SparkRTC

Best Practices

Issue	01
Date	2023-11-01

HUAWEI TECHNOLOGIES CO., LTD.

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 Cloud Recording and Playback

1.1 Overview

The cloud recording and playback functions of Huawei Cloud SparkRTC allow you to record and store audio/video calls or live video. Cloud recording supports two modes, as shown in **Table 1-1**.

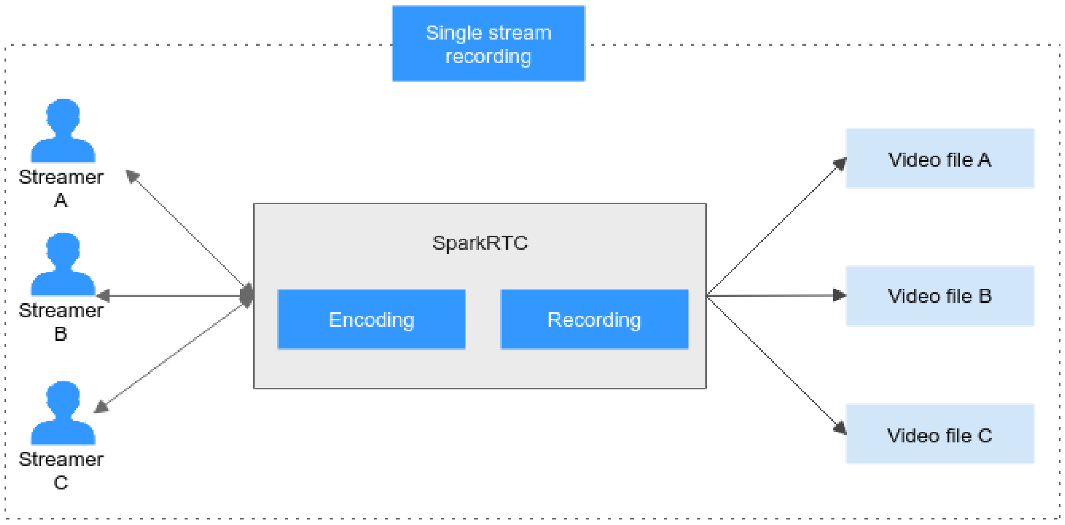
Table 1-1 Cloud recording modes

Mode	Description	Application Scenario
Single stream recording	● Single stream recording by subscriber is supported. ● After automatic recording is enabled, each stream in the room is recorded. ● Media types for recording can be specified. The options are audio only, video only, and audio/video. ● MP4 and HLS files can be recorded, and AAC audio files can be encoded. ● The playback URL in the callback message can be obtained for playback. The **downloadurl** field in the callback message is the OBS playback URL. If the URL is used for playback, the corresponding download traffic or bandwidth fee will be generated in OBS. ● Users can select camera streams or screen sharing streams. ● Resolution can be specified.	Online class and content review
Mixed stream recording	● Multi-channel video recording or multi-channel audio and video recording are supported. ● Media types for recording can be specified. The options are audio only, video only, and audio/video. ● MP4 and HLS files can be recorded, and AAC audio files can be encoded. ● Audio/video attributes can be configured, such as the bitrate, resolution, and frame rate. ● The playback URL in the callback message can be obtained for playback. The **downloadurl** field in the callback message is the OBS playback URL. If the URL is used for playback, the corresponding download traffic or bandwidth fee will be generated in OBS.	Co-hosting

1.2 Single Stream Recording

Scenario

The audio/video streams of each user in a room can be recorded and saved in independent files.



Working Principles

The **single stream recording** provided by SparkRTC can be enabled automatically, that is, **automatic single stream recording**. The following figure shows how the function is implemented.

Figure 1-1 Single stream recording

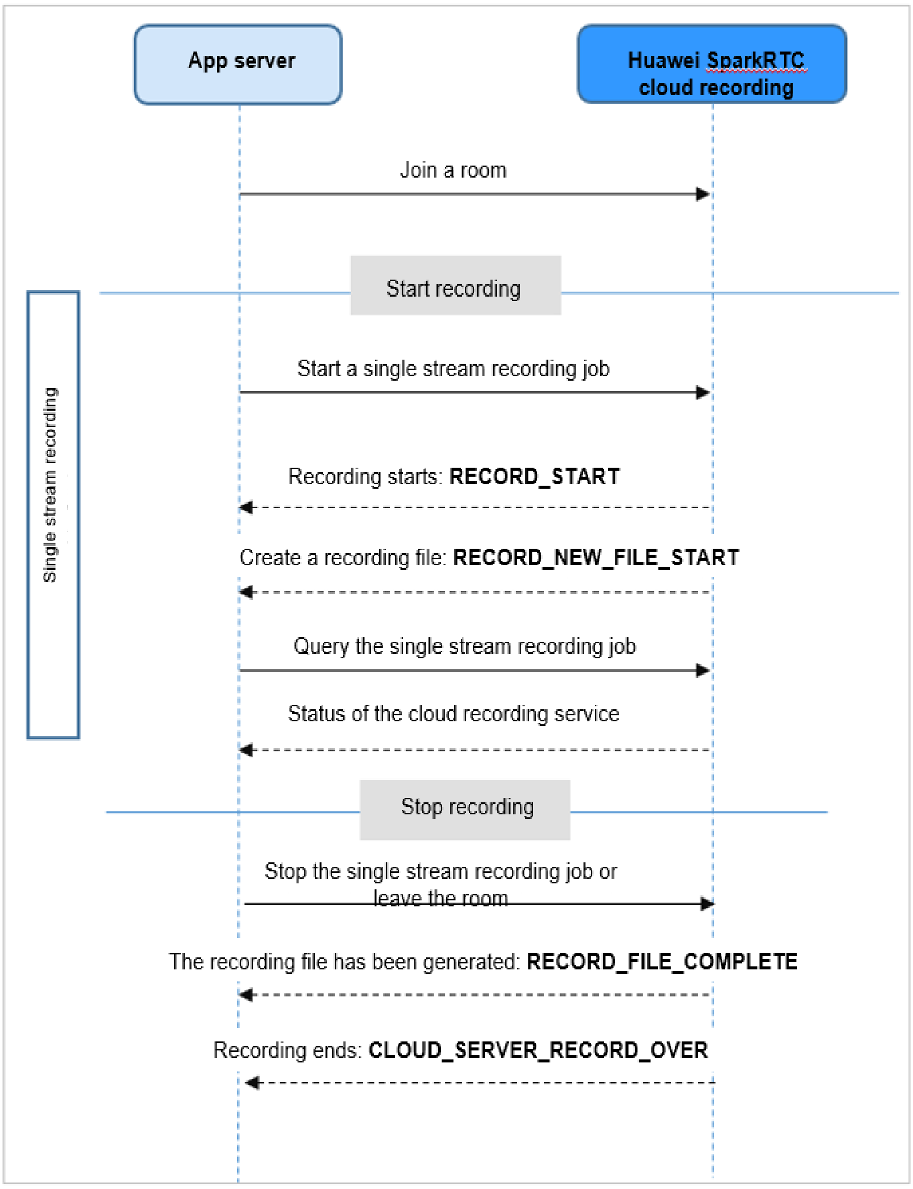
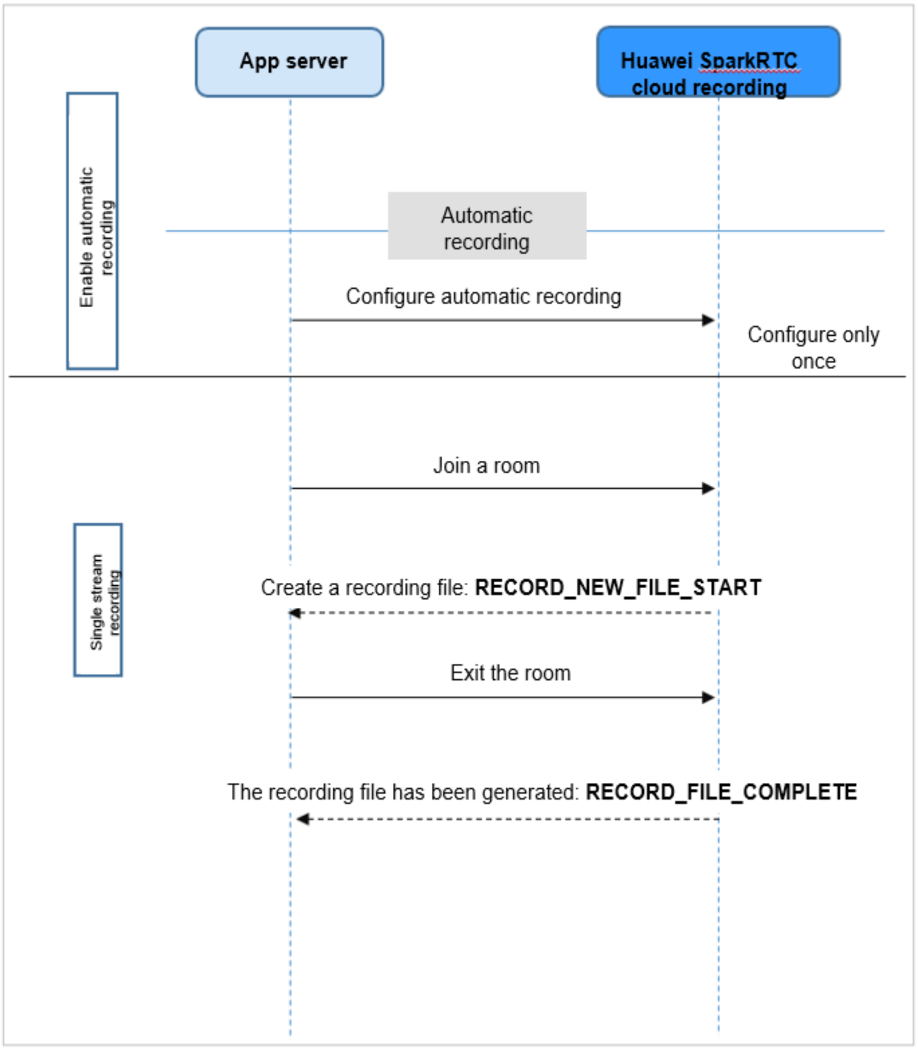
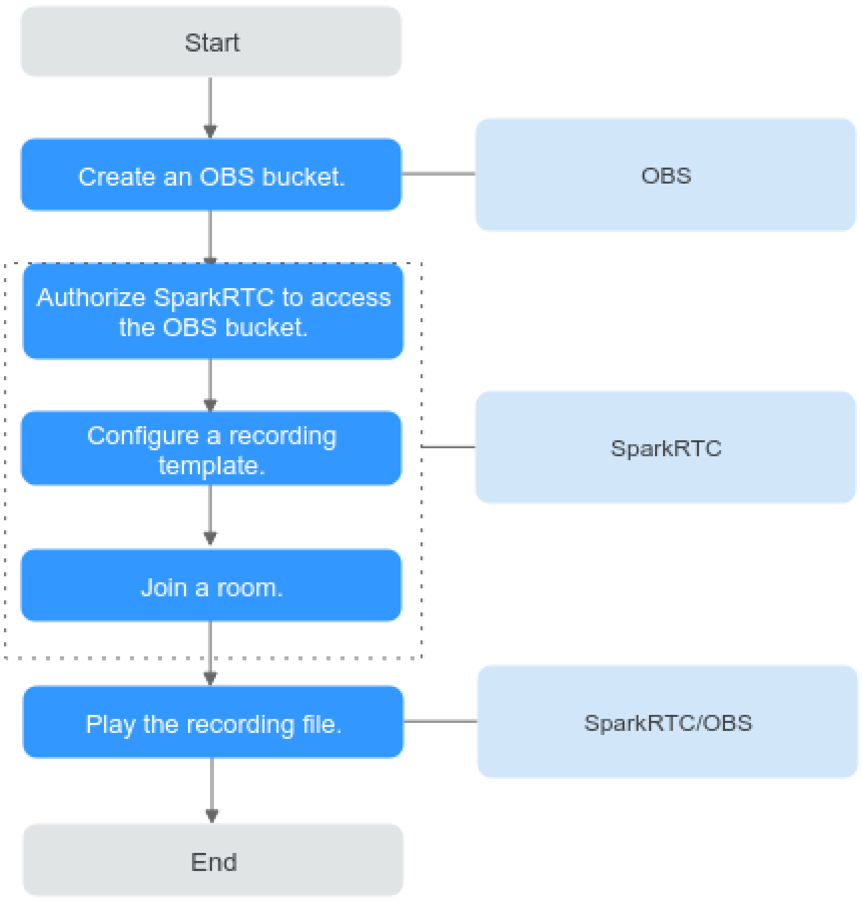

App server
Huawei SparkRTC cloud recording
Join a room
Start recording
Single stream recording
Start a single stream recording job
Recording starts: RECORD_START
Create a recording file: RECORD_NEW_FILE_START
Query the single stream recording job
Status of the cloud recording service
Stop recording
Stop the single stream recording job or leave the room
The recording file has been generated: RECORD_FILE_COMPLETE
Recording ends: CLOUD_SERVER_RECORD_OVER

Figure 1-2 Automatic single stream recording


App server
Huawei SparkRTC cloud recording
Enable automatic recording
Automatic recording
Configure automatic recording
Configure only once
Join a room
Single stream recording
Create a recording file: RECORD_NEW_FILE_START
Exit the room
The recording file has been generated: RECORD_FILE_COMPLETE

Implementation



1. **Create an OBS bucket** for storing recordings. If you already have one, go to **2**.

   NOTE

   The reliability of a single-AZ bucket is lower than that of a multi-AZ bucket. To prevent recording failures caused by OBS exceptions, you are advised to create multi-AZ buckets for storing recordings.

2. **Authorize SparkRTC to store recordings in the bucket**.
3. **Configure recording rules** for SparkRTC and enable automatic recording. After you join a room, the recording rule with the same recording rule ID as that in the application automatically takes effect. The recordings are stored in OBS based on the recording settings. You can set a callback URL to obtain notifications about the recording status.
4. **Join a room**: After configuring recording rules, you can use the SparkRTC app to join a room for real-time interaction. SparkRTC will record the ongoing audio/video stream based on the configured recording rules.

   NOTE

   If automatic recording is not enabled during recording rule configuration, you need to call the **SparkRTC API** to start, query, and control a cloud recording task after joining a room. Then SparkRTC can record images based on the recording rule ID in the API calling request.

5. **View recordings**. After the recording is complete, you can receive the callback message of the recording task in the configured callback URL. You can obtain the basic information about the recordings and manage the recordings in OBS, such as downloading, sharing, and deleting them.

   NOTE

   The resolution of recordings is the same as that of the pushed streams.

Procedure

Step 1 Create a bucket by referring to **OBS Help Center**. If you have one, go to **2**.

NOTE

The OBS bucket to be created must be located in the CN North-Beijing4 region.

Step 2 Authorize access to OBS.

1. Log in to the SparkRTC console.
2. In the navigation pane, choose **OBS Authorization**.
3. In the row of the target OBS bucket, click **Authorize**.

Step 3 Configure a recording rule.

1. Log in to the SparkRTC console.
2. In the navigation pane, choose **Apps**.
3. Click **Configure** in the row of the application for which you want to create a recording rule.
4. On the **Recording Rule** tab page, click **Add**. The **Add Recording Rule** page is displayed.

   NOTE

   Only one recording rule can be created for an application ID.

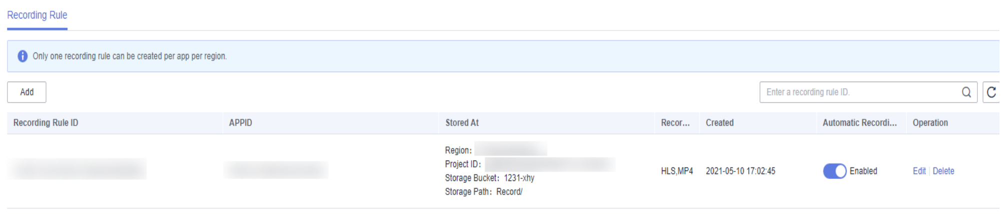
5. Configure recording parameters according to **Table 1-2**.

Table 1-2 Recording parameters

Parameter	Description
Storage Bucket	OBS bucket where the recording is stored. Currently, recorded files can be stored only in OBS buckets of **CN North-Beijing4**.
Region	Region where the OBS bucket is deployed.
Storage Path	Path of the OBS bucket where the recording is stored.
Record As	Recording format, which can be HLS and MP4.

Parameter		Description
HLS	File Naming	Storage path and filename prefix of M3U8 files. Default format: `{app_id}/{record_format}/{stream}_{file_start_time}/{stream}_{file_start_time}` The meanings of the preceding variables are as follows: – *app_id*: app ID – *record_format*: recording format – *stream*: stream name – *file_start_time*: file generation time
	Recording Length	The recording length can be 0 or 1 to 720 minutes (12 hours). If a stream has been recorded for more than 12 hours, another file will be created based on the naming rule. If the recording length is **0**, the entire stream is recorded as a file.
	Max Stream Pause Length	Values: – **Generate a new file when a stream is paused** – **Do not generate a new file when a stream is paused**: The maximum stream pause length is 30 days. – **Other**: If the stream pause length is within the specified range, the system does not generate a new file. Otherwise, a new file is generated.
MP4	File Naming	Storage path and filename prefix of MP4 files. Default format: `{app_id}/{record_format}/{stream}_{file_start_time}/{stream}_{file_start_time}` The meanings of the preceding variables are as follows: – *app_id*: app ID – *record_format*: recording format – *stream*: stream name – *file_start_time*: file generation time
	Recording Length	The recording length ranges from 1 to 180 minutes (3 hours). If a stream has been recorded for more than 3 hours, another file will be created based on the naming rule.
	Max Stream Pause Length	Values: – **Generate a new file when a stream is paused** – **Other**: If the stream pause length is within the specified range, the system does not generate a new file. Otherwise, a new file is generated.

6. Click **OK**.



Recording Rule

Only one recording rule can be created per app per region.

Add | Enter a recording rule ID.

Recording Rule ID	APPID	Stored At	Recor...	Created	Automatic Recordi...	Operation
[redacted]	[redacted]	Region: [redacted] Project ID: [redacted] Storage Bucket: 1231-xhy Storage Path: Record/	HLS,MP4	2021-05-10 17:02:45	Enabled	Edit | Delete

7. In the recording rule list, enable automatic recording as required. When automatic recording is enabled, if a new room is created under the app, the system automatically performs single stream recording during real-time communication in the room based on the configured recording rule.

   NOTE

   Automatic recording takes effect only for rooms created under the app after you enable it.

Step 4 Join a room.

After configuring recording rules, you can use the SparkRTC app to join a room for real-time interaction. SparkRTC will record the ongoing audio/video stream based on the configured recording rules.

NOTE

If automatic recording is not enabled during recording rule configuration, you need to call the **SparkRTC API** to start a cloud recording task after joining a room. Then SparkRTC can record images based on the recording rule ID in the API calling request.

Step 5 Replay recordings.

When the recording is complete, view recordings on the OBS console or through callback messages.

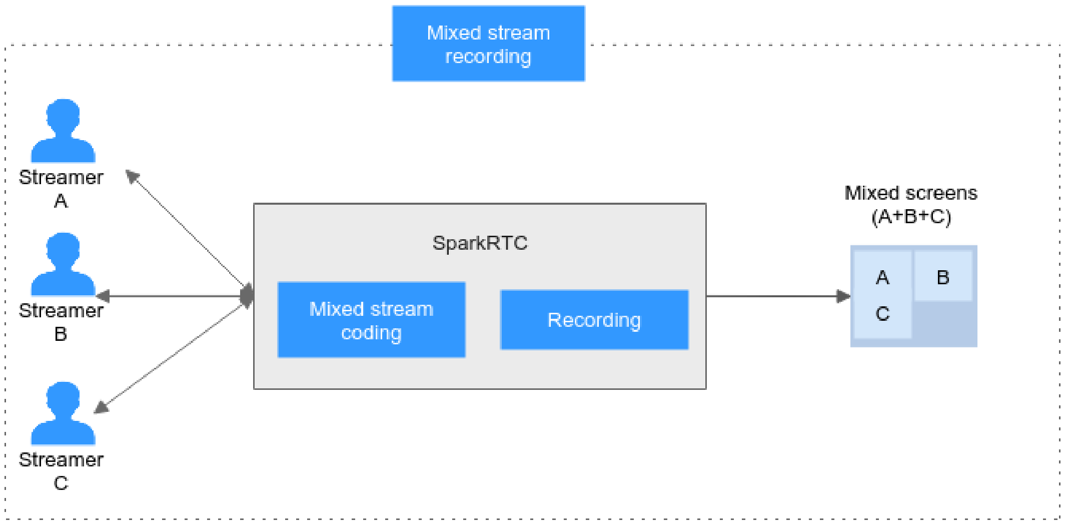
- Viewing recordings on the OBS console
  a. In the navigation pane of the OBS console, choose **Object Storage**.
  b. In the bucket list, click the bucket that stores SparkRTC recordings.
     The **Overview** page is displayed.
  c. In the navigation pane, choose **Objects** to view the recording information.
  d. Download and share the recordings as required. For details, see **OBS Documentation**.

----End

1.3 Mixed Stream Recording

Scenario

Audio/Video streams in a room can be mixed in the cloud, and then the mixed audio/video streams can be recorded and saved in one file.

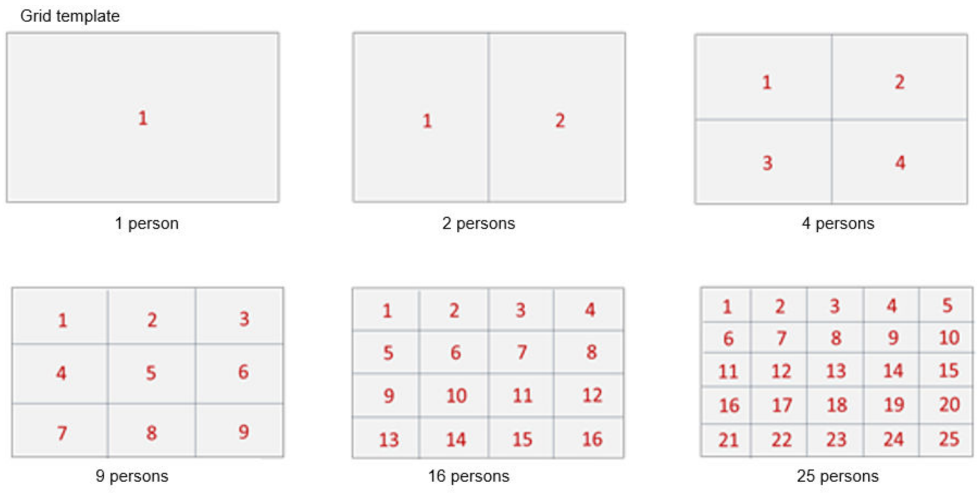


Layout

The mixed stream recording supports two preset layouts: grid template and screen sharing template (main window on the left or main window on the right). You can also customize the mixed stream layout (video pane position).

- **Grid template**

  Screens of each user are tiled and have the same size. The size and position of each screen are adjusted according to the number of users. A maximum of 25 screens are supported. The following figure shows the layouts with different user quantities.

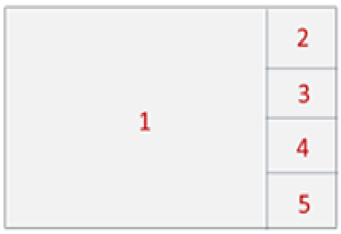


  - If the aspect ratio of the actual video stream is different from that of the screen, the video image will be cropped to adapt to the screen size.
  - If a user exits the room, the image of this user will be replaced by the next user who enters the room.
  - If there are not enough users in the room, the rest of the screen displays a background color.
  - A user who only sends audio still occupies a screen.
  - A background image is supported. If there are not enough users in the room, the rest of the screen displays a background image.

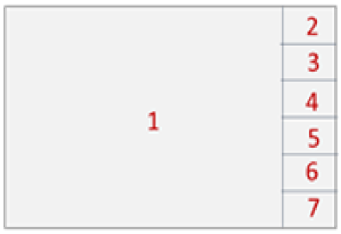
- **Screen sharing template**

  A shared screen or a presenter view is displayed in the main screen on the right or left. Other users are sorted out vertically on the side of the main screen. A maximum of 17 screens are supported. The following figure shows the layouts with different user quantities.

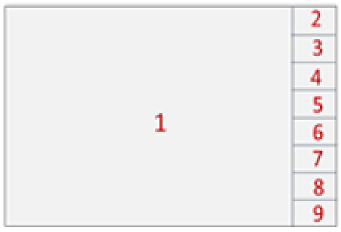
Screen sharing template: main screen on the left (**screen_share_left**)



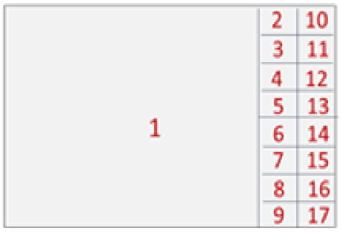
5 persons



7 persons

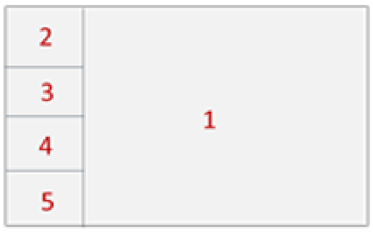


9 persons

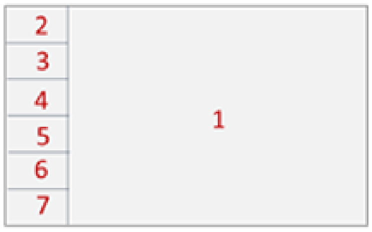


17 persons

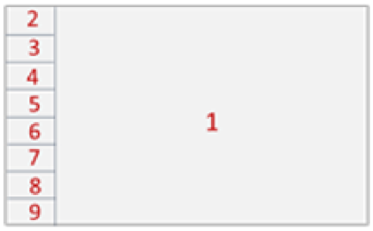
Screen sharing template: main screen on the right (**screen_share_right**)



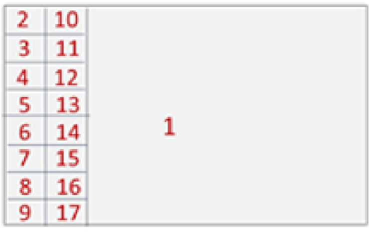
5 persons



7 persons



9 persons



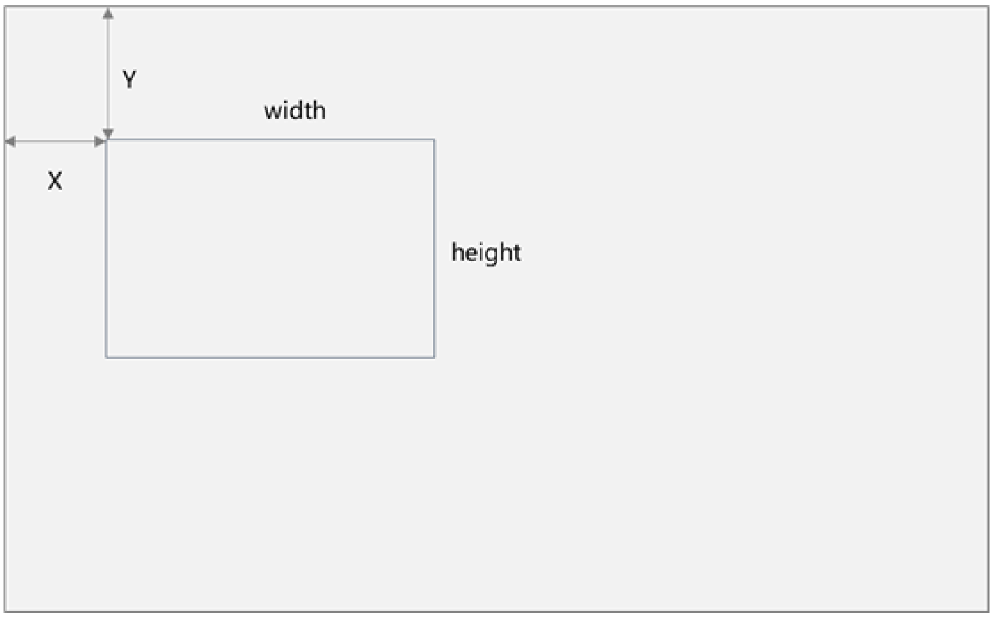
17 persons

  – The main screen displays a presenter view or a shared screen.
  – The main screen displays the video of a specific user. If no user is specified or the specified user does not enter the room, the main screen displays the background color.

- Images in the main screen on the left are zoomed in or out to display the complete content. Images in the small screens on the right are cropped to adapt to the screen size.
- The small screens of users on the right are sorted by the time when they enter the room.
- If a user in the right small screen exits the room, the image of this user will be replaced by the next user who enters the room.
- If there are not enough users in the room, the rest of the screen displays a background color.
- A user who only sends audio still occupies a screen.
- A background image is supported. If there are not enough users in the room, the rest of the screen displays a background image.

● **Custom layout template**

You can customize the mixed stream layout by setting the screen size and specifying the relative position of the screen on the video canvas.



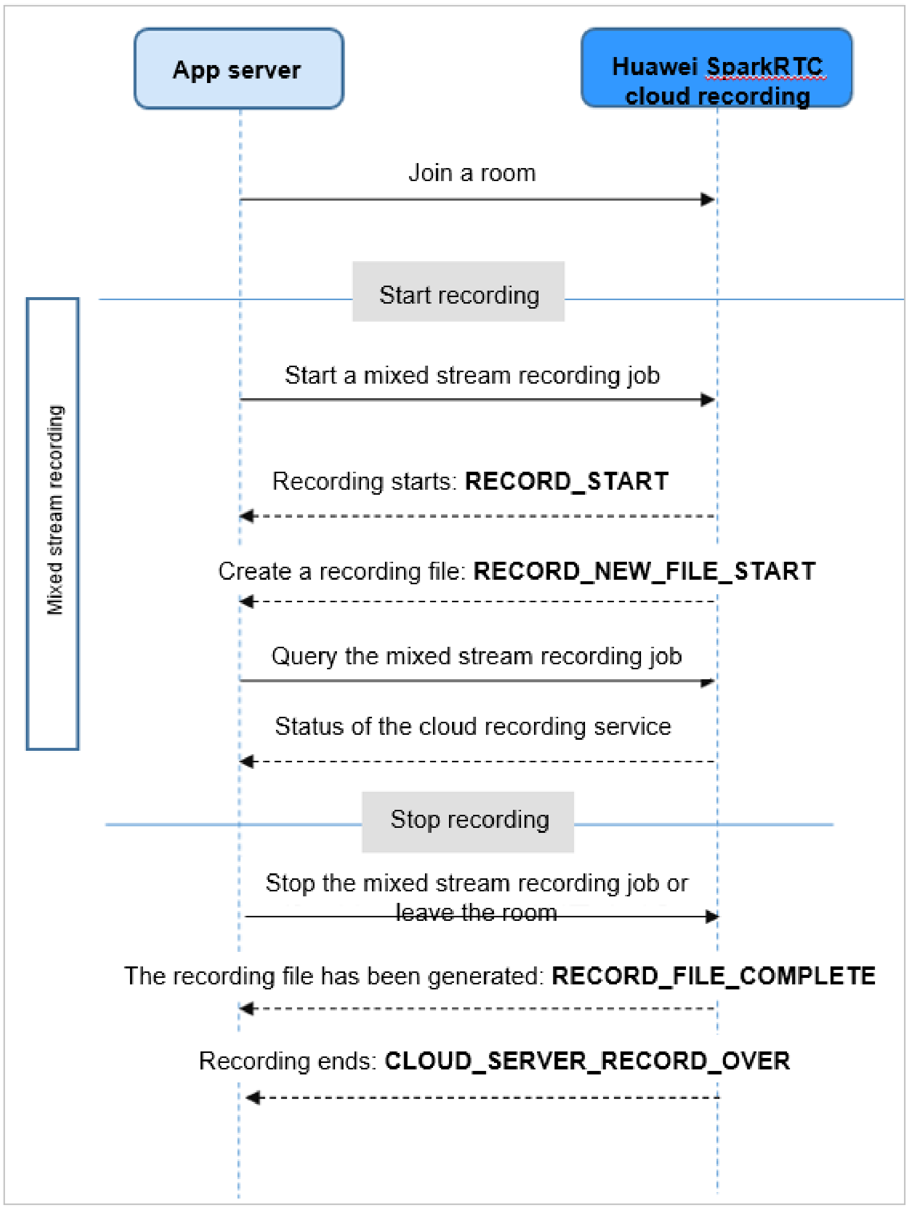
- You can customize the position of each video screen on the canvas.
- You can customize the width and height of each video screen.
- For each screen, you can specify a user in the room to be displayed using **user_id**.
- You can choose to display camera streams or screen sharing streams in a screen.
- If the aspect ratio of the actual video stream is different from that of the window, the video image will be cropped or zoomed in or out.
- If there are not enough users in the room, the rest of the screen displays a background color.
- A user who only sends audio still occupies a screen.
- A background image is supported. If there are not enough users in the room, the rest of the screen displays a background image.

Working Principles

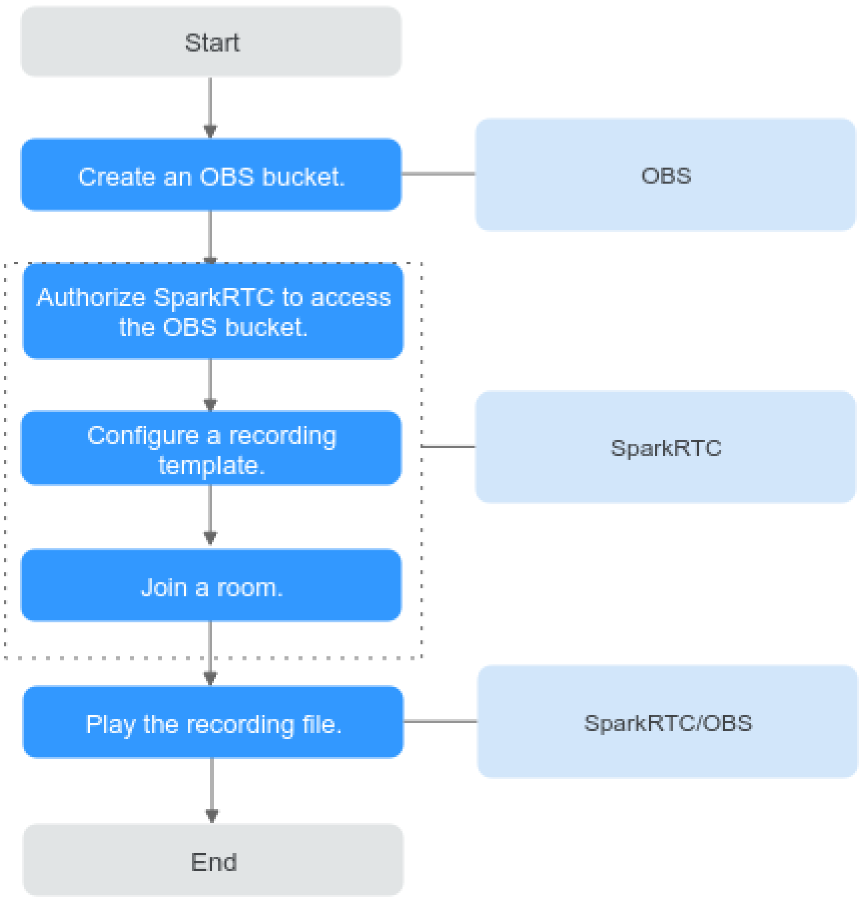
Figure 1-3 shows how SparkRTC implements **mixed stream recording**.

Note: When starting a mixed stream recording task, you need to specify **MixParam** and **layout_template**.

Figure 1-3 Mixed stream recording


App server
Huawei SparkRTC cloud recording
Join a room
Start recording
Mixed stream recording
Start a mixed stream recording job
Recording starts: RECORD_START
Create a recording file: RECORD_NEW_FILE_START
Query the mixed stream recording job
Status of the cloud recording service
Stop recording
Stop the mixed stream recording job or leave the room
The recording file has been generated: RECORD_FILE_COMPLETE
Recording ends: CLOUD_SERVER_RECORD_OVER

Implementation



1. **Create an OBS bucket** for storing recordings. If you already have one, go to **2**.

   NOTE

   The reliability of a single-AZ bucket is lower than that of a multi-AZ bucket. To prevent recording failures caused by OBS exceptions, you are advised to create multi-AZ buckets for storing recordings.

2. **Authorize SparkRTC to store recordings in the bucket**.
3. **Configure recording rules** for SparkRTC and enable automatic recording. After you join a room, the recording rule with the same recording rule ID as that in the application automatically takes effect. The recordings are stored in OBS based on the recording settings. You can set a callback URL to obtain notifications about the recording status.
4. **Join a room**: After configuring recording rules, you can use the SparkRTC app to join a room for real-time interaction. SparkRTC will record the ongoing audio/video stream based on the configured recording rules.

   NOTE

   If automatic recording is not enabled during recording rule configuration, you need to call the **SparkRTC API** to start, query, and control a cloud recording task after joining a room. Then SparkRTC can record images based on the recording rule ID in the API calling request.

5. **View recordings**. After the recording is complete, you can receive the callback message of the recording task in the configured callback URL. You can obtain the basic information about the recordings and manage the recordings in OBS, such as downloading, sharing, and deleting them.

   NOTE

   The resolution of recordings is the same as that of the pushed streams.

Procedure

Step 1 Create a bucket by referring to **OBS Help Center**. If you have one, go to **2**.

NOTE

The OBS bucket to be created must be located in the CN North-Beijing4 region.

Step 2 Authorize access to OBS.

1. Log in to the SparkRTC console.
2. In the navigation pane, choose **OBS Authorization**.
3. In the row of the target OBS bucket, click **Authorize**.

Step 3 Configure a recording rule.

1. Log in to the SparkRTC console.
2. In the navigation pane, choose **Apps**.
3. Click **Configure** in the row of the application for which you want to create a recording rule.
4. On the **Recording Rule** tab page, click **Add**. The **Add Recording Rule** page is displayed.

   NOTE

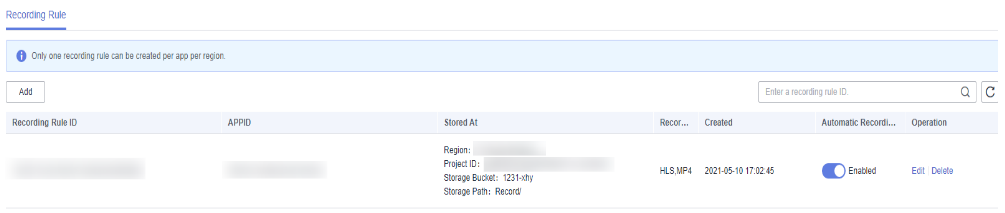
   Only one recording rule can be created for an application ID.
5. Configure recording parameters according to **Table 1-3**.

Table 1-3 Recording parameters

Parameter	Description
Storage Bucket	OBS bucket where the recording is stored. Currently, recorded files can be stored only in OBS buckets of **CN North-Beijing4**.
Region	Region where the OBS bucket is deployed.
Storage Path	Path of the OBS bucket where the recording is stored.
Record As	Recording format, which can be HLS and MP4.

Parameter		Description
HLS	File Naming	Storage path and filename prefix of M3U8 files. Default format: `{app_id}/{record_format}/{stream}_{file_start_time}/{stream}_{file_start_time}` The meanings of the preceding variables are as follows: – *app_id*: app ID – *record_format*: recording format – *stream*: stream name – *file_start_time*: file generation time
	Recording Length	The recording length can be 0 or 1 to 720 minutes (12 hours). If a stream has been recorded for more than 12 hours, another file will be created based on the naming rule. If the recording length is **0**, the entire stream is recorded as a file.
	Max Stream Pause Length	Values: – **Generate a new file when a stream is paused** – **Do not generate a new file when a stream is paused**: The maximum stream pause length is 30 days. – **Other**: If the stream pause length is within the specified range, the system does not generate a new file. Otherwise, a new file is generated.
MP4	File Naming	Storage path and filename prefix of MP4 files. Default format: `{app_id}/{record_format}/{stream}_{file_start_time}/{stream}_{file_start_time}` The meanings of the preceding variables are as follows: – *app_id*: app ID – *record_format*: recording format – *stream*: stream name – *file_start_time*: file generation time
	Recording Length	The recording length ranges from 1 to 180 minutes (3 hours). If a stream has been recorded for more than 3 hours, another file will be created based on the naming rule.
	Max Stream Pause Length	Values: – **Generate a new file when a stream is paused** – **Other**: If the stream pause length is within the specified range, the system does not generate a new file. Otherwise, a new file is generated.

6. Click **OK**.



Recording Rule

Only one recording rule can be created per app per region.

Add | Enter a recording rule ID.

Recording Rule ID	APPID	Stored At	Recor...	Created	Automatic Recordi...	Operation
[redacted]	[redacted]	Region: [redacted] Project ID: [redacted] Storage Bucket: 1231-xhy Storage Path: Record/	HLS,MP4	2021-05-10 17:02:45	Enabled	Edit | Delete

7. In the recording rule list, enable automatic recording as required. When automatic recording is enabled, if a new room is created under the app, the system automatically performs single stream recording during real-time communication in the room based on the configured recording rule.

   NOTE

   Automatic recording takes effect only for rooms created under the app after you enable it.

Step 4 Join a room.

After configuring recording rules, you can use the SparkRTC app to join a room for real-time interaction. SparkRTC will record the ongoing audio/video stream based on the configured recording rules.

NOTE

If automatic recording is not enabled during recording rule configuration, you need to call the **SparkRTC API** to start a cloud recording task after joining a room. Then SparkRTC can record images based on the recording rule ID in the API calling request.

Step 5 Replay recordings.

When the recording is complete, view recordings on the OBS console or through callback messages.

- Viewing recordings on the OBS console
  a. In the navigation pane of the OBS console, choose **Object Storage**.
  b. In the bucket list, click the bucket that stores SparkRTC recordings.
     The **Overview** page is displayed.
  c. In the navigation pane, choose **Objects** to view the recording information.
  d. Download and share the recordings as required. For details, see **OBS Documentation**.

----End

2 Implementing Audio and Video Calls

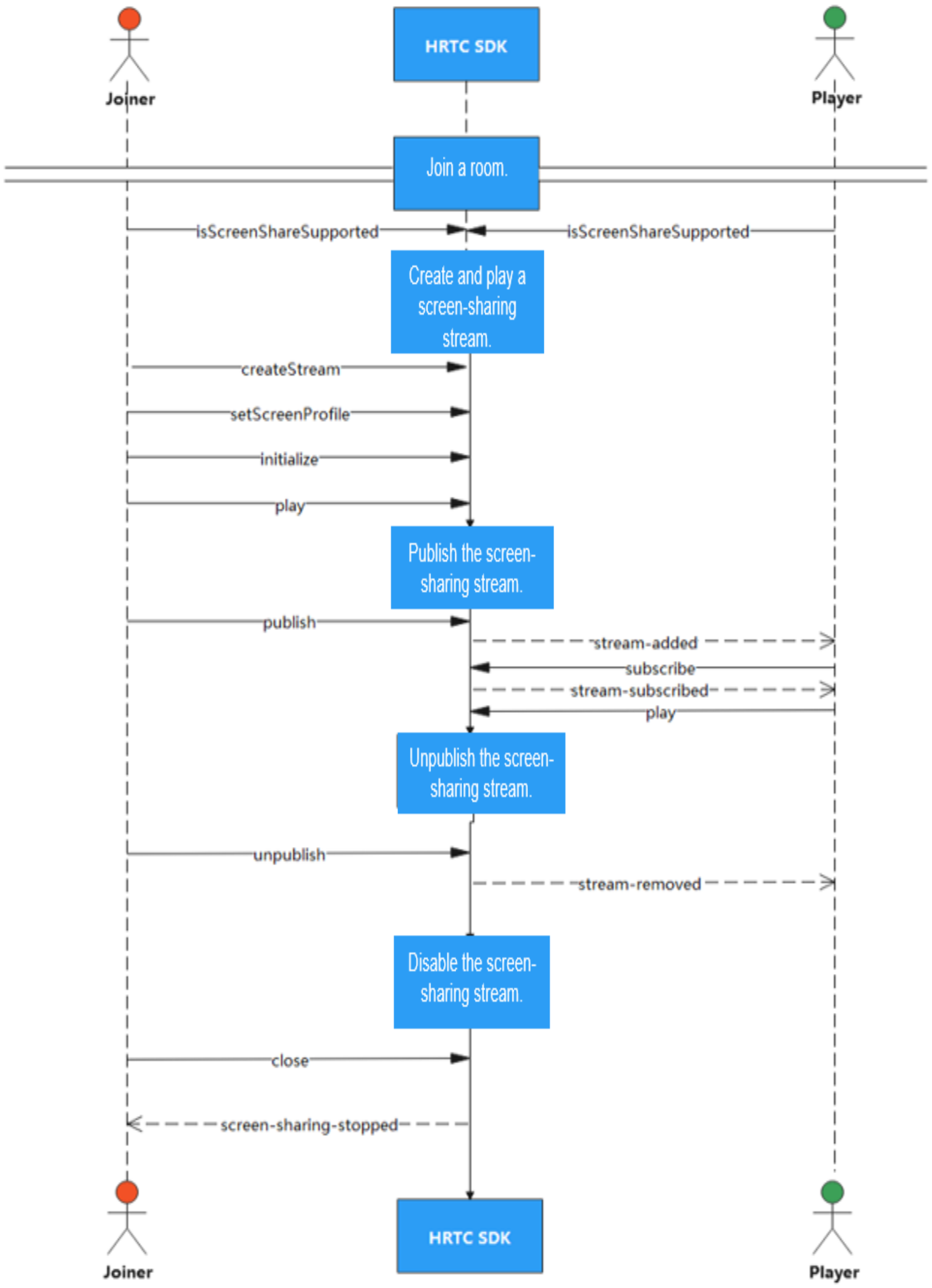
2.1 Web

2.1.1 Screen Sharing

Function

You can share the screen with other participants in video mode in a voice or video meeting. The desktop sharing function allows you to share the entire desktop or an application window.

API Calling



Implementation

1. **Join a room.**

   Refer to the sequence diagram for joining a room in **API Calling**.

2. **Create and play screen-sharing streams.**

   After joining a meeting, call the **isScreenShareSupported** API to check whether the browser supports screen-sharing streams. If yes, call

createStream to create a screen-sharing stream, **setScreenProfile** to configure the stream resolution, **initialize** to initialize the stream, **play** to play the stream, and **bindScreenAudio2RelatedStream** to determine whether to bind the background music of the shared screen to the associated stream object.

The sample code is as follows:

```
// screenAudio: indicates whether to share the video audio
let co = {screen: true , screenAudio: false}
// Create a screen-sharing stream.
let localAuxStream = HRTC.createStream(co)
// Configure the resolution of the screen-sharing stream.
localAuxStream.setScreenProfile('1080p')
// Initialize the screen-sharing stream.
localAuxStream.initialize()
.then(() => {   // Play c${this.clientIndex}-aux, which is the DOM element of the screen-sharing stream.
localAuxStream.play('c${this.clientIndex}-aux')
localAuxStream.bindScreenAudio2RelatedStream(this.localStream, screenAudio)})
.catch((error) => {
 console.error(error)
})
```

3. **Publish screen-sharing streams.**

   After the screen-sharing stream is played locally, call **publish** to publish the stream.

   The sample code is as follows:

```
this.client.publish(localAuxStream)
.then(() => {console.info('The screen-sharing stream has been published.')})
.catch((error) => {
 console.error ('Failed to publish the screen-sharing stream', error)
})
```

4. **Receive remote screen-sharing streams.**

   After receiving the **stream-added** notification indicating that the remote user has enabled screen-sharing streams, call **subscribe** to subscribe to the screen-sharing streams. When the subscription is successful, you will receive the **stream-subscribed** notification. Then call **play** to play the screen-sharing streams of the remote user in the specified window.

   The sample code is as follows:

```
this.client.on('stream-added', (event: any) => {
// For a screen-sharing stream:
if (event.stream.getType() === 'auxiliary') {
// remoteAuxStream: indicates a remote screen-sharing stream
let remoteAuxStream = event.stream
this.client.subscribe(remoteAuxStream )
.then(() => {console.info('You have subscribed to the screen-sharing stream.')})
.catch( (error) =>(console.info('Failed to subscribe to the screen-sharing stream.')))
 }})
this.client.on('stream-subscribed', (event: any) => {
// For a screen-sharing stream:
if (event.stream.getType() === 'auxiliary') {
// c${ clientIndex}-remoteAux: indicates the DOM element of the screen-sharing stream
 event.stream.play(`c${ clientIndex}-remoteAux`)
 }
})
```

5. **Unpublish screen-sharing streams.**

   After publishing a screen-sharing stream, you can call **unpublish** to unpublish the stream.

   The sample code is as follows:

```
this.client.unpublish(localAuxStream)
.then(() => {console.info('The screen-sharing stream has been unpublished.')})
```

```
.catch((error) => {
console.error('Failed to unpublish the screen-sharing stream', error)
})
```

6. **Stop playing screen-sharing streams.**

   After a screen-sharing stream is published, you can call **close** to stop playing the stream. Then the local end receives the **screen-sharing-stopped** callback.

   The sample code is as follows:

   ```
   localAuxStream.close()
   localAuxStream.on('screen-sharing-stopped', () => {
    console.log ('Screen sharing is stopped.')
    localAuxStream = null
   })
   ```

7. **Stop receiving remote screen-sharing streams.**

   After a remote screen-sharing stream is unpublished, the local end automatically unsubscribes to the stream and receives the **stream-removed** callback.

   The sample code is as follows:

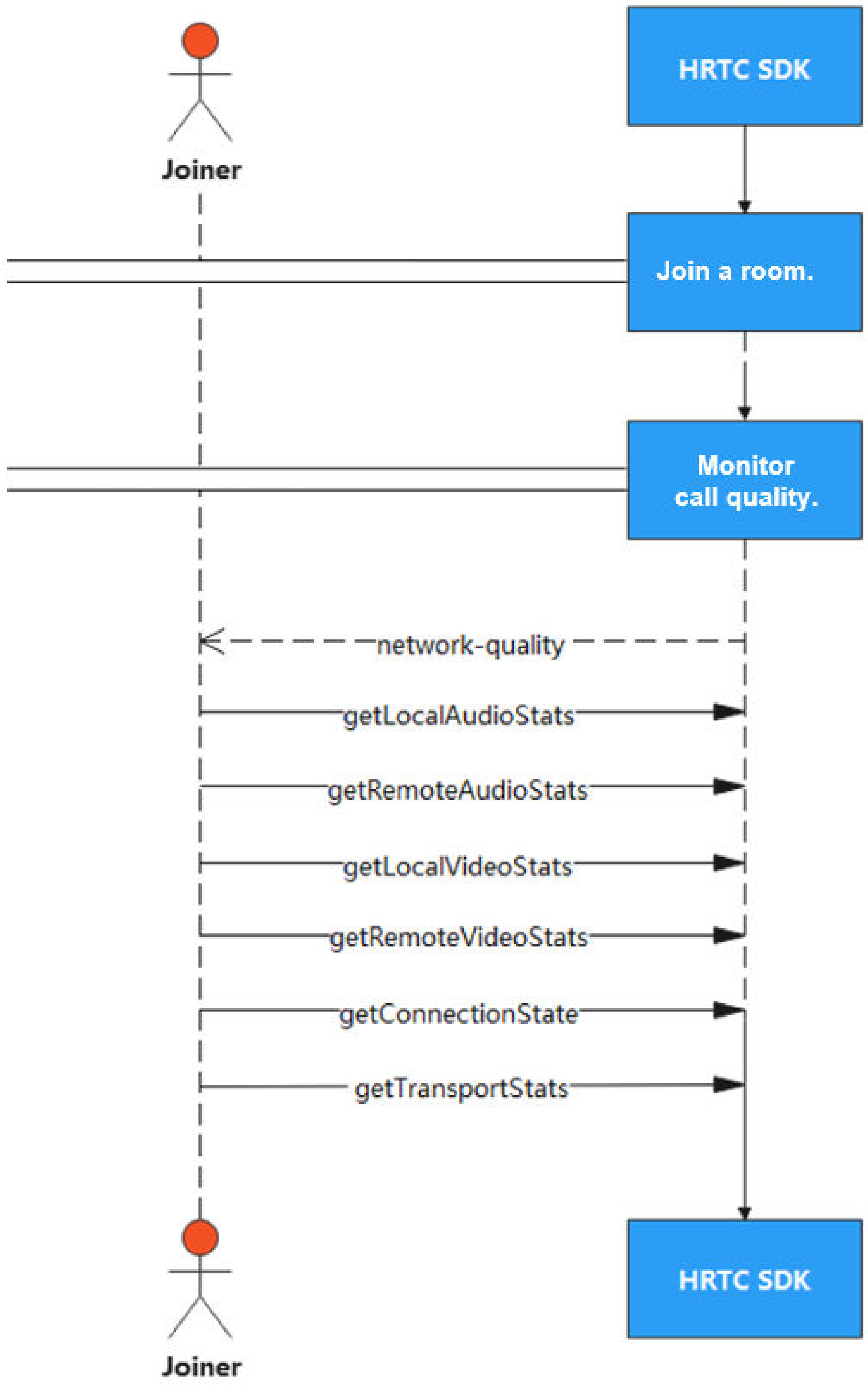
   ```
   this.client.on('stream-removed', (event: any) => {
    if (event.stream.getType() === 'auxiliary') {
    remoteAuxStream = null
    }
   })
   ```

2.1.2 Call Quality Monitoring

Function

After joining a room, the SDK triggers the callbacks related to the call quality to report the network quality of the current call and the local and remote audio and video statistics.

API Calling


HRTC SDK
Joiner
Join a room.
Monitor call quality.
network-quality
getLocalAudioStats
getRemoteAudioStats
getLocalVideoStats
getRemoteVideoStats
getConnectionState
getTransportStats
Joiner
HRTC SDK

Call Quality Reporting

After a user joins a room, the **network-quality** event is triggered when the network quality changes to report the uplink and downlink quality of the user's local network.

The sample code is as follows:

```
this.client.on('network-quality', (networkQualityInfo) => {
console.info(`network-quality:
uplinkNetworkQuality=${networkQualityInfo.uplinkNetworkQuality},
downlinkNetworkQuality = ${networkQualityInfo.downlinkNetworkQuality}`)
})
```

Obtaining Local Audio Stream Statistics

The **getLocalAudioStats** callback is used to obtain statistics on audio streams sent by a local device in the current call, including the number of bytes and packets.

The sample code is as follows:

```
this.client.getLocalAudioStats().then((stats) => {
 console.info(`getLocalAudioStats: ${stats}`)
})
```

Obtaining Remote Audio Stream Statistics

The **getRemoteAudioStats** callback is used to obtain the audio stream statistics of remote users in the current call, including the packet loss rate and the number of bytes and received packets.

The sample code is as follows:

```
this.client.getRemoteAudioStats().then((stats) => {
 console.info(`getRemoteAudioStats: ${stats}`)
})
```

Obtaining Local Video Stream Statistics

The **getLocalVideoStats** callback is used to obtain the current local video stream statistics, including the resolution and the number of frames, encoded frames, and sent bytes and packets.

The sample code is as follows:

```
this.client.getLocalVideoStats().then((stats) => {
 console.info(`getLocalVideoStats: ${stats}`)
})
```

Obtaining Remote Video Stream Statistics

The **getRemoteVideoStats** callback is used to obtain the current remote video stream statistics, including the resolution and the number of frames, encoded frames, and sent bytes and packets.

The sample code is as follows:

```
this.client.getRemoteVideoStats().then((stats) => {
 console.info(`getRemoteVideoStats: ${stats}`)
})
```

Checking the Client Connection Status

The **getConnectionState** callback is used to check the client connection status, which can be:

- **CONNECTING**
- **CONNECTED**
- **RECONNECTING**
- **DISCONNECTED**

The sample code is as follows:

```
console.info(`getConnectingState: ${this.client.getConnectionState()}`)
```

Obtaining the Network Transmission Statistics

The **getTransportStats** callback is used to obtain the current network transmission statistics, including the number of sent and received bytes, and the current output and input bitrates. This function is available only after the **publish** API is called.

The sample code is as follows:

```
this.client.getTransportStats().then(
 (rtt) => {
 console.info('###getTransportStats: bytesSent ' + rtt.bytesSent)
 console.info('###getTransportStats: bytesReceived ' + rtt.bytesReceived)
 console.info('###getTransportStats: sendBitrate ' + rtt.sendBitrate)
 console.info('###getTransportStats: recvBitrate ' + rtt.recvBitrate)
 console.info(`getTransportStats: ${rtt.rtt}`)
 },
 (error) => {
 console.info(`getTransportStats: ${error}`)
})
```

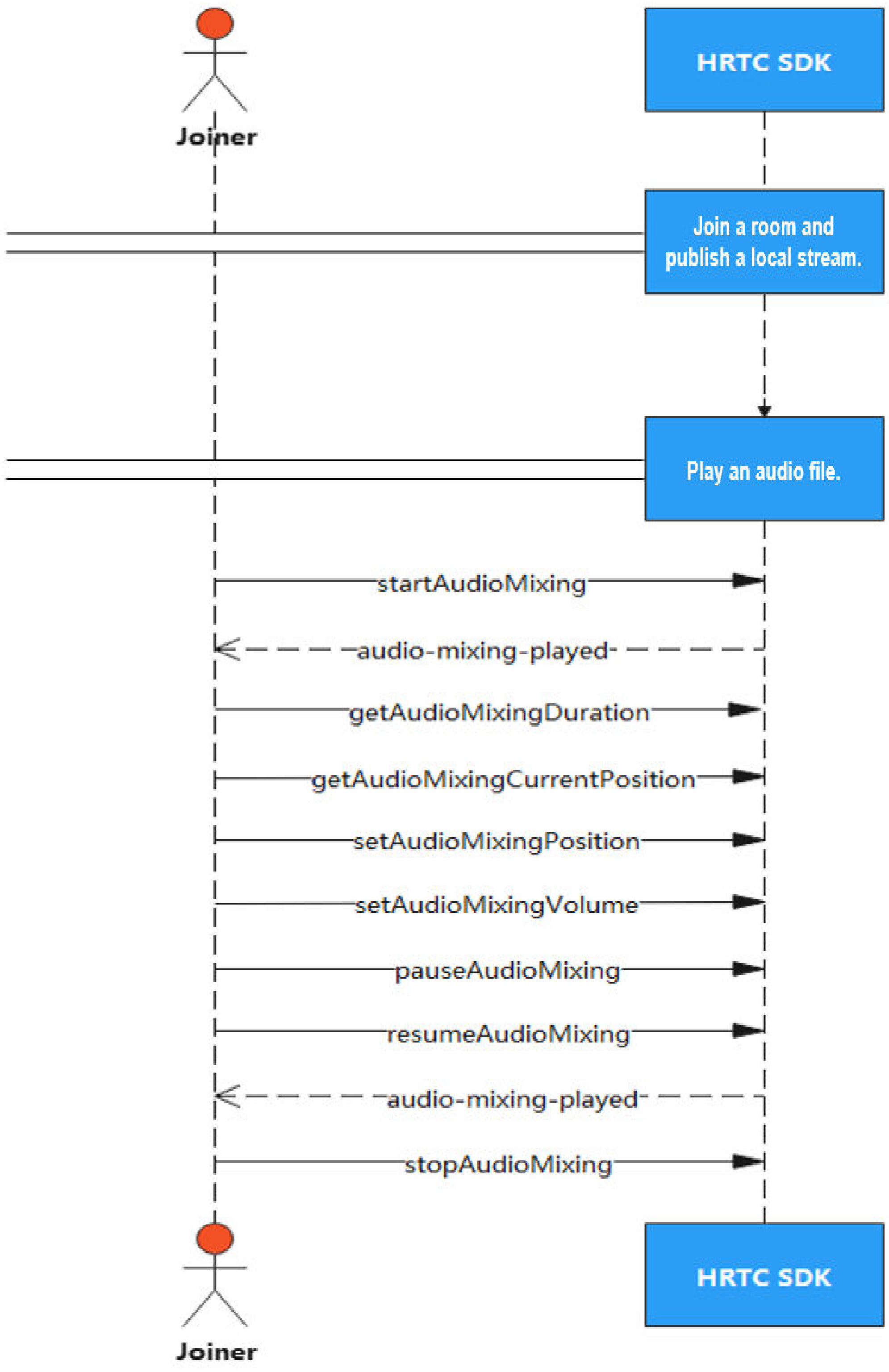
2.1.3 Playing an Audio File

Function

Audio mixing is the process of combining a music file with the audio captured by a microphone and is generally for background music or accompaniment that lasts for a long time. Only one audio file can be played at a time for other users in the room.

Local or online music files in WAV, PCM, or mono MP3 format can be played.

API Calling



Implementation

1. **Join a room and publish the local video stream.**

Join a room by referring to the sequence diagram in **API Calling** and publish the local video stream.

2. **Play an audio file.**

   Call **startAudioMixing** to play an audio file. The following is an example of parameter settings. Only one audio file can be played at a time.

```
// localStream indicates the local video stream.
localStream.startAudioMixing({
// filePath indicates the path for downloading an online audio file.
"filePath":"https://***.***.***.***:50007/music.mp3",
// startTime indicates the time when the audio file starts to be played. The default value is 0.
"startTime":0,
// replace indicates whether to replace the local audio stream with an audio file.
"replace":false,
// loop indicates whether infinite loop playback is required.
"loop":false,
// repeatCount indicates the number of times of repeating the audio file.
"repeatCount":0
})
```

3. **Set the volume of the audio file.**

   During playback, call **setAudioMixingVolume** to set the volume of the audio file.

```
// volume indicates the volume.
 let volume = 50
 localStream.setAudioMixingVolume (volume)
```

4. **Obtain the total duration of the audio file.**

   After an audio file is played, call **getAudioMixingDuration** to obtain its total duration, which can be used to refresh the progress bar on the UI.

```
localStream.getAudioMixingDuration()
```

5. **Obtain the progress of audio file playback.**

   After an audio file is played, call **getAudioMixingCurrentPosition** to obtain its current playback position.

```
localStream.getAudioMixingCurrentPosition()
```

6. **Set the playback position of the audio file.**

   During playback, call **setAudioMixingPosition** to set the playback position of the audio file and drag the progress bar to seek to the desired position.

```
localStream.getAudioMixingDuration()
```

7. **Pause the playback.**

   After the audio file is played, call **pauseAudioMixing** to pause the playback.

```
localStream.pauseAudioMixing()
```

8. **Resume the playback.**

   After the audio file is paused, call **resumeAudioMixing** to resume the playback.

```
localStream.resumeAudioMixing()
```

9. **Stop the playback.**

   After the audio file is played, call **stopAudioMixing** to stop the playback.

```
localStream.stopAudioMixing()
```

10. **Trigger a callback when the audio file playback starts.**

    When the audio file playback starts, the **audio-mixing-played** callback is triggered to notify the application.

```
localStream.on('audio-mixing-played', () => {
 console.info('audioMixing: audio-mixing-played')
})
```

11. **Trigger a callback when the audio file playback is complete.**

    When the audio file playback is complete, the **audio-mixing-finished** callback is triggered to notify the application.

```
localStream.on('audio-mixing-finished', () => {
 console.info('audioMixing: audio-mixing-finished')
})
```

2.1.4 Switching the Audio Mode

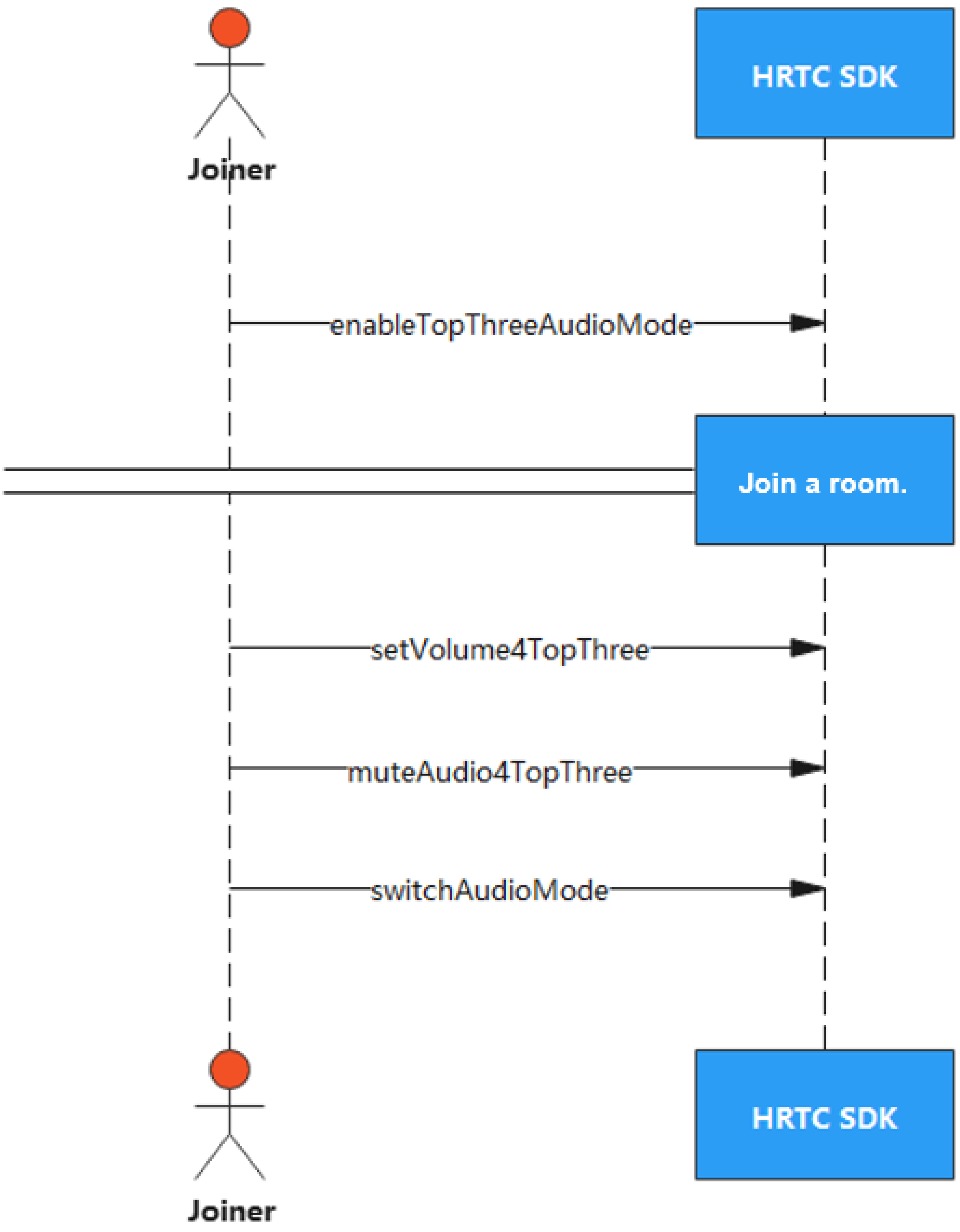
Function

Before joining a meeting, a user can call **enableTopThreeAudioMode** to receive the three loudest audio streams in the room.

During the meeting, **switchAudioMode(2)** can be called to switch the audio mode to the subscription mode. In this mode, the local user can receive the remote audio streams only after subscribing to them.

During the meeting, **switchAudioMode(3)** can be called to receive the three loudest audio streams in the room. In this mode, the local user can receive the three loudest audio streams in the room without subscribing to the remote audio streams.

API Calling



Implementation

1. **Determine whether to receive the three loudest audio streams in the room.**

   Before joining a meeting, call **enableTopThreeAudioMode** to determine whether to receive the three loudest audio streams in the room. **true** indicates that the function is enabled and **false** indicates that the function is disabled.

   The sample code is as follows:

```
this.client.enableTopThreeAudioMode(true)
```

2. **Join a room.**

   Refer to the sequence diagram for joining a room in **API Calling**.

3. **Set the maximum audio volume of the three loudest users.**

   Call **setVolume4TopThree** to set the maximum audio volume of the three loudest users. The value ranges from 0 to 100.

   The sample code is as follows:

```
// volume indicates the volume.
let volume = 50
this.client.setVolume4TopThree(volume)
```

4. **Enable or disable audio tracks of the three loudest users.**

   Call **muteAudio4TopThree** to enable or disable audio tracks of the three loudest users. **true** indicates that their audio tracks are disabled and **false** indicates that their audio tracks are enabled.

   The sample code is as follows:

```
this.client.muteAudio4TopThree(true)
```

5. **Switch the audio mode.**

   Call **switchAudioMode** to switch the audio mode.

   switchAudioMode(2) indicates the audio subscription mode and **switchAudioMode(3)** indicates that the three loudest audio streams are subscribed to.

   The sample code is as follows:

```
this.client.switchAudioMode(2)
this.client.switchAudioMode(3)
```