API Reference Guide

AudioCodes WebRTC Solutions for Enterprises

WebRTC Web Browser Client

Version 1.11

audiocodes

Table of Contents

This page is intentionally left blank.

Notice

Information contained in this document is believed to be accurate and reliable at the time of printing. However, due to ongoing product improvements and revisions, AudioCodes cannot guarantee accuracy of printed material after the Date Published nor can it accept responsibility for errors or omissions. Updates to this document can be downloaded from https://www.audiocodes.com/library/technical-documents.

This document is subject to change without notice.

Date Published: May-31-2020

WEEE EU Directive

Pursuant to the WEEE EU Directive, electronic and electrical waste must not be disposed of with unsorted waste. Please contact your local recycling authority for disposal of this product.

Customer Support

Customer technical support and services are provided by AudioCodes or by an authorized AudioCodes Service Partner. For more information on how to buy technical support for AudioCodes products and for contact information, please visit our Web site https://www.audiocodes.com/services-support/maintenance-and-support.

Stay in the Loop with AudioCodes

Abbreviations and Terminology

Each abbreviation, unless widely used, is spelled out in full when first used.

Related Documentation

Document Name
https://www.audiocodes.com/solutions-products/solutions/enterprise-voice/webrtc-connectivity

Document Revision Record

LTRT	Description
14040	Initial document release for Version 1.0
14041	Updated to Version 1.1
14042	Added deinit and setDtmfOptions
14043	Update for Version 1.3: Added setOAuthToken and setEnableAddVideo
14044	Updates for Version 1.4 (added the following new functions): • getBrowserName() • getServerAddress() • checkAvailableDevices() • getStreamInfo() • startSendingVideo() • hasVideo() • hasSendVideo() • hasReceiveVideo() • getVideoState() • setRemoteHoldState()
14045	Updates for Version 1.5: ▪ Added optional callbacks: ✓ callIncomingReinvite() ✓ transferorNotification() ✓ transfereeRefer() ✓ transfereeCreatedCall() ▪ Added function: sendRefer()
14046	Updates for Versions 1.6 and 1.7: ▪ Added support for Safari 12.0.0 and higher ▪ Added optional callback: incomingNotify()
14047	Updates for Version 1.8: ▪ Added support for incoming call with Replaces header (see incomingCall callback) ▪ Added support for sending/receiving a SIP MESSAGE ▪ Added support for attended call transfer.

LTRT	Description
14048	Updates for Version 1.9: ▪ Updated to use Chrome Unified SDP plan ▪ Removed SDP editing and instead used transceiver API ▪ Obsolete RTCPeerConnection addstream event replaced to track events ▪ Removed usage of obsolete RTCPeerConnection.getLocalStreams() and getRemoteStreams() As result of the modification method, phone.getWR().connection.getStreamInfo() was replaced to phone.getWR().stream.getInfo() Added new methods to use and instead removed call.getRTCLocalStream() and call.getRTCRemoteStream() ▪ Modified implementation of call video flags (hasVideo(), hasReceiveVideo(), ✓ hasSendVideo(), getVideoState(). ✓ Value call.data['_video' is no longer used. ✓ For incoming call the flags set according initial INVITE SDP. ▪ The flags updates according answer SDP (hold SDP is ignored) ▪ Added call.hasEnabledReceiveVideo() flag. Initially set according phone.enableAddVideo flag. Set also for outgoing video call, or when used answer with phone.VIDEO or phone.RECVONLY_VIDEO, or startSendingVideo(enableReceiveVideo=true) ▪ Added call.hasEnabledSendVideo() flag. Set for outgoing video call, answer with phone.VIDEO or then called startSendingVideo() method. ▪ Added support of arbitrary audio and video constraints using methods: ✓ phone.setConstraints() and phone.setBrowsersConstraints(). ✓ Marked as obsolete phone.setChromeAudioConstraints(). ✓ Please use instead phone.setConstraints('chrome', 'audio', {…}) ▪ Modified withVideo argument of call and answer. ✓ Instead of Boolean true/false, use phone.AUDIO, phone.VIDEO and phone.RECVONLY_VIDEO. ▪ Modified internal method: detectBrowser() - now phone.browser variable can be: ✓ One of 'chrome' (for Chrome and Chrome-based browsers, includes new Microsoft Edge), 'firefox', 'safari', and 'other' (cannot detect browser). ✓ Improved detection of Chromium-based browsers for logging ▪ Removed special support of Legacy Microsoft Edge. Now it is detected as 'other' browser in AudioCodes User Agent, AudioPlayer and AudioRecorder. ▪ Removed method call.isHoldInProgress(). ▪ Added method call.stopSendingVideo()
14049	Updates for Version 1.10: ▪ Removed the Authentication Bearer header from INVITE OK ▪ Added new optional argument to phone.setOAuthToken(token, useInInvite=true) ▪ Added phone.setModes method to configure SDK modes ▪ Added workaround to missed currently in Chrome RTP timeout detection (can be configured via phone.setModes method) ▪ Added a new argument 'hasSDP' for incomingCall callback ('true' when incoming SIP INVITE has SDP body) ▪ Added an optional argument pongDistribution, to setWebSocketKeepAlive. This argument is printed in Keep Alive statistics. Not only the minimum and maximum values of pong delay, but also the delay distribution. ▪ Added sound configuration to utils.js audioPlayer by editing the configuration file (See https://webrtcdemo.audiocodes.com/sdk/)

LTRT	Description
14050	Updates for Version 1.11: ▪ chrome_rtp_timeout_fix ▪ Added setModes setting: ice_timeout_fix ▪ Added setModes setting: sbc_ha_pairs_mode ▪ Added method sendInfo to class AudioCodesSession ▪ Added callback incomingInfo

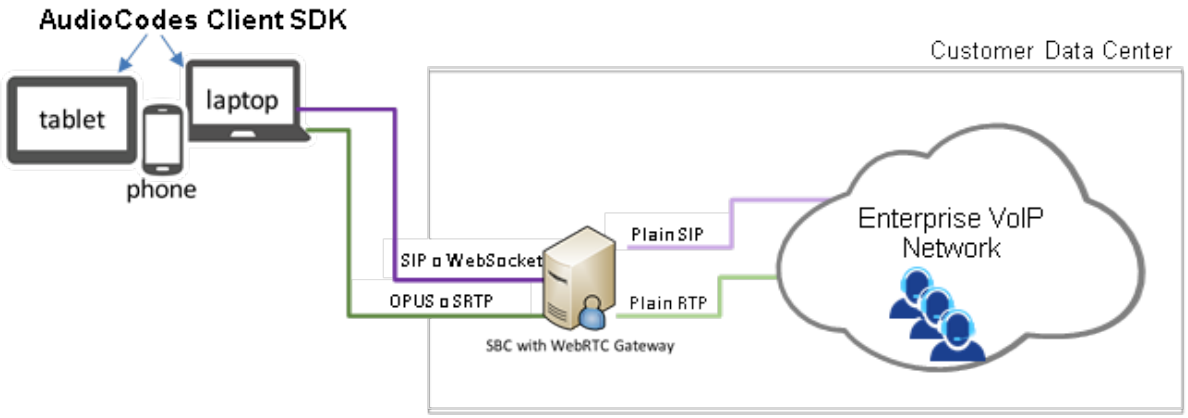
Documentation Feedback

AudioCodes continually strives to produce high quality documentation. If you have any comments (suggestions or errors) regarding this document, please fill out the Documentation Feedback form on our Web site at https://online.audiocodes.com/documentation-feedback.

1 Introduction

WebRTC technology enriches user experience by adding voice, video and data communication to the Web browser, as well as to mobile applications. AudioCodes WebRTC gateway provides seamless connectivity between WebRTC clients and existing VoIP deployments.

A typical WebRTC solution comprises a WebRTC Gateway, which is an integrated functionality on AudioCodes SBCs, and a client application running on a browser or a mobile application. AudioCodes WebRTC client SDK is a JavaScript code that allows web developers to integrate WebRTC functionality into the browser for placing calls from the browser to the SBC.



Note: For a simple click-to-call button use case, a WebRTC widget is offered which can be easily integrated into websites and blogs without any JavaScript knowhow. See the *WebRTC Widget Installation and Configuration Guide*.

1.1 Purpose

This *Reference Guide* defines Application Programming Interface (API) use of the Web Real-Time Communications (RTC) SDK.

1.2 Scope

The guide describes the API that must be implemented to use AudioCodes' Web RTC SDK to build a Web phone client that will interact with AudioCodes' server to establish voice and video calls. The guide may be used by web developers and application developers who want to use the AudioCodes-provided SDK to build Web RTC clients.

1.3 Benefits

The following summarizes the benefits you'll gain from the API:

- Simple deployment - a single WebRTC gateway device for both signaling and media
- Strong security and interoperability capabilities resulting from integration with the SBC
- Client SDK for browsers
- OPUS powered IP phones for superb, transcoder-less voice quality

This page is intentionally left blank.

2 API Classes

Two usable objects are available:

- AudioCodesUA – Audio Codes User Agent (single tone) – see below
- AudioCodesSession – for call representation – see below

2.1 AudioCodesUA

AudioCodesUA is used to initialize the framework before starting to make and receive calls. This class is mostly used to initialize the Web RTC engine and to register to the service.

```
Class AudioCodesUA{
    constructor()
    void setAccount (String userName, String displayName, String
    password, String authName=userName);
    void setServerConfig(List<InetSocketAddress> serverAddresses,
    String serverDomain, List<IceConfig> iceServers=[]);
    void setListeners (listeners);
    void init(Boolean autologin=true);
    void login();
    void logout();
    AudioCodesSession call(symbol videoOption, String  call_to,
    replacesHeader=null);
    void setRegisterExtraHeaders (List<SipHeader> extraHeaders);
    extraHeaders);
    void setUseSessionTimer(Boolean use);
    void setRegisterExpires(int expires);
    Boolean isInitialized();
    String version();
    void setUserAgent(String name);
    void setConstraints(String browser, String type, Object
    constrains);
    setBrowsersConstraints(Object object);
    void setAcLogger(Function logger);
    void setJsSipLogger(Function logger);
    void setWebSocketKeepAlive(int ping, int pong=0, int stats=0,
    dist=false);
    void setReconnectIntervals(int min, int max);
    void deinit();
    void setDtmfOptions(Boolean useWebRTC, int duration=null, int
    interToneGap=null)
    void setOAuthToken(String token, Boolean useInInvite=true)
    void setEnableAddVideo(Boolean enable)
    String getBrowserName()
    String getServerAddress()
    Promise checkAvailableDevices()
    Promise getWR().stream.getInfo(MediaStream stream)
    Promise
    getWR().connection.getTransceiversInfo(RTCPeerConnection conn)
    Promise getWR().connection.getStats(RTCPeerConnection conn,
    String array reportNames)
    void setModes(Object modes)

}
```

2.1.1 Standard Methods

2.1.1.1 constructor

Creates the object instance.

2.1.1.2 init

Initializes the user agent and establishes a connection with the AudioCodes Mediant server.

Parameter

- `Autologin:true` After connection, automatically call login()] (Optional). By default, 'true'.

Return Values

N/A

2.1.1.3 setServerConfig

Configures the AudioCodes Mediant server.

Parameters

- ServerAddresses :
  - serverAddresses: inetSocket Address List of the AudioCodes Mediant servers
  - serverAddresses [integer]: Two elements array list; where each array in the list contains the inetSocket Address of the AudioCodes Mediant servers and priority
- serverDomain [string]: String of the domain name to which to register
- iceServers: List of the STUN and TURN servers

  This optional parameter is by default set as an empty list [].

  If it is used as an empty list during the call opening, an external STUN server is not used. This mode is useful when the phone is used towards an SBC server.

Note: When the iceServers list is empty, after the call opens, the client still periodically sends STUN requests to the port used for the RTP stream for checking that the RTC channel is alive.

Return Values

N/A

2.1.1.4 setAccount

Defines the account details.

Parameters

- userName [string]: Authenticating user name
- displayName [string]: Displayed string name shown in the client interface
- password [string]: Authenticating user password
- authName [string]: Authorization user name (optional)

Return Values

N/A

2.1.1.5 login

Performs registration to the service.

Parameters

N/A

Return Values

N/A

2.1.1.6 logout

Performs de-registration from the service.

Parameters

N/A

Return Values

N/A

2.1.1.7 setListeners

Defines the listeners object.

Parameter

- Listener [Object that holds the methods to be triggered; See Section 4.2, User Agent: Set Listeners (Callbacks) for an example]

Return Values

N/A

2.1.1.8 call

Initiates an outgoing call.

Parameters

- videoOption [symbol] phone.VIDEO - If the call must be initiated with video, or phone.AUDIO
- call_to [string]: Defines a string of the destination address/number

Return Values

A call session object is defined here.

Note: This call is also provided with another parameter (see Section 2.1.2).

2.1.2 Advanced Methods

The advanced methods are optional. They provide the API, which is based on SIP (Session Initiation Protocol), with an extra level of flexibility. Developers familiar with SIP can utilize the advanced methods.

2.1.2.1 setRegisterExtraHeaders

Allows adding additional headers to the registration request.

Note: The headers must be SIP headers that conform to RFC 3261.

Parameter

- ExtraHeaders [List of headers]

Return Values

N/A

2.1.2.2 call

Initiates an outgoing call.

Parameters

- videoOption [symbol] phone.VIDEO if the call must be initiated with Video, or phone.AUDIO for audio call.
- call_to [string]: Destination address/number
- extraHeaders (Optional). An array of strings, each representing a SIP header, to be added to the INVITE request.

Note: See also Use Examples under Section 4 for adding a SIP 'Replaces' header to restore a call after a page reload.

Return Values

A call session object defined as shown under Section 2.2.

2.1.2.3 setUseSessionTimer

Allows enabling SIP session timers in the call session. If not used, the default value 'false' is used.

Parameter

- enable [Boolean]: 'true' if yes; 'false' if no

Return Values

N/A

2.1.2.4 setRegisterExpires

Changes the default registration interval from the default value (600).

Parameter

- expires [integer] in seconds.

Return Values

N/A

2.1.2.5 isInitialized

Checks if the init() method was called.

Parameters

N/A

Return Values

Boolean

2.1.2.6 version

Retrieves a string with the API version.

Parameters

N/A

Return Values

String containing the API version.

2.1.2.7 setChromeAudioConstraints

This method is now obsolete. Use setConstraints or setBrowserConstraints instead.

Note: The method with the “echoCancellation” argument, performs exactly the same as setConstraints(“chrome”, “audio”, {echoCancellation: true}).

It changes the echo cancellation and noise suppression flags, specifically for the Chrome browser.

Parameters

- String with parameters, delimited by comma. The following can be used:

```
"autoGainControl,echoCancellation"
or "autoGainControl"
or "echoCancellation"
or ""
```

Return Values

N/A

2.1.2.8 setConstraints

Note: Google releases new builds of the Chrome browser generally every six weeks. Each new release may change or ignore previously supported audio or video constraints. The developer can check which parameters are used by opening the phone prototype in one tab, and **chrome://webrtc-internals** in the other tab. Calls can then be made to the other phone and check which audio constraints are used in GetUserMediaRequest in the webrtc-internals page.

Parameters

- browser name [string] “chrome” or “firefox” or “safari” or “other”
- type [string] “audio” or “video”
- constraints [object] constraints in format described:
  https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API/Constraints

This can be used for complex formatting including the following keywords:

- min
- max
- ideal
- exact
- advanced

A media constraint can be specified as mandatory (using the keyword: “exact”), and as an optional (keyword: “ideal”) or without any keyword.

It is better to avoid mandatory constraints, since it cannot be confirmed whether the constraint is supported in the user's browser. It depends on what operating system, driver version and video and audio hardware is being used.

When using a mandatory format and the specified constraint is not supported, the call fails with a "User Denied Media Access" error.
In the console log, it appears as "getusermediafailed" [error:OverconstrainedError ...]

To prevent this error from occurring, it is recommended to use the optional constraint format.

This example causes the call to fail, if the Web camera you are using does not support the “facingMode” constraint:

```
setConstraints(“chrome”, “video”, {facingMode: {exact: “user” }});
```

To prevent such errors, optional constraints format should be used:

```
setConstraints(“chrome”, “video”, { facingMode: { ideal: “user”
}});
```

Before using this format, check if the constraint is supported by your computer hardware, and use ‘ideal’,’min’, ‘max’ constraints keywords instead of using ‘exact’.

```
let supported = navigator.mediaDevices.getSupportedConstraints();

// set audio constraints
let ac = {};

ac_log('Volume is supported: ', supported.volume ? true : false);
if (supported.volume){
    ac.volume = 0.7;
}

ac_log('Echo cancellation is supported: ',
supported.echoCancellation ? true : false);
if (supported.echoCancellation){
    ac.echoCancellation = true;
}

if (Object.keys(ac).length > 0){ // Is not empty ?
    phone.setConstraints(phone.getBrowser(), 'audio', ac);
}

// set video constraints
let vc = {};

ac_log('webcam facing mode is supported: ', supported.facingMode ?
true : false);
if (supported.facingMode){
    vc.facingMode = { ideal: 'user' };
}

ac_log('webcam aspect ratio is supported: ', supported.aspectRatio
? true : false);
if(supported.aspectRatio){
    vc.aspectRatio = 1.0;
```

```
}

if (Object.keys(vc).length > 0){ // Is not empty ?
    phone.setConstraints(phone.getBrowser(), 'video', vc);
}
```

To prevent the “overconstrained error” from occurring in the WebRTC getUserMedia method, check that the used constraint is supported and/or use the optional constraint format. Refer to the WebRTC specification constraints description keywords: “ideal”, and “advanced”.

Return Values

N/A

2.1.2.9 setBrowsersConstraints

Used to set constraints for all browsers.

Internally used setConstraints function with its limitations.

Note: In most situations, the exact camera or sound card each customer uses is unknown. To prevent the occurrence of the “overconstrained error”, use the optional constraint format.

For more information on WebRTC specification constraints descriptions for the “ideal”, and “advanced” keywords, refer to https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API/Constraints.

Parameters

Constraints for set of browsers [object].

For example:

```
{
    chrome: {
        audio: { echoCancellation: true, noiseSuppression: true,
audioGainControl: true },
        video: { aspectRatio: 1.0 }
    },
    firefox: {
        audio: { echoCancellation: true }
    }
}
```

The following browser names can be used: ‘chrome’, ‘firefox’, ‘safari’, ‘other’.

Return Values

N/A

2.1.2.10 setUserAgent

Configures the User Agent string used to build the SIP header User-Agent.

Parameter

- User agent [string]

Return Values

N/A

2.1.2.11 setAcLogger

Configures the logger function that is used by AudioCodes' API for logging. By default, the console log is used.

Parameter

- User-defined Logger function name.

Return Values

N/A

2.1.2.12 setJsSipLogger

Configures the logger function that is used by the JsSIP API for logging. By default, console.log is used.

Note: This function does not support all JsSIP logging functionality. For SIP errors, JsSIP uses other loggers that are not exposed and cannot be reassigned (see the 'debugerror' loggers in the JsSIP source code).

Parameter

- User-defined logger function name.

Return Values

N/A

2.1.2.13 setWebSocketKeepAlive

Used to send a CRLF X 2 Keep-alive message via WebSocket to the SBC. Based on RFC 7118, #6.

Purpose

Enables fast detection of a server connection failure, and reconnection.

Use

Used before SBC connection, before calling the "init" function.

Description

Sends a ping to the server and expects a ping response in a predefined timeout. If the first ping is not answered, the functionality is disabled for the registration session.

Parameters

- Ping interval [seconds]
- Ping timeout interval [seconds] (Optional): Timeout awaiting the ping response before declaring the connection as failed.
  - 0 (default): No timeout (managed by the operating system's TCP/IP stack)
- Printing interval [number] (Optional): The minimum and maximum values of pong delay intervals (in milliseconds) every set number of times, are printed to the logs.
  - 0 (default): No printing
- The pong delay distribution with time step 0.25 seconds, was added to the logs. [Boolean]
  - false (default): No print pong interval distribution

Return Values

N/A

2.1.2.14 setReconnectIntervals

After a connection failure, the JsSIP stack automatically reestablishes a connection starting from the minimum reconnection interval. If the reconnection is unsuccessful, the stack increases the interval before the next reconnection, up to the maximum value. By default, 2 and 30 seconds are used for the minimum and maximum values.

Parameters

- Minimum reconnection interval [integer]
- Maximum reconnection interval [integer]

Return Values

N/A

2.1.2.15 deinit

Disconnects a WebSocket connection to the SBC server after gracefully unregistering and terminating active sessions, if any.

isInitialized() returns 'false' after the method is used.

Parameters

N/A

Return Values

N/A

2.1.2.16 setDtmfOptions

Changes the DTMF options. If the method isn't called, DTMF is by default sent using WebRTC API with default settings.

Parameters

- useWebRTC [Boolean]: Used to send DTMF WebRTC API [true], or use SIP INFO [false]
- duration ms [integer] Optional. If the parameter is not set or is configured to 'null', 100 milliseconds is used by default.
- interToneGap ms [integer]: Optional. If the parameter is not set or is configured to 'null', 70 milliseconds is used by default for WebRTC and an interval of 500 milliseconds is used for separating SIP INFO messages.

Return Values

N/A

2.1.2.17 setOAuthToken

Sets the access token to OAuth2 authorization. This token is used while communicating with AudioCodes SBC to authorize the user access. The Oauth token usage makes the password usage redundant.

Parameters

- Token [string or null] .'null' can be used to clear the authorization token.
- useInInvite [Boolean] : Optional, by default 'true':
  - If 'false', the "Authorization: Bearer" header token is added only to the SIP REGISTER request. This is used if the SBC supports early versions of the Authorization bearer specification.
  - If 'true', the "Authorization: Bearer" header token is added to the SIP REGISTER and INVITE requests. This is used if the SBC supports the latest version of the Authorization bearer specification.

  This is set by default.

Return Values

N/A

2.1.2.18 setEnableAddVideo

If the call was opened as an audio call, and the other side sent a re-INVITE with the video, this enables the use of one-way incoming video. Two-way video cannot be added because video devices are requested with the getUserMedia command at call opening.

It may not be desirable to suddenly add one-way incoming video in the middle of a call. By default, this feature is disabled.

Parameters

- enabled [Boolean]

Return Values

N/A

2.1.2.19 getBrowserName

Returns browser name and version. This function can be used for logging purposes.

Parameters

N/A

Return Values

String including browser name and version.

2.1.2.20 getServerAddress

Returns the URL of the currently connected SBC server. This function can be used to restore the connection after reloading of Web page.

Parameters

N/A

Return Values

null (no connected server), or URL string of currently connected server.

2.1.2.21 checkAvailableDevices

This method has two functions:

- Checks if the WebRTC API is supported in the used browser. If not, the Promise object will be rejected with the following string: “WebRTC is not supported in the browser”.
- Checks available devices (speaker, microphone and camera). If the speaker is not connected, the speaker Promise object is rejected with the following string:

```
“Missing a speaker! Please connect one and reload”
```

If the microphone is not connected, the microphone is rejected with the following string:

```
"Missing a microphone! Please connect one and reload"
```

Parameters

N/A

Return Values

The Promise object is resolved with hasWebCamera Boolean value and is rejected with a string describing the problem (see above).

2.1.2.22 getWR().stream.getInfo

Gets stream information for debugging/logging purposes.

Parameters

- stream [MediaStream] – local or remote call media stream
  (see methods getRTCLocalStream and call.getRTCRemoteStream())

Return Values

The Promise object is resolved with a string value.

2.1.2.23 getWR().connection.getTransceiversInfo

Gets transceivers information for debugging/logging purposes.

Parameters

- connection [RTCPeerConnection] of current call.
  (see method getRTCPeerConnection)

Return Values

The Promise object is resolved with a string value.

2.1.2.24 getWR().connection.getStats

Gets connection statistics information for debugging / logging purposes.

Parameters

- connection [RTCPeerConnection] of current call. (see method getRTCPeerConnection)
- report names. [strings array] Report names, for example: ['outbound-rtp', 'inbound-rtp']

Return Values

The Promise object is resolved with a string value.

2.1.2.25 sendMessage

Sends text messages using SIP MESSAGE.

Notes:

- Currently the Mediant SBC does not support off-line messaging.
- To message recipient, the phone should be on-line (registered to SBC).
- If the recipient received a message, then the SIP response code is 2xx.
- If the recipient is off-line, the response code is 404 “User Not Found”
- The programmer should check the SBC response code using the returned Promise object. This will be resolved for 2xx SIP response codes, and failed for others.

Parameters

- to [string]: Recipient URL (format: user or user@host)
- body [string]: Text message
- contentType [string]: Optional. “text/plain” is used by default

Return Values

The Promise object is resolved if the SBC response includes response code 2xx, and fails for other response codes.

2.1.2.26 setModes

Configures the SDK internal modes or SDK patches parameters. Some patches used in SDK can cause problems and create new unforeseen problems for customers. Other problems can already be fixed in browsers, while some patches require numerical settings.

It is better to make patches configurable and configure them or turn them off, if they are not needed by specific customers.

All SDK patches have corresponding flags (some with numbers) in SDK. Each flag has a default value in the SDK.

The value can be changed via method phone.setModes().

To set values, it is recommended that you use the config.js configuration file, which can be changed without rebuilding the SDK or the phone.

Usage example:

Phone configuration file config.js:

```
let DefaultPhoneConfig = {
    . . .
    modes: {
        chrome_rtp_timeout_fix: 13
    },
    . . .
    version: '2-May-2020'
}
```

Set SDK modes or patches before phone.init()

```
// change default SDK configuration to custom.
```

```
phone.setModes(phoneConfig.modes);

phone.init(true);
```

Without rebuilding the phone, switching on/off and configuring SDK modes can be performed. Supported properties of the modes object:

- **chrome_rtp_timeout_fix** (Default: 13)

The existing workaround can be viewed in the current Chrome version bug **Chromium Issue 982793: iceconnectionstate does not go to failed if connection drops**. For more information, open the following link in the Chrome browser:

https://bugs.chromium.org/p/chromium/issues/detail?id=982793

By default, an RTP timeout of 20 seconds is used. It takes 7 seconds for Chrome to detect an RTP disconnection and then another 13 seconds to check that the disconnection state has not changed. If the internet connection is restored, it is changed to 'connected' state.

```
chrome_rtp_timeout_fix: 13
```

By changing the configuration value, the customer can increase or decrease the timeout, or completely disable the Chrome bug workaround by setting:

```
chrome_rtp_timeout_fix: undefined
```

In future releases, adding other SDK modes/patches is planned.

- **ice_timeout_fix** Default value: 0

  After setting a local description, the WebRTC starts ICE gathering, without a timeout in some cases. For Chrome, it can take 41 seconds.

  When AudioCodes' SBC is used, the phone functions without additional IPs provided by ICE gathering. ICE gathering cannot be disabled, but a local SDP can be sent without waiting for ICE gathering, so the default **0** means not to wait for ICE gathering at all. The value is set in milliseconds. In SDK versions preceding 1.11, a hardcoded value of **2000** was used.

  If set to an **undefined** value, the ICE gathering timeout will not be checked and sometimes (depending on the configuration of the user's computer's internet adapters) will add for a Chrome delay of 41 seconds for an outgoing call between the command 'make call' and sending a SIP INVITE, and for an incoming call between the command 'answer' and the call opening.

- **sbc_ha_pairs_mode** Default value: undefined

  If a customer uses multiple HA pairs, the value should be set to some integer value. In testing, 15 (seconds) was used. After an SBC disconnection, JsSIP is reconnected to a different URL (when multiple SBC URLs are used).

  When High Availability SBC pairs are used, best to attempt to reconnect to the same URL as was connected. If there is an undefined value, set a numeric (integer) value instead. After a disconnection, JsSIP attempts to reconnect to the same URL during the time (e.g., 15 seconds).

  If the connection cannot be restored, then the default JsSIP algorithm (reconnect to other URLs) will be used. The value should be undefined if a single URL is used, or if multiple URLs of the SBC in simple (not HA pairs) configuration is used.

Parameter

- modes object

Return Values

N/A

2.2 AudioCodesSession

Represents a call session that is used in the following scenarios:

- When initiating a call via the AudioCodesUA
- When receiving a callback of an incoming call

Syntax

```
class AudioCodesSession {
void answer (symbol videoOption)
void reject ()
void redirect(String callTo)
void terminate ()
void muteAudio(Boolean mute)
void muteVideo(Boolean mute)
Boolean isAudioMuted()
Boolean isVideoMuted()
void sendDTMF(char dtmf)
Boolean isOutgoing()
Map<String, Object>data;
int duration()
Boolean isLocalHold()
Boolean isRemoteHold()
Boolean isReadyToReOffer()
Promise hold(Boolean holdCall)
string getReplacesHeader()
Promise startSendingVideo(enabledReceiveVideo=true)
Promise stopSendingVideo()
Boolean hasVideo()
Boolean hasSendVideo()
Boolean hasReceiveVideo()
string getVideoState()
Boolean hasEnabledSendVideo()
Boolean hasEnabledReceiveVideo()
string getEnabledVideoState()
void setRemoteHoldState()
RTCPeerConnection getRTCPeerConnection()
MediaStream getRTCLocalStream()
MediaStream getRTCRemoteStream()
Promise sendReInvite(callShowStreams=true)
Boolean wasAccepted()
void sendInfo(body, contentType, extraHeaders=null)
}
```

2.2.1 Standard Methods

2.2.1.1 answer

Initiates the object and establishes the call.

Parameter

videoOption [symbol] one of phone.VIDEO, phone.RECVONLY_VIDEO or phone.AUDIO

Return Values

N/A

Note: This call is also provided with another parameter (see Section 2.2.2.1)

2.2.1.2 reject

Rejects a call.

Parameters

N/A

Return Values

N/A

Note: This call is also provided with another parameter (see Section 2.2.2.2)

2.2.1.3 redirect

Redirects the call and asks the caller to call the destination.

Parameter

- Call_to [string of destination address/number]: Used for SIP response code 302 with Contact header.

Return Values

N/A

Note: This call is also provided with another parameter (see Section 2.2.2.3)

2.2.1.4 terminate

Terminates an active call.

Parameters

N/A

Return Values

N/A

2.2.1.5 muteAudio

Defines the status of the audio mute (on/off).

Parameter

- Mute [Boolean]: 'true' to mute audio; 'false' to unmute audio

Return Values

N/A

2.2.1.6 muteVideo

Defines the status of the video mute (on/off).

Parameter

- mute [Boolean]: 'true' to mute video; 'false' to unmute video

Return Values

N/A

2.2.1.7 isAudioMute

Checks the audio mute status.

Parameters

N/A

Return Values

[Boolean]: 'true' if audio is muted, 'false' if audio is unmuted.

2.2.1.8 isVideoMute

Checks video mute status.

Parameters

N/A

Return Values

[Boolean]: 'true' if video is muted, 'false' if video is unmuted

2.2.1.9 sendDTMF

Sends a DTMF character.

Parameter

- dtmf [One DTMF character]

Return Values

N/A

2.2.1.10 isOutgoing

Checks if a call is outgoing.

Parameters

N/A

Return Values

[Boolean]: 'true' if a call is outgoing, 'false' if a call is incoming

2.2.1.11 data: map<String, Object>

Data are object variables, represented by a key / value list into which API developers can enter a string key and an object value for later use in the program flow. Example: String key 'label' and an object reference to the GUI object. This provides API developers with flexibility to use string keys and values for implementation. This data assists API developers to distinguish between implementation scenarios.

Note: Variables with a name starting with underscore '_' are reserved for internal API usage. Developers should not use these variables.

Predefined Parameters

`data['_user']`	Remote user name set according to the SIP header 'To' (outgoing call), or 'From' (incoming call).
`data['_host']`	Remote host set according to the SIP header 'To' (outgoing call), or 'From' (incoming call).
`data['_display_name']`	Remote user optional display name; set according to SIP header 'To/From' (the same as _user).
`data['_create_time']`	Timestamp(javascript Date()) of the call created. For call logging, the time and call duration (duration between call confirmation and call termination) are used.

2.2.1.12 duration

Defines the call duration (in seconds).

If this method is used before the call has been accepted (i.e., sends or receives SIP OK response), it returns "0".

If this method is used for an open call, it returns intervals after the call has been accepted until the current time.

After call termination, it returns call duration from the time the call has been accepted until the time the call terminated.

Parameters

N/A

Return Values

[int]: call duration in seconds

2.2.1.13 wasAccepted

Defines whether the call has been accepted (sends or receives a **SIP OK** response).

This method can be used after call termination, to check if the call was established or failed.

Parameters

N/A

Return Values

[Boolean]: 'true' when the call was accepted

2.2.1.14 isLocalHold

Defines whether the client initiates the hold state. This indicates that the client can release the hold.

Parameters

N/A

Return Values

[Boolean]: 'true' if the call is in a local hold, 'false' if it isn't in a local hold.

2.2.1.15 isRemoteHold

Defines whether the remote side initiated the hold. This indicates that the client cannot release the hold.

Parameters

N/A

Return Values

[Boolean]: 'true' if the call is in a remote hold. 'false' if it isn't in a remote hold.

2.2.1.16 IsReadyToReOffer

Implements a hold using SIP re-INVITE. It then checks, before using the hold, if the call is ready to re-offer (i.e., ready to send the re-INVITE).

Parameters

N/A

Return Values

[Boolean] 'true' if call is ready to re-offer, 'false' if it isn't ready to re-offer.

2.2.1.17 hold

Sets the call to on-hold (or to un-hold).

Parameter

- Hold [Boolean]: Sets the call to hold.

Return Values

Promise to wait until the end of the operation.

2.2.2 Advanced Methods

2.2.2.1 answer

Initiates the object and establishes the call.

Parameters

- videoOption [symbol]:
  - phone.VIDEO - This is used if the call uses two-way video
  - phone.RECVONLY_VIDEO - Does not send video, but receives video
  - phone.AUDIO - This is used for audio calls.
- extraHeaders (Optional): An array of strings, each representing a SIP header to add to the request.

Return Values

N/A

2.2.2.2 reject

Rejects a call.

Parameters

- Status code (Optional): Integer representing the reject reason (4xx or 6xx codes, 486 busy (default))
- extraHeaders (Optional): An array of strings, each representing a SIP header to be added to the request.

Return Values

N/A

2.2.2.3 redirect

Redirects the call; asks the caller to call the destination.

Parameters

- Call_to [string of the destination address/number]: Used for SIP response code 302 with Contact header.
- Status code [integer] (Optional): Integer representing the reject reason (3xx codes, 302 moved temporarily (default))
- extraHeaders [string array] (Optional): An array of strings, each representing a SIP header to add to the request.

Return Values

N/A

2.2.2.4 getReplacesHeader

Retrieves the SIP 'Replaces' header that can be used to restore the last call after page reloading. See the example in Section 4.11.

Parameters

N/A

Return Values

The string replaces the header value according RFC 3891, or 'null' if the call is not established.

The string may be used to re-establish a failed call session on the client side (if a session still exists on the SBC, for example, in the case of a page refresh).

See the Use Examples under Section 4 for more details.

2.2.2.5 getRTCPeerConnection

Retrieves the session internal RTCPeerConnection. For example, the object can be used to collect call statistics.

Note: If a call is terminated, peerConnection is closed and statistics are not available (closed by JsSip).

For more information on Peer Connection and GetStats, refer to
https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection and
https://www.callstats.io/2015/07/06/basics-webrtc-getstats-api/

See also the Use Examples under Section 4 for more details.

Parameters

N/A

Return Values

RTCPeerConnection object.

2.2.2.6 getRTCLocalStream

Retrieves the session internal media local stream.

For example, the object can be used to check audio and video tracks.

Parameters

N/A

Return Values

Local media stream

2.2.2.7 getRTCRemoteStream

Retrieves the session internal media remote stream.
For example, the object can be used to check audio and video tracks.

Note: Can be null, during call opening.

Parameters

N/A

Return Values

Remote media stream.

2.2.2.8 startSendingVideo

For audio calls, this methodstarts sending video stream to the other side.
Set internal call flag setEnabledSendVideo()

Parameters

- enableReceiveVideo [Boolean] by default is 'true'.
  The argument sets the internal call flag hasEnabledReceiveVideo().

Return Values

Promise object that is resolved if successfully started, and rejected otherwise.

2.2.2.9 stopSendingVideo

Stops sending video streams to the other side, for video calls.

Clears the setEnabledSendVideo() internal call flag.

Parameters

N/A

Return Values

The Promise object that is resolved if it successfully stops sending video streams. Otherwise, it is rejected.

2.2.2.10 hasVideo, hasSendVideo, hasReceiveVideo

Checks call video status:

- hasVideo: Returns true if two-way video is connected for the session.
- hasSendVideo: Returns true if the outgoing video stream is connected for the session.
- hasReceiveVideo: Returns true if the incoming video stream is connected for the session.

When a call is placed on hold, this method reflects the last status before the hold action. For example, two-way video is enabled and then a call is placed on hold (no video and no audio is active at this time); then the “hasVideo” value is returned.

This status is used for restoring call functionality and shows/hides GUI call video controls.

Parameters

N/A

Return Values

Boolean

2.2.2.11 getVideoStatus

Parameters

N/A

Return Value

[string] Representation of the call video status.

(one of ‘sendrecv’, ‘sendonly’, ‘recvonly’ or ‘inactive’)

Note: If a call was placed on hold, the video status will be the same as it was before it was put on hold.

2.2.2.12 hasEnabledSendVideo, hasEnabledReceiveVideo

Checks if calls were enabled to send or receive video:

- hasEnabledSendVideo: Returns 'true' when a call was enabled to send video.

  This is set when (phone.VIDEO), answer(phone.VIDEO) or startSendingVideo() are used.

  This is 'false', when call(phone.AUDIO), call(phone.RECVONLY_VIDEO) or after stopSendingVideo() are used.

  If 'true', the current call tries to send a video. (The other side phone may not accept it.)
- hasEnabledReceiveVideo: Returns 'true' if the call is enabled to receive video.

  This is initially set according to the phone.enableAddVideo flag.

  This is also set according to the videoOption argument of call(), answer(), startSendingVideo().

  If this is 'false', the current call does not receive video, even if the other side starts sending to it.

Return Values

Boolean

2.2.2.13 getEnabledVideoStatus

Parameters

N/A

Return Value

[string] representation of the call enabled video status

(one of ‘sendrecv’, ‘sendonly’, ‘recvonly’ or ‘inactive’)

2.2.2.14 setRemoteHoldState

Used to restore the call state after Web page reloading. Sets internal JsSIP Session state corresponding to the call remote hold state.

Parameters

N/A

Return Values

N/A

2.2.2.15 sendRefer

Used to start the blind call transfer process. The developer should set the call on-hold before using the sendRefer function. The transfer progress is checked by callback “transferorNotification” (it must be set when sendRefer is used).

Parameters

- transferTo [string]: Call transfer destination
- probeCall [A call session object]: Optional. This parameter is used for attended transfer. By default, null.

Return Values

N/A

2.2.2.16 sendReInvite

Sends the re-INVITE SIP message to the SBC server.

Parameters

- showStream [Boolean] By default true.
  Call callShowStreams callback after re INVITE transaction is completed.
  Used to pass values to the srcObject HTML video element.

Return Values

Promise. Resolved if re-INVITE SIP transaction is complete.

Rejected if re-INVITE transaction failed.

2.2.2.17 sendInfo

Sends the INFO message to the SBC server.

Parameters

- body [string]: Text message
- contentType [string] Body content type.
- extraHeaders [array of strings]: Optional. ‘null’ is used by default.

Return Values

N/A

This page is intentionally left blank.

3 API Callbacks / Listeners Interfaces

This API provides the capability to register to listen to different types of events. This section lists the interfaces that must be implemented to receive such events.

3.1 Standard Callbacks

The following are standard callbacks.

3.1.1 Login State Changed Event

Triggered when the login state is changed.

Syntax

```
void loginStateChanged(Boolean isLogin, string cause);
```

Parameter

- IsLogin is 'true' if logged in, and 'false' if not logged in.

  The cause is one of these strings:
  - "connected"
  - "disconnected"
  - "login failed"
  - "login"
  - "logout"

3.1.2 Incoming Call Event

Triggered when receiving an incoming call.

Syntax

```
void incomingCall(AudioCodesSession call);
```

Parameter

- AudioCodesSession: The call session object

3.1.3 Call Confirmed

Triggered when the call is established.

Syntax

```
void callConfirmed(AudioCodesSession call);
```

Parameter

- AudioCodesSession: The call session object

3.1.4 Call Terminated

Triggered when a call is terminated or fails.

Syntax

```
void callTerminated(AudioCodesSession call, message, cause,
redirectTo);
```

Parameters

- AudioCodesSession: The call session object
- Message: Reason of termination (optional)
- case [string]
- redirectTo [string]: (Optional) Destination of redirection, set when the 'case' parameter is 'Redirected'.

3.1.5 Outgoing Call Progress

Triggered when a SIP 'trying' response or a SIP 'ringing' response is received.

Syntax

```
void outgoigCallProgress (AudioCodesSession call);
```

Parameter

- AudioCodesSession: The call session object

3.1.6 Call Show Streams

Triggered when local and remote audio and video streams are ready to be shown in view panels.

Syntax

```
void callShowStreams(AudioCodesSession call, Stream localStream,
Stream remoteStream);
```

Note: Only relevant for the browser API.

Parameters

- AudioCodesSession: The call session object
- Localstream: The stream from the local camera and microphone
- remoteStream: The stream from the remote camera and microphone

3.2 Advanced Callbacks

The advanced callbacks are optional. They provide an extra level of flexibility to the API, which is based on SIP. Developers who are familiar with SIP can utilize the advanced callbacks.

3.2.1 Incoming call event

Triggered when receiving an incoming call.

Syntax

```
void incomingCall(AudioCodesSession call, SipRequest invite,
AudioCodesSession replacedCall, Boolean hasSDP);
```

Parameters

- AudioCodesSession [The call session object]
- SipRequest [The SIP request object]
- AudioCodesSession [The replaced call session object or null]

  The replacedCall argument is not null, if the received INVITE includes a Replace header. In this case, the programmer in the callback should terminate replacedCall, automatically answer the incoming call, and visually (in GUI panel or window) make it the replacement for the terminated call.
- hasSDP [Boolean]

  Enabled for the phone developer for a special case – incoming INVITE without SDP. (If it isn't known whether the other side supports video calls, an answer can be made with or without video).

3.2.2 Call Confirmed

Triggered when the call is established.

Syntax

```
void callConfirmed(AudioCodesSession call, SipMessage message,
String cause);
```

Parameters

- AudioCodesSession: The call session object
- SipMessage: The OK SIP message of an outgoing call, or 'null' for an incoming call

  The cause may be one of the following strings:
  - "received ack"
  - "sent ack"

3.2.3 Call Terminated

Triggered when a call is terminated or if it fails.

Syntax

```
void callTerminated(AudioCodesSession call, SipMessage message,
String cause);
```

Parameters

- AudioCodesSession: The call session object.
- SipMessage [The BYE SIP message; optional, might be 'null']

3.2.4 Outgoing Call Progress

Triggered when a SIP 'trying' response or a SIP 'ringing' response is received.

Syntax

```
void outgoingCallProgress(AudioCodesSession call, SipMessage
response);
```

Parameters

- AudioCodesSession: The call session object
- SipMessage: The Ringing / Trying SIP message

3.2.5 callHoldStateChanged

Triggered when a SIP local or remote hold state changes (incoming or outgoing re-INVITE).

Syntax

```
void callHoldStateChanged(AudioCodesSession call, Boolean isHold,
Boolean isRemote);
```

Parameters

- AudioCodesSession: The call session object
- isHold: Hold (true) or Un-Hold (false)
- IsRemote: Initiator remote side (true) or local side (false)

3.2.6 callIncomingReinvite

Triggered when the phone receives a re-INVITE. The callback is optional. The callback is called twice:

- When the phone receives a re-INVITE (argument start=true)
- After the phone sends an OK to the re-INVITE (argument start=false)

The callback can be used to check the phone, after the re-INVITE starts receiving video to update the video controls GUI.

Syntax

```
void callIncomingReinvite(AudioCodesSession call, Boolean start,
SipMessage request);
```

Parameters

- AudioCodesSession: The call session object
- start: Received re-INVITE (true) or sent OK response to re-INVITE (false)
- Request: Re-INVITE request, set then start = true

3.2.7 transferorNotification

Triggered after the phone starts the blind call transfer process, when the sent REFER was accepted or rejected, and when the NOTIFY message with the transfer result was received.

The callback is optional and should only be used if the phone can initiate a call transfer.

Syntax

```
void transferorNotification(AudioCodesSession call, integer state)
```

Parameters

- AudioCodesSession [The call session object]
- state [integer]
  - **-1:** Transfer failed (REFER was rejected or receive NOTIFY with >= 300)
    After this, the phone should un-hold the current call.
  - **0:** Transfer progress (receive NOTIFY 1xx)
    After this, the phone should un-hold the current call.
  - **1:** Transfer succeeds (receive NOTIFY 2xx).
    After this, the phone should terminate the current call.

3.2.8 transfereeRefer

Triggered when the phone receives a SIP REFER message. In the callback, the developer can check REFER message headers and can accept or reject an incoming REFER message.

The callback is optional and should be used only if the phone supports call transfer as the transferee.

Syntax

```
Boolean transfereeRefer(AudioCodesSession call,SipMessage refer);
```

Parameters

- AudioCodesSession: The call session object
- refer: REFER request

Return Value

- accept incoming REFER: accept (true) or reject (false)

3.2.9 transfereeCreatedCall

When the phone receives a REFER message, it calls the address extracted from the Refer-To header. The developer should use the callback to obtain the reference to the newly created call object. The callback is optional and should be used only if the phone supports call transfer as the transferee.

Syntax

```
void transfereeCreatedCall(AudioCodesSession call);
```

Parameters

- AudioCodesSession [Newly created call session object]

3.2.10 incomingNotify

Receives an incoming 'out of dialog' or 'in dialog' NOTIFY request. The callback is optional and should be used only if the SDK developer wants the phone to receive NOTIFY messages.

Syntax

```
Boolean incomingNotify(AudioCodesSession call, String eventName,
Object from, String contentType, String body, SipMessage request)
```

Parameters

- AudioCodesSession: The call session object: null for out of dialog NOTIFY
- String: event name: Value of Event header
- From object: Object with parameters: user, host, displayName: null or string
- String: content-type value of Content-Type header or null
- String: optional body or null
- SipMessage: NOTIFY request

Return Values

true: Accept incoming NOTIFY, send NOTIFY OK

false: Use default JsSIP NOTIFY processing

3.2.11 incomingMessage

Receives an incoming 'out of dialog' MESSAGE request. The callback is optional and should be used only if the SDK developer wants the phone to receive SIP MESSAGE requests.

Syntax

```
void incomingMessage(AudioCodesSession call, Object from, String
contentType, String body, SipMessage request)
```

Parameters

- call [AudioCodesSession]: Always null because its currently implemented based on outside of the dialog MESSAGE
- From object: Object with parameters: user [String], host [String], displayName: null or String
- content-type [string]: Value of Content-Type header or null
- Optional body or null [string]
- SipMessage [MESSAGE request]

3.2.12 incomingInfo

Receives an incoming 'in dialog' INFO request. The callback is optional and should be used only if the SDK developer wants the phone to receive SIP INFO requests.

Syntax

```
void incomingInfo(AudioCodesSession call, Object from, String
contentType, String body, SipMessage request)
```

Parameters

- call [AudioCodesSession]: The call session object
- From object: Object with parameters: user [String], host [String], displayName: null or String
- content-type [string]: Value of Content-Type header or null
- Optional body or null [string]
- SipMessage [INFO request]

4 Use Examples

This section provides examples that can guide your implementation.

4.1 User Agent: Create Instance, Set Server and Account

```
let phone = new AudioCodesUA(); // phone API
phone.setServerConfig(['wss://webrtclab.audiocodes.com'],
'audiocodes.com',
['74.125.140.127:19302','74.125.143.127:19302']);
phone.setAccount('Igor', 'Igor Kolosov', '?<user_password
string>');
```

4.2 User Agent: Set Listeners (Callbacks)

```
phone.setListeners({
    loginStateChanged: function(isLogin, cause) {Your code},
    outgoingCallProgress: function(call, response) { Your code },
    callTerminated: function(call, message, cause) { Your code },
    callConfirmed: function(call, message, cause) { Your code },
    callShowStreams: function(call, localStream, remoteStream) {
Your code },
    incomingCall: function(call, invite) { Your code }
    callHoldStateChanged(call, isHold, isRemote){ Your code }
});
```

4.3 User Agent Init: Connection to SBC Server and Login

```
phone.init(true);
```

4.4 Make a Call

```
let activeCall = phone.call(phone.VIDEO, 'ariel@audiocodes.com');
```

4.5 Send DTMF During Call

```
activeCall.sendDTMF('9');
```

4.6 Mute / Unmute During Call

```
activeCall.muteAudio(true);
activeCall.muteAudio(false);
```

4.7 Accept Incoming Call

```
activeCall.answer(phone.VIDEO);
```

4.8 Reject Incoming Call

```
activeCall.reject();
```

4.9 Terminate a Call

```
activeCall.terminate();
```

4.10 Use of Remote Streams Video

```
// set remote video html element
document.getElementById('remote_video').srcObject = remoteStream;
```

4.11 Restore Call after Page Refresh

Before closing the page, the 'beforeunload' event is called. In this event, the client checks if there's an active call and stores the relevant data in the local storage for further use.

```
window.addEventListener('beforeunload', onBeforeUnload);

function onBeforeUnload(){
        if (activeCall !== null && activeCall.isEstablished()) {
            let data = {
                callTo: activeCall.data['_user'],
                video: activeCall.getVideoState(),
                replaces: activeCall.getReplacesHeader(),
                time: new Date().getTime(),
                hold: `${activeCall.isLocalHold() ? 'local' :
''}${activeCall.isRemoteHold() ? 'remote' : ''}`,
                mute: `${activeCall.isAudioMuted() ? 'audio' :
''}${activeCall.isVideoMuted() ? 'video' : ''}`
            }
            localStorage.setItem('phoneRestoreCall',
JSON.stringify(data));
         }
}
```

After reloading the page and registering on the SBC server, the client checks if there was an active call and restores it.

```
let data = localStorage.getItem('phoneRestoreCall');
if( data !== null ){
  localStorage.removeItem('phoneRestoreCall');
  let r = JSON.parse(data);
  let delay = Math.ceil(Math.abs(r.time - new
Date().getTime())/1000);
  if( delay > 20 ){ // Call can be restored only 20 sec.
     console.log('Cannot restore call, delay is too big');
  } else {
     console.log('Try restore call...');
     activeCall = phone.call(r.video === 'sendrecv' || r.video ===
'sendonly' ? phone.VIDEO : phone.AUDIO, r.callTo, ['Replaces: ' +
r.replaces]);
  }
}
```

4.12 Set Custom Logger

The following shows an example of forwarding the logs to a specific destination using a custom logger function.

```
phone.setAcLogger(ac_log);         // Set AudioCodes API logger
phone.setJsSipLogger(jssip_log);   // Set JsSIP API logger

// Add time stamp and color function
function ac_log() {
  let args = [].slice.call(arguments);
console.log.apply(console, [createTimestamp() + '%c' +
    args[0]].concat(['color: BlueViolet;'], args.slice(1)));
}
// Add time stamp.  function
function jssip_log() {
  let args = [].slice.call(arguments);
console.log.apply(console, [createTimestamp() +
    args[0]].concat(args.slice(1)));
}
```

4.13 Getting Statistics

The following is an example for statistics retrieval using RTCPeerConnection and an example output to the console for outband-rtp and inbound-rtp.

Note: WebRTC API is used in the example.

SDK API can also be used.
See the getWR().connection.getStats() method.

```
function printCallStats() {
    if (activeCall === null) {
        ac_log('activeCall is null');
        return;
    }
    let conn = activeCall.getRTCPeerConnection();
    let str = '';
    conn.getStats(null).then(report=>report.forEach(now=>{
        switch (now.type) {
        case 'outbound-rtp':
        case 'inbound-rtp':
            //case 'track':
            //case 'stream':

            str += ' {';
            let first = true;
            for (let key of Object.keys(now)) {
                if (first)
                    first = false;
                else
```

```
                    str += ',';
                str += (key + '=' + now[key]);
            }
            str += '} ';
            break;
        default:
            break;
        }
    }
    )).then(()=>{
        ac_log('call stats: ' + str);
    }
    ).catch((err)=>{
        ac_log('stat error', err);
    }
    );
}
```

4.14 Incoming Call with Replaces Header

If the incoming INVITE contains a Replaces header (that points to an existing open call) in incomingCall callback argument, then replacedCall will be not null.

In the case where the programmer terminates the replaced call and automatically answers the incoming call, the incoming call visually (in GUI panel or window) replaces the terminated call.

```
        incomingCall: function (call, invite, replacedCall) {

            // If received INVITE with Replaces header
            if (replacedCall !== null) {
                ac_log('phone: incomingCall, INVITE with
Replaces');

                // close the replaced call.
                replacedCall.data['terminated_replaced'] = true;
                replacedCall.terminate();

                // auto answer to replaces call.
                activeCall = call;
                activeCall.data['open_replaced'] = true;

                let videoOption = replacedCall.hasVideo() ?
phone.VIDEO : (replacedCall.hasReceiveVideo() ?
phone.RECVONLY_VIDEO: phone.AUDIO);
                activeCall.answer(videoOption);
                return;
            }
```

4.15 Incoming Call with Custom Headers

The incoming INVITE may contain custom SIP headers. For example, using the header Alert-Info. To parse such a header, the programmer may write their own SIP header parser. The example below uses the custom AlertInfo parser that is defined in utils.js.

```
        incomingCall: function (call, invite, replacedCall) {

            . . . .
            // Can be used custom header in incoming INVITE
            // ------ begin of Alert-Info auto answer example ----
            // JsSIP parse Alert-Info as raw string. We use custom
parser defined in utils.js
            let alertInfo = new AlertInfo(invite);
            ac_log(`alert-info header ${alertInfo.exists() ? '
exists' : 'does not exist'}`);
            if (alertInfo.hasAutoAnswer()) {
                ac_log(`alert-info
delay=${alertInfo.getDelay()}`); // currently ignored
                ac_log('*** Used Alert-Info Auto answer ***');

                let videoOption = call.hasVideo() ? (hasCamera ?
phone.VIDEO : phone.RECVONLY_VIDEO) : phone.AUDIO;
                guiAnswerCall(videoOption);
                return;
            }
            //------ end of Alert-Info auto answer example ----
```

This page is intentionally left blank.

5 Tutorial

You can find a useful tutorial with AudioCodes-provided Web RTC examples at https://webrtcdemo.audiocodes.com/sdk/.

International Headquarters
1 Hayarden Street,
Airport City
Lod 7019900, Israel
Tel: +972-3-976-4000
Fax: +972-3-976-4040

AudioCodes Inc.
200 Cottontail Lane
Suite A101E
Somerset NJ 08873
Tel: +1-732-469-0880
Fax: +1-732-469-2298

Contact us: https://www.audiocodes.com/corporate/offices-worldwide
Website: https://www.audiocodes.com/

Document #: LTRT-14050