted</div><div>→</div></div><div><div>Storage</div><div>52.80 MB used in internal storage</div><div>→</div></div><div><div>Data usage</div><div>No data used</div><div>→</div></div><div><div>Battery</div><div>No battery use since last full charge</div><div>→</div></div><div><div>Open by default</div><div>No defaults set</div><div>→</div></div><div><div>Advanced</div><div><div>Display over other apps</div><div>Allowed</div><div>→</div></div><div><div>Uninstall</div><div>Force stop</div></div></div></div></div>	<div><div>TUICallKit</div><div><div>Device management</div><div>Autostart</div><div>Activated in the background</div><div>Associated startup</div><div>Floating window</div><div>Home screen shortcuts</div><div>Ask every ti</div><div>Display on lock screen</div><div>Background pop-ups</div><div>Camera</div><div>Use camera</div><div>While using the a</div><div>Microphone</div><div>Record</div><div>While using the a</div></div></div>

Suggestions and Feedback

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

Flutter

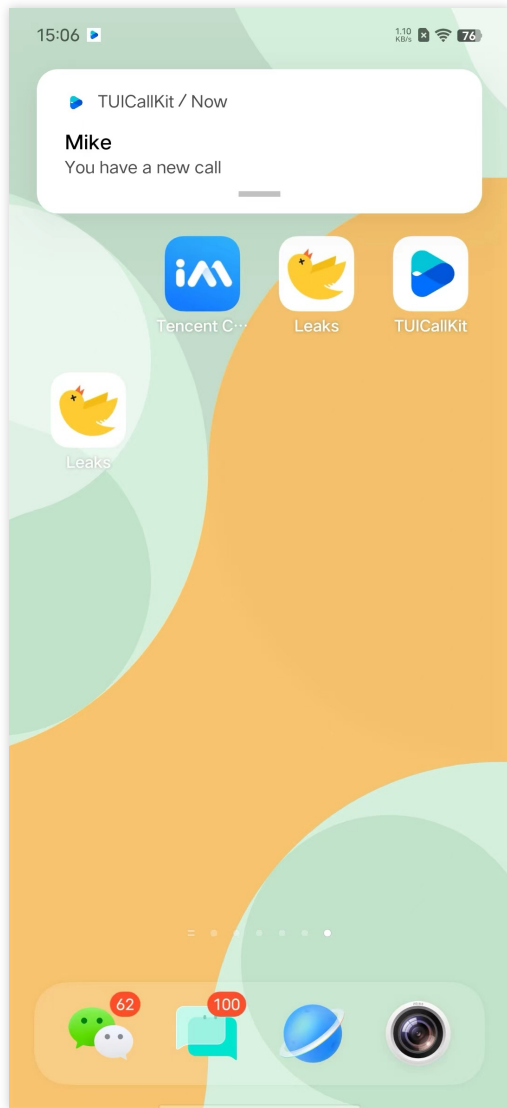
Notification

Last updated : 2025-02-27 16:09:56

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



Integration Guide

Please go to [TIMPush Integration Method](#).

Making an Offline Push Call

If you want to make an offline push call, you need to set `OfflinePushInfo` when calling [calls](#).

```
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.APNs;

TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);

TUICallKit.instance.calls(callUserIdList, TUICallMediaType.audio, params);
```

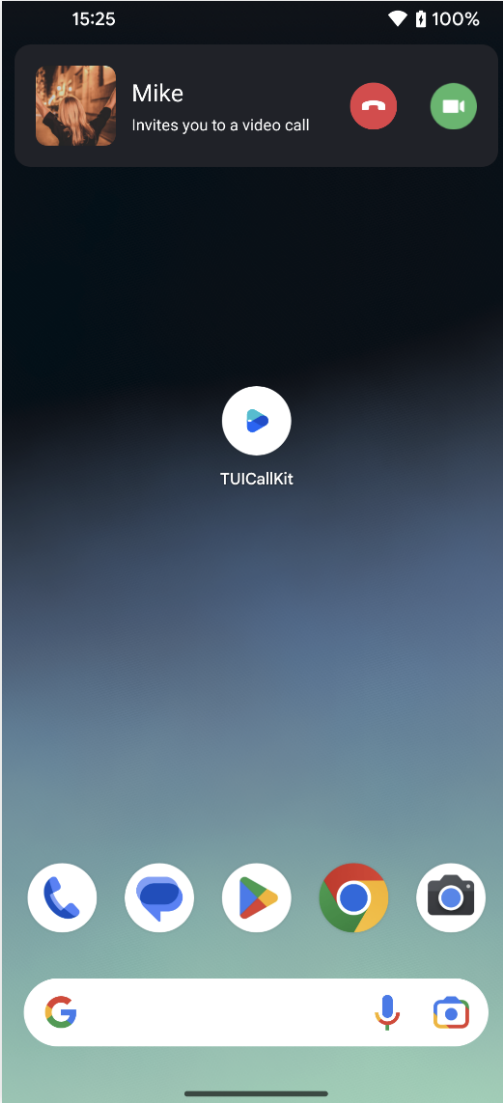
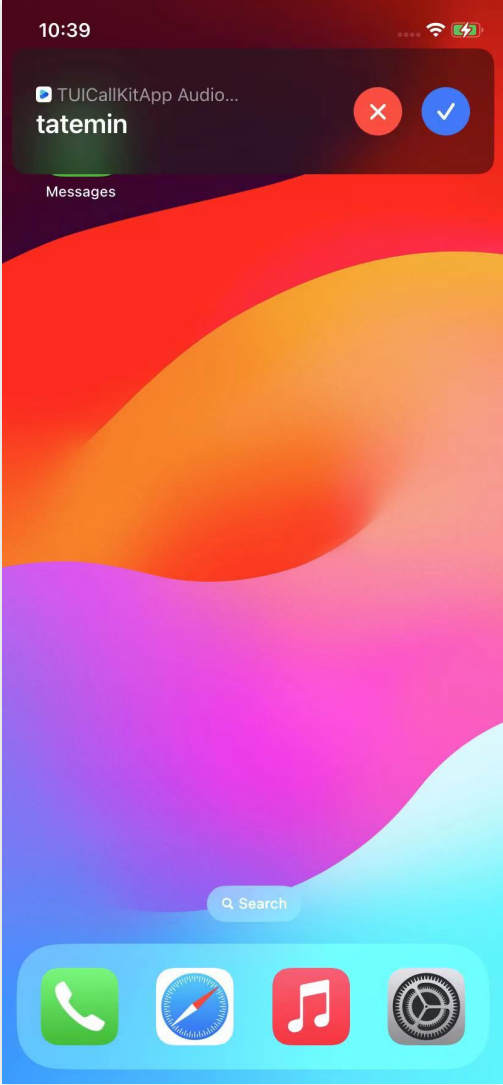
Note:

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 : 2025-02-27 16:09:56

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.

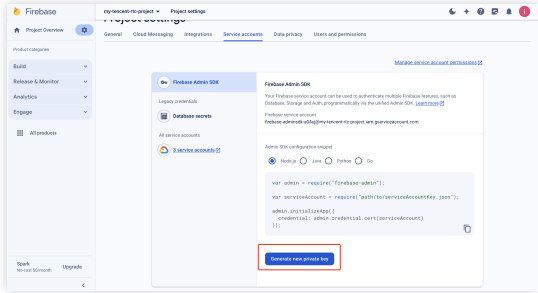
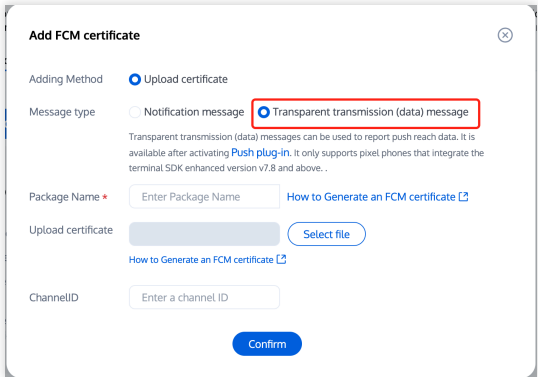
Integrated message push plugin

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

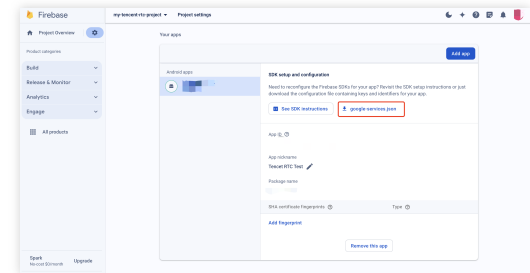
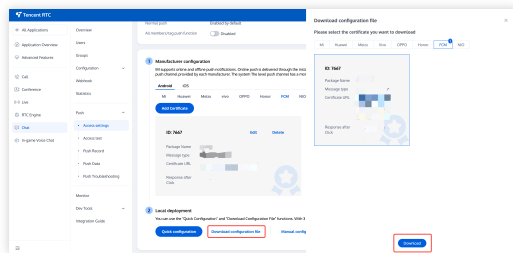
1. Download and add the configuration file

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-

Download the file google-services.json

configs.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:

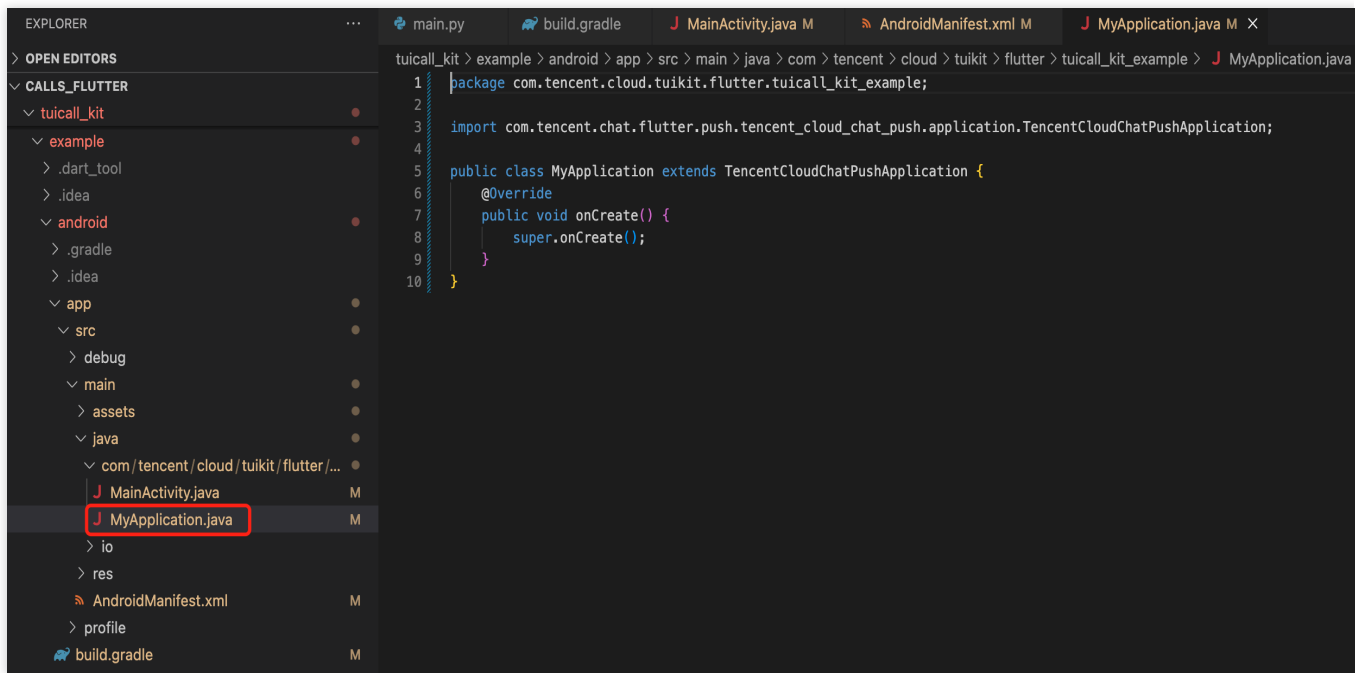
TIMPush requires integration with the Chat SDK version **7.9.5666** or above.

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

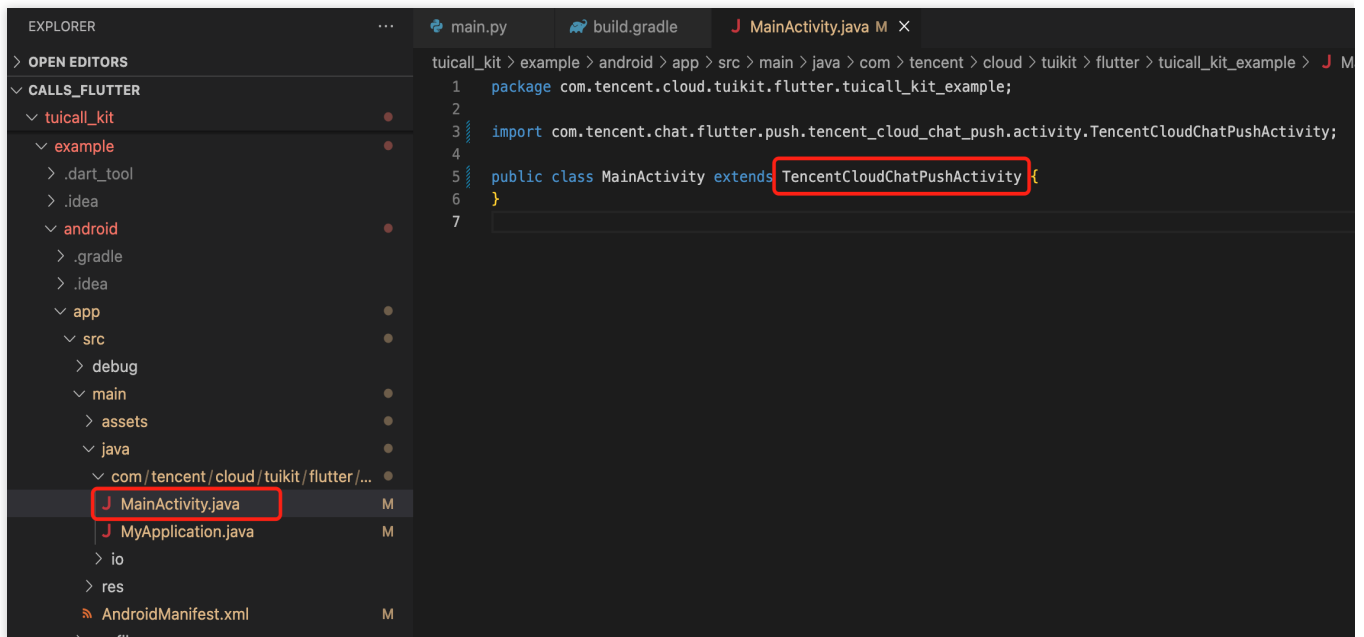
import com.tencent.chat.flutter.push.tencent_cloud_chat_push.application.TencentClo

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

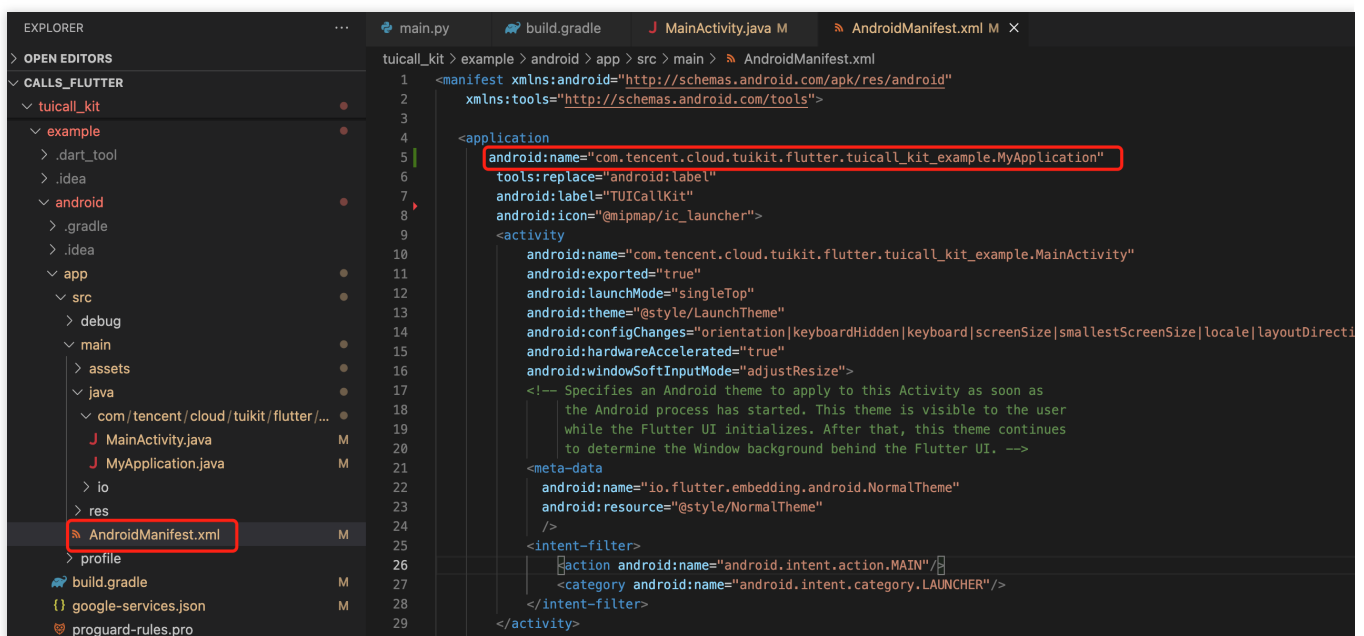
Note:

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 [calls](#).

```

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.calls(callUserIdList, 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);

```
5 TIMPushFCMMsgService ... Message data payload: {IMDesc=You have a new call, IMSound=phone_ringing, INTitle=258, ext={"timPushFeatures":{"fcmNotificationType":1,"fcmPushType":0},"entity":{"actio
```

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)

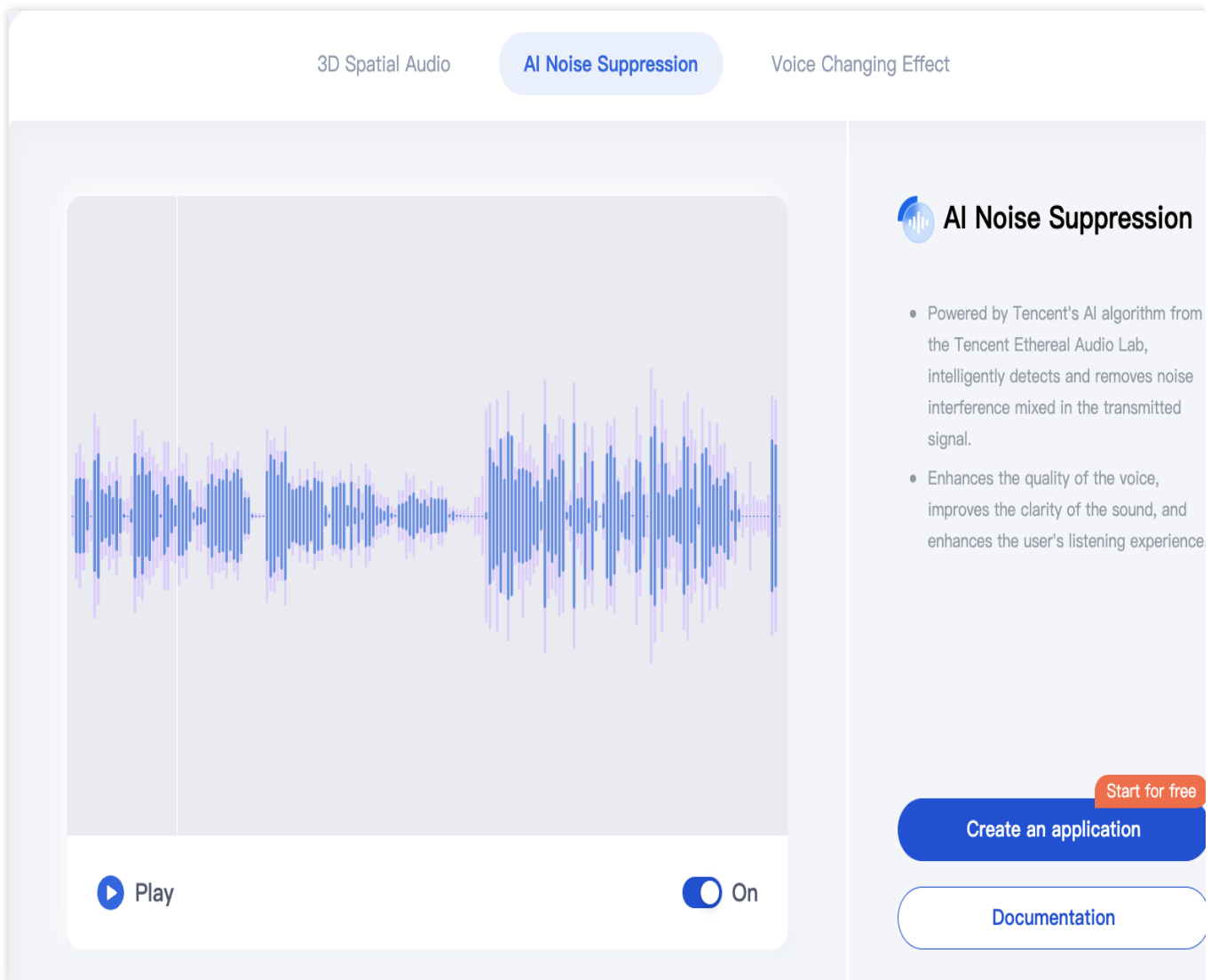
Last updated : 2024-11-22 16:53:51

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.

Try It Online

You can also enter our [Tencent RTC Demo](#) to try the excellent sound effects brought by AI Noise Suppression 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

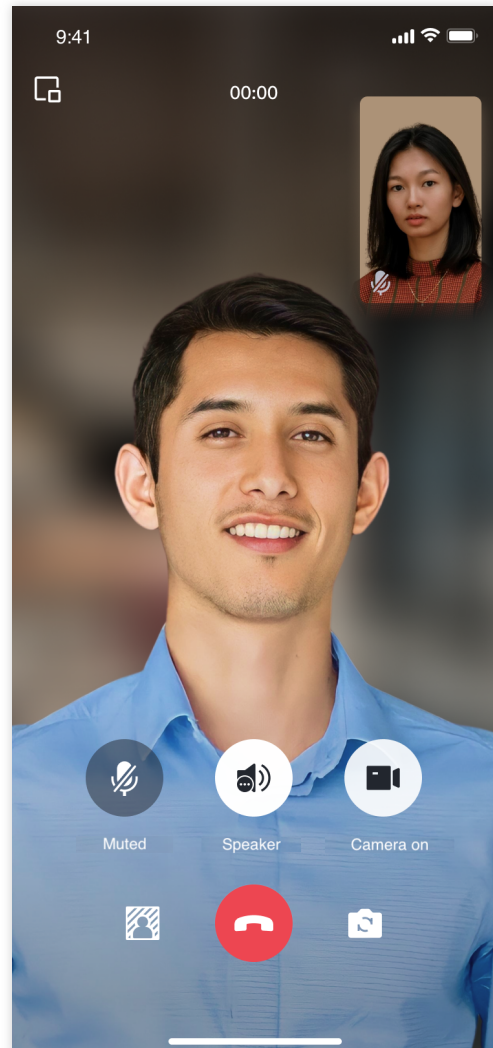
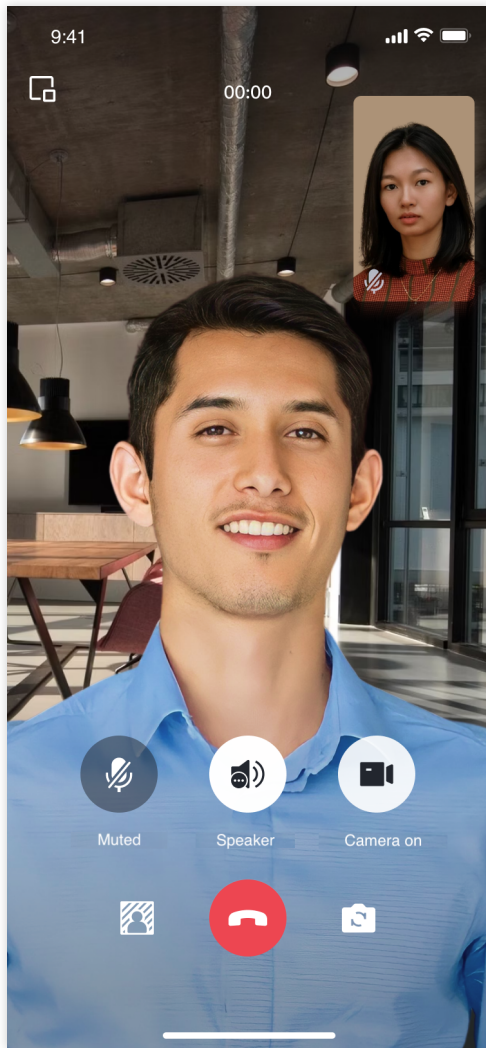
Last updated : 2024-07-22 17:00:45

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

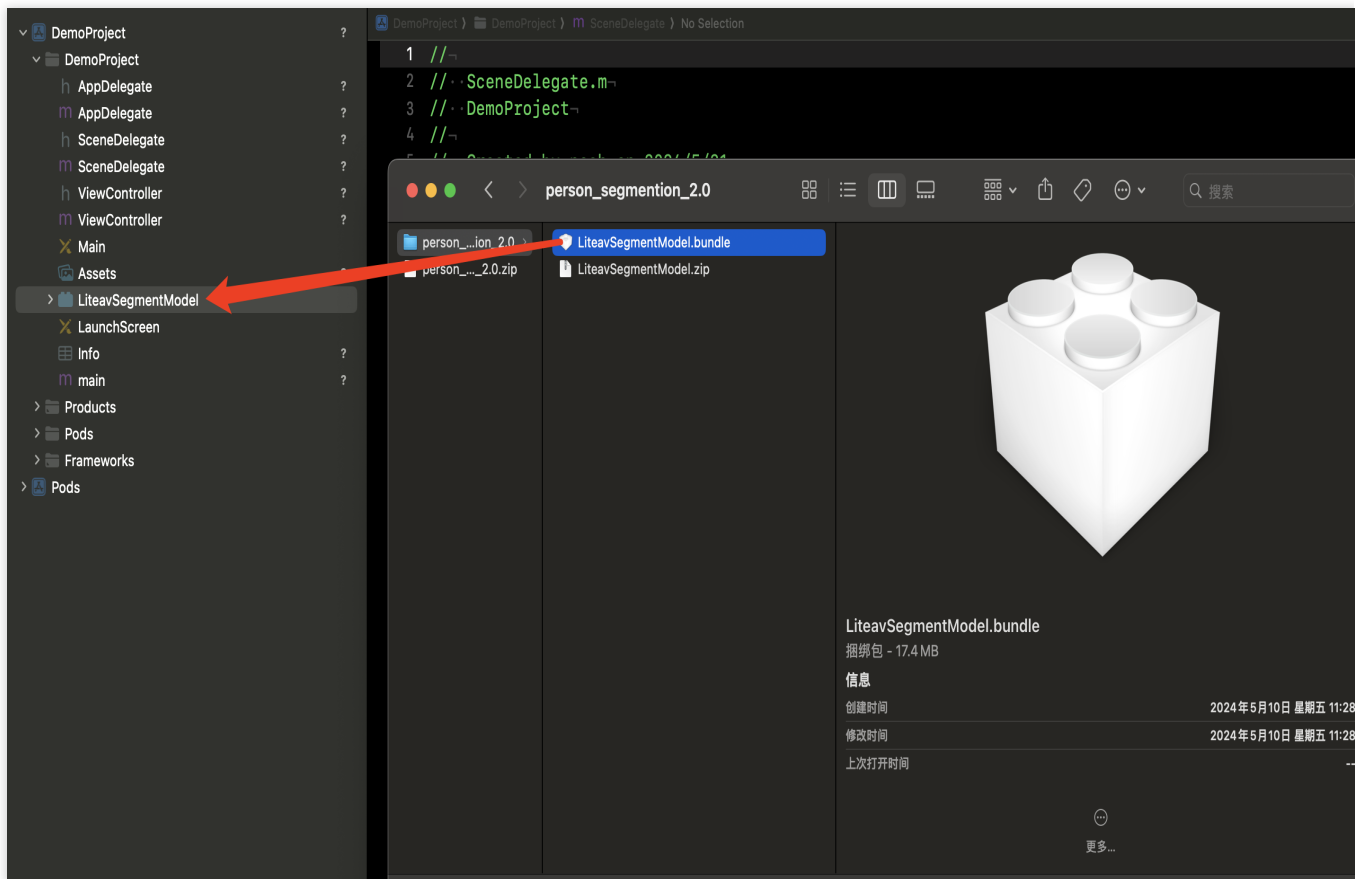
The display effect of the TUICallKit component after integrating the virtual background feature is as follows:

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

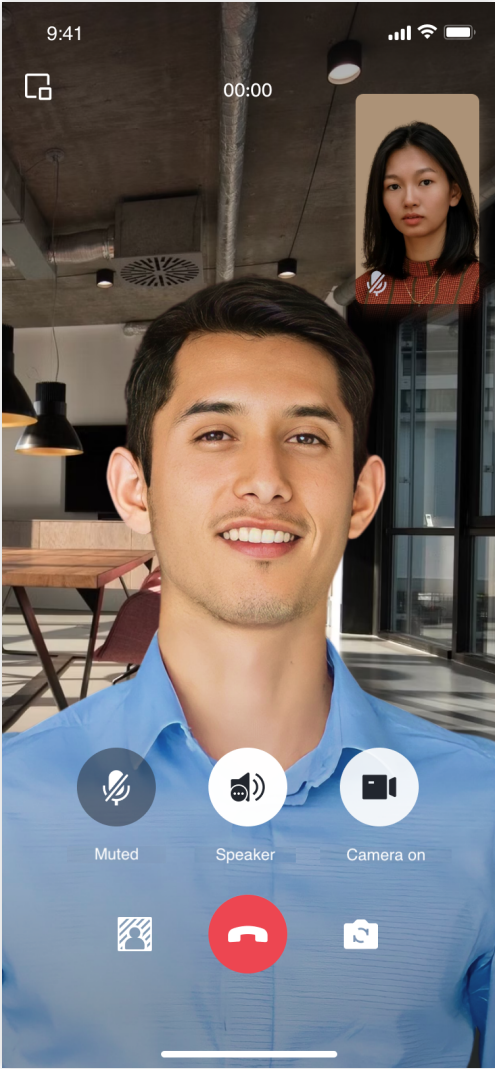
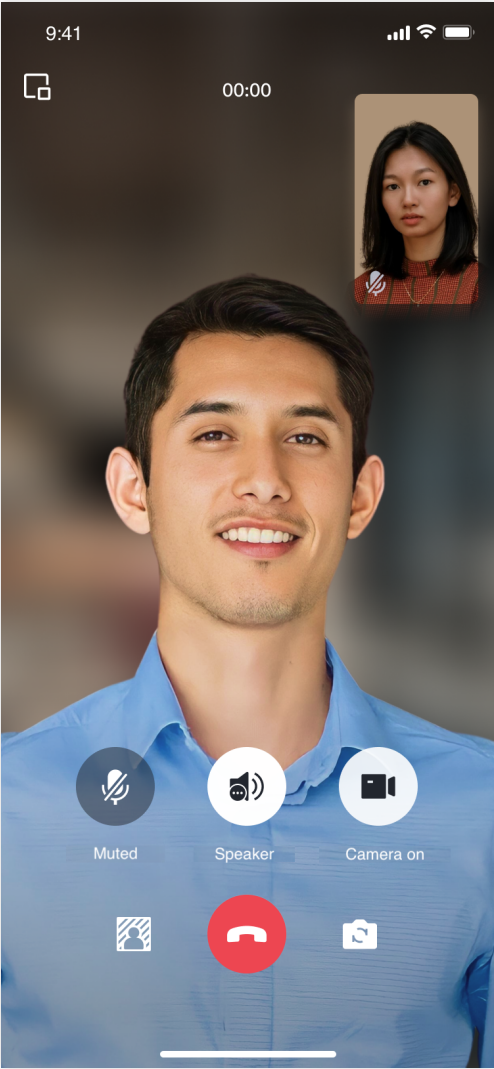
Android

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.

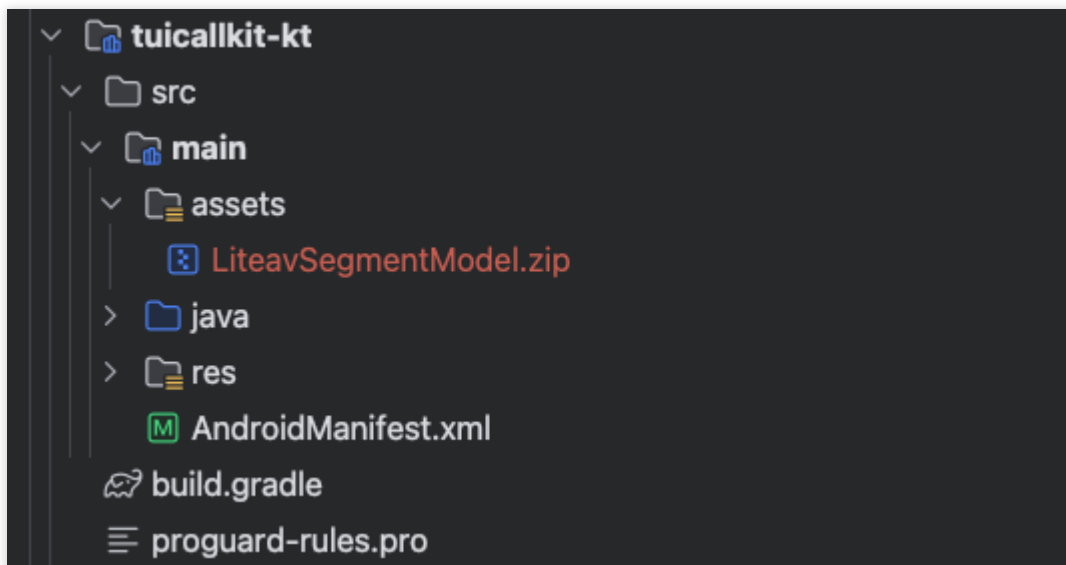
Integration effect

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	segmentation_1.0
com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176	segmentation_2.0

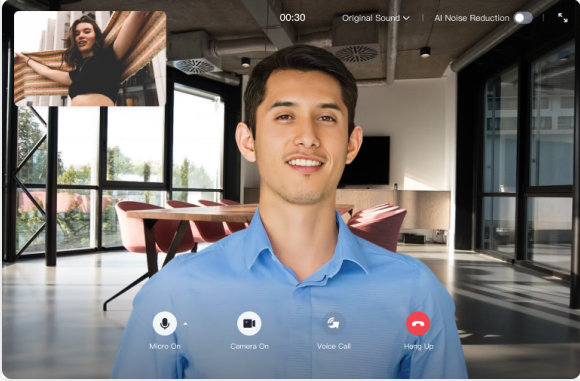

Web

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.

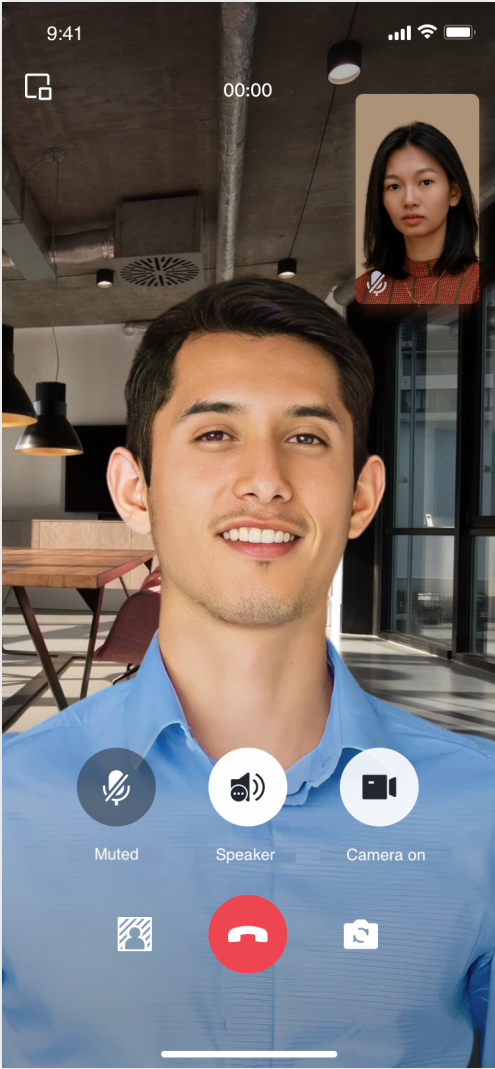
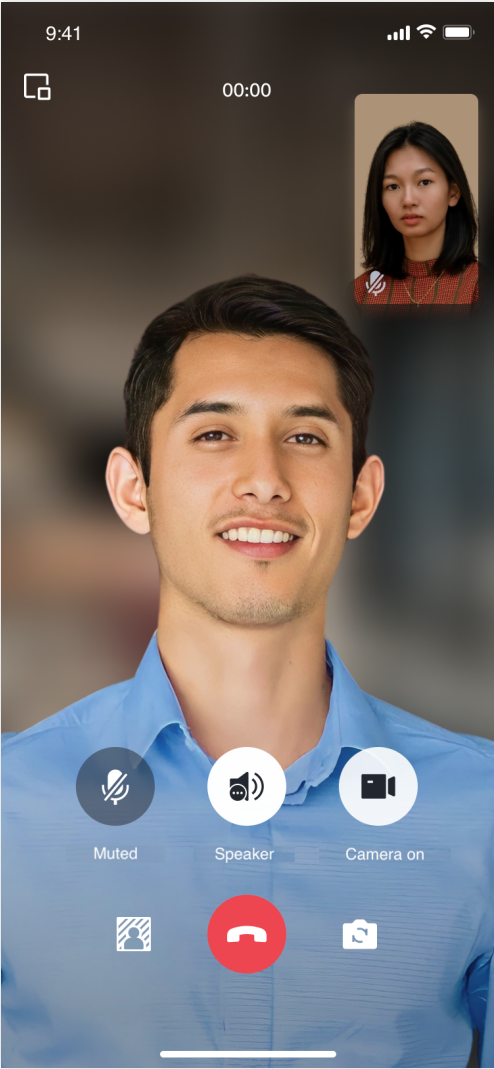
Flutter

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.

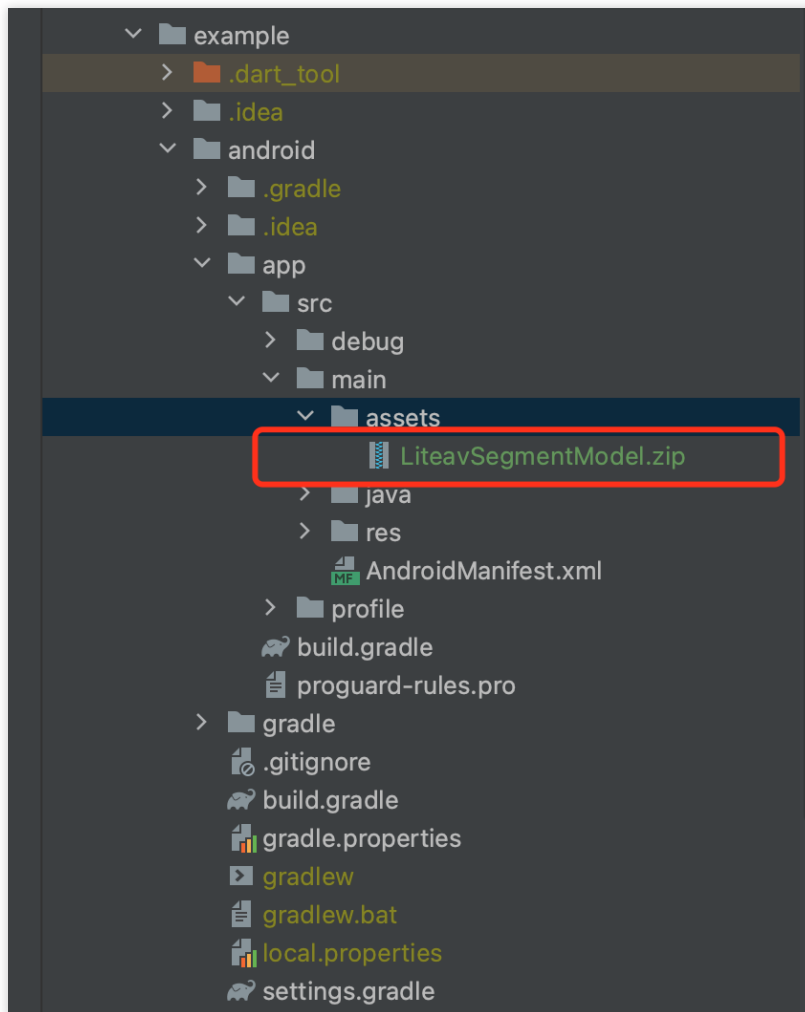
```
pod 'TXLiteAVSDK_Professional', '11.8.15669'
```

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

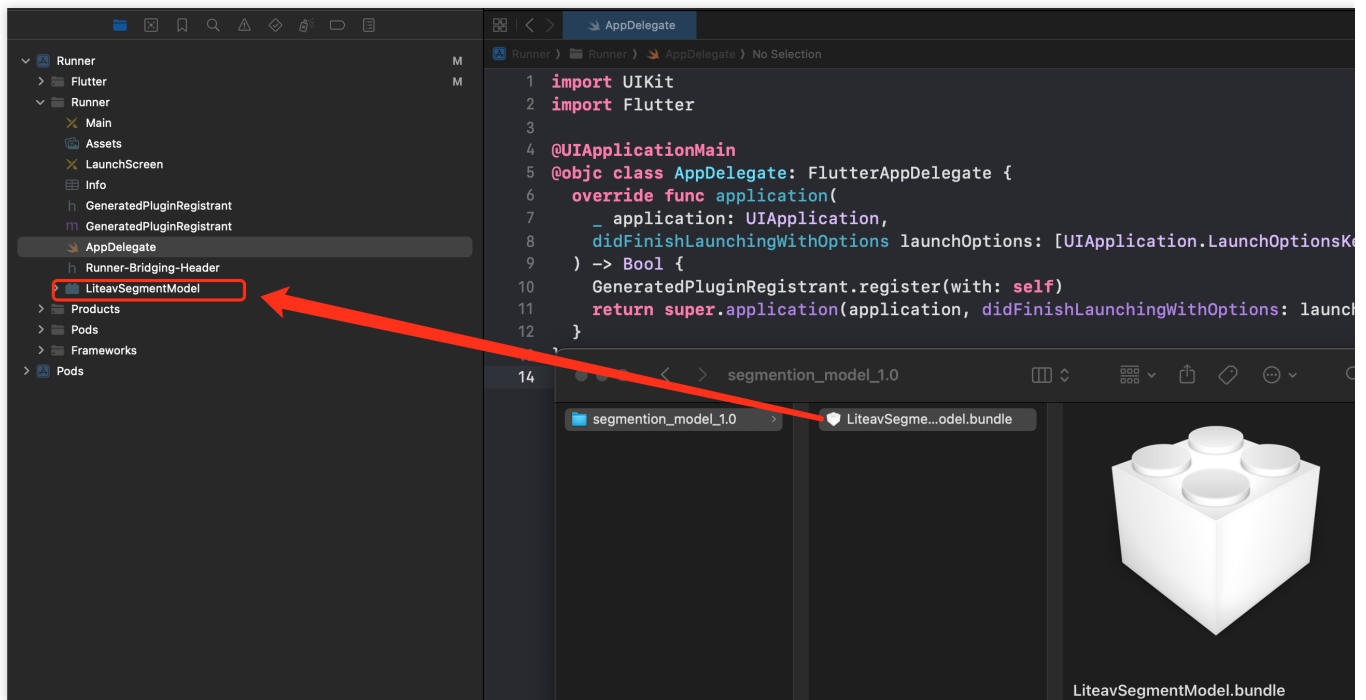
Android

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	segmentation_1.0
com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176	segmentation_2.0

Conversational Chat (TUICallKit)

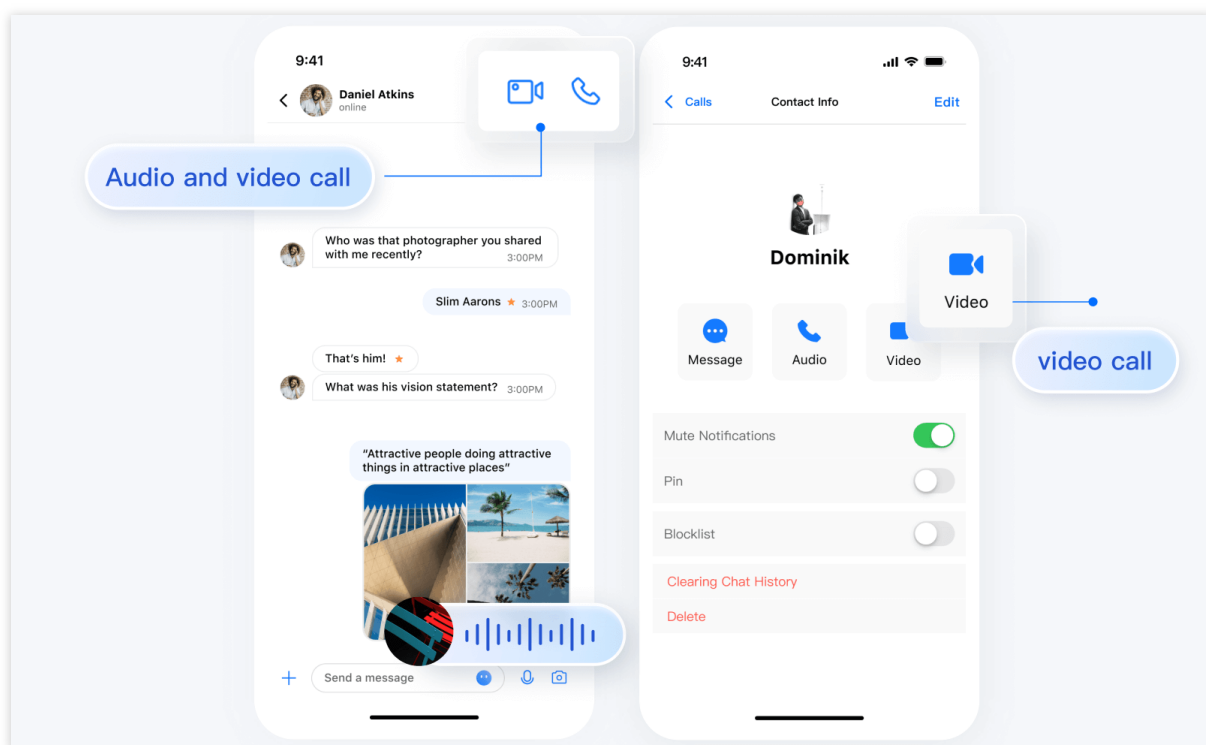
Android

Last updated : 2025-02-18 19:44:36

Chat provides multi-platform chat APIs, user interface components, and server-side APIs, enabling you to build a fully functional chat application in ten minutes. In the chat application, you can easily add audio and video call features with just up to three lines of code.

Integration demonstration

After integrating Chat and TUICallKit, you can easily initiate audio and video calls on the chat messages page and user information page.



Quick Integration

Preparations

Please ensure your application has [integrated Chat](#).

Activate Service

1. Log in to the [chat console](#) to activate audio and video services. For detailed steps, refer to the documentation [Activate Service](#).
2. In the pop-up dialog box for activating the TRTC service, click "Confirm". The system will create a TRTC application with the same SDKAppID as the current chat application. The [TRTC console](#) allows reuse of account and authentication.

Integrate TUICallKit.

Add TUICallKit dependency to the build.gradle file in the App:

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

After integrating the TUICallKit component, the chat interface and contact information interface will by default display "video call" and "voice call" buttons. When the user clicks the button, TUICallKit will automatically display the call invitation UI and send a call invitation request to the other party.

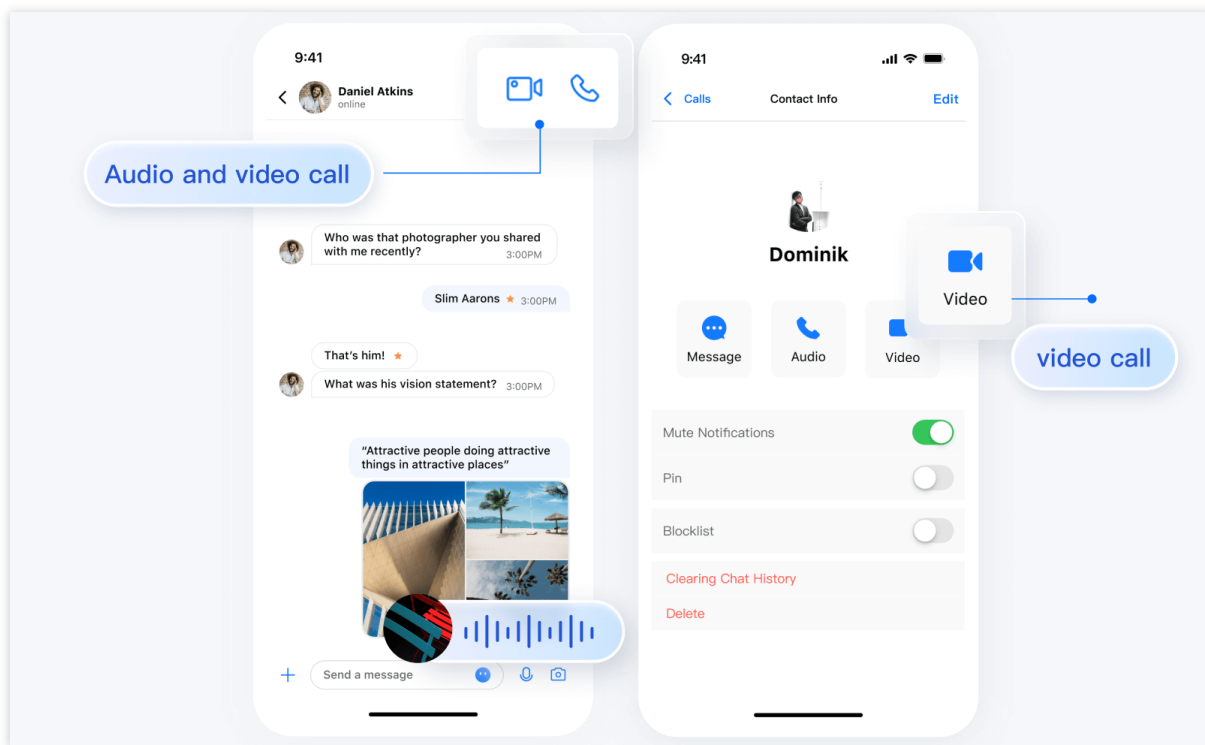
iOS

Last updated : 2024-12-11 09:23:34

Chat provides multi-platform chat APIs, user interface components, and server-side APIs, enabling you to build a fully functional chat application in ten minutes. In the chat application, you can easily add audio and video call features with just up to three lines of code.

Integration demonstration

After integrating Chat and TUICallKit, you can easily initiate audio and video calls on the chat messages page and user information page.



Quick Integration

Preparations

Please ensure your application has [integrated Chat](#).

Activate Service

1. Log in to the [chat console](#) to activate audio and video services. For detailed steps, refer to the documentation [Activate Service](#).

2. In the pop-up dialog box for activating the TRTC service, click "Confirm". The system will create a TRTC application with the same SDKAppID as the current chat application. The [TRTC console](#) allows reuse of account and authentication.

Integrate TUICallKit.

Add the following content to your Podfile:

```
// Integrate the TUICallKit component
pod 'TUICallKit'
```

After integrating the TUICallKit component, the chat interface and contact information interface will by default display "video call" and "voice call" buttons. When the user clicks the button, TUICallKit will automatically display the call invitation UI and send a call invitation request to the other party.

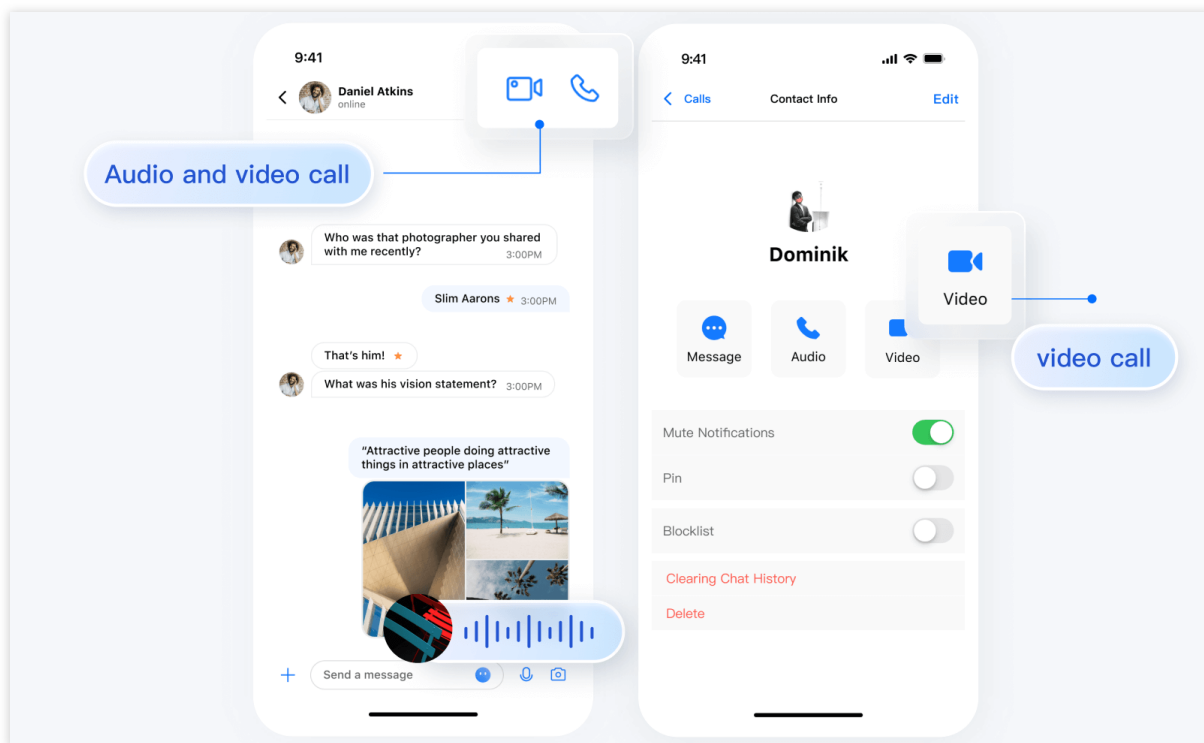
Flutter

Last updated : 2024-12-11 09:27:39

Chat provides multi-platform chat APIs, user interface components, and server-side APIs, enabling you to build a fully functional chat application in ten minutes. In the chat application, you can easily add audio and video call features with just up to three lines of code.

Integration demonstration

After integrating Chat and TUICallKit, you can easily initiate audio and video calls on the chat messages page and user information page.



Quick Integration

Preparations

Please ensure your application has [integrated Chat](#).

Activate Service

1. Log in to the [chat console](#) to activate audio and video services. For detailed steps, refer to the documentation [Activate Service](#).

2. In the pop-up dialog box for activating the TRTC service, click "Confirm". The system will create a TRTC application with the same SDKAppID as the current chat application. The [TRTC console](#) allows reuse of account and authentication.

Integrate TUICallKit.

1. In the console, navigate to your Flutter project's directory and run the following command to install the [tencent_calls_uikit](#) plugin:

```
flutter pub add tencent_calls_uikit
```

2. 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});

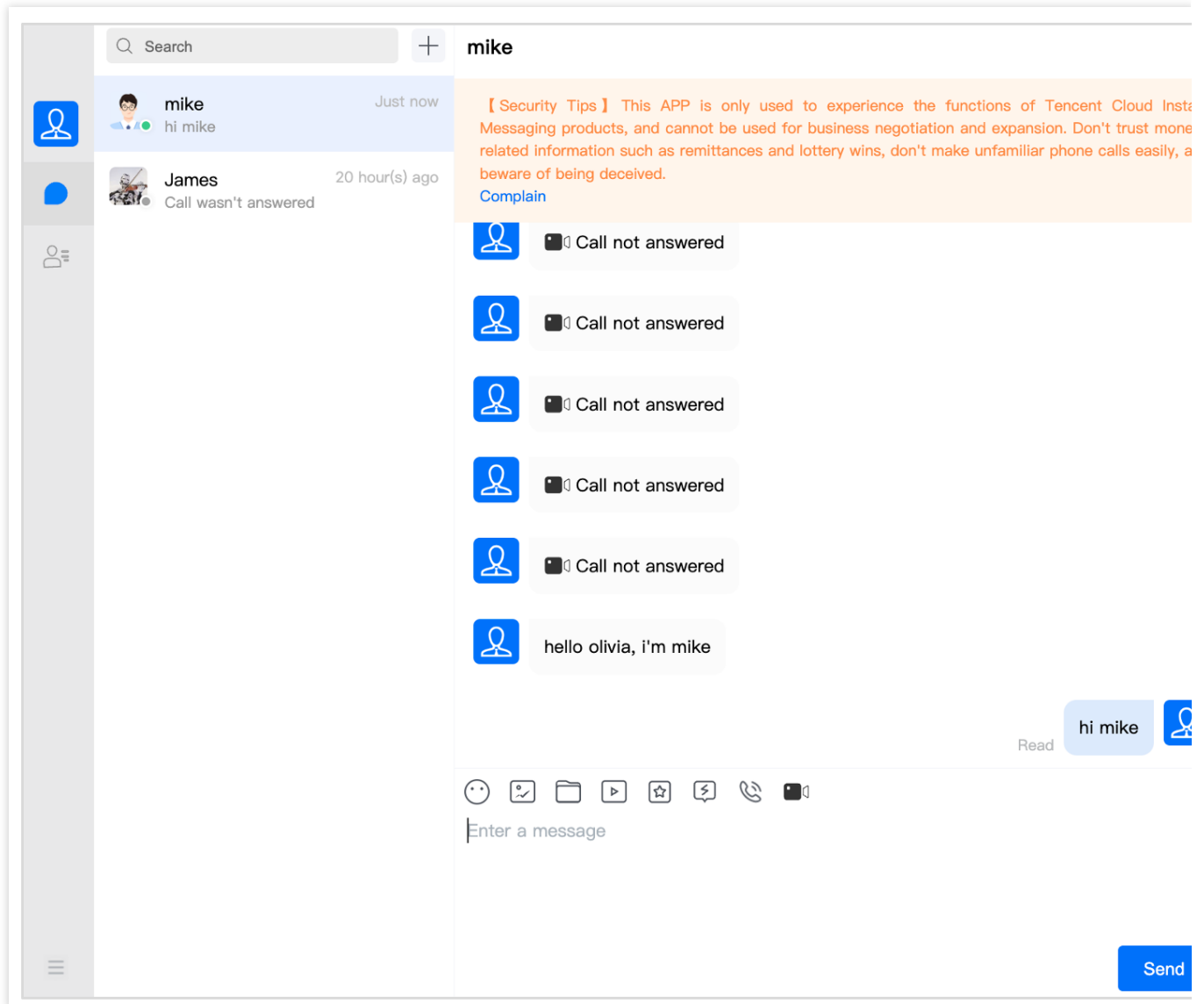
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      navigatorObservers: [TUICallKit.navigatorObserver],
      .....
    );
  }
}
```

After integrating the `TUICallKit` component, the chat interface and contact information interface will by default display "video call" and "voice call" buttons. When the user clicks the button, `TUICallKit` will automatically display the call invitation UI and send a call invitation request to the other party.

Vue

Last updated : 2025-01-16 15:15:37

By integrating Chat and CallKit, you can achieve the following effects. Click [Try It Online](#).



Environment Requirements

TypeScript

Sass (The version of sass-loader should be $\leq 10.1.1$)

node (node.js $\geq 16.0.0$)

npm (use a version that matches the Node version in use)

Step 1: Integrating Chat

For detailed steps, please refer to: [Quick Integration of Chat](#).

Step 2: Activating the Audio and Video Call Capability

Before using the audio and video services provided by Tencent Cloud, you need to go to the console to activate the service for the application. Refer to the specific steps in [Activate the Service](#).

Step 3: Download the TUICallKit Component

Download the TUICallKit component via [npm](#), version 3.3.6 and above:

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

Step 4: Introducing and Invoking the TUICallKit Component

As follows: Import `<TUICallKit />` where `<TUIKit />` is imported.

```
<template>
  <div id="app">
    <TUIKit :SDKAppID="YOUR_SDKAPPID" userID="YOUR_USERID" userSig="YOUR_USERSIG" /
    <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullScre
  </div>
</template>
<script lang="ts" setup>
import { TUIKit } from './TUIKit';
import { TUICallKit } from '@tencentcloud/call-uikit-vue';
</script>
<style lang="scss">
</style>
```

Step 5: Launching the Project

vite

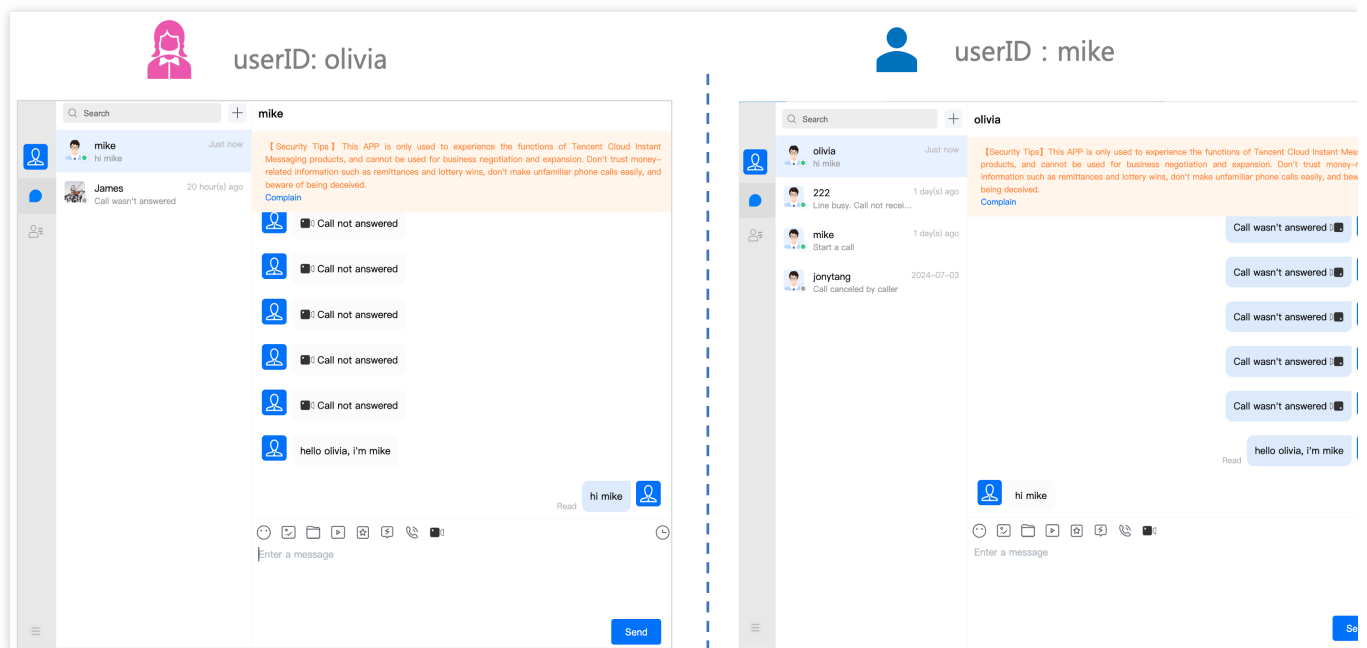
webpack

```
npm run dev
```

```
npm run serve
```

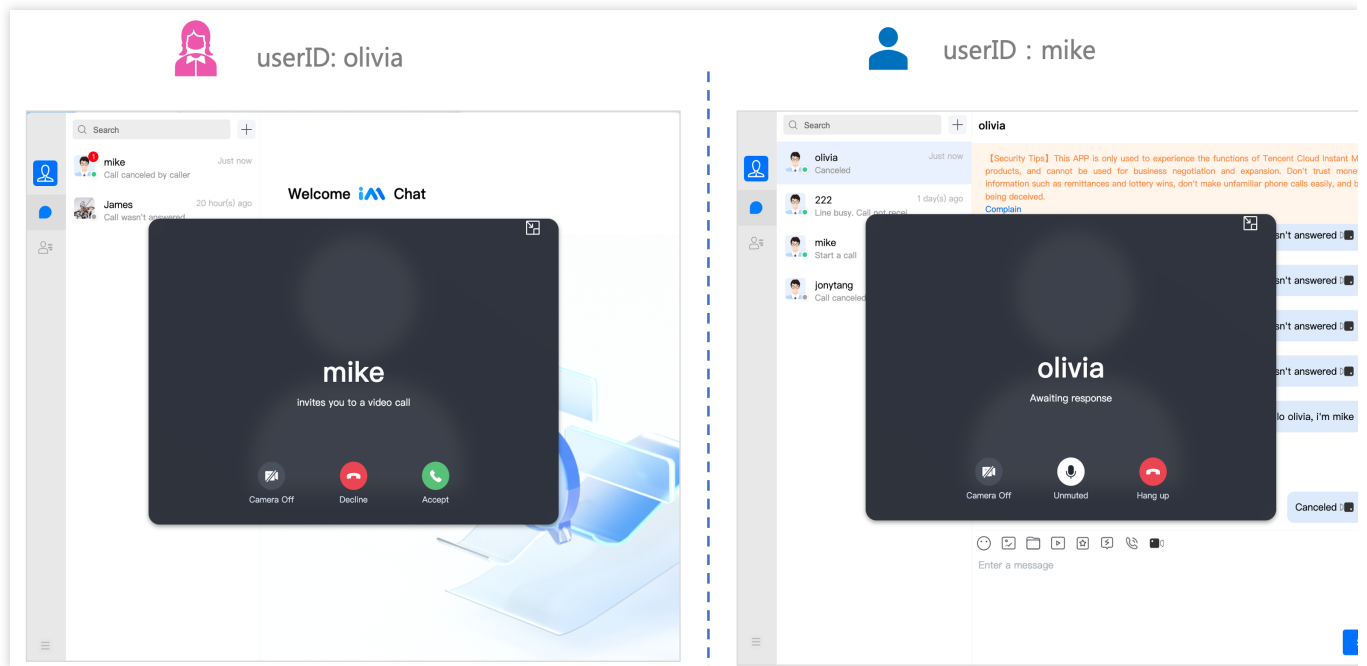
Step 6: Sending Your First Message

As shown in the figure, you can send messages to and from your friends.



Step 7. Make your first call

As shown in the figure, click the video/audio icon in the toolbar to realize the call effect.



FAQs

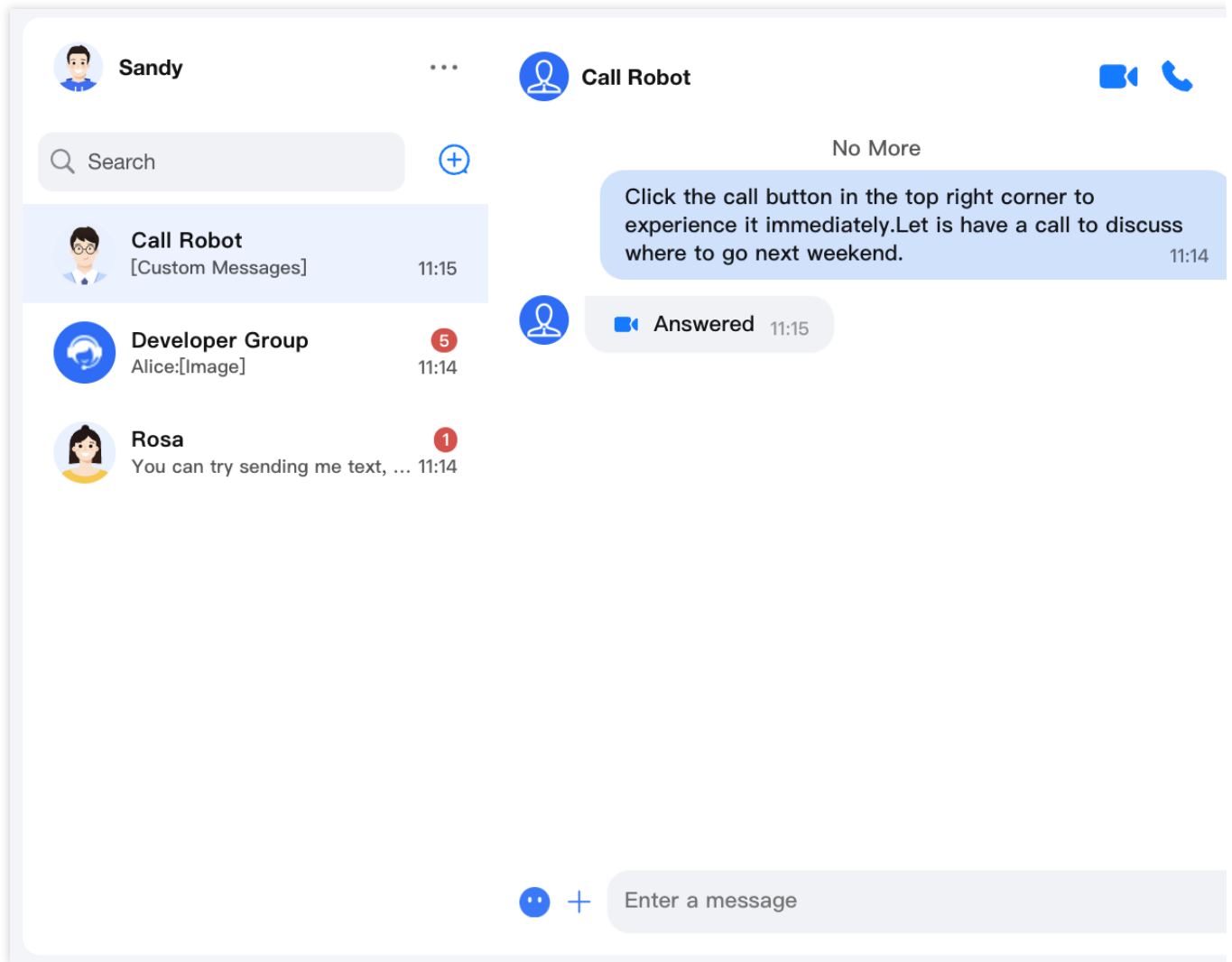
If you encounter any problems with access and use, please refer to [FAQs](#).

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

React

Last updated : 2025-01-16 15:07:42

By integrating Chat and CallKit, you can achieve the following effects. Click [Trying It Online](#).



Environment Requirements

React version 18+ (17.x versions are not supported)

TypeScript

[Node.js](#) version 16+

npm (use a version that matches the Node version in use)

Step 1: Integrating Chat

For detailed steps, please refer to: [Quick Integration of Chat](#).

Step 2: Activating the Audio and Video Call Capability

Before using the audio and video services provided by Tencent Cloud, you need to go to the console to activate the service for the application. Refer to the specific steps in [Activate the Service](#).

Step 3: Download the TUICallKit Component

Download the TUICallKit component via [npm](#), version 3.3.6 and above:

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

Step 4: Introducing and Invoking the TUICallKit Component

Just add two lines of code to experience the call feature. Introduce TUICallKit and invoke the TUICallKit component. Enable Call in the `<ChatHeader />` component, display the Icon, and set `enableCall` to true.

Note:

1. If you have not integrated TUIKit yet, please follow the [Integrate TUIKit](#) documentation to integrate TUIKit first. TUIKit version 2.2.8 and above.
2. The TUICallKit component can add style to display and control the position, width, height, and other styles of TUICallKit.

```
// 1. Import TUICallKit
import { TUICallKit } from '@tencentcloud/call-uikit-react';

// 2. If you are displaying on a PC, please add style to initialize the position of
const callStyle: React.CSSProperties = {
  position: 'fixed',
  top: '50%',
  left: '50%',
  zIndex: '999',
  transform: 'translate(-50%, -50%)',
};
```

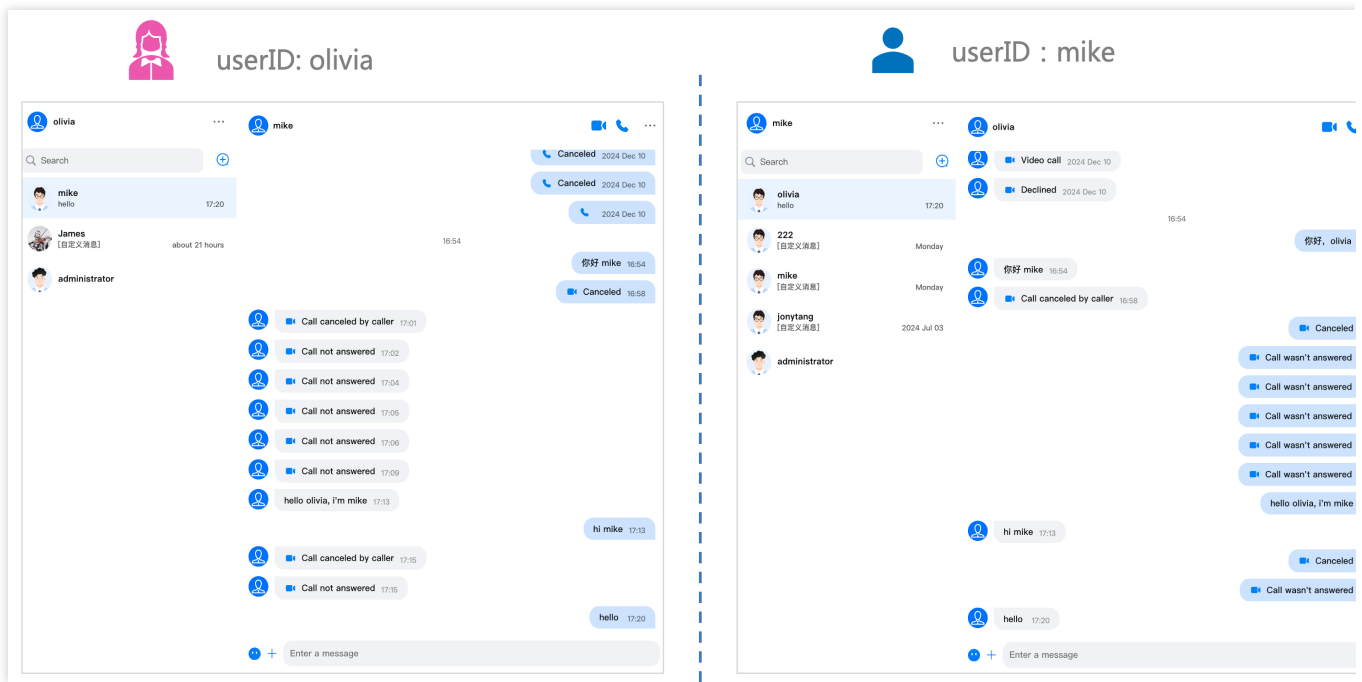
```
// 3. Please use the TUICallKit component in UIKitProvider
return (
  <div style={{display: 'flex', height: '100vh'}}>
    <UIKitProvider language={"en-US"}>
      <TUICallKit style={callStyle} />
      <div style={{maxWidth: '400px'}}>
        <Profile />
        <ConversationList />
      </div>
      <Chat>
        <ChatHeader enableCall={true}/>
        <MessageList />
        <MessageInput />
      </Chat>
      <ChatSetting />
    </UIKitProvider>
  </div>
);
```

Step 5: Launching the Project

```
npm run start
```

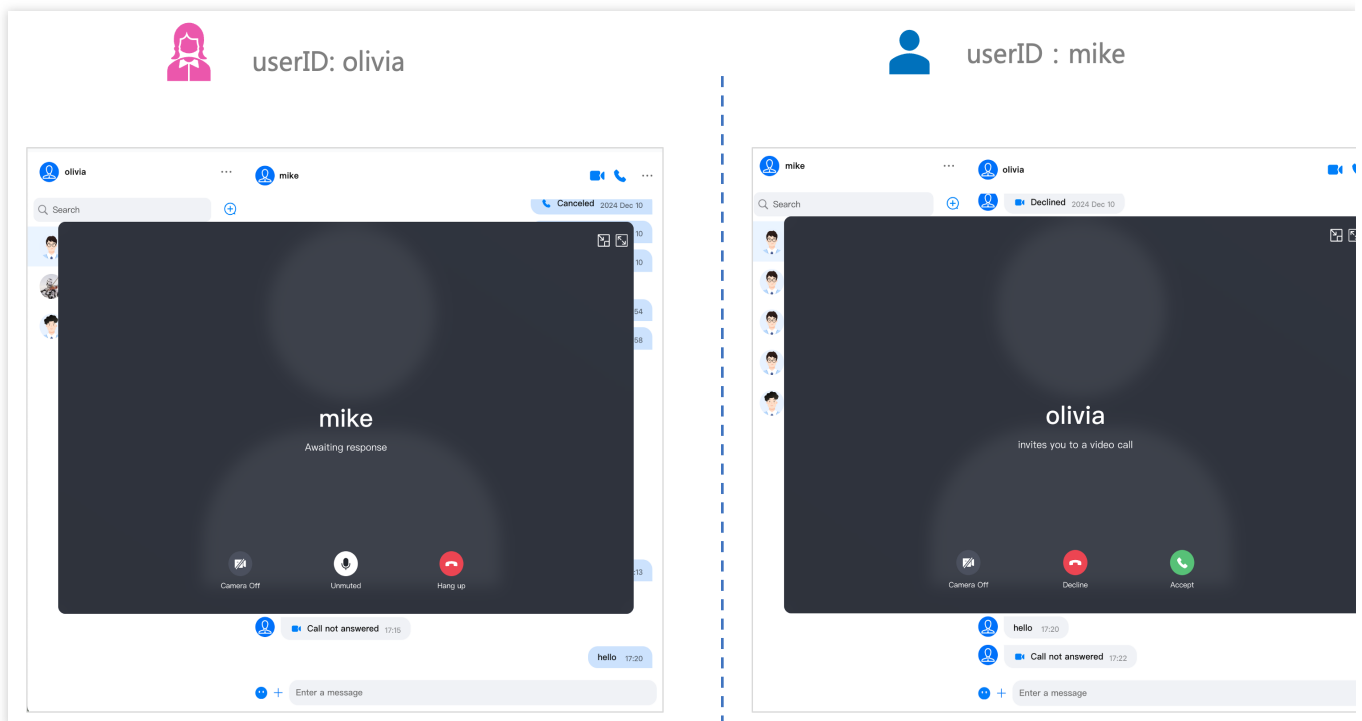
Step 6: Sending Your First Message

Enter your message in the input box and press Enter to send.



Step 7. Make your first call

As shown in the figure, click the video/audio icon in the toolbar to realize the call effect.



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.

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](#).

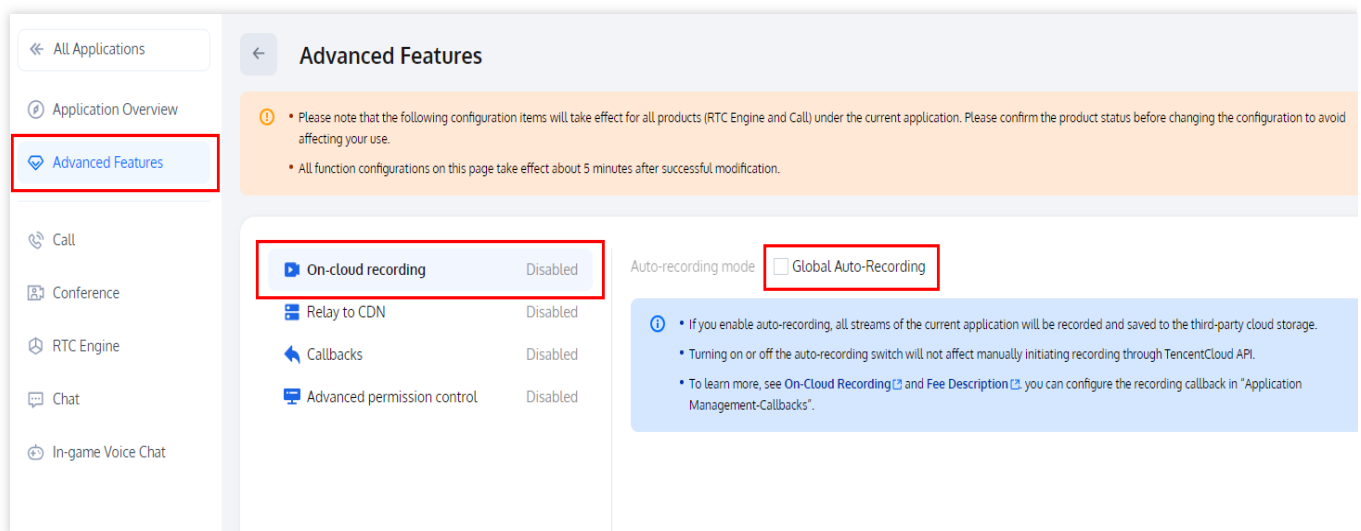
Note:

`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](#).

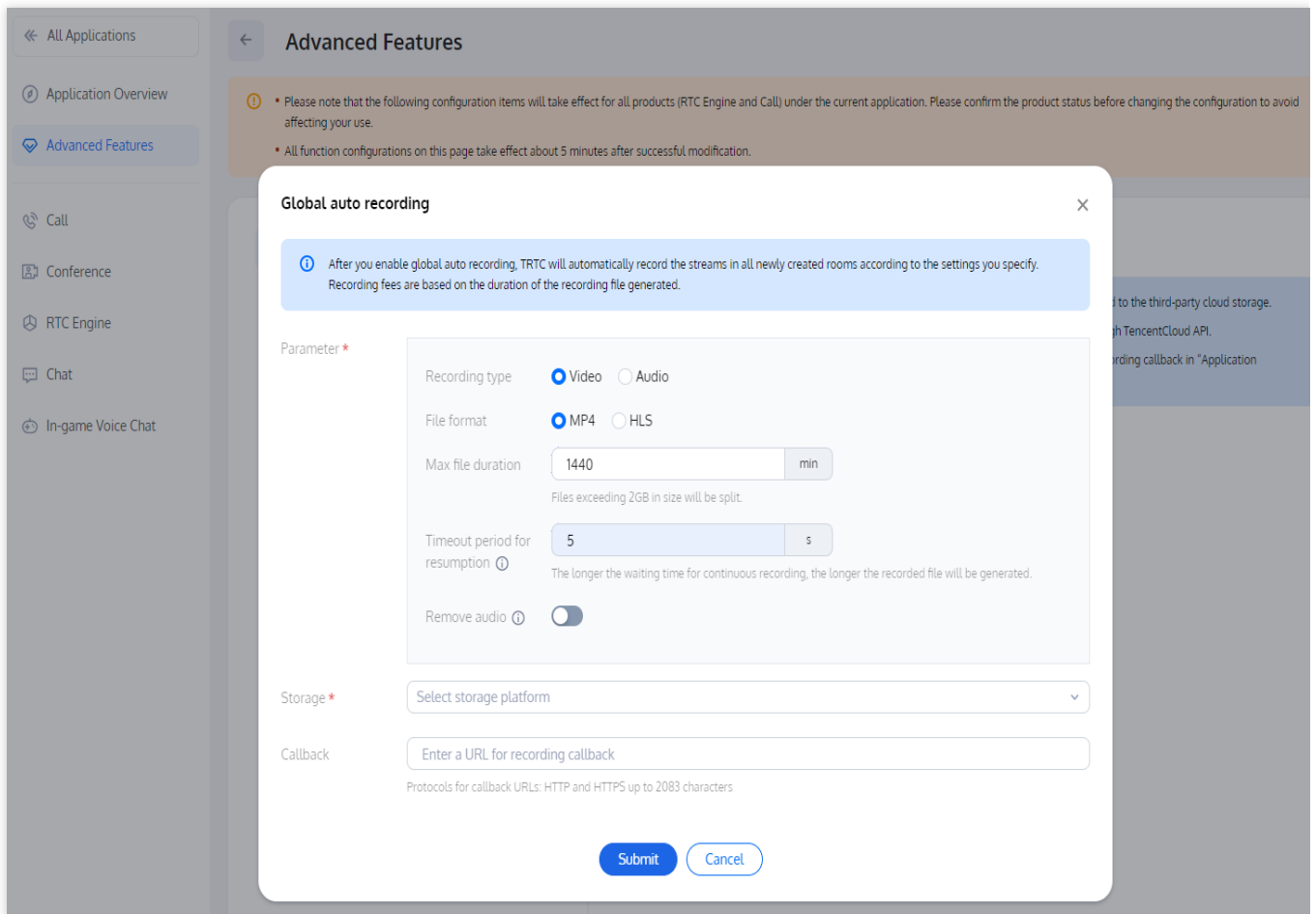
Scheme 1. Automatic Recording (Recommended)

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.

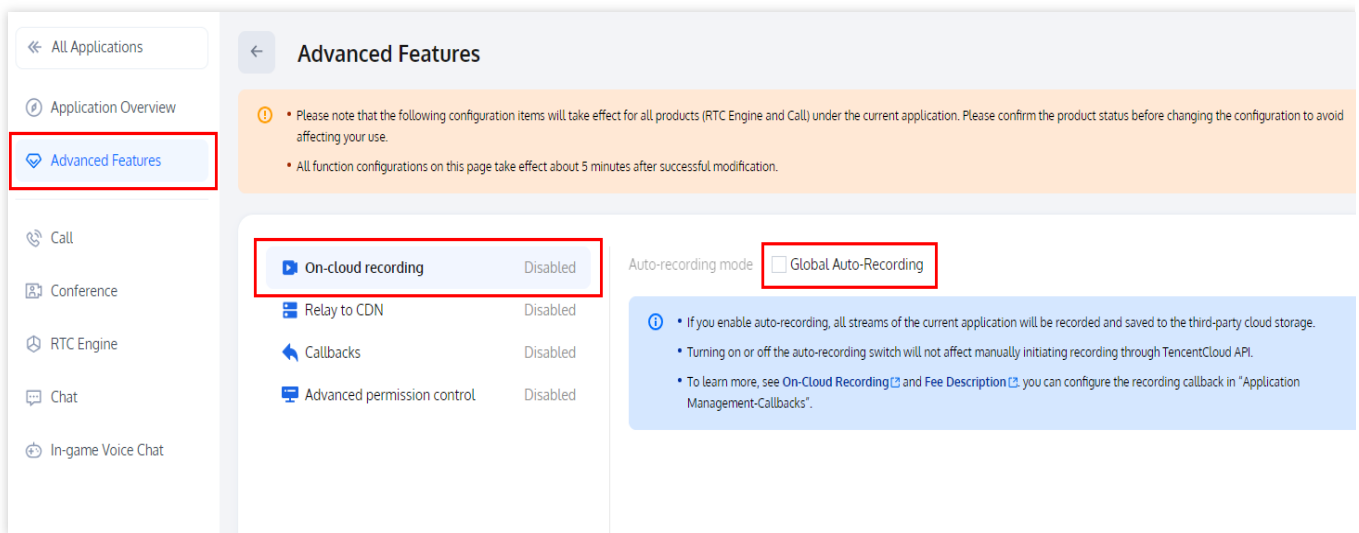
After the global automatic recording feature is enabled, when a call is answered and there is audio/video being upstreamed, a recording task will be triggered automatically, and recording will stop automatically when the call ends. If a user leaves the room due to poor network conditions or other exceptions, 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 relevant  
    }  
});
```

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(Android&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 sharedInstance] 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).


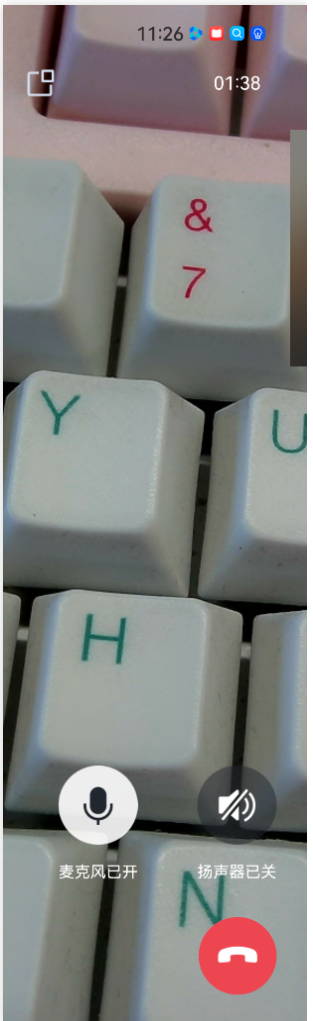
Due to the influence of system performance usage, camera collection capabilities, browser restrictions, and other factors, the actual values of video resolution, frame rate, and bit rate may not necessarily match the set values exactly. In such scenarios, the browser will automatically adjust the Profile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

USB Camera Capture

Last updated : 2024-09-11 16:42:13

This article introduces how to integrate a USB camera with the `TUICallKit` component on Android system devices to enable video call features. Currently, this feature supports Android TUICallkit components and Flutter (Android) TUICallKit components.

Integration effect

Physical Connection Photo	Video Call View
	

Environment preparations

Android 5.0 (SDK API Level 21) or later.

Gradle 4.2.1 or later.

Android system devices: mobile phone, tablet, or other custom devices.

Step 1: Prepare the conditions

1. Before using the USB Camera feature provided by Tencent Cloud, you need to go to the console to enable audio and video services for your application and purchase the Group Call Plan. For detailed steps, please refer to [Activate Service](#).

2. This plugin needs to be used in conjunction with the TUICallKit Component. Please integrate the TUICallKit Component first:

[Android](#)

[Flutter](#)

Step 2: Integrate the component

In the `build.gradle` file in the app directory of your project, add the following dependency code:

```
implementation "io.trtc.uikit:usb-camera:latest.release"
```

After completing the above steps, you can use an external camera for video calls in the `TUICallKit` component.

FAQs

Cannot open camera on Android 9 and Android 10 devices?

Cause: On Android 9 and Android 10, requesting USB permissions will also check for Camera permissions. However, due to an anomaly in the Android framework, even if Camera permissions are granted, the check still fails, preventing USB permissions from being granted.

Solution: Modify the app's `targetSdkVersion` to 27 or lower.

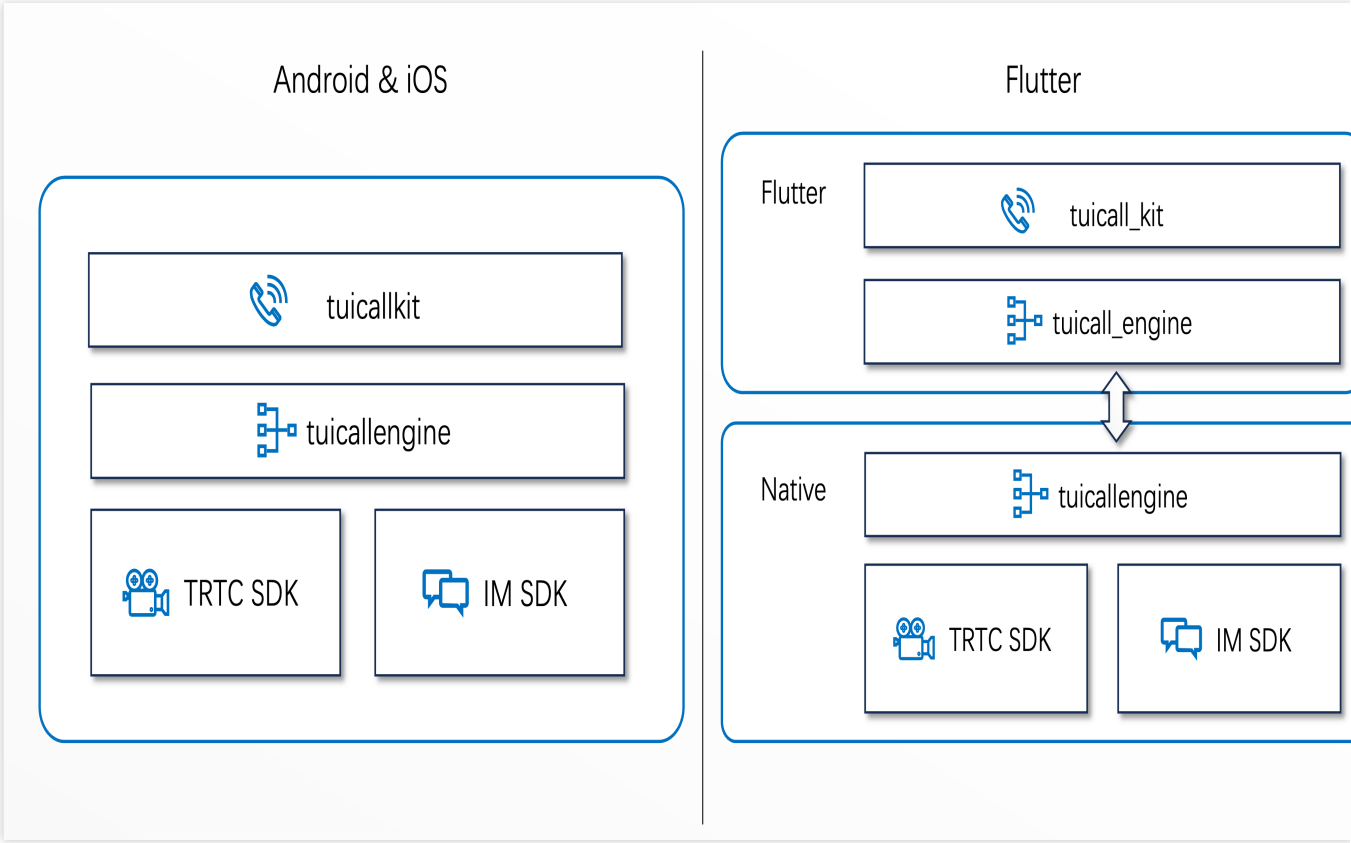
```
defaultConfig {  
    targetSdkVersion 27  
}
```

Volume Increment

Last updated : 2024-12-11 08:55:06

This document will analyze the potential volume increment in applications based on different development frameworks when integrating TUICallKit in detail.

TUICallKit Component Structure



TUICallKit Volume Increment Analysis

Taking TUICallKit 2.6.0 as an example, the following are the volume increments of applications output from each platform after integration with different development frameworks:

Android & iOS

System	CPU architecture	Increment
--------	------------------	-----------

Android	arm64-v8a	16.2 MB
	armeabi-v7a	14.8 MB
iOS	arm64	15.5 MB

Flutter

System	CPU architecture	Increment
Android	arm64-v8a	14.9 MB
	armeabi-v7a	13.7 MB
iOS	arm64	15.5 MB

Note:

In the volume test on the Android platform, the compression feature for jniLibs and dex is enabled by default.

The volume increment data in this document is for reference only. The actual increment may vary due to factors such as plugin version, dependencies, build configuration, platform, and resource file. It is recommended to perform an actual build after integrating the plugin to obtain accurate data.

Multi-Person Call

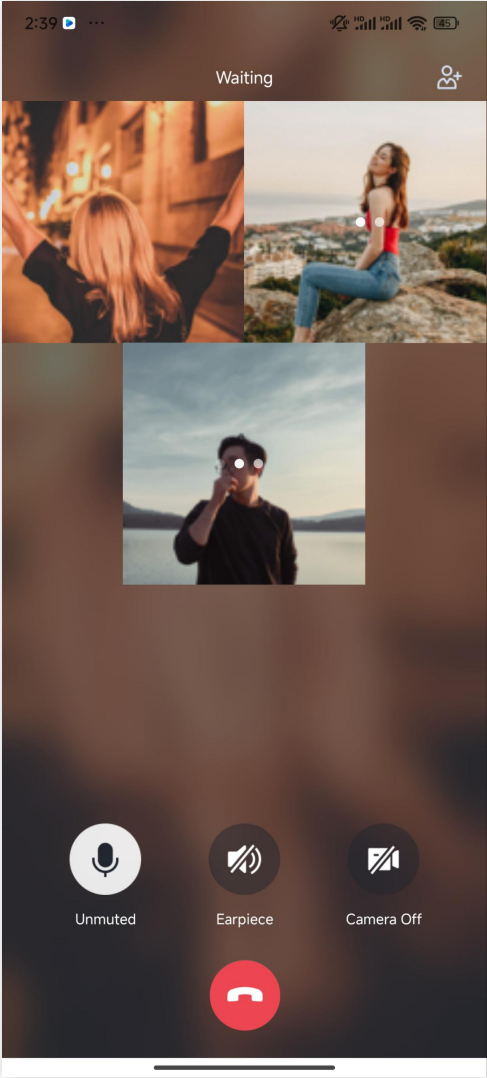
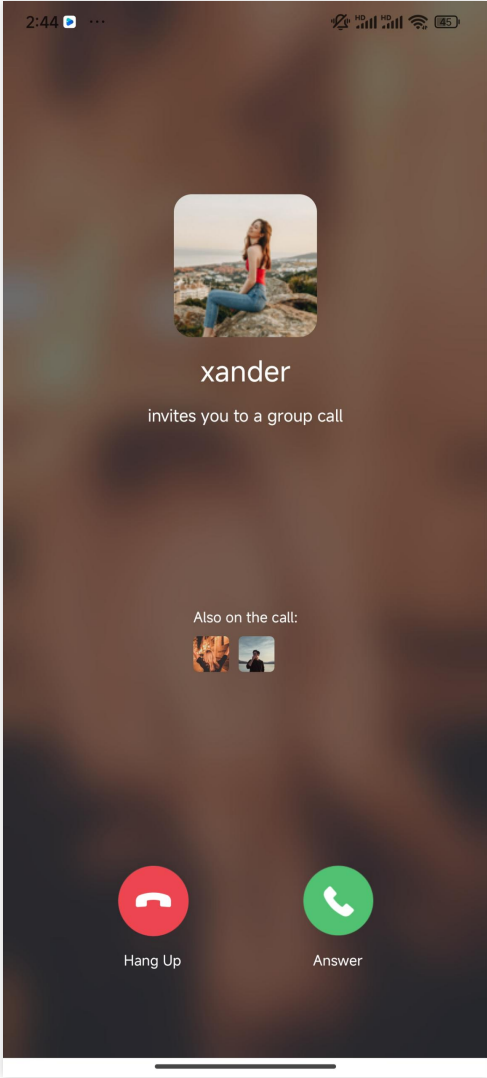
Android&iOS&Flutter

Last updated : 2025-02-18 20:58:21

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 multiplayer calls. The expected outcome is shown in the figure below.

Initiate a multiplayer call	Receive a multiplayer call request	
		

Initiate a Multiplayer Call

Initiate a group call by calling 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

val list = mutableListOf<String>()
list.add("mike")
list.add("tate")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, null,

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

List<String> list = new ArrayList<>();
list.add("mike")
list.add("tate")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, null,

import TUICallKit_Swift
import RTCRoomEngine

TUICallKit.createInstance().calls(userIdList: ["mike","tate"], callMediaType: .audi
} fail: { code, message in
}

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

[[TUICallKit sharedInstance] calls:@[@"mike", @"tate"] callMediaType:TUICallMediaTy
} fail:^(int code, NSString * _Nullable errMsg) {
}];

import 'package:tencent_calls_uikit/tencent_calls_uikit.dart';

void call() {
    List<String> userIdList = ['vince','mike'];
```

```
TUICallKit.instance.call(userIdList, TUICallMediaType.audio);  
}
```

Join a Call

Call the join API to actively join an existing audio and video call.

Android (Kotlin)

Android (Java)

iOS (Swift)

iOS (Objective-C)

Flutter (Dart)

```
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit  
  
TUICallKit.createInstance(context).join("12345678")  
  
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit;  
  
TUICallKit.createInstance(context).join("*****");  
  
import TUICallKit_Swift  
  
TUICallKit.createInstance().join(callId: "")  
  
#import "TUICallKit_Swift-Swift.h"  
  
[[TUICallKit sharedInstance] joinWithCallId: @*****];  
  
import 'package:tencent_calls_uikit/tencent_calls_uikit.dart';  
  
void join() {  
    TUICallKit.instance.join("*****")  
}
```

Web&H5

Last updated : 2025-06-10 15:31:01

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 (up to 9 people). The expected outcome is shown in the figure below.

Web	Mobile Client

Initiate a Multiplayer Call

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

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

Join a group call

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

```
try {
  const params = {
    callId: 'xxx'
  };
  await TUICallKitServer.join(params);
} catch (error: any) {
  console.error(`[TUICallKit] join failed. Reason: ${error}`);
}
```


uni-app (Anroid&iOS)

Last updated : 2025-05-27 17:57:15

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 Group Call Invitation

Group call

Initiate a group call

Launch a group call using the calls API.

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

Join a group call

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

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

```
const options = {
  callId: 'xxx',
};
TUICallKit.join(options, (res) => {
  if (res.code === 0) {
    console.log('join success');
  } else {
    console.log(join failed, error message = ${res.msg});
  }
});
```

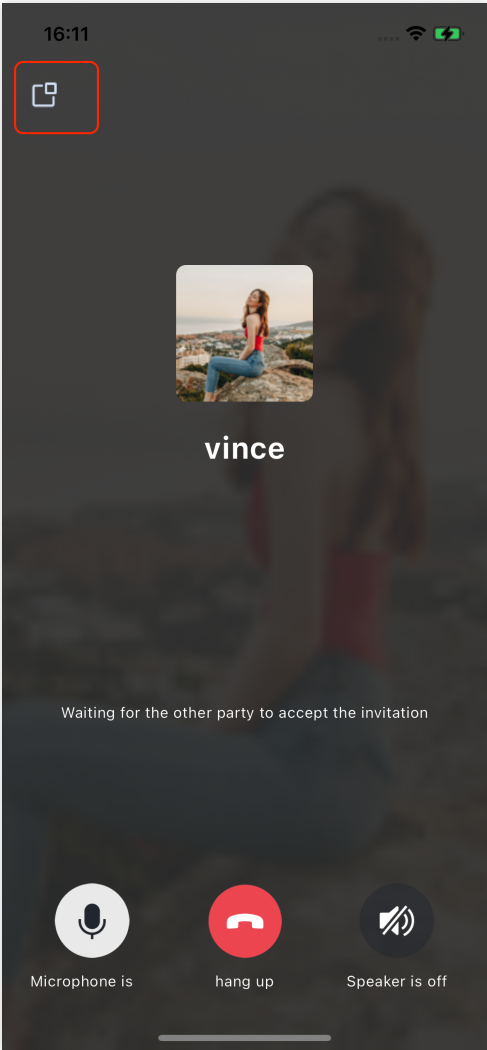
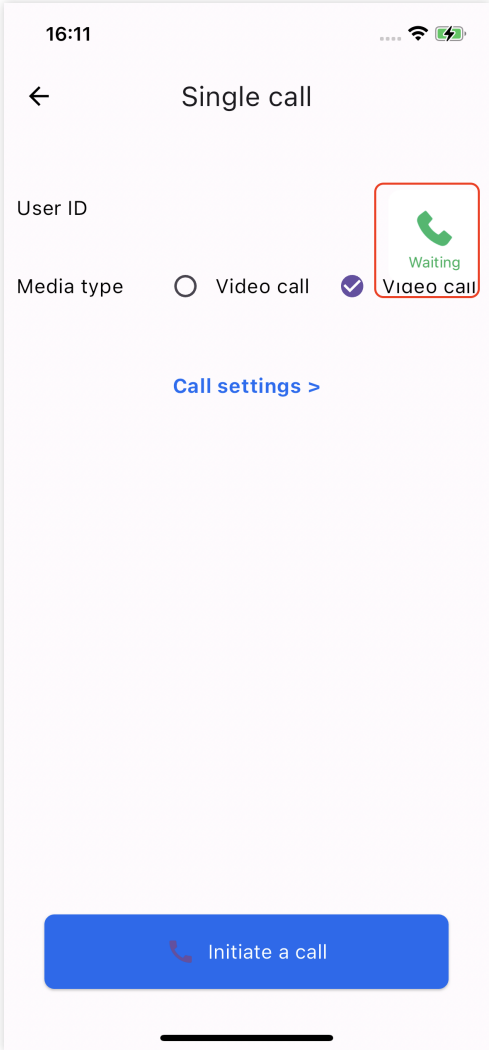
Floating Window

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
 <p>The screenshot shows the Tencent RTC app interface. At the top left, there is a floating window icon (a square with a smaller square inside) highlighted by a red box. Below it, there is a profile picture of a person and the name "vince". At the bottom, there are three buttons: "Microphone is", "hang up", and "Speaker is off".</p>	 <p>The screenshot shows the "Single call" interface. At the top, there is a back arrow and the title "Single call". Below it, there is a "User ID" field. To the right of the "User ID" field, there is a green phone icon with the text "Waiting" and "Video call" below it, highlighted by a red box. Below the "User ID" field, there is a "Media type" section with two options: "Video call" (selected) and "Video call" (unselected). At the bottom, there is a blue button labeled "Initiate a call".</p>

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 sharedInstance] enableFloatWindowWithEnable:YES];

import 'package:tencent_calls_uikit/tencent_calls_uikit.dart';

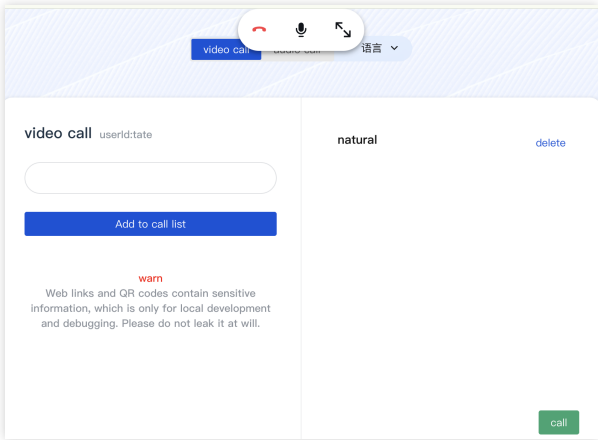
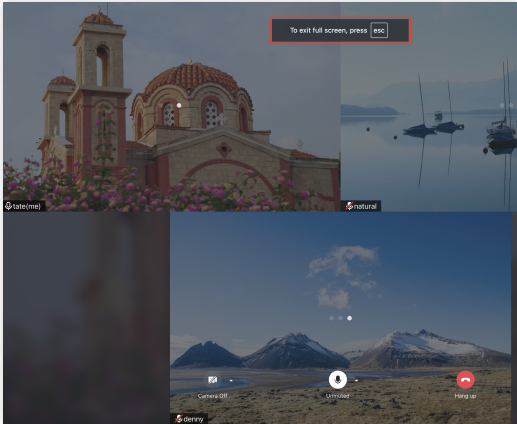
void enableFloatWindow() {
    TUICallKit.instance.enableFloatWindow(true);
}
```

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}  
>
```

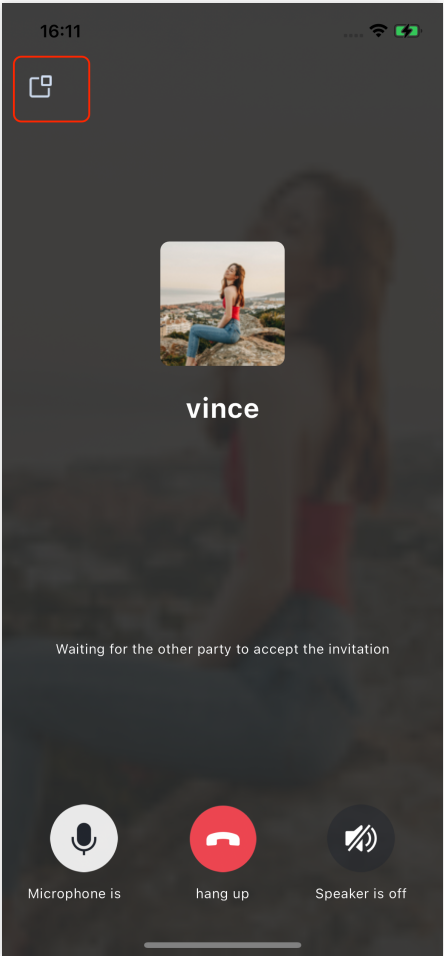
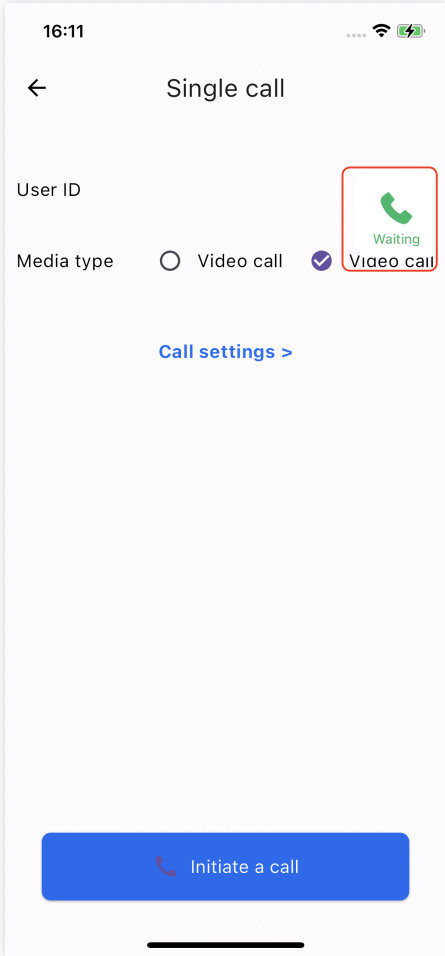
```
<TUICallKit  
  :allowedMinimized="true"  
  :allowedFullScreen="true"  
>
```

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

Floating Window feature

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

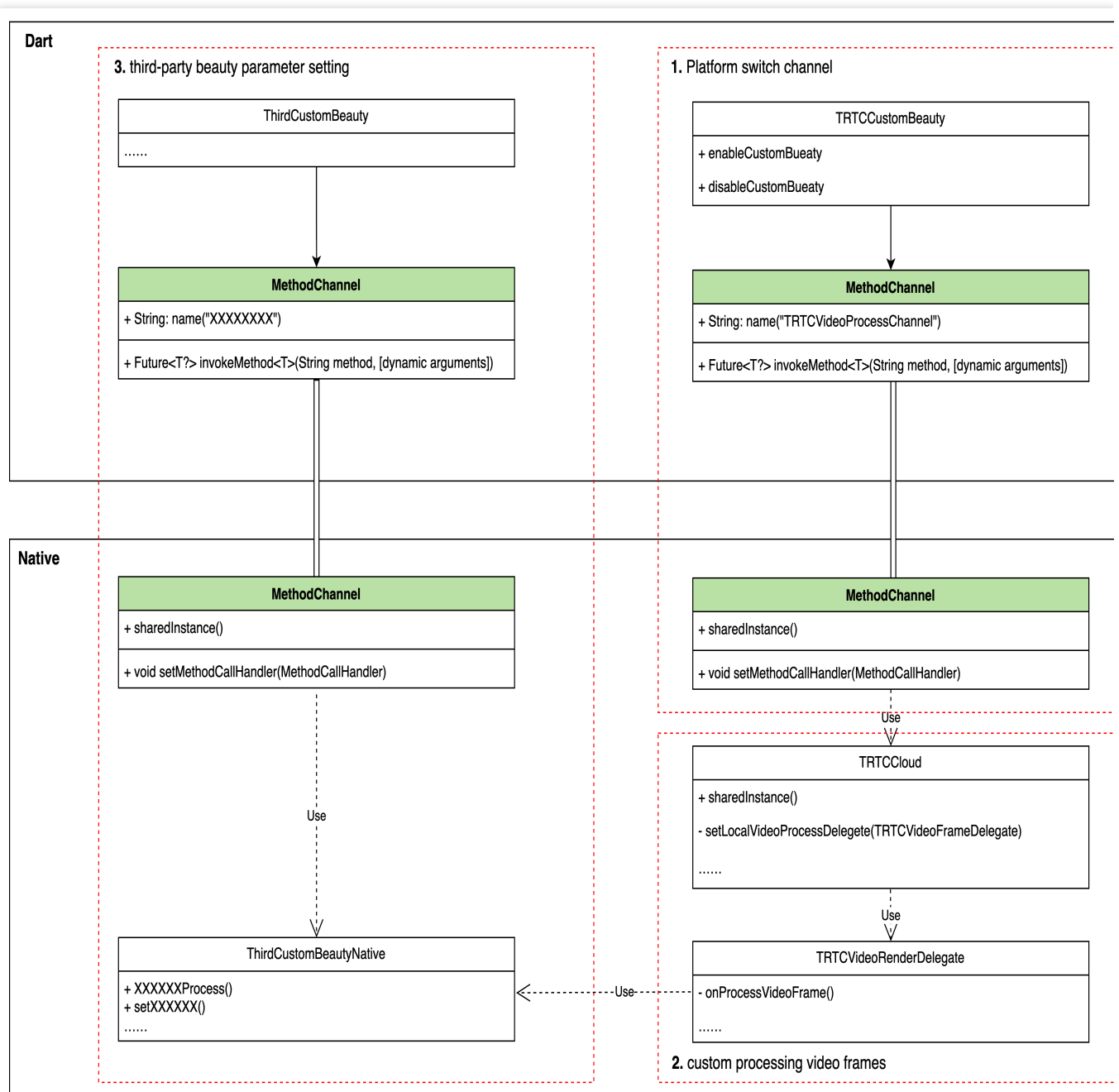

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');  
const enable = true;  
TUICallKit.enableFloatWindow(enable);
```

Beauty Effects Flutter

Last updated : 2025-01-17 11:23:37

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

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



The access plan is divided into 3 steps:

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

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

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

Integrate third-party beauty effects

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

Implement Dart layer interface:

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

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

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

Implement the corresponding Native layer interface:

java

swift

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

    private MethodChannel channel;

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

        channel = new MethodChannel(flutterEngine.getDartExecutor().getBinaryMessenger(),
            channelName);
        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:

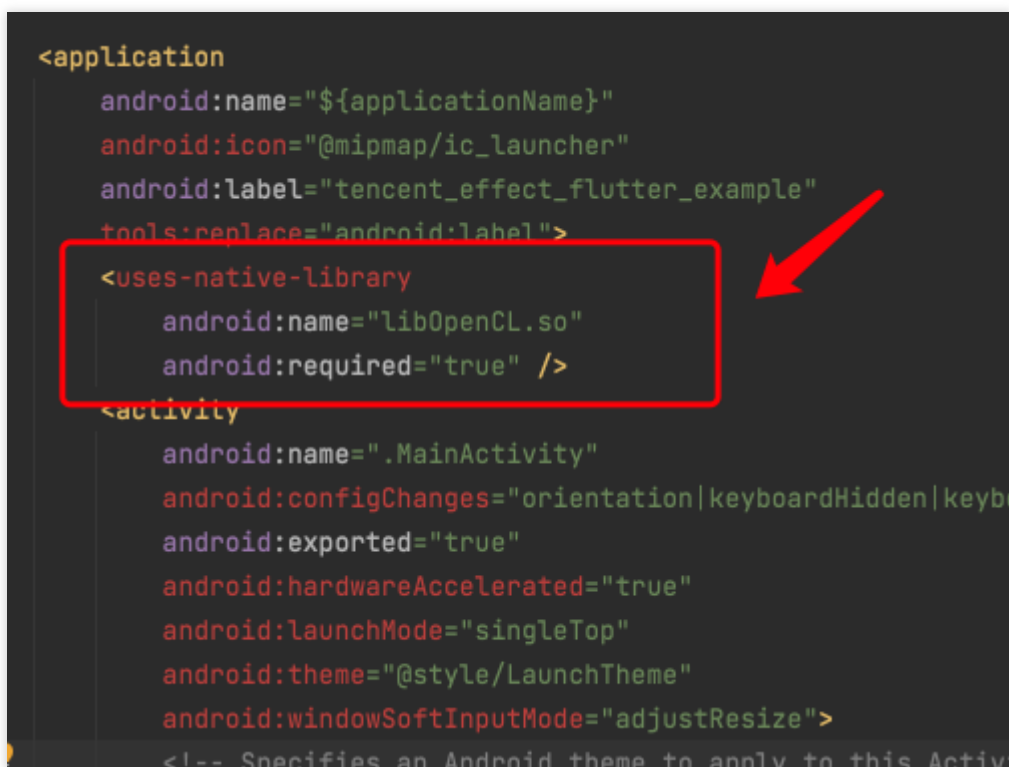
```
dependencies {
    implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

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

- 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.
- 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.

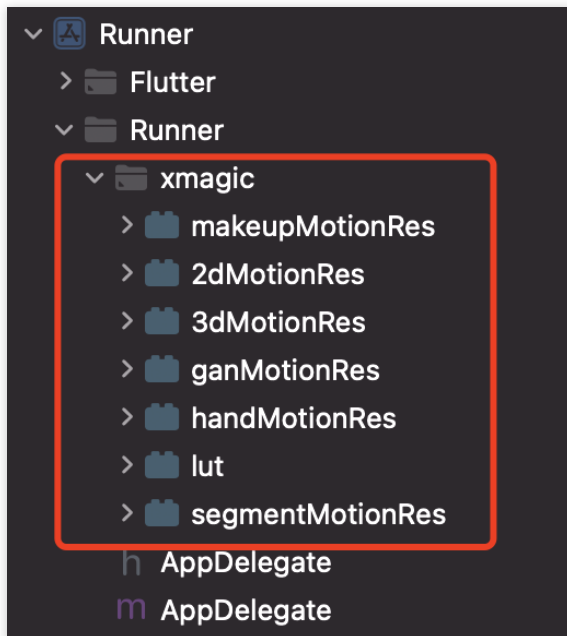
If you enable such compile optimization, you need to add these keep rules to prevent xmagic code from being cut off:

```
-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() {
    XmagicApiManager.getInstance().onCreateApi();
}

@Override
public void onGLContextDestory() {
    XmagicApiManager.getInstance().onDestroy();
}
}

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 : 2024-09-06 16:41:34

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:

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_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);
```

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);
```

Set Offline Push Ringtone

For Offline Push Ringtone Settings, please refer to: [FCM Offline Push customizes incoming call ringtone](#).

iOS

Last updated : 2024-09-06 16:41:34

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:

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 audio files under the `Resources\AudioFile` folder:

File Name	Use
phone_dialing.mp3	Ringtone when initiating a call
phone_ringing.mp3	Ringtone when receiving a call

2. Call 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:@""];
```

Set Mute Mode

If you do not require a ringtone, you can set the mute mode via [enableMuteMode](#) interface.

Swift

Objective-C

```
import TUICallKit_Swift

TUICallKit.createInstance().enableMuteMode(enable: true)

#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

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

Set 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 : 2024-09-06 16:41:34

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:

1. Replace Audio File

If you integrate the TUICallKit component via source code dependency, you can replace the 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_ringing.mp3	Ringtone when receiving a call

2. Call 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');
}
```

Set Mute Mode

If you do not need ringing, you can enable the silent mode through [enableMuteMode](#) interface.

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

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

Set 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.iOSound` Setting on the `iOS` platform for offline message ringtones.

2. Android

Note:

The interface supports Huawei, Xiaomi, FCM.

FCM's push ringtone is set as the application ringtone.

For Huawei, Xiaomi, and APNs push ringtone settings, please set the `TUIOfflinePushInfo.iOSound` 's `iOSSound` and `androidSound` fields when calling `Call` and `GroupCall`.

Monitoring Call Status

Android&iOS&Flutter

Last updated : 2025-07-03 11:34:29

This article explains the usage of TUICallKit call status callback. You can monitor call events (e.g., start, end, member join, and leave) via call callback registration.

Note:

On the Android platform, when setting TUICallObserver to listen for callbacks, ensure that the class hosting the callback will not be destroyed. For example, it is not recommended to add the observer in LoginActivity, as when LoginActivity is destroyed, the callback will also be destroyed; it is suggested to observe in the application's Application class or the main application interface.

Call Status Callback

Detect key state changes throughout the call lifecycle.

Kotlin

Swift

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 onCallReceived(callId: String?, callerId: String?, calleeIdList: M
    }

    override fun onCallBegin(roomId: TUICommonDefine.RoomId?, callMediaType: TUICal
    }

    override fun onCallEnd(roomId: TUICommonDefine.RoomId?, callMediaType: TUICallD
    }

    override fun onCallNotConnected(callId: String?, mediaType: TUICallDefine.Media
    }
    .....
}

private fun initData() {
```



```

        TUICallEngine.createInstance(context).addObserver(observer)
    }

import TUICallEngine

TUICallEngine.createInstance().addObserver(self)

func onCallReceived(_ callId: String, callerId: String, calleeIdList: [String], med
}

func onCallBegin(callId: String, mediaType: TUICallMediaType, info: TUICallObserver
}

func onCallEnd(callId: String, mediaType: TUICallMediaType, reason: TUICallEndReaso
}

func onCallNotConnected(callId: String, mediaType: TUICallMediaType, reason: TUICal
}

import 'package:tencent_calls_engine/tencent_calls_engine.dart';

TUICallObserver observer = TUICallObserver(
    onCallReceived: (String callId, String callerId, List<String> calleeIdList, TUI
    }, onCallBegin: (String callId, TUICallMediaType mediaType, CallObserverExtraIn
    }, onCallEnd: (String callId, TUICallMediaType mediaType, CallEndReason reason,
    }, onCallNotConnected: (String callId, TUICallMediaType mediaType, CallEndReaso
    }
    .....
)

void addObserver() {
    TUICallEngine.instance.addObserver(observer);
}

```

Call Member Update Callback

Track individual actions or status changes of call participants, primarily used for real-time display of member dynamics (such as join, leave, reject).

Kotlin

Swift

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 onUserJoin(userId: String?) {
    }

    override fun onUserLeave(userId: String?) {
    }

    override fun onUserInviting(userId: String?) {
    }

    override fun onUserReject(userId: String?) {
    }
    .....
}

private fun initData() {
    TUICallEngine.createInstance(context).addObserver(observer)
}

import TUICallEngine

TUICallEngine.createInstance().addObserver(self)

fun onUserJoin(userId: String) {
}

fun onUserLeave(userId: String) {
}

fun onUserInviting(userId: String) {
}

fun onUserReject(userId: String) {
}
.....

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

```
TUICallObserver observer = TUICallObserver(  
    onUserJoin: (String userId) {  
  
    }, onUserLeave: (String userId) {  
  
    }, onUserInviting: (String userId) {  
  
    }, onUserReject: (String userId) {  
  
    },  
    .....  
)  
  
void addObserver() {  
    TUICallEngine.instance.addObserver(observer);  
}
```

Web&H5

Last updated : 2025-05-27 17:56:28

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 '@tencentcloud/call-engine-js';

let handleUserEnter = function(event) {
  console.log('TUICallEvent.USER_ENTER: ', event);
};

TUICallKitServer.getTUICallEngineInstance().on(TUICallEvent.USER_ENTER, handleUserEnter);
TUICallKitServer.getTUICallEngineInstance().off(TUICallEvent.USER_ENTER, handleUserEnter);
```

Component Callback Event

The TUICallKit component offers call status callbacks, which can be used to implement more interaction logic on the business side. Please refer to the [TUICallKit Attribute Overview](#) for more details.

beforeCalling : Executed prior to the commencement of the call.

afterCalling : Executed upon the completion of the call.

React

Vue

```
function App() {
  const handleBeforeCalling = () => {
    console.log("[TUICallKit Demo] beforeCalling");
  };
  const handleAfterCalling = () => {
    console.log("[TUICallKit Demo] afterCalling");
  };
  return (
    <TUICallKit
      beforeCalling={handleBeforeCalling}
      afterCalling={handleAfterCalling} />
  );
}
```

```
)  
}  
  
<template>  
  <TUICallKit  
    :beforeCalling="handleBeforeCalling"  
    :afterCalling="handleAfterCalling" />  
</template>  
<script setup>  
function handleBeforeCalling() {  
  console.log("[TUICallKit Demo] beforeCalling");  
}  
function handleAfterCalling() {  
  console.log("[TUICallKit Demo] afterCalling");  
}  
</script>
```

uni-app (Android & iOS)

Last updated : 2024-04-30 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-uikit-vue/* ./src/components/TUICallKit
```

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

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;

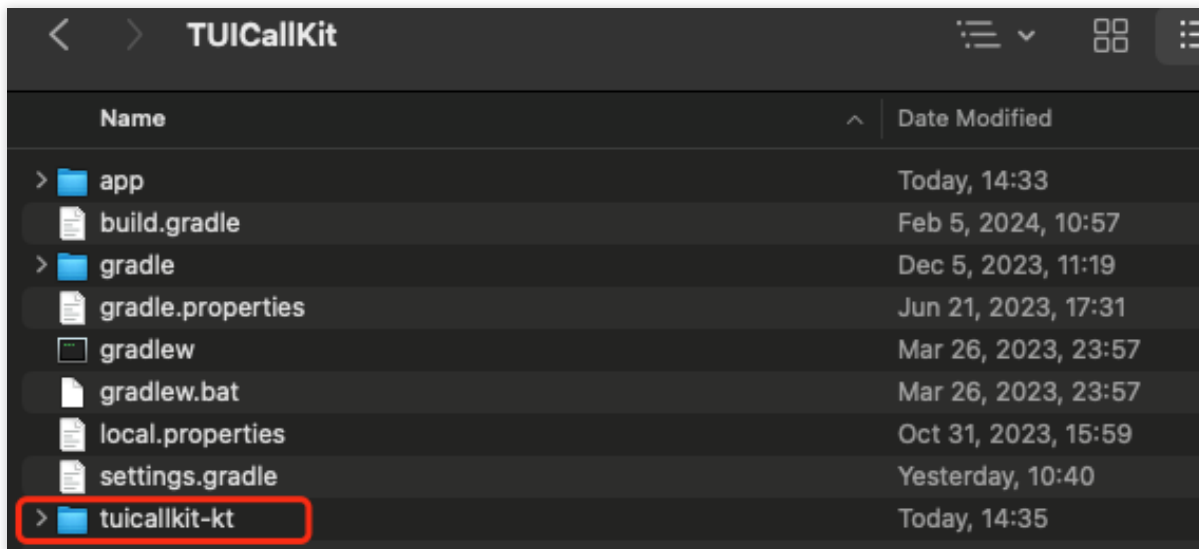
public class MainActivity extends BaseActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        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 .

Solution (TUICallKit)

WhatsApp Clone

Last updated : 2025-06-17 12:02:51

WhatsApp Clone is a project aimed at cloning WhatsApp, offering powerful chat and high-quality audio and video call features. It utilizes Tencent's ChatUIKit and CallKit components to achieve a smooth chatting experience and stable, clear audio and video calls. The project emphasizes user data security, employs advanced encryption technology, and is cross-platform, providing users with a convenient, efficient, and secure instant messaging experience.

--	--	--	--

WhatsApp Clone Feature Characteristics

Full Platform Support Supports over 20,000 cross-platform devices, including iOS, Android, Web, and Flutter.	Single/Group Chat Supports personal and group calls, and various message types such as text, image, audio, and video messages.	Session Management View information about unread messages, latest, pinned, or deleted chats.
Audio/Video Call Supports high-quality, low-latency audio and video calls.	Group Call Supports multi-user online audio and video calls, and allows inviting others to join during the call.	Group Management Create groups, manage group members, and customize avatars.
Call Notifications and Offline Push If the app is not in the foreground, push notifications will be sent.	Virtual Background During a video call, you can set a blurry background or image background.	Floating Window The call page can be displayed as a floating window.
AI-powered Noise Cancellation Remove background noise from your audio and video calls using AI	Weak Network Call Lagging Optimization	Global Deployment Providing services to users in over 200 countries and regions.

technology.

Reduce lagging rate and loading time in weak network environments.

Compliant with the <General Data Protection Regulation> (GDPR).

WhatsApp Clone Software Management

Dashboard View audio/video call and chat data through the dashboard.	Statistics Quickly obtain statistics and analyze message activities through the console.	On-cloud recording You can record and save audio and video calls.
Data Management You can manage users, groups, and other data in the Console.	Server API Manage your application through powerful server APIs.	Push Management Provides tools for push message lifecycle data statistics and troubleshooting.

Quick Start

To help you get started quickly with our project, we provide a detailed installation and configuration guide. Please follow the steps below:

1. Download the WhatsApp Clone project based on your target platform (iOS, Android, Web, Flutter).
2. Refer to the README file in the respective platform folder for project configuration.
3. Run the program to experience the provided chat, call, push, and other features.

Contact Us

If you encounter any issues while using our project or have any questions or suggestions, please don't hesitate to contact us through the following methods:

Visit our Official Website: trtc.io

Send an email to: info_rtc@tencent.com

We look forward to working with you to create a more powerful and user-friendly solution.

Server APIs (TUICallKit)

REST API

REST API Introduction

Last updated : 2025-04-17 16:28:40

REST API is the backend HTTP management API of CallKit SDK. Its main purpose is to provide developers with a simple management portal.

For security, REST API provides only HTTPS APIs.

Prerequisites

To call REST API, you must purchase or claim the required package for CallKit Sdk.

Warning:

If you use the deprecated interfaces `TUICallKit.call()` or `TUICallKit.groupCall()` to initiate a call, please check the **Deprecated Document** directory. If you have any questions, you can contact: info_rtc@tencent.com.

Calling Method

Request URL

The URL format of the REST API is as follows:

```
https://xxxxxx/$ver/$servicename/$command?  
sdkappid=$SDKAppID&identifier=$identifier&usersig=$usersig&random=99999999&cont  
enttype=json
```

The meaning and values of each parameter are as follows (parameter names and values are case-sensitive):

Parameter	Meaning	Value
https	Request protocol	The request protocol is HTTPS, and the request method is POST.
xxxxxx	Request domain	Dedicated domain name corresponding to the country/region where the SDKAppID is located: China: <code>console.tim.qq.com</code> Singapore: <code>adminapisgp.im.qcloud.com</code>

ver	Protocol version number	Fixed as <code>v4</code>
servicename	Internal service name. Different service names correspond to different service types.	<code>v4/call_engine_http_srv/get_call_info</code> , where <code>call_engine_http_srv</code> is the <code>servicename</code> . For more details, see REST API List .
command	Command word, combined with servicename to identify specific business functionality.	Example: <code>v4/call_engine_http_srv/get_call_info</code> , where <code>get_call_info</code> is the <code>command</code> . For more details, see REST API list .
sdkappid	App obtains the application identifier in the IM console.	Obtain when applying for access.
identifier	username must be an App administrator account when calling REST API.	see App admin
usersig	The password corresponding to the username.	see generate UserSig
random	Random number parameter indicating the current request	32-bit unsigned integer random number, value ranges from 0 to 4294967295
contenttype	Request format	The fixed value is <code>json</code> .

Note:

The App Server must use an App administrator account as the identifier when calling REST API.

The App can generate a UserSig for the administrator account each time it calls the REST API, or reuse a fixed UserSig. Special attention should be paid to the valid period of the UserSig.

HTTP Request Body Format

REST API only supports POST method. Its request body is in JSON format. Please refer to the detailed description of each API for the specific body format.

It is important to note that the POST body cannot be empty. Even if a protocol packet does not need to carry any information, an empty JSON Object, i.e., `{ }` , needs to be carried.

HTTP Return Code

Unless a network error (such as 502 error) occurs, the call result of the REST API is 200. The actual API Invocation Error Code and error information are returned in the HTTP response body.

HTTP Response Body Format

The response packet body of the REST API is also in JSON format. Its format has the following features:

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-70e312f1de024af5a36714b7b71da224-O-Seq-63504"
  // Other response contents of REST API
}
```

The response payload necessarily contains the attributes ActionStatus, ErrorInfo, ErrorCode, and RequestId, whose meanings are as follows:

Field	Type	Description
ActionStatus	String	Request processing result. OK means processing successful; FAIL means processing failed. If it is FAIL, ErrorInfo contains the failure reason.
ErrorInfo	String	Failure reason
ErrorCode	Integer	Error code: 0 means successful; any other value indicates failure. You can query the Error Code Table for specific reasons.
RequestId	String	Unique request ID. It is returned for each request. RequestId is required for locating a problem.

Calling Example

The following is an example of using the REST API to get all groups in the App.

HTTPS request

```
POST /v4/group_open_http_svc/get_appid_group_list?
usersig=xxx&identifier=admin&sdkappid=88888888&random=99999999&contenttype=json
HTTP/1.1
Host: console.tim.qq.com
Content-Length: 22
{
  "Limit": 2
}
```

HTTPS response

```
HTTP/1.1 200 OK
Server: nginx/1.7.10
Date: Fri, 09 Oct 2015 02:59:55 GMT
Content-Length: 156
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Requested-With
Access-Control-Allow-Methods: POST

{
  "ActionStatus": "OK",
  "ErrorCode": 0,
  "GroupIdList": [
    {
      "GroupId": "@TGS#1YTTZEAEG"
    },
    {
      "GroupId": "@TGS#1KVTZEA EZ"
    }
  ],
  "TotalCount": 58530
}
```

REST API Common Error Code

Error Code	Description of Meaning
60002	HTTP parsing error. Please check the HTTP Request URL Format.
60003	HTTP request JSON parsing error. Please check the JSON format.
60004	Error in account or signature in the request URL or JSON body.
60005	Error in account or signature in the request URL or JSON body.
60006	SDKAppID is invalid. Check the validity of the SDKAppID.
60007	The REST API call frequency exceeds the limit. Reduce the request frequency.
60008	Service request timeout or HTTP request format error. Please check and retry.

60009	Request resource error. Please check the request URL.
60010	The request requires App administrator permissions.
60011	The SDKAppID request frequency exceeds the limit. Reduce the request frequency.
60012	The RESTful API requires an SDKAppID. Please check the SDKAppID in the request URL.
60013	HTTP response packet JSON parsing error.
60014	Account switching timeout.
60015	Error in account type in request body. Please confirm that the account is in string format.
60016	SDKAppID is disabled.
60017	The request is disabled.
60018	Frequent requests. Please retry later.
60019	Frequent requests. Try again later.
60020	Your Professional Edition Package has expired and been disabled. Please log in to Chat purchase page to repurchase a package. It will take effect 5 minutes after purchase.
60021	The call source IP of the RestAPI is illegal.

FAQ

REST API Request May Timeout and Fail to Receive Any Response?

1. The timeout period for the Room backend REST API settings is 3 s. The timeout period set by the caller should be longer than 3 s.
2. `telnet console.tim.qq.com 443` Confirm whether you can connect to the service port.
3. Use `curl -I https://console.tim.qq.com` for a simple test to see if the status code is 200.
4. Confirm whether the dns server configuration of the machine is an internal dns server or a public dns server. If it is an internal dns server, please ensure that the network outbound of the dns server matches the regional carrier where the network outbound IP of this machine resides.
5. Recommend that business callers use the "persistent connection + connection pool" mode.

Note:

Since the TCP connection duration of HTTPS non-persistent connections is relatively large and there is TCP + tls handshake overhead for each request, so it's recommended to use persistent connection access for REST APIs. Scenarios using the standard HTTP library: For HTTP1.0, you need to specify the request header Connection: keep-alive. HTTP1.1 supports long connection by default. In scenarios where HTTPS requests are encapsulated based on TCP, TCP connections can be reused to send and receive requests.

REST API List

Last updated : 2025-03-31 16:28:50

Call Status

Feature Description	API
Retrieve the real-time call status	call_engine_http_srv/get_call_info

Call Record

Feature Description	API
Retrieve records by CallId	call_record_http_srv/get_record_by_callid
Retrieve records by conditions	call_record_http_srv/get_record_by_filter

Call Records

Get Records by CallId

Last updated : 2025-04-17 16:28:40

Feature Description

App admins can use this API to obtain the call record information of the specified CallId.

Warning:

If you use the deprecated interfaces `TUICallKit.call()` or `TUICallKit.groupCall()` to initiate a call, please check the **Deprecated Document** directory. If you have any questions, you can contact: info_rtc@tencent.com.

API Call Description

Sample Request URL

```
https://xxxxxx/v4/call_record_http_srv/get_record_by_callid?sdkappid=88888888&ident
```

Request Parameters

The following table only lists the parameters involved in modification when calling this API and their descriptions. For more parameter details, please see [REST API Introduction](#).

Parameter	Description
xxxxxx	Dedicated domain name corresponding to the country/region where the SDKAppID resides: China: <code>console.tim.qq.com</code> Singapore: <code>adminapisgp.im.qcloud.com</code> Seoul : <code>adminapikr.im.qcloud.com</code>
v4/call_record_http_srv/get_record_by_callid	Request API
sdkappid	Assigned SDKAppID in the Chat console when creating an application
identifier	Must be an App administrator account. For more details, see App Administrator

usersig	Generated signature of the App administrator account. For specific operations, see Generate UserSig
random	Enter a random 32-bit unsigned integer. The value ranges from 0 to 4294967295
contenttype	The request format has a fixed value of <code>json</code>

Maximum Calling Frequency

200 times per second.

Sample Request Packet

Basic form

```
{
  "CallId": "04c9a0ac-8e38-4a19-be45-349c5ce7911b"
}
```

Request Packet Fields

Field	Type	Required	Description
CallId	String	Required	Call ID

Response Package Example

Basic form

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-da9b3ee8bece466d951a9d93965c3d2c-O-Seq-324151555",
  "Response": {
    "CallRecord": {
      "CallId": "04c9a0ac-8e38-4a19-be45-349c5ce7911b",
      "Caller_Account": "user1",
      "MediaType": "Audio",
      "CallType": "MultiCall",
      "StartTime": 1739868165,
      "EndTime": 1740064224,
      "AcceptTime": 1739872743,
      "CallResult": "NormalEnd",
      "CalleeList_Account": [
```

```

        "user1",
        "user5",
        "user2"
    ],
    "RoomId": "roomid-1434",
    "RoomIdType": 2
}
}
}

```

Response Packet Fields

Field	Type	Description
ErrorCode	Integer	Error code, 0 indicates success, non-zero indicates failure
ErrorInfo	String	Error message
ActionStatus	String	Request processing result. OK: processing successful; FAIL: processing failed
RequestId	String	Unique request ID. It is returned for each request. RequestId is required for locating a problem
CallRecord	Struct	Call record information
CallId	String	Call ID
Caller_Account	String	Caller ID
MediaType	String	Media type: Video video call Audio audio call
CallType	String	Call Type: SingleCall one-to-one call MultiCall group call
StartTime	Integer	Call initiation timestamp (in seconds)
EndTime	Integer	Call end timestamp (in seconds)
AcceptTime	Integer	Call connection timestamp (in seconds)
CallResult	Integer	Call result Cancel : caller cancel the call before connecting Reject : recipient decline the call

		<code>NotAnswer</code> : recipient timeout before answering <code>NormalEnd</code> : Call connected and ended normally <code>CallBusy</code> : call busy <code>Interrupt</code> : Call interrupted due to reasons such as network issues
<code>CalleeList_Account</code>	Array	Call Member List
<code>RoomId</code>	String	TRTC Room ID
<code>RoomIdType</code>	Integer	RoomId type: 1 digit room number 2 string Room Number

Error Code Description

Unless a network error (for example, 502 error) occurs, the HTTP return code of this API is 200. The actual error code and error information are indicated by `ErrorCode` and `ErrorInfo` in the response packet body.

For common error codes (60000 to 79999), see the [Error Code](#) document.

The private error codes of this API are as follows:

Error Code	Description
101001	Internal server error, please retry.
101050	The call record does not exist.

Retrieve Records by Conditions

Last updated : 2025-04-17 16:28:40

Feature Description

App admins can retrieve call records within 7 days under specified conditions using this API.

Warning:

If you use the deprecated interfaces `TUICallKit.call()` or `TUICallKit.groupCall()` to initiate a call, please check the **Deprecated Document** directory. If you have any questions, you can contact: info_rtc@tencent.com.

API Call Description

Sample Request URL

```
https://xxxxxx/v4/call_record_http_srv/get_record_by_filter?sdkappid=88888888&ident
```

Request Parameters

The following table only lists the parameters involved in modification when calling this API and their descriptions. For more details about the parameters, please see [REST API Introduction](#).

Parameter	Description
xxxxxx	Dedicated domain name corresponding to the country/region where the SDKAppID resides: China: <code>console.tim.qq.com</code> Singapore: <code>adminapisgp.im.qcloud.com</code> Seoul : <code>adminapikr.im.qcloud.com</code>
v4/call_record_http_srv/get_record_by_filter	Request API
sdkappid	Assigned SDKAppID in the Chat console when creating an application
identifier	Must be an App administrator account. For more details, see App Administrator
usersig	Generated signature of the App administrator account. For specific operations, see Generate UserSig

random	Enter a random 32-bit unsigned integer in the range of [0,4294967295]
contenttype	The request format has a fixed value of <code>json</code>

Maximum Calling Frequency

200 times/second.

Sample Request Packet

Basic form

Query all call records within 7 days, default query the first 10 items on page 1.

```
{  
  
}
```

Specified condition format

```
{  
  "StartTime":0,           // Default to 7 days before the current time  
  "EndTime":1740531683,    // Default to the current time  
  "CallResult":"NormalEnd",  
  "CallType":"SingleCall",  
  "NumberPerPage":2,  
  "Page":2  
}
```

Request Packet Fields

Field	Type	Required	Description
StartTime	Integer	Optional	Search for the call record start time. Default to 7 days before the current time. If the entered time is less than the default time, the backend will automatically adjust it to the default time
EndTime	Integer	Optional	Search for the call record end time (in seconds). Default to the current time
NumberPerPage	Integer	Optional	Results per page. Default value: 10
Page	Integer	Optional	Query the number of pages. If not filled in, the default is page 1

Sample Response Package Body

Basic form

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-d3d6aa216dcc4cf4a4eee19c4942e740-O-Seq-2556133",
  "Response": {
    "TotalNum": 141,
    "Page": 2,
    "CallRecordList": [
      {
        "CallId": "04c9a0ac-8e38-4a19-be45-349c5ce7911b",
        "Caller_Account": "147",
        "MediaType": "Audio",
        "CallType": "SingleCall",
        "StartTime": 1739960242,
        "EndTime": 1739960245,
        "AcceptTime": 1739960244,
        "CallResult": "NormalEnd",
        "CalleeList_Account": [
          "147",
          "369"
        ],
        "RoomId": "roomid-1434",
        "RoomIdType": 2
      },
      {
        "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58",
        "Caller_Account": "3423",
        "MediaType": "Video",
        "CallType": "SingleCall",
        "StartTime": 1739960500,
        "EndTime": 1739960507,
        "AcceptTime": 1739960503,
        "CallResult": "NormalEnd",
        "CalleeList_Account": [
          "3423",
          "3243"
        ],
        "RoomId": "roomid-1434",
        "RoomIdType": 2
      }
    ]
  }
}
```

Response Packet Fields

Field	Type	Description
ErrorCode	Integer	Error code, 0 indicates success, non-zero indicates failure
ErrorInfo	String	Error message
ActionStatus	String	Request processing result. OK: processing successful; FAIL: processing failed
RequestId	String	Unique request ID. It is returned for each request. RequestId is required for locating a problem
TotalNum	Integer	Total number of this query
Page	Integer	When Page > 0, there is data. Incrementing Page by 1 allows you to Continue Request for subsequent data
CallId	String	Call ID
Caller_Account	String	Caller user ID
MediaType	String	Media type: Video video call Audio audio call
CallType	String	Call Type: SingleCall one-to-one call MultiCall group call
StartTime	Integer	Call initiation timestamp (in seconds)
EndTime	Integer	Call end timestamp (in seconds)
AcceptTime	Integer	Call connection timestamp (in seconds)
CallResult	Integer	Call result Cancel : caller cancel the call before connecting Reject : recipient decline the call NotAnswer : recipient timeout before answering NormalEnd : Call connected and ended normally CallBusy : busy line Interrupt : Call interrupted due to reasons such as network issues
CalleeList_Account	Array	Call Member List
RoomId	String	TRTC Room ID

RoomIdType	Integer	RoomId type: <div><div>1</div> digit room number <div>2</div> string Room Number</div>
------------	---------	---

Error Code Description

Unless a network error (for example, 502 error) occurs, the HTTP return code of this API is 200. The actual error code and error information are indicated by ErrorCode and ErrorInfo in the response packet body.

For common error codes (60000 to 79999), see the [Error Code](#) document.

The private error codes of this API are as follows:

Error Code	Description of Meaning
101001	Internal server error, please retry
101050	Call records do not exist

Get Real-Time Call Status

Last updated : 2025-04-17 16:28:40

Feature Description

App admins can get real-time call status use this API.

Warning:

If you use the deprecated interfaces `TUICallKit.call()` or `TUICallKit.groupCall()` to initiate a call, please check the **Deprecated Document** directory. If you have any questions, you can contact: info_rtc@tencent.com.

API Call Description

Sample Request URL

```
https://xxxxxx/v4/call_engine_http_srv/get_call_info?sdkappid=88888888&identifier=a
```

Request Parameters

The following table only lists the parameters involved in modification when calling this API and their descriptions. For more details about the parameters, please see [REST API Introduction](#).

Parameter	Description
xxxxxx	Dedicated domain name corresponding to the country/region where the SDKAppID resides: China: <code>console.tim.qq.com</code> Singapore: <code>adminapisgp.im.qcloud.com</code> Seoul : <code>adminapikr.im.qcloud.com</code>
v4/call_engine_http_srv/get_call_info	Request API
sdkappid	Assigned SDKAppID in the Chat console when creating an application
identifier	Must be an App administrator account. For more details, see App Administrator .
usersig	Generated signature of the App administrator account. For specific operations, see Generate UserSig

random	Enter a random 32-bit unsigned integer in the range of [0,4294967295]
contenttype	The request format has a fixed value of <code>json</code>

Maximum Calling Frequency

200 times/second.

Sample Request Packet

Basic form

```
{
  "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58"
}
```

Request Packet Fields

Field	Type	Required	Description
CallId	String	Required	Call ID

Sample Response Package Body

Basic form

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-431454f25a44462d8155bdf4fed38cc-0-Seq-19029769",
  "Response": {
    "CallInfo": {
      "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58",
      "MediaType": "Audio",
      "ChatGroupId": "",
      "RoomId": "",
      "RoomIdType": 0
    },
    "UserList": [{
      "User_Account": "user1",
      "Status": "Calling"
    },
    {
      "User_Account": "user2",
```

```
        "Status": "Waiting"
      }
    ]
  }
}
```

Response Packet Fields

Field	Type	Description
ErrorCode	Integer	Error code: 0 indicates success, non-zero indicates failure
ErrorInfo	String	Error message
ActionStatus	String	Request processing result. OK: processing successful; FAIL: processing failed
RequestId	String	Unique request ID. It is returned for each request. RequestId is required for locating a problem
CallId	String	Call ID
MediaType	String	Media type: Video video call Audio audio call
ChatGroupId	String	IM group ID
RoomId	String	TRTC Room ID
RoomIdType	Integer	TRTC room ID type: 1 digit room number 2 string Room Number
UserList	Array	Call Member List
User_Account	String	ID of the calling user
Status	String	Call status Calling on a call Waiting waiting to connect

Error Code Description

Unless a network error (for example, 502 error) occurs, the HTTP return code of this API is always 200. The actual error code and error information are represented by `ErrorCode` and `ErrorInfo` in the response packet body.

For common error codes (60000 to 79999), see the [Error Code](#) document.

The private error codes of this API are as follows:

Error Code	Description
101001	Internal server error, please retry.
101002	Invalid parameter. Check whether the request is correct according to the error description.
101004	The call does not exist, or it once existed but has now ended.

Third-Party Callback

Third-Party Callback Overview

Last updated : 2025-04-17 16:28:40

For convenience in controlling the functional form of your App, CallKit provides callback capability.

Feature Description

Users can configure a callback for a specified Url via the REST API method. When the executed CallbackCommand is in the configuration list, the callback will be triggered.

Must-Knows

Only one callback can be configured for a sdkAppId.

Ensure that the callback URL can be accessed normally.

Warning:

If you use the deprecated interfaces `TUICallKit.call()` or `TUICallKit.groupCall()` to initiate a call, please check the **Deprecated Document** directory. If you have any questions, you can contact: info_rtc@tencent.com.

Callback Command List

Last updated : 2025-04-14 15:11:04

Call Status

Callback Type	Callback Command Word
Callback after starting a call	Call.CallbackAfterStartCall
Callback after call ends	Call.CallbackAfterEndCall
Callback after member changed	Call.CallbackAfterMemberChanged

Callback Configuration

Query Callback Configuration

Last updated : 2025-04-17 16:28:40

Feature Description

App admins can query the callback configuration through this API.

Warning:

If you use the deprecated interfaces `TUICallKit.call()` or `TUICallKit.groupCall()` to initiate a call, please check the **Deprecated Document** directory. If you have any questions, you can contact: info_rtc@tencent.com.

API Call Description

Sample Request URL

```
https://console.tim.qq.com/v4/call_config/get_callback?sdkappid=88888888&identifier
```

Request Parameters

The following table only lists the parameters involved in modification when calling this API and their descriptions. For more parameter details, please see [REST API Introduction](#).

Parameter	Description
xxxxxx	Dedicated domain name corresponding to the country/region where the SDKAppID resides: China: <code>console.tim.qq.com</code> Singapore: <code>adminapisgp.im.qcloud.com</code> Seoul : <code>adminapikr.im.qcloud.com</code>
v4/call_record_http_srv/get_record_by_callid	Request API
sdkappid	Assigned SDKAppID in the Chat console when creating an application
identifier	Must be an App administrator account. For more details, see App Administrator

usersig	Generated signature of the App administrator account. For specific operations, see Generate UserSig
random	Enter a random 32-bit unsigned integer. The value ranges from 0 to 4294967295
contenttype	The request format has a fixed value of <code>json</code>

Maximum Calling Frequency

200 times/second.

Sample Request Packet

Basic form

```
{
}
```

Request Packet Fields

No.

Sample Response Package Body

Basic form

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-1cc8828fd3d84795ac866ced43b15b5c-O-Seq-61309",
  "Response": {
    "Url": "http://www.example.com/callback",
    "CallbackCommandList": [
      "Call.CallbackAfterStartCall",
      "Call.CallbackAfterEndCall"
    ]
  }
}
```

Response Packet Fields

Field	Type	Description
ErrorCode	Integer	Error code, 0 indicates success, non-zero indicates failure

ErrorInfo	String	Error message
ActionStatus	String	Request processing result. OK: successful processing; FAIL: failure
RequestId	String	Unique request ID. It is returned for each request. RequestId is required for locating a problem
Url	String	The callback address must start with <code>http/https</code> . It is recommend using the more secure <code>https</code>
CallbackCommandList	Array	The command word to trigger the callback, see Callback Command Word List

Error Code Description

Unless a network error (for example, 502 error) occurs, the HTTP return code of this API is always 200. The actual error code and error information are indicated by ErrorCode and ErrorInfo in the response packet body.

For common error codes (60000 to 79999), see the [Error Code](#) document.

The private error codes of this API are as follows:

Error Code	Description of Meaning
100001	Server internal error, please retry
100002	Invalid parameter. Check whether the request is correct according to the error description
100301	The callback configuration does not exist. You can use the create callback configuration interface to create it

Setting Callback Configuration

Last updated : 2025-04-17 16:28:40

Feature Description

App admins can set callback configurations through this API. If it is not the first time to set the configuration, the new configuration will overwrite the old one.

Warning:

If you use the deprecated interfaces `TUICallKit.call()` or `TUICallKit.groupCall()` to initiate a call, please check the **Deprecated Document** directory. If you have any questions, you can contact: info_rtc@tencent.com.

API Call Description

Sample Request URL

```
https://xxxxxx/v4/call_config/set_callback?sdkappid=88888888&identifier=admin&usersig
```

Request Parameters

The following table only lists the parameters involved in modification when calling this API and their descriptions. For more parameter details, please see [REST API Introduction](#).

Parameter	Description
xxxxxx	Dedicated domain name corresponding to the country/region where the SDKAppID resides: China: <code>console.tim.qq.com</code> Singapore: <code>adminapisgp.im.qcloud.com</code> Seoul : <code>adminapikr.im.qcloud.com</code>
v4/call_record_http_srv/get_record_by_callid	Request API
sdkappid	Assigned SDKAppID in the Chat console when creating an application
identifier	Must be an App administrator account. For more details, see App Administrator
usersig	Generated signature of the App administrator account. For

	specific operations, see Generate UserSig
random	Enter a random 32-bit unsigned integer. The value ranges from 0 to 4294967295
contenttype	The request format has a fixed value of <code>json</code>

Maximum Calling Frequency

200 times per second.

Sample Request Packet

Basic form

```
{
  "Url": "http://www.example.com/callback",
  "CallbackCommandList": [
    "Call.CallbackAfterStartCall",
    "Call.CallbackAfterEndCall",
    "Call.CallbackAfterMemberChanged"
  ]
}
```

Request Packet Fields

Field	Type	Required	Description
Url	String	Required	The callback address should start with <code>http/https</code> . It is recommend using the more secure <code>https</code>
CallbackCommandList	Array	Required	The command words that can trigger the callback, see Callback Command Word List

Response Package Example

Basic form

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-O-Seq-52720"
}
```

Response Packet Fields

Field	Type	Description
ActionStatus	String	Request processing result. OK indicates successful processing. FAIL indicates failure
ErrorInfo	String	Error message
ErrorCode	Integer	Error code, 0 indicates success, non-zero indicates failure
RequestId	String	Unique request ID. It will be returned for each request. RequestId is required for locating a problem

Error Code Description

Unless a network error (for example, 502 error) occurs, the HTTP return code of this API is 200. The actual error code and error information are indicated by ErrorCode and ErrorInfo in the response packet body.

For common error codes (60000 to 79999), see the [Error Code](#) document.

The private error codes of this API are as follows:

Error Code	Description of Meaning
100001	Server internal error. Please retry
100002	Invalid parameter. Check whether the request is correct according to the error description

Deleting Callback Configuration

Last updated : 2025-04-17 16:28:40

Feature Description

App admins can delete callback configurations use this API.

Warning:

If you use the deprecated interfaces `TUICallKit.call()` or `TUICallKit.groupCall()` to initiate a call, please check the **Deprecated Document** directory. If you have any questions, you can contact: info_rtc@tencent.com.

API Call Description

Sample Request URL

```
https://console.tim.qq.com/v4/call_config/delete_callback?sdkappid=88888888&identif
```

Request Parameters

The following table only lists the parameters involved in modification when calling this API and their descriptions. For more parameter details, refer to [REST API Introduction](#).

Parameter	Description
xxxxxx	Exclusive domain name corresponding to the country/region where the SDKAppID resides: China: <code>console.tim.qq.com</code> Singapore: <code>adminapisgp.im.qcloud.com</code> Seoul : <code>adminapikr.im.qcloud.com</code>
v4/call_config/delete_callback	Request API
sdkappid	Assigned SDKAppID in the Chat console when creating an application
identifier	Must be an App administrator account. For more details, see App Administrator
usersig	Generated signature of the App administrator account. For specific operations, see Generate UserSig
random	Enter a random 32-bit unsigned integer in the range of [0,4294967295]

contenttype	The request format has a fixed value of <code>json</code>
-------------	---

Maximum Calling Frequency

200 times/second.

Sample Request Packet

Basic form

```
{  
}
```

Request Packet Fields

No.

Sample Response Packet Body

Basic form

```
{  
  "ErrorCode": 0,  
  "ErrorInfo": "",  
  "ActionStatus": "OK",  
  "RequestId": "Id-1cc8828fd3d84795ac866ced43b15b5c-O-Seq-61309"  
}
```

Response Packet Fields

Field	Type	Description
ActionStatus	String	Request processing result. OK: processing successful; FAIL: processing failed
ErrorInfo	String	Error message
ErrorCode	Integer	Error code, 0 indicates success, non-zero indicates failure
RequestId	String	Unique request ID. It is returned for each request. RequestId is required for locating a problem

Error Code Description

Unless a network error (for example, 502 error) occurs, the HTTP return code of this API is always 200. The actual error code and error information are indicated by ErrorCode and ErrorInfo in the response packet body.

For common error codes (60000 to 79999), see the [Error Code](#) document.

The private error codes of this API are as follows:

Error Code	Description of Meaning
100001	Internal server error, please retry
100002	Invalid parameter. Check whether the request is correct according to the error description

Call Status Callback

Callback after Call Initiation

Last updated : 2025-04-14 15:11:04

Feature Description

The App backend can view the start dynamics of a call in real time through this callback.

Must-Knows

To enable the callback, you must configure the callback URL through the REST API and turn on the switch corresponding to this callback protocol. For the configuration method, see [Set Callback Configuration](#).

The direction of the callback is that the Call backend initiates an HTTP POST request to the App backend.

After receiving a callback request, the App backend needs to verify whether the value of the SDKAppID parameter in the request URL is its own SDKAppID.

Callback Triggering Scenarios

App user initiates a successful call through the client.

Callback Triggering Timing

After call initiation.

API Description

Sample Request URL

In the following example, the callback URL configured for the App is `https://www.example.com`.

Example:

```
https://www.example.com?  
SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&contenttype=json&ClientIP=$  
ClientIP&OptPlatform=$OptPlatform
```

Request Parameters

Parameter	Description
https	The request protocol is HTTPS, and the request method is POST
www.example.com	Callback URL
SdkAppid	Assigned SDKAppID in the Chat console when creating an application
CallbackCommand	Fixed as <code>Call.CallbackAfterStartCall</code>
contenttype	The fixed value is <code>json</code>
ClientIP	Client IP address, in the format of <code>127.0.0.1</code>
OptPlatform	Client platform. For the parameter meaning, see OptPlatform in Third-Party Callback Introduction: Callback Protocol

Sample Request Packet

```
{
  "CallbackCommand": "Call.CallbackAfterStartCall",
  "CallInfo": {
    "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58",
    "CallMediaType": "Audio",
    "ChatGroupId": "",
    "RoomId": "",
    "RoomIdType": 1
  },
  "UserList": [
    {
      "User_Account": 144115212830834855,
      "Status": "Calling"
    },
    {
      "User_Account": 144115212948889925,
      "Status": "Waiting"
    }
  ],
  "EventTime": 1740464128807
}
```

Request Packet Fields

--	--	--

Field	Type	Description
CallbackCommand	String	Callback command
CallId	String	Call ID
MediaType	String	Media type: Video video call Audio audio call
ChatGroupId	String	Chat Group ID
RoomId	String	TRTC Room ID
RoomIdType	Integer	TRTC room ID type: 1 digit room number 2 string Room Number
UserList	Array	Call Member List
User_Account	String	Calling User ID
Status	String	Call status Calling in a call Waiting waiting to connect
EventTime	Integer	Event-triggered millisecond timestamp

Sample Response Packet

The response packet is returned after the App backend synchronizes the data.

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0 // Ignore callback result
}
```

Response Packet Fields

Field	Type	Required	Description
ActionStatus	String	Required	Request processing result. OK indicates successful processing; FAIL indicates failure
ErrorInfo	String	Required	Error message

ErrorCode	Integer	Required	Error code. Enter 0 here to ignore the callback result
-----------	---------	----------	--

Callback after Call Ends

Last updated : 2025-04-17 16:33:08

Feature Description

The App backend can monitor in real-time the dynamics of call ending through this callback. Five seconds after call termination, call details can be queried through the call record inquiry API.

Must-Knows

To enable the callback, you must configure the callback URL through the REST API and turn on the switch corresponding to this callback protocol. For the configuration method, see [Set Callback Configuration](#).

The direction of the callback is that the Call backend initiates an HTTP POST request to the App backend.

After receiving a callback request, the App backend needs to verify whether the value of the SDKAppID parameter in the request URL is its own SDKAppID.

Callback Triggering Scenarios

Call termination.

Callback Triggering Timing

When the number of call participants is less than or equal to 1, trigger call termination and reclaim.

API Description

Sample Request URL

In the following example, the callback URL configured for the App is `https://www.example.com`.

Example:

```
https://www.example.com?  
SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&contenttype=json&ClientIP=$  
ClientIP&OptPlatform=$OptPlatform
```


Request Parameters

Parameter	Description
https	The request protocol is HTTPS, and the request method is POST
www.example.com	Callback URL
SdkAppid	Assigned SDKAppID in the Chat console when creating an application
CallbackCommand	Fixed as <code>Call.CallbackAfterEndCall</code>
contentType	The value is fixed as <code>json</code>
ClientIP	Client IP address, in the format of <code>127.0.0.1</code>
OptPlatform	Client platform. For the parameter values, see the description of the OptPlatform parameter in Third-Party Callback Introduction: Callback Protocol

Sample Request Packet

```
{
  "CallbackCommand": "Call.CallbackAfterEndCall",
  "CallRecord": {
    "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58",
    "Caller_Account": "10001",
    "MediaType": "Audio",
    "CallType": "SingleCall",
    "StartTime": 1741231146,
    "EndTime": 1741231296,
    "AcceptTime": 0,
    "CallResult": "NotAnswer",
    "CalleeList_Account": ["10001", "user2"],
    "RoomId": "roomid-1434",
    "RoomIdType": 2
  },
  "EventTime": 1703589922780
}
```

Request Packet Fields

Field	Type	Description
CallbackCommand	String	Callback command
CallRecord	Struct	Call record information

CallId	String	Call ID
Caller_Account	String	Caller ID
MediaType	String	Media type: Video video call Audio audio call
CallType	String	Call type: SingleCall one-to-one call MultiCall group call
StartTime	Integer	Call initiation timestamp (in seconds)
EndTime	Integer	Call end timestamp (in seconds)
AcceptTime	Integer	Call connection timestamp (in seconds)
CallResult	Integer	Call result Cancel : Caller cancels the call before connecting. Reject : recipient decline the call NotAnswer : recipient timeout before answering NormalEnd : Call connected and ended normally CallBusy : call busy Interrupt : Call interrupted due to reasons such as network issues Offline : The server detects heartbeat expiry, causing the call to end.
CalleeList_Account	Array	Call Member List
RoomId	String	TRTC Room ID
RoomIdType	Integer	RoomId type: 1 Digit room number 2 String room number
EventTime	Integer	Event-triggered timestamp in milliseconds

Sample Response Packet

The response packet is returned after the App backend synchronizes the data.

```
{  
  "ActionStatus": "OK",  
  "ErrorInfo": "",  
  "ErrorCode": 0 // Ignore the callback result.
```

```
}

```

Response Packet Fields

Field	Type	Required	Description
ActionStatus	String	Required	Request processing result. OK: successful processing; FAIL: processing failed
ErrorInfo	String	Required	Error message
ErrorCode	Integer	Required	Error code. Enter 0 here to ignore the callback result

Callback after Member Changed

Last updated : 2025-04-14 15:05:10

Feature Description

The App backend can view call member status changes in real time through this callback.

Must-Knows

To enable the callback, you must configure the callback URL via the REST API and turn on the switch corresponding to this callback protocol. For the configuration method, see [Callback Configuration Settings](#).

The direction of the callback is that the Call backend initiates an HTTP POST request to the App backend.

After receiving a callback request, the App backend needs to verify whether the value of the SDKAppID parameter in the request URL is its own SDKAppID.

Callback Triggering Scenarios

Status changes of members caused by operations such as connect, hang up, etc. performed by App users through the client will trigger this callback.

Callback Triggering Timing

After the call member status changes.

API Description

Sample Request URL

In the following example, the callback URL configured for the App is `https://www.example.com`.

Example:

```
https://www.example.com?
SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&contenttype=json&ClientIP=$
ClientIP&OptPlatform=$OptPlatform
```

Request Parameters

Parameters	Overview
https	The request protocol is HTTPS, and the request method is POST.
www.example.com	Callback URL
SdkAppid	Assigned SDKAppID in the Chat console when creating an application
CallbackCommand	Fixed as <code>Call.CallbackAfterMemberChanged</code>
contentType	The value is fixed as <code>json</code>
ClientIP	Client IP address, in the format of <code>127.0.0.1</code>
OptPlatform	Client platform. For the parameter values, see the description of the OptPlatform parameter in the Third-Party Callback Overview: Callback Protocol

Sample Request Packet

```
{
  "CallbackCommand": "Call.CallbackAfterMemberChanged",
  "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58",
  "UserList": [
    {
      "User_Account": "user1",
      "Status": "Calling"
    },
    {
      "User_Account": "user2",
      "Status": "Calling"
    }
  ],
  "ChangedUserList": [
    {
      "User_Account": "user2",
      "ActionType": "Answer" // Answer: Answer; Reject: Reject;
NotAnswer: Unanswered; Hangup: Hang up; Join: Proactively Join; Offline:
Offline (heartbeat expiry); Invite: Invite; Cancel_Invite: Cancel invitation;
    }
  ]
  "EventTime": 1740464128807
}
```

Request Packet Fields

Field	Type	Overview
CallbackCommand	String	Callback command
CallId	String	Call ID
UserList	Array	Call Member List
User_Account	String	Calling User ID
Status	String	Call status: Calling : on a call Waiting : waiting for connection
ChangedUserList	Array	Member list
ChangedUserList.User_Account	String	User ID of the member whose status has changed
ActionType	String	Change operation that causes changes Accept : Answer Reject : Deny NotAnswer : Unanswered Hang Up : Hang up Join : Join in Call Offline : Offline Invited : Invite to join a call by caller Canncel_Invite : Cancel Invite by caller
EventTime	Integer	Event-triggered timestamp in milliseconds

Sample Response Packet

The response packet is returned after the App backend synchronizes the data.

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0 // Ignore callback result
}
```

Response Packet Fields

Field	Type	Required	Overview
ActionStatus	String	Required	Request processing result. OK: processing

			successful; FAIL: processing failed.
ErrorInfo	String	Required	Error message.
ErrorCode	Integer	Required	Error code. Enter 0 here to ignore the callback result.

References

[Webhook Overview](#)

REST API: [Retrieve real-time call status](#)

Legacy Documentation

Call Status Callback

Call Status Callback

Last updated : 2025-03-28 11:19:05

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=$command&contenttype=json&clientip=$clientip&optplat  
form=$optplatform
```

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&optplatfo  
rm=iOS HTTP/1.1  
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=$command&contenttype=json&clientip=$clientip&optplat  
form=$optplatform
```

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=88888888&command=caller_start_call&contenttype=json&clientip=127.0.0.1&
optplatform=iOS HTTP/1.1
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:
		<code>start_call</code> Caller initiates call <code>call_accepted</code> Caller answers call <code>call_missed</code> Caller missed call <code>call_rejected</code> Caller rejects call <code>call_busy</code> Caller line busy <code>cancel_call</code> Caller cancels call <code>call_failed</code> Caller failed to initiate call <code>call_end</code> Caller call ended normally <code>call_interrupted</code> Caller call interrupted <code>receive_call</code> Callee receives call <code>accept_call</code> Callee answers call <code>not_answer_call</code> Callee does not answer <code>reject_call</code> Callee rejects call <code>ignore_call</code> Called party ignores call <code>call_canceled</code> Called party cancels call <code>call_end</code> Normal termination of called party's call <code>call_interrupted</code> Called party's call abnormally interrupted <code>invite_user</code> Midway invite user <code>join_in_group_call</code> Join the call midway
StartCallTs	Integer	Timestamp when call is initiated (second-level) only returned on <code>normal_end</code>
AcceptTs	Integer	Timestamp when call is answered (second-level) only returned on <code>normal_end</code>
EndTs	Integer	Timestamp when call ends (second-level) only returned on <code>normal_end</code>

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
<div><div>cancel</div>Cancel: Caller cancels the call before it is answered</div> <div><div>reject</div>Declined: The callee rejects the call</div> <div><div>not_answer</div>Missed call: The callee does not answer within the timeout period</div> <div><div>normal_end</div>Complete: The call is connected and ends normally</div> <div><div>call_busy</div>Busy Line: The call is on a busy line</div> <div><div>interrupt</div>Interruption: The call is interrupted due to network or other reasons</div>	<div><div>caller_start_call</div>Originator Initiates Call</div> <div><div>caller_call_accepted</div>Originator Answers Call</div> <div><div>caller_call_missed</div>Originator Misses Call</div> <div><div>caller_call_rejected</div>Caller Rejects Call</div> <div><div>caller_call_busy</div>Caller Line is Busy</div> <div><div>caller_cancel_call</div>Caller Cancels Call</div> <div><div>caller_call_failed</div>Caller Fails to Initiate Call</div> <div><div>caller_call_end</div>Caller Ends Call Normally</div> <div><div>caller_call_interrupted</div>Caller's Call Interrupted</div> <div><div>callee_receive_call</div>Callee Receives Call</div> <div><div>callee_accept_call</div>Callee Answers Call</div> <div><div>callee_not_answer_call</div>Callee Does Not Answer</div> <div><div>callee_reject_call</div>Callee Rejects Call</div> <div><div>callee_ignore_call</div>Callee Ignores Call</div> <div><div>callee_call_canceled</div>Callee Cancels Call</div> <div><div>callee_call_end</div>Callee Ends Call Normally</div> <div><div>callee_call_interrupted</div>Callee's Call Interrupted</div> <div><div>invite_user</div>Midway invite user</div>

	<code>join_in_group_call</code> Join the call midway
--	--

Establishing Callback Configuration

Last updated : 2024-10-29 17:23:23

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=administrator&usersig=x
```

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: <code>callkit-intl.trtc.tencent-cloud.com</code>
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 <code>json</code>

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 : 2024-10-29 17:23:23

Feature Overview

Administrators can query configured callbacks through this interface.

API description

Sample request URL

```
https://xxxxxx/v1/callback/get?
sdkappid=88888888&identifier=administrator&usersig=xxx&random=99999999&contentt
ype=json
```

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 : 2024-10-29 17:23:23

Feature Overview

Administrators can update callbacks through this interface.

API description

Sample request URL

```
https://xxxxxx/v1/callback/update?
sdkappid=88888888&identifier=administator&usersig=xxx&random=99999999&contentt
ype=json
```

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 : 2024-10-29 17:23:23

Feature Overview

The administrator can delete the callback through this interface.

API description

Sample request URL

```
https://xxxxxx/v1/callback/delete?
sdkappid=88888888&identifier=administrator&usersig=xxx&random=99999999&contentt
ype=json
```

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

Call Records

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 (SDKAppld) 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=$identifier&usersig=$usersig&random=99999999&cont  
enttype=json
```

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	<code>callkit-intl.trtc.tencent-cloud.com</code>
version	Protocol Version Number	Fixed as <code>v1</code>
kind	Management Classification	Example: <code>v1/records/get_record_by_callId</code> , where `records` is a kind
command	The word `command`, combined with `kind`, is used to indicate a specific business feature	Example: <code>v1/records/get_record_by_callId</code> , 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
<code>errorCode</code>	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=888888888&random=999999999&contenttype=json
HTTP/2
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": 888888888,
    "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 : 2024-10-29 17:23:23

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=adminis
```

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 <code>json</code>
-------------	---

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 <code>video</code> Video Call <code>audio</code> 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: <code>single</code> One-on-one call <code>group</code> Group Call
callResult	String	Call Result: <code>cancel</code> Cancel: The caller cancelled the call before connection.

		<div><div>reject</div> Declined call: The callee rejected.</div> <div><div>not_answer</div> Not Answered: The callee did not answer in time.</div> <div><div>normal_end</div> Completed: The call was connected and ended normally.</div> <div><div>call_busy</div> Busy: The line was busy during the call.</div> <div><div>interrupt</div> Interrupts: The call was interrupted due to network or other reasons.</div>
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 : 2024-10-29 17:23:23

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=admini
```

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 <code>video</code> Video Call <code>audio</code> 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 <code>single</code> One-on-one call <code>group</code> Group Call

callResult	String	Call Result: <div><div>cancel</div> Cancel: The caller cancelled the call before connection.</div> <div><div>reject</div> Declined call: The callee rejected.</div> <div><div>not_answer</div> Not Answered: The callee did not answer in time.</div> <div><div>normal_end</div> Completed: The call was connected and ended normally.</div> <div><div>call_busy</div> Busy: The line was busy during the call.</div> <div><div>interrupt</div> Interrupts: The call was interrupted due to network or other reasons.</div>
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

End Calls

Last updated : 2024-08-30 13:38:36

This document introduces how to end calls on the server side during audio and video calls for better management of call duration and enhanced service experience in various scenarios like medical consultations and 1V1 social interactions.

API description

API request domain: **trtc.tencentcloudapi.com**.

API description: Remove all users from the audio and video room to end the call.

Default API request frequency limit: **20 times/sec**.

Sample request

```
https://trtc.tencentcloudapi.com/?Action=DismissRoom
&SdkAppId=1400000001
&RoomId=1234
&<Common request parameters>
```

The following table only describes the modified parameters when this API is called. For more details, see: [TRTC API Request](#).

Parameter	Required	Type	Description
trtc.tencentcloudapi.com	Yes	String	Request API
Action	Yes	String	Common Parameters . For this API, the value is `DismissRoom`.
Version	Yes	String	Common Parameters , the value for this API is: 2019-07-22.
Region	Yes	String	Common Parameters . See the supported Region List . This API supports only: ap-beijing, ap-guangzhou, ap-mumbai.
SdkAppId	Yes	Integer	The SDKAppId for the call. Example value: 1400188366

RoomId	Yes	Integer	Audio and Video Room Number.
Common request parameters	Yes	/	For details, see: Common Parameters

Note:

To dissolve a room with a string Room Number, set Action to: DismissRoomByStrRoomId, with RoomId of type: String.

Sample response

```
{
  "Response": {
    "RequestId": "eac6b301-a322-493a-8e36-83b295459397"
  }
}
```

Parameter name	Type	Description
RequestId	String	Unique Request ID, generated by the server-side, is returned with each request (if a request fails to reach the server-side for other reasons, it will not receive a RequestId). Provide the RequestId of the request when troubleshooting.

Debug API

API Explorer : [API Explorer](#).

API Explorer makes it easy to make online API calls, verify signatures, generate SDK code, search for APIs, etc. You can also use it to query the content of each request as well as its response.

Error code

The error codes listed below are only relevant to the API business logic. For more error codes, refer to [Common Error Codes](#).

Error code	Description
FailedOperation.RoomNotExist	Room does not exist.
InternalError	An internal error occurs.

InternalError.GetRoomCacheIpError	Failed to inquire room.
InvalidParameter.RoomId	RoomId parameter error.
InvalidParameter.SdkAppId	SdkAppId parameter error.
InvalidParameterValue.RoomId	Incorrect RoomId value.
MissingParameter.RoomId	Missing RoomId parameter.
MissingParameter.SdkAppId	Missing SdkAppId parameter.
UnauthorizedOperation.SdkAppId	No permissions to operate SdkAppId.

Client APIs (TUICallKit)

Android

API Overview

Last updated : 2025-04-30 18:56:39

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.
calls	Initiate a one-to-one or multi-person call.
join	Proactively join a 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 <code>TUICallEngine</code> instance (singleton).
destroyInstance	Destroy <code>TUICallEngine</code> instance (singleton).
init	Authenticates the basic audio/video call capabilities.

addObserver	Add listener.
removeObserver	Remove listener.
calls	Initiate a one-to-one or multi-person call.
join	Proactively join a call.
accept	Accept call.
reject	Reject call.
hangup	Hang up call.
ignore	Ignore call.
inviteUser	Invite users to the current 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 <code>TUICallEngine</code> (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
<code>onError</code>	An error occurred during the call.
<code>onCallReceived</code>	A call was received.
<code>onCallCancelled</code>	The call was canceled.
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user has a video stream.
<code>onUserAudioAvailable</code>	Whether a user has an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user was kicked offline.
<code>onUserSigExpired</code>	The user sig is expired.

Definitions of Key Types

API	Description
<code>TUICallDefine.MediaType</code>	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 camera and Back camera.
TUICommonDefine.AudioPlaybackDevice	The audio playback device type. Enumeration type: Earpiece and Speakerphone.
TUICommonDefine.NetworkQualityInfo	The current network quality.

TUICallKit

Last updated : 2025-04-30 18:56:39

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 <code>TUICallKit</code> instance (singleton mode).
setSelfInfo	Sets the alias and profile photo.
calls	Initiate a one-to-one or multi-person call.
join	Proactively join a 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
userIdList	List<String>	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

calls

Initiate a one-to-one or multi-person call.

kotlin

Java

```
fun calls(
    userIdList: List<String?>?, mediaType: TUICallDefine.MediaType?,
    params: TUICallDefine.CallParams?, callback: TUICommonDefine.Callback?
)

void calls(List<String> userIdList, TUICallDefine.MediaType mediaType,
    TUICallDefine.CallParams params, TUICommonDefine.Callback callback)
```

The parameters are described below:

Parameter	Type	Description
userIdList	List<String>	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
--	-----------------------

join

Proactively join a call.

Kotlin

Java

```
fun join(callId: String?, callback: TUICommonDefine.Callback?)

void join(String callId, TUICommonDefine.Callback callback)
```

The parameters are described below:

Parameter	Type	Description
callId	String	Unique ID for this call

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)
```

Deprecated Interface

call

This API is used to make a (one-to-one) call. **Note: it is recommended to use the calls API.**

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. **Note: it is recommended to use the calls API.**

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: it is recommended to use the calls API.**

Note:

Before making a group call, you need to create an Chat group first.

For details about how to create a group, see [Chat Group Management](#), or you can use [Chat UIKit](#) to integrate chat, call and other scenarios.

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: it is recommended to use the calls API.**

Note:

Before making a group call, you need to create an Chat group first.

For details about how to create a group, see [Chat Group Management](#), or you can use [Chat UIKit](#) to integrate chat, call and other scenarios.

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 call
    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: it is recommended to use the join API.**

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.

For details about how to create a group, see [Chat Group Management](#), or you can use [Chat UIKit](#) to integrate chat, call and other scenarios.

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.

TUICallEngine

Last updated : 2025-04-30 18:56:40

TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

Overview

API	Description
<code>createInstance</code>	Creates a <code>TUICallEngine</code> instance (singleton mode).
<code>destroyInstance</code>	Terminates a <code>TUICallEngine</code> instance (singleton mode).
<code>Init</code>	Authenticates the basic audio/video call capabilities.
<code>addObserver</code>	Registers an event listener.
<code>removeObserver</code>	Unregisters an event listener.
<code>calls</code>	Initiate a one-to-one or multi-person call.
<code>join</code>	Proactively join a call.
<code>accept</code>	Accepts a call.
<code>reject</code>	Rejects a call.
<code>hangup</code>	Ends a call.
<code>ignore</code>	Ignores a call.
<code>inviteUser</code>	Invites users to the current group call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.
<code>openCamera</code>	Turns the camera on.
<code>closeCamera</code>	Turns the camera off.

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 <code>TUICallEngine</code> (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 <code>SDKAppID</code> 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. <code>onSuccess</code> indicates initialization is successful.

addObserver

This API is used to register an event listener to listen for `TUICallObserver` events.

```
void addObserver(TUICallObserver observer);
```

removeObserver

This API is used to unregister an event listener.

```
void removeObserver(TUICallObserver observer);
```

calls

Initiate a one-to-one or multi-person call.

```
void calls(List<String> userIdList, TUICallDefine.MediaType mediaType,  
          TUICallDefine.CallParams params, TUICommonDefine.Callback callback)
```

The parameters are described below:

Parameter	Type	Description
userIdList	List<String>	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

join

Proactively join a call.

```
void join(String callId, TUICommonDefine.Callback callback);
```

The parameters are described below:

Parameter	Type	Description
callId	String	Unique ID for this call

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.

Deprecated Interface

call

This API is used to make a (one-to-one) call. **Note: it is recommended to use the calls API.**

```
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: it is recommended to use the calls API.**

Note :

Before making a group call, you need to create an Chat group first.

```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType callMediaType, 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

joinInGroupCall

This API is used to join a group call. **Note: it is recommended to use the join API.**

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.Callback callback);
```

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.

TUICallObserver

Last updated : 2025-04-30 18:56:40

TUICallObserver APIs

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

Overview

API	Description
<code>onError</code>	A call occurred during the call.
<code>onCallReceived</code>	A call invitation was received.
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallNotConnected</code>	The call not connected.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserInviting</code>	A user is invited to join a call.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user had a video stream.
<code>onUserAudioAvailable</code>	Whether a user had an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user was kicked offline.

`onUserSigExpired`

The user sig is expired.

Details

onError

An error occurred.

Note:

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.

```
void onError(int code, String 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.

```
void onCallReceived(String callId, String callerId, List<String> calleeIdList,  
                    TUICallDefine.MediaType mediaType, TUICallDefine.CallObserverEx
```

The parameters are described below:

Parameter	Type	Description
callId	String	Unique ID of this call
callerId	String	Caller ID
calleeIdList	List<String>	Callee ID List
mediaType	TUICallDefine.MediaType	The call type, which can be video or audio.
info	TUICallDefine.CallObserverExtraInfo	Extended information

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(String callId, TUICallDefine.MediaType mediaType, TUICallDefine.CallObserverExtraInfo info)
```

The parameters are described below:

Parameter	Type	Description
callId	String	Unique ID of this call
mediaType	TUICallDefine.MediaType	The call type, which can be video or audio.
info	TUICallDefine.CallObserverExtraInfo	Extended information

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(String callId, TUICallDefine.MediaType mediaType, TUICallDefine.CallEndReason reason, String userId, long totalTime, TUICallDefine.CallObserverExtraInfo info)
```

The parameters are described below:

Parameter	Type	Description
callId	String	Unique ID of this call
mediaType	TUICallDefine.MediaType	The call type, which can be video or audio.
reason	TUICallDefine.CallEndReason	Call end reason
userId	String	Ended by user ID
totalTime	long	The call duration.
info	TUICallDefine.CallObserverExtraInfo	Extended information

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.

onCallNotConnected

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 [onCallNotConnected](#) callbacks (userId is his own ID); the callee receives the [onCallNotConnected](#) callback (userId is his own ID)

Callee rejection: The caller will simultaneously receive the [onUserReject](#) and [onCallNotConnected](#) callbacks (userId is his own ID); the callee receives the [onCallNotConnected](#) callback (userId is his own ID)

Callee busy: The caller will simultaneously receive the [onUserLineBusy](#) and [onCallNotConnected](#) 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 onCallNotConnected(String callId, TUICallDefine.MediaType mediaType, TUICallDe
String userId, TUICallDefine.CallObserverExtraInfo info);
```

The parameters are described below:

Parameter	Type	Description
callId	String	Unique ID of this call
mediaType	TUICallDefine.MediaType	The call type, which can be video or audio.
reason	TUICallDefine.CallEndReason	Call end reason
userId	String	Not connected by user ID
info	TUICallDefine.CallObserverExtraInfo	Extended information

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.

onUserInviting

A user is invited to join a call.

```
void onUserInviting(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the invitee.

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	The volume table, which includes the volume of each user (<code>userId</code>). Value

range: 0-100.

onUserNetworkQualityChanged

The network quality of all users.

```
void onUserNetworkQualityChanged(List<TUICallDefine.NetworkQualityInfo> networkQual
```

The parameters are described below:

Parameter	Type	Description
networkQualityList	List	The current network conditions for all users (<code>userId</code>).

onKickedOffline

The current user was kicked offline: At this time, you can prompt the user with a UI message and then invoke `init` again.

```
void onKickedOffline();
```

onUserSigExpired

The user sig is expired: At this time, you need to generate a new `userSig` , and then invoke `init` again.

```
void onUserSigExpired();
```

Deprecated Interface

onCallCancelled

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.

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.

Type Definition

Last updated : 2025-04-30 18:56:40

Common structures

TUICallDefine

Type	Description
CallParams	An additional parameter.
OfflinePushInfo	Offline push vendor configuration information.
CallObserverExtraInfo	Extended 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
CallEndReason	Call end reason

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
chatGroupId	String	chat group Id.

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

		<p><code>sound = IOS_OFFLINE_PUSH_NO_SOUND</code> , there will be no sound played when the message is received. When <code>sound =</code> <code>IOS_OFFLINE_PUSH_DEFAULT_SOUND</code> , the system sound will be played when the message is received. If you want to customize the iOSSound, you need to link the audio file into the Xcode project first, and then set the audio file name (with extension) to the iOSSound.</p>
androidSound	String	<p>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 <code>setAndroidFCMChannelID</code> to set the channelId for it to take effect.</p>
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 <code>setAndroidVIVOCategory</code> 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

CallObserverExtraInfo

Extended information.

Value	Type	Description
-------	------	-------------

roomId	TUICommonDefine.RoomId	room ID
role	Role	Call role
userData	NSString	Custom extended field when initiating a call.
chatGroupId	NSString	Group ID

RoomId

Room ID for audio and video in a call.

Note:

(1) `intRoomId` and `strRoomId` are mutually exclusive. If you choose to use `strRoomId`, `intRoomId` needs to be filled in as 0. If both are filled in, the SDK will prioritize `intRoomId`.

(2) Do not mix `intRoomId` and `strRoomId` because they are not interchangeable. For example, the number 123 and the string "123" represent two completely different rooms.

Value	Type	Description
<code>intRoomId</code>	int	Numeric room ID. range: 1 - 2147483647($2^{31}-1$)
<code>strRoomId</code>	String	String room ID. range: Limited to 64 bytes in length. The supported character set range is as follows (a total of 89 characters): Lowercase and uppercase English letters. (a-zA-Z) Number (0-9) space , ! , # , \$, % , & , (,) , + , - , : , ; , < , = , . , > , ? , @ , [,] , ^ , _ , { , } , , ~ , ,

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
SINGLE_CALL	0	Group call
GROUP_CALL	1	Anonymous group calling (not supported at this moment, please stay tuned).
MULTI_CALL	2	one to one call

IOSOfflinePushType

iOS offline push type

Type	Value	Description
APNs	0	APNs
VoIP	1	VoIP

CallEndReason

Call end reason

Value	Type	Description
UNKNOWN	0	Unknown
HANGUP	1	Hang up
REJECT	2	Deny
NO_RESPONSE	3	No response
OFFLINE	4	Offline
LINE_BUSY	5	Busy Line

CANCELED	6	Cancel call
OTHER_DEVICE_ACCEPTED	7	Other device answers
OTHER_DEVICE_REJECT	8	Other device denies
END_BY_SERVER	9	Backend ends

AudioPlaybackDevice

Audio route

Type	Value	Description
Speakerphone	0	Earpiece
Earpiece	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
GOOD	3	Poor
BAD	4	Bad
VERY_BAD	5	very bad
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_960_540	110	Aspect ratio: 16:9, resolution: 960x540, recommended bitrate: 850kbps
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 : 2025-04-30 18:56:40

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.
calls	Initiate a one-to-one or multi-person call
join	Proactively join a 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.

<code>removeObserver</code>	Remove listener.
<code>calls</code>	Initiate a one-to-one or multi-person call
<code>join</code>	Proactively join a call
<code>accept</code>	Accept call.
<code>reject</code>	Reject call.
<code>hangup</code>	Hang up call.
<code>ignore</code>	Ignore call.
<code>inviteUser</code>	Invite users to the current call.
<code>switchCallMediaType</code>	Switch the call media type, such as from video call to audio call.
<code>startRemoteView</code>	Subscribe to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribe from the video stream of a remote user.
<code>openCamera</code>	Turn on the camera.
<code>closeCamera</code>	Turn off the camera.
<code>switchCamera</code>	Switch camera.
<code>openMicrophone</code>	Enable microphone.
<code>closeMicrophone</code>	Disable the microphone.
<code>selectAudioPlaybackDevice</code>	Select the audio playback device (Earpiece/Speakerphone).
<code>setSelfInfo</code>	Set the user's profile picture and nickname.
<code>enableMultiDeviceAbility</code>	Sets whether to enable multi-device login for TUICallEngine (supported by the Group Call package).
<code>setVideoRenderParams</code>	Set the rendering mode of video.
<code>setVideoEncoderParams</code>	Set the encoding parameters of video encoder.
<code>getTRTCCloudInstance</code>	Advanced features.
<code>setBeautyLevel</code>	Set beauty level, support turning off default beauty.

TUICallObserver

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

API	Description
<code>onError</code>	An error occurred during the call.
<code>onCallReceived</code>	A call was received.
<code>onCallCancelled</code>	The call was canceled.
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user has a video stream.
<code>onUserAudioAvailable</code>	Whether a user has an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user was kicked offline.
<code>onUserSigExpired</code>	The user sig is expired.

Definitions of Key Typ

API	Description
<code>TUICallMediaType</code>	Call media type, Enumeration type: Unknown, Video, and Audio.
<code>TUICallRole</code>	Call role, Enumeration type: None, Call, and Called.
<code>TUICallStatus</code>	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 camera and Back camera.
TUIAudioPlaybackDevice	The audio playback device type. Enumeration type: Earpiece and Speakerphone.
TUINetworkQualityInfo	The current network quality.

TUICallKit

Last updated : 2025-04-30 18:56:40

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.
calls	Initiate a one-to-one or multi-person call
join	Proactively join a 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.

calls

Initiate a call.

Swift

Objective-C

```
func calls(userIdList: [String], callMediaType: TUICallMediaType, params: TUICallPa
        succ: @escaping TUICallSucc, fail: @escaping TUICallFail)
```

```
- (void)calls:(NSArray<NSString *> *)userIdList callMediaType:(TUICallMediaType)cal
```

The parameters are described below:

Parameter	Type	Meaning
userIdList	NSArray	List of target users' userId
callMediaType	TUICallMediaType	Media type of the call, such as video call or voice call
params	TUICallParams	Call extension parameters, such as room number, call invitation timeout, offline push content, etc

join

Proactively join a call.

Swift

Objective-C

```
func join(callId: String)
```

```
- (void)join:(NSString *)callId;
```

The parameters are described below:

Parameter	Type	Meaning
callId	NSString	Unique ID for this call

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)
```

Deprecated Interface

call

This API is used to make a (one-to-one) call. **Note: it is recommended to use the calls API**

```
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. **Note: it is recommended to use the calls API.**

```
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: it is recommended to use the calls API**

Note:

Before making a group call, you need to create an Chat group first.

For details about how to create a group, see [Chat Group Management](#), or you can use [Chat UIKit](#) to integrate chat, call and other scenarios.

```
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: it is recommended to use the calls API.**

Note:

Before making a group call, you need to create an Chat group first.

For details about how to create a group, see [Chat Group Management](#), or you can use [Chat UIKit](#) to integrate chat, call and other scenarios.

```
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. **it is recommended to use the join API.**

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.

For details about how to create a group, see [Chat Group Management](#), or you can use [Chat UIKit](#) to integrate chat, call and other scenarios.

```
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.

TUICallEngine

Last updated : 2025-04-30 18:56:40

TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

Overview

API	Description
<code>createInstance</code>	Create a TUICallEngine instance (singleton).
<code>destroyInstance</code>	Destroy TUICallEngine instance (singleton).
<code>Init</code>	Authenticates the basic audio/video call capabilities.
<code>addObserver</code>	Add listener.
<code>removeObserver</code>	Remove listener.
<code>calls</code>	Initiate a one-to-one or multi-person call
<code>join</code>	Proactively join a call
<code>accept</code>	Accept call.
<code>reject</code>	Reject call.
<code>hangup</code>	Hang up call.
<code>ignore</code>	Ignore call.
<code>inviteUser</code>	Invite users to the current call.
<code>startRemoteView</code>	Subscribe to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribe from the video stream of a remote user.
<code>openCamera</code>	Turn on the camera.
<code>closeCamera</code>	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 <code>SDKAppID</code> 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;
```

calls

Initiate a call.

```
- (void)calls:(NSArray<NSString *> *)userIdList callMediaType:(TUICallMediaType)callMedia
```

The parameters are described below:

Parameter	Type	Meaning
userIdList	NSArray	List of target users' userId
callMediaType	TUICallMediaType	Media type of the call, such as video call or voice call
params	TUICallParams	Call extension parameters, such as room number, call invitation timeout, offline push content, etc

join

Proactively join a call.

```
- (void)join:(NSString *)callId succ:(TUICallSucc)succ fail:(TUICallFail)fail
```

The parameters are described below:

Parameter	Type	Meaning

callId	NSString	Unique ID for this call
--------	----------	-------------------------

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 call.

This API is called by a participant of a 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.

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.

Deprecated Interface

call

This API is used to make a (one-to-one) call. **Note: it is recommended to use the calls API.**

```
- (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: it is recommended to use the calls API.**

```
- (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

joinInGroupCall

This API is used to join a group call.

This API is called by a group member to join the group's call. **Note: it is recommended to use the join API.**

```
- (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.

TUICallObserver

Last updated : 2025-04-30 18:56:40

TUICallObserver APIs

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

Overview

API	Description
<code>onError</code>	An error occurred during the call.
<code>onCallReceived</code>	A call was received.
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallNotConnected</code>	The call not connected.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserInviting</code>	A user is invited to join a call.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user had a video stream.
<code>onUserAudioAvailable</code>	Whether a user had an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user was kicked offline.

`onUserSigExpired`

The user sig is expired.

Details

`onError`

An error occurred.

Note:

This callback indicates that the SDK encountered an unrecoverable error. Such errors must be listened for, and UI reminders should be sent to users if necessary.

```
- (void)onError:(int)code 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 *)callId callerId:(NSString *)callerId calleeIdList:(NSArray *)calleeIdList;
```

The parameters are described below:

Parameter	Type	Description
callId	NSString	Unique ID of this call
callerId	NSString	Caller ID
calleeIdList	NSArray	Callee ID List
mediaType	TUICallMediaType	The call type, which can be video or audio.
info	TUICallObserverExtraInfo	Extended information

`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:(NSString *)callId mediaType:(TUICallMediaType)mediaType info:(
```

The parameters are described below:

Parameter	Type	Description
callId	NSString	Unique ID of this call
mediaType	TUICallMediaType	The call type, which can be video or audio.
info	TUICallObserverExtraInfo	Extended information

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:(NSString *)callId mediaType:(TUICallMediaType)mediaType reason:(
```

The parameters are described below:

Parameter	Type	Description
callId	NSString	Unique ID of this call
mediaType	TUICallMediaType	The call type, which can be video or audio.
reason	TUICallEndReason	Call end reason
userId	NSString	Ended by user ID
totalTime	float	The call duration.
info	TUICallObserverExtraInfo	Extended information

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.

onCallNotConnected

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 onCallNotConnected callbacks (userId is his own ID); the callee receives the onCallNotConnected callback (userId is his own ID)

Callee rejection: The caller will simultaneously receive the [onUserReject](#) and onCallNotConnected callbacks (userId is his own ID); the callee receives the onCallNotConnected callback (userId is his own ID)

Callee busy: The caller will simultaneously receive the [onUserLineBusy](#) and onCallNotConnected 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)onCallNotConnected:(NSString *)callId mediaType:(TUICallMediaType)mediaType
```

The parameters are described below:

Parameter	Type	Description
callId	NSString	Unique ID of this call
mediaType	TUICallMediaType	The call type, which can be video or audio.
reason	TUICallEndReason	Call not connected reason
userId	NSString	Not connected by user ID
info	TUICallObserverExtraInfo	Extended information

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.

onUserInviting

A user is invited to join a call.

```
- (void)onUserInviting:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID of the invite.

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

Parameter	Type	Description
userId	NSString	The ID of the user who left the call.

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.

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.

onUserVoiceVolumeChanged

The volumes of all users.

```
- (void)onUserVoiceVolumeChanged:(NSDictionary <NSString *, NSNumber *> *)volumeMap
```

The parameters are described below:

Parameter	Type	Description
volumeMap	NSDictionary	The volume table, which includes the volume of each user (<code>userId</code>). Value range: 0-100.

onUserNetworkQualityChanged

The network quality of all users.

```
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkQualityInfo *> *)networkQual
```

The parameters are described below:

Parameter	Type	Description
networkQualityList	NSArray	The current network conditions for all users (<code>userId</code>).

onKickedOffline

The current user was kicked offline : At this time, you can prompt the user with a UI message and then invoke `init` again.

```
- (void)onKickedOffline;
```

onUserSigExpired

The user sig is expired : At this time, you need to generate a new `userSig` , and then invoke `init` again.

```
- (void)onUserSigExpired;
```

Deprecated Interface

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.

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.

Type Definition

Last updated : 2025-04-30 19:15:25

Common structures

TUICallDefine

Type	Description
TUICallParams	An additional parameter.
TUIOfflinePushInfo	Offline push vendor configuration information
TUICallObserverExtraInfo	Extended 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
TUICallEndReason	Call end reason

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
chatGroupId	NSString	Chat Group ID

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

TUICallObserverExtralInfo

Callback extended information.

Type	Type	Description
roomId	TUIRoomId	room ID
role	TUICallRole	Call role
userData	String	Custom extended field when initiating a call
chatGroupId	String	Group ID

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 (a total of 89 characters): Lowercase and uppercase English letters. (a-zA-Z) Number (0-9) space , ! , # , \$, % , & , (,) , + , - , : , ; , < , = , . , > , ? , @ , [,] , ^ , _ , { , } , , ~ , /

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

TUICallEndReason

Call end reason

Type	Value	Description
TUICallEndReasonUnknown	0	Unknown
TUICallEndReasonHangup	1	Hang up
TUICallEndReasonReject	2	Deny
TUICallEndReasonNoResponse	3	No response

TUICallEndReasonOffline	4	Offline
TUICallEndReasonLineBusy	5	Busy Line
TUICallEndReasonCanceled	6	Cancel call
TUICallEndReasonOtherDeviceAccepted	7	Other device answers
TUICallEndReasonOtherDeviceReject	8	Other device denies
TUICallEndReasonEndByServer	9	Backend ends

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 · TUIVideoEncoderParamsResolutionModeLand: = 640 × 360
TUIVideoEncoderParamsResolutionModePortrait	1	Portrait resolution, such as : TUIVideoEncoderParamsResolution_640_360 · TUIVideoEncoderParamsResolutionModePortr: 360 × 640

TUIVideoEncoderParamsResolution

Video resolution

Type	Value	Description
TUIVideoEncoderParamsResolution_640_360	1	Aspect ratio: 16:9 ; resolution: 640x360 ; recommended bitrate: 500kbps
TUIVideoEncoderParamsResolution_960_540	2	Aspect ratio: 16:9 ; resolution: 960x540 ; recommended bitrate: 850kbps
TUIVideoEncoderParamsResolution_1280_720	3	Aspect ratio: 16:9 ; resolution: 1280x720 ; recommended bitrate: 1200kbps
TUIVideoEncoderParamsResolution_1920_1080	4	Aspect ratio: 16:9 ; resolution: 1920x1080 ; recommended bitrate: 2000kbps

Web

API Overview

Last updated : 2025-05-27 10:12:13

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.
calls	Initiate a one-to-one or multi-person call.
join	Proactively join a 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.
setLanguage	Set the call language for the TUICallKit component.
hideFeatureButton	Hidden Button.
setLocalViewBackgroundImage	Set the background image for the local user's call interface.
setRemoteViewBackgroundImage	Set the background image for the remote user's call interface.
setLayoutMode	Set the call interface layout mode.
setCameraDefaultState	et whether the camera is opened by default.
destroyed	Destroy TUICallKit.
getTUICallEngineInstance	Get TUICallEngine instance.

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
join	Proactively join a 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
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 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.ON_CALL_RECEIVED	Call request event, v1.4.6+ supported
TUICallEvent.ON_CALL_CANCELED	Call canceled event.
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

Document Link

[TUICallEngine](#)

[TUICallEvent](#)

TUICallKit

Last updated : 2025-02-26 15:43:01

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

API	Description
<TUICallKit/>	The core UI call component.
init	Initialize TUICallKit.
calls	Initiate a one-to-one or multi-person call.
join	Proactively join a 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.
setLanguage	Set the call language for the TUICallKit component.
hideFeatureButton	Hidden Button.
setLocalViewBackgroundImage	Set the background image for the local user's call interface.
setRemoteViewBackgroundImage	Set the background image for the remote user's call interface.
setLayoutMode	Set the call interface layout mode.
setCameraDefaultState	et whether the camera is opened by default.
destroyed	Destroy TUICallKit.
getTUICallEngineInstance	Get TUICallEngine instance.

<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_480
beforeCalling	This function will be executed prior to making a call and before receiving an invitation to talk	function(type, error)	No	-
afterCalling	This function will be executed after the termination of the call	function()	No	-
onMinimized	This function will be executed when the component	function(oldStatus, newStatus)	No	-

	switches to a minimized state. The explanation for the STATUS value is			
kickedOut	The events thrown by the component occur when the current logged-in user is ejected. The call will also automatically terminate	function()	No	-
statusChanged	Event thrown by the component; this event is triggered when the call status changes. For detailed types of call status, refer to STATUS value description	function({oldStatus, newStatus})	No	-

Sample code

React
Vue3

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

<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 } from "@tencentcloud/call-uikit-vue";

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

React

Vue3

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

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

init

Initialize TUICallKit.

```
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.

calls

Initiate a one-to-one or multi-person call.

Note:

v4.0.0+ supported.

```
try {
  await TUICallKitServer.calls({
    userIDList: ['jack', 'tom'],
    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 details.
chatGroupID	String	Yes	chat group Id.
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 strRoomID 2. don't mix roomID and strRoomID, because they are not interchangeable
timeout	Number	No	Call timeout, default: 30s, unit: seconds. timeout = 0, set to no timeout
userData	String	No	Customize the extended fields when initiating a call. The called user can receive the extended fields.
offlinePushInfo	Object	No	Customize offline message push parameters

join

Proactively join a call.

Note:

v4.0.0+ supported.

```
try {
  await TUICallKitServer.join({
    callId: 'xx'
  });
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the join API. Reason:${error}`);
}
```

The parameters are described below:

Parameter	Type	Required	Meaning
callId	String	Yes	Unique ID for this call

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 <code>en</code> , <code>zh-cn</code> and <code>ja_JP</code> .

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: ${error}`);
}
```

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: ${error}`);
}
```

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: ${error}`);
}
```

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 :

Supported 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 :

Supported 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 <code>setAndroidVIVOCategory</code> 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 implementation
  }
};
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
------------------------------	-------------------------------------

Deprecated Interface

call

Makes a one-to-one call, Support for custom room ID, call timeout, offline push content, etc.**Note: v3.4.0 deprecated, it is recommended to use the calls API.**

```
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 parameters
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 is Lowercase and uppercase English letters. (a-zA-Z) ; Number (0-9) ; Spaces 、 ! 、 # 、 \$ 、 % 、 & 、 (、) 、 + 、 ° 1. roomID and strRoomID are mutually exclusive, if you use strRoomID 2. don't mix roomID and strRoomID, because they are not interchangeable
timeout	Number	No	Call timeout, default: 30s, unit: seconds. timeout = 0, set to no timeout
userData	String	No	Customize the extended fields when initiating a call. The called user can receive the data
offlinePushInfo	Object	No	Customize offline message push parameters

groupCall

Makes a group call, Support for custom room ID, call timeout, offline push content, etc.**Note: v3.4.0 deprecated, it is recommended to use the calls API.**

```
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 the
groupID	String	Yes	Call group ID, the creation of groupID can be referred to chat-cre
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 strRoomID 2. don't mix roomID and strRoomID, because they are not interchangeable
timeout	Number	No	Call timeout, default: 30s, unit: seconds. timeout = 0, set to no timeout
userData	String	No	Customize the extended fields when initiating a call. The called user
offlinePushInfo	Object	No	Customize offline message push parameters

joinInGroupCall

Join an existing audio-video call in a group.**Note: v3.4.0 deprecated, it is recommended to use the calls API.**

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: ${error}`);
}
```

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

TUICallEngine

Last updated : 2025-04-01 09:59:42

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
startRemoteView	Initiate Remote Screen Rendering
stopRemoteView	Stop Remote Screen Rendering
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

Deprecated Interface

API	Description
call	Initiate a one-on-one call, Note: it is recommended to use the calls API.
groupCall	Group Chat Invitation Call, Note: it is recommended to use the calls API.
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
joinInGroupCall	This API is used to join a group call. Note: it is recommended to use the join API.
switchCallMediaType	Switch Audio and Video Calls

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
<code>userID</code>	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
<code>userID</code>	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, handleGroupInviteeListUpdate);
```

The parameters are described below:

Parameter	Type	Meaning
userIDList	Array<String>	Group update invitation list

INVITED

Receiving a new incoming call event, the called party will be notified. By listening to this event, you can decide whether to display the call answering interface.

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, currentMicrophoneID, currentCameraID }) {
    console.log(microphoneList, cameraList, currentMicrophoneID, currentCameraID);
};
tuiCallEngine.on(TUICallEvent.DEVICED_UPDATED, handleDeviceUpdated);
```

CALL_TYPE_CHANGED

Call type switching, this event will be received.

```
let handleCallTypeChanged = function({ oldCallType, newCallType }) {
    console.log(oldCallType, newCallType);
};
tuiCallEngine.on(TUICallEvent.CALL_TYPE_CHANGED, handleDeviceUpdated);
```

The parameters are described below:

Parameter	Type	Meaning
oldCallType	Number	Old call type
newCallType	Number	New call type

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, handleOnUserNetworkQualityChange);
```

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: <pre>networkQualityList: [{ userID: quality }]</pre> 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 : 2025-04-10 11:39:18

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.
calls	Start a call.
join	Join the 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.

<code>removeObserver</code>	Unregisters an event listener.
<code>calls</code>	Start a call.
<code>accept</code>	Accepts a call.
<code>reject</code>	Rejects a call.
<code>hangup</code>	Ends a call.
<code>ignore</code>	Ignores a call.
<code>inviteUser</code>	Invite others to join the call.
<code>join</code>	Join the call.
<code>switchCallMediaType</code>	Changes the call type, for example, from video call to audio call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.
<code>openCamera</code>	Turns the camera on.
<code>closeCamera</code>	Turns the camera off.
<code>switchCamera</code>	Switches between the front and rear cameras.
<code>openMicrophone</code>	Turns the mic on.
<code>closeMicrophone</code>	Turns the mic off.
<code>selectAudioPlaybackDevice</code>	Selects the audio playback device (receiver or speaker).
<code>setSelfInfo</code>	Sets the alias and profile photo.
<code>enableMultiDeviceAbility</code>	Sets whether to enable multi-device login for TUICallEngine (supported by the premium package).
<code>setVideoRenderParams</code>	Set the rendering mode of video image.
<code>setVideoEncoderParams</code>	Set the encoding parameters of video encoder.
<code>queryRecentCalls</code>	Query call record.
<code>deleteRecordCalls</code>	Delete call record.
<code>setBlurBackground</code>	Set Blurry Video Effect
<code>setVirtualBackground</code>	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
<code>onError</code>	An error occurred during the call.
<code>onUserInviting</code>	Callback when a user is invited to join a call.
<code>onCallReceived</code>	A call invitation was received.
<code>onCallNotConnected</code>	Callback for call cancellation
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user has a video stream.
<code>onUserAudioAvailable</code>	Whether a user has an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user is kicked offline
<code>onUserSigExpired</code>	Ticket expires while online

Key Type Definitions

Type	Description
VideoEncoderParams	Video encoding parameters.
VideoRenderParams	Video render parameters.
TUICallMediaType	Call media type.
TUICallParams	Call extension parameters.
TUICamera	Camera type.
TUIAudioPlaybackDevice	Audio playback device.
TUIRoomId	Room ID for audio and video in a call.

TUICallKit

Last updated : 2025-03-27 15:25:58

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.
calls	Start a call.
join	Join the 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
<code>sdkAppId</code>	<code>int</code>	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

calls

Initiate a call. **Supported by v2.9+.**

```
Future<TUIResult> calls(List<String> userIdList, TUICallMediaType mediaType, TUICal
```

Parameter	Type	Description
userIdList	List<String>	The target user IDs.
callMediaType	TUICallMediaType	The call type, which can be video or audio.

params	TUICallParams	Optional Extended call parameters, such as room number, call invitation timeout, offline push of custom content, etc.
--------	-------------------------------	--

join

Actively join the call. **Supported by v2.9+.**

```
Future<void> join(String callId)
```

Parameter	Type	Description
callId	String	Unique ID for this call.

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

Deprecated interfaces

call

This API is used to make a (one-to-one) call.

Note :

This interface has been deprecated in v2.9+. It is recommended to use the calls interface instead.

```
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.

Note :

This interface has been deprecated in v2.9+. It is recommended to use the calls interface instead.

```
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. Before making a group call, you need to create an IM group first.

Note :

This interface has been deprecated in v2.9+. It is recommended to use the join interface instead.

```
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.

TUICallEngine

Last updated : 2025-04-01 09:54:57

TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

Overview

API	Description
<code>init</code>	Authenticates the basic audio/video call capabilities.
<code>unInit</code>	The destructor function, which releases resources used by TUICallEngine.
<code>addObserver</code>	Registers an event listener.
<code>removeObserver</code>	Unregisters an event listener.
<code>calls</code>	Start a call.
<code>accept</code>	Accepts a call.
<code>reject</code>	Rejects a call.
<code>hangup</code>	Ends a call.
<code>ignore</code>	Ignores a call.
<code>inviteUser</code>	Invite others to join the call.
<code>join</code>	Join the call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.
<code>openCamera</code>	Turns the camera on.
<code>closeCamera</code>	Turns the camera off.
<code>switchCamera</code>	Switches between the front and rear cameras.

<code>openMicrophone</code>	Turns the mic on.
<code>closeMicrophone</code>	Turns the mic off.
<code>selectAudioPlaybackDevice</code>	Selects the audio playback device (receiver or speaker).
<code>setSelfInfo</code>	Sets the alias and profile photo.
<code>enableMultiDeviceAbility</code>	Sets whether to enable multi-device login for TUICallEngine (supported by the premium package).
<code>setVideoRenderParams</code>	Set the rendering mode of video image.
<code>setVideoEncoderParams</code>	Set the encoding parameters of video encoder.
<code>setBlurBackground</code>	Set Blurry Video Effect
<code>setVirtualBackground</code>	Set Virtual Background Image
<code>setBeautyLevel</code>	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)
```

calls

Initiate a call. **Supported by v2.9+.**

```
Future<TUIResult> calls(List<String> userIdList, TUICallMediaType mediaType, TUICal
```

Parameter	Type	Description
userIdList	List<String>	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.

```
Future<TUIResult> accept()
```

reject

This API is used to reject a call. After receiving the `onCallReceived()` callback, you can call this API to reject the call.

```
Future<TUIResult> reject()
```

ignore

This API is used to ignore a call. After receiving the `onCallReceived()`, you can call this API to ignore the call. The caller will receive the `onUserLineBusy` callback.

Note: If your project involves live streaming or conferencing, you can also use this API to implement the “in a meeting” or “on air” feature.

```
Future<TUIResult> ignore()
```

hangup

This API is used to end a call.

```
Future<TUIResult> hangup()
```

inviteUser

This API is used to invite users to the current group call.

This API is called by a participant of a group call to invite new users.

```
Future<void> ininviteUser(List<String> userIdList, TUICallParams params, TUIValueCal
```

Parameter	Type	Description
userIdList	List<String>	The target user IDs.
params	TUICallParams	An additional parameter. such as roomId, call timeout, offline push info, etc.

join

Actively join the call. **Supported by v2.9+.**

```
Future<void> join(String callId)
```

Parameter	Type	Description
callId	String	Unique ID for this call.

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

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.

Deprecated interfaces

call

This API is used to make a (one-to-one) call.

Note :

This interface has been deprecated in v2.9+. It is recommended to use the calls interface instead.

```
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.

Before making a group call, you need to create an IM group first.

Note :

This interface has been deprecated in v2.9+. It is recommended to use the calls interface instead.

```
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.

joinInGroupCall

This API is used to join a group call.

This API is called by a group member to join the group's call.

Note :

This interface has been deprecated in v2.9+. It is recommended to use the join interface instead.

```
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.

TUICallObserver

Last updated : 2025-04-10 11:39:18

TUICallObserver API

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

Overview

API	Description
<code>onError</code>	A call occurred during the call.
<code>onUserInviting</code>	Callback when a user is invited to join a call.
<code>onCallReceived</code>	A call invitation was received.
<code>onCallNotConnected</code>	Callback for call cancellation
<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user had a video stream.
<code>onUserAudioAvailable</code>	Whether a user had an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.

<code>onKickedOffline</code>	The current user was kicked offline.
<code>onUserSigExpired</code>	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.

onUserInviting

Callback when a user is invited to join a call.

```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onUserInviting: (String userId) {  
        // TODO  
    }  
));
```

The parameters are listed in the table below.

Parameter	Type	Meaning
userId	String	ID of the invited user

onCallReceived

Received a new incoming call request callback, the callee will receive it. You can listen to this event to determine whether to display the call answering interface.

```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onCallReceived: (String callId, String callerId, List<String> calleeIdList, TUI  
        // TODO  
    }  
));
```


The parameters are listed in the table below.

Parameter	Type	Meaning
callId	String	Unique ID of this call
callerId	String	Calling ID (inviter)
calleeIdList	List<String>	Called ID list (invitees)
mediaType	TUICallMediaType	Media type of the call. For example: <code>TUICallMediaType.video</code> or <code>TUICallMediaType.audio</code>
info	CallObserverExtraInfo	Extended information

onCallNotConnected

Callback for call cancellation.

```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onCallNotConnected: (String callId, TUICallMediaType mediaType, CallEndReason rea  
        String userId, CallObserverExtraInfo info) {  
        // TODO  
    }  
));
```

The parameters are listed in the table below.

Parameter	Type	Meaning
callId	String	Unique ID of this call
mediaType	TUICallMediaType	Media type of the call. For example: <code>TUICallMediaType.video</code> or <code>TUICallMediaType.audio</code>
reason	CallEndReason	The causes for call termination
userId	String	User ID ending the call
info	CallObserverExtraInfo	Extended information

onCallBegin

Indicates that the call is connected, and both the caller and callee can receive it. You can listen to this event to start processes such as cloud recording and content review.

```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onCallBegin: (String callId, TUICallMediaType mediaType, CallObserverExtraInfo
```

```
// TODO
}
));
```

The parameters are listed in the table below.

Parameter	Type	Meaning
callId	String	Unique ID of this call
mediaType	TUICallMediaType	Media type of the call. For example: <code>TUICallMediaType.video</code> or <code>TUICallMediaType.audio</code>
info	CallObserverExtraInfo	Extended information

onCallEnd

Indicates that the call is connected, and both the caller and callee can receive it. You can listen to this event to display information such as call duration and call type, or to stop the cloud recording process.

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallEnd: (String callId, TUICallMediaType mediaType, CallEndReason reason,
        String userId, double totalTime, CallObserverExtraInfo info) {
        // TODO
    }
));
```

The parameters are listed in the table below.

Parameter	Type	Meaning
callId	String	Unique ID of this call
mediaType	TUICallMediaType	Media type of the call. For example: <code>TUICallMediaType.video</code> or <code>TUICallMediaType.audio</code>
reason	CallEndReason	The causes for call termination
userId	String	User ID ending the call
totalTime	double	Duration of this call, unit ms
info	CallObserverExtraInfo	Extended information

Note:

Events on the client side are generally lost due to exceptions such as process termination. If you need to complete billing logic by monitoring call duration, it is recommended to use REST API to complete such processes.

onCallMediaTypeChanged

The call type changed.

```
TUICallEngine.instance.addObserver(TUICallObserver(  
    onCallMediaTypeChanged: (TUICallMediaType oldCallMediaType, TUICallMediaType 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.

```
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: () {  
    }  
));
```

Deprecated interfaces

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.

Type Definition

Last updated : 2025-04-15 15:55:25

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.
chatGroupId	String	Group ID.

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 <code>setAndroidFCMChannelID</code> 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 <code>setAndroidVIVOCategory</code> 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.

CallObserverExtraInfo

Callback extended information.

Type	Type	Description
roomId	TUIRoomId	room ID
role	TUICallRole	Call role
userData	String	Custom extended field when initiating a call. For details, see TUICallParams .
chatGroupId	String	Group ID

enum definition

TUICallMediaType

Call media type.

Type	Description
none	Unknown
audio	Audio call
video	Video call

TUICallRole

Call role.

Type	Description
none	Unknown
caller	Caller (inviter)
called	Callee (invitee)

TUICallStatus

Call status.

Type	Description
none	Unknown
waiting	The call is currently waiting
accept	The call has been accepted

TUICallScene

Call scene.

Type	Description
groupCall	Group call
singleCall	one to one call

TUINetworkQuality

Network quality.

Type	Description
unknown	Unknown
excellent	Excellent
good	Good
poor	Poor
bad	Bad
vBad	Very bad
down	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	Description
fill	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	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	Description
rotation_0	No rotation
rotation_90	Clockwise rotation by 90 degrees
rotation_180	Clockwise rotation by 180 degrees
rotation_270	Clockwise rotation by 270 degrees

ResolutionMode

Video aspect ratio mode.

Type	Description
landscape	Landscape resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360.
portrait	Portrait resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640.

Resolution

Video resolution.

Type	Description
resolution_640_360	Aspect ratio: 16:9 ; resolution: 640x360 ; recommended bitrate: 500kbps
resolution_960_540	Aspect ratio: 16:9 ; resolution: 960x540 ; recommended bitrate: 850kbps
resolution_1280_720	Aspect ratio: 16:9 ; resolution: 1280x720 ; recommended bitrate: 1200kbps
resolution_1920_1080	Aspect ratio: 16:9 ; resolution: 1920x1080 ; recommended bitrate: 2000kbps

TUICallIOSOfflinePushType

iOS offline push type.

Type	Description
APNs	APNs
VoIP	VoIP

TUICamera

Camera type.

Type	Description
front	Front camera.
back	Rear camera.

TUIAudioPlaybackDevice

Audio playback device.

Type	Description
speakerphone	Speaker
earpiece	Earpiece

TUICallResultType

Call recording type.

Type	Description
unknown	Unknown
missed	Missed
incoming	Incoming call
outgoing	Outgoing Call

CallEndReason

Call end reason.

Type	Description
unknown	Unknown
hangup	Hang up
reject	Deny
noResponse	No response
offline	Offline
lineBusy	Busy Line
canceled	Cancel call
otherDeviceAccepted	Other device answers
otherDeviceReject	Other device denies
endByServer	Backend ends

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.

<code>onCallBegin</code>	The call was connected.
<code>onCallEnd</code>	The call ended.
<code>onCallMediaTypeChanged</code>	The call type changed.
<code>onUserReject</code>	A user declined the call.
<code>onUserNoResponse</code>	A user didn't respond.
<code>onUserLineBusy</code>	A user was busy.
<code>onUserJoin</code>	A user joined the call.
<code>onUserLeave</code>	A user left the call.
<code>onUserVideoAvailable</code>	Whether a user has a video stream.
<code>onUserAudioAvailable</code>	Whether a user has an audio stream.
<code>onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>onUserNetworkQualityChanged</code>	The network quality of all users.
<code>onKickedOffline</code>	The current user was kicked offline.
<code>onUserSigExpired</code>	The user sig is expired.

TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description
<code>hangup</code>	Ends a call.
<code>accept</code>	Answers a call.
<code>setVideoRenderParams</code>	Set the rendering mode of video image.
<code>setVideoEncoderParams</code>	Set the encoding parameters of video encoder.

TUICallKit

Last updated : 2024-07-19 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 call
options.callMediaType	Number	Media type of the call, e.g., voice call (callMediaType = 1), video call (callMediaType = 2)
callback	Function	callback function, code = 0 means the call was successful; code != 0 means the call failed, see msg for the reason.

setCallingBell

To set a customized incoming call tone, here you are limited to passing in the local file address, and you need to make sure that the file directory is accessible to the application.

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

```
// 【1】 Save audio files locally through uni.saveFile. Reference.: https://zh.uniapp.cn/docs/uni-app/api/uni-save-file
const tempFilePath = './static/rain.mp3';
let musicFilePath = '';
uni.saveFile({
  tempFilePath: tempFilePath,
  success: (res) => {
    musicFilePath = res.savedFilePath;

    // 【2】 Convert relative path to absolute path, otherwise access will not be successful
    musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath);

    // 【3】 set ringtone
    TUICallKit.setCallingBell(musicFilePath, (res) => {
      if (res.code === 0) {
        console.log('setCallingBell success');
      } else {
        console.log(`setCallingBell failed, error message = ${res.msg}`);
      }
    });
  },
  fail: (err) => {
    console.error('save failed');
  },
});
```

Parameter	Type	Description
filePath	String	Ringtone local file address
callback	Function	callback function, code = 0 means the call was successful; code != 0 means the call failed, see msg for the reason.

enableMuteMode

This API is used to set whether to turn on the mute mode.

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

Parameter	Type	Description
enable	Boolean	Mute on, mute off; true means mute on

enableFloatWindow

This API is used to set whether to enable floating windows.

The default value is `false`, and the floating window button in the top left corner of the call view is hidden. If it is set to `true`, the button will become visible.

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

Parameter	Type	Description
enable	Boolean	Enable/disable the floating window function; true means floating window is enabled.

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
<code>hangup</code>	Ends a call.
<code>accept</code>	Accepts a call.
<code>setVideoRenderParams</code>	Set the rendering mode of video image.
<code>setVideoEncoderParams</code>	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-TUICallEngine');
const params = {
  userID: '234',
  fillMode: 0, // 0-fill mode, 1-adapter mode
  rotation: 1, // 0:Rotation_0; 1: Rotation_90; 2: Rotation_180; 3: Rotation_270;
};
TUICallEngine.setVideoRenderParams(params, (res) => {
  console.warn('res = ', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
userID	String	target userId
params	Object	Video frame rendering parameters, e.g. frame rotation angle, fill mode

setVideoEncoderParams

Set the encoding parameters of video encoder.

This setting can determine the quality of image viewed by remote users, which is also the image quality of on-cloud recording files.

```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngine');
const params = {
  resolution: 108,
  resolutionMode: 0, // 0--landscape, 1--portrait
};
TUICallEngine.setVideoEncoderParams(params, (res) => {
  console.warn('res = ', JSON.stringify(res));
});
```

The parameters are described below:

Parameter	Type	Description
resolution	Number	video resolution 62: aspect ratio 16:9 ; resolution 640x360 ; 64: aspect ratio 4:3 ; resolution 960x720 ; 108: aspect ratio 16:9 ; resolution 640x360 ; 110: aspect ratio 16:9 ; resolution 960x540 ; 112: aspect ratio 16:9 ; resolution 1280x720 ; 114: aspect ratio 16:9 ; resolution 1920x1080 ;

resolutionMode	Number	resolution mode 0: Landscape 1: Portrait
----------------	--------	--

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,	The volume table, which includes the volume of each user

Integer>

(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));  
});
```

React Native API Overview

Last updated : 2025-02-26 14:17:50

TUICallKit (including UI Interface)

TUICallKit APIs

UICallKit API, you can quickly implement a WeChat-like audio and video call scenario through simple interfaces.

API	Description
login	Login.
logout	Logout.
calls	Initiate a one-to-one or multi-person call.
call	To make a one-on-one call, supports custom room ID, call timeout, offline push content, and more.
groupCall	To make a group call, supports custom room ID, call timeout, offline push content, and more.
joinInGroupCall	Join a group call.
setCallingBell	Customize user's ringtone.
enableMuteMode	Enable/Disable Ringtone.
enableVirtualBackground	Enable/disable blurry background feature.
setScreenOrientation	Set Screen Orientation.
on	Listen to TUICallKit events.
off	Cancel listening to TUICallKit events.

TUICallEvent

TUICallEvent is the callback event class corresponding to TUICallKit. Through this callback, you can listen to the callback events you are interested in.

Event	Description
<code>TUICallEvent.onError</code>	Error Callback during Call.
<code>TUICallEvent.onCallReceived</code>	Callback for a Call Request.
<code>TUICallEvent.onCallCancelled</code>	Callback for Call Cancellation.
<code>TUICallEvent.onCallBegin</code>	Callback for Call Connection.
<code>TUICallEvent.onCallEnd</code>	Callback for Call Termination.
<code>TUICallEvent.onUserReject</code>	xxxx User declines the call Callback.
<code>TUICallEvent.onUserNoResponse</code>	xxxx User Non-response Callback.
<code>TUICallEvent.onUserLineBusy</code>	xxxx User Busy Line Callback.
<code>TUICallEvent.onUserJoin</code>	xxxx User Joins Call Callback.
<code>TUICallEvent.onUserLeave</code>	A user left the call.
<code>TUICallEvent.onCallMediaTypeChanged</code>	Callback for a change in the Call's Media Type.
<code>TUICallEvent.onKickedOffline</code>	Current user kicked offline.
<code>TUICallEvent.onUserSigExpired</code>	Ticket expired while online.
<code>TUICallEvent.onUserVideoAvailable</code>	Whether a user has a video stream.
<code>TUICallEvent.onUserAudioAvailable</code>	Whether a user has an audio stream.
<code>TUICallEvent.onUserVoiceVolumeChanged</code>	The volume levels of all users.
<code>TUICallEvent.onUserNetworkQualityChanged</code>	The network quality of all users.

TUICallEngine (No UI)

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

API	Description
<code>createInstance</code>	Creates a <code>TUICallEngine</code> instance (singleton mode).

<code>destroyInstance</code>	Terminates a <code>TUICallEngine</code> instance (singleton mode).
<code>login</code>	Authenticates the basic audio/video call capabilities.
<code>on</code>	Registers an event listener.
<code>off</code>	Unregisters an event listener.
<code>call</code>	Makes a one-to-one call.
<code>accept</code>	Accepts a call.
<code>reject</code>	Rejects a call.
<code>ignore</code>	Ignores a call.
<code>hangup</code>	Ends a call.
<code>switchCallMediaType</code>	Changes the call type, for example, from video call to audio call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.
<code>openCamera</code>	Turns the camera on.
<code>switchCamera</code>	Switches between the front and rear cameras.
<code>closeCamera</code>	Turns the camera off.
<code>openMicrophone</code>	Turns the mic on.
<code>closeMicrophone</code>	Turns the mic off.
<code>selectAudioPlaybackDevice</code>	Selects the audio playback device (receiver or speaker).
<code>setVideoRenderParams</code>	Set the rendering mode of video image.
<code>setVideoEncoderParams</code>	Set the encoding parameters of video encoder.
<code>setBeautyLevel</code>	Set beauty level, support turning off default beauty.
<code>setSelfInfo</code>	Sets the alias and profile photo.
<code>enableMultiDeviceAbility</code>	Sets whether to enable multi-device login for <code>TUICallEngine</code> .

TUICallKit

Last updated : 2025-02-26 14:17:50

TUICallKit API

The TUICallKit API is a **UI-inclusive interface** for the audio and video call component. By using the TUICallKit API, you can quickly implement a WeChat-like audio and video call scenario through simple interfaces. For more detailed integration steps, please refer to Quick Access to TUICallKit.

TUICallKit APIs

API	Description
login	Login.
logout	Logout.
calls	Initiate a one-to-one or multi-person call.
call	To make a one-on-one call, supports custom room ID, call timeout, offline push content, and more.
groupCall	To make a group call, supports custom room ID, call timeout, offline push content, and more.
joinInGroupCall	Join a group call.
setCallingBell	Customize user's ringtone.
enableMuteMode	Set your own nickname and profile photo.
enableVirtualBackground	Turn On/Off Ringtone.
setScreenOrientation	Set Screen Orientation.
on	Listen to TUICallKit events
off	Cancel listening to TUICallKit events

API Details

login

Login. This step is crucial. Only after successful login can you use the various features provided by TUICallKit normally.

```
TUICallKit.login(  
  {  
    sdkAppId: 0,  
    userId: '',  
    userSig: '',  
  },  
  (res) => {  
    console.log('login success');  
  },  
  (errCode, errMsg) => {  
    console.log('login error');  
  }  
);
```

Parameter	Type	Meaning
sdkAppId	Number	The unique identifier SDKAppID for the audio and video application created in Tencent RTC Console .
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.
userSig	String	SDKSecretKey for the audio and video application created in Tencent RTC Console .

logout

Logout. After logging out, no TUICallKit events will be listened to.

```
TUICallKit.logout(  
  (res) => {  
    console.log('login success', res);  
  },  
  (errCode, errMsg) => {  
    console.log('login error', errCode, errMsg);  
  }  
);
```

calls

Initiate a one-to-one or multi-person call.


```
TUICallKit.calls(  
  {  
    userIdList: userIDList,  
    mediaType: MediaType.Audio,  
    callParams: {  
      offlinePushInfo: {  
        title: '',  
        desc: '',  
      },  
    },  
  },  
  () => {  
    console.log('calls success');  
  },  
  () => {  
    console.log('calls error');  
  }  
);
```

Parameter	Type	Meaning
userIdList	Array<String>	The target user IDs.
mediaType	MediaType	The media type of the call, such as video call, voice call. MediaType.Audio: Voice Call. MediaType.Video: Video Call.
callParams	callParams	call extension parameters include room number, call invitation timeout, offline push custom content, etc.

call

Make a phone call (1v1 call), supports custom room number, call invitation timeout, offline push content, and more.

```
TUICallKit.call(  
  {  
    userId: calleeID,  
    mediaType: MediaType.Audio,  
    callParams: {  
      offlinePushInfo: {  
        title: '',  
        desc: '',  
      },  
    },  
  },  
  () => {
```

```
    console.log('call success');
  },
  () => {
    console.log('call error');
  }
);
```

Parameter	Type	Meaning
userId	String	target user's userId
mediaType	MediaType	The media type of the call, such as video call, voice call. MediaType.Audio: Voice Call. MediaType.Video: Video Call.
callParams	callParams	call extension parameters include room number, call invitation timeout, offline push custom content, etc.

groupCall

Initiate group communication.

```
TUICallKit.groupCall(
{
  userIdList: userIDList,
  mediaType: MediaType.Audio,
  groupId: '',
},
(res) => {
  console.log('groupCall success', res);
},
(errCode, errMsg) => {
  console.log('groupCall error', errCode, errMsg);
}
);
```

Parameter	Type	Meaning
groupId	String	The group ID.
userIdList	Array<String>	The target user IDs.
mediaType	MediaType	The media type of the call, such as video call, voice call MediaType.Audio: Voice Call. MediaType.Video: Video Call.
callParams	callParams	call extension parameters, for example: room number, call invitation timeout,

	custom content for offline push, etc.
--	---------------------------------------

joinInGroupCall

Join an existing audio-video call in a group.

```
TUICallKit.joinInGroupCall(  
  {  
    roomId: '',  
    groupId: '',  
    mediaType: '',  
  }  
);
```

Parameter	Type	Meaning
roomId	RoomId	Audio-Video Room ID for this call
groupId	String	Group ID associated with this group call
mediaType	MediaType	The media type of the call, such as video call, voice call

setCallingBell

Set a custom 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 introduce the ringtone file.

To reset the ringtone, pass in an empty string for `filePath`.

```
String filePath = '';  
TUICallKit.setCallingBell(filePath);
```

Parameter	Type	Meaning
filePath	String	Ringtone file address

enableMuteMode

Enable/Disable Ringtone. After enabling, the ringtone will not play when receiving a call request.

```
Boolean enable = true  
TUICallKit.enableMuteMode(enable);
```

Parameter	Type	Meaning
-----------	------	---------

enable	Boolean	Enable/Disable Ringtone. Default is false.
--------	---------	--

enableVirtualBackground

Enable/disable blurry background feature. By calling the interface, you can display the blurry background feature button on the UI, and click the button to directly enable the blurry background feature.

```
Boolean enable = true
TUICallKit.enableVirtualBackground(enable);
```

Parameter	Type	Meaning
enable	Boolean	enable = true Show blurry background button enable = false Do not show blurry background button

setScreenOrientation

Set screen display mode.

```
Number orientation = 0
TUICallKit.setScreenOrientation(orientation);
```

Parameter	Type	Meaning
orientation	Number	orientation = 0 : Portrait display. orientation = 1 : Landscape display. orientation = 2 : Automatically select the best display mode based on the current state of the device.

on

You can listen to TUICallKit events using the example code below. For event details, please refer to [TUICallEvent](#).

```
TUICallKit.on(TUICallEvent.onCallReceived, (res: any) => {
  console.log('onUserReject userId=' + res.userId);
});
```

Parameter	Type	Meaning
type	String	For the events you are listening to with TUICallKit, please refer to the event list in TUICallEvent .
params	Any	Information carried by the event, please refer to TUICallEvent for details.

off

You can use the following sample code to cancel listening to TUICallKit events.

```
TUICallKit.off(TUICallEvent.onCallReceived);
```

TUICallEngine

Last updated : 2025-02-26 14:17:50

TUICallEngine APIs

`TUICallEngine` is an audio/video call component that **does not include UI elements**. If `TUICallKit` does not meet your requirements, you can use the APIs of `TUICallEngine` to customize your project.

Overview

API	Description
<code>createInstance</code>	Creates a <code>TUICallEngine</code> instance (singleton mode).
<code>destroyInstance</code>	Terminates a <code>TUICallEngine</code> instance (singleton mode).
<code>login</code>	Authenticates the basic audio/video call capabilities.
<code>on</code>	Registers an event listener.
<code>off</code>	Unregisters an event listener.
<code>call</code>	Makes a one-to-one call.
<code>accept</code>	Accepts a call.
<code>reject</code>	Rejects a call.
<code>ignore</code>	Ignores a call.
<code>hangup</code>	Ends a call.
<code>switchCallMediaType</code>	Changes the call type, for example, from video call to audio call.
<code>startRemoteView</code>	Subscribes to the video stream of a remote user.
<code>stopRemoteView</code>	Unsubscribes from the video stream of a remote user.
<code>openCamera</code>	Turns the camera on.
<code>switchCamera</code>	Switches between the front and rear cameras.
<code>closeCamera</code>	Turns the camera off.

openMicrophone	Turns the mic on.
closeMicrophone	Turns the mic off.
selectAudioPlaybackDevice	Selects the audio playback device (receiver or speaker).
setVideoRenderParams	Set the rendering mode of video image.
setVideoEncoderParams	Set the encoding parameters of video encoder.
setBeautyLevel	Set beauty level, support turning off default beauty.
setSelfInfo	Sets the alias and profile photo.
enableMultiDeviceAbility	Sets whether to enable multi-device login for <code>TUICallEngine</code>

Details

CreateInstance

This API is used to create a TUICallEngine singleton.

```
TUICallKit.CreateInstance();
```

destroyInstance

This API is used to terminate a TUICallEngine singleton.

```
TUICallKit.destroyInstance();
```

login

This API is used to initialize TUICallEngine. Call it to authenticate the call service and perform other required actions before you call other APIs.

```
login(  
  sdkAppId: number;  
  userId: string;  
  userSig: string;  
  onSuccess?: (data: String) => void;  
  onError?: (errCode: number, errMsg: string) => void;  
): void {}
```

Parameter	Type	Description
sdkAppId	Number	The unique identifier SDKAppID for the audio and video application created in Tencent RTC Console .
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.
userSig	String	SDKSecretKey for the audio and video application created in Tencent RTC Console .

on

This API is used to register an event listener to listen for TUICallEvents events.

```
on(type: TUICallEvent, listener: (params: any) => void): void {}
```

Parameter	Type	Description
type	TUICallEvent	Registers an event listener.

off

This API is used to unregister an event listener.

```
off(type: TUICallEvent): void {}
```

Parameter	Type	Description
type	TUICallEvent	Unregister an event listener.

call

This API is used to make a (one-to-one) call.

```
call(  
  userId: string;  
  mediaType: MediaType;  
  callParams?: CallParams;  
  onSuccess?: (data: String) => void;  
  onError?: (errCode: number, errMsg: string) => void;  
): void {}
```

Parameter	Type	Description
userId	string	The target user ID.

mediaType	MediaType	which can be video or audio. For details, please refer to MediaType .
callParams	CallParams	An additional parameter, such as roomId, call timeout, offline push info, etc. For details, please refer to callParams

accept

This API is used to accept a call. After receiving the `onCallReceived()` callback, you can call this API to accept the call.

```
accept(  
  onSuccess?: (data: String) => void;  
  onError?: (errorCode: number, errMsg: string) => void;  
): void {}
```

reject

This API is used to reject a call. After receiving the `onCallReceived()` callback, you can call this API to reject the call.

```
reject(  
  onSuccess?: (data: String) => void;  
  onError?: (errorCode: number, errMsg: string) => void;  
): void {}
```

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.

```
ignore(  
  onSuccess?: (data: String) => void;  
  onError?: (errorCode: number, errMsg: string) => void;  
): void {}
```

hangup

This API is used to end a call.

```
hangup(  
  onSuccess?: (data: String) => void;  
  onError?: (errorCode: number, errMsg: string) => void;  
): void {}
```

switchCallMediaType

This API is used to change the call type.

```
switchCallMediaType(mediaType: MediaType): void {}
```

Parameter	Type	Description
mediaType	MediaType	The call type, which can be video or audio. For details, For details, please refer to MediaType .

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`.

```
startRemoteView(  
  userId: string;  
  viewId: string;  
  onPlaying: (params: any) => void;  
  onLoading: (params: any) => void;  
): void {}
```

Parameter	Type	Description
userId	string	The target user ID.
viewId	string	The target view ID.

stopRemoteView

This API is used to unsubscribe from the video stream of a remote user.

```
stopRemoteView(userId: string): void {}
```

Parameter	Type	Description
userId	string	The target user ID.

openCamera

This API is used to turn the camera on.

```
openCamera(  
  camera?: Camera;  
  viewId: string;  
  onSuccess?: (data: String) => void;  
  onError?: (errCode: number, errMsg: string) => void;): void {}
```

Parameter	Type	Description
camera	Camera	This parameter is used to set the front and back cameras. Camera.Front: Front camera. Camera.Back: Rear camera.
viewId	string	The target view ID. We recommend that you set the view ID to the user's user ID

switchCamera

This API is used to switch between the front and rear cameras.

```
switchCamera(camera: Camera): void {}
```

Parameter	Type	Description
camera	Camera	This parameter is used to set the front and back cameras. Camera.Front: Front camera. Camera.Back: Rear camera.

closeCamera

This API is used to turn the camera off.

```
closeCamera(): void {}
```

openMicrophone

This API is used to turn the mic on.

```
openMicrophone(  
  onSuccess?: (data: String) => void;  
  onError?: (errorCode: number, errMsg: string) => void;  
): void {}
```

closeMicrophone

This API is used to turn the mic off.

```
closeMicrophone(  
  onSuccess?: (data: String) => void;  
  onError?: (errorCode: number, errMsg: string) => void;  
): void {}
```

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.

```
selectAudioPlaybackDevice(  
  device: AudioPlaybackDevice;  
  onSuccess?: (data: String) => void;  
  onError?: (errCode: number, errMsg: string) => void;  
): void {}
```

Parameter	Type	Description
device	AudioPlaybackDevice	This parameter is used to set the front and back cameras. AudioPlaybackDevice.Earpiece AudioPlaybackDevice.Speakerphone

setVideoRenderParams

Set the rendering mode of video image.

```
setVideoRenderParams(  
  userId: string;  
  videoRender?: {  
    fillMode?: FillMode;  
    rotation?: Rotation;  
  };  
  onSuccess?: (data: String) => void;  
  onError?: (errCode: number, errMsg: string) => void;  
): void {}
```

Parameter	Type	Description
userId	string	The target user ID.
fillMode	FillMode	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.
rotation	Rotation	Rotation.Rotation_0: No rotation Rotation.Rotation_90: Clockwise rotation by 90 degrees Rotation.Rotation_180: Clockwise rotation by 180 degrees Rotation.Rotation_270: Clockwise rotation by 0 degrees

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.

```
setVideoEncoderParams(  
  videoEncoder?: {  
    resolution?: Resolution;  
    resolutionMode?: ResolutionMode;  
  };  
  onSuccess?: (data: String) => void;  
  onError?: (errCode: number, errMsg: string) => void;  
): void {}
```

Parameter	Type	Description
resolution	Resolution	Video resolution
resolutionMode	ResolutionMode	Video aspect ratio mode

setBeautyLevel

Set beauty level, support turning off default beauty.

```
setBeautyLevel(  
  level: number;  
  onSuccess?: (data: String) => void;  
  onError?: (errCode: number, errMsg: string) => void;  
): void {}
```

Parameter	Type	Description
level	number	Beauty level, range: 0 - 9, 0 means turning off the effect, 9 means the most obvious effect.

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.

```
setSelfInfo(  
  nickname: string;  
  avatar?: string;  
  onSuccess?: (data: String) => void;  
  onError?: (errCode: number, errMsg: string) => void;  
): void {}
```

--	--	--

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).

```
enableMultiDeviceAbility(  
  enable: boolean;  
  onSuccess?: (data: String) => void;  
  onError?: (errCode: number, errMsg: string) => void;  
): void {}
```

Parameter	Type	Description
enable	boolean	set whether to enable multi-device login for TUICallEngine

TUICallEvent

Last updated : 2025-02-21 15:40:11

TUICallEvent APIs

TUICallEvent API is the **Event Interface** of the Audio and Video Call Components.

Event	Description
TUICallEvent.onError	Error Callback during Call
TUICallEvent.onCallReceived	Callback for a Call Request
TUICallEvent.onCallCancelled	Callback for Call Cancellation
TUICallEvent.onCallBegin	Callback for Call Connection
TUICallEvent.onCallEnd	Callback for Call Termination
TUICallEvent.onUserReject	xxxx User declines the call Callback
TUICallEvent.onUserNoResponse	xxxx User Non-response Callback
TUICallEvent.onUserLineBusy	xxxx User Busy Line Callback
TUICallEvent.onUserJoin	xxxx User Joins Call Callback
TUICallEvent.onUserLeave	A user left the call.
TUICallEvent.onCallMediaTypeChanged	Callback for a change in the Call's Media Type
TUICallEvent.onKickedOffline	Current user kicked offline
TUICallEvent.onUserSigExpired	Ticket expired while online
TUICallEvent.onUserVideoAvailable	Whether a user has a video stream.
TUICallEvent.onUserAudioAvailable	Whether a user has an audio stream.
TUICallEvent.onUserVoiceVolumeChanged	The volume levels of all users.
TUICallEvent.onUserNetworkQualityChanged	The network quality of all users.

TUICallEvent details

onError

Error callback.

```
TUICallKit.on(TUICallEvent.onError, (res: any) => {  
  console.log('onError code=' + res.code + ',message=' + res.message);  
});
```

Parameter	Type	Description
code	Number	Error Code
message	String	Error message

onCallReceived

Received a callback for a new incoming call request.

```
TUICallKit.on(TUICallEvent.onCallReceived, (res: any) => {  
  console.log('onCallReceived callerId=' + res.callerId);  
});
```

Parameter	Type	Description
callerId	String	Caller ID (inviter)
calleeIdList	Array<String>	List of called IDs (invitees)
groupId	String	Group Call ID
callMediaType	Number	The media type of the call, such as video call, voice call. params.callMediaType = 0 : Voice Call. params.callMediaType = 1 : Video Call.
userData	String	User-added extension fields.

onCallCancelled

Indicates the call was canceled by the caller, missed by the callee, rejected, etc., covering multiple scenarios.

Caller Cancellation: The caller receives this callback (callerId is oneself); the callee receives this callback (callerId is **the caller's ID**)

Callee Timeout: The caller receives both onUserNoResponse and onCallCancelled callbacks (callerId is their own ID); the callee receives the onCallCancelled callback (callerId is their own ID)

Call rejected: The caller will receive both onUserReject and onCallCancelled callbacks (callerId is your own ID); the callee receives the onCallCancelled callback (callerId is your own ID)

Call busy: The caller will receive both onUserLineBusy and onCallCancelled callbacks (callerId is your own ID)

Unexpected interruption: The callee fails to receive the call and gets this callback (callerId is your own ID)

```
TUICallKit.on(TUICallEvent.onCallCancelled, (res: any) => {  
  console.log('onCallCancelled userId=' + res.callerId);  
});
```

Parameter	Type	Description
callerId	String	Caller ID (inviter)

onCallBegin

Indicates call connection. Both caller and called can receive it. You can start cloud recording, content review, etc., by listening to this event.

```
TUICallKit.on(TUICallEvent.onCallBegin, (res: any) => {  
  console.log('onCallBegin strRoomId=' + res.roomId.strRoomId + ', callMediaType='  
});
```

Parameter	Type	Description
roomId	RoomId	roomId.intRoomId: The audio and video room ID for this call (int type) roomId.strRoomId: The audio and video room ID for this call (String type)
callMediaType	Number	Media Type of the call, Video Call, Voice Call params.callMediaType = 0 : Voice Call params.callMediaType = 1 : Video Call
callRole	Number	Role, Enum Type: Caller, Called. params.callRole = 0: Unknown Type. params.callRole = 1: Caller (inviter). params.callRole = 2: Called (invitee).

onCallEnd

Indicates call disconnection. Both the caller and the called can receive it. You can listen to this event to display call duration, call type, etc., or to stop the cloud recording process.

```
TUICallKit.on(TUICallEvent.onCallEnd, (res: any) => {  
  console.log('onCallEnd strRoomId=' + res.roomId.strRoomId  
  + ', callMediaType=' + res.callMediaType  
  + ', callRole=' + res.callRole
```

```
+ ',totalTime=' + res.totalTime);
});
```

Parameter	Type	Description
roomId	RoomId	roomId.intRoomId: The audio and video room ID for this call (int type) roomId.strRoomId: The audio and video room ID for this call (String type)
callMediaType	Number	Media Type of the call, Video Call, Voice Call params.callMediaType = 0: Voice Call params.callMediaType = 1: Video Call
callRole	Number	role, Enumeration Type: Caller, Called params.callRole = 0: Unknown Type. params.callRole = 1: Caller (inviter). params.callRole = 2: Called (invitee).
totalTime	Number	The duration of this call, Unit: second

onUserReject

Callback for call rejection, in 1v1 calls, only the caller will receive the rejection callback; in group calls, all invitees can receive this callback.

```
TUICallKit.on(TUICallEvent.onUserReject, (res: any) => {
  console.log('onUserReject userId=' + res.userId);
});
```

Parameter	Type	Description
userId	String	ID of rejecting user

onUserNoResponse

Callback for no response from the other party.

```
TUICallKit.on(TUICallEvent.onUserNoResponse, (res: any) => {
  console.log('onUserNoResponse userId=' + res.userId);
});
```

Parameter	Type	Description
userId	String	User ID with no response

onUserLineBusy

Callback for call busy.

```
TUICallKit.on(TUICallEvent.onUserLineBusy, (res: any) => {  
  console.log('onUserLineBusy userId=' + res.userId);  
});
```

Parameter	Type	Description
userId	String	ID of rejecting user

onUserJoin

Callback for a user joining this call.

```
TUICallKit.on(TUICallEvent.onUserJoin, (res: any) => {  
  console.log('onUserJoin userId=' + res.userId);  
});
```

Parameter	Type	Description
userId	String	User ID joining the current call

onUserLeave

A user left the call.

```
TUICallKit.on(TUICallEvent.onUserLeave, (res: any) => {  
  console.log('onUserLeave userId=' + res.userId);  
});
```

Parameter	Type	Description
userId	String	The target userId

onCallMediaTypeChanged

Indicates a change in the call's media type.

```
TUICallKit.on(TUICallEvent.onCallMediaTypeChanged, (res: any) => {  
  console.log('onCallMediaTypeChanged oldCallMediaType=' + res.oldCallMediaType + '  
});
```

Parameter	Type	Description

oldCallMediaType	MediaType	Previous call type
newCallMediaType	MediaType	New call type

onKickedOffline

Current user kicked offline: You can notify the user on the UI and reinitialize.

```
TUICallKit.on(TUICallEvent.onKickedOffline, (res: any) => {  
  console.log('onKickedOffline');  
});
```

onUserSigExpired

Ticket expires when online: You need to generate a new userSig and reinitialize.

```
TUICallKit.on(TUICallEvent.onUserSigExpired, (res: any) => {  
  console.log('onUserSigExpired');  
});
```

onUserVideoAvailable

Whether a user is sending video.

```
TUICallKit.on(TUICallEvent.onUserVideoAvailable, (res: any) => {  
  console.log('onUserVideoAvailable userId=' + res.userId + 'isVideoAvailable=' + r  
});
```

Parameter	Type	Description
userId	String	The user ID.
isVideoAvailable	boolean	Whether the user has video.

onUserAudioAvailable

Whether a user is sending audio.

```
TUICallKit.on(TUICallEvent.onUserAudioAvailable, (res: any) => {  
  console.log('onUserAudioAvailable userId=' + res.userId + 'isAudioAvailable=' + r  
});
```

Parameter	Type	Description
userId	String	The user ID.

isAudioAvailable	boolean	Whether the user has audio.
------------------	---------	-----------------------------

onUserVoiceVolumeChanged

The volume levels of all users.

```
TUICallKit.on(TUICallEvent.onUserVoiceVolumeChanged, (res: any) => {  
  console.log('onUserVoiceVolumeChanged', res)  
});
```

Parameter	Type	Description
volumeMap	any	The volume table, which includes the volume of each user (userId). Value range: 0-100.

onUserNetworkQualityChanged

The network quality of all users.

```
TUICallKit.on(TUICallEvent.onUserNetworkQualityChanged, (networkQuality: any) => {  
  for (const [key, value] of networkQuality) {  
    console.log(`onUserNetworkQualityChanged userId: ${key}, network quality: ${value}`)  
  }  
});
```

Parameter	Type	Description
networkQuality	any	The key represents the userId, and the value represents the network quality of the user. value = Unknown value = Excellent value = Good value = Poor value = Bad value = Vbad value = Down

Definitions of Key Types

Last updated : 2025-02-26 14:17:50

callParams

In the [Tencent RTC Console](#), call extension parameters such as room number, call invitation timeout, offline push custom content, etc.

Parameter	Type	Required	Description
roomId	RoomId	No	Room ID, for details, please refer to RoomId.
timeout	Number	No	Call timeout, default: 40 seconds, unit: seconds. timeout = 0 means no timeout
userData	String	No	Custom extension fields when initiating a call
offlinePushInfo	OfflinePushInfo	No	Custom offline message push parameters
chatGroupId	String	No	chat group Id.

RoomId

Room ID, divided into String type and Number type.

Parameter	Type	Required	Description
strRoomId	String	No	Room ID, String type
intRoomId	Number	No	Room ID, String type

OfflinePushInfo

Parameter	Type	Required	Description
title	String	No	offlinepush notification title
desc	String	No	offlinepush notification description
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.
iOSSound	String	No	Offline push sound setting (only for iOS). When

			<p><code>sound =</code> <code>IOS_OFFLINE_PUSH_NO_SOUND</code> , there will be no sound played when the message is received. When <code>sound =</code> <code>IOS_OFFLINE_PUSH_DEFAULT_SOUND</code> , the system sound will be played when the message is received. If you want to customize the iOSSound, you need to link the audio file into the Xcode project first, and then set the audio file name (with extension) to the iOSSound.</p>
androidSound	String	No	<p>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 <code>setAndroidFCMChannelID</code> to set the channelID for it to take effect.</p>
androidOPPOChannelID	String	No	<p>Set the channel ID for OPPO phones with Android 8.0 and above systems.</p>
androidVIVOClassification	Number	No	<p>Classification of VIVO push messages (deprecated interface, VIVO push service will optimize message classification rules on April 3, 2023. It is recommended to use <code>setAndroidVIVOCategory</code> to set the message category). 0: Operational messages, 1: System messages. The default value is 1.</p>
androidXiaoMiChannelID	String	No	<p>Set the channel ID for Xiaomi phones with Android 8.0 and above systems.</p>
androidFCMChannelID	String	No	<p>Set the channel ID for google phones with Android 8.0 and above systems.</p>
androidHuaWeiCategory	String	No	<p>Classification of Huawei push messages, please refer to: Huawei message classification standard.</p>
isDisablePush	boolean	No	<p>Whether to turn off push notifications</p>

			(default is on).
iOSPushType	IOSOfflinePushType	No	iOS offline push type, default is APNs

IOSOfflinePushType

iOS offline push type

Parameter	Description
APNs	APNs
VoIP	VoIP

MediaType

Call media type

Parameter	Description
Audio	Audio call
Video	Video call

Camera

Front or Back Camera

Parameter	Description
Front	Front camera
Back	Rear camera

AudioPlaybackDevice

Audio route

Parameter	Description
Earpiece	Audio route is earpiece
Speakerphone	Audio route is Speakerphone

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:

--	--

Parameter	Description
Fill	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	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.

Parameter	Description
Rotation_0	No rotation
Rotation_90	Clockwise rotation by 90 degrees
Rotation_180	Clockwise rotation by 180 degrees
Rotation_270	Clockwise rotation by 0 degrees

Resolution

Video resolution

Parameter	Description
Resolution_640_360	Aspect ratio: 16:9 ; resolution: 640x360 ; recommended bitrate: 500kbps
Resolution_640_480	Aspect ratio: 4:3 ; resolution: 640x480 ; recommended bitrate: 600kbps
Resolution_960_540	Aspect ratio: 16:9 ; resolution: 960x540 ; recommended bitrate: 850kbps
Resolution_960_720	Aspect ratio: 4:3 ; resolution: 960x720 ; recommended bitrate: 1000kbps
Resolution_1280_720	Aspect ratio: 16:9 ; resolution: 1280x720 ; recommended bitrate: 1200kbps
Resolution_1920_1080	Aspect ratio: 16:9 ; resolution: 1920x1080 ; recommended bitrate: 2000kbps

ResolutionMode

Video aspect ratio mode

Parameter	Description
Landscape	Landscape resolution, such as Resolution.Resolution_640_360 +

	ResolutionMode.Landscape = 640 × 360.
Portrait	Portrait resolution, such as Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640.

ErrorCode(TUICallKit)

Last updated : 2025-06-17 12:02:51

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.

Note :

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 reenale the camera, restart the camera, or update the configuration program.
ERR_CAMERA_NOT_AUTHORIZED	-1314	No permission to access to the camera. This usually occurs on mobile devices and may be because the user denied access.
ERR_CAMERA_SET_PARAM_FAIL	-1315	Incorrect camera parameter settings (unsupported

		values or others).
ERR_CAMERA_OCCUPY	-1316	The camera is being used. Try another camera.
ERR_MIC_START_FAIL	-1302	Failed to turn the mic on. This may occur when there is a problem with the mic configuration program (driver) on Windows or macOS. Disable and reenale the mic, restart the mic, or update the configuration program.
ERR_MIC_NOT_AUTHORIZED	-1317	No permission to access to the mic. This usually occurs on mobile devices and may be because the user denied access.
ERR_MIC_SET_PARAM_FAIL	-1318	Failed to set mic parameters.
ERR_MIC_OCCUPY	-1319	The mic is being used. The mic cannot be turned on when, for example, the user is having a call on the mobile device.
ERR_TRTC_ENTER_ROOM_FAILED	-3301	Failed to enter the room. For the reason, refer to the error message for -3301.

Note :

More TRTC SDK error codes are available at : [TRTC Error Code](#)

CallKit Server Error Code

New version of Callkit RestAPI error codes, for example, when retrieving call status and call records. Common error codes are as follows:

Error Code	Description
101001	Internal server error, please try again.
101002	Invalid parameter, check if parameters meet requirements. Refer to error message for specific invalid field.
101004	Call does not exist or has already ended.
101050	Call record does not exist.

Release Notes (TUICallKit)

Web

Last updated : 2025-04-10 15:25:42

Note:

1. Version 4.0.2 involves changes to the TUICallEngine interface:

For users who need to upgrade to the calls interface, please pay attention to the version grayscale rollout. If you use the calls interface to make a call and the recipient's version is not upgraded, it will cause the call to fail.

2. TUICallKit Vue3 Github version address: [Github TUICallKit Web](#).

Version 4.0.4 @2025.03.31

New features

Added onCallNotConnected event.

Added onUserInviting event.

Added chatGroupID field in ON_CALL_RECEIVED, CALLING_END, and ON_CALL_BEGIN events.

Added online troubleshooting logs.

Feature Optimization

Optimized multi-device login issues.

Optimized the room dismissal logic via TRTC REST API.

Optimized local log printing.

Bug Fixes

Fixed log reporting issues (missing fields caused no records in Kibana).

Fixed the invalid setSelfInfo API issue.

Fixed the abnormal capability bit verification code issue in calls API.

Fixed the issue where onCallCanceled event was not triggered when all callees were busy during outgoing calls.

Removed the userLeave event triggered by TRTC room exit.

Version 4.0.3 @2025.03.13

Bug Fixes

When all callees are busy and the engine does not emit a cancel event, CallKit terminates the call based on the remote user list.

Version 4.0.2 @2025.02.18

New features

Added calls API, supporting initiating one-to-one calls or group calls, with more flexible call member management and more powerful REST API capabilities. Welcome to use it.

Added join API, working with calls API to support joining existing calls.

Version 3.3.9 @2024.10.21

Feature Optimization

Component text can be dynamically changed after language switching.

Bug Fixes

Fix the issue of resetting front/rear camera when closing camera.

Fix the problem where call interface disappears when inviteUser API reports an error during call.

Version 3.3.8 @2024.09.27

New features

Add theme switching functionality to React CallKit.

Feature Optimization

Optimize the parent element mounting for device list in React CallKit.

Version 3.3.7 @2024.09.13

Bug Fixes

Fixed the issue where the speaker remains off in the next call after being turned off in the first call.

Version 3.3.6 @2024.09.13

New features

Added CallMessage component in React for integrating on-screen message display with chat.

Feature Optimization

Optimized the on-screen message display logic to align with WeChat's message display.

Version 3.3.4 @2024.09.10

New features

React added onMinimized Callback Event.

Feature Optimization

React supports Floating Window Drag and Drop.

Optimized Mobile Terminal interaction for clicking the Speaker button, adjusted to turn on/off the Speaker.

Optimized React Device List hierarchy.

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 : 2025-03-11 10:40:37

The new version (2.9.0.1230) involves API changes for TUICallEngine:

If you encounter compilation errors after updating the Android code to the latest version during development, you can resolve them as follows:

Replace "com.tencent.qcloud.tuikit.tuicallengine" with "com.tencent.cloud.tuikit.engine.call".

Replace "com.tencent.qcloud.tuikit.TUICommonDefine" with
"com.tencent.cloud.tuikit.engine.common.TUICommonDefine".

The filename remains unchanged after the upgrade. If there are errors with other files, please re-import them yourself. If you encounter compilation errors after performing a pod update during development, you can resolve them as follows:

Replace "import TUICallEngine" with "import RTCRoomEngine".

For users who need to upgrade to the Calls API, please note the version grayscale. If you use the Calls API to make a call and the other party's version has not been upgraded, it will lead to a call failure.

Version 2.9.0.1230 @ 2025.2.18

New Features

Android & iOS: Added [calls](#) API, supporting single-person or multiple calls, more flexible call member management, and more powerful REST API. Welcome to use it.

Android & iOS: Added [join](#) API, which works with the Calls API to support joining existing calls.

Android & iOS: Support for incoming call vibrations.

Version 2.7.0.1130 @ 2024.11.27

New Features

Android&iOS : Added Traditional Chinese, optimized user experience.

Bug Fixes

Android: Fixed the issue of FCM push notifications not popping up.

Android: Fixed the issue of joining a voice call mid-way and not being able to retrieve the screen of users with the camera on.

iOS: Fixed the issue of other callee's abnormal view when caller entering in float window mode before answering.

iOS: Fixed the issue of camera status inconsistency after turning off the camera on the waiting page and then answering page.

Version 2.6.0.1080 @ 2024.9.29

Feature optimization

Android: Support landscape layout on large screen devices to enhance user experience.

iOS: Adaptation for iOS 16 and above, supports floating window outside the application.

Android: Supports floating window outside the application when “Display over other apps” permission is granted.

Bug Fixes

Android & iOS: Fixed no callback issue after initiating a call in rare scenarios.

Android & iOS: Fixed the issue of the hangup interface callback occurring twice.

iOS: Fixed the issue where switching the audio playback device was unavailable during VoIP usage.

iOS: Fixed the issue where the ringtone does not sound when entering the app after receiving an incoming call in active background status.

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 : 2025-02-27 16:09:56

For the comprehensive update log of Flutter, please visit the [changelog](#) on pub.dev.

Notes for version upgrade 2.9.0:

For users who need to upgrade to the calls API, please note the version grayscale. If you call using the calls API and the other party's version has not been upgraded, it will lead to a call failure.

FAQs(TUICallKit)

iOS

Last updated : 2025-04-17 16:28:40

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.

How to modify TUICallKit source code?

To import the component using `CocoaPods` , follow these steps:

1. Create a **TUICallKit** folder in the same directory as your project's **Podfile**.
2. Click to enter [Github/TUICallKit](#) , select Clone/Download code, and then copy the [TUICallKit_Swift](#) folder and [TUICallKit_Swift.podspec](#) file in the **iOS** directory to the **TUICallKit** folder you created in Step 1.
3. Add the following dependencies to your `Podfile` .

```
# :path => "Directed TUICallKit-Swift.podspec's phase path"
pod 'TUICallKit_Swift', :path => "TUICallKit/TUICallKit_Swift.podspec"
```

4. Execute the **pod install** command to complete the import.

Note:

The `TUICallKit_Swift` folder and `TUICallKit_Swift.podspec` file must be in the same directory.

After integrating the `TUICallKit_Swift` component, the effect is as follows:

Note :

After integrating the TUICallKit_Swift component, it supports hierarchical folder display, making it easier for you to read and modify the source code.

Xcode 15 compiler error?

- 1、Sandbox: rsync is displayed.

You can set **User Script Sandboxing** to **NO** in **Build Settings**:

2、 If SDK does not contain, compile error screenshot.

Add the following code to the Podfile:

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

post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = '13.0'
    end
  end
end
```

3、 If you run the simulator on an M-series computer, Linker command failed with exit code 1 (use-v to see invocation) may appear.

Add the following code to the Podfile:

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

post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['EXCLUDED_ARCHS[sdk=iphonesimulator*]'] = "arm64"
    end
  end
end
```

Is there a conflict between TUICallKit and the integrated audio and video library?

Tencent Cloud's audio and video libraries cannot be integrated at the same time, and there may be symbol conflicts. You can handle it according to the following scenarios.

1. If you are using the `TXLiteAVSDK_TRTC` library, there will be no symbol conflicts. You can directly add dependencies in the Podfile file.

```
pod 'TUICallKit_Swift'
```

2. If you are using the `TXLiteAVSDK_Professional` library, there will be symbol conflicts. You can add dependencies in the Podfile file.

```
pod 'TUICallKit_Swift/Professional'
```

3. If you are using the `TXLiteAVSDK_Enterprise` library, there will be symbol conflicts. It is recommended to upgrade to `TXLiteAVSDK_Professional` and then use `TUICallKit_Swift/Professional` .

Does TUICallKit support background operation?

Yes, if you need to continue running related functions in the background, you can select the current project, and in the Background Modes section under Capabilities, check **Audio, AirPlay, and Picture in Picture**, as shown in the following image:

How to view TRTC logs?

TRTC logs are compressed and encrypted by default, with the extension .xlog. Whether the log is encrypted can be controlled by `setLogCompressEnabled`. The file name containing C(compressed) is encrypted and compressed, and the file name containing R(raw) is plaintext.

iOS : Sandbox's `Documents/log`

Note:

To view the .xlog file, you need to download the [decryption tool](#) and run it directly in the Python 2.7 environment with the xlog file in the same directory using `python decode_mars_log_file.py`.

To view the .clog file (new log format after version 9.6), you need to download the [decryption tool](#) and run it directly in the Python 2.7 environment with the clog file in the same directory using `python decompress_clog.py`.

Android

Last updated : 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.

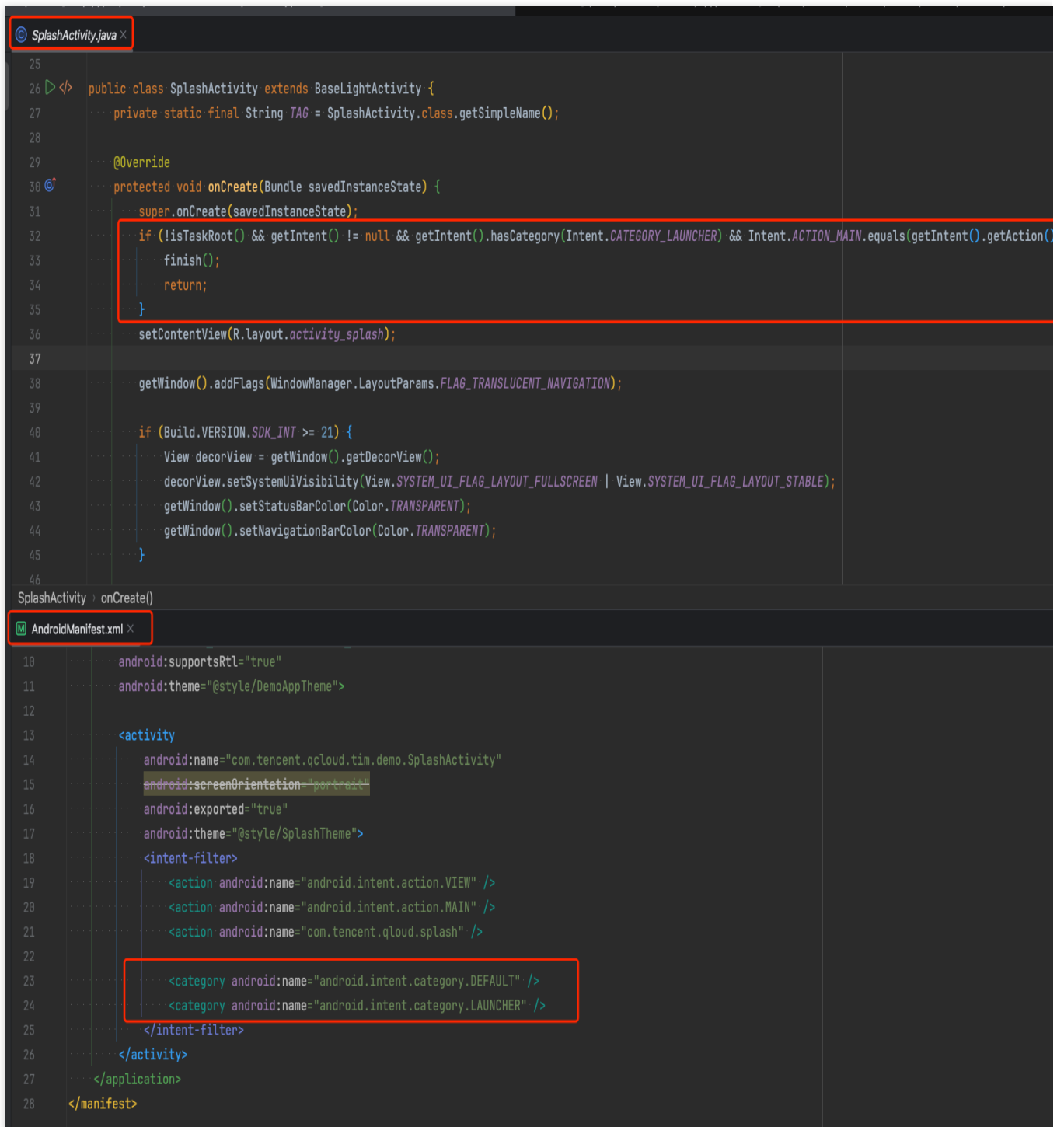
Scenario Three: When the app is in the background and an offline push notification is received, not clicking the push notification but entering the app via the desktop icon will either fail to bring up the CallKitActivity or cause the CallKitActivity to flash momentarily.

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

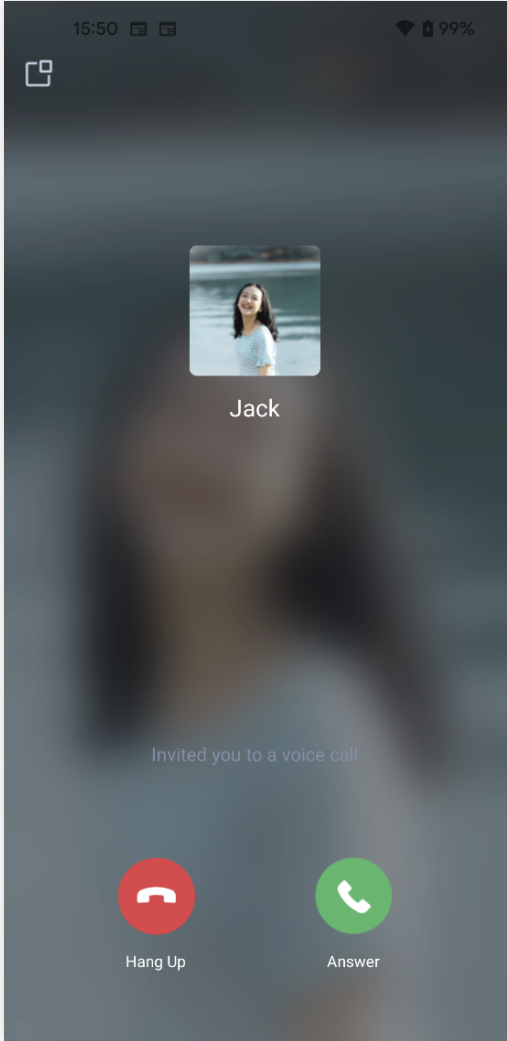
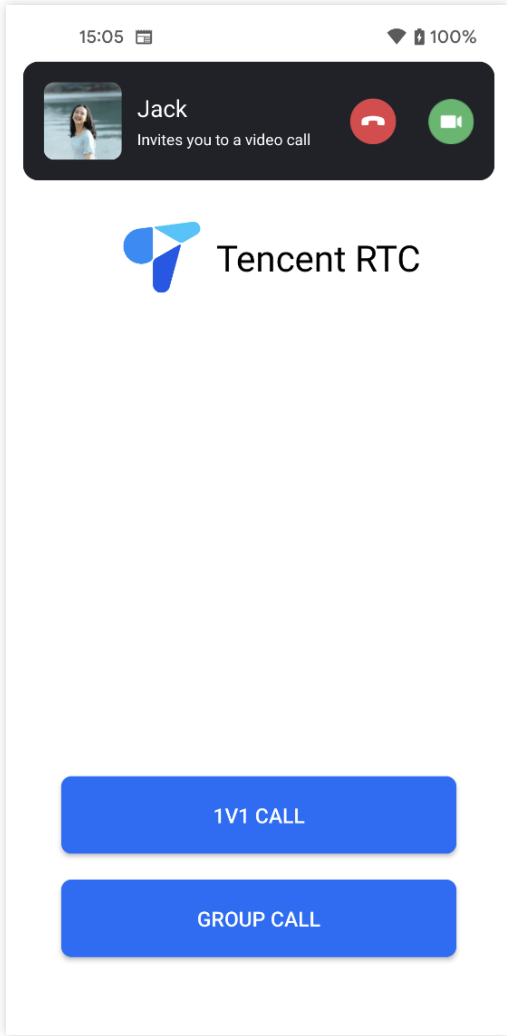
Location

```
if (!isTaskRoot() && getIntent() != null && getIntent().hasCategory(Intent.CATEGORY
    && Intent.ACTION_MAIN.equals(getIntent().getAction())) {
    finish();
    return;
}
```

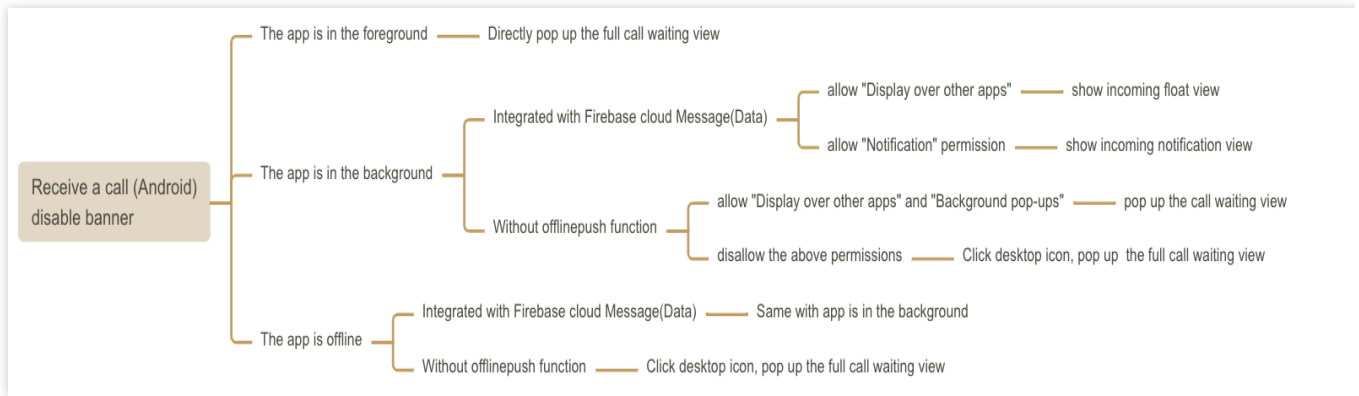


Incoming call display policy for the callee

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
 A full-screen mobile app interface for an incoming voice call. At the top, the status bar shows 15:50, signal strength, and 99% battery. A small square icon is in the top-left corner. The background is a blurred image of a person. In the center, there is a square profile picture of a woman, with the name 'Jack' below it. Below the name, the text 'Invited you to a voice call' is displayed. At the bottom, there are two large circular buttons: a red one with a white phone receiver icon labeled 'Hang Up' and a green one with a white phone receiver icon labeled 'Answer'.	 A mobile app interface showing an incoming call as a banner at the top. The status bar shows 15:05, signal strength, and 100% battery. The banner contains a small profile picture of a woman, the name 'Jack', and the text 'Invites you to a video call'. To the right of the text are two circular icons: a red one with a white phone receiver icon and a green one with a white video camera icon. Below the banner, the Tencent RTC logo is displayed. At the bottom, there are two blue rectangular buttons: '1V1 CALL' and 'GROUP CALL'.

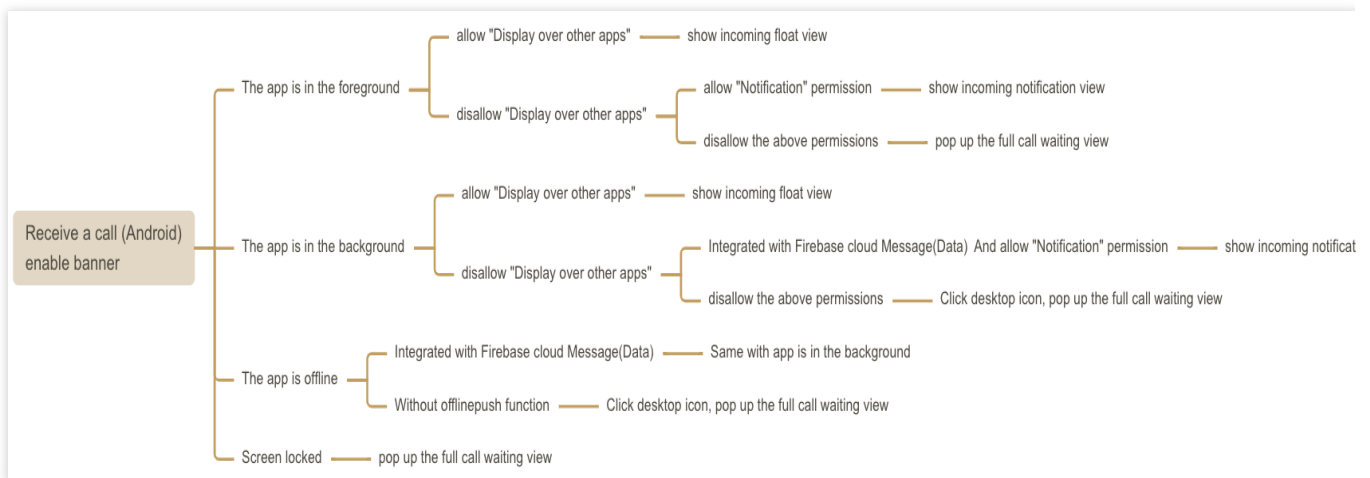
1. If you want the callee to pull up the full-screen call view when receiving a call, you can integrate the [tuicallkit-kt](#) code. By default, the incoming call display strategy for the callee is as follows:



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
<div><div>App info</div><div><div></div><div><div>TUICallKit</div><div>Version 1.0</div></div></div><div><div>Notifications</div><div>~0 notifications per week</div><div>></div></div><div><div>Permissions</div><div>No permissions granted</div><div>></div></div><div><div>Storage</div><div>52.80 MB used in internal storage</div><div>></div></div><div><div>Data usage</div><div>No data used</div><div>></div></div><div><div>Battery</div><div>No battery use since last full charge</div><div>></div></div><div><div>Open by default</div><div>No defaults set</div><div>></div></div><div><div>Advanced</div><div></div><div></div></div><div><div>Display over other apps</div><div>Allowed</div><div>></div></div><div><div></div><div></div></div><div><div>Uninstall</div><div>Force stop</div></div></div>	<div><div><</div><div>TUICallKit</div></div> <div><div>Device management</div><div></div></div> <div><div>Autostart</div><div>Activated in the background</div><div>></div></div> <div><div>Associated startup</div><div>></div></div> <div><div>Floating window</div><div>></div></div> <div><div>Home screen shortcuts</div><div>Ask every time</div></div> <div><div>Display on lock screen</div><div>></div></div> <div><div>Background pop-ups</div><div>></div></div> <div><div>Camera</div><div></div></div> <div><div>Use camera</div><div>While using the app</div></div> <div><div>Microphone</div><div></div></div> <div><div>Record</div><div>While using the app</div></div>

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;
}
```

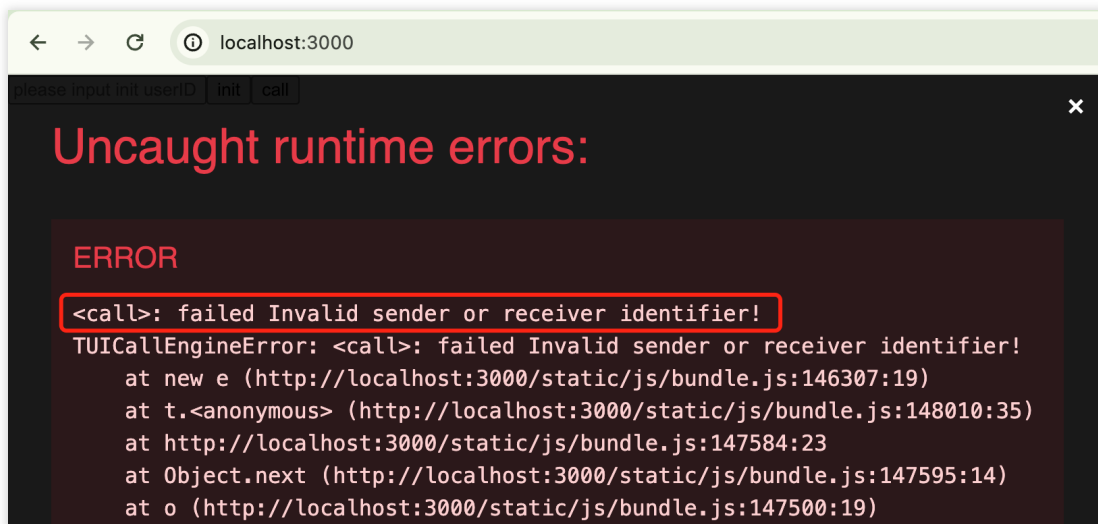
How do I create a userID?

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.

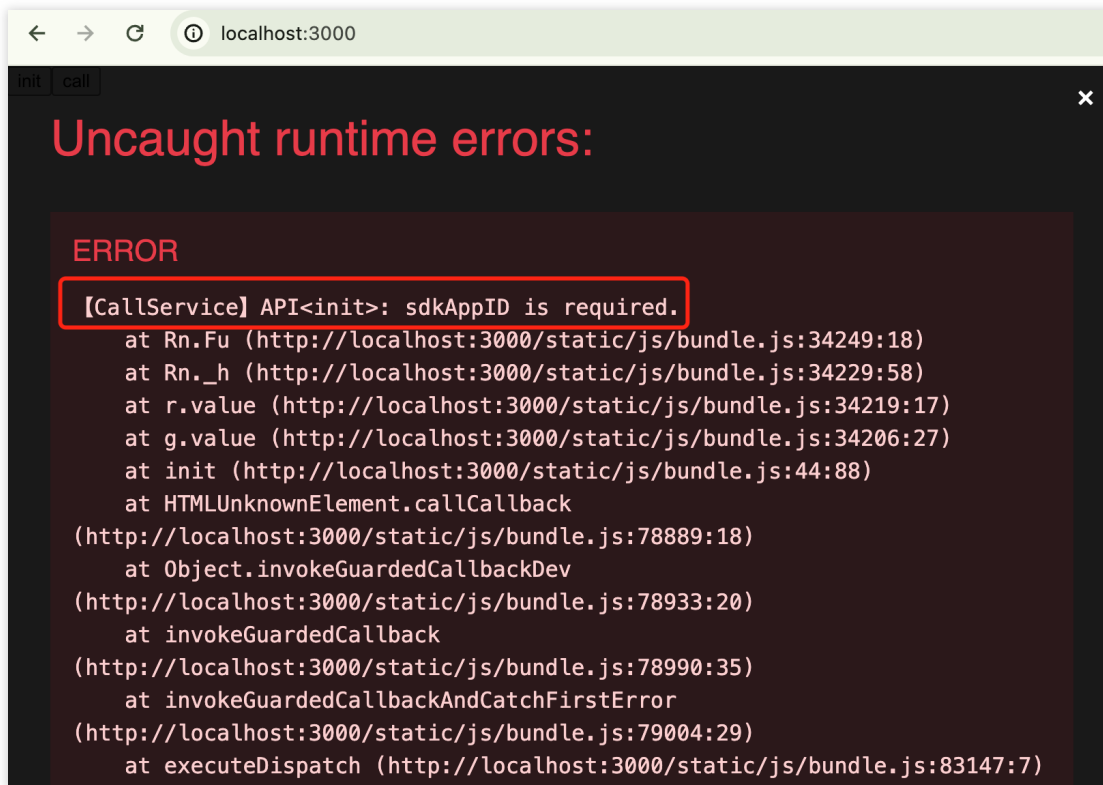
Create and get through the [Tencent RTC Console](#).

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 ?

```
natural@NATURALYUAN-MB4 ~/Desktop npm install -g create-react-app
npm ERR! code EACCES
npm ERR! syscall rename
npm ERR! path /usr/local/lib/node_modules/create-react-app
npm ERR! dest /usr/local/lib/node_modules/.create-react-app-DgI96EzL
npm ERR! errno -13
npm ERR! Error: EACCES: permission denied, rename '/usr/local/lib/node_modules/create-react-app' -> '/usr/local/lib/node_modules/.create-react-app-DgI96EzL'
npm ERR! [Error: EACCES: permission denied, rename '/usr/local/lib/node_modules/create-react-app' -> '/usr/local/lib/node_modules/.create-react-app-DgI96EzL'] {
npm ERR!   errno: -13,
npm ERR!   code: 'EACCES',
npm ERR!   syscall: 'rename',
npm ERR!   path: '/usr/local/lib/node_modules/create-react-app',
npm ERR!   dest: '/usr/local/lib/node_modules/.create-react-app-DgI96EzL'
npm ERR! }
npm ERR!
npm ERR! The operation was rejected by your operating system.
npm ERR! It is likely you do not have the permissions to access this file as the current user
npm ERR!
npm ERR! If you believe this might be a permissions issue, please double-check the
npm ERR! permissions of the file and its containing directories, or try running
npm ERR! the command again as root/Administrator.
npm ERR! A complete log of this run can be found in:
npm ERR! /Users/natural/.npm/_logs/2024-05-24T04_22_13_347Z-debug-0.log
natural@NATURALYUAN-MB4 ~/Desktop sudo npm install -g create-react-app SIG(115) 9083 12
```

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 ?

```
natural@NATURALYUAN-MB4 ~/Desktop npm install -g @vue/cli
npm ERR! code EACCES
npm ERR! syscall rename
npm ERR! path /usr/local/lib/node_modules/@vue/cli
npm ERR! dest /usr/local/lib/node_modules/@vue/.cli-pSsrUrBg
npm ERR! errno -13
npm ERR! Error: EACCES: permission denied, rename '/usr/local/lib/node_modules/@vue/cli' -> '/usr/local/lib/node_modules/@vue/.cli-pSsrUrBg'
npm ERR! [Error: EACCES: permission denied, rename '/usr/local/lib/node_modules/@vue/cli' -> '/usr/local/lib/node_modules/@vue/.cli-pSsrUrBg'
npm ERR!   errno: -13,
npm ERR!   code: 'EACCES',
npm ERR!   syscall: 'rename',
npm ERR!   path: '/usr/local/lib/node_modules/@vue/cli',
npm ERR!   dest: '/usr/local/lib/node_modules/@vue/.cli-pSsrUrBg'
npm ERR! }
npm ERR!
npm ERR! The operation was rejected by your operating system.
npm ERR! It is likely you do not have the permissions to access this file as the current user
npm ERR!
npm ERR! If you believe this might be a permissions issue, please double-check the
npm ERR! permissions of the file and its containing directories, or try running
npm ERR! the command again as root/Administrator.
npm ERR! A complete log of this run can be found in:
npm ERR! /Users/natural/.npm/_logs/2024-05-24T04_25_32_580Z-debug-0.log
natural@NATURALYUAN-MB4 ~/Desktop sudo npm install -g @vue/cli
```

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 : 2025-06-16 11:23:04

1. Error message [-100035]The package you purchased does not support this ability. Please purchase the package appears during voice or video call ?

To make voice or video calls, you need to subscribe to our Call Monthly Package. You can try it for free for 7 days. For more information on how to activate the service, please refer to: [Activate Service \(TUICallKit\)](#).

2. Invalid or non-existent UserID of message sender or receiver ?

If the user you are calling has not logged in, the call will fail. Please try calling a user who has successfully logged in.

3. Error: inviteID is invalid or the invitation has been processed ?

The inviteID may be invalid if it has been cancelled by the caller or if the invitation has already been accepted. In some cases, network bandwidth may also affect signaling, causing inviteID invalid errors.

4. How to get the call room number RoomId?

After the call is connected, you can obtain the call room number (RoomId) through the [onCallBegin](#) return roomid field.

5. Can TUICallKit be used without importing the Chat SDK?

No, all components of TUICallKit use Tencent Cloud Chat SDK as the communication base service, including call signaling, busy line signaling, and other core logic. If you have already purchased another Chat product, you can also refer to the TUICallKit logic for adaptation.

6. in the TUIChat + TUICallKit Integrated Scenarios, after Upgrading the TUICallKit Version, New Version Calls Old Version and Old Version Is Unable to Receive Call Issues

Starting from TUICallKit version 3.1, in the TUIChat + TUICallKit integrated scenarios, initiating a call through the call button in TUIChat by default uses the new calls API. Therefore, if your business has already gone live, you can use the advanced API to switch the call interface to call or groupCall to maintain interconnectivity with users of the online edition.

You need to call an advanced API once after successful log-in. Specific use on all platforms is as follows:

iOS(Swift)

iOS(Objective-C)

Android(java)

Android(kotlin)

Flutter(Dart)

Web & Mini Programs


```
let jsonParams: [String: Any] = ["api": "forceUseV2API",
                                "params": ["enable": true],]
guard let data = try? JSONSerialization.data(withJSONObject: jsonParams, options: J
guard let paramsString = NSString(data: data, encoding: String.Encoding.utf8.rawValue

TUICallKit.createInstance().callExperimentalAPI(jsonStr: paramsString)

NSMutableDictionary *jsonParams = @{@"api": @"forceUseV2API",
                                    @"params": @{@"enable": @YES}};
NSError *error = nil;
NSData *data = [NSJSONSerialization dataWithJSONObject:jsonParams options:0 error:&
if (error || !data) { return; }
NSString *paramsString = [[NSString alloc] initWithData:data encoding:NSUTF8StringE
if (!paramsString) { return;}
[[TUICallKit createInstance] callExperimentalAPIWithJsonStr:paramsString];

try {
    JSONObject params = new JSONObject();
    params.put("enable", true);
    JSONObject jsonObject = new JSONObject();
    jsonObject.put("api", "forceUseV2API");
    jsonObject.put("params", params);
    TUICallKit.createInstance(context).callExperimentalAPI(jsonObject.toString());
} catch (Exception e) {
    e.printStackTrace();
}

try {
    val params = JSONObject()
    params.put("enable", true)

    val jsonObject = JSONObject()
    jsonObject.put("api", "forceUseV2API")
    jsonObject.put("params", params)
    TUICallKit.createInstance(context).callExperimentalAPI(jsonObject.toString())
} catch (e: Exception) {
    e.printStackTrace()
}

final jsonParams = {'api': 'forceUseV2API',
```

```
        'params': {'enable': true} }];

try {
    final jsonString = json.encode(jsonParams);
    TUICallKit.instance.callExperimentalAPI(jsonStr: jsonString);
} catch (e) {
    return;
}

const jsonObject = {
    api: "forceUseV2API",
    params: {
        enable: true,
    }
};
TUICallKitServer.getTUICallEngineInstance().callExperimentalAPI(JSON.stringify(jsonObject));
```

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?

Details: When introducing flutter "tencent_calls_uikit" into our existing project, Android builds APK with the following error:

```
Duplicate class com.tencent.liteav.LiveSettingJni found in modules jetified-LiteAVS
(com.tencent.liteav:LiteAVSDK_Professional:10.7.0.13053) and jetified-LiteAVSDK_TRT
(com.tencent.liteav:LiteAVSDK_TRTC:10.3.0.11225)
```

The following error occurs when `pod install` is executed on iOS:

```
[!] The 'Pods-Runner' target has frameworks with conflicting names: txsoundtouch.xc
```

The 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 `MaterialApp`. The purpose is to navigate to the `TUICallKit` page when you receive a call invitation. Example code is as follows:

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

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

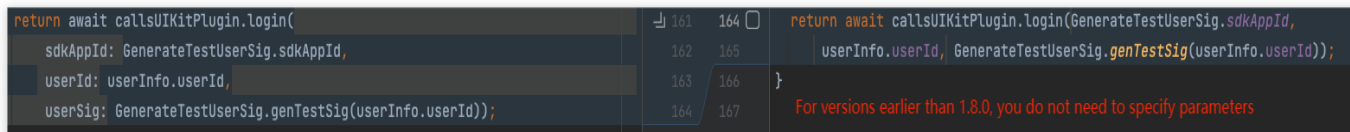
2. `tencent_calls_engine` Import files in plug-ins are uniformly replaced with new ones.

```
import 'package:tencent_calls_engine/tuicall_engine.dart';
import 'package:tencent_calls_engine/tuicall_observer.dart';
import 'package:tencent_calls_engine/tuicall_define.dart';
```

The above code block content is replaced with the following:

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

3. The login API adjustment is more standardized and no parameters need to be specified.



The screenshot shows a code diff comparing two versions of the `login` method call. The left side shows the old code with multiple parameters: `sdAppId`, `userId`, and `userSig`. The right side shows the new code with a simplified signature: `sdAppId`, `userInfo.userId`, and `GenerateTestUserSig.genTestSig(userInfo.userId)`. A note at the bottom right states: "For versions earlier than 1.8.0, you do not need to specify parameters".

```
return await callsUIKitPlugin.login(
  sdAppId: GenerateTestUserSig.sdAppId,
  userId: userInfo.userId,
  userSig: GenerateTestUserSig.genTestSig(userInfo.userId));
}

return await callsUIKitPlugin.login(GenerateTestUserSig.sdAppId,
  userInfo.userId, GenerateTestUserSig.genTestSig(userInfo.userId));
}

For versions earlier than 1.8.0, you do not need to specify parameters
```

4. Optimization of offline push parameter construction.