Tencent Real-Time Communication

Legacy Documentation

Product Documentation

Contents

Legacy Documentation
Integrating TUIRoom (Web)

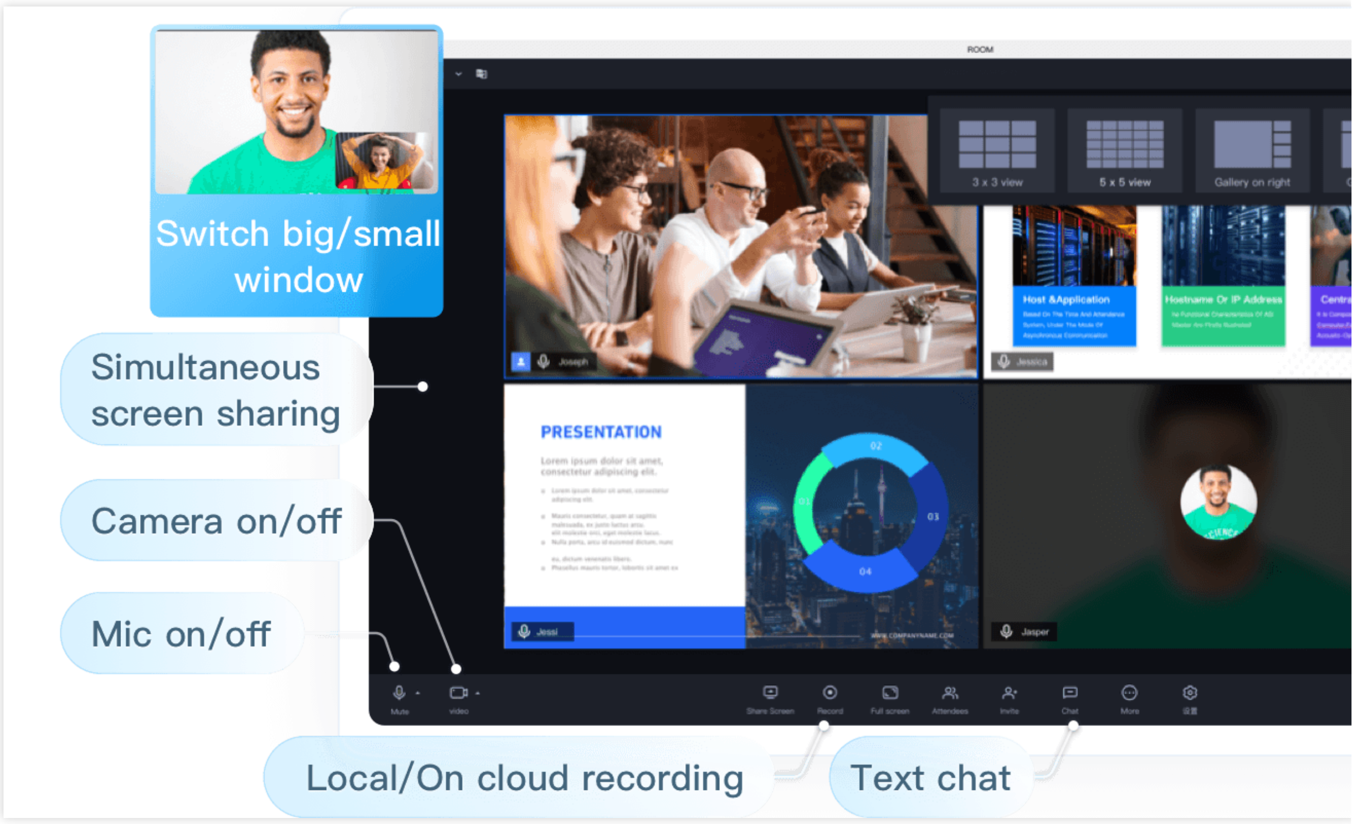
Last updated：2023-10-13 11:26:56

Overview

`TUIRoom` is an open-source audio/video component that comes with UI elements. It allows you to quickly integrate conferencing capabilities like screen sharing and chatting into your project.

Note

All components of TUIKit use Tencent Cloud's TRTC and Chat services. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will also be activated automatically. For Chat billing details, see Pricing.



Click here to try out more features of `TUIRoom` .

Click here to download the `TUIRoom` code and refer to this document to run the `TUIRoom` web demo.

This document shows you how to integrate the `TUIRoom` web component into your existing project.

Integration

The `TUIRoom` component is developed using Vue 3 + TypeScript + Pinia + Element Plus + SCSS, so your project must be based on Vue 3 + TypeScript.

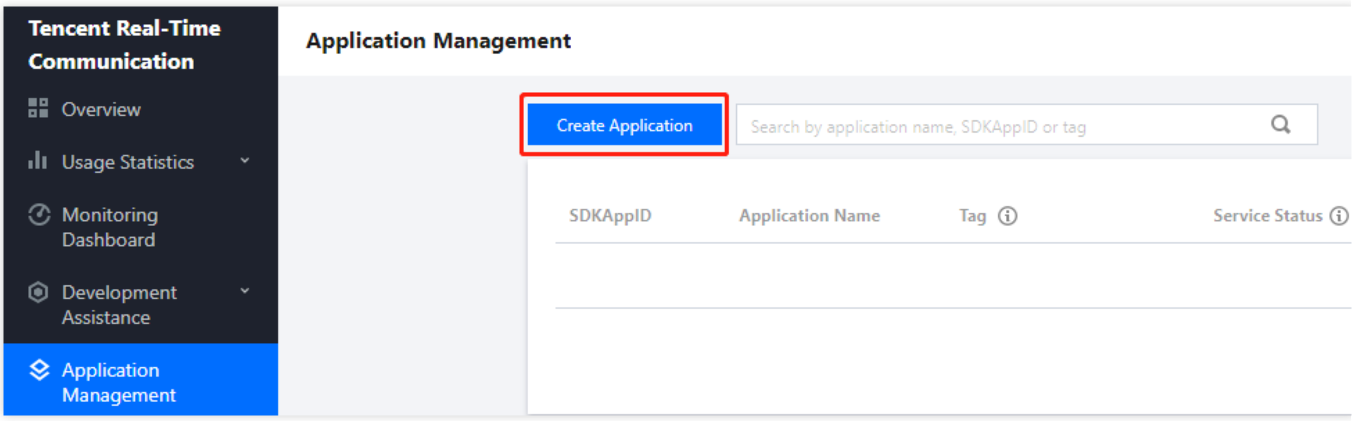
Step 1. Activate the TRTC service

`TUIRoom` is based on TRTC and Chat.

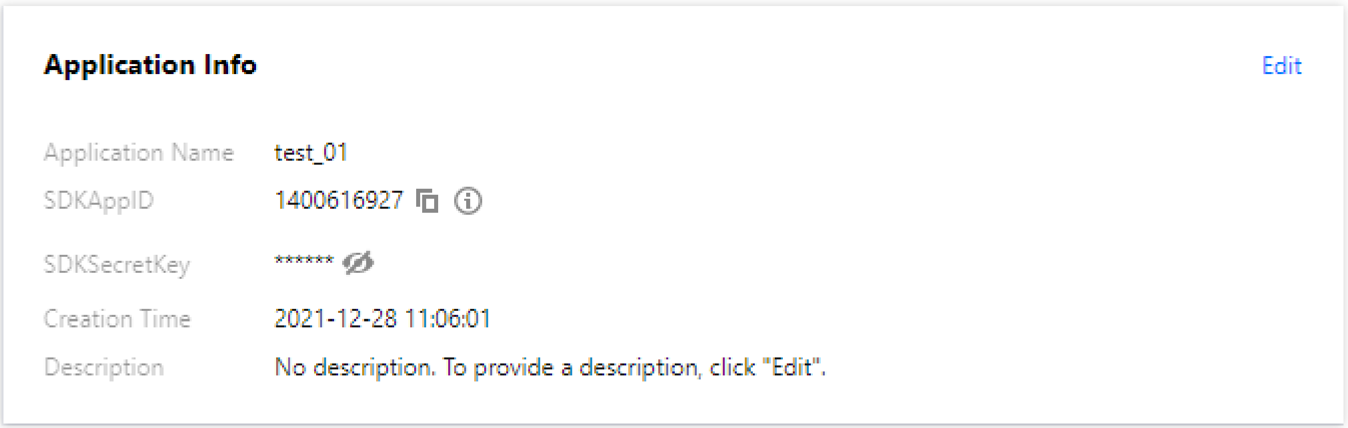
1. **Create a TRTC application**

If you don't have a Tencent Cloud account yet, sign up for one first.

In the TRTC console, click **Application Management** on the left sidebar and then click **Create Application**.



2. **Get the SDKAppID and key**
3. On the **Application Management** page, find the application you created, and click **Application Info** to view its `SDKAppID` (different applications cannot communicate with each other).

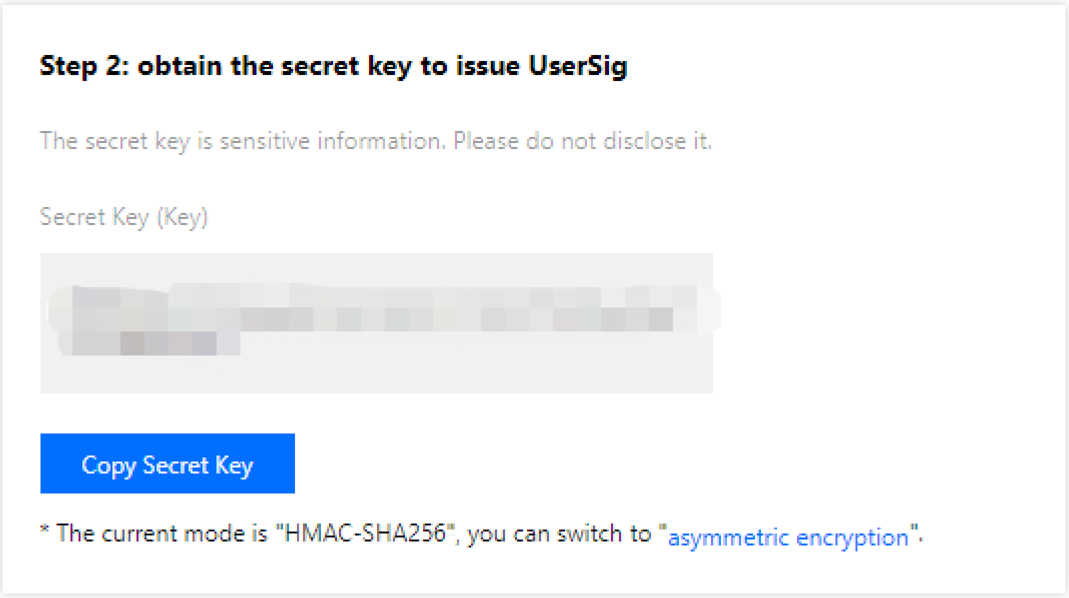


Application Info

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".

4. Select the **Quick Start** tab to view the application's secret key. Each `SDKAppID` corresponds to a secret key. They are used to generate the signature (`UserSig`) required to legitimately use TRTC services.


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".

5. **Generate UserSig** `UserSig` is a security signature designed by Tencent Cloud to prevent attackers from accessing your Tencent Cloud account. It is required when you initialize the `TUIRoom` component.

How do I calculate UserSig for debugging?

How do I calculate UserSig for production?

Step 2. Download and copy the `TUIRoom` component

1. Click here to clone or download the `TUIRoom` repository code.

2. Open your Vue3 + TypeScript project. You can use the build tool Vite or Webpack. If you don’t have a Vue3 + TypeScript project, create a template using either of the following two methods:

Create a Vue3 + Vite + TypeScript template

Create a Vue3 + Webpack + Typescript template

```
npm create vite@latest TUIRoom-demo -- --template vue
```

Note

During the creation process, press Enter first, select "Vue", and then select "vue-ts".

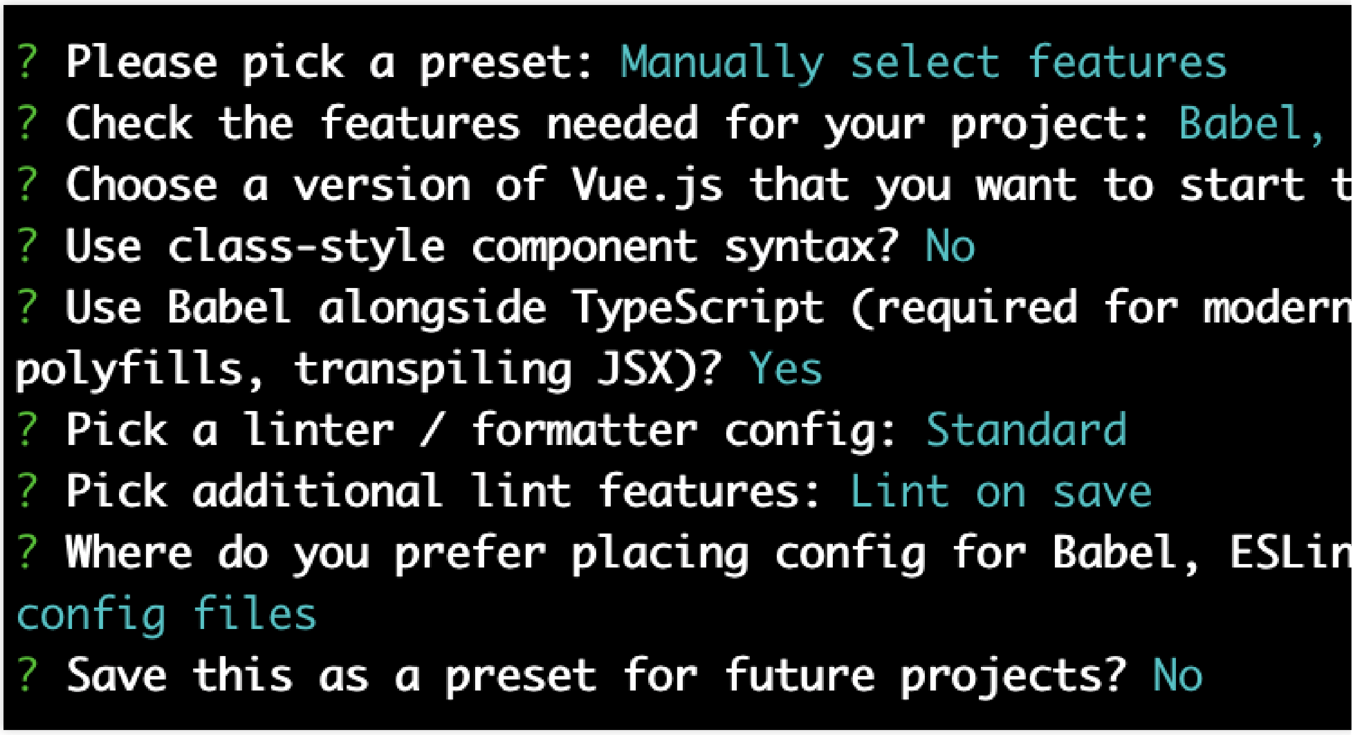
After the template is generated, run the script below:

```
cd TUIRoom-demo
npm install
npm run dev
```

```
// Install Vue CLI. If you use Vue CLI 4.x, your Node.js version must be v10 or lat
npm install -g @vue/cli
// Create a Vue3 + Webpack + TypeScript template
vue create TUIRoom-demo
```

notice

Select "Manually select features" as the template generation mode. For other settings, refer to the figure below:



```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel,
? Choose a version of Vue.js that you want to start t
? Use class-style component syntax? No
? Use Babel alongside TypeScript (required for modern
polyfills, transpiling JSX)? Yes
? Pick a linter / formatter config: Standard
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLin
config files
? Save this as a preset for future projects? No
```

After the template is generated, run the script below:

```
cd TUIRoom-demo
npm run serve
```

3. Copy the `TUIRoom/Web/src/TUIRoom` folder to the project's `src/` directory.

Step 3. Import the `TUIRoom` component

1. Import the `TUIRoom` component into your webpage, such as `App.vue` .

The `TUIRoom` component classifies users as hosts and members and offers APIs including init, createRoom, and enterRoom.
Hosts and members can call init to initialize application and user data. Hosts can call createRoom to create and enter rooms. Members can call enterRoom to join the rooms created by hosts.

```
<template>
 <room ref="TUIRoomRef"></room>
</template>

<script setup lang="ts">
 import { ref, onMounted } from 'vue';
```

```
// Import the `TUIRoom` component. Be sure to use the correct import path.
import Room from './TUIRoom/index.vue';
// Get the `TUIRoom` component elements used to call the component's APIs
const TUIRoomRef = ref();

 onMounted(async () => {
 // Initialize the `TUIRoom` component
 // A host needs to initialize the `TUIRoom` component before creating a room
 // A member needs to initialize the `TUIRoom` component before entering a room
 await TUIRoomRef.value.init({
     // Get the `SDKAppID` (see step 1)
     sdkAppId: 0,
     // The user's unique ID in your business
     userId: '',
     // For local development and debugging, you can quickly generate a `userSig`
     userSig: '',
     // The user's username in your business
     userName: '',
     // The URL of the user's profile photo in your business
     userAvatar: '',
     // The user's unique ID used for screen sharing. It must be in the format of
     shareUserId: '',
     // Refer to steps 1-3 above and use the `SDKAppID` and `shareUserId` to gener
     shareUserSig: '',
 })
  // By default, a room is created at this point. During actual implementation, yo
 await handleCreateRoom();
})

// The host creates a room. Call this API only when you need to create a room.
async function handleCreateRoom() {
 // `roomId` is the ID of the room to enter, which must be a number.
 // The valid values of `roomMode` are `FreeSpeech` (free speech mode) and `ApplyS
 // `roomParam` specifies whether to turn on the mic/camera upon room entry, as we
 const roomId = 123456;
 const roomMode = 'FreeSpeech';
 const roomParam = {
     isOpenCamera: true,
     isOpenMicrophone: true,
 }
 await TUIRoomRef.value.createRoom(roomId, roomMode, roomParam);
}

// The member enters a room. This API is called by a member to join an existing ro
async function handleEnterRoom() {
 // `roomId` is the ID of the room entered by the user, which must be a number.
 // `roomParam` specifies whether to turn on the mic/camera upon room entry, as we
```

```
    const roomId = 123456;
    const roomParam = {
        isOpenCamera: true,
        isOpenMicrophone: true,
    }
    await TUIRoomRef.value.enterRoom(roomId, roomParam);
  }
</script>

<style>
html, body {
  width: 100%;
  height: 100%;
  margin: 0;
}

#app {
  width: 100%;
  height: 100%;
}
</style>
```

Note

Copy the above code to your webpage and replace the parameter values for the APIs with the actual values.

Step 4. Configure the development environment

After the `TUIRoom` component is imported, in order to ensure that the project can run normally, the following configurations are required:

Set up the Vue3 + Vite + TypeScript development environment

Set up the Vue3 + Webpack + TypeScript environment

1. **Install dependencies**

Install development environment dependencies:

```
npm install sass typescript unplugin-auto-import unplugin-vue-components -S -D
```

Install production environment dependencies:

```
npm install element-plus events mitt pinia rtc-beauty-plugin tim-js-sdk trtc-js-sdk
```

2. **Register Pinia** `TUIRoom` uses Pinia for room data management. You need to register Pinia in the project entry file `src/main.ts` .

```
// `src/main.ts` file
import { createPinia } from 'pinia';

const app = createApp(App);
// Register Pinia
app.use(createPinia());
app.mount('#app');
```

3. **Import Element Plus components**

`TUIRoom` uses Element Plus UI components, which you need to import in `vite.config.ts` . You can import only the components you need.

Note

Add the code in the file. Do not delete the existing configuration.

```
// vite.config.ts
import AutoImport from 'unplugin-auto-import/vite';
import Components from 'unplugin-vue-components/vite';
import { ElementPlusResolver } from 'unplugin-vue-components/resolvers';

export default defineConfig({
    // ...
    plugins: [
        AutoImport({
            resolvers: [ElementPlusResolver()],
        }),
        Components({
            resolvers: [ElementPlusResolver({
                importStyle: 'sass',
            })],
        }),
    ],
    css: {
        preprocessorOptions: {
            scss: {
                // ...
                additionalData: `
                    @use "./src/TUIRoom/assets/style/element.scss" as *;
                `,
            },
        },
    },
});
```

Meanwhile, in order to ensure that Element Plus UI components can display styles properly, you need to load Element Plus component styles in the entry file `src/main.ts` .

```
// src/main.ts
import 'element-plus/theme-chalk/el-message.css';
import 'element-plus/theme-chalk/el-message-box.css';
```

1. **Install dependencies**

Install development environment dependencies:

```
npm install sass sass-loader typescript unplugin-auto-import unplugin-vue-component
```

Install production environment dependencies:

```
npm install element-plus events mitt pinia rtc-beauty-plugin tim-js-sdk trtc-js-sdk
```

2. **Register Pinia** `TUIRoom` uses Pinia for room data management. You need to register Pinia in the project entry file `src/main.ts` .

```
// `src/main.ts` file
import { createPinia } from 'pinia';

const app = createApp(App);
// Register Pinia
app.use(createPinia());
app.mount('#app');
```

3. **Import Element Plus components**

`TUIRoom` uses Element Plus UI components, which you need to import in `vue.config.js` . You can manually import only the components you need.

notice

Add the code in the file. Do not delete the existing configuration.

```
// vue.config.js
const { defineConfig } = require('@vue/cli-service')
const AutoImport = require('unplugin-auto-import/webpack')
const Components = require('unplugin-vue-components/webpack')
const { ElementPlusResolver } = require('unplugin-vue-components/resolvers')
const ElementPlus = require('unplugin-element-plus/webpack')

module.exports = defineConfig({
  transpileDependencies: true,
  css: {
    loaderOptions: {
      scss: {
        additionalData: '@use "./src/TUIRoom/assets/style/element.scss" as *;'
      }
    }
  },
  configureWebpack: {
    plugins: [
      AutoImport({
        resolvers: [ElementPlusResolver({ importStyle: 'sass' })]
      }),
      Components({
        resolvers: [ElementPlusResolver({ importStyle: 'sass' })]
      }),
      // Specify the theme color when importing the components
      ElementPlus({
        useSource: true
      })
    ]
  }
})
```

Meanwhile, in order to ensure that Element Plus UI components can display styles properly, you need to load Element Plus component styles in the entry file `src/main.ts` .

```
// src/main.ts
import 'element-plus/theme-chalk/el-message.css';
import 'element-plus/theme-chalk/el-message-box.css';
```

4. **Configure the TS file**

Add the following configuration to `src/shims-vue.d.ts` :

```
}
```

```
declare module 'tsignaling/tsignaling-js' {
  import TSignaling from 'tsignaling/tsignaling-js';
  export default TSignaling;
}

declare module 'tim-js-sdk' {
  import TIM from 'tim-js-sdk';
  export default TIM;
}

declare const Aegis: any;
```

Step 5. Run your project in the development environment

In the console, execute the development environment script. Then, open the page integrated with the `TUIRoom` component with a browser.

If you used the script in step 2 **to create a Vue + Vite + TypeScript project**, follow the steps below:

1. Run the development environment command.

```
npm run dev
```

2. Open `http://localhost:3000/` in a browser.

Note

Because Element Plus components are imported manually, it may take a relatively long time for the page to load in the development environment for the first time. This will not be an issue after packaging.

3. Try out the features of the `TUIRoom` component.

If you used the script in step 2 to create a Vue + Webpack + TypeScript project, follow the steps below:

1. Run the development environment command.

```
npm run serve
```

2. Open `http://localhost:8080/` in a browser.

Note

If an ESLint error occurs in the `src/TUIRoom` directory, you can disable ESLint by adding `/src/TUIRoom` to the `.eslintignore` file.

3. Try out the features of the `TUIRoom` component.

Step 6. Commercial Scenario Deployment

1. Package dist file

```
npm run build
```

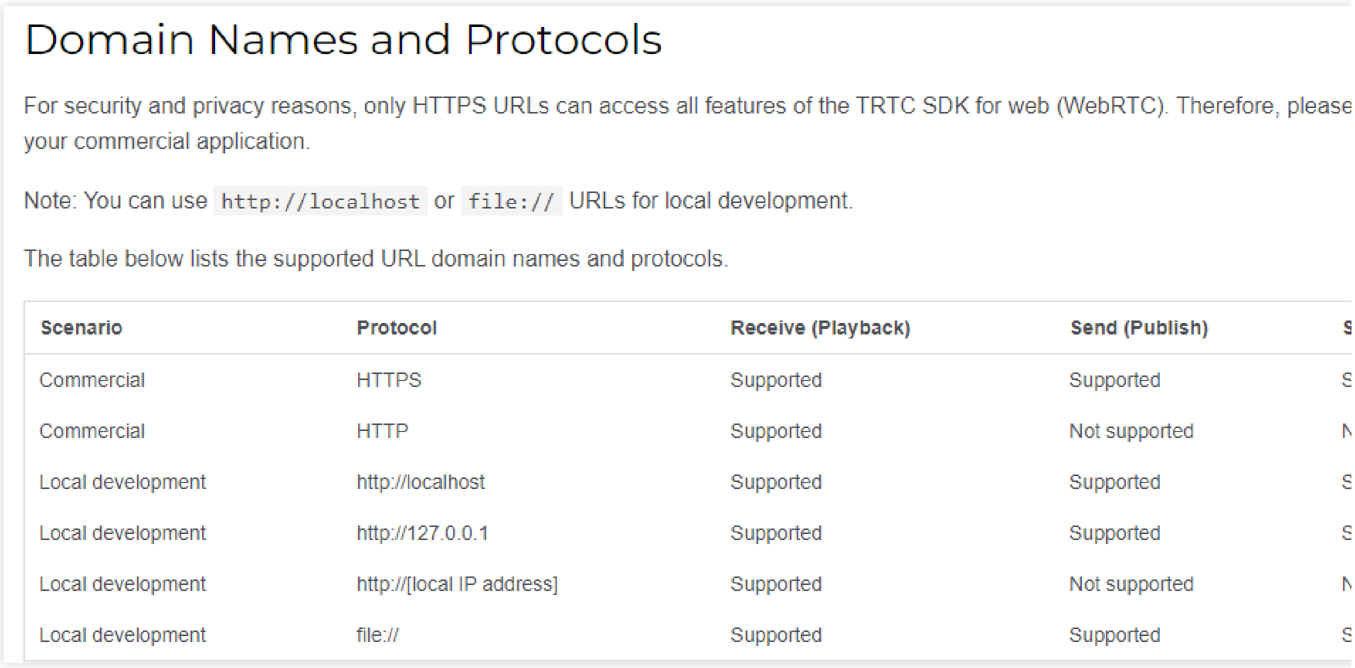
Note

Please check the package.json file for the actual packaging command.

2. Deploy the dist file to the server

Note

Commercial Scenario requires the use of https domain name.



Domain Names and Protocols

For security and privacy reasons, only HTTPS URLs can access all features of the TRTC SDK for web (WebRTC). Therefore, please your commercial application.

Note: You can use `http://localhost` or `file://` URLs for local development.

The table below lists the supported URL domain names and protocols.

Scenario	Protocol	Receive (Playback)	Send (Publish)	S
Commercial	HTTPS	Supported	Supported	S
Commercial	HTTP	Supported	Not supported	N
Local development	http://localhost	Supported	Supported	S
Local development	http://127.0.0.1	Supported	Supported	S
Local development	http://[local IP address]	Supported	Not supported	N
Local development	file://	Supported	Supported	S

Appendix: TUIRoom APIs

TUIRoom APIs

init

This API is used to initialize `TUIRoom` data. All users using `TUIRoom` need to call this API first.

```
TUIRoomRef.value.init(roomData);
```

The parameters are described below:

Parameter	Type	Description
roomData	object	-
roomData.sdkAppId	number	The SDKAppID.
roomData.userId	string	The unique user ID.

roomData.userSig	string	The UserSig.
roomData.userName	string	The username.
roomData.userAvatar	string	The user profile photo.
roomData.shareUserId	string	The `UserId` used for screen sharing, which must be in the format of `share_${userId}` . You don't need to pass this parameter if you don't need the screen sharing feature.
roomData.shareUserSig	string	`UserSig` used for screen sharing, which is optional.

createRoom

This API is used by a host to create a room.

```
TUIRoomRef.value.createRoom(roomId, roomMode, roomParam);
```

The parameters are described below:

Parameter	Type	Description
roomId	number	The room ID.
roomMode	string	The speech mode, including `FreeSpeech` (free speech) and `ApplySpeech` (request-to-speak). The default value is `FreeSpeech` , which is the only supported mode currently.

roomParam	Object	Optional
roomParam.isOpenCamera	string	Whether to turn on the camera upon room entry. This parameter is optional and the default is no.
roomParam.isOpenMicrophone	string	Whether to turn on the mic upon room entry. This parameter is optional and the default is no.
roomParam.defaultCameraId	string	The ID of the default camera, which is optional.
roomParam.defaultMicrophoneId	string	The ID of the default mic, which is optional.
roomParam.defaultSpeakerId	String	The ID of the default speaker, which is optional.

enterRoom

This API is used by a member to enter a room.

```
TUIRoomRef.value.enterRoom(roomId, roomParam);
```

The parameters are described below:

Parameter	Type	Description
roomId	number	The Room ID.
roomParam	Object	Optional
roomParam.isOpenCamera	string	Whether to turn on the camera upon room entry. This

		parameter is optional and the default is no.
roomParam.isOpenMicrophone	string	Whether to turn on the mic upon room entry. This parameter is optional and the default is no.
roomParam.defaultCameraId	string	The ID of the default camera, which is optional.
roomParam.defaultMicrophoneId	string	The ID of the default mic, which is optional.
roomParam.defaultSpeakerId	String	The ID of the default speaker, which is optional.

TUIRoom events

onRoomCreate

A room was created.

```
<template>
  <room ref="TUIRoomRef" @on-room-create="handleRoomCreate"></room>
</template>

<script setup lang="ts">
  // Import the `TUIRoom` component. Be sure to use the correct import path.
  import Room from './TUIRoom/index.vue';

  function handleRoomCreate(info) {
    if (info.code === 0) {
      console.log('Room created successfully')
    }
  }
</script>
```

onRoomEnter

A member entered the room.

```
<template>
  <room ref="TUIRoomRef" @on-room-enter="handleRoomEnter"></room>
</template>
```

```
<script setup lang="ts">
  // Import the `TUIRoom` component. Be sure to use the correct import path.
  import Room from './TUIRoom/index.vue';

  function handleRoomEnter(info) {
    if (info.code === 0) {
      console.log('Entered room successfully')
    }
  }
</script>
```

onRoomDestory

The host closed the room.

```
<template>
  <room ref="TUIRoomRef" @on-room-destory="handleRoomDestory"></room>
</template>

<script setup lang="ts">
  // Import the `TUIRoom` component. Be sure to use the correct import path.
  import Room from './TUIRoom/index.vue';

  function handleRoomDestory(info) {
    if (info.code === 0) {
      console.log('The host closed the room successfully')
    }
  }
</script>
```

onRoomExit

A member exited the room.

```
<template>
  <room ref="TUIRoomRef" @on-room-exit="handleRoomExit"></room>
</template>
```

```
<script setup lang="ts">
  // Import the `TUIRoom` component. Be sure to use the correct import path.
  import Room from './TUIRoom/index.vue';

  function handleRoomExit(info) {
    if (info.code === 0) {
      console.log('The member exited the room successfully')
    }
  }
</script>
```

onKickOff

A member was removed from the room by the host.

```
<template>
  <room ref="TUIRoomRef" @on-kick-off="handleKickOff"></room>
</template>

<script setup lang="ts">
  // Import the `TUIRoom` component. Be sure to use the correct import path.
  import Room from './TUIRoom/index.vue';

  function handleKickOff(info) {
    if (info.code === 0) {
      console.log('The member was removed from the room by the host')
    }
  }
</script>
```

Integrating TUIRoom (Android)

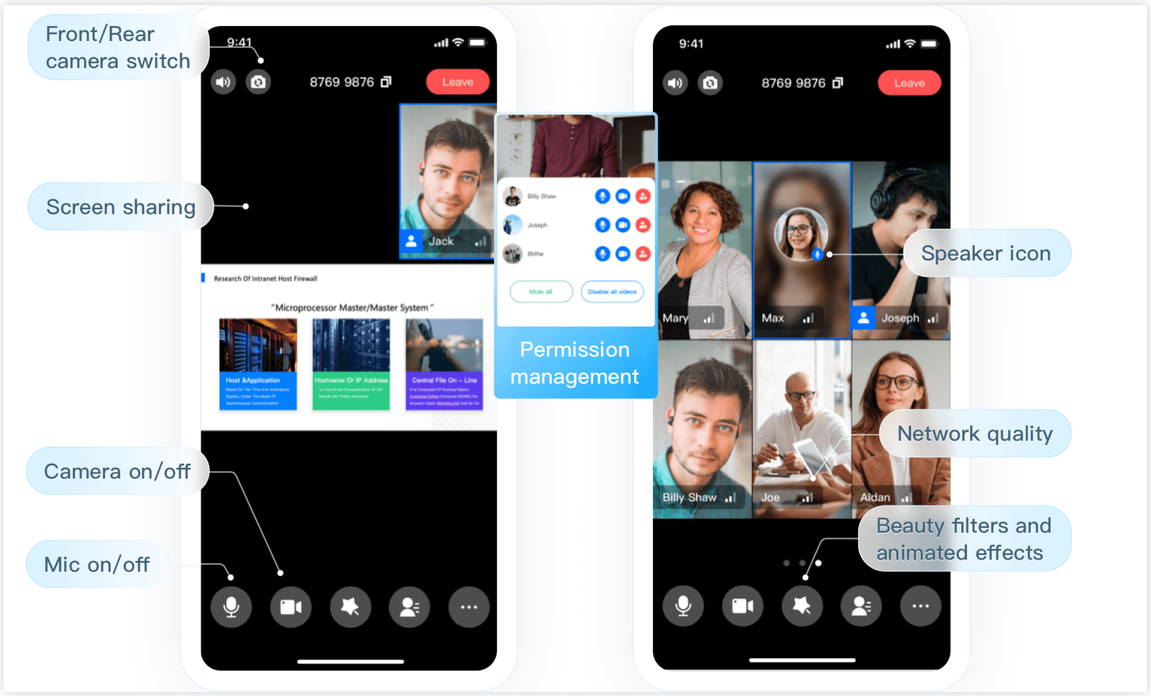
Last updated：2023-10-13 11:28:35

Overview

`TUIRoom` is an open-source UI component for audio/video communication. With just a few lines of code changes, you can integrate it into your project to implement screen sharing, beauty filters, low-latency video calls, and other features. In addition to the Android component, we also offer components for iOS, Windows, macOS, and more.

Note

All components of TUIKit use Tencent Cloud's TRTC and Chat services. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will also be activated automatically. For Chat billing details, see Pricing.



Integration

Step 1. Download and import the `TUIRoom` component

Go to the component's GitHub page, clone or download the code, and copy the `tuiroom` and `debug` , and `tuibeauty` folders in the `Android` directory to your project. Then, do the following to import the component:

1. Add the code below in `setting.gradle` :

```
include ':tuiroom'
include ':debug'
include ':tuibeauty'
```

2. Add dependencies on `tuiroom` , `debug` , and `tuibeauty` to the `build.gradle` file in `app` :

```
api project(':tuiroom')
api project(':debug')
api project(':tuibeauty')
```

3. Add the TRTC SDK (`liteavSdk`) and Chat SDK (`imsdk`) dependencies in `build.gradle` in the root directory:

```
ext {
 liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"
 imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"
}
```

Step 2. Configure permission requests and obfuscation rules

1. Configure permission requests for your app in `AndroidManifest.xml` . The SDKs need the following permissions (on Android 6.0 and later, the mic 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.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.CAMERA" />
<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. Create and initialize an instance of the component

```
    }
```

```
// 1. Log in to the component
TUILogin.addLoginListener(new TUILoginListener() {
    @Override
    public void onKickedOffline() {  // Callback for forced logout (for example, du
    }

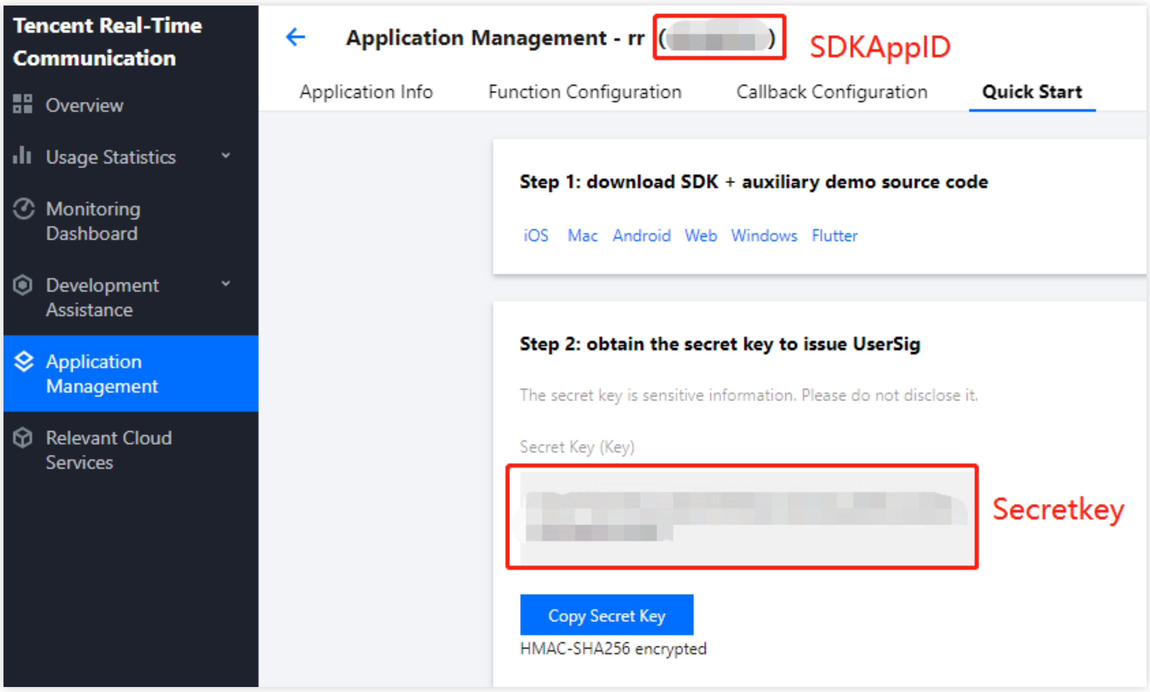
    @Override
    public void onUserSigExpired() { // Callback for `userSig` expiration
    }
});
```

```
TUILogin.login(context, "Your SDKAppId", "Your userId", "Your userSig", null);


// 2. Initialize the `TUIRoom` instance
TUIRoom tuiRoom = TUIRoom.sharedInstance(this);
```

Parameter description

SDKAppID: **TRTC application ID**. If you haven't activated TRTC, log in to the TRTC console, create a TRTC application, click **Application Info**, and select the **Quick Start** tab to view its `SDKAppID` .



Step 1: download SDK + auxiliary demo source code

Step 2: obtain the secret key to issue UserSig

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

Secretkey: **TRTC application key**. Each secret key corresponds to a `SDKAppID` . You can view your application’s secret key on the Application Management page 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 (_). We recommend that you keep it consistent with your user account system.

UserSig: Security signature calculated based on `SDKAppID` , `userId` , and `Secretkey` . You can click here to quickly generate a `UserSig` for testing. For more information, see UserSig.

Step 4. Implement group audio/video communication

1. **Create a room**

```
tuiRoom.createRoom(12345, TUIRoomCoreDef.SpeechMode.FREE_SPEECH, true, true);
```

2. **Join a room**

```
tuiRoom.enterRoom(12345, true, true);
```

Step 5. Implement room management (optional)

1. **The room owner calls** TUIRoomCore#destroyRoom **to close the room**.

```
// 1. The room owner calls the API below to close the room.
mTUIRoomCore.destroyRoom(new TUIRoomCoreCallback.ActionCallback() {
 @Override
 public void onCallback(int code, String msg) {

 }
});

Other users in the room will receive the `onDestroyRoom` callback.
@Override
public void onDestroyRoom() {
  // The room owner closes and exits the room.
}
```

2. **A user in the room calls** TUIRoomCore#leaveRoom **to leave the room**.

```
// 1. A user (not the room owner) calls the API below to leave the room.
mTUIRoomCore.leaveRoom(new TUIRoomCoreCallback.ActionCallback() {
  @Override
  public void onCallback(int code, String msg) {

  }
});
```

```
Other users in the room will receive the `onRemoteUserLeave` callback.
@Override
public void onRemoteUserLeave(String userId) {
     Log.d(TAG, "onRemoteUserLeave userId: " + userId);
}
```

Step 6. Implement screen sharing (optional)

Call TUIRoomCore#startScreenCapture to implement screen sharing.

```
// 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$TXScreenCaptureAssista
        android:theme="@android:style/Theme.Translucent" />
</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.p
        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 grants the permission.
                startScreenCapture();
            } else {
            }
        }
    }
}

// 4. Start screen sharing
private void startScreenCapture() {
        TRTCCloudDef.TRTCVideoEncParam encParams = new TRTCCloudDef.TRTCVideoEncPar
        encParams.videoResolution = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_1280_720;
        encParams.videoResolutionMode = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_POR
        encParams.videoFps = 10;
        encParams.enableAdjustRes = false;
        encParams.videoBitrate = 1200;

        TRTCCloudDef.TRTCScreenShareParams params = new TRTCCloudDef.TRTCScreenShar
        mTUIRoom.stopCameraPreview();
        mTUIRoom.startScreenCapture(encParams, params);
}
```

Suggestions and Feedback

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

Integrating TUIRoom (iOS)

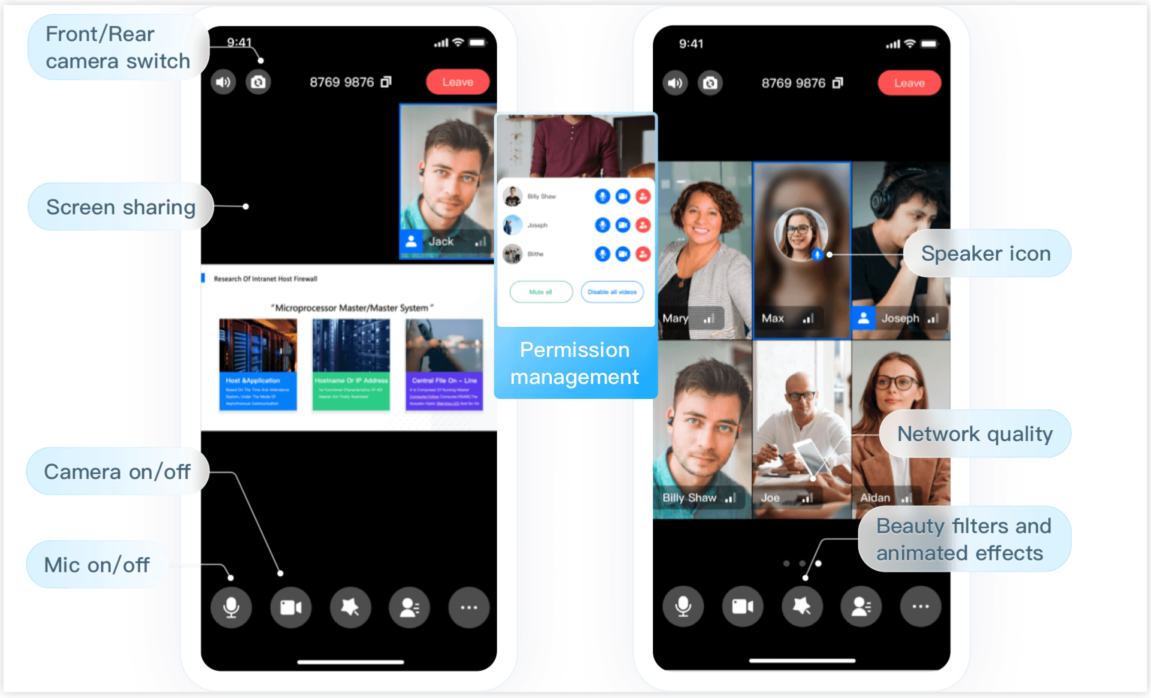
Last updated：2023-10-13 11:28:56

Overview

`TUIRoom` is an open-source UI component for audio/video communication. With just a few lines of code changes, you can integrate it into your project to implement screen sharing, beauty filters, low-latency video calls, and other features. In addition to the iOS component, we also offer components for Android, Windows, macOS, and more.

Note

All TUIKit components are based on Tencent Cloud's TRTC and Chat services. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will also be activated automatically. For Chat billing details, see Pricing.



Integration

Step 1. Import the `TUIRoom` component

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

1. Create a `TUIRoom` folder in the same directory as `Podfile` in your project.
2. Go to the component's GitHub page, clone or download the code, and copy the `Source` , `Resources` , `TUIBeauty` , and `TXAppBasic` folders and the `TUIRoom.podspec` file in TUIRoom/iOS/ to the `TUIRoom` folder in your project.
3. Add the following dependencies to your `Podfile` and run `pod install` to import the component.

```
# :path => "The relative path of `TUIRoom.podspec`"
pod 'TUIRoom', :path => "./TUIRoom/TUIRoom.podspec", :subspecs => ["TRTC"]
# :path => "The relative path of `TXAppBasic.podspec`"
```

```
pod 'TXAppBasic', :path => "./TUIRoom/TXAppBasic/"
# :path => "The relative path of `TUIBeauty.podspec`"
pod 'TUIBeauty', :path => "./TUIRoom/TUIBeauty/"
```

Note

The `Source` and `Resources` folders and the `TUIRoom.podspec` file must be in the same directory.

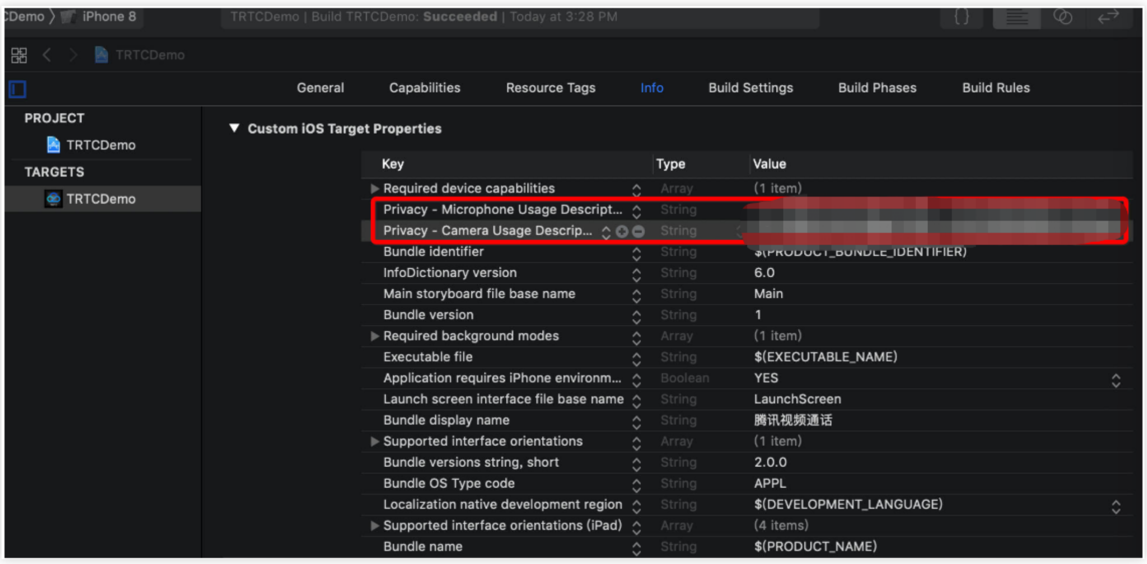
`TXAppBasic.podspec` is in the `TXAppBasic` folder.

`TUIBeauty.podspec` is in the `TCBeautyKit` folder.

Step 2. Configure permissions

Your app needs mic and camera permissions to implement audio/video communication. Add the two items below to `Info.plist` of your app. Their content is what users see in the mic and camera access pop-up windows.

```
<key>NSCameraUsageDescription</key>
<string>RoomApp needs to access your camera to capture video.</string>
<key>NSMicrophoneUsageDescription</key>
<string>RoomApp needs to access your mic to capture audio.</string>
```



Demo 〉 iPhone 8 TRTCDemo | Build TRTCDemo: **Succeeded** | Today at 3:28 PM

TRTCDemo

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

PROJECT
TRTCDemo
TARGETS
TRTCDemo

▼ Custom iOS Target Properties

Key	Type	Value
▶ Required device capabilities	Array	(1 item)
Privacy - Microphone Usage Descript...	String	[redacted]
Privacy - Camera Usage Descrip...	String	[redacted]
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Main storyboard file base name	String	Main
Bundle version	String	1
▶ Required background modes	Array	(1 item)
Executable file	String	$(EXECUTABLE_NAME)
Application requires iPhone environm...	Boolean	YES
Launch screen interface file base name	String	LaunchScreen
Bundle display name	String	腾讯视频通话
▶ Supported interface orientations	Array	(1 item)
Bundle versions string, short	String	2.0.0
Bundle OS Type code	String	APPL
Localization native development region	String	$(DEVELOPMENT_LANGUAGE)
▶ Supported interface orientations (iPad)	Array	(4 items)
Bundle name	String	$(PRODUCT_NAME)

Step 3. Create and initialize an instance of the component

Objective-C

Swift

```
@import TUIRoom;
@import TUICore;

// 1. Log in to the component
[TUILogin login:@"Your SDKAppID" userID:@"Your UserID" userSig:@"Your UserSig" succ

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

}];
// 2. Initialize the `TUIRoom` instance
TUIRoom *tuiRoom = [TUIRoom sharedInstance];
```

~~~
```
~~~

```
import TUIRoom
import TUICore

// 1. Log in to the component
TUILogin.login("Your SDKAppID", userID: "Your UserID", userSig: "Your UserSig") {

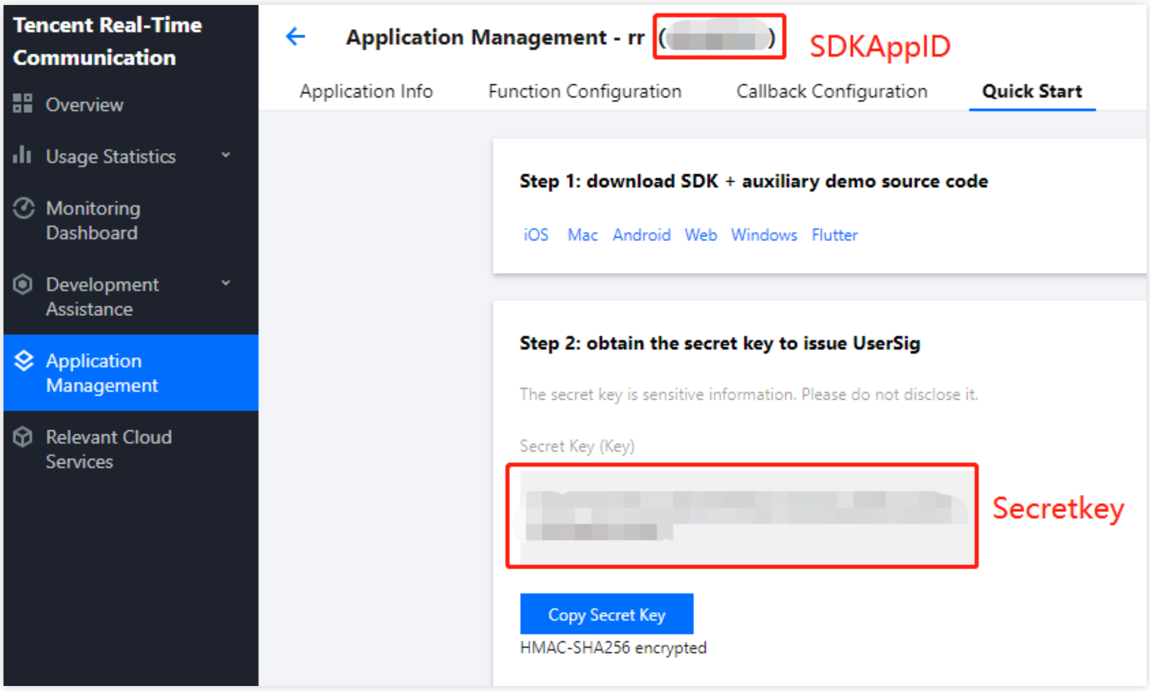
} fail: { code, msg in
```

```
}

// 2. Initialize the `TUIRoom` instance
let tuiRoom = TUIRoom.sharedInstance
```
```

Parameter description:

SDKAppID: **TRTC application ID**. If you haven't activated TRTC, log in to the TRTC console, create a TRTC application, click **Application Info**, and select the **Quick Start** tab to view its `SDKAppID` .



Secretkey: **TRTC application key**. Each secret key corresponds to an `SDKAppID` . You can view your application's secret key on the Application Management page of the TRTC console.

UserId: Current user ID, which is a custom string that can contain up to 32 bytes of letters and digits (special characters are not supported).

UserSig: Security signature calculated based on `SDKAppID` , `userId` , and `Secretkey` . You can click here to quickly generate a `UserSig` for testing or calculate it on your own by referring to our TUIRoom demo project. For more information, see UserSig.

Step 4. Implement group audio/video communication

1. **Create a room**

Objective-C

Swift
```

```
@import TUIRoom;

[tuiRoom createRoomWithRoomId:12345 speechMode:TUIRoomFreeSpeech isOpenCamera:YES i
```

```
import TUIRoom

tuiRoom.createRoom(roomId: 12345, speechMode: .freeSpeech, isOpenCamera: true, isOp
```
```

2. **Join a room**

Objective-C

Swift
```

```
@import TUIRoom;

[tuiRoom enterRoomWithRoomId:12345 isOpenCamera:YES isOpenMicrophone:YES]
```

~~~
import TUIRoom

tuiRoom.enterRoom(roomId: 12345, isOpenCamera: true, isOpenMicrophone: true)
```
~~~

Step 5. Implement room management (optional)

1. **The room owner calls** TUIRoomCore#destroyRoom **to close the room**.

Objective-C

Swift

```
@import TUIRoom;

[[TUIRoomCore shareInstance] destroyRoom:^(NSInteger code, NSString * _Nonnull mess

}];
```
```
```

~~~~
import TUIRoom

TUIRoomCore.shareInstance().destroyRoom { [weak self] _, _ in
    guard let self = self else { return }
    self.navigationController?.popViewController(animated: true)
}
```
~~~~

2. **A user in the room calls** TUIRoomCore#leaveRoom **to leave the room**.

Objective-C

Swift

~~~
@import TUIRoom;

[[TUIRoomCore shareInstance] leaveRoom:^(NSInteger code, NSString * _Nonnull messag

}];
```
~~~

~~~
import TUIRoom

TUIRoomCore.shareInstance().leaveRoom { [weak self] _, _ in
    guard let self = self else { return }
    self.navigationController?.popViewController(animated: true)
}
```
~~~

Step 6. Implement screen sharing (optional)

Call TUIRoomCore#startScreenCapture to implement screen sharing. For detailed directions, see Real-Time Screen Sharing (iOS).

Objective-C

Swift

```
@import TUIRoom;
@import TXLiteAVSDK_Professional;

TRTCVideoEncParam *params = [[TRTCVideoEncParam alloc] init];
params.videoResolution = TRTCVideoResolution_1280_720;
params.resMode = TRTCVideoResolutionModePortrait;
```

~~~
params.videoFps = 10;
params.enableAdjustRes = NO;
params.videoBitrate = 1500;

[[TUIRoomCore shareInstance] startScreenCapture:param];
```
~~~

````
import TUIRoom

 // 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)

```
```

FAQs

How do I install CocoaPods?

Enter the following command in a terminal window (you need to install Ruby on your Mac first):
```

```
sudo gem install cocoapods
```

Note

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

Integrating TUIRoom (Flutter)

Last updated：2023-10-13 11:29:19

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.

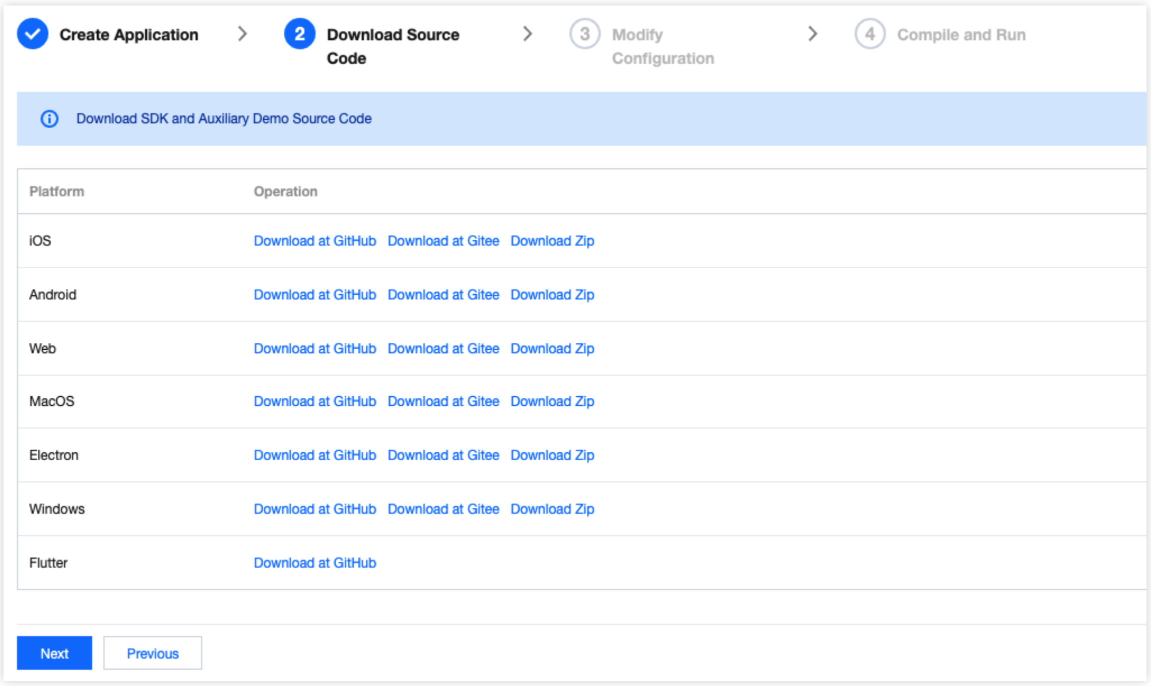
Note

All TUIKit components are based on Tencent Cloud's TRTC and Chat. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will also be activated automatically. For Chat billing details, see Pricing.

Using the App's UI

Step 1. Create an application

1. In the TRTC console, click **Development Assistance** > Demo Quick Run.
2. Enter an application name such as `TestMeetingRoom` and click **Create**.
3. Click **Next**.



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 relies on Tencent Cloud's TRTC and Chat services. When you activate TRTC, Chat will be activated automatically. For Chat billing details, see Pricing.

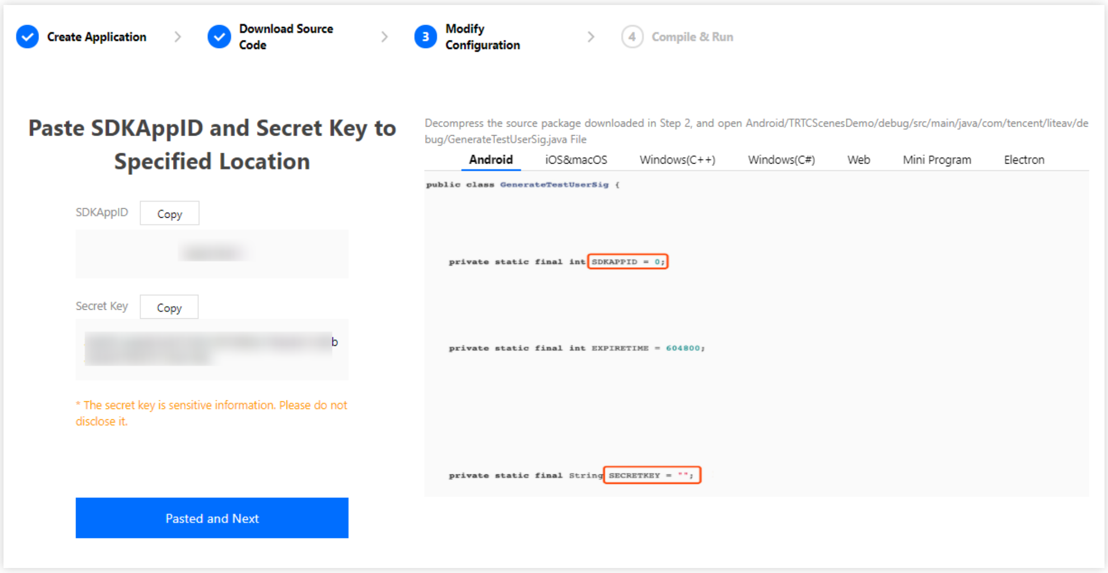
Step 2. Download the application source code

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: A placeholder by default. Set it to the actual `SDKAppID`.
SECRETKEY: A placeholder by default. Set it to the actual secret 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

In this document, the method to obtain `UserSig` is to configure the secret key in the client code. In this method, the secret key is vulnerable to decompilation and reverse engineering. Once your secret key is leaked, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and debugging**.

The best practice 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 your server for a dynamic `UserSig` . For more information, see How do I calculate `UserSig` during production?.

Step 4. Compile and run the demo

Caution

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

1. Open `\\ios project` in the source code directory with Xcode (11.0 or above).
2. Compile and run the demo project.

1. Run `flutter run` .
2. Open the demo project with Android Studio (3.5 or later), and click **Run**.

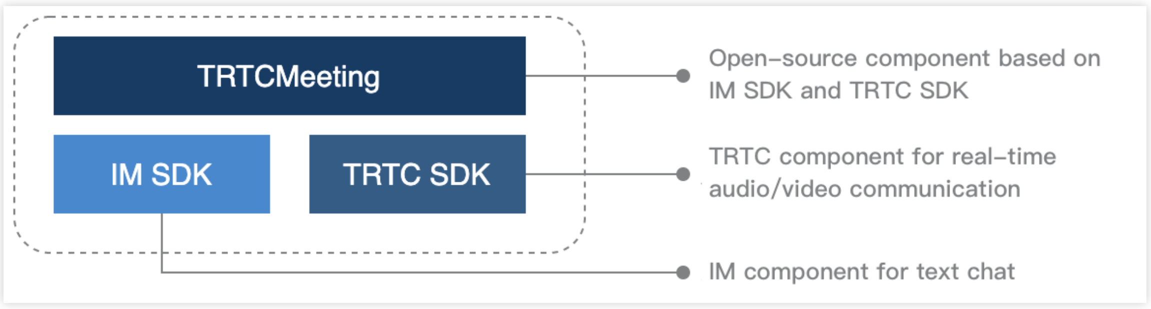
Step 5. Modify the demo source code

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 the 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 `TRTCMeeting` depends on the TRTC SDK and Chat 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

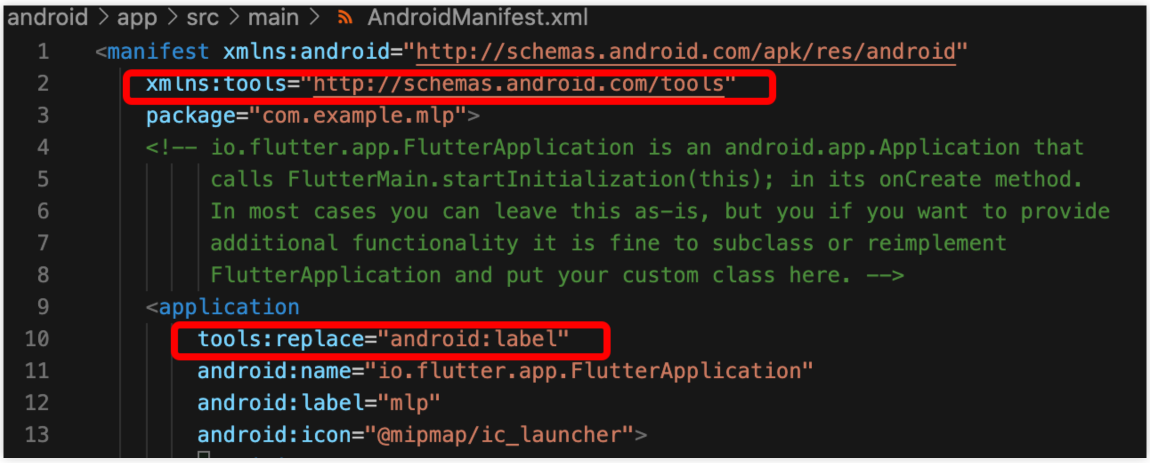
Add request for mic permission in `Info.plist` :

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

1. Open `/android/app/src/main/AndroidManifest.xml` .
2. Add `xmlns:tools="http://schemas.android.com/tools"` to `manifest` .
3. Add `tools:replace="android:label"` to `application` .

Note

Without the above steps, the Android Manifest merge failed error will occur and the compilation will fail.


android > app > src > main > AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.mlp">
    <!-- io.flutter.app.FlutterApplication is an android.app.Application that
         calls FlutterMain.startInitialization(this); in its onCreate method.
         In most cases you can leave this as-is, but you if you want to provide
         additional functionality it is fine to subclass or reimplement
         FlutterApplication and put your custom class here. -->
    <application
        tools:replace="android:label"
        android:name="io.flutter.app.FlutterApplication"
        android:label="mlp"
        android:icon="@mipmap/ic_launcher">
```

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.

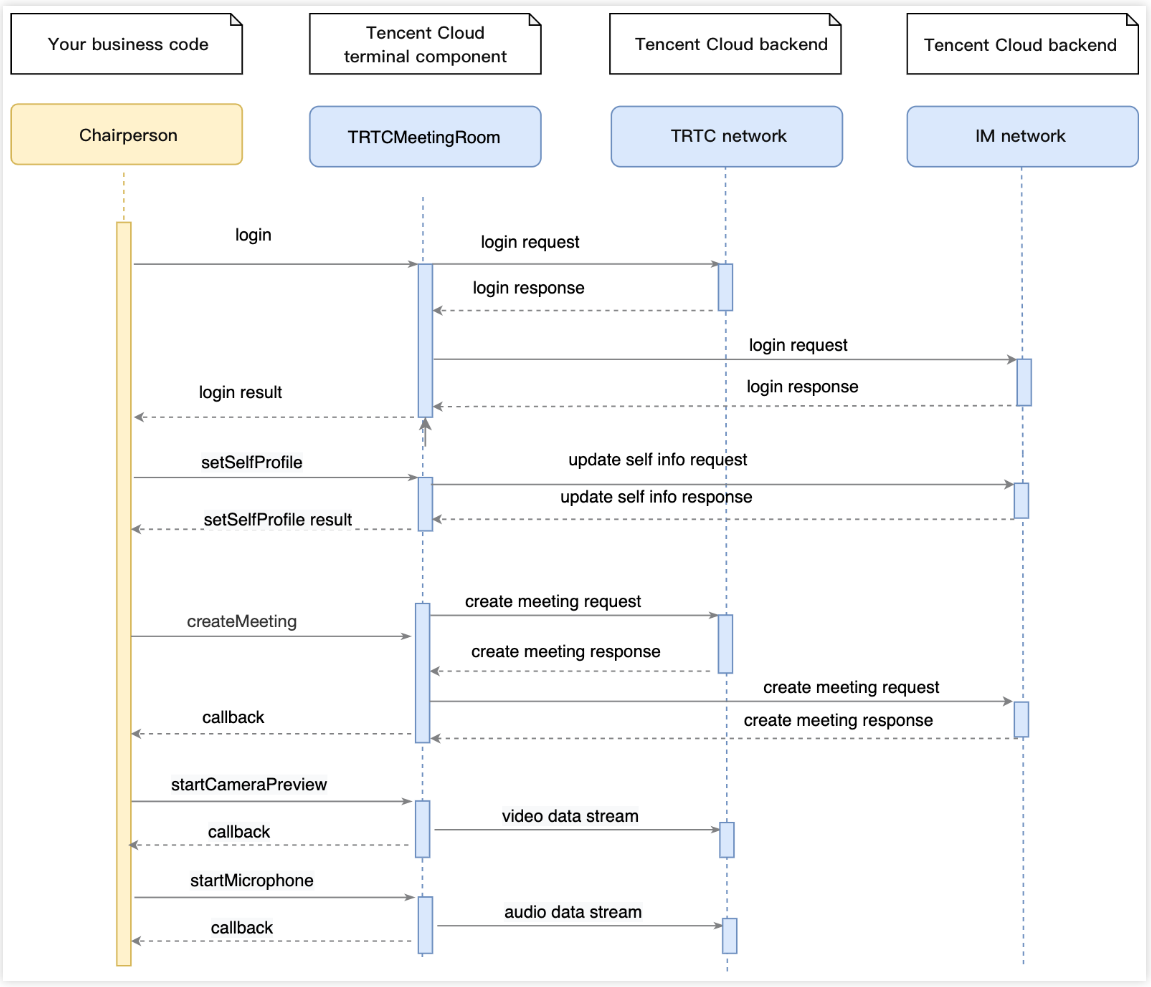
```
TRTCMeeting trtcMeeting = TRTCMeeting.sharedInstance();
trtcMeeting.registerListener(onListener);
ActionCallback res = await trtcMeeting.login(
    GenerateTestUserSig.sdkAppId,
    userId,
    GenerateTestUserSig.genTestSig(userId),
);
if (res.code == 0) {
    // Login succeeded
}
```

Step 5. Create a conference room

1. After performing step 4 to log in, call `setSelfProfile` to set your username and profile photo as a host.
2. Call `createMeeting` to create a meeting room.
3. Call `startCameraPreview` to capture video and `startMicrophone` 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.

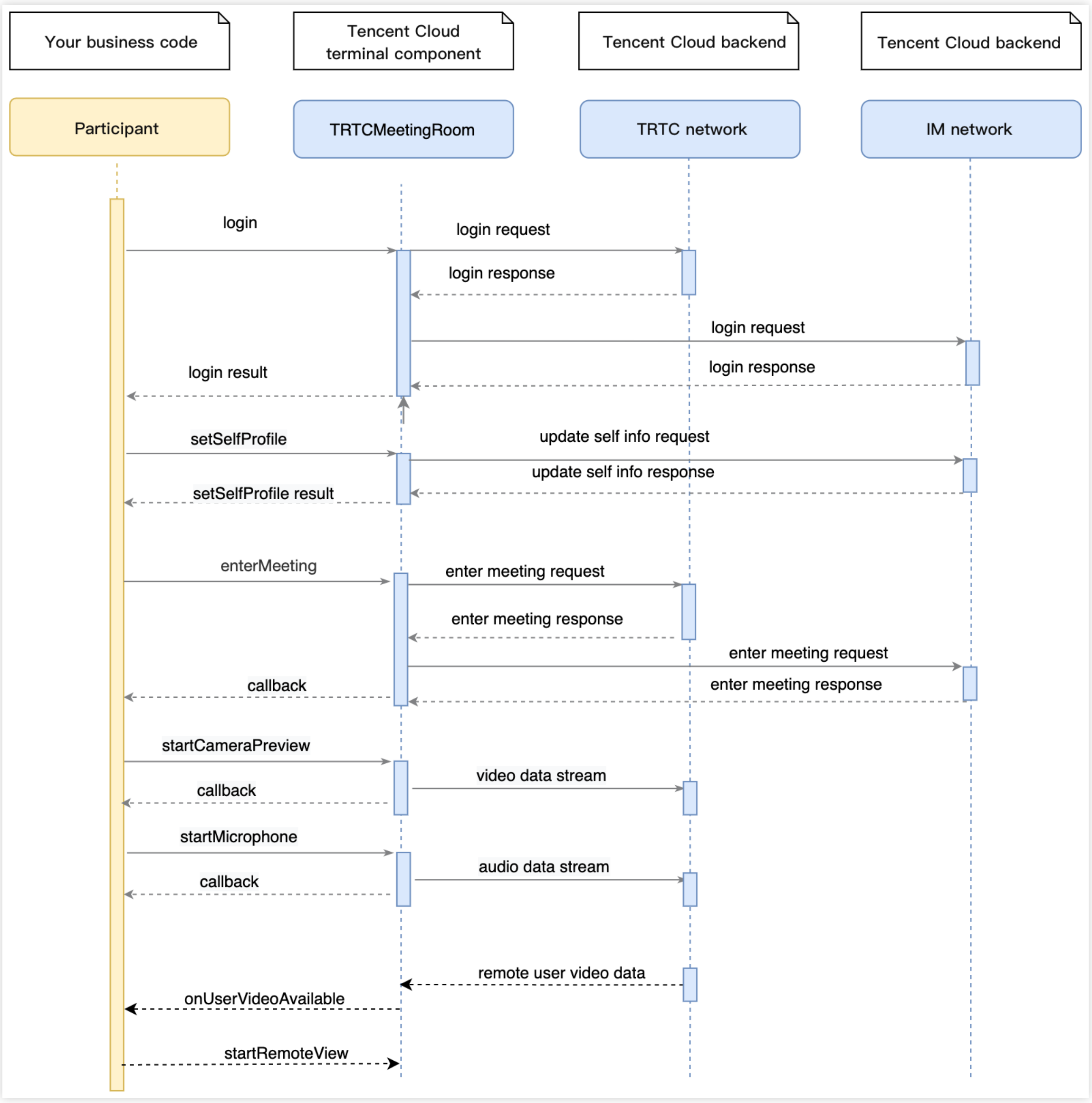

Your business code
Tencent Cloud terminal component
Tencent Cloud backend
Tencent Cloud backend
Chairperson
TRTCMeetingRoom
TRTC network
IM network
login
login request
login response
login request
login result
login response
setSelfProfile
update self info request
update self info response
setSelfProfile result
create meeting request
createMeeting
create meeting response
create meeting request
callback
create meeting response
startCameraPreview
video data stream
callback
startMicrophone
audio data stream
callback

```
// 1. Set your username and profile photo as a host
trtcMeeting.setSelfProfile('my_name', 'my_avatar');

// 2. The host creates a meeting.
ActionCallback res = await trtcMeeting.createMeeting(roomId);
if (res.code == 0) {
    // Created the meeting successfully
    // 3. The host 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_PI
    trtcMeeting.getBeautyManager().setBeautyLevel(6);
}
```

Step 6. Join a conference 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 conference 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 attendee's video.


Your business code
Tencent Cloud terminal component
Tencent Cloud backend
Tencent Cloud backend
Participant
TRTCMeetingRoom
TRTC network
IM network
login
login request
login response
login request
login result
login response
setSelfProfile
update self info request
update self info response
setSelfProfile result
enterMeeting
enter meeting request
enter meeting response
enter meeting request
callback
enter meeting response
startCameraPreview
video data stream
callback
startMicrophone
audio data stream
callback
remote user video data
onUserVideoAvailable
startRemoteView

```
// 1. Set your username and profile photo as a participant.
trtcMeeting.setSelfProfile('my_name', 'my_avatar');

// 2. Call `enterMeeting` to enter the meeting room.
ActionCallback res = await trtcMeeting.enterMeeting(roomId);
if (res.code == 0) {
    // Joined the meeting successfully
    // 3. The host 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_PI
    trtcMeeting.getBeautyManager().setBeautyLevel(6);
}

// 5. Receive the notification that another member enabled the camera and start pla
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` and pass in encoding parameters and the floating window during screen recording to start screen sharing. For more information, see TRTC SDK.
3. Other members in the room 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, please 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. Chat has its default content moderation rules. Text chat messages that contain restricted terms will not be forwarded by the cloud.

```
// Sender: Sends text chat messages
trtcMeeting.sendRoomTextMsg('Hello Word!');
// Receiver: Listens for text chat messages
trtcMeeting.registerListener(onListener);
onListener(TRTCMeetingDelegate type, param) {
    switch (type) {
        case TRTCMeetingDelegate.onRecvRoomTextMsg:
            print('Received a message from' + param['userName'] + ':' + param['mess
            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
// For example, use "CMD_MUTE_AUDIO" to indicate muting notifications
trtcMeeting.sendRoomCustomMsg('CMD_MUTE_AUDIO', '1');
// Receiver: Listens 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'] + '
                trtcMeeting.muteLocalAudio(message == '1');
            }
            break;
    }
}
```

Integrating TUIRoom (Windows and macOS)

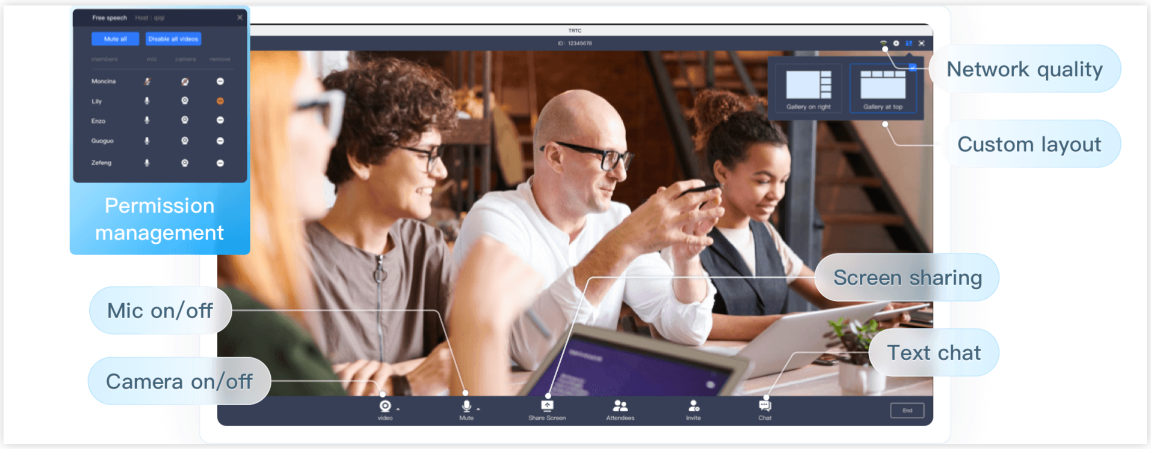
Last updated：2023-10-13 11:30:48

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 application for Windows or macOS to try out the component.

Note

All TUIKit components are based on Tencent Cloud's TRTC and Chat services. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will also be activated automatically. For Chat billing details, see Pricing.

Demo UI



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 Chat SDKs to implement basic logic control and provides APIs for you to call features easily.

Connection Guide

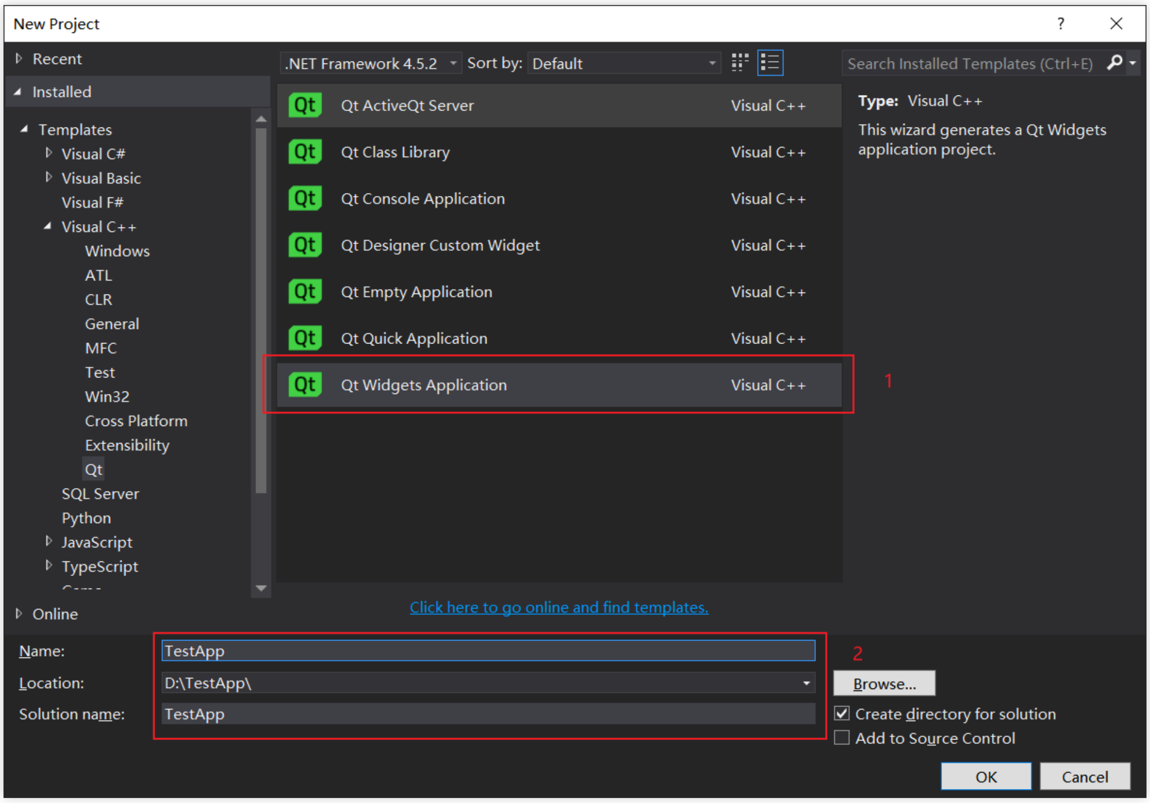
Two connection methods are recommended to help you quickly connect to the group audio/video room feature. You can select an appropriate method for secondary development.
Starting via external process
Customizing your own UI

Preparing the environment

Windows environment :
Integrated development environment: Visual Studio 2015 or later.
Qt 5.9.1 or later.
Qt Visual Studio Tools 2.2.0 or later.
Operating system: Windows 8 or later.
Make sure that you can develop the project normally in the integrated development environment.
macOS environment:
Qt 5.9.1 or later.
QtCreator integrated development environment. To use QtCreator, select it when installing Qt, and its version is the same as that of the Qt official installation package.
Make sure that you can develop the project normally in the QtCreator integrated development environment.

Starting via external process

1. **Compile the RoomApp program**.
The method of using an external process to start RoomApp depends on the executable program of the original RoomApp, which needs to be compiled in advance.
Go to RoomApp, clone the source code, and configure the project to compile and generate RoomApp.
2. **Create the `TestApp` project**.
Windows
macOS
1. Open Visual Studio, select the **Qt Widgets Application** project type, and create the `TestApp` project.



New Project

- Recent
- Installed
  - Templates
    - Visual C#
    - Visual Basic
    - Visual F#
    - Visual C++
      - Windows
      - ATL
      - CLR
      - General
      - MFC
      - Test
      - Win32
      - Cross Platform
      - Extensibility
      - Qt
    - SQL Server
    - Python
    - JavaScript
    - TypeScript
- Online

.NET Framework 4.5.2 Sort by: Default

Search Installed Templates (Ctrl+E)

Qt ActiveQt Server	Visual C++
Qt Class Library	Visual C++
Qt Console Application	Visual C++
Qt Designer Custom Widget	Visual C++
Qt Empty Application	Visual C++
Qt Quick Application	Visual C++
Qt Widgets Application	Visual C++

1

Type: Visual C++

This wizard generates a Qt Widgets application project.

Click here to go online and find templates.

Name: TestApp
Location: D:\TestApp\
Solution name: TestApp

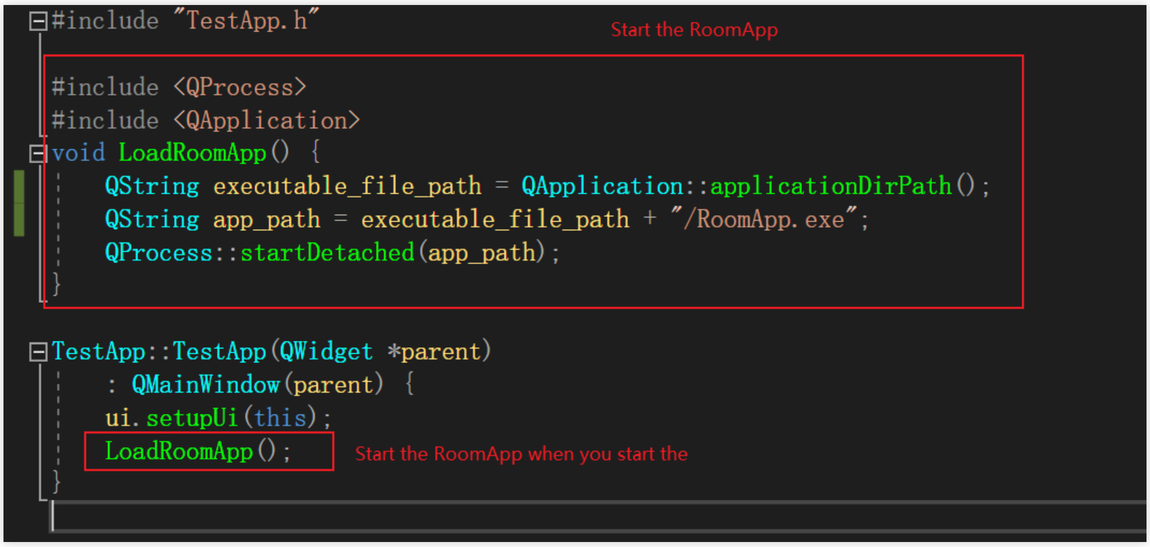
2

Browse...
☑ Create directory for solution
☐ Add to Source Control

OK Cancel

2. Write the process starting program and call the `LoadRoomApp` function at an appropriate position.

```
#include <QProcess>
#include <QApplication>
void LoadRoomApp() {
 QString executable_file_path = QApplication::applicationDirPath();
 QString app_path = executable_file_path + "/RoomApp.exe";
 QProcess::startDetached(app_path);
}
```



```
#include "TestApp.h"

#include <QProcess>
#include <QApplication>
void LoadRoomApp() {
    QString executable_file_path = QApplication::applicationDirPath();
    QString app_path = executable_file_path + "/RoomApp.exe";
    QProcess::startDetached(app_path);
}

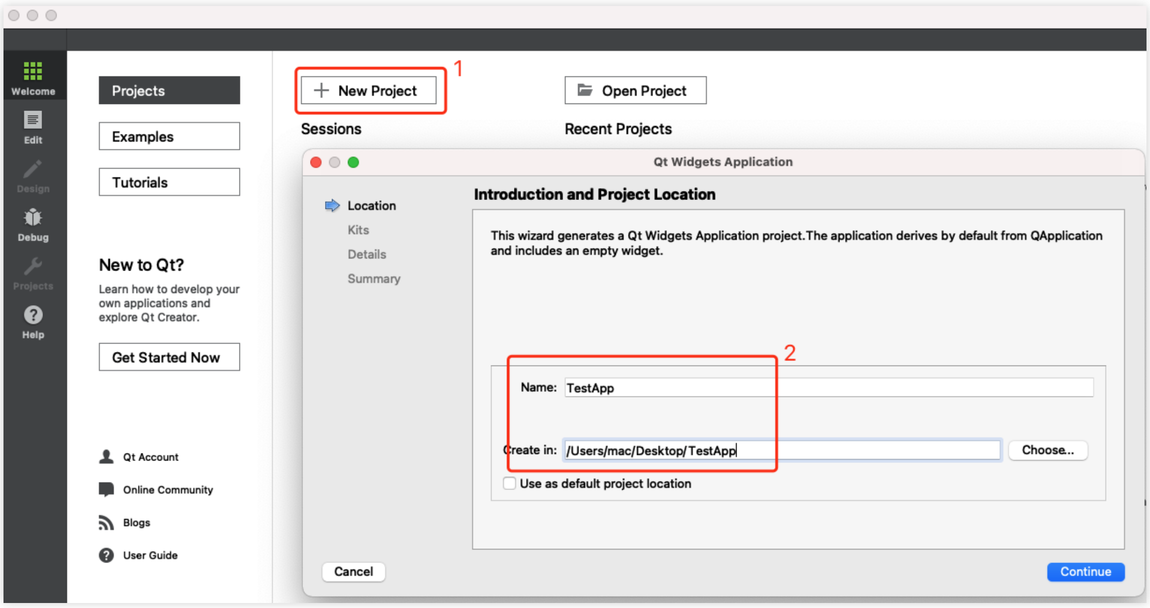
TestApp::TestApp(QWidget *parent)
    : QMainWindow(parent) {
    ui.setupUi(this);
    LoadRoomApp();
}
```

3. Compile the project and copy the RoomApp compilation result to the directory of the current executable program. Here, a `release x86` program is taken as an example:

Copy all files in the `TUIRoom\\Windows-Mac\\RoomApp\\bin\\Win32\\Release` directory to the current program directory.

4. Run the program to start TestApp and RoomApp at the same time.

1. Open QtCreator, select the **Qt Widgets Application** project, and create the `TestApp` project.



Introduction and Project Location

This wizard generates a Qt Widgets Application project.The application derives by default from QApplication and includes an empty widget.

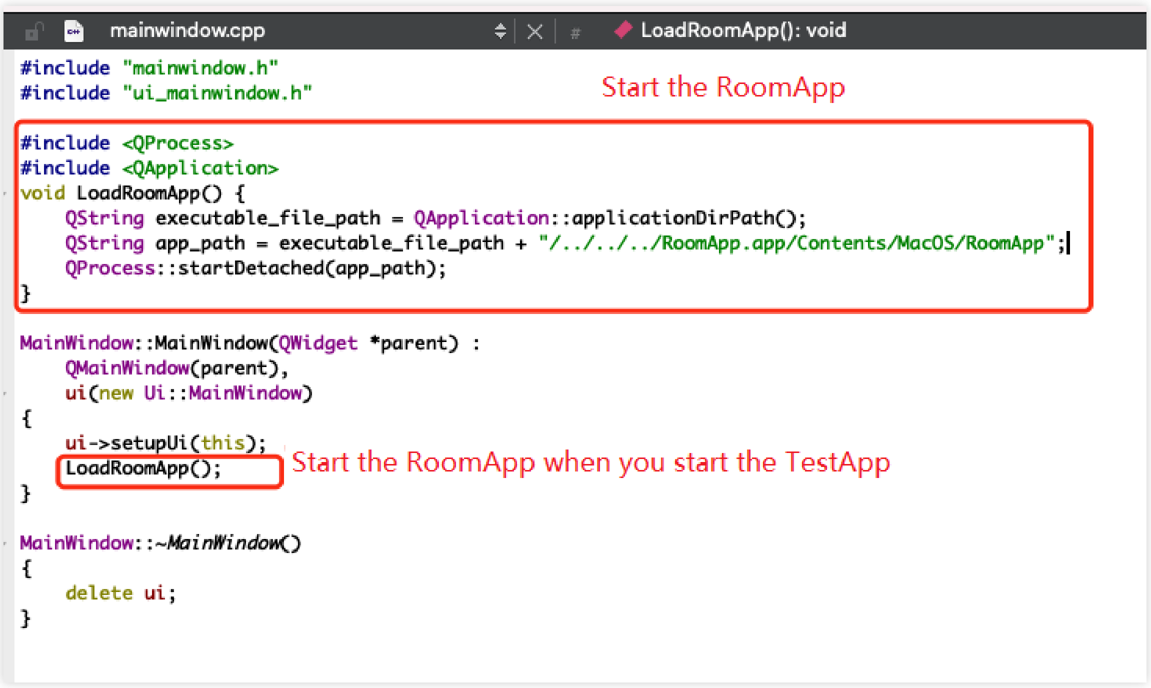
Name: TestApp

Create in: /Users/mac/Desktop/TestApp

Use as default project location

2. Write the process starting program and call the `LoadRoomApp` function at an appropriate position.

```
#include <QProcess>
#include <QApplication>
void LoadRoomApp() {
 QString executable_file_path = QApplication::applicationDirPath();
 QString app_path = executable_file_path + "/../../../RoomApp.app/Contents/MacOS/Ro
 QProcess::startDetached(app_path);
}
```



```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <QProcess>
#include <QApplication>
void LoadRoomApp() {
    QString executable_file_path = QApplication::applicationDirPath();
    QString app_path = executable_file_path + "/../../../RoomApp.app/Contents/MacOS/RoomApp";
    QProcess::startDetached(app_path);
}

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    LoadRoomApp();
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

3. Compile the project and copy the RoomApp compilation result `RoomApp.app` to the same level as the output of the current project. Here, the path of the project created in the above figure is taken as an example:
Copy `RoomApp.app` to the `/Users/mac/Desktop/TestApp/build-TestApp-Desktop_Qt_5_9_1_clang_64bit-Release` directory.
4. Run the program to start TestApp and RoomApp at the same time.

Customizing your own UI

You can modify the application we provide and adapt it to your needs. You can also use the `Module` module in the application source code to customize your own UI.
The `Module` module in the source code encapsulates the TRTC and Chat 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.
The `App` directory contains UI design and logic. You can modify the RoomApp source code for secondary development. The main features are described below:

Feature	File Directory
Homepage login	Windows-Mac\\RoomApp\\App\\LoginViewController.cpp
Device testing	Windows-Mac\\RoomApp\\App\\PresetDeviceController.cpp
Main UI	Windows-Mac\\RoomApp\\App\\MainWindow.cpp
Speaker list	Windows-Mac\\RoomApp\\App\\StageListController.cpp
Member lists	Windows-Mac\\RoomApp\\App\\MemberListViewController.cpp

Settings page	Windows-Mac\\RoomApp\\App\\SettingViewController.cpp
Chat room	Windows-Mac\\RoomApp\\App\\ChatRoomViewController.cpp
Screen sharing	Windows-Mac\\RoomApp\\App\\ScreenShareWindow.cpp
Bottom toolbar	Windows-Mac\\RoomApp\\App\\BottomBarController.cpp

Suggestions and Feedback

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

Integrating TUIRoom (Electron)

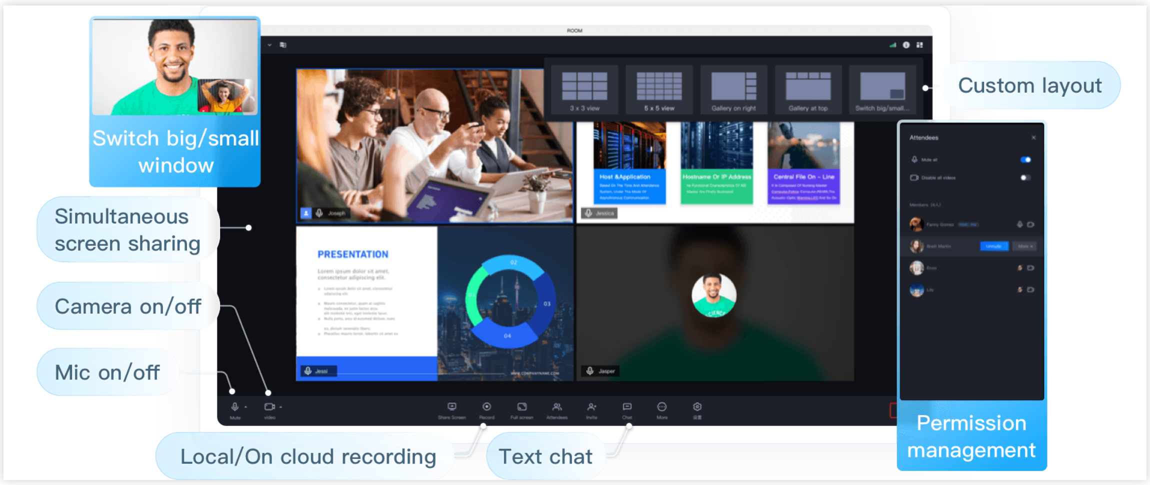
Last updated：2023-10-13 11:31:31

Overview

TUIRoom is an open-source audio/video component that comes with a UI kit. It allows you to quickly implement features including audio/video room, screen sharing, and chat messages into your project.

Note

All components of TUIKit use two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will be activated automatically. For the billing details of Chat, see Pricing.



You can download the macOS or Windows edition of our TUIRoom Electron demo to try out more features.
You can also download the code for TUIRoom and refer to this document to quickly implement a TUIRoom demo project.
This document shows you how to integrate the TUIRoom Electron component into your existing project.

Integration

The TUIRoom component is developed using Vue 3 + TypeScript + Pinia + Element Plus + SCSS, so your project must be based on Electron + Vue 3 + TypeScript.

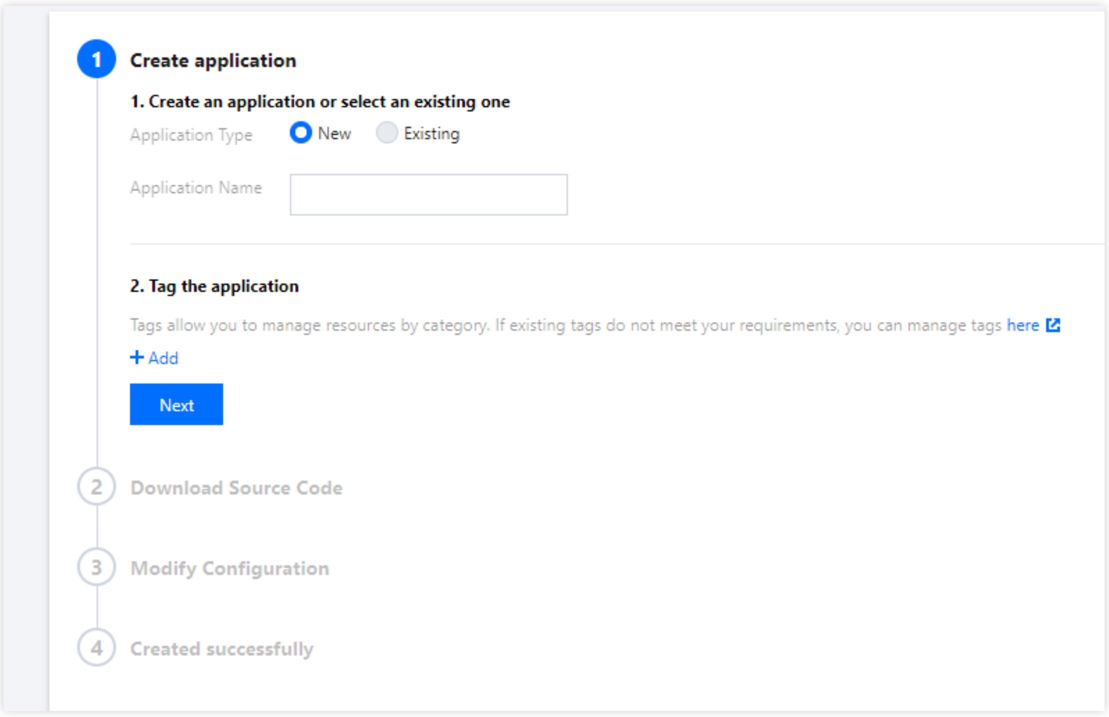
Step 1. Activate the TRTC service

TUIRoom is based on TRTC and Chat.

1. **Create a TRTC application**

Sign up for a Tencent Cloud account and complete identity verification.

In the TRTC console, click **Application Management** on the left sidebar and then click **Create Application**.



Create application

1. Create an application or select an existing one

Application Type [✓ New] Existing

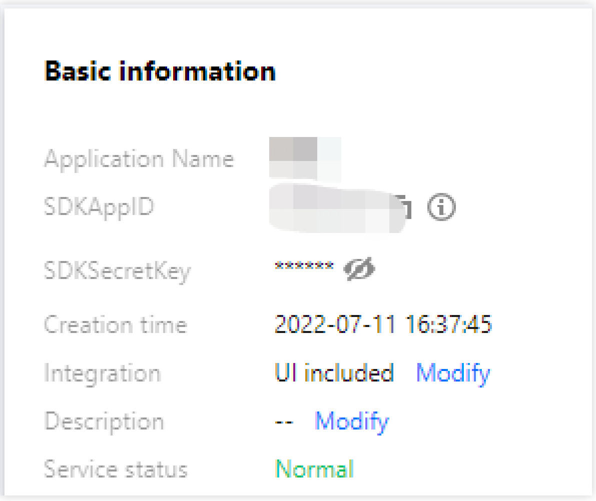
Application Name

2. Tag the application

Tags allow you to manage resources by category. If existing tags do not meet your requirements, you can manage tags here

2. **Get the SDKAppID and key**

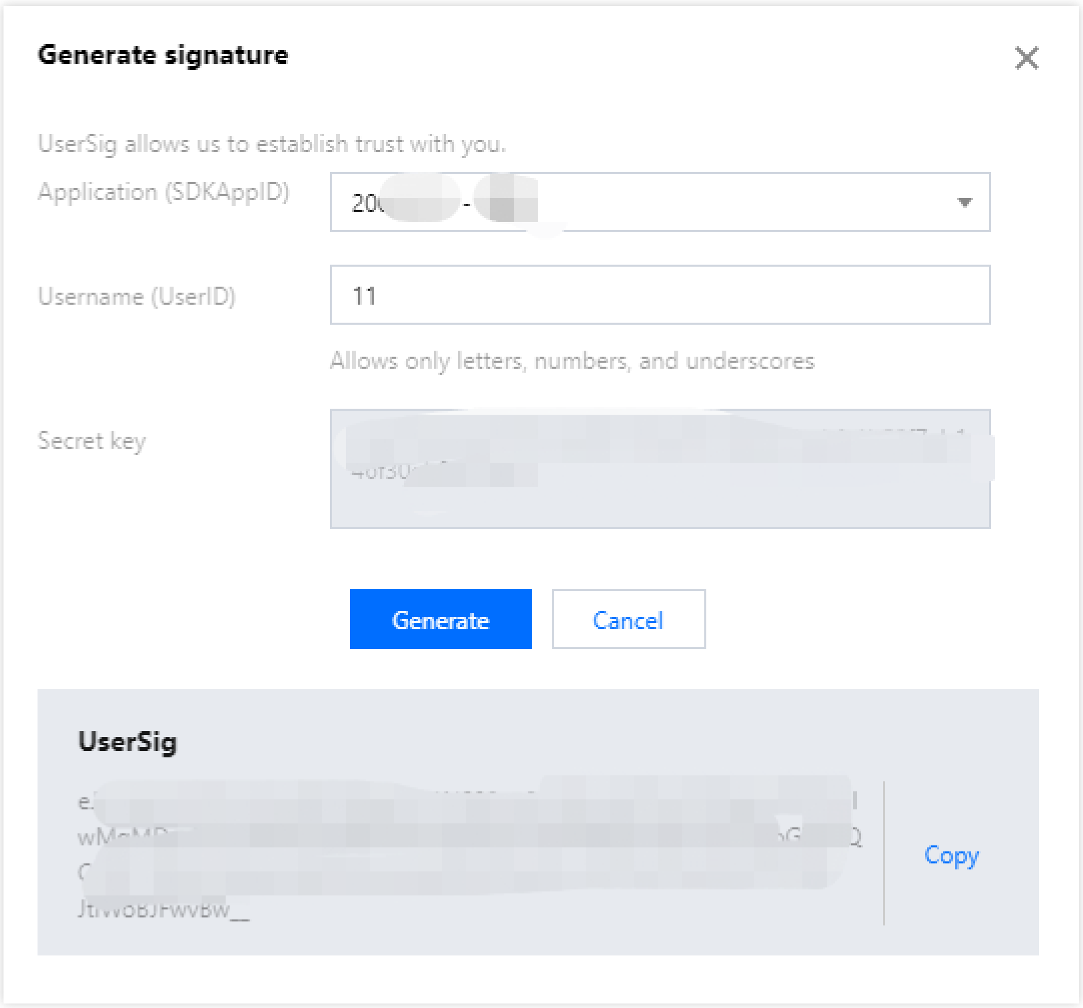
1. On the **Application Management** page, find the application you created, and click **Application Info** to view its `SDKAppID` (different applications cannot communicate with each other).



Basic information

Application Name	
SDKAppID	
SDKSecretKey	******
Creation time	2022-07-11 16:37:45
Integration	UI included Modify
Description	-- Modify
Service status	Normal

2. Select the **Quick Start** tab to view the application's secret key. Each `SDKAppID` corresponds to a secret key. They are used to generate the signature (`UserSig`) required to legitimately use TRTC services.



Generate signature

UserSig allows us to establish trust with you.

Application (SDKAppID) 20[redacted]

Username (UserID) 11

Allows only letters, numbers, and underscores

Secret key [redacted]

UserSig

[redacted]

3. **Generate UserSig** `UserSig` is a security signature designed by Tencent Cloud to prevent attackers from accessing your Tencent Cloud account. It is required when you initialize the TUIRoom component.

How do I calculate UserSig for debugging?

How do I calculate UserSig for production?

Step 2. Download and copy the TUIRoom component

1. Open an existing Electron + Vue3 + TypeScript project. If you don't have one, you can use this sample to create a project.

Caution

The steps in this document are based on electron-vite-vue 1.0.0.

We have updated the directory structure of electron-vite-vue. If you use the latest version, some of the paths and configuration described in this document may not apply.

2. After the template project is successfully generated, run the following script:

```
cd electron-vite-vue
npm install
npm run dev
```

3. Clone or download the TUIRoom code, and copy the `TUIRoom/Electron/packages/renderer/src/TUIRoom` folder to `packages/renderer/src/` of your project.

Step 3. Import the TUIRoom component

1. Import the TUIRoom component into your webpage, such as `App.vue` .

The TUIRoom component classifies users as hosts and participants and offers APIs including init, createRoom, and enterRoom.

Hosts and participants can call init to initialize application and user data. Hosts can call createRoom to create and enter rooms. Participants can call enterRoom to join the rooms created by hosts.

```
<template>
 <room ref="TUIRoomRef"></room>
</template>
```

```
<script setup lang="ts">
 import { ref, onMounted } from 'vue';
 // Import the TUIRoom component. Be sure to use the correct import path.
 import Room from './TUIRoom/index.vue';
 // Get the TUIRoom component elements used to call the component's APIs
 const TUIRoomRef = ref();

  onMounted(async () => {
  // Initialize the TUIRoom component
  // A host needs to initialize the TUIRoom component before creating a room
  // A participant needs to initialize the TUIRoom component before entering a room
  await TUIRoomRef.value.init({
      // Get the `SDKAppID` (see step 1)
      sdkAppId: 0,
      // The user's unique ID in your business
      userId: '',
      // For local development and debugging, you can quickly generate a `UserSig`
      userSig: '',
      // The user's username in your business
      userName: '',
      // The URL of the user's profile photo in your business
      userAvatar: '',
      // The user's unique ID used for screen sharing. It must be in the format of
      shareUserId: '',
      // Refer to steps 1-3 above and use the `SDKAppID` and `shareUserId` to gener
      shareUserSig: '',
  })
   // By default, a room is created at this point. During actual implementation, yo
  await handleCreateRoom();
 })

 // The host creates a room. Call this API only when you need to create a room.
 async function handleCreateRoom() {
  // `roomId` is the ID of the room to enter, which must be a number.
  // The valid values of `roomMode` are `FreeSpeech` (free speech mode) and `ApplyS
  // `roomParam` specifies whether to turn on the mic/camera upon room entry, as we
  const roomId = 123456;
  const roomMode = 'FreeSpeech';
  const roomParam = {
      isOpenCamera: true,
      isOpenMicrophone: true,
  }
  await TUIRoomRef.value.createRoom(roomId, roomMode, roomParam);
 }

 // The participant enters a room. This API is called by a participant to join an e
 async function handleEnterRoom() {
    // `roomId` is the ID of the room to enter, which must be a number.
    // `roomParam` specifies whether to turn on the mic/camera upon room entry, as we
    const roomId = 123456;
    const roomParam = {
        isOpenCamera: true,
        isOpenMicrophone: true,
    }
    await TUIRoomRef.value.enterRoom(roomId, roomParam);
  }
</script>

<style>
html, body {
  width: 100%;
  height: 100%;
  margin: 0;
}

#app {
  width: 100%;
  height: 100%;
}
</style>
```

Note

Copy the above code to your webpage and replace the parameter values for the APIs with the actual values.

Step 4. Set up the development environment

After the TUIRoom component is imported, to ensure that the project can run successfully, complete the following configuration:

1. **Install dependencies**

Install development environment dependencies:

```
npm install sass typescript unplugin-auto-import unplugin-vue-components -S -D
```

Install production environment dependencies:

```
npm install element-plus events mitt pinia trtc-electron-sdk tim-js-sdk tsignaling
```

2. **Register Pinia**

TUIRoom uses Pinia for room data management. You need to register Pinia in the project entry file `packages/renderer/src/main.ts`.

```
// `src/main.ts` file
import { createPinia } from 'pinia';

const app = createApp(App);
// Register Pinia
createApp(App)
  .use(createPinia())
  .mount('#app')
  .$nextTick(window.removeLoading)
```

3. **Import Element Plus components**

TUIRoom uses Element Plus UI components, which you need to import in `packages/renderer/vite.config.ts` . You can manually import only the components you need.

Caution

Add the code below in the file. Do not delete the existing configuration.

```
// vite.config.ts
import AutoImport from 'unplugin-auto-import/vite';
import Components from 'unplugin-vue-components/vite';
import { ElementPlusResolver } from 'unplugin-vue-components/resolvers';
const path = require('path');
```

```
export default defineConfig({
    // ...
    plugins: [
        AutoImport({
            resolvers: [ElementPlusResolver()],
        }),
        Components({
            resolvers: [ElementPlusResolver({
                importStyle: 'sass',
            })],
        }),
    ],
    css: {
    preprocessorOptions: {
    scss: {
        additionalData: `
          @use '${path.resolve(__dirname, 'src/TUIRoom/assets/style/element.scss')}
        `,
      },
    },
  },
});
```

Meanwhile, in order to ensure that Element Plus UI components can display styles properly, you need to load Element Plus component styles in the entry file `packages/renderer/src/main.ts`.

```
// src/main.ts
import 'element-plus/theme-chalk/el-message.css'
import 'element-plus/theme-chalk/el-message-box.css'
```

4. **Import trtc-electron-sdk**

In order to import `trtc-electron-sdk` using the import statement at the UI layer, you need to configure `packages/renderer/vite.config.ts` as follows (otherwise, you will have to use the require statement):

Caution

Replace the configuration in `resolve` with the following:

```
// vite.config.ts

export default defineConfig({
 // ...
 plugins: [
 resolve(
   {
     "trtc-electron-sdk": `
       const TRTCCloud = require("trtc-electron-sdk");
       const TRTCParams = TRTCCloud.TRTCParams;
       const TRTCAppScene = TRTCCloud.TRTCAppScene;
```

```
        const TRTCVideoStreamType = TRTCCloud.TRTCVideoStreamType;
        const TRTCScreenCaptureSourceType = TRTCCloud.TRTCScreenCaptureSourceType;
        const TRTCVideoEncParam = TRTCCloud.TRTCVideoEncParam;
        const Rect = TRTCCloud.Rect;
        const TRTCAudioQuality = TRTCCloud.TRTCAudioQuality;
        const TRTCScreenCaptureSourceInfo = TRTCCloud.TRTCScreenCaptureSourceInfo;
        const TRTCDeviceInfo = TRTCCloud.TRTCDeviceInfo;
        const TRTCVideoQosPreference = TRTCCloud.TRTCVideoQosPreference;
        const TRTCQualityInfo = TRTCCloud.TRTCQualityInfo;
        const TRTCStatistics = TRTCCloud.TRTCStatistics;
        const TRTCVolumeInfo = TRTCCloud.TRTCVolumeInfo;
        const TRTCDeviceType = TRTCCloud.TRTCDeviceType;
        const TRTCDeviceState = TRTCCloud.TRTCDeviceState;
        const TRTCBeautyStyle = TRTCCloud.TRTCBeautyStyle;
        const TRTCVideoResolution = TRTCCloud.TRTCVideoResolution;
        const TRTCVideoResolutionMode = TRTCCloud.TRTCVideoResolutionMode;
        const TRTCVideoMirrorType = TRTCCloud.TRTCVideoMirrorType;
        const TRTCVideoRotation = TRTCCloud.TRTCVideoRotation;
        const TRTCVideoFillMode = TRTCCloud.TRTCVideoFillMode;
        export {
          TRTCParams,
          TRTCAppScene,
          TRTCVideoStreamType,
          TRTCScreenCaptureSourceType,
          TRTCVideoEncParam,
          Rect,
          TRTCAudioQuality,
          TRTCScreenCaptureSourceInfo,
          TRTCDeviceInfo,
          TRTCVideoQosPreference,
          TRTCQualityInfo,
          TRTCStatistics,
          TRTCVolumeInfo,
          TRTCDeviceType,
          TRTCDeviceState,
          TRTCBeautyStyle,
          TRTCVideoResolution,
          TRTCVideoResolutionMode,
          TRTCVideoMirrorType,
          TRTCVideoRotation,
          TRTCVideoFillMode,
        };
        export default TRTCCloud.default;
      `,
    }
  ),
  ]
  // ...
});
```

5. **Configure `env.d.ts`**

Configure the `env.d.ts` file in `packages/renderer/src/env.d.ts` as follows:

Caution

Add the code below in `env.d.ts`. Do not delete the existing configuration in the file.

```
// env.d.ts

declare module 'tsignaling/tsignaling-js' {
  import TSignaling from 'tsignaling/tsignaling-js';
  export default TSignaling;
}

declare module 'tim-js-sdk' {
  import TIM from 'tim-js-sdk';
  export default TIM;
}
```

6. **If there are dynamic imports in your project, you need to modify the build configuration to generate an ES module.**

Modify the configuration in `packages/renderer/vite.config.ts` as follows.

Caution

Add the code below in the file. Do not delete the existing Vite configuration. Skip this step if your project does not have dynamic imports.

```
// vite.config.ts

export default defineConfig({
    // ...
    build: {
        rollupOptions: {
          output: {
              format: 'es'
          }
        }
    },
});
```

Step 5. Run your project in the development environment

In the console, execute the development environment script. Then, open the page integrated with the TUIRoom component with a browser.

If you used the script in step 2 to generate an Electron + Vue3 + TypeScript project, follow the steps below:

1. Run the development environment command.

```
npm run dev
```

Caution

Because Element Plus components are imported manually, it may take a relatively long time for the page to load in the development environment for the first time. This will not be an issue after building.

2. Try out the features of the TUIRoom component.

Step 6. Create an installer and run it

Run the following command in a terminal window to generate an installer in the `release` directory.

```
npm run build
```

Caution

You need macOS to create a macOS installer and Windows to create a Windows installer.

Appendix: TUIRoom APIs

TUIRoom APIs

init

This API is used to initialize TUIRoom data. Anyone using TUIRoom needs to call this API.

```
TUIRoomRef.value.init(roomData);
```

The parameters are described below:

Parameter	Type	Description
roomData	object	
roomData.sdkAppId	number	The SDKAppID.

roomData.userId	string	The unique user ID.
roomData.userSig	string	The UserSig.
roomData.userName	string	The username.
roomData.userAvatar	string	The user’s profile photo.
roomData.shareUserId	string	The `UserID` used for screen sharing, which must be in the format of `share_${userId}` . You don’t need to pass this parameter if you don’t need the screen sharing feature.
roomData.shareUserSig	string	The `UserSig` used for screen sharing, which is optional.

createRoom

This API is used by a host to create a room.

```
TUIRoomRef.value.createRoom(roomId, roomMode, roomParam);
```

The parameters are described below:

Parameter	Type	Description
roomId	number	The room ID.
roomMode	string	The speech mode, including `FreeSpeech` (free speech) and `ApplySpeech` (request-to-speak). The default value is `FreeSpeech` , which is the only supported mode currently.

roomParam	Object	Optional
roomParam.isOpenCamera	string	Whether to turn on the camera upon room entry. This parameter is optional and the default is no.
roomParam.isOpenMicrophone	string	Whether to turn on the mic upon room entry. This parameter is optional and the default is no.
roomParam.defaultCameraId	string	The ID of the default camera, which is optional.
roomParam.defaultMicrophoneId	string	The ID of the default mic, which is optional.
roomParam.defaultSpeakerId	String	The ID of the default speaker, which is optional.

enterRoom

This API is used by a participant to enter a room.

```
TUIRoomRef.value.enterRoom(roomId, roomParam);
```

The parameters are described below:

Parameter	Type	Description
roomId	number	The room ID.
roomParam	Object	Optional
roomParam.isOpenCamera	string	Whether to turn on the camera upon room entry. This

		parameter is optional and the default is no.
roomParam.isOpenMicrophone	string	Whether to turn on the mic upon room entry. This parameter is optional and the default is no.
roomParam.defaultCameraId	string	The ID of the default camera, which is optional.
roomParam.defaultMicrophoneId	string	The ID of the default mic, which is optional.
roomParam.defaultSpeakerId	String	The ID of the default speaker, which is optional.

TUIRoom events

onRoomCreate

A room was created.

```
<template>
  <room ref="TUIRoomRef" @on-room-create="handleRoomCreate"></room>
</template>

<script setup lang="ts">
  // Import the TUIRoom component. Be sure to use the correct import path.
  import Room from './TUIRoom/index.vue';

  function handleRoomCreate(info) {
    if (info.code === 0) {
      console.log('Room created successfully')
    }
  }
</script>
```

onRoomEnter

A user entered the room.

```
<template>
  <room ref="TUIRoomRef" @on-room-enter="handleRoomEnter"></room>
</template>
```

```
<script setup lang="ts">
  // Import the TUIRoom component. Be sure to use the correct import path.
  import Room from './TUIRoom/index.vue';

  function handleRoomEnter(info) {
    if (info.code === 0) {
      console.log('Entered room successfully')
    }
  }
</script>
```

onRoomDestory

The host closed the room.

```
<template>
  <room ref="TUIRoomRef" @on-room-destory="handleRoomDestory"></room>
</template>

<script setup lang="ts">
  // Import the TUIRoom component. Be sure to use the correct import path.
  import Room from './TUIRoom/index.vue';

  function handleRoomDestory(info) {
    if (info.code === 0) {
      console.log('The host closed the room successfully')
    }
  }
</script>
```

onRoomExit

A participant left the room.

```
<template>
  <room ref="TUIRoomRef" @on-room-exit="handleRoomExit"></room>
</template>
```

```
<script setup lang="ts">
  // Import the TUIRoom component. Be sure to use the correct import path.
  import Room from './TUIRoom/index.vue';

  function handleRoomExit(info) {
    if (info.code === 0) {
      console.log('The participant exited the room successfully')
    }
  }
</script>
```

TUIRoom APIs

TUIRoom (Android)

Last updated：2023-10-13 11:32:37

TUIRoom is based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat. With TUIRooom:

A host can create a room, and users can enter the room ID to join the room.

Participants can share their screens with each other.

All room members can send text chat messages and custom messages.

Note

All components of TUIKit use two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will be activated automatically. For the billing details of Chat, see Pricing.

TUIRoom is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, see Integrating TUIRoom (Android).

The TRTC SDK is used as a low-latency audio/video room component.

The Chat SDK (**Android edition**) is used to implement chat messages.

TUIRoom API Overview

TUIRoomCore Basic APIs

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 host).
destroyRoom	Closes the room (called by host).

enterRoom	Enters a room (called by participant)
LeaveRoom	Leaves a room (called by participant).
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 host permissions to another user (called by host).

Local audio/video operation APIs

API	Description
startCameraPreview	Enables preview of the local video.
stopCameraPreview	Stops local video capturing and preview.
startLocalAudio	Enables mic capturing.
stopLocalAudio	Stops mic capturing.
setVideoMirror	Sets the mirror mode for local video preview.
setSpeaker	Sets whether to play sound from the device’s 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
muteUserMicrophone	Enables/Disables the mic of a specified user.
muteAllUsersMicrophone	Enables/Disables the mics of all users and syncs the status to room information.
muteUserCamera	Enables/Disables the camera of a specified user.
muteAllUsersCamera	Enables/Disables the cameras of all users and syncs the status to room information.
muteChatRoom	Disables/Enables chat messages (called by host).
kickOffUser	Removes a specified user from the room (called by host).
startCallingRoll	Starts a roll call (called by host).
stopCallingRoll	Stops a roll call (called by host).
replyCallingRoll	Replies to a roll call (called by participant).
sendSpeechInvitation	Sends a speech invitation to a participant (called by host).
cancelSpeechInvitation	Cancels a speech invitation sent to a participant (called by host).
replySpeechInvitation	Accepts/Rejects the speech invitation of the host (called by participant).
SendSpeechApplication	Sends a speech request (called by participant).
replySpeechApplication	Approves/Rejects the speech request of a participant (called by host).
forbidSpeechApplication	Disables requests to speak (called by host).
sendOffSpeaker	Stops the speech of a participant (called by host).
sendOffAllSpeakers	Stops the speech of all members in the room (called by host).
ExitSpeechState	Exits the speaker mode (called by participant).

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 QoS control parameters.

SDK version APIs

API	Description
getSDKVersion	Gets the SDK version.

TUIRoomCoreListener API Overview

Callbacks for error events

API	Description
onError	Callback for error.

Basic event callbacks

API	Description
OnDestroyRoom	The room was closed.
onUserVoiceVolume	The audio volume of a user.
onRoomMasterChanged	The host changed.

Remote user event callbacks

API	Description
onRemoteUserEnter	A remote user entered the room.
onRemoteUserLeave	A remote user exited the room.
onRemoteUserCameraAvailable	A remote user turned on/off their camera.

onRemoteUserScreenVideoAvailable	A remote user started/stopped screen sharing.
onRemoteUserAudioAvailable	A remote user turned on/off their mic.
onRemoteUserEnterSpeechState	A remote user started speaking.
onRemoteUserExitSpeechState	A remote user stopped speaking.

Message event callback APIs

API	Description
onReceiveChatMessage	A text chat message was received.
onReceiveRoomCustomMsg	A custom message was received.

Room control event callbacks

API	Description
onReceiveSpeechInvitation	A participant received a speech invitation from the host.
onReceiveInvitationCancelled	The speech invitation sent to a participant was canceled by the host.
onReceiveSpeechApplication	The host received a speech request from a participant.
onSpeechApplicationCancelled	A participant canceled a speech request.
onSpeechApplicationForbidden	The host disabled requests to speak.
onOrderedToExitSpeechState	A participant was asked to stop speaking.
onCallingRollStarted	The host started a roll call (received by participants)
onCallingRollStopped	The host stopped a roll call (received by participants).
onMemberReplyCallingRoll	A participant replied to the roll call (received by the host).
onChatRoomMuted	The host disabled/enabled chat messages.
onMicrophoneMuted	The host disabled mic use.
onCameraMuted	The host disabled camera use.
onReceiveKickedOff	The host removed a participant from the room (received by the participant).

Callback of 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.

TUIRoomCore Basic APIs

getInstance

This API is used to get a TUIRoomCore singleton object.

```
public static TUIRoomCore getInstance(Context context);
```

The parameters are described below:

Parameter	Type	Description
context	Context	Android context, which will be converted to `ApplicationContext` for the calling of system APIs.

destroyInstance

```
void destroyInstance();
```

setListener

This API is used to set the event callbacks of TUIRoomCore. You can use `TUIRoomCoreListener` to get the callbacks.

```
void setListener(TUIRoomCoreListener listener);
```

The parameters are described below:

Parameter	Type	Description
listener	TUIRoomCoreListener	The event callback class.

createRoom

This API is used to create a room (called by the host).

```
void createRoom(String roomId, TUIRoomCoreDef.SpeechMode speechMode, TUIRoomCoreCal
```

The parameters are described below:

Parameter	Type	Description
roomId	String	The room ID. You need to assign and manage the IDs in a centralized manner.
speechMode	TUIRoomCoreDef.SpeechMode	The speech mode.

callback	TUIRoomCoreCallback.ActionCallback	The result of room creation.

Generally, a host may need to call the following APIs:

1. The **host** calls `createRoom()` to create a room, the result of which is returned via `TUIRoomCoreCallback.ActionCallback`.
2. The **host** calls `startCameraPreview()` to enable camera capturing and preview.
3. The **host** calls `startLocalAudio()` to enable the local mic.

destroyRoom

This API is used to close a room (called by the host).

```
void destroyRoom(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
callback	UIRoomCoreCallback.ActionCallback	The room closing result.

enterRoom

This API is used to leave a room (called by a participant).

```
void enterRoom(String roomId, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
roomId	String	The room ID.
callback	UIRoomCoreCallback.ActionCallback	The result.

Generally, a participant joins a room in the following steps:

1. The **participant** calls `enterRoom` (passing in `roomId`) 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` callback and calls `startRemoteView()` to start playback.

leaveRoom

This API is used to leave a room (called by a participant).

```
void leaveRoom(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
callback	UIRoomCoreCallback.ActionCallback	The 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<TUIRoomCoreDef.UserInfo> getRoomUsers();
```

getUserInfo

This API is used to get the information of a specified room member.

```
void getUserInfo(String userId, TUIRoomCoreCallback.UserInfoCallback callback);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
callback	UIRoomCoreCallback.UserInfoCallback	The room member details.

setSelfProfile

Set User Info

```
void setSelfProfile(String userName, String avatarURL, TUIRoomCoreCallback.ActionCa
```

The parameters are described below:

Parameter	Type	Description
userName	String	The username.
avatarURL	String	The URL of the user profile photo.

callback	TUIRoomCoreCallback.ActionCallback	Whether the setting succeeded.

transferRoomMaster

This API is used to transfer host permissions to another user.

```
void transferRoomMaster(String userId, TUIRoomCoreCallback.ActionCallback callback
```

The parameters are described below:

Parameter	Type	Description

userId	String	The user ID.
callback	TUIRoomCoreCallback.ActionCallback	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 described below:

Parameter	Type	Description
isFront	boolean	`true` : Front camera; `false` : Rear camera
view	TXCloudVideoView	The view 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 described below:

Parameter	Type	Description
quality	int	The audio quality. Valid values: <li/>TRTC_AUDIO_QUALITY_MUSIC<li/>TRTC_AUDIO_QUALITY_DEFAULT<li/>TRTC

stopLocalAudio

This API is used to stop mic capturing.

```
void stopLocalAudio();
```

setVideoMirror

This API is used to set the mirror mode for local video preview.

```
void setVideoMirror(int type);
```

The parameters are described below:

Parameter	Type	Description
type	int	The mirror mode.

setSpeaker

This API is used to set whether to play sound from the device’s speaker or receiver.

```
void setSpeaker(boolean isUseSpeaker);
```

The parameters are described below:

Parameter	Type	Description
isUseSpeaker	boolean	true: Speaker; false: 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
```

The parameters are described below:

Parameter	Type	Description
userId	String	The ID of the user whose video image is to be played back.

view	TXCloudVideoView	The view that loads video images
streamType	TUIRoomCoreDef.SteamType	The stream type.
callback	TUIRoomCoreCallback.ActionCallback	The 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 described below:

Parameter	Type	Description
userId	String	The ID of the user whose video image is to be stopped.
callback	TUIRoomCoreCallback.ActionCallback	The result.

switchCamera

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

```
void switchCamera(boolean isFront);
```

The parameters are described below:

Parameter	Type	Description
isFront	boolean	true: Front camera; false: Rear camera.

Message Sending APIs

sendChatMessage

This API is used to broadcast a text chat message in the room.

```
void sendChatMessage(String message, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description

message	String	The message content.
callback	TUIRoomCoreCallback.ActionCallback	The result.

sendCustomMessage

This API is used to send a custom message.

```
void sendCustomMessage(String data, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
data	String	The message content.
callback	TUIRoomCoreCallback.ActionCallback	The result.

Room Control APIs

muteUserMicrophone

This API is used to enable/disable the mic of the specified user.

```
void muteUserMicrophone(String userId, boolean mute, TUIRoomCoreCallback.ActionCall
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	The result.

muteAllUsersMicrophone

This API is used to enable/disable the mics of all users.

```
void muteAllUsersMicrophone(boolean mute, TUIRoomCoreCallback.ActionCallback callba
```

The parameters are described below:

Parameter	Type	Description
mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	The result.

muteUserCamera

This API is used to enable/disable the camera of the specified user.

```
void muteUserCamera(String userId, boolean mute, TUIRoomCoreCallback.ActionCallback
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.

mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	The result.

muteAllUsersCamera

This API is used to enable/disable the cameras of all users.

```
void muteAllUsersCamera(boolean mute, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

muteChatRoom

This API is used to disable/enable chat messages.

```
void muteChatRoom(boolean mute, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
mute	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	The result.

kickOffUser

This API is used by the host to remove a member from the room.

```
void kickOffUser(String userId, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
callback	TUIRoomCoreCallback.ActionCallback	The result.

startCallingRoll

This API is used by the host to start a roll call.

```
void startCallingRoll(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	The result.

stopCallingRoll

This API is used by the host to stop a roll call.

```
void stopCallingRoll(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	The result.

replyCallingRoll

This API is used by a participant to reply to a roll call.

```
void replyCallingRoll(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	The result.

sendSpeechInvitation

This API is used by the host to invite a participant to speak.

```
void sendSpeechInvitation(String userId, TUIRoomCoreCallback.InvitationCallback cal
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
callback	TUIRoomCoreCallback.InvitationCallback	The result.

cancelSpeechInvitation

This API is used by the host to cancel a speech invitation.

```
void cancelSpeechInvitation(String userId, TUIRoomCoreCallback.ActionCallback call
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
callback	TUIRoomCoreCallback.ActionCallback	The result.

replySpeechInvitation

This API is used by a participant to accept/reject the host’s invitation to speak.

```
void replySpeechInvitation(boolean agree, TUIRoomCoreCallback.ActionCallback callba
```

The parameters are described below:

Parameter	Type	Description
agree	bool	Whether to accept.

callback	TUIRoomCoreCallback.ActionCallback	The result.

sendSpeechApplication

This API is used by a participant to send a request to speak.

```
void sendSpeechApplication(TUIRoomCoreCallback.InvitationCallback callback);
```

The parameters are described below:

Parameter	Type	Description

callback	TUIRoomCoreCallback.InvitationCallback	The result.

cancelSpeechApplication

This API is used by a participant to cancel the request to speak.

```
void cancelSpeechApplication(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description

callback	TUIRoomCoreCallback.ActionCallback	The result.

replySpeechApplication

This API is used by the host to approve/reject a participant’s speech request.

```
void replySpeechApplication(boolean agree, String userId, TUIRoomCoreCallback.Actio
```

The parameters are described below:

Parameter	Type	Description

agree	boolean	Whether to approve.
userId	String	The user ID.
callback	TUIRoomCoreCallback.ActionCallback	Callback of the result.

forbidSpeechApplication

This API is used by the host to disable requests to speak.

```
void forbidSpeechApplication(boolean forbid, TUIRoomCoreCallback.ActionCallback ca
```

The parameters are described below:

Parameter	Type	Description
forbid	boolean	Whether to disable.
callback	TUIRoomCoreCallback.ActionCallback	The result.

sendOffSpeaker

This API is used by the host to stop the speech of a participant.

```
void sendOffSpeaker(String userId, TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
callback	TUIRoomCoreCallback.ActionCallback	The result.

sendOffAllSpeakers

This API is used by the host to stop the speech of all members.

```
void sendOffAllSpeakers(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	The result.

exitSpeechState

This API is used by a participant to exit the speaker mode.

```
void exitSpeechState(TUIRoomCoreCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomCoreCallback.ActionCallback	The result.

Screen Sharing APIs

startScreenCapture

This API is used to start screen sharing.

```
void startScreenCapture(TRTCCloudDef.TRTCVideoEncParam encParams, TRTCCloudDef.TRTC
```

The parameters are described 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.
screenShareParams	TRTCCloudDef.TRTCScreenShareParams	Special screen sharing configuration. We recommend you set this to `floatingView` , which can prevent the application from being closed by the system and help protect user privacy.

Note

For more information, 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 the beauty filter manger:

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

This API is used to set network QoS control parameters.

```
void setVideoQosPreference(TRTCCloudDef.TRTCNetworkQosParam preference);
```

The parameters are described below:

Parameter	Type	Description
preference	TRTCNetworkQosParam	The network QoS policy.

setAudioQuality

This API is used to set audio quality.

```
void setAudioQuality(int quality);
```

The parameters are described below:

Parameter	Type	Description
quality	int	The audio quality. For more information, see TRTC SDK.

setVideoResolution

This API is used to set the resolution.

```
void setVideoResolution(int resolution);
```

The parameters are described below:

Parameter	Type	Description
resolution	int	The video resolution. For more information, see TRTC SDK.

setVideoFps

This API is used to set the frame rate.

```
void setVideoFps(int fps);
```

The parameters are described below:

Parameter	Type	Description
fps	int	The video capturing frame rate.

Note

Recommended value: 15 or 20 fps. Video will stutter severely if the frame rate is lower than 5 fps and slightly if it is lower than 10 fps. Setting the frame rate to higher than 20 fps would be a waste of resources (the frame rate of films is 24 fps).

setVideoBitrate

This API is used to set the bitrate.

```
void setVideoBitrate(int bitrate);
```

The parameters are described below:

Parameter	Type	Description
bitrate	int	The bitrate. The SDK encodes streams at the target video bitrate. However, it may reduce the bitrate if network conditions are poor. For more information, see TRTC SDK.

Note

Recommended value: See the recommended bitrate for each `TRTCVideoResolution` value. For a better viewing experience, you can slightly increase the bitrate. For example, the recommended bitrate for `TRTC_VIDEO_RESOLUTION_1280_720` is 1,200 Kbps. You can set the bitrate to 1,500 Kbps.

enableAudioEvaluation

This API is used to enable the volume reminder.

```
void enableAudioEvaluation(boolean enable);
```

The parameters are described below:

Parameter	Type	Description
enable	boolean	true: Enable; false: Disable.

Note

After the volume reminder is enabled, the volumes measured by the SDK will be returned via the `onUserVolumeUpdate` callback.

setAudioPlayVolume

This API is used to set the playback volume.

```
void setAudioPlayVolume(int volume);
```

The parameters are described below:

Parameter	Type	Description
volume	int	The payback 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 described below:

Parameter	Type	Description
volume	int	The capturing volume. Value range: 0-100. Default value: 100.

startFileDumping

This API is used to start audio recording.

```
void startFileDumping(TRTCCloudDef.TRTCAudioRecordingParams trtcAudioRecordingParam
```

The parameters are described below:

Parameter	Type	Description
trtcAudioRecordingParams	TRTCCloudDef.TRTCAudioRecordingParams	The audio recording parameters. For more information, see TRTC SDK.

Note

After this API is called, the SDK will record all audios of a call, including the local audio, remote audios, and background music, into a single file. This API works regardless of whether you are in the room or not. When `leaveRoom` is called, audio recording will stop automatically.

stopFileDumping

This API is used to stop audio recording.

```
void stopFileDumping();
```

SDK Version APIs

getSdkVersion

This API is used to get the SDK version.

```
int getSdkVersion();
```

Error Event Callbacks

onError

```
void onError(int code, String message);
```

The parameters are described below:

Parameter	Type	Description
code	int	The error code
message	String	The error message.

Basic Event Callbacks

onDestroyRoom

The room was closed.

```
void onDestroyRoom();
```

onUserVoiceVolume

The audio volume of a user.

```
void onUserVoiceVolume(String userId, int volume);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
volume	int	The volume. Value range: 0-100.

onRoomMasterChanged

The host changed.

```
void onRoomMasterChanged(String previousUserId, String currentUserId);
```

The parameters are described below:

Parameter	Type	Description
previousUserId	String	The host’s user ID before the change.
currentUserId	String	The host’s user ID after the change.

Remote User Callbacks

onRemoteUserEnter

A remote user entered the room.

```
void onRemoteUserEnter(String userId);
```

The parameters are described below:

Parameter	Type	Description

userId	String	The user ID.

onRemoteUserLeave

A remote user exited the room.

```
void onRemoteUserLeave(String userId);
```

The parameters are described below:

Parameter	Type	Description

userId	String	The user ID.

onRemoteUserCameraAvailable

A remote user enabled/disabled their camera.

```
void onRemoteUserCameraAvailable(String userId, boolean available);
```

The parameters are described below:

Parameter	Type	Description

userId	String	The user ID.
available	boolean	true: Enabled; false: Disabled.

onRemoteUserScreenVideoAvailable

A member **enabled**/**disabled** video sharing.

```
void onRemoteUserScreenVideoAvailable(String userId, boolean available);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The User ID.
available	boolean	Whether the user enabled/disabled screen sharing.

onRemoteUserAudioAvailable

A remote user enabled/disabled their mic.

```
void onRemoteUserAudioAvailable(String userId, boolean available);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
available	boolean	Whether the user enabled/disabled their mic.

onRemoteUserEnterSpeechState

A remote user started speaking.

```
void onRemoteUserEnterSpeechState(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.

onRemoteUserExitSpeechState

A remote user stopped speaking.

```
void onRemoteUserExitSpeechState(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.

Chat Message Event Callbacks

onReceiveChatMessage

A text chat message was received.

```
void onReceiveChatMessage(String userId, String message);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
message	String	The message content.

onReceiveRoomCustomMsg

A custom message was received.

```
void onReceiveRoomCustomMsg(String userId, String data);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
message	String	The custom message content.

Room Control Message Callbacks

onReceiveSpeechInvitation

The host sent a speech invitation (received by a participant).

```
void onReceiveSpeechInvitation(String userId);
```

The parameters are described below:

Parameter	Type	Description

userId	String	The host's user ID.

onReceiveInvitationCancelled

The host canceled the speech invitation (received by a participant).

```
void onReceiveInvitationCancelled(String userId);
```

The parameters are described below:

Parameter	Type	Description

userId	String	The host's user ID.

onReceiveSpeechApplication

A participant sent a request to speak (received by the host).

```
void onReceiveSpeechApplication(String userId);
```

The parameters are described below:

Parameter	Type	Description

userId	String	The user ID.

onSpeechApplicationCancelled

A participant canceled a speech request.

```
void onSpeechApplicationCancelled(String userId);
```

The parameters are described below:

Parameter	Type	Description

userId	String	The user ID.

onSpeechApplicationForbidden

The host disabled requests to speak.

```
void onSpeechApplicationForbidden(boolean isForbidden);
```

The parameters are described below:

Parameter	Type	Description

isForbidden	boolean	Whether to disable.

onOrderedToExitSpeechState

A participant was asked to stop speaking.

```
void onOrderedToExitSpeechState(String userId);
```

The parameters are described below:

Parameter	Type	Description

userId	String	The host's user ID.

onCallingRollStarted

The host started a roll call (received by participants).

```
void onCallingRollStarted(String userId);
```

onCallingRollStopped

The host stopped a roll call (received by participants).

```
void onCallingRollStopped(String userId);
```

onMemberReplyCallingRoll

A participant replied to the roll call (received by the host).

```
void onMemberReplyCallingRoll(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.

onChatRoomMuted

The host disabled/enabled chat messages.

```
void onChatRoomMuted(boolean muted);
```

The parameters are described below:

Parameter	Type	Description
muted	boolean	Disabled or not.

onMicrophoneMuted

The host disabled mic use.

```
void onMicrophoneMuted(boolean muted);
```

The parameters are described below:

Parameter	Type	Description
muted	boolean	Disabled or not.

onCameraMuted

The host disabled camera use.

```
void onCameraMuted(boolean muted);
```

The parameters are described below:

Parameter	Type	Description
muted	boolean	Disabled or not.

onReceiveKickedOff

The host removed a member from the room.

```
void onReceiveKickedOff(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the host/admin.

Callback of Statistics on Network Quality and Technical Metrics

onStatistics

Callback of technical metric statistics.

```
void onStatistics(TRTCStatistics statistics);
```

The parameters are described below:

Parameter	Type	Description
statis	TRTCStatistics	Statistics.

onNetworkQuality

Network quality.

```
void onNetworkQuality(TRTCCloudDef.TRTCQuality localQuality, List<TRTCCloudDef.TRTC
```

The parameters are described below:

Parameter	Type	Description
localQuality	TRTCCloudDef.TRTCQuality	The upstream network quality.

remoteQuality	List&amp;lt;TRTCCloudDef.TRTCQuality&amp;gt;	The downstream network quality.

Note

For more information, see TRTC SDK.

Screen Sharing Event Callbacks

onScreenCaptureStarted

Screen sharing started.

```
void onScreenCaptureStarted();
```

onScreenCaptureStopped

Screen sharing stopped.

```
void onScreenCaptureStopped(int reason);
```

The parameters are described below:

Parameter	Type	Description
reason	int	The reason. 0: The user stopped screen sharing; 1: Screen sharing was interrupted by another application.

TUIRoom (iOS)

Last updated：2023-10-13 11:33:19

TUIRoom is based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat. Its features include:
A host can create a room, and users can enter the room ID to join the room.
Participants can share their screens with each other.
All room members can send text chat messages and custom messages.

Note

All components of TUIKit use two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will be activated automatically. For the billing details of Chat, see Pricing.

TUIRoom is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, see Integrating TUIRoom (iOS).
The TRTC SDK is used as a low-latency audio/video room component.
The Chat SDK (**iOS edition**) is used to implement chat messages.

TUIRoom API Overview

TUIRoomCore basic APIs

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 host).
destroyRoom	Closes the room (called by host).
enterRoom	Enters a room (called by participant)
leaveRoom	Leaves a room (called by participant).
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 host permissions to another user (called by host).

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 mirror mode for local video preview.
setSpeaker	Sets whether to play sound from the device's speaker or receiver.

Remote user APIs

API	Description
startRemoteView	Subscribes to and plays back the remote video image of a specified room 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
muteUserMicrophone	Enables/Disables the mic of a specified user.
muteAllUsersMicrophone	Enables/Disables the mics of all users and syncs the status to room information.

muteUserCamera	Enables/Disables the camera of a specified user.
muteAllUsersCamera	Enables/Disables the cameras of all users and syncs the status to room information.
muteChatRoom	Disables/Enables chat messages (called by host).
kickOffUser	Removes a specified user from the room (called by host).
startCallingRoll	Starts a roll call (called by host).
stopCallingRoll	Stops a roll call (called by host).
replyCallingRoll	Replies to a roll call (called by participant).
sendSpeechInvitation	Sends a speech invitation to a participant (called by host).
cancelSpeechInvitation	Cancels a speech invitation sent to a participant (called by host).
replySpeechInvitation	Accepts/Rejects the speech invitation of the host (called by participant).
SendSpeechApplication	Sends a speech request (called by participant).
replySpeechApplication	Approves/Rejects the speech request of a participant (called by host).
forbidSpeechApplication	Disables requests to speak (called by host).
sendOffSpeaker	Stops the speech of a participant (called by host).
sendOffAllSpeakers	Stops the speech of all room members (called by host).
ExitSpeechState	Exits the speaker mode (called by participant).

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 QoS control parameters.

SDK version APIs

API	Description
getSDKVersion	Gets the SDK version.

TUIRoomCoreDelegate API Overview

Callbacks for error events

API	Description
onError	Callback for error.

Basic event callbacks

API	Description
OnDestroyRoom	The room was closed.
onUserVoiceVolume	The audio volume of a user.
onRoomMasterChanged	The host changed.

Remote user event callbacks

API	Description
onRemoteUserEnter	A remote user entered the room.
onRemoteUserLeave	A remote user exited the room.
onRemoteUserCameraAvailable	A remote user enabled/disabled their camera.
onRemoteUserScreenVideoAvailable	A remote user started/stopped screen sharing.
onRemoteUserAudioAvailable	A remote user turned on/off their mic.

onRemoteUserEnterSpeechState	A remote user started speaking.
onRemoteUserExitSpeechState	A remote user stopped speaking.

Message event callbacks

API	Description
onReceiveChatMessage	A text chat message was received.

Room control event callbacks

API	Description
onReceiveSpeechInvitation	A participant received a speech invitation from the host.
onReceiveInvitationCancelled	The speech invitation sent to a participant was canceled by the host.
onReceiveSpeechApplication	The host received a speech request from a participant.
onSpeechApplicationCancelled	A participant canceled a speech request.
onSpeechApplicationForbidden	The host disabled requests to speak.
onOrderedToExitSpeechState	A participant was asked to stop speaking.
onCallingRollStarted	The host started a roll call (received by participants)
onCallingRollStopped	The host stopped a roll call (received by participants).
onMemberReplyCallingRoll	A participant replied to the roll call (received by the host).
onChatRoomMuted	The host disabled/enabled chat messages.
onMicrophoneMuted	The host disabled mic use.
onCameraMuted	The host disabled camera use.
onReceiveKickedOff	The host removed a participant from the room (received by the participant).

Callback of 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.

TUIRoomCore Basic APIs

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 callbacks of TUIRoomCore. You can use `TUIRoomCoreDelegate` to get the callbacks.

```
- (void)setDelegate:(id<TUIRoomCoreDelegate>)delegate;
```

The parameters are described below:

Parameter	Type	Description
delegate	TUIRoomCoreDelegate	The event callback class.

createRoom

This API is used to create a room (called by the host).

```
- (void)createRoom:(NSString *)roomId
         speechMode:(TUIRoomSpeechMode)speechMode
         callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
roomId	NSString	The room ID. You need to assign and manage the IDs in a centralized manner.

speechMode	TUIRoomSpeechMode	The speech mode.
callback	TUIRoomActionCallback	The room creation result.

Generally, a host may need to call the following APIs:

1. The **host** calls `createRoom()` to create a room, the result of which is returned via `TUIRoomActionCallback` .
2. The **host** calls `startCameraPreview()` to enable camera capturing and preview.
3. The **host** calls `startLocalAudio()` to enable the local mic.

destroyRoom

This API is used to close a room (called by the host).

```
- (void)destroyRoom:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomActionCallback	The room closing result.

enterRoom

This API is used to enter a room (called by a participant).

```
- (void)enterRoom:(NSString *)roomId
        callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
roomId	NSString	The room ID.
callback	TUIRoomActionCallback	The result.

Generally, a participant joins a room in the following steps:

1. The **participant** calls `enterRoom` (passing in `roomId`) 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` callback and calls `startRemoteView()` to start playback.

leaveRoom

This API is used to leave a room (called by a participant).

```
- (void)leaveRoom:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomActionCallback	The 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<TUIRoomUserInfo *> *)getRoomUsers;
```

getUserInfo

This API is used to get the information of a specified room member.

```
- (void)getUserInfo:(NSString *)userId
          callback:(TUIRoomUserInfoCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.
callback	TUIRoomUserInfoCallback	Room member details.

setSelfProfile

This API is used to set the user profile.

```
- (void)setSelfProfile:(NSString *)userName
        avatarURL:(NSString *)avatarURL
        callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description

userName	NSString	The username.
avatarURL	NSString	The URL of the user profile photo.
callback	TUIRoomActionCallback	Whether the setting succeeded.

transferRoomMaster

This API is used to transfer host permissions to another user.

```
- (void)transferRoomMaster:(NSString *)userId
```

```
                    callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.
callback	TUIRoomActionCallback	The 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 described 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 described below:

Parameter	Type	Description
quality	TRTCAudioQuality	The sound quality.

stopLocalAudio

This API is used to stop mic capturing.

```
- (void)stopLocalAudio;
```

setVideoMirror

This API is used to set the mirror mode for local video preview.

```
- (void)setVideoMirror:(TRTCVideoMirrorType)type;
```

The parameters are described below:

Parameter	Type	Description
type	TRTCVideoMirrorType	The mirror mode.

setSpeaker

This API is used to set whether to play sound from the device’s speaker or receiver.

```
- (void)setSpeaker:(BOOL)isUseSpeaker;
```

The parameters are described 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 described below:

Parameter	Type	Description

userId	NSString	The ID of the user whose video image is to be played back.
view	UIView	The view that loads video images.
streamType	TUIRoomStreamType	The stream type.
callback	TUIRoomActionCallback	The 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 described below:

Parameter	Type	Description
userId	NSString	The ID of the user whose video image is to be stopped.
streamType	TUIRoomStreamType	The stream type.
callback	TUIRoomActionCallback	The result.

switchCamera

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

```
- (void)switchCamera:(BOOL)isFront;
```

The parameters are described below:

Parameter	Type	Description
isFront	BOOL	YES: Front camera; NO: Rear camera.

Message Sending APIs

sendChatMessage

This API is used to broadcast a text chat message in the room.

```
- (void)sendChatMessage:(NSString *)message
               callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
message	NSString	The message content.
callback	TUIRoomActionCallback	The 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 described below:

Parameter	Type	Description
userId	NSString	The user ID.
mute	BOOL	Whether to disable.

callback	TUIRoomActionCallback	The result.

muteAllUsersMicrophone

This API is used to enable/disable the mics of all users.

```
- (void)muteAllUsersMicrophone:(BOOL)mute
                      callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description

mute	BOOL	Whether to disable.
callback	TUIRoomActionCallback	The 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 described below:

Parameter	Type	Description
userId	NSString	The user ID.
mute	BOOL	Whether to disable.
callback	TUIRoomActionCallback	The result.

muteAllUsersCamera

This API is used to enable/disable the cameras of all users.

```
- (void)muteAllUsersCamera:(BOOL)mute
                  callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
mute	BOOL	Whether to disable.
callback	TUIRoomActionCallback	The result.

muteChatRoom

This API is used to disable/enable chat messages.

```
- (void)muteChatRoom:(BOOL)mute
            callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
mute	BOOL	Whether to disable.

callback	TUIRoomActionCallback	The result.

kickOffUser

This API is used by the host to remove a member from the room.

```
- (void)kickOffUser:(NSString *)userId
           callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description

userId	NSString	The user ID.
callback	TUIRoomActionCallback	The result.

startCallingRoll

This API is used by the host to start a roll call.

```
- (void)startCallingRoll:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomActionCallback	The result.

stopCallingRoll

This API is used by the host to stop a roll call.

```
- (void)stopCallingRoll:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomActionCallback	The result.

replyCallingRoll

This API is used by a participant to reply to a roll call.

```
- (void)replyCallingRoll:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomActionCallback	The result.

sendSpeechInvitation

This API is used by the host to invite a participant to speak.

```
- (void)sendSpeechInvitation:(NSString *)userId
                    callback:(TUIRoomInviteeCallback)callback
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.
callback	TUIRoomInviteeCallback	The result.

cancelSpeechInvitation

This API is used by the host to cancel a speech invitation.

```
- (void)cancelSpeechInvitation:(NSString *)userId
                      callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.
callback	TUIRoomActionCallback	The result.

replySpeechInvitation

This API is used by a participant to accept/reject the host's invitation to speak.

```
- (void)replySpeechInvitation:(BOOL)agree
                     callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
agree	bool	Whether to approve.
callback	TUIRoomActionCallback	The result.

sendSpeechApplication

This API is used by a participant to send a request to speak.

```
- (void)sendSpeechApplication:(TUIRoomInviteeCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomInviteeCallback	The result.

cancelSpeechApplication

This API is used by a participant to cancel the request to speak.

```
- (void)cancelSpeechApplication:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomActionCallback	The result.

replySpeechApplication

This API is used by the host to approve/reject a participant's speech request.

```
- (void)replySpeechApplication:(BOOL)agree
                        userId:(NSString *)userId
                      callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description

agree	BOOL	Whether to approve.
userId	NSString	The user ID.
callback	TUIRoomActionCallback	The result.

forbidSpeechApplication

This API is used by the host to disable requests to speak.

```
- (void)forbidSpeechApplication:(BOOL)forbid
```

```
                    callback:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
forbid	BOOL	Whether to disable.
callback	TUIRoomActionCallback	The result.

sendOffSpeaker

This API is used by the host to stop the speech of a participant.

```
- (void)sendOffSpeaker:(NSString *)userId
             callback:(TUIRoomInviteeCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.
callback	TUIRoomInviteeCallback	The result.

sendOffAllSpeakers

This API is used by the host to stop the speech of all room members.

```
- (void)sendOffAllSpeakers:(TUIRoomInviteeCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomInviteeCallback	The result.

exitSpeechState

This API is used by a participant to exit the speaker mode.

```
- (void)exitSpeechState:(TUIRoomActionCallback)callback;
```

The parameters are described below:

Parameter	Type	Description
callback	TUIRoomActionCallback	The 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 described below:

Parameter	Type	Description

encParams	TRTCVideoEncParam	Sets encoding parameters for screen sharing.

explain

For more information, 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 the beauty filter manger:

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

This API is used to set network QoS control parameters.

```
- (void)setVideoQosPreference:(TRTCNetworkQosParam *)preference;
```

The parameters are described below:

Parameter	Type	Description
preference	TRTCNetworkQosParam	The network QoS policy.

setAudioQuality

This API is used to set audio quality.

```
- (void)setAudioQuality:(TRTCAudioQuality)quality;
```

The parameters are described below:

Parameter	Type	Description
quality	TRTCAudioQuality	The audio quality. For more information, see TRTC SDK.

setVideoResolution

This API is used to set the resolution.

```
- (void)setVideoResolution:(TRTCVideoResolution)resolution;
```

The parameters are described below:

Parameter	Type	Description
resolution	TRTCVideoResolution	The resolution. For details, see TRTC SDK.

setVideoFps

This API is used to set the frame rate.

```
- (void)setVideoFps:(int)fps;
```

The parameters are described below:

Parameter	Type	Description
fps	int	The video capturing frame rate.

explain

Recommended value: 15 or 20 fps. Video will stutter severely if the frame rate is lower than 5 fps and slightly if it is lower than 10 fps. Setting the frame rate to higher than 20 fps would be a waste of resources (the frame rate of films is 24 fps).

setVideoBitrate

This API is used to set the bitrate.

```
- (void)setVideoBitrate:(int)bitrate;
```

The parameters are described below:

Parameter	Type	Description
bitrate	int	The bitrate. The SDK encodes streams at the target video bitrate. However, it may reduce the bitrate if network conditions are poor. For more information, see TRTC SDK.

explain

Recommended value: See the recommended bitrate for each `TRTCVideoResolution` value. For a better viewing experience, you can slightly increase the bitrate. For example, the recommended bitrate for `TRTC_VIDEO_RESOLUTION_1280_720` is 1,200 Kbps. You can set the bitrate to 1,500 Kbps.

enableAudioEvaluation

This API is used to enable the volume reminder.

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

The parameters are described below:

Parameter	Type	Description
bEnable	BOOL	YES: Enable. NO: Disable.

explain

After the volume reminder is enabled, the volumes measured by the SDK will be returned via the `onUserVolumeUpdate` callback.

setAudioPlayVolume

This API is used to set the playback volume.

```
- (void)setAudioPlayVolume:(NSInteger)volume;
```

The parameters are described below:

Parameter	Type	Description

volume	int	The 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 described below:

Parameter	Type	Description

volume	int	The capturing volume. Value range: 0-100. Default value: 100.

startFileDumping

This API is used to start audio recording.

```
- (void)startFileDumping:(TRTCAudioRecordingParams *)params;
```

The parameters are described below:

Parameter	Type	Description

params	TRTCAudioRecordingParams	The recording parameters. For details, see TRTC SDK.

explain

After this API is called, the SDK will record all audios of a call, including the local audio, remote audios, and background music, into a single file. This API works regardless of whether you are in the room or not. When `leaveRoom` is called, audio recording will stop automatically.

stopFileDumping

This API is used to stop audio recording.

```
- (void)stopFileDumping;
```

SDK Version APIs

getSdkVersion

This API is used to get the SDK version.

```
- (NSInteger)getSdkVersion;
```

Error Event Callbacks

onError

```
- (void)onError:(NSInteger)code message:(NSString *)message;
```

The parameters are described below:

Parameter	Type	Description

code	NSInteger	The error code.
message	NSString	The error message.

Basic Event Callbacks

onDestroyRoom

The room was closed.

```
- (void)onDestroyRoom;
```

onUserVoiceVolume

The audio volume of a user.

```
- (void)onUserVoiceVolume:(NSString *)userId volume:(NSInteger)volume;
```

The parameters are described below:

Parameter	Type	Description

userId	NSString	The user ID.
volume	NSInteger	The volume. Value range: 0-100.

onRoomMasterChanged

The host changed.

```
- (void)onRoomMasterChanged:(NSString *)previousUserId
              currentUserId:(NSString *)currentUserId;
```

The parameters are described below:

Parameter	Type	Description
previousUserId	NSString	The host’s user ID before the change.
currentUserId	NSString	The host's user ID after the change.

Remote User Callbacks

onRemoteUserEnter

A remote user entered the room.

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

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.

onRemoteUserLeave

A remote user exited the room.

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

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.

onRemoteUserCameraAvailable

A remote user enabled/disabled their camera.

```
- (void)onRemoteUserCameraAvailable:(NSString *)userId
                          available:(BOOL)available;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The 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 described below:

Parameter	Type	Description
userId	NSString	The user ID.

available	BOOL	Whether the user enabled/disabled screen sharing.

onRemoteUserAudioAvailable

A remote user enabled/disabled their mic.

```
- (void)onRemoteUserAudioAvailable:(NSString *)userId
                         available:(BOOL)available;
```

The parameters are described below:

Parameter	Type	Description

userId	NSString	User ID.
available	BOOL	Whether the user enabled/disabled their mic.

onRemoteUserEnterSpeechState

A remote user started speaking.

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

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.

onRemoteUserExitSpeechState

A remote user stopped speaking.

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

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.

Chat Message Event Callbacks

onReceiveChatMessage

A text chat message was received.

```
- (void)onReceiveChatMessage:(NSString *)userId message:(NSString *)message;
```

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.
message	NSString	The message content.

Room Control Message Callbacks

onReceiveSpeechInvitation

The host sent a speech invitation (received by a participant).

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

The parameters are described below:

Parameter	Type	Description

userId	NSString	The host's user ID.

onReceiveInvitationCancelled

The host canceled the speech invitation (received by a participant).

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

The parameters are described below:

Parameter	Type	Description

userId	NSString	The host's user ID.

OnReceiveSpeechApplication

A participant sent a request to speak (received by the host).

```
void onReceiveSpeechApplication(String userId);
```

The parameters are described below:

Parameter	Type	Description

userId	NSString	User ID.

onSpeechApplicationCancelled

A participant canceled a speech request.

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

The parameters are described below:

Parameter	Type	Description

userId	NSString	The user ID.

onSpeechApplicationForbidden

The host disabled requests to speak.

```
- (void)onSpeechApplicationForbidden:(BOOL)isForbidden userId:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description

isForbidden	BOOL	Disabled or not.
userId	NSString	User ID.

onOrderedToExitSpeechState

A participant was asked to stop speaking.

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

The parameters are described below:

Parameter	Type	Description
userId	NSString	The host's user ID.

onCallingRollStarted

The host started a roll call (received by participants).

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

The parameters are described below:

Parameter	Type	Description
userId	NSString	The host's user ID.

onCallingRollStopped

The host stopped a roll call (received by participants).

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

The parameters are described below:

Parameter	Type	Description
userId	NSString	The host's user ID.

onMemberReplyCallingRoll

A participant replied to the roll call (received by the host).

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

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID.

onChatRoomMuted

The host disabled/enabled chat messages.

```
- (void)onChatRoomMuted:(BOOL)muted userId:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
muted	BOOL	Disabled or not.
userId	NSString	The host's user ID.

onMicrophoneMuted

The host disabled mic use.

```
- (void)onMicrophoneMuted:(BOOL)muted userId:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
muted	BOOL	Disabled or not.
userId	NSString	The host's user ID.

onCameraMuted

The host disabled camera use.

```
- (void)onCameraMuted:(BOOL)muted userId:(NSString *)userId;
```

The parameters are described below:

Parameter	Type	Description
muted	BOOL	Disabled or not.
userId	NSString	The host's user ID.

onReceiveKickedOff

The host removed a member from the room.

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

The parameters are described below:

Parameter	Type	Description
userId	NSString	The user ID of the host/admin.

Callbacks of statistics on network quality and technical metrics

onStatistics

Callback of technical metric statistics.

```
- (void)onStatistics:(TRTCStatistics *)statistics;
```

The parameters are described below:

Parameter	Type	Description
statis	TRTCStatistics	Statistics.

onNetworkQuality

Network quality.

```
- (void)onNetworkQuality:(TRTCQualityInfo *)localQuality remoteQuality:(NSArray<TRT
```

The parameters are described below:

Parameter	Type	Description
localQuality	TRTCQualityInfo	The upstream network quality.

remoteQuality	NSArray<TRTCQualityInfo *>	The downstream network quality.

explain

For more information, see TRTC SDK.

Screen Sharing Event Callbacks

onScreenCaptureStarted

Screen sharing started.

```
- (void)onScreenCaptureStarted;
```

onScreenCaptureStopped

Screen sharing stopped.

```
- (void)onScreenCaptureStopped:(NSInteger)reason;
```

The parameters are described below:

Parameter	Type	Description
reason	NSInteger	The reason. 0: The user stopped screen sharing; 1: Screen sharing was interrupted by another application.

TRTCMeeting API (Flutter)

Last updated：2023-10-13 11:33:57

`TRTCMeeting` has the following features based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat:

The host can create a meeting room, and 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.

Note

The TUIKit series of components are based on two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, the Chat SDK Trial Edition will be activated by default, which will support up to 100 DAUs. For ICha billing details, see Pricing.

`TRTCMeeting` is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, see Group Audio/Video Room (Flutter).

The TRTC SDK is used as the low-latency video conferencing component.

The `MeetingRoom` feature of the Chat 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 an event listener.
unRegisterListener	Terminates an event listener.
login	Logs in.
logout	Logs out.
setSelfProfile	Sets the profile.

Meeting room APIs

API	Description
createMeeting	Creates meeting room (called by host).
destroyMeeting	Closes the meeting room (called by host).
enterMeeting	Enters meeting room (called by participant).
leaveMeeting	Leaves meeting room (called by participant).
getUserInfoList	Gets the list of all users in the room. It takes effect only if it is called after `enterMeeting()` .
getUserInfo	Gets the details of a specified user in the room. It takes effect only if it is called after `enterMeeting()` .

Remote user APIs

API	Description
startRemoteView	Plays back the video of a specified remote member.
stopRemoteView	Stops playing back the video of a specified remote member.
setRemoteViewParam	Sets the rendering parameters of a remote video.
muteRemoteAudio	Mutes/Unmutes a specified remote member's audio.
muteAllRemoteAudio	Mutes/Unmutes all remote users.
muteRemoteVideoStream	Pauses/Resumes a specified remote member's video stream.
muteAllRemoteVideoStream	Pauses/Resumes receiving all 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 mirror mode for local preview.

Local audio APIs

API	Description
startMicrophone	Starts mic capturing.
stopMicrophone	Stops mic capturing.
muteLocalAudio	Mutes/Unmutes local audio.
setSpeaker	Sets whether to play sound from the device’s speaker or receiver.
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
sendRoomTextMsg	Broadcasts a text chat message in a meeting, which is generally used for chat.

sendRoomCustomMsg	Sends a custom text chat message.

TRTCLiveRoomDelegate API Overview

Common event callbacks

API	Description
onError	Callback for error.
onWarning	Callback for warning.
onKickedOffline	You were kicked offline because another user logged in with the same account.

Meeting room event callbacks

API	Description
onRoomDestroy	The meeting room was closed.
onNetworkQuality	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 their mic on/off.

onUserVideoAvailable	A member turned their camera on/off.
onUserSubStreamAvailable	A member enabled/disabled the substream image.

Message event callback APIs

API	Description
onRecvRoomTextMsg	A text chat 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<TRTCMeeting> 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

Login

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

The parameters are described below:

Parameter	Type	Description
sdkAppId	int	You can view the `SDKAppID` via Application Management > **Application Info** in the TRTC console.
userId	String	The 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 how to calculate and use it, see FAQs > UserSig.

logout

Log out

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

setSelfProfile

This API is used to set the profile.

```
Future<ActionCallback> setSelfProfile(String userName, String avatarURL);
```

The parameters are described 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 host).

```
Future<ActionCallback> createMeeting(int roomId);
```

The parameters are described below:

Parameter	Type	Description

roomId	int	The meeting room ID. You need to assign and manage the IDs in a centralized manner.

Generally, the host calls the APIs in the following steps:

1. The **host** calls `createMeeting()` and passes in `roomId` to create a meeting, the result of which is returned via `ActionCallback` .
2. The **host** calls `startCameraPreview()` to enable camera preview. At this time, the beauty filter parameters can be adjusted.
3. The **host** calls `startMicrophone()` to enable mic capturing.

destroyMeeting

This API is used to close a meeting room (called by the host). After creating a meeting, the host can call this API to terminate it.

```
Future<ActionCallback> destroyMeeting(int roomId);
```

The parameters are described below:

Parameter	Type	Description
roomId	int	The 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 a member).

```
Future<ActionCallback> enterMeeting(int roomId);
```

The parameters are described below:

Parameter	Type	Description
roomId	int	The meeting room ID.

Generally, a 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<ActionCallback> leaveMeeting();
```

getUserInfoList

This API is used to get the list of all users in the room. It takes effect only if it is called after `enterMeeting()` .

```
Future<UserListCallback> getUserInfoList(List<String> userIdList);
```

The parameters are described below:

Parameter	Type	Description
userIdList	List<String>	List of `userId` values to obtain.

getUserInfo

This API is used to get the details of a specified user in the room. It takes effect only if it is called after `enterMeeting()` .

```
Future<UserListCallback> getUserInfo(String userId);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.

Remote User APIs

startRemoteView

This API is used to play back the video of a specified remote member.

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

The parameters are described below:

Parameter	Type	Description

userId	String	The user ID.
streamType	int	The type of the video stream to watch. For more information, see TRTC SDK.
viewId	int	`viewId` generated by `TRTCCloudVideoView`

stopRemoteView

This API is used to stop playing back the video of a specified remote member.

```
Future<void> stopRemoteView(String userId, int streamType);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
streamType	int	The type of the video stream to watch. For more information, see TRTC SDK.

setRemoteViewParam

This API is used to set the rendering parameters of a specified remote member’s video.

```
Future<void> setRemoteViewParam(String userId, int streamType,
  {int fillMode, int rotation, int mirrorType});
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
streamType	int	The type of the video stream to watch. For more information, see TRTC SDK.
fillMode	int	The video image rendering mode. Valid values: Fill (default) or Fit. For more information, see TRTC SDK.
rotation	int	Clockwise video image rotation angle. For more information, see TRTC SDK.
type	int	The mirroring mode of the video. For more information, see TRTC SDK.

muteRemoteAudio

This API is used to mute/unmute a specified remote member.

```
Future<void> muteRemoteAudio(String userId, bool mute);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
mute	boolean	`true` : Mute; `false` : Unmute

muteAllRemoteAudio

This API is used to mute/unmute all remote members.

```
Future<void> muteAllRemoteAudio(bool mute);
```

The parameters are described below:

Parameter	Type	Description
mute	boolean	`true` : Mute; `false` : Unmute

muteRemoteVideoStream

This API is used to pause/resume a specified remote member's video stream.

```
Future<void> muteRemoteVideoStream(String userId, bool mute);
```

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
mute	boolean	true: Pause; false: Resume

muteAllRemoteVideoStream

This API is used to pause/resume all remote video streams.

```
Future<void> muteAllRemoteVideoStream(bool mute);
```

The parameters are described 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 described 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 described 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 described below:

Parameter	Type	Description
videoFps	int	Video capturing frame rate.
videoBitrate	int	The video 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	The resolution mode.

Note

For more information, see TRTC SDK.

setLocalViewMirror

This API is used to set the mirror mode for local preview.

```
Future<void> setLocalViewMirror(bool isMirror);
```

The parameters are described 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 described below:

Parameter	Type	Description
quality	int	The audio quality mode. For more information, 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 described below:

Parameter	Type	Description
mute	boolean	`true` : Mute; `false` : Unmute

setSpeaker

This API is used to set whether to play sound from the device’s speaker or receiver.

```
Future<void> setSpeaker(bool useSpeaker);
```

The parameters are described 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 described below:

Parameter	Type	Description
volume	int	The capture volume. Value range: 0–100. Default value: 100.

setAudioPlayoutVolume

This API is used to set the playback volume.

```
Future<void> setAudioPlayoutVolume(int volume);
```

The parameters are described below:

Parameter	Type	Description
volume	int	The playback volume. Value range: 0–100. Default value: 100.

startAudioRecording

This API is used to start audio recording.

```
Future<int?> startAudioRecording(String filePath);
```

The parameters are described below:

Parameter	Type	Description
filePath	String	The storage path of the audio recording file. Specify the path according to your specific needs 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 described below:

Parameter	Type	Description
intervalMs	int	Sets 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 described below:

Parameter	Type	Description
videoFps	int	The frame rate for video capturing.
videoBitrate	int	The video 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	The video resolution.
videoResolutionMode	int	The 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, 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 chat message in a meeting, which is generally used for chat.

```
Future<ActionCallback> sendRoomTextMsg(String message);
```

The parameters are described below:

Parameter	Type	Description
message	String	A text chat message.

sendRoomCustomMsg

This API is used to send a custom text message.

```
Future<ActionCallback> sendRoomCustomMsg(String cmd, String message);
```

The parameters are described below:

Parameter	Type	Description
cmd	String	A custom command word used to distinguish between different message types.
message	String	A text chat message.

`TRTCMeetingDelegate` 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.
The parameters are described below:

Parameter	Type	Description
errCode	int	The error code.
errMsg	String	The error message.
extraInfo	String	Extended field. Certain error codes may carry extra information for troubleshooting.

onWarning

Callback for warning.
The parameters are described below:

Parameter	Type	Description
warningCode	int	The warning code.
warningMsg	String	The 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 Callback APIs

onRoomDestroy

Callback for meeting room termination. When the host leaves the room, all users in the room will receive this callback.

The parameters are described below:

Parameter	Type	Description
roomId	String	The meeting room ID.

onNetworkQuality

Callback for network status.

The parameters are described below:

Parameter	Type	Description
localQuality	TRTCCloudDef.TRTCQuality	The upstream network quality.
remoteQuality	List<TRTCCloudDef.TRTCQuality>	The downstream network quality.

onUserVolumeUpdate

Call volume of a user.

The parameters are described below:

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

Member Entry/Exit Event Callbacks

onEnterRoom

You joined the meeting.

The parameters are described below:

Parameter	Type	Description
result	int	A value greater than 0 indicates the time (in ms) at which the meeting was joined. A value smaller than 0 indicates the error code thrown while joining the meeting.

onLeaveRoom

You left the meeting.

The parameters are described below:

Parameter	Type	Description
reason	int	The 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 closed.

onUserEnterRoom

A new member joined the meeting.

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the new member who enters the room.

onUserLeaveRoom

A member left the meeting.

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID of the member who leaves the room.

Member Audio/Video Event Callbacks

onUserAudioAvailable

A member turned their mic on/off.

The parameters are described below:

Parameter	Type	Description
userId	String	The user ID.
available	boolean	true: The mic is enabled; false: The mic is disabled.

onUserVideoAvailable

A member turned their camera on/off.

The parameters are described below:

Parameter	Type	Description
userId	String	The 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 described below:

Parameter	Type	Description
userId	String	The user ID.
available	boolean	true: The substream is enabled; false: The substream is disabled.

Message Event Callback APIs

onRecvRoomTextMsg

A text chat message was received.

The parameters are described below:

Parameter	Type	Description
message	String	A text chat message.
sendId	String	The message sender's user ID.
userAvatar	String	The sender's profile photo.
userName	String	The sender's username.

onRecvRoomCustomMsg

A custom message was received.

The parameters are described below:

Parameter	Type	Description
command	String	A custom command word used to distinguish between different message types.
message	String	A text chat message.

sendId	String	The sender's user ID.
userAvatar	String	The sender's profile photo.
userName	String	The sender's username.

Screen Sharing Event Callbacks

onScreenCaptureStarted

Screen sharing started.

onScreenCapturePaused

Screen sharing paused.

onScreenCaptureResumed

Screen sharing resumed.

onScreenCaptureStoped

Screen sharing stopped.

The parameters are described below:

Parameter	Type	Description
reason	int	The reason screen sharing stopped. 0: The user actively stopped; 1: Screen sharing stopped as the shared window was closed.

TUIRoom (Windows and macOS)

Last updated：2023-10-13 11:34:37

TUIRoom is based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat. With TUIRoom:
The host can create a room, and participants can enter the room ID to join the room.
The participants can share their screens with each other.
All room members can send text chat messages and custom messages.

Note

All components of TUIKit use two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports up to 100 DAUs) will be activated automatically. For the billing details of Chat, see Pricing.

TUIRoom is an open-source class depending on two closed-source Tencent Cloud SDKs. For the specific implementation process, see Integrating TUIRoom (Windows and macOS).
The TRTC SDK is used as a low-latency video conference component.
The Chat SDK (**C++ edition**) is used to implement chat messages.

TUIRoom API Overview

TUIRoomCore basic APIs

API	Description
GetInstance	Gets a singleton object.
DestroyInstance	Terminates a singleton object.
SetCallback	Sets event callbacks.

Room APIs

API	Description
login	Logs in.
logout	Logs out.
CreateRoom	Creates a room (called by host).
DestroyRoom	Closes the room (called by host).
EnterRoom	Enters a room (called by participant).

LeaveRoom	Leaves a room (called by participant).
GetRoomInfo	Gets the room information.
GetRoomUsers	Gets the information of all members in the room.
GetUserInfo	Gets the information of a user.
TransferRoomMaster	Transfers the host permissions (called by host).

Local audio/video operation APIs

API	Description
StartCameraPreview	Enables 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 mirror mode for local video preview.

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
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 mics of all users and syncs the status to room information.
MuteUserCamera	Enables/Disables the camera of a specified user.
MuteAllUsersCamera	Enables/Disables the cameras of all users and syncs the status to room information.
MuteChatRoom	Enables/Disables chat messages (called by host).
KickOffUser	Removes a specified user from the room (called by host).
StartCallingRoll	Starts a roll call (called by host).
StopCallingRoll	Stops a roll call (called by host).
ReplyCallingRoll	Replies to a roll call (called by participant).
SendSpeechInvitation	Sends a speech invitation to a participant (called by host).
CancelSpeechInvitation	Cancels a speech invitation sent to a participant (called by host).
ReplySpeechInvitation	Accepts/Rejects the speech invitation of the host (called by participant).
SendSpeechApplication	Sends a speech request (called by participant).
CancelSpeechApplication	Cancels a speech request (called by participant).
ReplySpeechApplication	Approves/Rejects the speech request of a participant (called by host).
ForbidSpeechApplication	Disables requests to speak (called by host).
SendOffSpeaker	Stops the speech of a participant (called by host).
SendOffAllSpeakers	Stops the speech of all members (called by host).
ExitSpeechState	Exits the speaker mode (called by participant).

Basic component APIs

API	Description

GetDeviceManager	Gets the local device management object `ITXDeviceManager` .
GetScreenShareManager	Gets the screen sharing management object `IScreenShareManager` .

On-cloud recording APIs

API	Description
StartCloudRecord	Starts on-cloud recording.
StopCloudRecord	Stops on-cloud recording.

Beauty filter APIs

API	Description
SetBeautyStyle	Sets beauty filters.

Settings APIs

API	Description
SetVideoQosPreference	Sets network QoS control parameters.

SDK version 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	The local user logged in.
OnLogout	The local user logged out.
OnCreateRoom	The room was created.
OnDestroyRoom	The room was closed.
OnEnterRoom	The local user entered the room.
OnExitRoom	The local user left the room.
OnFirstVideoFrame	Started rendering the first video frame.
onUserVoiceVolume	The audio volume of a user.
OnRoomMasterChanged	The host changed.

Remote user event callbacks

API	Description
OnRemoteUserEnter	A remote user entered the room.
OnRemoteUserLeave	A remote user exited the room.
OnRemoteUserCameraAvailable	A remote user enabled/disabled their camera.
OnRemoteUserScreenAvailable	A remote user started/stopped screen sharing.
OnRemoteUserAudioAvailable	A remote user turned on/off their mic.
OnRemoteUserEnterSpeechState	A remote user started speaking.
OnRemoteUserExitSpeechState	A remote user stopped speaking.

Message event callbacks

API	Description
OnReceiveChatMessage	A text chat message was received.
OnReceiveCustomMessage	A custom message was received.

Room control event callbacks

API	Description

OnReceiveSpeechInvitation	A participant received a speech invitation from the host.
OnReceiveInvitationCancelled	The speech invitation sent to a participant was canceled by the host.
OnReceiveReplyToSpeechInvitation	A participant accepted the speech invitation sent by the host.
OnReceiveSpeechApplication	The host received a speech request from a participant.
OnSpeechApplicationCancelled	A participant canceled a speech request.
OnReceiveReplyToSpeechApplication	The host approved a request to speak.
OnSpeechApplicationForbidden	The host disabled requests to speak.
OnOrderedToExitSpeechState	A participant was asked to stop speaking.
OnCallingRollStarted	The host started a roll call (received by participants)
OnCallingRollStopped	The host stopped a roll call (received by participants).
OnMemberReplyCallingRoll	A participant replied to the roll call (received by the host).
OnChatRoomMuted	The host disabled/enabled chat messages.
OnMicrophoneMuted	The host disabled mic use.
OnCameraMuted	The host disabled camera use.

Callbacks 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.

Video recording callbacks

API	Description

OnRecordError	Recording error.
OnRecordComplete	Recording completed.
OnRecordProgress	The recording progress.

Local device test callbacks

API	Description
OnTestSpeakerVolume	The speaker volume.
OnTestMicrophoneVolume	The mic volume.
OnAudioDeviceCaptureVolumeChanged	The system capturing volume changed.
OnAudioDevicePlayoutVolumeChanged	The system audio playback volume changed.

TUIRoomCore Basic APIs

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 callbacks of TUIRoomCore. You can use `TRTCChorusRoomDelegate` to get the callbacks.

```
virtual void SetCallback(const TUIRoomCoreCallback* callback) = 0;
```

Login

Login

```
virtual int Login(int sdk_appid, const std::string& user_id, const std::string& use
```

The parameters are described below:

Parameter	Type	Description
sdk_appid	int	You can view `SDKAppID` in Application Management > **Application Info** of the TRTC console.
user_id	string	The 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 own

		account system.
user_sig	string	Tencent Cloud's proprietary security signature. For how to calculate and use it, see FAQs > UserSig.

Logout

Log out

```
virtual int Logout() = 0;
```

CreateRoom

This API is used to create a room (called by the host).

```
virtual int CreateRoom(const std::string& room_id, TUISpeechMode speech_mode) = 0;
```

The parameters are described below:

Parameter	Type	Description
room_id	string	The room ID. You need to assign and manage the IDs in a centralized manner.

speech_mode	TUISpeechMode	The speech mode.

Generally, a host may need to call the following APIs:

1. The **host** calls `CreateRoom()` to create a room, the result of which is returned via `OnCreateRoom` .
2. The **host** calls `EnterRoom()` to enter the room.
3. The **host** calls `StartCameraPreview()` to enable camera capturing and preview.
4. The **host** calls `StartLocalAudio()` to enable the local mic.

DestroyRoom

This API is used to close a room (called by the host).

```
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 described below:

Parameter	Type	Description
room_id	string	The room ID.

Generally, a participant joins a meeting in the following steps:

1. The **participant** calls `EnterRoom` (passing 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` callback and calls `StartRemoteView()` to start playback.

LeaveRoom

This API is used to leave 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 members in the room.

```
virtual std::vector<TUIUserInfo> GetRoomUsers() = 0;
```

GetUserInfo

This API is used to get the information of a specified member.

```
virtual const TUIUserInfo* GetUserInfo(const std::string& user_id) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.

SetSelfProfile

This API is used to set the user profile.

```
virtual int SetSelfProfile(const std::string& user_name, const std::string& avatar_
```

The parameters are described below:

Parameter	Type	Description
user_name	string	The username.
avatar_url	string	The URL of the user profile photo.

TransferRoomMaster

This API is used to transfer host permissions to another user.

```
virtual int TransferRoomMaster(const std::string& user_id) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The 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 described below:

Parameter	Type	Description

view	liteav::TXView	The 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 local preview window.

```
virtual int UpdateCameraPreview(const liteav::TXView& view) = 0;
```

The parameters are described below:

Parameter	Type	Description
view	liteav::TXView	The window handle.

StartLocalAudio

This API is used to enable the local audio device.

```
virtual int StartLocalAudio(const liteav::TRTCAudioQuality& quality) = 0;
```

The parameters are described below:

Parameter	Type	Description
view	liteav::TXView	The 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 the mirror mode.

```
virtual int SetVideoMirror(bool mirror) = 0;
```

The parameters are described below:

Parameter	Type	Description
mirror	bool	Whether to mirror the video.

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 described below:

Parameter	Type	Description
user_id	string	The ID of the user whose video image is to be played back.

liteav::TXView	TXView	The view that loads the video.
type	TUIStreamType	The 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 described below:

Parameter	Type	Description
user_id	string	The ID of the user whose video image is to be stopped.
type	TUIStreamType	The stream type.

UpdateRemoteView

This API is used to update the rendering window of a remote video.

```
virtual int UpdateRemoteView(const std::string& user_id, TUIStreamType type, liteav
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
type	TUIStreamType	The stream type.
view	liteav::TXView	The rendering window handle.

Message Sending APIs

SendChatMessage

This API is used to send a text message.

```
virtual int SendChatMessage(const std::string& message) = 0;
```

The parameters are described below:

Parameter	Type	Description
message	string	The message content.

SendCustomMessage

This API is used to send a custom message.

```
virtual int SendCustomMessage(const std::string& message) = 0;
```

The parameters are described below:

Parameter	Type	Description
message	string	The 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 call
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
mute	bool	Whether to disable.

callback	Callback	API callback.

MuteAllUsersMicrophone

This API is used to enable/disable the mics of all users.

```
virtual int MuteAllUsersMicrophone(bool mute) = 0;
```

The parameters are described 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
```

The parameters are described below:

Parameter	Type	Description

user_id	string	The user ID.
mute	bool	Whether to disable.
callback	Callback	API callback.

MuteAllUsersCamera

This API is used to enable/disable the cameras of all users.

```
virtual int MuteAllUsersCamera(bool mute) = 0;
```

The parameters are described below:

Parameter	Type	Description
mute	bool	Whether to disable.

MuteChatRoom

This API is used to disable/enable chat messages.

```
virtual int MuteChatRoom(bool mute) = 0;
```

The parameters are described below:

Parameter	Type	Description
mute	bool	Whether to disable.

KickOffUser

This API is used by the host to remove a member from the room.

```
virtual int KickOffUser(const std::string& user_id, Callback callback) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
callback	Callback	API callback.

StartCallingRoll

This API is used by the host to start a roll call.

```
virtual int StartCallingRoll() = 0;
```

StopCallingRoll

This API is used by the host to stop a roll call.

```
virtual int StopCallingRoll() = 0;
```

ReplyCallingRoll

This API is used by a participant to reply to a roll call.

```
virtual int ReplyCallingRoll(Callback callback) = 0;
```

The parameters are described below:

Parameter	Type	Description
callback	Callback	API callback.

SendSpeechInvitation

This API is used by the host to invite a participant to speak.

```
virtual int SendSpeechInvitation(const std::string& user_id, Callback callback) = 0
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
callback	Callback	API callback.

CancelSpeechInvitation

This API is used by the host to cancel a speech invitation.

```
virtual int CancelSpeechInvitation(const std::string& user_id, Callback callback) =
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
callback	Callback	API callback.

ReplySpeechInvitation

This API is used by a participant to accept/reject the host’s invitation to speak.

```
virtual int ReplySpeechInvitation(bool agree, Callback callback) = 0;
```

The parameters are described below:

Parameter	Type	Description
agree	bool	Whether to approve.

callback	Callback	API callback.

SendSpeechApplication

This API is used by a participant to send a request to speak.

```
virtual int SendSpeechApplication(Callback callback) = 0;
```

The parameters are described below:

Parameter	Type	Description

callback	Callback	API callback.

CancelSpeechApplication

This API is used by a participant to cancel the request to speak.

```
virtual int CancelSpeechApplication(Callback callback) = 0;
```

The parameters are described below:

Parameter	Type	Description

callback	Callback	API callback.

ReplySpeechApplication

This API is used by the host to approve/reject a participant’s speech request.

```
virtual int ReplySpeechApplication(const std::string& user_id, bool agree, Callback
```

The parameters are described below:

Parameter	Type	Description

user_id	string	The user ID.
callback	Callback	API callback.

ForbidSpeechApplication

This API is used by the host to disable requests to speak.

```
virtual int ForbidSpeechApplication(bool forbid) = 0;
```

The parameters are described below:

Parameter	Type	Description
forbid	bool	Whether to disable.

SendOffSpeaker

This API is used by the host to stop the speech of a participant.

```
virtual int SendOffSpeaker(const std::string& user_id, Callback callback) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
callback	Callback	API callback.

SendOffAllSpeakers

This API is used by the host to stop the speech of all members.

```
virtual int SendOffAllSpeakers(Callback callback) = 0;
```

The parameters are described below:

Parameter	Type	Description
callback	Callback	API callback.

ExitSpeechState

This API is used by a participant to exit the speaker mode.

```
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 the strength of the beauty, skin brightening, and rosy skin effects

```
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 the beauty filter manger:

Set the beauty style to smooth or natural. The smooth style features more obvious skin smoothing effect.

Set the strength of the beauty effect. Value range: 0-9. 0 indicates to disable the effect. The larger the value, the more obvious the effect.

Set the skin brightening strength. Value range: 0-9. 0 indicates to disable the effect. The larger the value, the more obvious the effect.

The parameters are described below:

Parameter	Type	Description
style	liteav::TRTCBeautyStyle	The beauty style.
beauty_level	uint32_t	The strength of the beauty effect.
whiteness_level	uint32_t	The strength of the skin brightening effect.
ruddiness_level	uint32_t	The strength of the rosy skin effect.

Settings APIs

SetVideoQosPreference

This API is used to set network QoS control parameters.

```
virtual int SetVideoQosPreference(TUIVideoQosPreference preference) = 0;
```

The parameters are described below:

Parameter	Type	Description
preference	TUIVideoQosPreference	The network QoS policy.

SDK Version APIs

GetSDKVersion

This API is used to get the SDK version.

```
virtual const char* GetSDKVersion() = 0;
```

Error Event Callbacks

OnError

```
void OnError(int code, const std::string& message);
```

The parameters are described below:

Parameter	Type	Description
code	int	The error code.
message	string	The error message.

Basic Event Callbacks

OnLogin

```
virtual void OnLogin(int code, const std::string& message) = 0;
```

The parameters are described below:

Parameter	Type	Description
code	int	The error code.

message	string	The login information or error message for login failure.

OnLogout

```
virtual void OnLogout(int code, const std::string& message) = 0;
```

The parameters are described below:

Parameter	Type	Description

code	int	The error code.
message	string	The error message.

OnCreateRoom

A room was created.

```
virtual void OnCreateRoom(int code, const std::string& message) = 0;
```

The parameters are described below:

Parameter	Type	Description
code	int	The error code.
message	string	The error message.

OnDestroyRoom

The room was closed.

```
virtual void OnDestroyRoom(int code, const std::string& message) = 0;
```

The parameters are described below:

Parameter	Type	Description
code	int	The error code.
message	string	The error message.

OnEnterRoom

The local user entered the room.

```
virtual void OnEnterRoom(int code, const std::string& message) = 0;
```

The parameters are described below:

Parameter	Type	Description
code	int	The error code.
message	string	The error message.

OnExitRoom

The local user left the room.

```
virtual void OnExitRoom(TUIExitRoomType type, const std::string& message) = 0;
```

The parameters are described below:

Parameter	Type	Description
type	TUIExitRoomType	The room exit type.
message	string	The error message.

OnFirstVideoFrame

The first video frame of the local user or a remote user was rendered.

```
virtual void OnFirstVideoFrame(const std::string& user_id, const TUIStreamType stre
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
stream_type	TUIStreamType	The stream type.

OnUserVoiceVolume

The audio volume of a user.

```
virtual void OnUserVoiceVolume(const std::string& user_id, int volume)
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
volume	int	The volume level. Value range: 0-100.

OnRoomMasterChanged

The host changed.

```
virtual void OnRoomMasterChanged(const std::string& user_id) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.

Remote User Callbacks

OnRemoteUserEnter

A remote user entered the room.

```
virtual void OnRemoteUserEnter(const std::string& user_id) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.

OnRemoteUserLeave

A remote user exited the room.

```
virtual void OnRemoteUserLeave(const std::string& user_id) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.

OnRemoteUserCameraAvailable

A remote user enabled/disabled their camera.

```
virtual void OnRemoteUserCameraAvailable(const std::string& user_id, bool available
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
available	bool	true: Enabled; false: Disabled.

OnRemoteUserScreenAvailable

A remote user started/stopped screen sharing.

```
virtual void OnRemoteUserScreenAvailable(const std::string& user_id, bool available
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
available	bool	true: Enabled; false: Disabled.

OnRemoteUserAudioAvailable

A remote user enabled/disabled their mic.

```
virtual void OnRemoteUserAudioAvailable(const std::string& user_id, bool available)
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The 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 described below:

Parameter	Type	Description
user_id	string	The user ID.

OnRemoteUserExitSpeechState

A remote user stopped speaking.

```
virtual void OnRemoteUserExitSpeechState(const std::string& user_id) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.

Chat Message Event Callbacks

OnReceiveChatMessage

A text chat message was received.

```
virtual void OnReceiveChatMessage(const std::string& user_id, const std::string& me
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
message	string	The message content.

OnReceiveCustomMessage

A custom message was received.

```
virtual void OnReceiveCustomMessage(const std::string& user_id, const std::string&
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.

message	string	The custom message content.

Room Control Message Callbacks

OnReceiveSpeechInvitation

The host sent a speech invitation (received by a participant).

```
virtual void OnReceiveSpeechInvitation() = 0;
```

OnReceiveInvitationCancelled

The host canceled the speech invitation (received by a participant).

```
virtual void OnReceiveInvitationCancelled() = 0;
```

OnReceiveReplyToSpeechInvitation

A participant accepted/rejected a speech invitation (received by the host).

```
virtual void OnReceiveReplyToSpeechInvitation(const std::string& user_id, bool agre
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.
agree	bool	Whether the invitation was accepted.

OnReceiveSpeechApplication

A participant sent a request to speak (received by the host).

```
virtual void OnReceiveSpeechApplication(const std::string& user_id) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.

OnSpeechApplicationCancelled

A participant canceled a speech request.

```
virtual void OnSpeechApplicationCancelled(const std::string& user_id) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.

OnReceiveReplyToSpeechApplication

The host approved/rejected a request to speak.

```
virtual void OnReceiveReplyToSpeechApplication(bool agree) = 0;
```

The parameters are described below:

Parameter	Type	Description
agree	bool	Whether the request was approved.

OnSpeechApplicationForbidden

The host disabled/enabled requests to speak.

```
virtual void OnSpeechApplicationForbidden(bool forbidden) = 0;
```

The parameters are described below:

Parameter	Type	Description
forbidden	bool	Whether requests to speak were disabled.

OnOrderedToExitSpeechState

A participant was asked to stop speaking.

```
virtual void OnOrderedToExitSpeechState() = 0;
```

OnCallingRollStarted

The host started a roll call (received by participants).

```
virtual void OnCallingRollStarted() = 0;
```

OnCallingRollStopped

The host stopped a roll call (received by participants).

```
virtual void OnCallingRollStopped() = 0;
```

OnMemberReplyCallingRoll

A participant replied to the roll call (received by the host).

```
virtual void OnMemberReplyCallingRoll(const std::string& user_id) = 0;
```

The parameters are described below:

Parameter	Type	Description
user_id	string	The user ID.

OnChatRoomMuted

The host disabled/enabled chat messages.

```
virtual void OnChatRoomMuted(bool muted) = 0;
```

The parameters are described below:

Parameter	Type	Description
muted	bool	Disabled or not.

OnMicrophoneMuted

The host disabled mic use.

```
virtual void OnMicrophoneMuted(bool muted) = 0;
```

The parameters are described below:

Parameter	Type	Description
muted	bool	Disabled or not.

OnCameraMuted

The host disabled camera use.

```
virtual void OnCameraMuted(bool muted) = 0;
```

The parameters are described below:

Parameter	Type	Description
muted	bool	Disabled or not.

Statistics Collection and Quality Callbacks

OnStatistics

Callback of technical metric statistics.

```
virtual void OnStatistics(const liteav::TRTCStatistics& statis) {}
```

The parameters are described below:

Parameter	Type	Description
statis	liteav::TRTCStatistics	Statistics.

OnNetworkQuality

Network quality.

```
virtual void OnNetworkQuality(const liteav::TRTCQualityInfo& local_quality, liteav:
        uint32_t remote_quality_count) {}
```

The parameters are described below:

Parameter	Type	Description
local_quality	liteav::TRTCQualityInfo	The network quality of the local user.

remote_quality	liteav::TRTCQualityInfo*	The network quality of remote users.
remote_quality_count	uint32_t	The 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 described below:

Parameter	Type	Description

reason	int	The reason screen sharing stopped. 0: The user stopped screen sharing; 1: Screen sharing was interrupted by another application.

Video Recording Callbacks

OnRecordError

Recording error.

```
virtual void OnRecordError(TXLiteAVLocalRecordError error, const std::string& messg
```

The parameters are described below:

Parameter	Type	Description
error	TXLiteAVLocalRecordError	The error.
messgae	string	The error message.

OnRecordComplete

Recording completed.

```
virtual void OnRecordComplete(const std::string& path) {}
```

The parameters are described below:

Parameter	Type	Description
path	string	The error description.

OnRecordProgress

The recording progress.

```
virtual void OnRecordProgress(int duration, int file_size) {}
```

The parameters are described below:

Parameter	Type	Description
duration	int	The file duration.
file_size	int	The file size.

Local Device Test Callbacks

OnTestSpeakerVolume

The speaker volume.

```
virtual void OnTestSpeakerVolume(uint32_t volume) {}
```

The parameters are described below:

Parameter	Type	Description

volume	uint32_t	The volume level.

OnTestMicrophoneVolume

The mic volume.

```
virtual void OnTestMicrophoneVolume(uint32_t volume) {}
```

The parameters are described below:

Parameter	Type	Description

volume	uint32_t	The volume level.

OnAudioDeviceCaptureVolumeChanged

The system capturing volume changed.

```
virtual void OnAudioDeviceCaptureVolumeChanged(uint32_t volume, bool muted) {}
```

The parameters are described below:

Parameter	Type	Description

volume	uint32_t	The volume level.
muted	bool	Whether capturing was disabled.

OnAudioDevicePlayoutVolumeChanged

The system playback volume level changed.

```
virtual void OnAudioDevicePlayoutVolumeChanged(uint32_t volume, bool muted) {}
```

The parameters are described below:

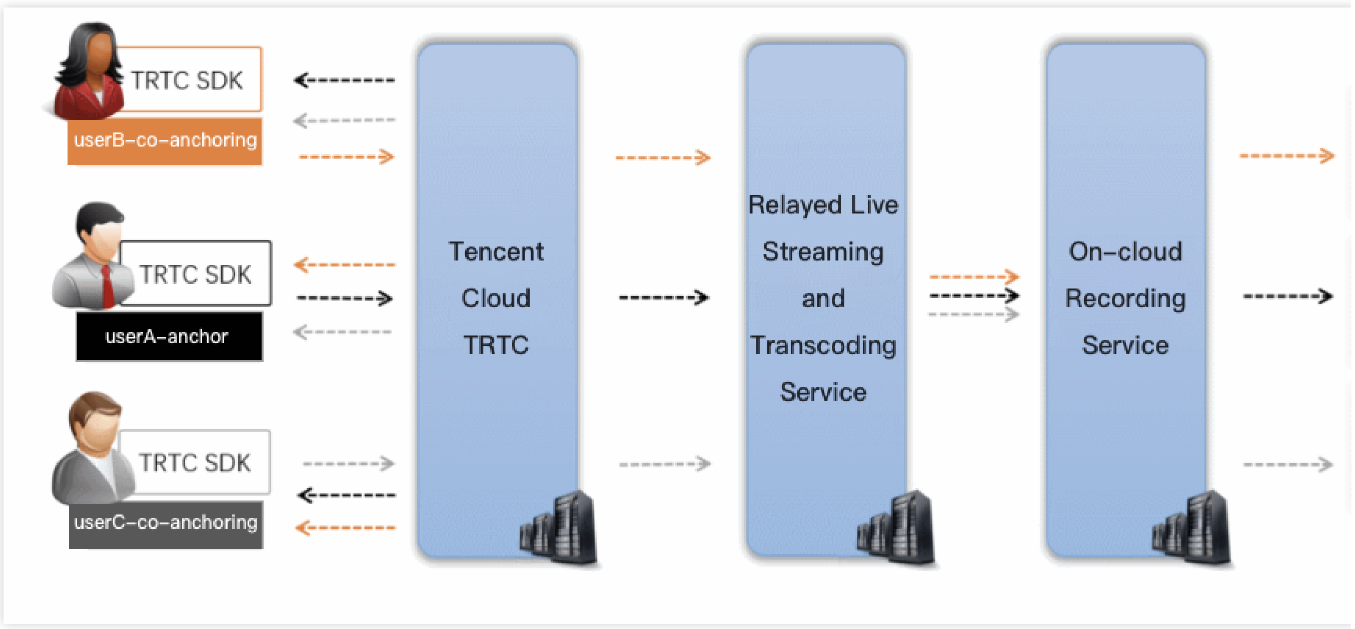
Parameter	Type	Description
volume	uint32_t	The volume level.
muted	bool	Whether playback was disabled.

On-Cloud Recording and Playback (Old)

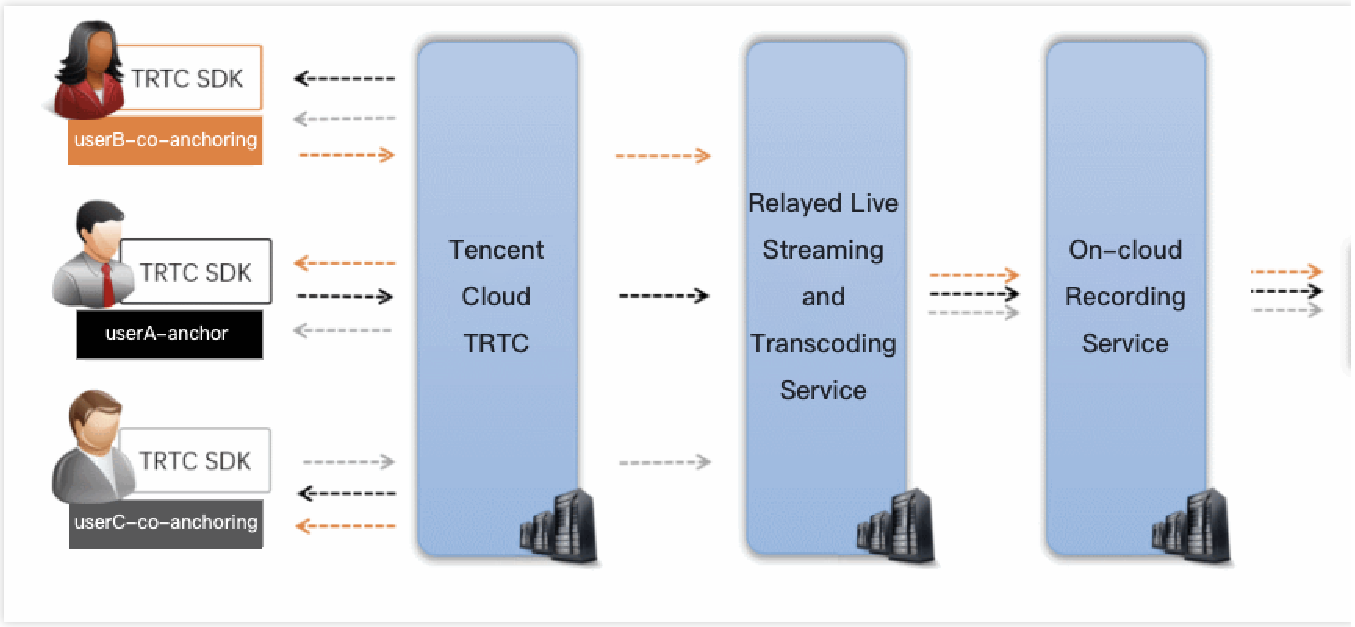
Last updated：2023-10-13 11:35:19

Use Cases

In application scenarios such as remote education, showroom streaming, video conferencing, remote loss assessment, financial transaction recording, and online healthcare, it is often necessary to record an entire video call or live streaming session and save the recording files for purposes such as evidence gathering, quality control, auditing, archiving, and playback.
With TRTC's on-cloud recording feature, you can record the audio/video streams of each user in a room into separate files.



You can also mix the audio/video streams in a room into one stream using On-Cloud MixTranscoding and then record the mixed stream into a single file.



Notes：

For users with an SDKAppID starting with 140, please refer to this cloud recording and playback operation guide for i ntegration and usage. If your application's SDKAppID starts with 200, please refer to On-Cloud Recording (New) to initiate the cloud recording function.

Console Guide

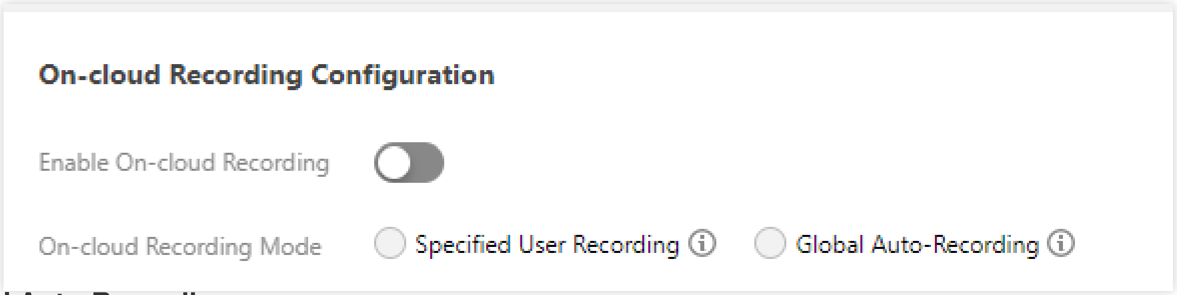
Enabling on-cloud recording

1. Log in to the TRTC console and click Application Management on the left sidebar.
2. Find your application and click **Function Configuration** on the right. If you haven’t created an application yet, click **Create Application**, enter an application name, and click **Confirm** to create one.
3. Click

next to **Enable On-Cloud Recording**. A recording configuration window will pop up.

Selecting the recording mode

TRTC supports recording in two modes: **Global Auto-Recording** and **Specified User Recording**.


On-cloud Recording Configuration

Enable On-cloud Recording

On-cloud Recording Mode ○ Specified User Recording ○ Global Auto-Recording

Global Auto-Recording

The upstream audio/video streams of all users in all TRTC rooms are automatically recorded. The recording starts and stops automatically. Because no human intervention is needed, this mode is simple and easy to use. For detailed instructions, see Scheme 1: Global Auto-Recording.

Specified User Recording

You can specify users whose streams you want to record. This is achieved using either the client-side SDK API or server-side RESTful API for on-cloud recording and requires additional development efforts. For detailed instructions, see Scheme 2: specified user recording (SDK API) and Scheme 3: specified user recording (RESTful API).

Select the file format

TRTC can record streams in four formats, namely HLS, MP4, FLV, and AAC. Their differences and application scenarios are listed in the table below.

Parameter	Description
File type	The following file types are supported: HLS: Files in this format can be played back on most browsers and are ideal for scenarios in which videos need to be replayed multiple times. When this format is selected, recording can resume from breakpoints, and no upper limit is set on the recording length of a file. FLV: Files in this format cannot be played back on browsers, but the format is simple and fault tolerant. You can use this format if you do not need to save recording files in VOD. Just download the files immediately after recording and delete the original files. MP4: Files in this format can be played back on browsers, but the format is not fault tolerant. Any packet loss during a video call may affect the playback quality of the recording file. AAC: Select this format if you want to record audio only.
Maximum file length (minutes)	You can set a maximum length for recording files based on your needs. The system will automatically segment files that exceed the limit. The value range is 1-120 (minutes). When **HLS** is selected for **File Type**, there is no limit on the length of recording files, and this parameter becomes invalid.
File retention duration (day)	You can set for how many days you want VOD to save your recording files. The value range is 0-1500 (days). Files that expire will be deleted and cannot be

	retrieved. `0` indicates the files will be saved indefinitely.
Resumption timeout (second)</td>	By default, if a call/live streaming session is interrupted due to network jitter or other reasons, the call will be recorded into multiple files. You can set this parameter to generate only one playback link for each call/live streaming session. When recording is cut off, if the interruption does not exceed the configured time period, only one file will be generated. You can get the file only after the timeout period elapses. The value range of this parameter is 0-1800 (seconds). 0 indicates that breakpoint resumption is disabled.

Notes

HLS allows a maximum resumption timeout period of 30 minutes, making it possible to generate only one playback link for each lecture. What's more, files in HLS format can be played back on most browsers, making the format ideal for video playback in online education scenarios.

Setting the path to save recording files

In TRTC, recording files are stored in Tencent Cloud's VOD platform by default. If more than one of your businesses share a Tencent Cloud VOD account, you may want to separate their recording files, which you can achieve through VOD's subapplication feature.

What are VOD primary applications and subapplications?

Primary applications and subapplications offer a way to separate your resources in VOD. A primary application is essentially your VOD account. Multiple subapplications can be created under a primary application, each of which functions as a sub-account of the VOD account. The resources of subapplications are managed separately and assigned their own storage space.

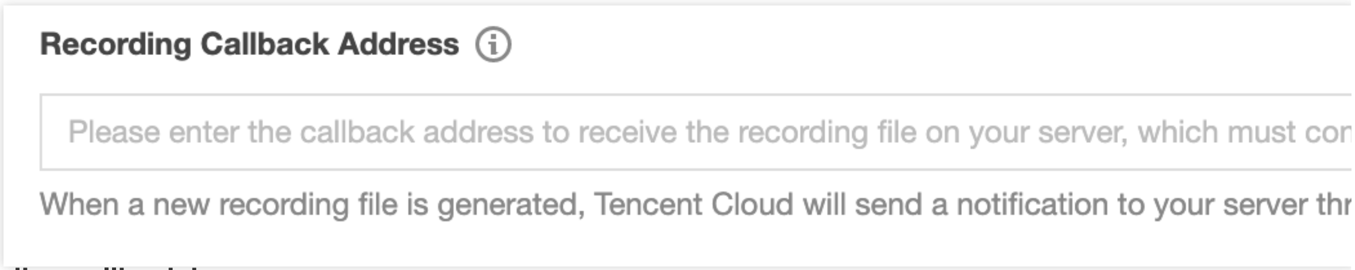
How do I enable the subapplication feature in VOD?

You can create subapplications in the VOD console as instructed in Subapplication System.

Configuring recording callback

Recording callback address:

To receive notifications about the generation of new recording files in real time, set the **Recording Callback Address** to an HTTP or HTTPS address on your server. When a new recording file is generated, Tencent Cloud will send a notification to your server via this address.


Recording Callback Address

Please enter the callback address to receive the recording file on your server, which must con

When a new recording file is generated, Tencent Cloud will send a notification to your server thr

Recording callback key:

The callback key is used to generate authentication signatures for receiving recording callbacks. The key must consist of not more than 32 letters and digits. For details, see Common callback parameters.

Notes

For more details on receiving callbacks of recording files and how to interpret the callbacks received, see Receiving recording files below.

Recording Schemes

TRTC offers three on-cloud recording schemes, namely global auto-recording, specified user recording (SDK API), and specified user recording (RESTful API). We will explain the following for each of the scheme.

How do I select the scheme in the console?

How to start a recording task?

How to end a recording task?

How do I mix multiple streams in a room into one stream?

How is a recording file named?

What platforms does the scheme support?

Scheme 1: Global auto-recording

Selecting the scheme in the console

To use this recording scheme, set On-Cloud Recording Mode to **Global Auto-Recording** in the console.

Starting a recording task

The audio/video streams of each user in a TRTC room are recorded automatically, with no need for human intervention.

Ending a recording task

A task stops automatically when an anchor stops sending audio/video data. However, if you have set the **Resumption Timeout** parameter when choosing the format for recording files, you will not receive the recording file until the timeout period elapses.

Mixing streams

In the global auto-recording mode, you can mix streams by using either the server-side RESTful API or the client-side SDK API. These two methods for mixing streams should not be used at the same time.

Stream mixing via the server-side RESTful API: The stream mixing API must be called on your server. This method works regardless of the platform the SDK runs on.
Stream mixing via the client-side API: The stream mixing API can be called from the client. This method works on iOS, Android, Windows, macOS, and browsers. It does not work for Weixin Mini Programs.

Naming of recording files

If an anchor has set userDefineRecordId during room entry, recording files will be named `userDefineRecordId_streamType__start time_end time` (`streamType` has two valid values: `main` , which indicates the primary stream, and `aux` , which indicates the substream.)

If an anchor has set streamId, but not userDefineRecordId during room entry, recording files will be named `streamId_start time_end time` .

If an anchor has set neither userDefineRecordId nor streamId during room entry, recording files will be named `sdkappid_roomid_userid_streamType_start time_end time` (`streamType` has two valid values: `main` , which indicates the primary stream, and `aux` , which indicates the substream.)

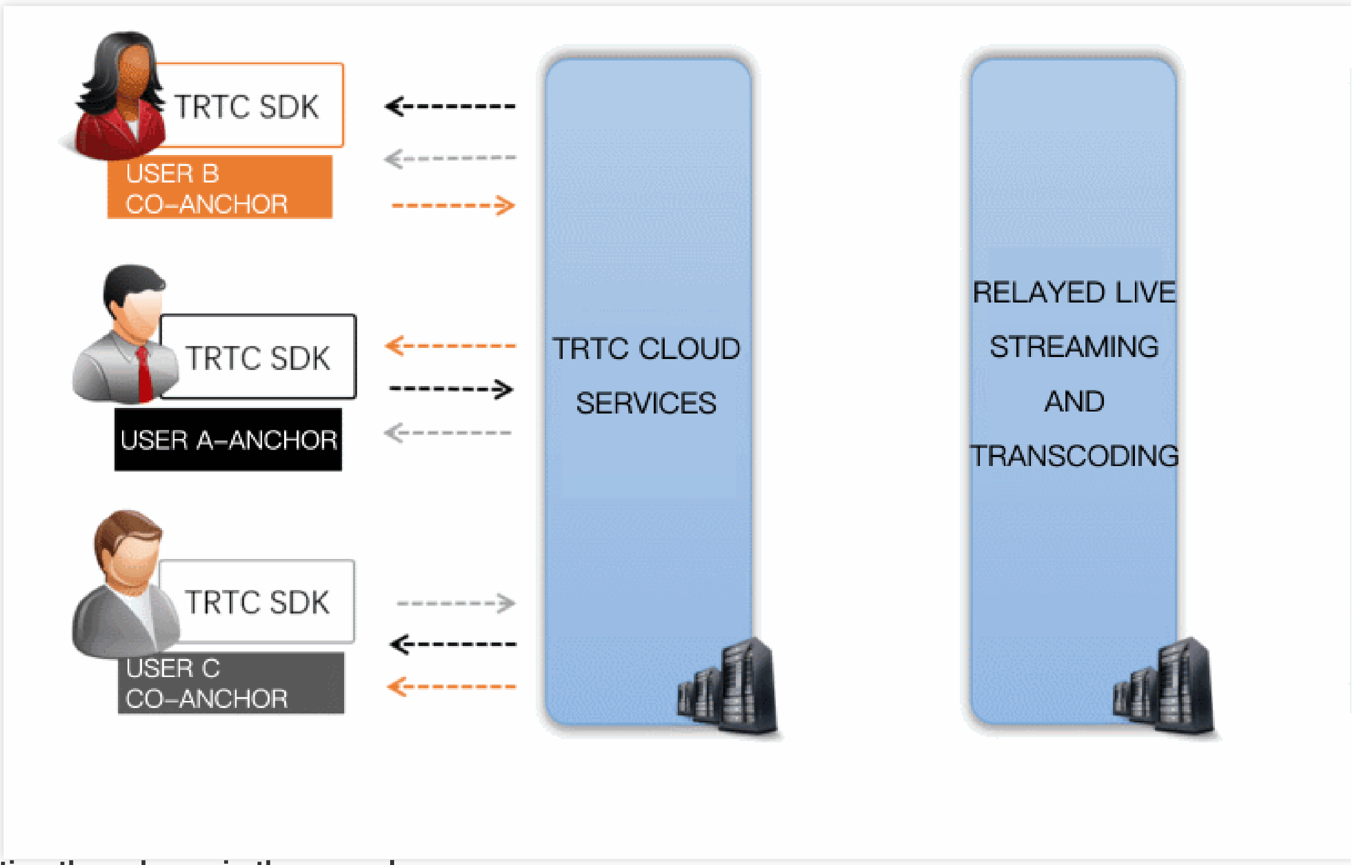
Supported platforms

Recording operations are initiated by your server and are not affected by the platform on which the SDK runs.

Scheme 2: Specified user recording (SDK API)

You can call APIs of the TRTC SDK to enable On-Cloud MixTranscoding, on-cloud recording, and CDN live streaming.

On-Cloud Capability	Enabling	Disabling
On-cloud recording	Specify `userDefineRecordId` in `TRTCParams` during room entry.	Recording stops automatically after the anchor leaves the room.
On-Cloud MixTranscoding	Call the SDK API setMixTranscodingConfig().	Stream mixing stops automatically after the anchor who starts it leaves the room. The anchor can also stop it manually by calling setMixTranscodingConfig() and setting the parameter to `null/nil` .
CDN live streaming	Specify the `streamId` field in `TRTCParams` during room entry.	Relaying stops automatically after the anchor leaves the room.



Selecting the scheme in the console

To use this recording scheme, set On-Cloud Recording Mode to **Specified User Recording** in the console.

Starting a recording task

If an anchor has set userDefineRecordId in the room entry parameter TRTCParams during room entry, the audio/video data sent by the anchor will be recorded on the cloud. The streams of anchors who have not set the parameter will not be recorded.

```
// Sample code: Record the streams of the user rexchang and set the recording ID to
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
TRTCParams *param = [[TRTCParams alloc] init];
param.sdkAppId = 1400000123;     // The SDKAppID, which is generated after you crea
param.roomId   = 1001;           // The room ID
param.userId   = @"rexchang";    // The user ID
param.userSig  = @"xxxxxxxx";    // The login signature
params.role     = TRTCRoleAnchor; // The role: anchor
param.userDefineRecordId = @"1001_rexchang";  // The recording ID. Specify this par
[trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE]; // Please use the `LIVE` mo
```

Ending a recording task

The task stops automatically after the anchor who has set userDefineRecordId during room entry stops sending audio/video. However, if you have set the **Resumption Timeout** parameter when choosing the format for recording files, you will not receive the recording file until the timeout period elapses.

Mixing streams

You can call the SDK API setMixTranscodingConfig() to mix the audio/video streams of other users in a room with the current user's audio/video streams. For details, see On-Cloud MixTranscoding.

Notes

Make sure that the `setMixTranscodingConfig` API is called by just one anchor (preferably the anchor who created the room) in a TRTC room. Calling of the API by multiple anchors may cause errors.

Naming of recording files

Recording files are named `userDefineRecordId_start time_end time` .

Supported platforms

Recording tasks can be initiated on iOS, Android, Windows, macOS, Electron, and web. Weixin Mini Programs are not supported currently.

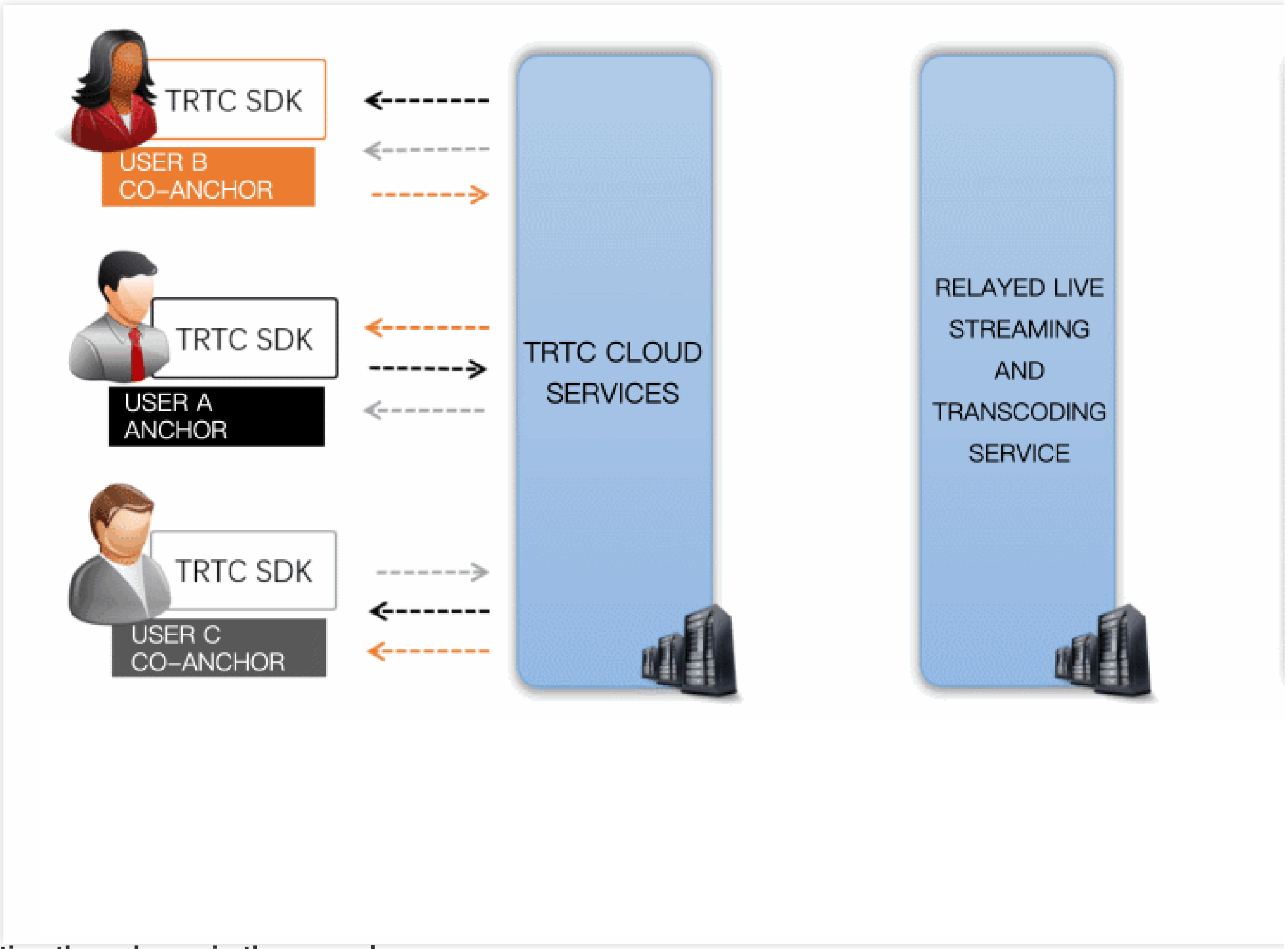
Scheme 3: Specified user recording (RESTful API)

TRTC's server provides a pair of RESTful APIs (StartMCUMixTranscode and StopMCUMixTranscode) for the implementation of On-Cloud MixTranscoding, on-cloud recording, and CDN live streaming.

On-Cloud Capability	Enabling	Disabling
On-cloud recording	Call StartMCUMixTranscode from your server, specifying `OutputParams.RecordId` .	Recording will stop automatically. You can also call StopMCUMixTranscode to manually stop it.
On-Cloud MixTranscoding	Call StartMCUMixTranscode, specifying `LayoutParams` (which determines the video layout).	Stream mixing will stop automatically. You can also call StopMCUMixTranscode to manually stop it.
CDN live streaming	Call StartMCUMixTranscode, specifying `OutputParams.StreamId` .	Relaying will stop automatically. You can also call StopMCUMixTranscode to manually stop it.

Notes

The two RESTful APIs work via TRTC’s core stream mixing MCU and send mixed streams to the recording system and live streaming CDNs, hence the name `Start/StopMCUMixTranscode` . In addition to stream mixing, the APIs can also be used to enable on-cloud recording and CDN live streaming.



Selecting the scheme in the console

To use this recording scheme, set On-Cloud Recording Mode to **Specified User Recording** in the console.

Starting a recording task

Specify `OutputParams.RecordId` when calling StartMCUMixTranscode from your server to enable stream mixing and recording.

```
// Sample code: Call the RESTful API to start an On-Cloud MixTranscoding and on-clo
https://trtc.tencentcloudapi.com/?Action=StartMCUMixTranscode
&SdkAppId=1400000123
&RoomId=1001
&OutputParams.RecordId=1400000123_room1001
&OutputParams.RecordAudioOnly=0
&EncodeParams.VideoWidth=1280
&EncodeParams.VideoHeight=720
&EncodeParams.VideoBitrate=1560
&EncodeParams.VideoFramerate=15
&EncodeParams.VideoGop=3
```

```
&EncodeParams.BackgroundColor=0
&EncodeParams.AudioSampleRate=48000
&EncodeParams.AudioBitrate=64
&EncodeParams.AudioChannels=2
&LayoutParams.Template=1
&<Common request parameters>
```

Notes

The RESTful API works only when at least one user has entered the room (`enterRoom`).

The RESTful API cannot be used to record single streams. If you want to record single streams, choose Scheme 1 or Scheme 2.

Ending a recording task

The task stops automatically. You can also call StopMCUMixTranscode to stop it manually.

Mixing streams

Set `LayoutParams` when calling StartMCUMixTranscode to enable On-Cloud MixTranscoding. The API can be called multiple times during live streaming, meaning that you can modify the `LayoutParams` parameter to change the layout of video images. Make sure you use the same `OutputParams.RecordId` and `OutputParams.StreamId` for each call; otherwise the recording will be interrupted and multiple recording files will be generated.

Notes

For more information about On-Cloud MixTranscoding, see On-Cloud MixTranscoding.

Naming of recording files

Recording files are named `OutputParams.RecordId_start time_ end time` .

`OutputParams.RecordId` is set when StartMCUMixTranscode is called.

Supported platforms

Recording operations are initiated by your server and are not affected by the platform on which the SDK runs.

Searching for Recording Files

When recording is enabled, files recorded in TRTC are saved in Tencent Cloud's VOD platform. You can search for files in the VOD console or use a RESTful API on your server for scheduled filtering.

Searching for files in the VOD console

1. Log in to the VOD console and click **Media Assets** on the left sidebar.
2. Click **Search by prefix** above the list, select **Search by prefix**, enter a keyword such as `1400000123_1001_rexchang_main` in the search box, and click the search icon.
3. You can also filter files by creation time.

Searching for files via a RESTful API

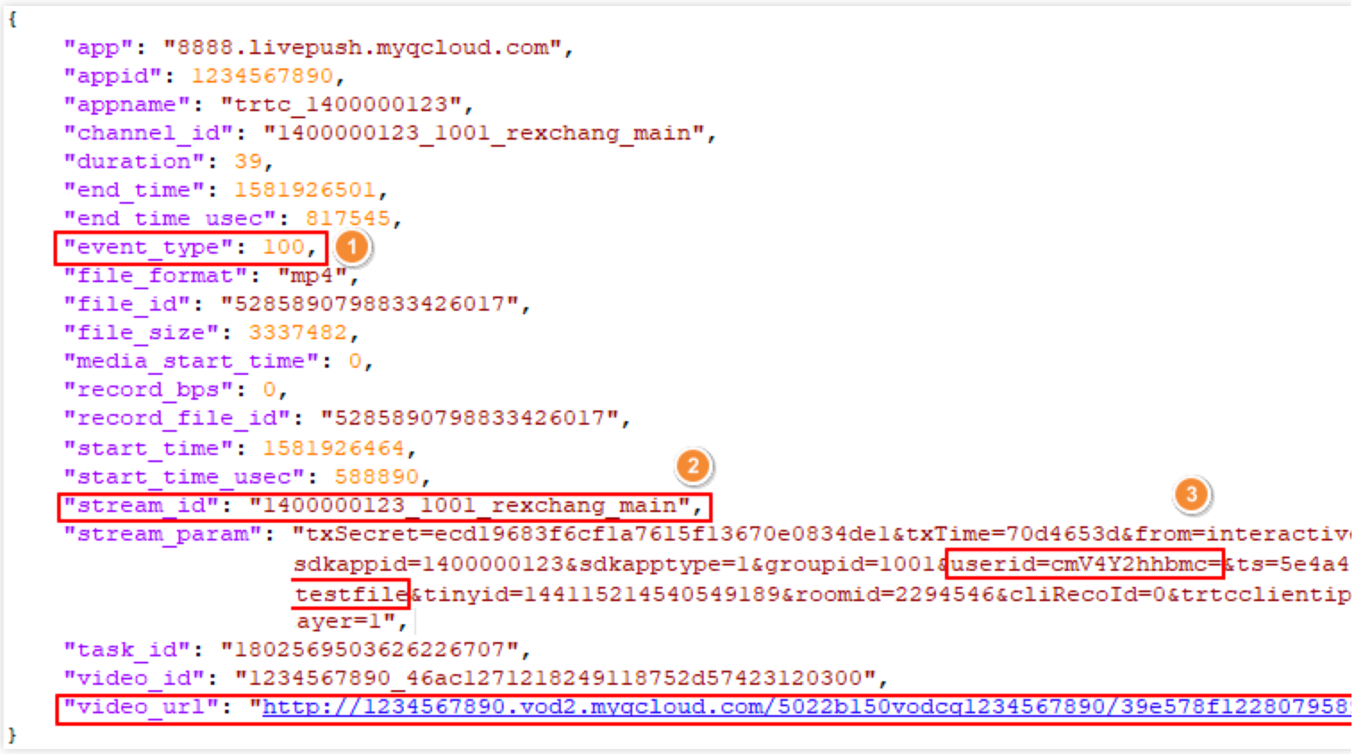
You can use the SearchMedia API to query files in VOD. Specify the request parameter `Text` for fuzzy searches or `StreamId` for exact searches.

RESTful request sample code:

```
https://vod.tencentcloudapi.com/?Action=SearchMedia
&StreamId=stream1001
&Sort.Field=CreateTime
&Sort.Order=Desc
&<Common request parameters>
```

Receive Recording Files

In addition to [#search!a41705a4893cb9904429d608433f5092), you can configure a callback address to have Tencent Cloud push notifications to your server when new recording files are generated.
When the last user exits the room, Tencent Cloud will stop recording and save the recording file in VOD. This normally takes about 30-120 seconds. If you have set a resumption timeout period (for example, 300 seconds), then the process will take 300 seconds longer. After saving the file, Tencent Cloud will send a notification to your server via the specified (HTTP/HTTPS) callback address.
Tencent Cloud sends information about all recording events to your server via the specified callback address. Below is an example of a callback message:



```
{
    "app": "8888.livepush.myqcloud.com",
    "appid": 1234567890,
    "appname": "trtc_1400000123",
    "channel_id": "1400000123_1001_rexchang_main",
    "duration": 39,
    "end_time": 1581926501,
    "end_time_usec": 817545,
    "event_type": 100,
    "file_format": "mp4",
    "file_id": "5285890798833426017",
    "file_size": 3337482,
    "media_start_time": 0,
    "record_bps": 0,
    "record_file_id": "5285890798833426017",
    "start_time": 1581926464,
    "start_time_usec": 588890,
    "stream_id": "1400000123_1001_rexchang_main",
    "stream_param": "txSecret=ecd19683f6cf1a7615f13670e0834de1&txTime=70d4653d&from=interactiv
                  sdkappid=1400000123&sdkapptype=1&groupid=1001&userid=cmV4Y2hhbmc=&ts=5e4a4
                  testfile&tinyid=144115214540549189&roomid=2294546&cliRecoId=0&trtcclientip
                  ayer=1",
    "task_id": "1802569503626226707",
    "video_id": "1234567890_46ac1271218249118752d57423120300",
    "video_url": "http://1234567890.vod2.myqcloud.com/5022b150vodcq1234567890/39e578f122807958
}
```

The fields in the table below help you determine which call/live streaming session a callback is about.

No.	Field	Description
1	event_type	The event type. `100` indicates that a recording file has been generated.
2	stream_id	The stream ID for CDN live streaming, which you can set by specifying the streamId field in `TRTCParams` during room entry (recommended), or when calling the `startPublishing` API of `TRTCCloud` .

③	stream_param.userid	The Base64-encoded user ID.
④	stream_param.userdefinerecordid	<td>A custom field. You can set this field by specifying the userDefineRecordId field in `TRTCParams` .
⑤	video_url	The URL of the recording file, which can be used for replay.

Notes

For information about other fields, see CSS-Recording Event Notification.

Deleting Recording Files

VOD provides a series of RESTful APIs for the management of audio/video files. You can call the DeleteMedia API to delete a file.

RESTful request sample code:

```
https://vod.tencentcloudapi.com/?Action=DeleteMedia
&FileId=52858907988664150587
&<Common request parameters>
```

Playing Back Recording Files

In application scenarios such as online education, live streaming sessions are often replayed to make the best of teaching resources.

Select the file format (HLS)

Select HLS as the file format.
HLS supports a maximum timeout period of 30 minutes for breakpoint resumption, making it possible to generate only one playback link for each live streaming session/lecture. What's more, files in HLS format can be played back on most browsers, making it an ideal format for replay.

Get the playback address (video_url)

After receiving a recording file callback, you can get the playback address of the file by looking for the **video_url** field in the callback message.

Integrate the VOD player

Integrate the VOD player. For detailed directions, see the following documents.
iOS
Android
Web
Notes
We recommend using the All-in-One SDK, which integrates the player component as well as Mobile Live Video Broadcasting (MLVB). This integrated edition adds less to the size of the application package than two independent SDKs do because the Tencent Cloud services share many of their underlying modules. It also allows you to avoid duplicate symbol issues.

Costs

The on-cloud recording and playback feature is powered by TRTC's on-cloud recording capability and VOD's storage, video processing, and playback capabilities. It also relies on the VOD playback capability of the SDKs.

Cloud service fees

Cloud service fees include the fees incurred for recording videos on the cloud and playing back recording files.
Notes
The prices used in the examples in this document are for reference only. In case of any inconsistencies, the prices specified in Billing of On-Cloud Recording, CSS Pricing and VOD Pricing will apply.
Recording fee: The computing cost of transcoding or remuxing
Because server computing resources are needed to transcode or remux audio/video streams during recording, a recording fee is charged based on the computing resources used for recording.
Notes
For Tencent Cloud accounts that created their first TRTC applications on or after July 1, 2020, on-cloud recording fees are charged as described in Billing of On-Cloud Recording.

For Tencent Cloud accounts that created applications before July 1, 2020, **live recording** fees will continue to apply. **Live recording** fees are calculated based on the peak number of concurrent recording channels. The higher the number, the higher the fee. For details, see CSS > Live Recording.

Formula:
Percentage of recording days = Number of days the recording feature is used in a month / Total number of days in that month.
Recording fees = Peak number of concurrent recording channels in a month x Percentage of recording days x Recording channel unit price.

Assume that you had 1,000 anchors in April. During peak times, you recorded the audio/video streams of 500 anchors at the same time, and the recording feature was used on four days that month (percentage of recording days is 6/30). If the service is priced 5.2941 USD per channel per month, then the live recording fee incurred in April would be `500 channels x 5.2941 USD/channel/month = 2,647.05 USD/month`.
If you select two file formats, both the recording and storage fees will double. If you select three, they will triple. To reduce cost, you are advised to select only one file format.

Transcoding fee: Incurred for using On-Cloud MixTranscoding
Because stream mixing involves encoding and decoding, an additional transcoding fee will be incurred if you enable On-Cloud MixTranscoding. The fee varies with the resolution used and the transcoding duration. The higher resolution used for anchors, and the longer co-anchoring (the most common application scenario for On-Cloud MixTranscoding) lasts, the higher the cost. For more information, see Live Transcoding.

Assume that you called setVideoEncoderParam() to set the bitrate (`videoBitrate`) for anchors to 1,500 Kbps and resolution to 720p, and the anchor co-anchored with a viewer for one hour, during which On-Cloud MixTranscoding was enabled. The transcoding fee incurred would be `0.0057 USD/min x 60 min = 0.342 USD`.

Storage fee: Incurred for storing files with VOD
Because storage requires disk resources, if you save recording files in VOD, a storage fee will be charged based on the disk resources used. The longer you save a file, the higher the cost. To reduce cost, you are advised to keep the storage duration as short as possible or store recording files on your own server. Storage fees are charged in the daily pay-as-you-go mode.

Assume that you called setVideoEncoderParam() to set the bitrate (`videoBitrate`) of an anchor to 1000 Kbps and recorded the anchor's stream (one file format) for one hour. The size of the recording file generated would be approximately `(1,000/8) KBps x 3,600s = 450,000 KB = 0.45 GB` . If you store the file with VOD, the storage fee per day would be `0.45 GB x 0.0009 USD/GB/day = 0.000405 USD` .

Playback fee: Incurred for playing back a file**
Playing back a recording file relies on VOD's CDN playback feature and consumes CDN traffic. By default, a video acceleration fee is charged based on the traffic consumed by playback. The larger audience you serve, the higher the cost. Playback fees are charged in the daily pay-as-you-go mode.

Assume that you recorded a 1 GB file on the cloud and played it back to 1,000 users from beginning to end. The traffic consumed amounted to 1 TB, and according to VOD's tiered pricing system, the playback fee incurred would be `1,000 x 1 GB x 0.0794 USD/GB = 79.4 USD` . If you have bought a traffic package, 24.5 USD would be deducted.
If you download files from Tencent Cloud to your server, a relatively small amount of traffic will also be consumed, which is billed on a monthly basis.

SDK playback license

TRTC All-in-One offers powerful player capabilities based on VOD. If you use our mobile SDK and its version is 10.1 or later, you need to buy a license to use the player capabilities.

Notes

You don't need a license for playback within TRTC.

Live Streaming with UI (Old)

Integrating TUILiveRoom (Android)

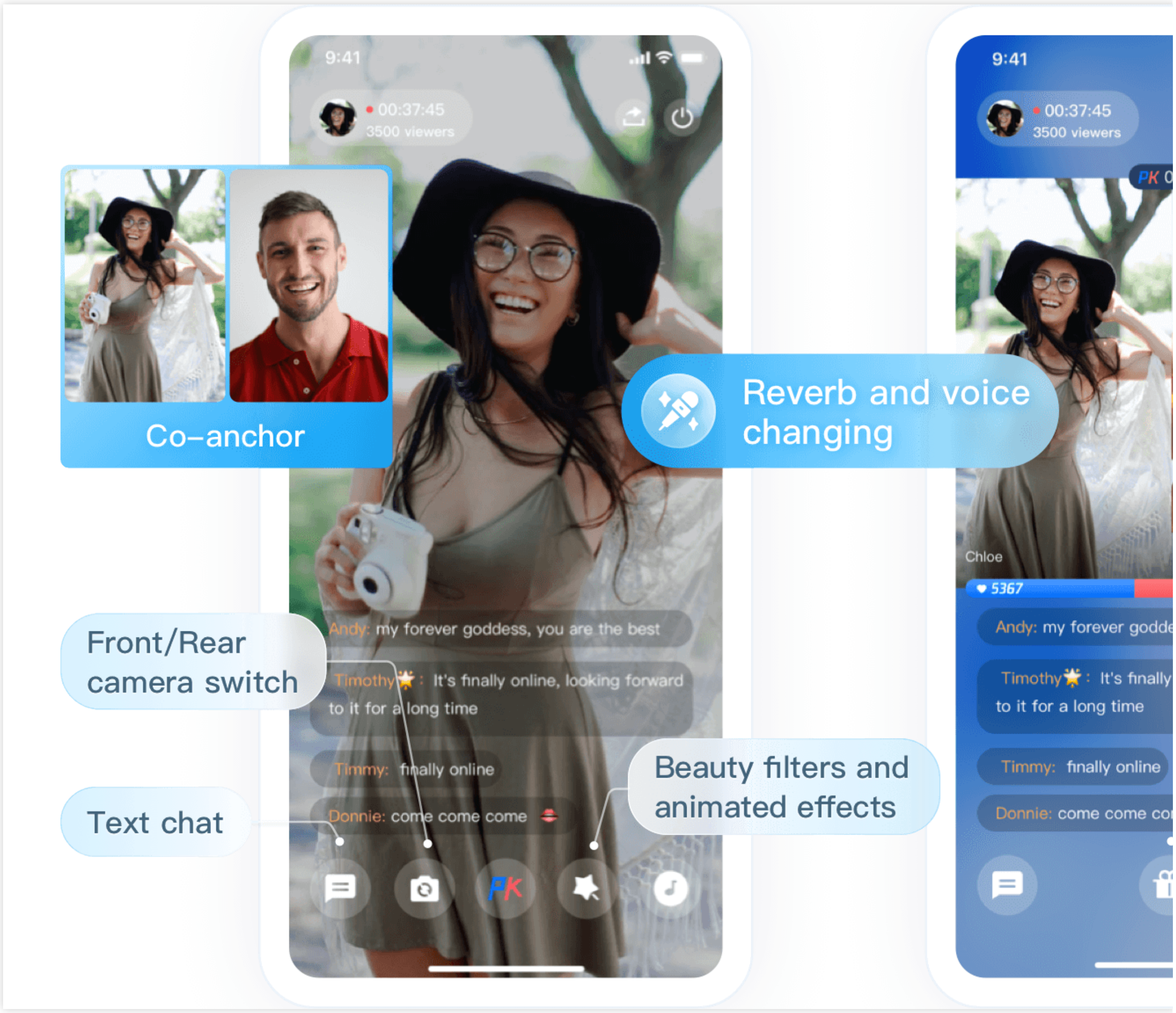
Last updated：2023-09-21 16:22:21

Overview

`TUILiveRoom` is an open-source video live streaming scenario UI component. After integrating it into your project, you can enable your application to support interactive video live streaming simply by writing a few lines of code. It provides source code for Android, iOS, and mini program platforms. Its basic features are as shown below:

Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see Pricing.


Co–anchor
Reverb and voice changing
Front/Rear camera switch
Text chat
Beauty filters and animated effects
9:41
00:37:45
3500 viewers
Andy: my forever goddess, you are the best
Timothy: It's finally online, looking forward to it for a long time
Timmy: finally online
Donnie: come come come
Chloe
5367
PK

Integration

Step 1. Download and import the `TUILiveRoom` component

Go to GitHub, clone or download the code, copy the `Android/debug` , `Android/tuiaudioeffect` , `Android/tuibarrage` , `Android/tuibeauty` , `Android/tuigift` , and `Android/tuiliveroom` directories to your project, and complete the following import operations:

Add the code below in `setting.gradle` :

```
include ':debug'
include ':tuibeauty'
include ':tuibarrage'
include ':tuiaudioeffect'
include ':tuigift'
include ':tuiliveroom'
```

Add dependencies on `tuiliveroom` to the `build.gradle` file in `app` :

```
api project(":tuiliveroom")
```

Add the TRTC SDK (`liteavSdk`) and Chat SDK (`imSdk`) dependencies in `build.gradle` in the root directory:

```
ext {
  liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"
  imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"
}
```

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 later, the camera and storage read 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.** { *;}
```

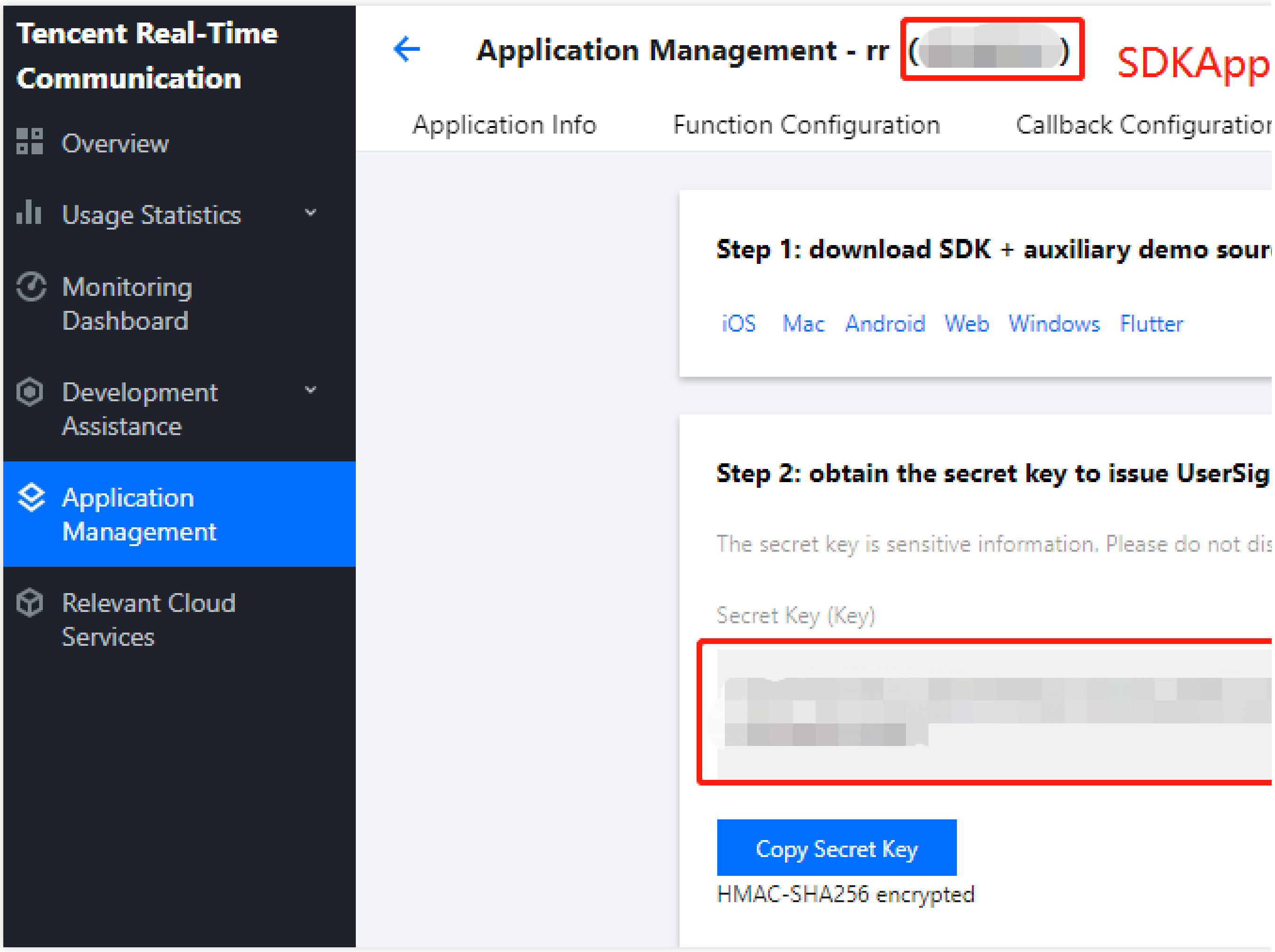
Step 3. Initialize and log in to the component

```
// 1. Add a listener and log in
TUILogin.addLoginListener(new TUILoginListener() {
    @Override
    public void onConnecting() {      // Connecting …
        super.onConnecting();
    }
    @Override
    public void onConnectSuccess() {  // Connected
        super.onConnectSuccess();
    }
    @Override
    public void onConnectFailed(int errorCode, String errorMsg) {  // Failed to con
        super.onConnectFailed(errorCode, errorMsg);
    }
    @Override
    public void onKickedOffline() {  // Callback for forced logout (for example, du
        super.onKickedOffline();
    }
    @Override
    public void onUserSigExpired() { // Callback for `userSig` expiration
        super.onUserSigExpired();
    }
});
TUILogin.login(mContext, "Your SDKAppID", "Your userId", "Your userSig", new TUICal
    @Override
    public void onSuccess() {
    }
    @Override
    public void onError(int errorCode, String errorMsg) {
        Log.d(TAG, "errorCode: " + errorCode + " errorMsg:" + errorMsg);
    }
});

// 2. Initialize the `TUILiveRoom` component
TUILiveRoom mLiveRoom = TUILiveRoom.sharedInstance(mContext);
```

Parameter description:

SDKAppID: **TRTC application ID**. If you haven't activated TRTC, log in to the TRTC console, create a TRTC application, click **Application Info**, and select the **Quick Start** tab to view its `SDKAppID` .



Secretkey: **TRTC application key**. Each secret key corresponds to a `SDKAppID` . You can view your application's secret key on the Application Management page 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 (_). We recommend that you keep it consistent with your user account system.

UserSig: Security signature calculated based on `SDKAppID` , `userId` , and `Secretkey` . You can click here to quickly generate a `UserSig` for testing. For more information, see UserSig.

Step 4. Implement an interactive video live room

1. **The anchor starts streaming**.

```
mLiveRoom.createRoom(int roomId, String roomName, String coverUrl);
```

2. **The audience member watches**.

```
mLiveRoom.enterRoom(roomId);
```

3. **Audience member and the anchor co-anchor together through** TRTCLiveRoom#requestJoinAnchor.

```
// 1. The audience member sends a co-anchoring request
// `LINK_MIC_TIMEOUT` is the timeout period
TRTCLiveRoom mTRTCLiveRoom=TRTCLiveRoom.sharedInstance(mContext);
mTRTCLiveRoom.requestJoinAnchor(mSelfUserId + "requested to co-anchor", LINK_MIC_TI
 new TRTCLiveRoomCallback.ActionCallback() {
 @Override
 public void onCallback(int code, String msg) {
     if (code == 0) {
         // The request is accepted by the anchor
         TXCloudVideoView view = new TXCloudVideoView(context);
         parentView.add(view);
         // The audience member turns on the camera and starts pushing streams
         mTRTCLiveRoom.startCameraPreview(true, view, null);
         mTRTCLiveRoom.startPublish(mSelfUserId + "_stream", null);
     }
 }
});

// 2. The anchor receives the co-anchoring request
mTRTCLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
 @Override
 public void onRequestJoinAnchor(final TRTCLiveRoomDef.TRTCLiveUserInfo userInfo,
     String reason, final int timeout) {
     // The anchor accepts the co-anchoring request
     mTRTCLiveRoom.responseJoinAnchor(userInfo.userId, true, "agreed to co-anchor")
 }

 @Override
 public void onAnchorEnter(final String userId) {
     // The anchor receives a notification that the co-anchoring audience member ha
     TXCloudVideoView view = new TXCloudVideoView(context);
     parentView.add(view);
     // The anchor plays the audience member's video
     mTRTCLiveRoom.startPlay(userId, view, null);
 }
});
```

4. **Anchors from different rooms communicate with each other by calling** TRTCLiveRoom#requestRoomPK.

```
// Create room 12345
mLiveRoom.createRoom(12345, "roomA", "Your coverUrl");
// Create room 54321
mLiveRoom.createRoom(54321, "roomB", "Your coverUrl");

// Anchor A:
TRTCLiveRoom mTRTCLiveRoom=TRTCLiveRoom.sharedInstance(mContext);

// 1. Send a cross-room communication request to anchor B
mTRTCLiveRoom.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
     }
 }
});

mTRTCLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
 @Override
 public void onAnchorEnter(final String userId) {
     // 6. Receive a notification about anchor B’s entry
     mTRTCLiveRoom.startPlay(userId, mTXCloudVideoView, null);
 }
});

// Anchor B:
// 2. Receive anchor A’s request
mTRTCLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
 @Override
 public void onRequestRoomPK(
    final TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, final int timeout) {
     // 3. Accept anchor A's request
     mTRTCLiveRoom.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
     mTRTCLiveRoom.startPlay(userId, mTXCloudVideoView, null);
 }
});
```

Suggestions and Feedback

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

Integrating TUILiveRoom (iOS)

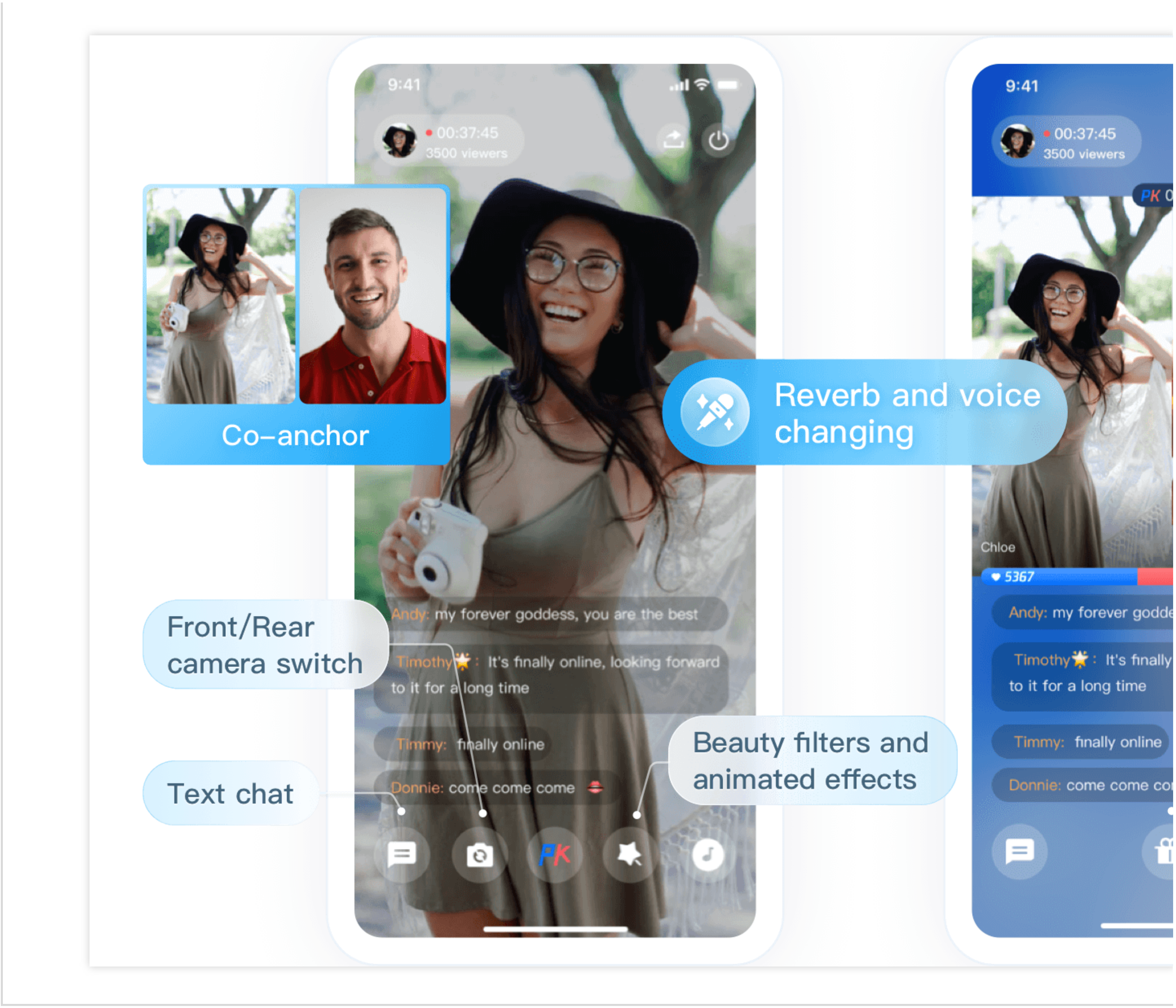
Last updated：2023-11-20 17:51:26

Overview

`TUILiveRoom` is an open-source video live streaming scenario UI component. After integrating it into your project, you can enable your application to support interactive video live streaming simply by writing a few lines of code. It provides source code for Android, iOS, and mini program platforms. Its basic features are as shown below:

Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see Pricing.


Co-anchor
Reverb and voice changing
Front/Rear camera switch
Text chat
Beauty filters and animated effects
9:41
00:37:45
3500 viewers
Andy: my forever goddess, you are the best
Timothy: It's finally online, looking forward to it for a long time
Timmy: finally online
Donnie: come come come
PK
Chloe
5367

Integration

Step 1. Import the `TUILiveRoom` component

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

1. Create a `TUILiveRoom` folder in the same directory as `Podfile` in your project.
2. Go to the component's GitHub page, clone or download the code, and copy the `Source` , `Resources` , `TUIBeauty` , `TUIAudioEffect` , `TUIBarrage` , `TUIGift` , and `TUIKitCommon` folders and the `TUILiveRoom.podspec` file in **TUILiveRoom/iOS/** to the `TUILiveRoom` folder in your project.
3. Add the following dependencies to your `Podfile` and run `pod install` to import the component.

```
# :path => "The relative path of `TUILiveRoom.podspec`"
pod 'TUILiveRoom', :path => "./TUILiveRoom/TUILiveRoom.podspec", :subspecs => ["TRT
# :path => "The relative path of `TUIKitCommon.podspec`"
pod 'TUIKitCommon', :path => "./TUILiveRoom/TUIKitCommon/"
# :path => "The relative path of `TUIBeauty.podspec`"
pod 'TUIBeauty', :path => "./TUILiveRoom/TUIBeauty/"
# :path => "The relative path of `TUIAudioEffect.podspec`"
pod 'TUIAudioEffect', :path => "./TUILiveRoom/TUIAudioEffect/"
# :path => "The relative path of `TUIBarrage.podspec`"
pod 'TUIBarrage', :path => "./TUILiveRoom/TUIBarrage/"
# :path => "The relative path of `TUIGift.podspec`"
```

```
pod 'TUIGift', :path => "./TUILiveRoom/TUIGift/"
```

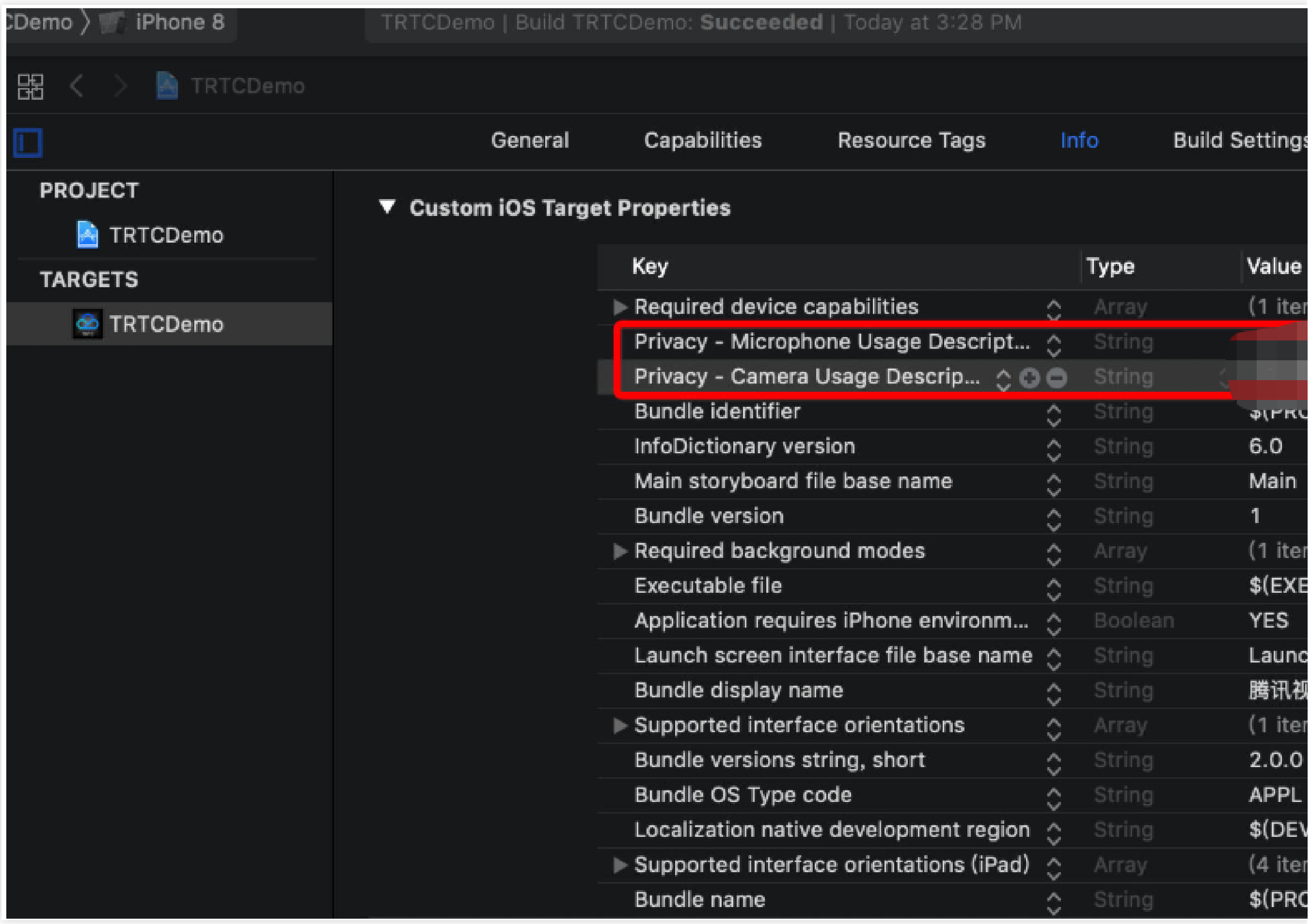
Note

The `Source` and `Resources` folders and the `TUILiveRoom.podspec` file must be in the same directory. `TUIKitCommon.podspec` is in the `TUIKitCommon` folder.

Step 2. Configure permissions

Your app needs mic and camera permissions to implement audio/video communication. Add the two items below to `Info.plist` of your app. Their content is what users see in the mic and camera access pop-up windows.

```
<key>NSCameraUsageDescription</key>
<string>RoomApp needs to access your camera to capture video.</string>
<key>NSMicrophoneUsageDescription</key>
<string>RoomApp needs to access your mic to capture audio.</string>
```



Demo 〉 iPhone 8 | TRTCDemo | Build TRTCDemo: **Succeeded** | Today at 3:28 PM

TRTCDemo

General | Capabilities | Resource Tags | Info | Build Settings

PROJECT
TRTCDemo
TARGETS
TRTCDemo

Custom iOS Target Properties

Key	Type	Value
▶ Required device capabilities	Array	(1 iter
Privacy - Microphone Usage Descript...	String	[redacted]
Privacy - Camera Usage Descrip...	String	[redacted]
Bundle identifier	String	$(PRC
InfoDictionary version	String	6.0
Main storyboard file base name	String	Main
Bundle version	String	1
▶ Required background modes	Array	(1 iter
Executable file	String	$(EXE
Application requires iPhone environm...	Boolean	YES
Launch screen interface file base name	String	Launc
Bundle display name	String	腾讯视
▶ Supported interface orientations	Array	(1 iter
Bundle versions string, short	String	2.0.0
Bundle OS Type code	String	APPL
Localization native development region	String	$(DEV
▶ Supported interface orientations (iPad)	Array	(4 iter
Bundle name	String	$(PRC

Step 3. Initialize and log in to the component

Objective-C

Swift

```
@import TUILiveRoom;
@import TUICore;

// 1. Log in to the component
[TUILogin login:@"Your SDKAppID" userID:@"Your UserID" userSig:@"Your UserSig" succ

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

}];
// 2. Initialize the `TUILiveRoom` instance
TUILiveRoom *mLiveRoom = [TUILiveRoom sharedInstance];
```

~~~
```
~~~

```
import TUILiveRoom
import TUICore

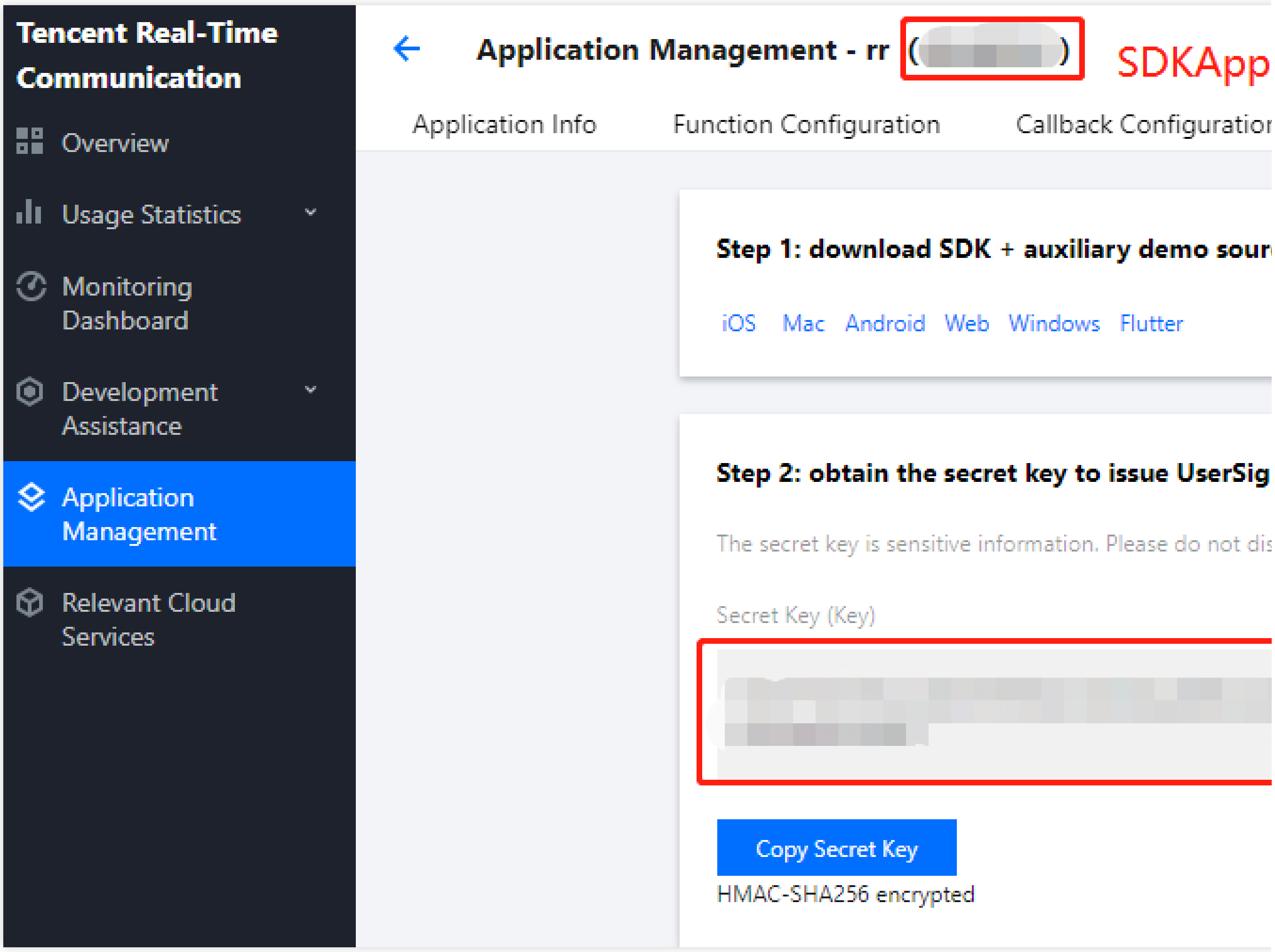
// 1. Log in to the component
TUILogin.login("Your SDKAppID", userID: "Your UserID", userSig: "Your UserSig") {

} fail: { code, msg in
```

```
}
// 2. Initialize the `TUILiveRoom` instance
let mLiveRoom = TUILiveRoom.sharedInstance
```
```

Parameter description:

SDKAppID: **TRTC application ID**. If you haven't activated TRTC, log in to the TRTC console, create a TRTC application, click **Application Info**, and select the **Quick Start** tab to view its `SDKAppID`.



Step 1: download SDK + auxiliary demo sour

Step 2: obtain the secret key to issue UserSig

The secret key is sensitive information. Please do not dis

HMAC-SHA256 encrypted

Secretkey: **TRTC application key**. Each secret key corresponds to an `SDKAppID`. You can view your application’s secret key on the Application Management page of the TRTC console.

UserId: Current user ID, which is a custom string that can contain up to 32 bytes of letters and digits (special characters are not supported).

UserSig: Security signature calculated based on `SDKAppID`, `userId`, and `Secretkey`. You can click here to quickly generate a `UserSig` for testing or calculate it on your own by referring to our TUILiveRoom demo project. For more information, see UserSig.
```

## Step 4. Implement an interactive video live room

1. **The anchor starts streaming**.

Objective-C

Swift

```
[mLiveRoom createRoomWithRoomId:123 roomName:@"test room" coverUrl:@""];
```

```
mLiveRoom.createRoom(roomId: 123, roomName: "test room", coverUrl:"")
```

2. **The audience member watches**.

Objective-C

Swift

```
[mLiveRoom enterRoomWithRoomId:123];
```

```
mLiveRoom.createRoom(roomId: 123)
```

3. **The audience member and the anchor co-anchor together through** TRTCLiveRoom#requestJoinAnchor.

Objective-C

Swift

```
// 1. The audience member sends a co-anchoring request
[TRTCLiveRoom shareInstance].delegate = self;
// @param mSelfUserId String Current user ID
NSString *mSelfUserId = @"1314";
[[TRTCLiveRoom shareInstance] requestJoinAnchor:[NSString stringWithFormat:@"%@ req
    if (agreed) {
        // The request is accepted by the anchor
      UIView *playView = [UIView new];
            [self.view addSubView:playView];
        // The audience member turns on the camera and starts pushing streams
      [[TRTCLiveRoom shareInstance] startCameraPreviewWithFrontCamera:YES view:play
```

```
        [[TRTCLiveRoom shareInstance] startPublishWithStreamID:[NSString stringWithFo
    }
}];

// 2. The anchor receives the co-anchoring request
#pragma mark - TRTCLiveRoomDelegate
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onRequestJoinAnchor:(TRTCLiveUser
    // The anchor accepts the co-anchoring request
    [[TRTCLiveRoom shareInstance] responseJoinAnchor:user.userId agree:YES reason:@
}

- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onAnchorEnter:(NSString *)userID
    // The anchor receives a notification that the co-anchoring audience member has
    UIView *playView = [UIView new];
    [self.view addSubview:playView];
    // The anchor plays the audience member's video
    [[TRTCLiveRoom shareInstance] startPlayWithUserID:userID view:playView callback
}
```

```
// 1. The audience member sends a co-anchoring request
TRTCLiveRoom.shareInstance().delegate = self
let mSelfUserId = "1314"
TRTCLiveRoom.shareInstance().requestJoinAnchor(reason: mSelfUserId + "requested to
    guard let self = self else { return }
  if agree {
        // The request is accepted by the anchor
        let playView = UIView()
        self.view.addSubView(playView)
        // The audience member turns on the camera and starts pushing streams
        TRTCLiveRoom.shareInstance().startCameraPreview(frontCamera: true, view: pl
        TRTCLiveRoom.shareInstance().startPublish(streamID: mSelfUserId + "_stream"
    }
}

// 2. The anchor receives the co-anchoring request
extension ViewController: TRTCLiveRoomDelegate {

    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRequestJoinAnchor user: TRTCL
        // The anchor accepts the co-anchoring request
        TRTCLiveRoom.shareInstance().responseRoomPK(userID: user.userId, agree: tru
    }

    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {
        // The anchor receives a notification that the co-anchoring audience member
        let playView = UIView()
        view.addSubview(playView)
        // The anchor plays the audience member's video
        TRTCLiveRoom.shareInstance().startPlay(userID: userID, view: playView);
    }
}
```

4. **Anchors from different rooms communicate with each other by calling** TRTCLiveRoom#requestRoomPK.

Objective-C

Swift

```
// Create room 12345
[[TUILiveRoom sharedInstance] createRoomWithRoomId:12345 roomName:@"roomA" coverUrl
// Create room 54321
[[TUILiveRoom sharedInstance] createRoomWithRoomId:54321 roomName:@"roomB" coverUrl

// Host A
// Send a cross-room communication request to anchor B
[[TRTCLiveRoom shareInstance] requestRoomPKWithRoomID:543321 userID:@"roomB userId"
    if (agreed) {
        // Anchor B accepts the request
    } else {
        // Anchor B rejects the request
    }
}];

// Anchor B:
// 2. Receive anchor A's request
#pragma mark - TRTCLiveRoomDelegate
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onRequestRoomPK:(TRTCLiveUserInfo
    // 3. Accept anchor A's request
    [[TRTCLiveRoom shareInstance] responseRoomPKWithUserID:user.userId agree:YES re
}

- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onAnchorEnter:(NSString *)userID
    // 4. Receive a notification about anchor A's entry and play anchor A's video
    [[TRTCLiveRoom shareInstance] startPlayWithUserID:userID view:playAView callbac
}
```

```
// Create room 12345
TUILiveRoom.sharedInstance.createRoom(roomId: 12345, roomName: "roomA")
// Create room 54321
TUILiveRoom.sharedInstance.createRoom(roomId: 54321, roomName: "roomB")

// Host A
// Send a cross-room communication request to anchor B
TRTCLiveRoom.shareInstance().requestRoomPK(roomID: 543321, userID: "roomB userId",
     guard let self = self else { return }
    if agreed {
        // Anchor B accepts the request
    } else {
        // Anchor B rejects the request
    }
}

// Anchor B:
// 2. Receive anchor A’s request
extension ViewController: TRTCLiveRoomDelegate {
    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRequestRoomPK user: TRTCLiveU
        // 3. Accept anchor A's request
        TRTCLiveRoom.shareInstance().responseRoomPK(userID: user.userId, agree: tru
    }

    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {
        // 4. Receive a notification about anchor A’s entry and play anchor A's vid
        TRTCLiveRoom.shareInstance().startPlay(userID: userID, view: playAView);
    }
}
```

Suggestions and Feedback

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

Integrating TUIPusher and TUIPlayer (Web)

Last updated：2023-09-25 10:56:13

Overview

`TUIPusher` and `TUIPlayer` are our web-based open-source components for interactive live streaming (UI included). You can use them together with our basic SDKs such as TRTC and Chat to quickly equip your live streaming applications (corporate live streaming, live shopping, vocational training, remote teaching, etc.) with web-based publishing and playback capabilities.

Note

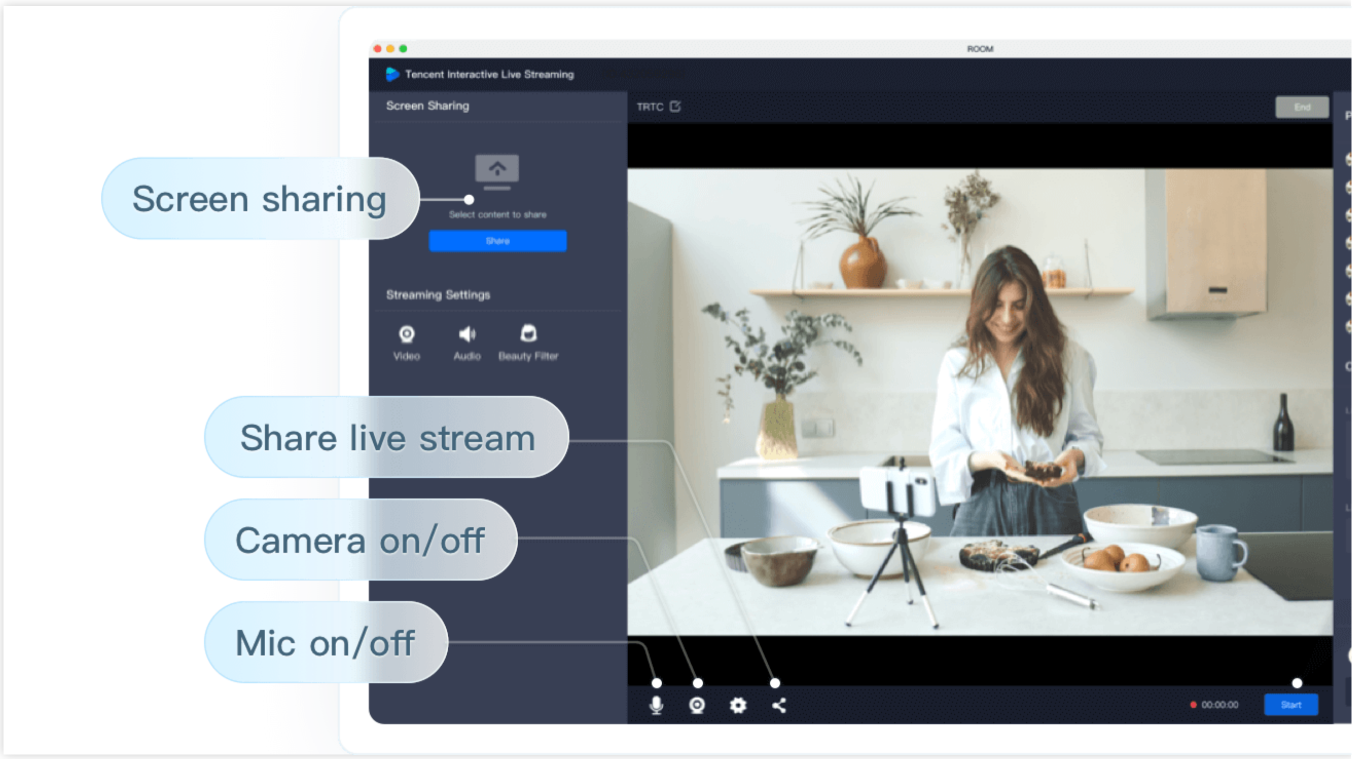
All components of TUIKit use two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, Chat and the trial edition of the Chat SDK (which supports only 100 DAUs) will be activated automatically. For the billing details of Chat, see Pricing.

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, Chat, and Superplayer, for excellent flexibility and scalability

Web-based, easy-to-use, and quick updates



Demos

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

Note

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

`TUIPusher` features

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` features

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

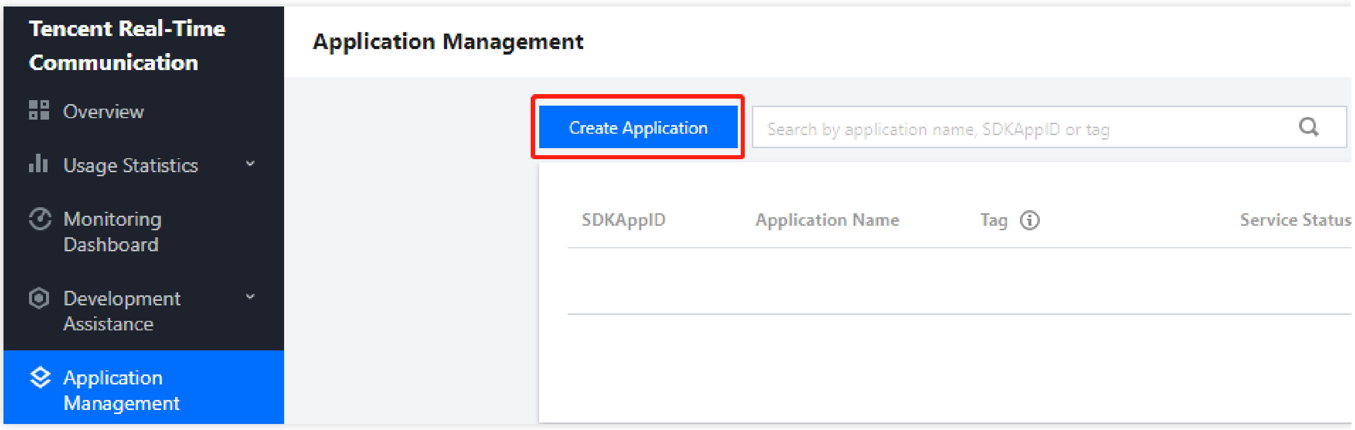
Step 1. Create an application

Note

`TUIPusher` and `TUIPlayer` are based on TRTC and Chat. Make sure you use the same `SDKAppID` for your TRTC and Chat applications so that they can share your account and authentication information.
You can use the basic content filtering capability of Chat to filter text messages. If you want to customize restricted words, go to the Chat 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 your server for a dynamic `UserSig`. For more information, see Generating UserSig.
Method 1: Via TRTC
Method 2: Via Chat

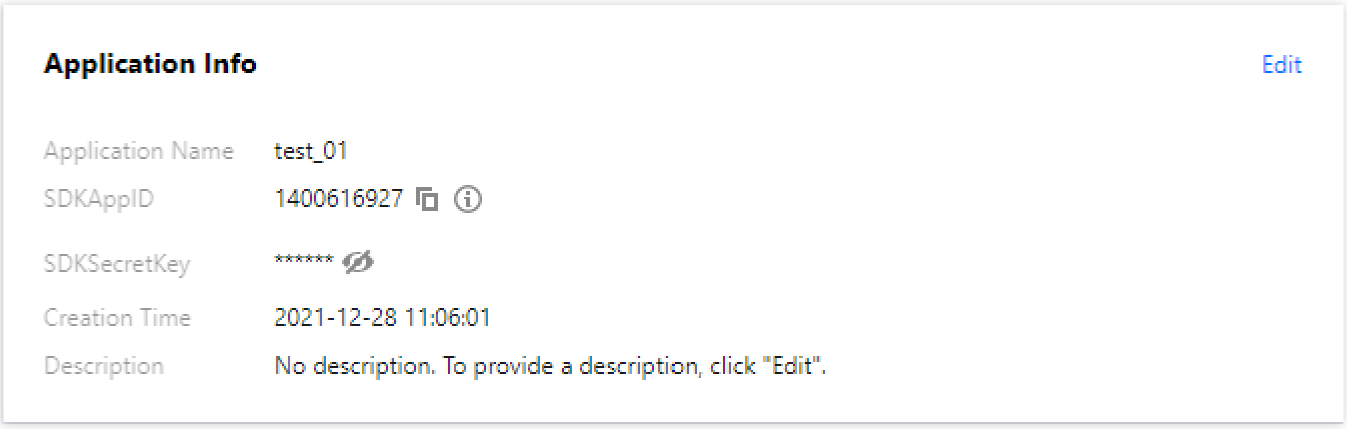
Step 1. Create a TRTC application

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

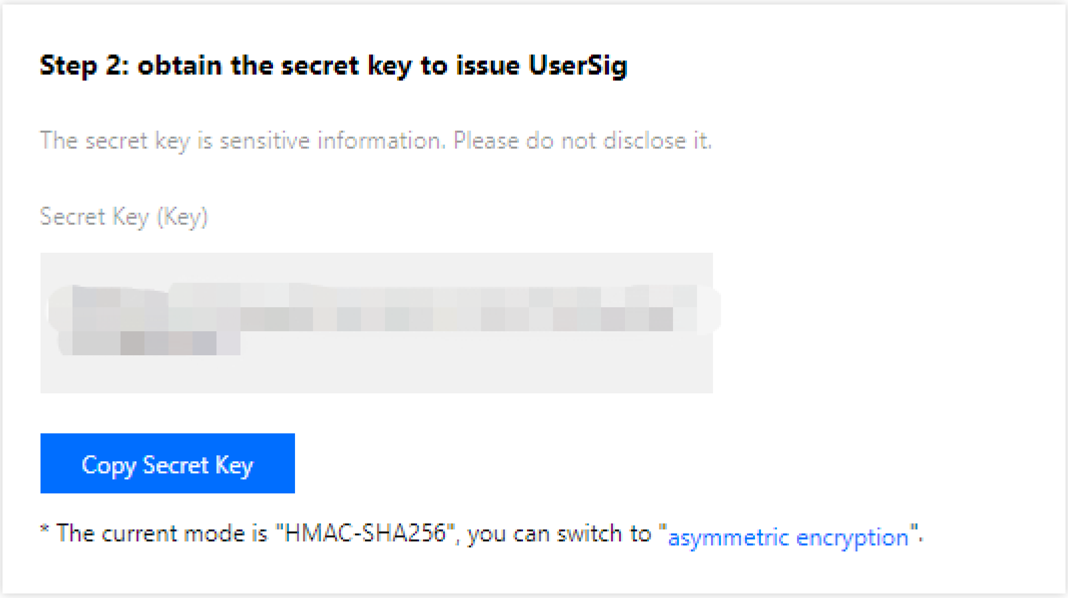
1. In the application list, find the application created and click **Application Info** to view the `SDKAppID`.



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]

Copy Secret Key

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

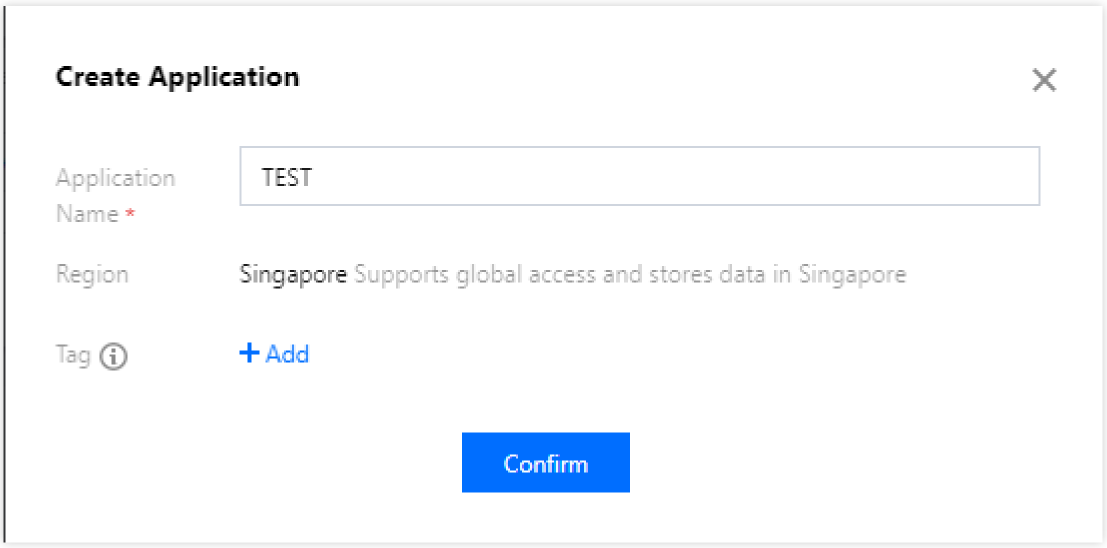
explain

Accounts creating their first application in the TRTC console will get a 10,000-minute free trial package.

After you create a TRTC application, a Chat application with the same SDKAppID will be created automatically. You can configure package information for the application in the Chat console.

Step 1. Create a Chat application

1. Log in to the Chat console, and click **Create Application**.

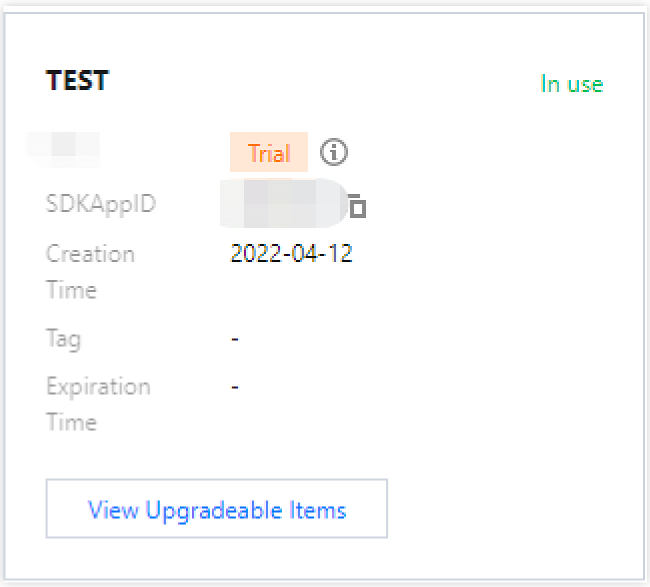

Create Application

Application Name * TEST

Region Singapore Supports global access and stores data in Singapore

Tag + Add

2. In the pop-up window, enter an application name and click **Confirm**.


TEST In use

Trial

SDKAppID

Creation Time 2022-04-12

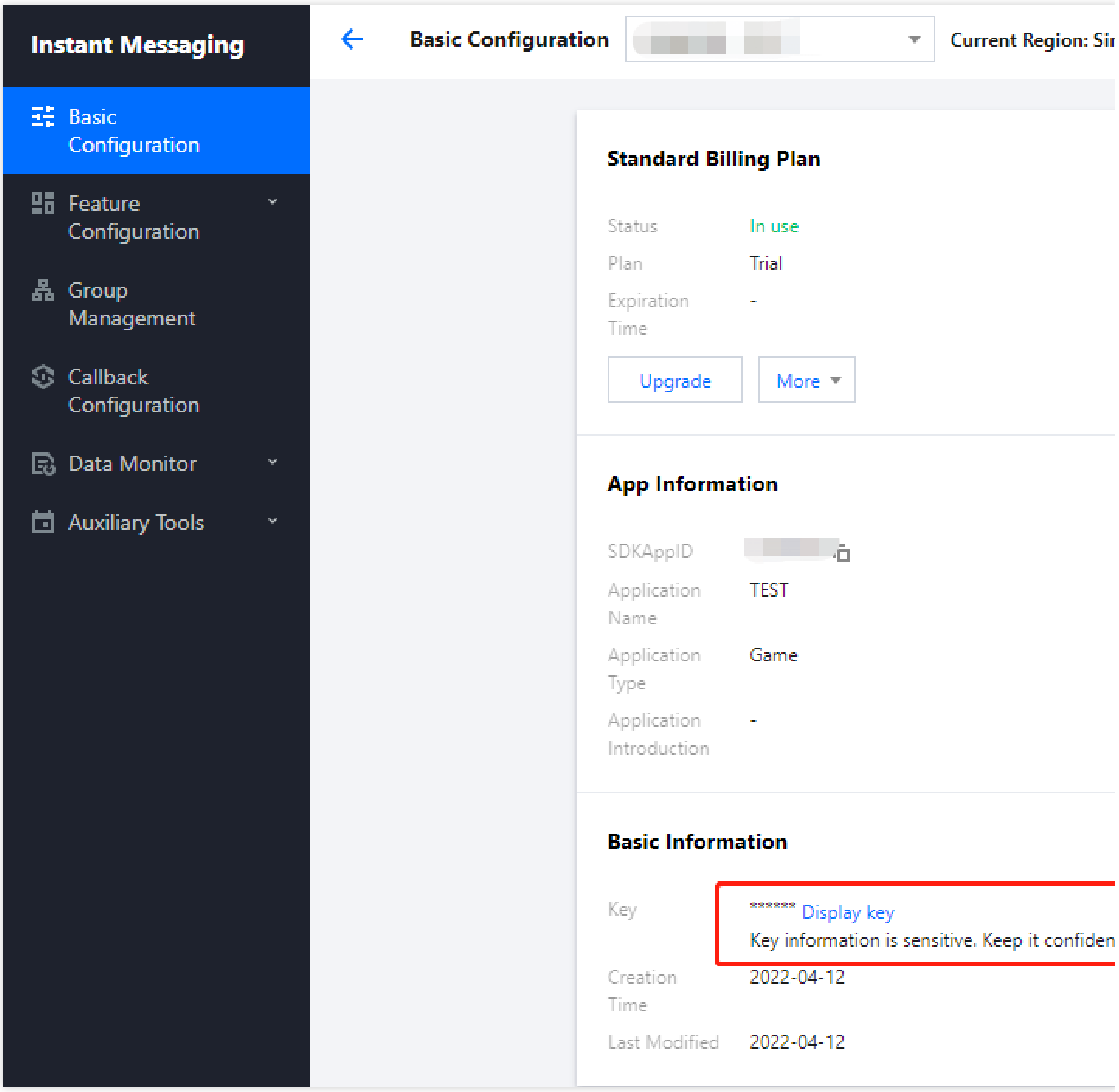
Tag -

Expiration Time -

3. Go to the overview page to view the status, edition, `SDKAppID` , creation time, and expiration time of the application created. Note down the `SDKAppID` .

Step 2. Obtain the key and activate TRTC

1. On the overview page, click the application created to go to the **Basic Configuration** page. In the **Basic Information** section, click **Display key**, and copy and save the key.



Standard Billing Plan

Status	In use
Plan	Trial
Expiration Time	-

App Information

SDKAppID	[redacted]
Application Name	TEST
Application Type	Game
Application Introduction	-

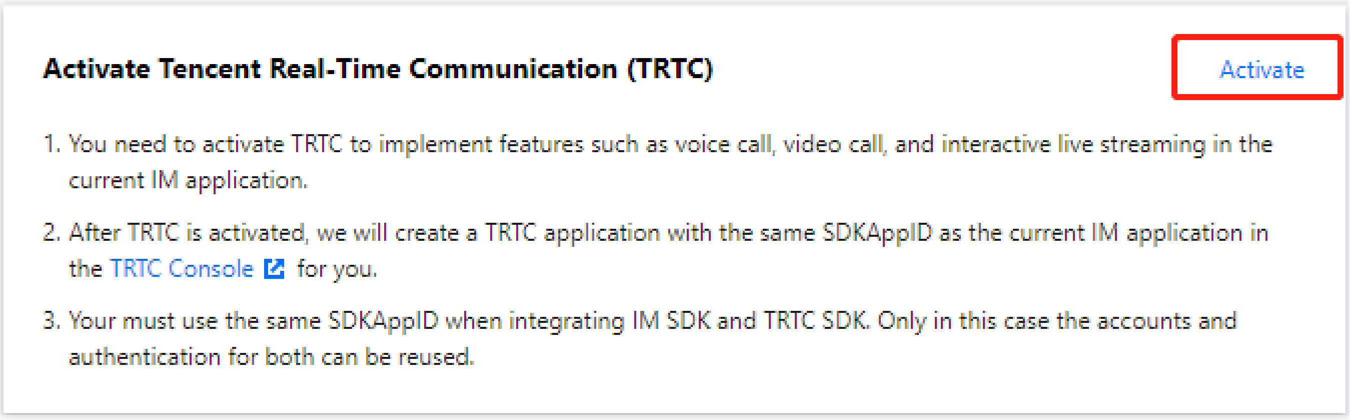
Basic Information

Key	****** Display key Key information is sensitive. Keep it confiden
Creation Time	2022-04-12
Last Modified	2022-04-12

Note

Please store the key information properly to prevent disclosure.

2. On the **Basic Configuration** page, activate TRTC.



Activate Tencent Real-Time Communication (TRTC)

1. You need to activate TRTC to implement features such as voice call, video call, and interactive live streaming in the current IM application.
2. After TRTC is activated, we will create a TRTC application with the same SDKAppID as the current IM application in the TRTC Console for you.
3. Your must use the same SDKAppID when integrating IM SDK and TRTC SDK. Only in this case the accounts and authentication for both can be reused.

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.
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 relay to CDN.

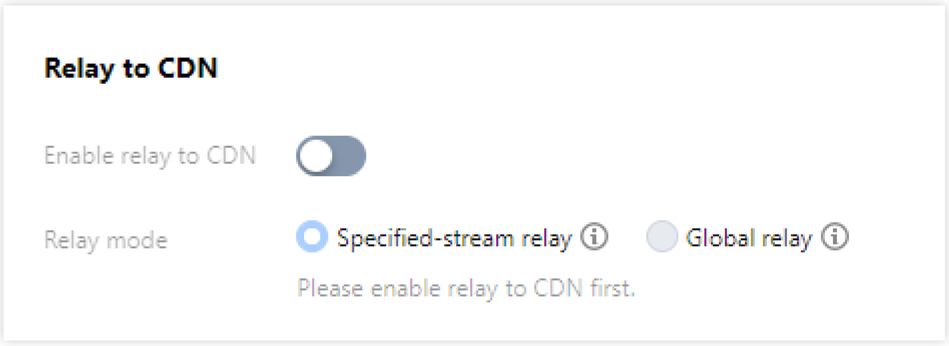
Local calculation of `UserSig` is for development and local debugging only and not for official launch. If your `SECRETKEY` is leaked, attackers will be able to steal your Tencent Cloud traffic.

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 your server for a dynamic `UserSig` . For more information, see Calculating UserSig on the server.

Step 3. Enable relay to CDN

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

1. In the TRTC console, enable relay to CDN for your application. You can choose **Specified-stream relay** or **Global relay** based on your needs.


Relay to CDN

Enable relay to CDN

Relay mode: [✓ Specified-stream relay] Global relay

Please enable relay to CDN 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

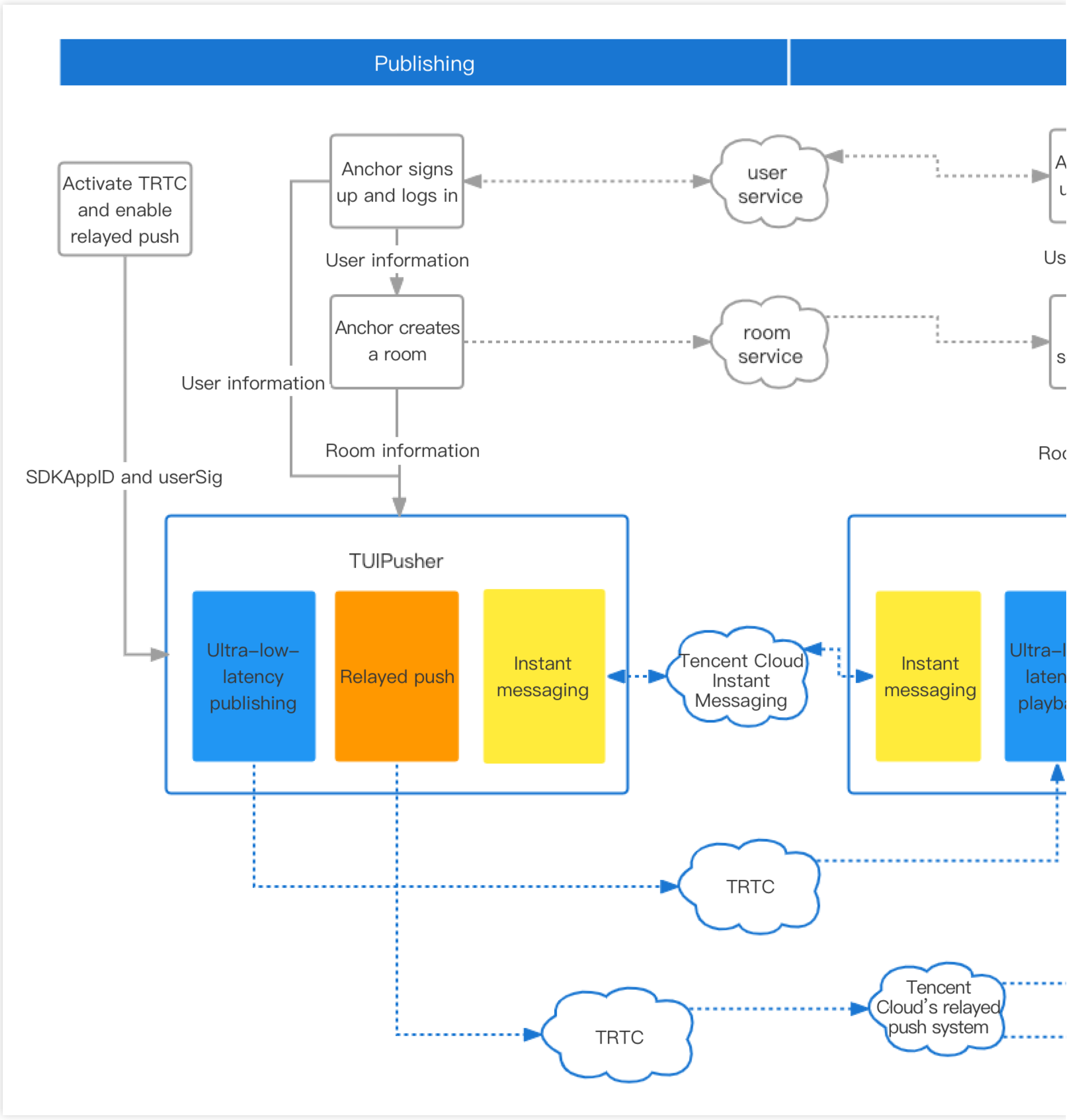
Generate `UserSig` on your server

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 disclosed, 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 your server for a dynamic `UserSig`. For more information, see How do I calculate UserSig during production?.

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



FAQs

How do I use beauty filters 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 over 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
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
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

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	-
Local development	`http://[local IP address]`	Supported	Not supported	Not supported	-

Firewall configuration

Firewall restrictions may cause audio/video calls to fail. To avoid this, add the ports and domains specified in Firewall Restrictions to the allowlist of your firewall.

Summary

In future versions, we plan to add support for communication between the web components and TRTC native SDKs (such as the iOS SDK and Android SDK), as well as 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.

TUILiveRoom APIs
TRTCLiveRoom APIs (iOS)

Last updated：2023-09-25 10:56:51

`TRTCLiveRoom` is based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat. With TRTCLiveRoom:

A user can create a room and start live streaming, or enter a room as an audience member.

The anchor can co-anchor with audience members in the room.

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.

Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see Pricing.

`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).

The TRTC SDK is used as a low-latency live streaming component.

The `AVChatRoom` feature of the Chat SDK is used to implement chat rooms, and Chat 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).
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 members 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 communication

API	Description
requestRoomPK	Sends a cross-room communication request (called by anchor).
responseRoomPK	Responds to a cross-room communication request (called by anchor).
quitRoomPK	Quits cross-room communication.

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 callbacks

API	Description
onError	Callback for error.
onWarning	Callback for warning.
onDebugLog	Callback of 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 co-anchoring.
onAudienceEnter	An audience member entered the room.
onAudienceExit	An audience member 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 cross-room communication events

API	Description
onRequestRoomPK	A cross-room communication request was received.
onQuitRoomPK	Cross-room communication 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

Login

```
/// Log in to the component.
/// - Parameters:
///   - sdkAppID: You can view `SDKAppID` in **[Application Management](https://con
///   - userID: ID of the current user, which is a string that can contain only let
///   - userSig: Tencent Cloud's proprietary security signature. For how to calcula
///   - config: global configuration information, which should be initialized durin
///   - callback: Callback for login. The return code is `0` if login is successful
/// - Note:
///   - We recommend setting the validity period of `userSig` to 7 days to avoid ca
- (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 described below:

Parameter	Type	Description
sdkAppId	Int	You can view `SDKAppID` in Application Management > **Application Info** of the TRTC console.
userID	String	The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_)
userSig	String	Tencent Cloud's proprietary security signature. For how to calculate and use it, see FAQs > 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

Log out

```
// Log out
/// - Parameter callback:  Callback for logout. The code is `0` if logout is succes
- (void)logout:(Callback _Nullable)callback
NS_SWIFT_NAME(logout(_:));
```

The parameters are described 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 Chat.
/// - Parameters:
///   - name: username
///   - avatarURL: profile picture URL
///   - callback: Callback for setting user profiles. The code is `0` if the operat
- (void)setSelfProfileWithName:(NSString *)name
                     avatarURL:(NSString * _Nullable)avatarURL
                      callback:(Callback _Nullable)callback
```

```
NS_SWIFT_NAME(setSelfProfile(name:avatarURL:callback:));
```

The parameters are described 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 c
/// The process of creating a room and starting live streaming as an anchor is as f
/// 1. A user calls `startCameraPreview()` to enable camera preview and set beauty 
/// 2. The user calls `createRoom()` to create a room, the result of which is retur
/// 3. The user calls `starPublish()` to push streams.
/// - Parameters:
///   - roomID: The room ID. You need to assign and manage the IDs in a centralized
///   - roomParam: room information, such as room name and cover information. If bo
///   - callback: Callback for room entry. The code is `0` if room entry is success
/// - Note:
///   - This API is called by an anchor to start live streaming. An anchor can crea
```

```
- (void)createRoomWithRoomID:(UInt32)roomID
                   roomParam:(TRTCCreateRoomParam *)roomParam
                    callback:(Callback _Nullable)callback
NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

The parameters are described below:

Parameter	Type	Description
roomID	UInt32	The 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 your own room lists.
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 ope
/// - 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 described 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 an audience member is a
/// 1. A user gets the latest room list from your server. The list may contain the
/// 2. The user selects a room and calls `enterRoom()` to enter the room.
/// 3. If the room list managed by your server contains the `userID` of the anchor
/// If the room list contains `roomID` only, upon room entry, the user will receive
/// The user can then call `startPlay(userID)`, passing in the `userID` obtained fr
/// - Parameters:
///   - roomID: The 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 success
/// - Note:
///   - This API is called by a user to enter a room.
///   - An anchor cannot use this API to enter a room he or she created, but must u
- (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 described below:

Parameter	Type	Description
roomID	UInt32	The 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 an audience member 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
/// - Note:
///   - This API is called by an audience member 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 described below:

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 c
/// - Parameter roomIDs: The 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 described 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
/// - Parameter callback: Callback of user details.
- (void)getAnchorList:(UserListCallback _Nullable)callback
NS_SWIFT_NAME(getAnchorList(callback:));
```

The parameters are described 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 members in a room. It takes effect only if it is called after `enterRoom()` .

```
/// Get the information of all audience members in a room. This API takes effect on
/// - Parameter callback: Callback of user details.
- (void)getAudienceList:(UserListCallback _Nullable)callback
NS_SWIFT_NAME(getAudienceList(callback:));
```

The parameters are described below:

Parameter	Type	Description
callback	(_ code: Int, _ message: String, _ userList: [TRTCLiveUserInfo]) ->	Callback of user

	Void	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 described 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 (push streams). This API can be called in the following sc
/// 1. An anchor starts live streaming.
/// 2. An audience member starts co-anchoring.
/// - Parameters:
///   - streamID: the `streamID` used to bind live streaming CDNs. You need to set
///   - callback: Callback for the operation
- (void)startPublishWithStreamID:(NSString *)streamID
                        callback:(Callback _Nullable)callback
NS_SWIFT_NAME(startPublish(streamID:callback:));
```

The parameters are described 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 the audience to receive the anchor's stream via CDNs.
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
/// 1. An anchor ends live streaming.
/// 2. An audience member ends co-anchoring.
/// - Parameter callback: Callback for the operation.
- (void)stopPublish:(Callback _Nullable)callback
NS_SWIFT_NAME(stopPublish(callback:));
```

The parameters are described 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
/// Common playback scenario
/// 1. If the room list managed by your server contains the `userID` of the anchor
/// 2. If the room list contains `roomID` only, upon room entry, a user will receiv
/// The user can then call `startPlay(userID)`, passing in the `userID` obtained fr
/// Co-anchoring scenario
/// After co-anchoring starts, the anchor will receive the `onAnchorEnter(userID)`
/// - Parameters:
///   - userID: The 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 described 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: The 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 described 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 an audience member sends a co-anchoring request, the anchor will
- (void)requestJoinAnchor:(NSString *)reason
                  timeout:(double)timeout
         responseCallback:(ResponseCallback _Nullable)responseCallback
NS_SWIFT_NAME(requestJoinAnchor(reason:timeout:responseCallback:));
```

The parameters are described 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 members 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 `
- (void)responseJoinAnchor:(NSString *)userID
                     agree:(BOOL)agree
                    reason:(NSString *)reason
NS_SWIFT_NAME(responseJoinAnchor(userID:agree:reason:));
```

The parameters are described below:

Parameter	Type	Description
userID	String	The user ID of the audience member.
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
- (void)kickoutJoinAnchor:(NSString *)userID
                 callback:(Callback _Nullable)callback
NS_SWIFT_NAME(kickoutJoinAnchor(userID:callback:));
```

The parameters are described below:

Parameter	Type	Description
userID	String	The ID of the user to remove from co-anchoring.
callback	(_ code: Int, _ message: String?) -> Void	Callback for the operation

APIs for Cross-Room Communication

requestRoomPK

This API is used to send a cross-room communication request (called by anchor).

```
/// Request cross-room communication
/// - 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 communication request is sent, the invited anchor wi
- (void)requestRoomPKWithRoomID:(UInt32)roomID
                         userID:(NSString *)userID
               responseCallback:(ResponseCallback _Nullable)responseCallback
NS_SWIFT_NAME(requestRoomPK(roomID:userID:responseCallback:));
```

The parameters are described below:

Parameter	Type	Description
roomID	UInt32	room ID of the anchor to invite
userID	String	user ID of the anchor to invite
responseCallback	(_ agreed: Bool, _ reason: String?) -> Void	Callback of the response

Anchors in different rooms can communicate with each other. The process of starting cross-room communication between anchor A and anchor B is as follows:

1. **Anchor A** calls `requestRoomPK()` to send a cross-room communication to anchor B.
2. **Anchor B** receives the `onRequestRoomPK()` callback of `TRTCLiveRoomDelegate` .
3. **Anchor B** calls `responseRoomPK()` to respond to the cross-room communication 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 communication request (called by anchor), after which the request sending anchor will receive the `responseCallback` passed in to `requestRoomPK` .

```
/// Respond to a cross-room communication request
/// Respond to a communication 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 anchor sending the reques
- (void)responseRoomPKWithUserID:(NSString *)userID
                           agree:(BOOL)agree
                          reason:(NSString *)reason
NS_SWIFT_NAME(responseRoomPK(userID:agree:reason:));
```

The parameters are described below:

Parameter	Type	Description
userID	String	User ID of the request sending anchor
agree	Bool	`true` : accept; `false` : reject
reason	String?	Reason for accepting/rejecting the request

quitRoomPK

This API is used to quit cross-room communication. If either anchor quits cross-room communication, the other anchor will receive the `trtcLiveRoomOnQuitRoomPK()` callback of `TRTCLiveRoomDelegate` .

```
/// End cross-room communication
/// - Parameter callback: Callback for ending cross-room communication
// - Note: If either anchor ends cross-room communication, the other anchor will re
- (void)quitRoomPK:(Callback _Nullable)callback
NS_SWIFT_NAME(quitRoomPK(callback:));
```

The parameters are described 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 described below:

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 described 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 described 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 described 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 described 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
 * - Adjust the hairline, eye spacing, eye corners, lip shape, nose wings, nose posi
 * - 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 chat message in a room, which is generally used for on-screen comments.

```
/// Send a text chat message that can be seen by all users in a room
/// - Parameters:
///   - message: text chat message
///   - callback: Callback for message sending
- (void)sendRoomTextMsg:(NSString *)message callback:(Callback _Nullable)callback
NS_SWIFT_NAME(sendRoomTextMsg(message:callback:));
```

The parameters are described 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
///    - message: text message
///    - callback: Callback for message sending
- (void)sendRoomCustomMsgWithCommand:(NSString *)command message:(NSString *)messag
NS_SWIFT_NAME(sendRoomCustomMsg(command:message:callback:));
```

The parameters are described 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 described 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 described 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 described 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 described 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 described 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 cross-room communication scenarios.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
    onRoomInfoChange:(TRTCLiveRoomInfo *)info
NS_SWIFT_NAME(trtcLiveRoom(_:onRoomInfoChange:));
```

The parameters are described 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 described 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 cross-room communication/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 described below:

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 described 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 described 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 described 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 described below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance

Callback APIs for Cross-Room Communication Events

onRequestRoomPK

Callback for receiving a cross-room communication 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 described below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
user	TRTCLiveUserInfo	Information of the anchor who requested communication.
timeout	Double	Timeout period for response from the anchor

onQuitRoomPK

Callback for ending cross-room communication.

```
 - (void)trtcLiveRoomOnQuitRoomPK:(TRTCLiveRoom *)liveRoom
NS_SWIFT_NAME(trtcLiveRoomOnQuitRoomPK(_:));
```

The parameters are described below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance

Message Event Callback APIs

onRecvRoomTextMsg

Callback for receiving a text chat message.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
   onRecvRoomTextMsg:(NSString *)message
            fromUser:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onRecvRoomTextMsg:fromUser:));
```

The parameters are described below:

Parameter	Type	Description
trtcLiveRoom	TRTCLiveRoomImpl	Current `TRTCLiveRoom` component instance
message	String	Text message
user	TRTCLiveUserInfo	Information of the sender

onRecvRoomCustomMsg

A custom message was received.

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRecvRoomCustomMsgWithCommand:(NSString *)command
              message:(NSString *)message
             fromUser:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onRecvRoomCustomMsg:message:fromUser:));
```

The parameters are described 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：2024-03-11 10:27:28

`TRTCLiveRoom` includes the following features which are based on Tencent Real-Time Communication (TRTC) and Tencent Cloud Chat.

A user can create a room and start live streaming, or enter a room as an audience member.

The anchor can co-anchor with audience members in the room.

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.

Note

All TUIKit components are based on two basic PaaS services of Tencent Cloud, namely TRTC and Chat. When you activate TRTC, the Chat SDK trial edition (which supports up to 100 DAUs) will be activated automatically. For Chat billing details, see Pricing.

`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).

The TRTC SDK is used as a low-latency audio chat component.

The `AVChatRoom` feature of the Chat SDK is used to implement chat rooms. The attribute APIs of Chat 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.

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 members 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 communication

API	Description
requestRoomPK	Sends a cross-room communication request (called by anchor).
responseRoomPK	Responds to a cross-room communication request (called by anchor).
quitRoomPK	Quits cross-room communication.

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 chat 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 callbacks

API	Description
onError	Callback for error.
onWarning	Callback for warning.
onDebugLog	Callback of 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	An audience member entered the room.
onAudienceExit	An audience member 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 cross-room communication events

API	Description
onRequestRoomPK	A cross-room communication request was received.
onQuitRoomPK	Cross-room communication ended.

Message event callback APIs

API	Description
onRecvRoomTextMsg	A text chat 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 described 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 the 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 described 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

Login

```
public abstract void login(int sdkAppId,
 String userId, String userSig,
 TRTCLiveRoomDef.TRTCLiveRoomConfig config,
 TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are described below:

Parameter	Type	Description

sdkAppId	int	You can view `SDKAppID` in Application Management > **Application Info** of the TRTC console.
userId	String	The ID of the current user, which is a string that can contain only letters (a-z and A-Z), digits (0-9), hyphens (-), and underscores (_)
userSig	String	Tencent Cloud's proprietary security signature. For how to calculate and use it, see FAQs > UserSig.
config	TRTCLiveRoomConfig	Global configuration information, which needs to be initialized during login and cannot be modified afterward. `useCDNFirst` : Specifies the way the audience watches live streams. `true` means that the audience watches live streams over CDNs, which is cost-efficient but has high latency. `false` means that the audience watches 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. It takes effect only if `useCDNFirst` is set to `true` . You can set it in Domain Management of the CSS console.
callback	ActionCallback	The callback for login. The code is `0` if login succeeds.

logout

Log out

```
public abstract void logout(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are described 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, TRTCLiveRoom
```

The parameters are described below:

Parameter	Type	Description
userName	String	Username
avatarURL	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 roo
```

The parameters are described below:

Parameter	Type	Description

roomId	int	The 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 your own room lists.
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 `starPublish()` 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 described 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 call
```

The parameters are described below:

Parameter	Type	Description
roomId	int	The 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 an audience member 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 described 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.Ro
```

The parameters are described 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 and co-anchoring viewers in a room. It takes effect only if it is called after `enterRoom()` .

```
public abstract void getAnchorList(TRTCLiveRoomCallback.UserListCallback callback);
```

The parameters are described below:

Parameter	Type	Description
callback	UserListCallback	Callback of user details

getAudienceList

This API is used to get the information of all audience members in a room. It takes effect only if it is called after `enterRoom()` .

```
public abstract void getAudienceList(TRTCLiveRoomCallback.UserListCallback callback
```

The parameters are described 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, TRT
```

The parameters are described 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.

An audience member starts co-anchoring.

```
public abstract void startPublish(String streamId, TRTCLiveRoomCallback.ActionCallb
```

The parameters are described 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 (pushing streams), which can be called in the following scenarios:

An anchor ends live streaming.

An audience member ends co-anchoring.

```
public abstract void stopPublish(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are described 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, TRTCLiveRoomCa
```

The parameters are described 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 ca
```

The parameters are described 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, TRTCLiveRoomCall
```

The parameters are described below:

Parameter	Type	Description

reason	String	Reason for co-anchoring
timeout	int	Timeout period
callback	ActionCallback	Callback of the anchor’s response

The process of co-anchoring between anchors and audience members 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 described below:

Parameter	Type	Description
userId	String	The user ID of the audience member.
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.ActionCa
```

The parameters are described below:

Parameter	Type	Description
userId	String	The ID of the co-anchoring user.

callback	ActionCallback	Callback for the operation

APIs for Cross-Room Communication

requestRoomPK

This API is used to send a cross-room communication request (called by anchor).

```
public abstract void requestRoomPK(int roomId, String userId, TRTCLiveRoomCallback.
```

The parameters are described 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 communication.

Anchors in different rooms can communicate with each other. The process of starting cross-room communication between anchor A and anchor B is as follows:

1. **Anchor A** calls `requestRoomPK()` to send a ccross-room communication request to anchor B.
2. **Anchor B** receives the `onRequestRoomPK()` callback of `TRTCLiveRoomDelegate` .
3. **Anchor B** calls `responseRoomPK()` to respond to the cross-room communication 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 communication 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 described 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 communication. If either anchor quits cross-room communication, the other anchor will receive the `onQuitRoomPk()` callback of `TRTCLiveRoomDelegate` .

```
public abstract void quitRoomPK(TRTCLiveRoomCallback.ActionCallback callback);
```

The parameters are described 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 described 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 described 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 described 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 described 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 chat message in a room, which is generally used for on-screen comments.

```
public abstract void sendRoomTextMsg(String message, TRTCLiveRoomCallback.ActionCal
```

The parameters are described 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, TRTCLiveRoomCall
```

The parameters are described below:

Parameter	Type	Description
cmd	String	A custom command word used to distinguish between different message types
message	String	Text message
callback	ActionCallback	Callback for the operation

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 described 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 described 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 described below:

Parameter	Type	Description
code	int	Error code
message	String	Warning message

onDebugLog

Callback for log.

```
void onDebugLog(String message);
```

The parameters are described below:

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 described 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 cross-room communication scenarios.

```
void onRoomInfoChange(TRTCLiveRoomDef.TRTCLiveRoomInfo roomInfo);
```

The parameters are described 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. The 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 described below:

Parameter	Type	Description
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 cross-room communication/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 described 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 described 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 described below:

Parameter	Type	Description
userInfo	TRTCLiveUserInfo	Information of the viewer who left the room

Callback APIs Audience-Anchor Co-anchoring Events

onRequestJoinAnchor

Callback for receiving a co-anchoring request from a viewer.

```
void onRequestJoinAnchor(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, String reason,
```

The parameters are described 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. After receiving this callback, a co-anchoring viewer needs to call `stopPublish()` of `TRTCLiveRoom` to quit co-anchoring.

```
void onKickoutJoinAnchor();
```

Callback APIs for Cross-Room Communication Events

onRequestRoomPK

Callback for receiving a cross-room communication 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 described below:

Parameter	Type	Description
userInfo	TRTCLiveUserInfo	Information of the anchor who requested cross-room communication.
timeout	int	Timeout period for response from the anchor

onQuitRoomPK

Callback for ending cross-room communication.

```
void onQuitRoomPK();
```

Message Event Callback APIs

onRecvRoomTextMsg

Callback for receiving a text chat message.

```
void onRecvRoomTextMsg(String message, TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

The parameters are described below:

Parameter	Type	Description
message	String	Text message
userInfo	TRTCLiveUserInfo	Information of the sender

onRecvRoomCustomMsg

A custom message was received.

```
void onRecvRoomCustomMsg(String cmd, String message, TRTCLiveRoomDef.TRTCLiveUserIn
```

The parameters are described below:

Parameter	Type	Description
command	String	A custom command word used to distinguish between different message types
message	String	Text message

userInfo	TRTCLiveUserInfo	Information of the sender

TRTCAudioEffectManager

playBGM

Background Music

```
void playBGM(String url, int loopTimes, int bgmVol, int micVol, TRTCCloud.BGMNotify
```

The parameters are described 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

stop playing background music

```
void stopBGM();
```

pauseBGM

pause background music

```
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 described 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 described 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 described 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 described 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 described 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 vol
```

The parameters are described 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 described 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 described 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 described 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 described 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 described below:

Parameter	Type	Description
volume	int	Volume. Value range: 0-100. Default value: `100`