Low Latency Live

Client SDK Reference

Issue 01
Date 2025-02-21

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
             Qianzhong Avenue
             Gui'an New District
             Gui Zhou 550029
             People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 SDK Overview

The SDK of Huawei Cloud Low Latency Live (LLL) encapsulates REST APIs provided by LLL to simplify development. You can directly call API functions provided by the LLL SDK to use LLL. For details about how to download and integrate the client SDK and use APIs, see **Table 1-1**.

> **NOTICE**
>
> Currently, LLL supports only SDK access, not signaling access.

Table 1-1 Client SDK

Client	SDK Download	Version	Developed By	Function	SDK Reference	API Reference
Web	SDK: **HWLLS_SDK_Web_2.10.9.tar.gz** Integrity check: **HWLLS_SDK_Web_2.10.9.tar.gz.sha256**	2.10.9	Huawei Cloud	LLL functions	**Web SDK Reference**	**Web API Reference**

Checking Software Package Integrity

Check the integrity of downloaded SDK packages, that is, check whether the packages are tampered with or packets are lost during download.

The procedure is as follows:

Step 1 Download the SDK package and its integrity verification SHA-256 package in **Table 1-1** to the local PC.

Step 2 Open the local CLI and run the following command to generate the SHA-256 value of the downloaded SDK package on the local PC.

In the following command, **D:\HWLLS_SDK_Web_2.6.0.tar.gz** indicates the local path for storing the SDK package and the SDK package name. Change it as needed.

certutil -hashfile D:\HWLLS_SDK_Web_2.6.0.tar.gz SHA256

Example command output:

D:\HWLLS_SDK_Web_2.6.0.tar.gz hash of SHA-256:
3ac83be852e8dcc9e90f236801fd4c494983073543e1ae66ee4d0c29043dccd1
CertUtil: -hashfile Command executed.

Step 3 Compare the SHA-256 value of the downloaded SDK package with that of the queried SDK package.

If they are the same, no tampering or packet loss occurred during download.

----End

2 Version Planning

This section describes the version planning of the LLL client SDK.

Version Description

The version number is in the format of a.b.c, where:

- a indicates the major version number, which is updated when the version architecture is reconstructed. For example, the major version number will be changed if the APIs of different versions are incompatible.
- b indicates the minor version number, which is an iterative version. If there are new functions or features, or APIs are added or optimized, the value of this field is incremented by 1.
- c indicates the patch version number. If a function is optimized or a defect is rectified, the value of this field is incremented by 1.

Example version number: 2.0.1

Version Lifecycle

By default, a version is released every one to two months, or a version is modified based on customer requirements.

Version Constraints

None. The old and new versions are compatible.

3 Web SDK

3.1 Browser Adaptation

This section describes the browser types and versions supported by the LLL web SDK and constraints.

Table 3-1 Browser adaptation

OS	Browser Type	Browser Version
Windows	Chrome browser	67+
	QQ Browser (fast kernel)	10.4+
	360 Secure Browser (fast mode)	12
	WeChat embedded browser	-
	Mozilla Firefox	90+
	Microsoft Edge	80+
	Opera browser	54+
macOS	Chrome browser	67+
	WeChat embedded browser	-
	Safari	13+
	Mozilla Firefox	90+
	Opera browser	56+
Android	WeChat embedded browser (TBS kernel)	-
	WeChat embedded browser (XWEB kernel)	-

OS	Browser Type	Browser Version
	Mobile Google Chrome	83+
	Mobile QQ Browser	12+
	Huawei browser	11.0.8+
iOS	WeChat embedded browser	iOS 14.3+ WeChat 6.5+
	Mobile Google Chrome	iOS 14.3+
	Mobile Safari	13+

Table 3-2 Constraints on browsers

Browser Type	Constraint
Chrome browser	1. On Huawei mobile devices, the Chrome browser (including the Huawei browser) supports WebRTC 91 or later. 2. WebView on Android mobile devices supports WebRTC. The support varies due to factors such as device vendors, browser kernels, and versions. You are advised to select the most compatible browser.
Safari	The audio on iOS Safari 14.2 and macOS Safari 14.0.1 may be intermittent.
Mozilla Firefox	Firefox on macOS devices with Apple M1 chips does not support H.264 codec.
Opera browser	On Huawei mobile devices, the Opera browser supports WebRTC 64 or later.

3.2 Preparations

Prerequisites

The SDK package has been downloaded.

Environment Requirements

- You are advised to install Microsoft Visual Studio Code 1.43.2 or a later version for compilation.

- If Node.js is used for development on the client, you are advised to install Node.js 14.19.1 or later.

- For details about the supported browsers, see **Browser Adaptation**.

- If you want to develop your client using TypeScript, the TypeScript version must be 3.8.3 or later.

SDK Integration

Step 1 Download the **SDK** to the local PC. You are advised to save the SDK package to a directory named **sdk** of your project.

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

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

```
<script src='./sdk/HWLLSPlayer.js'>
    console.log(HWLLSPlayer.getVersion())
</script>
```

- To introduce the SDK in npm modularization mode, install the HWLLSPlayer module and introduce HWLLSPlayer to the development dependency of package.json, for example, **"HWLLSPlayer": "./sdk/HWLLS_SDK_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 HWLLSPlayer from 'HWLLSPlayer'
console.log(HWLLSPlayer.getVersion())
```

----End

3.3 SDK Usage

Step 1 Create a container.

```
<body>
    <div id='preview' style='width:1280px; height:720px'>
    </div>
</body>
```

Step 2 Create a client by referring to **createClient**.

```
const client = HWLLSPlayer.createClient()
```

Step 3 Enter the stream URL and container ID to start playback. See **startPlay**.

```
const streamUrl = 'webrtc://domain/appname/streamname'
client.startPlay(streamUrl, {
    elementId: 'preview', // (Mandatory) Container ID. Generally, the ID of the div tag is transferred. Here,
the ID of the div container in step 1 is entered.
})
client.on('Error', (errorInfo)=>{
    // Listen on and handle playback errors.
    console.log(`Something error: ${errorInfo.getMsg()}`)
})
```

For details about the error, see **Client Error Codes**.

Step 4 Stop the playback by referring to **stopPlay**.

```
client.stopPlay()
```

Step 5 Release resources by referring to **destroyClient**.

```
client.destroyClient()
```

See **Advanced Usage**.

----End

3.4 Basic Usage Logic

The procedure is as follows:

- Before stream pull: Create a client.
- Stream pull for playback: Send a stream pull request.
- Stop the playback: Stop the playback request.
- After stream pull: Destroy the client.

You can click an API name in the following figure to view its description and usage method.
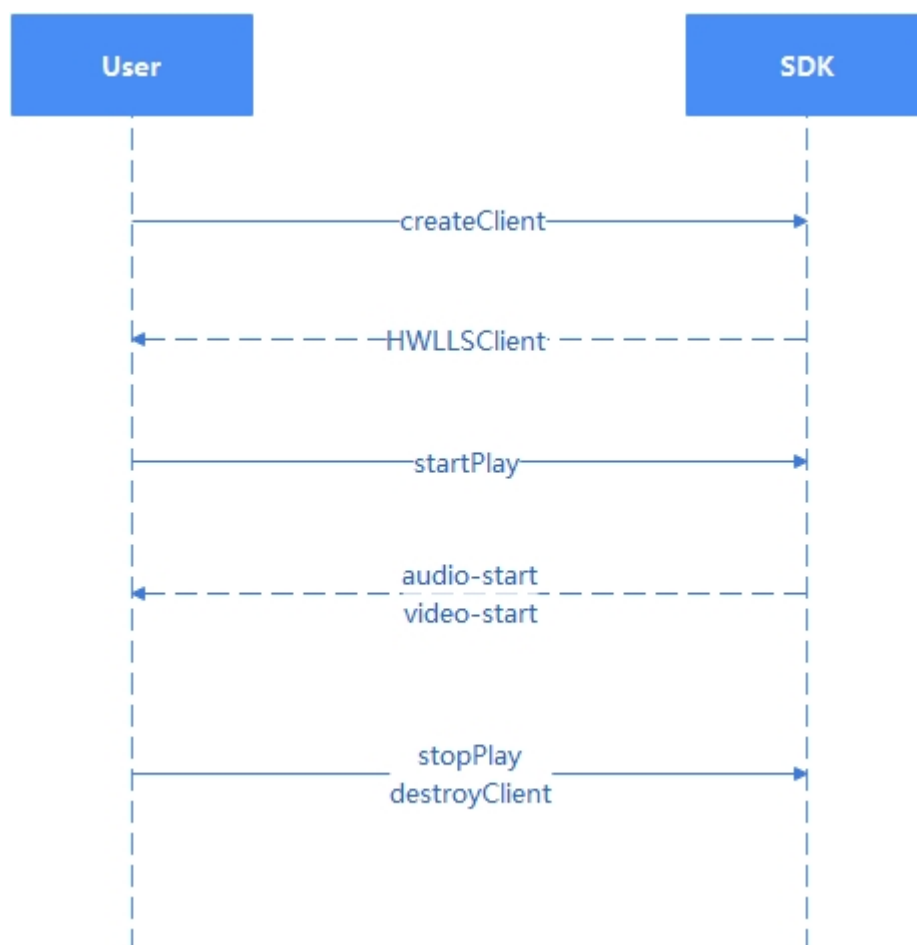
3.5 Best Practices

3.5.1 Advanced Usage

Overview

Advanced usage covers the following scenarios:

- **Scenario 1: Playback**
- **Scenario 2: Pausing and Resuming Playback**
- **Scenario 3: Switching to Another Video**
- **Scenario 4: Setting Full-Screen Playback**
- **Scenario 5: Muting**
- **Scenario 6: Stopping Playback**
- **Scenario 7: Destroying a Player**

For details about the complete code in the preceding scenarios, see **Advanced Usage**. You can copy the code to the local PC for testing.

Scenario 1: Playback

When the **startPlay** method is used, the configuration needs to be transferred. **elementId** is a mandatory DOM node ID, which specifies the video rendering node. You are advised to transfer **downgradeUrl** so that the specified URL can be used for playback to reduce playback failures when the browser does not support WebRTC or the network condition is poor. If you want the playback area to be filled with the video, set **objectFit** to **fill**. Three video display modes are supported. See **startPlay**.

```
// Playback configuration
const options = {
  // Playback DOM node ID.
  elementId: 'preview',
  // URL used after downgrade
  downgradeUrl: {
    flvUrl: 'https://xxxx/xx/xx/xx.flv',
    hlsUrl: 'https://xxxx/xx/xx/xx.m3u8'
  },
  // Video display adaptation
  objectFit: 'fill'
}

// Playback
const startPlay = function () {
  playClient.startPlay(url, options)
}
```

Scenario 2: Pausing and Resuming Playback

To pause the playback, call the **pause** method by referring to **pause**. To resume the playback, call the **resume** method by referring to **resume**.

```
// Pause the playback.
const pauseAction = function () {
  playClient.pause()
}

// Resume the playback.
const resumeAction = function () {
  playClient.resume()
}
```

Scenario 3: Switching to Another Video

To quickly switch to another video during playback, call the **switchPlay** method to transfer the target video URL. See **switchPlay**.

```
// Switch to another video.
const switchAction = function () {
```

```
    const url = 'a new url'
    playClient.switchPlay(url)
}
```

Scenario 4: Setting Full-Screen Playback

You can call the **fullScreenToggle** method to set full-screen playback. See **fullScreenToggle**.

```
// Full screen
const fullScreenAction = function () {
  playClient.fullScreenToggle()
}
```

Scenario 5: Muting

You can call the **muteAudio** method to mute or unmute a video. See **muteAudio**.

```
// Mute the video.
const muteAction = (() => {
  let mute = false
  return function () {
    mute = !mute
    playClient.muteAudio(mute)
  }
})()
```

Scenario 6: Stopping Playback

You can call the **stopPlay** method to stop playback. This method is different from **pause**. **pause** means temporarily stopping video playback, and the playback may be resumed after a short period of time. Stream pull continues but there are no image and sound. **stopPlay** means interrupting the stream. See **stopPlay**.

```
// Stop playback.
const stopAction = function () {
  playClient.stopPlay()
}
```

Scenario 7: Destroying a Player

After the playback task is complete, you can call the **destroyClient** method to destroy the player. Generally, if you create a player on a page, you can play multiple videos on the page and need to destroy the player when leaving the page. See **destroyClient**.

```
// Destroy the player.
const destroyAction = function () {
  playClient.destroyClient()
  playClient = null
}
```

3.5.2 Audio Autoplay Blocking

What Is Audio Autoplay Blocking?

Automatically starting the playback of audio (or videos with audio tracks) immediately upon page load can be an unwelcome surprise to users. In order to give users control over this, browsers often provide audio autoplay blocking policies. During audio autoplay, audio autoplay blocking policies may be triggered

when users do not engage with the page. However, audio autoplay will not be blocked in non-autoplay or muted autoplay scenarios. Specifically, if the **audio** or **video** tag contains the **autoplay** attribute, autoplay is blocked after the page is loaded. If the **play** API is forcibly called, an error message similar to **Uncaught (in promise) DOMException: play() failed because the user didn't interact with the document first** will be reported.

HTML:

```
<video src="/video_with_audio.mp4" autoplay></video>
```

JavaScript:

```
videoElement.play();
```

Autoplay of a video with an audio track on a new page will be denied by the browser.

Generally, audio autoplay is not allowed by the browser until users engage with the page. For example, the **play** API will be called when users click the playback button on the page.

```
PlayButton.addEventListener('click', () => {
  videoElement.play();
})
```

LLL SDK Error Messages Caused by Audio Autoplay Blocking

When the LLL SDK is used to autoplay an LLL livestream with sound:

```
const options = {
  elementId: 'elementId',
  autoPlay: true,
}
startPlayPromise = playClient.startPlay(streamUrl, options)
```

Set **autoPlay** to **true**. If **autoPlay** is not specified, the default value **true** is used. When audio autoplay is blocked, the browser console reports the following error message:

```
[HWLLS] [error] [HLLSTrack] [play audio failed: ["code": 51000000, "message": "the user didn't interact
with the document first, please trigger by gesture."] ]
### lll play SDK occur error: {"errCode":51000000,"errDesc":"the user didn't interact with the document
first, please trigger by gesture."}
```

If **Error** is reported during listening, error code **51000000** is displayed:

```
playClient.on('Error', (resp) => {
  if (resp.errCode === 51000000) {
    // Error message on audio autoplay blocking
  }
})
```
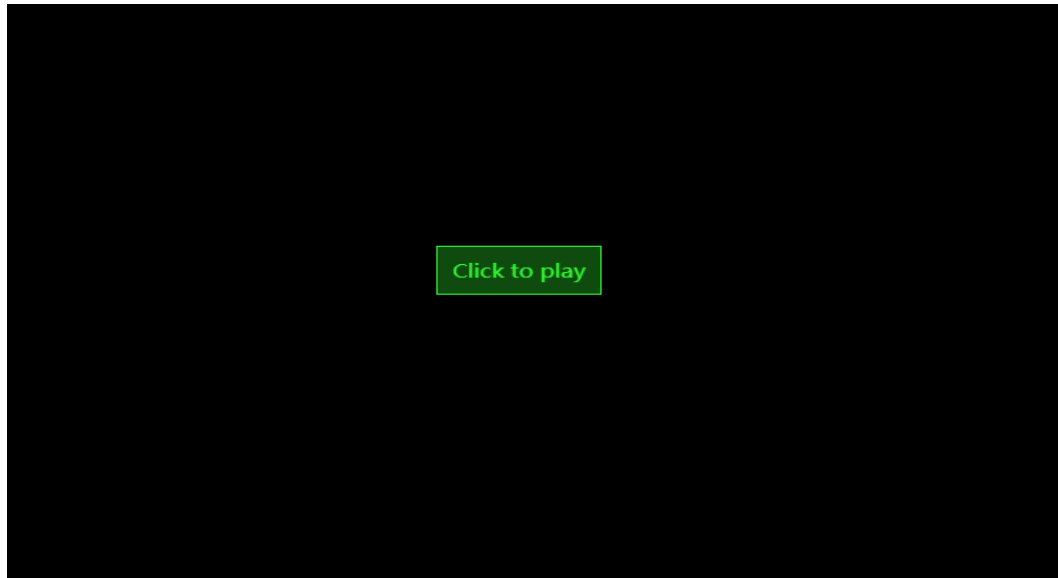
Best Practices

Manual playback

Example of muted autoplay using the LLL SDK:

```
const options = {
  elementId: 'elementId',
  autoPlay: false
}
startPlayPromise = playClient.startPlay(streamUrl, options)
```

Set **autoPlay** to **false** and add a playback button so that the playback starts only when the user clicks the button.

Figure 3-1 Effect picture

Listening on autoplay errors

Autoplay with sound:

```
const options = {
 elementId: 'elementId',
 autoPlay: true,
 }
startPlayPromise = playClient.startPlay(streamUrl, options)
```

Listening error code on audio autoplay blocking:

```
playClient.on('Error', (resp) => {
 if (resp.errCode === 51000000) {
   // Error message on audio autoplay blocking
   // Add the button unmute on the page.
 }
})
```

In this case, even if audio autoplay is blocked, the video is still played without sound. If the user wants to hear the sound, the user can click **unmute** to play the audio. When audio autoplay is not blocked, the video will be played with sound.
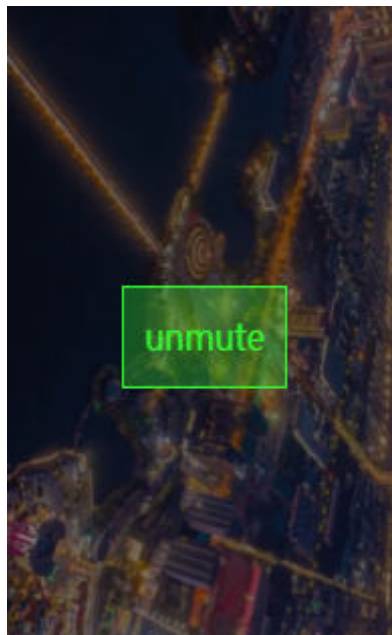
Figure 3-2 Effect picture 1

Figure 3-3 Effect picture 2

Sample Code

📖 NOTE

The LLL SDK needs to be decompressed to the **sdk/** directory.

● **Listening on Autoplay Errors** shows the sample code of listening on autoplay errors.

● **Manual Playback** shows the sample code of manual playback.

3.5.3 Downgrade

Scenarios

About downgrade

When the Low Latency Live (LLL) SDK is used to pull livestreams, the playback may fail. In this case, you can use other protocols for playback.

Trigger conditions

- The browser environment does not support WebRTC. You can call the **checkSystemRequirements** API to check this.

- The server request and link setup fail.

- The playback start times out and there is no decoded frame.

- Stream interruption occurs during playback and retry upon stream interruption is not enabled.

Methods

- **Automatic downgrade**

  By default, automatic downgrade is enabled for the LLL SDK. When automatic downgrade is triggered, the SDK automatically attempts to downgrade from LLL to FLV or HLS. The following example is an LLL URL:

  ```
  webrtc://domain/appname/streamname?arg1=v1
  ```

  The URL is automatically converted to an FLV or HLS URL that the device supports. The parameters in the original URL are combined to the converted URL after the downgrade. Example:

  ```
  https://domain/appname/streamname.flv?vhost=domain&arg1=v1
  Or
  https://domain/appname/streamname.m3u8?vhost=domain&arg1=v1
  ```

  To disable automatic downgrade, call the **setParameter** API to set the value of **AUTO_DOWNGRADE**. Example:

  ```
  HWLLSPlayer.setParameter('AUTO_DOWNGRADE', false) // true (automatic downgrade enabled, by
  default) and false (automatic downgrade disabled)
  ```

- **Specified downgrade**

  Call the **startPlay** API in **HWLLSClient** to specify the FLV URL or HLS URL of **downgradeUrl** in **options** to use the URL for playback when a downgrade occurs. If either **hlsUrl** or **flvUrl** is specified, the specified URL will be used. If both of them are specified, **hlsUrl** will be used first and then **flvUrl**. If the HLS is not supported or the HLS stream pull fails, the FLV URL will be used for playback after downgrade. Note that iOS devices do not support FLV playback.

  ```
  const client = HWLLSPlayer.createClient()
  client.startPlay(url, {
      …
      downgradeUrl: {
          hlsUrl: // HLS URL.
          flvUrl: // FLV URL.
      }
      …
  })
  ```

Downgrade Callbacks

When a downgrade is triggered, a callback will be initiated.

```
const client = HWLLSPlayer.createClient()
client.on('player-changed', (mediaFormat) => {
    // mediaFormat: hls, flv
})
```

Sample Code

> ☐ **NOTE**
>
> The LLL SDK needs to be decompressed to the **sdk/** directory.

Sample code of the two downgrade methods:

- See **Automatic Downgrade**.
- See **Specified Downgrade**.

3.5.4 Sample Code

3.5.4.1 Advanced Usage

Complete sample code of advanced usage:

```html
<!DOCTYPE html>
<html>
  <head>
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0, user-scalable=no"
    />
    <title>Advanced usage</title>
    <style>
      html, body {
        width: 100%;
        height: 100%;
        padding: 0;
        margin: 0;
      }
      .preview_player {
        width: 1280px;
        height: 720px;
        margin: 20px auto;
        border: 1px solid #ddd;
        position: relative;
      }
      .tools {
        width: 1280px;
        margin: 20px auto;
        display: flex;
        flex-wrap: wrap;
      }
      .play_btn {
        color: #fff;
        border: 1px solid rgb(2, 16, 201);
        background: rgba(2, 16, 201, 0.8);
        border-radius: 2px;
        cursor: pointer;
        margin-left: 10px;
      }
      .op_btn {
        margin-left: 10px;
        cursor: pointer;
      }
    </style>
  </head>
  <body>
    <div id="preview" class="preview_player"></div>
    <div class="tools">
      <input id="playUrl"></input>
      <button class="play_btn" onclick="playAction()">Play</button>
```

```
      <button class="op_btn" onclick="pauseAction()">Pause</button>
      <button class="op_btn" onclick="resumeAction()">Resume</button>
      <button class="op_btn" onclick="switchAction()">Switch video</button>
      <button class="op_btn" onclick="stopAction()">Stop</button>
      <button class="op_btn" onclick="fullScreenAction()">Full screen</button>
      <button class="op_btn" onclick="muteAction()">Mute/Unmute</button>
      <button class="op_btn" onclick="destroyAction()">Destroy</button>
    </div>
    <script type="text/javascript" src="./sdk/HWLLSPlayer.js"></script>
    <script type="text/javascript">
     const playUrlInput = document.getElementById('playUrl')
     playUrlInput.value = 'your webrtc url'

     let playClient = HWLLSPlayer.createClient()
     let isPlaying = false

     const playAction = function () {
       startPlay()
     }

     // Playback configuration
     const options = {
       // Playback DOM ID
       elementId: 'preview',

       // URL used after downgrade
       downgradeUrl: {
         flvUrl: 'your flv url',
         hlsUrl: 'your hls url'
       },

       // Video display adaptation
       objectFit: 'fill'
     }

       // Playback
     const startPlay = function () {
       if (!isPlaying) {
         isPlaying = true
         const url = playUrlInput.value
         playClient.startPlay(url, options)
       }
     }

     // Pause the playback.
     const pauseAction = function () {
       playClient.pause()
     }

     // Resume the playback.
     const resumeAction = function () {
       playClient.resume()
     }

     // Switch to another video.
     const switchAction = function () {
       const url = playUrlInput.value
       playClient.switchPlay(url)
     }

     // Stop playback.
     const stopAction = function () {
       playClient.stopPlay()
       isPlaying = false
     }

     // Full screen
     const fullScreenAction = function () {
       playClient.fullScreenToggle()
```

```
    }

    // Mute the video.
    const muteAction = (() => {
      let mute = false
      return function () {
        mute = !mute
        playClient.muteAudio(mute)
      }
    })()

    // Destroy the player.
    const destroyAction = function () {
      playClient.stopPlay()
      playClient.destroyClient()
      playClient = null
      isPlaying = false
    }
  </script>
 </body>
</html>
```

3.5.4.2 Listening on Autoplay Errors

Sample code of listening on autoplay errors:

```
<!DOCTYPE html>
<html>
 <head>
  <meta
    name="viewport"
    content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0, user-
scalable=no"
  />
  <title>Audio playback blocking demo</title>
  <style>
    html, body {
      width: 100%;
      height: 100%;
      padding: 0;
      margin: 0;
    }
    .preview_player {
      width: 1280px;
      height: 720px;
      margin: 20px auto;
      border: 1px solid #ddd;
      position: relative;
    }
    .preview_player_el {
      width: 100%;
      height: 100%;
    }
    .unmute_mask {
      position: absolute;
      top: 0;
      left: 0;
      width: 100%;
      height: 100%;
      background: rgba(0, 0, 0, 0.5);
      text-align: center;
      display: none;
    }
    .unmute_btn {
      display: inline-block;
      padding: 10px;
      color: rgb(50, 250, 50);
      border: 1px solid rgb(50, 250, 50);
      background: rgba(50, 250, 50, 0.3);
```

```
            transform: translateY(400px);
            cursor: pointer;
          }
      </style>
      <script type="text/javascript" src="./sdk/HWLLSPlayer.js"></script>
    </head>
    <body>
      <div class="preview_player">
        <div id="preview" class="preview_player_el"></div>
        <div class="unmute_mask" id="unmute-mask">
          <div class="unmute_btn" onclick="replayAction()">Click to unmute</div>
        </div>
      </div>
      <script type="text/javascript">
        const playClient = HWLLSPlayer.createClient()
        const unmuteMask = document.getElementById('unmute-mask')
        const replayAction = function () {
          playClient.replay()
          unmuteMask.style.display = 'none'
        }
        // Handling registered audio playback blocking
        const initEvent = function () {
          playClient.on('Error', (resp) => {
            // Error message on audio autoplay blocking
            if (resp.errCode === 51000000) {
              // Add the button unmute on the page.
              unmuteMask.style.display = 'block'
            }
          })
        }
        initEvent()
        // Video playback
        const streamUrl = 'your stream url'
        const options = {
          elementId: 'preview',
          autoPlay: true,
        }
        playClient.startPlay(streamUrl, options)
      </script>
    </body>
</html>
```

3.5.4.3 Manual Playback

Sample code of manual playback:

```
<!DOCTYPE html>
<html>
  <head>
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0, user-scalable=no"
    />
    <title>Manual playback demo</title>
    <style>
      html, body {
        width: 100%;
        height: 100%;
        padding: 0;
        margin: 0;
      }
      .preview_player {
        width: 1280px;
        height: 720px;
        margin: 20px auto;
        border: 1px solid #ddd;
        position: relative;
      }
```

```
       .preview_player_el {
         width: 100%;
         height: 100%;
       }
       .unmute_mask {
         position: absolute;
         top: 0;
         left: 0;
         width: 100%;
         height: 100%;
         background: rgba(0, 0, 0, 0.5);
         text-align: center;
       }
       .unmute_btn {
         display: inline-block;
         padding: 10px;
         color: rgb(50, 250, 50);
         border: 1px solid rgb(50, 250, 50);
         background: rgba(50, 250, 50, 0.3);
         transform: translateY(400px);
         cursor: pointer;
       }
     </style>
     <script type="text/javascript" src="./sdk/HWLLSPlayer.js"></script>
   </head>
   <body>
     <div class="preview_player">
       <div id="preview" class="preview_player_el"></div>
       <div class="unmute_mask" id="unmute-mask">
         <div class="unmute_btn" onclick="playAction()">Click to play</div>
       </div>
     </div>
     <script type="text/javascript">
       const playClient = HWLLSPlayer.createClient()
       const unmuteMask = document.getElementById('unmute-mask')
       const playAction = function () {
         playClient.resume()
         unmuteMask.style.display = 'none'
       }
       // Video playback
       const streamUrl = 'your stream url'
       const options = {
         elementId: 'preview',
         autoPlay: false,
       }
       playClient.startPlay(streamUrl, options)
     </script>
   </body>
</html>
```

3.5.4.4 Automatic Downgrade

Sample code of automatic downgrade:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Downgrade Demo</title>
   <script src="sdk/HWLLSPlayer.js"></script>
</head>
<body>
<div id="preview" style="width: 1280px; height: 720px; position:relative;"></div>
<button id="btnStartPlay">start play</button>
<button id="btnStopPlay">stop play</button>
<script>
   HWLLSPlayer.setParameter('AUTO_DOWNGRADE', true)

   const btnStartPlay = document.getElementById('btnStartPlay')
```

```
    const btnStopPlay = document.getElementById('btnStopPlay')

    const streamUrl = 'your stream url'

    const client = HWLLSPlayer.createClient()
    bindEvent(client)

    btnStartPlay.addEventListener('click', () => {
        startPlay(client, streamUrl, 'preview')
    })
    btnStopPlay.addEventListener('click', () => {
        stopPlay(client)
    })

    function bindEvent(client) {
        client.on('Error', (error) => {
            console.log(`error: ${JSON.stringify(error)}`)
        })
        client.on('player-changed', (mediaFormat) => {
            console.log(`player changed:${mediaFormat}`)
        })
    }

    function startPlay(client, url, elementId) {
        client.startPlay(url, {
            elementId: elementId,
        })
    }

    function stopPlay(client) {
        client.stopPlay()
    }
</script>
</body>
</html>
```

3.5.4.5 Specified Downgrade

Sample code of specified downgrade:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Downgrade Demo</title>
    <script src="sdk/HWLLSPlayer.js"></script>
</head>
<body>
<div id="preview" style="width: 1280px; height: 720px; position:relative;"></div>
<button id="btnStartPlay">start play</button>
<button id="btnStopPlay">stop play</button>
<script>
    const btnStartPlay = document.getElementById('btnStartPlay')
    const btnStopPlay = document.getElementById('btnStopPlay')

    const streamUrl = 'your stream url'
    const flvUrl = 'your flv url'

    const client = HWLLSPlayer.createClient()
    bindEvent(client)

    btnStartPlay.addEventListener('click', () => {
        startPlay(client, streamUrl, 'preview', {
            flvUrl: flvUrl
        })
    })
    btnStopPlay.addEventListener('click', () => {
        stopPlay(client)
    })
```

```
    function bindEvent(client) {
        client.on('Error', (error) => {
            console.log(`error: ${JSON.stringify(error)}`)
        })
        client.on('player-changed', (mediaFormat) => {
            console.log(`player changed:${mediaFormat}`)
        })
    }

    function startPlay(client, url, elementId, downgradeUrl) {
        client.startPlay(url, {
            elementId: elementId,
            downgradeUrl: downgradeUrl,
        })
    }

    function stopPlay(client) {
        client.stopPlay()
    }
</script>
</body>
</html>
```

3.6 API Reference

3.6.1 Main Entry (HWLLSPlayer)

This section describes the HWLLSPlayer APIs of the LLL web SDK.

Table 3-3 HRTC APIs

API	Description
checkSystemRequire-ments	Checks whether the browser supports the LLL web SDK.
getVersion	Obtains the SDK version number.
createClient	Creates a client object for pulling live streams. If multiple live streams need to be pulled, you need to create multiple client objects.
uploadLog	Uploads logs.
saveLog	Saves logs.
setParameter	Configures global parameters.
setLogLevel	Sets the level of logs to be printed on the console.
setReportConfig	Sets the dotting capability and the authentication policy for dotting and log uploading.
on	Registers the callback for a client object event.
off	Deregisters the callback for a client object event.

checkSystemRequirements

checkSystemRequirements(): Promise<boolean>

[Function Description]

Checks whether the browser supports the LLL web SDK.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: A Promise object is returned. The value **true** indicates that the browser is compatible with the LLL web SDK. If they are incompatible, an error is returned.

⚠ CAUTION

LLL requires the WebRTC capability but some browsers may not support WebRTC playback. In this case, you can downgrade the playback based on the specific error code (**HWLLS_ERROR_WEBRTC_UNSUPPORTED**). For details, see **SDK Usage**.

getVersion

getVersion(): string

[Function Description]

Obtains the current SDK version number.

[Request Parameters]

None

[Response Parameters]

string: current SDK version number.

createClient

createClient(): HWLLSClient

[Function Description]

Creates a client object for pulling live streams. If multiple live streams need to be pulled, you need to create multiple client objects.

[Request Parameters]

None

[Response Parameters]

client: client object for pulling streams.

createClient (Earlier than 2.10.6)

createClient(type: string): HWLLSClient | HWFlvClient | HWHlsClient

[Function Description]

Creates a client object for livestream pull. If multiple livestreams need to be pulled, you need to create multiple client objects.

[Request Parameters]

type: (optional) String type. Indicates the type of the client created for stream pull.

- Type of the client for pulling LLL streams: **webrtc**
- Type of the client for pulling FLV streams: **flv**
- Type of the client for pulling HLS streams: **hls** (reserved)

Default value: **webrtc**

[Response Parameters]

client: client object created for stream pull

⚠️ **CAUTION**

SDK 2.10.6 and later versions do not support the independent use of the HWFlvClient and HWHlsClient players. If you are using SDK of an earlier version, refer to **Client Object (HWFlvClient)** and **Client Object (HWHlsClient)**.

---

uploadLog

async uploadLog(): Promise<void>

[Function Description]

Uploads logs.

[Request Parameters]

None

[Response Parameters]

Promise<void>: If **tryCatch** is used to obtain errors as an array, error information corresponding to multiple app IDs will be returned.

saveLog

async saveLog(): Promise<Blob>

[Function Description]

Users can flexibly save logs.

[Request Parameters]

None

[Response Parameters]

Promise<Blob>: **Promise<Blob>** compressed in .zip format, which can be directly saved as a .zip file.

setParameter

setParameter(parameterKey: string, parameterValue: any): boolean

[Function Description]

Configures global parameters.

[Request Parameters]

Parameter	Value

LOADING_CONFIG	LoadingConfig is defined as:
	
	{
	
	netQualityLoading: (optional) The type is Boolean. **true** indicates that the loading effect is displayed based on the network quality. The default value is **false**, indicating that the loading effect is not displayed.
	
	netQualityLoadingThreshold: (optional) The type is number. Indicates the threshold of the **network-quality** for displaying the loading effect. The default network quality level is **5**.
	
	frameStuckLoading: (optional) The type is Boolean. **true** indicates that the loading effect is displayed based on the frame freezing duration. The default value is **false**, indicating that the loading effect is not displayed.
	
	frameStuckThreshold: (optional) The type is number. Indicates the frame freezing duration threshold for displaying the loading effect. The unit is 100 ms. The default value is **10**, indicating that the frame freezing duration is 1000 ms.
	
	}
	
	CAUTION
	You need to configure this parameter before **starting playback**.

DNS_QUERY_ENABLE	(Optional) Boolean type. **true** indicates that DNS resolution is enabled. **false** (default) indicates that DNS resolution is disabled.
ACCESS_DOMAIN	(Optional) String type. By default, this parameter is not specified and is used to configure the stream pull environment. You need to contact technical support when specifying it.
GLSB_DOMAIN	(Optional) String type. By default, this parameter is not specified and is used to configure the GSLB environment. You need to contact technical support when specifying it.

Parameter	Value
BACKGROUND_PLAY	(Optional) Boolean type. **true** indicates that background playback is enabled, and **false** (default) indicates that background playback is disabled.
AUTO_DOWNGRADE	(Optional) Boolean type. **true** (default) indicates that automatic downgrade is enabled, and **false** indicates that automatic downgrade is disabled.

[Response Parameters]

boolean: configuration parameter setting result. **true** indicates that the setting is successful. **false** indicates that the setting fails.

setLogLevel

setLogLevel(level: string): boolean

[Function Description]

Set the level of logs to be printed on the console. Otherwise, the default level is **error**.

[Request Parameters]

level: (mandatory) String type. A log level identifier.

- **none**: Disables the function of printing logs of all levels.
- **error**: Logs of the error level are printed.
- **warn**: Logs of the warn level and higher levels are printed.
- **info**: Logs of the info level and higher levels are printed.
- **debug**: Logs of the debug level and higher levels are printed.

[Response Parameters]

boolean: log level setting result. **true** indicates that the log level is set successfully. **false** indicates that the log level fails to be set.

setReportConfig

setReportConfig(reportConfig:ReportConfig):boolean

[Function Description]

Sets the dotting capability and the authentication policy for dotting and log uploading.

[Request Parameters]

reportConfig: (mandatory) ReportConfig type. **reportConfig** is defined as follows:

- **enable**: (mandatory) The value is of the Boolean type. **true** (default) indicates that dotting is enabled, and **false** indicates that dotting is disabled.

- **tokenConfig**: (optional) The object definition is as follows:
  - **enable**: The value is of the Boolean type. **true** indicates that authentication is enabled, and **false** (default) indicates that authentication is disabled.
  - **tokenInfo**: The value is of the array type. The array contains the **ReportTokenInfo** class. **ReportTokenInfo** is defined as follows:
    - **appid**: The value is of the string type.
    - **expTimestamp**: The value is of the string type, indicating the expiration timestamp. The value is the current UNIX timestamp plus the authentication expiration time. The recommended value is 7200 seconds. The maximum value is 43200 seconds, that is, 12 hours.

      For example, if the current UNIX timestamp is 1708531200 and the authentication expiration time is 7200 seconds, the expiration timestamp is 1708538400, indicating that the **verification string** expires at 02:00:00 on February 22, 2024.
    - **token**: The value is of the string type. It is a string generated by **hmac_sha256**. **hmac_sha256 (***shared key***, *expiration timestamp* + *appID*)** The shared key is controlled and obtained by the user.

[Response Parameters]

A Boolean value is returned. Returns **true** if the value is set successfully; returns **false** otherwise.

If the authentication policy is enabled, the actual request status is obtained by registering the **Error** callback through the **on** function.

on

```
on(event: string, handler: function, withTimeout?: boolean): void
```

[Function Description]

Registers the callback for a client object event.

[Request Parameters]

- **event**: (mandatory) event name, which is of the string type. It is used to register the **Error** event, listen for dotting, or upload error information in logs.
- **handler**: (mandatory) event processing method. The type is function.

  The **handler** method transfers the **errorInfo** parameter, which is an object defined as follows:

  {

  code: (mandatory) error code. The type is number.

  message: (mandatory) error description. The type is string.

  appid: (mandatory) error log ID. The type is string.

  }
- **withTimeout**: (optional) indicates whether a timeout error is reported. The value is of the Boolean type.

[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, which is of the string type. It is used to deregister the **Error** event.
- **handler**: (mandatory) event handling method. The type is function.

[Response Parameters]

None

3.6.2 Client Object (HWLLSClient)

This section describes the HWLLSClient APIs of the LLL web SDK.

Table 3-4 HRTC APIs

API	Description
startPlay	Starts playback. The client obtains the corresponding live stream from the server based on the entered URL.
switchPlay	Quickly switches to the next stream.
stopPlay	Stops the playback.
replay	Replays the video.
startPlayCustomize	Customizes the playback.
resume	Resumes the playback.
pause	Pauses the playback.
pauseVideo	Pauses the video playback.
resumeVideo	Resumes the video playback.
pauseAudio	Pauses the audio playback.
resumeAudio	Resumes the audio playback.
setPlayoutVolume	Sets the playback volume.
getPlayoutVolume	Obtains the audio volume.
muteAudio	Mutes a site or all sites.
streamStatistic	Specifies whether to enable stream statistics.

API	Description
enableStreamStateDetection	Enables or disables media stream status detection.
on	Registers the callback for a client object event.
off	Deregisters the callback for a client object event.
destroyClient	Destroys a client object.
fullScreenToggle	Enables/Disables full-screen display.

startPlay

startPlay(url: string, options: StartPlayOptions): Promise<void>

[Function Description]

Starts playback. The client obtains the corresponding live stream from the server based on the entered URL.

[Request Parameters]

- **url**: (mandatory) String type. A streaming URL is in the format of **webrtc://**
  {domain}/*{AppName}*/*{StreamName}*.
    - **webrtc://**: The value is fixed, indicating that streams are pulled using WebRTC.
    - *domain*: indicates the streaming domain name. Use the streaming domain name registered with Huawei Cloud.
    - *AppName*: indicates the application name. Use the application name registered with Huawei Cloud.
    - *StreamName*: indicates the stream name, which must be the same as the pushed stream name.
- **options**: (mandatory) Playback configuration parameter. The value is of StartPlayOptions type. The definition of StartPlayOptions is as follows: {
    - **elementId**: (mandatory) indicates the playback DOM ID.
    - **objectFit**: (optional) String type. Default value: **cover**. 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.

> **⚠ CAUTION**
>
> The browser on a mobile phone may create a control to overwrite the SDK player, resulting in invalid configuration. For example, this may occur when an HLS/FLV URL is used for playback after downgrade on an OPPO browser.

- **muted**: (optional) Boolean type. **true** indicates muted; **false** indicates unmuted. The default value is **false**.

- **sessionId**: (optional) String type. Indicates the unified ID of a complete session.

- **showLoading**: (optional) Boolean type. Indicates whether to enable the loading display effect. **true** indicates that the loading display effect is enabled. The default value is **false**. When this parameter is set to **true**, the loading effect upon playback start will also be enabled. The loading effect upon buffering during playback needs to be set based on the **LOADING_CONFIG** in the **setParameter** API.

> **⚠ CAUTION**
>
> - The QQ browser on Android devices does not support this function.
> - You are advised to leave **showLoading** blank or set it to **false**.

- **autoPlay**: (optional) Boolean type. **true** indicates that the autoplay function is enabled. **false** indicates that the autoplay function needs to be manually triggered. The default value is **true**.

- **poster**: (optional) The object definition is as follows: {

  - **url**: (optional) String type. Sets the complete address of the thumbnail image to be played. The image must be in JPG, PNG, or static GIF format, the size cannot exceed 1 MB, and the resolution cannot exceed 1920 x 1080. The file name cannot contain Chinese characters.

  - **mode**: (optional) String type. The default value is **crop**. The following enumerated values are supported: {

    - **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.

    - **crop**: original size of the video thumbnail. If the thumbnail exceeds the playback area, the excess part is cropped. Otherwise, the thumbnail is displayed in the middle of the playback window.

    }

- **startEnable**: (optional) Boolean type. Indicates whether to display the thumbnail when the playback starts. The options are **true** and **false**. The default value is **false**. This parameter takes effect only in non-autoplay scenarios.

- **pauseEnable**: (optional) Boolean type. Indicates whether to display the thumbnail on the playback page when the video is paused. The options are **true** and **false**. The default value is **false**.

}

– **webrtcConfig**: (optional) WebRTCConfig type. Specifies parameters for pulling streams of a specified media type. WebRTCConfig is defined as follows: {

- **receiveVideo**: (optional) Boolean type. Pulls a video for playback. **true** indicates that a video is pulled for playback, and **false** indicates that a video is not pulled for playback. The default value is **true**. This parameter and **receiveAudio** cannot be set to **false** at the same time.

- **receiveAudio**: (optional) Boolean type. Indicates whether to pull an audio for playback. **true** indicates that an audio is pulled for playback, and **false** indicates that an audio is not pulled for playback. The default value is **true**. This parameter and **receiveVideo** cannot be set to **false** at the same time.

}

– **domainPolicy**: (optional) DomainPolicy type. Specifies the policy of an access domain name. This setting takes effect only when **schedulePolicy** is set to **DNS**. **DomainPolicy** is defined as follows: {

- **0**: (optional) Number type. Indicates that the user-defined domain name is used. The default value is **0**.

- **1**: (optional) Number type. Indicates that the public access domain name is used.

}

– **autoDowngrade**: (optional) Boolean type. **true** indicates that automatic downgrade is enabled, and **false** (default) indicates not.

– **downgradeUrl**: (optional) The object definition is as follows: {

- **hlsUrl?**: (optional) String type. Identifies the downgraded HLS streaming URL.

- **flvUrl?**: (optional) String type. Identifies the downgraded FLV streaming URL.

}

[Response Parameters]

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

> **⚠ CAUTION**
>
> When the LLL service is faulty, the error code **HWLLS_BUSINESS_DOWNGRADE** can be used to downgrade the playback. For details, see **SDK Usage**.

switchPlay

switchPlay(url: string, options: StartPlayOptions): Promise<void>

[Function Description]

After the playback has been started, the system quickly switches to the next stream.

[Request Parameters]

- **url**: (mandatory) Ingest URL. A streaming URL is in the format of **webrtc://{domain}/{AppName}/{StreamName}**.
    - **webrtc://**: The value is fixed, indicating that streams are pulled using WebRTC.
    - *domain*: indicates the streaming domain name. Use the streaming domain name registered with Huawei Cloud.
    - *AppName*: indicates the application name. Use the application name registered with Huawei Cloud.
    - *StreamName*: indicates the stream name, which must be the same as the pushed stream name.
- **options**: (optional) Playback configuration parameter. The value is of StartPlayOptions type. If this parameter is not carried, the **options** data carried in the first playback start request is reused. The definition of StartPlayOptions is as follows: {
    - **elementId**: (mandatory) indicates the playback DOM ID.
    - **objectFit**: (optional) String type. Default value: **cover**. 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) Boolean type. **true** indicates muted; **false** indicates unmuted. The default value is **false**.

- **sessionId**: (optional) String type. Indicates the unified ID of a complete session.

- **showLoading**: (optional) Boolean type. Indicates whether to enable the loading display effect. **true** indicates that the loading display effect is enabled. The default value is **false**. When this parameter is set to **true**, the loading effect upon playback start will also be enabled. The loading effect upon buffering during playback needs to be set based on the **LOADING_CONFIG** in the **setParameter** API.

- **autoPlay**: (optional) Boolean type. **true** indicates that the autoplay function is enabled. **false** indicates that the autoplay function needs to be manually triggered. The default value is **true**.

- **poster**: (optional) The object definition is as follows: {

  - **url**: (optional) String type. Sets the complete address of the thumbnail image to be played. The image must be in JPG, PNG, or static GIF format, the size cannot exceed 1 MB, and the resolution cannot exceed 1920 x 1080. The file name cannot contain Chinese characters.

  - **mode**: (optional) String type. The default value is **cover**. The following enumerated values are supported: {

    - **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.

    - **crop**: original size of the video thumbnail. If the thumbnail exceeds the playback area, the excess part is cropped. Otherwise, the thumbnail is displayed in the middle of the playback window.

    }

  - **startEnable**: (optional) Boolean type. Indicates whether to display the thumbnail when the playback starts. The options are **true** and **false**. The default value is **false**. This parameter takes effect only in non-autoplay scenarios.

  - **pauseEnable**: (optional) Boolean type. Indicates whether to display the thumbnail on the playback page when the video is paused. The options are **true** and **false**. The default value is **false**.

  }

- **webrtcConfig**: (optional) WebRTCConfig type. Specifies parameters for pulling streams of a specified media type. WebRTCConfig is defined as follows: {

  - **receiveVideo**: (optional) Boolean type. Pulls a video for playback. **true** indicates that a video is pulled for playback, and **false** indicates that a video is not pulled for playback. The default value is **true**. This parameter and **receiveAudio** cannot be set to **false** at the same time.

  - **receiveAudio**: (optional) Boolean type. Indicates whether to pull an audio for playback. **true** indicates that an audio is pulled for playback, and **false** indicates that an audio is not pulled for

playback. The default value is **true**. This parameter and **receiveVideo** cannot be set to **false** at the same time.

}

– **schedulePolicy**: (optional) SchedulePolicy type. Specifies the access scheduling policy. The definition of **SchedulePolicy** is as follows: {

  ▪ **DNS**: (optional) String type. Indicates that the domain name is resolved by DNS for being accessed. The default value is **DNS**.

  ▪ **HTTPDNS**: (optional) String type. Indicates that **HTTPDNS** is used for the access domain name.

  }

– **domainPolicy**: (optional) DomainPolicy type. Specifies the policy of an access domain name. This setting takes effect only when **schedulePolicy** is set to **DNS**. **DomainPolicy** is defined as follows: {

  ▪ **0**: (optional) Number type. Indicates that the user-defined domain name is used. The default value is **0**.

  ▪ **1**: (optional) Number type. Indicates that the public access domain name is used.

  }

[Response Parameters]

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

⚠️ **CAUTION**

When the LLL service is faulty, the error code **HWLLS_BUSINESS_DOWNGRADE** can be used to downgrade the playback. For details, see **SDK Usage**.

---

stopPlay

stopPlay(): boolean

[Function Description]

Stops the playback.

[Request Parameters]

None

[Response Parameters]

boolean: result of stopping playback. The options are **true** (success) and **false** (failure).

replay

replay(): Promise<boolean>

[Function Description]

Replays the video.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: replay result. The options are **true** (success) and **false** (failure).

startPlayCustomize

startPlayCustomize(url: string, options: StartPlayOptions): Promise<boolean>

[Function Description]

Customizes the playback.

[Request Parameters]

- **url**: (mandatory) Ingest URL. Streaming URL, which is a customized URL for playback.

- **options**: (optional) Playback configuration parameter. The value is of StartPlayOptions type. If this parameter is not carried, the **options** data carried in the first playback start request is reused. The definition of StartPlayOptions is as follows: {

    - **elementId**: (mandatory) indicates the playback DOM ID.

    - **objectFit**: (optional) String type. Default value: **cover**. 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) Boolean type. **true** indicates muted; **false** indicates unmuted. The default value is **false**.

    - **sessionId**: (optional) String type. Indicates the unified ID of a complete session.

    - **showLoading**: (optional) Boolean type. Indicates whether to enable the loading display effect. **true** indicates that the loading display effect is enabled. The default value is **false**. When this parameter is set to **true**, the loading effect upon playback start will also be enabled. The loading effect upon buffering during playback needs to be set based on the **LOADING_CONFIG** in the **setParameter** API.

– **autoPlay**: (optional) Boolean type. **true** indicates that the autoplay function is enabled. **false** indicates that the autoplay function needs to be manually triggered. The default value is **true**.

– **poster**: (optional) The object definition is as follows: {

▪ **url**: (optional) String type. Sets the complete address of the thumbnail image to be played. The image must be in JPG, PNG, or static GIF format, the size cannot exceed 1 MB, and the resolution cannot exceed 1920 x 1080. The file name cannot contain Chinese characters.

▪ **mode**: (optional) String type. The default value is **cover**. The following enumerated values are supported: {

○ **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.

○ **crop**: original size of the video thumbnail. If the thumbnail exceeds the playback area, the excess part is cropped. Otherwise, the thumbnail is displayed in the middle of the playback window.

}

▪ **startEnable**: (optional) Boolean type. Indicates whether to display the thumbnail when the playback starts. The options are **true** and **false**. The default value is **false**. This parameter takes effect only in non-autoplay scenarios.

▪ **pauseEnable**: (optional) Boolean type. Indicates whether to display the thumbnail on the playback page when the video is paused. The options are **true** and **false**. The default value is **false**.

}

– **webrtcConfig**: (optional) WebRTCConfig type. Specifies parameters for pulling streams of a specified media type. WebRTCConfig is defined as follows: {

▪ **receiveVideo**: (optional) Boolean type. Pulls a video for playback. **true** indicates that a video is pulled for playback, and **false** indicates that a video is not pulled for playback. The default value is **true**. This parameter and **receiveAudio** cannot be set to **false** at the same time.

▪ **receiveAudio**: (optional) Boolean type. Indicates whether to pull an audio for playback. **true** indicates that an audio is pulled for playback, and **false** indicates that an audio is not pulled for playback. The default value is **true**. This parameter and **receiveVideo** cannot be set to **false** at the same time.

}

– **schedulePolicy**: (optional) SchedulePolicy type. Specifies the access scheduling policy. The definition of **SchedulePolicy** is as follows: {

▪ **DNS**: (optional) String type. Indicates that the domain name is resolved by DNS for being accessed. The default value is **DNS**.

▪ **HTTPDNS**: (optional) String type. Indicates that **HTTPDNS** is used for the access domain name.

}

– **domainPolicy**: (optional) DomainPolicy type. Specifies the policy of an access domain name. This setting takes effect only when **schedulePolicy** is set to **DNS**. **DomainPolicy** is defined as follows: {

▪ **0**: (optional) Number type. Indicates that the user-defined domain name is used. The default value is **0**.

▪ **1**: (optional) Number type. Indicates that the public access domain name is used.

}

[Response Parameters]

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

resume

resume(): Promise<boolean>

[Function Description]

Resumes the playback.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: result of resuming the audio/video playback. The options are **true** (success) and **false** (failure).

pause

pause(): boolean

[Function Description]

Pauses the audio/video playback.

[Request Parameters]

None

[Response Parameters]

boolean: playback pause result. The options are **true** (success) and **false** (failure).

pauseVideo

pauseVideo(): boolean

[Function Description]

This parameter is used to pause a video. After the video is paused, it freezes.

[Request Parameters]

None

[Response Parameters]

boolean: result of pausing video playback. The options are **true** (success) and **false** (failure).

resumeVideo

resumeVideo(): Promise<boolean>

[Function Description]

Resumes the video playback.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: result of resuming the video playback. The options are **true** (success) and **false** (failure).

pauseAudio

pauseAudio(): boolean

[Function Description]

Pauses the audio playback.

[Request Parameters]

None

[Response Parameters]

boolean: result of pausing the audio playback. The options are **true** (success) and **false** (failure).

resumeAudio

resumeAudio(): Promise<boolean>

[Function Description]

Resumes the audio playback.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: result of resuming the audio playback. The options are **true** (success) and **false** (failure).

setPlayoutVolume

setPlayoutVolume(volume: number): boolean

[Function Description]

Sets the audio volume. This API is not supported on iOS.

[Request Parameters]

volume: (mandatory) audio volume. The value is a number ranging from 0 to 100.

[Response Parameters]

boolean: whether the operation is successful. The options are **true** (success) and **false** (failure).

getPlayoutVolume

getPlayoutVolume(): number

[Function Description]

Obtains the audio volume.

[Request Parameters]

None

[Response Parameters]

number: volume value. The value ranges from 0 to 100.

muteAudio

muteAudio(isMute: boolean): void

[Function Description]

Mutes.

[Request Parameters]

- **isMute**: (mandatory) indicates whether the device is muted. The value is of the Boolean type. **true** indicates that the device is muted, and **false** indicates that the device is unmuted.

[Response Parameters]

None

streamStatistic

streamStatistic(enable: boolean, interval: number): void

[Function Description]

Specifies whether to enable stream statistics.

[Request Parameters]

- **enable**: (mandatory) Specifies whether to enable stream statistics. The value is of the Boolean type. The value **true** indicates that stream statistics are enabled.

- **interval**: (mandatory) Specifies the statistics interval, in seconds. The value is a number ranging from 1 to 60. The default value is **1**.

[Response Parameters]

None

enableStreamStateDetection

enableStreamStateDetection(enable: boolean, interval: number, interruptRetry:StreamInterruptRetry): boolean

[Function Description]

Enables or disables media stream status detection. After the function is enabled, the system can detect whether the stream has been interrupted at the stream push device.

[Request Parameters]

- **enable**: (mandatory) whether to enable media stream status detection. The value is of the Boolean type. **true** indicates enabled; **false** (default value) indicates disabled.

- **interval**: (mandatory) Specifies the interval in seconds. The value is a number ranging from 1 to 60. This parameter is used to determine when there is no media stream. The default value is **3** (recommended).

- **interruptRetry**: (optional) Parameter for configuring playback retry upon stream interruption. The value is of StreamInterruptRetry type. The definition of StreamInterruptRetry is as follows: {

    enable: The value is of the Boolean type. Attempt for automatically resuming playback is enabled after stream interruption. The default value is **false**, indicating that attempt for automatically resuming playback is disabled.

    retryInterval: retry interval for stream pulling, in seconds. The value type is number. The value ranges from 10 to 60 and defaults to **30**.

    retryTimes: maximum number of retry times for resuming playback. The value type is number. The minimum value is **1** and the default value is **30**.

    }

[Response Parameters]

boolean: whether the operation is successful. The options are **true** (success) and **false** (failure).

> ⚠ **CAUTION**
>
> The QQ browser on Android devices does not support this function.

---

on

on(event: string, handler: function, withTimeout?: boolean): void

[Function Description]

Registers the callback for a client object event.

[Request Parameters]

- **event**: (mandatory) event name. The type is string. For details, see **HWLLSClientEvent**.

- **handler**: (mandatory) event processing method. The type is function.
- **withTimeout**: (optional) indicates whether a timeout error is reported. The value is of the Boolean type.

[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 **HWLLSClientEvent**.
- **handler**: (mandatory) event processing method. The type is function.

[Response Parameters]

None

destroyClient

destroyClient(): void

[Function Description]

Destroys a client object.

[Request Parameters]

None

[Response Parameters]

None

fullScreenToggle

fullScreenToggle(isExit: boolean): void

[Function Description]

Enables/Disables full-screen display.

[Request Parameters]

isExit: (mandatory) Boolean type. The default value is **false**.

[Response Parameters]

None

3.6.3 Client Event Notification (HWLLSClientEvent)

This section describes the HWLLSClientEvent APIs of the LLL web SDK.

Table 3-5 HWLLSClientEvent APIs

API	Description
media-statistic	Media statistics event.
network-quality	Network quality report event.
video-broken	Video streaming interruption event (waiting for recovery).
audio-broken	Audio streaming interruption event (waiting for recovery).
video-recovery	Video streaming resumption event (playback resumed).
audio-recovery	Audio streaming resumption event (playback resumed).
audio-start	Audio playback start event.
video-start	Video playback start event.
video-stuck	Video playback pause event.
fullscreen-status-changed	Full-screen view event.
player-changed	Playback downgrade event.
Error	Client error event.

⚠️ **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.

media-statistic

[Event Description]

Media statistics event. This event is used together with the **streamStatistic** method.

[Callback Parameters]

StatisticInfo: media statistics. The value type is StatisticInfo.

StatisticInfo is defined as follows: {

- video: {

  mediaType: media type

  frameRate: video frame rate. The type is number.

  width: video width. The type is number.

height: video height. The type is number.

jitter: jitter. The type is number.

bitRate: bitrate (in kbit/s). The type is number.

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.

}

- audio: {

mediaType: media type

jitter: jitter. The type is number.

bitRate: bitrate (in kbit/s). The type is number.

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.

}

}

network-quality

[Event Description]

Network quality report event.

[Callback Parameters]

NetworkQualityTypes: network quality details. The type is NetworkQualityTypes.

The enumerated values of **NetworkQualityTypes** are as follows:

- **NETWORK_QUALITY_UNKNOW = 0**: The network quality is unknown.
- **NETWORK_QUALITY_GREAT = 1**: The network quality is excellent.
- **NETWORK_QUALITY_GOOD = 2**: User experience is almost the same as that of value **1**, but the bitrate may be slightly lower.
- **NETWORK_QUALITY_DEFECTS = 3**: User experience is defective but the watching is not affected.
- **NETWORK_QUALITY_WEAK = 4**: The network quality is poor and the video is not smooth.
- **NETWORK_QUALITY_BAD = 5**: The network quality is so poor that user experience is severely affected.
- **NETWORK_QUALITY_DISCONNECT = 6**: The network quality is poor and even disconnection occurs. The video cannot be watched.

video-broken

[Event Description]

Video streaming interruption event (waiting for recovery).

[Callback Parameters]

None

audio-broken

[Event Description]

Audio streaming interruption event (waiting for recovery).

[Callback Parameters]

None

video-recovery

[Event Description]

Video streaming resumption (non-EOF) event (playback resumed).

[Callback Parameters]

None

audio-recovery

[Event Description]

Audio streaming resumption (non-EOF) event (playback resumed).

[Callback Parameters]

None

audio-start

[Event Description]

Audio playback start event.

[Callback Parameters]

None

video-start

[Event Description]

Video playback start event.

[Callback Parameters]

None

video-stuck

[Event Description]

Video playback pause event.

[Callback Parameters]

Boolean value. **True** indicates paused, and **False** indicates not paused.

fullscreen-status-changed

[Event Description]

Full-screen view event.

[Callback Parameters]

- **isFullScreen**: indicates whether to enable full-screen display
- **isPause**: indicates whether to stop playback

player-changed

[Event Description]

Playback downgrade event.

[Callback Parameters]

Indicates the downgrade information. The value is a string.

- **webrtc**: LLL playback
- **hls**: HLS playback
- **flv**: FLV playback

Error

[Event Description]

This event is triggered when an unrecoverable client error occurs.

[Callback Parameters]

errorInfo: (mandatory) error information. The value is of the **HwLLSError** type.

errorInfo is defined as: {

code: (mandatory) error code. The type is number.

message: (mandatory) error description. The type is string.

getCode (): (mandatory) returned error code. The type is number.

getMsg (): (mandatory) returned error description. The type is string.

}

> ⚠ **CAUTION**
>
> If the network firewall is restricted (UDP port restriction) or playback fails on LLL for multiple times, you can downgrade the playback based on the specified error code (**HWLLS_MEDIA_NETWORK_ERROR** or **HWLLS_PLAY_WEBRTC_RETRY_FAILED**). For details, see **SDK Usage**.

3.6.4 Error Codes (HwLLSError)

getCode

getCode(): number

[Function Description]

Obtains an error code.

[Request Parameters]

None

[Response Parameters]

Error code value, which is a number.

getMsg

getMsg(): string

[Function Description]

Obtains error description.

[Request Parameters]

None

[Response Parameters]

Error code description, which is a string.

3.6.5 Public IP Addresses

Table 3-6 Public IP address list

Public IP Address	Information
log-collection-new.hwcloudlive.com	Log and dotting environment address in China
log-collection-ap-southeast-3.rocket-cdn.com	Log and dotting environment address outside China
hcdnl-pull302-global-gslb.livehwc3.cn	Default GSLB environment address

3.6.6 Client Error Codes

This section describes details about the error codes reported on the LLL Web client SDK.

Table 3-7 Error code description

Class Member	Error Code	Description	Error Cause or Handling Suggestion
HWLLS_OK	0	Succeeded.	-
HWLLS_ERROR_INV ALID_URL	50000 000	Invalid URL.	Check whether the URL is correct.
HWLLS_ERROR_INV ALID_PARAMETER	50000 001	Invalid parameter.	Parameter transfer error. Check whether the input parameters of the API meet the parameter validity requirements.
HWLLS_ERROR_SER VER_CONNECT_FAIL	50000 002	Connect to the server failed.	Check whether the network status is normal or contact technical support.
HWLLS_ERROR_SER VER_NO_RESPONSE	50000 003	The server does not respond.	Contact technical support.
HWLLS_ERROR_AUT H_FAIL	50000 004	Authentication failed.	Check whether the referer validation and URL validation configurations on the server are correct.
HWLLS_ERROR_STR EAM_NOT_EXIST	50000 005	The requested stream does not exist.	Use a stream available.
HWLLS_ERROR_WE BRTC_UNSUPPORTE D	50000 006	Your browser does not support the function.	1. Use a supported browser. For details, see **Browser Adaptation**. 2. Downgrade the FLV or HLS livestreaming.
HWLLS_MEDIA_NET WORK_ERROR	50000 007	Abnormal media network connection	Check whether the network status and firewall configuration are correct, or downgrade the FLV or HLS livestreaming.
HWLLS_ERROR_BAD _REQUEST	50000 008	Abnormal domain name configuration	Check whether the domain name of the requested stream is correct or contact technical support.
HWLLS_ERROR_STR EAM_INVALID_PARA METER	50000 009	Incorrect stream information	Check whether the URL of the requested stream is correct.
HWLLS_INTERNAL_ ERROR	50000 020	Other internal errors	Contact technical support.

Class Member	Error Code	Description	Error Cause or Handling Suggestion
HWLLS_BUSSINESS_ DOWNGRADE	50000 021	The service needs to be downgraded.	You are advised to downgrade the FLV or HLS livestreaming.
HWLLS_PLAY_WEBR TC_RETRY_FAILED	50000 022	After the LLL playback is interrupted, multiple attempts for resuming playback failed.	You are advised to downgrade the FLV or HLS livestreaming.
HWLLS_PLAY_FLV_R ETRY_FAILED	50000 023	After the FLV playback is interrupted, multiple attempts for resuming playback failed.	Contact technical support.
HWLLS_PLAY_HLS_R ETRY_FAILED	50000 024	After the HLS playback is interrupted, multiple attempts for resuming playback failed.	Contact technical support.
HWLLS_ERROR_LIVE _UNSUPPORTED	50000 030	The content format is not supported by the browser.	Contact technical support.
HWLLS_ERROR_UN EXPECTED_EOF	50000 031	The network EOF is abnormal during content playback.	Try again.
HWLLS_ERROR_ME DIA_ERROR	50000 032	The media content playback is abnormal.	Try again or contact technical support.
HWLLS_ERROR_REP ORT_TOKEN_ERROR	50000 033	Token exceptions occur during dotting and log uploading.	Contact technical support.

Class Member	Error Code	Description	Error Cause or Handling Suggestion
HWLLS_PLAY_NOT_ALLOW	51000 000	The playback permission is restricted. You need to manually trigger the playback.	Due to the restrictions of the browser's automatic playback security policy, this error code is returned when the browser directly starts the app and playback starts. According to this error code, you need to manually trigger the UI control on the page at the application layer and call the **replay** API to resume playback. **NOTICE** If Safari is used, perform the following operations: Open Safari, choose **Preferences** > **Websites** > **Auto-Play**, select a website and enable **Allow All Auto-Play** for it.
HWLLS_PLAY_TIME OUT	51000 001	Playback timeout. No valid frame data is pulled within 3s for LLL (10s for FLV/HLS).	Check the stream push status or contact technical support.

3.7 FAQs

- **Can the Huawei Cloud LLL web SDK be integrated if the service app can use only the HTTP?**

  The SDK can be integrated with some browsers such as Chrome but integration is not recommended. The browser compatibility is identified based on the WebRTC object exposed by the browser. When a non-HTTPS protocol is used, the object may not exist.

- **Why can't I use the Huawei Cloud LLL web SDK in Firefox?**

  Before using the Firefox browser, you need to install the H.264 codec plug-in. Enter **about:addons** in the address box of the browser. The plug-in installation page is displayed. Check whether the H.264 plug-in has been installed. If not, install it on the page.

- **What are the possible causes if the Huawei Cloud LLL web SDK does not work after being integrated?**

  - Check whether the user-defined domain names have been configured, such as ingest and streaming domain names, and check whether the HTTPS certificate is valid.

- Check whether the stream push end and stream is normal.
  - Check whether the streaming URL is correct, for example, appName and streamName.
  - Check whether the network connection is normal and whether the network firewall configuration is restricted. For example, check whether UDP ports 8000 to 8063 are bypassed.

- **Which browsers are supported by the Huawei Cloud LLL web SDK?**

  For details about supported browsers, see **Browser Adaptation**.

- **Why does stream pull on the Huawei Cloud LLL web SDK fail after receiving pushed streams?**

  Check whether the stream push encoding parameter of the stream push device is H.264+ without B-frames. Currently, the Huawei Cloud LLL web SDK supports only H.264+ streams without B-frames. As a result, if the original stream is H.265 or contains B-frames, you need to configure the corresponding transcoding template on the tenant console in advance and enable the transcoding service. However, this will introduce extra transcoding delay and fees. You are advised to push H.264+ streams without B-frames. You can adjust the video encoding parameters of the streaming software (such as OBS) to remove B-frames. If OBS is used to push streams, you can disable B-frames, as shown in the following figure.
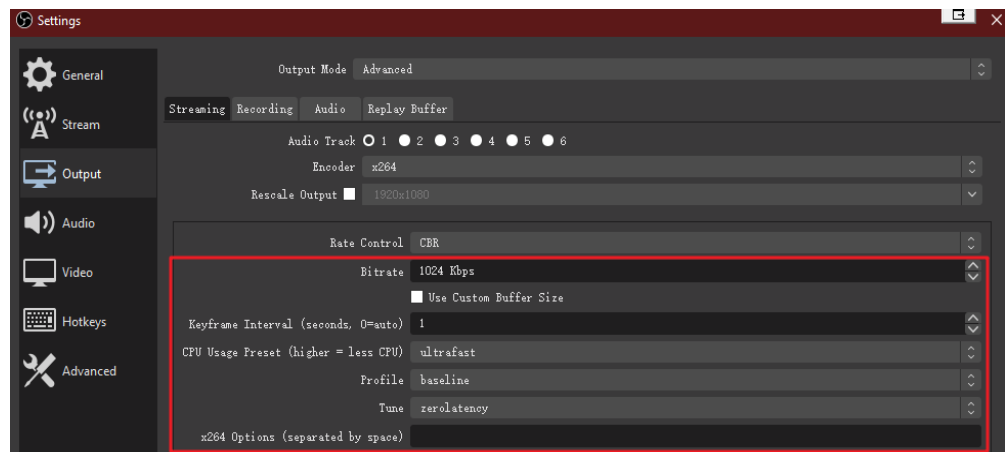
- **What do I do if the error message "NotAllowedError:xxx?" is displayed during playback using the Huawei Cloud LLL web SDK?**

  Due to the restrictions of the browser's automatic playback security policy, this error code is returned when the browser directly starts the app and playback starts. According to this error code, you need to manually trigger the UI control on the page at the application layer and call the **replay** API to resume playback.

- **How do I obtain token information when the authentication policy is enabled?**

  If the authentication policy is not enabled, service functions still work, and dotting and log upload are not affected.

  Enabling the authentication policy maintains the security of dotting data and uploaded logs.

  If the authentication policy is required, **submit a service ticket** to contact technical support and obtain the app ID and token.

- **How do I configure stream pull parameters?**

  Call **startPlay** to start pulling streams. The field **elementId** in **options** is mandatory.

  - **elementId**: container ID, which specifies the container for adapted video playback. Generally, the ID of the div tag is used.

  Other fields are optional:

  - **objectFit**: rendering mode, which can be set to **contain**, **cover**, and **fill**
  - **muted**: whether to mute the playback

- **What if audio autoplay blocking occurs?**

  Audio autoplay blocking can occur when audio is played without any interaction on the GUI. You can obtain related information by listening to **Error** events. For details, see **Best Practices**.

  ```
  client.on('Error', (error) => {
      if (error.errCode === 51000000) {
          // When audio autoplay blocking occurs, perform interactive operations and call the replay API.
      }
  })
  ```

- **How do I use loading to load animation?**

  You can use **loading** to load animation when the playback starts, video freezes, or network quality is poor. Methods:

  ```
  HWLLSPlayer.setParameter('LOADING_CONFIG', {
      netQualityLoading: true, // Display loading based on the network quality.
      netQualityLoadingThreshold: 5, // Display the loading threshold. The default value is 5.
      frameStuckLoading: true, // Display loading based on the freeze frame duration.
      frameStuckThreshold: 10, // Freeze frame duration threshold. The unit is 100 ms. The value 10
  indicates 1,000 ms.
  })
  ```

- **How do I use a video thumbnail?**

  The thumbnail settings are in the parameters of the **startPlay** API. Method:

  ```
  const options = {
      …
      poster: {
          url: // Poster URL
          mode: crop, // Poster filling mode, which can be set to fill or crop.
          startEnable: true, // Whether to display the thumbnail when the playback starts. This parameter
  takes effect only when the playback does not start automatically.
          pauseEnable: true, // Whether to display the thumbnail when the playback is paused.
      }
      …
  }

  client.startPlay(streamUrl, options)
  ```

- **How do I obtain statistics?**

  See **media-statistic**.

  Enable the stream statistics function:

  ```
  client.streamStatistic(true, 1)
  ```

  Obtain the statistics through the listening event callback:

  ```
  client.on('media-statistic', (statisticInfo) => {
      const audioStatisticInfo = statisticInfo.audio
      const videoStatisticInfo = statisticInfo.video
  })
  ```

- **How do I fix a black screen?**

  When there is a black screen, check whether the stream push works, including:

- Whether the video encoding format is H.264.
- Whether the video contains B-frames.
- Whether only audio is pushed.

If the stream push works, you can use FLV to pull streams to check the video. If the video can be played, **submit a service ticket**.

If a black screen occurs during the playback, perform the following operations:

- Check the audio. If the audio can be played, check the stream push device.
- If both the audio and video can be played, the stream interruption may be caused by a network issue. When the networking is restored, you can call the API for starting playback again or retry stream pull upon stream interruption.

- **How do I retry stream pull upon stream interruption?**

  This function allows automatically retrying stream pull upon stream interruption to improve user experience.

  Method for enabling the function:

  ```
  client.enableStreamStateDetection(
      true, // Switch of stream check.
      3, // Check interval, in second.
      {
          enable: true, // Switch of retry.
          retryInterval: 30, // Retry interval, in second.
          retryTimes: 30, // Retry times
      }
  )
  ```

- **How do I handle compatibility issues?**

  The browsers of some devices may not support WebRTC stream pull. In this case, you can use the downgrade policy.

- **What if functions such as full-screen operations do not work?**

  Check whether the Client is a single instance. When calling the Client method, you should use the correct Client. If a development framework such as Vue is used, you are advised not to set the Client instance to responsive data.

- **UHD video playback may not work on some mobile phones.**

  Playing UHD livestreams (higher than 4K and 8 Mbit/s) on some low-performance mobile phones (such as the Android 9 P20) may cause a black screen.

- **How do I use the downgrade function?**

  There are two downgrade methods: automatic downgrade and specified downgrade. For details, see **Best Practices**.

  - Automatic downgrade: default method of the SDK. When WebRTC stream pull is not supported, the streaming URL is automatically converted to an FLV or HLS URL.
  - Specified downgrade: Specify the URL used after downgrade in the parameters of the **startPlay** API.
    ```
    const options = {
        ...
        downgradeUrl: {
            hlsUrl: // HLS URL.
            flvUrl: // FLV URL.
    ```

```
    }
    …
}
    client.startPlay(streamUrl, options)
```

- **What if the UI display is abnormal due to browser hijacking?**

  Symptom: The browser may hijack the video player on the web page and add some UIs. As a result, the display does not meet the expectation. For example, the customized UI cannot be displayed due to hijacking and creation of a new playback layer, the object-fit setting does not take effect, or an ad is played when the playback is paused.

  Solution: Use the configuration solution provided by the browser vendor.

- **Why is a black screen displayed or the aspect ratio incorrect when the video is initialized but not played?**

  Cause: When the video is not automatically played, the browser determines the image to be displayed after the initialization is complete. The image may not be displayed or the image display is below expectations.

  Solution: Configure a poster and specify the thumbnail image displayed when the playback is not automatically played. For details, see the **poster** parameter in **startPlay**.

3.8 Change History

Table 3-8 Change history

Released On	Description
2024-02-17	1. **Event notifications** of playback exceptions were optimized and more **sample codes** were added. 2. Automatic downgrade policies were optimized. 3. The downgrade logic was optimized. The independent FLV and HLS players are no longer supported. Old documents were moved to **Appendixes**. 4. Two items were added in **FAQs**.
2024-12-02	Updates of the LLL web SDK: - Automatic downgrade and specified downgrade are supported. - Audio can be played in the background on mobile devices. - Device compatibility issues were resolved. - Multi-instance video tag ID is supported.
2024-11-12	**Best Practices** and related **Sample Code** were added.
2024-09-30	This issue is the first official release.

3.9 Appendixes

3.9.1 Client Object (HWFlvClient)

This section describes the HWFlvClient APIs of the LLL web SDK.

Table 3-9 HRTC APIs

API	Description
startPlay	Starts playback. The client obtains the corresponding live stream from the server based on the entered URL.
switchPlay	Quickly switches to the next stream.
stopPlay	Stops the playback.
replay	Replays the video.
resume	Resumes the playback.
pause	Pauses the playback.
pauseVideo	Pauses the video playback.
resumeVideo	Resumes the video playback.
pauseAudio	Pauses the audio playback.
resumeAudio	Resumes the audio playback.
setPlayoutVolume	Sets the playback volume.
getPlayoutVolume	Obtains the audio volume.
muteAudio	Mutes.
streamStatistic	Specifies whether to enable stream statistics.
enableStreamStateDetection	Enables or disables media stream status detection.
destroyClient	Destroys a client object.
fullScreenToggle	Enables/Disables full-screen display.

startPlay

startPlay(url: string, options: StartPlayOptions): Promise<void>

[Function Description]

Starts playback. The client obtains the corresponding live stream from the server based on the entered URL.

[Request Parameters]

- **url**: (mandatory) String type. Ingest URL that ends with **flv**.

- **options**: (optional) StartPlayOptions type. If this parameter is not carried, the **options** data carried in the first playback start request is reused. The definition of StartPlayOptions is as follows: {

  – **elementId**: (mandatory) indicates the playback DOM ID.

  – **objectFit**: (optional) String type. Default value: **cover**. The following enumerated values are supported:

    ■ **contain**: prioritizes the display of all video content. 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**: prioritizes the filling of the window. 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.

⚠️ **CAUTION**

The browser on a mobile phone, such as OPPO, may create a control to overwrite the SDK player, resulting in invalid configuration.

  – **muted**: (optional) Boolean type. **true** indicates muted; **false** indicates unmuted. The default value is **false**.

  – **sessionId**: This parameter does not need to be transferred.

  – **showLoading**: (optional) Boolean type. Indicates whether to enable the loading display effect. **true** indicates that the loading display effect is enabled. The default value is **false**. When this parameter is set to **true**, the loading effect upon playback start will also be enabled. The loading effect upon buffering during playback needs to be set based on the **LOADING_CONFIG** in the **setParameter** API.

  – **autoPlay**: (optional) Boolean type. **true** indicates that the autoplay function is enabled. **false** indicates that the autoplay function needs to be manually triggered. The default value is **true**.

  – **poster**: (optional) The object definition is as follows: {

    ■ **url**: (optional) String type. Sets the complete address of the thumbnail image to be played. The image must be in JPG, PNG, or static GIF format, the size cannot exceed 1 MB, and the resolution cannot exceed 1920 x 1080. The file name cannot contain Chinese characters.

    ■ **mode**: (optional) String type. The default value is **cover**. The following enumerated values are supported: {

○ **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.

○ **crop**: original size of the video thumbnail. If the thumbnail exceeds the playback area, the excess part is cropped. Otherwise, the thumbnail is displayed in the middle of the playback window.

}

▪ **startEnable**: (optional) Boolean type. Indicates whether to display the thumbnail when the playback starts. The options are **true** and **false**. The default value is **false**. This parameter takes effect only in non-autoplay scenarios.

▪ **pauseEnable**: (optional) Boolean type. Indicates whether to display the thumbnail on the playback page when the video is paused. The options are **true** and **false**. The default value is **false**.

}

– **webrtcConfig**: This parameter does not need to be transferred.

– **schedulePolicy**: This parameter does not need to be transferred.

– **domainPolicy**: This parameter does not need to be transferred.

– **downgradeUrl**: This parameter does not need to be transferred.

[Response Parameters]

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

switchPlay

switchPlay(url: string, options: StartPlayOptions): Promise<void>

[Function Description]

After the playback is started, the system quickly switches to the next stream.

[Request Parameters]

● **url**: (mandatory) String type. Ingest URL that ends with **flv**.

● **options**: (optional) StartPlayOptions type. If this parameter is not carried, the **options** data carried in the first playback start request is reused. The definition of StartPlayOptions is as follows: {

– **elementId**: (mandatory) indicates the playback DOM ID.

– **objectFit**: (optional) String type. Default value: **cover**. The following enumerated values are supported:

▪ **contain**: prioritizes the display of all video content. 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**: prioritizes the filling of the window. 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) Boolean type. **true** indicates muted; **false** indicates unmuted. The default value is **false**.

– **sessionId**: This parameter does not need to be transferred.

– **showLoading**: (optional) Boolean type. Indicates whether to enable the loading display effect. **true** indicates that the loading display effect is enabled. The default value is **false**. When this parameter is set to **true**, the loading effect upon playback start will also be enabled. The loading effect upon buffering during playback needs to be set based on the **LOADING_CONFIG** in the **setParameter** API.

– **autoPlay**: (optional) Boolean type. **true** indicates that the autoplay function is enabled. **false** indicates that the autoplay function needs to be manually triggered. The default value is **true**.

– **poster**: (optional) The object definition is as follows: {

- **url**: (optional) String type. Sets the complete address of the thumbnail image to be played. The image must be in JPG, PNG, or static GIF format, the size cannot exceed 1 MB, and the resolution cannot exceed 1920 x 1080. The file name cannot contain Chinese characters.

- **mode**: (optional) String type. The default value is **cover**. The following enumerated values are supported: {

  ○ **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.

  ○ **crop**: original size of the video thumbnail. If the thumbnail exceeds the playback area, the excess part is cropped. Otherwise, the thumbnail is displayed in the middle of the playback window.

  }

- **startEnable**: (optional) Boolean type. Indicates whether to display the thumbnail when the playback starts. The options are **true** and **false**. The default value is **false**. This parameter takes effect only in non-autoplay scenarios.

- **pauseEnable**: (optional) Boolean type. Indicates whether to display the thumbnail on the playback page when the video is paused. The options are **true** and **false**. The default value is **false**.

  }

– **webrtcConfig**: This parameter does not need to be transferred.

– **schedulePolicy**: This parameter does not need to be transferred.

– **domainPolicy**: This parameter does not need to be transferred.

– **downgradeUrl**: This parameter does not need to be transferred.

[Response Parameters]

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

stopPlay

stopPlay(): boolean

[Function Description]

Stops the playback.

[Request Parameters]

None

[Response Parameters]

boolean: result of stopping playback. The options are **true** (success) and **false** (failure).

replay

replay(): Promise<boolean>

[Function Description]

Replays the video.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: result of replay. The options are **true** (success) and **false** (failure).

resume

resume(): Promise<boolean>

[Function Description]

Resumes the playback.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: result of resuming the audio/video playback. The options are **true** (success) and **false** (failure).

pause

pause(): boolean

[Function Description]

Pauses the audio/video playback.

[Request Parameters]

None

[Response Parameters]

boolean: result of pausing playback. The options are **true** (success) and **false** (failure).

pauseVideo

pauseVideo(): boolean

[Function Description]

This API is not supported.

[Request Parameters]

None

[Response Parameters]

boolean: Only **false** is returned.

resumeVideo

resumeVideo(): Promise<boolean>

[Function Description]

This API is not supported.

[Request Parameters]

None

[Response Parameters]

boolean: Only **false** is returned.

pauseAudio

pauseAudio(): boolean

[Function Description]

Pauses the audio playback.

[Request Parameters]

None

[Response Parameters]

boolean: result of pausing audio playback. The options are **true** (success) and **false** (failure).

resumeAudio

resumeAudio(): Promise<boolean>

[Function Description]

Resumes the audio playback.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: result of resuming the audio playback. The options are **true** (success) and **false** (failure).

setPlayoutVolume

setPlayoutVolume(volume: number): boolean

[Function Description]

If you set the audio volume, sound will be heard.

[Request Parameters]

volume: (mandatory) audio volume. The value is a number ranging from 0 to 100.

[Response Parameters]

boolean: whether the audio volume has been set. The options are **true** (success) and **false** (failure).

getPlayoutVolume

getPlayoutVolume(): number

[Function Description]

Obtains the audio volume.

[Request Parameters]

None

[Response Parameters]

number: volume value. The value ranges from 0 to 100.

muteAudio

muteAudio(isMute: boolean): void

[Function Description]

Mutes.

[Request Parameters]

isMute: (mandatory) whether to mute. The value is of the Boolean type. **true** indicates muting, and **false** indicates unmuting.

[Response Parameters]

None

streamStatistic

streamStatistic(enable: boolean, interval: number): void

[Function Description]

Specifies whether to enable stream statistics.

[Request Parameters]

- **enable**: (mandatory) Specifies whether to enable stream statistics. The value is of the Boolean type. The value **true** indicates that stream statistics are enabled.

- **interval**: (mandatory) Specifies the statistics interval, in seconds. The value is a number ranging from 1 to 60. The default value is **1**.

[Response Parameters]

None

enableStreamStateDetection

enableStreamStateDetection(enable: boolean, interval: number, interruptRetry:StreamInterruptRetry): boolean

[Function Description]

Enables or disables media stream status detection. After the function is enabled, the system can detect whether the stream has been interrupted at the stream push device.

[Request Parameters]

- **enable**: (mandatory) whether to enable media stream status detection. The value is of the Boolean type. **true** indicates enabled; **false** (default value) indicates disabled.

- **interval**: (mandatory) Specifies the interval in seconds. The value is a number ranging from 1 to 60. This parameter is used to determine when there is no media stream. The default value is **3** (recommended).

- **interruptRetry**: (optional) Parameter for configuring playback retry upon stream interruption. The value is of the StreamInterruptRetry type. The definition of StreamInterruptRetry is as follows: {

  enable: The value is of the Boolean type, indicating that attempt for automatically resuming playback is enabled after stream interruption. The default value is **false**, indicating that attempt for automatically resuming playback is disabled.

  retryInterval: retry interval for stream pull, in seconds. The value type is number. The value ranges from 10 to 60 and defaults to **30**.

  retryTimes: maximum number of retry times for resuming playback. The value type is number. The minimum value is **1** and the default value is **30**.

  }

[Response Parameters]

boolean: whether the operation is successful. The options are **true** (success) and **false** (failure).

destroyClient

destroyClient(): void

[Function Description]

Destroys a client object.

[Request Parameters]

None

[Response Parameters]

None

fullScreenToggle

fullScreenToggle(isExit: boolean): void

[Function Description]

Enables/Disables full-screen display.

[Request Parameters]

isExit: (mandatory) Boolean type. The default value is **false**.

[Response Parameters]

None

3.9.2 Client Object (HWHlsClient)

This section describes the HWHlsClient APIs of the LLL web SDK.

Table 3-10 HRTC APIs

API	Description
startPlay	Starts playback. The client obtains the corresponding live stream from the server based on the entered URL.
switchPlay	Quickly switches to the next stream.
stopPlay	Stops the playback.
replay	Replays the video.
resume	Resumes the playback.
pause	Pauses the playback.
pauseVideo	Pauses the video playback.
resumeVideo	Resumes the video playback.
pauseAudio	Pauses the audio playback.
resumeAudio	Resumes the audio playback.
setPlayoutVolume	Sets the playback volume.
getPlayoutVolume	Obtains the audio volume.

API	Description
muteAudio	Mutes.
streamStatistic	Specifies whether to enable stream statistics.
enableStreamStateDetec-tion	Enables or disables media stream status detection.
destroyClient	Destroys a client object.
fullScreenToggle	Enables/Disables full-screen display.

startPlay

startPlay(url: string, options: StartPlayOptions): Promise<void>

[Function Description]

Starts playback. The client obtains the corresponding live stream from the server based on the entered URL.

[Request Parameters]

- **url**: (mandatory) String type. Ingest URL that ends with **m3u8**.

- **options**: (optional) StartPlayOptions type. If this parameter is not carried, the **options** data carried in the first playback start request is reused. The definition of StartPlayOptions is as follows: {

  - **elementId**: (mandatory) indicates the playback DOM ID.
  - **objectFit**: (optional) String type. Default value: **cover**. The following enumerated values are supported:

    - **contain**: prioritizes the display of all video content. 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**: prioritizes the filling of the window. 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.

⚠ **CAUTION**

The browser on a mobile phone, such as OPPO, may create a control to overwrite the SDK player, resulting in invalid configuration.

– **muted**: (optional) Boolean type. **true** indicates muted; **false** indicates unmuted. The default value is **false**.

– **sessionId**: This parameter does not need to be transferred.

– **showLoading**: (optional) Boolean type. Indicates whether to enable the loading display effect. **true** indicates that the loading display effect is enabled. The default value is **false**. When this parameter is set to **true**, the loading effect upon playback start will also be enabled. The loading effect upon buffering during playback needs to be set based on the **LOADING_CONFIG** in the **setParameter** API.

⚠️ **CAUTION**

- The QQ browser on Android devices does not support this function.
- You are advised to leave **showLoading** blank or set it to **false**.

– **autoPlay**: (optional) Boolean type. **true** indicates that the autoplay function is enabled. **false** indicates that the autoplay function needs to be manually triggered. The default value is **true**.

– **poster**: (optional) The object definition is as follows: {

  ▪ **url**: (optional) String type. Sets the complete address of the thumbnail image to be played. The image must be in JPG, PNG, or static GIF format, the size cannot exceed 1 MB, and the resolution cannot exceed 1920 x 1080. The file name cannot contain Chinese characters.

  ▪ **mode**: (optional) String type. The default value is **cover**. The following enumerated values are supported: {

    ○ **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.

    ○ **crop**: original size of the video thumbnail. If the thumbnail exceeds the playback area, the excess part is cropped. Otherwise, the thumbnail is displayed in the middle of the playback window.

  }

  ▪ **startEnable**: (optional) Boolean type. Indicates whether to display the thumbnail when the playback starts. The options are **true** and **false**. The default value is **false**. This parameter takes effect only in non-autoplay scenarios.

  ▪ **pauseEnable**: (optional) Boolean type. Indicates whether to display the thumbnail on the playback page when the video is paused. The options are **true** and **false**. The default value is **false**.

  }

– **webrtcConfig**: This parameter does not need to be transferred.

– **schedulePolicy**: This parameter does not need to be transferred.

– **domainPolicy**: This parameter does not need to be transferred.

– **downgradeUrl**: This parameter does not need to be transferred.

[Response Parameters]

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

switchPlay

switchPlay(url: string, options: StartPlayOptions): Promise<void>

[Function Description]

After the playback is started, the system quickly switches to the next stream.

[Request Parameters]

- **url**: (mandatory) String type. Ingest URL that ends with **m3u8**.

- **options**: (optional) StartPlayOptions type. If this parameter is not carried, the **options** data carried in the first playback start request is reused. The definition of StartPlayOptions is as follows: {

  – **elementId**: (mandatory) indicates the playback DOM ID.

  – **objectFit**: (optional) String type. Default value: **cover**. The following enumerated values are supported:

    ▪ **contain**: prioritizes the display of all video content. 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**: prioritizes the filling of the window. 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) Boolean type. **true** indicates muted; **false** indicates unmuted. The default value is **false**.

  – **sessionId**: This parameter does not need to be transferred.

  – **showLoading**: (optional) Boolean type. Indicates whether to enable the loading display effect. **true** indicates that the loading display effect is enabled. The default value is **false**. When this parameter is set to **true**, the loading effect upon playback start will also be enabled. The loading effect upon buffering during playback needs to be set based on the **LOADING_CONFIG** in the **setParameter** API.

  – **autoPlay**: (optional) Boolean type. **true** indicates that the autoplay function is enabled. **false** indicates that the autoplay function needs to be manually triggered. The default value is **true**.

  – **poster**: (optional) The object definition is as follows: {

- **url**: (optional) String type. Sets the complete address of the thumbnail image to be played. The image must be in JPG, PNG, or static GIF format, the size cannot exceed 1 MB, and the resolution cannot exceed 1920 x 1080. The file name cannot contain Chinese characters.

- **mode**: (optional) String type. The default value is **cover**. The following enumerated values are supported: {

  - **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.

  - **crop**: original size of the video thumbnail. If the thumbnail exceeds the playback area, the excess part is cropped. Otherwise, the thumbnail is displayed in the middle of the playback window.

  }

- **startEnable**: (optional) Boolean type. Indicates whether to display the thumbnail when the playback starts. The options are **true** and **false**. The default value is **false**. This parameter takes effect only in non-autoplay scenarios.

- **pauseEnable**: (optional) Boolean type. Indicates whether to display the thumbnail on the playback page when the video is paused. The options are **true** and **false**. The default value is **false**.

  }

  – **webrtcConfig**: This parameter does not need to be transferred.
  – **schedulePolicy**: This parameter does not need to be transferred.
  – **domainPolicy**: This parameter does not need to be transferred.
  – **downgradeUrl**: This parameter does not need to be transferred.

[Response Parameters]

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

stopPlay

stopPlay(): boolean

[Function Description]

Stops the playback.

[Request Parameters]

None

[Response Parameters]

boolean: result of stopping playback. The options are **true** (success) and **false** (failure).

replay

replay(): Promise<boolean>

[Function Description]

Replays the video.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: result of replay. The options are **true** (success) and **false** (failure).

resume

resume(): Promise<boolean>

[Function Description]

Resumes the playback.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: result of resuming the audio/video playback. The options are **true** (success) and **false** (failure).

pause

pause(): boolean

[Function Description]

Pauses the audio/video playback.

[Request Parameters]

None

[Response Parameters]

boolean: result of pausing playback. The options are **true** (success) and **false** (failure).

pauseVideo

pauseVideo(): boolean

[Function Description]

This API is not supported.

[Request Parameters]

None

[Response Parameters]

boolean: Only **false** is returned.

resumeVideo

resumeVideo(): Promise<boolean>

[Function Description]

This API is not supported.

[Request Parameters]

None

[Response Parameters]

boolean: Only **false** is returned.

pauseAudio

pauseAudio(): boolean

[Function Description]

Pauses the audio playback.

[Request Parameters]

None

[Response Parameters]

boolean: result of pausing audio playback. The options are **true** (success) and **false** (failure).

resumeAudio

resumeAudio(): Promise<boolean>

[Function Description]

Resumes the audio playback.

[Request Parameters]

None

[Response Parameters]

Promise<boolean>: result of resuming the audio playback. The options are **true** (success) and **false** (failure).

setPlayoutVolume

setPlayoutVolume(volume: number): boolean

[Function Description]

If you set the audio volume, sound will be heard.

[Request Parameters]

volume: (mandatory) audio volume. The value is a number ranging from 0 to 100.

[Response Parameters]

boolean: whether the audio volume has been set. The options are **true** (success) and **false** (failure).

getPlayoutVolume

getPlayoutVolume(): number

[Function Description]

Obtains the audio volume.

[Request Parameters]

None

[Response Parameters]

number: volume value. The value ranges from 0 to 100.

muteAudio

muteAudio(isMute: boolean): void

[Function Description]

Mutes.

[Request Parameters]

isMute: (mandatory) whether to mute. The value is of the Boolean type. **true** indicates muting, and **false** indicates unmuting.

[Response Parameters]

None

streamStatistic

streamStatistic(enable: boolean, interval: number): void

[Function Description]

Specifies whether to enable stream statistics.

[Request Parameters]

- **enable**: (mandatory) Specifies whether to enable stream statistics. The value is of the Boolean type. The value **true** indicates that stream statistics are enabled.
- **interval**: (mandatory) Specifies the statistics interval, in seconds. The value is a number ranging from 1 to 60. The default value is **1**.

[Response Parameters]

None

enableStreamStateDetection

enableStreamStateDetection(enable: boolean, interval: number, interruptRetry:StreamInterruptRetry): boolean

[Function Description]

Enables or disables media stream status detection. After the function is enabled, the system can detect whether the stream has been interrupted at the stream push device.

[Request Parameters]

- **enable**: (mandatory) whether to enable media stream status detection. The value is of the Boolean type. **true** indicates enabled; **false** (default value) indicates disabled.

- **interval**: (mandatory) Specifies the interval in seconds. The value is a number ranging from 1 to 60. This parameter is used to determine when there is no media stream. The default value is **3** (recommended).

- **interruptRetry**: (optional) Parameter for configuring playback retry upon stream interruption. The value is of the StreamInterruptRetry type. The definition of StreamInterruptRetry is as follows: {

    enable: The value is of the Boolean type, indicating that attempt for automatically resuming playback is enabled after stream interruption. The default value is **false**, indicating that attempt for automatically resuming playback is disabled.

    retryInterval: retry interval for stream pull, in seconds. The value type is number. The value ranges from 10 to 60 and defaults to **30**.

    retryTimes: maximum number of retry times for resuming playback. The value type is number. The minimum value is **1** and the default value is **30**.

    }

[Response Parameters]

boolean: whether the operation is successful. The options are **true** (success) and **false** (failure).

⚠️ **CAUTION**

The QQ browser on Android devices does not support this function.

---

destroyClient

destroyClient(): void

[Function Description]

Destroys a client object.

[Request Parameters]

None

[Response Parameters]

None

fullScreenToggle

fullScreenToggle(isExit: boolean): void

[Function Description]

This API is not supported.

[Request Parameters]

isExit: (mandatory) Boolean type. The default value is **false**.

[Response Parameters]

None

4 Change History

Table 4-1 Change history

Released On	Description
2025-02-17	The web SDK version has been updated to 2.10.9.
2024-12-02	The web SDK version has been updated to 2.10.3.
2024-09-30	This issue is the first official release.