SparkRTC

Client SDK Reference

Issue 01

Date 2023-11-14

HUAWEI

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to “Vul. Response Process”. For details about the policy, see the following website:https://www.huawei.com/en/psirt/vul-response-process
For enterprise customers who need to obtain vulnerability information, visit:https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Before You Start

1.1 Main Functions

SparkRTC provides basic room functions and cross-room functions. **Figure 1-1** shows the main function framework of each platform.

Note: Only general functions are displayed in **Figure 1-1**. For details about respective functions, see the SDK guide of each platform.

Figure 1-1 Functional framework

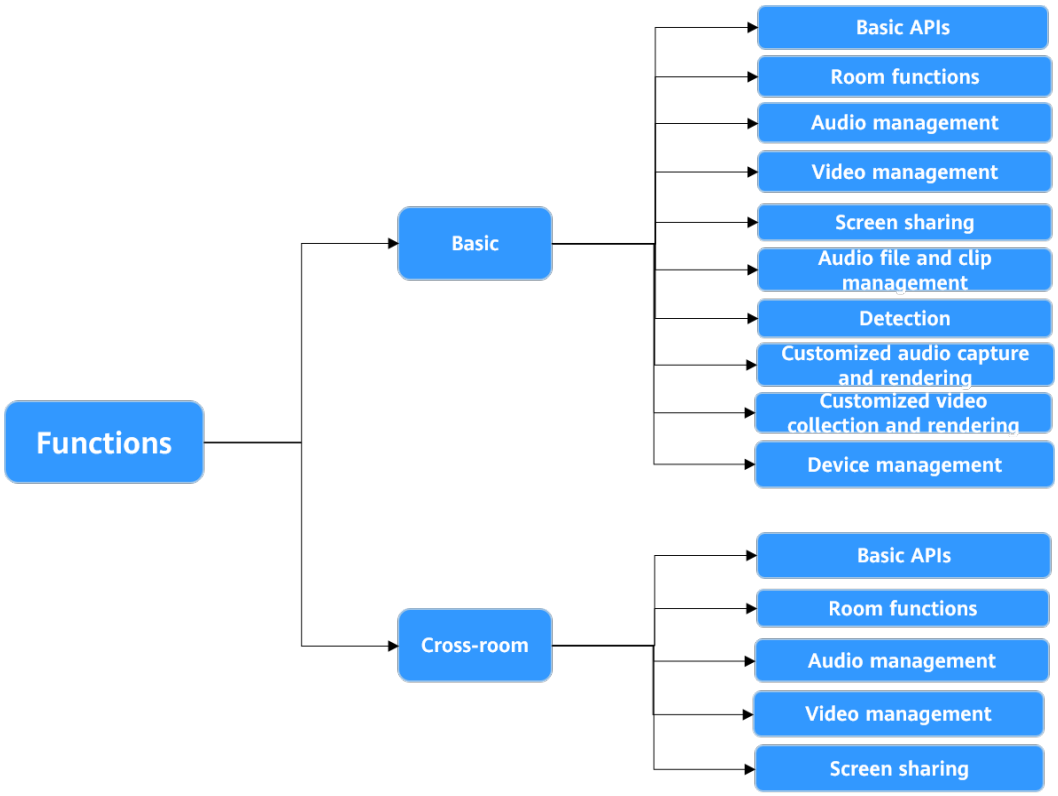


Table 1-1 Function description

Category	Function	Description
Basic	Basic functions	Creates and destroys the RTC engine and specifies the location for storing logs.
	Room functions	Joins/Leaves a room, sets the user role, and creates a cross-room engine.
	Audio management	Collects/Sends local audio streams, enables/disables remote audio stream receiving, adjusts the recording/playback volume, and sets the remote audio mode.
	Video management	Creates a local/remote view and configures related parameters, and specifies whether to receive remote video streams, mirrors, and cameras.
	Screen sharing	Enables/Disables presentation subscriptions, and sets the presentation rendering mode and rotation.
	Audio file and clip management	Starts/Stops/Pauses/Resumes the playback of audio files or audio clips, and adjusts the volume.
	Detection	Enables/Disables network detection before a user joins a meeting.
	Customized audio capture and rendering	Enables/Disables customized audio capture and external audio data pushes.
	Customized video capture and rendering	Enables/Disables customized video capture, external video data pushes, and customized video stream rendering.
	Device management	Switches cameras and sets the audio output device.

Category	Function	Description
Cross-room	Co-hosting across rooms refers to the scenario where the media stream of a live streamer is relayed to multiple room channels at the same time, implementing real-time communication between the live streamers. All the streamers can see each other in their rooms, and the audience in the rooms can see all the streamers. A maximum of four rooms can be connected at the same time. The ID of each room must be unique. Only one joiner can join a room at a time. If the local user is a joiner in another room, the user needs to change to a joiner from a player before joining that room. The user who joins other rooms as a player can receive streams but cannot send streams; joining rooms as a joiner allow both functions.	

1.2 How to Use This Document

- **1. Start from the basic structure.**

  A sequence diagram is provided to show you how to use the APIs on each platform. Click the name of an API in the diagram for details.
- **2: Get a glimpse of the API lists.**

  Know about the overall function flowchart and API lists of SparkRTC and quickly locate an API by function.
- **3: Pay attention to the "Caution"s.**

  Pay attention to the scenario and parameter description, especially the information in the "Caution" area, which describes the precautions for using the corresponding API and related callbacks.

1.3 FAQs

- Question 1: Why does a parameter error occurs when I call the **setVideoEncoderConfig** API?

  The resolution can be set successfully only when you comply with the code list recommended by the Huawei SDK system.
- Question 2: What are the possible causes for preventing users from joining other rooms?
  - 1: Only one joiner can join other rooms at a time.
  - 2: A user can join up to four rooms at a time, and the room IDs must be unique.
- Question 3: When the remote audio mode is set to **HRTC_REMOTE_AUDIO_SUBSCRIBED**, how do I mute the audio from remote users and make it a default setting?

  HRTC_REMOTE_AUDIO_SUBSCRIBED needs to be called manually for subscription. When a user joins a room (**joinRoom**), the method with the **HRTCJoinParam** class is called. After the instance of this class is created, you can set **autoSubscribeAudio** to **false**, so that the user, upon joining a room,

cannot hear the audio from remote users. To unmute the audio, manually call **muteRemoteAudio** and subscribe to a specified user by the user ID.

- Question 3: Why does the program crash when **onVideoStats**, **onAudioStatus**, or **onAuxiliaryStreamStatsNotify callback** is triggered?

  The pointers of the input parameters **localStats** and **remoteStats** of the callback function may be null. Ensure that the pointers are not null before you use the parameters. Otherwise, a null pointer error may occur.

- Question 4: Why can I hear my own voice from the local receiver?

  This is because you have set the user ID to your own when you call the **muteRemoteAudio** API.

- Question 5: Can I enable **setExternalAudioCapture** or **setExternalVideoCapture** in a room?

  No, they need to be called before a user joins a room.

2 SDK Overview

Software development kits (SDKs) of Huawei Cloud SparkRTC encapsulate the Representational state transfer (REST) application programming interfaces (APIs) provided by SparkRTC to simplify development. You can directly call API functions provided by the SDKs to use SparkRTC capabilities.

Submit a service ticket to contact Huawei Cloud technical service to obtain development packages. For details about Web SDK integration operations, API parameter definitions, and code examples, see **Web SDK Reference** and **Web API Reference**.

3 Web SDK

3.1 Browser Adaptation

This section describes the browser types, versions, and restrictions supported by the Web SDK.

Table 3-1 Browser adaptation

OS	Browser Type	Browser Version	SDK Version Constraints	Downlink (Playback)	Uplink (Speak)	Screen Sharing
Windows	Chrome browser	67+	• v1.10.0 or later • v2.0.0 or later	Supported	Supported	Supported (Chrome 73 or later)
	QQ Browser (fast kernel)	10.4+				Not supported

OS	Browser Type	Browser Version	SDK Version Constraints	Downlink (Playback)	Uplink (Speak)	Screen Sharing
	360 Secure Browser (fast mode)	12				Supported
	WeChat embedded browser	-	v2.0.0 or later	Supported	Not supported	Not supported
	Enterprise WeChat embedded browser	-				
	Mozilla Firefox	90+	v2.0.1 or later	Supported	Supported	Supported
	Microsoft Edge	80+	v2.0.2 or later	Supported	Supported	Supported
	Sogou Browser (high-speed mode)	11+				
	Opera browser	54+				Supported (Opera 60+ versions)
macOS	Chrome browser	67+	• v1.10.0 or later • v2.0.0 or later	Supported	Supported	Supported (Chrome 73 or later)
	WeChat embedded browser	-	v2.0.0 or later	Supported	Not supported	Not supported
	Enterprise WeChat embedded browser	-				
	Safari	11+	v2.0.1 or later	Supported	Supported	Supported (Safari 13+ versions)

OS	Browser Type	Browser Version	SDK Version Constraints	Downlink (Playback)	Uplink (Speak)	Screen Sharing
	Mozilla Firefox	90+				Supported
	Microsoft Edge	80+				
	Opera browser	56+	v2.0.2 or later	Supported	Supported	Supported (Opera 60+ versions)
Android	WeChat embedded browser (TBS kernel)	-	• v1.10.0 or later • v2.0.0 or later	Supported	Supported	Not supported
	WeChat embedded browser (XWEB kernel)	-				
	Enterprise WeChat embedded browser	-				
	Mobile Google Chrome	-				
	Mobile QQ Browser	12+				
	Mobile Huawei Browser	11.0.8+				
	Mobile UC Browser	-	-	Not supported	Not supported	Not supported
iOS	WeChat embedded browser	iOS 14.3+ WeChat 6.5+	• v1.10.0 or later • v2.0.0 or later	Supported	Supported	Not supported

OS	Browser Type	Browser Version	SDK Version Constraints	Downlink (Playback)	Uplink (Speak)	Screen Sharing
	Mobile Google Chrome	iOS 14.3+	v2.0.0 or later	Supported	Supported	Not supported
	Enterprise WeChat embedded browser	-			Not supported	
	Mobile Safari	11+	v2.0.1 or later	Supported	Supported	Not supported

Table 3-2 Constraints on browsers

Browser Type	Constraints
Chrome browser	• In the mobile browser, the **getSpeakers** API can obtain only the default audio output device. • On Huawei mobile devices, the Chrome browser (including the Huawei built-in browser) supports WebRTC 91+. • Before using screen sharing on the Chrome of a macOS device, ensure that Chrome screen recording authorization has been enabled in **Settings** > **Security & Privacy** > **Privacy** > **Screen Recording**.

Browser Type	Constraints
Safari	• On Safari 11 and 12, the navigator.mediaDevices.getUserMedia API needs to be called (the **createStream** API in the SDK is called to create local streams) to obtain the media permission before link establishment. Otherwise, media interaction fails (the media link cannot be established). • When Huawei Native SDK pushes a stream and you watch the stream in Safari 11 or 12, a green screen may be displayed on your device. • Safari 13 users may not hear the voice of remote users. • The audio on iOS Safari 14.2 and macOS Safari 14.0.1 may be intermittent. • An exception may occur when you push stream on Safari 15.1. As a result, the page breaks down. • Safari does not support the **getSpeakers** and **setAudioOutput** APIs because it cannot obtain output device information. • Safari does not support the **addTrack** and **removeTrack** APIs. • The Safari browser cannot call the local stream collection API for multiple times. Otherwise, a black screen may be displayed during collection. The previous collection must be closed before the audio and video collection API is called. • Safari does not support high-quality and low-quality streams. • The iOS mobile browser does not support audio mixing. • If you insert a headset to a macOS device with Safari 12 and enable audio mixing, remote users cannot hear your voice. • You must call the navigator.mediaDevices.getUserMedia API before hearing the audio of a remote user. Otherwise, you cannot hear the audio or obtain the audio volume.
Mozilla Firefox	• Firefox supports only 30 fps video frame rate. • Firefox on macOS devices with Apple M1 chips does not support H.264 codec. • Firefox does not support the **getSpeakers** and **setAudioOutput** APIs because it does not support the function of obtaining output device information. • When you push streams on Firefox 97 or later, black screen and frame freeze issues may occur on other clients. This issue is being resolved urgently. Use another browser to push stream.
Opera browser	On Huawei mobile devices, the Opera browser supports WebRTC 64 or later.
Other	The support for WebRTC varies with mobile browsers due to factors such as browser kernels, webviews, and versions of different vendors. In addition to the mobile browsers listed in **Table 3-1**, native SDK (Android/iOS) can be integrated.

CAUTION

- For users who have integrated Web SDK 1.0+ (2.0+ is not involved), upgrade it to 1.10.0+ as soon as possible. Otherwise, Web SDK may fail to be used on Chrome 96 or later.
- Web SDK 2.0+ is the mainstream build version. It provides new functions and optimizes user experience. You are advised to integrate it first. Only maintenance is provided for existing users of Web SDK 1.0+. New functions are not provided.
- Web SDK 1.0+ and Web SDK 2.0+ cannot interwork with each other.
- There are many constraints and known issues on the Safari browser. You are advised to use the Chrome browser or integrate the Native SDK.

3.2 Preparations

Prerequisites

You have **submitted a service ticket** to obtain the SDK package.

Environment Requirements

- You are advised to install Microsoft Visual Studio Code 1.43.2 or a later version for compilation.
- If you want to develop your client using Node.js, you are advised to install Node.js 14.19.1 or a later version.
- For details about the supported browsers, see **Web Browser Adaptation**.
- If you want to develop your client using TypeScript, the TypeScript version must be 3.8.3 or later.
- For browser security, the camera and microphone permissions can be requested only through https://*Domain name* or *localhost:port* after you build a local server.

SDK Integration

Step 1 Place the obtained SDK package in the **sdk** directory of your project.

Step 2 Introduce **hrtc** to the project code.

- To use the <script> mode to introduce Huawei WebRTC SDK, access HRTC to obtain the exported module:

```
<script src='./sdk/hrtc.js'>
    console.log(HRTC.VERSION)
</script>
```

- To directly reference the static JS file of Huawei WebRTC SDK, reference the following code to access HRTC:

```
import HRTC from '../../sdk/hrtc'
console.log(HRTC.VERSION)
```

- To introduce Huawei WebRTC SDK in npm modularization mode, install the hrtc module and introduce hrtc to the development dependency of package.json, for example, **"hrtc": "./sdk/RtcSdk_Web_*.*.*.***.tar.gz"**. Run

the **npm install** command on the terminal (replace the version number with the actual one), and then run the following commands:

```
import HRTC from 'hrtc'
console.log(HRTC.VERSION)
```

----End

3.3 SDK Usage

Step 1 Check whether a browser is compatible with the SDK. For details, see **checkSystemRequirements**.

```
async isBrowserSupport() {
  let check = false
  try {
    check = await HRTC.checkSystemRequirements()
    console.warn('browser isSupport: ' + check)
  } catch (error) {
    console.error(`check browser isSupport error: ${error.getCode()} - ${error.getMsg()}`)
    if (error.getCode() !== 90100025) {
      console.error(`browser Support part ability of RTC`)
      check = true
    }
  }
  return check
}
```

Step 2 Create a client. For details, see **createClient**.

```
let config = { appId,domain,countryCode }
let client = HRTC.createClient(config)
```

- **domain**: domain name of the server. The type is string[128]. This parameter is mandatory in SDK 1.0+ and optional in SDK 2.0+.
- **appId**: (mandatory) The type is string[128]. Only apps with the same app ID can access the same room.
- **countryCode**: (optional) The type is string[2]. For example, CN indicates the Chinese mainland, US indicates the United States, and HK indicates Hong Kong (China). For details about **countryCode** values, see **GRS Country Codes**.

NOTE

For details about how to obtain the values of **domain** and **appId**, **submit a service ticket**.

Step 3 Join a room. For details, see **join**.

```
let option = { userId: userId, userName: userName, signature: signature, ctime: ctime, role: role }
async joinRoom() {
  try{
    await client.join(roomId, option)
    console.log('join room success')
  } catch(error){
    console.log('join room fail',error)
  }
}
```

- **userId**: (mandatory) unique ID of a local user. The type is string[64].
- **userName**: (optional) user nickname encoded in UTF-8 format. The type is string[256].
- **signature**: (mandatory) authentication signature. You need to obtain it from the remote server. The type is string[512].

> **NOTE**
>
> You need to deploy the remote server. For details, see **Access Authentication**.

- **ctime**: (mandatory) UTC timestamp in seconds. The type is string.
- **role**: (mandatory) user role, which can be used to identify the media direction. The type is number. The options are as follows:
  - **0**: joiner (publish and watch media).
  - **2**: player (watch but not publish media).
- **roomId**: (mandatory) room ID. The type is string[64].

When the user joins the room, the peer client will receive the **peer-join** event.

Step 4 Create and publish a local stream. For details, see **createStream**, **initialize**, **addResolution**, **publish**, and **play**.

```
let stream = HRTC.createStream({ audio:true,microphoneId:xxx,video:true,cameraId:xxx })


stream.initialize().then(() => {
  stream.addResolution('90p_1')  // Optional. If you enable dual streams, you can add a video of another resolution.

  stream.play(elementId,{muted:true})   // Play the local stream.
  client.publish(stream)
})
```

- **audio**: (optional) whether to collect audio from microphones. The type is Boolean. The default value is **false**.
- **video**: (optional) whether to collect video from cameras. The type is Boolean. The default value is **false**.
- **microphoneId**: (optional) ID of the microphone for audio collection. This parameter is valid only when **audio** is set to **true**. The type is string. If this parameter is not passed, the system automatically uses the default value.
- **cameraId**: (optional) ID of the camera for video collection. This parameter is valid only when **video** is set to **true**. The type is string. If this parameter is not passed, the system automatically uses the default value.

Step 5 When receiving the **stream-added** event notification from the server, subscribe to the remote media. For details, see **stream-added**, **subscribe**, and **getStreamInfo**.

```
client.on('stream-added', (event) => {
  const stream = event.stream
  client.subscribe(stream,{ video:true, audio:true })
})
```

> **NOTE**
>
> ```
> Dual-stream scenario:
> client.on('stream-added', (event) => {
>   const stream = event.stream
>   const streamInfo = stream.getStreamInfo()  // Obtain information such as the stream resolution.
>   const resolutionIds = streamInfo.videoProfiles.map((profile) => profile.resolutionId) // The app selects the resolution based on the service scenario.
>   client.subscribe(stream,{video:true, audio:true, resolutionIds:resolutionIds})  // Subscribe to audio and video of the selected resolution.
> })
> ```

When the subscription is complete, the local client will receive the **stream-subscribed** event notification. Set a peer screen to play the audio and video of the peer client. For details, see **stream-subscribed** and **play**.


```
client.on('stream-subscribed', (event) => {
  const stream = event.stream
  stream.play(elementId, { objectFit: 'contain', muted: true, resolutionId: resolutionId })
})
```

- **elementId**: ID of an HTML tag.
- Play parameters:
  - **objectFit**: (optional) The type is string. The value can be **contain**, **cover**, or **fill**. Default value: **cover** for video; **contain** for presentation.
  - **muted**: (optional) The type is Boolean. **true** indicates muted; **false** indicates unmuted. The default value is **false**.
  - **resolutionId**: (optional) resolution of the video to be played. The type is string. By default, the video with the highest resolution is played.

If the user does not need to view the video of the peer client, unsubscribe from the audio and video of the peer client. For details, see **unsubscribe**.

```
client.unsubscribe(stream)
```

Step 6 When the remote client leaves the room, the local client will receive the **peer-leave** event. You can clear the resources of the remote client. For details, see **peer-leave**.

```
client.on('peer-leave', (event) => {
  // just do something...
})
```

event.userId: peer user ID, which is obtained by listening to the **peer-leave** event.

When the remote client leaves the room, the local client will receive the **stream-removed** event notification. You can close the video screen in the event processing function. For details, see **stream-removed**.

```
client.on('stream-removed', (event) => {
  event.stream.close()
})
```

The **close** function is called using the **stream** object. This function will remove the video tag element created using **play** and disable the camera and microphone.

Step 7 Leave the room. For details, see **leave**.

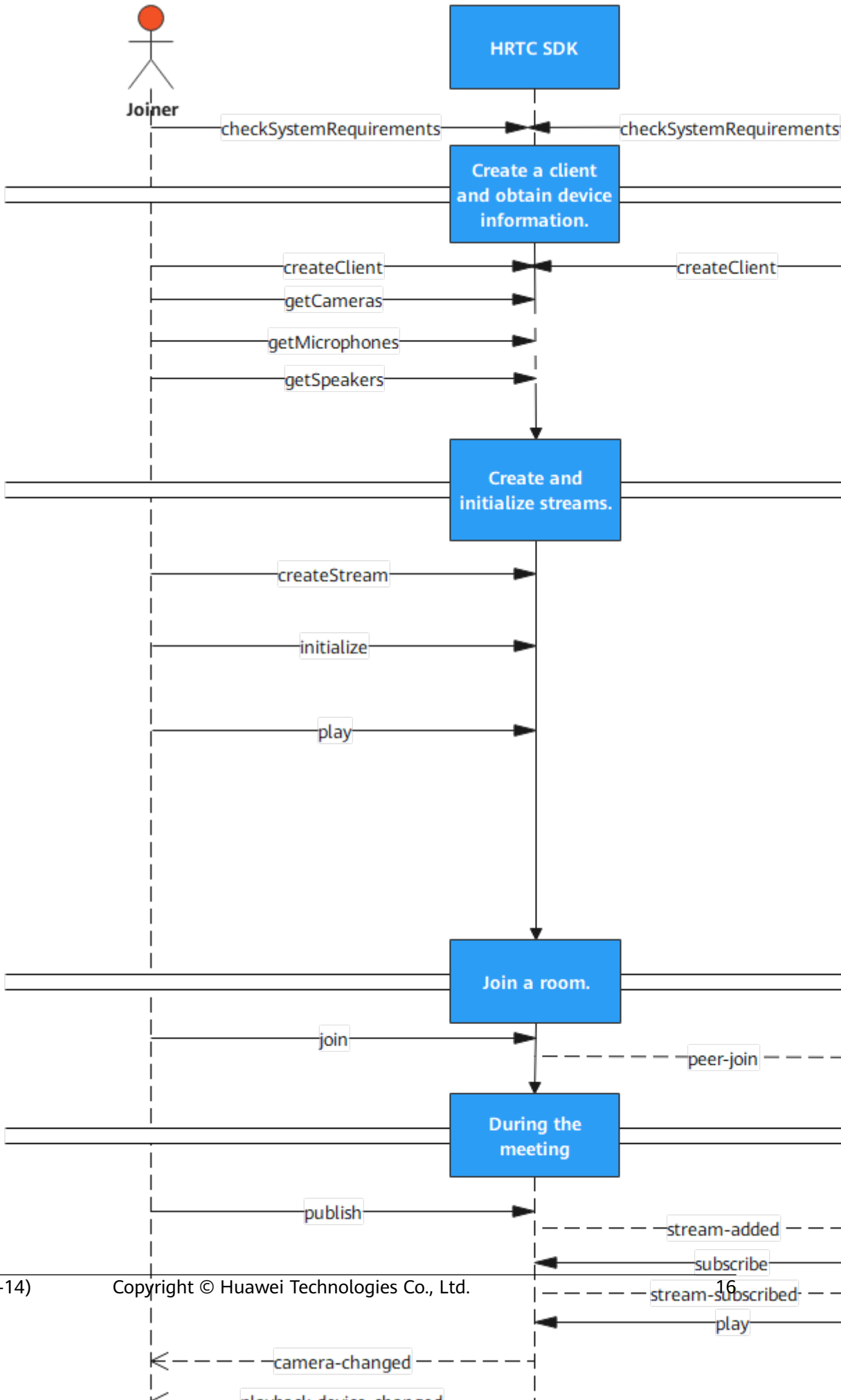
```
client.leave()
```

Call this API to leave the room when the voice or video call ends.

The process of a general voice and video call is completed.

----End

3.4 Basic Usage Logic



1. Create a project. After the SDK is imported, create a client and obtain the local audio and video device information.
2. Create and initialize local streams.
3. After a user joins a room, the system notifies other users in the room through callback. After receiving the callback message, other users can perform other operations such as subscribing to or unsubscribing from audio and video streams.
4. Configure local recording or playback devices during the meeting.
5. After a user leaves a room, other users in the room receive a callback message indicating that the user leaves the room. Resources related to the user need to be destroyed.

NOTE

In the **sequence diagram**, click the name of an API to view how it is used.

3.5 API Reference

3.5.1 Main Entry (HRTC)

This section describes the HRTC APIs of the Web SDK.

Table 3-3 HRTC APIs

API	Description
checkSystemRequirements	Checks whether the browser is compatible with the SparkRTC Web SDK. **NOTICE** If the Web SDK version is between 2.0.2 and 2.0.9.300, you need to update it to 2.0.9.301 or later before October 29, 2023. Otherwise, the **checkSystemRequirements** API will be unavailable. Users of WeChat browsers need to update the SDK version.
VERSION	Obtains the SparkRTC Web SDK version.
getDevices	Obtains the list of media input and output devices.
getCameras	Obtains the camera list.
getMicrophones	Obtains the microphone list.
getSpeakers	Obtains the speaker list.
isScreenShareSupported	Checks whether screen sharing is supported.
createClient	Creates a client object for real-time audio and video calls. One Client object indicates one room.

API	Description
createStream	Creates a local stream object. Streams are classified into video and presentation. Video refers to streams collected by cameras and microphones. Presentation refers to streams collected from shared screens.
setLogLevel	Sets the log level.

checkSystemRequirements

NOTICE

If the Web SDK version is between 2.0.2 and 2.0.9.300, you need to update it to 2.0.9.301 or later before October 29, 2023. Otherwise, the **checkSystemRequirements** API will be unavailable. Users of WeChat browsers need to update the SDK version.

```
(static) checkSystemRequirements(strictCheckBrowser: boolean): Promise<boolean>
```

[Function Description]

Checks whether the browser is compatible with the SparkRTC Web SDK.

[Request Parameters]

strictCheckBrowser: (optional) Boolean type. Default value: **true**. The value **true** indicates that the mobile browser in the whitelist configured and managed by Huawei is used. The value **false** indicates that the whitelist is not used. This parameter is added in version 2.0.2.

[Response Parameters]

Promise<boolean>: A Promise object is returned. The value **true** indicates that the browser is compatible with the SparkRTC Web SDK. If the browser is incompatible with the SparkRTC Web SDK, the corresponding error is returned.

VERSION

```
VERSION
```

[Function Description]

Obtains the SparkRTC Web SDK version.

[Request Parameters]

None

[Response Parameters]

string: current SDK version number.

getDevices

```
(static) getDevices(): Promise<MediaDeviceInfo[]>
```

[Function Description]

Obtains the list of media input and output devices. The **label** and **deviceId** parameters may be empty if the browser is not allowed to access cameras or microphones. It is recommended that you call this API after the browser is allowed to access the devices.

For details about how to grant the permission for accessing cameras or microphones to a browser, see **Granting the Permissions of Accessing Cameras or Microphones to a Browser**.

[Request Parameters]

None

[Response Parameters]

Promise<MediaDeviceInfo[]>: list of media input and output devices. **MediaDeviceInfo** is a basic web API.

getCameras

```
(static) getCameras(): Promise<MediaDeviceInfo[]>
```

[Function Description]

Obtains the camera list. The **label** and **deviceId** parameters may be empty if the browser is not allowed to access cameras. It is recommended that you call this API after the browser is allowed to access the devices.

For details about how to grant the permission for accessing cameras or microphones to a browser, see **Granting the Permissions of Accessing Cameras or Microphones to a Browser**.

[Request Parameters]

None

[Response Parameters]

Promise<MediaDeviceInfo[]>: camera list. **MediaDeviceInfo** is a basic web API.

getMicrophones

```
(static) getMicrophones(): Promise<MediaDeviceInfo[]>
```

[Function Description]

Obtains the microphone list. The **label** and **deviceId** parameters may be empty if the browser is not allowed to access microphones. It is recommended that you call this API after the browser is allowed to access the devices.

For details about how to grant the permission for accessing cameras or microphones to a browser, see **Granting the Permissions of Accessing Cameras or Microphones to a Browser**.

[Request Parameters]

None

[Response Parameters]

Promise<MediaDeviceInfo[]>: microphone list. **MediaDeviceInfo** is a basic web API.

getSpeakers

```
(static) getSpeakers(): Promise<MediaDeviceInfo[]>
```

[Function Description]

Obtains the speaker list.

[Request Parameters]

None

[Response Parameters]

Promise<MediaDeviceInfo[]>: speaker list. **MediaDeviceInfo** is a basic web API.

isScreenShareSupported

```
(static) isScreenShareSupported(): boolean
```

[Function Description]

Checks whether screen sharing is supported.

[Request Parameters]

None

[Response Parameters]

boolean: **true** indicates that the function is supported, and **false** indicates that the function is not supported.

createClient

```
(static)createClient(config: ClientConfig): Client
```

[Function Description]

Creates a client object for real-time audio and video calls. A client object can be added to only one room. You can create multiple client objects and add them to multiple rooms respectively.

[Request Parameters]

config: (mandatory) client object configuration. The type is ClientConfig.

ClientConfig is defined as: {

- **appId**: (mandatory) app ID. The type is string[128]. Only apps with the same ID can enter the same room for interaction. For details about how to obtain the value of **appId**, **submit a service ticket**.
- **domain**: (optional) domain name of the server. The type is string[128]. The value must be the same as the valid enterprise domain name registered with the SparkRTC platform. This parameter is mandatory in SDK 1.0+ and is optional in SDK 2.0+.

- **countryCode**: (optional) country or region code. The type is string[2]. The value must be an alpha-2 code specified in the **ISO 3166-1** standard. This parameter indicates the country or region code of the service access point of the SDK. For example, **CN** indicates the Chinese mainland, **US** indicates the United States, and **HK** indicates Hong Kong, China. For details about how to configure **countryCode**, see **GRS Country Codes**. This parameter was added in version 2.0.3 and was mandatory. It has become an optional parameter in version 2.0.7 or later.

}

[Response Parameters]

Client: client object.

createStream

```
(static) createStream(config: StreamConfig): Stream
```

[Function Description]

Creates a local stream object.

[Request Parameters]

config: (mandatory) parameters for creating a stream. The type is StreamConfig.

StreamConfig is defined as: {

- **screen**: (optional) The type is Boolean. If the value is **true**, the stream object collects presentation audio and video. Presentation refers to screen sharing streams. The default value is **false**, indicating that the stream object collects audio and video collected by microphones and cameras.
- **video**: (optional) whether to collect video from cameras. The type is Boolean. The default value is **false**.
- **audio**: (optional) whether to collect audio from microphones. The type is Boolean. The default value is **false**. This parameter is valid only when **screen** is set to **false**.
- **microphoneId**: (optional) ID of the microphone for audio collection. This parameter is valid only when **audio** is set to **true**. The type is string. If this parameter is not passed, the system automatically uses the default value.
- **cameraId**: (optional) ID of the camera for video collection. This parameter is valid only when **video** is set to **true**. The type is string. If this parameter is not passed, the system automatically uses the default value.
- **facingMode**: (optional) This parameter is valid only when **video** is set to **true**. **user** indicates the front-facing camera; **environment** indicates the rear-facing camera. The type is string.
- **screenAudio**: (optional) whether the screen sharing background sound is included. The type is Boolean. The default value is **false**. This function applies only to Chrome 74 and later versions on Windows. This parameter is added in version 1.4.0.
- **audioSource**: (optional) input audio track object. The type is MediaStreamTrack. Specify a sound track. **MediaStreamTrack** is a basic web API.

- **videoSource**: (optional) input video track object. The type is MediaStreamTrack. Specify a video track. **MediaStreamTrack** is a basic web API.
- **mirror**: (optional) whether the local video collected by cameras is mirrored. The type is Boolean. The default value is **false**.
- **userId**: (optional) ID of the user to which the stream belongs. The type is string.

}

[Response Parameters]

Stream: stream object.

CAUTION

- There are two methods of capturing video streams:
  - Configure **audioSource** and **videoSource** to capture audio and video, respectively. This method does not support high-quality and low-quality streams.
  - Configure **audio/microphoneId** and **video/cameraId/facingMode** to capture audio and video, respectively.
- If no audio or video source is specified, the created stream object does not include audio or video streams and cannot be played.
- A stream can be either a video stream or presentation stream.
- If screen sharing background audio is required, set **screen** and **screenAudio** to **true**. This parameter takes effect only in 1.4.0 and later versions.

setLogLevel

```
(static) setLogLevel(level: LogLevel): void
```

[Function Description]

Sets the log output level. Logs of the **info** level are generated by default.

[Request Parameters]

level: (mandatory) log level. The type is LogLevel.

LogLevel indicates the log level. The enumerated values are as follows:

- **none**: disabling SDK log print. The type is string.
- **error**: enabling the SDK error log level. The type is string.
- **warn**: enabling the SDK warning log level. The type is string.
- **info**: enabling the SDK information log level. The type is string.
- **debug**: enabling the SDK debug log level. The type is string.

[Response Parameters]

None

3.5.2 Client Object (Client)

This section describes the client APIs of the Web SDK.

Table 3-4 Client APIs

API	Description
join	Joins a specific room for an audio or video call.
leave	Leaves a room after a call ends.
publish	Publishes a local stream after a user joins a room.
unpublish	Cancels the publication of a local stream.
subscribe	Subscribes to a remote audio and video media stream.
unsubscribe	Unsubscribes from a remote audio and video media stream.
batchSubscribe	Subscribes to video media streams of remote users in batches.
switchRole	Switches the role of a user.
on	Registers the callback for a client object event.
off	Deregisters the callback for a client object event.
getConnectionState	Obtains the client connection state.
getTransportStats	Obtains the network transmission statistics.
getLocalAudioStats	Obtains local audio statistics.
getLocalVideoStats	Obtains local video statistics.
getRemoteAudioStats	Obtains remote audio statistics.
getRemoteVideoStats	Obtains remote video statistics.
enableTopThreeAudioMode	Sets whether to enable the top-N-audio mode.
setVolume4TopThree	Sets the volume in the top-N-audio mode. This API is added in version 1.4.0.

API	Description
muteAudio4TopThree	Enables or disables all audio tracks in the top-N-audio mode. This API is added in version 1.4.0.
enableStreamStateDetection	Enables or disables video stream state detection. This API is added in version 1.4.0.
changeUserName	Changes the nickname of a user. This API is added in version 1.5.0.
setProxyServer	Configures the proxy reverse server deployed in the enterprise. This API is added in version 2.0.3.
setTurnServer	Configures a proxy server. This API is added in version 2.0.3.
enableRtcStats	Configures whether to enable the RTC audio and video stream statistics event. This API is added in version 2.0.3.
setNetworkBandwidth	Configures the maximum media bandwidth. This API is added in version 2.0.5.
renewSignature	Updates a signature. This API is added in version 2.0.8.

join

```
async join(roomId: string, options: JoinConfig): Promise<void>
```

[Function Description]

Joins a room for an audio or video call.

[Request Parameters]

- **roomId**: unique ID of a room. (Mandatory) string[64] type.
- **options**: configuration of joining a room. (Mandatory) JoinConfig type.
  JoinConfig is defined as: {
  - **userId**: user ID, which must be unique in an application. (Mandatory) string[64] type.
  - **userName**: username. (Optional) string[256] type.
  - **signature**: signature. For details about how to generate a signature, see Access Authentication. (Mandatory) string[512] type.
  - **ctime**: (mandatory) authentication signature UTC timestamp, in seconds. The type is string.
  - **role**: (mandatory) user role, which can be used to identify the media direction. The type is number. The enumerated values of **role** are as follows:

- **0**: joiner, who can send and receive audio and video.
- **2**: player, who only receives others' audio and video but does not send their own audio and video.

[Response Parameters]

Promise<void>: returns a **Promise** object.

> ⚠ CAUTION
>
> - The value of **roomId** can contain letters, digits, hyphens (-), and underscores (_).
> - The value of **userId** can contain letters, digits, hyphens (-), and underscores (_).

leave

```
async leave(): Promise<void>
```

[Function Description]

Leaves a room after a call ends.

[Request Parameters]

None

[Response Parameters]

Promise<void>: returns a **Promise** object.

publish

```
async publish(stream: Stream, option?: PublishOption): Promise<void>
```

[Function Description]

Publishes the local stream after a user joins a room and create a local stream. Only **joiner** can publish local streams.

[Request Parameters]

- **stream**: (mandatory) local stream object. The type is Stream.
- **option**: (optional) whether to proactively publish video streams. The type is PublishOption. If this parameter is not passed, video streams are not proactively published.

PublishOption is defined as:

{

autoPushVideo: (optional) whether to proactively publish video streams. The type is Boolean. The default value is **false**.

}

[Response Parameters]

Promise<void>: returns a **Promise** object.

unpublish

```
async unpublish(stream: Stream): Promise<void>
```

[Function Description]

Cancels the publication of a local stream.

[Request Parameters]

stream: (mandatory) local stream object. The type is Stream.

[Response Parameters]

Promise<void>: returns a **Promise** object.

subscribe

```
async subscribe(stream: RemoteStream, option?: SubscribeOption): Promise<void>
```

[Function Description]

Subscribes to a remote audio and video media stream. When a user subscribes to a remote media stream, the user will receive the Client.on('**stream-subscribed**') event notification. Then, the user can play the stream.

[Request Parameters]

- **stream**: (mandatory) remote stream object, which can be obtained from the **stream-added** event. The type is RemoteStream.
- **option**: (optional) video or audio subscription option. If this parameter is not passed, the audio and video with the highest resolution are subscribed. The type is SubscribeOption.

SubscribeOption is defined as:

{

- **video**: (optional) whether to subscribe to video. The type is Boolean. The default value is **false**.
- **audio**: (optional) whether to subscribe to audio. The type is Boolean. The default value is **false**.
- **resolutionIds**: (optional) resolution IDs of the videos to be subscribed to. The type is string[]. This parameter is valid only when **video** is set to **true**. The value can be obtained by calling **getStreamInfo**. If **resolutionIds** is not passed, the video with the highest resolution is subscribed by default. This parameter is added in version 1.8.0.

}

[Response Parameters]

Promise<void>: returns a **Promise** object.

CAUTION

The **video** and **audio** parameters in SubscribeOption cannot be set to **false** at the same time.

unsubscribe

```
async unsubscribe(stream: Stream,option?: SubscribeOption): Promise<void>
```

[Function Description]

Unsubscribes from a remote audio and video media stream.

[Request Parameters]

- **stream**: (mandatory) remote stream object, which can be obtained from the **stream-added** event. The type is RemoteStream.
- **option**: (optional) unsubscription option. If this parameter is not passed, audio and videos of all resolutions will be unsubscribed from. The type is SubscribeOption.

  SubscribeOption is defined as: {

  - **video**: (optional) whether to unsubscribe from video. The type is Boolean. The default value is **false**.
  - **audio**: (optional) whether to unsubscribe from audio. The type is Boolean. The default value is **false**.
  - **resolutionIds**: (optional) resolution IDs of the videos to be unsubscribed from. The type is string[]. This parameter is valid only when **video** is set to **true**. The value can be obtained by calling **getStreamInfo**. If **resolutionIds** is not passed, videos of all resolutions will be unsubscribed from.

  }

[Response Parameters]

Promise<void>: returns a **Promise** object.

batchSubscribe

```
async batchSubscribe(subscribeInfos: SubscribeParam[]): Promise<void>
```

[Function Description]

Subscribes to videos of multiple users in batches. When a user subscribes to a remote media stream, the user will receive the Client.on('**stream-subscribed**') event notification. Then, users can play the subscribed stream.

[Request Parameters]

subscribeInfos: (mandatory) information of all remote users' video to be subscribed to. The type is SubscribeParam[].

SubscribeParam is defined as: {

- **userId**: (mandatory) ID of the user to be subscribed to. The type is string.
- **resolutionIds**: (optional) resolution of user videos to be subscribed to. The type is string[]. If **resolutionIds** is not passed, the video with the highest resolution is subscribed by default.
- **minResolution**: (optional) minimum resolution when adaptive video resolution is enabled. The type is ResolutionType. Enumerated values of **ResolutionType** are **FHD**, **HD**, **SD**, and **LD**.

}

[Response Parameters]

Promise<void>: returns a **Promise** object.

CAUTION

- This API is used to subscribe to the video of a remote user but not the audio of the remote user.
- Different from the incremental subscription mode of the **subscribe** API, the input parameters of this API form the list of all users to be subscribed to.

switchRole

```
async switchRole(role: number,authorization: Authorization): Promise<void>
```

[Function Description]

Switches the user role after a user joins a room.

[Request Parameters]

- **role**: (mandatory) The type is number.
  - **0**: joiner, who can send and receive audio and video.
  - **2**: player, who only receives others' audio and video but does not send their own audio and video.
- **authorization**: authentication information. This parameter is mandatory. The type is **Authorization**. This parameter is added in version 2.0.0. **Authorization** is defined as follows:
  - **signature**: (mandatory) authentication signature string. The type is string[512]. For details about how to generate a signature, see **Access Authentication**.
  - **ctime**: (mandatory) authentication signature UTC timestamp, in seconds. The type is string.

[Response Parameters]

Promise<void>: returns a **Promise** object.

on

```
on(event: string, handler: function): void
```

[Function Description]

Registers the callback for a client object event.

[Request Parameters]

- **event**: (mandatory) event name. The type is string. For details, see **ClientEvent**.
- **handler**: (mandatory) event processing method. The type is function.

[Response Parameters]

None

off

```
off(event: string, handler: function): void
```

[Function Description]

Deregisters the callback for a client object event.

[Request Parameters]

- **event**: (mandatory) event name. The type is string. For details, see **ClientEvent**.
- **handler**: (mandatory) event processing method. The type is function.

[Response Parameters]

None

getConnectionState

```
getConnectionState(): ConnectionState
```

[Function Description]

Obtains the client connection state.

[Request Parameters]

None

[Response Parameters]

ConnectionState: state of a WebSocket connection. The type is string. Values can be:

- CONNECTING
- CONNECTED
- RECONNECTING
- DISCONNECTED

getTransportStats

```
getTransportStats(): Promise<TransportStats>
```

[Function Description]

Obtains the network transmission statistics. This API can be called only after **publish** is called. In the dual-stream scenario, statistics of the video with the highest resolution collected by cameras are obtained by default.

[Request Parameters]

None

[Response Parameters]

The type of response parameters is TransportStats. Currently, TransportStats supports the following attributes: {

- **bytesSent**: number of sent bytes. The type is number.
- **bytesReceived**: number of received bytes. The type is number.

- **sendBitrate**: output bit rate, in kbit/s. The type is number.
- **recvBitrate**: input bit rate, in kbit/s. The type is number.
- **rtt**: round-trip time (RTT) from the SDK to the edge server, in milliseconds. The type is number.

  }

getLocalAudioStats

```
getLocalAudioStats(): Promise<Map<string, LocalAudioStats>>
```

[Function Description]

Obtains statistics of local audio that has been sent.

[Request Parameters]

None

[Response Parameters]

- **string**: user ID.
- **LocalAudioStats**: local audio statistics, including the following attributes: {
  - **bytesSent**: number of sent bytes. The type is number.
  - **packetsSent**: number of sent packets. The type is number.

    }

getLocalVideoStats

```
getLocalVideoStats(): Promise<Map<string, AllLocalVideoStats>>
```

[Function Description]

Obtains statistics of local video that has been sent.

[Request Parameters]

None

[Response Parameters]

- **string**: user ID.
- **AllLocalVideoStats**: local video statistics, including the following attributes: {
  - **mainStream**: local video statistics. The type is LocalVideoStats[].
  - **subStream**: local presentation statistics. The type is LocalVideoStats.

    }

  LocalVideoStats contains the following attributes: {
  - **bytesSent**: number of sent bytes. The type is number.
  - **packetsSent**: number of sent packets. The type is number.
  - **framesEncoded**: number of encoded frames. The type is number.
  - **framesSent**: number of sent frames. The type is number.
  - **frameWidth**: video width. The type is number.
  - **frameHeight**: video height. The type is number.

    }

> **CAUTION**
>
> getLocalVideoStats and getLocalAudioStats APIs can be called only after the local stream is published and subscribed to by a remote end.

getRemoteAudioStats

```
getRemoteAudioStats(): Promise<Map<string, RemoteAudioStats>>
```

[Function Description]

Obtains remote audio statistics.

[Request Parameters]

None

[Response Parameters]

- **string**: user ID.
- **RemoteAudioStats**: remote audio statistics indicators, which can be:
  - **bytesReceived**: number of received bytes. The type is number.
  - **packetsReceived**: number of received packets. The type is number.
  - **packetsLost**: number of lost packets. The type is number.

getRemoteVideoStats

```
getRemoteVideoStats(): Promise<Map<string, AllRemoteVideoStats>>
```

[Function Description]

Obtains remote video statistics.

[Request Parameters]

None

[Response Parameters]

- **string**: user ID.
- **AllRemoteVideoStats**: remote video statistics indicators, which can be: {
  - **mainStream**: remote video statistics, including statistics about high-quality and low-quality streams. The type is RemoteVideoStats[].
  - **subStream**: remote presentation statistics. The type is RemoteVideoStats.

  }

  RemoteVideoStats contains the following attributes: {
  - **bytesReceived**: number of received bytes. The type is number.
  - **packetsReceived**: number of received packets. The type is number.
  - **packetsLost**: number of lost packets. The type is number.
  - **framesDecoded**: number of decoded frames. The type is number.
  - **frameWidth**: video width. The type is number.
  - **frameHeight**: video height. The type is number.

  }

> **CAUTION**
>
> The resolution data of the remote video cannot be obtained on the Firefox browser.

enableTopThreeAudioMode

```
enableTopThreeAudioMode(enable: boolean): boolean
```

[Function Description]

Specifies whether to enable the top-N-audio mode (top three loudest participants) before joining a meeting.

[Request Parameters]

enable: (mandatory) The type is Boolean. The value **true** indicates that the top-N-audio mode is enabled, and the value **false** indicates that the top-N-audio mode is disabled. The default value is **false**.

[Response Parameters]

The type is Boolean. **true** indicates successful; **false** indicates failed.

> **CAUTION**
>
> This API needs to be called before users join the room. This API is added in version 1.2.0.

setVolume4TopThree

```
setVolume4TopThree(volume: number): void
```

[Function Description]

Configures the audio volume after the top-N-audio mode (top three loudest participants) is enabled.

[Request Parameters]

volume: (mandatory) audio volume. The type is number. The value range is [0, 100].

[Response Parameters]

None

> **CAUTION**
>
> This API needs to be called after **enableTopThreeAudioMode** is called. This API is added in version 1.4.0.

muteAudio4TopThree

```
muteAudio4TopThree(enable: boolean): void
```


[Function Description]

Enables or disables the audio track in top-N-audio mode (tho three loudest participants) after the top-N-audio mode is enabled.

[Request Parameters]

enable: (mandatory) whether to disable audio tracks of the top-N-audio mode. The type is Boolean. **true** indicates disabled; **false** indicates enabled. The default value is **false**.

[Response Parameters]

None

> ⚠ CAUTION
>
> This API needs to be called after **enableTopThreeAudioMode** is called. This API is added in version 1.4.0.

enableStreamStateDetection

```
async enableStreamStateDetection(enable: boolean, interval: number): Promise<void>
```

[Function Description]

Enables or disables video stream state detection. After this function is enabled, the system can detect whether any remote users who are subscribed to by the local user in the room have no video stream. If a remote user has no video stream, the local user will receive the **stream-interrupted** event. If the video stream of a remote user is restored, the local user will receive the **stream-recovered** event.

[Request Parameters]

- **enable**: (mandatory) whether to enable video stream state detection. The type is Boolean. **true** indicates enabled; **false** indicates disabled. The default value is **false**.
- **interval**: (mandatory) duration for determining whether there is no video stream, in seconds. The type is number. The value range is [1, 60]. The recommended value is 3s.

[Response Parameters]

Promise<void>: returns a **Promise** object.

> ⚠ CAUTION
>
> This API needs to be called after users join the room. This API is added in version 1.4.0.

changeUserName

```
async changeUserName(userName: string): Promise<boolean>
```

[Function Description]

Modifies the nickname of a user. Other users in the room will receive the **remote-user-name-changed** event notification.

[Request Parameters]

userName: (mandatory) new nickname of the user. The type is string[256].

[Response Parameters]

Promise<boolean>: **Promise** object. **true** indicates that the nickname is modified; **false** indicates that the nickname fails to be modified.

> ⚠ CAUTION
>
> This API needs to be called after users join the room. This API is added in version 1.5.0.

enableRtcStats

```
async enableRtcStats(enable: boolean, interval: number): Promise<void>
```

[Function Description]

Configures whether to enable the RTC audio and video stream statistics event. This API is added in version 2.0.3.

[Request Parameters]

enable: (mandatory) whether to enable the RTC audio and video stream statistics event. The type is Boolean. The value **true** indicates that the event is enabled, and the value **false** indicates that the event is disabled.

interval: (mandatory) interval of generating the RTC audio and video stream statistics event, in milliseconds. The type is number. This parameter is valid only when **enable** is set to **true**.

[Response Parameters]

Promise<void>: returns a **Promise** object.

setProxyServer

```
setProxyServer(server: string): void
```

[Function Description]

Configures the signaling proxy server. This API is used to deploy a reverse proxy server (such as Nginx) in an enterprise. It is newly added in version 2.0.3.

[Request Parameters]

servers: (mandatory) list of reverse proxy servers. The type is string. Proxy server format: http://ip:port / https://domain:port.

[Response Parameters]

None

> **CAUTION**
>
> setProxyServer and setTurnServer APIs must be called before calling the join API.

setTurnServer

```
setTurnServer(turnServerConfig: TurnServerConfig): void
```

[Function Description]

Configures a proxy server. This API is used to deploy a reverse proxy server (such as Nginx) in an enterprise. It is newly added in version 2.0.3.

[Request Parameters]

turnServerConfig: proxy server configuration. (Mandatory) TurnServerConfig type.

TurnServerConfig is defined as follows: {

- **turnServers**: reverse proxy server address. (Mandatory) string[] type.
- **udpPort**: UDP port. (Optional) Number type.
- **userName**: reverse proxy server username. (Optional) String type.
- **credential**: reverse proxy server password. (Optional) String type.

  }

[Response Parameters]

None

setNetworkBandwidth

```
setNetworkBandwidth(bandwidthParam: NetworkBandwidth): void
```

[Function Description]

Configures the maximum media bandwidth. This API is added in version 2.0.5.

[Request Parameters]

NetworkBandwidth is defined as: {

maxBandwidth: (mandatory) the maximum total media bandwidth. The value ranges from 3072 to 51200 in kbit/s. The type is number.

}

[Response Parameters]

None

renewSignature

```
renewSignature(ctime: string, signature: string): boolean
```

[Function Description]

Updates a signature.

[Request Parameters]

- **ctime**: time when the signature authentication expires. The value is the current UTC time (UNIX timestamp) of the system plus the authentication expiration time (recommended: 2 hours; maximum: < 12 hours). The unit is second. (Mandatory) string type.
- **signature**: signature. For details about how to generate a signature, see **Access Authentication**. (Mandatory) The value is of the string[512] type.

[Response Parameters]

boolean: A Boolean value is returned, indicating whether the signature has been updated.

CAUTION

This API is added in version 2.0.8.

3.5.3 Client Event Notification (ClientEvent)

This section describes the **ClientEvent** events of the Web SDK.

Table 3-5 ClientEvent events

API	Description
peer-join	A remote user joins a room.
peer-leave	A remote user leaves a room.
stream-added	A remote stream is added.
stream-removed	A remote stream is deleted.
stream-updated	A remote stream is updated.
stream-subscribed	A remote stream is successfully subscribed to.
client-banned	A user is banned from a room.
Error	An error occurs on the client.
connection-state-changed	The client connection status changes.
mute-audio	The audio of a remote stream is disabled.
unmute-audio	The audio of a remote stream is enabled.
mute-video	The video of a remote stream is disabled.
unmute-video	The video of a remote stream is enabled.
log-upload-result	Log upload result.
signature-expired	A signature has expired. This event is added in version 2.0.8.

API	Description
camera-changed	A camera is changed.
recording-device-changed	A recording device is changed.
playback-device-changed	A playback device is changed.
network-quality	Network uplink and downlink quality.
stream-interrupted	A remote stream is interrupted. This event is added in version 1.4.0.
stream-recovered	A remote stream is recovered. This event is added in version 1.4.0.
volume-indicator	A user speaks loudest in the top-N-audio mode (top three loudest participants). This event is added in version 1.5.0.
remote-user-name-changed	The nickname of a remote user is changed. This event is added in version 1.5.0.
rtc-stats	Audio and video stream data statistics event. This event is added in version 2.0.3.

CAUTION

Registration listening must be canceled when the service ends. Otherwise, memory leakage may occur when there are a certain number of registration listening events.

peer-join

[Event Description]

This event is triggered when a remote user joins a room.

[Callback Parameters]

peerJoinEvent: (mandatory) user information. The type is PeerJoin.

PeerJoin is defined as: {

- **userId**: (mandatory) user ID. The type is string[64].
- **userName**: (optional) user nickname. The type is string[256].

}

peer-leave

[Event Description]

This event is triggered when a remote user leaves a room.

[Callback Parameters]

peerLeaveEvent: (mandatory) user leaving information. The type is PeerLeaveInfo.

PeerLeaveInfo is defined as: {

- **userId**: (mandatory) user ID. The type is string[64].
- **userName**: (optional) user nickname. The type is string[256].
- **reason**: (optional) The type is HRTCLeaveReason.

}

HRTCLeaveReason is defined as: {

- **code**: enumeration of reasons for leaving the room. The type is number. **Table 3-6** lists the values.
- **msg**: reason description. The type is string.

}

Table 3-6 Reasons for leaving a room

Value	Description
0	The user exits the room.
1	A server exception occurs.
2	The SFU service is faulty.
3	The service is unavailable (503).
4	An internal error occurs.
5	The user is banned from the room.
6	The signature has expired.
7	Reconnection times out.
8	Network check. No error code is required for the UI.
9	The user is removed.
10	The room is dismissed.

stream-added

[Event Description]

This event is triggered when a remote user successfully sends a stream.

[Callback Parameters]

stream: (mandatory) remote stream object. The type is RemoteStream.

stream-removed

[Event Description]

This event is triggered when a remote user stops sending streams or exits the room.

[Callback Parameters]

stream: (mandatory) remote stream object. The type is RemoteStream.

stream-updated

[Event Description]

This event is triggered when the stream of a remote user changes. For example, audio and video tracks are added or removed, or the video track specifications change.

[Callback Parameters]

stream: (mandatory) remote stream object. The type is RemoteStream.

stream-subscribed

[Event Description]

This event is triggered when a remote stream is successfully subscribed to.

[Callback Parameters]

stream: (mandatory) remote stream object. The type is RemoteStream.

client-banned

[Event Description]

This event is triggered when a user is banned. When a user joins the same room on another client using the same user ID, the client on which the user is banned receives a notification.

[Callback Parameters]

clientBannedEvent: (mandatory) The type is ClientBanInfo.

ClientBanInfo is defined as: {

- **userId**: (mandatory) ID of a banned user. The type is string[64].
- **reason**: (mandatory) reason description. The type is string.

}

Error

[Event Description]

This event is triggered when an unrecoverable client error occurs.

[Callback Parameters]

errorInfo: (mandatory) error information. The type is ErrorInfo.

ErrorInfo is defined as: {

- **errorCode**: (mandatory) error code. The type is string.
- **errorMsg**: (mandatory) error description. The type is string.

}

connection-state-changed

[Event Description]

This event is triggered when the client connection status changes.

[Callback Parameters]

- ConnectionStateInfoEvent: {
  - **prevState**: (mandatory) previous state. The type is ConnectionState.
  - **curState**: (mandatory) current state. The type is ConnectionState.

}

The options of ConnectionState are as follows:

- CONNECTING
- CONNECTED
- RECONNECTING
- DISCONNECTED

mute-audio

[Event Description]

This event is triggered when a remote user mutes the audio.

[Callback Parameters]

mediaStatus: (mandatory) The type is MediaStatusNotifyInfo.

MediaStatusNotifyInfo has the following attributes.

- **roomId**: (mandatory) string[64] type.
- **userId**: (mandatory) string[64] type.
- **status**: (mandatory) MediaStatusAction type. The values of MediaStatusAction are as follows:
  - **1**: Media assets are available.
  - **2**: Media assets are unavailable.
- **reason**: (mandatory) MediaStatusReason type. The values of MediaStatusReason are as follows:
  - **0**: The media is offline.
  - **1**: The media is muted.
  - **2**: The media is not muted.

unmute-audio

[Event Description]

This event is triggered when a remote user unmutes the audio.

[Callback Parameters]

mediaStatus: (mandatory) The type is **MediaStatusNotifyInfo**.

mute-video

[Event Description]

This event is triggered when a remote user disables the video.

[Callback Parameters]

mediaStatus: (mandatory) The type is **MediaStatusNotifyInfo**.

unmute-video

[Event Description]

This event is triggered when a remote user enables the video.

[Callback Parameters]

mediaStatus: (mandatory) The type is **MediaStatusNotifyInfo**.

log-upload-result

[Event Description]

Log upload result.

[Callback Parameters]

status: (mandatory) log upload result. The type is number. **200** indicates that the log is successfully uploaded. Other values indicate that the log fails to be uploaded.

signature-expired

[Event Description]

This event is triggered when a signature has expired.

[Callback Parameters]

errorInfo: error information. (Mandatory) ErrorInfo type.

ErrorInfo is defined as: {

- **errorCode**: error code. (Mandatory) string type.
- **errorMsg**: (Mandatory) string type. The values are shown in the following example.

}

The values are shown in the following example.

- Signature expired: {
  errorCode: '90100030'
  errorMsg: 'signature expired'
  }
- Invalid signature: {
  errorCode: '90100031'
  errorMsg: 'signature invalid'
  }

CAUTION

After a signature expiration event is detected, the error code can be used to determine whether the signature is invalid or expired. After the signature expires, the **renewSignature** API can be called to update the signature.

camera-changed

[Event Description]

This event is triggered when the local camera is connected or disconnected.

[Callback Parameters]

DeviceChangedEvent: (mandatory) device change details. The type is DeviceChangedInfo.

DeviceChangedInfo is defined as: {

- **deviceId**: (mandatory) device ID. The type is string.
- **state**: (mandatory) The type is DeviceChangeMode. The values are as follows:
  - **ADD**: A device is connected.
  - **REMOVE**: A device is disconnected.

}

CAUTION

After a video capture device is removed and inserted, certain operations need to be performed at the application layer. For example, determine whether to use another video capture device for data collection when a camera is removed, and whether to use the newly inserted camera for data collection.

recording-device-changed

[Event Description]

This event is triggered when a local recording device is changed.

[Callback Parameters]

DeviceChangedEvent: (mandatory) device change details. The type is DeviceChangedInfo. For details about the definition of DeviceChangedInfo, see **camera-changed**.

CAUTION

After an audio capture device is removed and inserted, certain operations need to be performed at the application layer. For example, determine whether to use another audio capture device for data collection when a microphone is removed, and whether to use the newly inserted microphone for data collection.

playback-device-changed

[Event Description]

This event is triggered when a local audio playback device is changed.

[Callback Parameters]

DeviceChangedEvent: (mandatory) device change details. The type is DeviceChangedInfo. For details about the definition of DeviceChangedInfo, see **camera-changed**.

network-quality

[Event Description]

This event is triggered when the network quality changes after a user joins a room to report the uplink and downlink quality of the user's local network.

[Callback Parameters]

NetworkQualityEvent: (mandatory) uplink and downlink network quality details. The type is NetworkQualityInfo.

NetworkQualityInfo is defined as: {

- **uplinkNetworkQuality**: (mandatory) uplink network quality. The type is number. The values are as follows:
  - **0**: unknown.
  - **1**: excellent.
  - **2**: User experience is almost the same as that of value **1**, but the bit rate may be slightly lower.
  - **3**: User experience is unsatisfactory, but communication is smooth.
  - **4**: Communication is barely smooth.
  - **5**: The network quality is very poor, and communication is almost impossible.
  - **6**: The network is disconnected, and communication fails.
- **downlinkNetworkQuality**: (mandatory) downlink network quality. The type is number. The values are as follows:

- **0**: unknown.
- **1**: excellent.
- **2**: User experience is almost the same as that of value **1**, but the bit rate may be slightly lower.
- **3**: User experience is unsatisfactory, but communication is smooth.
- **4**: Communication is barely smooth.
- **5**: The network quality is very poor, and communication is almost impossible.
- **6**: The network is disconnected, and communication fails.

}

stream-interrupted

[Event Description]

This event is triggered when a remote stream is interrupted. The interrupt indicates that no valid video frame is received within the statistical period specified by the **interval** parameter of the **enableStreamStateDetection** API. This event is added in version 1.4.0.

[Callback Parameters]

streamInterruptedEvent: (mandatory) list of remote users who are subscribed to by the local user but have no video stream. The type is UserList[].

UserList is defined as: {

- **userId**: (mandatory) user ID. The type is string.
- **isScreen**: (mandatory) The type is Boolean. **true** indicates the user's presentation stream; **false** indicates the user's video stream captured by cameras.

}

stream-recovered

[Event Description]

This event is triggered when a remote stream is recovered.

[Callback Parameters]

streamRecoveredEvent: (mandatory) list of remote users who are subscribed to by the local user and the video streams of the remote users have been recovered. The type is UserList[]. For details about UserList, see **stream-interrupted**. This event is added in version 1.4.0.

volume-indicator

[Event Description]

This event is triggered when a user speaks loudest in the room in top-N-audio mode.

[Callback Parameters]

userVolumeInfos: (mandatory) The type is UserVolumeInfo[].

UserVolumeInfo is defined as: {

- **user_id**: (mandatory) user ID. The type is string.
- **volume**: (optional) The type is number. Value range: [0,100].

}

CAUTION

This event takes effect only in the top-N-audio mode and is added in version 1.5.0.

remote-user-name-changed

[Event Description]

This event is triggered when the nickname of a remote user is changed. This event is added in version 1.5.0.

[Callback Parameters]

userNameChangedEvent: (mandatory) UserNameInfo type.

UserNameInfo is defined as: {

- **roomId**: (mandatory) room ID. The type is string[64].
- **userId**: (mandatory) user ID. The type is string[64].
- **userName**: (mandatory) new nickname of the user. The type is string[256].

}

rtc-stats

[Event Description]

Audio and video stream data statistics event. This event is added in version 2.0.3.

[Callback Parameters]

rtcStatsInfo: (mandatory) rtcStatsInfo[] type.

rtcStatsInfo is defined as: {

- **userName**: (mandatory) user nickname. The type is string.
- **isRemote**: (mandatory) whether a stream is a remote stream. The type is Boolean. The value **true** indicates a remote stream, and the value **false** indicates a local stream.
- **streamType**: (mandatory) stream type. The type is ContentType. The enumerated values of ContentType are as follows: {
  - **main**: video stream. The type is string.
  - **middle1**: video stream. The type is string. The code streams from **middle1** to **middle4** are in descending order.
  - **middle2**: video stream. The type is string. The code streams from **middle1** to **middle4** are in descending order.

  - **middle3**: video stream. The type is string. The code streams from **middle1** to **middle4** are in descending order.
  - **middle4**: video stream. The type is string. The code streams from **middle1** to **middle4** are in descending order.
  - **slides**: video stream with the minimum resolution. The type is string.
  - **desktop**: presentation stream. The type is string.

  }

- **mediaType**: (mandatory) media type, which can be audio or video. The type is MediaType.
- **bitrate**: (mandatory) bitrate of an audio or video stream, in kbit/s. The type is number.
- **frameRate**: (mandatory) video frame rate, in fps. The type is number.
- **rtt**: (mandatory) round-trip time (RTT) from the SDK to the edge server, in milliseconds. The type is number. Only local streams have the RTT.
- **jitter**: (mandatory) jitter of the audio and video stream. The type is number.
- **pktLossRate**: (mandatory) packet loss rate of the audio and video stream. The type is number.

}

3.5.4 Stream Object (Stream)

This section describes the stream APIs of the Web SDK.

Table 3-7 Stream APIs

API	Description
play	Plays audio and video streams.
stop	Stops playing a video stream.
resume	Resumes audio and video playback.
close	Disables audio and video.
muteAudio	Mutes an audio track.
muteVideo	Disables a video track.
unmuteAudio	Unmutes an audio track.
unmuteVideo	Enables a video track.
getId	Obtains the unique ID of a stream.
getUserId	Obtains the ID of the user to which a stream belongs.
setAudioOutput	Sets an audio output device.
setAudioVolume	Sets audio volume.
getAudioLevel	Obtains the real-time audio volume level.

API	Description
hasAudio	Indicates whether a stream contains an audio track.
hasVideo	Indicates whether a stream contains a video track.
getAudioTrack	Obtains the audio track of a stream.
getVideoTrack	Obtains the video track of a stream.
getType	Obtains the stream type.
on	Registers the callback for a stream object event.
off	Deregisters the callback for a stream object event.
getStreamInfo	Obtains stream information.

play

```
async play(elementId: string, options?: Options): Promise<void>
```

[Function Description]

Plays audio and video streams. This API automatically creates a <audio> or <video> tag and plays audio and video with the specified tag. The tag is added to the div container named **elementId** on the page.

[Request Parameters]

- **elementId**: (mandatory) HTML tag ID. The type is string.
- **options**: (optional) playback option. The type is Options.

  Options is defined as:

  - **objectFit**: (optional) string type. The default value is **contain** for remote sharing streams and **cover** for other streams. The following enumerated values are supported:

    - **contain**: preferentially ensures that all video content is displayed. The video is scaled proportionally until one side of the video window is aligned with the window border. If the video size is inconsistent with the display window size, when the aspect ratio is locked and the video is zoomed in or out to fill the window, a black bar is displayed around the zoomed-in or zoomed-out video.
    - **cover**: preferentially ensures that the window is filled. The video is scaled proportionally until the entire window is filled with video. If the video size is inconsistent with the display window size, the video stream will be cropped or the image will be stretched to fill the display window.
    - **fill**: The window is filled with video. If the aspect ratio of the video does not match the window, the video will be stretched to fit the window.

  - **muted**: (optional) The type is Boolean. **true** indicates muted; **false** indicates unmuted. The default value is **false**.
  - **resolutionId**: (optional) The type is string. In the dual-stream scenario, specify the resolution ID of the video to be played. If the resolution ID is not specified, the video with the highest resolution is selected by default. This parameter is added in version 1.8.0.
  - }

[Response Parameters]

Promise<void>: returns a **Promise** object.

⚠ CAUTION

- Due to the restriction of the browser's automatic playback policy, when **play()** returns an error, the user needs to manually trigger the page control to call the **resume** API to resume the playback.
- To play local streams, you need to set **muted** to **true** (muted) to prevent the played sound from being captured by the microphone.
- On the app, a resolution corresponds to an audio/video playback window. The audio in the stream is common to all resolutions.

---

stop

```
stop(option?: StopOption): void
```

[Function Description]

Stops playing audio and video streams.

[Request Parameters]

option: (optional) option for stopping the playback. The type is StopOption. If this parameter is not passed, the audio of the stream and the videos of all resolutions are stopped.

StopOption is defined as: {

- **audio**: (optional) whether to stop the audio stream. The type is Boolean. The default value is **false**.
- **video**: (optional) whether to stop the video stream. The type is Boolean. The default value is **false**.
- **resolutionIds**: (optional) The type is string[]. It is valid when **video** is set to **true**. It is used to stop the video with a specific resolution ID. If **resolutionIds** is not passed, videos of all resolutions are stopped by default.

}

[Response Parameters]

None

resume

```
async resume(option?: ResumeOption): Promise<void>
```


[Function Description]

Resumes audio and video playback. In this scenario:

- In some browsers, if the div container transferred to **play()** is moved, the audio and video player may enter the PAUSED state. Therefore, this API needs to be called to resume the playback.
- Due to the restriction of the browser's automatic playback policy, when **play()** returns an error, the user needs to manually call this API to resume the playback.

[Request Parameters]

option: (optional) option for resuming the playback. The type is ResumeOption. If this parameter is not passed, the audio of the stream and videos of all resolutions are resumed.

ResumeOption is defined as:

- **audio**: (optional) whether to resume the audio stream. The type is Boolean. The default value is **false**.
- **video**: (optional) whether to resume the video stream. The type is Boolean. The default value is **false**.
- **resolutionIds**: (optional) The type is string. It is valid only when **video** is set to **true**. It is used to resume the video with a specific resolution ID. If **resolutionIds** is passed, videos of all resolutions are resumed by default.

}

[Response Parameters]

Promise<void>: returns a **Promise** object.

close

```
close(option?: CloseOption): void
```

[Function Description]

Closes the audio and video playback. For local streams, this method will also disable audio and video capture and release device resources.

[Request Parameters]

option: (optional) option for disabling the audio and video. The type is CloseOption. If **option** is not set, audio and videos of all resolutions are disabled.

CloseOption is defined as:

- **audio**: (optional) The type is Boolean. It indicates whether to disable the audio stream. The default value is **false**.
- **video**: (optional) The type is Boolean. It indicates whether to disable the video stream. The default value is **false**.
- **resolutionIds**: (optional) The type is string[]. It is valid when **video** is set to **true**. It is used to disable the video with a specific resolution ID. If **resolutionIds** is not passed, videos of all resolutions are disabled by default.

}

[Response Parameters]

None

muteAudio

```
muteAudio(): boolean
```

[Function Description]

Mutes an audio track.

[Request Parameters]

None

[Response Parameters]

The type is Boolean. **true** indicates that the audio track is successfully muted; **false** indicates that the audio track fails to be muted.

muteVideo

```
muteVideo(): boolean
```

[Function Description]

Disables a video track.

[Request Parameters]

None

[Response Parameters]

The type is Boolean. **true** indicates that the video track is successfully disabled; **false** indicates that the video track fails to be disabled.

unmuteAudio

```
unmuteAudio(): boolean
```

[Function Description]

Unmutes an audio track.

[Request Parameters]

None

[Response Parameters]

The type is Boolean. **true** indicates that the audio track is successfully unmuted; **false** indicates that the audio track fails to be unmuted.

unmuteVideo

```
unmuteVideo(): boolean
```

[Function Description]

Enables a video track.

[Request Parameters]

None

[Response Parameters]

The type is Boolean. **true** indicates that the video track is successfully enabled; **false** indicates that the video track fails to be enabled.

getId

```
getId(): string
```

[Function Description]

Obtains the unique ID of a stream. If a local stream is used, a valid ID can be obtained only after the stream is published.

[Request Parameters]

None

[Response Parameters]

Unique ID of a stream. The type is string.

getUserId

```
getUserId(): string
```

[Function Description]

Obtains the ID of the user to which a stream belongs. For a local stream, if this parameter is not set in the input parameter **StreamConfig** of **createStream**, **undefined** is returned.

[Request Parameters]

None

[Response Parameters]

ID of the user to which the stream belongs. The type is string.

setAudioOutput

```
setAudioOutput(deviceId: string): Promise<void>
```

[Function Description]

Sets an audio output device.

[Request Parameters]

deviceId: (mandatory) ID of an audio output device. The type is string.

[Response Parameters]

Promise<void>: returns a **Promise** object.

setAudioVolume

```
setAudioVolume(volume: number): void
```

[Function Description]

Sets audio volume.

[Request Parameters]

volume: (mandatory) volume. The type is number. The value range is [0,100].

[Response Parameters]

None

getAudioLevel

```
getAudioLevel(): number
```

[Function Description]

Obtains the real-time volume level.

[Request Parameters]

None

[Response Parameters]

The type is number. The return value range is (0, 1). Generally, a value greater than 0.1 indicates that the user is speaking.

hasAudio

```
hasAudio(): boolean
```

[Function Description]

Indicates whether a stream contains an audio track.

[Request Parameters]

None

[Response Parameters]

The type is Boolean. **true** indicates contained; **false** indicates not contained.

hasVideo

```
hasVideo(): boolean
```

[Function Description]

Indicates whether a stream contains a video track.

[Request Parameters]

None

[Response Parameters]

The type is Boolean. **true** indicates contained; **false** indicates not contained.

getAudioTrack

```
getAudioTrack(): MediaStreamTrack
```

[Function Description]

Obtains an audio track.

[Request Parameters]

None

[Response Parameters]

The type is MediaStreamTrack. For details about the MediaStreamTrack type, see **MediaStreamTrack**.

getVideoTrack

```
getVideoTrack(resolutionId?:string): MediaStreamTrack
```

[Function Description]

Obtains a video track.

[Request Parameters]

resolutionId: (optional) The type is string. Specify the resolution ID. If the resolution ID is not specified, the video with the highest resolution is selected by default.

[Response Parameters]

The type is MediaStreamTrack. For details about the MediaStreamTrack type, see **MediaStreamTrack**.

getType

```
getType(): string
```

[Function Description]

Obtains the stream type. This API is used to determine whether a stream is video or presentation. Generally, a presentation stream is a screen sharing stream.

[Request Parameters]

None

[Response Parameters]

The type is string. **local**: local stream; **main**: remote video stream; **auxiliary**: remote presentation stream.

on

```
on(event: string, handler: function): void
```

[Function Description]

Registers the callback for a stream object event.

[Request Parameters]

- **event**: (mandatory) event name. The type is string. For details about the event list, see **RTCStreamEvent**.
- **handler**: (mandatory) event processing method. The type is function.

[Response Parameters]

None

off

```
off(event: string, handler: function): void
```

[Function Description]

Deregisters the callback for a stream object event.

[Request Parameters]

- **event**: (mandatory) event name. The type is string. For details about the event list, see **RTCStreamEvent**.
- **handler**: (mandatory) event processing method. The type is function.

[Response Parameters]

None

getStreamInfo

```
getStreamInfo(): StreamInfo
```

[Function Description]

Obtains information about initialized local streams or received remote streams.

[Request Parameters]

None

[Response Parameters]

StreamInfo is defined as:

- **videoProfiles**: RTCVideoProfileInfo[] type.
- **audioProfile**: RTCAudioProfile type.

}

RTCVideoProfileInfo is defined as:

- **resolutionId**: (mandatory) resolution ID. The type is string.
- **hasTrack**: whether the video of this resolution has a playable track. The type is Boolean.
- **width**: width of the resolution, in pixels. The type is number.
- **height**: height of the resolution, in pixels. The type is number.
- **frameRate**: video frame rate, in fps. The type is number.
- **minBitrate**: minimum video bit rate, in bit/s. The type is number.
- **maxBitrate**: maximum video bit rate, in bit/s. The type is number.

}

RTCAudioProfile is defined as: {

- **sampleRate**: audio sampling ratio. The type is number.
- **channelCount**: number of audio channels. The type is number.

- **bitrate**: audio bitrate, in bit/s. The type is number.

}

3.5.5 Local Stream Object (LocalStream)

This object is inherited from the **Stream** object. The following APIs are added.

Table 3-8 LocalStream APIs

API	Description
initialize	Initializes local streams.
setAudioProfile	Sets audio stream profiling.
setVideoProfile	Sets video stream profiling.
setScreenProfile	Sets presentation stream profiling.
addAudioTrackCapture	Adds audio track capture if there is no audio track in the stream object after the stream is initialized. This API is added in version 1.10.0.
addVideoTrackCapture	Adds video track capture if there is no video track in the stream object after the stream is initialized. This API is added in version 1.10.0.
addResolution	Adds the videos with new resolutions to the initialized local streams.
removeResolution	Removes a video with a specific resolution from a stream.
addTrack	Adds audio and video tracks to a local stream object.
removeTrack	Removes audio and video tracks from a local stream object.
replaceTrack	Replaces audio and video tracks for a local stream object.
switchDevice	Switches the media input device.
startAudioMixing	Plays an online audio file.
stopAudioMixing	Stops playing an online audio file.
pauseAudioMixing	Pauses the playback of an online audio file.
resumeAudioMixing	Resumes the playback of an online audio file.
getAudioMixingDuration	Obtains the duration of an online audio file.
setAudioMixingVolume	Sets the volume of online audio.
setAudioMixingPosition	Sets the playback progress of online audio.

API	Description
getAudioMixingCurrent-Position	Obtains the playback progress of online audio.
bindScreenAudio2RelatedStream	Binds the screen sharing background audio to the related stream object. This API is added in version 1.4.0.

initialize

```
initialize(): Promise<StreamInitializeResult>
```

[Function Description]

Initializes a local audio and video stream object based on the input parameter **StreamConfig** of **createStream**. Streams can be played only after being initialized.

[Request Parameters]

None

[Response Parameters]

Stream initialization result, which is of the StreamInitializeResult type.

CAUTION

StreamInitializeResult indicates the stream initialization result. The object is inherited from **RtcError**. The following methods are provided:

- To obtain the initialization results of all media types, run the following command:
  ```
  getMediaCaptureResult(): MediaCaptureResult[]
  ```
- To obtain the initialization result of a specified media type, run the following command:
  ```
  getMediaCaptureResultByType(type: MediaType): MediaCaptureResult
  ```

MediaCaptureResult is defined as:

{

- **type**: (mandatory) media type. The type is MediaType. The values of MediaType can be **audio** and **video**.
- **track**: (optional) track generated if the media type is initialized successfully. The type is MediaStreamTrack.
- **error**: (optional) error generated if the media type fails to be initialized. The type is RtcError.

}

If the audio/video initialization fails, the error information is returned in the result. Related error codes are 90000001 and 90100017 to 90100020. For details, see **Table 3-13**.

setAudioProfile

```
setAudioProfile(profile: string|RTCAudioProfile): void
```

[Function Description]

Sets the audio stream sampling rate, number of audio channels, and bit rate. If this API is not called, the SDK uses the default value.

[Request Parameters]

profile: (mandatory) The type is string or RTCAudioProfile. If the type is string, the sampling rate, number of audio channels, and bitrate are shown in **Table 3-9**. If the type is RTCAudioProfile, you need to set this parameter as required. You are advised to use a defined **profile**. If an invalid value is used, the default profile **standard** is used.

Table 3-9 Sampling rate, number of audio channels, and bitrate of profile

profile	Sampling Rate (kHz)	Audio Channels	Bitrate (kbit/s)
low	16	1	24
standard	48	1	40
high	48	1	128

RTCAudioProfile is defined as: {

- **sampleRate**: (optional) sampling rate. The type is number.
- **channelCount**: (optional) number of audio channels. The type is number.
- **bitrate**: (optional) bitrate, in bit/s. The type is number.

}

[Response Parameters]

None

CAUTION

This method must be called before **initialize**.

setVideoProfile

```
setVideoProfile(profile: string|RTCVideoProfile,resolutionId?: string): void
```

[Function Description]

Sets video stream parameters, such as the resolution, frame rate, and bit rate. If this API is not called, the SDK uses the default value **360p_2**. If the video stream has been published, it will be automatically published to the remote end again.

[Request Parameters]

- **profile**: (mandatory) The type is string or RTCVideoProfile. If the type is string, the resolution, frame rate, and bitrate are shown in **Table 3-10**. If the type is RTCVideoProfile, you need to set this parameter as required. You are advised to use a defined **profile**. If an invalid value is used, the default profile **360p_2** is used.

  Table 3-10 Resolution, frame rate, and bitrate of profile

profile	Resolution	Frame Rate	Minimum Bitrate (kbit/s)	Maximum Bitrate (kbit/s)
90p_1	160 x 90	15	64	110
90p_2	120 x 90	15	64	110
120p_1	160 x 120	15	64	120
120p_2	120 x 120	15	64	110
180p_1	320 x 180	15	80	320
180p_2	240 x 180	15	80	170
180p_3	180 x 180	15	64	130
240p_1	320 x 240	15	100	400
240p_2	240 x 240	15	80	320
270p_1	480 x 270	15	160	600
300p_1	400 x 300	15	200	500
360p_1	640 x 360	15	200	800
360p_2	480 x 360	15	200	700
360p_3	360 x 360	15	150	600
450p_1	800 x 450	15	300	950
480p_1	640 x 480	15	250	900
480p_2	480 x 480	15	200	800
540p_1	960 x 540	15	400	1000
630p_1	1120 x 630	15	450	1150
720p_1	1280 x 720	15	500	1500
720p_2	960 x 720	15	450	1100
1080p_1	1920 x 1080	15	600	2000
1080p_2	1440 x 1080	15	550	1700

- **resolutionId**: (optional) The type is string. In the dual-stream scenario, specify the resolution ID of the video to be set. If the resolution ID is not specified, the video with the highest resolution is selected by default.

RTCVideoProfile is defined as: {

- **width**: (optional) width of the resolution, in pixels. The type is number.
- **height**: (optional) height of the resolution, in pixels. The type is number.
- **frameRate**: (optional) video frame rate, in fps. The type is number.
- **minBitrate**: (optional) minimum video bitrate, in bit/s. The type is number.
- **maxBitrate**: (optional) maximum video bitrate, in bit/s. The type is number.

}

[Response Parameters]

None

CAUTION

- This API cannot be dynamically called for custom streams created using **videoSource**. It can be called only for streams collected by cameras.
- Due to the limitations of the device capture capability, system performance, and browser, the actual video resolution, frame rate, and bitrate may not match the preset value. In this case, the browser automatically adjusts the resolution to match the preset value. The actual resolution is subject to the collected resolution.
- Whether 1080p or higher resolution data can be captured depends on the device capabilities and system performance. iOS Safari does not support 1080p data capture.
- The 1080p resolution in the profile table is newly added in version 2.0.0. If the 1080p resolution needs to be captured in version 1.0, customize a profile.
- Firefox does not support custom frame rate. The default frame rate is 30 fps.
- You can customize a profile to meet your service requirements.

setScreenProfile

```
setScreenProfile(profile: string|RTCScreenProfile): void
```

[Function Description]

Sets presentation stream parameters, such as the resolution, frame rate, and bit rate. If this API is not called, the SDK uses the default value **720p**.

[Request Parameters]

profile: (mandatory) The type is string or RTCScreenProfile. If the type is string, the resolution, frame rate, and bitrate are shown in **Table 3-11**. If the type is RTCScreenProfile, you need to set this parameter as required. You are advised to use a defined **profile**. If an invalid value is used, the default profile **720p** is used.

Table 3-11 Resolution, frame rate, and bit rate of profile

profile	Resolution	Frame Rate	Bitrate (kbit/s)
720p	1280 x 720	15	1200
1080p	1920 x 1080	15	2000

RTCScreenProfile is defined as: {

- **width**: (optional) width of the resolution, in pixels. The type is number.
- **height**: (optional) height of the resolution, in pixels. The type is number.
- **frameRate**: (optional) video frame rate, in fps. The type is number.
- **bitrate**: (optional) bitrate, in bit/s. The type is number.

}

[Response Parameters]

None

⚠ CAUTION

This method must be called before **initialize**.

addAudioTrackCapture

```
addAudioTrackCapture(microphoneId?: string): promise<MediaStreamTrack>
```

[Function Description]

Adds audio track capture to an initialized local stream object if the audio is not initialized or the audio fails to be initialized. If the local stream has been published, it will be automatically published to the remote end again. This API is added in version 1.10.0.

[Request Parameters]

microphoneId: (optional) ID of the microphone to be captured. The type is string. If this parameter is not passed, the SDK uses **microphoneId** specified in the input parameter **StreamConfig** of **createStream**.

[Response Parameters]

The type is MediaStreamTrack. It indicates the track that is successfully added.

⚠ CAUTION

After removing an audio track by calling the **removeTrack** AP, you can call this API to add the audio track again.

addVideoTrackCapture

```
addVideoTrackCapture(option?: VideoCaptureOption): promise<MediaStreamTrack>
```

[Function Description]

Adds video track capture to an initialized local stream object if the video is not initialized or the video fails to be initialized. If the local stream has been published, it will be automatically published to the remote end again. This API is added in version 1.10.0.

[Request Parameters]

option: (optional) capture parameters. The type is VideoCaptureOption.

VideoCaptureOption is defined as: {

- **cameraId**: (optional) The type is string. It specifies the ID of a camera. For Android devices, **user** indicates the front-facing camera, and **environment** indicates the rear-facing camera. If this parameter is not passed, the SDK uses **cameraId** and **facingMode** specified in the input parameter **StreamConfig** of **createStream**.
- **resolutionId**: (optional) The type is string. It specifies the resolution ID of the video to be configured. If this parameter is not passed, the video with the highest resolution is selected by default.

}

[Response Parameters]

The type is MediaStreamTrack. It indicates the track that is successfully added.

CAUTION

After removing a video track by calling the **removeTrack** API, you can call this API to add the video track again.

addResolution

```
addResolution(profile: string|RTCVideoProfile,audio?: boolean): promise<RTCVideoProfileInfo>
```

[Function Description]

Adds a video with a specific resolution to an initialized local stream object. If the local stream has been published, it will be automatically published to the remote end again.

[Request Parameters]

- **profile**: (mandatory) parameter information about the video resolution to be added. The type is string. For details about the RTCVideoProfile description, see **setVideoProfile**.
- **audio**: (optional) whether to create the audio. The type is Boolean. The value **true** indicates yes and the value **false** indicates no. The audio in a stream is common to all resolutions in the stream. If the audio is enabled but no audio track is configured, the audio is created by default.

[Response Parameters]

The type is RTCVideoProfileInfo. It indicates the profile information of the resolution of the video that is added successfully. For details about the definition of RTCVideoProfileInfo, see **getStreamInfo**.

If the operation fails, **StreamInitializeResult** is returned. For details about the definition of **StreamInitializeResult**, see **initialize**.

CAUTION

- Currently, a local stream supports a maximum of two resolutions.
- When the high- and low-quality streams are enabled, there are restrictions on the **setVideoProfile** API for setting stream resolution. The maximum resolution allowed is subject to the stream of higher resolution. Exercise caution when using this API.

removeResolution

```
removeResolution(resolutionId: string): promise<void>
```

[Function Description]

Removes the video resolution of a local stream object. If the local stream has been published, it will be automatically published to the remote end again.

[Request Parameters]

resolutionId: (mandatory) ID of the video resolution to be removed. The type is string.

[Response Parameters]

Promise<void>: returns a **Promise** object.

addTrack

```
addTrack(track: MediaStreamTrack,resolutionId?: string ): promise<void>
```

[Function Description]

Adds audio and video tracks to an initialized local stream object. If the local stream has been published, it will be automatically published to the remote end again.

[Request Parameters]

- **track**: (mandatory) track to be added. The type is MediaStreamTrack.
- **resolutionId**: (optional) The type is string. In the dual video stream scenario, specify the resolution ID. If the resolution ID is not specified, the video with the highest resolution is selected by default.

[Response Parameters]

Promise<void>: returns a **Promise** object.

CAUTION

If a video track with the same resolution already exists, the video track cannot be added again.

removeTrack

```
removeTrack(track: MediaStreamTrack): promise<void>
```

[Function Description]

Removes audio and video tracks from a local stream object. If the local stream has been published, it will be automatically published to the remote end again.

[Request Parameters]

track: (mandatory) track to be removed. The type is MediaStreamTrack.

[Response Parameters]

Promise<void>: returns a **Promise** object.

CAUTION

- If all audio tracks of a microphone are removed, the SDK will not access the microphone.
- If all video tracks of a camera are removed, the SDK will not access the camera.

replaceTrack

```
replaceTrack(track: MediaStreamTrack, type: "audio" | "video",resolutionId?: string): promise<void>
```

[Function Description]

Replaces audio and video tracks for an initialized local stream object. If the local stream has been published, it will be automatically published to the remote end again.

[Request Parameters]

- **track**: (mandatory) The type is MediaStreamTrack.
- **type**: (mandatory) **audio** or **video**. The type is string.
- **resolutionId**: (optional) The type is string. This parameter is valid when **type** is set to **video**. Specify the resolution ID of the video to be replaced. If the resolution ID is not specified, the video with the highest resolution is selected by default.

[Response Parameters]

Promise<void>: returns a **Promise** object.

switchDevice

```
switchDevice(deviceType: "audio" | "video", deviceId: string): Promise<void>
```

[Function Description]

Switches the media input device. If the local stream has been published, it will be automatically published to the remote end again.

[Request Parameters]

- **deviceType**: (mandatory) **audio** or **video**. The type is string.
- **deviceId**: (mandatory) ID of an input device. The type is string.

[Response Parameters]

Promise<void>: returns a **Promise** object.

startAudioMixing

```
startAudioMixing(option: AudioMixOption): Promise<void>
```

[Function Description]

Plays an online audio file.

[Request Parameters]

option: (mandatory) audio file playback parameters. The type is AudioMixOption.

AudioMixOption is defined as: {

- **filePath**: (mandatory) path for downloading an online audio file. The type is string. The supported audio formats include MP3, AAC, and other audio formats supported by the browser.
- **startTime**: (optional) time when the audio file starts to be played. The type is number. The default value is **0**.
- **replace**: (optional) The type is Boolean. **true** indicates that the local audio stream is replaced with the audio file. The default value is **false**.
- **loop**: (optional) The type is Boolean. **true** indicates that infinite loop playback is required. The default value is **false**.
- **repeatCount**: (optional) number of playback times. The type is number. **0** indicates that repetition is disabled. The default value is **0**.

}

[Response Parameters]

Promise<void>: returns a **Promise** object.

CAUTION

- When **loop** is set to **true**, **repeatCount** must be set to **0**.
- The audio mixing APIs can be called only after the **publish** API is called successfully.

stopAudioMixing

```
stopAudioMixing(): Promise<void>
```

[Function Description]

Stops playing an online audio file.

[Request Parameters]

None

[Response Parameters]

Promise<void>: returns a **Promise** object.

pauseAudioMixing

```
pauseAudioMixing(): void
```

[Function Description]

Pauses the playback of an online audio file.

[Request Parameters]

None

[Response Parameters]

None

resumeAudioMixing

```
resumeAudioMixing(): void
```

[Function Description]

Resumes the playback of an online audio file.

[Request Parameters]

None

[Response Parameters]

None

getAudioMixingDuration

```
getAudioMixingDuration(): number
```

[Function Description]

Obtains the duration of an online audio file. The unit is millisecond.

[Request Parameters]

None

[Response Parameters]

Duration of an audio file. The type is number.

setAudioMixingVolume

```
setAudioMixingVolume(level: number): void
```

[Function Description]

Sets the volume of online audio.

[Request Parameters]

level: (mandatory) volume level. The type is number.

[Response Parameters]

None

setAudioMixingPosition

```
setAudioMixingPosition(position: number): void
```

[Function Description]

Sets the playback progress of online audio.

[Request Parameters]

position: (mandatory) progress. The value cannot be greater than the actual audio duration. The type is number. The unit is millisecond.

[Response Parameters]

None

getAudioMixingCurrentPosition

```
getAudioMixingCurrentPosition(): number
```

[Function Description]

Obtains the playback progress of online audio.

[Request Parameters]

None

[Response Parameters]

Current audio playback progress. The type is number. The unit is millisecond.

bindScreenAudio2RelatedStream

```
bindScreenAudio2RelatedStream(bindStream: LocalStream, muteMainStreamAudio?: boolean): void
```

[Function Description]

Binds the screen sharing background audio to the related stream object.

[Request Parameters]

- **bindStream**: (mandatory) local video stream object that is successfully created and initialized. The type is LocalStream.
- **muteMainStreamAudio**: (optional) whether to mute the video stream. The type is Boolean. The default value is **false**. **true** indicates muted; **false** indicates unmuted.

[Response Parameters]

None

CAUTION

- This API can be called only by the presentation stream object created using **HRTC.createStream**.
- The screen sharing background audio needs to be sent through the audio channel of the video stream. If you want to subscribe to the sharing background audio, you need to subscribe to the video stream audio at least.
- In the screen sharing dialog box that is displayed, select the check box for audio sharing in the lower left corner. Otherwise, the background audio cannot be shared.
- This function applies only to Chrome 74 and later versions on Windows.

3.5.6 Remote Stream Object (RemoteStream)

This object is inherited from the **Stream** object.

3.5.7 Stream Event Notification (RTCStreamEvent)

This section describes the **RTCStreamEvent** events of the Web SDK.

Table 3-12 StreamEvent events

API	Description
player-state-change	The player status is changed.
screen-sharing-stopped	Screen sharing is stopped.
audio-mixing-played	Local mixed audio is played.
audio-mixing-finished	Local mixed audio playback is stopped.

CAUTION

Registration listening must be canceled when the service ends. Otherwise, memory leakage may occur when there are a certain number of registration listening events.

player-state-change

[Event Description]

This event is triggered when the player state is changed.

[Callback Parameters]

event: The type is playState. The fields are defined as follows:

- **type**: player type. The type is string. The options are **video** and **audio**.
- **id**: stream resolution ID. The type is string.

- **state**: current playback status. The type is string. The value is **PLAYING**, **STOPPED**, **PAUSED**, or **NONE**.
- **reason**: reason that triggers the playback status change. The type is string.

screen-sharing-stopped

[Event Description]

This event is triggered only when local screen sharing is stopped.

[Callback Parameters]

event: stream ID when screen sharing is stopped. The type is string.

audio-mixing-played

[Event Description]

Mixed audio is played. This event is triggered only when the local mixed audio is played.

[Callback Parameters]

None

audio-mixing-finished

[Event Description]

Mixed audio playback ends. This event is triggered only when the local mixed audio playback ends.

This event is not triggered when you manually call **stopAudioMixing** and **pauseAudioMixing**.

[Callback Parameters]

None

3.5.8 Error Code (RtcError)

getCode

```
getCode(): number
```

[Function Description]

Obtains an error code.

[Request Parameters]

None

[Response Parameters]

Error code. The type is number.

getMsg

```
getMsg(): string
```

[Function Description]

Obtains error description.

[Request Parameters]

None

[Response Parameters]

Error message. The type is string.

3.5.9 Error Codes Reported on the Client

This section describes details about the error codes (**RtcErrorCode**) reported on the client of the Web SDK.

Table 3-13 Error code description

Class Member	Error Code	Description	Error Cause or Handling Suggestion
RTC_ERR_CODE_SUCCESS	0	success	Succeeded.
RTC_ERR_CODE_RTC_SDK_ERROR	90000001	sdk internal error	SDK internal error. Contact Huawei technical support.
RTC_ERR_CODE_WAIT_RSP_TIMEOUT	90000004	message response timeout	Response timed out. Contact Huawei technical support.
RTC_ERR_CODE_INVALID_PARAMETER	90000005	invalid parameter	Incorrect parameter. For details, see the API documentation.
RTC_ERR_CODE_INVALID_OPERATION	90100001	illegal operation	Invalid operation. The user status is incorrect.
RTC_ERR_CODE_NOT_SUPPORT_MEDIA_DEVICES	90100002	not support enumerate devices	The browser does not support the enumerateDevices method.
RTC_ERR_CODE_NO_AVAILABLE_DEVICES	90100003	no available devices	No available device is found. Check whether the device is ready.
RTC_ERR_CODE_NO_AVAILABLE_VIDEO_INPUT_DEVICES	90100004	no available video input devices	No available camera device is found. Check whether the video capture device is ready.
RTC_ERR_CODE_NO_AVAILABLE_AUDIO_INPUT_DEVICES	90100005	no available audio input devices	No available audio device is found. Check whether the audio capture device is ready.

Class Member	Error Code	Description	Error Cause or Handling Suggestion
RTC_ERR_CODE_NO_AVAILABLE_AUDIO_OUTPUT_DEVICES	90100006	no available audio output devices	No audio output device found.
RTC_ERR_CODE_STATUS_ERROR	90100007	room status error	Incorrect room status. Check whether the user can join the meeting.
RTC_ERR_CODE_WEBSOCKET_NOT_CONNECTED	90100008	websocket connection state is not "CONNECTED"	WebSocket connection failed. Check the connection status.
RTC_ERR_CODE_WAIT_CONFIG_FAIL	90100009	wait server config fail	Failed to obtain the delivered configuration. Contact Huawei technical support.
RTC_ERR_CODE_PUBLISH_RESPONSE_FAIL	90100010	publish response fail	Failed to respond to the stream release request. Contact Huawei technical support.
RTC_ERR_CODE_REGION_NOT_COVERED	90100011	current region is not covered, service unavailable	The server IP address is not found. Contact Huawei engineers.
RTC_ERR_CODE_WEBSOCKET_CONNECT_TIMEOUT	90100012	websocket connect timeout	WebSocket connection establishment timed out. Contact Huawei technical support.
RTC_ERR_CODE_WEBSOCKET_RECONNECT_TIMEOUT	90100013	websocket reconnect timeout	WebSocket reconnection timed out. Contact Huawei technical support.
RTC_ERR_CODE_WEBSOCKET_NOT_OPEN	90100014	websocket is not open	WebSocket connection is not opened. Contact Huawei technical support.
RTC_ERR_CODE_WEBSOCKET_INTERRUPTED	90100015	websocket connection state is idle, interrupt operation	WebSocket connection is forcibly closed. The cause may be that the user leaves the meeting or reconnects to the meeting.
RTC_ERR_CODE_WEBSOCKET_CONNECT_ERROR	90100016	websocket connect error	WebSocket listening error. The server proactively closes the connection.

Class Member	Error Code	Description	Error Cause or Handling Suggestion
RTC_ERR_CODE_CAPTURE_PERMISSION_DENIED	90100017	capture failed, permission denied	Collection failed. The data collection permission is not granted to the audio/video device. You are advised to ask the user to grant the access permission to the camera/microphone.
RTC_ERR_CODE_CAPTURE_OVER_CONSTRAINED	90100018	capture failed, Constraint parameter invalid	Collection failed. The audio/video capture device does not support the configured collection restrictions.
RTC_ERR_CODE_CAPTURE_DEVICE_NOT_FOUND	90100019	capture failed, requested device not found	Collection failed because the device is not found. You are advised to ask the user to check whether the camera or microphone required for the call is ready before the call starts.
RTC_ERR_CODE_CAPTURE_DEVICE_NOT_READABLE	90100020	capture failed, maybe device is occupied by other application	Collection failed because the device has been occupied. Check the usage status. You are advised to give a prompt message indicating that the camera or microphone cannot be accessed temporarily, and ask the user to ensure that other applications are not requesting for accessing the camera or microphone and try again.
RTC_ERR_CODE_PLAY_NOT_ALLOW	90100021	the user didn't interact with the document first, please trigger by gesture	Playback is not allowed.
RTC_ERR_CODE_ROLE_NO_PERMISSION	90100022	the user role have no permission to operate	The user role does not have the permission. Check the user role.
RTC_ERR_CODE_ANSWER_SDP_INVALID	90100023	the answer sdp is invalid	SDP negotiation error. Contact Huawei technical support.

Class Member	Error Code	Description	Error Cause or Handling Suggestion
RTC_ERR_CODE_MEDIA_UPSTREAM_UNSUPPORTED	90100024	the upstream media is not supported	The browser does not support media capture.
RTC_ERR_CODE_INTERNAL_EXCEPTION	90100104	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_SEND_TO_RTCCTRL_FAIL	90100105	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_RTCCTRL_EXCEPTION	90100106	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_WS_NOT_SUPPORTED	90100107	websocket is unsupported	WebSocket is not supported.
RTC_ERR_CODE_REQUEST_FREQUENTLY	90100108	request too frequently	The request is too frequent. Check the program.
RTC_ERR_CODE_INVALID_REQUEST_BODY	90100109	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_INVALID_RESPONSE	90100110	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_INTERNAL_ERROR	90100199	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_INVALID_CRTL_PARAMETER	90100201	request parameter is invalid	Invalid request parameter. Contact Huawei technical support.
RTC_ERR_CODE_INVALID_REQUEST	90100202	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_JOIN_ROOM_FAIL	90100203	join room fail	Failed to join the room. Contact Huawei technical support.
RTC_ERR_CODE_ALREADY_JOINED	90100204	joined already	You have already joined the room. Contact Huawei technical support.

Class Member	Error Code	Description	Error Cause or Handling Suggestion
RTC_ERR_CODE_NO_SFU_RESOURCE	90100205	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_INVALID_SSRC	90100206	invalid ssrc	The SSRC number is invalid. Contact Huawei technical support.
RTC_ERR_CODE_STREAM_ALREADY_PUSHED	90100207	stream is pushed already	The stream has already been pushed. Contact Huawei technical support.
RTC_ERR_CODE_SAVE_USER_ERROR	90100208	save user fail	Failed to save user information. Contact Huawei technical support.
RTC_ERR_CODE_UPDATE_USER_ERROR	90100209	update user fail	Failed to update user information. Contact Huawei technical support.
RTC_ERR_CODE_FETCH_USER_ERROR	90100210	fetch user fail	Failed to obtain user information. Contact Huawei technical support.
RTC_ERR_CODE_SAVE_ROOM_ERROR	90100211	save room fail	Failed to save the room information. Contact Huawei technical support.
RTC_ERR_CODE_UPDATE_ROOM_ERROR	90100212	update room fail	Failed to update the room information. Contact Huawei technical support.
RTC_ERR_CODE_INVALID_SUBSCRIBED_INFO	90100213	invalid subscribe info	The subscription information is incorrect. Contact Huawei technical support.
RTC_ERR_CODE_INVALID_PUSHED	90100214	invalid push stream	The pushed stream is invalid. Contact Huawei technical support.
RTC_ERR_CODE_SFU_FAIL	90100215	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_REQUEST_SFU_FAIL	90100216	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_OVERLOAD_ROOM_CAPACITY	90100217	overload room capacity	The number of users in the room has reached the maximum. Contact Huawei technical support.

Class Member	Error Code	Description	Error Cause or Handling Suggestion
RTC_ERR_CODE_INVALID_ENCODING	90100218	invalid encoding	Encoding failed. Contact Huawei technical support.
RTC_ERR_CODE_APPLY_SSRC_FAIL	90100220	apply ssrc fail	Failed to apply for an SSRC number. Contact Huawei technical support.
RTC_ERR_CODE_CTRL_TO_ACS_FAIL	90100221	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_KICK_USER_FAIL	90100222	kick user fail	Failed to kick out the user. Contact Huawei technical support.
RTC_ERR_CODE_APPLY_TO_PUSH_STREAM_FAIL	90100223	apply to push stream fail	Failed to request for pushing the stream. Contact Huawei technical support.
RTC_ERR_CODE_ROOM_IDLE	90100224	room is released	The room has already been released. Contact Huawei technical support.
RTC_ERR_CODE_KICKED_USER	90100225	user has been kicked out	The user has already been kicked out. Contact Huawei technical support.
RTC_ERR_CODE_SUBSCRIBED_USER_CONNECT_ERROR	90100226	subscribed user connected error	The link of the subscribed-to user is incorrect. Contact Huawei technical support.
RTC_ERR_CODE_ROOM_NOT_EXIST	90100227	room is not exist	The room does not exist. Contact Huawei technical support.
RTC_ERR_CODE_MCU_FAIL	90100228	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_ROLE_CHANGED_FAIL	90100229	role is unchanged	The role does not change. Contact Huawei technical support.
RTC_ERR_CODE_NO_MCU_RESOURCE	90100230	no available resources	Resources are insufficient. Contact Huawei technical support.
RTC_ERR_CODE_CREATE_PUSH_TASK_FAIL	90100231	server internal exception	Internal server error. Contact Huawei technical support.

Class Member	Error Code	Description	Error Cause or Handling Suggestion
RTC_ERR_CODE_MIX_TASK_EXISTS	90100232	the task is exist	The mixed stream task exists. Contact Huawei technical support.
RTC_ERR_CODE_MIX_TASK_NOT_EXISTS	90100233	the task is not exist	The mixed stream task does not exist. Contact Huawei technical support.
RTC_ERR_CODE_MIX_USER_EMPTY	90100234	no available users	The mixed stream user is empty. Contact Huawei technical support.
RTC_ERR_CODE_STOP_AUDIO_FAIL	90100235	stop audio fail	Failed to stop the audio stream. Contact Huawei technical support.
RTC_ERR_CODE_STOP_VIDEO_FAIL	90100236	stop video fail	Failed to stop the video stream. Contact Huawei technical support.
RTC_ERR_CODE_ROOM_TYPE_NOT_MATCH	90100237	room mode is not matched	The room mode is incorrect. Contact Huawei technical support.
RTC_ERR_CODE_AUDIO_LEVEL_NOT_MATCH	90100238	audio level is not matched	The audio level is incorrect. Contact Huawei technical support.
RTC_ERR_CODE_NOT_SUPPORT_MIX	90100239	unsupported operation	Mixed streams are not supported. Contact Huawei technical support.
RTC_ERR_CODE_VERSION_NOT_COMPLICATE	90100240	not matched	The version is incompatible. Contact Huawei technical support.
RTC_ERR_CODE_NOT_SUPPORT_SWITCH_ROLE	90100241	switch role is unsupported	Role switching is not supported.
RTC_ERR_CODE_SEND_TO_MCU_FAIL	90100242	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_REDIRECT_MODE_NOT_MATCH	90100243	unsupported operation	Invalid operation.
RTC_ERR_CODE_ANCHOR_NUMBER_OUT_OF_RANGE	90100246	the number of anchors has reached the maximum	The number of streamers exceeds the maximum. Check the program.

Class Member	Error Code	Description	Error Cause or Handling Suggestion
RTC_ERR_CODE_TOPN_SUB_MODE_FAULT	90100247	topn invalid parameters	The TopN subscription parameter is invalid. Contact Huawei technical support.
RTC_ERR_CODE_NO_EFFECT_RESPONSE	90100248	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_SFU_CTRL_EXCEPTION	90100249	server internal exception	Internal server error. Contact Huawei technical support.
RTC_ERR_CODE_DESKTOP_NUMBER_OUT_OF_RANGE	90100250	the number of desktop streams has reached the maximum	The number of desktop sharing streams exceeds the maximum. Check the program.
RTC_ERR_CODE_CLOUD_INTERNAL_ERROR	90100299	server internal error	Internal server error. Contact Huawei technical support.

3.5.10 Server Error Codes

When a network or media error occurs during SDK running and the error cannot be automatically recovered, an error message must be displayed. The error code is generated by the server and returned through onError.

Table 3-14 Server error codes

Error Code	Description	Cause
RTC.10000001	Internal error.	Program or environment issues
RTC.31000000	The node does not exist.	Program or environment issues
RTC.31000001	Failed to verify the session.	Program or environment issues
RTC.31000003	Internal error.	Program or environment issues
RTC.31000004	Authentication failed.	User operation issues
RTC.31000005	Please try again.	User operation issues
RTC.31000006	Clock synchronization is required.	User operation issues

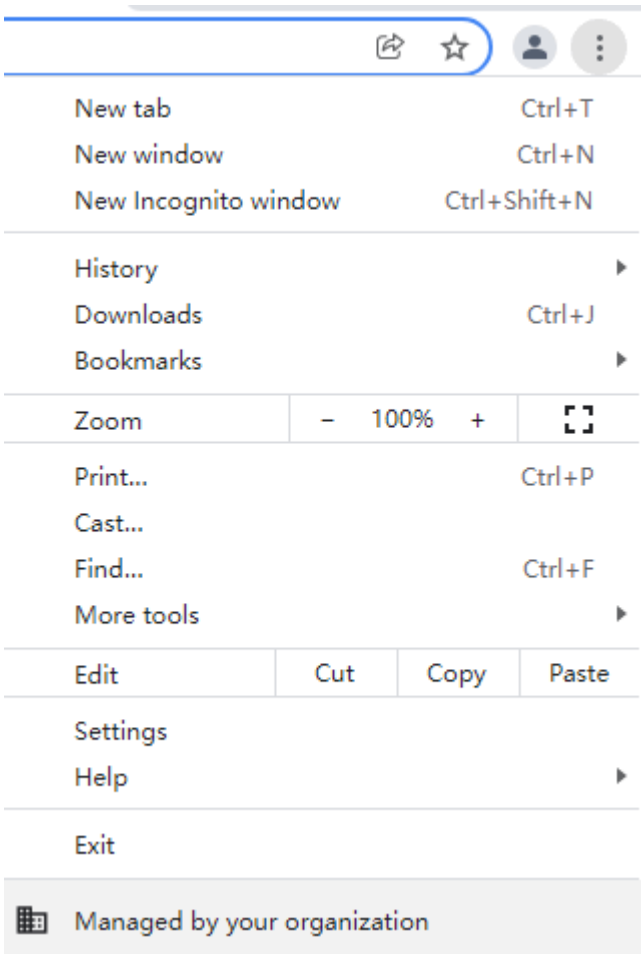
Error Code	Description	Cause
RTC.31000007	The requested resource does not exist.	Program or environment issues
RTC.32000000	Service error.	Program or environment issues
RTC.32000001	The maximum number of users in a selective forwarding unit (SFU) has been reached.	Program or environment issues
RTC.32000002	The SFU is empty.	Program or environment issues
RTC.32000004	Failed to deliver stream information.	Program or environment issues
RTC.32000005	Failed to add the adapter.	Program or environment issues
RTC.32000006	Failed to add the route.	Program or environment issues
RTC.32000007	Failed to obtain the user information.	Program or environment issues
RTC.32000010	The user to be viewed does not exist.	Program or environment issues
RTC.32000011	Invalid audio rate.	Program or environment issues
RTC.32000012	The user list is empty.	Program or environment issues
RTC.32000013	Invalid request parameter.	Program or environment issues
RTC.32000015	Internal calling error.	Program or environment issues
RTC.32000016	Internal calling error.	Program or environment issues
RTC.32000017	The site does not exist.	Program or environment issues
RTC.32000018	Incorrect encryption algorithm.	Program or environment issues
RTC.32000019	Failed to decrypt the Base64-encoded media key of the client.	Program or environment issues
RTC.32000020	Failed to generate the media key.	Program or environment issues
RTC.32000021	Failed to deliver encryption information.	Program or environment issues

Error Code	Description	Cause
RTC.32000022	Failed to obtain the SFU IP address.	Program or environment issues
RTC.32000024	Internal calling error.	Program or environment issues
RTC.32000025	Internal calling error.	Program or environment issues
RTC.32000028	Invalid operation.	Program or environment issues
RTC.32000030	Insufficient SFU resources.	Program or environment issues
RTC.32000032	The maximum number of rooms that a user can join has been reached.	User operation issues
RTC.32000033	Do not join the same room repeatedly.	User operation issues
RTC.32000034	Try again later.	Program or environment issues
RTC.33000000	Service error.	Program or environment issues
RTC.33000001	The node does not exist.	Program or environment issues
RTC.34000001	The room is full.	User operation issues
RTC.34000002	The room does not exist.	Program or environment issues
RTC.34000003	The site does not exist.	Program or environment issues
RTC.34000004	Internal calling error.	Program or environment issues
RTC.34000006	The user does not exist.	User operation issues
RTC.34000007	Presentation sharing is in progress.	User operation issues

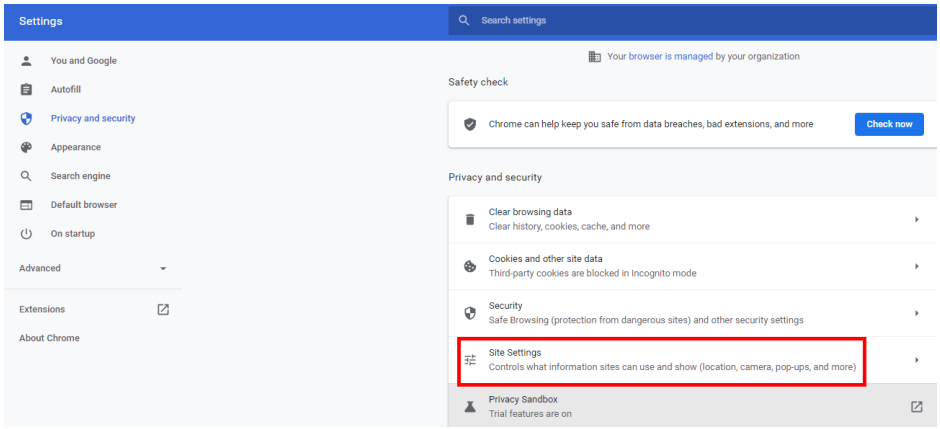
3.5.11 Granting the Permissions of Accessing Cameras or Microphones to a Browser

Google Chrome

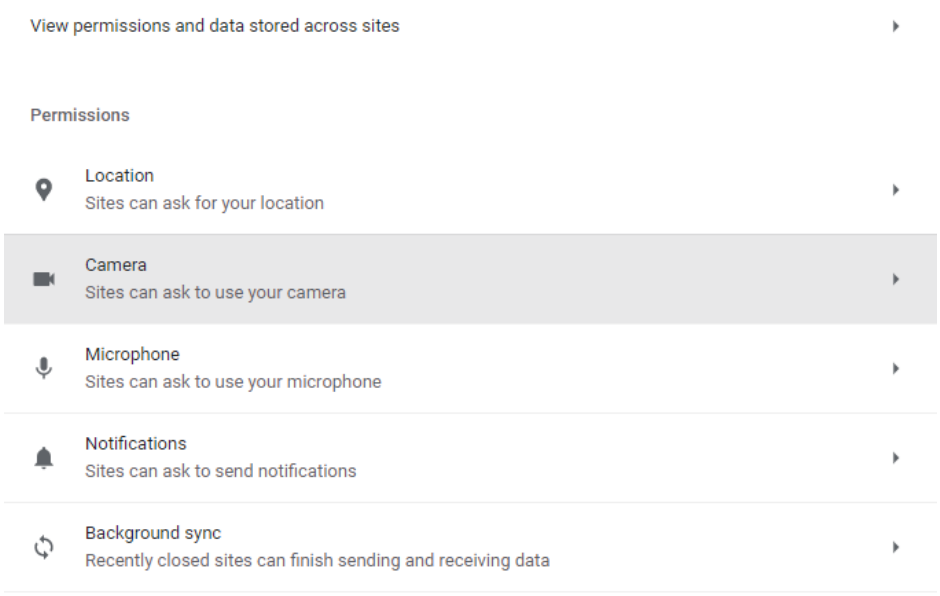
1. Open Google Chrome and click the settings icon in the upper right corner.



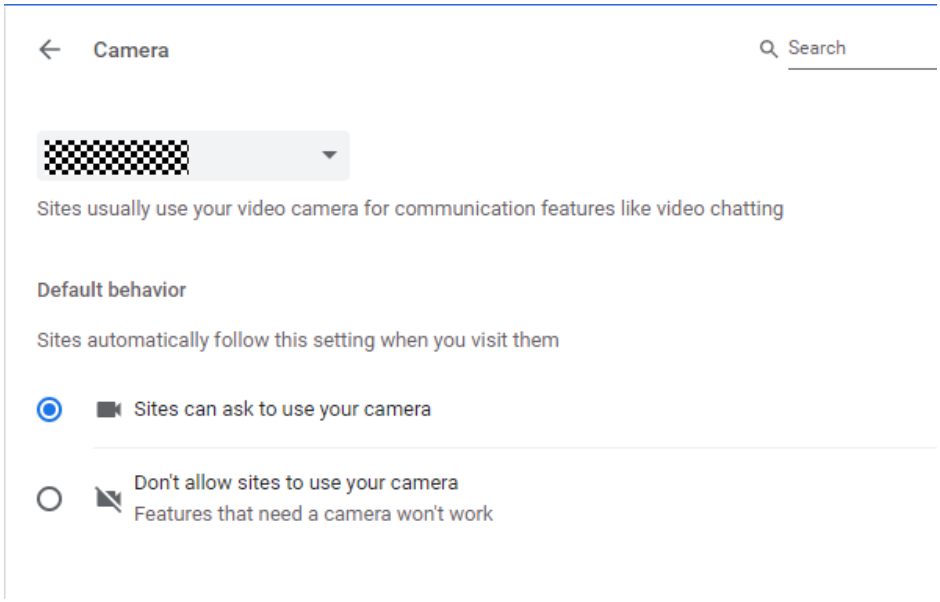
2. Choose **Settings** from the drop-down list. On the **Settings** page, select **Privacy and security**, and then click **Site Settings**.

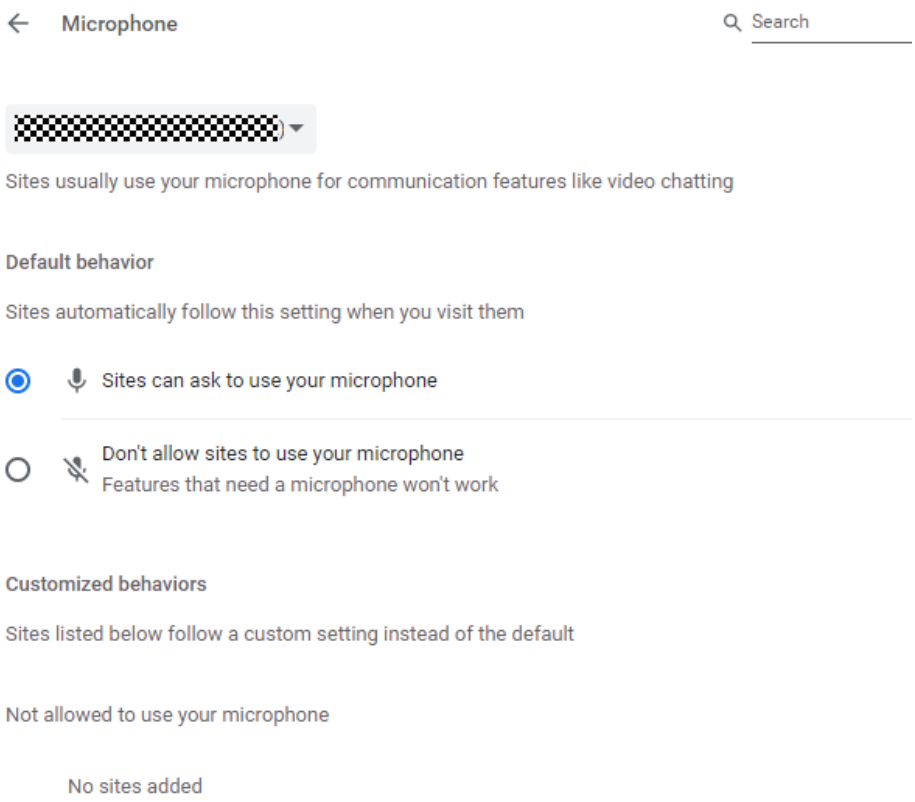


3. On the **Site Settings** page, click **Camera** or **Microphone**.

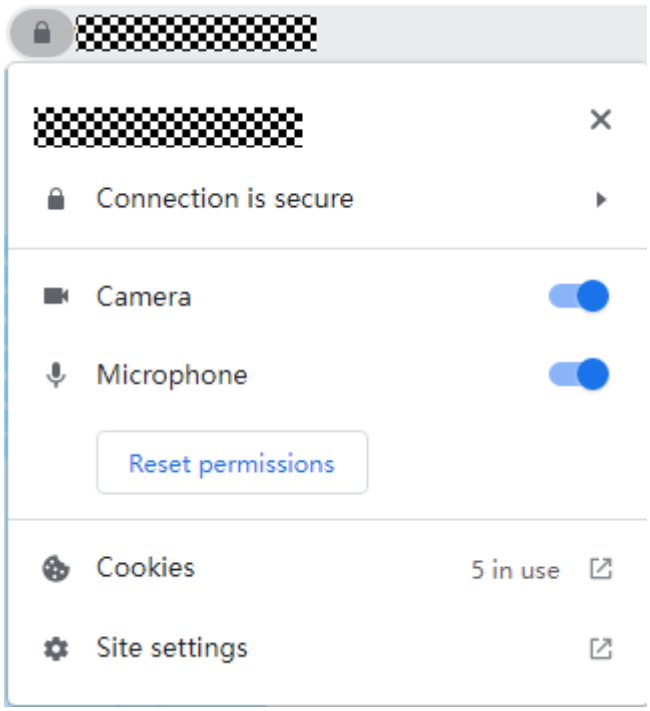


4. On the camera or microphone permissions page, select **Sites can ask to use your camera** or **Sites can ask to use your microphone**.



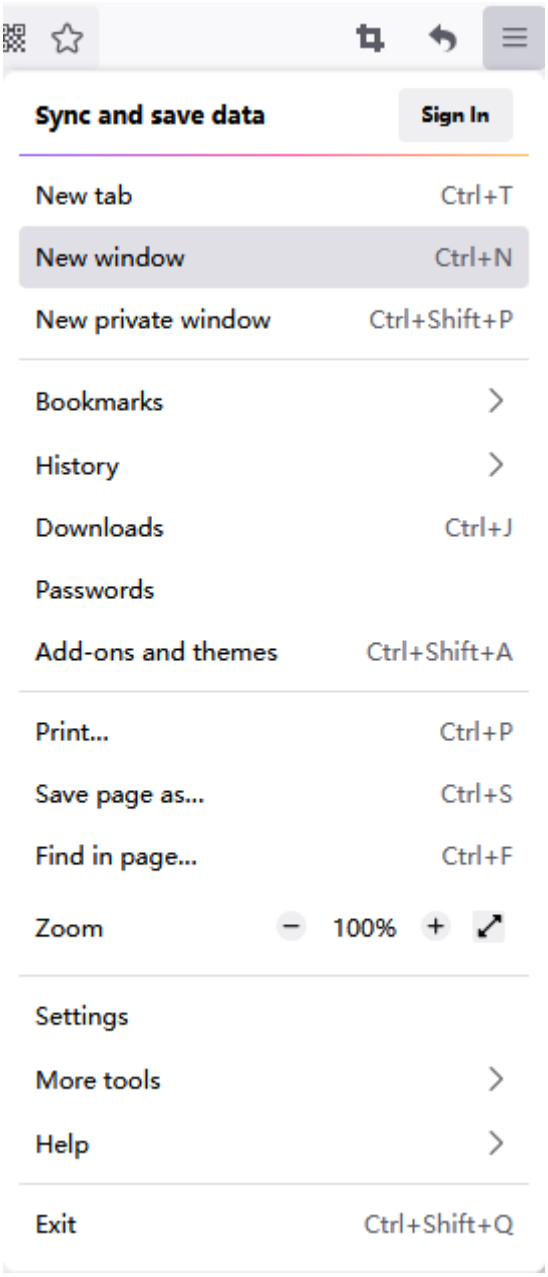


5. After the preceding settings are complete, the browser displays a dialog box asking you whether to allow a website to use the camera or microphone. Enable the **Camera** or **Microphone** switch.

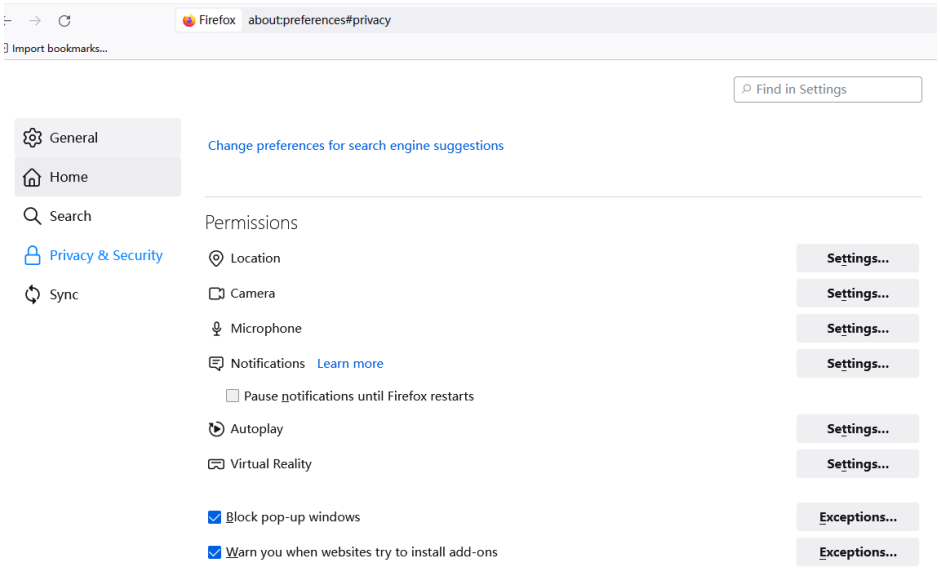


Firefox Browser

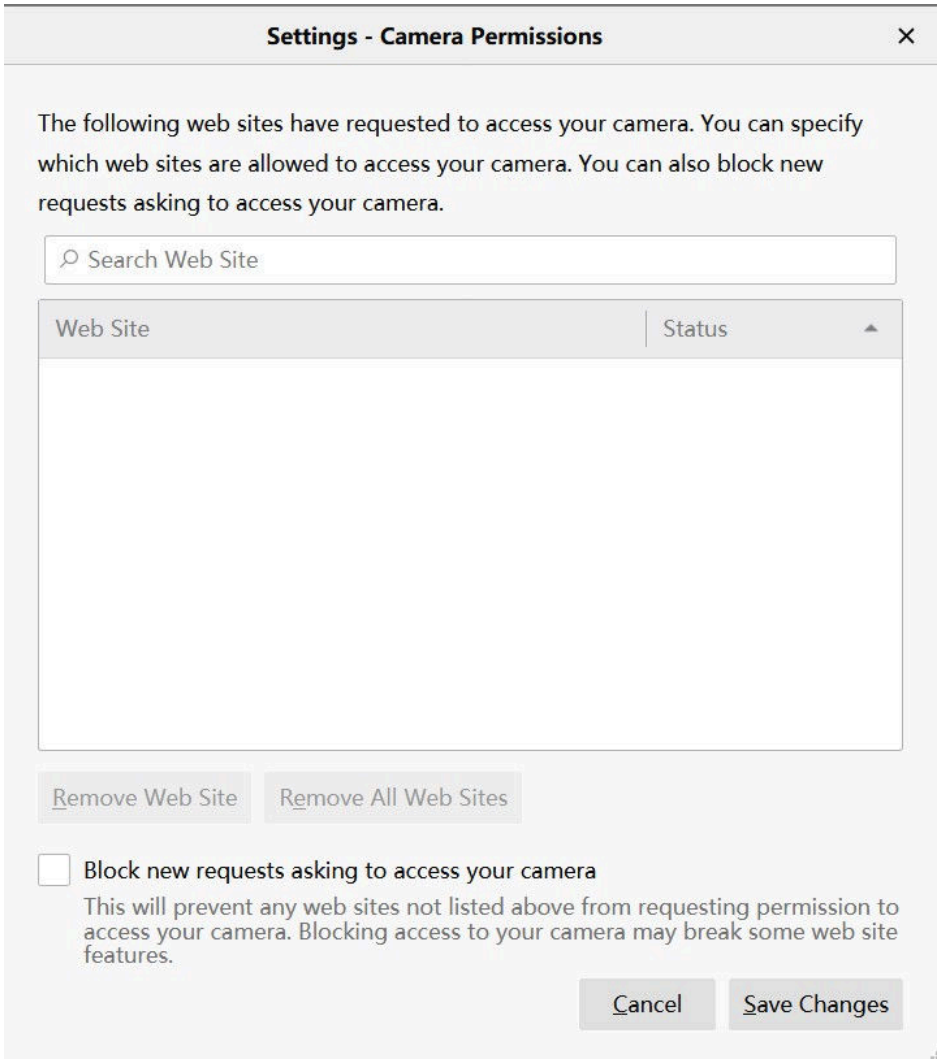
1. Open the Firefox browser, click the settings icon in the upper right corner, and choose **Settings** from the drop-down list.



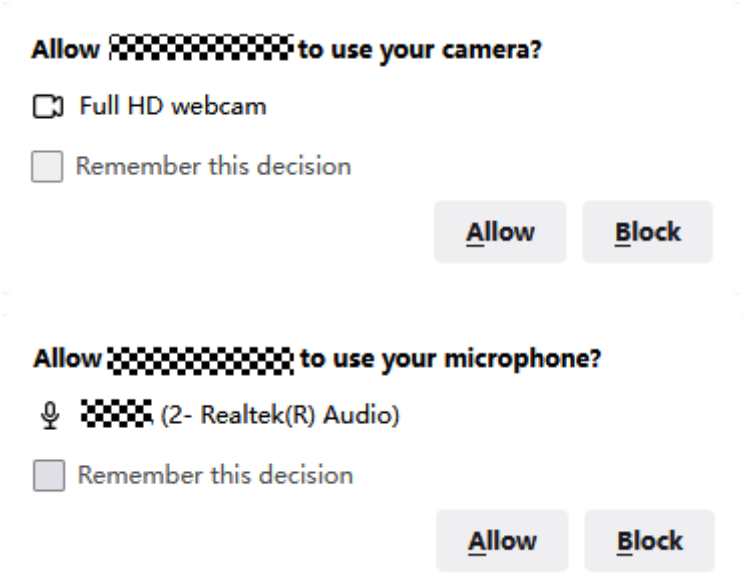
2. Click **Privacy & Security**, and click **Settings** nest to **Camera** or **Microphone**.



3. On the permissions settings page, add the websites that request the camera and microphone permissions to the list and click **Save Changes**.



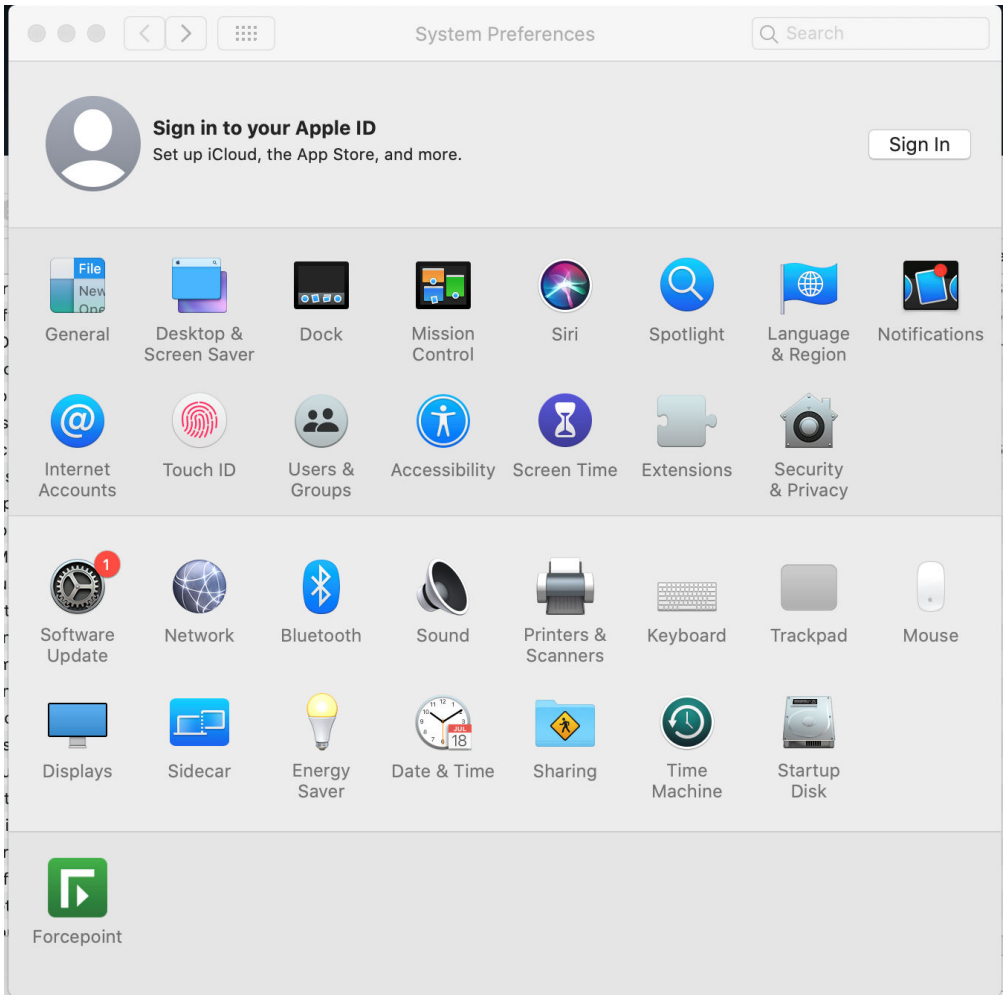
4. After the preceding settings are complete, the browser displays a dialog box asking you whether to allow a website to use the camera or microphone. Select **Allow**.



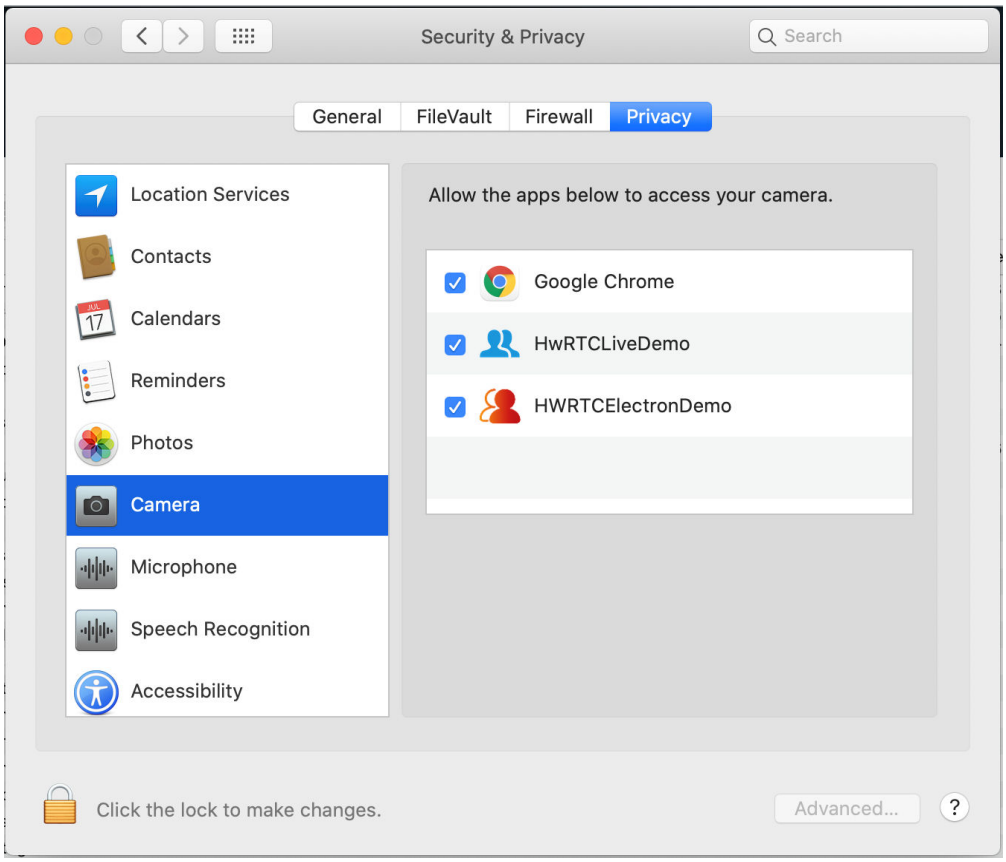
macOS Browser

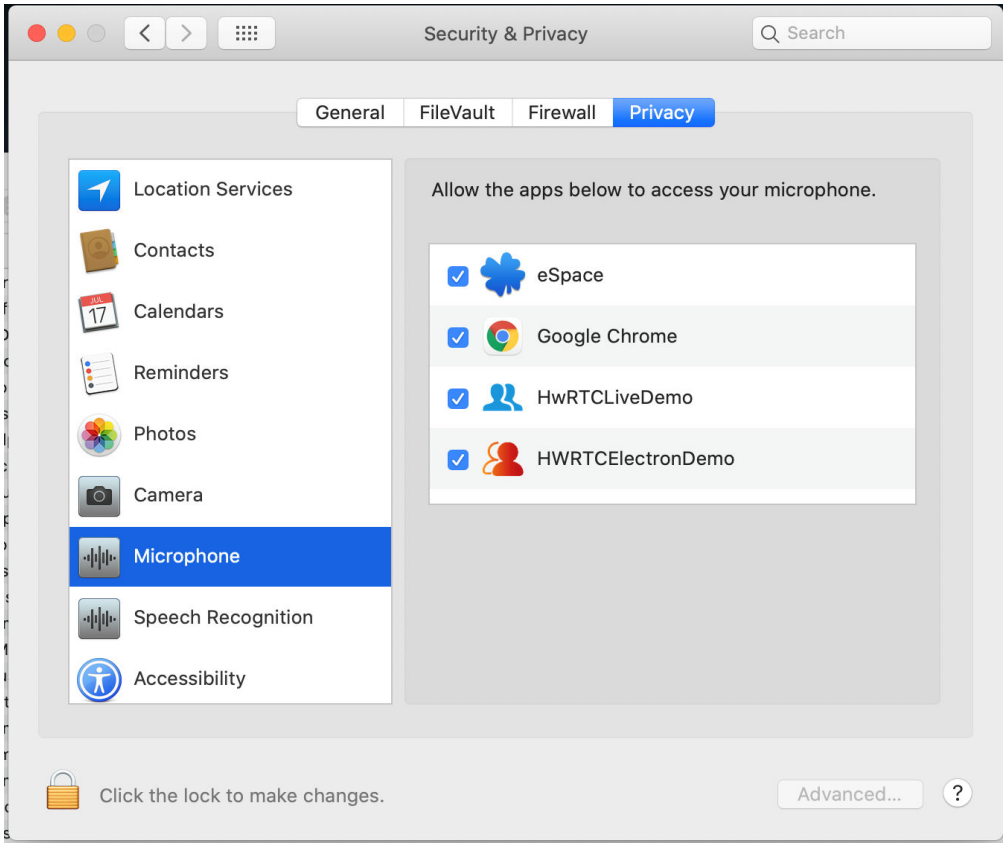
1. Click the **System Preferences** icon in the Dock.

2. Click **Security & Privacy**.



3. On the **Security & Privacy** page, click the **Privacy** tab, click **Camera** or **Microphone**, and select the apps that are allowed to use the camera or microphone.





3.6 FAQ

- **Is userName mandatory when I join a room?**

  This parameter is optional. The values of **userName** and **userId** are defined by the app and can be the same.

  - **userId**: (mandatory) user ID. The type is string[64]. The value of **userId** must be unique in the app. The value of **userId** can contain letters, digits, hyphens (-), and underscores (_).
  - **userName**: (optional) user nickname. The type is string[128].

- **How do I obtain microphoneId and cameraId? Why are they mandatory?**

  microphoneId and **cameraId** are mandatory for audio and video calls. The audio and video corresponding to the IDs are captured when a stream is created.

  You can call the **getDevices**, **getCameras**, and **getMicrophones** APIs to obtain the media input and output, microphone, and camera IDs.

- **If the camera is not turned on, will the default profile picture be displayed?**

  If the camera cannot be obtained but the microphone can be obtained, a black screen is displayed and audio streams exist. If neither the microphone nor the camera can be obtained, the local preview fails and the default profile picture is not displayed.

- **Do I need to release the camera after I exit a room?**

  You do not need to manually release the camera when you exit the room. The camera is automatically disabled and no longer captures data.

- **How and when do I perform authentication?**

  For details, see **Access Authentication**.

- **What do I do when the client fails to join the room as a joiner?**

  The role parameter transferred when the client calls **join** is incorrect. The value of role is a number. If the client transfers a string, room joining will fail.

- **What do I do if the local stream fails to be created and the message "Cannot read property 'getUserMedia' of undefined" is displayed on the console, indicating that the media source cannot be obtained?**

  The possible causes are as follows:

  Cause 1: The system does not allow the app to access media sources such as the camera. For details about how to solve the problem, see **Granting the Permissions of Accessing Cameras or Microphones to a Browser**.

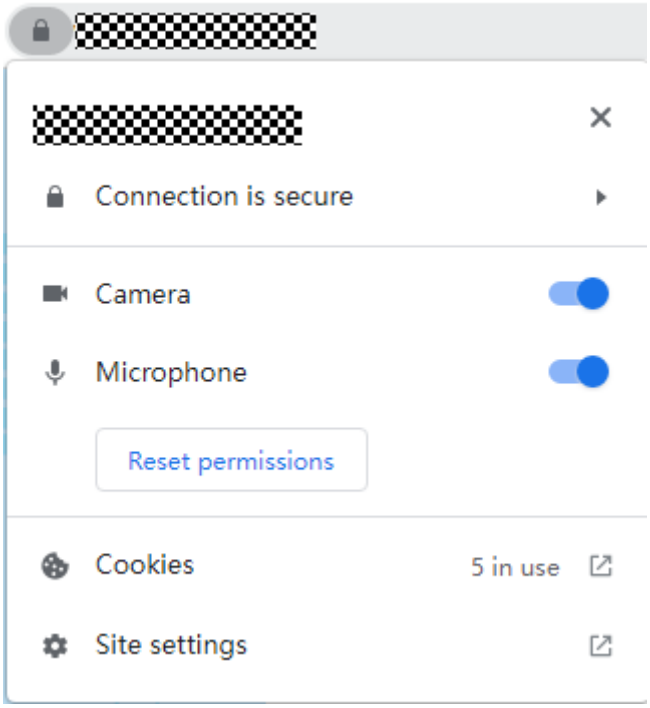
  Cause 2: According to the browser rules, the camera and microphone permissions can be accessed only in https:// or localhost mode.

  Cause 3: The camera or other devices are occupied by other applications.

- **If I have visited an app website developed using the Web SDK and deleted the permissions of the website, the camera and microphone may fail to be enabled. What do I do?**

  Ensure that the camera/microphone access permission is granted to the browser. For details, see **Granting the Permissions of Accessing Cameras or**

Microphones to a Browser. Click the icon in the upper left corner of the page and grant the permissions.



- **I enter an online audio URL that can be opened in a browser. However, audio mixing fails to be enabled when I use the audio mixing function of the Web SDK. What do I do?**

  Check whether the online audio file download server supports cross-domain access. Due to the security policy of the browser, cross-domain access must be supported. Otherwise, the request fails.

  Only the remote end can hear the mixed audio.

- **What is the top-N-audio mode (top three loudest participants)?**

  The top-N-audio mode is also called the top three loudest participant mode. After the top-N-audio mode is enabled, a user does not need to call an API to subscribe to the audio of a remote user. The local user can receive the audio of the three users who speaks loudest in the current room. For details about the API, see **Switching the Audio Mode**.

- Is the **setVolume4TopThree** API used to configure the volume of the top three loudest participants in the room? How many parameters should I pass?

  Yes. Only one parameter needs to be passed.

- **Can the SparkRTC Web SDK be integrated if the service app can use only the HTTP protocol?**

  It can be integrated but is not recommended. You need to manually disable the security policy function. Open a tab on the Chrome, enter **chrome://flags/#unsafely-treat-insecure-origin-as-secure**, enable the function, and add the app loading address to the ignore list.

- **What do I do if I fail to join a room using the Web SDK in Firefox?**

  Check whether the H.264 plug-in of the Firefox browser is installed. Enter **about:addons** in the address box of the browser. The plug-in installation page is displayed. Check whether the H.264 plug-in is installed. If the plug-in is not installed, install it on the page.

- **What do I do if screen sharing using macOS Chrome fails and "NotAllowedError: Permission denied by system" or "NotReadableError: Could not start video source" is displayed?**

  The possible cause is that the screen recording permission of the browser is not enabled. On your computer, go to **Settings** > **Security & Privacy** > **Privacy** > **Screen Recording**, enable Chrome screen recording authorization, and restart Chrome.

3.7 Change History

Table 3-15 Change history

Date	Change Description
2022-09-30	This issue is the first official release.

4 Access Authentication

To ensure the communication security of SparkRTC, access authentication is required when a user joins a room. This chapter describes the implementation principles of SparkRTC access authentication and how to generate an authentication signature.

Authentication Principles

Huawei Cloud SparkRTC uses digital signatures for access authentication. You need to set **signature** and **ctime** when joining a room using the SDK. **signature** is generated by the tenant using **app_key** and **app_id** provided by SparkRTC and **room_id**, **user_id**, and **ctime** based on the **signature generation example** of SparkRTC. For details, see **Table 4-1**.

```
signature = hmacSha256(app_key,(app_id + room_id + user_id + ctime))
```

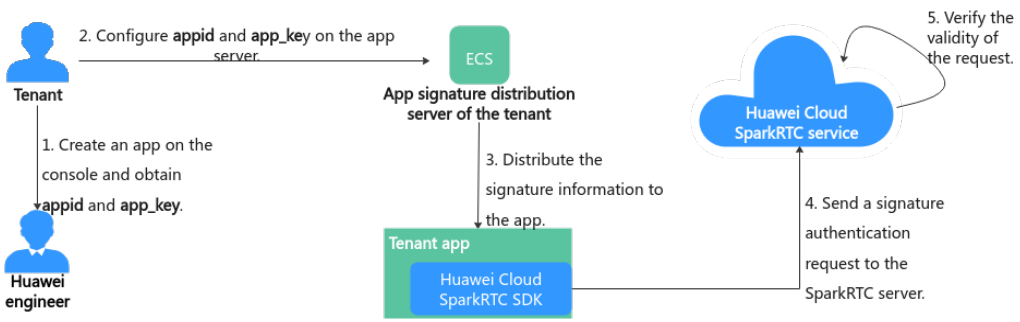
Table 4-1 Parameters

Parameter	Description
app_key	An authentication key is generated by SparkRTC for each app. The key must be securely stored. For details about how to obtain **app_key**, see **How Do I Obtain a Key?**
app_id	Application ID generated by SparkRTC. Obtain the value of **app_id** from **Applications** on the SparkRTC console.
room_id	ID of the room created by the tenant.
user_id	User ID used by the tenant to access SparkRTC.

Parameter	Description
ctime	Time when the signature authentication expires. The value is the current UTC time (UNIX timestamp) of the system plus the authentication expiration time (recommended: 2 hours; maximum: < 12 hours). The unit is second. **NOTE** The value of **ctime** is the creation time plus the expiration time. For example, if the current time is 9:00 and the authentication expiration time is 30 minutes, the value of **ctime** is 9:30. That is, the signature becomes invalid after 9:30.

Tenants are advised to build their own app signature distribution servers to prevent the **app_key** from being leaked to the app. **Figure 4-1** shows the authentication principles.

Figure 4-1 Authentication principles



Signature Generation Method

To generate a signature, perform the following steps:

1. Combine **app_id**, **room_id**, **user_id** , and **ctime** into a string.

```
long ctime = System.currentTimeMillis() / 1000 + 60 * 60;   // The validity period is 1 hour, in seconds.
String content = app_id + "+" + room_id + "+" + user_id + "+" + ctime;
```

2. Use **app_key** to encrypt the string **content** in HMAC-SHA256 mode to obtain the signature string.

```
String signatureStr = hmacSha(appKey, content, "HmacSHA256");
static String hmacSha(String KEY, String VALUE, String SHA_TYPE) {
    try {
        SecretKeySpec signingKey = new SecretKeySpec(KEY.getBytes("UTF-8"), SHA_TYPE);
        Mac mac = Mac.getInstance(SHA_TYPE);
        mac.init(signingKey);
        byte[] rawHmac = mac.doFinal(VALUE.getBytes("UTF-8"));

        byte[] hexArray = {
                (byte) '0', (byte) '1', (byte) '2', (byte) '3',
                (byte) '4', (byte) '5', (byte) '6', (byte) '7',
                (byte) '8', (byte) '9', (byte) 'a', (byte) 'b',
                (byte) 'c', (byte) 'd', (byte) 'e', (byte) 'f'
        };
        byte[] hexChars = new byte[rawHmac.length * 2];
        for (int j = 0; j < rawHmac.length; j++) {
            int v = rawHmac[j] & 0xFF;
            hexChars[j * 2] = hexArray[v >>> 4];
            hexChars[j * 2 + 1] = hexArray[v & 0x0F];
```



```
            }
            return new String(hexChars);
        } catch (Exception ex) {
            throw new RuntimeException(ex);
        }
    }
```

Signature Generation Example

To prevent the **app_key** from being leaked, you are advised to configure your own app signature distribution server. After **app_id**, **room_id**, **user_id**, and **ctime** are passed to the server, the server returns the signature. The following is sample code:

```
package com.xxx.xxx.utils;

import androidx.annotation.NonNull;

import com.alibaba.fastjson.JSON;
import com.huawei.rtcdemo.Constants;

import java.io.IOException;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import okhttp3.Response;

public class SignatureUtil {
    private static final String TAG = "SignatureUtil";

    public interface onSignatureSuccess {
        void onSuccess(String signature);
    }

    public static void getSignature(String appid, String roomid, String userid, long ctime, String signatureKey,
onSignatureSuccess callback) {
        if (Constants.RTC_SIGNATURE_USE_LOCAL) {
            getSignatureLocal(appid, roomid, userid, ctime, signatureKey, callback);
        } else {
            getSignatureRemote(appid, roomid, userid, ctime, callback);
        }
    }

    private static void getSignatureLocal(String appid, String roomid, String userid, long ctime, String
signatureKey, @NonNull onSignatureSuccess callback) {
        String content = appid + "+" + roomid + "+" + userid + "+" + ctime; // here "+" is real char in content.
        String signature = SignatureUtil.hmacSha256(signatureKey, content);
        callback.onSuccess(signature);
    }

    private static void getSignatureRemote(String appid, String roomid, String userid, long ctime, @NonNull
onSignatureSuccess callback) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                HttpUtil httpUtil = new HttpUtil();
                // Constants.RTC_SIGNATURE_URL indicates the distribution server address with the user's
application signature.
                String url = Constants.RTC_SIGNATURE_URL + "?appid=" + appid + "&roomid=" + roomid +
"&userid=" + userid + "&ctime=" + ctime;
                Response response = httpUtil.sendGetMethodWithHead(url, "X-AUTH-TOKEN",
Constants.RTC_AUTH_TOKEN);
                if (response == null) {
                    return;
                }

                if (response.isSuccessful()) {
```



```
                try {
                    String json = response.body().string();
                    String signature = JSON.parseObject(json).getString("signature");
                    callback.onSuccess(signature);
                } catch (IOException e) {
                    LogUtil.e(TAG, "getSignature failed:" + e.getMessage());
                }
            } else {
                LogUtil.e(TAG, "getSignature failed!");
            }
        }
    }).start();
}

public static String hmacSha256(String KEY, String VALUE) {
    return hmacSha(KEY, VALUE, "HmacSHA256");
}

private static String hmacSha(String KEY, String VALUE, String SHA_TYPE) {
    try {
        SecretKeySpec signingKey = new SecretKeySpec(KEY.getBytes("UTF-8"), SHA_TYPE);
        Mac mac = Mac.getInstance(SHA_TYPE);
        mac.init(signingKey);
        byte[] rawHmac = mac.doFinal(VALUE.getBytes("UTF-8"));

        byte[] hexArray = {
                (byte) '0', (byte) '1', (byte) '2', (byte) '3',
                (byte) '4', (byte) '5', (byte) '6', (byte) '7',
                (byte) '8', (byte) '9', (byte) 'a', (byte) 'b',
                (byte) 'c', (byte) 'd', (byte) 'e', (byte) 'f'
        };
        byte[] hexChars = new byte[rawHmac.length * 2];
        for (int j = 0; j < rawHmac.length; j++) {
            int v = rawHmac[j] & 0xFF;
            hexChars[j * 2] = hexArray[v >>> 4];
            hexChars[j * 2 + 1] = hexArray[v & 0x0F];
        }
        return new String(hexChars);
    } catch (Exception ex) {
        throw new RuntimeException(ex);
    }
}
}
```


5 Appendix

GRS Country Codes

5.1 GRS Country Codes

DR1: Chinese mainland

Country/Region	Code
Chinese mainland	CN

DR2: Asia, Africa, and Latin America (Singapore Site)

Country/Region	Code
The United Arab Emirates	AE
Afghanistan	AF
Antigua and Barbuda	AG
Anguilla	AI
Armenia	AM
Angola	AO
Antarctica	AQ
Argentina	AR
American Samoa	AS
Aruba	AW
Azerbaijan	AZ
Barbados	BB

Country/Region	Code
Bangladesh	BD
Burkina Faso	BF
Bahrain	BH
Burundi	BI
Benin	BJ
Saint Barthélemy	BL
Bermuda	BM
Brunei	BN
Bolivia	BO
Brazil	BR
Bahamas	BS
Bhutan	BT
Bouvet Island	BV
Botswana	BW
Belize	BZ
Cocos (Keeling) Islands	CC
Congo - Kinshasa	CD
Central African Republic	CF
Congo - Brazzaville	CG
Côte d'Ivoire	CI
Cook Islands	CK
Chile	CL
Cameroon	CM
Colombia	CO
Costa Rica	CR
Cape Verde	CV
Christmas Island	CX
Djibouti	DJ
Dominica	DM
Dominican Republic	DO

Country/Region	Code
Algeria	DZ
Ecuador	EC
Egypt	EG
Western Sahara	EH
Eritrea	ER
Ethiopia	ET
Fiji	FJ
Falkland Islands (Islas Malvinas)	FK
Micronesia	FM
Gabon	GA
Grenada	GD
Georgia	GE
French Guiana	GF
Ghana	GH
Gambia	GM
Guinea	GN
Guadeloupe	GP
Equatorial Guinea	GQ
South Georgia and the South Sandwich Islands	GS
Guatemala	GT
Guam	GU
Guinea-Bissau	GW
Guyana	GY
Hong Kong SAR, China	HK
Heard Island and McDonald Islands	HM
Honduras	HN
Haiti	HT
Indonesia	ID
India	IN

Country/Region	Code
British Indian Ocean Territory	IO
Iraq	IQ
Jamaica	JM
Jordan	JO
Kenya	KE
Kyrgyzstan	KG
Cambodia	KH
Kiribati	KI
Comoros	KM
Saint Kitts and Nevis	KN
Kuwait	KW
Cayman Islands	KY
Kazakhstan	KZ
Laos	LA
Lebanon	LB
Saint Lucia	LC
Sri Lanka	LK
Liberia	LR
Lesotho	LS
Libya	LY
Morocco	MA
Madagascar	MG
Marshall Islands	MH
Mali	ML
Myanmar	MM
Mongolia	MN
Macao SAR, China	MO
Northern Mariana Islands	MP
Martinique	MQ
Mauritania	MR

Country/Region	Code
Montserrat	MS
Mauritius	MU
Maldives	MV
Malawi	MW
Mexico	MX
Malaysia	MY
Mozambique	MZ
Namibia	NA
New Caledonia	NC
Niger	NE
Norfolk Island	NF
Nigeria	NG
Nicaragua	NI
Nepal	NP
Nauru	NR
Niue	NU
Oman	OM
Panama	PA
Peru	PE
French Polynesia	PF
Papua New Guinea	PG
Philippines	PH
Pakistan	PK
Pitcairn Islands	PN
Puerto Rico	PR
Palestine, State of	PS
Palau	PW
Paraguay	PY
Qatar	QA
Réunion	RE

Country/Region	Code
Rwanda	RW
Saudi Arabia	SA
Solomon Islands	SB
Seychelles	SC
Singapore	SG
Saint Helena	SH
Sierra Leone	SL
Senegal	SN
Somalia	SO
Suriname	SR
South Sudan	SS
São Tomé and Príncipe	ST
El Salvador	SV
Eswatini	SZ
Turks and Caicos Islands	TC
Chad	TD
French Southern Territories	TF
Togo	TG
Thailand	TH
Tajikistan	TJ
Tokelau	TK
Timor-Leste	TL
Turkmenistan	TM
Tunisia	TN
Tonga	TO
Trinidad and Tobago	TT
Tuvalu	TV
Taiwan, China	TW
Tanzania	TZ
Uganda	UG

Country/Region	Code
Uruguay	UY
Uzbekistan	UZ
Venezuela	VE
British Virgin Islands	VG
United States Virgin Islands	VI
Vietnam	VN
Vanuatu	VU
Wallis and Futuna	WF
Samoa	WS
Yemen	YE
Mayotte	YT
South Africa	ZA
Zambia	ZM
Zimbabwe	ZW
Cuba	CU
North Korea	KP
Sudan	SD
Syria	SY
Japan	JP
South Korea	KR

DR3: Europe (Germany Site)

Country/Region	Code
Andorra	AD
Albania	AL
Austria	AT
Åland Islands	AX
Bosnia and Herzegovina	BA
Belgium	BE

Country/Region	Code
Bulgaria	BG
Canada	CA
Switzerland	CH
Cyprus	CY
Czech Republic	CZ
Germany	DE
Denmark	DK
Estonia	EE
Spain	ES
Finland	FI
Faroe Islands	FO
France	FR
United Kingdom	GB
Guernsey	GG
Gibraltar	GI
Greenland	GL
Greece	GR
Croatia	HR
Hungary	HU
Ireland	IE
Israel	IL
Isle of Man	IM
Iceland	IS
Italy	IT
Jersey	JE
Liechtenstein	LI
Lithuania	LT
Luxembourg	LU
Latvia	LV
Monaco	MC

Country/Region	Code
Moldova	MD
Montenegro	ME
Saint Martin	MF
Republic of North Macedonia	MK
Malta	MT
Netherlands	NL
Norway	NO
Poland	PL
Saint Pierre and Miquelon	PM
Portugal	PT
Romania	RO
Serbia	RS
Sweden	SE
Slovenia	SI
Svalbard and Jan Mayen	SJ
Slovakia	SK
San Marino	SM
Sint Maarten	SX
Turkey	TR
United States Minor Outlying Islands	UM
United States	US
Vatican City	VA
Saint Vincent and the Grenadines	VC
Kosovo	XK->YK
Australia	AU
New Zealand	NZ
Netherlands Antilles	BQ->AN
Ukraine	UA
Curaçao	CW

