Tencent Real-Time Communication

Building Demo

Product Documentation

Copyright Notice

Trademark Notice

Tencent Cloud

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

Building Demo

Web Interactive Live Streaming

Web-Based Interactive Live Streaming

Last updated : 2022-04-02 17:02:04

This document introduces you to our web-based interactive live streaming solutions `TUIPusher` and `TUIPlayer` (UI included). You can integrate them into Tencent Cloud's basic SDKs such as TRTC and IM to quickly equip your live streaming applications (corporate live streaming, live shopping, vocational training, remote teaching, etc.) with web-based publishing and playback capabilities.

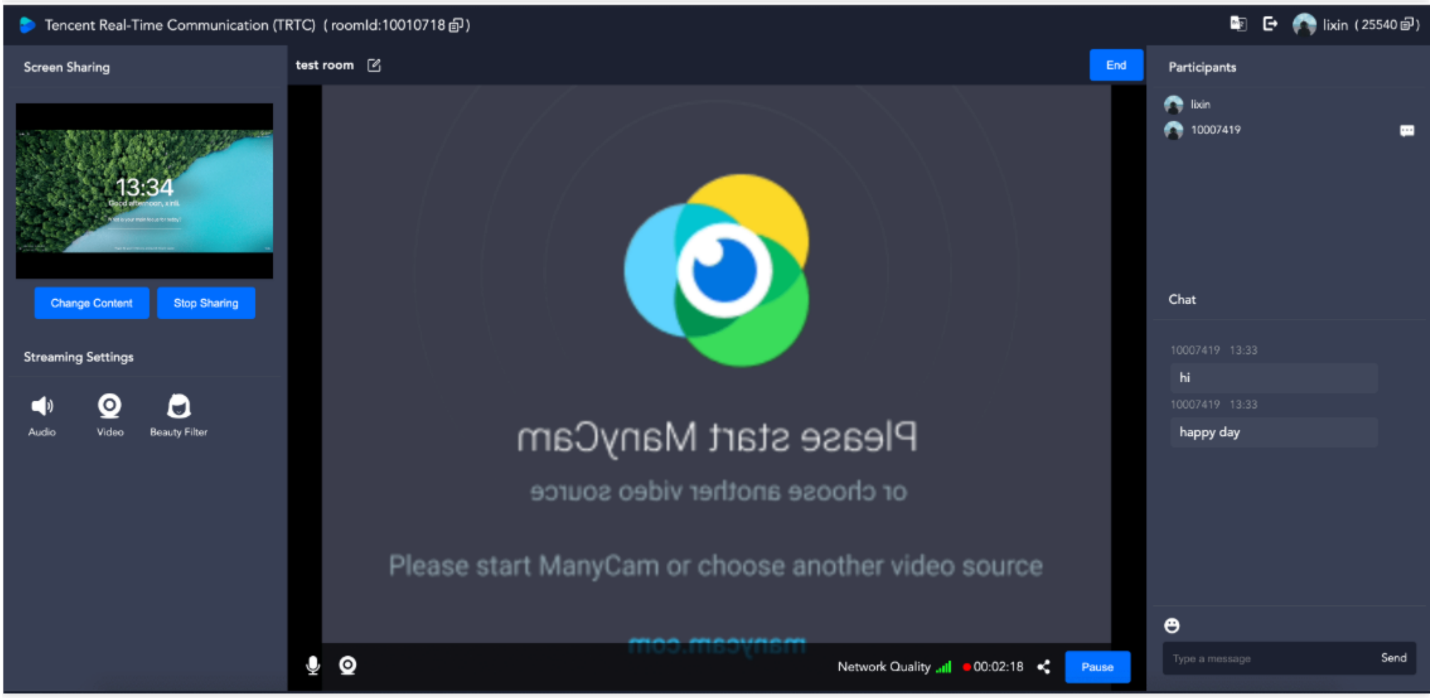
Strengths:

- A general-purpose live streaming solution with UI that includes common live streaming features such as device selection, beauty filters, publishing, playback, and live chat, helping you quickly bring your services to the market
- Easy integration into Tencent Cloud's basic SDKs, including TRTC, IM, and TCPlayer, for excellent flexibility and scalability
- Web-based, easy-to-use, and quick updates

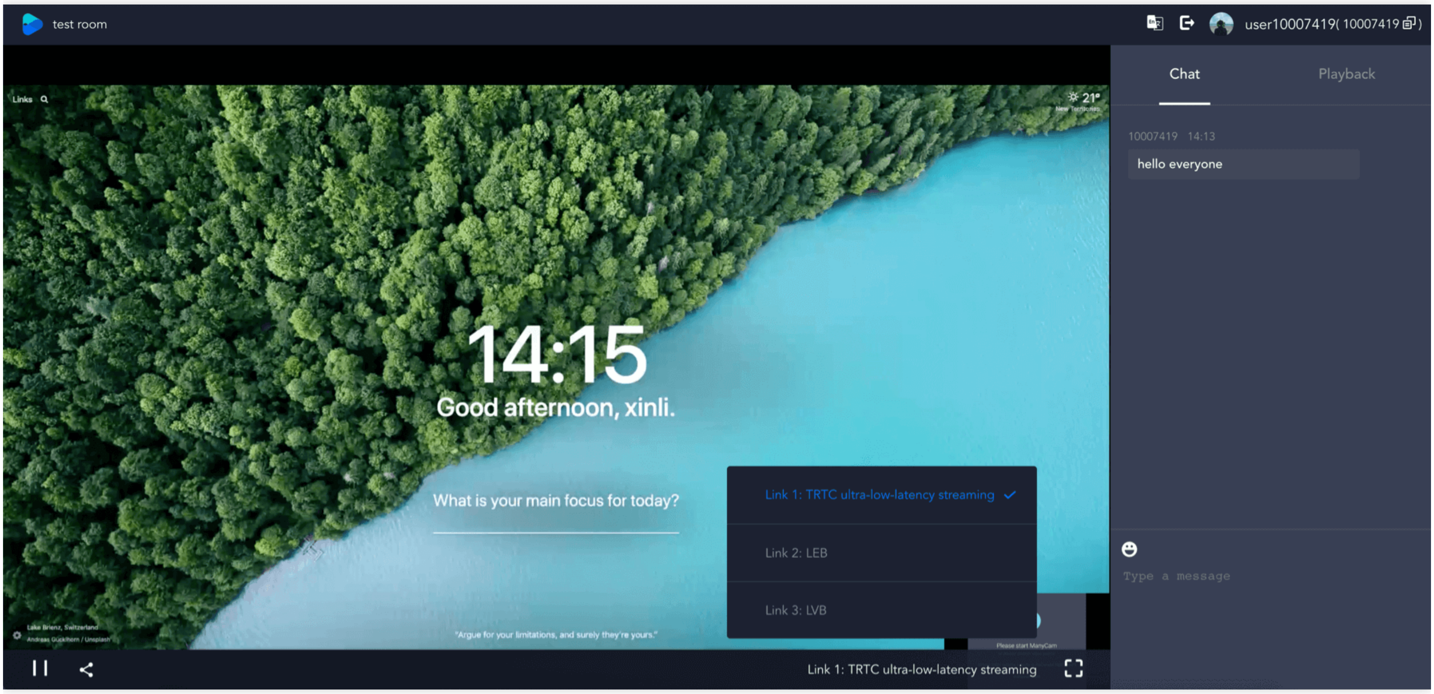
Demos

See below for a demonstration of the components' features. We also provide a TUIPusher Demo and a TUIPlayer Demo, with user and room management systems, for you to experiment with the features.

> Note :
> You need to log in with two different accounts to try `TUIPusher` and `TUIPlayer` at the same time.



10007419 13:33
hi
10007419 13:33
happy day


test room
user10007419(10007419)
Chat
Playback
14:15
Good afternoon, xinli.
What is your main focus for today?
Link 1: TRTC ultra-low-latency streaming ✓
Link 2: LEB
Link 3: LVB
Link 1: TRTC ultra-low-latency streaming
Type a message

10007419 14:13
hello everyone

Download

You can download `TUIPusher` and `TUIPlayer` from the following links:

- TUIPusher
- TUIPlayer

Overview

TUIPusher

- Capturing and publishing streams from camera and mic
  - Customizing video parameters including frame rate, resolution, and bitrate
  - Applying beauty filters and setting beauty filter parameters
- Capturing and publishing data from the screen
- Publishing to the TRTC backend and Tencent Cloud’s CDNs
- Text chatting with the anchor and other audience members
- Getting the audience list and muting audience members

TUIPlayer

- Playing the audio/video stream and screen sharing stream at the same time
- Text chatting with the anchor and other audience members
- Three playback options: ultra-low-latency live streaming (300 ms latency), high-speed live streaming (latency within 1,000 ms), and standard live streaming (ultra-high concurrency)
- Supports desktop and mobile browsers and landscape mode on mobile devices

> Note :
>
> If your browser does not support WebRTC and can play videos only using standard live streaming protocols, please use a different browser to try WebRTC playback.

Integration

Notes

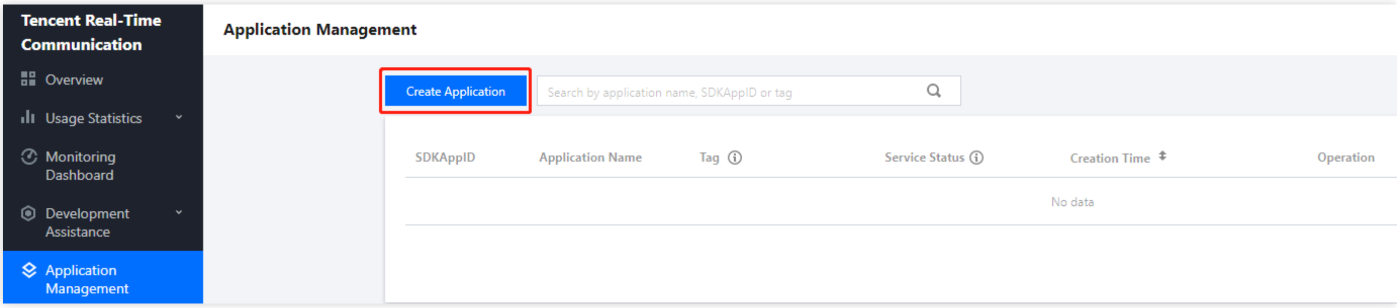
- `TUIPusher` and `TUIPlayer` are based on TRTC and IM. Make sure you use the same `SDKAppID` for your TRTC and IM applications so that they can share your account and authentication information.
- You can use the basic-edition content filtering capability of IM to filter chat messages. If you want to customize restricted words, go to the IM console > **Content Filtering**, and click **Upgrade**.
- Local `UserSig` calculation is for development and local debugging only and not for official launch. The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see Generating UserSig.

Step 1. Create an application

- Method 1: via TRTC
- Method 2: via IM

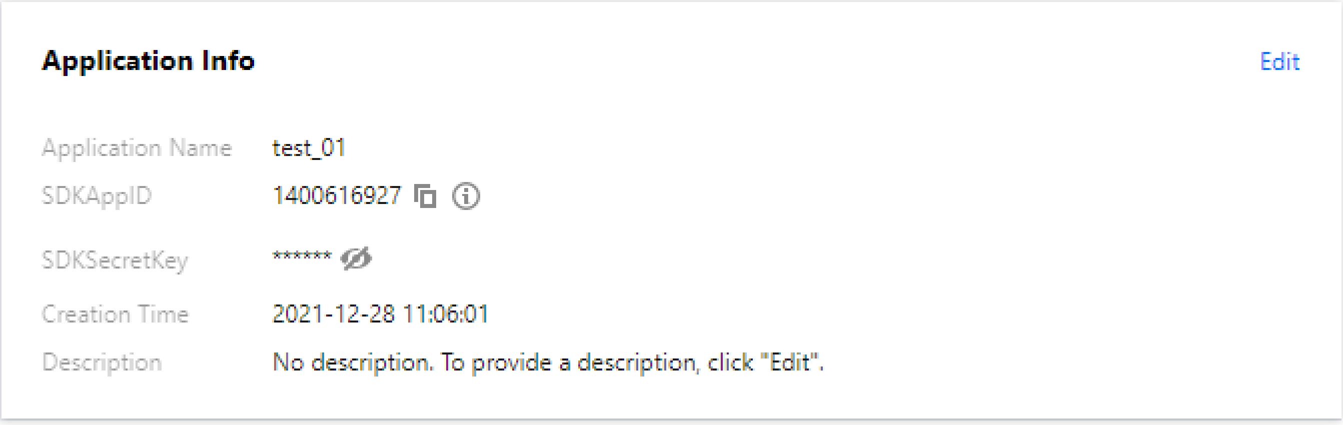
Step 1. Create a TRTC application

1. Sign up for a Tencent Cloud account and activate TRTC and IM.
2. In the TRTC console, click **Application Management > Create Application** to create an application.



Step 2. Get the TRTC key information

1. In the application list, find the application created and click **Application Info** to view the `SDKAppID` .


Application Info

Edit

Application Name test_01

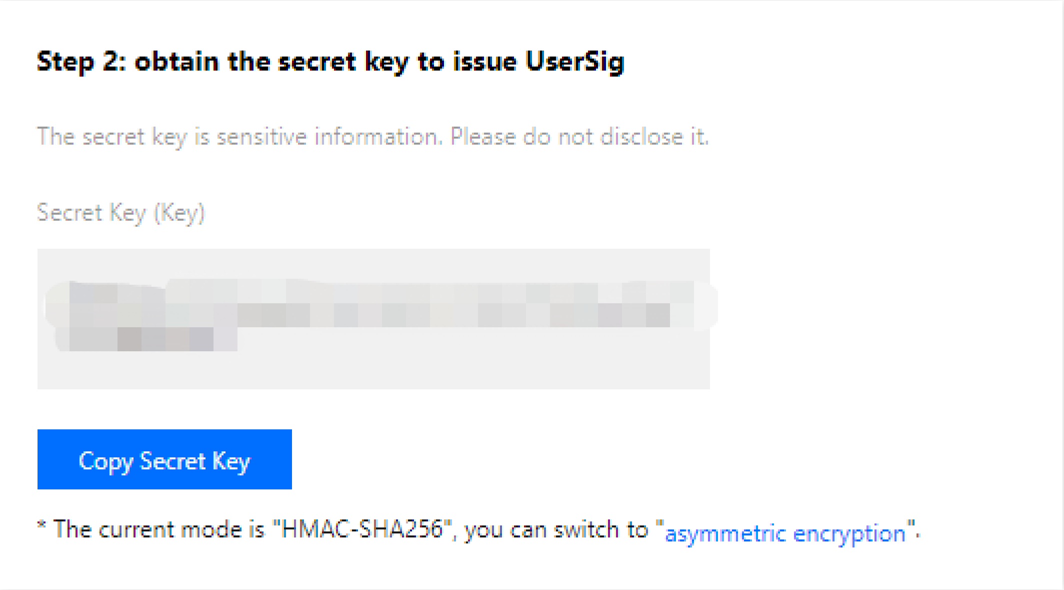
SDKAppID 1400616927

SDKSecretKey ******

Creation Time 2021-12-28 11:06:01

Description No description. To provide a description, click "Edit".

2. Select the **Quick Start** tab to view the application’s secret key.



Step 2: obtain the secret key to issue UserSig

The secret key is sensitive information. Please do not disclose it.

Secret Key (Key)

[redacted]

* The current mode is "HMAC-SHA256", you can switch to "asymmetric encryption".

Note

- Accounts creating their first application in the TRTC console will get a 10,000-minute free trial package.
- After you create a TRTC application, an IM application with the same `SDKAppID` will be created automatically. You can configure package information for the application in the IM console.

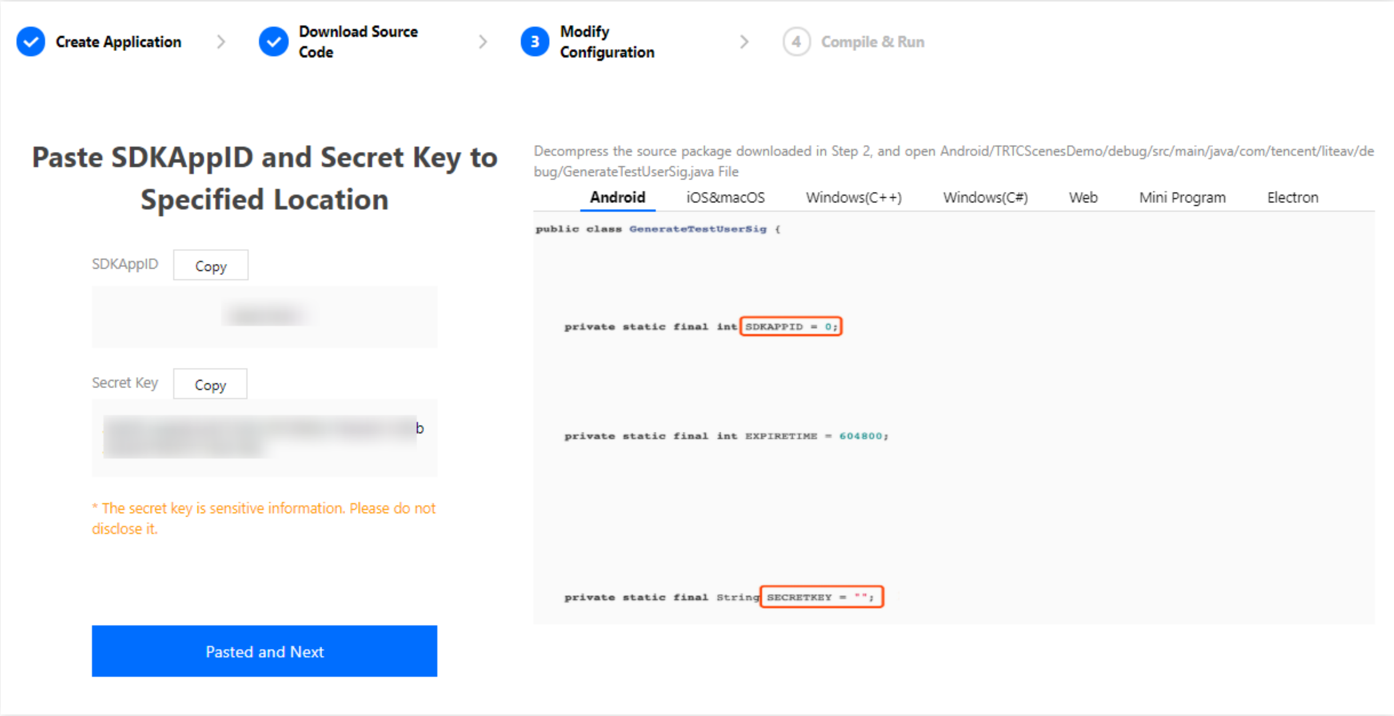
Step 2. Prepare your project

1. Download the code for `TUIPusher` and `TUIPlayer` at GitHub.

2. Install dependencies for `TUIPusher` and `TUIPlayer` .

```
cd Web/TUIPusher
npm install
cd Web/TUIPlayer
npm install
```

3. Paste `SDKAppID` and the secret key to the specified locations below in the `TUIPusher/src/config/basic-info-config.js` and `TUIPlayer/src/config/basic-info-config.js` files.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Pasted and Next

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Run `TUIPusher` and `TUIPlayer` in a local development environment.

```
cd Web/TUIPusher
npm run serve
cd Web/TUIPlayer
npm run serve
```

5. You can open `http://localhost:8080` and `http://localhost:8081` to try out the features of `TUIPusher` and `TUIPlayer` .

6. You can modify the room, anchor, and audience information in `TUIPusher/src/config/basic-info-config.js` and `TUIPlayer/src/config/basic-info-config.js` , but **make sure the room and anchor information is consistent in the two files**.

Note :

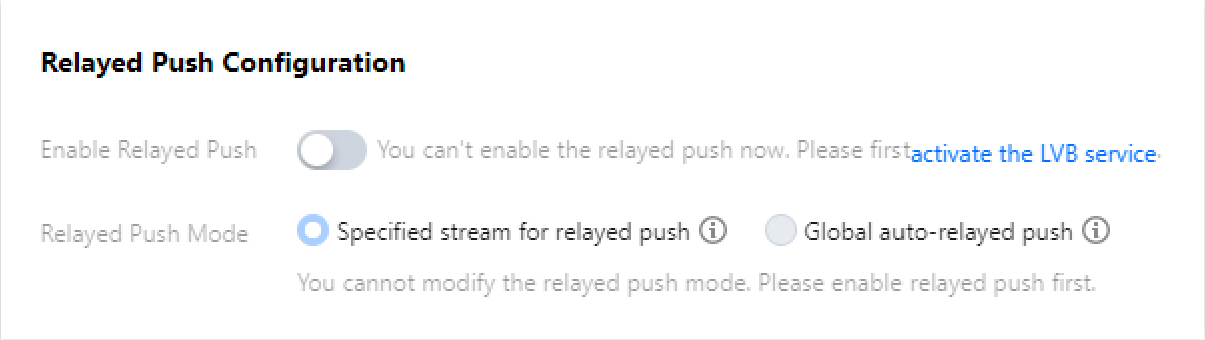
- You can now use `TUIPusher` and `TUIPlayer` for ultra-low-latency live streaming. If you want to support high-speed and standard live streaming too, see Step 3. Enable relayed push.
- Local calculation of `UserSig` is for development and local debugging only and not for official launch. If your `SECRETKEY` is leaked, attackers will be able to steal your Tencent Cloud traffic.

- The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 3. Enable relayed push

Because the high-speed and standard live streaming features of `TUIPusher` and `TUIPlayer` are powered by CSS, you need to enable relayed push to use these features.

1. In the TRTC console, enable relayed push for your application. You can choose **Specified stream for relayed push** or **Global auto-relayed push** based on your needs.



Relayed Push Configuration

Enable Relayed Push: You can't enable the relayed push now. Please first activate the LVB service.

Relayed Push Mode: ○ Specified stream for relayed push ○ Global auto-relayed push

You cannot modify the relayed push mode. Please enable relayed push first.

2. On the **Domain Management** page, add your playback domain name. For detailed directions, please see Adding Your Own Domain Names.
3. Configure the playback domain name in `TUIPlayer/src/config/basic-info-config.js`.

You can now use all features of `TUIPusher` and `TUIPlayer`, including ultra-low-latency live streaming, high-speed live streaming, and standard live streaming.

Step 4. Apply in a production environment

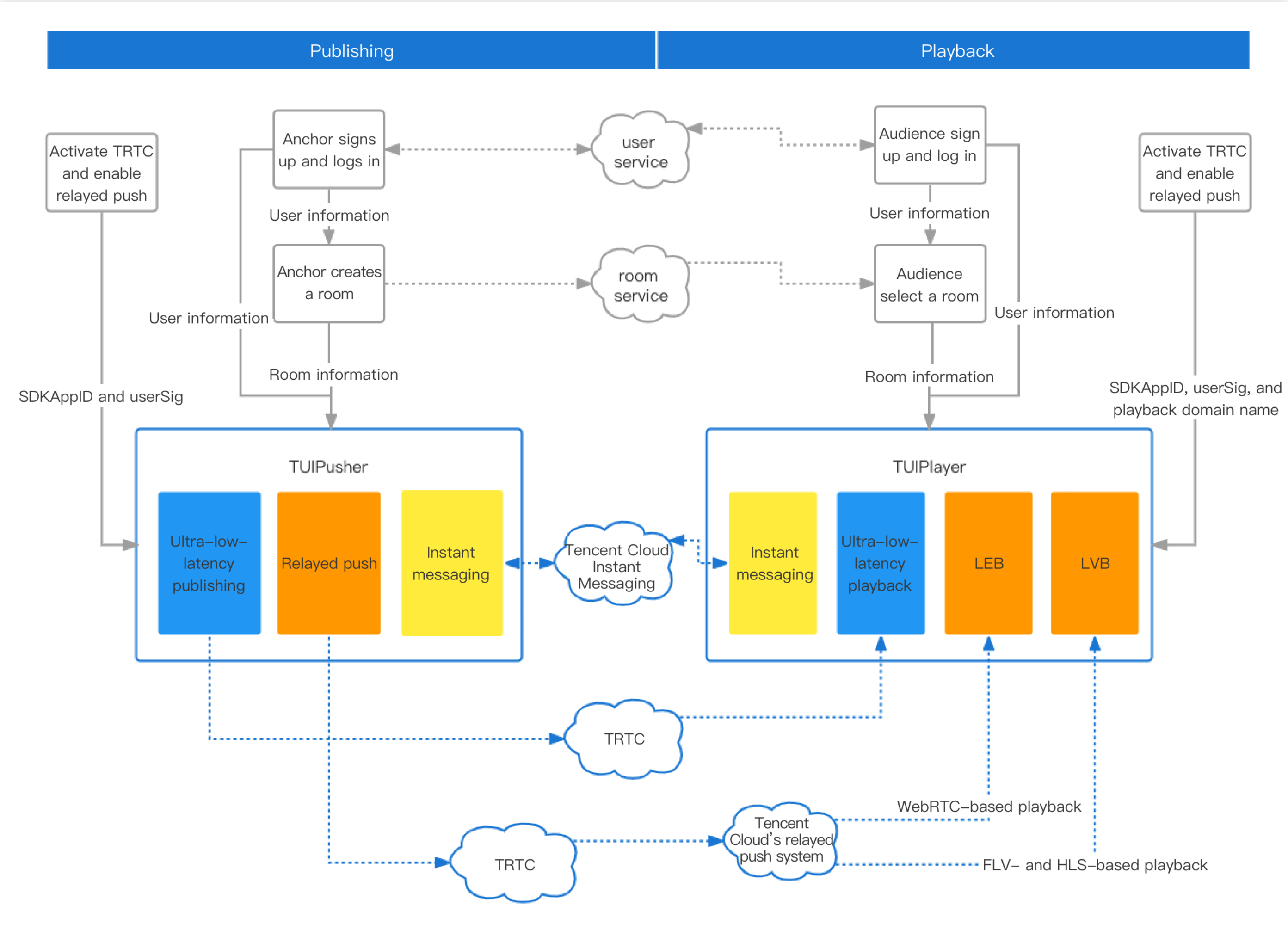
To apply `TUIPusher` and `TUIPlayer` to a production environment, in addition to integrating them into your project, you also need to do the following:

- Create a user management system to manage user information such as user IDs, usernames, and profile pictures
- Create a room management system to manage room information such as room IDs, room names, and anchors
- Generate `UserSig` on your server

Note :

- In this document, `UserSig` is generated on the client based on the `SDKAppID` and secret key you provide. The secret key may be easily decompiled and reversed, and if your key is leaked, attackers will be able to steal your traffic. Therefore, **this method is for local debugging only**.
- The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

- Submit account information such as user information, room information, `SDKAppID` , and `UserSig` to `store` of `vuex` for global storage, as in `TUIPusher/src/pusher.vue` and `TUIPlayer/src/player.vue` , and you will be able to use all features of the two components on publishing and playback clients. The diagram below shows the process in detail:


Publishing
Playback
Activate TRTC and enable relayed push
Anchor signs up and logs in
user service
Audience sign up and log in
Activate TRTC and enable relayed push
User information
User information
Anchor creates a room
room service
Audience select a room
User information
User information
SDKAppID and userSig
Room information
Room information
SDKAppID, userSig, and playback domain name
TUIPusher
Ultra-low-latency publishing
Relayed push
Instant messaging
Tencent Cloud Instant Messaging
TUIPlayer
Instant messaging
Ultra-low-latency playback
LEB
LVB
TRTC
WebRTC-based playback
Tencent Cloud's relayed push system
TRTC
FLV- and HLS-based playback

FAQs

How do I implement the beauty filter feature on the web?

See Enabling Beauty Filters.

How do I implement the screen sharing feature on the web?

See Screen Sharing.

How do I implement the on-cloud recording feature on the web?

1. For information about how to enable **on-cloud recording**, see On-Cloud Recording and Playback.
2. If you enable **specified user recording**, you can start recording on the web by specifying `userDefineRecordId` when calling the TRTC.createClient API.

How do I publish a stream to CDN on the web?

See Publishing to CDN.

How do I enable high-speed playback on the web?

Publish streams to CDNs using the TRTC Web SDK and play the streams using WebRTC.

Notes

Supported platforms

Operating System	Browser	Required Version	TUIPlayer	TUIPusher	TUIPusher Screen Sharing
macOS	Safari	11+	Supported	Supported	Supported (on Safari 13+)
macOS	Chrome	56+	Supported	Supported	Supported (on Chrome 72+)
macOS	Firefox	56+	Supported	Supported	Supported (on Firefox 66+)
macOS	Edge	80+	Supported	Supported	Supported

Operating System	Browser	Required Version	TUIPlayer	TUIPusher	TUIPusher Screen Sharing
macOS	WeChat built-in browser	-	Supported	Not supported	Not supported
macOS	WeCom built-in browser	-	Supported	Not supported	Not supported
Windows	Chrome	56+	Supported	Supported	Supported (on Chrome 72+)
Windows	QQ Browser (WebKit core)	10.4+	Supported	Supported	Not supported
Windows	Firefox	56+	Supported	Supported	Supported (on Firefox 66+)
Windows	Edge	80+	Supported	Supported	Supported
Windows	WeChat built-in browser	-	Supported	Not supported	Not supported
Windows	WeCom built-in browser	-	Supported	Not supported	Not supported
iOS	WeChat built-in browser	-	Supported	Not supported	Not supported
iOS	WeCom built-in browser	-	Supported	Not supported	Not supported
iOS	Safari	-	Supported	Not supported	Not supported
iOS	Chrome	-	Supported	Not supported	Not supported
Android	WeChat built-in browser	-	Supported	Not supported	Not supported
Android	WeCom built-in browser	-	Supported	Not supported	Not supported

Operating System	Browser	Required Version	TUIPlayer	TUIPusher	TUIPusher Screen Sharing
Android	Chrome	-	Supported	Not supported	Not supported
Android	QQ Browser	-	Supported	Not supported	Not supported
Android	Firefox	-	Supported	Not supported	Not supported
Android	UC Browser	-	Supported (only standard live streaming)	Not supported	Not supported

Domain requirements

For security and privacy reasons, only HTTPS URLs can access all features of `TUIPusher` and `TUIPlayer` . Therefore, please use the HTTPS protocol for the web page of your application in production environments.

> Note :
> Note: You can use `http://localhost` for local development.

The table below lists the supported domain names and protocols.

Scenario	Protocol	TUIPlayer	TUIPusher	TUIPusher Screen Sharing	Remarks
Production	HTTPS	Supported	Supported	Supported	Recommended
Production	HTTP	Supported	Not supported	Not supported	-
Local development	`http://localhost`	Supported	Supported	Supported	Recommended
Development	`http://127.0.0.1`	Supported	Supported	Supported	-

Scenario	Protocol	TUIPlayer	TUIPusher	TUIPusher Screen Sharing	Remarks
Local development	`http://[local IP address]`	Supported	Not supported	Not supported	-

Firewall configuration

`TUIPusher` and `TUIPlayer` rely on the following ports and domain for data transfer, which should be added to the allowlist of your firewall.

- TCP port: 8687
- UDP ports: 8000, 8080, 8800, 843, 443, 16285
- Domain name: qcloud.rtc.qq.com

Summary

In future versions, we plan to add support for communication between the web components and the TRTC SDKs for iOS, Android, etc. and introduce features such as co-anchoring, advanced filters, custom layout, relaying to multiple platforms, and image/text/music upload.

If you have any requirements or feedback, contact colleenyu@tencent.com.

Video Call

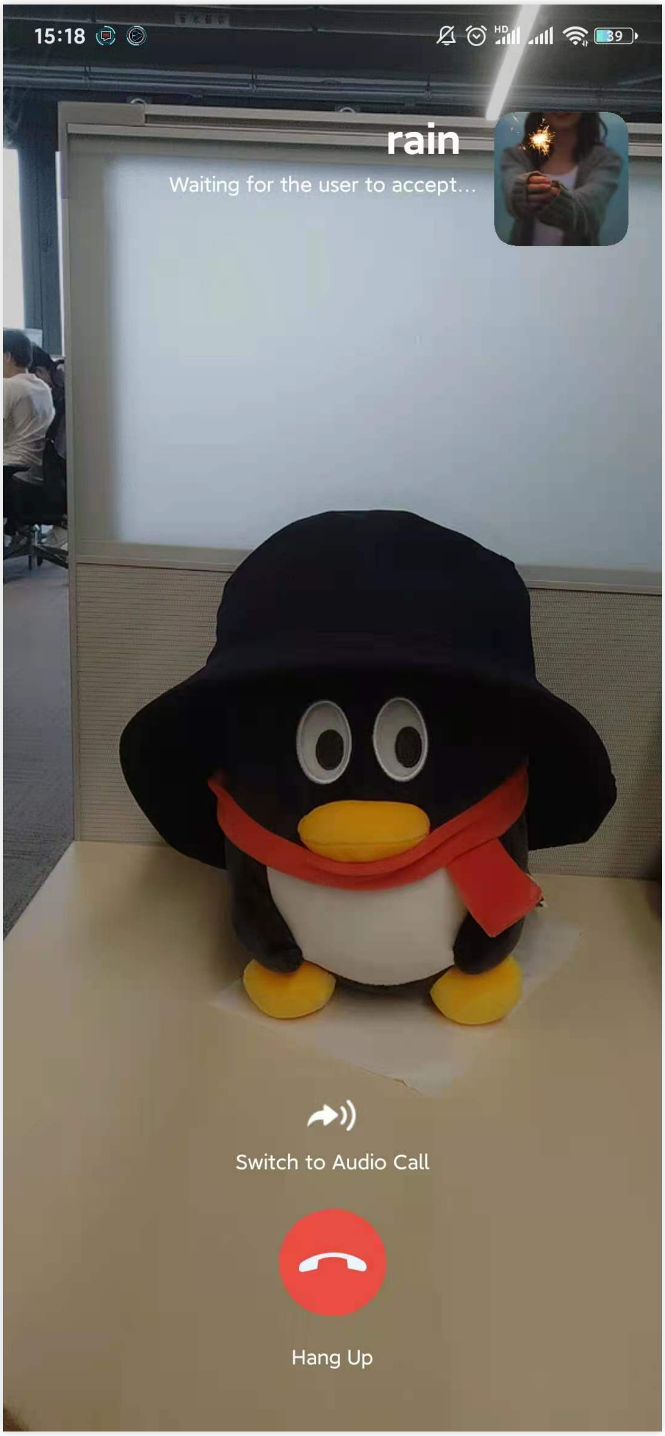
Video Call (iOS)

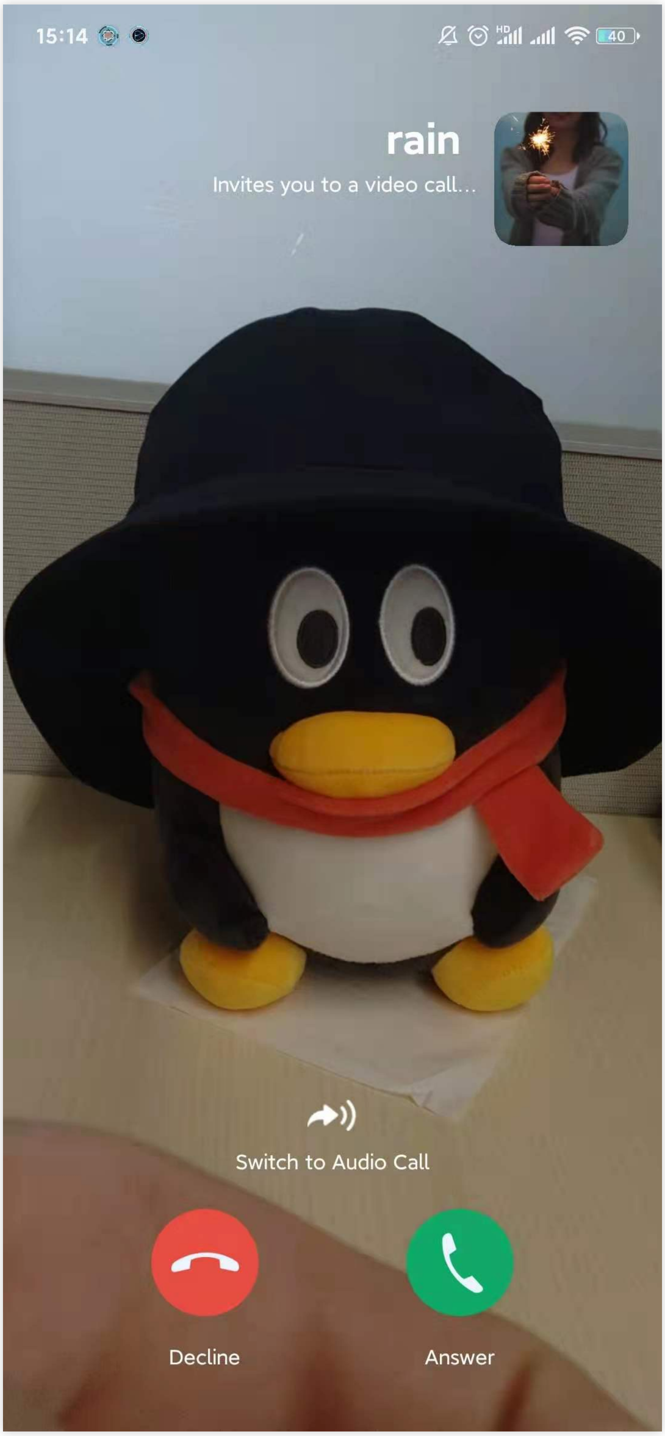
Last updated : 2022-03-17 15:38:17

Demo UI

You can download and install the app we provide to try out the real-time audio/video call feature.

Call	Answer





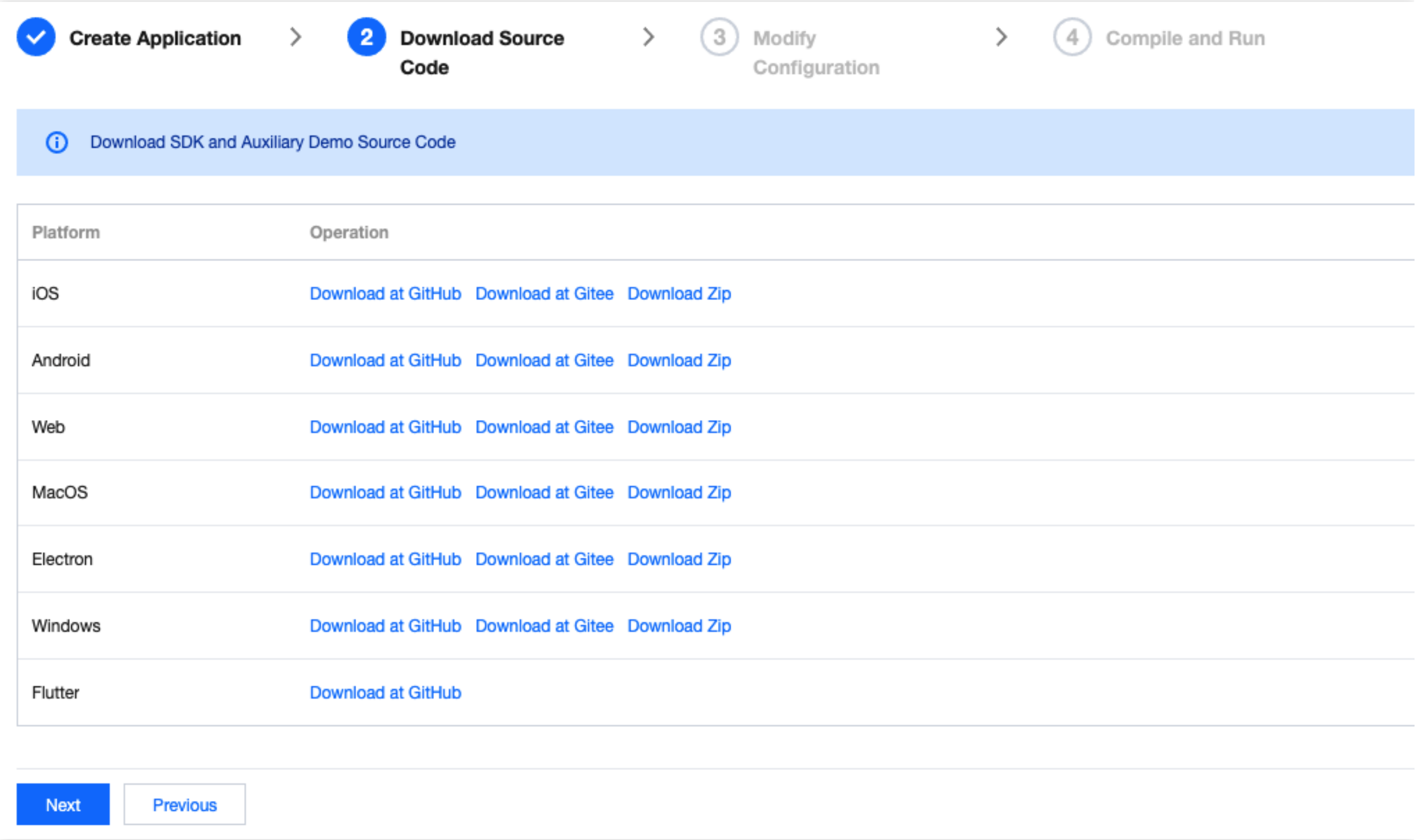
Note :

To make it easier for you to implement the real-time audio/video call feature, we have refactored the `TUICalling` component. You no longer need to pay attention to UI as it is now implemented within the `TUICalling` component.

Running the Demo

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestVideoCall` and click **Create**.
3. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Note :
This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the application source code

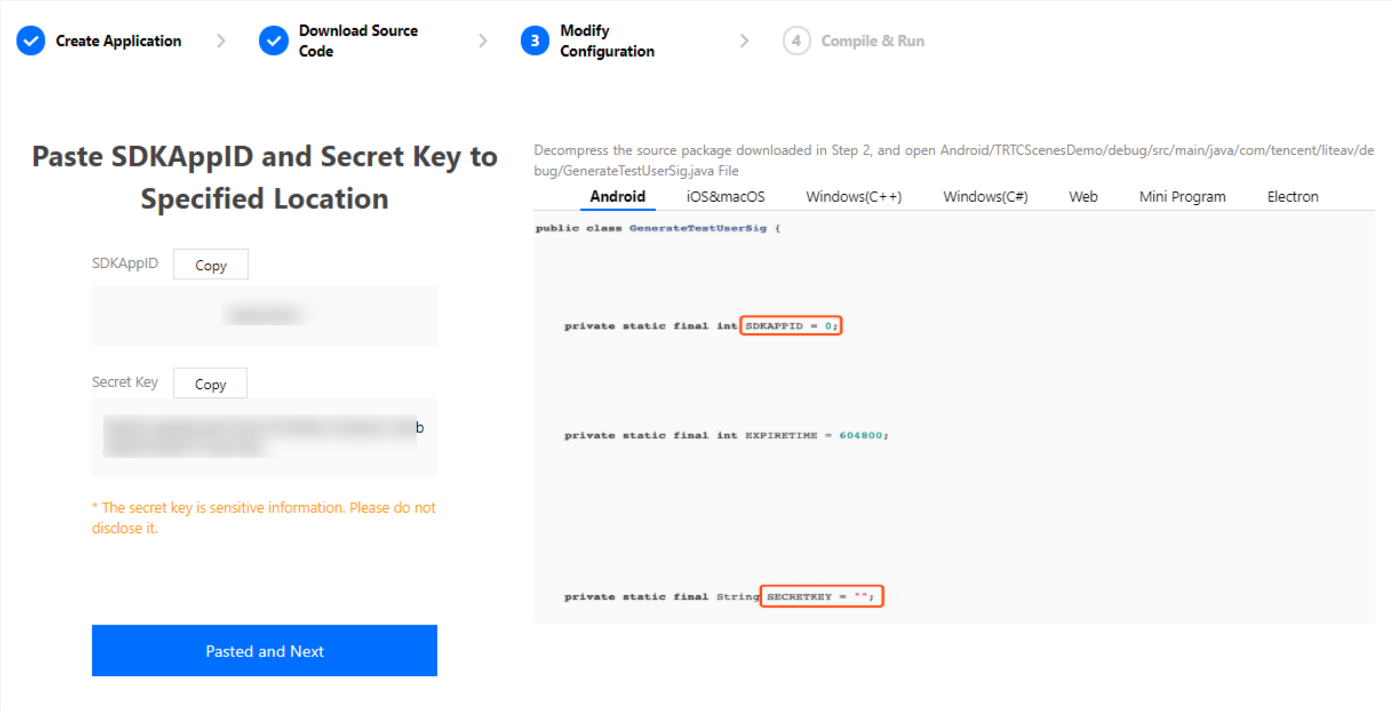
Clone or download the TUICalling source code.

Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `iOS/Example/Debug/GenerateTestUserSig.swift`.

3. Set the following parameters in `GenerateTestUserSig.swift`:
    - SDKAPPID: `0` by default. Set it to the actual `SDKAppID`.
    - SECRETKEY: left empty by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID

[redacted]

Secret Key

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {



    private static final int SDKAPPID = 0;



    private static final int EXPIRETIME = 604800;



    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note：
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your

application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUICalling/Example/TUICallingApp.xcworkspace` with Xcode (version 11.0 or above) and click **Run**.

Tryout

Note :

You need at least two devices to try out the application.

User A

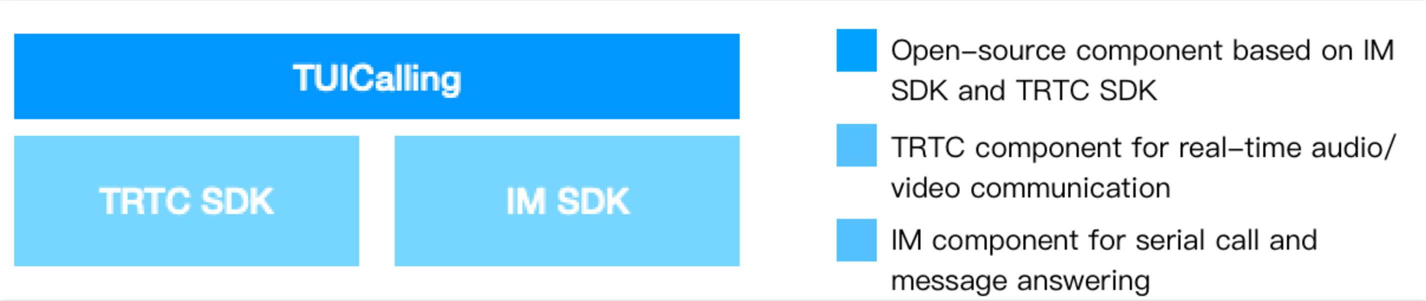
1. Enter a username (**which must be unique**) and log in.
2. Enter the `userId` of the person you want to call and tap **Search**.
3. Tap **Call** and select **Video Call** (**Make sure that the callee is active in the application, or the call may fail**).

User B

1. Enter a username (**which must be unique**) and log in.
2. Go to the homepage and wait for the incoming call.

Integration Directions

In the source code, the `Source` folder contains three subfolders: `ui`, `model`, and `Service`. The `Service` subfolder includes the open-source `TUICallingManager` component, which we share with the public. You can find the component’s APIs in the `TUICallingManager.h` file.


TUICalling
TRTC SDK
IM SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for serial call and message answering

You can enable the real-time audio/video call feature for your project with ease using the open-source `TUICalling` and `TUICallingManager` components, with no need to implement complicated call UI or logic by yourself.

Step 1. Integrate the SDKs

The call component `TRTCCalling` depends on the TRTC SDK and IM SDK. You can integrate the two SDKs into your project by following the steps below:

- **Method 1: adding dependencies via CocoaPods**

```
pod ’TXIMSDK_iOS’
pod ’TXLiteAVSDK_TRTC’
```

- **Method 2: adding dependencies through local files**
  If your access to the CocoaPods repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration Documentation
IM SDK	Download	Integration document

Step 2. Configure permission requests

Configure camera and mic permission requests by adding `Privacy - Camera Usage Description` and `Privacy - Microphone Usage Description` in `info.plist` .

Step 3. Import the `TUICalling` component

To import the component through CocoaPods, follow the steps below:

1. Copy the `Source` , `Resources` , and `TXAppBasic` folders and the `TUICalling.podspec` file under the demo project directory to your project directory.
2. Add the following dependencies to your `Podfile` and run `pod install` to complete the import.

```
pod ’TXAppBasic’, :path =&gt; ”../TXAppBasic/”
pod ’TXLiteAVSDK_TRTC’
pod ’TUICalling’, :path =&gt; ”../”, :subspecs =&gt; [”TRTC”]
```

Step 4. Initialize the component and log in

1. Call `TUICallingManager.sharedInstance()` to initialize the component.
2. Call `TUILogin.initWithSdkAppID(SDKAPPID)` to initialize login.
3. Call `TUILogin.login(userId, userSig)` to log in to the component. Specify the key parameters as described below:

Parameter	Description
sdkAppID	You can view \`SDKAppID\` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. For the calculation method, please see UserSig.

```
// Initialize the component
TUICallingManager.sharedInstance();
// Log in to the component
TUILogin.initWithSdkAppID(SDKAPPID)
TUILogin.login(userId, userSig) {
print(”login success”)
} fail: { code, errorDes in
print(”login failed, code:¥(code), error: ¥(errorDes ?? ”nil”)”)
}
```

Step 5. Make an audio/video call

1. Caller: call `call();` of `TUICallingManager` to initiate a call, passing in the user IDs (`userIDs`) and call type (`type`). For the call type, you can pass in `.audio` (audio call) or `.video` (video call). If only one user ID is passed in for `userIDs` , the call is a one-to-one call; if two or more user IDs are passed in, the call is a group call.

2. Callee: If a callee is logged in, the answering view will be displayed. If you want offline users to receive call invitations, please see Enable offline call answering.

```
// 2. Register the listener
TUICallingManager.shareInstance().setCallingListener(listener: TUICallingListerner())

// 2. Set whether to enable custom views (disabled by default)
TUICallingManager.shareInstance().enableCustomViewRoute(enable: true)

// 3. Set callbacks
public func shouldShowOnCallView() -&gt; Bool {
return true;
}

public func callStart(userIDs: [String], type: TUICallingType, role: TUICallingRole, viewController: UIViewController?) { if let vc = viewController {
callingVC = vc;
vc.modalPresentationStyle = .fullScreen

if var topController = UIApplication.shared.keyWindow?.rootViewController {
while let presentedViewController = topController.presentedViewController {
topController = presentedViewController
}

if let navigationVC = topController as? UINavigationController {
if navigationVC.viewControllers.contains(self) {
present(vc, animated: false, completion: nil)
} else {
navigationVC.popToRootViewController(animated: false)
navigationVC.pushViewController(self, animated: false)
navigationVC.present(vc, animated: false, completion: nil)
}
} else {
topController.present(vc, animated: false, completion: nil)
}
}
}
}

public func callEnd(userIDs: [String], type: TUICallingType, role: TUICallingRole, totalTime: Float) {
callingVC.dismiss(animated: true, completion: nil)
}

public func onCallEvent(event: TUICallingEvent, type: TUICallingType, role: TUICallingRole, message: String) {
}
// 4. Make a call
TUICallingManager.shareInstance().call(userIDs, .video)
```

Step 6. Enable offline call answering

> Note :
> If your project does not require the offline answering feature, for example, if it offers online customer service, your integration can end at step 5. If your project is a social networking service, we recommend you enable offline answering.

The IM SDK supports offline push, but additional configuration is required to enable the feature.

1. Apply for an Apple push certificate. For detailed directions, please see Obtaining Apple Push Notification Service Certificates.
2. Configure offline push on the backend and client. For detailed directions, please see Offline Push (iOS).
3. The offline push API has been integrated into the signaling API (`sendModel`) of `TRTCCallingImpl` . After completing the offline push configuration for your application, you will be able to send notifications to offline users.

Component APIs

The table below lists the APIs of the `TUICalling` component.

API	Description
call	Sends call invitations by user ID.
receiveAPNSCalled	Answers a call.
setCallingListener	Sets the listener.
setCallingBell	Sets the ringtone (preferably shorter than 30s).
enableMuteMode	Enables/Disables the mute mode.

API	Description
enableCustomViewRoute	Enables/Disables custom views. After enabling custom views, you will receive a `CallingView` instance in the callback for calling/being called, and can decide how to display the view by yourself. The view must be displayed full screen or in proportion to the screen size; otherwise, an error may occur.

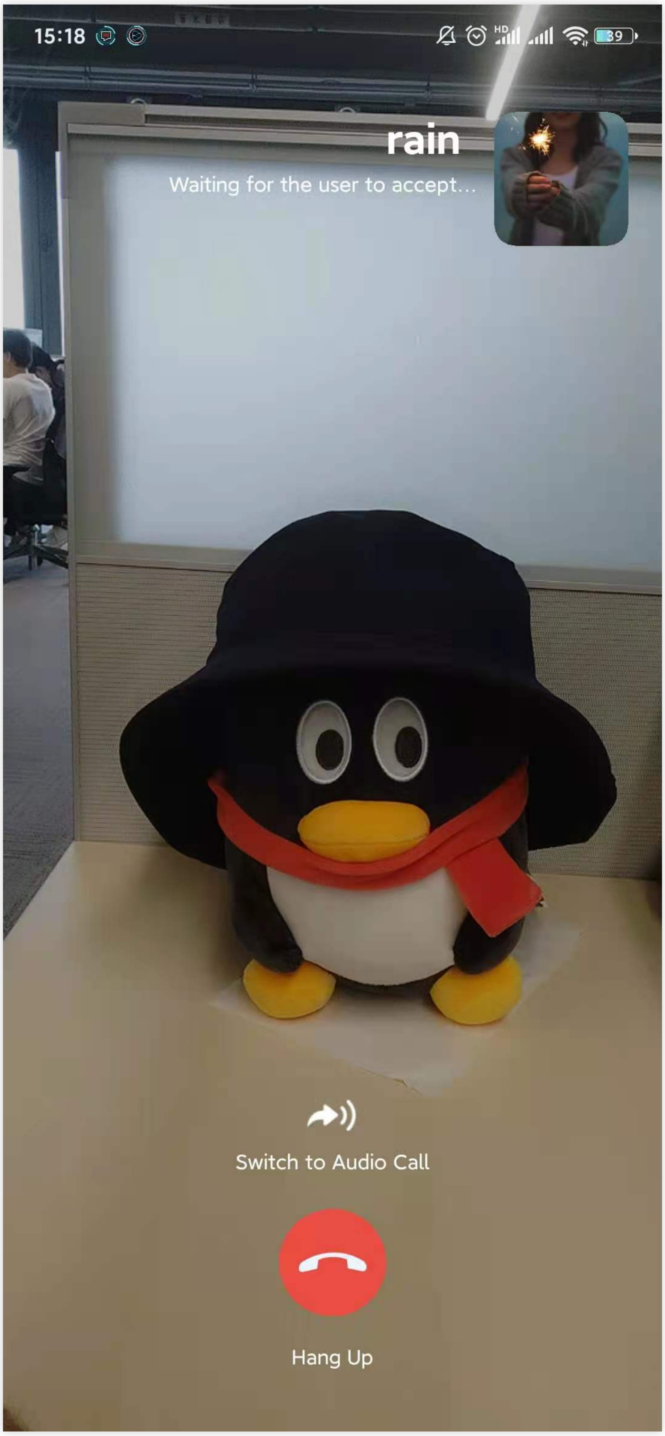
Video Call (Android)

Last updated : 2022-03-17 15:38:39

Demo UI

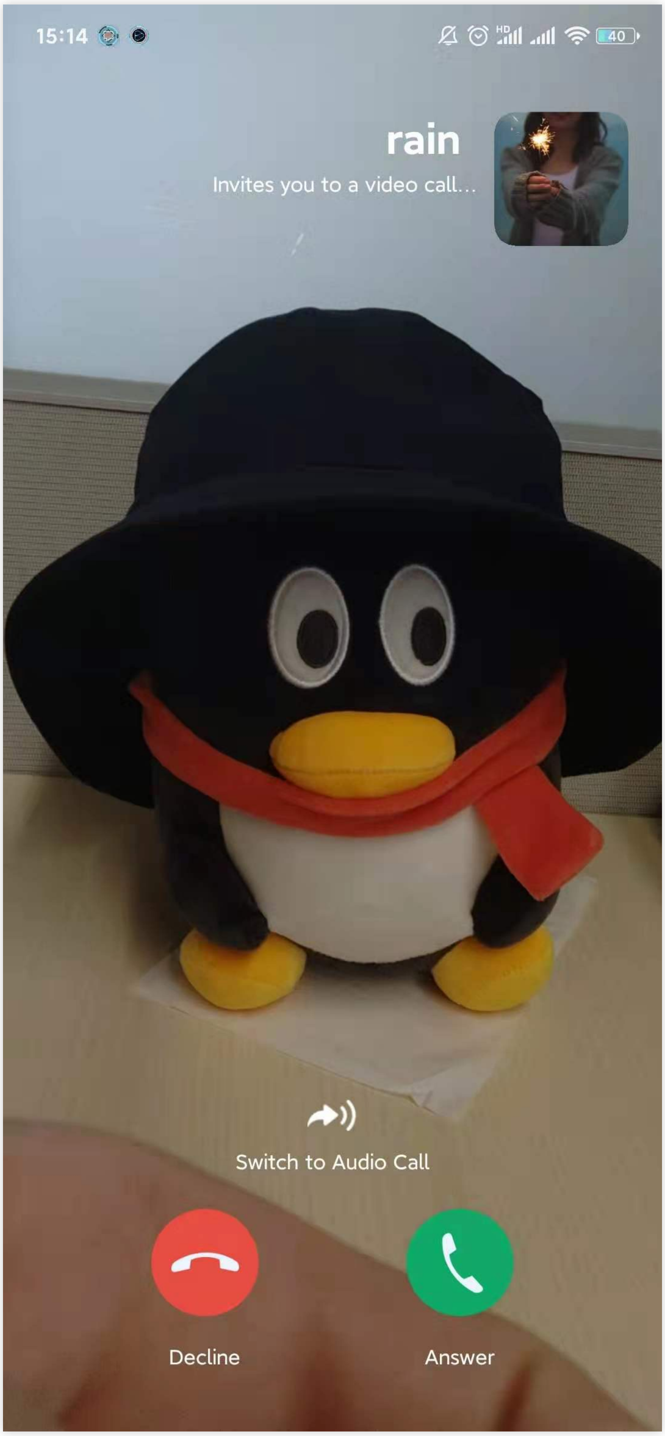
You can download and install the app we provide to try out the real-time audio/video call feature.

Call	Answer



rain

Waiting for the user to accept...



rain

Invites you to a video call...

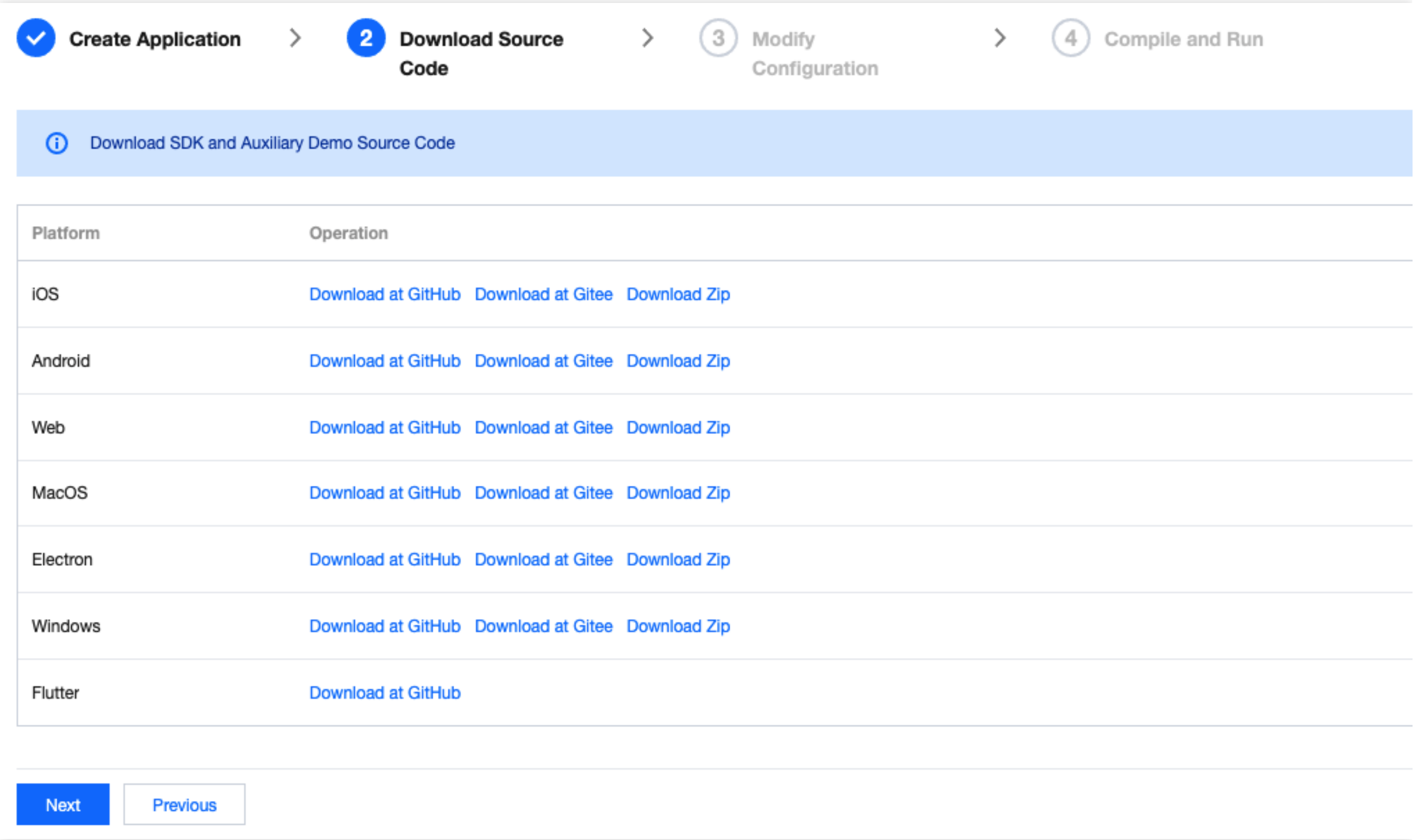
Note :

To make it easier for you to implement the real-time audio/video call feature, we have refactored the `TUICalling` component. You no longer need to pay attention to UI as it is now implemented within the `TUICalling` component.

Running the Demo

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestVideoCall` and click **Create**.
3. Click **Next**.



ⓘ Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Note :
This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the application source code

Clone or download the TUICalling source code.

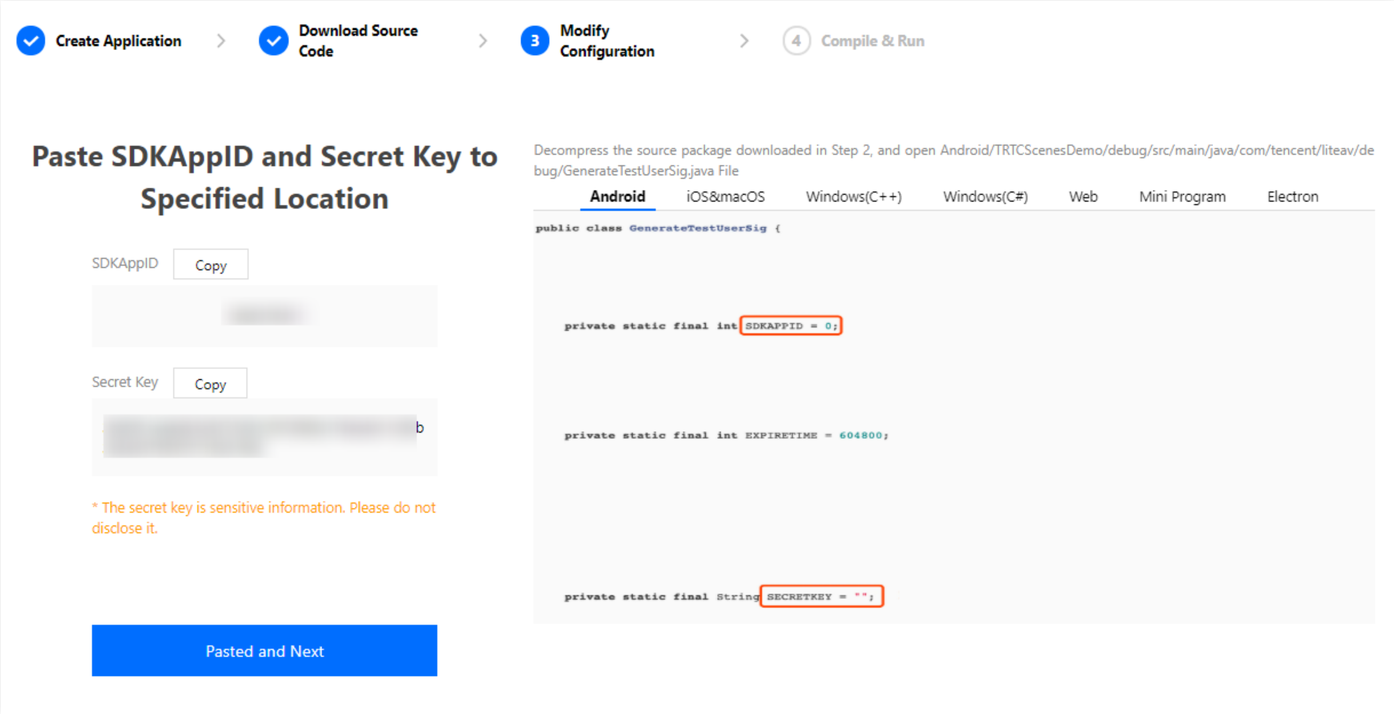
Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `Android/Debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java`.

3. Set parameters in `GenerateTestUserSig.java`:

   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID [redacted]

Secret Key [redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note:
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your

application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUICalling` with Android Studio (version 3.5 or above) and click **Run**.

Tryout

Note：
You need at least two devices to try out the application.

User A

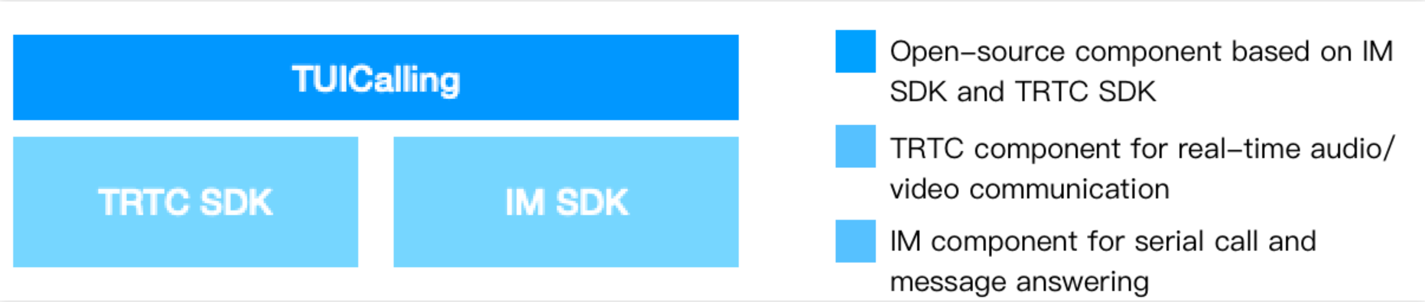
1. Enter a username (**which must be unique**) and log in.
2. Enter the `userId` of the person you want to call and tap **Search**.
3. Tap **Call** and select **Video Call** (**Make sure that the callee is active in the application, or the call may fail**).

User B

1. Enter a username (**which must be unique**) and log in.
2. Go to the homepage and wait for the incoming call.

Integration Directions

In the source code, the `Source` folder contains two subfolders: `ui` and `model`. The `model` subfolder includes the open-source `TUICallingManager` component, which we share with the public. You can find the component’s APIs in the `TUICalling.java` file.


TUICalling
TRTC SDK
IM SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for serial call and message answering

You can enable the real-time audio/video call feature for your project with ease using the open-source `TUICalling` and `TUICallingManager` components, with no need to implement complicated call UI or logic by yourself.

Step 1. Integrate the SDKs

The audio/video call component `TUICalling` depends on the TRTC SDK and IM SDK. Follow the steps below to integrate the two SDKs into your project:

Method 1: adding dependencies via mavenCentral

1. Add the TRTC SDK and IM SDK dependencies to `dependencies` .

```
dependencies {
compile ”com.tencent.liteav:LiteAVSDK_TRTC:latest.release”
compile ’com.tencent.imsdk:imsdk:latest.release’

// As we use Gson for parsing, you also need to add the Google Gson dependency.
compile ’com.google.code.gson:gson:latest.release’
}
```

2. In `defaultConfig` , specify the CPU architecture to be used by your application.

```
defaultConfig {
ndk {
abiFilters ”armeabi-v7a”
}
}
```

3. Click **Sync Now** to sync the SDKs.

> Note :
>
> If you have no problem connecting to mavenCentral, the SDKs will be downloaded and integrated into your project automatically.

Method 2: adding dependencies through local AAR files

If your access to the mavenCentral repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration document
IM SDK	Download	Integration documentation

Step 2. Configure permission requests and obfuscation rules

Configure permission requests for your application in `AndroidManifest.xml`. The SDKs need the following permissions (on Android 6.0 and above, the camera and read storage permissions must be requested at runtime.)

```
<uses-permission android:name=”android.permission.INTERNET” />
<uses-permission android:name=”android.permission.ACCESS_NETWORK_STATE” />
<uses-permission android:name=”android.permission.ACCESS_WIFI_STATE” />
<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE” />
<uses-permission android:name=”android.permission.READ_EXTERNAL_STORAGE” />
<uses-permission android:name=”android.permission.RECORD_AUDIO” />
<uses-permission android:name=”android.permission.MODIFY_AUDIO_SETTINGS” />
<uses-permission android:name=”android.permission.BLUETOOTH” />
<uses-permission android:name=”android.permission.CAMERA” />
<uses-permission android:name=”android.permission.READ_PHONE_STATE” />
<uses-feature android:name=”android.hardware.camera”/>
<uses-feature android:name=”android.hardware.camera.autofocus” />
```

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

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

Step 3. Import the `TUICalling` component

Copy the `Source` directory to your project and import it in `setting.gradle` as shown below:

```
include ’:Source’
```

Step 4. Initialize the component and log in

1. Call `TUICallingManager.sharedInstance()` to initialize the component.
2. Call `TUILogin.init(context, SDKAppID, config, listener)` to initialize login.
3. Call `TUILogin.login(userId, userSig, callback)` to log in to the component. Specify the key parameters as described below:

Parameter	Description

SDKAppID	You can view `SDKAppID` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. For the calculation method, please see UserSig.
config	SDK configuration, which is used to set the log output level and log callback. You can pass `null` for this parameter. For details, see the sample code below.
listener	IM listener, which is used to listen for some crucial callbacks, such as forced logout and `userSig` expiration. For details, see the sample code below.
callback	Callback for login, which indicates whether login is successful. For details, see the sample code below.

```
// Initialize the component
TUICallingManager manager = TUICallingManager.sharedInstance();
// Log in to the component
V2TIMSDKConfig config = new V2TIMSDKConfig();
config.setLogLevel(V2TIMSDKConfig.V2TIM_LOG_DEBUG);
config.setLogListener(new V2TIMLogListener() {
@Override
public void onLog(int logLevel, String logContent) {

}
});
TUILogin.init(this, ${Your `SDKAPPID`}, config, new V2TIMSDKListener() {
@Override
public void onKickedOffline() { // Callback for forced logout
mIsKickedOffline = true;
checkUserStatus();
}
@Override
public void onUserSigExpired() { // Callback for `userSig` expiration
mIsUserSigExpired = true;
checkUserStatus();
}
});
TUILogin.login(”${Your `userId`}”, ”${Your `userSig`}”, new V2TIMCallback() {
@Override
public void onError(int code, String msg) {
Log.d(TAG, ”code: ” + code + ” msg:” + msg);
}
@Override
public void onSuccess() {
Log.d(TAG, "onSuccess");
}
});
```

Step 5. Make an audio/video call

1. Caller: Call `call();` of `TUICallingManager` to initiate a call, passing in the user IDs (`userids`) and call type (`type`). For the call type, you can pass in `TUICalling.Type.AUDIO` (audio call) or `TUICalling.Type.VIDEO` (video call). If only one user ID is passed in for `userids` , the call is a one-to-one call; if two or more user IDs are passed in, the call is a group call.
2. Callee: If a callee is logged in, the answering view will be displayed. If you want offline users to receive call invitations, please see Enable offline call answering.

```
// 1. Initialize the component
TUICallingManager manager = TUICallingManager.sharedInstance();
// 2. Register the listener
manager.setCallingListener(new TUICalling.TUICallingListener() {
@Override
public boolean shouldShowOnCallView() {
return true;
}

@Override
public void onCallStart(String[] userIDs, TUICalling.Type type, TUICalling.Role role, final View
tuiCallingView) {
if (!shouldShowOnCallView() || null == tuiCallingView) {
return;
}
runOnUiThread(new Runnable() {
@Override
public void run() {
FrameLayout.LayoutParams params = new FrameLayout.LayoutParams(FrameLayout.LayoutParams.MATCH_PAR
ENT, FrameLayout.LayoutParams.MATCH_PARENT);
mCallingView = tuiCallingView;
addContentView(tuiCallingView, params);
}
});
}

@Override
public void onCallEnd(String[] userIDs, TUICalling.Type type, TUICalling.Role role, long totalTim
e) {
```

```
removeView();
}

@Override
public void onCallEvent(TUICalling.Event event, TUICalling.Type type, TUICalling.Role role, Strin
g message) {
if (TUICalling.Event.CALL_FAILED == event) {
removeView();
}
}
});
// 3. Make a call
manager.call(userIDs, TUICalling.Type.VIDEO);
```

Step 6. Enable offline call answering

> Note :
> If your project does not require the offline answering feature, for example, if it offers online customer service, your integration can end at step 5. If your project is a social networking service, we recommend you enable offline answering.

The IM SDK supports offline push. However, since the offline push service of Android phones varies from vendor to vendor, the configuration required to enable offline push for Android is more complicated than that for iOS.

1. Apply for a certificate required by the vendor's push channel, configure it in the IM console, and add the certificate and ID as required. For detailed directions, please see IM > Offline Push (Android).
2. The offline push API has been integrated into the signaling API (`sendModel`) of `TRTCCallingImpl` . After completing the offline push configuration for your application, you will be able to send notifications to offline users.

Component APIs

The table below lists the APIs of the `TUICalling` component.

API	Description
call	Sends call invitations by user ID.

API	Description
receiveAPNSCalled	Answers a call.
setCallingListener	Sets the listener.
setCallingBell	Sets the ringtone (preferably shorter than 30s).
enableMuteMode	Enables/Disables the mute mode.
enableCustomViewRoute	Enables/Disables custom views. After enabling custom views, you will receive a `CallingView` instance in the callback for calling/being called, and can decide how to display the view by yourself. The view must be displayed full screen or in proportion to the screen size; otherwise, an error may occur.

Video Call (Web)

Last updated : 2022-04-02 16:27:03

This document describes how to implement browser-based video calls.

- Part 1 describes how to activate the service and run the demo.
- Part 2 describes how to build your own video call feature using the `TRTCCalling` component.

Environment Requirements

Currently, the desktop version of Chrome offers better support for the features of the TRTC Web SDK; therefore, Chrome is recommended for the demo.

`TRTCCalling` uses the following ports and domain name for data transfer, which should be added to the allowlist of the firewall. After configuration, please use official demo to check whether the configuration has taken effect.

- TCP port: 8687
- UDP ports: 8000, 8080, 8800, 843, 443, 16285
- Domain name: qcloud.rtc.qq.com
  For details, please see Dealing with Firewall Restrictions.

Supported Platforms

The service supports the following platforms:

OS	Browser (Desktop)	Minimum Browser Version Requirement
macOS	Safari	11+
macOS	Chrome	56+
macOS	Firefox	56+
macOS	Edge	80+
Windows	Chrome	56+
Windows	QQ Browser (WebKit core)	10.4+
Windows	Firefox	56+

OS	Browser (Desktop)	Minimum Browser Version Requirement
Windows	Edge	80+

Note :

For more information on browser compatibility, please see Browsers Supported. You can also run an online test using the TRTC compatibility check page.

URL Protocol Support

Scenario	Protocol	Receive (Playback)	Send (Publish)	Share Screen	Remarks
Production	HTTPS	Supported	Supported	Supported	Recommended
Production	HTTP	Supported	Not supported	Not supported	-
Local development	http://localhost	Supported	Supported	Supported	Recommended
Local development	http://127.0.0.1	Supported	Supported	Supported	-
Local development	http://[local IP address]	Supported	Not supported	Not supported	-
Local development	file:///	Supported	Supported	Supported	-

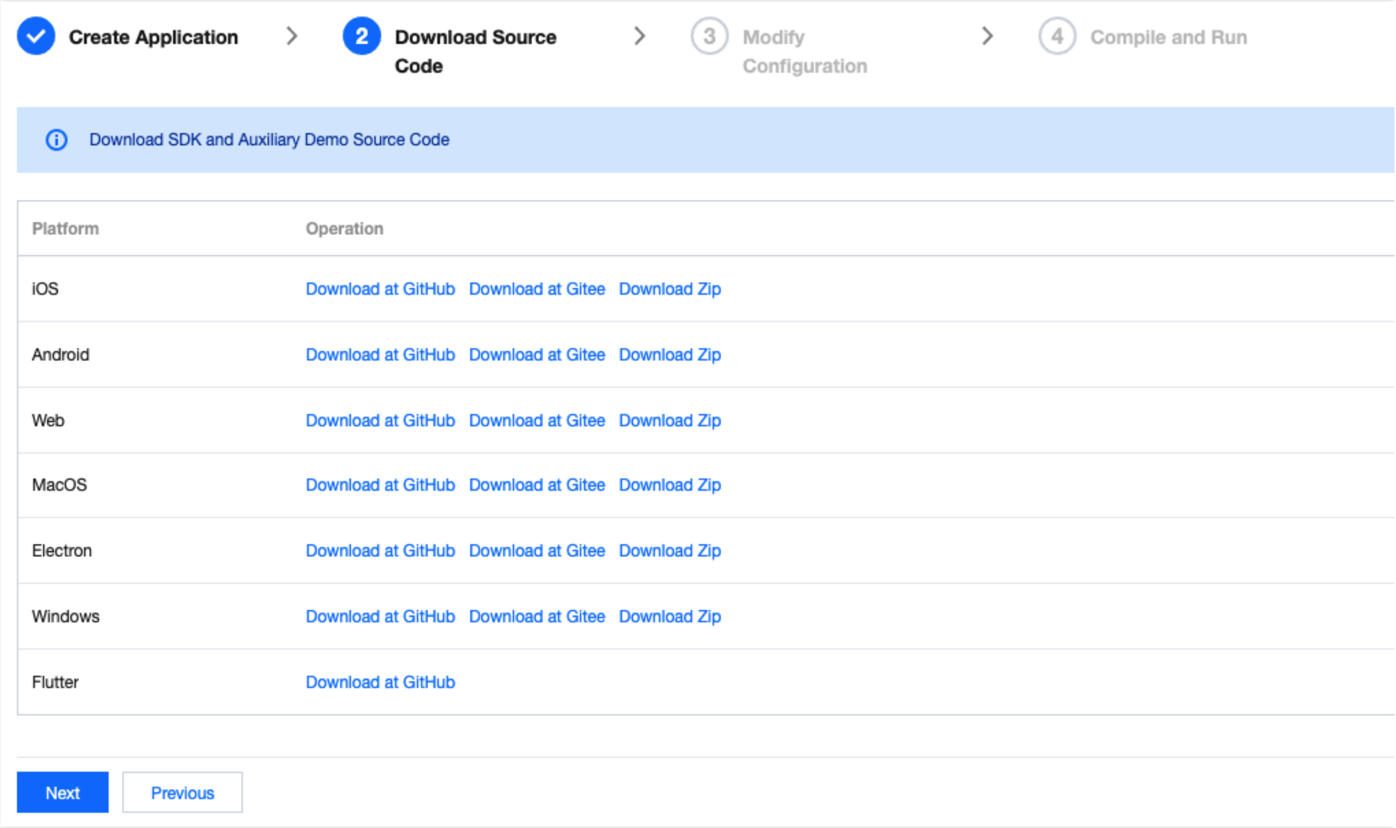
Running the Demo

Step 1. Create an application

1. Sign up for a Tencent Cloud account and verify your identity.
2. In the TRTC console, click **Development Assistance** > **Demo Quick Run**.
3. Enter an application name such as `TestTRTC` and click **Create**.

Step 2. Download the demo

1. You can download the source code of the web demo at TUICalling.
2. Click **Next**.

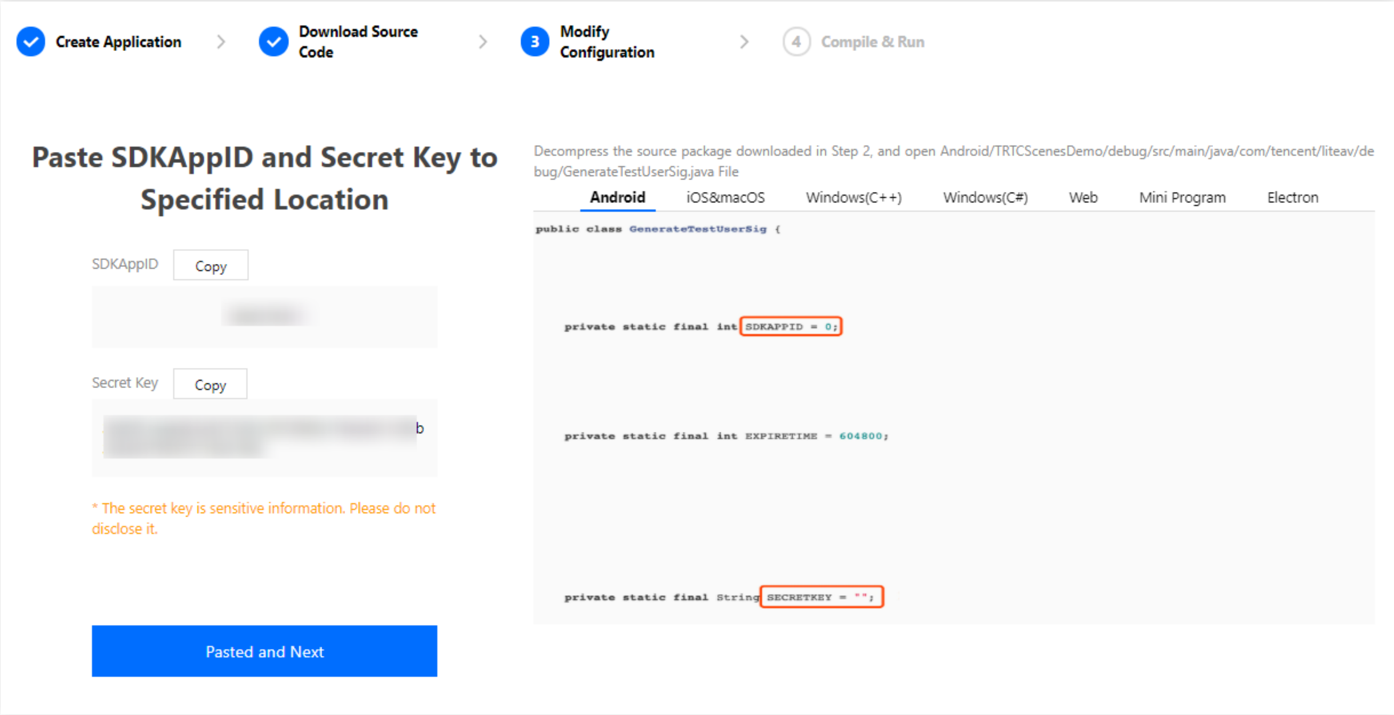


Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Step 3. Configure the demo project file

1. In the **Modify Configuration** step, select your platform.
2. Find and open the `Web/public/debug/GenerateTestUserSig.js` file.
3. Set the following parameters in the `GenerateTestUserSig.js` file:
   - SDKAPPID: `0` by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: Left empty by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.
5. After compilation, click **Return to Overview Page**.

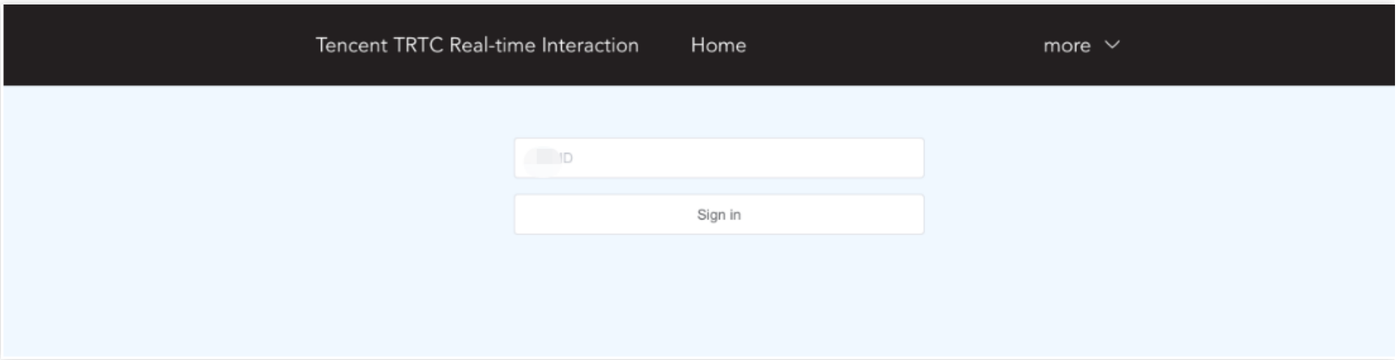
> Note :
>
> - In this document, the method to obtain UserSig is to configure a SECRETKEY in the client code. In this method, the SECRETKEY is vulnerable to decompilation and reverse engineering. Once your SECRETKEY is leaked, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and feature debugging**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 4. Run the demo

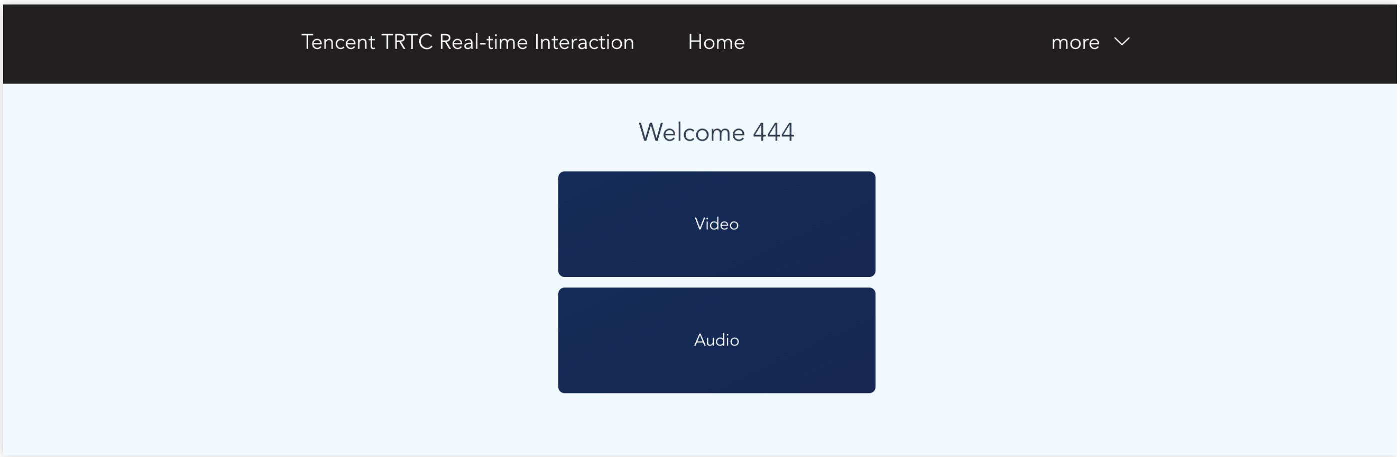
1. In the demo directory, run the following commands in turn:

```
npm install
npm run serve
```

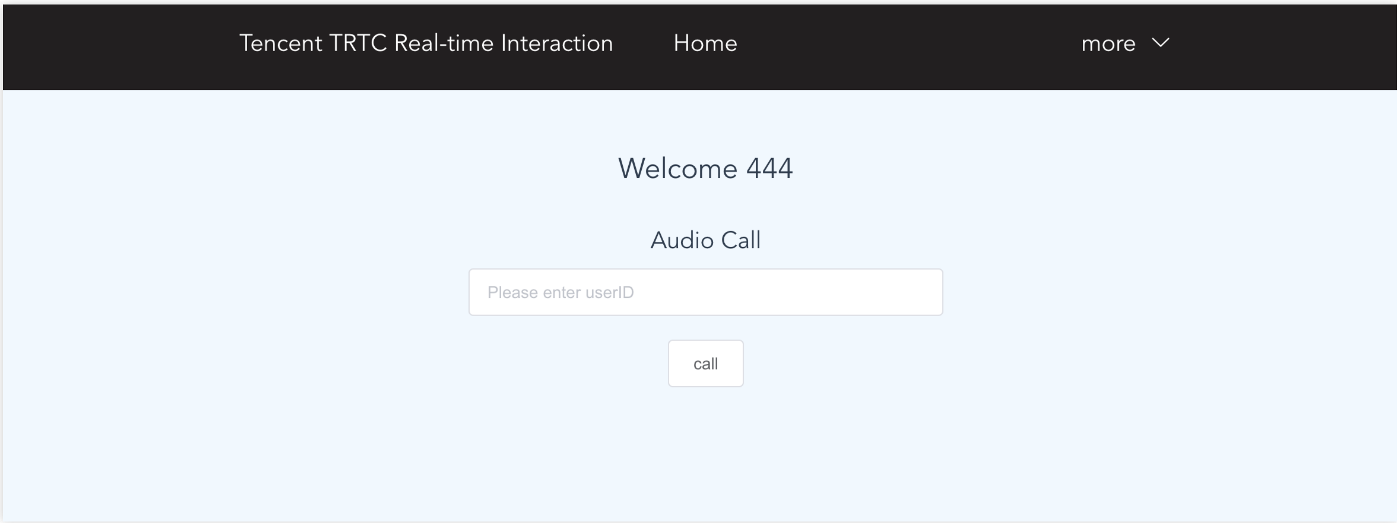
2. Open Chrome and visit `http://localhost:8080/` . If the above steps are performed correctly, you will see the page below:



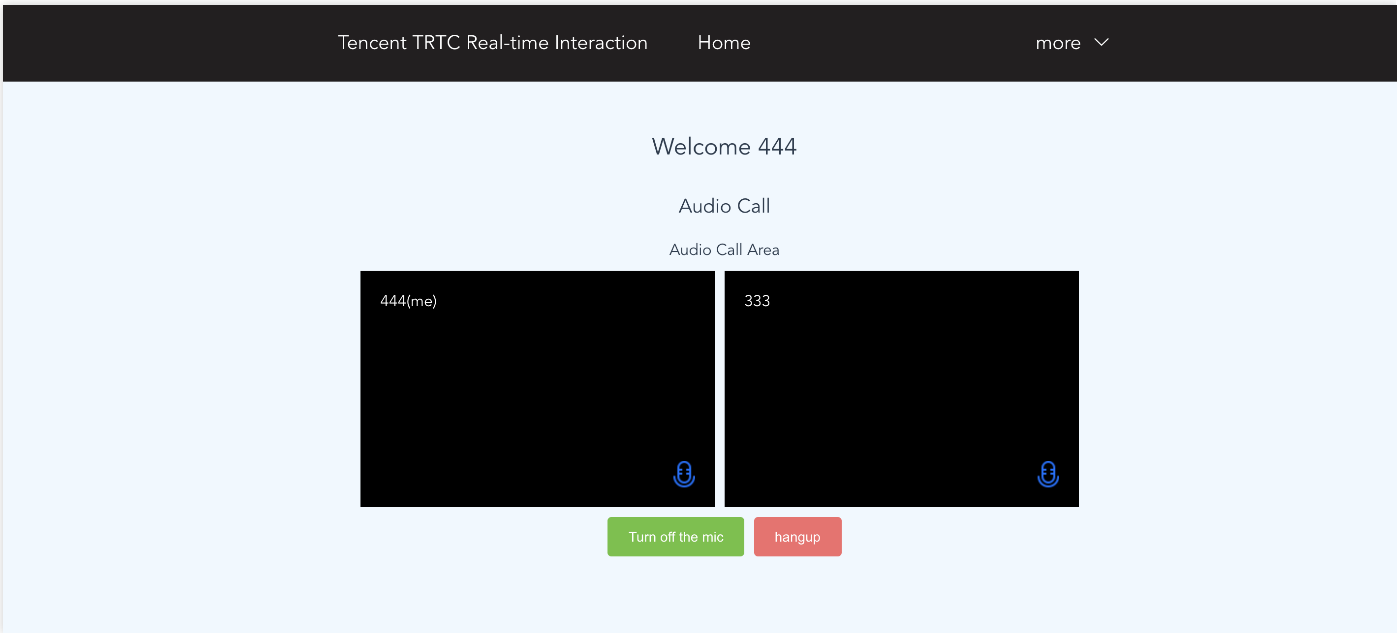
3. Enter a user ID, click **Sign in**, and select **Video**.



4. Enter the user ID of the callee and click **call**.



5. Start the video call.



Building Your Own Video Call Feature

Step 1. Import the `TRTCCalling` component

Note :

- Since version 0.6.0, you need to manually install dependencies trtc-js-sdk, tim-js-sdk, and tsignaling.
- To reduce the size of `trtc-calling-js.js` and prevent version conflict between `trtc-calling-js.js` and the already-in-use `trtc-js-sdk` , `tim-js-sdk` or `tsignaling` , the latter three are packaged as external dependencies, which you need to install manually before use.

```
// Import via npm
npm install trtc-js-sdk --save

npm install tim-js-sdk --save

npm install tsignaling --save

npm install trtc-calling-js --save
```

```
// If you import `trtc-calling-js` via a script, you need to manually import the following resour
ces in the specified order.

<script src=”./trtc.js”></script>
<script src=”./tim-js.js”></script>
<script src=”./tsignaling.js”></script>
<script src=”./trtc-calling-js.js”></script>
```

Step 2. Create a `TRTCCalling` object

Create a `TRTCCalling` object, setting `SDKAppID` to the `SDKAppID` of your application.

```
import TRTCCalling from ’trtc-calling-js’;

let options = {
SDKAppID: 0, // Replace 0 with your `SDKAppID` when connecting
// The `tim` parameter is added starting from v0.10.2
// The parameter guarantees the uniqueness of an existing TIM instance.
tim: tim
};
const trtcCalling = new TRTCCalling(options);
```

Step 3. Log in

Call `login` to log in, with `userID` set to your user ID and `userSig` set to your signature. For how to calculate and use UserSig, see FAQs > UserSig.

```
trtcCalling.login({
userID,
userSig,
});
```

Step 4. Make a one-to-one call

- **Caller: call a user**

```
trtcCalling.call({
userID, // User ID
type: 2, // Call type. `0`: unknown; `1`: audio call; `2`: video call
timeout // Timeout threshold, in seconds
});
```

- **Callee: process a call invitation**

```
// Answer
trtcCalling.accept();
// Reject
trtcCalling.reject()
```

- **Turn the local camera on**

```
trtcCalling.openCamera()
```

- **Play the video of a remote user**

```
trtcCalling.startRemoteView({
userID, // Remote user ID
videoViewDomID // The user’ s data will be rendered in this DOM node.
})
```

- **Show local video preview**

```
trtcCalling.startLocalView({
userID, // Local user ID
videoViewDomID // The user’ s data will be rendered in this DOM node.
})
```

- **Hang up**

```
trtcCalling.hangup()
```

FAQs

Why can’t I get through to a user? Why am I kicked offline?

The `TRTCCalling` component does not support login of multiple instances or **offline signaling** for the time being. Please make sure that your current login is unique.

Note :

- **Multiple instances**: A user logs in with the same account multiple times or on different devices, which disrupts signaling.
- **Offline signaling**: Only online instances can receive messages. Messages sent to offline instances will not be sent again when the instances go online.
  For FAQs, see TRTCCalling for Web.

Technical Support

If you have other questions, you can fill out a contact form or email colleenyu@tencent.com.

Learn More

- TRTCCalling web demo
- TRTCCalling for npm
- Source code of TRTCCalling web demo
- TRTCCalling web APIs
- FAQs

Video Call (Flutter)

Last updated : 2022-03-17 15:45:09

To quickly implement the video call feature, you can use the demo we provide and adapt it to your needs. You can also use the `TRTCCalling` component and customize your own UI.

Using the Demo UI

Step 1. Create an application

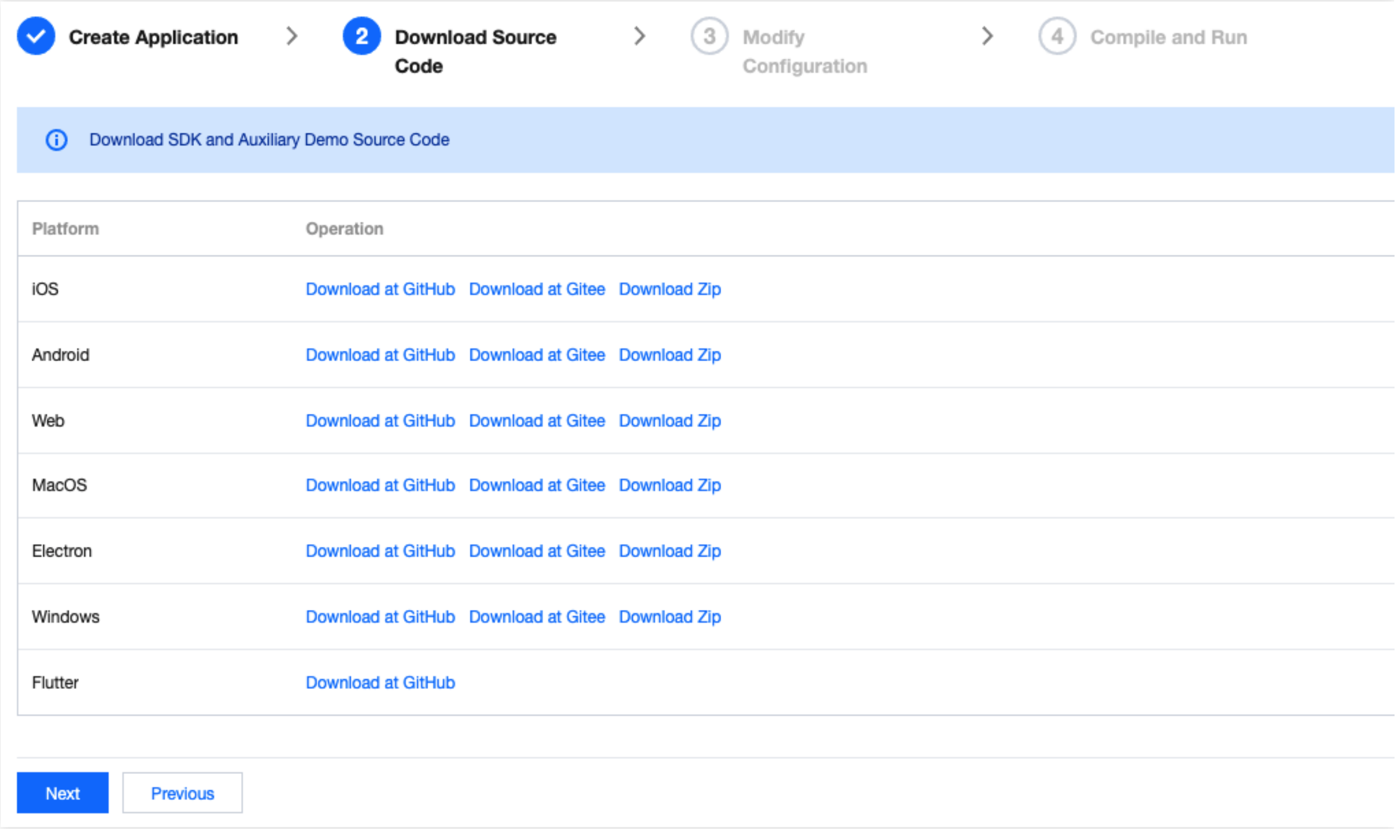
1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestVideoCall` and click **Create**.

> Note :
> This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the SDK and demo source code

1. Download the SDK and demo source code for your platform.

2. Click **Next**.



✓ Create Application > 2 Download Source Code > 3 Modify Configuration > 4 Compile and Run

ⓘ Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

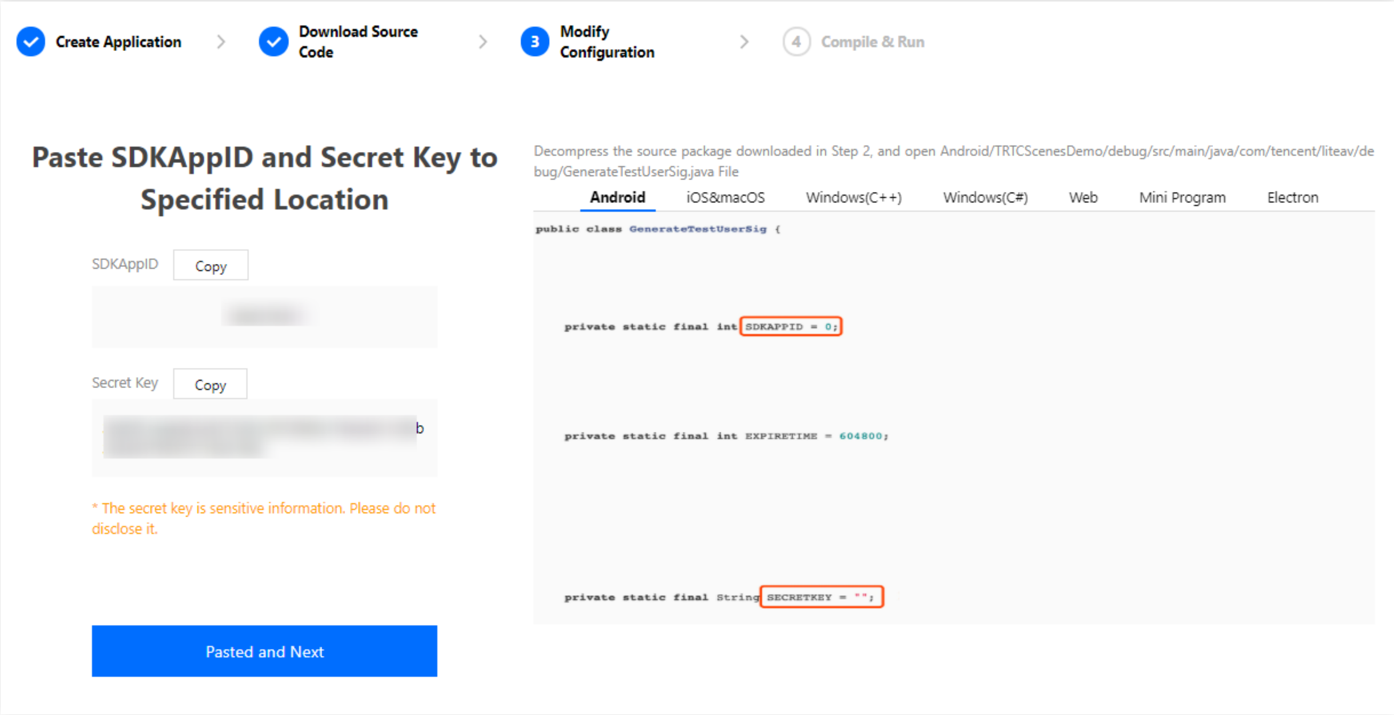
Step 3. Configure the demo project file

1. In the **Modify Configuration** step, select your platform.

2. Find and open `/example/lib/debug/GenerateTestUserSig.dart` .

3. Set parameters in `GenerateTestUserSig.dart` as follows.

   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.
   - `SECRETKEY`: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is leaked, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for the local execution and debugging of the demo**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Run the demo

> Note :

An Android project must be run on a real device rather than a simulator.

1. Run `flutter pub get` .
2. Compile, run, and test the project.
   - Android
   - iOS
   i. Run `flutter run` .
   ii. Open the demo project with Android Studio (3.5 or above), and click **Run**.

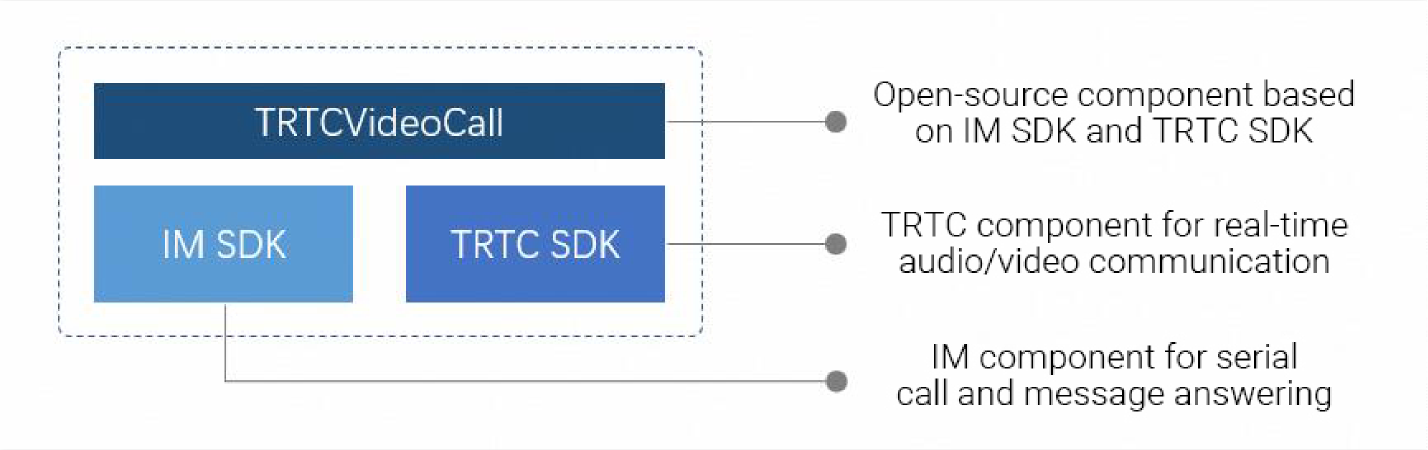
Step 5. Modify the demo source code

The `TRTCCallingDemo` folder in the source code contains two subfolders: `ui` and `model` . The `ui` folder contains the UI code.

File or Folder	Description
TRTCCallingVideo.dart	The main view for audio/video calls, where calls are answered/declined
TRTCCallingContact.dart	The view for contacts, where one can search for registered users to call

Customizing UI

The `TRTCCallingDemo` folder in the source code contains two subfolders: `ui` and `model` . The `model` subfolder contains the reusable open-source component `TRTCCalling` . You can find the component's APIs in `TRTCCalling.dart` .


TRTCVideoCall
IM SDK
TRTC SDK
Open-source component based on IM SDK and TRTC SDK
TRTC component for real-time audio/video communication
IM component for serial call and message answering

You can use the open-source component `TRTCCalling` to customize your own UI. This means you will use the model of the demo but design the UI by yourself.

Step 1. Integrate the SDKs

The audio/video call component `TRTCCalling` depends on the TRTC SDK and IM SDK. You can configure `pubspec.yaml` to download their updates automatically.

Add the following dependencies to `pubspec.yaml` of your project.

```
dependencies:
tencent_trtc_cloud: latest version number
tencent_im_sdk_plugin: latest version number
```

Step 2. Configure permission requests and obfuscation rules

- iOS
- Android

Add request for mic permission in `Info.plist` :

```
<key>NSMicrophoneUsageDescription</key>
<string>Audio calls are possible only with mic access.</string>
```

Step 3. Import the `TRTCCalling` component

Copy all the files in the directory below to your project:

```
/lib/TRTCCallingDemo/model
```

Step 4. Initialize the component and log in

1. Call `TRTCCalling.sharedInstance()` to get an instance of the component.
2. Call `login(SDKAppID, userId, userSig)` to log in to the component. For the key parameters passed in, see the table below.

Parameter	Description
SDKAppID	You can view \`SDKAppID\` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.

userSig	Tencent Cloud's proprietary security signature. For the calculation method, please see UserSig.

```
// Initialize
sCall = await TRTCCalling.sharedInstance();
sCall.login(1400000123, "userA", "xxxx");
```

Step 5. Make a one-to-one video call

1. The caller calls `call()` of `TRTCCalling`, passing in the user ID of the callee (`userid`) and call type (`type`). For a video call, the call type should be `TRTCCalling.typeVideoCall` .
2. The callee, if logged in, will receive the `onInvited()` callback and can start the corresponding view based on the call type set by the inviter, which is represented by `callType` in the callback.
3. The callee can call `accept()` to answer the call and `openCamera()` to turn the local camera on. He or she can also call `reject()` to reject the call.
4. After audio/video communication is established between the caller and callee, they will both receive the `onUserVideoAvailable()` event notification, which indicates that they have received each other's image. They can then call `startRemoteView()` to display each other’s image. Remote audio will be played back automatically by default.

Component APIs

The table below lists the APIs of the `TRTCCalling` component.

API	Description
registerListener	Registers a `TRTCCalling` listener, through which users can receive status notifications.
unRegisterListener	Unregisters a listener.
destroy	Destroys an instance.
login	Logs in to IM. All features can be used only after login.
logout	Logs out of IM. Calls cannot be made after logout.
call	Makes a C2C call. The invitee will receive the `onInvited` event notification.
accept	Answers a call.

API	Description
reject	Declines a call.
hangup	Ends a call.
startRemoteView	Renders the camera data of a remote user in the specified `TXCloudVideoView` .
stopRemoteView	Stops rendering the camera data of a remote user.
openCamera	Turns the camera on and renders the camera data in the specified `TXCloudVideoView` .
closeCamera	Turns the camera off.
switchCamera	Switches between the front and rear cameras.
setMicMute	Mutes/Unmutes the mic.
setHandsFree	Enables/Disables the hands-free mode.

TUICalling APIs (iOS)

Last updated : 2022-03-17 15:46:42

`TUICalling` is an open-source class based on two closed-source SDKs: Tencent Cloud Real-Time Communication (TRTC) and Instant Messaging (IM). It supports one-to-one and group video calls. For detailed instructions how to implement it, see Real-Time Video Call (iOS).

- TRTC SDK: The TRTC SDK is used as a low-latency audio/video call component.
- IM SDK: The IM SDK is used to send and process signaling messages.

`TUICalling` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton.
call	Sends call invitations by user ID.
receiveAPNSCalled	Answers a call.
setCallingListener	Sets the listener.
setCallingBell	Sets the ringtone (preferably shorter than 30s).
enableMuteMode	Enables/Disables the mute mode.
enableCustomViewRoute	Enables/Disables custom views.

`TUICallingListener` API Overview

Event callbacks

API	Description
shouldShowOnCallView	Callback of whether the answering view is displayed when there is an incoming call
onCallStart	Callback for starting calling. This callback is triggered for both callers and callees.

API	Description
onCallEnd	Callback for ending a call. This callback is triggered for both callers and callees.
onCallEvent	Call event callback

Type

Call type

Enumerated Type	Description
TUICallingTypeAudio	Audio call
TUICallingTypeVideo	Video call

Role

Role type

Enumerated Type	Description
TUICallingRoleCall	Caller
TTUICallingRoleCalled	Callee

Event

Event type

Enumerated Type	Description
TUICallingEventCallStart	The call started.
TUICallingEventCallSucceed	The call was connected successfully.
TUICallingEventCallEnd	The call ended.
TUICallingEventCallFailed	The call failed.

Basic SDK APIs

sharedInstance

This API is used to get a singleton of the `TUICallingManager` component.

```
+ (instancetype)shareInstance;
```

call

This API is used to send call invitations by user ID.

```
- (void)call:(NSArray<NSString *> *)userIDs type:(TUICallingType)type;
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	NSArray	List of the user IDs of call participants
type	TUICallingType	Call type: audio/video

receiveAPNSCalled

This API is used to answer a call.

```
- (void)receiveAPNSCalled:(NSArray<NSString *> *)userIDs type:(TUICallingType)type;
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	NSArray	List of the user IDs of call participants
type	TUICallingType	Call type: audio/video

setCallingListener

This API is used to set the listener.

```
- (void)setCallingListener:(id<TUICallingListerner>)listener;
```

The parameters are as detailed below:

Parameter	Type	Description
listener	TUICallingListener	Listener of the `TUIcalling` component

setCallingBell

This API is used to set the ringtone (preferably shorter than 30s).

```
- (void)setCallingBell:(NSString *)filePath;
```

The parameters are as detailed below:

Parameter	Type	Description
filePath	NSString	Path of the ringtone file

enableMuteMode

This API is used to enable/disable the mute mode.

```
- (void)enableMuteMode:(BOOL)enable;
```

The parameters are as detailed below:

Parameter	Type	Description
enable	BOOL	Whether to enable the mute mode

enableCustomViewRoute

This API is used to enable/disable custom views.
After enabling custom views, you will receive a `CallingViewController` instance in the callback for calling/being called, and can decide how to display the view by yourself.

> Note :
> The view must be displayed full screen; otherwise, an error may occur.

```
- (void)enableCustomViewRoute:(BOOL)enable;
```

The parameters are as detailed below:

Parameter	Type	Description
enable	BOOL	Whether to enable custom views

`TUICallingListener` Callback APIs

shouldShowOnCallView

Callback of whether the answering view is displayed when there is an incoming call.

```
- (BOOL)shouldShowOnCallView;
```

The parameters are as detailed below:

Parameter	Type	Description
Return value	BOOL	Whether the answering view is displayed

onCallStart

Callback for starting calling, which is triggered for both callers and callees.

```
- (void)callStart:(NSArray<NSString *> *)userIDs type:(TUICallingType)type role:(TUICallingRole)r
ole viewController:(UIViewController * _Nullable)viewController;
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	NSArray	List of the user IDs of call participants
type	TUICallingType	Call type: audio/video
role	TUICallingRole	Role type: caller/callee
viewController	UIViewController	Calling view controller

onCallEnd

Callback for ending a call, which is triggered for both callers and callees. If `enableCustomViewRoute` is set to `NO` , this callback will not be triggered.

```
- (void)callEnd:(NSArray<NSString *> *)userIDs type:(TUICallingType)type role:(TUICallingRole)rol
e totalTime:(CGFloat)totalTime;
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	NSArray	List of the user IDs of call participants
type	TUICallingType	Call type: audio/video
role	TUICallingRole	Role type: caller/callee
totalTime	CGFloat	Call duration (s)

onCallEvent

Call event callback, which is triggered for both callers and callees. If `enableCustomViewRoute` is set to `NO` , this callback will not be triggered.

```
- (void)onCallEvent:(TUICallingEvent)event type:(TUICallingType)type role:(TUICallingRole)role me
ssage:(NSString *)message;
```

The parameters are as detailed below:

Parameter	Type	Description
event	TUICallingEvent	Call event type
type	TUICallingType	Call type: audio/video
role	TUICallingRole	Role type: caller/callee
message	NSString	Event description

TUICalling APIs (Android)

Last updated : 2022-03-17 15:47:18

`TUICalling` is an open-source class based on two closed-source SDKs: Tencent Cloud Real-Time Communication (TRTC) and Instant Messaging (IM). It supports one-to-one and group video calls. For detailed instructions how to implement it, see Real-Time Video Call (Android).

- TRTC SDK: The TRTC SDK is used as a low-latency audio/video call component.
- IM SDK: The IM SDK is used to send and process signaling messages.

`TUICalling` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton.
call	Sends call invitations by user ID.
receiveAPNSCalled	Answers a call.
setCallingListener	Sets the listener.
setCallingBell	Sets the ringtone (preferably shorter than 30s).
enableMuteMode	Enables/Disables the mute mode.
enableCustomViewRoute	Enables/Disables custom views.

`TUICallingListener` API Overview

Event callbacks

API	Description
shouldShowOnCallView	Callback of whether the answering view is displayed when there is an incoming call
onCallStart	Callback for starting calling. This callback is triggered for both callers and callees.

API	Description
onCallEnd	Callback for ending a call. This callback is triggered for both callers and callees.
onCallEvent	Call event callback

Type

Call type

Enumerated Type	Description
AUDIO	Audio call
VIDEO	Video call

Role

Role type

Enumerated Type	Description
CALL	Caller
CALLED	Callee

Event

Event type

Enumerated Type	Description
CALL_START	The call started.
CALL_SUCCEED	The call was connected successfully.
CALL_END	The call ended.
CALL_FAILED	The call failed.

Basic SDK APIs

sharedInstance

This API is used to get a singleton of the `TUICalling` component.

```
public static TUICallingManager sharedInstance();
```

call

This API is used to send call invitations by user ID.

```
void call(String[] userIDs, Type type);
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	String[]	List of the user IDs of call participants
type	TUICalling.Type	Call type: audio/video

receiveAPNSCalled

This API is used to answer a call.

```
void receiveAPNSCalled(String[] userIDs, Type type);
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	String[]	List of the user IDs of call participants
type	TUICalling.Type	Call type: audio/video

setCallingListener

This API is used to set the listener.

```
void setCallingListener(TUICallingListener listener);
```

The parameters are as detailed below:

Parameter	Type	Description
listener	TUICallingListener	Listener of the `TUIcalling` component

setCallingBell

This API is used to set the ringtone (preferably shorter than 30s).

```
void setCallingBell(String filePath);
```

The parameters are as detailed below:

Parameter	Type	Description
filePath	String	Path of the ringtone file

enableMuteMode

This API is used to enable/disable the mute mode.

```
void enableMuteMode(boolean enable);
```

The parameters are as detailed below:

Parameter	Type	Description
enable	boolean	Whether to enable the mute mode

enableCustomViewRoute

This API is used to enable/disable custom views.
After enabling custom views, you will receive a `CallingView` instance in the callback for calling/being called, and can decide how to display the view by yourself.

> Note :
> The view must be displayed full screen or in proportion to the screen size; otherwise, an error may occur.

```
void enableCustomViewRoute(boolean enable);
```

The parameters are as detailed below:

Parameter	Type	Description
enable	boolean	Whether to enable custom views

`TUICallingListener` Callback APIs

shouldShowOnCallView

Callback of whether the answering view is displayed when there is an incoming call.

```
boolean shouldShowOnCallView();
```

The parameters are as detailed below:

Parameter	Type	Description
Return value	boolean	Whether the answering view is displayed

onCallStart

Callback for starting calling, which is triggered for both callers and callees.

```
void onCallStart(String[] userIDs, TUICalling.Type type, TUICalling.Role role, View tuiCallingVie
w);
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	String[]	List of the user IDs of call participants
type	TUICalling.Type	Call type: audio/video
role	TUICalling.Role	Role type: caller/callee
tuiCallingView	View	Calling view. When `enableCustomViewRoute` is `false` , `view` is null.

onCallEnd

Callback for ending a call, which is triggered for both callers and callees.

```
void onCallEnd(String[] userIDs, TUICalling.Type type, TUICalling.Role role, long totalTime);
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	String[]	List of the user IDs of call participants
type	TUICalling.Type	Call type: audio/video
role	TUICalling.Role	Role type: caller/callee
totalTime	long	Call duration (s)

onCallEvent

Call event callback.

```
void onCallEvent(TUICalling.Event event, TUICalling.Type type, TUICalling.Role role, String message);
```

The parameters are as detailed below:

Parameter	Type	Description
event	TUICalling.Event	Call event type
type	TUICalling.Type	Call type: audio/video
role	TUICalling.Role	Role type: caller/callee
message	String	Event description

TRTCCalling APIs (Web)

Last updated : 2022-04-06 14:08:33

`TRTCCalling` Overview

The TRTCCalling component is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). It supports one-to-one and group audio/video calls. For detailed instructions on how to implement it, see Real-Time Video Call (Web).

- TRTC SDK: The TRTC SDK is used as a low-latency audio/video call component.
- IM SDK: The IM SDK is used to send and process signaling messages.

Demo Download

You can download the source code of the web demo at TUICalling.

Environment Requirements

We recommend you use Chrome for PC to run the demo as it offers better support for the features of the TRTC Web SDK. For more information on environment requirements, see Environment Requirements.

URL Protocol Support

Scenario	Protocol	Receive (Playback)	Send (Publish)	Share Screen	Remarks
Production	HTTPS	Supported	Supported	Supported	Recommended
Production	HTTP	Supported	Not supported	Not supported	-
Local development	http://localhost	Supported	Supported	Supported	Recommended
Local development	http://127.0.0.1	Supported	Supported	Supported	-

Scenario	Protocol	Receive (Playback)	Send (Publish)	Share Screen	Remarks
Local development	http://[local IP address]	Supported	Not supported	Not supported	-
Local development	file:///	Supported	Supported	Supported	-

`TRTCCalling` APIs

Event subscribing/unsubscribing APIs

This component bases its management on the dispatching of events. The application layer can change UI interactions according to dispatched events.

API	Description
on(eventName, callback, context)	Subscribes to an event.
off(eventName, callback, context)	Unsubscribes from an event.

Basic SDK APIs

API	Description
login({userID, userSig})	Logs in to IM. All IM features can be used only after login.
logout()	Logs out. No calls can be made after logout.

Call operation APIs

API	Description
call({userID, type, offlinePushInfo}))	Invites a user to a one-to-one call.
groupCall({userIDList, type, groupID, offlinePushInfo})	Invites users to a group call.
accept()	Accepts a call.
reject()	Rejects a call.
hangup()	Hangs up.

Video APIs

API	Description
startRemoteView({userID, videoViewDomID})	Starts rendering the video of a remote user.
stopRemoteView({userID})	Stops rendering the video of a remote user.
startLocalView({userID, videoViewDomID})	Starts rendering the video of the local user.
stopLocalView({userID})	Stops rendering the video of the local user.
openCamera()	Turns the camera on.
closeCamera()	Turns the camera off.
setMicMute(isMute)	Mutes/Unmutes the mic.
setVideoQuality(profile)	Sets video quality.
switchToAudioCall()	Switches to audio call.
switchToVideoCall()	Switches to video call.
getCameras()	Gets the camera list.
getMicrophones()	Gets the mic list.
switchDevice({deviceType, deviceId})	Switches to a different camera or mic.

API Details

Creating a `TRTCCalling` instance

First, create an application in the TRTC console and get the `SDKAppID` .
Then, obtain an instance of the `TRTCCalling` component using `new TRTCCalling()` .

```
npm i trtc-js-sdk --save
npm i tim-js-sdk --save
npm i tsignaling --save
npm i trtc-calling-js --save
// If you download the dependency using Node.js, you can import it using an import command.
import TRTCCalling from 'trtc-calling-js';
// If you use JavaScript, you need to manually import the following resources in the specified or
der.
// trtc.js
```

```
<script src=”./trtc.js”></script>
// tim-js.js
<script src=”./tim-js.js”></script>
// tsignaling.js
<script src=”./tsignaling.js”></script>
// trtc-calling-js.js
<script src=”./trtc-calling-js.js”></script>
let options = {
SDKAppID: 0, // Replace `0` with the `SDKAppID` of your IM application when connecting
// The `tim` parameter is added starting from v0.10.2
// The parameter guarantees the uniqueness of an existing TIM instance.
tim: tim
};
let trtcCalling = new TRTCCalling(options);
```

Event subscribing/unsubscribing APIs

on(eventName, callback, context)

This API is used to subscribe to an event dispatched by the component. For a list of the events, see `TRTCCalling` Events.

```
let handleInvite = function ({inviteID, sponsor, inviteData}) {
console.log(`inviteID: ${inviteID}, sponsor: ${sponsor}, inviteData: ${JSON.stringify(inviteDat
a)}`);
};
trtcCalling.on('onInvited', handleInvite, this);
```

off(eventName, callback, context)

This API is used to unsubscribe from an event.

```
let handleInvite = function ({inviteID, sponsor, inviteData}) {
console.log(`inviteID: ${inviteID}, sponsor: ${sponsor}, inviteData: ${JSON.stringify(inviteDat
a)}`);
};
trtcCalling.off('onInvited', handleInvite, this);
```

Basic SDK APIs

login({userID, userSig})

This API is used to log in.

```
trtcCalling.login({userID, userSig})
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_)
userSig	String	Tencent Cloud's proprietary security signature. For how to calculate and use it, see FAQs > UserSig.

logout()

This API is used to log out.

```
trtcCalling.logout()
```

Call operation APIs

call({userID, type, offlinePushInfo})

This API is used to make a one-to-one call. `type` indicates the call type (`1` : audio call; `2` : video call).

> Note :
>
> - The `timeout` parameter is deleted from v1.0.0 and later versions.
> - A new parameter, `offlinePushInfo` , is introduced for offline notifications, **which are supported only on Android and iOS, not on web or WeChat Mini Program.**

```
// Versions earlier than v1.0.0
trtcCalling.call({userID, type, timeout})

// v1.0.0 and later versions
const offlinePushInfo = {
title: '',
description: 'You are invited to a call.',
}
trtcCalling.call({userID, type, offlinePushInfo})
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userID	String	`userID` of the invitee
type	Number	Call type. `1` : audio call; `2` : video call
timeout	Number	Timeout period (s). `0` means the call never times out. **This parameter is valid only for versions earlier than v1.0.0.**
offlinePushInfo	Object	Offline notifications (optional). **This parameter is valid only for v1.0.0 and later versions.**

offlinePushInfo (in v1.0.0 and later versions)

Parameter	Type	Description
title	String	Title of an offline notification (optional)
description	String	Content of an offline notification (optional)
androidOPPOChannelID	String	Channel ID for an offline notification on OPPO 8.0 and above (optional)
extension	String	Passthrough content of an offline notification (optional), **which is valid only for TRTCCalling v1.0.2 or above and TSignaling v0.9.0 or above**

groupCall({userIDList, type, groupID, offlinePushInfo})

The `groupID` parameter is the group ID in the IM SDK. If this parameter is set, call invitations will be broadcast by the group messaging system, which is a simple and reliable way of sending call invitations. If this parameter is left empty, the `TRTCCalling` component will send an invitation to every invitee.

> Note :
> In v1.0.0 and later versions, a new parameter, `offlinePushInfo` , is introduced for offline notifications, **which are supported only on Android and iOS, not on web or WeChat Mini Program.**

```
// Versions earlier than v1.0.0
trtcCalling.groupCall({userIDList, type, groupID})
```

```
// v1.0.0 and later versions
const offlinePushInfo = {
title: '',
description: 'You are invited to a call.',
}
trtcCalling.groupCall({userIDList, type, groupID, offlinePushInfo})
```

The parameters are as detailed below:

Parameter	Type	Description
userIDList	Array	List of the user IDs of invitees
type	Number	Call type. `1` : audio call; `2` : video call
groupID	String	IM group ID (optional)
offlinePushInfo	Object	Offline notifications (optional). **This parameter is valid only for v1.0.0 and later versions.**

offlinePushInfo (in v1.0.0 and later versions)

Parameter	Type	Description
title	String	Title of an offline notification (optional)
description	String	Content of an offline notification (optional)
androidOPPOChannelID	String	Channel ID for an offline notification on OPPO 8.0 and above (optional)
extension	String	Passthrough content of an offline notification (optional), **which is valid only for TRTCCalling v1.0.2 or above and TSignaling v0.9.0 or above**

accept()

This API is used to accept an invitation.

Note :

- If a prior invitation has not been processed, the component will return a message indicating that the line is busy.
- `params` is deleted from v1.0.0 and later versions.

```
import TRTCCalling from 'trtc-calling-js';
trtcCalling.on(TRTCCalling.EVENT.INVITED, ({inviteID, sponsor, inviteData}) => {
// ...
// Versions earlier than v1.0.0
const { roomID, callType } = inviteData;
trtcCalling.accept({inviteID, roomID, callType})
// v1.0.0 and later versions
trtcCalling.accept();
})
```

The parameters are as detailed below:

Parameter	Type	Description
inviteID	String	Invitation ID, which identifies an invitation and is returned by the `INVITED` callback. **This parameter is valid only for versions earlier than v1.0.0**.
roomID	Number	Call room ID, which is returned by the `INVITED` callback (in `inviteData`). **This parameter is valid only for versions earlier than v1.0.0**.
callType	Number	Call type, which is turned by the `INVITED` callback (in `inviteData`). `1` : audio call; `2` : video call. **This parameter is valid only for versions earlier than v1.0.0**.

reject()

This API is used to reject an invitation.

> Note :
> `params` is deleted from v1.0.0 and later versions.

```
import TRTCCalling from 'trtc-calling-js';
trtcCalling.on(TRTCCalling.EVENT.INVITED, ({inviteID, sponsor, inviteData}) => {
// ...
// Versions earlier than v1.0.0
const { callType } = inviteData;
trtcCalling.reject({inviteID, isBusy, callType})
// v1.0.0 and later versions
trtcCalling.reject();
})
```

The parameters are as detailed below:

Parameter	Type	Description
inviteID	String	Invitation ID, which identifies an invitation and is returned by the `INVITED` callback. **This parameter is valid only for versions earlier than v1.0.0**.
isBusy	Boolean	Whether the line is busy. **This parameter is valid only for versions earlier than v1.0.0.**
callType	Number	Call type, which is turned by the `INVITED` callback (in `inviteData`). `1` : audio call; `2` : video call. **This parameter is valid only for versions earlier than v1.0.0**.

hangup()

1. If you are in a call, you can use this API to end the call.
2. If your call is not answered yet, you can use this API to cancel the call.

```
trtcCalling.hangup()
```

Video APIs

startRemoteView({userID, videoViewDomID})

This API is used to render the camera data of a remote user in a specified DOM node.

```
trtcCalling.startRemoteView({userID, videoViewDomID})
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
videoViewDomID	String	The DOM node in which the user’s data is to be rendered. The data will be played via the video tag of the node.

stopRemoteView({userID})

This API is used to delete the DOM node in which the camera data of a remote user is rendered.

Note :
`videoViewDomID` is deleted from v1.0.0 and later versions.

```
// Versions earlier than v1.0.0
trtcCalling.stopRemoteView({userID, videoViewDomID});
// v1.0.0 and later versions
trtcCalling.stopRemoteView({userID});
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
videoViewDomID	String	The DOM node whose video tag is to be deleted. The playback will stop. **This parameter is valid only for versions earlier than v1.0.0.**

startLocalView({userID, videoViewDomID})

This API is used to render the camera data of the local user in a specified DOM node.

```
trtcCalling.startLocalView({userID, videoViewDomID})
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
videoViewDomID	String	The DOM node in which the local user’s data is to be rendered. The data will be played via the video tag of the node.

stopLocalView({userID})

This API is used to delete the DOM node in which the camera data of the local user is rendered.

Note :
`videoViewDomID` is deleted from v1.0.0 and later versions.

```
// Versions earlier than v1.0.0
trtcCalling.stopLocalView({userID, videoViewDomID});
// v1.0.0 and later versions
trtcCalling.stopLocalView({userID});
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
videoViewDomID	String	The DOM node whose video tag is to be deleted. The playback will stop. **This parameter is valid only for versions earlier than v1.0.0.**

openCamera()

This API is used to turn the local camera on.

```
trtcCalling.openCamera()
```

closeCamera()

This API is used to turn the camera off.

```
trtcCalling.closeCamera()
```

setMicMute(isMute)

This API is used to turn the mic on/off.

```
trtcCalling.setMicMute(true) // Turn the mic off
```

The parameters are as detailed below:

Parameter	Type	Description
isMute	Boolean	• `true` : turn the mic off; • `false` : turn the mic on

setVideoQuality(profile)

This API is used to set video quality.

Note :

- This is a new API in v0.8.0 and later versions.
- This API must be called before `call` , `groupCall` , or `accept` to take effect.

```
trtcCalling.setVideoQuality('720p') // Set video quality to `720p`
```

The parameters are as detailed below:

Parameter	Type	Description
profile	String	• `480p` : 640 × 480 • `720p` : 1280 × 720 • `1080p` : 1920 × 1080

switchToAudioCall()

This API is used to switch from video call to audio call.

Note :

- This is a new API in v0.10.0 and later versions.
- It can be used only in one-to-one calls.
- If you receive an error callback (code: 60001), the switching failed.

```
trtcCalling.switchToAudioCall() // Switch from video call to audio call
```

switchToVideoCall()

This API is used to switch from audio call to video call.

Note :

- This is a new API in v0.10.0 and later versions.
- It can be used only in one-to-one calls.
- If you receive an error callback (code: 60002), the switching failed.

```
trtcCalling.switchToVideoCall() // Switch from audio call to video call
```

getCameras()

This API is used to get the camera list.

> Note :
> This is a new API in v1.0.0 and later versions.

```
trtcCalling.getCameras() // Get the camera list
```

getMicrophones()

This API is used to get the mic list.

> Note :
> This is a new API in v1.0.0 and later versions.

```
trtcCalling.getMicrophones() // Get the mic list
```

switchDevice({deviceType, deviceId})

This API is used to switch to a different camera or mic.

> Note :
> This is a new API in v1.0.0 and later versions.

```
trtcCalling.switchDevice({deviceType: 'video', deviceId: deviceId}) // Switch to a different device
```

The parameters are as detailed below:

Parameter	Type	Description
deviceType	String	Device type. `video` : camera; `audio` : mic

Parameter	Type	Description
deviceId	String	Device ID. - You can use `getCameras()` to get the ID of a camera. - You can use `getMicrophones()` to get the ID of a mic.

`TRTCCalling` Events

The code below demonstrates how to listen for TRTCCalling events.

```
import TRTCCalling from 'trtc-calling-js';
// etc
function handleInviteeReject({userID}) {

}
trtcCalling.on(TRTCCalling.EVENT.REJECT, handleInviteeReject)
```

Invitation events

Code	Event Recipient	Description
REJECT	Inviter	The invitee rejected the call.
NO_RESP	Inviter	The invitation timed out without response from the invitee.
LINE_BUSY	Inviter	The invitee is in a call, i.e., the line is busy.
INVITED	Invitee	You received an invitation.
CALLING_CANCEL	Invitee	The call is canceled.
CALLING_TIMEOUT	Invitee	The invitation timed out.
USER_ENTER	Inviter and invitee	A user entered the room.
USER_LEAVE	Inviter and invitee	A user left the call.
CALL_END	Inviter and invitee	The call ended.

Code	Event Recipient	Description
KICKED_OUT	Inviter and invitee	A user was kicked out due to repeated login.
USER_VIDEO_AVAILABLE	Inviter and invitee	A remote user turned the camera on/off.
USER_AUDIO_AVAILABLE	Inviter and invitee	A remote user turned the mic on/off.

Common event callbacks

SDK_READY

The SDK is ready.

> Note :
> This is a new event callback in v1.0.0 and later versions.

```
let onSDKReady = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.SDK_READY, onSDKReady);
```

USER_ENTER

A user entered the room.

This event callback is triggered when a user joins the call.

```
let handleUserEnter = function({userID}) {
console.log(userID)
};
trtcCalling.on(TRTCCalling.EVENT.USER_ENTER, handleUserEnter);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID

USER_LEAVE

A user left the room.

This event callback is triggered when a user leaves the call.

```
let handleUserLeave = function({userID}) {
console.log(userID)
};
trtcCalling.on(TRTCCalling.EVENT.USER_LEAVE, handleUserLeave);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID

GROUP_CALL_INVITEE_LIST_UPDATE

The invitee list for a group call was updated.

> Note :
>
> This is a new event callback in v1.0.0 and later versions.

```
let handleGroupInviteeListUpdate = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.GROUP_CALL_INVITEE_LIST_UPDATE, handleGroupInviteeListUpdate);
```

CALL_END

The call ended.

This event callback is triggered when a call ends.

```
let handleCallingEnd = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.CALL_END, handleCallingEnd);
```

KICKED_OUT

A user was kicked out due to repeated login.

This event callback is triggered if a user logs in with the same account on another page.

```
let handleKickedOut = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.KICKED_OUT, handleKickedOut);
```

USER_VIDEO_AVAILABLE

A remote user turned the camera on/off.

This event callback is triggered when a remote user turns the camera on/off.

```
let handleUserVideoChange = function({userID, isVideoAvailable}) {
console.log(userID, isVideoAvailable)
};
trtcCalling.on(TRTCCalling.EVENT.USER_VIDEO_AVAILABLE, handleUserVideoChange);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
isVideoAvailable	Boolean	• `true` : The remote user turned the camera on. • `false` : The remote user turned the camera off.

USER_AUDIO_AVAILABLE

A remote user turned the mic on/off.

This event callback is triggered when a remote user turns the mic on/off.

```
let handleUserAudioChange = function({userID, isAudioAvailable}) {
console.log(userID, isAudioAvailable)
};
trtcCalling.on(TRTCCalling.EVENT.USER_AUDIO_AVAILABLE, handleUserAudioChange);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
isAudioAvailable	Boolean	• `true` : The remote user turned the mic on. • `false` : The remote user turned the mic off.

Event callbacks received by inviter

REJECT

The user rejected the call.

The inviter of a call will receive this callback if an invitee rejects the call.

```
let handleInviteeReject = function({userID}) {
console.log(userID)
};
trtcCalling.on(TRTCCalling.EVENT.REJECT, handleInviteeReject);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID

NO_RESP

The invitee did not answer.

In cases where a timeout period is specified for a one-to-one or group call, the inviter will receive this callback if an invitee does not answer the call within the specified timeout period.

```
let handleNoResponse = function({userID, userIDList}) {
console.log(userID, userIDList)
};
trtcCalling.on(TRTCCalling.EVENT.NO_RESP, handleNoResponse);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
userIDList	Array	List of timed out users

LINE_BUSY

The invitee is in a call, i.e., the line is busy.

The inviter of a call will receive this callback if the invitee is already in a call.

```
let handleLineBusy = function({userID}) {
console.log(userID)
};
trtcCalling.on(TRTCCalling.EVENT.LINE_BUSY, handleLineBusy);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID

Event callbacks received by invitee

INVITED

An invitation was received.

A user will receive this callback if he or she is invited to a call.

```
let handleNewInvitationReceived = function({
sponsor, userIDList, isFromGroup, inviteData, inviteID
}) {
console.log(sponsor, userIDList, isFromGroup, inviteData, inviteID)
};
trtcCalling.on(TRTCCalling.EVENT.INVITED, handleNewInvitationReceived);
```

The parameters are as detailed below:

Parameter	Type	Description
sponsor	String	Inviter
userIDList	Array	Users invited to the same call
isFromGroup	Boolean	Whether it is an IM group invitation
inviteData	Object	For a new user invitation: {version, callType, roomID}
inviteID	String	Invitation ID, which identifies an invitation

CALLING_CANCEL

The call is canceled.

An invitee of a call will receive this callback if the call is canceled.

```
let handleCallingCancel = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.CALLING_CANCEL, handleCallingCancel);
```

CALLING_TIMEOUT

The call timed out.
In cases where a timeout period is specified for a one-to-one or group call, an invitee will receive this callback if he or she does not answer the call within the specified timeout period.

```
let handleCallingTimeout = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.CALLING_TIMEOUT, handleCallingTimeout);
```

`TRTCCalling` Error Codes

You can register the error callback, as shown below, to handle the errors thrown by the component.

```
import TRTCCalling from 'trtc-calling-js';
let onError = function(error) {
console.log(error);
};
trtcCalling.on(TRTCCalling.EVENT.ERROR, onError);
```

Error codes

Code	Type	Description
60001	API call failure	Failed to call `switchToAudioCall` .
60002	API call failure	Failed to call `switchToVideoCall` .
60003	Access failure	No available mic.
60004	Access failure	No available camera.
60005	Access failure	The user denied access.
60006	Failure to pass environment check	The current environment does not support WebRTC (v1.0.4 or above is required).

Update Guide

- **Updating to TRTCCalling 1.0.2 or above**
  - You need to update TSignaling to v0.9.0 or above.

  - Reason: See Changelog.
- **Updating to TRTCCalling 1.0.0 or 1.0.1**
  - You need to update TSignaling to v0.8.0 or above.
  - Reason: See Changelog.

FAQs

Why can’t I get through to a user? Why am I kicked offline?

The `TRTCCalling` component does not support login of multiple instances or **offline signaling** for the time being. Please make sure that your current login is unique.

> Note :
>
> - **Multiple instances**: A user logs in with the same account multiple times or on different devices, which disrupts signaling.
> - **Offline signaling**: Only online instances can receive messages. Messages sent to offline instances will not be sent again when the instances go online.
>   For FAQs about TRTCCalling for web, see TRTCCalling for Web.

Technical Support

If you have other questions, you can fill out a contact form or email colleenyu@tencent.com.

Learn More

- TRTCCalling web demo
- TRTCCalling for npm
- Source code of TRTCCalling web demo
- TRTCCalling web APIs
- FAQs

TRTCCalling APIs (Flutter)

Last updated：2022-04-08 14:39:00

`TRTCCalling` is an open-source class based on two closed-source SDKs: Tencent Cloud Real-Time Communication (TRTC) and Instant Messaging (IM). It supports one-to-one audio/video calls. For detailed instructions how to implement it, please see Real-Time Video Call (Flutter).

- TRTC SDK: the TRTC SDK is used as a low-latency audio/video call component.
- IM SDK: the IM SDK is used to send and process signaling messages.

TRTCCalling API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton.
destroySharedInstance	Destroys a singleton.
registerListener	Registers an event listener.
unRegisterListener	Unregisters an event listener.
destroy	Destroys an instance that is no longer needed.
login	Logs in. All features can be used only after login.
logout	Logs out. No calls can be made after logout.

Call operation APIs

API	Description
call	Makes a one-to-one call.
accept	Accepts the current call.
reject	Declines the current call.
hangup	Ends the current call.

Stream pushing/pulling APIs

API	Description
startRemoteView	Renders the camera data of a remote user in the specified `TXCloudVideoView` .
stopRemoteView	Stops rendering the data of a remote user.

Audio/Video APIs

API	Description
openCamera	Turns on the camera and renders the camera data in the specified `TXCloudVideoView` .
switchCamera	Switches between the front and rear cameras.
closeCamera	Turns off the camera.
setMicMute	Mutes the local mic.
setHandsFree	Enables the hands-free mode.

TRTCCallingDelegate API Overview

Common event callback APIs

API	Description
onError	Callback for error.

Inviter callback APIs

API	Description
onReject	The call was declined.
onNoResp	The invitee did not answer.
onLineBusy	The line is busy.

Invitee callback APIs

API	Description

API	Description
onInvited	An invitation was received.
onCallingCancel	The call was canceled.
onCallingTimeOut	The call timed out.

General callback APIs

API	Description
onUserEnter	A user joined the call.
onUserLeave	A user left the call.
onUserAudioAvailable	Whether a user is sending audio
onUserVideoAvailable	Whether a user is sending video
onUserVoiceVolume	Call volume of the user
onCallEnd	The call ended.

Basic SDK APIs

sharedInstance

This API is used to get a singleton of the `TRTCCalling` component.

```
static Future<TRTCCalling> sharedInstance();
```

destroySharedInstance

This API is used to destroy a singleton of the `TRTCCalling` component.

```
static void destroySharedInstance();
```

destroy

This API is used to destroy an instance that is no longer needed.

```
void destroy();
```

registerListener

This API is used to register callbacks for TRTCCalling events. You can receive status notifications of TRTCCalling via `TRTCCallingDelegate` .

```
void registerListener(VoiceListenerFunc func);
```

unRegisterListener

This API is used to unregister event callbacks.

```
void unRegisterListener(VoiceListenerFunc func);
```

login

This API is used to log in to the component.

```
Future<ActionCallback> login(int sdkAppId, String userId, String userSig);
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppID	int	You can view the `SDKAppID` via **Application Management** > **Application Info** in the TRTC console.
userId	String	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 more information on how to get it, please see UserSig.

logout

This API is used to log out of the component.

```
Future<ActionCallback> logout();
```

Call Operation APIs

call

This API is used to make a one-to-one call. It can be called during a call to invite more users.

```
Future<ActionCallback> call(String userId, int type);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the callee
type	int	`1` : audio call; `2` : video call

accept

This API is used to accept the current call. After receiving the `onInvited()` callback, the invitee can call this API to accept the call.

```
Future<ActionCallback> accept();
```

reject

This API is used to decline the current call. After receiving the `onInvited()` callback, the invitee can call this API to decline the call.

```
Future<ActionCallback> reject();
```

hangup

This API is used to end the current call.

```
void hangup();
```

Stream Pushing/Pulling APIs

startRemoteView

This API is used to render the camera data of a remote user in the specified `TRTCCloudVideoView` .

```
void startRemoteView(String userId, int streamType, int viewId);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	String	ID of the remote user
streamType	int	Video stream type of the `userId` specified for stopping watching
viewId	int	`viewId` called back by `TRTCCloudVideoView` , the control that carries the video image

stopRemoteView

This API is used to stop rendering the data of a remote user.

```
void stopRemoteView(String userId, int streamType);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the remote user
streamType	int	Type of video stream of the specified `userId` to stop rendering

Audio/Video APIs

openCamera

This API is used to turn on the camera and render data in the specified `TRTCCloudVideoView` .

```
void openCamera(bool isFrontCamera, int viewId);
```

The parameters are as detailed below:

Parameter	Type	Description
isFrontCamera	bool	`true` : turns on the front camera; `false` : turns on the rear camera.
viewId	int	`viewId` returned by `TRTCCloudVideoView` , the class that loads video images

switchCamera

This API is used to switch between the front and rear cameras.

```
void switchCamera(bool isFrontCamera);
```

The parameters are as detailed below:

Parameter	Type	Description
isFrontCamera	bool	`true` : switches to the front camera; `false` : switches to the rear camera.

closeCamara

This API is used to turn the camera off.

```
void closeCamera();
```

setMicMute

This API is used to mute the local mic.

```
void setMicMute(bool isMute);
```

The parameters are as detailed below:

Parameter	Type	Description
isMute	bool	`true` : mutes the mic; `false` : unmutes the mic.

setHandsFree

This API is used to enable the hands-free mode.

```
void setHandsFree(bool isHandsFree);
```

The parameters are as detailed below:

Parameter	Type	Description
isHandsFree	bool	`true` : enables the hands-free mode; `false` : disables the hands-free mode.

`TRTCCallingDelegate` Callback APIs

Common Event Callback APIs

onError

Callback for error.

> 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.

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
msg	String	Error message

Inviter Callback APIs

onReject

The call was declined.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the invitee who declined the call

onNoResp

The invitee did not answer.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the invitee who did not answer

onLineBusy

The line is busy.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the invitee whose line is busy

Invitee Callback APIs

onInvited

A call invitation was received.

The parameters are as detailed below:

Parameter	Type	Description
sponsor	String	User ID of the inviter
userIds	List<String>	IDs of other users invited
isFromGroup	bool	Whether it is a group call
type	int	`1` : audio call; `2` : video call

onCallingCancel

The call was canceled. The invitee will receive this callback if the inviter cancels the invitation before he or she handles it.

onCallingTimeOut

The call timed out.

General Callback APIs

onUserEnter

A user joined the call.

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	String	ID of the user who joined the call

onUserLeave

A user left the call.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user who left the call

onUserAudioAvailable

Whether a user is sending audio.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
available	boolean	Whether the user has available audio

onUserVideoAvailable

Whether a user is sending video. After receiving this callback, you can call `startRemoteView` to render the remote user's video.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
available	boolean	Whether the user has available video

onUserVoiceVolume

Call volume of a user.

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userVolumes	List	Volume of every speaking user in the room. Value range: 0-100.
totalVolume	int	Total volume of all remote users. Value range: 0-100.

onCallEnd

The call ended.

Audio Call

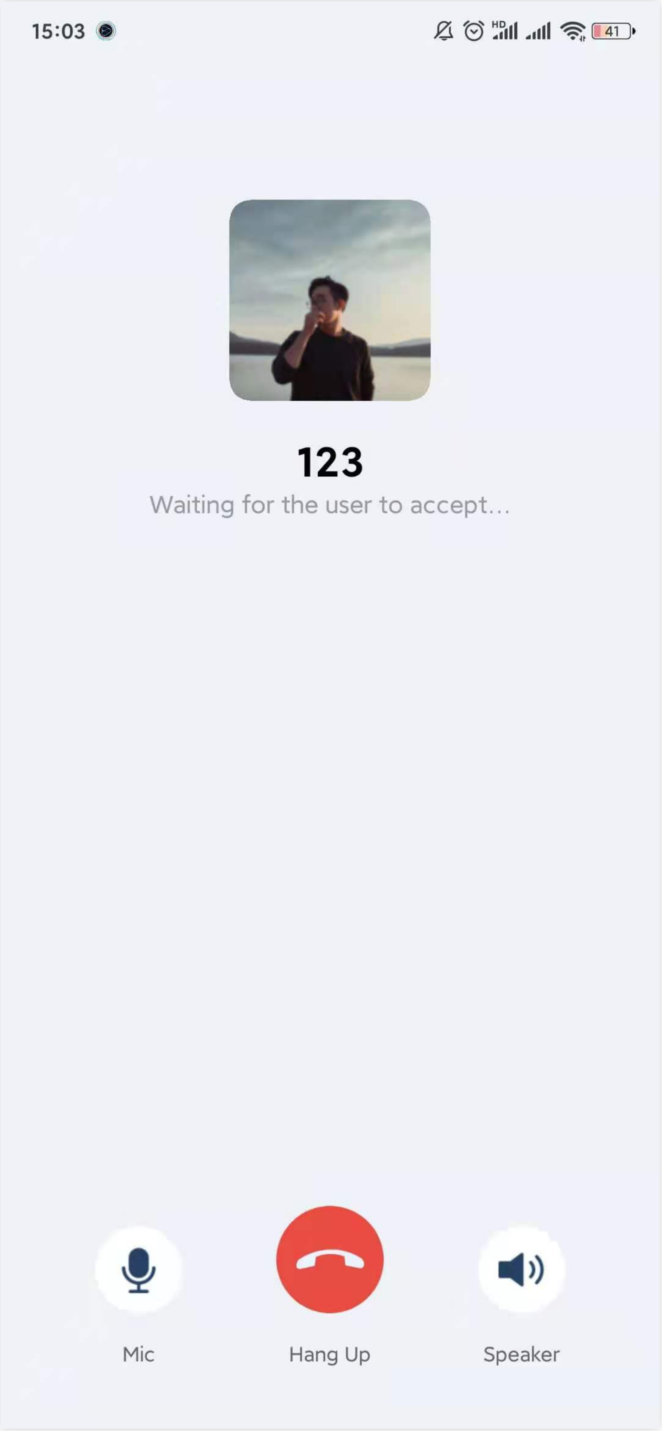
Audio Call (iOS)

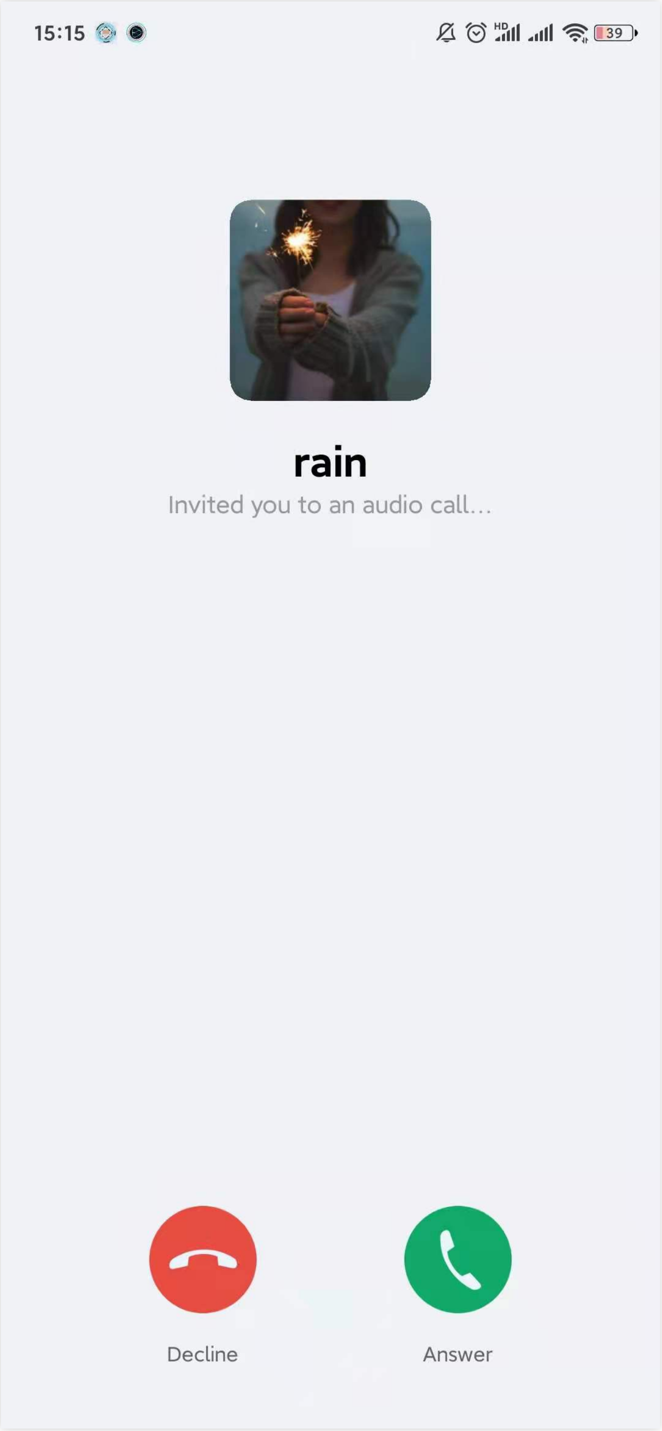
Last updated : 2022-03-17 15:40:35

Demo UI

You can download and install the app we provide to try out the real-time audio/video call feature.

Call	Answer





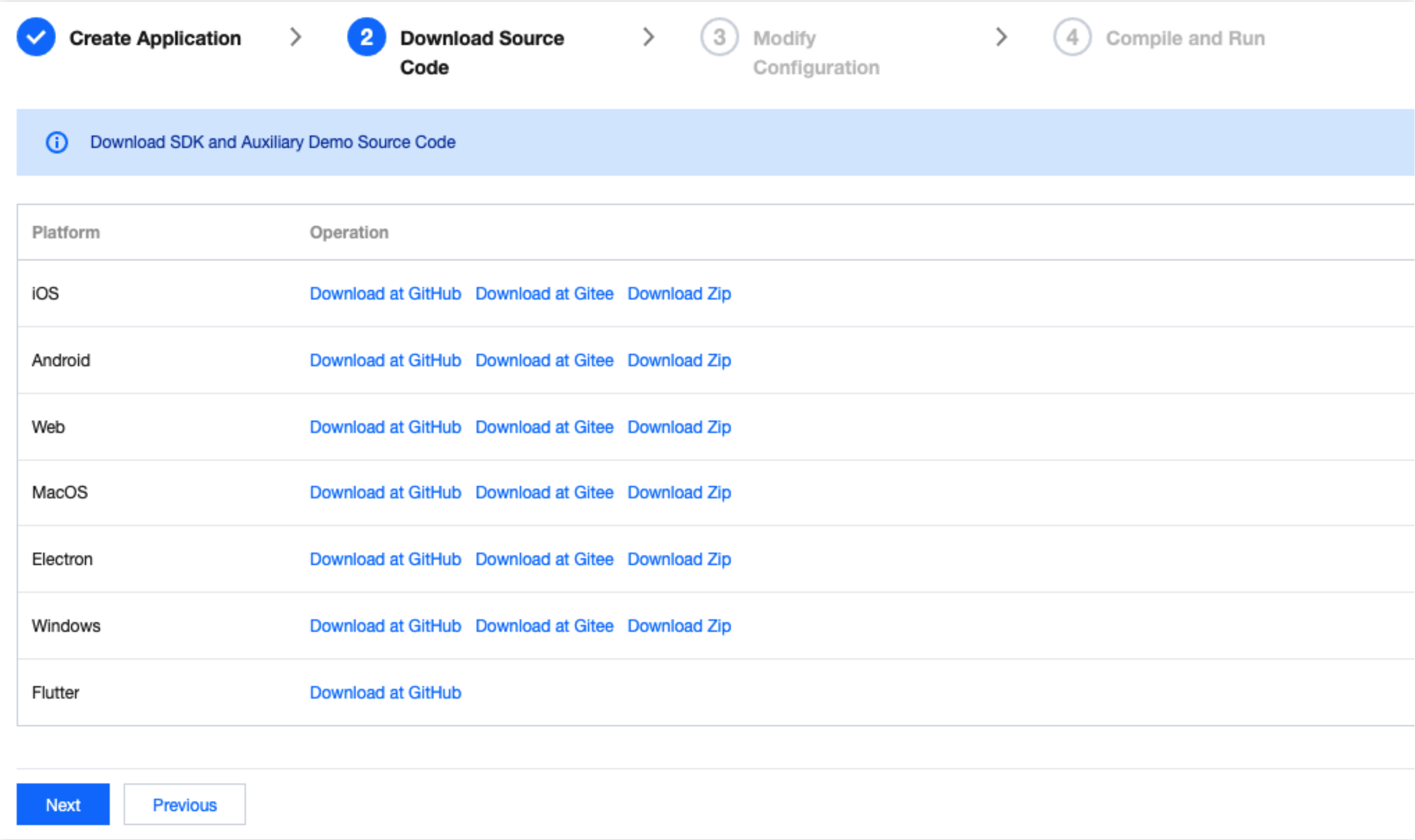
Note：

To make it easier for you to implement the real-time audio/video call feature, we have refactored the `TUICalling` component. You no longer need to pay attention to UI as it is now implemented within the `TUICalling` component.

Running the Demo

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestVideoCall` and click **Create**.
3. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

> Note :
> This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the application source code

Clone or download the TUICalling source code.

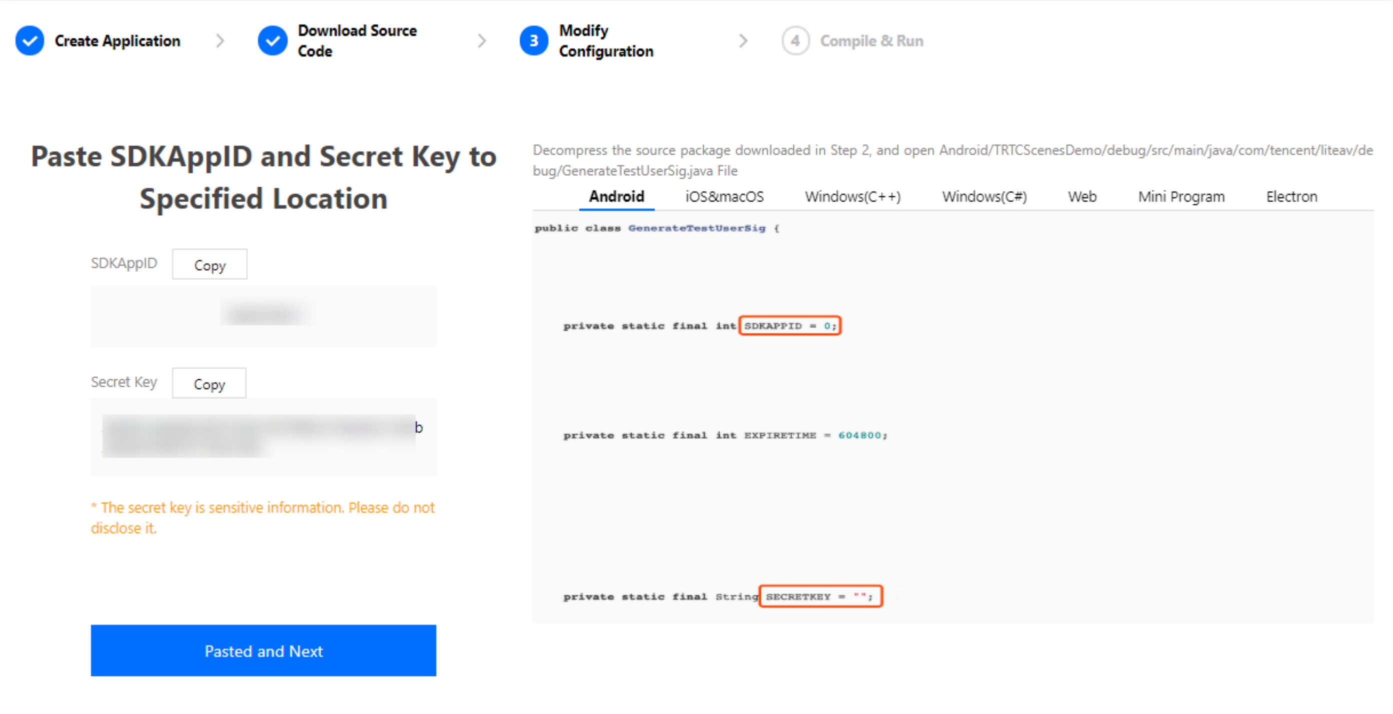
Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `iOS/Example/Debug/GenerateTestUserSig.swift` .

3. Set the following parameters in `GenerateTestUserSig.swift` :

   - SDKAPPID: \`0\` by default. Set it to the actual \`SDKAppID\`.
   - SECRETKEY: left empty by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Pasted and Next

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

Android | iOS&macOS | Windows(C++) | Windows(C#) | Web | Mini Program | Electron

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

Note :

- The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
- The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your

application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUICalling/Example/TUICallingApp.xcworkspace` with Xcode (version 11.0 or above) and click **Run**.

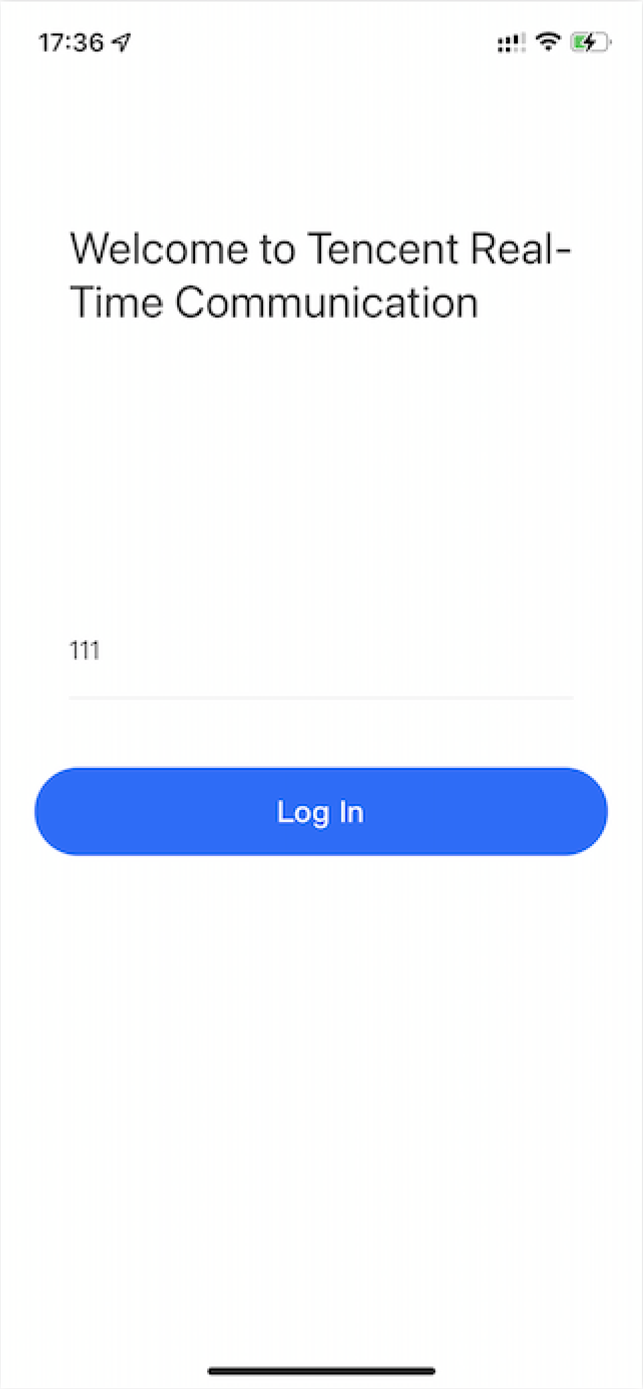
Tryout

Note :

You need at least two devices to try out the application.

User A

1. Enter a username (**which must be unique**) and log in.

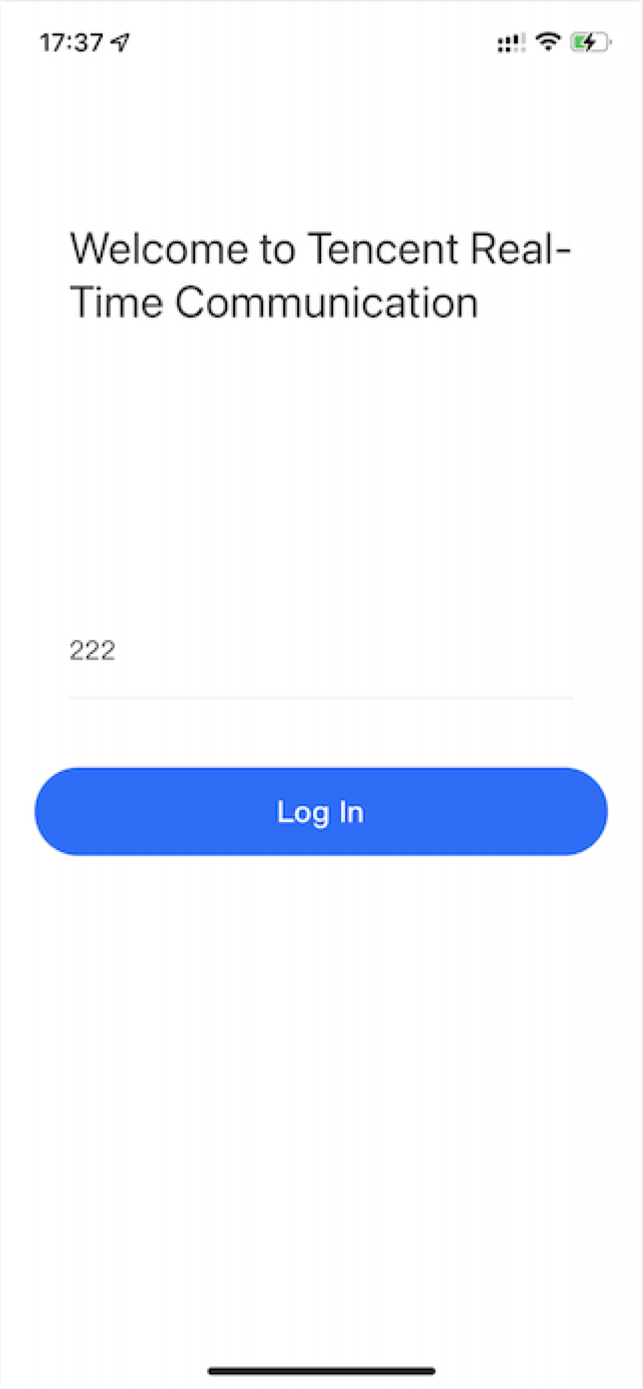


2. Enter the `userId` of the person you want to call and tap **Search**.

3. Tap **Call** and select **Video Call** (**Make sure that the callee is active in the application, or the call may fail**).

User B

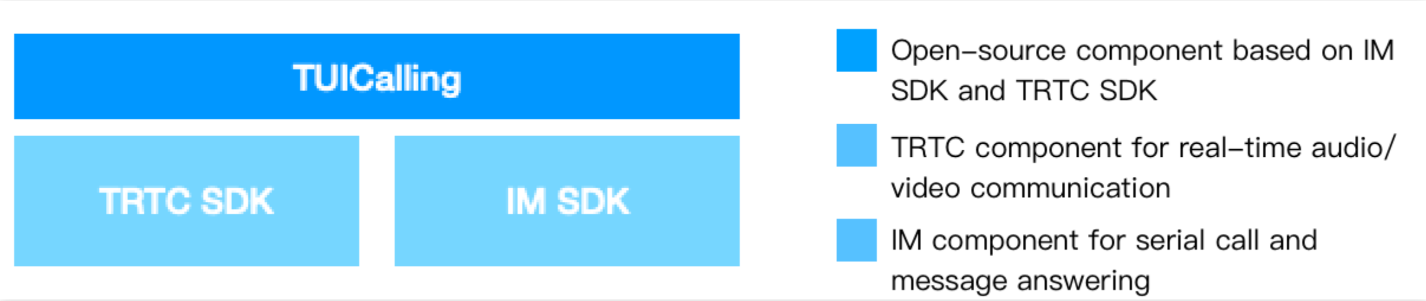
1. Enter a username (**which must be unique**) and log in.



2. Go to the homepage and wait for the incoming call.

Integration Directions

In the source code, the `Source` folder contains three subfolders: `ui` , `model` , and `Service` . The `Service` subfolder includes the open-source `TUICallingManager` component, which we share with the public. You can find the component’s APIs in the `TUICallingManager.h` file.


TUICalling
TRTC SDK
IM SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for serial call and message answering

You can enable the real-time audio/video call feature for your project with ease using the open-source `TUICalling` and `TUICallingManager` components, with no need to implement complicated call UI or logic by yourself.

Step 1. Integrate the SDKs

The call component `TRTCCalling` depends on the TRTC SDK and IM SDK. You can integrate the two SDKs into your project by following the steps below:

- **Method 1: adding dependencies via CocoaPods**

```
pod ’TXIMSDK_iOS’
pod ’TXLiteAVSDK_TRTC’
```

- **Method 2: adding dependencies through local files**
  If your access to the CocoaPods repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration Documentation
IM SDK	Download	Integration document

Step 2. Configure permission requests

Configure camera and mic permission requests by adding `Privacy - Camera Usage Description` and `Privacy - Microphone Usage Description` in `info.plist`.

Step 3. Import the `TUICalling` component

To import the component through CocoaPods, follow the steps below:

1. Copy the `Source` , `Resources` , and `TXAppBasic` folders and the `TUICalling.podspec` file under the demo project directory to your project directory.
2. Add the following dependencies to your `Podfile` and run `pod install` to complete the import.

```
pod 'TXAppBasic', :path =&gt; "../TXAppBasic/"
pod 'TXLiteAVSDK_TRTC'
pod 'TUICalling', :path =&gt; "../", :subspecs =&gt; ["TRTC"]
```

Step 4. Initialize the component and log in

1. Call `TUICallingManager.sharedInstance()` to initialize the component.
2. Call `TUILogin.initWithSdkAppID(SDKAPPID)` to initialize login.
3. Call `TUILogin.login(userId, userSig)` to log in to the component. Specify the key parameters as described below:

Parameter	Description
sdkAppID	You can view `SDKAppID` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. For the calculation method, please see UserSig.

```
// Initialize the component
TUICallingManager.sharedInstance();
// Log in to the component
TUILogin.initWithSdkAppID(SDKAPPID)
TUILogin.login(userId, userSig) {
print("login success")
} fail: { code, errorDes in
print("login failed, code:¥(code), error: ¥(errorDes ?? "nil")")
}
```

Step 5. Make an audio/video call

1. Caller: call `call();` of `TUICallingManager` to initiate a call, passing in the user IDs (`userIDs`) and call type (`type`). For the call type, you can pass in `.audio` (audio call) or `.video` (video call). If only one user ID is passed in for `userIDs` , the call is a one-to-one call; if two or more user IDs are passed in, the call is a group call.

2. Callee: If a callee is logged in, the answering view will be displayed. If you want offline users to receive call invitations, please see Enable offline call answering.

```
// 2. Register the listener
TUICallingManager.shareInstance().setCallingListener(listener: TUICallingListerner())

// 2. Set whether to enable custom views (disabled by default)
TUICallingManager.shareInstance().enableCustomViewRoute(enable: true)

// 3. Set callbacks
public func shouldShowOnCallView() -&gt; Bool {
return true;
}

public func callStart(userIDs: [String], type: TUICallingType, role: TUICallingRole, viewControll
er: UIViewController?) { if let vc = viewController {
callingVC = vc;
vc.modalPresentationStyle = .fullScreen

if var topController = UIApplication.shared.keyWindow?.rootViewController {
while let presentedViewController = topController.presentedViewController {
topController = presentedViewController
}

if let navigationVC = topController as? UINavigationController {
if navigationVC.viewControllers.contains(self) {
present(vc, animated: false, completion: nil)
} else {
navigationVC.popToRootViewController(animated: false)
navigationVC.pushViewController(self, animated: false)
navigationVC.present(vc, animated: false, completion: nil)
}
} else {
topController.present(vc, animated: false, completion: nil)
}
}
}
}

public func callEnd(userIDs: [String], type: TUICallingType, role: TUICallingRole, totalTime: Flo
at) {
callingVC.dismiss(animated: true, completion: nil)
}

public func onCallEvent(event: TUICallingEvent, type: TUICallingType, role: TUICallingRole, messa
ge: String) {
}
// 4. Make a call
TUICallingManager.shareInstance().call(userIDs, .video)
```

Step 6. Enable offline call answering

> Note：
> If your project does not require the offline answering feature, for example, if it offers online customer service, your integration can end at step 5. If your project is a social networking service, we recommend you enable offline answering.

The IM SDK supports offline push, but additional configuration is required to enable the feature.

1. Apply for an Apple push certificate. For detailed directions, please see Obtaining Apple Push Notification Service Certificates.
2. Configure offline push on the backend and client. For detailed directions, please see Offline Push (iOS).
3. The offline push API has been integrated into the signaling API (`sendModel`) of `TRTCCallingImpl` . After completing the offline push configuration for your application, you will be able to send notifications to offline users.

Component APIs

The table below lists the APIs of the `TUICalling` component.

API	Description
call	Sends call invitations by user ID.
receiveAPNSCalled	Answers a call.
setCallingListener	Sets the listener.
setCallingBell	Sets the ringtone (preferably shorter than 30s).
enableMuteMode	Enables/Disables the mute mode.

API	Description
enableCustomViewRoute	Enables/Disables custom views. After enabling custom views, you will receive a `CallingView` instance in the callback for calling/being called, and can decide how to display the view by yourself. The view must be displayed full screen or in proportion to the screen size; otherwise, an error may occur.

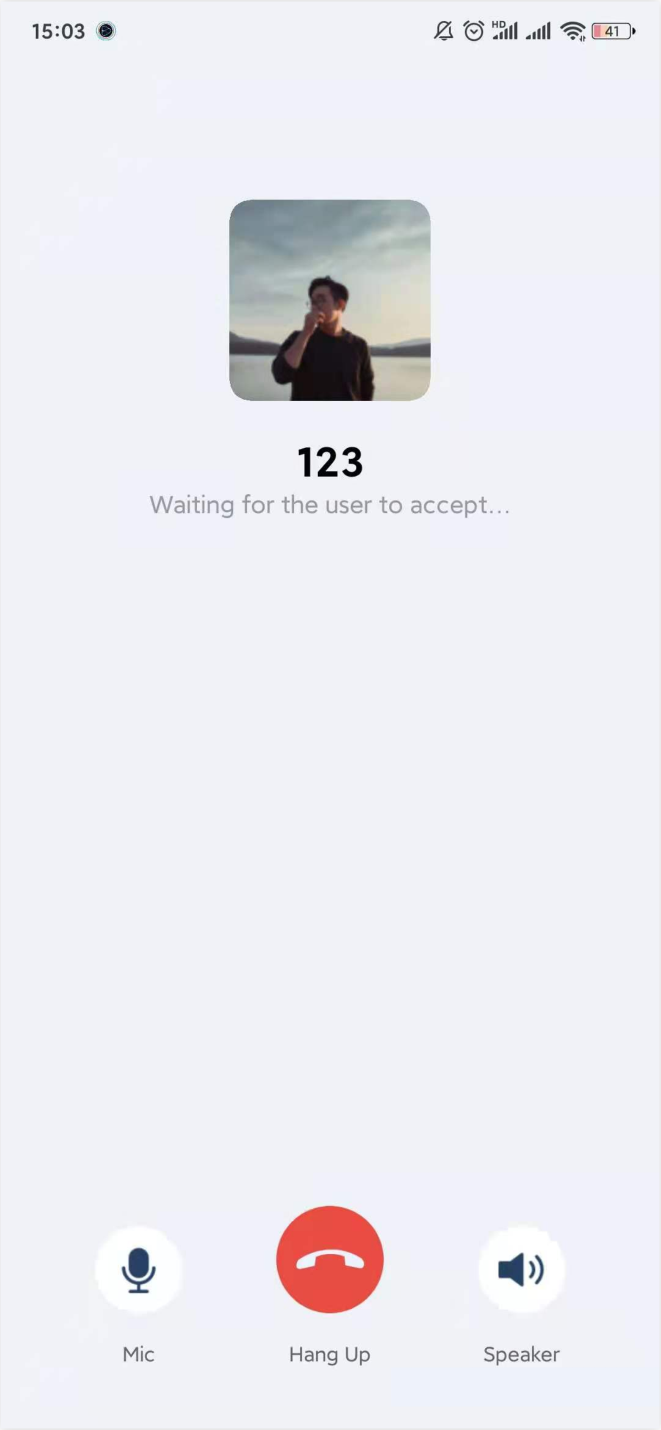
Audio Call (Android)

Last updated : 2022-03-17 15:41:03

Demo UI

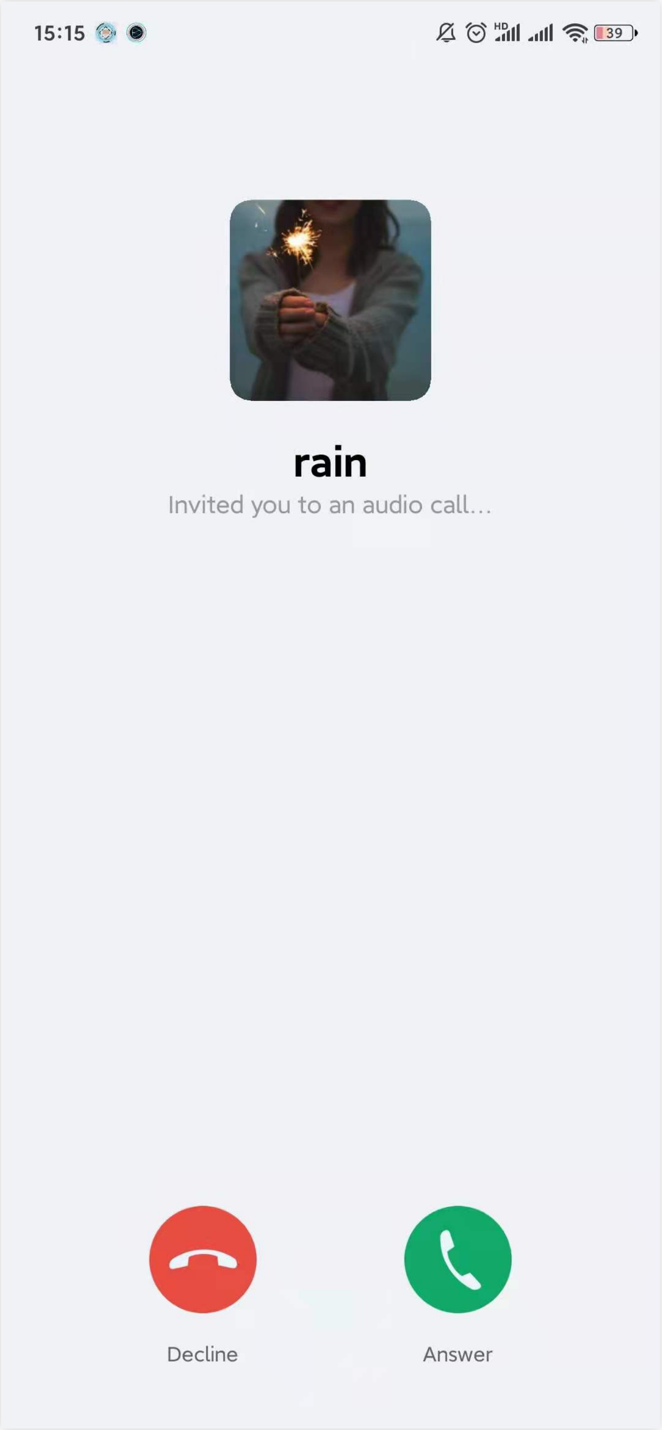
You can download and install the app we provide to try out the real-time audio/video call feature.

Call	Answer



123

Waiting for the user to accept...



rain

Invited you to an audio call...

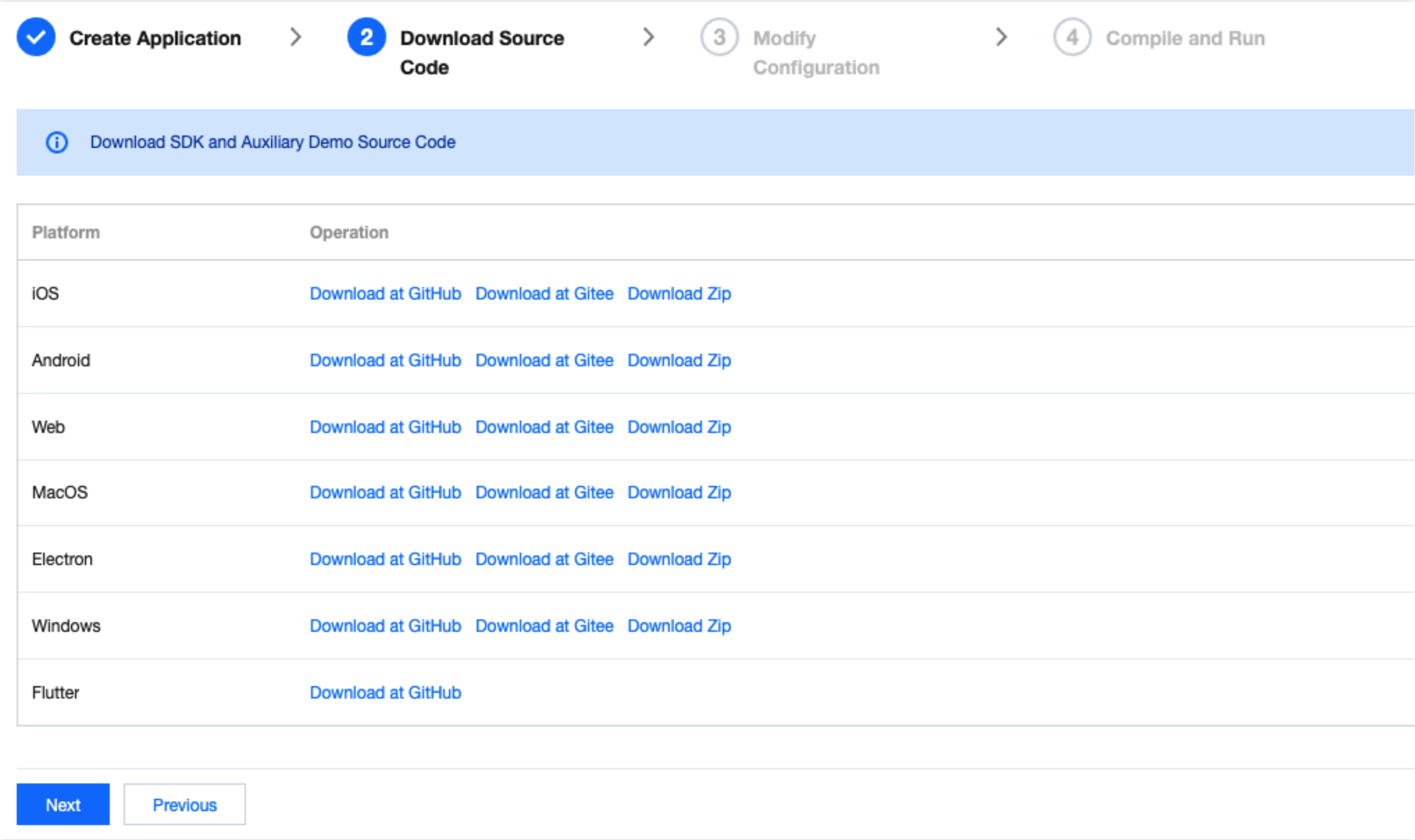
Note：

To make it easier for you to implement the real-time audio/video call feature, we have refactored the `TUICalling` component. You no longer need to pay attention to UI as it is now implemented within the `TUICalling` component.

Running the Demo

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestVideoCall` and click **Create**.
3. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

> Note :
> This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Value-added Service Pricing for its billing details.

Step 2. Download the application source code

Clone or download the TUICalling source code.

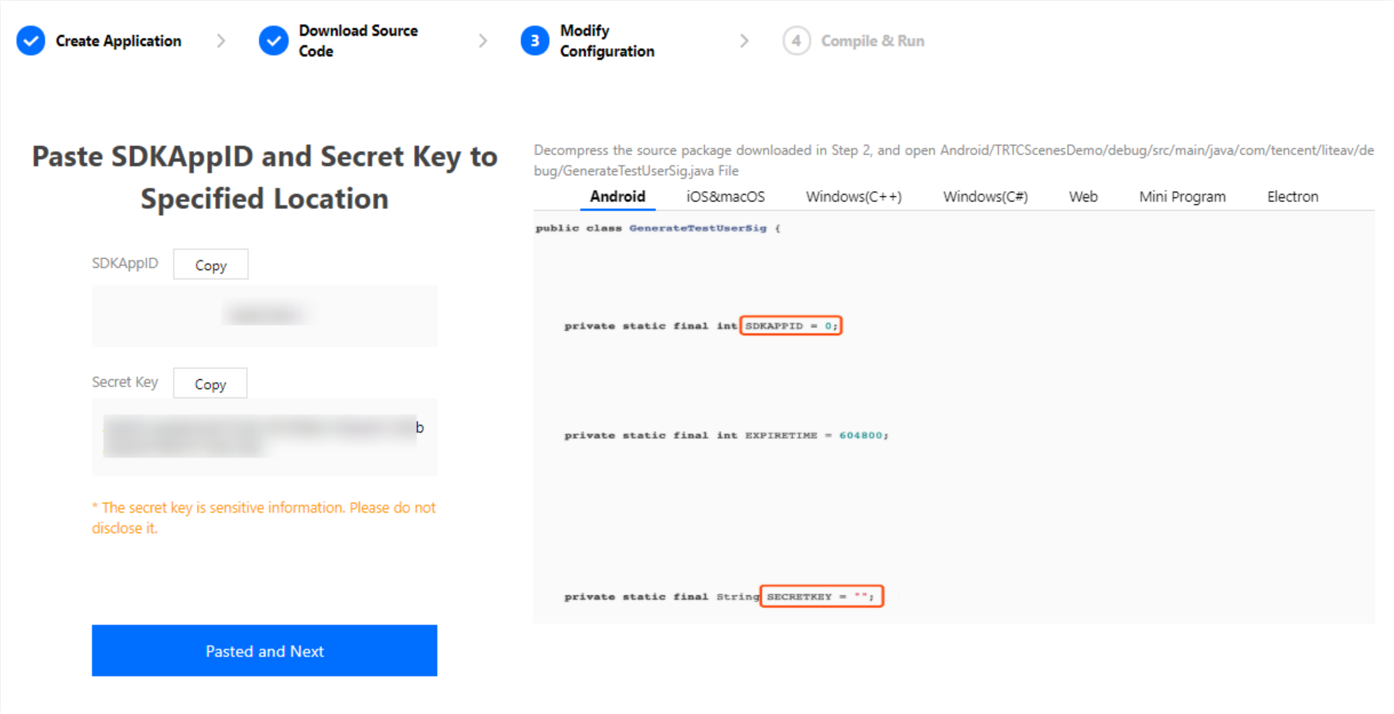
Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `Android/Debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java`.

3. Set parameters in `GenerateTestUserSig.java`:

   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your

application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUICalling` with Android Studio (version 3.5 or above) and click **Run**.

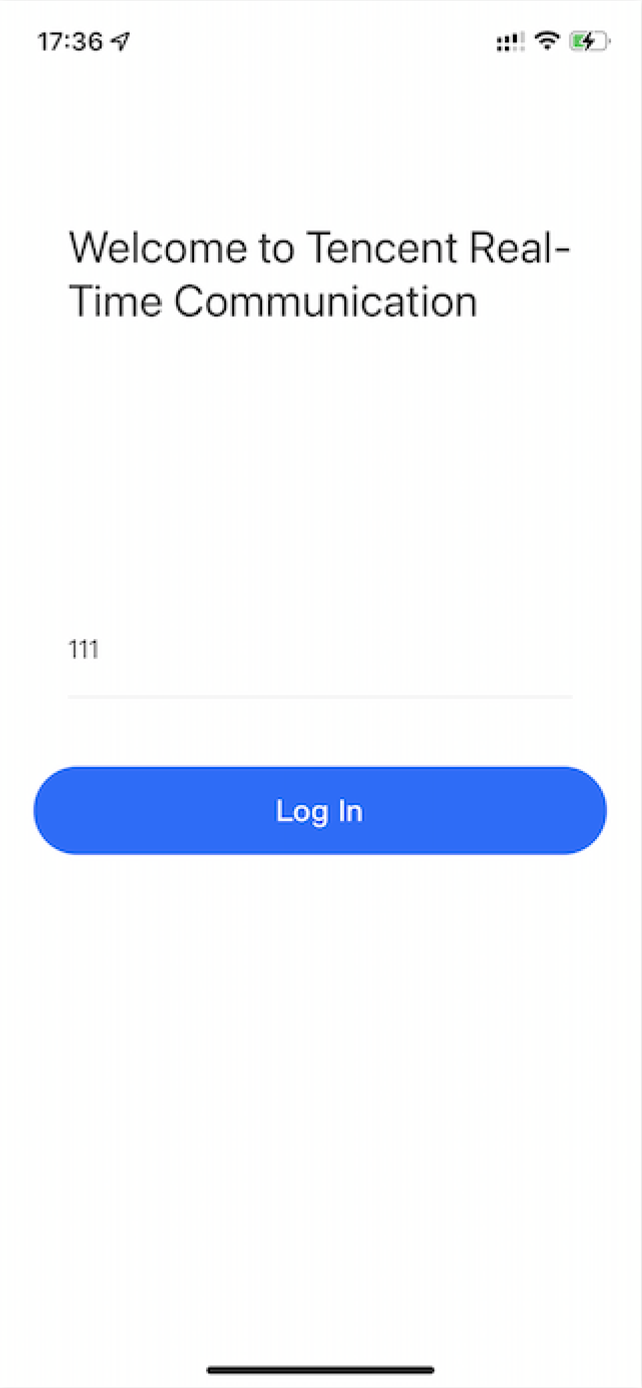
Tryout

Note :

You need at least two devices to try out the application.

User A

1. Enter a username (**which must be unique**) and log in.



17:36

Welcome to Tencent Real-Time Communication

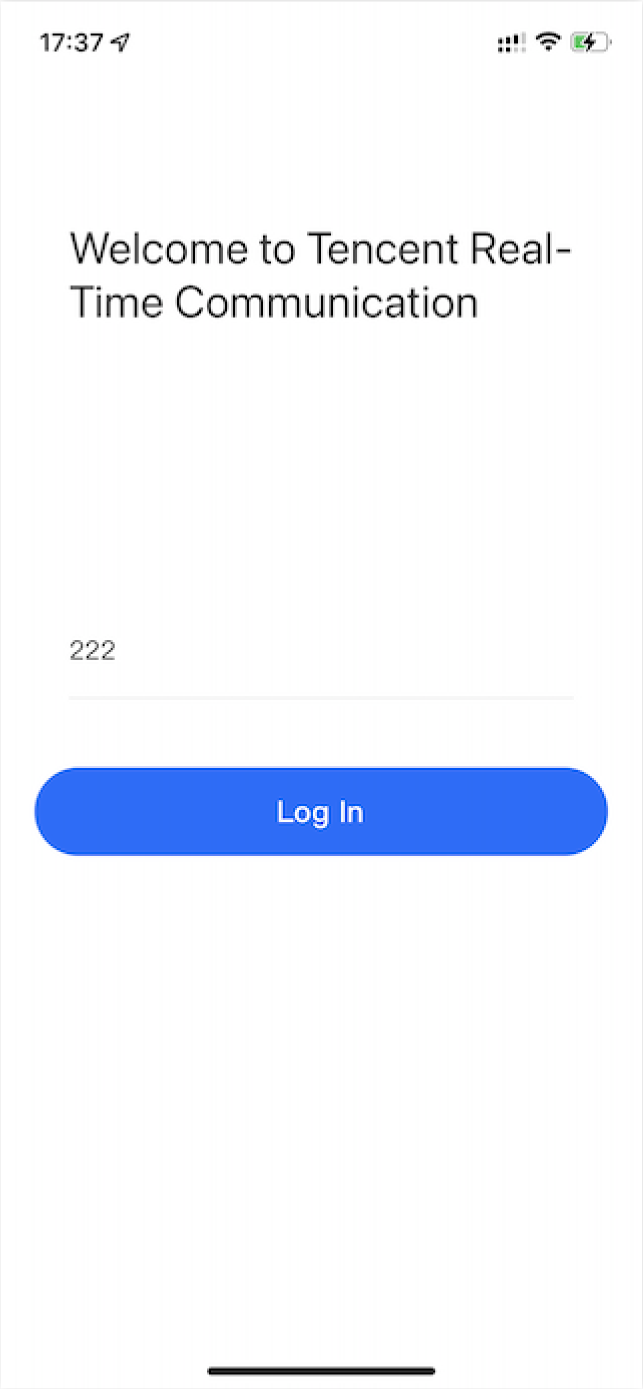
111

2. Enter the `userId` of the person you want to call and tap **Search**.

3. Tap **Call** and select **Video Call** (**Make sure that the callee is active in the application, or the call may fail**).

User B

1. Enter a username (**which must be unique**) and log in.

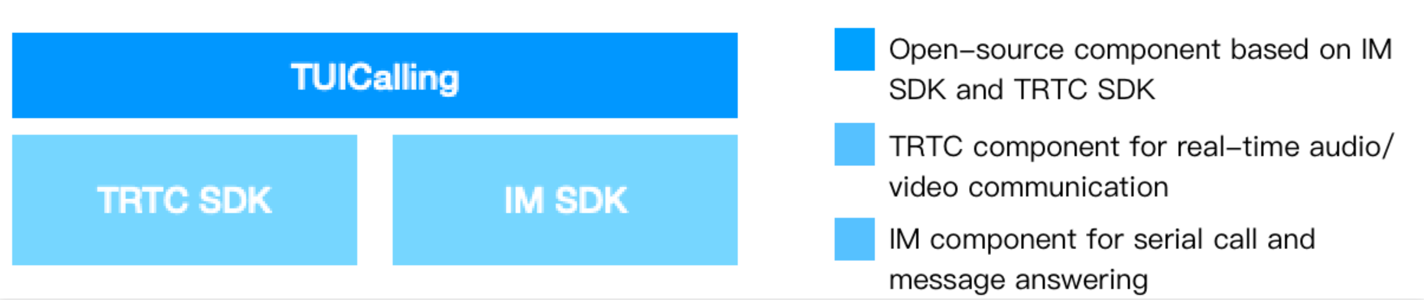


2. Go to the homepage and wait for the incoming call.

Integration Directions

In the source code, the `Source` folder contains two subfolders: `ui` and `model`. The `model` subfolder includes the open-source `TUICallingManager` component, which we share with the public.

You can find the component's APIs in the `TUICalling.java` file.


TUICalling
TRTC SDK
IM SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for serial call and message answering

You can enable the real-time audio/video call feature for your project with ease using the open-source `TUICalling` and `TUICallingManager` components, with no need to implement complicated call UI or logic by yourself.

Step 1. Integrate the SDKs

The audio/video call component `TUICalling` depends on the TRTC SDK and IM SDK. Follow the steps below to integrate the two SDKs into your project:

Method 1: adding dependencies via mavenCentral

1. Add the TRTC SDK and IM SDK dependencies to `dependencies` .

```
dependencies {
compile "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"
compile 'com.tencent.imsdk:imsdk:latest.release'

// As we use Gson for parsing, you also need to add the Google Gson dependency.
compile 'com.google.code.gson:gson:latest.release'
}
```

2. In `defaultConfig` , specify the CPU architecture to be used by your application.

```
defaultConfig {
ndk {
abiFilters "armeabi-v7a"
}
}
```

3. Click **Sync Now** to sync the SDKs.

Note :

If you have no problem connecting to mavenCentral, the SDKs will be downloaded and integrated into your project automatically.

Method 2: adding dependencies through local AAR files

If your access to the mavenCentral repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration document
IM SDK	Download	Integration documentation

Step 2. Configure permission requests and obfuscation rules

Configure permission requests for your application in `AndroidManifest.xml` . The SDKs need the following permissions (on Android 6.0 and above, the camera and read storage permissions must be requested at runtime.)

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-feature android:name="android.hardware.camera"/>
<uses-feature android:name="android.hardware.camera.autofocus" />
```

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

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

Step 3. Import the `TUICalling` component

Copy the `Source` directory to your project and import it in `setting.gradle` as shown below:

```
include ':Source'
```

Step 4. Initialize the component and log in

1. Call `TUICallingManager.sharedInstance()` to initialize the component.
2. Call `TUILogin.init(context, SDKAppID, config, listener)` to initialize login.
3. Call `TUILogin.login(userId, userSig, callback)` to log in to the component. Specify the key parameters as described below:

Parameter	Description
SDKAppID	You can view `SDKAppID` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. For the calculation method, please see UserSig.
config	SDK configuration, which is used to set the log output level and log callback. You can pass `null` for this parameter. For details, see the sample code below.
listener	IM listener, which is used to listen for some crucial callbacks, such as forced logout and `userSig` expiration. For details, see the sample code below.
callback	Callback for login, which indicates whether login is successful. For details, see the sample code below.

```
// Initialize the component
TUICallingManager manager = TUICallingManager.sharedInstance();
// Log in to the component
V2TIMSDKConfig config = new V2TIMSDKConfig();
config.setLogLevel(V2TIMSDKConfig.V2TIM_LOG_DEBUG);
config.setLogListener(new V2TIMLogListener() {
@Override
public void onLog(int logLevel, String logContent) {

}
});
TUILogin.init(this, ${Your `SDKAPPID`}, config, new V2TIMSDKListener() {
@Override
public void onKickedOffline() { // Callback for forced logout
mIsKickedOffline = true;
checkUserStatus();
}
@Override
public void onUserSigExpired() { // Callback for `userSig` expiration
mIsUserSigExpired = true;
checkUserStatus();
}
});
TUILogin.login("${Your `userId`}", "${Your `userSig`}", new V2TIMCallback() {
@Override
public void onError(int code, String msg) {
Log.d(TAG, "code: " + code + " msg:" + msg);
}
@Override
public void onSuccess() {
Log.d(TAG, "onSuccess");
}
});
```

Step 5. Make an audio/video call

1. Caller: Call `call();` of `TUICallingManager` to initiate a call, passing in the user IDs (`userids`) and call type (`type`). For the call type, you can pass in `TUICalling.Type.AUDIO` (audio call) or `TUICalling.Type.VIDEO` (video call). If only one user ID is passed in for `userids` , the call is a one-to-one call; if two or more user IDs are passed in, the call is a group call.
2. Callee: If a callee is logged in, the answering view will be displayed. If you want offline users to receive call invitations, please see Enable offline call answering.

```
// 1. Initialize the component
TUICallingManager manager = TUICallingManager.sharedInstance();
// 2. Register the listener
manager.setCallingListener(new TUICalling.TUICallingListener() {
@Override
public boolean shouldShowOnCallView() {
return true;
}

@Override
public void onCallStart(String[] userIDs, TUICalling.Type type, TUICalling.Role role, final View
tuiCallingView) {
if (!shouldShowOnCallView() || null == tuiCallingView) {
return;
}
runOnUiThread(new Runnable() {
@Override
public void run() {
FrameLayout.LayoutParams params = new FrameLayout.LayoutParams(FrameLayout.LayoutParams.MATCH_PARENT, FrameLayout.LayoutParams.MATCH_PARENT);
mCallingView = tuiCallingView;
addContentView(tuiCallingView, params);
}
});
}

@Override
public void onCallEnd(String[] userIDs, TUICalling.Type type, TUICalling.Role role, long totalTime) {
removeView();
}

@Override
public void onCallEvent(TUICalling.Event event, TUICalling.Type type, TUICalling.Role role, String message) {
if (TUICalling.Event.CALL_FAILED == event) {
removeView();
}
}
});
// 3. Make a call
manager.call(userIDs, TUICalling.Type.VIDEO);
```

Step 6. Enable offline call answering

> Note :
> If your project does not require the offline answering feature, for example, if it offers online customer service, your integration can end at step 5. If your project is a social networking service, we recommend you enable offline answering.

The IM SDK supports offline push. However, since the offline push service of Android phones varies from vendor to vendor, the configuration required to enable offline push for Android is more complicated than that for iOS.

1. Apply for a certificate required by the vendor's push channel, configure it in the IM console, and add the certificate and ID as required. For detailed directions, please see IM > Offline Push (Android).
2. The offline push API has been integrated into the signaling API (`sendModel`) of `TRTCCallingImpl` . After completing the offline push configuration for your application, you will be able to send

notifications to offline users.

Component APIs

The table below lists the APIs of the `TUICalling` component.

API	Description
call	Sends call invitations by user ID.
receiveAPNSCalled	Answers a call.
setCallingListener	Sets the listener.
setCallingBell	Sets the ringtone (preferably shorter than 30s).
enableMuteMode	Enables/Disables the mute mode.
enableCustomViewRoute	Enables/Disables custom views. After enabling custom views, you will receive a `CallingView` instance in the callback for calling/being called, and can decide how to display the view by yourself. The view must be displayed full screen or in proportion to the screen size; otherwise, an error may occur.

Audio Call (Web)

Last updated : 2022-04-02 16:27:03

This document describes how to implement browser-based audio calls.

- Part 1 describes how to activate the service and run the demo.
- Part 2 describes how to build your own audio call feature using the `TRTCCalling` component.

Environment Requirements

Currently, the desktop version of Chrome offers better support for the features of the TRTC Web SDK; therefore, Chrome is recommended for the demo.

`TRTCCalling` uses the following ports and domain name for data transfer, which should be added to the allowlist of the firewall. After configuration, please use official demo to check whether the configuration has taken effect.

- TCP port: 8687
- UDP ports: 8000, 8080, 8800, 843, 443, 16285
- Domain name: qcloud.rtc.qq.com
  For details, please see Dealing with Firewall Restrictions.

Supported Platforms

The service supports the following platforms:

OS	Browser (Desktop)	Minimum Browser Version Requirement
macOS	Safari	11+
macOS	Chrome	56+
macOS	Firefox	56+
macOS	Edge	80+
Windows	Chrome	56+
Windows	QQ Browser (WebKit core)	10.4+
Windows	Firefox	56+

OS	Browser (Desktop)	Minimum Browser Version Requirement
Windows	Edge	80+

> Note :
> For more information on browser compatibility, please see Browsers Supported. You can also run an online test using the TRTC compatibility check page.

URL Protocol Support

Scenario	Protocol	Receive (Playback)	Send (Publish)	Share Screen	Remarks
Production	HTTPS	Supported	Supported	Supported	Recommended
Production	HTTP	Supported	Not supported	Not supported	-
Local development	http://localhost	Supported	Supported	Supported	Recommended
Local development	http://127.0.0.1	Supported	Supported	Supported	-
Local development	http://[local IP address]	Supported	Not supported	Not supported	-
Local development	file:///	Supported	Supported	Supported	-

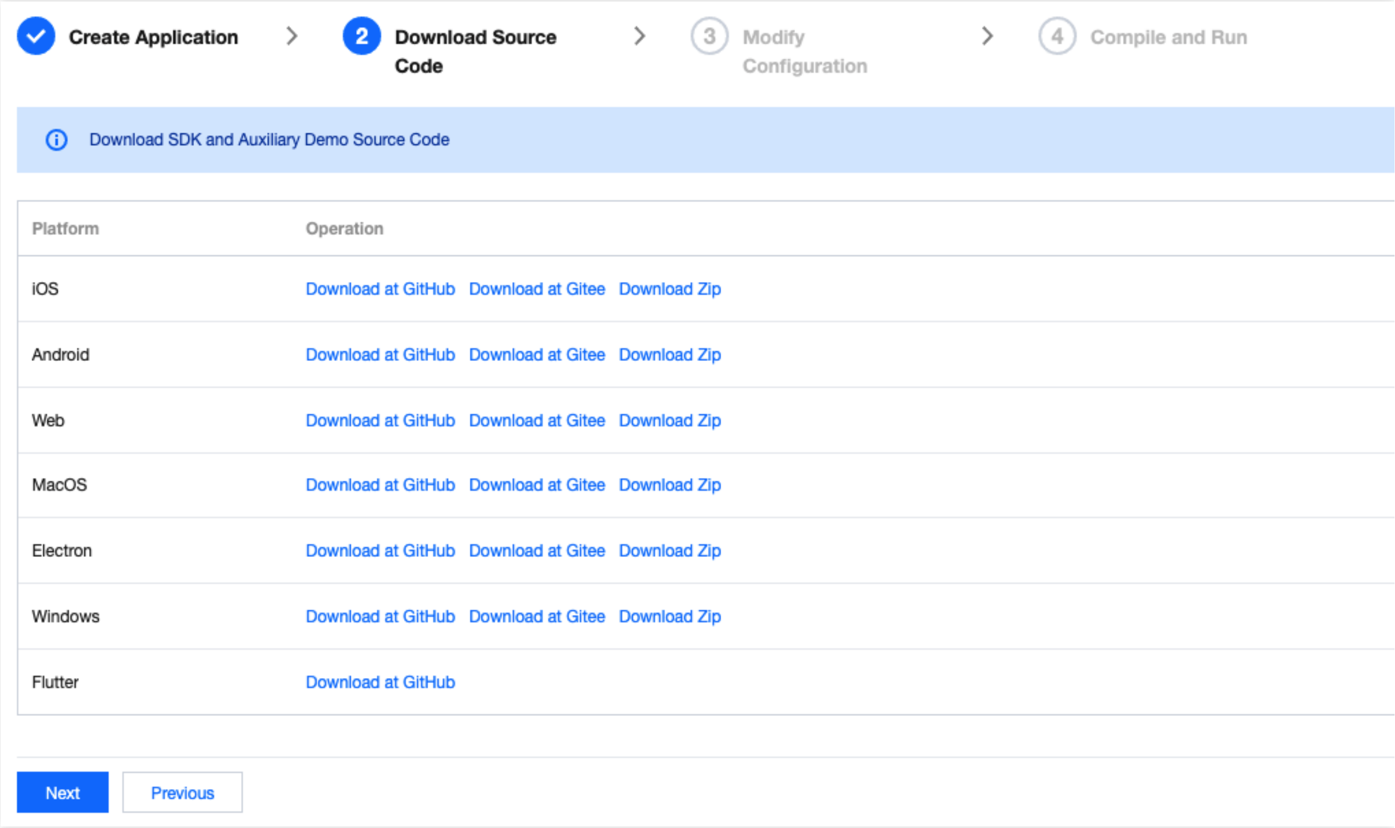
Running the Demo

Step 1. Create an application

1. Sign up for a Tencent Cloud account and verify your identity.
2. In the TRTC console, click **Development Assistance** > **Demo Quick Run**.
3. Enter an application name such as `TestTRTC` and click **Create**.

Step 2. Download the demo

1. You can download the source code of the web demo at TUICalling.
2. Click **Next**.



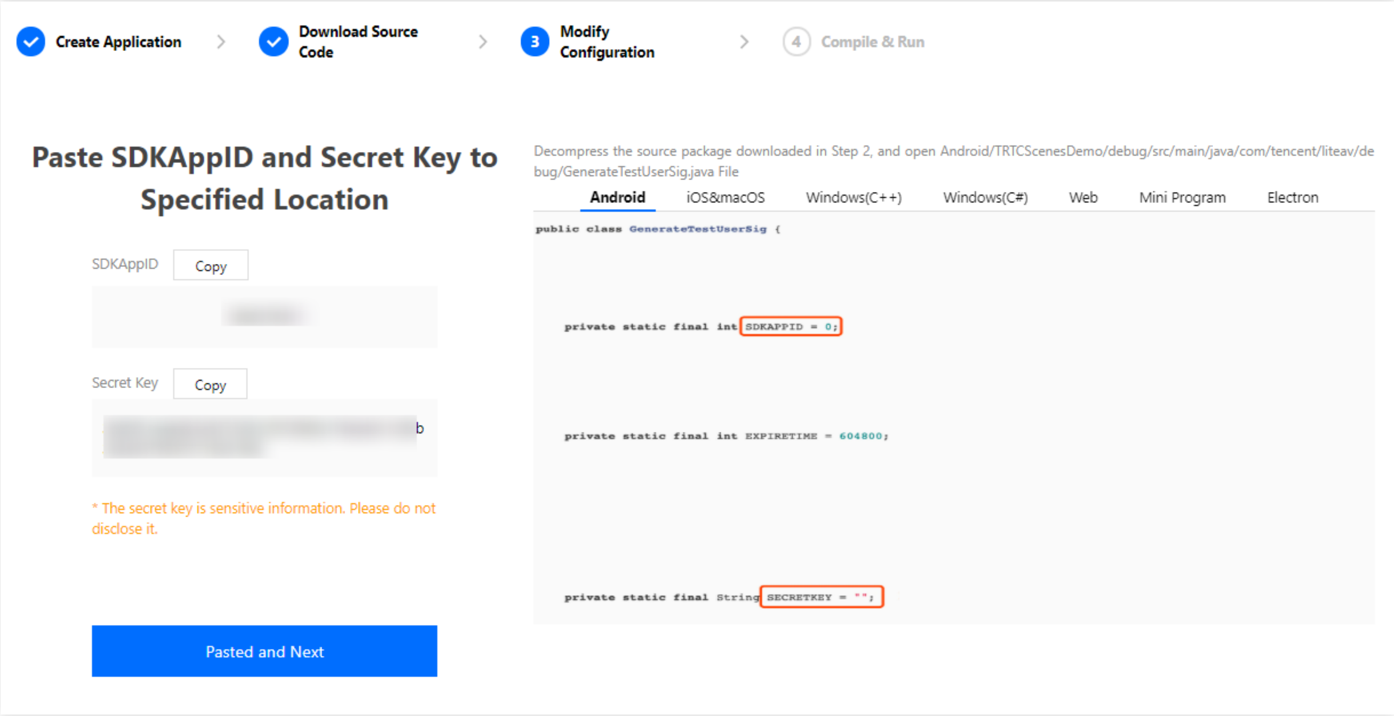
✓ Create Application > 2 Download Source Code > 3 Modify Configuration > 4 Compile and Run

ⓘ Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Step 3. Configure the demo project file

1. In the **Modify Configuration** step, select your platform.
2. Find and open the `Web/public/debug/GenerateTestUserSig.js` file.
3. Set the following parameters in the `GenerateTestUserSig.js` file:
   - SDKAPPID: `0` by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: Left empty by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted] b

* The secret key is sensitive information. Please do not disclose it.

Pasted and Next

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.
5. After compilation, click **Return to Overview Page**.

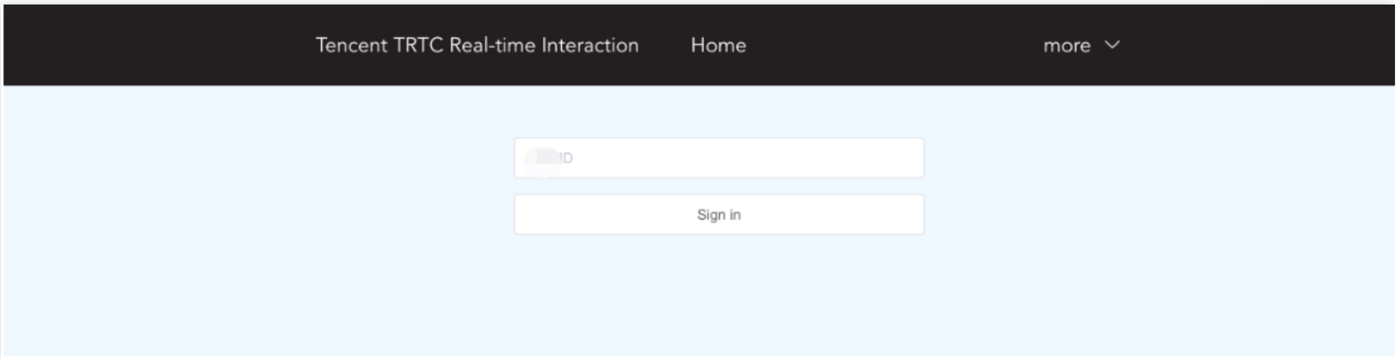
> Note :
>
> - In this document, the method to obtain UserSig is to configure a SECRETKEY in the client code. In this method, the SECRETKEY is vulnerable to decompilation and reverse engineering. Once your SECRETKEY is leaked, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and feature debugging**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 4. Run the demo

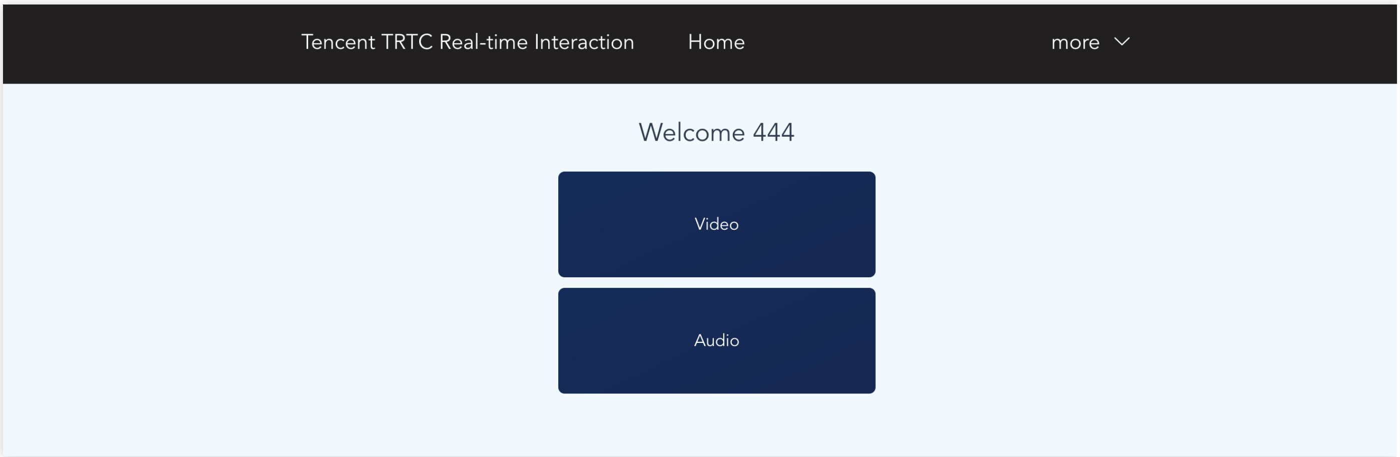
1. In the demo directory, run the following commands in turn:

```
npm install
npm run serve
```

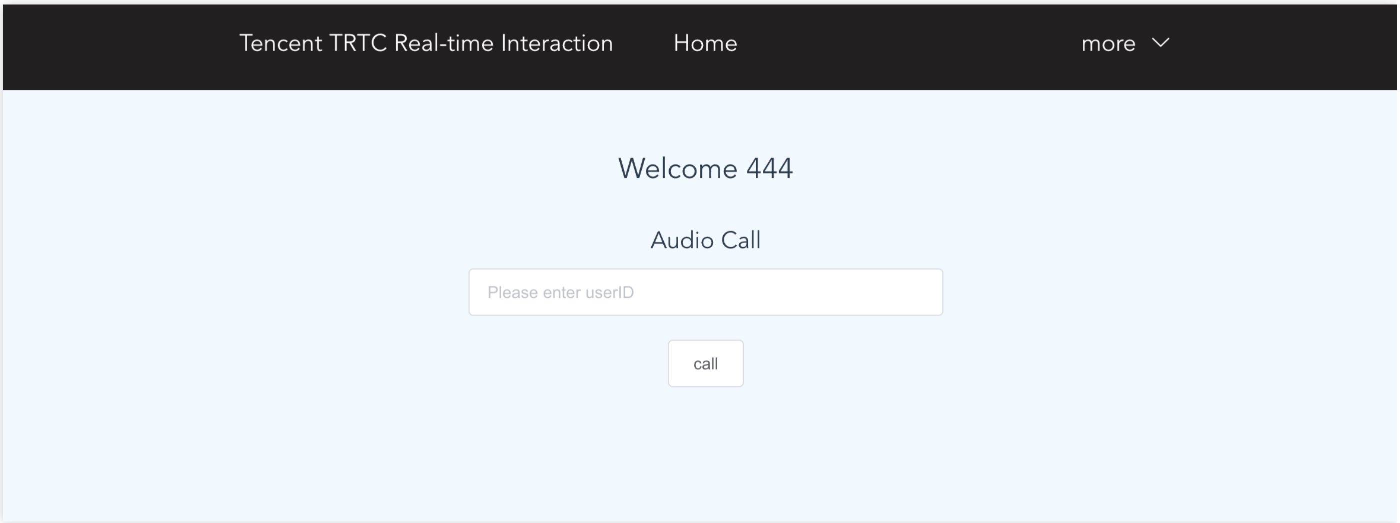
2. Open Chrome and visit `http://localhost:8080/` . If the above steps are performed correctly, you will see the page below:



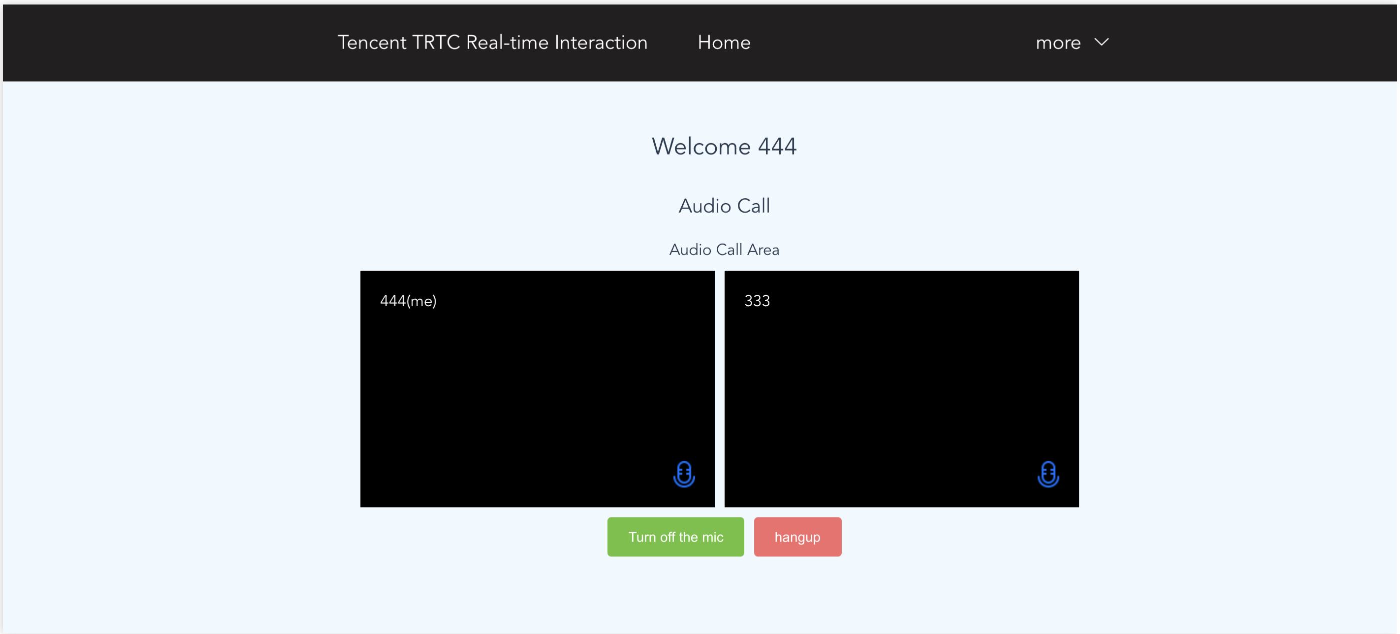
3. Enter a user ID, click **Sign in**, and select **Audio**.



4. Enter the user ID of the callee and click **call**.



5. Start the audio call.



Building Your Own Audio Call Feature

Step 1. Import the `TRTCCalling` component

Note :

- Since version 0.6.0, you need to manually install dependencies trtc-js-sdk, tim-js-sdk, and tsignaling.
- To reduce the size of `trtc-calling-js.js` and prevent version conflict between `trtc-calling-js.js` and the already-in-use `trtc-js-sdk`, `tim-js-sdk` or `tsignaling`, the latter three are packaged as external dependencies, which you need to install manually before use.

```
// Import via npm
npm install trtc-js-sdk --save

npm install tim-js-sdk --save

npm install tsignaling --save

npm install trtc-calling-js --save
```

```
// If you import `trtc-calling-js` via a script, you need to manually import the following resources in the specified order.

<script src=”./trtc.js”></script>
<script src=”./tim-js.js”></script>
<script src=”./tsignaling.js”></script>
<script src=”./trtc-calling-js.js”></script>
```

Step 2. Create a `TRTCCalling` object

Create a `TRTCCalling` object, setting `SDKAppID` to the `SDKAppID` of your application.

```
import TRTCCalling from ’trtc-calling-js’;

let options = {
SDKAppID: 0, // Replace 0 with your `SDKAppID` when connecting
// The `tim` parameter is added starting from v0.10.2
// The parameter guarantees the uniqueness of an existing TIM instance.
tim: tim
};
const trtcCalling = new TRTCCalling(options);
```

Step 3. Log in

```
trtcCalling.login({
userID,
userSig,
});
```

Step 4. Make a one-to-one call

- **Caller: call a user**

```
trtcCalling.call({
userID, // User ID
type: 1, // Call type. `0`: unknown; `1`: audio call; `2`: video call
timeout // Timeout threshold, in seconds
});
```

- **Callee: process a call invitation**

```
// Answer
trtcCalling.accept();
// Reject
trtcCalling.reject()
```

- **Hang up**

```
trtcCalling.hangup()
```

FAQs

Why can’t I get through to a user? Why am I kicked offline?

The `TRTCCalling` component does not support login of multiple instances or **offline signaling** for the time being. Please make sure that your current login is unique.

> Note :
>
> - **Multiple instances**: A user logs in with the same account multiple times or on different devices, which disrupts signaling.
> - **Offline signaling**: Only online instances can receive messages. Messages sent to offline instances will not be sent again when the instances go online.
>   For FAQs, see TRTCCalling for Web.

Technical Support

If you have other questions, you can fill out a contact form or email colleenyu@tencent.com.

Learn More

- TRTCCalling web demo
- TRTCCalling for npm
- Source code of TRTCCalling web demo
- TRTCCalling web APIs
- FAQs

Audio Call (Flutter)

Last updated : 2022-03-17 15:45:31

To quickly implement the audio call feature, you can use the demo we provide and adapt it to your needs. You can also use the `TRTCCalling` component and customize your own UI.

Using the Demo UI

Step 1. Create an application

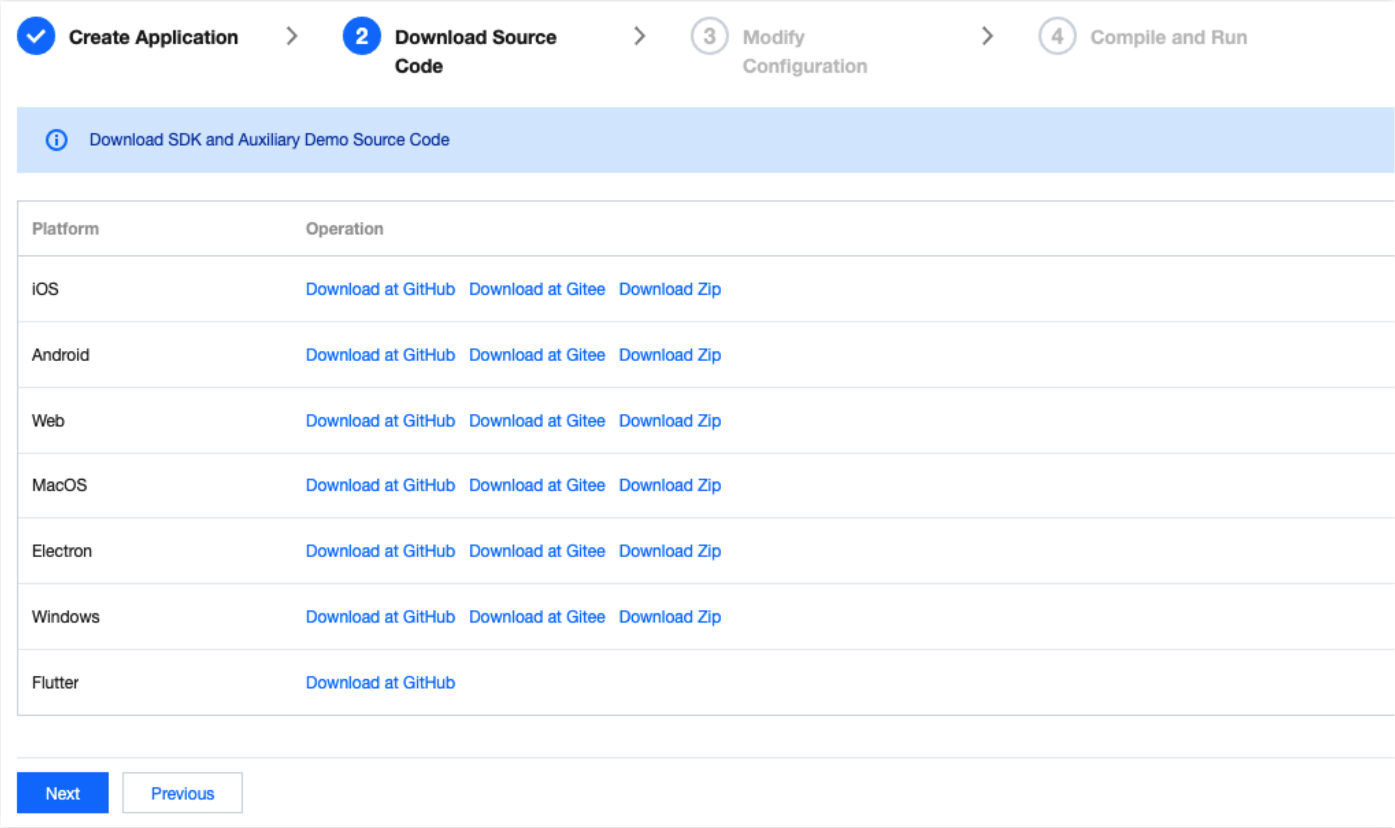
1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestAudioCall` , and click **Create**.

> Note :
> This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the SDK and demo source code

1. Download the SDK and demo source code for your platform.

2. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Step 3. Configure the demo project file

1. In the **Modify Configuration** step, select your platform.
2. Find and open `/example/lib/debug/GenerateTestUserSig.dart` .
3. Set parameters in `GenerateTestUserSig.dart` as follows.
   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.
   - `SECRETKEY`: a placeholder by default. Set it to the actual key.

   ![](https://main.qcloudimg.com/raw/87dc814a675692e76145d76aab91b414.png)
4. Click **Next** to complete the creation.
5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is leaked, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for the local execution and debugging of the demo**.

- The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Run the demo

Note :

An Android project must be run on a real device rather than a simulator.

1. Run `flutter pub get`.
2. Compile, run, and test the project.
   - Android
   - iOS
   i. Run `flutter run`.
   ii. Open the demo project with Android Studio (3.5 or above), and click **Run**.

Step 5. Modify the demo source code

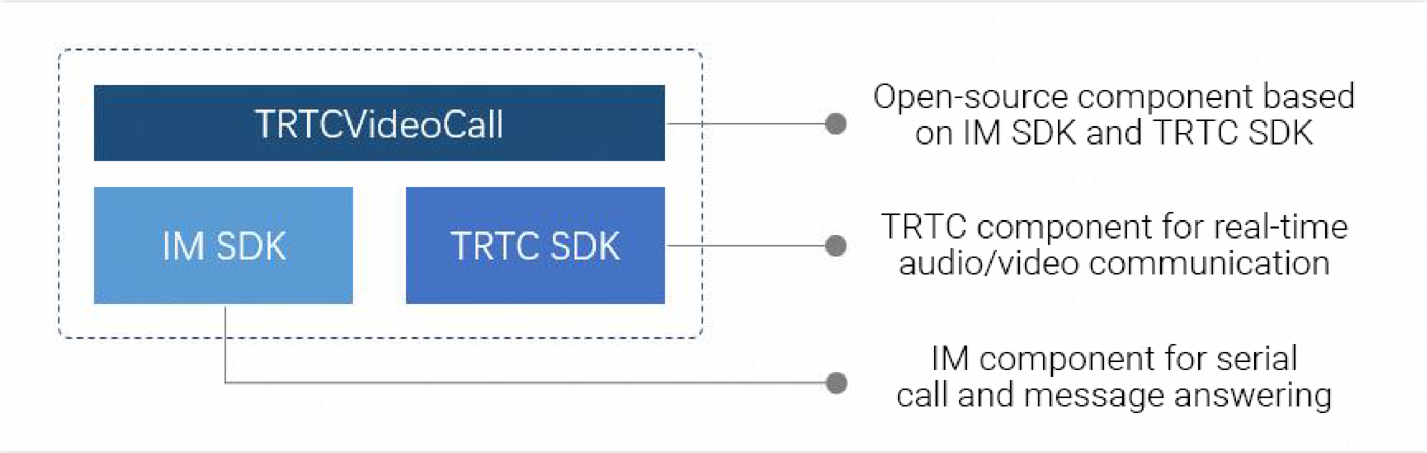
The `TRTCCallingDemo` folder in the source code contains two subfolders: `ui` and `model`. The `ui` folder contains the UI code.

File or Folder	Description
TRTCCallingVideo.dart	The main view for audio/video calls, where calls are answered/declined
TRTCCallingContact.dart	The view for contacts, where one can search for registered users to call

Customizing UI

The `TRTCCallingDemo` folder in the source code contains two subfolders: `ui` and `model`. The `model` subfolder contains the reusable open-source component `TRTCCalling`. You can find the component's

APIs in `TRTCCalling.dart` .


TRTCVideoCall
IM SDK
TRTC SDK
Open-source component based on IM SDK and TRTC SDK
TRTC component for real-time audio/video communication
IM component for serial call and message answering

You can use the open-source component `TRTCCalling` to customize your own UI. This means you will use the model of the demo but design the UI by yourself.

Step 1. Integrate the SDK

The audio/video call component `TRTCCalling` depends on the TRTC SDK and IM SDK. You can configure `pubspec.yaml` to download their updates automatically.

Add the following dependencies to `pubspec.yaml` of your project.

```
dependencies:
tencent_trtc_cloud: latest version number
tencent_im_sdk_plugin: latest version number
```

Step 2. Configure permission requests and obfuscation rules

- iOS
- Android

Add request for mic permission in `Info.plist` :

```
<key>NSMicrophoneUsageDescription</key>
<string>Audio calls are possible only with mic access.</string>
```

Step 3. Import the `TRTCCalling` component

Copy all the files in model to your project.

```
/lib/TRTCCallingDemo/model
```

Step 4. Initialize the component and log in

1. Call `TRTCCalling.sharedInstance()` to get an instance of the component.
2. Call `login(SDKAppID, userId, userSig)` to log in to the component. For the key parameters passed in, see the table below.

Parameter	Description
SDKAppID	You can view \`SDKAppID\` in the TRTC console.
userId	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 (_). We recommend that you keep it in line with your user account system.
userSig	Tencent Cloud's proprietary security signature. For how to calculate and use a signature, please see UserSig.

```
// Initialize
sCall = await TRTCCalling.sharedInstance();
sCall.login(1400000123, ”userA”, ”xxxx”);
```

Step 5. Make a one-to-one audio call

1. The caller calls `call()` of `TRTCCalling`, passing in the user ID of the callee (`userid`) and call type (`type`). For an audio call, the call type should be `TRTCCalling.typeAudioCall` .
2. The callee, if logged in, will receive the `onInvited()` callback and can start the corresponding view based on the call type set by the inviter, which is represented by `callType` in the callback.
3. The callee can call `accept()` to answer the call and `openCamera()` to turn the local camera on. He or she can also call `reject()` to reject the call.
4. After communication is established between the caller and callee, they will both receive the `onUserAudioAvailable()` event notification, which indicates that they have received each other’s audio. Remote audio will be played back automatically by default.

Component APIs

The table below lists the APIs of the `TRTCCalling` component.

API	Description
registerListener	Registers a `TRTCCalling` listener, through which users can receive status notifications.

API	Description
unRegisterListener	Unregisters a listener.
destroy	Destroys an instance.
login	Logs in to IM. All features can be used only after login.
logout	Logs out of IM. Calls cannot be made after logout.
call	Makes a C2C call. The invitee will receive the `onInvited` event notification.
accept	Answers a call.
reject	Declines a call.
hangup	Ends a call.
setMicMute	Mutes/Unmutes the mic.
setHandsFree	Enables/Disables the hands-free mode.

TUICalling APIs (iOS)

Last updated : 2022-03-17 15:49:09

`TUICalling` is an open-source class based on two closed-source SDKs: Tencent Cloud Real-Time Communication (TRTC) and Instant Messaging (IM). It supports one-to-one and group audio calls. For detailed instructions how to implement it, see Real-Time Audio Call (iOS).

- TRTC SDK: The TRTC SDK is used as a low-latency audio/video call component.
- IM SDK: The IM SDK is used to send and process signaling messages.

`TUICalling` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton.
call	Sends call invitations by user ID.
receiveAPNSCalled	Answers a call.
setCallingListener	Sets the listener.
setCallingBell	Sets the ringtone (preferably shorter than 30s).
enableMuteMode	Enables/Disables the mute mode.
enableCustomViewRoute	Enables/Disables custom views.

`TUICallingListener` API Overview

Event callbacks

API	Description
shouldShowOnCallView	Callback of whether the answering view is displayed when there is an incoming call
onCallStart	Callback for starting calling. This callback is triggered for both callers and callees.

API	Description
onCallEnd	Callback for ending a call. This callback is triggered for both callers and callees.
onCallEvent	Call event callback

Type

Call type

Enumerated Type	Description
TUICallingTypeAudio	Audio call
TUICallingTypeVideo	Video call

Role

Role type

Enumerated Type	Description
TUICallingRoleCall	Caller
TTUICallingRoleCalled	Callee

Event

Event type

Enumerated Type	Description
TUICallingEventCallStart	The call started.
TUICallingEventCallSucceed	The call was connected successfully.
TUICallingEventCallEnd	The call ended.
TUICallingEventCallFailed	The call failed.

Basic SDK APIs

sharedInstance

This API is used to get a singleton of the `TUICallingManager` component.

```
+ (instancetype)shareInstance;
```

call

This API is used to send call invitations by user ID.

```
- (void)call:(NSArray<NSString *> *)userIDs type:(TUICallingType)type;
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	NSArray	List of the user IDs of call participants
type	TUICallingType	Call type: audio/video

receiveAPNSCalled

This API is used to answer a call.

```
- (void)receiveAPNSCalled:(NSArray<NSString *> *)userIDs type:(TUICallingType)type;
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	NSArray	List of the user IDs of call participants
type	TUICallingType	Call type: audio/video

setCallingListener

This API is used to set the listener.

```
- (void)setCallingListener:(id<TUICallingListerner>)listener;
```

The parameters are as detailed below:

Parameter	Type	Description
listener	TUICallingListener	Listener of the `TUIcalling` component

setCallingBell

This API is used to set the ringtone (preferably shorter than 30s).

```
- (void)setCallingBell:(NSString *)filePath;
```

The parameters are as detailed below:

Parameter	Type	Description
filePath	NSString	Path of the ringtone file

enableMuteMode

This API is used to enable/disable the mute mode.

```
- (void)enableMuteMode:(BOOL)enable;
```

The parameters are as detailed below:

Parameter	Type	Description
enable	BOOL	Whether to enable the mute mode

enableCustomViewRoute

This API is used to enable/disable custom views.
After enabling custom views, you will receive a `CallingViewController` instance in the callback for calling/being called, and can decide how to display the view by yourself.

> Note :
> The view must be displayed full screen; otherwise, an error may occur.

```
- (void)enableCustomViewRoute:(BOOL)enable;
```

The parameters are as detailed below:

Parameter	Type	Description
enable	BOOL	Whether to enable custom views

`TUICallingListener` Callback APIs

shouldShowOnCallView

Callback of whether the answering view is displayed when there is an incoming call.

```
- (BOOL)shouldShowOnCallView;
```

The parameters are as detailed below:

Parameter	Type	Description
Return value	BOOL	Whether the answering view is displayed

onCallStart

Callback for starting calling, which is triggered for both callers and callees.

```
- (void)callStart:(NSArray<NSString *> *)userIDs type:(TUICallingType)type role:(TUICallingRole)r
ole viewController:(UIViewController * _Nullable)viewController;
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	NSArray	List of the user IDs of call participants
type	TUICallingType	Call type: audio/video
role	TUICallingRole	Role type: caller/callee
viewController	UIViewController	Calling view controller

onCallEnd

Callback for ending a call, which is triggered for both callers and callees. If `enableCustomViewRoute` is set to `NO` , this callback will not be triggered.

```
- (void)callEnd:(NSArray<NSString *> *)userIDs type:(TUICallingType)type role:(TUICallingRole)rol
e totalTime:(CGFloat)totalTime;
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	NSArray	List of the user IDs of call participants
type	TUICallingType	Call type: audio/video
role	TUICallingRole	Role type: caller/callee
totalTime	CGFloat	Call duration (s)

onCallEvent

Call event callback, which is triggered for both callers and callees. If `enableCustomViewRoute` is set to `NO` , this callback will not be triggered.

```
- (void)onCallEvent:(TUICallingEvent)event type:(TUICallingType)type role:(TUICallingRole)role me
ssage:(NSString *)message;
```

The parameters are as detailed below:

Parameter	Type	Description
event	TUICallingEvent	Call event type
type	TUICallingType	Call type: audio/video
role	TUICallingRole	Role type: caller/callee
message	NSString	Event description

TUICalling APIs (Android)

Last updated：2022-03-17 15:49:52

`TRTCCalling` is an open-source class based on two closed-source SDKs: Tencent Cloud Real-Time Communication (TRTC) and Instant Messaging (IM). It supports one-to-one and group audio calls. For detailed instructions how to implement it, see Real-Time Audio Call (Android).

- TRTC SDK: The TRTC SDK is used as a low-latency audio/video call component.
- IM SDK: The IM SDK is used to send and process signaling messages.

`TUICalling` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton.
call	Sends call invitations by user ID.
receiveAPNSCalled	Answers a call.
setCallingListener	Sets the listener.
setCallingBell	Sets the ringtone (preferably shorter than 30s).
enableMuteMode	Enables/Disables the mute mode.
enableCustomViewRoute	Enables/Disables custom views.

`TUICallingListener` API Overview

Event callbacks

API	Description
shouldShowOnCallView	Callback of whether the answering view is displayed when there is an incoming call
onCallStart	Callback for starting calling. This callback is triggered for both callers and callees.

API	Description
onCallEnd	Callback for ending a call. This callback is triggered for both callers and callees.
onCallEvent	Call event callback

Type

Call type

Enumerated Type	Description
AUDIO	Audio call
VIDEO	Video call

Role

Role type

Enumerated Type	Description
CALL	Caller
CALLED	Callee

Event

Event type

Enumerated Type	Description
CALL_START	The call started.
CALL_SUCCEED	The call was connected successfully.
CALL_END	The call ended.
CALL_FAILED	The call failed.

Basic SDK APIs

sharedInstance

This API is used to get a singleton of the `TUICalling` component.

```
public static TUICallingManager sharedInstance();
```

call

This API is used to send call invitations by user ID.

```
void call(String[] userIDs, Type type);
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	String[]	List of the user IDs of call participants
type	TUICalling.Type	Call type: audio/video

receiveAPNSCalled

This API is used to answer a call.

```
void receiveAPNSCalled(String[] userIDs, Type type);
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	String[]	List of the user IDs of call participants
type	TUICalling.Type	Call type: audio/video

setCallingListener

This API is used to set the listener.

```
void setCallingListener(TUICallingListener listener);
```

The parameters are as detailed below:

Parameter	Type	Description
listener	TUICallingListener	Listener of the `TUIcalling` component

setCallingBell

This API is used to set the ringtone (preferably shorter than 30s).

```
void setCallingBell(String filePath);
```

The parameters are as detailed below:

Parameter	Type	Description
filePath	String	Path of the ringtone file

enableMuteMode

This API is used to enable/disable the mute mode.

```
void enableMuteMode(boolean enable);
```

The parameters are as detailed below:

Parameter	Type	Description
enable	boolean	Whether to enable the mute mode

enableCustomViewRoute

This API is used to enable/disable custom views.
After enabling custom views, you will receive a `CallingView` instance in the callback for calling/being called, and can decide how to display the view by yourself.

> Note :
> The view must be displayed full screen or in proportion to the screen size; otherwise, an error may occur.

```
void enableCustomViewRoute(boolean enable);
```

The parameters are as detailed below:

Parameter	Type	Description
enable	boolean	Whether to enable custom views

`TUICallingListener` Callback APIs

shouldShowOnCallView

Callback of whether the answering view is displayed when there is an incoming call.

```
boolean shouldShowOnCallView();
```

The parameters are as detailed below:

Parameter	Type	Description
Return value	boolean	Whether the answering view is displayed

onCallStart

Callback for starting calling, which is triggered for both callers and callees.

```
void onCallStart(String[] userIDs, TUICalling.Type type, TUICalling.Role role, View tuiCallingView);
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	String[]	List of the user IDs of call participants
type	TUICalling.Type	Call type: audio/video
role	TUICalling.Role	Role type: caller/callee
tuiCallingView	View	Calling view. When `enableCustomViewRoute` is `false` , `view` is null.

onCallEnd

Callback for ending a call, which is triggered for both callers and callees.

```
void onCallEnd(String[] userIDs, TUICalling.Type type, TUICalling.Role role, long totalTime);
```

The parameters are as detailed below:

Parameter	Type	Description
userIDs	String[]	List of the user IDs of call participants
type	TUICalling.Type	Call type: audio/video
role	TUICalling.Role	Role type: caller/callee
totalTime	long	Call duration (s)

onCallEvent

Call event callback.

```
void onCallEvent(TUICalling.Event event, TUICalling.Type type, TUICalling.Role role, String messa
ge);
```

The parameters are as detailed below:

Parameter	Type	Description
event	TUICalling.Event	Call event type
type	TUICalling.Type	Call type: audio/video
role	TUICalling.Role	Role type: caller/callee
message	String	Event description

TRTCCalling APIs (Web)

Last updated : 2022-04-06 14:07:47

`TRTCCalling` Overview

The TRTCCalling component is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). It supports one-to-one and group audio/video calls. For detailed instructions on how to implement it, see Real-Time Audio Call (Web).

- TRTC SDK: The TRTC SDK is used as a low-latency audio/video call component.
- IM SDK: The IM SDK is used to send and process signaling messages.

Demo Download

You can download the source code of the web demo at TUICalling.

Environment Requirements

We recommend you use Chrome for PC to run the demo as it offers better support for the features of the TRTC Web SDK. For more information on environment requirements, see Environment Requirements.

URL Protocol Support

Scenario	Protocol	Receive (Playback)	Send (Publish)	Share Screen	Remarks
Production	HTTPS	Supported	Supported	Supported	Recommended
Production	HTTP	Supported	Not supported	Not supported	-
Local development	http://localhost	Supported	Supported	Supported	Recommended
Local development	http://127.0.0.1	Supported	Supported	Supported	-

Scenario	Protocol	Receive (Playback)	Send (Publish)	Share Screen	Remarks
Local development	http://[local IP address]	Supported	Not supported	Not supported	-
Local development	file:///	Supported	Supported	Supported	-

`TRTCCalling` APIs

Event subscribing/unsubscribing APIs

This component bases its management on the dispatching of events. The application layer can change UI interactions according to dispatched events.

API	Description
on(eventName, callback, context)	Subscribes to an event.
off(eventName, callback, context)	Unsubscribes from an event.

Basic SDK APIs

API	Description
login({userID, userSig})	Logs in to IM. All IM features can be used only after login.
logout()	Logs out. No calls can be made after logout.

Call operation APIs

API	Description
call({userID, type, offlinePushInfo}))	Invites a user to a one-to-one call.
groupCall({userIDList, type, groupID, offlinePushInfo})	Invites users to a group call.
accept()	Accepts a call.
reject()	Rejects a call.
hangup()	Hangs up.

Video APIs

API	Description
startRemoteView({userID, videoViewDomID})	Starts rendering the video of a remote user.
stopRemoteView({userID})	Stops rendering the video of a remote user.
startLocalView({userID, videoViewDomID})	Starts rendering the video of the local user.
stopLocalView({userID})	Stops rendering the video of the local user.
openCamera()	Turns the camera on.
closeCamera()	Turns the camera off.
setMicMute(isMute)	Mutes/Unmutes the mic.
setVideoQuality(profile)	Sets video quality.
switchToAudioCall()	Switches to audio call.
switchToVideoCall()	Switches to video call.
getCameras()	Gets the camera list.
getMicrophones()	Gets the mic list.
switchDevice({deviceType, deviceId})	Switches to a different camera or mic.

API Details

Creating a `TRTCCalling` instance

First, create an application in the TRTC console and get the `SDKAppID` .
Then, obtain an instance of the `TRTCCalling` component using `new TRTCCalling()` .

```
npm i trtc-js-sdk --save
npm i tim-js-sdk --save
npm i tsignaling --save
npm i trtc-calling-js --save
// If you download the dependency using Node.js, you can import it using an import command.
import TRTCCalling from 'trtc-calling-js';
// If you use JavaScript, you need to manually import the following resources in the specified or
der.
// trtc.js
```

```
<script src="./trtc.js"></script>
// tim-js.js
<script src="./tim-js.js"></script>
// tsignaling.js
<script src="./tsignaling.js"></script>
// trtc-calling-js.js
<script src="./trtc-calling-js.js"></script>
let options = {
SDKAppID: 0, // Replace `0` with the `SDKAppID` of your IM application when connecting
// The `tim` parameter is added starting from v0.10.2
// The parameter guarantees the uniqueness of an existing TIM instance.
tim: tim
};
let trtcCalling = new TRTCCalling(options);
```

Event subscribing/unsubscribing APIs

on(eventName, callback, context)

This API is used to subscribe to an event dispatched by the component. For a list of the events, see `TRTCCalling` Events.

```
let handleInvite = function ({inviteID, sponsor, inviteData}) {
console.log(`inviteID: ${inviteID}, sponsor: ${sponsor}, inviteData: ${JSON.stringify(inviteDat
a)}`);
};
trtcCalling.on('onInvited', handleInvite, this);
```

off(eventName, callback, context)

This API is used to unsubscribe from an event.

```
let handleInvite = function ({inviteID, sponsor, inviteData}) {
console.log(`inviteID: ${inviteID}, sponsor: ${sponsor}, inviteData: ${JSON.stringify(inviteDat
a)}`);
};
trtcCalling.off('onInvited', handleInvite, this);
```

Basic SDK APIs

login({userID, userSig})

This API is used to log in.

```
trtcCalling.login({userID, userSig})
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_)
userSig	String	Tencent Cloud's proprietary security signature. For how to calculate and use it, see FAQs > UserSig.

logout()

This API is used to log out.

```
trtcCalling.logout()
```

Call operation APIs

call({userID, type, offlinePushInfo})

This API is used to make a one-to-one call. `type` indicates the call type (`1` : audio call; `2` : video call).

Note :

- The `timeout` parameter is deleted from v1.0.0 and later versions.
- A new parameter, `offlinePushInfo` , is introduced for offline notifications, **which are supported only on Android and iOS, not on web or WeChat Mini Program.**

```
// Versions earlier than v1.0.0
trtcCalling.call({userID, type, timeout})

// v1.0.0 and later versions
const offlinePushInfo = {
title: '',
description: 'You are invited to a call.',
}
trtcCalling.call({userID, type, offlinePushInfo})
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userID	String	User ID of the invitee
type	Number	`1` : audio call; `2` : video call
timeout	Number	Timeout period (s). `0` means the call never times out. **This parameter is valid only for versions earlier than v1.0.0.**
offlinePushInfo	Object	Offline notifications (optional). **This parameter is valid only for v1.0.0 and later versions.**

offlinePushInfo (in v1.0.0 and later versions)

Parameter	Type	Description
title	String	Title of an offline notification (optional)
description	String	Content of an offline notification (optional)
androidOPPOChannelID	String	Channel ID for an offline notification on OPPO 8.0 and above (optional)
extension	String	Passthrough content of an offline notification (optional), **which is valid only for TRTCCalling v1.0.2 or above and TSignaling v0.9.0 or above**

groupCall({userIDList, type, groupID, offlinePushInfo})

The `groupID` parameter is the group ID in the IM SDK. If this parameter is set, call invitations will be broadcast by the group messaging system, which is a simple and reliable way of sending call invitations. If this parameter is left empty, the `TRTCCalling` component will send an invitation to every invitee.

> Note :
> In v1.0.0 and later versions, a new parameter, `offlinePushInfo` , is introduced for offline notifications, **which are supported only on Android and iOS, not on web or WeChat Mini Program.**

```
// Versions earlier than v1.0.0
trtcCalling.groupCall({userIDList, type, groupID})
```

```
// v1.0.0 and later versions
const offlinePushInfo = {
title: ’’,
description: ’You are invited to a call.’,
}
trtcCalling.groupCall({userIDList, type, groupID, offlinePushInfo})
```

The parameters are as detailed below:

Parameter	Type	Description
userIDList	Array	List of the user IDs of invitees
type	Number	`1` : audio call; `2` : video call
groupID	String	ID of the IM group (optional)
offlinePushInfo	Object	Offline notifications (optional). **This parameter is valid only for v1.0.0 and later versions.**

offlinePushInfo (in v1.0.0 and later versions)

Parameter	Type	Description
title	String	Title of an offline notification (optional)
description	String	Content of an offline notification (optional)
androidOPPOChannelID	String	Channel ID for an offline notification on OPPO 8.0 and above (optional)
extension	String	Passthrough content of an offline notification (optional), **which is valid only for TRTCCalling v1.0.2 or above and TSignaling v0.9.0 or above**

accept()

This API is used to accept an invitation.

Note :

- If a prior invitation has not been processed, the component will return a message indicating that the line is busy.
- `params` is deleted from v1.0.0 and later versions.

```
import TRTCCalling from 'trtc-calling-js';
trtcCalling.on(TRTCCalling.EVENT.INVITED, ({inviteID, sponsor, inviteData}) => {
// ...
// Versions earlier than v1.0.0
const { roomID, callType } = inviteData;
trtcCalling.accept({inviteID, roomID, callType})
// v1.0.0 and later versions
trtcCalling.accept();
})
```

The parameters are as detailed below:

Parameter	Type	Description
inviteID	String	Invitation ID, which identifies an invitation and is returned by the `INVITED` callback. **This parameter is valid only for versions earlier than v1.0.0**.
roomID	Number	Call room ID, which is returned by the `INVITED` callback (in `inviteData`). **This parameter is valid only for versions earlier than v1.0.0**.
callType	Number	Call type, which is turned by the `INVITED` callback (in `inviteData`). `1` : audio call; `2` : video call. **This parameter is valid only for versions earlier than v1.0.0**.

reject()

This API is used to reject an invitation.

> Note :
> `params` is deleted from v1.0.0 and later versions.

```
import TRTCCalling from 'trtc-calling-js';
trtcCalling.on(TRTCCalling.EVENT.INVITED, ({inviteID, sponsor, inviteData}) => {
// ...
// Versions earlier than v1.0.0
const { callType } = inviteData;
trtcCalling.reject({inviteID, isBusy, callType})
// v1.0.0 and later versions
trtcCalling.reject();
})
```

The parameters are as detailed below:

Parameter	Type	Description
inviteID	String	Invitation ID, which identifies an invitation and is returned by the `INVITED` callback. **This parameter is valid only for versions earlier than v1.0.0**.
isBusy	Boolean	Whether the line is busy. **This parameter is valid only for versions earlier than v1.0.0.**
callType	Number	Call type, which is turned by the `INVITED` callback (in `inviteData`). `1` : audio call; `2` : video call. **This parameter is valid only for versions earlier than v1.0.0**.

hangup()

1. If you are in a call, you can use this API to end the call.
2. If your call is not answered yet, you can use this API to cancel the call.

```
trtcCalling.hangup()
```

Video APIs

startRemoteView({userID, videoViewDomID})

This API is used to render the camera data of a remote user in a specified DOM node.

```
trtcCalling.startRemoteView({userID, videoViewDomID})
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
videoViewDomID	String	The DOM node in which the user’s data is to be rendered. The data will be played via the video tag of the node.

stopRemoteView({userID})

This API is used to delete the DOM node in which the camera data of a remote user is rendered.

Note :
`videoViewDomID` is deleted from v1.0.0 and later versions.

```
// Versions earlier than v1.0.0
trtcCalling.stopRemoteView({userID, videoViewDomID});
// v1.0.0 and later versions
trtcCalling.stopRemoteView({userID});
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
videoViewDomID	String	The DOM node whose video tag is to be deleted. The playback will stop. **This parameter is valid only for versions earlier than v1.0.0.**

startLocalView({userID, videoViewDomID})

This API is used to render the camera data of the local user in a specified DOM node.

```
trtcCalling.startLocalView({userID, videoViewDomID})
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
videoViewDomID	String	The DOM node in which the local user's data is to be rendered. The data will be played via the video tag of the node.

stopLocalView({userID})

This API is used to delete the DOM node in which the camera data of the local user is rendered.

Note :
`videoViewDomID` is deleted from v1.0.0 and later versions.

```
// Versions earlier than v1.0.0
trtcCalling.stopLocalView({userID, videoViewDomID});
// v1.0.0 and later versions
trtcCalling.stopLocalView({userID});
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
videoViewDomID	String	The DOM node whose video tag is to be deleted. The playback will stop. **This parameter is valid only for versions earlier than v1.0.0.**

openCamera()

This API is used to turn the local camera on.

```
trtcCalling.openCamera()
```

closeCamera()

This API is used to turn the camera off.

```
trtcCalling.closeCamera()
```

setMicMute(isMute)

This API is used to turn the mic on/off.

```
trtcCalling.setMicMute(true) // Turn the mic off
```

The parameters are as detailed below:

Parameter	Type	Description
isMute	Boolean	• `true` : turn the mic off • `false` : turn the mic on

setVideoQuality(profile)

This API is used to set video quality.

Note :

- This is a new API in v0.8.0 and later versions.
- This API must be called before `call` , `groupCall` , or `accept` to take effect.

```
trtcCalling.setVideoQuality('720p') // Set video quality to `720p`
```

The parameters are as detailed below:

Parameter	Type	Description
profile	String	- `480p` : 640 × 480 - `720p` : 1280 × 720 - `1080p` : 1920 × 1080

switchToAudioCall()

This API is used to switch from video call to audio call.

Note :

- This is a new API in v0.10.0 and later versions.
- It can be used only in one-to-one calls.
- If you receive an error callback (code: 60001), the switching failed.

```
trtcCalling.switchToAudioCall() // Switch from video call to audio call
```

switchToVideoCall()

This API is used to switch from audio call to video call.

Note :

- This is a new API in v0.10.0 and later versions.
- It can be used only in one-to-one calls.
- If you receive an error callback (code: 60002), the switching failed.

```
trtcCalling.switchToVideoCall() // Switch from audio call to video call
```

getCameras()

This API is used to get the camera list.

> Note：
> This is a new API in v1.0.0 and later versions.

```
trtcCalling.getCameras() // Get the camera list
```

getMicrophones()

This API is used to get the mic list.

> Note：
> This is a new API in v1.0.0 and later versions.

```
trtcCalling.getMicrophones() // Get the mic list
```

switchDevice({deviceType, deviceId})

This API is used to switch to a different camera or mic.

> Note：
> This is a new API in v1.0.0 and later versions.

```
trtcCalling.switchDevice({deviceType: 'audio', deviceId: deviceId}) // Switch to a different devi
ce
```

The parameters are as detailed below:

Parameter	Type	Description
deviceType	String	Device type. `video` : camera; `audio` : mic

Parameter	Type	Description
deviceId	String	Device ID. • You can use `getCameras()` to get the ID of a camera. • You can use `getMicrophones()` to get the ID of a mic.

`TRTCCalling` Events

The code below demonstrates how to listen for TRTCCalling events.

```
import TRTCCalling from 'trtc-calling-js';
// etc
function handleInviteeReject({userID}) {

}
trtcCalling.on(TRTCCalling.EVENT.REJECT, handleInviteeReject)
```

Invitation events

Code	Event Recipient	Description
REJECT	Inviter	The invitee rejected the call.
NO_RESP	Inviter	The invitation timed out without response from the invitee.
LINE_BUSY	Inviter	The invitee is in a call, i.e., the line is busy.
INVITED	Invitee	An invitation was received.
CALLING_CANCEL	Invitee	The call is canceled.
CALLING_TIMEOUT	Invitee	The invitation timed out.
USER_ENTER	Inviter and invitee	A user entered the room.
USER_LEAVE	Inviter and invitee	A user left the call.
CALL_END	Inviter and invitee	The call ended.

Code	Event Recipient	Description
KICKED_OUT	Inviter and invitee	A user was kicked out due to repeated login.
USER_VIDEO_AVAILABLE	Inviter and invitee	A remote user turned the camera on/off.
USER_AUDIO_AVAILABLE	Inviter and invitee	A remote user turned the mic on/off.

Common event callbacks

SDK_READY

The SDK is ready.

> Note :
> This is a new event callback in v1.0.0 and later versions.

```
let onSDKReady = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.SDK_READY, onSDKReady);
```

USER_ENTER

A user entered the room.

This event callback is triggered when a user joins the call.

```
let handleUserEnter = function({userID}) {
console.log(userID)
};
trtcCalling.on(TRTCCalling.EVENT.USER_ENTER, handleUserEnter);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID

USER_LEAVE

A user left the room.

This event callback is triggered when a user leaves the call.

```
let handleUserLeave = function({userID}) {
console.log(userID)
};
trtcCalling.on(TRTCCalling.EVENT.USER_LEAVE, handleUserLeave);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID

GROUP_CALL_INVITEE_LIST_UPDATE

The invitee list for a group call was updated.

> Note :
>
> This is a new event callback in v1.0.0 and later versions.

```
let handleGroupInviteeListUpdate = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.GROUP_CALL_INVITEE_LIST_UPDATE, handleGroupInviteeListUpdate);
```

CALL_END

The call ended.

This event callback is triggered when a call ends.

```
let handleCallingEnd = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.CALL_END, handleCallingEnd);
```

KICKED_OUT

A user was kicked out due to repeated login.

This event callback is triggered if a user logs in with the same account on another page.

```
let handleKickedOut = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.KICKED_OUT, handleKickedOut);
```

USER_VIDEO_AVAILABLE

A remote user turned the camera on/off.

This event callback is triggered when a remote user turns the camera on/off.

```
let handleUserVideoChange = function({userID, isVideoAvailable}) {
console.log(userID, isVideoAvailable)
};
trtcCalling.on(TRTCCalling.EVENT.USER_VIDEO_AVAILABLE, handleUserVideoChange);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
isVideoAvailable	Boolean	• `true` : The remote user turned the camera on. • `false` : The remote user turned the camera off.

USER_AUDIO_AVAILABLE

A remote user turned the mic on/off.

This event callback is triggered when a remote user turns the mic on/off.

```
let handleUserAudioChange = function({userID, isAudioAvailable}) {
console.log(userID, isAudioAvailable)
};
trtcCalling.on(TRTCCalling.EVENT.USER_AUDIO_AVAILABLE, handleUserAudioChange);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
isAudioAvailable	Boolean	• `true` : The remote user turned the mic on. • `false` : The remote user turned the mic off.

Event callbacks received by inviter

REJECT

The user rejected the call.

The inviter of a call will receive this callback if an invitee rejects the call.

```
let handleInviteeReject = function({userID}) {
console.log(userID)
};
trtcCalling.on(TRTCCalling.EVENT.REJECT, handleInviteeReject);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID

NO_RESP

The invitee did not answer.

In cases where a timeout period is specified for a one-to-one or group call, the inviter will receive this callback if an invitee does not answer the call within the specified timeout period.

```
let handleNoResponse = function({userID, userIDList}) {
console.log(userID, userIDList)
};
trtcCalling.on(TRTCCalling.EVENT.NO_RESP, handleNoResponse);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
userIDList	Array	List of timed out users

LINE_BUSY

The invitee is in a call, i.e., the line is busy.

The inviter of a call will receive this callback if the invitee is already in a call.

```
let handleLineBusy = function({userID}) {
console.log(userID)
};
trtcCalling.on(TRTCCalling.EVENT.LINE_BUSY, handleLineBusy);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID

Event callbacks received by invitee

INVITED

An invitation was received.

A user will receive this callback if he or she is invited to a call.

```
let handleNewInvitationReceived = function({
sponsor, userIDList, isFromGroup, inviteData, inviteID
}) {
console.log(sponsor, userIDList, isFromGroup, inviteData, inviteID)
};
trtcCalling.on(TRTCCalling.EVENT.INVITED, handleNewInvitationReceived);
```

The parameters are as detailed below:

Parameter	Type	Description
sponsor	String	Inviter
userIDList	Array	Users invited to the same call
isFromGroup	Boolean	Whether it is an IM group invitation
inviteData	Object	For a new user invitation: {version, callType, roomID}
inviteID	String	Invitation ID, which identifies an invitation

CALLING_CANCEL

The call is canceled.

An invitee of a call will receive this callback if the call is canceled.

```
let handleCallingCancel = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.CALLING_CANCEL, handleCallingCancel);
```

CALLING_TIMEOUT

The call timed out.
In cases where a timeout period is specified for a one-to-one or group call, an invitee will receive this callback if he or she does not answer the call within the specified timeout period.

```
let handleCallingTimeout = function(event) {
console.log(event)
};
trtcCalling.on(TRTCCalling.EVENT.CALLING_TIMEOUT, handleCallingTimeout);
```

`TRTCCalling` Error Codes

You can register the error callback, as shown below, to handle the errors thrown by the component.

```
import TRTCCalling from 'trtc-calling-js';
let onError = function(error) {
console.log(error);
};
trtcCalling.on(TRTCCalling.EVENT.ERROR, onError);
```

Error codes

Code	Type	Description
60001	API call failure	Failed to call `switchToAudioCall` .
60002	API call failure	Failed to call `switchToVideoCall` .
60003	Access failure	No available mic.
60004	Access failure	No available camera.
60005	Access failure	The user denied access.
60006	Failure to pass environment check	The current environment does not support WebRTC (v1.0.4 or above is required).

Update Guide

- **Updating to TRTCCalling 1.0.2 or above**
  - You need to update TSignaling to v0.9.0 or above.

  - Reason: See Changelog.
- **Updating to TRTCCalling 1.0.0 or 1.0.1**
  - You need to update TSignaling to v0.8.0 or above.
  - Reason: See Changelog.

FAQs

Why can’t I get through to a user? Why am I kicked offline?

The `TRTCCalling` component does not support login of multiple instances or **offline signaling** for the time being. Please make sure that your current login is unique.

> Note :
>
> - **Multiple instances**: A user logs in with the same account multiple times or on different devices, which disrupts signaling.
> - **Offline signaling**: Only online instances can receive messages. Messages sent to offline instances will not be sent again when the instances go online.
>   For FAQs about TRTCCalling for web, see TRTCCalling for Web.

Technical Support

If you have other questions, you can fill out a contact form or email colleenyu@tencent.com.

Learn More

- TRTCCalling web demo
- TRTCCalling for npm
- Source code of TRTCCalling web demo
- TRTCCalling web APIs
- FAQs

TRTCCalling APIs (Flutter)

Last updated : 2022-03-17 15:50:50

`TRTCCalling` is an open-source class based on two closed-source SDKs: Tencent Cloud Real-Time Communication (TRTC) and Instant Messaging (IM). It supports one-to-one audio/video calls. For detailed instructions how to implement it, please see Real-Time Audio Call (Flutter).

- TRTC SDK: the TRTC SDK is used as a low-latency audio/video call component.
- IM SDK: the IM SDK is used to send and process signaling messages.

`TRTCCalling` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton.
destroySharedInstance	Destroys a singleton.
registerListener	Registers an event listener.
unRegisterListener	Unregisters an event listener.
destroy	Destroys an instance that is no longer needed.
login	Logs in. All features can be used only after login.
logout	Logs out. No calls can be made after logout.

Call operation APIs

API	Description
call	Makes a one-to-one call.
accept	Accepts the current call.
reject	Declines the current call.
hangup	Ends the current call.

Audio control APIs

API	Description
setMicMute	Mutes the local mic.
setHandsFree	Enables the hands-free mode.

`TRTCCallingDelegate` API Overview

Common event callback APIs

API	Description
onError	Callback for error.

Inviter callback APIs

API	Description
onReject	The call was declined.
onNoResp	The invitee did not answer.
onLineBusy	The line is busy.

Invitee callback APIs

API	Description
onInvited	An invitation was received.
onCallingCancel	The call was canceled.
onCallingTimeOut	The call timed out.

General callback APIs

API	Description
onUserEnter	A user joined the call.
onUserLeave	A user left the call.
onUserAudioAvailable	Whether a user is sending audio

API	Description
onUserVoiceVolume	Call volume of the user
onCallEnd	The call ended.

Basic SDK APIs

sharedInstance

This API is used to get a singleton of the `TRTCCalling` component.

```
static Future<TRTCCalling> sharedInstance();
```

destroySharedInstance

This API is used to destroy a singleton of the `TRTCCalling` component.

```
static void destroySharedInstance();
```

destroy

This API is used to destroy an instance that is no longer needed.

```
void destroy();
```

registerListener

This API is used to register callbacks for TRTCCalling events. You can receive status notifications of TRTCCalling via `TRTCCallingDelegate` .

```
void registerListener(VoiceListenerFunc func);
```

unRegisterListener

This API is used to unregister event callbacks.

```
void unRegisterListener(VoiceListenerFunc func);
```

login

This API is used to log in to the component.

```
Future<ActionCallback> login(int sdkAppId, String userId, String userSig);
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppID	int	You can view the `SDKAppID` via **Application Management** > **Application Info** in the TRTC console.
userId	String	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 protection signature. For more information on how to get it, please see How to Calculate UserSig.

logout

This API is used to log out of the component.

```
Future<ActionCallback> logout();
```

Call Operation APIs

call

This API is used to make a one-to-one call. It can be called during a call to invite more users.

```
Future<ActionCallback> call(String userId, int type);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the callee
type	int	`1` : audio call; `2` : video call

accept

This API is used to accept the current call. After receiving the `onInvited()` callback, the invitee can call this API to accept the call.

```
Future<ActionCallback> accept();
```

reject

This API is used to decline the current call. After receiving the `onInvited()` callback, the invitee can call this API to decline the call.

```
Future<ActionCallback> reject();
```

hangup

This API is used to end the current call.

```
void hangup();
```

Audio Control APIs

setMicMute

This API is used to mute the local mic.

```
void setMicMute(bool isMute);
```

The parameters are as detailed below:

Parameter	Type	Description
isMute	bool	`true` : mutes the mic; `false` : unmutes the mic.

setHandsFree

This API is used to enable the hands-free mode.

```
void setHandsFree(bool isHandsFree);
```

The parameters are as detailed below:

Parameter	Type	Description
isHandsFree	bool	`true` : enables the hands-free mode; `false` : disables the hands-free mode.

`TRTCCallingDelegate` Callback APIs

Common Event Callback APIs

onError

Callback for error.

> 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.

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
msg	String	Error message

Inviter Callback APIs

onReject

The call was declined.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the invitee who declined the call

onNoResp

The invitee did not answer.

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	String	User ID of the invitee who did not answer

onLineBusy

The line is busy.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the invitee whose line is busy

Invitee Callback APIs

onInvited

A call invitation was received.

The parameters are as detailed below:

Parameter	Type	Description
sponsor	String	User ID of the inviter
userIds	List<String>	IDs of other users invited
isFromGroup	bool	Whether it is a group call
type	int	`1` : audio call; `2` : video call

onCallingCancel

The call was canceled. The invitee will receive this callback if the inviter cancels the invitation before he or she handles it.

onCallingTimeOut

The call timed out.

General Callback APIs

onUserEnter

A user joined the call.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user who joined the call

onUserLeave

A user left the call.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user who left the call

onUserAudioAvailable

Whether a user is sending audio.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
available	boolean	Whether the user has available audio

onUserVoiceVolume

Call volume of a user.

The parameters are as detailed below:

Parameter	Type	Description
userVolumes	List	Volume of every speaking user in the room. Value range: 0-100.
totalVolume	int	Total volume of all remote users. Value range: 0-100.

onCallEnd

The call ended.

Interactive Video Streaming

Interactive Video Streaming (iOS)

Last updated：2022-03-31 10:47:54

Demo UI

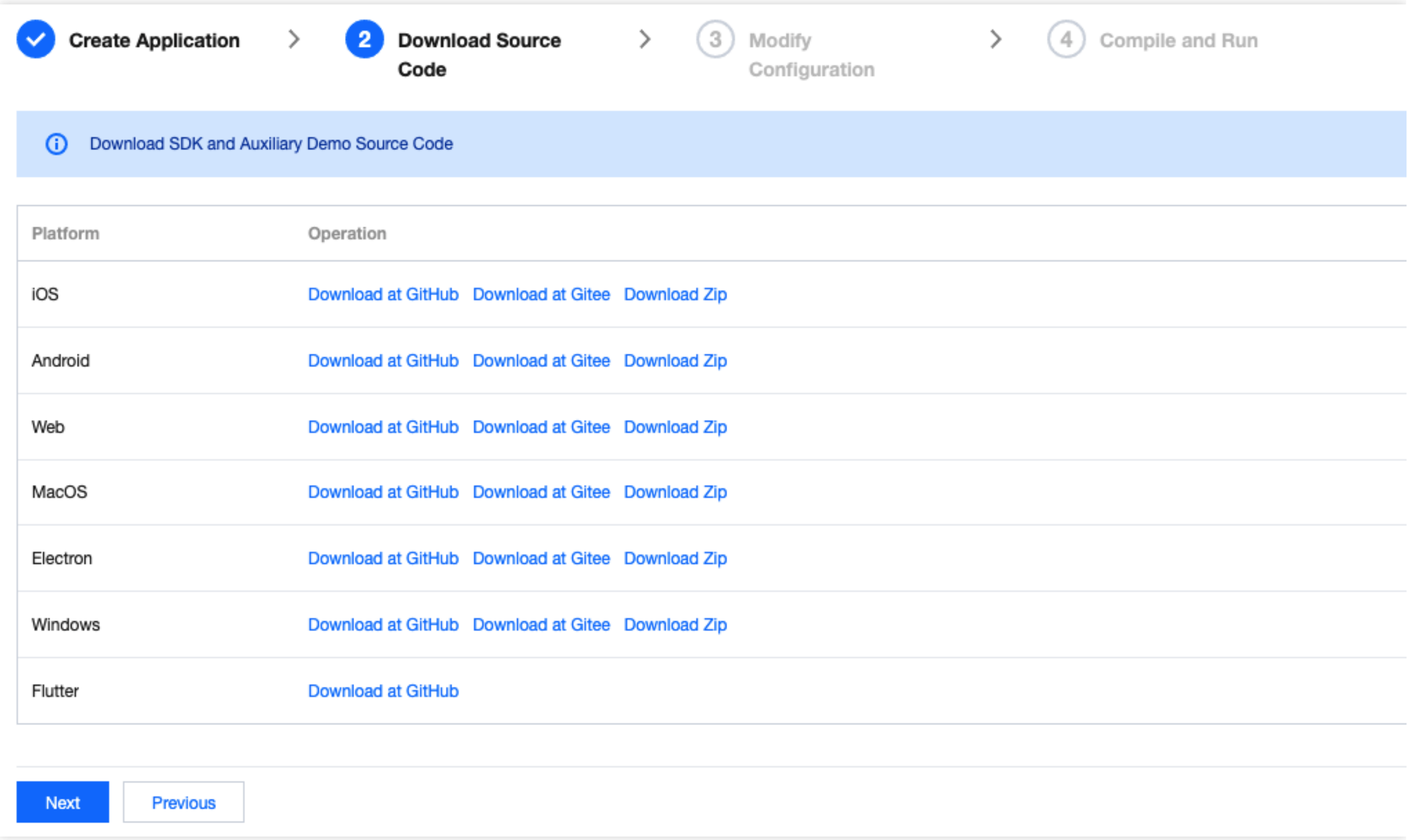
You can download and install the demo app we provide to try out TRTC’s interactive live streaming features, including co-anchoring, anchor competition, low-latency watch, and on-screen comments.

To quickly enable the interactive live video streaming feature, you can modify the demo app we provide and adapt it to your needs. You can also use the `TUILiveRoom` component and customize your own UI.

Using the App’s UI

Step 1. Create an application

1. In the TRTC console, select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name, e.g. `TestLiveRoom` , and click **Create**.
3. Click **Next** to skip this step.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Note :

This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the application source code

Clone or download the TUILiveRoom source code.

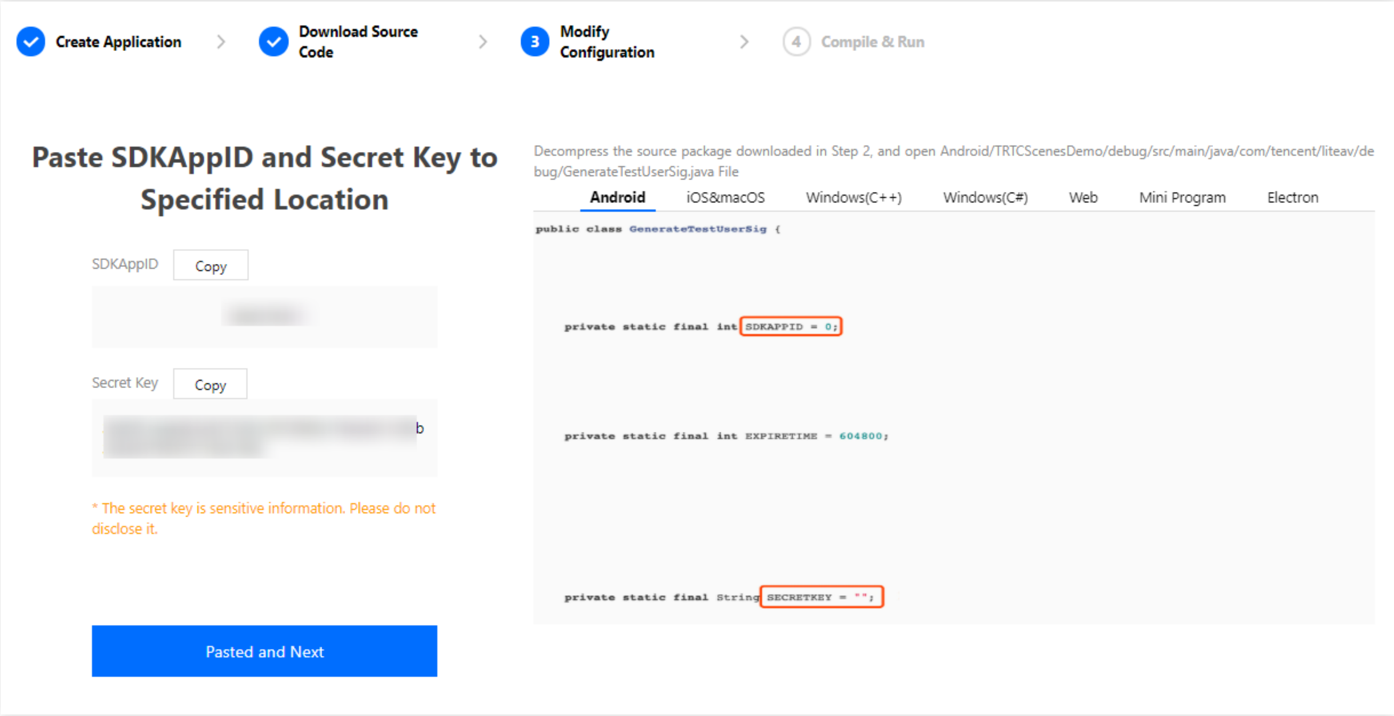
Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open the `TUILiveRoom/Debug/GenerateTestUserSig.swift` file.

3. Set the following parameters in `GenerateTestUserSig.swift` :

   - SDKAPPID: `0` by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: left empty by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Pasted and Next

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUILiveRoom/TUILiveRoomApp.xcworkspace` with Xcode (version 11.0 or above) and click **Run**.

Step 5. Modify the app’s source code

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `ui` folder contains the UI code. The table below lists the Swift files (folders) and UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
Anchor	Implementation code for anchor-end views
Audience	Implementation code for viewer-end views
ChatRoom	Implementation code for the text chat room and on-screen comment views
Common	Implementation code for some reusable UI components
StatusView	Floating status view, which floats over video images and displays log information and video loading animations
LiveRoomMainViewController.swift	The main view for interactive live video streaming

Tryout

> Note :
> You need at least two devices to try out the application.

User A

1. Enter a username (**which must be unique**) and log in.
2. Click **Create Room**.
3. Enter a room name and tap **Start**.

User B

1. Enter a username (**which must be unique**) and log in.
2. Enter the ID of the room created by user A, and tap **Join**.

> Note :
> The room ID can be obtained from the system pop-up window after user A's room is created.

Listening for Room Status and Getting Room List for Co-Anchoring

You can use `TRTCLiveRoom` to listen for room status.

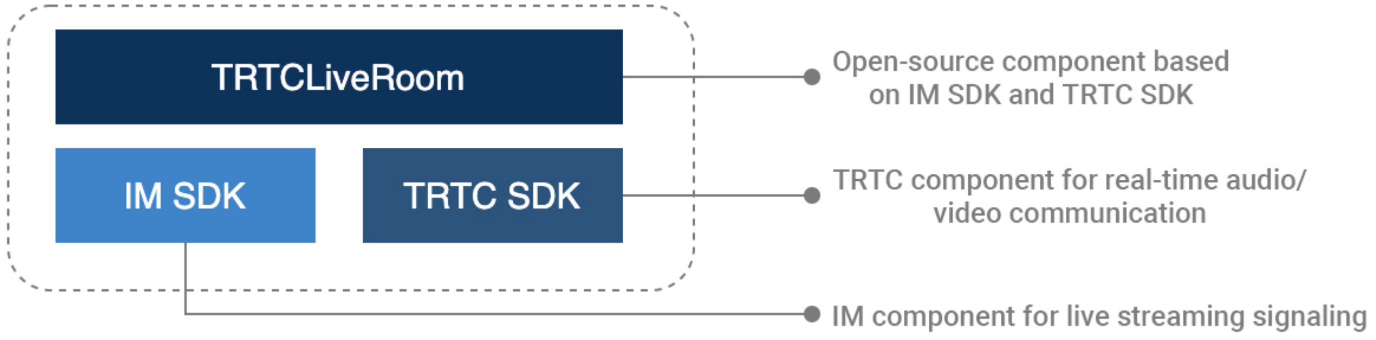
```
/////////////////////////////////////////////////////////////
//
// Room management
//
/////////////////////////////////////////////////////////////
/// Create a room (called by anchor.) If the room does not exist, the system will create the room automatically.
/// The process of creating a room and starting live streaming as an anchor is as follows:
/// 1. A user calls `startCameraPreview()` to enable camera preview and set beauty filters.
/// 2. The user calls `createRoom()` to create a room, the result of which is returned via a callback.
/// 3. The user calls `starPublish()` to push streams.
/// - Parameters:
/// - roomID: room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a live room list. Currently, Tencent Cloud does not provide list management services. Please manage the list on your own.
/// - roomParam: room information, such as room name and cover information. If both the room list and room information are managed by yourself, you can ignore this parameter.
/// - callback: callback for room entry. The code is `0` if room entry is successful.
/// - Note:
/// - This API is called by an anchor to start live streaming. An anchor can create a room he or she created before.
- (void)createRoomWithRoomID:(UInt32)roomID
roomParam:(TRTCCreateRoomParam *)roomParam
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
/// Terminate a room (called by anchor)
/// After creating a room, an anchor can call this API to terminate it.
/// - parameter callback: callback for room termination. The code is `0` if the operation is successful.
/// - Note:
/// - After creating a room, an anchor can call this API to terminate it
- (void)destroyRoom:(Callback _Nullable)callback
NS_SWIFT_NAME(destroyRoom(callback:));
/// Get room details
/// The information is provided by anchors via the `roomInfo` parameter when they call `createRoom()`. You don’t need this API if both the room list and room information are managed by yourself.
/// - Parameter roomIDs: list of room IDs
/// - Parameter callback: callback of room details
```

```
- (void)getRoomInfosWithRoomIDs:(NSArray<nsnumber *="">> *)roomIDs
callback:(RoomInfoCallback _Nullable)callback
NS_SWIFT_NAME(getRoomInfos(roomIDs:callback:));
```

Getting the room list may involve async operations. Using callbacks to get the list provides greater convenience and flexibility.

Customizing UI

The `Source` folder in the source code contains two subfolders: `ui` and `model` . The `model` subfolder contains the reusable open-source component `TRTCLiveRoom` . You can find the component's APIs in `TRTCLiveRoom.h` and use them to customize your own UI.


TRTCLiveRoom
IM SDK
TRTC SDK
Open-source component based on IM SDK and TRTC SDK
TRTC component for real-time audio/ video communication
IM component for live streaming signaling

Step 1. Integrate the SDKs

The `TRTCLiveRoom` component depends on the TRTC SDK and IM SDK. Follow the steps below to integrate them into your project.

- **Method 1: adding dependencies via CocoaPods**

```
pod 'TXIMSDK_iOS'
pod 'TXLiteAVSDK_TRTC'
```

- **Method 2: adding dependencies through local files**

  If your access to the CocoaPods repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration document

IM SDK	Download	Integration document

Step 2. Configure permission requests

Configure camera and mic permission requests by adding `Privacy > Camera Usage Description` and `Privacy > Microphone Usage Description` in `info.plist` .

Step 3. Import the `TUILiveRoom` component

To import the component through CocoaPods, follow the steps below:

1. Copy the `Source` , `Resources` , `TCBeautyKit` , and `TXAppBasic` folders and the `TUILiveRoom.podspec` files in the demo directory to your project directory.
2. Add the following dependencies to your `Podfile` and run `pod install` to complete the import.

```
pod ’TXAppBasic’, :path =&gt; ”TXAppBasic/”
pod ’TCBeautyKit’, :path =&gt; ”TCBeautyKit/”
pod ’TXLiteAVSDK_TRTC’
pod ’TUILiveRoom’, :path =&gt; ”./”, :subspecs =&gt; [”TRTC”]
```

Step 4. Create an instance and log in

1. Call the `init` API of `TRTCLiveRoom` to create an instance of the `TRTCLiveRoom` component.
2. Create a `TRTCLiveRoomConfig` object, and set its `useCDNFirst` and `CDNPlayDomain` attributes.

- `useCDNFirst` : specifies the way audience watch live streams. `true` means that audience watch live streams over CDNs, which is cost-efficient but has high latency. `false` means that audience watch live streams in the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is within 1 second.
- `CDNPlayDomain` : specifies the domain name for CDN live streaming, which takes effect only if `useCDNFirst` is set to `true` . You can set it in CSS console > **Domain Management**.

3. Call the `login` API to log in to the component, and set the key parameters as described below.

Parameter	Description
SDKAppID	You can view \`SDKAppID\` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.

config	Global configuration information. Please initialize it during login as it cannot be modified after login. ◦ `useCDNFirst` : specifies the way audience watch live streams. `true` means that audience watch live streams over CDNs, which is cost-efficient but has high latency. `false` means that audience watch live streams in the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is within 1 second. ◦ `CDNPlayDomain` : specifies the domain name for CDN live streaming, which takes effect only if `useCDNFirst` is set to `true` . You can set it in CSS console > **Domain Management**.
callback	Login callback. The code is \`0\` if login is successful.

```
class LiveRoomController: UIViewController {
let mLiveRoom = TRTCLiveRoom()
}
// useCDNFirst: `true` means that audience watch live streams over CDNs, and `false` means that audience watch live streams in the low latency mode.
//CDNPlayDomain: the playback domain name for CDN live streaming
let config = TRTCLiveRoomConfig(useCDNFirst: useCDNFirst, cdnPlayDomain: yourCDNPlayDomain)
mLiveRoom.login(SDKAPPID, userID, userSig, config) { (code, error) in
if code == 0 {
// Logged in
}
}
```

Step 5. Create a room and push streams

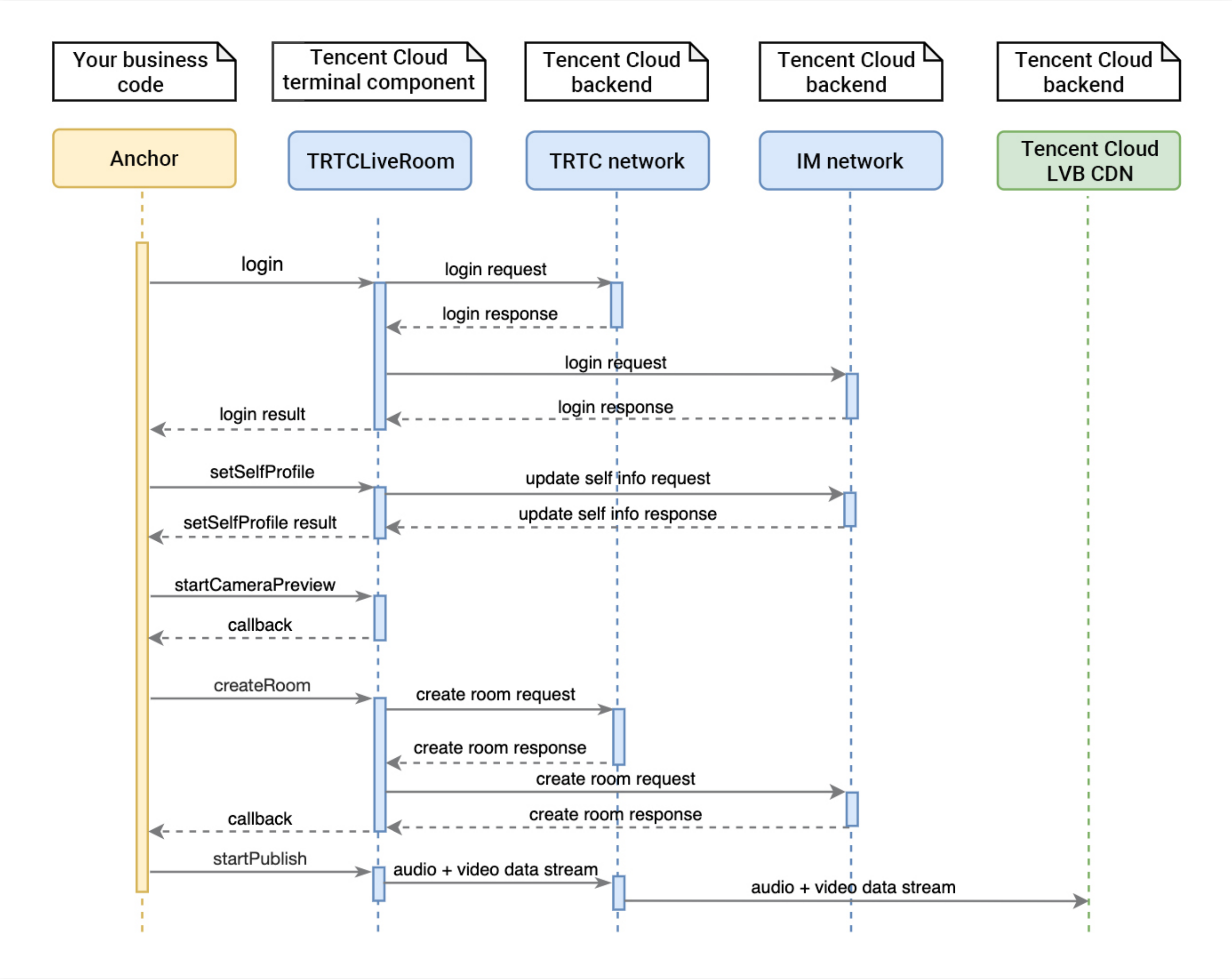
i. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.

ii. Before streaming, you can call `startCameraPreview` to enable camera preview, add beauty filter buttons to the UI, and set beauty filters through `getBeautyManager` .

> Note :
> Only the Enterprise Edition SDK supports advanced beauty filter features such as face changing and stickers.

iii. After setting beauty filters, call `createRoom` to create a live streaming room.

iv. Call `startPublish` to start streaming. To enable CDN live streaming, specify `useCDNFirst` and `CDNPlayDomain` in the `TRTCLiveRoomConfig` parameter, which is passed in during login, and specify `streamID` for playback when calling `startPublish` .



```
// 1. Set your username and profile photo as an anchor
mLiveRoom.setSelfProfile(name: "A", avatarURL: "faceUrl", callback: nil)
// 2. Enable camera preview and set beauty filters before streaming
let view = UIView()
parentView.add(view)
mLiveRoom.startCameraPreview(frontCamera: true, view: view, callback: nil)
mLiveRoom.getBeautyManager().setBeautyStyle(.nature)
mLiveRoom.getBeautyManager().setBeautyLevel(6)
// 3. Create a room
let param = TRTCCreateRoomParam(roomName: "Test room", coverUrl: "")
mLiveRoom.createRoom(roomID: 123456789, roomParam: param) { [weak self] (code, error) in
if code == 0 {
// 4. Start streaming and publish the streams to CDNs
self?.mLiveRoom.startPublish(streamID: mSelfUserId + "_stream", callback: nil)
}
}
```

Step 6. Play back streams

i. After performing step 4 to log in, you can call `setSelfProfile` to set your nickname and profile photo.

ii. Get the latest room list from the backend.

> Note :
> The room list in the demo app is for demonstration only. The business logic of live streaming room lists varies significantly. Tencent Cloud does not provide list management services for the time being. Please manage the list by yourself.

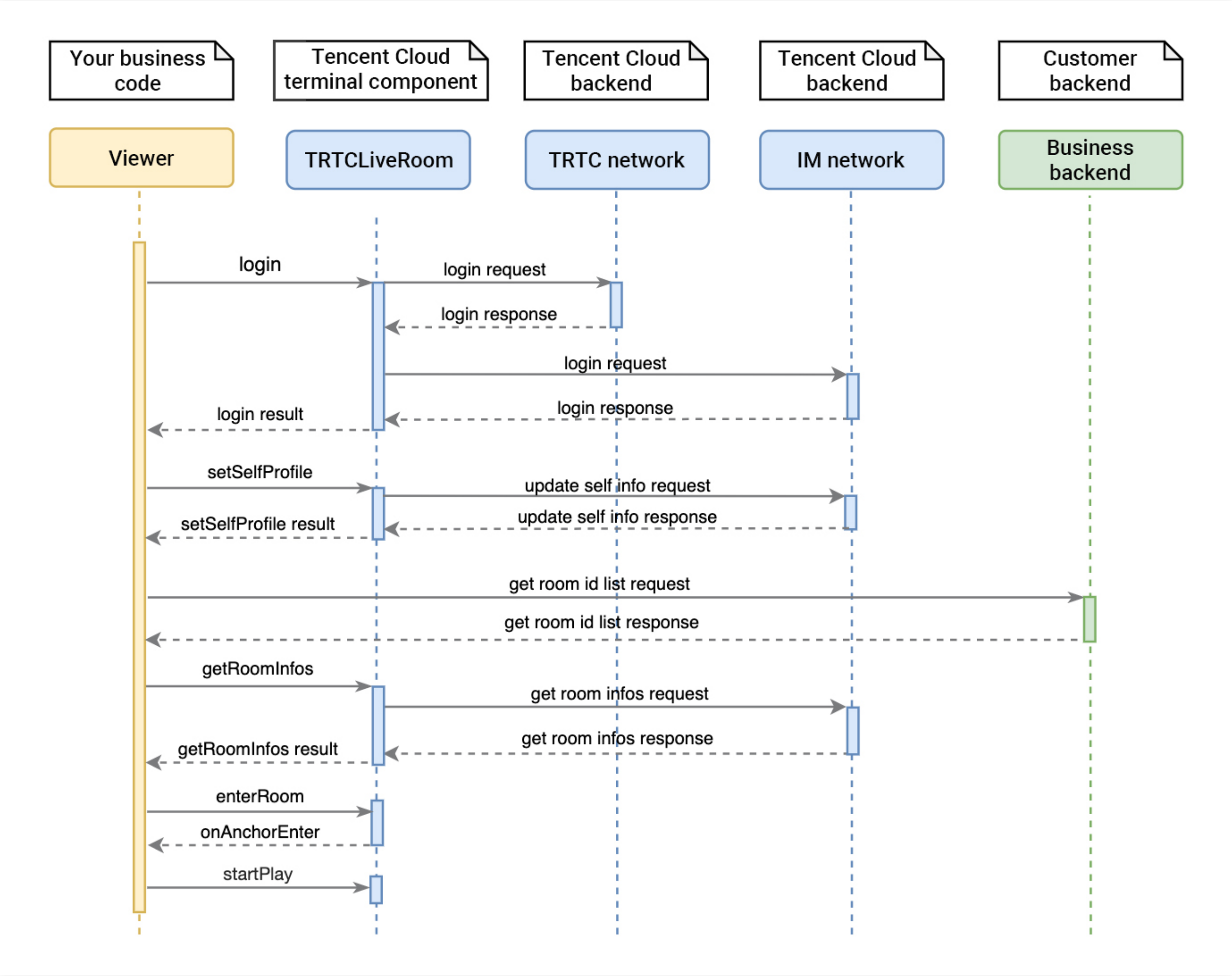
iii. Call `getRoomInfos` to get short descriptions of the rooms, which are provided by anchors when they call `createRoom` to create the rooms.

> Note :
> If your room list already displays enough information, you can skip the step of calling `getRoomInfos` .

iv. Select a room, call `enterRoom` , with the room ID passed in to enter the room.

v. Call `startPlay` , with the anchor's `userId` passed in to start playback.

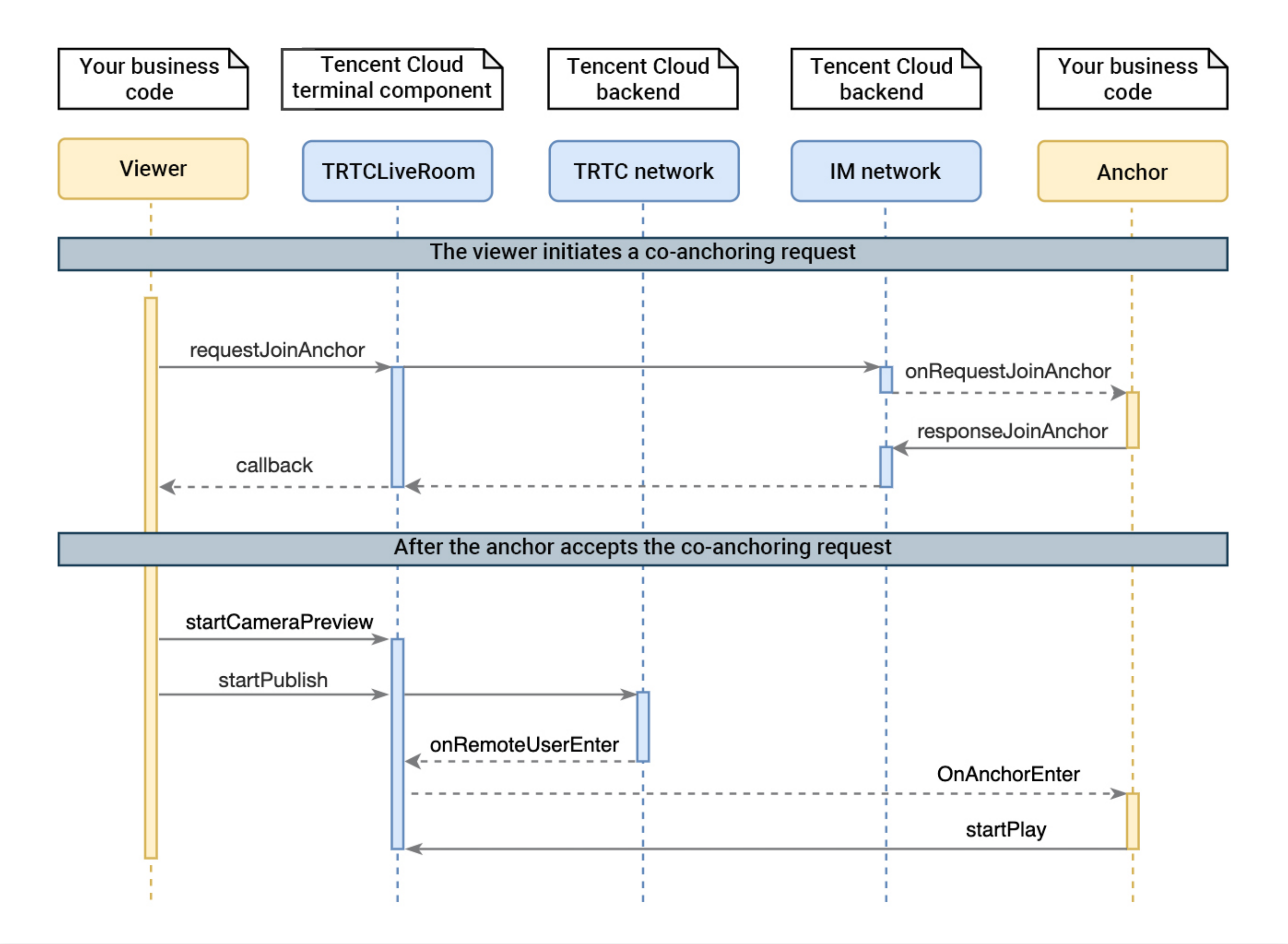
- If the room list displays the `userId` of the anchor, you can call `startPlay` , passing in the anchor’s `userId` to start playback.
- If you do not know the anchor's `userId` , you can find it in the `onAnchorEnter` event callback, which you will receive after entering the room. You can then call `startPlay` to start playback.



```
// 1. Get the room list from the backend. Suppose it is `roomList`
var roomList: [UInt32] = GetRoomList()
// 2. Call `getRoomInfos` to get the details of the room
mliveRoom.getRoomInfos(roomIDs: roomList, callback: { (code, msg, list) in
if code == 0 {
// After getting the room information, you can display on the anchor list page the anchor’s ni
ckname, profile photo, and other information
}
})
// 3. Select a `roomid` and enter the room
mliveRoom.enterRoom(roomID: roomID, callback: callback)
// 4. After receiving the notification about the anchor’ s entry, start playback
public func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {
// 5. Play the anchor’s video
mliveRoom.startPlay(userID: userID, view: renderView, callback: nil)
}
```

Step 7. Co-anchor

i. Audience calls `requestJoinAnchor` to send a co-anchoring request to the anchor.
ii. The anchor receives a notification (`TRTCLiveRoomDelegate#onRequestJoinAnchor`) that a viewer requested to co-anchor.
iii. The anchor calls `responseJoinAnchor` to accept or decline the co-anchoring request.
iv. The audience receives the `TRTCLiveRoomDelegate#responseCallback` event notification, which carries the anchor’s response.
v. If the anchor accepts the co-anchoring request, the audience can call `startCameraPreview` to turn the local camera on and then `startPublish` to push streams.
vi. The anchor receives a notification (`TRTCLiveRoomDelegate#onAnchorEnter`) that a new stream is available, which carries the audience’s `userId` .
vii. The anchor calls `startPlay` to play the audience’s video.

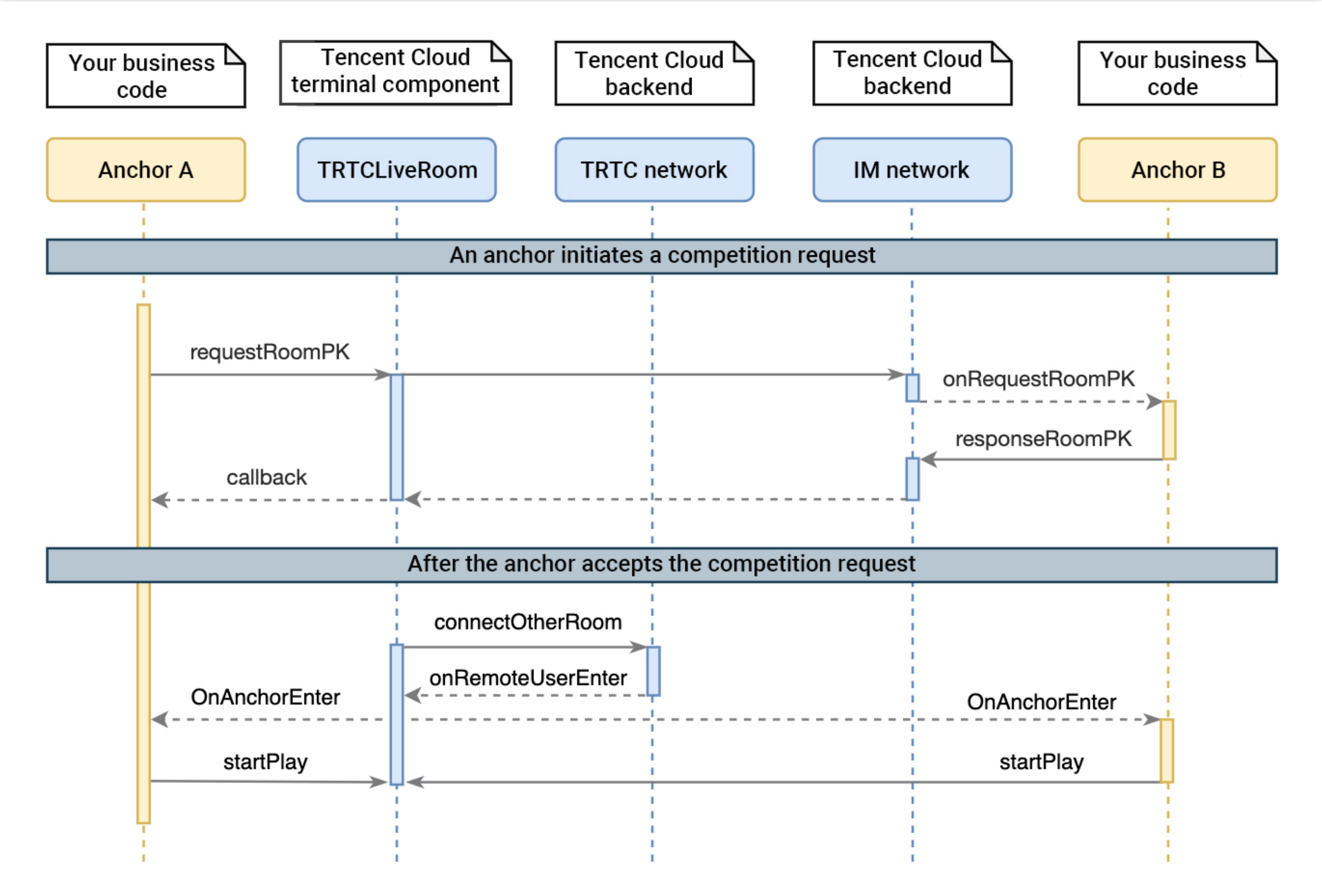


```
// Viewer:
// 1. The audience sends a co-anchoring request
```

```
mliveRoom.requestJoinAnchor(reason: mSelfUserId + ”requested to co-anchor”, responseCallback:
{ [weak self] (agreed, reason) in
// 4. The request is accepted by the anchor
if agreed {
// 5. The audience turns on the camera and starts pushing streams
self?.mliveRoom.startCameraPreview(frontCamera: true, view: localView, callback: nil)
self?.mliveRoom.startPublish(streamID: streamID, callback: nil)
}
}, callback: callback)
// Anchor:
// 2. The anchor receives the co-anchoring request
public func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRequestJoinAnchor user: TRTCLiveUserI
nfo, reason: String?, timeout: Double) {
// 3. The anchor accepts the co-anchoring request
mliveRoom.responseJoinAnchor(userID: userID, agree: true, reason: ”agreed to co-anchor”)
}
// 6. The anchor receives a notification that the co-anchoring audience has turned on the mic
public func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {
// 7. The anchor plays the audience’ s video
mliveRoom.startPlay(userID: userID, view: view, callback: nil)
}
```

Step 8. Compete across rooms

i. Anchor A calls `requestRoomPK` to send a competition request to anchor B.
ii. Anchor B receives the `TRTCLiveRoomDelegate onRequestRoomPK` notification.
iii. Anchor B calls `responseRoomPK` to accept or decline the competition request.
iv. Anchor B accepts the request, waits for the `TRTCLiveRoomDelegate onAnchorEnter` notification, and calls `startPlay` to play anchor A's video image.
v. Anchor A receives the `responseCallback` notification, which specifies whether the competition request is accepted.
vi. The request is accepted, and after receiving the `TRTCLiveRoomDelegate onAnchorEnter` notification, anchor A calls `startPlay` to play anchor B's video.



```
// Anchor A:
// Create room 12345
mLiveRoom.createRoom(roomID: 12345, roomParam: param, callback: nil)

// 1. Send a competition request to anchor B
mLiveRoom.requestRoomPK(roomID: 54321, userID: ”B”, responseCallback: { (agree, reason) in
// 5. Receive a callback of whether the request is accepted by anchor B
if agree {
}
}, callback: callback)

// Anchor A receives the callback of anchor B’ s entry.
public func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {
// 6. After receiving the notification of anchor B’ s entry, anchor A plays anchor B’s video image.
mLiveRoom.startPlay(userID: userID, view: view, callback: callback)
}

// Anchor B:
// Create room 54321
```

```
mLiveRoom.createRoom(roomID: 54321, roomParam: param, callback: nil)

// 2. Receive anchor A’ s request
public func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRequestRoomPK user: TRTCLiveUserInfo, timeout: Double) {
// 3. Accept anchor A’s request
mLiveRoom.responseRoomPK(userID: userID, agree: true, reason: reason)
}

public func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {
// 4. Receive a notification about anchor A’ s entry and play anchor A’s video
mLiveRoom.startPlay(userID: userID, view: view, callback: callback)
}
```

Step 9. Enable text chat and on-screen comments

- Call `sendRoomTextMsg` to send common text messages. All users in the room will receive the `onRecvRoomTextMsg` callback.

  IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: send text messages
mLiveRoom.sendRoomTextMsg(message: ”Hello Word!”, callback: callback)
// Recipient: listen for text messages
mLiveRoom.delegate = self
public func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRecvRoomTextMsg message: String, fromUser user: TRTCLiveUserInfo) {
debugPrint(”Received a text message from ¥(user.userName): ¥(message)”)
}
```

- Call `sendRoomCustomMsg` to send custom (signaling) messages. All users in the room will receive an `onRecvRoomCustomMsg` callback.

  Custom messages are often used to transfer custom signals, e.g., to give and broadcast likes.

```
// Sender: customize CMD to distinguish on-screen comments and likes
// For example, use ”CMD_DANMU” to indicate on-screen comments and ”CMD_LIKE” to indicate likes.
mLiveRoom.sendRoomCustomMsg(command: ”CMD_DANMU”, message: ”Hello world”, callback: nil)
mLiveRoom.sendRoomCustomMsg(command: ”CMD_LIKE”, message: ””, callback: nil)
// Recipient: listen for custom messages
mLiveRoom.delegate = self
public func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRecvRoomCustomMsg command: String, message: String, fromUser user: TRTCLiveUserInfo) {
 if "CMD_DANMU" == command {
// An on-screen comment is received.
debugPrint("Received an on-screen comment from ¥(user.userName): ¥(message)")
} else if "CMD_LIKE" == command {
// Receive a like
debugPrint("¥(user.userName) liked you.")
}
}
```

Interactive Video Streaming (Android)

Last updated : 2022-04-11 17:09:09

Demo UI

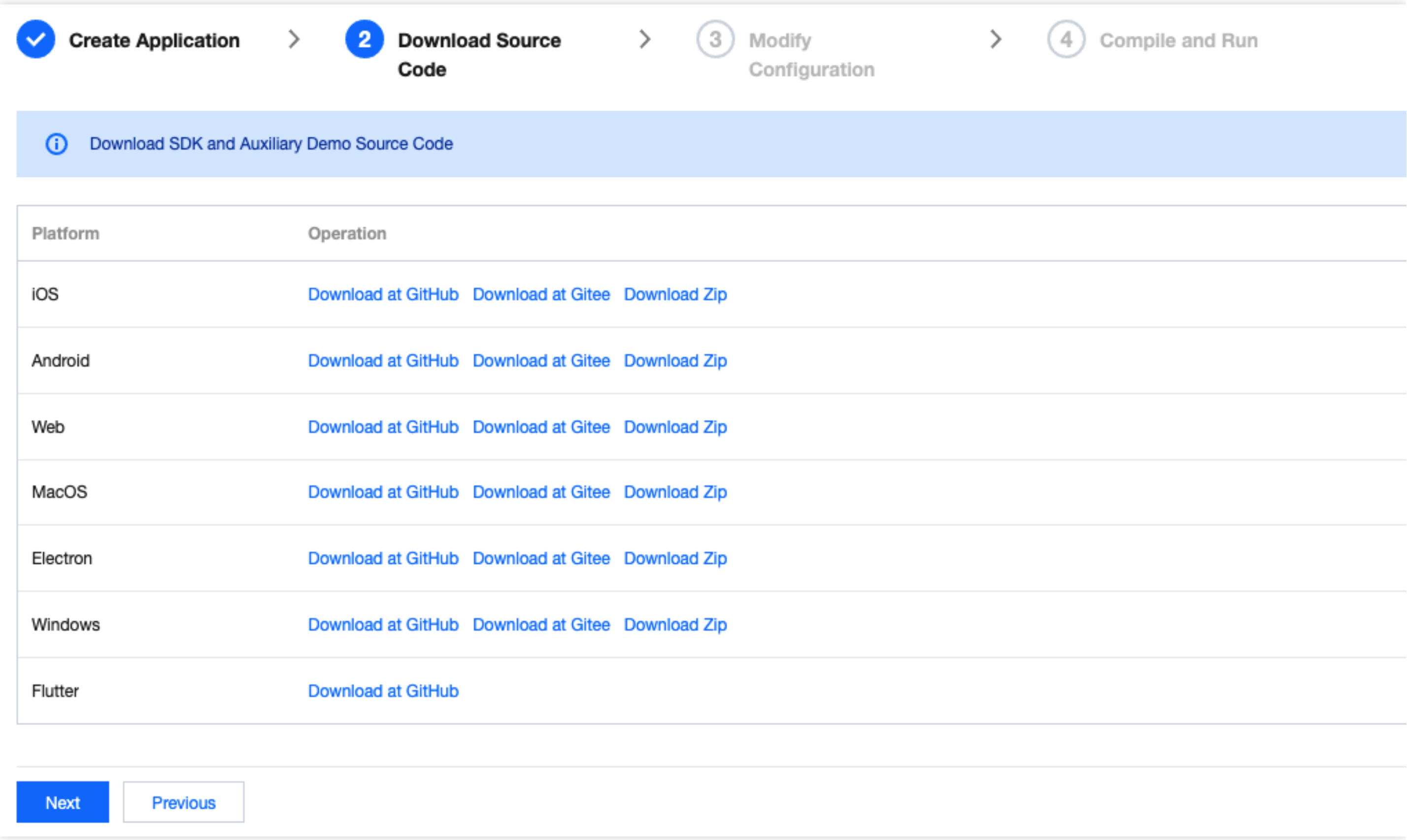
You can download and install the demo app we provide to try out TRTC’s interactive live streaming features, including co-anchoring, anchor competition, low-latency watch, and on-screen comments.

To quickly enable the interactive live video streaming feature, you can modify the demo app we provide and adapt it to your needs. You can also use the `TRTCLiveRoom` component and customize your own UI.

Using the App’s UI

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestLiveRoom` and click **Create**.
3. Click **Next** to skip this step.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Note :
The video conferencing feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Value-added Service Pricing for its billing details.

Step 2. Download the application source code

Clone or download the TUILiveRoom source code.

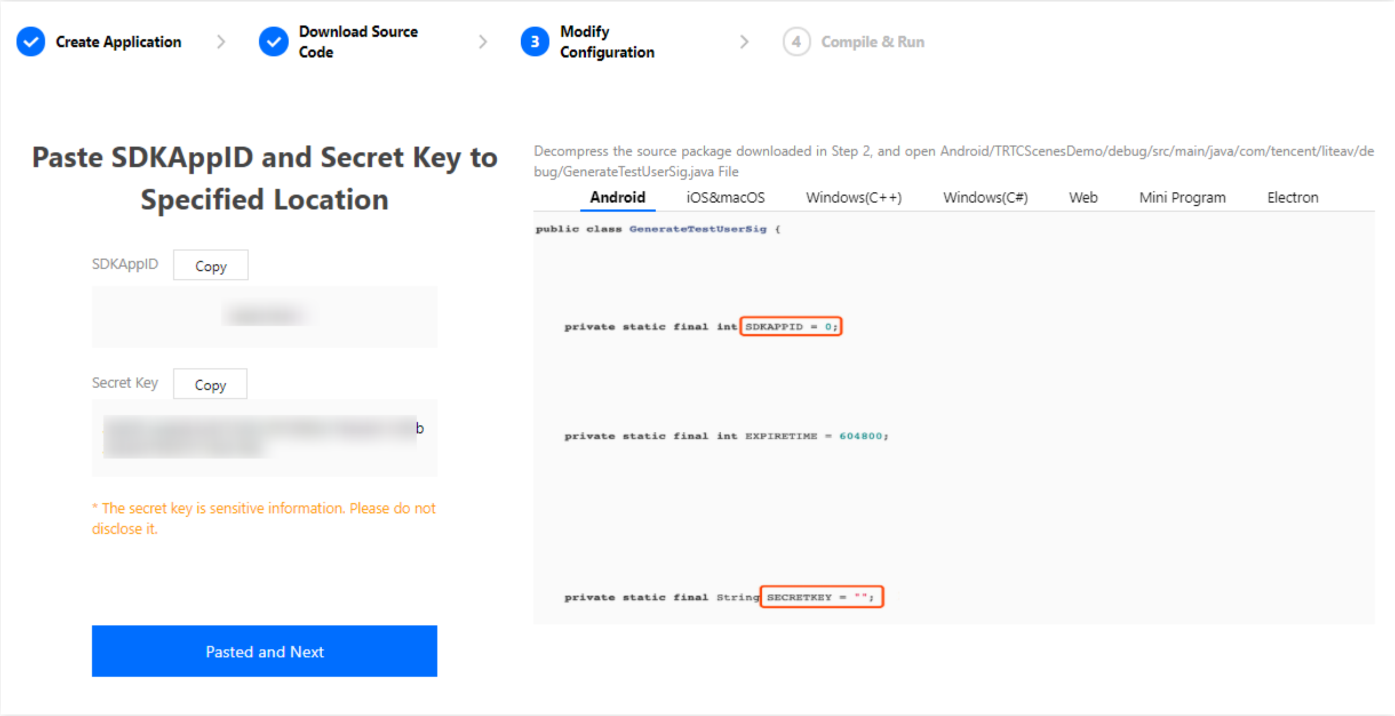
Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `Android/Debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java` .

3. Set parameters in `GenerateTestUserSig.java` :

   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {



    private static final int SDKAPPID = 0;



    private static final int EXPIRETIME = 604800;



    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note :
> The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUILiveRoom` with Android Studio (version 3.5 or above) and click **Run**.

Step 5. Modify the app’s source code

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `ui` subfolder contains UI code. The table below lists the files (folders) and the UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
anchor	Implementation code for anchor-end views
audience	Implementation code for audience-end views
common	Implementation code for common UI components
widget	Common controls

Tryout

Note :

You need at least two devices to try out the application.

User A

1. Enter a username (**which must be unique**) and log in.
2. Click **Create Room**.
3. Enter a room name and tap **Start**.

User B

1. Enter a username (**which must be unique**).
2. Enter the ID of the room created by user A, and tap **Join**.

Note :

You can find the room ID at the top of user A’s room view.

Listening for Room Status and Getting Room List for Co-Anchoring

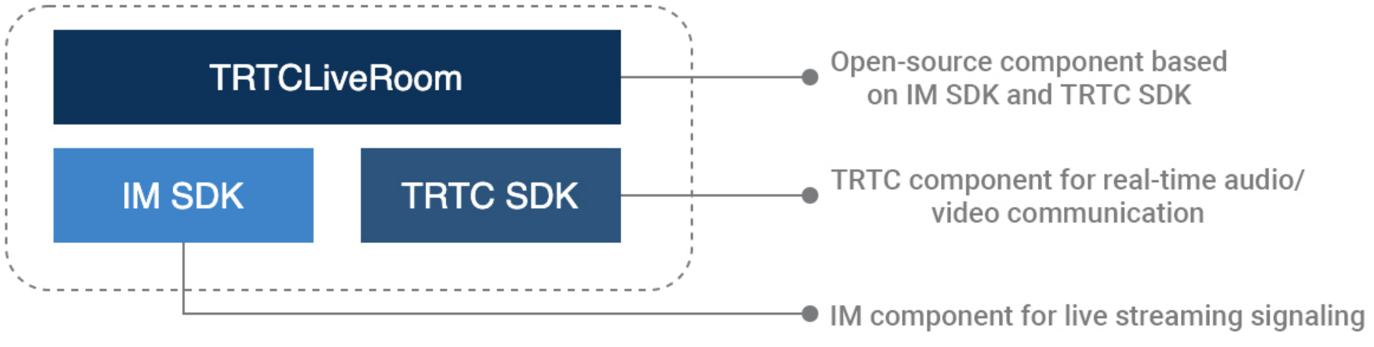
You can use `LiveRoomManager` to listen for room status.

```
LiveRoomManager.getInstance().addCallback(new LiveRoomManager.RoomCallback() {
/**
* A room was created
* @param roomId
* @param callback The result of internal processing
*/
@Override
public void onRoomCreate(int roomId, final LiveRoomManager.ActionCallback callback) {
// doSomething
}

/**
* A room was terminated
* @param roomId
* @param callback The result of internal processing
*/
@Override
public void onRoomDestroy(int roomId, final LiveRoomManager.ActionCallback callback) {
// doSomething
}

/**
* The room list was obtained
* @param callback
*/
@Override
public void onGetRoomIdList(final LiveRoomManager.GetCallback callback) {
// The room list was obtained, which is used for co-anchoring and must be maintained by yourself
if(callback != null) {
if(success) {
// Callback successful, and the room list was returned
callback.onSuccess(roomList);
} else {
// Failed to get the room list
callback.onError(code, message);
}
}
}
});
```

Getting the room list may involve async operations. Using callbacks to get the list provides greater convenience and flexibility.

Customizing UI

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `model` subfolder contains the reusable open-source component `TRTCLiveRoom`. You can find the component's APIs in `TRTCLiveRoom.java` and use them to customize your own UI.


TRTCLiveRoom
IM SDK
TRTC SDK
Open-source component based on IM SDK and TRTC SDK
TRTC component for real-time audio/video communication
IM component for live streaming signaling

Step 1. Integrate the SDK

The interactive live streaming component `TRTCLiveRoom` depends on the TRTC SDK and IM SDK. Follow the steps below to integrate them into your project.

Method 1: adding dependencies via Maven

1. Add the TRTC SDK and IM SDK dependencies to `dependencies`.

```
dependencies {
compile "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"
compile 'com.tencent.imsdk:imsdk:latest.release'
}
```

2. In `defaultConfig`, specify the CPU architecture to be used by your application.

```
defaultConfig {
ndk {
abiFilters "armeabi-v7a"
}
}
```

3. Click **Sync Now** to automatically download the SDKs and integrate them into your project.

Method 2: adding dependencies through local AAR files

If your access to the Maven repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration document
IM SDK	Download	Integration document

Step 2. Configure permission requests and obfuscation rules

Configure permission requests for your application in `AndroidManifest.xml` . The SDKs need the following permissions (on Android 6.0 and above, the camera and read storage permissions must be requested at runtime.)

```
<uses-permission android:name=”android.permission.INTERNET”>
<uses-permission android:name=”android.permission.ACCESS_NETWORK_STATE”>
<uses-permission android:name=”android.permission.ACCESS_WIFI_STATE”>
<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE”>
<uses-permission android:name=”android.permission.READ_EXTERNAL_STORAGE”>
<uses-permission android:name=”android.permission.RECORD_AUDIO”>
<uses-permission android:name=”android.permission.MODIFY_AUDIO_SETTINGS”>
<uses-permission android:name=”android.permission.BLUETOOTH”>
<uses-permission android:name=”android.permission.CAMERA”>
<uses-permission android:name=”android.permission.READ_PHONE_STATE”>
<uses-feature android:name=”android.hardware.camera”>
<uses-feature android:name=”android.hardware.camera.autofocus”>
```

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

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

Step 3. Import the `TRTCLiveRoom` component

Copy all the files in the directory below to your project:

```
src/main/java/com/tencent/liteav/liveroom/model
```

Step 4. Create an instance and log in

1. Call the `sharedInstance` API to create an instance of the `TRTCLiveRoom` component.
2. Create a `TRTCLiveRoomConfig` object, and set its `useCDNFirst` and `CDNPlayDomain` attributes.

- `useCDNFirst` : specifies the way audience watch live streams. `true` means that audience watch live streams over CDNs, which is cost-efficient but has high latency. `false` means that audience watch live streams in the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is within 1 second.
- `CDNPlayDomain` : specifies the domain name for CDN live streaming, which takes effect only if `useCDNFirst` is set to `true` . You can set it in CSS console > **Domain Management**.

3. Call the `login` API to log in to the component, and set the key parameters as described below.

Parameter	Description
SDKAppID	You can view \`SDKAppID\` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.
config	Global configuration information. Please initialize it during login as it cannot be modified after login. ◦ `useCDNFirst` : specifies the way audience watch live streams. `true` means that audience watch live streams over CDNs, which is cost-efficient but has high latency. `false` means that audience watch live streams in the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is within 1 second. ◦ `CDNPlayDomain` : specifies the domain name for CDN live streaming, which takes effect only if `useCDNFirst` is set to `true` . You can set it in CSS console > **Domain Management**.
callback	Login callback. The code is \`0\` if login is successful.

```
TRTCLiveRoom mLiveRoom = TRTCLiveRoom.sharedInstance(this);
// useCDNFirst: `true` means that audience watch live streams over CDNs, and `false` means tha
t audience watch live streams in the low latency mode.
// yourCDNPlayDomain: the playback domain name for CDN live streaming
TRTCLiveRoomDef.TRTCLiveRoomConfig config =
new TRTCLiveRoomDef.TRTCLiveRoomConfig(useCDNFirst, yourCDNPlayDomain);
mLiveRoom.login(SDKAPPID, userId, userSig, config,
new TRTCLiveRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// Logged in
}
```

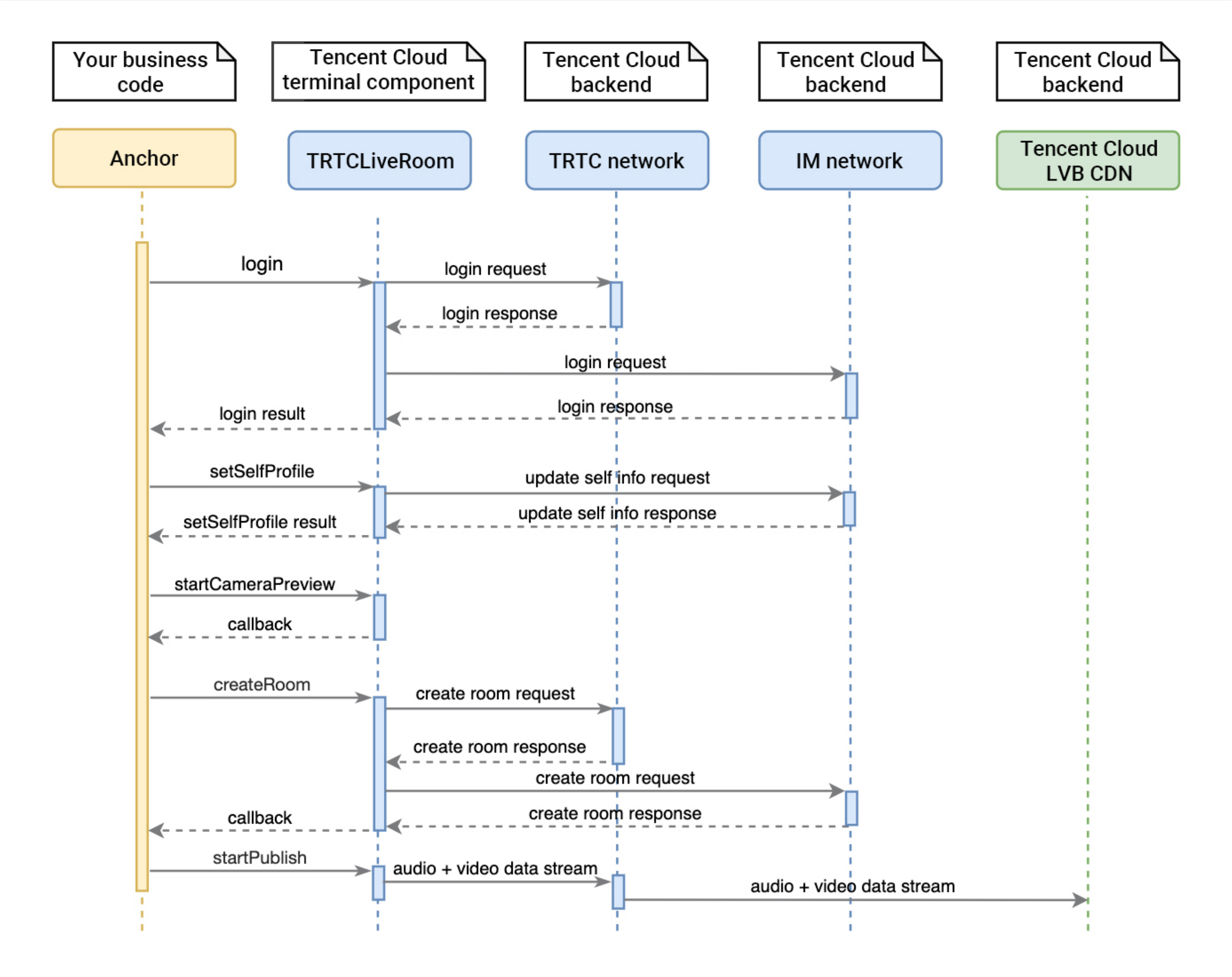
```
    }
});
```

Step 5. Create a room and push streams

i. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
ii. Before streaming, you can call `startCameraPreview` to enable camera preview, add beauty filter buttons to the UI, and set beauty filters through `getBeautyManager` .

> Note :
> Only the Enterprise Edition SDK supports advanced beauty filter features such as face changing and stickers.

iii. After setting beauty filters, call `createRoom` to create a live streaming room.
iv. Call `startPublish` to start streaming. To enable CDN live streaming, specify `useCDNFirst` and `CDNPlayDomain` in the `TRTCLiveRoomConfig` parameter, which is passed in during login, and specify `streamID` for playback when calling `startPublish` .



```
// 1. Set your username and profile photo as an anchor
mLiveRoom.setSelfProfile("A", "your_face_url", null);

// 2. Enable camera preview and set beauty filters before streaming
TXCloudVideoView view = new TXCloudVideoView(context);
parentView.add(view);
mLiveRoom.startCameraPreview(true, view, null);
mLiveRoom.getBeautyManager().setBeautyStyle(1);
mLiveRoom.getBeautyManager().setBeautyLevel(6);

// 3. Create a room
TRTCLiveRoomDef.TRTCCreateRoomParam param = new TRTCLiveRoomDef.TRTCCreateRoomParam();
param.roomName = "Test room";
mLiveRoom.createRoom(123456789, param, new TRTCLiveRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
```

```
if (code == 0) {
// 4. Start streaming and publish the streams to CDNs
mLiveRoom.startPublish(mSelfUserId + ”_stream”, null);
}
}
});
```

Step 6. Play back streams

i. After performing step 4 to log in, you can call `setSelfProfile` to set your nickname and profile photo.

ii. Get the latest room list from the backend.

> Note :
> The room list in the demo app is for demonstration only. The business logic of live streaming room lists varies significantly. Tencent Cloud does not provide list management services for the time being. Please manage the list by yourself.

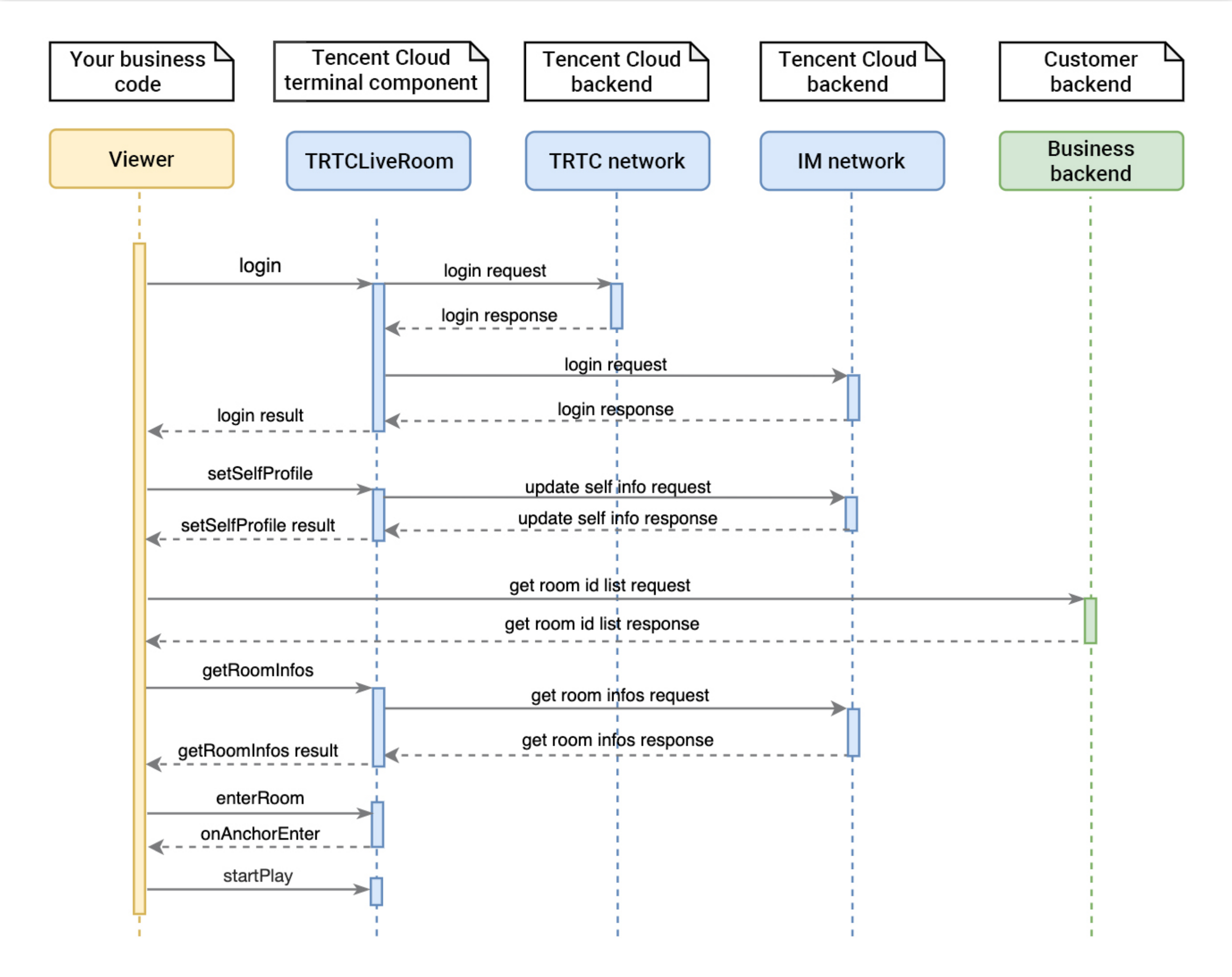
iii. Call `getRoomInfos` to get short descriptions of the rooms, which are provided by anchors when they call `createRoom` to create the rooms.

> Note :
> If your room list already displays enough information, you can skip the step of calling `getRoomInfos` .

iv. Select a room, call `enterRoom` , with the room ID passed in to enter the room.

v. Call `startPlay` , with the anchor’s `userId` passed in to start playback.

- If the room list displays the `userId` of the anchor, you can call `startPlay` , passing in the anchor’s `userId` to start playback.
- If you do not know the anchor's `userId` , you can find it in the `onAnchorEnter` event callback, which you will receive after entering the room. You can then call `startPlay` to start playback.

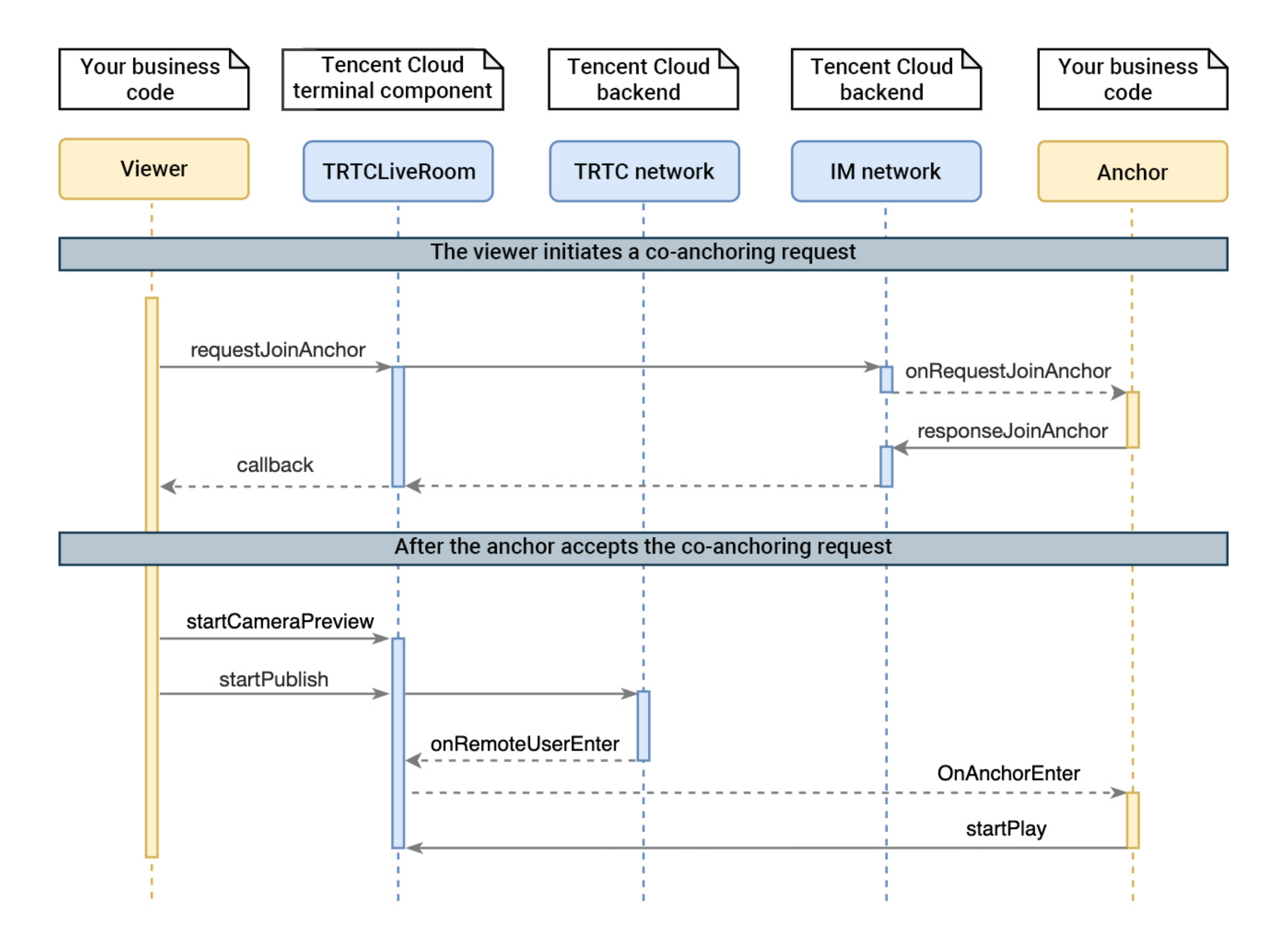


```
// 1. Get the room list from the backend. Suppose it is `roomList`
List<integer> roomList = GetRoomList();

// 2. Call `getRoomInfos` to get the details of the room
mLiveRoom.getRoomInfos(roomList, new TRTCLiveRoomCallback.RoomInfoCallback() {
@Override
public void onCallback(int code, String msg, List<trtcliveroomdef.trtcliveroominfo> list) {
if (code == 0) {
// After getting the room information, you can display on the anchor list page the anchor’s nickname, profile photo, and other information
}
}
})

// 3. Select a `roomid` and enter the room
mLiveRoom.enterRoom(roomid, null);
```

```
// 4. After receiving the notification about the anchor’ s entry, start playback
mLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
@Override
public void onAnchorEnter(final String userId) {
// 5. Play the anchor’s video
mLiveRoom.startPlay(userId, mTXCloudVideoView, null);
}
});
```

Step 7. Co-anchor

i. Audience calls `requestJoinAnchor` to send a co-anchoring request to the anchor.
ii. The anchor receives a notification (`TRTCLiveRoomDelegate#onRequestJoinAnchor`) about the co-anchoring request.
iii. The anchor calls `responseJoinAnchor` to accept or decline the co-anchoring request.
iv. The audience receives the `TRTCLiveRoomDelegate#responseCallback` event notification, which carries the anchor’s response.
v. If the anchor accepts the co-anchoring request, the audience can call `startCameraPreview` to turn the local camera on and then `startPublish` to push streams.
vi. The anchor receives a notification (`TRTCLiveRoomDelegate#onAnchorEnter`) that a new stream is available, which carries the audience’s `userId` .
vii. The anchor calls `startPlay` to play the audience’s video.



```
// 1. The audience sends a co-anchoring request
mLiveRoom.requestJoinAnchor(mSelfUserId + "requested to co-anchor",
new TRTCLiveRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// 4. The request is accepted by the anchor
TXCloudVideoView view = new TXCloudVideoView(context);
parentView.add(view);
// 5. The audience turns on the camera and starts pushing streams
mLiveRoom.startCameraPreview(true, view, null);
mLiveRoom.startPublish(mSelfUserId + "_stream", null);
}
}
});

// 2. The anchor receives the co-anchoring request
mLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
```

```
@Override
public void onRequestJoinAnchor(final TRTCLiveRoomDef.TRTCLiveUserInfo userInfo,
String reason, final int timeout) {
// 3. The anchor accepts the co-anchoring request
mLiveRoom.responseJoinAnchor(userInfo.userId, true, "agreed to co-anchor");
}

@Override
public void onAnchorEnter(final String userId) {
// 6. The anchor receives a notification that the co-anchoring audience has turned on the mic
TXCloudVideoView view = new TXCloudVideoView(context);
parentView.add(view);
// 7. The anchor plays the audience’ s video
mLiveRoom.startPlay(userId, view, null);
}
});
```

Step 8. Compete across rooms

i. Anchor A calls `requestRoomPK` to send a competition request to anchor B.

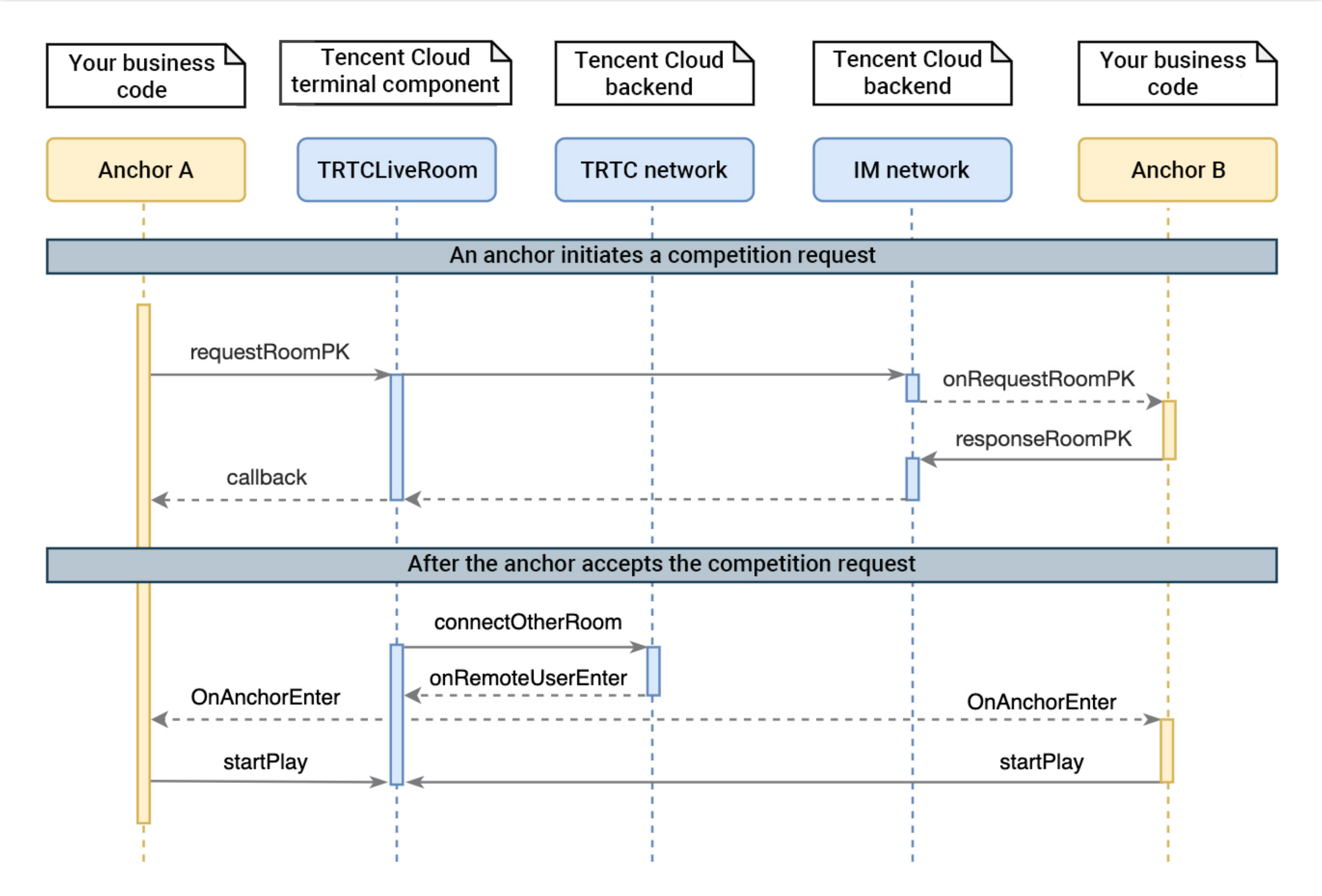
ii. Anchor B receives the `TRTCLiveRoomDelegate onRequestRoomPK` notification.

iii. Anchor B calls `responseRoomPK` to accept or decline the competition request.

iv. Anchor B accepts the request and, after receiving the `TRTCLiveRoomDelegate onAnchorEnter` notification, calls `startPlay` to play anchor A's video.

v. Anchor A receives the `responseCallback` notification, which specifies whether the competition request is accepted.

vi. The request is accepted, and after receiving the `TRTCLiveRoomDelegate onAnchorEnter` notification, anchor A calls `startPlay` to play anchor B's video.



```
// Anchor A:
// Create room 12345
mLiveRoom.createRoom(12345, param, null);

mLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
@Override
public void onAnchorEnter(final String userId) {
// 6. Receive a notification about anchor B’ s entry
mLiveRoom.startPlay(userId, mTXCloudVideoView, null);
}
});

// 1. Send a competition request to anchor B
mLiveRoom.requestRoomPK(54321, ”B”,
new TRTCLiveRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
// 5. Receive a callback of whether the request is accepted by anchor B
if (code == 0) {
// Accepted
} else {
// Declined
}
}
});

// Anchor B:
// Create room 54321
mLiveRoom.createRoom(54321, param, null);

// 2. Receive anchor A’ s request
mLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
@Override
public void onRequestRoomPK(
final TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, final int timeout) {
// 3. Accept anchor A’s request
mLiveRoom.responseRoomPK(userInfo.userId, true, ””);
}
@Override
public void onAnchorEnter(final String userId) {
// 4. Receive a notification about anchor A’ s entry and play anchor A’s video
mLiveRoom.startPlay(userId, mTXCloudVideoView, null);
}
});
```

Step 9. Enable text chat and on-screen comments

- Call `sendRoomTextMsg` to send common text messages. All users in the room will receive the `onRecvRoomTextMsg` callback.

  IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

  ```
  // Sender: send text messages
  mLiveRoom.sendRoomTextMsg(”Hello Word!”, null);
  // Recipient: listen for text messages
  mLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
  @Override
  public void onRecvRoomTextMsg(String roomId,
  String message, TRTCLiveRoomDef.TRTCLiveUserInfo userInfo) {
  Log.d(TAG, ”Received a message from” + userInfo.userName + ”: ” + message);
  }
  });
  ```

- Call `sendRoomCustomMsg` to send custom (signaling) messages. All users in the room will receive an `onRecvRoomCustomMsg` callback.

  Custom messages are often used to transfer custom signals, e.g., to give and broadcast likes.

```
// For example, use ”CMD_DANMU” to indicate on-screen comments and ”CMD_LIKE” to indicate li
kes.
mLiveRoom.sendRoomCustomMsg(”CMD_DANMU”, ”Hello world”, null);
mLiveRoom.sendRoomCustomMsg(”CMD_LIKE”, ””, null);
// Recipient: listen for custom messages
mLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
@Override
public void onRecvRoomCustomMsg(String roomId, String cmd,
String message, TRTCLiveRoomDef.TRTCLiveUserInfo userInfo) {
if (”CMD_DANMU”.equals(cmd)) {
// An on-screen comment is received.
Log.d(TAG, ”Received an on-screen comment from” + userInfo.userName + ”: ” + message);
} else if (”CMD_LIKE”.equals(cmd)) {
// A like is received.
Log.d(TAG, userInfo.userName + ”liked you.”);
}
}
});
```

Interactive Video Streaming (Flutter)

Last updated：2022-03-31 11:38:29

To quickly enable the interactive live video streaming feature, you can modify the demo we provide and adapt it to your needs. You can also use the `TRTCLiveRoom` component and customize your own UI.

Using the Demo UI

Step 1. Create an application

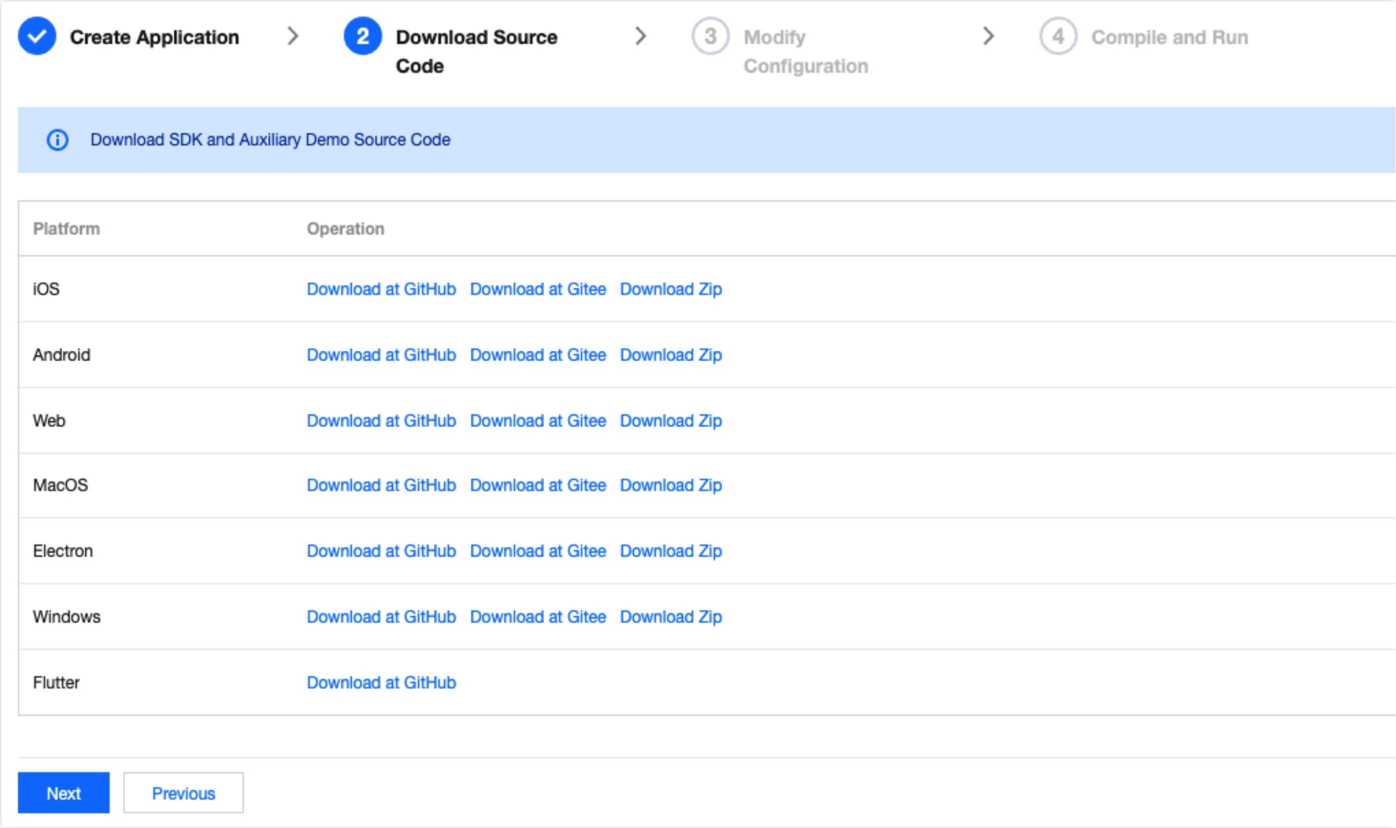
1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestLive` and click **Create**.

> Note：
> This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Value-added Service Pricing for its billing details.

Step 2. Download the SDK and demo source code

1. Download the SDK and demo source code for your platform.

2. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

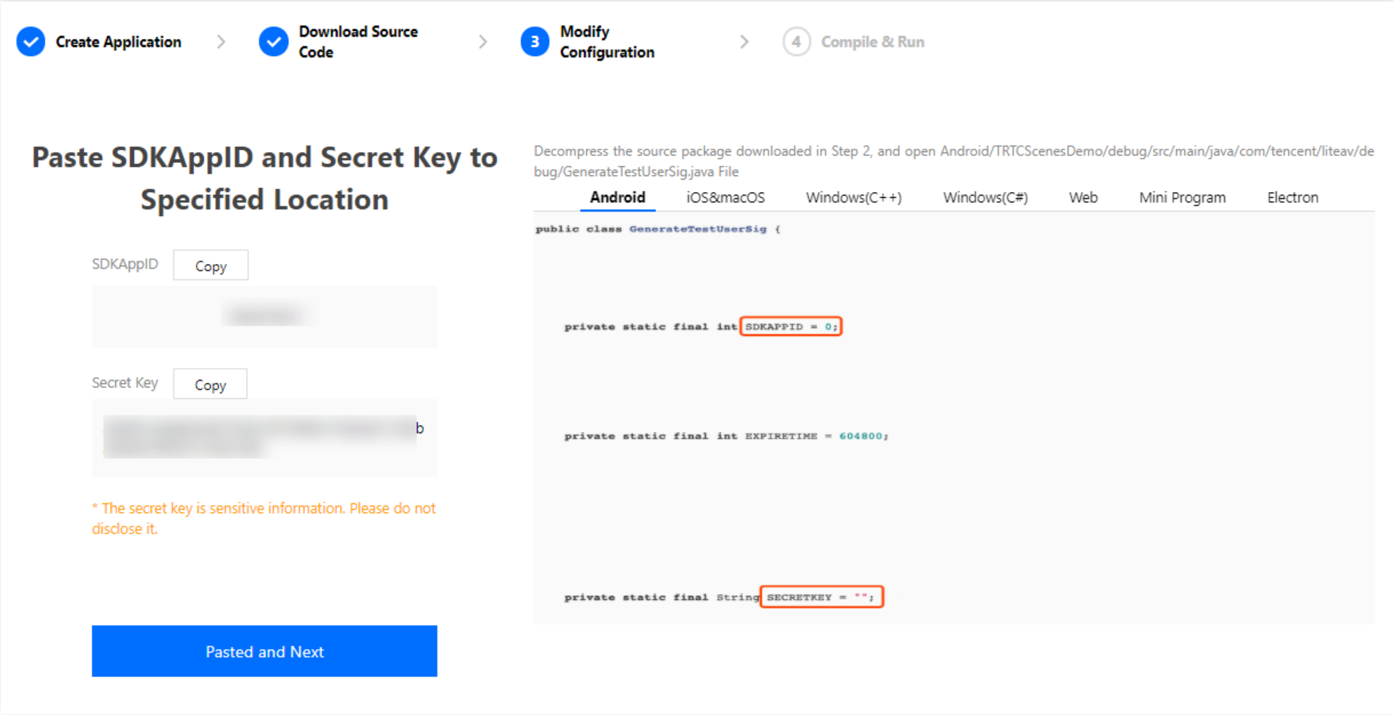
Step 3. Configure the demo project file

1. In the **Modify Configuration** step, select your platform.

2. Find and open `/example/lib/debug/GenerateTestUserSig.dart` .

3. Set parameters in `GenerateTestUserSig.dart` as follows.

   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Pasted and Next

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {



    private static final int SDKAPPID = 0;



    private static final int EXPIRETIME = 604800;



    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is leaked, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for the local execution and debugging of the demo**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Compile and run

> Note :
>
> An Android project must be run on a real device rather than a simulator.

1. Run `flutter pub get` .
2. Compile, run, and test the project.
   - Android
   - iOS
   1. Run `flutter run` .
   2. Open the demo project with Android Studio (3.5 or above), and click **Run**.

Step 5. Modify the demo source code

The `TRTCLiveRoom` folder in the source code contains two subfolders: `ui` and `model` . The `ui` subfolder contains UI code. The table below lists the files (folders) and the UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
base	Basic classes used by the UI
list	The list and room creation views
room	Main room views for speakers and listeners

Customizing UI

The `TRTCLiveRoom` folder in the source code contains two subfolders: `ui` and `model` . The `model` subfolder contains the reusable open-source component `TRTCLiveRoom` . You can find the component's APIs in `TRTCLiveRoom.dart` and use them to customize your own UI.

Step 1. Integrate the SDKs

The interactive live streaming component `TRTCLiveRoom` depends on the TRTC SDK and IM SDK. You can configure `pubspec.yaml` to download their updates automatically.

Add the following dependencies to `pubspec.yaml` of your project.

```
dependencies:
tencent_trtc_cloud: latest version number
tencent_im_sdk_plugin: latest version number
```

Step 2. Configure permission requests and obfuscation rules

- iOS
- Android

Add request for mic permission in `Info.plist` :

```
<key>NSMicrophoneUsageDescription</key>
<string>Audio calls are possible only with mic access.</string>
```

Step 3. Import the `TRTCLiveRoom` component

Copy all the files in the directory below to your project:

```
lib/TRTCLiveRoom/model/
```

Step 4. Create an instance and log in

1. Call the `sharedInstance` API to create an instance of the `TRTCLiveRoom` component.
2. Call the `registerListener` function to register event callbacks of the component.
3. Call the `login` API to log in to the component, and set the key parameters as described below.

Parameter	Description
SDKAppID	You can view `SDKAppID` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.
useCDNFirst	Specifies the way audience watch live streams. `true` means that audience watch live streams over CDNs, which is cost-efficient but has high latency. `false` means that audience watch live streams in the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is within 1 second.
CDNPlayDomain	Specifies the domain name for CDN live streaming, which takes effect only if `useCDNFirst` is set to `true`. You can set it in CSS console > Domain Management.

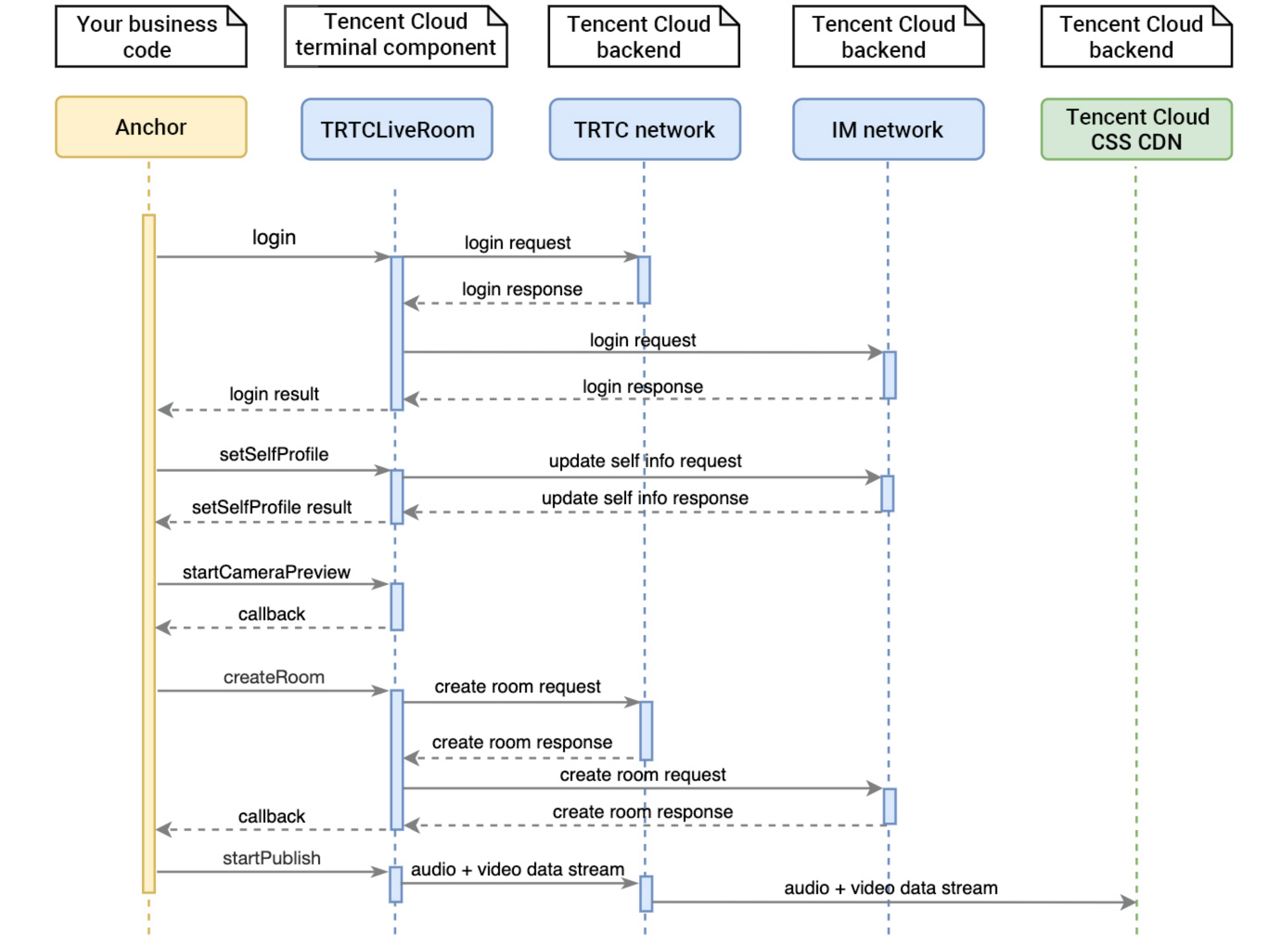
Step 5. Create a room and push streams

1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. Before streaming, you can call `startCameraPreview` to enable camera preview, add beauty filter buttons to the UI, and set beauty filters through `getBeautyManager` .

Note :

> Only the Enterprise Edition SDK supports face changing and stickers.

3. After setting beauty filters, call `createRoom` to create a live streaming room.
4. Call `startPublish` to start streaming. To enable CDN live streaming, specify `useCDNFirst` and `CDNPlayDomain` in the `TRTCLiveRoomConfig` parameter, which is passed in during login, and specify `streamID` for playback when calling `startPublish` .



Step 6. Play back streams

1. After performing step 4 to log in, you can call `setSelfProfile` to set your nickname and profile photo.
2. Get the latest room list from the backend.

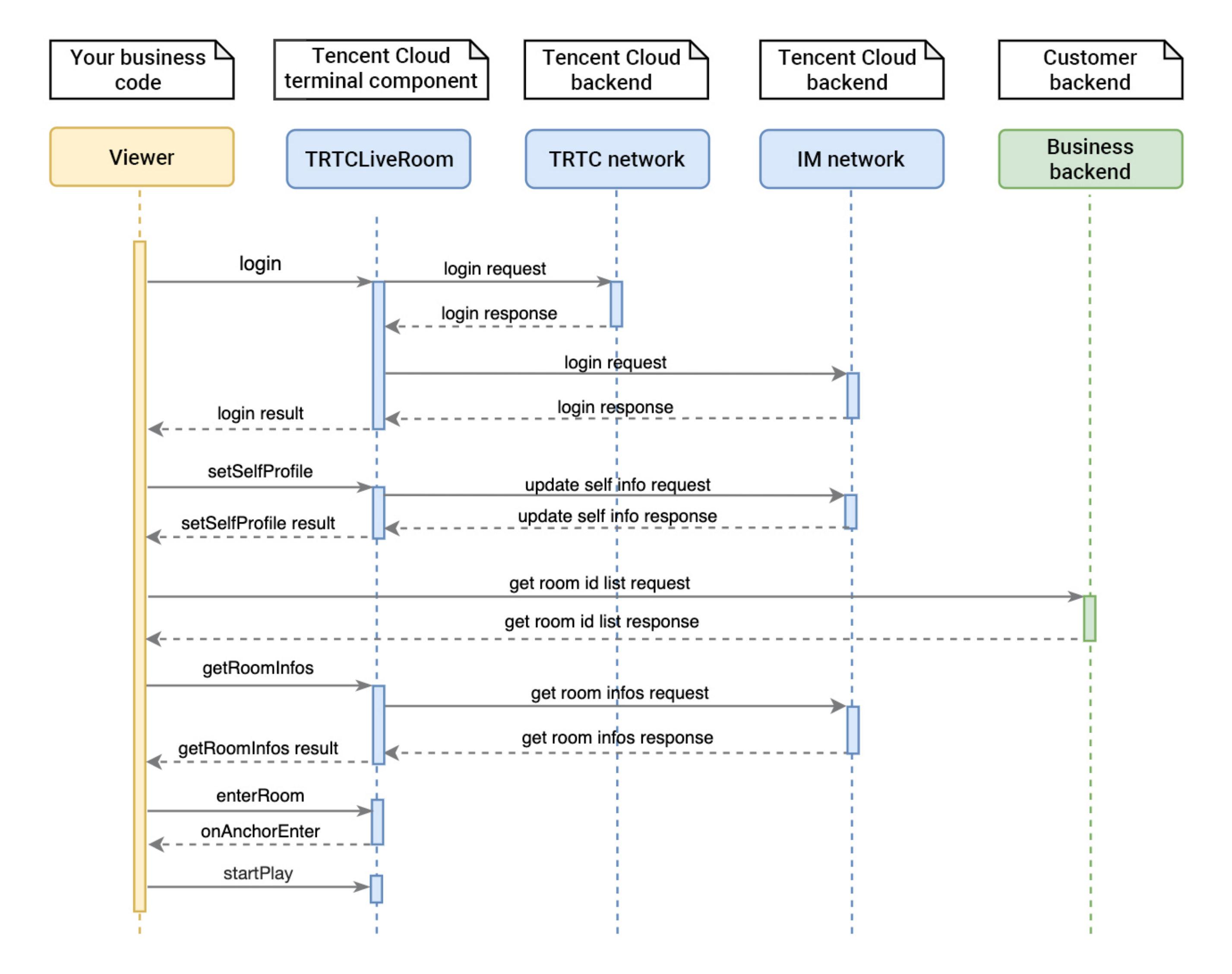
> Note :
> The room list in the demo is for demonstration only. The business logic of live streaming room lists vary significantly. Tencent Cloud does not provide list management services for the time being. Please manage the list by yourself.

3. Call `getRoomInfos` to get short descriptions of the rooms, which are provided by anchors when they call `createRoom` to create the rooms.

> Note :
> If your room list already displays enough information, you can skip the step of calling `getRoomInfos` .

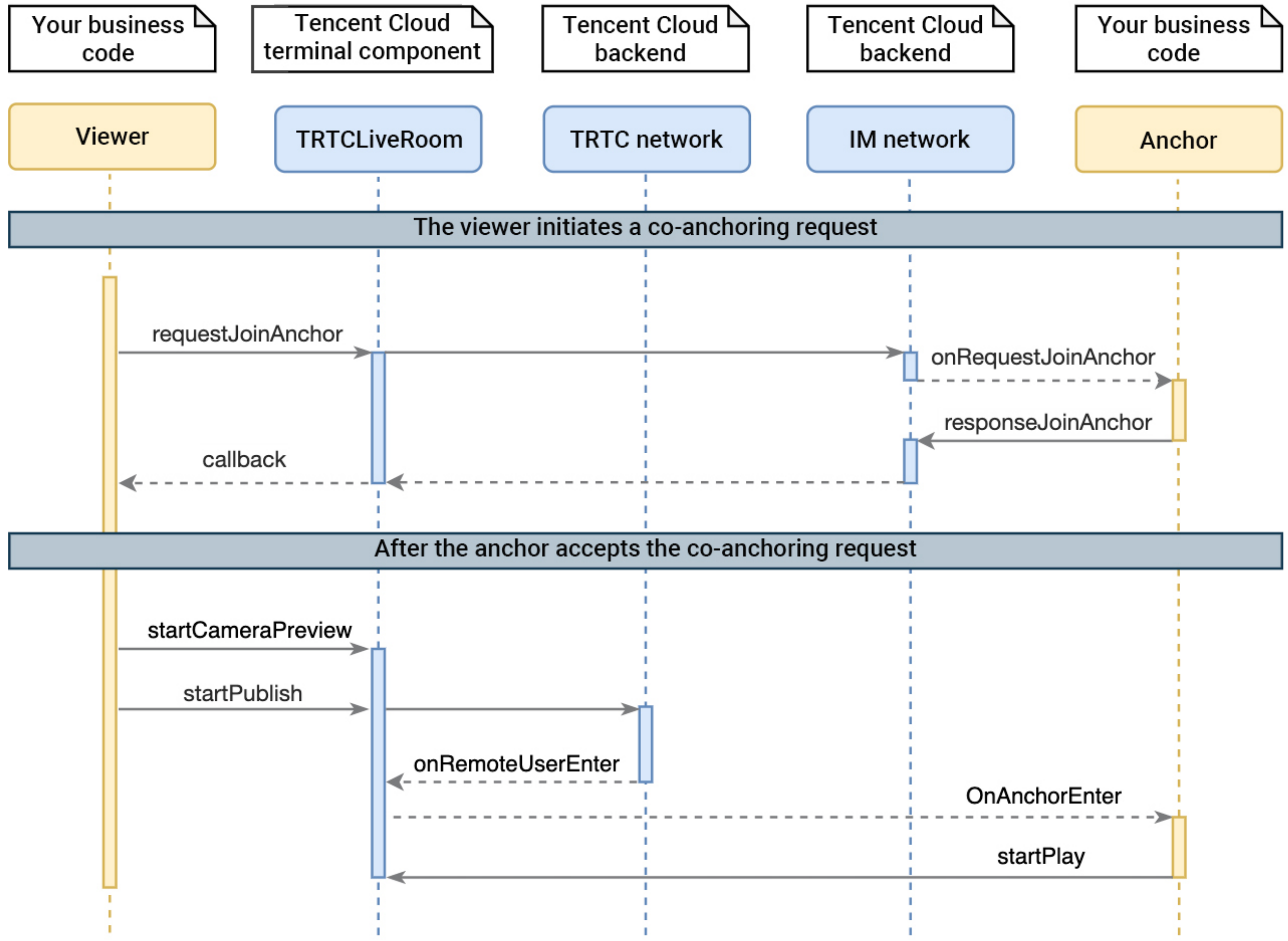
4. Select a room, call `enterRoom` , with the room ID passed in to enter the room.
5. Call `startPlay` , with the anchor's `userId` passed in to start playback.

- If the room list displays the `userId` of the anchor, you can call `startPlay` , passing in the anchor's `userId` to start playback.
- If you do not know the anchor's `userId` , you can find it in the `onAnchorEnter` event callback, which you will receive after entering the room. You can then call `startPlay` to start playback.



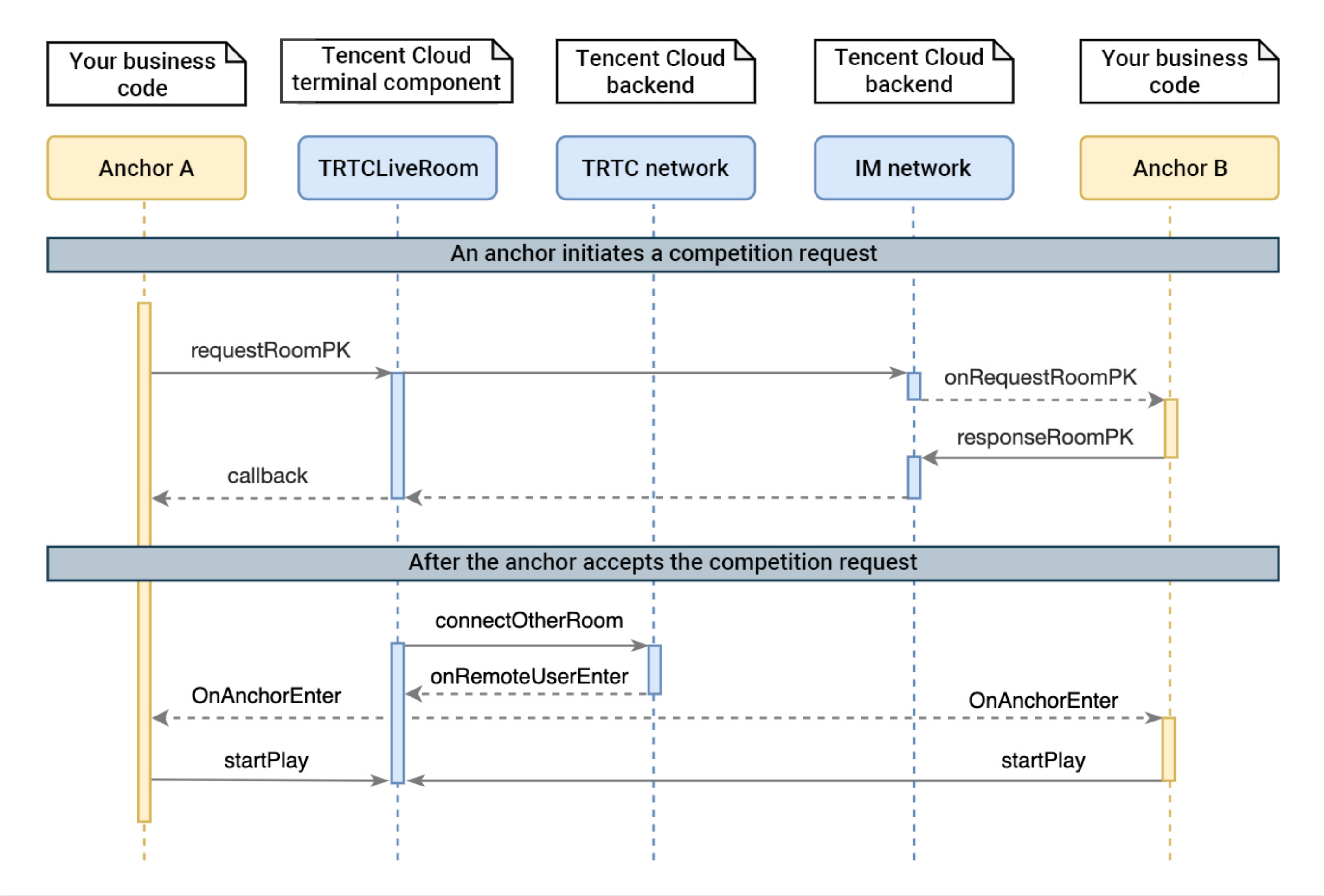
Step 7. Co-anchor

1. Audience calls `requestJoinAnchor` to send a co-anchoring request to the anchor.
2. The anchor receives a notification (`TRTCLiveRoomDelegate#onRequestJoinAnchor`) about the co-anchoring request.
3. The anchor calls `responseJoinAnchor` to accept or decline the co-anchoring request.
4. The audience receives the `onAnchorAccepted` event notification, which carries the anchor's response.
5. If the anchor accepts the co-anchoring request, the audience can call `startCameraPreview` to turn the local camera on and then `startPublish` to push streams.
6. The anchor receives a notification (`TRTCLiveRoomDelegate#onAnchorEnter`) that a new stream is available, which carries the audience’s `userId` .
7. The anchor calls `startPlay` to play the audience’s video.



Step 8. Compete across rooms

1. Anchor A calls `requestRoomPK` to send a competition request to anchor B.
2. Anchor B receives the `TRTCLiveRoomDelegate onRequestRoomPK` notification.
3. Anchor B calls `responseRoomPK` to accept or decline the competition request.
4. Anchor B accepts the request and, after receiving the `TRTCLiveRoomDelegate onAnchorEnter` notification, calls `startPlay` to play anchor A's video.
5. Anchor A receives the `onRoomPKAccepted` notification, which specifies whether the competition request is accepted.
6. The request is accepted, and after receiving the `TRTCLiveRoomDelegate onAnchorEnter` notification, anchor A calls `startPlay` to play anchor B's video.



Step 9. Enable text chat and on-screen comments

- Call `sendRoomTextMsg` to send common text messages. All users in the room will receive the `onRecvRoomTextMsg` callback.

  IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: send text messages
trtcLiveCloud.sendRoomTextMsg("Hello Word!");


// Recipient: listen for text messages
onListenerFunc(type, params) async {
switch (type) {
case TRTCLiveRoomDelegate.onRecvRoomTextMsg:
// Group text messages are received. This feature can be used to enable text chat rooms.
break;
}
}
```

- Call `sendRoomCustomMsg` to send custom (signaling) messages. All users in the room will receive an `onRecvRoomCustomMsg` callback.
  Custom messages are often used to transfer custom signals, e.g., to give and broadcast likes.

TRTCLiveRoom APIs (iOS)

Last updated : 2022-04-08 14:42:51

`TRTCLiveRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create a room and start live streaming, or enter a room as audience.
- The anchor and audience in a room can co-anchor with each other.
- Anchors in two rooms can compete and interact with each other.
- All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

`TRTCLiveRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Interactive Live Video Streaming (iOS).

- TRTC SDK: The TRTC SDK is used as a low-latency live streaming component.
- IM SDK: The `AVChatRoom` feature of the IM SDK is used to implement chat rooms, and IM messages are used to facilitate the co-anchoring process.

TRTCLiveRoom API Overview

Basic SDK APIs

API	Description
delegate	Sets event callbacks.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets the profile.

Room APIs

API	Description
createRoom	Creates a room (called by anchor). If the room does not exist, the system will create the room automatically.
destroyRoom	Terminates a room (called by anchor).

API	Description
enterRoom	Enters a room (called by audience).
exitRoom	Exits a room (called by audience).
getRoomInfos	Gets room list details.
getAnchorList	Gets the anchors and co-anchoring viewers in a room. This API works only if it is called after `enterRoom()` .
getAudienceList	Gets the information of all audience in a room. This API works only if it is called after `enterRoom()` .

Stream pushing/pulling APIs

API	Description
startCameraPreview	Enables preview of the local video.
stopCameraPreview	Stops local video capturing and preview.
startPublish	Starts live streaming (pushing streams).
stopPublish	Stops live streaming (pushing streams).
startPlay	Plays a remote video. This API can be called in common playback and co-anchoring scenarios.
stopPlay	Stops rendering a remote video.

APIs for anchor-audience co-anchoring

API	Description
requestJoinAnchor	Requests co-anchoring (called by audience).
responseJoinAnchor	Responds to a co-anchoring request (called by anchor).
kickoutJoinAnchor	Removes a user from co-anchoring (called by anchor).

APIs for cross-room anchor competition

API	Description
requestRoomPK	Requests cross-room competition (called by anchor).

API	Description
responseRoomPK	Responds to a cross-room competition request (called by anchor).
quitRoomPK	Quits cross-room competition.

Audio/Video APIs

API	Description
switchCamera	Switches between the front and rear cameras.
setMirror	Sets the mirror mode.
muteLocalAudio	Mutes/Unmutes the local user.
muteRemoteAudio	Mutes/Unmutes a remote user.
muteAllRemoteAudio	Mutes/Unmutes all remote users.

Background music and audio effect APIs

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

Beauty filter APIs

API	Description
getBeautyManager	Gets the beauty filter management object TXBeautyManager.

Message sending APIs

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

Debugging APIs

API	Description
showVideoDebugLog	Specifies whether to display debugging information on the UI.

TRTCLiveRoomDelegate API Overview

Common event callback APIs

API	Description
onError	Error
onWarning	Warning
onDebugLog	Log

Room event callback APIs

API	Description
onRoomDestroy	The room was terminated.
onRoomInfoChange	The room information changed.

Callback APIs for entry/exit of anchors/audience

API	Description
onAnchorEnter	There is a new anchor/co-anchoring viewer in the room.
onAnchorExit	An anchor/co-anchoring viewer quit competition/co-anchoring.
onAudienceEnter	A viewer entered the room.
onAudienceExit	A viewer left the room.

Callback APIs for anchor-audience co-anchoring events

API	Description
onRequestJoinAnchor	A co-anchoring request was received.
onKickoutJoinAnchor	A user was removed from co-anchoring.

Callback APIs for anchor competition events

API	Description
onRequestRoomPK	A cross-room competition request was received.
onQuitRoomPK	Cross-room competition ended.

Message event callback APIs

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

Basic SDK APIs

delegate

This API is used to get callbacks for the events of TRTCLiveRoom. You can use `TRTCLiveRoomDelegate` to get the callbacks.

```
@property(nonatomic, weak)id<TRTCLiveRoomDelegate> delegate;
```

> Note :
> `delegate` is the delegate callback of `TRTCLiveRoom` .

login

This API is used to log in.

```
/// Log in to the component.
/// - Parameters:
/// - sdkAppID: You can view `SDKAppID` in **[Application Management](https://console.cloud.tencent.com/trtc/app)** > **Application Info** of the TRTC console.
/// - userID: 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: Tencent Cloud's proprietary security signature. For how to get it, please see [UserSig](https://intl.cloud.tencent.com/document/product/647/35166).
/// - config: global configuration information, which should be initialized during login and cann
```

```
ot be modified afterward. Use `isAttachedTUIKit` to specify whether to import TUIKit into your pr
oject.
/// - callback: callback for login. The return code is `0` if login is successful.
/// - Note:
/// - You are advised to set the validity period of `userSig` to 7 days to avoid cases where mess
age sending/receiving and co-anchoring fail due to expired `userSig`.
- (void)loginWithSdkAppID:(int)sdkAppID
userID:(NSString *)userID
userSig:(NSString *)userSig
config:(TRTCLiveRoomConfig *)config
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(login(sdkAppID:userID:userSig:config:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	Int	You can view `SDKAppID` in **Application Management** > **Application Info** of the TRTC console.
userID	String	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 obtain it, please see UserSig.
config	TRTCLiveRoomConfig	Global configuration information, which needs to be initialized during login and cannot be modified afterward. • `useCDNFirst` : specifies the way audience watch live streams. `true` means watching over CDNs, which is cost-efficient but has high latency. `false` means watching under the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is lower than 1 second. • `CDNPlayDomain` : specifies the domain name for CDN live streaming. It takes effect only if `useCDNFirst` is set to `true` . You can set it in **Domain Management** of the CSS console.
callback	(_ code: Int, _ message: String?) -> Void	Callback for login. The code is `0` if login is successful.

logout

This API is used to log out.

```
// Log out
/// - Parameter callback: callback for logout. The code is `0` if logout is successful.
- (void)logout:(Callback _Nullable)callback
NS_SWIFT_NAME(logout(_:));
```

The parameters are as detailed below:

Parameter	Type	Description
callback	(_ code: Int, _ message: String?) -> Void	Callback for logout. The code is `0` if logout is successful.

setSelfProfile

This API is used to set the profile.

```
/// Set user profiles. The information will be stored in Tencent Cloud IM.
/// - Parameters:
/// - name: username
/// - avatarURL: profile picture URL
/// - callback: callback for setting user profiles. The code is `0` if the operation is successfu
l.
- (void)setSelfProfileWithName:(NSString *)name
avatarURL:(NSString * _Nullable)avatarURL
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(setSelfProfile(name:avatarURL:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
name	String	Username
avatarURL	String	Profile picture URL
callback	(_ code: Int, _ message: String?) -> Void	Callback for setting user profiles. The code is `0` if the operation is successful.

Room APIs

createRoom

This API is used to create a room (called by anchor).

```
/// Create a room (called by anchor.) If the room does not exist, the system will create the room automatically.
/// The process of creating a room and starting live streaming as an anchor is as follows:
/// 1. A user calls `startCameraPreview()` to enable camera preview and set beauty filters.
/// 2. The user calls `createRoom()` to create a room, the result of which is returned via a call back.
/// 3. The user calls `starPublish()` to push streams.
/// - Parameters:
/// - roomID: room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a live room list. Currently, Tencent Cloud does not provide list management services. Please manage the list on your own.
/// - roomParam: room information, such as room name and cover information. If both the room list and room information are managed by yourself, you can ignore this parameter.
/// - callback: callback for room entry. The code is `0` if room entry is successful.
/// - Note:
/// - This API is called by an anchor to start live streaming. An anchor can create a room he or she created before.
- (void)createRoomWithRoomID:(UInt32)roomID
roomParam:(TRTCCreateRoomParam *)roomParam
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomID	UInt32	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a live room list. Currently, Tencent Cloud does not provide list management services. Please manage the list on your own.
roomParam	TRTCCreateRoomParam	Room information, such as room name and cover information. If both the room list and room information are managed by yourself, you can ignore this parameter.
callback	(_ code: Int, _ message: String?) -> Void	Callback for room creation. The code is `0` if the operation is successful.

The process of creating a room and starting live streaming as an anchor is as follows:

1. A user calls `startCameraPreview()` to enable camera preview and set beauty filters.
2. The user calls `createRoom()` to create a room, the result of which is returned via a callback.

3. The user calls `starPublish()` to push streams.

destroyRoom

This API is used to terminate a room (called by anchor).

```
/// Terminate a room (called by anchor)
/// After creating a room, an anchor can call this API to terminate it.
/// - parameter callback: callback for room termination. The code is `0` if the operation is successful.
/// - Note:
/// - After creating a room, an anchor can call this API to terminate it
- (void)destroyRoom:(Callback _Nullable)callback
NS_SWIFT_NAME(destroyRoom(callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
callback	(_ code: Int, _ message: String?) -> Void	Callback for room termination. The code is `0` if the operation is successful.

enterRoom

This API is used to enter a room (called by audience).

```
/// Enter a room (called by audience)
/// The process of entering a room and starting playback as audience is as follows:
/// 1. A user gets the latest room list from your server. The list may contain the `roomID` and other information of multiple rooms.
/// 2. The user selects a room and call `enterRoom()` to enter the room.
/// 3. If the room list managed by your server contains the `userID` of the anchor of each room, after entering the room, the user can call `startPlay(userID)` to play the anchor’s video.
/// If the room list contains `roomID` only, upon room entry, the user will receive the `onAnchorEnter(userID)` callback from `TRTCLiveRoomDelegate`.
/// The user can then call `startPlay(userID)`, passing in the `userID` obtained from the callback, to play the anchor’s video.
/// - Parameters:
/// - roomID: room ID
/// - useCDNFirst: whether to play streams via CDNs whenever possible
/// - cdnDomain: CDN domain name
/// - callback: callback for room entry. The code is `0` if room entry is successful.
/// - Note:
/// - This API is called by audience to enter a room.
/// - An anchor cannot use this API to enter a room he or she created, but must use createRoom instead.
```

```
- (void)enterRoomWithRoomID:(UInt32)roomID
useCDNFirst:(BOOL)useCDNFirst
cdnDomain:(NSString * _Nullable)cdnDomain
callback:(Callback)callback
NS_SWIFT_NAME(enterRoom(roomID:useCDNFirst:cdnDomain:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomID	UInt32	Room ID
callback	(_ code: Int, _ message: String?) -> Void	Callback for room entry. The code is `0` if the operation is successful.

The process of entering a room and starting playback as audience is as follows:

1. A user gets the latest room list from your server. The list may contain the `roomID` and other information of multiple rooms.
2. The user selects a room and calls `enterRoom()` to enter the room.
3. The user calls `startPlay(userID)`, passing in the anchor’s `userID` to start playback.

- If the room list contains anchors’ `userID`, the user can call `startPlay(userID)` to start playback.
- If the user does not have the anchor's `userID` before room entry, he or she can find it in the `onAnchorEnter(userID)` callback of `TRTCLiveRoomDelegate`, which is returned after room entry, and can then call `startPlay(userID)` to start playback.

exitRoom

This API is used to leave a room.

```
/// Leave a room (called by audience)
/// - Parameter callback: callback for room exit. The code is `0` if the operation is successful.
/// - Note:
/// - This API is called by audience to leave a room.
/// - An anchor cannot use this API to leave a room.
- (void)exitRoom:(Callback _Nullable)callback
NS_SWIFT_NAME(exitRoom(callback:));
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
callback	(_ code: Int, _ message: String?) -> Void	Callback for room exit. The code is `0` if the operation is successful.

getRoomInfos

This API is used to get room list details, which are set by anchors via `roomInfo` when they call `createRoom()` .

> Note :
> You don’t need this API if both the room list and room information are managed on your server.

```
/// Get room details
/// The information is provided by anchors via the `roomInfo` parameter when they call `createRoom()`. You don’t need this API if both the room list and room information are managed by yourself.
/// - Parameter roomIDs: list of room IDs
/// - Parameter callback: callback of room details
- (void)getRoomInfosWithRoomIDs:(NSArray<NSNumber *> *)roomIDs
callback:(RoomInfoCallback _Nullable)callback
NS_SWIFT_NAME(getRoomInfos(roomIDs:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomIDs	[UInt32]	List of room IDs
callback	(_ code: Int, _ message: String?, _ roomList: [TRTCLiveRoomInfo]) -> Void	Callback of room details

getAnchorList

This API is used to get the anchors and co-anchoring viewers in a room. It takes effect only if it is called after `enterRoom()` .

```
/// Get the anchors and co-anchoring viewers in a room. This API takes effect only if it is called after `enterRoom()`.
/// - Parameter callback: callback of user details
- (void)getAnchorList:(UserListCallback _Nullable)callback
NS_SWIFT_NAME(getAnchorList(callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
callback	(_ code: Int, _ message: String, _ userList: [TRTCLiveUserInfo]) -> Void	Callback of user details

getAudienceList

This API is used to get the information of all audience in a room. It takes effect only if it is called after `enterRoom()` .

```
/// Get the information of all audience in a room. This API takes effect only if it is called aft
er `enterRoom()`.
/// - Parameter callback: callback of user details
- (void)getAudienceList:(UserListCallback _Nullable)callback
NS_SWIFT_NAME(getAudienceList(callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
callback	(_ code: Int, _ message: String, _ userList: [TRTCLiveUserInfo]) -> Void	Callback of user details

Stream Pushing/Pulling APIs

startCameraPreview

This API is used to enable local video preview.

```
/// Enable local video preview
/// - Parameters:
/// - frontCamera: `true`: front camera; `false`: rear camera
/// - view: the control that loads video images
/// - callback: callback for the operation
- (void)startCameraPreviewWithFrontCamera:(BOOL)frontCamera
view:(UIView *)view
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(startCameraPreview(frontCamera:view:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
frontCamera	Bool	`true` : front camera; `false` : rear camera
view	UIView	The control that loads video images
callback	(_ code: Int, _ message: String?) -> Void	Callback for the operation

stopCameraPreview

This API is used to stop local video capturing and preview.

```
/// Stop local video capturing and preview
- (void)stopCameraPreview;
```

startPublish

This API is used to start live streaming (pushing streams), which can be called in the following scenarios:

- An anchor starts live streaming.
- A viewer starts co-anchoring.

```
/// Start live streaming (pushing streams). This API can be called in the following scenarios:
/// 1. An anchor starts live streaming.
/// 2. A viewer starts co-anchoring.
/// - Parameters:
/// - streamID: the `streamID` used to bind live streaming CDNs. You need to set it to the `strea
mID` of the anchor if you want audience to play the anchor’s stream via CDNs.
/// - callback: callback for the operation
- (void)startPublishWithStreamID:(NSString *)streamID
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(startPublish(streamID:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
streamID	String	The `streamID` used to bind live streaming CDNs. You need to set it to the `streamID` of the anchor if you want audience to play the anchor’s stream via CDNs.

Parameter	Type	Description
callback	(_ code: Int, _ message: String?) -> Void	Callback for the operation

stopPublish

This API is used to stop live streaming (pushing streams), which can be called in the following scenarios:

- An anchor ends live streaming.
- A viewer ends co-anchoring.

```
/// Stop live streaming (pushing streams). This API can be called in the following scenarios:
/// 1. An anchor ends live streaming.
/// 2. A viewer ends co-anchoring.
/// - Parameter callback: callback for the operation
- (void)stopPublish:(Callback _Nullable)callback
NS_SWIFT_NAME(stopPublish(callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
callback	(_ code: Int, _ message: String?) -> Void	Callback for the operation

startPlay

This API is used to play a remote video. It can be called in common playback and co-anchoring scenarios.

```
/// Play a remote video. This API can be called in common playback and co-anchoring scenarios.
/// Common playback scenario
/// 1. If the room list managed by your server contains the `userID` of the anchor of each room,
after entering a room, a user can call `startPlay(userID)` to play the anchor’ s video.
/// 2. If the room list contains `roomID` only, upon room entry, a user will receive the `onAncho
rEnter(userID)` callback from `TRTCLiveRoomDelegate`.
/// The user can then call `startPlay(userID)`, passing in the `userID` obtained from the callbac
k, to play the anchor’ s video.
/// Co-anchoring scenario
/// After co-anchoring starts, the anchor will receive the `onAnchorEnter(userID)` callback from
`TRTCLiveRoomDelegate` and can call `startPlay(userID)`, passing in the `userID` obtained from th
```

```
e callback to play the co-anchoring user’ s video.
/// - Parameters:
/// - userID: ID of the user whose video is to be played
/// - view: the control that loads video images
/// - callback: callback for the operation
- (void)startPlayWithUserID:(NSString *)userID
view:(UIView *)view
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(startPlay(userID:view:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	ID of the user whose video is to be played
view	UIView	The control that loads video images
callback	(_ code: Int, _ message: String?) -> Void	Callback for the operation

Common playback scenario

- If the room list contains anchors’ `userID` , after entering a room, a user can call `startPlay(userID)` to play the anchor's video.
- If a user does not have the anchor's `userID` before room entry, he or she can find it in the `onAnchorEnter(userID)` callback of `TRTCLiveRoomDelegate` , which is returned after room entry, and can then call `startPlay(userID)` to play the anchor’s video.

Co-anchoring scenario

After co-anchoring starts, the anchor will receive the `onAnchorEnter(userID)` callback from `TRTCLiveRoomDelegate` and can call `startPlay(userID), passing in the ` userID` obtained from the callback to play the co-anchoring user’s video.

stopPlay

This API is used to stop rendering a remote video. It needs to be called after the `onAnchorExit()` callback is received.

```
/// Stop rendering a remote video
/// - Parameters:
/// - userID: ID of the remote user
/// - callback: callback for the operation
/// - Note:
```

```
/// - Call this API after receiving the `onAnchorExit` callback.
- (void)stopPlayWithUserID:(NSString *)userID
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(stopPlay(userID:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	ID of the remote user
callback	(_ code: Int, _ message: String?) -> Void	Callback for the operation

APIs for Anchor-Audience Co-anchoring

requestJoinAnchor

This API is used to send a co-anchoring request (called by audience).

```
/// Send a co-anchoring request
/// - Parameters:
/// - reason: reason for co-anchoring
/// - responseCallback: callback of the response
/// - Note: After a viewer sends a co-anchoring request, the anchor will receive the `onRequestJoinAnchor` callback.
- (void)requestJoinAnchor:(NSString *)reason
timeout:(double)timeout
responseCallback:(ResponseCallback _Nullable)responseCallback
NS_SWIFT_NAME(requestJoinAnchor(reason:timeout:responseCallback:));
```

The parameters are as detailed below:

Parameter	Type	Description
reason	String	Reason for co-anchoring
timeout	long	Timeout period for the co-anchoring request
responseCallback	(_ agreed: Bool, _ reason: String?) -> Void	Callback of the anchor’s response

The process of co-anchoring between anchors and audience is as follows:

1. A **viewer** calls `requestJoinAnchor()` to send a co-anchoring request to the anchor.

2. The **anchor** receives the `onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate` .
3. The **anchor** calls `responseJoinAnchor()` to accept or reject the co-anchoring request.
4. The **viewer** receives the `responseCallback` callback, which carries the anchor’s response.
5. If the request is accepted, the **viewer** calls `startCameraPreview()` to enable local camera preview.
6. The **viewer** calls `startPublish()` to push streams.
7. After the viewer starts pushing streams, the **anchor** receives the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` .
8. The **anchor** calls `startPlay()` to play the co-anchoring viewer’s video.
9. If there are other viewers co-anchoring with the anchor in the room, the new co-anchoring **viewer** will receive the `onAnchorEnter()` callback and can call `startPlay()` to play other co-anchoring viewers’ video.

responseJoinAnchor

This API is used to respond to a co-anchoring request (called by anchor) after receiving the `onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate` .

```
/// Respond to a co-anchoring request
/// - Parameters:
/// - user: user ID of the viewer
/// - agree: `true`: accept; `false`: reject
/// - reason: reason for accepting/rejecting the request
/// - Note: After the anchor responds to the request, the viewer will receive the `responseCallback` passed in to `requestJoinAnchor`.
- (void)responseJoinAnchor:(NSString *)userID
agree:(BOOL)agree
reason:(NSString *)reason
NS_SWIFT_NAME(responseJoinAnchor(userID:agree:reason:));
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID of the viewer
agree	Bool	`true` : accept; `false` : reject
reason	String?	Reason for accepting/rejecting the request

kickoutJoinAnchor

This API is used to remove a user from co-anchoring (called by anchor). The removed user will receive the `onKickoutJoinAnchor()` callback of `TRTCLiveRoomDelegate` .

```
/// Remove a user from co-anchoring
/// - Parameters:
/// - userID: ID of the user to remove from co-anchoring
/// - callback: callback for the operation
/// - Note: After the anchor calls this API to remove a user from co-anchoring, the user will rec
eive the `trtcLiveRoomOnKickoutJoinAnchor()` callback.
- (void)kickoutJoinAnchor:(NSString *)userID
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(kickoutJoinAnchor(userID:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	ID of the user to remove from co-anchoring
callback	(_ code: Int, _ message: String?) -> Void	Callback for the operation

APIs for Cross-Room Anchor Competition

requestRoomPK

This API is used to request cross-room competition (called by anchor).

```
/// Request cross-room competition
/// - Parameters:
/// - roomID: room ID of the anchor to invite
/// - userID: user ID of the anchor to invite
/// - responseCallback: callback of the response
/// - Note: After a cross-room competition request is sent, the invited anchor will receive the `
onRequestRoomPK` callback.
- (void)requestRoomPKWithRoomID:(UInt32)roomID
userID:(NSString *)userID
responseCallback:(ResponseCallback _Nullable)responseCallback
NS_SWIFT_NAME(requestRoomPK(roomID:userID:responseCallback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomID	UInt32	room ID of the anchor to invite

Parameter	Type	Description
userID	String	user ID of the anchor to invite
responseCallback	(_ agreed: Bool, _ reason: String?) -> Void	Callback of the response

Anchors in different rooms can compete with each other. The process of starting cross-room competition between anchor A and anchor B is as follows:

1. **Anchor A** calls `requestRoomPK()` to send a co-anchoring request to anchor B.
2. **Anchor B** receives the `onRequestRoomPK()` callback of `TRTCLiveRoomDelegate` .
3. **Anchor B** calls `responseRoomPK()` to respond to the competition request.
4. If **anchor B** accepts the request, he or she would wait for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and call `startPlay()` to play anchor A's video.
5. **Anchor A** receives the `responseCallback` callback, which carries anchor B’s response.
6. If the request is accepted, **anchor A** waits for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and calls `startPlay()` to play anchor B’s video.

responseRoomPK

This API is used to respond to a cross-room competition request (called by anchor), after which the request sending anchor will receive the `responseCallback` passed in to `requestRoomPK` .

```
/// Respond to a cross-room competition request
/// Respond to a competition request from another anchor
/// - Parameters:
/// - user: user ID of the request sending anchor
/// - agree: `true`: accept; `false`: reject
/// - reason: reason for accepting/rejecting the request
/// - Note: After the anchor responds to the request, the request sending anchor will receive the
`responseCallback` passed in to `requestRoomPK`.
- (void)responseRoomPKWithUserID:(NSString *)userID
agree:(BOOL)agree
reason:(NSString *)reason
NS_SWIFT_NAME(responseRoomPK(userID:agree:reason:));
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID of the request sending anchor
agree	Bool	`true` : accept; `false` : reject

Parameter	Type	Description
reason	String?	Reason for accepting/rejecting the request

quitRoomPK

This API is used to quit cross-room competition. If either anchor quits cross-room competition, the other anchor will receive the `trtcLiveRoomOnQuitRoomPK()` callback of `TRTCLiveRoomDelegate` .

```
/// Quit cross-room competition
/// - Parameter callback: callback for quitting cross-room competition
// - Note: If either anchor quits cross-room competition, the other anchor will receive the `trtc
LiveRoomOnQuitRoomPK` callback.
- (void)quitRoomPK:(Callback _Nullable)callback
NS_SWIFT_NAME(quitRoomPK(callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
callback	(_ code: Int, _ message: String?) -> Void	Callback for the operation

Audio/Video APIs

switchCamera

This API is used to switch between the front and rear cameras.

```
/// Switch between the front and rear cameras
- (void)switchCamera;
```

setMirror

This API is used to set the mirror mode.

```
/// Specify whether to mirror video
/// - Parameter isMirror: enable/disable mirroring
- (void)setMirror:(BOOL)isMirror
NS_SWIFT_NAME(setMirror(isMirror:));
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
isMirror	Bool	Enable/Disable mirroring

muteLocalAudio

This API is used to mute or unmute the local user.

```
/// Mute or unmute the local user
/// - Parameter isMuted: `true`: mute; `false`: unmute
- (void)muteLocalAudio:(BOOL)isMuted
NS_SWIFT_NAME(muteLocalAudio(isMuted:));
```

The parameters are as detailed below:

Parameter	Type	Description
isMuted	Bool	`true` : mute; `false` : unmute

muteRemoteAudio

This API is used to mute or unmute a remote user.

```
/// Mute or unmute a remote user
/// - Parameters:
/// - userID: ID of the remote user
/// - isMuted: `true`: mute; `false`: unmute
- (void)muteRemoteAudioWithUserID:(NSString *)userID isMuted:(BOOL)isMuted
NS_SWIFT_NAME(muteRemoteAudio(userID:isMuted:));
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	ID of the remote user
isMuted	Bool	`true` : mute; `false` : unmute

muteAllRemoteAudio

This API is used to mute or unmute all remote users.

```
/// Mute or unmute all remote users
/// - Parameter isMuted: `true`: mute; `false`: unmute
```

```
- (void)muteAllRemoteAudio:(BOOL)isMuted
NS_SWIFT_NAME(muteAllRemoteAudio(_:));
```

The parameters are as detailed below:

Parameter	Type	Description
isMuted	Bool	`true` : mute; `false` : unmute

setAudioQuality

This API is used to set audio quality.

```
/// Set audio quality. Valid values: `1` (low), `2` (average), `3` (high)
/// - Parameter quality: audio quality
- (void)setAudioQuality:(NSInteger)quality
NS_SWIFT_NAME(setAudioiQuality(quality:));
```

The parameters are as detailed below:

Parameter	Type	Description
quality	NSInteger	`1` : speech; `2` : standard; `3` : music

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

```
/// Get the audio effect management object
- (TXAudioEffectManager *)getAudioEffectManager;
```

Beauty Filter APIs

getBeautyManager

This API is used to get the beauty filter management object TXBeautyManager.

```
/* Get the beauty filter management object TXBeautyManager
*
```

```
* You can do the following using TXBeautyManager:
* - Set the beauty filter style and apply effects including skin brightening, rosy skin, eye enla
rging, face slimming, chin slimming, chin lengthening/shortening, face shortening, nose narrowin
g, eye brightening, teeth whitening, eye bag removal, wrinkle removal, and smile line removal.
* - Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose position, lip thic
kness, and face shape.
* - Apply animated effects such as face widgets (materials).
* - Add makeup effects.
* - Recognize gestures.
*/
- (TXBeautyManager *)getBeautyManager;
```

You can do the following using `TXBeautyManager` :

- Set the beauty filter style and apply effects including skin brightening, rosy skin, eye enlarging, face slimming, chin slimming, chin lengthening/shortening, face shortening, nose narrowing, eye brightening, teeth whitening, eye bag removal, wrinkle removal, and smile line removal.
- Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose position, lip thickness, and face shape.
- Apply animated effects such as face widgets (materials).
- Add makeup effects.
- Recognize gestures.

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a room, which is generally used for on-screen comments.

```
/// Send a text message that can be seen by all users in a room
/// - Parameters:
/// - message: text message
/// - callback: callback for message sending
- (void)sendRoomTextMsg:(NSString *)message callback:(Callback _Nullable)callback
NS_SWIFT_NAME(sendRoomTextMsg(message:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
callback	(_ code: Int, _ message: String?) -> Void	Callback for message sending

sendRoomCustomMsg

This API is used to send a custom text message.

```
/// Send a custom message
/// - Parameters:
/// - command: custom command word used to distinguish between different message types
/// - message: text message
/// - callback: callback for message sending
- (void)sendRoomCustomMsgWithCommand:(NSString *)command message:(NSString *)message callback:(Ca
llback _Nullable)callback
NS_SWIFT_NAME(sendRoomCustomMsg(command:message:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
command	String	Custom command word used to distinguish between different message types
message	String	Custom text message
callback	(_ code: Int, _ message: String?) -> Void	Callback for message sending

Debugging APIs

showVideoDebugLog

This API is used to specify whether to display debugging information on the UI.

```
/// Specify whether to display debugging information on the UI
/// - Parameter isShow: show/hide debugging information
- (void)showVideoDebugLog:(BOOL)isShow
NS_SWIFT_NAME(showVideoDebugLog(_:));
```

The parameters are as detailed below:

Parameter	Type	Description
isShow	Bool	Show/Hide debugging information

`TRTCLiveRoomDelegate` Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

> 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)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onError:(NSInteger)code
message:(NSString *)message
NS_SWIFT_NAME(trtcLiveRoom(_:onError:message:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
code	Int	Error code
message	String?	Error message

onWarning

Callback for warning.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onWarning:(NSInteger)code
message:(NSString *)message
NS_SWIFT_NAME(trtcLiveRoom(_:onWarning:message:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
code	Int	TRTCWarningCode
message	String?	Warning message

onDebugLog

Callback for log.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onDebugLog:(NSString *)log
NS_SWIFT_NAME(trtcLiveRoom(_:onDebugLog:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
log	String	Log information

Room Event Callback APIs

onRoomDestroy

Callback for room termination. All users in a room will receive this callback after the anchor leaves the room.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRoomDestroy:(NSString *)roomID
NS_SWIFT_NAME(trtcLiveRoom(_:onRoomDestroy:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
roomID	String	room ID

onRoomInfoChange

Callback for change of room information. This callback is usually used to notify users of room status change in co-anchoring and anchor competition scenarios.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRoomInfoChange:(TRTCLiveRoomInfo *)info
NS_SWIFT_NAME(trtcLiveRoom(_:onRoomInfoChange:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
info	TRTCLiveRoomInfo	room information

Callback APIs for Entry/Exit of Anchors/Audience

onAnchorEnter

Callback for a new anchor/co-anchoring viewer. Audience will receive this callback when there is a new anchor/co-anchoring viewer in the room and can call `startPlay()` of `TRTCLiveRoom` to play the video of the anchor/co-anchoring viewer.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onAnchorEnter:(NSString *)userID
NS_SWIFT_NAME(trtcLiveRoom(_:onAnchorEnter:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
userID	String	User ID of the new anchor/co-anchoring viewer

onAnchorExit

Callback for the quitting of an anchor/co-anchoring viewer. The anchors and co-anchoring viewers in a room will receive this callback after an anchor/co-anchoring viewer quits competition/co-anchoring and can call `stopPlay()` of `TRTCLiveRoom` to stop playing the video of the anchor/co-anchoring viewer.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onAnchorExit:(NSString *)userID
NS_SWIFT_NAME(trtcLiveRoom(_:onAnchorExit:));
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
userID	String	User ID of the anchor/co-anchoring viewer

onAudienceEnter

Callback for room entry by a viewer.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onAudienceEnter:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onAudienceEnter:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
user	TRTCLiveUserInfo	Information of the user who entered the room

onAudienceExit

Callback for room exit by a viewer.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onAudienceExit:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onAudienceExit:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
user	TRTCLiveUserInfo	Information of the user who left the room

Callback APIs Audience-Anchor Co-anchoring Events

onRequestJoinAnchor

Callback for receiving a co-anchoring request from a viewer.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRequestJoinAnchor:(TRTCLiveUserInfo *)user
reason:(NSString * _Nullable)reason
timeout:(double)timeout
NS_SWIFT_NAME(trtcLiveRoom(_:onRequestJoinAnchor:reason:timeout:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
user	TRTCLiveUserInfo	Information of the user who requested co-anchoring
reason	String?	Reason for co-anchoring
timeout	Double	Timeout period for response from the anchor

onKickoutJoinAnchor

Callback for being removed from co-anchoring. After receiving this callback, a co-anchoring viewer needs to call `stopPublish()` of `TRTCLiveRoom` to quit co-anchoring.

```
- (void)trtcLiveRoomOnKickoutJoinAnchor:(TRTCLiveRoom *)liveRoom
NS_SWIFT_NAME(trtcLiveRoomOnKickoutJoinAnchor(_:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance

Callback APIs for Anchor Competition Events

onRequestRoomPK

Callback for receiving a cross-room competition request. If an anchor accepts the request, he or she should wait for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and call `startPlay()` to play the other anchor’s video.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRequestRoomPK:(TRTCLiveUserInfo *)user
```

```
timeout:(double)timeout
NS_SWIFT_NAME(trtcLiveRoom(_:onRequestRoomPK:timeout:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
user	TRTCLiveUserInfo	Information of the anchor who requested competition
timeout	Double	Timeout period for response from the anchor

onQuitRoomPK

Callback for ending cross-room competition.

```
- (void)trtcLiveRoomOnQuitRoomPK:(TRTCLiveRoom *)liveRoom
NS_SWIFT_NAME(trtcLiveRoomOnQuitRoomPK(_:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance

Message Event Callback APIs

onRecvRoomTextMsg

Callback for receiving a text message.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRecvRoomTextMsg:(NSString *)message
fromUser:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onRecvRoomTextMsg:fromUser:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
message	String	Text message

Parameter	Type	Description
user	TRTCLiveUserInfo	Information of the sender

onRecvRoomCustomMsg

Callback for receiving a custom message.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRecvRoomCustomMsgWithCommand:(NSString *)command
message:(NSString *)message
fromUser:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onRecvRoomCustomMsg:message:fromUser:));
```

The parameters are as detailed below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
command	String	Custom command word used to distinguish between different message types
message	String	Text message
user	TRTCLiveUserInfo	Information of the sender

TRTCLiveRoom APIs (Android)

Last updated : 2021-12-24 17:12:01

`TRTCLiveRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create a room and start live streaming, or enter a room as audience.
- The anchor and audience in a room can co-anchor with each other.
- Anchors in two rooms can compete and interact with each other.
- All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

`TRTCLiveRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Interactive Live Video Streaming (Android).

- TRTC SDK: The TRTC SDK is used as a low-latency live streaming component.
- IM SDK: The `AVChatRoom` feature of the IM SDK is used to implement chat rooms, and IM messages are used to facilitate the co-anchoring process.

`TRTCLiveRoom` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
setDelegate	Sets event callbacks.
setDelegateHandler	Sets the thread where event callbacks are.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets the profile.

Room APIs

API	Description
createRoom	Creates a room (called by anchor). If the room does not exist, the system will create the room automatically.
destroyRoom	Terminates a room (called by anchor).
enterRoom	Enters a room (called by audience).
exitRoom	Exits a room (called by audience).
getRoomInfos	Gets room list details.
getAnchorList	Gets the anchors in a room. This API works only if it is called after `enterRoom()` .
getAudienceList	Gets the information of all audience in a room. This API works only if it is called after `enterRoom()` .

Stream pushing/pulling APIs

API	Description
startCameraPreview	Enables preview of the local video.
stopCameraPreview	Stops local video capturing and preview.
startPublish	Starts live streaming (pushing streams).
stopPublish	Stops live streaming (pushing streams).
startPlay	Plays a remote video. This API can be called in common playback and co-anchoring scenarios.
stopPlay	Stops rendering a remote video.

APIs for anchor-audience co-anchoring

API	Description
requestJoinAnchor	Requests co-anchoring (called by audience).
responseJoinAnchor	Responds to a co-anchoring request (called by anchor).
kickoutJoinAnchor	Removes a user from co-anchoring (called by anchor).

APIs for cross-room anchor competition

API	Description
requestRoomPK	Requests cross-room competition (by anchor).
responseRoomPK	Responds to a cross-room competition request (called by anchor).
quitRoomPK	Quits cross-room competition.

Audio/Video APIs

API	Description
switchCamera	Switches between the front and rear cameras.
setMirror	Specifies whether to mirror video.
muteLocalAudio	Mutes/Unmutes the local user.
muteRemoteAudio	Mutes/Unmutes a remote user.
muteAllRemoteAudio	Mutes/Unmutes all remote users.

Background music and audio effect APIs

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

Beauty filter APIs

API	Description
getBeautyManager	Gets the beauty filter management object TXBeautyManager.

Message sending APIs

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

Debugging APIs

API	Description
showVideoDebugLog	Specifies whether to display debugging information on the UI.

TRTCLiveRoomDelegate API Overview

Common event callback APIs

API	Description
onError	Error
onWarning	Warning
onDebugLog	Log

Room event callback APIs

API	Description
onRoomDestroy	The room was terminated.
onRoomInfoChange	The room information changed.

Callback APIs for entry/exit of anchors/audience

API	Description
onAnchorEnter	There is a new anchor in the room.
onAnchorExit	An anchor left the room.
onAudienceEnter	A viewer entered the room.
onAudienceExit	A viewer left the room.

Callback APIs for anchor-audience co-anchoring events

API	Description
onRequestJoinAnchor	A co-anchoring request was received.

API	Description
onKickoutJoinAnchor	A user was removed from co-anchoring.

Callback APIs for anchor competition events

API	Description
onRequestRoomPK	A cross-room competition request was received.
onQuitRoomPK	Cross-room competition ended.

Message event callback APIs

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

Basic SDK APIs

sharedInstance

This API is used to get a TRTCLiveRoom singleton object.

```
public static synchronized TRTCLiveRoom sharedInstance(Context context);
```

The parameters are as detailed below:

Parameter	Type	Description
context	Context	Android context, which will be converted to `ApplicationContext` for the calling of system APIs

destroySharedInstance

This API is used to terminate a TRTCLiveRoom singleton object.

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

```
public static void destroySharedInstance();
```

setDelegate

This API is used to get callbacks for the events of TRTCLiveRoom. You can use `TRTCLiveRoomDelegate` to get the callbacks.

```
public abstract void setDelegate(TRTCLiveRoomDelegate delegate);
```

> Note :
> `setDelegate` is the delegate callback of `TRTCLiveRoom` .

setDelegateHandler

This API is used to set the thread where event callbacks are.

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

The parameters are as detailed below:

Parameter	Type	Description
handler	Handler	Callbacks for the events of `TRTCLiveRoom` are returned via this handler. Do not use it together with `setDelegate` .

login

This API is used to log in.

```
public abstract void login(int sdkAppId,
String userId, String userSig,
TRTCLiveRoomDef.TRTCLiveRoomConfig config,
TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
sdkAppId	int	You can view `SDKAppID` in **Application Management** > **Application Info** of the TRTC console.
userId	String	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 obtain it, please see UserSig.
config	TRTCLiveRoomConfig	Global configuration information, which needs to be initialized during login and cannot be modified afterward. • `useCDNFirst` : specifies the way audience watch live streams. `true` means watching over CDNs, which is cost-efficient but has high latency. `false` means watching under the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is lower than 1 second. • `CDNPlayDomain` : specifies the domain name for CDN live streaming. It takes effect only if `useCDNFirst` is set to `true` . You can set it in **Domain Management** of the CSS console.
callback	ActionCallback	Callback for login. The code is `0` if login succeeds.

logout

This API is used to log out.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for logout. The code is `0` if logout succeeds.

setSelfProfile

This API is used to set the profile.

```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCLiveRoomCallback.Actio
nCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userName	String	Username
avatar	String	Profile photo URL
callback	ActionCallback	Callback for profile setting. The code is `0` if the operation succeeds.

Room APIs

createRoom

This API is used to create a room (called by anchor).

```
public abstract void createRoom(int roomId, TRTCLiveRoomDef.TRTCCreateRoomParam roomParam, TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomId` values can be aggregated into a live room list. Currently, Tencent Cloud does not provide list management services. Please manage the list on your own.
roomParam	TRTCCreateRoomParam	Room information, such as room name and cover information. If both the room list and room information are managed on your server, you can ignore this parameter.
callback	ActionCallback	Callback for room creation. The code is `0` if the operation succeeds.

The process of creating a room and starting live streaming as an anchor is as follows:

1. A user calls `startCameraPreview()` to enable camera preview and set beauty filters.
2. The user calls `createRoom()` to create a room, the result of which is returned via the `ActionCallback` callback.
3. The user calls `startPublish()` to push streams.

destroyRoom

This API is used to terminate a room (called by anchor).

```
public abstract void destroyRoom(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

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

enterRoom

This API is used to enter a room (called by audience).

```
public abstract void enterRoom(int roomId, TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

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

The process of entering a room and starting playback as audience is as follows:

1. A user gets the latest room list from your server. The list may contain the `roomId` and other information of multiple rooms.
2. The user selects a room and calls `enterRoom()` to enter the room.
3. The user calls `startPlay(userId)` , passing in the anchor's `userId` to start playback.
    - If the room list contains the anchor's `userId` , the user can call `startPlay(userId)` to start playback.
    - If the user does not have the anchor's `userId` before room entry, he or she can find it in the `onAnchorEnter(userId)` callback of `TRTCLiveRoomDelegate` , which is returned after room entry, and can then call `startPlay(userId)` to start playback.

exitRoom

This API is used to leave a room.

```
public abstract void exitRoom(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

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

getRoomInfos

This API is used to get room list details, which are set by anchors via `roomInfo` when they call `createRoom()` .

> Note :
> You don’t need this API if both the room list and room information are managed on your server.

```
public abstract void getRoomInfos(List<Integer> roomIdList, TRTCLiveRoomCallback.RoomInfoCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomIdList	List<Integer>	Room ID list
callback	RoomInfoCallback	Callback of room details

getAnchorList

This API is used to get the anchors in a room. It takes effect only if it is called after `enterRoom()` .

```
public abstract void getAnchorList(TRTCLiveRoomCallback.UserListCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	UserListCallback	Callback of user details

getAudienceList

This API is used to get the information of all audience in a room. It takes effect only if it is called after `enterRoom()` .

```
public abstract void getAudienceList(TRTCLiveRoomCallback.UserListCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	UserListCallback	Callback of user details

Stream Pushing/Pulling APIs

startCameraPreview

This API is used to enable local video preview.

```
public abstract void startCameraPreview(boolean isFront, TXCloudVideoView view, TRTCLiveRoomCallb
ack.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
isFront	boolean	`true` : front camera; `false` : rear camera
view	TXCloudVideoView	The control that loads video images
callback	ActionCallback	Callback for the operation

stopCameraPreview

This API is used to stop local video capturing and preview.

```
public abstract void stopCameraPreview();
```

startPublish

This API is used to start live streaming (pushing streams), which can be called in the following scenarios:

- An anchor starts live streaming.
- A viewer starts co-anchoring.

```
public abstract void startPublish(String streamId, TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
streamId	String	The `streamId` used to bind live streaming CDNs. You need to set it to the `streamId` of the anchor if you want audience to play the anchor’s stream via live streaming CDNs.
callback	ActionCallback	Callback for the operation

stopPublish

This API is used to stop live streaming (push), which is suitable for the following scenarios:

- An anchor ends live streaming.
- A viewer ends co-anchoring.

```
public abstract void stopPublish(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for the operation

startPlay

This API is used to play a remote video. It can be called in common playback and co-anchoring scenarios.

```
public abstract void startPlay(String userId, TXCloudVideoView view, TRTCLiveRoomCallback.ActionC
allback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user whose video is to be played
view	TXCloudVideoView	The control that loads video images
callback	ActionCallback	Callback for the operation

Common playback scenario

- If the room list contains anchors’ `userId` , after entering a room, a user can call `startPlay(userId)` to play the anchor's video.
- If a user does not have the anchor's `userId` before room entry, he or she can find it in the `onAnchorEnter(userId)` callback of `TRTCLiveRoomDelegate` , which is returned after room entry, and can then call `startPlay(userId)` to play the anchor’s video.

Co-anchoring scenario

After co-anchoring starts, the anchor will receive the `onAnchorEnter(userId)` callback from `TRTCLiveRoomDelegate` and can call `startPlay(userId)` , passing in the `userId` obtained from the callback to play the co-anchoring user’s video.

stopPlay

This API is used to stop rendering a remote video. It needs to be called after the `onAnchorExit()` callback is received.

```
public abstract void stopPlay(String userId, TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the remote user
callback	ActionCallback	Callback for the operation

APIs for Anchor-Audience Co-anchoring

requestJoinAnchor

This API is used to send a co-anchoring request (called by audience).

```
public abstract void requestJoinAnchor(String reason, int timeout, TRTCLiveRoomCallback.ActionCal
lback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
reason	String	Reason for co-anchoring

Parameter	Type	Description
timeout	int	Timeout period
callback	ActionCallback	Callback of the anchor’s response

The process of co-anchoring between anchors and audience is as follows:

1. A **viewer** calls `requestJoinAnchor()` to send a co-anchoring request to the anchor.
2. The **anchor** receives the `onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate` .
3. The **anchor** calls `responseJoinAnchor()` to accept or reject the co-anchoring request.
4. The **viewer** receives the `responseCallback` callback, which carries the anchor’s response.
5. If the request is accepted, the **viewer** calls `startCameraPreview()` to enable local camera preview.
6. The **viewer** calls `startPublish()` to push streams.
7. After the viewer starts pushing streams, the **anchor** receives the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` .
8. The **anchor** calls `startPlay()` to play the co-anchoring viewer’s video.
9. If there are other viewers co-anchoring with the anchor in the room, the new co-anchoring **viewer** will receive the `onAnchorEnter()` callback and can call `startPlay()` to play other co-anchoring viewers’ video.

responseJoinAnchor

This API is used to respond to a co-anchoring request (called by anchor) after receiving the `onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate` .

```
public abstract void responseJoinAnchor(String userId, boolean agree, String reason);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the viewer
agree	boolean	`true` : accept; `false` : reject
reason	String	Reason for accepting/rejecting the request

kickoutJoinAnchor

This API is used to remove a user from co-anchoring (called by anchor). The removed user will receive the `onKickoutJoinAnchor()` callback of `TRTCLiveRoomDelegate` .

```
public abstract void kickoutJoinAnchor(String userId, TRTCLiveRoomCallback.ActionCallback callbac
k);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the co-anchoring user
callback	ActionCallback	Callback for the operation

APIs for Cross-Room Anchor Competition

requestRoomPK

This API is used to request cross-room competition (called by anchor).

```
public abstract void requestRoomPK(int roomId, String userId, TRTCLiveRoomCallback.ActionCallback
callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID of the anchor to call
userId	String	User ID of the anchor to call
callback	ActionCallback	Callback for requesting cross-room competition

Anchors in different rooms can compete with each other. The process of starting cross-room competition between anchor A and anchor B is as follows:

1. **Anchor A** calls `requestRoomPK()` to send a competition request to anchor B.
2. **Anchor B** receives the `onRequestRoomPK()` callback of `TRTCLiveRoomDelegate` .
3. **Anchor B** calls `responseRoomPK()` to respond to the competition request.
4. If **anchor B** accepts the request, he or she would wait for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and call `startPlay()` to play anchor A's video.
5. **Anchor A** receives the `responseCallback` callback, which carries anchor B’s response.
6. If the request is accepted, **anchor A** waits for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and calls `startPlay()` to play anchor B’s video.

responseRoomPK

This API is used to respond to a cross-room competition request (called by anchor), after which the request sending anchor will receive the `responseCallback` passed in to `requestRoomPK` .

```
public abstract void responseRoomPK(String userId, boolean agree, String reason);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the request sending anchor
agree	boolean	`true` : accept; `false` : reject
reason	String	Reason for accepting/rejecting the request

quitRoomPK

This API is used to quit cross-room competition. If either anchor quits cross-room competition, the other anchor will receive the `onQuitRoomPk()` callback of `TRTCLiveRoomDelegate` .

```
public abstract void quitRoomPK(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for the operation

Audio/Video APIs

switchCamera

This API is used to switch between the front and rear cameras.

```
public abstract void switchCamera();
```

setMirror

This API is used to set the mirror mode.

```
public abstract void setMirror(boolean isMirror);
```

The parameters are as detailed below:

Parameter	Type	Description
isMirror	boolean	Enable/Disable mirroring

muteLocalAudio

This API is used to mute or unmute the local user.

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

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	`true` : mute; `false` : unmute

muteRemoteAudio

This API is used to mute or unmute a remote user.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the remote user
mute	boolean	`true` : mute; `false` : unmute

muteAllRemoteAudio

This API is used to mute or unmute all remote users.

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

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	`true` : mute; `false` : unmute

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

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

Beauty Filter APIs

getBeautyManager

This API is used to get the beauty filter management object TXBeautyManager.

```
public abstract TXBeautyManager getBeautyManager();
```

You can do the following using `TXBeautyManager` :

- Set the beauty filter style and apply effects including skin brightening, rosy skin, eye enlarging, face slimming, chin slimming, chin lengthening/shortening, face shortening, nose narrowing, eye brightening, teeth whitening, eye bag removal, wrinkle removal, and smile line removal.
- Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose position, lip thickness, and face shape.
- Apply animated effects such as face widgets (materials).
- Add makeup effects.
- Recognize gestures.

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a room, which is generally used for on-screen comments.

```
public abstract void sendRoomTextMsg(String message, TRTCLiveRoomCallback.ActionCallback callbac
k);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
callback	ActionCallback	Callback for message sending

sendRoomCustomMsg

This API is used to send a custom text message.

```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCLiveRoomCallback.ActionCal
lback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message
callback	ActionCallback	Callback for message sending

Debugging APIs

showVideoDebugLog

This API is used to specify whether to display debugging information on the UI.

```
public abstract void showVideoDebugLog(boolean isShow);
```

The parameters are as detailed below:

Parameter	Type	Description
isShow	boolean	Show/Hide debugging information

`TRTCLiveRoomDelegate` Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

> 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 as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Error message

onWarning

Callback for warning.

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

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Warning message

onDebugLog

Callback for log.

```
void onDebugLog(String message);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
message	String	Log information

Room Event Callback APIs

onRoomDestroy

Callback for room termination. All users in a room will receive this callback after the anchor leaves the room.

```
void onRoomDestroy(String roomId);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	String	Room ID

onRoomInfoChange

Callback for change of room information. This callback is usually used to notify users of room status change in co-anchoring and anchor competition scenarios.

```
void onRoomInfoChange(TRTCLiveRoomDef.TRTCLiveRoomInfo roomInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
roomInfo	TRTCLiveRoomInfo	Room information

Callback APIs for Entry/Exit of Anchors/Audience

onAnchorEnter

Callback for a new anchor/co-anchoring viewer. Audience will receive this callback when there is a new anchor/co-anchoring viewer in the room and can call `startPlay()` of `TRTCLiveRoom` to play the video of the anchor/co-anchoring viewer.

```
void onAnchorEnter(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the new anchor/co-anchoring viewer

onAnchorExit

Callback for the quitting of an anchor. The anchors (including co-anchoring viewers) in a room will receive this callback after an anchor/co-anchoring viewer quits competition/co-anchoring and can call `stopPlay()` of `TRTCLiveRoom` to stop playing the video of the anchor/co-anchoring viewer.

```
void onAnchorExit(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user who quit

onAudienceEnter

Callback for room entry by a viewer.

```
void onAudienceEnter(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	TRTCLiveUserInfo	Information of the viewer who entered the room

onAudienceExit

Callback for room exit by a viewer.

```
void onAudienceExit(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	TRTCLiveUserInfo	Information of the viewer who left the room

Callback APIs for Anchor-Audience Co-anchoring Events

onRequestJoinAnchor

Callback for receiving a co-anchoring request from a viewer.

```
void onRequestJoinAnchor(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, String reason, int timeOut);
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	TRTCLiveUserInfo	Information of the viewer who requested co-anchoring
reason	String	Reason for co-anchoring
timeout	int	Timeout period for response from the anchor. If the anchor does not respond to the request within the period, it will be discarded automatically.

onKickoutJoinAnchor

Callback for being removed from co-anchoring. A co-anchoring viewer needs to call `stopPublish()` of `TRTCLiveRoom` to quit co-anchoring after receiving this callback.

```
void onKickoutJoinAnchor();
```

Callback APIs for Anchor Competition Events

onRequestRoomPK

Callback for receiving a cross-room competition request. If an anchor accepts the request, he or she should wait for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and call `startPlay()` to play the other anchor's video.

```
void onRequestRoomPK(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, int timeout);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userInfo	TRTCLiveUserInfo	Information of the anchor who requested cross-room competition
timeout	int	Timeout period for response from the anchor

onQuitRoomPK

Callback for ending cross-room competition.

```
void onQuitRoomPK();
```

Message Event Callback APIs

onRecvRoomTextMsg

Callback for receiving a text message.

```
void onRecvRoomTextMsg(String message, TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
userInfo	TRTCLiveUserInfo	Information of the sender

onRecvRoomCustomMsg

Callback for receiving a custom message.

```
void onRecvRoomCustomMsg(String cmd, String message, TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
command	String	Custom command word used to distinguish between different message types
message	String	Text message

Parameter	Type	Description
userInfo	TRTCLiveUserInfo	Information of the sender

TRTCAudioEffectManager

playBGM

This API is used to play back background music.

```
void playBGM(String url, int loopTimes, int bgmVol, int micVol, TRTCCloud.BGMNotify notify);
```

The parameters are as detailed below:

Parameter	Type	Description
url	String	Path of the music file
loopTimes	int	Loop times
bgmVol	int	Volume of background music
micVol	int	Audio capturing volume
notify	TRTCCloud.BGMNotify	Playback notification

stopBGM

This API is used to stop background music playback.

```
void stopBGM();
```

pauseBGM

This API is used to pause background music playback.

```
void pauseBGM();
```

resumeBGM

This API is used to resume background music playback.

```
void resumeBGM();
```

setBGMVolume

This API is used to set the playback volume of background music.

```
void setBGMVolume(int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Volume. Value range: 0-100. Default value: `100`

setBGMPosition

This API is used to set the background music playback progress.

```
int setBGMPosition(int position);
```

The parameters are as detailed below:

Parameter	Type	Description
position	int	Playback progress of background music in milliseconds (ms)

Return code

`0` : successful

setMicVolume

This API is used to set the mic volume. It can be used to control the volume of the mic when background music is mixed.

```
void setMicVolume(int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
volume	Int	Volume. Value range: 0-100. Default value: `100`

setReverbType

This API is used to set the reverb effect.

```
void setReverbType(int reverbType);
```

The parameters are as detailed below:

Parameter	Type	Description
reverbType	int	Reverb effect. For details, please see the definitions of TRTC_REVERB_TYPE in `TRTCCloudDef` .

setVoiceChangerType

This API is used to set the voice changing effect.

```
void setVoiceChangerType(int type);
```

The parameters are as detailed below:

Parameter	Type	Description
type	int	Voice changing effect. For details, please see the definitions of TRTC_VOICE_CHANGER_TYPE in `TRTCCloudDef` .

playAudioEffect

This API is used to play an audio effect. For each audio effect, you need to assign an ID, which is used to start and stop the playback of an audio effect as well as set its playback volume. Supported formats include AAC, MP3, and M4A.

```
void playAudioEffect(int effectId, String path, int count, boolean publish, int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
effectId	int	Audio effect ID
path	String	Audio effect path
count	int	Loop times
publish	boolean	Whether to push the audio effect. `true` : push the effect to remote users; `false` : preview the effect locally only
volume	int	Volume. Value range: 0-100. Default value: `100`

pauseAudioEffect

This API is used to pause playing an audio effect.

```
void pauseAudioEffect(int effectId);
```

The parameters are as detailed below:

Parameter	Type	Description
effectId	int	Audio effect ID

resumeAudioEffect

This API is used to resume playing an audio effect.

```
void resumeAudioEffect(int effectId);
```

The parameters are as detailed below:

Parameter	Type	Description
effectId	int	Audio effect ID

stopAudioEffect

This API is used to stop playing an audio effect.

```
void stopAudioEffect(int effectId);
```

The parameters are as detailed below:

Parameter	Type	Description
effectId	int	Audio effect ID

stopAllAudioEffects

This API is used to stop playing all audio effects.

```
void stopAllAudioEffects();
```

setAudioEffectVolume

This API is used to set the volume of an audio effect.

```
void setAudioEffectVolume(int effectId, int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
effectId	int	Audio effect ID
volume	int	Volume. Value range: 0-100. Default value: `100`

setAllAudioEffectsVolume

This API is used to set the volume of all audio effects.

```
void setAllAudioEffectsVolume(int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Volume. Value range: 0-100. Default value: `100`

TRTCLiveRoom APIs (Flutter)

Last updated : 2022-02-11 16:17:52

`TRTCLiveRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create a room and start live streaming, or enter a room as audience.
- The anchor and audience in a room can co-anchor with each other.
- Anchors in two rooms can compete and interact with each other.
- All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

`TRTCLiveRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Interactive Live Video Streaming (Flutter).

- TRTC SDK: the TRTC SDK is used as the low-latency live streaming component.
- IM SDK: the `AVChatRoom` feature of the IM SDK is used to implement chat rooms, and IM messages are used to facilitate the co-anchoring process between anchors.

`TRTCLiveRoom` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
registerListener	Registers an event listener.
unRegisterListener	Removes an event listener.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates a room (called by anchor). If the room does not exist, the system will automatically create a room.
destroyRoom	Closes a room (called by anchor).
enterRoom	Enters a room (called by audience).
exitRoom	Exits a room (called by audience).
getRoomInfos	Gets room list details.
getAnchorList	Gets the anchor and co-anchors in a room. This API works only if it is called after `enterRoom()` .
getRoomMemberList	Gets the list of all users in a room. This API works only if it is called after `enterRoom()` .

Stream pushing/pulling APIs

API	Description
startCameraPreview	Enables preview of local video.
stopCameraPreview	Stops local video capturing and preview.
startPublish	Starts live streaming (pushing streams).
stopPublish	Stops live streaming (pushing streams).
startPlay	Plays a remote video. This API can be called in common playback and co-anchoring scenarios.
stopPlay	Stops rendering a remote video.

APIs for co-anchoring between anchors and audience

API	Description
requestJoinAnchor	Requests co-anchoring (called by audience).
responseJoinAnchor	Responds to a co-anchoring request (called by anchor).
kickoutJoinAnchor	Removes a user from co-anchoring (called by anchor).

APIs for cross-room anchor competition

API	Description
requestRoomPK	Requests cross-room competition (called by anchor).
responseRoomPK	Responds to a cross-room competition request (called by anchor).
quitRoomPK	Quits cross-room competition.

Audio/Video APIs

API	Description
switchCamera	Switches between the front and rear cameras.
setMirror	Sets the mirror mode.
muteLocalAudio	Mutes/Unmutes the local user.
muteRemoteAudio	Mutes/Unmutes a remote user.
muteAllRemoteAudio	Mutes/Unmutes all remote users.

Background music and audio effect APIs

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

Beauty filter APIs

API	Description
getBeautyManager	Gets the beauty filter management object TXBeautyManager.

Message sending APIs

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

TRTCLiveRoomDelegate API Overview

Common event callback APIs

API	Description
onError	Error
onWarning	Warning
onKickedOffline	You were kicked offline because another user logged in to the account.

Room event callback APIs

API	Description
onEnterRoom	You entered the room.
onUserVideoAvailable	Whether a remote user has playable video in the primary stream (usually used for camera images)
onRoomDestroy	The room was closed.

Callback APIs for anchors and audience

API	Description
onAnchorEnter	An anchor entered the room.
onAnchorExit	An anchor left the room.
onAudienceEnter	A viewer entered the room.
onAudienceExit	A viewer left the room.

Callback APIs for co-anchoring between anchors and audience

API	Description
onRequestJoinAnchor	A co-anchoring request was received.
onAnchorAccepted	The co-anchoring request was accepted.
onAnchorRejected	The co-anchoring request was rejected.

API	Description
onKickoutJoinAnchor	A user was removed from co-anchoring.

Anchor competition callback APIs

API	Description
onRequestRoomPK	A cross-room competition request was received.
onRoomPKAccepted	The cross-room competition request was accepted.
onRoomPKRejected	The cross-room competition request was rejected.
onQuitRoomPK	The cross-room competition ended.

Message event callback APIs

API	Description
onRecvRoomTextMsg	Receipt of a text message
onRecvRoomCustomMsg	Receipt of a custom message

Basic SDK APIs

sharedInstance

This API is used to get a TRTCLiveRoom singleton object.

```
static Future<TRTCLiveRoom> sharedInstance()
```

destroySharedInstance

This API is used to terminate a TRTCLiveRoom singleton object.

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

```
static void destroySharedInstance()
```

registerListener

This API is used to get the event callback of TRTCLiveRoom. You can use `TRTCLiveRoomDelegate` to get various status notifications of TRTCLiveRoom.

```
void registerListener(VoiceListenerFunc func);
```

> Note :
> `registerListener` is the delegate callback of `TRTCLiveRoom` .

unRegisterListener

This API is used to remove the event listener.

```
void unRegisterListener(VoiceListenerFunc func);
```

login

This API is used to log in.

```
Future<ActionCallback> login(
int sdkAppId, String userId, String userSig, TRTCLiveRoomConfig config);
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	int	You can view the `SDKAppID` via **Application Management** > **Application Info** in the TRTC console.
userId	String	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 obtain it, please see UserSig.

Parameter	Type	Description
config	TRTCLiveRoomConfig	Global configuration information, which needs to be initialized during login and cannot be modified afterwards. • `useCDNFirst` : specifies the way audience watch live streams. `true` means watching over CDNs, which is cost-efficient but has high latency. `false` means watching under the low latency mode, the cost of which is between that of CDN live streaming and co-anchoring, but the latency is lower than 1 second. • `CDNPlayDomain` : specifies the domain name for CDN live streaming. It takes effect only if `useCDNFirst` is set to `true` . You can set it in **Domain Management** of the CSS console.

logout

This API is used to log out.

```
Future<ActionCallback> logout();
```

setSelfProfile

This API is used to set profile.

```
Future<ActionCallback> setSelfProfile(String userName, String avatarURL);
```

The parameters are as detailed below:

Parameter	Type	Description
userName	String	Nickname
avatarURL	String	Profile photo address

Room APIs

createRoom

This API is used to create a room (called by anchor).

```
Future<ActionCallback> createRoom(int roomId, TRTCCreateRoomParam roomParam);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomId` values can be aggregated into a live room list. Currently, Tencent Cloud does not provide list management services. Please manage the list on your own.
roomParam	RoomParam	Room information, such as room name and cover information. If both the room list and room information are managed on your server, you can ignore this parameter.

The process of creating a room and starting live streaming as an anchor is as follows:

1. A user calls `startCameraPreview()` to enable camera preview and set beauty filters.
2. The user calls `createRoom()` to create a room, the result of which is returned via the `ActionCallback` callback.
3. The user calls `starPublish()` to push streams.

destroyRoom

This API is used to close a room (called by anchor).

```
Future<ActionCallback> destroyRoom();
```

enterRoom

This API is used to enter a room (called by audience).

```
Future<ActionCallback> enterRoom(int roomId);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID

The process of entering a room and starting playback as audience is as follows:

1. A user gets the latest room list from your server. The list may contain `roomId` and other room information of multiple rooms.
2. The user selects a room and calls `enterRoom()` to enter the room.

3. The user calls `startPlay(userId)` , passing in the anchor's `userId` to start playback.
   - If the room list contains the anchor's `userId` , the user can call `startPlay(userId)` to start playback.
   - If the user does not have the anchor's `userId` before room entry, he or she can find it in the `onAnchorEnter(userId)` callback of `TRTCLiveRoomDelegate` , which is returned after room entry. The user can then call `startPlay(userId)` to start playback.

exitRoom

This API is used to leave a room.

```
Future<ActionCallback> exitRoom();
```

getRoomInfos

This API is used to get room list details, which are set by anchors via `roomInfo` when they call `createRoom()` .

> Note :
> You don't need this API if both the room list and room information are managed on your server.

```
Future<RoomInfoCallback> getRoomInfos(List<String> roomIdList);
```

The parameters are as detailed below:

Parameter	Type	Description
roomIdList	List<String>	Room ID list

getAnchorList

This API is used to get the anchor and co-anchors in a room. It takes effect only if it is called after `enterRoom()` .

```
Future<UserListCallback> getAnchorList();
```

getRoomMemberList

This API is used to get the information of all audience in the room. It takes effect only if it is called after `enterRoom()` .

```
Future<UserListCallback> getRoomMemberList(int nextSeq)
```

The parameters are as detailed below:

Parameter	Type	Description
nextSeq	int	Pulling-by-page flag. It is set to 0 when the information is pulled for the first time. If the callback succeeds and `nextSeq` is not 0, pagination is needed. The value of this field is passed in for the next pull until the value becomes 0.

Stream Pushing/Pulling APIs

startCameraPreview

This API is used to enable local video preview.

```
Future<void> startCameraPreview(bool isFrontCamera, int viewId);
```

The parameters are as detailed below:

Parameter	Type	Description
isFrontCamera	bool	`true` : turns the front camera on; `false` : turns the rear camera on.
viewId	int	Called back video view ID

stopCameraPreview

This API is used to stop local video capturing and preview.

```
Future<void> stopCameraPreview();
```

startPublish

This API is used to start live streaming (pushing streams), which can be called in the following scenarios:

- An anchor starts live streaming.
- A viewer starts co-anchoring.

```
Future<void> startPublish(String streamId);
```

The parameters are as detailed below:

Parameter	Type	Description
streamId	String?	The `streamId` used to bind live streaming CDNs. You need to specify the `streamId` of the anchor if you want audience to play the anchor's streams via live streaming CDNs.

stopPublish

This API is used to stop live streaming (pushing streams), which can be called in the following scenarios:

- An anchor ends live streaming.
- A viewer ends co-anchoring.

```
Future<void> stopPublish();
```

startPlay

This API is used to play a remote video. It can be called in common playback and co-anchoring scenarios.

```
Future<void> startPlay(String userId, int viewId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user whose video is to be played
viewId	int	Called back video view ID

- **Common playback:**
  - If the room list contains the anchor's `userId` , audience can call `startPlay(userId)` to play the anchor's video after `enterRoom()` .
  - If audience do not have the anchor's `userId` before room entry, they can find it in the `onAnchorEnter(userId)` callback of `TRTCLiveRoomDelegate` , which is returned after room entry. They can then call `startPlay(userId)` to play the anchor's video.

- **Co-anchoring:**
  After co-anchoring is initiated, the anchor will receive the `onAnchorEnter(userId)` callback from `TRTCLiveRoomDelegate` and can call `startPlay(userId), passing in the` userId` returned by the callback to play co-anchoring video.

stopPlay

This API is used to stop rendering remote video. It needs to be called when the `onAnchorExit()` callback is received.

```
Future<void> stopPlay(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the remote user

APIs for Co-anchoring Between Anchor and Audience

requestJoinAnchor

This API is used by audience to send a co-anchoring request.

```
Future<ActionCallback> requestJoinAnchor();
```

The process of co-anchoring between anchor and audience is as follows:

1. A **viewer** calls `requestJoinAnchor()` to send a co-anchoring request to the anchor.
2. The **anchor** receives the `onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate` .
3. The **anchor** calls `responseJoinAnchor()` to accept or reject the co-anchoring request.
4. The **viewer** receives the `responseCallback` callback, which carries the anchor's response.
5. If the request is accepted, the **viewer** calls `startCameraPreview()` to enable local camera preview.
6. The **viewer** calls `startPublish()` to push streams.
7. After the viewer starts pushing streams, the **anchor** receives the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` .
8. The **anchor** calls `startPlay()` to play the co-anchoring viewer's video.
9. If there are other viewers co-anchoring with the anchor in the room, the new co-anchoring **viewer** will receive the `onAnchorEnter()` callback and can call `startPlay()` to play other co-anchoring viewers' video.

responseJoinAnchor

This API is used by anchors to respond to a co-anchoring request after receiving the `onRequestJoinAnchor()` callback of `TRTCLiveRoomDelegate`.

```
Future<ActionCallback> responseJoinAnchor(String userId, boolean agreee);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the viewer
agree	bool	`true` : accepts; `false` : rejects.

kickoutJoinAnchor

This API is used by anchors to remove a user from co-anchoring. After this API is called, the removed user will receive the `onKickoutJoinAnchor()` callback of `TRTCLiveRoomDelegate`.

```
Future<ActionCallback> kickoutJoinAnchor(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the co-anchoring user

APIs for Cross-Room Anchor Competition

requestRoomPK

This API is used by anchors to request cross-room competition.

```
Future<ActionCallback> requestRoomPK(int roomId, String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID of the anchor to call
userId	String	User ID of the anchor to call

Two anchors in different rooms can compete with each other. The process is as follows:

1. **Anchor A** calls `requestRoomPK()` to send a co-anchoring request to anchor B.
2. **Anchor B** receives the `onRequestRoomPK()` callback of `TRTCLiveRoomDelegate` .
3. **Anchor B** calls `responseRoomPK()` to respond to the competition request.
4. After accepting the request, **anchor B** waits for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and calls `startPlay()` to play anchor A's video.
5. **Anchor A** receives the `onRoomPKAccepted` or `onRoomPKRejected` callback.
6. If the request is accepted, **anchor A** waits for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and calls `startPlay()` to play anchor B's video.

responseRoomPK

This API is used by anchors to respond to a cross-room competition request, after which the request sending anchor will receive the `responseCallback` callback, which carries the response passed in to `requestRoomPK` .

```
Future<ActionCallback> responseRoomPK(String userId, boolean agree);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the anchor sending the competition request
agree	bool	`true` : accepts; `false` : rejects.

quitRoomPK

This API is used to quit cross-room competition. If either anchor quits cross-room competition, the other anchor will receive the `onQuitRoomPk()` callback of `TRTCLiveRoomDelegate` .

```
Future<ActionCallback> quitRoomPK();
```

Audio/Video APIs

switchCamera

This API is used to switch between the front and rear cameras.

```
Future<void> switchCamera(boolean isFrontCamera);
```

setMirror

This API is used to set the mirror mode.

```
Future<void> setMirror(boolean isMirror);
```

The parameters are as detailed below:

Parameter	Type	Description
isMirror	bool	Enables/Disables mirroring.

muteLocalAudio

This API is used to mute or unmute the local user.

```
Future<void> muteLocalAudio(boolean mute);
```

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	`true` : mute; `false` : unmute

muteRemoteAudio

This API is used to enable the hands-free mode.

```
Future<void> muteRemoteAudio(String userId, boolean mute);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the remote user
mute	boolean	`true` : mute; `false` : unmute

muteAllRemoteAudio

This API is used to mute or unmute all remote users.

```
Future<void> muteAllRemoteAudio(boolean mute);
```

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	`true` : mute; `false` : unmute

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

```
getAudioEffectManager();
```

Beauty Filter APIs

getBeautyManager

This API is used to get the beauty filter management object TXBeautyManager.

```
getBeautyManager();
```

You can do the following using `TXBeautyManager` :

- Set the beauty filter style and apply effects including skin brightening, rosy skin, eye enlarging, face slimming, chin slimming, chin lengthening/shortening, face shortening, nose narrowing, eye brightening, teeth whitening, eye bag removal, wrinkle removal, and smile line removal.
- Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose position, lip thickness, and face shape.
- Apply animated effects such as face widgets (materials).
- Add makeup effects.
- Recognize gestures.

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a room, which is generally used for on-screen comments.

```
Future<ActionCallback> sendRoomTextMsg(String message);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message

sendRoomCustomMsg

This API is used to send a custom text message.

```
Future<ActionCallback> sendRoomCustomMsg(String cmd, String message);
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message

`TRTCLiveRoomDelegate` Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

> 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.

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
errCode	int	Error code
errMsg	String	Error message

onWarning

Callback for warning.

The parameters are as detailed below:

Parameter	Type	Description
warningCode	int	Warning code
warningMsg	String	Warning message

onKickedOffline

Callback for being kicked offline because another user logged in to the same account.

Room Event Callback APIs

onRoomDestroy

Callback for room closing. All users in the room will receive this callback after the anchor leaves the room.

onEnterRoom

Callback for room entry by the local user.

The parameters are as detailed below:

Parameter	Type	Description
result	int	If `result` is greater than 0, it indicates the time (ms) room entry took; if `result` is less than 0, it represents the error code for room entry.

onUserVideoAvailable

Callback for whether a remote user has playable video in the primary stream (usually used for camera images).

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
available	boolean	Whether the user's video is enabled

Callback APIs for Anchors and Audience

onAnchorEnter

Callback for a new anchor/co-anchoring viewer. Audience will receive this callback when there is a new anchor/co-anchoring viewer in the room and can call `startPlay()` of `TRTCLiveRoom` to play the video of the anchor/co-anchoring viewer.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the new anchor/co-anchoring viewer
userName	String	Username
userAvatar	String	Profile photo address

onAnchorExit

Callback for the quitting of a competing anchor/co-anchoring viewer. The anchor and co-anchoring viewers in a room will receive this callback after a competing anchor/co-anchoring viewer quits co-anchoring and can call `stopPlay()` of `TRTCLiveRoom` to stop playing the video of the competing anchor/co-anchoring viewer.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user who quitted co-anchoring
userName	String	Username
userAvatar	String	Profile photo address

onAudienceEnter

Callback for room entry by a viewer.

```
void onAudienceEnter(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	TRTCLiveRoomDef.TRTCLiveUserInfo	Information of the viewer who enters the room, such as user ID, nickname, and profile photo.

onAudienceExit

Callback for room exit by a viewer.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the viewer
userName	String	Username
userAvatar	String	Profile photo address

onRequestJoinAnchor

Callback for receiving a co-anchoring request from a viewer.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the request sending user
userName	String	Username
userAvatar	String	Profile photo address

onAnchorAccepted

Callback for accepting a co-anchoring request.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the anchor

onAnchorRejected

Callback for rejecting a co-anchoring request.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the anchor

onKickoutJoinAnchor

Callback for being removed from co-anchoring. A co-anchoring viewer needs to call `stopPublish()` of `TRTCLiveRoom` to quit co-anchoring after receiving this callback.

Anchor Competition Callback APIs

onRequestRoomPK

Callback for receiving a cross-room competition request. If an anchor accepts the request after receiving this callback, he or she should wait for the `onAnchorEnter()` callback of `TRTCLiveRoomDelegate` and call `startPlay()` to play the other anchor's video.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the request sending anchor
userName	String	Username
userAvatar	String	Profile photo address

onRoomPKAccepted

Callback for accepting a cross-room competition request.

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	String	User ID of the anchor who accepted the request

onRoomPKRejected

Callback for rejecting a cross-room competition request.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the anchor who rejected the request

onQuitRoomPK

Callback for ending cross-room competition.

Message Event Callback APIs

onRecvRoomTextMsg

A text message was received.

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message

onRecvRoomCustomMsg

A custom message was received.

The parameters are as detailed below:

Parameter	Type	Description
command	String	Custom command word used to distinguish between different message types
message	String	Text message

Audio Chat Room

Audio Chat Room (iOS)

Last updated : 2022-03-18 22:58:12

Demo UI

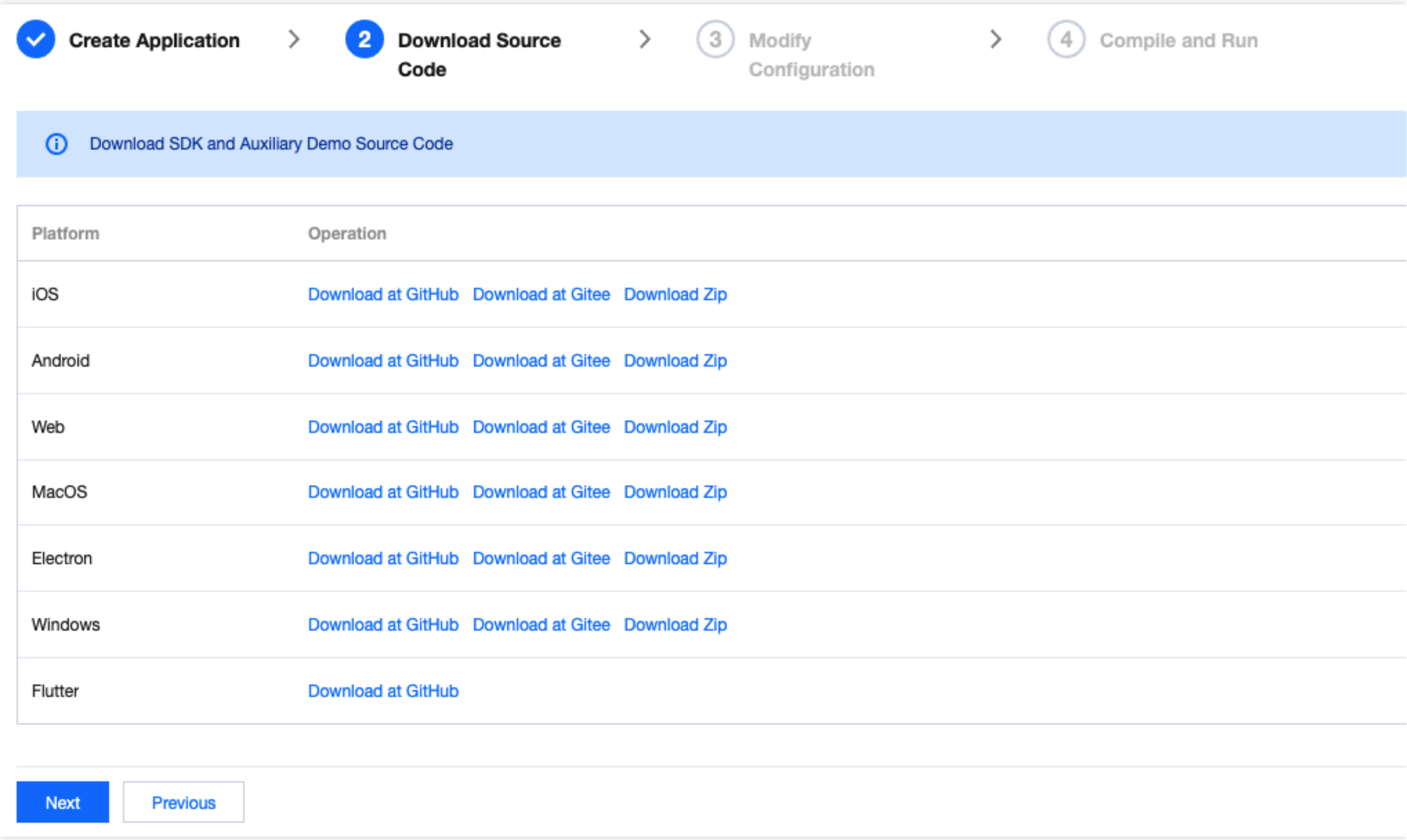
You can download and install the app we provide to try out TRTC's audio chat room features, including seat management, low-latency audio interaction, text chat, etc.

To quickly implement the audio chat room feature, you can modify the app we provide and adapt it to your needs. You can also use the `TUIVoiceRoom` component and customize your own UI.

Using the App's UI

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestVoiceRoom` and click **Create**.
3. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

> Note :
> The voice chat room feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Value-added Service Pricing for its billing details.

Step 2. Download the application source code

Clone or download the TUIVoiceRoom source code.

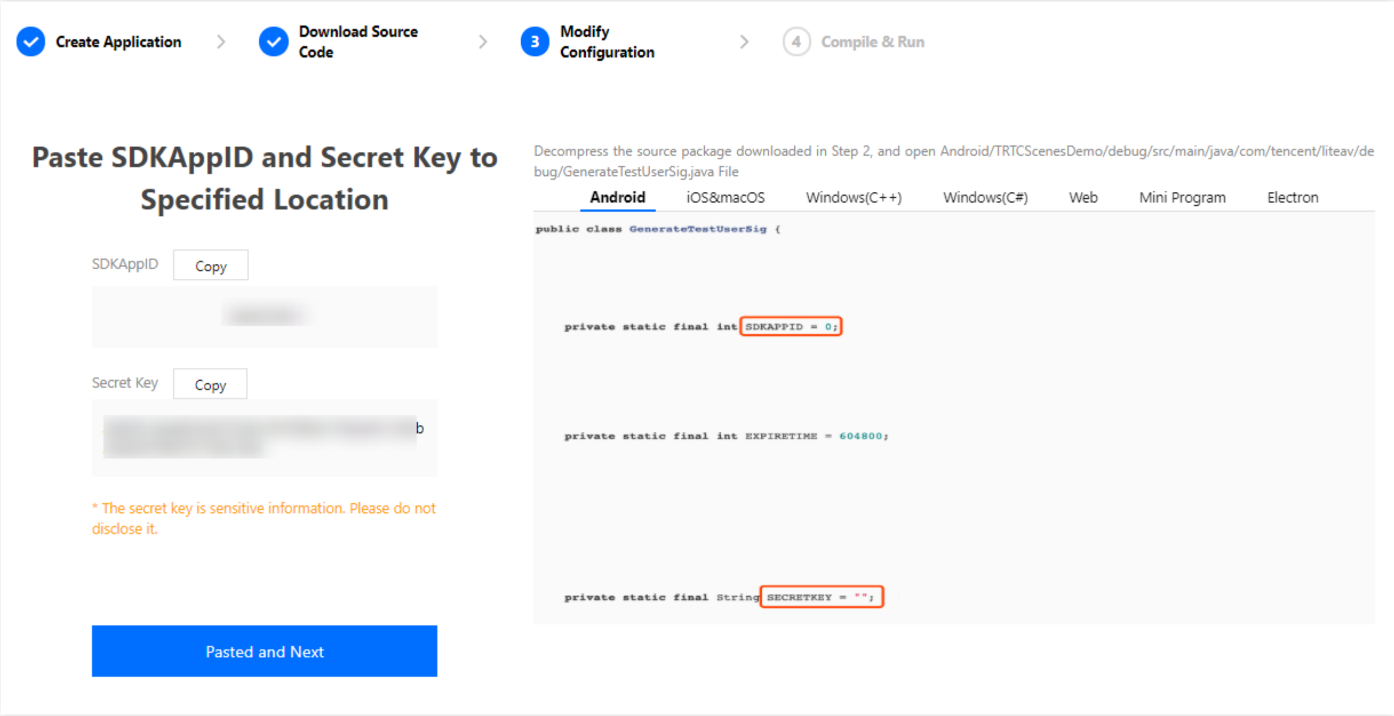
Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `TUIVoiceRoom/Debug/GenerateTestUserSig.swift` .

3. Set the following parameters in `GenerateTestUserSig.swift` :

   - SDKAPPID: `0` by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: left empty by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Pasted and Next

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

Note：

- The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
- The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open `TUIVoiceRoom/TUIVoiceRoomApp.xcworkspace` with Xcode (11.0 or above) and click the run button.

Step 5. Modify the app’s source code

The `Source` folder in the source code contains two subfolders: `ui` and `model` . The `ui` subfolder contains UI code and UI-related logic. The table below lists the Swift files (folders) and the UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
TRTCVoiceRoomEnteryControl.swift	The initialization method for all view controllers. You can use the instance to quickly get a `ViewController` object.
TRTCCreateVoiceRoomViewController	Logic for the room creation view
TRTCVoiceRoomViewController	Logic for the main room views for room owners and listeners

Each `TRTC’XXXX’ViewController` folder contains `ViewController` , `RootView` , and `ViewModel` , whose use is described below.

File	Description
ViewController.swift	Page controller, which is responsible for routing pages and binding `RootView` and `ViewModel`
RootView.swift	Layout of all views
ViewModel.swift	View controller, which is responsible for responding to users’ interactions with views and returning response status

Tryout

Note :
You need at least two devices to try out the application.

User A

1. Enter a username (**which must be unique**) and log in.
2. Click **Create Room**.
3. Type a subject for the conference and tap **Let’s go**.

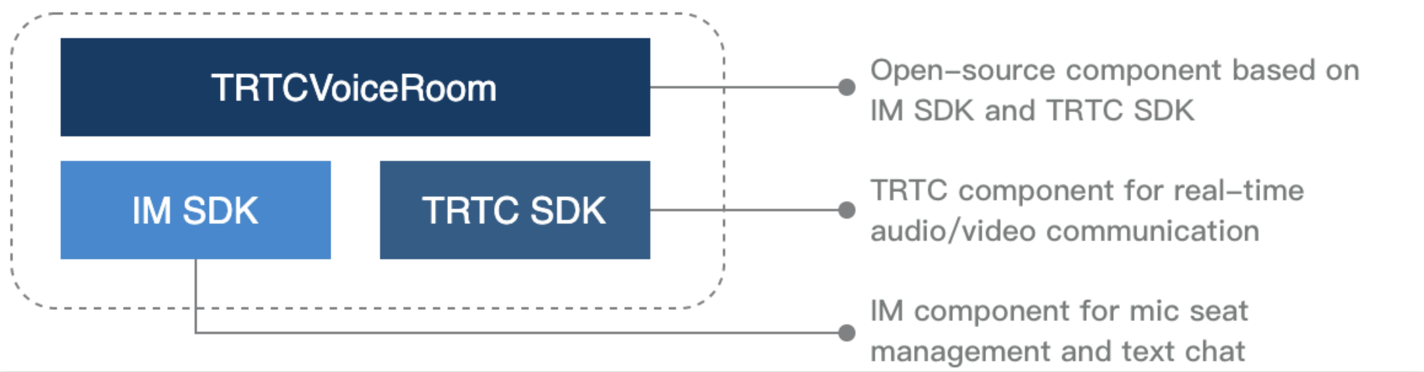
User B

1. Enter a username (**which must be unique**) and log in.
2. Enter the ID of the room created by user A and tap **Join**.

> Note :
> You can find the room ID at the top of user A's room view.

Customizing UI

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `model` subfolder contains the reusable open-source component `TRTCVoiceRoom`. You can find the component's APIs in `TRTCVoiceRoom.h` and use them to customize your own UI.


TRTCVoiceRoom
IM SDK
TRTC SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for mic seat management and text chat

Step 1. Integrate the SDK

The audio chat room component `TRTCVoiceRoom` depends on the TRTC SDK and IM SDK. Follow the steps below to integrate the two SDKs into your project.

- **Method 1: adding dependencies via CocoaPods**

```
pod 'TXIMSDK_iOS'
pod 'TXLiteAVSDK_TRTC'
```

- **Method 2: adding dependencies through local files**
  If your access to the CocoaPods repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide

TRTC SDK	Download	Integration document
IM SDK	Download	Integration document

Step 2. Configure permission requests

Configure the mic permission request by adding `Privacy > Microphone Usage Description` in `info.plist` .

Step 3. Import the `TUIVoiceRoom` component

Import the component using CocoaPods. See below for detailed directions.

1. Copy the `Source` , `Resources` , and `TXAppBasic` folders and the `TUIVoiceRoom.podspec` file to your project directory.
2. Add the following dependencies to your `Podfile` and run `pod install` to complete the import.

```
pod 'TXAppBasic', :path =&gt; "TXAppBasic/"
pod 'TXLiteAVSDK_TRTC'
pod 'TUIVoiceRoom', :path =&gt; "./", :subspecs =&gt; ["TRTC"]
```

Step 4. Create an instance and log in

1. Call the `sharedInstance` class method of `TRTCVoiceRoom` to create an instance that complies with `TRTCVoiceRoom` 's protocol, or call the `shared` class method to get a `TRTCVoiceRoomImp` instance. There is no difference between the two methods with respect to API usage.
2. Call the `setDelegate` function to register event callbacks of the component.
3. Call the `login` API to log in to the component, and set the key parameters as described below.

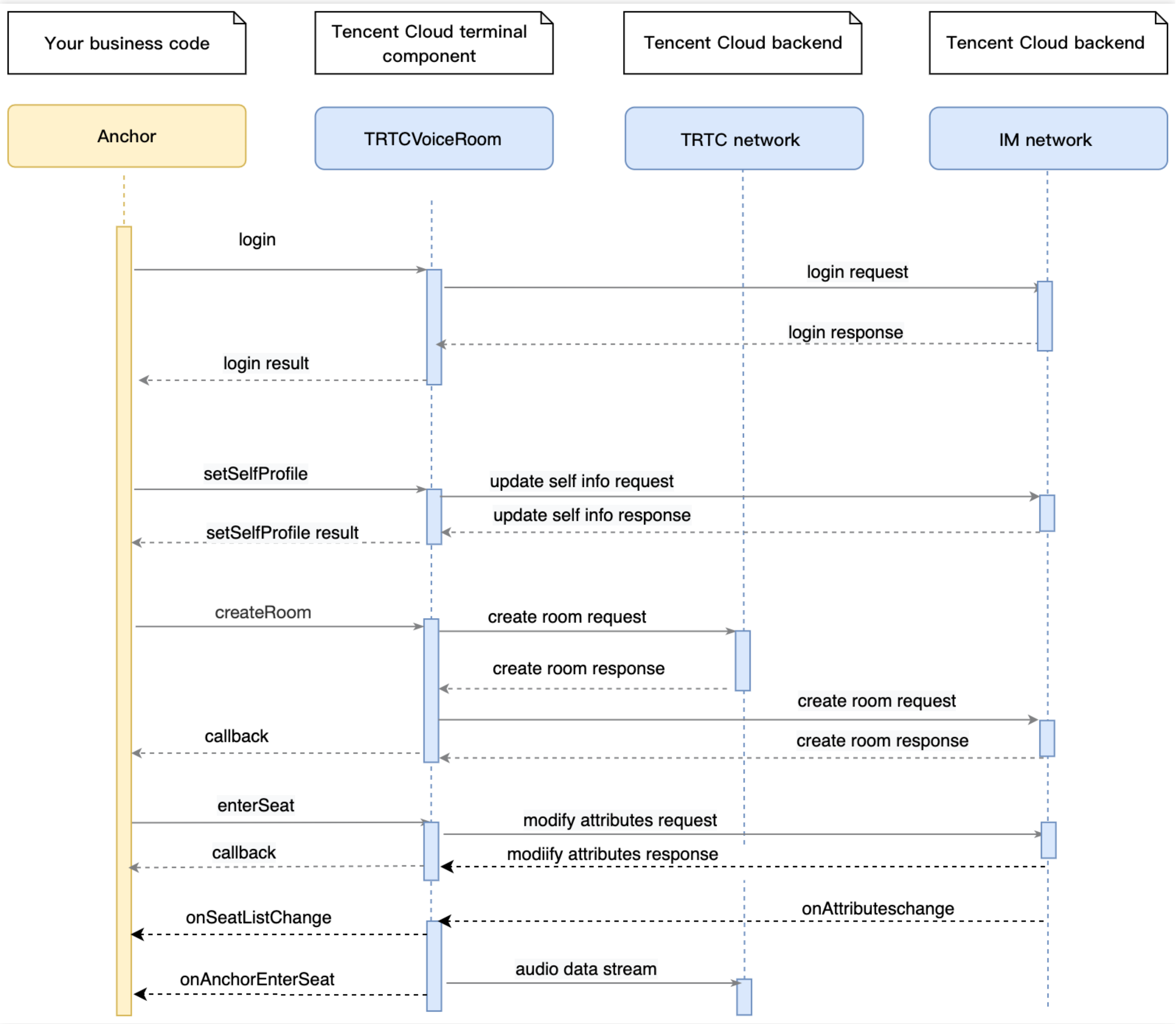
Parameter	Description
SDKAppID	You can view `SDKAppID` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.
callback	Login callback. The code is `0` if login is successful.

```
// Swift example
// The class responsible for business logic in your code
class YourController {
```

```
// Calculate attributes to get a singleton object.
var voiceRoom: TRTCVoiceRoomImp {
return TRTCVoiceRoomImp.shared()
}

// Other code logic
......
}
// Set a `voiceroom` delegate
self.vocieRoom. setDelegate(delegate: voiceRoomDelegate)

// Below is the calling method. We recommend that you use `weak self` in the closure to preven
t circular references. The weak self part is not included in the sample code below.
self.vocieRoom.login(sdkAppId: sdkAppID, userId: userId, userSig: userSig) { [weak self] (cod
e, message) in
guard let `self` = self else { return }
// Your callback business logic
}
```

Step 5. Create a room and become a speaker

1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. A user calls `createRoom` to create an audio chat room, passing in room attributes (e.g. room ID, whether listeners require room owner’s consent to speak, number of seats).
3. After creating the room, the user calls `enterSeat` to become a speaker.
4. You will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
5. You will also receive an `onAnchorEnterSeat` notification about the occupation of a seat, and mic capturing will be enabled automatically.



Sample code:

```
// 1. Set your nickname and profile photo.
self.voiceRoom.setSelfProfile(userName: userName, avatarUrl: avatarURL) { (code, message) in
// Callback of the result
}

// 2. Create a room.
let param = VoiceRoomParam.init()
param.roomName = "Room name"
param.needRequest = true // Whether your consent is required for listeners to speak
param.coverUrl = "Cover URL"
param.seatCount = 7 // Number of room seats. In this example, the number is 7. 6 seats are availa
```

```
ble after you take one.
param.seatInfoList = []

for _ in 0..&lt; param.seatCount {
let seatInfo = VoiceRoomSeatInfo.init()
param.seatInfoList.append(seatInfo)
}

self.voiceRoom.createRoom(roomID: yourRoomID, roomParam: param) { (code, message) in
guard code == 0 else { return }
// Take a seat after creating the room
self.voiceRoom.enterSeat(seatIndex: 0) { [weak self] (code, message) in
guard let `self` = self else { return }
if code == 0 {
// Seat taken successfully
} else {
// Failed to take a seat
}
}
}

// 3. After taking a seat, you receive an `onSeatListChange` notification
func onSeatListChange(seatInfoList: [VoiceRoomSeatInfo]) {
// Refresh your seat list
}

// 4. You receive an `onAnchorEnterSeat` notification
func onAnchorEnterSeat(index: Int, user: VoiceRoomUserInfo) {
// Handle the seat taking event
}
```

Step 6. Enter a room as a listener

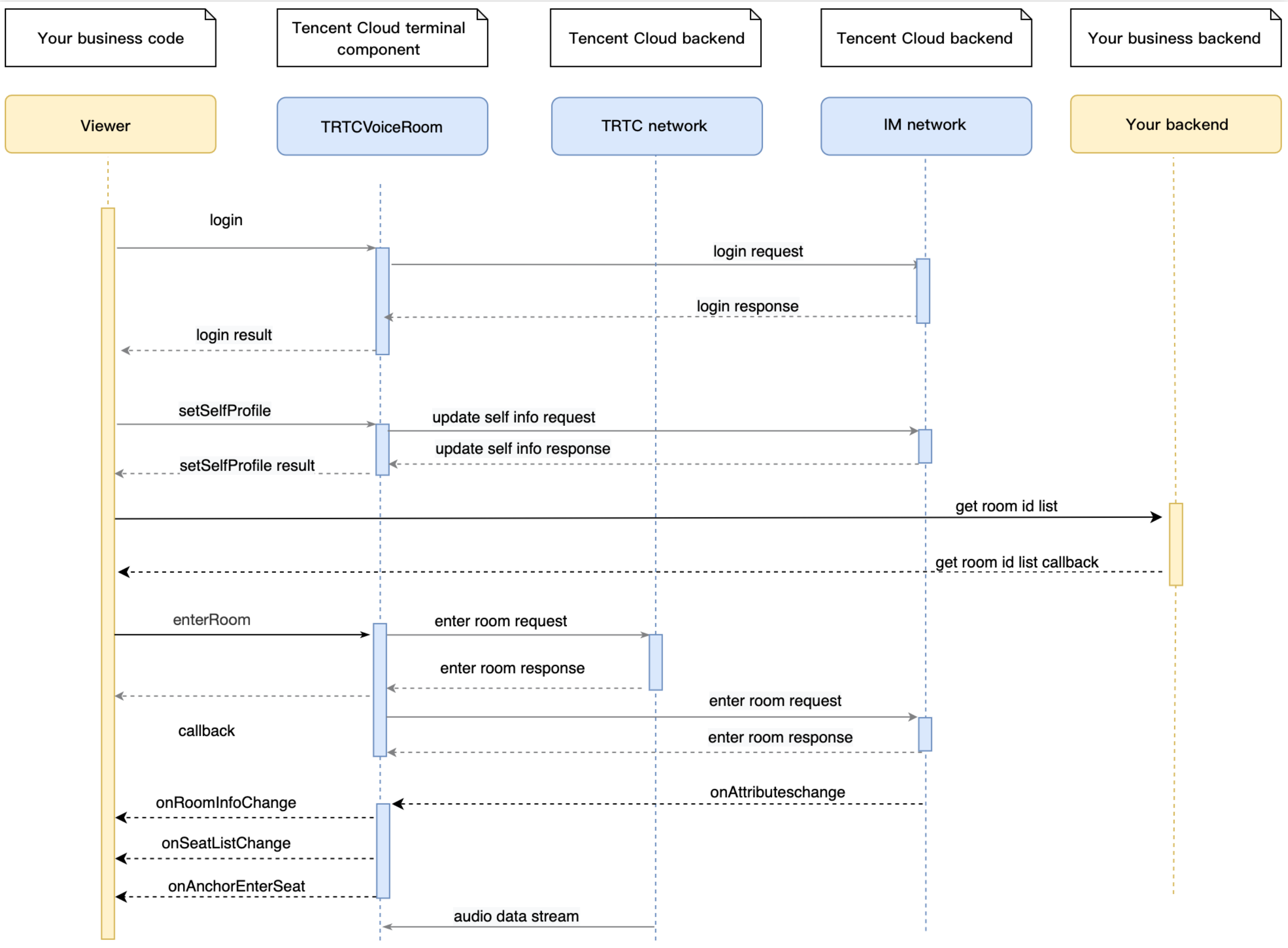
1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. Get the latest audio chat room list from the backend.

> Note :
> The audio chat room list in the app is for demonstration only. The logic of audio chat room lists varies significantly. Tencent Cloud does not provide list management services for the time being. Please manage the list by yourself.

3. Call `getRoomInfoList` to get short descriptions of the rooms, which are provided by the room owner during room creation via the calling of `createRoom` .

> Note：
> If your audio chat room list already displays enough room information, you can skip the step of calling `getRoomInfoList` .

4. The user selects a room, and calls `enterRoom` with the room ID passed in to enter the room.
5. After entering the room, you will receive an `onRoomInfoChange` notification about room change from the component. Record the room information, including room name, whether the room owner's consent is required for listeners to speak, etc., and update it to the UI.
6. You will receive an `onSeatListChange` notification about seat change from the component. Update the change to the UI.
7. You will also receive an `onAnchorEnterSeat` notification about the occupation of a seat.



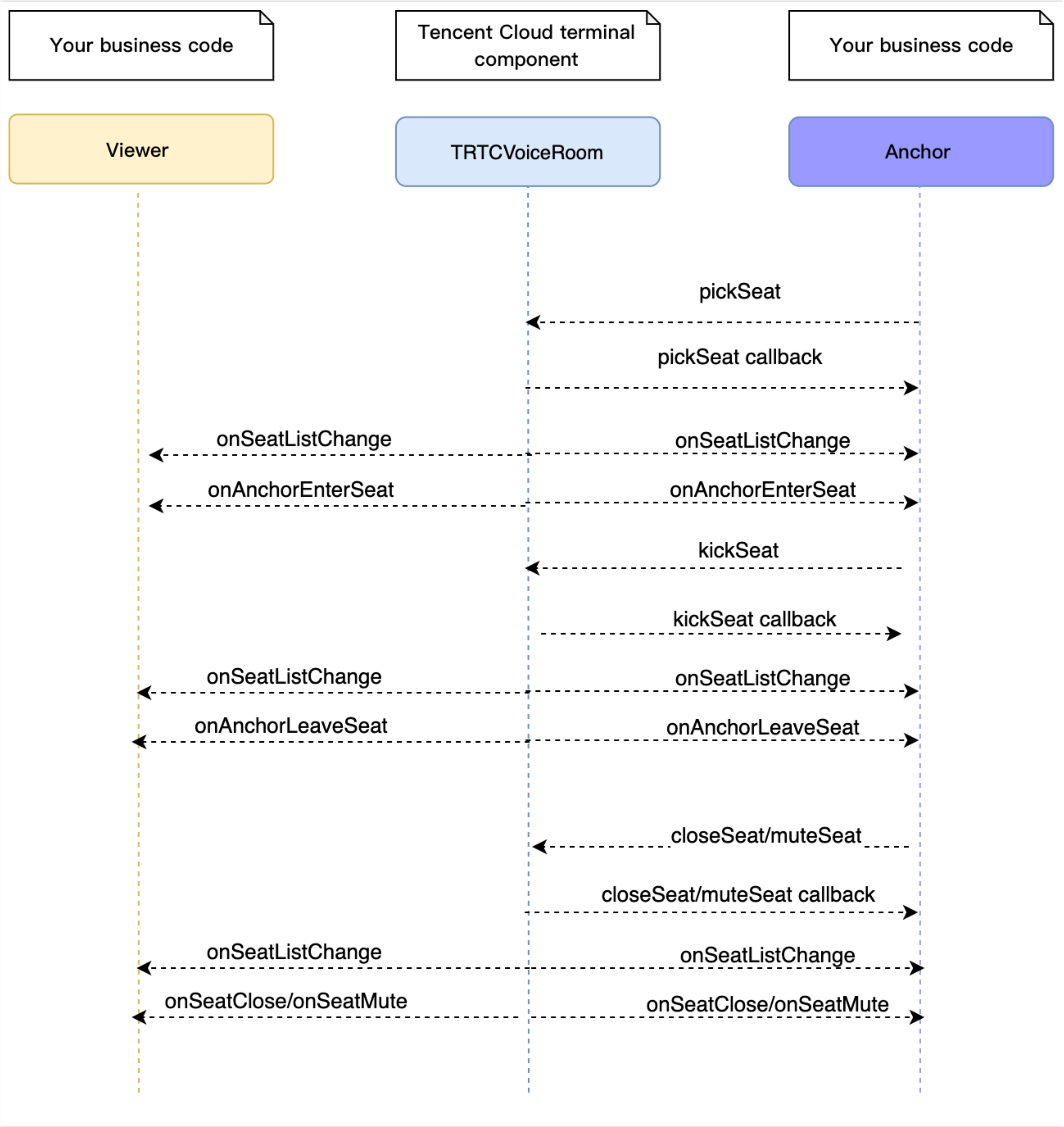
```
// 1. Set your nickname and profile photo.
self.voiceRoom.setSelfProfile(userName: userName, avatarUrl: avatarURL) { (code, message) in
// Callback of the result
}

// 2. Get the room list from the backend. Suppose it is `roomList`
let roomList: [Int] = getRoomIDList() // The function you use to get the list of room IDs

// 3. Call `getRoomInfoList` to get details of the rooms.
self.voiceRoom.getRoomInfoList(roomIdList: roomIdsInt) { (code, message, roomInfos: [VoiceRoomInfo]) in
// Refresh the UI after getting the result.
}

// 4. Select an audio chat room, and pass in the `roomId` to enter it
self.voiceRoom.enterRoom(roomID: roomInfo.roomID) { (code, message) in
// Callback of the room entry result
if code == 0 {
// Entered room
}
}

// 5. After successful room entry, you receive an `onRoomInfoChange` notification.
func onRoomInfoChange(roomInfo: VoiceRoomInfo) {
// Update the room name and other information.
}

// 6. You receive an `onSeatListChange` notification
func onSeatListChange(seatInfoList: [VoiceRoomSeatInfo]) {
// Refresh the seat list
}

// 7. You receive an `onAnchorEnterSeat` notification
func onAnchorEnterSeat(index: Int, user: VoiceRoomUserInfo) {
// Handle the seat taking event
}
```

Step 7. Manage seats

- Room owner
- Listener

1. Call `pickSeat` , passing in a seat number and the `userId` of a listener to place the listener in the seat. All users in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

2. Call `kickSeat` , passing in a seat number to remove the speaker from the seat. All users in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.
3. Call `muteSeat` , passing in a seat number to mute/unmute the seat. All members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.
4. Call `closeSeat` , passing in a seat number to block/unblock the seat. Listeners cannot take a blocked seat, and all users in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.

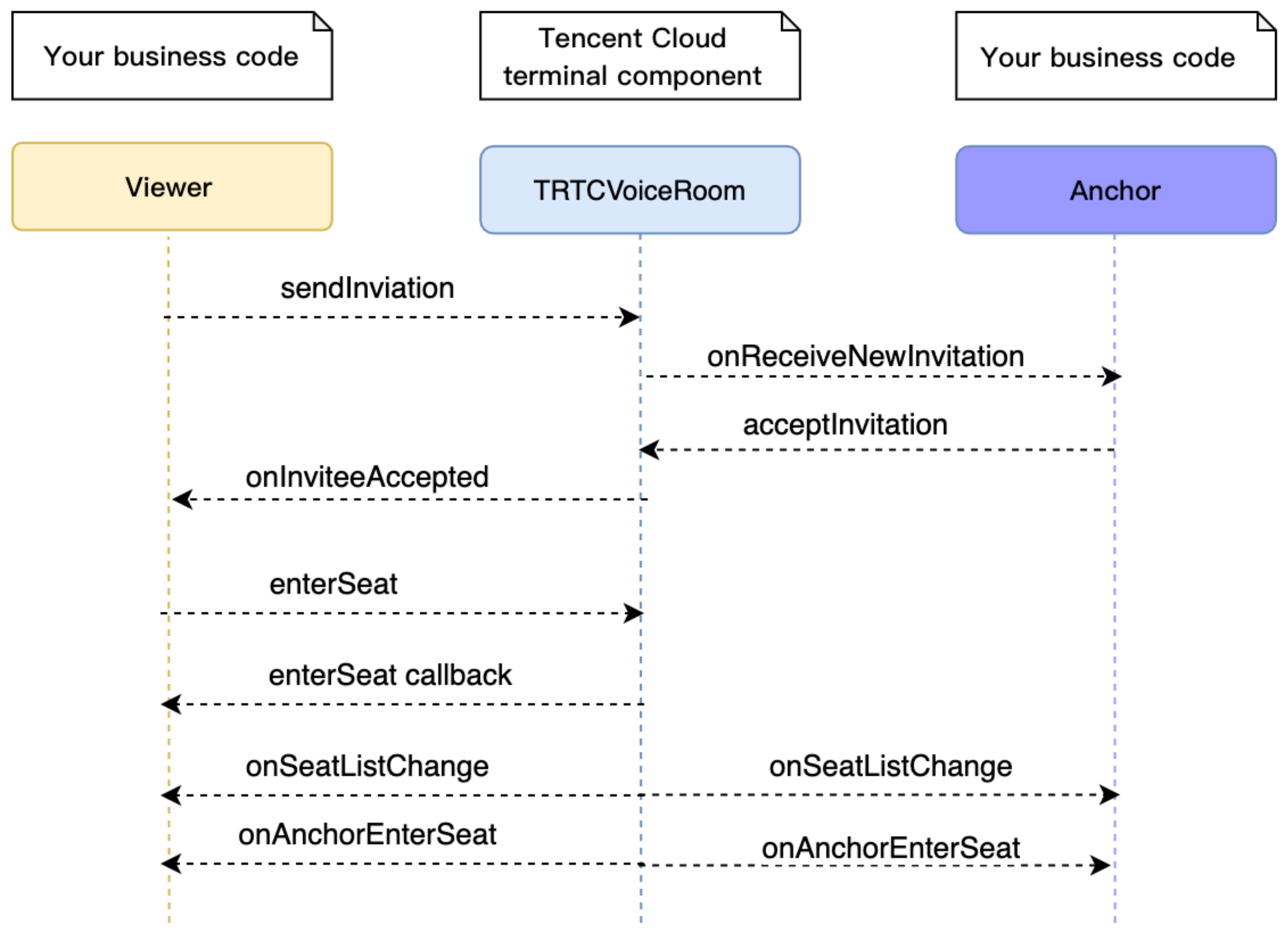


Step 8. Use signaling for invitations

In seat management, listeners can take and leave seats without the room owner's consent, and the room owner can put listeners in seats without the listeners' consent.
If you want listeners and room owners to obtain each other's consent before performing the above actions in your application, you can use signaling for invitation sending.

- Listener requesting to take seat
- Room owner inviting listener to take seat

1. A listener calls `sendInvitation` , passing in information including the room owner's `userId` and custom command words. The API will return an `inviteId` , which should be recorded.
2. The room owner receives an `onReceiveNewInvitation` notification, and a window pops up on the UI asking the room owner whether to accept the request.
3. The room owner calls `acceptInvitation` with the `inviteId` passed in to accept the request.
4. The listener receives an `onInviteeAccepted` notification and calls `enterSeat` to become a speaker.



```
// Listener
// 1. Call sendInvitation to request to take seat 1
let inviteId = self.voiceRoom.sendInvitation(cmd: "ENTER_SEAT", userId: ownerUserId, content: "1"
) { (code, message) in
// Callback of the result
}
// 2. Place the user in the seat after the invitation is accepted
func onInviteeAccepted(identifier: String, invitee: String) {
if identifier == selfID {
self.voiceRoom.enterSeat(seatIndex: ) { (code, message) in
// Callback of the result
}
}
}

// Room owner
// 1. The room owner receives the request.
func onReceiveNewInvitation(identifier: String, inviter: String, cmd: String, content: String) {
if cmd == "ENTER_SEAT" {
// 2. The room owner accepts the request.
self.voiceRoom.acceptInvitation(identifier: identifier, callback: nil)
}
}
```

Step 9. Enable text chat and on-screen comments

- Call `sendRoomTextMsg` to send a common text message. All users in the room will receive an `onRecvRoomTextMsg` callback.

  IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: send text messages
self.voiceRoom.sendRoomTextMsg(message: message) { (code, message) in


}
// Recipient: listen for text messages
func onRecvRoomTextMsg(message: String, userInfo: VoiceRoomUserInfo) {
// Handling of the messages received
}
```

- Call `sendRoomCustomMsg` to send custom (signaling) messages. All users in the room will receive an `onRecvRoomCustomMsg` callback.

  Custom messages are often used to transfer custom signals, e.g., to give and broadcast likes.

```
// For example, a sender can customize commands to distinguish on-screen comments and likes.
// For example, use ”CMD_DANMU” to indicate on-screen comments and ”CMD_LIKE” to indicate likes.
self.vocieRoom.sendRoomCustomMsg(cmd:  “CMD_DANMU” , message: ”hello world”, callback: nil)
self.voiceRoom.sendRoomCustomMsg(cmd: ”CMD_LIKE”, message: ””, callback: nil)
// Receiver: listen for custom messages
func onRecvRoomCustomMsg(cmd: String, message: String, userInfo: VoiceRoomUserInfo) {
if cmd == ”CMD_DANMU” {
// An on-screen comment is received.
}
if cmd == ”CMD_LIKE” {
// A like is received.
}
}
```

Audio Chat Room (Android)

Last updated : 2022-03-18 22:58:50

Demo UI

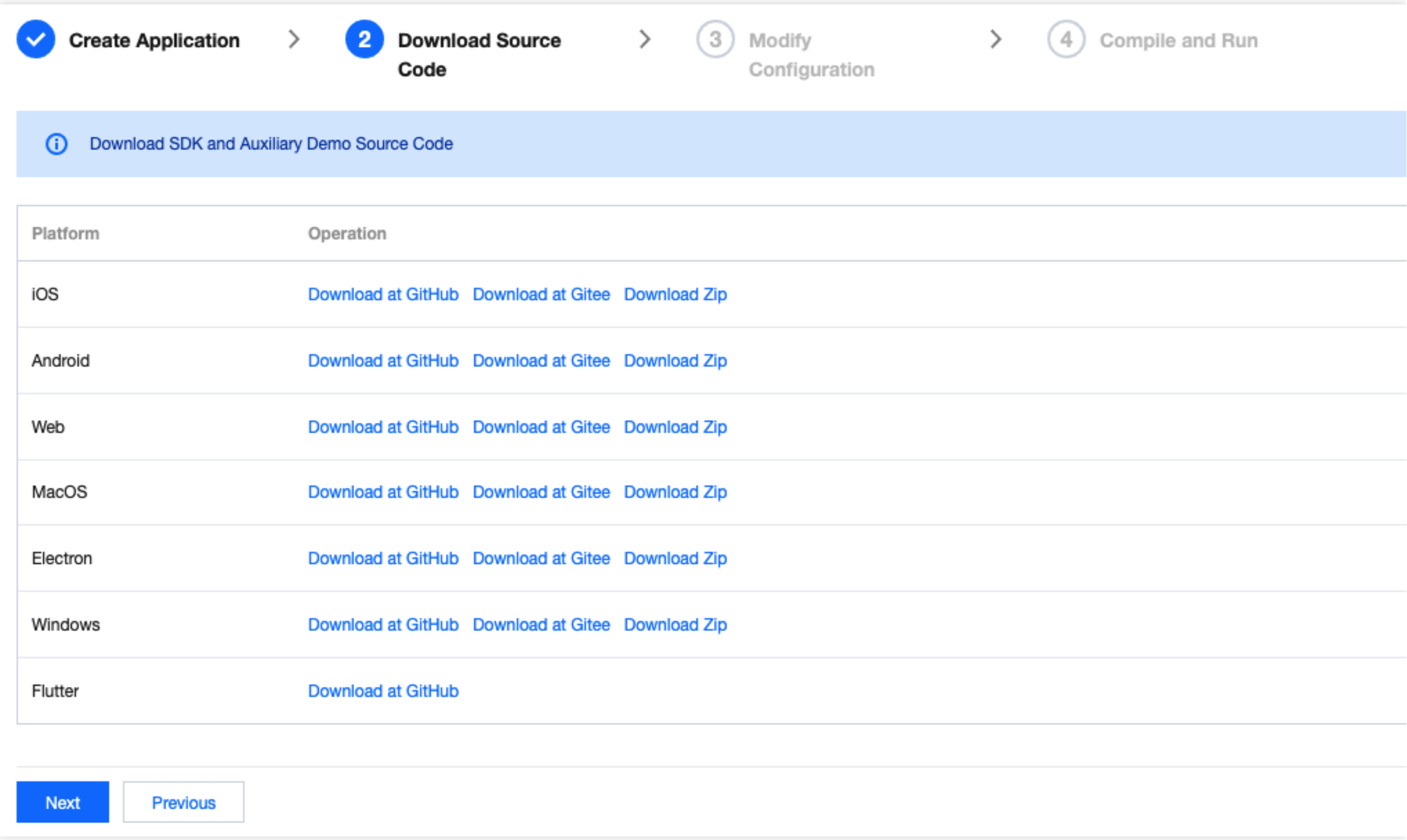
You can download and install the app we provide to try out TRTC’s audio chat room features, including seat management, low-latency audio interaction, text chat, etc.

To quickly implement the audio chat room feature, you can modify the app we provide and adapt it to your needs. You can also use the `TRTCVoiceRoom` component and customize your own UI.

Using the App’s UI

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestVoiceRoom` and click **Create**.
3. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

> Note :
> This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the application source code

Clone or download the TUIVoiceRoom source code.

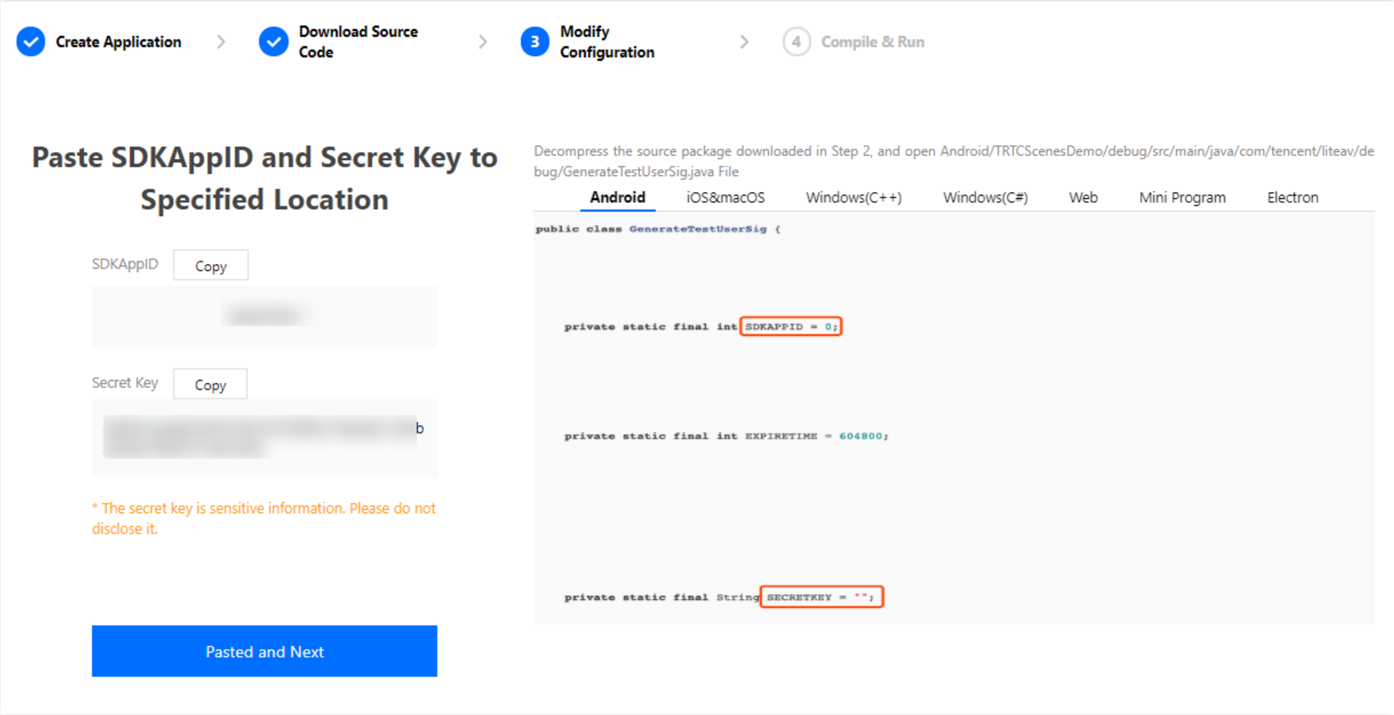
Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `Android/Debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java` .

3. Set parameters in `GenerateTestUserSig.java` :

   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUIVoiceRoom` with Android Studio (version 3.5 or above) and click **Run**.

Step 5. Modify the app’s source code

The `Source` folder in the source code contains two subfolders: `ui` and `model` . The `ui` subfolder contains UI code. The table below lists the files (folders) and the UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
base	Basic classes used by the UI
room	Main room views for room owner and listener
widget	Common controls

Tryout

> Note :
>
> You need at least two devices to try out the application.

User A

1. Enter a username (**which must be unique**) and log in.
2. Click **Create Room**.
3. Type a subject for the conference and tap **Let’s go**.

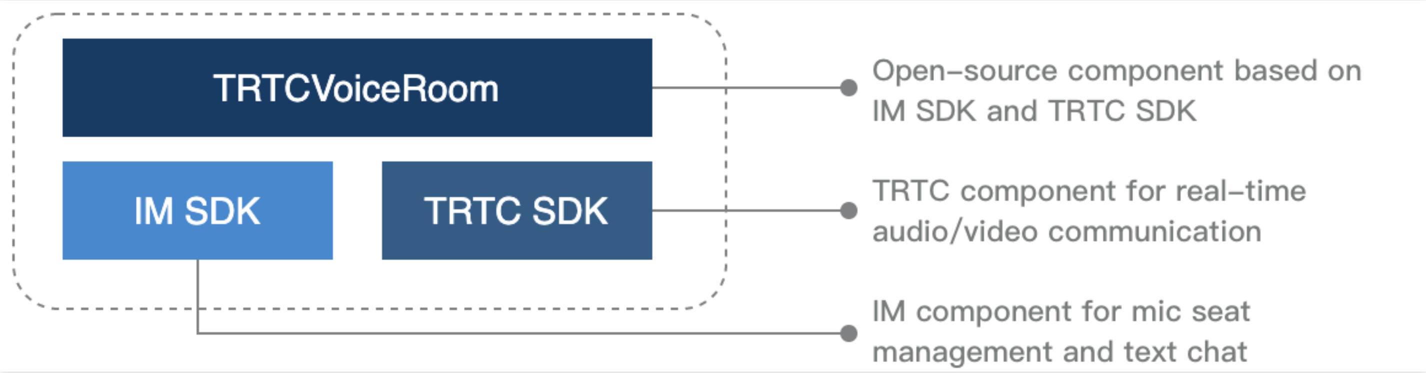
User B

1. Enter a username (**which must be unique**) and log in.
2. Enter the ID of the room created by user A and tap **Join**.

> Note :
>
> You can find the room ID at the top of user A’s room view.

Customizing UI

The `Source` folder in the source code contains two subfolders: `ui` and `model` . The `model` subfolder contains the reusable open-source component `TRTCVoiceRoom` . You can find the component's APIs in `TRTCVoiceRoom.java` and use them to customize your own UI.


TRTCVoiceRoom
IM SDK
TRTC SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for mic seat management and text chat

Step 1. Integrate the SDK

The audio chat room component `TRTCVoiceRoom` depends on the TRTC SDK and IM SDK. Follow the steps below to integrate the two SDKs into your project.

Method 1: adding dependencies via Maven

1. Add the TRTC SDK and IM SDK dependencies to `dependencies`.

```
dependencies {
complie "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"
complie 'com.tencent.imsdk:imsdk:latest.release'
compile 'com.google.code.gson:gson:2.3.1'
}
```

2. In `defaultConfig`, specify the CPU architecture to be used by your application.

```
defaultConfig {
ndk {
abiFilters "armeabi-v7a"
}
}
```

3. Click **Sync Now** to automatically download the SDKs and integrate them into your project.

Method 2: adding dependencies through local AAR files

If your access to the Maven repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration document
IM SDK	Download	Integration document

Step 2. Configure permission requests and obfuscation rules

Configure permission requests for your app in `AndroidManifest.xml` . The SDKs need the following permissions (on Android 6.0 and above, the read storage permission must be requested at runtime.)

```
<uses-permission android:name=”android.permission.INTERNET”>
<uses-permission android:name=”android.permission.ACCESS_NETWORK_STATE”>
<uses-permission android:name=”android.permission.ACCESS_WIFI_STATE”>
<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE”>
<uses-permission android:name=”android.permission.READ_EXTERNAL_STORAGE”>
<uses-permission android:name=”android.permission.RECORD_AUDIO”>
<uses-permission android:name=”android.permission.MODIFY_AUDIO_SETTINGS”>
<uses-permission android:name=”android.permission.BLUETOOTH”>
<uses-permission android:name=”android.permission.READ_PHONE_STATE”>
```

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

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

Step 3. Import the `TRTCVoiceRoom` component

Copy all the files in the directory below to your project:

```
Source/src/main/java/com/tencent/liteav/trtcvoiceroom/model
```

Step 4. Create an instance and log in

1. Call the `sharedInstance` API to create an instance of the `TRTCVoiceRoom` component.
2. Call the `setDelegate` API to register event callbacks of the component.
3. Call the `login` API to log in to the component, and set the key parameters as described below.

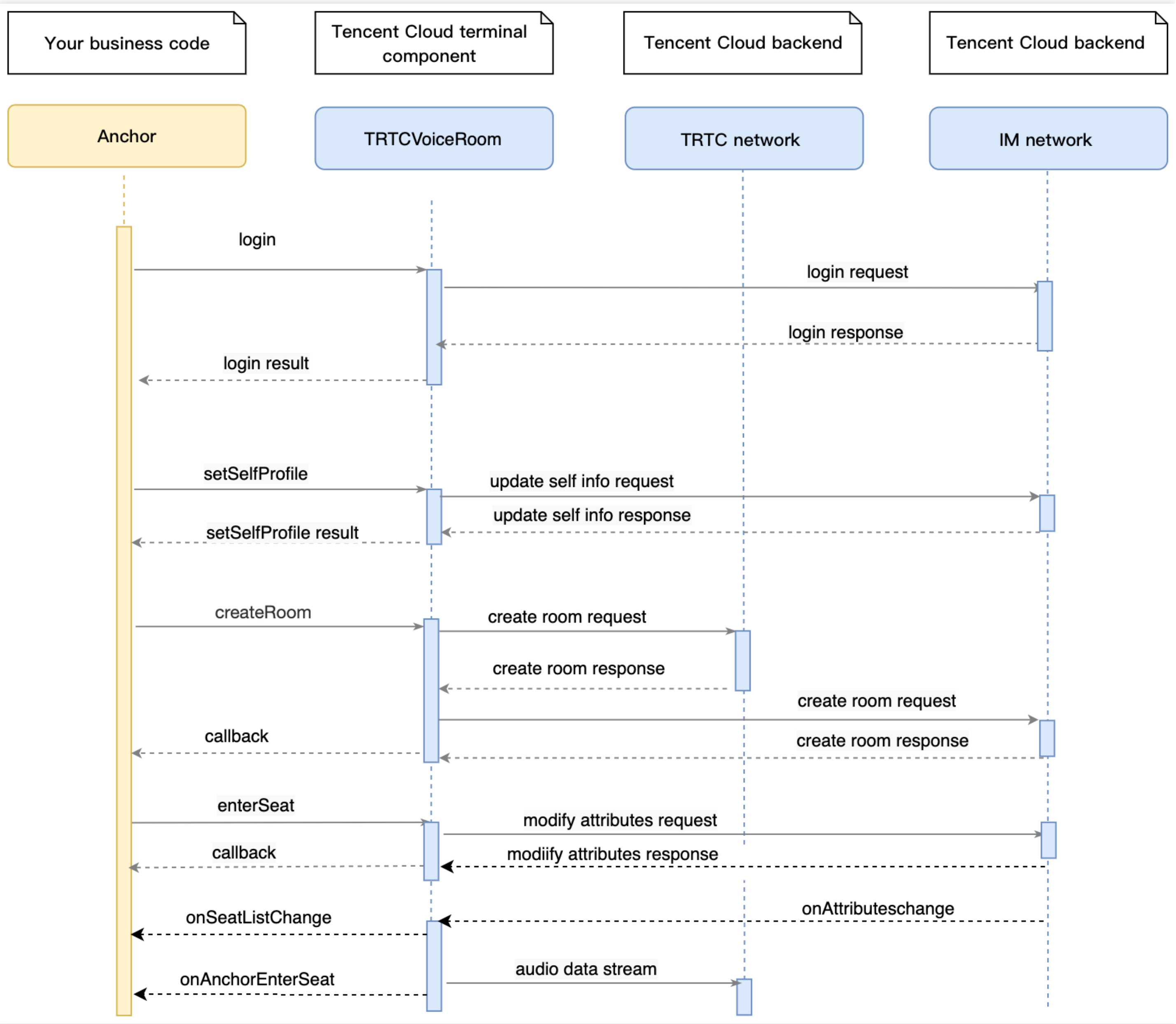
Parameter Name	Description
SDKAppID	You can view \`SDKAppID\` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.

userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.
callback	Login callback. The code is `0` if login is successful.

```
TRTCVoiceRoom mTRTCVoiceRoom = TRTCVoiceRoom.sharedInstance(this);
mTRTCVoiceRoom.setDelegate(this);
mTRTCVoiceRoom.login(SDKAPPID, userId, userSig, new TRTCVoiceRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {![](https://main.qcloudimg.com/raw/3151fc0df16c063db7a75f6c1facf562.png)
// Logged in
}
}
});
```

Step 5. Create a room and become a speaker

1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. A user calls `createRoom` to create an audio chat room, passing in room attributes (e.g. room ID, whether listeners require room owner’s consent to speak, number of seats).
3. After creating the room, the user calls `enterSeat` to become a speaker.
4. You will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
5. You will also receive an `onAnchorEnterSeat` notification about the occupation of a seat, and mic capturing will be enabled automatically.



```
// 1. Set your nickname and profile photo.
mTRTCVoiceRoom.setSelfProfile("my_name", "my_face_url", null);

// 2. Call `createRoom` to create a room
final TRTCVoiceRoomDef.RoomParam roomParam = new TRTCVoiceRoomDef.RoomParam();
roomParam.roomName = "Room name";
roomParam.needRequest = false; // Whether your consent is required for listeners to speak
roomParam.seatCount = 7; // Number of room seats. In this example, the number is 7. 6 seats are available after you take one.
roomParam.coverUrl = "URL of room cover image";
mTRTCVoiceRoom.createRoom(mRoomId, roomParam, new TRTCVoiceRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
```

```
if (code == 0) {
// 3. Take seat 0
mTRTCVoiceRoom.enterSeat(0, new TRTCVoiceRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
}
}
});
}
}
});

// 4. After taking a seat, you receive an `onSeatListChange` notification
@Override
public void onSeatListChange(final List<trtcvoiceroomdef.seatinfo> seatInfoList) {
// Display the seat list
}

// 5. You receive an `onAnchorEnterSeat` notification
@Override
public void onAnchorEnterSeat(TRTCVoiceRoomDef.UserInfo userInfo) {
}
```

Step 6. Enter a room as a listener

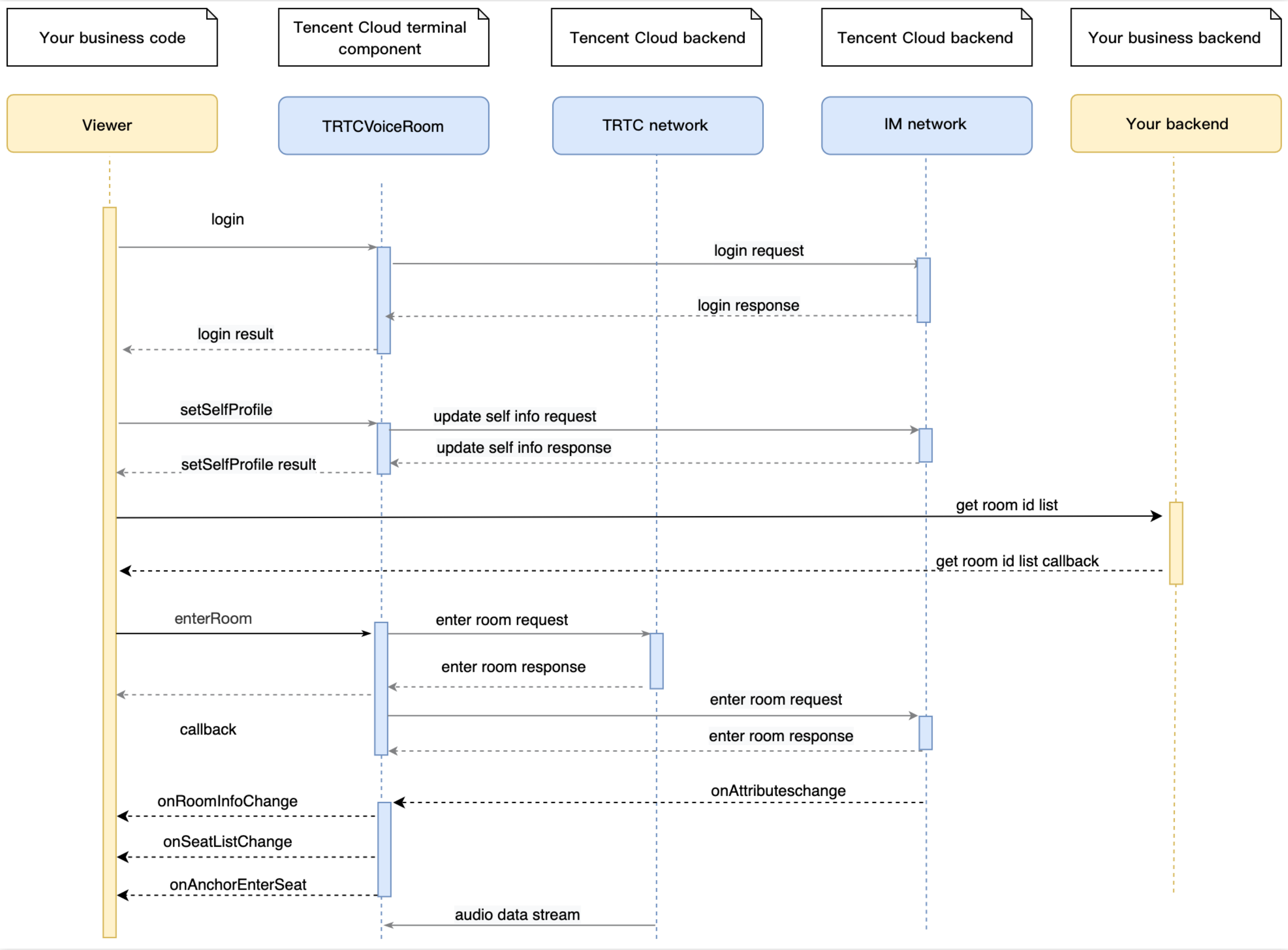
1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. Get the latest audio chat room list from the backend.

> Note :
> The audio chat room list in the app is for demonstration only. The logic of audio chat room lists varies significantly. Tencent Cloud does not provide list management services for the time being. Please manage the list by yourself.

3. Call `getRoomInfoList` to get short descriptions of the rooms, which are provided by the room owner during room creation via the calling of `createRoom` .

> Note :
> If your audio chat room list already displays enough room information, you can skip the step of calling `getRoomInfoList` .

4. The user selects a room, and calls `enterRoom` with the room ID passed in to enter the room.
5. After entering the room, you will receive an `onRoomInfoChange` notification about room change from the component. Record the room information, including room name, whether the room owner's consent is required for listeners to speak, etc., and update it to the UI.
6. You will receive an `onSeatListChange` notification about seat change from the component. Update the change to the UI.
7. You will also receive an `onAnchorEnterSeat` notification that someone becomes a speaker.



```
// 1. Set your nickname and profile photo.
mTRTCVoiceRoom.setSelfProfile("my_name", "my_face_url", null);

// 2. Get the room list from the backend. Suppose it is `roomList`.
List<integer> roomList = GetRoomList();

// 3. Call `getRoomInfoList` to get details of the rooms.
mTRTCVoiceRoom.getRoomInfoList(roomList, new TRTCVoiceRoomCallback.RoomInfoCallback() {
```

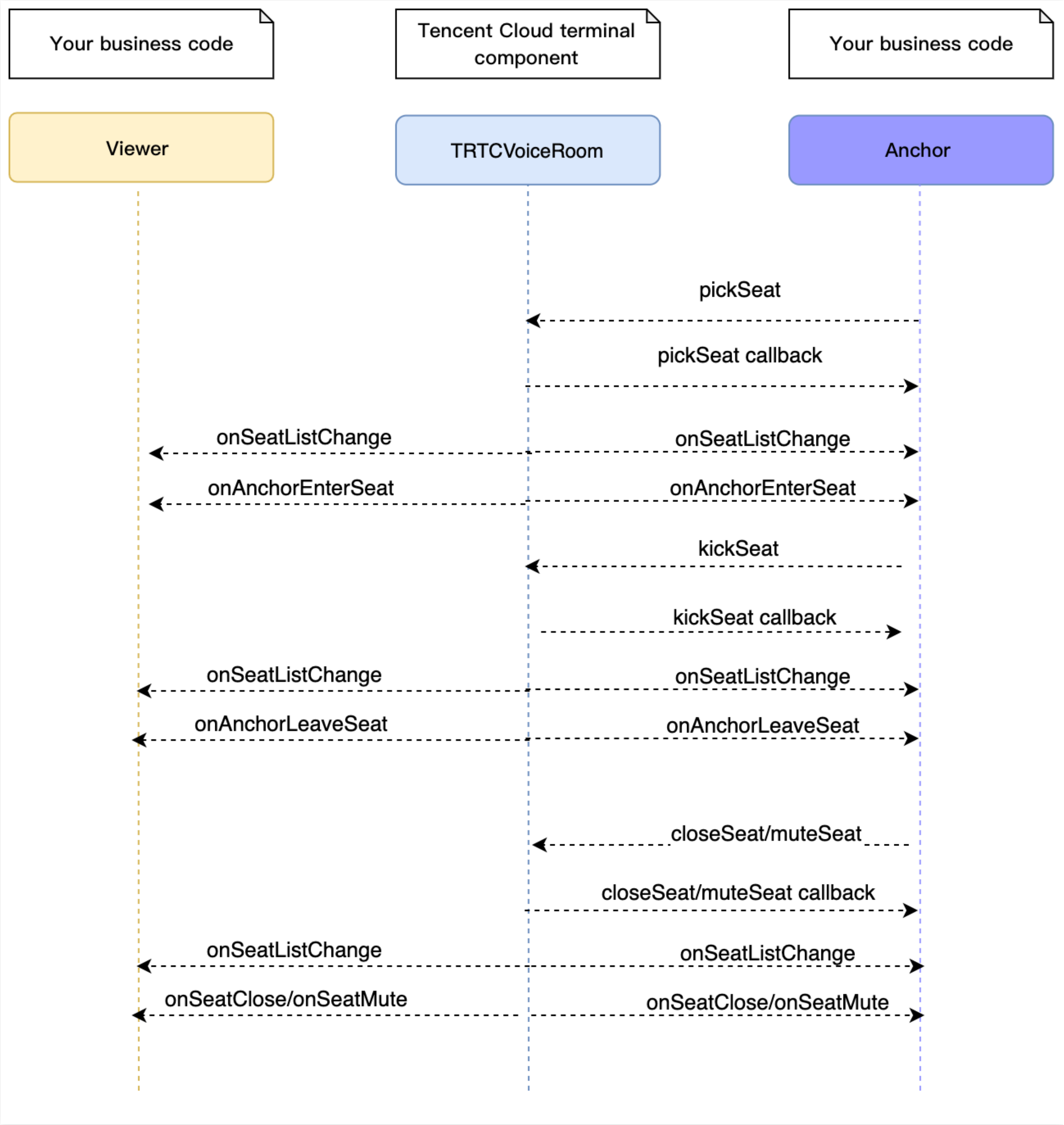
```
@Override
public void onCallback(int code, String msg, List<trtcvoiceroomdef.roominfo> list) {
if (code == 0) {
// Refresh the room list on your UI
}
}
});

// 4. Select an audio chat room, and pass in the `roomId` to enter it
mTRTCVoiceRoom.enterRoom(roomId, new TRTCVoiceRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// Entered room successfully
}
}
});

// 5. After successful room entry, you receive an `onRoomInfoChange` notification.
@Override
public void onRoomInfoChange(TRTCVoiceRoomDef.RoomInfo roomInfo) {
mNeedRequest = roomInfo.needRequest;
mRoomName = roomInfo.roomName;
// The UI can display the title and other information
}

// 6. You receive an `onSeatListChange` notification
@Override
public void onSeatListChange(final List<trtcvoiceroomdef.seatinfo> seatInfoList) {
// Display the seat list
}

// 7. You receive an `onAnchorEnterSeat` notification
@Override
public void onAnchorEnterSeat(TRTCVoiceRoomDef.UserInfo userInfo) {
}
```

Step 7. Manage seats

- Room owner
- Listener

1. Call `pickSeat` , passing in a seat number and the `userId` of a listener to place the listener in the seat. All users in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

2. Call `kickSeat` , passing in a seat number to remove the speaker from the seat. All users in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.
3. Call `muteSeat` , passing in a seat number to mute/unmute the seat. All members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.
4. Call `closeSeat` , passing in a seat number to block/unblock the seat. Listeners cannot take a blocked seat, and all users in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.

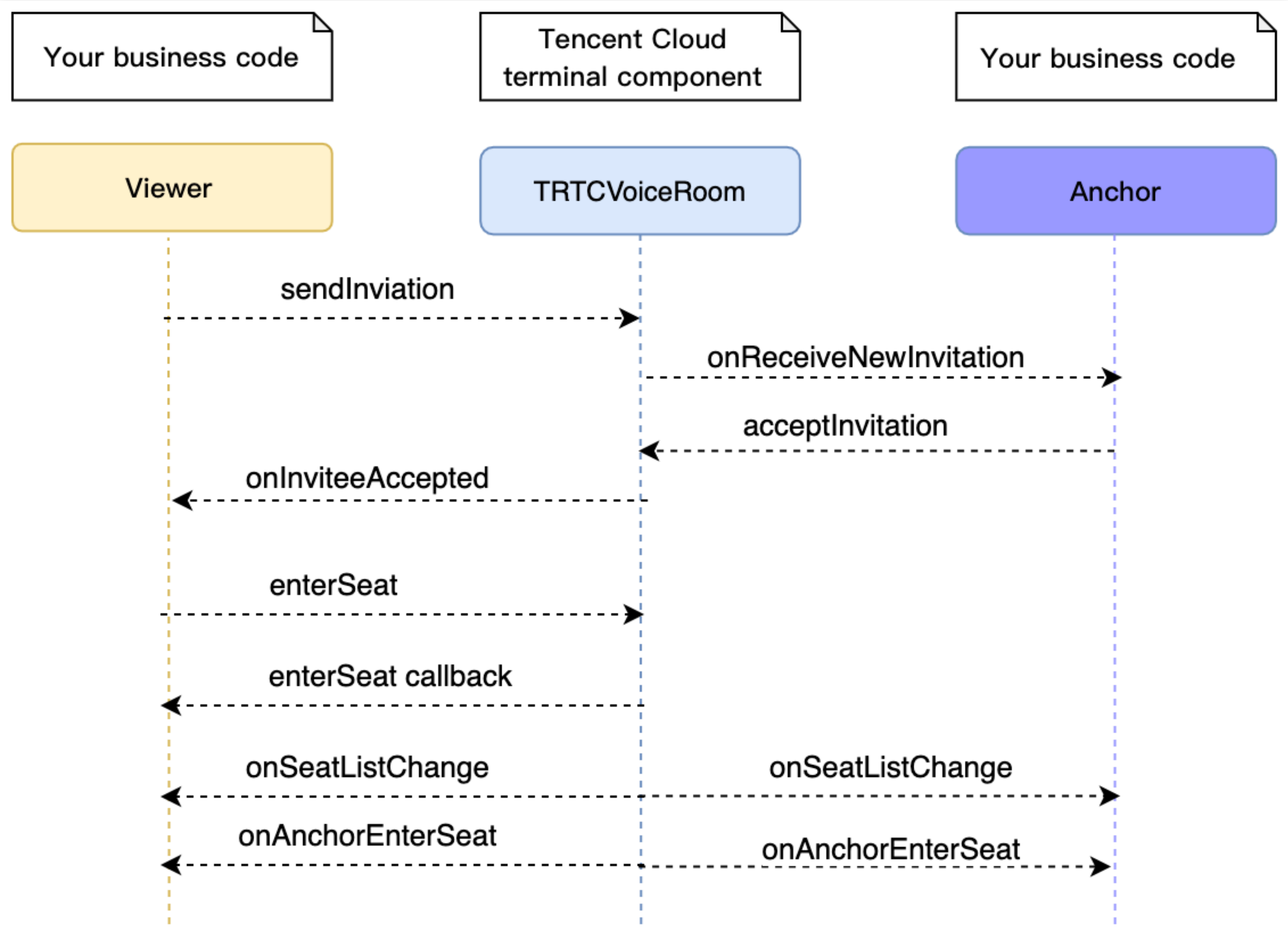


Step 8. Use signaling for invitations

In seat management, listeners can take and leave seats without the room owner's consent, and the room owner can put listeners in seats without the listeners' consent.
If you want listeners and room owners to obtain each other's consent before performing the above actions in your application, you can use signaling for invitation sending.

- Listener requesting to take seat
- Room owner inviting listener to take seat

1. A listener calls `sendInvitation` , passing in information including the room owner's `userId` and custom command words. The API will return an `inviteId` , which should be recorded.
2. The room owner receives an `onReceiveNewInvitation` notification, and a window pops up on the UI asking the room owner whether to accept the request.
3. The room owner calls `acceptInvitation` with the `inviteId` passed in to accept the request.
4. The listener receives an `onInviteeAccepted` notification and calls `enterSeat` to become a speaker.



```
// Listener
// 1. Call sendInvitation to request to take seat 1
String inviteId = mTRTCVoiceRoom.sendInvitation("ENTER_SEAT", ownerUserId, "1", null);

// 2. Place the user in the seat after the invitation is accepted
@Override
public void onInviteeAccepted(String id, String invitee) {
if(id.equals(inviteId)) {
mTRTCVoiceRoom.enterSeat(1, null);
}
}

// Room owner
// 1. The room owner receives the request.
@Override
public void onReceiveNewInvitation(final String id, String inviter, String cmd, final String cont
ent) {
if (cmd.equals("ENTER_SEAT")) {
// 2. The room owner accepts the request.
mTRTCVoiceRoom.acceptInvitation(id, null);
}
}
```

Step 9. Enable text chat and on-screen comments

- Call `sendRoomTextMsg` to send common text messages. All users in the room will receive an `onRecvRoomTextMsg` callback.

  IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: send text messages
mTRTCVoiceRoom.sendRoomTextMsg("Hello Word!", null);
// Recipient: listen for text messages
mTRTCVoiceRoom.setDelegate(new TRTCVoiceRoomDelegate() {
@Override
public void onRecvRoomTextMsg(String message, TRTCVoiceRoomDef.UserInfo userInfo) {
Log.d(TAG, "Received a message from" + userInfo.userName + ": " + message);
}
});
```

- Call `sendRoomCustomMsg` to send custom (signaling) messages. All users in the room will receive an `onRecvRoomCustomMsg` callback.

  Custom messages are often used to transfer custom signals, e.g., to give and broadcast likes.

```java
// A sender can customize CMD to distinguish on-screen comments and likes.
// For example, use "CMD_DANMU" to indicate on-screen comments and "CMD_LIKE" to indicate likes.
mTRTCVoiceRoom.sendRoomCustomMsg("CMD_DANMU", "Hello world", null);
mTRTCVoiceRoom.sendRoomCustomMsg("CMD_LIKE", "", null);
// Receiver: listen for custom messages
mTRTCVoiceRoom.setDelegate(new TRTCVoiceRoomDelegate() {
@Override
public void onRecvRoomCustomMsg(String cmd, String message, TRTCVoiceRoomDef.UserInfo userInfo) {
if ("CMD_DANMU".equals(cmd)) {
// An on-screen comment is received.
Log.d(TAG, "Received an on-screen comment from" + userInfo.userName + ": " + message);
} else if ("CMD_LIKE".equals(cmd)) {
// A like is received.
Log.d(TAG, userInfo.userName + "liked you.");
}
}
});
```

TRTCVoiceRoom (iOS)

Last updated : 2022-01-05 15:50:42

`TRTCVoiceRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create an audio chat room and become a speaker, or enter an audio chat room as a listener.
- The room owner can invite a listener to speak as well as remove a speaker.
- The room owner can also block a seat. Listeners cannot request to take a blocked seat.
- A listener can request to speak and become a speaker. A speaker can also become a listener.
- All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

`TRTCVoiceRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Audio Chat Room (iOS).

- TRTC SDK: the TRTC SDK is used as a low-latency audio chat component.
- IM SDK: the `AVChatRoom` feature of the IM SDK is used to implement chat rooms. The attribute APIs of IM are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

TRTCVoiceRoom API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
setDelegate	Sets event callbacks.
setDelegateHandler	Sets the thread where event callbacks are.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
destroyRoom	Terminates a room (called by room owner).
enterRoom	Enters a room (called by listener).
exitRoom	Exits a room (called by listener).
getRoomInfoList	Gets room list details.
getUserInfoList	Gets the user information of the specified `userId` . If the value is `nil` , the information of all users in the room is obtained.

Seat management APIs

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

Local audio APIs

API	Description
startMicrophone	Starts mic capturing.
stopMicrophone	Stops mic capturing.
setAudioQuality	Sets audio quality.
muteLocalAudio	Mutes/Unmutes local audio.

API	Description
setSpeaker	Sets whether to use the speaker or receiver.
setAudioCaptureVolume	Sets mic capturing volume.
setAudioPlayoutVolume	Sets playback volume.
setVoiceEarMonitorEnable	Enables/Disables in-ear monitoring.

Remote audio APIs

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

Background music and audio effect APIs

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

Message sending APIs

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

Invitation signaling APIs

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

TRTCVoiceRoomDelegate API Overview

Common event callback APIs

API	Description
onError	Error
onWarning	Warning
onDebugLog	Log

Room event callback APIs

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

Seat list change callback APIs

API	Description
onSeatListChange	All seat changes
onAnchorEnterSeat	Someone became a speaker or was made a speaker by the room owner.
onAnchorLeaveSeat	Someone became a listener or was moved to listeners by the room owner.
onSeatMute	The room owner muted a seat.
onUserMicrophoneMute	Whether a user’s mic is muted
onSeatClose	The room owner blocked a seat.

Callback APIs for room entry/exit by listener

API	Description
onAudienceEnter	A listener entered the room.

API	Description
onAudienceExit	A listener exited the room.

Message event callback APIs

API	Description
onRecvRoomTextMsg	Receipt of a text message
onRecvRoomCustomMsg	A custom message was received.

Signaling event callback APIs

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

Basic SDK APIs

sharedInstance

This API is used to get a TRTCVoiceRoom singleton object.

```
/**
* Get a `TRTCVoiceRoom` singleton object
*
* - returns: `TRTCVoiceRoom` instance
* - note: to terminate a singleton object, call {@link TRTCVoiceRoom#destroySharedInstance()}.
*/
+ (instancetype)sharedInstance NS_SWIFT_NAME(shared());
```

destroySharedInstance

This API is used to terminate a TRTCVoiceRoom singleton object.

Note :

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

```
/**
* Terminate a `TRTCVoiceRoom` singleton object
*
* - note: after the instance is terminated, the externally cached `TRTCVoiceRoom` instance can no longer be used. You need to call {@link TRTCVoiceRoom#sharedInstance()} again to get a new instance.
*/
+ (void)destroySharedInstance NS_SWIFT_NAME(destroyShared());
```

setDelegate

This API is used to set the event callback of TRTCVoiceRoom. You can use `TRTCVoiceRoomDelegate` to get different status notifications of TRTCVoiceRoom.

```
/**
* Set the event callbacks of the component
*
* You can use `TRTCVoiceRoomDelegate` to get different status notifications of `TRTCVoiceRoom`
*
* - parameter delegate Callback API
* - note: callback events in `TRTCVoiceRoom` are called back to you in the main queue by default. If you need to specify a queue for event callback, please use {@link TRTCVoiceRoom#setDelegateQueue(queue)}.
*/
- (void)setDelegate:(id<TRTCVoiceRoomDelegate>)delegate NS_SWIFT_NAME(setDelegate(delegate:));
```

Note :

`setDelegate` is the delegate callback of `TRTCVoiceRoom` .

setDelegateQueue

This API is used to set the thread queue for event callbacks. The main thread (MainQueue) is used by default.

```
/**
* Set the queue for event callbacks
*
* - parameter queue Queue. Various status callback notifications in `TRTCVoiceRoom` will be sent
to the queue you specify.
*/
- (void)setDelegateQueue:(dispatch_queue_t)queue NS_SWIFT_NAME(setDelegateQueue(queue:));
```

The parameters are as detailed below:

Parameter	Type	Description
queue	dispatch_queue_t	The status notifications of `TRTCVoiceRoom` are sent to the thread queue you specify.

login

This API is used to log in to the Tencent backend server.

```
- (void)login:(int)sdkAppID
userId:(NSString *)userId
userSig:(NSString *)userSig
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(login(sdkAppID:userId:userSig:callback
:));
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	int	You can view `SDKAppID` in **Application Management** > **Application Info** of the TRTC console.
userId	NSString	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 more information on how to get it, please see UserSig.
callback	ActionCallback	Callback for login. The code is 0 if login succeeds.

logout

This API is used to log out of the Tencent backend server.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for logout. The code is `0` if logout succeeds.

setSelfProfile

This API is used to set the profile.

```
- (void)setSelfProfile:(NSString *)userName avatarURL:(NSString *)avatarURL callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(setSelfProfile(userName:avatarURL:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userName	NSString	Username
avatarURL	NSString	Profile picture URL
callback	ActionCallback	Callback for profile setting. The code is `0` if the operation succeeds.

Room APIs

createRoom

This API is used to create a room (called by room owner).

```
- (void)createRoom:(int)roomID roomParam:(VoiceRoomParam *)roomParam callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
roomId	int	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a karaoke room list. Currently, Tencent Cloud does not provide management services for room lists. Please manage the list on your own.
roomParam	VoiceRoomParam	Room information, such as room name, seat list information, and cover information. To manage seats, you must enter the number of seats in the room.
callback	ActionCallback	Callback for room creation. The code is 0 if the operation succeeds.

The process of creating a karaoke room and becoming a speaker is as follows:

1. A user calls `createRoom` to create an audio chat room, passing in room attributes (e.g. room ID, whether listeners require room owner's consent to speak, number of seats).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
4. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

destroyRoom

This API is used to terminate a room (called by room owner).

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

The parameters are as detailed below:

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

enterRoom

This API is used to enter a room (called by listener).

```
- (void)enterRoom:(NSInteger)roomID callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(ent
erRoom(roomID:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	NSInteger	Room ID
callback	ActionCallback	Callback for room entry. The code is `0` if the operation succeeds.

The process of entering a room as a listener is as follows:

1. A user gets the latest audio chat room list from your server. The list may contain the `roomId` and room information of multiple audio chat rooms.
2. The user selects a room, and calls `enterRoom` with the room ID passed in to enter the room.
3. After entering the room, the user receives an `onRoomInfoChange` notification about room attribute change from the component. The attributes can be recorded, and corresponding changes can be made to the UI, including room name, whether room owner's consent is required for listeners to speak, etc.
4. The user will receive an `onSeatListChange` notification about the change of the seat list and can update the change to the UI.
5. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker.

exitRoom

This API is used to exit a room.

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

The parameters are as detailed below:

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

getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()` .

Note :

You don’t need this API if both the room list and room information are managed on your server.

```
- (void)getRoomInfoList:(NSArray<NSNumber *> *)roomIdList callback:(VoiceRoomInfoCallback _Nullab
le)callback NS_SWIFT_NAME(getRoomInfoList(roomIdList:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomIdList	NSArray<NSNumber>	Room ID list
callback	RoomInfoCallback	Callback of room details

getUserInfoList

This API is used to get the information of specific users (`userId`).

```
- (void)getUserInfoList:(NSArray<NSString *> * _Nullable)userIDList callback:(VoiceRoomUserListCa
llback _Nullable)callback NS_SWIFT_NAME(getUserInfoList(userIDList:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userIdList	NSArray<NSString>	IDs of the users to query. If this parameter is `null` , the information of all users in the room is queried.
userlistcallback	UserListCallback	Callback of user details

Seat Management APIs

enterSeat

This API is used to become a speaker (called by room owner or listener).

> Note :
>
> After a user becomes a speaker, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

```
- (void)enterSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME
(enterSeat(seatIndex:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to take
callback	ActionCallback	Callback for the operation

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

moveSeat

This API is used to change one's seat (called by speaker).

> Note :
> After the seat change, all users in the room will receive the `onSeatListChange` , `onAnchorLeaveSeat` , and `onAnchorEnterSeat` notifications. This API will only change the user's seat number, not the user role.

```
- (NSInteger)moveSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(moveSeat(seatIndex:callback:))
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to change to
callback	ActionCallback	Callback for the operation

Response parameters:

Parameter	Type	Description
code	NSInteger	Result of seat change. `0` : operation successful; `10001` : API rate limit exceeded; other values: operation failed

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

leaveSeat

This API is used to become a listener (called by speaker).

> Note :
> After a speaker becomes a listener, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for the operation

pickSeat

This API is used to place a user in a seat (called by room owner).

> Note :
> After the room owner places a user in a seat, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

```
- (void)pickSeat:(NSInteger)seatIndex userId:(NSString *)userId callback:(ActionCallback _Nullabl
e)callback NS_SWIFT_NAME(pickSeat(seatIndex:userId:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to place the user in
userId	NSString	User ID

Parameter	Type	Description
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. In cases where the room owner needs listeners' consent to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept` , call `pickSeat` .

kickSeat

This API is used to remove a speaker (called by room owner).

> Note :
> After a speaker is removed, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.

```
- (void)kickSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(k
ickSeat(seatIndex:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	NSInteger	Seat number of the speaker to remove
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list.

muteSeat

This API is used to mute/unmute a seat (called by room owner).

> Note :
> After a seat is muted/unmuted, all members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.

```
- (void)muteSeat:(NSInteger)seatIndex isMute:(BOOL)isMute callback:(ActionCallback _Nullable)call
back NS_SWIFT_NAME(muteSeat(seatIndex:isMute:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to mute/unmute
isMute	BOOL	`YES` : mute the seat; `NO` : unmute the seat
callback	ActionCallback	Callback for the operation

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

closeSeat

This API is used to block/unblock a seat (called by room owner).

> Note :
> After a seat is blocked/unblocked, all members in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.

```
- (void)closeSeat:(NSInteger)seatIndex isClose:(BOOL)isClose callback:(ActionCallback _Nullable)c
allback NS_SWIFT_NAME(closeSeat(seatIndex:isClose:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	NSInteger	Number of the seat to block/unblock
isClose	BOOL	`YES` : block the seat; `NO` : unblock the seat
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will leave the seat.

Local Audio APIs

startMicrophone

This API is used to start mic capturing.

```
- (void)startMicrophone;
```

stopMicrophone

This API is used to stop mic capturing.

```
- (void)stopMicrophone;
```

setAudioQuality

This API is used to set audio quality.

```
- (void)setAuidoQuality:(NSInteger)quality NS_SWIFT_NAME(setAuidoQuality(quality:));
```

The parameters are as detailed below:

Parameter	Type	Description
quality	NSInteger	Audio quality. For more information, please see setAudioQuality().

muteLocalAudio

This API is used to mute/unmute local audio.

```
- (void)muteLocalAudio:(BOOL)mute NS_SWIFT_NAME(muteLocalAudio(mute:));
```

The parameters are as detailed below:

Parameter	Type	Description
mute	BOOL	Whether to mute or unmute audio. For more information, please see muteLocalAudio().

setSpeaker

This API is used to set whether to use the speaker or receiver.

```
- (void)setSpeaker:(BOOL)userSpeaker NS_SWIFT_NAME(setSpeaker(userSpeaker:));
```

The parameters are as detailed below:

Parameter	Type	Description
useSpeaker	BOOL	`YES` : speaker; `NO` : receiver

setAudioCaptureVolume

This API is used to set the mic capturing volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	NSInteger	Capturing volume. Value range: 0-100. Default value: 100

setAudioPlayoutVolume

This API is used to set the playback volume.

```
- (void)setAudioPlayoutVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioPlayoutVolume(volume:));
```

The parameters are as detailed below:

Parameter	Type	Description
volume	NSInteger	Playback volume. Value range: 0-100. Default value: 100

muteRemoteAudio

This API is used to mute/unmute a specified user.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	ID of the user to mute/unmute
mute	BOOL	`YES` : mute; `NO` : unmute

muteAllRemoteAudio

This API is used to mute/unmute all users.

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

The parameters are as detailed below:

Parameter	Type	Description
mute	BOOL	`YES` : mute; `NO` : unmute

setVoiceEarMonitorEnable

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

```
- (void)setVoiceEarMonitorEnable:(BOOL)enable NS_SWIFT_NAME(setVoiceEarMonitor(enable:));
```

The parameters are as detailed below:

Parameter	Type	Description
enable	BOOL	`YES` : enable in-ear monitoring; `NO` : disable in-ear monitoring

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

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

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a room, which is generally used for on-screen comments.

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

The parameters are as detailed below:

Parameter	Type	Description
message	NSString	Text message
callback	ActionCallback	Callback for the operation

sendRoomCustomMsg

This API is used to send a custom text message.

```
- (void)sendRoomCustomMsg:(NSString *)cmd message:(NSString *)message callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendRoomCustomMsg(cmd:message:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	NSString	Custom command word used to distinguish between different message types
message	NSString	Text message
callback	ActionCallback	Callback for the operation

Invitation Signaling APIs

sendInvitation

This API is used to send an invitation.

```
- (NSString *)sendInvitation:(NSString *)cmd
userId:(NSString *)userId
content:(NSString *)content
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendInvitation(cmd:userId:content:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
cmd	NSString	Custom command of business
userId	NSString	ID of the user to invite
content	NSString	Invitation content
callback	ActionCallback	Callback for the operation

Response parameters:

Returned Value	Type	Description
inviteId	NSString	Invitation ID

acceptInvitation

This API is used to accept an invitation.

```
- (void)acceptInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)callback NS_SW
IFT_NAME(acceptInvitation(id:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
id	NSString	Invitation ID
callback	ActionCallback	Callback for the operation

rejectInvitation

This API is used to decline an invitation.

```
- (void)rejectInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)callback NS_SW
IFT_NAME(rejectInvitation(id:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
id	NSString	Invitation ID
callback	ActionCallback	Callback for the operation

cancelInvitation

This API is used to cancel an invitation.

```
- (void)cancelInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(cancelInvitation(id:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
id	NSString	Invitation ID
callback	ActionCallback	Callback for the operation

TRTCVoiceRoomDelegate Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

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

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

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	NSString	Error message

onWarning

Callback for warning.

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

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	NSString	Warning message

onDebugLog

Callback for log.

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

The parameters are as detailed below:

Parameter	Type	Description
message	NSString	Log information

Room Event Callback APIs

onRoomDestroy

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.

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

The parameters are as detailed below:

Parameter	Type	Description
roomId	NSString	Room ID

onRoomInfoChange

Callback for change of room information. This callback is sent after successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.

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

The parameters are as detailed below:

Parameter	Type	Description
roomInfo	VoiceRoomInfo	Room information

onUserMicrophoneMute

Callback of whether a user's mic is muted. When a user calls `muteLocalAudio` , all members in the room will receive this callback.

```
- (void)onUserMicrophoneMute:(NSString *)userId mute:(BOOL)mute
NS_SWIFT_NAME(onUserMicrophoneMute(userId:mute:));
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID
mute	BOOL	`YES` : muted; `NO` : unmuted

onUserVolumeUpdate

Callback of the volume of each member in the room after the volume reminder is enabled.

```
- (void)onUserVolumeUpdate:(NSArray<TRTCVolumeInfo *> *)userVolumes totalVolume:(NSInteger)totalV
olume
NS_SWIFT_NAME(onUserVolumeUpdate(userVolumes:totalVolume:));
```

The parameters are as detailed below:

Parameter	Type	Description
userVolumes	NSArray	List of user volumes
totalVolume	NSInteger	Total volume. Value range: 0-100

Seat Callback APIs

onSeatListChange

Callback for all seat changes.

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

The parameters are as detailed below:

Parameter	Type	Description
seatInfoList	NSArray<VoiceRoomSeatInfo>	Full seat list

onAnchorEnterSeat

Someone became a speaker or was made a speaker by the room owner.

```
- (void)onAnchorEnterSeat:(NSInteger)index
user:(VoiceRoomUserInfo *)user
NS_SWIFT_NAME(onAnchorEnterSeat(index:user:));
```

The parameters are as detailed below:

Parameter	Type	Description
index	NSInteger	Number of the seat taken
user	VoiceRoomUserInfo	Information of the user who took the seat

onAnchorLeaveSeat

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

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

The parameters are as detailed below:

Parameter	Type	Description
index	NSInteger	Number of the seat the user left

Parameter	Type	Description
user	VoiceRoomUserInfo	Information of the user who left the seat

onSeatMute

The room owner muted/unmuted a seat.

```
- (void)onSeatMute:(NSInteger)index
isMute:(BOOL)isMute
NS_SWIFT_NAME(onSeatMute(index:isMute:));
```

The parameters are as detailed below:

Parameter	Type	Description
index	NSInteger	The seat muted/unmuted
isMute	BOOL	`YES: The seat was muted;` NO`: The seat was unmuted.

onSeatClose

The room owner blocked/unblocked a seat.

```
- (void)onSeatClose:(NSInteger)index
isClose:(BOOL)isClose
NS_SWIFT_NAME(onSeatClose(index:isClose:));
```

The parameters are as detailed below:

Parameter	Type	Description
index	NSInteger	The seat blocked/unblocked
isClose	BOOL	`YES` : The seat was blocked; `NO` : The seat was unblocked.

Callback APIs for Room Entry/Exit by Listener

onAudienceEnter

A listener entered the room.

```
- (void)onAudienceEnter:(VoiceRoomUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceEnter(userInfo:));
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	VoiceRoomUserInfo	Information of the listener who entered

onAudienceExit

A listener exited the room.

```
- (void)onAudienceExit:(VoiceRoomUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceExit(userInfo:));
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	VoiceRoomUserInfo	Information of the user who left

Message Event Callback APIs

onRecvRoomTextMsg

Callback for receiving a text message.

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

The parameters are as detailed below:

Parameter	Type	Description
message	NSString	Text message
userInfo	VoiceRoomUserInfo	Information of the sender

onRecvRoomCustomMsg

Callback for receiving a custom message.

```
- (void)onRecvRoomCustomMsg:(NSString *)command
message:(NSString *)message
```

```
userInfo:(VoiceRoomUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomCustomMsg(command:message:userInfo:));
```

The parameters are as detailed below:

Parameter	Type	Description
command	NSString	Custom command word used to distinguish between different message types
message	NSString	Text message
userInfo	VoiceRoomUserInfo	Information of the sender

Invitation Signaling Callback APIs

onReceiveNewInvitation

An invitation was received.

```
- (void)onReceiveNewInvitation:(NSString *)identifier
inviter:(NSString *)inviter
cmd:(NSString *)cmd
content:(NSString *)content
NS_SWIFT_NAME(onReceiveNewInvitation(id:inviter:cmd:content:));
```

The parameters are as detailed below:

Parameter	Type	Description
id	NSString	Invitation ID
inviter	NSString	Inviter’s user ID
cmd	NSString	Custom command word specified by business
content	NSString	Content specified by business

onInviteeAccepted

The invitee accepted the invitation

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

The parameters are as detailed below:

Parameter	Type	Description
id	NSString	Invitation ID
invitee	NSString	Invitee’s user ID

onInviteeRejected

The invitee declined the invitation

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

The parameters are as detailed below:

Parameter	Type	Description
id	NSString	Invitation ID
invitee	NSString	Invitee’s user ID

onInvitationCancelled

The inviter canceled the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	NSString	Invitation ID
inviter	NSString	Inviter’s user ID

TRTCVoiceRoom (Android)

Last updated : 2022-01-05 15:51:32

`TRTCVoiceRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create an audio chat room and become a speaker, or enter an audio chat room as a listener.
- The room owner can invite a listener to speak as well as remove a speaker.
- The room owner can also block a seat. Listeners cannot request to take a blocked seat.
- A listener can request to speak and become a speaker. A speaker can also become a listener.
- All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

`TRTCVoiceRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Audio Chat Room (Android).

- TRTC SDK: the TRTC SDK is used as a low-latency audio chat component.
- IM SDK: the `AVChatRoom` feature of the IM SDK is used to implement chat rooms. The attribute APIs of IM are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

TRTCVoiceRoom API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
setDelegate	Sets event callbacks.
setDelegateHandler	Sets the thread where event callbacks are.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
destroyRoom	Terminates a room (called by room owner).
enterRoom	Enters a room (called by listener).
exitRoom	Exits a room (called by listener).
getRoomInfoList	Gets room list details.
getUserInfoList	Gets the user information of the specified `userId`. If the value is `null`, the information of all users in the room is obtained.

Seat management APIs

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

Local audio APIs

API	Description
startMicrophone	Starts mic capturing.
stopMicrophone	Stops mic capturing.
setAudioQuality	Sets audio quality.
muteLocalAudio	Mutes/Unmutes local audio.

API	Description
setSpeaker	Sets whether to use the speaker or receiver.
setAudioCaptureVolume	Sets mic capturing volume.
setAudioPlayoutVolume	Sets playback volume.
setVoiceEarMonitorEnable	Enables/Disables in-ear monitoring.

Remote audio APIs

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

Background music and audio effect APIs

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

Message sending APIs

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

Invitation signaling APIs

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

TRTCVoiceRoomDelegate API Overview

Common event callback APIs

API	Description
onError	Error
onWarning	Warning
onDebugLog	Log

Room event callback APIs

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

Seat list change callback APIs

API	Description
onSeatListChange	All seat changes
onAnchorEnterSeat	Someone became a speaker or was made a speaker by the room owner.
onAnchorLeaveSeat	Someone became a listener or was moved to listeners by the room owner.
onSeatMute	The room owner muted a seat.
onUserMicrophoneMute	Whether a user’s mic is muted
onSeatClose	The room owner blocked a seat.

Callback APIs for room entry/exit by listener

API	Description
onAudienceEnter	A listener entered the room.

API	Description
onAudienceExit	A listener exited the room.

Message event callback APIs

API	Description
onRecvRoomTextMsg	Receipt of a text message
onRecvRoomCustomMsg	A custom message was received.

Signaling Event Callback APIs

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

Basic SDK APIs

sharedInstance

This API is used to get a TRTCVoiceRoom singleton object.

```
public static synchronized TRTCVoiceRoom sharedInstance(Context context);
```

The parameters are as detailed below:

Parameter	Type	Description
context	Context	Android context, which will be converted to `ApplicationContext` for the calling of system APIs

destroySharedInstance

This API is used to terminate a TRTCVoiceRoom singleton object.

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

```
public static void destroySharedInstance();
```

setDelegate

This API is used to set the event callback of TRTCVoiceRoom. You can use `TRTCVoiceRoomDelegate` to get different status notifications of TRTCVoiceRoom.

```
public abstract void setDelegate(TRTCVoiceRoomDelegate delegate);
```

> Note :
> `setDelegate` is the delegate callback of `TRTCVoiceRoom` .

setDelegateHandler

This API is used to set the thread where event callbacks are.

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

The parameters are as detailed below:

Parameter	Type	Description
handler	Handler	The status notifications of `TRTCVoiceRoom` are sent to the handler thread you specify.

login

This API is used to log in to the Tencent backend server.

```
public abstract void login(int sdkAppId,
String userId, String userSig,
TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	int	You can view `SDKAppID` in **Application Management** > **Application Info** of the TRTC console.
userId	String	ID of 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 protection signature. For more information on how to get it, please see How to Calculate UserSig.
callback	ActionCallback	Callback for login. The code is 0 if login succeeds.

logout

This API is used to log out of the Tencent backend server.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for logout. The code is `0` if logout succeeds.

setSelfProfile

This API is used to set the profile.

```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userName	String	Username
avatar	String	Profile photo address
callback	ActionCallback	Callback for profile setting. The code is `0` if the operation succeeds.

Room APIs

createRoom

This API is used to create a room (called by room owner).

```
public abstract void createRoom(int roomId, TRTCVoiceRoomDef.RoomParam roomParam, TRTCVoiceRoomCa
llback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a karaoke room list. Currently, Tencent Cloud does not provide management services for room lists. Please manage the list on your own.
roomParam	TRTCCreateRoomParam	Room information, such as room name, seat list information, and cover information. To manage seats, you must enter the number of seats in the room.
callback	ActionCallback	Callback for room creation. The code is 0 if the operation succeeds.

The process of creating a karaoke room and becoming a speaker is as follows:

1. A user calls `createRoom` to create an audio chat room, passing in room attributes (e.g. room ID, whether listeners require room owner’s consent to speak, number of seats).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
4. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

destroyRoom

This API is used to terminate a room (called by room owner).

```
public abstract void destroyRoom(TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

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

enterRoom

This API is used to enter a room (called by listener).

```
public abstract void enterRoom(int roomId, TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

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

The process of entering a room as a listener is as follows:

1. A user gets the latest audio chat room list from your server. The list may contain the `roomId` and room information of multiple audio chat rooms.
2. The user selects a room, and calls `enterRoom` with the room ID passed in to enter the room.
3. After entering the room, the user receives an `onRoomInfoChange` notification about room attribute change from the component. The attributes can be recorded, and corresponding changes can be made to the UI, including room name, whether room owner's consent is required for listeners to speak, etc.
4. The user will receive an `onSeatListChange` notification about the change of the seat list and can update the change to the UI.
5. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker.

exitRoom

This API is used to exit a room.

```
public abstract void exitRoom(TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description

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

getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()` .

> Note :
> You don’t need this API if both the room list and room information are managed on your server.

```
public abstract void getRoomInfoList(List<Integer> roomIdList, TRTCVoiceRoomCallback.RoomInfoCall
back callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomIdList	List<Integer>	Room ID list
callback	RoomInfoCallback	Callback of room details

getUserInfoList

This API is used to get the user information of a specified `userId` .

```
public abstract void getUserInfoList(List<String> userIdList, TRTCVoiceRoomCallback.UserListCallb
ack userlistcallback);
```

The parameters are as detailed below:

Parameter	Type	Description
userIdList	List<String>	IDs of the users to query. If this parameter is `null` , the information of all users in the room is queried.
userlistcallback	UserListCallback	Callback of user details

Seat Management APIs

enterSeat

This API is used to become a speaker (called by room owner or listener).

> Note :
> After a user becomes a speaker, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

```
public abstract void enterSeat(int seatIndex, TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to take
callback	ActionCallback	Callback for the operation

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

moveSeat

This API is used to change one's seat (called by speaker).

> Note :
> After the seat change, all users in the room will receive the `onSeatListChange` , `onAnchorLeaveSeat` , and `onAnchorEnterSeat` notifications. This API will only change the user's seat number, not the user role.

```
public abstract int moveSeat(int seatIndex, TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
seatIndex	int	The number of the seat to change to
callback	ActionCallback	Callback for the operation

Response parameters:

Parameter	Type	Description
code	int	Result of seat change. `0` : operation successful; `10001` : API rate limit exceeded; other values: operation failed

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

leaveSeat

This API is used to become a listener (called by speaker).

> Note :
> After a speaker becomes a listener, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.

```
public abstract void leaveSeat(TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for the operation

pickSeat

This API is used to place a user in a seat (called by room owner).

> Note :

After the room owner places a user in a seat, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

```
public abstract void pickSeat(int seatIndex, String userId, TRTCVoiceRoomCallback.ActionCallback
callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to place the listener in
userId	String	User ID
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. In cases where the room owner needs listeners' consent to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept` , call `pickSeat` .

kickSeat

This API is used to remove a speaker (called by room owner).

Note :
After a speaker is removed, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.

```
public abstract void kickSeat(int seatIndex, TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to remove the speaker from
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list.

muteSeat

This API is used to mute/unmute a seat (called by room owner).

> Note :
> After a seat is muted/unmuted, all members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.

```
public abstract void muteSeat(int seatIndex, boolean isMute, TRTCVoiceRoomCallback.ActionCallback
callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to block/unblock
isMute	boolean	`true` : mute; `false` : unmute
callback	ActionCallback	Callback for the operation

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

closeSeat

This API is used to block/unblock a seat (called by room owner).

> Note :
> After a seat is blocked/unblocked, all members in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.

```
public abstract void closeSeat(int seatIndex, boolean isClose, TRTCVoiceRoomCallback.ActionCallba
ck callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to block/unblock

Parameter	Type	Description
isClose	boolean	`true` : block; `false` : unblock
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will leave the seat.

Local Audio APIs

startMicrophone

This API is used to start mic capturing.

```
public abstract void startMicrophone();
```

stopMicrophone

This API is used to stop mic capturing.

```
public abstract void stopMicrophone();
```

setAudioQuality

This API is used to set audio quality.

```
public abstract void setAudioQuality(int quality);
```

The parameters are as detailed below:

Parameter	Type	Description
quality	int	Audio quality. For more information, please see setAudioQuality().

muteLocalAudio

This API is used to mute/unmute local audio.

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

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	Whether to mute or unmute audio. For more information, please see muteLocalAudio().

setSpeaker

This API is used to set whether to use the speaker or receiver.

```
public abstract void setSpeaker(boolean useSpeaker);
```

The parameters are as detailed below:

Parameter	Type	Description
useSpeaker	boolean	`true` : speaker; `false` : receiver

setAudioCaptureVolume

This API is used to set the mic capturing volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Capturing volume. Value range: 0-100 (default: 100)

setAudioPlayoutVolume

This API is used to set the playback volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Playback volume. Value range: 0-100 (default: 100)

muteRemoteAudio

This API is used to mute/unmute a specified user.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
mute	boolean	`true` : mute; `false` : unmute

muteAllRemoteAudio

This API is used to mute/unmute all users.

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

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	`true` : mute; `false` : unmute

setVoiceEarMonitorEnable

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

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

The parameters are as detailed below:

Parameter	Type	Description
enable	boolean	`true` : enable; `false` : disable

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

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

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a room, which is generally used for on-screen comments.

```
public abstract void sendRoomTextMsg(String message, TRTCVoiceRoomCallback.ActionCallback callbac
k);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
callback	ActionCallback	Callback for the operation

sendRoomCustomMsg

This API is used to send a custom text message.

```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCVoiceRoomCallback.ActionCa
llback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message
callback	ActionCallback	Callback for the operation

Invitation Signaling APIs

sendInvitation

This API is used to send an invitation.

```
public abstract String sendInvitation(String cmd, String userId, String content, TRTCVoiceRoomCal
lback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command of business
userId	String	Invitee’s user ID
content	String	Invitation content
callback	ActionCallback	Callback for the operation

Response parameters:

Returned Value	Type	Description
inviteId	String	Invitation ID

acceptInvitation

This API is used to accept an invitation.

```
public abstract void acceptInvitation(String id, TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

rejectInvitation

This API is used to decline an invitation.

```
public abstract void rejectInvitation(String id, TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

cancelInvitation

This API is used to cancel an invitation.

```
public abstract void cancelInvitation(String id, TRTCVoiceRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

TRTCVoiceRoomDelegate Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

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

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

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Error message

onWarning

Callback for warning.

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

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Warning message

onDebugLog

Callback for log.

```
void onDebugLog(String message);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Log information

Room Event Callback APIs

onRoomDestroy

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.

```
void onRoomDestroy(String roomId);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	String	Room ID

onRoomInfoChange

Callback for change of room information. This callback is sent after successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.

```
void onRoomInfoChange(TRTCVoiceRoomDef.RoomInfo roomInfo);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
roomInfo	RoomInfo	Room information

onUserMicrophoneMute

Callback of whether a user's mic is muted. When a user calls `muteLocalAudio` , all members in the room will receive this callback.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
mute	boolean	`true` : muted; `false` : unmuted

onUserVolumeUpdate

Notification to all members of the volume after the volume reminder is enabled.

```
void onUserVolumeUpdate(List<TRTCCloudDef.TRTCVolumeInfo> userVolumes, int totalVolume);
```

The parameters are as detailed below:

Parameter	Type	Description
userVolumes	ListList<trtcclouddef.trtcvolumeinfo>	List of user IDs
totalVolume	int	Total volume. Value range: 0-100

Seat Callback APIs

onSeatListChange

Callback for all seat changes.

```
void onSeatListChange(List<SeatInfo> seatInfoList);
```

The parameters are as detailed below:

Parameter	Type	Description
seatInfoList	List<SeatInfo>	Full seat list

onAnchorEnterSeat

Someone became a speaker or was made a speaker by the room owner.

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

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat taken
user	UserInfo	Details of the user who took the seat

onAnchorLeaveSeat

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

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

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat previously occupied by the speaker
user	UserInfo	Details of the user who took the seat

onSeatMute

The room owner muted/unmuted a seat.

```
void onSeatMute(int index, boolean isMute);
```

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat muted/unmuted
isMute	boolean	`true` : muted; `false` : unmuted

onSeatClose

The room owner blocked/unblocked a seat.

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

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat blocked/unblocked
isClose	boolean	`true` : blocked; `false` : unblocked

Callback APIs for Room Entry/Exit by Listener

onAudienceEnter

A listener entered the room.

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

The parameters are as detailed below:

Parameter	Type	Description
userInfo	UserInfo	Information of the listener who entered the room

onAudienceExit

A listener exited the room.

```
void onAudienceExit(TRTCVoiceRoomDef.UserInfo userInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	UserInfo	Information of the listener who exited the room

Message Event Callback APIs

onRecvRoomTextMsg

Callback for receiving a text message.

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

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
userInfo	UserInfo	Information of the sender

onRecvRoomCustomMsg

Callback for receiving a custom message.

```
void onRecvRoomCustomMsg(String cmd, String message, TRTCVoiceRoomDef.UserInfo userInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message
userInfo	UserInfo	Information of the sender

Invitation Signaling Callback APIs

onReceiveNewInvitation

An invitation was received.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID

Parameter	Type	Description
inviter	String	Inviter’s user ID
cmd	String	Custom command word specified by business
content	String	Content specified by business

onInviteeAccepted

The invitee accepted the invitation

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee’s user ID

onInviteeRejected

The invitee declined the invitation

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee’s user ID

onInvitationCancelled

The inviter canceled the invitation.

```
void onInvitationCancelled(String id, String inviter);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
id	String	Invitation ID
inviter	String	Inviter’s user ID
</trtcclouddef.trtcvolumeinfo>		

Group Audio/Video Room

Group Audio/Video Room (iOS)

Last updated : 2022-03-18 22:59:30

Demo UI

You can download and install the demo app we provide to try out TRTC’s video call features, including screen sharing, beauty filters, and low-latency video call.

Solution Strengths

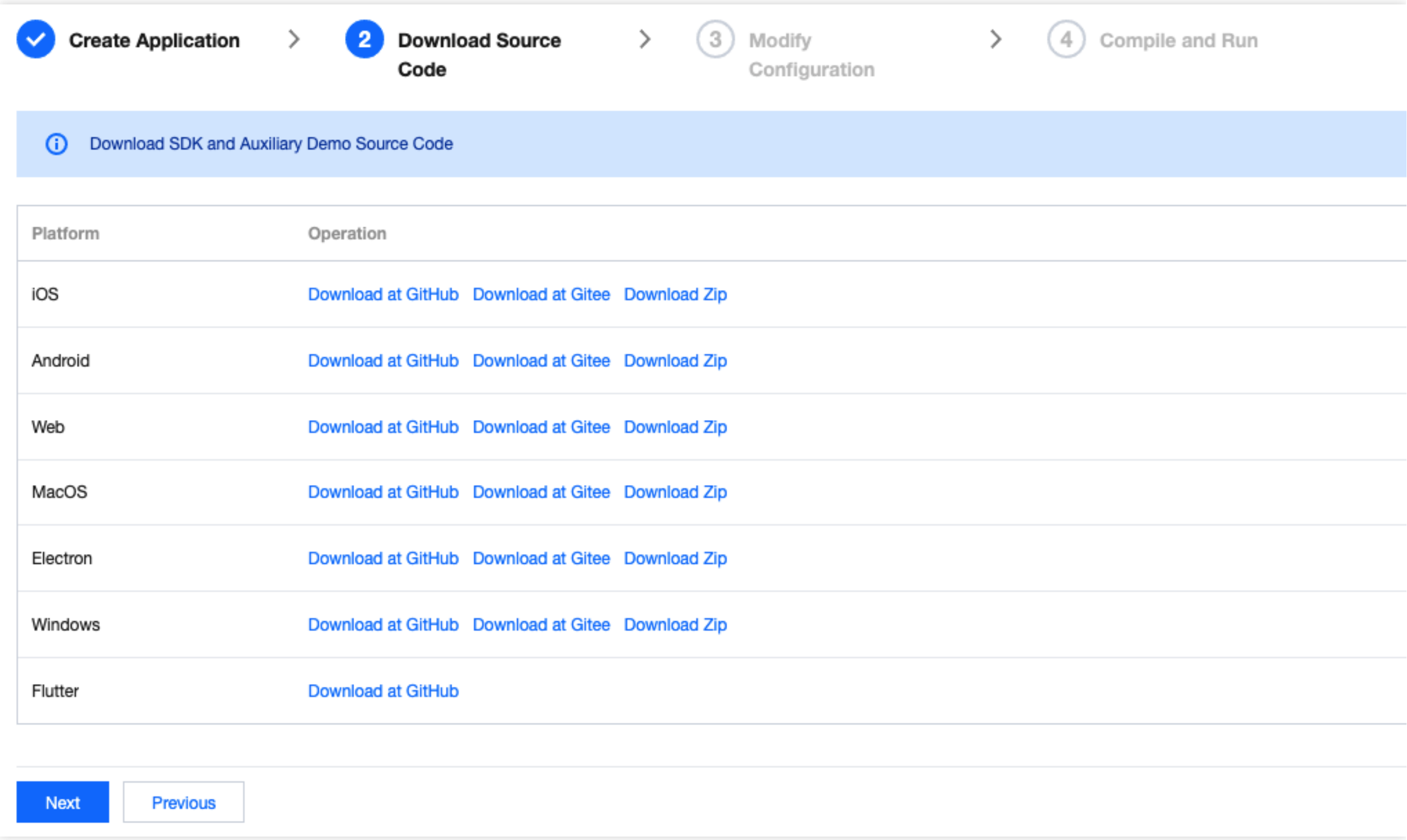
- `TUIRoom` integrates various capabilities such as ultra-low-latency audio/video calling, screen sharing, and beauty filters, covering the common features of group audio/video room.
- It can be further developed as needed to quickly implement custom UI and layout, helping you quickly launch your business.
- It encapsulates the basic TRTC and IM SDKs to implement basic logic control and provides APIs for you to call features easily.

Connection Guide

To quickly implement the group audio/video room feature, you can modify the demo app we provide and adapt it to your needs. You can also use the `Module` module in the app and customize your own UI.

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestTUIRoom` and click **Create**.
3. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Note :
This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the application source code

Click TUIRoom to clone or download the source code.

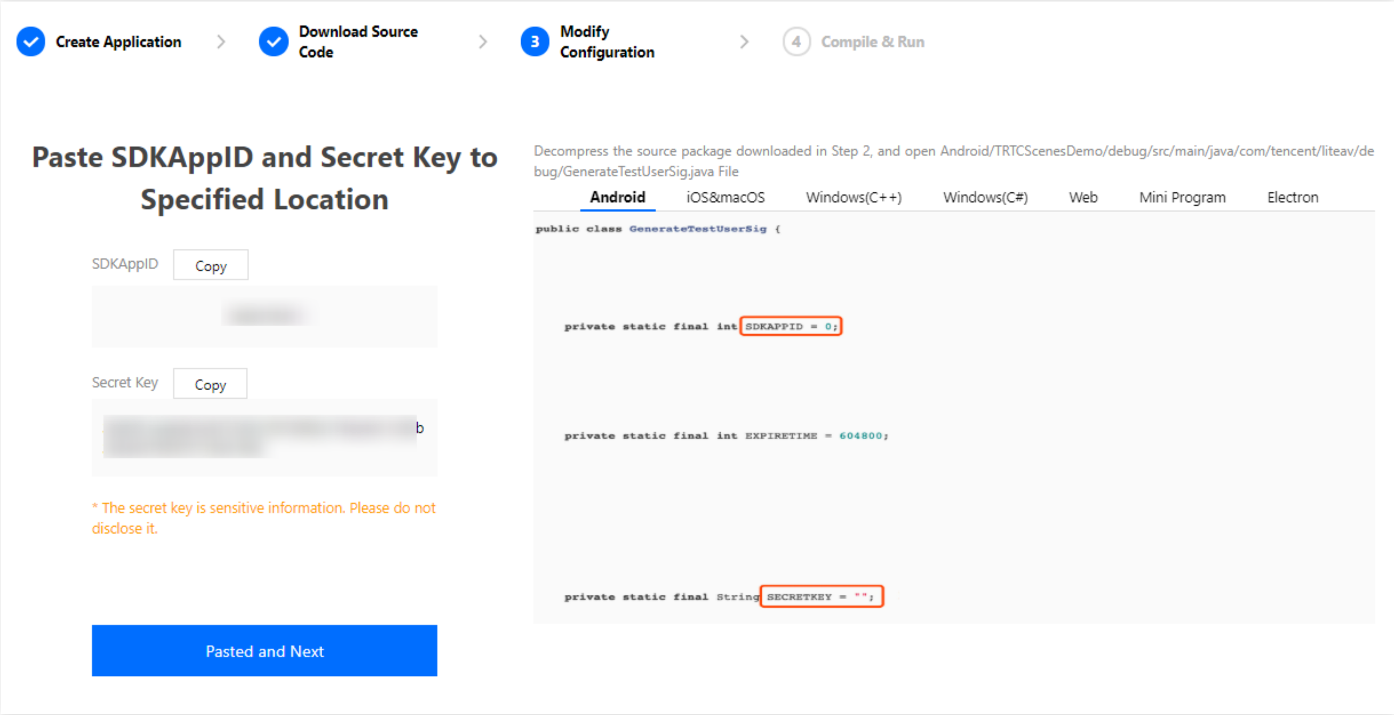
Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `Example/Debug/GenerateTestUserSig.swift` .

3. Set the following parameters in `GenerateTestUserSig.swift` :

   - SDKAPPID: `0` by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: left empty by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Pasted and Next

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Run the project

Open `Example/DemoApp.xcworkspace` with Xcode (11.0 or above) and click **Run**.

Step 5. Modify the project source code

The `Source` folder in the source code contains the UI folder, which contains UI code. The table below lists the folders and the UI views they represent. You can refer to it when making UI changes.

Folder	Description
TUIRoomEnter	Implementation code for the `TUIRoom` entry. The `TUIRoomEntranceViewController` class is a public CocoaPods class.
TUIRoomMain	Implementation code for the `TUIRoom` main UI. This is a private CocoaPods class.
TUIRoomMemberList	Implementation code for the participant list view. This is a private CocoaPods class.
TUIRoomSet	Implementation code for the settings UI. This is a private CocoaPods class.

Tryout

> Note :
>
> You need at least two devices to try out the application.

Entry page

Select **Create Room** or **Enter Room**.

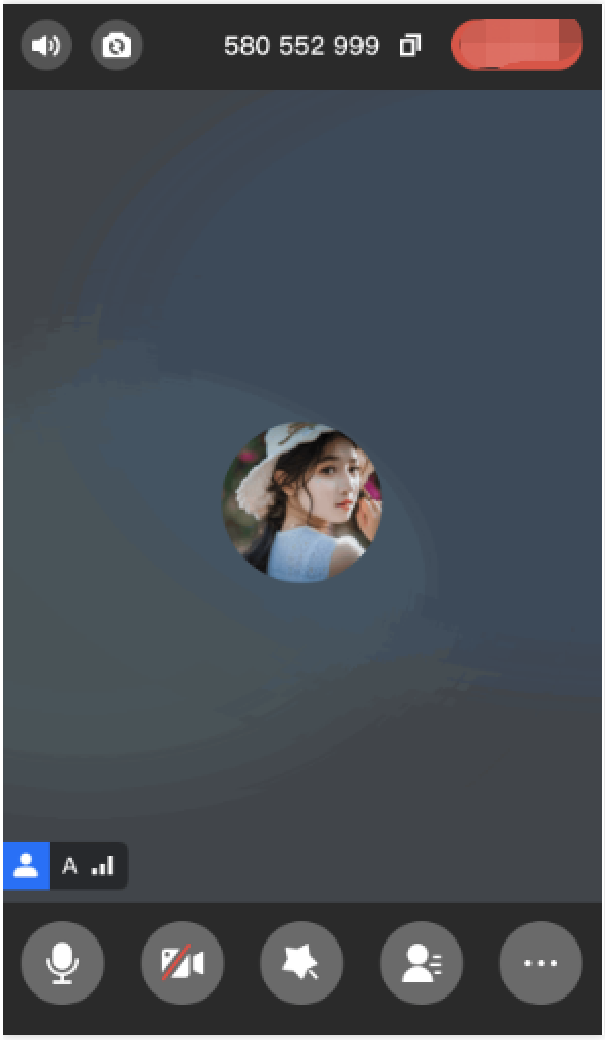
Room creation page

When user A creates a room, the room ID will be automatically generated. User A can tap **Create Room** to access the main UI.

Room entry page

User B enters the ID of the room created by user A and taps **Enter Room** to access the main UI.

Main UI (user A)



Mic-on list

It displays all users in the current room. If a user enables the camera and mic, you can watch the video image and hear the voice of the user.

Top toolbar

It implements features such as mic switch, front/rear camera switch, room information display, and room exit.

Bottom toolbar

It implements features such as local mic/camera control, beauty filter control, user list, and settings.

Beauty filter

Beauty filters and special effects can be displayed on the screen during live streaming.

Settings window

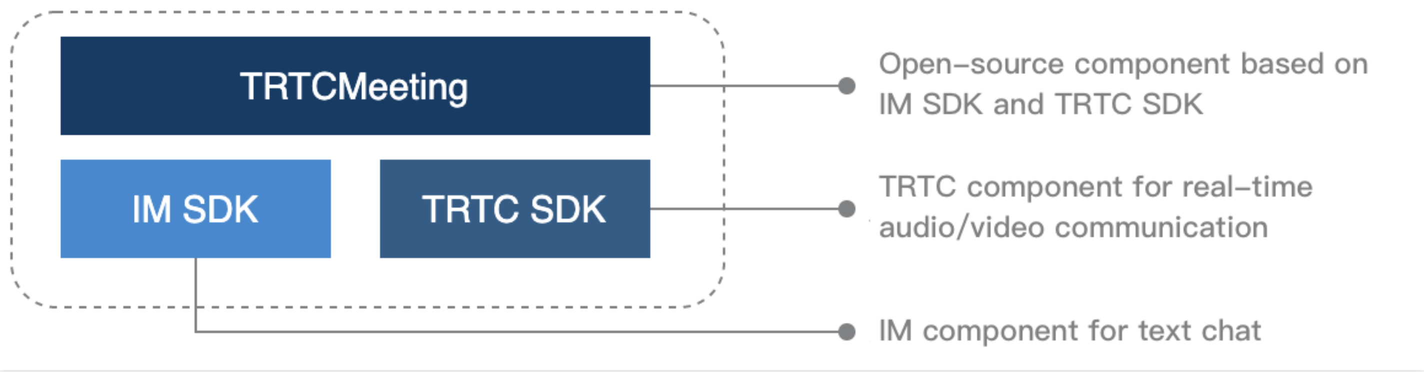
Audio and video parameters can be set, and **screen sharing** can be enabled.

Exiting a room

- **Anchor**: dismisses the room, so all users need to exit the room.
- **Non-anchor**: leaves the room.

Customizing UI

The `Source` folder in the source code contains `UI` and `Presenter` folders. The `Presenter` folder contains the reusable open-source component `TUIRoomCore` . You can find the component's APIs in `TUIRoomCore.h` and use them to customize your own UI.


TRTCMeeting
IM SDK
TRTC SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for text chat

Step 1. Integrate the SDK

The group audio/video room component `TUIRoom` depends on the TRTC SDK and IM SDK. Follow the steps below to integrate the two SDKs into your project:

- **Method 1: adding dependencies via CocoaPods**

```
pod 'TXIMSDK_iOS'
pod 'TXLiteAVSDK_TRTC'
```

> Note :
> You can view the latest version numbers of the two SDKs by visiting their GitHub pages at TRTC and IM.

- **Method 2: adding dependencies through local files**
  If your access to the CocoaPods repository is slow, you can download the ZIP files of the SDKs and

manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration Documentation
IM SDK	Download	Integration Documentation

Step 2. Configure permission requests

Configure camera and mic permission requests by adding `Privacy > Camera Usage Description` and `Privacy > Microphone Usage Description` in `info.plist`.

Step 3. Import the `TUIRoom` component

To import the component through CocoaPods, follow the steps below:

1. Copy the `Source`, `Resources`, `TCBeautyKit`, and `TXAppBasic` folders and the `TUIRoom.podspec` file from the demo directory to your project directory.
2. Add the following dependencies to your `Podfile` and run `pod install` to complete the import.

```
# Local dependency library
def local
pod 'TXAppBasic', :path => "../../TXAppBasic/"
pod 'TCBeautyKit', :path => "../../TCBeautyKit/"
pod 'TXLiteAVSDK_TRTC',:path => "../../../SDK/"
end
def pod_local(type)
loadLocalPod('TUIRoom', type)
end
def loadLocalPod(name, type)
pod "#{name}/#{type}", :path => "../"
end
target 'DemoApp' do
local
pod_local('TRTC')
end
```

Step 4. Create an instance and log in

1. Call `TUILogin` in `TUICore` to log in as shown below:

```
TUILogin.initWithSdkAppID(Int32(SDKAPPID))
let userID = ProfileManager.shared().curUserID()
let userSig = GenerateTestUserSig.genTestUserSig(userID)
TUILogin.login(userID, userSig: userSig, succ: {
```

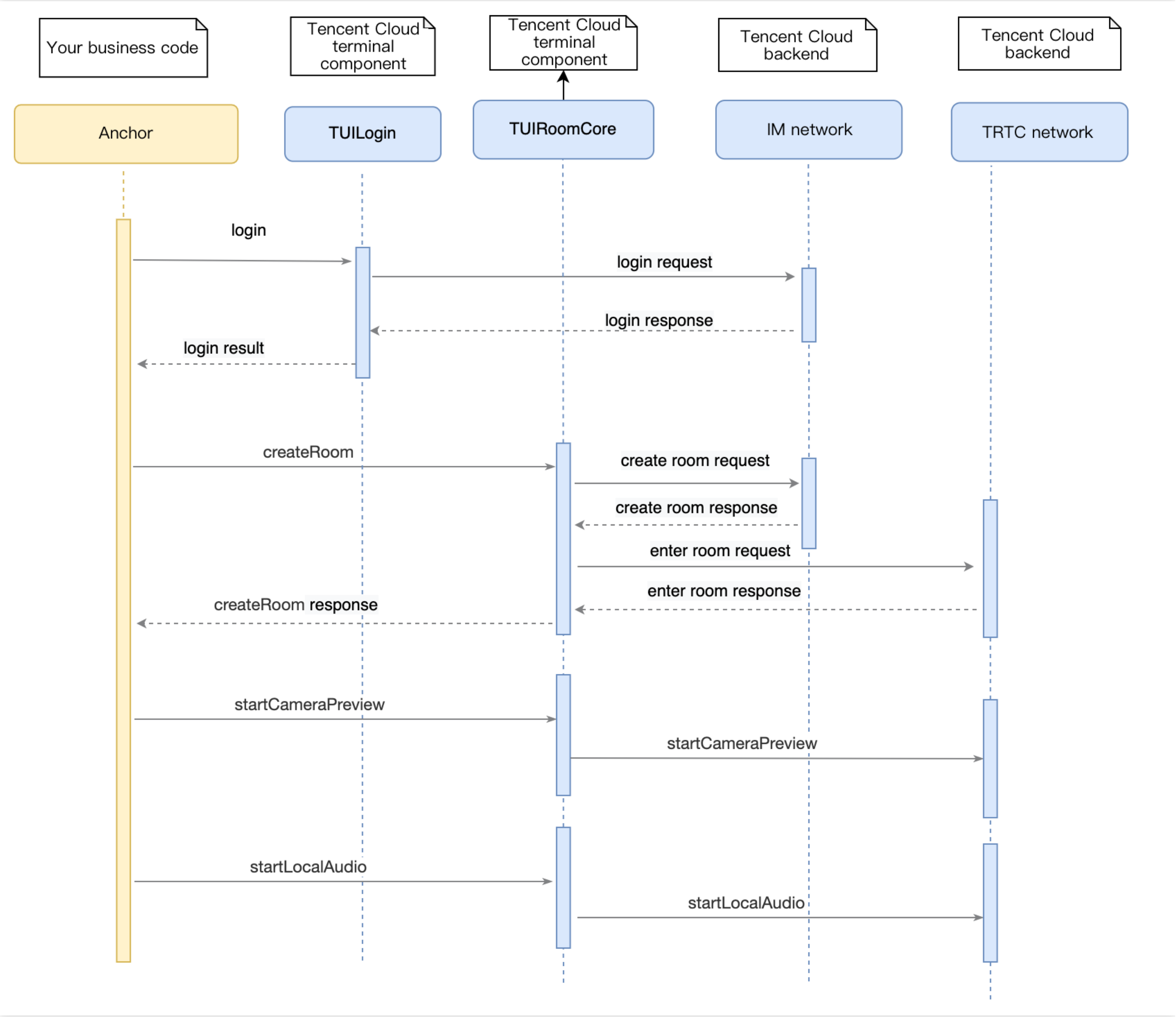
```
debugPrint("login success")
}, fail: { (code, errorDes) in
debugPrint("login failed, code:¥(code), error: ¥(errorDes ?? "nil")")
})
```

2. After logging in, call `TUIRoomCore` to create a room.

```
let roomId = 123
TUIRoomCore.shareInstance().createRoom("¥(roomId)",speechMode: .freeSpeech,callback: { [weak self] code, message in
if code == 0 {
debugPrint("create room success")
} else {
}
})
```

3. Enter the main UI after successful creation.

```
let vc = TUIRoomMainViewController(roomId: roomId, isVideoOn: openCameraSwitch.isOn, isAudioOn: openMicSwitch.isOn)
TUIRoomCore.shareInstance().setDelegate(vc)
navigationController?.pushViewController(vc, animated: true)
```



Step 5. Members enter the room

1. Members call `TUILogin` in `TUICore` to log in as shown below:

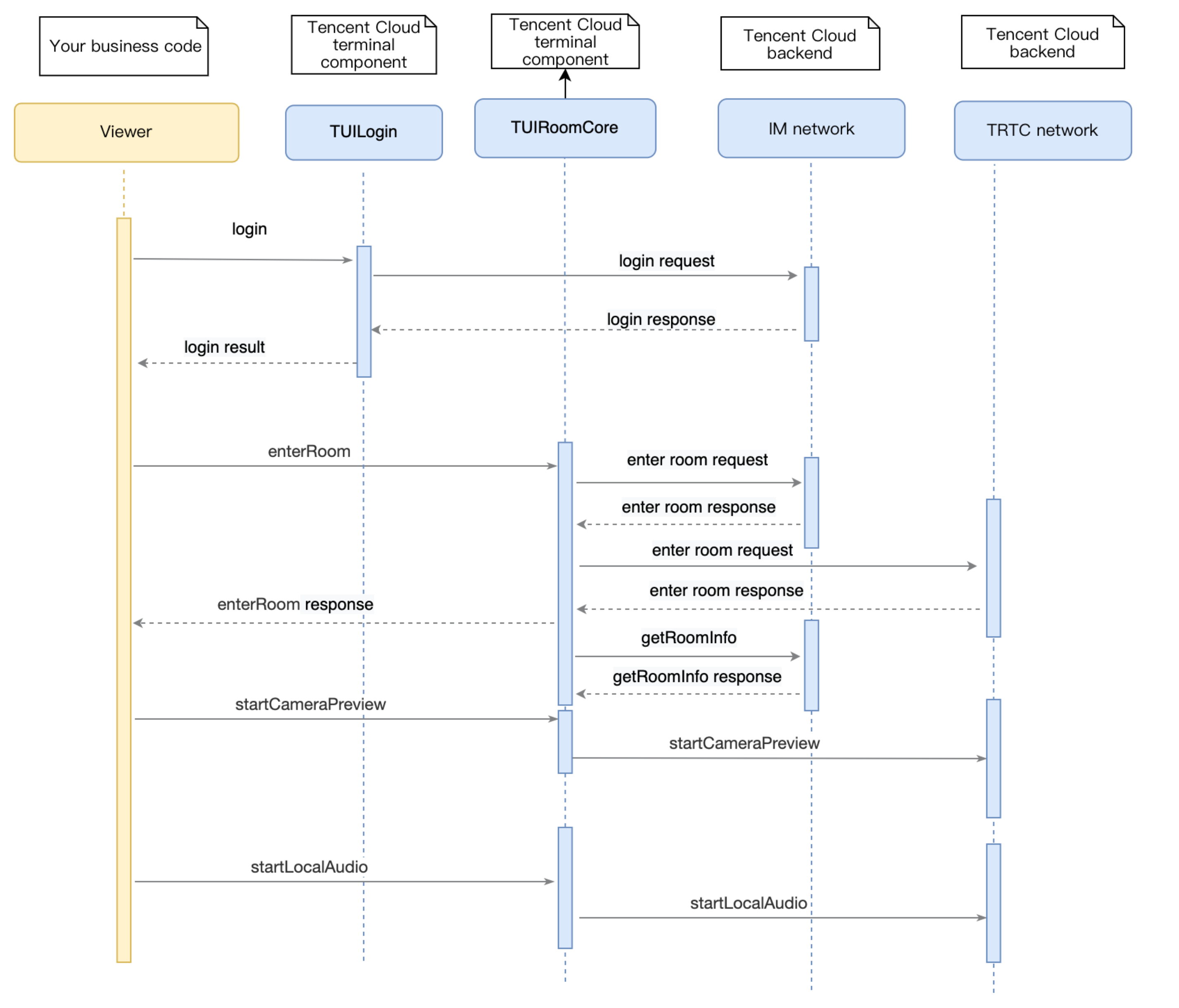
```
TUILogin.initWithSdkAppID(Int32(SDKAPPID))
let userID = ProfileManager.shared().curUserID()
let userSig = GenerateTestUserSig.genTestUserSig(userID)
TUILogin.login(userID, userSig: userSig, succ: {
debugPrint("login success")
}, fail: { (code, errorDes) in
debugPrint("login failed, code:¥(code), error: ¥(errorDes ?? "nil")")
})
```

2. After logging in, call `TUIRoomCrre` to enter a room.

```
let roomId = 123
TUIRoomCore.shareInstance().enterRoom(”¥(roomId)”, callback: { [weak self] code, message in
if code == 0 {
debugPrint(”enter room success”)
} else {
}
})
```

3. Enter the main UI after successful room entry.

```
let vc = TUIRoomMainViewController(roomId: roomId, isVideoOn: openCameraSwitch.isOn, isAudioOn: openMicSwitch.isOn)
TUIRoomCore.shareInstance().setDelegate(vc)
navigationController?.pushViewController(vc, animated: true)
```



Step 6. Enable screen sharing.

1. Call `startScreenCapture` of `TUIRoomCore` to share the screen.
2. Other members in the room will receive the `onRemoteUserScreenVideoAvailable` event notification.

```
// Click the button to enable screen sharing.
if #available(iOS 12.0, *) {
// Start screen sharing.
let params = TRTCVideoEncParam()
params.videoResolution = TRTCVideoResolution._1280_720
params.resMode = TRTCVideoResolutionMode.portrait
params.videoFps = 10
params.enableAdjustRes = false
params.videoBitrate = 1500
TUIRoomCore.shareInstance().startScreenCapture(params)
TUIRoomBroadcastExtensionLauncher.launch()
} else {
view.window?.makeToast(.versionLowToastText)
}
```

Step 7. Exit a room

- The **anchor** calls the `destoryRoom` API to dismiss the room and IM group chat and exit the TRTC room. Room members receive the `onDestroyRoom` callback message that notifies them of group dismissal and exit the TRTC room.
- A **member** calls the `leaveRoom` API to leave the room and IM group chat and exit the TRTC room. Other room members receive the `onRemoteUserLeave` callback message that notifies them of the member exiting the room.

```
if isHomeowner {
TUIRoomCore.shareInstance().destroyRoom { [weak self] _, _ in
guard let self = self else { return }
self.navigationController?.popViewController(animated: true)
}
} else {
TUIRoomCore.shareInstance().leaveRoom { [weak self] _, _ in
guard let self = self else { return }
self.navigationController?.popViewController(animated: true)
}
}
```

Group Audio/Video Room (Android)

Last updated : 2022-03-24 09:34:04

Demo UI

You can download and install the demo app we provide to try out TRTC’s group audio/video call features, including screen sharing, beauty filters, and low-latency video call.

Solution Strengths

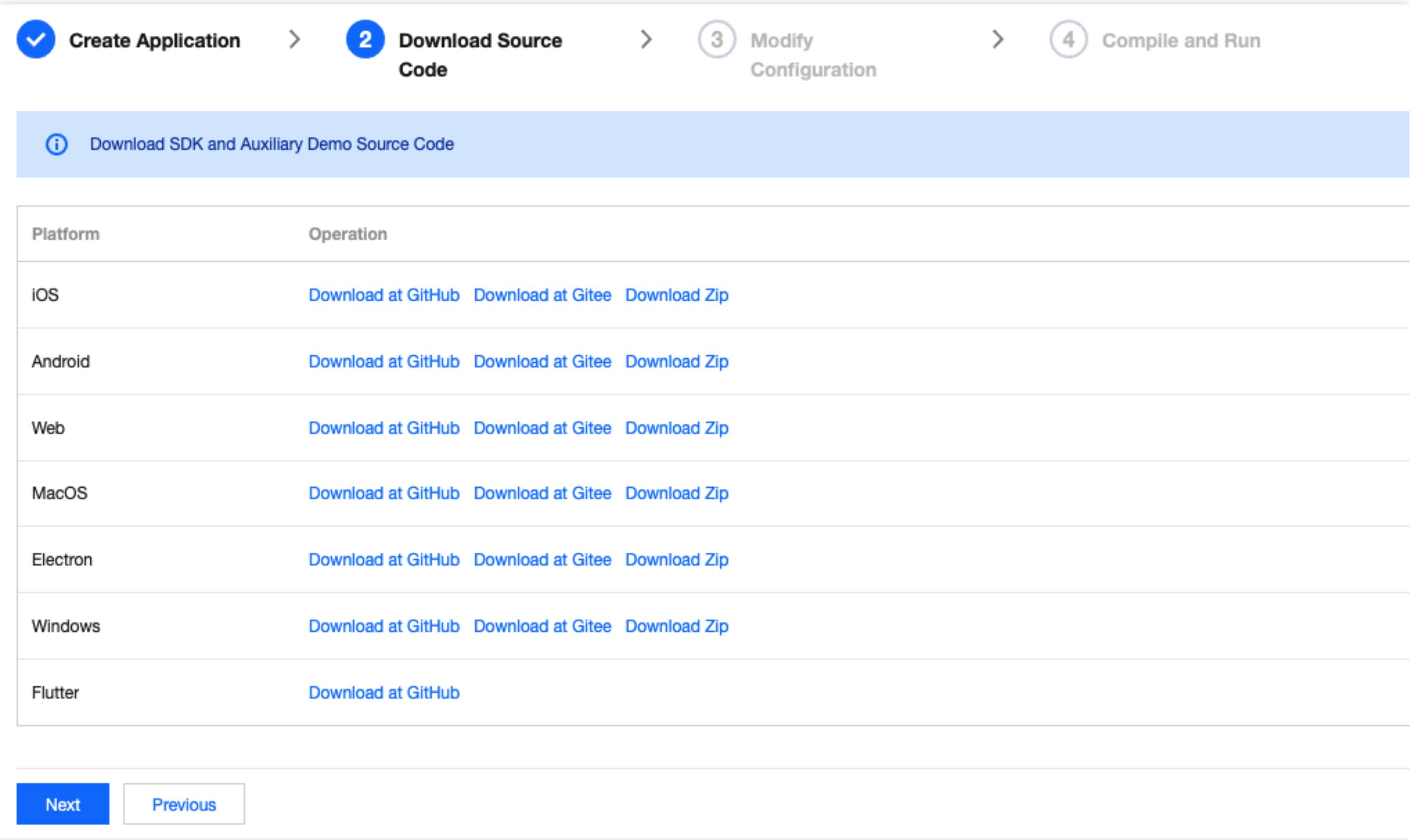
- `TUIRoom` integrates various capabilities such as ultra-low-latency audio/video call, screen sharing, and beauty filter, covering the common features of group audio/video room.
- It can be further developed as needed to quickly implement custom UI and layout, helping you quickly launch your business.
- It encapsulates the basic TRTC and IM SDKs to implement basic logic control and provides APIs for you to call features easily.

Connection Guide

To quickly implement the group audio/video room feature, you can modify the demo app we provide and adapt it to your needs. You can also use the `Module` module in the app and customize your own UI.

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestTUIRoom` and click **Create**.
3. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

> Note :
> The video conferencing feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Value-added Service Pricing for its billing details.

Step 2. Download the application source code

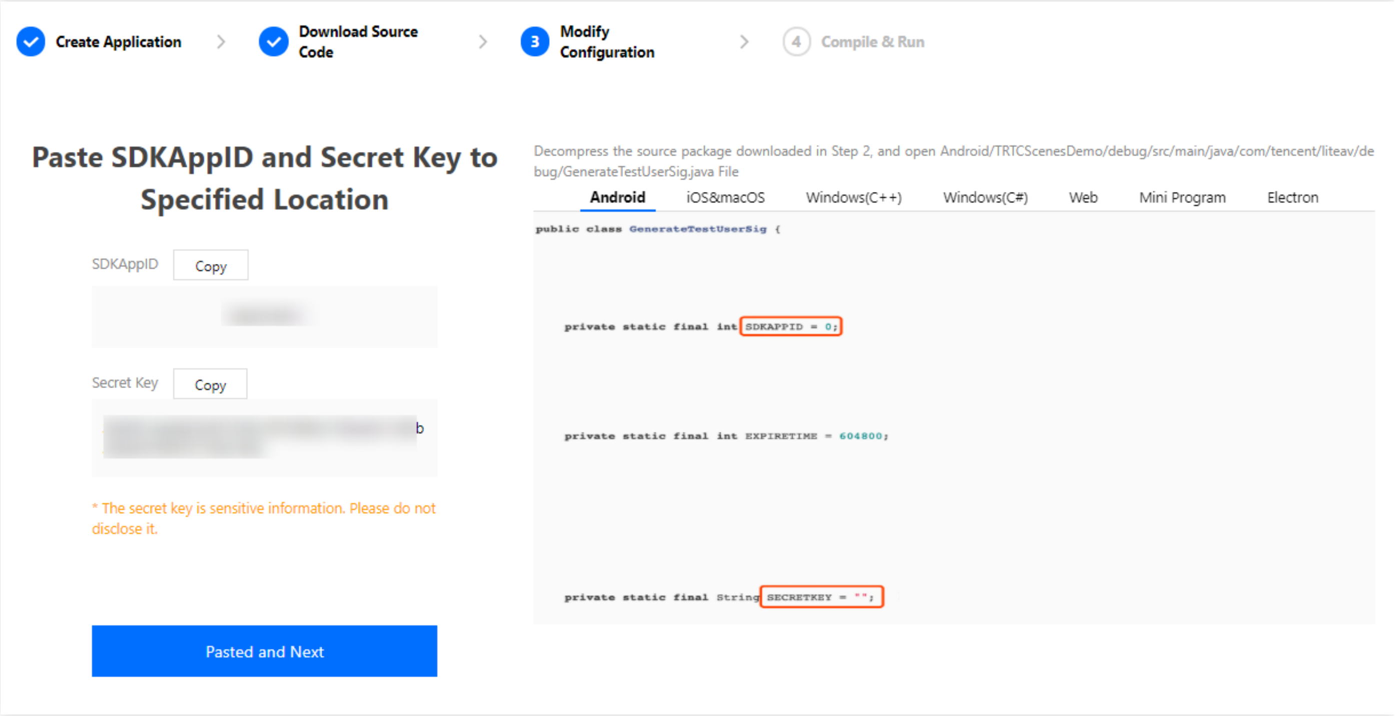
Click TUIRoom to clone or download the source code.

Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `Android/Debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java` .

3. Set parameters in `GenerateTestUserSig.java` :
   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Pasted and Next

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {



    private static final int SDKAPPID = 0;



    private static final int EXPIRETIME = 604800;



    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUIRoom` with Android Studio (version 3.5 or above) and click **Run**.

Step 5. Modify the project source code

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `ui` folder contains the UI code. The table below lists the files (folders) and UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
remote	The remote user list view
widget	Common UI component
CreateRoomActivity.java	The audio/video room creation view
RoomMainActivity.java	The main view for audio/video room
RoomVideoView.java	Include `TXCloudVideoView` , which displays the video data of the local user and remote users
MemberEntity.java	User data at the UI layer
MemberListAdapter.java	Adapter for the main audio/video room view

Tryout

> Note :
> You need at least two devices to try out the application.

Entry page

Select **Create Room** or **Enter Room**.

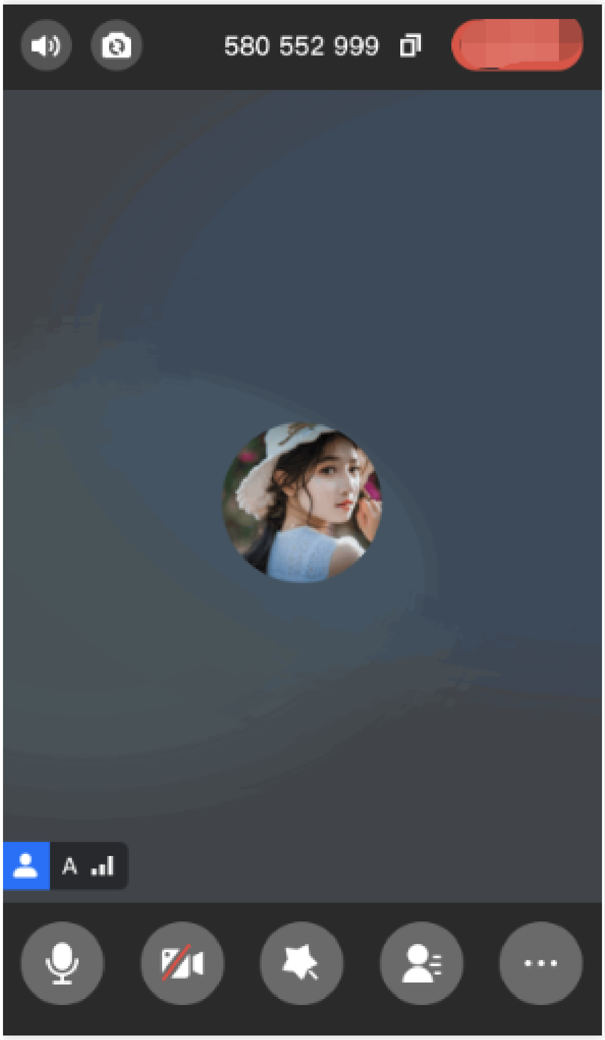
Room creation page

When user A creates a room, the room ID will be automatically generated. User A can tap **Create Room** to access the main UI.

Room entry page

User B enters the ID of the room created by user A and taps **Enter Room** to access the main UI.

Main UI (user A)



Mic-on list

It displays all users in the current room. If a user enables the camera and mic, you can watch the video image and hear the voice of the user.

Top toolbar

It implements features such as mic switch, front/rear camera switch, room information display, and room exit.

Bottom toolbar

It implements features such as local mic/camera control, beauty filter control, user list, and settings.

Beauty filter

Beauty filters and special effects can be displayed on the screen during live streaming.

Settings window

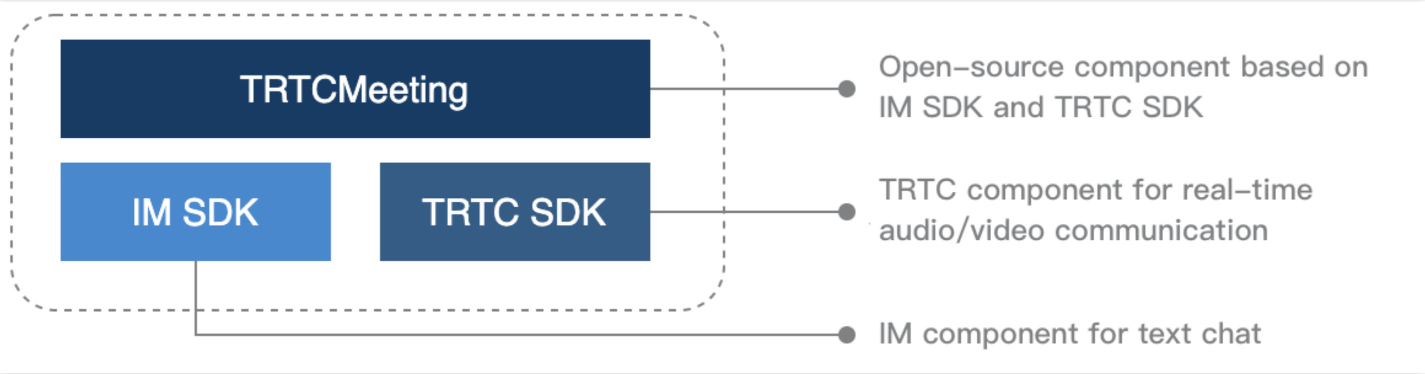
Audio and video parameters can be set, and **screen sharing** can be enabled.

Exiting a room

- **Anchor**: dismisses the room, so all users need to exit the room.
- **Non-anchor**: leaves the room.

Customizing UI

The `Source` folder in the source code contains two subfolders: `ui` and `model` . The `model` subfolder contains the reusable open-source component `TUIRoomCore` . You can find the component's APIs in `TUIRoomCore.java` and use them to customize your own UI.


TRTCMeeting
IM SDK
TRTC SDK
Open-source component based on IM SDK and TRTC SDK
TRTC component for real-time audio/video communication
IM component for text chat

Step 1. Integrate the SDK

The group audio/video room component `TUIRoomCore` depends on the TRTC SDK and IM SDK. Follow the steps below to integrate the two SDKs into your project:

- **Method 1: adding dependencies via Maven**
  i. Add the TRTC SDK and IM SDK dependencies to `dependencies` .

```
dependencies {
compile "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"
compile 'com.tencent.imsdk:imsdk:latest.release'
}
```

```
2. In `defaultConfig`, specify the CPU architecture to be used by your application.


defaultConfig {
ndk {
abiFilters "armeabi-v7a"
}
}

3. Click **Sync Now** to automatically download the SDKs and integrate them into your project.
```

- **Method 2: adding dependencies through local AAR files**
  If your access to the Maven repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration document
IM SDK	Download	Integration document

Step 2. Configure permission requests and obfuscation rules

1. Configure permission requests for your application in `AndroidManifest.xml` . The SDKs need the following permissions (on Android 6.0 and above, the camera and read storage permissions must be requested at runtime.)

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-feature android:name="android.hardware.camera"/>
<uses-feature android:name="android.hardware.camera.autofocus" />
```

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

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

Step 3. Import the `TUIRoomCore` component

Copy all the files in the directory below to your project:

```
src/main/java/com/tencent/liteav/tuiroom/model
```

Step 4. Create an instance and log in

Call `TUILogin` in `TUICore` to log in as shown below:

```
TUILogin.init(this, ”sdkAppid”, null, new V2TIMSDKListener() {
@Override
public void onKickedOffline() {
}
@Override
public void onUserSigExpired() {
}
});
TUILogin.login(”userId”, ”userSig”, new V2TIMCallback() {
@Override
public void onError(int code, String msg) {
}
@Override
public void onSuccess() {
}
});
```

Step 5. Set a nickname

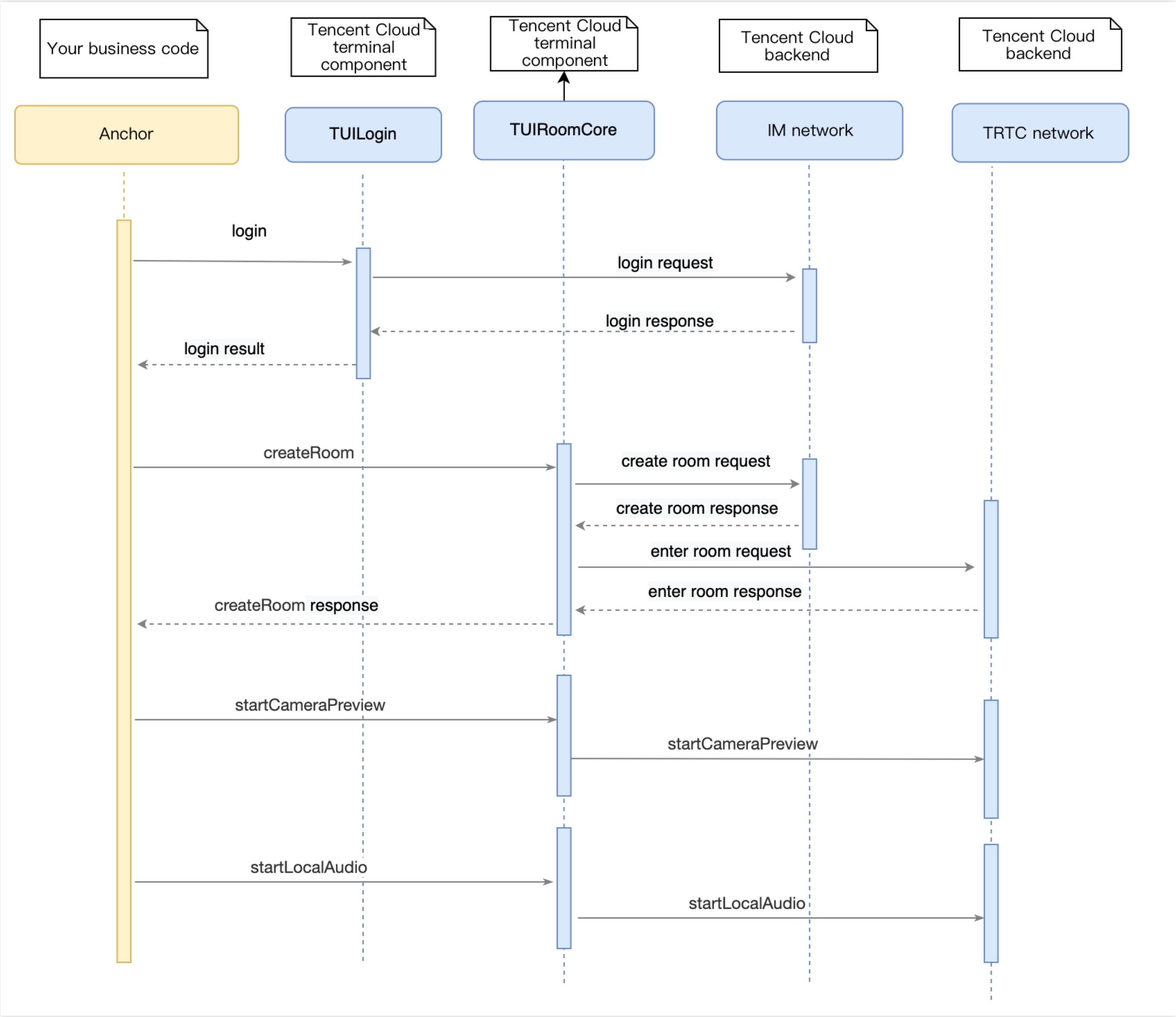
After successful login, call `setSelfProfile` to set the user profile.

Step 6. Create a room

1. A member calls the `createRoom` API to create a room. After successful room creation, the member enters the rooms (including chat room and TRTC room) as the anchor.

```
TUIRoomCore tuiRoomCore = TUIRoomCore.getInstance(context);
tuiCore.setListener(listener);
tuiCore.createRoom(”roomId”, TUIRoomCoreDef.SpeechMode.FREE_SPEECH,
new TUIRoomCoreCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// Room created successfully
} else {
}
}
});
```

2. Call the `startCameraPreview` API to start capturing and displaying the local video.



Step 7. Dismiss the room

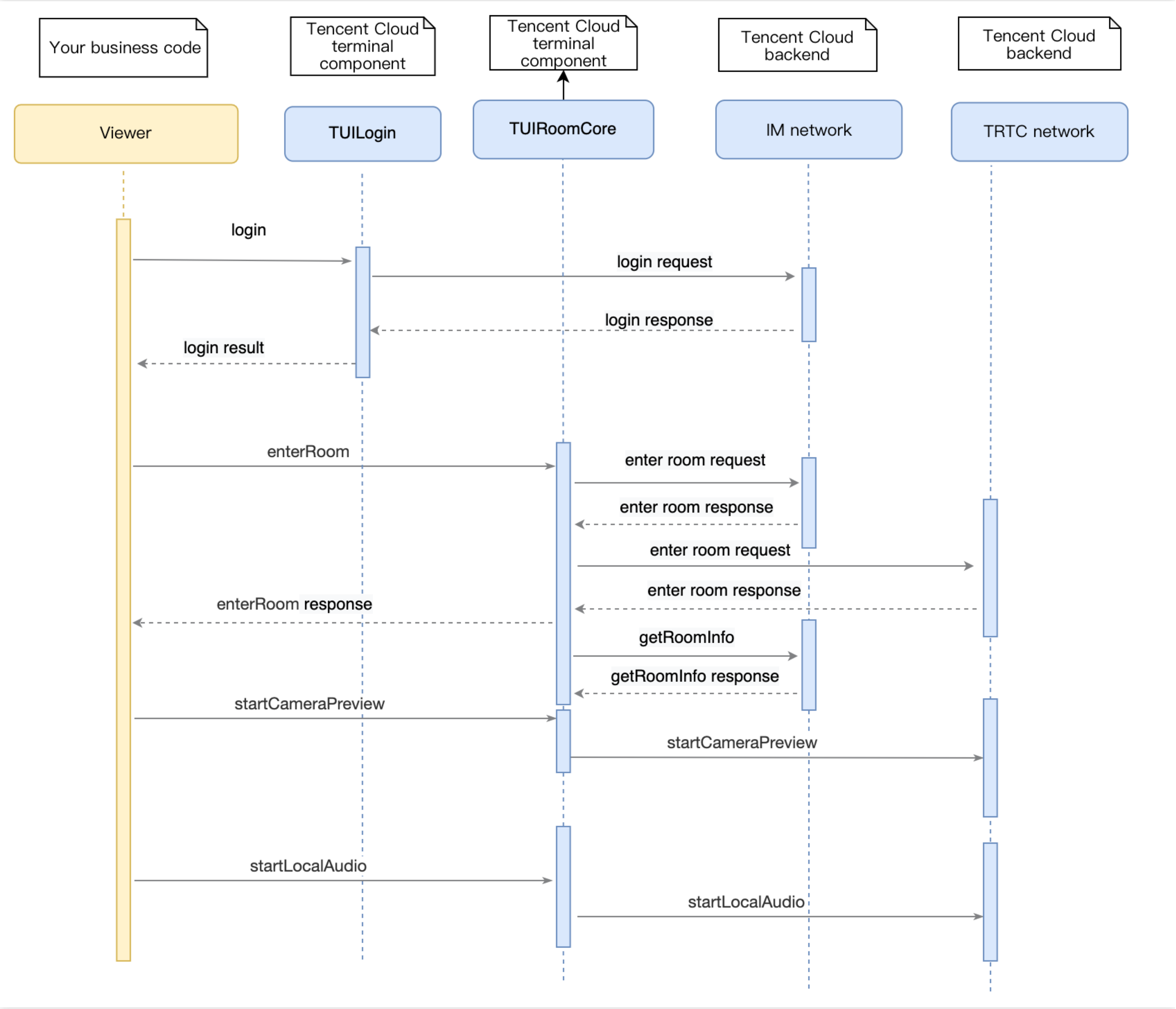
- The anchor calls the `destoryRoom` API to dismiss the room and IM group chat and exit the TRTC room.
- Room members receive the `onDestroyRoom` callback message that notifies them of group dismissal and exit the TRTC room.

Step 8. Enter a room

1. The room entry process is basically the same as the room creation process. The member needs to call the `enterRoom` API to enter the room.

2. Other members receive the `onRemoteUserEnter` callback that notifies them of room entry by the member.

```
TUIRoomCore tuiRoomCore = TUIRoomCore.getInstance(context);
tuiCore.setListener(listener);
tuiCore.enterRoom(”roomId”, new TUIRoomCoreCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// Entered the room successfully
} else {
}
}
});
```



Step 9. Exit a room

1. The member calls the `leaveRoom` API to exit the IM and TRTC rooms.
2. Other members receive the `onRemoteUserLeave` callback that notifies them of the room exit by the member.

Step 10. Enable screen sharing.

1. The screen sharing feature requires the floating window permission, so you need to include the permission requesting logic in your UI.
2. Call `startScreenCapture` of `TUIRoomCore` and pass in encoding parameters and the floating window during screen recording to start screen sharing. For more information, please see TRTC

SDK.

3. Other members in the room will receive the `onRemoteUserScreenVideoAvailable` event notification.

> Note :
> Screen sharing and camera capturing are mutually exclusive. Before enabling screen sharing, you need to call `stopCameraPreview` to disable camera capturing.

```
// 1. Add the SDK’ s screen sharing activity and permission in `AndroidManifest.xml`.
<uses-permission android:name=”android.permission.SYSTEM_ALERT_WINDOW”>
<application>
<activity android:name=”com.tencent.rtmp.video.TXScreenCapture$TXScreenCaptureAssistantActivity”
android:theme=”@android:style/Theme.Translucent”>
</activity></application>
```

```
// 2. Request the floating window permission in your UI.
if (Build.VERSION.SDK_INT >= 23) {
if (!Settings.canDrawOverlays(getApplicationContext())) {
Intent intent = new Intent(Settings.ACTION_MANAGE_OVERLAY_PERMISSION, Uri.parse(”package:” + getPackageName()));
startActivityForResult(intent, 100);
} else {
startScreenCapture();
}
} else {
startScreenCapture();
}

// 3. System callback result
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
if (requestCode == 100) {
if (Build.VERSION.SDK_INT >= 23) {
if (Settings.canDrawOverlays(this)) {
// The user grant the permission.
startScreenCapture();
} else {
}
}
}
}

// 4. Enable screen sharing.
private void startScreenCapture() {
TRTCCloudDef.TRTCVideoEncParam encParams = new TRTCCloudDef.TRTCVideoEncParam();
encParams.videoResolution = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_1280_720;
encParams.videoResolutionMode = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT;
encParams.videoFps = 10;
encParams.enableAdjustRes = false;
encParams.videoBitrate = 1200;

TRTCCloudDef.TRTCScreenShareParams params = new TRTCCloudDef.TRTCScreenShareParams();
mTUIRoom.stopCameraPreview();
mTUIRoom.startScreenCapture(encParams, params);
}
```

Group Audio/Video Room (Flutter)

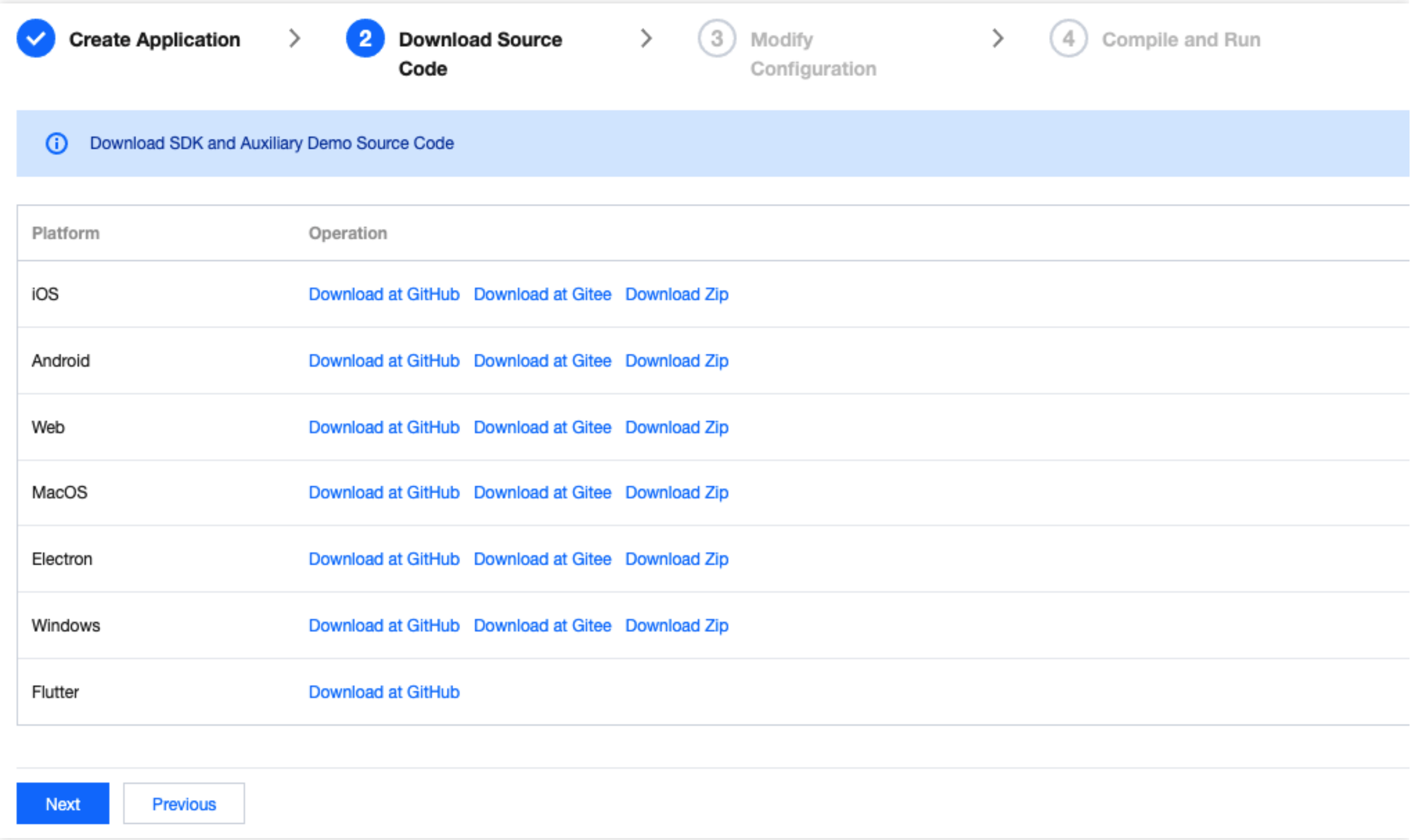
Last updated：2022-03-01 12:18:47

You can download and install the demo app we provide to try out TRTC’s group audio/video room features, including screen sharing, beauty filters, and low-latency conferencing.

Using the App’s UI

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestMeetingRoom` and click **Create**.
3. Click **Next**.



ⓘ Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Note：
This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. For IM billing

details, see Pricing.

Step 2. Download the application source code

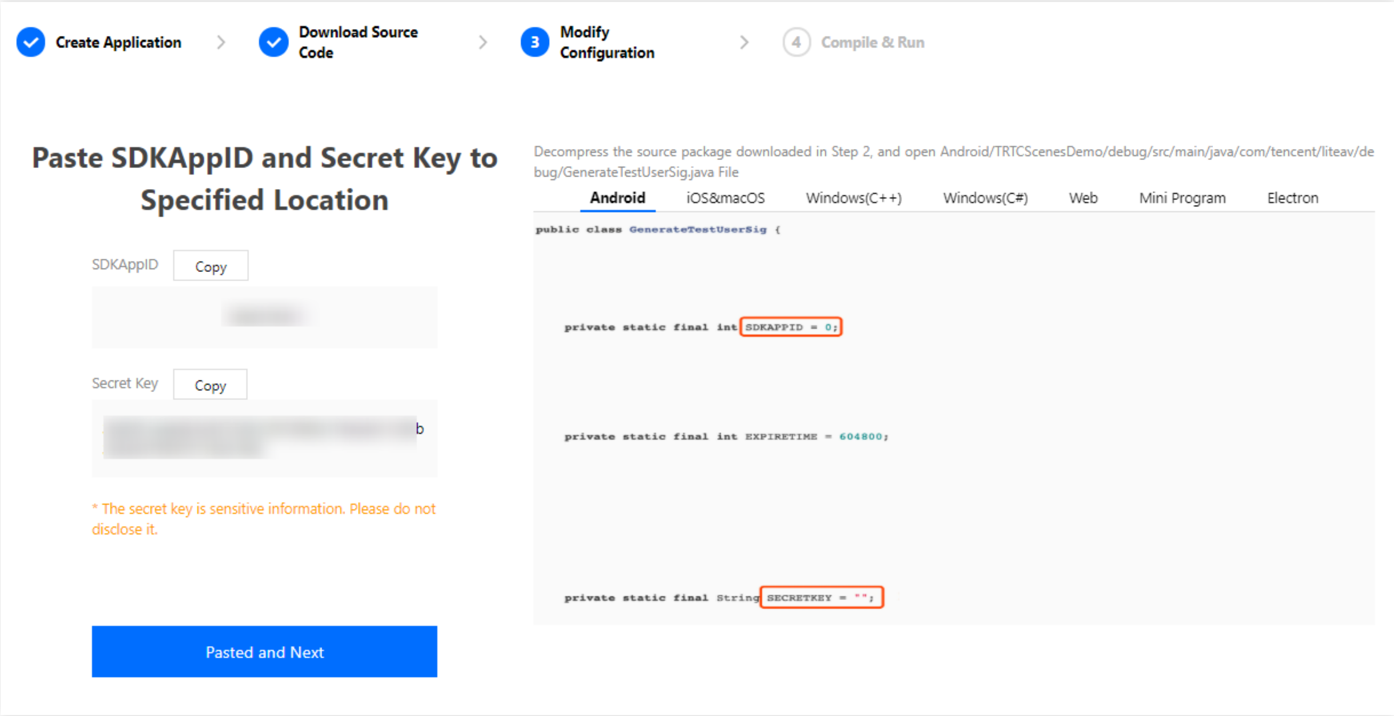
Clone or download the TRTCFlutterScenesDemo source code.

Step 3. Configure the application file

1. In the **Modify Configuration** step, select your platform.

2. Find and open `/lib/debug/GenerateTestUserSig.dart` .

3. Set parameters in `GenerateTestUserSig.dart` as follows.
   - SDKAPPID: It is `PLACEHOLDER` by default. You need to replace it with the real `SDKAppID`.
   - SECRETKEY: It is `PLACEHOLDER` by default. You need to replace it with the real key information.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted] b

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

Note :

- The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is leaked, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for the local execution and debugging of the demo**.
- The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4: Compile and Run the Demo

Note :
An Android project must be run on a real device rather than a simulator.

1. Run `flutter pub get` .
2. Compile, run, and test the project.
   - iOS
   - Android
   i. Open `¥ios project` in the source code directory with Xcode (11.0 or above).
   ii. Compile and run the demo project.

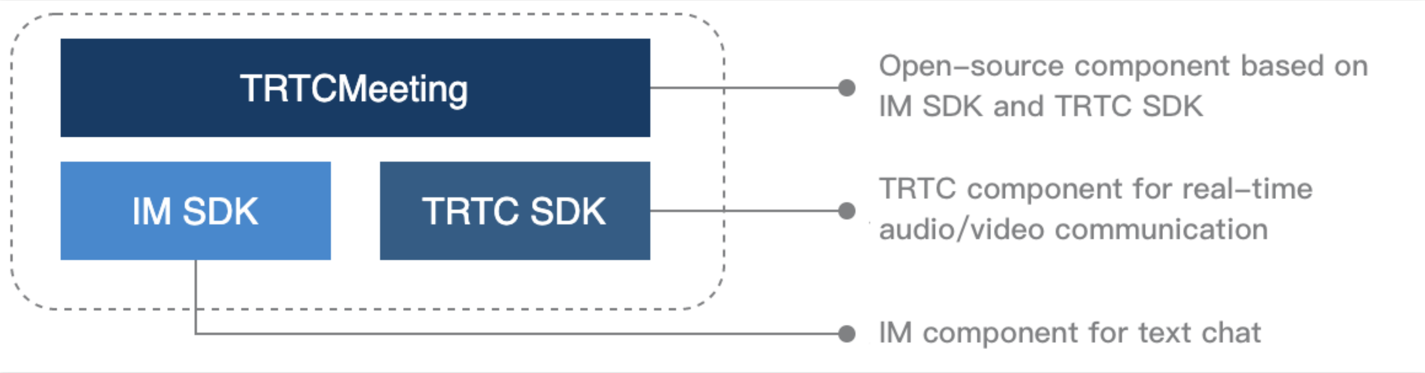
Step 5. Modify the demo source code

The `TRTCMeetingDemo` folder in the source code contains two subfolders: `ui` and `model` . The `ui` subfolder contains UI code. The table below lists the files (folders) and the UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
TRTCMeetingIndex.dart	The view for meeting creation or join
TRTCMeetingRoom.dart	The main view for video conferencing
TRTCMeetingMemberList.dart	The view for participant list
TRTCMeetingSetting.dart	The view for video conferencing parameter settings

Customizing UI

The `TRTCMeetingDemo` folder in the source code contains two subfolders: `ui` and `model`. The `model` subfolder contains the reusable open-source component `TRTCMeeting`. You can find the component's APIs in `TRTCMeeting.dart` and use them to customize your own UI.


TRTCMeeting
IM SDK
TRTC SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for text chat

Step 1. Integrate the SDK

The interactive live streaming component `RTCMeeting` depends on the TRTC SDK and IM SDK. You can configure `pubspec.yaml` to download their updates automatically.

Add the following dependencies to `pubspec.yaml` of your project.

```
dependencies:
tencent_trtc_cloud: latest version number
tencent_im_sdk_plugin: latest version number
```

Step 2. Configure permission requests and obfuscation rules

- iOS
- Android

Add request for mic permission in `Info.plist`:

```
<key>NSMicrophoneUsageDescriptionkey>
<string>Audio calls are possible only with mic access.string>
```

Step 3. Import the `TRTCMeeting` component.

Copy all the files in the directory below to your project:

```
lib/TRTCMeetingDemo/model/
```

Step 4. Create an instance and log in

1. Call the `sharedInstance` API to create an instance of the `TRTCMeeting` component.
2. Call the `registerListener` function to register event callbacks of the component.
3. Call the `login` API to log in to the component, and set the key parameters as described below.

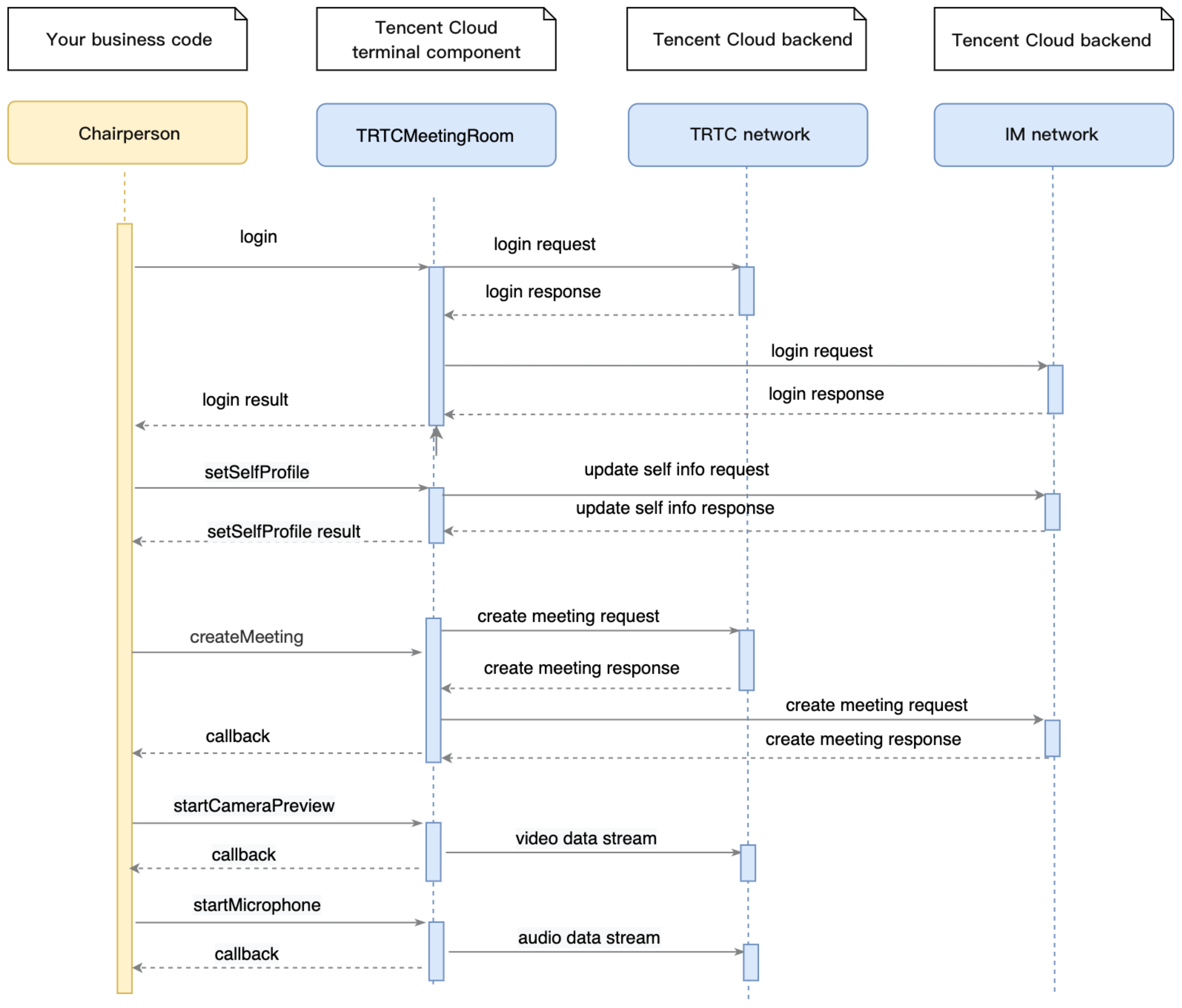
Parameter	Description
SDKAppID	You can view \`SDKAppID\` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.

```
TRTCMeeting trtcMeeting = TRTCMeeting.sharedInstance();
trtcMeeting.registerListener(onListener);
ActionCallback res = await trtcMeeting.login(
GenerateTestUserSig.sdkAppId,
userId,
GenerateTestUserSig.genTestSig(userId),
);
if (res.code == 0) {
// Login succeeded
}
```

Step 5. Create a meeting as an anchor

1. After performing step 4 to log in, call `setSelfProfile` to set your username and profile photo as an anchor.
2. Call `createMeeting` to create a meeting room.
3. Call `startCameraPreview` to capture video and `artMicrophone` to capture audio.
4. To use beauty filters, you can add beauty filter buttons to the UI and set beauty filters through `getBeautyManager` .

> Note :
> Only the Enterprise Edition SDK supports face changing and stickers.

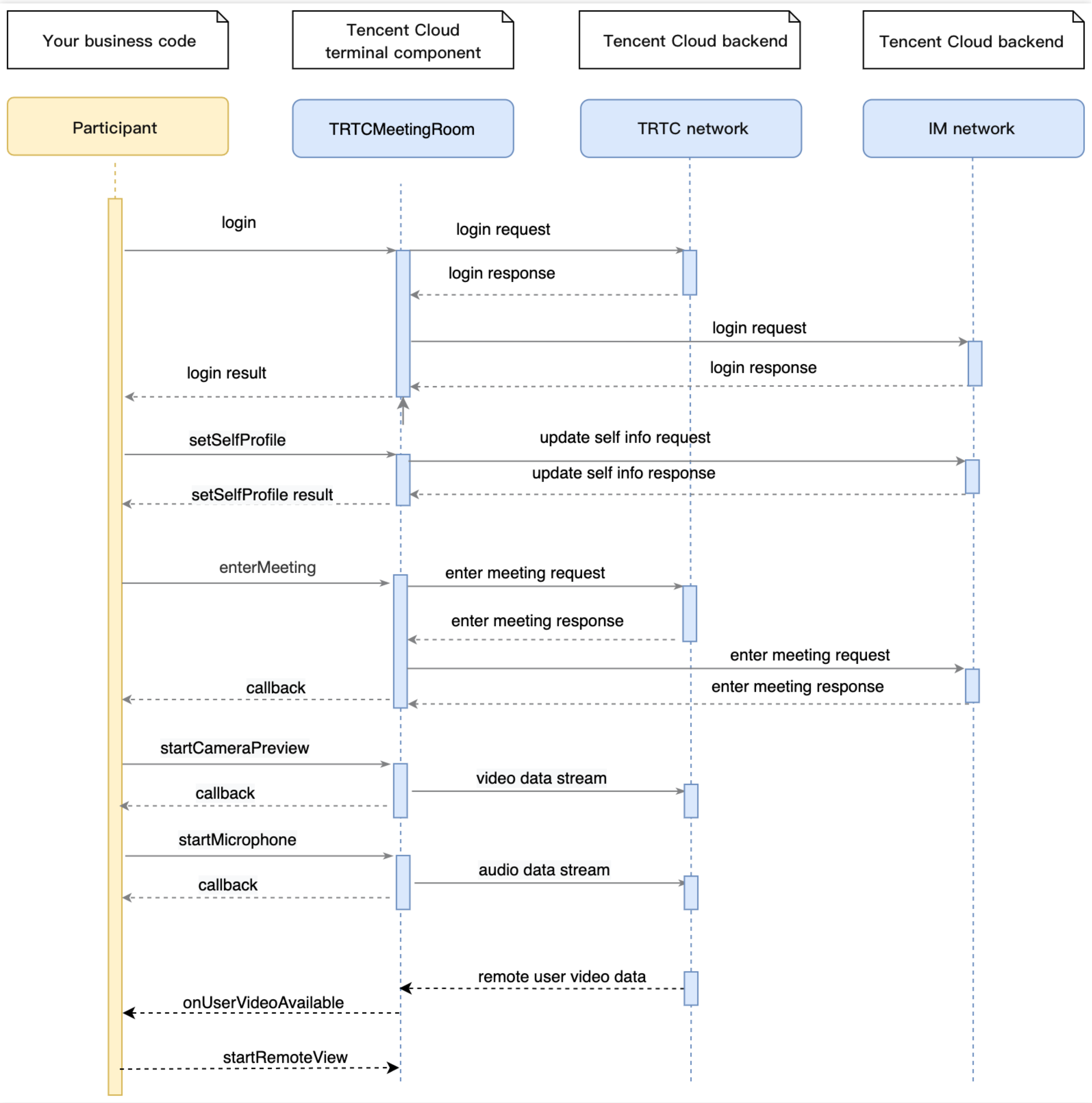


```
// 1. The anchor sets his or her nickname and profile photo.
trtcMeeting.setSelfProfile('my_name', 'my_avatar');

// 2. The anchor creates a meeting.
ActionCallback res = await trtcMeeting.createMeeting(roomId);
if (res.code == 0) {
// Created the meeting successfully
// 3. The anchor turns the camera on and enables audio capturing.
trtcMeeting.startCameraPreview(true, TRTCCloudVideoViewId);
trtcMeeting.startMicrophone();
// 4. Set the beauty filter.
trtcMeeting.getBeautyManager().setBeautyStyle(TRTCCloudDef.TRTC_BEAUTY_STYLE_PITU);
```

```
  trtcMeeting.getBeautyManager().setBeautyLevel(6);
  }
```

Step 6. Join a meeting as a participant

1. After performing step 4 to log in, call `setSelfProfile` to set your username and profile photo as a participant.
2. Call `enterMeeting` , passing in the meeting room ID to enter the room.
3. Call `startCameraPreview` to capture video and `startMicrophone` to capture audio.
4. If another participant turns the camera on, you will receive the `onUserVideoAvailable` notification, and can call `startRemoteView` and pass in the `userId` to play the participant’s video.



```
// 1. Set your username and profile photo as a participant
trtcMeeting.setSelfProfile('my_name', 'my_avatar');

// 2. The participant calls `enterMeeting` to enter the meeting room.
ActionCallback res = await trtcMeeting.enterMeeting(roomId);
if (res.code == 0) {
// Joined the meeting successfully.
// 3. The anchor turns the camera on and enables audio capturing.
trtcMeeting.startCameraPreview(true, TRTCCloudVideoViewId);
trtcMeeting.startMicrophone();
// 4. Set the beauty filter.
trtcMeeting.getBeautyManager().setBeautyStyle(TRTCCloudDef.TRTC_BEAUTY_STYLE_PITU);
trtcMeeting.getBeautyManager().setBeautyLevel(6);
}

// 5. The participant receives the notification that another member enables the camera and starts
playback
trtcMeeting.registerListener(onListener);
onListener(TRTCMeetingDelegate type, param) {
switch (type) {
case TRTCMeetingDelegate.onUserVideoAvailable:
if (param['available']) {
trtcMeeting.startCameraPreview(
param['userId'],
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG,
TRTCCloudVideoViewId
);
} else {
trtcMeeting.stopRemoteView(
param['userId'],
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG
);
}
break;
}
}
```

Step 7. Share the screen

1. The screen sharing feature requires the floating window permission, so you need to include the permission requesting logic in your UI.
2. Call `startScreenCapture` , passing in encoding parameters and the floating window during screen recording to start screen sharing. For more information, see TRTC SDK.
3. Other participants in the meeting will receive the `onUserVideoAvailable` event notification.

> Note :
> Screen sharing and camera capturing are mutually exclusive. Before enabling screen sharing, you need to call `stopCameraPreview` to disable camera capturing. For more information, see TRTC SDK.

```
await trtcMeeting.stopCameraPreview();
trtcMeeting.startScreenCapture(
videoFps: 10,
videoBitrate: 1600,
videoResolution: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_1280_720,
videoResolutionMode: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT,
appGroup: iosAppGroup,
);
```

Step 8. Implement text chat and muting notifications

- Call `sendRoomTextMsg` to send text messages. All participants in the meeting will receive the `onRecvRoomTextMsg` callback. IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: send text messages
trtcMeeting.sendRoomTextMsg('Hello Word!');
// Recipient: listen for text messages
trtcMeeting.registerListener(onListener);
onListener(TRTCMeetingDelegate type, param) {
switch (type) {
case TRTCMeetingDelegate.onRecvRoomTextMsg:
print('Received a message from' + param['userName'] + ':' + param['message']);
break;
}
}
```

- Call `sendRoomCustomMsg` to send custom (signaling) messages, and all participants in the meeting will receive the `onRecvRoomCustomMsg` callback. Custom messages are often used to transfer custom signals, such as muting notifications and notifications about other meeting controls.

```
// Sender: customize CMD to distinguish a muting notification
// E.g., use "CMD_MUTE_AUDIO" to indicate muting notifications
trtcMeeting.sendRoomCustomMsg('CMD_MUTE_AUDIO', '1');
// Recipient: listen for custom messages
trtcMeeting.registerListener(onListener);
onListener(TRTCMeetingDelegate type, param) {
switch (type) {
case TRTCMeetingDelegate.onRecvRoomCustomMsg:
if (param['command'] == 'CMD_MUTE_AUDIO') {
// Receive a muting notification.
print('Received a muting notification from' + param['userName'] + ':' + param['message']);
trtcMeeting.muteLocalAudio(message == ’1’);
}
break;
}
}
```

Group Audio/Video Room (Windows and macOS)

Last updated : 2022-03-18 23:00:53

This document describes the `TUIRoom` component for PC, an audio/video communication and collaboration tool with flexible layout and high adaptability. It can be used in various scenarios such as online collaboration, remote recruitment, remote diagnosis, insurance claim, online customer service, video interview, digital government services, finance digitization, online conferencing, and online education. It is integrated in depth with many industrial scenarios to help enterprises reduce costs, improve efficiency, and promote digitization for higher competitiveness.
You can download and install the app we provide to try out the component.

Solution Strengths

- `TUIRoom` integrates various capabilities such as ultra-low-latency audio/video call, chat room, screen sharing, beauty filter, device detection, and statistics, covering the common features of group audio/video room.
- It can be further developed as needed to quickly implement custom UI and layout, helping you quickly launch your business.
- It encapsulates the basic TRTC and IM SDKs to implement basic logic control and provides APIs for you to call features easily.

Connection Guide

To quickly implement the group audio/video room feature, you can modify the demo app we provide and adapt it to your needs. You can also use the `Module` module in the app and customize your own UI.

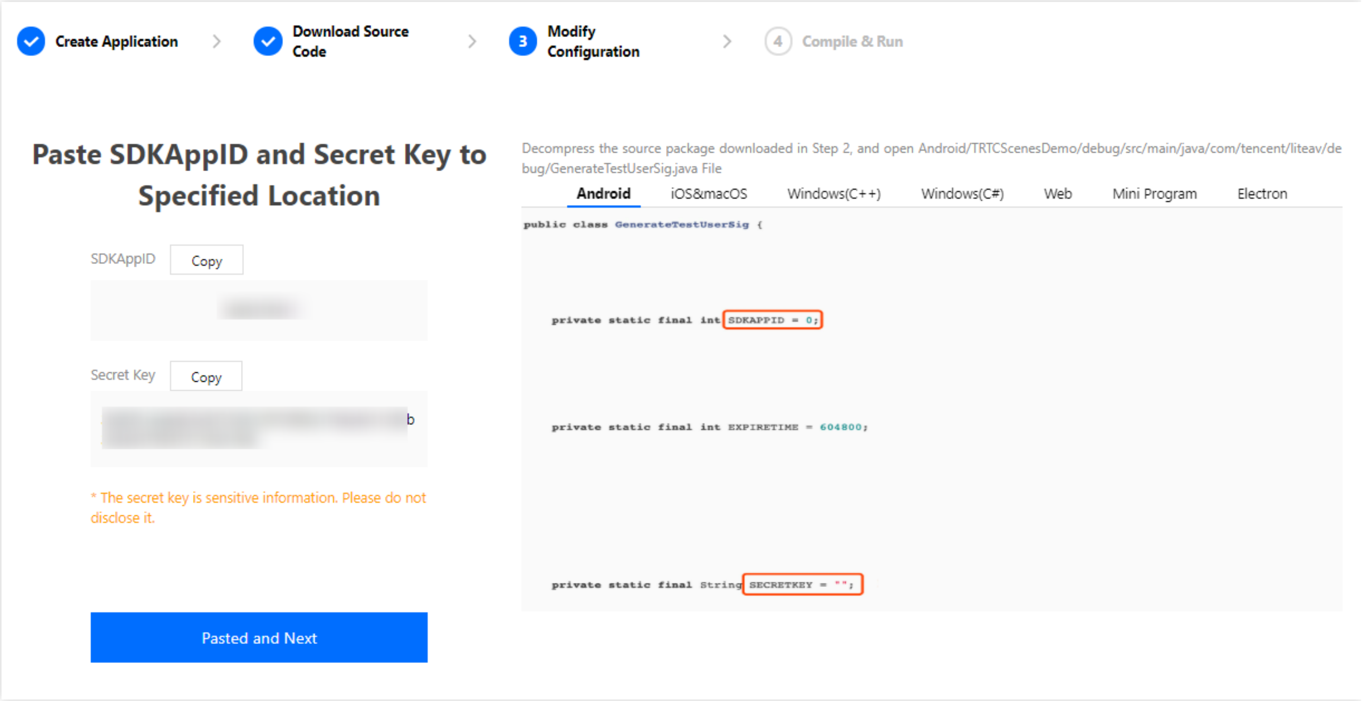
Step 1. Download the app source code

Click RoomApp to clone or download the source code.

Step 2. Configure the application file

- Configuration for Windows
- Configuration for macOS

1. Find and open `Windows¥RoomApp¥utils¥usersig¥win¥GenerateTestUserSig.h` .
2. Set parameters in `GenerateTestUserSig.h` :
    - **SDKAPPID**: `0` by default. Set it to the actual `SDKAPPID` .
    - **SECRETKEY**: empty by default. Set it to the actual `SECRETKEY` .



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted] b

* The secret key is sensitive information. Please do not disclose it.

Pasted and Next

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

Note :

- The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
- The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 3. Configure and run the app

- Configuration for Windows
- Configuration for macOS

1. Use Virtual Studio 2015 or above as the compilation environment.
2. The app UI is the currently popular Qt framework on 5.9. Download and install Qt and add it to Visual Studio, or modify the Qt configuration in the app project based on your Qt version.
3. Open the `RoomApp.vcxproj` source code project under `RoomApp` and compile and run the program.

Step 4. Modify the demo app’s source code

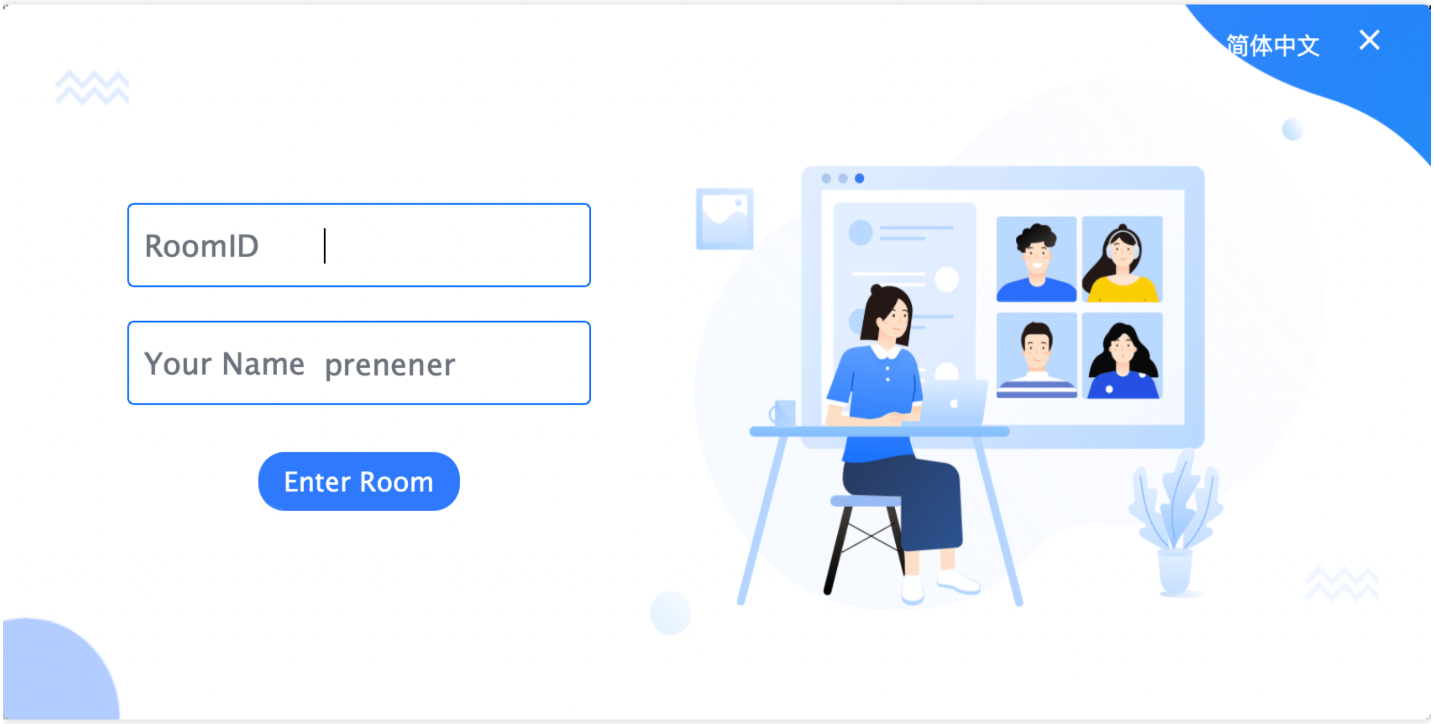
- The module layer encapsulates TRTC and IM to implement basic logic control and provides feature APIs for you to call features easily.
- The `App` module is a UI module, which contains the UI feature implementation. You can further develop UI as needed or replace the entire UI.

Tryout

> Note :
> You need at least two devices to try out the application.

Login page

Enter a room ID and a unique username to log in as shown below:

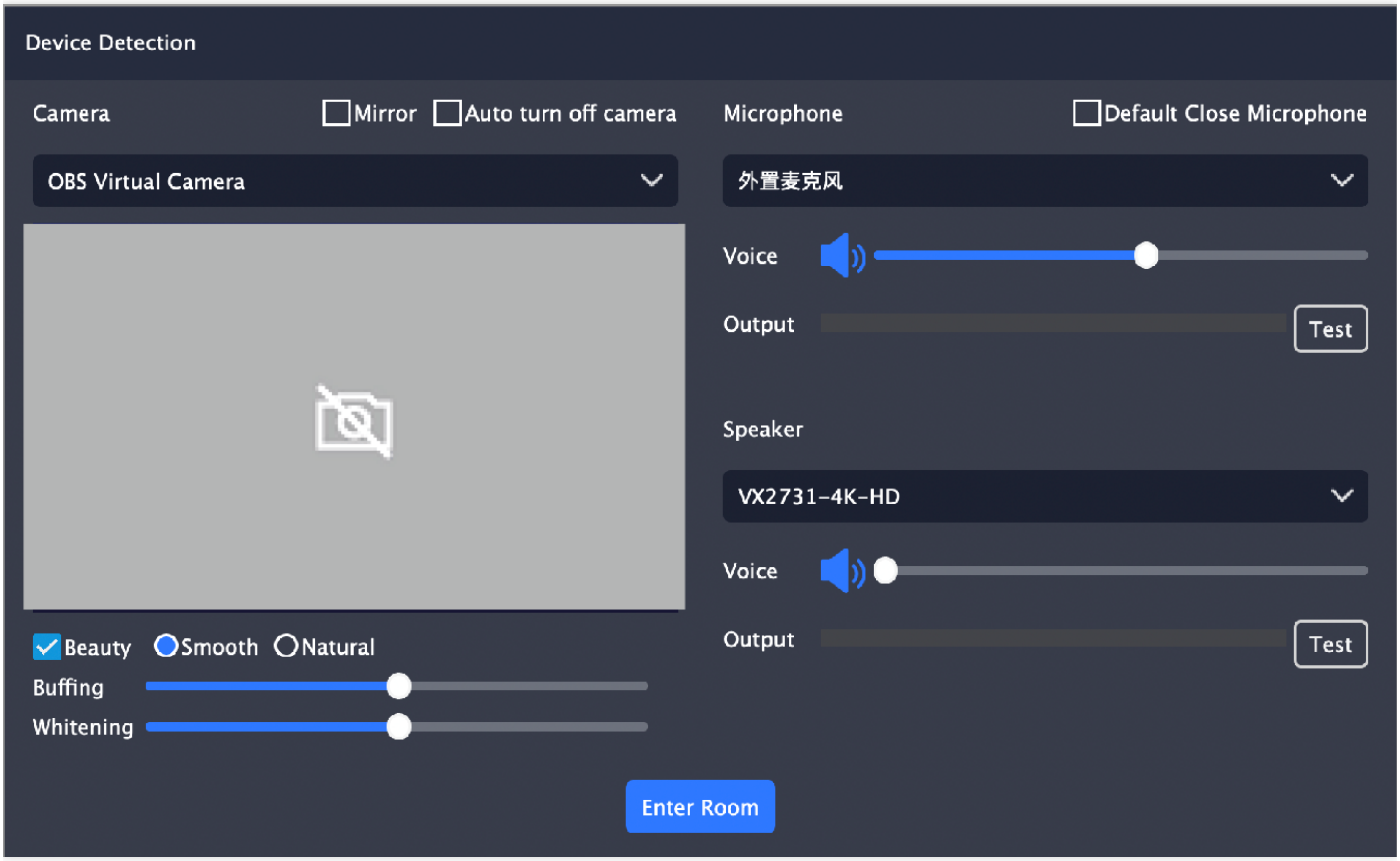


RoomID

Your Name prenener

Device detection page

Here, you can detect devices or set the device that will be used when a user enters the room and set the beauty filter as needed.



Device Detection

Camera □ Mirror □ Auto turn off camera

OBS Virtual Camera

☑ Beauty ● Smooth ○ Natural

Buffing

Whitening

Microphone □ Default Close Microphone

外置麦克风

Voice

Output

Speaker

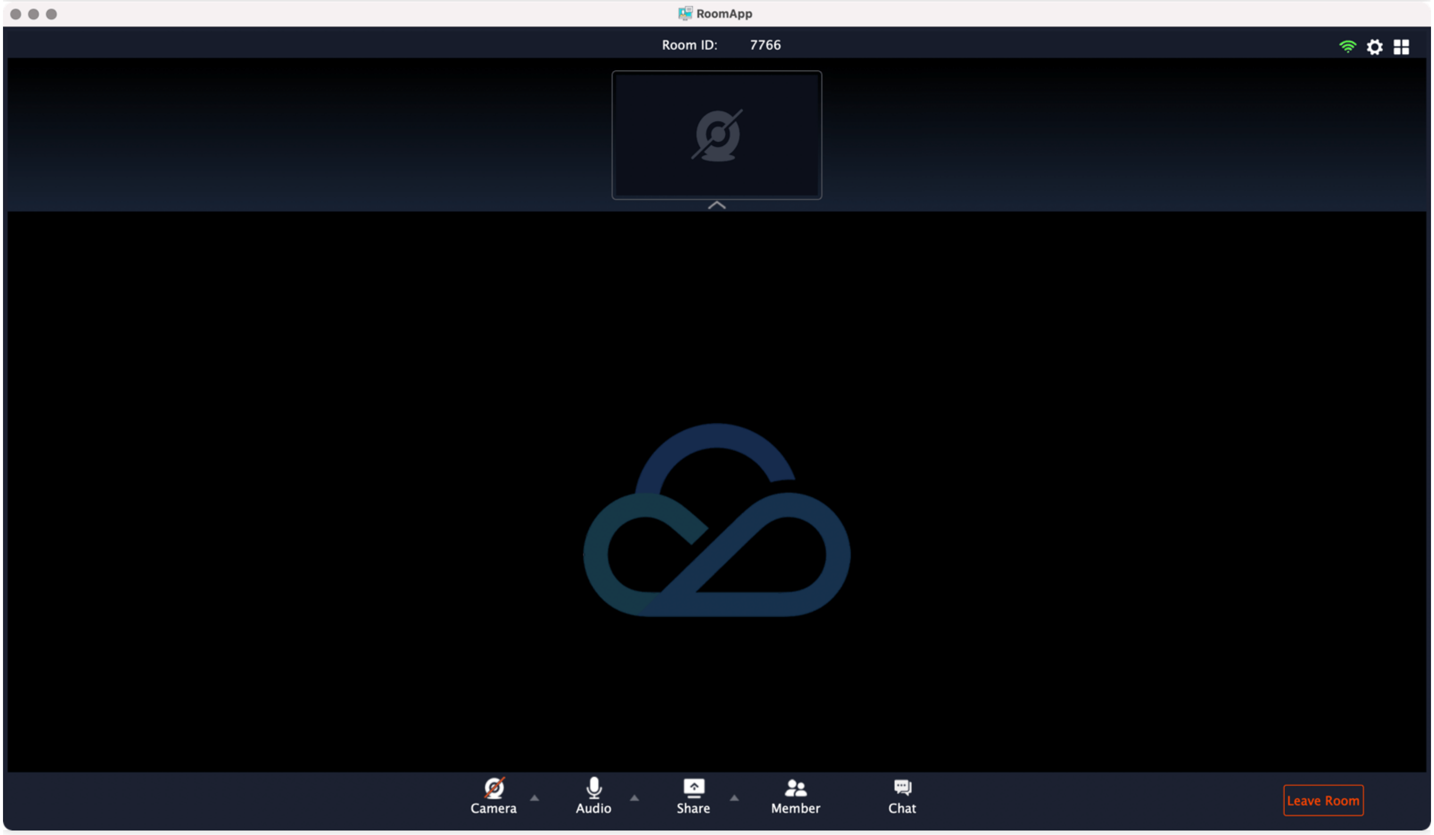
VX2731-4K-HD

Voice

Output

Homepage

The first user entering the room will be the anchor. The homepage contains components such as the mic-on list, bottom toolbar, and top toolbar.



Mic-on list

It displays all users in the current room. If a user enables the camera and mic, you can watch the video image and hear the voice of the user.

Bottom toolbar

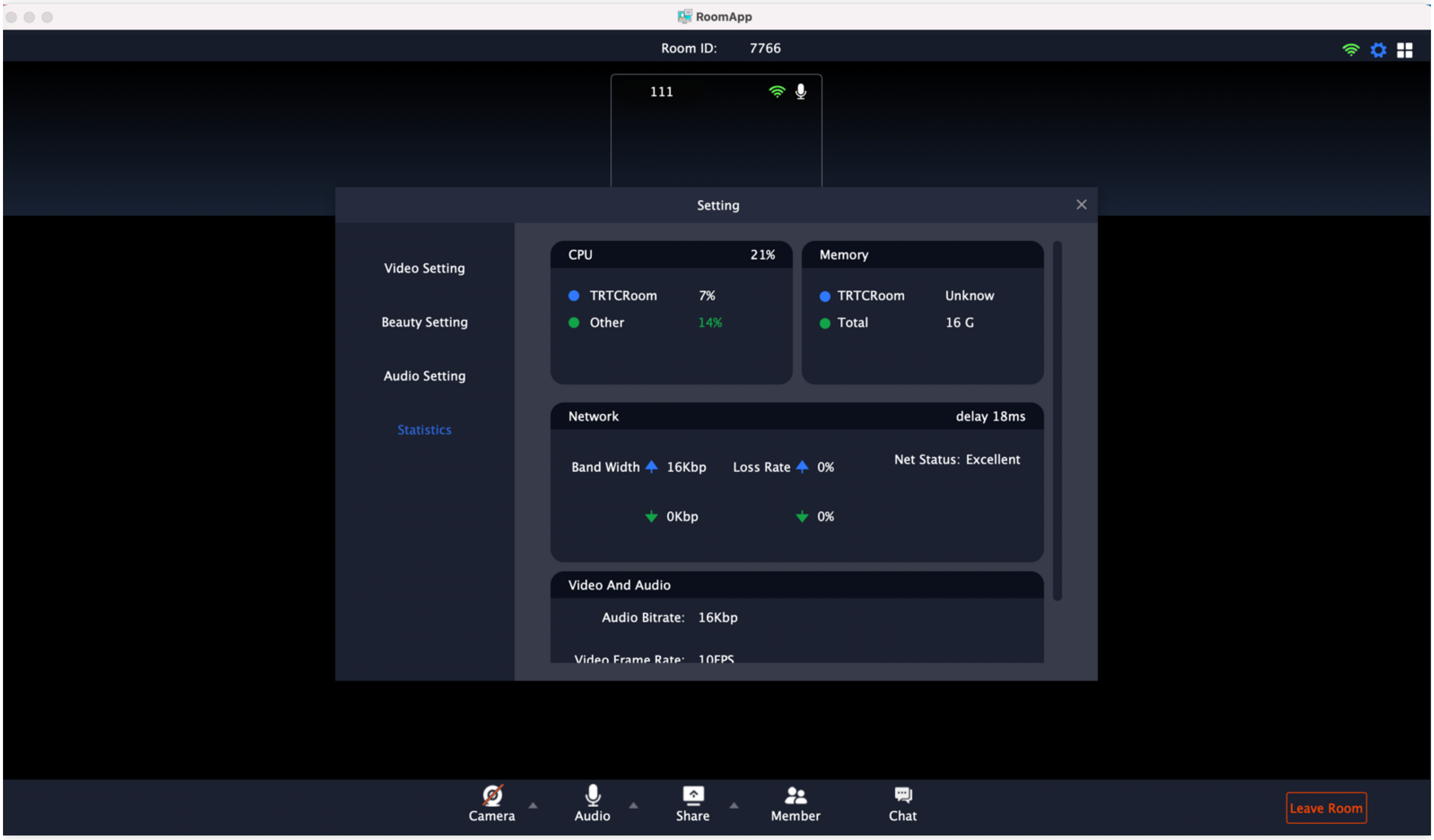
It implements features such as local mic/camera control, screen sharing control, user list, and chat room.

Top toolbar

It implements features such as network information display, page settings, and mic-on list layout.

Settings window

In the settings window, you can select devices, set beauty filter parameters, and set other parameters.



CPU 21%

- TRTCRoom 7%
- Other 14%

Memory

- TRTCRoom Unknow
- Total 16 G

Network delay 18ms

Band Width ↑ 16Kbp Loss Rate ↑ 0%

↓ 0Kbp ↓ 0%

Net Status: Excellent

Video And Audio

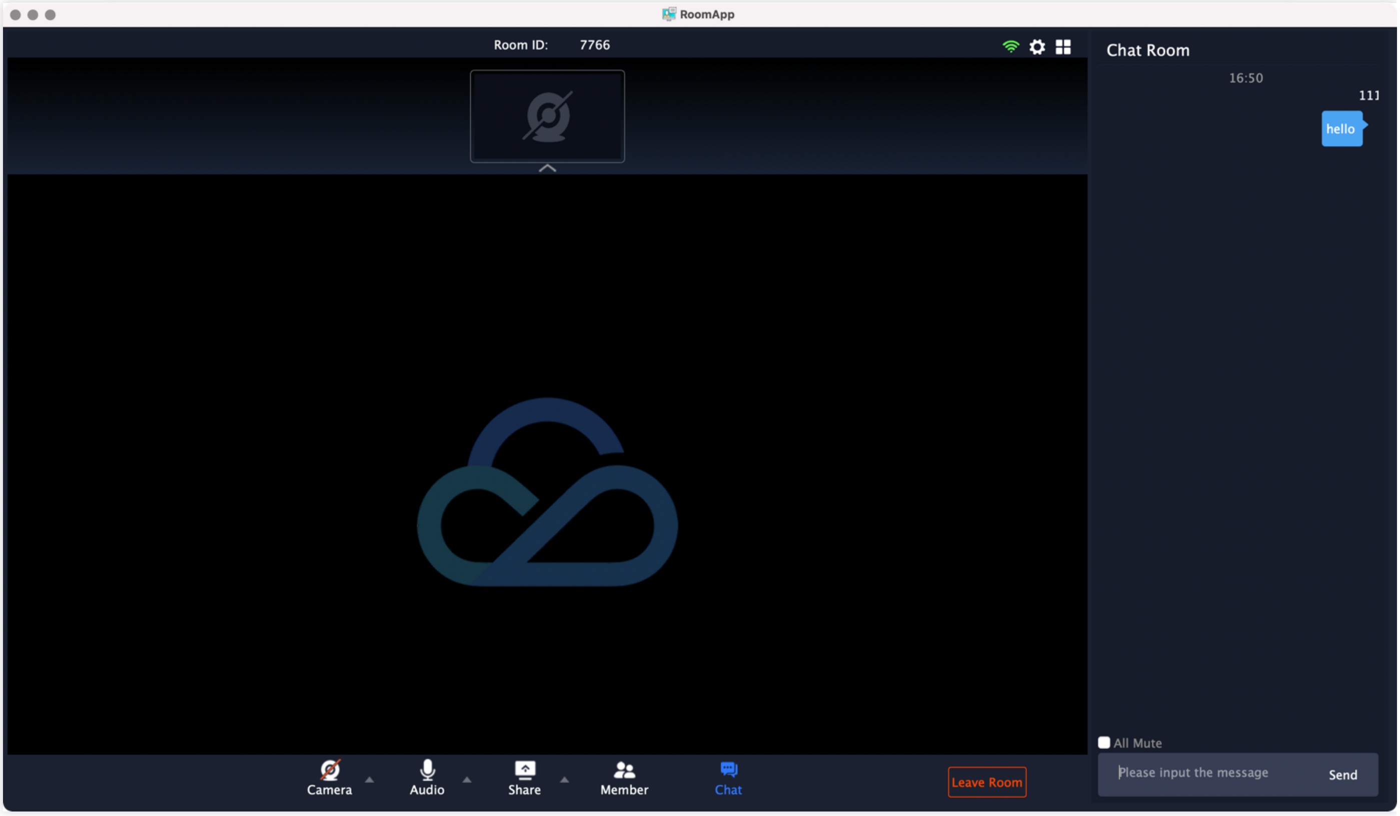
Audio Bitrate: 16Kbp

Video Frame Rate: 10FPS

Chat room

- Group members can chat in the chat room.

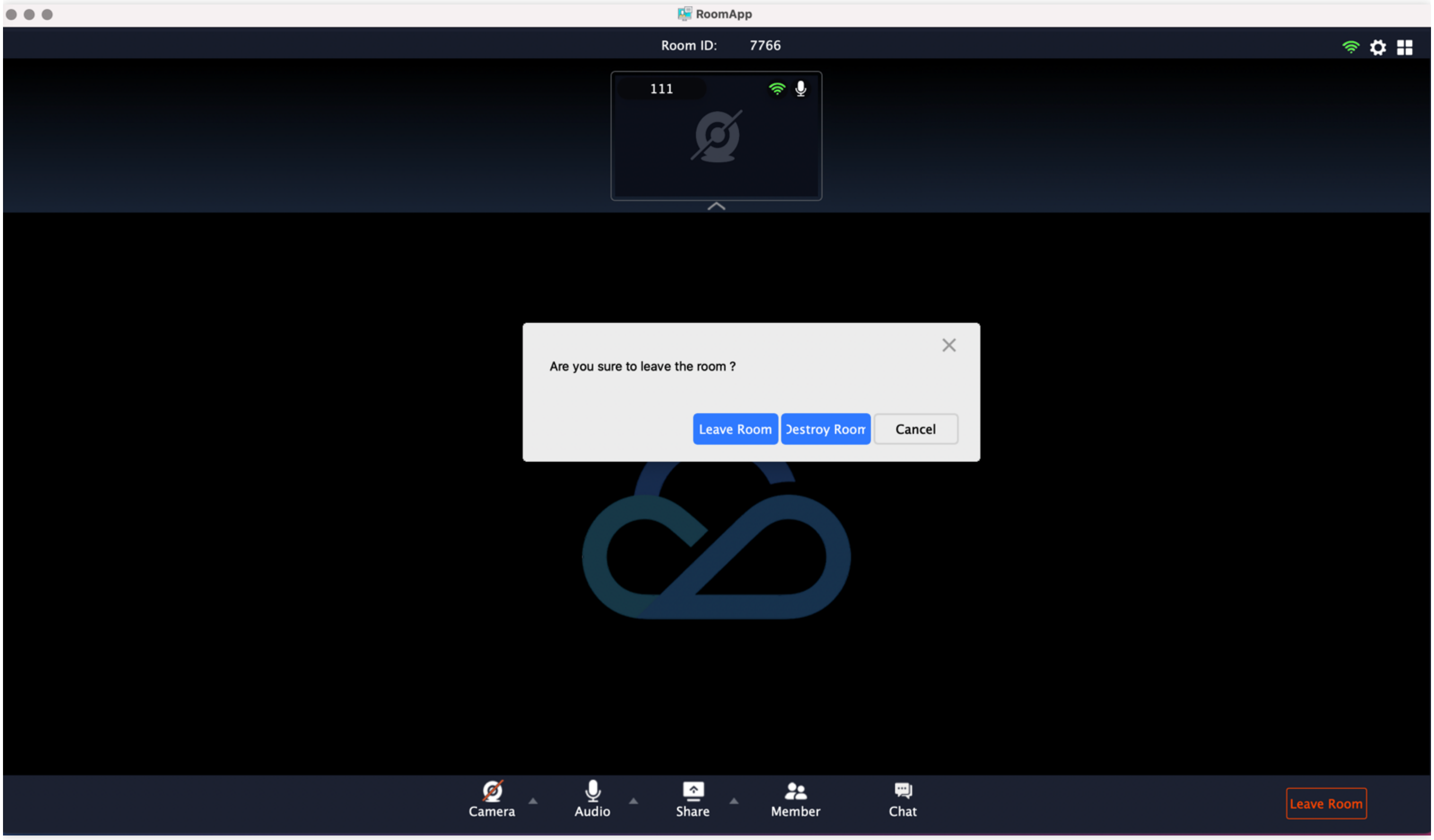
- The anchor can mute/unmute all members.



Leaving a room

There are two options when the anchor leaves a room:

- **Destroy Room**: All users need to leave the room.
- **Leave Room**: The room's anchor permission will be transferred to another user.



Are you sure to leave the room ?

Leave Room | Destroy Room | Cancel

Customizing UI

The `Module` module in the source code encapsulates the TRTC and IM SDKs. You can view the API functions and other definitions provided by this module in `TUIRoomCore.h` , `TUIRoomCoreCallback.h` , and `TUIRoomDef.h` files and use the corresponding APIs to implement your own custom UI.

Step 1. Integrate the SDK

Download the TRTC SDK and IM SDK at the official website. Then, replace the files in the `IM SDK` and `LiteAVSDK` directories in the SDK directory at the outer layer with them respectively.

> Note :
> The IM SDK for C++ is used.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Windows

SDK	Download Page	Integration Guide
IM SDK	Download	SDK Integration (Windows)

Step 2. Configure the SDK directories and library files of the project

- **For Windows**: in Visual Studio, configure the header file directory, static library directory, and static dependency library files.
- **For macOS**: add the dependency libraries and file directories to the `pro` and `pri` files.

Step 3. Create a singleton and log in

1. Call the `IUIRoomCore::Instance` API to create and use the `IUIRoomCore` singleton.
2. Call the `AddCallback` API to set the callback class in order to receive the callback notifications of the SDK.
3. Call the `Login` API for login.

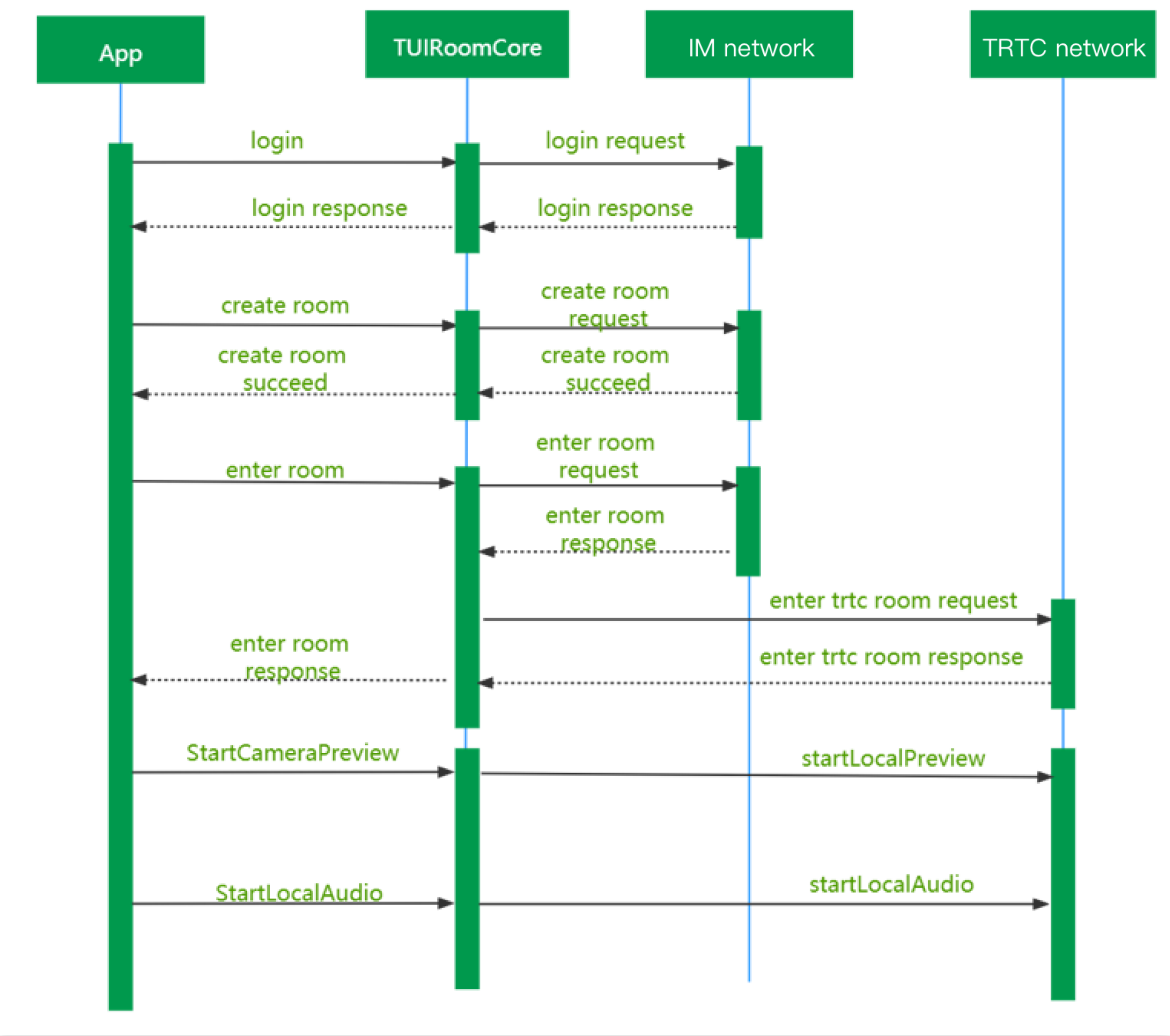
Parameter	Description
sdk_appid	You can view `SDKAppID` in the TRTC console.
user_id	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
user_sig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.

Step 4. Set a nickname

After successful login, call `SetSelfProfile` to set the user profile.

Step 5. Create a room

1. A member calls the `CreateRoom` API to create a room. After successful room creation, the member enters the rooms (including chat room and TRTC room) as the anchor.
2. The member calls the `StartCameraPreview` API to start capturing and displaying the local video.



Step 6. Dismiss the room

1. The anchor calls the `DestoryRoom` API to dismiss the room and IM group chat and exit the TRTC room.
2. Room members receive the `OnExitRoom` callback message that notifies them of group dismissal and exit the TRTC room.

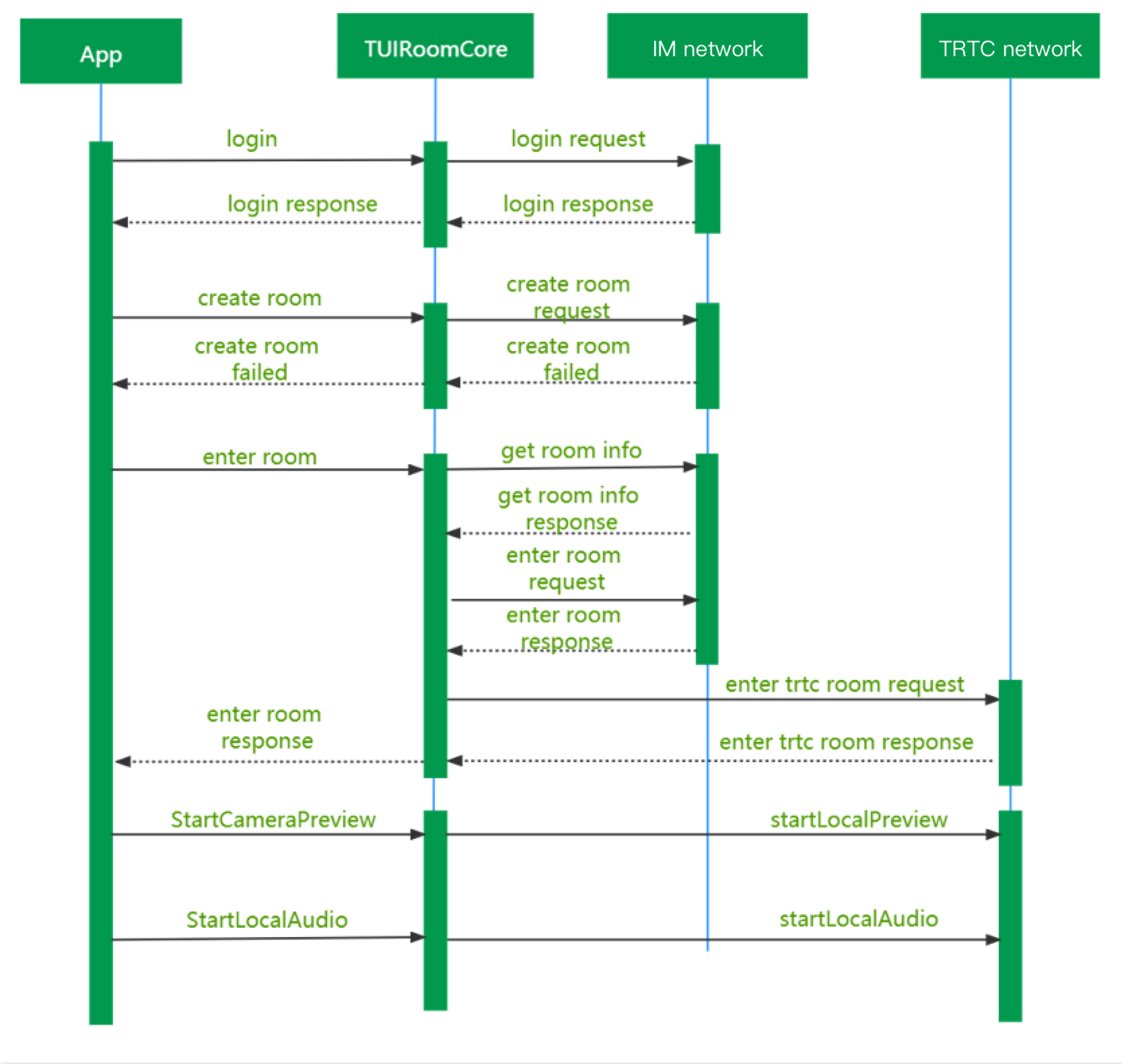
Step 7. Transfer the room

1. The anchor calls the `TransferRoomMaster` API to transfer the room to another user before leaving the room.

2. Room members receives the `OnRoomMasterChanged` callback message that notifies them of room anchor change.
3. The new anchor gets the anchor permission to control the group members.

Step 8. Enter a room

1. The room entry process is basically the same as the room creation process. When entering a room, the member needs to create a room first and will enter the room as a member if room creation fails.
2. After room creation fails, the member will get the room information first and then enter the room.
3. Other members receive the `OnRemoteUserEnter` callback that notifies them of room entry by the member.



Step 9. Exit a room

1. The member calls the `LeaveRoom` API to exit the IM and TRTC rooms.
2. Other members receive the `OnRemoteUserLeave` callback that notifies them of the room exit by the member.

Step 10. Enable screen sharing.

1. The member calls the `StartScreenCapture` API for screen sharing.

2. Other members receive the `OnRemoteUserScreenAvailable` callback that notifies them of screen sharing by the member.

> Note :
> You need to implement the window selection logic for the screen sharing module. For the specific implementation, see `ScreenShareWindow.h` in `App` .

Step 11. Implement text chat and muting notifications.

1. Call the `SendChatMessage` API to send a text message. Other members in the room can receive the `OnRecevieChatMessage` callback message and get the chat message for display.
2. The anchor can call the `MuteChatRoom` API to mute/unmute other members in the chat room. The members will receive the `OnChatRoomMuted` callback, and the UI layer will process the callback accordingly.

TUIRoom (iOS)

Last updated : 2022-03-01 12:18:47

`TUIRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). `TUIRoom` includes the following features:

- The anchor can create a room, and members can enter the room ID to join the room.
- Room members can share their screens with each other.
- All users can send various text and custom messages.

`TUIRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Group Audio/Video Room (for iOS).

- TRTC SDK: The TRTC SDK is used as a low-latency audio/video room component.
- IM SDK: The IM SDK for **iOS** is used to implement the chat room feature.

`TUIRoom` API Overview

Basic functions of `TUIRoomCore`

API	Description
shareInstance	Gets a singleton object.
destroyInstance	Terminates a singleton object.
setDelegate	Sets event callbacks.

Room APIs

API	Description
createRoom	Creates a room (called by anchor).
destroyRoom	Terminates a room (called by anchor).
enterRoom	Enters a room (called by room member).
leaveRoom	Exits a room (called by other room members).
getRoomInfo	Gets the room information.

API	Description
getRoomUsers	Gets the information of all users in the room.
getUserInfo	Gets the information of a user.
transferRoomMaster	Transfers the anchor permission (called by anchor).

Local audio/video operation APIs

API	Description
startCameraPreview	Enables the preview image of local video.
stopCameraPreview	Stops local video capturing and preview.
startLocalAudio	Enables mic capturing.
stopLocalAudio	Stops mic capturing.
setVideoMirror	Sets the mirroring preview mode of the local image.
setSpeaker	Sets whether to use the speaker or receiver.

Remote user APIs

API	Description
startRemoteView	Subscribes to and plays back the remote video image of a specified member.
stopRemoteView	Unsubscribes from and stops the playback of a remote video image.

Chat message sending APIs

API	Description
sendChatMessage	Sends a chat message.
sendCustomMessage	Sends a custom message.

Room control APIs

API	Description

API	Description
muteUserMicrophone	Enables/Disables the mic of a specified user.
muteAllUsersMicrophone	Enables/Disables the mic of all users and syncs the status to room information.
muteUserCamera	Enables/Disables the camera of a specified user.
muteAllUsersCamera	Enables/Disables the camera of all users and syncs the status to room information.
muteChatRoom	Mutes/Unmutes the chat room (called by anchor).
kickOffUser	Removes a specified user in the room (called by anchor).
startCallingRoll	Starts calling roll by the anchor.
stopCallingRoll	Stops calling roll by the anchor.
replyCallingRoll	Replies to roll call by a member.
sendSpeechInvitation	Invites a member to speak by the anchor.
cancelSpeechInvitation	Cancels invitation to a member for speech by the anchor.
replySpeechInvitation	Accepts/Rejects the speech invitation from the anchor by a member.
sendSpeechApplication	Applies for speech by a member.
replySpeechApplication	Approves/Rejects the speech application of a member by the anchor.
forbidSpeechApplication	Forbids speech application by the anchor.
sendOffSpeaker	Stops the speech of a member by the anchor.
sendOffAllSpeakers	Stops the speech of all members by the anchor.
exitSpeechState	Stops speaking by a member and changes their role to audience.

Screen sharing APIs

API	Description
startScreenCapture	Starts screen sharing.
stopScreenCapture	Stops screen sharing.

Beauty filter APIs

API	Description
getBeautyManager	Gets the beauty filter management object TXBeautyManager.

Settings APIs

API	Description
setVideoQosPreference	Sets network bandwidth limit parameters.

SDK version acquisition APIs

API	Description
getSDKVersion	Gets the SDK version.

`TUIRoomCoreDelegate` API Overview

Callbacks for error events

API	Description
onError	Error

Basic event callbacks

API	Description
onDestroyRoom	Room dismissal.
onUserVoiceVolume	Volume level.
onRoomMasterChanged	Anchor change.

Remote user event callbacks

API	Description
onRemoteUserEnter	A remote user entered the room.
onRemoteUserLeave	A remote user exited the room.

API	Description
onRemoteUserCameraAvailable	Whether a remote user enabled the camera.
onRemoteUserScreenVideoAvailable	Whether a remote user enabled screen sharing.
onRemoteUserAudioAvailable	Whether a remote user enabled sending audio.
onRemoteUserEnterSpeechState	A remote user started speaking.
onRemoteUserExitSpeechState	A remote user stopped speaking.

Message event callback APIs

API	Description
onReceiveChatMessage	A text message was received.

Room control event callbacks

API	Description
onReceiveSpeechInvitation	A member received a speech invitation from the anchor.
onReceiveInvitationCancelled	A member received a speech invitation cancellation from the anchor.
onReceiveSpeechApplication	The anchor received a speech application from a member.
onSpeechApplicationCancelled	A member canceled a speech application.
onSpeechApplicationForbidden	The anchor rejected a speech application.
onOrderedToExitSpeechState	A member was asked to stop speaking.
onCallingRollStarted	The anchor started a roll call.
onCallingRollStopped	The anchor stopped a roll call.
onMemberReplyCallingRoll	A member replied to roll call.
onChatRoomMuted	The anchor muted/unmuted the room.
onMicrophoneMuted	The anchor disabled the mic.
onCameraMuted	The anchor disabled the camera.

API	Description
onReceiveKickedOff	The anchor removed a member.

Callback APIs for statistics on network quality and technical metrics

API	Description
onStatistics	Statistics on technical metrics.
onNetworkQuality	Network quality.

Screen sharing event callbacks

API	Description
onScreenCaptureStarted	Screen sharing started.
onScreenCaptureStopped	Screen sharing stopped.

Basic Functions of `TUIRoomCore`

getInstance

This API is used to get a TUIRoomCore singleton object.

```
+ (instancetype)shareInstance;
```

destroyInstance

```
+ (void)destroyInstance;
```

setDelegate

This API is used to set the event callback of TUIRoomCore. You can use `TUIRoomCoreDelegate` to get different status notifications of TUIRoomCore.

```
- (void)setDelegate:(id)delegate;
```

The parameters are as detailed below:

Parameter	Type	Description
delegate	TUIRoomCoreDelegate	Event callback class.

createRoom

This API is used to create a room (called by the anchor).

```
- (void)createRoom:(NSString *)roomId
speechMode:(TUIRoomSpeechMode)speechMode
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	NSString	Room ID. You need to assign and manage the IDs in a centralized manner.
speechMode	TUIRoomSpeechMode	Speech mode.
callback	TUIRoomActionCallback	Room creation result.

Generally, the anchor calls the APIs in the following steps:

1. The **anchor** calls `createRoom()` to create a room, the result of which is returned via `TUIRoomActionCallback` .
2. The **anchor** calls `startCameraPreview()` to enable camera capturing and preview.
3. The **anchor** calls `startLocalAudio()` to enable the local mic.

destroyRoom

This API is used to terminate a room (called by the anchor).

```
- (void)destroyRoom:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomActionCallback	Room termination result.

enterRoom

This API is used to enter a room (called by a member).

```
- (void)enterRoom:(NSString *)roomId
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	NSString	Room ID.
callback	TUIRoomActionCallback	Result.

Generally, a member enters a room in the following steps:

1. The **member** calls `enterRoom` and passes in `roomId` to enter the room.
2. The **member** calls `startCameraPreview()` to enable camera preview and calls startLocalAudio()` to enable mic capturing.
3. The **member** receives the `onRemoteUserCameraAvailable` event and calls `startRemoteView()` to start playback.

leaveRoom

This API is used to exit a room (called by a member).

```
- (void)leaveRoom:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomActionCallback	Result.

getRoomInfo

This API is used to get the room information.

```
- (nullable TUIRoomInfo *)getRoomInfo;
```

getRoomUsers

This API is used to get the information of all users in the room.

```
- (nullable NSArray *)getRoomUsers;
```

getUserInfo

This API is used to get the information of a user in the room.

```
- (void)getUserInfo:(NSString *)userId
callback:(TUIRoomUserInfoCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.
callback	TUIRoomUserInfoCallback	Room member details.

setSelfProfile

Set User Info

```
- (void)setSelfProfile:(NSString *)userName
avatarURL:(NSString *)avatarURL
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
userName	NSString	User name.
avatarURL	NSString	User profile photo URL.
callback	TUIRoomActionCallback	Whether the setting succeeded.

transferRoomMaster

This API is used to transfer a group to another user.

```
- (void)transferRoomMaster:(NSString *)userId
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	NSString	User ID.
callback	TUIRoomActionCallback	Result.

Local Push APIs

startCameraPreview

This API is used to start the preview of the local camera.

```
- (void)startCameraPreview:(BOOL)isFront
view:(UIView *)view;
```

The parameters are as detailed below:

Parameter	Type	Description
isFront	BOOL	YES: front camera; NO: rear camera.
view	UIView	Control that carries the video image.

stopCameraPreview

This API is used to stop the preview of the local camera.

```
- (void)stopCameraPreview;
```

startLocalAudio

This API is used to start mic capturing.

```
- (void)startLocalAudio:(TRTCAudioQuality)quality;
```

The parameters are as detailed below:

Parameter	Type	Description
quality	TRTCAudioQuality	Captured sound quality.

stopLocalAudio

This API is used to stop mic capturing.

```
- (void)stopLocalAudio;
```

setVideoMirror

This API is used to set the mirroring preview mode of local video image.

```
- (void)setVideoMirror:(TRTCVideoMirrorType)type;
```

The parameters are as detailed below:

Parameter	Type	Description
type	TRTCVideoMirrorType	Mirroring type.

setSpeaker

This API is used to set whether to use the speaker or receiver.

```
- (void)setSpeaker:(BOOL)isUseSpeaker;
```

The parameters are as detailed below:

Parameter	Type	Description
isUseSpeaker	BOOL	YES: speaker; NO: receiver.

Remote User APIs

startRemoteView

This API is used to subscribe to a remote user's video stream.

```
- (void)startRemoteView:(NSString *)userId
view:(UIView *)view
streamType:(TUIRoomStreamType)streamType
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	NSString	ID of the user whose video image is to be played back.
view	UIView	The control that loads video images.
streamType	TUIRoomStreamType	Stream type.
callback	TUIRoomActionCallback	Result.

stopRemoteView

This API is used to unsubscribe from and stop the playback of a remote video image.

```
- (void)stopRemoteView:(NSString *)userId
streamType:(TUIRoomStreamType)streamType
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	ID of the user whose video image is to be stopped.
streamType	TUIRoomStreamType	Stream type.
callback	TUIRoomActionCallback	Result.

switchCamera

This API is used to switch between the front and rear cameras.

```
- (void)switchCamera:(BOOL)isFront;
```

The parameters are as detailed below:

Parameter	Type	Description
isFront	BOOL	YES: front camera; NO: rear camera.

Message Sending APIs

sendChatMessage

This API is used to broadcast a text message in a room, which is generally used for text chat.

```
- (void)sendChatMessage:(NSString *)message
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
message	NSString	Message content.
callback	TUIRoomActionCallback	Sending result.

Room Control APIs

muteUserMicrophone

This API is used to enable/disable the mic of the specified user.

```
- (void)muteUserMicrophone:(NSString *)userId
mute:(BOOL)mute
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.
mute	BOOL	Whether to disable.
callback	TUIRoomActionCallback	Result.

muteAllUsersMicrophone

This API is used to enable/disable the mic of all users.

```
- (void)muteAllUsersMicrophone:(BOOL)mute
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
mute	BOOL	Whether to disable.
callback	TUIRoomActionCallback	Result.

muteUserCamera

This API is used to enable/disable the camera of the specified user.

```
- (void)muteUserCamera:(NSString *)userId
mute:(BOOL)mute
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.
mute	BOOL	Whether to disable.
callback	TUIRoomActionCallback	Result.

muteAllUsersCamera

This API is used to enable/disable the camera of all users.

```
- (void)muteAllUsersCamera:(BOOL)mute
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
mute	BOOL	Whether to disable.
callback	TUIRoomActionCallback	Result.

muteChatRoom

This API is used to forbid/allow text chat.

```
- (void)muteChatRoom:(BOOL)mute
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
mute	BOOL	Whether to disable.
callback	TUIRoomActionCallback	Result.

kickOffUser

This API is used by the anchor to remove a member.

```
- (void)kickOffUser:(NSString *)userId
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.
callback	TUIRoomActionCallback	Result.

startCallingRoll

This API is used by the anchor to start roll call.

```
- (void)startCallingRoll:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomActionCallback	Result.

stopCallingRoll

This API is used by the anchor to stop roll call.

```
- (void)stopCallingRoll:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
callback	TUIRoomActionCallback	Result.

replyCallingRoll

This API is used by a member to reply to roll call.

```
- (void)replyCallingRoll:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomActionCallback	Result.

sendSpeechInvitation

This API is used by the anchor to invite a member to speak.

```
- (void)sendSpeechInvitation:(NSString *)userId
callback:(TUIRoomInviteeCallback)callback
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.
callback	TUIRoomInviteeCallback	Result.

cancelSpeechInvitation

This API is used by the anchor to cancel the speech invitation to a member.

```
- (void)cancelSpeechInvitation:(NSString *)userId
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.
callback	TUIRoomActionCallback	Result.

replySpeechInvitation

This API is used by a member to accept/reject the speech invitation from the anchor.

```
- (void)replySpeechInvitation:(BOOL)agree
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
agree	BOOL	Whether to approve.
callback	TUIRoomActionCallback	Result.

sendSpeechApplication

This API is used by a member to apply to speak.

```
- (void)sendSpeechApplication:(TUIRoomInviteeCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomInviteeCallback	Result.

cancelSpeechApplication

This API is used by a member to cancel the speech application.

```
- (void)cancelSpeechApplication:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomActionCallback	Result.

replySpeechApplication

This API is used by the anchor to approve/reject the speech application of a member.

```
- (void)replySpeechApplication:(BOOL)agree
userId:(NSString *)userId
```

```
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
agree	BOOL	Whether to approve.
userId	NSString	User ID.
callback	TUIRoomActionCallback	Result.

forbidSpeechApplication

This API is used by the anchor to forbid speech application.

```
- (void)forbidSpeechApplication:(BOOL)forbid
callback:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
forbid	BOOL	Whether to forbid.
callback	TUIRoomActionCallback	Result.

sendOffSpeaker

This API is used by the anchor to stop the speech of the specified member.

```
- (void)sendOffSpeaker:(NSString *)userId
callback:(TUIRoomInviteeCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.
callback	TUIRoomInviteeCallback	Result.

sendOffAllSpeakers

This API is used by the anchor to stop the speech of all members.

```
- (void)sendOffAllSpeakers:(TUIRoomInviteeCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomInviteeCallback	Result.

exitSpeechState

This API is used for a member to stop speaking and change their role to audience.

```
- (void)exitSpeechState:(TUIRoomActionCallback)callback;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomActionCallback	Result.

Screen Sharing APIs

startScreenCapture

This API is used to start screen sharing.

```
- (void)startScreenCapture:(TRTCVideoEncParam *)encParam API_AVAILABLE(ios(11.0));
```

The parameters are as detailed below:

Parameter	Type	Description
encParams	TRTCVideoEncParam	Sets encoding parameters for screen sharing.

Note :
For more information, please see TRTC SDK.

stopScreenCapture

This API is used to stop screen capturing.

```
- (void)stopScreenCapture API_AVAILABLE(ios(11.0));
```

Beauty Filter APIs

getBeautyManager

This API is used to get the beauty filter management object TXBeautyManager.

```
- (TXBeautyManager *)getBeautyManager;
```

You can do the following using `TXBeautyManager` :

- Set the beauty filter style and apply effects including skin brightening, rosy skin, eye enlarging, face slimming, chin slimming, chin lengthening/shortening, face shortening, nose narrowing, eye brightening, teeth whitening, eye bag removal, wrinkle removal, and smile line removal.
- Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose position, lip thickness, and face shape.
- Apply animated effects such as face widgets (materials).
- Add makeup effects.
- Recognize gestures.

Settings APIs

setVideoQosPreference

set QoS parameters

```
- (void)setVideoQosPreference:(TRTCNetworkQosParam *)preference;
```

The parameters are as detailed below:

Parameter	Type	Description
preference	TRTCNetworkQosParam	Network bandwidth limit policy.

setAudioQuality

This API is used to set audio quality.

```
- (void)setAudioQuality:(TRTCAudioQuality)quality;
```

The parameters are as detailed below:

Parameter	Type	Description
quality	TRTCAudioQuality	Audio quality. For more information, please see TRTC SDK.

setVideoResolution

This API is used to set the resolution.

```
- (void)setVideoResolution:(TRTCVideoResolution)resolution;
```

The parameters are as detailed below:

Parameter	Type	Description
resolution	TRTCVideoResolution	Video resolution. For more information, please see TRTC SDK.

setVideoFps

This API is used to set the frame rate.

```
- (void)setVideoFps:(int)fps;
```

The parameters are as detailed below:

Parameter	Type	Description
fps	int	Video capturing frame rate.

Note :

Recommended value: 15 or 20 fps. If the frame rate is lower than 5 fps, there will be obvious lag; if lower than 10 fps but higher than 5 fps, there will be slight lag; if higher than 20 fps, excessive resources will be wasted (the frame rate of movies is generally 24 fps).

setVideoBitrate

This API is used to set the bitrate.

```
- (void)setVideoBitrate:(int)bitrate;
```

The parameters are as detailed below:

Parameter	Type	Description
bitrate	int	Bitrate. The SDK encodes streams at the target video bitrate and will actively reduce the bitrate only if the network conditions are poor. For more information, please see TRTC SDK.

> Note :
> **Recommended value:** see the optimal bitrate for each tier in `TRTCVideoResolution` . You can also slightly increase the bitrate. For example, `TRTC_VIDEO_RESOLUTION_1280_720` corresponds to the target bitrate of 1,200 Kbps, and you can also set the bitrate to 1,500 Kbps for higher definition.

enableAudioEvaluation

This API is used to enable the volume reminder.

```
- (void)enableAudioEvaluation:(BOOL)enable;
```

The parameters are as detailed below:

Parameter	Type	Description
enable	BOOL	YES: enable; NO: disable

> Note :
> After this feature is enabled, the result of volume evaluation by the SDK will be obtained in `onUserVolumeUpdate` .

setAudioPlayVolume

This API is used to set the playback volume.

```
- (void)setAudioPlayVolume:(NSInteger)volume;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
volume	int	Playback volume. Value range: 0–100. Default value: 100.

setAudioCaptureVolume

This API is used to set the mic capturing volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Capture volume. Value range: 0–100. Default value: 100.

startFileDumping

This API is used to start audio recording.

```
- (void)startFileDumping:(TRTCAudioRecordingParams *)params;
```

The parameters are as detailed below:

Parameter	Type	Description
params	TRTCAudioRecordingParams	Audio recording parameters. For more information, please see TRTC SDK.

> Note :
> After this API is called, the SDK will record all audio of a call, including local audio, remote audio, and background music, into a file. This API works regardless of whether a user is in the room. When `leaveRoom` is called, audio recording will stop automatically.

stopFileDumping

This API is used to stop audio recording.

```
- (void)stopFileDumping;
```

SDK Version Acquisition APIs

getSdkVersion

This API is used to get SDK version information.

```
- (NSInteger)getSdkVersion;
```

Error Event Callbacks

onError

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

The parameters are as detailed below:

Parameter	Type	Description
code	NSInteger	Error code.
message	NSString	Error message

Basic Event Callbacks

onDestroyRoom

Room dismissal.

```
- (void)onDestroyRoom;
```

onUserVoiceVolume

User volume level.

```
- (void)onUserVoiceVolume:(NSString *)userId volume:(NSInteger)volume;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	NSString	User ID.
volume	NSInteger	User volume level. Value range: 0–100.

onRoomMasterChanged

Anchor change.

```
- (void)onRoomMasterChanged:(NSString *)previousUserId
currentUserId:(NSString *)currentUserId;
```

The parameters are as detailed below:

Parameter	Type	Description
previousUserId	NSString	Anchor's user ID before change.
currentUserId	NSString	Anchor's user ID after change.

Remote User Callbacks

onRemoteUserEnter

A remote user entered the room.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID

onRemoteUserLeave

A remote user exited the room.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID

onRemoteUserCameraAvailable

Whether a remote user enabled the camera.

```
- (void)onRemoteUserCameraAvailable:(NSString *)userId
available:(BOOL)available;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.
available	BOOL	YES: enabled; NO: disabled.

onRemoteUserScreenVideoAvailable

A member **enabled/disabled** video sharing.

```
- (void)onRemoteUserScreenVideoAvailable:(NSString *)userId
available:(BOOL)available;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.
available	BOOL	Whether screen sharing stream data is available.

onRemoteUserAudioAvailable

Whether a remote user is sending audio.

```
- (void)onRemoteUserAudioAvailable:(NSString *)userId
available:(BOOL)available;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	NSString	User ID.
available	BOOL	Whether audio data is available.

onRemoteUserEnterSpeechState

A remote user started speaking.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID

onRemoteUserExitSpeechState

A remote user stopped speaking.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID

Chat Room Message Event Callbacks

onReceiveChatMessage

Callback for receiving a text message.

```
- (void)onReceiveChatMessage:(NSString *)userId message:(NSString *)message;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID.

Parameter	Type	Description
message	NSString	Text message

Room Control Message Callbacks

onReceiveSpeechInvitation

A user received a speech invitation from the anchor.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	Anchor's user ID.

onReceiveInvitationCancelled

A user received a speech invitation cancellation from the anchor.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	Anchor's user ID.

OnReceiveSpeechApplication

The anchor received a speech application from a member.

```
void onReceiveSpeechApplication(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID

onSpeechApplicationCancelled

A user canceled a speech application.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID

onSpeechApplicationForbidden

The anchor forbidden a speech application.

```
- (void)onSpeechApplicationForbidden:(BOOL)isForbidden userId:(NSString *)userId;
```

The parameters are as detailed below:

Parameter	Type	Description
isForbidden	BOOL	Whether to forbid.
userId	NSString	User ID

onOrderedToExitSpeechState

A member was asked to stop speaking.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	Anchor's user ID.

onCallingRollStarted

The anchor started a roll call.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	Anchor's user ID.

onCallingRollStopped

The anchor stopped a roll call.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	Anchor's user ID.

onMemberReplyCallingRoll

A member replied to roll call.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	User ID

onChatRoomMuted

The anchor muted/unmuted the room.

```
- (void)onChatRoomMuted:(BOOL)muted userId:(NSString *)userId;
```

The parameters are as detailed below:

Parameter	Type	Description
muted	BOOL	Whether to disable.
userId	NSString	Anchor's user ID.

onMicrophoneMuted

The anchor disabled the mic.

```
- (void)onMicrophoneMuted:(BOOL)muted userId:(NSString *)userId;
```

The parameters are as detailed below:

Parameter	Type	Description
muted	BOOL	Whether to disable.
userId	NSString	Anchor's user ID.

onCameraMuted

The anchor disabled the camera.

```
- (void)onCameraMuted:(BOOL)muted userId:(NSString *)userId;
```

The parameters are as detailed below:

Parameter	Type	Description
muted	BOOL	Whether to disable.
userId	NSString	Anchor's user ID.

onReceiveKickedOff

The anchor removed a member.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	NSString	Anchor/Admin's user ID.

Statistics Collection and Quality Callbacks

onStatistics

Callback of technical metric statistics.

```
- (void)onStatistics:(TRTCStatistics *)statistics;
```

The parameters are as detailed below:

Parameter	Type	Description
statis	TRTCStatistics	Statistics.

onNetworkQuality

Network quality.

```
- (void)onNetworkQuality:(TRTCQualityInfo *)localQuality remoteQuality:(NSArray *)remoteQuality;
```

The parameters are as detailed below:

Parameter	Type	Description
localQuality	TRTCQualityInfo	Upstream network quality.
remoteQuality	NSArray<TRTCQualityInfo *>	Downstream network quality.

> Note :
> For more information, please see TRTC SDK.

Screen Sharing Event Callbacks

onScreenCaptureStarted

Screen sharing started.

```
- (void)onScreenCaptureStarted;
```

onScreenCaptureStopped

Screen sharing stopped.

```
- (void)onScreenCaptureStopped:(NSInteger)reason;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
reason	NSInteger	Reason for stop. 0: the user stopped proactively; 1: stopped due to preemption by another application.

TUIRoom (Android)

Last updated : 2022-03-01 12:18:48

1. `TUIRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). `TUIRoom` includes the following features:

- The anchor can create a room, and members can enter the room ID to join the room.
- Room members can share their screens with each other.
- All users can send various text and custom messages.

2. `TUIRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Group Audio/Video Room (for Android).

- TRTC SDK: the TRTC SDK is used as a low-latency audio/video room component.
- IM SDK: the IM SDK for **Android** is used to implement the chat room feature.

`TUIRoom` API Overview

Basic functions of `TUIRoomCore`

API	Description
getInstance	Gets a singleton object.
destroyInstance	Terminates a singleton object.
setListener	Sets event callbacks.

Room APIs

API	Description
createRoom	Creates a room (called by anchor).
destroyRoom	Terminates a room (called by anchor).
enterRoom	Enters a room (called by room member).
leaveRoom	Exits a room (called by other room members).
getRoomInfo	Gets the room information.

API	Description
getRoomUsers	Gets the information of all users in the room.
getUserInfo	Gets the information of a user.
transferRoomMaster	Transfers the anchor permission (called by anchor).

Local audio/video operation APIs

API	Description
startCameraPreview	Enables the preview image of local video.
stopCameraPreview	Stops local video capturing and preview.
startLocalAudio	Enables mic capturing.
stopLocalAudio	Stops mic capturing.
setVideoMirror	Sets the mirroring preview mode of the local image.
setSpeaker	Sets whether to use the speaker or receiver.

Remote user APIs

API	Description
startRemoteView	Subscribes to and plays back the remote video image of a specified member.
stopRemoteView	Unsubscribes from and stops the playback of a remote video image.

Chat message sending APIs

API	Description
sendChatMessage	Sends a chat message.
sendCustomMessage	Sends a custom message.

Room control APIs

API	Description

API	Description
muteUserMicrophone	Enables/Disables the mic of a specified user.
muteAllUsersMicrophone	Enables/Disables the mic of all users and syncs the status to room information.
muteUserCamera	Enables/Disables the camera of a specified user.
muteAllUsersCamera	Enables/Disables the camera of all users and syncs the status to room information.
muteChatRoom	Mutes/Unmutes the chat room (called by anchor).
kickOffUser	Removes a specified user in the room (called by anchor).
startCallingRoll	Starts calling roll by the anchor.
stopCallingRoll	Stops calling roll by the anchor.
replyCallingRoll	Replies to roll call by a member.
sendSpeechInvitation	Invites a member to speak by the anchor.
cancelSpeechInvitation	Cancels invitation to a member for speech by the anchor.
replySpeechInvitation	Accepts/Rejects the speech invitation from the anchor by a member.
sendSpeechApplication	Applies for speech by a member.
replySpeechApplication	Approves/Rejects the speech application of a member by the anchor.
forbidSpeechApplication	Forbids speech application by the anchor.
sendOffSpeaker	Stops the speech of a member by the anchor.
sendOffAllSpeakers	Stops the speech of all members by the anchor.
exitSpeechState	Stops speech by a member and changes their role to audience.

Screen sharing APIs

API	Description
startScreenCapture	Starts screen sharing.
stopScreenCapture	Stops screen sharing.

Beauty filter APIs

API	Description
getBeautyManager	Gets the beauty filter management object TXBeautyManager.

Settings APIs

API	Description
setVideoQosPreference	Sets network bandwidth limit parameters.

SDK version acquisition APIs

API	Description
getSDKVersion	Gets the SDK version.

`TUIRoomCoreListener` API Overview

Callbacks for error events

API	Description
onError	Error

Basic event callbacks

API	Description
onDestroyRoom	Room dismissal.
onUserVoiceVolume	Volume level.
onRoomMasterChanged	Anchor change.

Remote user event callbacks

API	Description
onRemoteUserEnter	A remote user entered the room.
onRemoteUserLeave	A remote user exited the room.

API	Description
onRemoteUserCameraAvailable	Whether a remote user enabled the camera.
onRemoteUserScreenVideoAvailable	Whether a remote user enabled screen sharing.
onRemoteUserAudioAvailable	Whether a remote user enabled sending audio.
onRemoteUserEnterSpeechState	A remote user started speaking.
onRemoteUserExitSpeechState	A remote user stopped speaking.

Message event callback APIs

API	Description
onReceiveChatMessage	A text message was received.
onReceiveRoomCustomMsg	A custom message was received.

Room control event callbacks

API	Description
onReceiveSpeechInvitation	A member received a speech invitation from the anchor.
onReceiveInvitationCancelled	A member received a speech invitation cancellation from the anchor.
onReceiveSpeechApplication	The anchor received a speech application from a member.
onSpeechApplicationCancelled	A member canceled a speech application.
onSpeechApplicationForbidden	The anchor rejected a speech application.
onOrderedToExitSpeechState	A member was asked to stop speaking.
onCallingRollStarted	The anchor started a roll call.
onCallingRollStopped	The anchor stopped a roll call.
onMemberReplyCallingRoll	A member replied to roll call.
onChatRoomMuted	The anchor muted/unmuted the room.
onMicrophoneMuted	The anchor disabled the mic.

API	Description
onCameraMuted	The anchor disabled the camera.
onReceiveKickedOff	The anchor removed a member.

Callback APIs for statistics on network quality and technical metrics

API	Description
onStatistics	Statistics on technical metrics.
onNetworkQuality	Network quality.

Screen sharing event callbacks

API	Description
onScreenCaptureStarted	Screen sharing started.
onScreenCaptureStopped	Screen sharing stopped.

Basic Functions of `TUIRoomCore`

getInstance

This API is used to get a TUIRoomCore singleton object.

```
public static TUIRoomCore getInstance(Context context);
```

The parameters are as detailed below:

Parameter	Type	Description
context	Context	Android context, which will be converted to `ApplicationContext` for the system APIs to call.

destroyInstance

```
void destroyInstance();
```

setListener

This API is used to set the event callback of TUIRoomCore. You can use `TUIRoomCoreListener` to get different status notifications of TUIRoomCore.

```
void setListener(TUIRoomCoreListener listener);
```

The parameters are as detailed below:

Parameter	Type	Description
listener	TUIRoomCoreListener	Event callback class.

createRoom

This API is used to create a room (called by the anchor).

```
void createRoom(String roomId, TUIRoomCoreDef.SpeechMode speechMode, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	String	Room ID. You need to assign and manage the IDs in a centralized manner.
speechMode	TUIRoomCoreDef.SpeechMode	Speech mode.
callback	TUIRoomCoreCallback.ActionCallback	Room creation result.

Generally, the anchor calls the APIs in the following steps:

1. The **anchor** calls `createRoom()` to create a room, the result of which is returned via `TUIRoomCoreCallback.ActionCallback` .
2. The **anchor** calls `startCameraPreview()` to enable camera capturing and preview.
3. The **anchor** calls `startLocalAudio()` to enable the local mic.

destroyRoom

This API is used to terminate a room (called by the anchor).

```
void destroyRoom(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	UIRoomCoreCallback.ActionCallback	Room termination result.

enterRoom

This API is used to enter a room (called by a member).

```
void enterRoom(String roomId, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	String	Room ID.
callback	UIRoomCoreCallback.ActionCallback	Result.

Generally, a member enters a room in the following steps:

1. The **member** calls `enterRoom` and passes in `roomId` to enter the room.
2. The **member** calls `startCameraPreview()` to enable camera preview and calls startLocalAudio()` to enable mic capturing.
3. The **member** receives the `onRemoteUserCameraAvailable` event and calls `startRemoteView()` to start playback.

leaveRoom

This API is used to exit a room (called by a member).

```
void leaveRoom(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	UIRoomCoreCallback.ActionCallback	Result.

getRoomInfo

This API is used to get the room information.

```
TUIRoomCoreDef.RoomInfo getRoomInfo();
```

getRoomUsers

This API is used to get the information of all users in the room.

```
List getRoomUsers();
```

getUserInfo

This API is used to get the information of a user in the room.

```
void getUserInfo(String userId, TUIRoomCoreCallback.UserInfoCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
callback	UIRoomCoreCallback.UserInfoCallback	Room member details.

setSelfProfile

Set User Info

```
void setSelfProfile(String userName, String avatarURL, TUIRoomCoreCallback.ActionCallback callbac
k);
```

The parameters are listed below:

Parameter	Type	Description
userName	String	User name.
avatarURL	String	User profile photo URL.
callback	TUIRoomCoreCallback.ActionCallback	Whether the setting succeeded.

transferRoomMaster

This API is used to transfer a group to another user.

```
void transferRoomMaster(String userId, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

Local Push APIs

startCameraPreview

This API is used to start the preview of the local camera.

```
void startCameraPreview(boolean isFront, TXCloudVideoView view);
```

The parameters are as detailed below:

Parameter	Type	Description
isFront	Boolean	`true` : front camera; `false` : rear camera
view	TXCloudVideoView	The control that loads video images

stopCameraPreview

This API is used to stop the preview of the local camera.

```
void stopCameraPreview();
```

startLocalAudio

This API is used to start mic capturing.

```
void startLocalAudio(int quality);
```

The parameters are as detailed below:

Parameter	Type	Description
quality	int	Captured sound quality: • TRTC_AUDIO_QUALITY_MUSIC • TRTC_AUDIO_QUALITY_DEFAULT • TRTC_AUDIO_QUALITY_SPEECH

stopLocalAudio

This API is used to stop mic capturing.

```
void stopLocalAudio();
```

setVideoMirror

This API is used to set the mirroring preview mode of local video image.

```
void setVideoMirror(int type);
```

The parameters are as detailed below:

Parameter	Type	Description
type	int	Mirroring type.

setSpeaker

This API is used to set whether to use the speaker or receiver.

```
void setSpeaker(boolean isUseSpeaker);
```

The parameters are as detailed below:

Parameter	Type	Description
isUseSpeaker	boolean	true: device speaker; false: device receiver.

Remote User APIs

startRemoteView

This API is used to subscribe to a remote user's video stream.

```
void startRemoteView(String userId, TXCloudVideoView view, TUIRoomCoreDef.SteamType streamType, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	String	ID of the user whose video image is to be played back.
view	TXCloudVideoView	The control that loads video images.
streamType	TUIRoomCoreDef.SteamType	Stream type.
callback	TUIRoomCoreCallback.ActionCallback	Result.

stopRemoteView

This API is used to unsubscribe from and stop the playback of a remote video image.

```
void stopRemoteView(String userId, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user whose video image is to be stopped.
callback	TUIRoomCoreCallback.ActionCallback	Result.

switchCamera

This API is used to switch between the front and rear cameras.

```
void switchCamera(boolean isFront);
```

The parameters are as detailed below:

Parameter	Type	Description
isFront	Boolean	true: front camera; false: rear camera.

Message Sending APIs

sendChatMessage

This API is used to broadcast a text message in a room, which is generally used for text chat.

```
void sendChatMessage(String message, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Message content.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the operation.

sendCustomMessage

Sending Custom Messages

```
void sendCustomMessage(String data, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
data	String	Message content.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the operation.

Room Control APIs

muteUserMicrophone

This API is used to enable/disable the mic of the specified user.

```
void muteUserMicrophone(String userId, boolean mute, TUIRoomCoreCallback.ActionCallback callback)
;
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.
mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

muteAllUsersMicrophone

This API is used to enable/disable the mic of all users.

```
void muteAllUsersMicrophone(boolean mute, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

muteUserCamera

This API is used to enable/disable the camera of the specified user.

```
void muteUserCamera(String userId, boolean mute, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are detailed below:

Parameter	Type	Description
userId	String	User ID.
mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

muteAllUsersCamera

This API is used to enable/disable the camera of all users.

```
void muteAllUsersCamera(boolean mute, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

muteChatRoom

This API is used to forbid/allow text chat.

```
void muteChatRoom(boolean mute, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

kickOffUser

This API is used by the anchor to remove a member.

```
void kickOffUser(String userId, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

startCallingRoll

This API is used by the anchor to start roll call.

```
void startCallingRoll(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

stopCallingRoll

This API is used by the anchor to stop roll call.

```
void stopCallingRoll(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

replyCallingRoll

This API is used by a member to reply to roll call.

```
void replyCallingRoll(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

sendSpeechInvitation

This API is used by the anchor to invite a member to speak.

```
void sendSpeechInvitation(String userId, TUIRoomCoreCallback.InvitationCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.
callback	TUIRoomCoreCallback.InvitationCallback	Result.

cancelSpeechInvitation

This API is used by the anchor to cancel the speech invitation to a member.

```
void cancelSpeechInvitation(String userId, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

replySpeechInvitation

This API is used by a member to accept/reject the speech invitation from the anchor.

```
void replySpeechInvitation(boolean agree, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
agree	boolean	Whether the invitation was accepted.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

sendSpeechApplication

This API is used by a member to apply to speak.

```
void sendSpeechApplication(TUIRoomCoreCallback.InvitationCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.InvitationCallback	Result.

cancelSpeechApplication

This API is used by a member to cancel the speech application.

```
void cancelSpeechApplication(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

replySpeechApplication

This API is used by the anchor to approve/reject the speech application of a member.

```
void replySpeechApplication(boolean agree, String userId, TUIRoomCoreCallback.ActionCallback call
back);
```

The parameters are as detailed below:

Parameter	Type	Description
agree	Boolean	Whether the invitation was accepted.
userId	String	User ID.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

forbidSpeechApplication

This API is used by the anchor to forbid speech application.

```
void forbidSpeechApplication(boolean forbid, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
forbid	boolean	Whether speech application is forbidden.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

sendOffSpeaker

This API is used by the anchor to stop the speech of the specified member.

```
void sendOffSpeaker(String userId, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

sendOffAllSpeakers

This API is used by the anchor to stop the speech of all members.

```
void sendOffAllSpeakers(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

exitSpeechState

This API is used for a member to stop speaking and change their role to audience.

```
void exitSpeechState(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

Screen Sharing APIs

startScreenCapture

This API is used to start screen sharing.

```
void startScreenCapture(TRTCCloudDef.TRTCVideoEncParam encParams, TRTCCloudDef.TRTCScreenSharePar
ams screenShareParams);
```

The parameters are as detailed below:

Parameter	Type	Description
encParams	TRTCCloudDef.TRTCVideoEncParam	Screen sharing encoding parameters. We recommend you use the above configuration. If you set `encParams` to `null` , the encoding parameter settings before `startScreenCapture` is called will be used.

Parameter	Type	Description
screenShareParams	TRTCCloudDef.TRTCScreenShareParams	Special screen sharing configuration. We recommend you set `floatingView` which can prevent the application from being killed by the system and help protect user privacy.

> Note :
> For more information, please see TRTC SDK.

stopScreenCapture

This API is used to stop screen capturing.

```
void stopScreenCapture();
```

Beauty Filter APIs

getBeautyManager

This API is used to get the beauty filter management object TXBeautyManager.

```
TXBeautyManager getBeautyManager();
```

You can do the following using `TXBeautyManager` :

- Set the beauty filter style and apply effects including skin brightening, rosy skin, eye enlarging, face slimming, chin slimming, chin lengthening/shortening, face shortening, nose narrowing, eye brightening, teeth whitening, eye bag removal, wrinkle removal, and smile line removal.
- Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose position, lip thickness, and face shape.
- Apply animated effects such as face widgets (materials).
- Add makeup effects.
- Recognize gestures.

Settings APIs

setVideoQosPreference

set QoS parameters

```
void setVideoQosPreference(TRTCCloudDef.TRTCNetworkQosParam preference);
```

The parameters are as detailed below:

Parameter	Type	Description
preference	TRTCCloudDef.TRTCNetworkQosParam	Network bandwidth limit policy.

setAudioQuality

This API is used to set audio quality.

```
void setAudioQuality(int quality);
```

The parameters are as detailed below:

Parameter	Type	Description
quality	int	Audio quality. For more information, please see TRTC SDK.

setVideoResolution

This API is used to set the resolution.

```
void setVideoResolution(int resolution);
```

The parameters are as detailed below:

Parameter	Type	Description
resolution	int	Video resolution. For more information, please see TRTC SDK.

setVideoFps

This API is used to set the frame rate.

```
void setVideoFps(int fps);
```

The parameters are as detailed below:

Parameter	Type	Description
fps	int	Video capturing frame rate.

> Note :
> **Recommended value:** 15 or 20 fps. If the frame rate is lower than 5 fps, there will be obvious lagging; if lower than 10 fps but higher than 5 fps, there will be slight lagging; if higher than 20 fps, too many resources will be wasted (the frame rate of movies is generally 24 fps).

setVideoBitrate

This API is used to set the bitrate.

```
void setVideoBitrate(int bitrate);
```

The parameters are as detailed below:

Parameter	Type	Description
bitrate	int	Bitrate. The SDK encodes streams at the target video bitrate and will actively reduce the bitrate only if the network conditions are poor. For more information, please see TRTC SDK.

> Note :
> **Recommended value:** please see the optimal bitrate for each tier in `TRTCVideoResolution` . You can also slightly increase the optimal bitrate. For example, `TRTC_VIDEO_RESOLUTION_1280_720` corresponds to the target bitrate of 1,200 Kbps, and you can also set the bitrate to 1,500 Kbps for higher definition.

enableAudioEvaluation

This API is used to enable the volume reminder.

```
void enableAudioEvaluation(boolean enable);
```

The parameters are as detailed below:

Parameter	Type	Description
enable	boolean	true: enable; false: disable.

> Note :
> After this feature is enabled, the result of volume evaluation by the SDK will be obtained in `onUserVolumeUpdate` .

setAudioPlayVolume

This API is used to set the playback volume.

```
void setAudioPlayVolume(int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Playback volume. Value range: 0–100. Default value: 100.

setAudioCaptureVolume

This API is used to set the mic capturing volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Capture volume. Value range: 0–100. Default value: 100.

startFileDumping

This API is used to start audio recording.

```
void startFileDumping(TRTCCloudDef.TRTCAudioRecordingParams trtcAudioRecordingParams);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
trtcAudioRecordingParams	TRTCCloudDef.TRTCAudioRecordingParams	Audio recording parameters. For more information, please see TRTC SDK.

> Note :
> After this API is called, the SDK will record all audio of a call, including local audio, remote audio, and background music, into a file. This API works regardless of whether a user is in the room. When `leaveRoom` is called, audio recording will stop automatically.

stopFileDumping

This API is used to stop audio recording.

```
void stopFileDumping();
```

SDK Version Acquisition APIs

getSdkVersion

This API is used to get SDK version information.

```
int getSdkVersion();
```

Error Event Callbacks

onError

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

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
code	int	Error code
message	String	Error message

Basic Event Callbacks

onDestroyRoom

Room dismissal.

```
void onDestroyRoom();
```

onUserVoiceVolume

User volume level.

```
void onUserVoiceVolume(String userId, int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
volume	int	User volume level. Value range: 0–100.

onRoomMasterChanged

Anchor change.

```
void onRoomMasterChanged(String previousUserId, String currentUserId);
```

The parameters are as detailed below:

Parameter	Type	Description
previousUserId	String	Anchor's user ID before change.
currentUserId	String	Anchor's user ID after change.

Remote User Callbacks

onRemoteUserEnter

A remote user entered the room.

```
void onRemoteUserEnter(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID

onRemoteUserLeave

A remote user exited the room.

```
void onRemoteUserLeave(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID

onRemoteUserCameraAvailable

Whether a remote user enabled the camera.

```
void onRemoteUserCameraAvailable(String userId, boolean available);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.
available	boolean	true: enabled; false: disabled.

onRemoteUserScreenVideoAvailable

A member **enabled/disabled** video sharing.

```
void onRemoteUserScreenVideoAvailable(String userId, boolean available);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.
available	boolean	Whether screen sharing stream data is available.

onRemoteUserAudioAvailable

Whether a remote user is sending audio.

```
void onRemoteUserAudioAvailable(String userId, boolean available);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.
available	boolean	Whether audio data is available.

onRemoteUserEnterSpeechState

A remote user started speaking.

```
void onRemoteUserEnterSpeechState(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID

onRemoteUserExitSpeechState

A remote user stopped speaking.

```
void onRemoteUserExitSpeechState(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	String	User ID

Chat Room Message Event Callbacks

onReceiveChatMessage

Callback for receiving a text message.

```
void onReceiveChatMessage(String userId, String message);
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID
message	String	Text message

onReceiveRoomCustomMsg

Callback for receiving a custom message.

```
void onReceiveRoomCustomMsg(String userId, String data);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
message	String	Custom message content.

Room Control Message Callbacks

onReceiveSpeechInvitation

A user received a speech invitation from the anchor.

```
void onReceiveSpeechInvitation(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Anchor's user ID.

onReceiveInvitationCancelled

A user received a speech invitation cancellation from the anchor.

```
void onReceiveInvitationCancelled(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Anchor's user ID.

onReceiveSpeechApplication

The anchor received a speech application from a member.

```
void onReceiveSpeechApplication(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID

onSpeechApplicationCancelled

A user canceled a speech application.

```
void onSpeechApplicationCancelled(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID

onSpeechApplicationForbidden

The anchor forbidden a speech application.

```
void onSpeechApplicationForbidden(boolean isForbidden);
```

The parameters are as detailed below:

Parameter	Type	Description
isForbidden	boolean	Whether to forbid.

onOrderedToExitSpeechState

A member was asked to stop speaking.

```
void onOrderedToExitSpeechState(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Anchor's user ID.

onCallingRollStarted

The anchor started a roll call.

```
void onCallingRollStarted(String userId);
```

onCallingRollStopped

The anchor stopped a roll call.

```
void onCallingRollStopped(String userId);
```

onMemberReplyCallingRoll

A member replied to roll call.

```
void onMemberReplyCallingRoll(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID

onChatRoomMuted

The anchor muted/unmuted the room.

```
void onChatRoomMuted(boolean muted);
```

The parameters are as detailed below:

Parameter	Type	Description
muted	boolean	Whether to disable.

onMicrophoneMuted

The anchor disabled the mic.

```
void onMicrophoneMuted(boolean muted);
```

The parameters are as detailed below:

Parameter	Type	Description
muted	boolean	Whether to disable.

onCameraMuted

The anchor disabled the camera.

```
void onCameraMuted(boolean muted);
```

The parameters are as detailed below:

Parameter	Type	Description
muted	boolean	Whether to disable.

onReceiveKickedOff

The anchor removed a member.

```
void onReceiveKickedOff(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	String	Anchor/Admin's user ID.

Statistics Collection and Quality Callbacks

onStatistics

Callback of technical metric statistics.

```
void onStatistics(TRTCStatistics statistics);
```

The parameters are as detailed below:

Parameter	Type	Description
statis	TRTCStatistics	Statistics.

onNetworkQuality

Network quality.

```
void onNetworkQuality(TRTCCloudDef.TRTCQuality localQuality, List remoteQuality);
```

The parameters are as detailed below:

Parameter	Type	Description
localQuality	TRTCCloudDef.TRTCQuality	Upstream network quality.
remoteQuality	List<TRTCCloudDef.TRTCQuality>	Downstream network quality.

Note :

For more information, please see TRTC SDK.

Screen Sharing Event Callbacks

onScreenCaptureStarted

Screen sharing started.

```
void onScreenCaptureStarted();
```

onScreenCaptureStopped

Screen sharing stopped.

```
void onScreenCaptureStopped(int reason);
```

The parameters are as detailed below:

Parameter	Type	Description
reason	int	Reason for stop. 0: the user stopped proactively; 1: stopped due to preemption by another application.

TRTCMeeting (Flutter)

Last updated : 2022-03-01 12:18:48

`TRTCMeeting` has the following features based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM):

- The anchor can create a meeting room, and the participants can enter the room ID to join the meeting.
- The participants can share their screens with each other.
- All users can send various text and custom messages.

`TRTCMeeting` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, see Group Audio/Video Room (for Flutter).

- TRTC SDK: The TRTC SDK is used as the low-latency video conferencing component.
- IM SDK: The `MeetingRoom` feature of the IM SDK is used to implement chat rooms in meetings.

TRTCMeeting API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
registerListener	Sets event listener.
unRegisterListener	Terminates event listener.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets the profile.

Meeting room APIs

API	Description
createMeeting	Creates a meeting room (called by anchor).

API	Description
destroyMeeting	Terminates a meeting room (called by anchor).
enterMeeting	Enters a meeting room (called by participant).
leaveMeeting	Leaves a meeting room (called by participant).
getUserInfoList	Gets the list of all users in room. This API will take effect only if it is called after `enterMeeting()` succeeds.
getUserInfo	Gets the details of a specified user in the room. This API will take effect only if it is called after `enterMeeting()` succeeds.

Remote user APIs

API	Description
startRemoteView	Plays back the remote video image of a specified member.
stopRemoteView	Stops playing back the remote video image of a specified member.
setRemoteViewParam	Sets the remote image rendering parameters of a specified member.
muteRemoteAudio	Mutes/Unmutes a specified member's remote audio
muteAllRemoteAudio	Mutes/Unmutes all members' remote audio.
muteRemoteVideoStream	Pauses/Resumes a specified member's remote video stream.
muteAllRemoteVideoStream	Pauses/Resumes all members' remote video streams.

Local video operation APIs

API	Description
startCameraPreview	Enables preview of the local video.
stopCameraPreview	Stops local video capturing and preview.
switchCamera	Switches between the front and rear cameras.
setVideoEncoderParam	Sets video encoder parameters.
setLocalViewMirror	Sets the mirroring preview mode of local image.

Local audio APIs

API	Description
startMicrophone	Starts mic capturing.
stopMicrophone	Stops mic capturing.
muteLocalAudio	Mutes/Unmutes local audio.
setSpeaker	Turns the device speaker or device receiver on.
setAudioCaptureVolume	Sets mic capturing volume.
setAudioPlayoutVolume	Sets playback volume.
startAudioRecording	Starts audio recording.
stopAudioRecording	Stops audio recording.
enableAudioVolumeEvaluation	Enables volume reminder.

Screen sharing APIs

API	Description
startScreenCapture	Starts screen sharing.
stopScreenCapture	Stops screen sharing.
pauseScreenCapture	Pauses screen sharing.
resumeScreenCapture	Resumes screen sharing.

Management object acquisition APIs

API	Description
getDeviceManager	Gets the device management object TXDeviceManager.
getBeautyManager	Gets the beauty filter management object TXBeautyManager.

Message sending APIs

API	Description

API	Description
sendRoomTextMsg	Broadcasts a text message in a meeting, which is generally used for chat.
sendRoomCustomMsg	Sends a custom text message.

`TRTCLiveRoomDelegate` API Overview

Common event callback APIs

API	Description
onError	Error
onWarning	Warning
onKickedOffline	Callback for being kicked offline because another user logged in to the account.

Meeting room event callbacks

API	Description
onRoomDestroy	Callback for meeting room termination.
onNetworkQuality	Callback for network status.
onUserVolumeUpdate	User volume

Member entry/exit event callbacks

API	Description
onEnterRoom	You entered the meeting.
onLeaveRoom	You left the meeting.
onUserEnterRoom	A new member joined the meeting.
onUserLeaveRoom	A member left the meeting.

Member audio/video event callbacks

API	Description
onUserAudioAvailable	A member turned the mic on/off.
onUserVideoAvailable	A member turned the camera on/off.
onUserSubStreamAvailable	A member enabled/disabled the substream image.

Message event callback APIs

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

Screen sharing event callbacks

API	Description
onScreenCaptureStarted	Screen sharing started.
onScreenCapturePaused	Screen sharing paused.
onScreenCaptureResumed	Screen sharing resumed.
onScreenCaptureStoped	Screen sharing stopped.

Basic SDK APIs

sharedInstance

This API is used to get the TRTCMeeting singleton object.

```
static Future sharedInstance();
```

destroySharedInstance

This API is used to terminate the TRTCMeeting singleton object.

```
static void destroySharedInstance();
```

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

registerListener

This API is used to set an event listener. You can use `TRTCMeetingDelegate` to get various status notifications of TRTCMeeting.

```
void registerListener(MeetingListenerFunc func);
```

Note :
`func` is the delegation callback of `TRTCMeeting` .

unRegisterListener

This API is used to terminate an event listener.

```
void unRegisterListener(MeetingListenerFunc func);
```

login

This API is used to log in to the Tencent backend server.

```
Future login(int sdkAppId, String userId, String userSig);
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	int	You can view the `SDKAppID` via **Application Management** > **Application Info** in the TRTC console.
userId	String	ID of 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 more information on how to get it, please see UserSig.

logout

This API is used to log out of the Tencent backend server.

```
Future logout();
```

setSelfProfile

This API is used to set the profile.

```
Future setSelfProfile(String userName, String avatarURL);
```

The parameters are as detailed below:

Parameter	Type	Description
userName	String	Username.
avatarURL	String	User profile photo URL.

Meeting Room APIs

createMeeting

This API is used to create a meeting (called by the anchor).

```
Future createMeeting(int roomId);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Meeting room ID. You need to assign and manage the IDs in a centralized manner.

Generally, the anchor calls the APIs in the following steps:

1. The **anchor** calls `createMeeting()` and passes in `roomId` to create a meeting. No matter whether the room is successfully created, the result will be notified to the anchor through `ActionCallback` .
2. The **anchor** calls `startCameraPreview()` to enable camera preview. At this time, the beauty filter parameters can be adjusted.
3. The **anchor** calls `startMicrophone()` to enable mic capturing.

destroyMeeting

This API is used to terminate a meeting room (called by the anchor). After creating a meeting, the anchor can call this API to terminate it.

```
Future destroyMeeting(int roomId);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Meeting room ID. You need to assign and manage the IDs in a centralized manner.

enterMeeting

This API is used to enter a meeting room (called by participant).

```
Future enterMeeting(int roomId);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Meeting room ID.

Generally, the participant joins a meeting in the following steps:

1. The **participant** calls `enterMeeting()` and passes in `roomId` to enter the meeting room.
2. The **participant** calls `startCameraPreview()` to enable camera preview and calls `startMicrophone()` to enable mic capturing.
3. The **participant** receives the `onUserVideoAvailable` event and calls `startRemoteView()` and passes in the `userId` of the target member to start playback.

leaveMeeting

This API is used to leave a meeting room (called by participant).

```
Future leaveMeeting();
```

getUserInfoList

This API is used to get the list of all users in the room. It will take effect only if it is called after `enterMeeting()` succeeds.

```
Future getUserInfoList(List<String> userIdList);
```

The parameters are as detailed below:

Parameter	Type	Description
userIdList	List	List of `userId` values to obtain.

getUserInfo

This API is used to get the details of a specified user in the room. It will take effect only if it is called after `enterMeeting()` succeeds.

```
Future getUserInfo(String userId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Specified user ID.

Remote User APIs

startRemoteView

This API is used to play back the remote video image of a specified member.

```
Future<void> startRemoteView(String userId, int streamType, int viewId);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Specified user ID.
streamType	int	Type of the video stream to watch. For more information, please see TRTC SDK.
viewId	int	`viewId` generated by `TRTCCloudVideoView` .

stopRemoteView

This API is used to stop playing back the remote video image of a specified member.

```
Future<void> stopRemoteView(String userId, int streamType);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Specified user ID.
streamType	int	Type of the video stream to watch. For more information, please see TRTC SDK.

setRemoteViewParam

This API is used to set the remote video image rendering parameters of a specified member.

```
Future<void> setRemoteViewParam(String userId, int streamType,
{int fillMode, int rotation, int mirrorType});
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Specified user ID.
streamType	int	Type of the video stream to watch. For more information, please see TRTC SDK.
fillMode	int	Video image rendering mode: fill (default value) or fit. For more information, please see TRTC SDK.
rotation	int	Clockwise video image rotation angle. For more information, please see TRTC SDK.
mirrorType	int	Mirroring mode. For more information, please see TRTC SDK.

muteRemoteAudio

This API is used to mute/unmute a specified member's remote audio.

```
Future<void> muteRemoteAudio(String userId, bool mute);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	String	Specified user ID.
mute	boolean	true: mute; false: unmute.

muteAllRemoteAudio

This API is used to mute/unmute all members' remote audio.

```
Future<void> muteAllRemoteAudio(bool mute);
```

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	true: mute; false: unmute.

muteRemoteVideoStream

This API is used to pause/resume a specified member's remote video stream.

```
Future<void> muteRemoteVideoStream(String userId, bool mute);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Specified user ID.
mute	boolean	true: pause; false: resume.

muteAllRemoteVideoStream

This API is used to pause/resume all members' remote video streams.

```
Future<void> muteAllRemoteVideoStream(bool mute);
```

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	true: pause; false: resume.

Local Video Operation APIs

startCameraPreview

This API is used to enable local video preview.

```
Future<void> startCameraPreview(bool isFront, int viewId);
```

The parameters are as detailed below:

Parameter	Type	Description
isFront	boolean	true: front camera; false: rear camera.
viewId	int	`viewId` generated by `TRTCCloudVideoView` .

stopCameraPreview

This API is used to stop local video capturing and preview.

```
Future<void> stopCameraPreview();
```

switchCamera

This API is used to switch between the front and rear cameras.

```
Future<void> switchCamera(bool isFront);
```

The parameters are as detailed below:

Parameter	Type	Description
isFront	boolean	true: front camera; false: rear camera.

setVideoEncoderParam

This API is used to set video encoder parameters.

```
Future<void> setVideoEncoderParam({
int videoFps,
int videoBitrate,
int videoResolution,
int videoResolutionMode,
});
```

The parameters are as detailed below:

Parameter	Type	Description
videoFps	int	Video capturing frame rate.
videoBitrate	int	Bitrate. The SDK encodes streams at the target video bitrate and will actively reduce the bitrate only if the network conditions are poor.
videoResolution	int	Resolution.
videoResolutionMode	int	Resolution mode.

Note :
For more information, please see TRTC SDK.

setLocalViewMirror

This API is used to set the mirroring preview mode of local video image.

```
Future<void> setLocalViewMirror(bool isMirror);
```

The parameters are as detailed below:

Parameter	Type	Description
isMirror	boolean	Whether to enable mirroring preview mode. `true` : yes; `false` : no.

Local Audio APIs

startMicrophone

This API is used to start mic capturing.

```
Future<void> startMicrophone({int quality});
```

The parameters are as detailed below:

Parameter	Type	Description
quality	int	Audio quality. For more information, please see TRTC SDK.

stopMicrophone

This API is used to stop mic capturing.

```
Future<void> stopMicrophone();
```

muteLocalAudio

This API is used to mute/unmute local audio.

```
Future<void> muteLocalAudio(bool mute);
```

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	true: mute. false: unmute.

setSpeaker

This API is used to turn the speaker or receiver on.

```
Future<void> setSpeaker(bool useSpeaker);
```

The parameters are as detailed below:

Parameter	Type	Description
useSpeaker	boolean	`true` : speaker; `false` : receiver

setAudioCaptureVolume

This API is used to set the mic capturing volume.

```
Future<void> setAudioCaptureVolume(int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Capturing volume. Value range: 0-100 (default: 100)

setAudioPlayoutVolume

This API is used to set the playback volume.

```
Future<void> setAudioPlayoutVolume(int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Playback volume. Value range: 0-100 (default: 100)

startAudioRecording

This API is used to start audio recording.

```
Future<int?> startAudioRecording(String filePath);
```

The parameters are as detailed below:

Parameter	Type	Description
filePath	String	Storage path of the audio recording file. Please specify it by yourself and ensure that the specified path exists and is writable. This path must be accurate to the file name and extension. The extension determines the format of the audio recording file. Currently, supported formats include PCM, WAV, and AAC.

stopAudioRecording

This API is used to stop audio recording.

```
Future<void> stopAudioRecording();
```

enableAudioVolumeEvaluation

This API is used to enable the volume reminder.

```
Future<void> enableAudioVolumeEvaluation(int intervalMs);
```

The parameters are as detailed below:

Parameter	Type	Description
intervalMs	int	Determines the interval in ms for triggering the `onUserVoiceVolume` callback. The minimum interval is 100 ms. If the value is smaller than 0, the callback will be disabled. We recommend you set this parameter to 300 ms.

Screen Sharing APIs

startScreenCapture

This API is used to start screen sharing.

```
Future<void> startScreenCapture({
int videoFps,
int videoBitrate,
int videoResolution,
int videoResolutionMode,
String appGroup,
});
```

The parameters are as detailed below:

Parameter	Type	Description
videoFps	int	Video capturing frame rate.
videoBitrate	int	Bitrate. The SDK encodes streams at the target video bitrate and will actively reduce the bitrate only if the network conditions are poor.
videoResolution	int	Resolution.
videoResolutionMode	int	Resolution mode.
appGroup	String	This parameter takes effect only for iOS and can be ignored for Android. It is the `Application Group Identifier` shared by the primary app and broadcast process.

> Note :
> For more information, please see TRTC SDK.

stopScreenCapture

This API is used to stop screen capturing.

```
Future<void> stopScreenCapture();
```

pauseScreenCapture

This API is used to pause screen capturing.

```
Future<void> pauseScreenCapture();
```

resumeScreenCapture

This API is used to resume screen capturing.

```
Future<void> resumeScreenCapture();
```

Management Object Acquisition APIs

getDeviceManager

This API is used to get the device management object TXDeviceManager.

```
getDeviceManager();
```

getBeautyManager

This API is used to get the beauty filter management object TXBeautyManager.

```
getBeautyManager();
```

You can do the following using `TXBeautyManager` :

- Set the beauty filter style and apply effects including skin brightening, rosy skin, eye enlarging, face slimming, chin slimming, chin lengthening/shortening, face shortening, nose narrowing, eye brightening, teeth whitening, eye bag removal, wrinkle removal, and smile line removal.
- Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose position, lip thickness, and face shape.
- Apply animated effects such as face widgets (materials).
- Add makeup effects.
- Recognize gestures.

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a meeting, which is generally used for chat.

```
Future sendRoomTextMsg(String message);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message

sendRoomCustomMsg

This API is used to send a custom text message.

```
Future sendRoomCustomMsg(String cmd, String message);
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message

`TRTCMeetingDelegate` Event Callbacks

Common Event Callback APIs

onError

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

The parameters are as detailed below:

Parameter	Type	Description
errCode	int	Error code.
errMsg	String	Error message

Parameter	Type	Description
extraInfo	String	Extended field. Certain error codes may carry extra information for troubleshooting.

onWarning

Callback for warning.

The parameters are as detailed below:

Parameter	Type	Description
warningCode	int	Warning code.
warningMsg	String	Warning message.
extraInfo	String	Extended field. Certain error codes may carry extra information for troubleshooting.

onKickedOffline

Callback for being kicked offline because another user logged in to the same account.

Meeting Room Event Callbacks

onRoomDestroy

Callback for meeting room termination. When the anchor leaves the room, all users in the room will receive this callback.

The parameters are as detailed below:

Parameter	Type	Description
roomId	String	Meeting room ID.

onNetworkQuality

Callback for network status.

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
localQuality	TRTCCloudDef.TRTCQuality	Upstream network quality.
remoteQuality	List<TRTCCloudDef.TRTCQuality>	Downstream network quality.

onUserVolumeUpdate

Call volume of a user.

The parameters are as detailed below:

Parameter	Type	Description
userVolumes	List	Volume of every speaking user in the room. Value range: 0-100.
totalVolume	int	Total volume of all remote users. Value range: 0-100.

Member Entry/Exit Event Callbacks

onEnterRoom

You joined the meeting.

The parameters are as detailed below:

Parameter	Type	Description
result	int	A value greater than 0 indicates the time used for joining the meeting, in ms. A value smaller than 0 indicates the error code for joining the meeting.

onLeaveRoom

You left the meeting.

The parameters are as detailed below:

Parameter	Type	Description
reason	int	Reason for leaving the meeting. 0: the user actively called `leaveMeeting` to leave the meeting; 1: the user was kicked out of the meeting by the server; 2: the meeting was dismissed.

onUserEnterRoom

A new member joined the meeting.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the new member who joins the meeting.

onUserLeaveRoom

A member left the meeting.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the member who leaves the meeting.

Member Audio/Video Event Callbacks

onUserAudioAvailable

A member turned the mic on/off.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
available	boolean	true: the mic is enabled; false: the mic is disabled.

onUserVideoAvailable

A member turned the camera on/off.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
available	boolean	true: the camera is enabled; false: the camera is disabled.

onUserSubStreamAvailable

A member enabled/disabled the substream image.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
available	boolean	true: the substream image is enabled; false: the substream image is disabled.

Message Event Callback APIs

onRecvRoomTextMsg

A text message was received.

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
sendId	String	Sender's user ID.
userAvatar	String	Sender's profile photo.
userName	String	Sender's nickname.

onRecvRoomCustomMsg

A custom message was received.

The parameters are as detailed below:

Parameter	Type	Description
command	String	Custom command word used to distinguish between different message types
message	String	Text message
sendId	String	Sender's user ID.
userAvatar	String	Sender's profile photo.

Parameter	Type	Description
userName	String	Sender's nickname.

Screen Sharing Event Callbacks

onScreenCaptureStarted

Screen sharing started.

onScreenCapturePaused

Screen sharing paused.

onScreenCaptureResumed

Screen sharing resumed.

onScreenCaptureStoped

Screen sharing stopped.

The parameters are as detailed below:

Parameter	Type	Description
reason	int	Reason for stop. 0: the user stopped proactively; 1: screen sharing stopped as the shared window was closed.

TUIRoom (Windows and macOS)

Last updated : 2022-03-01 12:23:20

`TUIRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- The anchor can create a room, and the member can enter the room ID to join the room.
- The participants can share their screens with each other.
- All users can send various text and custom messages.

`TUIRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Group Audio/Video Room (for Windows and macOS).

- TRTC SDK: the TRTC SDK is used as the low-latency video conferencing component.
- IM SDK: the IM SDK for **C++** is used to implement the chat room feature.

`TUIRoom` API Overview

Basic functions of `TUIRoomCore`

API	Description
GetInstance	Gets a singleton object.
DestroyInstance	Terminates a singleton object.
SetCallback	Sets event callback.

Room APIs

API	Description
login	Logs in.
logout	Logs out.
CreateRoom	Creates a room (called by anchor).
DestroyRoom	Terminates a room (called by anchor).
EnterRoom	Enters a room (called by participant).

API	Description
LeaveRoom	Exits a room (called by participant or anchor).
GetRoomInfo	Gets the room information.
GetRoomUsers	Gets the information of all users in the room.
GetUserInfo	Gets the information of a user.
TransferRoomMaster	Transfers the anchor permission (called by anchor).

Local audio/video operation APIs

API	Description
StartCameraPreview	Enables the preview of the local video.
StopCameraPreview	Stops local video capturing and preview.
UpdateCameraPreview	Updates the local video rendering window.
StartLocalAudio	Enables mic capturing.
StopLocalAudio	Stops mic capturing.
StartSystemAudioLoopback	Enables system audio capturing.
StopSystemAudioLoopback	Disables system audio capturing.
SetVideoMirror	Sets the mirroring preview mode of the local image.

Remote user APIs

API	Description
StartRemoteView	Subscribes to and plays back the remote video image of a specified member.
StopRemoteView	Unsubscribes from and stops the playback of a remote video image.
UpdateRemoteView	Updates the video rendering window of a remote user.

Chat message sending APIs

API	Description

API	Description
SendChatMessage	Sends a chat message.
SendCustomMessage	Sends a custom message.

Room control APIs

API	Description
MuteUserMicrophone	Enables/Disables the mic of a specified user.
MuteAllUsersMicrophone	Enables/Disables the mic of all users and syncs the status to room information.
MuteUserCamera	Enables/Disables the camera of a specified user.
MuteAllUsersCamera	Enables/Disables the camera of all users and syncs the status to room information.
MuteChatRoom	Mutes/Unmutes the chat room (called by anchor).
KickOffUser	Removes a specified user in the room (called by anchor).
StartCallingRoll	Starts calling roll by the anchor.
StopCallingRoll	Stops calling roll by the anchor.
ReplyCallingRoll	Replies to roll call by a member.
SendSpeechInvitation	Invites a member to speak by the anchor.
CancelSpeechInvitation	Cancels invitation to a member for speech by the anchor.
ReplySpeechInvitation	Accepts/Rejects the speech invitation from the anchor by a member.
SendSpeechApplication	Applies for speech by a member.
CancelSpeechApplication	Cancels speech application by a member.
ReplySpeechApplication	Approves/Rejects the speech application of a member by the anchor.
ForbidSpeechApplication	Forbids speech application by the anchor.
SendOffSpeaker	Stops the speech of a member by the anchor.

API	Description
SendOffAllSpeakers	Stops the speech of all members by the anchor.
ExitSpeechState	Stops speaking by a member and changes their role to audience.

Basic component API functions

API	Description
GetDeviceManager	Gets the local settings management object `ITXDeviceManager` .
GetScreenShareManager	Gets the screen sharing management object `IScreenShareManager` .

On-cloud recording API functions

API	Description
StartCloudRecord	Starts on-cloud recording.
StopCloudRecord	Stops on-cloud recording.

Beauty filter APIs

API	Description
SetBeautyStyle	Sets a beauty filter.

Settings APIs

API	Description
SetVideoQosPreference	Sets network bandwidth limit parameters.

SDK version acquisition APIs

API	Description
GetSDKVersion	Gets the SDK version.

`TUIRoomCoreCallback` API Overview

Callbacks for error events

API	Description
OnError	Callback for error.

Basic event callbacks

API	Description
OnLogin	Login.
OnLogout	Logout.
OnCreateRoom	Room creation.
OnDestroyRoom	Room dismissal.
OnEnterRoom	Room entry.
OnExitRoom	Room exit.
OnFirstVideoFrame	The first video frame.
OnUserVoiceVolume	Volume level.
OnRoomMasterChanged	Anchor change.

Remote user event callbacks

API	Description
OnRemoteUserEnter	A remote user entered the room.
OnRemoteUserLeave	A remote user exited the room.
OnRemoteUserCameraAvailable	Whether a remote user enabled the camera.
OnRemoteUserScreenAvailable	Whether a remote user enabled screen sharing.
OnRemoteUserAudioAvailable	Whether a remote user enabled mic.
OnRemoteUserEnterSpeechState	A remote user started speaking.
OnRemoteUserExitSpeechState	A remote user stopped speaking.

Message event callback APIs

API	Description
OnReceiveChatMessage	A text message was received.
OnReceiveCustomMessage	A custom message was received.

Room control event callbacks

API	Description
OnReceiveSpeechInvitation	A member received a speech invitation from the anchor.
OnReceiveInvitationCancelled	A member received a speech invitation cancellation from the anchor.
OnReceiveReplyToSpeechInvitation	The anchor received the acceptance to a speech invitation by a member.
OnReceiveSpeechApplication	The anchor received a speech application from a member.
OnSpeechApplicationCancelled	A member canceled a speech application.
OnReceiveReplyToSpeechApplication	The anchor approved a speech application.
OnSpeechApplicationForbidden	The anchor rejected a speech application.
OnOrderedToExitSpeechState	A member was asked to stop speaking.
OnCallingRollStarted	The anchor started a roll call.
OnCallingRollStopped	The anchor stopped a roll call.
OnMemberReplyCallingRoll	A member replied to roll call.
OnChatRoomMuted	The anchor muted/unmuted the room.
OnMicrophoneMuted	The anchor disabled the mic.
OnCameraMuted	The anchor disabled the camera.

Callback APIs for statistics on network quality and technical metrics

API	Description
OnStatistics	Statistics on technical metrics.

API	Description
OnNetworkQuality	Network quality.

Screen sharing event callbacks

API	Description
OnScreenCaptureStarted	Screen sharing started.
OnScreenCaptureStopped	Screen sharing stopped.

Video recording callbacks

API	Description
OnRecordError	Recording error.
OnRecordComplete	Recording completion.
OnRecordProgress	Recording progress.

Local device test callbacks

API	Description
OnTestSpeakerVolume	Speaker volume level.
OnTestMicrophoneVolume	Mic volume level.
OnAudioDeviceCaptureVolumeChanged	System capturing volume level adjustment.
OnAudioDevicePlayoutVolumeChanged	System playback volume level adjustment.

Basic Functions of `TUIRoomCore`

GetInstance

This API is used to get a TUIRoomCore singleton object.

```
static TUIRoomCore* GetInstance();
```

DestroyInstance

```
static void DestroyInstance();
```

SetCallback

This API is used to set the event callback of TUIRoomCore. You can use `TUIRoomCoreCallback` to get different status notifications of TUIRoomCore.

```
virtual void SetCallback(const TUIRoomCoreCallback* callback) = 0;
```

Login

This API is used to log in to the Tencent backend server.

```
virtual int Login(int sdk_appid, const std::string& user_id, const std::string& user_sig) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
sdk_appid	int	You can view `SDKAppID` in **Application Management** > **Application Info** of the TRTC console.
user_id	string	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
user_sig	string	Tencent Cloud's proprietary security signature. For more information on how to get it, please see UserSig.

Logout

This API is used to log out of the Tencent backend server.

```
virtual int Logout() = 0;
```

CreateRoom

This API is used to create a room (called by the anchor).

```
virtual int CreateRoom(const std::string& room_id, TUISpeechMode speech_mode) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
room_id	string	Room ID. You need to assign and manage the IDs in a centralized manner.
speech_mode	TUISpeechMode	Speech mode.

Generally, the anchor calls the APIs in the following steps:

1. The **anchor** calls `CreateRoom()` to create a room, the result of which is returned via `OnCreateRoom` .
2. The **anchor** calls `EnterRoom()` to enter the room.
3. The **anchor** calls `StartCameraPreview()` to enable camera capturing and preview.
4. The **anchor** calls `StartLocalAudio()` to enable the local mic.

DestroyRoom

This API is used to terminate a room (called by the anchor).

```
virtual int DestroyRoom() = 0;
```

EnterRoom

This API is used to enter a room (called by a participant).

```
virtual int EnterRoom(const std::string& room_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
room_id	string	Room ID.

Generally, a participant enters a room in the following steps:

1. The **participant** calls `EnterRoom` and passes in `room_id` to enter the room.
2. The **participant** calls `startCameraPreview()` to enable camera preview and calls `StartLocalAudio()` to enable mic capturing.
3. The **participant** receives the `OnRemoteUserCameraAvailable` event and calls `StartRemoteView()` to start playback.

LeaveRoom

This API is used to exit a room (called by a participant).

```
virtual int LeaveRoom() = 0;
```

GetRoomInfo

This API is used to get the room information.

```
virtual TUIRoomInfo GetRoomInfo() = 0;
```

GetRoomUsers

This API is used to get the information of all users in the room.

```
virtual std::vector GetRoomUsers() = 0;
```

GetUserInfo

This API is used to get the information of a user in the room.

```
virtual const TUIUserInfo* GetUserInfo(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.

SetSelfProfile

This API is used to set the user attributes.

```
virtual int SetSelfProfile(const std::string& user_name, const std::string& avatar_url) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_name	string	User name.
avatar_url	string	User profile photo URL.

TransferRoomMaster

This API is used to transfer a group to another user.

```
virtual int TransferRoomMaster(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.

Local Push APIs

StartCameraPreview

This API is used to start the preview of the local camera.

```
virtual int StartCameraPreview(const liteav::TXView& view) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
view	liteav::TXView	Window handle.

StopCameraPreview

This API is used to stop the preview of the local camera.

```
virtual int StopCameraPreview() = 0;
```

UpdateCameraPreview

This API is used to update the preview image of the local video.

```
virtual int UpdateCameraPreview(const liteav::TXView& view) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
view	liteav::TXView	Window handle.

StartLocalAudio

This API is used to enable the local audio device.

```
virtual int StartLocalAudio(const liteav::TRTCAudioQuality& quality) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
view	liteav::TXView	Window handle.

StopLocalAudio

This API is used to disable the local audio device.

```
virtual int StopLocalAudio() = 0;
```

StartSystemAudioLoopback

This API is used to enable system audio capturing.

```
virtual int StartSystemAudioLoopback() = 0;
```

StopSystemAudioLoopback

This API is used to disable system audio capturing.

```
virtual int StopSystemAudioLoopback() = 0;
```

SetVideoMirror

This API is used to set mirroring mode.

```
virtual int SetVideoMirror(bool mirror) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
mirror	bool	Whether to enable the mirroring mode.

Remote User APIs

StartRemoteView

This API is used to subscribe to a remote user's video stream.

```
virtual int StartRemoteView(const std::string& user_id, const liteav::TXView& view,
TUIStreamType type = TUIStreamType::kStreamTypeCamera) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	ID of the user whose video image is to be played back.
liteav::TXView	TXView	`view` control that carries the video image.
type	TUIStreamType	Stream type.

StopRemoteView

This API is used to unsubscribe from and stop the playback of a remote video image.

```
virtual int StopRemoteView(const std::string& user_id,
TUIStreamType type = TUIStreamType::kStreamTypeCamera) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	ID of the user whose video image is to be stopped.
type	TUIStreamType	Stream type.

UpdateRemoteView

This API is used to updates the rendering window of a remote video.

```
virtual int UpdateRemoteView(const std::string& user_id, TUIStreamType type, liteav::TXView& vie
w) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
user_id	string	User ID.
type	TUIStreamType	Stream type.
view	liteav::TXView	Rendering window handle.

Message Sending APIs

SendChatMessage

This API is used to send text messages.

```
virtual int SendChatMessage(const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
message	string	Message content.

SendCustomMessage

Sending Custom Messages

```
virtual int SendCustomMessage(const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
message	string	Message content.

Room Control APIs

MuteUserMicrophone

This API is used to enable/disable the mic of the specified user.

```
virtual int MuteUserMicrophone(const std::string& user_id, bool mute, Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
mute	bool	Whether to disable.
callback	Callback	API callback.

MuteAllUsersMicrophone

This API is used to enable/disable the mic of all users.

```
virtual int MuteAllUsersMicrophone(bool mute) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
mute	bool	Whether to disable.

MuteUserCamera

This API is used to enable/disable the camera of the specified user.

```
virtual int MuteUserCamera(const std::string& user_id, bool mute, Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
mute	bool	Whether to disable.
callback	Callback	API callback.

MuteAllUsersCamera

This API is used to enable/disable the camera of all users.

```
virtual int MuteAllUsersCamera(bool mute) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
mute	bool	Whether to disable.

MuteChatRoom

This API is used to mute/unmute the chat room.

```
virtual int MuteChatRoom(bool mute) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
mute	bool	Whether to disable.

KickOffUser

This API is used by the anchor to remove a member.

```
virtual int KickOffUser(const std::string& user_id, Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
callback	Callback	API callback.

StartCallingRoll

This API is used by the anchor to start roll call.

```
virtual int StartCallingRoll() = 0;
```

StopCallingRoll

This API is used by the anchor to stop roll call.

```
virtual int StopCallingRoll() = 0;
```

ReplyCallingRoll

This API is used by a member to reply to roll call.

```
virtual int ReplyCallingRoll(Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	Callback	API callback.

SendSpeechInvitation

This API is used by the anchor to invite a member to speak.

```
virtual int SendSpeechInvitation(const std::string& user_id, Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
callback	Callback	API callback.

CancelSpeechInvitation

This API is used by the anchor to cancel the speech invitation to a member.

```
virtual int CancelSpeechInvitation(const std::string& user_id, Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
callback	Callback	API callback.

ReplySpeechInvitation

This API is used by a member to accept/reject the speech invitation from the anchor.

```
virtual int ReplySpeechInvitation(bool agree, Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
agree	bool	Whether to approve.
callback	Callback	API callback.

SendSpeechApplication

This API is used by a member to apply to speak.

```
virtual int SendSpeechApplication(Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	Callback	API callback.

CancelSpeechApplication

This API is used by a member to cancel the speech application.

```
virtual int CancelSpeechApplication(Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	Callback	API callback.

ReplySpeechApplication

This API is used by the anchor to approve/reject the speech application of a member.

```
virtual int ReplySpeechApplication(const std::string& user_id, bool agree, Callback callback) = 0
;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
callback	Callback	API callback.

ForbidSpeechApplication

This API is used by the anchor to forbid speech application.

```
virtual int ForbidSpeechApplication(bool forbid) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
forbid	bool	Whether to forbid.

SendOffSpeaker

This API is used by the anchor to stop the speech of the specified member.

```
virtual int SendOffSpeaker(const std::string& user_id, Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
callback	Callback	API callback.

SendOffAllSpeakers

This API is used by the anchor to stop the speech of all members.

```
virtual int SendOffAllSpeakers(Callback callback) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
callback	Callback	API callback.

ExitSpeechState

This API is used for a member to stop speaking and change their role to audience.

```
virtual int ExitSpeechState() = 0;
```

Basic Component APIs

GetDeviceManager

This API is used to get the device management object pointer.

```
virtual liteav::ITXDeviceManager* GetDeviceManager() = 0;
```

GetScreenShareManager

This API is used to get the screen sharing management object pointer.

```
virtual IScreenShareManager* GetScreenShareManager() = 0;
```

On-Cloud Recording APIs

StartCloudRecord

This API is used to start on-cloud recording.

```
virtual int StartCloudRecord() = 0;
```

StopCloudRecord

This API is used to stop on-cloud recording.

```
virtual int StopCloudRecord() = 0;
```

Beauty Filter APIs

SetBeautyStyle

This API is used to set effect levels of beauty, brightening, and rosy skin filters.

```
virtual int SetBeautyStyle(liteav::TRTCBeautyStyle style, uint32_t beauty_level,
uint32_t whiteness_level, uint32_t ruddiness_level) = 0;
```

You can do the following using `TXBeautyManager` :

- Set the beauty filter style to smooth or natural. The smooth style features more obvious skin smoothing effect.
- Set the strength of the beauty filter. Value range: 0-9. `0` indicates that the filter is disabled. The larger the value, the more obvious the effect.
- Set the strength of the skin brightening filter. Value range: 0-9. `0` indicates that the filter is disabled. The larger the value, the more obvious the effect.

The parameters are as detailed below:

Parameter	Type	Description
style	liteav::TRTCBeautyStyle	Beauty filter style.
beauty_level	uint32_t	strength of the beauty filter.
whiteness_level	uint32_t	Strength of the skin brightening filter.
ruddiness_level	uint32_t	Strength of the rosy skin filter.

Settings APIs

SetVideoQosPreference

set QoS parameters

```
virtual int SetVideoQosPreference(TUIVideoQosPreference preference) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
preference	TUIVideoQosPreference	Network bandwidth limit policy.

SDK Version Acquisition APIs

GetSDKVersion

This API is used to get SDK version information.

```
virtual const char* GetSDKVersion() = 0;
```

Error Event Callbacks

OnError

```
void OnError(int code, const std::string& message);
```

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	string	Error message.

Basic Event Callbacks

OnLogin

```
virtual void OnLogin(int code, const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	string	Login message or error message of login failure.

OnLogout

```
virtual void OnLogout(int code, const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	string	Error message.

OnCreateRoom

Room creation.

```
virtual void OnCreateRoom(int code, const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	string	Error message.

OnDestroyRoom

Room dismissal.

```
virtual void OnDestroyRoom(int code, const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	string	Error message.

OnEnterRoom

Room entry.

```
virtual void OnEnterRoom(int code, const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	string	Error message.

OnExitRoom

Room exit.

```
virtual void OnExitRoom(TUIExitRoomType type, const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
type	TUIExitRoomType	Room exit type.
message	string	Error message.

OnFirstVideoFrame

Rendering the first frame of the local video or a remote user started.

```
virtual void OnFirstVideoFrame(const std::string& user_id, const TUIStreamType stream_type) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
stream_type	TUIStreamType	Stream type.

OnUserVoiceVolume

User volume level.

```
virtual void OnUserVoiceVolume(const std::string& user_id, int volume)
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
volume	int	User volume level. Value range: 0–100.

OnRoomMasterChanged

Anchor change.

```
virtual void OnRoomMasterChanged(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
user_id	string	User ID.

Remote User Callbacks

OnRemoteUserEnter

A remote user entered the room.

```
virtual void OnRemoteUserEnter(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.

OnRemoteUserLeave

A remote user exited the room.

```
virtual void OnRemoteUserLeave(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.

OnRemoteUserCameraAvailable

Whether a remote user enabled the camera.

```
virtual void OnRemoteUserCameraAvailable(const std::string& user_id, bool available) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
available	bool	true: enabled; false: disabled.

OnRemoteUserScreenAvailable

Whether a remote user enabled screen sharing.

```
virtual void OnRemoteUserScreenAvailable(const std::string& user_id, bool available) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
available	bool	true: enabled; false: disabled.

OnRemoteUserAudioAvailable

Whether a remote user enabled mic.

```
virtual void OnRemoteUserAudioAvailable(const std::string& user_id, bool available) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
available	bool	true: enabled; false: disabled.

OnRemoteUserEnterSpeechState

A remote user started speaking.

```
virtual void OnRemoteUserEnterSpeechState(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.

OnRemoteUserExitSpeechState

A remote user stopped speaking.

```
virtual void OnRemoteUserExitSpeechState(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.

Chat Room Message Event Callbacks

OnReceiveChatMessage

Callback for receiving a text message.

```
virtual void OnReceiveChatMessage(const std::string& user_id, const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
message	string	Text message.

OnReceiveCustomMessage

Callback for receiving a custom message.

```
virtual void OnReceiveCustomMessage(const std::string& user_id, const std::string& message) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
message	string	Custom message.

Room Control Message Callbacks

OnReceiveSpeechInvitation

A user received a speech invitation from the anchor.

```
virtual void OnReceiveSpeechInvitation() = 0;
```

OnReceiveInvitationCancelled

A user received a speech invitation cancellation from the anchor.

```
virtual void OnReceiveInvitationCancelled() = 0;
```

OnReceiveReplyToSpeechInvitation

The anchor received the acceptance to a speech invitation by a member.

```
virtual void OnReceiveReplyToSpeechInvitation(const std::string& user_id, bool agree) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.
agree	bool	Whether the invitation was accepted.

OnReceiveSpeechApplication

The anchor received a speech application from a member.

```
virtual void OnReceiveSpeechApplication(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.

OnSpeechApplicationCancelled

A user canceled a speech application.

```
virtual void OnSpeechApplicationCancelled(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
user_id	string	User ID.

OnReceiveReplyToSpeechApplication

The anchor approved a speech application.

```
virtual void OnReceiveReplyToSpeechApplication(bool agree) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
agree	bool	Whether to approve.

OnSpeechApplicationForbidden

The anchor forbidden a speech application.

```
virtual void OnSpeechApplicationForbidden(bool forbidden) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
forbidden	bool	Whether to forbid.

OnOrderedToExitSpeechState

A member was asked to stop speaking.

```
virtual void OnOrderedToExitSpeechState() = 0;
```

OnCallingRollStarted

The anchor started a roll call.

```
virtual void OnCallingRollStarted() = 0;
```

OnCallingRollStopped

The anchor stopped a roll call.

```
virtual void OnCallingRollStopped() = 0;
```

OnMemberReplyCallingRoll

A member replied to roll call.

```
virtual void OnMemberReplyCallingRoll(const std::string& user_id) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
user_id	string	User ID.

OnChatRoomMuted

The anchor muted/unmuted the room.

```
virtual void OnChatRoomMuted(bool muted) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
muted	bool	Whether to disable.

OnMicrophoneMuted

The anchor disabled the mic.

```
virtual void OnMicrophoneMuted(bool muted) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
muted	bool	Whether to disable.

OnCameraMuted

The anchor disabled the camera.

```
virtual void OnCameraMuted(bool muted) = 0;
```

The parameters are as detailed below:

Parameter	Type	Description
muted	bool	Whether to disable.

Statistics Collection and Quality Callbacks

OnStatistics

Callback of technical metric statistics.

```
virtual void OnStatistics(const liteav::TRTCStatistics& statis) {}
```

The parameters are as detailed below:

Parameter	Type	Description
statis	liteav::TRTCStatistics	Statistics.

OnNetworkQuality

Network quality.

```
virtual void OnNetworkQuality(const liteav::TRTCQualityInfo& local_quality, liteav::TRTCQualityInfo* remote_quality,
uint32_t remote_quality_count) {}
```

The parameters are as detailed below:

Parameter	Type	Description
local_quality	liteav::TRTCQualityInfo	Local user quality information.
remote_quality	liteav::TRTCQualityInfo*	Pointer to the remote user quality information.
remote_quality_count	uint32_t	Number of remote users.

Screen Sharing Event Callbacks

OnScreenCaptureStarted

Screen sharing started.

```
virtual void OnScreenCaptureStarted() {}
```

OnScreenCaptureStopped

Screen sharing stopped.

```
void OnScreenCaptureStopped(int reason) {}
```

The parameters are as detailed below:

Parameter	Type	Description
reason	int	Reason for stop. 0: the user stopped proactively; 1: stopped due to preemption by another application.

Video Recording Callbacks

OnRecordError

Recording error.

```
virtual void OnRecordError(TXLiteAVLocalRecordError error, const std::string& messgae) {}
```

The parameters are as detailed below:

Parameter	Type	Description
error	TXLiteAVLocalRecordError	Error message.
messgae	string	Error description.

OnRecordComplete

Recording completion.

```
virtual void OnRecordComplete(const std::string& path) {}
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
path	string	Error description.

OnRecordProgress

Recording progress.

```
virtual void OnRecordProgress(int duration, int file_size) {}
```

The parameters are as detailed below:

Parameter	Type	Description
duration	int	File duration.
file_size	int	File size.

Local Device Test Callbacks

OnTestSpeakerVolume

Speaker volume level.

```
virtual void OnTestSpeakerVolume(uint32_t volume) {}
```

The parameters are as detailed below:

Parameter	Type	Description
volume	uint32_t	Volume level.

OnTestMicrophoneVolume

Mic volume level.

```
virtual void OnTestMicrophoneVolume(uint32_t volume) {}
```

The parameters are as detailed below:

Parameter	Type	Description
volume	uint32_t	Volume level.

OnAudioDeviceCaptureVolumeChanged

System capturing volume level adjustment.

```
virtual void OnAudioDeviceCaptureVolumeChanged(uint32_t volume, bool muted) {}
```

The parameters are as detailed below:

Parameter	Type	Description
volume	uint32_t	Volume level.
muted	bool	Whether to disable.

OnAudioDevicePlayoutVolumeChanged

System playback volume level adjustment.

```
virtual void OnAudioDevicePlayoutVolumeChanged(uint32_t volume, bool muted) {}
```

The parameters are as detailed below:

Parameter	Type	Description
volume	uint32_t	Volume level.
muted	bool	Whether to disable.

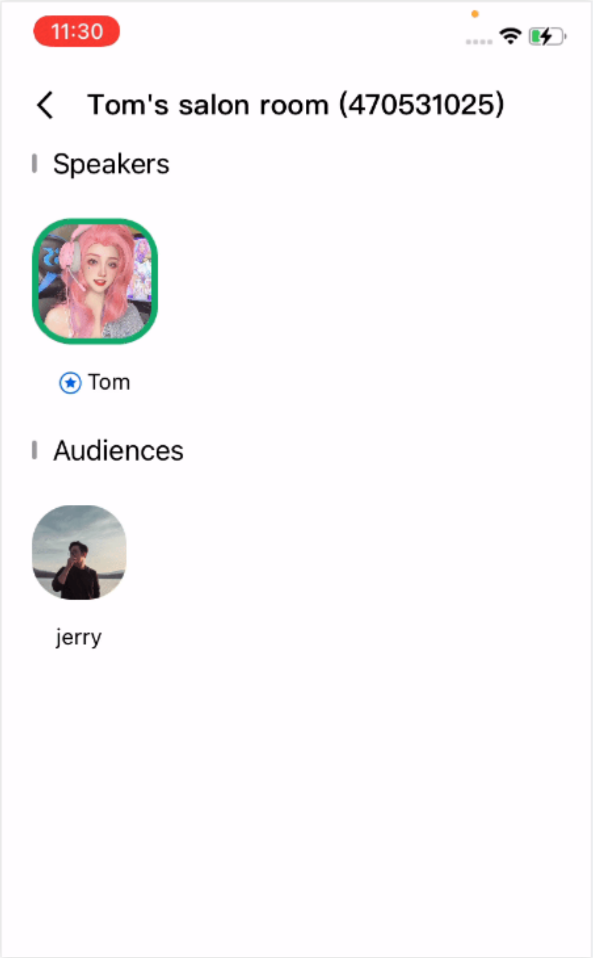
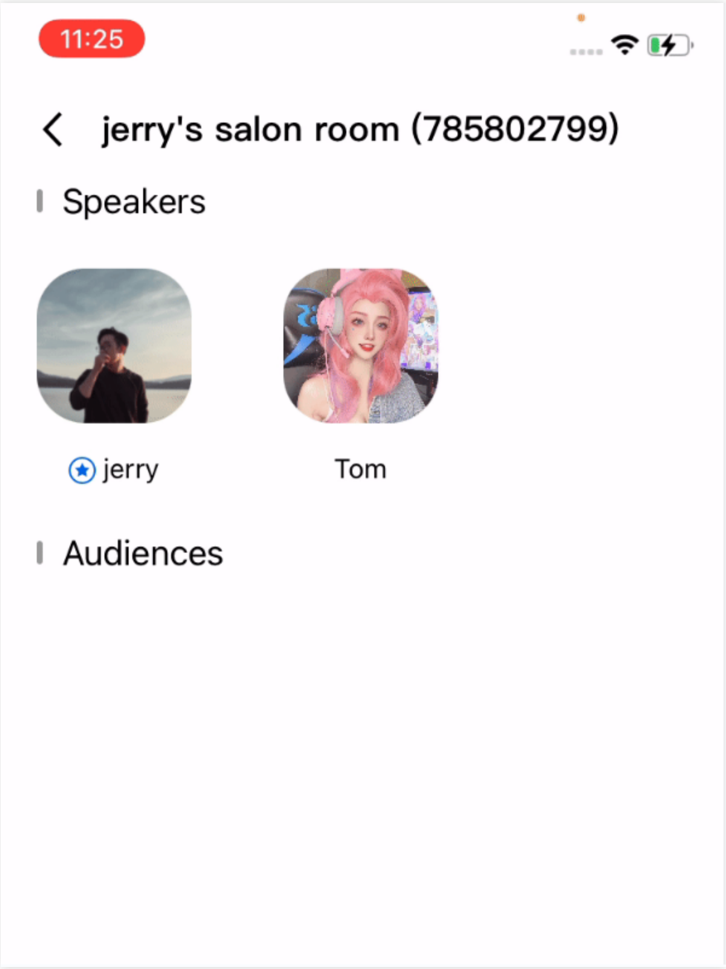
Chat Salon

Chat Salon (Android)

Last updated : 2022-03-18 23:02:38

Demo UI

You can download and install the demo app we provide to try out TRTC features in the chat salon scenario, including audio chat, mic on/off, low-latency audio interaction, etc.

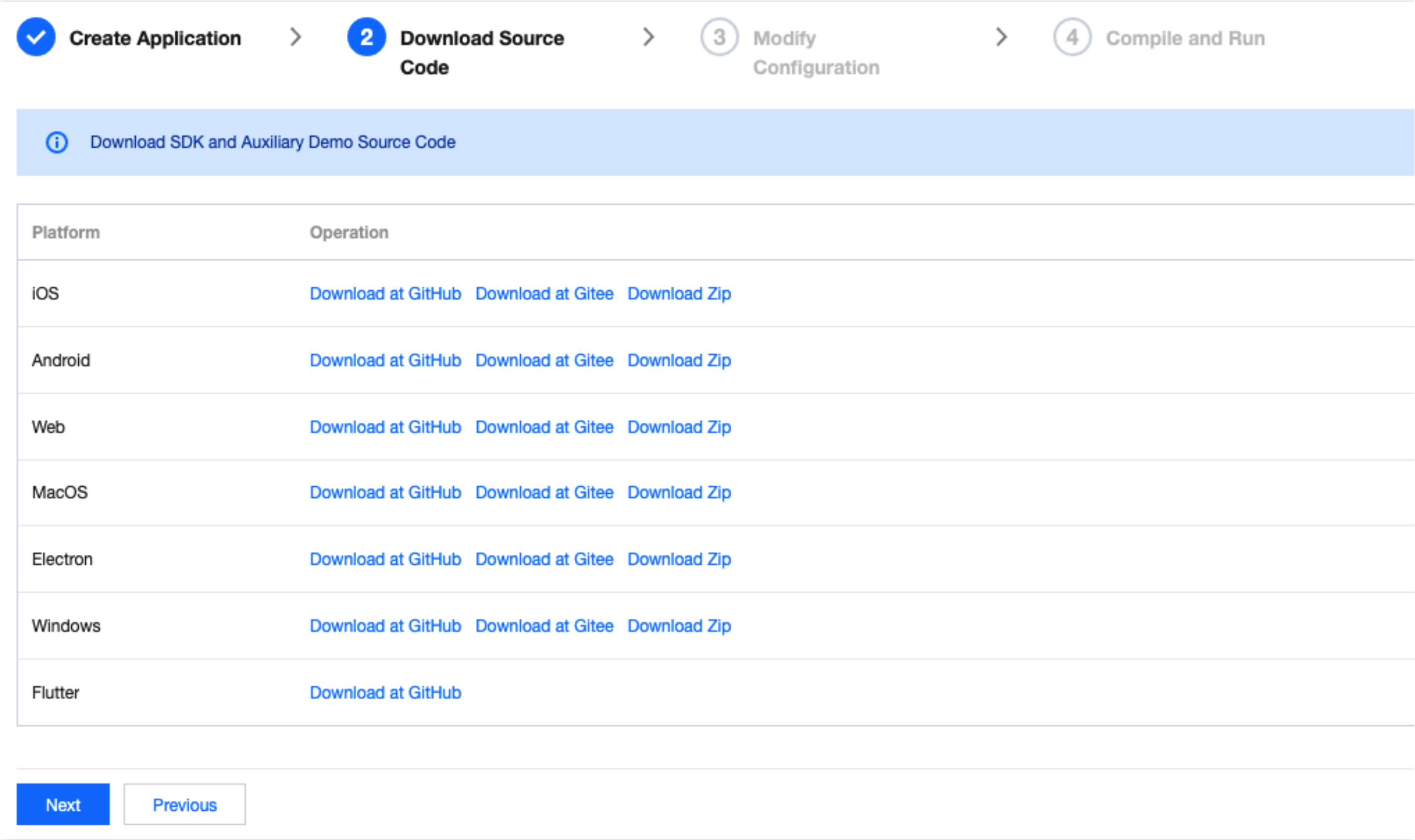
Room Owner	Listener
 11:30 Tom's salon room (470531025) Speakers Tom Audiences jerry	 11:25 jerry's salon room (785802799) Speakers jerry Tom Audiences

To quickly enable the chat salon feature, you can modify the demo app we provide and adapt it to your needs. You may also use the `TUIChatSalon` component and customize your own UI.

Using the App’s UI

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TRTCChatSalon` and click **Create**.
3. Click **Next**.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Note :
This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the application source code

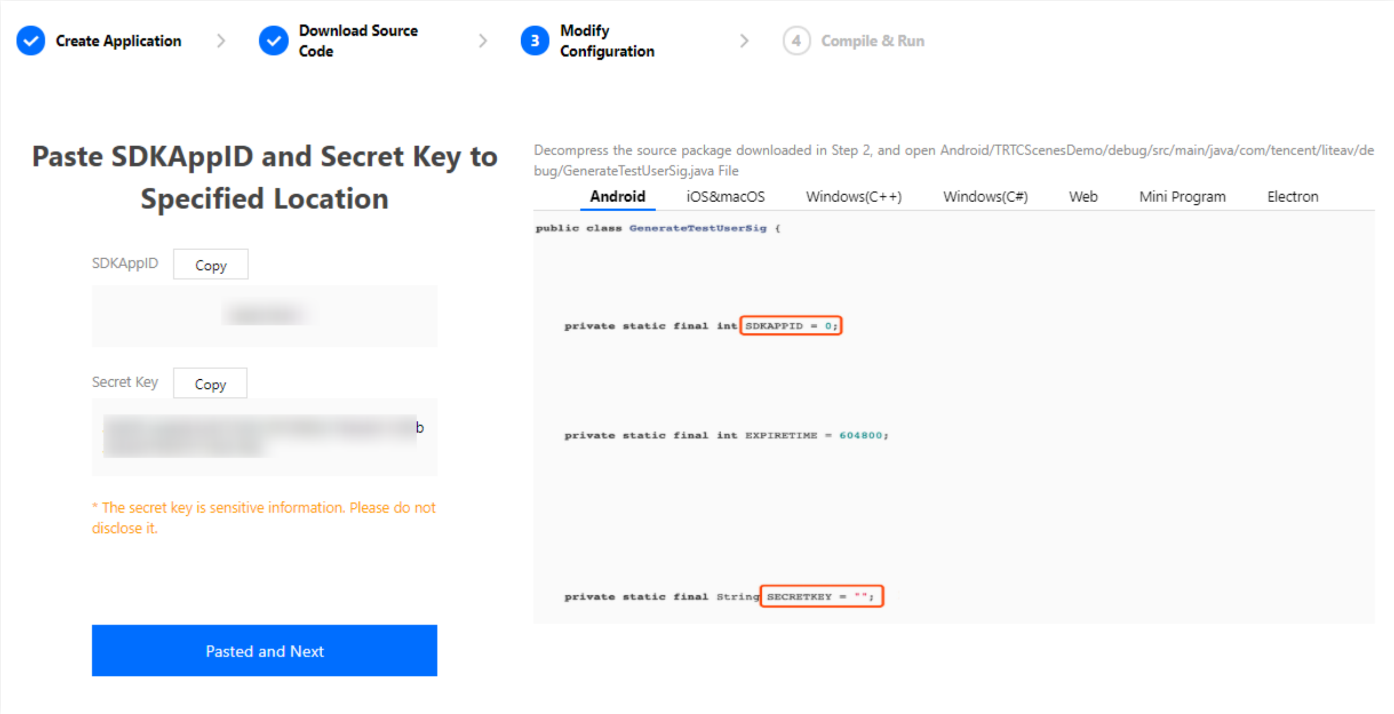
Click TUIChatSalon to clone or download the source code.

Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `Android/Debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java`.

3. Set parameters in `GenerateTestUserSig.java`:
   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

Note :

- The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
- The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your

application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUIChatSalon` with Android Studio (version 3.5 or above) and click **Run**.

Step 5. Modify the app's source code

The `Source` folder in the source code contains two subfolders: `ui` and `model` . The `ui` subfolder contains UI code. The table below lists the files (folders) and the UI views they represent. You can refer to it when making UI changes.

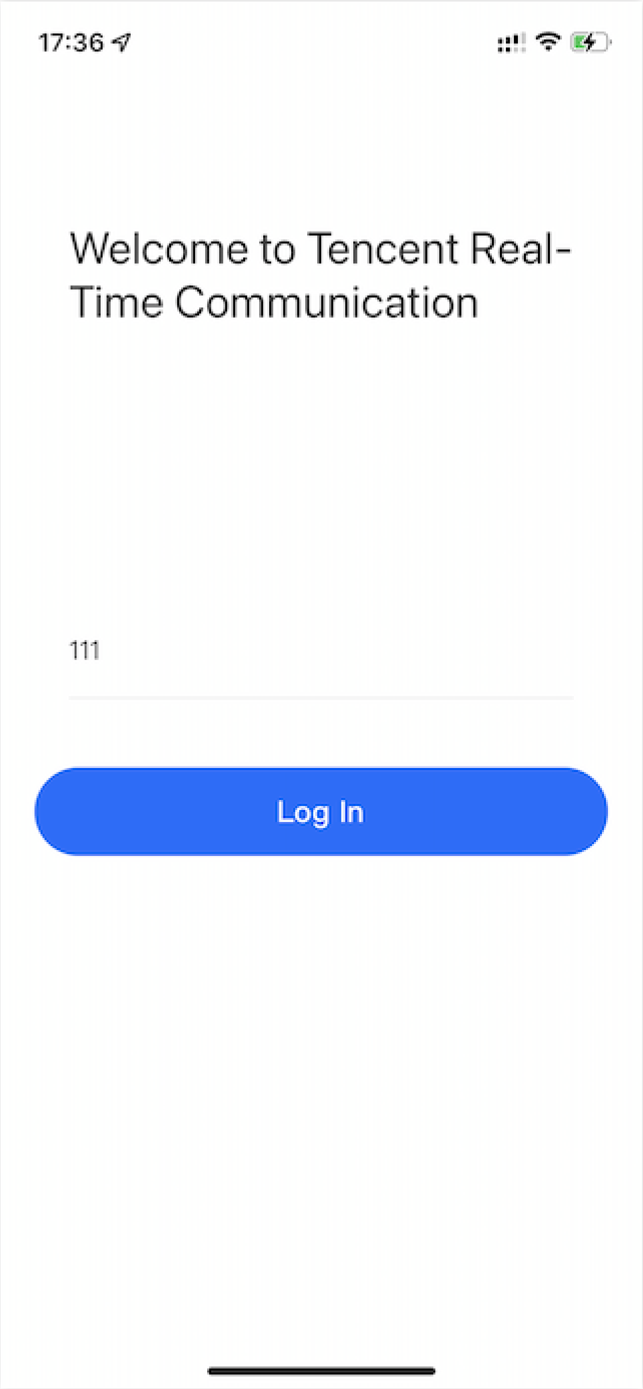
File or Folder	Description
base	Basic classes used by the UI
list	Room creation page.
room	The main room views for room owner and listener
widget	Common controls

Tryout

Note :
You need at least two devices to try out the application.

User A

1. Enter a username (**which must be unique**) and log in.

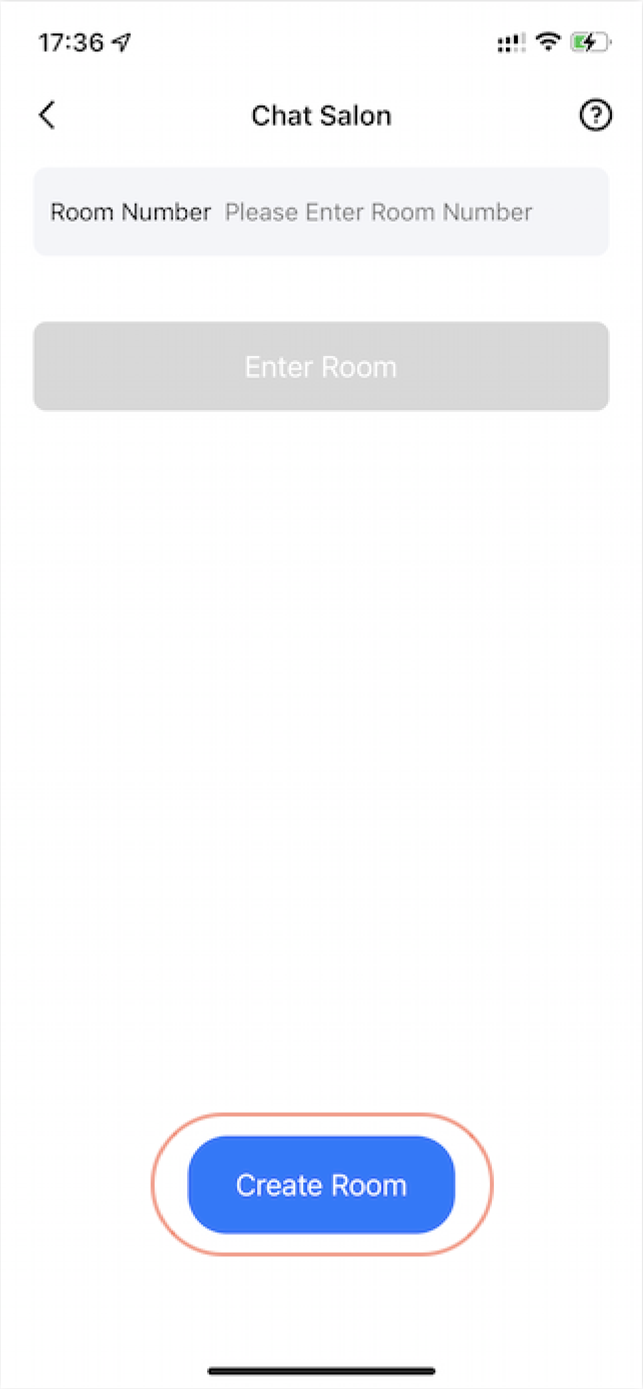


17:36

Welcome to Tencent Real-Time Communication

111

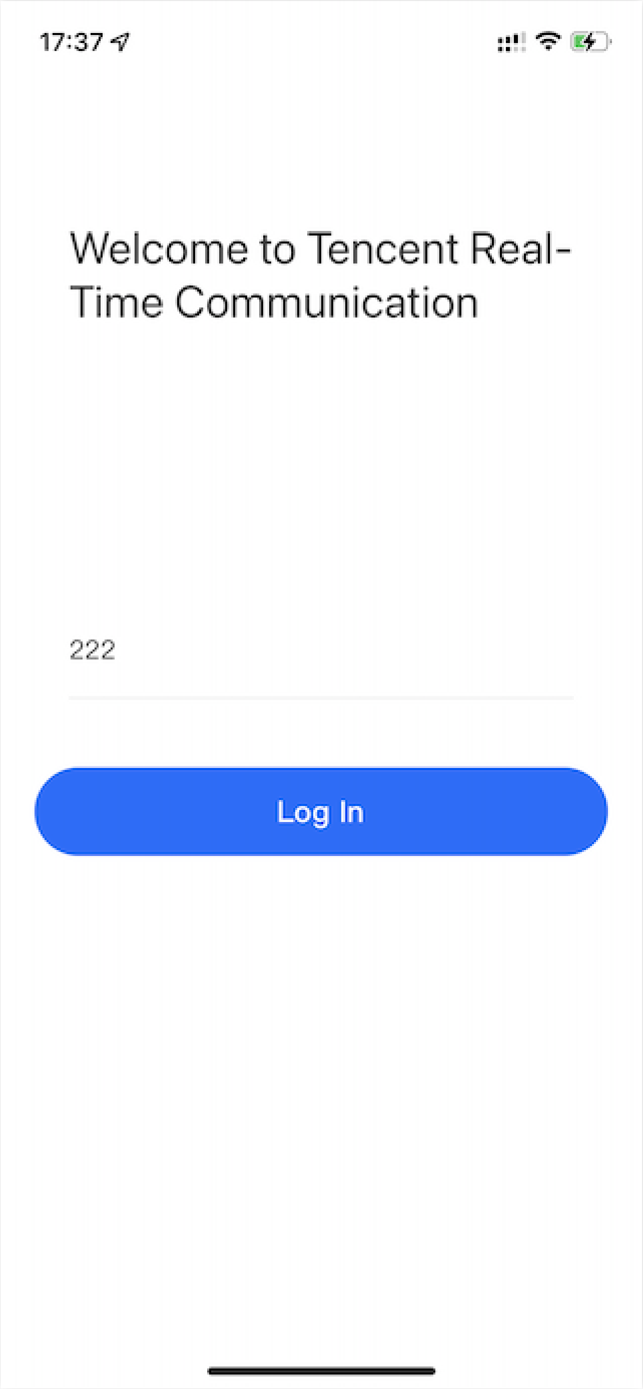
2. Tap **Create Room**.



3. Type a subject for the room and tap **Let’s go**.

User B

1. Enter a username (**which must be unique**) and log in.

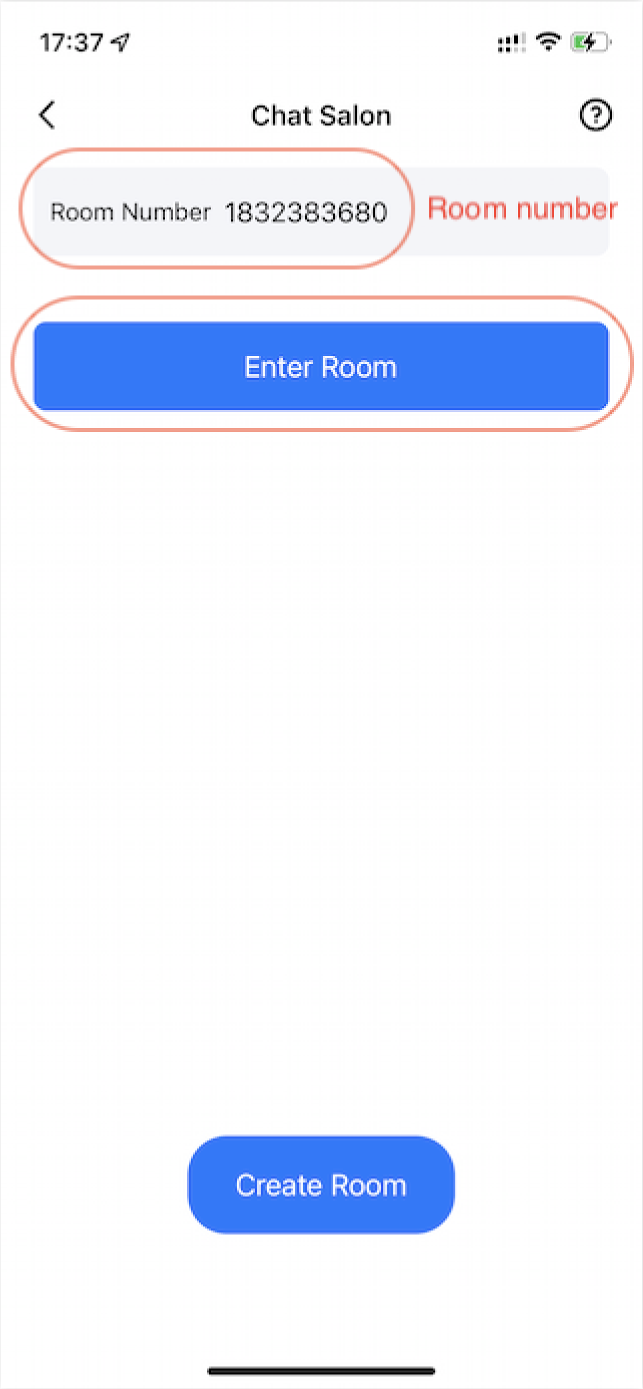


17:37

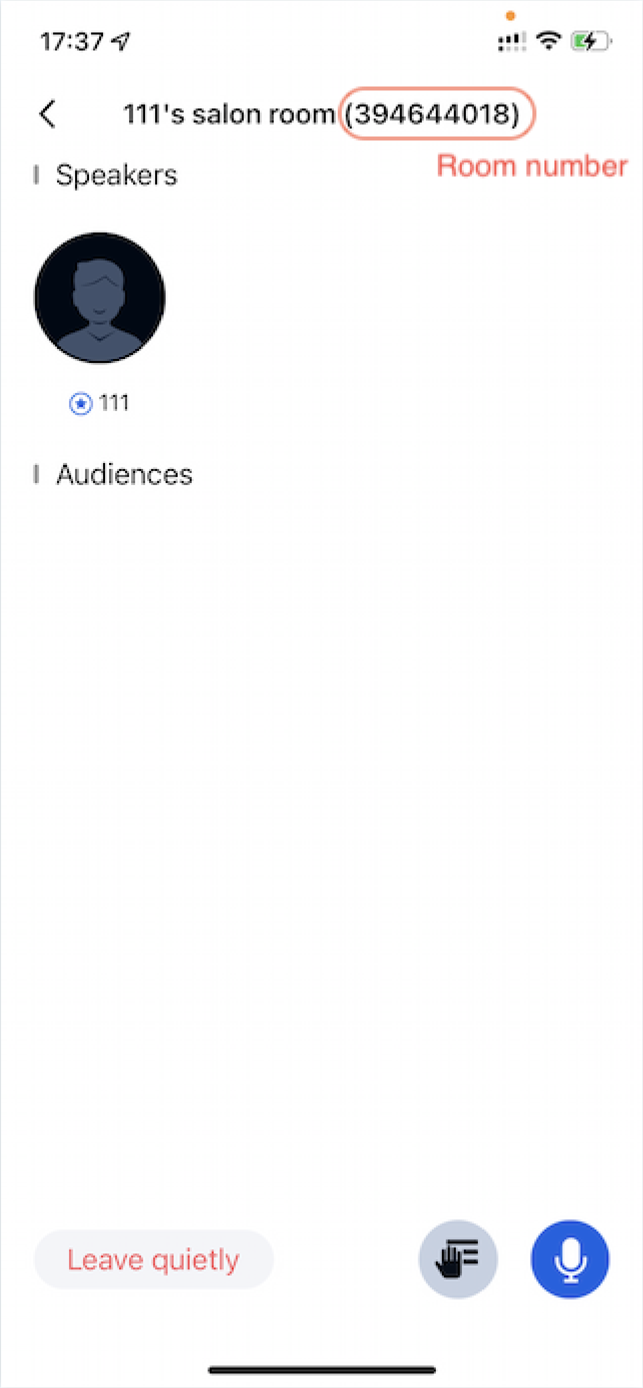
Welcome to Tencent Real-Time Communication

222

2. Enter the number of the room created by user A and tap **Enter Room**.

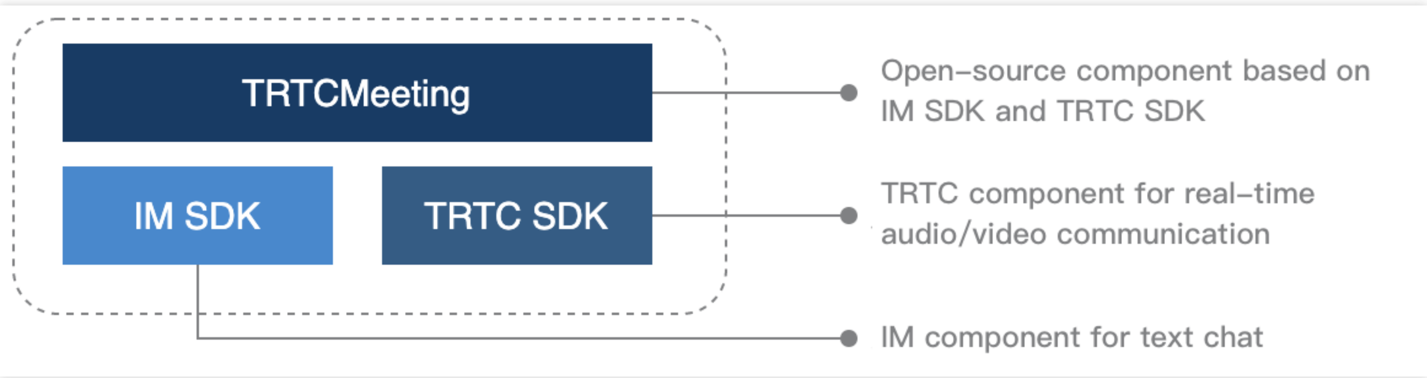


Note：
You can find the room number at the top of user A’s room view.



Customizing Your Own UI

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `model` subfolder contains the reusable open-source component `TRTCChatSalon`. You can find the component’s APIs in `TRTCChatSalon.java` and use them to customize your own UI.


TRTCMeeting
IM SDK
TRTC SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for text chat

Step 1. Integrate the SDKs

The chat salon component's `Source` depends on the TRTC SDK and IM SDK. Follow the steps below to integrate them into your project.

Method 1: adding dependencies via Maven

1. Add the TRTC SDK and IM SDK dependencies to `dependencies` .

```
dependencies {
complie ”com.tencent.liteav:LiteAVSDK_TRTC:latest.release”
complie ’com.tencent.imsdk:imsdk:latest.release’
compile ’com.google.code.gson:gson:2.3.1’
}
```

2. In `defaultConfig` , specify the CPU architecture to be used by your application.

```
defaultConfig {
ndk {
abiFilters ”armeabi-v7a”
}
}
```

3. Click **Sync Now** to automatically download the SDKs and integrate them into your project.

Method 2: adding dependencies through local AAR files

If your access to the Maven repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration document
IM SDK	Download	Integration document

Step 2. Configure permission requests and obfuscation rules

Configure permission requests for your app in `AndroidManifest.xml` . The SDKs need the following permissions (on Android 6.0 and above, the read storage permissions must be requested at runtime):

```
<uses-permission android:name="android.permission.INTERNET">
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE">
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE">
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE">
<uses-permission android:name="android.permission.RECORD_AUDIO">
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS">
<uses-permission android:name="android.permission.BLUETOOTH">
<uses-permission android:name="android.permission.READ_PHONE_STATE">
```

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

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

Step 3. Import the `TRTCChatSalon` component

Copy all the files in the directory below to your project:

```
Source/src/main/java/com/tencent/liteav/trtcchatsalon/model
```

Step 4. Create an instance and log in

1. Call the `sharedInstance` API to create an instance of the `TRTCChatSalon` component.
2. Call the `setDelegate` API to register event callbacks of the component.
3. Call the `login` API to log in to the component, and set the key parameters as described below.

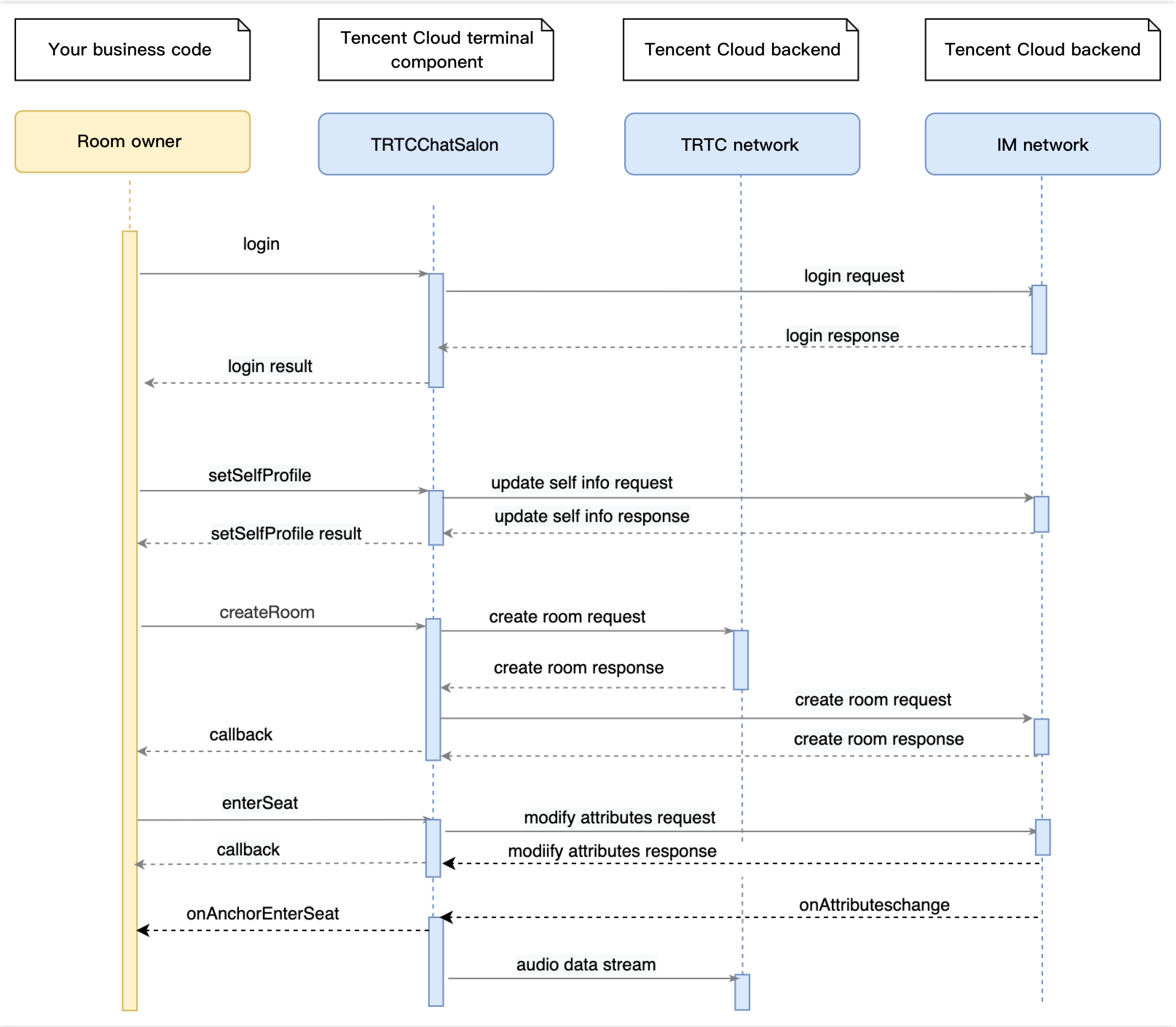
Parameter	Description
SDKAppID	You can view `SDKAppID` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.

userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.
callback	Login callback. The code is `0` if login is successful.

```
TRTCChatSalon mTRTCChatSalon = TRTCChatSalon.sharedInstance(this);
mTRTCChatSalon.setDelegate(this);
mTRTCChatSalon.login(SDKAPPID, userId, userSig, new TRTCChatSalonCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// Logged in
}
}
});
```

Step 5. Create a room and become a speaker

1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. Call `createRoom` to create a chat salon, passing in room-related parameters such as room ID, whether your consent is required for listeners to speak, and the room type.
3. You will receive an `onAnchorEnterSeat` notification that someone becomes a speaker, and mic capturing will be enabled automatically.



```
// 1. Set your nickname and profile photo.
mTRTCChatSalon.setSelfProfile("my_name", "my_face_url", null);

// 2. Call `createRoom` to create a room
final TRTCChatSalonDef.RoomParam roomParam = new TRTCChatSalonDef.RoomParam();
roomParam.roomName = "Room name";
roomParam.needRequest = true; // Whether your consent is required for listeners to speak
roomParam.coverUrl = "URL of room cover image";
mTRTCChatSalon.createRoom(mRoomId, roomParam, new TRTCChatSalonCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// 3. Become a speaker
mTRTCChatSalon.enterSeat(new TRTCChatSalonCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
}
}
});
}
}
});

// 4. You receive an `onAnchorEnterSeat` notification after becoming a speaker
@Override
public void onAnchorEnterSeat(TRTCChatSalonDef.UserInfo userInfo) {
}
```

Step 6. Enter a room as a listener

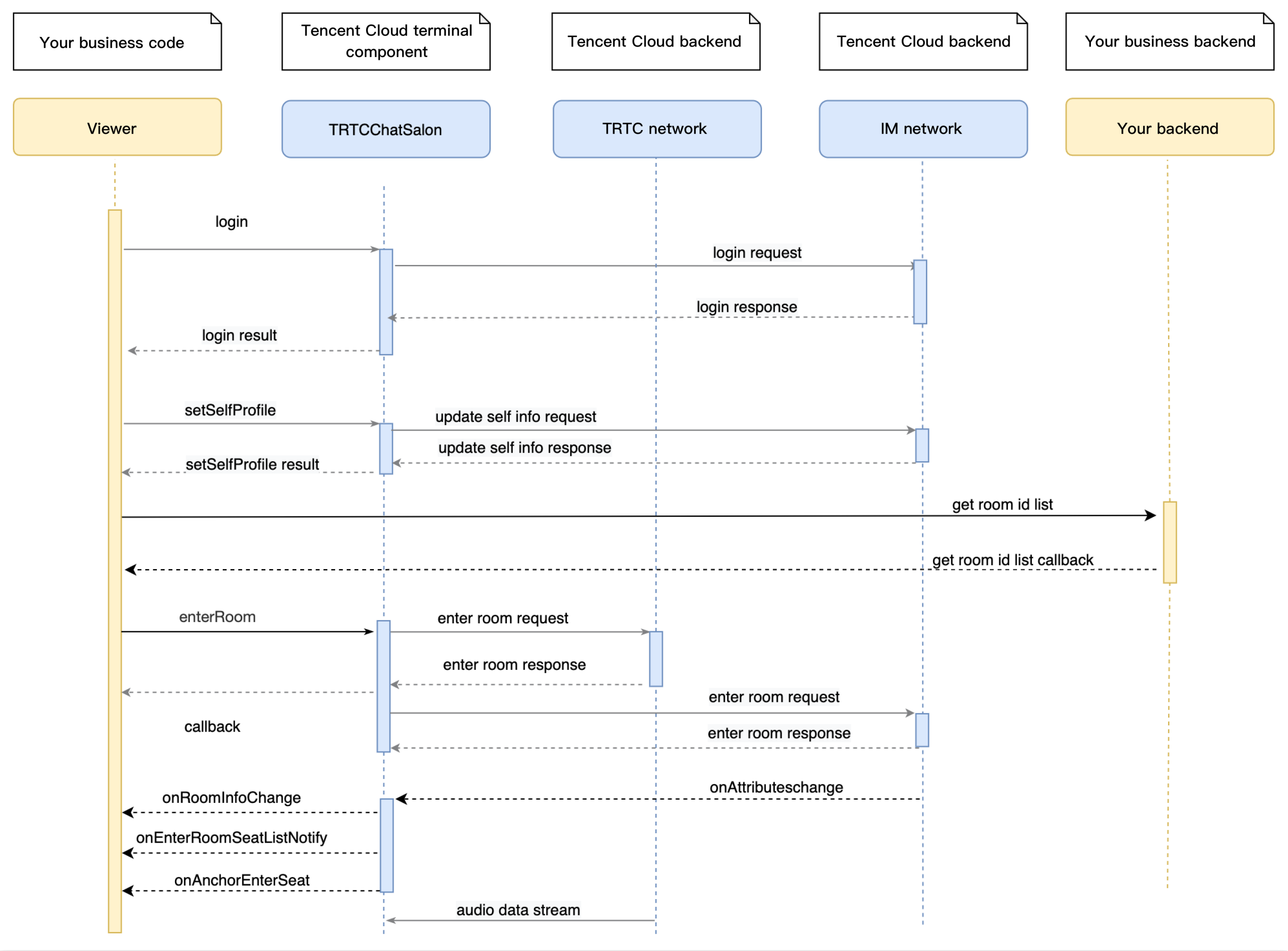
1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. Get the latest chat salon room list from the backend.

> Note :
> The chat salon list in the demo app is for demonstration only. The business logic of the chat salon list varies significantly. Tencent Cloud does not provide list management services for the time being. Please manage the list by yourself.

3. Call `getRoomInfoList` to get short descriptions of the rooms, which are provided by room owners when they call `createRoom` .

> Note :
> If your chat salon list already contains enough room information, you can skip the step of calling `getRoomInfoList` .

4. Select a chat salon, and call `enterRoom` with the room ID passed in to enter.
5. After entering the room, you will receive an `onRoomInfoChange` notification about room change from the component. Record the room information, including room name, whether the room owner’s consent is required for listeners to speak, etc., and update it to the UI.
6. You will also receive an `onAnchorEnterSeat` notification that someone becomes a speaker.



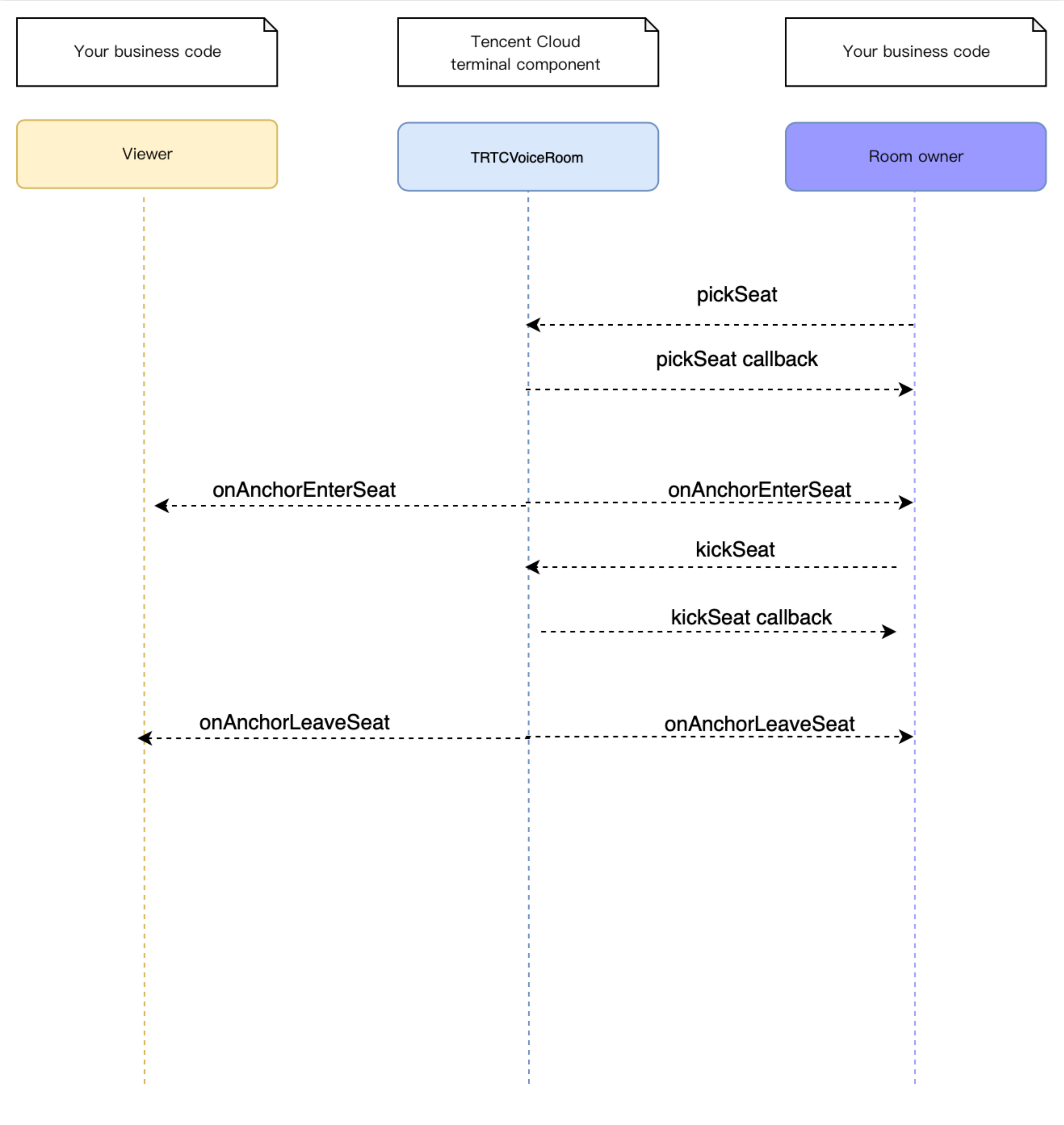
```
// 1. Set your nickname and profile photo.
mTRTCChatSalon.setSelfProfile("my_name", "my_face_url", null);

// 2. Get the room list from the backend. Suppose it is `roomList`.
List<integer> roomList = GetRoomList();

// 3. Call `getRoomInfoList` to get details of the rooms.
mTRTCChatSalon.getRoomInfoList(roomList, new TRTCChatSalonCallback.RoomInfoCallback() {

@Override
public void onCallback(int code, String msg, List<trtcchatsalondef.roominfo> list) {
if (code == 0) {
// Refresh the room list on your UI
}
}

});
```

```
// 4. Pass in `roomId` to enter a room.
mTRTCChatSalon.enterRoom(roomId, new TRTCChatSalonCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// Entered room successfully
}
}
});

// 5. After successful room entry, you receive an `onRoomInfoChange` notification.
@Override
public void onRoomInfoChange(TRTCChatSalonDef.RoomInfo roomInfo) {
mNeedRequest = roomInfo.needRequest;
mRoomName = roomInfo.roomName;
// The UI can display the title and other information
}

// 6. You receive an `onAnchorEnterSeat` notification.
@Override
public void onAnchorEnterSeat(TRTCChatSalonDef.UserInfo userInfo) {
}
```

Step 7. Mic on/off

- Room owner
- Listener

1. A room owner can make a listener speaker by passing in the `userId` of the listener to `pickSeat` . All members in the room will receive an `onAnchorEnterSeat` notification.
2. A room owner can remove a speaker by passing in the speaker’s `userId` to `kickSeat` . All members in the room will receive an `onAnchorLeaveSeat` notification.



After a speaker list operation, the order in which different notifications are sent is: callbacks > independent events such as `onAnchorEnterSeat` .

```
// 1. The room owner makes a listener speaker.
mTRTCChatSalon.pickSeat(”123”, new TRTCChatSalonCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
```

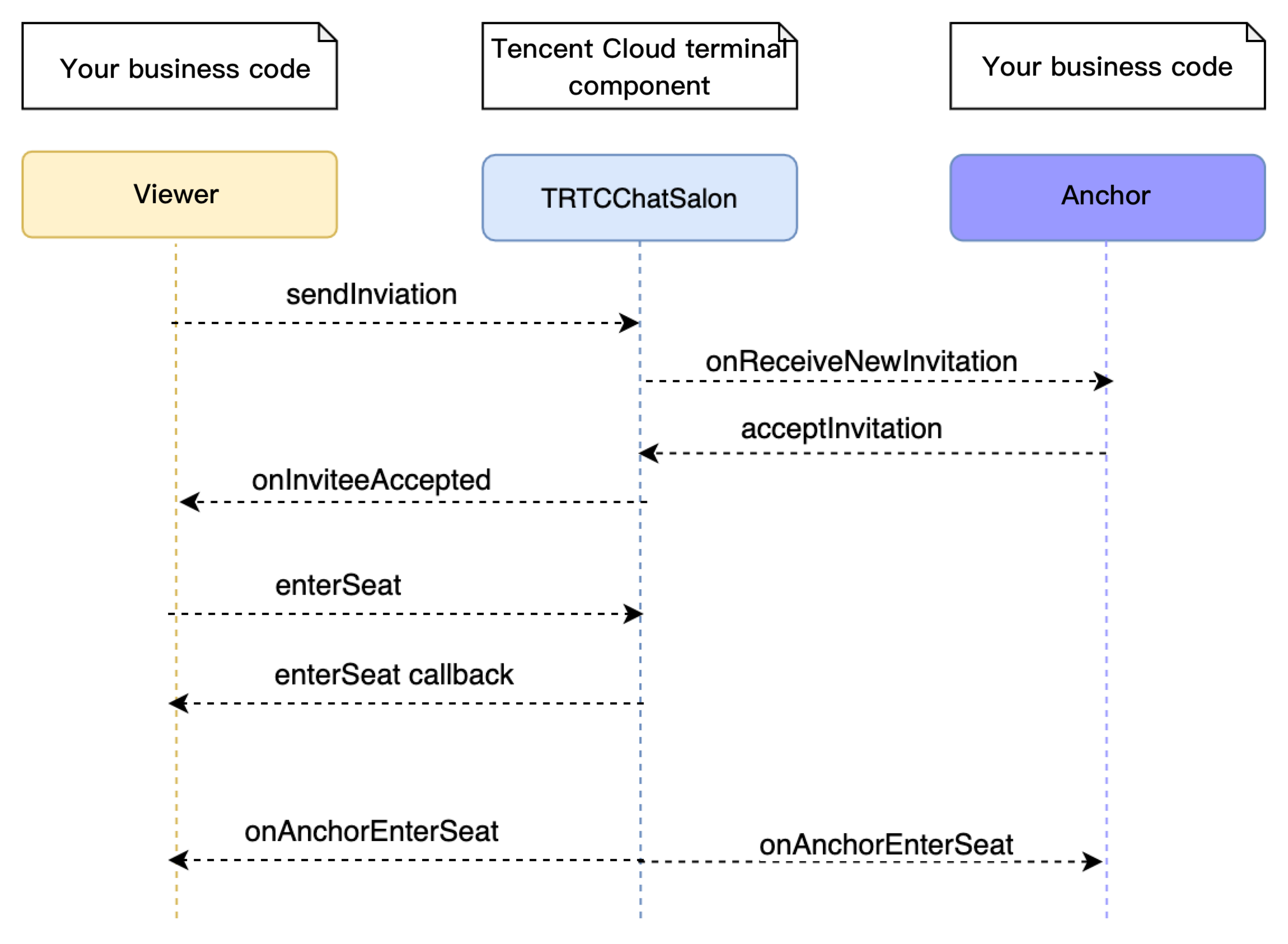
```
// 2. A callback is returned.
if (code == 0) {
}
}
});

// 3. The room owner receives a notification that someone became a speaker, and can determine whether it is the listener he or she intended to make speaker.
public void onAnchorEnterSeat(TRTCChatSalonDef.UserInfo user) {
}
```

Step 8. Use signaling for invitations

If you want listeners and room owners to obtain each other's consent before performing the above actions in your application, you can use signaling for invitation sending.

- Listener requesting to take seat
- Room owner inviting listener to take seat

1. A listener calls `sendInvitation` , passing in information including the room owner's `userId` and custom command words. The API will return an `inviteId` , which should be recorded.
2. The room owner receives an `onReceiveNewInvitation` notification, and a window pops up on the UI asking the room owner whether to accept the request.
3. The room owner calls `acceptInvitation` with the `inviteId` passed in to accept the request.
4. The listener receives an `onInviteeAccepted` notification and calls `enterSeat` to become a speaker.



```
// Listener
// 1. A listener calls sendInvitation to request to speak.
String inviteId = mTRTCChatSalon.sendInvitation(”ENTER_SEAT”, ownerUserId, ”123”, null);

// 2. Place the user in the seat after the invitation is accepted
@Override
public void onInviteeAccepted(String id, String invitee) {
if(id.equals(inviteId)) {
mTRTCChatSalon.enterSeat(null);
}
}

// Room owner
// 1. The room owner receives the request.
@Override
public void onReceiveNewInvitation(final String id, String inviter, String cmd, final String content) {
if (cmd.equals(”ENTER_SEAT”)) {
```

```
// 2. The room owner accepts the request.
mTRTCChatSalon.acceptInvitation(id, null);
}
}
```

Step 9. Enable text chat and on-screen comments

- Call `sendRoomTextMsg` to send common text messages. All users in the room will receive an `onRecvRoomTextMsg` callback.

  IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: send text messages
mTRTCChatSalon.sendRoomTextMsg(”Hello Word!”, null);
// Recipient: listen for text messages
mTRTCChatSalon.setDelegate(new TRTCChatSalonDelegate() {
@Override
public void onRecvRoomTextMsg(String message, TRTCChatSalonDef.UserInfo userInfo) {
Log.d(TAG, ”Received a message from” + userInfo.userName + ”: ” + message);
}
});
```

- Call `sendRoomCustomMsg` to send custom (signaling) messages. All users in the room will receive an `onRecvRoomCustomMsg` callback.

  Custom messages are often used to transfer custom signals, e.g., to give and broadcast likes.

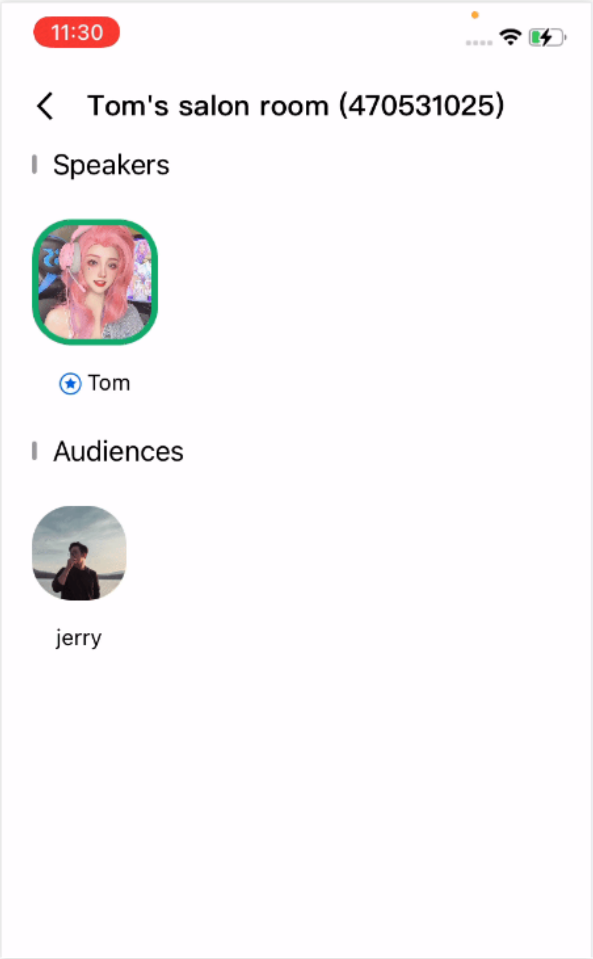
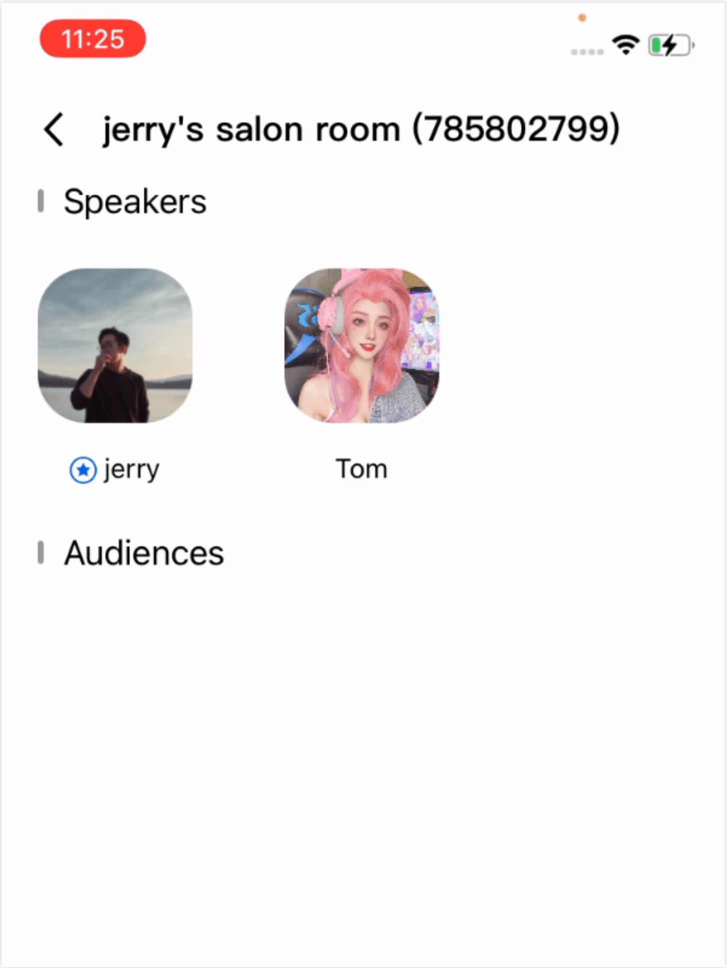
```
// A sender can customize CMD to distinguish on-screen comments and likes.
// For example, use ”CMD_DANMU” to indicate on-screen comments and ”CMD_LIKE” to indicate like
s.
mTRTCChatSalon.sendRoomCustomMsg(”CMD_DANMU”, ”Hello world”, null);
mTRTCChatSalon.sendRoomCustomMsg(”CMD_LIKE”, ””, null);
// Recipient: listen for custom messages
mTRTCChatSalon.setDelegate(new TRTCChatSalonDelegate() {
@Override
public void onRecvRoomCustomMsg(String cmd, String message, TRTCChatSalonDef.UserInfo userInf
o) {
if (”CMD_DANMU”.equals(cmd)) {
// An on-screen comment is received.
Log.d(TAG, ”Received an on-screen comment from” + userInfo.userName + ”: ” + message);
} else if (”CMD_LIKE”.equals(cmd)) {
// A like is received.
Log.d(TAG, userInfo.userName + ”liked you.”);
}
}
});
```

Chat Salon (Flutter)

Last updated : 2022-03-18 23:03:18

Demo UI

To quickly enable the chat salon feature, you can modify the demo we provide and adapt it to your needs. You may also use the `TRTCChatSalon` component and customize your own UI.

Room Owner	Listener
 11:30 Tom's salon room (470531025) Speakers Tom Audiences jerry	 11:25 jerry's salon room (785802799) Speakers jerry Tom Audiences

Using the Demo UI

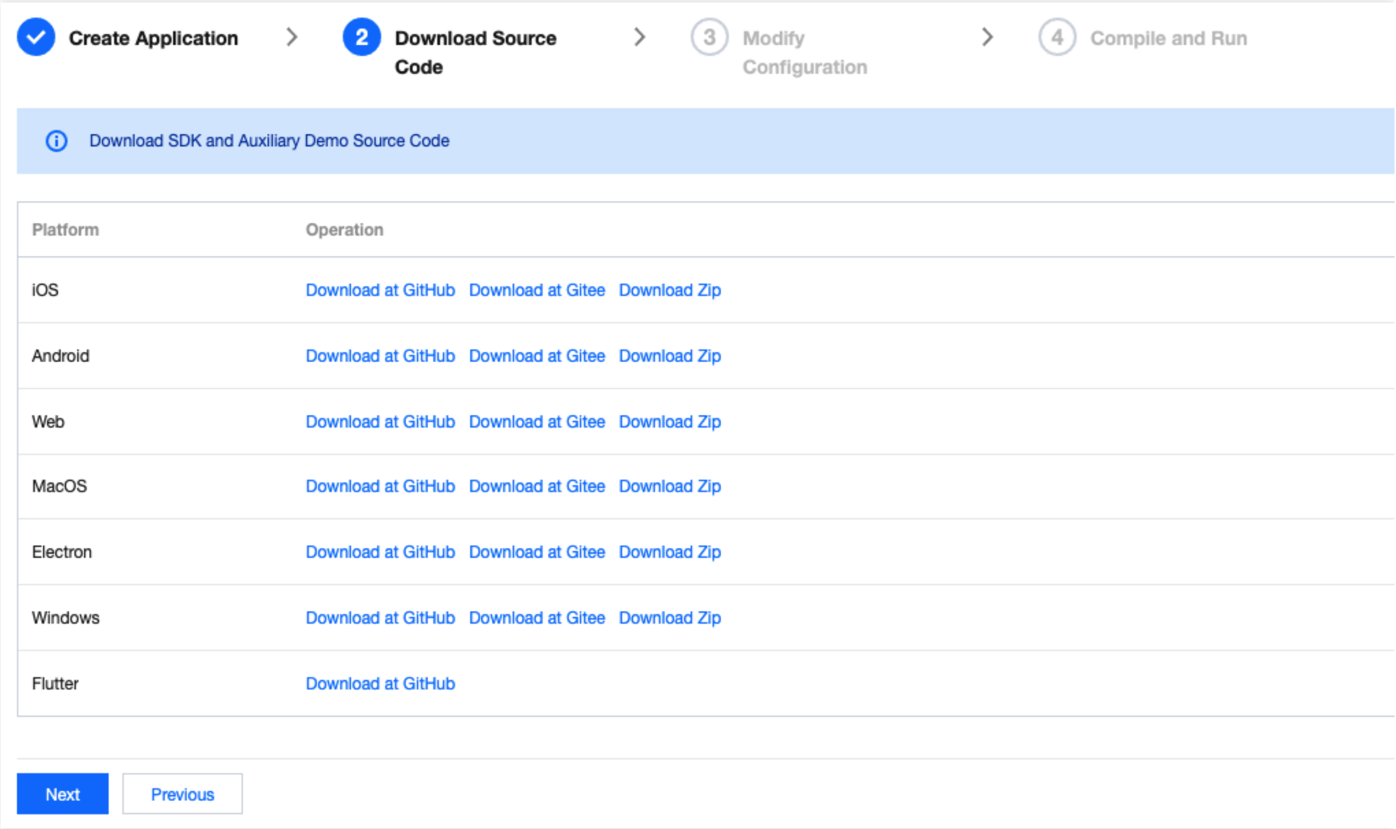
Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestChatSalon` and click **Create**.

> Note :
> This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

Step 2. Download the SDK and demo source code

1. Download the SDK and demo source code for your platform.
2. Click **Next**.



Download SDK and Auxiliary Demo Source Code

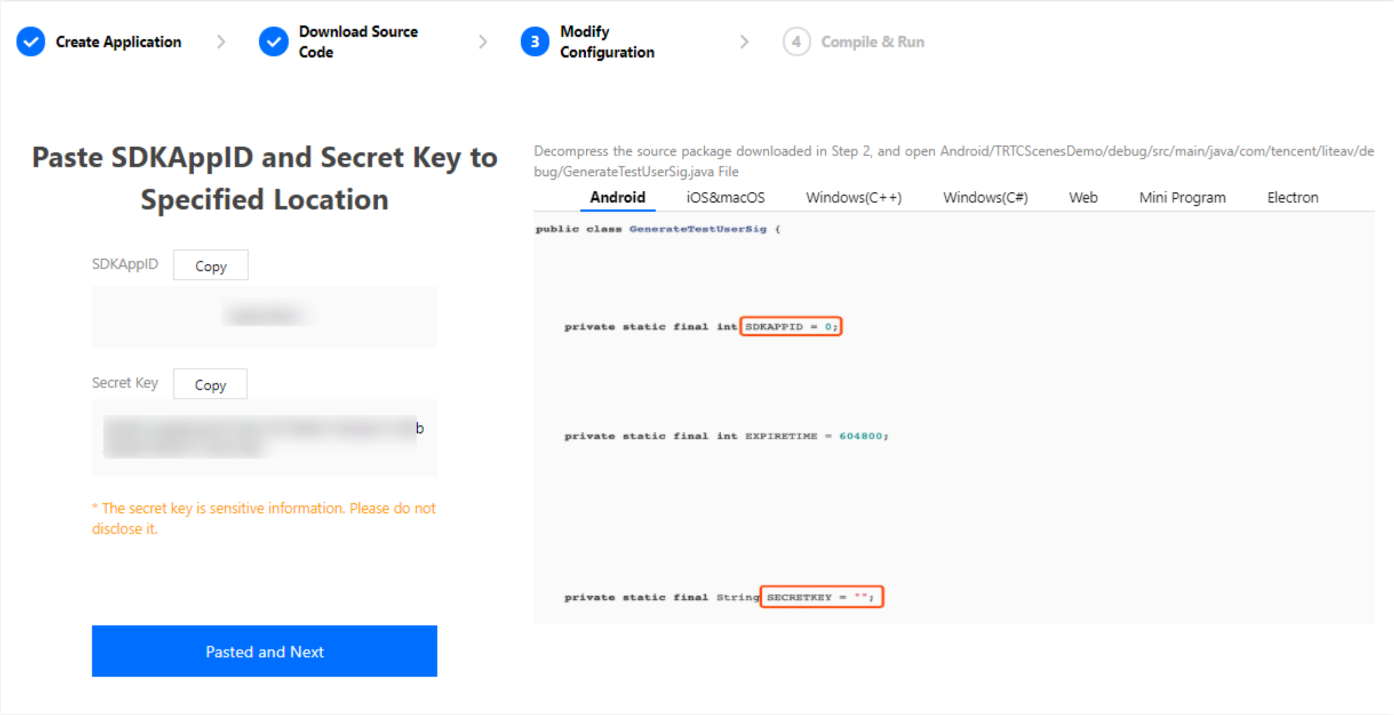
Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Step 3. Configure the demo project file

1. In the **Modify Configuration** step, select your platform.

2. Find and open `/example/lib/debug/GenerateTestUserSig.dart` .

3. Set parameters in `GenerateTestUserSig.dart` as follows.
    - `SDKAPPID`: a placeholder by default. Set it to the actual `SDKAppID`.
    - `SECRETKEY`: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID [redacted]

Secret Key [redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note：
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is leaked, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for the local execution and debugging of the demo**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see How do I calculate UserSig on the server?.

Step 4. Compile and run

> Note：
> An Android project must be run on a real device rather than a simulator.

1. Run `flutter pub get` .
2. Compile, run, and test the project.
   - Android
   - iOS
   1. Run `flutter run` .
   2. Open the demo project with Android Studio (3.5 or above), and click **Run**.

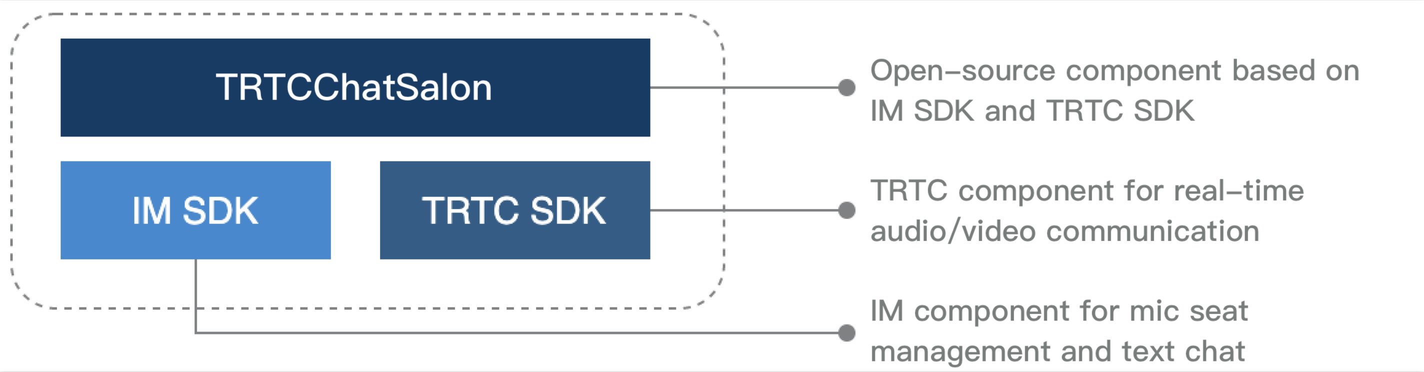
Step 5. Modify the demo source code

The `trtcchatsalondemo` folder in the source code contains two subfolders: `ui` and `model` . The `ui` folder contains the UI code. The table below lists the files (folders) and the UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
base	Basic classes used by the UI
list	The list and room creation views
room	The main room views for room owner and listener
widget	Common controls

Customizing UI

The `trtcchatsalondemo` folder in the source code contains two subfolders: `ui` and `model` . The `model` subfolder contains the reusable open-source component `TRTCChatSalon` . You can find the component's APIs in `TRTCChatSalon.dart` and use them to customize your own UI.


TRTCChatSalon
IM SDK
TRTC SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for mic seat management and text chat

Step 1. Integrate the SDKs

The chat salon component `trtcchatsalondemo` depends on the TRTC SDK and IM SDK. You can configure `pubspec.yaml` to download their updates automatically.

Add the following dependencies to `pubspec.yaml` of your project.

```
dependencies:
tencent_trtc_cloud: latest version number
tencent_im_sdk_plugin: latest version number
```

Step 2. Configure permission requests and obfuscation rules

- iOS
- Android

Add requests for mic permission in `Info.plist` :

```
<key>NSMicrophoneUsageDescription</key>
<string>Audio calls are possible only with mic access.</string>
```

Step 3. Import the `TRTCChatSalon` component

Copy all the files in the directory below to your project:

```
lib/TRTCChatSalonDemo/model/
```

Step 4. Create an instance and log in

1. Call the `sharedInstance` API to create an instance of the `TRTCChatSalon` component.
2. Call the `registerListener` function to register event callbacks of the component.

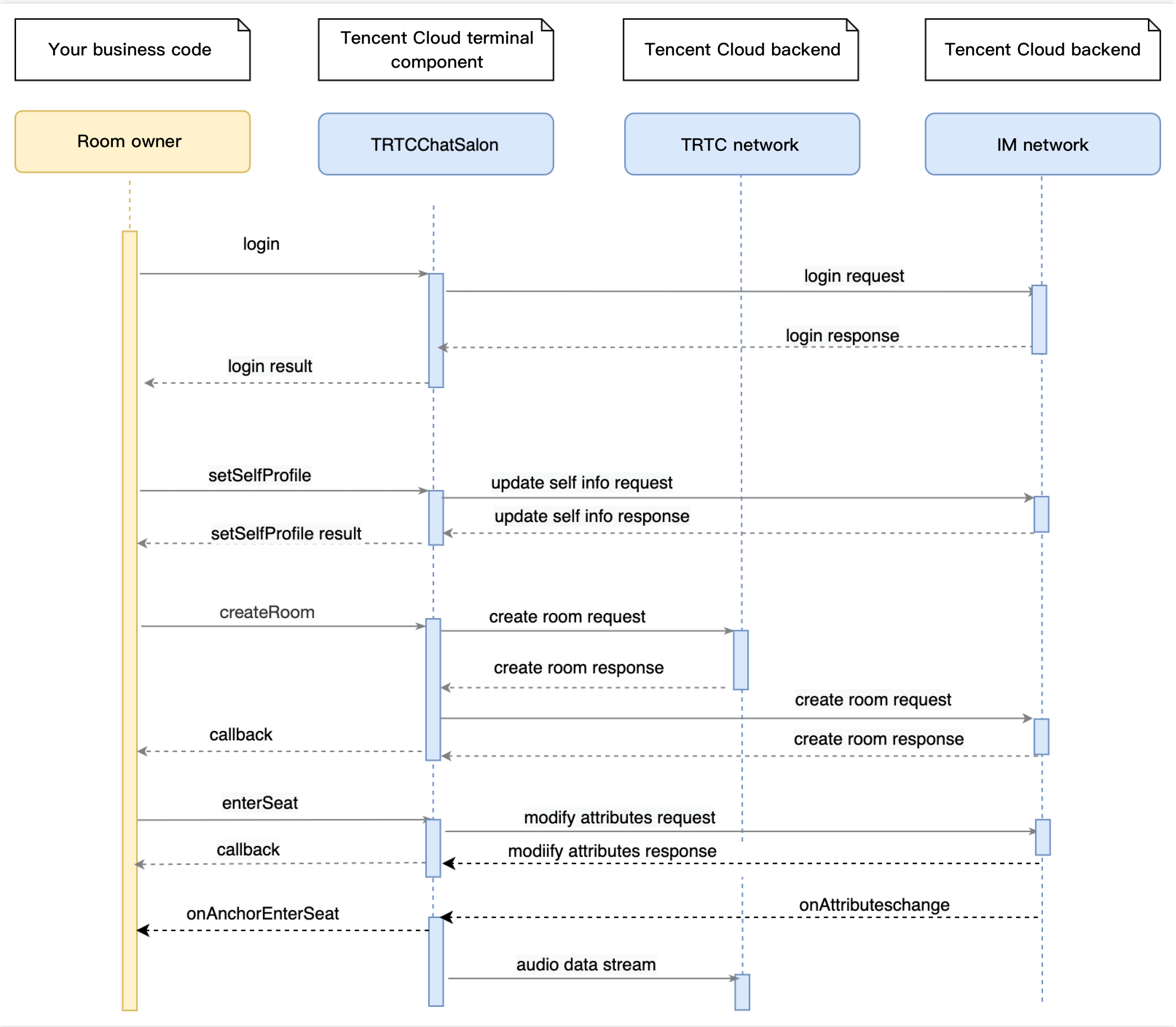
3. Call the `login` API to log in to the component, and set the key parameters as described below.

Parameter	Description
SDKAppID	You can view `SDKAppID` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.

```
TRTCChatSalon trtcVoiceRoom = await TRTCChatSalon.sharedInstance();
trtcVoiceRoom.registerListener(onVoiceListener);
ActionCallback resValue = await trtcVoiceRoom.login(
GenerateTestUserSig.sdkAppId,
userId,
GenerateTestUserSig.genTestSig(userId),
);
if (resValue.code == 0) {
// Logged in
}
```

Step 5. Create a room and become a speaker

1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. Call `createRoom` to create a chat salon, passing in room attributes such as room ID and room name.
3. You will receive a `TRTCChatSalonDelegate.onAudienceEnter` notification that someone entered the room, and mic capturing will be enabled automatically.



```
// 1. Set your nickname and profile photo.
trtcVoiceRoom.setSelfProfile("my_name", "my_face_url", null);

// 2. Call `createRoom` to create a room
ActionCallback resp = await trtcVoiceRoom.createRoom(
roomId,
RoomParam(
coverUrl: 'Room cover URL',
roomName: 'Room name',
),
);
if (resp.code == 0) {
//3. Become a speaker
}

// 4. Receive a `TRTCChatSalonDelegate.onAudienceEnter` notification
onVoiceListener(type, param) async {
switch (type) {
case TRTCChatSalonDelegate.onAudienceEnter:
// A listener enters the room
break;
}
}
```

Step 6. Enter a room as a listener

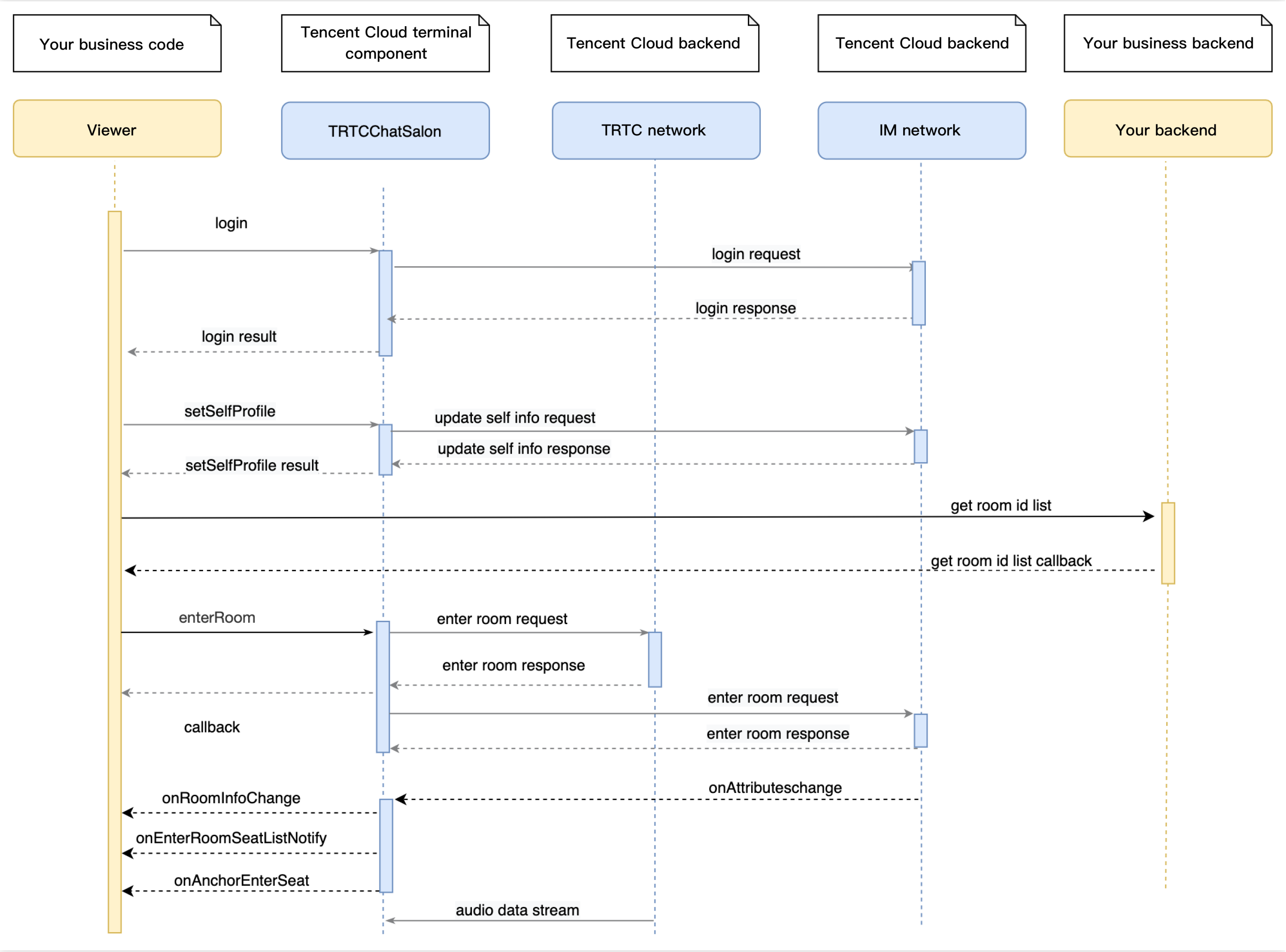
1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. Get the latest chat salon room list from the backend.

> Note :
> The chat salon list in the demo is for demonstration only. The business logic of the chat salon list varies significantly. Tencent Cloud does not provide list management services for the time being. Please manage the list by yourself.

3. Call `getRoomInfoList` to get short descriptions of the rooms, which are provided by the room owner during room creation via the calling of `createRoom` .

> Note :
> If your chat salon list already contains enough room information, you can skip the step of calling `getRoomList` . Just pass in the room ID to enter the room.

4. After room entry, you will receive the `TRTCChatSalonDelegate.onAudienceEnter` and `TRTCChatSalonDelegate.onAudienceExit` notifications about listeners' entry/exit. Refresh the information on the UI accordingly.
5. You will also receive the `TRTCChatSalonDelegate.onAnchorEnterMic` and `TRTCChatSalonDelegate.onAnchorLeaveMic` notifications that someone becomes a speaker or listener.



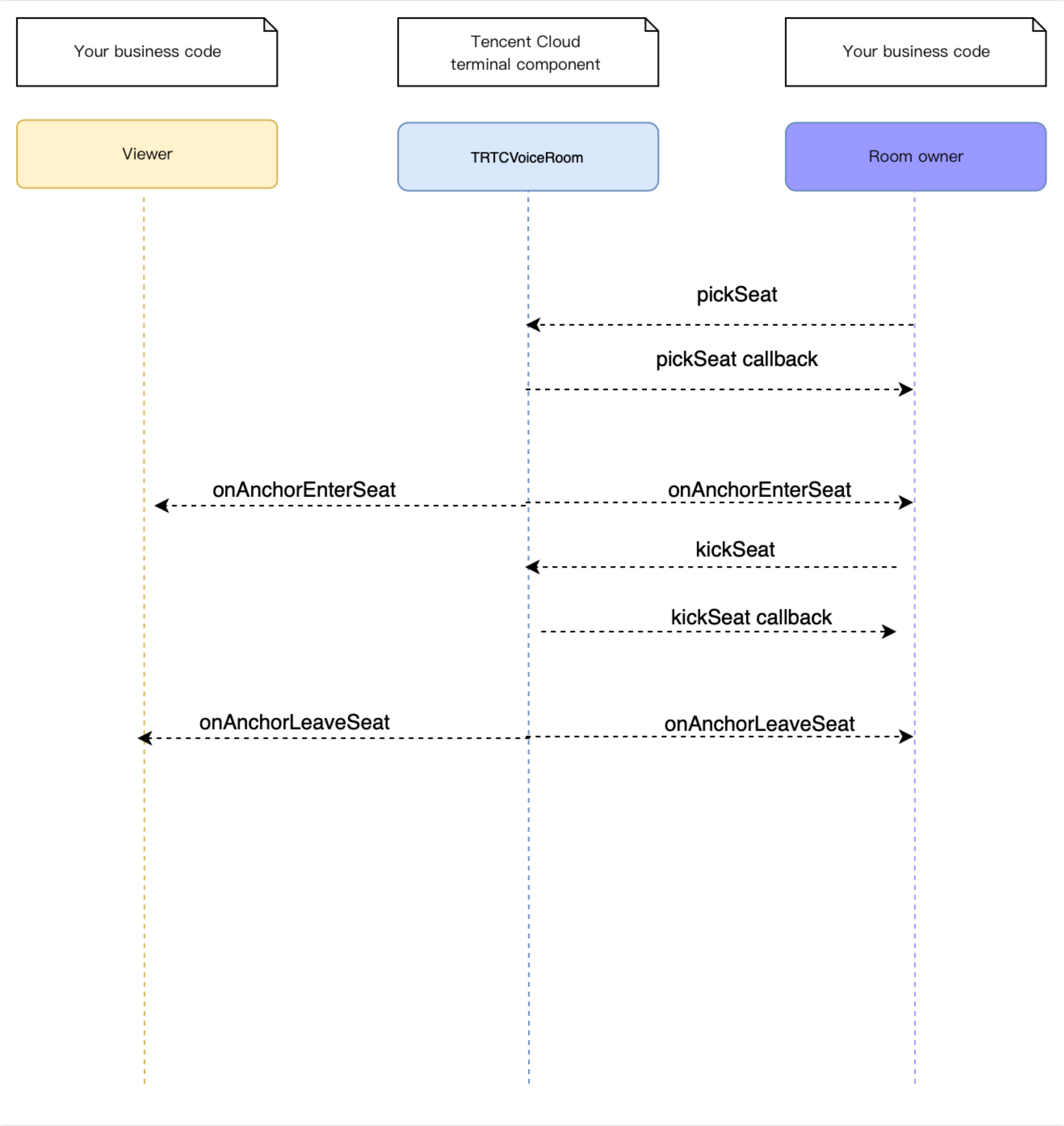
```
// 1. Set your nickname and profile photo.
trtcVoiceRoom.setSelfProfile("my_name", "my_face_url");

// 2. Get the room list from the backend. Suppose it is `roomList`.
List<integer> roomList = GetRoomList();

// 3. Call `getRoomInfoList` to get details of the rooms.
RoomInfoCallback resp = await trtcVoiceRoom.getRoomInfoList(roomList);
if (resp.code == 0) {
//Refresh the room list on your UI
}

// 4. Pass in `roomId` to enter a room.
ActionCallback enterRoomResp =
await trtcVoiceRoom.enterRoom(_currentRoomId);
if (enterRoomResp.code == 0) {
// Entered room successfully
}
```

```
// 5. After successful room entry, you receive an event notification
onVoiceListener(type, param) async {
switch (type) {
case TRTCChatSalonDelegate.onAudienceEnter:
// A listener enters the room
break;
case TRTCChatSalonDelegate.onAudienceExit:
//A listener leaves the room
break;
case TRTCChatSalonDelegate.onAnchorLeaveMic:
//The room owner leaves the room
break;
case TRTCChatSalonDelegate.onAnchorEnterMic:
//The room owner enters the room
break;
}
}
```

Step 7. Mic on/off

- Room owner
- Listener

1. A room owner can call `leaveMic` to become a listener. All users in the room will receive an `onAnchorLeaveMic` notification.
2. A room owner can remove a speaker by passing in the speaker's userId to `kickMic` . All users in the room will receive an `onAnchorLeaveMic` notification.



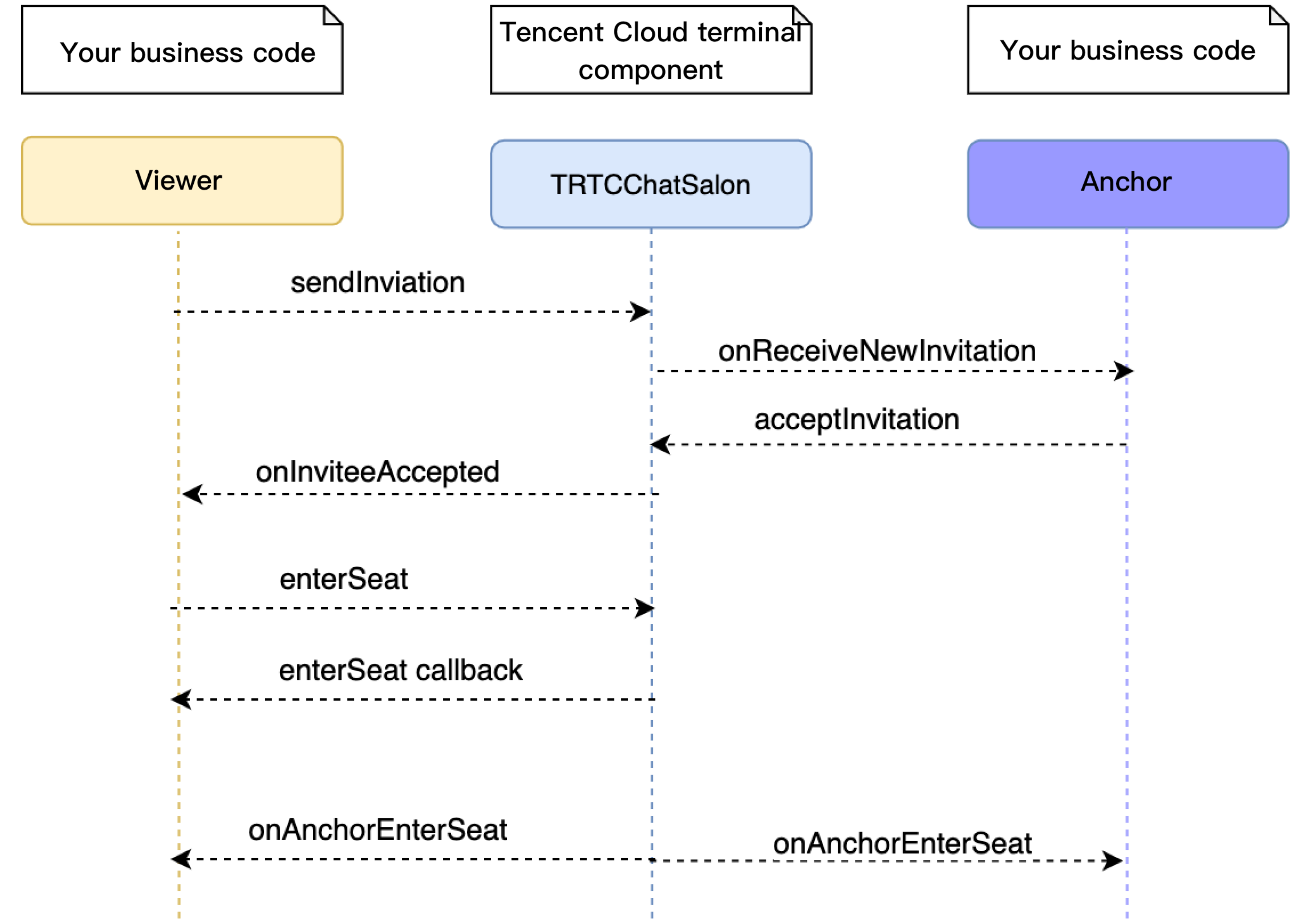
```
// 1. Become a listener
trtcVoiceRoom.leaveMic();

//2. Remove a speaker
trtcVoiceRoom.kickMic(userId);
```

Step 8. Use signaling for invitations

If you want listeners and room owners to obtain each other's consent before performing the above actions in your application, you can use signaling for invitation sending.

::: Listener requesting to speak

1. A listener calls `raiseHand` to request to speak.
2. The room owner receives an `onRaiseHand` notification, and a window pops up on the UI asking the room owner whether to accept the request.
3. The room owner accepts the request and calls `agreeToSpeak` , with the listener's `userId` passed in.
4. The listener receives an `onAgreeToSpeak` notification and calls `enterMic` to become a speaker.



```
// Listener
// 1. A listener calls `sendInvitation` to request to speak.
trtcVoiceRoom.raiseHand();
```

```
// 2. Place the user in the seat after the invitation is accepted
onVoiceListener(type, param) async {
switch (type) {
case TRTCChatSalonDelegate.onRaiseHand:
trtcVoiceRoom.enterMic();
break;
}
}

// Room owner
// 1. The room owner receives the request.
onVoiceListener(type, param) async {
switch (type) {
case TRTCChatSalonDelegate.onAgreeToSpeak:
trtcVoiceRoom.agreeToSpeak();
break;
}
}
```

Step 9. Enable text chat and on-screen comments

Call `sendRoomTextMsg` to send common text messages. All users in the room will receive an `onRecvRoomTextMsg` callback.

IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: send text messages
trtcVoiceRoom.sendRoomTextMsg("Hello Word!");
// Recipient: listen for text messages
onVoiceListener(type, param) async {
switch (type) {
case TRTCChatSalonDelegate.onRecvRoomTextMsg:
// Group text messages are received. This feature can be used to enable text chat rooms.
break;
}
}
```

TRTCChatSalon (iOS)

Last updated : 2022-04-21 16:19:12

`TRTCChatSalon` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create a chat salon and become a speaker, or enter a salon as a listener.
- The room owner can invite a listener to speak as well as remove a speaker.
- A listener can request to speak and become a speaker. A speaker can also become a listener.
- All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

`TRTCChatSalon` is an open-source class depending on two closed-source Tencent Cloud SDKs.

- TRTC SDK: the TRTC SDK is used as a low-latency audio chat component.
- IM SDK: the `AVChatRoom` feature of the IM SDK is used to implement chat rooms. The attribute APIs of IM are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

`TRTCChatSalon` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
setDelegate	Sets event callback.
delegateQueue	Sets the thread where the event callback is.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
destroyRoom	Terminates a room (called by room owner).
enterRoom	Enters a room (called by listener).
exitRoom	Exits a room (called by listener).
getRoomInfoList	Gets room list details.
getUserInfoList	Gets the user information of the specified `userId` . If the value is `nil` , the information of all users in the room is obtained.

Mic APIs

API	Description
enterSeat	Becomes a speaker (called by room owner or listener).
leaveSeat	Becomes a listener (called by speaker).
pickSeat	Invites a listener to speak (called by room owner).
kickSeat	Removes a speaker (called by room owner).

Local audio APIs

API	Description
startMicrophone	Enables mic capturing.
stopMicrophone	Stops mic capturing.
setAudioQuality	Sets audio quality.
muteLocalAudio	Mutes/Unmutes local audio.
setSpeaker	Turns the speaker on.
setAudioCaptureVolume	Sets mic capturing volume.
setAudioPlayoutVolume	Sets playback volume.

Remote audio APIs

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

Background music and audio effect APIs

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

Message sending APIs

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

Invitation signaling APIs

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

`TRTCChatSalonDelegate` API Overview

Common event callback APIs

API	Description
onError	Error

API	Description
onWarning	Warning
onDebugLog	Log

Room event callback APIs

API	Description
onRoomDestroy	Room termination
onRoomInfoChange	Room information change
onUserVolumeUpdate	User volume

Seat list change callback APIs

API	Description
onAnchorEnterSeat	Someone became a speaker after requesting or being invited by the room owner.
onAnchorLeaveSeat	Someone became a listener or was moved to listeners by the room owner.
onSeatMute	The room owner muted a speaker.

Callback APIs for room entry/exit by listeners

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

Message event callback APIs

API	Description
onRecvRoomTextMsg	Receipt of a text message
onRecvRoomCustomMsg	Receipt of a custom message

Signaling event callback APIs

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

Basic SDK APIs

sharedInstance

This API is used to get a TRTCChatSalon singleton object.

```
/**
* Get a `TRTCChatSalon` singleton object
*
* - returns: `TRTCChatSalon` instance
* - note: you can call `{@link TRTCChatSalon#destroySharedInstance()}` to terminate a singleton object.
*/
+ (instancetype)sharedInstance NS_SWIFT_NAME(shared());
```

destroySharedInstance

This API is used to terminate a TRTCChatSalon singleton object.

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

```
/**
* Terminate the `TRTCChatSalon` singleton object
*
* - note: after the instance is terminated, the externally cached `TRTCChatSalon` instance can no longer be used. You need to call `{@link TRTCChatSalon#sharedInstance()}` again to get a new instance.
```

```
*/
+ (void)destroySharedInstance NS_SWIFT_NAME(destroyShared());
```

setDelegate

This API is used to set the event callback of TRTCChatSalon. You can use `TRTCChatSalonDelegate` to get different status notifications of TRTCChatSalon.

```
/**
* Set the component callback
*
* You can use `TRTCChatSalonDelegate` to get status notifications of `TRTCChatSalon`
*
* - parameter delegate Callback API
* - note: events in `TRTCChatSalon` are called back to you in the main queue by default. If you n
eed to specify a queue for event callback, please use `{@link TRTCChatSalon#setDelegateQueue(queu
e)}`
*/
- (void)setDelegate:(id<TRTCChatSalonDelegate>)delegate NS_SWIFT_NAME(setDelegate(delegate:));
```

> Note :
> `setDelegate` is the delegate callback of `TRTCChatSalon` .

setDelegateQueue

This API is used to set the thread queue for event callback. The main thread (MainQueue) is used by default.

```
/**
* Set the queue for event callback
*
* - parameter queue. The status notifications of `TRTCChatSalon` will be sent to the queue you sp
ecify.
*/
- (void)setDelegateQueue:(dispatch_queue_t)queue NS_SWIFT_NAME(setDelegateQueue(queue:));
```

The parameters are as detailed below:

Parameter	Type	Description
queue	dispatch_queue_t	The status notifications of `TRTCChatSalon` will be sent to the thread queue you specify.

login

This API is used to log in.

```
- (void)login:(int)sdkAppID
userID:(NSString *)userID
userSig:(NSString *)userSig
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(login(sdkAppID:userID:userSig:callback
:));
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	int	You can view the `SDKAppID` via **Application Management** > **Application Info** in the TRTC console.
userId	String	ID of 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 more information on how to get it, please see UserSig.
callback	ActionCallback	Callback for login. The `code` will be 0 if login succeeds.

logout

This API is used to log out.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for logout. The code is 0 if logout succeeds.

setSelfProfile

This API is used to set profile.

```
- (void)setSelfProfile:(NSString *)userName avatarURL:(NSString *)avatarURL callback:(ActionCallb
ack _Nullable)callback NS_SWIFT_NAME(setSelfProfile(userName:avatarURL:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userName	String	Nickname
avatar	String	Profile photo address
callback	ActionCallback	Callback for profile setting. The code is 0 if the operation succeeds.

Room APIs

createRoom

This API is used to create a room (called by room owner).

```
- (void)createRoom:(int)roomID roomParam:(ChatSalonParam *)roomParam callback:(ActionCallback _Nu
llable)callback NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a chat salon room list. Currently, Tencent Cloud does not provide management services for chat salon room lists. Please manage the list on your own.
roomParam	ChatSalonParam	Room information, such as room name, speaker list information, and cover information
callback	ActionCallback	Callback for room creation result. The code is 0 if the operation succeeds.

The process of creating an audio chat room and becoming a speaker is as follows:

1. A user calls `createRoom` to create a chat salon, passing in room attributes (e.g., room ID and whether listeners require room owner’s consent to speak).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

destroyRoom

This API is used to terminate a room (called by room owner).

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for room termination result. The `code` is 0 if the operation succeeds.

enterRoom

This API is used to enter a room (called by listener).

```
- (void)enterRoom:(NSInteger)roomID callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(ent
erRoom(roomID:callback:));
```

The parameters are as detailed below:

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

The process of entering a room as a listener is as follows:

1. A user gets the latest chat salon list from your server. The list may contain the `roomId` and room information of multiple chat salons.
2. Select a chat salon, and call `enterRoom` with the room ID passed in to enter.
3. After entering the room, you will receive an `onRoomInfoChange` notification about room change from the component. Record the room information, including room name, whether the room owner’s consent is required for listeners to speak, etc., and update it to the UI.
4. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker.

exitRoom

This API is used to exit a room.

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

The parameters are as detailed below:

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

getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()` .

> Note :
> You don’t need this API if both the room list and room information are managed on your server.

```
- (void)getRoomInfoList:(NSArray<NSNumber *> *)roomIdList callback:(ChatSalonInfoCallback _Nullable)callback NS_SWIFT_NAME(getRoomInfoList(roomIdList:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomIdList	List<Integer>	Room ID list
callback	ChatSalonInfoCallback	Callback for room details

getUserInfoList

This API is used to get the user information of a specified `userId` .

```
- (void)getUserInfoList:(NSArray<NSString *> * _Nullable)userIDList callback:(ChatSalonUserListCallback _Nullable)callback NS_SWIFT_NAME(getUserInfoList(userIDList:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userIdList	List<String>	List of user IDs to obtain. If this parameter is `null` , the information of all users in the room is obtained.
callback	ChatSalonUserListCallback	Callback for user details

Mic APIs

enterSeat

This API is used to become a speaker (called by room owner or listener).

> Note :
> After a user becomes a speaker, all members in the room will receive an `onAnchorEnterSeat` notification.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for operation

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

leaveSeat

This API is used to remove a speaker (called by room owner).

> Note :
> After a speaker becomes a listener, all members in the room will receive an `onAnchorLeaveSeat` notification.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for operation

pickSeat

This API is used to invite a listener to speak (called by room owner).

> Note :
> After a listener becomes a speaker following the room owner's invitation, all members in the room will receive an `onAnchorEnterSeat` notification.

```
- (void)pickSeat:(NSString *)userID callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(pickSeat(userID:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
UserID	String	User ID
callback	ActionCallback	Callback for operation

Calling this API will immediately modify the seat list. In cases where the room owner needs listeners’ consent to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept` , call `pickSeat` .

kickSeat

This API is used to remove a speaker (called by room owner).

> Note :
> After a speaker is removed, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.

```
- (void)kickSeat:(NSString *)userID callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(kickSeat(userID:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	User ID of the speaker to remove

Parameter	Type	Description
callback	ActionCallback	Callback for operation

Calling this API will immediately modify the speaker list.

Local Audio APIs

startMicrophone

This API is used to enable mic capturing.

```
- (void)startMicrophone;
```

stopMicrophone

This API is used to stop mic capturing.

```
- (void)stopMicrophone;
```

setAudioQuality

This API is used to set the audio quality.

```
- (void)setAuidoQuality:(NSInteger)quality NS_SWIFT_NAME(setAuidoQuality(quality:));
```

The parameters are as detailed below:

Parameter	Type	Description
quality	int	Audio quality. For more information, please see TRTC SDK.

muteLocalAudio

This API is used to mute/unmute the local audio.

```
- (void)muteLocalAudio:(BOOL)mute NS_SWIFT_NAME(muteLocalAudio(mute:));
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
mute	boolean	Mutes/Unmutes. For more information, please see TRTC SDK.

setSpeaker

This API is used to turn the speaker on.

```
- (void)setSpeaker:(BOOL)userSpeaker NS_SWIFT_NAME(setSpeaker(userSpeaker:));
```

The parameters are as detailed below:

Parameter	Type	Description
userSpeaker	boolean	`true` : speaker; `false` : receiver

setAudioCaptureVolume

This API is used to set the mic capturing volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Capturing volume. Value range: 0-100 (default: 100)

setAudioPlayoutVolume

This API is used to set the playback volume.

```
- (void)setAudioPlayoutVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioPlayoutVolume(volume:));
```

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Playback volume. Value range: 0-100 (default: 100)

muteRemoteAudio

This API is used to mute/unmute a specified user.

```
- (void)muteRemoteAudio:(NSString *)userID mute:(BOOL)mute NS_SWIFT_NAME(muteRemoteAudio(userId:m
ute:));
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Specified user ID
mute	boolean	`true` : mute; `false` : unmute

muteAllRemoteAudio

This API is used to mute/unmute all users.

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

The parameters are as detailed below:

Parameter	Type	Description
isMute	boolean	`true` : mute; `false` : unmute

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

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

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a room, which is generally used for on-screen comments.

```
- (void)sendRoomTextMsg:(NSString *)message callback:(ActionCallback _Nullable)callback NS_SWIFT_
NAME(sendRoomTextMsg(message:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
callback	ActionCallback	Callback for operation

sendRoomCustomMsg

This API is used to send a custom text message.

```
- (void)sendRoomCustomMsg:(NSString *)cmd message:(NSString *)message callback:(ActionCallback _N
ullable)callback NS_SWIFT_NAME(sendRoomCustomMsg(cmd:message:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message
callback	ActionCallback	Callback for operation

Invitation Signaling APIs

sendInvitation

This API is used to send an invitation.

```
- (NSString *)sendInvitation:(NSString *)cmd
userID:(NSString *)userID
content:(NSString *)content
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendInvitation(cmd:userId:content:callb
ack:));
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command of business

Parameter	Type	Description
userID	String	Invitee’s user ID
content	String	Invitation content
callback	ActionCallback	Callback for operation

Returned value:

Returned Value	Type	Description
inviteId	String	Invitation ID

acceptInvitation

This API is used to accept an invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
identifier	String	Invitation ID
callback	ActionCallback	Callback for sending result

rejectInvitation

This API is used to decline an invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
identifier	String	Invitation ID
callback	ActionCallback	Callback for sending result

cancelInvitation

This API is used to cancel an invitation.

```
- (void)cancelInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(cancelInvitation(identifier:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
identifier	String	Invitation ID
callback	ActionCallback	Callback for sending result

TRTCChatSalonDelegate Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

> 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*)message
NS_SWIFT_NAME(onError(code:message:));
```

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Error message

onWarning

Callback for warning.

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

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Warning message

onDebugLog

Callback for log.

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

The parameters are as detailed below:

Parameter	Type	Description
message	String	Log information

Room Event Callback APIs

onRoomDestroy

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.

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

The parameters are as detailed below:

Parameter	Type	Description
roomId	String	Room ID.

onRoomInfoChange

Callback for successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.

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

The parameters are as detailed below:

Parameter	Type	Description
roomInfo	ChatSalonInfo	Room information

onUserVolumeUpdate

Callback of the volume of each member in the room after the volume reminder is enabled.

```
- (void)onUserVolumeUpdate:(NSArray<TRTCVolumeInfo *> *)userVolumes totalVolume:(NSInteger)totalVolume
NS_SWIFT_NAME(onUserVolumeUpdate(userVolumes:totalVolume:));
```

The parameters are as detailed below:

Parameter	Type	Description
userVolumes	NSArray<TRTCVolumeInfo *>	Volume of each user
totalVolume	int	Overall volume

Seat Callback APIs

onAnchorEnterSeat

Someone became a speaker after requesting or being invited by the room owner.

```
- (void)onAnchorEnterSeat:(ChatSalonUserInfo *)user
NS_SWIFT_NAME(onAnchorEnterSeat(user:));
```

The parameters are as detailed below:

Parameter	Type	Description
user	ChatSalonUserInfo	Details of the user who became a speaker

onAnchorLeaveSeat

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

```
- (void)onAnchorLeaveSeat:(ChatSalonUserInfo *)user
NS_SWIFT_NAME(onAnchorLeaveSeat(user:));
```

The parameters are as detailed below:

Parameter	Type	Description
user	ChatSalonUserInfo	Details of the user who became a speaker

onSeatMute

The room owner muted a seat.

```
- (void)onSeatMute:(NSString *)userID
isMute:(BOOL)isMute
NS_SWIFT_NAME(onSeatMute(userID:isMute:));
```

The parameters are as detailed below:

Parameter	Type	Description
userID	String	ID of the speaker muted
isMute	boolean	`true` : muted; `false` : unmuted

Callback APIs for Room Entry/Exit by Listener

onAudienceEnter

A listener entered the room.

```
- (void)onAudienceEnter:(ChatSalonUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceEnter(userInfo:));
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	ChatSalonUserInfo	Information of the user who entered the room

onAudienceExit

A listener exited the room.

```
- (void)onAudienceExit:(ChatSalonUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceExit(userInfo:));
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	ChatSalonUserInfo	Information of the user who exited the room

Message Event Callback APIs

onRecvRoomTextMsg

A text message was received.

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

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
userInfo	ChatSalonUserInfo	User information of sender

onRecvRoomCustomMsg

A custom message was received.

```
- (void)onRecvRoomCustomMsg:(NSString *)cmd
message:(NSString *)message
userInfo:(ChatSalonUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomCustomMsg(cmd:message:userInfo:));
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message
userInfo	ChatSalonUserInfo	User information of sender

Invitation Signaling Callback APIs

onReceiveNewInvitation

An invitation was received.

```
- (void)onReceiveNewInvitation:(NSString *)identifier
inviter:(NSString *)inviter
cmd:(NSString *)cmd
content:(NSString *)content
NS_SWIFT_NAME(onReceiveNewInvitation(identifier:inviter:cmd:content:));
```

The parameters are as detailed below:

Parameter	Type	Description
identifier	String	Invitation ID.
inviter	String	Inviter’s user ID
cmd	String	Custom command word specified by business
content	String	Content specified by business

onInviteeAccepted

The invitee accepted the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
identifier	String	Invitation ID
invitee	String	Invitee’s user ID

onInviteeRejected

The invitee declined the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
identifier	String	Invitation ID
invitee	String	Invitee’s user ID

onInvitationCancelled

The inviter canceled the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
identifier	String	Invitation ID.
invitee	String	Invitee’s user ID

onInvitationTimeout

The invitation timed out.

```
- (void)onInvitationTimeout:(NSString *)identifier;
```

The parameters are as detailed below:

Parameter	Type	Description
identifier	String	Invitation ID

TRTCChatSalon (Android)

Last updated：2021-09-06 11:18:26

`TRTCChatSalon` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create a chat salon and become a speaker, or enter a salon as a listener.
- The room owner can invite a listener to speak as well as remove a speaker.
- A listener can request to speak and become a speaker. A speaker can also become a listener.
- All users can send text and custom messages. Custom messages can be used to send on-screen comments, give likes, and send gifts.

`TRTCChatSalon` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Chat Salon (Android).

- TRTC SDK: the TRTC SDK is used as a low-latency audio chat component.
- IM SDK: the `AVChatRoom` feature of the IM SDK is used to implement chat rooms. The attribute APIs of IM are used to store room information such as the speaker list, and invitation signaling is used to send requests to speak or invite others to speak.

`TRTCChatSalon` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets singleton object.
destroySharedInstance	Terminates singleton object.
setDelegate	Sets event callback.
setDelegateHandler	Sets the thread where the event callback is.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates room (called by room owner). If the room does not exist, the system will automatically create a room.
destroyRoom	Terminates room (called by room owner).
enterRoom	Enters room (called by listener).
exitRoom	Exits room (called by listener).
getRoomInfoList	Gets room list details.
getUserInfoList	Gets the user information of the specified `userId` . If the value is `null` , the information of all users in the room is obtained.

Mic APIs

API	Description
enterSeat	Becomes a speaker (called by room owner or listener).
leaveSeat	Becomes a listener (called by speaker).
pickSeat	Invites a listener to speak (called by room owner).
kickSeat	Removes a speaker (called by room owner).

Local audio APIs

API	Description
startMicrophone	Enables mic capturing.
stopMicrophone	Stops mic capturing.
setAudioQuality	Sets audio quality.
muteLocalAudio	Mutes/Unmutes local audio.
setSpeaker	Turns the speaker on.
setAudioCaptureVolume	Sets mic capturing volume.
setAudioPlayoutVolume	Sets playback volume.

Remote audio APIs

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

Background music and sound effect APIs

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

Message sending APIs

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

Invitation signaling APIs

API	Description
sendInvitation	Sends invitation.
acceptInvitation	Accepts invitation.
rejectInvitation	Declines invitation.
cancelInvitation	Cancels invitation.

`TRTCChatSalonDelegate` API Overview

Common event callback APIs

API	Description
onError	Error

API	Description
onWarning	Warning
onDebugLog	Log

Room event callback APIs

API	Description
onRoomDestroy	Room termination
onRoomInfoChange	Room information change
onUserVolumeUpdate	User volume

Speaker list change callback APIs

API	Description
onAnchorEnterSeat	Someone became a speaker after requesting or being invited by the room owner.
onAnchorLeaveSeat	Someone became a listener or was moved to listeners by the room owner.
onSeatMute	The room owner muted a speaker.

Callback APIs for room entry/exit by listeners

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

Message event callback APIs

API	Description
onRecvRoomTextMsg	Receipt of text message
onRecvRoomCustomMsg	Receipt of custom message

Signaling event callback APIs

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

Basic SDK APIs

sharedInstance

This API is used to get a TRTCChatSalon singleton object.

```
public static synchronized TRTCChatSalon sharedInstance(Context context);
```

The parameters are as detailed below:

Parameter	Type	Description
context	Context	Android context, which will be converted to `ApplicationContext` for the calling of system APIs

destroySharedInstance

This API is used to terminate a TRTCChatSalon singleton object.

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

```
public static void destroySharedInstance();
```

setDelegate

This API is used to set the event callbacks of TRTCChatSalon. You can use `TRTCChatSalonDelegate` to get different status notifications of TRTCChatSalon.

```
public abstract void setDelegate(TRTCChatSalonDelegate delegate);
```

> Note：
> `setDelegate` is the delegate callback of `TRTCChatSalon` .

setDelegateHandler

This API is used to set the thread where the event callback is.

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

The parameters are as detailed below:

Parameter	Type	Description
handler	Handler	Status notifications in `TRTCChatSalon` will be sent to the handler thread you specify.

login

This API is used to log in.

```
public abstract void login(int sdkAppId,
String userId, String userSig,
TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	int	You can view the `SDKAppID` via **Application Management** > **Application Info** in the TRTC console.
userId	String	ID of 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 more information on how to get it, please see UserSig.
callback	ActionCallback	Callback for login. The `code` will be 0 if login succeeds.

logout

This API is used to log out.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for logout. The code is 0 if logout succeeds.

setSelfProfile

This API is used to set profile.

```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCChatSalonCallback.Acti
onCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userName	String	Nickname
`avatar`	`String`	Profile photo address
callback	ActionCallback	Callback for profile setting. The code is 0 if the operation succeeds.

Room APIs

createRoom

This API is used to create a room (called by room owner).

```
public abstract void createRoom(int roomId, TRTCChatSalonDef.RoomParam roomParam, TRTCChatSalonCa
llback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
roomId	int	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a chat salon room list. Currently, Tencent Cloud does not provide management services for chat salon room lists. Please manage the list by yourself.
roomParam	RoomParam	Room information, such as room name, speaker list information, and cover information
callback	ActionCallback	Callback for room creation result. The code is 0 if the operation succeeds.

The process of creating a chat salon and becoming a speaker is as follows:

1. A user calls `createRoom` to create a chat salon, passing in room attributes (e.g., room ID and whether listeners require room owner’s consent to speak).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user receives an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

destroyRoom

This API is used to terminate a room (called by room owner).

```
public abstract void destroyRoom(TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

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

enterRoom

This API is used to enter a room (called by listener).

```
public abstract void enterRoom(int roomId, TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description

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

The process of entering a room as a listener is as follows:

1. A user gets the latest chat salon list from your server. The list may contain the `roomId` and room information of multiple chat salons.
2. The user selects a chat salon, and calls `enterRoom` with the room ID passed in to enter.
3. After entering the room, the user receives an `onRoomInfoChange` notification about room attribute change from the component. The attributes can be recorded, and corresponding changes can be made to the UI, including room name, whether room owner’s consent is required for listeners to speak, etc.
4. The user receives an `onAnchorEnterSeat` notification that someone became a speaker.

exitRoom()

This API is used to exit a room.

```
public abstract void exitRoom(TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

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

getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()` .

> Note :
> If both the room list and room information are managed on your server, you can ignore this parameter.

```
public abstract void getRoomInfoList(List<Integer> roomIdList, TRTCChatSalonCallback.RoomInfoCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomIdList	List&dxlt;Integer&dxgt;	Room ID list
callback	RoomInfoCallback	Callback for room details

getUserInfoList

This API is used to get the user information of a specified `userId`.

```
public abstract void getUserInfoList(List<String> userIdList, TRTCChatSalonCallback.UserListCallback userlistcallback);
```

The parameters are as detailed below:

Parameter	Type	Description
userIdList	List	IDs of the users to query. If this parameter is `null`, the information of all users in the room is obtained.
userlistcallback	UserListCallback	Callback for user details

Mic APIs

enterSeat

This API is used to become a speaker (called by room owner or listener).

> Note :
> After a user becomes a speaker, all members in the room will receive an `onAnchorEnterSeat` notification.

```
public abstract void enterSeat(TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for operation

Calling this API will immediately modify the speaker list. In cases where listeners need the room owner’s consent to speak, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept` , call `enterSeat` .

leaveSeat

This API is used to become a listener (called by speaker).

> Note :
> After a speaker becomes a listener, all members in the room will receive an `onAnchorLeaveSeat` notification..

```
public abstract void leaveSeat(TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for operation

pickSeat

This API is used to invite a listener to speak (called by room owner).

> Note :
> After a listener becomes a speaker following the room owner's invitation, all members in the room will receive an `onAnchorEnterSeat` notification.

```
public abstract void pickSeat(String userId, TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userID	String	User ID
callback	ActionCallback	Callback for operation

Calling this API will immediately modify the speaker list. In cases where the room owner needs listeners' consent to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept` , call `pickSeat` .

kickSeat

This API is used to remove a speaker (called by room owner).

> Note :
> After a speaker is removed, all members in the room will receive an `onAnchorLeaveSeat` notification.

```
public abstract void kickSeat(String userId, TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the speaker to remove
callback	ActionCallback	Callback for operation

Calling this API will immediately modify the speaker list.

Local Audio APIs

startMicrophone

This API is used to enable mic capturing.

```
public abstract void startMicrophone();
```

stopMicrophone

This API is used to stop mic capturing.

```
public abstract void stopMicrophone();
```

setAudioQuality

This API is used to set audio quality.

```
public abstract void setAudioQuality(int quality);
```

The parameters are as detailed below:

Parameter	Type	Description
quality	int	Audio quality. For more information, please see TRTC SDK.

muteLocalAudio

This API is used to mute/unmute the local audio.

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

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	Mutes/Unmutes. For more information, please see TRTC SDK.

setSpeaker

This API is used to turn the speaker on.

```
public abstract void setSpeaker(boolean useSpeaker);
```

The parameters are as detailed below:

Parameter	Type	Description
useSpeaker	boolean	`true` : speaker; `false` : receiver

setAudioCaptureVolume

This API is used to set the mic capturing volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Capturing volume. Value range: 0-100 (default: 100)

setAudioPlayoutVolume

This API is used to set the playback volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Playback volume. Value range: 0-100 (default: 100)

muteRemoteAudio

This API is used to mute/unmute a specified member.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Specified user ID
mute	boolean	`true` : mute; `false` : unmute

muteAllRemoteAudio

This API is used to mute/unmute all members.

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

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
mute	boolean	`true` : mute; `false` : unmute

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

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

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in the room, which is generally used for on-screen comments.

```
public abstract void sendRoomTextMsg(String message, TRTCChatSalonCallback.ActionCallback callbac
k);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
callback	ActionCallback	Callback for sending result

sendRoomCustomMsg

This API is used to send custom text messages.

```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCChatSalonCallback.ActionCa
llback callback);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message
callback	ActionCallback	Callback for sending result

Invitation Signaling APIs

sendInvitation

This API is used to send an invitation.

```
public abstract String sendInvitation(String cmd, String userId, String content, TRTCChatSalonCal
lback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command of business
userId	String	Invitee’s user ID
content	String	Invitation content
callback	ActionCallback	Callback for sending result

Returned value:

Returned Value	Type	Description
inviteId	String	Invitation ID

acceptInvitation

This API is used to accept an invitation.

```
public abstract void acceptInvitation(String id, TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for sending result

rejectInvitation

This API is used to decline an invitation.

```
public abstract void rejectInvitation(String id, TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for sending result

cancelInvitation

This API is used to cancel an invitation.

```
public abstract void cancelInvitation(String id, TRTCChatSalonCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for sending result

`TRTCChatSalonDelegate` Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

> 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 as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Error message

onWarning

Callback for warning.

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

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Warning message

onDebugLog

Callback for log.

```
void onDebugLog(String message);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Log information

Room Event Callback APIs

onRoomDestroy

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.

```
void onRoomDestroy(String roomId);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	String	Room ID

onRoomInfoChange

Callback for successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.

```
void onRoomInfoChange(TRTCChatSalonDef.RoomInfo roomInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
roomInfo	RoomInfo	Room information

onUserVolumeUpdate

Callback of the volume of each member in the room after the volume reminder is enabled.

```
void onUserVolumeUpdate(String userId, int volume);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
volume	int	Volume. Value range: 0-100

Speaker List Callback APIs

onAnchorEnterSeat

Someone became a speaker after requesting or being invited by the room owner.

```
void onAnchorEnterSeat(TRTCChatSalonDef.UserInfo user);
```

The parameters are as detailed below:

Parameter	Type	Description
user	UserInfo	Details of the user who became a speaker

onAnchorLeaveSeat

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

```
void onAnchorLeaveSeat(TRTCChatSalonDef.UserInfo user);
```

The parameters are as detailed below:

Parameter	Type	Description
user	UserInfo	Details of the user who became a listener

onSeatMute

The room owner muted a speaker.

```
void onSeatMute(String userId, boolean isMute);
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the speaker who was muted
isMute	boolean	`true` : muted; `false` : unmuted

Callback APIs for Room Entry/Exit by Listener

onAudienceEnter

A listener entered the room.

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

The parameters are as detailed below:

Parameter	Type	Description
userInfo	UserInfo	Information of the user who entered the room

onAudienceExit

A listener exited the room.

```
void onAudienceExit(TRTCChatSalonDef.UserInfo userInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	UserInfo	Information of the listener who exited the room

Message Event Callback APIs

onRecvRoomTextMsg

A text message was received.

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

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
userInfo	UserInfo	User information of sender

onRecvRoomCustomMsg

A custom message was received.

```
void onRecvRoomCustomMsg(String cmd, String message, TRTCChatSalonDef.UserInfo userInfo);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message
userInfo	UserInfo	User information of sender

Invitation Signaling Event Callbacks

onReceiveNewInvitation

A new invitation was received.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
inviter	String	Inviter’s user ID
cmd	String	Custom command word specified by business
content	String	Content specified by business

onInviteeAccepted

The invitee accepted the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee’s user ID

onInviteeRejected

The invitee declined the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee’s user ID

onInvitationCancelled

The inviter canceled the invitation.

```
void onInvitationCancelled(String id, String inviter);
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
inviter	String	Inviter’s user ID

onInvitationTimeout

The invitation timed out.

```
void onInvitationTimeout(String id);
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID

TRTCChatSalon (Flutter)

Last updated : 2022-02-10 14:19:09

`TRTCChatSalon` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create a chat salon and become a speaker, or enter a salon as a listener.
- The room owner can accept a listener's request to speak as well as remove a speaker.
- A listener can request to speak and become a speaker. A speaker can also become a listener.
- All users can send text messages.

`TRTCChatSalon` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Chat Salon (Flutter).

- TRTC SDK: the TRTC SDK is used as a low-latency audio chat component.
- IM SDK: the `AVChatRoom` feature of the IM SDK is used to implement chat rooms. The attribute APIs of IM are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

`TRTCChatSalon` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates singleton object.
registerListener	Sets event listener.
unRegisterListener	Terminates event listener.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
destroyRoom	Terminates a room (called by room owner).
enterRoom	Enters a room (called by listener).
exitRoom	Exits a room (called by listener).
getRoomInfoList	Gets room list details.
getRoomMemberList	Gets the information of all users in the room.
getArchorInfoList	Gets the list of speakers in the room.
getUserInfoList	Gets the user information of the specified `userId` .

Mic APIs

API	Description
enterMic	Becomes a speaker.
leaveMic	Becomes a listener.
muteMic	Mutes/Unmutes a speaker (called by room owner).
kickMic	Removes a speaker (called by room owner).

Local audio APIs

API	Description
startMicrophone	Enables mic capturing.
stopMicrophone	Stops mic capturing.
muteLocalAudio	Mutes/Unmutes local audio.
setSpeaker	Turns the speaker on.
setAudioCaptureVolume	Sets mic capturing volume.
setAudioPlayoutVolume	Sets playback volume.

Remote audio APIs

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

Background music and audio effect APIs

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

Message sending APIs

API	Description
sendRoomTextMsg	Broadcasts a text message in the room. This API is generally used for on-screen comments.

APIs for sending requests to speak

API	Description
raiseHand	Requests to speak.
agreeToSpeak	Accepts the request to speak (called by room owner).
refuseToSpeak	Rejects the request to speak (called by room owner).

`TRTCChatSalonDelegate` API Overview

Common event callback APIs

API	Description
onError	Error
onWarning	Warning
onKickedOffline	Kicked offline

Room event callback APIs

API	Description
onRoomDestroy	The room was closed.
onAnchorListChange	Speaker list change
onUserVolumeUpdate	User volume

Seat list change callback APIs

API	Description
onAnchorEnterMic	Someone became a speaker after requesting or being invited by the room owner.
onAnchorLeaveMic	Someone became a listener or was moved to listeners by the room owner.
onMicMute	The room owner muted a speaker.

Callback APIs for room entry/exit by listeners

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

Message event callback APIs

API	Description
onRecvRoomTextMsg	A text message was received.

Callback APIs for sending requests to speak

API	Description
onRaiseHand	A listener requested to speak.
onAgreeToSpeak	The room owner accepted the listener’s request to speak.

API	Description
onRefuseToSpeak	The room owner rejected the listener’s request to speak.

Basic SDK APIs

sharedInstance

This API is used to get a `TRTCChatSalon` singleton object.

```
static Future<TRTCChatSalon> sharedInstance()
```

destroySharedInstance

This API is used to terminate a `TRTCChatSalon` singleton object.

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

```
static void destroySharedInstance()
```

registerListener

This API is used to set an event listener.

```
void registerListener(VoiceListenerFunc func)
```

> Note :
> `setDelegate` is the delegate callback of `TRTCChatSalon` .

unRegisterListener

This API is used to remove the component event listener.

```
void unRegisterListener(VoiceListenerFunc func)
```

The parameters are as detailed below:

Parameter	Type	Description
func	VoiceListenerFunc	Status notifications in `TRTCChatSalon` are sent to the function you specify.

login

This API is used to log in.

```
Future<ActionCallback> login(int sdkAppId, String userId, String userSig)
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	int	You can view the `SDKAppID` via **Application Management** > **Application Info** in the TRTC console.
userId	String	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 more information on how to get it, please see UserSig.

logout

This API is used to log out.

```
Future<ActionCallback> logout()
```

setSelfProfile

This API is used to set profile.

```
Future<ActionCallback> setSelfProfile(String userName, String avatarURL)
```

The parameters are as detailed below:

Parameter	Type	Description
userName	String	Nickname
avatarURL	String	Profile photo address

Room APIs

createRoom

This API is used to create a room.

```
Future<ActionCallback> createRoom(int roomId, RoomParam roomParam)
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a chat salon room list. Currently, Tencent Cloud does not provide management services for chat salon room lists. Please manage the list by yourself.
roomParam	RoomParam	Room information, such as room name and cover information

The process of creating an audio chat room and becoming a speaker is as follows:

1. A user calls `createRoom` to create a chat salon, passing in room attributes such as room ID.
2. The user receives an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing is enabled automatically.

destroyRoom

This API is used to terminate a room (called by room owner).

```
Future<ActionCallback> destroyRoom()
```

enterRoom

This API is used to enter a room (called by listener).

```
Future<ActionCallback> enterRoom(int roomId)
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID

The process of entering a room as a listener is as follows:

1. A user gets the latest chat salon list from your server. The list may contain the `roomId` and room information of multiple chat salons.
2. Select a chat salon, and call `enterRoom` with the room ID passed in to enter.
3. After room entry, the user can call `getArchorInfoList` to get the speaker list and call `getRoomMemberList` to get the user list. The user list minus the speaker list is the listener list.
4. The user receives an `onAnchorEnterMic` notification that someone became a speaker.

exitRoom

This API is used to exit a room.

```
Future<ActionCallback> exitRoom()
```

getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()` .

> Note :
> You don’t need this API if both the room list and room information are managed on your server.

```
Future<RoomInfoCallback> getRoomInfoList(List<String> roomIdList)
```

The parameters are as detailed below:

Parameter	Type	Description
roomIdList	List<String>	Room ID list

getRoomMemberList

This API is used to get the user list of a room.

> Note :
> By default, the user list includes only the latest 31 users who entered an IM live chat group.

```
Future<MemberListCallback> getRoomMemberList(double nextSeq)
```

The parameters are as detailed below:

Parameter	Type	Description
nextSeq	double	Pulling-by-page flag. It is set to 0 when the information is pulled for the first time. If the callback succeeds and `nextSeq` is not 0, pagination is needed. The value of this field is passed in for the next pull until the value becomes 0.

getArchorInfoList

This API is used to get the list of speakers in the room.

```
Future<UserListCallback> getArchorInfoList()
```

getUserInfoList

This API is used to get the user information of a specified `userId` .

```
Future<UserListCallback> getUserInfoList(List<String> userIdList)
```

The parameters are as detailed below:

Parameter	Type	Description
userIdList	List	IDs of the users to query. If this parameter is `null` , the information of all users in the room is obtained.

Mic APIs

enterMic

This API is used to become a speaker (called by room owner or listener).

> Note :
> After a user becomes a speaker, all members in the room will receive an `onAnchorEnterSeat` notification.

```
Future<ActionCallback> enterMic();
```

Calling this API will immediately modify the speaker list. A listener needs to call `raiseHand` first to send a request to the room owner and call this API after receiving `onAgreeToSpeak` .

leaveMic

A speaker became a listener.

> Note :
> After a speaker becomes a listener, all members in the room will receive an `onAnchorLeaveMic` notification.

```
Future<ActionCallback> leaveMic()
```

muteMic

This API is used to mute/unmute a seat (called by room owner).

> Note :
> After the speaker list changes, all users in the room will receive `onAnchorListChange` and `onMicMute` notifications.

```
Future<ActionCallback> muteMic(bool mute)
```

kickMic

This API is used to remove a speaker (called by room owner).

> Note :
> After the room owner removes a speaker, all users in the room will receive an `onAnchorLeaveMic` notification.

```
Future<ActionCallback> kickMic(String userId)
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the speaker to remove

Calling this API will immediately modify the speaker list.

Local Audio APIs

startMicrophone

This API is used to enable mic capturing.

```
void startMicrophone(int quality)
```

The parameters are as detailed below:

Parameter	Type	Description
quality	int	Audio quality. For more information, please see TRTC SDK.

stopMicrophone

This API is used to stop mic capturing.

```
void stopMicrophone()
```

muteLocalAudio

This API is used to mute/unmute the local audio.

```
void muteLocalAudio(bool mute)
```

The parameters are as detailed below:

Parameter	Type	Description
mute	bool	Mutes/Unmutes. For more information, please see TRTC SDK.

setSpeaker

This API is used to turn the speaker on.

```
void setSpeaker(bool useSpeaker)
```

The parameters are as detailed below:

Parameter	Type	Description
useSpeaker	bool	`true` : speaker; `false` : receiver

setAudioCaptureVolume

This API is used to set the mic capturing volume.

```
void setAudioCaptureVolume(int volume)
```

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Capturing volume. Value range: 0-100 (default: 100)

setAudioPlayoutVolume

This API is used to set the playback volume.

```
void setAudioPlayoutVolume(int volume)
```

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Playback volume. Value range: 0-100 (default: 100)

muteRemoteAudio

This API is used to mute/unmute a specified user.

```
void muteRemoteAudio(String userId, bool mute)
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	Specified user ID

Parameter	Type	Description
mute	bool	`true` : mutes; `false` : unmutes.

muteAllRemoteAudio

This API is used to mute/unmute all users.

```
void muteAllRemoteAudio(bool mute)
```

The parameters are as detailed below:

Parameter	Type	Description
mute	bool	`true` : mutes; `false` : unmutes.

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

```
TXAudioEffectManager getAudioEffectManager()
```

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a room, which is generally used for on-screen comments.

```
Future<ActionCallback> sendRoomTextMsg(String message)
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message.

APIs for sending requests to speak

raiseHand

This API is used to send a request to speak (called by listener).

```
void raiseHand()
```

agreeToSpeak

This API is used to accept a request to speak (called by room owner).

```
Future<ActionCallback> agreeToSpeak(String userId)
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.

refuseToSpeak

This API is used to reject a request to speak (called by room owner).

```
Future<ActionCallback> refuseToSpeak(String userId)
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID.

`TRTCChatSalonDelegate` Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

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.

The parameters are as detailed below:

Parameter	Type	Description
errCode	int	Error code
errMsg	String	Error message
extraInfo	String	Extended field. Certain error codes may carry extra information for troubleshooting.

onWarning

Callback for warning.

The parameters are as detailed below:

Parameter	Type	Description
warningCode	int	Warning code
warningMsg	String	Warning message
extraInfo	String	Extended field. Certain error codes may carry extra information for troubleshooting.

onKickedOffline

Callback for being kicked offline because another user logged in to the same account.

Room Event Callback APIs

onRoomDestroy

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.

onAnchorListChange

Callback for speaker list change.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
mute	bool	Muted or not

onUserVolumeUpdate

Callback of the volume of each member in the room after the volume reminder is enabled.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
volume	int	Volume. Value range: 0-100

Seat Callback APIs

onAnchorEnterMic

Someone became a speaker.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user who became a speaker
userName	String	Nickname
userAvatar	String	Profile photo address
mute	bool	Muted or not. Default: unmuted

onAnchorLeaveMic

A speaker became a listener.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user who became a listener

onMicMute

Whether a user is muted by the room owner.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
mute	bool	Muted or not

Callback APIs for Room Entry/Exit by Listener

onAudienceEnter

A listener entered the room.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user who entered the room
userName	String	Nickname
userAvatar	String	Profile photo address

onAudienceExit

A listener exited the room.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID

Message Event Callback APIs

onRecvRoomTextMsg

A text message was received.

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
sendId	String	Sender’s user ID
userAvatar	String	Sender’s profile photo
userName	String	Sender’s nickname

Callback APIs for sending requests to speak (hand raising)

onRaiseHand

A request to speak was received.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user who requested to speak

onAgreeToSpeak

The room owner accepted the request to speak.

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID of the room owner

onRefuseToSpeak

The room owner rejected the request to speak.

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
userId	String	User ID of the room owner

Real-Time Interactive Teaching

Real-Time Interactive Teaching (Electron)

Last updated : 2022-04-02 16:27:03

Free Demo

Windows macOS

Demonstration

You can install our demo app to experiment with our real-time interactive teaching solution. It offers not only basic capabilities such as audio/video call, screen sharing, whiteboard, and chat, but also advanced features such as mute all, raise hand to speak, invite users to speak, roll call, and sign-in.

Using the Source Code for Real-Time Interactive Teaching

Step 1. Create an application and get the SDKAppID and key

If you already have a TRTC application, you can skip this step and use your application's SDKAppID and key.

1. Log in to the TRTC console, click **Development Assistance** > **Demo Quick Run**, enter your application name, such as `TestTRTC` , and click **Create**.

2. Click **Next**. In the **Modify Configuration** step, note the SDKAppID and key.

Step 2. Configure the IM application

> Note :
> The real-time interactive teaching solution is based on two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. For its billing details, see Pricing.

1. Click **Relevant Cloud Services** in the left sidebar and then click **IM Console > Application List**.

2. Click the application you created.

3. Click **Feature Configuration** > **Login and Message**. Under **Login Settings**, click **Edit**, and set the **Max Login Instances per User per Platform** for web to 2 or larger (the demo app requires login to only 2 IM instances, but you can set this value higher in case you need more in the future).

Step 3. Set up the environment

Node.js and Yarn are required for the source code to run.

1. **Install Node.js:**
   Install Node.js (preferably a version higher than 14.16.0). Run the terminal command below to check the version.

```
node --version
```

2. **Install Yarn:**

- If the Node.js version is lower than 14.16.0, run the terminal command below to install Yarn.

```
npm i -g corepack
```

- If the Node.js version is higher than 14.16.0, run the terminal command below to install Yarn.

```
corepack enable
```

> Note :
> On Windows 10 or 11, if an error occurs because of insufficient permissions, try running the commands as administrator in the Command Prompt.

Step 4. Clone the code

Download a ZIP file of the code. After decompressing the file, you can find the code in `trtc-education-electron` . If you use git to clone the code, run the following terminal command:

```
git clone https://github.com/TencentCloud/trtc-education-electron.git
cd trtc-education-electron
```

Step 5. Obtain the `SDKAppID` and key

1. Find and open `src/main/config/generateUserSig.js` .
2. Set the parameters required to generate UserSig:
   - SDKAPPID: `0` by default. Set it to the `SDKAppID` obtained in Step 1.
   - SECRETKEY: an empty string by default. Set it to the key obtained in Step 1.

> Note :
>
> - In this document, the method to obtain UserSig is to configure a SECRETKEY in the client code. In this method, the SECRETKEY is vulnerable to decompilation and reverse engineering. Once your SECRETKEY is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and feature debugging**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 6. Run the code

Open `trtc-education-electron` in a terminal and run the following command:

```
yarn
yarn start
```

> Note :
>
> - On Windows 10 or 11, when you run the Yarn command for the first time to install dependencies, if an error occurs because of insufficient permissions, run the command as administrator in the Command Prompt first. After that, you can run the command as an

ordinary user in the Command Prompt or a terminal of your code editor such as Visual Studio Code or WebStorm.
- If your download of the Electron code is slow or stuck, you can contact us for help.

Step 7. Create an installer and run it

Open `trtc-education-electron` in a terminal and run the command below to create an installer. After it is created, find the installer in `trtc-education-electron/build/release` , and then run it.

```
yarn package
```

Note :
You need macOS to create a macOS installer and Windows to create a Windows installer.

Technical Support

If you have other questions, you can fill out a contact form or email colleenyu@tencent.com.

Learn More

- SDK API Guide
- Release Notes (Electron)
- Simple Demo Source Code
- API Example Source Code
- FAQs

trtc-electron-education APIs

Last updated : 2022-02-11 16:18:24

The `trtc-electron-education` component is a secondary encapsulation of TRTC and IM capabilities used in real-time interactive teaching scenarios. In addition to basic audio/video chat and screen sharing capabilities, it also offers various teaching features such as Q&A, raise hand, invite, and end answering for online education scenarios.
`trtc-electron-education` is an open-source npm component depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Real-Time Interactive Teaching (Electron).

- TRTC SDK: the TRTC SDK is used as the low-latency audio/video call component.
- IM SDK: the IM SDK is used to send and process signaling messages.

Component Integration

```
// Import by using YARN
yarn add trtc-electron-education
// Import by using npm
npm i trtc-electron-education --save
```

Component Parameters

The required key parameters are as detailed below:

Parameter	Type	Description
sdkAppId	number	`SDKAppID` , which is required and can be viewed in the TRTC Console.
userID	string	User ID, which is required and can be specified by your account system.
userSig	string	Identity signature, which is required and acts as the login password. It can be calculated based on the `userID` . For more information on the calculation method, please see How to Calculate UserSig.

Initialization Sample

```
import TrtcElectronEducation from 'trtc-electron-education';
const rtcClient = new TrtcElectronEducation({
sdkAppId: 1400***803,
userID: '123'
userSig: 'eJwtzM9****-reWMQw_'
});
```

Component Overview

Basic APIs

on(EventCode, handler, context)

This API is used to listen on the events delivered by the component.
Parameters:

Parameter Name	Type	Description
EventCode	String	Event code
handler	Function	Listener function
context	Object	Current execution context

Sample code:

```
const EVENT = rtcClient.EVENT
rtcClient.on(EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

off(EventCode, handler)

This API is used to cancel listening on events.
Parameters:

Parameter Name	Type	Description
EventCode	String	Event code
handler	Function	Named listener function to be canceled

Sample code:

```
const EVENT = rtcClient.EVENT
rtcClient.off(EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

createRoom(params: CreateRoomParams)

This API is used by the teacher to create a classroom.

Parameters:

Parameter Name	Type	Description
classId	number	Classroom ID
nickName	string	Nickname
avatar	string	Profile photo address, which is optional

Sample code:

```
interface CreateRoomParams {
classId: number; // Classroom ID
nickName: string; // Nickname
avatar?:string; // Profile photo address
}
rtcClient.createRoom(params).then(() => {
})
```

destroyRoom(classId: number)

This API is used by the teacher to exit and terminate a classroom.

Parameters:

Parameter Name	Type	Description
classId	number	Classroom ID

Sample code:

```
rtcClient.destroyRoom(classId)
```

enterRoom(params: EnterRoomParams)

This API is used by the teacher to start teaching or by the student to enter the classroom and prepare to listen.

Parameters:

Parameter Name	Type	Description
classId	number	Classroom ID
nickName	string	Nickname
role	string	Role. Valid values: teacher, student
avatar	string	Profile photo address, which is optional

Sample code:

```
interface EnterRoomParams {
role: string; // Role
classId: number; // Classroom ID
nickName?: string; // Nickname
avatar?:string; // Profile photo address
}
rtcClient.enterRoom(params).then(() => {
})
```

exitRoom(role:string, classId: number)

This API is used by the teacher to end teaching or by the student to exit the classroom.
Parameters:

Parameter Name	Type	Description
classId	number	Classroom ID
role	string	Role. Valid values: teacher, student

Sample code:

```
rtcClient.exitRoom(role, classId);
```

Raise hand APIs

#### startQuestionTime(classId: number) This API is used by the teacher to start Q&A. The teacher broadcasts a notification, and the student receives the Q&A start event and can raise hand.
Parameters:

Parameter Name	Type	Description
classId	number	Classroom ID

Sample code:

```
rtcClient.startQuestionTime(classId)
```

raiseHand()

This API is used by the student to raise hand. The student sends a raise hand notification, which will be received by the teacher.

Parameters: none

Sample code:

```
rtcClient.raiseHand()
```

inviteToPlatform(userID: string)

This API is used by the teacher to invite a student to answer a question. The teacher selects the `userID` of a student in the "raise hand" list and sends an invitation notification. The invited student will receive the invitation event and mic on. If no student raises hand, the teacher can directly select a student. The selected student will receive the answering invitation event and mic on.

Parameters:

Parameter Name	Type	Description
userID	string	User ID

Sample code:

```
rtcClient.inviteToPlatform(userID).then(() => {
})
```

finishAnswering(userID: string)

This API is used to end answering. The teacher ends answering by the student. The student receives the answering end notification and exits co-anchoring.

Parameters:

Parameter Name	Type	Description

Parameter Name	Type	Description
userID	string	User ID

Sample code:

```
rtcClient.finishAnswering(userID).then(() => {
})
```

stopQuestionTime(classId: number)

This API is used to stop Q&A. The teacher stops Q&A, and students receive the Q&A stop notification. The co-anchoring student needs to stop co-anchoring and disable the "raise hand" feature.

Parameters:

Parameter Name	Type	Description
classId	number	Classroom ID

Sample code:

```
rtcClient.stopQuestionTime(classId)
```

Push/Pull APIs

getScreenShareList()

This API is used to get the list of windows for screen sharing.

Parameters: none

Sample code:

```
rtcClient.getScreenShareList();
```

startScreenCapture(source: SourceParam)

This API is used to select the shared screen and start push.

Parameters:

Parameter Name	Type	Description
type	number	Capturing source type

Parameter Name	Type	Description
sourceId	string	Capturing source ID. For a window, this field indicates a window handle; for a screen, this field indicates a screen ID
sourceName	string	Capturing source name encoded in UTF-8

Sample code:

```
interface SourceParam {
type: number; // Capturing source type
sourceId: string; // Capturing source ID
sourceName: string; // Capturing source name encoded in UTF-8
}
rtcClient.startScreenCapture({
type,
sourceId,
sourceName
})
```

startRemoteView(params: RemoteParams)

This API is used to start displaying the remote video image or screen sharing image.

Parameters:

Parameter Name	Type	Description
userID	string	User ID
streamType	number	Image type. 1: big image; 2: small image; 3: screen sharing
view	HTMLElement	DOM that carries the displayed image

Sample code:

```
interface RemoteParams {
userID: string; // User ID
streamType: number; // Image type. 1: big image; 2: small image; 3: screen sharing
view: HTMLElement; // DOM that carries the displayed image
}
const view = document.getElementById('localVideo');
```

```
rtcClient.startRemoteView({
userID: userID,
streamType: 1,// 1: big image; 2: small image; 3: screen sharing
view: view
});
```

stopRemoteView(params: StopRemoteParams)

This API is used to stop displaying the remote video image or screen sharing image and pulling the data stream from the remote user.
Parameters:

Parameter Name	Type	Description
userID	string	User ID
streamType	number	Image type. 1: big image; 2: small image; 3: screen sharing

Sample code:

```
interface StopRemoteParams {
userID: string; // User ID
streamType: number; // Image type. 1: big image; 2: small image; 3: screen sharing
}
rtcClient.stopRemoteView({
userID: userID,
streamType: 1 // 1: big image; 2: small image; 3: screen sharing
});
```

Message sending/receiving APIs

sendTextMessage(params: MessageParams)

This API is used to send a message in the chat room.
Parameters:

Parameter Name	Type	Description
classId	number	Classroom ID
message	string	Message text

Sample code:

```
interface MessageParams {
classId: number; // Classroom ID
message: string; // Message text
}
rtcClient.sendTextMessage(params).then(() => {
})
```

sendCustomMessage(userID: string, data: string)

This API is used to send a custom C2C message.

Parameters:

Parameter Name	Type	Description
userID	string	User ID
data	string	Custom message

Sample code:

```
rtcClient.sendCustomMessage(userID, JSON.stringify(params)
```

sendGroupCustomMessage(classId: number, data: string)

This API is used to send a custom group message.

Parameters:

Parameter Name	Type	Description
classId	number	Classroom ID
data	string	Custom message

Sample code:

```
rtcClient.sendGroupCustomMessage(classId, JSON.stringify(params))
```

Device operation APIs

openCamera(view: HTMLElement)

This API is used to enable the camera.

Parameters:

Parameter Name	Type	Description
view	HTMLElement	DOM that carries the displayed image

Sample code:

```
const domEle = document.getElementById(’localVideo’);
rtcClient.openCamera(domEle);
```

closeCamera()

This API is used to disable the camera.

Parameters: none

Sample code:

```
rtcClient.closeCamera();
```

getCameraList()

This API is used to get the list of cameras.

Parameters: none

Sample code:

```
rtcClient.getCameraList()
```

setCurrentCamera(deviceId:string)

This API is used to set the camera. The parameter is a device ID obtained from `getCameraDevicesList` .

Parameters:

Parameter Name	Type	Description
deviceId	string	Device ID

Sample code:

```
rtcClient.setCurrentCamera(deviceId)
```

openMicrophone()

This API is used to enable the mic.

Parameters: none

Sample code:

```
rtcClient.openMicrophone();
```

closeMicrophone()

This API is used to disable the mic.

Parameters: none

Sample code:

```
rtcClient.closeMicrophone();
```

getMicrophoneList()

This API is used to get the list of mics.

Parameters: none

Sample code:

```
rtcClient.getMicrophoneList()
```

setCurrentMicDevice(micId:string)

This API is used to set the mic. The parameter is a device ID obtained from `getMicDevicesList` .

Parameters:

Parameter Name	Type	Description
micId	string	Device ID

Sample code:

```
rtcClient.setCurrentMicDevice(micId)
```

setBeautyStyle(params: BeautyParams)

This API is used to set the effect levels of beauty, brightening, and rosy skin filters.

Parameters:

Parameter Name	Type	Description

Parameter Name	Type	Description
beautyStyle	number	1: smooth, which is suitable for shows since it has more obvious effect 2: natural, which retains more facial details and seems more natural subjectively
beauty	number	Effect level of the beauty filter. Value range: 0–9; 0 indicates that the filter is disabled, and the greater the value, the more obvious the effect
white	number	Effect level of the brightening filter. Value range: 0–9; 0 indicates that the filter is disabled, and the greater the value, the more obvious the effect
ruddiness	number	Effect level of the rosy skin filter. Value range: 0–9; 0 indicates that the filter is disabled, and the greater the value, the more obvious the effect. This parameter does not take effect on Windows currently

Sample code:

```
interface BeautyParams {
beautyStyle: number;// Smooth or natural
beauty: number; // Effect level of beauty filter
white: number; // Effect level of brightening filter
ruddiness: number; // Effect level of rosy skin filter
}
rtcClient.setBeautyStyle({
beautyStyle: 1,
beauty: 5,
white: 5,
ruddiness: 5
})
```

Component Events

Sample code

```
const EVENT = rtcClient.EVENT
rtcClient.on(EVENT.STUDENT_RAISE_HAND, () => {
// A student raises hand
})
```

Detailed events

Code	Description
ENTER_ROOM_SUCCESS	Entered room successfully
LEAVE_ROOM_SUCCESS	Exited room successfully
TEACHER_ENTER	The teacher entered the room
TEACHER_LEAVE	The teacher exited the room
STUDENT_ENTER	The student entered the room
STUDENT_LEAVE	The student exited the room
SCREEN_SHARE_ADD	The teacher started screen sharing
SCREEN_SHARE_REMOVE	The teacher stopped screen sharing
REMOTE_VIDEO_ADD	A remote video stream was added. This notification will be received when a remote user publishes a video stream
REMOTE_VIDEO_REMOVE	A remote video stream was removed. This notification will be received when a remote user cancels video stream release
REMOTE_AUDIO_ADD	A remote audio stream was added
REMOTE_AUDIO_REMOVE	A remote audio stream was removed
ROOM_DESTROYED	The room was terminated
QUESTION_TIME_STARTED	Q&A started
QUESTION_TIME_STOPPED	Q&A ended
STUDENT_RAISE_HAND	A student raised hand
BE_INVITED_TO_PLATFORM	A student was invited to answer
ANSWERING_FINISHED	Answering ended, and audio was muted
MESSAGE_RECEIVED	A message was received
KICKED_OUT	The same account logged in at another place and was kicked out
ERROR	Exception

Code	Description
WARNING	Warning
</span id="startquestiontime">	

Karaoke

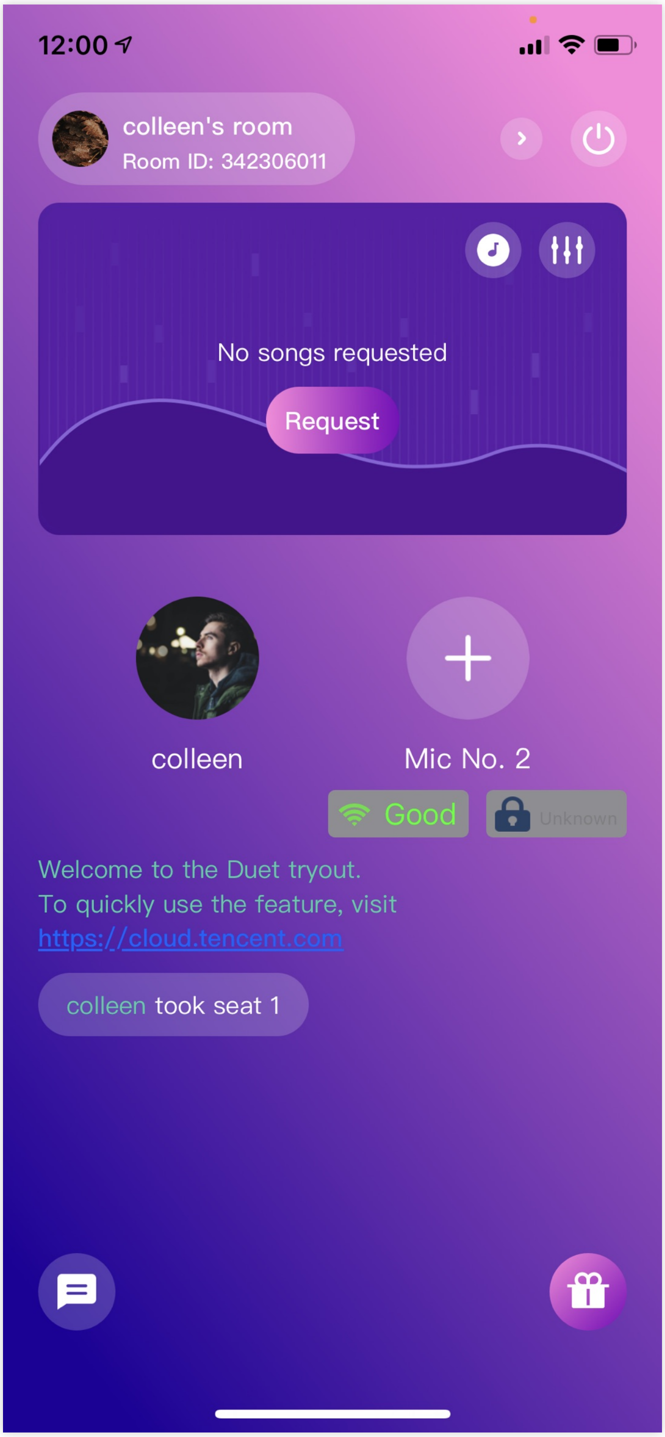
Karaoke (iOS)

Last updated : 2022-03-31 10:56:05

Demo UI

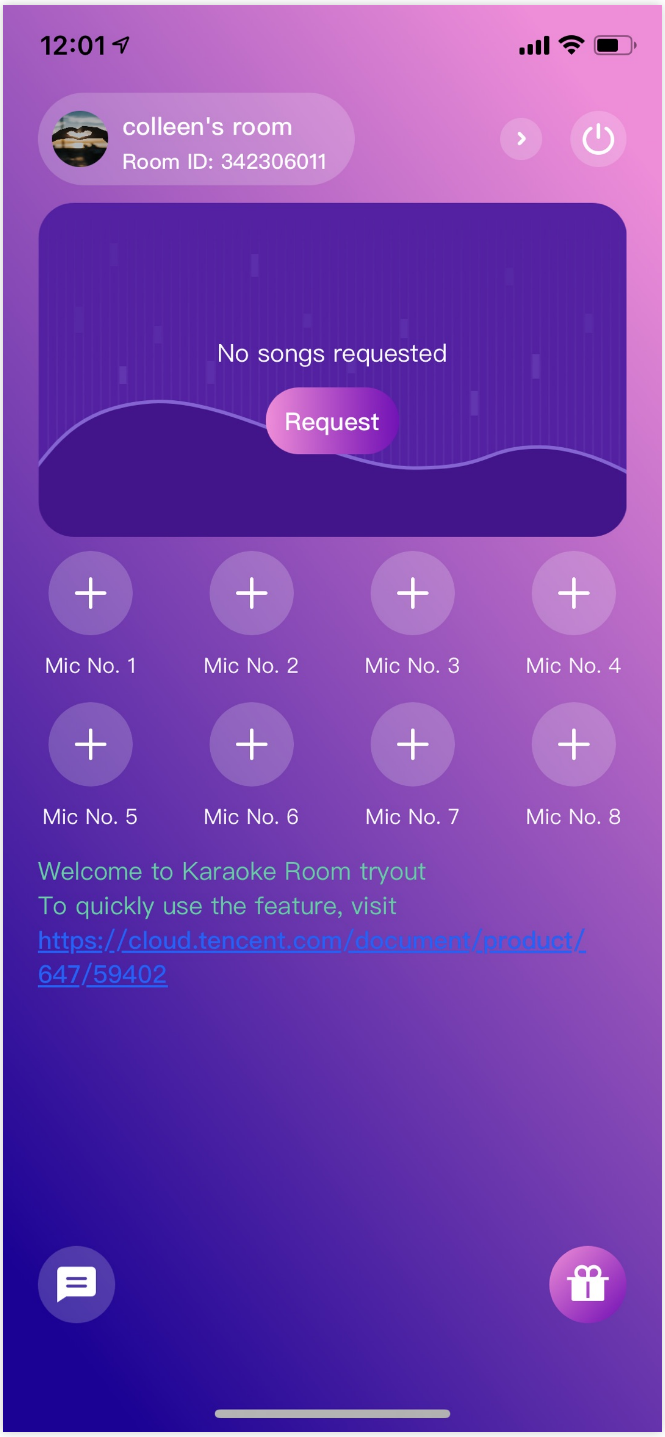
You can download and install the demo we provide to try out TRTC's karaoke features, including low-latency karaoke, seat management, gift sending and receiving, text chat, etc.

Room Owner	Listener



Welcome to the Duet tryout.
To quickly use the feature, visit
https://cloud.tencent.com

colleen took seat 1



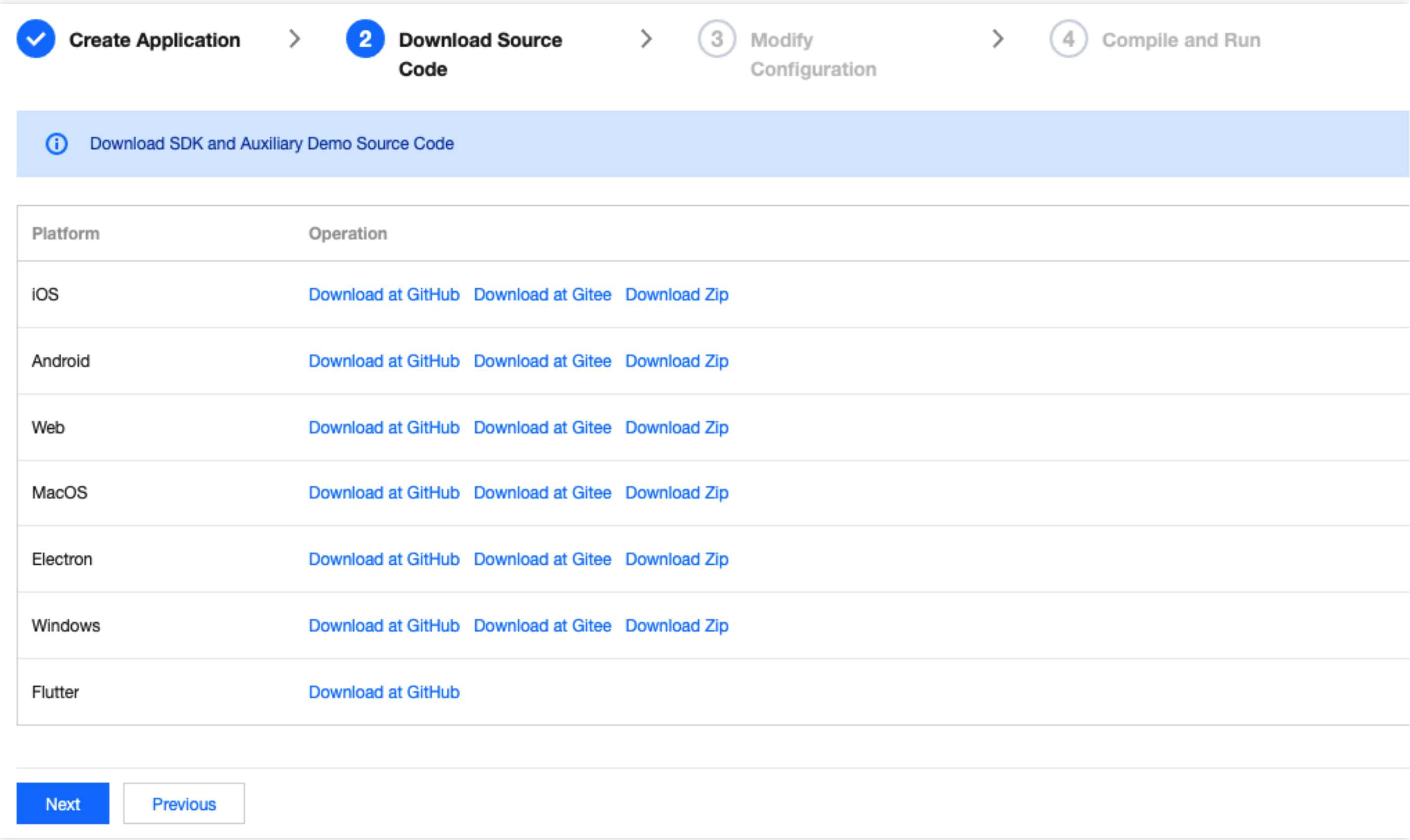
Welcome to Karaoke Room tryout
To quickly use the feature, visit
https://cloud.tencent.com/document/product/647/59402

To quickly enable the karaoke feature, you can modify the demo app we provide and adapt it to your needs. You may also use the `TUIKaraoke` component and customize your own UI.

Using the App’s UI

Step 1. Create an application

1. Log in to the TRTC console and select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestKaraoke` and click **Create**.
3. Click **Next** to skip this step.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

> Note :
> The voice chat room feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Value-added Service Pricing for its billing details.

Step 2. Download the application source code

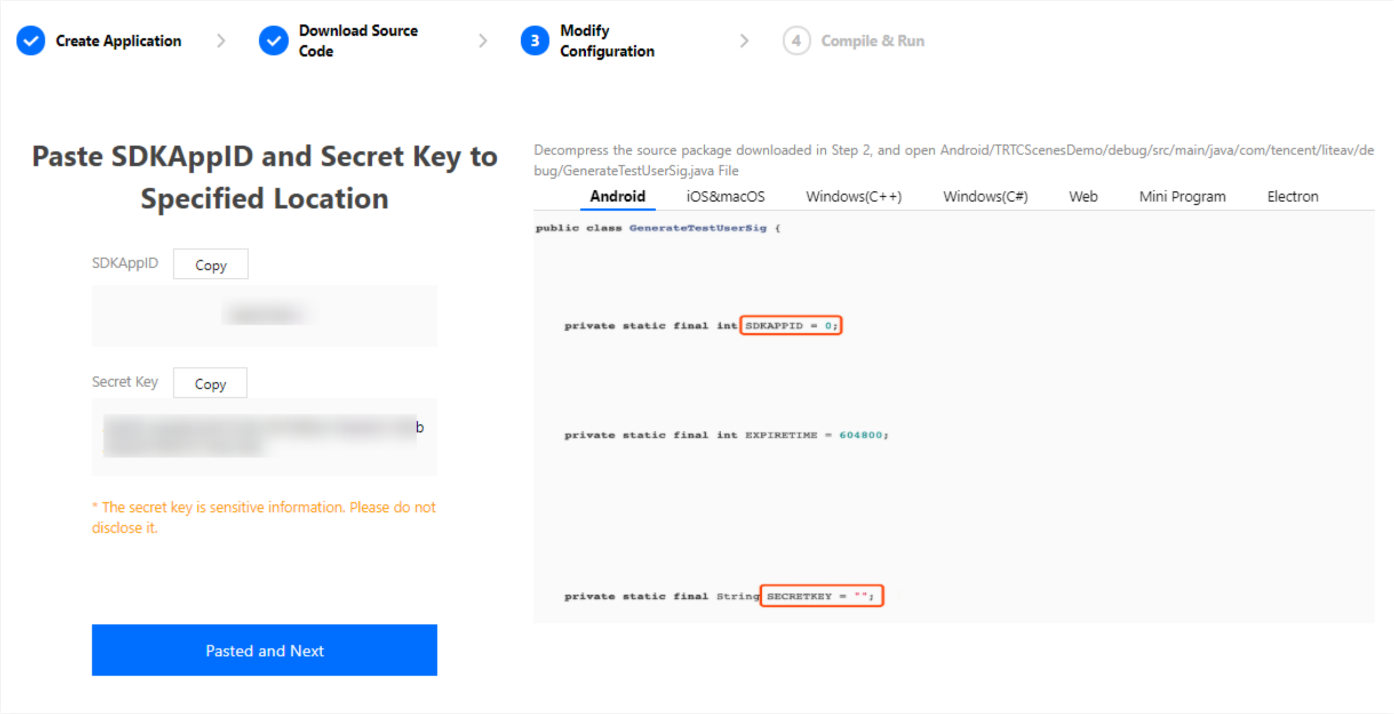
Clone or download the TUIKaraoke source code.

Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.

2. Find and open `TUIKaraoke/Debug/GenerateTestUserSig.swift` .

3. Set the following parameters in `GenerateTestUserSig.swift` :
   - SDKAPPID: `0` by default. Set it to the actual `SDKAppID`.
   - SECRETKEY: left empty by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID: [redacted]

Secret Key: [redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.

5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUIKaraoke/TUIKaraokeApp.xcworkspace` with Xcode (version 11.0 or above) and click **Run**.

Step 5. Modify the app’s source code

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `ui` subfolder contains UI code and UI-related logic. The table below lists the Swift files (folders) and the UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
TRTCKaraokeEnteryControl.swift	The initialization method for all view controllers. You can use the instance to quickly get a `ViewController` object.
TRTCCreateKaraokeViewController	Logic for karaoke room creation page
TRTCKaraokeViewController	Main room views for room owner and listener

Each `TRTC’XXXX’ViewController` folder contains `ViewController`, `RootView`, and `ViewModel`, whose use is described below.

File	Description
ViewController.swift	Page controller, which is responsible for routing pages and binding `RootView` and `ViewModel`
RootView.swift	Layout of all views
ViewModel.swift	View controller, which is responsible for responding to users’ interactions with views and returning response status

Tryout

> Note :
> You need at least two devices to try out the application.

User A

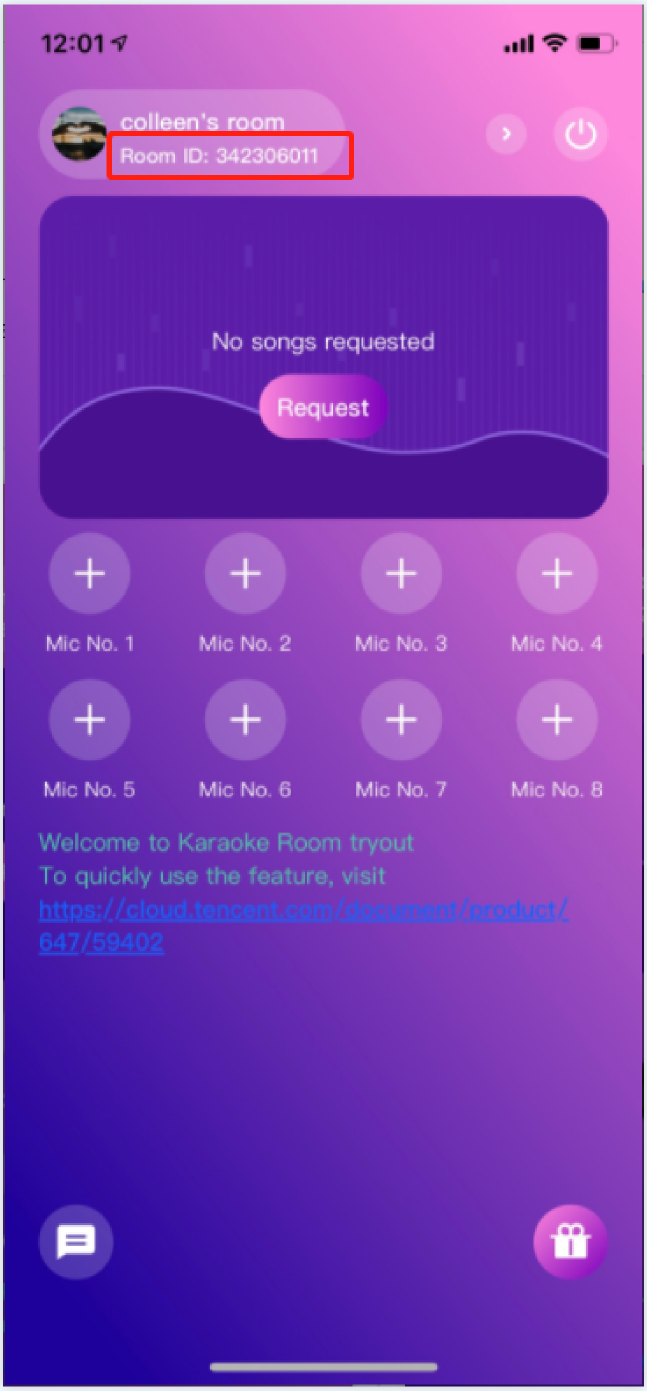
1. Enter a username (**which must be unique**) and log in.
2. Tap **Create Room**.
3. Enter the room subject and tap **Join**.

User B

1. Enter a username (**which must be unique**) and log in.
2. Enter the ID of the room created by user A, and tap **Enter Room**.

Note :

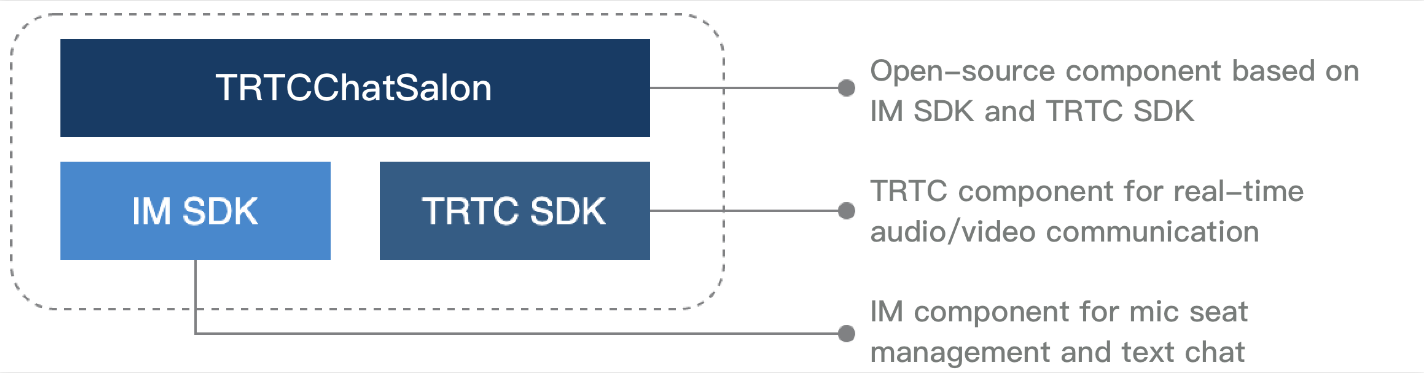
You can find the room ID at the top of user A’s room view.



Welcome to Karaoke Room tryout
To quickly use the feature, visit
https://cloud.tencent.com/document/product/647/59402

Customizing UI

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `model` folder contains the reusable open-source component `TRTCKaraokeRoom`. You can find the component's APIs in `TRTCKaraokeRoom.h` and use them to customize your own UI.


TRTCChatSalon
IM SDK
TRTC SDK
Open–source component based on IM SDK and TRTC SDK
TRTC component for real–time audio/video communication
IM component for mic seat management and text chat

Step 1. Integrate the SDKs

The `TRTCKaraokeRoom` component depends on the TRTC SDK and IM SDK. Follow the steps below to integrate them into your project.

- **Method 1: adding dependencies via CocoaPods**

```
pod 'TXIMSDK_iOS'
pod 'TXLiteAVSDK_TRTC'
```

- **Method 2: adding dependencies through local files**
  If your access to the CocoaPods repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration Documentation
IM SDK	Download	Integration Documentation

Step 2. Configure permission requests

In `info.plist`, add `Privacy > Microphone Usage Description` to request mic access.

Step 3. Import the `TUIKaraoke` component

Import the component using CocoaPods. See below for detailed directions.

1. Copy the `Source`, `Resources`, and `TXAppBasic` folders as well as the `TUIKaraoke.podspec` file in the demo directory to your project directory.
2. Add the following dependencies to your `Podfile` and run `pod install` to complete the import.

```
pod ’TXAppBasic’, :path =&gt; ”TXAppBasic/”
pod ’TXLiteAVSDK_TRTC’
pod ’TUIKaraoke’, :path =&gt; ”./”, :subspecs =&gt; [”TRTC”]
```

Step 4. Create an instance and log in

1. Call the `sharedInstance` class method of `TRTCKaraokeRoom` to create an instance that complies with `TRTCKaraokeRoom’s` protocol, or call the `shared` class method to get a `TRTCKaraokeRoom` instance. There is no difference between the two methods with respect to API usage.
2. Call the `setDelegate` function to register event callbacks of the component.
3. Call the `login` API to log in to the component, and set the key parameters as described below.

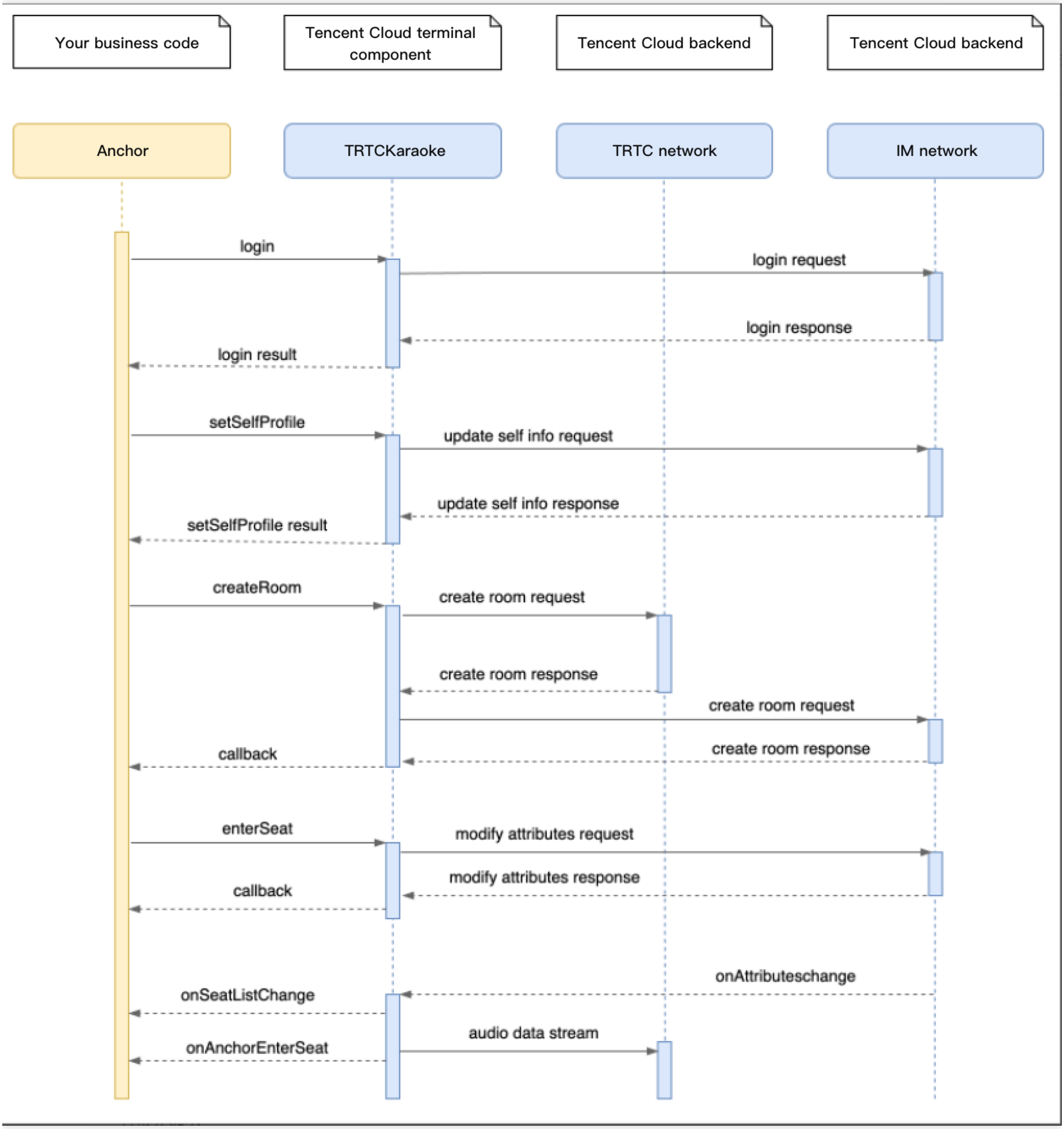
Parameter	Description
SDKAppID	You can view `SDKAppID` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.
userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.
callback	Login callback. The code is `0` if login is successful.

```
// Swift example
// The class responsible for business logic in your code
class YourController {
// Calculate attributes to get a singleton object.
var karaokeRoom: TRTCKaraokeRoom {
return TRTCKaraokeRoom.shared()
}

// Other code logic
......
}
// Set the karaoke proxy
self.karaokeRoom.setDelegate(delegate: karaokeRoomDelegate)

// Below is the calling method. We recommend that you use `weak self` in the closure to preven
```

```
t circular references. The weak self part is not included in the sample code below.
self.karaokeRoom.login(sdkAppId: sdkAppID, userId: userId, userSig: userSig) { [weak self] (co
de, message) in
guard let `self` = self else { return }
// Your callback business logic
}
```

Step 5. Create a room

1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. A user calls `createRoom` to create a karaoke room, passing in room attributes (e.g., room ID, whether listeners require room owner’s consent to speak, number of seats).
3. After creating the room, call `enterSeat` to take a seat.
4. You will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
5. You will also receive an `onAnchorEnterSeat` notification about the occupation of a seat, and mic capturing will be enabled automatically.



Sample code:

```
// 1. Set your nickname and profile photo.
self.karaokeRoom.setSelfProfile(userName: userName, avatarUrl: avatarURL) { (code, message) in
// Callback of the result
}
```

```
// 2. Create a room.
let param = RoomParam.init()
param.roomName = "Room name"
param.needRequest = true // Whether your consent is required for listeners to speak
param.coverUrl = "Cover URL"
param.seatCount = 8 // Number of seats in the room. Set it to `8`.
param.seatInfoList = []

for _ in 0..&lt; param.seatCount {
let seatInfo = SeatInfo.init()
param.seatInfoList.append(seatInfo)
}

self.karaokeRoom.createRoom(roomID: yourRoomID, roomParam: param) { (code, message) in
guard code == 0 else { return }
// Room created successfully

// 3. Take a seat
self.karaokeRoom.enterSeat(seatIndex: 0) { [weak self] (code, message) in
guard let `self` = self else { return }
if code == 0 {
// Seat taken successfully
} else {
// Failed to take a seat
}
}
}

// 4. After taking a seat, you receive an `onSeatListChange` notification
func onSeatListChange(seatInfoList: [SeatInfo]) {
// Refresh your seat list
}

// 5. You receive an `onAnchorEnterSeat` notification
func onAnchorEnterSeat(index: Int, user: UserInfo) {
// Handle the seat taking event
}
```

Step 6. Enter a room as a listener

1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. Get the latest karaoke room list from the backend.

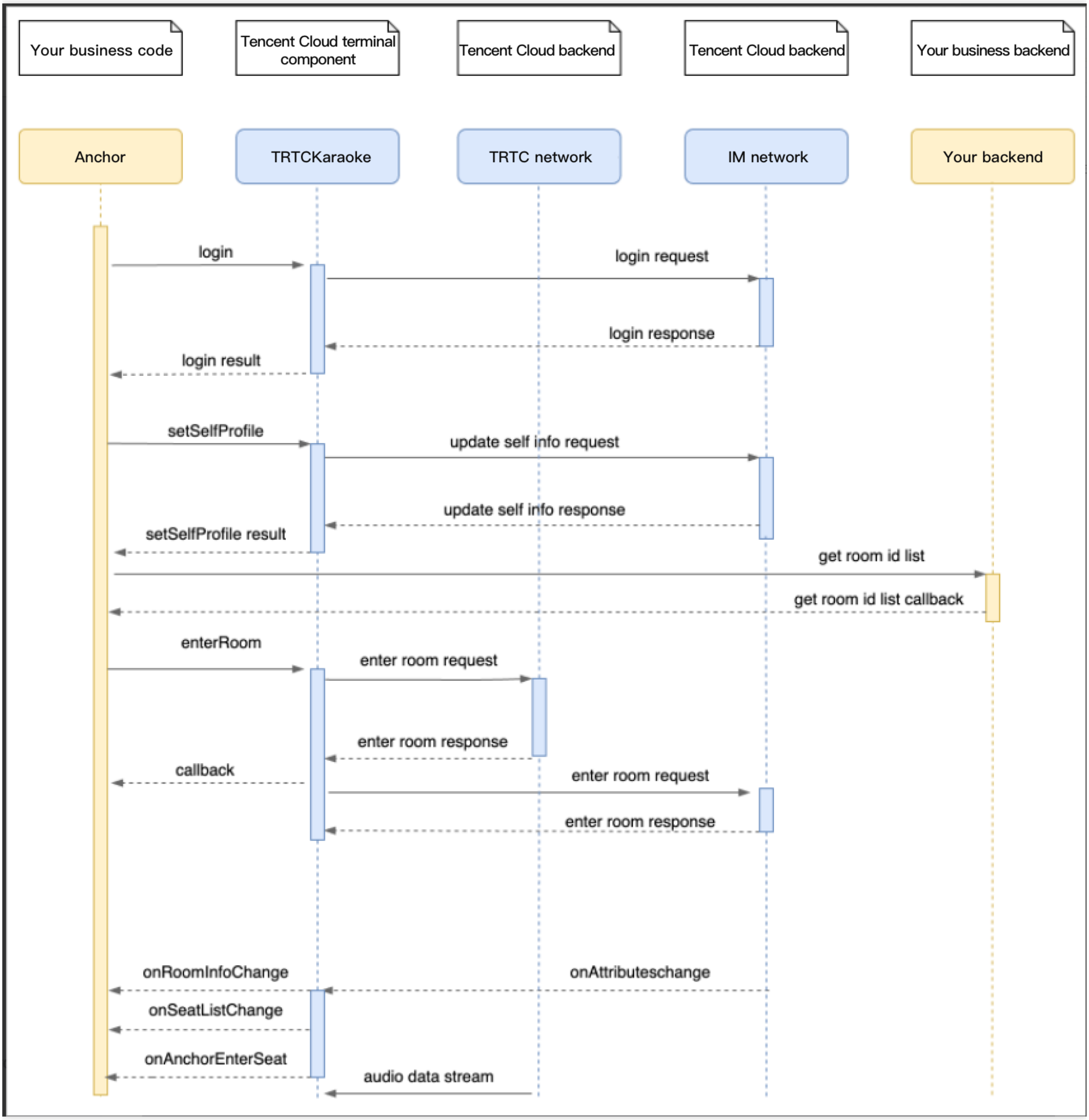
> Note :

> The room list in the demo app is for demonstration only. The business logic of karaoke room lists varies significantly. Tencent Cloud does not provide list management services for the time being. Please manage the list by yourself.

3. Call `getRoomInfoList` to get short descriptions of the karaoke rooms, which are provided by room owners when they call `createRoom` .

> Note :
> If your karaoke room list already contains enough room information, you can skip the step of calling `getRoomInfoList` .

4. Select a karaoke room, and call `enterRoom` with the room ID passed in to enter the room.
5. After entering the room, you will receive an `onRoomInfoChange` notification about room change from the component. Record the room information, including room name, whether the room owner’s consent is required for listeners to speak, etc., and update it to the UI.
6. You will receive an `onSeatListChange` notification about seat change from the component. Update the change to the UI.
7. You will also receive an `onAnchorEnterSeat` notification about the occupation of a seat.



```
// 1. Set your nickname and profile photo.
self.karaokeRoom.setSelfProfile(userName: userName, avatarUrl: avatarURL) { (code, message) in
// Callback of the result
}

// 2. Get the room list from the backend. Suppose it is `roomList`.
let roomList: [Int] = getRoomIDList() // The function you use to get the list of room IDs
```

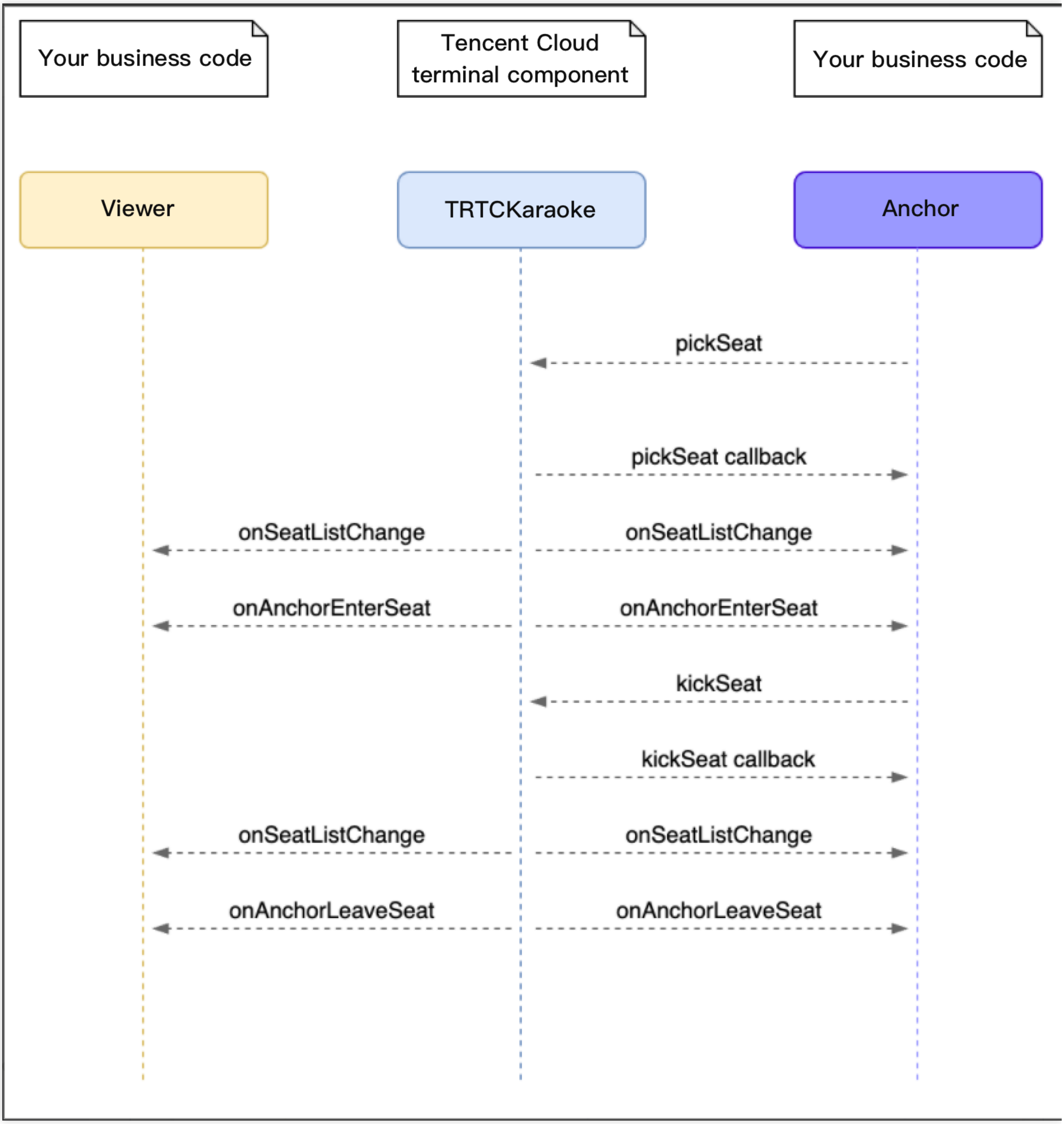
```
// 3. Call `getRoomInfoList` to get details of the rooms.
self.karaokeRoom.getRoomInfoList(roomIdList: roomIdsInt) { (code, message, roomInfos: [RoomInfo])
in
// Refresh the UI after getting the result.
}

// 4. Select a karaoke room and pass in the `roomId` to enter it
self.karaokeRoom.enterRoom(roomID: roomInfo.roomID) { (code, message) in
// Callback of the room entry result
if code == 0 {
// Entered room
}
}

// 5. After successful room entry, you receive an `onRoomInfoChange` notification.
func onRoomInfoChange(roomInfo: RoomInfo) {
// Update the room name and other information.
}

// 6. You receive an `onSeatListChange` notification
func onSeatListChange(seatInfoList: [SeatInfo]) {
// Refresh the seat list
}

// 7. You receive an `onAnchorEnterSeat` notification
func onAnchorEnterSeat(index: Int, user: UserInfo) {
// Handle the mic-on event.
}
```

Step 7. Manage seats

- Room owner
- Listener

1. Call `pickSeat` , passing in a seat number and the `userId` of a listener to place the listener in the seat. All users in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.
2. Call `kickSeat` , passing in a seat number to remove the speaker from the seat. All users in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.
3. Call `muteSeat` , passing in a seat number to mute/unmute the seat. All members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.
4. Call `closeSeat` , passing in a seat number to block/unblock the seat. Listeners cannot take a blocked seat, and all users in the room will receive an `onSeatListChange` notification and an

`onSeatClose` notification.



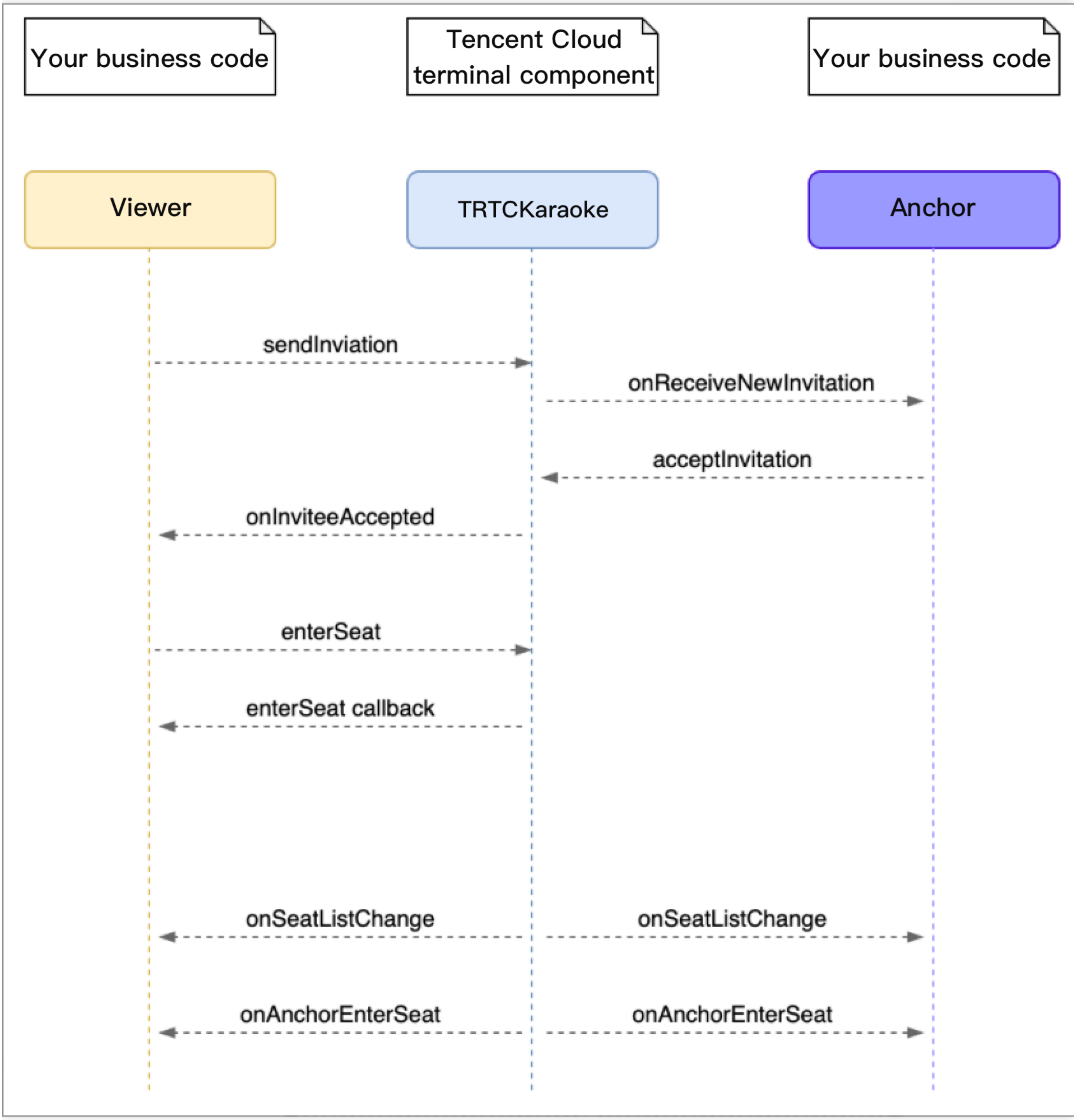
Step 8. Use signaling for invitations

In seat management, listeners can take and leave seats without the room owner’s consent, and the room owner can put listeners in seats without the listeners’ consent.

If you want listeners and room owners to obtain each other's consent before performing the above actions in your application, you can use signaling for invitation sending.

- Listener requesting to take seat
- Room owner inviting listener to take seat

1. A listener calls `sendInvitation` , passing in information including the room owner's `userId` and custom command words. The API will return an `inviteId` , which should be recorded.
2. The room owner receives an `onReceiveNewInvitation` notification, and a window pops up on the UI asking the room owner whether to accept the request.
3. The room owner calls `acceptInvitation` with the `inviteId` passed in to accept the request.
4. The listener receives an `onInviteeAccepted` notification and calls `enterSeat` to become a speaker.



```
// Listener
// 1. Call sendInvitation to request to take seat 1
let inviteId = self.karaokeRoom.sendInvitation(cmd: ”ENTER_SEAT”, userId: ownerUserId, content:
”1”) { (code, message) in
// Callback of the result
}
// 2. Place the user in the seat after the invitation is accepted
```

```
func onInviteeAccepted(identifier: String, invitee: String) {
if identifier == selfID {
self.karaokeRoom.enterSeat(seatIndex: ) { (code, message) in
// Callback of the result
}
}
}

// Room owner
// 1. The room owner receives the request.
func onReceiveNewInvitation(identifier: String, inviter: String, cmd: String, content: String) {
if cmd == ”ENTER_SEAT” {
// 2. The room owner accepts the request.
self.karaokeRoom.acceptInvitation(identifier: identifier, callback: nil)
}
}
```

Step 9. Enable text chat and on-screen comments

- Call `sendRoomTextMsg` to send a common text message. All users in the room will receive an `onRecvRoomTextMsg` callback.

  IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: send text messages
self.karaokeRoom.sendRoomTextMsg(message: message) { (code, message) in


}
// Recipient: listen for text messages
func onRecvRoomTextMsg(message: String, userInfo: UserInfo) {
// Handling of the messages received
}
```

- Call `sendRoomCustomMsg` to send custom (signaling) messages. All users in the room will receive an `onRecvRoomCustomMsg` callback.

  Custom messages are often used to transfer custom signals, e.g., to give and broadcast likes.

```
// For example, a sender can customize commands to distinguish on-screen comments and likes.
// For example, use ”CMD_DANMU” to indicate on-screen comments and ”CMD_LIKE” to indicate likes.
self.karaokeRoom.sendRoomCustomMsg(cmd: ”CMD_DANMU”, message: ”hello world”, callback: nil)
self.karaokeRoom.sendRoomCustomMsg(cmd: ”CMD_LIKE”, message: ””, callback: nil)
// Recipient: listen for custom messages
func onRecvRoomCustomMsg(cmd: String, message: String, userInfo: UserInfo) {
if cmd == ”CMD_DANMU” {
```

```
// An on-screen comment is received.
}
if cmd == ”CMD_LIKE” {
// A like is received.
}
}
```

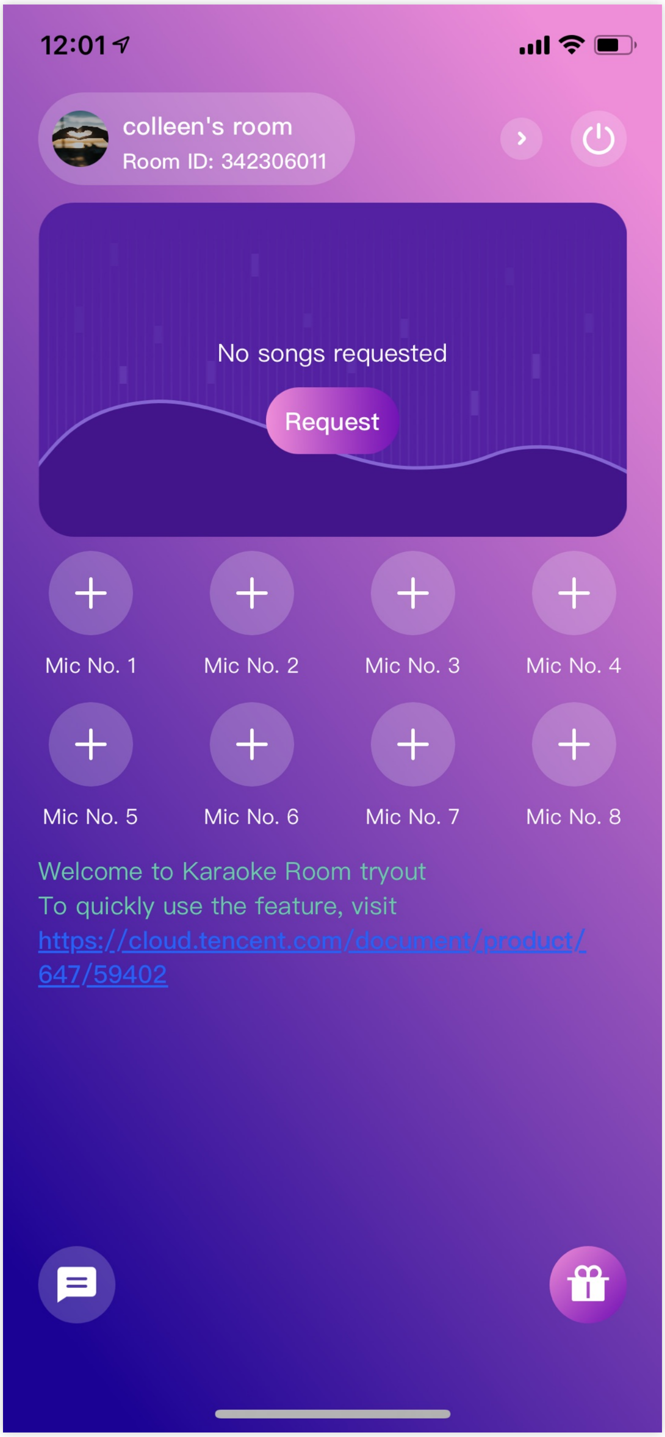
Karaoke (Android)

Last updated : 2022-03-09 17:18:42

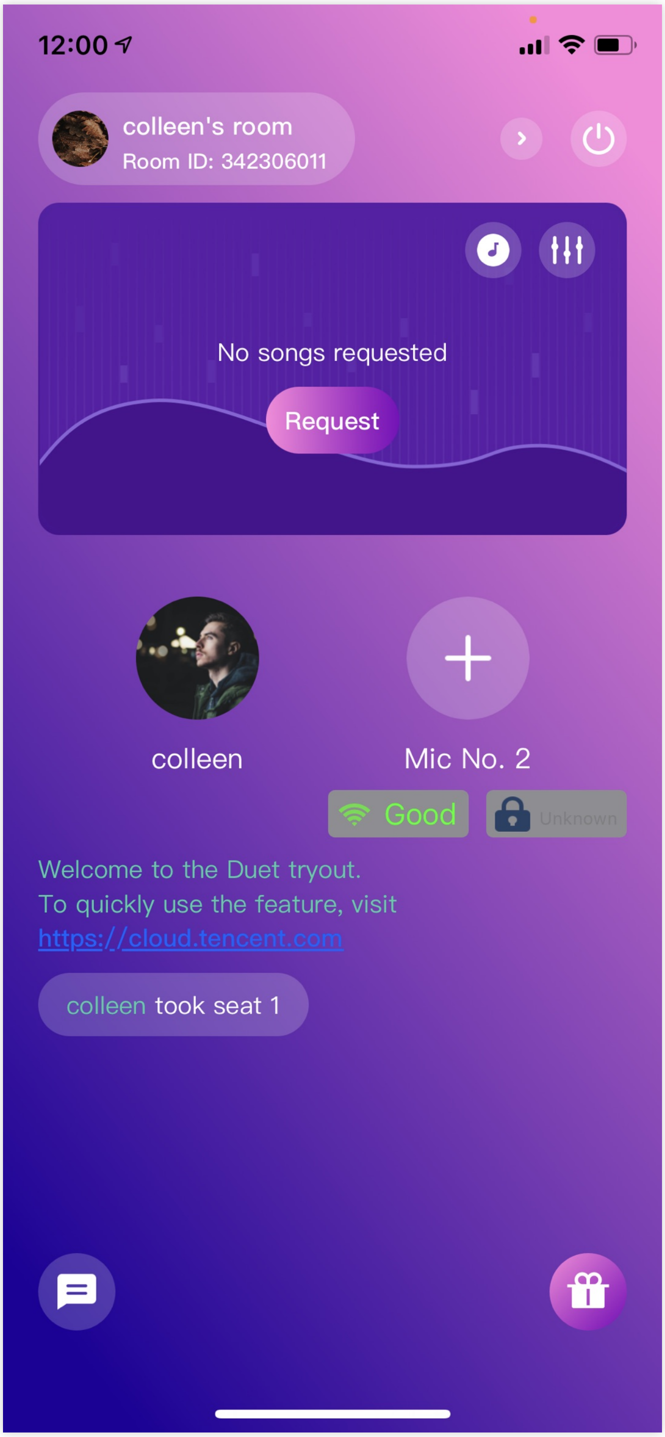
Demo UI

You can download and install the demo we provide to try out TRTC's karaoke features, including low-latency karaoke, seat management, gift sending and receiving, text chat, etc.

Room Owner	Listener



Welcome to Karaoke Room tryout
To quickly use the feature, visit
https://cloud.tencent.com/document/product/647/59402



Welcome to the Duet tryout.
To quickly use the feature, visit
https://cloud.tencent.com

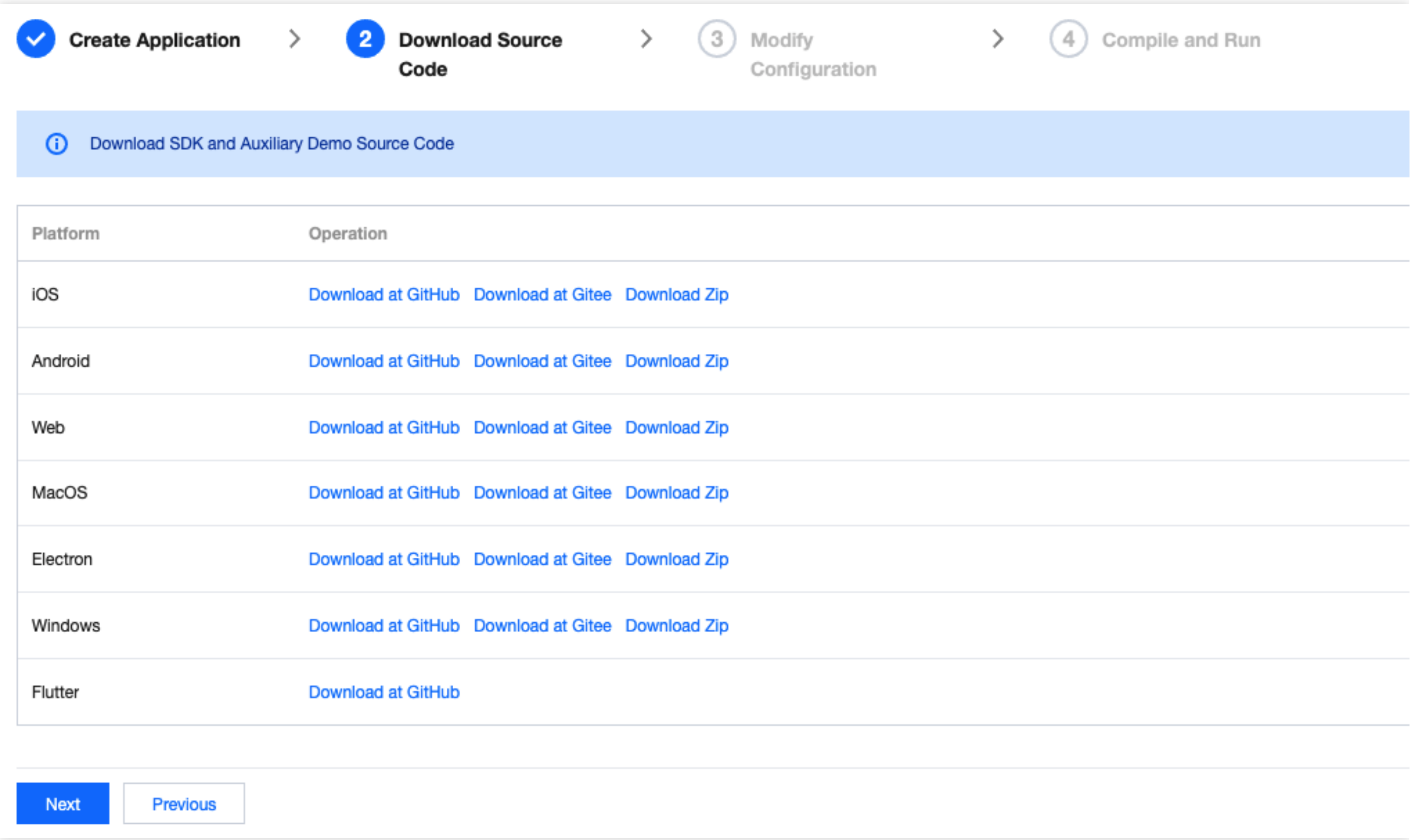
colleen took seat 1

To quickly enable the karaoke feature, you can modify the demo app we provide and adapt it to your needs. You may also use the `TUIKaraoke` component and customize your own UI.

Using the App’s UI

Step 1. Create an application

1. In the TRTC console, select **Development Assistance** > **Demo Quick Run**.
2. Enter an application name such as `TestKaraoke` and click **Create**.
3. Click **Next** to skip this step.



Download SDK and Auxiliary Demo Source Code

Platform	Operation
iOS	Download at GitHub Download at Gitee Download Zip
Android	Download at GitHub Download at Gitee Download Zip
Web	Download at GitHub Download at Gitee Download Zip
MacOS	Download at GitHub Download at Gitee Download Zip
Electron	Download at GitHub Download at Gitee Download Zip
Windows	Download at GitHub Download at Gitee Download Zip
Flutter	Download at GitHub

Note :
This feature uses two basic PaaS services of Tencent Cloud, namely TRTC and IM. When you activate TRTC, IM will be activated automatically. IM is a value-added service. See Pricing for its billing details.

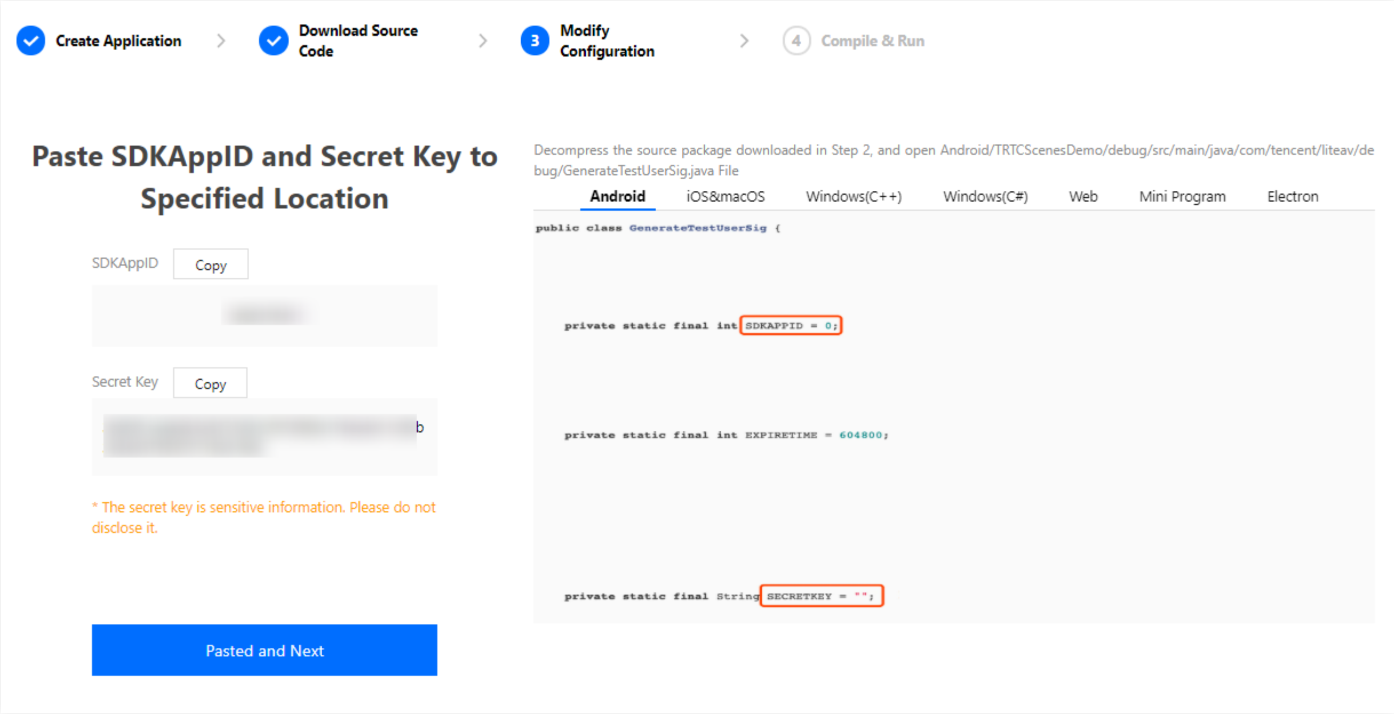
Step 2. Download the application source code

Clone or download the TUIKaraoke source code.

Step 3. Configure application project files

1. In the **Modify Configuration** step, select your platform.
2. Find and open `Android/Debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java` .
3. Set parameters in `GenerateTestUserSig.java` :
   - SDKAPPID: a placeholder by default. Set it to the actual `SDKAppID`.

- SECRETKEY: a placeholder by default. Set it to the actual key.



Paste SDKAppID and Secret Key to Specified Location

SDKAppID Copy

[redacted]

Secret Key Copy

[redacted]

* The secret key is sensitive information. Please do not disclose it.

Decompress the source package downloaded in Step 2, and open Android/TRTCScenesDemo/debug/src/main/java/com/tencent/liteav/debug/GenerateTestUserSig.java File

```
public class GenerateTestUserSig {

    private static final int SDKAPPID = 0;

    private static final int EXPIRETIME = 604800;

    private static final String SECRETKEY = "";
```

4. Click **Next** to complete the creation.
5. After compilation, click **Return to Overview Page**.

> Note :
>
> - The method for generating `UserSig` described in this document involves configuring `SECRETKEY` in client code. In this method, `SECRETKEY` may be easily decompiled and reversed, and if your key is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is suitable only for the local execution and debugging of the app**.
> - The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see How do I calculate UserSig on the server?.

Step 4. Run the application

Open the source code project `TUIKaraoke` with Android Studio (version 3.5 or above) and click **Run**.

Step 5. Modify the app’s source code

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `ui` subfolder contains UI code. The table below lists the files (folders) and the UI views they represent. You can refer to it when making UI changes.

File or Folder	Description
base	Basic classes used by the UI
room	Main room views for room owner and listener
widget	Common controls

Tryout

> Note :
>
> You need at least two devices to try out the application.

User A

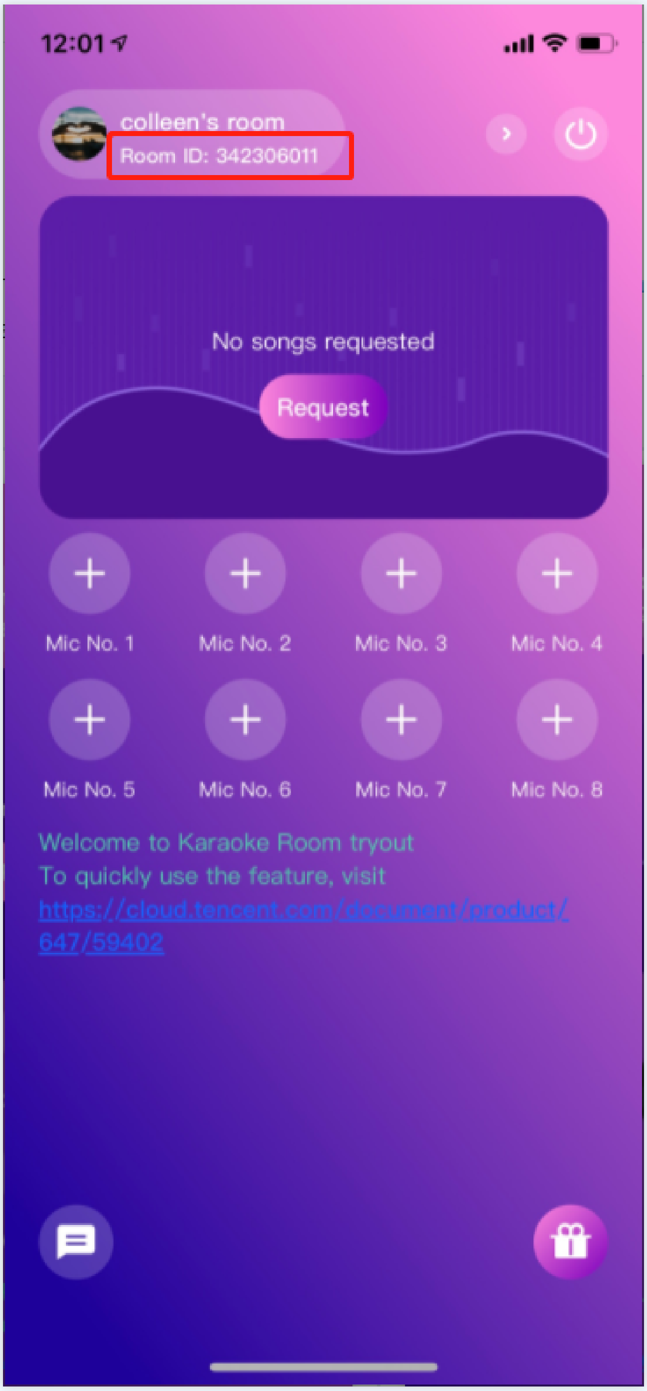
1. Enter a username (**which must be unique**) and log in.
2. Click **Create Room**.
3. Enter the room subject and click **Sing Together**.

User B

1. Enter a username (**which must be unique**) and log in.
2. Enter the ID of the room created by user A, and tap **Join**.

> Note :

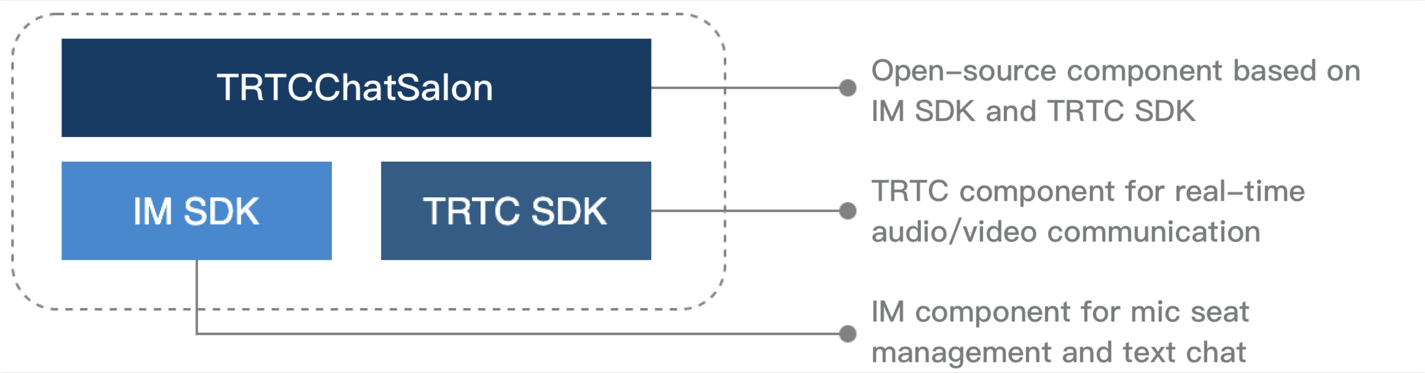
You can find the room ID at the top of user A’s room view.



Welcome to Karaoke Room tryout
To quickly use the feature, visit
https://cloud.tencent.com/document/product/647/59402

Customizing UI

The `Source` folder in the source code contains two subfolders: `ui` and `model`. The `model` subfolder contains the reusable open-source component `TRTCKaraokeRoom`. You can find the component’s APIs in `TRTCKaraokeRoom.java` and use them to customize your own UI.



Open–source component based on IM SDK and TRTC SDK

TRTC component for real–time audio/video communication

IM component for mic seat management and text chat

Step 1. Integrate the SDKs

The `TRTCKaraokeRoom` component depends on the TRTC SDK and IM SDK. Follow the steps below to integrate them into your project.

Method 1: adding dependencies via Maven

1. Add the TRTC SDK and IM SDK dependencies to `dependencies` .

```
dependencies {
complie ”com.tencent.liteav:LiteAVSDK_TRTC:latest.release”
complie ’com.tencent.imsdk:imsdk:latest.release’
compile ’com.google.code.gson:gson:2.3.1’
}
```

2. In `defaultConfig` , specify the CPU architecture to be used by your application.

```
defaultConfig {
ndk {
abiFilters ”armeabi-v7a”
}
}
```

3. Click **Sync Now** to automatically download the SDKs and integrate them into your project.

Method 2: adding dependencies through local AAR files

If your access to the Maven repository is slow, you can download the ZIP files of the SDKs and manually integrate them into your project as instructed in the documents below.

SDK	Download Page	Integration Guide
TRTC SDK	Download	Integration document
IM SDK	Download	Integration document

Step 2. Configure permission requests and obfuscation rules

Configure permission requests for your app in `AndroidManifest.xml` . The SDKs need the following permissions (on Android 6.0 and above, the read storage permission must be requested at runtime.)

```
<uses-permission android:name=”android.permission.INTERNET”>
<uses-permission android:name=”android.permission.ACCESS_NETWORK_STATE”>
<uses-permission android:name=”android.permission.ACCESS_WIFI_STATE”>
<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE”>
<uses-permission android:name=”android.permission.READ_EXTERNAL_STORAGE”>
<uses-permission android:name=”android.permission.RECORD_AUDIO”>
<uses-permission android:name=”android.permission.MODIFY_AUDIO_SETTINGS”>
<uses-permission android:name=”android.permission.BLUETOOTH”>
<uses-permission android:name=”android.permission.READ_PHONE_STATE”>
```

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

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

Step 3. Import the `TRTCKaraoke` component

Copy all the files in the directory below to your project:

```
Source/src/main/java/com/tencent/liteav/tuikaraoke/model
```

Step 4. Create an instance and log in

1. Call the `sharedInstance` API to create an instance of the `TRTCKaraoke` component.
2. Call the `setDelegate` API to register event callbacks of the component.
3. Call the `login` API to log in to the component, and set the key parameters as described below.

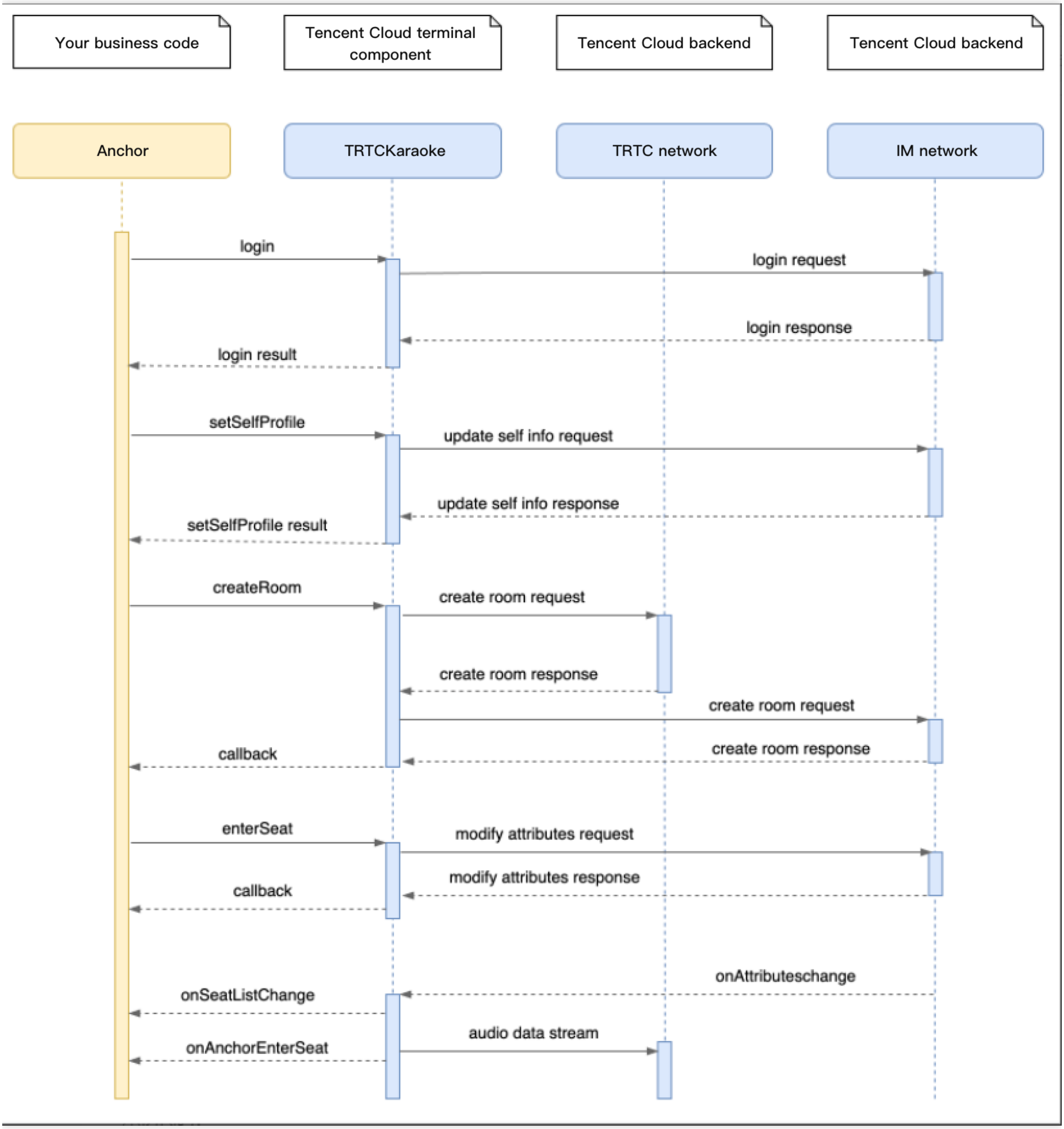
Parameter	Description
SDKAppID	You can view `SDKAppID` in the TRTC console.
userId	ID of the current user, which is a string that can contain letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_). We recommend you set it based on your business account system.

userSig	Tencent Cloud's proprietary security signature. To obtain one, see UserSig.
callback	Login callback. The code is `0` if login is successful.

```
TRTCKaraoke mTRTCKaraokeRoom = TRTCKaraokeRoom.sharedInstance(this);
mTRTCKaraokeRoom.setDelegate(this);
mTRTCKaraokeRoom.login(SDKAPPID, userId, userSig, new TRTCKaraokeRoomCallback.ActionCallback()
{
@Override
public void onCallback(int code, String msg) {
if (code == 0) {![](https://main.qcloudimg.com/raw/7e61af0f3de8302260a1e10d7da6231e.png)
// Logged in
}
}
});
```

Step 5. Create a room and become a speaker

1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. A user calls `createRoom` to create a karaoke room, passing in room attributes (e.g., room ID, whether listeners require room owner’s consent to speak, number of seats).
3. After creating the room, call `enterSeat` to take a seat.
4. You will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
5. You will also receive an `onAnchorEnterSeat` notification about the occupation of a seat, and mic capturing will be enabled automatically.



```
// 1. Set your nickname and profile photo.
mTRTCKaraokeRoom.setSelfProfile("my_name", "my_face_url", null);

// 2. Call `createRoom` to create a room
final TRTCKaraokeRoomDef.RoomParam roomParam = new TRTCKaraokeRoomDef.RoomParam();
roomParam.roomName = "Room name";
roomParam.needRequest = false; // Whether your consent is required for listeners to speak
roomParam.seatCount = 7; // Number of room seats. In this example, the number is 7. 6 seats are a
vailable after you take one.
roomParam.coverUrl = ”URL of room cover image”;
mTRTCKaraokeRoom.createRoom(mRoomId, roomParam, new TRTCKaraokeRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// 3. Take seat 0
mTRTCKaraokeRoom.enterSeat(0, new TRTCKaraokeRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
}
}
});
}
}
});

// 4. After taking a seat, you receive an `onSeatListChange` notification
@Override
public void onSeatListChange(final List<trtckaraokeroomdef.seatinfo> seatInfoList) {
// Display the seat list
}

// 5. You receive an `onAnchorEnterSeat` notification
@Override
public void onAnchorEnterSeat(TRTCKaraokeRoomDef.UserInfo userInfo) {
}
```

Step 6. Enter a room as a listener

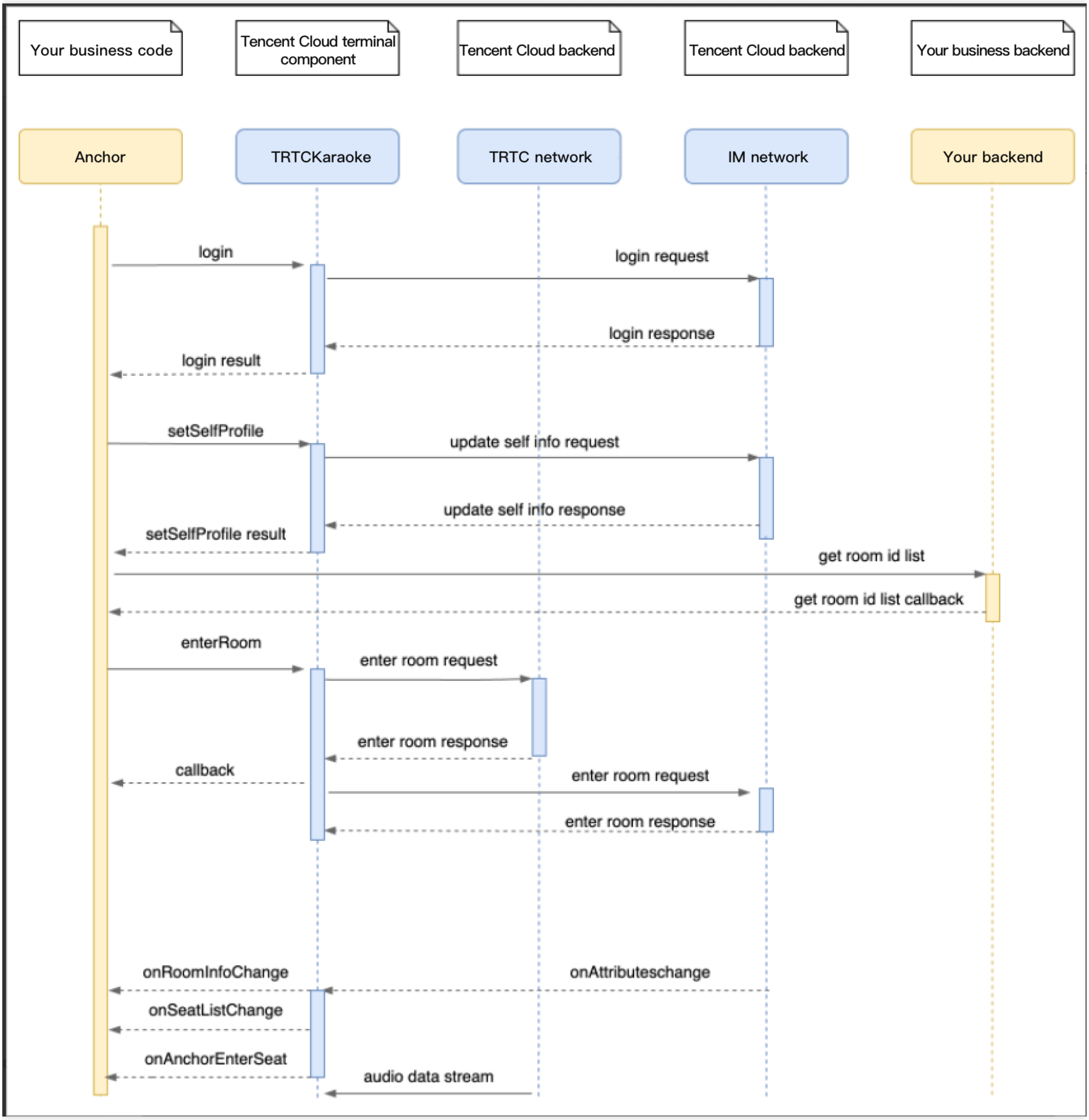
1. After performing step 4 to log in, call `setSelfProfile` to set your nickname and profile photo.
2. Get the latest karaoke room list from the backend.

> Note :
> The room list in the demo app is for demonstration only. The business logic of karaoke room lists varies significantly. Tencent Cloud does not provide list management services for the time being. Please manage the list by yourself.

3. Call `getRoomInfoList` to get short descriptions of the rooms, which are provided by room owners when they call `createRoom` .

> Note :
> If your karaoke room list already contains enough room information, you can skip the step of calling `getRoomInfoList` .

4. Select a karaoke room, and call `enterRoom` with the room ID passed in to enter the room.
5. After entering the room, you will receive an `onRoomInfoChange` notification about room change from the component. Record the room information, including room name, whether the room owner’s consent is required for listeners to speak, etc., and update it to the UI.
6. You will receive an `onSeatListChange` notification about seat change from the component. Update the change to the UI.
7. You will also receive an `onAnchorEnterSeat` notification that someone becomes a speaker.



```
// 1. Set your nickname and profile photo.
mTRTCKaraokeRoom.setSelfProfile("my_name", "my_face_url", null);

// 2. Get the room list from the backend. Suppose it is `roomList`.
List<integer> roomList = GetRoomList();

// 3. Call `getRoomInfoList` to get details of the rooms.
mTRTCKaraokeRoom.getRoomInfoList(roomList, new TRTCKaraokeRoomCallback.RoomInfoCallback() {
```

```
@Override
public void onCallback(int code, String msg, List<trtckaraokeroomdef.roominfo> list) {
if (code == 0) {
// Refresh the room list on your UI
}
}
});

// 4. Select a karaoke room and pass in the `roomId` to enter it
mTRTCKaraokeRoom.enterRoom(roomId, new TRTCKaraokeRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// Entered room successfully
}
}
});

// 5. After successful room entry, you receive an `onRoomInfoChange` notification.
@Override
public void onRoomInfoChange(TRTCKaraokeRoomDef.RoomInfo roomInfo) {
mNeedRequest = roomInfo.needRequest;
mRoomName = roomInfo.roomName;
// The UI can display the title and other information
}

// 6. You receive an `onSeatListChange` notification
@Override
public void onSeatListChange(final List<trtckaraokeroomdef.seatinfo> seatInfoList) {
// Display the seat list
}

// 7. You receive an `onAnchorEnterSeat` notification
@Override
public void onAnchorEnterSeat(TRTCKaraokeRoomDef.UserInfo userInfo) {
}
```

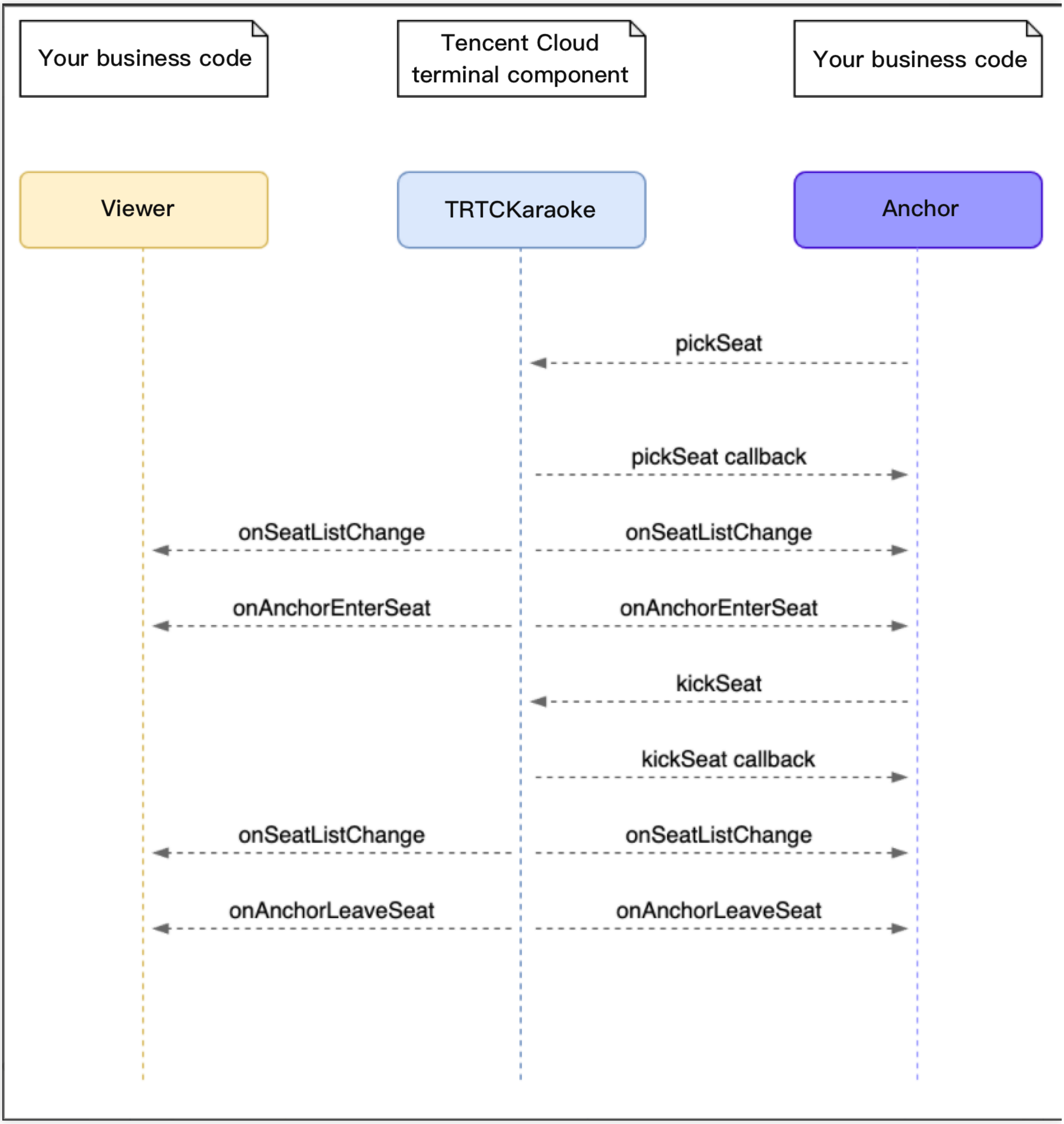
Step 7. Manage seats

- Room owner
- Listener

1. Call `pickSeat` , passing in a seat number and the `userId` of a listener to place the listener in the seat. All users in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

2. Call `kickSeat` , passing in a seat number to remove the speaker from the seat. All users in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.
3. A room owner can mute or unmute a seat by passing in the seat number to `muteSeat` . All members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.
4. Call `closeSeat` , passing in a seat number to block/unblock the seat. Listeners cannot take a blocked seat, and all users in the room will receive an `onSeatListChange` notification and an

`onSeatClose` notification.



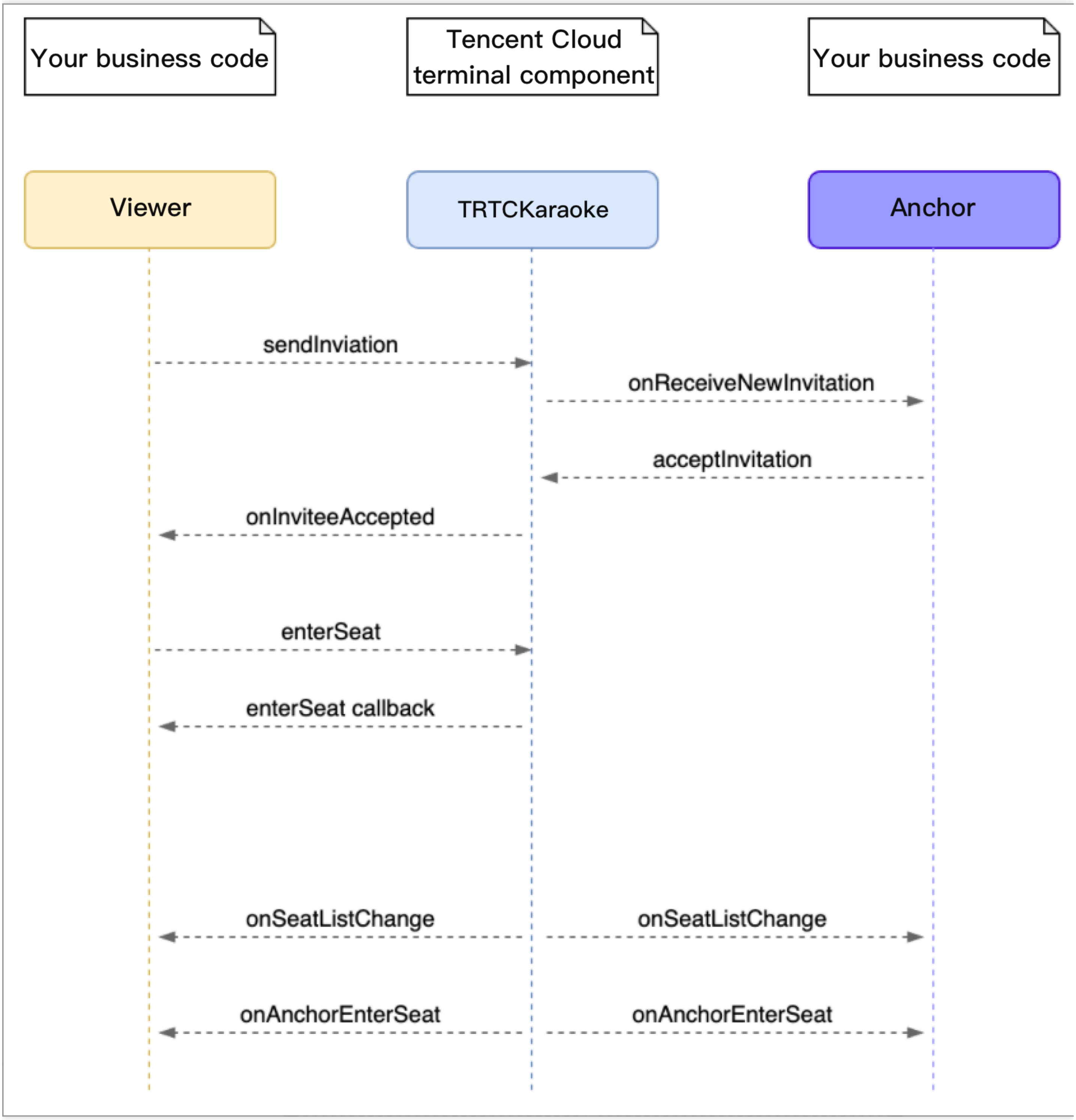
Step 8. Use signaling for invitations

In seat management, listeners can take and leave seats without the room owner’s consent, and the room owner can put listeners in seats without the listeners’ consent.

If you want listeners and room owners to obtain each other's consent before performing the above actions in your application, you can use signaling for invitation sending.

- Listener requesting to take seat
- Room owner inviting listener to take seat

1. A listener calls `sendInvitation` , passing in information including the room owner's `userId` and custom command words. The API will return an `inviteId` , which should be recorded.
2. The room owner receives an `onReceiveNewInvitation` notification, and a window pops up on the UI asking the room owner whether to accept the request.
3. The room owner calls `acceptInvitation` with the `inviteId` passed in to accept the request.
4. The listener receives an `onInviteeAccepted` notification and calls `enterSeat` to become a speaker.



```
// Listener
// 1. Call sendInvitation to request to take seat 1
String inviteId = mTRTCKaraokeRoom.sendInvitation("ENTER_SEAT", ownerUserId, "1", null);

// 2. Place the user in the seat after the invitation is accepted
@Override
public void onInviteeAccepted(String id, String invitee) {
```

```
if(id.equals(inviteId)) {
mTRTCKaraokeRoom.enterSeat(1, null);
}
}

// Room owner
// 1. The room owner receives the request.
@Override
public void onReceiveNewInvitation(final String id, String inviter, String cmd, final String cont
ent) {
if (cmd.equals("ENTER_SEAT")) {
// 2. The room owner accepts the request.
mTRTCKaraokeRoom.acceptInvitation(id, null);
}
}
```

Step 9. Enable text chat and on-screen comments

- Call `sendRoomTextMsg` to send common text messages. All users in the room will receive an `onRecvRoomTextMsg` callback.

  IM has its default content moderation rules. Text messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: send text messages
mTRTCKaraokeRoom.sendRoomTextMsg("Hello Word!", null);
// Recipient: listen for text messages
mTRTCKaraokeRoom.setDelegate(new TRTCKaraokeRoomDelegate() {
@Override
public void onRecvRoomTextMsg(String message, TRTCKaraokeRoomDef.UserInfo userInfo) {
Log.d(TAG, "Received a message from" + userInfo.userName + ": " + message);
}
});
```

- Call `sendRoomCustomMsg` to send custom (signaling) messages. All users in the room will receive an `onRecvRoomCustomMsg` callback.

  Custom messages are often used to transfer custom signals, e.g., to give and broadcast likes.

```
// A sender can customize CMD to distinguish on-screen comments and likes.
// For example, use "CMD_DANMU" to indicate on-screen comments and "CMD_LIKE" to indicate like
s.
mTRTCKaraokeRoom.sendRoomCustomMsg("CMD_DANMU", "Hello world", null);
mTRTCKaraokeRoom.sendRoomCustomMsg("CMD_LIKE", "", null);
// Recipient: listen for custom messages
mTRTCKaraokeRoom.setDelegate(new TRTCKaraokeRoomDelegate() {
@Override
public void onRecvRoomCustomMsg(String cmd, String message, TRTCKaraokeRoomDef.UserInfo userIn
fo) {
if ("CMD_DANMU".equals(cmd)) {
// An on-screen comment is received.
Log.d(TAG, "Received an on-screen comment from" + userInfo.userName + ": " + message);
} else if ("CMD_LIKE".equals(cmd)) {
// A like is received.
Log.d(TAG, userInfo.userName + "liked you.");
}
}
});
```

TRTCKaraoke (iOS)

Last updated : 2021-12-24 16:08:01

`TRTCKaraokeRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create a karaoke room and become a speaker, or enter a karaoke room as a listener.
- The room owner can manage song requests as well as remove a speaker.
- The room owner can also block a seat. Listeners cannot request to take a blocked seat.
- A listener can become a speaker to request songs and sing. A speaker can also become a listener.
- All users can send gifts and text and custom messages. Custom messages can be used to send on-screen comments and give likes.

`TRTCKaraokeRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Karaoke (iOS).

- TRTC SDK: the TRTC SDK is used as a low-latency audio chat component.
- IM SDK: the `AVChatRoom` feature of the IM SDK is used to implement chat rooms. The attribute APIs of IM are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

`TRTCKaraokeRoom` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
setDelegate	Sets event callbacks.
delegateQueue	Sets the thread where event callbacks are.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
destroyRoom	Terminates a room (called by room owner).
enterRoom	Enters a room (called by listener).
exitRoom	Exits a room (called by listener).
getRoomInfoList	Gets room list details.
getUserInfoList	Gets the user information of the specified `userId` . If the value is `nil` , the information of all users in the room is obtained.

Music playback APIs

API	Description
startPlayMusic	Starts music.
stopPlayMusic	Stops music.
pausePlayMusic	Pauses music.
resumePlayMusic	Resumes music.

Seat management APIs

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

Local audio APIs

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

Remote audio APIs

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

Background music and audio effect APIs

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

Message sending APIs

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

Invitation signaling APIs

API	Description

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

`TRTCKaraokeRoomDelegate` API Overview

Common event callback APIs

API	Description
onError	Error
onWarning	Warning
onDebugLog	Log

Room event callback APIs

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

Seat list change callback APIs

API	Description
onSeatListChange	All seat changes
onAnchorEnterSeat	Someone became a speaker or was made a speaker by the room owner.
onAnchorLeaveSeat	Someone became a listener or was moved to listeners by the room owner.

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

Callback APIs for room entry/exit by listener

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

Message event callback APIs

API	Description
onRecvRoomTextMsg	Receipt of a text message
onRecvRoomCustomMsg	Receipt of a custom message

Signaling event callback APIs

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

Song event callback APIs

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

Basic SDK APIs

sharedInstance

This API is used to get a TRTCKaraokeRoom singleton object.

```
/**
* Get a `TRTCKaraokeRoom` singleton object
*
* - returns: `TRTCKaraokeRoom` instance
* - note: To terminate a singleton object, call {@link TRTCKaraokeRoom#destroySharedInstance()}.
*/
+ (instancetype)sharedInstance NS_SWIFT_NAME(shared());
```

destroySharedInstance

This API is used to terminate a TRTCKaraokeRoom singleton object.

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

```
/**
* Terminate the `TRTCKaraokeRoom` singleton object
*
* - note: After the instance is terminated, the externally cached `TRTCKaraokeRoom` instance can no longer be used. You need to call {@link TRTCKaraokeRoom#sharedInstance()} again to get a new instance.
*/
+ (void)destroySharedInstance NS_SWIFT_NAME(destroyShared());
```

setDelegate

This API is used to set the event callbacks of TRTCKaraokeRoom. You can use `TRTCKaraokeRoomDelegate` to get different status notifications of TRTCKaraokeRoom.

```
/**
* Set the event callbacks of the component
*
* You can use `TRTCKaraokeRoomDelegate` to get different status notifications of `TRTCKaraokeRoom`.
```

```
*
* - parameter delegate Callback API
* - note: Callbacks in `TRTCKaraokeRoom` are sent to you in the main queue by default. If you nee
d to specify a queue for event callbacks, please use {@link TRTCKaraokeRoom#setDelegateQueue(queu
e)}.
*/
- (void)setDelegate:(id<TRTCKaraokeRoomDelegate>)delegate NS_SWIFT_NAME(setDelegate(delegate:));
```

Note :
`setDelegate` is the delegate callback of `TRTCKaraokeRoom` .

setDelegateQueue

This API is used to set the thread queue for event callbacks. The main thread (MainQueue) is used by default.

```
/**
* Set the queue for event callbacks
*
* - parameter queue. The status notifications of `TRTCKaraokeRoom` will be sent to the queue you
specify.
*/
- (void)setDelegateQueue:(dispatch_queue_t)queue NS_SWIFT_NAME(setDelegateQueue(queue:));
```

The parameters are as detailed below:

Parameter	Type	Description
queue	dispatch_queue_t	The status notifications of `TRTCKaraokeRoom` are sent to the thread queue you specify.

login

This API is used to log in.

```
- (void)login:(int)sdkAppID
userId:(NSString *)userId
userSig:(NSString *)userSig
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(login(sdkAppID:userId:userSig:callback
:));
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	int	You can view the `SDKAppID` via **Application Management** > **Application Info** in the TRTC console.
userId	String	ID of 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 more information on how to get it, please see UserSig.
callback	ActionCallback	Callback for login. The code is 0 if login succeeds.

logout

This API is used to log out.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for logout. The code is 0 if logout succeeds.

setSelfProfile

This API is used to set the profile.

```
- (void)setSelfProfile:(NSString *)userName avatarURL:(NSString *)avatarURL callback:(ActionCallb
ack _Nullable)callback NS_SWIFT_NAME(setSelfProfile(userName:avatarURL:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userName	String	Username
avatar	String	Profile photo address
callback	ActionCallback	Callback for profile setting. The code is 0 if the operation succeeds.

Room APIs

createRoom

This API is used to create a room (called by room owner).

```
- (void)createRoom:(int)roomID roomParam:(RoomParam *)roomParam callback:(ActionCallback _Nullabl
e)callback NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a karaoke room list. Currently, Tencent Cloud does not provide management services for room lists. Please manage the list on your own.
roomParam	TRTCCreateRoomParam	Room information, such as room name, seat list information, and cover information. To manage seats, you must enter the number of seats in the room.
callback	ActionCallback	Callback for room creation result. The code is 0 if the operation succeeds.

The process of creating a karaoke room and becoming a speaker is as follows:

1. A user calls `createRoom` to create a karaoke room, passing in room attributes (e.g., room ID, whether listeners need room owner’s consent to speak, number of seats).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
4. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

destroyRoom

This API is used to terminate a room (called by room owner).

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

The parameters are as detailed below:

Parameter	Type	Description

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

enterRoom

This API is used to enter a room (called by listener).

```
- (void)enterRoom:(NSInteger)roomID callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(enterRoom(roomID:callback:));
```

The parameters are as detailed below:

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

The process of entering a room as a listener is as follows:

1. A user gets the latest karaoke room list from your server. The list may contain the `roomId` and room information of multiple karaoke rooms.
2. The user selects a room, and calls `enterRoom` with the room ID passed in to enter.
3. After entering the room, the user receives an `onRoomInfoChange` notification about room attribute change from the component. The attributes can be recorded, and corresponding changes can be made to the UI, including room name, whether room owner’s consent is required for listeners to speak, etc.
4. The user will receive an `onSeatListChange` notification about the change of the seat list and can update the change to the UI.
5. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker.

exitRoom

This API is used to exit a room.

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

The parameters are as detailed below:

Parameter	Type	Description

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

getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()`.

> Note :
> You don’t need this API if both the room list and room information are managed on your server.

```
- (void)getRoomInfoList:(NSArray<NSNumber *> *)roomIdList callback:(KaraokeInfoCallback _Nullable)callback NS_SWIFT_NAME(getRoomInfoList(roomIdList:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
roomIdList	List<Integer>	Room ID list
callback	RoomInfoCallback	Callback of room details

getUserInfoList

This API is used to get the user information of a specified `userId`.

```
- (void)getUserInfoList:(NSArray<NSString *> * _Nullable)userIDList callback:(KaraokeUserListCallback _Nullable)callback NS_SWIFT_NAME(getUserInfoList(userIDList:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
userIdList	List	IDs of the users to query. If this parameter is `null`, the information of all users in the room is obtained.
userlistcallback	UserListCallback	Callback of user details

Music Playback APIs

startPlayMusic

This API is used to play music (called after mic-on).

> Note :
>
> - After music playback starts, you will receive an `onMusicPrepareToPlay` notification.
> - During music playback, all members in the room will keep receiving an `onMusicProgressUpdate` notification.
> - After music playback stops, you will receive an `onMusicCompletePlaying` notification.

```
- (void)startPlayMusic:(int32_t)musicID originalUrl:(NSString *)originalUrl accompanyUrl:(NSString *)backingUrl NS_SWIFT_NAME(startPlayMusic(musicID:originalUrl:accompanyUrl:));
```

The parameters are as detailed below:

Parameter	Type	Description
musicID	int32_t	Music ID
originalUrl	String	Absolute path of the vocal track
accompanyUrl	String	Absolute path of the instrumental track

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

stopPlayMusic

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

> Note :
>
> After music playback stops, you will receive an `onMusicCompletePlaying` notification.

```
- (void)stopPlayMusic NS_SWIFT_NAME(stopPlayMusic());
```

pausePlayMusic

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

Note :

- The `onMusicProgressUpdate` notification will be paused.
- No `onMusicCompletePlaying` notification will be received.

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

resumePlayMusic

This API is used to resume music (called after pause).

Note :
No `onMusicPrepareToPlay` notification will be received.

```
- (void)resumePlayMusic NS_SWIFT_NAME(resumePlayMusic());
```

Seat Management APIs

enterSeat

This API is used to become a speaker (called by room owner or listener).

Note :
After a user becomes a speaker, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

```
- (void)enterSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME
(enterSeat(seatIndex:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
seatIndex	int	The number of the seat to take
callback	ActionCallback	Callback for the operation

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

leaveSeat

This API is used to become a listener (called by speaker).

> Note :
> After a speaker becomes a listener, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for the operation

pickSeat

This API is used to place a user in a seat (called by room owner).

> Note :
> After the room owner places a user in a seat, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

```
- (void)pickSeat:(NSInteger)seatIndex userId:(NSString *)userId callback:(ActionCallback _Nullabl
e)callback NS_SWIFT_NAME(pickSeat(seatIndex:userId:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to place the listener in
userId	String	User ID
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. In cases where the room owner needs listeners' consent to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept` , call `pickSeat` .

kickSeat

This API is used to remove a speaker (called by room owner).

Note :
After a speaker is removed, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.

```
- (void)kickSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(k
ickSeat(seatIndex:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to remove the speaker from
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list.

muteSeat

This API is used to mute/unmute a seat (called by room owner).

Note :
After a seat is muted/unmuted, all members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.

```
- (void)muteSeat:(NSInteger)seatIndex isMute:(BOOL)isMute callback:(ActionCallback _Nullable)call
back NS_SWIFT_NAME(muteSeat(seatIndex:isMute:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to mute/unmute
isMute	boolean	`true` : mute; `false` : unmute
callback	ActionCallback	Callback for the operation

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

closeSeat

This API is used to block/unblock a seat (called by room owner).

> Note :
> After a seat is blocked/unblocked, all members in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.

```
- (void)closeSeat:(NSInteger)seatIndex isClose:(BOOL)isClose callback:(ActionCallback _Nullable)c
allback NS_SWIFT_NAME(closeSeat(seatIndex:isClose:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to block/unblock
isClose	boolean	`true` : block; `false` : unblock
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will leave the seat.

Local Audio APIs

startMicrophone

This API is used to start mic capturing.

```
- (void)startMicrophone;
```

stopMicrophone

This API is used to stop mic capturing.

```
- (void)stopMicrophone;
```

setAudioQuality

This API is used to set audio quality.

```
- (void)setAuidoQuality:(NSInteger)quality NS_SWIFT_NAME(setAuidoQuality(quality:));
```

The parameters are as detailed below:

Parameter	Type	Description
quality	int	Audio quality. For more information, please see setAudioQuality().

muteLocalAudio

This API is used to mute/unmute local audio.

```
- (void)muteLocalAudio:(BOOL)mute NS_SWIFT_NAME(muteLocalAudio(mute:));
```

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	Whether to mute or unmute audio. For more information, please see muteLocalAudio().

setSpeaker

This API is used to set whether to use the speaker or receiver.

```
- (void)setSpeaker:(BOOL)userSpeaker NS_SWIFT_NAME(setSpeaker(userSpeaker:));
```

The parameters are as detailed below:

Parameter	Type	Description
useSpeaker	boolean	`true` : speaker; `false` : receiver

setAudioCaptureVolume

This API is used to set the mic capturing volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Capturing volume. Value range: 0-100 (default: 100)

setAudioPlayoutVolume

This API is used to set the playback volume.

```
- (void)setAudioPlayoutVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioPlayoutVolume(volume:));
```

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Playback volume. Value range: 0-100 (default: 100)

muteRemoteAudio

This API is used to mute/unmute a specified user.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
mute	boolean	`true` : mute; `false` : unmute

muteAllRemoteAudio

This API is used to mute/unmute all users.

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

The parameters are as detailed below:

Parameter	Type	Description
isMute	boolean	`true` : mute; `false` : unmute

setVoiceEarMonitorEnable

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

```
- (void)setVoiceEarMonitorEnable:(BOOL)enable NS_SWIFT_NAME(setVoiceEarMonitor(enable:));
```

The parameters are as detailed below:

Parameter	Type	Description
enable	boolean	`true` : enable; `false` : disable

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

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

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a room, which is generally used for on-screen comments.

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

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
callback	ActionCallback	Callback for the operation

sendRoomCustomMsg

This API is used to send a custom text message.

```
- (void)sendRoomCustomMsg:(NSString *)cmd message:(NSString *)message callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendRoomCustomMsg(cmd:message:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message
callback	ActionCallback	Callback for the operation

Invitation Signaling APIs

sendInvitation

This API is used to send an invitation.

```
- (NSString *)sendInvitation:(NSString *)cmd
userId:(NSString *)userId
content:(NSString *)content
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendInvitation(cmd:userId:content:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
cmd	String	Custom command of business
userId	String	Invitee’s user ID
content	String	Invitation content
callback	ActionCallback	Callback for the operation

Returned value:

Returned Value	Type	Description
inviteId	String	Invitation ID

acceptInvitation

This API is used to accept an invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

rejectInvitation

This API is used to decline an invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

cancelInvitation

This API is used to cancel an invitation.

```
- (void)cancelInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)callback NS_SW
IFT_NAME(cancelInvitation(identifier:callback:));
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

`TRTCKaraokeRoomDelegate` Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

> 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*)message
NS_SWIFT_NAME(onError(code:message:));
```

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Error message

onWarning

Callback for warning.

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

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Warning message

onDebugLog

Callback for log.

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

The parameters are as detailed below:

Parameter	Type	Description
message	String	Log information

Room Event Callback APIs

onRoomDestroy

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.

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

The parameters are as detailed below:

Parameter	Type	Description
message	String	Callback information

onRoomInfoChange

Callback for change of room information. This callback is sent after successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.

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

The parameters are as detailed below:

Parameter	Type	Description
roomInfo	RoomInfo	Room information

onUserMicrophoneMute

Callback of whether a user's mic is muted. When a user calls `muteLocalAudio` , all members in the room will receive this callback.

```
- (void)onUserMicrophoneMute:(NSString *)userId mute:(BOOL)mute
NS_SWIFT_NAME(onUserMicrophoneMute(userId:mute:));
```

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
mute	boolean	Volume. Value range: 0-100

onUserVolumeUpdate

Callback of the volume of each member in the room after the volume reminder is enabled.

```
- (void)onUserVolumeUpdate:(NSArray<TRTCVolumeInfo *> *)userVolumes totalVolume:(NSInteger)totalV
olume
NS_SWIFT_NAME(onUserVolumeUpdate(userVolumes:totalVolume:));
```

The parameters are as detailed below:

Parameter	Type	Description
userVolumes	List	List of user volumes
totalVolume	int	Total volume. Value range: 0-100

Seat Callback APIs

onSeatListChange

Callback for all seat changes.

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

The parameters are as detailed below:

Parameter	Type	Description
seatInfoList	List<SeatInfo>	Full seat list

onAnchorEnterSeat

Someone became a speaker or was made a speaker by the owner.

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

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat taken
user	UserInfo	Details of the user who took the seat

onAnchorLeaveSeat

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

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

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat previously occupied by the speaker

Parameter	Type	Description
user	UserInfo	Details of the user who became a listener

onSeatMute

The room owner muted/unmuted a seat.

```
- (void)onSeatMute:(NSInteger)index
isMute:(BOOL)isMute
NS_SWIFT_NAME(onSeatMute(index:isMute:));
```

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat muted/unmuted
isMute	boolean	`true` : muted; `false` : unmuted

onSeatClose

The room owner blocked/unblocked a seat.

```
- (void)onSeatClose:(NSInteger)index
isClose:(BOOL)isClose
NS_SWIFT_NAME(onSeatClose(index:isClose:));
```

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat blocked/unblocked
isClose	boolean	`true` : blocked; `false` : unblocked

Callback APIs for Room Entry/Exit by Listener

onAudienceEnter

A listener entered the room.

```
- (void)onAudienceEnter:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceEnter(userInfo:));
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	UserInfo	Information of the user who entered the room

onAudienceExit

A listener exited the room.

```
- (void)onAudienceExit:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceExit(userInfo:));
```

The parameters are as detailed below:

Parameter	Type	Description
userInfo	UserInfo	Information of the listener who exited the room

Message Event Callback APIs

onRecvRoomTextMsg

A text message was received.

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

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
userInfo	UserInfo	Information of the sender

onRecvRoomCustomMsg

A custom message was received.

```
- (void)onRecvRoomCustomMsg:(NSString *)cmd
message:(NSString *)message
```

```
userInfo:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomCustomMsg(cmd:message:userInfo:));
```

The parameters are as detailed below:

Parameter	Type	Description
command	String	Custom command word used to distinguish between different message types
message	String	Text message
userInfo	UserInfo	Information of the sender

Invitation Signaling Callback APIs

onReceiveNewInvitation

An invitation was received.

```
- (void)onReceiveNewInvitation:(NSString *)identifier
inviter:(NSString *)inviter
cmd:(NSString *)cmd
content:(NSString *)content
NS_SWIFT_NAME(onReceiveNewInvitation(identifier:inviter:cmd:content:));
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
inviter	String	Inviter’s user ID
cmd	String	Custom command word specified by business
content	UserInfo	Content specified by business

onInviteeAccepted

The invitee accepted the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee’s user ID

onInviteeRejected

The invitee declined the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee’s user ID

onInvitationCancelled

The inviter canceled the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
inviter	String	Inviter’s user ID

Music Playback Status Callback APIs

onMusicPrepareToPlay

Music playback is ready.

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

The parameters are as detailed below:

Parameter	Type	Description
musicID	int32_t	`musicID` passed in for playback

onMusicProgressUpdate

Music playback progress.

```
- (void)onMusicProgressUpdate:(int32_t)musicID
progress:(NSInteger)progress total:(NSInteger)total
NS_SWIFT_NAME(onMusicProgressUpdate(musicID:progress:total:));
```

The parameters are as detailed below:

Parameter	Type	Description
musicID	int32_t	`musicID` passed in for playback
progress	NSInteger	Current playback progress in ms
total	NSInteger	Total duration in ms

onMusicCompletePlaying

Music playback was completed.

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

The parameters are as detailed below:

Parameter	Type	Description
musicID	int32_t	`musicID` passed in for playback

TRTCKaraoke (Android)

Last updated : 2021-12-24 16:08:01

`TRTCKaraokeRoom` is based on Tencent Real-Time Communication (TRTC) and Instant Messaging (IM). Its features include:

- A user can create a karaoke room and become a speaker, or enter a karaoke room as a listener.
- The room owner can manage song requests as well as remove a speaker.
- The room owner can also block a seat. Listeners cannot request to take a blocked seat.
- A listener can become a speaker to request songs and sing. A speaker can also become a listener.
- All users can send gifts and text and custom messages. Custom messages can be used to send on-screen comments and give likes.

`TRTCKaraokeRoom` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, please see Karaoke (Android).

- TRTC SDK: the TRTC SDK is used as a low-latency audio chat component.
- IM SDK: the `AVChatRoom` feature of the IM SDK is used to implement chat rooms. The attribute APIs of IM are used to store room information such as the seat list, and invitation signaling is used to send requests to speak or invite others to speak.

`TRTCKaraokeRoom` API Overview

Basic SDK APIs

API	Description
sharedInstance	Gets a singleton object.
destroySharedInstance	Terminates a singleton object.
setDelegate	Sets event callbacks.
setDelegateHandler	Sets the thread where event callbacks are.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets profile.

Room APIs

API	Description
createRoom	Creates a room (called by room owner). If the room does not exist, the system will automatically create a room.
destroyRoom	Terminates a room (called by room owner).
enterRoom	Enters a room (called by listener).
exitRoom	Exits a room (called by listener).
getRoomInfoList	Gets room list details.
getUserInfoList	Gets the user information of the specified `userId` . If the value is `null` , the information of all users in the room is obtained.

Music playback APIs

API	Description
startPlayMusic	Starts music.
stopPlayMusic	Stops music.
pausePlayMusic	Pauses music.
resumePlayMusic	Resumes music.

Seat management APIs

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

Local audio APIs

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

Remote audio APIs

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

Background music and audio effect APIs

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

Message sending APIs

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

Invitation signaling APIs

API	Description

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

`TRTCKaraokeRoomDelegate` API Overview

Common event callback APIs

API	Description
onError	Error
onWarning	Warning
onDebugLog	Log

Room event callback APIs

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

Seat list change callback APIs

API	Description
onSeatListChange	All seat changes
onAnchorEnterSeat	Someone became a speaker or was made a speaker by the room owner.
onAnchorLeaveSeat	Someone became a listener or was moved to listeners by the room owner.

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

Callback APIs for room entry/exit by listener

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

Message event callback APIs

API	Description
onRecvRoomTextMsg	Receipt of a text message
onRecvRoomCustomMsg	Receipt of a custom message

Signaling Event Callback APIs

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

Song event callback APIs

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

API	Description
onMusicCompletePlaying	Music playback was completed.

Basic SDK APIs

sharedInstance

This API is used to get a TRTCKaraokeRoom singleton object.

```
public static synchronized TRTCKaraokeRoom sharedInstance(Context context);
```

The parameters are as detailed below:

Parameter	Type	Description
context	Context	Android context, which will be converted to `ApplicationContext` for the calling of system APIs

destroySharedInstance

This API is used to terminate a TRTCKaraokeRoom singleton object.

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

```
public static void destroySharedInstance();
```

setDelegate

This API is used to set the event callbacks of TRTCKaraokeRoom. You can use `TRTCKaraokeRoomDelegate` to get different status notifications of TRTCKaraokeRoom.

```
public abstract void setDelegate(TRTCKaraokeRoomDelegate delegate);
```

> Note :

`setDelegate` is the delegate callback of `TRTCKaraokeRoom` .

setDelegateHandler

This API is used to set the thread where event callbacks are.

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

The parameters are as detailed below:

Parameter	Type	Description
handler	Handler	The status notifications of `TRTCKaraokeRoom` are sent to the handler thread you specify.

login

This API is used to log in.

```
public abstract void login(int sdkAppId,
String userId, String userSig,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
sdkAppId	int	You can view the `SDKAppID` via **Application Management** > **Application Info** in the TRTC console.
userId	String	ID of 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 protection signature. For more information on how to get it, please see How to Calculate UserSig.
callback	ActionCallback	Callback for login. The code is 0 if login succeeds.

logout

This API is used to log out.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for logout. The code is 0 if logout succeeds.

setSelfProfile

This API is used to set the profile.

```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
userName	String	Username
avatar	String	Profile photo address
callback	ActionCallback	Callback for profile setting. The code is 0 if the operation succeeds.

Room APIs

createRoom

This API is used to create a room (called by room owner).

```
public abstract void createRoom(int roomId, TRTCKaraokeRoomDef.RoomParam roomParam, TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID. You need to assign and manage the IDs in a centralized manner. Multiple `roomID` values can be aggregated into a karaoke room list. Currently, Tencent Cloud does not provide management services for karaoke room lists. Please manage the list on your own.

Parameter	Type	Description
roomParam	TRTCCreateRoomParam	Room information, such as room name, seat list information, and cover information. To manage seats, you must enter the number of seats in the room.
callback	ActionCallback	Callback for room creation. The code is 0 if the operation succeeds.

The process of creating a room and becoming a speaker is as follows:

1. A user calls `createRoom` to create a karaoke room, passing in room attributes (e.g., room ID, whether listeners need room owner's consent to speak, number of seats).
2. After creating the room, the user calls `enterSeat` to become a speaker.
3. The user will receive an `onSeatListChanget` notification about the change of the seat list, and can update the change to the UI.
4. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker, and mic capturing will be enabled automatically.

destroyRoom

This API is used to terminate a room (called by room owner).

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

The parameters are as detailed below:

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

enterRoom

This API is used to enter a room (called by listener).

```
public abstract void enterRoom(int roomId, TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	int	Room ID

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

The process of entering a room as a listener is as follows:

1. A user gets the latest karaoke room list from your server. The list may contain the `roomId` and room information of multiple karaoke rooms.
2. The user selects a room, and calls `enterRoom` with the room ID passed in to enter.
3. After entering the room, the user receives an `onRoomInfoChange` notification about room attribute change from the component. The attributes can be recorded, and corresponding changes can be made to the UI, including room name, whether room owner’s consent is required for listeners to speak, etc.
4. The user will receive an `onSeatListChange` notification about the change of the seat list and can update the change to the UI.
5. The user will also receive an `onAnchorEnterSeat` notification that someone became a speaker.

exitRoom

This API is used to exit a room.

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

The parameters are as detailed below:

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

getRoomInfoList

This API is used to get room list details. The room name and cover are set by the room owner via `roomInfo` when calling `createRoom()` .

> Note :
> You don’t need this API if both the room list and room information are managed on your server.

```
public abstract void getRoomInfoList(List<Integer> roomIdList, TRTCKaraokeRoomCallback.RoomInfoCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
roomIdList	List<Integer>	Room ID list
callback	RoomInfoCallback	Callback of room details

getUserInfoList

This API is used to get the user information of a specified `userId` .

```
public abstract void getUserInfoList(List<String> userIdList, TRTCKaraokeRoomCallback.UserListCallback userlistcallback);
```

The parameters are as detailed below:

Parameter	Type	Description
userIdList	List<String>	IDs of the users to query. If this parameter is `null` , the information of all users in the room is queried.
userlistcallback	UserListCallback	Callback of user details

Music Playback APIs

startPlayMusic

This API is used to play music (called after mic-on).

Note :

- After music playback starts, you will receive an `onMusicPrepareToPlay` notification.
- During music playback, all members in the room will keep receiving an `onMusicProgressUpdate` notification.
- After music playback stops, you will receive an `onMusicCompletePlaying` notification.

```
public abstract void startPlayMusic(int musicID, String originalUrl, String accompanyUrl);
```

The parameters are as detailed below:

Parameter	Type	Description
musicID	int	Music ID
originalUrl	String	Absolute path of the vocal track
accompanyUrl	String	Absolute path of the instrumental track

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

stopPlayMusic

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

> Note :
> After music playback stops, you will receive an `onMusicCompletePlaying` notification.

```
public abstract void stopPlayMusic();
```

pausePlayMusic

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

> Note :
> - The `onMusicProgressUpdate` notification will be paused.
> - No `onMusicCompletePlaying` notification will be received.

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

resumePlayMusic

This API is used to resume music (called after pause).

> Note :

No `onMusicPrepareToPlay` notification will be received.

```
public abstract void resumePlayMusic();
```

Seat Management APIs

enterSeat

This API is used to become a speaker (called by room owner or listener).

> Note :
> After a user becomes a speaker, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

```
public abstract void enterSeat(int seatIndex, TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to take
callback	ActionCallback	Callback for the operation

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

leaveSeat

This API is used to become a listener (called by speaker).

> Note :
> After a speaker becomes a listener, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.

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

The parameters are as detailed below:

Parameter	Type	Description
callback	ActionCallback	Callback for the operation

pickSeat

This API is used to place a user in a seat (called by room owner).

> Note :
> After the room owner places a user in a seat, all members in the room will receive an `onSeatListChange` notification and an `onAnchorEnterSeat` notification.

```
public abstract void pickSeat(int seatIndex, String userId, TRTCKaraokeRoomCallback.ActionCallbac
k callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to place the listener in
userId	String	User ID
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. In cases where the room owner needs listeners' consent to make them speakers, you can call `sendInvitation` first to send a request and, after receiving `onInvitationAccept` , call `pickSeat` .

kickSeat

This API is used to remove a speaker (called by room owner).

> Note :
> After a speaker is removed, all members in the room will receive an `onSeatListChange` notification and an `onAnchorLeaveSeat` notification.

```
public abstract void kickSeat(int seatIndex, TRTCKaraokeRoomCallback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to remove the speaker from
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list.

muteSeat

This API is used to mute/unmute a seat (called by room owner).

> Note :
> After a seat is muted/unmuted, all members in the room will receive an `onSeatListChange` notification and an `onSeatMute` notification.

```
public abstract void muteSeat(int seatIndex, boolean isMute, TRTCKaraokeRoomCallback.ActionCallba
ck callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to mute/unmute
isMute	boolean	`true` : mute; `false` : unmute
callback	ActionCallback	Callback for the operation

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

closeSeat

This API is used to block/unblock a seat (called by room owner).

Note :
After a seat is blocked/unblocked, all members in the room will receive an `onSeatListChange` notification and an `onSeatClose` notification.

```
public abstract void closeSeat(int seatIndex, boolean isClose, TRTCKaraokeRoomCallback.ActionCall
back callback);
```

The parameters are as detailed below:

Parameter	Type	Description
seatIndex	int	The number of the seat to block/unblock
isClose	boolean	`true` : block; `false` : unblock
callback	ActionCallback	Callback for the operation

Calling this API will immediately modify the seat list. The speaker on the seat specified by `seatIndex` will leave the seat.

Local Audio APIs

startMicrophone

This API is used to start mic capturing.

```
public abstract void startMicrophone();
```

stopMicrophone

This API is used to stop mic capturing.

```
public abstract void stopMicrophone();
```

setAudioQuality

This API is used to set audio quality.

```
public abstract void setAudioQuality(int quality);
```

The parameters are as detailed below:

Parameter	Type	Description
quality	int	Audio quality. For more information, please see setAudioQuality().

muteLocalAudio

This API is used to mute/unmute local audio.

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

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	Whether to mute or unmute audio. For more information, please see muteLocalAudio().

setSpeaker

This API is used to set whether to use the speaker or receiver.

```
public abstract void setSpeaker(boolean useSpeaker);
```

The parameters are as detailed below:

Parameter	Type	Description
useSpeaker	boolean	`true` : speaker; `false` : receiver

setAudioCaptureVolume

This API is used to set the mic capturing volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Capturing volume. Value range: 0-100 (default: 100)

setAudioPlayoutVolume

This API is used to set the playback volume.

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

The parameters are as detailed below:

Parameter	Type	Description
volume	int	Playback volume. Value range: 0-100 (default: 100)

muteRemoteAudio

This API is used to mute/unmute a specified user.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	String	ID of the user to mute/unmute
mute	boolean	`true` : mute; `false` : unmute

muteAllRemoteAudio

This API is used to mute/unmute all users.

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

The parameters are as detailed below:

Parameter	Type	Description
mute	boolean	`true` : mute; `false` : unmute

setVoiceEarMonitorEnable

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

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

The parameters are as detailed below:

Parameter	Type	Description
enable	boolean	`true` : enable; `false` : disable

Background Music and Audio Effect APIs

getAudioEffectManager

This API is used to get the background music and audio effect management object TXAudioEffectManager.

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

Message Sending APIs

sendRoomTextMsg

This API is used to broadcast a text message in a room, which is generally used for on-screen comments.

```
public abstract void sendRoomTextMsg(String message, TRTCKaraokeRoomCallback.ActionCallback callb
ack);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
callback	ActionCallback	Callback for message sending

sendRoomCustomMsg

This API is used to send a custom text message.

```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCKaraokeRoomCallback.Action
Callback callback);
```

The parameters are as detailed below:

Parameter	Type	Description

Parameter	Type	Description
cmd	String	Custom command word used to distinguish between different message types
message	String	Text message
callback	ActionCallback	Callback for the operation

Invitation Signaling APIs

sendInvitation

This API is used to send an invitation.

```
public abstract String sendInvitation(String cmd, String userId, String content, TRTCKaraokeRoomC
allback.ActionCallback callback);
```

The parameters are as detailed below:

Parameter	Type	Description
cmd	String	Custom command of business
userId	String	Invitee’s user ID
content	String	Invitation content
callback	ActionCallback	Callback for the operation

Response parameters:

Returned Value	Type	Description
inviteId	String	Invitation ID

acceptInvitation

This API is used to accept an invitation.

```
public abstract void acceptInvitation(String id, TRTCKaraokeRoomCallback.ActionCallback callback)
;
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

rejectInvitation

This API is used to decline an invitation.

```
public abstract void rejectInvitation(String id, TRTCKaraokeRoomCallback.ActionCallback callback)
;
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

cancelInvitation

This API is used to cancel an invitation.

```
public abstract void cancelInvitation(String id, TRTCKaraokeRoomCallback.ActionCallback callback)
;
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
callback	ActionCallback	Callback for the operation

`TRTCKaraokeRoomDelegate` Event Callback APIs

Common Event Callback APIs

onError

Callback for error.

> 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 as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Error message

onWarning

Callback for warning.

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

The parameters are as detailed below:

Parameter	Type	Description
code	int	Error code
message	String	Warning message

onDebugLog

Callback for log.

```
void onDebugLog(String message);
```

The parameters are as detailed below:

Parameter	Type	Description
message	String	Log information

Room Event Callback APIs

onRoomDestroy

Callback for room termination. When the owner terminates the room, all users in the room will receive this callback.

```
void onRoomDestroy(String roomId);
```

The parameters are as detailed below:

Parameter	Type	Description
roomId	String	Room ID

onRoomInfoChange

Callback for change of room information. This callback is sent after successful room entry. The information in `roomInfo` is passed in by the room owner during room creation.

```
void onRoomInfoChange(TRTCKaraokeRoomDef.RoomInfo roomInfo);
```

The parameters are as detailed below:

Parameter	Type	Description
roomInfo	RoomInfo	Room information

onUserMicrophoneMute

Callback of whether a user’s mic is muted. When a user calls `muteLocalAudio` , all members in the room will receive this callback.

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

The parameters are as detailed below:

Parameter	Type	Description
userId	String	User ID
mute	boolean	Volume. Value range: 0-100

onUserVolumeUpdate

Callback of the volume of each member in the room after the volume reminder is enabled.

```
void onUserVolumeUpdate(List<TRTCCloudDef.TRTCVolumeInfo> userVolumes, int totalVolume);
```

The parameters are as detailed below:

Parameter	Type	Description
userVolumes	List	List of user volumes
totalVolume	int	Total volume. Value range: 0-100

Seat Callback APIs

onSeatListChange

Callback for all seat changes.

```
void onSeatListChange(List<SeatInfo> seatInfoList);
```

The parameters are as detailed below:

Parameter	Type	Description
seatInfoList	List<SeatInfo>	Full seat list

onAnchorEnterSeat

Someone became a speaker or was made a speaker by the room owner.

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

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat taken
user	UserInfo	Details of the user who took the seat

onAnchorLeaveSeat

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

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

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat previously occupied by the speaker
user	UserInfo	Details of the user who became a listener

onSeatMute

The room owner muted/unmuted a seat.

```
void onSeatMute(int index, boolean isMute);
```

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat muted/unmuted
isMute	boolean	`true` : muted; `false` : unmuted

onSeatClose

The room owner blocked/unblocked a seat.

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

The parameters are as detailed below:

Parameter	Type	Description
index	int	The seat blocked/unblocked
isClose	boolean	`true` : blocked; `false` : unblocked

Callback APIs for Room Entry/Exit by Listener

onAudienceEnter

A listener entered the room.

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

The parameters are as detailed below:

Parameter	Type	Description
userInfo	UserInfo	Information of the listener who entered the room

onAudienceExit

A listener exited the room.

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

The parameters are as detailed below:

Parameter	Type	Description
userInfo	UserInfo	Information of the listener who exited the room

Message Event Callback APIs

onRecvRoomTextMsg

A text message was received.

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

The parameters are as detailed below:

Parameter	Type	Description
message	String	Text message
userInfo	UserInfo	Information of the sender

onRecvRoomCustomMsg

A custom message was received.

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

The parameters are as detailed below:

Parameter	Type	Description
command	String	Custom command word used to distinguish between different message types
message	String	Text message
userInfo	UserInfo	Information of the sender

Invitation Signaling Callback APIs

onReceiveNewInvitation

An invitation was received.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
inviter	String	Inviter’s user ID
cmd	String	Custom command word specified by business
content	String	Content specified by business

onInviteeAccepted

The invitee accepted the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee’s user ID

onInviteeRejected

The invitee declined the invitation.

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

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
invitee	String	Invitee’s user ID

onInvitationCancelled

The inviter canceled the invitation.

```
void onInvitationCancelled(String id, String inviter);
```

The parameters are as detailed below:

Parameter	Type	Description
id	String	Invitation ID
inviter	String	Inviter’s user ID

Music Playback Status Callback APIs

onMusicPrepareToPlay

Music playback is ready.

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

The parameters are as detailed below:

Parameter	Type	Description
musicID	int	`musicID` passed in for playback

onMusicProgressUpdate

Music playback progress.

```
void onMusicProgressUpdate(int musicID, long progress, long total);
```

The parameters are as detailed below:

Parameter	Type	Description
musicID	int	`musicID` passed in for playback
progress	long	Current playback progress in ms
total	long	Total duration in ms

onMusicCompletePlaying

Music playback was completed.

```
void onMusicCompletePlaying(int musicID);
```

The parameters are as detailed below:

Parameter	Type	Description
musicID	int	`musicID` passed in for playback